ORACLE OPEN WORLD

#MySQL #oow16

MySQL Performance Tuning 101

Jesper Wisborg Krogh

ORACLE

MySQL

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

MySQL Performance Tuning 101

1. MySQL Performance Tuning Best Practices
2. Think
3. Monitoring
4. Create the Initial MySQL Configuration File
5. Buffer and Caches
6. Data consistency versus performance
7. Query Tuning
8. Consider the whole stack
9. Summary

MySQL Performance Tuning Best Practices

ORACLE

MySQL Performance Tuning Best Practices

- Be wary about “best practices”
  - No two systems are the same
  - What was true earlier may no longer hold

MySQL Performance Tuning Best Practices

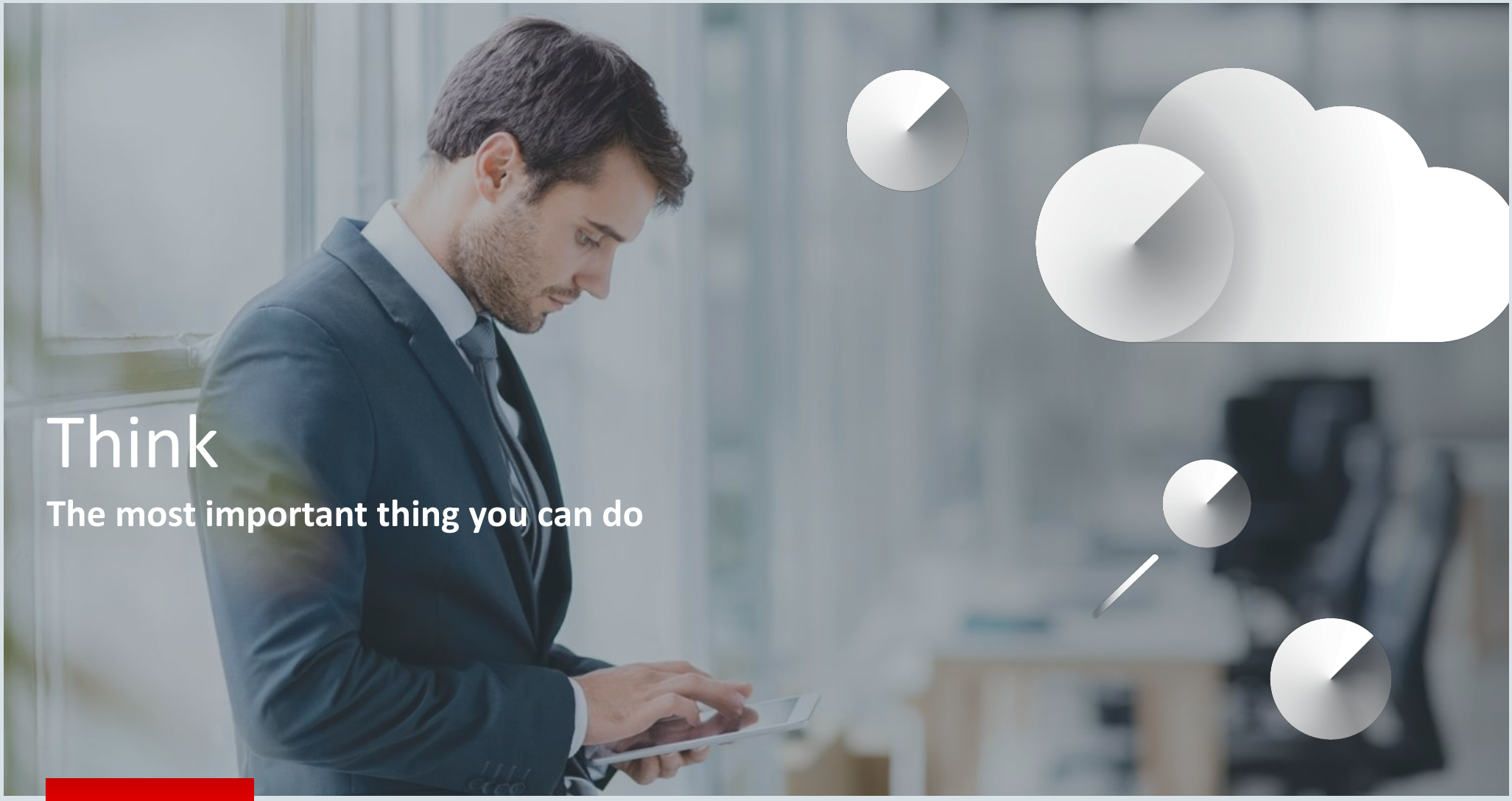
- Be wary about “best practices”
  - No two systems are the same
  - What was true earlier may no longer hold
- That said some guidelines can be given

MySQL Performance Tuning Best Practices

- Think – consider what you are doing!
- Monitor your system
- Ensure you test your changes before deploying to production
  - The testing must reflect your production usage
- Make incremental changes
  - One change at a time
  - Relatively small changes

MySQL Performance Tuning Best Practices

- Be mindful of your requirements
  - Some options give the choice between performance or data safety – what do you need?
- Often the default value is the best value
- Ensure all tables have a PRIMARY KEY
- InnoDB organizes the data according to the PRIMARY KEY:
  - The PRIMARY KEY is included in all secondary indexes in order to be able to locate the actual row => smaller PRIMARY KEY gives smaller secondary indexes.
  - A mostly sequential PRIMARY KEY is in general recommended to avoid inserting rows between existing rows.



Think

- Thinking is your best defense
- Analyze your performance problem:
  - Make sure you understand what the issue is:
    - No: performance is too slow
    - Yes:
      - The query takes 10 seconds, but it is used for interactive use so must complete in 0.1 second or less
      - The server must handle 200k queries per second
  - Determine the cause
    - Don’t jump to conclusions
    - Consider the whole stack
    - Justify why you think you have found the cause

Think

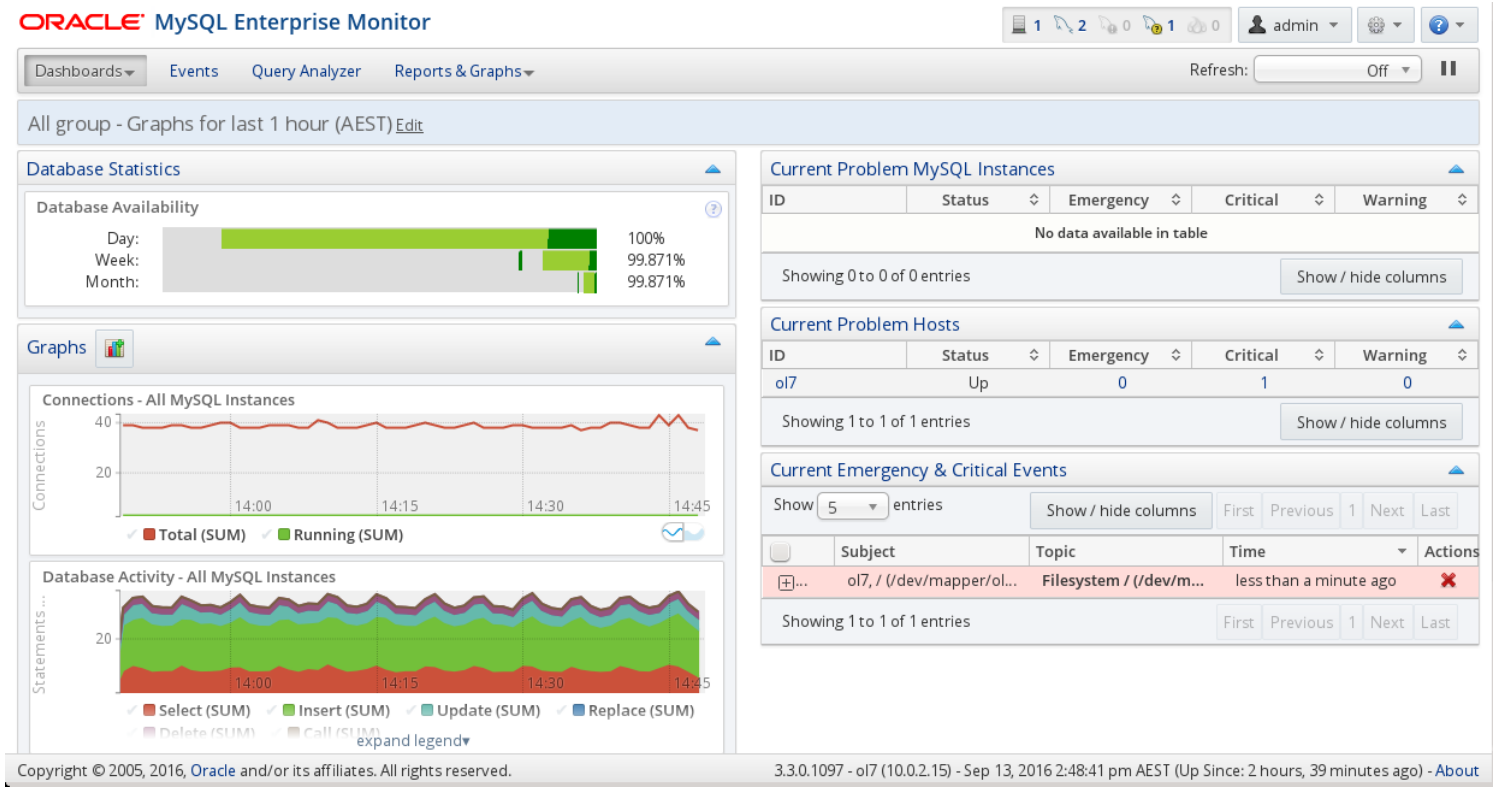
- Implement the solution
  - List all the solutions you can think of
    - Make sure you think outside the box
  - Explain why the solutions will work
  - Implement an action plan:
    - Test the action plan and update if necessary
    - Ensures you implement the solution the same on the test and production systems
    - If a regression occurs, you have a record of the steps you made

Monitoring

- Gives you a base line
- Tells you what has happened – useful to investigate performance issues
- Allows you to proactively handle potential issues

Monitoring

MySQL Enterprise Monitor



Monitoring

- Many options available:
  - MySQL Enterprise Monitor
  - MySQL Plugin for Oracle Enterprise Monitor
  - Cacti
  - Zabbix
  - Nagios
  - And many more
- Make sure you configure alerts so you react appropriately to all events based on the severity level!

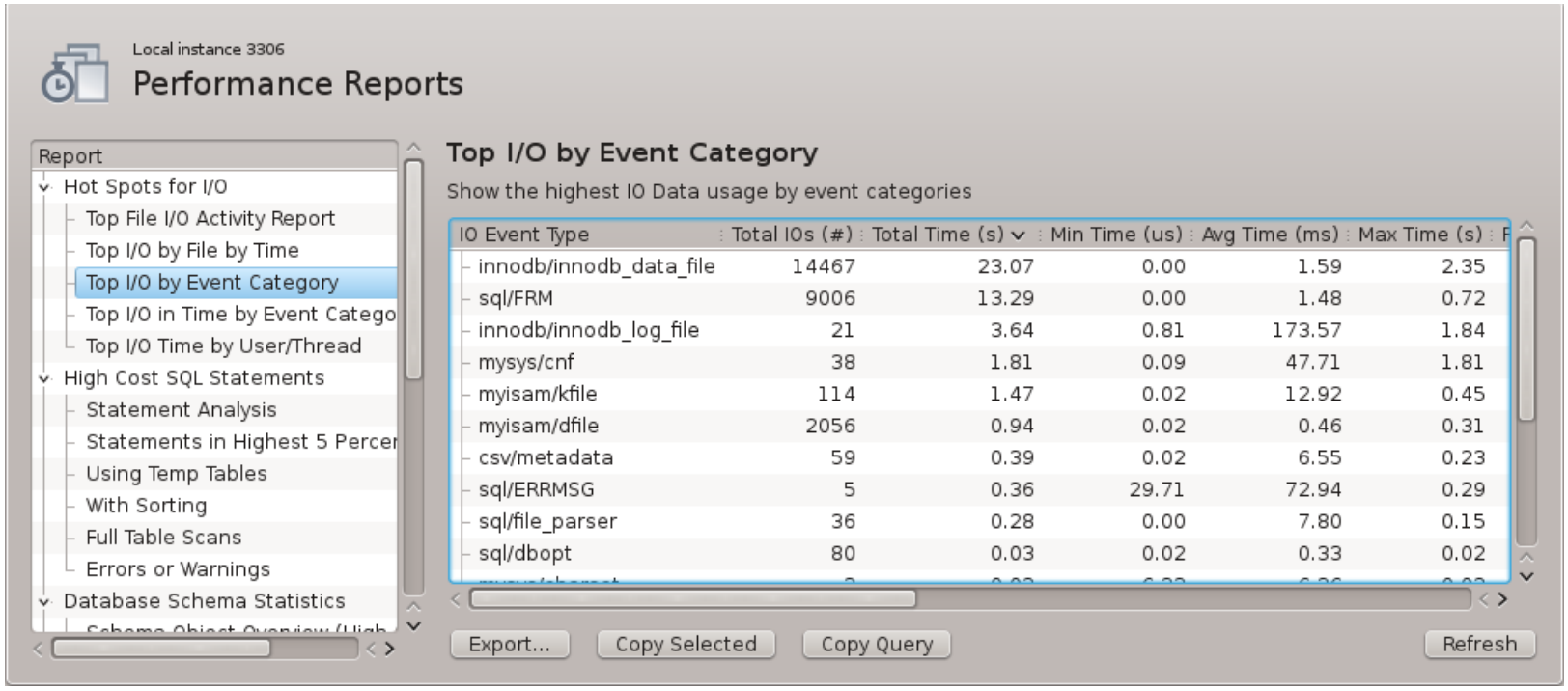
Monitoring

“Real time” monitoring

- On Linux perf is a great tool for live monitoring and can record over a time period
  - https://perf.wiki.kernel.org/index.php/Main_Page
  - http://www.brendangregg.com/perf.html
- MySQL Enterprise Monitor has some reports to get snapshot data
- Workbench has performance reports based on the sys schema

Monitoring

MySQL Workbench Performance Reports



Monitoring

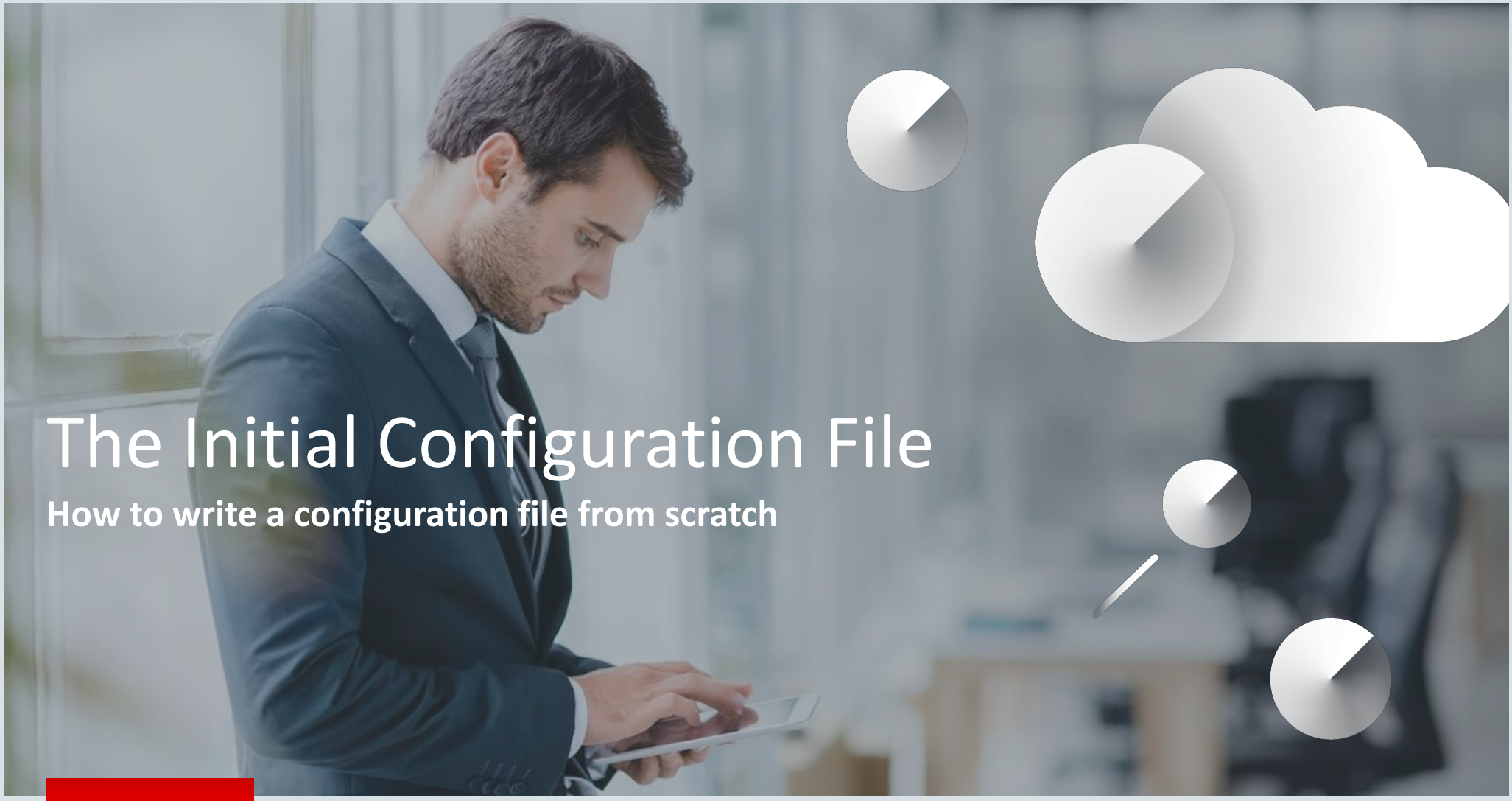
What was that sys schema?

- The sys schema is a collection of views, functions, and procedures aiming to make the Information Schema and Performance Schema simpler to use.
- Included by default in MySQL 5.7+
- Also available from https://github.com/mysql/mysql-sys for MySQL 5.6
- Formerly known as ps_helper by Mark Leith
- For a deep dive into the sys schema, see Mark Leith's talk Thursday at 1:15pm in Park Central - City
- https://dev.mysql.com/doc/refman/5.7/en/sys-schema.html

Monitoring

Monitoring has overhead

- While monitoring is good, there is such as thing as too much monitoring
- All monitoring has overhead – some more than others
- Particularly be careful with queries like:
  - SHOW PROCESSLIST
  - SHOW ENGINE INNODB STATUS
- If performed too often they can cause an outage on their own
- In 5.6+: Use performance_schema.threads instead of SHOW PROCESSLIST



The Initial Configuration File

How to write a configuration file from scratch

Initial Configuration File

1. Start with an empty configuration file
2. Set paths, port, etc.
3. Enable additional monitoring
4. Set capacity settings
5. Don’t do much more!

Initial Configuration File

Start with empty configuration file

- Remember the configuration templates in MySQL 5.5 and earlier?
  - my-huge.cnf/my-huge.ini
  - my-innodb-heavy-4G.cnf/my-innodb-heavy-4G.ini (20kB!)
  - my-large.cnf/my-large.ini
  - my-medium.cnf/my-medium.cnf
  - my-small.cnf/my-small.cnf
- In MySQL 5.6+:
  - my-default.cnf (1.1kB)

Initial Configuration File

my-default.cnf

```
[mysqld]

# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M

# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin

# These are commonly set, remove the # and set as required.
# basedir = .....
# datadir = .....
# port = .....
# server_id = .....
# socket = .....

# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M

sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
```

Initial Configuration File

Start with empty configuration file

- Lots of work has gone into improving the defaults in MySQL 5.6 and 5.7
- The work continues
- So it is no longer necessary with the huge templates

Initial Configuration File

Paths

- Some important paths:
  - datadir – where the data lives by default – default location for:
    - Data and indexes for non-InnoDB tables
    - InnoDB file-per-table, general, system, and undo log tablespaces
  - innodb_data_home_dir – default location for the InnoDB system tablespace (ibdata1)
  - innodb_log_group_home_dir – path to the InnoDB redo log files
  - innodb_undo_directory – path to the InnoDB undo tablespaces
  - log_bin – dual functionality: enable binary logging and set path/file name prefix
  - log_error – makes sure you know where it is

Initial Configuration File

Paths

- It can be an advantage to have the paths pointing to separate disk systems
  - Reduces contention
  - I/O often becomes a bottleneck
  - Can place the hot files on faster disks
    - For example with innodb_flush_log_at_trx_commit = 1 and high transaction commit rate, it can be necessary to place the InnoDB redo log on SSDs to support the high rate of flushes.
- Note: File-per-table tablespaces and general tablespaces can located outside datadir when they are created

Initial Configuration File

Enable additional monitoring

- If you use InnoDB, enable all the INNODB_METRICS counters:
  - innodb_monitor_enable = '%'
    - The overhead has turned out to be small so worth the extra details
- Ensure the Performance Schema is enabled
  - Does have overhead, but provides very useful information for performance tuning
  - Consider enabling additional consumers and instruments as required (for example for transactions in MySQL 5.7)
  - The Performance Schema can also be configured dynamically at runtime

Initial Configuration File

Capacity settings

- Most important options for performance:
  - innodb_buffer_pool_size for InnoDB workloads
  - key_buffer_size for MyISAM workloads
- Other important InnoDB options:
  - innodb_buffer_pool_instances
  - innodb_log_file_size / innodb_log_files_in_group
  - innodb_thread_concurrency
  - innodb_undo_tablespaces
  - innodb_file_per_table (enabled by default in 5.6+)

Initial Configuration File

Capacity settings

- Other capacity options:
  - max_connections
  - table_definition_cache
  - table_open_cache
  - table_open_cache_instances

Initial Configuration File

Capacity settings - innodb_buffer_pool_size

- MySQL Reference Manual (about innodb_buffer_pool_size):
  "On a dedicated database server, you might set the buffer pool size to 80% of the machine's physical memory size."
  - What if you have 1TB of memory? Do you still want to reserve 20% for other uses?
- Instead:
  - How much memory does the host have?
  - Subtract memory required by OS and other processes
  - Subtract memory required by MySQL other then the InnoDB buffer pool
  - Choose minimum of this and the size of the "working data set"

innodb_buffer_pool_size – Side Note

- In MySQL 5.7, innodb_buffer_pool_size can be changed dynamically:

```
mysql> SELECT (POOL_SIZE*@@global.innodb_page_size/1024/1024) AS 'PoolSize (MB)'
         FROM information_schema.INNODB_BUFFER_POOL_STATS\G
*************************** 1. row ***************************
PoolSize (MB): 127.98437500

mysql> SELECT VARIABLE_VALUE
         FROM performance_schema.global_status
        WHERE VARIABLE_NAME = 'Innodb_buffer_pool_resize_status'\G
*************************** 1. row ***************************
VARIABLE_VALUE: Completed resizing buffer pool at 160912 18:49:50.

mysql> SELECT (POOL_SIZE*@@global.innodb_page_size/1024/1024) AS 'PoolSize (MB)'
         FROM information_schema.INNODB_BUFFER_POOL_STATS\G
*************************** 1. row ***************************
PoolSize (MB): 4095.98437500
```

Initial Configuration File

Capacity settings - innodb_buffer_pool_instances

- Specifies how many instances to split the buffer pool into - can reduce contention for concurrent workload
- Rule of thumb:
  - innodb_buffer_pool_size <= 1G: 1 instance
  - innodb_buffer_pool_size <= 16G: 8 instances
  - innodb_buffer_pool_size > 16G: 2G per instance
  - Max allowed value: 64
- But test with your workload!
- Requires restart

Initial Configuration File

Capacity settings – InnoDB redo log

- Total redo log size defined by two options:
  - innodb_log_file_size
  - innodb_log_files_in_group
- Total size = innodb_log_file_size * innodb_log_files_in_group
  - Max supported total redo log size:
    - MySQL 5.5 and earlier: just below 4G
    - MySQL 5.6 and later: just below 512G
- Should be large enough to avoid “excessive” checkpointing

Initial Configuration File

Capacity settings – InnoDB redo log – Is it large enough?

- Get current Log sequence number (LSN) and last checkpoint:
  - SHOW ENGINE INNOD STATUS:

```
---
LOG
---
Log sequence number 602763740
Log flushed up to   602763740
Pages flushed up to 584668961
Last checkpoint at  555157885
```

  - INNODB_METRICS:

```
mysql> SELECT NAME, COUNT
         FROM information_schema.INNODB_METRICS
        WHERE NAME IN ('log_lsn_current', 'log_lsn_last_checkpoint');
+--------------------------+-----------+
| NAME                     | COUNT     |
+--------------------------+-----------+
| log_lsn_last_checkpoint | 555157885 |
| log_lsn_current          | 602763740 |
+--------------------------+-----------+
```

Initial Configuration File

Capacity settings – InnoDB redo log – Is it large enough?

- Get current Log sequence number (LSN) and last checkpoint:
  - Sys schema metrics view:

```
mysql> SELECT *
         FROM sys.metrics
        WHERE Variable_name IN ('log_lsn_current', 'log_lsn_last_checkpoint');
+-------------------------+----------------+---------------------------+---------+
| Variable_name           | Variable_value | Type                      | Enabled |
+-------------------------+----------------+---------------------------+---------+
| log_lsn_current         | 602763740      | InnoDB Metrics - recovery | YES     |
| log_lsn_last_checkpoint | 555157885      | InnoDB Metrics - recovery | YES     |
+-------------------------+----------------+---------------------------+---------+
```

Initial Configuration File

Capacity settings – InnoDB redo log – Is it large enough?

- Calculate the amount of used redo log:
  - Used log = log_lsn_current – log_lsn_last_checkpoint
    = 602763740 - 555157885
    = 47605855 (bytes)
- Compare to total size:
  - Used % = (Used log / Total log) * 100
    = (47605855 / (innodb_log_file_size * innodb_log_files_in_group)) * 100
    = (47605855 / 100663296) * 100
    = 47.29 %

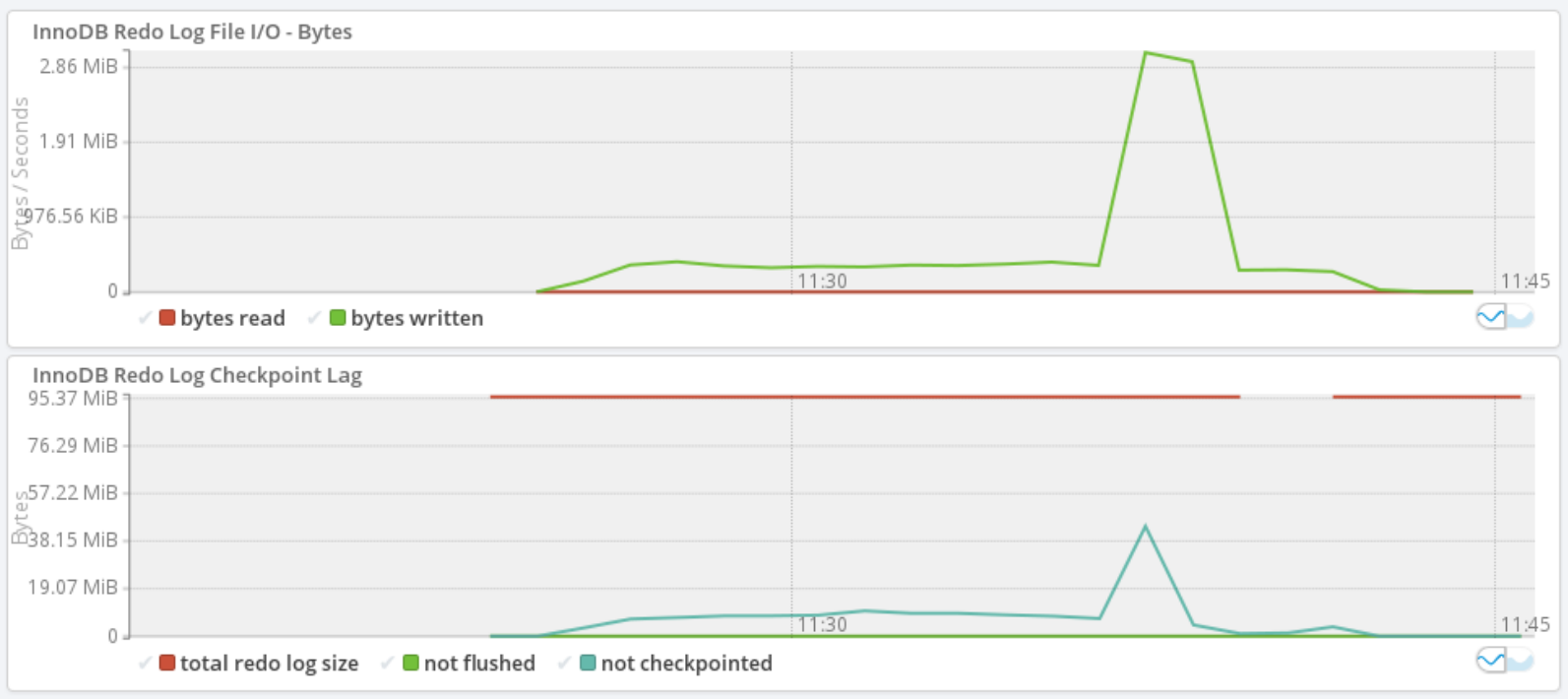
Initial Configuration File

Capacity settings – InnoDB redo log – Is it large enough?

- If the usage reaches 75% an asynchronous flush is triggered
  - I/O can be so intensive that all other work stalls
  - Main InnoDB thread may be in the state: flushing buffer pool pages
- Ensure you have enough head room to handle peak loads
  - For example aim at using at most 60% or 70% of the redo log
- Important that your monitoring solution monitors the redo log usage
- Improvements to the flushing algorithm and new options to control the I/O have been added in newer MySQL versions

Initial Configuration File

Capacity settings – InnoDB redo log – Is it large enough?



Initial Configuration File

Capacity settings - innodb_thread_concurrency

- Defines how many queries InnoDB will allow to execute concurrently
- 0 means unlimited and is often a good value if the number of vCPUs is the limiting factor
- Otherwise benchmark to find your optimal value – often good values are:
  - MySQL 5.5: Between 8 and 24
  - MySQL 5.6: Between 16 and 64
  - MySQL 5.7: Between 32 and 128
- Do not mix up with thread_concurrency (removed in 5.7!)

Initial Configuration File

InnoDB Undo Log

- innodb_undo_tablespaces can only be set before initializing the datadir!
- If set creates undo log tablespaces outside the system tablespace (ibdata1)
- Advantages:
  - Keeps the system tablespace smaller
  - Can keep the undo log on faster disks
  - In MySQL 5.7 the undo log tablespaces can be truncated
- Each undo tablespace file is 10M initially
- Maximum of 95 undo tablespaces in 5.7

Initial Configuration File

Other capacity options

- max_connections – be careful setting this too large as each connection requires memory
- table_definition_cache – ensure all tables can be in the cache. If you expect 4000 tables, set table_definition_cache > 4000
- table_open_cache – each table can be open more than once
- table_open_cache_instances - Typically a good value is 16

Initial Configuration File

Other capacity options

- max_connections, table_definition_cache, and table_open_cache determines the default value for several Performance Schema options
  - Small instance:
    - max_connections <= 151 && table_definition_cache <= 400 && table_open_cache <= 2000
  - Medium instance:
    - max_connections <= 302 && table_definition_cache <= 800 && table_open_cache <= 4000
  - Otherwise large instance

Initial Configuration File

Other capacity options – MySQL 5.6 Performance Schema Memory Usage

- max_connections = 151; table_open_cache = 200

table_definition_cache	Instance Size	Total Performance Schema Memory Usage
400	Small	52.6 MB
401	Medium	90.7 MB
800	Medium	98.4 MB
801	Large	400 MB

- MySQL 5.7 has more dynamic memory usage

Oracle MySQL Cloud Service

Simple, Automated, Integrated & Enterprise-Ready

- The #1 Open Source Database in Oracle Cloud
- Only public cloud integrating MySQL Enterprise Edition
- Rapidly, securely and cost-effectively develop & deploy modern MySQL-based applications

Cloud.oracle.com/mysql: Learn more & Free Trial!

ORACLE

Buffers And Caches

Several buffers and caches available

- Query Cache - query_cache_size
- Per query buffers:
  - join_buffer_size
  - read_buffer_size
  - read_rnd_buffer_size
  - sort_buffer_size
- And more

Buffers And Caches

Several buffers and caches available

- Surely enabling caches is a good thing?
- Surely the larger caches and buffers the better?

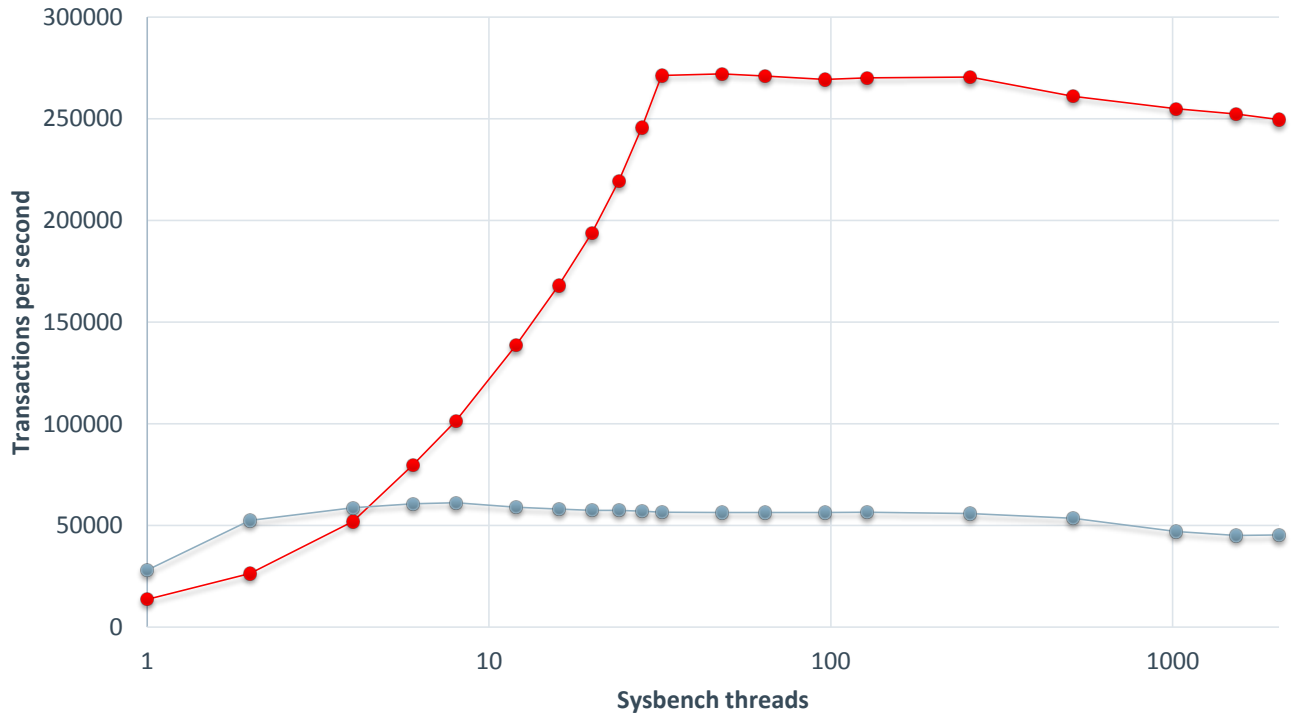
Buffers And Caches

Several buffers and caches available

- Surely enabling caches is a good thing?
- Surely the larger caches and buffers the better?
- **No!**

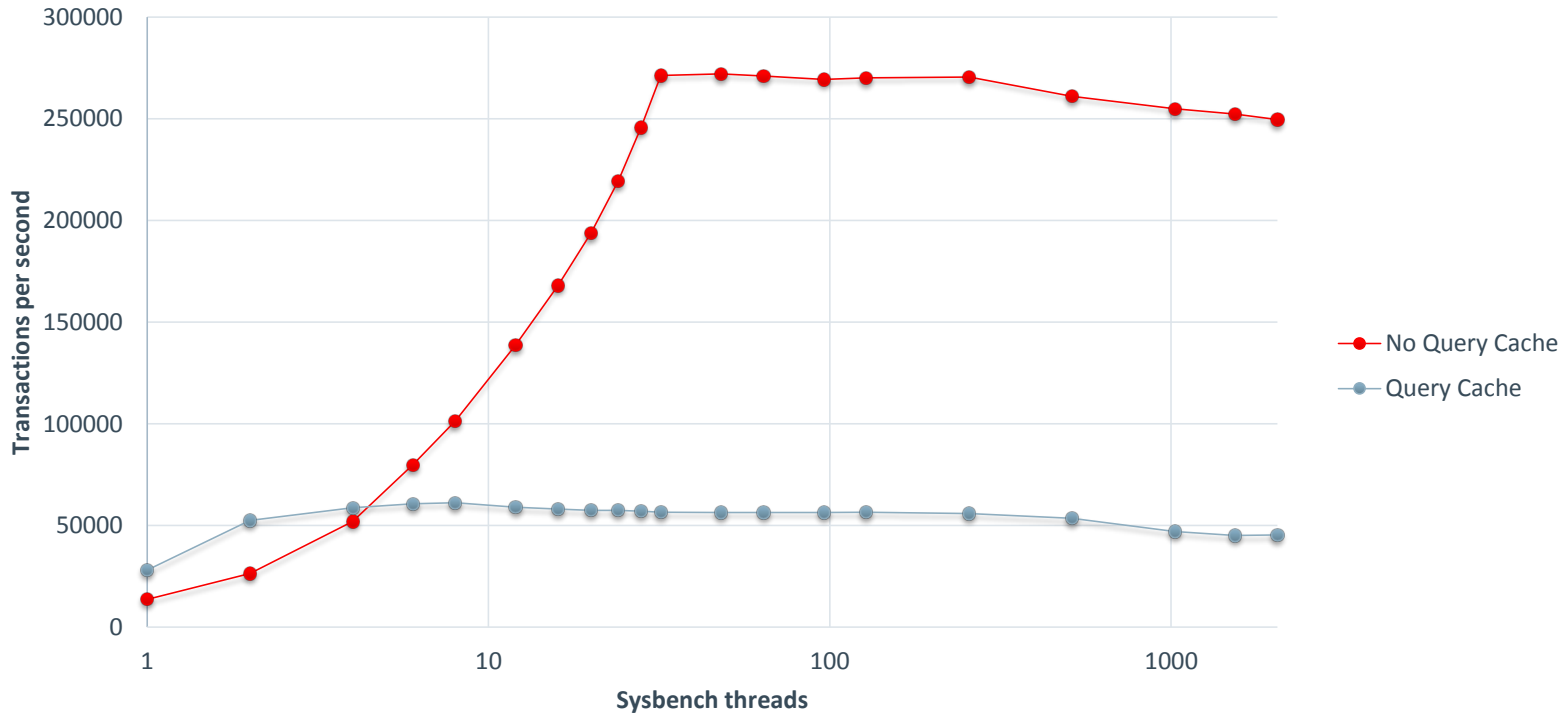
Buffers And Caches

Query Cache



Buffers And Caches

Query Cache



Buffers And Caches

Query Cache

- It is rarely good to enable the Query Cache
  - Guarded by a single mutex
- Most workloads is best left with the Query Cache disabled (the default):
  - query_cache_type = 0
- If you think your workload benefits from the Query Cache, test it
  - The more writes, the less benefit
  - The more data fitting into the buffer pool, the less benefit
  - The more complex queries and the larger scans, the more benefit
- Often other caching solutions are a better option.

Buffers And Caches

join_buffer_size

- In MySQL 5.5 and earlier: only used for plain index scans, range index scans, and joins that do not use indexes
  - No reason to have it larger than the size of each matching row
- In MySQL 5.6+ also used for Batched Key Access (BKA)
  - Queries using BKA can benefit from a larger join buffer
- Is the minimum size allocated!
- Usually a small global (default for new connections) value is best: 32k-256k
- Increase for a session as needed

Buffers And Caches

sort_buffer_size

- Like for join_buffer_size a small global value is usually best: 32k-256k
- Can monitor using the global status variable Sort_merge_passes:

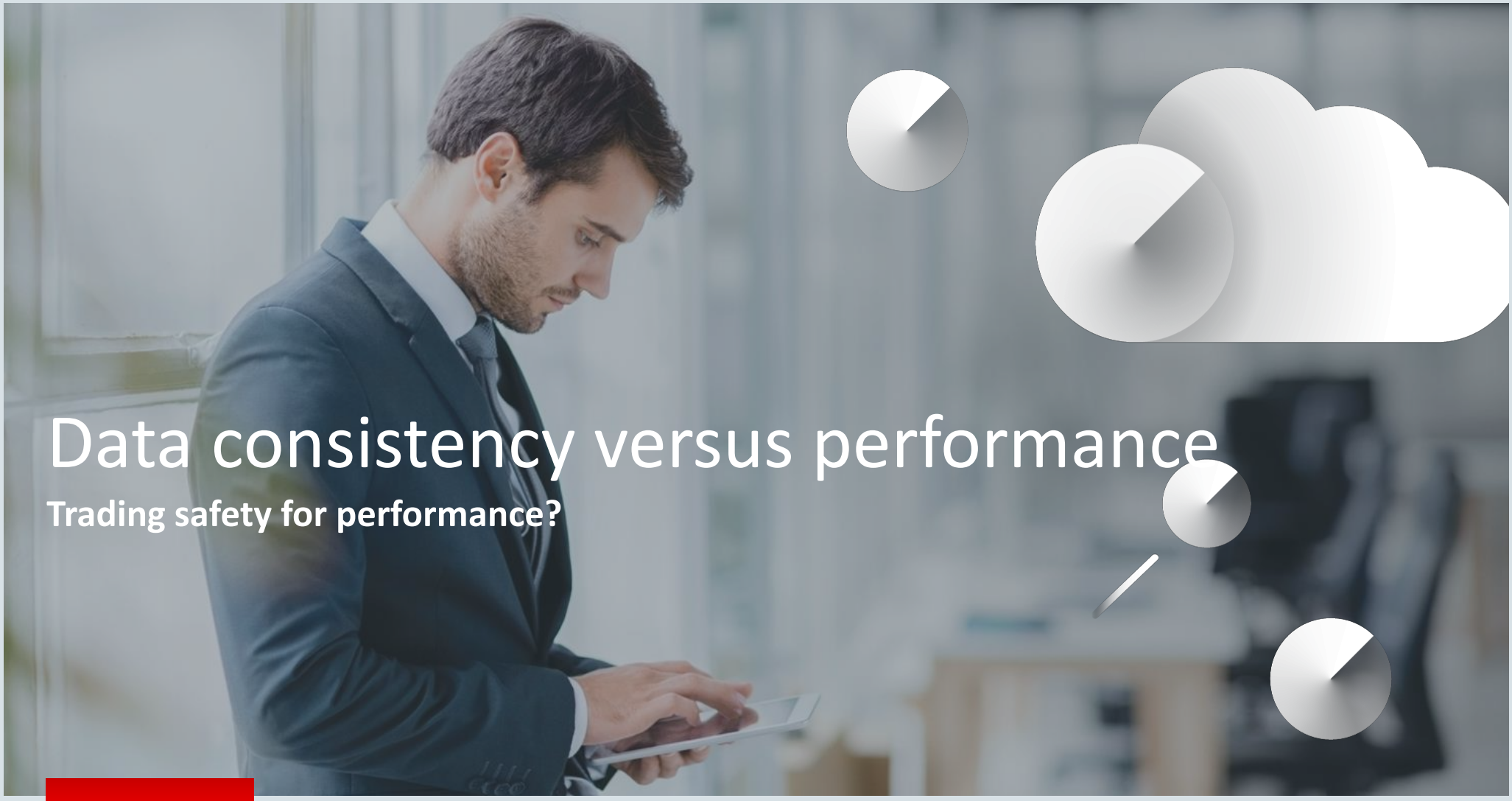
```
mysql> SHOW GLOBAL STATUS LIKE 'Sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.00 sec)
```

- Aim at a few Sort_merge_passes per second on a busy server
- Increase for a session as needed

Buffers And Caches

Why is it important with small global values?

- Some buffers are allocated in full each time they are used
- One query may use several buffers, so large buffers can cause significant memory usage
- Memory allocations are relatively expensive
- For example Linux glibc malloc changes memory allocation algorithm when crossing threshold (typical 256k or 512k).
  - Algorithm for larger allocations can be 40 times slower than for smaller allocation



Data consistency versus performance

Trading safety for performance?

Data Consistency Versus Performance

- The two options most commonly considered:
  - innodb_flush_log_at_trx_commit
  - sync_binlog
- There are others as well

Data Consistency Versus Performance

innodb_flush_log_at_trx_commit

- Possible values in order of data safety:
  - 1: is theoretically the slowest, but with fast SSD it may be around as fast as 2 and 0
  - 2: flushed every second (innodb_flush_log_at_timeout in 5.6+)
    - Transactions may be lost if the OS crashes
  - 0: MySQL never fsyncs
    - Transactions may be lost if the OS crashes
- Defaults to 1 – flush the redo log after each commit
  - Required for D in ACID
  - For this reason it is the recommended value

Data Consistency Versus Performance

innodb_flush_log_at_trx_commit

- What if innodb_flush_log_at_trx_commit = 1 is too slow and you want the D in ACID?
  - Make sure the redo logs are on separate disks
    - Do not be tempted to have the individual redo logs on different disks
  - Consider SSD particularly if you have a high commit rate
  - Battery-backed disk cache also makes flushes cheaper

Data Consistency Versus Performance

sync_binlog

- Specifies how many transactions between each flush of the binary log
  - 0: when rotating the binary log or when the OS decides
  - 1: at every transaction commit – this is the safest
  - N: every N commits
- Default value:
  - MySQL 5.6 and earlier: 0
  - MySQL 5.7 and later: 1
- MySQL 5.6 and later support group commit for InnoDB giving less overhead of sync_binlog = 1

Data Consistency Versus Performance

sync_binlog

- sync_binlog != 1 means that slaves most likely will have to be rebuild if the master crashes
- But sync_binlog = 0 must be the best performance wise, right?

Data Consistency Versus Performance

sync_binlog

- sync_binlog != 1 means that slaves most likely will have to be rebuild if the master crashes
- But sync_binlog = 0 must be the best performance wise, right?
  - By default: max_binlog_size = 1G
  - 1G is not a lot of memory nowadays
  - So the OS may buffer the entire binary log
  - When rotating the binary log, up to 1G will be flushed to disk
    - Stalls all other commits until the binary log rotation is done
- So in short: sync_binlog = 0 may give the best throughput, but sync_binlog = 1 may give the most predictable performance

Query Tuning

1. Determine the query to optimize
2. Optimize the query

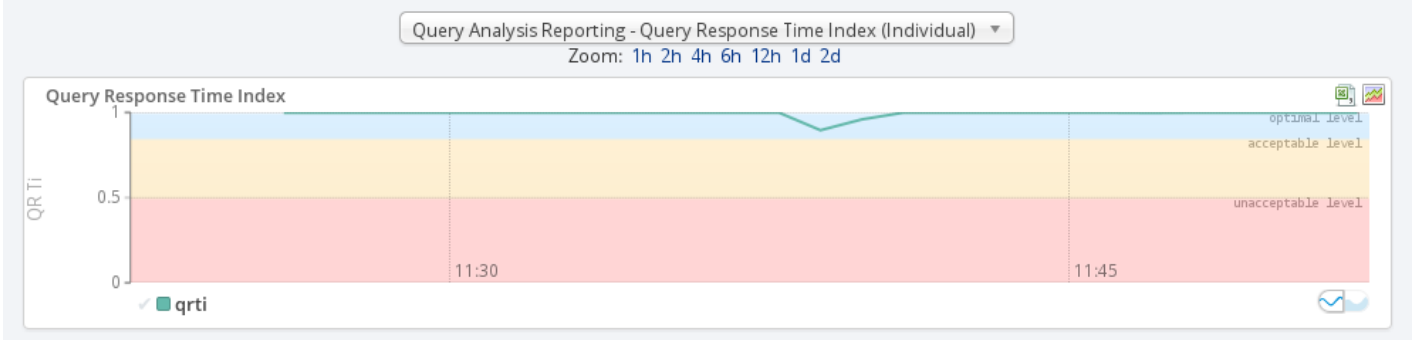
Query Tuning

Finding queries to optimize

- Application metrics
- Manually executing query and determining it is too slow or uses too many resources
- Slow Query Log
- MySQL Enterprise Monitor Query Analyzer
- performance_schema.events_statements_summary_by_digest
- sys.statements_with_runtimes_in_95th_percentile

Query Tuning

Finding queries to optimize - MySQL Enterprise Monitor Query Analyzer



Show 10 entries | Export data options... | Showing 1 to 10 of 83 entries | First Previous 1 2 3 4 5 Next Last

Query	Counts			QRTi	Latency (hh:mm:ss.ms)				Rows	
	Exec	Err	Warn		Total	Max	Avg	Avg History	Total	Avg
INSERT INTO `salaries` VALUES (...) /* , ... */	114	0	0	0.00	50.061	4.157	0.439		2,844,047	24,9
INSERT INTO `dept_emp` VALUES (...) /* , ... */	14	0	0	0.00	10.042	2.458	0.717		331,603	23,6
INSERT INTO `titles` VALUES (...) /* , ... */	21	0	0	0.00	10.765	1.735	0.513		443,308	21,1
INSERT INTO `employees` VALUES (...) /* , ... */	17	0	0	0.50	6.042	1.680	0.355		300,024	17,6
COMMIT	114	0	0	1.00	7.058	0.993	0.062		0	
DROP SCHEMA IF EXISTS `employees`	2	0	1	0.50	0.225	0.849	0.112		6	
CREATE TABLE `dept_mana...mp_no`, `dept_no`))	1	0	0	1.00	0.013	0.220	0.013		0	
SELECT COUNT (?) AS `cn...est` GROUP BY `avg_us`	5	0	0	0.90	0.345	0.217	0.069		40	

Query Tuning

Finding queries to optimize – Performance Schema Digest Summary

```
mysql> SELECT LEFT(DIGEST_TEXT, 64) AS DIGEST_TEXT, COUNT_STAR, SUM_TIMER_WAIT, MAX_TIMER_WAIT
         FROM performance_schema.events_statements_summary_by_digest
        ORDER BY MAX_TIMER_WAIT DESC
        LIMIT 10;
+------------------------------------------------------------------+------------+-----------------+----------------+
| DIGEST_TEXT                                                      | COUNT_STAR | SUM_TIMER_WAIT  | MAX_TIMER_WAIT |
+------------------------------------------------------------------+------------+-----------------+----------------+
| INSERT INTO `salaries` VALUES (...) /* , ... */                  |        342 | 159811231808000 |  4156961573000 |
| INSERT INTO `dept_emp` VALUES (...) /* , ... */                  |         42 |  31561264335000 |  2458392698000 |
| INSERT INTO `titles` VALUES (...) /* , ... */                    |         63 |  35738435708000 |  1735350241000 |
| INSERT INTO `employees` VALUES (...) /* , ... */                 |         51 |  18004605187000 |  1679817477000 |
| INSERT INTO `sbtest` ( `k` , `c` , `pad` ) VALUES (...) /* , ... |         10 |   5241286782000 |  1247361451000 |
| COMMIT                                                           |        342 |  31984662051000 |   992714081000 |
| DROP SCHEMA IF EXISTS `employees`                                |          6 |   1252459420000 |   848771265000 |
| CREATE TABLE `sbtest` ( `id` INTEGER UNSIGNED NOT NULL AUTO_INCR |          1 |    565468324000 |   565468324000 |
| CREATE TABLE `dept_manager` ( `dept_no` CHARACTER (?) NOT NULL , |          3 |    355874700000 |   220491035000 |
| SELECT COUNT (?) AS `cnt` , `round` ( ( `performance_schema` . ` |          6 |    386062170000 |   217206520000 |
+------------------------------------------------------------------+------------+-----------------+----------------+
10 rows in set (0.00 sec)
```

Query Tuning

Finding queries to optimize – 95th percentile

```
mysql> SELECT query, exec_count, total_latency, max_latency
         FROM sys.statements_with_runtimes_in_95th_percentile;
+-----------------------------------------------------------------------+------------+---------------+-------------+
| query                                                                 | exec_count | total_latency | max_latency |
+-----------------------------------------------------------------------+------------+---------------+-------------+
| INSERT INTO `dept_emp` VALUES (...) /* , ... */                       |         42 | 31.56 s       | 2.46 s      |
| INSERT INTO `titles` VALUES (...) /* , ... */                         |         63 | 35.74 s       | 1.74 s      |
| CREATE TABLE `sbtest` ( `id` I ...  ( `id` ) ) ENGINE = `innodb`      |          1 | 565.47 ms     | 565.47 ms   |
| INSERT INTO `sbtest` ( `k` , ` ... d` ) VALUES (...) /* , ... */      |         10 | 5.24 s        | 1.25 s      |
| INSERT INTO `salaries` VALUES (...) /* , ... */                       |        342 | 2.66 m        | 4.16 s      |
| INSERT INTO `employees` VALUES (...) /* , ... */                      |         51 | 18.00 s       | 1.68 s      |
| DROP SCHEMA IF EXISTS `employees`                                     |          6 | 1.25 s        | 848.77 ms   |
| LOAD DATA LOCAL INFILE ? INTO TABLE `City` CHARACTER SET `utf8`       |          1 | 167.90 ms     | 167.90 ms   |
+-----------------------------------------------------------------------+------------+---------------+-------------+
8 rows in set (0.01 sec)
```

Query Tuning

Optimize the query

- EXPLAIN is your best friend:
  - Shows information about the query plan
  - Tells where indexes can be used and not used
- In MySQL 5.6 and earlier use EXPLAIN EXTENDED/EXPLAIN PARTITIONS
- In MySQL 5.6+ use FORMAT=JSON for more information

Query Tuning

Optimize the query – Example 1

```
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`)
) ENGINE=InnoDB AUTO_INCREMENT=4080 DEFAULT CHARSET=latin1

CREATE TABLE `Country` (
  `Code` char(3) NOT NULL DEFAULT '',
  `Name` char(52) NOT NULL DEFAULT '',
  `Continent` enum('Asia','Europe','North America',…) NOT NULL DEFAULT 'Asia',
  `Region` char(26) NOT NULL DEFAULT '',
  …
  PRIMARY KEY (`Code`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Query Tuning

Optimize the query – Example 1

```
mysql> SELECT sys.ps_thread_id(NULL); -- Gets the Performance Schema THREAD_ID for the current connection
+------------------------+
| sys.ps_thread_id(NULL) |
+------------------------+
|                  11419 |
+------------------------+
1 row in set (0.00 sec)

mysql> SELECT ci.Name, co.Name AS Country, ci.Population
         FROM Country co
              STRAIGHT_JOIN City ci ON ci.CountryCode = co.Code
        WHERE co.Continent = 'North America'
        ORDER BY ci.Population DESC
        LIMIT 5;
+-------------------+---------------+------------+
| Name              | Country       | Population |
+-------------------+---------------+------------+
| Ciudad de México  | Mexico        |    8591309 |
| New York          | United States |    8008278 |
| Los Angeles       | United States |    3694820 |
| Chicago           | United States |    2896016 |
| La Habana         | Cuba          |    2256000 |
+-------------------+---------------+------------+
5 rows in set (0.05 sec)
```

Query Tuning

Optimize the query – Example 1

```
mysql> SELECT * FROM performance_schema.events_statements_current WHERE THREAD_ID = 11419\G
/* Some columns removed */
              THREAD_ID: 11419
TIMER_WAIT: 53361516000
              LOCK_TIME: 392000000
               SQL_TEXT: SELECT ci.Name, co.Name AS Country, ... DESC LIMIT 5
                 DIGEST: d27cbfe06a40d745a5c498f7f5824529
            DIGEST_TEXT: SELECT `ci` . `Name` , `co` . `Name` AS `Country` , ... DESC LIMIT ?
                 ERRORS: 0
               WARNINGS: 0
              ROWS_SENT: 5
          ROWS_EXAMINED: 4904
CREATED_TMP_DISK_TABLES: 0
     CREATED_TMP_TABLES: 1
       SELECT_FULL_JOIN: 1
 SELECT_FULL_RANGE_JOIN: 0
           SELECT_RANGE: 0
     SELECT_RANGE_CHECK: 0
            SELECT_SCAN: 1
      SORT_MERGE_PASSES: 0
             SORT_RANGE: 0
              SORT_ROWS: 5
              SORT_SCAN: 1
          NO_INDEX_USED: 1
     NO_GOOD_INDEX_USED: 0
```

Query Tuning

Optimize the query – Example 1

```
mysql> SELECT * FROM sys.session WHERE thd_id = 11419\G
/* Some columns removed */
*************************** 1. row ***************************
                thd_id: 11419
               conn_id: 11394
                  user: root@localhost
          lock_latency: 392.00 us
         rows_examined: 4904
             rows_sent: 5
         rows_affected: 0
            tmp_tables: 1
       tmp_disk_tables: 0
             full_scan: YES
        last_statement: SELECT ci.Name, co.Name AS Cou ...  BY ci.Population DESC LIMIT 5
last_statement_latency: 53.36 ms
        current_memory: 72.13 KiB
             last_wait: NULL
     last_wait_latency: NULL
                source: NULL
           trx_latency: NULL
             trx_state: NULL
        trx_autocommit: NULL
                   pid: 31971
          program_name: mysql
```

Query Tuning

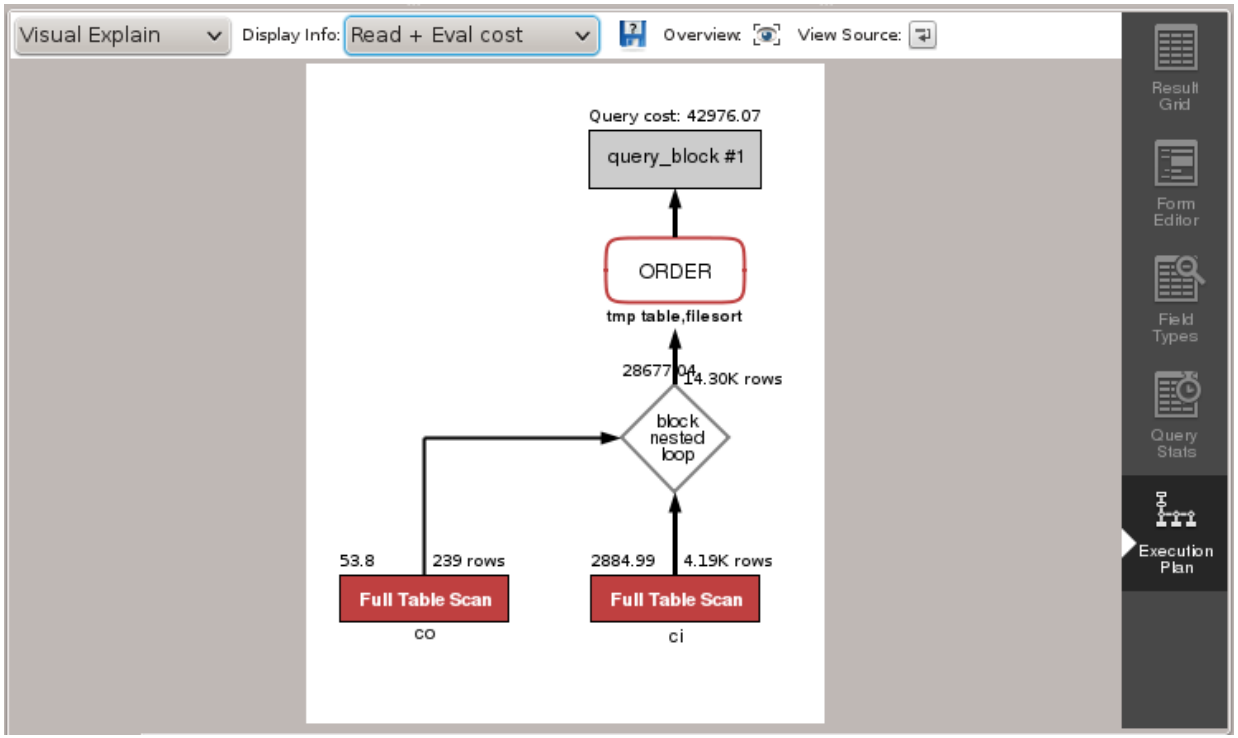
Optimize the query – Example 1

```
mysql> EXPLAIN SELECT ci.Name, co.Name AS Country, ci.Population
                FROM Country co
                     STRAIGHT_JOIN City ci ON ci.CountryCode = co.Code
               WHERE co.Continent = 'North America'
               ORDER BY ci.Population DESC
               LIMIT 5;
+----+-------------+-------+------+---------------+------+------+-----------------------------------------------------+
| id | select_type | table | type | possible_keys | key  | rows | Extra                                               |
+----+-------------+-------+------+---------------+------+------+-----------------------------------------------------+
|  1 | SIMPLE      | co    | ALL  | PRIMARY       | NULL |  239 | Using where; Using temporary; Using filesort        |
|  1 | SIMPLE      | ci    | ALL  | NULL          | NULL | 4188 | Using where; Using join buffer (Block Nested Loop)  |
+----+-------------+-------+------+---------------+------+------+-----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `world`.`ci`.`Name` AS `Name`,`world`.`co`.`Name` AS `Country`,`world`.`ci`.`Population`
AS `Population` from `world`.`Country` `co` straight_join `world`.`City` `ci` where ((`world`.`ci`.`CountryCode` =
`world`.`co`.`Code`) and (`world`.`co`.`Continent` = 'North America')) order by `world`.`ci`.`Population` desc limit 5
1 row in set (0.00 sec)
```

Query Tuning

Optimize the query – Example 1



Query Tuning

Optimize the query – Example 1

- The EXPLAIN output shows:
  - No indexes used for joins (full table scans)
  - No index used for WHERE clause
  - No index used for ORDER BY
- First consider an index useful for the JOIN:
  - FROM Country co STRAIGHT_JOIN City ci ON ci.CountryCode = co.Code
  - Add an index in the City.CountryCode column:
    - ALTER TABLE City ADD INDEX (`CountryCode`);

Query Tuning

Optimize the query – Example 1

- New EXPLAIN:

```
+----+-------+------+---------------+-------------+---------+---------------+------+-----------------------------+
| id | table | type | possible_keys | key         | key_len | ref           | rows | Extra                       |
+----+-------+------+---------------+-------------+---------+---------------+------+-----------------------------+
|  1 | co    | ALL  | PRIMARY       | NULL        | NULL    | NULL          |  239 | Using where; ... (same) ... |
|  1 | ci    | ref  | CountryCode   | CountryCode | 3       | world.co.Code |   18 | NULL                        |
+----+-------+------+---------------+-------------+---------+---------------+------+-----------------------------+
```

- Much better, but still no index used for the Country table
  - Try add:
    - An index on the Country.Continent column for the WHERE clause
    - An index on the City.Population column for the ORDER BY
  - ALTER TABLE Country ADD INDEX (Continent);
  - ALTER TABLE City ADD INDEX (Population);

Query Tuning

Optimize the query – Example 1

- New EXPLAIN:

```
+-------+------+-------------------+-------------+---------+---------------+------+---------------------------------+
| table | type | possible_keys     | key         | key_len | ref           | rows | Extra                           |
+-------+------+-------------------+-------------+---------+---------------+------+---------------------------------+
| co    | ref  | PRIMARY,Continent | Continent   | 1       | const         |   37 | Using temporary; Using filesort |
| ci    | ref  | CountryCode       | CountryCode | 3       | world.co.Code |   18 | NULL                            |
+-------+------+-------------------+-------------+---------+---------------+------+---------------------------------+
```

- Good for the Country table
- But no help for the ORDER BY
  - MySQL can only use one index for each table
  - Adding (CountryCode, Population) will not help as City table is the second table
    - ALTER TABLE City ADD INDEX (CountryCode, Population);

Query Tuning

Optimize the query – Example 1

- New EXPLAIN:

```
+-------+------+---------------------------+-------------+---------------+------+---------------------------------+
| table | type | possible_keys             | key         | ref           | rows | Extra                           |
+-------+------+---------------------------+-------------+---------------+------+---------------------------------+
| co    | ref  | PRIMARY,Continent         | Continent   | const         |   37 | Using temporary; Using filesort |
| ci    | ref  | CountryCode,CountryCode_2 | CountryCode | world.co.Code |   18 | NULL                            |
+-------+------+---------------------------+-------------+---------------+------+---------------------------------+
```

- What if the JOIN order is reversed?

```
SELECT ci.Name, co.Name AS Country, ci.Population
  FROM City ci
       STRAIGHT_JOIN Country co ON ci.CountryCode = co.Code
 WHERE co.Continent = 'North America'
 ORDER BY ci.Population ASC
 LIMIT 5;
```

Query Tuning

Optimize the query – Example 1

- New EXPLAIN:

```
mysql> EXPLAIN SELECT ci.Name, co.Name AS Country, ci.Population
                 FROM City ci
                      STRAIGHT_JOIN Country co ON ci.CountryCode = co.Code
                WHERE co.Continent = 'North America'
                ORDER BY ci.Population ASC
                LIMIT 5;
+-------+--------+---------------------------+---------+---------+----------------------+------+----------------+
| table | type   | possible_keys             | key     | key_len | ref                  | rows | Extra          |
+-------+--------+---------------------------+---------+---------+----------------------+------+----------------+
| ci    | ALL    | CountryCode,CountryCode_2 | NULL    | NULL    | NULL                 | 4188 | Using filesort |
| co    | eq_ref | PRIMARY,Continent         | PRIMARY | 3       | world.ci.CountryCode |    1 | Using where    |
+-------+--------+---------------------------+---------+---------+----------------------+------+----------------+
```

- So eliminated the internal temporary table (but not related to the indexes)

Query Tuning

Optimize the query – Example 1

- Now the City table looks like:

```
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`),
  KEY `CountryCode_2` (`CountryCode`,`Population`),
  KEY `Population` (`Population`)
) ENGINE=InnoDB AUTO_INCREMENT=4080 DEFAULT CHARSET=latin1
```

- Note: two indexes starting with the CountryCode table
  - (CountryCode, Population) can be used instead of (CountryCode) as they have same left prefix.
  - Best to drop one of them

Query Tuning

Optimize the query – Example 1

- Side note – you can check for unused indexes:

```
SELECT OBJECT_SCHEMA,
       OBJECT_NAME,
       INDEX_NAME
  FROM performance_schema.table_io_waits_summary_by_index_usage
 WHERE INDEX_NAME IS NOT NULL
       AND COUNT_STAR = 0
       AND OBJECT_SCHEMA != 'mysql'
       AND INDEX_NAME != 'PRIMARY'
 ORDER BY OBJECT_SCHEMA, OBJECT_NAME;
```

- Or:

```
mysql> SELECT * FROM sys.schema_unused_indexes WHERE object_schema = 'world';
+---------------+-----------------+---------------+
| object_schema | object_name     | index_name    |
+---------------+-----------------+---------------+
| world         | City            | CountryCode   |
| world         | City            | CountryCode_2 |
| world         | City            | Population    |
| world         | CountryLanguage | CountryCode   |
+---------------+-----------------+---------------+
```

Query Tuning

Optimize the query – Example 2

- Employees sample database – find all employees that has been in more than one department:

```
SELECT *
  FROM employees
 WHERE emp_no IN (SELECT emp_no FROM dept_emp GROUP BY emp_no HAVING COUNT(*) > 1);
```

- MySQL 5.5: Killed it after 4 hours

Query Tuning

Optimize the query – Example 2

- Employees sample database – find all employees that has been in more than one department:

```
SELECT *
  FROM employees
 WHERE emp_no IN (SELECT emp_no FROM dept_emp GROUP BY emp_no HAVING COUNT(*) > 1);
```

- MySQL 5.5: Killed it after 4 hours
- EXPLAIN – note the dependent subquery:

```
+----+--------------------+-----------+-------+---------------+--------+---------+------+--------+-------------+
| id | select_type        | table     | type  | possible_keys | key    | key_len | ref  | rows   | Extra       |
+----+--------------------+-----------+-------+---------------+--------+---------+------+--------+-------------+
|  1 | PRIMARY            | employees | ALL   | NULL          | NULL   | NULL    | NULL | 300030 | Using where |
|  2 | DEPENDENT SUBQUERY | dept_emp  | index | NULL          | emp_no | 4       | NULL |      1 | Using index |
+----+--------------------+-----------+-------+---------------+--------+---------+------+--------+-------------+
```

Query Tuning

Optimize the query – Example 2

- Potential rewrites:
  - Use derived table: 0.15 second

```
SELECT employees.*
  FROM (SELECT emp_no FROM dept_emp GROUP BY emp_no HAVING COUNT(*) > 1) tmp
       INNER JOIN employees USING (emp_no);
```

  - Regular JOIN: 0.4 second

```
SELECT e.emp_no, e.birth_date, e.first_name, e.last_name, e.gender, e.hire_date
  FROM employees e
       INNER JOIN dept_emp USING (emp_no)
 GROUP BY emp_no
HAVING COUNT(*) > 1;
```

Query Tuning

Optimize the query – Example 2

- Potential rewrites:
  - Use derived table: 0.15 second

```
SELECT employees.*
  FROM (SELECT emp_no FROM dept_emp GROUP BY emp_no HAVING COUNT(*) > 1) tmp
       INNER JOIN employees USING (emp_no);
```

  - Regular JOIN: 0.4 second

```
SELECT e.emp_no, e.birth_date, e.first_name, e.last_name, e.gender, e.hire_date
  FROM employees e
       INNER JOIN dept_emp USING (emp_no)
 GROUP BY emp_no
HAVING COUNT(*) > 1;
```

- Or upgrade to 5.6+ (can materialize the subquery)
  - Original query: 0.48 second

Query Tuning

Optimize the query – Example 3

- Schema:

```
CREATE TABLE `t1` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1

CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- Query:

```
mysql> SELECT COUNT(*) FROM t1 INNER JOIN t2 USING (id);
+----------+
| COUNT(*) |
+----------+
|    50000 |
+----------+
1 row in set (2 min 3.36 sec)
```

Query Tuning

Optimize the query – Example 3

- EXPLAIN:

```
+-------+------+---------------+------+---------+------+-------+-------------------------------------------------+
| table | type | possible_keys | key  | key_len | ref  | rows  | Extra                                           |
+-------+------+---------------+------+---------+------+-------+-------------------------------------------------+
| t1    | ALL  | NULL          | NULL | NULL    | NULL | 50283 | NULL                                            |
| t2    | ALL  | NULL          | NULL | NULL    | NULL | 50283 | Using where; Using join buffer (Block Nested Loop) |
+-------+------+---------------+------+---------+------+-------+-------------------------------------------------+
```

- Obviously best choice is to add indexes, but assume that is not possible

Query Tuning

Optimize the query – Example 3

- In 5.6+ rewrite using subqueries
  - In 5.7 it is necessary to switch off the derived_merge optimizer switch:

```
mysql> SET optimizer_switch='derived_merge=off';
Query OK, 0 rows affected (0.00 sec)
```

  - The new query:

```
mysql> SELECT COUNT(*) FROM (SELECT * FROM t1) AS t1 INNER JOIN (SELECT * FROM t2) AS t2 USING (id);
+----------+
| COUNT(*) |
+----------+
|    50000 |
+----------+
1 row in set (0.12 sec)
```

- A factor 1000 speedup! Why?

Query Tuning

Optimize the query – Example 3

- New EXPLAIN:

```
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
| id | select_type | table      | type | possible_keys | key         | key_len | ref   | rows  | Extra       |
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | Using where |
|  1 | PRIMARY     | <derived3> | ref  | <auto_key0>   | <auto_key0> | 5       | t1.id |    10 | NULL        |
|  3 | DERIVED     | t2         | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | NULL        |
|  2 | DERIVED     | t1         | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | NULL        |
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
```

- The optimizer added an auto key!

Query Tuning

Optimize the query – Example 3

- New EXPLAIN:

```
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
| id | select_type | table      | type | possible_keys | key         | key_len | ref   | rows  | Extra       |
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | Using where |
|  1 | PRIMARY     | <derived3> | ref  | <auto_key0>   | <auto_key0> | 5       | t1.id |    10 | NULL        |
|  3 | DERIVED     | t2         | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | NULL        |
|  2 | DERIVED     | t1         | ALL  | NULL          | NULL        | NULL    | NULL  | 50283 | NULL        |
+----+-------------+------------+------+---------------+-------------+---------+-------+-------+-------------+
```

- The optimizer added an auto key!
- Far fetched? A real world use case is for the schema_table_statistics view in the sys schema.

Consider the Whole Stack

The issue may not be in MySQL, or may be an interaction issue

Consider the Whole Stack

- The issue may occur at any place in the stack, e.g.:
  - Application
  - Application host/hardware
  - Network between application and host with MySQL
  - MySQL host/hardware
  - MySQL
- This should also be considered when monitoring
- Consider the OS/hardware settings

Consider the Whole Stack

I/O settings on the operating system

- Test with various file systems, for example:
  - Ext4
  - XFS
  - ZFS
- The optimal file system varies with time
- Use a battery backed file system:
  - Allows you to disable the write barrier in Ext4 and XFS

Consider the Whole Stack

I/O settings on the operating system

- If the file system guarantees a full page write, the InnoDB doublewrite buffer can be disabled
- ZFS is the only main file system that makes this guarantee

Consider the Whole Stack

I/O settings on the operating system

- I/O Scheduler
  - Several Linux distributions use the CFQ scheduler by default
    - OK for reads
    - Serializes writes!
  - NOOP and deadline are usually better for MySQL workloads
    - Deadline is the default I/O scheduler for Oracle Linux

Consider the Whole Stack

I/O settings on the operating system

- I/O Scheduler
  - Check the current scheduler:

```
shell# cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

  - Update the scheduler dynamically:

```
shell# echo deadline > /sys/block/sda/queue/scheduler
shell# cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

  - To set at boot time, use the “elevator=deadline” boot option.

Consider the Whole Stack

Memory allocation library

- Linux glibc malloc can be a performance bottleneck
- Better to use alternative malloc library:
  - tcmalloc
  - jemalloc

Consider the Whole Stack

Memory allocation library

- Set alternative malloc library:
  - mysqld_safe

```
[mysqld_safe]
malloc-lib = /usr/lib64/libjemalloc.so.1
```

  - systemd distributions:
    - Set LD_PRELOAD in /etc/sysconfig/mysql

Consider the Whole Stack

Case study – Real world example

- MySQL Cluster, but illustrates well how you cannot focus on one part of the stack
- Data nodes often stalled
- Eventually data nodes were shut down when a stall took longer than 18 seconds by the watchdog to protect the rest of the cluster
- Always when trying to send data through TCP

Consider the Whole Stack

Case study – Real world example

- Turned out OS tools such as mpstat, iostat, and cat /proc/insterrupts also stalled
- Longest stall was 30+ seconds
- Collected data using the Linux perf tool:
  - For every single stall, a memory compaction was ongoing at the OS level
- Memory very fragmented:

```
shell$ cat /proc/buddyinfo
...
Node 0, zone Normal 18232 103 32 1400 4 0 0 0 0 0 0
```

Consider the Whole Stack

Case study – Real world example

- MySQL Cluster allocates and touches memory up front, so should not cause fragmentation after startup
- Turned out fragmentation was triggered by the I/O cache at the OS level
- TCP required Direct Memory Access (DMA) which much use contiguous memory, so with fragmented memory triggered memory compaction
- Enabling Direct I/O resolved the issue

Summary

Wrapping it all up

Summary

- MySQL performance tuning is similar to all other performance tuning:
  - Premature optimization is bad
  - Only change one thing at a time and don't make too large changes
  - Don't micro manage
  - One size does not fit all
  - Base your decisions on measurements
  - Understand what an option does before changing it
  - Understand your system (data)
  - Understand what you need
  - Consider the whole stack

Some Related Talks

- Tuesday
  - 4:00 p.m. | Park Central – City
    - MySQL Monitoring: Finding Meaning in Beautiful Graphs [CON2385]
    - Peter Zaitsev, Percona Inc
  - 6:15 p.m. | Park Central – City
    - Meet the MySQL Engineering Team [BOF1967]
- Wednesday
  - 3:00 p.m. | Park Central – Stanford
    - MySQL Performance: Demystified Tuning and Best Practices [CON1993]
    - Dimitri Kravtchuk, Oracle
  - 3:00 p.m. | Park Central – City
    - MySQL Optimizer: What's New in 5.7 and Sneak Peek at 8.0 [CON6112]
    - Manyi Lu, Director Software Engineering, Oracle

Some Related Talks

- Thursday
  - 9:30 a.m. | Park Central – City
    - MySQL Replication Troubleshooting for Oracle DBAs [CON1685]
    - Sveta Smirnova, Percona
  - 9:45 a.m. | Hotel Nikko – Peninsula
    - Solving Performance Problems Using MySQL Enterprise Monitor [HOL5412]
    - Mark Matthews, Oracle

- Thursday (cont.)
  - 1:15 p.m. | Park Central – City
    - A MySQL Sys Schema Deep Dive [CON4589]
    - Mark Leith, Oracle
  - 2:30 p.m. | Park Central – Stanford
    - MySQL Security Best Practices [CON4680]
    - Georgi Kodinov and Mike Frank, Oracle

Oracle MySQL Cloud Service

Learn More @ OpenWorld & JavaOne

Conference Presentation

- ***CON4851***

Introducing MySQL Cloud Service

Park Central Hotel – 3rd Floor / Stanford Room

Sept 20th (Tues) @ 4PM

- ***CON3810***

Deep Dive: Oracle MySQL Cloud Service

Park Central Hotel – 3rd Floor / Stanford

Sept 21st (Wed) @ 11AM

Hands-on Lab

- ***HOL4079***

Hands on with Oracle MySQL Cloud Service

Nikko Hotel - Peninsula

Sept 21st (Wed) @ 3:30PM

Demo Pod

- ***Database Showcase @ OpenWorld***

Oracle MySQL Cloud Service

Moscone Center South - Exhibition Hall

- ***JavaHub @ JavaOne***

IaaS & Databases: Compute VMs, Containers, MySQL, NoSQL, and Oracle Database in the Cloud

Hilton Hotel – Grand Ballroom Exhibition Hall

ORACLE

MySQL Community Reception @ Oracle OpenWorld

Celebrate, Have Fun and Mingle with Oracle's MySQL Engineers & Your Peers

- Tuesday, September 20 @ 7 pm

- Jillian's at Metreon: 175 Fourth Street, San Francisco

  At the corner of Howard and 4th st.; only 2-min walk from Moscone Center (same place as last year)

Join us!

Thank You

Q&A


ORACLE®