

# Наиболее полезные конструкции PostgreSQL

## Содержание

<b>1</b>	<b>Логический порядок обработки инструкции SELECT</b>	<b>1</b>
<b>2</b>	<b>Смена схемы базы данных</b>	<b>2</b>
<b>3</b>	<b>Обновление записей</b>	<b>2</b>
<b>4</b>	<b>Общие табличные выражения</b>	<b>2</b>
4.1	Оператор WITH . . . . .	3
4.2	Оператор WITH RECURSIVE . . . . .	4
4.3	Изменение данных в WITH . . . . .	7
<b>5</b>	<b>Приемы работы в pgAdmin 4</b>	<b>7</b>
<b>6</b>	<b>Приемы работы в psql</b>	<b>7</b>
	<b>Список литературы</b>	<b>7</b>

## 1. Логический порядок обработки инструкции SELECT

Порядок обработки инструкции **SELECT** определяет, когда объекты, определенные в одном шаге, становятся доступными для предложений в последующих шагах. Например, если обработчик запросов можно привязать к таблицам или представлениям, определенным в предложении **FROM**, эти объекты и их столбцы становятся доступными для всех последующих шагов.

Общая процедура выполнения **SELECT** следующая (подробности см. в документации [SELECT](#)):

1. **WITH**: выполняются все запросы в списке **WITH**; по сути они формируют временные таблицы, к которым затем можно обращаться в списке **FROM**; запрос в **WITH** выполняется только один раз, даже если он фигурирует в списке **FROM** неоднократно,
2. **FROM**: вычисляются все элементы в списке **FROM** (каждый элемент в списке **FROM** представляет собой реальную или виртуальную таблицу); другими словами конструируются таблицы из списка **FROM**,
3. **ON**: выбираются строки, удовлетворяющие заданному условию,
4. **JOIN**: выполняется объединение таблиц,
5. **WHERE**: исключаются строки, не удовлетворяющие заданному условию,
6. **GROUP BY**: вывод разделяется по группам строк, соответствующим одному или нескольким значениям, а затем вычисляются результаты агрегатных функций,
7. **HAVING**: исключаются группы, не удовлетворяющие заданному условию,
8. **SELECT**,

9. **DISTINCT**: исключаются *повторяющиеся* строки; **SELECT DISTINCT ON** исключает строки, совпадающие по всем указанным выражениям; **SELECT ALL** (по умолчанию) возвращает все строки результата, включая дубликаты,
10. **UNION**, **INTERSECT** и **EXCEPT**: объединяется вывод нескольких команд **SELECT** в один результирующий набор.
11. **ORDER BY**: строки сортируются в указанном порядке; в отсутствие **ORDER BY** строки возвращаются в том порядке, в каком системе будет проще их выдавать,
12. **LIMIT** (или **FETCH FIRST**), либо **OFFSET**: возвращается только подмножество строк результата.
13. Если указано **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE** или **FOR KEY SHARE**, оператор **SELECT** блокирует выбранные строки, защищая их от одновременных изменений.

## 2. Смена схемы базы данных

Вывести список доступных схем

```
SHOW search_path;
```

Задать схему

```
SET search_path TO new_schema;
```

или, если требуется доступ к нескольким схемам

```
SET search_path TO new_schema1, new_schema2, public;
```

## 3. Обновление записей

Изменить слово Drama на Dramatic в столбце kind таблицы films

```
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

Изменить значение температуры и сбросить уровень осадков к значению по умолчанию в одной строке таблицы weather

```
UPDATE
  weather
SET
  temp_lo = temp_lo + 1,
  temp_hi = temp_lo + 15,
  prcp = DEFAULT
WHERE
  city = 'San Francisco' AND
  dates = '2003-07-03';
```

## 4. Общие табличные выражения

В конструкциях общих табличных выражений с **WITH** имена временных таблиц указываются без перечисления имен столбцов, а в конструкциях с **WITH RECURSIVE** – с перечислением, например, **WITH RECURSIVE** tab(col1, col2, ...) **AS** (...).

## 4.1. Оператор WITH

Основное предназначение SELECT в предложении WITH (Common Table Expression, Общие Табличные Выражения) заключается в разбиении сложных запросов на простые части. Например, пусть задана некоторая таблица `orders`<sup>1</sup>

```
WITH --part 1, common table expression
    regional_sales AS ( --def temp_table1
        SELECT region, sum(amount) AS total_sales
        FROM orders --base table
        GROUP BY region
    ),
    top_regions AS ( --def temp_table2
        SELECT region
        FROM regional_sales --temp_table1
        WHERE total_sales > (
            SELECT SUM(total_sales)/10
            FROM regional_sales --temp_table2
        )
    )
SELECT --part 2
    region,
    product,
    SUM(quantity) AS product_units,
    SUM(amount) AS product_sales
FROM orders
WHERE region IN (
    SELECT region
    FROM top_regions --temp_table2
)
GROUP BY region, product;
```

Здесь в инструкции WITH объявляются две *временные таблицы* `regional_sales` и `top_regions`. Вторая временная таблица `top_regions` ссылается на временную таблицу `regional_sales`, сформированную в первых строках настоящего запроса. Во второй части запроса также используется временная таблица `top_regions`.

Еще один пример. Пусть задана таблица

```
# SELECT * FROM test_tab;
id | cae_name      | solver  | num_cores
---+---+---+---
 1 | ANSYS         | Direct  |        32
 3 | Comsole       | Direct  |        16
 4 | LMS Virtual Lab | Direct  |        32
 2 | Nastran       | Iterativ |        16
(4 строки)
```

Требуется выяснить сколько CAE-пакетов имеют прямой, а сколько итерационный решатель. Эту задачу можно решить следующим образом

```
WITH sub_tab AS ( --make temp table
    SELECT solver, 1 AS count
    FROM test_tab
)
SELECT solver, sum(count)
FROM sub_tab --link to temp table
GROUP BY solver;
```

<sup>1</sup>См. документацию PostgreSQL <https://postgrespro.ru/docs/postgrespro/9.5/queries-with>

Часть с WITH возвращает

```
# SELECT solver, 1 AS count FROM test_tab;
solver | count
=====
Direct | 1
Direct | 1
Direct | 1
Iterativ | 1
(4 строки)
```

Полезный пример с использованием конструкции CASE...END и WHEN...THEN

```
WITH cte_film AS ( --part 1
    SELECT
        film_id,
        title,
        (CASE --start block
            WHEN length < 30 THEN 'Short'
            WHEN length < 90 THEN 'Medium'
            ELSE 'Long'
        END) length
    FROM
        film
)
SELECT --part 2
    film_id,
    title,
    length
FROM
    cte_film
WHERE
    length = 'Long'
ORDER BY
    title;
```

Пример с использованием логических операторов

```
WITH cte_films AS (
    SELECT
        film_id,
        title,
        (CASE
            WHEN length < 30 THEN 'Short'
            WHEN length >= 30 AND length < 90 THEN 'Medium'
            WHEN length > 90 THEN 'Long'
        END) length
    FROM
        film
)
```

## 4.2. Оператор WITH RECURSIVE

Если к WITH добавить RECURSIVE, то можно будет получить доступ к промежуточному результату. Например,

```
WITH RECURSIVE tbl(n) AS ( --part 1
    SELECT 1 --or VALUES(1). This is nonrecursive part
    UNION ALL
    SELECT n+1 FROM tbl WHERE n < 10 --and this is recursive part
```

```
)
SELECT sum(n) from tbl; --part 2
```

На первой итерации в таблице `tbl` в атрибуте `n` находится значение 1. На этом вычисления некурсивной части заканчиваются. Далее переходим к вычислениям в рекурсивной части. Таблица `tbl` ссылается на последнее вычисленное значение, поэтому на второй итерации удастся выполнить `n+1`, после чего новым значением таблицы `tbl` станет 2 (`tbl -> 2`). Проверяем условие `n < 10`, а затем переходим к следующей итерации и т.д.

Удобно представлять, что вычисленные значения хранятся в некоторой промежуточной области в порядке вычисления, а таблица `tbl` всегда ссылается на последнее вычисленное значение.

На последнем этапе 1 объединяется с 2, 3 и т.д., т.е. в итоге получается последовательность от 1 до 10. Во второй части запроса остается лишь просуммировать элементы этой последовательности и вывести на экран.

Рассмотрим еще такой пример

```
WITH RECURSIVE
  included_parts(sub_part, part, quantity) AS (
    SELECT --nonrecursive part
      sub_part,
      part,
      quantity
    FROM parts --base table
    WHERE part = "our_product"
    UNION ALL
    SELECT --recursive part
      p.sub_part,
      p.part,
      p.quantity
    FROM
      included_parts pr,
      parts p
    WHERE p.part = pr.sub_part
  )
SELECT sub_part, SUM(quantity) AS total_quantity
FROM included_parts
GROUP BY sub_part
```

На первой итерации временная таблица `included_parts`, вычисленная в некурсивной части, представляет собой результат выборки строк и столбцов из таблицы `parts`. В рекурсивной части можно получить доступ к этой таблице. В завершении выполняем выборку из таблицы `included_parts` по столбцу `sub_part`, группируем по нему и выводим сумму по `quantity`.

Еще один полезный пример. Пусть дана таблица сотрудников

employees				
id	name	salary	job	manager_id
1	John	10000	CEO	null
2	Ben	1400	Junior Developer	5
3	Barry	500	Intern	5
4	George	1800	Developer	5
5	James	3000	Manager	7
6	Steven	2400	DevOps Engineer	7
7	Alice	4200	VP	1
8	Jerry	3500	Manager	1
9	Adam	2000	Data Analyst	8

10	Grace	2500	Developer	8
11	Leor	5000	Data Scientist	6

Выведем иерархию подчинения сотрудников в компании

```
WITH RECURSIVE managers(id, name, manager_id, job, level) AS (
  SELECT id, name, manager_id, job, 1
  FROM employees --base table
  WHERE id = 7
  UNION ALL
  SELECT e.id, e.name, e.manager_id, e.job, m.level+1
  FROM employees e JOIN managers m ON e.manager_id = m.id
)
SELECT * FROM managers;
```

Сначала в *некурсивной* части WITH RECURSIVE объявляется временная таблица `managers(id, name, ...)`. Она строится на базе таблицы `employees`, к которой слева добавляется столбец, состоящий из одних единиц. Затем выбираются строки, удовлетворяющие условию `WHERE`; в данном случае это одна строка `e.manager_id=m.id`.

И, таким образом, на данном этапе во *временную* таблицу `managers` попадет только одна строка

managers, вычисленная в некурсивной части

id	name	manager_id	job	level
=====				
7	Alice	1	VP	1

Переходим в рекурсивную часть CTE. Из базовой таблицы `employees` выбираем те строки, которые в столбце `manager_id` имеют те же значения, что и в столбце `id` временной таблицы `managers` (на данном этапе таблица состоит из одной строки). Другими словами, выбрать нужно те строки, у которых в столбце `manager_id` таблицы `employees` стоит цифра 7.

В результате временная таблица `managers` на текущем этапе будет иметь вид

managers, вычисленная в рекурсивной части

id	name	manager_id	job	level
=====				
5	James	7	Manager	2
6	Steven	7	DevOps Engineer	2

Временные таблицы *рекурсивных общих табличных выражений* всегда ссылаются на результаты последних вычислений, т.е. на данном этапе временная таблица `managers` ссылается на таблицу, состоящую из двух строк.

Теперь мы снова выбираем из базовой таблицы `employees` и временной таблицы те строки, у которых в столбцах `e.manager_id` и `m.id` стоят одинаковые числа (в данном случае 5 и 6).

Таким образом

managers, вычисленная в рекурсивной части на 2-ой итерации

id	name	manager_id	job	level
=====				
2	Ben	5	Junior Developer	3
3	Barry	5	Intern	3
4	George	5	Developer	3
11	Leor	6	Data Scientist	3

Наконец все временные подтаблицы «склеиваются» и конструкция `SELECT * FROM managers` возвращает таблицу `managers`

id	name	manager_id	job	level
7	Alice	1	VP	1 --step 1
5	James	7	Manager	2 --step 2
6	Steven	7	DevOps Engineer	2 --step 2
2	Ben	5	Junior Developer	3 --step 3
3	Barry	5	Intern	3 --step 3
4	George	5	Developer	3 --step 3
11	Leor	6	Data Scientist	3 --step 3

### 4.3. Изменение данных в WITH

В предложении `WITH` можно также использовать операторы, изменяющие данные (`INSERT`, `UPDATE` или `DELETE`). Это позволяет выполнять в одном запросе сразу несколько разных операций. Например

```
WITH moved_rows AS (  
  DELETE FROM products  
  WHERE  
    dates >= '2010-10-01' AND  
    dates < '2010-11-01'  
  RETURNING *  
)  
INSERT INTO products_log (SELECT * FROM moved_rows);
```

Этот запрос фактически перемещает строки из таблицы `products` в таблицу `products_log` (таблица должна уже существовать на момент выполнения запроса). Оператор `DELETE` удаляет указанные строки из `products` и возвращает их содержимое в предложении `RETURNING`, а затем главный запрос читает это содержимое и вставляет в таблицу `products_log`.

## 5. Приемы работы в pgAdmin 4

## 6. Приемы работы в psql

### Список литературы

1. Чакон С., Штрауб Б. Git для профессионального программиста. – СПб.: Питер, 2020. – 496 с.
2. Соболев М. Linux. Администрирование и системное программирование. 2-е изд. – СПб.: Питер, 2011. – 880 с.