Лекция №5

SQL – оконные функции

Сегодня мы продолжим изучение языка SQL, начав с изучения оконных функций.

Сегодня мы узнаем:

1. Групповые функции без сортировки
2. Использование сортировки в оконной функции
3. Функции ранжирования, функции смещения, агрегатные и аналитические
4. порядок выполнения запроса — полная схема
5. представления (view) – использование, типы

Термины лекции

Оконная функция в SQL - *функция, которая работает с выделенным набором строк (окном, партицией) и выполняет вычисление для этого набора строк в отдельном столбце.*

Партиции (окна из набора строк) - *это набор строк, указанный для оконной функции по одному из столбцов или группе столбцов таблицы. Партиции для каждой оконной функции в запросе могут быть разделены по различным колонкам таблицы.*

Агрегатные функции (агрегации) — *это функции, которые вычисляются от группы значений и объединяют их в одно результирующее.*

Функции смещения *– это функции, которые позволяют перемещаться и обращаться к разным строкам в окне, относительно текущей строки, а также обращаться к значениям в начале или в конце окна.*

Аналитические функции *— это функции которые возвращают информацию о распределении данных и используются для статистического анализа.*

Предикат в SQL это:

в широком смысле, — любое выражение, результатом которого являются значения булевого типа — TRUE, FALSE, а так же UNKNOWN

в узком смысле, — некий уточняющий фильтр. Самым явным примером предиката служит оператор WHERE

Оглавление

Оконные функции — это мощнейший инструмент аналитика, который с легкостью помогает решать множество задач.

Если вам нужно произвести вычисление над заданным набором строк, объединенных каким-то одним признаком, например идентификатором клиента, вам на помощь придут именно они.

Можно сравнить их с агрегатными функциями, но, в отличие от обычной агрегатной функции, при использовании оконной функции несколько строк не группируются в одну, а продолжают существовать отдельно. При этом результаты работы оконных функций просто добавляются к результирующей выборке как еще одно поле. Этот функционал очень полезен для построения аналитических отчетов, расчета скользящего среднего и нарастающих итогов, а также для расчетов различных моделей атрибуции.

Оконные функции SQ

Оконные функции позволяют решать задачи запросов новыми, более простыми способами и с большей производительностью.Предположим, что у нас есть sales таблица, в которой хранятся продажи по сотрудникам и финансовым годам: (слайд на презентации)

	sales_employee	fiscal_year	sale
▶	Alice	2016	150.00
	Alice	2017	100.00
	Alice	2018	200.00
	Bob	2016	100.00
	Bob	2017	150.00
	Bob	2018	200.00
	John	2016	200.00
	John	2017	150.00
	John	2018	250.00

.

Чтобы понять оконные функции, начнем разбор с агрегатных функций, пройденных нами ранее. Агрегатные функции суммируют данные из нескольких строк в одну результирующую строку. Например, следующая SUM()функция возвращает общий объем продаж всех сотрудников за зарегистрированные годы:

```
1 SELECT SUM(sale)
2 FROM sales;
```

Предложение GROUP BY позволяет применять агрегатные функции к подмножеству строк. Например, вы можете рассчитать общий объем продаж по финансовым годам:

	fiscal_year	SUM(sale)
▶	2016	450.00
	2017	400.00
	2018	650.00

Как и агрегатные функции с GROUP BY предложением, оконные функции также работают с подмножеством строк, но они не уменьшают количество строк, возвращаемых запросом. Например, следующий запрос возвращает продажи для каждого сотрудника, а также общий объем продаж сотрудников по финансовому году:

```
1 SELECT
2     fiscal_year,
3     SUM(sale)
4 FROM
5     sales
6 GROUP BY
7     fiscal_year;
```

Принцип работы

У вас может возникнуть вопрос – «Что значит оконные? При обычном запросе, все множество строк обрабатывается как бы единым «цельным куском», для которого считаются агрегаты. А

при использовании оконных функций, запрос делится на части (окна) и уже для каждой из отдельных частей считаются свои агрегаты.

Синтаксис

Окно определяется с помощью обязательной инструкции OVER(). Давайте рассмотрим синтаксис этой инструкции:

```
SELECT
Название функции (столбец для вычислений)
OVER
(
      PARTITION BY столбец для группировки
      ORDER BY столбец для сортировки
      ROWS или RANGE выражение для ограничения строк в пределах группы
)
```

Теперь разберем как поведет себя множество строк при использовании того или иного ключевого слова функции. А тренироваться будем на простой табличке содержащей дату, канал с которого пришел пользователь и количество конверсий: (на слайде).

OVER()

Откроем окно при помощи OVER() и просуммируем столбец «Conversions». Мы использовали инструкцию OVER() без предложений. В таком варианте окном будет весь набор данных и никакая сортировка не применяется. Появился новый столбец «Sum» и для каждой строки выводится одно и то же значение 14. Это сквозная сумма всех значений колонки «Conversions».

PARTITION BY

Теперь применим инструкцию PARTITION BY, которая определяет столбец, по которому будет производиться группировка и является ключевой в разделении набора строк на окна:

```
SELECT
  Date,
  Medium,
  Conversions,
  SUM(Conversions) OVER(PARTITION BY Date) AS 'Sum'
FROM Order;
```

Инструкция PARTITION BY сгруппировала строки по полю «Date». Теперь для каждой группы рассчитывается своя сумма значений столбца «Conversions».

ORDER BY

Попробуем отсортировать значения внутри окна при помощи ORDER BY.

К предложению PARTITION BY добавилось ORDER BY по полю «Medium». Таким образом мы указали, что хотим видеть сумму не всех значений в окне, а для каждого значения «Conversions» сумму со всеми предыдущими. То есть мы посчитали нарастающий итог.

Партиции (окна из набора строк) - это набор строк, указанный для оконной функции по одному из столбцов или группе столбцов таблицы. Партиции для каждой оконной функции в запросе могут быть разделены по различным колонкам таблицы.

ROVER или RANGE

Инструкция ROWS позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей. Инструкция RANGE, в отличие от ROWS, работает не со строками, а с диапазоном строк в инструкции ORDER BY. То есть под одной строкой для RANGE могут пониматься несколько физических строк одинаковых по рангу. Обе инструкции ROWS и RANGE всегда используются вместе с ORDER BY. В выражении для ограничения строк ROWS или RANGE также можно использовать следующие ключевые слова:

UNBOUNDED PRECEDING — указывает, что окно начинается с первой строки группы;

UNBOUNDED FOLLOWING – с помощью данной инструкции можно указать, что окно заканчивается на последней строке группы;

CURRENT ROW – инструкция указывает, что окно начинается или заканчивается на текущей строке;

BETWEEN «граница окна» AND «граница окна» — указывает нижнюю и верхнюю границу окна;

«Значение» PRECEDING – определяет число строк перед текущей строкой (не допускается в предложении RANGE).;

«Значение» FOLLOWING — определяет число строк после текущей строки (не допускается в предложении RANGE).

Разберемся на примере:

```sql
SELECT
  Date,
  Medium,
  Conversions,
  SUM(Conversions) OVER(PARTITION BY Date ORDER BY Conversions ROWS BETWEEN
 CURRENT ROW AND 1 FOLLOWING) AS 'Sum'
FROM Orders;
```

В данном случае сумма рассчитывается по текущей и следующей ячейке в окне. А последняя строка в окне имеет то же значение, что и столбец «Conversions», потому что больше не с чем складывать. Комбинируя ключевые слова, вы можете подогнать диапазон работы оконной функции под вашу специфическую задачу.

Виды функций

Оконные функции можно подразделить на следующие группы:

1. Агрегатные функции;
2. Ранжирующие функции;
3. Функции смещения;
4. Аналитические функции.

В одной инструкции SELECT с одним предложением FROM можно использовать сразу несколько оконных функций. Давайте подробно разберем каждую группу и пройдемся по основным функциям.

Агрегатные функции

Агрегатные функции – это функции, которые выполняют на наборе данных арифметические вычисления и возвращают итоговое значение.

SUM – возвращает сумму значений в столбце;

COUNT — вычисляет количество значений в столбце (значения NULL не учитываются);

AVG — определяет среднее значение в столбце;

MAX — определяет максимальное значение в столбце;

MIN — определяет минимальное значение в столбце.
Пример использования агрегатных функций с оконной конструкцией OVER на слайде.

Ранжирующие функции

Ранжирующие функции – это функции, которые ранжируют значение для каждой строки в окне. Например, их можно

использовать для того, чтобы присвоить порядковый номер строки или составить рейтинг.

ROW_NUMBER – функция возвращает номер строки и используется для нумерации;

RANK — функция возвращает ранг каждой строки. В данном случае значения уже анализируются и, в случае нахождения одинаковых, возвращает одинаковый ранг с пропуском следующего значения;

DENSE_RANK — функция возвращает ранг каждой строки. Но в отличие от функции RANK, она для одинаковых значений возвращает ранг, не пропуская следующий;

NTILE – это функция, которая позволяет определить к какой группе относится текущая строка. Количество групп задается в скобках.

Функции смещения

Функции смещения – это функции, которые позволяют перемещаться и обращаться к разным строкам в окне, относительно текущей строки, а также обращаться к значениям в начале или в конце окна.

LAG или LEAD – функция LAG обращается к данным из предыдущей строки окна, а LEAD к данным из следующей строки. Функцию можно использовать для того, чтобы сравнивать текущее значение строки с предыдущим или следующим. Имеет три параметра: столбец, значение которого необходимо вернуть, количество строк для смещения (по умолчанию 1), значение, которое необходимо вернуть если после смещения возвращается значение NULL;

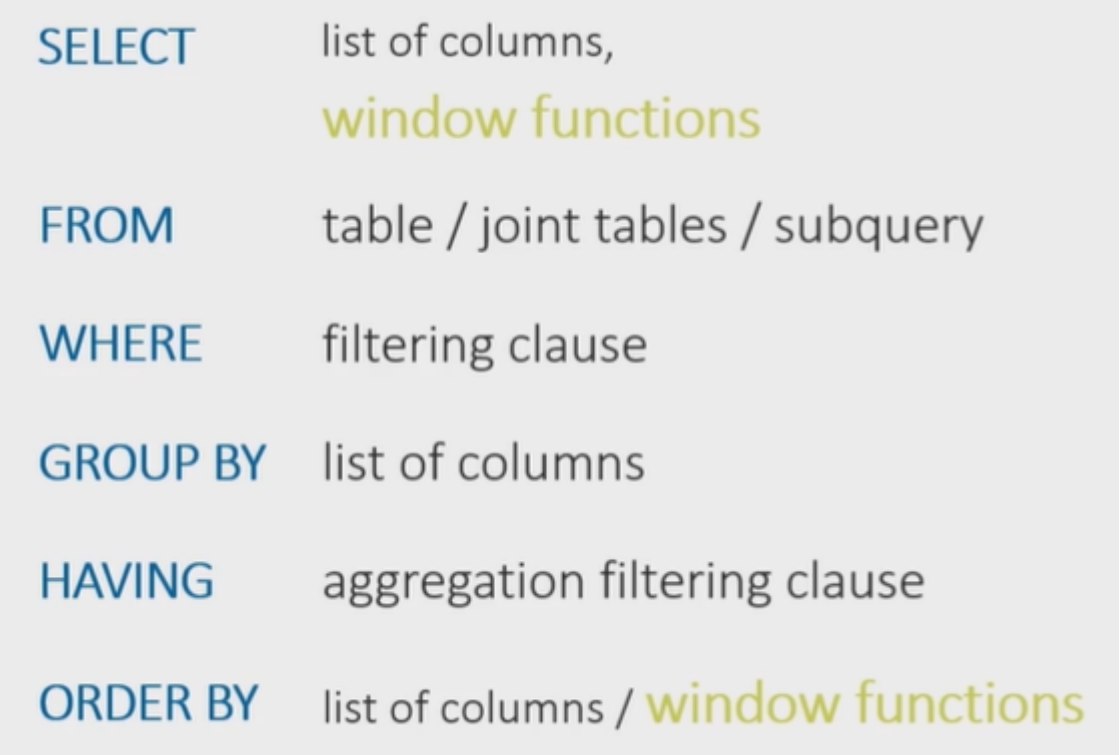
FIRST_VALUE или LAST_VALUE — с помощью функции можно получить первое и последнее значение в окне. В качестве параметра принимает столбец, значение которого необходимо вернуть.

Аналитические функции

Аналитические функции — это функции которые возвращают информацию о распределении данных и используются для статистического анализа.
CUME_DIST — вычисляет интегральное распределение (относительное положение) значений в окне;
PERCENT_RANK — вычисляет относительный ранг строки в окне;

PERCENTILE_CONT — вычисляет процентиль на основе постоянного распределения значения столбца. В качестве параметра принимает процентиль, который необходимо вычислить (в этой статье я рассказываю как посчитать медиану, благодаря этой функции);
PERCENTILE_DISC — вычисляет определенный процентиль для отсортированных значений в наборе данных. В качестве параметра принимает процентиль, который необходимо вычислить.
Важно! У функций PERCENTILE_CONT и PERCENTILE_DISC, столбец, по которому будет происходить сортировка, указывается с помощью ключевого слова WITHIN GROUP.
Подытожим: ***в чем заключается главное отличие оконных функций от функций агрегации с группировкой?***
При использовании агрегирующих функций предложение GROUP BY сокращает количество строк в запросе с помощью их группировки.
При использовании оконных функций количество строк в запросе не уменьшается по сравнении с исходной таблицей.

Порядок расчета оконных функций в SQL запросе



Сначала выполняется команда выборки таблиц, их объединения и возможные подзапросы под командой FROM.
Далее выполняются условия фильтрации WHERE, группировки GROUP BY и возможная фильтрация с HAVING

Только потом применяется команда выборки столбцов SELECT и расчет оконных функций под выборкой.
После этого идет условие сортировки ORDER BY, где тоже можно указать столбец расчета оконной функции для сортировки.
Здесь важно уточнить, что партиции или окна оконных функций создаются после разделения таблицы на группы с помощью команды GROUP BY, если эта команда используется в запросе. Результаты наших запросов можно помещать в специальные виртуальные таблички, о которых мы с вами поговорим в следующей главе.

Представление (VIEW)

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению. Представления иногда называют «виртуальными таблицами». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится. При этом, механизм кэширования запросов (query cache) работает на уровне запросов пользователя безотносительно к тому, обращается ли пользователь к таблицам или представлениям. Представления могут основываться как на таблицах, так и на других представлениях, т.е. могут быть вложенными (до 32 уровней вложенности).

Общий синтаксис представления:

```
CREATE [OR REPLACE] VIEW view_name AS

  SELECT columns

  FROM tables

  [WHERE conditions];
```

Параметры или аргументы

OR REPLACE - необязательный. Если вы не укажете этот атрибут и VIEW уже существует, оператор CREATE VIEW вернет ошибку.
view_name - имя VIEW, которое вы хотите создать в MySQL.
WHERE conditions - необязательный. Условия, которые должны быть выполнены для записей, которые должны быть включены в VIEW.

Пример:

```
CREATE VIEW Londonstaff
          AS SELECT *
          FROM Salespeople
          WHERE city = 'London';
```

Теперь вы имеете представление, называемое Londonstaff. Вы можете использовать это представление точно так же как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из, и соединена с, другими таблицами и представлениями. Давайте сделаем запрос такого представления

```
SELECT *
FROM Londonstaff;

```

Когда вы приказываете SQL выбрать(SELECT) все строки (*) из

предсдавления, он выполняет запрос содержащий в определении - Loncfonstaff, и возвращает все из его вывода. Имея предикат в запросе представления, можно вывести только те строки из представления, которые будут удовлетворять этому предикату.
(Предикат в SQL это: в широком смысле, — любое выражение, результатом которого являются значения булевого типа — TRUE, FALSE, а так же UNKNOWNв узком смысле, — некий уточняющий фильтр. Самым явным примером предиката служит оператор WHERE)
Пример №2.

Пусть имеется таблица, из которой мы создадим представление - таблицу customer_archive с полями id, имя, номер и город по условию: сумма покупки > 10000

Операции с представлениями

1. DROP: представление/виртуальную таблицу можно удалить с помощью команды DROP VIEW. Если мы хотим удалить таблицу customer_archive.

2. Объединение: мы также можем создать представление, объединив несколько таблиц. Это соединение будет извлекать совпадающие записи из обеих таблиц. (текст к слайду: Выше приведен пример внутреннего соединения. Таким же образом мы можем применить и другие соединения. В приведенном выше примере представление будет создано путем объединения записей, присутствующих как в table_name1, так и в

table_name2, на основе общего поля)

3. Создание рассмотрено выше

4.Оператор MySQL ALTER VIEW изменяет определение существующего представления. Синтаксис оператора ALTER VIEW аналогичен CREATE VIEW оператору.Вот пример того, как вы будете использовать оператор ALTER VIEW в MySQL (на экране).Этот пример ALTER VIEW в MySQL обновить определение VIEW с именем hardware_suppliers, не удаляя его. В этом примере мы добавляем столбцы address и city в VIEW

Преимущества:

Безопасность: существует множество таблиц, доступ к которым ограничен для многих пользователей, поскольку некоторые атрибуты в этих таблицах будут очень конфиденциальными. Таким образом, если мы можем создавать представления с некоторыми специфическими атрибутами для соответствующих пользователей, то пользователям может быть предоставлено разрешение на доступ к некоторому набору представлений в базе данных, которая им разрешена. Это может поддерживать безопасность и целостность данных, а также пользователи могут выполнять свои задачи с соответствующими авторизованными столбцами.

Простота запроса. Представление может быть создано путем выборки данных из нескольких таблиц. Таким образом, все совокупные записи из всех таблиц могут быть представлены

одной таблицей с помощью запроса просмотра.

Структурная простота: мы можем создать специализированное или персонализированное представление для конкретного пользователя. Таким образом, мы можем представить базу данных как набор виртуальных таблиц, понятных пользователю.

Непротиворечивость: мы упоминаем здесь согласованность, потому что это представление может представлять собой непротиворечивое и неизменное изображение структуры базы данных, даже если мы выполняем некоторые манипуляции с основной таблицей или главной таблицей.

Целостность данных: если данные доступны для представления, база данных всегда проверяет данные, чтобы убедиться, что они удовлетворяют ограничениям целостности или нет.

Недостатки:

Производительность. Представления — это виртуальные таблицы или представители основных таблиц. Когда мы запускаем некоторые запросы для создания представления, СУБД преобразует эти запросы к представлениям в запросы в базовых таблицах. Таким образом, если запрос представления очень сложный, содержит несколько источников и сложные алгоритмы, то простые действия с этими представлениями занимают значительное время.

Ограничения обновления: при изменении строк в представлении СУБД должна преобразовать запрос в

обновление строк базовой исходной таблицы. Обновление можно выполнить в простом запросе, но в случае сложного запроса СУБД не позволит обновить, так как представления часто ограничены только чтением.

ЧТО НЕ МОГУТ ДЕЛАТЬ ПРЕДСТАВЛЕНИЯ?

Имеется большое количество типов представлений (включая многие из наших примеров в этой главе), которые являются доступными только для чтения. Это означает, что их можно запрашивать, но они не могут подвергаться действиям команд модификации. Имеются также некоторые виды запросов, которые недопустимы в определениях представлений.

- Одиночное представление должно основываться на одиночном запросе.

- ОБЪЕДИНИТЬ (UNION) и ОБЪЕДИНИТЬ ВСЕ (UNION ALL

- УПОРЯДОЧИТЬ ПО (ORDER BY) никогда не используется в определении представлений.

Вывод запроса формирует содержание представления, которое напоминает базовую таблицу и является - по определению - неупорядоченным

РЕЗЮМЕ

Теперь, когда вы можете использовать представления, ваша способность отслеживать и обрабатывать содержание БД значительно расширилась. Всё, что вы можете создать запросом, вы всегда сможете определить как представление. Запросы этих

представлений это, фактически, запрос запроса. Использование представлений и для удобства, и для защиты также удобно, как и многие возможности представлений для форматирования и получения значений из постоянно меняющегося содержания вашей БД.Имеется один главный вывод относительно представлений: это способность к модификации. Как показано, вы можете модифицировать представления так же, как и базовую таблицу, с помощью изменений, применяемых к таблице, из которой получается представление, но это не всегда возможно.

Задача на 5 минут: создайте представление, которое показывало бы всех заказчиков, имеющих самые высокие рейтинги. Привязке к таблице нет, колонки таблицы воображаемые

```
CREATE VIEW Highratings
      AS SELECT *
         FROM Customers
         WHERE rating =
           (SELECT MAX (rating)
            FROM Customers);
```

Итоги

Сегодня мы узнали, как выполнять запросы в отдельных виртуальных таблицах, называемых представлениями, разобрались с понятиями оконных функции и узнали разницу между ними.

На следующей лекции мы изучим:

Понятие транзакции, свойства ACID.

Понятие сессии, временные таблицы, область видимости

Переменные, вывод текстовых сообщений

Оператор IF

Оператор While