How to Analyze and Tune MySQL Queries for Better Performance

Øystein Grøvlen
Senior Principal Software Engineer
MySQL Optimizer Team, Oracle
April 16, 2015

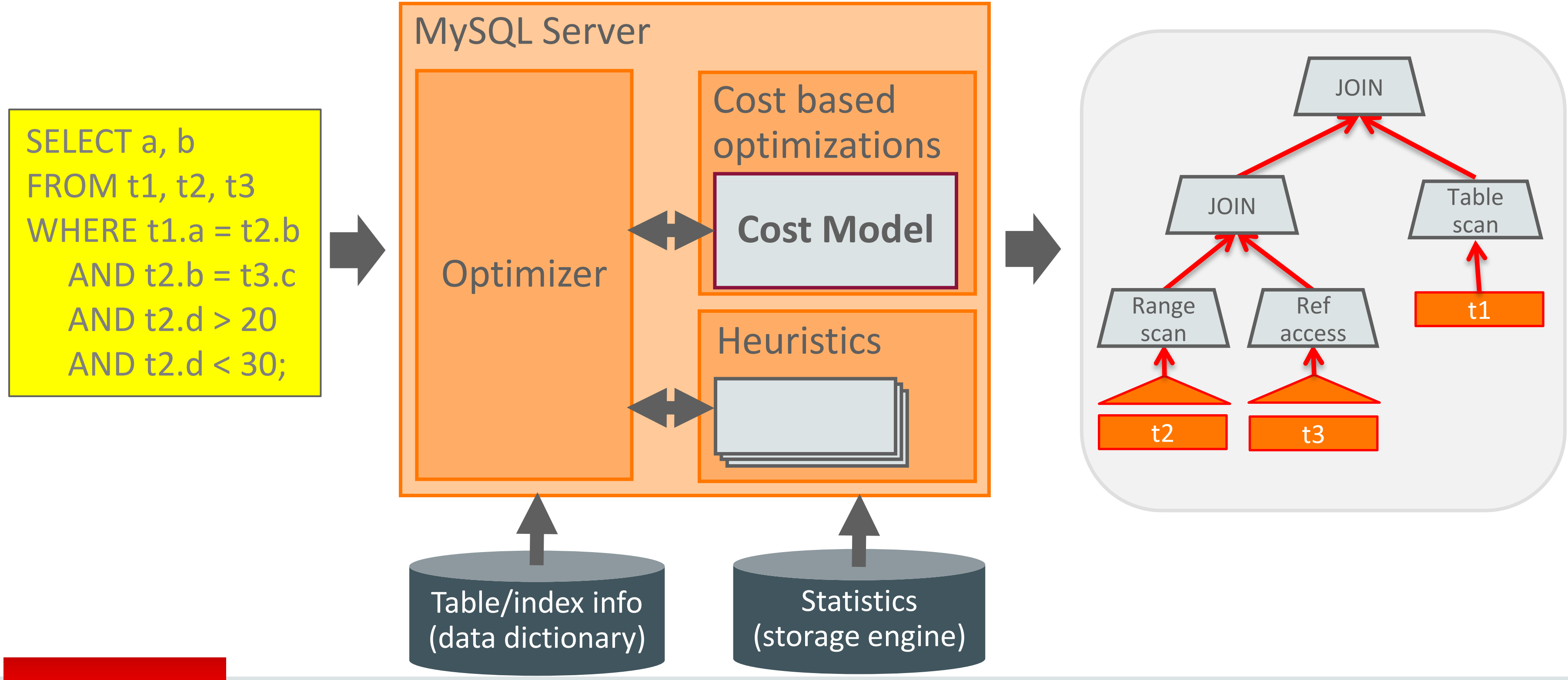
Program Agenda

1. Introduction to MySQL cost-based optimizer
2. Selecting data access method
3. Join optimizer
4. Sorting
5. Tools for monitoring, analyzing, and tuning queries
6. Influencing the optimizer

Program Agenda

1. Introduction to MySQL optimizer
2. Selecting data access method
3. Join optimizer
4. Sorting
5. Tools for monitoring, analyzing, and tuning queries
6. Influencing the optimizer

MySQL Optimizer

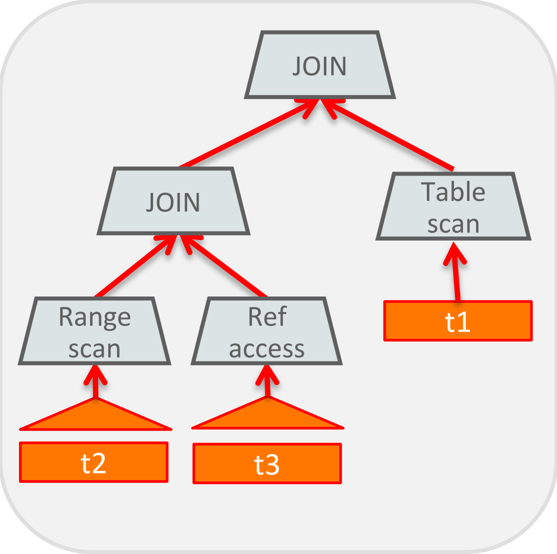


```
SELECT a, b
FROM t1, t2, t3
WHERE t1.a = t2.b
    AND t2.b = t3.c
    AND t2.d > 20
    AND t2.d < 30;
```

Cost-based Query Optimization

General idea

- Assign cost to operations
- Compute cost of partial or alternative plans
- Search for plan with lowest cost



- Cost-based optimizations:

Access method

Join order

Subquery strategy

Input to Cost Model

- **IO-cost:**
  - Estimates from storage engine based on number of pages to read
  - Both index and data pages
- **Schema:**
  - Length of records and keys
  - Uniqueness for indexes
  - Nullability

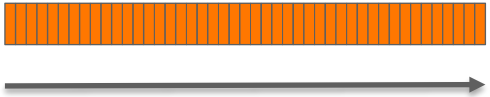
- **Statistics:**
  - Number of rows in table
  - Key distribution/Cardinality:
    - Average number of records per key value
    - Only for indexed columns
    - Maintained by storage engine
  - Number of records in an index range

Cost Model Example

SELECT SUM(o_totalprice) FROM orders
WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-12-31';

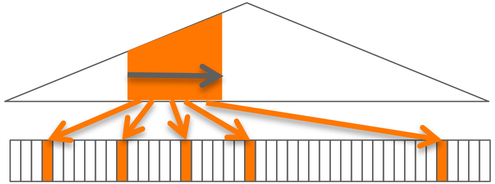
Table scan:

- IO-cost: #pages in table
- CPU cost: #rows * ROW_EVALUATE_COST



Range scan (on secondary index):

- IO-cost: #pages to read from index + #rows_in_range
- CPU cost: #rows_in_range * ROW_EVALUATE_COST



Cost Model

Example

```
EXPLAIN SELECT SUM(o_totalprice) FROM orders
WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-12-31';
```

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ALL	i_o_orderdate	NULL	NULL	NULL	15000000	Using where

```
EXPLAIN SELECT SUM(o_totalprice) FROM orders
WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-06-30';
```

Id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	range	i_o_orderdate	i_o_orderdate	4	NULL	2235118	Using index condition

Cost Model Example: Optimizer Trace

join_optimization / row_estimation / table : orders / range_analysis

```
"table_scan": {
  "rows": 15000000,
  "cost": 3.12e6
} /* table_scan */,
"potential_range_indices": [
  {
    "index": "PRIMARY",
    "usable": false,
    "cause": "not_applicable“
  },
  {
    "index": "i_o_orderdate",
    "usable": true,
    "key_parts": [ "o_orderDATE", "o_orderkey" ]
  }
] /* potential_range_indices */,
…
```

```
"analyzing_range_alternatives": {
  "range_scan_alternatives": [
    {
      "index": "i_o_orderdate",
      "ranges": [ "1994-01-01 <= o_orderDATE <= 1994-12-31"
      ],
      "index_dives_for_eq_ranges": true,
      "rowid_ordered": false,
      "using_mrr": false,
      "index_only": false,
      "rows": 4489990,
      "cost": 5.39e6,
      "chosen": false,
      "cause": "cost"
    }
  ] /* range_scan_alternatives */,
  …
} /* analyzing_range_alternatives */
```

Cost Model vs Real World

Measured Execution Times

	Data in Memory	Data on Disk	Data on SSD
Table scan	6.8 seconds	36 seconds	15 seconds
Index scan	**5.2 seconds**	2.5 hours	30 minutes

Force Index Range Scan:

```
SELECT SUM(o_totalprice)
FROM orders FORCE INDEX (i_o_orderdate)
WHERE o_orderdate BETWEEN '1994-01-01' AND '1994-12-31‘;
```

Performance Schema

Disk I/O

```
SELECT event_name, count_read, avg_timer_read/1000000000.0 “Avg Read Time (ms)”,
          sum_number_of_bytes_read   “Bytes Read”
  FROM performance_schema.file_summary_by_event_name
  WHERE event_name='wait/io/file/innodb/innodb_data_file';
```

Table Scan

event_name	count_read	Avg Read Time (ms)	Bytes Read
wait/io/file/innodb/innodb_data_file	115769	0.0342	1896759296

Index Range Scan

event_name	count_read	Avg Read Time (ms)	Bytes Read
wait/io/file/innodb/innodb_data_file	2188853	4.2094	35862167552

Program Agenda

1 Introduction to MySQL optimizer

2 Selecting data access method

3 Join optimizer

4 Sorting

5 Tools for monitoring, analyzing, and tuning queries

6 Influencing the optimizer

Selecting Access Method

Finding the optimal method to read data from storage engine

- For each table, find the best access method:
  - Check if the access method is useful
  - Estimate cost of using access method
  - Select the cheapest to be used
- Choice of access method is cost based

Main access methods:

- Table scan
- Index scan
- Ref access
- Range scan
- Index merge
- Loose index scan

Ref Access

Single Table Queries

EXPLAIN SELECT * FROM customer WHERE c_custkey = 570887;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	customer	const	PRIMARY	PRIMARY	4	const	1	NULL

EXPLAIN SELECT * FROM orders WHERE o_orderdate = ‘1992-09-12’;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ref	i_o_orderdate	i_o_orderdate	4	const	6271	NULL

Ref Access

Join Queries

```
EXPLAIN SELECT *
FROM orders JOIN customer ON c_custkey = o_custkey
WHERE o_orderdate = ‘1992-09-12’;
```

Id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_orderdate	4	const	6271	Using where
1	SIMPLE	customer	eq_ref	PRIMARY	PRIMARY	4	dbt3.orders. o_custkey	1	NULL

Ref Access

Join Queries, continued

**EXPLAIN SELECT *
**

```
EXPLAIN SELECT *
FROM orders JOIN customer ON c_custkey = o_custkey
WHERE c_acctbal < -1000;
```

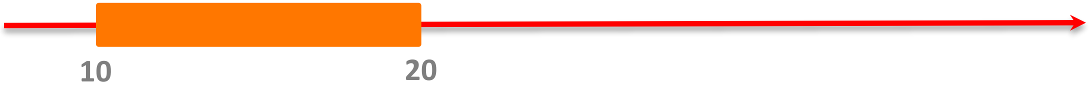
Id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	Using where
1	SIMPLE	orders	ref	i_o_custkey	i_o_custkey	5	dbt3.customer.c_custkey	7	NULL

Range Optimizer

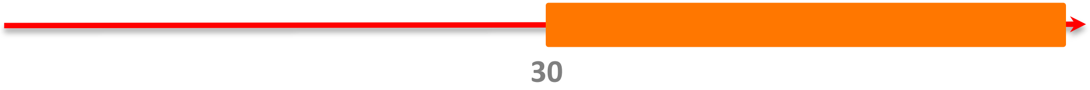
- Goal: find the “minimal” ranges for each index that needs to be read
- Example:

  SELECT * FROM t1 WHERE (key1 > 10 AND key1 < 20) AND key2 > 30

- Range scan using INDEX(key1):



- Range scan using INDEX(key2):



Range Optimizer

Optimizer Trace show ranges

```
SELECT a, b FROM t1
WHERE a > 10
   AND a < 25
   AND a NOT IN (11, 19))
   AND (b < 5 OR b > 10);
```

```
"analyzing_range_alternatives": {
  "range_scan_alternatives": [
    {
      "index": "i_a",
      "ranges": [
        "10 < a < 11",
        "11 < a < 19",
        "19 < a < 25"
      ],
      "index_dives_for_eq_ranges": true,
      "rowid_ordered": false,
      "using_mrr": false,
      "index_only": false,
      "rows": 3,
      "cost": 6.61,
      "chosen": true
    },
    {
      "index": "i_b",
      "ranges": [
        "NULL < b < 5",
        "10 < b"
      ],
      "index_dives_for_eq_ranges": true,
      "rowid_ordered": false,
      ...
```

Range Optimizer: Case Study

Why table scan?

SELECT * FROM orders
WHERE YEAR(o_orderdate) = 1997 AND MONTH(o_orderdate) = 5
AND o_clerk = 'Clerk#000001866';

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ALL	NULL	NULL	NULL	NULL	15000000	Using where

Index not considered

```
mysql> SELECT * FROM orders WHERE year(o_orderdate) = 1997 AND MONTH(…
...
15 rows in set (8.91 sec)
```

Some Reasons Why Index can not be Used

- Indexed column is used as argument to function

  YEAR(o_orderdate) = 1997

- Looking for a suffix:

  name LIKE '%son'

- First column(s) of compound index NOT used

  b = 10 when index defined over **(a, b)**

- Type mismatch

  my_string = 10

- Character set / collation mismatch

  t1 LEFT JOIN t2 ON t1.utf8_string = t2. latin1_string

Range Optimizer: Case Study

Rewrite query to avoid functions on indexed columns

SELECT * FROM orders
WHERE o_orderdate BETWEEN '1997-05-01' AND '1997-05-31'
AND o_clerk = 'Clerk#000001866';

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	range	i_o_orderdate	i_o_orderdate	4	NULL	376352	Using index condition; Using where

```
mysql> SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' AND …
...
15 rows in set (0.91 sec)
```

Range Optimizer: Case Study

Adding another index

CREATE INDEX i_o_clerk ON orders(o_clerk);

SELECT * FROM orders
WHERE o_orderdate BETWEEN '1997-05-01' AND '1997-05-31'
AND o_clerk = 'Clerk#000001866';

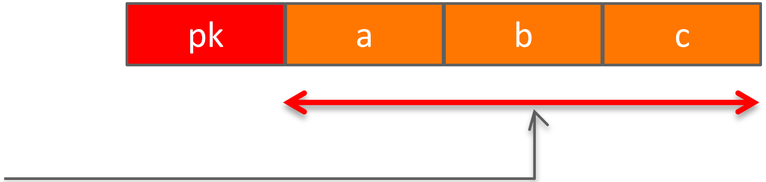
id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	range	i_o_orderdate, i_o_clerk	i_o_clerk	16	NULL	1504	Using index condition; Using where

```
mysql> SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' AND …
...
15 rows in set (0.01 sec)
```

Range Access for Multi-Column Index

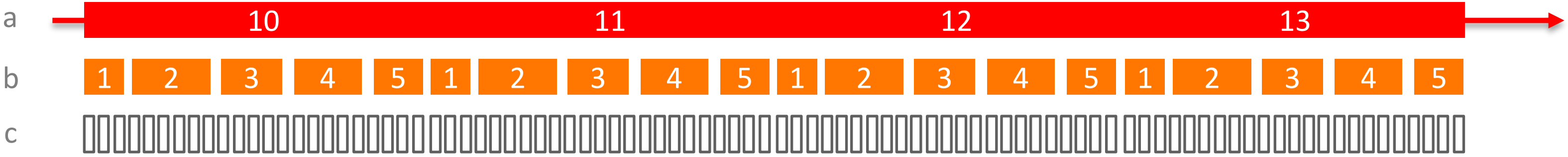
Example table with multi-column index

- Table:



- INDEX idx(a, b, c);

- Logical storage layout of index:



Range Optimizer: Case Study

Create multi-column index

CREATE INDEX i_o_clerk_date ON orders(o_clerk, o_orderdate);

SELECT * FROM orders
WHERE o_orderdate BETWEEN '1997-05-01' AND '1997-05-31'
AND o_clerk = 'Clerk#000001866';

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	range	i_o_orderdate, i_o_clerk, i_o_clerk_date	i_o_clerk_date	20	NULL	14	Using index condition

```
mysql> SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' AND …
...
15 rows in set (0.00 sec)
```

Performance Schema: Query History

UPDATE performance_schema.setup_consumers
SET enabled='YES' WHERE name = 'events_statements_history';

MySQL 5.7:
Enabled by default

```
mysql> SELECT sql_text, (timer_wait)/1000000000.0 “Time(ms)”, rows_examined Rows FROM
  performance_schema.events_statements_history ORDER BY timer_start;
  +---------------------------------------------------------------+----------+------+
  | sql_text                                                      | Time(ms) | Rows |
  +---------------------------------------------------------------+----------+------+
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 8.1690   | 1505 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 7.2120   | 1505 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 8.1613   | 1505 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 7.0535   | 1505 |
  | CREATE INDEX i_o_clerk_date ON orders(o_clerk,o_orderdate) |82036.4190   |    0 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 0.7259   |   15 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 0.5791   |   15 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 0.5423   |   15 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 0.6031   |   15 |
  | SELECT * FROM orders WHERE o_orderdate BETWEEN '1997-05-01' … | 0.2710   |   15 |
  +---------------------------------------------------------------+----------+------+
```

Program Agenda

1 Introduction to MySQL optimizer

2 Selecting data access method

3 Join optimizer

4 Sorting

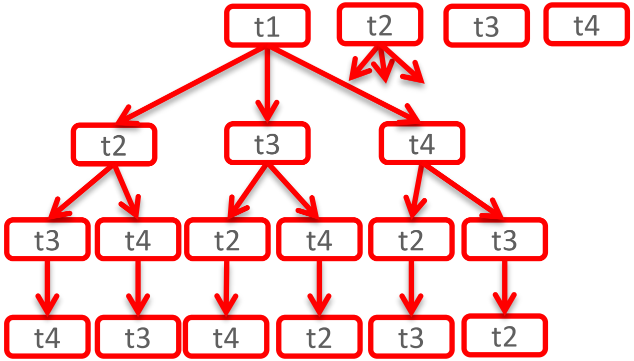
5 Tools for monitoring, analyzing, and tuning queries

6 Influencing the optimizer

Join Optimizer

”Greedy search strategy”

- Goal: Given a JOIN of N tables, find the best JOIN ordering
- Strategy:
  - – Start with all 1-table plans
  - – Expand each plan with remaining tables
    - Depth-first
  - – If “cost of partial plan” > “cost of best plan”:
    - “prune” plan
  - – Heuristic pruning:
    - Prune less promising partial plans
    - May in rare cases miss most optimal plan (turn off with **set optimizer_prune_level = 0**)

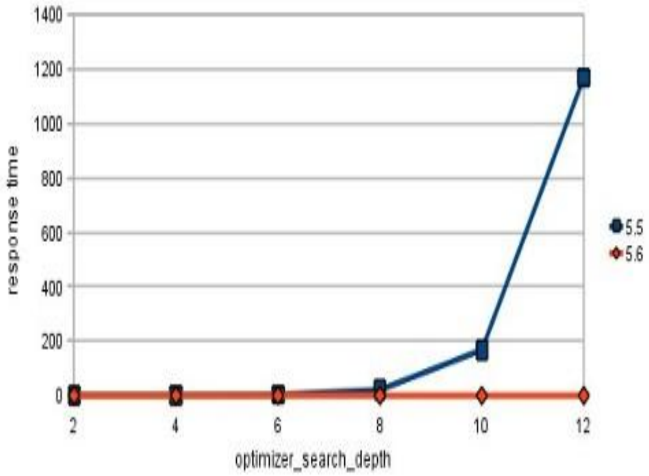


Complexity and Cost of Join Optimizer

Join of N tables: *N!* possible plans to evaluate

Heuristics to reduce the number of plans to evaluate:

- Use **optimizer_search_depth** to limit the number of tables to consider
- Pre-sort tables on *size* and *key dependency* order (Improved in MySQL 5.6)
- When adding the next table to a partial plan, add all tables that it has an equality reference to (New in MySQL 5.6)



Join Optimizer: Case study

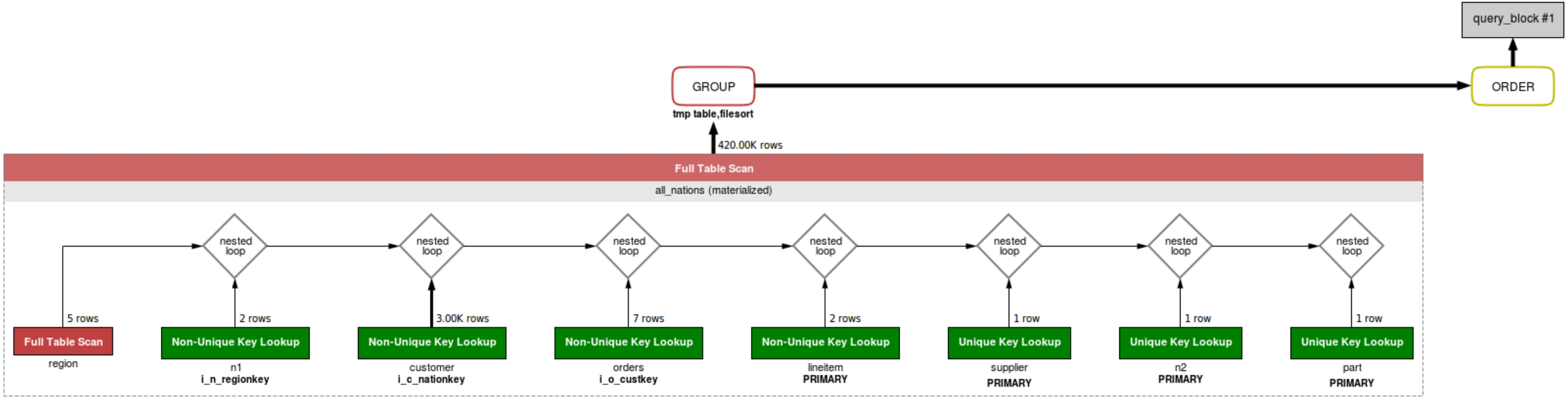
DBT-3 Query 8: National Market Share Query

```
SELECT o_year, SUM(CASE WHEN nation = 'FRANCE' THEN volume ELSE 0 END)  / SUM(volume) AS
        mkt_share

FROM (

        SELECT EXTRACT(YEAR FROM o_orderdate) AS o_year,
                   l_extendedprice * (1 - l_discount) AS volume, n2.n_name AS nation
        FROM part
          JOIN lineitem ON p_partkey = l_partkey
          JOIN supplier ON s_suppkey = l_suppkey
          JOIN orders ON l_orderkey = o_orderkey
          JOIN customer ON o_custkey = c_custkey
          JOIN nation n1 ON c_nationkey = n1.n_nationkey
          JOIN region ON  n1.n_regionkey = r_regionkey
          JOIN nation n2 ON s_nationkey = n2.n_nationkey
        WHERE r_name = 'EUROPE’ AND o_orderdate BETWEEN '1995-01-01' AND '1996-12-31’
            AND p_type = 'PROMO BRUSHED STEEL'

 ) AS all_nations GROUP BY o_year ORDER BY o_year;
```

Join Optimizer: Case Study

MySQL Workbench: Visual EXPLAIN

Execution time: 3 min. 28 sec.



Join Optimizer: Case Study

Force early processing of high selectivity predicates

```
SELECT o_year, SUM(CASE WHEN nation = 'FRANCE' THEN volume ELSE 0 END)  / SUM(volume) AS
        mkt_share
FROM (
        SELECT EXTRACT(YEAR FROM o_orderdate) AS o_year,
            l_extendedprice * (1 - l_discount) AS volume, n2.n_name AS nation
        FROM part
          STRAIGHT_JOIN lineitem ON p_partkey = l_partkey
          JOIN supplier ON s_suppkey = l_suppkey
          JOIN orders ON l_orderkey = o_orderkey
          JOIN customer ON o_custkey = c_custkey
          JOIN nation n1 ON c_nationkey = n1.n_nationkey
          JOIN region ON  n1.n_regionkey = r_regionkey
          JOIN nation n2 ON s_nationkey = n2.n_nationkey
        WHERE r_name = 'EUROPE’ AND o_orderdate BETWEEN '1995-01-01' AND '1996-12-31’
            AND p_type = 'PROMO BRUSHED STEEL'
 ) AS all_nations GROUP BY o_year ORDER BY o_year;
```

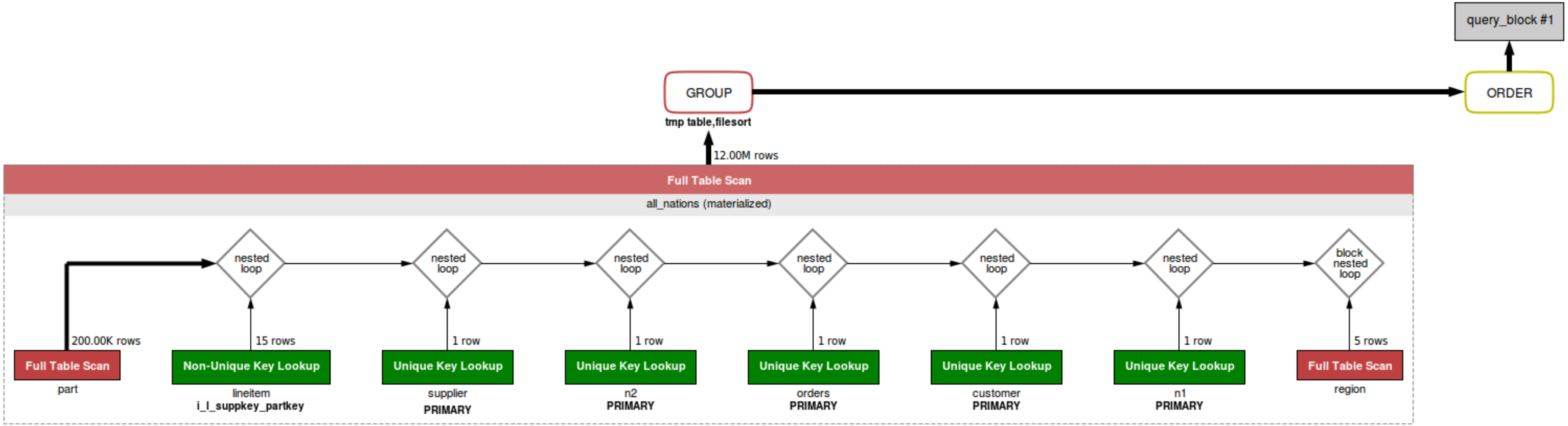
part before lineitem

Highest selectivity

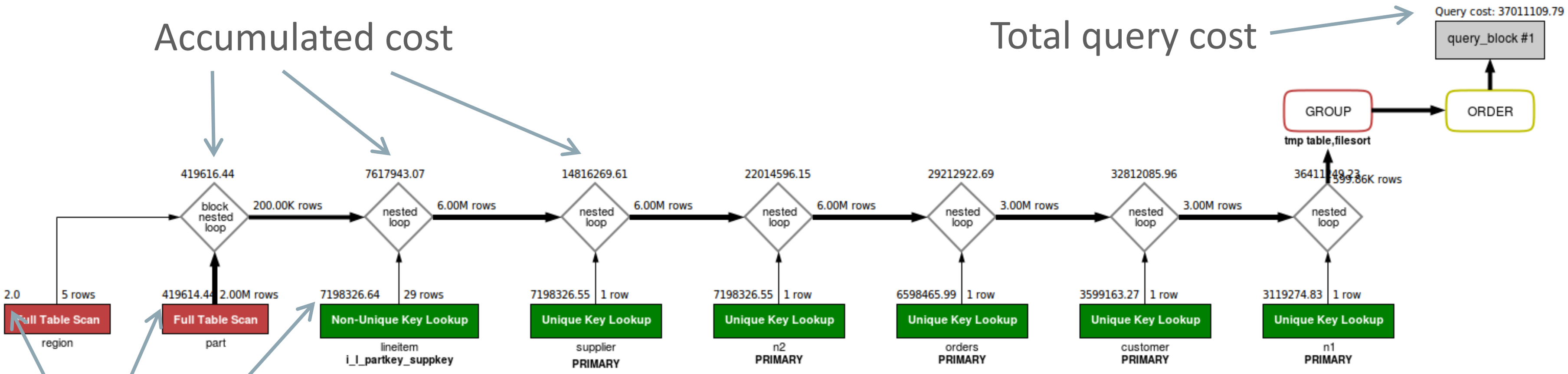
Join Optimizer: Case Study

Improved join order

Execution time: 7 seconds



MySQL 5.7: Cost Information in Structured EXPLAIN



Accumulated cost

Total query cost

Cost per table

Improvements to Query 8 in MySQL 5.7:

- Filtering on non-indexed columns are taken into account
  - No need for hint to force **part** table to be processed early
- Merge derived tables into outer query
  - No temporary table

Program Agenda

1 Introduction to MySQL optimizer

2 Selecting data access method

3 Join optimizer

4 Sorting

5 Tools for monitoring, analyzing, and tuning queries

6 Influencing the optimizer

ORDER BY Optimizations

- General solution; “Filesort”:
  - Store query result in temporary table before sorting
  - If data volume is large, may need to sort in several passes with intermediate storage on disk
- Optimizations:
  - Take advantage of index to generate query result in sorted order
  - For ”LIMIT *n*” queries, maintain priority queue of *n* top items in memory instead of filesort. (New in MySQL 5.6)

Filesort

SELECT * FROM orders ORDER BY o_totalprice ;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ALL	NULL	NULL	NULL	NULL	15000000	Using filesort

SELECT c_name, o_orderkey, o_totalprice
FROM orders JOIN customer ON c_custkey = o_custkey
WHERE c_acctbal < -1000 ORDER BY o_totalprice ;

id	select type	table	type	possible keys	key	key len	ref	rows	extra

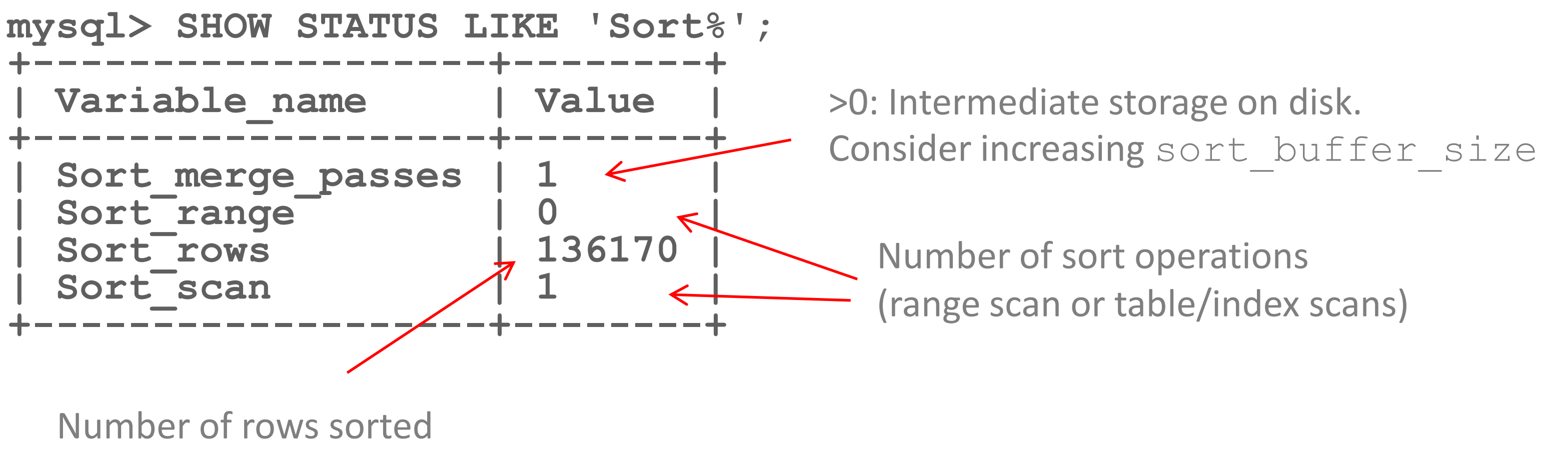
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	Using where;
									Using temporary;
									Using filesort

1	SIMPLE	orders	ref	i_o_custkey	i_o_custkey	5	...	7	NULL

Filesort

Status variables

Status variables related to sorting:



```
mysql> SHOW STATUS LIKE 'Sort%';
+-------------------+--------+
| Variable_name     | Value  |
+-------------------+--------+
| Sort_merge_passes | 1      |
| Sort_range        | 0      |
| Sort_rows         | 136170 |
| Sort_scan         | 1      |
+-------------------+--------+
```

>0: Intermediate storage on disk.
Consider increasing `sort_buffer_size`

Number of sort operations
(range scan or table/index scans)

Number of rows sorted

Filesort

Performance Schema

Sorting status per statement available from Performance Schema

```
mysql> SELECT sql_text,sort_merge_passes,sort_range,sort_rows,sort_scan
       FROM performance_schema.events_statements_history
       ORDER BY timer_start DESC LIMIT 1;

+--------------+-------------------+------------+-----------+-----------+
| sql_text     | sort_merge_passes | sort_range | sort_rows | sort_scan |
+--------------+-------------------+------------+-----------+-----------+
| SELECT ...   |                 1 |          0 |    136170 |         1 |
+--------------+-------------------+------------+-----------+-----------+
```

Filesort: Case Study

```
mysql> FLUSH STATUS;

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT AVG(o_totalprice) FROM (
  SELECT * FROM orders
    ORDER BY o_totalprice DESC
    LIMIT 100000) td;
+-------------------+
| AVG(o_totalprice) |
+-------------------+
| 398185.986158     |
+-------------------+
1 row in set (24.65 sec)
```

Unnecessary large data volume!

```
mysql> SHOW STATUS LIKE 'Sort%';
+-------------------+--------+
| Variable_name     | Value  |
+-------------------+--------+
| Sort_merge_passes |   1432 |
| Sort_range        |      0 |
| Sort_rows         | 100000 |
| Sort_scan         |      1 |
+-------------------+--------+
4 rows in set (0.00 sec)
```

Many intermediate sorting steps!

Filesort: Case Study

Reduce amount of data to be sorted

```
mysql> SELECT AVG(o_totalprice) FROM (SELECT o_totalprice FROM orders ORDER BY
  o_totalprice DESC LIMIT 100000) td;
+-------------------+
| AVG(o_totalprice) |
+-------------------+
| 398185.986158     |
+-------------------+
1 row in set (8.18 sec)
mysql> SELECT sql_text, sort_merge_passes FROM performance_schema.
  events_statements_history ORDER BY timer_start DESC LIMIT 1;

+----------------------------------------------------+-------------------+
| sql_text                                           | sort_merge_passes |
+----------------------------------------------------+-------------------+
| SELECT AVG(o_totalprice) FROM (SELECT o_totalprice |               229 |
+----------------------------------------------------+-------------------+
```

Filesort: Case Study

Default is 256 kB

Increase sort buffer (1 MB)

```
mysql> SET sort_buffer_size = 1024*1024;

mysql> SELECT AVG(o_totalprice) FROM (SELECT o_totalprice FROM orders ORDER BY
  o_totalprice DESC LIMIT 100000) td;
+-------------------+
| AVG(o_totalprice) |
+-------------------+
| 398185.986158     |
+-------------------+
1 row in set (7.24 sec)
mysql> SELECT sql_text, sort_merge_passes FROM performance_schema.
  events_statements_history ORDER BY timer_start DESC LIMIT 1;
+----------------------------------------------------+-------------------+
| sql_text                                           | sort_merge_passes |
+----------------------------------------------------+-------------------+
| SELECT AVG(o_totalprice) FROM (SELECT o_totalprice |                57 |
+----------------------------------------------------+-------------------+
```

Filesort: Case Study

Increase sort buffer even more (8 MB)

```
mysql> SET sort_buffer_size = 8*1024*1024;

mysql> SELECT AVG(o_totalprice) FROM (SELECT o_totalprice FROM orders ORDER BY
  o_totalprice DESC LIMIT 100000) td;
+-------------------+
| AVG(o_totalprice) |
+-------------------+
| 398185.986158     |
+-------------------+
1 row in set (6.30 sec)
mysql> SELECT sql_text, sort_merge_passes FROM performance_schema.
  events_statements_history ORDER BY timer_start DESC LIMIT 1;
+--------------------------------------------------------+-------------------+
| sql_text                                               | sort_merge_passes |
+--------------------------------------------------------+-------------------+
| SELECT AVG(o_totalprice) FROM (SELECT o_totalprice     |                 0 |
+--------------------------------------------------------+-------------------+
```

Use Index to Avoid Sorting

CREATE INDEX i_o_totalprice ON orders(o_totalprice);

SELECT AVG(o_totalprice) FROM
(SELECT o_totalprice FROM orders ORDER BY o_totalprice DESC LIMIT 100000) td;

id	select type	table	Type	possible keys	key	key len	ref	rows	extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	100000	NULL
2	DERIVED	orders	index	NULL	i_o_totalprice	6	NULL	15000000	Using index

```
mysql> SELECT AVG(o_totalprice) FROM (
   SELECT o_totalprice FROM orders
   ORDER BY o_totalprice DESC LIMIT 100000) td;
...
1 row in set (0.06 sec)
```

Program Agenda

1 Introduction to MySQL optimizer

2 Selecting data access method

3 Join optimizer

4 Sorting

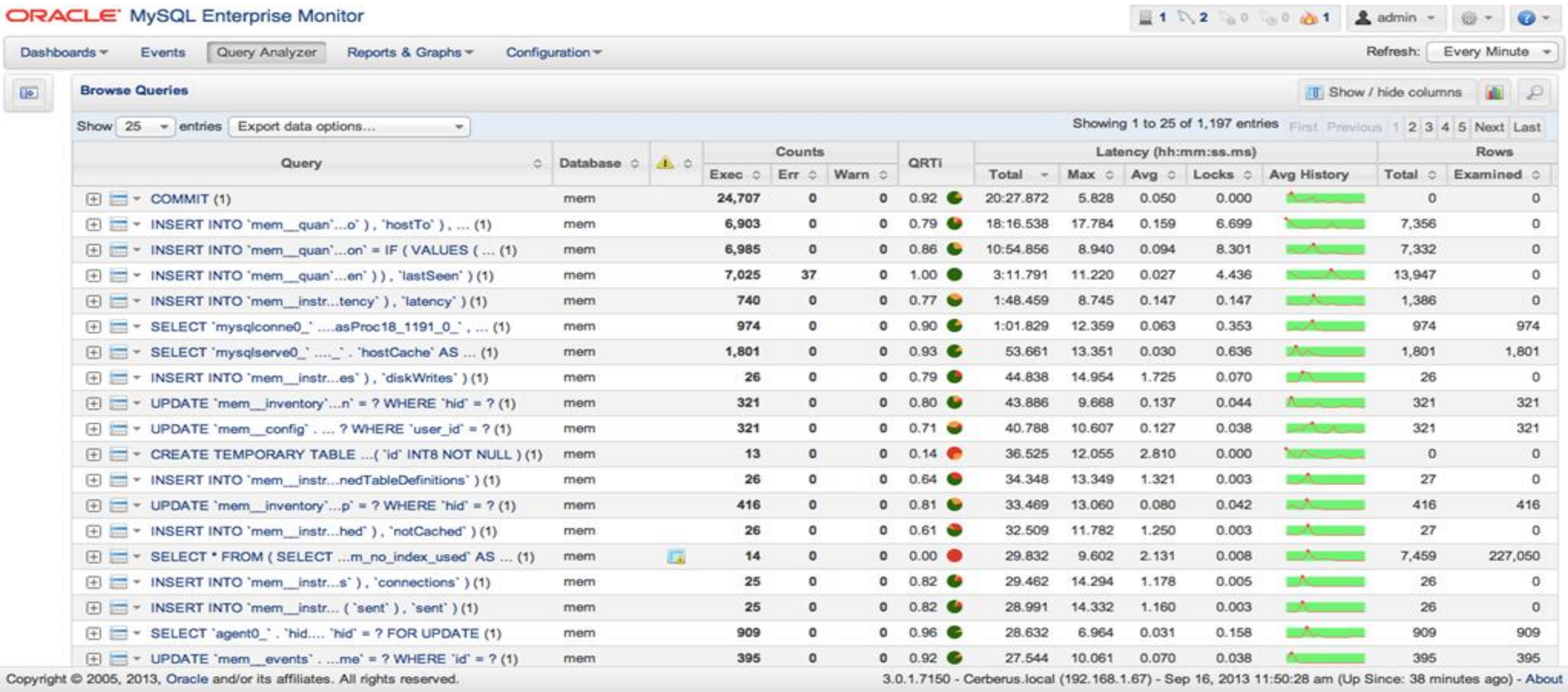
5 Tools for monitoring, analyzing, and tuning queries

6 Influencing the optimizer

Useful tools

- MySQL Enterprise Monitor (MEM), Query Analyzer
  - Commercial product
- Performance schema, MySQL SYS schema
- EXPLAIN
- Structured EXPLAIN (FORMAT=JSON)
- Visual EXPLAIN (MySQL Workbench)
- Optimizer trace
- Slow log
- Status variables (SHOW STATUS LIKE ’Sort%’)

MySQL Enterprise Monitor, Query Analyzer



Refresh: Every Minute

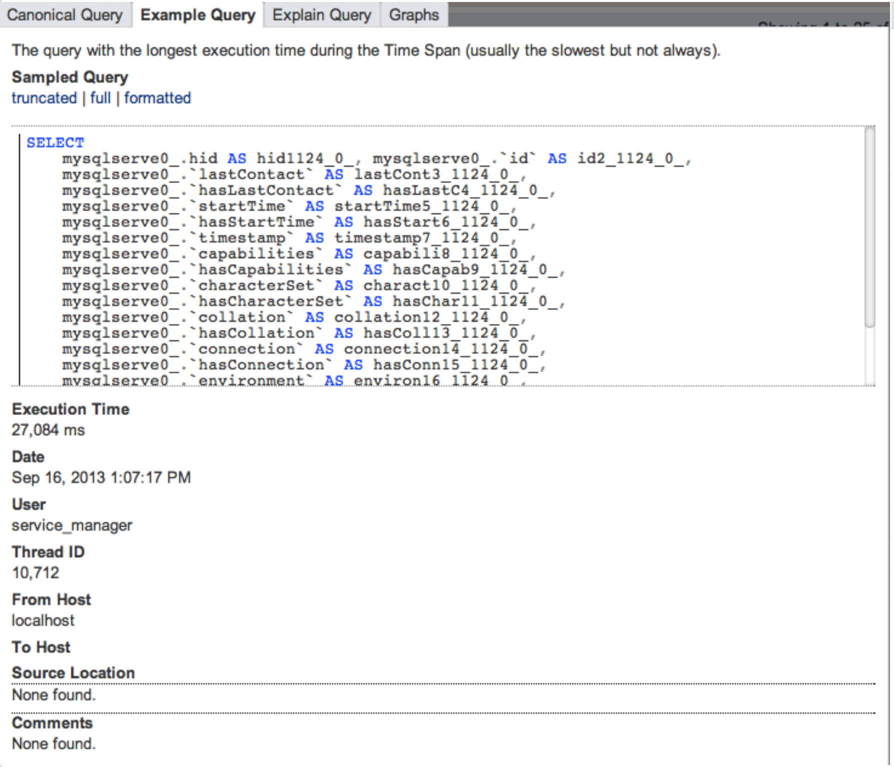
Show 25 entries Export data options...

Showing 1 to 25 of 1,197 entries

Query	Database	⚠	Counts			QRTi	Latency (hh:mm:ss.ms)					Rows	
			Exec	Err	Warn		Total	Max	Avg	Locks	Avg History	Total	Examined
COMMIT (1)	mem		24,707	0	0	0.92	20:27.872	5.828	0.050	0.000		0	0
INSERT INTO 'mem__quan'...o') , 'hostTo') , ... (1)	mem		6,903	0	0	0.79	18:16.538	17.784	0.159	6.699		7,356	0
INSERT INTO 'mem__quan'...on' = IF (VALUES (... (1)	mem		6,985	0	0	0.86	10:54.856	8.940	0.094	8.301		7,332	0
INSERT INTO 'mem__quan'...en')) , 'lastSeen') (1)	mem		7,025	37	0	1.00	3:11.791	11.220	0.027	4.436		13,947	0
INSERT INTO 'mem__instr...tency') , 'latency') (1)	mem		740	0	0	0.77	1:48.459	8.745	0.147	0.147		1,386	0
SELECT 'mysqlconne0_' ....asProc18_1191_0_' , ... (1)	mem		974	0	0	0.90	1:01.829	12.359	0.063	0.353		974	974
SELECT 'mysqlserve0_' ...._' . 'hostCache' AS ... (1)	mem		1,801	0	0	0.93	53.661	13.351	0.030	0.636		1,801	1,801
INSERT INTO 'mem__instr...es') , 'diskWrites') (1)	mem		26	0	0	0.79	44.838	14.954	1.725	0.070		26	0
UPDATE 'mem__inventory'...n' = ? WHERE 'hid' = ? (1)	mem		321	0	0	0.80	43.886	9.668	0.137	0.044		321	321
UPDATE 'mem__config' . ... ? WHERE 'user_id' = ? (1)	mem		321	0	0	0.71	40.788	10.607	0.127	0.038		321	321
CREATE TEMPORARY TABLE ...('id' INT8 NOT NULL) (1)	mem		13	0	0	0.14	36.525	12.055	2.810	0.000		0	0
INSERT INTO 'mem__instr...nedTableDefinitions') (1)	mem		26	0	0	0.64	34.348	13.349	1.321	0.003		27	0
UPDATE 'mem__inventory'...p' = ? WHERE 'hid' = ? (1)	mem		416	0	0	0.81	33.469	13.060	0.080	0.042		416	416
INSERT INTO 'mem__instr...hed') , 'notCached') (1)	mem		26	0	0	0.61	32.509	11.782	1.250	0.003		27	0
SELECT * FROM (SELECT ...m_no_index_used' AS ... (1)	mem		14	0	0	0.00	29.832	9.602	2.131	0.008		7,459	227,050
INSERT INTO 'mem__instr...s') , 'connections') (1)	mem		25	0	0	0.82	29.462	14.294	1.178	0.005		26	0
INSERT INTO 'mem__instr... ('sent') , 'sent') (1)	mem		25	0	0	0.82	28.991	14.332	1.160	0.003		26	0
SELECT 'agent0_' . 'hid.... 'hid' = ? FOR UPDATE (1)	mem		909	0	0	0.96	28.632	6.964	0.031	0.158		909	909
UPDATE 'mem__events' . ...me' = ? WHERE 'id' = ? (1)	mem		395	0	0	0.92	27.544	10.061	0.070	0.038		395	395

3.0.1.7150 - Cerberus.local (192.168.1.67) - Sep 16, 2013 11:50:28 am (Up Since: 38 minutes ago) - About

Query Analyzer Query Details



The query with the longest execution time during the Time Span (usually the slowest but not always).

Sampled Query
truncated | full | formatted

```
SELECT
    mysqlserve0_.hid AS hid1124_0_, mysqlserve0_.`id` AS id2_1124_0_,
    mysqlserve0_.`lastContact` AS lastCont3_1124_0_,
    mysqlserve0_.`hasLastContact` AS hasLastC4_1124_0_,
    mysqlserve0_.`startTime` AS startTime5_1124_0_,
    mysqlserve0_.`hasStartTime` AS hasStart6_1124_0_,
    mysqlserve0_.`timestamp` AS timestamp7_1124_0_,
    mysqlserve0_.`capabilities` AS capabili8_1124_0_,
    mysqlserve0_.`hasCapabilities` AS hasCapab9_1124_0_,
    mysqlserve0_.`characterSet` AS charact10_1124_0_,
    mysqlserve0_.`hasCharacterSet` AS hasChar11_1124_0_,
    mysqlserve0_.`collation` AS collation12_1124_0_,
    mysqlserve0_.`hasCollation` AS hasColl13_1124_0_,
    mysqlserve0_.`connection` AS connection14_1124_0_,
    mysqlserve0_.`hasConnection` AS hasConn15_1124_0_,
    mysqlserve0_.`environment` AS environ16_1124_0_,
```

Execution Time
27,084 ms

Date
Sep 16, 2013 1:07:17 PM

User
service_manager

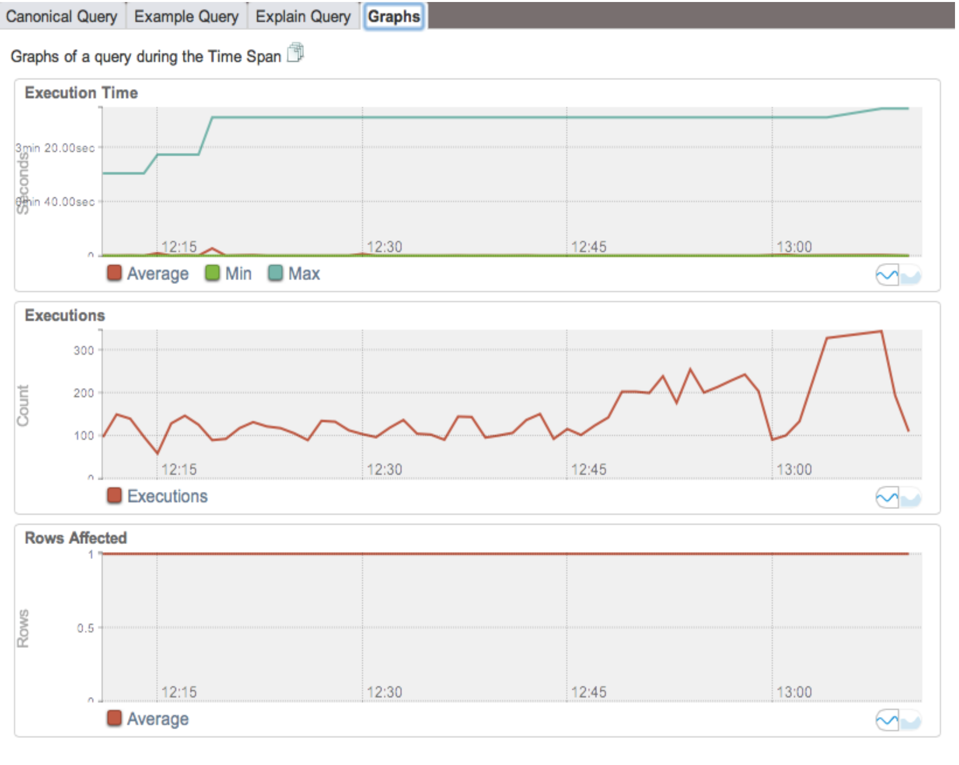
Thread ID
10,712

From Host
localhost

To Host

Source Location
None found.

Comments
None found.



Graphs of a query during the Time Span

Performance Schema

Some useful tables

- **events_statements_history**
  events_statements_history_long
  - Most recent statements executed
- **events_statements_summary_by_digest**
  - Summary for similar statements (same statement digest)
- **file_summary_by_event_name**
  - Interesting event: wait/io/file/innodb/innodb_data_file
- **table_io_waits_summary_by_table**
  table_io_waits_summary_by_index_usage
  - Statistics on storage engine access per table and index

Performance Schema

Statement digest

- Normalization of queries to group statements that are similar to be grouped and summarized:

```
SELECT * FROM orders WHERE o_custkey=10 AND o_totalprice>20
SELECT * FROM orders WHERE o_custkey = 20 AND o_totalprice > 100
```

➡ SELECT * FROM orders WHERE o_custkey = ? AND o_totalprice > ?

- `events_statements_summary_by_digest`

  DIGEST, DIGEST_TEXT, COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT, SUM_LOCK_TIME, SUM_ERRORS, SUM_WARNINGS, SUM_ROWS_AFFECTED, SUM_ROWS_SENT, SUM_ROWS_EXAMINED, SUM_CREATED_TMP_DISK_TABLES, SUM_CREATED_TMP_TABLES, SUM_SELECT_FULL_JOIN, SUM_SELECT_FULL_RANGE_JOIN, SUM_SELECT_RANGE, SUM_SELECT_RANGE_CHECK, SUM_SELECT_SCAN, SUM_SORT_MERGE_PASSES, SUM_SORT_RANGE, SUM_SORT_ROWS, SUM_SORT_SCAN, SUM_NO_INDEX_USED, SUM_NO_GOOD_INDEX_USED, FIRST_SEEN, LAST_SEEN

Performance Schema

Statement events

- Tables:
  `events_statements_current` (Current statement for each thread)
  `events_statements_history` (10 most recent statements per thread)
  `events_statements_history_long` (10000 most recent statements)
- Columns:

  THREAD_ID, EVENT_ID, END_EVENT_ID, EVENT_NAME, SOURCE, TIMER_START, TIMER_END, TIMER_WAIT, LOCK_TIME, SQL_TEXT, DIGEST, DIGEST_TEXT, CURRENT_SCHEMA, OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_INSTANCE_BEGIN, MYSQL_ERRNO, RETURNED_SQLSTATE, MESSAGE_TEXT, ERRORS, WARNINGS, ROWS_AFFECTED, ROWS_SENT, ROWS_EXAMINED, CREATED_TMP_DISK_TABLES, CREATED_TMP_TABLES, SELECT_FULL_JOIN, SELECT_FULL_RANGE_JOIN, SELECT_RANGE, SELECT_RANGE_CHECK, SELECT_SCAN, SORT_MERGE_PASSES, SORT_RANGE, SORT_ROWS, SORT_SCAN, NO_INDEX_USED, NO_GOOD_INDEX_USED, NESTING_EVENT_ID, NESTING_EVENT_TYPE

MySQL SYS Schema / ps_helper

- Started as a collection of views, procedures and functions, designed to make reading raw Performance Schema data easier
- Implements many common DBA and Developer use cases
- MySQL 5.7.7: Included by default
- Bundled within MySQL Workbench
- Also available on GitHub
  - https://github.com/MarkLeith/mysql-sys
- Examples of very useful functions:
  - format_time() , format_bytes(), format_statement()

MySQL SYS Schema

Example

statement_analysis: Lists a normalized statement view with aggregated statistics, mimics the MySQL Enterprise Monitor Query Analysis view, ordered by the total execution time per normalized statement

```
mysql> select * from statement_analysis limit 1\G
   *************************** 1. row ***************************
   query: INSERT INTO `mem__quan` . `nor ... nDuration` = IF ( VALUES ( ...
   db: mem
   full_scan:
   exec_count: 1110067
   err_count: 0
   warn_count: 0
   total_latency: 1.93h
   max_latency: 5.03 s
   avg_latency: 6.27 ms
   lock_latency: 00:18:29.18
   rows_sent: 0
   rows_sent_avg: 0
   rows_examined: 0
   rows_examined_avg: 0
   tmp_tables: 0
   tmp_disk_tables: 0
   rows_sorted: 0
   sort_merge_passes: 0
   digest: d48316a218e95b1b8b72db5e6b177788!
   first_seen: 2014-05-20 10:42:17
```

Structured EXPLAIN

FORMAT=JSON

```
EXPLAIN FORMAT=JSON

SELECT l_returnflag, l_linestatus, SUM(l_quantity)

FROM   lineitem

WHERE l_shipdate <=

         DATE_SUB('1998-12-01',  INTERVAL '118' DAY)

GROUP BY l_returnflag, l_linestatus

ORDER BY l_returnflag, l_linestatus;
```

EXPLAIN

```
{ "query_block": {
    "select_id": 1,
    "ordering_operation": {
      "using_filesort": false,
      "grouping_operation": {
        "using_temporary_table": true,
        "using_filesort": true,
        "table": {
          "table_name": "lineitem",
          "access_type": "ALL",
          "possible_keys": [
             "i_l_shipdate”
          ],
          "rows": 2829575,
          "filtered": 50,
          "attached_condition":
              "(`dbt3`.`lineitem`.`l_shipDATE` <=
              <cache>(('1998-12-01' - interval '118' day)))"
        }  /* table */
      }  /* grouping_operation */
    } /*ordering_operation */
 } /*query_block */  }
```

Structured EXPLAIN

Additional information compared to traditional EXPLAIN

- attached_condition

  "attached_condition": "(`test`.`t1`.`b` <> 30)"

- index_condition

  “index_condition": "(`test`.`t1`.`c` = 10)”

- used_key_parts

```
"used_key_parts": [
  "o_clerk",
   "o_orderDATE"
],
```

- rows_examined_per_join (5.7)

```
"rows_examined_per_scan": 1,
"rows_produced_per_join": 3,
```

- Cost (5.7)

```
 "query_block": {
     "select_id": 1,
     "cost_info": {
        "query_cost": "6.41"
      } /* cost_info */,

...

    "table": {  …

       "cost_info": {
          "read_cost": "3.00",
          "eval_cost": "0.60",
          "prefix_cost": "6.41",
          "data_read_per_join": "24"
       } /* cost_info */,
```

Optimizer Trace: Query Plan Debugging

- EXPLAIN shows the selected plan
- TRACE shows WHY the plan was selected:
  - Alternative plans
  - Estimated costs
  - Decisions made
- JSON format

Optimizer Trace: Example

```
SET optimizer_trace= “enabled=on“, end_markers_in_json=on;
SELECT * FROM t1, t2 WHERE f1=1 AND f1=f2 AND f2>0;
SELECT trace INTO DUMPFILE <filename>
    FROM information_schema.optimizer_trace;
SET optimizer_trace="enabled=off";
```

QUERY	SELECT * FROM t1,t2 WHERE f1=1 AND f1=f2 AND f2>0;
TRACE	“steps”: [{ "join_preparation": { "select#": 1,… } … } …]
MISSING_BYTES_BEYOND_MAX_MEM_SIZE	0
INSUFFICIENT_PRIVILEGES	0

Program Agenda

1. Introduction to MySQL optimizer
2. Selecting data access method
3. Join optimizer
4. Sorting
5. Tools for monitoring, analyzing, and tuning queries
6. Influencing the optimizer

Influencing the Optimizer

When the optimizer does not do what you want

- Add indexes
- Force use of specific indexes:
  - USE INDEX, FORCE INDEX, IGNORE INDEX
- Force specific join order:
  - STRAIGHT_JOIN
- Adjust session variables
  - optimizer_switch flags: set optimizer_switch="index_merge=off"
  - Buffer sizes: set sort_buffer=8*1024*1024;
  - Other variables: set optimizer_prune_level = 0;

MySQL 5.7: New Optimizer Hints

- Ny hint syntax:
  - **SELECT /*+ HINT1(*args*) HINT2(*args*) */ … FROM …**
- New hints:
  - BKA(*tables*)/NO_BKA(*tables*)
  - BNL(*tables*)/NO_BNL(*tables*)
  - MRR(*table indexes*)/NO_MRR(*table indexes*)
  - NO_ICP(*table indexes*)
  - NO_RANGE_OPTIMIZATION(*table indexes*)
  - QB_NAME(*name*)
- Finer granularilty than **optimizer_switch** session variable

Optimizer Hints

Future hints

- Hints for subquery / semi-join execution:

```
SELECT /*+ SEMIJOIN(@subq1 LOOSESCAN)
                   NO_SEMIJOIN(@subq2 DUPSWEEDOUT) */ a, b FROM t1
WHERE a IN (SELECT /*+ QB_NAME(subq1) c FROM t2 WHERE d > 10)
    AND b IN (SELECT /*+ QB_NAME(subq2) e FROM t3);
```

- Other hints to consider
  - Enable/disable merge of views and derived tables
  - Force/ignore index_merge alternatives
  - Join order: LEADING(*t1 t2 ...*)
- Plan to reimplement existing hints in new syntax

MySQL 5.7: Query Rewrite Plugin

- Rewrite problematic queries without the need to make application changes
  - Add hints
  - Modify join order
  - Much more …
- Add rewrite rules to table:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement ) VALUES
 ("SELECT * FROM t1 WHERE a > ? AND b = ?",
  "SELECT * FROM t1 FORCE INDEX (a_idx) WHERE a > ? AND b = ?");
```

- New pre and post parse query rewrite APIs
  - Users can write their own plug-ins

More information

- My blog:
  - http://oysteing.blogspot.com/
- Optimizer team blog:
  - http://mysqloptimizerteam.blogspot.com/
- MySQL Server Team blog
  - http://mysqlserverteam.com/
- MySQL forums:
  - Optimizer & Parser: http://forums.mysql.com/list.php?115
  - Performance: http://forums.mysql.com/list.php?24

Q&A