The MariaDB/MySQL Query Executor In-depth

Presented by: Timour Katchaounov

Optimizer team: Igor Babaev, Sergey Petrunia, Timour Katchaounov

MariaDB

Monty Program
Outline

What's **IN**
- Query engine architecture
- Execution model
- Representation of query execution plans (QEPs)
- Single-table access methods
- Join methods
- Questions

What's **NOT IN**
- Query optimization
- Subquery execution
- Sorting/Grouping/Distinct
- Prepared statements
- Stored procedures
- INSERT/UPDATE/DELETE

Comparing latest development versions: MariaDB 5.3 vs. MySQL 5.5

MariaDB

Monty Program
Query engine architecture [MariaDB 5.3]

- Parser, Preprocessor
- Query optimizer (rewrites, cost-based)
- Plan Refinement
- Query Executioner
- Storage engine API
- Storage engine(s)

Constant table optimization, MIN/MAX/COUNT
Query plan 'shape' – bushy vs linear

General bushy query plans

Left-deep query plans
MariaDB 5.3: bushy query plans with semi-join (IN subqueries), and derived tables
Query plans as operator sequences

T1 -> I2 -> T2

T3

J123

J12

T1

T2

T4

I4

JOIN_TAB

JOIN_TAB

JOIN_TAB

JOIN_TAB

JOIN

JOIN_TAB

JOIN_TAB

JOIN_TAB

JOIN_TAB
### Query plans and EXPLAIN

<table>
<thead>
<tr>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>ALL</td>
<td>PRIMARY</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>984</td>
</tr>
<tr>
<td>T2</td>
<td>range</td>
<td>K1, K2</td>
<td>K2</td>
<td>3</td>
<td>NULL</td>
<td>30</td>
</tr>
<tr>
<td>T3</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>C3</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>4</td>
<td>C4</td>
<td>1</td>
</tr>
</tbody>
</table>

### Monty Program

[Diagram of database query plan]
Example database and query

```sql
SELECT Country.name, City.name, City.population
FROM Country, City
WHERE City.country = Country.code and
      City.id = Country.capital and
      City.population > 5000000 and
      City.name LIKE 'SAN%';
```
**1. row**

- **id**: 1
- **select_type**: SIMPLE
- **table**: City
- **type**: range
- **possible_keys**: PRIMARY, country, population
- **key**: population
- **key_len**: 4
- **ref**: NULL
- **rows**: 25
- **Extra**: Using index condition; Using where; Using MRR

**2. row**

- **id**: 1
- **select_type**: SIMPLE
- **table**: Country
- **type**: eq_ref
- **possible_keys**: PRIMARY
- **key**: PRIMARY
- **key_len**: 3
- **ref**: world.City.CountryCode
- **rows**: 1
- **Extra**: Using where
Plan operators

condition

LIKE

Name 'SAN%'

record_buffer

<table>
<thead>
<tr>
<th>Id</th>
<th>Population</th>
<th>Country</th>
<th>Name</th>
</tr>
</thead>
</table>

table
country

access_method

index: Population
lower_bound: 5000000
upper_bound: infinity
range for (population > 5M)

join_method

procedure index_nested_loops_join

file: sql/sql_select.h, class JOIN_TAB
QEPs and nested loop join execution

**Query engine**

AND

> 5M

LIKE 'SAN%'

Pop < 5M

range

record buffer for City

**Storage engine**

index Population

City

index Code

Country

**Code**

Name

Surface

Pop

Capital

eq_ref key buffer for Country.Code

record buffer for Country
procedure nested_loops_join
input: <OP_i, ..., OP_n> // remaining QEP operators no yet joined
{
    if (init_record_scan(Table_i, Access_method_i) == EOF)
        return

    while (curr_record_i = get_next_record(Table_i, Access_method_i))
    {
        joined_record = <record_buffer_1 || ... || record_buffer_i>
        if join_condition_i(joined_record) /* Test the join condition. */
        {
            if joined_record is a complete result record (i.e. i = n)
                output joined_record
            else
                nested_loops_join(OP_[i+1], ..., OP_n)
        }
    }
}
enum_nested_loop_state sub_select(JOIN_TAB *remainder)
{
    error= (*join_tab->read_first_record)(join_tab);
    rc= evaluate_join_record(join, join_tab, error);
    while (rc == NESTED_LOOP_OK)
    {
        error= info->read_record(info);
        rc= evaluate_join_record(join, join_tab, error);
    }
}
enum_nested_loop_state evaluate_join_record(JOIN_TAB *remainder)
{
    found= test(select_cond->val_int());
    If (found)
    {
        rc= (*join_tab->next_select)(join, join_tab+1, 0);
    }
}
The pull-push execution model

Plan Operator

Row buffer

next

nested loops

Plan Operator

Row buffer

next

Plan Operator

Row buffer

next

Data sink:
client, temp table

send

Query engine

Handler

Send

Storage engine

Table

Table

Table
The pull-push execution model

Data sink: client, temp table

Plan Operator

Table

Row buffer

next

nested loops

Plan Operator

Table

Row buffer

next

nested loops

Plan Operator

Table

Row buffer

next

send

Handler

Query engine

Storage engine

Plan Operator

Table

Row buffer

next

nested loops

Plan Operator

Table

Row buffer

next

nested loops

Plan Operator

Table

Row buffer

next

nested loops
The pull-push execution model

Data sink: client, temp table

Plan Operator

Row buffer

Table

Plan Operator

Row buffer

Table

Plan Operator

Row buffer

Table

nested loops

next

send

Query engine

Handler

Storage engine

Monty Program
The pull-push execution model

Data sink: client, temp table

Plan Operator

Row buffer

next

Plan Operator

Row buffer

next

Plan Operator

Row buffer

next

Table

Table

Table

Storage engine

Query engine

Monty Program

MariaDB
Thank you

Questions?