MySQL / MariaDB SQL Queries

This document belongs to complete DB SQL Queries .

Database Size

Description	SQL Statement - Queries
Database Size by DB wise	SELECT table_schema AS "Database", ROUND(SUM(data_length + index_length) / 1024 / 1024 / 1024, 2) AS "Size (GB)" FROM information_schema.TABLES GROUP BY table_schema;
All Databases together in Single Column	SELECT table_schema "DB Name",ROUND(SUM(data_length + index_length) / 1024 / 1024 / 1024 , 1) "DB Size in GB" FROM information_schema.tables;

Table Size

Description	SQL Statement - Queries

Table Size in given Daatabase Name	SELECT table_name AS `Table Name`, ROUND(((data_length + index_length) / 1024 / 1024 / 1024), 2) AS `Size in GB` FROM information_schema.tables WHERE table_schema = 'your_database_name' ORDER BY (data_length + index_length) DESC;

All Objectes Rows count

Description	SQL Statement - Queries
Views Count	SELECT count(*) from information_schema.views;
Stored Procedure Count	SELECT count(*) from information_schema.routines where routine_type='procedure'; SELECT COUNT(*), ROUTINE_NAME FROM information_schema.ROUTINES WHERE ROUTINE_TYPE='PROCEDURE' GROUP BY ROUTINE_NAME; SELECT ROUTINE_SCHEMA,ROUTINE_NAME,ROUTINE_TYPE from information_schema.ROUTINES where ROUTINE_TYPE='PROCEDURE';
Events Count	SELECT count(*) from information_schema.events;
Total Number of Tables	SELECT count(*) AS TOTAL_NUMBER_OF_TABLES FROM INFORMATION_SCHEMA.TABLES;
Triggers Count	SELECT count(*) from information_schema.triggers;
Functions Count	SELECT COUNT(*), ROUTINE_NAME FROM information_schema.ROUTINES WHERE ROUTINE_TYPE = 'FUNCTION' and ROUTINE_SCHEMA = '<DBNAME>' GROUP BY ROUTINE_NAME;
Table Count	SELECT IFNULL(table_schema,'Total') "Database",Table_Count FROM (SELECT COUNT(1) Table_Count,table_schema FROM information_schema.tables WHERE table_schema NOT IN ('information_schema','mysql') GROUP BY table_schema WITH ROLLUP) A ORDER BY Table_Count; SELECT TABLE_NAME, SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = '<DBNAME>' GROUP BY TABLE_NAME;
Mysql Show All Schema Objects	SELECT OBJECT_TYPE,OBJECT_SCHEMA,OBJECT_NAME FROM (SELECT 'TABLE' AS OBJECT_TYPE,TABLE_NAME AS OBJECT_NAME,TABLE_SCHEMA AS OBJECT_SCHEMA FROM information_schema.TABLES UNION SELECT 'VIEW' AS OBJECT_TYPE,TABLE_NAME AS OBJECT_NAME,TABLE_SCHEMA AS OBJECT_SCHEMA FROM information_schema.VIEWS UNION SELECT 'INDEX[Type:Name:Table]' AS OBJECT_TYPE,CONCAT (CONSTRAINT_TYPE,' : ',CONSTRAINT_NAME,' : ',TABLE_NAME) AS OBJECT_NAME,TABLE_SCHEMA AS OBJECT_SCHEMA FROM information_schema.TABLE_CONSTRAINTS UNION SELECT ROUTINE_TYPE AS OBJECT_TYPE,ROUTINE_NAME AS OBJECT_NAME,ROUTINE_SCHEMA AS OBJECT_SCHEMA FROM information_schema.ROUTINES UNION SELECT 'TRIGGER[Schema:Object]' AS OBJECT_TYPE,CONCAT (TRIGGER_NAME,' : ',EVENT_OBJECT_SCHEMA,' : ',EVENT_OBJECT_TABLE) AS OBJECT_NAME,TRIGGER_SCHEMA AS OBJECT_SCHEMA FROM information_schema.triggers) R WHERE R.OBJECT_SCHEMA = '**<DBNAME>**'; Or SELECT DB,OBJECT,COUNT(OBJECT_NAME) as cnt from (

	SELECT COUNT*(),TABLE_TYPE FROM information_schema.TABLES WHERE TABLE_SCHEMA='<dbname>' GROUP BY TABLE_TYPE union all* *SELECT COUNT()*,TABLE_NAME FROM information_schema.VIEWS WHERE TABLE_SCHEMA = '*<dbname>*' GROUP BY TABLE_NAME union all SELECT COUNT*() FROM information_schema.ROUTINES WHERE ROUTINE_TYPE='PROCEDURE' and ROUTINE_SCHEMA = '<dbname>' union all* *SELECT COUNT()* FROM information_schema.ROUTINES WHERE ROUTINE_TYPE='FUNCTION' and ROUTINE_SCHEMA = '*<dbname>*' union all SELECT COUNT*() FROM information_schema.EVENTS WHERE EVENT_SCHEMA= '<dbname>' union all* *SELECT COUNT()* FROM information_schema.TRIGGERS WHERE TRIGGER_SCHEMA='*<dbname>*') FINAL GROUP BY DB,OBJECT;
Get the report - List of Objectes along with count of each object [Table . View . Stored Procedure , Trgger , Functions]	(1) set @table_Schema='<dbname>'; select db, object, sum(cnt) as cnt from (select @table_schema as db, 'tables' as object, count(table_name) as cnt from information_schema.tables where table_schema = @table_schema and table_type = 'BASE TABLE' union all select @table_schema as db, 'views' as object, count(table_name) as cnt from information_schema.tables where table_schema = @table_schema and table_type != 'BASE TABLE' union all select @table_schema as db, 'routine' as object, count(routine_name) as cnt from information_schema.routines where routine_schema = @table_schema union all select @table_schema as db, 'trigger' as object, count(trigger_name) as cnt from information_schema.triggers where TRIGGER_SCHEMA = @table_schema) final group by db, object; **Or** SELECT 'TABLE' AS DB, TABLE_TYPE AS OBJECT, COUNT*() AS cnt, '<dbname' AS DB_name* *FROM information_schema.TABLES* *WHERE TABLE_SCHEMA='<dbname'* *GROUP BY TABLE_TYPE* *UNION ALL* *SELECT 'VIEW' AS DB, TABLE_NAME AS OBJECT, COUNT()* AS cnt, '*<dbname*' AS DB_name FROM information_schema.VIEWS WHERE TABLE_SCHEMA = '*<dbname*' GROUP BY TABLE_NAME UNION ALL SELECT 'PROCEDURE' AS DB, 'PROCEDURE' AS OBJECT, COUNT*() AS cnt, '<dbname' AS DB_name* *FROM information_schema.ROUTINES* *WHERE ROUTINE_TYPE='PROCEDURE' AND ROUTINE_SCHEMA = '<dbname'* *UNION ALL* *SELECT 'FUNCTION' AS DB, 'FUNCTION' AS OBJECT, COUNT()* AS cnt, '*<dbname*' AS DB_name FROM information_schema.ROUTINES WHERE ROUTINE_TYPE='FUNCTION' AND ROUTINE_SCHEMA = '*<dbname*' UNION ALL SELECT 'EVENT' AS DB, 'EVENT' AS OBJECT, COUNT*() AS cnt, '<dbname' AS DB_name* *FROM information_schema.EVENTS* *WHERE EVENT_SCHEMA= '<dbname'* *UNION ALL*

	SELECT 'TRIGGER' AS DB, 'TRIGGER' AS OBJECT, COUNT() AS cnt, '<*dbname*' AS DB_name
	FROM information_schema.TRIGGERS
	WHERE TRIGGER_SCHEMA='<*dbname*';
	
	Or
	
	SELECT
	  CASE
	  WHEN table_type IS NOT NULL THEN 'TABLE'
	  WHEN routine_type = 'PROCEDURE' THEN 'PROCEDURE'
	  WHEN routine_type = 'FUNCTION' THEN 'FUNCTION'
	  WHEN event_type IS NOT NULL THEN 'EVENT'
	  WHEN trigger_type IS NOT NULL THEN 'TRIGGER'
	  END AS DB,
	  COALESCE(table_type, routine_type, event_type, trigger_type) AS OBJECT,
	  COUNT(*) AS cnt,
	  '<*dbname*>' AS DB_name
	FROM (
	  SELECT TABLE_TYPE AS table_type, NULL AS routine_type, NULL AS event_type, NULL AS trigger_type
	  FROM information_schema.TABLES
	  WHERE TABLE_SCHEMA='<*dbname*>'
	  UNION ALL
	  SELECT NULL, ROUTINE_TYPE, NULL, NULL
	  FROM information_schema.ROUTINES
	  WHERE ROUTINE_SCHEMA='<*dbname*>'
	  UNION ALL
	  SELECT NULL, NULL, 'EVENT', NULL
	  FROM information_schema.EVENTS
	  WHERE EVENT_SCHEMA='<*dbname*>'
	  UNION ALL
	  SELECT NULL, NULL, NULL, 'TRIGGER'
	  FROM information_schema.TRIGGERS
	  WHERE TRIGGER_SCHEMA='<*dbname*>'
	) AS combined_data
	GROUP BY COALESCE(table_type, routine_type, event_type, trigger_type);

```
Note : - MariaDB has some functions that are not present in MySQL, like JSON_DETAILED, which is called JSON_PRET
         in MySQL 8.0.

The list of those functions is present in MariaDB’s documentation, however check it twice as some information
regarding MySQL 8.0 is sometimes outdated, especially on this page, (like invisible columns, virtual columns,
wait, intersect, except, …).

This is not a blocking factor for migration, unless these functions are present in the default values of columns
But of course, if your application uses some of these functions, it may be necessary to modify it to use the
appropriate one in MySQL 8.0.

To illustrate this, let’s use the ADD_MONTHS function.

First let’s see if we have this function as default for some columns:

SELECT TABLE_NAME, COLUMN_NAME FROM information_schema.COLUMNS WHERE COLUMN_NAME LIKE 'ADD_MONTHS%';
+------------+-------------+
```

```
| TABLE_NAME | COLUMN_NAME |
+------------+-------------+
| date1      | ADD_MONTHS  |
+------------+-------------+
1 row in set (0.06 sec)

Great !….  But I’m sure I’ve created a table with that specific function as default. This is what I did:

ALTER TABLE t6 ADD COLUMN future DATETIME DEFAULT (ADD_MONTHS(NOW(), 2));

In fact, several functions are acting like aliases. If we check the output of SHOW CREATE TABLE statement,
we can see that the function is translated:

SHOW CREATE TABLE t6\G
*************************** 1. row ***************************
       Table: t6
Create Table: CREATE TABLE t6 (
  id int(11) NOT NULL DEFAULT nextval(mydatabase.s3),
  b int(11) DEFAULT NULL,
  future datetime DEFAULT (current_timestamp() + interval 2 month),
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
1 row in set (0.000 sec)
```

SHOW Commands with few Parameters

SQL Statement - Queries
SELECT VERSION(); SHOW VARIABLES LIKE 'lower_case_table_names'; SHOW VARIABLES LIKE '%char%'; SHOW VARIABLES LIKE '%collat%'; SHOW DATABASES;
show variables like '%slow_query%'; show variables like 'log_bin_trust_function_creators'; show variables like 'long_query_time';
SELECT SYSDATE() as Current_Date_And_Time ;

Index Related

Description	SQL Statement - Queries
Index Count	SELECT COUNT(*), INDEX_NAME FROM information_schema.STATISTICS WHERE TABLE_SCHEMA = '<DBNAME>' GROUP BY INDEX_NAME;
Duplicate index checking	SELECT DISTINCT TABLE_SCHEMA, TABLE_NAME, group_concat(INDEX_NAME) duplic8_UK, COLUMN_NAMES FROM (SELECT DISTINCT TABLE_SCHEMA, TABLE_NAME, INDEX_NAME, GROUP_CONCAT(COLUMN_NAME ORDER BY COLUMN_NAME SEPARATOR ',') AS COLUMN_NAMES FROM information_schema.STATISTICS WHERE NON_UNIQUE = 0 AND INDEX_NAME!='PRIMARY' AND INDEX_TYPE = 'BTREE' GROUP BY TABLE_SCHEMA, TABLE_NAME, INDEX_NAME) X group by TABLE_SCHEMA, TABLE_NAME, COLUMN_NAMES having count(*)> 1;
Find tables without PK	SELECT t.table_schema,t.table_name,t.engine FROM information_schema.tables t JOIN information_schema.columns c ON t.table_schema=c.table_schema AND t.table_name=c.table_name WHERE t.table_schema NOT IN ('mysql', ‘information_schema', ‘sys',

	'performance_schema') AND t.table_type = 'BASE TABLE' GROUP BY t.table_schema,t.table_name, t.engine HAVING SUM(IF(column_key IN ('PRI','UNI'), 1,0)) = 0;
Find tables with non-integer PK's	SELECT table_schema, table_name, column_name, data_type, character_maximum_length FROM information_schema.columns WHERE column_key IN ('PRI','UNI') AND ordinal_position=1 AND data_type NOT IN ('tinyint', 'smallint', 'mediumint', 'int', 'bigint', 'timestamp', 'datetime') AND table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema');
Find tables and indexes with the most latency	The overall performance improvement gained by optimizing a single part of a system, is limited by the fraction of time that the improved part is actually used - SELECT * FROM sys.schema_table_statistics WHERE table_schema='test' AND table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema'); SELECT * FROM sys.schema_index_statistics WHERE table_schema='test' AND table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema');
Find tables whose indexes > data by 50%	SELECT table_schema, table_name, index_length, data_length, index_length/data_length AS index_to_data_ratio FROM information_schema.tables WHERE table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema') AND INDEX_LENGTH > DATA_LENGTH*1.5;
Find tables with duplicate indexes	SELECT table_schema,table_name,redundant_index_name AS redundant_index, redundant_index_columns AS redundant_columns, dominant_index_name AS covered_by_index,sql_drop_index FROM sys.schema_redundant_indexes WHERE table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema');
Find unused indexes	SELECT * FROM sys.schema_unused_indexes WHERE object_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema');
Calculating the Index Size	→ Table size (data + index) SELECT table_schema "database", sum(data_length + index_length)/1024/1024 "size in MB" FROM information_schema.TABLES where table_schema='<dbname>' GROUP BY table_schema; → individual index size (before/after analyze table) **Before analyze table** SELECT database_name, index_name, sum(ROUND(stat_value * @@innodb_page_size / 1024 / 1024, 2)) size_in_mb from mysql.innodb_index_stats wHERE database_name ='<dbname>' AND stat_name = 'size' and table_name like 'node%' group by database_name, index_name order by size_in_mb desc; **After analyze table (analyze table command: " analyze table <tablename> for all; ")** SELECT database_name, index_name, sum(ROUND(stat_value * @@innodb_page_size / 1024 / 1024, 2)) size_in_mb from mysql.innodb_index_stats wHERE database_name ='dv_repo' AND stat_name = 'size' and table_name like 'node%' group by database_name, index_name order by size_in_mb desc; SELECT database_name, sum(ROUND(stat_value * @@innodb_page_size / 1024 / 1024, 2)) size_in_mb from mysql.innodb_index_stats WHERE database_name ='<dbname>' AND stat_name = 'size' and table_name like 'node%'; SELECT database_name, sum(ROUND(stat_value * @@innodb_page_size / 1024 / 1024, 2)) size_in_mb from mysql.innodb_index_stats WHERE database_name ='<dbname>' AND stat_name =

	'size' and table_name like 'node%'; **Table size from OS command** mysql@mariadb-test01 0 15:39:19 /mysql/data/dv_repo $ pwd /mysql/data/dv_repo mysql@mariadb-test01 0 15:44:11 /mysql/data/dv_repo $ du -ch node* |grep total 116G total
Getting the index related information by providing the given DB name	SELECT distinct table_name, index_name, column_name from information_schema.statistics where table_schema = '<dbname>' and index_name not like 'SYS%' and index_name != 'PRIMARY' order by 1, 2, seq_in_index;

Processlist and Queries

Description	SQL Statement - Queries
Count of Current Processlist	SELECT count(*) from information_schema.processlist;
List of Users , Host , DB count from current Processlist	SELECT USER,HOST, DB,COUNT(*) from information_schema.processlist group by user, db order by 4;
List of Users , Host , DB , Queries count from current Processlist	SELECT id,HOST,PROGRESS,user,time,substr(info, 1, 100) FROM information_schema.processlist where command != 'Sleep' order by time;
Display FULL Current processlist	SHOW FULL PROCESSLIST; SHOW PROCESSLIST;
Display all list of Current Process	SELECT * FROM information_schema.processlist;
Display number of connections for each user	SELECT USER, COUNT(*) FROM information_schema.processlist GROUP BY USER ;
Display number of connections for each host	SELECT HOST, COUNT(*) FROM information_schema.processlist GROUP BY HOST;
Display root user activity	SELECT * FROM information_schema.processlist WHERE USER = 'root';
Display processes associated with SELECT queries	SELECT * FROM information_schema.processlist WHERE INFO LIKE 'SELECT %';
Display average query time for each database	SELECT DB, AVG(TIME) FROM information_schema.processlist GROUP BY DB;
Stop/Kill Queries using the Transaction IDs taken from the PROCESS LIST commands	CALL mysql.rds_kill(processID);

Killing the process / Sessions by using concatination Script

```
You have 2 option , either you can restart the MariaDB service which in turn will remove these thread or
alternate option is to kill then either one by one or you can use below query to do the same in bulk.

# mysql -u<user> -p<password>  -e "select concat('KILL ',id,';')  into outfile '/tmp/sleep_processes.txt'
from information_schema.processlist where Command = 'Sleep' and USER not in ('system user', monitor_user)"

```

```
Note: List out all the USER  in above query which you don't want to kill , make sure none of the system_user or
replication user defined should get killed , better you can add them in above query in section "....USER not in
('system user', 'monitor_user', '......','...........')""

Run the generated sleep_processes.txt output with in MariaDB server , it will kill all of the sleeping threads.

# mysql -u<user> -p<password>  -e "source /tmp/sleep_processes.txt;"

For example :-

# mysql -u root  -pxxxx  -e "select concat('KILL ',id,';')  into outfile '/tmp/sleep_processes.txt' from
information_schema.processlist where Command = 'Sleep' and USER not in ('system user', 'monitor_user')"

ls -ltrh
-rw-rw-rw- 1 mysql mysql 19 Apr 27 05:03 sleep_processes.txt

# cat sleep_processes.txt
KILL 104;
KILL 93;

You can randomly verify the above thread id with in MariaDB server to cross check none of the valid user get
killed .

[root@gcn_01 tmp]# mysql -u root -pxxxxxx
MariaDB [(none)]> show processlist;
+-----+-------------+----------------------+------+---------+---------+--------------------+------------------+
| Id  | User        | Host                 | db   | Command | Time    | State              | Info             |
+-----+-------------+----------------------+------+---------+---------+--------------------+------------------+
|   1 | system user |                      | NULL | Sleep   | 2075783 | wsrep aborter idle | NULL             |
|   2 | system user |                      | NULL | Sleep   | 2075783 | closing tables     | NULL             |
|  93 | maxmon_gc   | 192.168.140.100:37858 | NULL | Sleep   |       0 |                    | NULL             |
| 104 | app_user    | 192.168.140.156:37156 | NULL | Sleep   |      47 |                    | NULL             |
| 108 | root        | localhost            | NULL | Query   |       0 | starting           | show processlist |
+-----+-------------+----------------------+------+---------+---------+--------------------+------------------+
5 rows in set (0.000 sec)
```

MySQL Users information SQL Queries

Description	SQL Statement - Queries
Display the MySQL Database users	SELECT USER,HOST FROM mysql.user;

CHARSET and COLLATIONS

Description	SQL Statement - Queries
To display each database CHARSET and COLLATION details	SELECT SCHEMA_NAME 'database', default_character_set_name 'charset', DEFAULT_COLLATION_NAME 'collation' FROM information_schema.SCHEMATA;

Troubleshooting Locks and Waits

Description	SQL Statement - Queries

Troubleshooting Locks	SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX; SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCKS; → MySQL 5.7 DB Version SELECT * FROM performance_schema.data_locks; → MySQL 8 DB Version
Troubleshooting Lock Waits	SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS; → MySQL 5.7 DB Version SELECT * FROM performance_schema.data_lock_waits; → MySQL 8 DB Version
Find Waiting Transactions and the Locks blocking the resource required [MySQL 5.7 DB Version]	SELECT r.trx_id waiting_trx_id, r.trx_mysql_thread_id waiting_thread, r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_mysql_thread_id blocking_thread, b.trx_query blocking_query FROM information_schema.innodb_lock_waits w INNER JOIN information_schema.innodb_trx b ON b.trx_id = w.blocking_trx_id INNER JOIN information_schema.innodb_trx r ON r.trx_id = w.requesting_trx_id;
Find Waiting Transactions and the Locks blocking the resource required [MySQL 8 DB Version]	SELECT r.trx_id waiting_trx_id,r.trx_mysql_thread_id waiting_thread,r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_mysql_thread_id blocking_thread, b.trx_query blocking_query FROM performance_schema.data_lock_waits w INNER JOIN information_schema.innodb_trx b ON b.trx_id = w.blocking_engine_transaction_id INNER JOIN information_schema.innodb_trx r ON r.trx_id = w.requesting_engine_transaction_id;
Mariadb lock scripts	SELECT r.trx_id waiting_trx_id, r.trx_mysql_thread_id waiting_thread, r.trx_query waiting_query, b.trx_id blocking_trx_id, b.trx_mysql_thread_id blocking_thread, b.trx_query blocking_query FROM information_schema.innodb_lock_waits w INNER JOIN information_schema.innodb_trx b ON b.trx_id = w.blocking_trx_id INNER JOIN information_schema.innodb_trx r ON r.trx_id = w.requesting_trx_id\G

InnoDB related DB Queries

Description	SQL Statement - Queries
Find tables not using InnoDB	SELECT t.table_schema,t.table_name,t.engine FROM information_schema.tables t WHERE t.table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema') AND t.engine <> 'InnoDB' AND t.table_type = 'BASE TABLE'; Or based on the amount of data in tables retrieved from Information Schema: SELECT 'Sum_index_length' as '', IFNULL(SUM(INDEX_LENGTH),0) from information_schema.TABLES where ENGINE='InnoDB'; SELECT 'Sum_data_length' as '', IFNULL(SUM(DATA_LENGTH),0) from information_schema.TABLES where ENGINE='InnoDB'; Or based also on the physical tablespace file: SELECT FILE_SIZE FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES WHERE ..
Query that you can use to verify the Storage Engines actually used on your database	SELECT COUNT(*) as '# TABLES', CONCAT(ROUND(sum(data_length) / (1024 * 1024 * 1024), 2), 'G') DATA,CONCAT(ROUND(sum(index_length) / (1024 * 1024 * 1024), 2), 'G') INDEXES,CONCAT(sum(ROUND((data_length + index_length) / (1024 * 1024 * 1024), 2)), 'G') 'TOTAL SIZE', ENGINE FROM information_schema.TABLES WHERE TABLE_SCHEMA NOT IN

	('mysql', 'information_schema', 'performance_schema', 'sys') GROUP BY engine;
Display the Storage Engines other than MyISAM and InnoDB	SELECT TABLE_SCHEMA, TABLE_NAME, ENGINE FROM information_schema.TABLES WHERE TABLE_SCHEMA NOT IN ('mysql', 'information_schema', 'performance_schema', 'sys') AND engine NOT IN ('MyISAM','InnoDB'); Note :- If we see any Engine = aria or any other Engines , from above command , There are two ways to fix this, either we modify the engine directly on the MariaDB server (recommended), or we modify the engine when loading the data on MySQL 8.0. Recommended way: ALTER TABLE mydatabase.t4 ENGINE=InnoDB; I will illustrate the dump in the next part without first changing the storage engine to InnoDB, but be aware that sometimes there may be limitations related to the size of the rows, it also depends on the character set used.

Performance_schema Related Database Queries

Description	SQL Statement - Queries
The MySQL DBA has also access to the disk space usage via the SQL interface using Performance_Schema. In MySQL Database Service, Performance_Schema provides some extra tables that are part of the Health Monitor:	select * from health_block_device order by timestamp desc limit 10;
Using performance_schema you can also find the size of your dataset and the space used on disk:	SELECT format_bytes(sum(data_length)) DATA_SIZE, format_bytes(sum(index_length)) INDEX_SIZE, format_bytes(sum(data_length+index_length)) TOTAL_SIZE, format_bytes(sum(data_free)) DATA_FREE, format_bytes(sum(FILE_SIZE)) FILE_SIZE, format_bytes((sum(FILE_SIZE)/10 - (sum(data_length)/10 + sum(index_length)/10))*10) WASTED_SIZE FROM information_schema.TABLES as t JOIN information_schema.INNODB_TABLESPACES as it ON it.name = concat(table_schema,"/",table_name) ORDER BY (data_length + index_length);
In the result of the previous SQL statement, we can see in the last column (WASTED_SIZE) that there are almost 650MB of wasted disk space. This column represents gaps in tablespaces. Let's find out for which tables and how to recover it: Disk Space Utilization	SELECT NAME, TABLE_ROWS, format_bytes(data_length) DATA_SIZE, format_bytes(index_length) INDEX_SIZE, format_bytes(data_length+index_length) TOTAL_SIZE, format_bytes(data_free) DATA_FREE, format_bytes(FILE_SIZE) FILE_SIZE, format_bytes((FILE_SIZE/10 - (data_length/10 + index_length/10))*10) WASTED_SIZE FROM information_schema.TABLES as t JOIN information_schema.INNODB_TABLESPACES as it ON it.name = concat(table_schema,"/",table_name) ORDER BY (data_length + index_length) desc LIMIT 10; set information_schema_stats_expiry=0;

	SELECT NAME, TABLE_ROWS, format_bytes(data_length) DATA_SIZE, format_bytes(index_length) INDEX_SIZE, format_bytes(data_length+index_length) TOTAL_SIZE, format_bytes(data_free) DATA_FREE, format_bytes(FILE_SIZE) FILE_SIZE, format_bytes((FILE_SIZE/10 - (data_length/10 + index_length/10))*10) WASTED_SIZE FROM information_schema.TABLES as t JOIN information_schema.INNODB_TABLESPACES as it ON it.name = concat(table_schema,"/",table_name) ORDER BY (data_length + index_length) desc LIMIT 10;
Performance Schema feature of MySQL to fetch the most I/O utilized queries. Here's an example query that will show the top 10 queries by I/O utilization:	SELECT QUERY_ID, SUM(IO_READ_REQUESTS) AS TOTAL_READ_REQUESTS, SUM(IO_READ_BYTES) AS TOTAL_READ_BYTES, SUM(IO_WRITE_REQUESTS) AS TOTAL_WRITE_REQUESTS, SUM(IO_WRITE_BYTES) AS TOTAL_WRITE_BYTES, SUM(IO_MISC_REQUESTS) AS TOTAL_MISC_REQUESTS, SUM(IO_PENDING) AS TOTAL_PENDING FROM performance_schema.events_statements_summary_by_digest WHERE DIGEST_TEXT LIKE '%YOUR_SEARCH_STRING%' GROUP BY QUERY_ID ORDER BY TOTAL_READ_REQUESTS + TOTAL_WRITE_REQUESTS DESC LIMIT 10; In this query, you can replace YOUR_SEARCH_STRING with a keyword or part of a query that you want to search for. This will return the top 10 queries that match the search string based on their I/O utilization. Note that in order for this query to work, you need to have the Performance Schema enabled in your MySQL installation, and you need to have the appropriate privileges to access the performance_schema database.

WATCH Commands

```
watch -n 1 'mysql --defaults-file=/etc/my.cnf -h 127.0.0.1 -P 6032 -t -e "select b.weight, c.* from
stats_mysql_connection_pool c left JOIN runtime_mysql_servers b ON  c.hostgroup=b.hostgroup_id and
c.srv_host=b.hostname and c.srv_port = b.port where hostgroup in( 50,52,70,71) order by hostgroup,
srv_host desc;" -e " select srv_host,command,avg(time_ms), count(ThreadID) from stats_mysql_processlist
group by srv_host,command;" -e "select * from stats_mysql_users;";
mysql  --defaults-file=~/.my.cnf -h 127.0.0.1 -P 6032  -t -e "select * from stats_mysql_global "|egrep -i
"(mirror|memory|stmt|processor)"'

+--------+-----------+-----------------------------------------------------------------------------+----------+----
| weight | hostgroup | srv_host                                                                    | srv_port | STA
+--------+-----------+-----------------------------------------------------------------------------+----------+----
| 1000   | 70        | proxysqltestdb.c7wzm8xxmrze.eu-central-1.rds.amazonaws.com                  | 3306     | ONL
| 1000   | 71        | proxysqltestdb2.c7wzm8xxmrze.eu-central-1.rds.amazonaws.com                 | 3306     | ONL
| 1000   | 71        | proxysqltestdb.c7wzm8xxmrze.eu-central-1.rds.amazonaws.com                  | 3306     | ONL
| 1     | 71        | proxysqltestdb-eu-central-1b.c7wzm8xxmrze.eu-central-1.rds.amazonaws.com | 3306     | ONLI
+--------+-----------+-----------------------------------------------------------------------------+----------+----
+----------+---------------------+--------------------------+
| username | frontend_connections | frontend_max_connections |
+----------+---------------------+--------------------------+
| m8_test  | 0                   | 10000                    |
+----------+---------------------+--------------------------+
| Query_Processor_time_nsec    | 0             |
| Com_backend_stmt_prepare     | 0             |
| Com_backend_stmt_execute     | 0             |
```

```
| Com_backend_stmt_close       | 0             |
| Com_frontend_stmt_prepare    | 0             |
| Com_frontend_stmt_execute    | 0             |
| Com_frontend_stmt_close      | 0             |
| Mirror_concurrency           | 0             |
| Mirror_queue_length          | 0             |
| SQLite3_memory_bytes         | 2652288       |
| ConnPool_memory_bytes        | 712720        |
| Stmt_Client_Active_Total     | 0             |
| Stmt_Client_Active_Unique    | 0             |
| Stmt_Server_Active_Total     | 0             |
| Stmt_Server_Active_Unique    | 0             |
| Stmt_Max_Stmt_id             | 1             |
| Stmt_Cached                  | 0             |
| Query_Cache_Memory_bytes     | 0             |

Reference Links :-

https://www.tusacentral.com/joomla/index.php/mysql-blogs/199-how-to-implement-proxysql-with-aws-aurora
https://www.tusacentral.com/joomla/index.php/mysql-blogs/168-how-to-mess-up-your-data
-using-one-command-in-mysqlgalera
```

Top Queries for Security Checking in MySQL

Description	SQL Statement - Queries
MySQL Uptime	SELECT VARIABLE_VALUE AS Uptime_seconds, NOW() AS "Now", NOW() - INTERVAL VARIABLE_VALUE SECOND AS "Up since", DATEDIFF(NOW(), NOW() - INTERVAL VARIABLE_VALUE SECOND) AS "Uptime_days" FROM performance_schema.session_status WHERE VARIABLE_NAME = 'Uptime'; SELECT TIME_FORMAT(SEC_TO_TIME(VARIABLE_VALUE),'%Hh %im') as Uptime from performance_schema.global_status where VARIABLE_NAME='Uptime'; SELECT TIME_FORMAT(SEC_TO_TIME(VARIABLE_VALUE),'%Hh %im') as Uptime from information_schema.GLOBAL_STATUS where VARIABLE_NAME='Uptime';
Find the Tables without Primary Key	SELECT table_name FROM information_schema.tables WHERE table_type = 'BASE TABLE' AND table_name NOT IN (SELECT DISTINCT table_name FROM information_schema.statistics WHERE index_name = 'PRIMARY'); SELECT t.table_schema,t.table_name,t.engine FROM information_schema.tables t JOIN information_schema.columns c ON t.table_schema=c.table_schema AND t.table_name=c.table_name WHERE t.table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema') AND t.table_type = 'BASE TABLE' GROUP BY t.table_schema,t.table_name, t.engine HAVING SUM(IF(column_key IN ('PRI','UNI'), 1,0)) = 0;
Top 10 Large Tables	SELECT table_name AS TABLE_NAME round(((data_length + index_length) / 1024 / 1024), 2) Table_Size_MB FROM information_schema.TABLES WHERE table_schema = "<your_database_name>" ORDER BY (data_length + index_length) DESC LIMIT 10 ;
Top 10 Fragmented Tables	SELECT table_name AS Table_Name, round((data_free / 1024 / 1024), 20 Free_Space_MB FROM information_schema.TABLES WHERE table_schema ="<your_database_name>" AND data_free > 0 ORDER BY data_free DESC LIMIT 10;

Duplicate Index	SELECT GROUP_CONCAT(DISTINCT(index_name) SEPARATOR ', ') AS Duplicate_Indexes FROM information_schema.statistics WHERE table_schema = '<Your_database_name>' AND non_unique = 0 GROUP BY table_name, index_name HAVING COUNT(*) > 1; Or SELECT DISTINCT TABLE_SCHEMA, TABLE_NAME, group_concat(INDEX_NAME) duplic8_UK, COLUMN_NAMES FROM (SELECT DISTINCT TABLE_SCHEMA, TABLE_NAME, INDEX_NAME, GROUP_CONCAT(COLUMN_NAME ORDER BY COLUMN_NAME SEPARATOR ',') AS COLUMN_NAMES FROM information_schema.STATISTICS WHERE NON_UNIQUE = 0 AND INDEX_NAME!='PRIMARY' AND INDEX_TYPE = 'BTREE' GROUP BY TABLE_SCHEMA, TABLE_NAME, INDEX_NAME) X group by TABLE_SCHEMA, TABLE_NAME, COLUMN_NAMES having count(*)> 1; Or SELECT table_schema,table_name,redundant_index_name AS redundant_index, redundant_index_columns AS redundant_columns, dominant_index_name AS covered_by_index,sql_drop_index FROM sys.schema_redundant_indexes WHERE table_schema NOT IN ('mysql', 'information_schema', 'sys', 'performance_schema'); Reference Link :- Take This Unique Quiz About Duplicate Indexes In MySQL | pt-duplicate-key-checker

Connections

Description	SQL Statement - Queries
Checking the Secure Connections	The first thing we can check is that all our clients encrypt their connection to the MySQL server. We use again Performance_Schema to retrieve the relevant information: select connection_type, substring_index(substring_index(name,"/",2),"/",-1) name,sbt.variable_value AS tls_version, t2.variable_value AS cipher,processlist_user AS user, processlist_host AS host from performance_schema.status_by_thread AS sbt join performance_schema.threads AS t on t.thread_id = sbt.thread_id join performance_schema.status_by_thread AS t2 on t2.thread_id = t.thread_id where sbt.variable_name = 'Ssl_version' and t2.variable_name = 'Ssl_cipher' order by connection_type, tls_version; Note :- If we want to force the user to use encrypted connections to our MySQL DB system, we modify the user like this: alter user fred require ssl;
Failed Connections	As a DBA, you also need to verify who is trying to connect unsuccessfully to your database server. It could be a user without the right credentials, using a non supported TLS or cipher version or eventually a malicious person/program. Plugin Installation :- [https://blogs.oracle.com/mysql/post/mysql-connection-control] Let's test the plugin to show how it works. The plugin library is connection_control.so that is installed in the plugin directory. On Oracle Linux/RedHat Enterprise/CentOS/Fedora the default path, if MySQL Server was installed via RPM, is: /usr/lib64/mysql/plugin/connection_control.so. It's possible to install the plugin in my.cnf, see the manual, but we will see how to use it interactively: INSTALL PLUGIN CONNECTION_CONTROL SONAME 'connection_control.so'; INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS SONAME 'connection_control.s

	Once installed, the plugin can be listed from Information_Schema: SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE 'connection%'; Now, let's configure the plugin. We will set the threshold of consecutive failed connection tentative to 4 and add a minimum of 1,5 seconds: SET GLOBAL connection_control_failed_connections_threshold = 4; SET GLOBAL connection_control_min_connection_delay = 1500 If there are failed login attempts, we will be able to list them in Information_Schema using the table connection_control_failed_login_attempts, for the moment it's still empty: select * from information_schema.connection_control_failed_login_attempts; Empty set (0.0003 sec)
Verify the failed attempts from the information_schema.connection_control_failed_loin_attempts table:	select * from information_schema.connection_control_failed_login_attempts; select logged, data from performance_schema.error_log where error_code in ('MY-010926', 'MY-010914') order by logged desc limit 10; select * from performance_schema.global_variables where variable_name like 'connection_control%'; In Performance_Schema, there is a table providing us some useful summary: host_cache: select ip, count_ssl_errors, count_handshake_errors, count_authentication_errors from host_cache;
memory tracking	The current global memory consumption (including background threads and admin) can be returned by the following query: SELECT format_bytes(variable_value) global_connection_memory FROM performance_schema.global_status WHERE variable_name='Global_connection_memory';
Error Logging	select * from (select * from performance_schema.error_log order by logged desc limit 10) a order by logged\G select subsystem, count(*) from performance_schema.error_log group by subsystem order by subsystem; select prio, count(*) from performance_schema.error_log group by prio order by prio; The error log provides a lot of information about how healthy is your system, about health monitor, InnoDB, replication, authentication failures, etc… For example, we can see the disk usage (see the previous post) in the error_log table too: select * from error_log where subsystem="Health" and data like 'DISK:%' order by logged desc limit 4\G select * from performance_schema.global_variables where variable_name like 'log_error%'; select * from performance_schema.global_status where variable_name in ('rapid_resize_status','rapid_service_status','rapid_cluster_ready_number'); select * from (select * from performance_schema.error_log order by logged desc limit 10) a order by logged\G

```
Testing for Failed Connections :-
---------------------------------

Let’s test our settings by trying to connect to our MySQL Server using MySQL Shell and an invalid password.
We will try 10 times in a row and only display the real time spent by the command (the connection attempt):

$ TIMEFORMAT=%R
$ for i in `seq 1 10`
  do
   time mysql mysql://fred:ffr@127.0.0.2 2>&1 >/dev/null | grep meh
  done
```

```
0.747
0.749
0.731
0.735
2.265
2.753
3.744
4.757
5.750
6.741

It’s obvious that after the 4th attempt the response time kept increasing.
We can verify the status variable Connection_control_delay_generated, that contains the number of times the
server added a delay to its response to a failed connection attempt:

show global status like 'connection_control_%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| Connection_control_delay_generated | 6     |
+-------------------------------------+-------+

And this time the connection_control_failed_login_attempts table is not empty:

select * from information_schema.connection_control_failed_login_attempts;
+------------+-----------------+
| USERHOST   | FAILED_ATTEMPTS |
+------------+-----------------+
| 'fred'@'%' |              10 |
+------------+-----------------+
1 row in set (0.0007 sec)

To reset those counters, you just have to assign again a value to the variable
connection_control_failed_connections_threshold:

SET GLOBAL connection_control_failed_connections_threshold = 4;
Query OK, 0 rows affected (0.0001 sec)

select * from information_schema.connection_control_failed_login_attempts;
Empty set (0.0007 sec

show global status like 'connection_control_%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| Connection_control_delay_generated | 0     |
+-------------------------------------+-------+

Now we can also test without using MySQL Shell or any other MySQL protocol aware client:

$ for i in `seq 1 10`
  do
   time echo meh | nc 127.0.0.3 3306 2>&1 >/dev/null | grep meh
  done
0.027
0.024
0.023
0.022
```

```
1.526
2.032
3.028
4.027
5.032
6.065

We can see that the response time is also increased even if there was no real MySQL authentication attempt.
We can see that the info is also available:

<span style="white-space: normal">select * from information_schema.connection_control_failed_login_attempts;
</span>

+----------------+-----------------+
| USERHOST       | FAILED_ATTEMPTS |
+----------------+-----------------+
| ''@'localhost' |              10 |
+----------------+-----------------+
1 row in set (0.0006 sec)

show global status like 'connection_control_%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| Connection_control_delay_generated | 6     |
+-------------------------------------+-------+
1 row in set (0.0021 sec)

Conclusion

MySQL Connection Control can be very useful in some environments to avoid or limit the inconvenience of a
brute force attack or inappropriate TCP connections.

In MySQL HeatWave Database Service on Oracle Cloud Infrastructure, Connection Control Plugin is enabled by
default.

These are the values used:

show global variables like 'connection_control_%';
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| connection_control_failed_connections_threshold | 3     |
| connection_control_max_connection_delay         | 10000 |
| connection_control_min_connection_delay         | 1000  |
+-----------------------------------------------+-------+
```

InnoDB Table Fragmentation and Defragmentation, Free Space, Within MySQL Server

Notes

```

--> It is not necessary for the proper functioning of MySQL or InnoDB to do any defragmentation.
--> It is solely a possible performance or free space effect.
--> Usually it is just a waste of time and resources to even consider it. But not always.
```

```
--> There can be significant benefits for table or index scans.
--> The first defragmentation of a table will normally take much more time than later ones when the bulk of the
    table is already defragmented.

--> From MySQL 5.6.17 OPTIMIZE TABLE, ALTER TABLE FORCE or ALTER TABLE ENGINE=INNODB for a table that is already
    InnoDB will use online DDL to allow DML during the operation if possible.
--> This makes it far more practical to carry out the internal defragmentation since you can just use
    OPTIMIZE TABLE and then perform filesystem defragmentation.
--> This is the preferred way if you are using a suitable version of the server. You should still try to
    carry out the operation at a low load time because the disk I/O will slow the server significantly.
--> Ensure that innodb_online_alter_log_max_size is large enough to hold all DML on the table during the
    online operation.
--> Use innodb_sort_buffer_size to improve the speed of index rebuilding during the operation; the default
    of 1M is quite small and some tens or hundreds of megabytes can be useful for larger indexes to reduce the
    number of sort merge passes used.
--> The non-blocking inplace method will not be used if old_alter_table is on, the table contains fulltext index
    skip-new is enabled and you are using OPTIMIZE TABLE or if you use COPY to force a copy instead of inplace;
    in these cases the instructions below may be more suitable and you cannot carry out the operation without
    blocking all DML during it.
```

Types of fragmentation

There are several things that may be called fragmentation that can be of interest. There is no single definition of the word fragmentation that applies to InnoDB since the term is used to refer to many different things.

Free space

Sometimes people think of fragmentation as free space that could be reused. Mostly you should ignore this but sometimes it is of use to try to make the free space available to the filesystem or other tables. Any rebuilding of a table will do this, a null ALTER TABLE, say. If you want to check the free space you can do queries like

```
SELECT table_schema, table_name, data_free / 1024 / 1024 AS data_free_in_MB FROM information_schema.tables WHERE
engine LIKE 'InnoDB' AND data_free > 5000*1024*1024;
```

This will return tables with more than five thousand megabytes of reported fee data space. It requires MySQL 5.1.21 or later. Freeing disk space can be useful for many reasons so this can be more useful than other things that are called fragmentation.

Combined defragmentation for all causes

```

--> From version 5.6.17 and later OPTIMIZE TABLE will use online DDL with algorithm=inplace to leave the table
    modifiable during the operation. You should still try to carry out the task at a low load time.
--> Before 5.6.17, if you want to reduce all of the types of fragmentation and are using innodb_file_per_table
    you can do this:

1. Optional, defragment within the filesystem using a filesystem defragmentation tool or copying the *.ibd files
   somewhere and back to the original place.
2. ALTER TABLE tablename DROP INDEX index1, DROP INDEX index2 etc. to drop all of the non-unique secondary index
   Skip this if not using MySQL 5.5 or later.
3. ALTER TABLE tablename ENGINE = InnoDB or OPTIMIZE TABLE tablename or in MySQL 5.5 and later ALTER TABLE
   tablename FORCE to rebuild the clustered index and data rows.
4. ALTER TABLE tablename ADD INDEX index1 and its details, ADD INDEX index2 and its details etc. to add back the
   dropped indexes. Skip this if not using MySQL 5.5 or later.
5. Optional, there may now be some filesystem fragmentation, consider whether it is worth using a filesystem
   defragmentation tool.

```

```
You should not expect to be able to have the server in production service when you do this work. It is extremely
diskintensive and makes very heavy use of the buffer pool so performance for other work will be severely
impacted unless the tables are small.

If a table is changed after this work the fragmentation will gradually return. The effect will be greatest for
secondary indexes or for a clustered/primary index where rows are not inserted in primary key order.

A clustered/primary index will remain well defragmented for existing data if new data is added sequentially and
no deletes or updates are done.

There is no specific schedule for doing this work. You can schedule all or some of it based on your observation
of the rates at which each individual table becomes fragmented enough to make a difference. It is normal for
different tables and indexes to be affected at different rates and to benefit to different degrees.
```

Fragmentation within the shared tablespace, mainly if not using innodb_file_per_table

```
If not using innodb_file_per_table (manual), space is allocated within the shared tablespace: the ibdata1 file
and others you may have allocated. Any ALTER TABLE operation with COPY algorithm will reduce this fragmentation.

It can be quite significant for performance when doing table scans. This cannot affect you for a table that is
using innodb_file_per_table, except that the undo log can cause the file to grow and that can sometimes cause a
little fragmentation.

If doing this defragmenting you should do the smaller tables before the larger ones because a copy of the table
is made within the shared tablespace. That copy will increase the shared tablespace size. If you do the small
tables first, the space freed by rebuilding them will be available to later tables and this will reduce the
amount of growth or possibly eliminate it.

Alternatively, if no foreign keys will be affected, you could alter the table to MyISAM and back to InnoDB.
The conversion to MyISAM will free the space within the shared tablespace and the conversion back to InnoDB
will be able to reuse that space, usually preventing growth. If you want to use the MyISAM method use:

1. ALTER TABLE tablename ENGINE=MyISAM;
2. ALTER TABLE tablename ENGINE=InnoDB;
```

Fragmentation within the filesystem when using innodb_file_per_table

```
This is the most serious form and can cause significant performance issues, most visible when dropping a table.
To check the amount of fragmentation use filefrag on Linux, chkdsk on Windows with FAT32 or the defragmentation
tool in other Windows filesystems. If you find that you have fragmented tables the best way to defragment them
without using a defragmentation tool is to:

1. Shut down mysqld
2. Make a copy of the *.ibd file.
3. Delete the original *.ibd file.
4. Move the copied file to the location of the original

The *.ibd files are located in the sub-directory that normally has the same name as the database the table
belongs to. If you do have a defragmentation tool, it is better to use that.

This is likely to be much faster than the other steps and is likely to deliver the greatest benefit
if there is significant filesystem fragmentation.
```

Encryption Queries

```
SELECT NAME, CURRENT_KEY_VERSION from information_schema.INNODB_TABLESPACES_ENCRYPTION;

To see, which users are using an encrypted connection, run the following query
--------------------------------------------------------------------------------

Note :- performance_schema variable should be turned "ON"

SELECT sbt.variable_value AS tls_version, t2.variable_value AS cipher, processlist_user AS user,
processlist_host AS host FROM performance_schema.status_by_thread  AS sbt JOIN
performance_schema.threads AS t ON t.thread_id = sbt.thread_id JOIN
performance_schema.status_by_thread AS t2 ON t2.thread_id = t.thread_id WHERE
sbt.variable_name = 'Ssl_version' and t2.variable_name = 'Ssl_cipher' ORDER BY tls_version;
```

MySQL Transaction Reporting

Report latest query for transactions active longer than 1 second

```
--
-- Example 8-2. Report latest query for transactions active longer than 1 second
--

SELECT
  ROUND(trx.timer_wait/1000000000000,3) AS trx_runtime,
  trx.thread_id AS thread_id,
  trx.event_id AS trx_event_id,
  trx.isolation_level,
  trx.autocommit,
  stm.current_schema AS db,
  stm.sql_text AS query,
  stm.rows_examined AS rows_examined,
  stm.rows_affected AS rows_affected,
  stm.rows_sent AS rows_sent,
  IF(stm.end_event_id IS NULL, 'running', 'done') AS exec_state,
  ROUND(stm.timer_wait/1000000000000,3) AS exec_time
FROM
       performance_schema.events_transactions_current trx
  JOIN performance_schema.events_statements_current   stm USING (thread_id)
WHERE
      trx.state = 'ACTIVE'
  AND trx.timer_wait > 1000000000000 * 1\G
```

Report transaction summary

```
--
-- Example 8-3. Report transaction summary
--

SELECT
  trx.thread_id AS thread_id,
  MAX(trx.event_id) AS trx_event_id,
  MAX(ROUND(trx.timer_wait/1000000000000,3)) AS trx_runtime,
  SUM(ROUND(stm.timer_wait/1000000000000,3)) AS exec_time,
  SUM(stm.rows_examined) AS rows_examined,
  SUM(stm.rows_affected) AS rows_affected,
  SUM(stm.rows_sent) AS rows_sent
```

```
FROM
       performance_schema.events_transactions_current trx
  JOIN performance_schema.events_statements_history   stm
    ON stm.thread_id = trx.thread_id AND stm.nesting_event_id = trx.event_id
WHERE
      stm.event_name LIKE 'statement/sql/%'
  AND trx.state = 'ACTIVE'
  AND trx.timer_wait > 1000000000000 * 1
GROUP BY trx.thread_id\G
```

Report transaction history

```
--
-- Example 8-4. Report transaction history
--

SELECT
  stm.rows_examined AS rows_examined,
  stm.rows_affected AS rows_affected,
  stm.rows_sent AS rows_sent,
  ROUND(stm.timer_wait/1000000000000,3) AS exec_time,
  stm.sql_text AS query
FROM
  performance_schema.events_statements_history stm
WHERE
       stm.thread_id = 0
  AND  stm.nesting_event_id = 0
ORDER BY stm.event_id;
```

Report basic metrics for committed transactions

```
--
-- Example 8-5. Report basic metrics for committed transactions
--

SELECT
  ROUND(MAX(trx.timer_wait)/1000000000,3) AS trx_time,
  ROUND(SUM(stm.timer_end-stm.timer_start)/1000000000,3) AS query_time,
  ROUND((MAX(trx.timer_wait)-SUM(stm.timer_end-stm.timer_start))/1000000000, 3)
    AS idle_time,
  COUNT(stm.event_id)-1 AS query_count,
  SUM(stm.rows_examined) AS rows_examined,
  SUM(stm.rows_affected) AS rows_affected,
  SUM(stm.rows_sent) AS rows_sent
FROM
      performance_schema.events_transactions_history trx
 JOIN performance_schema.events_statements_history   stm
   ON stm.nesting_event_id = trx.event_id
WHERE
      trx.state = 'COMMITTED'
  AND trx.nesting_event_id IS NOT NULL
GROUP BY
  trx.thread_id, trx.event_id;
```

All tables rows count by using Store Procedure

```
mysql> use test;
Database changed

mysql> DELIMITER $$
mysql> CREATE PROCEDURE `COUNT_ROWS_COUNTS_BY_TABLE`(dbName varchar(128))
    BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE TNAME CHAR(255);
    DECLARE table_names CURSOR for
    SELECT CONCAT("`", TABLE_SCHEMA, "`.`", table_name, "`") FROM INFORMATION_SCHEMA.TABLES where
    TABLE_SCHEMA = dbName;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN table_names;
    DROP TABLE IF EXISTS TABLES_ROWS_COUNTS;
    CREATE TEMPORARY TABLE TABLES_ROWS_COUNTS
    (
         TABLE_NAME CHAR(255),
         RECORD_COUNT INT
       ) ENGINE = MEMORY;

     WHILE done = 0 DO
     FETCH NEXT FROM table_names INTO TNAME;
    IF done = 0 THEN
    SET @SQL_TXT = CONCAT("INSERT INTO TABLES_ROWS_COUNTS(SELECT '" , TNAME  , "' AS TABLE_NAME, COUNT(*) AS
    RECORD_COUNT FROM ", TNAME, ")");
    PREPARE stmt_name FROM @SQL_TXT;
    EXECUTE stmt_name;
    DEALLOCATE PREPARE stmt_name;
    END IF;
    END WHILE;
    CLOSE table_names;

    SELECT * FROM TABLES_ROWS_COUNTS;
    SELECT SUM(RECORD_COUNT) AS TOTAL_DATABASE_RECORD_CT FROM TABLES_ROWS_COUNTS;
  END$$
  DELIMITER ;
Query OK, 0 rows affected (0.01 sec)

mysql> show procedure status LIKE 'COUNT_ROWS_COUNTS_BY_TABLE';

mysql> use test;
Database changed

mysql> call COUNT_ROWS_COUNTS_BY_TABLE('mrsdb');
+-----------------------------------+--------------+
| TABLE_NAME                        | RECORD_COUNT |
+-----------------------------------+--------------+
| `mrsdb`.`mri_container`           |         6979 |
| `mrsdb`.`mri_cp_s_bg_attr_def`    |            0 |
| `mrsdb`.`mri_cr_folder`           |         6048 |
| `mrsdb`.`mri_cr_p_app_project`    |            7 |
| `mrsdb`.`mri_cr_p_app_spec`       |            0 |
| `mrsdb`.`mri_cr_p_application`    |            0 |
| `mrsdb`.`mri_cr_p_data_obj_config`|            4 |
...
```