Optimizer in 10.2 and 10.3

Vicențiu Ciorbaru
Software Engineer @ MariaDB Foundation
What's new in MariaDB Optimizer

- Most features in 10.3 are additions over 10.2 features.

- Improved support / optimizations for CTEs and Window Functions
What are CTEs?

Syntax

```sql
WITH engineers AS (  
    SELECT *  
    FROM employees  
    WHERE dept="Engineering"  
)  
SELECT *  
FROM engineers  
WHERE ...
```
What are CTEs?

```
WITH engineers AS ( 
    SELECT * 
    FROM employees 
    WHERE dept="Engineering"
) 
SELECT * 
FROM engineers 
WHERE ...
```
What are CTEs?

Syntax

WITH engineers AS (  
    SELECT *  
    FROM employees  
    WHERE dept="Engineering"  
)  
SELECT *  
FROM engineers  
WHERE ...
What are CTEs?

Syntax

```sql
WITH engineers AS (  
    SELECT *  
    FROM employees  
    WHERE dept="Engineering" 
  )  
SELECT *  
FROM engineers  
WHERE ...  
```

CTE Body
What are CTEs?

Syntax

WITH engineers AS (  
  SELECT *  
  FROM employees  
  WHERE dept="Engineering"
)  
SELECT *  
FROM engineers  
WHERE ...

CTE Usage
What are CTEs?

CTEs are similar to derived tables.

WITH engineers AS (
    SELECT *
    FROM employees
    WHERE dept="Engineering"
)
SELECT *
FROM engineers
WHERE ...

SELECT *
FROM (SELECT *
    FROM employees
    WHERE dept="Engineering") AS engineers
WHERE ...
What are CTEs?

CTEs are more readable than derived tables.

WITH engineers AS (  
    SELECT *  
    FROM employees  
    WHERE dept="Engineering"
),

eu_engineers AS (  
    SELECT *  
    FROM engineers  
    WHERE country IN ("CN",...) 
)

SELECT *  
FROM eu_engineers  
WHERE ...

SELECT *  
FROM (SELECT *  
    FROM (SELECT *  
        FROM employees  
        WHERE dept="Engineering") AS engineers  
    WHERE country IN ("CN",...))
WHERE ...
What are CTEs?

CTEs are more readable than derived tables.

WITH engineers AS (  
   SELECT *  
   FROM employees  
   WHERE dept="Engineering"
),  
eu_engineers AS (  
   SELECT *  
   FROM engineers  
   WHERE country IN ("CN",...)  
  )  
SELECT *  
FROM eu_engineers  
WHERE ...

SELECT *  
FROM (SELECT *  
   FROM (SELECT *  
      FROM employees  
      WHERE dept="Engineering"
   ) AS engineers  
   WHERE country IN ("CN",...)  
)  
WHERE ...

Linear View

Nested View
What are CTEs?

Example: Year-over-year comparisons

WITH sales_product_year AS (  
    SELECT  
        product,  
        year(ship_date) as year,  
        SUM(price) as total_amt  
    FROM  
        item_sales  
    GROUP BY  
        product, year  
)  
SELECT *  
FROM  
    sales_product_year CUR,  
    sales_product_year PREV,  
WHERE  
    CUR.product = PREV.product AND  
    CUR.year = PREV.year + 1 AND  
    CUR.total_amt > PREV.total_amt
Recursive CTEs

- MariaDB also supports recursive references to CTEs
- Makes SQL language Turing Complete
- Ability to express hierarchical queries
  - Ex. List all employees below CTO
  - We are working on supporting CONNECT BY syntax from Oracle
Recursive CTEs

with recursive ancestors as (  
    select * from folks  
    where name = 'Alex'     <--- Base (Anchor) part  
    union [all]              <--- Keyword  
    select f.*              <--- Recursive part  
    from folks as f, ancestors AS a  
    where  
        f.id = a.father or f.id = a.mother  
)  
select * from ancestors;
Recursive CTEs

with recursive ancestors as (  
    select * from folks  
    where name = 'Alex'  
    union  
    select f.*  
    from folks as f, ancestors AS a  
    where  
        f.id = a.father or f.id = a.mother  
  )  
select * from ancestors;

+------+--------------+--------+--------+
| id   | name         | father | mother |
+------+--------------+--------+--------+
| 100  | Alex         |     20 |     30 |
|   20 | Dad          |     10 |   NULL |
|   30 | Mom          |   NULL |   NULL |
|   10 | Grandpa Bill |   NULL |   NULL |
|   98 | Sister Amy   |     20 |     30 |
+------+--------------+--------+--------+
Recursive CTEs

with recursive ancestors as (  
   select * from folks  
   where name = 'Alex'  
   union  
   select f.*  
   from folks as f, ancestors AS a  
   where  
      f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

+------+--------------+--------+--------+  
| id   | name         | father | mother |  
+------+--------------+--------+--------+  
|  100 | Alex         |     20 |     30 |  
|   20 | Dad          |     10 |   NULL |  
|   30 | Mom          |   NULL |   NULL |  
|   10 | Grandpa Bill |   NULL |   NULL |  
|   98 | Sister Amy   |     20 |     30 |  
+------+--------------+--------+--------+  

Step 2: Get values for anchor
Recursive CTEs

```
with recursive ancestors as (
    select * from folks
    where name = 'Alex'
    union
    select f.*
    from folks as f, ancestors AS a
    where
      f.id = a.father or f.id = a.mother
)
select * from ancestors;
```

Step 2: Get values for anchor

```
+------+--------------+--------+--------+
| id   | name         | father | mother |
| 100  | Alex         | 20     | 30     |
| 20   | Dad          | 10     | NULL   |
| 30   | Mom          | NULL   | NULL   |
| 10   | Grandpa Bill | NULL   | NULL   |
| 98   | Sister Amy   | 20     | 30     |
```
Recursive CTEs

Step 3: Compute recursive iteration

with recursive ancestors as (
    select * from folks
    where name = 'Alex'
    union
    select f.*
    from folks as f, ancestors AS a
    where
        f.id = a.father or f.id = a.mother
)
select * from ancestors;

+------+--------------+--------+--------+
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
+------+--------------+--------+--------+
Recursive CTEs

Step 3: Compute recursive iteration

with recursive ancestors as (  
    select * from folks  
    where name = 'Alex'  
    union  
    select f.*  
    from folks as f, ancestors AS a  
    where  
        f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

+------+--------------+--------+--------+  
| id   | name         | father | mother |  
+------+--------------+--------+--------+  
|  100 | Alex         |     20 |     30 |  
|  20  | Dad          |     10 |   NULL |  
|  30  | Mom          |   NULL |   NULL |  
|  10  | Grandpa Bill |   NULL |   NULL |  
|  98  | Sister Amy   |     20 |     30 |  
+------+--------------+--------+--------+  

| id  | name     | father | mother |  
+-----+----------+--------+--------+  
| 100 | Alex     |  20    |  30    |  
|  20 | Dad      |  10    | NULL   |  
|  30 | Mom      | NULL   | NULL   |  
|  10 | Grandpa  | NULL   | NULL   |  
|  98 | Sister   |  20    |  30    |  
+-----+----------+--------+--------+  

Recursive CTEs

with recursive ancestors as (  
  select * from folks  
  where name = 'Alex'  
  union  
  select f.*  
  from folks as f, ancestors AS a  
  where  
    f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

+------+--------------+--------+--------+  +-----------------------------+  
| id   | name         | father | mother |  | id   | name     | father | mother |  
+------+--------------+--------+--------+  +-----------------------------+  
|  100 | Alex         |     20 |     30 |  |  100 | Alex     |       |       |  
|   20 | Dad          |     10 |   NULL |  |     |          |       |       |  
|   30 | Mom          |   NULL |   NULL |  |     |          |       |       |  
|   10 | Grandpa Bill |   NULL |   NULL |  |     |          |       |       |  
|   98 | Sister Amy   |     20 |     30 |  |     |          |       |       |  
+------+--------------+--------+--------+  +-----------------------------+  

Step 3: Compute recursive iteration
Recursive CTEs

with recursive ancestors as (  
    select * from folks  
    where name = 'Alex'  
    union  
    select f.*  
    from folks as f, ancestors AS a  
    where  
        f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
with recursive ancestors as (  
    select * from folks  
    where name = 'Alex'  
    union  
    select f.*  
    from folks as f, ancestors AS a  
    where  
        f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
with recursive ancestors as (  
  select * from folks  
  where name = 'Alex'  
  union  
  select f.*  
  from folks as f, ancestors AS a  
  where  
    f.id = a.father or f.id = a.mother
)
select * from ancestors;
Recursive CTEs

with recursive ancestors as (  
  select * from folks  
  where name = 'Alex'  
  union  
  select f.*  
  from folks as f, ancestors AS a  
  where  
    f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Step 3: Compute recursive iteration

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
Recursive CTEs

Step 3: Compute recursive iteration

with recursive ancestors as (
    select * from folks
    where name = 'Alex'
    union
    select f.*
    from folks as f, ancestors AS a
    where
        f.id = a.father or f.id = a.mother
) select * from ancestors;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>
Recursive CTEs

Step 3: Compute recursive iteration

with recursive ancestors as (  
  select * from folks  
  where name = 'Alex'  
  union  
  select f.*  
  from folks as f, ancestors AS a  
  where  
    f.id = a.father or f.id = a.mother  
)  
select * from ancestors;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>father</th>
<th>mother</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Alex</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>Dad</td>
<td>10</td>
<td>NULL</td>
</tr>
<tr>
<td>30</td>
<td>Mom</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>Grandpa Bill</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>98</td>
<td>Sister Amy</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

No new rows! Done!
 Summary so far

- CTEs are essentially “query local views”
- Allow for greater optimization potential than views
- Can express hierarchical queries using recursion
What can window functions do?

- Can access multiple rows from the current row.
What can window functions do?

- Can access multiple rows from the current row.
- Eliminate self-joins.
What can window functions do?

- Can access multiple rows from the current row.
- Eliminate self-joins.
- Get faster running queries.
What are window functions?

- Similar to aggregate functions
  - Computed over a sequence of rows
- But they provide one result per row
  - Like regular functions!
- Identified by the OVER clause.
What are window functions?

Similar to regular functions

SELECT

    email, first_name,
    last_name, account_type
FROM users
ORDER BY email;
What are window functions?

Let’s start with a “function like” example

```
SELECT
    row_number() over () as rnum,
    email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

<table>
<thead>
<tr>
<th>rnum</th>
<th>email</th>
<th>first_name</th>
<th>last_name</th>
<th>account_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="mailto:admin@boss.org">admin@boss.org</a></td>
<td>Admin</td>
<td>Boss</td>
<td>admin</td>
</tr>
<tr>
<td>2</td>
<td><a href="mailto:bob.carlsen@foo.bar">bob.carlsen@foo.bar</a></td>
<td>Bob</td>
<td>Carlsen</td>
<td>regular</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:eddie.stevens@data.org">eddie.stevens@data.org</a></td>
<td>Eddie</td>
<td>Stevens</td>
<td>regular</td>
</tr>
<tr>
<td>4</td>
<td><a href="mailto:john.smith@xyz.org">john.smith@xyz.org</a></td>
<td>John</td>
<td>Smith</td>
<td>regular</td>
</tr>
<tr>
<td>5</td>
<td><a href="mailto:root@boss.org">root@boss.org</a></td>
<td>Root</td>
<td>Chief</td>
<td>admin</td>
</tr>
</tbody>
</table>
What are window functions?

Let’s start with a “function like” example

```
SELECT
    row_number() over () as rnum,
    email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

| rnum | email                  | first_name | last_name | account_type |
|------+------------------------+------------+-----------+--------------|
| 1    | admin@boss.org         | Admin      | Boss      | admin        |
| 2    | bob.carlsen@foo.bar    | Bob        | Carlsen   | regular      |
| 3    | eddie.stevens@data.org | Eddie      | Stevens   | regular      |
| 4    | john.smith@xyz.org     | John       | Smith     | regular      |
| 5    | root@boss.org          | Root       | Chief     | admin        |

This order is not deterministic!
What are window functions?

Let’s start with a “function like” example

```sql
SELECT row_number() over () as rnum,
       email, first_name,
       last_name, account_type
FROM users
ORDER BY email;
```

This is also valid!

<table>
<thead>
<tr>
<th>rnum</th>
<th>email</th>
<th>first_name</th>
<th>last_name</th>
<th>account_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><a href="mailto:admin@boss.org">admin@boss.org</a></td>
<td>Admin</td>
<td>Boss</td>
<td>admin</td>
</tr>
<tr>
<td>1</td>
<td><a href="mailto:bob.carlsen@foo.bar">bob.carlsen@foo.bar</a></td>
<td>Bob</td>
<td>Carlsen</td>
<td>regular</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:eddie.stevens@data.org">eddie.stevens@data.org</a></td>
<td>Eddie</td>
<td>Stevens</td>
<td>regular</td>
</tr>
<tr>
<td>5</td>
<td><a href="mailto:john.smith@xyz.org">john.smith@xyz.org</a></td>
<td>John</td>
<td>Smith</td>
<td>regular</td>
</tr>
<tr>
<td>4</td>
<td><a href="mailto:root@boss.org">root@boss.org</a></td>
<td>Root</td>
<td>Chief</td>
<td>admin</td>
</tr>
</tbody>
</table>
What are window functions?

Let's start with a “function like” example

```
SELECT
    row_number() over () as rnum,
    email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

| rnum | email                  | first_name | last_name | account_type |
|------|------------------------+------------+-----------+--------------|
| 5    | admin@boss.org         | Admin      | Boss      | admin        |
| 4    | bob.carlsen@foo.bar    | Bob        | Carlsen   | regular      |
| 3    | eddie.stevens@data.org | Eddie      | Stevens   | regular      |
| 2    | john.smith@xyz.org     | John       | Smith     | regular      |
| 1    | root@boss.org          | Root       | Chief     | admin        |

And this one...
### What are window functions?

Let's start with a “function like” example:

```sql
SELECT row_number() over (ORDER BY email) as rnum, email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

<table>
<thead>
<tr>
<th>rnum</th>
<th>email</th>
<th>first_name</th>
<th>last_name</th>
<th>account_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="mailto:admin@boss.org">admin@boss.org</a></td>
<td>Admin</td>
<td>Boss</td>
<td>admin</td>
</tr>
<tr>
<td>2</td>
<td><a href="mailto:bob.carlsen@foo.bar">bob.carlsen@foo.bar</a></td>
<td>Bob</td>
<td>Carlsen</td>
<td>regular</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:eddie.stevens@data.org">eddie.stevens@data.org</a></td>
<td>Eddie</td>
<td>Stevens</td>
<td>regular</td>
</tr>
<tr>
<td>4</td>
<td><a href="mailto:john.smith@xyz.org">john.smith@xyz.org</a></td>
<td>John</td>
<td>Smith</td>
<td>regular</td>
</tr>
<tr>
<td>5</td>
<td><a href="mailto:root@boss.org">root@boss.org</a></td>
<td>Root</td>
<td>Chief</td>
<td>admin</td>
</tr>
</tbody>
</table>

Now only this one is valid!
Let’s start with a “function like” example

```sql
SELECT row_number() over (ORDER BY email) as rnum, email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

<table>
<thead>
<tr>
<th>rnum</th>
<th>email</th>
<th>first_name</th>
<th>last_name</th>
<th>account_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="mailto:admin@boss.org">admin@boss.org</a></td>
<td>Admin</td>
<td>Boss</td>
<td>admin</td>
</tr>
<tr>
<td>2</td>
<td><a href="mailto:bob.carlsen@foo.bar">bob.carlsen@foo.bar</a></td>
<td>Bob</td>
<td>Carlsen</td>
<td>regular</td>
</tr>
<tr>
<td>3</td>
<td><a href="mailto:eddie.stevens@data.org">eddie.stevens@data.org</a></td>
<td>Eddie</td>
<td>Stevens</td>
<td>regular</td>
</tr>
<tr>
<td>4</td>
<td><a href="mailto:john.smith@xyz.org">john.smith@xyz.org</a></td>
<td>John</td>
<td>Smith</td>
<td>regular</td>
</tr>
<tr>
<td>5</td>
<td><a href="mailto:root@boss.org">root@boss.org</a></td>
<td>Root</td>
<td>Chief</td>
<td>admin</td>
</tr>
</tbody>
</table>

How do we “group” by account type?
What are window functions?

Let’s start with a “function like” example

```sql
SELECT
    row_number() over (PARTITION BY account_type ORDER BY email) as rnum,
    email, first_name, last_name, account_type
FROM users
ORDER BY account_type, email;
```

| rnum | email                  | first_name | last_name | account_type |
|------|------------------------+------------+-----------+--------------|
| 1    | admin@boss.org         | Admin      | Boss      | admin        |
| 2    | root@boss.org          | Root       | Chief     | admin        |
| 1    | bob.carlsen@foo.bar    | Bob        | Carlsen   | regular      |
| 2    | eddie.stevens@data.org | Eddie      | Stevens   | regular      |
| 3    | john.smith@xyz.org     | John       | Smith     | regular      |
```

row_number() resets for every partition
What are window functions?

How about that aggregate similarity?

```
SELECT
    time, value
FROM data_points
ORDER BY time;
```

Sensor Data

- Value
- Time

© 2017 MariaDB Foundation
What are window functions?

How about that aggregate similarity?

SELECT
  time, value
FROM data_points
ORDER BY time;

SELECT
  time, value
  avg(value) over (ORDER BY time
),
FROM data_points
ORDER BY time;
What are window functions?

How about that aggregate similarity?

```
SELECT
  time, value
FROM data_points
ORDER BY time;
```

```
SELECT
  time, value
  avg(value) over (ORDER BY time
  ROWS BETWEEN 3 PRECEDING
  AND 3 FOLLOWING),
FROM data_points
ORDER BY time;
```
What are window functions?

How about that aggregate similarity?

```sql
SELECT
    time, value
FROM data_points
ORDER BY time;
```

```sql
SELECT
    time, value
    avg(value) over (ORDER BY time
                    ROWS BETWEEN 6 PRECEDING
                    AND 6 FOLLOWING),
FROM data_points
ORDER BY time;
```

Sensor Data

- Raw Data
- 2x Smoothed Average
Window Functions in MariaDB

- We support in 10.2:
  - ROW_NUMBER, RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST, NTILE
  - FIRST_VALUE, LAST_VALUE, NTH_VALUE, LEAD, LAG
  - All regular aggregate functions except GROUP_CONCAT
Window Functions in MariaDB

- In 10.3 we (will) support:
  - Advanced window functions such as:
    - PERCENTILE_CONT, PERCENTILE_DISC, MEDIAN
  - Feature parity with ColumnStore engine.
  - Performance optimizations for MIN/MAX when result sets are already ordered. (To be pushed before 10.3 is Beta)