

データ分析基盤の Redshiftへの移行と活用方法

2019年9月24日(火)

Amazon Analytics 事例祭り
データウェアハウスマイグレーション

株式会社レコチョク

事業システム部 エンタープライズディストリビューショングループ

佐藤 俊之

人と音楽の新しい関係をデザインする。



2019/10/2



佐藤 俊之 (Toshiyuki Sato)

データドリブンチーム リーダー

- データベースエンジニア (Oracle, PostgreSQL, MySQLなど)
- データ分析基盤管理者
- BIスペシャリスト
- CRM、レコメンデーションなどデータ関連システム担当

好きなAWSサービス

- Redshift
- S3

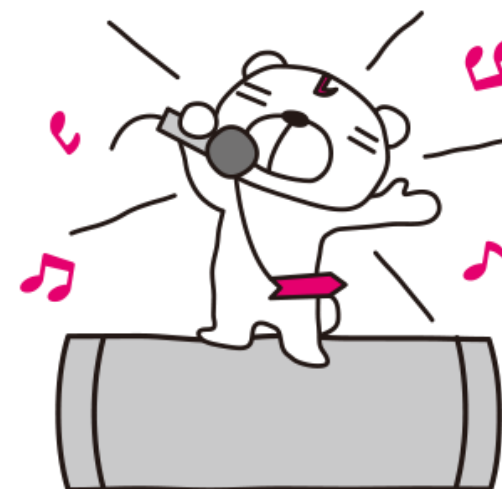


1. データ分析基盤の移行

- Redshift選定理由
- 移行方法
- 移行時の問題
- 移行した結果

2. Redshiftの活用

- Redshiftの役割
- WLM(ワークロード管理)の設定
- データ取込の工夫
- Spectrumの利用
- 活用方法と注意点



レコチョクはどんな会社か？

① デジタル音源配信事業

自社(レコチョク)サービス

従量課金制



定額課金制



協業サービス

従量課金制

dミュージック

Music Store
powered by レコチョク

UARI TV
ミュージック

定額課金制

dヒッツ

dミュージック
月額コース

UARI TV
ミュージック

音 × 楽
OTORAKU

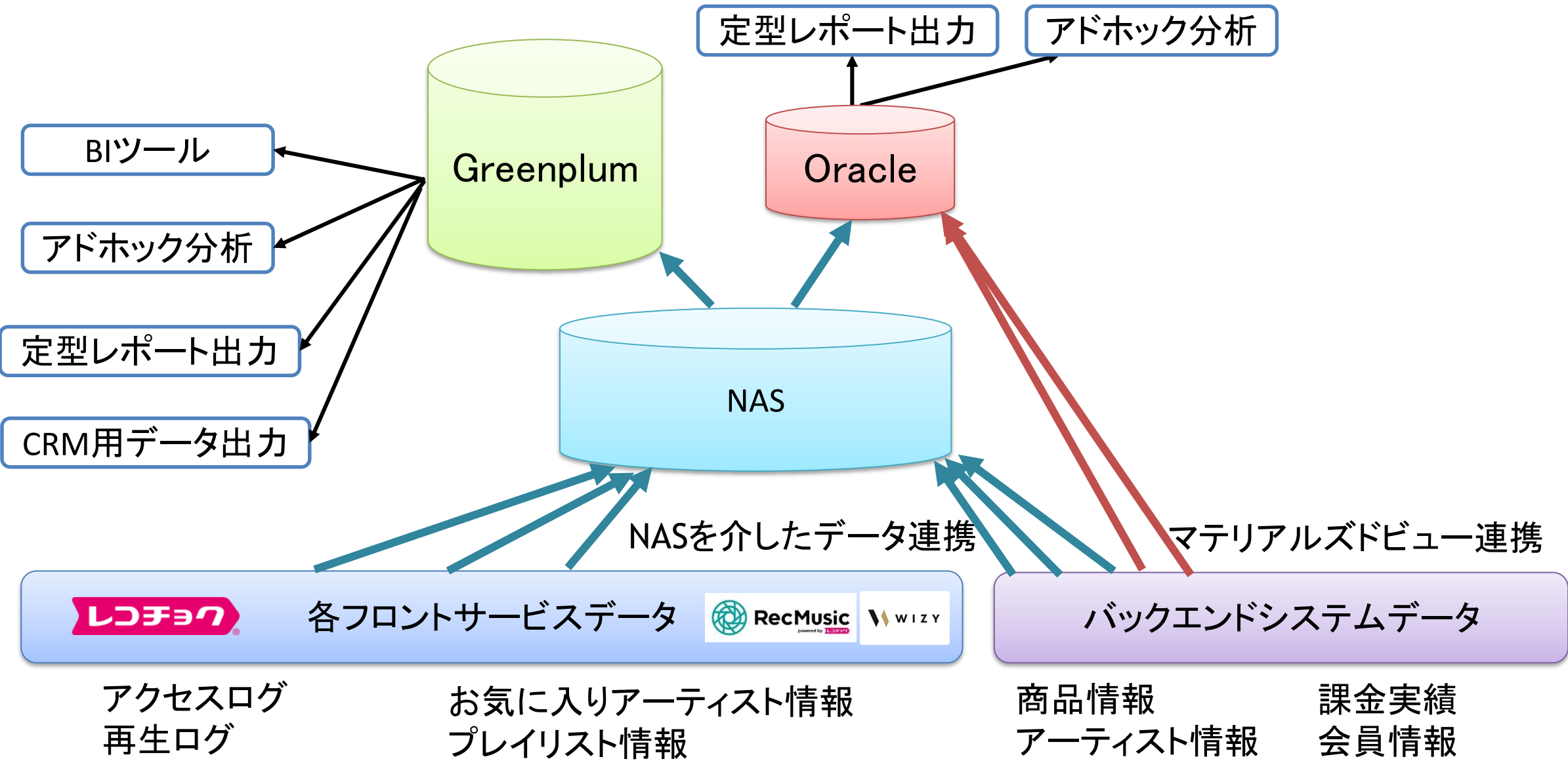
② 音楽ソリューションサービス



様々な音楽サービスを提供している
音楽についてひたすら考えている事業会社



移行前の構成とワークロード

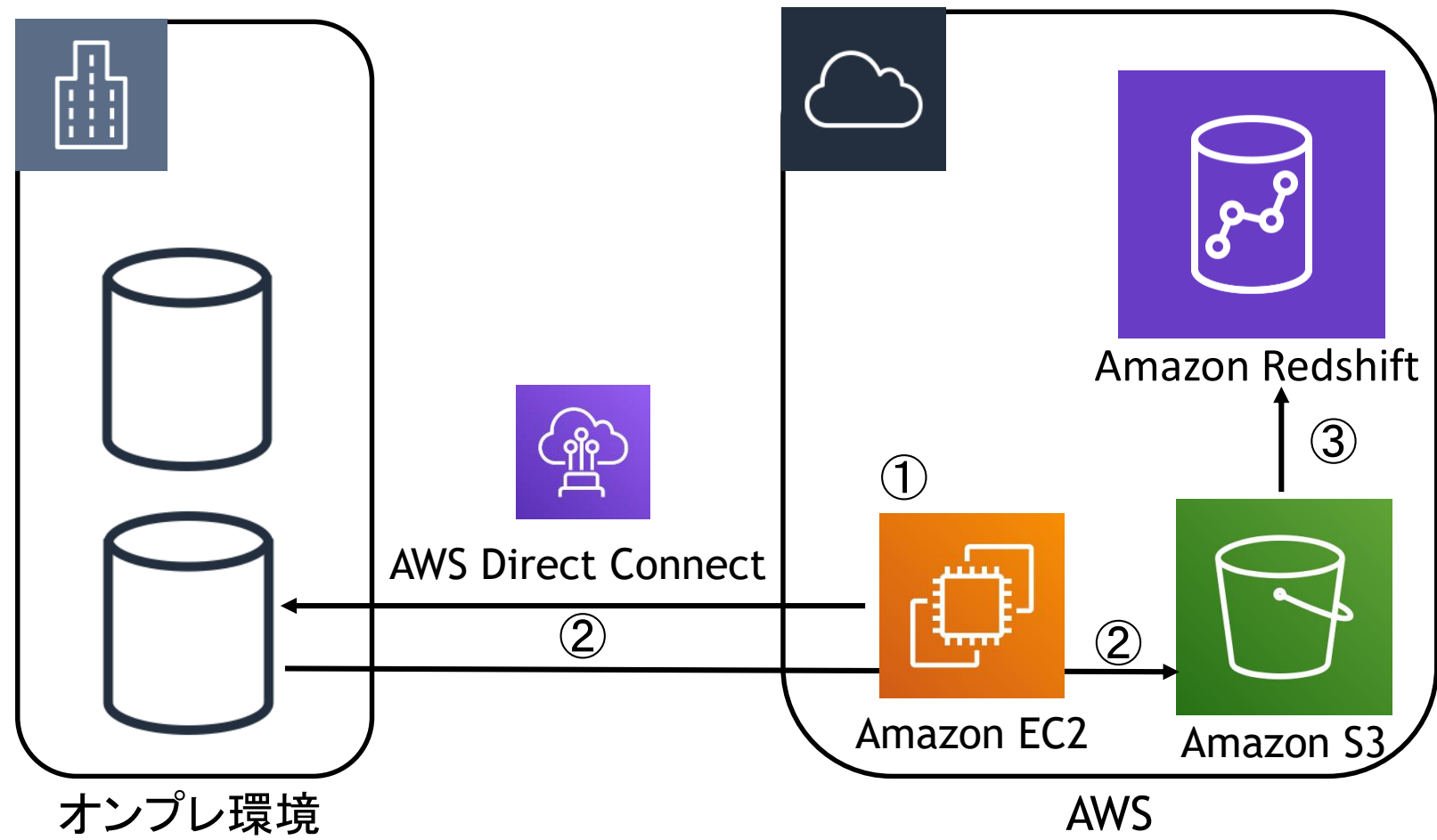


1. レコチョク全システムをオンプレ環境からAWSへ全面移行
2. DWH環境はGreenplum(PostgreSQL互換)を利用していた
3. ミッションクリティカルな別の集計システムはOracleを利用していた
4. PoC、取込、集計処理、データ圧縮

SQL移行の親和性と性能からRedshiftを選定



データ移行方法



■ 移行方法

1. データ移行用のEC2を構築
2. EC2からDirectConnect経由でオンプレデータベースのTSVデータを分割してS3へ出力
→ノード数の倍数に分割して、zip圧縮
→文字列データは改行やタブ、クォーテーションなどをどう取り扱うか検討
3. S3からデータロード

・ オンプレデータベース情報
Greenplum : 48ノード 16TB
Oracle 11G Enterprise Edition :
2CPU HAクラスター構成 10TB

・ Redshift情報
ds2.xlarge **8ノード 16TB**
(1ノード: メモリ31GB HDD2TB)



- **Greenplumのデータ加工バッチのSQLはルールにしたがって一括変換**
 - 文字列関数や日付関数など
 - PL/PGSQLのストアドプロシージャは、UDF (User-Defined Functions:ユーザー独自の定義関数) にて新規作成
 - 1000本以上のSQLの変換および**動作確認**を実施

- **Redshiftへのデータ取込処理は新規作成**

- AWSの特性に合わせて1から設計
 - ファイル配置、ファイル配置確認、ファイル分割、データ取込、重複チェック、バックアップ
- データ取込処理は基本的にS3のファイルをCOPYコマンドで取込む
- RDSのデータを連携する場合もS3にファイル出力してからRedshiftへ取込む
- S3の設計も大事

- **DDLは新規作成**

- ソートキーと分散キーを設定
- カラム圧縮タイプを設定(現在は基本的にはzstdで問題ない)
- varchar型は文字数ではなくバイト数になるため注意



移行時の問題点(当時)

- **移行支援ツールが無かった(現在はある)**
 - AWS Schema Conversion Tool (SCT) : 異なるRDBのDDLを変換してくれるツール
 - AWS Database Migration Service (DMS) : データ移行サービス
- **ストアドプロシージャ(PL/PGSQL)が無かった(現在はある)**
 - オンプレでは使っていた
 - UDF (User-Defined Functions:ユーザー独自の定義関数) で作成
 - 2019年春に新機能追加された
- **プライマリキー制約が既存と異なる**
 - 重複はエラーとして検知されないので注意
 - NotNull制約は効く



移行によって変わったこと

- **パフォーマンスが改善**
- **データ容量削減**



移行によって変わったこと

- **オンプレデータベースの保守/運用が無くなった**
 - サーバ、ストレージのハードウェア障害
 - ハードウェア交換
 - ネットワーク障害
 - OSバージョンアップ、ファームウェアバージョンアップ
 - バージョンアップやハードウェア交換や障害時の各種テスト
 - 容量拡張、ストレージ管理
 - マテリアルズドビューの管理(Oracle)
 - 表領域の管理(Oracle)
 - インデックス再作成、統計情報の固定化(Oracle)
 - バックアップ/リストア

インフラ、DBA業務からの解放



- Redshiftへのデータ移行はシンプルで取込は早い
- DDL含めSQLの変換はサポートツールが用意されている
- **データベースの運用 / 保守の多くが不要となる**
- Redshiftでの処理の方が4倍以上早い場合がある
- データ容量は26TB→11TBと約4割ほどに圧縮できた



1. データ分析基盤の移行

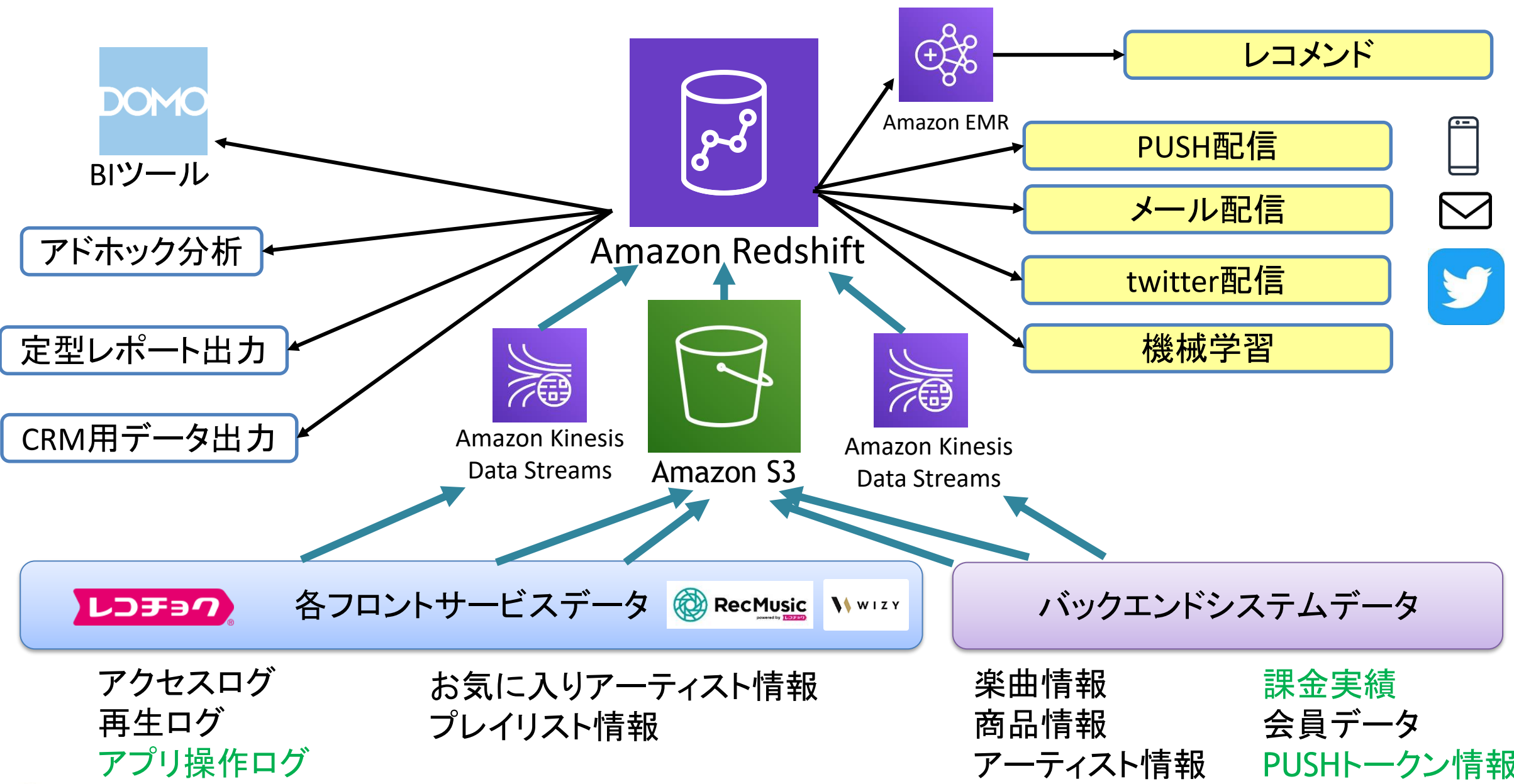
- Redshift選定理由
- 移行方法
- 移行時の問題
- 移行した結果

2. Redshiftの活用

- Redshiftの役割
- WLM(ワークロード管理)の設定
- データ取込の工夫
- Spectrumの利用
- 活用方法と注意点



Redshiftの活用



データ分析基盤の役割を整理

1. データ収集

- レコチョク全サービスの分析用データを一元管理
- S3ファイル連携、RDS直接連携など連携方法を共通化
- Kinesisからユーザの行動データ(アプリ操作ログ、課金実績など)をほぼリアルタイムで収集

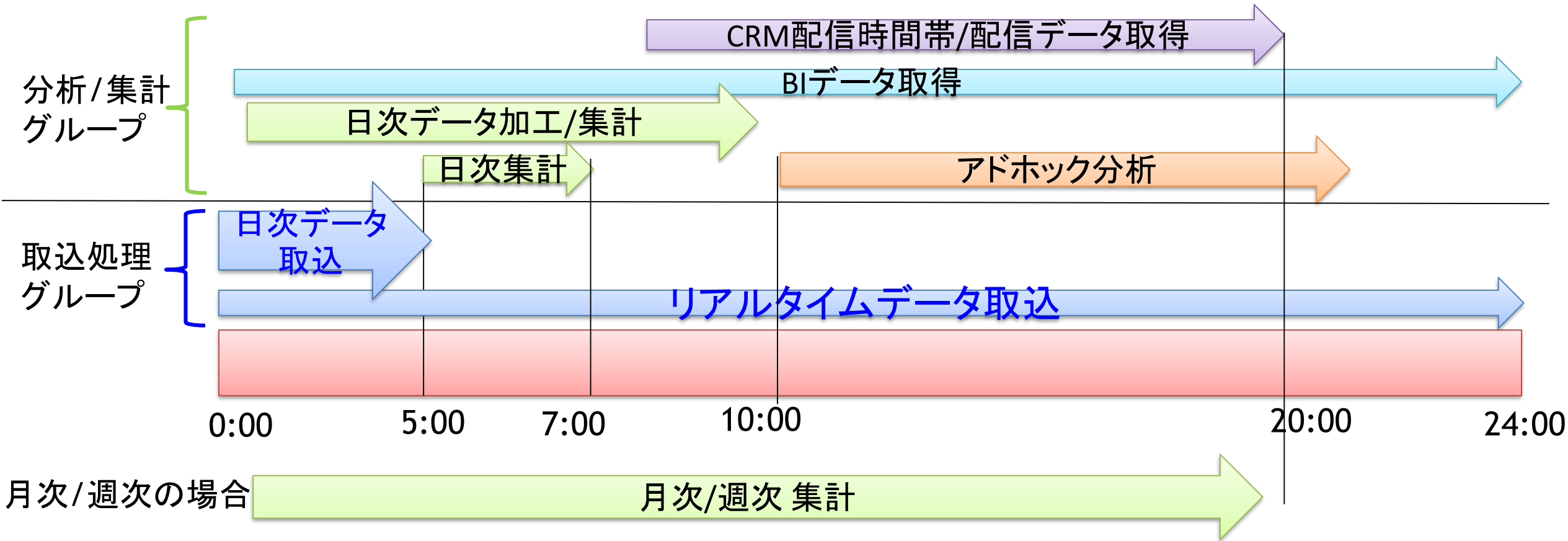
2. データ加工

- CRM PUSH/メール配信、効果測定、レコメンデーション、BI用などの加工データを作成
- 機械学習でユーザ毎に活動時間を分析し、CRM配信に利用

3. データ活用

- 販売促進：パーソナライズ、レコメンド、CRM配信
- 分析：効果測定、KPI分析、分析データの提供、リアルタイム分析
- 報告：日次/週次/月次レポート、アドホック分析
- 予測：売上予測、退会抑止





特に毎月月初5営業日までは月次集計処理が多く高負荷状態になりやすい

WLM(ワークロード管理)の設定

分析/集計用

24時間

取込用

0:00-10:00

- 取込処理でメモリを多く利用する処理はslot数を変更
wlm_query_slot_count
- 10:00以降の小さい集計処理は取込処理グループのリソースを利用することもある

メモリ

同時実行数

75% x 14

メモリ

同時実行数

25% x 25

0:00-10:00の間に取込処理が遅延しないように、かつ、分析/集計処理に多くリソースが使われるように設定している

データ取込の工夫 一時テーブルとCOPYコマンドオプションMAXERROR

- MAXERRORを利用すると、不正フォーマットデータが連携される場合に不正データを取込せずに**正常なデータのみ取込できる**
- COPYコマンドは一時テーブルに対して実行する



Amazon S3

```
COPY `一時テーブル` FROM 's3://...'
DATEFORMAT 'auto' TIMEFORMAT 'auto' EMPTY AS NULL
TRUNCATE COLUMNS MAXERROR AS 100000;
```

COPYコマンド



一時テーブル

MAXERROR : ※Redshift開発者ガイドより
 ロードのエラー数が *error_count* 以上である場合、ロードは失敗します。
 ロードのエラーがそれより少ない場合、処理は続行されます。
 データの形式エラーやその他の不整合のために一部の行をテーブルにロードできないときにロードを継続するには、このパラメータを使用します。



取込先テーブル

```
INSERT INTO `取込先テーブル` AS
SELECT column1 , LEFT( column2 , 20) ,column4
FROM `一時テーブル`
WHERE column_flg = 1 ;
```


INSERTコマンド

例
 column2を切り捨てる
 column3を取込対象外
 column_flg=1に絞る



MAXERROR でエラーとなったデータは stl_load_errors で確認

```
select * from stl_load_errors limit 1;
-[ RECORD 1 ]-----+-----
starttime      | 2019-09-24 15:24:11.190924
filename       | s3://bucket-name/table.tsv
line_number    | 104
colname        | column1
type           | varchar
col_length     | 20
position       | 15
raw_line       | 10001      testtesttesttest  1002810591
raw_field_value | testtesttesttest
err_code       | 1204
err_reason     | String length exceeds DDL length
```



エラー通知 APP 10:00 AM

20190918 データ取込エラー 820件

TARGET:2019-09-18
 BATCHID:B-S14
 TABLE_NAME:x
 FILE_NAME:s3://
 LINENO:18
 REASON:String length exceeds DDL length
 EXECUTE_DAY:2019-09-18 01:00:50.942951

TARGET:2019-09-18
 BATCHID:B-S14
 TABLE_NAME:x
 FILE_NAME:s3://
 LINENO:231
 REASON:String length exceeds DDL length
 EXECUTE_DAY:2019-09-18 01:00:50.942951

エラーをまとめてslackで通知



Amazon Redshift Spectrumとは

- Amazon S3 に置かれたデータに対して、RedshiftのSQLクエリを実行可能
 - Redshiftのストレージ領域は使わないため、容量を節約できる
- Amazon Athena と似ているが、大きなデータに対して、複数のクラスタで**並列処理が可能**なため、クエリを高速に実行できる
- S3にあるデータとRedshift内のデータを結合できる



- **課題**

- **Redshiftのデータ量の上限**

- 日々データが蓄積されてRedshiftのデータ量が増加してきている
 - 古いデータをS3へ退避しても、退避したデータが必要になった場合は、再度Redshiftへデータを戻す作業が発生し運用コストがかかる

- **Redshiftのリソース不足と高負荷**

- 複数のワークロードでCPUやメモリなどのリソースを共有しているが、リソースが枯渇し高負荷となることがある
 - 特に月初期間中の営業日は、Redshiftの負荷状況を監視し、処理タイミングを調整し手動で処理を実行する場合もあり、運用コストがかかっている



Spectrumを利用する前に検証したこと

1. Spectrumを利用することによって運用作業を増やしたくない
 - 同じテーブルのデータを自動で定期的にS3に退避させたい
 - 退避したデータがS3に追加されていても、その度にSQLを変更するといった作業を実施せずに利用したい
2. Redshift上のデータと結合して使いたい
3. Spectrum(過去データ)を意識しないで使いたい
 - 最新データはXXXテーブル、過去データはXXX_OLDテーブルというようにテーブルを2つに分けるのではなく、1つのテーブルとして扱いたい
 - VIEWを利用したい
4. マルチクラスターでクエリを実行したい



検証1.データのS3退避の自動化

検証内容: S3にファイルを追加すると追加分もSpectrumからSELECT可能か

結論 : 追加分もSELECT可能

```
--Redshiftのあるテーブルのデータを
s3://test-recochoku-jp/配下に圧縮したTSVファイルとして配置
UNLOAD ('SELECT * FROM recochoku_artist
WHERE data_load_target_date = "2019-09-12" ')
TO 's3://test-recochoku-
jp/recochoku_artist/recochoku_artist_201909'
DELIMITER '¥t' ESCAPE ALLOWOVERWRITE GZIP
CREDENTIALS 'xxx' ;

--Spectrumが参照する外部テーブル作成
CREATE EXTERNAL TABLE spectrum.sp_recochoku_artist(
creator_user_id bigint,
recochoku_artist_id bigint,
data_load_target_date date,
data_load_datetime timestamp )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '¥t' STORED AS
TEXTFILE
LOCATION 's3://test-recochoku-jp/recochoku_artist/';
```

```
--Spectrumのデータ件数を確認
SELECT data_load_target_date,COUNT(1) FROM
spectrum.sp_recochoku_artist GROUP BY 1;
2019-09-12 220 ← 1件データが確認できた

-- S3にデータファイルを追加で退避
UNLOAD ('SELECT * FROM recochoku_artist WHERE
data_load_target_date = "2019-09-18" ')
TO 's3://test-recochoku-
jp/recochoku_artist/recochoku_artist_201909'
DELIMITER '¥t' ESCAPE ALLOWOVERWRITE GZIP
CREDENTIALS 'xxx' ;

--同じSQLでデータ件数を確認
SELECT data_load_target_date,COUNT(1) FROM
spectrum.sp_recochoku_artist GROUP BY 1;
2019-09-12 220
2019-09-18 223 ← 追加データが確認できた
```



検証2.Redshift上のテーブルと外部テーブルを結合

検証内容:Redshift上のテーブルと外部テーブルを結合してSELECT可能か
結論 :SELECT可能

```
SELECT sp.recochoku_artist_id ,tst.artist_name  
FROM spectrum.sp_recochoku_artist sp  
INNER JOIN tst_artist tst -- Redshift上のテーブル  
ON sp.recochoku_artist_id = tst.recochoku_artist_id ;
```

```
-- 結果  
100001 mike  
100002 bob
```



検証内容: Spectrumを意識しないで使いたい。

結論 Redshift上のテーブルと外部テーブルをUNIONしたVIEWをSELECT可能か
:VIEWを作成してSELECT可能

```
-- VIEW作成
CREATE VIEW view_recochoku_artist
AS
SELECT * FROM recochoku_artist
UNION ALL
SELECT * FROM spectrum.sp_recochoku_artist
WITH NO SCHEMA BINDING ;

SELECT COUNT(1) FROM view_recochoku_artist;
663 – 結果
```



検証4.マルチクラスターでのSpectrum利用

検証内容:別のRedshiftクラスターから同じS3のデータをSELECT可能か
結論 :SELECT可能

```
--Spectrumが参照する外部テーブル作成
CREATE EXTERNAL TABLE newspectrum.new_sp_recochoku_artist(
creator_user_id bigint,
recochoku_artist_id bigint,
data_load_target_date date,
data_load_datetime timestamp )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '¥t' STORED AS TEXTFILE LOCATION
's3://test-recochoku-jp/recochoku_artist/';

--同じSQLでデータ件数を確認
SELECT data_load_target_date,COUNT(1)
FROM newspectrum.new_sp_recochoku_artist GROUP BY 1;
2019-09-12 220
2019-09-18 223
```



- Redshift内のデータ容量を4.5TB削減
- 定常的に利用するデータはRedshift内に保持しておき、それ以外のデータ、特に数年前データなどはS3に圧縮したファイルで保持する運用とする
- データ量が多いテーブルは自動的にS3へ退避する運用とする
- VIEWを作成しておくことで、1つのテーブルとしてSpectrumを意識せずに利用可能

その他の活用方法と注意点

- **PrimaryKey制約**
 - PKの重複チェック機能をつける
- **vacuum実行(毎週日曜日 10-23時まで)**
 - stv_tbl_perm
- **メンテナンスウィンドウ**
- **一時ディスク領域に注意**
 - stl_queryまたはマネジメントコンソールを確認
- **varcharカラムの桁数を余計に大きくしない**
- **viewの運用 No schema bindingオプションを利用**
- **分散スタイル**
 - 最近新機能追加でAutoが出てきた、基本はEVENを利用
- **カラム圧縮 analyze compressionを利用**



- Redshiftへの移行は難しい
- 処理速度は向上しデータ容量は移行前よりも少なくなった
- オンプレ運用から解放された
- WLMのチューニング、設計は重要
- Spectrumの利用をおすすめ



We're hiring !



<https://recruit.recochoku.jp/>

エンジニアブログ公開中



<https://techblog.recochoku.jp/>

ご清聴ありがとうございました

