Table of Contents

- Performance Schema Configuration
- 5.6+: Statements Instrumentation
- 5.7+: Prepared Statements
- 5.7+: Stored Routines Instrumentation
- 5.7+: Locks Diagnostic
- 5.7+: Memory Usage
- 5.7+: Replication
- 5.7+: Variables
- 8.0+: Errors Summary
Who We Are
Who We Are: Sveta Smirnova

- MySQL Support engineer
- Author of
  - MySQL Troubleshooting
  - JSON UDF functions
  - FILTER clause for MySQL
- Speaker
  - Percona Live, OOW, Fosdem, DevConf, HighLoad...
Who We Are: Alexander Rubin

- Principal Consultant in Architecture & Projects
- With MySQL for > 12 years
- Speaker
  - Percona Live, OOW, HighLoad...
Performance Schema Configuration
<table>
<thead>
<tr>
<th>Version</th>
<th>Tables</th>
<th>Instructions</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>52</td>
<td>554</td>
<td>31</td>
</tr>
<tr>
<td>5.7</td>
<td>87</td>
<td>1019</td>
<td>42</td>
</tr>
<tr>
<td>8.0</td>
<td>101</td>
<td>1193</td>
<td>44</td>
</tr>
</tbody>
</table>
Performance Schema Defaults

- **Defaults**
  - ON

5.7: Only global, thread, statements and transactions instrumentation

8.0: Memory and MDL
Performance Schema Defaults

- **Defaults**
  - **ON**
  - **5.7:** Only global, thread, statements and transactions instrumentation
  - **8.0:** Memory and MDL

- All other instruments/consumers disabled
Prepare

• We will turn required instrumentation ON for each exercise separately
We will turn required instrumentation ON for each exercise separately

We will use pattern

```sql
update performance_schema.setup_consumers set enabled='yes'
where name like 'OUR_REQUIREMENT_%';

update performance_schema.setup_instruments set enabled='yes', timed='yes'
where name like 'OUR_REQUIREMENT_%';
```
Prepare

- We will turn required instrumentation ON for each exercise separately
- Or easier

  call sys.ps_setup_enable_consumer(YOUR_CONSUMER);

  call sys.ps_setup_enable_instrument(YOUR_INSTRUMENT);
Prepare

- We will turn required instrumentation ON for each exercise separately
- Be careful!
  - They are memory and CPU intensive
  - Do not turn them all ON until needed
5.6+: Statements Instrumentation
What Can We Discover?

Why statements are slow?
- Examine more rows than return/change
- Use disk instead of memory
- Full table scan instead of index
- This is not full list!
What Can We Discover?

- Why statements are slow?
- Performance Schema has
  - Per-query statistics
  - Most evolving stages
Why Statements are Slow?

- `events_statements_*` and `prepared_statements_instances` tables
  - Important field names
    - CREATED_TMP_DISK_TABLES
    - CREATED_TMP_TABLES
    - SELECT_FULL_JOIN
    - SELECT_RANGE_CHECK
    - SELECT_SCAN
    - SORT_MERGE_PASSES
    - SORT_SCAN
Why Statements are Slow?

- `events_statements_*` and `prepared_statements_instances` tables
- Views in `sys` schema
  - Important view names
    - `statement_analysis`
    - `statements_with_full_table_scans`
    - `statements_with_runtimes_in_95th_percentile`
    - `statements_with_sorting`
    - `statements_with_temp_tables`
    - `statements_with_errors_or_warnings`
Why Statements are Slow?

- `events_statements_*` and `prepared_statements_instances` tables
- Views in `sys` schema
- Digest tables
  - Combined statistics
    - `events_statements_summary_by_account_by_event_name`
    - `events_statements_summary_by_host_by_event_name`
    - `events_statements_summary_by_thread_by_event_name`
    - `events_statements_summary_by_user_by_event_name`
    - `events_statements_summary_global_by_event_name`
    - 5.7+: `events_statements_summary_by_program`
Why Statements are Slow?

- `events_statements_*` and `prepared_statements_instances` tables
- Views in `sys` schema
- Digest tables
  - `events_statements_summary_by_digest`
    - `SCHEMA_NAME`
    - `DIGEST`
    - `DIGEST_TEXT`
    - `8.0+: QUERY_SAMPLE_TEXT`
Which Queries Do Not Use Indexes?

mysql> SELECT THREAD_ID TID, SUBSTR(SQL_TEXT, 1, 50) SQL_TEXT, ROWS_SENT RS,
-> ROWS_EXAMINED RE, CREATED_TMP_TABLES, NO_INDEX_USED, NO_GOOD_INDEX_USED
-> FROM performance_schema.events_statements_history
-> WHERE NO_INDEX_USED=1 OR NO_GOOD_INDEX_USED=1\G

************************** 1. row **************************

TID: 10124
SQL_TEXT: select emp_no, first_name, last_name from employee
RS: 97750
RE: 397774
CREATED_TMP_TABLES: 0
NO_INDEX_USED: 1
NO_GOOD_INDEX_USED: 0
...
Take it Easy: Index Usage with sys Schema

```
mysql> SELECT query, total_latency, no_index_used_count, rows_sent,  
    -> rows_examined  
    -> FROM sys.statements_with_full_table_scans  
    -> WHERE db='employees' AND query NOT LIKE '%performance_schema%'\G  
********************** 1. row **********************
query: SELECT COUNT ( 'emp_no' ) FROM ... 'emp_no' )  
    WHERE 'title' = ?
  
total_latency: 805.37 ms
no_index_used_count: 1
  
rows_sent: 1
rows_examined: 397774
...
```
Take it Easy: with Digest Tables

mysql> select DIGEST, DIGEST_TEXT, COUNT_STAR, SUM CREATED_TMP_DISK_TABLES,
   -> SUM_SELECT_FULL_JOIN, SUM_SELECT_RANGE, SUM_SELECT_SCAN, SUM_NO_INDEX_USED,
   -> SUM_ROWS_SENT, SUM_ROWS_EXAMINED
   -> from events_statements_summary_by_digest where SUM_NO_INDEX_USED > 0\G

*************************** 1. row ***************************
DIGEST: 3884185b07312b354c4918f2368d8fe2c431aeb8e39bf8ff5c3dc6d
DIGEST_TEXT: SELECT ‘c’ FROM ‘sbtest1‘ WHERE ‘id‘ BETWEEN ? AND ?
COUNT_STAR: 1501791
SUM CREATED_TMP_DISK_TABLES: 0
SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_RANGE: 1501840
SUM_SELECT_SCAN: 4
SUM_NO_INDEX_USED: 4
SUM_ROWS_SENT: 150872400
SUM_ROWS_EXAMINED: 152872000

...
Statements: practice

- Login into EC2 instance
  - Login: see your card
  - Password: see your card

- Run load ./statements_summary.sh
- CALL help_task()
- CALL help_solve()

- We need to find slowest queries:
  - With largest response time
  - With large scanned rows number
  - Not using indexes
Run load

./statements_summary.sh
CALL help_task()
CALL help_solve()

We need to find slowest queries:
- With largest response time
- With large scanned rows number
- Not using indexes
- Creating temporary tables
Statements Deep Dive

- `events_stages_* tables`
Statements Deep Dive

- `events_stages_*` tables
- Same information as in table `INFORMATION_SCHEMA.PROCESSLIST` or `SHOW PROCESSLIST` output
  - `init`
  - `executing`
  - `Opening tables`
Statements Deep Dive

- `events_stages_*` tables
- Same information as in table `INFORMATION_SCHEMA.PROCESSLIST` or `SHOW PROCESSLIST` output
  - `init`
  - `executing`
  - Opening tables
- Replacement for `SHOW PROFILE`
Statements Deep Dive

- *events_stages_* tables
- Same information as in table INFORMATION_SCHEMA.PROCESSLIST or SHOW PROCESSLIST output
  - init
  - executing
  - Opening tables
- Replacement for SHOW PROFILE
- Only server-level
  - No storage engine information!
Stages Shortcuts

- Everything, related to temporary tables
  - `EVENT_NAME LIKE 'stage/sql/%tmp%'`

- Everything, related to locks
  - `EVENT_NAME LIKE 'stage/sql/%lock%'

- Everything in state "Waiting for"
  - `EVENT_NAME LIKE 'stage/%/Waiting for%'

- Frequently met issues
Stages Shortcuts

- Everything, related to temporary tables
- Everything, related to locks
- Everything in state "Waiting for"
- Frequently met issues
  - EVENT_NAME='stage/sql/freeing items'
  - EVENT_NAME='stage/sql/Sending data'
  - EVENT_NAME='stage/sql/cleaning up'
  - EVENT_NAME='stage/sql/closing tables'
  - EVENT_NAME='stage/sql/end'
Stages Example: Which Stage Run Critically Long?

mysql> SELECT eshl.event_name, sql_text, eshl.timer_wait/1000000000000 w_s
    -> FROM performance_schema.events_stages_history_long eshl
    -> JOIN performance_schema.events_statements_history_long esthl
    -> ON (eshl.nesting_event_id = esthl.event_id)
    -> WHERE eshl.timer_wait > 1*1000000000000\G

*************************** 1. row ***************************
event_name: stage/sql/Sending data
sql_text: SELECT COUNT(emp_no) FROM employees JOIN salaries USING(emp_no)
    WHERE hire_date=from_date
w_s: 0.8170
1 row in set (0.00 sec)
5.7+: Prepared Statements
• Contains current prepared statements
Table prepared_statements_instances

- Contains current prepared statements
- Statistics by
  - Which thread owns the statement
  - How many times executed
  - Optimizer statistics, similar to events_statements_*
Example: Prepared Statement

```sql
mysql1> prepare stmt from 'select count(*) from employees where hire_date > ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
mysql1> set @hd='1995-01-01';
Query OK, 0 rows affected (0.00 sec)
mysql1> execute stmt using @hd;
```

```
+----------+
| count(*) |
+----------+
| 34004    |
+----------+
1 row in set (1.44 sec)
```
Example: Prepared Statement

```sql
mysql1> prepare stmt from 'select count(*) from employees where hire_date > ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
mysql1> set @hd='1995-01-01';
Query OK, 0 rows affected (0.00 sec)
mysql1> execute stmt using @hd;
+----------+
| count(*)  |
+----------+
| 34004    |
+----------+
1 row in set (1.44 sec)
```

- **Try EXECUTE with different values**
Example: diagnosis

```
mysql2> select statement_name, sql_text, owner_thread_id, count_reprepare,
  -> count_execute, sum_timer_execute from prepared_statements_instances\G
*************************** 1. row ***************************
  statement_name: stmt
    sql_text: select count(*) from employees where hire_date > ?
  owner_thread_id: 22
  count_reprepare: 0
  count_execute: 3
  sum_timer_execute: 4156561368000
1 row in set (0.00 sec)

mysql1> drop prepare stmt;
Query OK, 0 rows affected (0.00 sec)

mysql2> select * from prepared_statements_instances\G
Empty set (0.00 sec)
```
Prepared Statements: practice

- Run load
  ```bash
  ./prepared.sh
  CALL help_task()
  CALL help_solve()
  ```

- We need to find out how effective is prepared statement
5.7+: Stored Routines Instrumentation
### New Instruments

```sql
mysql> select * from setup_instruments where name like 'statement/sp%';
+----------------------------+
<table>
<thead>
<tr>
<th>NAME</th>
<th>ENABLED</th>
<th>TIMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement/sp/stmt</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/set</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/set_trigger_field</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/jump</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/jump_if_not</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/freturn</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/hpush_jump</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>statement/sp/hpop</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
+----------------------------+
```

16 rows in set (0.00 sec)
Stored Routines Instrumentation

- What happens inside the routine
Stored Routines Instrumentation

- What happens inside the routine
- Queries, called from the routine
  - statement/sp/stmt
We will use this procedure

CREATE DEFINER='root'@'localhost' PROCEDURE 'sp_test'(val int)
BEGIN
  DECLARE CONTINUE HANDLER FOR 1364, 1048, 1366
  BEGIN
    INSERT IGNORE INTO t1 VALUES('Some string');
    GET STACKED DIAGNOSTICS CONDITION 1 @stacked_state = RETURNED_SQLSTATE;
    GET STACKED DIAGNOSTICS CONDITION 1 @stacked_msg = MESSAGE_TEXT;
  END;
  INSERT INTO t1 VALUES(val);
END
We will use this procedure

```
CREATE DEFINER='root'@'localhost' PROCEDURE 'sp_test'(val int)
BEGIN
  DECLARE CONTINUE HANDLER FOR 1364, 1048, 1366
  BEGIN
    INSERT IGNORE INTO t1 VALUES('Some string');
    GET STACKED DIAGNOSTICS CONDITION 1 @stacked_state = RETURNED_SQLSTATE;
    GET STACKED DIAGNOSTICS CONDITION 1 @stacked_msg = MESSAGE_TEXT;
  END;
  INSERT INTO t1 VALUES(val);
END
```

When HANDLER called?
Correct Value

mysql> call sp_test(1);
Query OK, 1 row affected (0.07 sec)

mysql> select thread_id, event_name, sql_text from events_statements_history
    -> where event_name like 'statement/sp%';

+-----------+-------------------------+----------------------------+
| thread_id | event_name          | sql_text            |
+-----------+-------------------------+----------------------------+
| 24        | statement/sp/hpush_jump | NULL                |
| 24        | statement/sp/stmt      | INSERT INTO t1 VALUES(val) |
| 24        | statement/sp/hpop      | NULL                |
+-----------+-------------------------+----------------------------+
3 rows in set (0.00 sec)
mysql> call sp_test(NULL);
Query OK, 1 row affected (0.07 sec)

mysql> select thread_id, event_name, sql_text from events_statements_history
    -> where event_name like 'statement/sp%';

+-----------+-------------------------+-------------------------------------------+
<table>
<thead>
<tr>
<th>thread_id</th>
<th>event_name</th>
<th>sql_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>statement/sp/hpush_jump</td>
<td>NULL</td>
</tr>
<tr>
<td>24</td>
<td>statement/sp(stmt</td>
<td>INSERT INTO t1 VALUES(val)</td>
</tr>
</tbody>
</table>
| 24        | statement/sp(stmt | INSERT IGNORE INTO t1 VALUES('Some str...
| 24        | statement/sp(stmt | GET STACKED DIAGNOSTICS CONDITION 1 @s...
| 24        | statement/sp(stmt | GET STACKED DIAGNOSTICS CONDITION 1 @s...
| 24        | statement/sp/hreturn | NULL |
| 24        | statement/sp/hpop | NULL |
+-----------+-------------------------+-------------------------------------------+
• Run load
  
  ./crazy_timing.sh
  CALL help_task();
  CALL help_solve();
  CALL task_prepare();

• We need to find out why procedure takes different time each run

• For better output set pager to less:
  
  mysql> \P less
5.7+: Locks Diagnostic
5.7+: MDL

- Table METADATA_LOCKS
5.7+: MDL

- Table METADATA_LOCKS
- Which thread is waiting for a lock
Table `METADATA_LOCKS`

- Which thread is waiting for a lock
- Which thread holds the lock

5.7+: MDL
5.7+: MDL

- Table METADATA_LOCKS
- Which thread is waiting for a lock
- Which thread holds the lock
- Not only for tables:
  - GLOBAL, SCHEMA, TABLE, FUNCTION, PROCEDURE, EVENT, COMMIT, USER LEVEL
  - LOCK, TABLESPACE
mysql> select processlist_id, object_type, lock_type, lock_status, source 
    -> from metadata_locks join threads on (owner_thread_id=thread_id) 
    -> where object_schema='employees' and object_name='titles'

*************************** 1. row ***************************
processlist_id: 4
  object_type: TABLE
  lock_type: EXCLUSIVE
  lock_status: PENDING -- waits
  source: mdl.cc:3263
*************************** 2. row ***************************
processlist_id: 5
  object_type: TABLE
  lock_type: SHARED_READ
  lock_status: GRANTED -- holds
  source: sql_parse.cc:5707
MDL: practice

- Run load
  
  ```
  ./test1.sh
  CALL help_task();
  CALL help_solve();
  CALL task_prepare();
  ```

- We need to find out what prevents ALTER from finishing
8.0.+: Data Locks

- Information about locks, held by engine
8.0.+: Data Locks

- Information about locks, held by engine
- Only for engines with own locking models
8.0.+: Data Locks

- Information about locks, held by engine
- Only for engines with own locking models
- Currently only InnoDB
8.0.+: Data Locks

- Information about locks, held by engine
- Only for engines with own locking models
- Currently only InnoDB
- Replacement for IS tables
  - INNODB_LOCKS
  - INNODB_LOCK_WAITS
**Which lock is held**

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGINE</td>
<td>INNODB</td>
</tr>
<tr>
<td>ENGINE_LOCK_ID</td>
<td>2408:0:393:2</td>
</tr>
<tr>
<td>ENGINE_TRANSACTION_ID</td>
<td>2408</td>
</tr>
<tr>
<td>THREAD_ID</td>
<td>34</td>
</tr>
<tr>
<td>OBJECT_SCHEMA</td>
<td>test</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>t</td>
</tr>
<tr>
<td>INDEX_NAME</td>
<td>PRIMARY</td>
</tr>
<tr>
<td>LOCK_TYPE</td>
<td>RECORD</td>
</tr>
<tr>
<td>LOCK_MODE</td>
<td>X</td>
</tr>
<tr>
<td>LOCK_STATUS</td>
<td>GRANTED</td>
</tr>
<tr>
<td>LOCK_DATA</td>
<td>12345</td>
</tr>
</tbody>
</table>

**4. row**

---

**Table DATA_LOCKS**

---
Which lock is held
Which lock is requested

***************************************************************************
ENGINE: INNODB
ENGINE_LOCK_ID: 2409:0:393:2
ENGINE_TRANSACTION_ID: 2409
THREAD_ID: 36
OBJECT_SCHEMA: test
OBJECT_NAME: t
INDEX_NAME: PRIMARY
LOCK_TYPE: RECORD
LOCK_MODE: X
LOCK_STATUS: WAITING
LOCK_DATA: 12345
***************************************************************************
**Table DATA_LOCKS**

- Which lock is held
- Which lock is requested
- Both record-level and table level

```
p_s> select * from data_locks\G
*************************** 1. row ***************************
...                        
   LOCK_TYPE: TABLE
   LOCK_MODE: IX
   LOCK_STATUS: GRANTED
   LOCK_DATA: NULL
*************************** 2. row ***************************
...                        
   LOCK_TYPE: RECORD
```
Table DATA_LOCK_WAITS

- Maps lock waits with granted locks
Maps lock waits with granted locks
Only granted blocking other transactions

```
p_s> select ENGINE, ... from data_lock_waits\G
*************************** 1. row ***************************
ENGINE: INNODB
REQUESTING_ENGINE_LOCK_ID: 2409:0:393:2
REQUESTING_ENGINE_TRANSACTION_ID: 2409
REQUESTING_THREAD_ID: 36
BLOCKING_ENGINE_LOCK_ID: 2408:0:393:2
BLOCKING_ENGINE_TRANSACTION_ID: 2408
BLOCKING_THREAD_ID: 34
1 row in set (0.01 sec)
```
New Information

- Partition
- Subpartition
- Lock data
- Requesting and blocking thread id
In sys Schema

- **View innodb_lock_waits**
In sys Schema

- **View `innodb_lock_waits`**
- **Takes additional information from `INFORMATION_SCHEMA.INNODB_TRX`**
View innodb_lock_waits

sys> select locked_table, ... -> from innodb_lock_waits\G

*************************** 1. row ***************************
locked_table: 'test'.'t' blocking_pid: 4
locked_index: PRIMARY blocking_query: NULL
locked_type: RECORD blocking_trx_rows_locked: 1
waiting_trx_rows_locked: 1 blocking_trx_rows_modified: 1
waiting_trx_rows_modified: 0 sql_kill_blocking_query: KILL QUERY 4
waiting_pid: 6 sql_kill_blocking_connection: KILL 4

waiting_query: UPDATE t SET f='bar' WHERE id=12345
Data Locks: Practice

- **Run load**
  ```
  ./data_locks.sh
  CALL help_task()\G
  CALL help_solve()\G
  ```

- **We need to find**
  - Which transaction holds the lock
  - What is the missed statement
  - Which row is locked
  - Which partition is locked
5.7+: Memory Usage
Memory Diagnostic

- Memory, used by internal `mysqld` structures
Memory Diagnostic

- Memory, used by internal `mysqld` structures
- Aggregated by
  - Global
  - Thread
  - Account
  - Host
  - User
Memory Diagnostic

- Memory, used by internal `mysqld` structures
- Aggregated by
  - Global
  - Thread
  - Account
  - Host
  - User
- Nice views in `sys` schema
### Memory Usage by Thread

```sql
mysql> select thread_id tid, user, current_allocated ca, total_allocated -> from sys.memory_by_thread_by_current_bytes;
```

<table>
<thead>
<tr>
<th>tid</th>
<th>user</th>
<th>ca</th>
<th>total_allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sql/main</td>
<td>2.53 GiB</td>
<td>2.69 GiB</td>
</tr>
<tr>
<td>150</td>
<td>root@127.0.0.1</td>
<td>4.06 MiB</td>
<td>32.17 MiB</td>
</tr>
<tr>
<td>146</td>
<td>sql/slave_sql</td>
<td>1.31 MiB</td>
<td>1.44 MiB</td>
</tr>
<tr>
<td>145</td>
<td>sql/slave_io</td>
<td>1.08 MiB</td>
<td>2.79 MiB</td>
</tr>
<tr>
<td>60</td>
<td>innodb/io_read_thread</td>
<td>0 bytes</td>
<td>384 bytes</td>
</tr>
<tr>
<td>139</td>
<td>innodb/srv_purge_thread</td>
<td>-328 bytes</td>
<td>754.21 KiB</td>
</tr>
<tr>
<td>69</td>
<td>innodb/io_write_thread</td>
<td>-1008 bytes</td>
<td>34.28 KiB</td>
</tr>
<tr>
<td>68</td>
<td>innodb/io_write_thread</td>
<td>-1440 bytes</td>
<td>298.05 KiB</td>
</tr>
<tr>
<td>74</td>
<td>innodb/io_write_thread</td>
<td>-1656 bytes</td>
<td>103.55 KiB</td>
</tr>
<tr>
<td>4</td>
<td>innodb/io_log_thread</td>
<td>-2880 bytes</td>
<td>132.38 KiB</td>
</tr>
</tbody>
</table>
```

145 rows in set (2.65 sec)
Threads Statistics

```
mysql> select * from sys.memory_by_thread_by_current_bytes
    -> order by current_allocated desc\G
*************************** 1. row ***************************
    thread_id: 152
      user: lj@127.0.0.1
 current_count_used: 325
 current_allocated: 36.00 GiB
 current_avg_alloc: 113.43 MiB
 current_max_alloc: 36.00 GiB
    total_allocated: 37.95 GiB
...
```

• Find threads, eating memory, in a second!
RAW Performance Schema tables

- `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`
RAW Performance Schema tables

- 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

You must enable memory instrumentation!
RAW Performance Schema tables

- `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`

You must enable memory instrumentation!

sys schema includes user name
Users in `sys.memory`* tables

- **NAME@HOST** - regular user
Users in sys.memory_* tables

- **NAME@HOST** - regular user
- **System users**
  - sql/main
  - innodb/*
  - ...

Users in sys.memory_

- NAME@HOST - regular user
- System users
  - sql/main
  - innodb/*
  - ...
- Data comes from table THREADS
Memory Usage: practice

- Run load
  ```sh
  ./test2.sh
  CALL help_task();
  CALL help_solve();
  CALL task_prepare();
  ```

- We need to find out how much memory uses SysBench load, running in parallel
- To identify how much RAM used by whole server run
  ```sql
  select * from sys.memory_global_total;
  ```
5.7+: Replication
Major Improvements

- Data from `SHOW SLAVE STATUS` available in `replication_*` tables
Major Improvements

- Data from `SHOW SLAVE STATUS` available in replication_* tables
- Support of Replication Channels (Multi-master slave)
Major Improvements

- Data from `SHOW SLAVE STATUS` available in replication_* tables
- Support of Replication Channels (Multi-master slave)
- More instruments for GTID
SLAVE STATUS

- No need to parse `SHOW output`
SLAVE STATUS

- No need to parse `SHOW` output
- Configuration
  - `replication_connection_configuration`
  - `replication_applier_configuration`
SLAVE STATUS

- No need to parse `SHOW` output
- Configuration
- IO thread
  - `replication_connection_status`
SLAVE STATUS

- No need to parse `SHOW output`
- Configuration
- IO thread
- SQL thread
  - `replication_applier_status`
  - `replication_applier_status_by_coordinator`
    - MTS only
  - `replication_applier_status_by_worker`
SLAVE STATUS

**Configuration**

```sql
mysql> select * from replication_connection_configuration
    -> join replication_applier_configuration using(channel_name)
G
*************************** 1. row ***************************
CHANNEL_NAME:
    HOST: 127.0.0.1
    PORT: 13000
    USER: root
NETWORK_INTERFACE:
    AUTO_POSITION: 1
    SSL_ALLOWED: NO
    SSL_CA_FILE:
...
CHANNEL_NAME:
    DESIRED_DELAY: 0
```
SLAVE STATUS

State of IO Thread

mysql> select * from replication_connection_status\G
*************************** 1. row ***************************
CHANNEL_NAME: 
GROUP_NAME: 
SOURCE_UUID: d0753e78-14ec-11e5-b3fb-28b2bd7442fd
THREAD_ID: 21
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 17
LAST_HEARTBEAT_TIMESTAMP: 2015-06-17 15:49:08
RECEIVED_TRANSACTION_SET:
  LAST_ERROR_NUMBER: 0
  LAST_ERROR_MESSAGE: 
  LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (0.00 sec)
Performance Schema: State of SQL Thread

- Coordinator thread for multiple workers

```
mysql> select * from replication_applier_status join
    -> replication_applier_status_by_coordinator using(channel_name)
G
*************************** 1. row *********************
    CHANNEL_NAME: 
    SERVICE_STATE: ON
    REMAINING_DELAY: NULL
    COUNT_TRANSACTIONS_RETRIES: 0
    THREAD_ID: 22
    SERVICE_STATE: ON
    LAST_ERROR_NUMBER: 0
    LAST_ERROR_MESSAGE: 
    LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (0.00 sec)
```
Performance Schema: State of SQL Thread

- Coordinator thread for multiple workers
- Other cases

```sql
mysql> select * from replication_applier_status join replication_applier_status_by_worker using(channel_name)
G
*************************** 1. row ***********************
  CHANNEL_NAME: master-1
  SERVICE_STATE: OFF
  REMAINING_DELAY: NULL
  COUNT_TRANSACTIONS_RETRIES: 0
  WORKER_ID: 0
  THREAD_ID: NULL
  SERVICE_STATE: OFF
  LAST_SEEN_TRANSACTION: ANONYMOUS
  LAST_ERROR_NUMBER: 1032
  LAST_ERROR_MESSAGE: Could not execute Update_rows...
```
Performance Schema: State of SQL Thread

- Coordinator thread for multiple workers
- Other cases

************************************************ 2. row ************************************************

    CHANNEL_NAME: master-2
    SERVICE_STATE: ON
    REMAINING_DELAY: NULL
    COUNT_TRANSACTIONS RETRIES: 0
    WORKER_ID: 0
    THREAD_ID: 42
    SERVICE_STATE: ON
    LAST_SEEN_TRANSACTION: ANONYMOUS
    LAST_ERROR_NUMBER: 0
    LAST_ERROR_MESSAGE: 
    LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00

2 rows in set (0.00 sec)
GTID Diagnostics

- RECEIVED_TRANSACTION_SET in table replication_connection_status
GTID Diagnostics

- RECEIVED_TRANSACTION_SET in table replication_connection_status
- LAST_SEEN_TRANSACTION in replication_applier_status_by_worker
GTID: All in One Place

- Single-threaded slave

```sql
mysql> select cs.CHANNEL_NAME, cs.SOURCE_UUID, cs.RECEIVED_TRANSACTION_SET, 
    -> asw.LAST_SEEN_TRANSACTION, aps.SERVICE_STATE from 
    -> replication_connection_status cs join replication_applier_status_by_worker 
    -> asw using(channel_name) join replication_applier_status aps 
    -> using(channel_name) \G
```

```
*************************** 1. row ***************************
  CHANNEL_NAME: 
  SOURCE_UUID: 9038967d-7164-11e6-8c88-30b5c2208a0f
  RECEIVED_TRANSACTION_SET: 9038967d-7164-11e6-8c88-30b5c2208a0f:1-2
  LAST_SEEN_TRANSACTION: 9038967d-7164-11e6-8c88-30b5c2208a0f:2
  SERVICE_STATE: ON
```

1 row in set (0,00 sec)
GTID: All in One Place

- Single-threaded slave
- Multi-threaded

******************** 1. row ********************
THREAD_ID: 30
SERVICE_STATE: ON
RECEIVED_TRANSACTION_SET: 9038967d-7164-11e6-8c88-30b5c2208a0f:1-3
LAST_SEEN_TRANSACTION: ...

******************** 8. row ********************
THREAD_ID: 37
SERVICE_STATE: ON
RECEIVED_TRANSACTION_SET: 9038967d-7164-11e6-8c88-30b5c2208a0f:1-3
LAST_SEEN_TRANSACTION: 9038967d-7164-11e6-8c88-30b5c2208a0f:3

8 rows in set (0.00 sec)
More Diagnostic

- Tables in mysql schema
  - slave_master_info
  - slave_relay_log_info
  - slave_worker_info
  - Join with Performance Schema tables
More Diagnostic

- Tables in mysql schema
  - slave_master_info
  - slave_relay_log_info
  - slave_worker_info
  - Join with Performance Schema tables

- New instruments
  - memory
  - wait
  - stage
Replication: practice

- Run load
  ```
  ./repl.sh
  CALL help_task()
  CALL help_solve()
  ```

- Connection commands in another terminal
  - **Master:** /training/sandboxes/rsandbox_Percona-Server-5_7_17/m
  - **Slave:** /training/sandboxes/rsandbox_Percona-Server-5_7_17/s1

- We need to find out why replication is broken and fix it
5.7+: Variables
Variables Instrumentation

- Variables
  - global_variables
  - session_variables
  - user_variables_by_thread
  - variables_by_thread
Variables Instrumentation

- Variables
- Status variables
  - `global_status`
  - `session_status`
  - `status_by_[account|host|thread|user]`
Variables Instrumentation

- Variables
- Status variables
- `show_compatibility_56 = 0`
Global and Session Variables

- Same information which is in:
  - `SHOW [GLOBAL] STATUS`
  - `I_S.GLOBAL_VARIABLES`
    - Deprecated in 5.7
    - Removed in 8.0.1
  - `I_S.SESSION_VARIABLES`
    - Deprecated in 5.7
    - Removed in 8.0.1
Global and Session Variables

- Same information which is in
  - `SHOW [GLOBAL] STATUS`
  - `I_S.GLOBAL_VARIABLES`
    - Deprecated in 5.7
    - Removed in 8.0.1
  - `I_S.SESSION_VARIABLES`
    - Deprecated in 5.7
    - Removed in 8.0.1

- Helps to watch session variables changes
Status Variables

- Same information which is in
  - `SHOW [GLOBAL] STATUS`
  - `IS.GLOBAL_STATUS`
    - Deprecated in 5.7
    - Removed in 8.0.1
  - `IS.SESSION_STATUS`
    - Deprecated in 5.7
    - Removed in 8.0.1
Status Variables

mysql> SELECT ss.variable_name, ss.variable_value FROM session_status ss
    -> LEFT JOIN global_status gs USING(variable_name)
    -> WHERE ss.variable_value != gs.variable_value OR gs.variable_value IS NULL;

+----------------------------+----------------+
| variable_name | variable_value |
+----------------------------+----------------+
| Bytes_sent     | 197774         |
| Handler_commit | 0              |
| Handler_external_lock | 44    |
| Handler_read_first | 3     |
| Handler_read_key  | 523          |
| Handler_read_next  | 0           |
| Handler_read_rnd_next | 7241  |
| Opened_table_definitions | 0     |

...
Possible to Group

- `variables_by_thread`
- `status_by_`
  - `account`
  - `host`
  - `thread`
  - `user`
Possible to Group

**variables_by_thread**

```sql
mysql> select * from variables_by_thread where variable_name='tx_isolation';
+-----------+---------------+-----------------+
| THREAD_ID | VARIABLE_NAME | VARIABLE_VALUE |
+-----------+---------------+-----------------+
| 71        | tx_isolation  | REPEATABLE-READ |
| 83        | tx_isolation  | REPEATABLE-READ |
| 84        | tx_isolation  | SERIALIZABLE    |
+-----------+---------------+-----------------+
3 rows in set, 3 warnings (0.00 sec)
```
variable_by_thread

status_by_

mysql> select * from status_by_thread where variable_name='Handler_write';

+-----------+---------------+----------------+
<table>
<thead>
<tr>
<th>THREAD_ID</th>
<th>VARIABLE_NAME</th>
<th>VARIABLE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>Handler_write</td>
<td>94</td>
</tr>
<tr>
<td>83</td>
<td>Handler_write</td>
<td>477</td>
</tr>
<tr>
<td>84</td>
<td>Handler_write</td>
<td>101</td>
</tr>
</tbody>
</table>

3 rows in set (0.00 sec)
• Grouped by connection
• Sometimes can help to find tricky bugs with persistent connections

```sql
mysql> select * from user_variables_by_thread;
+-----------+---------------+----------------+
| THREAD_ID | VARIABLE_NAME | VARIABLE_VALUE |
+-----------+---------------+----------------+
| 71        | baz           | boo            |
| 84        | foo           | bar            |
+-----------+---------------+----------------+
2 rows in set (0.00 sec)
```
8.0+: Variables Info

**VARIABLES_INFO**

- Source of variable
  - COMPiled
  - EXPLICIT
  - COMMAND_LINE
  - DYNAMIC

- Path of option file if specified
- Minimum and maximum values
8.0+: Variables Info

**VARIABLES_INFO**

```sql
mysql> select * from variables_info \G
*************************** 1. row ***************************
VARIABLE_NAME: auto_increment_increment
VARIABLE_SOURCE: COMPILED
VARIABLE_PATH:
  MIN_VALUE: 1
  MAX_VALUE: 65535
*************************** 2. row ***************************
VARIABLE_NAME: basedir
VARIABLE_SOURCE: EXPLICIT
VARIABLE_PATH: /home/sveta/build/mysql-8.0/mysql-test/var/my.cnf
  MIN_VALUE: 0
  MAX_VALUE: 0
...```
8.0+: Variables Info

- **VARIABLES_INFO**
  - Source of variable
    - COMPILTED
    - EXPLICIT
    - COMMAND_LINE
    - DYNAMIC
  - Path of option file if specified
  - Minimum and maximum values

- No variable values in this table!
Run load

./variables.sh
CALL help_task()\G
CALL help_solve()\G
CALL task_prepare();

We need to watch progress of INSERT command, running by stored routine.

Note what there is parallel load, caused by SysBench. We are not interested in its statistics.
8.0+: Errors Summary
Errors Summary Tables

Traditionally aggregated

- `events_errors_summary_by_account_by_error`
- `events_errors_summary_by_host_by_error`
- `events_errors_summary_by_thread_by_error`
- `events_errors_summary_by_user_by_error`
- `events_errors_summary_global_by_error`
Errors Summary Tables

- Traditionally aggregated
- All tables have similar structure

```sql
mysql> DESCR events_errors_summary_global_by_error;
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_NUMBER</td>
<td>int(11)</td>
<td>YES</td>
<td>UNI</td>
<td>NULL</td>
</tr>
<tr>
<td>ERROR_NAME</td>
<td>varchar(64)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>SQL_STATE</td>
<td>varchar(5)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>SUM_ERROR_RAISED</td>
<td>bigint(20) unsigned</td>
<td>NO</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>SUM_ERROR_HANDLED</td>
<td>bigint(20) unsigned</td>
<td>NO</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>FIRST_SEEN</td>
<td>timestamp</td>
<td>YES</td>
<td></td>
<td>0000-00-00 00:00:00</td>
</tr>
<tr>
<td>LAST_SEEN</td>
<td>timestamp</td>
<td>YES</td>
<td></td>
<td>0000-00-00 00:00:00</td>
</tr>
</tbody>
</table>

7 rows in set (0.03 sec)
Errors Summary: Which Accounts Raise More Errors?

```
mysql> select * from events_errors_summary_by_account_by_error
-> where SUM_ERROR_RAISED > 100\G

*************** 1. row ***************  2. row ***************
USER: root  USER: root
HOST: localhost  HOST: localhost
ERROR_NUMBER: 1213  ERROR_NUMBER: 1287
ERROR_NAME: ER_LOCK_DEADLOCK  ERROR_NAME: ER_WARN_DEPRECATED_SYNTAX
SQL_STATE: 40001  SQL_STATE: HY000
SUM_ERROR_RAISED: 221  SUM_ERROR_RAISED: 279
SUM_ERROR_HANDLED: 0  SUM_ERROR_HANDLED: 0
LAST_SEEN: 2016-09-28 01:47:02  LAST_SEEN: 2016-09-28 01:47:05
```

Errors Summary: Which Accounts Raise More Errors?
Special thanks

Nickolay Ihalainen for practice setup idea and 5.7 examples
More information

Blog of MySQL developers team
Mark Leith: author of sys schema
Official reference manual
Webinar ”Performance Schema for MySQL Troubleshooting”
Thank you!

http://www.slideshare.net/SvetaSmirnova
https://twitter.com/svetsmirnova
https://www.linkedin.com/in/alexanderrubin