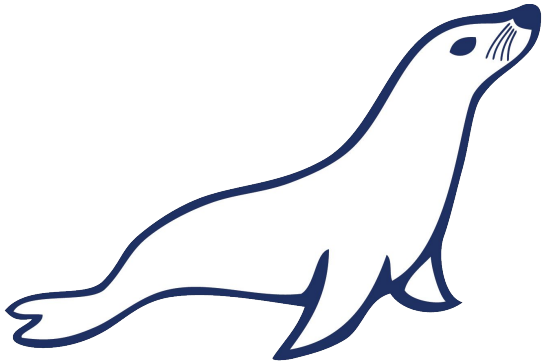


MariaDB®
FOUNDATION

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!

Background about optimizations

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

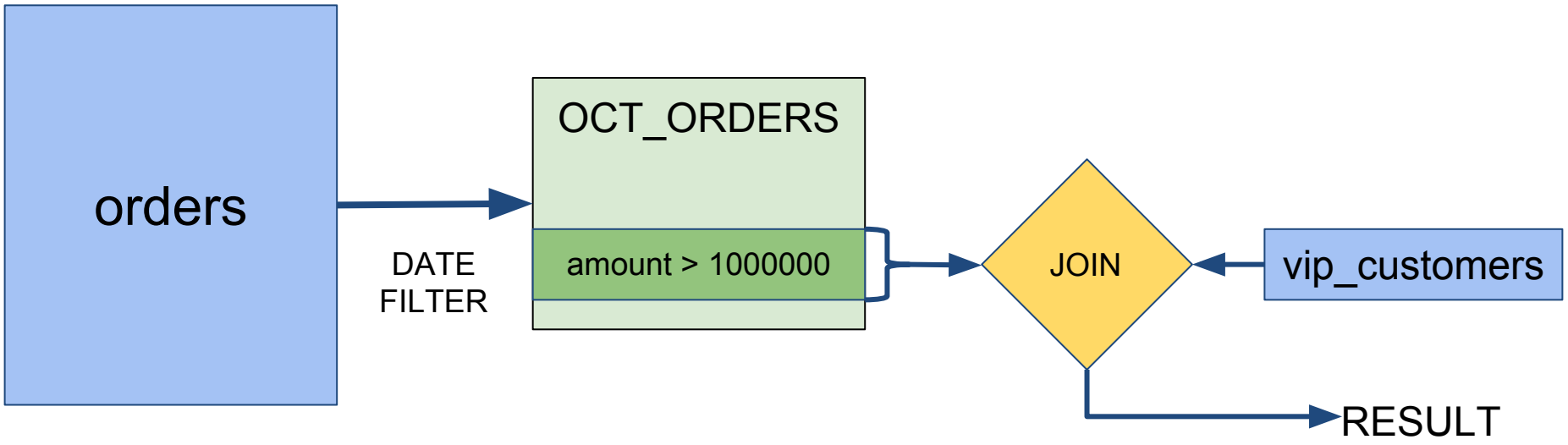
```
SELECT * FROM (SELECT a from t1) der_t1;
```

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

```
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
where
  OCT_ORDERS.amount > 1M and
  OCT_ORDERS.customer_id =
          vc.customer_id and
  order_date between
   '2017-10-01' and '2017-10-31';
```

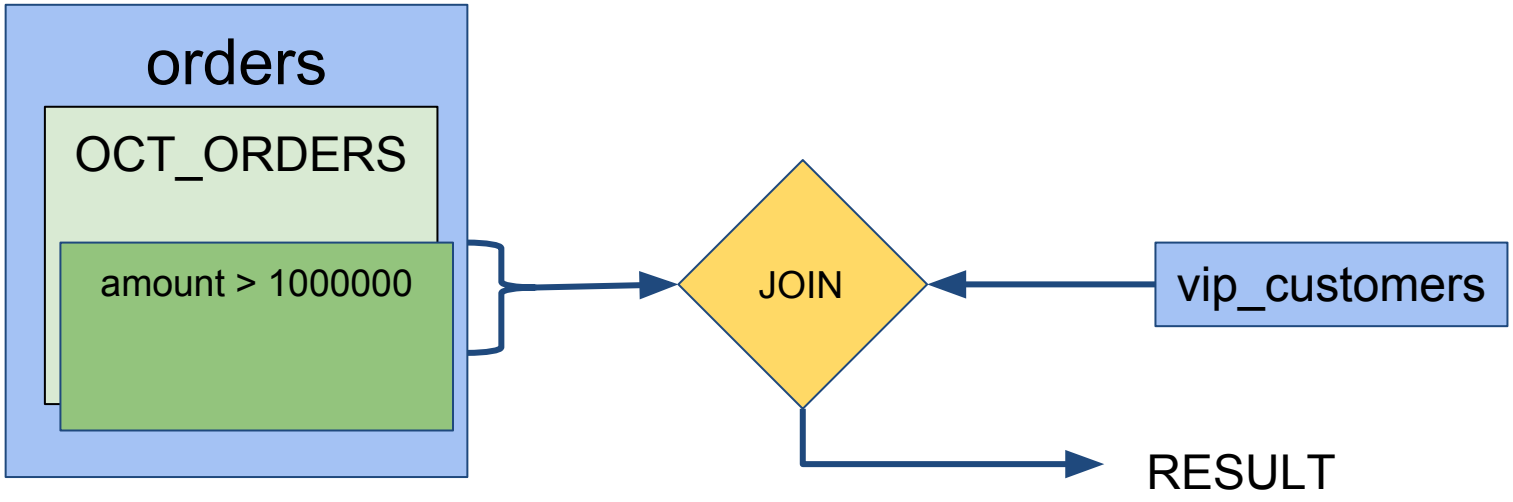
Explain shows the table being merged

```
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)
```

```
+----+-------------+---------------+------+..+---------+-------------+
|id  | select_type | table         | type |..| rows    | Extra       |
+----+-------------+---------------+------+..+---------+-------------+
|  1 | SIMPLE      | vip_customers | ALL  |..|     101 |             |
|  1 | SIMPLE      | orders        | ALL  |..| 1000000 | Using where;|
+----+-------------+---------------+------+..+---------+-------------+
```

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';
```

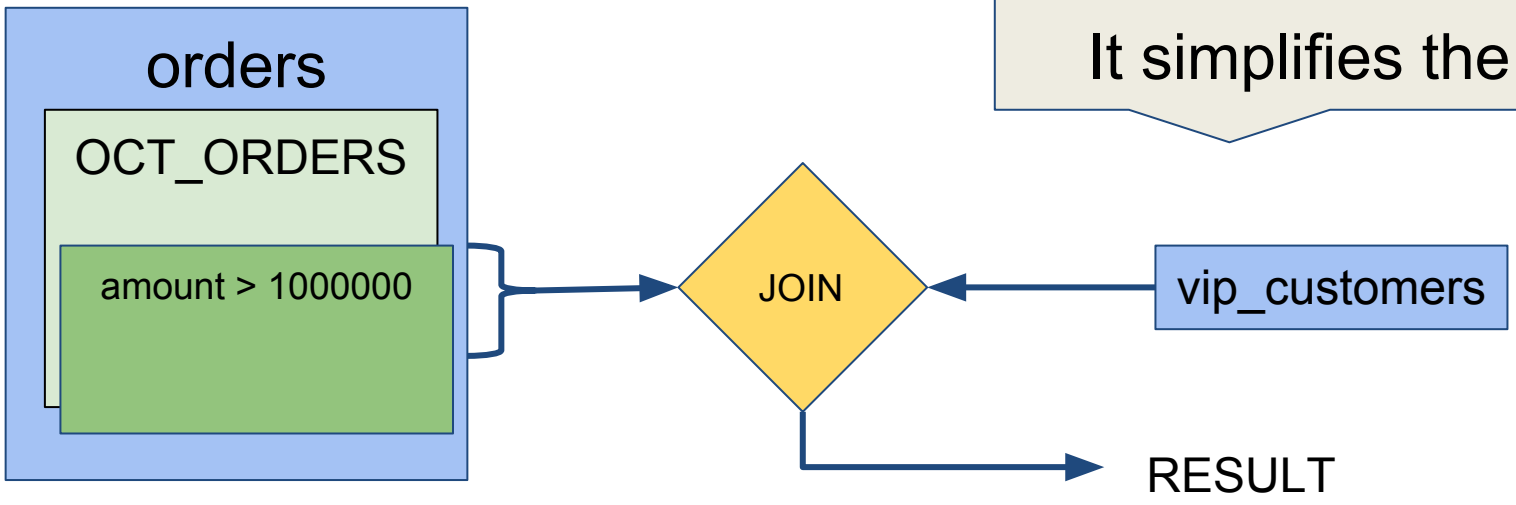


Execution after merge

```
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
```

Merging is good!

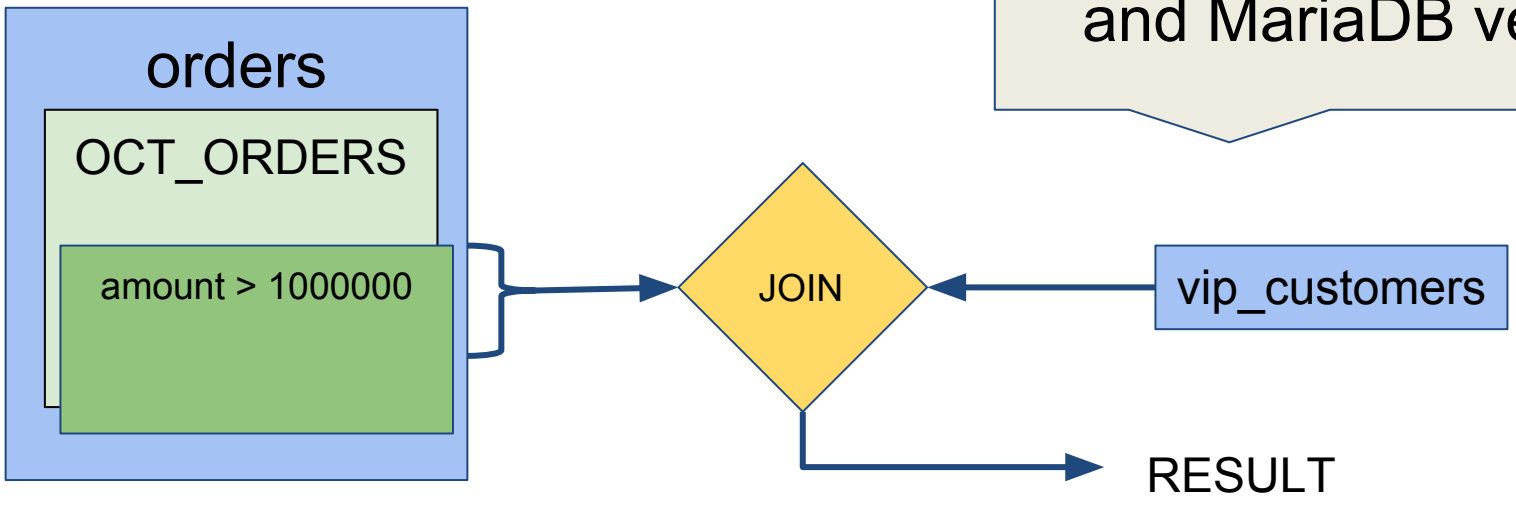
It simplifies the query.



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
```

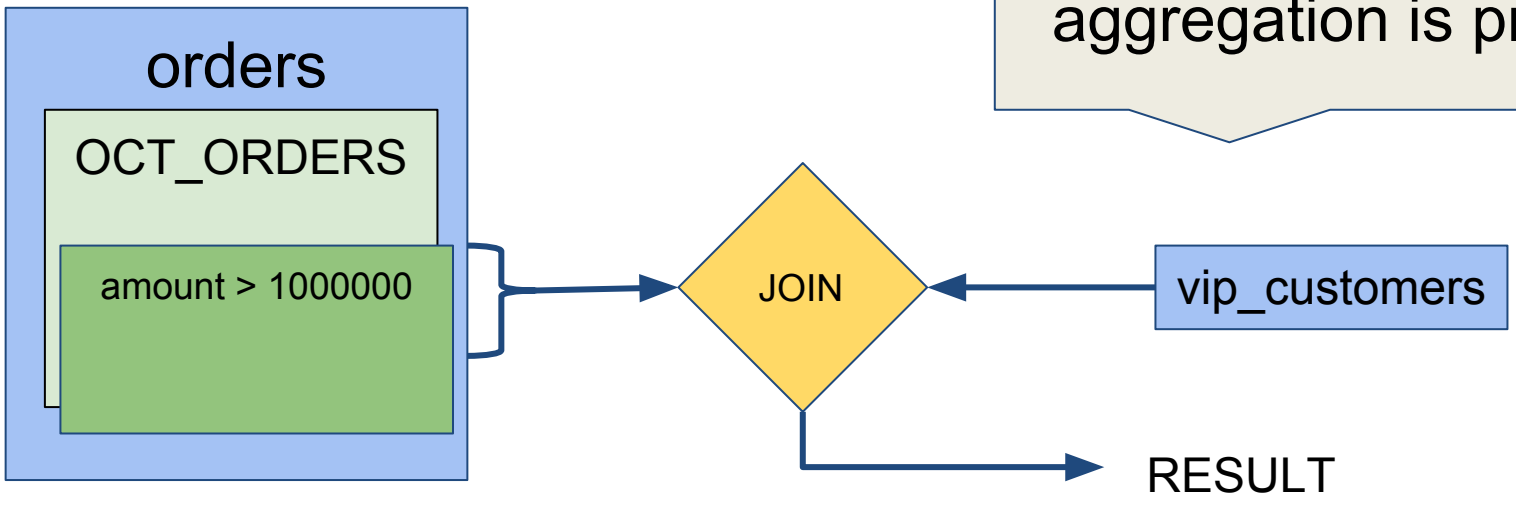
Works in all stable MySQL and MariaDB versions



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
```

Can not be used when aggregation is present :(



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
```

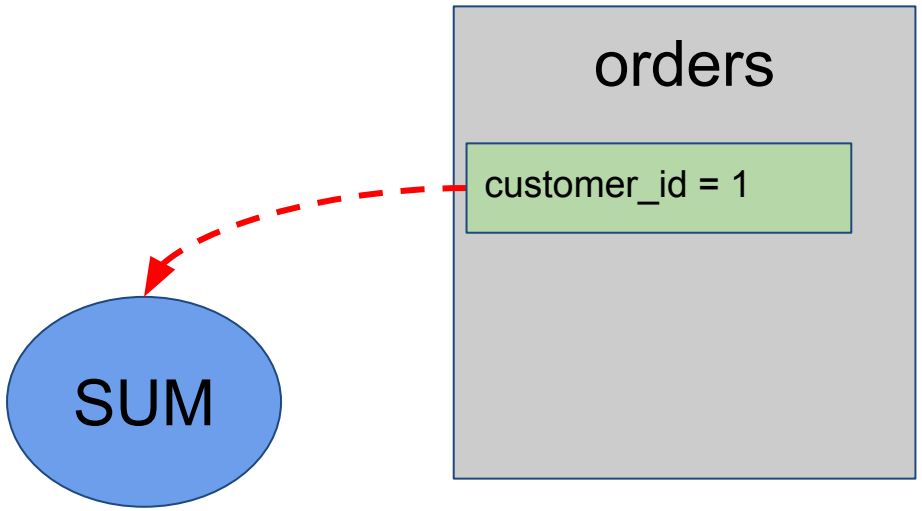
```
select *
from OCT_TOTALS
where customer_id=1
```

We can push the condition to the where clause!

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
```

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!

Condition pushdown through Partition By

```
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
```

Condition pushdown through Partition By

```
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 |
.....................................
```

Condition pushdown through Partition By

```
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

```
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

MariaDB 10.2, MySQL 8.0, MariaDB 10.3 Comparison

MariaDB 10.2, MySQL 8.0

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

MariaDB 10.3 (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)

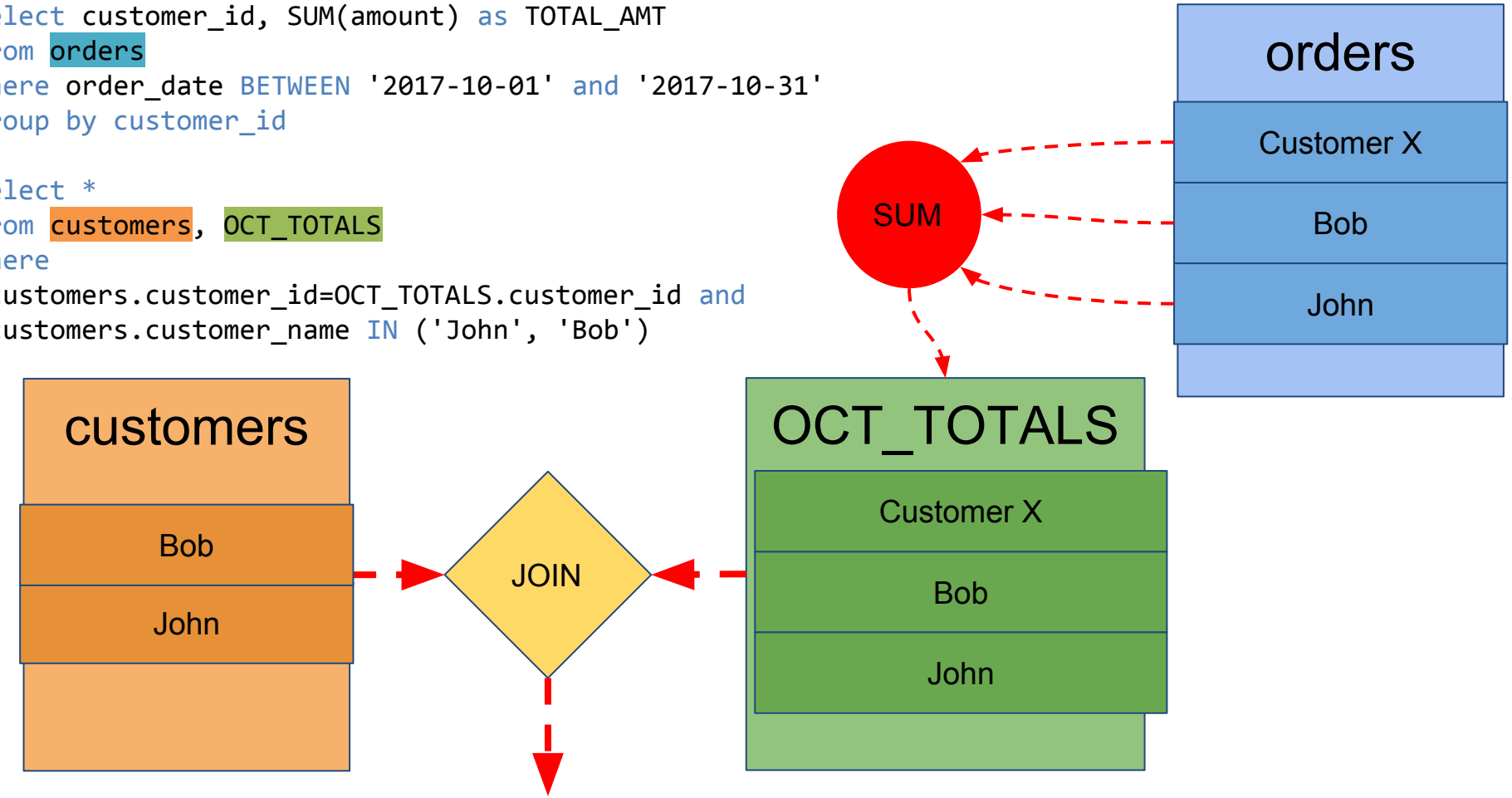
Split grouping for derived

```
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')
```

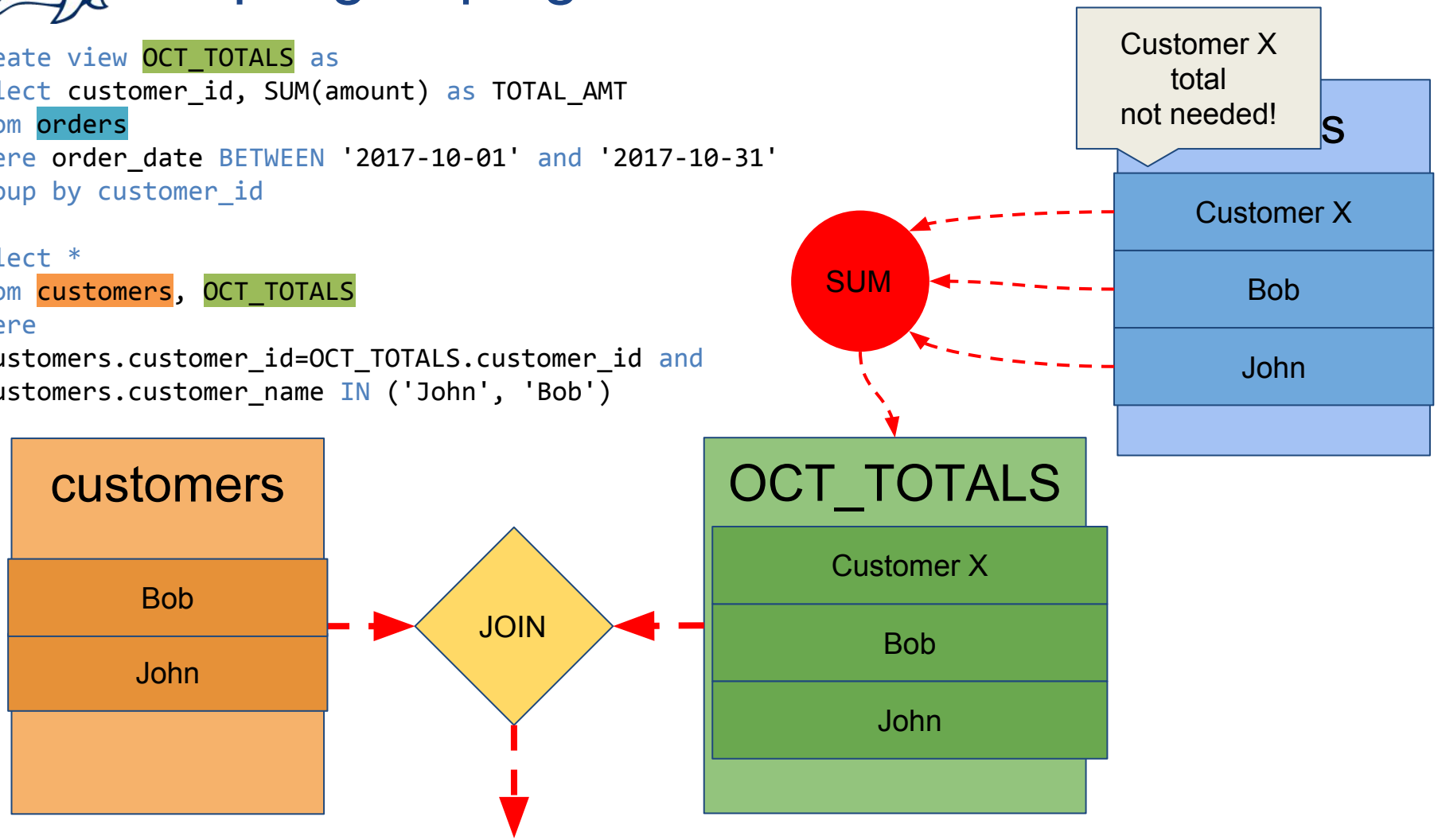
Split grouping for derived

```
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 for derived

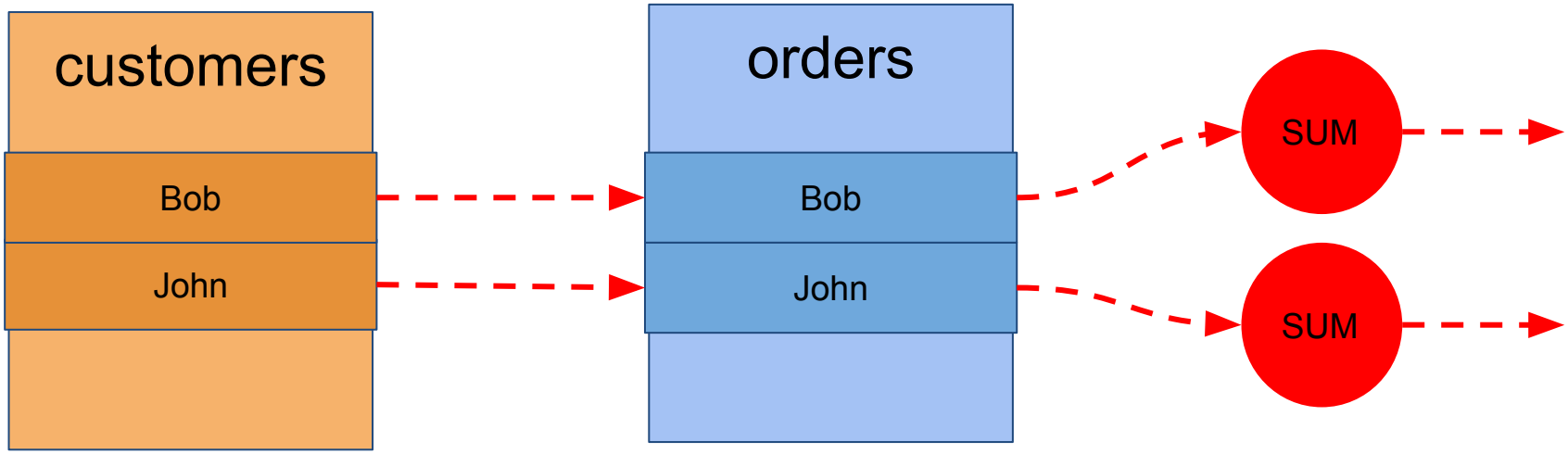
```
create view OCT_TOTALS as
select customer_id, SUM(amount) as TOTAL_AMT
from orders
where order_date BETWEEN '2017-10-01' and '2017-10-31'
group by customer_id

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.



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

	MySQL 5.7	MySQL 8.0	MariaDB 10.1	MariaDB 10.2	MariaDB 10.3
Derived Table / View Merge	✔	✔	✔	✔	✔
Condition Pushdown through Group BY	✘	✘	✘	✔	✔
Window Functions	✘	✔	✘	✔	✔
Condition Pushdown through Partition BY	✘	✘	✘	✘	✔
Split Table Grouping	✘	✘	✘	✘	✔

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