МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра компьютерных интеллектуальных технологий проектирования

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

По дисциплине: «Системы хранения и обработки данных»

Тема: «Формирование запросов к базе данных»

Выполнил работу студент группы мИИВТ-241: Бокарев С.А.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Цель работы: изучить синтаксис и основные команды для формирования запросов к базе данных, освоить процесс формирования SQL-запросов.

Основные задачи:

• Разработать структуру базы данных в СУБД Postgres.

• Наполнить базу данных данными.

• Написать ряд запросов к базе данных для отработки навыка формирования SQL-запросов.

• Научиться ставить задачу по формированию выборки необходимых данных из базы данных и решать её с помощью SQL-запросов.

После изучения синтаксиса SQL-запросов в СУБД Postgres, основ формирования SQL-запросов и способов оптимизации запросов, приступим к выполнению практических заданий.

Общее задание:

Формирование SQL-запросов для создания таблиц.

CREATE TABLE Customers (

CustomerID SERIAL PRIMARY KEY,

FirstName VARCHAR(255),

LastName VARCHAR(255),

Email VARCHAR(255)

);

CREATE TABLE Orders (

OrderID SERIAL PRIMARY KEY,

CustomerID INT,

OrderDate TIMESTAMP,

TotalAmount DECIMAL(10, 2)

);

CREATE TABLE OrderDetails (

OrderDetailID SERIAL PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

UnitPrice DECIMAL(10, 2)

);

Далее необходимо наполнить базу данных данными, для этого были использованы команды "INSERT INTO" для добавления новых записей в таблицу базы данных. Фраза "VALUES" в этом контексте указывает на то, что нужно указать конкретные значения для каждого столбца вставляемой записи.

INSERT INTO Customers (FirstName, LastName, Email)

VALUES

INSERT INTO Orders (CustomerID, OrderDate, TotalAmount)

VALUES

INSERT INTO OrderDetails (OrderID, ProductID, Quantity, UnitPrice)

VALUES

Затем был сформирован SQL-запрос, который возвращает список клиентов и суммарную стоимость заказов каждого клиента.

SELECT c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName;

Также полученный список был отсортирован по убыванию суммарной стоимости заказов клиента.

SELECT c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY TotalOrderAmount DESC, c.LastName, c.FirstName;

После был добавлен столбец со средней суммарной стоимостью заказов.

SELECT c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount

COALESCE(AVG(SUM(o.TotalAmount)) OVER (), 0) AS AvgTotalOrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY TotalOrderAmount DESC, c.LastName, c.FirstName;

Затем, был выведен клиент с наибольшей суммарной стоимостью заказов.

SELECT c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName

ORDER BY TotalOrderAmount DESC LIMIT 1;

Для каждого клиента с наибольшей суммарной стоимостью заказов был выведен список его заказов порядке возрастания стоимости заказа.

WITH RankedCustomers AS (SELECT c.CustomerID, c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount,

RANK() OVER (ORDER BY COALESCE(SUM(o.TotalAmount), 0) DESC) AS rnk

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName)

SELECT rc.FirstName, rc.LastName, o.OrderID, o.TotalAmount AS OrderTotalAmount

FROM RankedCustomers rc

JOIN Orders o ON rc.CustomerID = o.CustomerID WHERE rc.rnk = 1

ORDER BY o.TotalAmount;

Также были выведены только те клиенты, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов.

WITH CustomerOrderTotals AS (SELECT c.CustomerID, c.FirstName, c.LastName,

COALESCE(SUM(o.TotalAmount), 0) AS TotalOrderAmount

FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.FirstName, c.LastName), AverageOrderTotal AS (SELECT AVG(TotalOrderAmount) AS AvgOrderAmount

FROM CustomerOrderTotals)

SELECT cot.FirstName, cot.LastName, cot.TotalOrderAmount, aot.AvgOrderAmount

FROM CustomerOrderTotals cot

JOIN AverageOrderTotal aot ON 1=1

WHERE cot.TotalOrderAmount > aot.AvgOrderAmount;

Индивидуальное задание, вариант 4:

Формирование SQL-запросов для создания таблиц.

CREATE TABLE Автомобиль (

Автомобиль\_ID SERIAL PRIMARY KEY,

Марка VARCHAR(50),

Модель VARCHAR(50),

Год\_выпуска INT,

Номерной\_знак VARCHAR(10),

Владелец\_ID INT REFERENCES "Владелец автомобиля"(Владелец\_Автомобиля\_ID),

Страховая\_стоимость\_ID INT REFERENCES "Страховая стоимость автомобиля"(Страховая\_стоимость\_автомобиля\_ID)

);

CREATE TABLE "Владелец автомобиля" (

Владелец\_Автомобиля\_ID SERIAL PRIMARY KEY,

Имя VARCHAR(50),

Фамилия VARCHAR(50),

Дата\_рождения DATE,

Адрес VARCHAR(100),

Номер\_телефона VARCHAR(20)

);

CREATE TABLE Водитель (

Водитель\_ID SERIAL PRIMARY KEY,

Имя VARCHAR(50),

Фамилия VARCHAR(50),

Дата\_Рождения DATE,

Адрес VARCHAR(100),

Номер\_телефона VARCHAR(20),

Автомобиль\_ID INT REFERENCES Автомобиль(Автомобиль\_ID)

);

CREATE TABLE "Нарушение ПДД" (

Нарушение\_ПДД\_ID SERIAL PRIMARY KEY,

Описание\_нарушения VARCHAR(100,

Дата\_и\_время\_нарушения TIMESTAMP,

Место\_нарушения VARCHAR(100),

Водитель\_ID INT REFERENCES Водитель(Водитель\_ID)

);

CREATE TABLE Штраф (

Штраф\_ID SERIAL PRIMARY KEY,

Сумма\_штрафа DECIMAL(10,2),

Дата\_оплаты\_штрафа DATE,

Нарушение\_ПДД\_ID INT REFERENCES "Нарушение ПДД"(Нарушение\_ПДД\_ID)

);

CREATE TABLE "Страховая стоимость автомобиля" (

Страховая\_стоимость\_автомобиля\_ID SERIAL PRIMARY KEY,

Сумма\_страхования DECIMAL(10,2),

Дата\_начала\_страхования DATE,

Дата\_окончания\_страхования DATE

);

Далее необходимо наполнить базу данных данными:

INSERT INTO Автомобиль (Марка, Модель, Год\_выпуска, Номерной\_знак, Владелец\_ID)

VALUES

INSERT INTO Владелец\_автомобиля (Имя, Фамилия, Дата\_рождения, Адрес, Номер\_телефона)

VALUES

INSERT INTO Водитель (Имя, Фамилия, Дата\_рождения, Адрес, Номер\_телефона, Автомобиль\_ID)

VALUES

INSERT INTO Нарушение\_ПДД (Описание\_нарушени, Дата\_и\_время\_нарушения, Место\_нарушения, Водитель\_ID)

VALUES

INSERT INTO Штраф (Сумма\_штрафа, Дата\_оплаты\_штрафа, Нарушение\_ПДД\_ID)

VALUES

INSERT INTO Страховая\_стоимость\_автомобиля (Сумма\_страхования, Дата\_начала\_страхования, Дата\_окончания\_страхования)

VALUES

Следующий запрос выводит информацию о суммарной оплате каждого водителя автомобиля за каждый штраф, включая название автомобиля, год выпуска, имя и фамилию водителя, а также суммарную оплату.

SELECT Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска, SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

GROUP BY Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска;

Данный запрос дополняет предыдущий запрос сортировкой списка по стоимости убывания штрафов.

SELECT Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска, SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

GROUP BY Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска

ORDER BY Суммарная\_оплата DESC;

Данный запрос дополняет предыдущий запрос средней суммой штрафа.

SELECT AVG(Суммарная\_оплата) AS Средняя\_оплата

FROM (

SELECT SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

GROUP BY Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска

) AS Суммы\_штрафов;

Данный запрос дополняет предыдущий запрос дополнением колонки с наибольшей суммой штрафа с помощью использования подзапроса в SELECT-выражении.

SELECT AVG(Суммарная\_оплата) AS Средняя\_оплата, MAX(Суммарная\_оплата) AS Максимальная\_оплата

FROM (

SELECT SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата, MAX(Штраф.Сумма\_штрафа) AS Максимальный\_штраф

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

GROUP BY Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска

) AS Суммы\_штрафов;

Этот запрос выводит список водителей автомобилей с максимальным штрафом.

SELECT Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска, SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

WHERE Штраф.Сумма\_штрафа = (

SELECT MAX(Штраф.Сумма\_штрафа)

FROM Штраф

)

GROUP BY Водитель.Имя,

Водитель.Фамилия,

Автомобиль.Марка,

Автомобиль.Модель,

Автомобиль.Год\_выпуска;

Выводим список водителей автомобиля, у которых сумма штрафа превышает среднюю сумму штрафов.

SELECT Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска, SUM(Штраф.Сумма\_штрафа) AS Суммарная\_оплата

FROM Водитель

INNER JOIN Автомобиль ON Водитель.Автомобиль\_ID = Автомобиль.Автомобиль\_ID

INNER JOIN "Нарушение ПДД" ON Водитель.Водитель\_ID = "Нарушение ПДД".Водитель\_ID

INNER JOIN Штраф ON "Нарушение ПДД".Нарушение\_ПДД\_ID = Штраф.Нарушение\_ПДД\_ID

GROUP BY Водитель.Имя, Водитель.Фамилия, Автомобиль.Марка, Автомобиль.Модель, Автомобиль.Год\_выпуска

HAVING SUM(Штраф.Сумма\_штрафа) > (

SELECT AVG(Сумма\_штрафа)

FROM (

SELECT SUM(Штраф.Сумма\_штрафа) AS Сумма\_штрафа

FROM Штраф

GROUP BY "Нарушение\_ПДД\_ID"

) AS Средняя\_сумма\_штрафов

);

Здесь мы используем подзапрос, чтобы вычислить среднюю сумму штрафов для каждого нарушения ПДД, а затем находим среднее значение для всех нарушений и сравниваем его с суммой штрафов для каждого водителя. Если сумма штрафов больше среднего значения, то водитель попадает в результаты запроса.

Вывод: в данной работе были изучен синтаксис и основные команды для формирования запросов к базе данных, освоен процесс формирования SQL-запросов.

**Контрольные вопросы:**

1. Что такое подзапрос?

Подзапрос — это запрос, который является частью другого запроса в базе данных. Он используется для получения данных из одной или нескольких таблиц, которые затем могут быть использованы в основном запросе. Подзапросы позволяют более гибко и эффективно извлекать данные из базы данных, особенно в случаях, когда требуется выполнить сложные операции или комбинировать результаты из разных таблиц.

1. Как заполнить таблицу данными?

Чтобы заполнить таблицу данными в SQL, можно использовать оператор INSERT INTO. Нужно указать название таблицы и столбцов, в которые необходимо вставить данные, а затем указать значения для каждого столбца в соответствующем порядке. Если нужно вставить несколько строк данных, можно продолжить добавлять блоки VALUES для каждой строки.

1. Агрегатные функции. Основные сведения.

Агрегатные функции являются одной из основных групп функций в базах данных и языках запросов. Они используются для выполнения вычислений на наборе значений и возвращают единственное значение в результате. Основная цель агрегатных функций - анализировать данные путем суммирования, подсчета, нахождения среднего значения, нахождения минимального или максимального значения и других операций с набором значений. Агрегатные функции очень полезны при анализе и обработке данных в базах данных. Они позволяют суммировать, подсчитывать, находить среднее значение, минимум или максимум в больших наборах данных, что упрощает анализ информации и принятие решений на основе этих данных.

1. Как объединить таблицы в запросе?

В SQL для объединения таблиц в запросе используется оператор JOIN.

1. Способы объединения таблиц. Краткая характеристика.

Существует несколько видов JOIN:

* INNER JOIN: возвращает только те строки, которые имеют соответствие в обеих таблицах.
* LEFT JOIN: возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если в правой таблице нет соответствия, то значения будут заполнены NULL.
* RIGHT JOIN: возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если в левой таблице нет соответствия, то значения будут заполнены NULL.
* FULL JOIN: возвращает все строки из обеих таблиц. Если нет соответствия, то значения будут заполнены NULL.

1. План запроса. Краткое описание.

Общий план, который может быть адаптирован под конкретные требования задачи:

1. Определение цели: определить, что именно требуется выполнить с базой данных, например, выборка данных, добавление новых записей, обновление или удаление существующих записей.
2. Выбор таблицы: указать, с какой таблицей вы будете работать в запросе. Может быть одна или несколько таблиц, в зависимости от ваших требований.
3. Условия выборки: если требуется выбрать определенные записи из таблицы, задать условия фильтрации. Например, выбрать записи, где значение в определенном столбце соответствует определенному условию.
4. Выбор столбцов: определить, какие столбцы данных выбрать из таблицы. Можно выбрать все столбцы или только определенные.
5. Сортировка результатов: если требуется отсортировать результаты по определенному столбцу, указать критерий сортировки.
6. Группировка данных (по необходимости): если надо сгруппировать данные по определенному столбцу, использовать оператор GROUP BY.
7. Агрегатные функции (по необходимости): если требуется получить сумму, среднее значение, максимальное или минимальное значение и т. д., использовать агрегатные функции, такие как SUM, AVG, MAX, MIN.
8. Подзапросы (по необходимости): если требуется выполнить запрос внутри другого запроса, использовать подзапросы.
9. Объединение таблиц (по необходимости): если требуется объединить данные из нескольких таблиц, использовать операторы JOIN.
10. Ограничение количества записей (при необходимости): если требуется ограничить количество возвращаемых записей, использовать операторы LIMIT (в MySQL) или TOP (в SQL Server).
11. Выполнение запроса: после определения всех параметров запроса выполнить его и получить результат.
12. Оптимизация запроса. Краткое описание.

Оптимизация запросов в SQL — это процесс улучшения производительности выполнения запросов к базе данных. Оптимизация запросов помогает ускорить время выполнения запросов, уменьшить потребление ресурсов и повысить общую эффективность работы с базой данных.

Для оптимизации запросов в SQL можно использовать следующие методы:

* Индексы: Создание соответствующих индексов на столбцах, используемых в условиях JOIN, WHERE и ORDER BY, позволяет ускорить выполнение запросов. Индексы помогают базе данных быстро находить нужные записи.
* Правильное использование JOIN: когда объединяется несколько таблиц, лучше использовать подходящий тип объединения (INNER JOIN, LEFT JOIN, RIGHT JOIN) в зависимости от требуемого результата. Также важно указывать правильные условия соединения, чтобы уменьшить количество извлекаемых записей.
* Ограничение вывода: если нужно получить только несколько строк из большой таблицы, указать это с помощью операторов LIMIT или TOP, чтобы избежать излишнего извлечения и обработки данных.
* Избегать выборки неиспользуемых столбцов: указывать только те столбцы, которые действительно нужны. Избыточные столбцы могут замедлить выполнение запроса.
* Предварительное вычисление выражений: если есть сложные выражения или функции, которые должны быть вычислены для каждой строки, рассмотрите возможность вычисления их заранее и сохранения результатов во временной таблице или представлении.
* Использовать подзапросы с умом: подзапросы могут быть мощным инструментом, но они также могут замедлить выполнение запроса, особенно если они используются неправильно или вложены в циклы.

1. Оконные функции. Краткое описание.

Оконные функции являются частью операции агрегации данных в SQL. Они позволяют выполнять вычисления над набором строк, который определен с помощью окна, которое представляет собой подмножество данных, сгруппированных по определенным критериям. Оконные функции могут использоваться для вычисления различных статистических показателей и применяются к каждой строке в окне. Они позволяют выполнять расчеты, такие как сумма, среднее, минимум, максимум, ранжирование, накопление и т.д., и могут быть полезны для анализа данных и создания отчетов.

1. Команда и синтаксис для группировки данных.

В SQL для группировки данных используется команда GROUP BY. Она позволяет группировать данные по одному или нескольким столбцам и выполнять агрегатные функции (например, суммирование, подсчет количества, нахождение среднего и т.д.) для каждой группы.

Синтаксис команды GROUP BY выглядит следующим образом:

SELECT column1, column2, ..., aggregate\_function(column)

FROM table

GROUP BY column1, column2, ...

1. Сортировка данных. Краткое описание.

Сортировка данных в SQL является одной из основных операций. Она позволяет упорядочить результаты запроса по заданному столбцу или нескольким столбцам. Для сортировки данных в SQL используется оператор ORDER BY. Данный оператор указывается после блока SELECT и указывает по какому столбцу нужно проводить сортировку. Можно указывать как один, так и несколько столбцов для сортировки. По умолчанию, сортировка происходит по возрастанию. Если нужна сортировка по убыванию, используйте ключевое слово DESC после имени столбца.

1. Условия для выборки данных (WHERE, HAVING). Краткое описание.

Оба оператора WHERE и HAVING могут использоваться в SELECT-запросах для уточнения результатов выборки, в соответствии с указанными условиями. WHERE применяется перед группировкой и агрегатными функциями, а HAVING - после них. WHERE используется для фильтрации строк на основе условий, которые должны быть истинны. Он позволяет указать условия, которым должны соответствовать значения столбцов. HAVING используется для фильтрации результатов группировки по условию, которое должно быть истинным. Он позволяет указать условия для сгруппированных значений.