Troubleshooting Slow Queries

Sveta Smirnova
Principal Support Engineer
April, 28, 2016

Table of Contents

Before we start

When you see slow query first

- You develop an application and find out that some queries are running slow
- After a while you find some slow queries in the slow query log
- All such queries are always slow
- We would not discuss cases when concurrency affects performance
  - This is subject of the next webinar

Slow is relative

- Mind you data!
- 75,000,000 rows
  - (INT, INT)
    - 75,000,000 * (4 + 4) = 600,000,000 bytes = 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

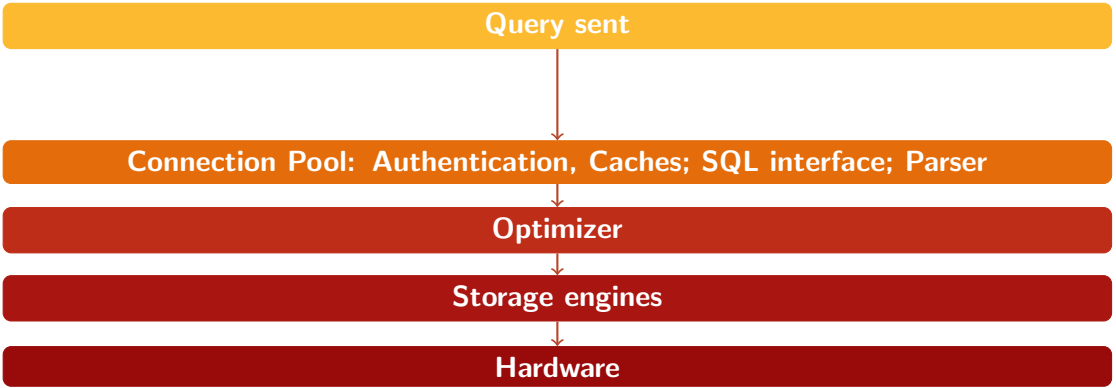
- Mind you data!
- Mind use case
  - Popular website
  - Admin interface
  - Weekly cron job

Slow is relative

- Mind you data!
- Mind use case
- Mind location
  - Server, used by multiple connections
  - Dedicated for OLAP queries

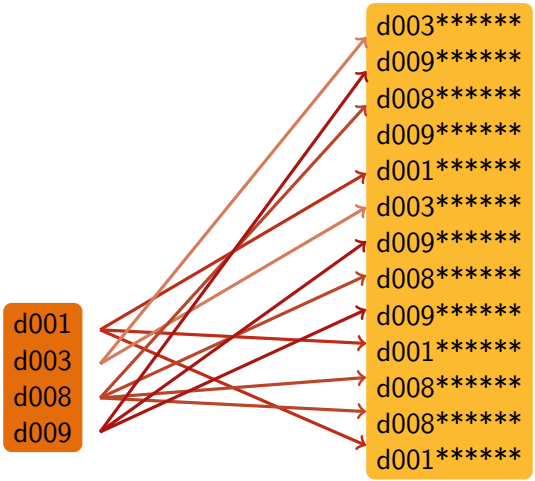
What affects query execution

Query execution workflow

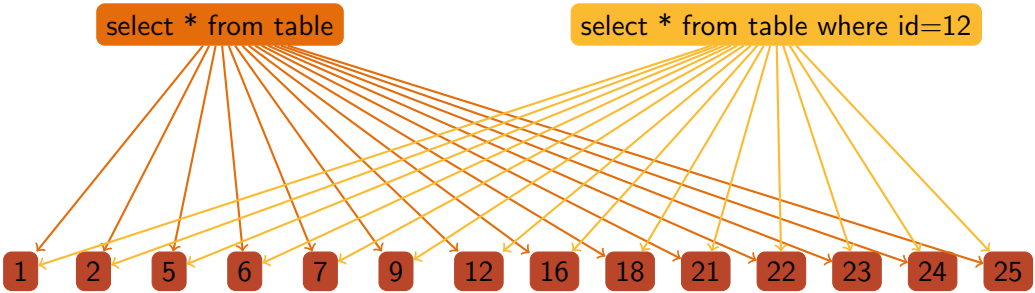


MySQL Indexes

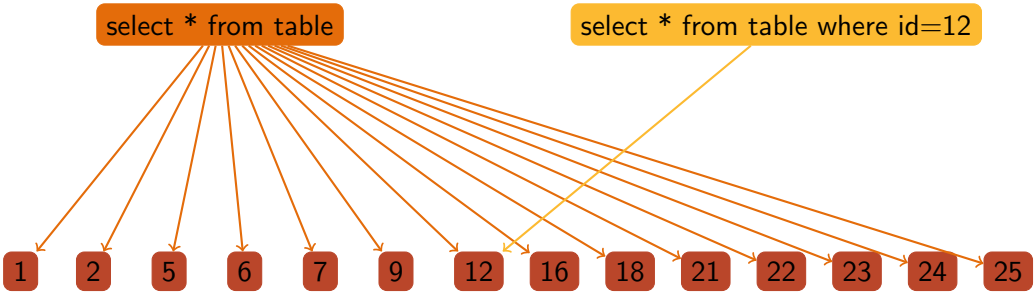
- B-Tree (Mostly)
- Fractal Tree
- R-Tree (Spatial)
- Hash (Memory SE)
- Engine-dependent



No index access



After index added



EXPLAIN: how to find out how optimizer works

MySQL EXPLAIN

- Estimates what happens during query execution
- Displays plan which Optimizer choses
- Provides information how rows should be accessed

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
```

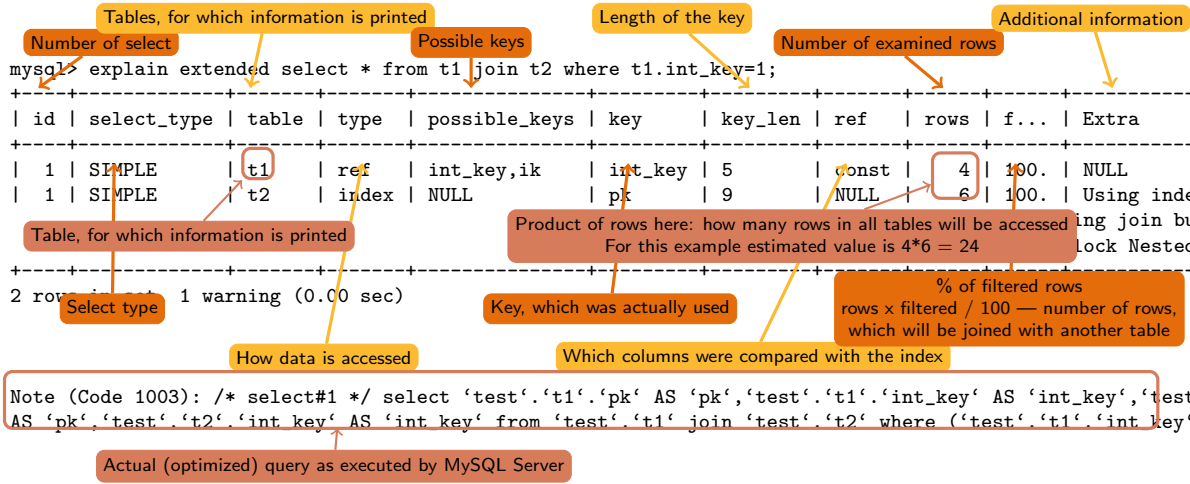
Same number of examined rows for both queries

Effect of indexes: after

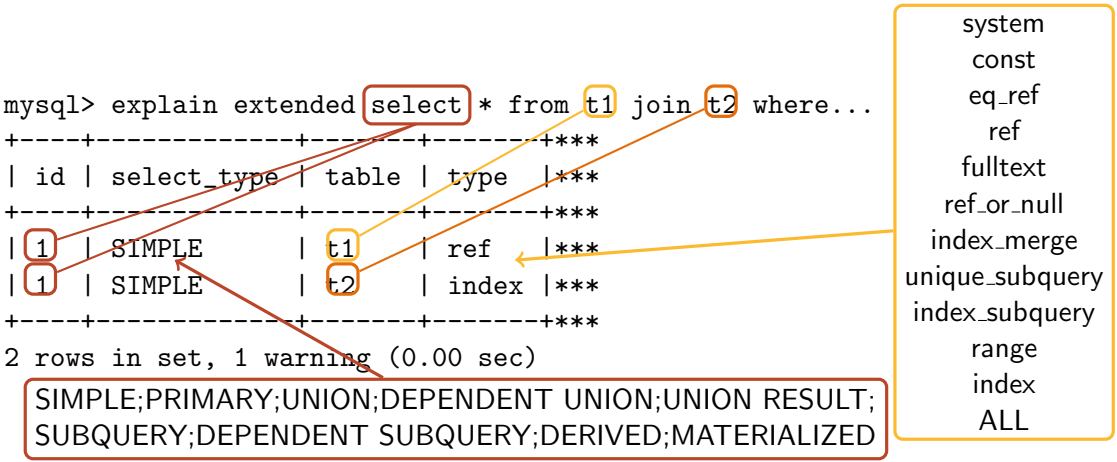
```
mysql> alter table t1 add index(f2);
Query OK, 12 rows affected (0.07 sec)
Records: 12  Duplicates: 0  Warnings: 0

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)
```

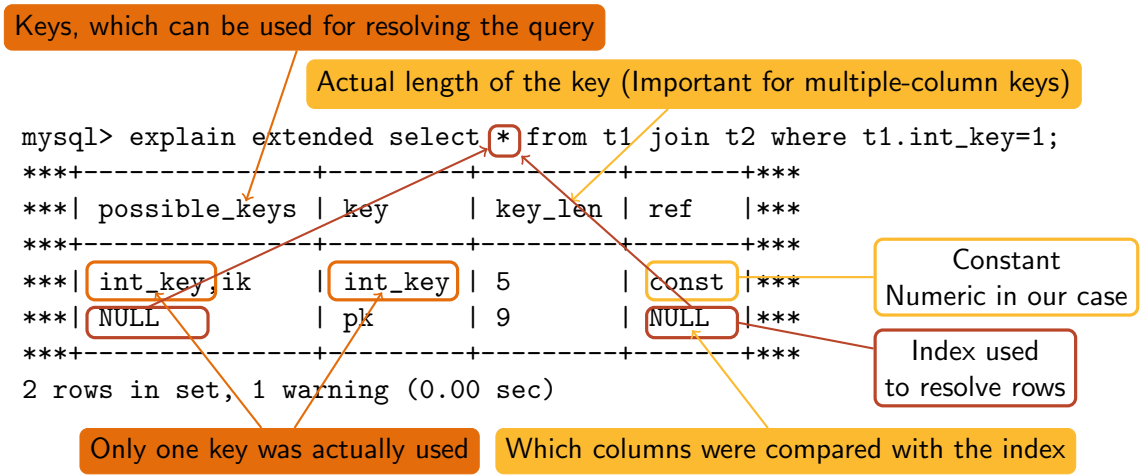
EXPLAIN: overview



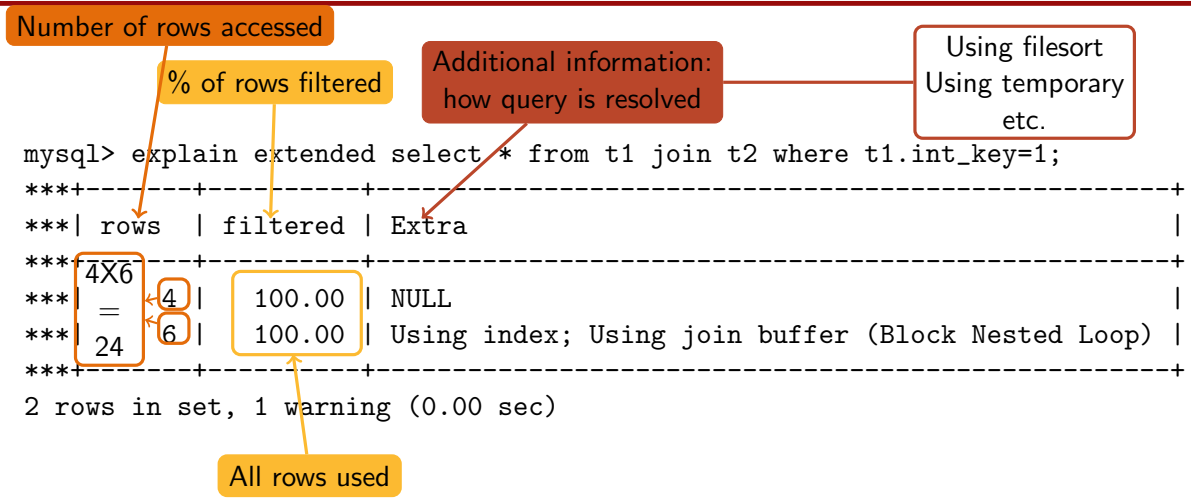
EXPLAIN in details



EXPLAIN in details: keys



EXPLAIN in details: rows



EXPLAIN type by example: ALL

```
mysql> explain select count(*) from employees where hire_date > '1995-01-01'\G
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: employees
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 300157
        Extra: Using where
1 row in set (0.00 sec)
```

All rows in the table examined
Worst plan ever!

EXPLAIN type by example: index

```
mysql> explain select count(*) from titles where title='Senior Engineer'\G
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: titles
         type: index
possible_keys: NULL
          key: emp_no
      key_len: 4
          ref: NULL
         rows: 444033
        Extra: Using where; Using index
1 row in set (0.11 sec)
```

No row in the table was accessed to resolve the query!
Only index used
Still all records in the index were scanned

EXPLAIN type by example: range

- We need to add index to table employees first
- ```
  mysql> alter table employees add index(hire_date);
  Query OK, 0 rows affected (3.48 sec)
  Records: 0 Duplicates: 0 Warnings: 0
  ```
  ```

EXPLAIN type by example: range

```
mysql> explain select count(*) from employees where hire_date>'1995-01-01'\G
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: employees
         type: range
possible_keys: hire_date
          key: hire_date
      key_len: 3
          ref: NULL
         rows: 68654
        Extra: Using where; Using index
1 row in set (0.00 sec)
```

Only rows from given range used

Compare with ALL:
300157/68654 = 4.3720
4 times less rows examined!

EXPLAIN type by example: index_subquery

```
mysql> explain select count(*), title from titles where from_date in
    -> (select hire_date from employees where hire_date > '1995-01-01')
*********************** 1. row ***********************
           id: 1
  select_type: PRIMARY
        table: titles
         type: index
possible_keys: NULL
          key: emp_no
      key_len: 4
          ref: NULL
         rows: 443951
        Extra: Using where; Using index; Using temporary; Using filesort
```

EXPLAIN type by example: index_subquery

```
********************** 2. row **********************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: employees
         type: index_subquery
possible_keys: hire_date
          key: hire_date
      key_len: 3
          ref: func
         rows: 37
        Extra: Using index; Using where
2 rows in set (0.00 sec)
```

EXPLAIN type by example: unique_subquery

```
mysql> explain select count(*), title from titles where emp_no in (select
    -> emp_no from employees where hire_date > '1995-01-01') group by title\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: titles
         type: index
possible_keys: NULL
          key: emp_no
      key_len: 4
          ref: NULL
         rows: 443951
        Extra: Using where; Using index; Using temporary; Using filesort
```

EXPLAIN type by example: unique_subquery

```
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: employees
         type: unique_subquery
possible_keys: PRIMARY,hire_date
          key: PRIMARY
      key_len: 4
          ref: func
         rows: 1
        Extra: Using where
2 rows in set (0.00 sec)
```

EXPLAIN type by example: index_merge

- We need to modify table to run this test

```
mysql> create table dept_emp_copy (emp_no int, dept_no int);
Query OK, 0 rows affected (0.13 sec)

mysql> alter table dept_emp_copy add key(dept_no);
Query OK, 0 rows affected (1.32 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> insert into dept_emp_copy(emp_no)
    -> select distinct emp_no from dept_emp;
Query OK, 300024 rows affected (4.63 sec)
Records: 300024 Duplicates: 0 Warnings: 0
```

EXPLAIN type by example: index_merge

```
mysql> alter table dept_emp_copy add primary key(emp_no);
Query OK, 300024 rows affected (5.48 sec)
Records: 300024 Duplicates: 0 Warnings: 0

mysql> update dept_emp_copy, dept_emp
    -> set dept_emp_copy.dept_no=dept_emp.dept_no
    -> where dept_emp_copy.emp_no=dept_emp.emp_no;
Query OK, 300024 rows affected, 65535 warnings (15.66 sec)
Rows matched: 300024 Changed: 300024 Warnings: 0
```

EXPLAIN type by example: index_merge

```
mysql> explain select * from dept_emp_copy where
    -> dept_no > 5 or (emp_no > 10000 and emp_no < 20000)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: dept_emp_copy
         type: index_merge
possible_keys: PRIMARY,dept_no
          key: dept_no,PRIMARY
      key_len: 5,4
          ref: NULL
         rows: 21103
        Extra: Using sort_union(dept_no,PRIMARY); Using where
```

EXPLAIN type by example: original table

```
mysql> explain select * from dept_emp where dept_no in ('d005', 'd006',
    -> 'd007', 'd008', 'd009') or (emp_no > 10000 and emp_no < 20000)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: dept_emp
         type: ALL
possible_keys: PRIMARY,emp_no,dept_no
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 332289
        Extra: Using where
```

EXPLAIN type by example: ref

```
mysql> explain select * from dept_emp where dept_no = 'd005'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: dept_emp
         type: ref
possible_keys: dept_no
          key: dept_no
      key_len: 4
          ref: const
         rows: 145708
        Extra: Using where
1 row in set (0.00 sec)
```

EXPLAIN type by example: eq_ref

```
mysql> explain select * from dept_manager
    -> join employees using(emp_no) limit 10\G
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: dept_manager
         type: ALL
possible_keys: PRIMARY,emp_no
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 24
        Extra:
```

EXPLAIN type by example: eq_ref

```
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.dept_manager.emp_no
         rows: 1
        Extra:
2 rows in set (0.00 sec)
```

EXPLAIN type by example: const

```
mysql> explain select * from departments where dept_no='d005'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: departments
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
        Extra:
1 row in set (0.00 sec)
```

EXPLAIN PARTITIONS

```
mysql> explain partitions select count(*)
    -> from employees_part where hire_date > '1991-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees_part
   partitions: p1,p2
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 7
          ref: NULL
         rows: 135214
```

EXPLAIN EXTENDED

```
mysql> explain extended select count(*) from employees join titles
    -> using(emp_no) where title='Senior Engineer'\G
*************************** 1. row ***************************
...
2 rows in set, 1 warning (0.00 sec)
mysql> show warnings\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select count(0) AS `count(*)` from `employees`.`employees` join
`employees`.`titles` where ((`employees`.`titles`.`emp_no` = `employees`
.`employees`.`emp_no`) and (`employees`.`titles`.`title` = 'Senior Engineer'))
1 row in set (0.01 sec)
```

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
  - Cost statistics

```
"duplicates_removal": {
...
"cost_info": {
          "read_cost": "1252.00",
          "eval_cost": "88544.80",
          "prefix_cost": "89796.80",
          "data_read_per_join": "27M"
        },
```

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
  - Cost statistics
  - Which part of index chosen

```
mysql> explain format=json SELECT first_name, last_name FROM employees
    -> WHERE first_name='Steve' and last_name like 'V%'
    -> and hire_date > '1990-01-01'\G
*************************** 1. row ***************************
EXPLAIN: {
...
"used_key_parts": [
        "first_name",
        "last_name"
      ],
```

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
  - Cost statistics
  - Which part of index chosen
  - Columns, used to resolve query

```
mysql> explain format=json select count(*) from Country
    -> where Continent='Africa' and Population > 1000000\G
*************************** 1. row ***************************
...
      "used_columns": [
        "Continent",
        "Population"
      ],
...
```

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
- Better structured view
  - Clear distinction for which of operations particular optimization used

```
mysql> explain format=json select distinct last_name
    -> from employees order by last_name asc\G
  ...
    "ordering_operation": {
      "using_filesort": false, - No temporary table here!
      "duplicates_removal": {
        "using_temporary_table": true,
        "using_filesort": true,
```

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
- Better structured view
  - Clear distinction for which of operations particular optimization used
  - Easier to find out ”which table belongs to which subselect” for complicated queries

EXPLAIN FORMAT = JSON

- More information than in regular EXPLAIN
- Better structured view
  - Clear distinction for which of operations particular optimization used
  - Easier to find out ”which table belongs to which subselect” for complicated queries
  - Separate member for each kind of optimization: grouping, ordering, duplicates_removal, etc.

Data matters: again

EXPLAIN: All, Index

- ”All” always used for queries like SELECT * FROM table; without WHERE condition
- ”Index” always used for queries like SELECT indexed_field FROM table; without WHERE condition
- Think carefully if you need to improve such queries, especially in cases when this is small table, joined with bigger one

EXPLAIN: index_merge, ref_or_null, ref

- index_merge does not work with strings
- ref_or_null cannot become ref if you need NULL values
- ref cannot become eq_ref if you need not unique values

Queries limitations

- Queries which use LIKE '%foo%' cannot use indexes
- Function usage can prevent Optimizer to choose index access
  - Internal functions: with exceptions
  - Stored functions: always
  - UDF functions: always

What really happened

Inside optimizer

Optimizer trace

- INFORMATION SCHEMA.OPTIMIZER_TRACE
- Shows what happens inside Optimizer during query processing
- Typical use case

```
SET optimizer_trace="enabled=on";
SELECT ...;
SELECT * FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
SET optimizer_trace="enabled=off";
```

Optimizer trace example

- The table

```
mysql> show create table titles\G
*************************** 1. row ***************************
       Table: titles
Create Table: CREATE TABLE `titles` (
  `emp_no` int(11) NOT NULL,
  `title` varchar(50) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date DEFAULT NULL,
  PRIMARY KEY (`emp_no`,`title`,`from_date`),
  KEY `emp_no` (`emp_no`),
  CONSTRAINT `titles_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES...
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Optimizer trace example

- The query

```
mysql> select distinct title from titles where year(from_date) > '1990';
+--------------------+
| title              |
+--------------------+
| Staff              |
| Senior Engineer    |
...
```

Optimizer trace example

- Possible and chosen keys

```
mysql> explain select distinct title from titles
    -> where year(from_date) > '1990'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: titles
   partitions: NULL
         type: index
possible_keys: PRIMARY,emp_no
          key: emp_no
      key_len: 4
          ...
```

Optimizer trace example

- EXPLAIN FORMAT=JSON sheds some light
  - Query cost for key emp_no:

```
mysql> explain format=json select distinct title from titles
    -> where year(from_date) > '1990'\G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "89630.20"
    },
```

Optimizer trace example

- EXPLAIN FORMAT=JSON sheds some light
  - Query cost for PRIMARY KEY:

```
mysql> explain format=json select distinct title from titles
    -> force index(primary) where year(from_date) > '1990'\G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "530270.20"
    },
```

Optimizer trace example

- Difference is (530270.20/89630.20) - 6 times, but why?

Optimizer trace example

- Lets check optimizer trace
  - join_optimization. rows_estimation. potential_range_indexes

```
{
    "index": "PRIMARY",
    "usable": true,
...
},
{
    "index": "emp_no",
    "usable": true,
...
```

Optimizer trace example

- Lets check optimizer trace
  - best_covering_index_scan

```
"best_covering_index_scan": {
  "index": "emp_no",
  "cost": 91752,
  "chosen": false,
  "cause": "cost"
},
```

Optimizer trace example

- Lets check optimizer trace
  - Chosen?

```
"group_index_range": {
  "distinct_query": true,
  "potential_group_range_indexes": [
    {
      "index": "PRIMARY",
      "covering": true,
      "usable": false,
      "cause": "select_attribute_not_prefix_in_index"
    },
```

Optimizer trace example

- Lets check optimizer trace
  - Chosen?

```
{
  "index": "emp_no",
  "covering": true,
  "usable": false,
  "cause": "select_attribute_not_prefix_in_index"
}
```

Optimizer trace example

- Lets check optimizer trace
  - Both indexes are not usable!

```
"best_access_path": {
  "considered_access_paths": [
    {
      "rows_to_scan": 441891,
      "access_type": "scan",
      "resulting_rows": 441891,
      "cost": 89630,
      "chosen": true
    }
  ]
},
```

Optimizer trace example

- Lets check optimizer trace
  - emp_no not used to resolve rows:

```
"best_covering_index_scan": {
  "index": "emp_no",
  "cost": 91752,
  "chosen": false,
  "cause": "cost"
},
```

Optimizer trace example

- Lets check optimizer trace
  - So why we see emp_no in EXPLAIN output?

```
"reconsidering_access_paths_for_index_ordering": {
  "clause": "GROUP BY",
  "index_order_summary": {
    "table": "`titles`",
    "index_provides_order": false,
    "order_direction": "undefined",
    "index": "emp_no",
    "plan_changed": false
  }
```

ANALYZE in MariaDB

- Sometimes EXPLAIN lies

```
MariaDB [test]> 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: 31
        table: ol                |    Extra: Using where
         type: index
possible_keys: thread_id
          key: PRIMARY
      key_len: 4
1 row in set (0.00 sec)
```

ANALYZE in MariaDB

- Sometimes EXPLAIN lies
- ANALYZE will show real numbers

```
MariaDB [test]> analyze select * from ol where thread_id=10432
            -> and site_id != 9939 order by id limit 3\G
*************************** 1. row ***************************
           id: 1           |          ref: const
  select_type: SIMPLE      |         rows: 93283
        table: ol          |       r_rows: 100000.00
         type: index       |     filtered: 9.61
possible_keys: thread_id   |  r_filtered: 0.00
          key: PRIMARY     |        Extra: Using where
      key_len: 4
1 row in set (0.03 sec)
```

Inside storage engine

Handler_* status variables

- Same case as in previous example

```
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

- Same case as in previous example
- Status variables 'Handler_*' will show truth

```
mysql> flush status; 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 |
```

Inside the server

PROCESSLIST

- SHOW [FULL] PROCESSLIST
- INFORMATION_SCHEMA.PROCESSLIST
- Your first alert in case of performance issue
- Shows all queries, run at the current moment

Execution stages

- Can be seen in 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'
...
```

Execution stages

- Can be seen in PROCESSLIST
  - Very useful when you need to answer on question: ”What is my server doing now?”

Execution stages

- PERFORMANCE_SCHEMA.EVENTS_STAGES_*

```
mysql> select eshl.event_name, substr(sql_text, 1, 15) as '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 job

- 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 job

- 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 job

- Status variables
- PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_*

```
mysql> select * from performance_schema.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 job

- Status variables
- PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_*
- 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 job

- Status variables
- PERFORMANCE_SCHEMA.EVENTS_STATEMENTS_*
- 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 plans

What affects optimizer plans?

- Index statistics
- Optimizer switches
- Bugs in optimizer

Index statistics

- Collected by storage engine
- Optimizer decides which index to choose based on that

Index statistics

- Can be viewed by SHOW INDEX command

```
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 table
    - OPTIMIZE TABLE
    - ALTER TABLE ENGINE=INNODB; ANALYZE TABLE

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
- Can be not helpful
  - Especially for queries, tuned for previous versions
- 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 index for no reason
- Statistics is up to date
- Solution
  - Use index hints
    - FORCE INDEX
    - IGNORE INDEX
- On every upgrade
  - Remove index hints
  - Test if query improved
  - You must do it even for minor version upgrades!

Summary

Summary

- Understanding EXPLAIN output is essential for query tuning
- But 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! series
- Troubleshooting Performance add-ons
- Optimizer Hints
- Custom Hint Plugin

Place for your questions

???

Thank you!

http://www.slideshare.net/SvetaSmirnova

https://twitter.com/svetsmirnova