IT CAREER HUB

Урок 9.2. Агрегирующие оконные функции

Основные агрегирующие оконные функции 2
Примеры функций 3
Задание для закрепления 6
Подсчет кумулятивных значений 8

Основные агрегирующие оконные функции

 **Важно!**
База данных с доступом на чтение:
hostname: ich-db.edu.itcareerhub.de
username: ich1
password: password

Агрегирующие оконные функции — это функции, позволяющие выполнять вычисления, которые агрегируют данные в рамках определенного окна (группы строк), сохраняя при этом детализированные строки.

Они полезны для выполнения сложных аналитических задач, таких как вычисление кумулятивных сумм, ранжирование, нахождение средних значений и других статистических операций.

В рамках одного и того же окна агрегирующая функция будет вычислять одно значение если в OVER не указан порядок сортировки, то есть присутствует только OVER (PARTITION BY).

Основные агрегирующие оконные функции:

1. **SUM()**
2. **AVG()**
3. **MIN()**
4. **MAX()**
5. **COUNT()**

IT CAREER HUB

Примеры функций

1. **Задача: Для каждого клиента посчитать общую сумму всех его заказов.**

```
Unset

SELECT OrderID, CustomerID, OrderAmount,

       SUM(OrderAmount) OVER (PARTITION BY CustomerID) AS TotalCustomerOrders

FROM Orders;
```

OrderID	CustomerID	OrderAmount
1	101	300
2	101	500
3	102	150
4	101	700
5	102	200

OrderID	CustomerID	OrderAmount	TotalCustomerOrders
1	101	300	1500
2	101	500	1500
4	101	700	1500
3	102	150	350
5	102	200	350

SUM(OrderAmount) OVER (PARTITION BY CustomerID) вычисляет сумму заказов для каждого клиента, независимо от порядка строк. PARTITION BY CustomerID разделяет строки на группы по значению CustomerID. В результате каждая строка в группе получает одно и то же значение общей суммы заказов этого клиента.
В данном случае, все строки для клиента с CustomerID = 101 получат значение 1500, а для клиента с CustomerID = 102 — 350.

2. **Задача: Подсчитать количество заказов для каждого клиента.**

```
Unset
SELECT OrderID, CustomerID, OrderAmount,
       COUNT(*) OVER (PARTITION BY CustomerID) AS OrdersPerCustomer
FROM Orders;
```

OrderID	CustomerID	OrderAmount	OrdersPerCustomer
1	101	300	3
2	101	500	3
4	101	700	3
3	102	150	2
5	102	200	2

COUNT(*) OVER (PARTITION BY CustomerID) подсчитывает количество заказов для каждого клиента. Все строки для клиента с CustomerID = 101 получат значение 3, а для клиента с CustomerID = 102 — 2. Этот результат получается без учета порядка строк, так как не используется ORDER BY.

IT CAREER HUB

Задание для закрепления

1. Из таблицы products выведите максимальный list_price для каждой строки, имя продукта и его list_price.

```
Unset
SELECT product_name, list_price, MAX(list_price) OVER ()

FROM products
```

2. Используя предыдущий запрос, посчитайте разницу в процентах между ценой продукта и максимальной ценой.

```
Unset

SELECT product_name, (MAX(list_price) OVER () - list_price)/(MAX(list_price)
OVER ())*100

FROM products;
```

3. Посчитайте количество продуктов в каждой категории с помощью оконной функции. Оптимально ли использование оконной функции для выполнения этого задания.

```
Unset

SELECT category, COUNT(id) OVER (partition by category)

FROM products
```

4. Найдите разницу между standard_cost продукта и средним list_price по всей таблицы для каждой строки.

```
Unset

SELECT product_name, standard_cost - AVG(list_price) OVER ()

FROM products
```

5. Можно ли решить предыдущее задание без оконных функций.

Unset

```
with avg_price as (SELECT AVG(list_price) as ap FROM products)

SELECT product_name,   standard_cost - avg_price.ap

FROM products JOIN avg_price ON 1=1
```

IT CAREER HUB

Подсчет кумулятивных значений

Кумулятивная сумма — это сумма значений от начала до текущей строки.

Например, если у вас есть список продаж, кумулятивная сумма покажет, как общая сумма продаж накапливается с каждым новым заказом.

- При подсчете кумулятивных значений с помощью оконных функций в оператор OVER дополнительно вписывается порядок сортировки ORDER BY.
- Кумулятивные значения рассчитываются только при использовании столбца с датами в ORDER BY.

1. **Задача: Рассчитать кумулятивную сумму продаж по датам.**

```
Unset
SELECT SaleID, SaleDate, SaleAmount,
       SUM(SaleAmount) OVER (ORDER BY SaleDate) AS CumulativeSales
FROM Sales;
```

SaleID	SaleDate	SaleAmount
1	2024-01-01	100
2	2024-01-02	150
3	2024-01-03	200

SaleID	SaleDate	SaleAmount	CumulativeSales
1	2024-01-01	100	100
2	2024-01-02	150	250
3	2024-01-03	200	450

Здесь функция SUM(SaleAmount) OVER (ORDER BY SaleDate) вычисляет кумулятивную сумму (CumulativeSales) для каждой строки, упорядоченную по дате продажи. Кумулятивная сумма увеличивается по мере добавления каждой новой строки.

2. **Задача: Рассчитать текущее среднее значение суммы заказов для каждого клиента**.

```
Unset
SELECT OrderID, CustomerID, OrderDate, OrderAmount,
       AVG(OrderAmount) OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS
RunningAvg
FROM Orders;
```

OrderID	CustomerID	OrderDate	OrderAmount
1	101	2024-01-01	300
2	101	2024-01-05	500
3	102	2024-01-02	150
4	101	2024-01-10	700
5	102	2024-01-07	200

OrderID	CustomerID	OrderDate	OrderAmount	RunningAvg
1	101	2024-01-01	300	300
2	101	2024-01-05	500	400

4	101	2024-01-10	700	500
3	102	2024-01-02	150	150
5	102	2024-01-07	200	175

Функция AVG(OrderAmount) OVER (PARTITION BY CustomerID ORDER BY OrderDate) вычисляет текущее среднее значение (RunningAvg) для каждого клиента. Она используется для определения средней суммы заказов по мере поступления новых заказов.