Query Optimizer in MariaDB 10.4

Sergei Petrunia,
Query Optimizer Developer
MariaDB Corporation

2019 MariaDB Developers Unconference
New York

New Optimizer features in MariaDB 10.4

- Optimizer trace
- Sampling for histogram collection
- Rowid filtering
- New default settings
- Condition Pushdown into IN-subqueries
- Condition Pushdown from HAVING into WHERE

New default settings

New default settings for statistics

- Do use condition selectivity

```
-optimizer_use_condition_selectivity=1
+optimizer_use_condition_selectivity=4
```

```
1 Use selectivity of predicates as in MariaDB 5.5.
2 Use selectivity of all range predicates supported by indexes.
3 Use selectivity of all range predicates estimated without histogram.
4 Use selectivity of all range predicates estimated with histogram.
```

- Make use of EITS statistics (incl. Histograms if they are available)

```
-use_stat_tables=NEVER
+use_stat_tables=PREFERABLY_FOR_QUERIES
```

  – But don’t collect stats unless explicitly told to do so

- Do build histograms when collecting EITS statistics

```
-histogram_size=0
+histogram_size=254
-histogram_type=SINGLE_PREC_HB
+histogram_type=SINGLE_PREC_HB
```

New default settings

- Join buffer will auto-size itself

```
-optimize_join_buffer_size=OFF
+optimize_join_buffer_size=ON
```

  - (can use ANALYZE for statements to see the size)

- Use index statistics (cardinality) instead of records_in_range for large IN-lists

```
-eq_range_index_dive_limit=10
+eq_range_index_dive_limit=200
```

  - Just following MySQL here

Sampling for histograms

Histograms in MariaDB

- Introduced in MariaDB 10.0
  - Manual command to collect, ANALYZE … PERSISTENT FOR …
  - Optimizer settings to use them
  - Histogram is collected from **ALL** table data
    - Other statistics (avg_frequency, avg_length), too.
- Results
  - A few users
  - Histogram collection **is expensive**
    - Cost of full table scan + full index scans, and even more than that

Histograms in MariaDB 10.4

- MariaDB 10.4
  - “Bernoulli sampling” - roll the dice for each row
  - Controlled with @@**analyze_sample_percentage**
    - 100 (the default) – “use all data”
    - 0 – (recommended) – “Determine sample ratio automatically”
- MySQL 8.0 also added histograms
  - Uses Bernoulli sampling
  - @@**histogram_generation_max_mem_size**=20MB.
- PostgreSQL has genuine random-jump sampling
  - default_statistics_target

Histogram collection performance

- See MDEV-17886, (TODO: Vicentiu’s data?)
- Both MariaDB and MySQL: ANALYZE for columns is as fast as full table scan.

```
ANALYZE TABLE PERSISTENT FOR COLUMNS (...) INDEXES ();
```

- “Persistent for ALL” will also scan indexes

```
ANALYZE TABLE PERSISTENT FOR ALL;
```

- PostgreSQL is much faster with genuine sampling
  - Vicentiu’s has a task in progress for this.

Histogram precision

- MariaDB histograms are very compact
  - min/max column values, then 1-byte or 2-byte bounds (SINGLE|DOUBLE_PREC_HB)
  - 255 bytes per histogram => 128 or 255 buckets max.
- MySQL
  - Histogram is stored as JSON, bounds are stored as values
  - 100 Buckets by default, max is 1024
    - In our tests, more buckets help in some cases
- PostgreSQL
  - Histogram bounds stored as values
  - 100 buckets by default, up to 10K allowed
- Testing is still in progress :-(, the obtained data varies.

Problem with correlated conditions

```
select ...
from order_items
where shipdate='2015-12-15' AND item_name='christmas light'
                                          'swimsuit'
```

- Possible selectivities
  - MIN(1/n, 1/m)
  - **(1/n) * (1/m)**
  - 0

Problem with correlated conditions

```
select ...
from order_items
where shipdate='2015-12-15' AND item_name='christmas light'
                                          'swimsuit'
```

- PostgreSQL: Multi-variate statistics
  - Detects functional dependencies, col1=F(col2)
  - Only used for equality predicates
  - Also #DISTINCT(a,b)
- MariaDB: MDEV-11107: Use table check constraints in optimizer
  - Stalled?

Histograms: conclusions

- 10.4
  - Sampling makes **ANALYZE TABLE … PERSISTENT FOR COLUMNS** run at full-table-scan speed.
  - **@@analyze_sample_rows**
- Further directions
  - Do real sampling (in progress)
  - More space for the histograms (?)
  - Handle correlations (how?)

Optimizer trace

Optimizer trace

- Available in MySQL since MySQL 5.6

```
mysql> set optimizer_trace=1;

mysql> <query>;

mysql> select * from
    ->   information_schema.optimizer_trace;
```

- Now, similar feature in MariaDB

```
{
 "steps": [
  {
   "join_preparation": {
     "select#": 1,
     "steps": [
      {
        "expanded_query": "/* select#1 */ select `t1`.`col1` AS `col1`,`t1`.`col2`
AS `col2` from `t1` where (`t1`.`col1` < 4)"
      }
     ]
   }
  },
  {
   "join_optimization": {
     "select#": 1,
     "steps": [
      {
        "condition_processing": {
          "condition": "WHERE",
          "original_condition": "(`t1`.`col1` < 4)",
          "steps": [
            {
              "transformation": "equality_propagation",
              "resulting_condition": "(`t1`.`col1` < 4)"
            },
            {
              "transformation": "constant_propagation",
              "resulting_condition": "(`t1`.`col1` < 4)"
            },
            {
              "transformation": "trivial_condition_removal",
              "resulting_condition": "(`t1`.`col1` < 4)"
            }
          ]
        }
      },
      {
```

The goal is to understand the optimizer

- “Why was query plan X not chosen”
  - Subquery was not converted into semi-join
    - This would exceed MAX_TABLES
  - Subquery materialization was not used
    - Different collations
  - Ref acess was not used
    - Incompatible collations

- What changed between the two hosts/versions
  - diff trace_from_host1 trace_from_host2

Developer point of view

- The trace is always compiled in
- RAII-objects to start/end writing a trace
- Disabled trace added ~1-2% overhead
- Intend to add more tracing
  - Expect to get more output

Rowid filtering

What is PK-filter: in details

```
SELECT *
FROM orders JOIN lineitem ON o_orderkey=l_orderkey
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-06-30' AND
      o_totalprice between 200000 and 230000;
```

Filter for `lineitem` table built with condition

`l_shipdate BETWEEN '1997-01-01' AND '1997-06-30'`:

is a container that contains primary keys of rows from `lineitem` which `l_shipdate` value satisfy the above condition.

What is PK-filter: in details

```
SELECT *
FROM orders JOIN lineitem ON o_orderkey=l_orderkey
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-06-30' AND
      o_totalprice between 200000 and 230000;
```

Filter for `lineitem` table built with condition

`l_shipdate BETWEEN '1997-01-01' AND '1997-06-30'`:

is a container that contains primary keys of rows from `lineitem` which `l_shipdate` value satisfy the above condition.

What is PK-filter: in details

```
SELECT *
FROM orders JOIN lineitem ON o_orderkey=l_orderkey
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-02-01' AND
      o_totalprice > 200000;
```

1. There is index `i_l_shipdate` on `lineitem(l_shipdate)`

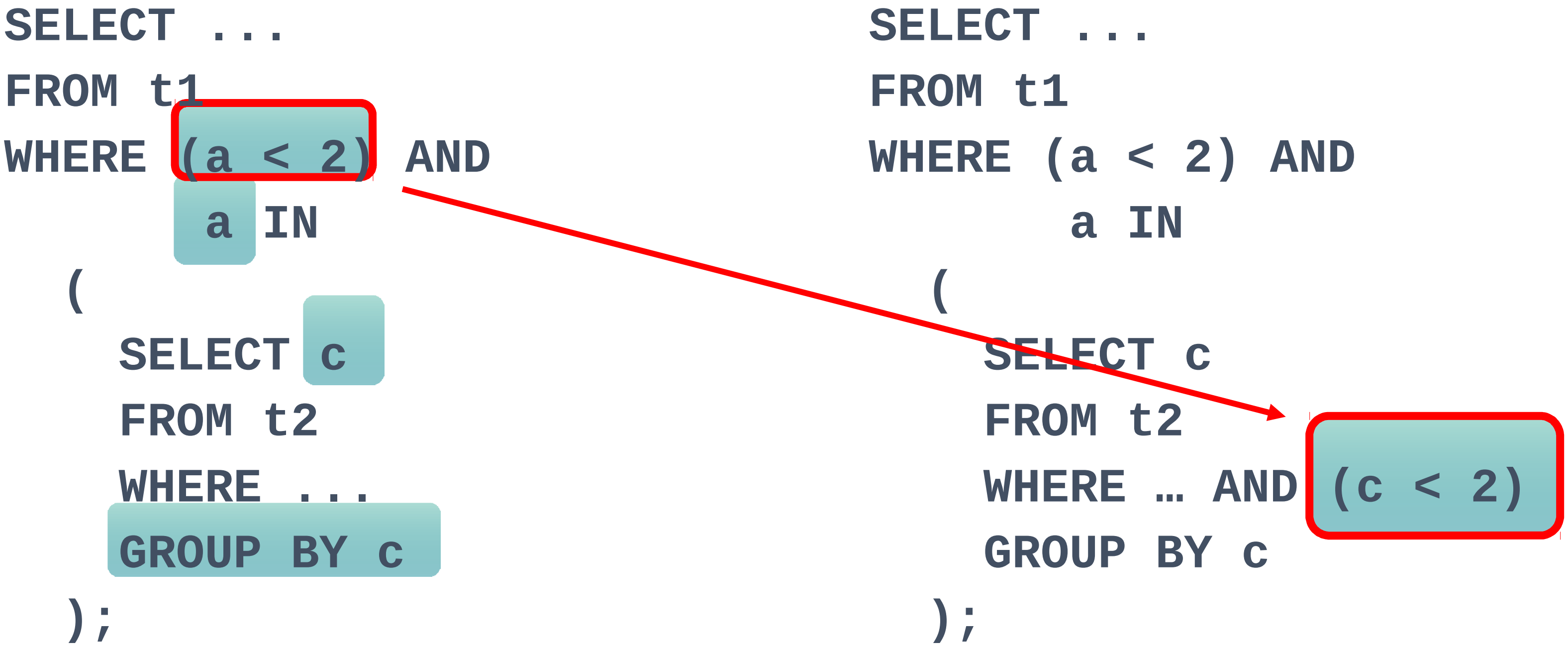
What is PK-filter: in details

```
SELECT *
FROM orders JOIN lineitem ON o_orderkey=l_orderkey
WHERE l_shipdate BETWEEN '1997-01-01' AND '1997-06-30' AND
      o_totalprice between 200000 and 230000;
```

2.

Condition pushdown...

How condition pushdown is made



```
SELECT ...
FROM t1
WHERE (a < 2) AND
        a IN
  (
    SELECT c
    FROM t2
    WHERE ...
    GROUP BY c
  );
```

```
SELECT ...
FROM t1
WHERE (a < 2) AND
        a IN
  (
    SELECT c
    FROM t2
    WHERE … AND (c < 2)
    GROUP BY c
  );
```

Thanks!