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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

По дисциплине: «Системы хранения и обработки данных»

Тема: «Формирование запросов к базе данных»

Выполнил работу студент группы мИИВТ-231 Костюков К.А.

Принял: Короленко В.В.

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Воронеж 2023

**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.

**Ход работы:**

После изучения синтаксиса SQL-запросов в СУБД Postgres, основ формирования SQL-запросов и способов оптимизации запросов, были сформированы SQL-запросы для создания таблиц.

Тренировочное задание:

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)

);

Индивидуальное задание:

CREATE TABLE Абитуриент (

Абитуриент\_ID SERIAL PRIMARY KEY,

Имя VARCHAR(50),

Фамилия VARCHAR(50),

Дата\_рождения DATE,

Адрес VARCHAR(100)

);

CREATE TABLE Экзаменатор (

Экзаменатор\_ID SERIAL PRIMARY KEY,

Имя VARCHAR(50),

Фамилия VARCHAR(50),

Специализация VARCHAR(50),

Уровень\_оплаты DECIMAL(10, 2)

);

CREATE TABLE Экзамен (

Экзамен\_ID SERIAL PRIMARY KEY,

Название VARCHAR(100),

Дата\_проведения DATE,

Аудитория VARCHAR(50)

);

CREATE TABLE Результат\_экзамена (

Результат\_ID SERIAL PRIMARY KEY,

Оценка INTEGER,

Дата\_сдачи DATE,

Абитуриент\_ID INTEGER REFERENCES Абитуриент(Абитуриент\_ID),

Экзамен\_ID INTEGER REFERENCES Экзамен(Экзамен\_ID),

Экзаменатор\_ID INTEGER REFERENCES Экзаменатор(Экзаменатор\_ID)

);

CREATE TABLE Оплата\_экзаменатора (

Оплата\_ID SERIAL PRIMARY KEY,

Сумма\_оплаты DECIMAL(10, 2),

Дата\_оплаты DATE,

Экзаменатор\_ID INTEGER REFERENCES Экзаменатор(Экзаменатор\_ID)

);

Далее необходимо было 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.

Тренировочное задание:

INSERT INTO Customers (FirstName, LastName, Email)

VALUES

INSERT INTO Orders (CustomerID, OrderDate, TotalAmount)

VALUES

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

VALUES

Индивидуальное задание:

INSERT INTO Абитуриент (Имя, Фамилия, Дата\_рождения, Адрес)

VALUES

INSERT INTO Экзаменатор (Имя, Фамилия, Специализация, Уровень\_оплаты)

VALUES

INSERT INTO Экзамен (Название, Дата\_проведения, Аудитория)

VALUES

INSERT INTO Результат\_экзамена (Оценка, Дата\_сдачи, Абитуриент\_ID, Экзамен\_ID, Экзаменатор\_ID)

VALUES

INSERT INTO Оплата\_экзаменатора (Сумма\_оплаты, Дата\_оплаты, Экзаменатор\_ID)

VALUES

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

Тренировочное задание:

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;

Индивидуальное задание:

Данный запрос выводит информацию о суммарной оплате каждого экзаменатора за каждый экзамен, включая название экзамена, имя и фамилию экзаменатора, а также суммарную оплату.

SELECT Экзамен.Название AS Название\_экзамена, Экзаменатор.Имя AS Имя\_экзаменатора, Экзаменатор.Фамилия AS Фамилия\_экзаменатора,

SUM(Оплата\_экзаменатора.Сумма\_оплаты) AS Суммарная\_оплата

FROM Экзамен

JOIN Результат\_экзамена ON Экзамен.Экзамен\_ID = Результат\_экзамена.Экзамен\_ID

JOIN Экзаменатор ON Результат\_экзамена.Экзаменатор\_ID = Экзаменатор.Экзаменатор\_ID

JOIN Оплата\_экзаменатора ON Экзаменатор.Экзаменатор\_ID = Оплата\_экзаменатора.Экзаменатор\_ID

GROUP BY Экзамен.Экзамен\_ID, Экзаменатор.Экзаменатор\_ID;

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

Тренировочное задание:

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;

Индивидуальное задание:

Данный запрос дополняет предыдущий запрос сортировкой экзаменов по алфавиту.

SELECT Экзамен.Название AS Название\_экзамена, Экзаменатор.Имя AS Имя\_экзаменатора, Экзаменатор.Фамилия AS Фамилия\_экзаменатора,

SUM(Оплата\_экзаменатора.Сумма\_оплаты) AS Суммарная\_оплата

FROM Экзамен

JOIN Результат\_экзамена ON Экзамен.Экзамен\_ID = Результат\_экзамена.Экзамен\_ID

JOIN Экзаменатор ON Результат\_экзамена.Экзаменатор\_ID = Экзаменатор.Экзаменатор\_ID

JOIN Оплата\_экзаменатора ON Экзаменатор.Экзаменатор\_ID = Оплата\_экзаменатора.Экзаменатор\_ID

GROUP BY Экзамен.Экзамен\_ID, Экзаменатор.Экзаменатор\_ID

ORDER BY Экзамен.Экзамен\_ID;

После был добавлен столбец со средней суммарной стоимостью заказов.

Тренировочное задание:

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;

Индивидуальное задание:

Данный запрос дополняет предыдущий запрос средним результатом всех сдающих.

SELECT Экзамен.Название AS Название\_экзамена,

AVG(Результат\_экзамена.Оценка) AS Средняя\_оценка, Экзаменатор.Имя AS Имя\_экзаменатора, Экзаменатор.Фамилия AS Фамилия\_экзаменатора,

SUM(Оплата\_экзаменатора.Сумма\_оплаты) AS Суммарная\_оплата

FROM Экзамен

JOIN Результат\_экзамена ON Экзамен.Экзамен\_ID = Результат\_экзамена.Экзамен\_ID

JOIN Экзаменатор ON Результат\_экзамена.Экзаменатор\_ID = Экзаменатор.Экзаменатор\_ID

JOIN Оплата\_экзаменатора ON Экзаменатор.Экзаменатор\_ID = Оплата\_экзаменатора.Экзаменатор\_ID

GROUP BY Экзамен.Экзамен\_ID, Экзаменатор.Экзаменатор\_ID

ORDER BY Экзамен.Экзамен\_ID;

Затем, был выведен клиент с наибольшей суммарной стоимостью заказов.

Тренировочное задание:

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;

Индивидуальное задание:

Данный запрос дополняет предыдущий запрос дополнением колонки с наибольшим суммарным баллом за экзамены.

SELECT Экзамен.Название AS Название\_экзамена, MAX(Результат\_экзамена.Оценка) AS Максимальная\_оценка,

AVG(Результат\_экзамена.Оценка) AS Средняя\_оценка, Экзаменатор.Имя AS Имя\_экзаменатора, Экзаменатор.Фамилия AS Фамилия\_экзаменатора,

SUM(Оплата\_экзаменатора.Сумма\_оплаты) AS Суммарная\_оплата

FROM Экзамен

JOIN Результат\_экзамена ON Экзамен.Экзамен\_ID = Результат\_экзамена.Экзамен\_ID

JOIN Экзаменатор ON Результат\_экзамена.Экзаменатор\_ID = Экзаменатор.Экзаменатор\_ID

JOIN Оплата\_экзаменатора ON Экзаменатор.Экзаменатор\_ID = Оплата\_экзаменатора.Экзаменатор\_ID

GROUP BY Экзамен.Экзамен\_ID, Экзаменатор.Экзаменатор\_ID

ORDER BY Экзамен.Экзамен\_ID;

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

Тренировочное задание:

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;

Индивидуальное задание:

Этот запрос выводит список экзаменов и экзаменаторов с максимальным баллом.

SELECT Экзамен.Название AS Название\_экзамена,

MAX(Результат\_экзамена.Оценка) AS Максимальная\_оценка, Экзаменатор.Имя AS Имя\_экзаменатора, Экзаменатор.Фамилия AS Фамилия\_экзаменатора,

SUM(Оплата\_экзаменатора.Сумма\_оплаты) AS Суммарная\_оплата

FROM Экзамен

JOIN Результат\_экзамена ON Экзамен.Экзамен\_ID = Результат\_экзамена.Экзамен\_ID

JOIN Экзаменатор ON Результат\_экзамена.Экзаменатор\_ID = Экзаменатор.Экзаменатор\_ID

JOIN Оплата\_экзаменатора ON Экзаменатор.Экзаменатор\_ID = Оплата\_экзаменатора.Экзаменатор\_ID

WHERE (Экзамен.Экзамен\_ID, Результат\_экзамена.Оценка) IN (SELECT Экзамен\_ID, MAX(Оценка) AS Максимальная\_оценка

FROM Результат\_экзамена GROUP BY Экзамен\_ID)

GROUP BY Экзамен.Экзамен\_ID, Экзаменатор.Экзаменатор\_ID

ORDER BY Экзамен.Экзамен\_ID;

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

Тренировочное задание:

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:

Список абитуриентов, у которых суммарный балл превышает средний балл всех сдающих.

SELECT Абитуриент.Абитуриент\_ID, Абитуриент.Имя, Абитуриент.Фамилия,

SUM(Результат\_экзамена.Оценка) AS Суммарный\_балл

FROM Абитуриент

JOIN Результат\_экзамена ON Абитуриент.Абитуриент\_ID = Результат\_экзамена.Абитуриент\_ID

GROUP BY Абитуриент.Абитуриент\_ID HAVING SUM(Результат\_экзамена.Оценка) > (SELECT AVG(Оценка) FROM Результат\_экзамена);

**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 группированных значений.