ANALYZE for statements
MariaDB's new tool for diagnosing the optimizer

Sergei Petrunia
MariaDB

FOSDEM 2016
Background: optimizer troubleshooting

- EXPLAIN shows the query plan

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SIMPLE</td>
<td>orders</td>
<td>ALL</td>
<td>PRIMARY,...</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>1507320</td>
<td>Using where</td>
</tr>
<tr>
<td></td>
<td>SIMPLE</td>
<td>lineitem</td>
<td>ref</td>
<td>PRIMARY,...</td>
<td>PRIMARY</td>
<td>4</td>
<td>orders.o_orderkey</td>
<td>1</td>
<td>Using where</td>
</tr>
</tbody>
</table>

- Sometimes problem is apparent
- Sometimes not
  - Query plan vs reality?
  - Where the time was spent?
Counters give summary information

- **Slow query log:** *Rows_examined*
- **Handler_XXX** status variables
- **Userstat:**
  ```sql
  SHOW (TABLE|INDEX)_STATISTICS
  ```
- **PERFORMANCE_SCHEMA:**
  ```sql
  table_io_waits_summary_by_table
  ```

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>orders</td>
<td>const</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>const</td>
<td>1</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>lineitem</td>
<td>ref</td>
<td>PRIMARY,i...</td>
<td>PRIMARY</td>
<td>4</td>
<td>const</td>
<td>2</td>
<td>Using where; Start temporary</td>
</tr>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>lineitem</td>
<td>ref</td>
<td>PRIMARY,i...</td>
<td>i_suppkey</td>
<td>5</td>
<td>lineitem.l_partkey</td>
<td>14</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>orders</td>
<td>eq_ref</td>
<td>PRIMARY,i...</td>
<td>PRIMARY</td>
<td>4</td>
<td>lineitem.l_orderkey</td>
<td>1</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>customer</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>orders.o_custkey</td>
<td>1</td>
<td>End temporary</td>
</tr>
</tbody>
</table>

- All are query-global
- Or server-global
- => Analysis is difficult.
Solution in MariaDB 10.1

ANALYZE statement

- Like EXPLAIN ANALYZE in PostgreSQL
- Or V$SQL_PLAN_STATISTICS in Oracle
ANALYZE vs EXPLAIN

**EXPLAIN**

- Optimize the query
- Produce EXPLAIN output

**ANALYZE**

- Optimize the query
- Run the query
  - Collect execution statistics
  - Discard query output
- Produce EXPLAIN output
  - With also execution statistics
analyze select *
from lineitem, orders
where o_orderkey=l_orderkey and
  o_orderdate between '1990-01-01' and '1998-12-06' and
  l_extendedprice > 1000000

r_ is for “real”

- r_rows – observed #rows
- r_filtered – observed condition selectivity.
### Interpreting `r_rows`

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th><code>r_rows</code></th>
<th>filtered</th>
<th><code>r_filtered</code></th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>orders</td>
<td>ALL</td>
<td>PRIMARY,i...</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>1504278</td>
<td>1500000</td>
<td>50.00</td>
<td>100.00</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>lineitem</td>
<td>ref</td>
<td>PRIMARY,i...</td>
<td>PRIMARY</td>
<td>4</td>
<td>orders.o_orderkey</td>
<td>2</td>
<td>4.00</td>
<td>100.00</td>
<td>0.00</td>
<td>Using where</td>
</tr>
</tbody>
</table>

- **ALL/index**
  - `r_rows` ≈ `rows`
  - Different in case of LIMIT or subqueries

- **range/index_merge**
  - Up to ~2x difference with InnoDB
  - Bigger difference in edge cases (IGNORE INDEX?)
Interpreting \texttt{r\_rows} (2)

For \texttt{ref} access

- rows is \text{AVG(records for a key)}

- Some discrepancy is normal

- Big discrepancy (>10x) is worth investigating
  - rows=1, \( r\_rows \gg rows \) ? No index statistics (\text{ANALYZE TABLE})
  - Column has lots of NULL values? (\text{innodb\_stats\_method})
  - Skewed data distribution?
    - Complex, \text{IGNORE INDEX}.  

**Interpreting \textit{r\_filtered}**

```
analyze select *
from lineitem, orders
where o_orderkey=l_orderkey and
  o_orderdate between '1990-01-01' and '1998-12-06' and
  l_extendedprice > 1000000
```

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>r_rows</th>
<th>filtered</th>
<th>r_filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>orders</td>
<td>ALL</td>
<td>PRIMARY,i_...</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>1504278</td>
<td>1500000</td>
<td>50.00</td>
<td>100.00</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>lineitem</td>
<td>ref</td>
<td>PRIMARY,i_...</td>
<td>PRIMARY</td>
<td>4</td>
<td>orders.o_orderkey</td>
<td>2</td>
<td>4.00</td>
<td>100.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \textit{(r\_)}filtered is selectivity of “Using where”
- \textit{r\_filtered} $\ll 100\% \rightarrow$ reading and discarding lots of rows
  - Check if conditions allow index use
    - Use \texttt{EXPLAIN|ANALYZE FORMAT=JSON} to see the condition
  - Consider adding indexes
- Don't chase \textit{r\_filtered}=100.00, tradeoff between reads and writes.
**r_filtered** and query plans

- **filtered**
  - Shows how many rows will be removed from consideration
  - Is important for N-way join optimization

- **r_filtered ≠ filtered**
  - Optimizer doesn't know condition selectivity → poor plans

- MariaDB's data for filtered: EITS Statistics (Histograms)
  - Improves plans for complex queries

```sql
SET histogram_size=100;
for each table $tbl with r_filtered ≠ filtered {
   ANALYZE TABLE $tbl PERSISTENT FOR ALL;
}
```
New columns

- `r_rows`
- `r_filtered`

Can check estimates vs reality

Can find [possibly] sub-optimal plans

Can find where optimizer is not aware of condition selectivity.
ANALYZE FORMAT=JSON

EXPLAIN FORMAT=JSON + ANALYZE = ANALYZE FORMAT=JSON
ANALYZE FORMAT=JSON

```json
{
    "query_block": {
        "select_id": 1,
        "r_loops": 1,
        "r_total_time_ms": 191747,
        "table": {
            "table_name": "orders",
            "access_type": "ALL",
            "possible_keys": ["PRIMARY", "i_o_orderdate"],
            "r_loops": 1,
            "rows": 1498194,
            "r_rows": 1.5e6,
            "r_total_time_ms": 14261,
            "filtered": 50,
            "r_filtered": 100,
            "attached_condition": "(orders.o_orderDATE between 1990-01-01 and 1998-12-06)"
        },
        "table": {
            "table_name": "lineitem",
            "access_type": "ref",
            "possible_keys": ["PRIMARY", "i_l_orderkey", "i_l_orderkey_quantity"],
            "key": "PRIMARY",
            "key_length": "4",
            "used_key_parts": ["l_orderkey"],
            "ref": ["dbt3sf1.orders.o_orderkey"],
            "r_loops": 1500000,
            "rows": 1,
            "r_rows": 4.0008,
            "r_total_time_ms": 170456,
            "filtered": 100,
            "r_filtered": 0,
            "attached_condition": "(lineitem.l_extendedprice > 1000000)"
        }
    }
}
```
ANALYZE FORMAT=JSON

```
{
    "query_block": {
        "select_id": 1,
        "r_loops": 1,
        "r_total_time_ms": 191747,
        "table": {
            "table_name": "orders",
            "access_type": "ALL",
            "possible_keys": ["PRIMARY", "i_o_orderdate"],
            "r_loops": 1,
            "rows": 1498194,
            "r_rows": 1.5e6,
            "r_total_time_ms": 14261,
            "filtered": 50,
            "r_filtered": 100,
            "attached_condition": "(orders.o_orderDATE between 1990-01-01 and 1998-12-06)"
        },
        "table": {
            "table_name": "lineitem",
            "access_type": "ref",
            "possible_keys": ["PRIMARY", "i_l_orderkey", "i_l_orderkey_quantity"],
            "key": "PRIMARY",
            "key_length": "4",
            "used_key_parts": ["l_orderkey"],
            "ref": ["dbt3sf1.orders.o_orderkey"],
            "r_loops": 1500000,
            "rows": 1,
            "r_rows": 4.0008,
            "r_total_time_ms": 170456,
            "filtered": 100,
            "r_filtered": 0,
            "attached_condition": "(lineitem.l_extendedprice > 1000000)"
        }
    }
}
```
ANALYZE FORMAT=JSON basics

- Structured like EXPLAIN FORMAT=JSON
  - Subquery – `query_block`
  - Table access – `table`
  - Also nodes for `filesort`, `materialized`, `temptable`, etc

- ANALYZE data in each node
  - `r_members`
  - `r_rows`, `r_filtered` – like in tabular form
  - `r_loops` – number of times node executed
  - `r_total_time_ms` ← !!!
"table_name": "customer", "r_total_time_ms": 1354.8,
"table_name": "orders", "r_total_time_ms": 11444,
"table_name": "lineitem", "r_total_time_ms": 22040,
"table_name": "orders", "r_total_time_ms": 10493,

**#1**

- can be confusing with joins or subqueries

**#2**

- Not all rows are equal

**#3**

- **r_total_time_ms** shows which table/subquery took the time.
Buffer sizes

- join_buffer_size, sort_buffer_size
  - What size is used? Is it enough?

```
select * from t1, t2
where t1.col1<100 and t2.col1<100 and t1.col2=t2.col2
```

Join buffer optimization

- Reads rows into buffer, then sorts
- Can see it used in EXPLAIN
  - But what about buffer size?
ANALYZE and buffer sizes

Can see buffers used

- Can see used buffer size

- Child's `r_loops` gives a clue whether buffer refilled.
Range checked for each record

```sql
select * from orders A, orders B
where
    A.o_clerk='Clerk#000000001' and
    B.o_orderdate between DATE_SUB(A.o_orderdate, interval 1 day) and
    DATE_ADD(A.o_orderdate, interval 1 day)
and
    B.o_shipdate between DATE_SUB(A.o_shipdate, interval 1 day) and
    DATE_ADD(A.o_shipdate, interval 1 day)
```

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>A</td>
<td>ref</td>
<td>i_o_order_clerk_date</td>
<td>i_o_clerk</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>B</td>
<td>ALL</td>
<td>i_o_orderdate, o_shipDATE</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>1466</td>
<td>Using index condition</td>
</tr>
<tr>
<td>NULL</td>
<td>1499649</td>
<td>Range checked for each record (index map: 0x22)</td>
</tr>
</tbody>
</table>
Range checked for each record (2)

- **r_keys** shows what keys were useful
  - Not useful? IGNORE INDEX
• EXPLAIN lies!
  Bug#69795: EXPLAIN FORMAT=JSON doesn't show Using filesort for UNION
  Bug#74462: EXPLAIN FORMAT=JSON produces ordering_operation when no ordering ...
  Bug#74661: EXPLAIN FORMAT=JSON says two temptables are used, execution shows just one
  Bug#74744: EXPLAIN FORMAT=JSON produces duplicates_removal where there is none
  Bug#76679: EXPLAIN incorrectly shows Distinct for tables using join buffer

• Scared yet?

• Both Oracle and MariaDB are working to fix this

• ANALYZE FORMAT=JSON
  - Is already available
  - Cannot lie, by design.
ANALYZE FORMAT=JSON data

- **Query plan node**
  - `r_loops` – number of executions
  - `r_total_time_ms` – total time spent
  - `r_rows` – number of rows read

- **Conditions**
  - `r_filtered` - %rows left after conditions

- **Join buffer, filesort, etc**
  - `r_buffer_size` – Buffer size used

- **filesort**
  - `r_limit`, `r_output_rows`
  - `r_priority_queue_used`

- **Subquery cache**
  - `r_hit_ratio`

- **Range checked for each record**
  - `r_indexes`

- **...**
Summary

- MariaDB 10.1 GA
- New statements
  - `ANALYZE <statement>`
  - `ANALYZE FORMAT=JSON <statement>`
- Gives EXPLAIN + execution statistics
  - Allows to spot query optimization problems.
Thanks!