MariaDB 10.5 New Features for Troubleshooting

Valerii Kravchuk, Principal Support Engineer, MariaDB

vkravchuk@gmail.com

Who am I and What Do I Do?

Valerii (aka **Valeriy**) **Kravchuk**:

- MySQL Support Engineer in MySQL AB, Sun and Oracle, 2005-2012
- Principal Support Engineer in Percona, 2012-2016
- Principal Support Engineer in MariaDB Corporation since March 2016
- **http://mysqlentomologist.blogspot.com** - my blog about MariaDB and MySQL (including some **HowTo**s, not only bugs marketing)
- **https://www.facebook.com/valerii.kravchuk** - my Facebook page
- **http://bugs.mysql.com** - my personal playground
- **@mysqlbugs #bugoftheday**
- **MySQL Community Contributor of the Year 2019**
- I speak about MySQL and MariaDB in public. Some slides from previous talks are here and there…
- "I solve problems", "I drink and I know things"

Disclaimers

- Since September, 2012 I act as an Independent Consultant providing services to different companies
- All views, ideas, conclusions, statements and approaches in my presentations and blog posts are mine and may not be shared by any of my previous, current and future employers, customers and partners
- All examples are either based on public information or are truly fictional and has nothing to do with any real persons or companies. Any similarities are pure coincidence :)
- The information presented is true to the best of my knowledge

What is this session about?

- MariaDB 10.5 new features that may help DBAs and application developers to find out what's going on when a problem occurs:
  - Performance Schema updates to match MySQL 5.7 instrumentation (and add some more)
  - New tables in the INFORMATION_SCHEMA to monitor the internals of a generic thread pool and few new server variables
  - Improvements of ANALYZE for statements
- Some related examples, blog posts and discussions

Performance Schema: 10.5 vs MySQL 5.7 vs 10.4

```
MySQL [information_schema]> select version(), count(*) from tables where
table_schema='performance_schema';
+-----------+----------+
| version() | count(*) |
+-----------+----------+
| 5.7.30    |       87 |
+-----------+----------+
1 row in set (0,001 sec)

MariaDB [information_schema]> select version(), count(*) from tables where
table_schema='performance_schema';
+----------------+----------+
| version()      | count(*) |
+----------------+----------+
| 10.5.6-MariaDB |       80 | -- was 52 in 10.4
+----------------+----------+
1 row in set (0,060 sec)

MySQL [information_schema]> select version(), count(*) from
performance_schema.global_variables where variable_name like 'performance%';
+-----------+----------+
| version() | count(*) |
+-----------+----------+
| 5.7.30    |       42 | -- was 32 in 10.4, 42 in 10.5
+-----------+----------+
1 row in set (0,002 sec)
```

What's new in Performance Schema?

- Memory instrumentation
- Metadata locking (MDL) instrumentation
- Prepared statements instrumentation
- Status variables instrumentation
- Stored procedures instrumentation
- SX-locks instrumentation
- Transactions instrumentation
- User variables instrumentation
- Replication-related tables added
- Now some memory for P_S is allocated dynamically

P_S Memory Instrumentation: Instruments

- 270 additional instruments (not properly documented, see MDEV-23436 and this blog post):

```
MariaDB [performance_schema]> select name from
performance_schema.setup_instruments where name like 'memory%';
+-----------------------------------------------------------------+
| name                                                            |
+-----------------------------------------------------------------+
| memory/performance_schema/mutex_instances                       |
| memory/performance_schema/rwlock_instances                      |
| memory/performance_schema/cond_instances                        |
| memory/performance_schema/file_instances                        |
...
| memory/sql/udf_mem                                              |
+-----------------------------------------------------------------+
270 rows in set (0,001 sec)
```

P_S Memory Instrumentation: Summary Tables

- 5 summary tables
- KB does not help much with them, so I add some hints here:

```
MariaDB [performance_schema]> show tables like '%memory%';
+----------------------------------------+
| Tables_in_performance_schema (%memory%) |
+----------------------------------------+
| memory_summary_by_account_by_event_name | -- user, host
| memory_summary_by_host_by_event_name    | -- host char(60)
| memory_summary_by_thread_by_event_name  | -- threads.thread_id
| memory_summary_by_user_by_event_name    | -- user char(32)
| memory_summary_global_by_event_name     |
+----------------------------------------+
5 rows in set (0,019 sec)
```

P_S Memory Instrumentation: Tables Structure

- Common columns (see also MySQL 5.7 manual):

```
MariaDB [performance_schema]> desc
memory_summary_global_by_event_name;
...
| EVENT_NAME                  | varchar(128)         ...
| COUNT_ALLOC                 | bigint(20) unsigned ...
| COUNT_FREE                  | bigint(20) unsigned ...
| SUM_NUMBER_OF_BYTES_ALLOC   | bigint(20) unsigned ...
| SUM_NUMBER_OF_BYTES_FREE    | bigint(20) unsigned ...
| LOW_COUNT_USED              | bigint(20)          ...
| CURRENT_COUNT_USED          | bigint(20)          ...
| HIGH_COUNT_USED             | bigint(20)          ...
| LOW_NUMBER_OF_BYTES_USED    | bigint(20)          ...
| CURRENT_NUMBER_OF_BYTES_USED | bigint(20)         ...
| HIGH_NUMBER_OF_BYTES_USED   | bigint(20)          ...
```

P_S Memory Instrumentation: Example

- Let's see what memory was allocated most often for:

```
MariaDB [performance_schema]> select * from
memory_summary_global_by_event_name order by count_alloc desc
limit 1\G
*************************** 1. row ***************************
                 EVENT_NAME: memory/sql/QUICK_RANGE_SELECT::alloc
               COUNT_ALLOC: 147976
                COUNT_FREE: 147976
SUM_NUMBER_OF_BYTES_ALLOC: 600190656
 SUM_NUMBER_OF_BYTES_FREE: 600190656
           LOW_COUNT_USED: 0
       CURRENT_COUNT_USED: 0
          HIGH_COUNT_USED: 68
  LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 0
HIGH_NUMBER_OF_BYTES_USED: 275808
1 row in set (0,069 sec)
```

P_S Memory Instrumentation in MariaDB 10.5

- The implementation is different vs MySQL (MDEV-22841):
  - In MariaDB, there is an array of PSI keys, and the index of the key is determined at the compile time (derived from __FILE__ using some constexpr C++11 magic)
  - MariaDB has more keys, because MySQL did not instrument some files, even if they were using the UT_NEW allocator.
- Memory for **performance_schema** may now be allocated dynamically after startup
- We can see it from **performance_schema** (demo):

```
openxs@ao756:~/dbs/maria10.5$ bin/mysql
--socket=/tmp/mariadb105.sock -e"select
sum(CURRENT_NUMBER_OF_BYTES_USED) used from
performance_schema.memory_summary_global_by_event_name where
event_name like 'memory/performance%'"
```

P_S Memory Instrumentation: Problems

- Negative values in summary tables (that question correctness of instrumentation or summarizing)
- This is not only MariaDB 10.5 specific:
  - Bug #81804 - “Memory leak or bug in P_S instrumentation for InnoDB threads”
  - Bug #79285 - “Undocumented negative low estimates in p_s.memory_summary_global_by_event_name”
- But some cases seem to be different/unique:
  - MDEV-23934 - "Wrong (negative) values in P_S memory summary tables for memory/sql/global_system_variables"
  - MDEV-23936 - "Wrong (negative) values in P_S memory summary tables for memory/innodb/std"

Performance Schema: MDL Instrumentation

- There are different ways to study metadata locks…
- In MariaDB 10.5 we can now use performance_schema:

```
MariaDB [performance_schema]> show tables like '%metadata%';
+-----------------------------------------+
| Tables_in_performance_schema (%metadata%) |
+-----------------------------------------+
| metadata_locks                          |
+-----------------------------------------+
1 row in set (0,001 sec)
MariaDB [performance_schema]> select * from setup_instruments where
name like 'wait/lock/metadata%';
+----------------------------+---------+-------+
| NAME                       | ENABLED | TIMED |
+----------------------------+---------+-------+
| wait/lock/metadata/sql/mdl | NO      | NO    |
+----------------------------+---------+-------+
1 row in set (0,001 sec)
```

MDL Instrumentation: Basic Usage

- Enable:

```
MariaDB [performance_schema]> update setup_instruments set
enabled='YES', timed='YES' where name like 'wait/lock/metadata%';
Query OK, 1 row affected (0,016 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- Check:

```
MariaDB [performance_schema]> select * from metadata_locks\G
*************************** 1. row ***************************
          OBJECT_TYPE: TABLE
        OBJECT_SCHEMA: performance_schema
          OBJECT_NAME: metadata_locks
OBJECT_INSTANCE_BEGIN: 139893728670576
            LOCK_TYPE: SHARED_READ
        LOCK_DURATION: TRANSACTION
          LOCK_STATUS: GRANTED
               SOURCE:
     OWNER_THREAD_ID: 129 -- join to p_s.threads.thread_id
      OWNER_EVENT_ID: 1
```

MDL Instrumentation: Problems

- I summarized my experience in the blog post...
- Let's check a quick demo related to a real life use case of **set global read_only=1** blocked…
- MariaDB 10.4 introduced backup locks and **BACKUP STAGE** statements. As a result we have OBJECT_TYPE = BACKUP and some metadata locks are reported differently than in MySQL
- MDEV-23882 - "Document possible LOCK_TYPE values for metadata locks"
- MDEV-23827 - "performance_schema.metadata_locks.source column is always empty".

Performance Schema: PS Instrumentation

- *Active* prepared statements are instrumented by default:

```
MariaDB [performance_schema]> show tables like '%prepare%';
+----------------------------------------+
| Tables_in_performance_schema (%prepare%) |
+----------------------------------------+
| prepared_statements_instances          |
+----------------------------------------+
1 row in set (0,001 sec)
MariaDB [performance_schema]> select * from setup_instruments where name
like 'statement/%/prepare%' or name like 'statement/%/execute%';
+---------------------------------+---------+-------+
| NAME                            | ENABLED | TIMED |
+---------------------------------+---------+-------+
| statement/sql/prepare_sql       | YES     | YES   | -
mysql_stmt_prepare()
| statement/sql/execute_sql       | YES     | YES   | -
mysql_stmt_execute()
| statement/sql/execute_immediate | YES     | YES   |
| statement/com/Prepare           | YES     | YES   | - PREPARE
| statement/com/Execute           | YES     | YES   | - EXECUTE
+---------------------------------+---------+-------+
5 rows in set (0,001 sec)
```

Prepared Statements Instrumentation: Example

- Let's run some **sysbench** test and check (demo):

```
MariaDB [(none)]> select count(*) from prepared_statements_instances;
+-----------+
| count(*) |
+-----------+
|   204    |
+-----------+
1 row in set (0,001 sec)

MariaDB [(none)]> select * from
performance_schema.prepared_statements_instances limit 1\G
*************************** 1. row ***************************
    OBJECT_INSTANCE_BEGIN: 139894074271256
             STATEMENT_ID: 18
             STATEMENT_NAME: NULL
                 SQL_TEXT: COMMIT
         OWNER_THREAD_ID: 234
             OWNER_EVENT_ID: 3
         OWNER_OBJECT_TYPE: NULL
...
```

Performance Schema: Status Variables

- Getting current values of status variables in some thread with **gdb** is possible, but not trivial
- Status variables are instrumented more ar less like in MySQL 5.7.
- But there are 3 more summary tables it seems:

```
MariaDB [performance_schema]> show tables like '%status%';
+----------------------------------------+
| Tables_in_performance_schema (%status%)   |
+----------------------------------------+
| global_status                          |
...
| session_status                         |
| status_by_account                      | -- user, host
| status_by_host                         | -- host
| status_by_thread                       | -- threads.thread_id
| status_by_user                         | -- user
+----------------------------------------+
8 rows in set (0,001 sec)
```

Status Variables Instrumentation: Example

```
MariaDB [performance_schema]> select s.*, t.processlist_user,
t.processlist_info from status_by_thread s, threads t where
t.thread_id = s.thread_id and variable_name like 'Rows%' and
t.processlist_id != connection_id() limit 2\G
*************************** 1. row ***************************
    THREAD_ID: 45
   VARIABLE_NAME: Rows_read
  VARIABLE_VALUE: 411
processlist_user: openxs
processlist_info: UPDATE sbtest1 SET k=k+1 WHERE id=?
*************************** 2. row ***************************
    THREAD_ID: 45
   VARIABLE_NAME: Rows_sent
  VARIABLE_VALUE: 0
processlist_user: openxs
processlist_info: UPDATE sbtest1 SET k=k+1 WHERE id=?
```

Performance Schema: Stored Procedures Instrumentation

- Along the lines of MySQL WL#5766
- New instrumentable object types added:

```
MariaDB [performance_schema]> select distinct object_type from
setup_objects;
+-------------+
| object_type |
+-------------+
| EVENT       |
| FUNCTION    |
| PROCEDURE   |
| TABLE       |
| TRIGGER     |
+-------------+
5 rows in set (0,023 sec)
```

- Enabled/timed by default in non-system databases

P_S Stored Procedures Instrumentation: Details

- 20 related instruments added:

```
MariaDB [performance_schema]> select * from setup_instruments where
name like 'statement/sp/%' or name like 'statement/scheduler%';
+--------------------------------+---------+-------+
| NAME                           | ENABLED | TIMED |
+--------------------------------+---------+-------+
| statement/sp/stmt              | YES     | YES   |
| statement/sp/set               | YES     | YES   |
| statement/sp/set_trigger_field | YES     | YES   |
| statement/sp/jump              | YES     | YES   |
| statement/sp/jump_if_not       | YES     | YES   |
| statement/sp/freturn           | YES     | YES   |
...
| statement/sp/set_case_expr     | YES     | YES   |
| statement/scheduler/event      | YES     | YES   |
+--------------------------------+---------+-------+
20 rows in set (0,002 sec)
```

P_S Stored Procedures Instrumentation: Details

- New **events_statements_summary_by_program** table added
- KB just lists columns without much details
- Some additional columns with statistics about nested statements invoked during stored program execution:

```
...
| COUNT_STATEMENTS          | bigint(20) unsigned ...
| SUM_STATEMENTS_WAIT       | bigint(20) unsigned ...
| MIN_STATEMENTS_WAIT       | bigint(20) unsigned ...
| AVG_STATEMENTS_WAIT       | bigint(20) unsigned ...
| MAX_STATEMENTS_WAIT       | bigint(20) unsigned …
...
```

- Let’s run a quick demo along the lines of this blog post...

Performance Schema: SX-locks Instrumentation

- See MySQL WL#7445 - "**PERFORMANCE SCHEMA: instrument SX-lock for rw_lock**"
- Instrumentation for read/write locks is enhanced to support SX-lock operations
- 10 new **wait/synch/sxlock/%** instruments, disabled by default:

```
MariaDB [performance_schema]> select * from setup_instruments where name
like '%sxlock%';
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
+-----------------------------------------------------+---------+-------+
| wait/synch/sxlock/innodb/btr_search_latch           | NO      | NO    |
| wait/synch/sxlock/innodb/dict_operation_lock        | NO      | NO    |
| wait/synch/sxlock/innodb/fil_space_latch            | NO      | NO    |
| wait/synch/sxlock/innodb/fts_cache_rw_lock          | NO      | NO    |
| wait/synch/sxlock/innodb/fts_cache_init_rw_lock     | NO      | NO    |
| wait/synch/sxlock/innodb/trx_i_s_cache_lock         | NO      | NO    |
| wait/synch/sxlock/innodb/trx_purge_latch            | NO      | NO    |
| wait/synch/sxlock/innodb/index_tree_rw_lock         | NO      | NO    |
| wait/synch/sxlock/innodb/index_online_log           | NO      | NO    |
| wait/synch/sxlock/innodb/dict_table_stats           | NO      | NO    |
+-----------------------------------------------------+---------+-------+
10 rows in set (0,003 sec)
```

SX-locks Instrumentation: Example

- Let's consider an example (demo) and check the SX-locks set during some **sysbench** tests
- **--threads=4 /usr/share/sysbench/oltp_update_index.lua, lat. 155.80**:

```
MariaDB [performance_schema]> select event_name, count_star,
sum_timer_wait/1000000000 time from
events_waits_summary_global_by_event_name where event_name like
'wait/synch/sxlock%' and count_star > 0 order by sum_timer_wait desc;
+---------------------------------------------+------------+----------+
| event_name                                  | count_star | time     |
+---------------------------------------------+------------+----------+
| wait/synch/sxlock/innodb/index_tree_rw_lock |      32500 | 780.0318 |
| wait/synch/sxlock/innodb/fil_space_latch    |       6791 | 105.7227 |
| wait/synch/sxlock/innodb/trx_purge_latch    |      16441 |   6.2344 |
+---------------------------------------------+------------+----------+
3 rows in set (0,005 sec)
```

- **--threads=128 /usr/share/sysbench/oltp_update_index.lua, lat. 1109.09**:

```
| wait/synch/sxlock/innodb/index_tree_rw_lock |      65186 | 165472.7347 |
```

Performance Schema: Transactions Instrumentation

- Transactions are now instrumented similarly to MySQL 5.7 (see MySQL manual and compare to MariaDB KB). Some minor problems exist (see MDEV-23944)
- **events_transactions_%** tables and consumers. disabled by default:

```
MariaDB [performance_schema]> select * from setup_consumers where name
like '%transaction%';
+---------------------------------+---------+
| NAME                            | ENABLED |
+---------------------------------+---------+
| events_transactions_current     | NO      |
| events_transactions_history     | NO      |
| events_transactions_history_long | NO      |
+---------------------------------+---------+
3 rows in set (0,001 sec)
```

- "**transaction**" instrument, disabled by default
- P_S events hierarchy is extended:

  transactions → statements → stages → waits

Transactions Instrumentation: Example

- Let’s consider an example (demo) of getting the sequence of statements executed in frames of current active transactions.
- With proper instrumentation and consumers enabled:

```
MariaDB [performance_schema]> select t.thread_id, h.event_id,
h.sql_text from events_statements_history_long h,
events_transactions_current t where h.nesting_event_id =
t.event_id and t.state = 'ACTIVE';
+-----------+----------+-----------------------------------+
| thread_id | event_id | sql_text                          |
+-----------+----------+-----------------------------------+
|        15 |      565 | show variables like 'gtid%'       |
|        15 |      612 | select * from t limit 1           |
|        15 |      731 | update t set c1 = 2 where id = 1  |
|        15 |      782 | update t set c1 = 3 where id = 2  |
+-----------+----------+-----------------------------------+
4 rows in set (0,001 sec)
```

Performance Schema: User Variables Instrumentation

- It was really hard (but possible with **gdb**) to find the value of user variables in specific thread in the past...
- User variables instrumentation in P_S helps to make it trivial:

```
MariaDB [performance_schema]> desc user_variables_by_thread;
+----------------+---------------------+------+-----+---------+-------+
| Field          | Type                | Null | Key | Default | Extra |
+----------------+---------------------+------+-----+---------+-------+
| THREAD_ID      | bigint(20) unsigned | NO   |     | NULL    |       |
| VARIABLE_NAME  | varchar(64)         | NO   |     | NULL    |       |
| VARIABLE_VALUE | longblob            | YES  |     | NULL    |       |
+----------------+---------------------+------+-----+---------+-------+
3 rows in set (0,002 sec)
```

P_S User Variables Instrumentation: Example

- Let's check how to find user variables in the current thread:

```
MariaDB [performance_schema]> set @a := 10;
Query OK, 0 rows affected (0,000 sec)

MariaDB [performance_schema]> select uv.* from
user_variables_by_thread uv join threads t where t.thread_id =
uv.thread_id and t.processlist_id=connection_id()\G
*************************** 1. row ***************************
    THREAD_ID: 239
 VARIABLE_NAME: a
VARIABLE_VALUE: 10
1 row in set (0,001 sec
```

Performance Schema: Replication Instrumentation

- Related tables, subset of those in MySQL 5.7:

```
MariaDB [performance_schema]> show tables like 'replication%';
+-----------------------------------------+
| Tables_in_performance_schema (replication%) |
+-----------------------------------------+
| replication_applier_configuration       |
| replication_applier_status              |
| replication_applier_status_by_coordinator |
| replication_connection_configuration    |
+-----------------------------------------+
4 rows in set (0,001 sec)
```

- Still work in progress, too early to use for many cases as **%_status_by_worker** is missing. See my MDEV-23590
- Only partially documented in the KB...

Replication Instrumentation: Examples

- Let's try to do a quick test (demo)... See also this and that
- Then let's check how this is supposed to work:

```
openxs@ao756:~/dbs/maria10.5/mysql-test/suite/perfschema/t$grep
-li replication *
ddl_replication_applier_configuration.test
ddl_replication_applier_status_by_coordinator.test
ddl_replication_applier_status_by_worker.test
...
dml_replication_connection_status.test
dml_replication_group_member_stats.test
dml_replication_group_members.test
rpl_group_member_stats.test
rpl_group_members.test
rpl_statements.test
rpl_threads.test
show_sanity.test
```

What's new in Thread Pool?

- MariaDB Thread Pool (since 5.5!) is cool!
- Information Schema tables (4) were added in 10.5 for internals of generic thread pool (MDEV-19313)
- **thread_pool_dedicated_listener** - the queueing time in the THREAD_POOL_QUEUES and the actual queue size in the THREAD_POOL_GROUPS table will be more exact, since IO requests are immediately dequeued from pool, without delay
- **thread_pool_exact_stats** - better queueing time statistics by using a high precision timestamp, at a small performance cost, for the time when the connection was added to the queue. This timestamp helps calculate the queuing time shown in the THREAD_POOL_QUEUES table.
- KB still misses details about the tables, columns, output examples...
- This commit is a useful reading
- Let's just check what we can see in these tables (demo)

Thread Pool Instrumentation: Groups

```
MariaDB [information_schema]> select * from THREAD_POOL_GROUPS\G
*************************** 1. row ***************************
       GROUP_ID: 0
    CONNECTIONS: 8
        THREADS: 9
 ACTIVE_THREADS: 1
STANDBY_THREADS: 3
   QUEUE_LENGTH: 3
   HAS_LISTENER: 1
     IS_STALLED: 0
*************************** 2. row ***************************
       GROUP_ID: 1
    CONNECTIONS: 9
        THREADS: 9
 ACTIVE_THREADS: 1
STANDBY_THREADS: 0
   QUEUE_LENGTH: 0
   HAS_LISTENER: 0
     IS_STALLED: 0
2 rows in set (0,015 sec)
```

Thread Pool Instrumentation: Queues

```
MariaDB [information_schema]> select * from THREAD_POOL_QUEUES;
+----------+----------+----------+---------------+----------------------------+
| GROUP_ID | POSITION | PRIORITY | CONNECTION_ID | QUEUEING_TIME_MICROSECONDS |
+----------+----------+----------+---------------+----------------------------+
|        0 |        0 |        1 |            34 |                     401133 |
|        0 |        1 |        1 |            20 |                     401133 |
|        0 |        2 |        1 |            26 |                     401133 |
|        0 |        3 |        1 |            22 |                     401133 |
|        1 |        0 |        1 |            31 |                     401139 |
|        1 |        1 |        1 |            33 |                     401139 |
|        1 |        2 |        1 |            21 |                     401139 |
|        1 |        3 |        1 |            27 |                     401139 |
+----------+----------+----------+---------------+----------------------------+
8 rows in set (0,200 sec)
```

Thread Pool Instrumentation: Statistics

```
MariaDB [information_schema]> select * from  THREAD_POOL_STATS\G
*************************** 1. row ***************************
                  GROUP_ID: 0
         THREAD_CREATIONS: 9
THREAD_CREATIONS_DUE_TO_STALL: 0
                     WAKES: 2170
       WAKES_DUE_TO_STALL: 0
                 THROTTLES: 0
                    STALLS: 0
          POLLS_BY_LISTENER: 8757
            POLLS_BY_WORKER: 944
       DEQUEUES_BY_LISTENER: 2691
         DEQUEUES_BY_WORKER: 7985
*************************** 2. row ***************************
                  GROUP_ID: 1
         THREAD_CREATIONS: 10
                     WAKES: 2192
...
2 rows in set (0,001 sec)
```

Thread Pool Instrumentation: Waits

```
MariaDB [information_schema]> select * from THREAD_POOL_WAITS;
+----------------+-------+
| REASON         | COUNT |
+----------------+-------+
| UNKNOWN        |     0 |
| SLEEP          |     0 |
| DISKIO         |  3419 |
| ROW_LOCK       |     7 |
| GLOBAL_LOCK    |     0 |
| META_DATA_LOCK |     0 |
| TABLE_LOCK     |     0 |
| USER_LOCK      |     0 |
| BINLOG         |     0 |
| GROUP_COMMIT   | 40097 |
| SYNC           |     0 |
| NET            |    25 |
+----------------+-------+
12 rows in set (0,001 sec)
```

What's new in ANALYZE?

- Execute the statement, and then produce EXPLAIN output instead of the result set, annotated with execution stats
- ANALYZE FORMAT=JSON for statements is improved, now it also shows the time spent checking the WHERE clause and doing other auxiliary operations (MDEV-20854)
- We now count the "gap" time between table accesses and display it as **r_other_time_ms** in the "table" element
- Table access time is reported as **r_table_time_ms** (former **r_total_time_ms**)
- Let's consider the example (demo)
- Compare to MySQL 8.0.18+ EXPLAIN ANALYZE

ANALYZE FORMAT=JSON: Example

- See **analyze_stmt_orderby**.MTR test

```
MariaDB [test]> analyze format=json
    -> select col1 f1, col2 f2, col1 f3 from t2 group by f1\G
*************************** 1. row ***************************
ANALYZE: {
  "query_block": {
    "select_id": 1,
    "r_loops": 1,
    "r_total_time_ms": 0.17553371,
    "table": {
    "table_name": "t2",
    "access_type": "range",
...
    "r_rows": 20,
    "r_table_time_ms": 0.124037665,
    "r_other_time_ms": 0.044389173,
    "filtered": 100,
    "r_filtered": 100,
    "using_index_for_group_by": true
...
```

Summary

- **MariaDB 10.5** added a lot of useful and interesting features and improvements that may help for troubleshooting performance issues
- Documentation for many of them (P_S improvements specifically) is not yet completed. We have to rely on MySQL manual, tests and source code review when in doubts
- There are some bugs in new P_S instrumentation
- So, there is still a lot of work to do for Engineering, Documentation team, users and bloggers (like me)

Thank you!

Questions and Answers?

Please, search and report bugs at:

https://jira.mariadb.org