Специалист по Data Science

Курс по SQL

Матвеева Анна, Владислав Бояр

1 Занятие Первое: введение в SQL, основные блоки запросов, типы данных

На первом уроке мы разбиремся с общим понятием SQL. Познакомимся с основными блоками-командами, узнаем, какие бывают типы данных. И наконец подключимся к Базе данных и выполним первые запросы.

1.1 Введение в SQL

Каждый, кто начинает знакомиться с SQL, в начале процесса обучения задается вопросами: “Что это такое?” , “Как технически реализуется?”, “Какую роль играет при разработке того или иного продукта?” и тд. Постепенно в этом курсе мы ответим на все эти вопросы и даже больше.

Для начала, остановимся на первом вопросе: “Что это такое?”.

Определение: 1 *SQL (Structured Query Language) - это стандартный язык программирования (его еще называют языком запросов) для управления реляционными базами данных.*

SQL был разработан для создания, модификации и управления данными в базах данных. SQL является мощным инструментом для обработки и анализа

информации, а также для создания отчетов.

В определении SQL встретилось такое понятие как “Реляционные базы данных” (РБД).

Определение: 2 *Реляционная база данных (РБД) - это собрание данных, организованных в таблицы по реляционной модели. Данные хранятся в виде строк и столбцов, таблицы связаны между собой, используются ключи для идентификации записей, поддерживается целостность данных, и осуществляются операции CRUD (Create, Read, Update, Delete). РБД широко применяются в различных областях благодаря своей гибкости и эффективности.*

Так вот, сам по себе SQL не может существовать отдельно от РБД (в этом не будет никакого смысла, так как главная задача SQL - обратиться к источнику за данными), поэтому SQL - это всё же язык запросов, а не полноценный язык программирования (как питон, C++ и тд.).

Теперь, можно перейти ко второму вопросу: “Как технически реализуется?”. Из компонентов, чтобы представить технически работу SQL, у нас есть: сам SQL и РБД. По сути SQL - это текст, а РБД - объект, так где и как этот текст будет обращаться к РБД (а может даже через что-то)? Если у читателя сейчас возникло чувство, что явно чего-то не хватает - все верно, теперь познакомимся с СУБД.

Определение: 3 *Понятие СУБД (система управления базами данных) относится к программному обеспечению, которое позволяет создавать, управлять и обрабатывать базы данных. СУБД предоставляет средства для хранения, организации, обновления и извлечения данных, а также для управления доступом и обеспечения целостности данных.*

СУБД является широким термином, охватывающим различные типы баз данных и их реализации. Некоторые из наиболее распространенных СУБД

включают MySQL, PostgreSQL, Oracle, Microsoft SQL Server и SQLite. Каждая из этих СУБД имеет свои особенности, функциональность, производительность и возможности.

Итак, теперь есть все необходимое, чтобы понять работу SQL.

Рис. 1: Схема работы SQL запроса посредством СУБД.

Пользователь прописывает SQL запрос, СУБД его принимает и обрабатывает для РБД. После чего полученная информация также через СУБД доставляется пользователю. В нашем курсе мы будем учиться как раз писать запросы SQL разной сложности и получать данные для дальнейшей работы с ними.

Итак, теперь мы полностью разобрались, как именно запрос SQL работает с РБД. Однако, когда мы вводили определение СУБД, было упомянуто и не случайно, что они могут быть разными. Под каждую СУБД есть свой **диалект SQL**.

Определение: 4 *Понятие Диалекта SQL относится к различным вариациям SQL, имеющие некоторые отличия в синтаксисе и функциональности, хотя основные концепции SQL остаются примерно одинаковыми. Каждая СУБД может иметь свой собственный диалект SQL или поддерживать несколько диалектов для взаимодействия с данными.*

Например, SQL-запросы, написанные для PostgreSQL, могут отличаться от SQL-запросов для Oracle или MySQL. Это может включать различия в ключевых словах, операторах, функциях и расширенных возможностях, предоставляемых конкретным диалектом SQL.

Для понимания, что такое диалект, можно провести аналогию с английским языком, где международный английски - это SQL как он есть, а американский/британский/австралиский акценты - это его диалекты.

И вот мы подошли к третьему вопросу, думаю уже становится очевидным то, что роль SQL, как навыка для специалиста DS - огромна. Пока что мы остановимся на этом, но в процессе обучения не раз еще подтвердятся эти слова.

1.2 Подключение к БД

Итак, теперь мы перешли уже к изучению самого SQL. Первый вопрос, который вероятно возникнет у читателя - какой же именно диалект мы будем изучать? Ответ: **PostgreSQL**.

Это один из самых наиболее распространенных диалектов в девелопменте. Он является мощной объектно-реляционной системой управления базами данных (СУБД), которая предлагает множество расширений и возможностей. Также поддерживает сложные запросы, полнотекстовый поиск, триггеры, представления и многие другие функции, которые делают его популярным выбором для приложений с высокими требованиями к данным.

1.2.1 Скачивание и установка

Работа с БД осуществляется через специальные инструменты (приложения, среду разработки). Самый простой и доступный - DBeaver. Именно его мы и будем использовать. Установить DBeaver можно **тут**.

Чтобы установить DBeaver на Windows/macOS, нужно:

1. Запустить установочный файл.
2. Выбрать язык и нажать кнопку «OK».
3. Следовать указаниям программы (выбрать all users).
4. Выбрать все компоненты программы и нажать «Далее».
5. Выбрать папку для установки программы.
6. Следовать указаниям установщика.
7. Дождаться окончания установки.
8. Нажать кнопку «Готово» для завершения установки.

1.2.2 Создание нового соединения с БД

Чтобы создать новое соединение с базой данных в DBeaver, нужно:

1. Запустить DBeaver.
2. Выбрать «Новое соединение» в меню «База данных».
3. Выбрать тип базы данных, с которой нужно создать соединение.
4. Указать параметры соединения — хост, порт, имя пользователя и пароль. Значения этих параметров зависят от типа базы данных, с которой предстоит работать.
5. Нажать кнопку «Тест соединения» для проверки правильности ввода данных и работы соединения.
6. Нажать кнопку «Готово» для создания соединения.

Доступ к нашей БД курса

Host: rc1b-7ng6ih3jte3824x8.mdb.yandexcloud.net
Port: 6432
Database: demo
Username: student
Password: student!

1.3 Основные блоки запросов

Блоки запросов SQL состоят из нескольких ключевых элементов, которые позволяют выполнить различные операции с данными. Для понимания, как строится запрос SQL, мы сначала познакомимся с основными обязательными и опциональными блоками запросов, а именно для чего каждый из элементов нужен:

SELECT (выбрать): Оператор SELECT (обязательно) используется для извлечения данных из базы данных. Он позволяет указать столбцы, которые необходимо выбрать, а также условия фильтрации и сортировки.

FROM (откуда): Ключевое слово FROM (обязательно) указывает таблицу или таблицы, из которых следует извлечь данные. Оно определяет источник данных для запроса.

WHERE (где/на условии): Ключевое слово WHERE используется для определения условий фильтрации данных. Оно позволяет указать критерии, которым должны соответствовать данные, чтобы быть включенными в результаты запроса.

JOIN ON (добавляя данные из другой таблицы): Ключевое слово JOIN используется для объединения данных из двух или более таблиц на основе связей между ними. Оно позволяет объединить данные из разных таблиц на основе общих значений ключевых полей.

GROUP BY (группируя по): Ключевое слово GROUP BY используется для группировки данных по определенным столбцам. Это позволяет выполнять агрегатные функции, такие как сумма, среднее или подсчет, на группах данных.

ORDER BY (сортируя по): Ключевое слово ORDER BY используется для сортировки результатов запроса по одному или нескольким столбцам. Оно позволяет указать порядок сортировки данных, как по возрастанию, так и по убыванию.

Сформируем вид **самого тривиального запроса**, то есть, такой запрос, в котором будут участовать только обязательные элементы: SELECT и FROM.

```
1. SELECT * FROM shema_name.table_name

2. SELECT column1, column2 FROM shema_name.table_name
```

Результатом выполнения 1-ого запроса - вся таблица со всеми столбцами. Произойдет это потому, что между SELECT и FROM стоит * - означет, что мы хотим увидеть все столбцы.

Результатом выполнения 2-ого запроса - пользователь увидет только столбцы column1 и column2, так как они были непосредственно прописаны в запросе.

Сейчас читателю впервые встретилось такое понятие как схема (shema name). Многие разработчики опускают название схемы после FROM, оставляя только название таблицы.

Определение: 5 *Схема в SQL - логическая организация и контейнер для объектов базы данных. Она позволяет организовать и структурировать объекты, разделять доступ и управлять разрешениями. Схема обеспечи-*

вает изоляцию, управление доступом и повторное использование объектов базы данных.

Так лучше не делать, из-за возможной путаницы между таблицами в зависимости от подключения SQL Editor ко всей БД или только к одной схеме.

Теперь перейдем к **WHERE**. Так как это условие фильтрации таблицы, то напишем два немного разных запроса.

```
1. SELECT * FROM shema_name.table_name WHERE column1 = '2'

2. SELECT column1, column2 FROM shema_name.table_name WHERE column3 = 'a'
```

Результатом выполнения 1-ого запроса будет вся таблица со всеми столбцами, НО выведутся только те строки, в которых в столбце column1 есть значение 2.

Результатом выполнения 2-ого запроса - пользователь увидет только столбцы column1 и column2, И ТОЛЬКО ТЕ СТРОКИ, для которых в столбце column3 есть значение a.

Есть еще один оператор, с которым мы познакомимся в этом уроке - ALIAS (AS), хоть он не входит в перечень основных - это не отменяет его полезность.

AS (псевдоним): Оператор, который используется для того, чтобы присвоить столбцу или таблице новое имя - псевдоним. Больше используется как вспомогательный инструмент.

```
1. SELECT * FROM shema_name.table_name [as] Name

2. SELECT column1 [as] cl1 , column2 [as] cl2 FROM shema_name.table_name
```

В 1-ом запросе особо ничего не меняется, НО теперь наша таблица, из которой мы тянем данные (в рамках этого запроса), называется Name. Позже нам пригодится это, когда будем работать с JOIN.

Во 2-ом запросе мы уже присваиваем новые имена столбцам column1 на cl1 и column2 на cl2. Позже эта возможность нам пригодится, когда мы будем

проходить агрегационные фун-ции.

Обращу внимание читателя, что оператор в запросах заключен в [квадратные скобки] - это значит, что сам по себе оператор можно не писать и просто написать рядом со столбцом/таблицей желаемый псевдоним. Очень советую на первое время не опускать само слово оператора AS, иначе есть вероятность запутаться, что где было переименовано.

С остальными элементами мы познакомимся в следующих уроках!

1.4 Типы данных

SQL поддерживает разнообразные типы данных, которые определяют виды значений, которые могут быть сохранены в столбцах таблиц базы данных. Некоторые из распространенных типов данных в SQL включают:

1. **Числовые типы данных**:
- INTEGER: целочисленный тип данных для хранения целых чисел.
- FLOAT или DOUBLE: тип данных с плавающей точкой для хранения чисел с десятичной точкой.
- DECIMAL или NUMERIC: тип данных для хранения чисел с фиксированной точностью и масштабом.

2. **Символьные и текстовые типы данных**:
- CHAR: фиксированная длина строки символов.
- VARCHAR: переменная длина строки символов.
- TEXT: тип данных для хранения длинных текстовых значений.

3. **Даты и времена**:
- DATE: тип данных для хранения даты.
- TIME: тип данных для хранения времени.
- TIMESTAMP: тип данных для хранения даты и времени.

4. **Булев тип данных**:
- BOOLEAN: тип данных, который может принимать значения TRUE или FALSE.

5. **Бинарные типы данных**:
- BLOB: тип данных для хранения двоичных больших объектов, таких как изображения или файлы.

- BINARY: тип данных для хранения двоичных значений фиксированной длины.

6. **Другие типы данных**:
- ENUM: тип данных, который позволяет задать список возможных значений для столбца.
- JSON: тип данных для хранения данных в формате JSON.

Это лишь некоторые из типов данных, поддерживаемых SQL. Различные СУБД могут также предлагать расширенные и специфические типы данных в зависимости от их функциональности и особенностей.

1.5 Практика

Итак, в этой части занятия мы напишем простые запросы к БД.
Для того, чтобы начать писать запросы, нужно:

1. Подлючиться к БД (см. пункт 1.2). Если подключение прошло успешно, в итоге БД должна высветиться слева с зеленой галочкой.

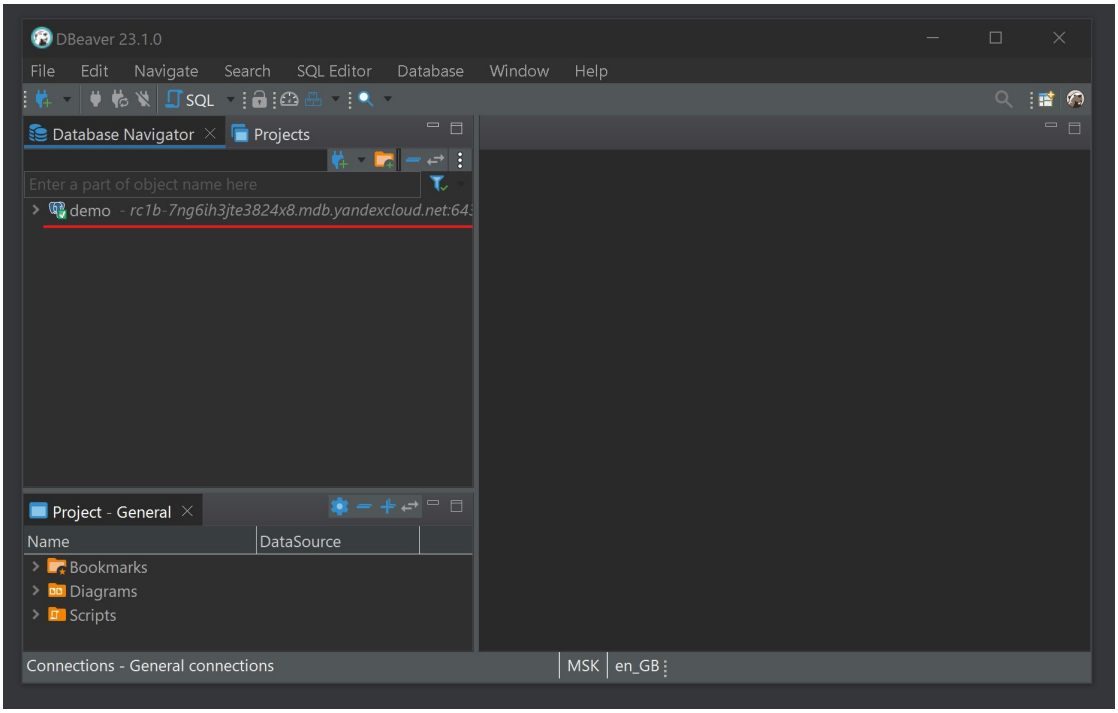

Рис. 2: Окно работы DBeaver и успешное подключение к базе demo.

2. Теперь нужно открыть область (щелкнув правой кнопкой мыши по БД demo), в которой мы прописываем запросы. Работать можно и просто в консоле (Open SQL console), но я рекомендую создавать файл (New SQL script) и прописывать в нем запросы.

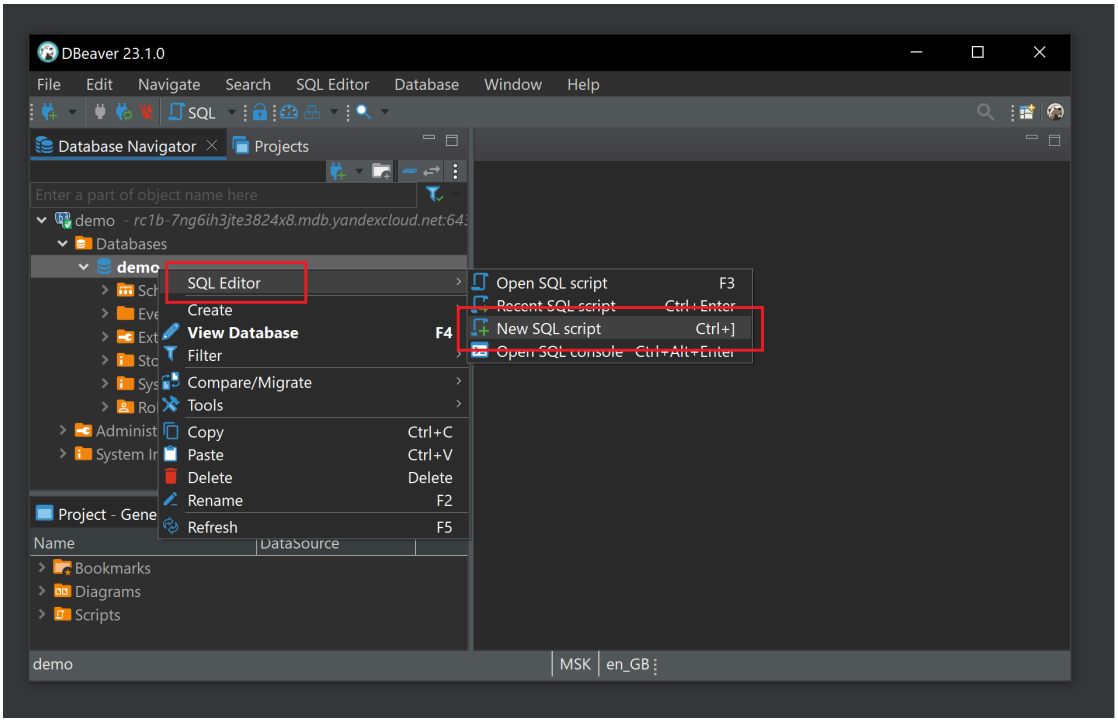

Рис. 3: Открываем SQL Editor как новый файл.

Бывает так, что хочется посмотреть, какие запросы ты писал вчера/позавчера, консоль не имеет физической памяти (только на время работы с ней, после закрытия DBeaver память отчистится), поэтому в таком случае лучше вести работу в файле (все файлы можно будет найти на диске C:/, куда был установлен Dbeaver).

3. Если все прошло успешно, должно появиться вот такое окно и как раз при наведении на *<demo> Script* вылезет данные по подключению и пусть до этого файла.

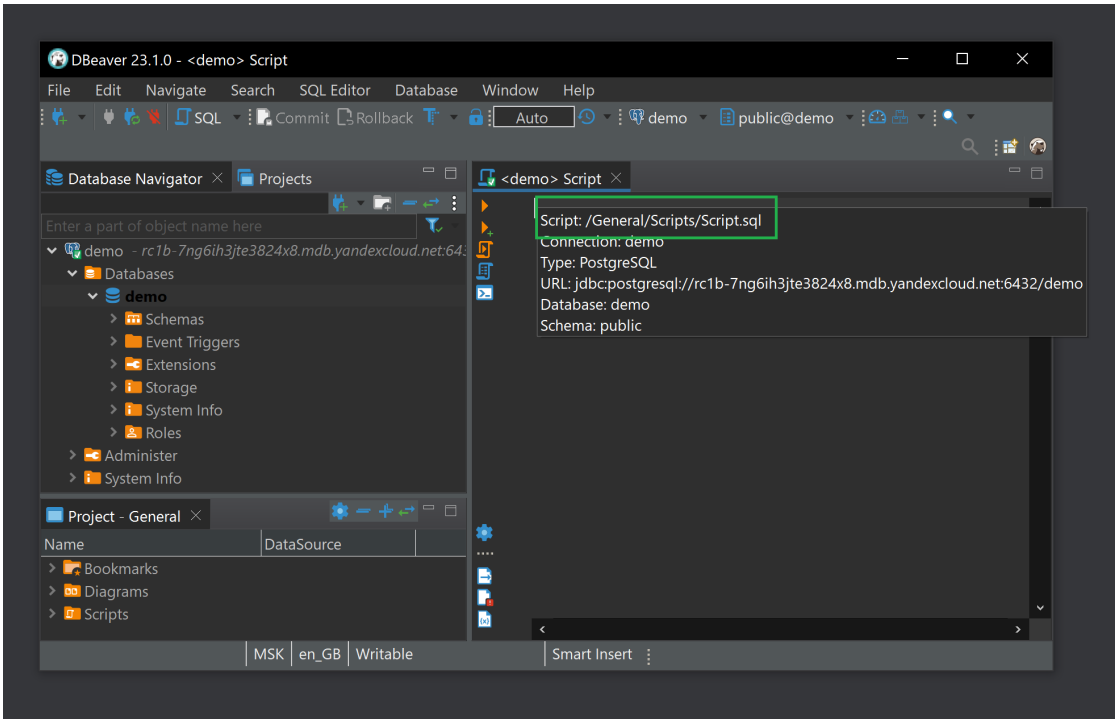

Рис. 4: Созданное пространство для написание запросов.

4. Теперь выполним первый запрос: посмотрим всю иформафию, которая содержится в схеме bookings в табличке flights.

```
SELECT * FROM bookings.flights
```

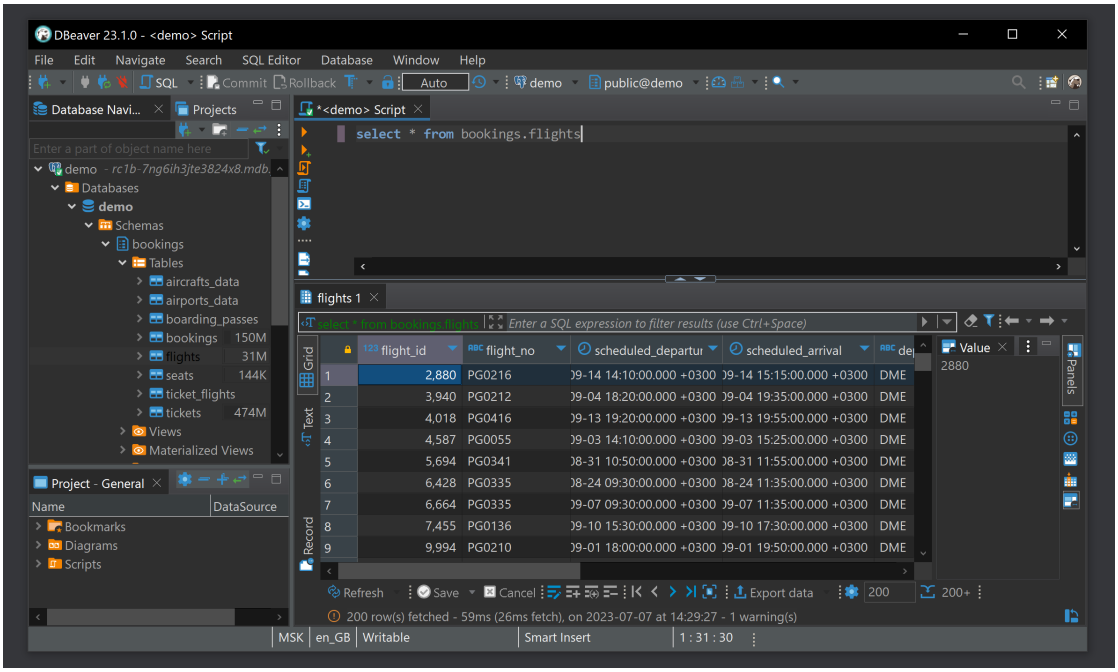

Рис. 5: Результат выполнения запроса.

Задания для отработки навыка написания базовых запросов:

1. Вывести всю информацию из таблицы tickets.

2. Вывести всю информацию из таблицы boarding passes.

3. Вывести всю информацию из таблицы tickets, где имя пассажира является VIKTORIYA SMIRNOVA

4. Вывести всю информацию из таблицы flights, где номер полета является PG0216

5. Вывести столбцы seat no из таблицы seats, где состояние кресла Business

2 Занятие Второе: Агрегационные функции и подзапросы.

Агрегационные функции в PostgreSQL предоставляют мощный инструментарий для суммирования, подсчета, вычисления статистических показа-

телей и других операций над данными в базе данных. Они позволяют группировать данные по критериям и применять агрегатные операции к каждой группе.

Определение: 6 *Агрегационная функция в SQL - это функция, применяемая к группе значений для вычисления единственного агрегированного результата, такого как сумма, среднее значение или количество. Они используются с оператором GROUP BY для группировки данных и вычисления агрегатных значений по каждой группе.*

Напомним читателю опередеделение оператора GROUP BY.

Определение: 7 *Оператор GROUP BY используется для группировки результирующего набора данных по одному или нескольким столбцам. Он позволяет объединить строки с одинаковыми значениями в указанных столбцах и применить агрегационные функции к каждой группе отдельно.*

На данный момент понятно, что агрегационная функция считает какое-то единственное агрегированное (сгруппированное) значение по группе (группа может быть как и единым столбцом, так и несколькими столбцами).

2.1 Тривиальные случаи использования GROUP BY и агрегационных функций

Можно заметить, что GROUP BY как будто всегда идет в комбинации с агрегационной функцией, поэтому в свою очередь агрегационная функция не может существовать без оператора GROUP BY. Вполне вероятно, что тогда возникают два вопроса:

1. Может ли GROUP BY существовать без агрегационной функции?

2. Может ли агрегационная функция существовать без GROUP BY?

Итак, ответом на оба эти вопроса будет ДА, что возможно удивит читателя. Поэтому стоит сказать, что случай, когда 1 или 2 ситуация возможна, очень тривиальный.

1. Оператор действительно может существоать без какой-либо функции. В таком случае GROUP BY будет использоваться только для группировки строк по заданным столбцам, без применения агрегаций к каждой группе. Результатом будет набор уникальных комбинаций значений в указанных столбцах, и каждая строка в результирующем наборе будет представлять одну группу. ВАЖНО, мы выбираем либо один столбец в теле запроса и один столбец в GROUP BY, либо два столбца и более в теле запроса и два и более столбца в GROUP BY, то есть, у нас ВСЕГДА должны быть одни и те же столбцы в теле запроса (всегда должны быть такие одинаковые "пары") и в операторе, чтобы узнать уникальные группы.

Группировка по одному столбцу

```
SELECT column1 FROM shema_name.table_name
GROUP BY column1
```

Группировка по двум столбцам

```
SELECT column1, column2 FROM shema_name.table_name
GROUP BY column1 column2
```

Группировка по трем столбцам

```
SELECT column1, column2, column3 FROM shema_name.table_name
GROUP BY column1, column2, column3
```

2. Когда вы используете агрегационную функцию без оператора GROUP BY, она будет применяться ко всему набору данных, возвращенному запросом, и вернет единственное агрегированное значение для всего набора данных. Например, следующий запрос вычисляет общее количество строк в таблице.

```
SELECT COUNT(*) FROM shema_name.table_name
```

Здесь нет оператора GROUP BY, и агрегационная функция COUNT(*) применяется ко всему набору данных, возвращенному запросом, и возвращает единственное значение - общее количество строк.

Приведем еще один пример для понимания. Теперь мы используем агре-

гационную функцию SUM() к столбцу column1, то есть, мы просто считаем сумму по всему этому столбцу и НИКАК не группируем.

```
SELECT SUM(column1) FROM shema_name.table_name
```

Таким образом, сейчас мы разобрали с вами тривиальные случаи использования GROUP BY и агрегационных функций.

2.2 Типы агрегационных функций

Приведем сейчас наиболее используемые агрегационные функции:

– Функции подсчета и суммирования: COUNT, SUM

– Функции для вычисления среднего значения: AVG

– Функции нахождения максимального/минимального значения: MAX, MIN

– Функции создания массивов и объединения строк: ARRAY_AGG, STRING_AGG

– Функции для вычисления статистических значений: STDDEV, VARIANCE

– Функции для работы с JSON и XML: JSON_AGG, JSONB_AGG, XMLAGG

– Функции для работы с логическими значениями: BOOL_AND, BOOL_OR

– Функции для вычисления перцентилей: PERCENTILE_DISC, PERCENTILE_CONT

– Функция для нахождения моды: MODE

2.3 Расположение агрегационных функций

Агрегационные функции могут находиться в различных частях SQL-запроса, в зависимости от того, как они должны быть применены и какой результат они должны вернуть.

1. В операторе **SELECT**: Агрегационные функции могут использоваться в списке выбора, чтобы вычислить агрегированные значения и включить их в результирующий набор данных. Например:

```
SELECT
    COUNT(*) AS total_rows,
    AVG(price) AS average_price
FROM table_name
```

2. В операторе **HAVING**: Агрегационные функции могут использоваться в операторе HAVING для фильтрации групп данных на основе агрегированных значений. Например:

```
SELECT
    category,
    COUNT(*) AS count
FROM table_name
GROUP BY category
    HAVING COUNT(*) > 5
```

3. В операторе **ORDER BY**: Агрегационные функции могут использоваться в операторе ORDER BY для упорядочивания результирующего набора данных на основе агрегированных значений. Например:

```
SELECT category,
    COUNT(*) AS count
FROM table_name
GROUP BY category
    ORDER BY COUNT(*) DESC
```

4. В операторе **WHERE**: Агрегационные функции могут использоваться в операторе WHERE для фильтрации данных на основе агрегированных значений. Например:

```
SELECT
    column1,
    column2
FROM table
    WHERE COUNT(column1) > 10;
```

Или более интересный случай использования, тут важно понять, что сначала мы будем фильтровать по WHERE и условию средней цены в 10, а уже потом применяем группировку и агрегацию в SELECT.

```
SELECT
    category,
    AVG(price) AS avg_price
FROM table_name
GROUP BY category
    WHERE AVG(price) > 100
```

Важно отметить, что в операторе WHERE агрегационные функции обычно требуют использования предиката HAVING, так как агрегированные значения недоступны на этапе фильтрации данных в WHERE. Оператор WHERE фильтрует строки до группировки и агрегации, а оператор HAVING фильтрует группы после группировки и агрегации.

2.4 Подзапросы

Подзапросы (subqueries) в PostgreSQL представляют собой вложенные запросы, которые выполняются внутри других запросов. Они используются для извлечения данных из одной таблицы или набора таблиц и использования полученных результатов в других частях запроса.

Определение: 8 *Подзапросы (subqueries) в PostgreSQL - это запросы, которые вложены в другие запросы и используются для получения данных из базы данных. Они используются для выполнения дополнительных операций, фильтрации, сравнения или вычислений на основе результатов других запросов.*

Определение подзапросов можно разделить на две категории: подзапросы с возвратом **набора строк** (результаты запроса) и подзапросы с возвратом **единственного значения**. Рассмотрим оба случая с примерами.

2.4.1 Подзапросы с возвратом набора строк

Подзапросы, возвращающие наборы строк, могут использоваться в различных частях SQL-запроса, таких как предложения SELECT, FROM, WHERE, HAVING, JOIN и других.

– Пример 1: Фильтрация данных. Выбрать всех пользователей, у которых есть заказы в определенной категории:

```
SELECT *
FROM users
WHERE user_id IN
(SELECT user_id FROM orders WHERE category = 'Electronics')
```

– Пример 2: Вложенный запрос. Получить сумму заказов для каждого пользователя, используя подзапрос в предложении SELECT

```
SELECT
    user_id,
    ( SELECT SUM(order_total)
        FROM orders WHERE user_id = users.user_id
    ) AS total_orders
FROM users
```

2.4.2 Подзапросы с возвратом единственного значения

Подзапросы, возвращающие единственное значение, могут быть использованы для сравнения, вычисления или фильтрации данных.

– Пример 1: Сравнение значений. Выбрать продукты, у которых цена выше средней цены:

```
SELECT *
FROM products
WHERE price > (SELECT AVG(price) FROM products)
```

– Пример 2: Фильтрация данных. Выбрать всех пользователей, у которых общая сумма заказов превышает заданную величину:

```
SELECT *
FROM users
WHERE (SELECT SUM(order_total) FROM orders
WHERE user_id = users.user_id) > 1000
```

2.5 Практика

1. Найти аэропорты (departure airport), где количество рейсов превышает среднее количество рейсов по всем аэропортам.

2. Найти рейсы, где суммарная стоимость билетов превышает среднюю стоимость билетов всех рейсов.

3. Найти среднюю продолжительность полета для каждого аэропорта прибытия.

4. Найти аэропорты с наибольшим количеством различных моделей самолетов, прилетающих в них.

5. Найти аэропорты, в которых есть рейсы, отправляющиеся во все другие аэропорты.