**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;

**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

**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;

**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)

**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

**OTB 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

**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