WiXEngineering

MySQL Database Architecture & Administration Fundamentals

baruchao@wix.com twitter@baruchoso linkedin/baruch-assif-osoveskiy github.com/barucho

About Me: Baruch Assif (osoveskiy)

- Senior dba @ wix.com
- Linux/UNIX SYSADMIN (from 1999)
- DBA – Oracle, MySQL, MongoDB, Elasticsearch, Cassandra
- Working with MySQL since 2000.
- Talk about databases since 2012 (john bryce,oracle week ,geek week ,MySQL user Groups…).

@baruchoso github.com/barucho

Recommended blogs/sites :

http://datacharmer.blogspot.co.il/

http://mysqlsandbox.net/index.html

http://dom.as/

https://www.percona.com/blog/

https://code.facebook.com/

http://dev.mysql.com/

https://lefred.be

Code will be in github :

https://github.com/barucho/ow2018

You can send me interesting bugs @:

baruchao@wix.com

Find more presentations:

http://www.slideshare.net/baruchosoveskiy

My blog

https://barucho.github.io/blog/

At the end of each section will be Q&A slide you can ask question I will try to answer

AGENDA

- “MySQL in the real world”- How WIX is Using MySQL
- MySQL a Quick Installation Guide
- MySQL Architecture
- Storage Engines- InnoDB and MyRocks
- MySQL Parameter File and parameters recommendations
- MySQL Information Schema
- MySQL Performance Schema
- Quick intro to MySQL 8.0 New Features and JSON store

I Will Not Talk About

MySQL < 5.6 – please upgrade.

MyISAM – Just don't.

Query cache - The query cache is deprecated as of MySQL 5.7.20, and is removed in MySQL 8.0.

Use only query_cache_type=2 or query_cache_type=0

Free text search (< 8.0) – use Elasticsearch

Exotics Engines – don't have time for this.

Cloud\RDS\Aurora – you get what you pay for.

MySQL Cluster (NDB) - not like InnoDB not relevant to most case usage.

For This Talk

We can compare

MariaDB 10 ~= Percona server 5.6 ~= Facebook 5.6 ~= MySQL 5.6

Percona 5.7 ~= MySQL 5.7 != MariaDB 10.x

https://www.percona.com/blog/2017/11/02/mysql-vs-mariadb-reality-check/

We are in Oracle week so .. I will run most of the Demos in (Oracle) MySQL 8.0.12

For my talk Oracle MySQL Enterprise = MySQL Community

WiXEngineering

Q&A

Install_MySQL.md

WiXEngineering

Q&A

MySQL introduction.

MySQL

But MySQL its not like oracle or sql-server.

But its good for small data.

Its only for Linux, I don’t know Linux..

But we need Real database/NoSQL

You need to pay Oracle.

I need enterprise database.

MySQL don’t have RAC.

I need to run in the cloud.

Its not mongo.

MySQL

MySQL is the world's most popular open source database

Leading database choice for web-based applications

Used by high profile web properties including Facebook, Twitter, YouTube, Alibaba, Booking.com, LinkedIn etc..

MySQL is under dual license GPLv2 and FOSS

Have many forks like Oracle MySQL, Percona , Mariabd, WebScale.org ,aurora,facebook 5.6 etc.…

Active Active Cluster – (Innodb cluster)

Most popular databases in 2018 according to StackOverflow survey

https://db-engines.com/en/ranking_trend

MySQL Version History

5.0 stored procedure, view, triggers, query optimizer

5.1 NDB, record replication InnoDB plugin default install

5.5 Oracle First Version Multi process support,

~300% performance improvement

5.6 and 7.4 cluster(NDB) (can be installed in windows)

5.7 GA Version

8.0.2 Current GA Version **

** Production recommendation is 5.7

MySQL 3.19
MySQL 3.20
MySQL 3.21
MySQL 3.22
MySQL 3.23
MySQL 4.0
MySQL 4.1
MySQL 5.0
MySQL 5.1
Percona Server 5.0
MySQL Cluster 6.2
MySQL 6.0
MySQL 5.2
Percona Server 5.1
MySQL Cluster 6.3
Drizzle 7.0
MariaDB 5.1
MySQL Cluster 7.0
Drizzle 7.1
MySQL 5.4
MariaDB 5.2
MySQL Cluster 7.1
Drizzle 7.2
MySQL 5.5
MariaDB 5.3
MySQL Cluster 7.2
MySQL 5.6
Percona Server 5.5
MySQL Cluster 7.3
MySQL 5.7
Percona Server 5.6
MariaDB 5.5
MySQL Cluster 7.4
Percona Server 5.7
MariaDB 10.0
MySQL Cluster 7.5
MySQL 8.0
MariaDB 10.1
MySQL Cluster 7.6
Percona Server 8.0
MariaDB 10.2
MariaDB 10.3

Who uses MySQL?

Insert your logo here

Mission Critical Flight Planning Applications at the US Navy Run on MySQL

ARCHITECTURE

MySQL Architecture

MySQL client/server model

Client Programs

- Connect to the MySQL server to retrieve, modify, add, or remove data.
- Use these client programs to perform the following actions:

  mysql: Issue queries and view results.

  mysqladmin: Administer the server.

  mysqlcheck: Check the integrity of database tables.

  mysqldump: Create logical backups.

  mysqlimport: Import text data files.

  mysqlshow: Show database, table, and column information.

  mysqlslap: Emulate client load.

- Use MySQL Workbench for database management.

MySQL Server

The database server program called mysqld

Not the same as a “host”

Single process and is multithreaded

Manages access to databases on disk and in memory

Supports simultaneous client connections

Supports multiple storage engines

Supports both transactional and nontransactional tables

Uses memory in the form of:

Caching

Buffering

Server Process

Connection Layer

Communication Protocols

Protocol	Types of Connections	Supported Operating Systems
TCP/IP	Local, remote	All
UNIX socket file	Local only	UNIX only
Shared memory	Local only	Windows only
Named pipes	Local only	Windows only

Protocols are implemented in the client libraries and drivers.

The speed of a connection protocol varies with the local settings.

SQL Layer

Application
Application
Application
mysqld
Connection layer
SQL layer
• Parser
• Authorization
• Optimizer
• Query execution
• Query cache
• Query logging
Storage layer

SQL Statement Processing

MySQL CBO Optimizer 5.7+

Storage Layer

Storage Engine: Overview

Storage engines are server components that act as handlers for different table types and Stroger Types.

Storage engines are used to:

- Store data
- Retrieve data
- Find data through an index

Storage Engine: Overview

Two-tier processing

Upper tier includes SQL parser and optimizer.

Lower tier consists of a set of storage engines.

SQL tier is not dependent on the storage engine:

The engine does not affect SQL processing.

*There are some exceptions.

Features Dependent on Storage Engine

Storage medium

Transactional capabilities

Locking

Backup and recovery

Optimization

Special features

Full-text search

Referential integrity

Spatial data handling

How MySQL Uses Disk Space

How MySQL Uses Memory

- Global
  - Allocated once
  - Shared by the server process and its threads
- Session
  - Allocated for each thread
  - Dynamically allocated and deallocated
  - Used for handling query results
  - Buffer sizes per session

Memory Structures

Server allocates memory in three different categories:

Server/Shared

- Query cache
- Thread cache

Storage Engine/ Shared

- Log buffer
- Buffer pool

Connection/ Session

- Sort buffer
- Read buffer
- Temporary table

Connection/Session

- Internal temporary tables: In some cases of query execution, MySQL creates a temporary table to resolve the query. The temporary table can be created in memory or on disk depending on its size or contents or on the query syntax.
- Client-specific buffers: Are specifically designed to support the individual clients that are connected. Examples of the buffers include:
- Communications buffer for exchanging information
- Table read buffers (including buffers that support joins)
- Sort operations

MySQL Plugin Interface

- Daemon plugins, run by the server
- Plugin API allows loading and unloading of server components.

  Supports dynamic loading, without restarting server

  Supports full-text parser

  Supports different authentication methods

  Supports storage engines installed as plugins

  Supports INFORMATION_SCHEMA plugins
- Requires the PLUGINS table in the mysql database
- MySQL supports both client and server plugins.

MySQL Server Parameters

WiXEngineering

Q&A

Hardware

OS

OS

- Linux (all) best, Windows also fine (5.6+).
- ext4 (or xfs), mount with noatime
- Scheduler – use deadline or noop

  add "elevator=deadline" to grub.conf
- SWAP: swappiness => 1 (keep a look on omm-kiler)

  add "vm.swappiness = 1" to /etc/sysctl.conf
- NUMA : numactl --interleave=all
- CPU no powersave mode enabled
- Check Open file limit ulimit –a

Hardware

- 4-8 Cores (5.1 can use 1), 8-16 cores for 5.5 and up
- Linux or ~~Solaris~~ best, Windows also fine (5.6+).
- RAID 1+0 for most, RAID 5 OK if very read intensive
- Hardware RAID battery backed up cache critical!
- More disks are always better!

  4+ recommended, 8-16 can increase IO performance if needed
- Memory should be at least 1/3 data size (Live Data)

Linux Filesystem Recommendations

- Avoid using ext2 or ext3.
- Mount ext4 with the following options:
  - rw,noatime,nodiratime,nobarrier,data=ordered
- When using XFS or ZFS file systems:
  - Turn off atime (noatime) for the partition.
    - On XFS, also turn off diratime (nodiratime).
  - Consider setting numactl –interleave all.
- Where possible, set swappiness to a value between 1 and 5, and set the deadline I/O scheduler.
- Use SWAP on linux > 7.x (oom_kill.c)
- Consider disabling the InnoDB doublewrite buffer on any filesystem that supports atomic writes:
  - Such as ZFS, BTRFS, DIRECTFS, and ext4 with data=journal option

WiXEngineering

Q&A

Option Files

Option Files

- Most MySQL programs can read startup options from option files (sometimes called configuration files).
- Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program.
- Option files can be included and chained
- The last value of the option name will be applied

Option Files

Table 4.2 Option Files Read on Unix and Unix-Like Systems

File Name	Purpose
`/etc/my.cnf`	Global options
`/etc/mysql/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options (server only)
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User-specific options
`~/.mylogin.cnf`	User-specific login path options (clients only)

Table 4.1 Option Files Read on Windows Systems

File Name	Purpose
`%PROGRAMDATA%\MySQL\MySQL Server 5.7\my.ini`, `%PROGRAMDATA%\MySQL\MySQL Server 5.7\my.cnf`	Global options
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`BASEDIR``\my.ini`, *`BASEDIR`*`\my.cnf`	Global options
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`%APPDATA%\MySQL\.mylogin.cnf`	Login path options (clients only)

Option Files

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- `#comment`, `;comment`

  Comment lines start with `#` or `;`. A `#` comment can start in the middle of a line as well.

- `[group]`

  group is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case sensitive.

- *opt_name*

  This is equivalent to `--opt_name` on the command line.

- *opt_name*=*value*

  This is equivalent to `--opt_name=value` on the command line. In an option file, you can have spaces around the = character, something that is not true on the command line. The value optionally can be enclosed within single quotation marks or double quotation marks, which is useful if the value contains a `#` comment character.

Option Files

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

Option Files

- Option files can be chain by include file or directory

```
1  !include /home/mydir/myopt.cnf
```

```
1  !includedir /home/mydir
```

WiXEngineering

Q&A

TOOLS

MySQL Performance Monitoring Tools

- Use tools to **monitor** the MySQL server
- Use **benchmarks** tools in the MySQL server environment
- Use **workload** tools that stresses the MySQL server

MySQL Performance Monitoring Tools

- Manual - SHOW STATUS, MYSQLADMIN, SHOW OPEN TABLES
- MySQL Enterprise Monitor
- **Workbanch**
- information_schema , **performance_schema**
- **SYS schema**
- **Explain**
- Slow Query Log
- DBTuna,nagios ,MONyog ,Centra etc…
- Innotop,Mytop etc…
- MySQLTuner-perl*
- pt-query-digest*

Information Schema

- Is an online data dictionary implemented as a database schema: `INFORMATION_SCHEMA`
- Holds the information in virtual tables
- Can be queried using SQL
  - Use prepared statements, stored procedures, and so on
  - Control how the information is displayed
- Contains detailed information about InnoDB:
  - Includes buffer pool, transactions, and locking details
  - Supports enabling/disabling each metric individually
  - Is easier to access than `SHOW ENGINE INNODB STATUS`

MySQL Monitoring Tools

- `mysql` client commands:
  - `SHOW [SESSION | GLOBAL] STATUS`
  - `SHOW ENGINE INNODB STATUS`
  - `SHOW PROCESSLIST`
- The `mysqladmin` client program
- MySQL system databases:
  - Information Schema
  - Performance Schema
- MySQL GUI Tools:
  - MySQL Workbench
  - MySQL Enterprise Monitor

SHOW [SESSION | GLOBAL] STATUS

- Lists server status variables in key/value pairs
  - Filter using `LIKE` and `WHERE`
- `SHOW STATUS` or `SHOW SESSION STATUS` displays only status values for the current connection
- `SHOW GLOBAL STATUS` displays status values for the instance as a whole.
- Values are either counters or metrics
  - Counters increment when the server completes an action:
    - For example, `Select_scan` increments for every full table scan
    - They are reset when their maximum value is reached
  - Metrics can increase and decrease
    - For example, `Connections` shows number of open connections
- Reset session status variables with `FLUSH STATUS`

Filtering `SHOW STATUS` Output

- Handler operations
  - `SHOW GLOBAL STATUS LIKE 'Handler_%';`
- Temporary tables and files
  - `SHOW GLOBAL STATUS LIKE 'Created_tmp%';`
- Command counters
  - `SHOW GLOBAL STATUS LIKE 'Com_%';`
- Thread counters
  - `SHOW GLOBAL STATUS LIKE 'Threads_%';`
- Other metrics include:
  - `Uptime`: the server's uptime, in seconds
  - `Binlog_*`: binary log status
  - `Connections`: the number of connection attempts
  - `Questions`: the number of queries sent to the server

Filtering `SHOW STATUS` Output (Notes Only)

SHOW STATUS Example

SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);

```
mysql> SHOW STATUS LIKE 'Created_tmp%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
...
| Created_tmp_tables      | 2     |
+-------------------------+-------+
3 rows in set (0.00 sec)
```

SELECT Name, Population FROM Country WHERE code IN (SELECT countrycode FROM CountryLanguage WHERE Percentage > 50);

```
mysql> SHOW GLOBAL STATUS LIKE 'Handler_read%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 5     |
...
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 1256  |
+-----------------------+-------+
7 rows in set (0.00 sec)
```

This value is high if you are doing a lot of table scans.

SHOW ENGINE INNODB STATUS

The `SHOW ENGINE INNODB STATUS` command reports activity within InnoDB.

- Contains lots of detailed information, including:
  - Background thread
  - Semaphores
  - Foreign key errors
  - Deadlocks
  - Transactions
  - File I/O
  - Insert buffer and adaptive hash index
  - Log
  - Buffer pool and memory
  - Row operations

SHOW PROCESSLIST

- Use the `SHOW PROCESSLIST` command to display information about currently running queries, including:
  - How long the query has been running
  - The user who executed the query
  - The hostname or IP address where the query originated
  - The current state of the query
- `SHOW PROCESSLIST` helps you to determine if:
  - A query is taking too long to complete
  - You have too many open queries
  - A client has issued a `LOCK TABLE`
- The output of `SHOW PROCESSLIST` shows only the first 100 characters of the query text by default.
  - Use `SHOW FULL PROCESSLIST` to display the full query

MySQL Enterprise Monitor: Overview

MySQL Enterprise Monitor provides real-time visibility into the performance and availability of all your MySQL databases.

- Enables real-time, continuous performance monitoring
  - Generates alerts when performance differs significantly from baseline values
  - Best practice advisors recommend changes to improve performance.
- Manages multiple MySQL servers using a visual interface
  - Remotely monitors cloud servers without installing agents
- Enables visual query analysis
  - Monitors query performance in real time
  - Identifies and fixes problem queries
- Monitors key InnoDB metrics that affect performance

MySQL Enterprise Monitor: Dashboard

Firefox | Overview : MySQL Enterprise Dashboard

https://mylab:18443/Monitor.action?_x=x&assetSelection=[{"assetClass"%3A"com.mysql.etools.monitor.ServerGroup"%2C"id"%3A"2147483646"}%2C{"assetClass"%3A"com.mysql.etools.inventory.model.mysql.Mys

ORACLE MySQL Enterprise Monitor

1 4 0 0 0 manager

Dashboards | Events | Query Analyzer | Reports & Graphs | Configuration | Refresh: Off

Group Overview: Production Replication

Database Statistics

Database Availability

Day	100%
Week	100%
Month	100%

Connections - All MySQL Instances

Connections 10 5 0 10:45 11:00 11:15 11:30

Total (SUM) Running (SUM)

Database Activity - All MySQL Instances

Statements ... 100 50 0 10:45 11:00 11:15 11:30

Select (SUM) Insert (SUM) Update (SUM) Replace (SUM) Delete (SUM) Call (SUM)

Query Response Time Index

QR Ti 1 0.50 0 optimal level acceptable level unacceptable level 10:45 11:00 11:15 11:30

qrti

Current Problem MySQL Instances

Show / hide columns

ID	Status	Emergency	Critical	Warning
mylab.localdomain:3307	Up	0	4	15
mylab.localdomain:3306	Up	0	3	14

Showing 1 to 2 of 2 entries

Current Problem Hosts

Show / hide columns

ID	Status	Emergency	Critical	Warning
mylab.localdomain	Up	0	1	1

Showing 1 to 1 of 1 entries

Emergency & Critical Events

Show All entries | Show / hide columns | First Previous 1 Next Last

Subject	Topic	Time	Actions
mylab.localdomain, / (/dev/mappe...	Filesystem / (/dev/mapper/vg_...	about a minute ago	
mylab.localdomain, mylab.locald...	Root Account Without Password	4 minutes ago	
mylab.localdomain, mylab.locald...	Root Account Without Password	4 minutes ago	
mylab.localdomain, mylab.locald...	Root Account Without Password	4 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Accounts Without ...	9 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Anonymous Accou...	9 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Accounts Without ...	4 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Anonymous Accou...	9 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Accounts Without ...	4 minutes ago	
mylab.localdomain, mylab.locald...	Server Has Anonymous Accou...	9 minutes ago	
mylab.localdomain, mylab.locald...	Slave Too Far Behind Master	2 minutes ago	

Showing 1 to 11 of 11 entries | First Previous 1 Next Last

Copyright © 2005, 2013, Oracle and/or its affiliates. All rights reserved.

3.0.0.2886 - mylab.localdomain (192.168.1.10) - Aug 27, 2013 11:36:01 am (Up Since: 1 hour, 24 minutes ago) - About

WiXEngineering

Q&A

Memory Buffers

Global Buffers

Global buffers are allocated memory during the server startup process and hold onto this memory until the server process ends. Memory is allocated regardless of the number of connections.

Examples:

- `query_cache_size`: The amount of memory available for caching the results of previous queries (`query_cache_type=2`)
- `table_open_cache`: The number of open tables for **all threads**
- `innodb_buffer_pool_size`: The amount of memory available for caching **InnoDB data and indexes**
- `innodb_log_buffer_size:` The size of the buffer that **InnoDB** uses to **write to the log files on disk**

MySQL Thread Handling

- MySQL is a multithreaded server.
  - A single process, comprised of multiple threads
- Each connection has its own thread.
  - 1,000 connections require 1,000 threads.
- Additional helper threads include:
  - Connection manager threads
  - Signal threads (alarms)
  - InnoDB read and write threads
  - InnoDB purge threads
  - Event scheduler
- Thread caching allows unused threads to remain in memory for reuse.

Per-Thread Buffers

- **`sort_buffer_size`**
  - The amount of memory available for buffering queries that use `ORDER BY` or `GROUP BY` options.
  - many Sort_merge_passes increase the sort_buffer_size
- **`read_rnd_buffer_size`**
  - The amount of memory available for InnoDB table queries doing **range scans on secondary indexes**.
- **`join_buffer_size`**
  - The **minimum size** of the buffer that is used for plain index scans, range index scans, **and joins that do not use indexes** and perform full table scans as a result.
  - keep the value of join_buffer_size small and only increase it in sessions that are performing large joins
  - allocated once for every two tables involved in the join
- **`binlog_cache_size` / `binlog_stmt_cache_size`**
  - The size of the caches that store changes to the binary log during a transaction.

Per-Thread Buffers

- **`tmp_table_size`**
  - Determines the **size of internal memory** tables that can be stored, in conjunction with **`max_heap_table_size`**.
  - Associated status variables:
    - **`Created_tmp_disk_tables`**
    - **`Created_tmp_tables`**
- **`thread_stack`**
  - Determines the size of the stack assigned to each thread:
    - If the thread stack size is too small, it limits the complexity of the SQL operations the thread can handle.
    - Default value is 256KB for 64-bit systems.
    - Do not reduce the size of the thread_stack

Calculating Maximum MySQL Memory Usage

Maximum global buffer memory usage is the sum of the global buffers and Perfomance Schema memory usage:

```
query_cache +
table_open cache +
innodb_buffer_pool_size +
innodb_log_file_size +
performance_schema.memory
```

Total Maximum Memory Usage = Max global buffer memory + Max thread buffer memory

Maximum thread buffer memory usage is the sum of the per-thread buffers multiplied by the maximum number of connections:

`max_connections` *

(`sort_buffer_size` +
`read_rnd_buffer_size` +
`join_buffer_size` +
`binlog_cache_size` +
`thread_stack` +
[the smaller of `tmp_table_size` and `max_heap_table_size`])

Scenario: Total MySQL Memory Usage

Estimate the maximum memory requirements for a server with the following per-thread buffer settings. Are these settings optimal for a server with 4 GB of memory? Do we swap or not ?

```
join_buffer_size = 512K
read_rnd_buffer_size = 2M
sort_buffer_size = 2M
thread_stack = 512K
binlog_cache_size = 256K
max_connections = 300
max_heap_table_size = 24M
tmp_table_size = 32M
```

WiXEngineering

Q&A

STORAGE ENGINES

Choosing the Right Storage Engine

- many storage engines supported by MySQL Some of them are open source some are not.
- Oracle will not support all of the storage engines
- **XtraDB/InnoDB** are the best choice for most case usage

Choosing the Right Storage Engine

General purpose :

- **XtraDB/InnoDB** is the best choice in the majority of cases.
- **Aria**, MariaDB's more modern improvement on **MyISAM**, has a small footprint and allows for easy copying between systems.
- **MyISAM** has a small footprint and allows for easy copying between systems. MyISAM is MySQL's oldest storage engine.
- **Facebook MyRocks** A space and write optimized

Choosing the Right Storage Engine

Scaling, Partitioning ,High Load

- **Galera** - a synchronous multi-master cluster.
- **NDB** – Cluser engine provide sharding and high avlibality
- **TokuDB** is a transactional storage engine which is optimized for workloads that do not fit in memory, and provides a good compression ratio.
- **Spider** uses partitioning to provide data sharding through multiple servers.

specialized storage engines

- **SphinxSE** is used as a proxy to run statements on a remote Sphinx database server (mainly useful for advanced fulltext searches).
- **Sequence** allows the creation of ascending or descending sequences of numbers (positive integers) with a given starting value, ending value and increment, creating virtual, ephemeral tables automatically when you need them.
- **BLACKHOLE** storage engine accepts data but does not store it and always returns an empty result. This can be useful in replication environments, for example, if you want to run complex filtering rules on a slave without incurring any overhead on a master.
- **OQGRAPH** allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).

INNODB

InnoDB

- Transactional and fully ACID compliant Behavior most like traditional databases such as Oracle, DB2, SQL Server, etc.
- MVCC = Non-blocking reads in most cases
- Fast, reliable recovery from crashes with zero
- **Always** clustered on the primary key

– Lookups by primary key, very fast

– Range scans on primary key also very fast

– Non-Primary key lookups use the primary key to find the record, this means 2 key lookups

– Important to keep primary key small

http://dev.mysql.com/doc/refman/5.5/en/innodb.html

InnoDB

Unlike MyISAM InnoDB uses a single cache for both index and data

– **Innodb_buffer_pool_size** - should be 70-80% of available memory.

– It is not uncommon for this to be very large, i.e. 40GB on a system with 44GB of memory

– Make sure its not set so large as to cause swapping!

mysql>show status like 'Innodb_buffer%' ;

- InnoDB can use direct IO on systems that support it, Linux, FreeBSD, Solaris.

– **Innodb_flush_method** = O_DIRECT

Cache hot application data in memory

DBT-2 (W200)	Transactions per Minute	%user	%iowait
Buffer pool 1G	1125.44	2%	30%
Buffer pool 2G	1863.19	3%	28%
Buffer pool 5G	4385.18	5.5%	33%
Buffer pool 30G (All data in cache)	36784.76	36%	8%

- DBT-2 benchmark (write intensive)
- 20-25GB hot data (200 warehouses, running 1 hour)
- Nehalem 2.93GHz x 8 cores, MySQL 5.5.2, 4 RAID1+0 HDDs
- RAM size affects everything. Not only for SELECT, but also for INSERT/UPDATE/DELETE
  - INSERT: Random reads/writes happen when inserting into indexes in random order
  - UPDATE/DELETE: Random reads/writes happen when modifying records

Copyright 2010 Sun Microsystems inc

InnoDB Buffers and Logs: Reading Data

```
SELECT * FROM world.Country
WHERE NAME = 'China'
```

InnoDB Buffers and Logs: Writing Data

```
UPDATE world.Country SET HeadOfState = 'Xi Jinping'
WHERE NAME = 'China'
```

Dumping and Restoring the Buffer Pool

- Large buffer pools take a long time to warm up.
  - Your applications must recreate the buffer pool by issuing queries, updates, and so on.
- Speed up server restarts by saving and then restoring the buffer pool.
  - MySQL has to only copy disk pages into memory.
  - It does not delay database startup.
- MySQL configuration file options:
  - Saving the buffer pool state at shutdown:
  - Restoring the buffer pool at startup:

```
innodb_buffer_pool_dump_at_shutdown=ON;
```

```
innodb_buffer_pool_load_at_startup=ON;
```

Sizing the InnoDB Transaction Log Files

- The InnoDB transaction log files must be large enough to ensure that writes are fast and durable.
  - Larger log files reduce checkpoint flushes and disk I/O.
- If the transaction logs are too large, crash recovery can take a long time.
- The transaction logs should be large enough to store a maximum of one hour's worth of transactions.
  - Enough time for MySQL to optimize flushing
  - Reasonable crash recovery time
- The `innodb_log_file_size` server variable specifies the size (in bytes) of each log file in a *log group*.
  - `innodb_log_files_in_group` specifies the number of log files in a log group (default is `2`).

Other InnoDB Transaction Log Settings

- `innodb_flush_log_at_trx_commit` defines how often the log buffer is written out to the log file and when the flush to disk operation takes place.
  - `0` writes the log buffer to the log file and flushes the log file to disk every second (approximately).
  - **`1` writes and flushes the log file at transaction commit.**
  - **`2` writes the log buffer to the log file after each commit and flushes the file to disk every second (approximately).**
- `innodb_log_buffer_size` sets the size (in bytes) of the buffer that InnoDB uses to write to the log files on disk.

Thread Concurrency

- `innodb_thread_concurrency` defines the number of operating system threads that are allowed to run concurrently in InnoDB:
  - `0` is interpreted as infinite concurrency (no concurrency checking).
  - **Most workloads run well without limiting the number of concurrent threads**
- If a limit is used, adjust the value of `innodb_concurrency_tickets` from its default value of `5000` so that 99 percent of queries run with one allocation.
- Check system (not user) CPU usage.
  - If overall CPU usage is very high, this can be a sign of too many threads running (100+) and causing mutex contention in InnoDB. Limit thread concurrency, or use a thread pool.

InnoDB Buffer and Log File Flushing

- `innodb_flush_method` specifies the operating system call used to flush both the data and log files.
- On Linux systems:
  - `fdatasync` uses `fsync()` and is the default setting.
  - `O_DSYNC` uses `O_SYNC` to open and flush the log files and `fsync()` to flush the data files (usually slow).
  - `O_DIRECT` uses `O_DIRECT` to open and flush the log files and datafiles and `fsync()` to flush the data files.
- On Windows systems, the only available option is `async_unbuffered`.
- InnoDB uses *doublewrite* buffering by default.
  - To turn off doublewrite buffering, set `innodb_doublewrite=0`.

Use Direct IO (O_DIRECT)

- Direct I/O is important to fully utilize Memory (In linux)
- innodb_flush_method=O_DIRECT
- Alignment: File i/o unit must be a factor of 512 bytes
- **Can’t use O_DIRECT for InnoDB Log File, Binary Log File, MyISAM (don’t use MyISAM)
- PostgreSQL data files, etc

Adaptive Flushing

With adaptive flushing, InnoDB estimates the required rate of flushing, based on the speed of transaction log generation and the current rate of flushing.

- When InnoDB tries to flush everything from the buffer pool at the same time, this is known as “furious flushing”
  - Severely degrades performance
  - During furious flushing, `SHOW ENGINE INNODB STATUS` shows `main thread state: flushing`.
    - This status does not always indicate furious flushing.
- Adapting flushing:
  - Flushes small sets of modified pages from the buffer pool instead of all at the same time.
  - Used when `innodb_adaptive_flushing` is enabled (`ON` by default)

Purge Behavior

- `innodb_purge_threads` defines the number of background threads devo to the InnoDB purge operation.
- `innodb_io_capacity` defines the maximum number of I/O operations pe second that InnoDB background operations perform.
- `innodb_io_capacity_max` is the limit up to which InnoDB can extend `innodb_io_capacity` in an emergency.

Undo Logs

- `innodb_undo_directory` enables you to store undo logs outside the system tablespace.
- This improves performance:
  - Reduces the growth of the system tablespace
  - Enables parallel I/O if you place undo logs on separate fast storage
- Related settings are `innodb_undo_logs` and `innodb_undo_tablespaces`.

Recommended InnoDB Settings for OLTP and Benchmarking

- Include the following in your `my.cnf` or `my.ini` file:

```
innodb_read_io_threads = 16
innodb_write_io_threads = 4
table_open_cache_instances = 16
```

- Adjust the following where appropriate:
  - `innodb_log_file_size`
  - `innodb_flush_neighbors`
  - `innodb_page_size`
  - `innodb_io_capacity` and `innodb_io_capacity_max`
  - `innodb_checksum_algorithm (CRC)`
  - `innodb_logged_compressed_pages`
  - `binlog_row_image`
  - `innodb_stats_persistent`

Recommended InnoDB Settings for OLTP and Benchmarking

- **`innodb_stats_persistent`**

By default, MySQL stores InnoDB index statistics created by **`ANALYZE TABLE`** on disk, only replacing them when `ANALYZE TABLE` is executed again.

If you disable `innodb_stats_persistent`, MySQL recalculates index statistics at other times, such as on server restart.

Recommended InnoDB Settings for Replication

- Single/Master server:
  - `innodb_flush_log_at_trx_commit = 1`
- Slave, server with no binlog, unsafe with binlog:
  - `innodb_flush_log_at_trx_commit = 0`
    - Can be set to safer setting if desired
  - binlog off
- Both master and slave:
  - Set `innodb_log_file_size` as large as possible.
    - Be aware that there is a minimum of two log files, and that this is the default setting.
  - `innodb_purge_threads = 1`

Linux Filesystem Recommendations

- Avoid using `ext2` or `ext3`.
- Mount `ext4` with the following options:
  - `rw,noatime,nodiratime,nobarrier,data=ordered`
- When using `XFS` or `ZFS` file systems:
  - Turn off `atime` (`noatime`) for the partition.
    - On XFS, also turn off `diratime` (`nodiratime`).
  - Consider setting `numactl -interleave all`.
- Where possible, set swappiness to a value between 1 and 5, and set the deadline I/O scheduler.
- Use SWAP on linux > 7.x (oom_kill.c)
- Consider disabling the InnoDB doublewrite buffer on any filesystem that supports atomic writes:
  - Such as `ZFS`, `BTRFS`, `DIRECTFS`, and `ext4` with `data=journal` option

InnoDB Support in `INFORMATION_SCHEMA`

The following `INFORMATION_SCHEMA` tables provide valuable information about InnoDB status:

- `INNODB_CMP` and `INNODB_CMP_RESET`
  - Contains status information about the operations related to compressed tables
- `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET`
  - Contains status information about the compressed pages that reside in the buffer pool
- `INNODB_TRX`
  - Contains information about every transaction currently executing inside InnoDB

InnoDB Support in `INFORMATION_SCHEMA`

- `INNODB_LOCKS` and `INNODB_LOCK_WAITS`
  - Contains information about the locks that are blocking any transactions
- `INNODB_BUFFER_POOL_STATS`
  - Contains similar information to the buffer pool information shown in `SHOW ENG INNODB STATUS`
  - Available in graphical format in MySQL Enterprise Monitor reports:

  - Contains information about each page in the buffer pool

Information Schema InnoDB Metrics

- Use the `INFORMATION_SCHEMA.INNODB_METRICS` table to monitor InnoDB metrics of interest.
- The `NAME` and `COUNT` columns give the monitor name and its value.
- Other columns provide extra information:
  - Status (running or stopped)
  - Maximum, minimum, and average values since monitoring began and when the counters were last reset
  - Type (whether value is incremental or a resource counter)
  - Timing information
- Monitors are grouped into modules.
- Control monitoring by setting option variables.

InnoDB Support in `performance_schema`

- The Performance Schema includes instrumentation for monitoring and troubleshooting InnoDB file I/O issues:
  - `/wait/io/file/innodb/innodb_data_file`: File I/O for central or single tablespace data files
  - `/wait/io/file/innodb/innodb_log_file`: File I/O for the redo logs
  - `/wait/io/file/innodb/innodb_temp_file`: File I/O against a sort merge file that is used when adding indexes
- The `mutex_instances` and `rwlock_instances` tables help you diagnose issues with *latches.*

SHOW ENGINE INNODB STATUS

Provides detailed information about the state of the InnoDB storage engine.

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2014-07-02 04:42:02 7f1988c28700 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 4 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 29 srv_active, 0 srv_shutdown, 97 srv_idle
srv_master_thread log flush and writes: 126
...
```

SHOW ENGINE INNODB STATUS:
Semaphores and Transactions

```
...
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 1048
OS WAIT ARRAY INFO: signal count 5813
Mutex spin waits 11158, rounds 9022, OS waits 183
RW-shared spins 2244, rounds 36680, OS waits 692
RW-excl spins 392, rounds 12305, OS waits 118
Spin rounds per wait: 0.81 mutex, 16.35 RW-shared, 31.39 RW-excl
------------
TRANSACTIONS
------------
Trx id counter 12610529
Purge done for trx's n:o < 8205659 undo n:o < 0 state: running but idle
History list length 496
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 12610453, not started
MySQL thread id 69, OS thread handle 0x7f19e80de700, query id 235692
localhost oracle freeing items
SELECT ID, Name, Population FROM city_huge WHERE Name = 'Salerno‘
...
```

SHOW ENGINE INNODB STATUS: File I/O

```
...
--------
FILE I/O
--------
I/O thread 0 state: waiting for completed aio requests (insert buffer
thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
Pending normal aio reads: 0 [0, 0, 0, 0] , aio writes: 0 [0, 0, 0, 0] ,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
4378 OS file reads, 34 OS file writes, 34 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 1.00 writes/s, 1.00 fsyncs/s
...
```

SHOW ENGINE INNODB STATUS:
Insert Buffer and Adaptive Hash Index

This section displays the status of two structures that InnoDB uses to optimize performance:

- Insert buffer
- Adaptive hash index

```
...
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 276671, node heap has 603 buffer(s)
868983.25 hash searches/s, 48836.04 non-hash searches/s
...
```

SHOW ENGINE INNODB STATUS: Log

Displays information about the InnoDB transaction logs, including:

- Current log sequence number
- What proportion of the logs have been flushed to disk
- Position of the last checkpoint

```
...
---
LOG
---
Log sequence number 4373830059
Log flushed up to   4373829789
Pages flushed up to 4373820571
Last checkpoint at  4373820571
0 pending log writes, 0 pending chkp writes
37 log i/o's done, 1.00 log i/o's/second
...
```

SHOW ENGINE INNODB STATUS:
Buffer Pool and Memory

```
...
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 137363456; in additional pool allocated 0
Dictionary memory allocated 59952
Buffer pool size   8191
Free buffers       3224
Database pages     4364
Old database pages 1630
Modified db pages  1
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 4364, created 0, written 1
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead
0.00/s
LRU len: 4364, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]...
```

SHOW ENGINE INNODB STATUS: Row Operations

- Shows the status of the main thread.
- Describes the number of insert, update, delete, and read operations

```
...
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
...
```

SHOW ENGINE INNODB STATUS:
Latest Foreign Key Error

```
------------------------
LATEST FOREIGN KEY ERROR
------------------------
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195
inserting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b`
     (`A`,
  `D`) ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2:
 len 4; hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;;
     4:
 len 7; hex 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc
     khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits ...
```

SHOW ENGINE INNODB STATUS:
Latest Detected Deadlock

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185
inserting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t',
'e187358f','g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d
%H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index
symbole trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex
    61613335383138;
asc aa35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190
inserting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update ...
```

WiXEngineering

Q&A

MyRocks

H/W trends and limitations

- SSD/Flash is getting affordable, but MLC Flash is still a bit expensive
- HDD: Large enough capacity but very limited IOPS
  - Reducing read/write IOPS is very important -
    - Reducing write is harder
- SSD/Flash: Great read iops but limited space and write endurance
  - Reducing space is higher priority

Log Structured merge tree

- Data become larger, more write
- Non-Relational Databases emerge, HBase, Cassandra
- Database are also used for analysis and decision making

Databases using LSM Tree

- Bigtable
- Cassandra
- HBase
- PNUTS (from Yahoo! 阿里他爸)
- LevelDB && RocksDB
- MongoDB (wired tiger)
- SQLite (optional)
- InfluxDB

Databases using LevelDB/RocksDB

- Riak KV (TS)
- TiKV
- InfluxDB (before 1.0)
- MySQL (in facebook)
- MongoDB (in facebook's dismissed Parse)

Facebook: eat my own Rocks

Log Structured merge tree

- the log-structured merge-tree (or LSM tree) is a data structure with performance characteristics that make it attractive for providing indexed access to files with high insert volume, such as transactional log data.

Log Structured merge tree

Compaction continues creating fewer, larger and larger files

SSTable: Sorted String Table

- very simple, but also a very useful way to exchange large, sorted data segments.

How LSM works

Log Structured merge tree

```
INSERT INTO message (user_id) VALUES (31);
INSERT INTO message (user_id) VALUES (99999);
INSERT INTO message (user_id) VALUES (10000);
```

N random row modifications => A few sequential reads & writes

Bloom Filter

Checking key may exist or not without reading data,
and skipping read i/o if it definitely does not exist

RocksDB

- http://rocksdb.org/
- Forked from LevelDB
  - Key-Value LSM persistent store
  - Embedded
  - Data stored locally
  - Optimized for fast storage
- LevelDB was created by Google
- Facebook forked and developed RocksDB
- Used at many backend services at Facebook, and many external large services
- MyRocks== MySQL with RocksDB Storage Engine

MyRocks(RocksDBstorage engine for MySQL)

- Taking both LSM advantages and MySQL features
- LSM advantage: Smaller space and lower write amplification
- MySQL features: SQL, Replication, Connectors and many tools
- Fully Open Source
- Working with MariaDB Company
- Currently RC stage (Prod in percona server)
- https://github.com/facebook/mysql-5.6/

Major feature sets in MyRocks

- Similar feature sets as InnoDB
- Transaction
  - Atomicity
  - MVCC / Non locking consistent reads
  - Read Committed, Repeatable Read (PostgreSQL-style)
  - Crash safe slave and master
- Online Backup
- Logical backup by mysqldump
- Binary backup by myrocks_hotbackup

Performance and Efficiency in MyRocks

- Much smaller space and write amplification compared to InnoDB
- Reverse order index (Reverse Column Family)
- SingleDelete
- Prefix bloom filter
  - “SELECT … WHERE id=1 and time >= X” => using bloom filter for id
- Mem-comparable keys when using case sensitive collations
- Optimizer statistics without diving into pages

RocksDB Architecture

Write Path (1)

Memory
Persistent Storage
Write Request
Active MemTable
WAL
Switch
Switch
MemTable
WAL
Flush
Files
File1
File2
File3
File4
File5
File6
Compaction

Write Path (2)

Memory
Persistent Storage
Write Request
Active MemTable
WAL
Switch
Switch
MemTable
WAL
Flush
Files
File1
File2
File3
File4
File5
File6
Compaction

Write Path (3)

Memory
Persistent Storage
Write Request
Active MemTable
WAL
Switch
Switch
MemTable
WAL
Flush
Files
File1
File2
File3
File4
File5
File6
Compaction

Write Path (4)

Memory
Persistent Storage
Write Request
Active MemTable
WAL
Switch
Switch
MemTable
WAL
Flush
Files
File1
File2
File3
File4
File5
File6
Compaction

Read Path

Memory
Persistent Storage
Active MemTable
Bloom Filter
WAL
MemTable
Bloom Filter
WAL
Read Request
Index and Bloom
Filters cached In
Memory
File 1
File 3
Files
File1
File2
File3
File4
File5
File6
Block Cache

LSM Compaction Algorithm -- Level

- For each level, data is sorted by key
- Read Amplification: 1 ~ number of levels (depending on cache -- L0~L3 are usually cached)
- Write Amplification: 1 + 1 + fanout * (number of levels – 2) / 2
- Space Amplification: 1.11

Read Penalty on LSM

SELECT id1, id2, time FROM t WHERE id1=100 AND id2=100 ORDER BY time DESC LIMIT 1000;
Index on (id1, id2, time)

InnoDB

Range Scan with covering index is done by just reading leaves sequentially, and ordering is guaranteed (very efficient)

RocksDB

Merge is needed to do range scan with ORDER BY
(L0-L2 are usually cached, but in total it needs more CPU cycles than InnoDB)

Delete penalty

- "Delete" adds a tombstone
  - When reading, ignoring *all* Puts for the same key
- Tombstones can't disappear until bottom level compaction happens
- Some reads need to scan lots of tombstones => inefficient
  - In this example, reading 5 entries is needed just for getting one row

“SingleDelete” optimization in RocksDB

INSERT INTO t
VALUES (1),(2),(3),(4),(5);

DELETE FROM t WHERE
id <= 4;

SELECT COUNT(*) FROM t;

- If Put for the same key is guaranteed to happen only once, SingleDelete can remove Put and itself
  - Reading just one entry is ok – more efficient than reading 5 entries
- MyRocks uses SingleDelete whenever possible

LSM on Disk

- Main Advantage
  - Lower write penalty
- Main Disadvantage
  - Higher read penalty

- Good fit for write heavy applications

my.cnf(minimal configuration)

```
[mysqld]
rocksdb
default-storage-engine=rocksdb
skip-innodb
default-tmp-storage-engine=MyISAM
collation-server=latin1_bin (or utf8_bin, binary)
log-bin
binlog-format=ROW
```

Add “allow-multiple-engines” in my.cnf, if you really want to mix InnoDB and MyRocks

Create Table

```
CREATE TABLE t (id INT PRIMARY KEY,
value1 INT,
value2 VARCHAR (100),
INDEX (value1)) ENGINE=RocksDB COLLATE latin1_bin;
```

- It is generally recommended to have a PRIMARY KEY, though MyRocksallows tables without PRIMARY KEYs
- Tables are automatically compressed.

WiXEngineering

Q&A

PERFORMANCE SCHEMA

The Performance Schema

- New feature in MySQL 5.5 (Peter Gulutzan, Marc Alff)
- Monitors low-level server events (mutex, rwlock and IO etc.)
- Instrumentations give you both information about “Elapsed time” and “counter” for the events
- Expose as SQL tables for easy query, can be aggregated, averaged, ordered etc.

The Performance Schema

In MySQL 5.5 the Performance Schema caused around an 8% to 11% overhead even when it was collecting no data,

and 19% to 25% with all consumers enabled, depending on whether it was a read-only or read/write workload.

Whether this is a little or a lot is up to you to decide.

The Performance Schema: The Idea

Executing a query is broken down to hundreds of smaller tasks

There are background tasks as well

We want to instrument it all to know where server is spending time

The Performance Schema: Enable

from MySQL 5.6, you can enable the Performance Schema, but leave it inactive until you want to gather some information:

In the [mysqld] section of the my.cnf

```
performance_schema=on
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
performance-schema_instrument='wait/synch/cond/%=counted'
```

Performance Schema

Helps you investigate performance problems

Contains many tables

- Tables are held in memory
- You must understand the schema to use it effectively
- Consider using the MySQL `sys` schema helper tool

Uses its own storage engine: `PERFORMANCE_SCHEMA`

Enables runtime configuration via setup tables:

- **Instruments:** What is monitored
- **Consumers:** What information is stored about the instruments
- **Timers:** What timer is used for each event type
- **Actors:** Which client connections to monitor
- **Objects:** Which database objects to monitor

Actors
User
User
User
Instruments
Stages
Waits
Statements
Database Objects
DB object
DB object
DB object
Consumers
events_statements tables
events_stages tables
events_waits tables

Configuring Performance Schema

Configure Performance Schema by making entries in the `setup_*` tables:

```
mysql> SHOW TABLES LIKE 'setup\_%';
+----------------------------------------+
| Tables_in_performance_schema (setup\_%) |
+----------------------------------------+
| setup_actors                           |
| setup_consumers                        |
| setup_instruments                      |
| setup_objects                          |
| setup_timers                           |
+----------------------------------------+
5 rows in set (0.00 sec)
```

The setup_actors Table

Specifies the user accounts that are instrumented

All user accounts are enabled by default

```
mysql> SELECT * FROM setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
1 row in set (0.00 sec)
```

Background threads are always instrumented

Unless they are disabled in the `threads` table.

Changes affect only new connections

The setup_instruments Table

```
mysql> SHOW CREATE TABLE performance_schema.setup_instruments\G
*************************** 1. row ***************************
Table: setup_instruments
Create Table: CREATE TABLE `setup_instruments` (
  `NAME` varchar(128) NOT NULL,
  `ENABLED` enum('YES','NO') NOT NULL,
  `TIMED` enum('YES','NO') NOT NULL
) ENGINE=performance_schema DEFAULT CHARSET=utf8
```

NAME: The name of the instrument. Names are hierarchical, like a file path.

For example:

```
wait/io/file/innodb/innodb_data_file
wait/sync/mutex/sql/PAGE::lock
statement/sql/select
```

ENABLED: Whether the instrument is turned on.

TIMED: Whether instrumented events are timed.

The setup_consumers Table

```
mysql> SELECT * FROM setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
12 rows in set (0.18 sec)
```

- Lists the destinations where event information is sent and stored.
- Consumers form a hierarchy from higher levels to lower
- Consumers are only active if the consumers they depend on are active.

The setup_timers Table

```
mysql> SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2489004975 |                1 |             21 |
| NANOSECOND  |      1000000000 |                1 |             42 |
| MICROSECOND |         1000000 |                1 |             42 |
| MILLISECOND |            1038 |                1 |             48 |
| TICK        |             106 |                1 |            580 |
+-------------+-----------------+------------------+----------------+
5 rows in set (0.00 sec)
```

Configuring Instrumentation

Monitoring can affect performance.

- Only monitor what you need to.
- Benchmark before and after making changes.

Update the `setup_instruments` table.

Examples:Turn off timing for all CSV-related instruments:

```
UPDATE setup_instruments SET TIMED = 'NO'
WHERE NAME LIKE '%CSV%';
```

Disable all CSV-related instruments:

```
UPDATE setup_instruments SET ENABLED = 'NO'
WHERE NAME LIKE '%CSV%';
```

Enable only I/O-related instruments:

```
UPDATE setup_instruments SET ENABLED =
IF(NAME LIKE '%/io/%', 'YES', 'NO');
```

The Connection Tables

The `users`, `hosts`, and `accounts` tables store connection metrics.

Example: Connections per account:

```
mysql> SELECT * FROM accounts;
+------+-----------+---------------------+--------------------+
| USER | HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+------+-----------+---------------------+--------------------+
| root | localhost |                   1 |                 15 |
| NULL | NULL      |                  18 |                 20 |
+------+-----------+---------------------+--------------------+
2 rows in set (0.00 sec)
```

Attribute tables provide more detailed information about individual connections:

`session_connect_attrs`

`session_account_connect_attrs`

The Instance Tables

Store information about the objects being instrumented

```
mysql> SHOW TABLES LIKE '%\_instances';
+---------------------------------------------+
| Tables_in_performance_schema (%\_instances) |
+---------------------------------------------+
| cond_instances                              |
| file_instances                              |
| mutex_instances                             |
| rwlock_instances                            |
| socket_instances                            |
+---------------------------------------------+
5 rows in set (0.00 sec)
```

Contain the following information:

- A name for the event
- The object associated with the event
- Status information about the object

The Event Tables

- Three groups of tables store events based on their type:
  - **Stages**: Correspond to thread states in the State column of SHOW PROCESSLIST, for example: Sorting data
  - **Statements**: SQL statements that are executing or have been executed recently
  - **Waits**: Where the server is spending time
- Three tables store the raw data for each event type:
  events_[type]_current
  events_[type]_history
  events_[type]_history_long
- Each event type also has a number of summary tables
  Provide aggregated information for all events over time
- Every event in the events tables is named after its corresponding instrument.

Example: Examining Raw Event Data

Which `SELECT` queries did not use indexes?

```
SELECT SQL_TEXT FROM events_statements_history_long
WHERE no_index_used !=0 AND
EVENT_NAME="statement/sql/select";
```

What were the five longest waits for a particular thread?

```
SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
FROM events_waits_history
WHERE THREAD_ID = 33
ORDER BY TIMER_WAIT DESC LIMIT 5;
```

What stage is taking the longest for a particular thread?

```
SELECT EVENT_NAME, MAX(TIMER_WAIT)
FROM events_stages_history
WHERE THREAD_ID=37;
```

Example: Examining Raw Event Data (Notes Only)

Example: Retrieving Statement Summary Information

```
mysql> SELECT * FROM events_statements_summary_by_digest\G

********************************** 1. row **************************************
               SCHEMA_NAME: world_innodb        SUM_CREATED_TMP_TABLES: 0
                    DIGEST: 497c7…                SUM_SELECT_FULL_JOIN: 1
                DIGEST_TEXT: SELECT NAME    SUM_SELECT_FULL_RANGE_JOIN: 0
    FROM `Country` WHERE CODE IN ( SELECT             SUM_SELECT_RANGE: 0
    CODE FROM `CountryLanguage` WHERE           SUM_SELECT_RANGE_CHECK: 0
    `population` > ? )                                 SUM_SELECT_SCAN: 1
                 COUNT_STAR: 1                   SUM_SORT_MERGE_PASSES: 0
            SUM_TIMER_WAIT: 84878009000                 SUM_SORT_RANGE: 0
            MIN_TIMER_WAIT: 84878009000                  SUM_SORT_ROWS: 0
            AVG_TIMER_WAIT: 84878009000                  SUM_SORT_SCAN: 0
            MAX_TIMER_WAIT: 84878009000             SUM_NO_INDEX_USED: 1
             SUM_LOCK_TIME: 1054000000          SUM_NO_GOOD_INDEX_USED: 0
                 SUM_ERRORS: 0                              FIRST_SEEN: 2014-03-07
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0                                12:17:54
              SUM_ROWS_SENT: 163                                         LAST_SEEN:
          SUM_ROWS_EXAMINED: 239                2014-03-07
SUM_CREATED_TMP_DISK_TABLES: 0
                                                             12:17:54
                                          ...
```

Event Nesting

- Events form a hierarchy, where one event triggers another

Represented by columns in the non-summary event tables

NESTING_EVENT_ID and NESTING_EVENT_TYPE

```
SELECT ... FROM events_stages_history_long WHERE THREAD_ID=3462536;

+----------+-------------------------------+-------------------+---------------------+
| EVENT_ID | EVENT_NAME                    | NESTING_EVENT_ID  | NESTING_EVENT_TYPE  |
+----------+-------------------------------+-------------------+---------------------+
|     1413 | stage/sql/init                |              1412 | STATEMENT           |
|     1417 | stage/sql/checking permissions |             1412 | STATEMENT           |
...
```

```
SELECT ... FROM events_statements_history_long WHERE EVENT_ID=1412;

+-----------+----------+----------------------+------------------+
| THREAD_ID | EVENT_ID | EVENT_NAME           | SQL_TEXT         |
+-----------+----------+----------------------+------------------+
|   3462536 |     1412 | statement/sql/select | select * from ...|
+-----------+----------+----------------------+------------------+
```

Event Nesting (Notes Only)

WiXEngineering

Q&A

SYS SCHEMA

The MySQL sys Schema

*ps_helper

- Makes it easier to interpret raw performance data

Contains:

  Views

  Functions

  Stored procedures

- Joins `performance_schema` tables together to provide extra information
- Implements many common DBA and developer use cases
- Can be installed and enabled from within MySQL Workbench, or independently
- Is available for download from: https://github.com/MarkLeith/mysql-sys/

Using sys Views to Analyse Statements

Use the statement analysis views for monitoring and debugging query performance:

```
statement_analysis:
*************************** 1. row ***************************
        query: SELECT * FROM `world_innodb` . `Country` ORDER BY NAME
    full_scan: *
   exec_count: 2
    err_count: 0
   warn_count: 0
total_latency: 46.28 ms
  max_latency: 38.80 ms
  avg_latency: 23.14 ms
    rows_sent: 478
rows_sent_avg: 239
 rows_scanned: 956
       digest: c77db8834a05545537422262ac511cd5
...
statements_with_full_table_scans
statements_with_runtimes_in_95th_percentile
```

Using sys Views for Schema Statistics

Use the following views to diagnose schema issues, particularly missing or unused indexes:

- schema_table_statistics
- schema_index_statistics
- schema_unused_indexes

```
mysql> SELECT * FROM sys.schema_unused_indexes WHERE object_schema='world_innodb';
+---------------+-----------------+-------------+
| object_schema | object_name     | index_name  |
+---------------+-----------------+-------------+
| world_innodb  | City            | PRIMARY     |
| world_innodb  | City            | CountryCode |
| world_innodb  | CountryLanguage | PRIMARY     |
+---------------+-----------------+-------------+
3 rows in set (0.02 sec)
```

- schema_tables_with_full_table_scans

The sys Utility Functions

The `sys` schema contains several utility functions. These include:

`format_time()/format_bytes()`: Make raw output easier to read.

Example (picoseconds to `HH:MM:SS`):

```
mysql> SELECT format_time(342342342342345) AS TidyTime;
+----------+
| TidyTime |
+----------+
| 00:05:42 |
+----------+
1 row in set (0.00 sec)
```

`format_statement()/format_path()`: Compress data for display at the command-line.

`extract_[schema | table]_from_file_name()`: Extract the schema name from a raw file path.

The sys Stored Procedures

For `performance_schema` configuration:

- List enabled or disabled instruments
- Enable or disable an instrument
- Enable or disable specific threads
  - Example: Enable thread instrumentation for current connection:

```
mysql> CALL sys.ps_setup_enable_thread(CONNECTION_ID());
+-------------------+
| summary           |
+-------------------+
| Enabled 1 thread  |
+-------------------+
1 row in set (0.00 sec)
```

- Enable or disable all background threads
- Save or restore current configuration

For instrumentation analysis:

- Tracing threads
- Analyzing statement digests

WiXEngineering

Q&A

WIXEngineering

Thank You

Schema Design and index usage

B+trees

- When we mean “add an index” we usually mean “add a B+tree index”:
  - Includes PRIMARY, UNIQUE, INDEX type indexes.
- Understanding the basic structure of B+trees helps with optimization and schema performance

Binary Tree

- Not the same as a B+tree
- Understand Binary Tree first then compare and contrast

Binary Tree

- The main property of a binary tree is that it has binary search.
- This means that to find the value 829813 in a (balanced) binary tree of 1 million items it should take no more than 20 hops.
- This is a considerable improvement over scanning 1 million entries in a list:

B+trees

- Amortizes disk accesses by clustering into pages
- B-trees are wide not deep

The diagram shows that the root index page points to 25 different index pages, which themselves point to on average 1538 leaf pages. The path to row 829813 is highlighted:

From the root page: values >= 800788 but < 829908 are on page 16386.
From page 16386: values >= 829804 but < 829830 are on leaf page 32012.

B+trees

- Keeping the root page *plus* 25 internal index pages in memory only requires 400KiB of memory (pages are 16KiB each).
- The 38463 leaf pages will require 600MiB for storage, but do not necessarily have to all be in memory at once.
- Most data sets have some pages that are hot, and others that are untouched. Internally InnoDB is keeping track of page access using a modified LRU algorithm and will evict the less frequently accessed pages if it needs to make room.

B+trees

- It is usually possible to have a total data size far exceed RAM. We refer to the portion of the data that needs to be kept in memory as the working set. Every data set has different working set requirements, and the goal of the database administrator is to find optimizations that can reduce it.

- A 500GB database with a 300GB working set requirement is often much harder to optimize than a 1TB database with a 100GB working set.

B+trees

CREATE TABLE `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(32) NOT NULL,
`payload` text,
PRIMARY KEY (`id`)
) ENGINE=InnoDB
SELECT * FROM users
WHERE id = 829813
Root Page
>1 #36
>14548 #37
>43668 #38
>72788 #39
>101908 #40
>131028 #41
>160148 #42
>189268 #43
>218388 #44
>247508 #45
x15 more enties... ..
>713428 #61
>742548 #62
>771668 #63
>800788 #16386
>829908 #16387
>859028 #16388
>888148 #34368
>917268 #34369
>946388 #34370
>975508 #34371
96% Remaining Free Space
25 Level 1 Index Pages
>800788 #30896
>800814 #30897
>800840 #30898
>800866 #30899
>800892 #30900
>800918 #30901
>800944 #30902
>800970 #30903
>800996 #30904
>801022 #30905
x1105 more enties... ..
>829778 #32011
>829804 #32012
>829830 #32013
>829856 #32014
>829882 #32015
38463 Leaf Pages
829804 {row payload, ..}
829805 {row payload, ..}
829806 {row payload, ..}
829807 {row payload, ..}
829808 {row payload, ..}
829809 {row payload, ..}
829810 {row payload, ..}
829811 {row payload, ..}
829812 {row payload, ..}
829813 {row payload, ..}
829814 {row payload, ..}
829815 {row payload, ..}
829816 {row payload, ..}
829817 {row payload, ..}
829818 {row payload, ..}
829819 {row payload, ..}
829820 {row payload, ..}
829821 {row payload, ..}
829822 {row payload, ..}
829823 {row payload, ..}
829824 {row payload, ..}
829825 {row payload, ..}
829826 {row payload, ..}
829827 {row payload, ..}
829828 {row payload, ..}
829829 {row payload, ..}

What Operations can BTREE Index do ?

- Find all rows with KEY=5 (point lookup)
- Find all rows with KEY>5 (open range)
- Find all rows with 5<KEY
- NOT find all rows with last digit of the KEY is Zero
- LIKE “ABC%” means

“ABC[LOWEST]”<KEY

- LIKE “%ABC” can’t be optimized by use of the index

Multiple Column Indexes

- Sort Order is defined, comparing leading column, then second etc
- It is still one BTREE Index

```
KEY(col1,col2,col3)
(1,2,3) < (1,3,1)
```

Multiple Column Indexes

- Index (A,B,C) - order of columns matters
- Will use Index for lookup (all listed keyparts)
  - A>5
  - A=5 AND B>6
  - A=5 AND B=6 AND C=7
  - A=5 AND B IN (2,3) AND C>5
- Will NOT use Index
  - B>5 – Leading column is not referenced
  - B=6 AND C=7 - Leading column is not referenced
- Will use Part of the index
  - A>5 AND B=2 - range on first column; only use this key part
  - A=5 AND B>6 AND C=2 - range on second column, use 2 parts

Multiple Column Indexes

- MySQL will stop using key parts in multi part index as soon as it met the real range (, BETWEEN), it however is able to continue using key parts further to the right if IN(…) range is used

Using Index for Sorting

- SELECT * FROM PLAYERS ORDER BY SCORE DESC LIMIT 10
  - Will use index on **SCORE** column
  - Without index MySQL will do "**filesort**" (external sort) which is very expensive
- SELECT * FROM PLAYERS WHERE COUNTRY="US" ORDER BY SCORE DESC LIMIT 10
  - Index on (COUNTRY,SCORE)

Using Index for Sorting

KEY(A,B)

- Will use Index for Sorting
  - ORDER BY A - sorting by leading column
  - A=5 ORDER BY B - EQ filtering by 1st and sorting by 2nd
  - ORDER BY A DESC, B DESC - Sorting by 2 columns in same order
  - A>5 ORDER BY A - Range on the column, sorting on the same
- Will NOT use Index for Sorting
  - ORDER BY B - Sorting by second column in the index
  - A>5 ORDER BY B – Range on first column, sorting by second
  - A IN(1,2) ORDER BY B - In-Range on first column
  - ORDER BY A ASC, B DESC - Sorting in the different order

Using Index for Sorting

- can’t sort in different order by 2 columns
- can only have Equality comparison (=) for columns which are not part of ORDER BY

Min/Max Optimizations

Index help MIN()/MAX() aggregate functions But only these

- SELECT MAX(ID) FROM TBL;
- SELECT MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT_ID
  - Will benefit from (DEPT_ID,SALARY) index
  - “Using index for group-by”

MySQL Performs Joins as “Nested Loops” Only

*SELECT * FROM POSTS,COMMENTS WHERE*
AUTHOR=“Peter”
AND
COMMENTS.POST_ID=POSTS.ID

- Scan table POSTS finding all posts which have Peter as an Author
- For every such post go to COMMENTS table to fetch all comments
- The index on **POSTS.ID** is not needed for this query performance
- Very important to have all JOINs Indexed

Overhead of The Indexing

- Indexes are expensive Do not add more than you need.
- In most cases extending index is better than adding new one.
- Updating indexes is often major cost of database writes
- On read - Wasted space on disk and in memory additional overhead during query optimization.

INNODB

InnoDB uses a Clustered Index

- In InnoDB the data rows are also stored in a B+tree, organized by the primary key
- Secondary key indexes always include the value of the primary key

NULL

- Always specify NOT NULL if possible
- An SQL NULL value reserves one or two bytes in the record directory.
- An SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column.
- In a fixed-length column, it reserves the fixed length of the column in the data part of the record.

NULL

- Don’t do this :

```
CREATE TABLE t1 (
  id INT UNSIGNED NOT NULL
AUTO_INCREMENT PRIMARY KEY
, dt DATETIME NOT NULL DEFAULT '0000-00-
00 00:00:00'
) ENGINE = InnoDB;
```

- do this :

```
CREATE TABLE t2 (
  id INT UNSIGNED NOT NULL
AUTO_INCREMENT PRIMARY KEY
, dt DATETIME NULL DEFAULT NULL
) ENGINE = InnoDB;
```

DATA TYPE

- If you have a field that will hold values between 1 and 10, don't specify that field as BIGINT (or even INT, for that matter). Store it in a TINYINT instead
- A Tinyint field can store values between -128 and 127 (0 and 256 if Unsigned), and is only 1-byte in size (4 bit int 8 bit bigint)
- If you know your username's or passwords cannot be longer than 40 characters, don't store them in a CHAR(256)

Numeric Data Type

- Integer: TINYINT, SMALLINT,
- MEDIUMINT, INT, BIGINT
- Floating Point: FLOAT, DOUBLE
- Fixed Point: DECIMAL
- Other: BIT, ENUM

Numeric Data Type

- INT(1) - 1 does not mean 1 digit

(1) represents client output display format only

- INT is 4 Bytes, TINYINT is 1 Byte
- TINYINT UNSIGNED can store from 0 – 255
- BIT is even better when values are 0 - 1

DATA TYPE

- Use ENUM

```
player_type      VARCHAR(32);   -- can be: 'Admin',
'Standard'
```

```
player_type      ENUM('Admin', 'Standard');   --
```

DATA TYPE

- Always used UNSIGNED when possible if filed cannot or should not contain negative values, store them as UNSIGNED. Not only will this double the amount of values that can be held in the variable, but it also ensures data integrity

DATA TYPE

- VARCHAR(255) – don’t do this . Poor Design

```
CREATE TABLE `XXX` (
`orderHandle` varchar(255) NOT NULL default '',
`personName` varchar(255) default NULL,
`addressLines` varchar(255) default NULL,
`city` varchar(255) default NULL,
`state` varchar(255) default NULL,
`postalCode` varchar(255) default NULL,
`countryCode` varchar(255) default NULL,
`phone` varchar(255) default NULL,
`email` varchar(255) default NULL,
`shipMethod` varchar(255) default NULL,
`shipTo` varchar(255) default NULL,
`receiveByDate` date default NULL,
`currency` varchar(3) default NULL,
`price` varchar(255) default NULL,
`flags` int(11) default '0',
```

DATA TYPE
TEXT/LONGTEXT

- TEXT/BLOB – try not to use :
  - Using SELECT * - > MySQL Internal Temporary table will force Temp Disk Table
- Innodb will Stores first 768 bytes, then a separate 16k data page per row per TEXT/BLOB field
- If using longtext/text for JSON use JSON type.
- Placing optional TEXT/BLOB columns in second table.

WiXEngineering

Q&A

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

partitions

Best use of partition is to fast delete data

Partitions can support deleting data by dropping a partition

- If you insert 1,000,000 rows a day, eventually you need to delete 1,000,000 rows a day
- Only works with **Range** and **List** partitioning
- Very useful for rolling date/time range
- Can be very useful even for small increments, i.e.1 hour
- Very fast, can be ~ 1-2 Secs

```
ALTER TABLE Sal DROP PARTITION May;
```

```
Alter table sal add PARTITION
AUG VALUES LESS
 THAN (TO_DAYS('2013-09-01'))
```

partitions

```
ALTER TABLE Sal DROP
PARTITION May;
```

```
Alter table sal add PARTITION
AUG VALUES LESS
 THAN (TO_DAYS('2013-09-01'))
```

partitions

Partitions: Faster non-index Data Access

- The MySQL optimizer is aware of the partitioning expression and can eliminate partitions to scan when the columns used in the partitioning expression are in the queries where condition
- Reduce or even **eliminate** indexes!

RANGE Partitioning

Use **PARTITION BY RANGE** to partition rows that contain specified values within a set of specified ranges.

In this "store orders" example, search ranges are specified for the data related to the orders:

- A partition for orders with ID numbers less than 10,000
- A partition for orders with ID numbers between 10,000 and 19,999
- And so on, through 50,000

LIST Partitioning

Use `PARITION BY LIST` to partitions rows that contain specified values.

In this “store orders” example, partitions are specified for orders by store IDs, per district:

HASH Partitioning

`PARTITION BY HASH` ensures an even distribution of data among partitions.

Unlike `RANGE` or `LIST`, `HASH` does not require individual partition definition.

In this "store orders" example, an integer is specified to partition the table into four equal partitions:

KEY Partitioning

PARTITION BY KEY is similar to **HASH** except that the MySQL server uses its own hashing expression.

Does not require that the partition expression return an integer or **NULL**.

In this "store orders" example, partitions are specified by date of order:

Subpartitioning

RANGE and **LIST** partitioned tables can be subpartitioned.

The subpartitions themselves can be **HASH** or **KEY**.

In this "store orders" database example, based on the `orders_range` table, you can further divide a table so that it splits each partition into two subpartitions:

partitions

Optimizations can be done by partition

- If data is only being added to one partition then you can OPTIMIZE ***only that partition*** instead of running OPTIMIZE on the whole table

parallel query execution

simple query from US Flights Ontime performance database

http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

The query is simple, however, it will have to scan 150M rows

```
explain select yeard, count(*) from ontime group by yeard G
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: ontime
type: index
possible_keys: YearD,comb1
key: YearD
key_len: 1
ref: NULL
rows: 148046200
Extra: Using index
1 row in set (0.00 sec)
```

parallel query execution

```
select yeard, count(*) from ontime group by yeard;
+-------+----------+
| yeard | count(*) |
+-------+----------+
| 1988  | 5202096  |
| 1989  | 5041200  |
| 1990  | 5270893  |
| 1991  | 5076925  |
…….
| 2010  | 6450117  |
| 2011  | 6085281  |
| 2012  | 6096762  |
| 2013  | 5349447  |
+-------+----------+
26 rows in set (54.10 sec)26 rows in set (54.10 sec)
```

parallel query execution

We can run 26 parallel queries, each will count its own year.:

```
#!/bin/bash
date
for y in {1988..2013}
do
  sql="select yeard, count(*) from ontime where yeard=$y"
  mysql -vvv ontime -e "$sql" &>par_sql1/$y.log &
done
wait
Date
```

parallel query execution

```
par_sql1/1988.log:1 row in set (3.70 sec)
par_sql1/1989.log:1 row in set (4.08 sec)
par_sql1/1990.log:1 row in set (4.59 sec)
par_sql1/1991.log:1 row in set (4.26 sec)
par_sql1/1992.log:1 row in set (4.54 sec)
.....
par_sql1/2008.log:1 row in set (5.59 sec)
par_sql1/2009.log:1 row in set (4.44 sec)
par_sql1/2010.log:1 row in set (4.91 sec)
par_sql1/2011.log:1 row in set (5.08 sec)
par_sql1/2012.log:1 row in set (4.85 sec)
par_sql1/2013.log:1 row in set (4.56 sec)
```

Start: 11:41:21 EST 2014
End: 11:41:26 EST 2014
(10x faster)