

MariaDB[®]
FOUNDATION

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



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?

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

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

Linear View

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



Summary on CTEs

- Identified by the WITH clause.
- Similar to derived tables in the FROM clause.
- More efficient than derived tables in some cases.
- More expressive and provide cleaner code.



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;
```

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 (),
    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 (),
  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 (),
  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 (),
  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),
  email, first_name,
  last_name, account_type
FROM users
ORDER BY email;
```

Now only this one is valid!

rnun	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),
  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),
  email, first_name,
  last_name, account_type
FROM users
ORDER BY account_type, email;
```

row_number() resets for every partition

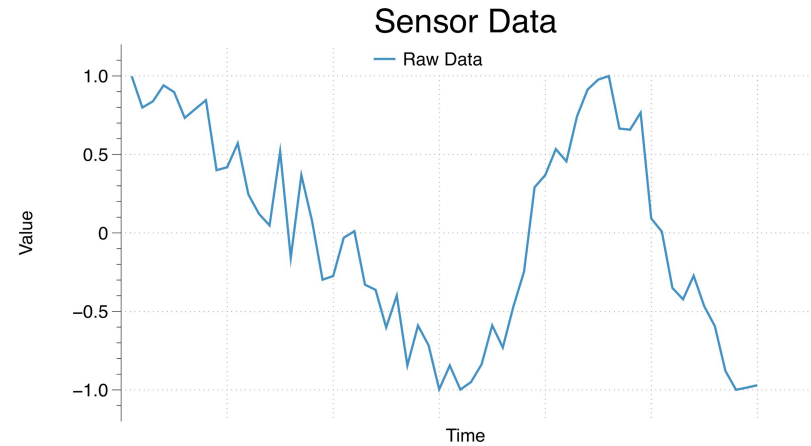
rnun	email	first_name	last_name	account_type
1	admin@boss.org	Admin	Boss	admin
2	root@boss.org	Root	Chief	admin
1	bob.carlsen@foo.bar	Bob	Carlsen	regular
2	eddie.stevens@data.org	Eddie	Stevens	regular
3	john.smith@xyz.org	John	Smith	regular



What are window functions?

How about that aggregate similarity?

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

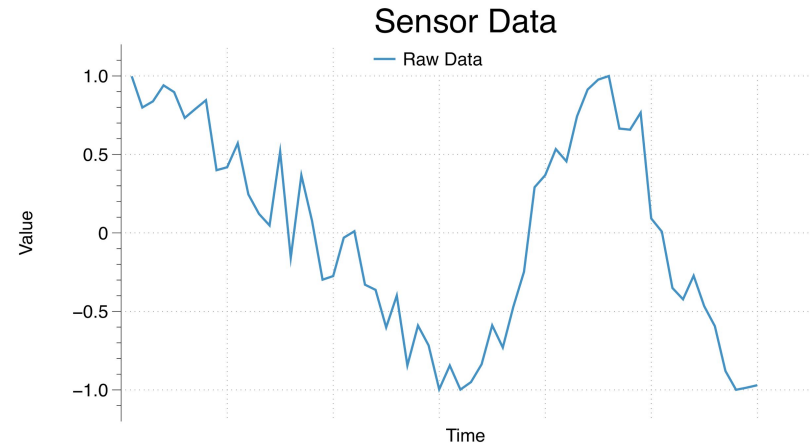




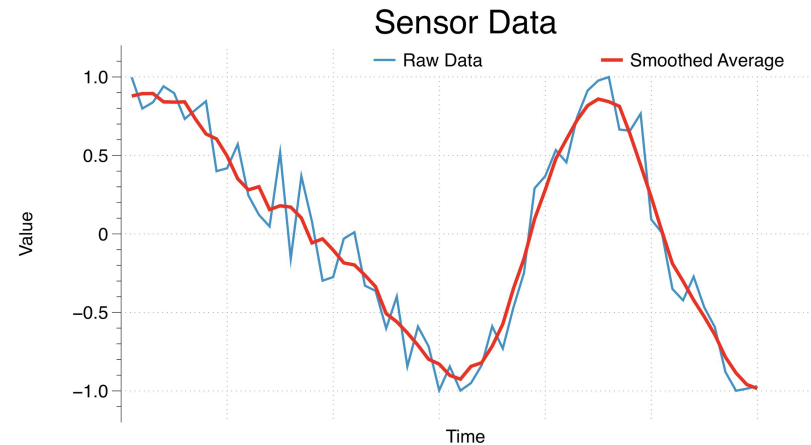
What are window functions?

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;
```

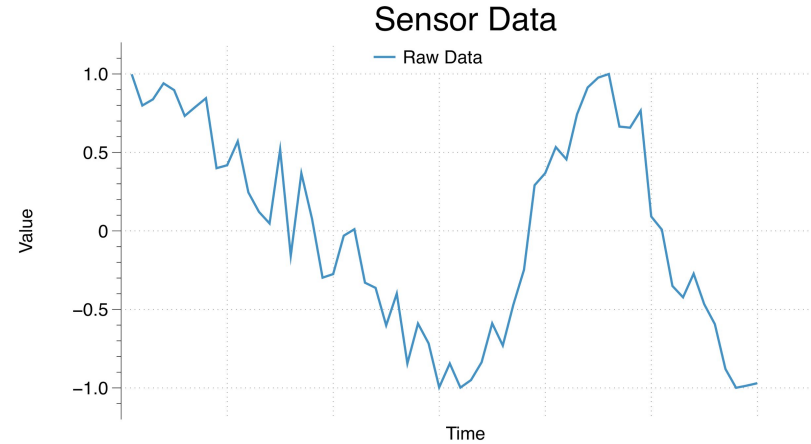




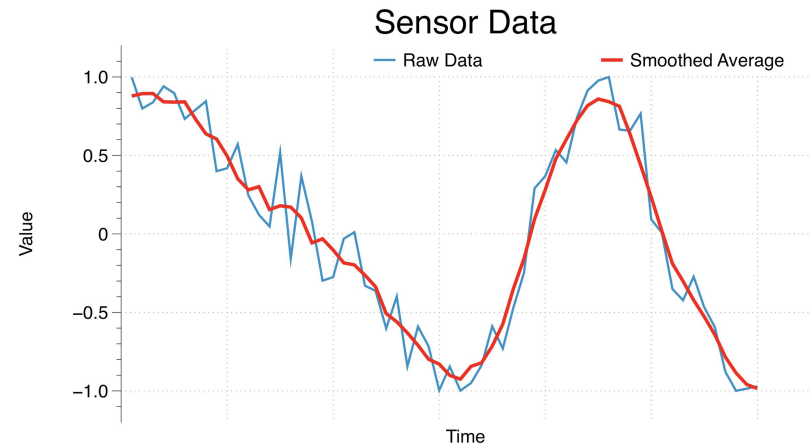
What are window functions?

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;
```

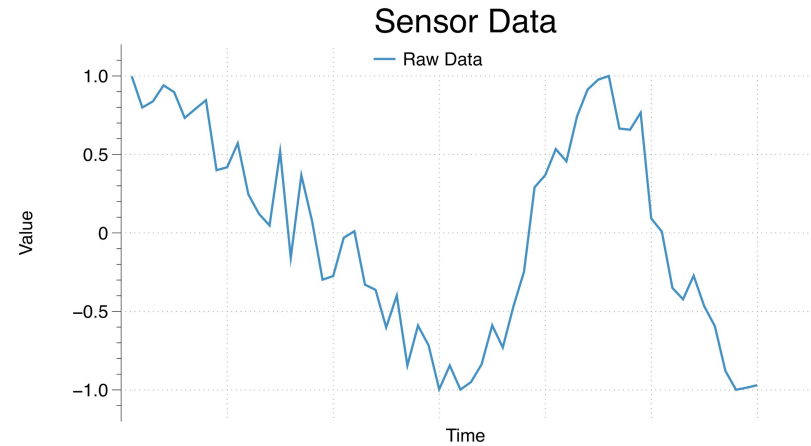




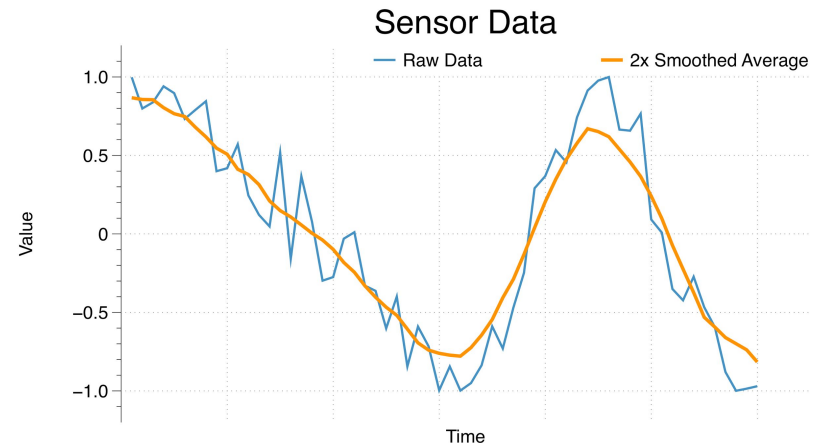
What are window functions?

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;
```





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	
11:00:00	5	
12:00:00	4	
13:00:00	4	
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	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	
11:00:00	5	
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;
```

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

(2 + 5)

```
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	
12:00:00	4	
13:00:00	4	
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5 + 4)



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
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	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)

```
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	

(2 + 5 + 4)
(2 + 5 + 4 + 4)



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
11:00:00	5	11
12:00:00	4	13
13:00:00	4	
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)

```
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	16
13:00:00	4	
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5 + 4)
(2 + 5 + 4 + 4)
(2 + 5 + 4 + 4 + 1)



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
11:00:00	5	11
12:00:00	4	13
13:00:00	4	9
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
 (2 + 5 + 4)
 (5 + 4 + 4)
 (4 + 4 + 1)

```

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	16
13:00:00	4	19
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5 + 4)
 (2 + 5 + 4 + 4)
 (2 + 5 + 4 + 4 + 1)
 (5 + 4 + 4 + 1 + 5)



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.

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

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



Scenario 1 - Regular SQL

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

```
SELECT timestamp, transaction_id, customer_id,  
       (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;
```

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 - Window Functions

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

```
SELECT timestamp, transaction_id, customer_id,  
       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



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.

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

dept	name	salary
Sales	John	200
Sales	Tom	300
Sales	Bill	150
Sales	Jill	400
Sales	Bob	500
Sales	Axel	250
Sales	Lucy	300
Eng	Tim	1000
Eng	Michael	2000
Eng	Andrew	1500
Eng	Scarlett	2200
Eng	Sergei	3000
Eng	Kristian	3500
Eng	Arnold	2500
Eng	Sami	2800



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.

```
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

What if I want a “rank” column?



Scenario 2 - Regular SQL

Retrieve the top 5 earners by department.

```
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;
```

ranking	dept	name	salary
1	Eng	Kristian	3500
2	Eng	Sergei	3000
3	Eng	Sami	2800
4	Eng	Arnold	2500
5	Eng	Scarlett	2200
1	Sales	Bob	500
2	Sales	Jill	400
3	Sales	Lucy	300
3	Sales	Tom	300
5	Sales	Axel	250

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;
```

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



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
WHERE ranking <= 5;
```

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



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
WHERE ranking <= 5;
```

No Window Functions in
the WHERE clause :(

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



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;
```

ranking	dept	name	salary
1	Eng	Kristian	3500
2	Eng	Sergei	3000
3	Eng	Sami	2800
4	Eng	Arnold	2500
5	Eng	Scarlett	2200
1	Sales	Bob	500
2	Sales	Jill	400
3	Sales	Lucy	300
3	Sales	Tom	300
5	Sales	Axel	250



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;
```

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



Scenario 3 - Regular SQL

Compute average time between machine services.

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

We want the difference between two consecutive entries.
(For the same machine)

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



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;
```

machine_id	avg_diff
0	25:38:15.0505
1	26:42:27.6969
2	24:43:18.4646
3	23:57:55.9797
4	26:30:11.6565
5	25:38:12.7070
6	27:58:24.9494
7	20:47:57.7272
8	28:16:02.0303
9	25:38:48.0505
10	28:34:57.8686
11	27:05:40.4040
12	24:09:13.5050
13	22:04:08.8383
14	26:42:41.8888
15	19:24:46.8888



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;
```

machine_id	avg_diff
0	25:38:15.0505
1	26:42:27.6969
2	24:43:18.4646
3	23:57:55.9797
4	26:30:11.6565
5	25:38:12.7070
6	27:58:24.9494
7	20:47:57.7272
8	28:16:02.0303
9	25:38:48.0505
10	28:34:57.8686
11	27:05:40.4040
12	24:09:13.5050
13	22:04:08.8383
14	26:42:41.8888
15	19:24:46.8888



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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	?	
14:00:00	1		
15:00:00	5		
15:00:00	2		
15:00:00	2		



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(5 + 4 + 4)



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(5 + 4 + 4 - 5)



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(5 + 4 + 4 - 5 + 1)



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(5 + 4 + 4 - 5 + 1)



Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(4 + 4 + 1)

Does this work all the time?



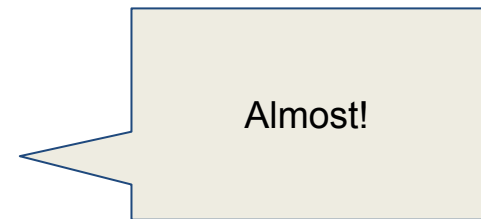
Why are window functions fast?

- It all has to do with how they are computed!
 - “On line computation”

```
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
13:00:00	4	?
14:00:00	1	
15:00:00	5	
15:00:00	2	
15:00:00	2	

(2 + 5)
(2 + 5 + 4)
(5 + 4 + 4)
(4 + 4 + 1)





Why are window functions fast?

- Difficult functions:
 - MIN and MAX (adding and removing is not trivial)
 - VAR, STDDEV functions (for now...)



Practical Use Cases - Conclusions

- Main points to remember!
 - Window functions can be used to eliminate self-joins.
 - Computation of (most) window functions is constant time per row.



Window Functions in MariaDB

- The new 10.2 Beta release added support for:
 - LEAD, LAG, FIRST_VALUE, NTH_VALUE, LAST_VALUE
 - All regular aggregate functions in MariaDB
 - Including MIN/MAX, STDDEV, VAR, etc.
 - Except GROUP_CONCAT
 - Fixed a lot of bugs. (Thank you for the use cases!)
 - Extra optimizations:
 - Computing multiple window functions during a single pass.



Window Functions in MariaDB

- Things still left to do:
 - We still have a few bugs left to solve. (Main Focus)
 - A few more functions described in the standard
 - Median would be really useful!
 - Support for RANGE type frames with DateTime Columns
 - Getting support for GROUP_CONCAT
 - Possible optimizations using condition pushdown.

Thank You!

Contact me at:

vicentiu@mariadb.org

vicentiu@ciorbaru.io

Blog: vicentiu.ciorbaru.io
