Optimizing Queries Using CTEs and Window Functions

Vicențiu Ciorbaru
Software Engineer @ MariaDB Foundation
Agenda

- What are Common Table Expressions (CTEs)?
- What are Window Functions?
- Practical use cases
- Why are window functions fast?
- Development status in MariaDB
What are CTEs?

Syntax

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 ...
```
What are CTEs?

Syntax

```sql
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 ("NL",...)  
)  
SELECT *  
FROM eu_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 ("NL",...)
)
SELECT *  
FROM eu_engineers  
WHERE ...

SELECT *  
FROM (SELECT *  
  FROM (SELECT *  
    FROM employees  
    WHERE dept="Engineering")) AS engineers  
  WHERE country IN ("NL",...)
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
Summary on CTEs

- Identified by the WITH clause.
- Similar to derived tables in the FROM clause.
- More expressive and provide cleaner code.
- Can produce more efficient query plans.
### CTE execution

#### Basic algorithm

**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

- Materialize each CTE occurrence into a Temporary Table
- Often Not optimal!
CTE optimization #1

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

- Materialize each CTE occurrence into a Temporary Table

We can reuse CTE here!
CTE optimization #1

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

- Materialize each distinct CTE occurrence into a Temporary Table

Materialize only once!
CTE optimization #1

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

● Materialize each distinct CTE occurrence into a Temporary Table
● Not compatible with other optimizations.

Materialize only once!
WITH engineers AS (  
  SELECT * FROM EMPLOYEES  
  WHERE  
    dept='Development'  
)  
SELECT  
  ...  
FROM  
  engineers E,  
  support_cases SC  
WHERE  
  E.name=SC.assignee and  
  SC.created='2017-04-10' and  
  E.location='New York'

Requirements:  
- CTE is used in a JOIN, no GROUP BY, DISTINCT, etc.
WITH engineers AS (  
    SELECT * FROM EMPLOYEES  
    WHERE  
        dept='Development'  
  )  
SELECT ...  
FROM  
    engineers E,  
    support_cases SC  
WHERE  
    E.name=SC.assignee and  
    SC.created='2017-04-10' and  
    E.location='New York'  

SELECT ...  
FROM  
    employees E,  
    support_cases SC  
WHERE  
    E.name=SC.assignee and  
    SC.created='2017-04-10' and  
    E.location='New York'  
    E.dept='Development'

Requirements:  
- CTE is used in a JOIN, no GROUP BY, DISTINCT, etc.
WITH engineers AS (  
    SELECT * FROM EMPLOYEES  
    WHERE  
        dept='Development'  
  )  
SELECT  
    ...  
FROM  
    engineers E,  
    support_cases SC  
WHERE  
    E.name=SC.assignee and  
    SC.created='2017-04-10' and  
    E.location='New York'  

CTE merged into parent join.  
Now optimizer can pick any query plan.  
Same algorithm is used for VIEWS (ALGORITHM = MERGE)  

Requirements:  
● CTE is used in a JOIN, no GROUP BY, DISTINCT, etc.
WITH sales_per_year AS (  
    SELECT  
        year(order.date) AS year  
        sum(order.amount) AS sales  
    FROM  
        order  
    GROUP BY  
        year  
    )  
SELECT *  
FROM sales_per_year  
WHERE  
    year in ('2015','2016')

Condition pushdown
WITH sales_per_year AS (  
    SELECT  
        year(order.date) AS year  
        , sum(order.amount) AS sales  
    FROM  
        order  
    GROUP BY  
        year  
)
SELECT *
FROM sales_per_year
WHERE  
    year in ('2015','2016')

Requirements:
- Merging is not possible (GROUP BY exists)
- Conditions in outer select
CTE optimization #3

Requirements:
- Merging is not possible (GROUP BY exists)
- Conditions in outer select
CTE optimization #3

Condition pushdown

- Makes temporary tables smaller.
- Can filter out whole groups.
- Works for derived tables and views.
- Implemented as a GSoC project:

  “Pushing conditions into non-mergeable views and derived tables in MariaDB”

WITH sales_per_year AS (  
    SELECT  
      year(order.date) as year  
      sum(order.amount) as sales  
    FROM  
      order  
    WHERE  
      year in ('2015','2016')  
      GROUP BY  
      year  
  )  
SELECT *  
FROM sales_per_year
## CTE Optimizations Summary

<table>
<thead>
<tr>
<th></th>
<th>CTE Merge</th>
<th>Condition pushdown</th>
<th>CTE reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>MariaDB 10.2</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>MS SQL Server</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>MySQL 8.0.0-labs-optimizer</td>
<td>✔</td>
<td>✘</td>
<td>✔*</td>
</tr>
</tbody>
</table>

- Merge and condition pushdown are most important
  - Can not be used at the same time as CTE reuse
- PostgreSQL considers CTEs optimization barriers
- MySQL (8.0) tries merging, otherwise reuse
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.
- Convert queries that take 10 hours to only take 1 minute.
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?

Let’s start with a “function like” example

```
SELECT
    email, first_name, last_name, account_type
FROM users
ORDER BY email;
```

<table>
<thead>
<tr>
<th>email</th>
<th>first_name</th>
<th>last_name</th>
<th>account_type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:admin@boss.org">admin@boss.org</a></td>
<td>Admin</td>
<td>Boss</td>
<td>admin</td>
</tr>
<tr>
<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><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><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><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

```sql
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;
```

<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>

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;
```

<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>

This is also valid!
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>5</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>4</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>2</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>1</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

```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!
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;
```

<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;
```

<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:root@boss.org">root@boss.org</a></td>
<td>Root</td>
<td>Chief</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>2</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>3</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>
</tbody>
</table>
```

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

SELECT
time, value
FROM data_points
ORDER BY time;

How about that aggregate similarity?
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?

SELECT time, value
FROM data_points
ORDER BY time;

How about that aggregate similarity?

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

Value

Time
What are window functions?

So how do frames work?

```sql
SELECT
time, value
sum(value) OVER ( 
ORDER BY time
ROWS BETWEEN 1 PRECEDING
AND 1 FOLLOWING)
FROM data_points
ORDER BY time;
```

```
+----------+-------+------+
<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
+----------+-------+------+
```

```sql
SELECT
time, value
sum(value) OVER ( 
ORDER BY time
ROWS BETWEEN 2 PRECEDING
AND 2 FOLLOWING)
FROM data_points
ORDER BY time;
```

```
+----------+-------+------+
<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
+----------+-------+------+
```
What are window functions?

So how do frames work?

SELECT
time, value
sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 1 PRECEDING
    AND 1 FOLLOWING)
FROM data_points
ORDER BY time;

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

(2 + 5)

SELECT
time, value
sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 2 PRECEDING
    AND 2 FOLLOWING)
FROM data_points
ORDER BY time;

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

(2 + 5 + 4)
What are window functions?

SELECT
  time, value
  sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 1 PRECEDING
    AND 1 FOLLOWING)
FROM data_points
ORDER BY time;

+----------+-------+------+
| time     | value | sum  |
+----------+-------+------+
| 10:00:00 | 2     | 7    |
| 11:00:00 | 5     | 11   |
| 12:00:00 | 4     |      |
| 13:00:00 | 4     |      |
| 14:00:00 | 1     |      |
| 15:00:00 | 5     |      |
| 15:00:00 | 2     |      |
+----------+-------+------+

So how do frames work?

SELECT
  time, value
  sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 2 PRECEDING
    AND 2 FOLLOWING)
FROM data_points
ORDER BY time;

+----------+-------+------+
| time     | value | sum  |
+----------+-------+------+
| 10:00:00 | 2     | 11   |
| 11:00:00 | 5     | 15   |
| 12:00:00 | 4     |      |
| 13:00:00 | 4     |      |
| 14:00:00 | 1     |      |
| 15:00:00 | 5     |      |
| 15:00:00 | 2     |      |
| 15:00:00 | 2     |      |
+----------+-------+------+
What are window functions?

So how do frames work?

SELECT
  time, value
  sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 1 PRECEDING
    AND 1 FOLLOWING)
FROM data_points
ORDER BY time;

SELECT
  time, value
  sum(value) OVER (
    ORDER BY time
    ROWS BETWEEN 2 PRECEDING
    AND 2 FOLLOWING)
FROM data_points
ORDER BY time;
What are window functions?

SELECT time, value, sum(value) OVER (ORDER BY time ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) FROM data_points ORDER BY time;

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

SELECT time, value, sum(value) OVER (ORDER BY time ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) FROM data_points ORDER BY time;

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

So how do frames work?
What are window functions?

So how do frames work?

Every new row adds a value and removes a value!
What are window functions?

So how do frames work?

We can do “on-line” computation!

<table>
<thead>
<tr>
<th>time</th>
<th>value</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:00:00</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>11:00:00</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>12:00:00</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>13:00:00</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>14:00:00</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>15:00:00</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

SELECT
  time, value
  sum(value) OVER (ORDER BY time
  ROWS BETWEEN 1 PRECEDING
  AND 1 FOLLOWING)
FROM data_points
ORDER BY time;

SELECT
  time, value
  sum(value) OVER (ORDER BY time
  ROWS BETWEEN 2 PRECEDING
  AND 2 FOLLOWING)
FROM data_points
ORDER BY time;

© 2017 MariaDB Foundation
What are window functions?

So how do frames work?

SELECT
    time, value
    sum(value) OVER (
        ORDER BY time
        ROWS BETWEEN 1 PRECEDING
        AND 1 FOLLOWING)
FROM data_points
ORDER BY time;

+----------+-------+------+
| time     | value | sum  |
+----------+-------+------+
| 10:00:00 |     2 |    7 | (2 + 5)
| 11:00:00 |     5 |   11 | (2 + 5 + 4)
| 12:00:00 |     4 |   13 | (5 + 4 + 4)
| 13:00:00 |     4 |    9 | (4 + 4 + 1)
| 14:00:00 |     1 |   10 | (4 + 1 + 5)
| 15:00:00 |     5 |    8 | (1 + 5 + 2)
| 15:00:00 |     2 |    9 | (5 + 2 + 2)
| 15:00:00 |     2 |    4 | (2 + 2)
+----------+-------+------+

SELECT
    time, value
    sum(value) OVER (
        ORDER BY time
        ROWS BETWEEN 2 PRECEDING
        AND 2 FOLLOWING)
FROM data_points
ORDER BY time;

+----------+-------+------+
| time     | value | sum  |
+----------+-------+------+
| 10:00:00 |     2 |   11 | (2 + 5 + 4)
| 11:00:00 |     5 |   15 | (2 + 5 + 4 + 4)
| 12:00:00 |     4 |   16 | (2 + 5 + 4 + 4 + 1)
| 13:00:00 |     4 |   19 | (5 + 4 + 4 + 1 + 5)
| 14:00:00 |     1 |   16 | (4 + 4 + 1 + 5 + 2)
| 15:00:00 |     5 |   14 | (4 + 1 + 5 + 2 + 2)
| 15:00:00 |     2 |   10 | (1 + 5 + 2 + 2)
| 15:00:00 |     2 |    9 | (5 + 2 + 2)
+----------+-------+------+
Scenario 1 - Regular SQL

Given a set of bank transactions, compute the account balance after each transaction.

```sql
SELECT timestamp, transaction_id, customer_id, amount,
FROM transactions
ORDER BY customer_id, timestamp;
```

<table>
<thead>
<tr>
<th>timestamp</th>
<th>transaction_id</th>
<th>customer_id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-09-01 10:00:00</td>
<td>1</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>2016-09-01 11:00:00</td>
<td>2</td>
<td>1</td>
<td>-200</td>
</tr>
<tr>
<td>2016-09-01 12:00:00</td>
<td>3</td>
<td>1</td>
<td>-600</td>
</tr>
<tr>
<td>2016-09-01 13:00:00</td>
<td>5</td>
<td>1</td>
<td>400</td>
</tr>
<tr>
<td>2016-09-01 12:10:00</td>
<td>4</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>2016-09-01 14:00:00</td>
<td>6</td>
<td>2</td>
<td>500</td>
</tr>
<tr>
<td>2016-09-01 15:00:00</td>
<td>7</td>
<td>2</td>
<td>400</td>
</tr>
</tbody>
</table>
Scenario 1 - Regular SQL

Given a set of bank transactions, compute the account balance after each transaction.

SELECT timestamp, transaction_id, customer_id, amount, 
  (SELECT sum(amount) 
   FROM transactions AS t2 
   WHERE t2.customer_id = t1.customer_id AND 
     t2.timestamp <= t1.timestamp) AS balance 
FROM transactions AS t1 
ORDER BY customer_id, timestamp;

<table>
<thead>
<tr>
<th>timestamp</th>
<th>transaction_id</th>
<th>customer_id</th>
<th>amount</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-09-01 10:00:00</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2016-09-01 11:00:00</td>
<td>2</td>
<td>1</td>
<td>-200</td>
<td>800</td>
</tr>
<tr>
<td>2016-09-01 12:00:00</td>
<td>3</td>
<td>1</td>
<td>-600</td>
<td>200</td>
</tr>
<tr>
<td>2016-09-01 13:00:00</td>
<td>5</td>
<td>1</td>
<td>400</td>
<td>600</td>
</tr>
<tr>
<td>2016-09-01 12:10:00</td>
<td>4</td>
<td>2</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>2016-09-01 14:00:00</td>
<td>6</td>
<td>2</td>
<td>500</td>
<td>800</td>
</tr>
<tr>
<td>2016-09-01 15:00:00</td>
<td>7</td>
<td>2</td>
<td>400</td>
<td>1200</td>
</tr>
</tbody>
</table>
Scenario 1 - Window Functions

Given a set of bank transactions, compute the account balance after each transaction.

```
SELECT timestamp, transaction_id, customer_id, amount,
    sum(amount) OVER (PARTITION BY customer_id
        ORDER BY timestamp
        ROWS BETWEEN UNBOUNDED PRECEDING AND
        CURRENT ROW) AS balance
FROM transactions AS t1
ORDER BY customer_id, timestamp;
```

+---------------------+----------------+-------------+--------+---------+
| timestamp           | transaction_id | customer_id | amount | balance |
| 2016-09-01 10:00:00 |              1 |           1 |   1000 |    1000 |
| 2016-09-01 11:00:00 |              2 |           1 |   -200 |     800 |
| 2016-09-01 12:00:00 |              3 |           1 |   -600 |     200 |
| 2016-09-01 13:00:00 |              5 |           1 |    400 |     600 |
| 2016-09-01 12:10:00 |              4 |           2 |    300 |     300 |
| 2016-09-01 14:00:00 |              6 |           2 |    500 |     800 |
| 2016-09-01 15:00:00 |              7 |           2 |    400 |    1200 |
Scenario 1 - Performance

Given a set of bank transactions, compute the account balance after each transaction.

<table>
<thead>
<tr>
<th>#Rows</th>
<th>Regular SQL (seconds)</th>
<th>Regular SQL + Index (seconds)</th>
<th>Window Functions (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 000</td>
<td>0.29</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>100 000</td>
<td>2.91</td>
<td>0.09</td>
<td>0.16</td>
</tr>
<tr>
<td>1 000 000</td>
<td>29.1</td>
<td>2.86</td>
<td>3.04</td>
</tr>
<tr>
<td>10 000 000</td>
<td>346.3</td>
<td>90.97</td>
<td>43.17</td>
</tr>
<tr>
<td>100 000 000</td>
<td>4357.2</td>
<td>813.2</td>
<td>514.24</td>
</tr>
</tbody>
</table>
Practical Use Cases - Scenario 2

- “Top-N” queries
- Retrieve the top 5 earners by department.
Scenario 2 - Regular SQL

Retrieve the top 5 earners by department.

```sql
SELECT dept, name, salary
FROM employee_salaries
ORDER BY dept;
```

```
+-------+----------+--------+
<table>
<thead>
<tr>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td>Sales</td>
<td>Bill</td>
<td>150</td>
</tr>
<tr>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
<tr>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td>Eng</td>
<td>Tim</td>
<td>1000</td>
</tr>
<tr>
<td>Eng</td>
<td>Michael</td>
<td>2000</td>
</tr>
<tr>
<td>Eng</td>
<td>Andrew</td>
<td>1500</td>
</tr>
<tr>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
</tbody>
</table>
```
Scenario 2 - Regular SQL

Retrieve the top 5 earners by department.

```
SELECT dept, name, salary
FROM employee_salaries AS t1
WHERE (SELECT count(*)
      FROM employee_salaries AS t2
      WHERE t1.name != t2.name AND
        t1.dept = t2.dept AND
        t2.salary > t1.salary)
  < 5
ORDER BY dept, salary DESC;
```

```
+-------+----------+--------+
| dept  | name     | salary |
+-------+----------+--------+
| Eng   | Kristian | 3500   |
| Eng   | Sergei   | 3000   |
| Eng   | Sami     | 2800   |
| Eng   | Arnold   | 2500   |
| Eng   | Scarlett | 2200   |
| Sales | Bob      | 500    |
| Sales | Jill     | 400    |
| Sales | Lucy     | 300    |
| Sales | Tom      | 300    |
| Sales | Axel     | 250    |
```
Scenario 2 - Regular SQL

Retrieve the top 5 earners by department.

```sql
SELECT dept, name, salary
FROM employee_salaries AS t1
WHERE (SELECT count(*)
    FROM employee_salaries AS t2
    WHERE t1.name != t2.name AND
t1.dept = t2.dept AND
t2.salary > t1.salary) < 5
ORDER BY dept, salary DESC;
```

<table>
<thead>
<tr>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
<tr>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
</tbody>
</table>

What if I want a “rank” column?
Scenario 2 - Regular SQL

Retrieve the top 5 earners by department.

```sql
SELECT (SELECT count(*) + 1
FROM employee_salaries as t2
WHERE t1.name != t2.name and
    t1.dept = t2.dept and
    t2.salary > t1.salary)
AS ranking,
    dept, name, salary
FROM employee_salaries AS t1
WHERE (SELECT count(*)
FROM employee_salaries AS t2
WHERE t1.name != t2.name AND
    t1.dept = t2.dept AND
    t2.salary > t1.salary) < 5
ORDER BY dept, salary DESC;
```

<table>
<thead>
<tr>
<th>ranking</th>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td>2</td>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td>3</td>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
<tr>
<td>4</td>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td>5</td>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td>1</td>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
</tbody>
</table>

What if I want a “rank” column?
Scenario 2 - Window Functions

Retrieve the top 5 earners by department.

```
SELECT
    rank() OVER (PARTITION BY dept ORDER BY salary DESC) AS ranking,
    dept, name, salary
FROM employee_salaries;
```

<table>
<thead>
<tr>
<th>ranking</th>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td>2</td>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td>3</td>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
<tr>
<td>4</td>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td>5</td>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td>6</td>
<td>Eng</td>
<td>Michael</td>
<td>2000</td>
</tr>
<tr>
<td>7</td>
<td>Eng</td>
<td>Andrew</td>
<td>1500</td>
</tr>
<tr>
<td>8</td>
<td>Eng</td>
<td>Tim</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
<tr>
<td>6</td>
<td>Sales</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>Sales</td>
<td>Bill</td>
<td>150</td>
</tr>
</tbody>
</table>
```
Scenario 2 - Window Functions

Retrieve the top 5 earners by department.

```sql
SELECT
    rank() OVER (
        PARTITION BY dept
        ORDER BY salary DESC)
AS ranking,
    dept, name, salary
FROM employee_salaries
WHERE ranking <= 5;
```

<table>
<thead>
<tr>
<th>ranking</th>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Michael</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Andrew</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td>Tim</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Sales</td>
<td>Bill</td>
<td>150</td>
</tr>
</tbody>
</table>
Scenario 2 - Window Functions

Retrieve the top 5 earners by department.

```sql
SELECT
    rank() OVER (
        PARTITION BY dept
        ORDER BY salary DESC)
AS ranking,
    dept, name, salary
FROM employee_salaries
WHERE ranking <= 5;

<table>
<thead>
<tr>
<th>ranking</th>
<th>dept</th>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Eng</td>
<td>Kristian</td>
<td>3500</td>
</tr>
<tr>
<td>2</td>
<td>Eng</td>
<td>Sergei</td>
<td>3000</td>
</tr>
<tr>
<td>3</td>
<td>Eng</td>
<td>Sami</td>
<td>2800</td>
</tr>
<tr>
<td>4</td>
<td>Eng</td>
<td>Arnold</td>
<td>2500</td>
</tr>
<tr>
<td>5</td>
<td>Eng</td>
<td>Scarlett</td>
<td>2200</td>
</tr>
<tr>
<td>6</td>
<td>Eng</td>
<td>Michael</td>
<td>2000</td>
</tr>
<tr>
<td>7</td>
<td>Eng</td>
<td>Andrew</td>
<td>1500</td>
</tr>
<tr>
<td>8</td>
<td>Eng</td>
<td>Tim</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>Sales</td>
<td>Bob</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>Sales</td>
<td>Jill</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Tom</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>Sales</td>
<td>Lucy</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>Sales</td>
<td>Axel</td>
<td>250</td>
</tr>
<tr>
<td>6</td>
<td>Sales</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>Sales</td>
<td>Bill</td>
<td>150</td>
</tr>
</tbody>
</table>
```

No Window Functions in the WHERE clause :(

© 2017 MariaDB Foundation
Scenario 2 - Window Functions

Retrieve the top 5 earners by department.

WITH salary_ranks AS (  
    SELECT  
        rank() OVER (  
            PARTITION BY dept  
            ORDER BY salary DESC)  
        AS ranking,  
        dept, name, salary  
    FROM employee_salaries  
)  
SELECT *  
FROM salary_ranks  
WHERE ranking <= 5  
ORDER BY dept, ranking;
Scenario 2 - Performance

Retrieve the top 5 earners by department.

<table>
<thead>
<tr>
<th>#Rows</th>
<th>Regular SQL (seconds)</th>
<th>Regular SQL + Index (seconds)</th>
<th>Window Functions (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000</td>
<td>1.31</td>
<td>0.14</td>
<td>0.00</td>
</tr>
<tr>
<td>20,000</td>
<td>123.6</td>
<td>12.6</td>
<td>0.02</td>
</tr>
<tr>
<td>200,000</td>
<td>10,000+</td>
<td>1539.79</td>
<td>0.21</td>
</tr>
<tr>
<td>2,000,000</td>
<td>...</td>
<td>...</td>
<td>5.61</td>
</tr>
<tr>
<td>20,000,000</td>
<td>...</td>
<td>...</td>
<td>76.04</td>
</tr>
</tbody>
</table>
Practical Use Cases - Scenario 3

- We have a number of machines that need servicing.
- Servicing times are logged.
- What is the average time between services, for each machine?
Scenario 3 - Regular SQL

Compute average time between machine services.

SELECT time, machine_id
FROM maintenance_activity;

+---------------------+------------+
<table>
<thead>
<tr>
<th>time</th>
<th>machine_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-01-04 11:02:31</td>
<td>5879</td>
</tr>
<tr>
<td>2016-10-31 21:30:19</td>
<td>8580</td>
</tr>
<tr>
<td>2017-01-16 11:33:58</td>
<td>7489</td>
</tr>
<tr>
<td>2016-11-01 17:09:07</td>
<td>9590</td>
</tr>
<tr>
<td>2016-10-03 23:33:21</td>
<td>6913</td>
</tr>
<tr>
<td>2016-11-02 11:02:08</td>
<td>6892</td>
</tr>
<tr>
<td>2017-01-07 15:43:52</td>
<td>4190</td>
</tr>
<tr>
<td>2016-12-18 03:27:40</td>
<td>8578</td>
</tr>
<tr>
<td>2016-12-06 21:57:11</td>
<td>3563</td>
</tr>
<tr>
<td>2017-01-20 21:16:18</td>
<td>4434</td>
</tr>
</tbody>
</table>

....
....
....
....
Scenario 3 - Regular SQL

Compute average time between machine services.

```sql
SELECT time, machine_id FROM maintenance_activity;
```

<table>
<thead>
<tr>
<th>time</th>
<th>machine_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-01-04 11:02:31</td>
<td>5879</td>
</tr>
<tr>
<td>2016-10-31 21:30:19</td>
<td>8580</td>
</tr>
<tr>
<td>2017-01-16 11:33:58</td>
<td>7489</td>
</tr>
<tr>
<td>2016-11-01 17:09:07</td>
<td>9590</td>
</tr>
<tr>
<td>2016-10-03 23:33:21</td>
<td>6913</td>
</tr>
<tr>
<td>2016-11-02 11:02:08</td>
<td>6892</td>
</tr>
<tr>
<td>2017-01-07 15:43:52</td>
<td>4190</td>
</tr>
<tr>
<td>2016-12-18 03:27:40</td>
<td>8578</td>
</tr>
<tr>
<td>2016-12-06 21:57:11</td>
<td>3563</td>
</tr>
<tr>
<td>2017-01-20 21:16:18</td>
<td>4434</td>
</tr>
</tbody>
</table>

We want the difference between two consecutive entries. (For the same machine)
Scenario 3 - Regular SQL

Compute average time between machine services.

WITH time_diffs AS
(
    SELECT
        t1.machine_id,
        TIMEDIFF(t1.time, max(t2.time)) AS diff
    FROM maintenance_activity AS t1,
        maintenance_activity AS t2
    WHERE
        t1.machine_id = t2.machine_id and
        t2.time < t1.time
    GROUP BY t1.machine_id, t1.time
)
SELECT machine_id,
    AVG(diff) AS avg_diff
FROM time_diffs
GROUP BY machine_id
ORDER BY machine_id;
Scenario 3 - Regular SQL

Compute average time between machine services.

WITH time_diffs AS
  (SELECT machine_id, time,
       lag(time) OVER (
           PARTITION BY machine_id
           ORDER BY time)
       AS prev_time
  FROM maintenance_activity)
SELECT machine_id,
       AVG(TIME_DIFF(time, prev_time))
       AS avg_diff
FROM time_diffs
ORDER BY machine_id;
Scenario 3 - Performance

Compute average time between machine services.

<table>
<thead>
<tr>
<th>#Rows</th>
<th>Regular SQL (seconds)</th>
<th>Regular SQL + Index (seconds)</th>
<th>Window Functions (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>5.19</td>
<td>0.51</td>
<td>0.02</td>
</tr>
<tr>
<td>100000</td>
<td>526.48</td>
<td>4.67</td>
<td>0.17</td>
</tr>
<tr>
<td>1000000</td>
<td>50000+</td>
<td>47.13</td>
<td>2.92</td>
</tr>
<tr>
<td>10000000</td>
<td>...</td>
<td>505.10</td>
<td>43.62</td>
</tr>
<tr>
<td>100000000</td>
<td>...</td>
<td>5353.15</td>
<td>472.15</td>
</tr>
</tbody>
</table>
Window functions summary

- Can help eliminate expensive subqueries.
- Can help eliminate self-joins.
- Make queries more readable.
- Make (some) queries faster.
We support:

- 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

- We do not (yet) support:
  - Time interval range-type frames
  - DISTINCT clause
  - GROUP_CONCAT function
  - Advanced window functions such as:
    PERCENTILE_CONT, PERCENTILE_DISC
Thank You!

Contact me at:

vicentiu@mariadb.org
vicentiu@ciorbaru.io

Blog: vicentiu.ciorbaru.io