September 21 – 23, 2013
Hilton San Francisco

Improving Performance with MySQL Performance Schema

Jesper Wisborg Krogh
Principal Technical Support Engineer, MySQL

ORACLE

Background Information

To login to the virtual machine:

Username	ouser
Password	Oracle123
Root password	Oracle123

Installed software:

The following MySQL software has been installed:

Software	Abbreviation	Notes
MySQL Server 5.7.2	MySQL	
MySQL Enterprise Monitor 3.0.1	MEM	Backend MySQL is 5.6.13
MySQL Workbench 6.0.7	Workbench	Launch using button in the top panel
MySQL Utilities 1.3.5	Utilities	Scripts are located in /usr/bin
MySQL Enterprise Backup 3.9.0	MEB	Located in /opt/mysql/enterprise/bin/mysqlbackup

Additionally some processes are running in the background generating some activity in the database.

To login to MySQL:

Description	Shell	Workbench
Using the Unix socket	mysql --login-path=socket	Socket
Using TCP/IP	mysql --login-path=tcp	TCP
To access the MEM database	mysql --login-path=mem	MEM

MySQL Workbench 6.0 is also available – to launch Workbench, click on the button with the Workbench logo (with the dolphin) on the panel at the top of the screen. MySQL Workbench has one connection predefined for each of the three cases above named Socket, TCP, and MEM respectively.

Databases

The following databases are available in MySQL:

Database	Description
information_schema	The standard information schema with metadata and some performance related data.
employees	The employees sample database. Approximately 160M data in 4 million rows.
mysql	The MySQL system database.
performance_schema	The main database for the MySQL Performance Schema.
ps_helper	Mark Leith's ps_helper views and procedures for the Performance Schema. http://www.markleith.co.uk/ps_helper/ https://github.com/MarkLeith/dbahelper
ps_tools	Similar to ps_helper by Jesper Krogh. http://mysql.wisborg.dk
sakila	A medium sized sample database with views, stored programs, etc.
test	An empty test database.
world	The standard World sample database.

Starting and Stopping MySQL and MySQL Enterprise Monitor

Both MySQL and MySQL Enterprise Monitor has been started automatically with the VM. However should it be necessary to stop, start, or restart either it can be done as follows:

Action	Shell
MySQL – Start	sudo service mysql start
MySQL – Stop	sudo service mysql stop
MySQL – Restart	sudo service mysql restart
MEM Dashboard – Start	sudo service mysql-monitor-server start
MEM Dashboard – Stop	sudo service mysql-monitor-server stop
MEM Dashboard – Restart	sudo service mysql-monitor-server restart
MEM Dashboard – Start MySQL only	sudo service mysql-monitor-server start mysql
MEM Dashboard – Stop MySQL only	sudo service mysql-monitor-server stop mysql
MEM Dashboard – Restart MySQL only	sudo service mysql-monitor-server restart mysql
MEM Dasbboard – Start Tomcat only	sudo service mysql-monitor-server start tomcat
MEM Dashboard – Stop Tomcat only	sudo service mysql-monitor-server stop tomcat
MEM Dashboard – Restart Tomcat only	sudo service mysql-monitor-server restart tomcat
MEM Agent – Start	sudo service mysql-monitor-agent start
MEM Agent – Stop	sudo service mysql-monitor-agent stop
MEM Agent – Restart	sudo service mysql-monitor-agent restart
Queries – Start	sudo service mysql_queries start
Queries – Stop	sudo service mysql_queries stop

Useful resources

The following resources may be useful during the lab or at home:

Resource	URL
MySQL 5.7 Reference Manual	https://dev.mysql.com/doc/refman/5.7/en/
Performance Schema	https://dev.mysql.com/doc/refman/5.7/en/performance-schema.html
Information Schema	https://dev.mysql.com/doc/refman/5.7/en/information-schema.html
Mark Leith's ps_helper	http://www.markleith.co.uk/ps_helper/ https://github.com/MarkLeith/dbahelper

See also the reference list at the end of the workbook.

Tour of the MySQL Performance Schema

Configuration

We will start out taking a look at how MySQL has been configured with respect to the MySQL Performance Schema.

Starting with MySQL 5.6, a subset of the Performance Schema is enabled by default. The MySQL instances on the VM are using the default configuration. If you want to enable all consumers and instruments (see later for more information on these), you can do it in one of the following ways:

Enable consumers and instruments through /etc/my.cnf

Add the following options to /etc/my.cnf and restart MySQL:

```
performance_schema_instrument                                = '%=on'
performance_schema_consumer_events_stages_current            = ON
performance_schema_consumer_events_stages_history            = ON
performance_schema_consumer_events_stages_history_long       = ON
performance_schema_consumer_events_statements_current        = ON
performance_schema_consumer_events_statements_history        = ON
performance_schema_consumer_events_statements_history_long = ON
performance_schema_consumer_events_waits_current             = ON
performance_schema_consumer_events_waits_history             = ON
performance_schema_consumer_events_waits_history_long        = ON
performance_schema_consumer_global_instrumentation           = ON
performance_schema_consumer_thread_instrumentation           = ON
performance_schema_consumer_statements_digest                = ON
```

The first setting `performance_schema_instrument = '%=on'` switched on all instruments (% is a wildcard that matches all instruments – this can be used similar to a LIKE clause to enable a subset of instruments).

For the consumers it is necessary to enable each consumer explicitly. This is done by pre-pending the name of the consumer with `performance_schema_consumer_`, for example to enable the `statements_digest` consumer use the setting `performance_schema_consumer_statements_digest` and set it to `ON`.

Enable using update statements in the performance_schema database

All consumers and instruments can be enabled as:

```
UPDATE performance_schema.setup_consumers SET ENABLED = 'YES';
UPDATE performance_schema.setup_instruments SET ENABLED = 'YES', TIMED = 'YES';
```

The change will take effect immediately.

Enable using ps_tools

The ps_tools database includes a stored procedure to enable all consumers and instruments with a single statement:

```
CALL ps_tools.ps_enable_all();
```

Resetting the settings

To reset all settings (not only consumers and instruments) to the default settings (i.e. not taking /etc/my.cnf into consideration):

```
CALL ps_helper.reset_to_default(FALSE);
```

Performance Schema Variables

In addition to the options for which instruments and consumers are enabled at start up, there are a number of variables:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'performance\_schema%';
+-------------------------------------------------------+-------+
| Variable_name                                         | Value |
+-------------------------------------------------------+-------+
| performance_schema                                    | ON    |
| performance_schema_accounts_size                      | 100   |
| performance_schema_digests_size                       | 10000 |
…
| performance_schema_setup_actors_size                  | 100   |
| performance_schema_setup_objects_size                 | 100   |
| performance_schema_users_size                         | 100   |
+-------------------------------------------------------+-------+
34 rows in set (0.00 sec)
```

These defines the size of the various Performance Schema tables. Several of the values are automatically calculated based on other settings such as `max_connections`.

As all the Performance Schema data is in-memory, changing the size of the tables affects the memory usage. The memory usage of the Performance Schema can be checked with SHOW ENGINE PERFORMANCE_SCHEMA STATUS:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS;
+--------------------+-----------------------------------------------------------+------------+
| Type               | Name                                                      | Status     |
+--------------------+-----------------------------------------------------------+------------+
| performance_schema | events_waits_current.size                                 | 184        |
| performance_schema | events_waits_current.count                                | 2412       |
| performance_schema | events_waits_history.size                                 | 184        |
…
| performance_schema | memory_summary_by_host_by_event_name.memory               | 1600000    |
| performance_schema | performance_schema.memory                                 | 456413784  |
+--------------------+-----------------------------------------------------------+------------+
178 rows in set (0.02 sec)
```

The last row with `Name = performance_schema.memory` has the total memory usage for the Performance Schema.

Setup Tables

There are five setup tables for the Performance Schema:

```
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 tables include the current settings and allow for dynamic changes of the settings at runtime.

Changes to the setup tables in general takes effect immediately. One exception is changes to `setup_actors` which will only affect new connections.

Note: while it is possible to configure most of the Performance Schema settings dynamically, these changes are not persistent when MySQL restarts.

setup_actors

The `setup_actors` table controls which user accounts are instrumented by default (see also the threads table later). The `setup_actors` table has the following content by default:

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

The `HOST` and `USER` fields correspond to the same fields in `mysql.user`. The `ROLE` field is currently not used.

The rule is that if any row in `setup_actors` matches the user account, the connection will be instrumented. For background threads which do not have a user account, the thread is always instrumented unless turned off in the `threads` table.

setup_objects

The table `setup_objects` define which database object will be instrumented. In MySQL 5.6 this can only be configured for tables, however in 5.7 events, triggers, functions, and procedures have been added. The wildcard '%' is allowed. By default everything is enabled except objects in the `mysql`, `performance_schema`, and `information_schema` databases.

The default content of the table is:

```
mysql> SELECT * FROM setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| EVENT       | mysql              | %           | NO      | NO    |
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %                  | %           | YES     | YES   |
| FUNCTION    | mysql              | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %                  | %           | YES     | YES   |
| PROCEDURE   | mysql              | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %                  | %           | YES     | YES   |
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
| TRIGGER     | mysql              | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
20 rows in set (0.00 sec)
```

For setup_objects the most specific match is used. The difference between ENABLED and TIMED is when an object is instrumented whether the events are only counted or also timed.

To demonstrate the use of the setup_objects table, consider the following example:

```
mysql> TRUNCATE table_io_waits_summary_by_table;
Query OK, 0 rows affected (0.00 sec)
```

This resets the `table_io_waits_summary_by_table` table.

```
mysql> SELECT OBJECT_SCHEMA, OBJECT_NAME, COUNT_STAR, SUM_TIMER_WAIT FROM
table_io_waits_summary_by_table WHERE OBJECT_SCHEMA = 'world' AND OBJECT_NAME =
'Country';
Empty set (0.01 sec)
```

So the table does not have any rows for the world.Country table at this point – just as would be expected just after truncating a table.

```
mysql> SELECT Code, Name, Continent FROM world.Country WHERE NAME = 'United States';
+------+---------------+---------------+
| Code | Name          | Continent     |
+------+---------------+---------------+
| USA  | United States | North America |
+------+---------------+---------------+
1 row in set (0.00 sec)
```

After executing a query using the `world.Country` table, what does the `table_io_waits_summary_by_table` now show?

```
mysql> SELECT OBJECT_SCHEMA, OBJECT_NAME, COUNT_STAR, SUM_TIMER_WAIT FROM
table_io_waits_summary_by_table WHERE OBJECT_SCHEMA = 'world' AND OBJECT_NAME =
'Country';
+---------------+-------------+------------+----------------+
| OBJECT_SCHEMA | OBJECT_NAME | COUNT_STAR | SUM_TIMER_WAIT |
+---------------+-------------+------------+----------------+
| world         | Country     |        240 |      963058006 |
+---------------+-------------+------------+----------------+
1 row in set (0.00 sec)
```

So there are 240 events for the world.Country table now and a total of 963058006 picoseconds (10^{-12} seconds) has been spent using the table.

Now try the same again, but with a rule in the `setup_objects` table that turns off timing of the events on the `world.Country` table:

```
mysql> INSERT INTO setup_objects VALUES ('TABLE', 'world', 'Country', 'YES', 'NO');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> TRUNCATE table_io_waits_summary_by_table;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT OBJECT_SCHEMA, OBJECT_NAME, COUNT_STAR, SUM_TIMER_WAIT FROM
table_io_waits_summary_by_table WHERE OBJECT_SCHEMA = 'world' AND OBJECT_NAME =
'Country';
+---------------+-------------+------------+----------------+
| OBJECT_SCHEMA | OBJECT_NAME | COUNT_STAR | SUM_TIMER_WAIT |
+---------------+-------------+------------+----------------+
| world         | Country     |          0 |              0 |
+---------------+-------------+------------+----------------+
1 row in set (0.00 sec)
```

Now what is that? We just truncated the `table_io_waits_summary_by_table` table, but there is still content in it! For summary tables in the Performance Schema, `TRUNCATE` does in general not delete any of the existing rows; instead the counters are set to 0. This is what happened in this case.

```
mysql> SELECT Code, Name, Continent FROM world.Country WHERE NAME = 'United States';
+------+---------------+---------------+
| Code | Name          | Continent     |
+------+---------------+---------------+
| USA  | United States | North America |
+------+---------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT OBJECT_SCHEMA, OBJECT_NAME, COUNT_STAR, SUM_TIMER_WAIT FROM
table_io_waits_summary_by_table WHERE OBJECT_SCHEMA = 'world' AND OBJECT_NAME =
```

```
'Country';
+---------------+-------------+------------+----------------+
| OBJECT_SCHEMA | OBJECT_NAME | COUNT_STAR | SUM_TIMER_WAIT |
+---------------+-------------+------------+----------------+
| world         | Country     |        240 |              0 |
+---------------+-------------+------------+----------------+
1 row in set (0.01 sec)
```

Here the effect of setting `TIMED = 'NO'` is that the timer fields (here `SUM_TIMER_WAIT`) is not updated, but we can still see how many times `world.Country` has been accessed.

Finally we will reset the settings:

```
mysql> CALL ps_helper.reset_to_default(FALSE);
Query OK, 0 rows affected (0.00 sec)
```

setup_timers

The `setup_timers` table defines which timer is sued for each of the instrument types:

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

The `TIMER_NAME` can be set to any of the values available from the `performance_timer` table:

```
mysql> SELECT * FROM performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2277546341 |                1 |          13699 |
| NANOSECOND  |      1000000000 |                1 |          16107 |
| MICROSECOND |         1000000 |                1 |          15876 |
| MILLISECOND |            1038 |                1 |          16347 |
| TICK        |             103 |                1 |          17443 |
+-------------+-----------------+------------------+----------------+
5 rows in set (0.00 sec)
```

From the `performance_timer` table you can also see the timer frequency, resolution, and overhead (in number of cycles) using that particular timer.

Note: while `CYCLE` has the lowest overhead, it is also the least precise as the frequency is not completely constant (e.g. the CPU frequency might be changed by the OS depending on the workload). So timers using `CYCLE` tend to drift a bit compared to other timers. For the measurement of a duration this is generally not a problem, but sorting by the start time of the events should be avoided if not all events use the same timer.

setup_instruments

The `setup_instruments` table contain one row per instrumentation point in the source code. These are the events that can be collected. It is possible to specify both whether an instrument is producing events and if so whether it is timed; this is very similar to the `setup_objects` table:

```
mysql> SELECT * FROM setup_instruments LIMIT 1;
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc | NO      | NO    |
+----------------------------------------+---------+-------+
1 row in set (0.00 sec)
```

The name is constructed by components which form a hierarchy:

Class/Order/Family/Genus/Species

The number of components depends on the Class. The components are separated by '/'. When `ENABLED` is `YES`, the instrument produces events. `TIMED` is whether the events are timed or just counted.

The default for which instruments are enabled can be set in the MySQL configuration file using the `performance_schema_instrument` option.

setup_consumers

The last setup table is `setup_consumers` which lists the consumers of events from the instruments and allows you to specify whether it is enabled or not:

```
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 (0.00 sec)
```

The consumers also form a hierarchy – you can use the ps_setup_tree_consumers procedure in ps_tools to generate one from the Linux shell:

```
[ouser@localhost ~]$ echo -e "$(mysql --login-path=socket -Ee "CALL
```

```
ps_tools.ps_setup_tree_consumers('Text: Left-Right', TRUE)")"
*************************** 1. row ***************************
Consumers:
                        +--statements_digest
                        |                                                    +--events_stages_history
                        |                          +--events_stages_current------+
global_instrumentation--+                          |                         +--events_stages_history_long
                        |                          |
                        |                          |                         +--events_statements_history
                        +--thread_instrumentation--+--events_statements_current--+
                                                   |                         +--events_statements_history_long
                                                   |
                                                   |                         +--events_waits_history
                                                   +--events_waits_current-------+
                                                                             +--events_waits_history_long

Legend: Enabled - Partially enabled - Disabled
```

The `ps_setup_tree_consumers` procedure takes two arguments:

- The format which can be one of:
  - 'Text: Left-Right'
  - 'Text: Top-Bottom'
  - 'Dot: Left-Right'
  - 'Dot: Top-Bottom'
- Whether to use color or brackets to indicate whether the consumer is effectively enabled.

For a consumer to collect events, it is not enough that the consumer itself is enabled; all consumers above it in the hierarchy must be enabled as well. The `ps_setup_tree_consumers` procedure takes this into account.

As an alternative to the above procedure, the view `ps_tools.ps_setup_consumers` is the `setup_consumers` table with an additional column displaying whether the consumer is effectively enabled.

The Left-Right and Top-Bottom parts of the formats describes the direction of the graph.

The two dot formats can be used to generate for example a PNG or PDF version of the above graph. To create a dot formatted output:

```
mysql --login-path=socket -rBNe "CALL
ps_tools.ps_setup_tree_consumers('Dot: Left-Right', TRUE)" > consumers.dot
```

DOT FILES

The dot format is graph description language. The format is plain text so can be read using any text editor.

The VM has been installed with the dot program from the graphviz library. This program can be used to convert the text based dot formatted file to for example PNG images or PDF files:

```
dot -Tpdf graph.dot -o graph.pdf
```

or

```
dot -Tpng graph.dot -o graph.png
```

Programs that can be used to open the files created:

- PDF: `evince graph.pdf`
- PNG: `eog graph.png`

Instance Tables

The instance tables include information about the objects being instrumented. They provide event names and explanatory notes or status information. The relation to the setup tables is that the instance table has a NAME or EVENT_NAME column that corresponds to the NAME column in the setup_instruments table.

```
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)
```

Event Tables

The event tables are the main entry point for looking at the collected data. There are three groups of event tables depending on the type of event:

- Stages: The same stages as in the `State` column of `SHOW PROCESSLIST`, for example `Sending data`.
- Statements: The SQL statements that have been run on the server.
- Waits: Where the server is spending time

Each event has an event name that comes from the corresponding instrument in the `setup_instruments` table, e.g. `statement/sql/select` for a `SELECT` statement.

For each event type there are three tables with the actual (raw) data collected:

- `%_current`: the last event for each thread. Note that in some events are *molecular* events, so there can be more than one current event for one thread.
- `%_history`: the last N (around 10 by default) events for each thread. The number of events per thread can be configured using the `performance_schema_events_%_history_size` options.
- `%_history_long`: the last M (around 10000 by default) events irrespectively of the thread. The size of the tables can be configured with the `performance_schema_events_%_history_long_size` options.

Additionally there are a number of summary tables for each event type. The naming convention for the event summary tables is that the table name has two or more parts:

- `event_%_summary`: specified the event type and it is a summary table.
- One or more `_by_<field>`: specifies a field the summary is grouped by.

An example is `events_stages_summary_by_account_by_event_name`: a summary of stages grouped by account and event name.

Other Summary Tables

In addition to the event summary tables above, there are also a few other summary tables:

- For objects (currently only tables)
- For files
- For table I/O and Lock Wait
- For sockets
- For memory usage (5.7.2+ only)

Connection Tables

There are tables showing the current and total number of connections per user, host, or account (user@host). For example for accounts:

```
mysql> SELECT * FROM accounts;
+------+-----------+---------------------+-------------------+
| USER | HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+------+-----------+---------------------+-------------------+
| NULL | NULL      |                  18 |                20 |
| root | localhost |                   1 |                25 |
+------+-----------+---------------------+-------------------+
2 rows in set (0.00 sec)
```

This shows another aspect of the Performance Schema: note the row having both USER and HOST set to NULL. That is for the background threads, so not only can the Performance Schema give information about the client connections (foreground threads), it can also give insight into what the internal threads such as the InnoDB threads are doing.

Connection Attribute Tables

Related to the connection tables are two tables giving access to connection attributes:

- `session_account_connect_attrs`
- `session_connect_attrs`

```
mysql> SELECT * FROM session_connect_attrs;
+----------------+-----------------+------------+------------------+
| PROCESSLIST_ID | ATTR_NAME       | ATTR_VALUE | ORDINAL_POSITION |
+----------------+-----------------+------------+------------------+
|              8 | _os             | Linux      |                0 |
|              8 | _client_name    | libmysql   |                1 |
|              8 | _pid            | 7635       |                2 |
|              8 | _client_version | 5.7.2-m12  |                3 |
|              8 | _platform       | x86_64     |                4 |
|              8 | program_name    | mysql      |                5 |
+----------------+-----------------+------------+------------------+
6 rows in set (0.00 sec)
```

The difference between the two tables is that `session_connect_attrs` includes all the connections whereas `session_account_connect_attrs` only includes the connections for the same account as the current user. That is, you can get the content of `session_account_connect_attrs` from `session_connect_attrs` using the query:

```
SELECT a.*
  FROM session_connect_attrs a
       INNER JOIN threads t USING (PROCESSLIST_ID)
 WHERE     t.PROCESSLIST_USER = SUBSTRING_INDEX(USER(), '@', 1)
       AND t.PROCESSLIST_HOST = SUBSTRING_INDEX(USER(), '@', -1);
```

Threads

The `threads` table is one of the most central tables in the Performance Schema. The `THREAD_ID` is for example a "key" for all of the non-summary event tables.

The example below includes both a background thread (`THREAD_ID = 16`) and a foreground thread (`THREAD_ID = 27`).

Background threads are the ones created by MySQL to handle the internal server activity – in this case it is the master InnoDB thread.

Foreground threads are client connections where `PROCESSLIST_ID` is the same as the Id displayed by `SHOW PROCESSLIST`. The active connection's processlist id can be found using the `CONNECTION_ID()` function.

The `INSTRUMENTED` column tells whether the thread is being instrumented. This column is updatable, so for a given thread, instrumentation can be enabled and disabled on demand.

```
mysql> SELECT * FROM threads WHERE NAME = 'thread/innodb/srv_master_thread' OR
PROCESSLIST_ID = CONNECTION_ID()\G
*************************** 1. row ***************************
          THREAD_ID: 16
               NAME: thread/innodb/srv_master_thread
               TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: NULL
  PROCESSLIST_STATE: NULL
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
*************************** 2. row ***************************
          THREAD_ID: 27
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 8
   PROCESSLIST_USER: root
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM threads WHERE NAME = 'thread/innodb/srv_master_thread'
OR PROCESSLIST_ID = CONNECTION_ID()
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
2 rows in set (0.00 sec)
```

Tools to Help with Ad-Hoc Configuration Changes

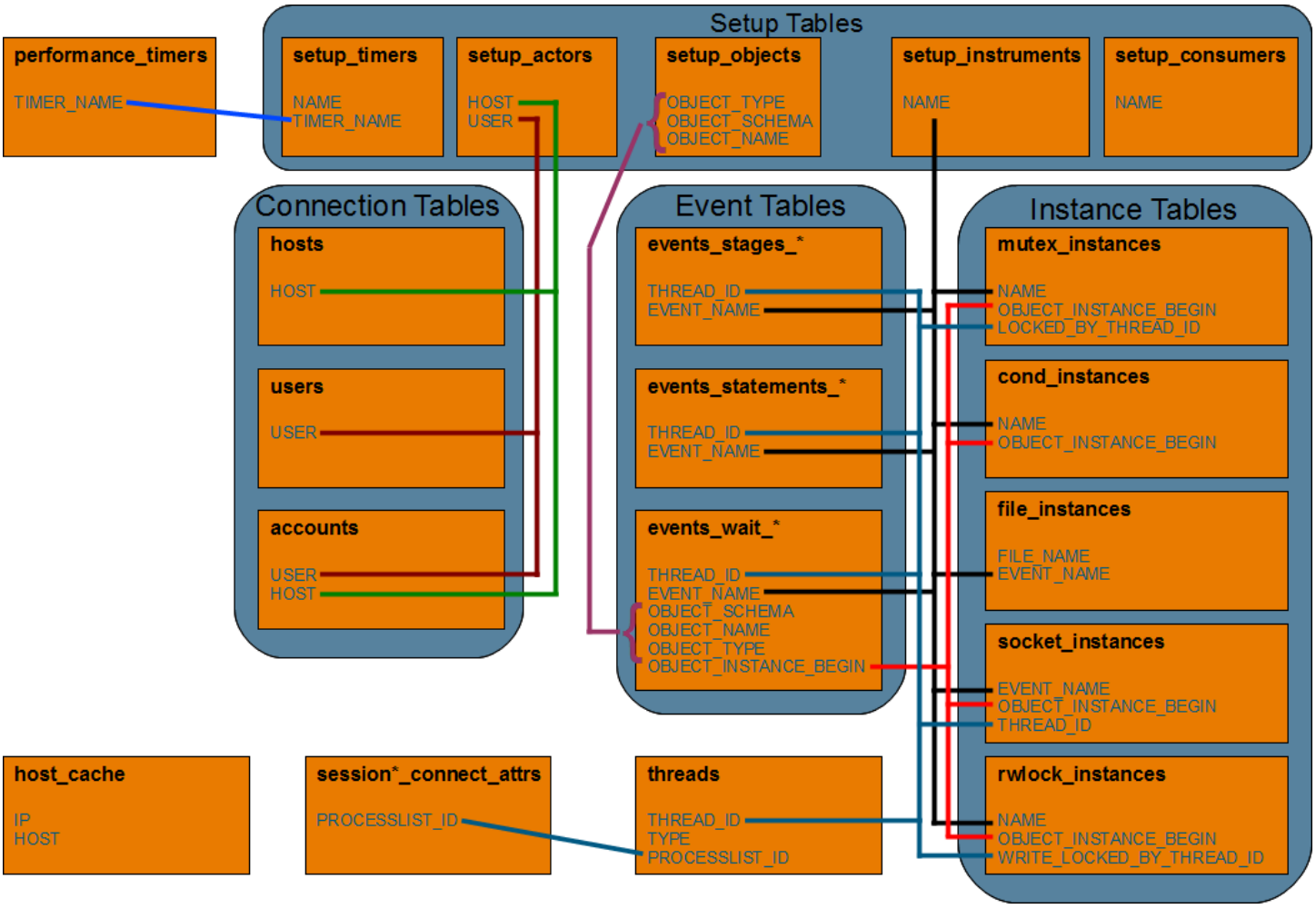
As can be seen from the above, there are several tables to keep track of when changing the configuration of the Performance Schema. In addition to `ps_tools.ps_enable_all()` and `ps_helper.reset_to_default(FALSE)` discussed earlier, `ps_helper` has a few other procedures that makes life easier when you need to change the configuration in order to investigate an issue.

- `save_current_config()` saves the current configuration in a set of temporary tables.
- `reload_saved_config()` restores the saved configuration.
- `truncate_all(FALSE)` truncates all the events and summary tables. This is important to consider to avoid making observations where the settings have changed. The procedure takes one Boolean argument which specifies whether the executed statements should be printed or not.

```
mysql> CALL ps_helper.save_current_config();
Query OK, 19 rows affected (0.00 sec)

mysql> CALL ps_helper.truncate_all(FALSE);
Query OK, 0 rows affected (0.02 sec)

mysql> -- Perform investigation
mysql> -- ...
mysql> -- ...
mysql> -- ...
mysql> CALL ps_helper.reload_saved_config();
Query OK, 0 rows affected (0.20 sec)
```

Overview of the Relation Between Tables

The following diagram shows how the Performance Schema tables relate to each other – summary tables are not included:



Investigating Queries

The following will look at the options for investigating which queries are executed on the server. The topics are:

- `SHOW PROCESSLIST`
- Digests
- `ps_helper` views and procedures
- MySQL Enterprise Monitor (MEM) 3.0 Query Analyzer

In the following it can be an advantage to turn on background queries to general some background queries. The queries display a range of queries from simple primary key lookups to badly written queries scanning large tables without `WHERE` clauses as well as queries causing errors.

To enable the queries execute in the Linux shell:

```
[ouser@localhost ~]$ sudo service mysql_queries start
Starting MySQL Queries.........                            [  OK  ]
```

SHOW PROCESSLIST

Using the Performance Schema to get the equivalent of `SHOW PROCESSLIST` has several advantages:

- Less locking, so less impact on other queries
- Possible to get more details
- Uses regular `SELECT` statements

The simplest way to get the processlist is to just use the `threads` table:

```
mysql> SELECT PROCESSLIST_ID AS Id, PROCESSLIST_USER AS User, PROCESSLIST_HOST AS Host, PROCESSLIST_DB AS db,
PROCESSLIST_COMMAND as Command, PROCESSLIST_TIME AS Time, PROCESSLIST_STATE AS State, LEFT(PROCESSLIST_INFO,
100) AS Info FROM threads WHERE PROCESSLIST_ID = CONNECTION_ID()\G
*************************** 1. row ***************************
     Id: 16
   User: root
   Host: localhost
     db: performance_schema
Command: Query
   Time: 0
  State: Sending data
   Info: SELECT PROCESSLIST_ID AS Id, PROCESSLIST_USER AS User, PROCESSLIST_HOST AS Host, PROCESSLIST_DB AS d
1 row in set (0.18 sec)
```

This works no matter which consumers and instruments are enabled.

However if the consumer `events_statements_current` is enabled, a much more interesting processlist can be obtained by joining on `events_statements_current`. An example of a new processlist can be found in `ps_helper.processlist`. With the default Performance Schema settings it returns:

```
mysql> SELECT * FROM ps_helper.processlist WHERE conn_id = CONNECTION_ID()\G
*************************** 1. row ***************************
               thd_id: 35
              conn_id: 16
                 user: root@localhost
                   db: performance_schema
              command: Query
                state: Sending data
                 time: 0
    current_statement: SELECT * FROM ps_helper.proces ... HERE conn_id = CONNECTION_ID()
         lock_latency: 195.30 ms
        rows_examined: 0
            rows_sent: 0
        rows_affected: 0
           tmp_tables: 2
      tmp_disk_tables: 0
            full_scan: YES
       current_memory: 1.41 MiB
       last_statement: NULL
last_statement_latency: NULL
            last_wait: NULL
    last_wait_latency: NULL
               source: NULL
1 row in set (0.26 sec)
```

To get all available data from `ps_helper.processlist` the following must be enabled:

- Consumer `events_statements_current`
- Consumer `events_waits_current`
- Instruments `memory/%` must be `ENABLED`

Some things to note about the output:

- The statements, latencies, and the memory usage is formatted. This is done with the `ps_helper` functions:
  - `format_statement()`
  - `format_time()`
  - `format_bytes()`
- Additionally there is `format_path()` for file paths (see the section on I/O later).
- `last_statement` will be set if the thread is not currently executing a statement.
- The memory usage is new as of MySQL 5.7.2
- The rest of the output is also available in MySQL 5.6

Consider the example:

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES' WHERE NAME =
'events_statements_history';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> CALL ps_helper.truncate_all(FALSE);
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT Code, Name FROM world.Country WHERE Code = 'AUS';
+------+-----------+
| Code | Name      |
+------+-----------+
| AUS  | Australia |
+------+-----------+
1 row in set (0.00 sec)

mysql> SELECT Code, Name FROM world.Country WHERE Code = 'USA';
+------+---------------+
| Code | Name          |
+------+---------------+
| USA  | United States |
+------+---------------+
1 row in set (0.00 sec)

mysql> SELECT DIGEST, DIGEST_TEXT, SQL_TEXT FROM events_statements_history WHERE SQL_TEXT
LIKE 'SELECT Code, Name FROM world.Country %' AND THREAD_ID =
ps_tools.ps_thread_id(NULL)\G
*************************** 1. row ***************************
     DIGEST: 192967f1f46a922c0837f0782f28a9cc
DIGEST_TEXT: SELECT CODE , NAME FROM `world` . `Country` WHERE CODE = ?
   SQL_TEXT: SELECT Code, Name FROM world.Country WHERE Code = 'AUS'
*************************** 2. row ***************************
     DIGEST: 192967f1f46a922c0837f0782f28a9cc
DIGEST_TEXT: SELECT CODE , NAME FROM `world` . `Country` WHERE CODE = ?
   SQL_TEXT: SELECT Code, Name FROM world.Country WHERE Code = 'USA'
2 rows in set (0.01 sec)
```

In the last query, `ps_tools.ps_thread_id(NULL)` is used to get the thread id of the connection.

Note how the `DIGEST` and `DIGEST_TEXT` is the same for the two `SELECT` queries. When the consumer `statements_digest` is enabled (this is the default), the Performance Schema normalizes (creates the `DIGEST_TEXT`) all queries. This process is similar to what `mysqldumpslow` does when analyzing the Slow Query Log. The `DIGEST_TEXT` is then used to calculate the `DIGEST` by using md5 sum.

The `DIGEST` is then used to aggregate statistics for similar queries in the `events_statements_summary_by_digest` table:

```
mysql> SELECT * FROM events_statements_summary_by_digest WHERE DIGEST =
'192967f1f46a922c0837f0782f28a9cc'\G
*************************** 1. row ***************************
                SCHEMA_NAME: performance_schema
                     DIGEST: 192967f1f46a922c0837f0782f28a9cc
                DIGEST_TEXT: SELECT CODE , NAME FROM `world` . `Country` WHERE CODE = ?
                 COUNT_STAR: 2
             SUM_TIMER_WAIT: 678298000
             MIN_TIMER_WAIT: 310374000
             AVG_TIMER_WAIT: 339149000
             MAX_TIMER_WAIT: 367924000
              SUM_LOCK_TIME: 282000000
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 2
          SUM_ROWS_EXAMINED: 2
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
                 FIRST_SEEN: 2013-09-18 18:58:39
                  LAST_SEEN: 2013-09-18 18:58:43
1 row in set (0.00 sec)
```

ps_helper Views and Procedures

The `events_statements_summary_by_digest` table is in itself an excellent source of information and it is there by default.

However it is also possible to use it as a base for other views. Examples from `ps_helper` are:

- `statement_analysis`
- `statements_with_runtimes_in_95th_percentile`
- `statements_with_full_table_scans`
- `statements_with_sorting`
- `statements_with_temp_tables`

Additionally `ps_helper` has two procedures that can be used to investigate queries.

dump_thread_stack()

The procedure `dump_thread_stack()` will generate a dot formatted file with the stack trace for one thread. The procedure takes the arguments:

- The thread id to investigate
- The output file – this file must not exist
- How long to collect data (in seconds)
- The time between sampling (in seconds)
- Whether to reset all Performance Schema data before starting the data collection
- Whether to automatically enable all consumers/instruments and disable other threads – this uses the `save_current_config()` and `reload_saved_config()`
- Whether to use debug mode (adds source information)

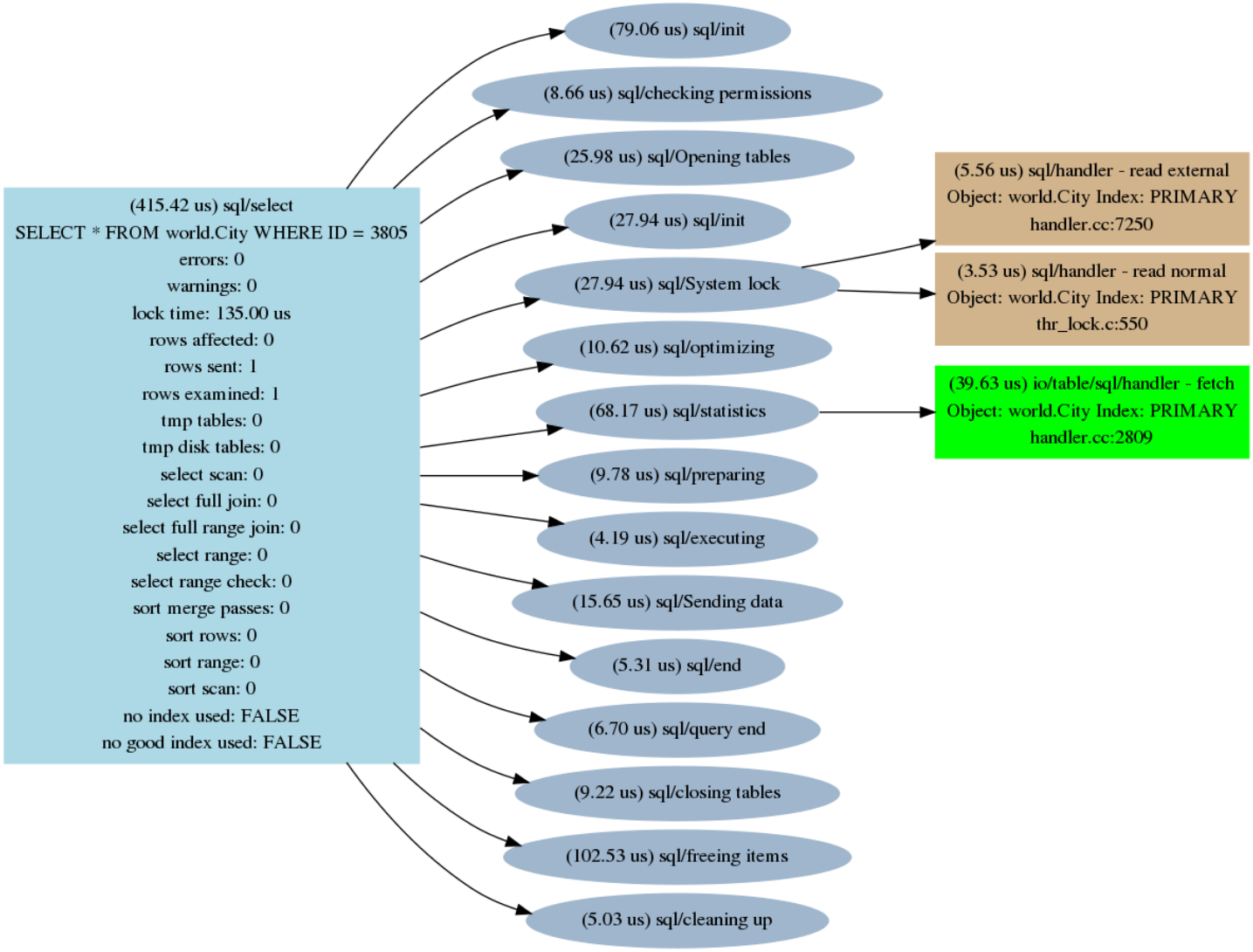
As an example create two connections. In the connection that should be monitored, get the thread id and enter the query to investigate, but do not submit:

```
mysql> SELECT ps_tools.ps_thread_id(NULL);
+-----------------------------+
| ps_tools.ps_thread_id(NULL) |
+-----------------------------+
|                          23 |
+-----------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM world.City WHERE ID = 3805;
```

Start the `dump_stread_stack` procedure – make sure it runs for long enough to switch to the other connection and execute the query while the procedure is in progress:

```
mysql> CALL ps_helper.dump_thread_stack(23, '/tmp/stack.dot', 20, 0.1, TRUE, TRUE, TRUE);
+-------------------------------------------+
| Info                                      |
+-------------------------------------------+
| Data collection starting for THREAD_ID = 23 |
+-------------------------------------------+
1 row in set (0.03 sec)

+--------------------------------------+
| Info                                 |
+--------------------------------------+
| Stack trace written to /tmp/stack.dot |
+--------------------------------------+
1 row in set (20.20 sec)

+-----------------------------------------------+
| Convert to PDF                                |
+-----------------------------------------------+
| dot -Tpdf -o /tmp/stack_23.pdf /tmp/stack.dot |
+-----------------------------------------------+
1 row in set (20.20 sec)

+-----------------------------------------------+
| Convert to PNG                                |
+-----------------------------------------------+
| dot -Tpng -o /tmp/stack_23.png /tmp/stack.dot |
+-----------------------------------------------+
1 row in set (20.20 sec)

Query OK, 0 rows affected (20.20 sec)
```

The last two outputs are sample commands to convert the dot formatted output file to a PDF or PNG file respectively. See also the sidebar on 'DOT FILES' above.



analyze_statement_digest()

If you have a common query on the server and want to collect information about it, you can use `analyze_statement_digest()` which takes the following parameters:

- The digest to investigate
- How many seconds to collect data for
- How often to take a snapshot (in seconds)
- Whether to truncate the `events_statements_history_long` and `events_stages_history_log` tables before starting
- Whether to automatically turn on required consumers

Like `dump_thread_stack()` the `save_current_config()` and `reload_saved_config()` procedures are used if the required consumers are automatically turned on.

With the mysql_queries running, one often executed query has the digest `6f4ad8c048e735f01f42121fdd81f3e3`:

```
mysql> CALL ps_helper.analyze_statement_digest('6f4ad8c048e735f01f42121fdd81f3e3', 30, 1.0, TRUE, TRUE);
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
1 row in set (30.50 sec)

+------------+-----------+-----------+-----------+---------------+---------------+------------+------------+
| executions | exec_time | lock_time | rows_sent | rows_affected | rows_examined | tmp_tables | full_scans |
+------------+-----------+-----------+-----------+---------------+---------------+------------+------------+
|         40 | 69.02 ms  | 2.85 ms   |        40 |             0 |          9560 |          0 |         40 |
+------------+-----------+-----------+-----------+---------------+---------------+------------+------------+
1 row in set (30.50 sec)

Empty set (30.50 sec)

+---------------------------+
| LONGEST RUNNING STATEMENT |
+---------------------------+
| LONGEST RUNNING STATEMENT |
+---------------------------+
1 row in set (30.50 sec)

+-----------+-----------+-----------+-----------+---------------+---------------+------------+-----------+
| thread_id | exec_time | lock_time | rows_sent | rows_affected | rows_examined | tmp_tables | full_scan |
+-----------+-----------+-----------+-----------+---------------+---------------+------------+-----------+
|      1536 | 3.95 ms   | 56.00 us  |         1 |             0 |           239 |          0 |         1 |
+-----------+-----------+-----------+-----------+---------------+---------------+------------+-----------+
1 row in set (30.50 sec)

+-----------------------------------------------------+
| sql_text                                            |
+-----------------------------------------------------+
| SELECT * FROM world.Country WHERE NAME = 'Morocco' |
+-----------------------------------------------------+
1 row in set (30.50 sec)

Empty set (30.50 sec)

+----+-------------+---------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table   | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+---------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | Country | ALL  | NULL          | NULL | NULL    | NULL |  239 | Using where |
+----+-------------+---------+------+---------------+------+---------+------+------+-------------+
1 row in set (31.01 sec)

Query OK, 0 rows affected (31.01 sec)
```

MySQL Enteprise Monitor (MEM) 3.0 Query Analyzer

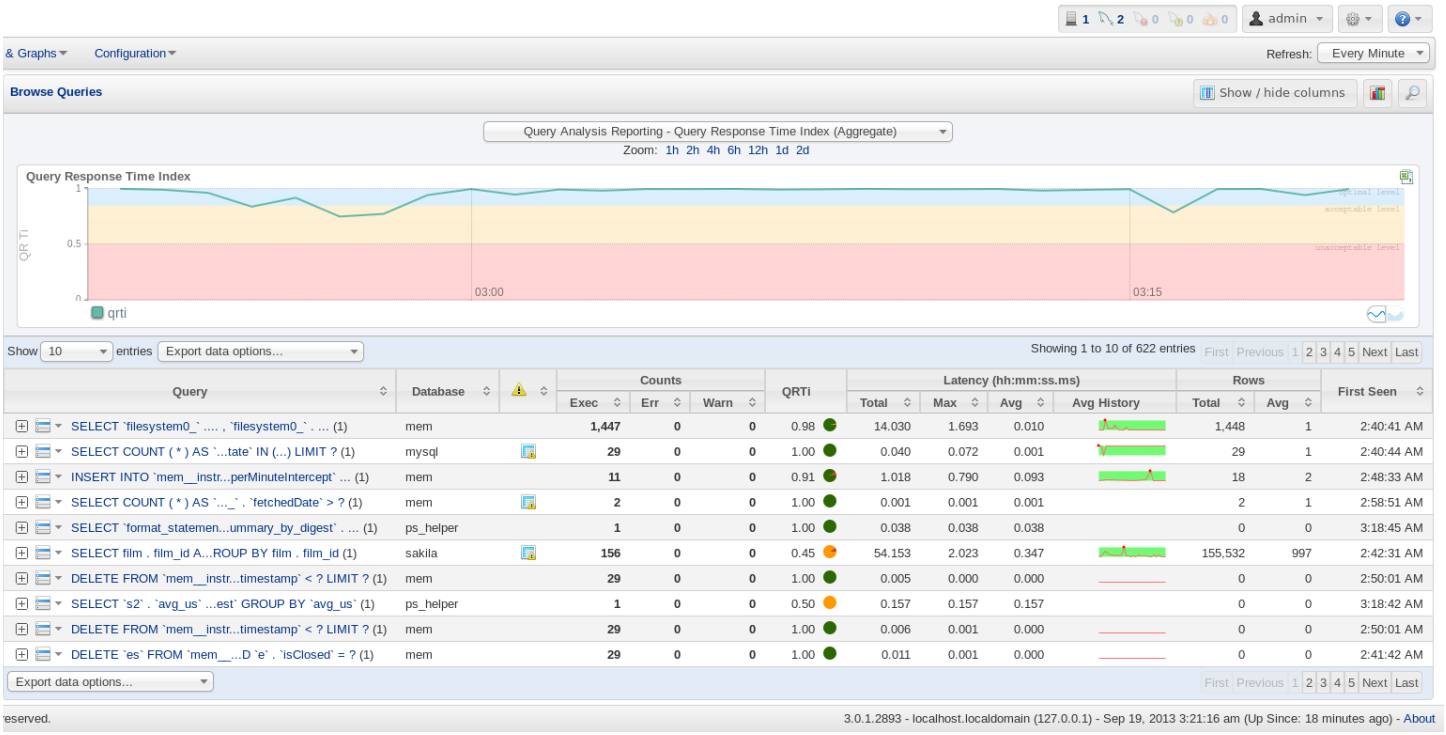
In MEM 2.3 and earlier, to use the Query Analyzer required it was required to use a proxy or a connector that could send the necessary data to the Query Analyzer.

With MEM 3.0 when monitoring MySQL 5.6.14 or later or MySQL 5.7.2 or later, the Query Analyzer can take advantage of the Performance Schema to get the data. Even with just the default settings, the `events_statements_summary_by_digest` is enough to get started, although in order to get sample queries it is also necessary to keep the history.

To use MEM's Query Analyzer launch Firefox from the menu at the top of the screen. The login is:

Username	admin
Password	Oracle123

Go to the Query Analyzer tab:



Schema, Disk, and Memory

There are other factors to keep an eye on other than the queries themselves. In this part the topics are:

- The Schema
- Disk I/O
- Memory usage – new in MySQL 5.7.2

Schema

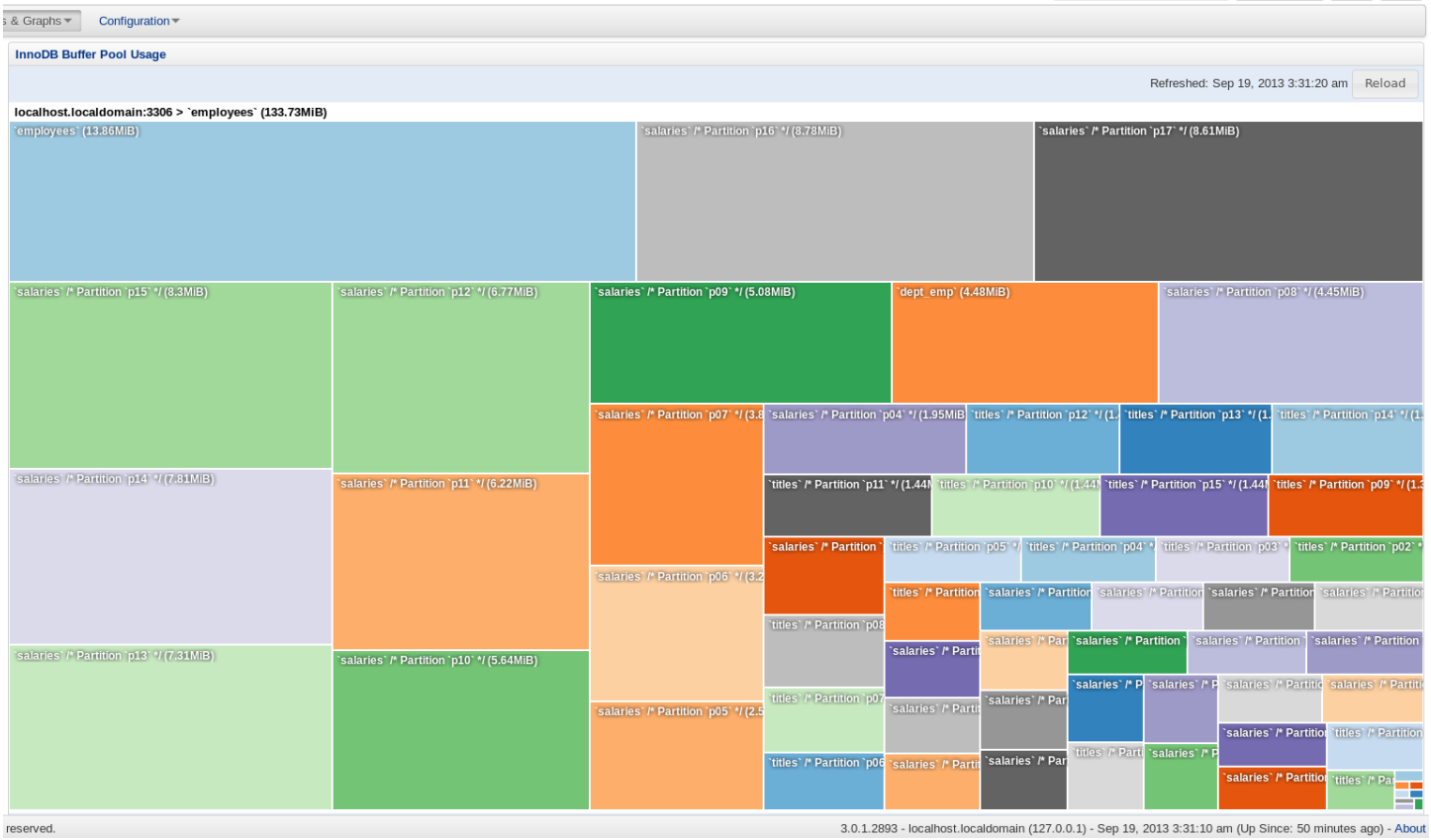
The Performance Schema has several tables with data about the schema. These can be used to find out which tables are hotspots, which indexes are missing, and whether there are any indexes that are not used at all? `ps_helper` can again help organizing the data.

Table Statistics

The `schema_table_statistics` and `schema_table_statistics_with_buffer` gives a summary of how much each table is used and the latency involved with the operations. `schema_table_statistics_with_buffer` additionally uses the `INNODB_BUFFER_PAGE` in the Information Schema to determine how much data the table contributes with in the InnoDB Buffer Pool:

```
mysql> SELECT * FROM ps_helper.schema_table_statistics_with_buffer LIMIT 1\G
*************************** 1. row ***************************
              table_schema: employees
                table_name: salaries
              rows_fetched: 115624831
             fetch_latency: 00:14:44.10
             rows_inserted: 0
            insert_latency: 0 ps
              rows_updated: 71974322
            update_latency: 00:14:54.64
              rows_deleted: 0
            delete_latency: 0 ps
          io_read_requests: 15
                   io_read: 3.90 KiB
           io_read_latency: 7.49 ms
         io_write_requests: 0
                  io_write: 0 bytes
          io_write_latency: 0 ps
          io_misc_requests: 18
           io_misc_latency: 306.66 us
   innodb_buffer_allocated: NULL
        innodb_buffer_data: NULL
       innodb_buffer_pages: NULL
innodb_buffer_pages_hashed: NULL
   innodb_buffer_pages_old: NULL
 innodb_buffer_rows_cached: NULL
1 row in set (1.55 sec)
```

This can be related to MEM as well. While from the Information Schema rather than the Performance Schema, the content of the InnoDB Buffer Pool is displayed in a graphical fashion in the InnoDB Buffer Pool Usage report:



Index Statistics

To determine the usage of the indexes, the view `schema_index_statistics` can give an overview:

```
mysql> SELECT * FROM ps_helper.schema_index_statistics LIMIT 1\G
*************************** 1. row ***************************
  table_schema: employees
    table_name: salaries
    index_name: PRIMARY
 rows_selected: 65810773
select_latency: 00:15:23.52
 rows_inserted: 0
insert_latency: 0 ps
  rows_updated: 1717
update_latency: 2.75 s
  rows_deleted: 0
delete_latency: 0 ps
1 row in set (0.01 sec)
```

Full Table Scans

The `schema_tables_with_full_table_scans` can be used to locate the tables that are seeing table scans:

```
mysql> SELECT * FROM ps_helper.schema_tables_with_full_table_scans;
+---------------+-------------+-------------------+
| object_schema | object_name | rows_full_scanned |
+---------------+-------------+-------------------+
| employees     | salaries    |         129979175 |
| world         | City        |          27446160 |
| employees     | employees   |          10800900 |
| world         | Country     |           1750560 |
| sakila        | category    |             21590 |
| sakila        | staff       |               942 |
| employees     | departments |               180 |
+---------------+-------------+-------------------+
7 rows in set (0.02 sec)
```

Unused Indexes

An unused index take up storage and causes overhead as it is still kept up to date. While not all unused indexes can be removed – some may be the PRIMARY KEY, others be part of foreign key definitions – it is good to keep an eye on which are not used. This can be done with the `schema_unused_indexes` view:

```
mysql> SELECT * FROM ps_helper.schema_unused_indexes;
+---------------+---------------+-----------------------------+
| object_schema | object_name   | index_name                  |
+---------------+---------------+-----------------------------+
| employees     | departments   | PRIMARY                     |
| employees     | departments   | dept_name                   |
| employees     | dept_emp      | emp_no                      |
| employees     | dept_emp      | PRIMARY                     |
| employees     | salaries      | emp_no                      |
| employees     | titles        | emp_no                      |
| sakila        | actor         | idx_actor_last_name         |
| sakila        | address       | idx_fk_city_id              |
| sakila        | category      | PRIMARY                     |
| sakila        | city          | idx_fk_country_id           |
| sakila        | film          | idx_fk_language_id          |
| sakila        | film          | idx_title                   |
| sakila        | film          | idx_fk_original_language_id |
| sakila        | film_actor    | PRIMARY                     |
| sakila        | film_category | PRIMARY                     |
| sakila        | film_text     | PRIMARY                     |
| sakila        | film_text     | idx_title_description       |
| sakila        | staff         | idx_fk_address_id           |
| sakila        | staff         | idx_fk_store_id             |
| sakila        | staff         | PRIMARY                     |
| world         | Country       | PRIMARY                     |
+---------------+---------------+-----------------------------+
21 rows in set (0.01 sec)
```

Unused Stored Procedures and Functions

Starting with MySQL 5.7.2 stored functions, procedures, triggers, and events are also instrumented in the Performance Schema. This can for example be used to find those functions and procedures that are not used. The view `schema_unused_routines` in `ps_tools` does that:

```
mysql> SELECT * FROM ps_tools.schema_unused_routines WHERE object_schema = 'sakila';
+---------------+--------------------+-------------+
| object_schema | object_name        | object_type |
+---------------+--------------------+-------------+
| sakila        | film_in_stock      | PROCEDURE   |
| sakila        | film_not_in_stock  | PROCEDURE   |
| sakila        | rewards_report     | PROCEDURE   |
+---------------+--------------------+-------------+
3 rows in set (0.01 sec)
```

Disk I/O

The disk – particularly with spinning disks – can quickly become a bottleneck. I/O was among the first to be instrumented in the Performance Schema and dates back to MySQL 5.5. Combining the information from this section with the previous can for example give hints to whether it is worth moving some data files, the InnoDB log files, the binary log, etc. to another disk system

Latest I/O

The `ps_helper` view `latest_file_io` gives an overview of the latest I/O wait events. The view is ever changing:

```
mysql> SELECT * FROM ps_helper.latest_file_io LIMIT 10;
+-----------------------+-----------------------------------------+----------+-----------+------------+
| thread                | file                                    | latency  | operation | requested  |
+-----------------------+-----------------------------------------+----------+-----------+------------+
| io_write_thread:9     | @@datadir/employees/salaries#P#p05.ibd | 3.47 ms  | sync      | NULL       |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 89.52 us | write     | 304.00 KiB |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 3.28 ms  | sync      | NULL       |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 55.70 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 26.87 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 28.29 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 45.10 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 25.39 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 24.05 us | write     | 16.00 KiB  |
| page_cleaner_thread:18 | @@datadir/ibdata1                      | 45.64 us | write     | 16.00 KiB  |
+-----------------------+-----------------------------------------+----------+-----------+------------+
10 rows in set (0.21 sec)

mysql> SELECT * FROM ps_helper.latest_file_io LIMIT 10;
+--------------------------+--------------------------+-----------+-----------+-----------+
| thread                   | file                     | latency   | operation | requested |
+--------------------------+--------------------------+-----------+-----------+-----------+
| root@localhost:35030:8774 | @@datadir/ib_logfile1    | 1.70 ms   | write     | 3.94 MiB  |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 181.34 us | create    | NULL      |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYD | 50.92 us  | create    | NULL      |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 24.40 us  | write     | 176 bytes |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 6.20 us   | write     | 100 bytes |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 5.44 us   | write     | 7 bytes   |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 4.97 us   | write     | 7 bytes   |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 5.23 us   | write     | 7 bytes   |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 5.00 us   | write     | 7 bytes   |
| root@localhost:4138       | @@tmpdir//#sql_acc_0.MYI | 5.20 us   | write     | 7 bytes   |
+--------------------------+--------------------------+-----------+-----------+-----------+
10 rows in set (0.10 sec)
```

In the first output, it's all InnoDB. A few seconds later, it's almost all MyISAM temporary tables.

Note how the file paths uses `@@datadir` and `@@tmpdir` – this is the `format_path()` function in `ps_helper` that makes that substitution.

Thread I/O

If you need to find out which background thread or connection is causing I/O, you can use the `io_by_thread_by_latency` view:

```
mysql> SELECT * FROM ps_helper.io_by_thread_by_latency LIMIT 1\G
*************************** 1. row ***************************
           user: page_cleaner_thread
     count_star: 900887
  total_latency: 00:17:33.14
    min_latency: 2.91 ns
    avg_latency: 10.42 ms
    max_latency: 7.28 s
      thread_id: 18
processlist_id: NULL
1 row in set (0.01 sec)
```

Global I/O Views

There are four related views to monitor the global I/O (as :

- `io_global_by_file_by_bytes`
- `io_global_by_file_by_latency`
- `io_global_by_wait_by_bytes`
- `io_global_by_wait_by_latency`

An example is:

```
mysql> SELECT * FROM ps_helper.io_global_by_file_by_bytes LIMIT 2\G
*************************** 1. row ***************************
         file: @@datadir/ibdata1
   count_read: 1210
   total_read: 18.91 MiB
     avg_read: 16.00 KiB
  count_write: 491768
total_written: 20.32 GiB
    avg_write: 43.33 KiB
        total: 20.34 GiB
    write_pct: 99.91
*************************** 2. row ***************************
         file: @@datadir/ib_logfile0
   count_read: 4
   total_read: 3.50 KiB
     avg_read: 896 bytes
  count_write: 6500
total_written: 7.83 GiB
    avg_write: 1.23 MiB
        total: 7.83 GiB
    write_pct: 100.00
2 rows in set (0.00 sec)
```

Memory Usage

New in MySQL 5.7.2 is the instrumentation of memory usage. While not yet complete – particularly InnoDB is missing, it can still be used to compare the memory usage of different connections.

It has already been shown how `ps_helper.processlist` in MySQL 5.7 includes the memory usage.

The raw tables for this are:

```
mysql> SHOW TABLES LIKE 'memory%';
+----------------------------------------+
| Tables_in_performance_schema (memory%) |
+----------------------------------------+
| memory_summary_by_account_by_event_name |
| memory_summary_by_host_by_event_name    |
| memory_summary_by_thread_by_event_name  |
| memory_summary_by_user_by_event_name    |
| memory_summary_global_by_event_name     |
+----------------------------------------+
5 rows in set (0.00 sec)
```

Memory instrumentation is disabled by default:

```
mysql> SELECT * FROM setup_instruments WHERE NAME LIKE 'memory/%' LIMIT 10;
+-------------------------------------------------------+---------+-------+
| NAME                                                  | ENABLED | TIMED |
+-------------------------------------------------------+---------+-------+
| memory/sql/buffered_logs                              | NO      | NO    |
| memory/sql/Locked_tables_list::m_locked_tables_root | NO      | NO    |
| memory/sql/THD::transactions::mem_root                | NO      | NO    |
| memory/sql/Delegate::memroot                          | NO      | NO    |
| memory/sql/sql_acl_mem                                | NO      | NO    |
| memory/sql/sql_acl_memex                              | NO      | NO    |
| memory/sql/thd::main_mem_root                         | NO      | NO    |
| memory/sql/help                                       | NO      | NO    |
| memory/sql/new_frm_mem                                | NO      | NO    |
| memory/sql/TABLE_SHARE::mem_root                      | NO      | NO    |
+-------------------------------------------------------+---------+-------+
10 rows in set (0.00 sec)
```

References

- https://dev.mysql.com/doc/refman/5.7/en/performance-schema.html
- http://www.markleith.co.uk/
- http://www.markleith.co.uk/ps_helper/
- https://github.com/MarkLeith/dbahelper
- http://www.drdobbs.com/database/detailed-profiling-of-sql-activity-in-my/240154959?pgno=1
- http://mysql.wisborg.dk/
- http://mysqlblog.fivefarmers.com/tag/performance_schema/
- http://en.wikipedia.org/wiki/DOT_(graph_description_language)
- http://www.graphviz.org/doc/info/lang.html
- MySQL Enterprise Monitor 3.0
    - https://dev.mysql.com/doc/mysql-monitor/3.0/en/mem-qanal-using-ui.html
    - https://dev.mysql.com/doc/mysql-monitor/3.0/en/mem-features-qrti.html