MySQL 8 New Features

The Good, The Bad And The Ugly

PERCONA LIVE

Vinicius Grippa

Lead Senior Support Engineer at Percona

Vinicius Grippa is a Percona Senior Support Engineer, Oracle Ace, and author of the book Learning MySQL. Vinicius has a Bachelor's degree in Computer Science and has been working with databases for 16 years. He has experience in designing databases for mission-critical applications and, in the last few years, has become a specialist in MySQL and MongoDB ecosystems. Working in the Support team, he has helped Percona customers with hundreds of different cases featuring a vast range of scenarios and complexities. Vinicius is also active in the OS community, participating in virtual rooms like Slack, and speaking at MeetUps, and presenting conferences in Europe, Asia, North and South America.

Agenda

Agenda

- ❑ DDL improvements
- ❑ Parallel read threads
- ❑ REDO log capabilities (archiving, disabling)
- ❑ EXPLAIN tree format
- ❑ Options and Variables
- ❑ Performance schema instrumentation
- ❑ Histograms

MySQL 8

MySQL 8

- MySQL 8 Milestone(8.0.0) release 2016-09-12
- MySQL 8 first GA release(8.0.11) is from 2018-04-19
- MySQL is the 2nd in the DB-Engines ranking (behind Oracle)
- Current MySQL GA version 8.0.30 and 8.0.31 being cooked
- Latest Percona Server for MySQL 8.0.29

DDL improvements

ALGORITHM=INSTANT

- Since MySQL 8.0.12, InnoDB supports `ALGORITHM=INSTANT`
- Operations that support `ALGORITHM=INSTANT`only modify metadata in the data dictionary.
- No exclusive metadata locks are taken on the table during preparation and execution phases of the operation, and table data is unaffected, making the operations instantaneous.

ALGORITHM=INSTANT

```
mysql> alter table sbtest2 add g varchar(100) not null default 0,
algorithm=COPY;
Query OK, 20000000 rows affected (2503.49 sec)
```

```
mysql> alter table sbtest2 add g varchar(100) not null default 0,
algorithm=INSTANT;
Query OK, 0 rows affected (0.38 sec)
```

ALGORITHM=INSTANT

- Still requires a metadata lock!

```
  97 |                                  | NULL                                                                                          |
 585 | Waiting for table metadata lock | alter table sbtest1 add column  test VARCHAR(200) default '', ALGORITHM=INSTANT, LOCK=DEFAULT |
   0 | init                             | show processlist                                                                              |
-----+----------------------------------+-----------------------------------------------------------------------------------------------+
```

Parallel threads for online DDL operations

- The number of parallel threads that can be used to scan clustered index is defined by the `innodb_parallel_read_threads`variable. The default setting is 4.
- The number of parallel threads that sort and load data is controlled by the `innodb_ddl_threads` variable, introduced in MySQL 8.0.27. The default setting is 4.
- The new `innodb_ddl_buffer_size`variable defines the maximum buffer size for DDL operations. Defining a buffer size limit avoids potential out of memory errors for online DDL operations that create or rebuild secondary indexes.

PERCONA LIVE

Parallel threads for online DDL operations

```
mysql> set session innodb_ddl_threads=16;
mysql> set session innodb_parallel_read_threads=16;
mysql> alter table sbtest2 add g varchar(100) not null default 0, algorithm=inplace;
Query OK, 0 rows affected (1 min 13.90 sec)
mysql> alter table sbtest2 add g varchar(100) not null default 0, algorithm=inplace;
Query OK, 0 rows affected (41.66 sec)
```

Parallel read threads

Parallel read threads

- MySQL 8.0.14 has (for now limited) an ability to perform parallel query execution. The `innodb_parallel_read_threads` (default 4) defines the number of threads for parallel index reads.
- It is limited to select count(*) from table queries as well as check table queries.

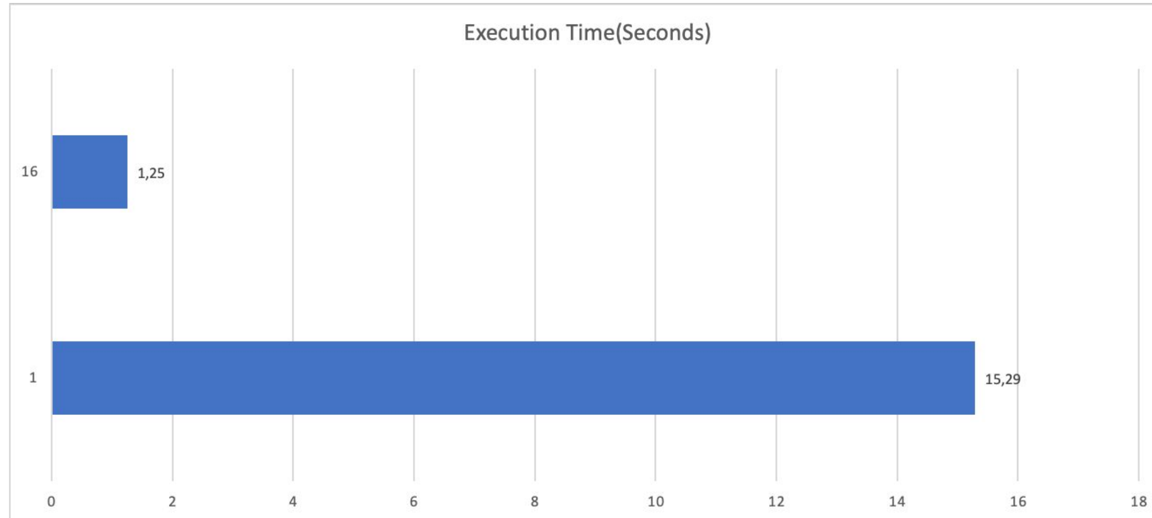
Parallel read threads

```
mysql> set session innodb_parallel_read_threads=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select count(*) from joinit;
+-----------+
| count(*)  |
+-----------+
| 134217728 |
+-----------+
1 row in set (15.29 sec)
```

Parallel read threads

```
mysql> set session innodb_parallel_read_threads=16;
Query OK, 0 rows affected (0.00 sec)

mysql> select count(*) from joinit;
+-----------+
| count(*)  |
+-----------+
| 134217728 |
+-----------+
1 row in set (1.25 sec)
```

Parallel read threads



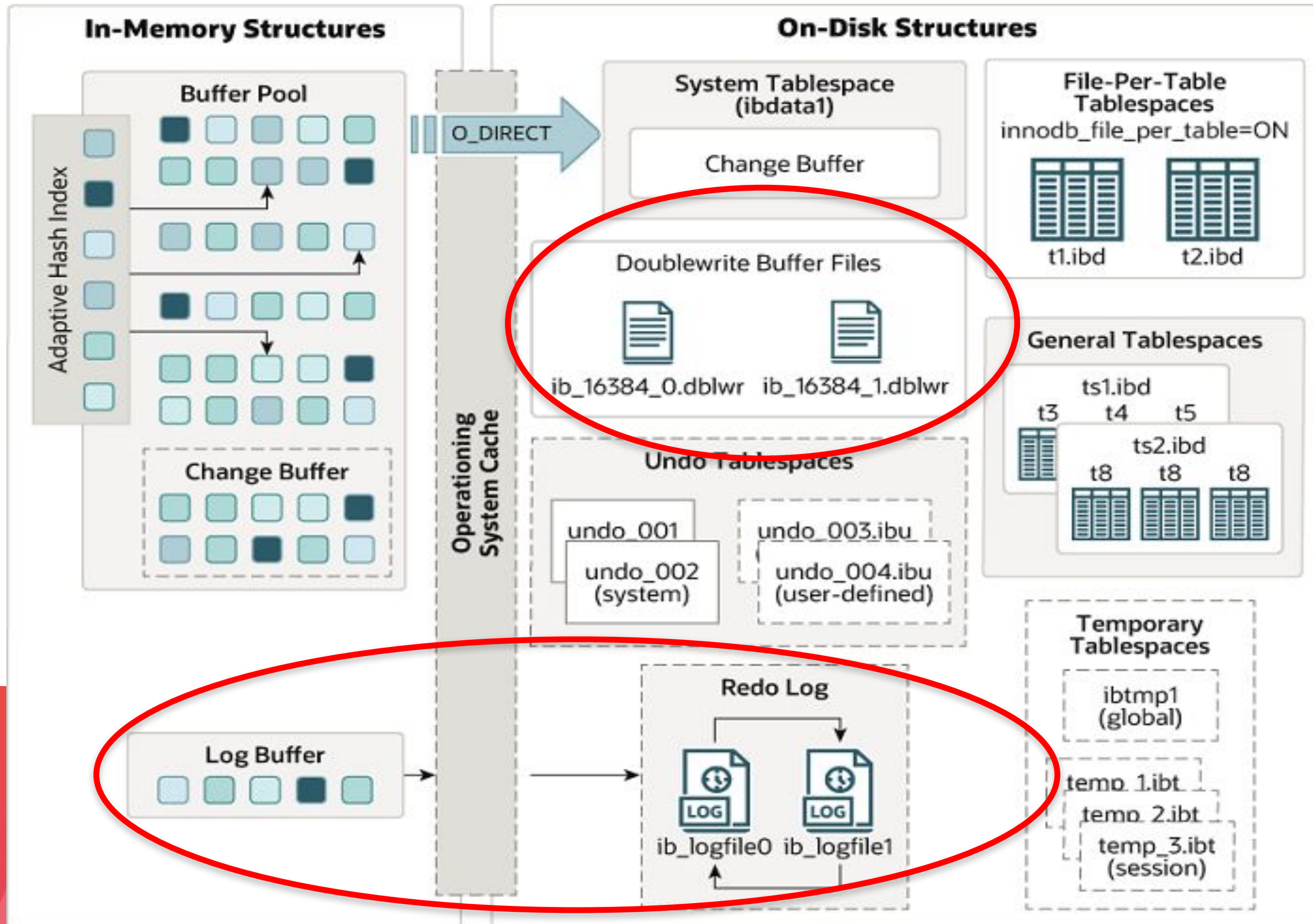
REDO log capabilities

Redo log archiving

- MySQL 8.0.17 releases the redo log archiving(`innodb_redo_log_archive_dirs`).
- Backup utilities that copy redo log records may sometimes fail to keep pace with redo log generation while a backup operation is in progress, resulting in lost redo log records due to those records being overwritten.

Disabling redo log

- As of MySQL 8.0.21, we can disable redo logging using the `ALTER INSTANCE DISABLE INNODB REDO_LOG` statement.
- This functionality is intended for loading data into a new MySQL instance. Disabling redo logging speeds up data loading by avoiding redo log writes and doublewrite buffering.


In-Memory Structures
Buffer Pool
Adaptive Hash Index
Change Buffer
Log Buffer
Operationing System Cache
O_DIRECT
On-Disk Structures
System Tablespace (ibdata1)
Change Buffer
File-Per-Table Tablespaces
innodb_file_per_table=ON
t1.ibd
t2.ibd
Doublewrite Buffer Files
ib_16384_0.dblwr
ib_16384_1.dblwr
General Tablespaces
ts1.ibd
t3
t4
t5
ts2.ibd
t8
t8
t8
Undo Tablespaces
undo_001
undo_002 (system)
undo_003.ibu
undo_004.ibu (user-defined)
Temporary Tablespaces
ibtmp1 (global)
temp_1.ibt
temp_2.ibt
temp_3.ibt (session)
Redo Log
LOG
LOG
ib_logfile0
ib_logfile1

Disabling redo log - benchmark

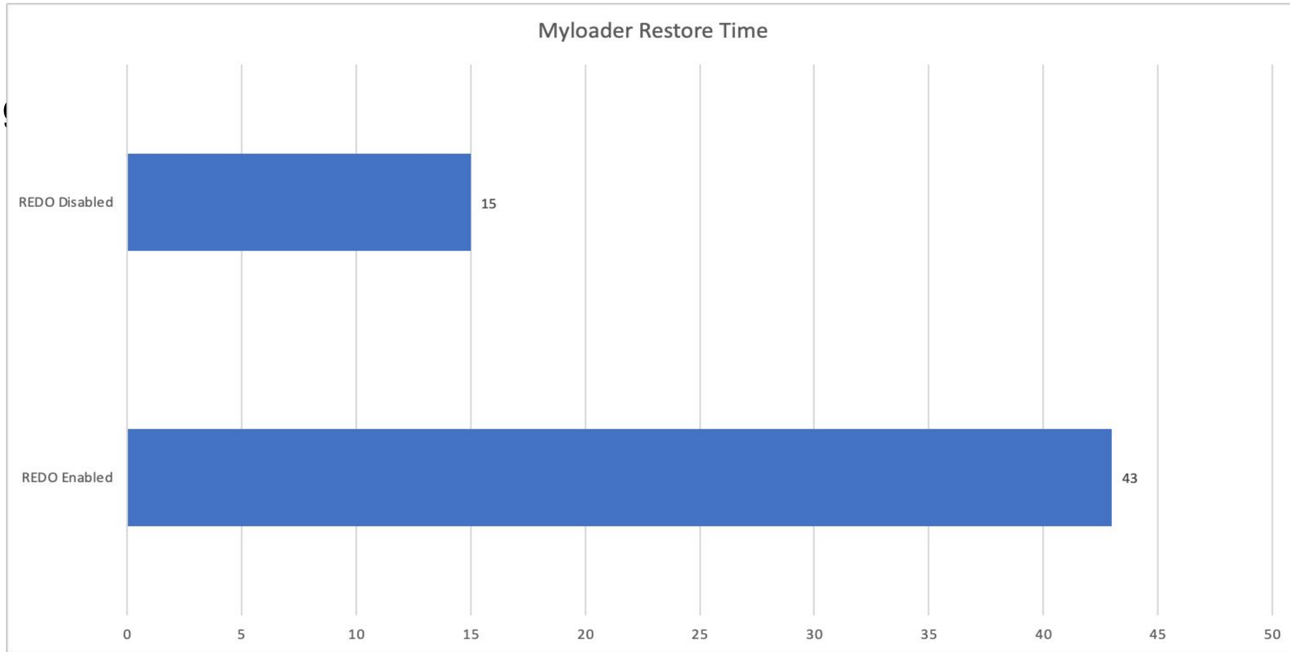
- **Hardware**
- c5.4xlarge
- 16 CPU
- 32 GB RAM
- 400GB provisioned disk with 3000 IOPs

Disabling redo log - benchmark

- **MySQL 8.0.29**
- 20GB of InnoDB Buffer Pool
- 20GB of redo log size
- sync_binlog=0
- innodb_flush_log_at_trx_commit=0
- ~95GB Data Set

PERCONA LIVE

Disablin



Dedicated log writer threads

- As of **MySQL 8.0.22**, you can enable or disable log writer threads using the `innodb_log_writer_threads` variable(default `ON`).
- Dedicated log writer threads can improve performance on high-concurrency systems, but for low-concurrency systems, disabling dedicated log writer threads provides better performance.

Dedicated log writer threads

- As of **MySQL 8.0.22**, you can enable or disable log writer threads using the `innodb_log_writer_threads`variable(default `ON`).
- Dedicated log writer threads can improve performance on high-concurrency systems, but for low-concurrency systems, disabling dedicated log writer threads provides better performance.

Dynamic Redo logs

- As of **MySQL 8.0.30**, you can change the redo log capacity dynamically.
- The new innodb_redo_log_capacity supersede innodb_log_file_size and innodb_log_files_in_group.

Dynamic Redo logs

```
2022-08-03T23:03:12.160014Z 0 [Warning] [MY-013869] [InnoDB] Ignored deprecated configuration parameter innodb_log_file_size. Used innodb_redo_log_capacity instead.
```

Dynamic Redo logs

```
mysql [localhost:47007] {msandbox} ((none)) > set global
innodb_redo_log_capacity = 2*1024*1024*1024;
Query OK, 0 rows affected (0.23 sec)

mysql [localhost:47007] {msandbox} ((none)) > show global variables like
'innodb_redo_log_capacity';
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| innodb_redo_log_capacity | 2147483648 |
+--------------------------+------------+
1 row in set (0.01 sec)
```

EXPLAIN tree format

Explain Tree Format

- As of MySQL 8.0.21, query execution information is displayed using the TREE output format.

```
EXPLAIN FORMAT=TREE
SELECT first_name, last_name, SUM(amount) AS total
FROM staff INNER JOIN payment
  ON staff.staff_id = payment.staff_id
     AND
     payment_date LIKE '2005-08%'
GROUP BY first_name, last_name;

-> Table scan on <temporary>
    -> Aggregate using temporary table
        -> Nested loop inner join (cost=1757.30 rows=1787)
            -> Table scan on staff (cost=3.20 rows=2)
            -> Filter: (payment.payment_date like '2005-08%')  (cost=117.43 rows=894)
                -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)  (cost=117.43 
rows=8043)
```

PERCONA LIVE

Explain tree format

```
EXPLAIN ANALYZE
SELECT first_name, last_name, SUM(amount) AS total
FROM staff INNER JOIN payment
  ON staff.staff_id = payment.staff_id
     AND
     payment_date LIKE '2005-08%'
GROUP BY first_name, last_name;

-> Table scan on <temporary>  (actual time=0.001..0.001 rows=2 loops=1)
    -> Aggregate using temporary table  (actual time=58.104..58.104 rows=2 loops=1)
        -> Nested loop inner join  (cost=1757.30 rows=1787)(actual time=0.816..46.135 rows=5687
loops=1)
            -> Table scan on staff  (cost=3.20 rows=2)(actual time=0.047..0.051 rows=2 loops=1)
            -> Filter: (payment.payment_date like '2005-08%')  (cost=117.43 rows=894) (actual
time=0.464..22.767 rows=2844 loops=2)
                -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)  (cost=117.43
rows=8043) (actual time=0.450..19.988 rows=8024 loops=2)
```

Explain tree format

There are several new measurements here:

- Actual time to get first row (in milliseconds)
- Actual time to get all rows (in milliseconds)
- Actual number of rows read
- Actual number of loops

Filter: (payment.payment_date like '2005-08%')
(cost=117.43 **rows=894**)
(actual time=0.464..22.767 **rows=2844 loops=2**)

```
1 EXPLAIN ANALYZE
2 SELECT first_name, last_name, SUM(amount) AS total
3 FROM staff INNER JOIN payment
4   ON staff.staff_id = payment.staff_id
5      AND
6      payment_date LIKE '2005-08%'
7 GROUP BY first_name, last_name;
8
9 -> Table scan on <temporary>  (actual time=0.001..0.001 rows=2 loops=1)
10     -> Aggregate using temporary table  (actual time=58.104..58.104 rows=2 loops=1)
11         -> Nested loop inner join  (cost=1757.30 rows=1787) (actual time=0.816..46.135 rows=5687
loops=1)
12             -> Table scan on staff  (cost=3.20 rows=2) (actual time=0.047..0.051 rows=2 loops=1)
13             -> Filter: (payment.payment_date like '2005-08%')  (cost=117.43 rows=894) (actual
time=0.464..22.767 rows=2844 loops=2)
14                 -> Index lookup on payment using idx_fk_staff_id (staff_id=staff.staff_id)
(cost=117.43 rows=8043) (actual time=0.450..19.988 rows=8024 loops=2)
```

PERCONALIVE

Explain tree format

What can we do with this information?

- If you wonder what it’s taking so long, look at the timing. Where does the executor spend its time?
- If you wonder why the optimizer chose that plan, look at the row counters. A large difference (i.e., a couple of orders of magnitude or more) between the estimated number of rows and the actual number of rows is a sign that you should look closer at it. The optimizer chooses its plan based on the estimate, but looking at the actual execution may tell you that another plan would have been better.

Options and variables

innodb_dedicated_server

- Introduced in MySQL 8.0.3

When `innodb_dedicated_server` is enabled, InnoDB automatically configures the following variables:

- `innodb_buffer_pool_size`
- `innodb_log_file_size`
- `innodb_log_files_in_group` (as of MySQL 8.0.14)
- `innodb_flush_method`

innodb_buffer_pool_in_core_file

- Introduced in MySQL 8.0.14
- Disabling the `innodb_buffer_pool_in_core_file`variable reduces the size of core files by excluding InnoDB buffer pool pages.

PERCONA LIVE

innodb_use_fdatasync

- Introduced in MySQL 8.0.26
- Allows MySQL to use ftadasync() instead of fsync().
- An fdatasync() call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

PERCONALIVE

innodb_deadlock_detect

- Introduced in MySQL 8.0
- This option is used to disable deadlock detection.

binlog_expire_logs_auto_purge

- Introduced in MySQL 8.0.29
- Enables or disables automatic purging of binary log files (default `ON`).
- This takes precedence over any setting for `binlog_expire_logs_seconds`

global_connection_memory_limit

- Introduced in MySQL 8.0.28
- The default is virtually infinite (**16 EB**)
- Set the total amount of memory that can be used by all user connections (including `root` user)

global_connection_memory_tracking

- Introduced in MySQL 8.0.28
- Determines whether the server calculates `Global_connection_memory` (Default `FALSE`).

admin_port / admin_address

- Introduced in MySQL 8.0.14
- TCP/IP number to use for connections on administrative interface and IP address to bind for connections.

Performance schema instrumentation

Memory Instrumentation

- Introduced in MySQL 5.7 and enabled by default in MySQL 8.0, the Performance_Schema's Memory instrumentation allows us to have a better overview of what MySQL is allocating and why.

Memory Instrumentation

```
SELECT SUBSTRING_INDEX(event_name,'/',2) AS code_area,
       sys.format_bytes(SUM(current_alloc)) AS current_alloc
FROM sys.x$memory_global_by_current_bytes
GROUP BY SUBSTRING_INDEX(event_name,'/',2)
ORDER BY SUM(current_alloc) DESC;
```

Memory Instrumentation

```
+---------------------------+-----------------+
| code_area                 | current_alloc   |
+---------------------------+-----------------+
| memory/innodb             | 21.09 GiB       |
| memory/performance_schema | 236.14 MiB      |
| memory/sql                | 36.26 MiB       |
| memory/mysys              | 9.43 MiB        |
| memory/temptable          | 1.00 MiB        |
| memory/mysqld_openssl     | 604.85 KiB      |
| memory/csv                | 18.07 KiB       |
| memory/mysqlx             | 3.31 KiB        |
| memory/myisam             |  728 bytes      |
| memory/blackhole          |  120 bytes      |
| memory/vio                |   80 bytes      |
+---------------------------+-----------------+
11 rows in set, 1 warning (0.03 sec)
```

TempTable Memory Usage

```
mysql>select * from sys.memory_global_by_current_bytes  where event_name like
'%temp%'\G
```

PERCONA LIVE

Memory Usage Per User

```
mysql> select * from sys.memory_by_user_by_current_bytes\G
```

PERCONALIVE

Memory Usage Per User

```
*************************** 1. row ***************************
              user: root
current_count_used: 13094
 current_allocated: 4.51 MiB
 current_avg_alloc:  361 bytes
 current_max_alloc: 1.29 MiB
   total_allocated: 15.33 TiB
*************************** 2. row ***************************
              user: sysbench
current_count_used: 0
 current_allocated:    0 bytes
 current_avg_alloc:    0 bytes
 current_max_alloc:    0 bytes
   total_allocated: 4.34 TiB
2 rows in set (0.00 sec)
```

PERCONALIVE

Memory Usage Per Thread

```
mysql> select * from memory_by_thread_by_current_bytes\G
```

Memory Usage Per Thread

```
*************************** 1. row ***************************
         thread_id: 1072
              user: root@localhost
current_count_used: 13100
 current_allocated: 11.57 MiB
 current_avg_alloc:  925 bytes
 current_max_alloc: 8.00 MiB
   total_allocated: 152.40 MiB
*************************** 2. row ***************************
         thread_id: 1
              user: sql/main
current_count_used: 6552
 current_allocated: 1.14 MiB
 current_avg_alloc:  182 bytes
 current_max_alloc: 357.13 KiB
   total_allocated: 6.04 MiB
```

PERCONA LIVE

Histograms

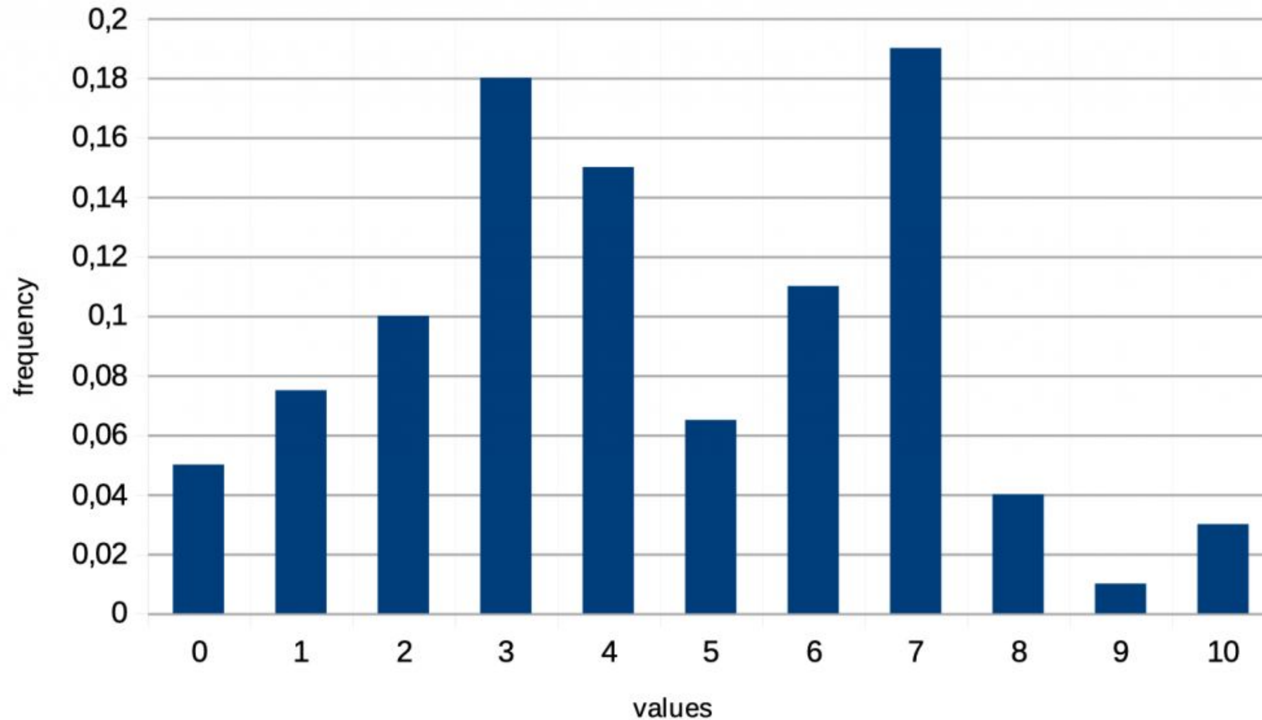
Histograms

- We can define a histogram as a good approximation of the data distribution of the values in a column.
- So in MySQL, histograms help the optimizer to find the most efficient query Plan.

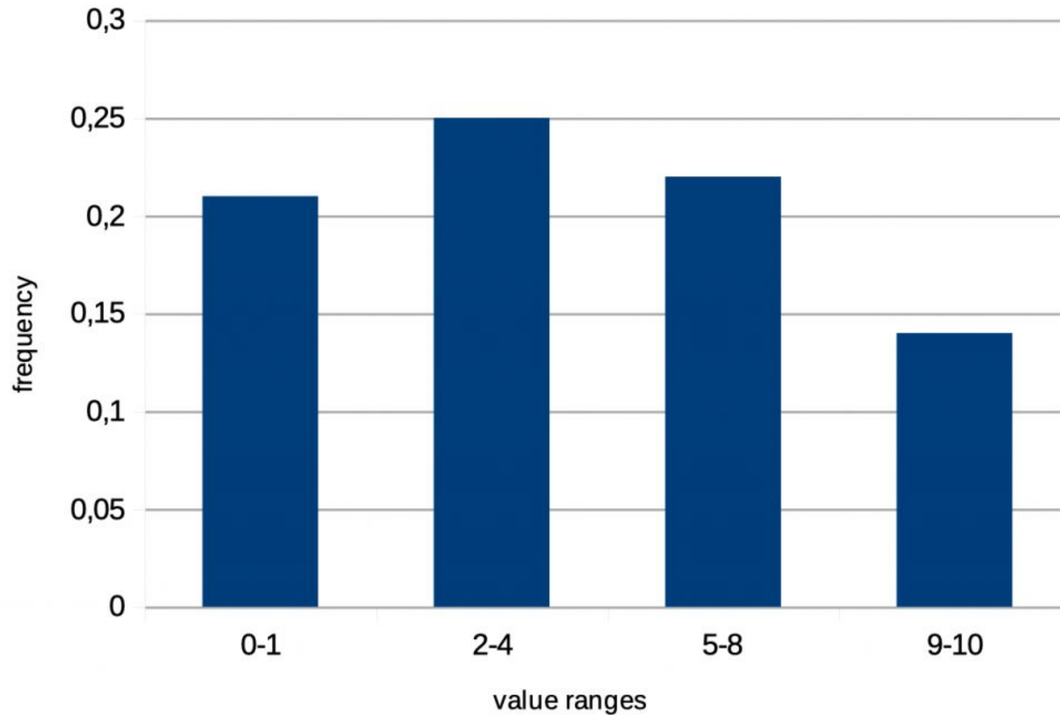
Histograms

- Histograms can be of two types
  - Singleton
  - Equi-height

Histograms - Singleton



Histograms - Equi-height



PERCONA LIVE

Histograms

```
mysql>  ANALYZE TABLE country UPDATE HISTOGRAM ON Region;
```

Histograms

```
mysql> > SELECT SUBSTRING_INDEX(v, ':', -1) value, concat(round(c*100,1),'%') cumulfreq,
CONCAT(round((c - LAG(c, 1, 0) over()) * 100,1), '%') freq      FROM information_schema.column_statistics,
JSON_TABLE(histogram->'$.buckets','$[*]' COLUMNS(v VARCHAR(60) PATH '$[0]', c double PATH '$[1]')) hist
WHERE schema_name  = 'world' and table_name = 'country' and column_name = 'region';
+---------------------------+-----------+------+
| value                     | cumulfreq | freq |
+---------------------------+-----------+------+
| Antarctica                | 2.1%      | 2.1% |
| Australia and New Zealand | 4.2%      | 2.1% |
[…]
| Central Africa            | 20.1%     | 3.8% |
| Central America           | 23.4%     | 3.3% |
| Eastern Africa            | 31.8%     | 8.4% |
[...]
| Western Africa            | 96.2%     | 7.1% |
| Western Europe            | 100%      | 3.8% |
+---------------------------+-----------+------+
25 rows in set (0.00 sec)
```

Histograms

- Histogram statistics are particularly useful for non-indexed columns, as shown in the example.

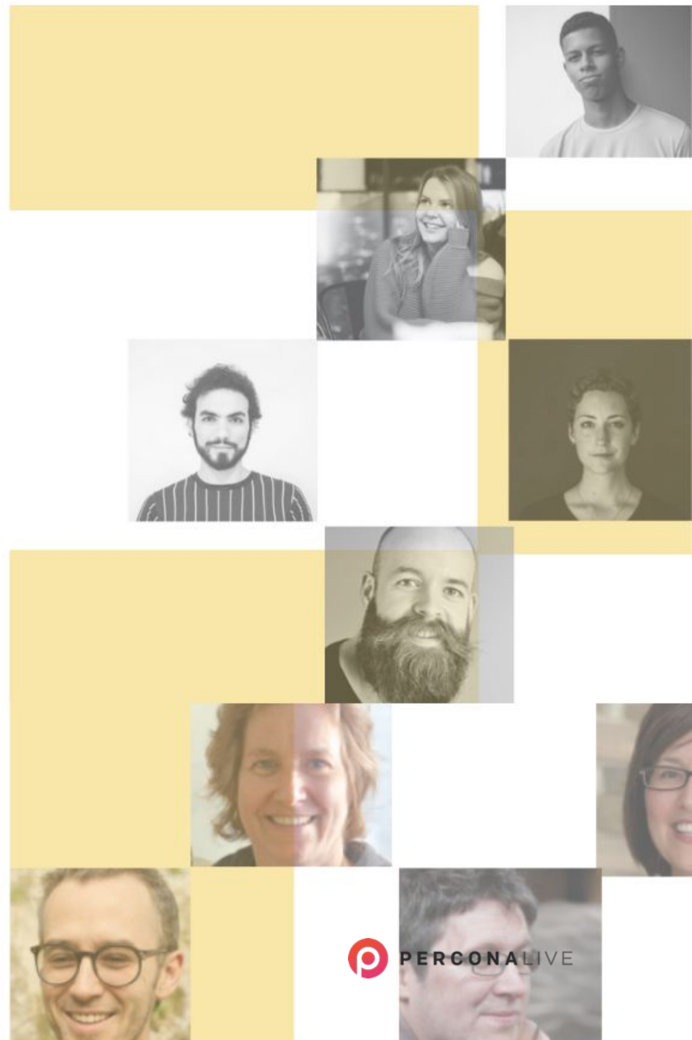
- Execution plans that can rely on indexes are usually the best, but histograms can help in some edge cases or when creating a new index is a bad idea.

PERCONA

We're hiring

Join our team!

OPEN POSITIONS



#RemoteWork

APPLY NOW: percona.com/careers

谢谢

Thank you

Grazie

Obrigado

Gracias

percona.com/contact | info@percona.com

vinicius.grippa@percona.com