**ЗАПРОСЫ**

|  |  |  |
| --- | --- | --- |
| SQL | Neo4j | PostgeSQL |
| select \* from emp; | match (a:employees) return a |  |
| select \* from dept; | match (a:departments) return a |  |
| select ename, sal from emp; | MATCH (a:employees)RETURN a.ENAME, a.sal |  |
| Select ename from emp; | MATCH (a:employees)RETURN a.ENAME |  |
| Select \* from emp where ename = ‘SMITH’; | MATCH (a:employees {ENAME:'SMITH'})  RETURN a | Select \* from emp where ename = 'SMITH'; |
| Select \* from emp where empno in (7369, 15, 46, 16); | MATCH (n)  WHERE id(n) IN [0, 3, 5]  RETURN n |  |
| Select dept.deptno, emp.empno from dept join emp on dept.deptno = emp.deptno where dname = ‘ACCOUNTING’; | MATCH ()-[r]->()  WHERE id(r) = 0  RETURN r |  |
| Select dept.deptno, emp.empno from dept join emp on dept.deptno = emp.deptno where dname = ‘WORKS’; | MATCH ()-[r]->()  WHERE id(r) = 15  RETURN r | Select dept.deptno, emp.empno from dept join emp on dept.deptno = emp.deptno where dname = 'WORKS'; |
| Select \* from dept join emp on dept.deptno = emp.deptno where dname = ‘ACCOUNTING’; | MATCH (a)-[r]-(b)  WHERE id(r) = 0  RETURN a, b |  |
| Select \* from dept join emp on dept.deptno = emp.deptno where dname = ‘RESEARCH; | MATCH (a)-[r]-(b)  WHERE id(r) = 5  RETURN a, b |  |
| Select \* from emp join dept on emp.empno = 7369 and dept.deptno = 20; | MATCH (a:employees),(b:departments) WHERE a.EMPNO = 7369 AND b.DEPNO = 20 return a,b |  |
| select a.deptno, b.ename from dept a, emp b | MATCH (a:employees),(b:departments) return a.ENAME, b.DEPNO |  |
| Select a, b from dept a, emp b; | MATCH (a:employees), (b:departments) RETURN a, b |  |
| select a.deptno, b.ename from dept a, emp b | MATCH (a:employees), (b:departments) RETURN a.ENAME, b.DEPTNO |  |
| Select \* from dept join emp on dept.deptno = emp.deptno where emp.ename = ‘SMITH’; | MATCH (:employees {ENAME: 'SMITH'})--(m:departments)  RETURN m |  |
| **ВЫБОРКА(SELECT) (σ):**  SELECT \* from EMP WHERE SAL>2500; | MATCH (a:employees) WHERE a.SAL > 2500 return a; |  |
| **ПРОЕКЦИЯ(PROJECT) (π):**  SELECT DISTINCT ENAME,DEPTNO from EMP;  SELECT DISTINCT DEPTNO, ENAME from EMP;  SELECT DISTINCT DEPTNO from EMP;  **ПРОИЗВЕДЕНИЕ(PRODUCT) (×):**  SELECT \* from EMP,DEPT; | MATCH (a:employees) return DISTINCT a.ENAME, a.DEPTNO;  MATCH (a:employees) return DISTINCT a.ENAME, a.DEPTNO;  MATCH (a:employees) return DISTINCT a.DEPTNO; |  |
| **ОБЪЕДИНЕНИЕ(UNION) (∪):**  SELECT DEPTNO,DNAME, LOC from DEPT;  SELECT ENAME, DEPTNO from EMP UNION SELECT DNAME, DEPTNO from DEPT;  SELECT DEPTNO from EMP UNION SELECT DEPTNO from DEPT; | ? |  |
| **ПЕРЕСЕЧЕНИЕ(INTERSECT) (∩):**  SELECT DEPTNO from EMP INTERSECT SELECT DEPTNO from DEPT; | ? |  |
| **СОЕДИНЕНИЕ(JOIN) (∏):**  SELECT \* from DEPT, EMP WHERE SAL>3000; | ? |  |

|  |  |
| --- | --- |
| SQL | PostgeSQL |
| 1. Выберите сотрудников 20 отдела  select \*  from emp  where deptno = 20;  2. Укажите имя, номер и название отдела для «клерков».  select e.ENAME Name, e.Empno number, e.job, e.DEPTNO number,  (select d.dname from dept d where e.deptno = d.deptno)  from emp e  where e.job = 'CLERK';  3. Найдите сотрудников с комиссионными выше зарплаты.  select \*  from emp  where comm > sal;  4. Выясните, у кого комиссия выше 60% от зарплаты сотрудника.  select e.\*, e.sal, e.sal \* 0.6  from emp e  where comm > e.sal \* 0.6  5. Узнайте обо всех офисных работниках  отдела 10 и отдела 20  select \*  from emp  where (deptno = 10 and job = 'MANAGER')  or  (deptno = 20 and job = 'CLERK');  6. Выбрать информацию  из emp  где comm не равен нулю и comm > 0,  группировать по должностям;  select job  from emp  where comm is not null and comm > 0  group by job;  7. Найдите информацию, где не взимается комиссий или сборов  select \*  from emp  where comm is null or comm < 100;  8. Найдите всех сотрудников, занятых на последний день каждого месяца.  select last\_day(sysdate) from dual;  select \*  from emp  where hiredate = last\_day(hiredate);  9, Find out earlier than25Employees employed earlier  select \*  from emp  where hiredate < add\_months(sysdate, -25\*12);  Or  select \*  from emp  where add\_months(hiredate, 25\*12) < sysdate;  10, Display only all employees with the first letters  select ename, substr(ename,1,1)  from emp  where substr(ename,1,1) >= 'A' and substr(ename,1,1) <= 'Z';  11The showing is just6Employee name  select ename  from emp  where length(ename) = 6;  12, Display without'R'Employee name  select ename  from emp  where ename not like '%R%';  13, Showing the first three characters of all employees  select substr(ename, 1, 3)  from emp;  14, Display all the names of all employees, replace all with A'A'  select ename, replace(ename, 'A', 'a')  from emp;  15, Display all the names of all employees and full10Date of annual service years  select ename, add\_months(hiredate, 10 \* 12)  from emp;  16, Display the detailed information of the employee, sort by name  select \*  from emp  order by ename;  17Display the name of the employee, according to its service years, the oldest employee is in front  select \*  from emp  order by hiredate;  18Display all employees' name, work, salary, order in descending order, while the same order is ascended  select ename,job,sal  from emp  order by job desc, sal asc;  19Show all employees' names and the year of joining the company, sorting in the month of employees in employees,  Put the earliest items in the forefront  select e.ename, extract(Year from e.hiredate) y\_ear,  extract(Month from e.hiredate) m\_onth  from emp e order by y\_ear asc, m\_onth asc;  20, Displayed in one month30All employees' day salary  select e.\*, round(nvl (sal, 0) / 30, 2) Annuity  from emp e;  21, Find out (any year)2All employees hired by month  select \*  from emp  where extract(MONTH FROM hiredate) = 2;  22For each employee, show its number of days to join the company  - The result of the date reduction is the number of days  select round(sysdate - hiredate)  from emp;  23, Display any location of the name field, contain "A" All employees named  select ename  from emp  where ename like '%A%';  24, Year, month, and day show all employees' service years  select  trunc(sysdate - hiredate) / 365 years,  trunc(sysdate - hiredate) / 30 month,  trunc(sysdate - hiredate) Day number  from emp;  25, Query the employee number, name, position, annual salary of the EMP table(Salary\*13[13salary] +  subsidy(300[Meal] + 180[Transport Allowance])\*12)。  select Empno number, ENAME Name, Job position, sal \* 13 +  nvl(comm, 0) \* 12 + (300 + 180) \* 12 Annual salary  from emp;  26, Inquiry in the EMP table, the basic salary is higher than1500All information about employees  select \*  from emp  where sal > 1500  27, Inquiry EMP table, basic salary<Do not>In 1200~2500All information between employees  select \*  from emp  where sal not between 1200 and 2500;  28, Inquiry in the EMP table, all time HIREDATE is1981Annual employment information.1981Year:  （1981-01-01~1981-12-31）  select \*  from emp  where hiredate between to\_date('1981-01-01','yyyy-mm-dd') and  to\_date('1981-12-31','yyyy-mm-dd');  29In the query EMP table, the number is 7499 Or 7844 Or 9999 Or 7566 Employee information.  select \*  from emp  where empno in (7499, 7844, 9999, 7566);  30, Inquiry EMP table, employee name is employee information ending with S character  select \*  from emp  where ename like '%S';  31In the query EMP table, the second character named by the employee is an employment information of A.  select \*  from emp  where ename like '\_A%';  32Query different types of work in each department in the EMP table, and the results are ascended in accordance with department numbers  select deptno,job  from emp  group by deptno,job  order by deptno;  33, The number of different types under each department in the EMP table is greater than or equal to2Record  select deptno,job,count(1)  from emp  group by deptno,job  having count(1) >= 2;  34, Query the salary in the EMP table3000Number of employees above  select count(1)  from emp  where sal >= 3000;  35, Query all sales sum of all salespersons in the EMP table  select sum(sal)  from emp  where job = 'SALESMAN';  36, Check out all the annual salary summates of all salesmen and all managers' total salary  (Salary = Salaries per month13Salary + Commission + 300Meal + 200Transport Allowance)  - Packet and polymerization  select job,sum(sal \* 13 + nvl(comm,0) \* 12 + (300 + 200) \* 12)  from emp  where job = 'SALESMAN' or job = 'MANAGER'  group by job;  37, Query the sum of all employees under various departments, greater than120000Number of people,  (Salary = Salaries per month13Salary + Commission + 300Meal + 200Transport Allowance)  select deptno,sum(sal \* 13 + nvl(comm,0) \* 12 +  (300 + 200) \* 12) Payroll, count(1)  from emp  group by deptno  having sum(sal \* 13 + nvl(comm,0) \* 12 + (300 + 200) \* 12) > 120000;  38, Query the salary1500Below below,1500yuan~3000Between the yuan,3000Number of people above Yuan  select  sum(case when sal < 1500 then 1 else 0 end),  sum(case when sal between 1500 and 3000 then 1 else 0 end),  sum(case when sal > 3000 then 1 else 0 end)  from emp;  Or  select  count(case when sal < 1500 then 1 else null end),  count(case when sal between 1500 and 3000 then 1 else null end),  count(case when sal > 3000 then 1 else null end)  from emp;  39, Query employee information with the highest salary DELETE  - subquery  select \*  from emp  where sal = (select max(sal) from emp);  40Issue the number of people under all employees under various departments, as well as average salary, maximum wage, minimum wage, Total salary  select deptno,count(1), round(avg(sal),2), max(sal), min(sal), sum(sal)  from emp  group by deptno;  41. The highest average salary of all employees under each position  select job, round(avg(sal))  from emp  group by job;  select max(round(avg(sal), 2))  from emp  group by job;  42. Number of positions in statistical employee table  select job,count(1)  from emp  group by job;  or  select distinct job  from emp  select count(distinct job)  from emp; | select extract (day from now());  create or replace function last\_day(date) returns date as 'select  cast(date\_trunc(''month'', $1) + ''1 month''::interval as date) - 1'  language sql;  select e.\*, round(COALESCE(sal, 0) / 30, 2) Annuity  from emp e;  select  (now() - hiredate) / 365 y\_ears,  (now() - hiredate) / 30 m\_onth,  (now() - hiredate) D\_ay  from emp;  select Empno number, ENAME Name, Job position, sal \* 13 +  COALESCE(comm, 0) \* 12 + (300 + 180) \* 12 Annual\_salary  from emp;  select job,sum(sal \* 13 + COALESCE(comm,0) \* 12 + (300 + 200) \* 12)  from emp  where job = 'SALESMAN' or job = 'MANAGER'  group by job; |

SELECT [ ALL | DISTINCT | DISTINCT ON (distinct\_expressions) ] expressions FROM tables [WHERE conditions] [GROUP BY expressions] [HAVING condition] [ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS FIRST | NULLS LAST ]] [LIMIT [ number\_rows | ALL] [OFFSET offset\_value [ ROW | ROWS ]] [FETCH { FIRST | NEXT } [ fetch\_rows ] { ROW | ROWS } ONLY] [FOR { UPDATE | SHARE } OF table [ NOWAIT ]];

* **ALL:**Это необязательный параметр, который поможет вернуть все совпадающие строки на основе упомянутого условия.
* **DISTINCT:** Это необязательный параметр, который используется для удаления повторяющихся значений из результирующего набора.
* **DISTINCT ON:**Это необязательный параметр, который используется для возврата выходных данных путем удаления дубликатов на основе **distinct\_expressions**.
* **expressions:** Это обязательный параметр, который должен использоваться вместе с инструкцией PostgreSQL SELECT. Он представляет все столбцы или вычисления, из которых вы хотите извлечь данные. Даже вы можете использовать знак \*, если хотите извлечь данные из всех столбцов одной или нескольких таблиц.
* **tables:** • Это обязательный параметр, который должен использоваться вместе с оператором PostgreSQL SELECT, т.е. после предложения FROM должна быть хотя бы одна таблица. Он представляет собой имя одной или нескольких таблиц, из которых вы хотите извлечь данные.
* **WHERE conditions:** Это необязательный параметр. Наряду с предложением WHERE вы можете указать условия, на основании которых вы хотите извлекать записи.
* **GROUP BY expressions:**• Это необязательный параметр. Наряду с предложением GROUP BY вы можете указать названия столбцов или выражения, на основе которых вы можете сгруппировать результаты.
* **HAVING condition:**Это необязательный параметр. Он используется вместе с предложением GROUP BY для ограничения возвращаемых строк на основе условия, которое имеет значение true.
* **ORDER BY expression:**Это необязательный параметр. Наряду с предложением ORDER BY вам необходимо указать название столбца (столбцов), на основе которого вам нужно отсортировать записи в вашем результирующем наборе.
* **LIMIT:**Это необязательный параметр. Это предложение помогает управлять максимальным количеством записей для извлечения и отображения в результирующем наборе. Количество строк, указанное параметром number\_rows, является максимальным количеством записей, которые возвращаются в результирующем наборе. offset\_value помогает определить первую строку, возвращаемую предложением
* **FETCH:**Это необязательный параметр, который определяет максимальное количество извлекаемых записей. Количество записей, указанное параметром fetch\_rows, будет возвращено в результирующем наборе. offset\_value - это значение, которое помогает определить первую строку, возвращаемую предложением FETCH.
* **FOR UPDATE:**Это необязательный параметр. Записи, затронутые запросом, блокируются на запись до завершения транзакции.
* **FOR SHARE:** это необязательный параметр. Записи, на которые повлиял запрос, могут быть использованы другими транзакциями, но они не могут быть обновлены или удалены.

SELECT \* FROM public.emp ORDER BY “id” ASC;

SELECT emp\_id from Employee;

SELECT emp\_id, first\_name, last\_name, address from Employee;

SELECT \* from Employee;

SELECT employee.emp\_name, department.dept\_name FROM employee INNER JOIN department ON employee.dept\_id = department.dept\_id ORDER BY emp\_name;

3. Create PostgreSQL Trigger Function

This function will be invoked before the insert, delete or update operation. It does the following:

* Before delete operation, it inserts the old data into backup\_tbl.
* Before update operation, it inserts the old data into backup\_tbl.
* Before insert operation, it inserts the new data into backup\_tbl.

CREATE FUNCTION ins\_function() RETURNS trigger AS '

BEGIN

IF tg\_op = ''DELETE'' THEN

INSERT INTO backup\_tbl(empid, empname, salary, operation)

VALUES (old.empid, old.empname, old.salary, tg\_op);

RETURN old;

END IF;

IF tg\_op = ''INSERT'' THEN

INSERT INTO backup\_tbl(empid, empname, salary, operation)

VALUES (new.empid, new.empname, new.salary, tg\_op);

RETURN new;

END IF;

IF tg\_op = ''UPDATE'' THEN

INSERT INTO backup\_tbl(empid, empname, salary, operation)

VALUES (old.empid, old.empname, old.salary, tg\_op);

RETURN new;

END IF;

END

' LANGUAGE plpgsql;

4. Create PostgreSQL Trigger

Using the following create trigger sql command, create a trigger which will invoke the function ‘ins\_function’ after insert, delete or update operation.

CREATE TRIGGER audit\_ins AFTER INSERT OR DELETE OR UPDATE

ON emp\_table FOR each ROW

EXECUTE PROCEDURE ins\_function();

Make sure to backup the postgreSQL database using [pg\_dump and psql](https://www.thegeekstuff.com/2009/01/how-to-backup-and-restore-postgres-database-using-pg_dump-and-psql/) command.

5. Test the PostgreSQL Trigger

Try inserting sample data into the emptable, which will automatically insert the data to the backup\_table as per the trigger logic.

# INSERT INTO emp\_table (empid, empname, salary) values (101, 'sathiya', '3000');

INSERT 0 1

# SELECT \* from backup\_tbl ;

empid | empname | salary | operation

-------+---------+--------+-----------

101 | sathiya | 3000 | INSERT

(1 row)

Try updating the data in emptable, which will automatically insert the old data to the backup\_tbl as per the trigger logic.

# UPDATE emp\_table SET salary = '2500' where empid = '101';

UPDATE 1

# SELECT \* from backup\_tbl ;

empid | empname | salary | operation

-------+---------+--------+-----------

101 | sathiya | 3000 | INSERT

101 | sathiya | 3000 | UPDATE

(2 rows)

Try deleting the data in emptable, which will automatically insert the old data to the backup\_tbl as per the trigger logic.

# DELETE FROM emp\_table WHERE empid = '101';

DELETE 1

# SELECT \* from backup\_tbl ;

empid | empname | salary | operation

-------+---------+--------+-----------

101 | sathiya | 3000 | INSERT

101 | sathiya | 3000 | UPDATE

101 | sathiya | 2500 | DELETE

Команда CREATE VIEW может использоваться с предложением WHERE.

**Пример:**

Код:

CREATE VIEW emp\_view

AS SELECT employee\_id, first\_name,last\_name, hire\_date

FROM employees

WHERE department\_id = 200;

Copy

Приведенный выше оператор PostgreSQL создаст представление emp\_view, содержащее записи (для столбцов employee\_id, first\_name, last\_name и hire\_date) таблицы сотрудников, если эти записи содержат значение 200 для столбца Department\_id.

PostgreSQL CREATE VIEW с помощью AND и OR

Команда CREATE VIEW может использоваться с операторами AND и OR.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT \*

FROM locations

WHERE (country\_id='US' AND city='Seattle')

OR (country\_id=JP' AND city='Tokyo');

Copy

Приведенный выше оператор PostgreSQL создаст представление 'my\_view', в котором будут записаны записи для всех столбцов таблицы местоположений, если (A) (i) значение столбца country\_id равно US, и (ii) значение города - Сиэтл; или (B) (i) значение столбца country\_id - JP, а (ii) значение города - Токио.

PostgreSQL СОЗДАТЬ ВИД с GROUP BY

Команда CREATE VIEW может использоваться с предложением GROUP BY.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT department\_id, count(\*)

FROM employees

GROUP BY department\_id;

Copy

Приведенный выше оператор создаст представление «my\_view», в котором будут собраны все записи, сгруппированные по департаменту\_id, а также сохранен идентификатор департамента и число сотрудников для каждого отдела (отдел\_идентификатора) из таблицы сотрудников.

PostgreSQL СОЗДАТЬ ВИД с ORDER BY

Команда CREATE VIEW может использоваться с предложением ORDER BY.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT department\_id,count(\*)

FROM employees

GROUP BY department\_id

ORDER BY department\_id;

Copy

Приведенный выше оператор PostgreSQL создаст представление «my\_view», в котором будут собраны все записи, сгруппированные по Department\_id и отсортированные по Department\_id и количеству сотрудников для каждого отдела (отдел\_ид) из таблицы сотрудников.

PostgreSQL СОЗДАТЬ ПРОСМОТР МЕЖДУ И В

Команда CREATE VIEW может использоваться с оператором BETWEEN и IN.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT \*

FROM employees

WHERE first\_name BETWEEN 'A' AND 'H'

AND salary IN(4000,7000,9000,10000,12000);

Copy

Приведенный выше оператор создаст представление «my\_view», в котором будут приниматься все таблицы записей о сотрудниках, если (A) first\_name сотрудника начинается с любого из символов от «A» до «H», а (B) - это любой из следующих зарплат 4000,7000,9000,10000,12000.

PostgreSQL СОЗДАТЬ ВИД с LIKE

Команда CREATE VIEW может использоваться с оператором LIKE.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT \*

FROM employees

WHERE first\_name

NOT LIKE 'T%' AND last\_name NOT LIKE 'T%';

Copy

Вышеприведенный оператор PostgreSQL создаст представление «my\_view», принимающее все записи таблицы сотрудников, если (A) first\_name сотрудника не начинается с «T» и (B) last\_name сотрудника не начинается с «T».

PostgreSQL CREATE VIEW с использованием подзапросов

Команда CREATE VIEW может использоваться с подзапросами.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT employee\_id,first\_name,last\_name

FROM employees

WHERE department\_id IN(

SELECT department\_id

FROM departments

WHERE location\_id IN (1500,1600,1700)

);

Copy

Приведенный выше оператор PostgreSQL создаст представление «my\_view», в котором будут храниться все записи таблицы employee\_id, first\_name, last\_name of employee, если Department\_id удовлетворяет условию, определенному в подзапросе (за которым следует Department\_id IN).

Подзапрос извлекает те идентификаторы департамента\_идейства из таблицы отделов, где указатель местоположения\_идентификатора - любой из списка 1500, 1600, 1700.

PostgreSQL СОЗДАТЬ ВИД с JOIN

Команда CREATE VIEW может использоваться вместе с оператором JOIN.

**Пример:**

Код:

CREATE VIEW my\_view

AS SELECT a.employee\_id,a.first\_name,a.last\_name, b.department\_name,

b.location\_id

FROM employees a,departments b

WHERE a.department\_id=b.department\_id;

Copy

Приведенный выше оператор PostgreSQL создаст представление my\_view вместе с оператором JOIN.

Здесь оператор JOIN извлекает employee\_id, first\_name, last\_name из таблицы сотрудников, а Department\_id и location\_id из таблицы расположений, если отдел\_ид таблицы сотрудников и местоположение находятся на одном уровне.

PostgreSQL СОЗДАТЬ ВИД с UNION

Команда CREATE VIEW может использоваться с UNION.

**Пример:**

Код:

CREATE VIEW my\_view AS

SELECT \*

FROM employees

WHERE manager\_id=100

UNION

SELECT \*

FROM employees

WHERE first\_name BETWEEN 'P' AND 'W'

UNION

SELECT \*

FROM employees

WHERE salary IN(7000,9000,10000,12000);

Copy

Приведенный выше оператор PostgreSQL создаст представление «my\_view», содержащее столбцы, как в «employee».

Переименовать представление abc в xyz

Код:

ALTER VIEW my\_view RENAME TO myview;

Copy

Чтобы прикрепить значение столбца по умолчанию к обновляемому представлению:

Код:

CREATE TABLE testtable (id int, tz timestamptz);

CREATE VIEW test\_view AS SELECT \* FROM testtable

ALTER VIEW test\_view ALTER COLUMN tz SET DEFAULT now();

Copy

Оставить представление

Оператор DROP VIEW используется для удаления представлений. Чтобы удалить представление, вы должны иметь привилегию DROP для каждого представления. Вот синтаксис:

DROP VIEW test\_view;

Основы использования оконных функций в PostgreSQL

3 октября 2022

[Оконные функции (window functions)](https://www.postgresql.org/docs/current/tutorial-window.html) — это фича, позволяющая производить манипуляции *между* строками, возвращаемыми одним SQL-запросом. Похоже на GROUP BY, но, в отличие от него, строки не объединяются в одну. Есть задачи, в которых оконные функции исключительно удобны. Например, когда нужно показать некие значения (выручку, посещаемость) за месяц, и рядом с ними — насколько это больше или меньше по сравнению с прошлым месяцем.

Для начала, выведем числа от 1 до 3:

SELECT x FROM generate\_series(1,3) AS x;  
  
 x  
===  
 1  
 2  
 3

Теперь перепишем запрос вот таким странным образом:

=# SELECT x, array\_agg(x) OVER ()  
   FROM generate\_series(1,3) AS x;  
  
 x | array\_agg  
==============  
 1 | {1,2,3}  
 2 | {1,2,3}  
 3 | {1,2,3}

Каждый раз, когда вы видите синтаксис OVER (...), можете быть уверены — запрос использует оконные функции. В данном примере используется простой OVER () без указания чего-либо в скобочках. Поэтому функция array\_agg() видит все строки, возвращаемые запросом. Эти строки называются *фреймом*, а array\_agg() выступает в роли *оконной функции*.

Запрос имеет доступ только к тому, что было извлечено по WHERE условию:

=# SELECT x, array\_agg(x) OVER ()  
   FROM generate\_series(1,3) AS x  
   WHERE x > 1;  
  
 x | array\_agg  
===============  
 2 | {2,3}  
 3 | {2,3}

Это правило действует всегда. Если нужно обратиться к чему-то, что не удовлетворяет WHERE-условию, необходимо использовать подзапросы.

Помимо array\_agg() можно использовать sum(), count(), и другие агрегаты:

=# SELECT x, count(x) OVER (), sum(x) OVER ()  
   FROM generate\_series(1,3) AS x;  
  
 x | count | sum  
=================  
 1 |     3 |   6  
 2 |     3 |   6  
 3 |     3 |   6

Кроме обычных агрегатов еще есть [специализированные оконные функции](https://www.postgresql.org/docs/current/functions-window.html). Некоторые из них будут рассмотрены далее.

OVER () с пустыми скобочками на самом деле эквивалентен:

=# SELECT x, array\_agg(x) OVER (  
     RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW  
   ) FROM generate\_series(1,3) AS x;  
  
 x | array\_agg  
===============  
 1 | {1,2,3}  
 2 | {1,2,3}  
 3 | {1,2,3}

Чтобы несколько раз не писать OVER (...), можно воспользоваться таким синтаксисом:

=# SELECT x, count(x) OVER w, sum(x) OVER w  
   FROM generate\_series(1,3) AS x  
   WINDOW w AS (RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW);  
  
 x | count | sum  
=================  
 1 |     3 |   6  
 2 |     3 |   6  
 3 |     3 |   6

В рассмотренных примерах фрейм всегда содержал все возвращаемые строки. Это можно исправить:

=# SELECT x, array\_agg(x) OVER w  
   FROM generate\_series(1,3) AS x  
   WINDOW w AS (  
     ORDER BY x  
     ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW  
   );  
  
 x | array\_agg  
===============  
 1 | {1}  
 2 | {1,2}  
 3 | {1,2,3}  
  
 =# 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 UNBOUNDED FOLLOWING  
    );  
  
 x | array\_agg  
===============  
 1 | {1,2,3}  
 2 | {2,3}  
 3 | {3}

Часть про ORDER BY x не является обязательной. Ведь нам известно, что generate\_series() возвращает числа в порядке возрастания. Но в общем случае, при SELECT‘е из таблицы, таких гарантий нет.

Внимательные читатели могли заметить, что вместо синтаксиса RANGE здесь мы перешли на ROWS. Это сделано не случайно. Дело в том, что вызов оконных функций может происходить в разных режимах. Они так и называются, RANGE mode и ROWS mode. Существует также GROUPS mode. RANGE mode немного запутанный. В частности, он переопределяет смысл CURRENT ROW в зависимости от того, какая часть остального синтаксиса используется в запросе. Сделано так, по всей видимости, чтобы используемый по умолчанию RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW работал по принципу наименьшего удивления, и эту часть запроса можно было опускать. Читателям, желающим основательно во всем этом разобраться, стоит [обратиться к официальной документации](https://www.postgresql.org/docs/current/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS). Здесь же мы тактично обойдем стороной этот вопрос.

Рассмотрим еще примеры:

=# 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  
   );  
  
 x | array\_agg  
===============  
 1 | {1}  
 2 | {2}  
 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  
    );  
  
 x | array\_agg  
===============  
 1 | {1}  
 2 | {1,2}  
 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  
    );  
  
 x | array\_agg  
===============  
 1 | {1,2}  
 2 | {2,3}  
 3 | {3}

Текущую строку не обязательно включать во фрейм:

=# SELECT x, sum(x) OVER w  
   FROM generate\_series(1,3) AS x  
   WINDOW w AS (  
     ORDER BY x  
     ROWS BETWEEN 1 PRECEDING AND CURRENT ROW EXCLUDE CURRENT ROW  
   );  
  
 x | sum  
=========  
 1 |  
 2 |   1  
 3 |   2

Смотрите, как удобно. Можно просто получить соседнее значение из выборки. Далее мы рассмотрим еще более простой способ это сделать.

Еще есть *партиции*. Рассмотрим их на более интересных данных:

=# 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 \* FROM employees;  
  
 department |  name  | salary  
==============================  
 dep\_a      | emp\_a1 |    971  
 dep\_a      | emp\_a2 |    972  
 dep\_a      | emp\_a3 |    973  
 dep\_b      | emp\_b1 |    981  
 dep\_b      | emp\_b2 |    982  
 dep\_b      | emp\_b3 |    983  
 dep\_c      | emp\_c1 |    991  
 dep\_c      | emp\_c2 |    992  
 dep\_c      | emp\_c3 |    993  
  
=# SELECT name, salary, array\_agg(salary) OVER w  
   FROM employees  
   WINDOW w AS (PARTITION BY department);  
  
  name  | salary |   array\_agg  
=================================  
 emp\_a1 |    971 | {971,972,973}  
 emp\_a2 |    972 | {971,972,973}  
 emp\_a3 |    973 | {971,972,973}  
 emp\_b1 |    981 | {981,982,983}  
 emp\_b2 |    982 | {981,982,983}  
 emp\_b3 |    983 | {981,982,983}  
 emp\_c1 |    991 | {991,992,993}  
 emp\_c2 |    992 | {991,992,993}  
 emp\_c3 |    993 | {991,992,993}

О партиции можно думать, как о такой специальной штуке, ограничивающей фрейм. Здесь мы партицируем данные по отделам. Ни один из фреймов не выходит за границы своей партиции. В остальном все работает так же, как раньше:

=# SELECT 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  
   );  
  
  name  | salary | sum  
=======================  
 emp\_a1 |    971 |  
 emp\_a2 |    972 | 971  
 emp\_a3 |    973 | 972  
 emp\_b1 |    981 |  
 emp\_b2 |    982 | 981  
 emp\_b3 |    983 | 982  
 emp\_c1 |    991 |  
 emp\_c2 |    992 | 991  
 emp\_c3 |    993 | 992

Партицировать можно и по условию:

=# 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  
     *-- попробуйте убрать следующую строчку; объясните результат*  
     ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING  
   );  
  
 INT4 | x | array\_agg  
======================  
    0 | 1 | {1,2,3}  
    0 | 2 | {1,2,3}  
    0 | 3 | {1,2,3}  
    1 | 4 | {4,5,6}  
    1 | 5 | {4,5,6}  
    1 | 6 | {4,5,6}

Выше говорилось о существовании [специализированных оконных функций](https://www.postgresql.org/docs/current/functions-window.html). Самая простая из них — это row\_number():

=# 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 |  name  | salary | sum  
====================================  
          1 | emp\_a1 |    971 |  
          2 | emp\_a2 |    972 | 971  
          3 | emp\_a3 |    973 | 972  
          1 | emp\_b1 |    981 |  
          2 | emp\_b2 |    982 | 981  
          3 | emp\_b3 |    983 | 982  
          1 | emp\_c1 |    991 |  
          2 | emp\_c2 |    992 | 991  
          3 | emp\_c3 |    993 | 992

Функция возвращает номер строки в партиции. Нумерация начиная с единицы.

Еще две полезные функции — это lag() и lead():

=# 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  | salary | lag | lead  
==============================  
 emp\_a1 |    971 |     |  972  
 emp\_a2 |    972 | 971 |  973  
 emp\_a3 |    973 | 972 |  
 emp\_b1 |    981 |     |  982  
 emp\_b2 |    982 | 981 |  983  
 emp\_b3 |    983 | 982 |  
 emp\_c1 |    991 |     |  992  
 emp\_c2 |    992 | 991 |  993  
 emp\_c3 |    993 | 992 |

Функции позволяют подсмотреть вперед или назад на заданное число строк в рамках партиции.

Также есть first\_value() и last\_value():

=# 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  
   );  
  
  name  | salary | first\_value | last\_value  
============================================  
 emp\_a1 |    971 |         971 |        973  
 emp\_a2 |    972 |         971 |        973  
 emp\_a3 |    973 |         971 |        973  
 emp\_b1 |    981 |         981 |        983  
 emp\_b2 |    982 |         981 |        983  
 emp\_b3 |    983 |         981 |        983  
 emp\_c1 |    991 |         991 |        993  
 emp\_c2 |    992 |         991 |        993  
 emp\_c3 |    993 |         991 |        993

Заметьте, что условие BETWEEN ... было переписано. Дело в том, что lag() и lead() работают на уровне *партиции*, и им не важно, какое условие было указано в BETWEEN .... Но функции first\_value() и last\_value() работают с *фреймом* и учитывают эти условия.

Еще существует nth\_value(). Функция также работает с фреймом:

=# 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  | salary | nth\_value  
=============================  
 emp\_a1 |    971 |       972  
 emp\_a2 |    972 |       972  
 emp\_a3 |    973 |       972  
 emp\_b1 |    981 |       982  
 emp\_b2 |    982 |       982  
 emp\_b3 |    983 |       982  
 emp\_c1 |    991 |       992  
 emp\_c2 |    992 |       992  
 emp\_c3 |    993 |       992

Довольно часто используется функция rank():

=# UPDATE employees SET salary = 970 WHERE salary < 980;  
  
=# SELECT name, salary, row\_number() OVER w, rank() OVER w  
   FROM employees  
   WINDOW w AS (PARTITION BY department ORDER BY salary DESC);  
  
  name  | salary | row\_number | rank  
=====================================  
 emp\_a1 |    970 |          1 |    1  
 emp\_a3 |    970 |          2 |    1  
 emp\_a2 |    970 |          3 |    1  
 emp\_b3 |    983 |          1 |    1  
 emp\_b2 |    982 |          2 |    2  
 emp\_b1 |    981 |          3 |    3  
 emp\_c3 |    993 |          1 |    1  
 emp\_c2 |    992 |          2 |    2  
 emp\_c1 |    991 |          3 |    3

Функция возвращает номер строки в соответствии с указанным порядком сортировки. Часть синтаксиса про ROWS BETWEEN ... здесь была опущена, поскольку она не влияла на результат.

Наконец, никто не говорил, что нельзя указать несколько OVER (...):

 =# 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  | salary | company\_rank | department\_rank  
==================================================  
 emp\_c3 |    993 |            1 |               1  
 emp\_c2 |    992 |            2 |               2  
 emp\_c1 |    991 |            3 |               3  
 emp\_b3 |    983 |            4 |               1  
 emp\_b2 |    982 |            5 |               2  
 emp\_b1 |    981 |            6 |               3  
 emp\_a1 |    970 |            7 |               1  
 emp\_a2 |    970 |            7 |               1  
 emp\_a3 |    970 |            7 |               1

Это далеко не все возможности оконных функций. Однако их должно хватать для очень многих практических задач.

<https://pro-prof.com/forums/topic/%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-postgresql>

drop table IF EXISTS emp;

drop table IF EXISTS dept;

drop table IF EXISTS salgrade;

drop table IF EXISTS bonus;

CREATE TABLE dept (

deptno NUMERIC CONSTRAINT PK\_DEPT PRIMARY KEY,

dname VARCHAR(14) ,

loc VARCHAR(13)

) ;

CREATE TABLE emp (

empno NUMERIC CONSTRAINT PK\_EMP PRIMARY KEY,

ename VARCHAR(10),

job VARCHAR(9),

mgr NUMERIC,

hiredate DATE,

sal NUMERIC(7,2),

comm NUMERIC(7,2),

deptno NUMERIC(2) CONSTRAINT FK\_DEPTNO REFERENCES DEPT

);

CREATE TABLE bonus (

enamE VARCHAR(10) ,

job VARCHAR(9) ,

sal NUMERIC,

comm NUMERIC

) ;

CREATE TABLE salgrade (

grade NUMERIC,

losal NUMERIC,

hisal NUMERIC

);

- Insert test data - dept

INSERT INTO dept VALUES (10,'ACCOUNTING','NEW YORK');

INSERT INTO dept VALUES (20,'RESEARCH','DALLAS');

INSERT INTO dept VALUES (30,'SALES','CHICAGO');

INSERT INTO dept VALUES (40,'OPERATIONS','BOSTON');

- Insert test data - emp

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,to\_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);

INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,to\_date('20-2-1981','dd-mm-yyyy'),1600,300,30);

INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,to\_date('22-2-1981','dd-mm-yyyy'),1250,500,30);

INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,to\_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);

INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,to\_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);

INSERT INTO emp VALUES (7698,'BLAKE','MANAGER',7839,to\_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);

INSERT INTO emp VALUES (7782,'CLARK','MANAGER',7839,to\_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);

INSERT INTO emp VALUES (7788,'SCOTT','ANALYST',7566,to\_date('13-07-87','dd-mm-yyyy')-85,3000,NULL,20);

INSERT INTO emp VALUES (7839,'KING','PRESIDENT',NULL,to\_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);

INSERT INTO emp VALUES (7844,'TURNER','SALESMAN',7698,to\_date('8-9-1981','dd-mm-yyyy'),1500,0,30);

INSERT INTO emp VALUES (7876,'ADAMS','CLERK',7788,to\_date('13-07-87','dd-mm-yyyy')-51,1100,NULL,20);

INSERT INTO emp VALUES (7900,'JAMES','CLERK',7698,to\_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);

INSERT INTO emp VALUES (7902,'FORD','ANALYST',7566,to\_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);

INSERT INTO emp VALUES (7934,'MILLER','CLERK',7782,to\_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);

- Insert test data - salgrade

INSERT INTO salgrade VALUES (1,700,1200);

INSERT INTO salgrade VALUES (2,1201,1400);

INSERT INTO salgrade VALUES (3,1401,2000);

INSERT INTO salgrade VALUES (4,2001,3000);

INSERT INTO salgrade VALUES (5,3001,9999);

- Transaction commit

COMMIT;

https://www.programmersought.com/article/19489663018/