Let's talk Database Optimizers

Vicențiu Ciorbaru
Software Engineer @ MariaDB Foundation
vicentiu@mariadb.org
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, the query optimizer doesn't have all the information. (missing indexes, inaccurate statistics, etc.)

- Optimizers are constantly evolving!
A derived table is a table in the FROM clause, defined as a subquery.

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

```
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;
```

![Diagram showing Naive Execution](diagram.png)
Derived Table Merge

```sql
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;
```

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

```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;
```

16649 rows in set (7.64 sec)
Execution after merge

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

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

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

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

Works in all stable MySQL and MariaDB versions
Execution after merge

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

Can not be used when aggregation is present :(

```
orders
  OCT_ORDERS
  amount > 1000000
JOIN
vip_customers
RESULT
```
Condition pushdown

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
Condition pushdown

```
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
```

There are a lot of customers and we only want a total for one.
Condition pushdown

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_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

select *
from OCT_TOTALS
where customer_id=1

All this is available in MariaDB 10.2
Condition pushdown

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

This tactic works with 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 ordered_orders where order_rank < 3
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 ordered_orders
where order_rank < 3

+-------------+--------+------------+
<table>
<thead>
<tr>
<th>customer_id</th>
<th>amount</th>
<th>order_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9500</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>400</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3200</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>3</td>
</tr>
</tbody>
</table>
```sql
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 ordered_orders  
where order_rank < 3
```

<table>
<thead>
<tr>
<th>customer_id</th>
<th>amount</th>
<th>order_rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9500</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>400</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3200</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>3</td>
</tr>
</tbody>
</table>

```
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 ordered_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 |
|           2 |    400 |          3 |
+-------------+--------+------------+

select * from top_three_orders where customer_id=1
Condition pushdown through PARTITION BY

<table>
<thead>
<tr>
<th>MariaDB 10.2, MySQL 8.0</th>
<th>MariaDB 10.3 (and e.g. PostgreSQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute top_three_orders for all customers</td>
<td>Only compute top_three_orders for customer_id=1</td>
</tr>
<tr>
<td>Select rows with customer_id=1</td>
<td>This can be much faster!</td>
</tr>
<tr>
<td></td>
<td>Can make use of index(customer_id)</td>
</tr>
</tbody>
</table>
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 customers, OCT_TOTALS
where customers.customer_id=OCT_TOTALS.customer_id and customers.customer_name IN ('John', 'Bob')
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 customers, OCT_TOTALS
where customers.customer_id=OCT_TOTALS.customer_id and customers.customer_name IN ('John', 'Bob')
Split grouping for derived

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 customers, OCT_TOTALS
where customers.customer_id=OCT_TOTALS.customer_id and customers.customer_name IN ('John', 'Bob')
Split grouping execution

Figure out which orders we need to aggregate first!

Aggregate each group individually.

customers

Bob

John

orders

Bob

John

SUM

SUM
Split grouping execution requirements

■ Can be used when doing join from customer to orders

■ Must have equalities for GROUP BY columns:
  OCT_TOTALS.customer_id=customer.customer_id
  ○ This allows to select one group

■ The underlying table (orders) must have an index on the GROUP BY column (customer_id)
  ○ This allows to use ref access
## Conclusions

<table>
<thead>
<tr>
<th></th>
<th>MySQL 5.7</th>
<th>MySQL 8.0</th>
<th>MariaDB 10.1</th>
<th>MariaDB 10.2</th>
<th>MariaDB 10.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived Table / View Merge</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Condition Pushdown through Group BY</td>
<td>✕</td>
<td>✕</td>
<td>✕</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Window Functions</td>
<td>✕</td>
<td>✔</td>
<td>✕</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Condition Pushdown through Partition BY</td>
<td>✕</td>
<td>✕</td>
<td>✕</td>
<td>✕</td>
<td>✔</td>
</tr>
<tr>
<td>Split Table Grouping</td>
<td>✕</td>
<td>✕</td>
<td>✕</td>
<td>✕</td>
<td>✔</td>
</tr>
</tbody>
</table>

Not comprehensive comparison, only optimizations discussed in this talk!
Conclusions

- **MariaDB 10.2:** *Condition pushdown for derived tables optimization*
  - Push a condition into derived table
  - Used when derived table cannot be merged
  - Biggest effect is for subqueries with GROUP BY

- **MariaDB 10.3:** *Condition Pushdown through Window functions' partition by*

- **MariaDB 10.3:** *Split grouping for derived optimization*
  - When doing a join, can’t do condition pushdown
  - So, split grouping derived is used.
  - It allows to only examine GROUP BY groups that match other tables. It needs index on grouped columns
  - Work in progress (optimization process is very basic ATM)
Thank You!

Contact me at:
vicentiu@mariadb.org
vicentiu@ciorbaru.io

Blog:
mariadb.org/blog