How a database optimizer gets your data, fast

Vicențiu Ciorbaru
Software Developer Team Lead
@ MariaDB Foundation

Percona Live Europe 2019
30 Sept - 2 Oct Amsterdam, Netherlands
whoami

- Vicențiu Ciorbaru
- MariaDB Foundation, Software Developer Team Lead
- MariaDB developer since 2013-
- Implemented Roles, Window Functions and others
Goal of a query optimizer

● Produce a query plan that executes your query in the fastest time possible.

● Optimizer has many tools at its disposal:
  ○ It can choose to pre-read tables
  ○ Cache results (such as uncorrelated subqueries)
  ○ Use indexes to look up values
  ○ Use indexes to access data in-order and avoid sorting
  ○ Rewrite a query (more on this later)
  ○ And more...

● Number of possible plans grows exponentially with # tables
Goal of a query optimizer

- Not enough time to try out every possible plan

- In a "perfect world" any query should be performing as fast as possible.

- Many queries do!

- But sometimes...
Statistics based optimizations
What optimizer can ask the engine

- “how many rows does the table have?”
- “what would it cost to scan the whole table?”
- “what would it cost to read that many rows from this index?”
- “how many distinct key values are in this index?”
- “how many keys lie in that range of values in this index?”
A problem: instability

- **DBT-3 Q8**: National Market Share Query — 8 InnoDB tables
  - 4 different plans
  - from 7 minutes to 1.2 hours

- **DBT-3 Q7**: Volume Shipping Query — 6 InnoDB tables
  - 7 different plans
  - from 12 minutes to many hours (and timeout)
A problem: instability

- **DBT-3 Q8**: National Market Share Query — 8 InnoDB tables
  - 4 different plans
  - from 7 minutes to 1.2 hours

- **DBT-3 Q7**: Volume Shipping Query — 6 InnoDB tables
  - 7 different plans
  - from 12 minutes to many hours (and timeout)

- What about InnoDB persistent statistics?
A problem: All engines are liars

MariaDB> CREATE TABLE t1 (a INT, b INT, c INT, KEY(a,b)) ENGINE=MyISAM;
MariaDB> INSERT t1 VALUES (RAND()*1000000, RAND()*1000000, RAND()*1000000);
MariaDB> ... 400 000 rows ...

MariaDB> SELECT COUNT(DISTINCT a) AS cardinality FROM t1;

+----------+
| cardinality |
+----------+
|        97794 |
+----------+
A problem: All engines are liars

MariaDB> SELECT cardinality FROM information_schema.statistics --- WHERE table_name='t1' AND column_name='a';

+-------------+
| cardinality |
+-------------+
|       98304 |
+-------------+

MariaDB> ALTER TABLE t1 ENGINE=InnoDB;

MariaDB> SELECT cardinality FROM information_schema.statistics --- WHERE table_name='t1' AND column_name='a';

+-------------+
| cardinality |
+-------------+
|      196914 |
+-------------+
A problem: An index is required

- Takes storage space
- Needs to be updated for every INSERT / UPDATE / DELETE
- More indexes make INSERT / UPDATE / DELETE slow
A problem: An index is required

- Takes storage space
- Needs to be updated for every INSERT / UPDATE / DELETE
- More indexes make INSERT / UPDATE / DELETE slow
- Too expensive if you only need statistics!
Solution: Engine Independent Table Statistics

- **Stable** - Solves Instability
- **Precise** - Solves Storage Engine lying
- **Detailed** - Stores more information that most storage engines
- **Identical** for all engines
- **Comparable** - Format is "humanly readable"
How does it work?

- new tables in the mysql schema
- @@use_stat_tables = [ never | complementary | preferably ]
- @@optimizer_use_condition_selectivity = 1 ... 5
- ANALYZE TABLE ... [ PERSISTENT FOR ... ]
An example is worth a thousand words...

- https://github.com/datacharmer/test_db
- 300,000 employees, 2.8 millions salary payments, 167 MB of data

```
MariaDB> source employees.sql

MariaDB> SET USE_STAT_TABLES=PREFERABLY;
Query OK, 0 rows affected (0.00 sec)

MariaDB> ANALYZE TABLE departments, dept_emp, dept_manager,
-> employees, salaries, titles;
12 rows in set (18.49 sec)
```
Example: per-column statistics

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=1;
Query OK, 0 rows affected (0.01 sec)

MariaDB> SELECT * FROM departments JOIN dept_emp USING (dept_no)
   -> JOIN employees USING (emp_no)
   -> JOIN titles USING (emp_no)
   -> WHERE title='Manager';
24 rows in set (15.13 sec)
Example: per-column statistics

```sql
MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=1;
Query OK, 0 rows affected (0.01 sec)

MariaDB> SELECT * FROM departments JOIN dept_emp USING (dept_no)
    -> JOIN employees USING (emp_no)
    -> JOIN titles USING (emp_no)
    -> WHERE title='Manager';
24 rows in set (15.13 sec)

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;
Query OK, 0 rows affected (0.01 sec)

MariaDB> SELECT * FROM departments JOIN dept_emp USING (dept_no)
    -> JOIN employees USING (emp_no)
    -> JOIN titles USING (emp_no)
    -> WHERE title='Manager';
24 rows in set (0.53 sec)
```
Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>Using index condition</td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>14.29</td>
<td>Using where</td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>Using where; Using join</td>
</tr>
</tbody>
</table>

MariaDB Foundation
https://mariadb.org
## Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>Using index condition</td>
</tr>
</tbody>
</table>

```
MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>14.29</td>
<td>Using where</td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>Using where; Using join</td>
</tr>
</tbody>
</table>
Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>Using index condition</td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>14.29</td>
<td>Using where</td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>Using where; Using join</td>
</tr>
</tbody>
</table>
Equi-height Histograms
Equi-height Histograms
Example: non-uniform distribution

MariaDB> SELECT (MIN(salary)+MAX(salary))/2 FROM salaries;
+-----------------------------+
| 98421.5000 |
+-----------------------------+

MariaDB> SELECT * FROM employees JOIN salaries USING (emp_no)
   -> WHERE salary > 100000;
166014 rows in set (18.79 sec)

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=4;
Query OK, 0 rows affected (0.00 sec)

MariaDB> SELECT * FROM employees JOIN salaries USING (emp_no)
   -> WHERE salary > 100000;
166014 rows in set (6.64 sec)
Example: per-column statistics

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>48.68</td>
<td>Using where</td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=4;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>salaries</td>
<td>ALL</td>
<td>NULL</td>
<td>2844047</td>
<td>4.76</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
Example: per-column statistics

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>48.68</td>
<td>Using where</td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=4;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>salaries</td>
<td>ALL</td>
<td>NULL</td>
<td>2844047</td>
<td>4.76</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
Example: per-column statistics

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=3;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>titles</td>
<td>ALL</td>
<td>NULL</td>
<td>443308</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>48.68</td>
<td></td>
</tr>
</tbody>
</table>

Using where

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=4;

<table>
<thead>
<tr>
<th>id</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>salaries</td>
<td>ALL</td>
<td>NULL</td>
<td>2844047</td>
<td>4.76</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>titles</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

Using where
Example: sampling

```
MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=4;

MariaDB> SELECT dept_name, AVG(salary)
  --> FROM departments JOIN dept_emp USING (dept_no)
  -->       JOIN employees USING (emp_no)
  -->       JOIN salaries USING (emp_no)
  --> WHERE last_name LIKE '%off' GROUP BY dept_name;
9 rows in set (5.06 sec)

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=5;

MariaDB> SELECT dept_name, AVG(salary)
  --> FROM departments JOIN dept_emp USING (dept_no)
  -->       JOIN employees USING (emp_no)
  -->       JOIN salaries USING (emp_no)
  --> WHERE last_name LIKE '%off' GROUP BY dept_name;
9 rows in set (0.40 sec)
```
Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td>temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>where</td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=5;

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>employees</td>
<td>ALL</td>
<td>NULL</td>
<td>300024</td>
<td>1.00</td>
<td>where; temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>index</td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>where; join buffer (flat,</td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
### Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td>temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

```sql
MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=5;
```

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>employees</td>
<td>ALL</td>
<td>NULL</td>
<td>300024</td>
<td>1.00</td>
<td>where; temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>index</td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>where; join buffer (flat, JOIN,...)</td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

[Link to MariaDB Foundation](https://mariadb.org)
Example: per-column statistics

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>100.00</td>
<td>temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>dept_no</td>
<td>36844</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>employees</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>where</td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

MariaDB> SET OPTIMIZER_USE_CONDITION_SELECTIVITY=5;

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>key</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>employees</td>
<td>ALL</td>
<td>NULL</td>
<td>300024</td>
<td>1.00</td>
<td>where; temporary; filesort</td>
</tr>
<tr>
<td>dept_emp</td>
<td>ref</td>
<td>PRIMARY</td>
<td>1</td>
<td>100.00</td>
<td>index</td>
</tr>
<tr>
<td>departments</td>
<td>ALL</td>
<td>NULL</td>
<td>9</td>
<td>77.78</td>
<td>where; join buffer (flat,</td>
</tr>
<tr>
<td>salaries</td>
<td>ref</td>
<td>PRIMARY</td>
<td>9</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>
MariaDB 10.0 (GA 2014)

- Per table: number of rows
- Per index: number of distinct values
- Per column:
  - number of distinct values
  - min and max values, amount of NULLs, average value length
- Equi-height histogram
- Sampling for LIKE predicates
MySQL 8.0 (GA 2018)

- Optimizer Statistics
- Per column:
  - Amount of NULLs
  - Equi-height and “singleton” histograms
  - Faster than MariaDB 10.0 on large tables
  - Skips values to keep the data set small
MariaDB 10.4 (GA 2019)

- Statistics Tables are Enabled by default

- Can collect histograms through sampling
  - Uses Bernoulli Sampling
  - Server variable `analyze_sample_percentage`
  - Set to 0 to let MariaDB decide how many rows to sample.

- Uses the unsmoothed first-order jackknife estimator to estimate total cardinality.

- At least as fast as MySQL 8.0 for large tables
Query rewriting
Background on optimizations

- A derived table is a table in the FROM clause, defined as a subquery.

```sql
SELECT * FROM (SELECT a FROM t1) der_t1;
```
VIP Customers and their orders

```sql
select *
from vip_customers,
    (select * 
        from orders 
        where order_date 
            between '2017-10-01' and '2017-10-31') as OCT_ORDERS
where OCT_ORDERS.amount > 1000000 and 
    OCT_ORDERS.customer_id = vip_customers.customer_id;
```
Naive execution

```sql
select *
from vip_customers,
(select * from orders
    where order_date between '2017-10-01' and '2017-10-31') as OCT_ORDERS
where OCT_ORDERS.amount > 1000000 and
    OCT_ORDERS.customer_id = vip_customers.customer_id;
```
Derived Table Merge

```
select *
from
    vip_customers vc,
    (select *
        from orders
        where
            order_date between '2017-10-01' and '2017-10-31'
    ) as OCT_ORDERS
where
    OCT_ORDERS.amount > 1M and
    OCT_ORDERS.customer_id = vc.customer_id;
```

```
select *
from
    vip_customers vc,
    orders o
where
    o.amount > 1M and
    o.customer_id = vc.customer_id
    and
    o.order_date between '2017-10-01' and '2017-10-31';
```
Explain shows table is merged

explain select *
from vip_customers,
(select *
from orders
where order_date
between '2017-10-01' and '2017-10-31') as OCT_ORDERS
where OCT_ORDERS.amount > 1000000 and
OCT_ORDERS.customer_id = vip_customers.customer_id;

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>vip_customers</td>
<td>ALL</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>orders</td>
<td>ALL</td>
<td>1000000</td>
<td>Using where;</td>
</tr>
</tbody>
</table>
Execution after merge

```sql
select *
from
  vip_customers vc,
  orders o
where
  o.amount > 1M and
  o.customer_id = vc.customer_id and
  o.order_date between '2017-10-01' and '2017-10-31';
```
Execution after merge

```sql
select * 
from
  vip_customers vc,
  orders o
where
  o.amount > 1M and
  o.customer_id = vc.customer_id and
  o.order_date between '2017-10-01' and '2017-10-31';
```

Merging is good! It simplifies the query!
Execution after merge

```sql
select *
from
    vip_customers vc,
    orders o
where
    o.amount > 1M and
    o.customer_id = vc.customer_id and
    o.order_date between '2017-10-01' and
    '2017-10-31';
```

Merging is good! It simplifies the query!

Works in all stable MariaDB & MySQL versions.

https://mariadb.org
Execution after merge

select *
from
  vip_customers vc,
  orders o
where
  o.amount > 1M and
  o.customer_id = vc.customer_id and
  o.order_date between '2017-10-01' and '2017-10-31';

Merging is good!
It simplifies the query!

Works in all stable MariaDB & MySQL versions.

Can not be used when aggregation is present. :(

Can not be used when aggregation is present. :(

MariaDB Foundation
https://mariadb.org
create view OCT_TOTALS as
select customer_id, SUM(amount) as TOTAL_AMT
from orders
where order_date between '2017-10-01' and '2017-10-31'
group by customer_id

select *
from OCT_TOTALS
where customer_id=1
create view OCT_TOTALS as
select customer_id, SUM(amount) as TOTAL_AMT
from orders
where order_date between '2017-10-01' and '2017-10-31'
group by customer_id

There are a lot of customers and we only want the data for 1.
CREATE VIEW OCT_TOTALS AS
SELECT customer_id, SUM(amount) AS TOTAL_AMT
FROM orders
WHERE order_date BETWEEN '2017-10-01' AND '2017-10-31'
GROUP BY customer_id

SELECT *
FROM OCT_TOTALS
WHERE customer_id = 1

We can push the condition to the WHERE clause!
create view OCT_TOTALS as
select customer_id, SUM(amount) as TOTAL_AMT
from orders
where order_date between '2017-10-01' and '2017-10-31'
group by

select *
from OCT_TOTALS
where customer_id=1

This tactic works for Window Functions too!
create view top_three_orders as
select *
from (
    select customer_id, amount,
         rank() over (partition by customer_id
                      order by amount desc) as order_rank
    from orders)
    as ranked_orders
where order_rank < 3

+-------------+--------+------------+
| customer_id | amount | order_rank |
+-------------+--------+------------+
| 1           | 10000  | 1          |
| 1           | 9500   | 2          |
| 1           | 400    | 3          |
| 2           | 3200   | 1          |
| 2           | 1000   | 2          |

select * from top_three_orders where customer_id = 1
create view top_three_orders as
select * from (select customer_id, amount,
    rank() over (partition by customer_id
                  order by amount desc) as order_rank
    from orders) as ranked_orders
where order_rank < 3

+-------------+--------+------------+
| customer_id | amount | order_rank |
+-------------+--------+------------+
| 1           | 10000  | 1          |
| 1           | 9500   | 2          |
| 1           | 400    | 3          |
| 2           | 3200   | 1          |
| 2           | 1000   | 2          |
+-------------+--------+------------+

select * from top_three_orders where customer_id = 1
MariaDB - MySQL Comparison

MariaDB 10.2, MySQL 8.0

- Compute top_three_orders for all customers
- Select rows with customer_id = 1

MariaDB 10.3 and onwards (and e.g. PostgreSQL)

- Only compute top_three_orders for customer_id=1
- This can be much faster!
- Can make use of index(customer_id)
This talk would not have been possible if it not for the MariaDB Foundation's sponsors.

Sponsors of MariaDB Foundation

- Alibaba Cloud
- Booking.com
- Tencent Cloud
- Microsoft
- DBS
- Visma
- Tencent Games
- MariaDB
- IBM
- Acronis
- TDSQL
- Percona
- nexedi
- Tencent Game DBA
- Virtuozzo
- Verkkokauppa.com
Thank you!

Contact details:

vicentiu@mariadb.org

About:

https://mariadb.org/vicentiu