ORACLE

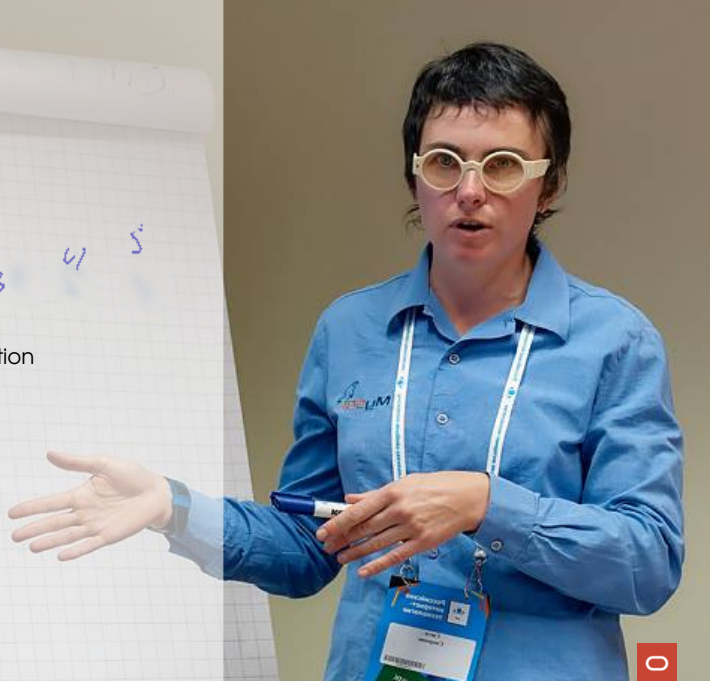
Introduction into MySQL Query Tuning

for Dev[Op]s.

Sveta Smirnova
Principal Support Engineering Coordinator
Percona, Support
July 22, 2025

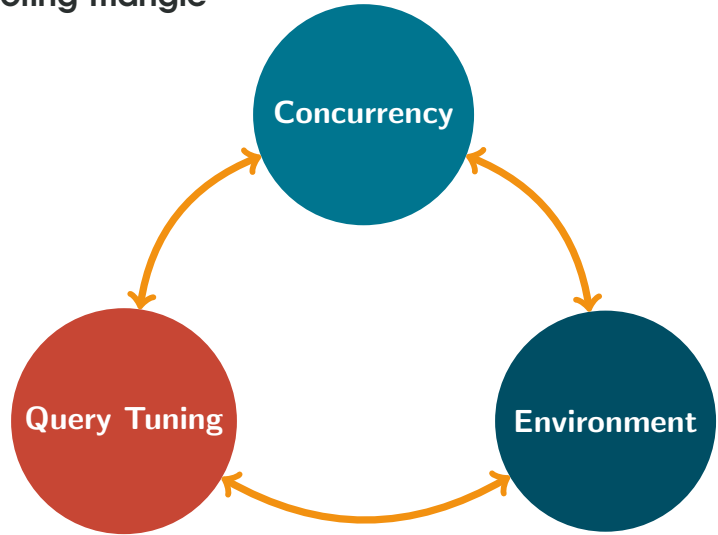
Sveta Smirnova

- MySQL Support Engineer
- Author of MySQL Troubleshooting
- Author of MySQL Cookbook, 4th Edition
- Maintainer of Percona Toolkit
- Product Manager of PMM Dump

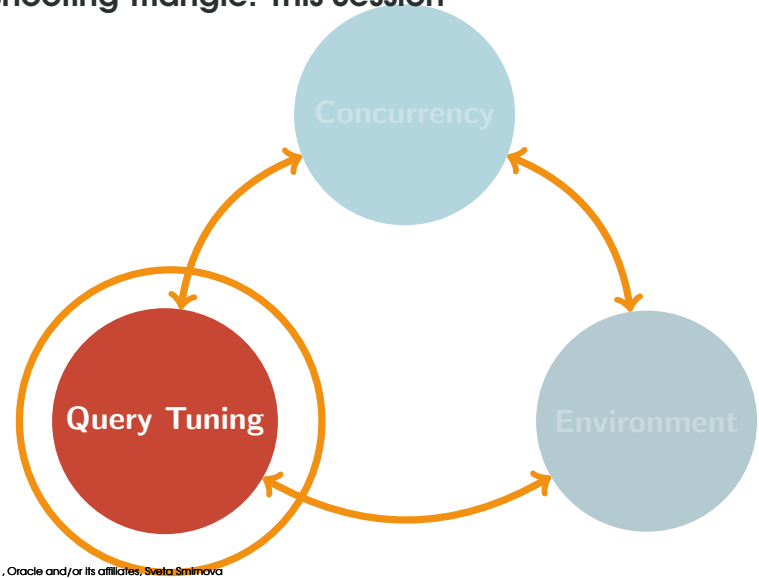


Introduction

Troubleshooting Triangle



Troubleshooting Triangle: This Session



Slow Query

- Standalone Server
  - Runs for a long time
  - Reads more data than needed
  - Sets more locks than needed
  - Holds locks longer
  - **Affects parallel connections**

Slow Query

- Standalone Server
  - Runs for a long time
  - Reads more data than needed
  - Sets more locks than needed
  - Holds locks longer
  - **Affects parallel connections**
- Asynchronous Source Server
  - Overall usage increase of CPU, disk, and memory slows down all the operations

Slow Query

- Standalone Server
  - Runs for a long time
  - Reads more data than needed
  - Sets more locks than needed
  - Holds locks longer
  - **Affects parallel connections**
- Asynchronous Source Server
  - Overall usage increase of CPU, disk, and memory slows down all the operations
- Asynchronous Replica
  - Statement-based queries replicate slow
  - Overall performance drop will affect how fast events from the source are replicated

Slow Query

- Standalone Server
  - Runs for a long time
  - Reads more data than needed
  - Sets more locks than needed
  - Holds locks longer
  - **Affects parallel connections**
- Asynchronous Source Server
  - Overall usage increase of CPU, disk, and memory slows down all the operations
- Asynchronous Replica
  - Statement-based queries replicate slow
  - Overall performance drop will affect how fast events from the source are replicated
- Synchronous Writer
  - Performance decreases with scale factor of number of nodes
  - More chances for the local transaction to be rolled back due to conflict on multiple primary
  - Overall usage increase of CPU, disk, and memory slows down all the operations

Slow Query

- Standalone Server
  - Runs for a long time
  - Reads more data than needed
  - Sets more locks than needed
  - Holds locks longer
  - **Affects parallel connections**
- Asynchronous Source Server
  - Overall usage increase of CPU, disk, and memory slows down all the operations
- Asynchronous Replica
  - Statement-based queries replicate slow
  - Overall performance drop will affect how fast events from the source are replicated
- Synchronous Writer
  - Performance decreases with scale factor of number of nodes
  - More chances for the local transaction to be rolled back due to conflict on multiple primary
  - Overall usage increase of CPU, disk, and memory slows down all the operations
- Synchronous Reader
  - Higher resource usage may delay or even stop replication

Emergency Situations

- No time for query tuning

Emergency Situations

- No time for query tuning
- When?
  - Black Friday
  - Cyber Monday
  - Competitions
  - **Any time-limited event**

Emergency Situations

- No time for query tuning
- When?
- Typical Support suggestion
  - Adjust options
    - **Ensure performance configuration is optimal**
    - Turn off safety
    - Pray no hardware would fail
  - Buy hardware

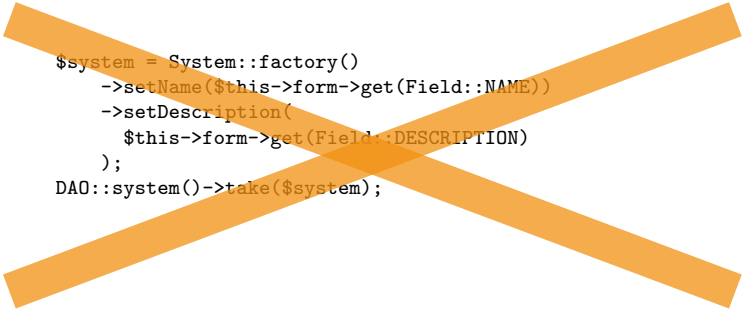
Emergency Situations

- No time for query tuning
- When?
- Typical Support suggestion
  - Adjust options
    - **Ensure performance configuration is optimal**
    - Turn off safety
    - Pray no hardware would fail
  - Buy hardware
- ☹ We have no other choice!

Emergency Situations

- No time for query tuning
- When?
- Typical Support suggestion
  - Adjust options
    - **Ensure performance configuration is optimal**
    - Turn off safety
    - Pray no hardware would fail
  - Buy hardware
- ☹ We have no other choice!
- **You have!**

Always Tune Raw Query



```
$system = System::factory()
    ->setName($this->form->get(Field::NAME))
    ->setDescription(
      $this->form->get(Field::DESCRIPTION)
    );
DAO::system()->take($system);
```

Always Tune Raw Query

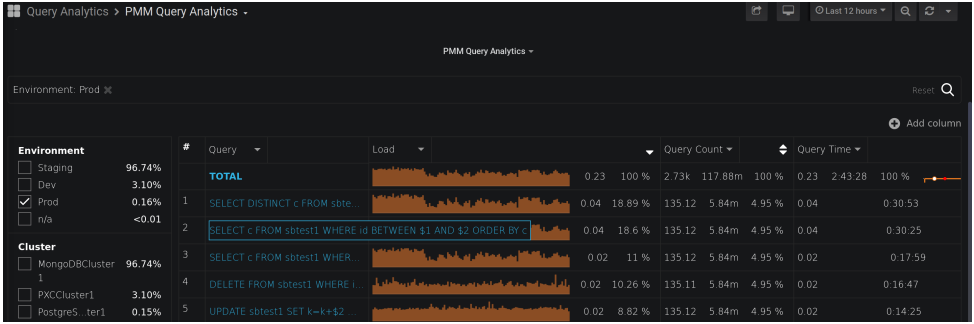
```
cursor = conn.cursor()
q = '''UPDATE `foo` SET my_date=NOW(),
    subject = %s,
    msg = %s,
    address = %s,
    updated_at = NOW()
    WHERE id=%s
    '''
cursor.execute(q, [
  remote_resp.get('subject'),
  remote_resp.get('msg'),
  remote_resp.get('address'),
  my_id
])
```

Always Tune Raw Query

```
SELECT dept_name, title, gender,
       min(salary) AS mins,
       max(salary) AS maxs
FROM employees
JOIN salaries USING(emp_no)
JOIN titles USING(emp_no)
JOIN dept_emp USING(emp_no)
JOIN departments USING(dept_no)
JOIN dept_manager USING(dept_no)
WHERE dept_manager.to_date = '9999-01-01'
GROUP BY dept_name, title, gender
ORDER BY gender, maxs DESC;
```

Sources of the Raw Query

- Application log
- Slow Query Log
- Performance Schema
- Monitoring software, e.g. PMM QAN



Slow is Relative

- Mind your data!
- 75,000,000 rows
  - `(INT, INT)`
    - 75,000,000 * (4 + 4) = **600,000,000 B** = **572 MB**
  - `(INT, INT, DATETIME, VARCHAR(255), VARCHAR(255))`
    - 75,000,000 * (4 + 4 + 8 + 256 + 256) = **39,600,000,000** bytes = **37 G**
  - 39,600,000,000 / 600,000,000 = 66

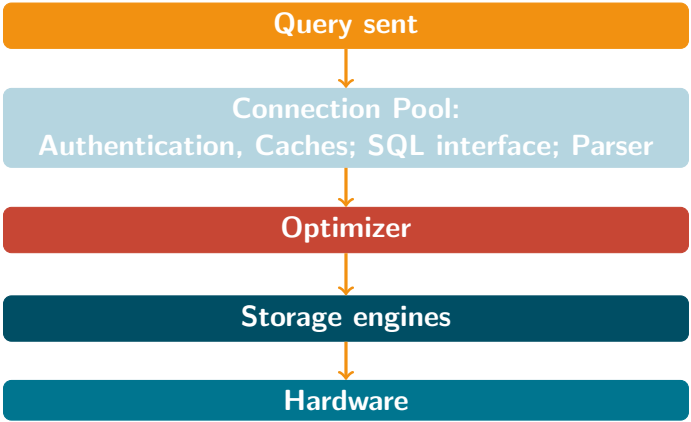
Slow is Relative

- Mind your data!
- 75,000,000 rows
  - `(INT, INT)`
    - 75,000,000 * (4 + 4) = **600,000,000 B** = **572 MB**
  - `(INT, INT, DATETIME, VARCHAR(255), VARCHAR(255))`
    - 75,000,000 * (4 + 4 + 8 + 256 + 256) = **39,600,000,000** bytes = **37 G**
  - 39,600,000,000 / 600,000,000 = 66
- Mind use case
  - Popular website
  - Admin interface
  - Weekly cron job

Slow is Relative

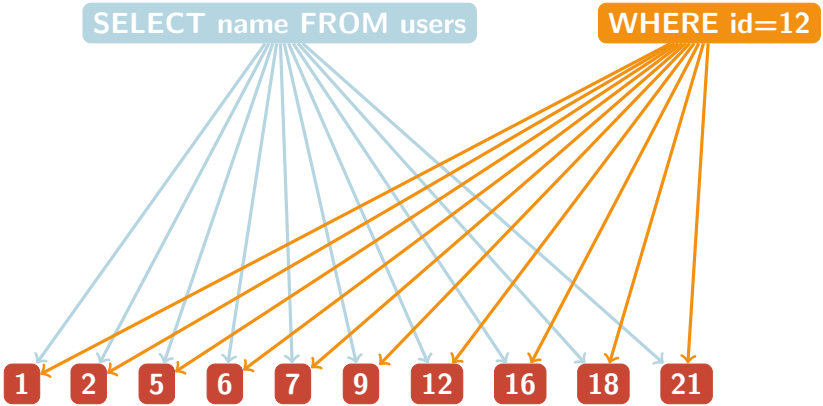
- Mind your data!
- 75,000,000 rows
  - `(INT, INT)`
    - 75,000,000 * (4 + 4) = **600,000,000 B** = **572 MB**
  - `(INT, INT, DATETIME, VARCHAR(255), VARCHAR(255))`
    - 75,000,000 * (4 + 4 + 8 + 256 + 256) = **39,600,000,000** bytes = **37 G**
  - 39,600,000,000 / 600,000,000 = 66
- Mind use case
  - Popular website
  - Admin interface
  - Weekly cron job
- Mind location
  - Server, used by multiple connections
  - Dedicated for OLAP queries

Query Execution Workflow

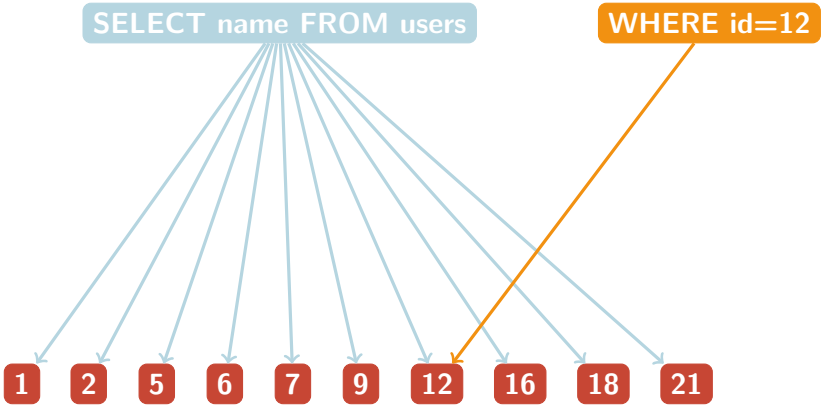


MySQL Indexes

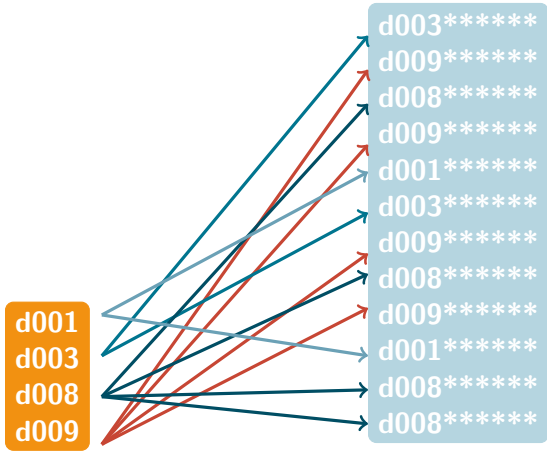
Full Table Scan



Index Access



MySQL Indexes



How to Create an Index

- Single column

```
CREATE INDEX index_name ON the_table(the_column)
```

- Multiple columns

```
CREATE INDEX index_name ON the_table(column1, column2)
```

How to Create an Index

- Single column

```
CREATE INDEX index_name ON the_table(the_column)
ALTER TABLE table_name ADD INDEX [index_name] (the_column)
```

- Multiple columns

```
CREATE INDEX index_name ON the_table(column1, column2)
ALTER TABLE table_name ADD INDEX [index_name] (column1, column2)
```

When MySQL Uses Indexes

- Conditions
  - `WHERE the_column = a_value`
  - `WHERE the_column IN(value1, value2, value3)`
  - `WHERE the_column LIKE 'value%'`
  - ~~`WHERE the_column LIKE '%value'`~~

When MySQL Uses Indexes

- Conditions
  - `WHERE the_column = a_value`
  - `WHERE the_column IN(value1, value2, value3)`
  - `WHERE the_column LIKE 'value%'`
  - ~~`WHERE the_column LIKE '%value'`~~
  - `WHERE left_part = value1 AND right_part = value2`
  - `WHERE left_part = value1 OR right_part = value2`
  - `WHERE right_part = value1 AND left_part = value2`
  - ~~`WHERE right_part = value1 OR left_part = value2`~~

When MySQL Uses Indexes

- Conditions
- Joins
  - `table1 JOIN table2 ON table1.column1 = table2.column2`
  - Same as `FROM table1, table2 WHERE table1.column1 = table2.column2`

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
  - `GROUP BY the_column`
  - `GROUP BY left_part, right_part`
  - ~~`GROUP BY right_part, left_part`~~
  - ~~`GROUP BY the_index, another_index`~~

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
- `ORDER BY`
  - `ORDER BY the_column`
  - `ORDER BY left_part, right_part`
  - ~~`ORDER BY right_part, left_part`~~
  - ~~`ORDER BY the_index, another_index`~~

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
- `ORDER BY`
  - `ORDER BY the_column`
  - `ORDER BY left_part, right_part`
  - ~~`ORDER BY right_part, left_part`~~
  - ~~`ORDER BY the_index, another_index`~~

5.7 ~~`ORDER BY left_part DESC, right_part ASC`~~

8.0+ `ORDER BY left_part DESC, right_part ASC`

- `left_part` **must** be **de**scending
- `right_part` **must** be **a**scending
- `the_index(left_part DESC, right_part ASC)`

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
- `ORDER BY`
- Expressions
  - Deterministic, **built-in**
    - Return same value for the same argument
    - `WHERE the_column = FLOOR(123.45)`

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
- `ORDER BY`
- Expressions
  - Deterministic, **built-in**
    - Return same value for the same argument
    - `WHERE the_column = FLOOR(123.45)`
  - Non-deterministic
    - Return different values for different invocations
    - ~~`WHERE the_column = RAND() * 100`~~

When MySQL Uses Indexes

- Conditions
- Joins
- `GROUP BY`
- `ORDER BY`
- Expressions
  - Deterministic, **built-in**
    - Return same value for the same argument
    - `WHERE the_column = FLOOR(123.45)`
  - Non-deterministic
    - Return different values for different invocations
    - ~~`WHERE the_column = RAND() * 100`~~
  - Stored functions and UDFs
    - Indexes are not used

Use generated column indexes

When to Add Indexes?

- Identify queries not using indexes
  - Status variables

```
mysql> select * from performance_schema.session_status
    -> where variable_name in ('Created_tmp_tables',
    -> 'Created_tmp_disk_tables', 'Select_full_join',
    -> 'Select_full_range_join', 'Select_range',
    -> 'Select_range_check', 'Select_scan',
    -> 'Sort_merge_passes', 'Sort_range', 'Sort_rows',
    -> 'Sort_scan') and variable_value > 0;
+------------------------+----------------+
| VARIABLE_NAME          | VARIABLE_VALUE |
+------------------------+----------------+
| Select_scan            | 2              |
+------------------------+----------------+
1 row in set (0,00 sec)
```

When to Add Indexes?

- Identify queries not using indexes

```
mysql> show global status like 'Handler_read%';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 31    |
| Handler_read_key      | 1909  |
| Handler_read_last     | 0     |
| Handler_read_next     | 4393  |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 1135  |
+-----------------------+-------+
```

When to Add Indexes?

- Identify queries not using indexes
  - PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_*

```
mysql> select * from events_statements_history_long
    -> where sql_text like 'select count(*) from employees join %'\G
*************************** 1. row ***************************
...
ROWS_SENT: 1                          SELECT_RANGE_CHECK: 0
ROWS_EXAMINED: 541058                 SELECT_SCAN: 1
CREATED_TMP_DISK_TABLES: 0            SORT_MERGE_PASSES: 0
CREATED_TMP_TABLES: 0                 SORT_RANGE: 0
SELECT_FULL_JOIN: 0                   SORT_ROWS: 0
SELECT_FULL_RANGE_JOIN: 0             SORT_SCAN: 0
SELECT_RANGE: 0                       NO_INDEX_USED: 0
```

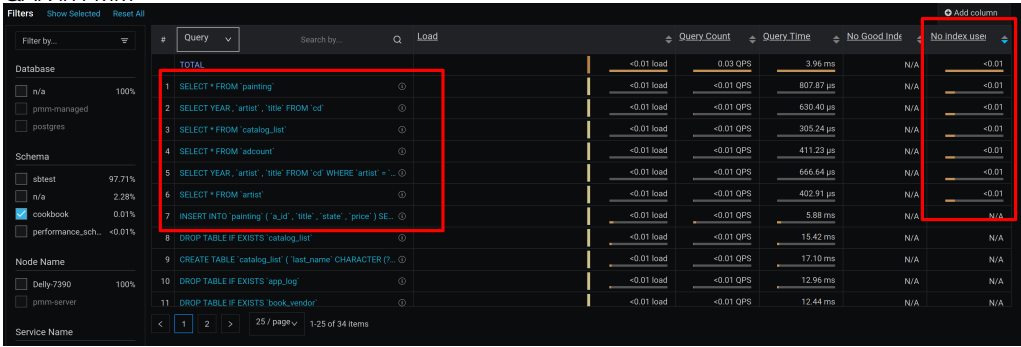
When to Add Indexes?

- Identify queries not using indexes
  - Slow query log

```
# Time: 2020-10-11T23:34:03.701871Z
# User@Host: root[root] @ localhost [127.0.0.1]  Id:   506
# Query_time: 0.024106  Lock_time: 0.000091
Rows_sent: 1  Rows_examined: 1000
SET timestamp=1602459243;
SELECT c FROM sbtest1 WHERE id=996;
```

When to Add Indexes?

- Identify queries not using indexes
  - QAN in PMM



When to Add Indexes?

- Identify queries not using indexes
- Analyze if adding an index will help

When to Add Indexes?

- Identify queries not using indexes
- Analyze if adding an index will help
- Add the index

When to Add Indexes?

- Identify queries not using indexes
- Analyze if adding an index will help
- Add the index
- **Test first!**
  - `ADD INDEX` is an expensive operation

Use `pt-online-schema-change`

Play with index visibility while testing

EXPLAIN: How Optimizer Works

EXPLAIN: rows

```
mysql> EXPLAIN SELECT first_name, last_name, title, salary FROM employees
    -> JOIN salaries USING(emp_no) JOIN titles USING(emp_no)
    -> WHERE salary = (SELECT MAX(salary) FROM salaries
    ->                 JOIN titles WHERE titles.to_date > CURDATE());
+----+-------------+-----------+------------+------+---------------+---------+---------+
| id | select_type | table     | partitions | type | possible_keys | key     | key_len |
+----+-------------+-----------+------------+------+---------------+---------+---------+
|  1 | PRIMARY     | employees | NULL       | ALL  | PRIMARY       | NULL    | NULL ...
|  1 | PRIMARY     | salaries  | NULL       | ref  | PRIMARY       | PRIMARY | 4    ...
|  1 | PRIMARY     | titles    | NULL       | ref  | PRIMARY       | PRIMARY | 4    ...
|  2 | SUBQUERY    | titles    | NULL       | ALL  | NULL          | NULL    | NULL ...
|  2 | SUBQUERY    | salaries  | NULL       | ALL  | NULL          | NULL    | NULL ...
+----+-------------+-----------+------------+------+---------------+---------+---------+
+----+-...-+---------------------------+---------+----------+-----------------------------+
| id | ... | ref                       | rows    | filtered | Extra                       |
+----+-...-+---------------------------+---------+----------+-----------------------------+
|  1 | ... | NULL                      |  299113 |   100.00 | NULL                        |
|  1 | ... | employees.employees.emp_no |       9 |    10.00 | Using where                 |
|  1 | ... | employees.employees.emp_no |       1 |   100.00 | Using index                 |
|  2 | ... | NULL                      |  442189 |    33.33 | Using where                 |
|  2 | ... | NULL                      | 2838426 |   100.00 | Using join buffer (hash join) |
+----+-...-+---------------------------+---------+----------+-----------------------------+
5 rows in set, 1 warning (0,01 sec)
```

```
299113 * 9 * 1 * 442189 * 2838426 = 3,378,806,408,204,514,738
```

EXPLAIN: type

```
mysql> EXPLAIN SELECT title, MAX(salary) AS maxs FROM employees
    -> JOIN salaries USING(emp_no) JOIN titles USING(emp_no)
    -> GROUP BY title ORDER BY maxs DESC;
+----+-------------+-----------+------------+-------+---------------+---------+---------+
| id | select_type | table     | partitions | type  | possible_keys | key     | key_len |
+----+-------------+-----------+------------+-------+---------------+---------+---------+
|  1 | SIMPLE      | employees | NULL       | index | PRIMARY       | PRIMARY | 4 ...   |
|  1 | SIMPLE      | titles    | NULL       | ref   | PRIMARY,title | PRIMARY | 4 ...   |
|  1 | SIMPLE      | salaries  | NULL       | ref   | PRIMARY       | PRIMARY | 4 ...   |
+----+-------------+-----------+------------+-------+---------------+---------+---------+
```

Query Execution Type

ALL Read all rows

Query Execution Type

ALL	Read all rows
index	Read all rows from the index

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries

- `index_subquery`
- `unique_subquery`

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries
merge	Combination of two indexes

- `index_merge`

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries
merge	Combination of two indexes
ref	Comparision with single value

- `ref_or_null`
- `ref`
- `eq_ref`

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries
merge	Combination of two indexes
ref	Comparision with single value
fulltext	Fulltext index

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries
merge	Combination of two indexes
ref	Comparision with single value
fulltext	Fulltext index
const	Single row

Query Execution Type

ALL	Read all rows
index	Read all rows from the index
range	Range: `N ... M`
subquery	Subqueries
merge	Combination of two indexes
ref	Comparision with single value
fulltext	Fulltext index
const	Single row
system	Table has single row

EXPLAIN: key and possible_keys

```
mysql> EXPLAIN SELECT title, MAX(salary) AS maxs FROM employees
    -> JOIN salaries USING(emp_no) JOIN titles USING(emp_no)
    -> GROUP BY title ORDER BY maxs DESC;
+----+-------------+-----------+------------+-------+---------------+---------+---------+
| id | select_type | table     | partitions | type  | possible_keys | key     | key_len |
+----+-------------+-----------+------------+-------+---------------+---------+---------+
|  1 | SIMPLE      | employees | NULL       | index | PRIMARY       | PRIMARY | 4 ...   |
|  1 | SIMPLE      | titles    | NULL       | ref   | PRIMARY,title | PRIMARY | 4 ...   |
|  1 | SIMPLE      | salaries  | NULL       | ref   | PRIMARY       | PRIMARY | 4 ...   |
+----+-------------+-----------+------------+-------+---------------+---------+---------+
```

EXPLAIN: JOIN order

```
mysql> EXPLAIN SELECT title, MAX(salary) AS maxs FROM employees
    -> JOIN salaries USING(emp_no)
    -> JOIN titles USING(emp_no)
    -> GROUP BY title ORDER BY maxs DESC;
+----+-------------+-----------+------------+-------+---------------+---------+---------+
| id | select_type | table     | partitions | type  | possible_keys | key     | key_len |
+----+-------------+-----------+------------+-------+---------------+---------+---------+
|  1 | SIMPLE      | employees | NULL       | index | PRIMARY       | PRIMARY | 4 ...
|  1 | SIMPLE      | titles    | NULL       | ref   | PRIMARY,title | PRIMARY | 4 ...
|  1 | SIMPLE      | salaries  | NULL       | ref   | PRIMARY       | PRIMARY | 4 ...
+----+-------------+-----------+------------+-------+---------------+---------+---------+
```

Actual Query

```
mysql> EXPLAIN SELECT first_name, last_name, title, salary FROM employees
    -> JOIN salaries USING(emp_no) JOIN titles USING(emp_no)
    -> WHERE salary IN (SELECT MAX(salary) FROM salaries
    ->                  JOIN titles WHERE titles.to_date > CURDATE());
    ...
5 rows in set, 1 warning (0,01 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
Level: Note
Code: 1003
Message: /* select#1 */ select `employees`.`employees`.`first_name` AS `first_name`,
`employees`.`employees`.`last_name` AS `last_name`,`employees`.`salaries`.`salary` AS `salary`
from `employees`.`employees` join `employees`.`salaries`
where ((`employees`.`salaries`.`emp_no` = `employees`.`employees`.`emp_no`)
and <in_optimizer>(`employees`.`salaries`.`salary`,`employees`.`salaries`.`salary` in
  ( <materialize> (/* select#2 */
    select max(`employees`.`salaries`.`salary`)
      from `employees`.`salaries` join `employees`.`titles`
    where (`employees`.`titles`.`to_date` > <cache>(curdate())) having true ),
     <primary_index_lookup>(`employees`.`salaries`.`salary` in
       <temporary table> on <auto_distinct_key>
     where ((`employees`.`salaries`.`salary` = `<materialized_subquery>`.`MAX(salary)`)))))
1 row in set (0,00 sec)
```

Effect of Indexes: Before

```
mysql> explain select * from t1\G
*************************** 1. row ***************************
...
rows: 12
Extra: NULL

mysql> explain select * from t1 where f2=12\G
*************************** 1. row ***************************
...
key: NULL
...
rows: 12
Extra: Using where
```

Effect of Indexes: After

```
mysql> alter table t1 add index(f2);
Query OK, 12 rows affected (0.07 sec)
Records: 12  Duplicates: 0  Warnings: 0

mysql> explain select * from t1 where f2=12\G
*************************** 1. row ***************************
...
key: f2
key_len: 5
ref: const
rows: 1
Extra: NULL
1 row in set (0.00 sec)
```

Real Numbers

Handler_* Status Variables

- EXPLAIN is optimistic

```
mysql> explain select * from ol
    -> where thread_id=10432 and site_id != 9939
    -> order by id limit 3\G
*************************** 1. row ***************************
id: 1                     |         ref: NULL
select_type: SIMPLE       |        rows: 33
table: ol                 |    filtered: 8.07
partitions: NULL          |       Extra: Using where
type: index
possible_keys: thread_id
key: PRIMARY
key_len: 4
1 row in set, 1 warning (0,00 sec)
```

Handler_* Status Variables

- 'Handler_*' show truth

```
mysql> FLUSH STATUS;
mysql> select * from ol
    -> where thread_id=10432 and site_id != 9939
    -> order by id limit 3;
mysql> show status like 'Handler%';
+----------------------------+--------+
| Variable_name              | Value  |
+----------------------------+--------+
...
| Handler_read_first         | 1      |
| Handler_read_key           | 1      |
| Handler_read_last          | 0      |
| Handler_read_next          | 100000 |
...
```

PROCESSLIST

- `SHOW [FULL] PROCESSLIST`
- `performance_schema.processlist`
- `INFORMATION_SCHEMA.PROCESSLIST`
- `performance_schema.threads`
- Your first alert of performance issues shows all running queries

Execution Stages

- Can be seen in the PROCESSLIST

```
mysql> show processlist\G
*********************** 1. row ***********************
     Id: 7
   User: root
   Host: localhost:48799
     db: employees
Command: Query
   Time: 2
  State: Sending data
   Info: select count(*) from employees
         join titles using(emp_no)
         where title='Senior Engineer'
...
```

  - Very useful when you need to answer on the question: "What is my server doing now?"

Execution Stages

- PERFORMANCE_SCHEMA.EVENTS_STAGES_*

```
mysql> select eshl.event_name, substr(sql_text, 1, 15) ‘sql‘,
    -> eshl.timer_wait/1000000000000 w_s
    -> from  events_stages_history_long
    -> eshl join  events_statements_history_long esthl on
    -> (eshl.nesting_event_id = esthl.event_id) where
    -> esthl.current_schema=’employees’ and sql_text like
    -> ’select count(*) from employees%’
    -> order by eshl.timer_start asc;
+-------------------------------+-----------------+--------+
| event_name                    | sql             | w_s    |
+-------------------------------+-----------------+--------+
| stage/sql/starting            | select count(*) | 0.0002 |
| stage/sql/checking permissions | select count(*) | 0.0000 |
...
```

Execution Stages

- PERFORMANCE_SCHEMA.EVENTS_STAGES_*

```
...
| stage/sql/checking permissions | select count(*) | 0.0000 |
| stage/sql/Opening tables       | select count(*) | 0.0000 |
| stage/sql/init                 | select count(*) | 0.0001 |
| stage/sql/System lock          | select count(*) | 0.0000 |
| stage/sql/optimizing           | select count(*) | 0.0000 |
| stage/sql/statistics           | select count(*) | 0.0001 |
| stage/sql/preparing            | select count(*) | 0.0000 |
| stage/sql/executing            | select count(*) | 0.0000 |
| stage/sql/Sending data         | select count(*) | 5.4915 |
| stage/sql/end                  | select count(*) | 0.0000 |
...
```

Temporary Tables and Other Operations

- Status variables

```
mysql> flush status;
Query OK, 0 rows affected (0,01 sec)
mysql> select count(*) from employees join titles using(emp_no)
    -> where title='Senior Engineer';
+----------+
| count(*) |
+----------+
|    97750 |
+----------+
1 row in set (5,44 sec)
```

Temporary Tables and Other Operations

- Status variables

```
mysql> select * from performance_schema.session_status
    -> where variable_name in ('Created_tmp_tables',
    -> 'Created_tmp_disk_tables', 'Select_full_join',
    -> 'Select_full_range_join', 'Select_range',
    -> 'Select_range_check', 'Select_scan', 'Sort_merge_passes',
    -> 'Sort_range', 'Sort_rows', 'Sort_scan')
    -> and variable_value > 0;
+------------------------+----------------+
| VARIABLE_NAME          | VARIABLE_VALUE |
+------------------------+----------------+
| Select_scan            | 2              |
+------------------------+----------------+
1 row in set (0,00 sec)
```

Temporary Tables and Other Operations

- PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_*

```
mysql> select * from events_statements_history_long
    -> where sql_text like
    -> 'select count(*) from employees join %'\G
*************************** 1. row ***************************
...
ROWS_SENT: 1                          SELECT_RANGE_CHECK: 0
ROWS_EXAMINED: 541058                 SELECT_SCAN: 1
CREATED_TMP_DISK_TABLES: 0            SORT_MERGE_PASSES: 0
CREATED_TMP_TABLES: 0                 SORT_RANGE: 0
SELECT_FULL_JOIN: 0                   SORT_ROWS: 0
SELECT_FULL_RANGE_JOIN: 0             SORT_SCAN: 0
SELECT_RANGE: 0                       NO_INDEX_USED: 0
```

Temporary Tables and Other Operations

- sys.statement_analysis

```
mysql> select * from statement_analysis where query like
    -> 'SELECT COUNT ( * ) FROM `emplo%` and db='employees'\G
*************************** 1. row ***************************
query: SELECT COUNT ( * ) FROM `emplo ...  `emp_no` ) WHE...
db: employees              max_latency: 5.59 s
full_scan:                 avg_latency: 5.41 s
exec_count: 7              lock_latency: 2.24 ms
err_count: 0               rows_sent: 7
warn_count: 0              rows_sent_avg: 1
total_latency: 37.89 s     rows_examined: 3787406
```

Temporary Tables and Other Operations

- `sys.statement_analysis`

```
rows_examined_avg: 541058
    rows_affected: 0
rows_affected_avg: 0
       tmp_tables: 0
  tmp_disk_tables: 0
      rows_sorted: 0
sort_merge_passes: 0
           digest: 4086bc3dc6510a1d9c8f2fe1f59f0943
       first_seen: 2016-04-14 15:19:19
        last_seen: 2016-04-14 16:13:14
```

How to Affect Query Plans

What has an Effect on Query Optimizer Plans?

- Index statistics
- Histogram statistics
- Optimizer switches
- Bugs in optimizer

Index Statistics

- Collected by storage engine

Index Statistics

- Collected by storage engine
- Used by Optimizer

Index Statistics

- Can be examined by `SHOW INDEX`

```
mysql> show index from sbtest1;
+---------+----------+-------------+-------------+
| Table   | Key_name | Column_name | Cardinality |
+---------+----------+-------------+-------------+
| sbtest1 | k_1      | k           |       49142 |
+---------+----------+-------------+-------------+
mysql> select count(distinct id),
    -> count(distinct k) from sbtest1;
+--------------------+-------------------+
| count(distinct id) | count(distinct k) |
+--------------------+-------------------+
|             100000 |             17598 |
+--------------------+-------------------+
```

Index Statistics

- Can be updated
  - `ANALYZE TABLE`
  - If does not help: rebuild the table
    - `OPTIMIZE TABLE`
    - `ALTER TABLE ENGINE=INNODB; ANALYZE TABLE`

Histogram Statistics

- Since version 8.0
- Collected and used by the Optimizer
- Visible in Information Schema

```
mysql> select HISTOGRAM from information_schema.column_statistics
    -> where table_name='example'\G
*************************** 1. row ***************************
HISTOGRAM: {"buckets": [[1, 0.6], [2, 0.8], [3, 1.0]],
"data-type": "int", "null-values": 0.0, "collation-id": 8,
"last-updated": "2018-11-07 09:07:19.791470",
"sampling-rate": 1.0, "histogram-type": "singleton",
"number-of-buckets-specified": 3}
1 row in set (0.00 sec)
```

More details

Optimizer Switches

```
mysql> select @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,index_merge_sort_union=on,
index_merge_intersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,
mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,
materialization=on,semijoin=on,loosescan=on,firstmatch=on,
duplicateweedout=on,subquery_materialization_cost_based=on,
use_index_extensions=on,condition_fanout_filter=on,derived_merge=on
1 row in set (0,00 sec)
```

Optimizer Switches

- Turn `ON` and `OFF` particular optimization
- Work with them as with any other option
  - Turn `OFF` and try

```
SET optimizer_switch = 'use_index_extensions=off';
SELECT ...
EXPLAIN SELECT ...
```

  - If helps implement in queries

```
SELECT /*+ SEMIJOIN(FIRSTMATCH, LOOSESCAN) */ * FROM t1 ...;
SELECT /*+ BKA(t1) NO_BKA(t2) */ * FROM t1 INNER JOIN t2 WHERE ...;
```

Bugs in Optimizer

- Optimizer choses wrong plan for no reason
- Statistics are up to date
- Histograms are not usable
- Solution
  - Use index hints
    - `FORCE INDEX`
    - `IGNORE INDEX`
- On every upgrade
  - Remove index hints
  - Test if the query improved
  - **You must do it even for minor version upgrades!**

Summary

- `EXPLAIN` is essential for query tuning
- Real job is done by storage engine
- Index statistics affect query execution plan
- All index hints, optimizer hints and other workarounds must be validated on each upgrade

More Information

EXPLAIN Syntax

EXPLAIN FORMAT=JSON is Cool!

Troubleshooting add-ons

Optimizer Statistics aka Histograms

Optimizer Hints

Tracing the Optimizer

Thank you!

www.slideshare.net/SvetaSmirnova

twitter.com/svetsmirnova

github.com/svetasmirnova