Работа с PostgreSQL ч. 1

Занятие 1.5

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Алексей Кузьмин

Директор разработки,
Data Scientist ДомКлик.ру

Что сегодня изучим

1. Оконные и аналитические функции
2. CTE
3. Рекурсивные CTE

Оконные и аналитические функции

1

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Оконные и аналитические функции

Оконные функции — полезный инструмент
для построения сложных аналитических запросов.

Для их использования нужно задать параметры
окна и функцию, которую хотим посчитать
на каждом объекте внутри окна.

Пример: функция ROW_NUMBER(), которая нумерует
строки внутри окна. Пронумеруем аренду для каждого
пользователя в порядке убывания даты аренды.

```
SELECT
    customer_id, rental_id, rental_date,
    ROW_NUMBER() OVER (PARTITION BY customer_id
    ORDER BY rental_date DESC) as rental_rank
FROM (
    SELECT customer_id, rental_id, rental_date
    FROM rental
    WHERE staff_id = 1 LIMIT 500
) as sample
ORDER BY
    customer_id,
    rental_date DESC,
    rental_rank
LIMIT 20;
```

Результат

customer_id	rental_id	rental_date	rental_rank
1	573	2005-05-28 10:35:23	1
3	830	2005-05-29 22:43:55	1
5	731	2005-05-29 07:25:16	1
6	916	2005-05-30 11:25:01	1
7	748	2005-05-29 09:27:00	1
8	866	2005-05-30 03:43:54	1
9	877	2005-05-30 05:48:59	1
11	987	2005-05-30 22:59:12	1
12	988	2005-05-30 23:08:03	1
14	815	2005-05-29 20:24:28	1
14	525	2005-05-28 04:25:33	2
14	151	2005-05-26 00:37:28	3
16	1017	2005-05-31 02:53:36	1
17	884	2005-05-30 06:41:32	1
17	287	2005-05-26 19:44:54	2
18	116	2005-05-25 19:27:51	1
19	696	2005-05-29 01:59:10	1
19	337	2005-05-27 03:22:30	2
19	179	2005-05-26 04:26:06	3
20	546	2005-05-28 07:16:25	1

Параметры запроса:

ROW_NUMBER — функция, которую применяем к окну.

OVER — описание окна.

Описание окна содержит:

- PARTITION BY — поле или список полей, которые описывают группу строк для применения оконной функции

- ORDER BY — поле, которое задаёт порядок записей внутри окна. Для полей внутри ORDER BY можно применять стандартные модификаторы DESC, ASC

Параметры запроса:

Оконная функция никак не меняет количество строк в выдаче, но к каждой строке добавляется полезная информация. Например, про порядковый номер строки внутри окна.

Названия функций обычно отражают их смысл. Далее будут приведены примеры использования и результаты запросов.

SUM()

Суммирует значения внутри окна.

Посчитаем странную метрику — разделим каждый платёж пользователя на сумму его первых трёх платежей

```
SELECT customer_id, payment_id, amount,
   amount / SUM(amount) OVER (PARTITION BY customer_id) as strange_rating_metric
FROM (SELECT *
        FROM (SELECT customer_id, payment_id, amount,
                ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY payment_date)
                FROM payment p) t
        WHERE row_number < 4) AS sample
ORDER BY customer_id, amount DESC
LIMIT 20;
```

Обратите внимание: так как оконная функция работает с результатом запроса, то написать напрямую условие к результату оконной функции нельзя, необходимо использовать подзапрос.

В подзапросе t получаем список платежей и для каждого пользователя проставляем порядковый номер в порядке даты платежей. В подзапросе sample используем условие, согласно которому фильтруем записи, где порядковый номер менее 4

```
FROM (SELECT *
      FROM (SELECT customer_id, payment_id, amount,
            ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY payment_date)
            FROM payment p) t
      WHERE row_number < 4) AS sample
```

Результат

customer_id	payment_id	amount	strange_rating_metric
1	18497	9.99	0.58868591632292280495
1	18495	5.99	0.35297583971714790807
1	18496	0.99	0.05833824395992928698
2	18502	2.99	0.60160965794768611670
2	22691	0.99	0.19919517102615694165
2	22692	0.99	0.19919517102615694165
3	18503	8.99	0.39138006094906399652
3	18504	6.99	0.30430996952546800174
3	18505	6.99	0.30430996952546800174
4	18507	4.99	0.55629877369007803790
4	18509	2.99	0.33333333333333333333
4	18508	0.99	0.11036789297658862876
5	18515	4.99	0.41687552213868003342
5	18513	3.99	0.33333333333333333333
5	18514	2.99	0.24979114452798663325
6	18518	3.99	0.50062735257214554580
6	18519	2.99	0.37515683814303638645
6	18520	0.99	0.12421580928481806775
7	18523	2.99	0.60160965794768611670
7	18522	0.99	0.19919517102615694165

COUNT(), AVG()

Счётчик элементов внутри окна, а также функция Average().
Используем их одновременно — результаты не должны отличаться.
Вычислим полезную метрику — отклонение платежа пользователя
от среднего значения по первым трём платежам.

```
SELECT customer_id, payment_id, amount,
    amount - AVG(amount) OVER (PARTITION BY customer_id) payment_deviance_simplex,
 amount - SUM(amount) OVER (PARTITION BY customer_id) / COUNT(amount)
OVER (PARTITION BY customer_id) as payment_deviance_complex
FROM (SELECT *
        FROM (SELECT customer_id, payment_id, amount,
                ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY payment_date)
                FROM payment p) t
        WHERE row_number < 4) sample
ORDER BY customer_id, amount DESC
LIMIT 20;
```

Результат

customer_id	payment_id	amount	payment_deviance_simplex	payment_deviance_complex
1	18497	9.99	4.3333333333333333	4.3333333333333333
1	18495	5.99	0.3333333333333333	0.3333333333333333
1	18496	0.99	-4.6666666666666667	-4.6666666666666667
2	18502	2.99	1.3333333333333333	1.3333333333333333
2	22691	0.99	-0.6666666666666667	-0.6666666666666667
2	22692	0.99	-0.6666666666666667	-0.6666666666666667
3	18503	8.99	1.3333333333333333	1.3333333333333333
3	18504	6.99	-0.6666666666666667	-0.6666666666666667
3	18505	6.99	-0.6666666666666667	-0.6666666666666667
4	18507	4.99	2.000000000000000	2.000000000000000
4	18509	2.99	0.000000000000000	0.000000000000000
4	18508	0.99	-2.000000000000000	-2.000000000000000
5	18515	4.99	1.000000000000000	1.000000000000000
5	18513	3.99	0.000000000000000	0.000000000000000
5	18514	2.99	-1.000000000000000	-1.000000000000000
6	18518	3.99	1.3333333333333333	1.3333333333333333
6	18519	2.99	0.3333333333333333	0.3333333333333333
6	18520	0.99	-1.6666666666666667	-1.6666666666666667
7	18523	2.99	1.3333333333333333	1.3333333333333333
7	18522	0.99	-0.6666666666666667	-0.6666666666666667

Ещё функции:

Для нахождения максимального или минимального значения в рамках оконной функции используются функции MIN() или MAX().

Для работы с рангами используются функции:

- RANK() — ранг текущей строки с пропусками; то же, что и ROW_NUMBER для первой родственной ей строки
- DENSE_RANK() — ранг текущей строки без пропусков; эта функция считает группы родственных строк

Ещё функции:

Для работы с предыдущими или последующими значениями
из выборки используются функции LAG(значение, шаг)
или LEAD(значение, шаг) соответственно.

Давайте сравним суммы платежей за предыдущий,
текущий и следующий месяц

```
SELECT DATE_TRUNC('month', payment_date),
       LAG(SUM(amount), 1) OVER () AS "Предыдущее значение",
       SUM(amount) AS "Текущее значение",
       LEAD(SUM(amount), 1) OVER () AS "Следующее значение"
FROM payment
GROUP BY DATE_TRUNC('month', payment_date)
```

Результат

date_trunc	Предыдущее значение	Текущее значение	Следующее значение
2007-02-01 00:00:00		8341.86	23886.56
2007-03-01 00:00:00	8341.86	23886.56	28559.46
2007-04-01 00:00:00	23886.56	28559.46	514.18
2007-05-01 00:00:00	28559.46	514.18	7.99
2008-02-01 00:00:00	514.18	7.99	1.99
2009-02-01 00:00:00	7.99	1.99	

Время практики

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Практика 1

- Выведите таблицу с тремя полями: название фильма, имя актера и количество фильмов, в которых он снимался

Практика 1. Решение

```
SELECT f.title, a.last_name, COUNT(f.film_id) OVER (PARTITION BY a.actor_id)
FROM film f
JOIN film_actor fa USING (film_id)
JOIN actor a USING (actor_id)
```

PostgreSQL CTE

2

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

CTE

CTE — это временный результат запроса, который можно использовать с другими запросами.

Временный = существует только в рамках запроса

Синтаксис:

```
WITH cte_name (column_list) AS (
    CTE_query_definition
)
statement;
```

CTE

- Указывается название CTE
- Опционально список имён колонок
- Запрос CTE
- Основной SQL-запрос

Обычно используются для упрощения сложных join-запросов и подзапросов. Кроме того, поддерживают рекурсивные запросы

```sql
WITH cte_film 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
)
SELECT film_id, title, length
FROM
    cte_film
WHERE
    length = 'Long'
ORDER BY title;
```

Рекурсивные CTE

3

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

ОТВ и рекурсивные запросы

Вычисление чего-то итерациями до того,
как будет выполнено некоторое условие.

```
WITH имя_ОТВ (список__столбцов) AS
(
стартовый__запрос
union [all]
рекурсивный__запрос__к__имя_ОТВ
)
внешний_запрос
```

Задача

Посчитать факториал:

n! = 1234 ... (n-1)n

[ ]

```
WITH RECURSIVE r AS (
  -- стартовая часть рекурсии (т.н. "anchor")
  SELECT
    1 AS i,
    1 AS factorial

  UNION
```

Задача

-- рекурсивная часть

```
 SELECT
       i+1 AS i,
       factorial * (i+1) as factorial
    FROM r
    WHERE i < 10
)
SELECT * FROM r;
```

Алгоритм примерно такой:

1. извлечь стартовые данные

2. подставить полученные данные с предыдущей итерации в рекурсивную часть запроса

3. если в текущей итерации рекурсивной части не пустая строка, то добавляем её в результирующую выборку. Также нужно пометить данные для следующего вызова рекурсивной части (п.2). В противном случае необходимо завершить обработку

Время практики

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Практика 2

- При помощи CTE выведите таблицу со следующим содержанием: фамилия, имя сотрудника (staff) и количество прокатов DVD (rental), которые он реализовал

Практика 2. Решение

```
WITH cte AS (
        SELECT s.last_name, r.rental_id, s.staff_id
        FROM staff s
        JOIN rental r ON r.staff_id = s.staff_id)
SELECT last_name, COUNT(rental_id)
FROM cte
GROUP BY staff_id, last_name
```

Вопросы

Полезные материалы

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру

Полезные материалы

- https://habr.com/ru/post/269497/
- https://medium.com/@hakibenita/be-careful-with-cte-in-postgresql-fca5e24d2119
  *актуально для PostgreSQL ниже 12 версии
- https://postgrespro.ru/docs/postgrespro/9.5/using-explain
- https://habr.com/ru/post/203320/

Спасибо за внимание!

Алексей Кузьмин
Директор разработки,
Data Scientist ДомКлик.ру