名状しがたき DB アンチパターン

KLab 株式会社 開発部
Engineering manager
牧内大輔

自己紹介

NAME 牧内大輔 (@makki_d)
JOB 開発部 Engineering manager

能力値および能力値ロール

STR/筋力	7	APP/外見	11	EDU/教育	17
CON/体力	10	SAN/正気度	32	IDEA/ｱｲﾃﾞｨｱ	75
SIZ/体格	12	INT/知性	15	LUC/幸運	70
DEX/敏捷	8	POW/精神力	14	KNW/知識	85

探索者の技能

言いくるめ	65%	機械修理	50%	天文学	30%
医学	35%	生物学	90%	目星	60%
運転	40%	説得	40%		

シナリオ

- 自己紹介
- ソーシャルゲームと DB
- インデックスとは
- インデックスとソート
- インデックスとロック
- まとめ

ソーシャルゲームとDB

- リクエスト数が桁違い
  - 秒間数千リクエストは普通
- 多くのページでユーザの状態が変化する
  - 更新系クエリが多い
  - 負荷がマスタDBに集中

スキーマ・クエリのチューニングが必須
インデックスは超重要！

インデックスとは

インデックスの基本

ID	名前	身長	体重
1	岡部倫太郎	177	59
2	牧瀬紅莉栖	160	45
3	椎名まゆり	152	45
4	阿万音鈴羽	163	51
5	フェイリス	143	43
6	漆原るか	161	44
7	桐生萌郁	167	54
8	橋田至	164	98

身長	ID
143	5
152	3
160	2
161	6
163	4
164	8
167	7
177	1

Index (身長)

- 検索の効率化

WHERE 身長 =152

WHERE 身長 >165

複合インデックス

体重	身長	ID
43	143	5
44	161	6
45	152	3
45	160	2
51	163	4
54	167	7
59	177	1
98	164	8

Index (体重 , 身長)

- 複数のカラムでソート
  - 同体重の中で身長順ソート
- 複合条件での検索に有効

  WHERE 体重 =45 AND 身長 =160

- 複数の範囲条件には使えない

  WHERE 体重 >50 AND 身長 >165
  - ２つめの条件がソートされてない

インデックスとソート

ソートは重い

1. 検索条件にマッチするデータを抽出
2. 並べ替える
3. クライアントへ返す

- 抽出データがメモリに乗り切らない場合
  - 一旦ディスクに書き出してからソート (filesort)
  - 余分な Disc I/O が発生

インデックスを使ったソート

体重	身長	ID
43	143	5
44	161	6
45	152	3
45	160	2
51	163	4
54	167	7
59	177	1
98	164	8

Index (体重 , 身長)

- インデックスを使って検索した場合
  - 抽出データはインデックスの順
  - 既に並んでいるなら並び替え不要
  - Filesort が発生しない

WHERE 体重 =45 ORDER BY 身長

- 使えない場合もある

WHERE 体重 >50 ORDER BY 身長

インデックスを冒涜するような
オプティマイザの想像し難い振る舞いに、
わたしは慄然とするほかなかった。

高負荷と slave 遅延

- とあるイベント中に DB 負荷増大
- slave 遅延が止まらない
  - 無視できないほどの不整合が見えはじめる

- slow_log の調査
  - 時間のかかったクエリ
  - インデックスを使っていないクエリ

問題のあったクエリ

```
SELECT * FROM tea_time_history
WHERE guest_perosn_id='95230' ORDER BY created_at DESC LIMIT 10;
```

```
CREATE TABLE `tea_time_history` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `person_id` int(11) DEFAULT NULL,
   `guest_person_id` int(11) DEFAULT NULL,
   `tea_time_count` int(11) DEFAULT '0',
   `text_id` varchar(22) DEFAULT NULL,
   `friend_status` tinyint(4) DEFAULT NULL,
   `created_at` datetime DEFAULT NULL,
   PRIMARY KEY (`id`),
   KEY `person_id_idx` (`person_id`,`created_at`),
   KEY `guest_person_idx` (`guest_person_id`,`created_at`),
   KEY `guest_person_text_idx` (`guest_person_id`,`text_id`,`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

EXPLAIN

- どんな index が使われるか、どんな検索をするか

```
mysql> EXPLAIN SELECT  * FROM tea_time_history WHERE
guest_perosn_id='95230' ORDER BY created_at DESC LIMIT 10;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tea_time_history
         type: ref
possible_keys: guest_person_idx,guest_person_text_idx
          key: guest_person_text_idx
      key_len: 5
          ref: const
         rows: 16224
        Extra: Using where; Using filesort
```

違うインデックスが使われている

WHERE 句を工夫

```
mysql> EXPLAIN SELECT  * FROM tea_time_history WHERE
guest_perosn_id='95230' AND created_at>'2011-09-01 00:00:00'
ORDER BY created_at DESC LIMIT 10;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tea_time_history
         type: range
possible_keys: guest_person_idx,guest_person_text_idx
          key: guest_person_idx
      key_len: 14
          ref: NULL
         rows: 971
        Extra: Using where
```

正しいインデックスが使われ、Using filesort が消えた

高負荷と slave 遅延の原因

1. 検索に意図しない index が使われていた
2. 巨大な一時テーブルの filesort が発生
3. Disc I/O が多発
4. 書き込み待ちで slave 遅延

正しいインデックスが使われていれば防げた

気をつけるべき点

- インデックスは使われなければ意味が無い
  - 張るだけではダメ
  - 使われるようにクエリを工夫
- 似ているインデックスは極力作らない
  - 間違って使われるかもしれない
- 必要なら FORCE INDEX 構文で強制
  - 間違いなく使ってほしい時の手段

インデックスとロック

ロック（排他制御）とは

- 同時にアクセスされても処理を２回走らせない
  - アイテム増殖や様々なチートの元
  - 厳密な制御が必要
- ロックしすぎも問題
  - スループットの低下
  - 待ち行列の発生
  - タイムアウト多発

いかに少ないロックで排他制御するか

インデックスでロック範囲制御

- インデックスで絞り込めたレコードのみロック

```
CREATE TABLE `person_dxp` (
  `person_id` int(11) NOT NULL,
  `dxp` int(11) NOT NULL,
  `last_added_at` datetime NOT NULL,
  `expired_at` datetime NOT NULL,
  `updated_at` datetime DEFAULT NULL,
  `created_at` datetime DEFAULT NULL,
  PRIMARY KEY (`person_id`),
  KEY `idx_expired_at` (`expired_at`)
);
```

期限切れバッチ処理:

```
update person_dxp set dxp=0 where expired_at<=? and dxp>0;
```

ぞっとするようなオプティマイザの奇妙な行動は、
わたしの目論見を嘲笑うように、
ロック範囲を拡げつづけるのだった。

インデックスが使われない

```
mysql> explain select * from person_dxp where expired_at<='2012-12-01 00:00:00' and dxp>0;
+----+-------------+------------+------+----------------+------+---------+------+------+-------------+
| id | select_type | table      | type | possible_keys  | key  | key_len | ref  | rows | Extra       |
+----+-------------+------------+------+----------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | person_dxp | ALL  | idx_expired_at | NULL | NULL    | NULL |   47 | Using where |
+----+-------------+------------+------+----------------+------+---------+------+------+-------------+
1 row in set (0.05 sec)
```

- インデックスが使われない場合もある
  - 統計情報によって自動判別
  - 全件検索は全行ロック

```
mysql> explain select * from person_dxp force index (idx_expired_at) where expired_at<='2012-12-01 00:00:00'
 and dxp>0;
+----+-------------+------------+-------+----------------+----------------+---------+------+------+-------------+
| id | select_type | table      | type  | possible_keys  | key            | key_len | ref  | rows | Extra       |
+----+-------------+------------+-------+----------------+----------------+---------+------+------+-------------+
|  1 | SIMPLE      | person_dxp | range | idx_expired_at | idx_expired_at | 8       | NULL |   23 | Using where |
+----+-------------+------------+-------+----------------+----------------+---------+------+------+-------------+
1 row in set (0.15 sec)
```

気をつけるべき点

- インデックスは使われなければ意味が無い
  - 確実に使われるように FORCE INDEX
- 範囲選択によるロックは避けたほうがよい
  - Next key lock
  - Gap lock

まとめ

まとめ

- インデックスを使いこなそう
  - 検索だけでなく、ソートやロックにも
- 使われなければ意味が無い
  - 間違ったインデックスが使われないように工夫

インデックスを制する者が DB を制す

Ph'nglui mglw'nafh Cthulhu
R'lyeh wgah'nagl fhtagn.