Дата последней сборки документа:
27 марта 2020 г. в 22:59

Наиболее полезные конструкции PostgreSQL

Содержание

1. Общие табличные выражения

Основное предназначение SELECT в предложении WITH заключается в разбиении сложных запросов на простые части. Например, пусть задана некоторая таблица orders[1]

```
WITH --part 1
    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
```

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

```
  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;
```

Часть с `WITH` возвращает

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

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 следующий 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`.

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.