
ORACLE®

How to Take Advantage of Optimizer Improvements in MySQL 8.0

Norvald H. Ryeng
Software Development Senior Manager
MySQL Optimizer Team
October, 2018

ORACLE OPEN WORLD

ORACLE

Safe Harbor Statement

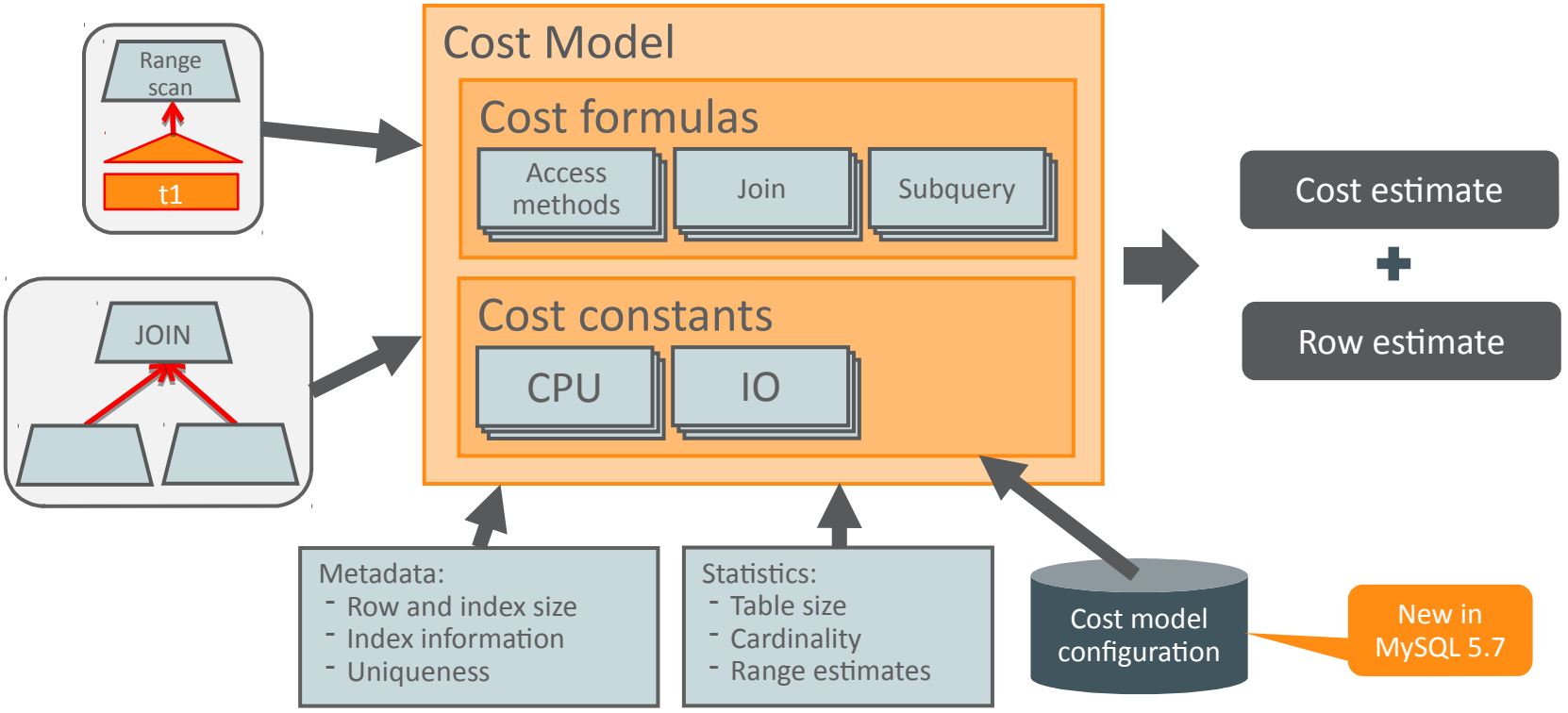
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development,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

Program Agenda

1. Cost model improvements
2. Histograms
3. Optimizer hints
4. Improved performance with CTEs
5. Improvements in query execution

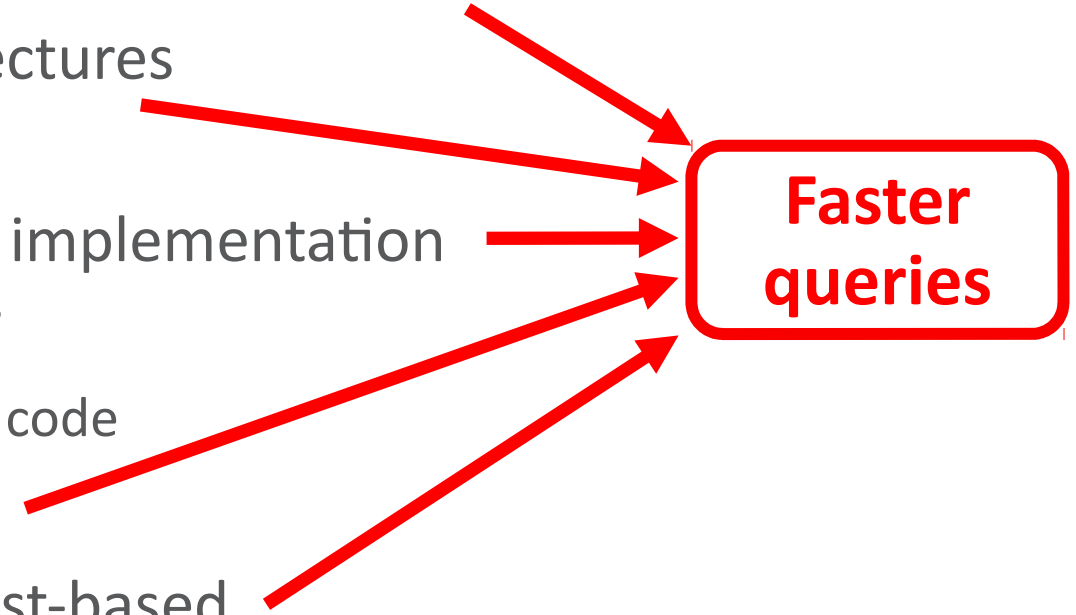
Cost Model Improvements

Optimizer Cost Model



Motivation for Improving the MySQL Cost Model

- Produce more correct cost estimates
  - Better decisions by the optimizer should improve performance
- Adapt to new hardware architectures
  - SSD, larger memories, caches
- More maintainable cost model implementation
  - Avoid hard coded “cost constants”
  - Refactoring of existing cost model code
- Configurable and tunable
- Make more of the optimizer cost-based

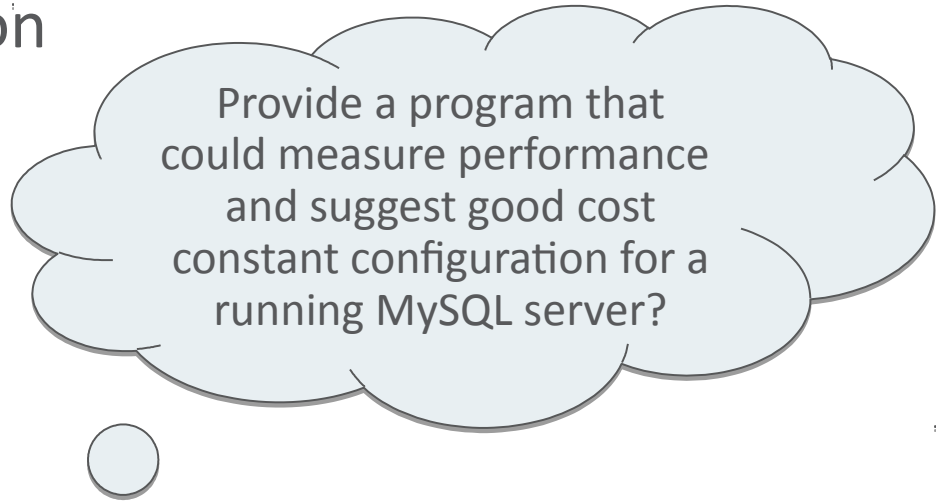


New Storage Technologies

- Time to do a table scan of 10 million records:

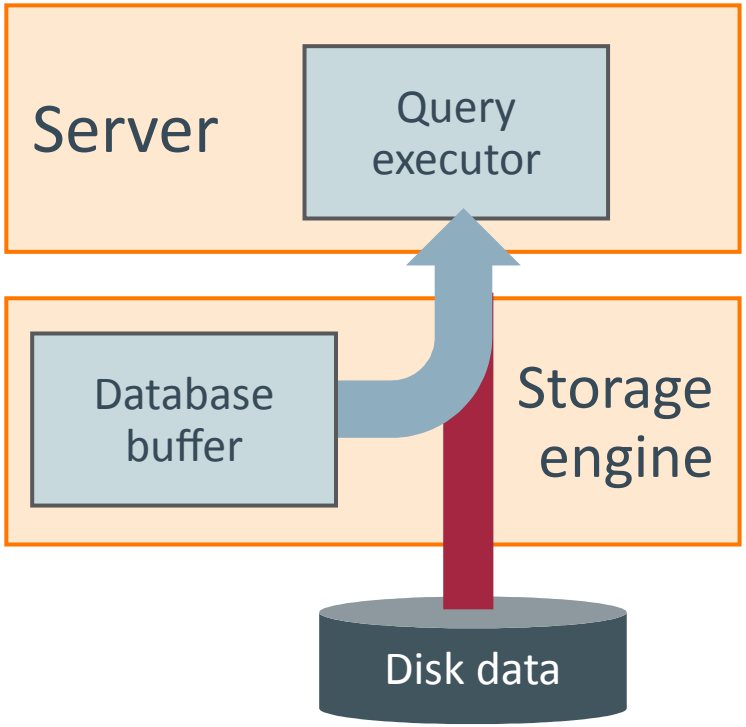
Memory	5 s
SSD	20 - 146 s
Hard disk	32 - 1465 s

- Adjust cost model to support different storage technologies
- Provide configurable cost constants for different storage technologies



Memory Buffer Aware Cost Estimates

- Storage engines:
  - Estimate for how much of data and indexes are in a memory buffer
  - Estimate for hit rate for memory buffer
- Optimizer cost model:
  - Take into account whether data is already in memory or need to be read from disk



Disk vs. Memory Access

- New defaults for const constants:

Same cost constant

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/rows	0.1	0.05

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

DBT-3 Query 8

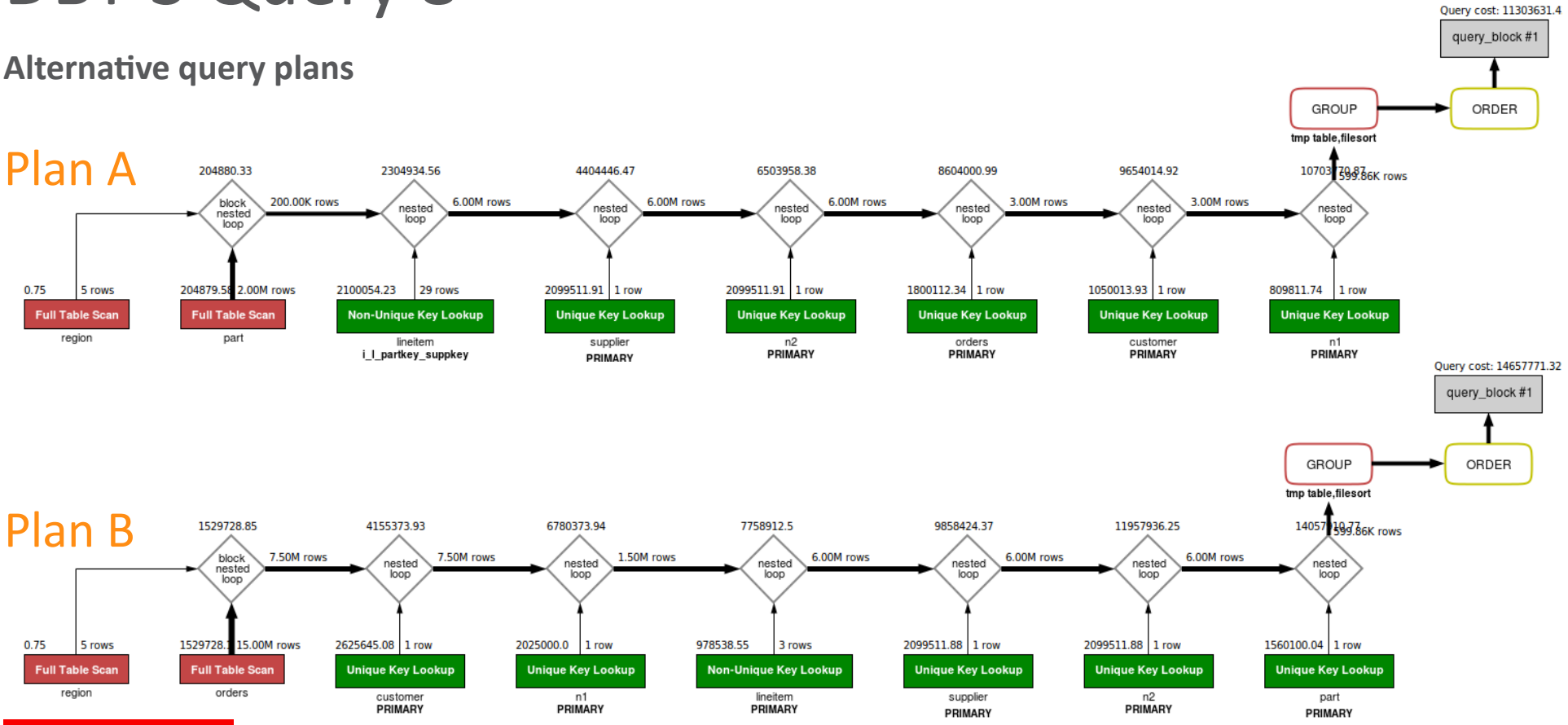
National market share

```
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;
```

High selectivity predicate

DBT-3 Query 8

Alternative query plans



DBT-3 Query 8

Execution time (8.0.3)

	In-memory	Disk-bound
Plan A	**5.8 secs**	9 min 47 secs
Plan B	77.5 secs	**3 min 49 secs**

Selected plan

	In-memory	Disk-bound
MySQL 5.6	Plan B	
MySQL 5.7	Plan A	
MySQL 8.0	Plan A	Plan B

How to Configure your Own Cost Constants

UPDATE *mysql.engine_cost* SET ***cost_value*=0.1** ← Default 0.25
WHERE *cost_name*='**memory_block_read_cost**';

FLUSH OPTIMIZER_COSTS; ← Make optimizer read new cost constants

Note:

- Only new connectons will see updated cost constants

Future ideas:

- Create tool to tune cost constants to specific hardware
- Use machine learning to find optimal values for specific user load

Histograms

Histograms

- Provides the optimizer with information about column value distribution
- To create/recalculate histogram for a column:

  ANALYZE TABLE *table* **UPDATE HISTOGRAM ON** *column* **WITH** *n* **BUCKETS**;

- May use sampling
  - Sample size is based on available memory (histogram_generation_max_mem_size)
- Automatically chooses between two histogram types:
  - Singleton: One value per bucket
  - Equi-height: Multiple values per bucket
  - Max 1024 buckets

Histograms Example

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

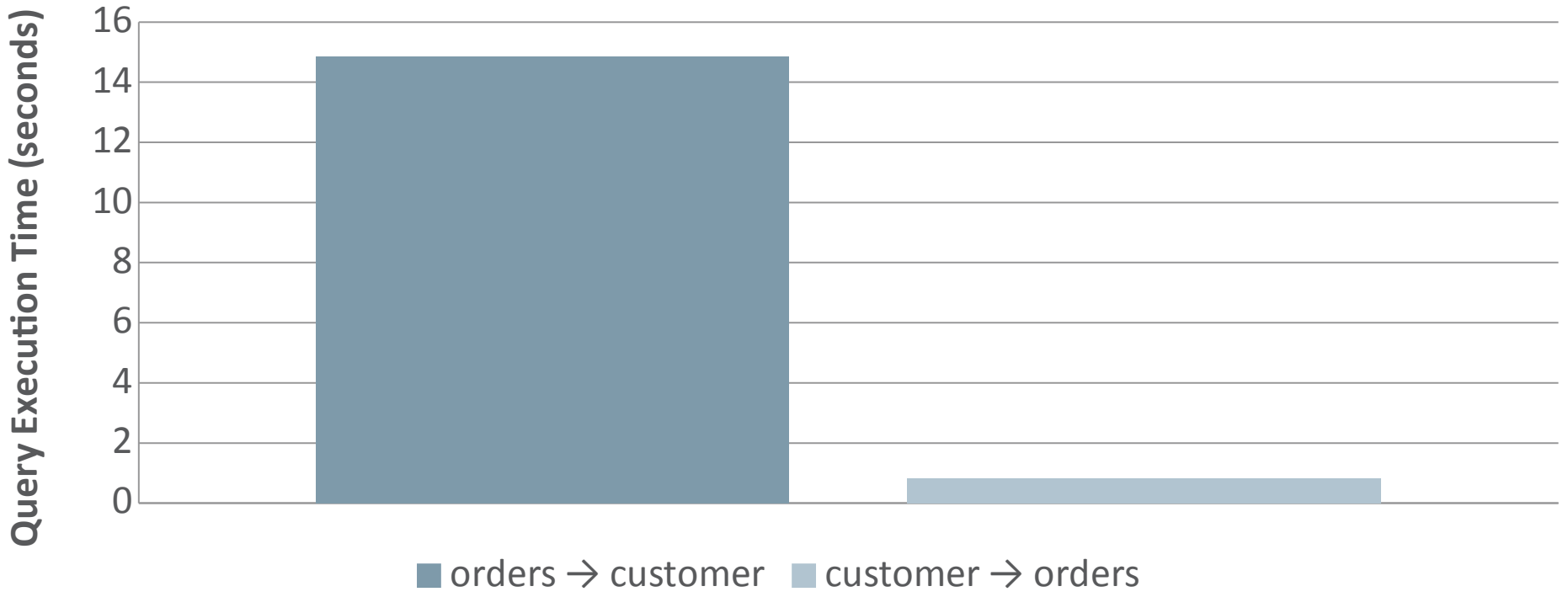
id	select type	table	type	possible keys	key	key len	ref	rows	filtered	extra
1	SIMPLE	orders	ALL	i_o_orderdate, i_o_custkey	NULL	NULL	NULL	15000000	31.19	Using where
1	SIMPLE	customer	eq_ref	PRIMARY	PRIMARY	4	dbt3.orders. o_custkey	1	33.33	Using where

Histograms Example

ANALYZE TABLE *customer* UPDATE HISTOGRAM ON *c_acctbal* WITH 1024 BUCKETS;
EXPLAIN SELECT *
FROM *customer* JOIN *orders* ON *c_custkey* = *o_custkey*
WHERE ***c_acctbal* < -1000** AND *o_orderdate* < '1993-01-01';

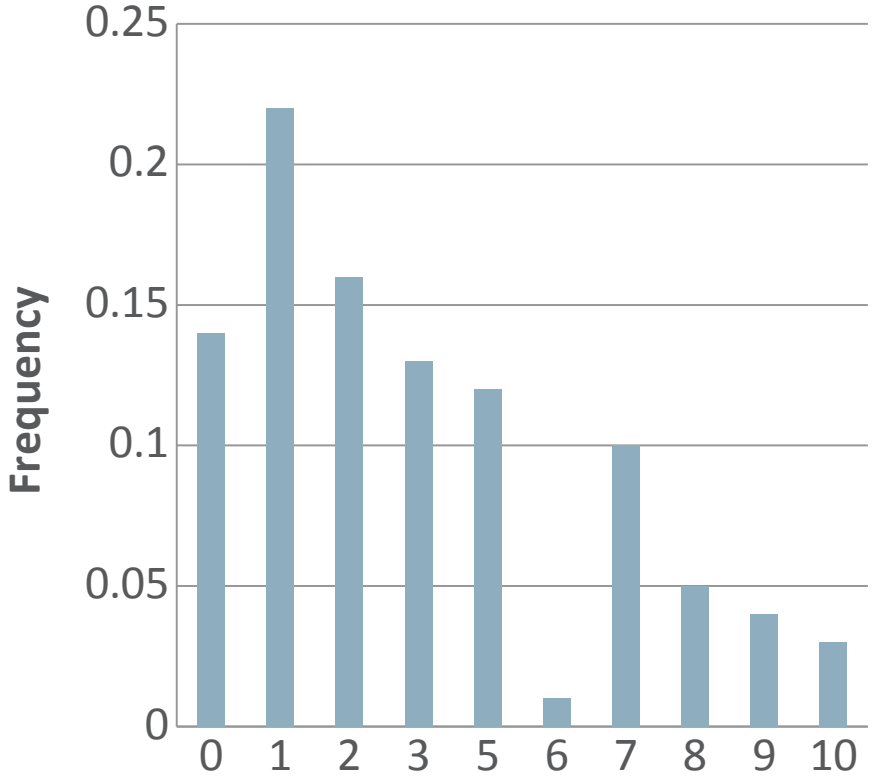
id	select type	table	type	possible keys	key	key len	ref	rows	filtered	extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	0.00	Using where
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_custkey	5	dbt3. customer. c_custkey	1	31.19	Using where

Histograms Example



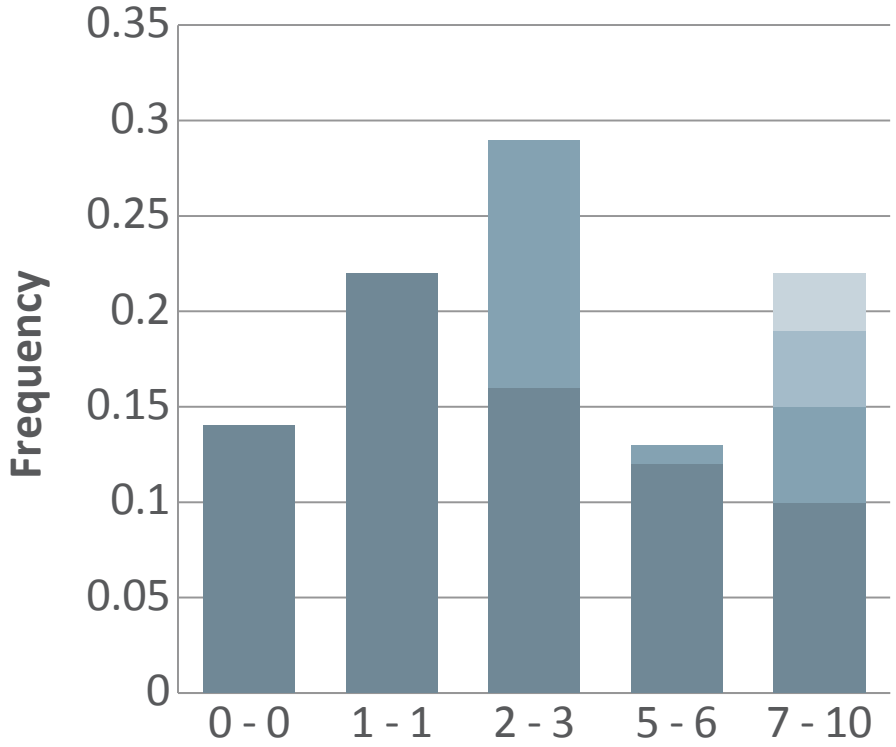
Singleton Histogram

- One value per bucket
- Each bucket stores:
  - Value
  - Cumulative frequency
- Well suited to estimate both equality and range predicates



Equi-Height Histogram

- Multiple values per bucket
- Not quite equi-height
  - Values are not split across buckets
  - Frequent values in separate buckets
- Each bucket stores:
  - Minimum value
  - Maximum value
  - Cumulative frequency
  - Number of distinct values
- Best suited for range predicates



Usage

- Create or refresh histogram(s) for column(s):

  ANALYZE TABLE *table* UPDATE HISTOGRAM ON *column* [, *column*] WITH *n* BUCKETS;

  - Note: Will only update histogram, not other statistics
- Drop histogram:

  ANALYZE TABLE *table* DROP HISTOGRAM ON *column* [, *column*];
- Based on entire table or sampling:
  - Depends on available memory: **histogram_generaton_max_mem_size** (default: 20 MB)
- New storage engine API for sampling
  - Default implementation: Full table scan even when sampling
  - Storage engines may implement more efficient sampling

Storage

- Stored in a JSON column in data dictionary
- Can be inspected in Information Schema view

```
SELECT JSON_PRETTY(histogram)
FROM information_schema.column_statistics
WHERE schema_name = 'dbt3_sf1'
    AND table_name ='lineitem'
    AND column_name = 'l_linenumber';
```

Histogram Content


```
{
  "buckets": [[1, 0.2499493852494869 8], [2, 0.4642106640072052 3],
    [3, 0.6427401784471978], [4, 0.7855470933802572],
    [5, 0.8927398868395817], [6, 0.964237075325 58], [7, 1] ],
  "data-type": "int",
  "null-values": 0.0,
  "collation-id": 8,
  "last-updated": "2018-02-03 21:05:21.690872",
  "sampling-rate": 0.2082911543745725 2,
  "histogram-type": "singleton",
  "number-of-buckets-specified": 1024
}
```

When to Create Histograms

- Histograms are useful for columns that are
  - Not the first column of any index, and
  - Used in WHERE conditions of
    - JOIN queries
    - Queries with IN subqueries
    - ORDER BY … LIMIT queries
- Best fit
  - Low cardinality columns (e.g., gender, orderStatus, dayOfWeek, enums)
  - Columns with uneven distribution (skew)
  - Stable distribution over time

When *Not* to Create Histograms

- First column of an index
- Column that is never used in WHERE clauses
- Monotonically increasing column values (e.g., timestamps)
  - Histograms will need frequent updates
  - Consider creating and index instead

How Many Buckets?

- If possible, enough to create a singleton histogram
- Equi-height: 100 buckets should be enough
  - 1 % resolution

Optimizer Hints

New Optimizer Hints in 8.0

- Join order
  - JOIN_ORDER(*tables*)
  - JOIN_PREFIX(*tables*)
  - JOIN_SUFFIX(*tables*)
  - JOIN_FIXED_ORDER()
- Setting system variables
  - SET_VAR(*sysvar=value*)
- View/derived table/CTE merge
  - MERGE(*views*)
  - NO_MERGE(*views*)
- Index merge
  - INDEX_MERGE(*indexes*)
  - NO_INDEX_MERGE(*indexes*)

Join Order Hints

- No need to reorganize the FROM clause to add join order hints like you will for STRAIGHT_JOIN
- /*+ JOIN_ORDER(*t1*, *t2*, …) */
- /*+ JOIN_PREFIX(*t1*, *t2*, …) */
- /*+ JOIN_SUFFIX(…, *ty*, *tz*) */
- /*+ JOIN_FIXED_ORDER() */
  - Replacement for STRAIGHT_JOIN

Join Order Hint Example

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()

View/Derived Table/CTE Merge Hints

- Views, derived tables and CTEs are, if possible, merged into outer query
- NO_MERGE can override default behavior

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

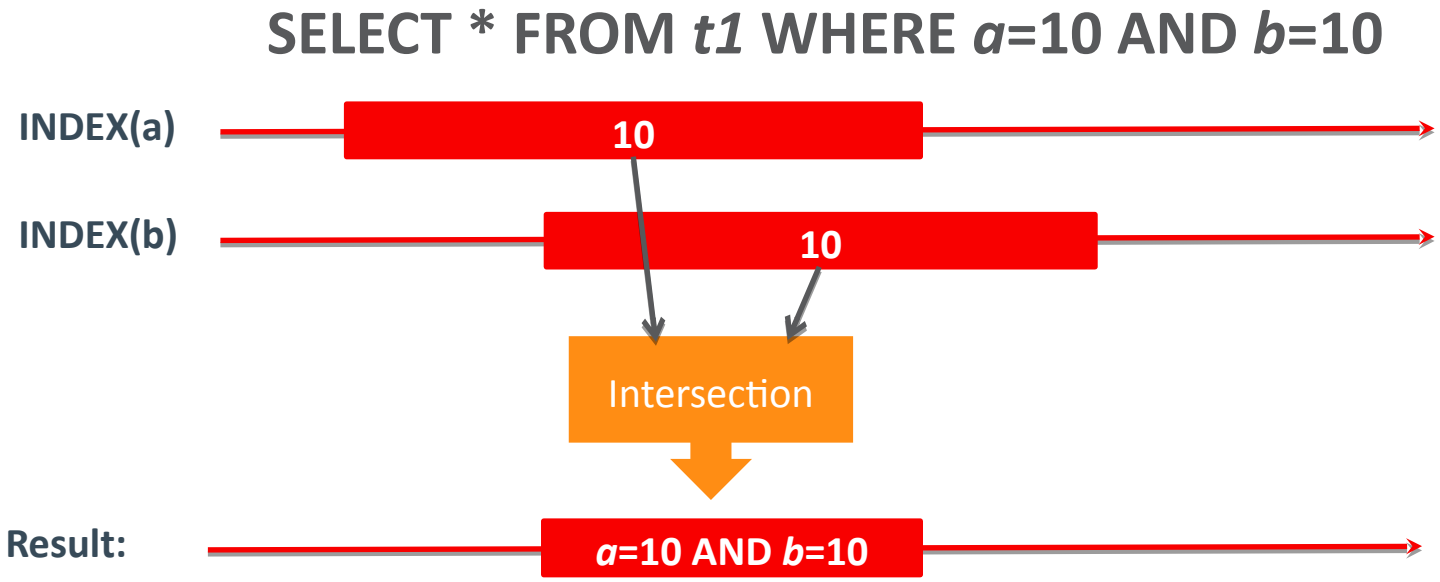
- MERGE will force a merge

```
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 CTEs

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

Index Merge Example



Index Merge Hints

- Index merge: Merge rows from multiple range scans on a single table
- Algorithms: union, intersection, sort union
- Users can specify which indexes to use for index merge
  - /*+ INDEX_MERGE(*t1 idx1*, *idx2*, …) */
  - /*+ NO_INDEX_MERGE(*t1 idx1*, *idx2*, …) */

Index Merge Hint Example

Low selectivity predicate

EXPLAIN SELECT count(*) FROM users
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	users	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 users WHERE user_type=2 AND status=0 AND
parent_id=0;
...
1 row in set (1.37 sec)
mysql> SELECT /*+ INDEX_MERGE(users user_type, status) */ count(*)
    -> FROM users WHERE user_type=2 AND status=0 AND parent_id=0;
...
1 row in set (0.18 sec)
```

Hints to Set Session Variables

- Set a session variable for the duration of a single statement
- Examples:

```
SELECT /*+ SET_VAR(sort_buffer_size = 16M) */ name FROM people ORDER BY name;

INSERT /*+ SET_VAR(foreign_key_checks = OFF) */ INTO t2 VALUES (1, 1), (2, 2), (3, 3);

SELECT /*+ SET_VAR(optimizer_switch = 'condition_fanout_filter = off') */ *
FROM customer JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < 0 AND o_orderdate < '1993-01-01';
```

- Note: Not all session variables are settable through hints:

```
mysql> SELECT /*+ SET_VAR(max_allowed_packet=128M) */ * FROM t1;
Empty set, 1 warning (0,00 sec)
Warning (Code 4537): Variable 'max_allowed_packet' cannot be set using SET_VAR hint.
```

Skip Index Dives for FORCE INDEX Hint

- Estimate number of rows in a range
  - The optimizer asks the storage engine for number of rows to be scanned from an index
  - Index dives: InnoDB finds first and last row in range and estimates distance
  - Example:

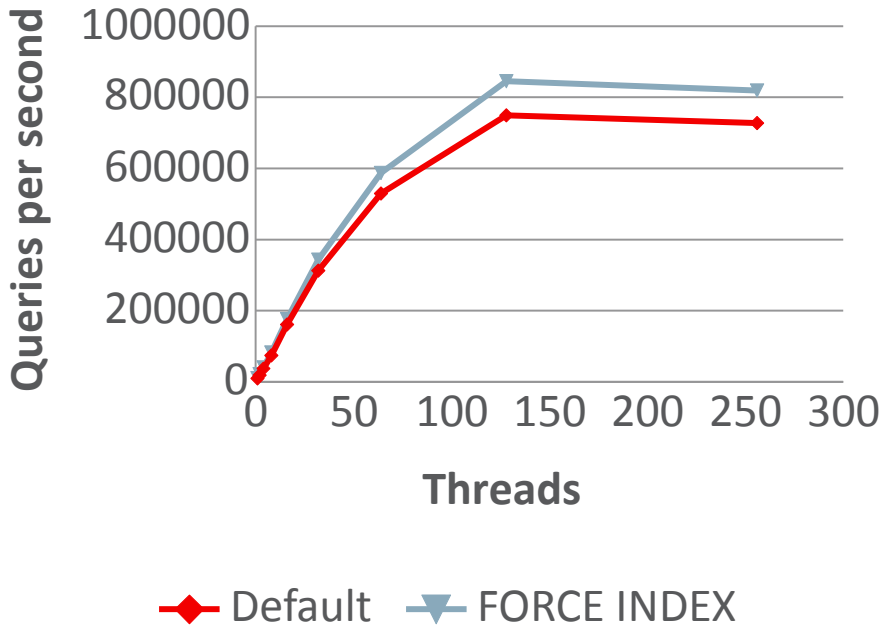
    SELECT * FROM *t1* WHERE (*key1* > 10 AND *key1* < 20) AND *key2* > 30

    Indexes on *key1* and *key2* will be used to get estimates for the two ranges
- MySQL 8.0 will skip index dives when all of these are satisfied:
  - FORCE INDEX applies to a single index
  - Only a single table is accessed in the query
  - No subqueries
  - No GROUP BY, DISTINCT or ORDER BY

Skip Index Dives Performance

- Sysbench schema
  - 8 tables
  - 1 million rows per table
  - Secondary index on integer column *k*
  - *k* values are randomly and uniformly distributed 1–1000000
- Queries

  SELECT *c* FROM *sbtest* WHERE *k* = ?
  SELECT *c* FROM *sbtest* FORCE INDEX(*k_1*)
  WHERE *k* = ?
- 10–13 % improvement



Invisible Index

- Index is maintained by the storage engine, but invisible to the optimizer
  - Override with SET optimizer_switch='use_invisible_indexes=on';
- Primary key can't be invisible
- Use case: Check for performance drope *before* removing index

ALTER TABLE *t1* **ALTER INDEX** *idx* **INVISIBLE**;
SHOW INDEXES FROM *t1*;

```
+---------+----------------+--------------------+--------------+
| Table   | Key_name       | Column_name        | Visible      |
+---------+----------------+--------------------+--------------+
| t1      | idx            | a                  | NO           |
+---------+----------------+--------------------+--------------+
```

Improved Performance with CTEs

CTEs Instead of Derived Tables

- A derived table is a subquery in the FROM clause:

  SELECT … FROM (*subquery*) AS *derived*, *t1* …

- A CTE is just like a derived table, but the declaration comes before the query block:

  WITH *derived* AS (*subquery*)
  SELECT … FROM *derived*, *t1* …

- May precede SELECT, UPDATE, DELETE and be used in subqueries:

  WITH *derived* AS (*subquery*)
  DELETE FROM *t1* WHERE *t1.a* IN (SELECT *b* FROM *derived*);

Better Performance with CTEs

- Derived tables can't be referenced twice

```
SELECT …
FROM (SELECT a, b, SUM(c) AS s FROM t1 GROUP BY a, b) AS d1
JOIN (SELECT a, b, SUM(c) AS s FROM t1 GROUP BY a, b) AS d2 ON d1.b = d2.a;
```

- CTEs can

```
WITH d AS (SELECT a, b, SUM(c) AS s FROM t1 GROUP BY a, b)
SELECT … FROM d AS d1 JOIN d AS d2 ON d1.b = d2.a;
```

- Better performance with materialization
  - Multiple CTE references are only materialized once
  - Derived tables and views will be materialized once per reference

DBT-3 Query 15 Top Supplier

Using view

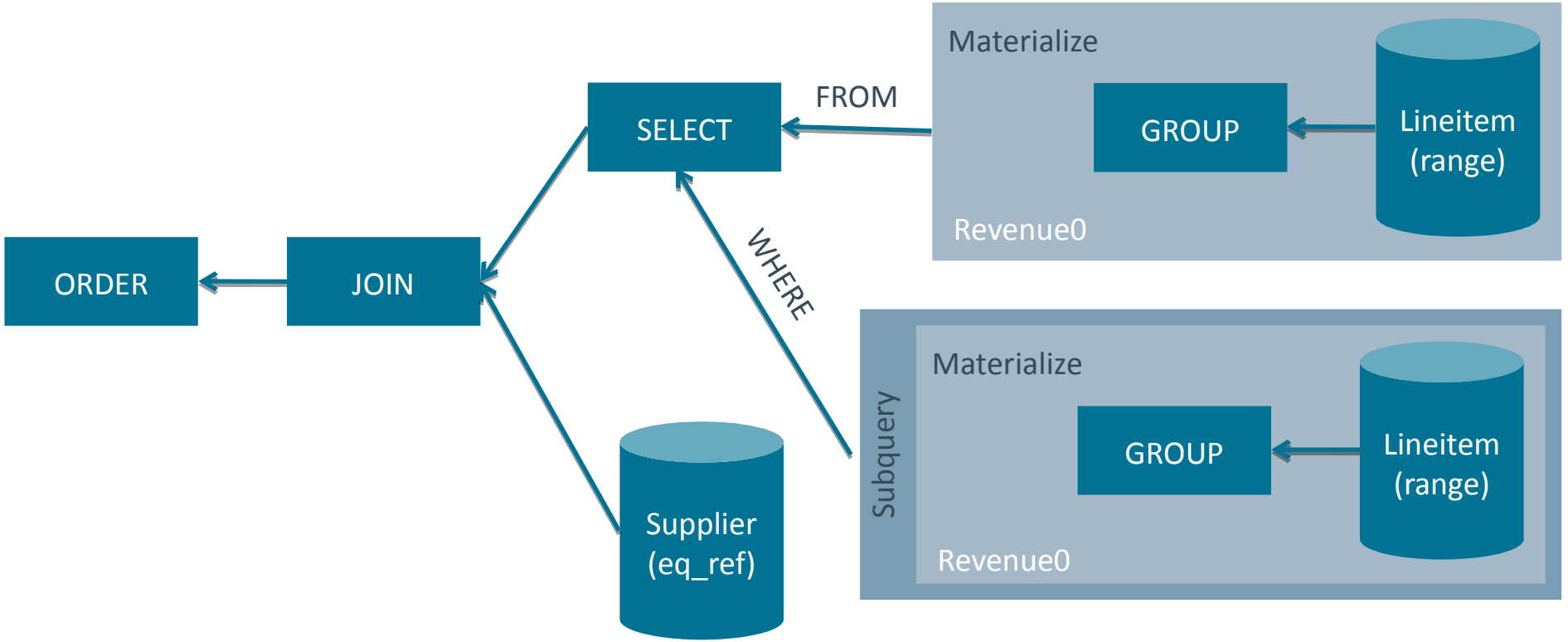
```
CREATE VIEW revenue0 (supplier_no, total_revenue) AS
(
   SELECT l_suppkey,
          SUM(l_extendedprice * (1- l_discount))
   FROM  lineitem
   WHERE l_shipdate >= '1996-07-01'
      AND l_shipdate < DATE_ADD('1996-07-01',
          INTERVAL '90' DAY)
   GROUP BY l_suppkey
);
```

Using CTE

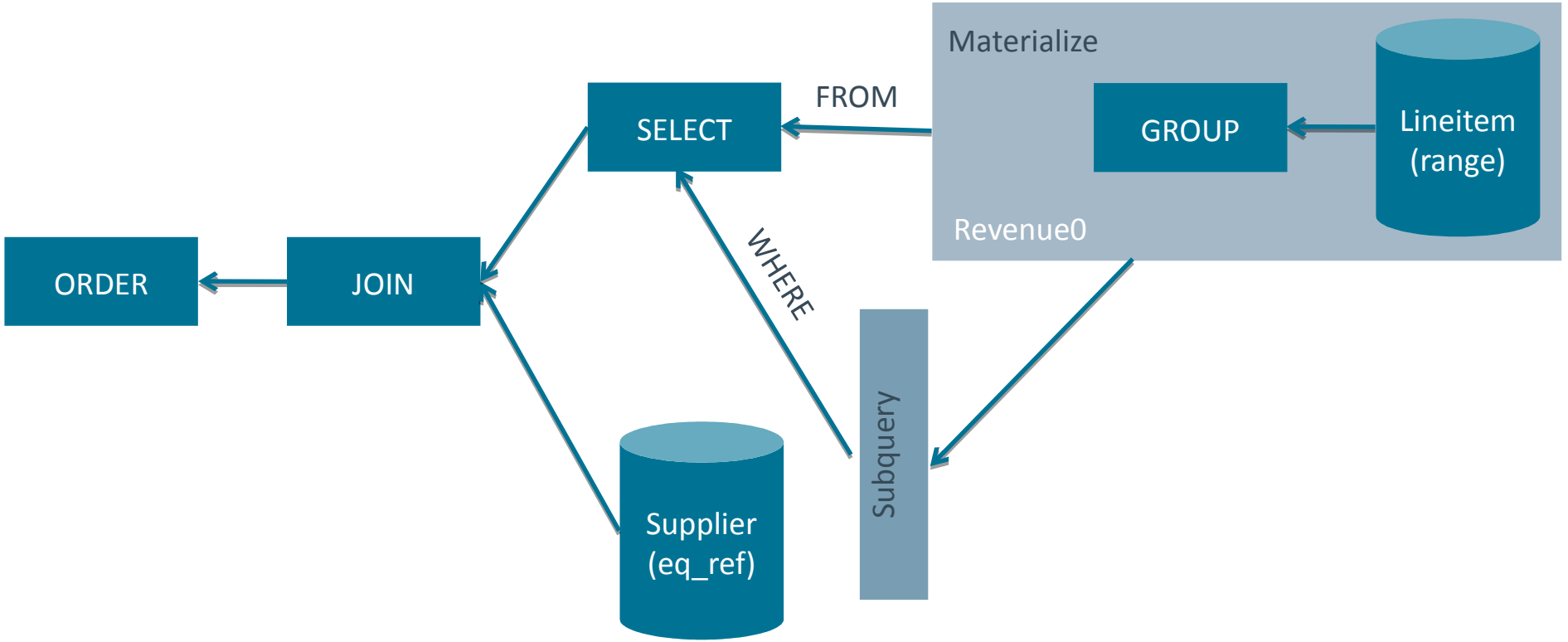
```
WITH revenue0 (supplier_no, total_revenue) AS
(
   SELECT l_suppkey,
          SUM(l_extendedprice *  (1-l_discount))
   FROM  lineitem
   WHERE l_shipdate >= '1996-07-01'
      AND l_shipdate < DATE_ADD('1996-07-01',
          INTERVAL '90' DAY)
   GROUP BY  l_suppkey
)
```

```
SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
FROM supplier, revenue0
WHERE s_suppkey = supplier_no AND total_revenue = (SELECT MAX(total_revenue) FROM revenue0)
ORDER BY s_suppkey;
```

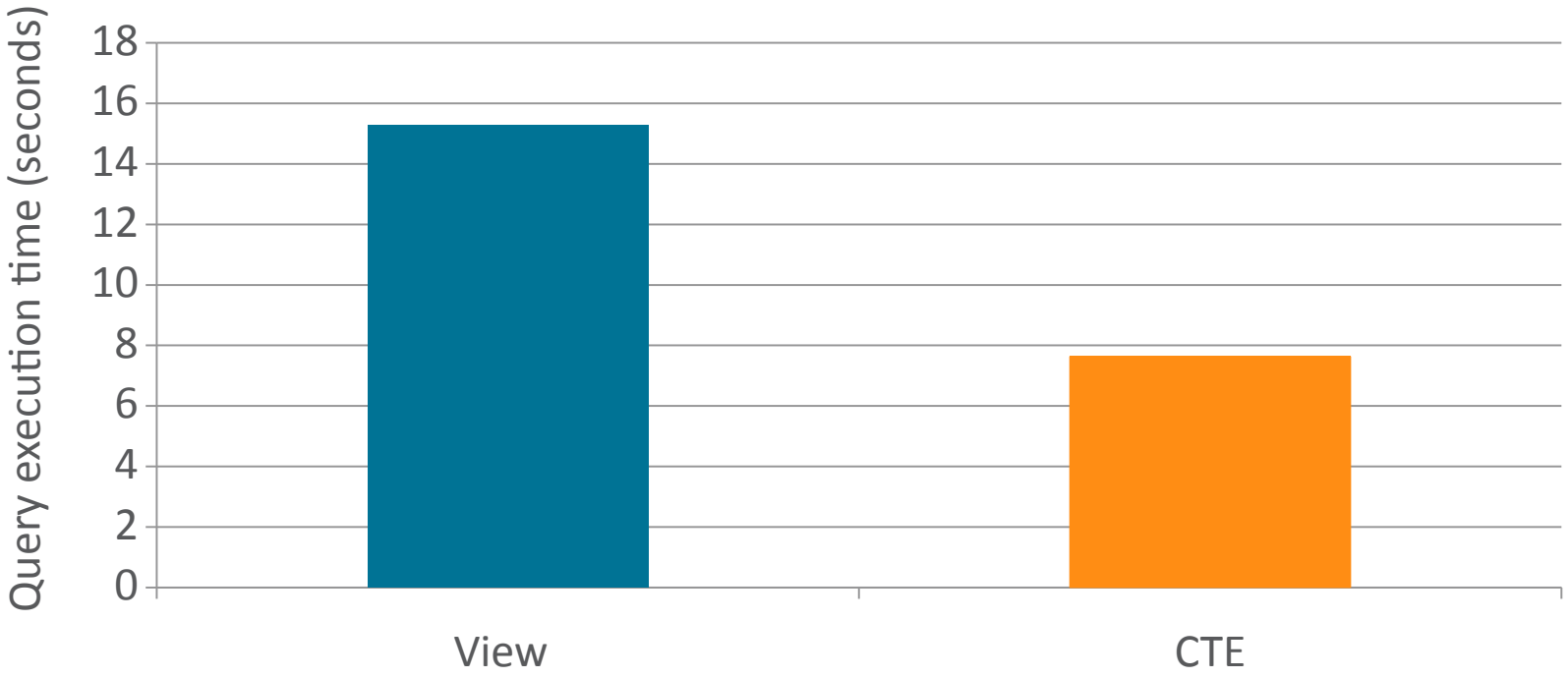
DBT-3 Query 15 with View Materialization



DBT-3 Query 15 with CTE Materialization



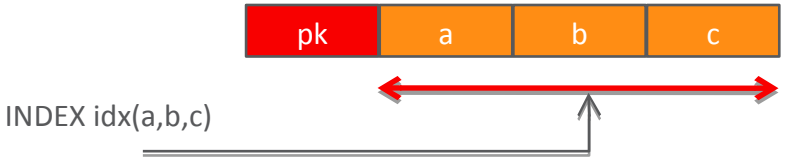
DBT-3 Query 15 Performance



Improvements in Query Execution

Index Skip-Scan — New in 8.0.13

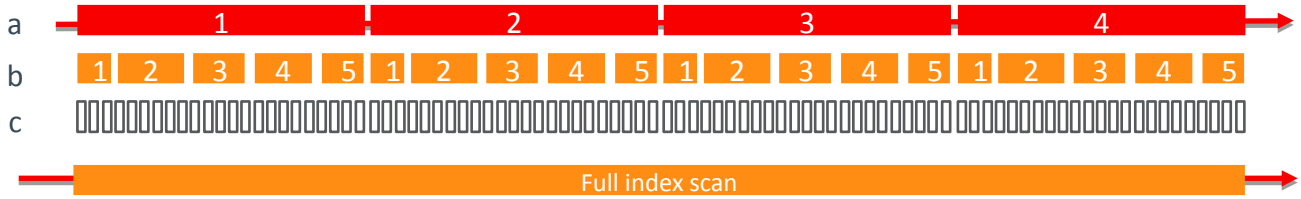
- Used for multi-column index
- Condition not on first part of index
- Cost-based choice



SELECT * FROM *t1* WHERE *b* < 3;

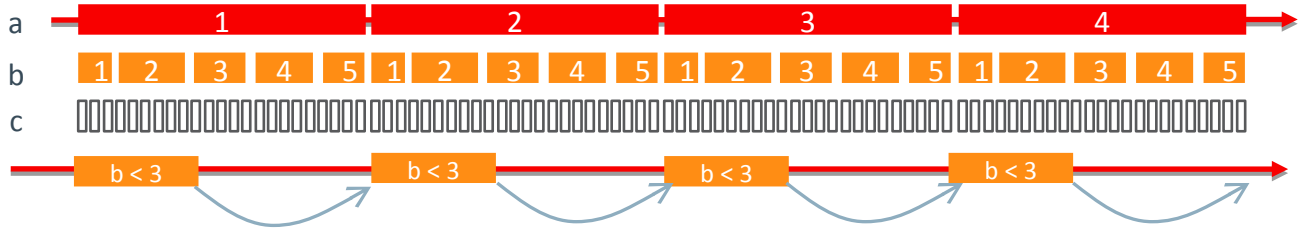
Before MySQL 8.0.13:

Full index scan



MySQL 8.0.13:

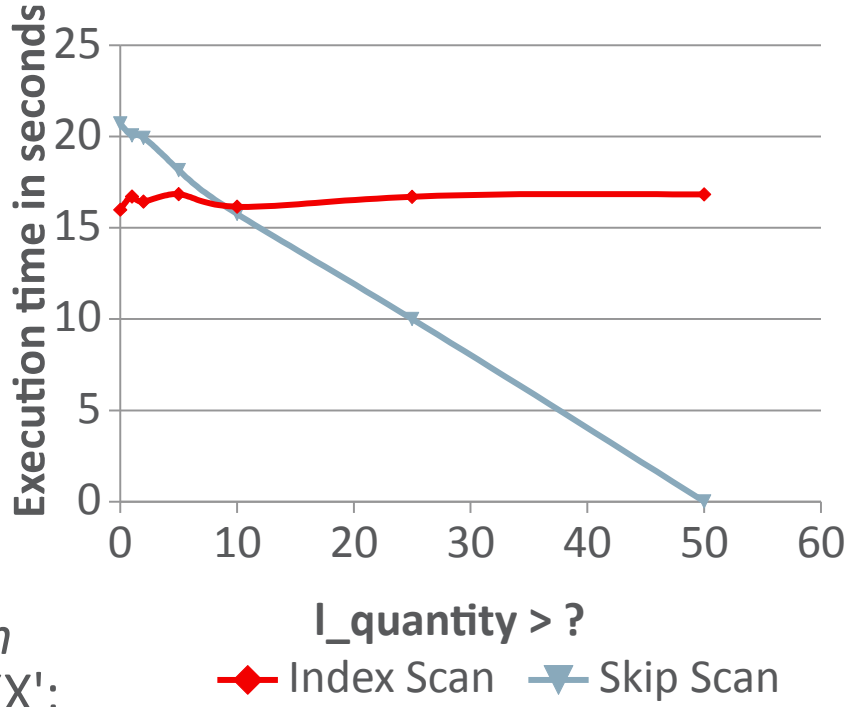
Skip-scan



Index Skip-Scan Performance Example

- DBT-3 scale factor 10 database
- Create index on *lineitem*
  - (*l_shipdate*, *l_quantity*, *l_shipmode*)
  - 60 million rows
  - Cardinality:
    - *l_shipdate*: 2526
    - *l_quantity*: 50
- Query:

```
SELECT l_shipdate, l_quantity FROM lineitem
WHERE l_quantity > ? AND l_shipmode = 'XXX';
```



- Skip-scan is faster when less than 80 % of the rows are accessed!

Thanks for the patch, Facebook!

Improved Scan Performance

- Motivation
  - InnoDB already uses an internal buffer to fetch records in batches
  - The optimizer knows the operation and estimated number of records to read
- The optimizer knows how large the buffer should be
- 8.0 extends the handler API
  - Provide a buffer to the storage engine when expecting more than one row
- Example queries with improved performance:

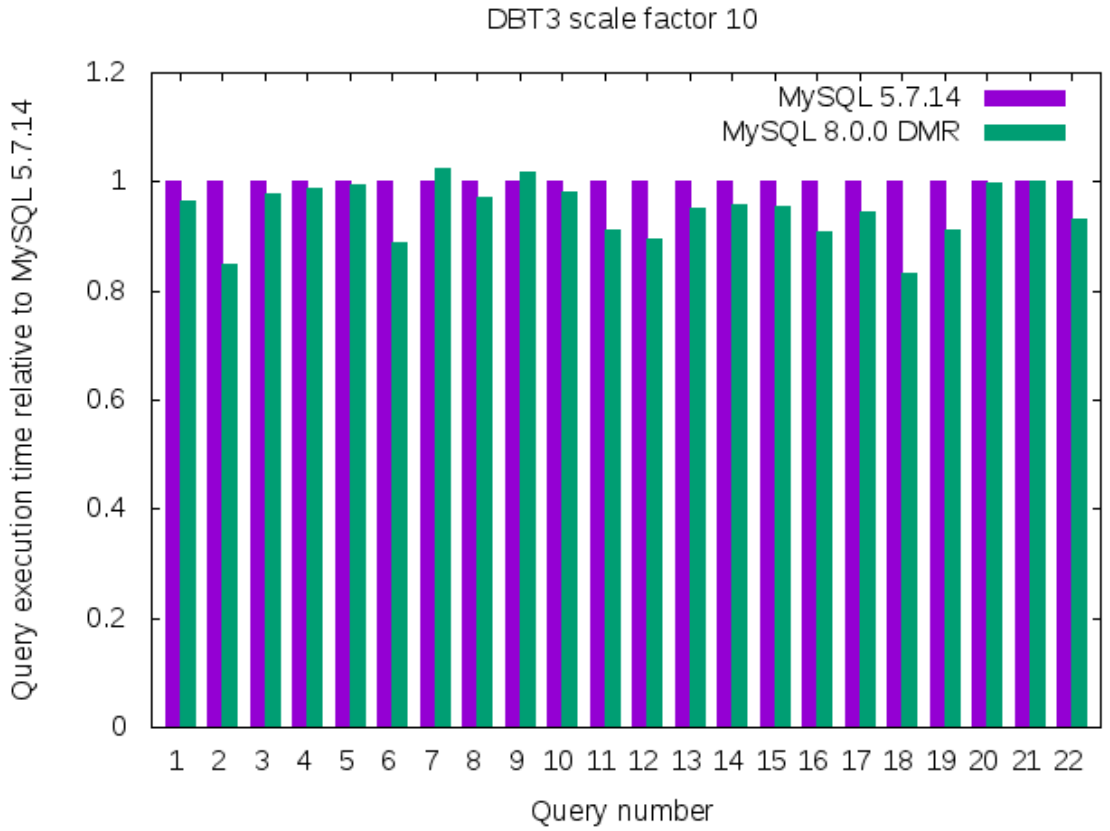
  SELECT * FROM *t*;

  SELECT * FROM *t* WHERE *pk* BETWEEN 1000 AND 10000;

  SELECt * FROM *t1*, *t2* WHERE *t1.pk=t2.pk*; /* Buffer for the outer table */

Improved Scan Performance

- 18 out of 22 queries got improved performance
- 4 queries show more than 10 % improvement
- 2 queries show more than 15 % improvement

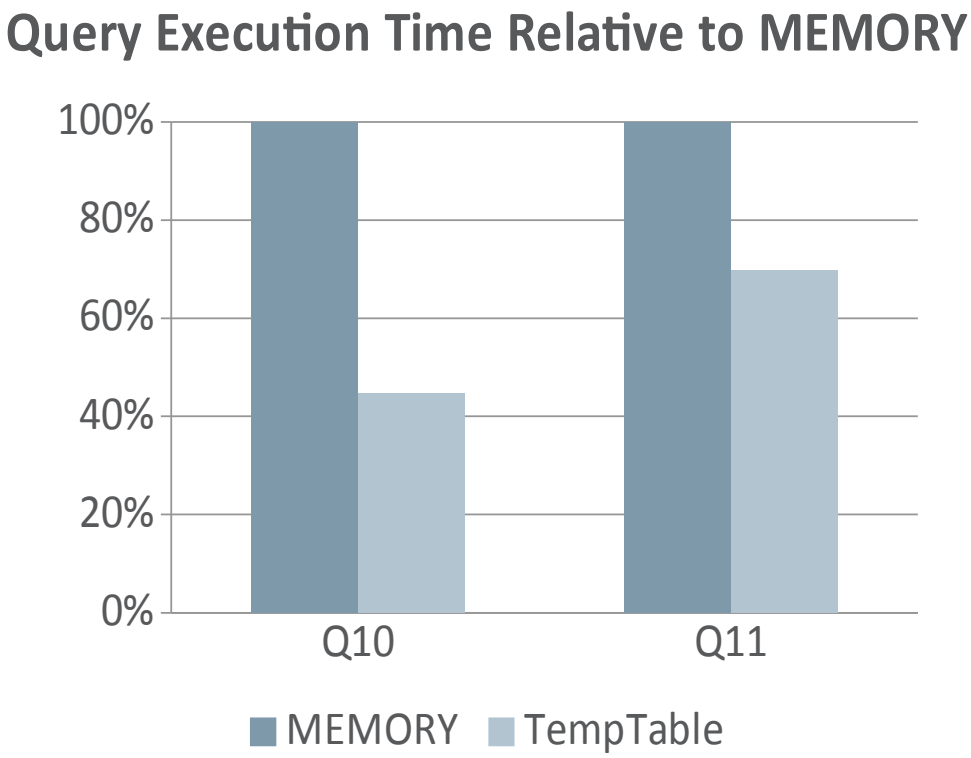


Internal Temporary Table Storage Engine

- Variable-length rows
- Supports JSON, TEXT, BLOB and geometry types
  - **New in MySQL 8.0.13!**
- Supports overflow to disk
  - No overhead for converting to InnoDB/MyISAM when table becomes big
- Common memory pool for all internal temporary tables
  - No fixed limit per table
  - temptable_max_ram (default 1 GB)
- Performance schema events for memory/disk usage
  - memory/temptable/physical_ram, memory/temptable/physical_disk

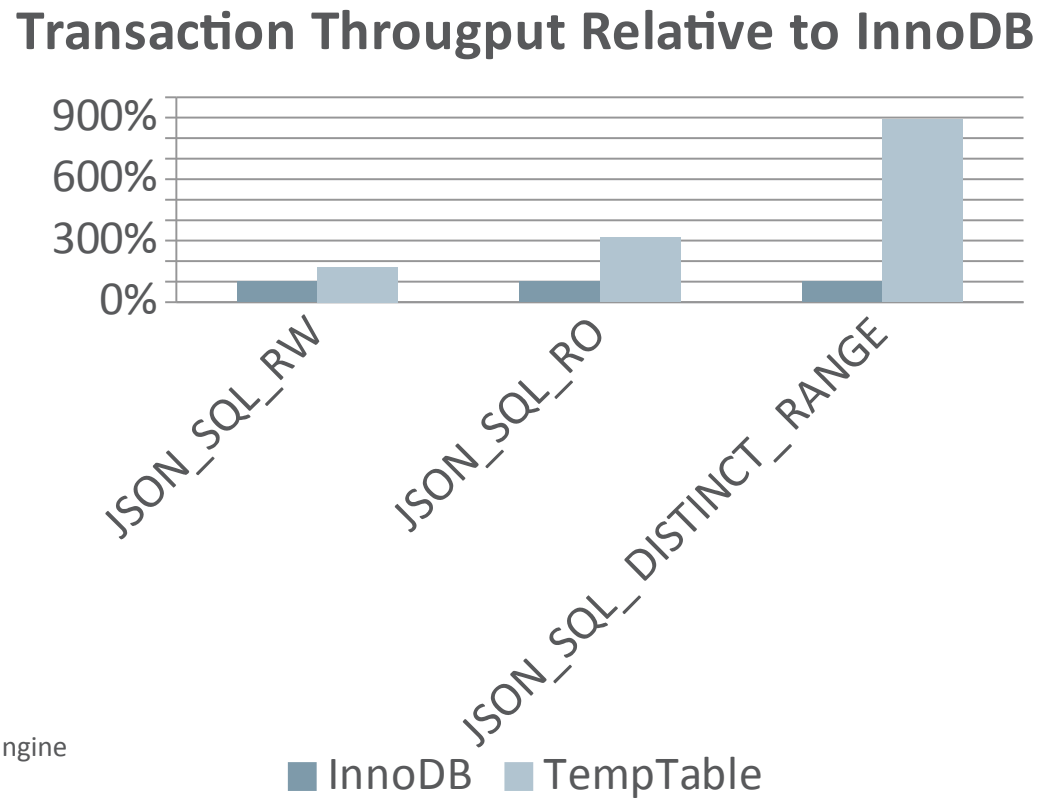
DBT-3 with TempTable vs. MEMORY

- MySQL 8.0.13
- DBT-3, Scale factor 10
- innodb_buffer_pool_size = 32G (CPU-bound)
- internal_tmp_mem_storage_engine = "TempTable" or "MEMORY"
- Other 20 queries have same performance for both engines



JSON Data in TempTable

- MySQL 8.0.13
- JSON variant of sysbench tests
  - Range size: 500
  - Data size: 127 GB
- 64 threads



http://mysqlserverteam.com/mysql-8-0-support-for-blobs-in-temptable-engine

Descending Index

```
CREATE TABLE t1 (
   a INT,
   b INT,
   INDEX a_b (a DESC, b ASC)
);
```

- In 5.7: Index in ascending order is created, server scans it backwards
- In 8.0: Index in descending order is created, server scans it forwards
  - Works on B-tree indexes only
  - Benefits:
    - Use indexes instead of filesort for ORDER BY clause with ASC/DESC sort key
    - Forward index scan is slightly faster than backward index scan

Feature descriptions and design details directly from the source.

http://mysqlserverteam.com/

Integrated Cloud

Applications & Platform Services


ORACLE®