MySQL 5.7入門(チューニング基礎編)

Yoshiaki Yamasaki / 山﨑 由章
MySQL Senior Sales Consultant, Asia Pacific and Japan

SAFE HARBOR STATEMENT

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。
また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。
以下の事項は、マテリアルやコード、機能を提供することをコミットメントするものではない為、
購買決定を行う際の判断材料になさらないで下さい。

オラクル製品に関して記載されている機能の開発、リリースおよび時期については、
弊社の裁量により決定されます。

はじめに

本資料で紹介している設定値、コマンド、実行結果等は、注釈が無い限り
MySQL 5.7 を対象としています。
MySQL 5.6 以前では、同様の設定やコマンドで動作しない場合や、
実行結果が異なる場合もありますので、ご注意下さい。

Program Agenda

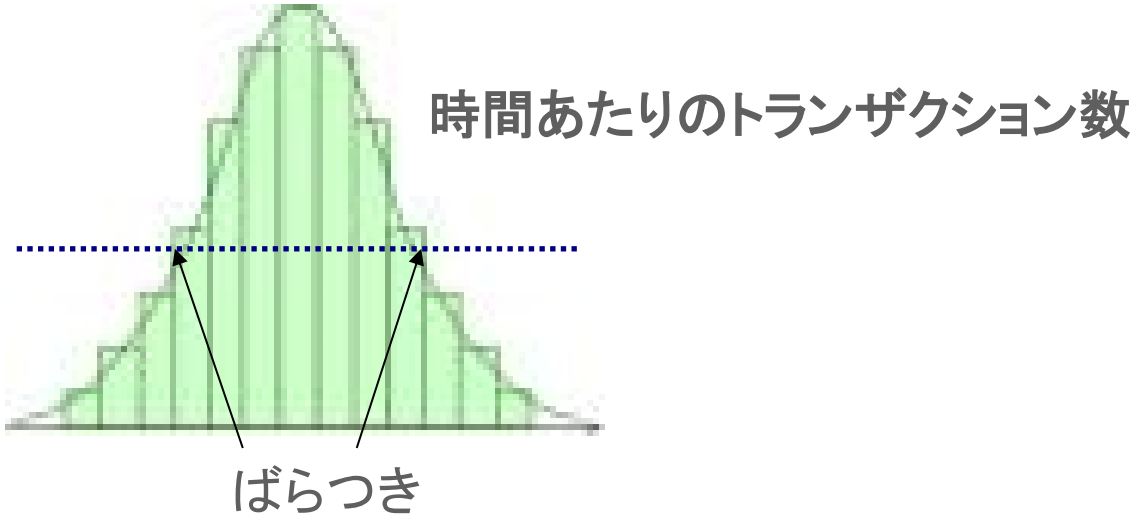
1. チューニング概論
2. MySQL ServerチューニングTIPS
3. SQLチューニングTIPS
4. 参考情報

Program Agenda

1. チューニング概論
2. MySQL Serverチューニング TIPS
3. SQLチューニング TIPS
4. 参考情報

パフォーマンスチューニングとは？

- 単純な言葉ではあるが多くの意味が
- チューニングの目的
  - ユーザを満足させるため
    ⇒限られたシステム・リソースの中で、最大限のパフォーマンス効果を出すこと
- パフォーマンスの指標の例
  - スループット
  - レスポンスタイム / レイテンシ
  - スケーラビリティ
  - 上記の組合せ



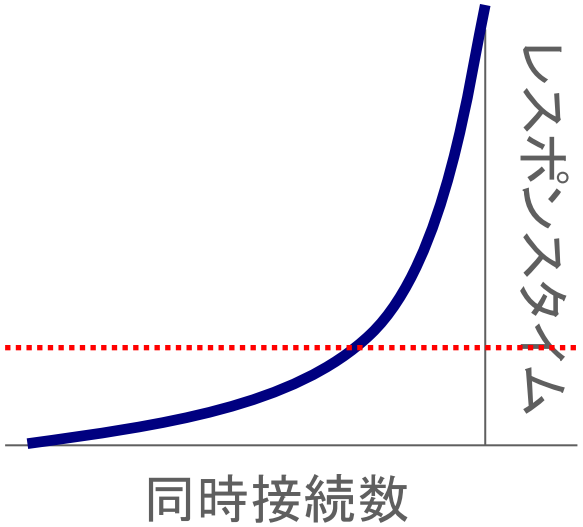
チューニングすべき点を明確にすることが重要

- スループット
  - 単位時間当たりの処理能力(例：TPS)
- レスポンスタイム
  - 処理を実行してから結果が返ってくるまでの時間

※レスポンスタイムの向上が、必ずしもスループットの向上につながるとは限らない
⇒例：特定処理のレスポンスタイム向上のためにインデックスを付けたところ、更新処理の
スループットが低下した (インデックスは、更新処理にとってはオーバーヘッドとなるため)

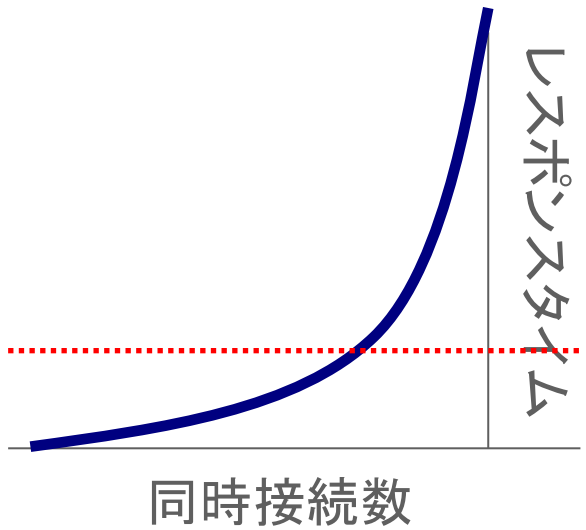
キューイング

- 複数のユーザ/リクエストがある場合に発生
  - 実行される前に「キュー」で待たさせる
- レスポンスタイム = キューイングによる遅延 + 実行時間
  - 実社会での例 – サポートセンターへの電話
    ⇒電話が混雑した場合、スループットは低下していなくても
    レスポンスタイムは悪化する
- “ホッケースティック” - システムが飽和状態に近づくと、
  キューイングによる遅延が急激に増大する



キューイング

- パフォーマンスの改善時に、キューイングによる遅延か実行時間のどちらかを改善する必要がある
  - 同時実行数を増やすことで、キューイングせずに実行できる処理が増加(例:オペレーターの人数を増加)
  - 実行時間を短縮することで、キューイングによる遅延を改善(例:オペレーターの対応時間を減少)



実行時間を計測する：ボトルネックを見極めることが重要

- 確認事項 – どこで実行時間が使われているか？
  - ネットワーク, CPU, ディスク, ロック…
- 直接的な計測方法
  - 例）Webのページを表示するためのクエリ実行時間の合計
- 間接的な計測方法
  - CPU利用率、ディスクIOのレイテンシ、ネットワークトラフィック、処理待ちプロセス数、同時実行中のクエリ数、etc . . .

ベンチマークテスト

- 重要なツール
  - アプリケーションのパフォーマンスの計測
  - 変更点のパフォーマンスに与える影響の確認
  - スケーラビリティの評価
- ただし。。。
  - 実行方法を間違えると誤った指針となりうる
  - 結果を正しく読み取ることができる必要がある
- ありがちな間違い
  - データサイズが本番環境では100GBなのに1GBでテスト
  - データやリクエストのばらつきを考慮しないでテスト
  - 1ユーザだけでテスト
  - 実際のアプリケーションの特性と異なるベンチマークテストの結果から、全力でパフォーマンスチューニング

MySQLのベンチマークツール

- 独自のベンチマークツールを作成
  - 一般ログからSQL文を作成する
  - MySQL ProxyやTCP DumpからSQL文を作成する
- mysqlslap
  - http://dev.mysql.com/doc/refman/5.6/ja/mysqlslap.html
  - http://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html
- SysBench
  - http://sysbench.sourceforge.net/

ビジネス面からの考慮

- どのようなパフォーマンスチューニングでもコストがかかる
- 他の可能性の検討
  - ハードウェアの交換の方が、アプリの修正より安いことも
- そのパフォーマンス、スケーラビリティ、信頼性は本当に必要？
  - 99.999% の可用性は 99.9% よりも格段にコストがかかる
  - ピーク時の性能は平常時の何倍必要なのか？
- 常に全体像を把握しておくこと
  - 「これが最大のリスク/ボトルネックなのか？」
- どのチューニングがビジネスにとって重要か確認すること
  - “全て”をチューニングすることは多くの場合に無駄
  - 次善策を取ったときのコストとビジネスへのインパクトは？

チューニングのアプローチ

- DBチューニング(全体最適)
  - サーバー全体のパフォーマンス(主にスループット)を向上させる
  - 主なチューニング要素はパラメータ

- SQLチューニング(個別最適)
  - 個別の処理のパフォーマンス(主にレスポンスタイム)を向上させる
  - SQLチューニングによって、数倍～数十倍に性能が向上することは珍しいことではない(DBをパフォーマンスよく利用する上で、SQLチューニングは凄く重要)

Program Agenda

1. チューニング概論
2. MySQL Serverチューニング TIPS
3. SQLチューニング TIPS
4. 参考情報

基本1: システム変数(サーバー設定)の確認

- MySQLサーバーの設定は、システム変数で設定する

```
mysql> show variables like 'auto%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
| autocommit               | ON    |
| automatic_sp_privileges  | ON    |
+--------------------------+-------+
4 rows in set (0.00 sec)
```

```
shell> mysqladmin -uroot -S /tmp/mysql.sock variables | grep auto
| auto_increment_increment                  | 1
| auto_increment_offset                     | 1
| autocommit                                | ON
| automatic_sp_privileges                   | ON
```

- 設定方法
  - オプションファイルで設定 : my.cnf / my.ini
  - 一時的な変更 : SET [GLOBAL] <variable>=<value>
    - セッション単位(LOCAL)、サーバー全体 (GLOBAL)での変更が可能
      (システム変数によっては、動的に変更できないものもある)

基本2: ステータス変数(稼働統計)の確認

- MySQLサーバーの動作を監視するためにステータス変数を確認する

```
mysql> show status like 'innodb_buf%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Innodb_buffer_pool_pages_data         | 142   |
| Innodb_buffer_pool_pages_dirty        | 0     |
```

```
shell> mysqladmin -uroot -S /tmp/mysql.sock extended
+-----------------------------------------+-------------+
| Variable_name                           | Value       |
+-----------------------------------------+-------------+
| Aborted_clients                         | 0           |
| Aborted_connects                        | 0           |
```

- 特定のクエリ(SQL)について調査する場合

  mysql> FLUSH STATUS; <クエリ実行>; SHOW STATUS;
- 定期的に確認する例(15秒間隔で、ステータス変数の差分のみ表示)

  shell> mysqladmin -u <ユーザー名> -p ex -i 15 -r | grep -v '0'

システム変数、ステータス変数のマニュアル、など

5.1.4 Server System Variables
http://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html
5.1.4. サーバーシステム変数
http://dev.mysql.com/doc/refman/5.6/ja/server-system-variables.html

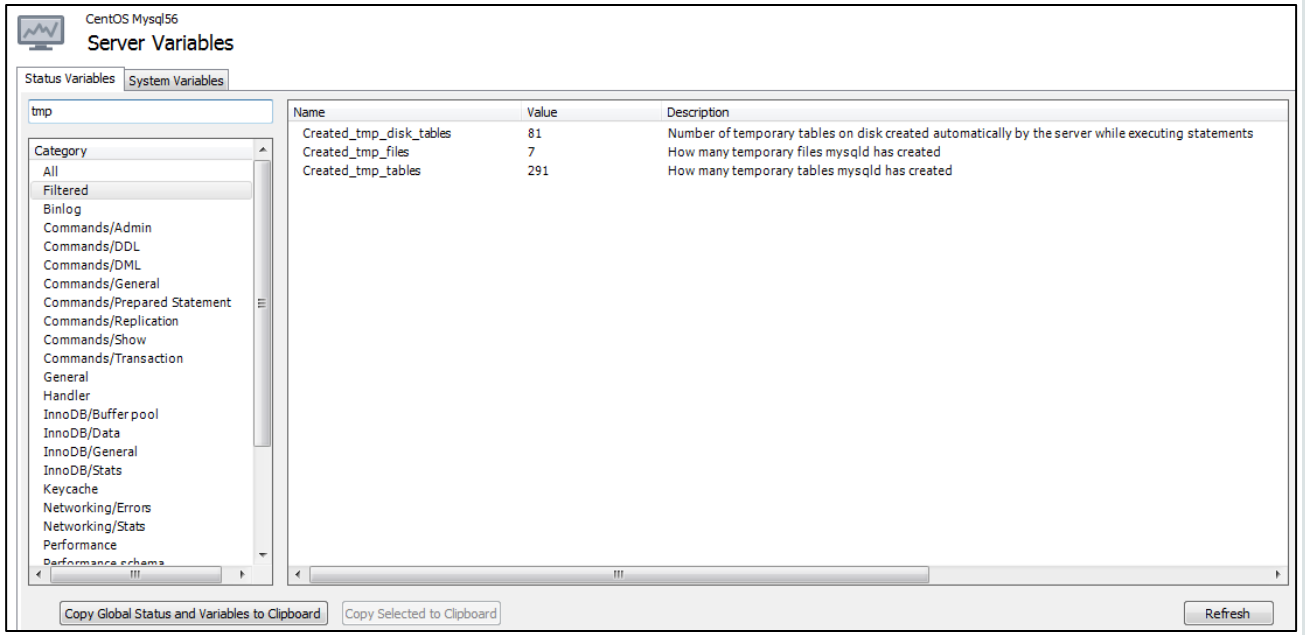
5.1.6 Server Status Variables
http://dev.mysql.com/doc/refman/5.7/en/server-status-variables.html
5.1.6 サーバーステータス変数
http://dev.mysql.com/doc/refman/5.6/ja/server-status-variables.html

※MySQL 5.6 と MySQL 5.7 のシステム変数の設定の違いについて解説した資料を以下で公開しています。

MySQL 5.7とMySQL 5.6設定パラメータ比較
http://downloads.mysql.com/presentations/20151030_05_MySQL-Parameter-comparison.pdf

補足：MySQL Workbenchからシステム変数/ステータス変数を簡単に確認可能

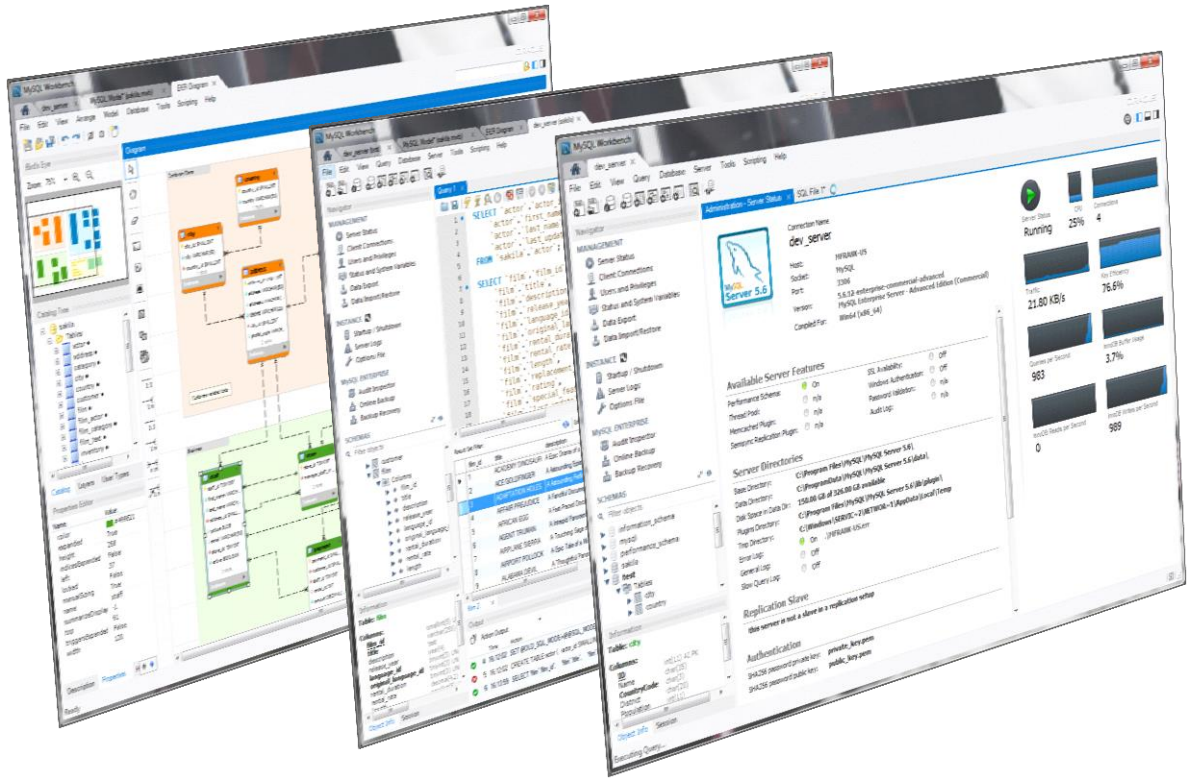
- システム変数/ステータス変数を確認するためのGUIがある
- ウィンドウに変数名の一部を入力すると、中間一致で絞り込める
- あらかじめ各変数がカテゴリー分けされているので、関連するシステム変数/ステータス変数をまとめて確認可能
- カスタムのカテゴリも作成可能であるため、頻繁に確認する変数をカテゴライズしておくと便利



補足：MySQL Workbench

MySQL Databaseの統合開発環境

- MySQLの公式GUIツール
- SQL開発、Server管理、
  データモデリングなどの機能が
  1つにまとめられたツール
- チューニングに役立つ機能も有り
  - ビジュアルEXPLAIN
  - オブジェクト定義の確認
  - 実行中のSQLをキャプチャ、、、など



補足：MySQL Workbench

MySQL Databaseの統合開発環境

- コミュニティ版と商用版があるが、大半の機能は
  コミュニティ版でも利用可能
  http://www-jp.mysql.com/products/workbench/features.html
- ダウンロード先
  https://dev.mysql.com/downloads/workbench/
- マニュアル
  http://dev.mysql.com/doc/index-gui.html
- ブログ
  http://mysqlworkbench.org

パフォーマンススキーマからもシステム変数/ステータス変数を簡単に確認可能

- SQL文でシステム変数、ステータス変数を確認できる
- システム変数の確認例

```
mysql> SELECT * FROM performance_schema.global_variables;
mysql> SELECT * FROM performance_schema.session_variables;
```

- ステータス変数の確認例

```
mysql> SELECT * FROM performance_schema.global_status;
mysql> SELECT * FROM performance_schema.session_status;
```

パフォーマンススキーマ

- 性能統計情報分析のためのしくみ
- performance_schemaストレージエンジンとして実装されている
- MySQLサーバ内の「イベント」毎の処理時間を記録
  - 処理時間はピコ秒単位での表示(ただし実際の精度はプラットフォームや設定による)
- その他必要となる情報を記録
  - 処理データ量
  - ソースコードでの位置
  - 各種メタデータ

sysスキーマ

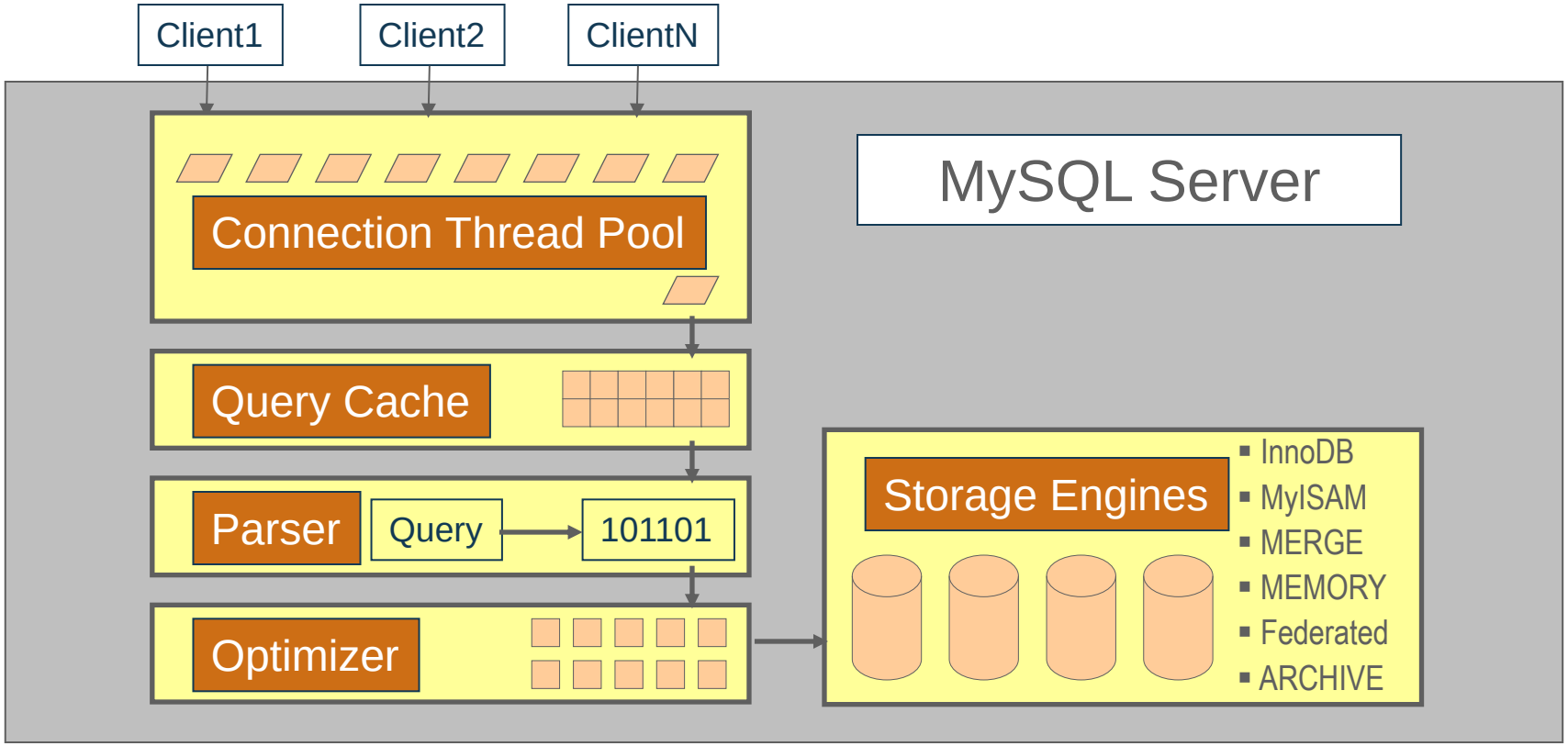
Performance Schemaとインフォメーションスキーマをシンプルなビューに

- パフォーマンススキーマを便利に使うための
  ビュー、プロシージャ-、ファンクションのセット
- データベース管理者のタスクを支援
  - 稼働統計や成長率などの主要な統計値の監視
  - 性能問題の検出、診断および改善
- sysスキーマのビューにより、シンプルに状況の詳細を確認
  - IO量の高いファイルや処理
  - コストの高いSQL文
  - テーブル、インデックス、スキーマの統計
  - レイテンシや待ち時間の分析、ロック
  - InnoDBの稼働統計

MySQL Server チューニング(全体最適)の流れ

- ステータス変数などの情報からMySQL Serverの稼働情報を確認し、システム変数の設定値が適切か判断する
  ⇒必要に応じて設定値を調整する

MySQL Serverのアーキテクチャ

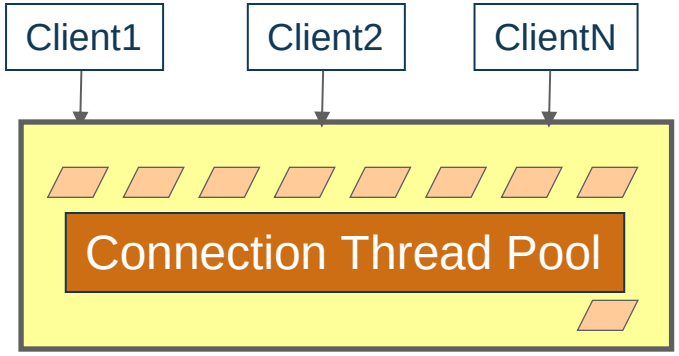


サーバのコネクション＆スレッド

設定ファイル my.cnf :

- **max_connections (151)**
  - サーバが許容可能なコネクション数
  - 多すぎるとメモリを消費しきる可能性あり
- **thread_cache_size (9) ※**
  - スレッドをコネクションの切断後にもキャッシュしておく数
  - 一般的には **max_connections/3**

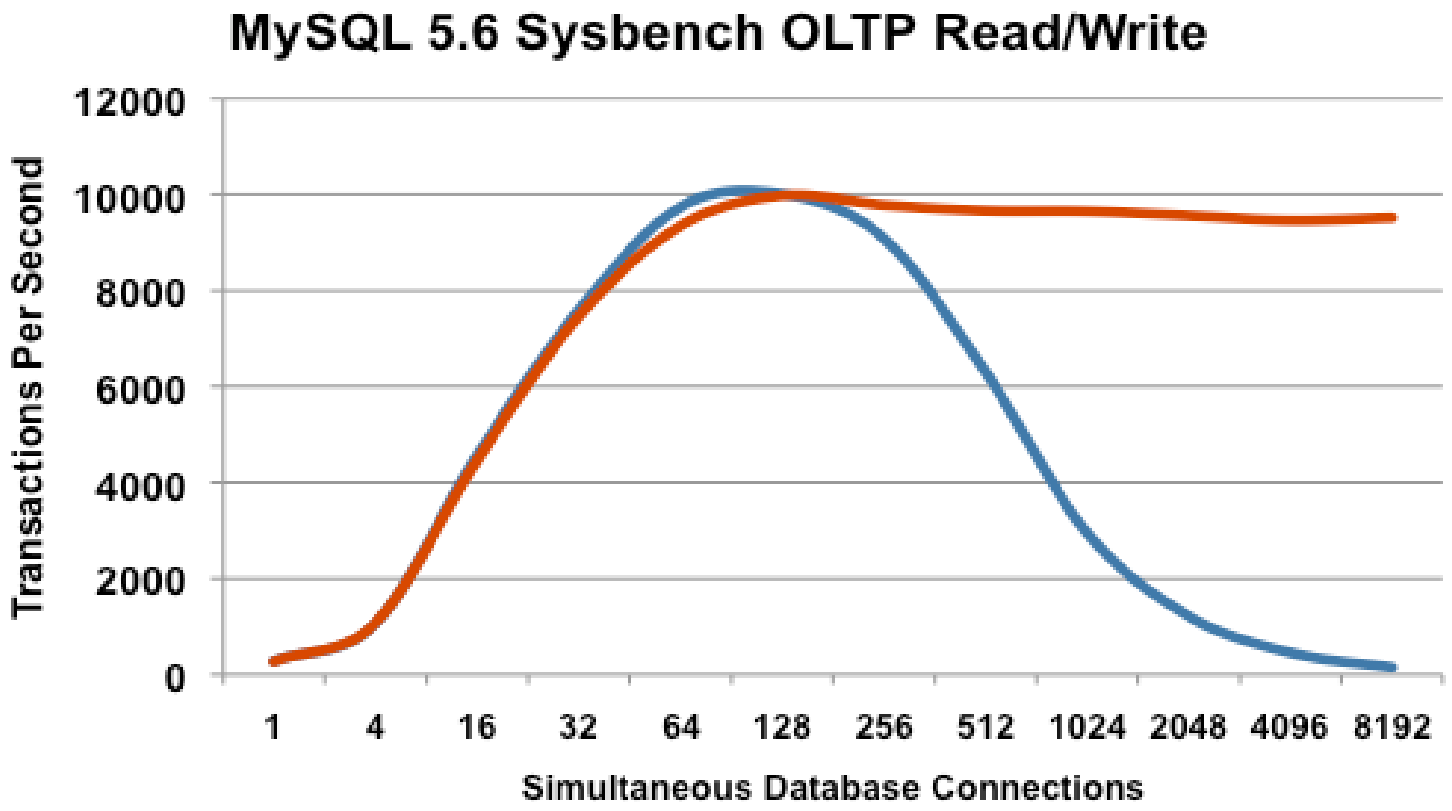
※以下計算式により自動計算
8 + (max_connections ／ 100)



- **mysql> show status;**
  - **Max_used_connections**
    - max_connections とあわせてチェック
  - **Threads_created**
    - thread_cache ミス
    - 低い数値であるべき

コネクション数(同時実行ユーザー数)が多い場合に有効な商用版限定機能

MySQL Enterprise Scalability (Thread Pool)



MySQL Enterprise Edition

Thread Pool有り

MySQL Community Edition

Thread Pool無し

MySQL 5.6.11
Oracle Linux 6.3、Unbreakable Kernel 2.6.32
4 sockets、24 cores、48 Threads
Intel(R) Xeon(R) E7540 2GHz CPUs
512GB DDR RAM

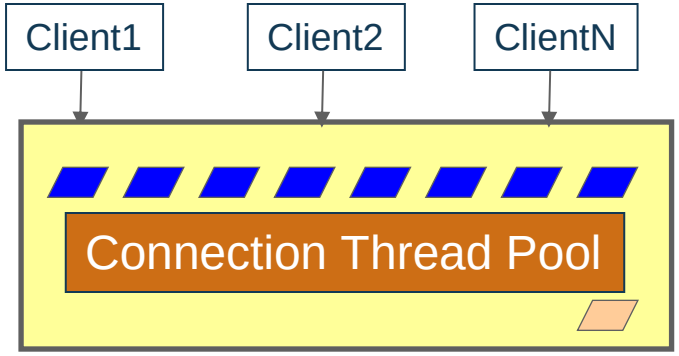
同時実行ユーザー数が増えても性能が落ちない

参照： MySQL Enterprise Scalability

コネクションスレッド毎のバッファ

設定ファイル my.cnf :

- **sort_buffer_size (256KB)**
  - ソート用のメモリサイズ。このサイズを超えるソートはディスクを利用。
  - MySQL 5.6でデフォルト値が縮小された (2MB⇒256KB)
- **その他のread, read_rnd, etc… バッファはデフォルトで問題ないケースも多い**
- **バッチ処理などの場合、処理実行前に動的にこれらのパラメタを変更可能**

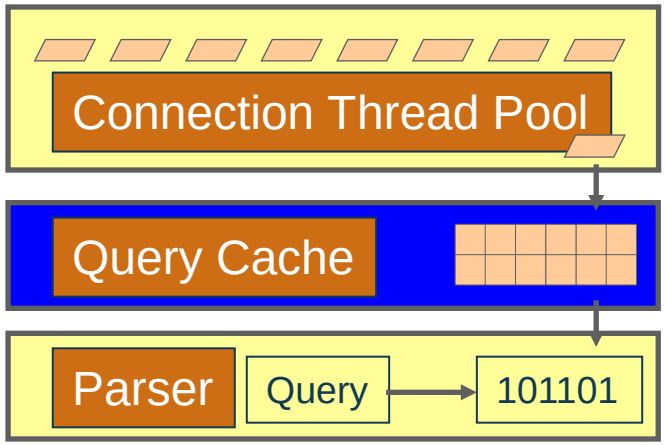


- **mysql> show status;**
  - **Sort_merge_passes**
    - ファイルを利用したマージソートのパス数
    - ソートがメモリ上だけで収まらない場合には要確認
    - インデックスの利用を検討

クエリキャッシュ

設定ファイル my.cnf :

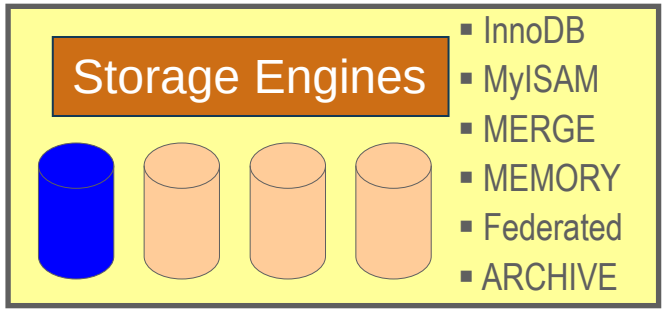
- **query_cache_size (1MB)**
  - クエリキャッシュに割り当てるメモリサイズ
  - 一般的には32MでOK
- **query_cache_type (OFF)**
  - 最悪のケースではパフォーマンスのオーバーヘッドが約15%
  - SELECTの比率が高いサーバで有効
  - DEMANDに設定すると、クエリ実行時にSQL_CACHE句を付けたクエリだけキャッシュ可能



- **mysql> show status;**
  - **Qcache_hits, Qcache_inserts**
    - キャッシュヒット/登録件数、ヒット率が小さければクエリキャッシュを無効にすることも検討
  - **Qcache_lowmem_prunes**
    - メモリ不足のためにキャッシュが削除された回数

InnoDB パフォーマンス Tips

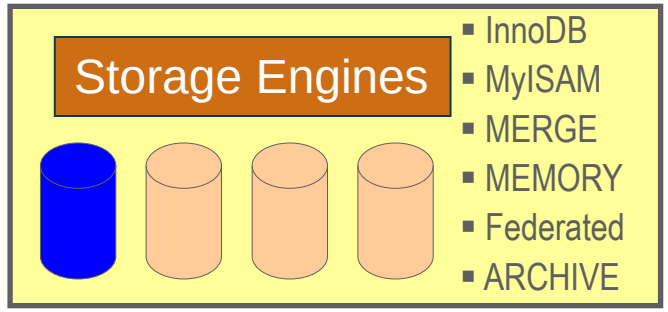
- **innodb_buffer_pool_size (128MB)**
  - MySQL&InnoDBのみを利用していれば、メインメモリの80%程度を割り当てる
  - データとインデックスの両方をキャッシュ
- **innodb_log_file_size (48MB)**
  - innodb_buffer_pool_sizeの25%～100%
  - ログファイルがどの程度頻繁に切り替わっているかをチェック
  - 値を大きくするとクラッシュ後のリカバリ時間が長くなる
- **innodb_file_per_table (ON)**
  - テーブル単位でOS上のファイルを分ける設定
  - ディスクI/Oの分散やibdataファイルの肥大化を防ぐために”ON”を推奨



- **mysql> SHOW ENGINE INNODB STATUS;**

  InnoDBの内部での稼働情報
  - ファイル IO
  - バッファプール
  - ログ情報
  - 行/ロック情報、など

InnoDB パフォーマンス Tips(続き)



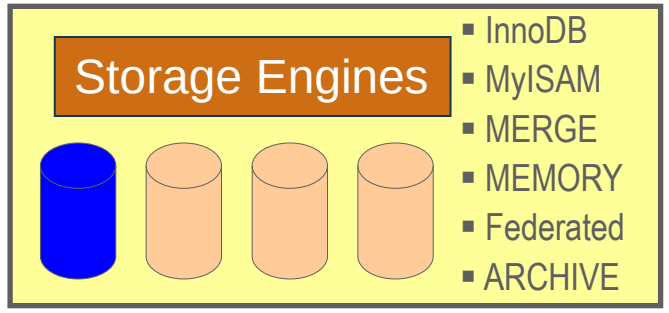
- **innodb_flush_log_at_trx_commit(1)**
  - 1 (遅い) コミット時にログをフラッシュ。真のACID
  - 2 (速い) コミット時にはOSのキャッシュにログを フラッシュ、ディスクとのシンクは毎秒1回
  - 0 (最速) ログを毎秒1回(またはそれ以下)フラッシュ
- **innodb_flush_method = O_DIRECT**
  - データファイルアクセス時にOSキャッシュを使用しない設定
  - Linux環境では、多くの場合O_DIRECTにした方がオーバーヘッドが下がる
- **innodb_buffer_pool_instances (1)**
  - mutex競合を避けるため
  - innodb_buffer_pool_sizeが大きくmutex競合がオーバーヘッドとなっている場合は、2以上に設定

補足：innodb_buffer_poolのmutex競合確認方法

- 確認例

```
mysql> select event_name, count_star, sum_timer_wait/1000000000 sum_timer_wait_ms
    ->  from performance_schema.events_waits_summary_global_by_event_name
    ->  where event_name like '%buf_pool_mutex%';
+-------------------------------------+------------+--------------------+
| event_name                          | count_star | sum_timer_wait_ms |
+-------------------------------------+------------+--------------------+
| wait/synch/mutex/innodb/buf_pool_mutex |          0 |             0.0000 |
+-------------------------------------+------------+--------------------+
1 row in set (0.00 sec)
```

InnoDB パフォーマンス Tips(続き)



- **innodb_io_capacity (200)**
  - InnoDBのバックグラウンドタスクに使用する I/Oキャパシティ(IOPS)の上限を設定する
  - 高速なストレージを使用している場合は拡大する
  - デフォルト値の200は、ストライプされた2本のディスクを目安にした値
- **innodb_read_io_threads (4)**
- **innodb_write_io_threads (4)**
  - 高速なストレージを使用している場合は拡大する
  - デフォルト値の4は、一般的には十分な値

補足：sysスキーマからInnoDB Buffer Poolの使用状況を確認できる

- 割り当てられているメモリサイズをテーブル単位で集計したビューとスキーマ単位で集計したビューがある
- スキーマ単位での確認例

```
mysql> select * from sys.innodb_buffer_stats_by_schema;
+---------------+------------+------------+-------+--------------+-----------+-------------+
| object_schema | allocated  | data       | pages | pages_hashed | pages_old | rows_cached |
+---------------+------------+------------+-------+--------------+-----------+-------------+
| employees     | 95.55 MiB  | 86.89 MiB  |  6115 |         6115 |      6115 |     2850143 |
| InnoDB System | 3.75 MiB   | 3.24 MiB   |   240 |          240 |       240 |        3121 |
| world         | 784.00 KiB | 538.18 KiB |    49 |           49 |        49 |        5205 |
| mysql         | 272.00 KiB | 52.65 KiB  |    17 |           17 |        17 |         561 |
+---------------+------------+------------+-------+--------------+-----------+-------------+
4 rows in set (0.13 sec)
```

- テーブル単位での確認例

```
mysql> select * from sys.innodb_buffer_stats_by_table;
```

スキーマーのデザイン

- 正規化
  - OLTPや書き込み多い環境
  - データの冗長性を削減
  - JOINのオーバーヘッド
  - トータルのデータサイズは小さくなる
  - E/R図とスムーズに連携
- 非正規化
  - OLAPやレポーティング
  - データの冗長性が高い
  - JOINを削減できる

- データ型
  - tinyint, smallint, mediumint, など、小さなデータ型を使いデータ量を削減
  - JOINに使う列は同じデータ型に
  - charではなくvarcharを使う
  - 可能なところはNOT NULLを宣言
  - INTが使える場合は、NUMERIC/DECIMALよりINTを優先する
- インデックス
  - 複数列
  - プレフィックス
  - "covering index"

Program Agenda

1. チューニング概論
2. MySQL ServerチューニングTIPS
3. SQLチューニングTIPS
4. 参考情報

用語解説

- 実行計画
  - SQL処理する時の内部的処理手順
    - SQLは内部的な処理手順を定めていないため、内部的な処理手順はRDBMSが決めている
  - 同じSQLであっても、実行計画が違えば大きくパフォーマンスが変わることがある
    - SQLチューニングによって、SQLのレスポンスタイムが大幅に向上する場合があるのは、より良い実行計画を選択することによる効果
- オプティマイザ
  - SQLの実行計画を作成する役割を持つ
  - MySQLでは、コストベースのオプティマイザを採用しているため、コストに基づいて実行計画を作成する
  - オプティマイザの判断が必ずしも最適だとは限らない
  - オプティマイザがより良い実行計画を作成できるように、SQLチューニングを行う

用語解説：統計情報とは？

- コストベースオプティマイザがコストを見積る際に参照するテーブル、インデックスの情報
  - 例）
    - テーブルに含まれる行数
    - 各インデックスのサイズ（ページ数）
    - 各インデックスのリーフページのサイズ（ページ数）
    - 各インデックスのカーディナリティ（何種類の値が存在するか？）
- 統計情報が同じであれば、オプティマイザは同じ実行計画を選択する（前提条件：オプティマイザ関連の設定を変更していない）
- 統計情報は、テーブル内のデータが10%更新されたタイミングで再計算される

SQLチューニングの流れ

1. 問題となるSQLの特定
   - スロークエリログ
   - SHOW FULL PROCESSLIST
   - パフォーマンススキーマ(sysスキーマ)
   - MySQL Query Analyzer
2. SQLの実行計画やSQL実行時の稼働統計等の確認
   - EXPLAIN、Visual EXPLAIN (MySQL Workbench)
   - SHOW STATUS、Query Statistics (MySQL Workbench)

SQLチューニングの流れ

3. SQLチューニング実施
   - INDEX作成(削除)
   - SQL オプティマイザの制御
     (ヒント句を使用して、JOIN順番を変更したり、特定のINDEXを使用させる)
   - システム変数調整
   - テーブル設計修正
   - SQL書換え、、、など

SQLチューニング : 問題となるSQLの特定

スロークエリログ

- 実行時間が指定した時間以上のクエリを出力する
- デフォルトでは出力されていない(システム変数log-slow-queries を設定して出力)
- long-query-time：秒単位で指定(0.5と指定すれば500ms)
- log-queries-not-using-indexes：インデックスを使っていないクエリを全て出力
- デフォルトの出力先はOS上のファイルだが、log-outputオプションで
  テーブルへも出力可能

注意事項

- クエリの実行が完了してからスロークエリログに出力される
  ⇒実行中のクエリは、スロークエリログでは確認できない

```
#Time: 08073101 16:25:24
#User@Host: root[root] @ localhost [127.0.0.1]
#Query_time: 8 Lock_time: 0 Rows_sent: 20 Rows_examined: 243661
SELECT part_num FROM `inventory`.`parts` WHERE
(`ven` = "foo") ORDER BY `delivery_datetime` DESC LIMIT 100;
```

mysqldumpslow

- スロークエリーログの集計ツール
- スロークエリログの出力が多い時に、優先してチューニングすべきクエリを特定する場合などに便利

使用例

– mysqldumpslow -s at <スロークエリログファイル名>

※"-s at"は、クエリー時間または平均クエリー時間でソートするオプション(デフォルト)

備考

- 特定時間帯のクエリーのみを抽出することは出来ない

⇒ MySQL Query Analyzerを使えば、任意の時間帯のクエリーを簡単に調査可能

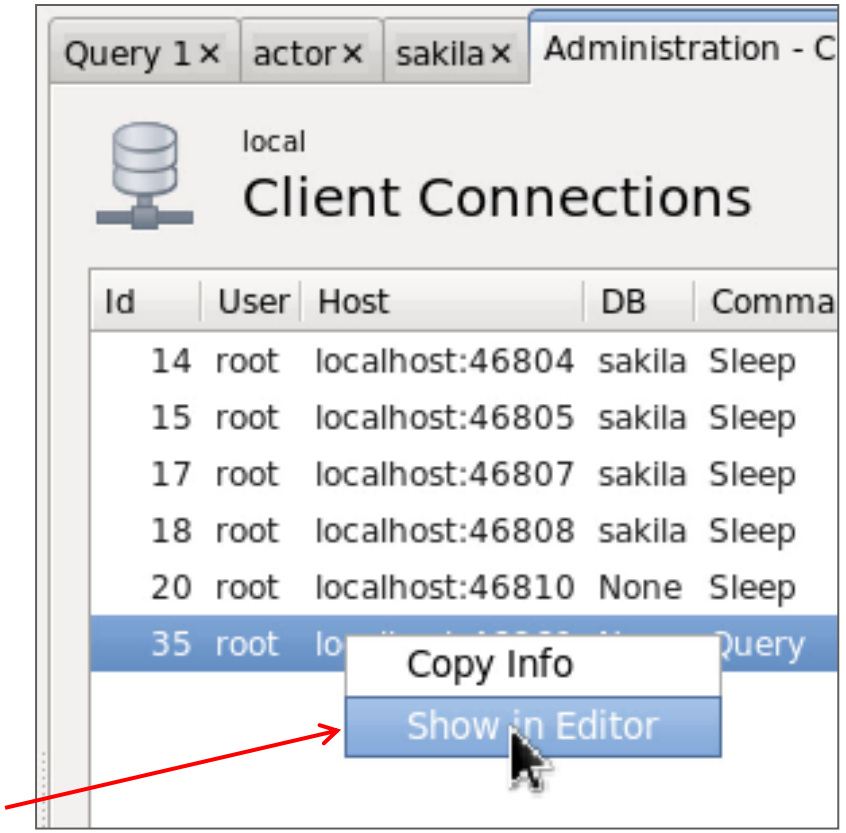
SHOW FULL PROCESSLIST

- メリット
  - 現在実行中の時間がかかっている クエリを特定可能
    - 「Command: Query」となっているものから、「Time」の値が大きなものを探す
  - クライアントホスト名など、問題のある クエリの追跡に役立つ情報も含まれる
- デメリット
  - 定常的に自動監視するためには、スクリプト等を作成する必要がある

```
mysql> SHOW FULL PROCESSLIST¥G
******** 1. row ******************
Id: 1
User: MyUser
Host: localhost
db: inventory
Command: Query
Time: 1030455
State: Sending Data
Info: SELECT part_num from 'inv';
   …..
2 rows in set (0.00 sec)
```

補足：MySQL WorkbenchからSHOW FULL PROCESSLISTを確認可能

- MySQL Workbenchの活用例
  - クエリを実行中のコネクションを探し、
    実行時間が長いクエリをSQLエディタで確認
    ⇒ビジュアルEXPLAINで実行計画を確認、
    スキーマツリーからテーブル定義や
    インデックス定義を確認
    ⇒SQLをチューニング



SHOW FULL PROCESSLISTの情報をGUIで確認し、
気になるコネクションを右クリック(Show in Editorを選択)

問題となるSQLの特定に使えるsysスキーマのビュー

- statement_analysis：SQL毎に実行時間、実行回数、トータル実行時間、平均実行時間、などを集計したビュー
- innodb_lock_waits：ロック待ちSQLの調査ができるビュー
  ブロックしているトランザクションも特定できる

sysスキーマ使用例:トータル実行時間が長いSQLの調査

```
mysql> SELECT * FROM sys.statement_analysis LIMIT 1¥G
*************************** 1. row ***************************
             query: SELECT * FROM `salaries` WHERE `emp_no` = ?
                db: employees
         full_scan:
        exec_count: 28
         err_count: 0
        warn_count: 0
     total_latency: 34.42 ms
       max_latency: 4.32 ms
       avg_latency: 1.23 ms
      lock_latency: 4.53 ms
         rows_sent: 290
     rows_sent_avg: 10
     rows_examined: 290
rows_examined_avg: 10
     rows_affected: 0
rows_affected_avg: 0
        tmp_tables: 0
   tmp_disk_tables: 0
      rows_sorted: 0
sort_merge_passes: 0
            digest: 0f9c46f90df1e6f06f923437ce0d3288
       first_seen: 2015-11-27 07:26:10
        last_seen: 2015-11-27 07:26:50
1 row in set (0.00 sec)
```

デフォルトではトータル実行時間が
長いものでソートされている

22.4.3.45 The user_summary_by_statement_latency and x$user_summary_by_statement_latency Views
https://dev.mysql.com/doc/refman/5.7/en/sys-user-summary-by-statement-latency.html

sysスキーマ使用例：ロック待ちSQLの調査

```
mysql> select * from sys.innodb_lock_waits¥G
*************************** 1. row ***************************
                wait_started: 2015-11-27 00:21:47
                    wait_age: 00:00:11
               wait_age_secs: 11
                locked_table: `world`.`City`
                locked_index: PRIMARY
                 locked_type: RECORD
             waiting_trx_id: 5956
        waiting_trx_started: 2015-11-27 00:21:39
            waiting_trx_age: 00:00:19
    waiting_trx_rows_locked: 2
  waiting_trx_rows_modified: 1
                waiting_pid: 15
              waiting_query: update City set Population=1780000 where ID=1
            waiting_lock_id: 5956:115:5:2
          waiting_lock_mode: X
            blocking_trx_id: 5955
               blocking_pid: 14
             blocking_query: NULL
           blocking_lock_id: 5955:115:5:2
         blocking_lock_mode: X
       blocking_trx_started: 2015-11-27 00:21:34
           blocking_trx_age: 00:00:24
   blocking_trx_rows_locked: 2
 blocking_trx_rows_modified: 2
    sql_kill_blocking_query: KILL QUERY 14
sql_kill_blocking_connection: KILL 14
1 row in set (0.00 sec)
```

ロック待ちしているSQL

22.4.3.9 The innodb_lock_waits and x$innodb_lock_waits Views
https://dev.mysql.com/doc/refman/5.7/en/sys-innodb-lock-waits.html

sysスキーマでの調査の注意点

- statement_analysisの情報は累積値であるため、mysqldumpslowと同じく特定時間帯のみのクエリーのみを抽出することはできない
  ⇒MySQL Query Analyzerを使えば、任意の時間帯のクエリーを簡単に調査可能

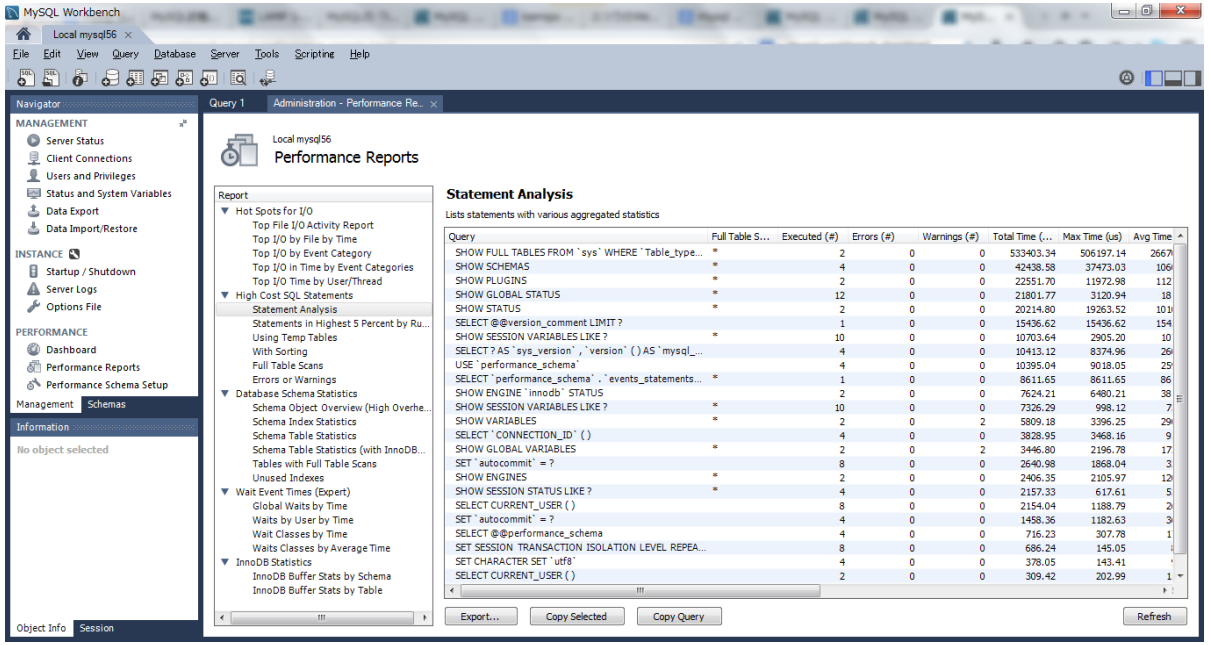
※sysスキーマのps_truncate_all_tablesプロシージャを使うと、簡単にパフォーマンススキーマの情報を初期化できる

- 使用例

```
mysql> CALL sys.ps_truncate_all_tables(FALSE);
+---------------------+
| summary             |
+---------------------+
| Truncated 44 tables |
+---------------------+
1 row in set (0.03 sec)
```

補足：問題となるSQLの特定にも役立つMySQL Workbench

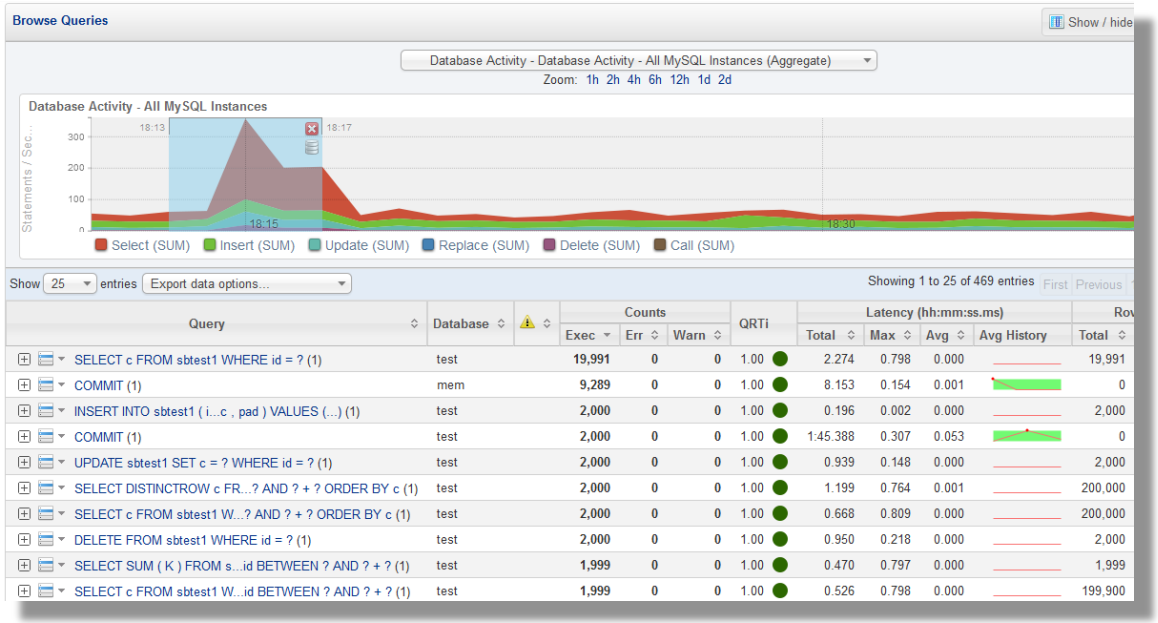
- MySQL Workbenchの"Performance Reports"から
  "High Cost SQL Statements"を選択（sysスキーマの情報を基にしている）



- 備考
  - mysqldumpslowと同じく、特定時間帯のクエリーのみを抽出することは出来ない
    ⇒ MySQL Query Analyzerを使えば、任意の時間帯のクエリーを簡単に調査可能

MySQL Query Analyzer(クエリ解析機能)

- 全てのMySQLサーバの
  全てのSQL文を一括監視
- vmstatなどのOSコマンドやMySQLの
  SHOWコマンドの実行、
  ログファイルの個別の監視は不要
- クエリの実行回数、エラー回数、
  実行時間、転送データ量などを一覧表示
- チューニングのための解析作業を省力化

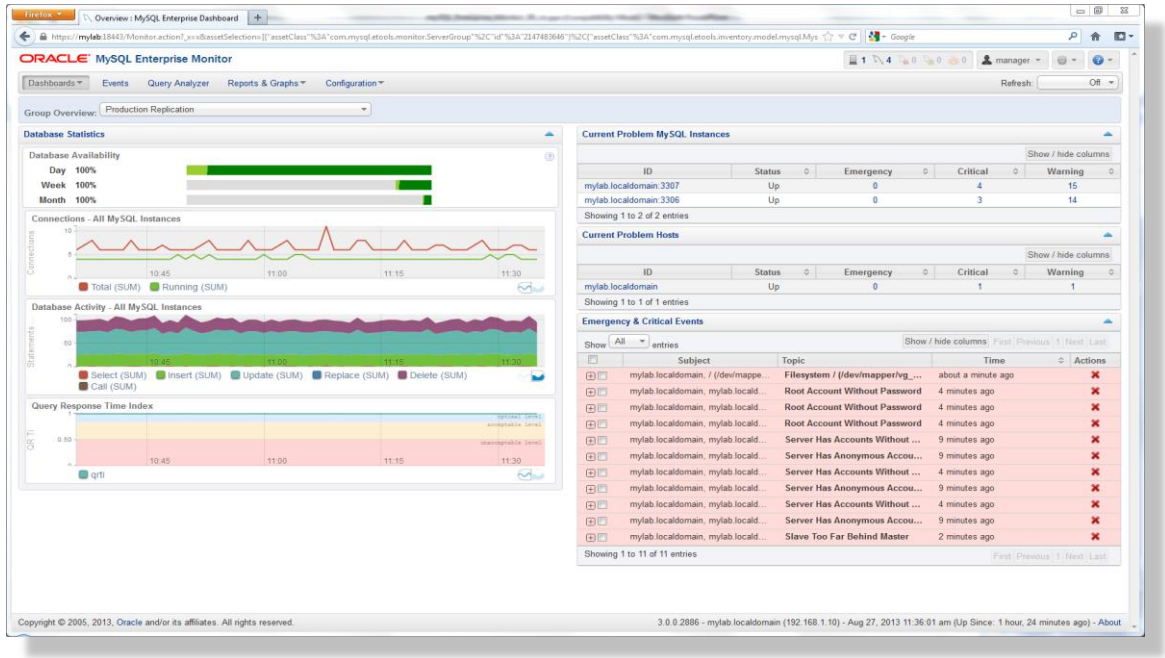


「「MySQL Query Analyzer を使用することで、問題のあるSQL コードを特定および解析して、データベースパフォーマンスを 3 倍に改善できました。さらに重要なことに、これは、何週間もかからずに、わずか3日で実現できました」」

Big Fish Games 社
ソフトウェア開発エンジニア
キース・ソーラダ氏 (Keith Souhrada)

MySQL Enterprise Monitor

- 複数のMySQLサーバを一括監視可能なダッシュボード
- システム中のMySQLサーバやレプリケーション構成を自動的に検出し監視対象に追加
- ルールに基づく監視と警告
- **問題が発生する前に通知**
- 問題のあるSQL文の検出、統計情報の分析が可能なQuery Analyzer



“バーチャルなMySQL DBA”アシスタント

参照： [MySQL Enterprise Monitor]

SQLチューニング :
SQLの実行計画やSQL実行時の稼働統計等の確認

EXPLAIN

- SQLの実行計画を確認可能
  - レコードへのアクセス方法？
  - どのインデックスを使用するか？
  - フェッチする行数がどれくらいか？
  - ファイルソートが発生していないか？、など
- SELECT/INSERT/UPDATE/DELETE/REPLACE に対して実行可能
- 実行計画が望ましくない場合は、以下の対応を行う等してチューニングする
  - インデックス追加/削除
  - テーブル定義/データ型変更
  - SQL書換え、など

```
EXPLAIN SELECT part_num
FROM `inventory`.`parts`
WHERE (`ven` = "foo")
ORDER BY `delivery_datetime`
DESC LIMIT 100;¥G

******** 1. row **************
           ID: 1
  select_type: SIMPLE
        table: parts
   partitions: NULL
         type: ref
possible_keys: ven, part#
          key: ven
      key_len: 3
          ref: null
         rows: 872
     filtered: NULL
        Extra: Using WHERE
1 row in set (0.00 sec)
```

EXPLAINの構文

```
EXPLAIN  [explain_type] {explainable_stmt | FOR CONNECTION connection_id}

explain_type: {
    FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

※便宜上割愛している構文もあります。
　厳密な構文は、以下のマニュアルでご確認下さい。

13.8.2 EXPLAIN Syntax
https://dev.mysql.com/doc/refman/5.7/en/explain.html
https://dev.mysql.com/doc/refman/5.6/ja/explain.html

EXPLAINの使用例

インデックス（主キー）で
アクセス出来ている

```
mysql> EXPLAIN SELECT * FROM world.City WHERE ID='1532';
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | City  | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

インデックスでアクセス出来ていない

```
mysql> EXPLAIN SELECT * FROM world.City WHERE District='Tokyo-to';
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | City  | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 4188 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> ALTER TABLE world.City ADD INDEX City_District(District);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

District列にインデックス追加

```
mysql> EXPLAIN SELECT * FROM world.City WHERE District='Tokyo-to';
+----+-------------+-------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key           | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | City  | NULL       | ref  | City_District | City_District | 20      | const |   18 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

インデックスでアクセス出来るようになった

EXPLAINの使用例（前ページで参照しているテーブルの情報）

```
mysql> SHOW CREATE TABLE world.City¥G
*************************** 1. row ***************************
       Table: City
Create Table: CREATE TABLE `City` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`) REFERENCES `Country` (`Code`)
) ENGINE=InnoDB AUTO_INCREMENT=4080 DEFAULT CHARSET=latin1

mysql> SELECT COUNT(*) FROM world.City;
+----------+
| COUNT(*) |
+----------+
|     4079 |
+----------+

mysql> SELECT COUNT(*) FROM world.City WHERE District='Tokyo-to';
+----------+
| COUNT(*) |
+----------+
|       18 |
+----------+
```

補足：EXPLAINの各項目

- テーブルごとに1行出力される

列	意味
id	クエリのID（テーブルのIDではないので注意）
select_type	クエリの種類
table	対象のテーブル
partitions	対象のパーティション（パーティションテーブルでない場合はNULLが出力される）
type	レコードアクセスタイプ（どのようにテーブルにアクセスされるかを示す）
possible_keys	利用可能なインデックス
key	選択されたインデックス
key_len	選択されたインデックスの長さ
ref	インデックスと比較される列
rows	行数の概算見積もり
filtered	条件によってフィルタリングされる行の割合
Extra	追加情報

補足：select_typeの出力でクエリの種類を把握する

select_typeの値	意味
SIMPLE	単純なSELECT（UNION やサブクエリーを使用しない）
PRIMARY	もっとも外側のSELECT
UNION	UNION内の2つめ以降のSELECTステートメント
DEPENDENT UNION	UNION内の2つめ以降のSELECTステートメントで、外側のクエリーに依存する
UNION RESULT	UNIONの結果
SUBQUERY	サブクエリー内の最初のSELECT
DEPENDENT SUBQUERY	サブクエリー内の最初のSELECTで、外側のクエリーに依存する
DERIVED	派生テーブルSELECT（FROM句内のサブクエリー）
MATERIALIZED	実体化されたサブクエリー
UNCACHEABLE SUBQUERY	結果をキャッシュできず、外側のクエリーの行ごとに再評価される必要があるサブクエリー
UNCACHEABLE UNION	キャッシュ不可能なサブクエリー（UNCACHEABLE SUBQUERY）に属するUNION内の2つめ以降のSELECT

補足：typeの出力でレコードへのアクセスタイプを把握する

typeの値	意味
system	1行しかないテーブル(systemテーブル)　※constの特殊な例
const	PRIMARY/UNIQUEインデックスによる等価検索（一意検索）
eq_ref	PRIMARY/UNIQUEインデックスによるJOIN
ref	ユニークでないインデクスによる等価検索、JOIN
fulltext	全文検索インデックスを使用した全文検索
ref_or_null	ユニークでないインデックスによる等価検索とIS NULLのOR
index_merge	複数のインデックスをマージ
unique_subquery	サブクエリ内で、PRIMARY/UNIQUEインデックスで等価検索（一意検索）
index_subquery	サブクエリ内で、ユニークでないインデクスによる等価検索、
range	範囲検索
index	インデックスのフルスキャン
ALL	フルテーブルスキャン

望ましい

望ましくない

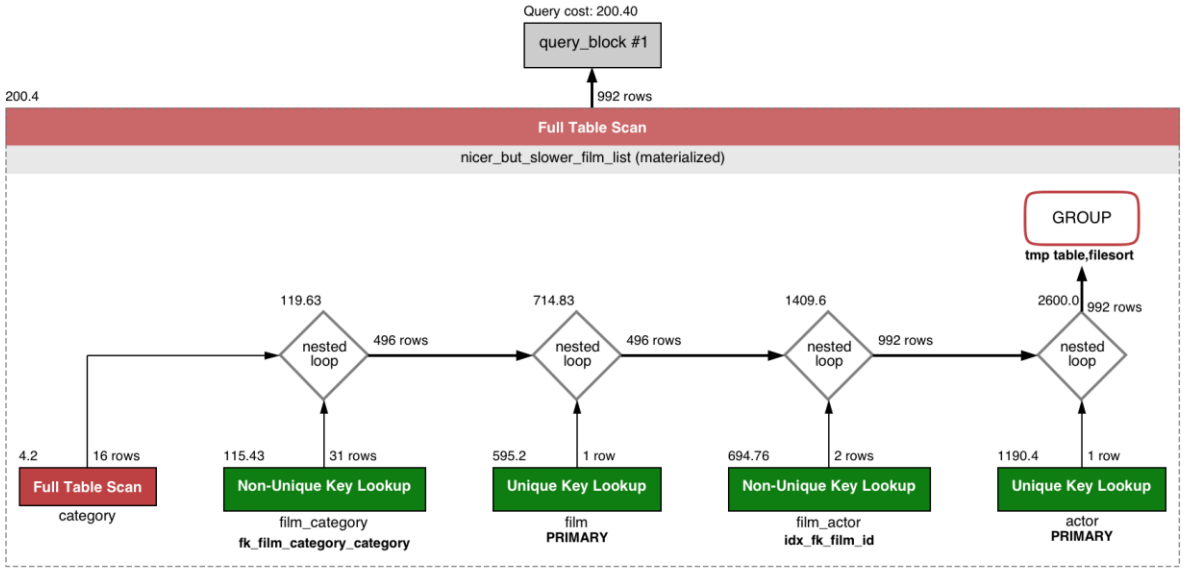
補足：Extraの出力でクエリーの処理方法を確認する

Extraの値	意味
Using filesort	ファイルソート（クイックソート）によってソートをする
Using index	インデックスにアクセスするだけでクエリの結果を返せる（テーブルへアクセス不要）
Using index condition	インデックスコンディションプッシュダウンを行う （ストレージエンジン側でWHERE句による絞込みを行う）
Using index for group-by	GROUP BY または DISTINCT を使ったクエリーを、インデックスだけで処理できる
Using join buffer (Block Nested Loop) Using join buffer (Batched Key Access)	JOINバッファを使用して結合処理を行う 結合アルゴリズムによって、Block Nested Loop / Batched Key Access が出力される
Using MRR	MMR(Multi-Range Read)最適化を行う
Using temporary	クエリーを処理するために、一時テーブルを作成する必要がある
Using where	行をフェッチした後、WHERE句の条件によってさらに絞り込みを行う

※上記は代表的な出力のみを掲載しています。詳細は以下のマニュアルをご確認下さい。
8.8.2 EXPLAIN Output Format : EXPLAIN Extra Information
https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain-extra-information
https://dev.mysql.com/doc/refman/5.6/ja/explain-output.html#explain-extra-information

Visual EXPLAIN

- MySQL WorkbenchからVisual EXPLAINを取得可能
- オブジェクトへのアクセスタイプを一目で確認可能(typeの値を色で判別可能)
- JOINの順番も一目で分かる



※チュートリアル：7.5 Tutorial: Using Visual Explain to improve query performance
http://dev.mysql.com/doc/workbench/en/wb-tutorial-visual-explain-dbt3.html

Visual EXPLAIN（typeの値による色の違い）

typeの値	色	意味
system	青色	1行しかないテーブル(systemテーブル)　※constの特殊な例
const	青色	PRIMARY/UNIQUEインデックスによる等価検索（一意検索）
eq_ref	緑色	PRIMARY/UNIQUEインデックスによるJOIN
ref	緑色	ユニークでないインデクスによる等価検索、JOIN
fulltext	黄色	全文検索インデックスを使用した全文検索
ref_or_null	緑色	ユニークでないインデックスによる等価検索とIS NULLのOR
index_merge	緑色	複数のインデックスをマージ
unique_subquery	橙色	サブクエリ内で、PRIMARY/UNIQUEインデックスで等価検索（一意検索）
index_subquery	橙色	サブクエリ内で、ユニークでないインデクスによる等価検索、
range	橙色	範囲検索
index	赤色	インデックスのフルスキャン
ALL	赤色	フルテーブルスキャン

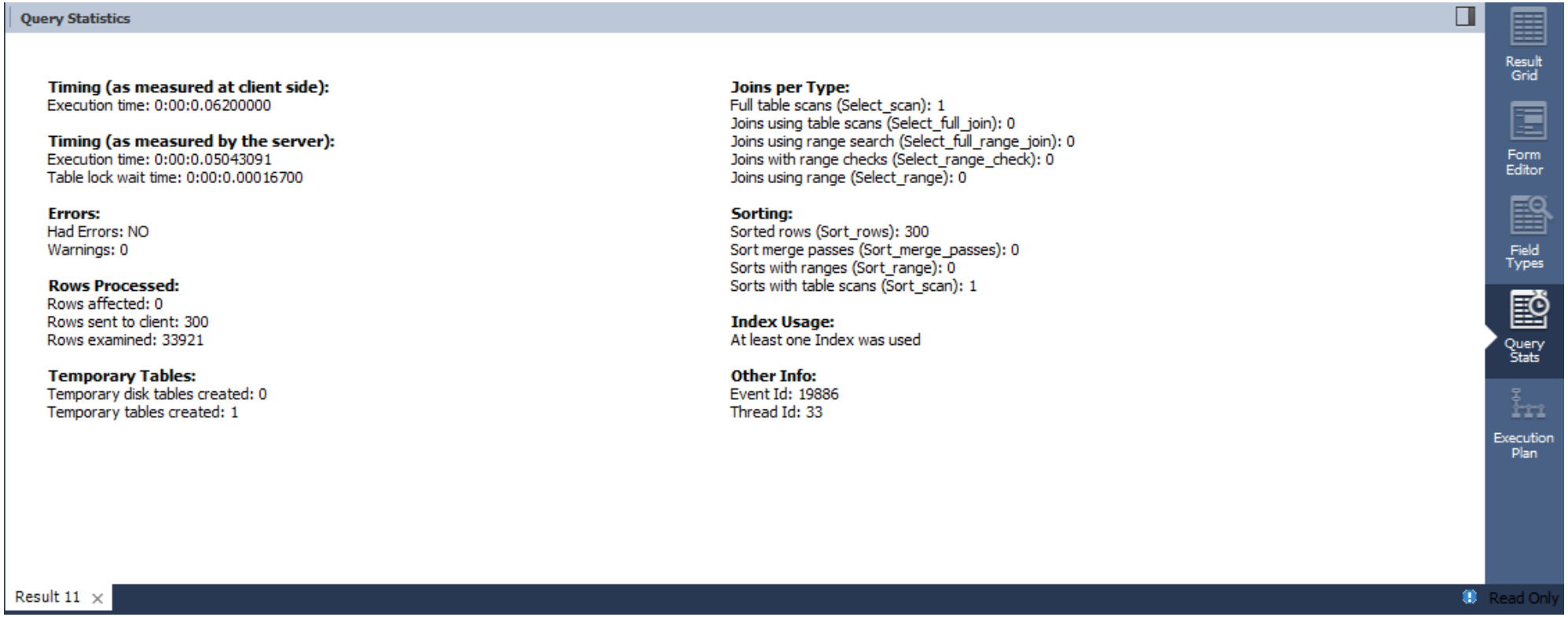
SHOW STATUS

- 特定のクエリ(SQL)についてステータス変数を調査
  mysql> FLUSH STATUS; <クエリ実行>; SHOW STATUS;

- 確認ポイント例
  - Sort_merge_passes
    - ファイルを利用したマージソートのパス数
    - カウントアップされている場合は、インデックスを使ってクエリを処理できないないか検討する
    - インデックスが使えない場合は、sort_buffer_sizeを拡張する
  - Created_tmp_disk_tables
    - 一時テーブルをディスク上に作成した回数
    - カウントアップされている場合は、一時テーブルをメモリ上に作成するために
      tmp_table_size、max_heap_table_sizeを拡張する

Query Statistics

- MySQL WorkbenchのQuery Statisticsから、SQLチューニング時に確認すべき基本的な情報をまとめて確認できる



SQLチューニング : SQLチューニング実施

SQLチューニングの基本

- インデックスが使えていないクエリーは、インデックスを使って処理できないか検討する
  - インデックスを使うことで、表の中から少量のデータを高速に取り出せる
  - 大量データにアクセスする場合(※)は、インデックスを使わない方が高速になる

    ※例：表データの全件を取得する場合
  - UPDATEでインデックスが使えていない場合は、ロック待ちを過剰に発生させる可能性があるので、要注意
    - InnoDBでは、処理した行では無く、アクセスした行に対してロックを取得するため、1件しか更新しないUPDATE文であっても、インデックスが使えていない場合はテーブルロックになってしまう
- 3テーブル以上JOINする時は、結果セットが少量のテーブルからJOINする方が効率が良い(より絞込みができるテーブルからJOINする)
  - JOIN対象の列で絞込みをする場合など、WHERE句による絞込みをどのテーブルに対して実施する方が効率的かも考えて指定する

インデックスの考慮事項

- インデックスを付け過ぎない
  - インデックスは参照時の性能は向上するが、更新時はオーバーヘッドになる
  - 重複するようなインデックスは利用しない
    - key(a, b) があるなら key(a) は削除
  - カーディナリティが低い(取りうる値の種類が少ない)列にはインデックスを付けない
    - 例）性別に対するインデックス
- サイズの小さなインデックスを活用する
  - プレフィックス index(name(8))
- MySQLはインデックス内で順序が先の列のみ利用可能
  - key (a,b)　.... where b=5 はインデックスを使わない

インデックスの考慮事項(続き)

- ユニークなインデクスにはUNIQUE キーワードをつける
- BTREE インデックスはソートされた結果を返すため、インデックスを使ってデータにアクセスすることで、内部的にソート処理を省略できる場合も有る
  - select * from t where b=5 order by c … key(b,c) optimal
- マルチカラムインデックスを活用する
  - MySQLのオプティマイザが同時に利用できるインデックスは基本的に1つだけ
- “covering indexes”はインデックスにアクセスするだけで必要なデータを取り出せるため、高速
  - select c from t where b=5 …の場合、key(b,c) が良い
- OPTIMIZE TABLE … でインデックスの再編成(最適化)が可能

オプティマイザの制御

- ヒントを使うことで、オプティマイザに指示を出して実行計画を変更できる

- ヒントの種類
  - インデックスヒント
    - 使用する（使用しない）インデックスを指定
  - STRAIGHT_JOINヒント
    - JOINの順番を指定
  - オプティマイザヒント
    - インデックスヒント、STRAIGHT_JOINヒント では指定できない、より詳細な制御が可能

インデックスヒント

- USE INDEX、FORCE INDEX、IGNORE INDEX の3種類がある
- FOR句を使って、スコープを指定することも可能
  - JOINのため、ORDER BYのため、GROUP BYのため
- USE INDEX
  - 特定のインデックスを使用するように、オプティマイザに指示を出す
- FORCE INDEX
  - USE INDEXに似ているが、USE INDEXに加えて「テーブルスキャンを選択しない」という指示を出す
- IGNORE INDEX
  - 特定のインデックスを使用しないように、オプティマイザに指示を出す

インデックスヒントの使用例

```
mysql> EXPLAIN SELECT * FROM world.Country WHERE Continent='Africa' OR Continent='Asia';
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | Country | NULL       | ALL  | Cont          | NULL | NULL    | NULL |  239 |    45.61 | Using where |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

インデックスを使っていない

```
mysql> EXPLAIN SELECT * FROM world.Country USE INDEX(Cont) WHERE Continent='Africa' OR Continent='Asia';
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | Country | NULL       | ALL  | Cont          | NULL | NULL    | NULL |  239 |    45.61 | Using where |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

USE INDEXヒントでもインデックスを使わなかった
（オプティマイザがテーブルスキャンの方が効率的と判断した）

インデックスヒントの使用例

```
mysql> EXPLAIN SELECT * FROM world.Country FORCE INDEX(Cont) WHERE Continent='Africa' OR Continent='Asia';
+----+-------------+---------+------------+-------+---------------+------+---------+------+------+----------+-----------------------+
| id | select_type | table   | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+---------+------------+-------+---------------+------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | Country | NULL       | range | Cont          | Cont | 1       | NULL |  109 |   100.00 | Using index condition |
+----+-------------+---------+------------+-------+---------------+------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

FORCE INDEXヒントでインデックスの使用を強制し、
インデックスを使うようになった

[注意事項]
この例は、インデックスヒントの使用方法を説明するための例であり、チューニングの例としては不適切です。
（絞込みがあまりできない場合は、インデックススキャンの方がテーブルスキャンより非効率であるため）

補足：インデックスヒントの構文

```
tbl_name [[AS] alias] [index_hint_list]

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

※詳細は以下のマニュアルでご確認下さい。

8.9.4 Index Hints
https://dev.mysql.com/doc/refman/5.7/en/index-hints.html
13.2.9.3 インデックスヒントの構文
https://dev.mysql.com/doc/refman/5.6/ja/index-hints.html

STRAIGHT_JOINヒント

- JOIN（結合）の順番を指定できるヒント
- 指定すると、左側のテーブルが右側のテーブルより先に読み取られる
  - MySQLがJOINで使用するアルゴリズムは Nested Loop JOIN とその改良系
  - Nested Loop JOIN では、基本的に先に読み取る表（駆動表）が後に読み取る表（内部表）よりも小さい方が効率的に結合できる
  - 3テーブル以上JOINする時は、結果セットが少量のテーブルからJOINする方が効率が良い(より絞込みができるテーブルからJOINする)
- OUTER JOIN（外部結合）の時は使えない
  - OUTER JOINの時は、結合条件に一致しない行を含むテーブルを先に読み取る必要があるため

STRAIGHT_JOINヒントの使用例

```
mysql> EXPLAIN SELECT CountryLanguage.Language,Country.Name,COUNT(*)
    ->  FROM City JOIN Country ON City.CountryCode=Country.Code
    ->   JOIN CountryLanguage ON City.CountryCode=CountryLanguage.CountryCode
    ->  GROUP BY CountryLanguage.Language,Country.Name
    ->  ORDER BY CountryLanguage.Language;
+----+-------------+-----------------+------------+------+---------------------+-------------+---------+  <<次ページへ続く>>
| id | select_type | table           | partitions | type | possible_keys       | key         | key_len |  <<次ページへ続く>>
+----+-------------+-----------------+------------+------+---------------------+-------------+---------+  <<次ページへ続く>>
|  1 | SIMPLE      | Country         | NULL       | ALL  | PRIMARY             | NULL        | NULL    |  <<次ページへ続く>>
|  1 | SIMPLE      | CountryLanguage | NULL       | ref  | PRIMARY,CountryCode | CountryCode | 3       |  <<次ページへ続く>>
|  1 | SIMPLE      | City            | NULL       | ref  | CountryCode         | CountryCode | 3       |  <<次ページへ続く>>
+----+-------------+-----------------+------------+------+---------------------+-------------+---------+  <<次ページへ続く>>
3 rows in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT CountryLanguage.Language,Country.Name,COUNT(*)
    ->  FROM City STRAIGHT_JOIN Country ON City.CountryCode=Country.Code
    ->   STRAIGHT_JOIN CountryLanguage ON City.CountryCode=CountryLanguage.CountryCode
    ->  GROUP BY CountryLanguage.Language,Country.Name
    ->  ORDER BY CountryLanguage.Language;
+----+-------------+-----------------+------------+--------+---------------------+-------------+---------+  <<次ページへ続く>>
| id | select_type | table           | partitions | type   | possible_keys       | key         | key_len |  <<次ページへ続く>>
+----+-------------+-----------------+------------+--------+---------------------+-------------+---------+  <<次ページへ続く>>
|  1 | SIMPLE      | City            | NULL       | index  | CountryCode         | CountryCode | 3       |  <<次ページへ続く>>
|  1 | SIMPLE      | Country         | NULL       | eq_ref | PRIMARY             | PRIMARY     | 3       |  <<次ページへ続く>>
|  1 | SIMPLE      | CountryLanguage | NULL       | ref    | PRIMARY,CountryCode | CountryCode | 3       |  <<次ページへ続く>>
+----+-------------+-----------------+------------+--------+---------------------+-------------+---------+  <<次ページへ続く>>
3 rows in set, 1 warning (0.00 sec)
```

JOIN順番がFROM句で指定した順番になっている

STRAIGHT_JOINヒントの使用例（EXPLAIN出力の続き）

```
--------------------+------+----------+---------------------------------+
 ref                | rows | filtered | Extra                           |
--------------------+------+----------+---------------------------------+
 NULL               |  239 |   100.00 | Using temporary; Using filesort |
 world.Country.Code |    4 |   100.00 | Using index                     |
 world.Country.Code |   18 |   100.00 | Using index                     |
--------------------+------+----------+---------------------------------+
```

[注意事項]
この例は、STRAIGHT_JOINヒントの使用方法を
説明するための例であり、チューニングの
例としては不適切です。
（内部表が小さい方がNested Loop JOINは効率的）

```
------------------------+------+----------+-----------------------------------------------+
 ref                    | rows | filtered | Extra                                         |
------------------------+------+----------+-----------------------------------------------+
 NULL                   | 4188 |   100.00 | Using index; Using temporary; Using filesort |
 world.City.CountryCode |    1 |   100.00 | NULL                                          |
 world.City.CountryCode |    4 |   100.00 | Using index                                   |
------------------------+------+----------+-----------------------------------------------+
```

補足:オプティマイザヒント

ヒント名	説明	スコープ
BKA, NO_BKA	BKA(Batched Key Access) join の有効/無効	クエリーブロック, テーブル
BNL, NO_BNL	BNL(Block Nested-Loop) join の有効/無効	クエリーブロック, テーブル
MAX_EXECUTION_TIME	クエリーの実行時間制限	グローバル
MRR, NO_MRR	MRR(Multi-Range Read) の有効/無効	テーブル, インデックス
NO_ICP	ICP(Index Condition Pushdown) の有効/無効	テーブル, インデックス
NO_RANGE_OPTIMIZATION	インデックスレンジスキャン、インデックスマージ、ルースインデックススキャン(Using index for group-by)の無効	テーブル, インデックス
QB_NAME	クエリーブロック(1つ1つのサブクエリ)に名前を付ける QB_NAMEヒントで名付けた名前を、他のヒント(BKAなど)で指定できる	クエリーブロック
SEMIJOIN, NO_SEMIJOIN	純結合変換による最適化の有効/無効 最適化を行う場合に指摘できる方式は以下の4種類 DUPSWEEDOUT, FIRSTMATCH, LOOSESCAN, MATERIALIZATION	クエリーブロック
SUBQUERY	サブクエリーの最適化方法を指定(INTOEXISTS or MATERIALIZATION)	クエリーブロック

※参考マニュアル:8.9.3 Optimizer Hints
https://dev.mysql.com/doc/refman/5.7/en/optimizer-hints.html

チューニングが上手くいかない場合は？

- MySQL Standard Edition/Enterprise Edition(商用版)の契約があれば、チューニングに関する問合せもサポート対象になるため、弊社のサポートエンジニアからアドバイスを受けることが可能
- 他にも、パーティショニング設計のレビューやスキーマ設計のレビュー等も対応可能
- 詳細はこちらをご確認下さい
  - MySQL コンサルティング・サポート
    https://www-jp.mysql.com/support/consultative.html

Program Agenda

1. チューニング概論
2. MySQLチューニングTIPS
3. SQLチューニングTIPS
4. 参考情報

MySQL Enterprise Edition

MySQL Enterprise Edition

ビジネス・クリティカルな環境において、最高レベルのMySQLスケーラビリティ、セキュリティ、信頼性、アップタイムを実現し、ビジネス・クリティカルな環境においてリスクとコストの削減を実現

- **MySQL導入の最適化**
- **ROIの最適化をサポート**
- **ユーザビリティ・顧客満足の向上**

MySQL Enterprise Edition のサービスカテゴリー

拡張機能

- 拡張性
- 高可用性
- セキュリティ
- 監査
- 暗号化

管理ツール

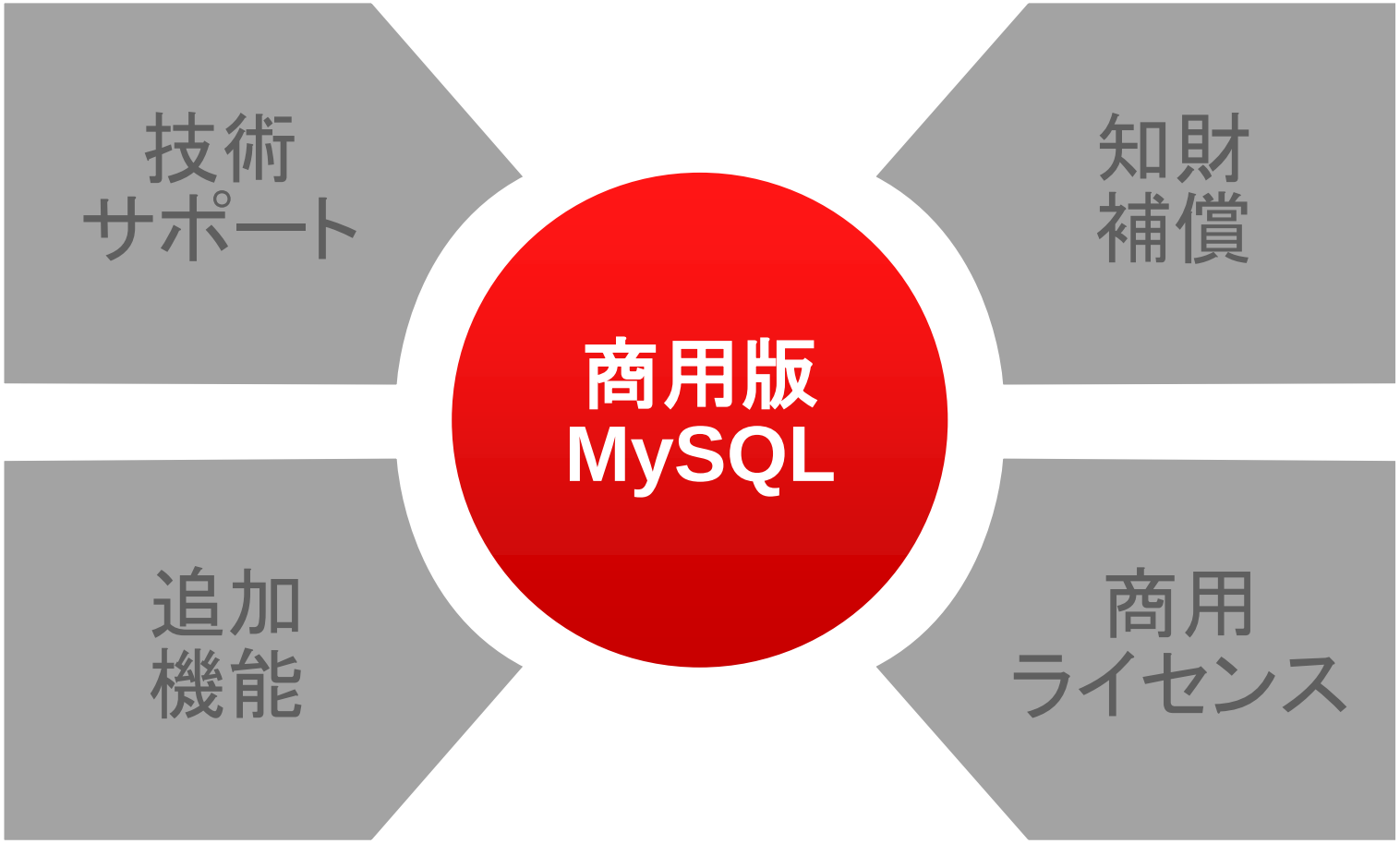
- 監視
- バックアップ
- 開発
- 管理
- マイグレーション

サポート

- 技術サポート
- コンサルティングサポート
- オラクル製品との動作保証

商用版MySQLがご提供する価値

費用対効果の高い付加価値



	MySQL Editions		
	Standard Edition	Enterprise Edition	Cluster CGE
機能概要			
MySQL Database	✔	✔	✔
MySQL Connectors	✔	✔	✔
MySQL Replication	✔	✔	✔
MySQL Fabric, MySQL Utilities		✔	✔
MySQL Partitioning		✔	✔
MySQL Router		✔	✔
Storage Engine: MyISAM, InnoDB	✔	✔	✔
Storage Engine: NDB (ndbcluster)			✔
MySQL Workbench SE/EE*	✔	✔	✔
MySQL Enterprise Monitor*		✔	✔
MySQL Enterprise Backup*		✔	✔
MySQL Enterprise Authentication (外部認証サポート) *		✔	✔
MySQL Enterprise Audit (ポリシーベース監査機能) *		✔	✔
MySQL Enterprise Encryption（非対称暗号化）*		✔	✔
MySQL Enterprise Firewall（SQLインジェクション対策）*		✔	✔
MySQL Enterprise Scalability (スレッドプール) *		✔	✔
MySQL Enterprise High Availability (HAサポート) *		✔	✔
Oracle Enterprise Manager for MySQL*		✔	✔
MySQL Cluster Manager (MySQL Cluster管理) *			✔
MySQL Cluster Geo-Replication			✔

*商用版のみで利用可能な追加機能

	MySQL Editions		
	Standard SE	Enterprise EE	Cluster CGE
Oracle Premium Support			
24時間365日サポート	✔	✔	✔
インシデント数無制限	✔	✔	✔
ナレッジベース	✔	✔	✔
バグ修正&パッチ提供	✔	✔	✔
コンサルティングサポート	✔	✔	✔
オラクル製品との動作保証			
Oracle Linux	✔	✔	✔
Oracle VM	✔	✔	✔
Oracle Solaris	✔	✔	✔
Oracle Enterprise Manager		✔	✔
Oracle GoldenGate		✔	✔
Oracle Data Integrator		✔	✔
Oracle Fusion Middleware		✔	✔
Oracle Secure Backup		✔	✔
Oracle Audit Vault and Database Firewall		✔	✔

※最新の対比表は、MySQL Editionsのサイトを参照下さい。

MySQL Supportの特徴

- 「パフォーマンス・チューニング」や「SQLチューニング」まで通常サポートの範囲内
  - コンサルティングサポートが含まれており、「クエリ・レビュー」、「パフォーマンス・チューニング」、「レプリケーション・レビュー」、「パーティショニング・レビュー」などに対応可能
    http://www-jp.mysql.com/support/consultative.html
- ソースコードレベルでサポート可能
  - ほとんどのサポートエンジニアがソースを読めるため、対応が早い
  - 開発エンジニアとサポートエンジニアも密に連携している
- 物理サーバー単位課金
  - CPU数、コア数に依存しない価格体系
- オラクルのライフタイムサポート
  - サポートポリシーが明確であるため、長期的な計画を立てやすい
    http://www-jp.mysql.com/support/

Appendix

統計情報のサンプリング方式

- デフォルト設定では、インデックスページを20ページサンプリングして、統計情報を見積る
  - システム変数 innodb_stats_persistent_sample_pages でサンプリングするページ数を変更可能
  - CREATE TABLE、ALTER TABLE時にSTATS_SAMPLE_PAGES句を使用して、テーブル毎にサンプリングするページ数を変更可能
    ⇒大規模なテーブルについて、統計情報の精度を向上したい場合など、任意でチューニング可能

統計情報を再計算するタイミング

- テーブルデータの10%を更新した時
  - 統計情報再計算の処理はバックグラウンドで非同期で行われる
- ANALYZE TABLE実行時
  - 明示的に任意のタイミングで統計情報を再計算できる

14.3.11.1 Configuring Persistent Optimizer Statistics Parameters
http://dev.mysql.com/doc/refman/5.7/en/innodb-persistent-stats.html
14.13.16.1 永続的オプティマイザ統計のパラメータの構成
http://dev.mysql.com/doc/refman/5.6/ja/innodb-persistent-stats.html

統計情報のサンプリング方式変更例

```
mysql> SELECT COUNT(*) FROM world.City;
+----------+
| COUNT(*) |
+----------+
|     4079 |
+----------+
1 row in set (0.00 sec)

mysql> select * from mysql.innodb_table_stats where table_name='City';
+---------------+------------+---------------------+--------+----------------------+--------------------------+
| database_name | table_name | last_update         | n_rows | clustered_index_size | sum_of_other_index_sizes |
+---------------+------------+---------------------+--------+----------------------+--------------------------+
| world         | City       | 2015-11-26 20:42:50 |   4188 |                   25 |                        8 |
+---------------+------------+---------------------+--------+----------------------+--------------------------+
1 row in set (0.00 sec)

mysql> select * from mysql.innodb_index_stats where table_name='City';
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
| database_name | table_name | index_name  | last_update         | stat_name    | stat_value | sample_size | stat_description                 |
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
| world         | City       | CountryCode | 2015-11-26 20:42:50 | n_diff_pfx01 |        232 |           7 | CountryCode                      |
| world         | City       | CountryCode | 2015-11-26 20:42:50 | n_diff_pfx02 |       4079 |           7 | CountryCode,ID                   |
| world         | City       | CountryCode | 2015-11-26 20:42:50 | n_leaf_pages |          7 |        NULL | Number of leaf pages in the index |
| world         | City       | CountryCode | 2015-11-26 20:42:50 | size         |          8 |        NULL | Number of pages in the index     |
| world         | City       | PRIMARY     | 2015-11-26 20:42:50 | n_diff_pfx01 |       4188 |          20 | ID                               |
| world         | City       | PRIMARY     | 2015-11-26 20:42:50 | n_leaf_pages |         24 |        NULL | Number of leaf pages in the index |
| world         | City       | PRIMARY     | 2015-11-26 20:42:50 | size         |         25 |        NULL | Number of pages in the index     |
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
7 rows in set (0.00 sec)
```

サンプリングの影響により、正確な値では無い

同様

統計情報のサンプリング方式変更例

```
mysql> ALTER TABLE world.City STATS_SAMPLE_PAGES=25;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ANALYZE TABLE world.City;
+------------+---------+----------+----------+
| Table      | Op      | Msg_type | Msg_text |
+------------+---------+----------+----------+
| world.City | analyze | status   | OK       |
+------------+---------+----------+----------+
1 row in set (0.01 sec)

mysql> select * from mysql.innodb_table_stats where table_name='City';
+---------------+------------+---------------------+--------+----------------------+--------------------------+
| database_name | table_name | last_update         | n_rows | clustered_index_size | sum_of_other_index_sizes |
+---------------+------------+---------------------+--------+----------------------+--------------------------+
| world         | City       | 2015-11-26 20:48:26 |   4079 |                   25 |                        8 |
+---------------+------------+---------------------+--------+----------------------+--------------------------+
1 row in set (0.00 sec)

mysql> select * from mysql.innodb_index_stats where table_name='City';
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
| database_name | table_name | index_name  | last_update         | stat_name    | stat_value | sample_size | stat_description                 |
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
| world         | City       | CountryCode | 2015-11-26 20:48:26 | n_diff_pfx01 |        232 |           7 | CountryCode                      |
| world         | City       | CountryCode | 2015-11-26 20:48:26 | n_diff_pfx02 |       4079 |           7 | CountryCode,ID                   |
| world         | City       | CountryCode | 2015-11-26 20:48:26 | n_leaf_pages |          7 |        NULL | Number of leaf pages in the index |
| world         | City       | CountryCode | 2015-11-26 20:48:26 | size         |          8 |        NULL | Number of pages in the index     |
| world         | City       | PRIMARY     | 2015-11-26 20:48:26 | n_diff_pfx01 |       4079 |          24 | ID                               |
| world         | City       | PRIMARY     | 2015-11-26 20:48:26 | n_leaf_pages |         24 |        NULL | Number of leaf pages in the index |
| world         | City       | PRIMARY     | 2015-11-26 20:48:26 | size         |         25 |        NULL | Number of pages in the index     |
+---------------+------------+-------------+---------------------+--------------+------------+-------------+----------------------------------+
7 rows in set (0.00 sec)
```

サンプリングページ数の増加

統計情報再計算

精度が向上している

同様

オプティマイザのコスト見積もりをチューニング可能

- MySQL 5.7からオプティマイザのコスト見積もりをチューニング可能
- 以下のテーブルに値を設定してチューニングする
  - mysql.server_cost
  - mysql.engine_cost
- 例）engine_cost テーブルの io_block_read_cost の cost_value を
  大きくすると、ディスクI/Oのコストがより大きく見積もられる

- 詳細情報
  8.9.5 The Optimizer Cost Model
  http://dev.mysql.com/doc/refman/5.7/en/cost-model.html

オプティマイザの動作の制御

- システム変数 optimizer_switch により、オプティマイザの動作を制御可能
- BKA(Batched Key Access)、BNL(Block Nested Loop)、MRR(Multi-Range Read)、などの最適化アルゴリズムの有効/無効を制御できる

- 詳細情報
  8.9.2 Controlling Switchable Optimizations
  http://dev.mysql.com/doc/refman/5.7/en/switchable-optimizations.html
  8.8.5.2 切り替え可能な最適化の制御
  http://dev.mysql.com/doc/refman/5.6/ja/switchable-optimizations.html

パーティショニング

- パーティショニングとは？
  - データを特定のカラムの値によって分割して管理できる機能
- 利点
  - データへのアクセスをパーティション単位に絞り込めることにより、レスポンスタイムが向上する可能性がある
  - データの運用管理性が上がる
    - 例）過去データを削除する時に、パーティション単位でTRUNCATE/DROPできる
- 欠点
  - 外部キーが使えないなど、一部制限事項がある

※制限事項の詳細は以下のマニュアルでご確認下さい
18.6 Restrictions and Limitations on Partitioning
https://dev.mysql.com/doc/refman/5.7/en/partitioning-limitations.html
19.6 パーティショニングの制約と制限
https://dev.mysql.com/doc/refman/5.6/ja/partitioning-limitations.html

パーティショニング

- パーティションのタイプは4つ
  - RANGE：カラム値の範囲を指定
  - LIST：任意のカラム値を指定（「東京」、「大阪」、など）
  - HASH：数値カラムのハッシュ値で分ける
  - KEY：文字列カラムのハッシュ値で分ける
- 2つのパーティショニングを組み合わせたサブパーティショニングや、複数列をパーティショニングキーに指定するCOLUMNSパーティショニングも利用できる

※パーティショニングタイプの詳細は以下のマニュアルでご確認下さい
18.2 Partitioning Types
https://dev.mysql.com/doc/refman/5.7/en/partitioning-types.html
19.2 パーティショニングタイプ
https://dev.mysql.com/doc/refman/5.6/ja/partitioning-types.html

その他の参考情報

参考情報

- MySQL Webサイト
  https://www-jp.mysql.com/
- MySQLコミュニティWebページ
  http://dev.mysql.com/
- 日本MySQLユーザー会(メーリングリスト有り)
  http://www.mysql.gr.jp/
- イベント案内
  - mysql.comのイベントページ
    https://www-jp.mysql.com/news-and-events/events/
  - オラクル社全体のイベントページ(OTN Japan - イベント・セミナー)
    http://events.oracle.com/search/search

MySQLのドキュメント

- MySQL Developer Zone(http://dev.mysql.com/)にドキュメント類が公開されている
- 以下のドキュメントは2015年6月に日本語版が公開された
  - MySQL 5.6 リファレンスマニュアル (含むMySQL Cluster 7.3-7.4マニュアル)
    http://dev.mysql.com/doc/refman/5.6/ja/index.html
  - MySQL Enterprise Monitor 3.0.18 マニュアル
    http://dev.mysql.com/doc/mysql-monitor/3.0/ja/index.html
  - MySQL Enterprise Backup ユーザーズガイド (バージョン 3.11.1)
    http://dev.mysql.com/doc/mysql-enterprise-backup/3.11/ja/index.html
- 上記日本語版公開以降に英語版ドキュメントのみ修正されている内容もあるため、ドキュメント参照時は英語版ドキュメントも合わせてご参照下さい。(URLの”ja”部分を”en”に変更すると、英語版ドキュメントが表示可能)

MySQLのドキュメント

- MySQL Documentation: MySQL Reference Manuals
  http://dev.mysql.com/doc/
- MySQL Documentation: MySQL Workbench
  http://dev.mysql.com/doc/index-gui.html
- MySQL Documentation: MySQL Utilities/MySQL Fabric
  http://dev.mysql.com/doc/index-utils-fabric.html
- MySQL Documentation: Connectors and APIs
  http://dev.mysql.com/doc/index-connectors.html

MySQLのドキュメント

- MySQL Documentation: Other MySQL Documentation
  http://dev.mysql.com/doc/index-other.html
  ⇒”world database”などのサンプルデータベースもダウンロード可能
- MySQL Documentation: MySQL Enterprise Products
  http://dev.mysql.com/doc/index-enterprise.html
  ⇒商用版製品に関するドキュメント

MySQL Enterprise Edition & Cluster CGEの試使用
30日間トライアル

- Oracle Software Delivery Cloud
  http://edelivery.oracle.com/
- 製品パックを選択:
  “MySQL Database”
- 製品マニュアル
  http://dev.mysql.com/doc/index-enterprise.html

オラクルユニバーシティ MySQL 研修

コース名	日数	価格 (税込)	開催日程
MySQL for Beginners	4	¥220,320	お問い合わせください
MySQL データベース管理 I	3	¥165,240	2016/01/18 - 20, 2016/02/17 - 19 2016/03/07 - 09
MySQL データベース管理 II	2	¥110,160	2016/01/21 - 22, 2016/02/22 - 23, 2016/03/14 - 15
MySQL High Availability	3	¥231,336	お問い合わせください

※ MySQL データーベース管理I/IIはMySQL5.5対応、MySQL 入門は MySQL 5.0/5.1対応です。
※ コース開催予定は2015年11月現在のものです。開催日程の最新情報はOracle University ホームページ (http://www.oracle.com/jp/education/) にてご確認ください。
※ 価格(税込み)は**2015年11月現在の価格です**。Oracle PartnerNetwork 会員様は、パートナー割引価格で受講いただけます。

管理者向け MySQL 5.6 対応認定資格

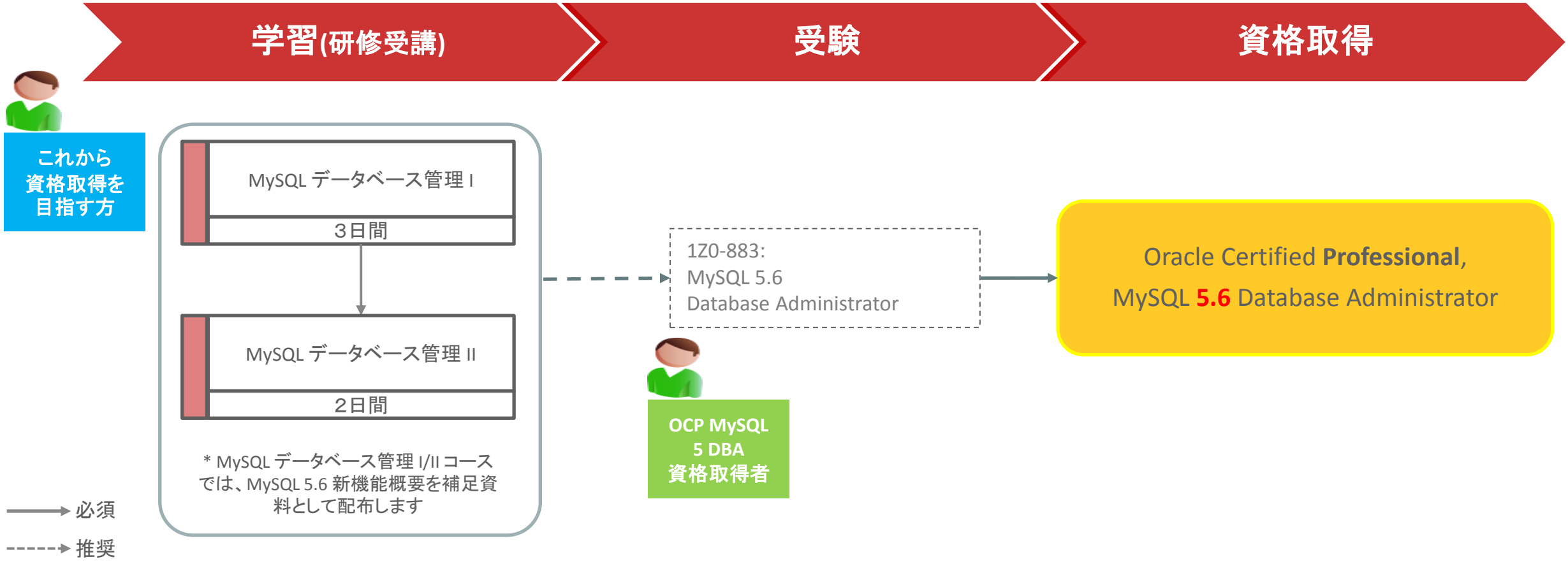
- Oracle Certified **Professional**, MySQL **5.6** Database Administrator
  - Oracle Certified Professional, MySQL 5.6 Database Administrator 資格は、パーティショニング、およびレプリケーションにおける機能強化やパフォーマンス監視と診断のPERFORMANCE_SCHEMAの使用などMySQL 5.6の新機能を含むMySQLデータベースのインストール、複製、チューニング、およびセキュリティ設定など幅広い管理スキルを証明します。
- 認定試験:
  - MySQL 5.6 Database Administrator (1Z0-883)
    - 本試験に合格することで、資格取得できます
    - 日本語試験、英語試験共に受験可能

開発者向け MySQL 5.6 対応認定資格

- Oracle Certified **Professional**, MySQL **5.6** Developer
  - Oracle Certified Professional, MySQL 5.6 Database Administrator 資格は、MySQL データ・タイプや SQLシンタックス、テーブルやスキーマなどの各種オブジェクト、ストアド・プロシージャ、ビュー、結合など、MySQL データベースを使用したアプリケーション開発に必要なスキルを証明します。
- 認定試験:
  - MySQL 5.6 Developer (1Z0-882)
    - 本試験に合格することで、資格取得できます
    - 日本語試験、英語試験共に受験可能

管理者向け MySQL 5.6 認定資格取得パス

新規取得もアップグレードも一試験で。



Integrated Cloud

Applications & Platform Services


ORACLE®