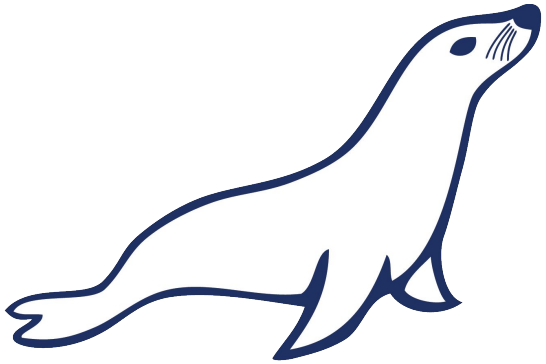


MariaDB®
FOUNDATION

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

```
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 ...
```

Keyword

What are CTEs?

Syntax

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

CTE Name

What are CTEs?

Syntax

```
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 ...
```

Linear View

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

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

Step 1: Get table header and types from Anchor

```
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 |
|   20 | Dad          |     10 |   NULL |
|   30 | Mom          |   NULL |   NULL |
|   10 | Grandpa Bill |   NULL |   NULL |
|   98 | Sister Amy   |     20 |     30 |
+------+--------------+--------+--------+
```

Recursive CTEs

Step 2: Get values for anchor

```
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
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL
98	Sister Amy	20	30

Recursive CTEs

Step 2: Get values for anchor

```
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 |
+------+--------------+--------+--------+
```

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 |
+------+--------------+--------+--------+
```

```
+------+--------------+--------+--------+
| 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;
```

```
+------+--------------+--------+--------+
| 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 |
+------+--------------+--------+--------+
```

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 |
+------+--------------+--------+--------+
```

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 |
+------+--------------+--------+--------+
```

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 |
+------+--------------+--------+--------+
```

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

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 Bill |   NULL |   NULL |
+------+--------------+--------+--------+
```

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 Bill |   NULL |   NULL |
+------+--------------+--------+--------+
```

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 |
+------+--------------+--------+--------+
```

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

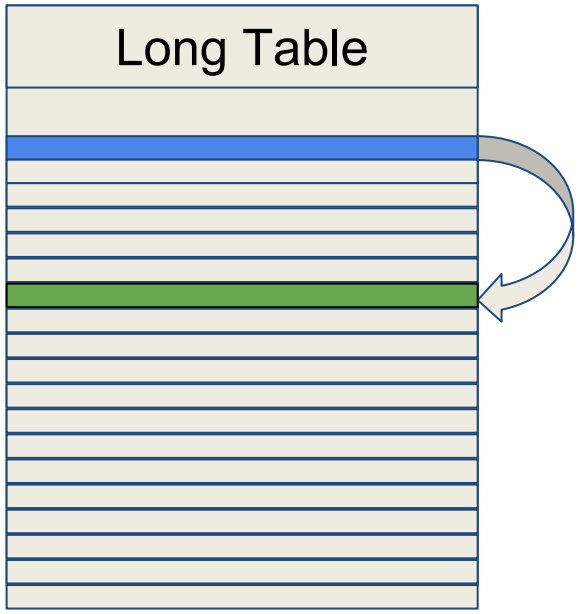
No new rows!
Done!

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?

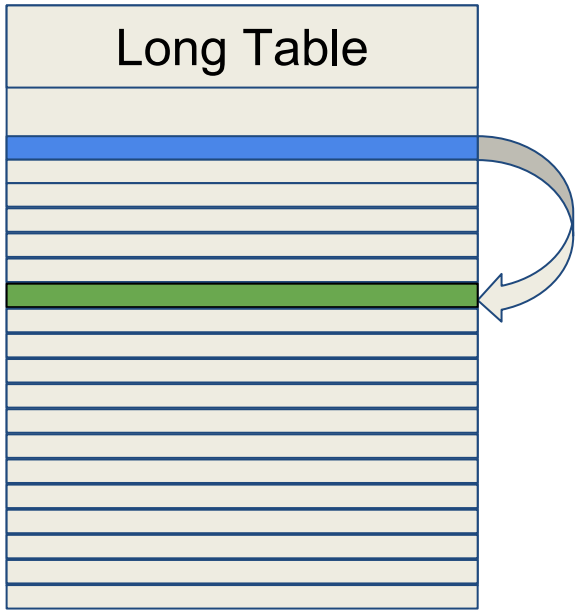
- Can access multiple rows from the current row.



What can window functions do?

- Can access multiple rows from the current row.

- Eliminate self-joins.

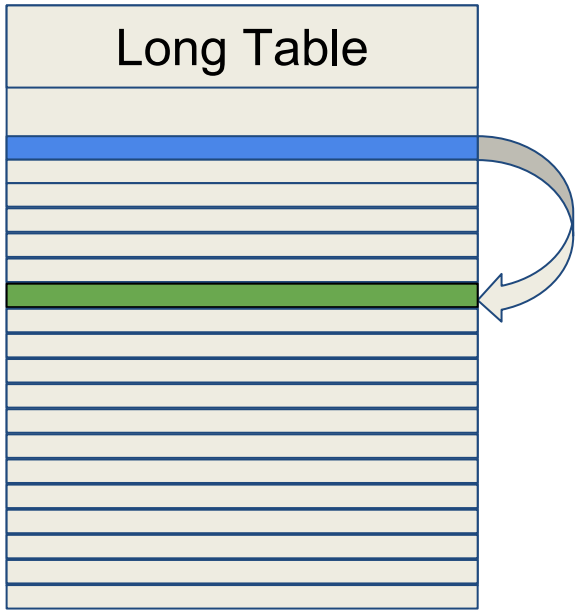


What can window functions do?

- 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;
```

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

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        |
+------+-------------------------+------------+-----------+--------------+
```

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;
```

This order is not deterministic!

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

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;
```

This is also valid!

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

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;
```

And this one...

```
+------+-----------------------+------------+-----------+--------------+
| 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        |
+------+-----------------------+------------+-----------+--------------+
```

What are window functions?

Let's start with a "function like" example

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

Now only this one is valid!

```
+------+-----------------------+------------+-----------+--------------+
| 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        |
+------+-----------------------+------------+-----------+--------------+
```

What are window functions?

Let's start with a "function like" example

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

How do we "group" by account type?

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

What are window functions?

Let's start with a "function like" example

```
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;
```

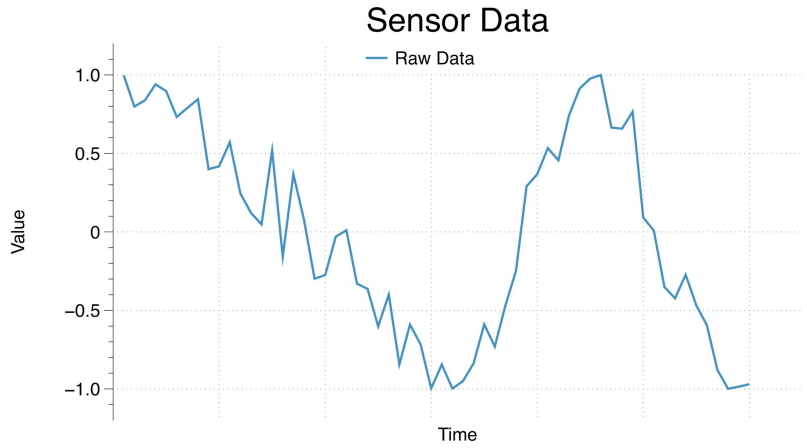
row_number() resets for every partition

```
+------+-----------------------+------------+-----------+--------------+
| 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      |
+------+-----------------------+------------+-----------+--------------+
```

What are window functions?

How about that aggregate similarity?

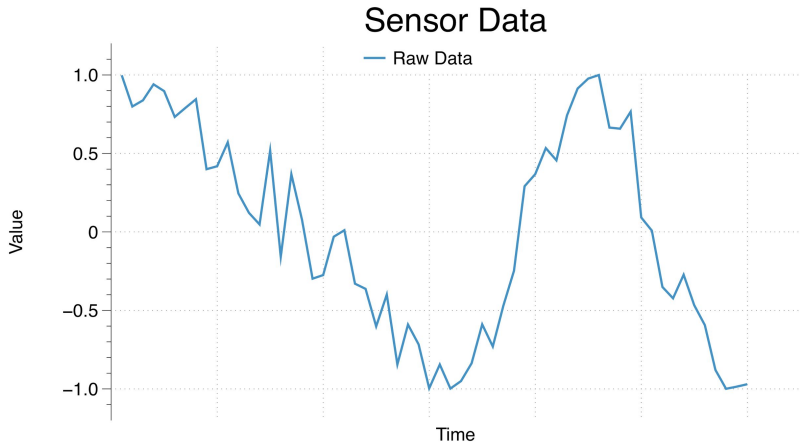
```
SELECT
      time, value
FROM data_points
ORDER BY time;
```



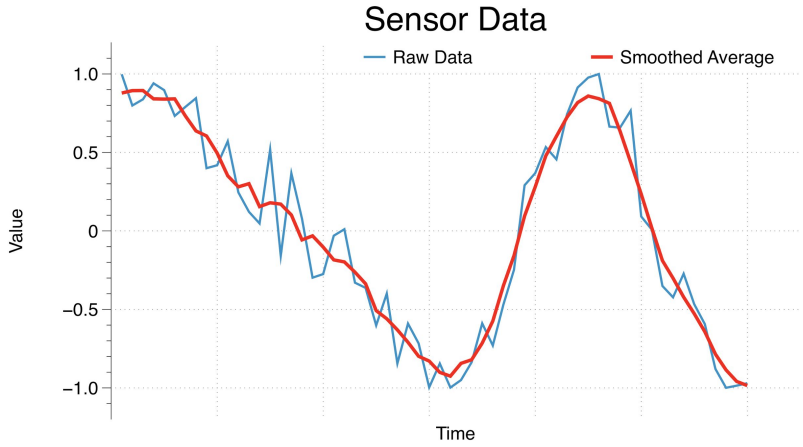
What are window functions?

How about that aggregate similarity?

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



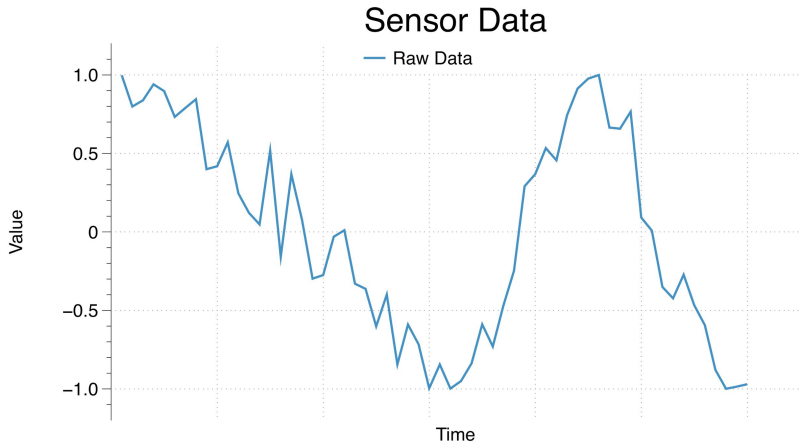

```
SELECT
      time, value
      avg(value) over (ORDER BY time
                                      ),
FROM data_points
ORDER BY time;
```



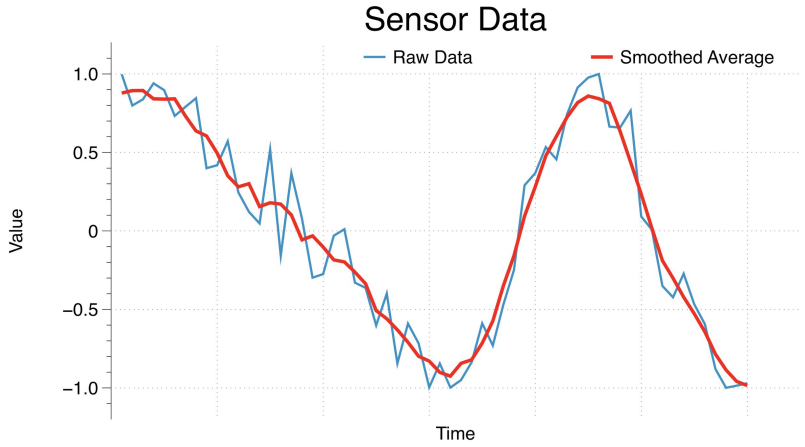
What are window functions?

How about that aggregate similarity?

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



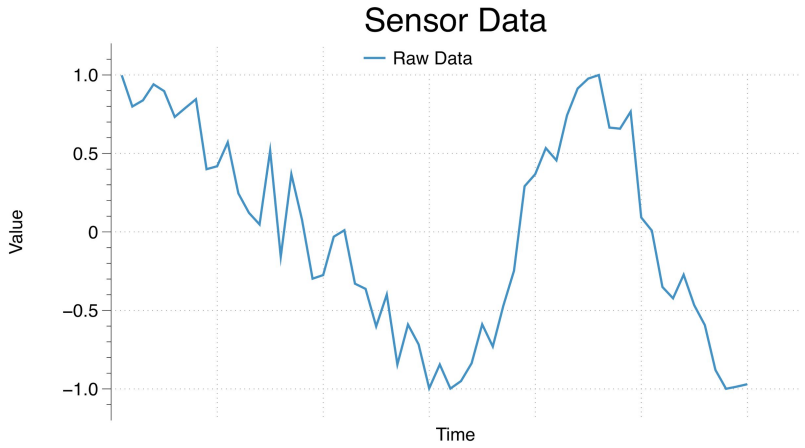

```
SELECT
      time, value
      avg(value) over (ORDER BY time
                       ROWS BETWEEN 3 PRECEDING
                                AND 3 FOLLOWING),
FROM data_points
ORDER BY time;
```



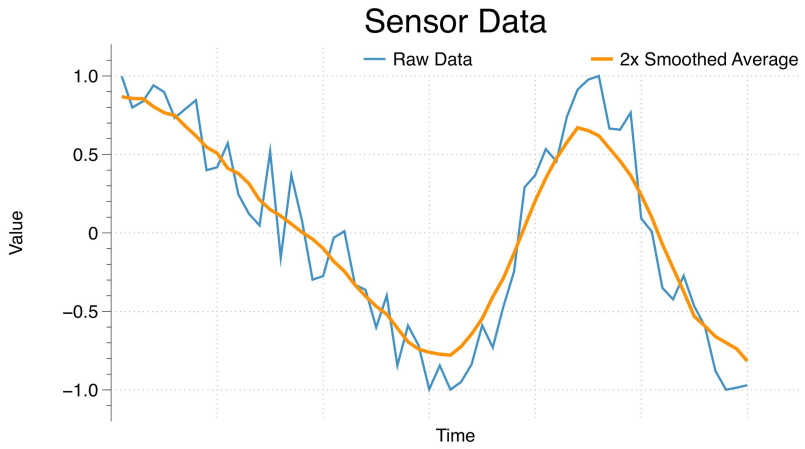
What are window functions?

How about that aggregate similarity?

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




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



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)