How to Analyze and Tune MySQL Queries for Better Performance

Øystein Grøvlen
Senior Principal Software Engineer
MySQL Optimizer Team, Oracle

ORACLE

MySQL

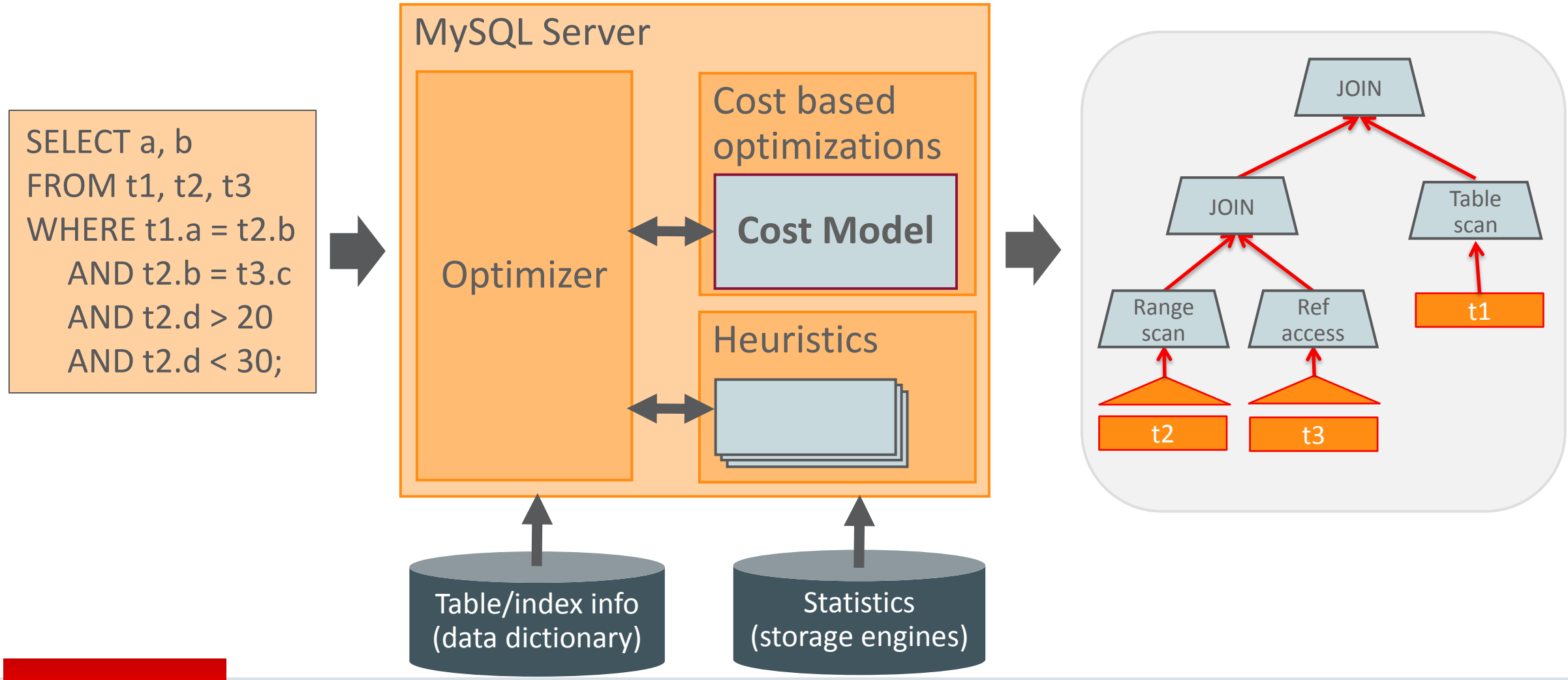
Program Agenda

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. Subqueries
6. Sorting
7. 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. Subqueries
6. Sorting
7. Influencing the optimizer

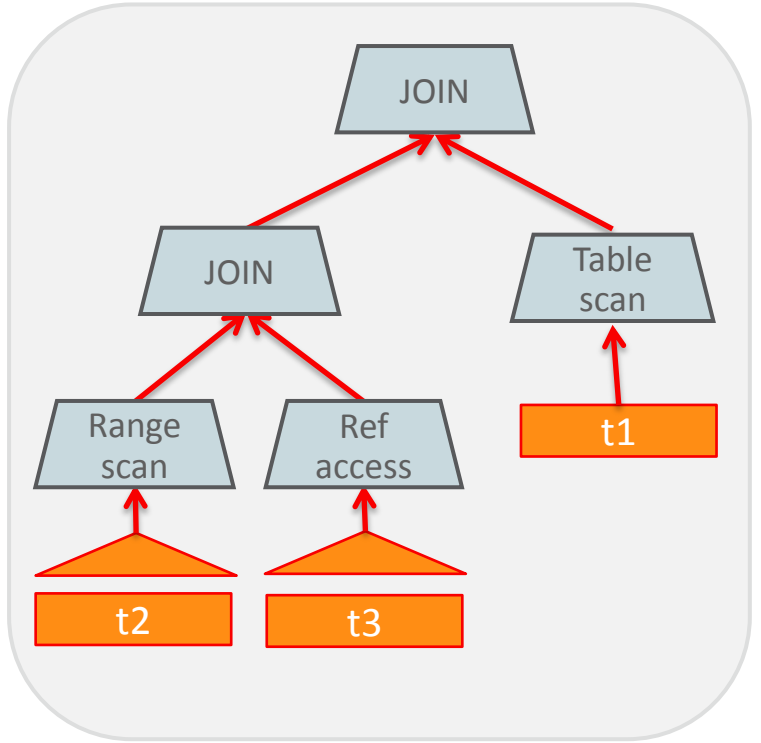
MySQL Optimizer



Cost-based Query Optimization

General idea

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

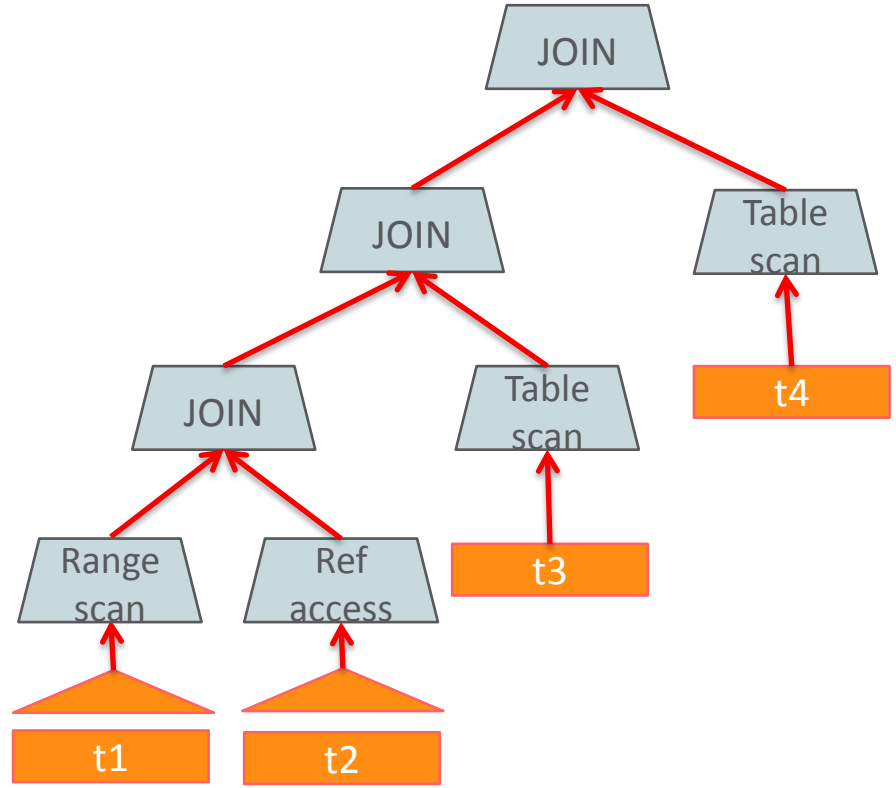


- Cost-based optimizations:

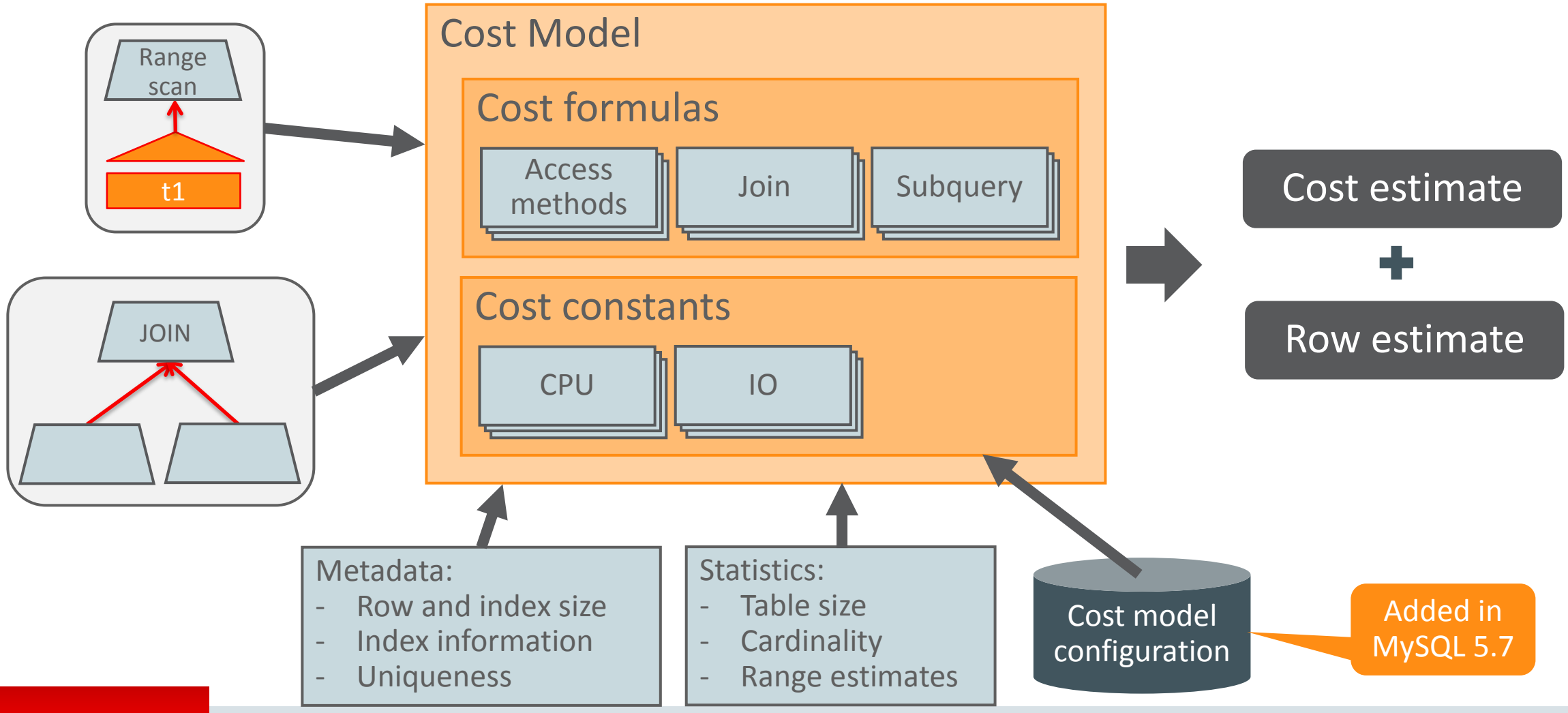
Access method	Join order	Subquery strategy

MySQL Optimizer Characteristics

- Produce the query plan that uses least resources
  - IO and CPU
- Optimizes a single query
  - No inter-query optimizations
- Produces left-deep linear query execution plan



Optimizer Cost Model



Cost Estimates

- The **cost** for **executing** a query
- **Cost unit:**
  - "read a random data page from disk"
- Main cost factors:
  - IO cost:
    - #pages read from table
    - #pages read from index
  - CPU cost:
    - Evaluating query conditions
    - Comparing keys/records
    - Sorting keys

- Main cost constants:

Cost	Default value
Read a random disk page	1.0
Read a data page from memory buffer	1.0
Evaluate query condition	0.2
Compare keys/records	0.1

MySQL 5.7: Configurable

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 records 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
  - MySQL 8.0: Percentage of table/index in InnoDB buffer pool

InnoDB Persistent Statistics

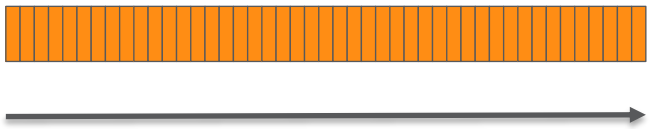
- More accurate statistics
  - New algorithm for sampling
  - Less variance between servers
- More stable statistics (Will not be changed by restart)
- Turned on by default
- Automatically recalculates statistics after significant changes
  - May turn off automatic recalculations
- ANALYZE TABLE forces recalculation of statistics
- May increase precision by changing number of samples
  - **innodb_stats_persistent_sample_pages**

Cost Model Example

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

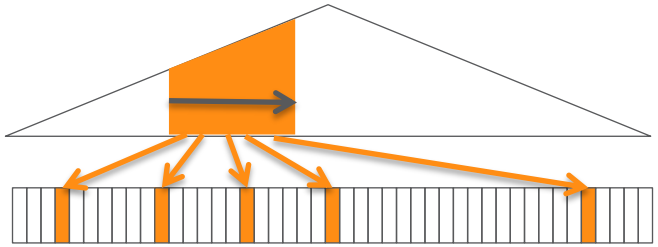
Table scan:

- 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	rows	filtered	Extra
1	SIMPLE	orders	ALL	i_o_orderdate	NULL	NULL	15000000	29.93	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	rows	filtered	Extra
1	SIMPLE	orders	range	i_o_orderdate	i_o_orderdate	4	2235118	100.00	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

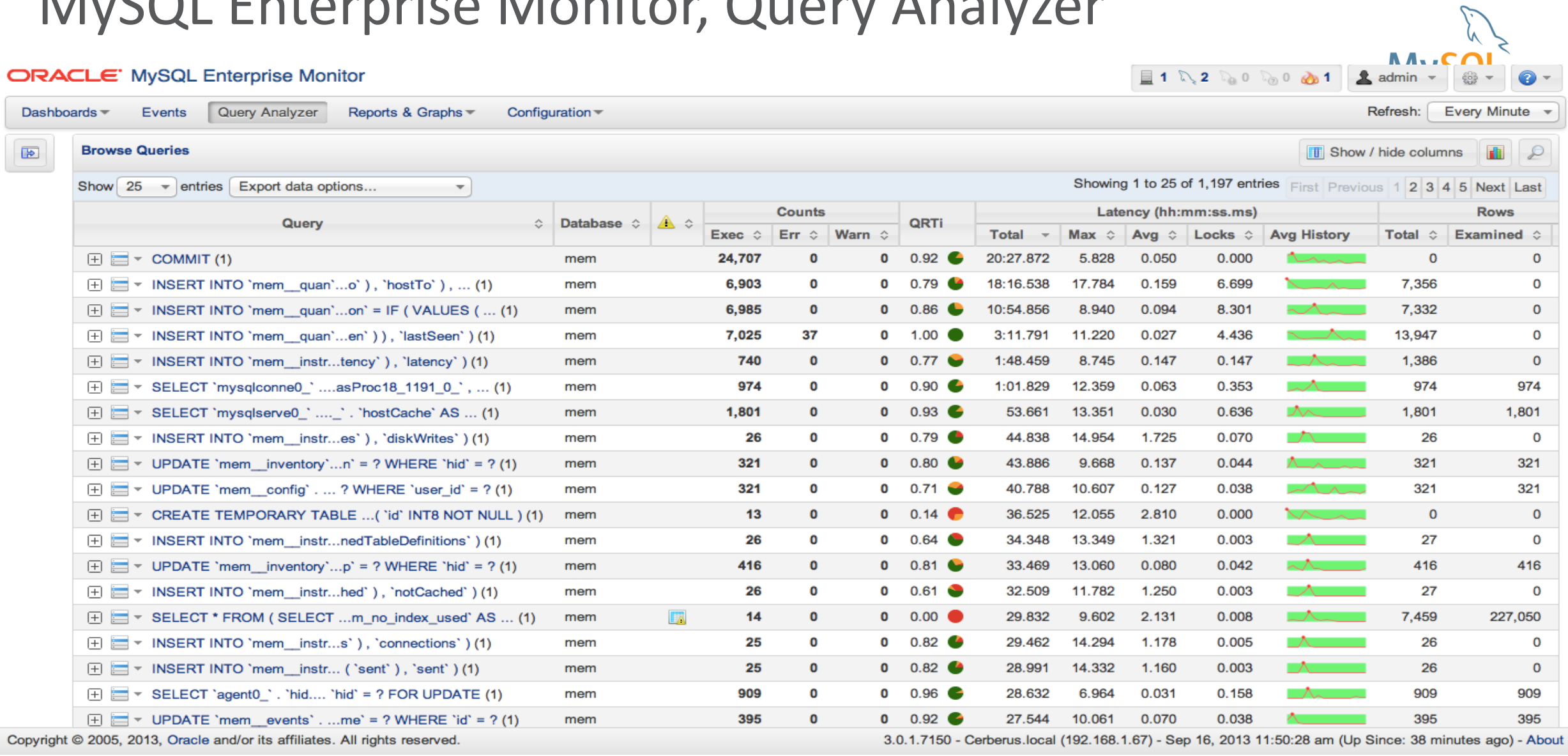
Program Agenda

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. Subqueries
6. Sorting
7. 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

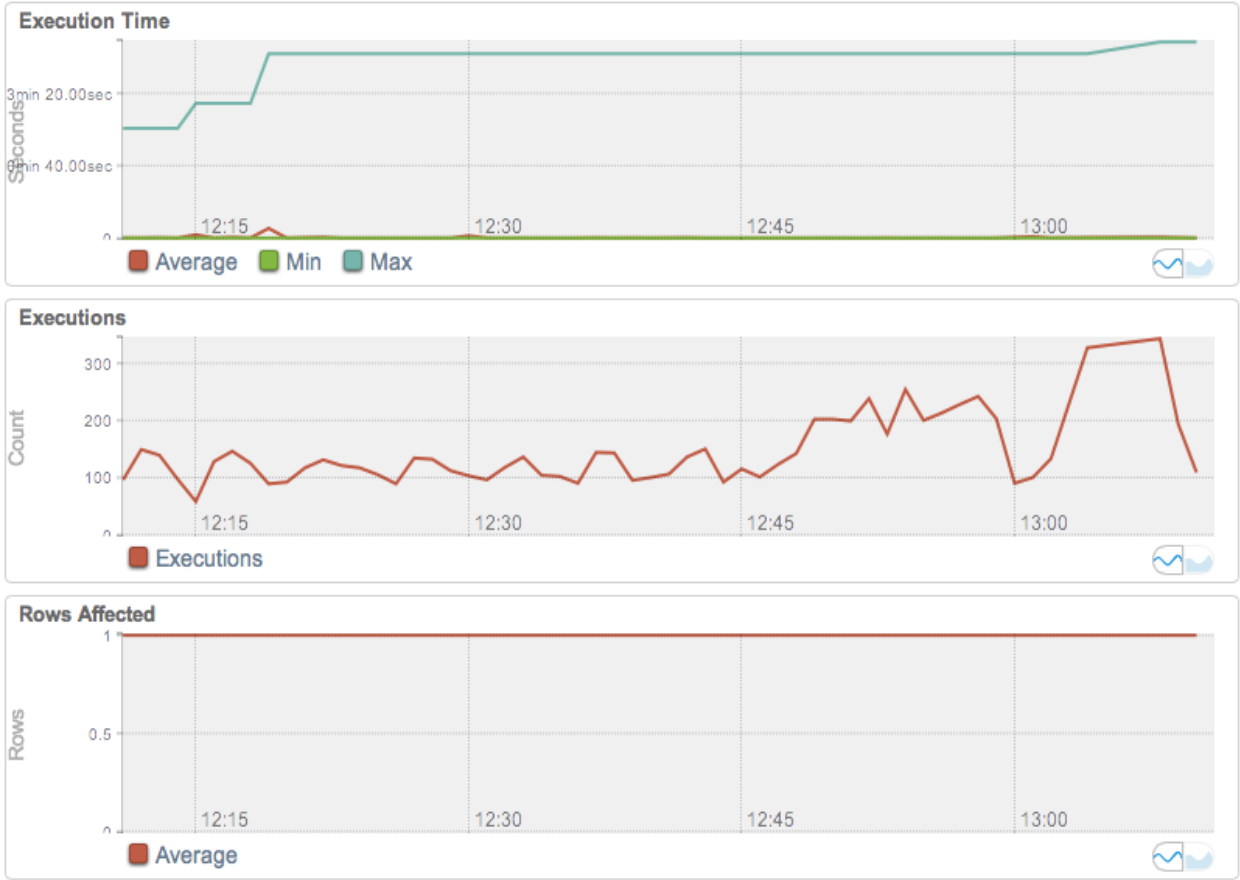
To Host

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_type	table	partitions	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	t1	NULL	range	PRIMARY, idx1	idx1	4	NULL	12	33.33	Using index condition
2	SIMPLE	t2	NULL	ref	idx2	idx2	4	t1.a	1	100.00	NULL

- Explain for a running query (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

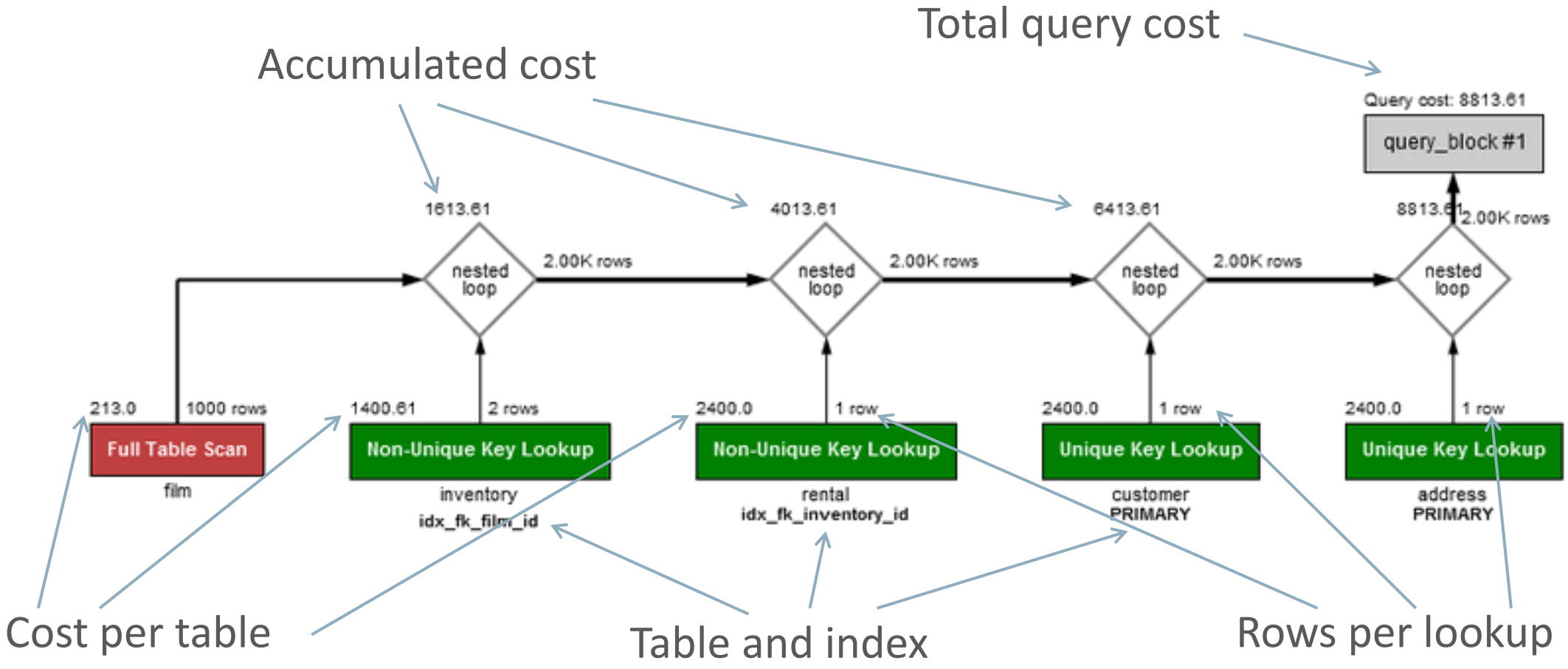
Assigning Conditions to Tables

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 */
}
```

Visual EXPLAIN (MySQL Workbench)



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

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 */
```

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. Subqueries
6. Sorting
7. 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

Index Lookup (Ref Access)

- Read all records with a given key value using an index:
- Examples:

  SELECT * FROM t1 WHERE t1.key = 7;

  SELECT * FROM t1, t2 WHERE t1.key = t2.key;

- "eq_ref":
  - Reading from a unique index, max one record returned
- "ref":
  - Reading from a non-unique index or a prefix of an index, possibly multiple records returned
  - The record estimate is based on cardinality number from index statistics

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	filtered	Extra
1	SIMPLE	customer	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL

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

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	orders	ref	i_o_orderdate	i_o_orderdate	4	const	6272	100.00	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	filtered	Extra
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_orderdate	4	const	6272	100.00	Using where
1	SIMPLE	customer	eq_ref	PRIMARY	PRIMARY	4	dbt3. orders. o_custkey	1	100.00	NULL

Ref Access

Join Queries, continued

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	filtered	Extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	33.33	Using where
1	SIMPLE	orders	ref	i_o_custkey	i_o_custkey	5	dbt3. customer. c_custkey	7	100.00	NULL

Range Optimizer

- Goal: find the "minimal" ranges for each index
- 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, cont.

- Range optimizer selects the “useful” parts of the WHERE condition:
  - Conditions comparing a column value with a constant:

key > 3

key = 4

key BETWEEN 4 AND 6

key IN (10,12,..)

key IS NULL

key LIKE "abc%"

  - Nested AND/OR conditions are supported
- Result: list of disjoint ranges that need to be read from index:

- Cost estimate based on number of records in each range:
  - Record estimate is found by asking the Storage Engine (“index dives”)

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)
```

ORACLE

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)
```

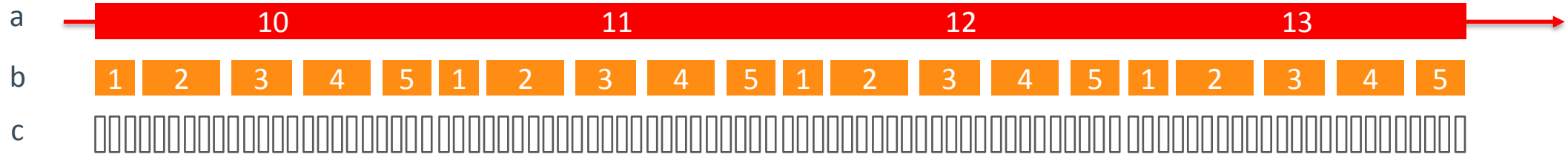
Range Access for Multi-Column Indexes

Example table with multi-part index

- Table:

- INDEX idx(a, b, c);

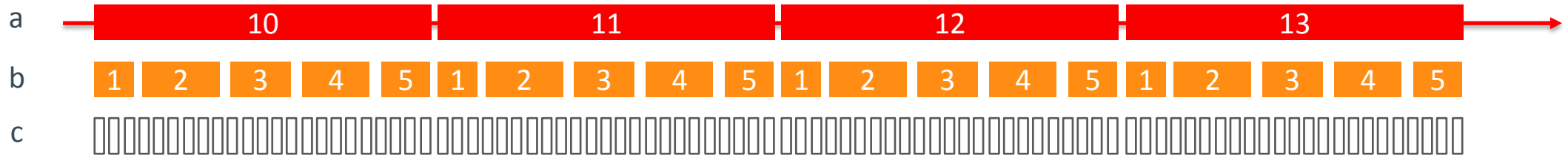
- Logical storage layout of index:



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)

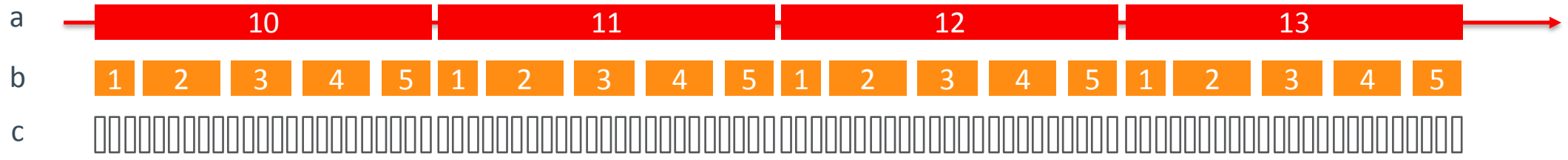


- Resulting range scan:

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 |
+-----------------------------------------------------------+---------+------+
```

Index Merge

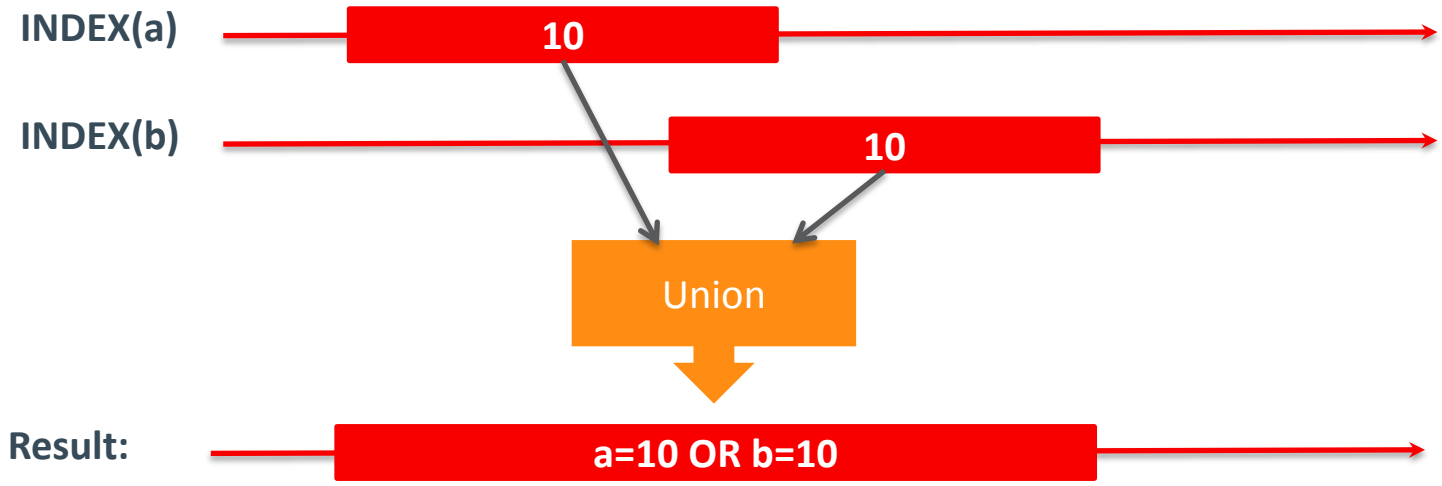
- Uses multiple indexes on the same table
- Implemented index merge strategies:
  - **Index Merge Union**
    - OR-ed conditions between different indexes
  - **Index Merge Intersection**
    - AND conditions between different indexes
  - **Index Merge Sort-Union**
    - OR-ed conditions where condition is a range

Index Merge Union

- Single index cannot handle ORed conditions on different columns
- Example:

  SELECT * FROM t1 WHERE a=10 OR b=10

- Index Merge Union:

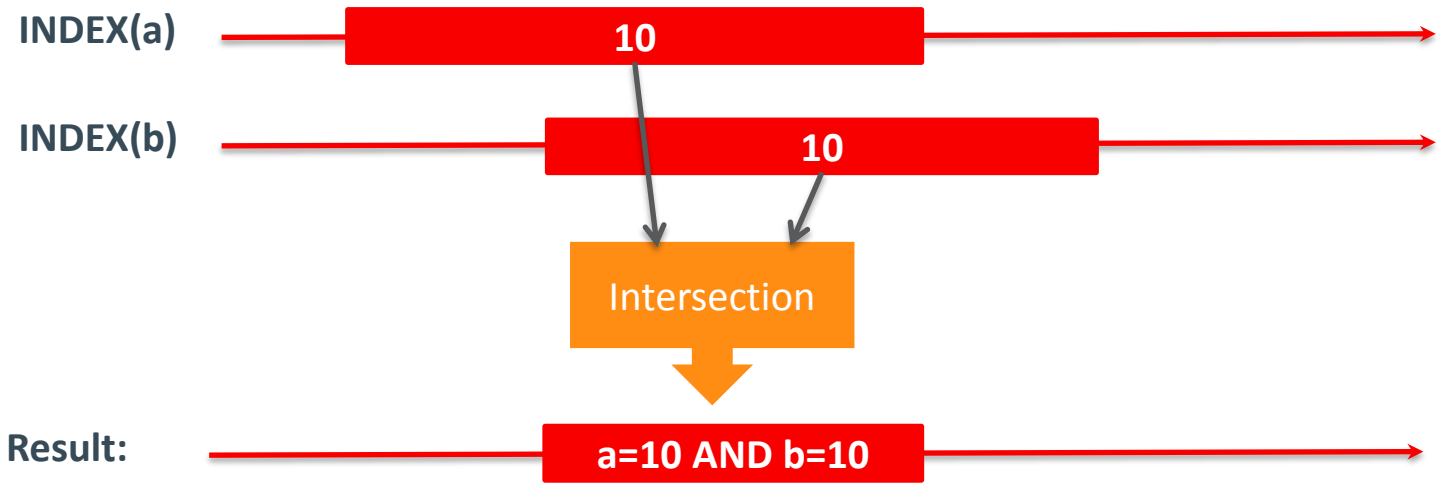


Index Merge Intersection

- Combine several indexes to reduce number of (or avoid) accesses to base table for ANDed conditions
- Example:

  SELECT * FROM t1 WHERE a=10 AND b=10

- Index Merge Intersection:



Index Merge Intersection: Example 1

SELECT COUNT(*) FROM lineitem
WHERE l_shipdate = '1997-05-01' AND l_commitdate = '1997-05-01';

id	select type	table	type	possible keys	key	key len	ref	rows	Extra
1	SIMPLE	lineitem	index_ merge	i_l_shipdate, i_l_commitdate	i_l_shipdate, i_l_commitdate	4,4	NULL	43	Using intersect (i_l_shipdate, i_l_commitdate); Using where; Using index

```
mysql> SELECT COUNT(*) FROM lineitem WHERE l_shipdate = '1997-05-01' …
...
1 row in set (0.02 sec)
mysql> SET optimizer_switch='index_merge_intersection=off';
mysql> SELECT COUNT(*) FROM lineitem WHERE l_shipdate = '1997-05-01' …
...
1 row in set (0.11 sec)
```

ORACLE

Index Merge Intersection: Example 2

Beware of low-selectivity indexes!

Low selectivity

SELECT count(*) FROM user
WHERE user_type=2 AND status=0 AND parent_id=0;

id	select type	table	type	possible keys	key	key len	ref	rows	Extra
1	SIMPLE	user	index_ merge	parent_id, status, user_type	user_type, status, parent_id	1,1,4	NULL	2511	Using intersect (user_type, status, parent_id); Using where; Using index

```
mysql> SELECT count(*) FROM user WHERE user_type=2 AND status=0 AND parent_id=0;
...
1 row in set (1.37 sec)
mysql> SELECT count(*) FROM user IGNORE INDEX (parent_id) WHERE user_type=2 …
...
1 row in set (0.18 sec)
```

Source: http://www.mysqlperformanceblog.com/2012/12/14/the-optimization-that-often-isnt-index-merge-intersection/

Index Merge Intersection: Example 2

Performance schema – index usage

```
mysql> TRUNCATE performance_schema.
    -> table_io_waits_summary_by_index_usage;

mysql> SELECT count(*) FROM user
    -> WHERE user_type=2 AND status=0
    -> AND parent_id=0;
...
1 row in set (1.37 sec)

mysql> SELECT object_name, index_name,
    -> count_read FROM performance_schema.
    -> table_io_waits_summary_by_index_usage
    -> WHERE object_name = 'users';
+-------------+------------+------------+
| object_name | index_name | count_read |
+-------------+------------+------------+
| users       | PRIMARY    |          0 |
| users       | parent_id  |    3936529 |
| users       | status     |     240103 |
| users       | user_type  |     237677 |
| users       | NULL       |          0 |
+-------------+------------+------------+
```

```
mysql> TRUNCATE performance_schema.
    -> table_io_waits_summary_by_index_usage;

mysql> SELECT count(*) FROM user
    -> IGNORE INDEX(parent_id)
    -> WHERE user_type=2 AND status=0
    -> AND parent_id=0;
...
1 row in set (0.18 sec)

mysql> SELECT object_name, index_name,
    -> count_read FROM performance_schema.
    -> table_io_waits_summary_by_index_usage
    -> WHERE object_name = 'users';
+-------------+------------+------------+
| object_name | index_name | count_read |
+-------------+------------+------------+
| users       | PRIMARY    |          0 |
| users       | parent_id  |          0 |
| users       | status     |     240103 |
| users       | user_type  |     237678 |
| users       | NULL       |      11814 |
+-------------+------------+------------+
```

MySQL 8.0: Index Merge Hints

- INDEX_MERGE(*table idx1, idx2, ...*)

```
SELECT /*+ INDEX_MERGE(users  user_type, status) */  count(*)
FROM users WHERE user_type=2 AND status=0 AND parent_id=0;
```

- NO_INDEX_MERGE(*table idx1, idx2, ...*)

```
SELECT /*+ NO_INDEX_MERGE(users  parent_id) */  count(*)
FROM users WHERE user_type=2 AND status=0 AND parent_id=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. Subqueries
6. Sorting
7. Influencing the optimizer

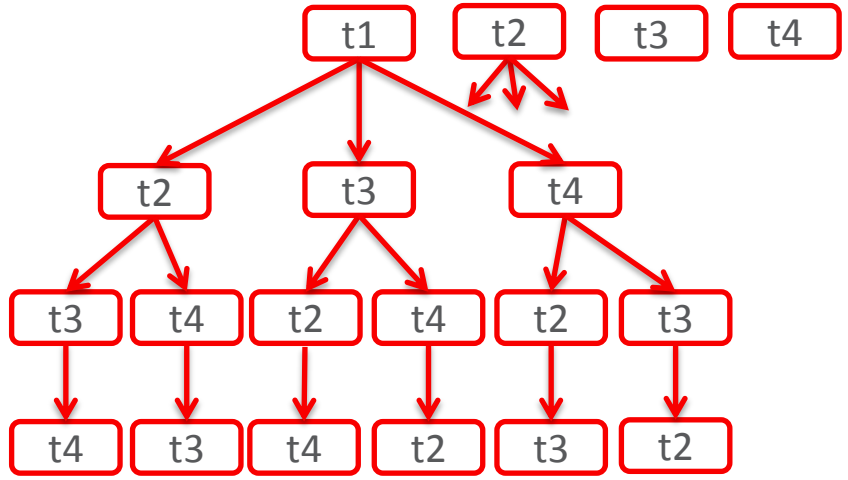
Join Optimizer

”Greedy search strategy”

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

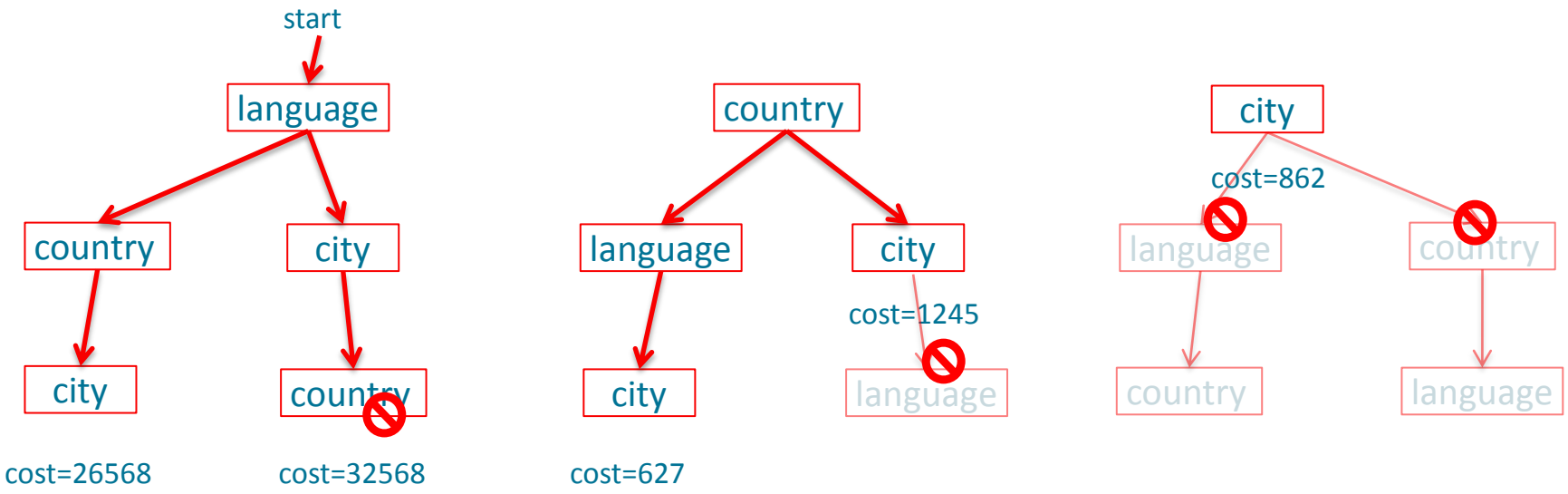
N! possible plans

- 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 AS capital, language.name
FROM city
  JOIN country ON city.country_id = country.country_id
  JOIN language ON country.country_id = language.country_id
WHERE city.city_id = country.capital
```



Join Optimizer

Example

EXPLAIN SELECT *
FROM customers JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < -1000 AND o_orderdate < '1993-01-01';

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	orders	ALL	i_o_orderdate, i_o_custkey	NULL	NULL	NULL	15000000	31.19	Using where
1	SIMPLE	customer	eq_ref	PRIMARY	PRIMARY	4	dbt3.orders.o_custkey	1	33.33	Using where

Join Optimizer

Change join order with STRAIGHT_JOIN

```
EXPLAIN SELECT STRAIGHT_JOIN *
FROM customer JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < -1000 AND o_orderdate < '1993-01-01';
```

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	33.33	Using where
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_custkey	5	dbt3. customer. c_custkey	15	31.19	Using where

Join Order

Performance

Join Order Hints

MySQL 8.0.1

EXPLAIN SELECT /*+ JOIN_ORDER(customer, orders) */ *
FROM customer JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < -1000 AND o_orderdate < '1993-01-01';

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	33.33	Using where
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_custkey	5	dbt3. customer. c_custkey	15	31.19	Using where

Alternatives with same effect for this query:

JOIN_PREFIX(customer) JOIN_SUFFIX(orders) JOIN_FIXED_ORDER()

DBT-3 Query 8

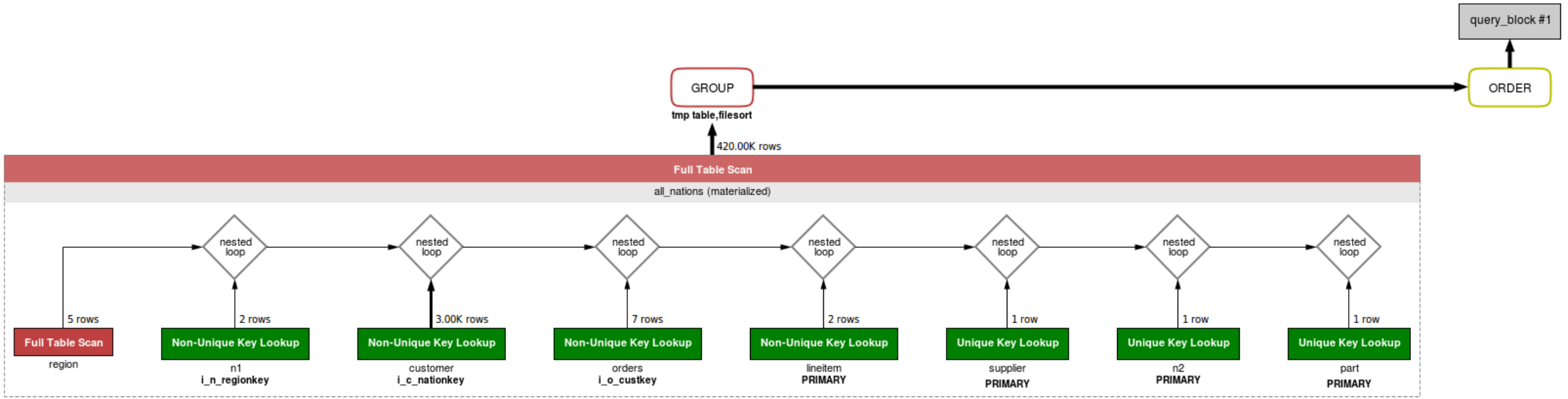
National Market Share Query

```
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;
```

DBT-3 Query 8

MySQL Workbench: Visual EXPLAIN (MySQL 5.6)

Execution time: 21 seconds



DBT-3 Query 8

Force early processing of high selectivity conditions

```
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;
```

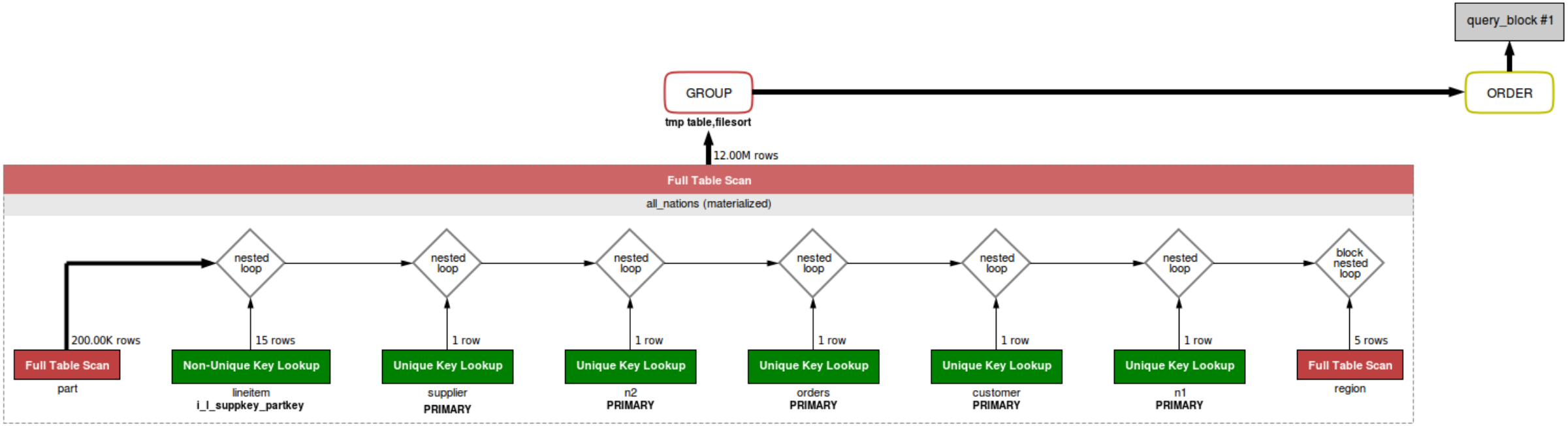
part before lineitem

Highest selectivity

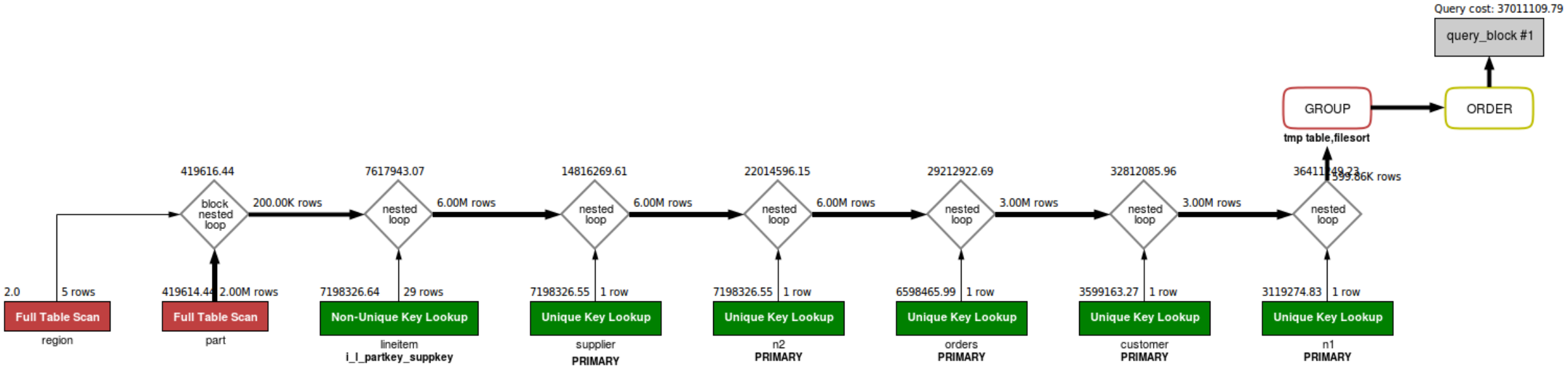
DBT-3 Query 8

Improved join order

Execution time: 3 seconds



MySQL 5.7: Improved join order



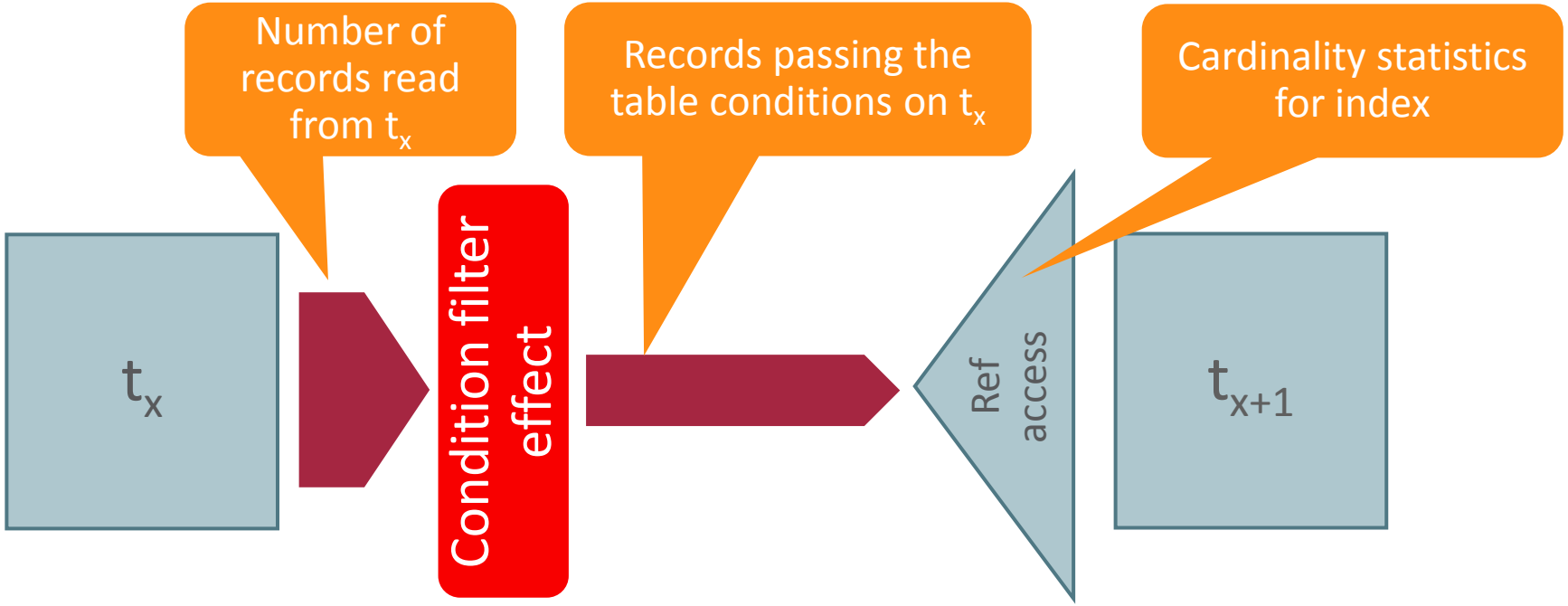
Improvements to Query 8 in MySQL 5.7:

- 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

Record and Cost Estimates for JOIN

Condition filter effect

- t_x JOIN t_{x+1}



- records(t_{x+1}) = records(t_x) * condition_filter_effect * records_per_key

How to Calculate Condition Filter Effect, step 1

```
SELECT office_name
FROM office JOIN employee
WHERE office.id = employee.office_id AND
      employee.name = 'John' AND
            employee.first_office_id <> office.id;
```

A condition contributes to the condition filter effect for a table only if:

- It references a field in the table
- It is **not** used by the access method
- It depends on an available value:
  - employee.name = 'John' ⟸ will always contribute to filter on employee
  - employee.first_office_id <> office.id; ⟸ depends on JOIN order

How to Calculate Condition Filter Effect, step 2


```
SELECT *
FROM office JOIN employee ON office.id = employee.office_id
WHERE office_name = 'San Francisco' AND
      employee.name = 'John' AND age > 21 AND
            hire_date BETWEEN '2014-01-01' AND '2014-06-01';
```

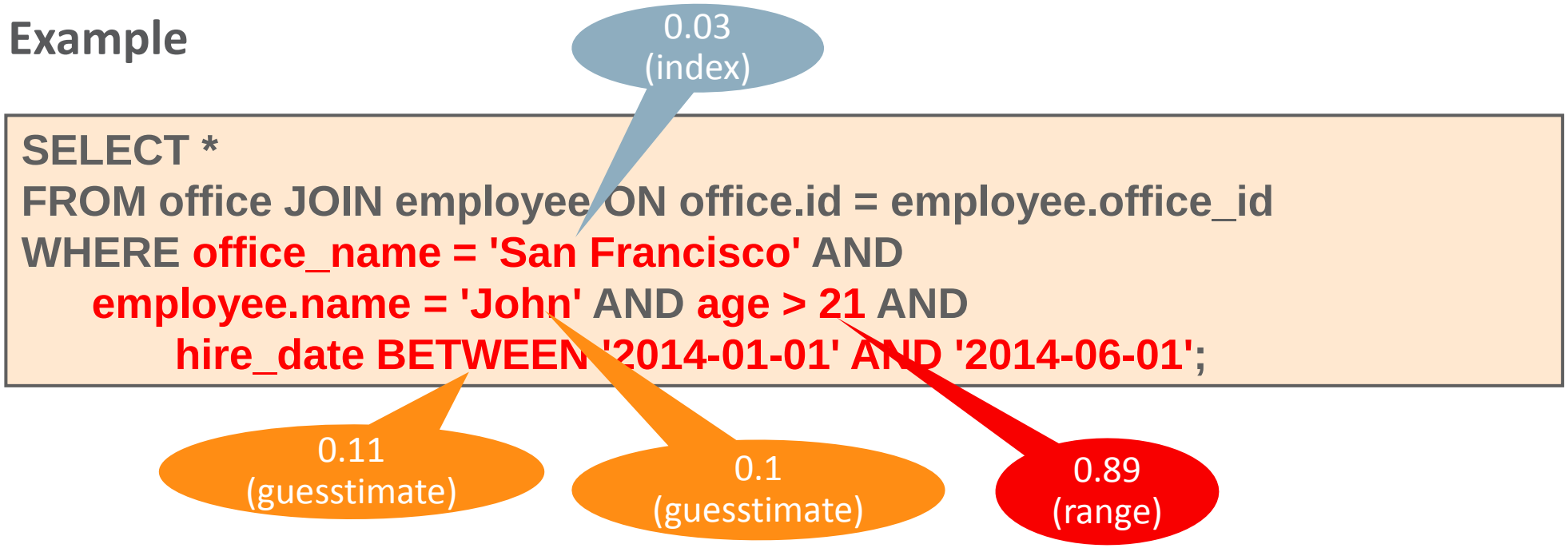
Filter estimate based on what is available:

1. Range estimate
2. Index statistics
3. Guesstimate

=	0.1
<=,<,>,>=	1/3
BETWEEN	1/9
NOT <op>	1 – SEL(<op>)
AND	P(A and B) = P(A) * P(B)
OR	P(A or B) = P(A) + P(B) – P(A and B)
…	…

Calculating Condition Filter Effect for Tables

Example



Condition filter effect for tables:

- office: 0.03
- employee: 0.1 * 0.11 * 0.89 ≈ 0.01

DBT-3 Query 21

Suppliers Who Kept Orders Waiting Query

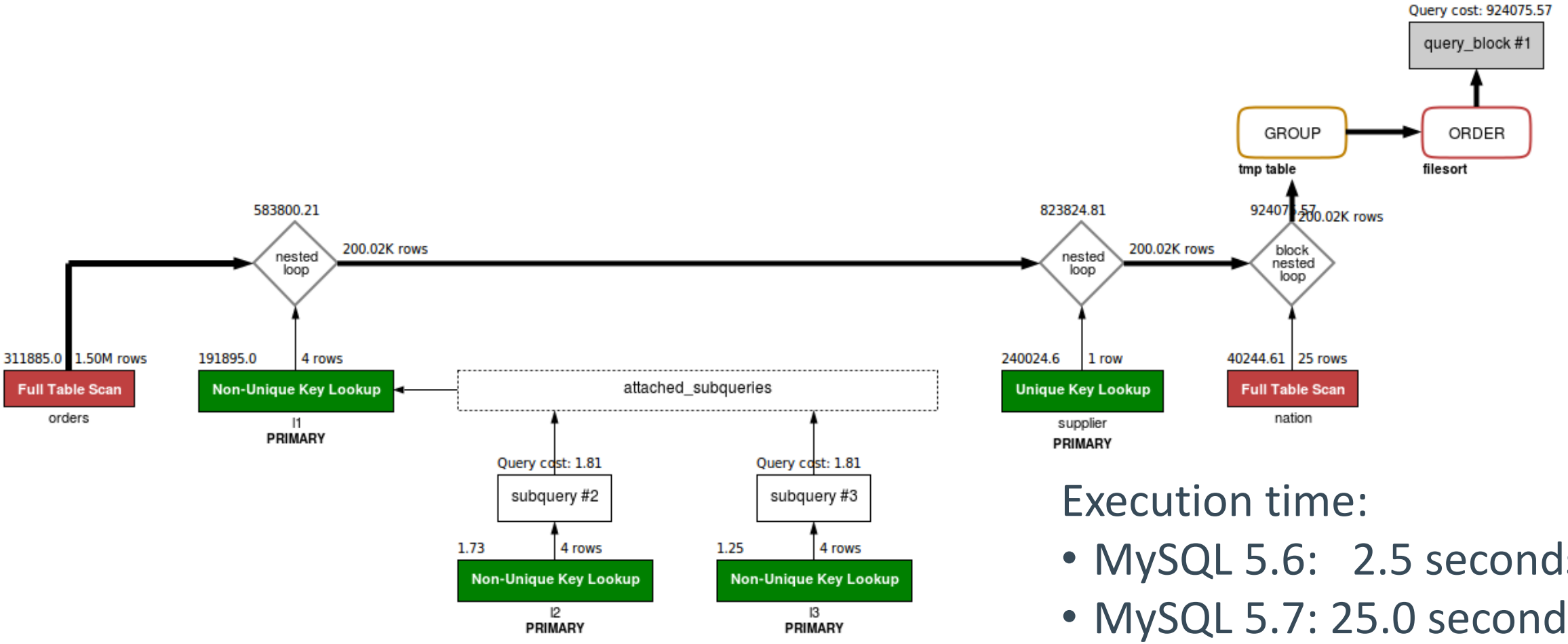
```
SELECT s_name, COUNT(*) AS numwait
FROM supplier
JOIN lineitem l1 ON s_suppkey = l1.l_suppkey
JOIN orders ON o_orderkey = l1.l_orderkey
JOIN nation ON s_nationkey = n_nationkey
WHERE o_orderstatus = 'F'
    AND l1.l_receiptdate > l1.l_commitdate
    AND EXISTS (
            SELECT * FROM lineitem l2
            WHERE l2.l_orderkey = l1.l_orderkey
                AND l2.l_suppkey <> l1.l_suppkey
     )
     AND NOT EXISTS (
            SELECT * FROM lineitem l3
            WHERE l3.l_orderkey = l1.l_orderkey
                AND l3.l_suppkey <> l1.l_suppkey
                AND l3.l_receiptdate > l3.l_commitdate
     )
    AND n_name = 'JAPAN'
GROUP BY s_name ORDER BY numwait DESC, s_name LIMIT 100;
```

Guesstimate: 0.10
Real value: 0.50

Guesstimate: 0.10
Real value: 0.04

DBT-3 Query 21

Query Plan, MySQL 5.7



Execution time:

- MySQL 5.6: 2.5 seconds
- MySQL 5.7: 25.0 seconds

DBT-3 Query 21

EXPLAIN

id	select_type	table	type	key	rows	filtered	Extra
1	PRIMARY	orders	ALL	NULL	15000000	10.00	Using where; Using temporary; Using filesort
1	PRIMARY	l1	ref	PRIMARY	4	33.33	Using where
1	PRIMARY	supplier	eq_ref	PRIMARY	1	100.00	Using index condition
1	PRIMARY	nation	ALL	NULL	25	4.00	Using join buffer (Block Nested Loop)
3	DEPENDENT SUBQUERY	l3	ref	PRIMARY	4	30.00	Using where
2	DEPENDENT SUBQUERY	l2	ref	PRIMARY	4	90.00	Using where

When filtering effect is overestimated

What to do?

1. Create an index
   - More accurate estimates
2. Add join order hints
   - STRAIGHT_JOIN
3. Disable Condition Filtering
   - **SET optimizer_switch='condition_fanout_filter=off'**
4. Wait for histograms
   - Working on it!

DBT-3 Query 21

Index on orders.o_orderstatus

id	select_type	table	type	key	rows	filtered	Extra
1	PRIMARY	nation	ALL	NULL	25	10.00	Using where; Using temporary; Using filesort
1	PRIMARY	supplier	ref	i_s_nationkey	400	100.00	NULL
1	PRIMARY	l1	ref	i_l_suppkey	600	33.33	Using where
1	PRIMARY	orders	eq_ref	PRIMARY	1	50.00	Using where
3	DEPENDENT SUBQUERY	l3	ref	PRIMARY	4	30.00	Using where
2	DEPENDENT SUBQUERY	l2	ref	PRIMARY	4	90.00	Using where

DBT-3 Query 21

Index on nation.n_name

id	select_type	table	type	key	rows	filtered	Extra
1	PRIMARY	nation	ref	i_n_name	1	100.00	Using where; Using temporary; Using filesort
1	PRIMARY	supplier	ref	i_s_nationkey	400	100.00	NULL
1	PRIMARY	l1	ref	i_l_suppkey	600	33.33	Using where
1	PRIMARY	orders	eq_ref	PRIMARY	1	10.00	Using where
3	DEPENDENT SUBQUERY	l3	ref	PRIMARY	4	30.00	Using where
2	DEPENDENT SUBQUERY	l2	ref	PRIMARY	4	90.00	Using where

DBT-3 Query 21

... nation STRAIGHT_JOIN orders ...

id	select_type	table	type	key	rows	filtered	Extra
1	PRIMARY	nation	ALL	NULL	25	10.00	Using where; Using temporary; Using filesort
1	PRIMARY	supplier	ref	i_s_nationkey	400	100.00	NULL
1	PRIMARY	l1	ref	i_l_suppkey	600	33.33	Using where
1	PRIMARY	orders	eq_ref	PRIMARY	1	10.00	Using where
3	DEPENDENT SUBQUERY	l3	ref	PRIMARY	4	30.00	Using where
2	DEPENDENT SUBQUERY	l2	ref	PRIMARY	4	90.00	Using where

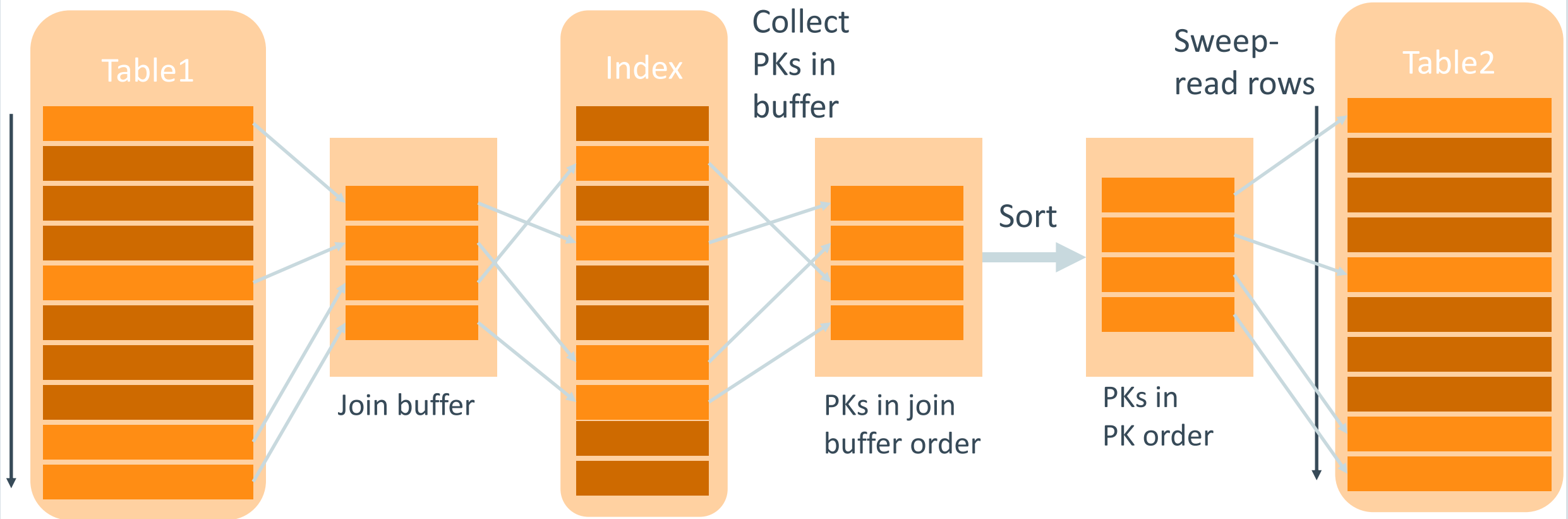
DBT-3 Query 21

SET optimizer_switch='condition_fanout_filter=off'

id	select type	table	type	key	rows	filtered	Extra
1	PRIMARY	nation	ALL	NULL	25	100.00	Using where; Using temporary; Using filesort
1	PRIMARY	supplier	ref	i_s_nationkey	400	100.00	NULL
1	PRIMARY	l1	ref	i_l_suppkey	600	100.00	Using where
1	PRIMARY	orders	eq_ref	PRIMARY	1	100.00	Using where
3	DEPENDENT SUBQUERY	l3	ref	PRIMARY	4	100.00	Using where
2	DEPENDENT SUBQUERY	l2	ref	PRIMARY	4	100.00	Using where

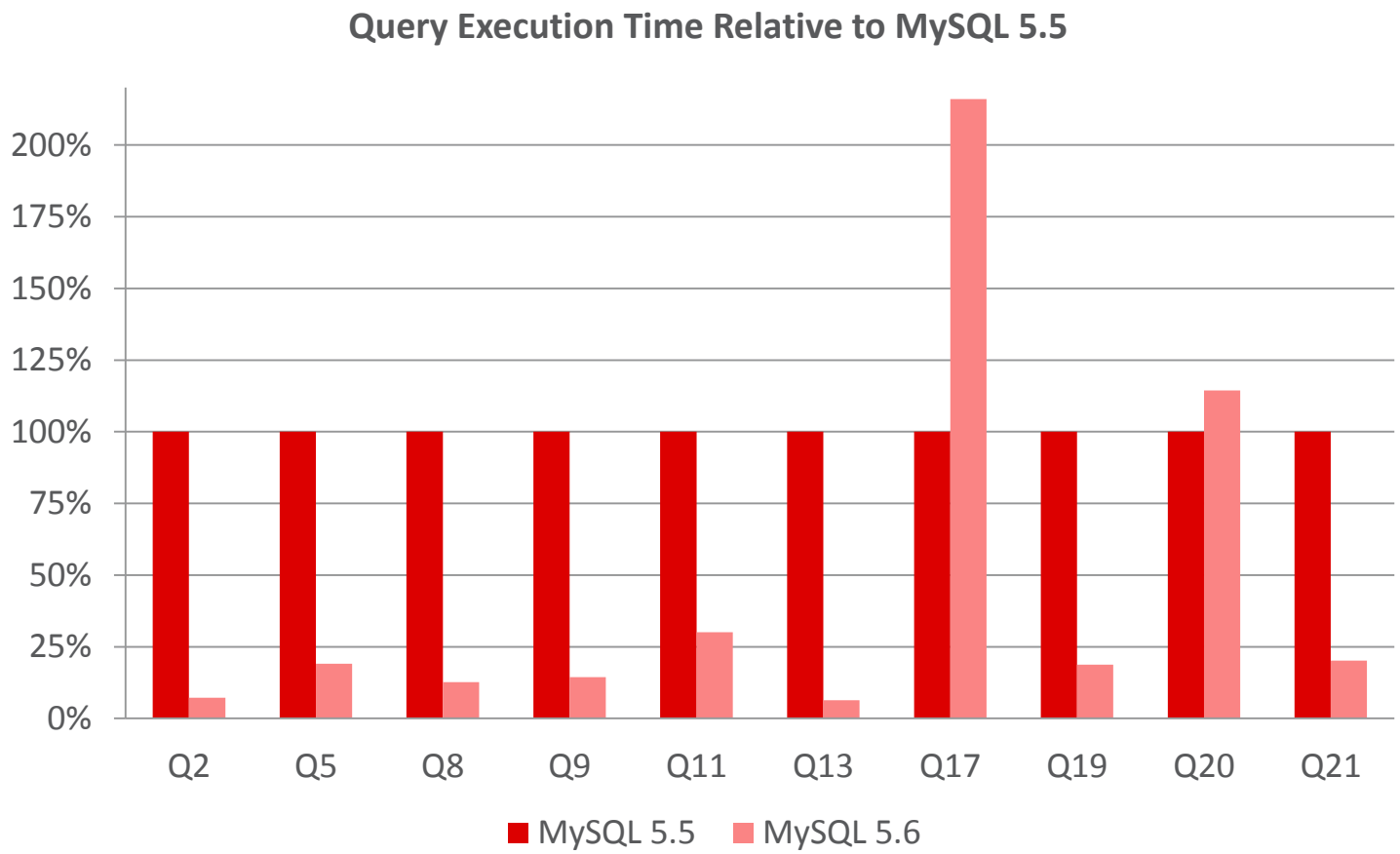
Batched Key Access (BKA)

Sequential disk access



MySQL 5.5 vs MySQL 5.6: Queries using BKA

- DBT-3, Scale 10 (23 GB)
  - innodb_buffer_pool_size= 1 GB (disk-bound)
  - join_buffer_size = 4 MB
  - optimizer_switch = 'batched_key_access=on, mrr_cost_based=off'



Batched Key Access

Usage

- Default: Off
- Force BKA on:

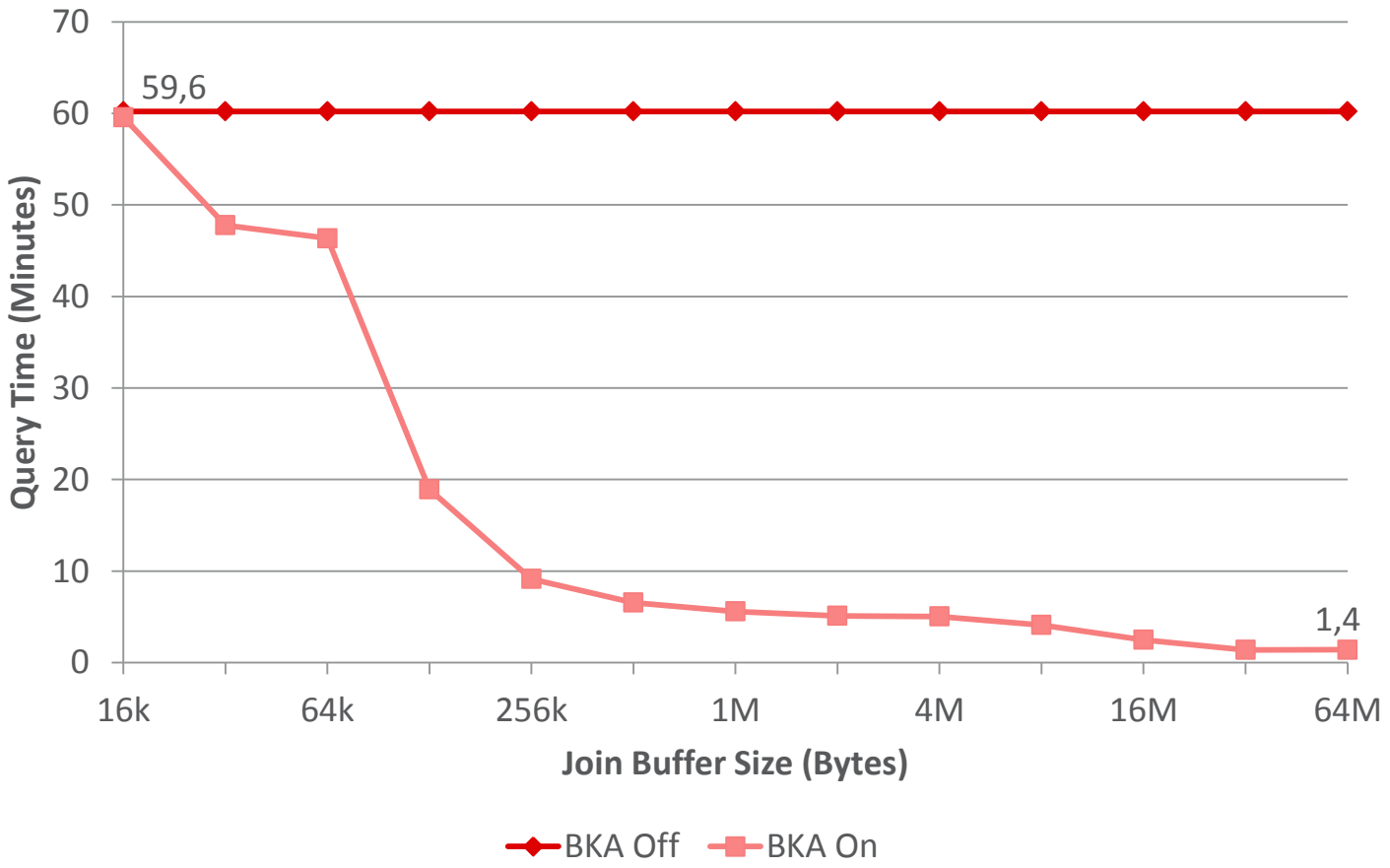
```
set optimizer_switch =
  'batched_key_access=on,mrr_cost_based=off';
```

- Configurable size for buffering keys to sort:

  `join_buffer_size` (Default: 256 kB)

Batched Key Access: Buffer Size Matters

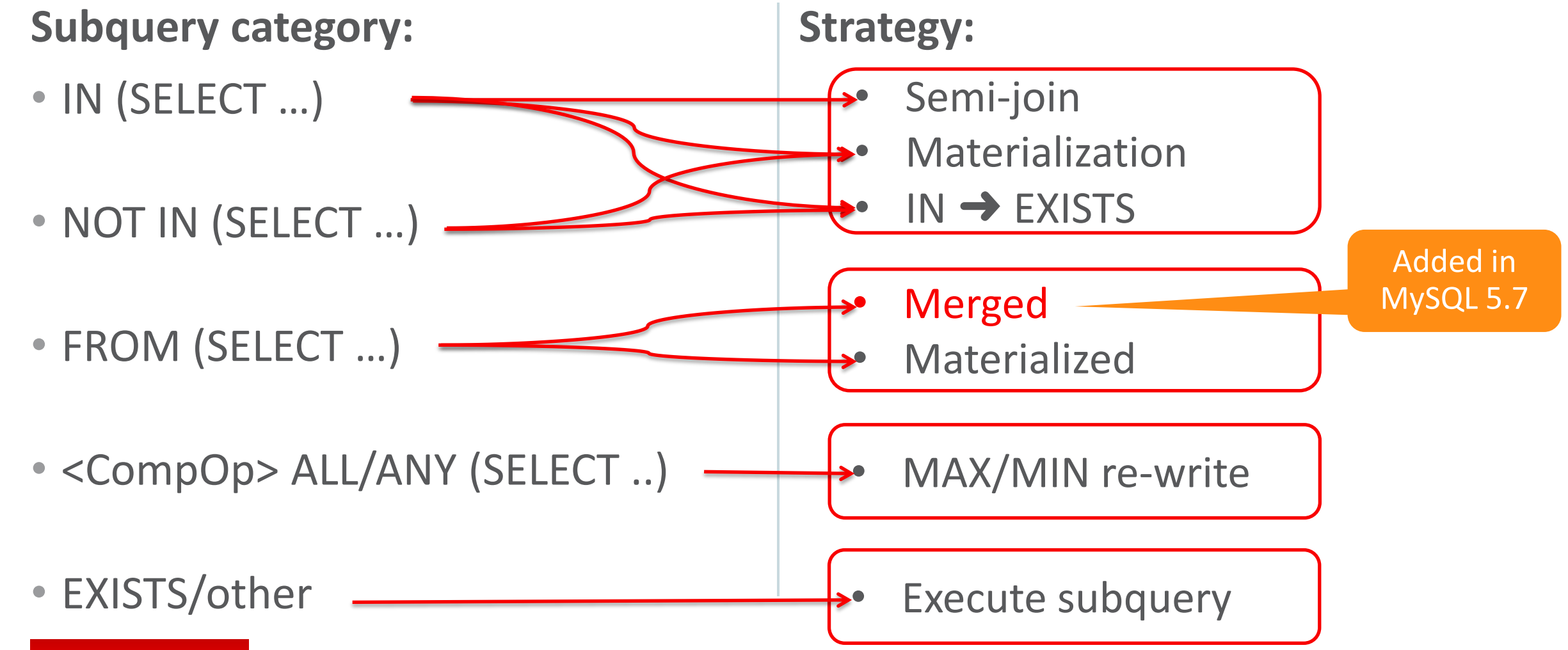
- DBT-3, Query 2
  Scale 10 (23 GB)
- innodb_buffer_pool_size= 1 GB (disk-bound)
- Varying join_buffer_size
- optimizer_switch = 'batched_key_access=on, mrr_cost_based=off'



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. Subqueries
6. Sorting
7. Influencing the optimizer

Overview of Subquery Optimizations



Traditional Optimization of IN Subqueries

IN ➔ EXISTS transformation

- Convert IN subquery to EXISTS subquery by “push-down” IN-equality to subquery:

```
SELECT title FROM film
WHERE film_id IN (SELECT film_id FROM actor WHERE name=“Bullock”)
```

```
SELECT title FROM film
WHERE EXISTS (SELECT 1 FROM actor
              WHERE name=“Bullock” AND film.film_id = actor.film_id)
```

- Benefit: subquery will evaluate fewer records
- Note: Special handling if pushed down expressions can be NULL

Semi-join

- Convert subquery to inner join, BUT
  - Need some way to remove duplicates
- Different strategies for duplicate removal:
  - **FirstMatch** (equivalent to IN→EXISTS execution)
  - **LooseScan** (index scan, skip duplicates)
  - Materialization: **MatLookup** (like subquery materialization), **MatScan** (materialized table is first in join order)
  - **Duplicate WeedOut** (insert result rows of semi-join query into temporary table with unique index; duplicate rows will be rejected. Any join order.)
- If duplicate removal is not necessary:
  - **Table pull-out**

Semi-join

Continued

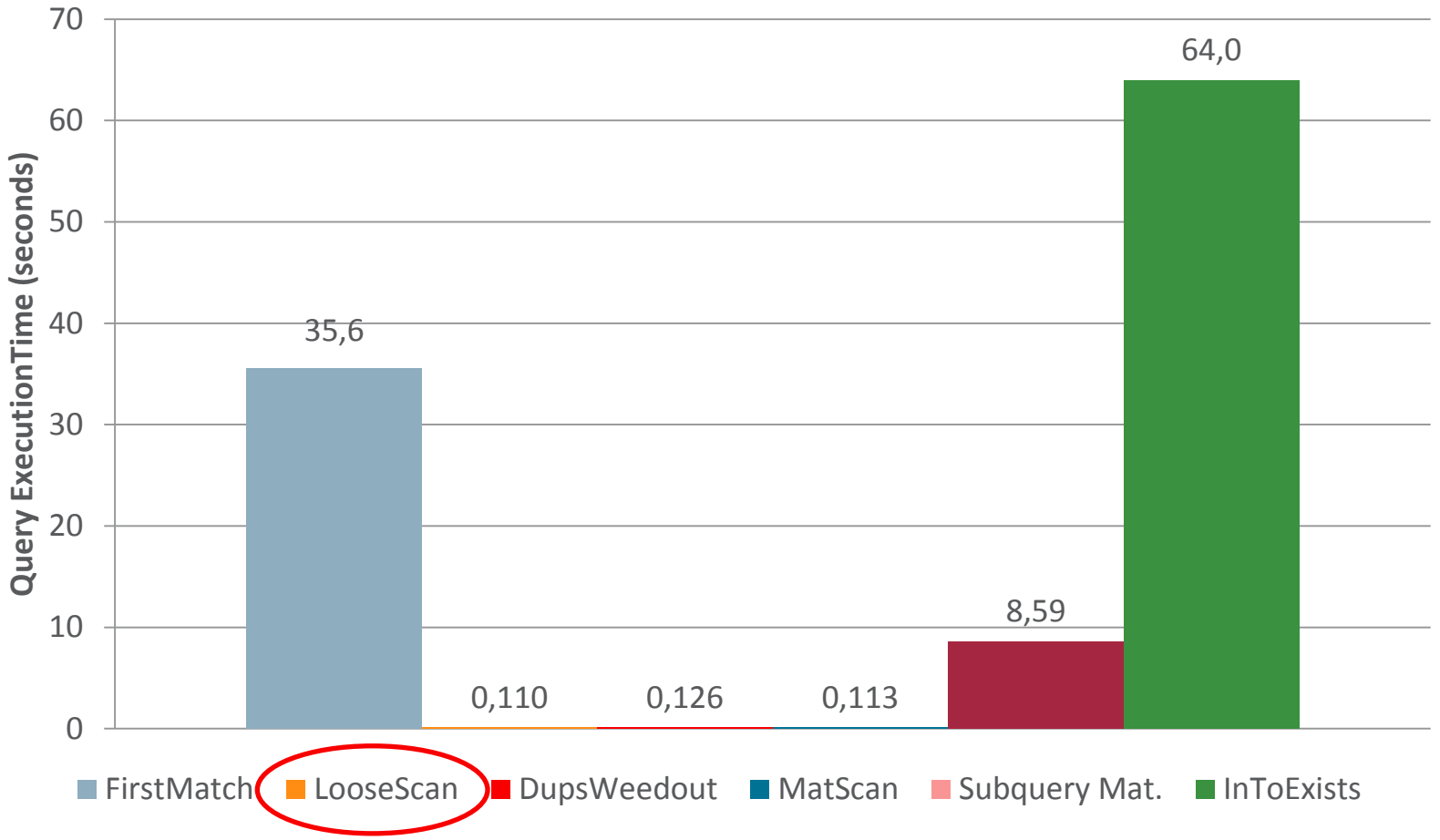
- Main advantage:
  - Opens up for more optimal ”join orders”.
  - Example:

```
SELECT o_orderdate, o_totalprice FROM orders
WHERE o_orderkey IN
        (SELECT l_orderkey FROM lineitem WHERE l_shipDate='1996-09-30');
```

    Will process less rows if starting with **lineitem** instead of **orders**

- Restriction:
  - Cannot use semi-join if subquery contains union or aggregation

MySQL 5.6: Semi-join: Example

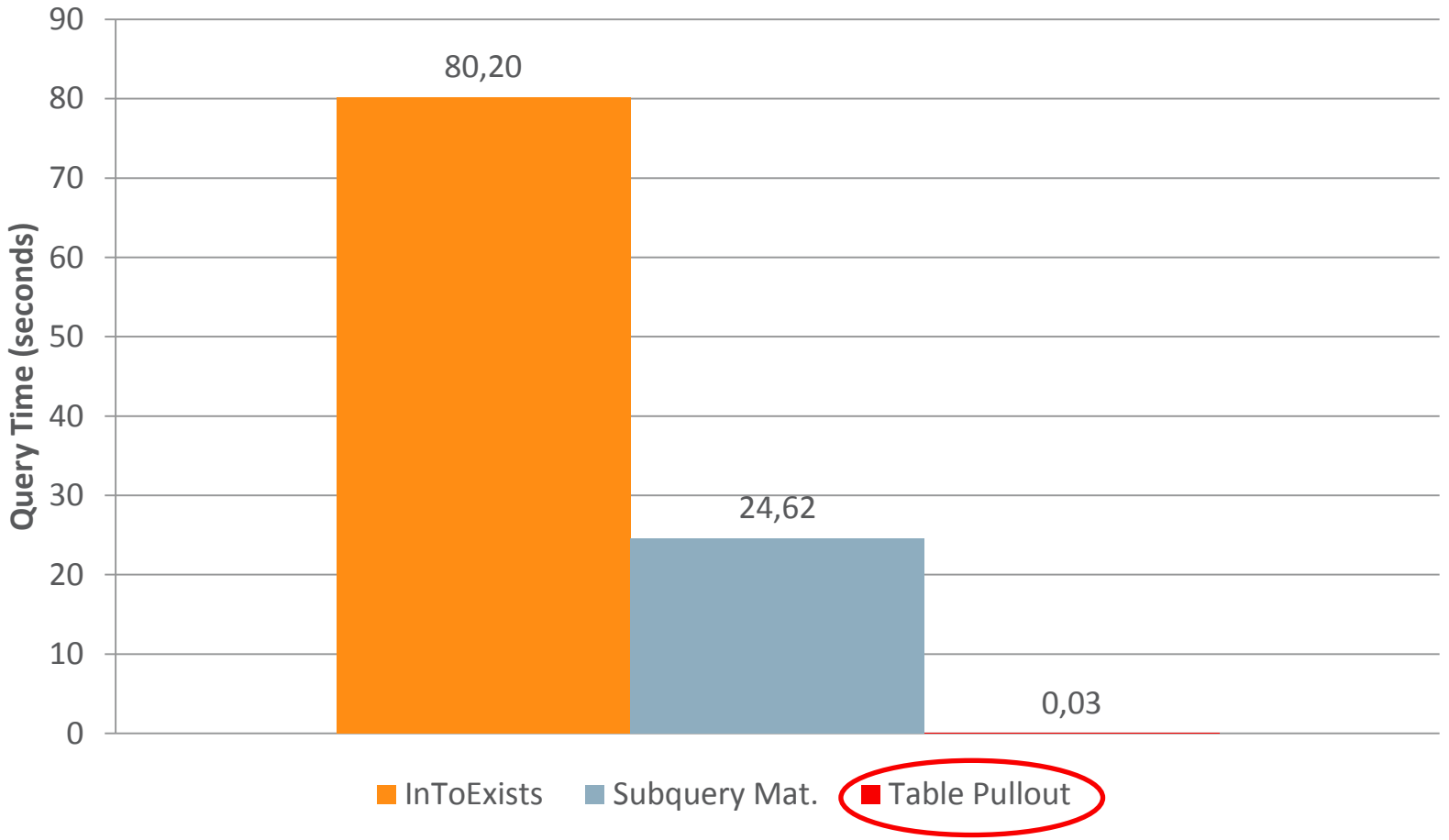



```
SELECT o_totalprice
FROM orders
WHERE o_orderkey IN
(SELECT l_orderkey
  FROM lineitem
  WHERE l_shipdate =
    '1996-09-30');
```

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 32 GB (CPU-bound)

MySQL 5.6: Semi-join: Example 2

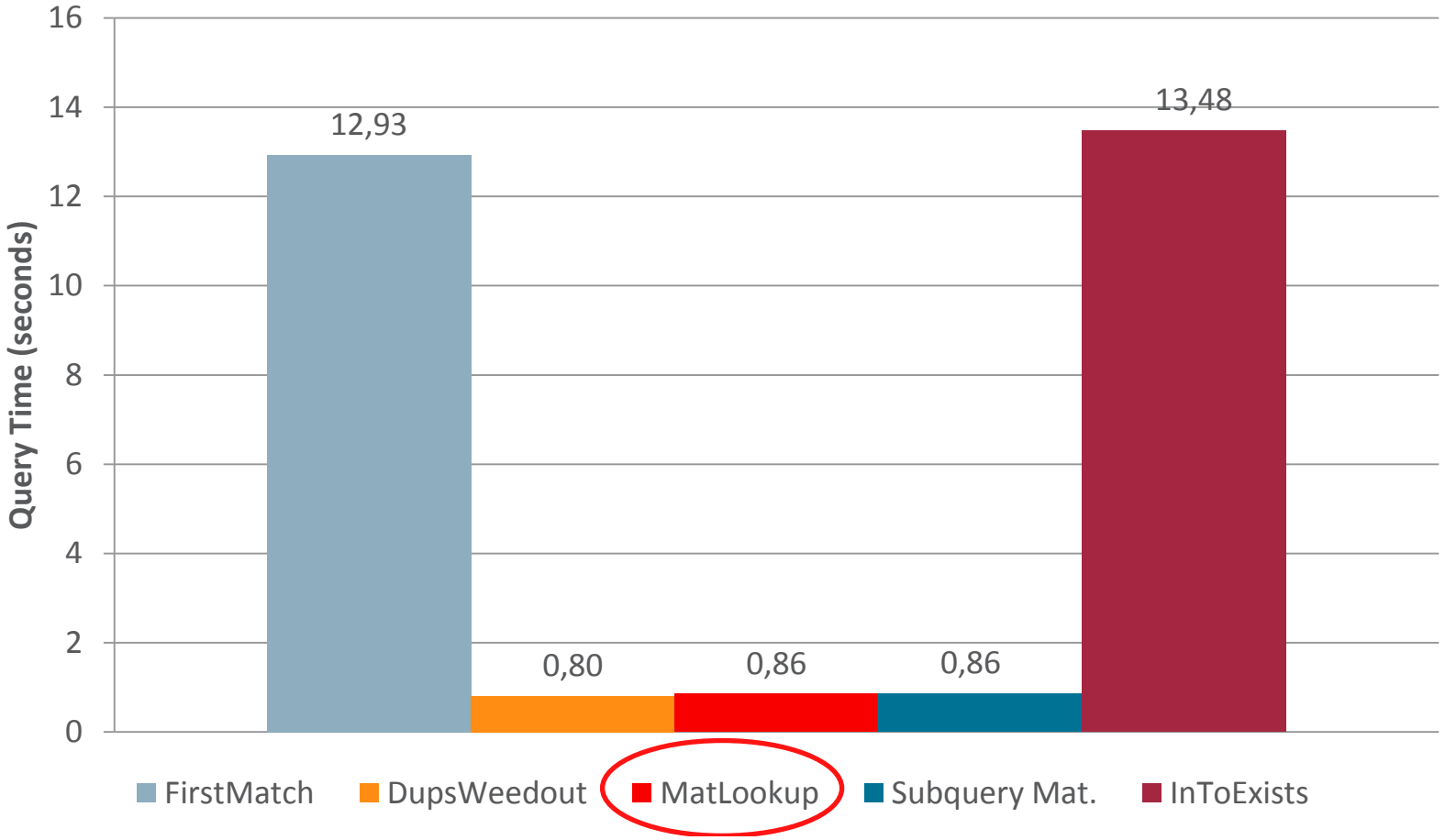



```
SELECT
 SUM(l_quantity*
   l_extendedprice)
FROM lineitem
WHERE l_orderkey IN
(SELECT o_orderkey
  FROM orders
  WHERE o_orderdate =
    '1996-09-30');
```

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 32 GB (CPU-bound)

MySQL 5.6: Semi-join: Example 3




```
SELECT s_name, s_address
FROM supplier
WHERE s_suppkey IN
(SELECT ps_suppkey
 FROM partsupp, part
 WHERE
      ps_partkey=p_partkey
  AND p_name LIKE 'grey%‘
  AND ps_availqty > 9990);
```

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 32 GB (CPU-bound)

Semi-join

FirstMatch

SELECT o_totalprice FROM orders WHERE o_orderkey IN
(SELECT l_orderkey FROM lineitem WHERE l_discount > 0.10);

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	orders	ALL	NULL	NULL	NULL	NULL	1500000	NULL
1	SIMPLE	lineitem	ref	PRIMARY, ...	PRIMARY	4	dbt3.orders.o_orderkey	2	Using where; FirstMatch (orders)

Semi-join
MatLookup

SELECT s_name, s_address FROM supplier WHERE s_suppkey IN
(SELECT ps_suppkey FROM partsupp, part WHERE ps_partkey=p_partkey
AND p_name LIKE 'grey%' AND ps_availqty > 9990);

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	supplier	ALL	PRIMARY	NULL	NULL	NULL	1000	Using where
1	SIMPLE	<suquery2>	eq_ref	<auto_key>	<auto_key>	4	...	1	NULL
2	MATERIALIZED	part	ALL	PRIMARY	NULL	NULL	NULL	20000	Using where
2	MATERIALIZED	partsupp	ref	PRIMARY	PRIMARY	4	...	2	Using where

Semi-join
LooseScan

SELECT o_totalprice FROM orders WHERE o_orderkey IN
(SELECT l_orderkey FROM lineitem WHERE l_shipdate = '1996-09-30');

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	lineitem	ref	PRIMARY, i_l_shipdate	i_l_shipdate	4	const	2532	Using index; Loosescan
1	SIMPLE	orders	eq_ref	PRIMARY	PRIMARY	4	...	1	NULL

Semi-join
Duplicate WeedOut

```
SELECT o_totalprice FROM orders WHERE o_orderkey IN
        (SELECT l_orderkey FROM lineitem
         WHERE l_shipdate BETWEEN '1996-09-24' AND '1996-09-30');
```

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	lineitem	range	PRIMARY, i_l_shipdate	i_l_shipdate	4	NULL	37124	Using where; Using index; Start temporary
1	SIMPLE	orders	eq_ref	PRIMARY	PRIMARY	4	...	1	End temporary

MySQL 5.7: SEMIJOIN Hints

- 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: SEMIJOIN Hints

- 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);
```

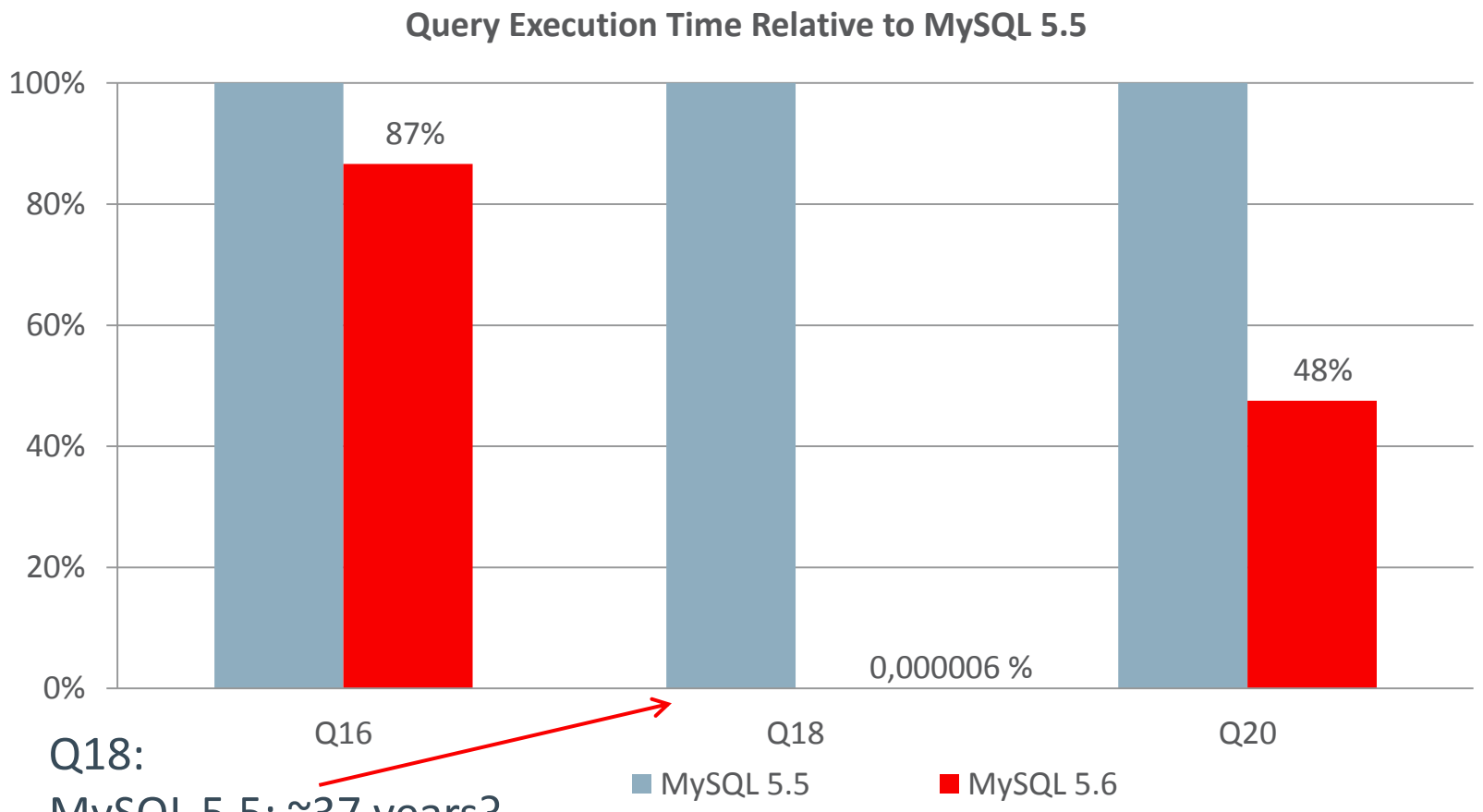
id	select_type	table	type	possible keys	key	key len	ref	rows	Extra
1	SIMPLE	t2	index	a	a	4	NULL	4	Using where; Using index
1	SIMPLE	<subquery2>	eq_ref	<auto_key>	<auto_key>	4	test.t2.a	1	NULL
2	MATERIALIZED	t3	index	a	a	4	NULL	3	Using index

Subquery Materialization

1. Execute subquery once and store result in a temporary table
   - Table has unique index for quick look-up and duplicate removal.
2. Execute outer query and check for matches in temporary table.

```
SELECT o_orderdate, o_totalprice
FROM orders
WHERE o_orderkey IN (
    SELECT l_orderkey
    FROM lineitem
    GROUP BY l_orderkey
    HAVING SUM(l_quantity) > 313
);
```

Comparing Subquery Materialization and IN ➜ EXISTS



DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 24 GB
(CPU-bound)

Q18:
MySQL 5.5: ~37 years?
MySQL 5.6: 69 seconds

Subquery Materialization

SELECT o_orderdate, o_totalprice FROM orders WHERE o_orderkey IN (
SELECT l_orderkey FROM lineitem GROUP BY l_orderkey HAVING SUM(l_quantity) > 313);

id	select_type	table	type	possible keys	key	key len	ref	rows	Extra
1	PRIMARY	orders	ALL	NULL	NULL	NULL	NULL	1500000	Using where
2	SUBQUERY	lineitem	index	PRIMARY, ...	PRIMARY	8	NULL	6001215	NULL

SELECT o_orderdate, o_totalprice FROM orders WHERE o_orderkey IN (SELECT /*+ SUBQUERY(INTOEXISTS)*/
l_orderkey FROM lineitem GROUP BY l_orderkey HAVING SUM(l_quantity) > 313);

id	select_type	table	type	possible keys	key	key len	ref	rows	Extra
1	PRIMARY	orders	ALL	NULL	NULL	NULL	NULL	1500000	Using where
2	DEPENDENT SUBQUERY	lineitem	index	PRIMARY, ...	PRIMARY	8	NULL	6001215	NULL

Derived Tables

- Subquery in FROM clause

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

- MySQL 5.6 and earlier: Executed separately and result stored in a temporary table (materialization)
- MySQL 5.6 and later: If useful, index will be created on the temporary table
- MySQL 5.7: Treat derived tables like views: May be merged with outer query block

Index on Materialized Derived Table

Added in MySQL 5.6

```
SELECT o_clerk, price - o_totalprice
FROM
        (SELECT l_orderkey, SUM( l_extendedprice * (1 - l_discount)) price
         FROM lineitem GROUP by l_orderkey) t1
JOIN
        (SELECT o_clerk, o_orderkey, o_totalprice
         FROM orders WHERE o_orderdate BETWEEN '1995-01-01' AND '1995-12-31') t2
ON t1.l_orderkey = t2.o_orderkey WHERE  t1.price > t2.o_totalprice;
```

Create index for join

DBT-3 Scale Factor 10:

- MySQL 5.5: ? months; MySQL 5.6: 2 minutes

Materialization of Derived Tables

EXPLAIN

```
mysql> explain select o_clerk, price - o_totalprice from
         (select l_orderkey, sum( l_extendedprice * (1 - l_discount)) price
            from lineitem group by l_orderkey) t1 join
         (select o_clerk, o_orderkey, o_totalprice from orders
            where o_orderdate between '1995-01-01' and '1995-12-31') t2
       on t1.l_orderkey = t2.o_orderkey where t1.price > t2.o_totalprice;
```

id	select_type	table	type	possible_keys	key	...
1	PRIMARY	<derived3>	ALL	NULL	NULL	...
1	PRIMARY	<derived2>	ref	<auto_key0>	<auto_key0>	...
3	DERIVED	orders	ALL	i_o_orderdate	NULL	...
2	DERIVED	lineitem	index	PRIMARY, …	PRIMARY	...

Merge Derived Table with Outer Query

Added in MySQL 5.7

```
mysql> explain select o_clerk, price - o_totalprice from
         (select l_orderkey, sum( l_extendedprice * (1 - l_discount)) price
            from lineitem group by l_orderkey) t1 join
         (select o_clerk, o_orderkey, o_totalprice from orders
            where o_orderdate between '1995-01-01' and '1995-12-31') t2
       on t1.l_orderkey = t2.o_orderkey where t1.price > t2.o_totalprice;

+----+-------------+------------+-------+--------------+------------+ ...
| id | select_type | table      | type  | possible_keys| key        | ...
+----+-------------+------------+-------+--------------+------------+ ...
|  1 | PRIMARY     | <derived2> | ALL   | NULL         | NULL       | ...
|  1 | PRIMARY     | orders     | eq_ref| PRIMARY, …   | PRIMARY    | ...
|  2 | DERIVED     | lineitem   | index | PRIMARY, …   | PRIMARY    | ...
+----+-------------+------------+-------+--------------+------------+ ...
```

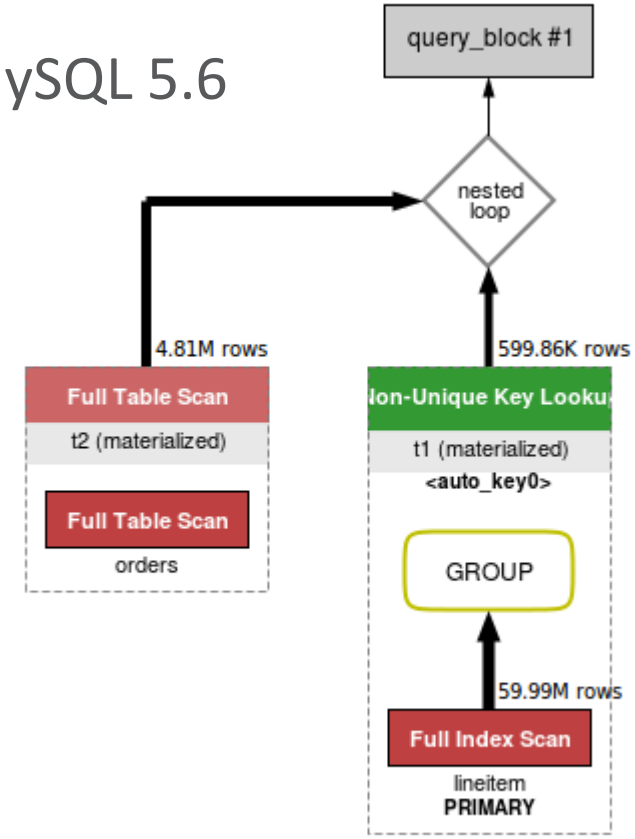
- No merge for derived tables with GROUP BY, DISTINCT, LIMIT, etc.

MySQL 5.7: 1.5 minutes (DBT-3 SF10)

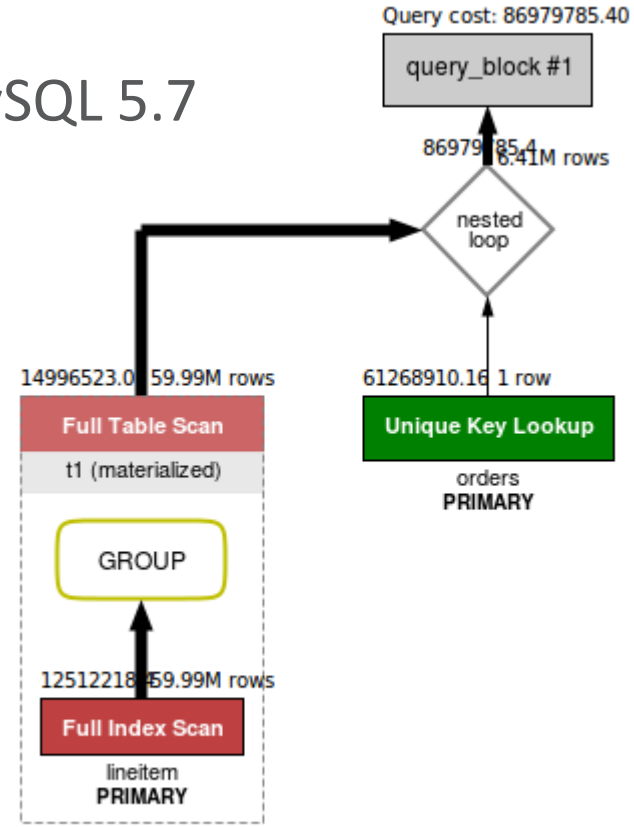
Merge Derived Table

Visual EXPLAIN

MySQL 5.6



MySQL 5.7

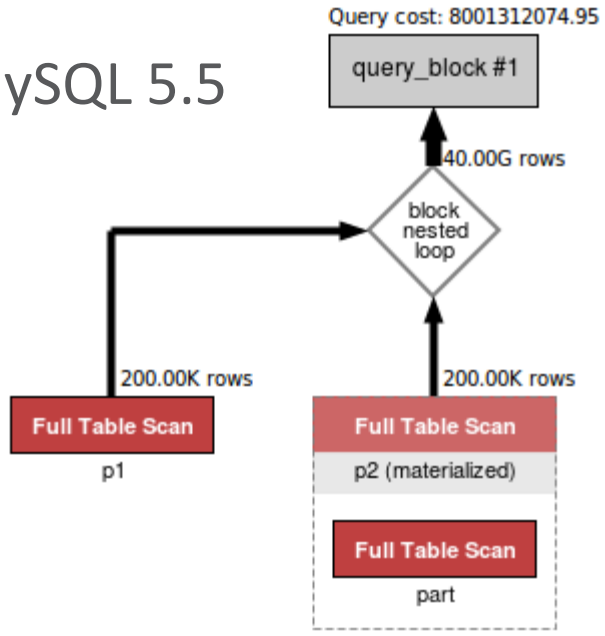


Derived Tables

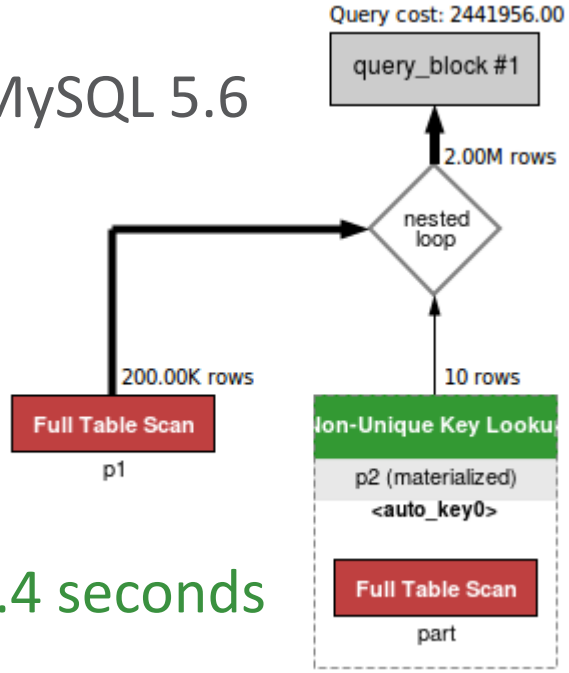
Undesirable Merge

```
SELECT * FROM part p1 JOIN
(SELECT * FROM part WHERE p_type LIKE '%STEEL%') p2 ON p1.p_name = p2.p_name
WHERE p1.p_type LIKE '%COPPER%';
```

MySQL 5.5

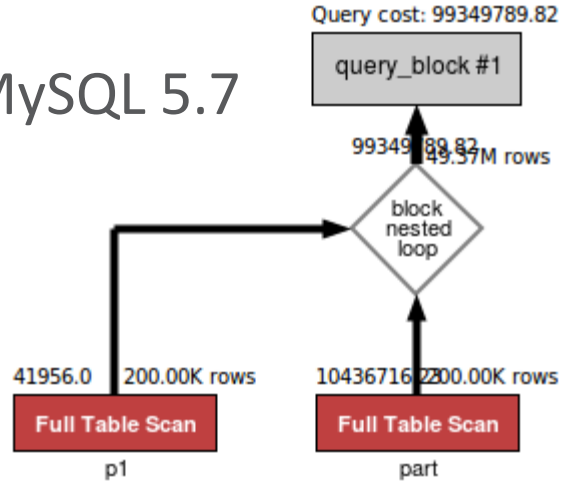


MySQL 5.6



0.4 seconds

MySQL 5.7



6 minutes

Hint: Merge/Materialize Derived Table or View

MySQL 8.0.1

- Derived tables/views are, if possible, merged into outer query
- NO_MERGE hint can be used to override default behavior:

```
SELECT /*+ NO_MERGE(dt) */ *
FROM t1 JOIN (SELECT x, y FROM t2) dt ON  t1.x = dt.x;
```

- MERGE hint will force a merge

```
SELECT /*+ MERGE(dt) */ *
FROM t1 JOIN (SELECT x, y FROM t2) dt ON  t1.x = dt.x;
```

- Can also use MERGE/NO_MERGE hints for views and CTE

```
SELECT /*+ NO_MERGE(v) */ * FROM t1 JOIN v ON t1.x = v.x;
```

Derived Tables

NO_MERGE hint

```
SELECT /*+ NO_MERGE(p2) */ * FROM part p1 JOIN
(SELECT * FROM part WHERE p_type LIKE '%STEEL%') p2 ON p1.p_name = p2.p_name
WHERE p1.p_type LIKE '%COPPER%';
```

id	select type	table	type	possible keys	key	key len	ref	rows	Extra
1	PRIMARY	p1	ALL	NULL	NULL	NULL	NULL	200000	Using where
1	PRIMARY	<derived2>	ref	<auto_key0>	<auto_key0>	58	dbt3.p1.p_name	10	NULL
2	DERIVED	part	ALL	NULL	NULL	NULL	NULL	200000	Using where

MySQL 8.0: Common Table Expressions

- Better readability:

```
WITH
t1 AS (SELECT l_orderkey, SUM(l_extendedprice * (1 - l_discount)) price
       FROM lineitem GROUP by l_orderkey),
t2 AS (SELECT o_clerk, o_orderkey, o_totalprice FROM orders
       WHERE o_orderdate BETWEEN '1995-01-01' AND '1995-12-31')
SELECT o_clerk, price - o_totalprice
FROM t1 JOIN t2 ON t1.l_orderkey = t2.o_orderkey
WHERE  t1.price > t2.o_totalprice;
```

- Can be referenced multiple times
  - Only materialized once
- Can refer to other CTEs

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. Subqueries
6. Sorting
7. 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.

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)
```

Aggregate Queries

GROUP BY

- General solution:
  1. Sort the rows and insert into a new temporary table so that all rows from each group are consecutive
  2. Scan temporary table and compute aggregated result
- Optimizations:
  - Use an index which is ordered on grouping column
    - Requires that all GROUP BY columns are from same index
  - Loose index scan

Aggregate Queries

Examples

SELECT SUM(o_totalprice) FROM orders GROUP BY o_clerk;

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

SELECT p_name, AVG(l_quantity) FROM lineitem JOIN part ON l_partkey=p_partkey GROUP BY p_partkey;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	part	index	PRIMARY	PRIMARY	4	NULL	2000000	NULL
1	SIMPLE	lineitem	ref	i_l_partkey	i_l_partkey	5	...	29	NULL

Loose Index Scan

- Optimization for GROUP BY and DISTINCT:

  SELECT a, b FROM t1 GROUP BY a, b;
  SELECT DISTINCT a, b FROM t1;
  SELECT a, MIN(b) FROM t1 GROUP BY a;

- GROUP BY/DISTINCT must be on the prefix of the index

Loose Index Scan

Example

SELECT a, MIN(b) FROM t1 GROUP BY a;

id	select type	table	type	possible keys	key	key len	ref	rows	extra
1	SIMPLE	t1	range	abc	abc	10	NULL	11	Using index for group-by

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. Subqueries
6. Sorting
7. 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

Optimizer Hints

Future

- New hints in 8.0
  - Enable/disable merge of views and derived tables:
    - MERGE(*derived_table*) NO_MERGE(*derived_table*)
  - Join order
    - JOIN_ORDER(*tables*) JOIN_PREFIX(*tables)* JOIN_SUFFIX(*tables*) JOIN_FIXED_ORDER()
  - Force/ignore index_merge alternatives
    - INDEX_MERGE(*table indexes*) NO_INDEX_MERGE(*table indexes*)
- Hints we consider to add
  - Reimplement index hints using the new syntax
  - Temporarily set session variables for the duration of the query

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

MySQL 5.7: Adjustable Cost Constants

Use with caution!

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	rows	filtered	Extra
1	SIMPLE	orders	ALL	i_o_orderdate	NULL	NULL	15000000	29.93	Using where

UPDATE mysql.engine_cost SET cost_value=0.25 ← Default: 1.0
WHERE cost_name='memory_block_read_cost';

FLUSH COST_CONSTANTS; ← Make server read new cost constants

MySQL 5.7: Adjustable Cost Constants

Improved plan

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	rows	filtered	Extra
1	SIMPLE	orders	range	i_o_orderdate	i_o_orderdate	4	4489990	100.00	Using index condition

Note:

- Heuristic used: If table is smaller than 20% of database buffer, all pages are in memory
- Only new connections will see updated cost constants

MySQL 8.0: Disk vs memory access

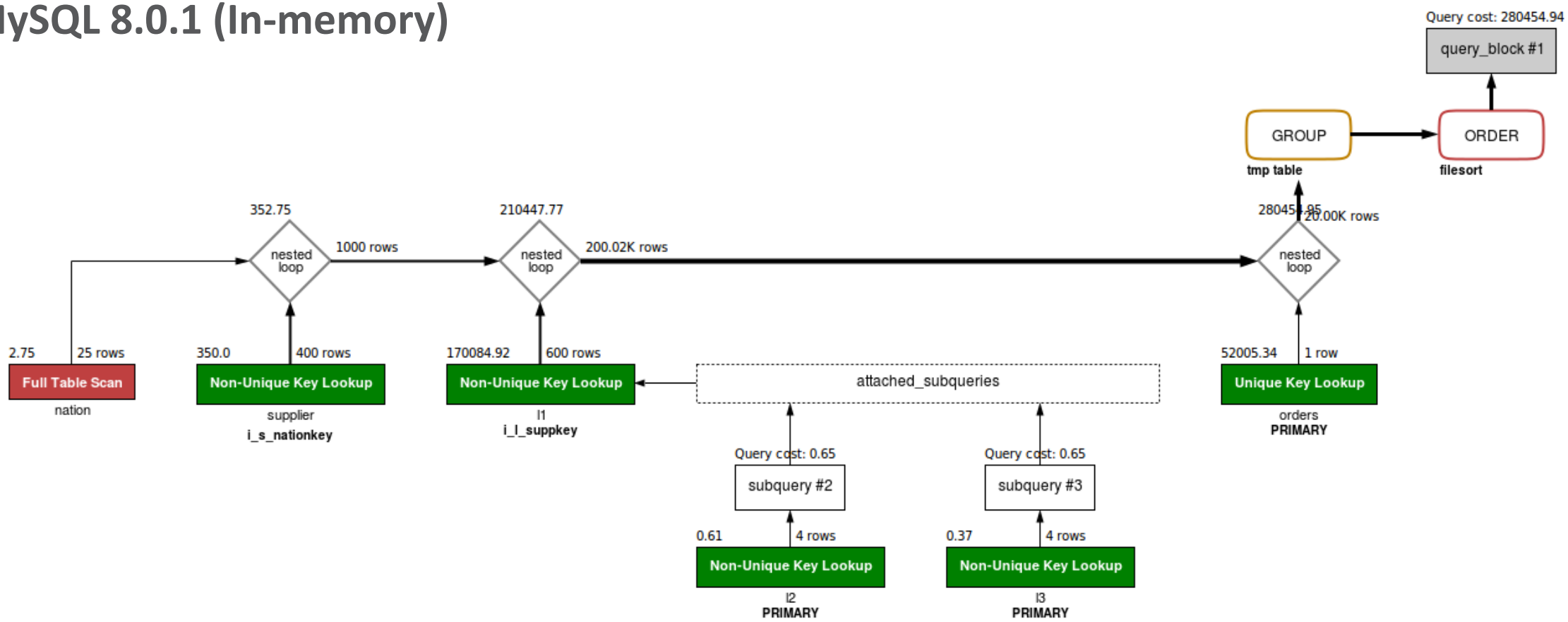
- New defaults for const constants:

Cost	MySQL 5.7	MySQL 8.0
Read a random disk page	1.0	1.0
Read a data page from memory buffer	1.0	0.25
Evaluate query condition	0.2	0.1
Compare keys/records	0.1	0.05

- InnoDB reports for each table/index percentage of data cached in buffer pool
- Note: Query plan may change between executions

DBT-3 Query 21

MySQL 8.0.1 (In-memory)



More information

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

Optimizer Related Presentations at Percona Live 2017

- MySQL 8.0 Optimizer Guide
  - Morgan Tocker, Monday 1:30pm
- MySQL 8.0: What is New in the Optimizer?
  - Manyi Lu, Tuesday 1:20pm
- Cookbook for Creating INDEXes - All about Indexing
  - Rick James, Wednesday 11:10pm
- Recursive Query Throwdown in MySQL 8
  - Bill Karwin, Thursday 12:50pm
- Meet the MySQL Server Development Team
  - BOF, ???

Safe Harbor Statement

The preceding 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.

Integrated Cloud

Applications & Platform Services


ORACLE®