

Оконные Функции

Tags

▼ Что такое оконные функции?

Оконная функция в SQL - функция, которая работает с выделенным набором строк (окном, партицией) и выполняет вычисление для этого набора строк *в отдельном столбце*. Не меняет количество результирующих строк в таблице, в отличие от агрегаций.

Партиции оконной функции (в данном примере по полю Имя)

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3



Обрати внимание!

Оконные функции нельзя использовать нигде (where, having ect...), кроме **SELECT, ORDER BY**.

SELECT	list of columns, window functions
FROM	table / joint tables / subquery
WHERE	filtering clause
GROUP BY	list of columns
HAVING	aggregation filtering clause
ORDER BY	list of columns / window functions

Порядок расчета оконных функций в SQL запросе

▼ Синтаксис

```
OVER (  
  [ <PARTITION BY clause> ]  
  [ <ORDER BY clause> ]  
  [ <ROW or RANGE clause> ]  
)
```

Оконные функции можно прописывать как под командой SELECT, так и в отдельном ключевом слове WINDOW, где окну дается алиас (псевдоним), к которому можно обращаться в SELECT выборке.

дубли определения окна

```
select name, subject, grade,  
row_number() over (partition by name order by grade desc),  
rank() over (partition by name order by grade desc),  
dense_rank() over (partition by name order by grade desc)  
from student_grades;
```

=

```
select name, subject, grade,  
row_number() over name_grade,  
rank() over name_grade,  
dense_rank() over name_grade  
from student_grades  
window name_grade as (partition by name order by grade desc);
```

▼ Назови примеры/классы оконных функций

Множество оконных функций можно разделять на 3 класса:

▼ Агрегирующие (Aggregate)

Используются со всеми агрегатными функциями, например: **AVG()**, **SUM()**, **MIN()**, **MAX()**, **COUNT()** + менее применяемые (но тоже агр.) см. <https://www.postgresql.org/docs/current/functions-aggregate.html>

```
select name, subject, grade,  
sum(grade) over (partition by name) as sum_grade,  
avg(grade) over (partition by name) as avg_grade,  
count(grade) over (partition by name) as count_grade,  
min(grade) over (partition by name) as min_grade,
```

```
max(grade) over (partition by name) as max_grade
from student_grades;
```

name	subject	grade	sum_grade	avg_grade	count_grade	min_grade	max_grade
Маша	история	3	15	3,75	4	3	5
Маша	математика	4	15	3,75	4	3	5
Маша	русский	3	15	3,75	4	3	5
Маша	физика	5	15	3,75	4	3	5
Петя	математика	3	16	4	4	3	5
Петя	русский	4	16	4	4	3	5
Петя	физика	5	16	4	4	3	5
Петя	история	4	16	4	4	3	5



Обрати внимание!

Агрегирующие функции не обрабатывают NULL и просто его пропускают.

Например, у нас есть 13 строк, но в одной из них в столбце **SALARY** стоит **NULL**. Следующий запрос выведет всего 12 строк:

```
select *, COUNT(SALARY) OVER() from emp;
```

▼ Ранжирующие (Ranking)

Ранжирующие функции определяют порядок следования строк (например, пронумеровать их).

- **ROW_NUMBER()** - функция вычисляет последовательность ранг (порядковый номер) строк внутри партии, **НЕЗАВИСИМО** от того, есть ли в строках повторяющиеся значения или нет. Для корректной работы **необходимо использовать в совокупности с ORDER BY** (но можно без него, порядок будет случайный)
- **RANK()** - функция вычисляет ранг каждой строки внутри партии. Если есть повторяющиеся значения, функция возвращает одинаковый ранг для таких строчек, пропуская при этом следующий числовой ранг. Для корректной работы **необходимо использовать в совокупности с ORDER BY** (иначе все строки будут 1)
- **DENSE_RANK()** - то же самое что и RANK, только в случае одинаковых значений **DENSE_RANK не пропускает следующий числовой ранг, а идет последовательно.** Для корректной работы **необходимо использовать в совокупности с ORDER BY** (иначе все строки будут 1)

▼ Пример работы с кодом

```
select name, subject, grade,
row_number() over (partition by name order by grade desc),
rank() over (partition by name order by grade desc),
dense_rank() over (partition by name order by grade desc)
from student_grades;
```

name	subject	grade	row_number	rank	dense_rank
Маша	физика	5	1	1	1
Маша	математика	4	2	2	2
Маша	история	3	3	3	3
Маша	русский	3	4	3	3
Петя	физика	5	1	1	1
Петя	русский	4	2	2	2
Петя	история	4	3	2	2
Петя	математика	3	4	4	3

Пропуск значения «3»

Без пропуска значения

- **NTILE(n)** - Эта функция возвращает номер группы, в которую попадает соответствующая строка результирующего набора, распределяя набор на n групп (**кол-во групп нужно обязательно указать, ORDER BY, PARTITION BY - опциональны**)

▼ *Пример работы с кодом*

Задача. Распределить баллончики по 3-м группам поровну. Группы заполняются в порядке возрастания v_id.

http://www.sql-tutorial.ru/ru/book_ntile_function.html

```
SELECT *, NTILE(3) OVER(ORDER BY v_id) gr FROM utv ORDER BY v_id;
```

Если мы захотим распределить порознь баллончики каждого цвета, то, как и для других функций ранжирования, можно добавить конструкцию **PARTITION BY** в предложение **OVER**:

```
SELECT *, NTILE(3) OVER(PARTITION BY v_color ORDER BY v_id) gr  
FROM utv ORDER BY v_color, v_id;
```



Обрати внимание!

- В случае, когда число строк не делится нацело на число групп, функция NTILE помещает в последние группы на одну строку меньше, чем в первые.
- Если аргумент функции NTILE окажется больше числа строк, то будет сформировано количество групп, равное числу строк, и в каждой группе окажется по одной строке.

Пример, который можно вставить и посмотреть работу самостоятельно:

```
-- Create a table named "sales" with two columns: salesperson_id and sales_amount  
CREATE TABLE sales (  
    salesperson_id INT PRIMARY KEY,  
    sales_amount DECIMAL(10, 2)  
);  
  
-- Insert data into the "sales" table  
INSERT INTO sales (salesperson_id, sales_amount) VALUES  
    (1, 5000.00),  
    (2, 7000.00),  
    (3, 6000.00),  
    (4, 4500.00),  
    (5, 8000.00),  
    (6, 5500.00),  
    (7, 7500.00),  
    (8, 6800.00);
```

salesperson_id	sales_amount
1	5000
2	7000
3	6000
4	4500
5	8000
6	5500
7	7500
8	6800

```
SELECT
    salesperson_id,
    sales_amount,
    NTILE(3) OVER (ORDER BY sales_amount) AS sales_bucket
FROM
    sales;
```

salesperson_id	sales_amount	sales_bucket
4	4500	1
6	5500	1
1	5000	2
3	6000	2
2	7000	2
8	6800	3
7	7500	3
5	8000	3

- **PERCENT_RANK()** - Вычисляет относительный ранг строки из группы строк в SQL.

▼ *Синтаксис*

```
PERCENT_RANK( )
OVER ( [ partition_by_clause ] order_by_clause )
```

Аргумент *order_by_clause* является обязательным. В функции PERCENT_RANK нельзя указывать <строки или предложение диапазона> синтаксиса OVER.

Процентное значение рассчитывается по формуле (ранг определяется функцией RANK - 1) / (количество целевых строк - 1), а первая строка всегда равна 0.

```
SELECT
  StudentName,
  Score,
  PERCENT_RANK() OVER(PARTITION BY Subject ORDER BY Score) AS PercentileRank
FROM
  Scores;
```

StudentName	Score	PercentileRank
Alice	80	0.33
Bob	90	0.83
Carol	85	0.66
David	75	0.16
Emily	95	1.00
Frank	70	0.00

▼ Функции смещения (Value)

Предоставляют информацию о партии окна или положении строки в нем.

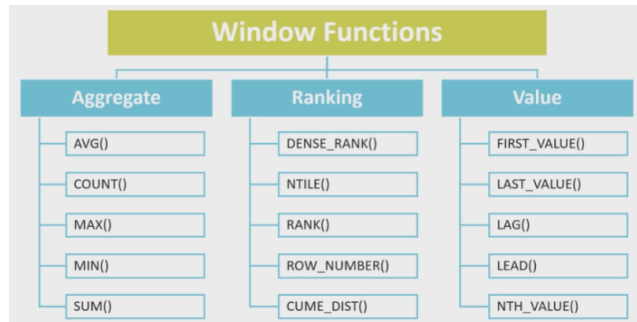
- **LAG (expression, [offset] , [default])** - функция, возвращающая предыдущее значение столбца по порядку сортировки.
 - Дефолт применяется, если мы покидаем рамки столбца и не является обязательным параметром (**NULL по умолчанию**).
 - Смещение не является обязательным параметром (**1 по умолчанию**) и **не может быть отрицательным**.
- **LEAD(value_expression, [offset], [default_value])** - функция, возвращающая следующее значение столбца по порядку сортировки.
 - Дефолт применяется, если мы покидаем рамки столбца и не является обязательным параметром (**NULL по умолчанию**).
 - Смещение не является обязательным параметром (**1 по умолчанию**) и **не может быть отрицательным**.



Обрати внимание!

Дефолтом лучше не пренебрегать. Желательно сделать его вот так LAG(id, 1, id) или LEAD(id, 1, null) в зависимости от того, что мы хотим видеть на границах. Например вот [тут](#) нам очень понадобится помнить о такой возможности.

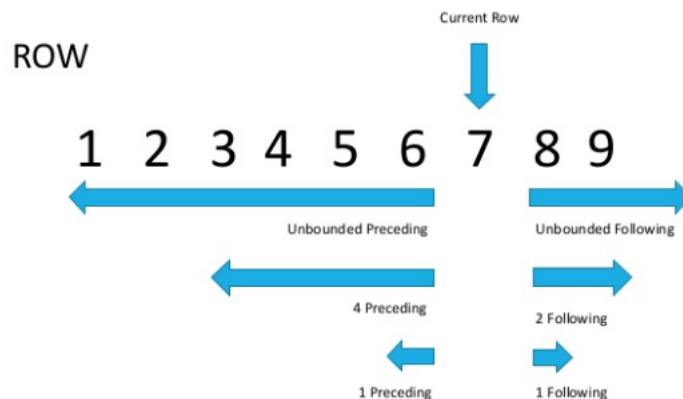
- **FIRST_VALUE(столбец)** - возвращает первое значение из упорядоченного набора значений
- **LAST_VALUE(столбец)** - возвращает последнее значение из упорядоченного набора значений



▼ Что такое ROWS и RANGE?

- **ROWS** позволяет **ограничить строки в окне**, указывая фиксированное количество строк, предшествующих или следующих за текущей.
- **RANGE**, работает не со строками, а с **диапазоном строк** в инструкции ORDER BY. То есть под одной строкой для RANGE могут пониматься несколько физических строк одинаковых по рангу.

Диапазоном называются строки с одинаковым значением параметра сортировки.



Виды ограничений внутри ROW/RANGE:

- **UNBOUNDED PRECEDING** — указывает, что окно начинается с первой строки группы;
- **UNBOUNDED FOLLOWING** — с помощью данной инструкции можно указать, что окно заканчивается на последней строке группы;
- **CURRENT ROW** — инструкция указывает, что окно начинается или заканчивается на текущей строке;
- **BETWEEN «граница окна» AND «граница окна»** — указывает нижнюю и верхнюю границу окна;
- **«Значение» PRECEDING** — определяет число строк перед текущей строкой (не допускается в предложении RANGE).;
- **«Значение» FOLLOWING** — определяет число строк после текущей строки (не допускается в предложении RANGE).

Если значение находится за пределами таблицы или группы (например, последняя строка, идет границы нижняя на following 1), то просто будут игнорироваться строки (так как нам не с чем агрегировать).

▼ Пример

```
SELECT
  Date
, Medium
, Conversions
, SUM(Conversions) OVER(PARTITION BY Date ORDER BY Conversions ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS 'Sum'
FROM Orders
```

Просто примеры синтаксиса

```
select
  object_id
, [preceding] = count(*) over(order by object_id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
, [central] = count(*) over(order by object_id ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING )
, [following] = count(*) over(order by object_id ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
from sys.objects
order by object_id asc
```

	Date	Medium	Conversions	Sum	
	10.05.2020	cpa	1	2	= 1+1
	10.05.2020	organic	1	3	= 1+2
	10.05.2020	cpc	2	2	= 2
	11.05.2020	cpa	1	2	= 1+1
	11.05.2020	direct	1	3	= 1+2
	11.05.2020	organic	2	5	= 2+3
	11.05.2020	cpc	3	3	= 3
	12.05.2020	cpc	1	3	= 1+2
	12.05.2020	organic	2	2	= 2

▼ Пример-сравнение ROW и RANGE

```
SELECT sales_id, customer_id, price_per_item, cnt,
SUM(cnt) OVER (ORDER BY customer_id, price_per_item) AS cum_uniq,
cnt,
SUM(cnt) OVER (ORDER BY customer_id, price_per_item ROWS UNBOUNDED PRECEDING) AS current_and_all_before,
customer_id,
cnt,
SUM(cnt) OVER (ORDER BY customer_id, price_per_item RANGE UNBOUNDED PRECEDING) AS current_and_all_before2
FROM sales
ORDER BY 2, price_per_item;
```

Можно посмотреть на таблицу. Параметры сортировки [customer_id, price_per_item], значит, что каждую группу будет выделен свой RANGE. Например [150 - 5000]. В current_and_all_before сразу посчитали *UNBOUNDED PRECEDING* т.е. взять предыдущий результат (12) и прибавили сразу все, что есть в группе (1 и 5) → 18. Это значение будет на весь "range".

Для ROWS вычисления всегда происходят построчно.

sales_id	customer_id	price_per_item	cnt	cum_uniq	cnt	current_and_all_before	customer_id	cnt	current_and_all_before2
2	100	5.0000	1	7	1	1	100	1	7
4	100	5.0000	5	7	5	6	100	5	7
10	100	5.0000	1	7	1	7	100	1	7
8	100	8.0000	1	8	1	8	100	1	8
6	100	17.0000	1	9	1	9	100	1	9
1	100	30.1500	2	11	2	11	100	2	11
3	100	50.0000	1	12	1	12	100	1	12
5	150	5.0000	1	18	1	13	150	1	18
11	150	5.0000	5	18	5	18	150	5	18
7	150	30.1500	2	20	2	20	150	2	20
9	170	8.0000	3	23	3	23	170	3	23
12	170	30.1500	1	24	1	24	170	1	24



Обрати внимание!

"ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW" и "ROWS UNBOUNDED PRECEDING" - одно и то же!

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW дает несколько больше ясности, при работе со сложными запросами, а "ROWS UNBOUNDED PRECEDING" просто немного короче.

Спецификация рамки по умолчанию зависит от других аспектов определения окна:

- если указано предложение **ORDER BY** и функция принимает спецификацию рамки, то спецификация рамки определяется RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW,
- в противном случае спецификация кадра определяется ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

▼ Как рассчитать скользящую среднюю с помощью оконной функции?

Пояснение: у нас есть строки с цифрами, упорядоченные в каком-то порядке. В отдельной колонке мы хотим их среднее арифметическое, учитывая то, на каком элементе мы находимся. $(a_1+a_2+a_3+...+a_n)/n$. Хотим всю такую таблицу

```
-- Create the sales_data table
CREATE TABLE sales_data (
    date DATE,
    revenue DECIMAL(10, 2)
);

-- Insert data into the sales_data table
INSERT INTO sales_data (date, revenue) VALUES
    ('2023-08-01', 100.00),
    ('2023-08-02', 150.00),
    ('2023-08-03', 200.00),
    ('2023-08-04', 120.00),
    ('2023-08-05', 180.00),
    ('2023-08-06', 250.00);

Run this script, and it will create the sales_data table and populate it with the provided data. You can then use the example query from
```

Решение: достаточно использовать оконную функцию + использовать ограничение ROW с границей "до текущего элемента"

```
SELECT
    date,
    revenue,
    AVG(revenue) OVER (ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS moving_avg
FROM
    sales_data;
```

date	revenue	moving_avg
2023-08-01	100.00	100.00
2023-08-02	150.00	125.00
2023-08-03	200.00	150.00
2023-08-04	120.00	142.50
2023-08-05	180.00	150.00
2023-08-06	250.00	175.00

По аналогии можно решать задачи:

“Предположим, у вас есть таблица `sales_data` со столбцами `date` и `revenue`, и вы хотите рассчитать 3-дневное скользящее среднее значение выручки для каждой строки.”

```
SELECT
  date,
  revenue,
  AVG(revenue) OVER (ORDER BY date ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS moving_avg
FROM
  sales_data;
```

date	revenue	moving_avg
2023-08-01	100	100.00
2023-08-02	150	125.00
2023-08-03	200	150.00
2023-08-04	120	156.67
2023-08-05	180	166.67
2023-08-06	250	183.33

▼ Вопросы по теме “Оконные Функции”

1. Что такое оконная функция?
2. В чем её отличие от агрегаций?
3. Где можно использовать оконную функцию?
4. Можно ли использовать алиас для оконной функции?
5. Какие параметры можно указать в **OVER()**?
6. Какие категории есть у оконных функций?
7. Как обрабатывают Агрегирующие оконные функции значение NULL?
8. В чем отличие **RANK**, от **DENSE_RANK** и от **ROW_NUMBER**?
9. Какая оконная функция вычисляет процентное соотношение для каждой строки относительно всей партии?
10. Что будет, если использовать каждую из ранжирующих функций без ORDER BY?
11. Что такое **NTILE** и как она работает?
12. Что будет, если количество строк в таблице не будет делиться нацело на количество корзин, которые мы указали для **NTILE** ?
13. Что будет, если количество строк в таблице будет меньше, чем количество, которые мы указали для **NTILE** ?
14. Что такое функции смещения?
15. Отличие **LAG()** от **LEAD()**? Какие параметры туда можно указать?
16. Какие значения по умолчанию установлены для не обязательных параметров **LAG()** и **LEAD()**?
17. Как будет работать **LAG()** и **LEAD()** при с первой и последней строкой?
18. Что такое **ROW** и **RANGE**?
19. Какие виды ограничений есть для **ROW** и **RANGE**?

20. Как будут работать ограничения у **ROW** и **RANGE**, если мы выйдем за границы (группы или таблицы)?
21. Как рассчитать скользящую среднюю с помощью оконной функции?
22. В чем отличие **"ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW"** и **"ROWS UNBOUNDED PRECEDING"**?
23. Какие значения по дефолту по **ROW/RANGE** в **OVER**
24. Найти топ три зарплаты в каждом отделе

▼ Задачи по теме “Оконные Функции”

- ▼ **[Top N with Grouping]** Write a query to find the top 3 salaries within each department using the RANK() function.

```
-- Create the employees table
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(50),
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);

-- Insert data into the employees table
INSERT INTO employees (employee_id, employee_name, department, salary) VALUES
    (1, 'John', 'HR', 60000.00),
    (2, 'Jane', 'IT', 75000.00),
    (3, 'Alice', 'HR', 55000.00),
    (4, 'Bob', 'IT', 80000.00),
    (5, 'Carol', 'HR', 62000.00),
    (6, 'David', 'IT', 72000.00);
```

Примечание: так как мы не можем использовать оконные функции ни с чем по фильтрации, то придется делать вложенные запросы. Также надо иметь в виду, что у собеседующего надо уточнить, что делать, если три человека получают 60к (нам нужно топ три разных зп или любых? а что делать, если два сотрудника получают 100, а другие два 99 и так далее).

Примерное решение выглядит так:

```
SELECT employee_id, employee_name, department, salary
FROM (
    SELECT employee_id, employee_name, department, salary,
           DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS ranking
    FROM employees
) ranked
WHERE ranking <= 3;
```

- ▼ **[Running Total]** Running Total For Different Genders Problem (Leetcode Medium)

```
--таблица
CREATE TABLE player_scores (
    player_name VARCHAR(255),
    gender CHAR(1),
    day DATE,
    score_points INT
);

INSERT INTO your_table_name (player_name, gender, day, score_points)
VALUES
    ('Aron', 'F', '2020-01-01', 17),
    ('Alice', 'F', '2020-01-07', 23),
    ('Bajrang', 'M', '2020-01-07', 7),
    ('Khalil', 'M', '2019-12-25', 11),
    ('Slaman', 'M', '2019-12-30', 13),
    ('Joe', 'M', '2019-12-31', 3),
    ('Jose', 'M', '2019-12-18', 2),
    ('Priya', 'F', '2019-12-31', 23),
    ('Priyanka', 'F', '2019-12-30', 17);
```

```
+-----+-----+
| Column Name | Type |
+-----+-----+
```

```

| player_name | varchar |
| gender      | varchar |
| day         | date    |
| score_points | int     |
+-----+
(gender, day) is the primary key for this table.
A competition is held between females team and males team.
Each row of this table indicates that a player_name and with gender has scored score_point in someday.
Gender is 'F' if the player is in females team and 'M' if the player is in males team.

```

Write an SQL query to find the total score for each gender at each day. Return the result table ordered by **gender** and **day** in **ascending order**.

```

SELECT
  gender,
  day,
  SUM(score_points) OVER(PARTITION BY gender
    ORDER BY gender, day) as total
FROM player_scores

```

▼ [Calculating Running/Moving Average in SQL] Найти подвижный результат (с определенной границей)

<http://www.silota.com/docs/recipes/sql-running-average.html>

```

-- Create the amazon_revenue table
CREATE TABLE amazon_revenue (
  quarter DATE PRIMARY KEY,
  revenue DECIMAL(10, 3)
);

-- Insert data into the amazon_revenue table
INSERT INTO amazon_revenue (quarter, revenue) VALUES
  ('2001-1', 700.356),
  ('2001-2', 667.625),
  ('2001-3', 639.281),
  ('2001-4', 1115.171),
  -- Add more rows for other quarters here
  ('2008-3', 4265.000),
  ('2008-4', 6703.000);
This script will create the "amazon_revenue" table with columns for "quarter" and "revenue" and then insert the provided data into




```

Надо найти “подвижное среднее”, где для каждой строки рассчитывается среднее за три квартала:

```

SELECT
  quarter,
  revenue,
  AVG(revenue) OVER (ORDER BY quarter ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) AS moving_avg
FROM
  amazon_revenue
ORDER BY
  quarter;

```

	quarter [PK] character varying (10) 	revenue numeric (10,3) 	moving_avg numeric 
1	2001-1	700.356	700.3560000000000000
2	2001-2	667.625	683.9905000000000000
3	2001-3	639.281	669.0873333333333333
4	2001-4	1115.171	780.6082500000000000
5	2008-3	4265.000	1671.7692500000000000
6	2008-4	6703.000	3180.6130000000000000

Обычно используют для решения сл. типов:

1. Сравнить среднюю зп с общей внутри департамента (и вывести всю таблицу вместе)

<http://thisisdata.ru/blog/uchimsya-primenyat-okonnyye-funktsii/#:~:text=посчитали нарастающий итог-,ROWS или RANGE,строк в инструкции ORDER BY.>

<https://www.stratascratch.com/blog/sql-window-functions-interview-questions/>

Порешать онлайн:

1. https://platform.stratascratch.com/coding/9917-average-salaries?code_type=1
2. https://platform.stratascratch.com/coding/10172-best-selling-item?code_type=1
3. <https://lifewithdata.com/2022/05/25/sql-interview-questions-running-total-for-different-genders/>

▼ На разбор

- ☐ <https://stackoverflow.com/questions/35096414/why-doesnt-last-value-return-the-last-value> - на разбор. В чём проблема описанного случая: Last_value возвращал не то. Дело в том, что без указания области “окна”, иногда она просто берёт default значения, что для аналитических функций является текущим выбранным столбцом. Решение: задать область, выбрав нужные столбцы (в конкретном примере нужно было вернуть значение по всей таблице, поэтому правильно было указать ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING).