ORACLE TECHNOLOGY NETWORK

Virtual Technology Summit

Hands-On Learning With Oracle and Community Experts

Where Technology and Community Meet

How to Analyze and Tune MySQL Queries for Better Performance

Øystein Grøvlen
Senior Principal Software Engineer
MySQL Optimizer Team, Oracle
March/April, 2016

Please Stand By. This session will begin promptly at the time indicated on the agenda. Thank You.

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

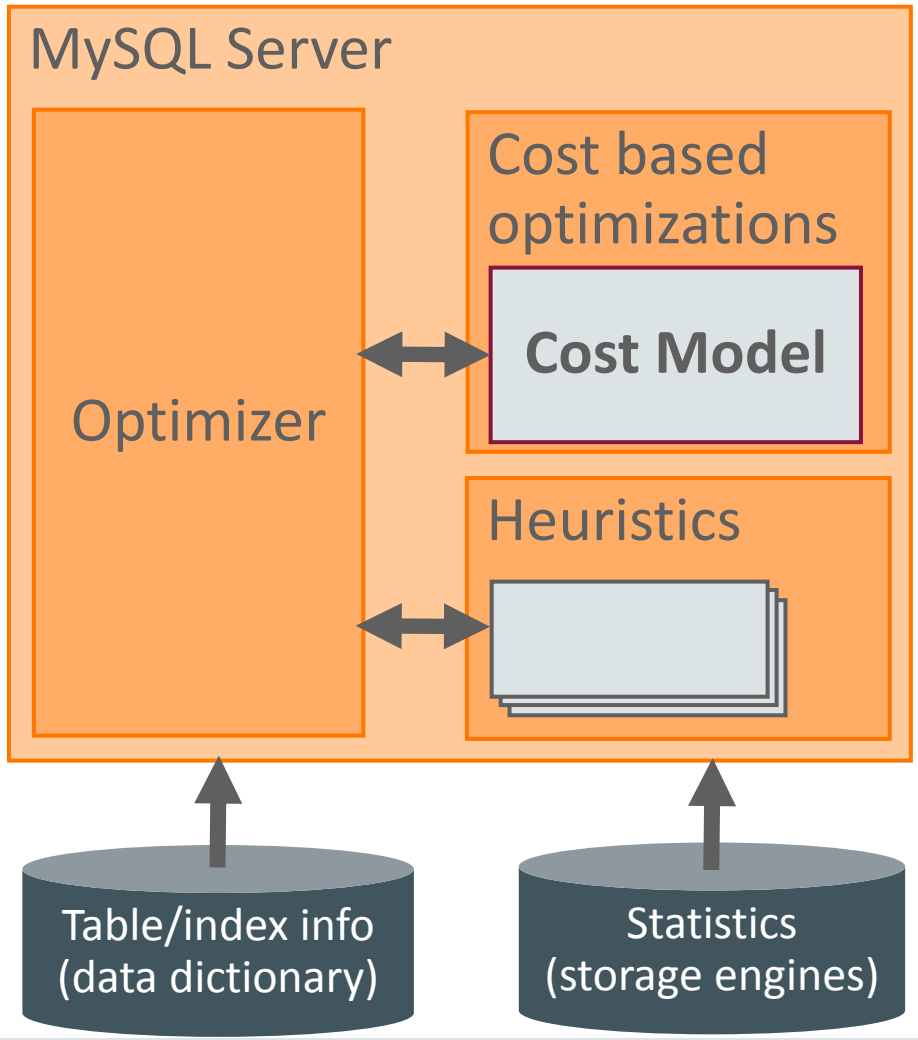
1. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
6. Influencing the optimizer

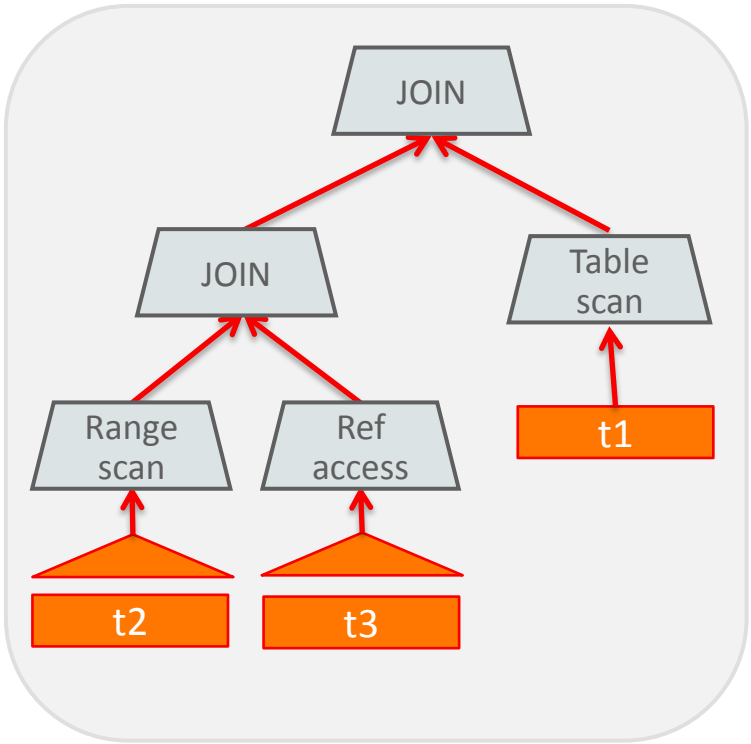
Program Agenda

1. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
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;
```

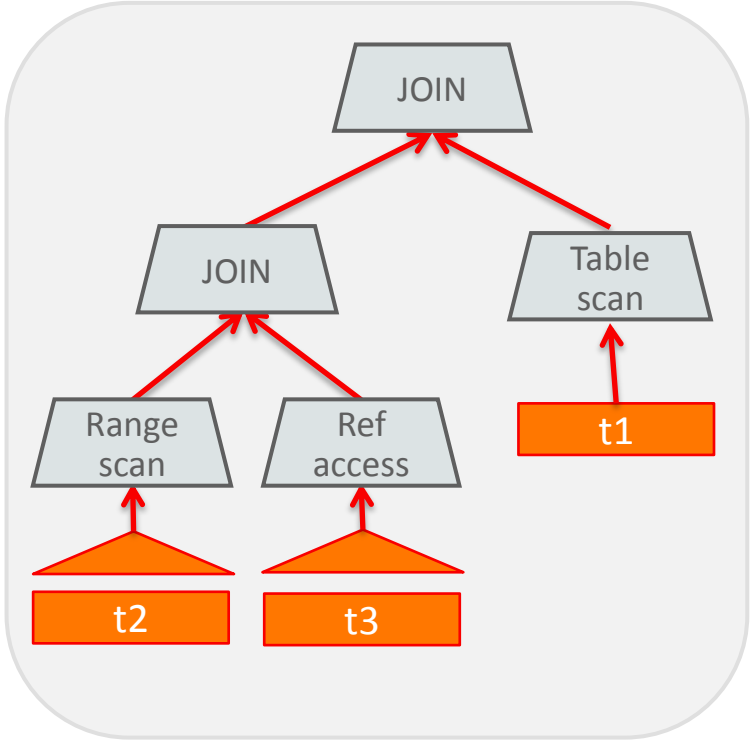




Cost-based Query Optimization

General idea

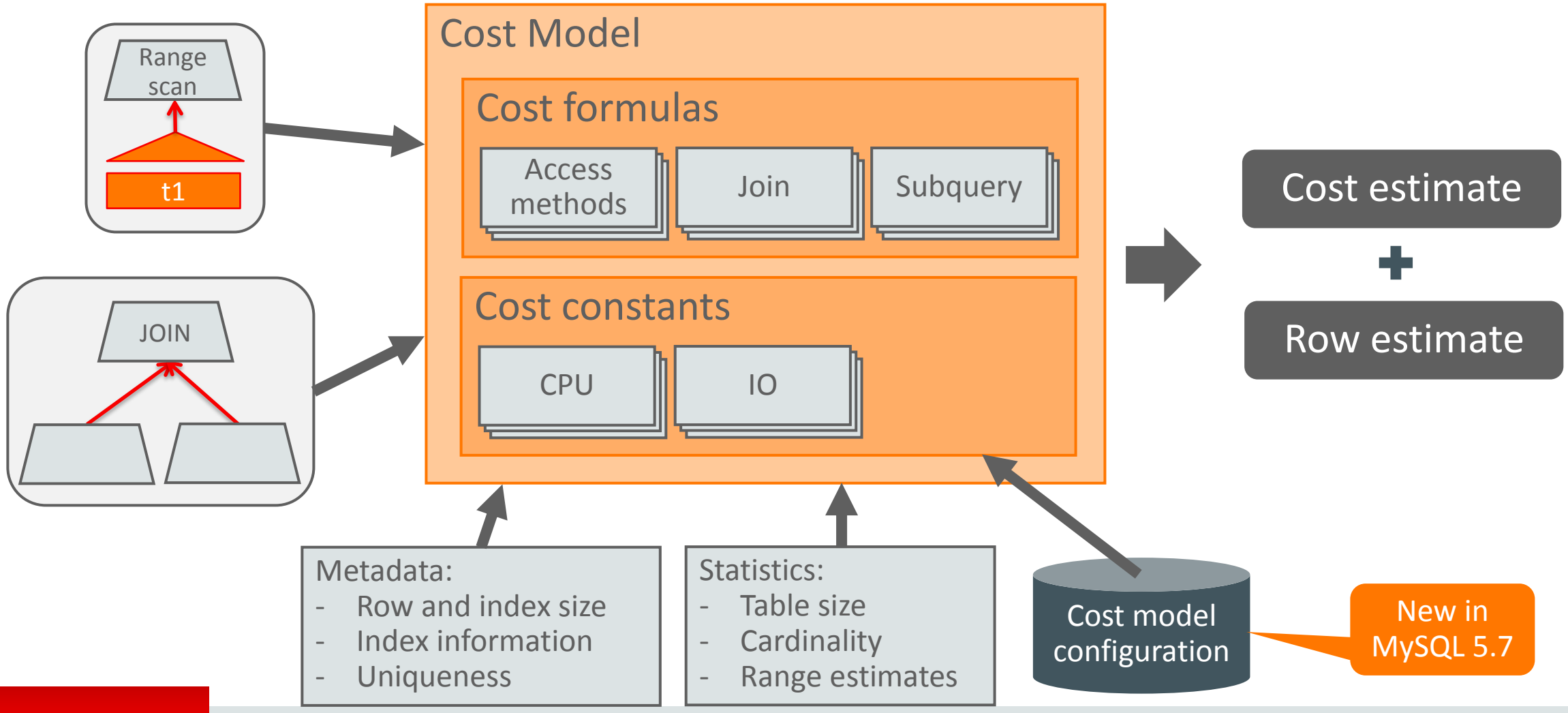
- Assign cost to operations
- Assign cost to partial or alternative plans
- Search for plan with lowest cost



- Cost-based optimizations:

Access method | Join order | Subquery strategy

Optimizer Cost Model

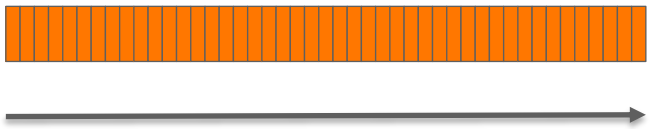


Cost Model Example

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

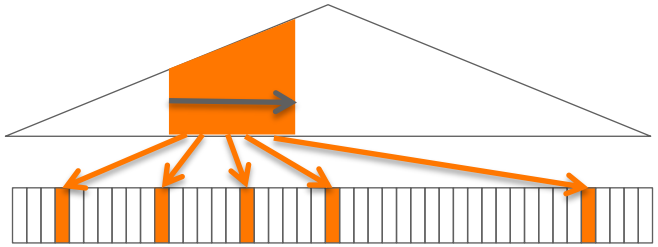
Table scan:

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



Range scan (on secondary index):

- IO-cost: #rows_in_range * IO_BLOCK_READ_COST
- 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 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 Scan

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

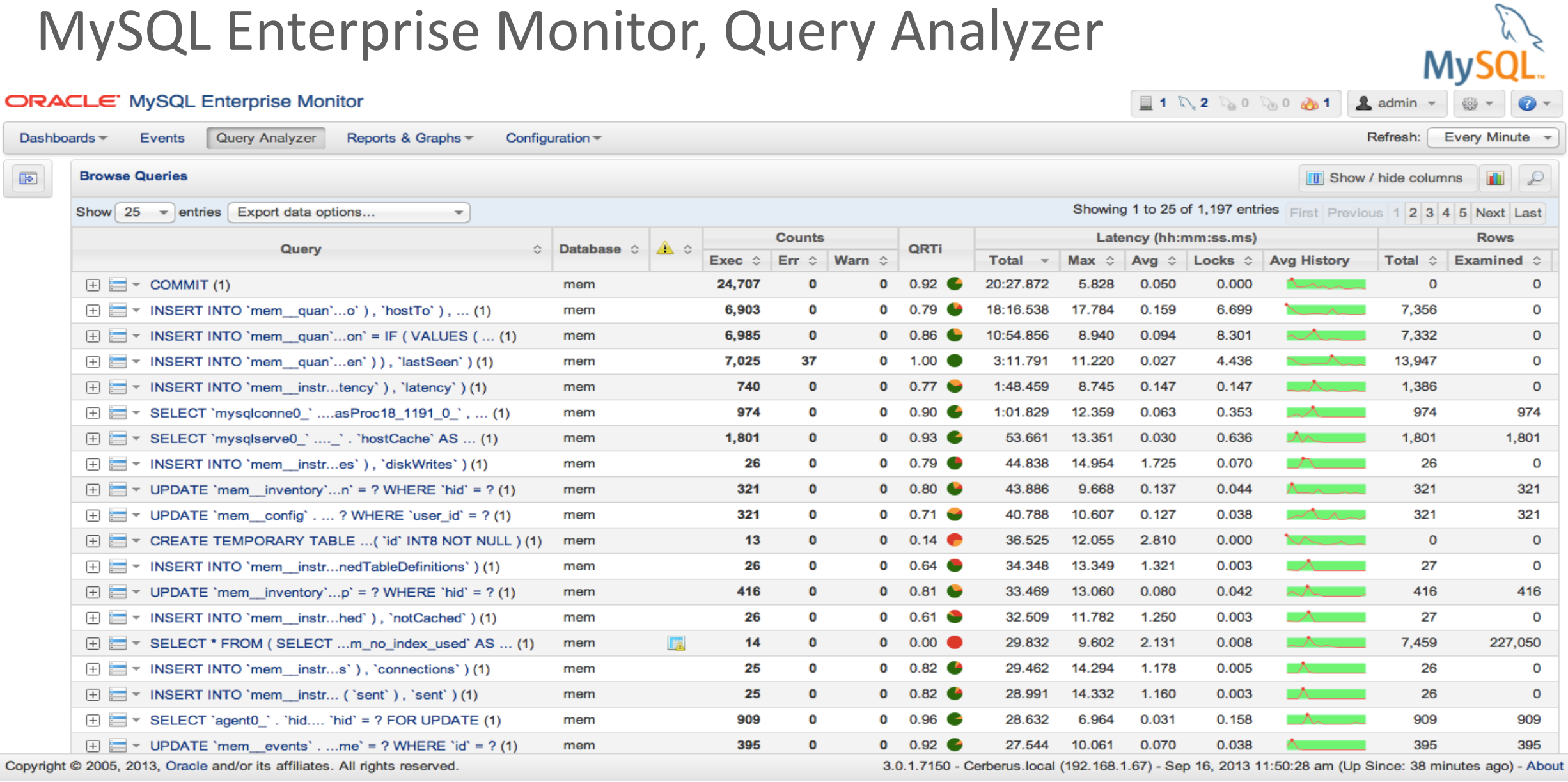
Program Agenda

1. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
6. Influencing the optimizer

Useful tools

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

MySQL Enterprise Monitor, Query Analyzer



Query Analyzer Query Details

Canonical Query | **Example Query** | Explain Query | Graphs

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

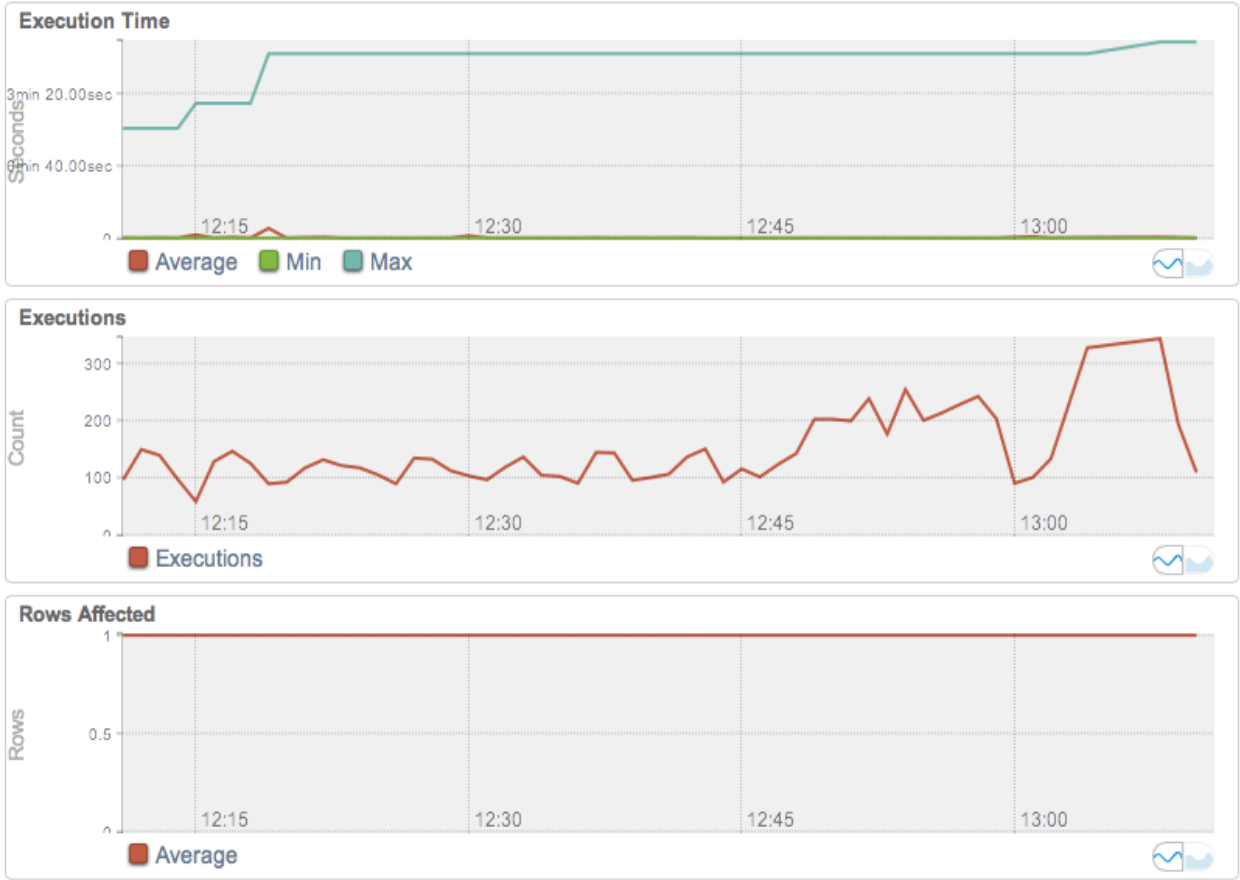
To Host

Source Location
None found.

Comments
None found.

Canonical Query | Example Query | Explain Query | **Graphs**

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 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

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

MySQL sys Schema

- A collection of views, procedures and functions, designed to make reading raw Performance Schema data easier
- Implements many common DBA and Developer use cases
  - File IO usage per user
  - Which indexes is never used?
  - Which queries use full table scans?
- Examples of very useful functions:
  - format_time() , format_bytes(), format_statement()
- Included with MySQL 5.7
- Bundled with MySQL Workbench

MySQL sys Schema

Example

statement_analysis: Lists a normalized statement view with aggregated statistics, ordered by the total execution time per normalized statement

```
mysql> SELECT * FROM sys.statement_analysis LIMIT 1\G
*************************** 1. row ***************************
query: INSERT INTO `mem__quan` . `nor ... nDuration` = IF ( VALUES ( ...
db: mem
full_scan: 0
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
```

EXPLAIN

Understand the query plan

- Use EXPLAIN to print the final query plan:

```
EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.a = t2.a WHERE b > 10 AND c > 10;
+----+--------+-------+------------+------+---------------+-----+---------+-----+------+----------+------------+
| id | select…| table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra      |
+----+--------+-------+------------+------+---------------+-----+---------+-----+------+----------+------------+
|  1 | SIMPLE | t1    | NULL       | range| idx1          | idx1| 4       | NULL|   12 |    33.33 | Using where|
|  1 | SIMPLE | t2    | NULL       | ref  | idx2          | idx2| 4       | t1.a|    1 |   100.00 | NULL       |
+----+--------+-------+------------+------+---------------+-----+---------+-----+------+----------+------------+
```

- Explain for a running query (New in MySQL 5.7):

EXPLAIN FOR CONNECTION *connection_id*;

Structured EXPLAIN

- JSON format:

  EXPLAIN FORMAT=JSON SELECT …

- Contains more information:
  - Used index parts
  - Pushed index conditions
  - Cost estimates
  - Data estimates

Added in MySQL 5.7

```
EXPLAIN FORMAT=JSON
SELECT * FROM t1 WHERE b > 10 AND c > 10;
EXPLAIN
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "17.81"
    },
    "table": {
      "table_name": "t1",
      "access_type": "range",
      "possible_keys": [
        "idx1"
      ],
      "key": "idx1",
      "used_key_parts": [
        "b"
      ],
      "key_length": "4",
      "rows_examined_per_scan": 12,
      "rows_produced_per_join": 3,
      "filtered": "33.33",
      "index_condition": "(`test`.`t1`.`b` > 10)",
      "cost_info": {
        "read_cost": "17.01",
        "eval_cost": "0.80",
        "prefix_cost": "17.81",
        "data_read_per_join": "63"
      },
      .........
      "attached_condition": "(`test`.`t1`.`c` > 10)"
    }
  }
}
```

Structured EXPLAIN

Assigning Conditions to Tables

EXPLAIN FORMAT=JSON SELECT * FROM t1, t2
WHERE t1.a=t2.a AND t2.a=9 AND (NOT (t1.a > 10 OR t2.b >3) OR (t1.b=t2.b+7 AND t2.b = 5));

EXPLAIN


```
{
 "query_block": {
  "select_id": 1,
  "nested_loop": [
   {
    "table": {
     "table_name": "t1",
     "access_type": "ALL",
     "rows": 10,
     "filtered": 100,
     "attached_condition": "(t1.a = 9)"
    } /* table */
   },
   {
    "table": {
     "table_name": "t2",
     "access_type": "ALL",
     "rows": 10,
     "filtered": 100,
     "using_join_buffer": "Block Nested Loop",
     "attached_condition": "((t2.a = 9) and ((t2.b <= 3) or ((t2.b = 5) and (t1.b = 12))))"
    } /* table */
   }
  ] /* nested_loop */
 } /* query_block */
}
```

Optimizer Trace: Query Plan Debugging

- EXPLAIN shows the selected plan
- Optimizer trace shows WHY the plan was selected

```
SET optimizer_trace= "enabled=on";
SELECT * FROM t1,t2 WHERE f1=1 AND f1=f2 AND f2>0;
SELECT trace FROM information_schema.optimizer_trace
        INTO OUTFILE <filename> LINES TERMINATED BY '';
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. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
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
- Index look-up (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

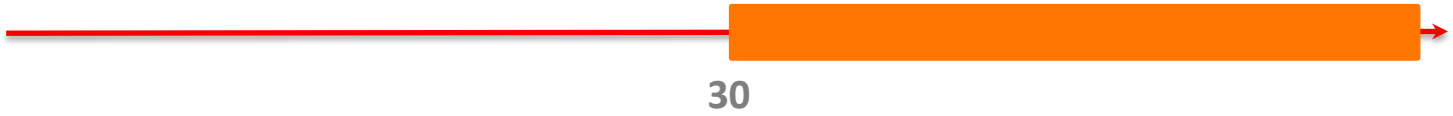
Range Optimizer

- 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: 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)
```

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)
```

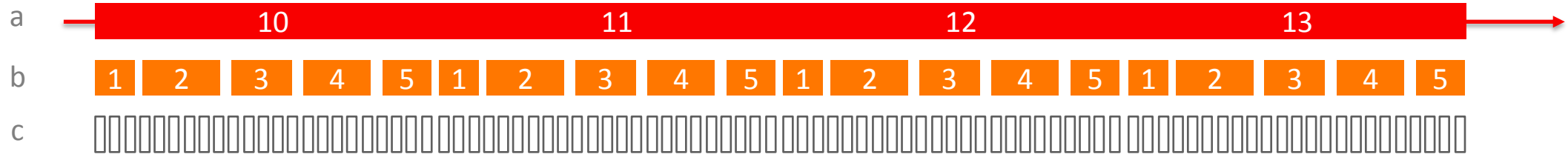
Range Access for Multi-Column Indexes

Example table with multi-part index

- Table:

- INDEX idx(a, b, c);

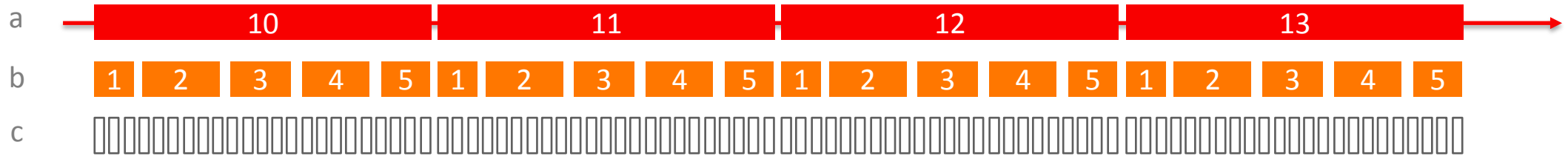
- Logical storage layout of index:



Range Access for Multi-Column Indexes, cont

- Equality on 1st index column?
  - Can add condition on 2nd index column to range condition
- Example:

  SELECT * from t1 WHERE a IN (10,11,13) AND (b=2 OR b=4)

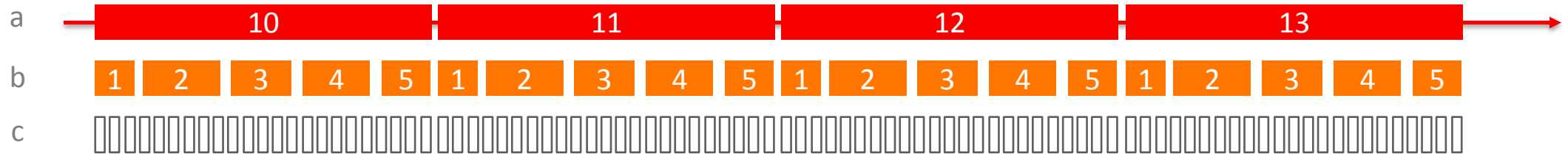


- Resulting range scan:

Range Access for Multi-Column Indexes, cont

- Non-Equality on 1st index column:
  - Can **NOT** add condition on 2nd index column to range condition
- Example:

  SELECT * from t1 WHERE a > 10 AND a < 13 AND (b=2 OR b=4)



- Resulting range scan:

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

MySQL 5.7:
Enabled by default

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

```
mysql> SELECT sql_text, (timer_wait)/1000000000.0 "t (ms)", rows_examined rows
         FROM performance_schema.events_statements_history ORDER BY timer_start;
+-----------------------------------------------------------+--------+------+
| sql_text                                                  | t (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. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
6. Influencing the optimizer

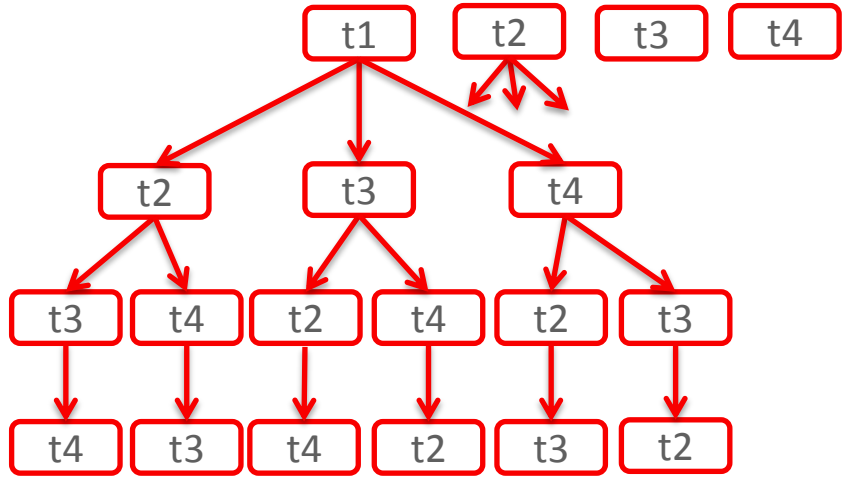
Join Optimizer

”Greedy search strategy”

- Goal: Given a JOIN of N tables, find the best JOIN ordering

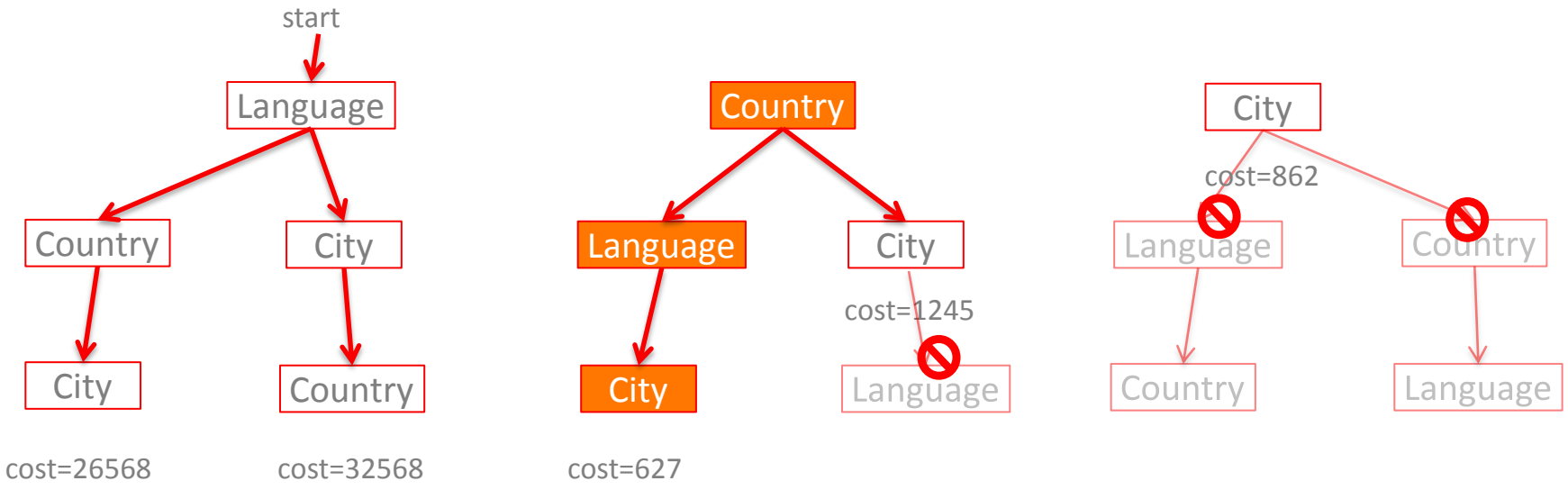
N! possible plans

- Strategy:
  - Start with all 1-table plans (Sorted based on *size* and *key dependency*)
  - 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**)



JOIN Optimizer Illustrated

```
SELECT City.Name, Language FROM Language, Country, City
WHERE City.CountryCode = Country.Code
  AND City.ID = Country.Capital
  AND City.Population >= 1000000
  AND Language.Country = Country.Code;
```



Join Optimizer: Case study

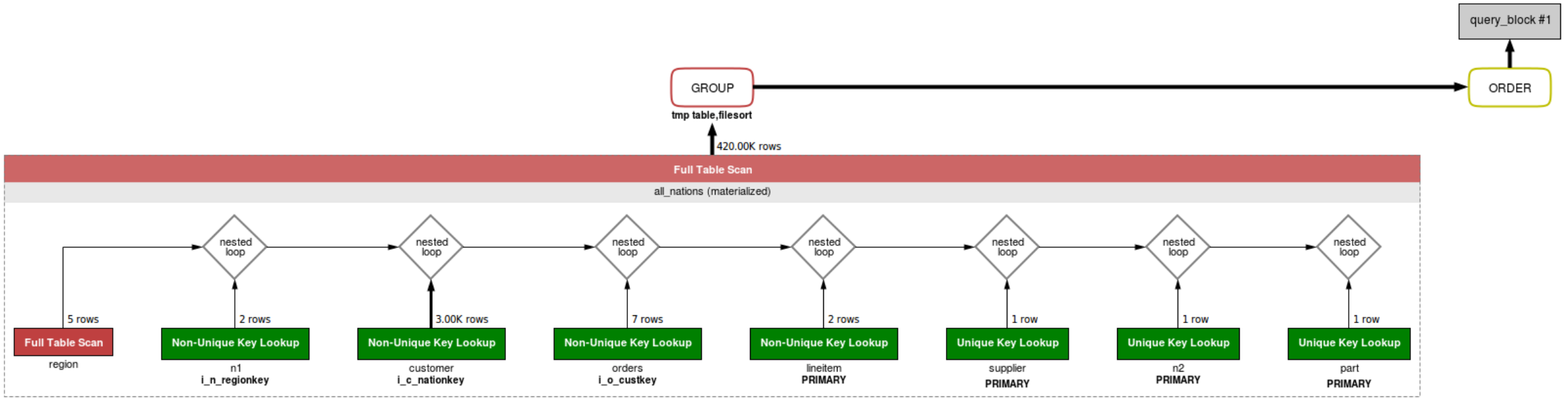
DBT-3 Query 8: National Market Share Query

```
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: 21 seconds



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;
```

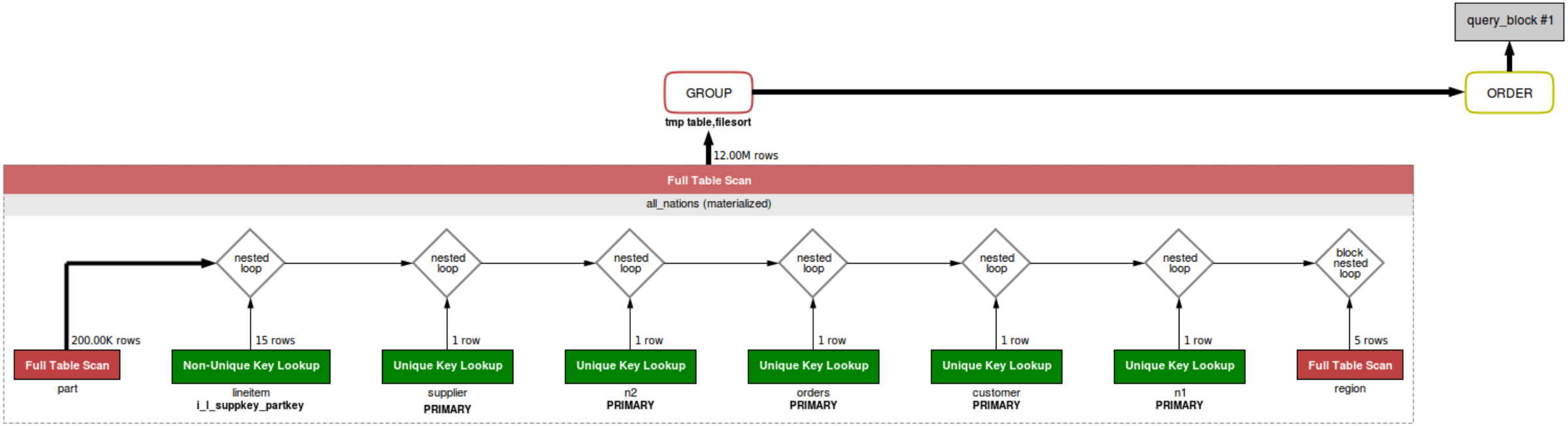
part before lineitem

Highest selectivity

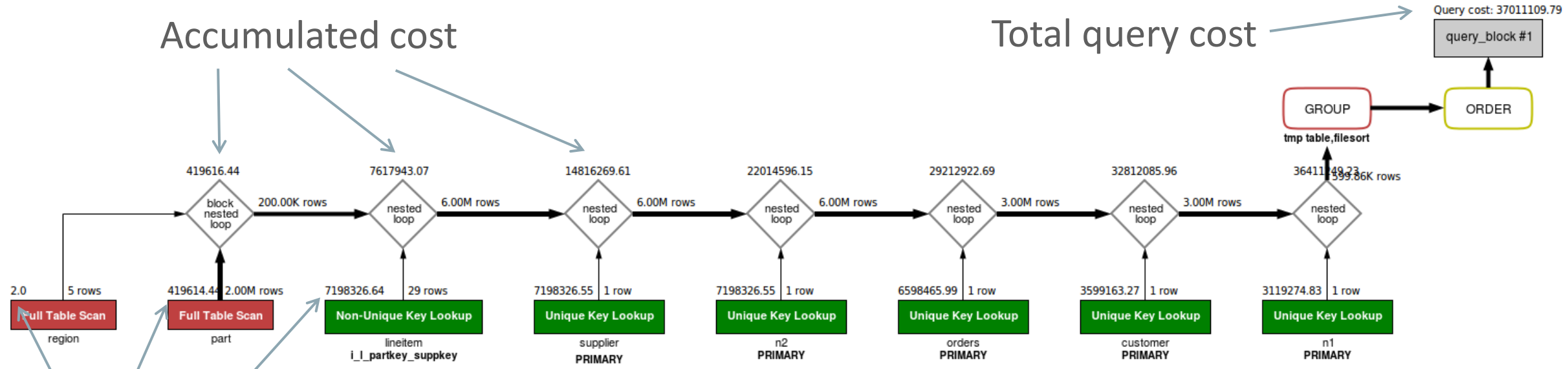
Join Optimizer: Case study

Improved join order

Execution time: 3 seconds



MySQL 5.7: Cost Information in Structured EXPLAIN



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. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
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. (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 |
+-----------------------------------------------+--------------------+
```

Using Index to Avoid Sorting

CREATE INDEX i_o_totalprice ON orders(o_totalprice);

SELECT o_orderkey, o_totalprice FROM orders ORDER BY o_totalprice ;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	index	NULL	i_o_totalprice	6	NULL	15000000	Using index

However, still (due to total cost):

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

Using Index to Avoid Sorting

Case study revisited

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. Cost-based query optimization in MySQL
2. Tools for monitoring, analyzing, and tuning queries
3. Data access and index selection
4. Join optimizer
5. Sorting
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_search_depth = 10;

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*)
  - SEMIJOIN/NO_SEMIJOIN(*strategies*), SUBQUERY(*strategy)*
  - NO_ICP(*table indexes*)
  - NO_RANGE_OPTIMIZATION(*table indexes*)
  - QB_NAME(*name*)
- Finer granularilty than **optimizer_switch** session variable

MySQL 5.7: Hint Example: SEMIJOIN

- No hint, optimizer chooses semi-join algorithm LooseScan:

EXPLAIN SELECT * FROM t2 WHERE t2.a IN (SELECT a FROM t3);

id	select_type	table	type	possible_keys	key	key_len	ref	rows	extra
1	SIMPLE	t3	index	a	a	4	NULL	3	Using where; LooseScan
1	SIMPLE	t2	ref	a	a	4	test.t3.a	1	Using index

- Disable semi-join with hint:

EXPLAIN SELECT * FROM t2 WHERE t2.a IN (SELECT /*+ NO_SEMIJOIN() */ a FROM t3);

id	select_type	table	type	possible_keys	key	key_len	ref	rows	extra
1	PRIMARY	t2	index	null	a	4	NULL	4	Using where; Using index
2	DEPENDENT SUBQUERY	t3	Index_subquery	a	a	4	func	1	Using index

MySQL 5.7: Hint Example: SEMIJOIN

- Force Semi-join Materialization to be used

```
EXPLAIN SELECT /*+ SEMIJOIN(@subq MATERIALIZATION) */ * FROM t2
        WHERE t2.a IN (SELECT  /*+ QB_NAME(subq) */ a FROM t3);
```

id	select_type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	t2	index	a	a	4	NULL	4	Using where; Using index
1	SIMPLE	<subquery2>	eq_ref	<auto_key>	<auto_key>	4	test.t2.a	1	NULL
2	MATERIALIZED	t3	index	a	a	4	NULL	3	Using index

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

Thank You for Joining Us Today
Please Move to the Next Session.

Integrated Cloud

Applications & Platform Services


ORACLE®