

Оконные функции

Оконные функции

Оконная функция выполняет вычисления для набора строк, некоторым образом связанных с текущей строкой. Её действие можно сравнить с вычислением, производимым агрегатной функцией. Однако с оконными функциями строки не группируются в одну выходную строку, что имеет место с обычными, не оконными, агрегатными функциями. Вместо этого, эти строки остаются отдельными сущностями. Внутри же, оконная функция, как и агрегатная, может обращаться не только к текущей строке результата запроса.

- Позволяют обрабатывать группы строк без образования группировок в результирующем наборе (без использования GROUP BY)
- Делятся на: агрегатные (SUM, AVG, MIN, MAX, COUNT) и ранжирования (ROW_NUMBER, RANK, LAG, LEAD)
- Отрабатывают после JOIN, WHERE, GROUP BY, HAVING, но перед ORDER BY

Оконные функции

Окно – некоторое выражение, описывающее набор строк, которые будет обрабатывать функция и порядок этой обработки

function OVER (expression)

При отсутствии *expression* результатом будут являться все строки, которые возвращает запрос

function OVER ([PARTITION BY expression], [ORDER BY expression])

PARTITION BY – группировка, ORDER BY – сортировка

SELECT

Название функции (столбец для вычислений)

OVER (

PARTITION **BY** столбец для группировки






ORDER BY столбец для сортировки

ROWS или RANGE выражение для ограничения строк в пределах группы


)

Оконные функции

```
CREATE TABLE public."Employee"  
(  
    id bigint NOT NULL,  
    name character(20) COLLATE pg_catalog."default",  
    otdel character(20) COLLATE pg_catalog."default",  
    oklad bigint,  
    CONSTRAINT "Employee_pkey" PRIMARY KEY (id)  
)
```

	 id [PK] bigint 	name character (20) 	otdel character (20) 	oklad bigint 
1	1	Иванов Иван	1	500
2	2	Петров Петр	1	250
3	3	Павлов Павел	1	750
4	4	Романов Роман	2	200
5	5	Николаев Никола...	2	300

```
SELECT name, oklad, ROUND(AVG (oklad) OVER (PARTITION BY otdeel)::numeric,2) AS avg_oklad  
FROM "Employee"
```

	name character (20) 	oklad bigint 	avg_oklad numeric 
1	Иванов Иван	500	500.00
2	Петров Петр	250	500.00
3	Павлов Павел	750	500.00
4	Романов Роман	200	250.00
5	Николаев Никола...	300	250.00

Оконные функции

```
SELECT name, oklad, otdel, ROUND(SUM (oklad) OVER (PARTITION BY otdel ORDER BY name)::numeric,2)  
AS sum_oklad  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	sum_oklad numeric
1	Иванов Иван	500	1	500.00
2	Павлов Павел	750	1	1250.00
3	Петров Петр	250	1	1500.00
4	Николаев Никола...	300	2	300.00
5	Романов Роман	200	2	500.00

```
SELECT name, oklad, otdel, ROUND(SUM (oklad) OVER (PARTITION BY otdel ORDER BY oklad)::numeric,2)  
AS sum_oklad  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	sum_oklad numeric
1	Петров Петр	250	1	250.00
2	Иванов Иван	500	1	750.00
3	Павлов Павел	750	1	1500.00
4	Романов Роман	200	2	200.00
5	Николаев Никола...	300	2	500.00

Оконные функции

```
SELECT name, oklad, otdel, ROUND(SUM (oklad) OVER ()::numeric,2) AS sum_oklad  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	sum_oklad numeric
1	Иванов Иван	500	1	2250.00
2	Павлов Павел	750	1	2250.00
3	Романов Роман	200	2	2250.00
4	Николаев Никола...	300	2	2250.00
5	Петров Петр	500	1	2250.00

Есть ещё одно важное понятие, связанное с оконными функциями: для каждой строки существует набор строк в её разделе, называемый **рамкой окна**. Некоторые оконные функции обрабатывают только строки рамки окна, а не всего раздела.

Так как в этом примере нет указания **ORDER BY** в предложении **OVER**, рамка окна содержит все строки раздела, а он, в свою очередь, без предложения **PARTITION BY** включает все строки таблицы; другими словами, сумма вычисляется по всей таблице и мы получаем один результат для каждой строки результата.

Оконные функции

```
SELECT name, oklad, otdel, ROUND(SUM (oklad) OVER (ORDER BY name)::numeric,2) AS sum_oklad  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	sum_oklad numeric
1	Иванов Иван	500	1	500.00
2	Николаев Никола...	300	2	800.00
3	Павлов Павел	750	1	1550.00
4	Петров Петр	500	1	2050.00
5	Романов Роман	200	2	2250.00

Здесь в сумме накапливаются зарплаты от первой (самой низкой) до текущей, включая повторяющиеся текущие значения (обратите внимание на результат в строках с одинаковой зарплатой).

ROWS или RANGE

Инструкция **ROWS** позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей.

Инструкция **RANGE**, в отличие от **ROWS**, 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. Обе инструкции **ROWS** и **RANGE** всегда используются вместе с **ORDER BY**.

В выражении для ограничения строк **ROWS** или **RANGE** также можно использовать следующие ключевые слова:

- **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**).

Оконные функции

Когда в запросе вычисляются несколько оконных функций для одинаково определённых окон, конечно можно написать для каждой из них отдельное предложение OVER, но при этом оно будет дублироваться, что неизбежно будет провоцировать ошибки. Поэтому лучше определение окна выделить в предложение WINDOW, а затем ссылаться на него в OVER

```
SELECT sum(salary) OVER w, avg(salary) OVER w  
FROM empsalary  
WINDOW w AS (PARTITION BY depname ORDER BY salary DESC);
```

```
CREATE TABLE employees (  
    "department" TEXT,  
    "name" TEXT,  
    "salary" INT);
```

```
INSERT INTO employees  
SELECT 'dep_' || chr(d), 'emp_' || chr(d) || e, d*10 + e*1  
FROM generate_series(ascii('a'), ascii('c')) AS d,  
     generate_series (1,3) AS e;
```

```
SELECT name, salary, array_agg(salary) OVER w  
FROM employees  
WINDOW w AS (PARTITION BY department);
```

	name text	salary integer	array_agg integer[]
1	emp_a1	971	{971,972,973}
2	emp_a2	972	{971,972,973}
3	emp_a3	973	{971,972,973}
4	emp_b1	981	{981,982,983}
5	emp_b2	982	{981,982,983}
6	emp_b3	983	{981,982,983}
7	emp_c1	991	{991,992,993}
8	emp_c2	992	{991,992,993}
9	emp_c3	993	{991,992,993}

```
SELECT name,department, salary, sum(salary) OVER w  
FROM employees  
WINDOW w AS (  
    PARTITION BY department  
    ORDER BY salary  
    ROWS BETWEEN 1 PRECEDING AND CURRENT ROW EXCLUDE  
    CURRENT ROW  
);
```

	name text	department text	salary integer	sum bigint
1	emp_a1	dep_a	971	[null]
2	emp_a2	dep_a	972	971
3	emp_a3	dep_a	973	972
4	emp_b1	dep_b	981	[null]
5	emp_b2	dep_b	982	981
6	emp_b3	dep_b	983	982
7	emp_c1	dep_c	991	[null]
8	emp_c2	dep_c	992	991
9	emp_c3	dep_c	993	992

Без EXCLUDE CURRENT ROW

	name text	department text	salary integer	sum bigint
1	emp_a1	dep_a	971	971
2	emp_a2	dep_a	972	1943
3	emp_a3	dep_a	973	1945
4	emp_b1	dep_b	981	981
5	emp_b2	dep_b	982	1963
6	emp_b3	dep_b	983	1965
7	emp_c1	dep_c	991	991
8	emp_c2	dep_c	992	1983
9	emp_c3	dep_c	993	1985

```
SELECT x, array_agg(x) OVER w
FROM generate_series(1,3) AS x
WINDOW w AS (
  ORDER BY x
  ROWS BETWEEN CURRENT ROW AND CURRENT ROW
);
```

	<div><div>x</div><div>integer</div></div>		<div><div>array_agg</div><div>integer[]</div></div>
1	1		{1}
2	2		{2}
3	3		{3}





```
SELECT x, array_agg(x) OVER w
FROM generate_series(1,3) AS x
WINDOW w AS (
  ORDER BY x
  ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
);
```

	<div><div>x</div><div>integer</div></div>		<div><div>array_agg</div><div>integer[]</div></div>
1	1		{1}
2	2		{1,2}
3	3		{2,3}





```
SELECT x, array_agg(x) OVER w
FROM generate_series(1,3) AS x
WINDOW w AS (
  ORDER BY x
  ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING
);
```

	<div><div>x</div><div>integer</div></div>		<div><div>array_agg</div><div>integer[]</div></div>
1	1		{1,2}
2	2		{2,3}
3	3		{3}

```
SELECT INT4(x > 3), x, array_agg(x) OVER w
FROM generate_series(1,6) AS x
WINDOW w AS (
  PARTITION BY x > 3
  ORDER BY x
);
```

	 int4 integer		x integer		array_agg integer[]	
1		0		1	{1}	
2		0		2	{1,2}	
3		0		3	{1,2,3}	
4		1		4	{4}	
5		1		5	{4,5}	
6		1		6	{4,5,6}	

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

	int4 integer		x integer		array_agg integer[]	
1		0		1	{1,2,3}	
2		0		2	{1,2,3}	
3		0		3	{1,2,3}	
4		1		4	{4,5,6}	
5		1		5	{4,5,6}	
6		1		6	{4,5,6}	

Оконные функции

- ROW_NUMBER – присвоение уникального значения строкам
- RANK – присвоение ранга (веса) строкам с пропусками
- DENSE_RANK - присвоение ранга (веса) строкам без пропусков
- LANG – присвоение значения текущей строке, основанное на значении в предыдущей
- LEAD - присвоение значения текущей строке, основанное на значении в следующей

LANG и LEAD имеет три параметра: столбец, значение которого необходимо вернуть, количество строк для смещения (*по умолчанию 1*), значение, которое необходимо вернуть если после смещения возвращается значение NULL.

```

SELECT row_number() OVER w, name, salary, sum(salary) OVER w
FROM employees
WINDOW w AS (
  PARTITION BY department
  ORDER BY salary
  ROWS BETWEEN 1 PRECEDING AND CURRENT ROW EXCLUDE CURRENT ROW
);

```

	row_number bigint	name text	salary integer	sum bigint
1	1	emp_a1	971	[null]
2	2	emp_a2	972	971
3	3	emp_a3	973	972
4	1	emp_b1	981	[null]
5	2	emp_b2	982	981
6	3	emp_b3	983	982
7	1	emp_c1	991	[null]
8	2	emp_c2	992	991
9	3	emp_c3	993	992

Оконные функции

```
SELECT name, oklad, otdel, RANK() OVER (ORDER BY oklad)  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	rank bigint
1	Романов Роман	200	2	1
2	Николаев Никола...	300	2	2
3	Иванов Иван	500	1	3
4	Петров Петр	500	1	3
5	Павлов Павел	750	1	5

RANK — функция возвращает ранг каждой строки. В данном случае значения уже анализируются и, в случае нахождения одинаковых, возвращает одинаковый ранг с пропуском следующего значения

```
SELECT name,  
       salary,  
       rank() OVER w1 AS company_rank,  
       rank() OVER w2 AS department_rank  
FROM employees  
WINDOW w1 AS (ORDER BY salary DESC),  
       w2 AS (PARTITION BY department ORDER BY salary DESC);
```

	name text	salary integer	company_rank bigint	department_rank bigint
1	emp_c3	993	1	1
2	emp_c2	992	2	2
3	emp_c1	991	3	3
4	emp_b3	983	4	1
5	emp_b2	982	5	2
6	emp_b1	981	6	3
7	emp_a3	973	7	1
8	emp_a2	972	8	2
9	emp_a1	971	9	3

```
w2 AS (PARTITION BY department ORDER BY salary);
```

	name text	salary integer	company_rank bigint	department_rank bigint
1	emp_c3	993	1	3
2	emp_c2	992	2	2
3	emp_c1	991	3	1

Оконные функции

```
SELECT name, oklad, otdel, DENSE_RANK() OVER (ORDER BY oklad)  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	dense_rank bigint
1	Романов Роман	200	2	1
2	Николаев Никола...	300	2	2
3	Иванов Иван	500	1	3
4	Петров Петр	500	1	3
5	Павлов Павел	750	1	4

DENSE_RANK — функция возвращает ранг каждой строки. Но в отличие от функции RANK, она для одинаковых значений возвращает ранг, не пропуская следующий.

Оконные функции

Если нужно отфильтровать или сгруппировать строки после вычисления оконных функций, можно использовать вложенный запрос

```
SELECT *  
FROM  
(SELECT name, oklad, otдел, RANK() OVER (ORDER BY oklad) AS  
rank_oklad  
FROM table2) as subquery  
WHERE rank_oklad >= 2
```

Данный запрос покажет только те строки внутреннего запроса, у которых rank (порядковый номер) больше или равен 2.

```
SELECT *  
FROM  
(SELECT name, oklad, otдел, DENSE_RANK() OVER (ORDER BY oklad) AS rank_oklad  
FROM table2) as subquery  
WHERE rank_oklad >= 2
```

Оконные функции

SELECT *

FROM

(SELECT name, oklad, otdel, **DENSE_RANK()** OVER (PARTITION BY otdel ORDER BY oklad) as rank_oklad

FROM "Employee") as subquery

WHERE rank_oklad >= 2

	<div>name</div> <div>character (20)</div>	<div>oklad</div> <div>bigint</div>	<div>otdel</div> <div>character (20)</div>	<div>rank_oklad</div> <div>bigint</div>
1	Павлов Павел	750	1	2
2	Николаев Никола...	300	2	2

Оконные функции

```
SELECT name, oklad, otdel, DENSE_RANK()  
OVER (ORDER BY  
CASE  
WHEN oklad<600 THEN 1  
WHEN oklad>600 THEN 2  
ELSE 3  
END)  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	dense_rank bigint
1	Иванов Иван	500	1	1
2	Романов Роман	200	2	1
3	Николаев Никола...	300	2	1
4	Петров Петр	500	1	1
5	Павлов Павел	750	1	2

Оконные функции

SELECT name, oklad, otdel, **LAG(oklad) OVER (ORDER BY oklad)** AS oklad_lag
FROM "Employee"

	name character (20)	oklad bigint	otdel character (20)	oklad_lag bigint
1	Романов Роман	200	2	[null]
2	Николаев Никола...	300	2	200
3	Иванов Иван	500	1	300
4	Петров Петр	500	1	500
5	Павлов Павел	750	1	500

Оконные функции

```
SELECT name, oklad, otdel, oklad-LAG(oklad) OVER (ORDER BY oklad) AS oklad_lag  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	oklad_lag bigint
1	Романов Роман	200	2	[null]
2	Николаев Никола...	300	2	100
3	Иванов Иван	500	1	200
4	Петров Петр	500	1	0
5	Павлов Павел	750	1	250

Оконные функции

```
SELECT name, oklad, otдел, oklad-LEAD(oklad) OVER (ORDER BY oklad) AS oklad_lag  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	oklad_lag bigint
1	Романов Роман	200	2	-100
2	Николаев Никола...	300	2	-200
3	Иванов Иван	500	1	0
4	Петров Петр	500	1	-250
5	Павлов Павел	750	1	[null]

Оконные функции

```
SELECT name, oklad, otdel, oklad-LEAD(oklad,2) OVER (ORDER BY oklad) AS oklad_lag  
FROM "Employee"
```

	name character (20)	oklad bigint	otdel character (20)	oklad_lag bigint
1	Романов Роман	200	2	-300
2	Николаев Никола...	300	2	-200
3	Иванов Иван	500	1	-250
4	Петров Петр	500	1	[null]
5	Павлов Павел	750	1	[null]


```

SELECT name, salary, lag(salary, 1) OVER w, lead(salary, 1) OVER w
FROM employees
WINDOW w AS (
    PARTITION BY department
    ORDER BY salary
    ROWS BETWEEN 1 PRECEDING AND CURRENT ROW EXCLUDE CURRENT ROW
);

```

	name text	salary integer	lag integer	lead integer
1	emp_a1	971	[null]	972
2	emp_a2	972	971	973
3	emp_a3	973	972	[null]
4	emp_b1	981	[null]	982
5	emp_b2	982	981	983
6	emp_b3	983	982	[null]
7	emp_c1	991	[null]	992
8	emp_c2	992	991	993
9	emp_c3	993	992	[null]

lag() и lead() работают на уровне партиции (группы), и им не важно, какое условие было указано в BETWEEN

```
SELECT name,  
       salary,  
       first_value(salary) OVER w,  
       last_value(salary) OVER w  
FROM employees  
WINDOW w AS (  
  PARTITION BY department  
  ORDER BY salary  
  ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING  
);
```

Функции first_value() и last_value() работают с фреймом и учитывают эти условия.




	name text	salary integer	first_value integer	last_value integer
1	emp_a1	971	971	973
2	emp_a2	972	971	973
3	emp_a3	973	971	973
4	emp_b1	981	981	983
5	emp_b2	982	981	983
6	emp_b3	983	981	983
7	emp_c1	991	991	993
8	emp_c2	992	991	993
9	emp_c3	993	991	993

FIRST_VALUE или **LAST_VALUE** — с помощью функций можно получить первое и последнее значение в окне. В качестве параметра принимает столбец, значение которого необходимо вернуть.

ORDER BY salary DESC

	name text	salary integer	first_value integer	last_value integer
1	emp_a3	973	973	971
2	emp_a2	972	973	971
3	emp_a1	971	973	971

```
SELECT name, salary, nth_value(salary, 2) OVER w
FROM employees
WINDOW w AS (
  PARTITION BY department
  ORDER BY salary
  ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
);
```

	 name text	 salary integer	 nth_value integer
1	emp_a1	971	972
2	emp_a2	972	972
3	emp_a3	973	972
4	emp_b1	981	982
5	emp_b2	982	982
6	emp_b3	983	982
7	emp_c1	991	992
8	emp_c2	992	992
9	emp_c3	993	992