Полезные трюки PostgreSQL / Хабрахабр

В мануале есть всё. Но чтобы его целиком прочитать и осознать, можно потратить годы. Поэтому один из самых эффективных методов обучения новым возможностям Postgres — это посмотреть, как делают коллеги. На конкретных примерах. Эта статья может быть интересна тем, кто хочет глубже использовать возможности postgres или рассматривает переход на эту СУБД.

Пример 1

Предположим, надо получить строки из таблицы, которых нет в другой точно такой же таблице, причем с проверкой всех полей на идентичность.
Традиционно можно было бы написать так (предположим, в таблице 3 поля):

```
SELECT t1.*
FROM table1 t1
    LEFT JOIN table2 t2
        ON t1.field1 = t2.field1
            AND t1.field2 = t2.field2
            AND t1.field3 = t2.field3
WHERE
    t2.field1 IS NULL;
```

Слишком многословно, на мой взгляд, и зависит от конкретных полей.
В посгресе же можно использовать тип Record. Получить его из таблицы можно используя само название таблицы.

```
postgres=# SELECT table1 FROM table1;
 table1
---------
 (1,2,3)
 (2,3,4)
```

(Выведет в скобочках)

Теперь, наконец отфильтруем строки с идентичными полями

```
SELECT table1.*
    FROM table1
        LEFT JOIN table2
            ON table1 = table2
WHERE
    table2 Is NULL;
```

или чуть более читабельно:

```
SELECT *
FROM table1
WHERE NOT EXISTS (
    SELECT *
    FROM table2
    WHERE
        table2 = table1
);
```

Пример 2

Очень жизненная задача. Приходит письмо “Вставь, пожалуйста, для юзеров 100, 110, 153, 100500 такие-то данные”.
Т.е. надо вставить несколько строк, где id разные, а остальное одинаковое.
Можно вручную составить такую “портянку”:

```
INSERT INTO important_user_table
(id, date_added, status_id)
VALUES
(100, '2015-01-01', 3),
(110, '2015-01-01', 3),
(153, '2015-01-01', 3),
(100500, '2015-01-01', 3);
```

Если id много, то это слегка напрягает. Кроме того, у меня аллергия на дублирование кода.

Для решения подобных проблем в посгресе есть тип данных “массив”, а также функция unnest, которая из массива делает строки с данными.

Например

```
postgres=# select unnest(array[1,2,3]) as id;
 id
----
  1
  2
  3
(3 rows)
```

Т.е. в нашем примере мы можем написать так

```
INSERT INTO important_user_table
(id, date_added, status_id)
SELECT
     unnest(array[100, 110, 153, 100500]), '2015-01-01', 3;
```

т.е. список id просто копипастим из письма. Очень удобно.

Кстати, если же вам наоборот нужен массив из запроса, то для этого есть функция, которая так и называется — array(). Например, select array(select id from important_user_table);

Пример 3

Для схожих целей можно использовать еще один трюк. Мало кто знает, что синтаксис

```
VALUES (1, 'one'), (2, 'two'), (3, 'three')
```

можно использовать не только в INSERT запросах, но и в SELECT, надо только в скобочки взять

```
SELECT * FROM (
    VALUES (1, 'one'), (2, 'two'), (3, 'three')
) as t (digit_number, string_number);
 digit_number | string_number
--------------+---------------
            1 | one
            2 | two
            3 | three
(3 rows)
```

Очень удобно для обработки пар значений.

Пример 4

Допустим, вам нужно что-то вставить, проапдейтить, и получить id затронутых элементов. Чтобы сделать это, не обязательно делать много запросов и создавать временные таблицы. Достаточно всё это запихать в CTE.

```
WITH
updated AS (
       UPDATE table1
 SET x = 5, y = 6
 WHERE z > 7
        RETURNING id
),
inserted AS (
       INSERT INTO table2
        (x, y, z)
        VALUES
        (5, 7, 10)
        RETURNING id
)
SELECT id
FROM updated
UNION
SELECT id
FROM inserted;
```

Но будьте очень внимательны. Все подвыражения CTE выполняеются параллельно друг с другом, и их последовательность никак не определена. Более того, они используют одну и ту же версию (snapshot), т.е. если в одном подвыражении вы прибавили что-то к полю таблицы, в другом вычли, то возможно, что сработает что-то одно из них.

Пример 5

Допустим в какой-то таблице под названием stats есть данные только за один день:

```
postgres=# select * from stats;
  added_at  | money
------------+--------
 2016-04-04 | 100.00
(1 row)
```

А вам надо вывести стату за какой-то период, заменив отсутствующие данные нулями. Это можно сделать с помощью generate_series

```
SELECT gs.added_at, coalesce(stats.money, 0.00) as money
FROM
    generate_series('2016-04-01'::date, '2016-04-07'::date , interval '1 day') as
gs(added_at)
    LEFT JOIN stats
        ON stats.added_at = gs.added_at;

        added_at        | money
------------------------+--------
 2016-04-01 00:00:00+03 |   0.00
 2016-04-02 00:00:00+03 |   0.00
 2016-04-03 00:00:00+03 |   0.00
 2016-04-04 00:00:00+03 | 100.00
 2016-04-05 00:00:00+03 |   0.00
 2016-04-06 00:00:00+03 |   0.00
 2016-04-07 00:00:00+03 |   0.00
(7 rows)
```

Разумеется, этот трюк работает не только с датами, но и с числами. Причем можно использовать несколько generate_series в одном запросе:

```
teasernet_maindb=> select  generate_series (1,10), generate_series(1,2);
 generate_series | generate_series
-----------------+-----------------
               1 |               1
               2 |               2
               3 |               1
               4 |               2
               5 |               1
               6 |               2
               7 |               1
               8 |               2
               9 |               1
              10 |               2
(10 rows)
```

Пример n+1

Вообще, я пишу статьи на хабр, чтобы получить немного нового опыта из коментов) Пожалуйста, напишите, что вы используете в повседневной работе. Что-нибудь такое, что возможно не для всех очевидно, особенно для людей, переехавших с других СУБД, например, с того же mysql?