Дата последней сборки документа:
28 марта 2020 г. в 21:06

Наиболее полезные конструкции PostgreSQL

Содержание

1. 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;
```

2. 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';
```

3. 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 (...)`.

3.1. 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`[1]

```
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

[1] См. документацию `PostgreSQL` https://postgrespro.ru/docs/postgrespro/9.5/queries-with

```
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;
```

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
)
```

3.2. 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 удается 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.

3.3. 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. 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.

4. Приемы работы в pgAdmin 4

5. Приемы работы в psql

Список литературы

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