МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ 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 Ахлестин А.И. (подпись) Фамилия, инициалы

Принял \_\_\_\_\_\_\_\_\_\_\_\_Короленко В.В.  
 (подпись) Фамилия, инициалы

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

Воронеж 2023

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

Задание на лабораторную работу:

1. Разработать структуру базы данных в СУБД Postgres;
2. Наполнить базу данных данными;
3. Написать ряд запросов к базе данных для отработки навыка формирования SQL-запросов;
4. Научиться ставить задачу по формированию выборки необходимых данных из базы данных и решать её с помощью SQL-запросов.

Ход выполнения задания:

# 1 Разработать структуру базы данных в СУБД Postgres

# Изучить синтаксис SQL-запросов в СУБД Postgres

# Изучение синтаксиса SQL-запросов в СУБД PostgreSQL предполагает освоение основных конструкций языка структурированных запросов. Ниже представлен обзор базовых элементов синтаксиса SQL в контексте PostgreSQL:

# Создание таблицы:

# CREATE TABLE table\_name (

# column1 datatype1,

# column2 datatype2,

# ...

# );

# Вставка данных в таблицу:

# INSERT INTO table\_name (column1, column2, ...)

# VALUES (value1, value2, ...);

# Выборка данных:

# SELECT column1, column2, ...

# FROM table\_name

# WHERE condition;

# Обновление данных:

# UPDATE table\_name

# SET column1 = value1, column2 = value2, ...

# WHERE condition;

# Удаление данных:

# DELETE FROM table\_name

# WHERE condition;

# Создание индекса:

# CREATE INDEX index\_name

# ON table\_name (column1, column2, ...);

# Группировка и агрегация:

# SELECT column1, COUNT(\*)

# FROM table\_name

# GROUP BY column1;

# Объединение таблиц:

# SELECT column1, column2, ...

# FROM table1

# INNER JOIN table2 ON table1.column = table2.column;

# Сортировка результатов:

# SELECT column1, column2, ...

# FROM table\_name

# ORDER BY column1 ASC|DESC, column2 ASC|DESC, ...;

# Изучить основы формирования SQL-запросов

# Приведем примеры формирования запросов основываясь на задании из лабораторной работы номер 7.

# Чтобы получить всех пациентов, которые посетили врача с определенной специальностью, нам необходимо выполнить следующий подзапрос в условии WHERE:

# SELECT \*

# FROM Patients

# WHERE Patient\_ID IN (SELECT Patient\_ID FROM Appointments WHERE Doctor\_ID IN (SELECT Doctor\_ID FROM Doctors WHERE Specialty = 'Cardiologist'));

# Мы также можем выполнить подзапрос в операторе FROM и получить средний возраст пациентов, записанных на приём:

# SELECT AVG(Age)

# FROM (SELECT EXTRACT(YEAR FROM AGE(Appointment\_Date, Birth\_Date)) AS Age FROM Patients) AS AgeTable;

# При помощи агрегатной функции мы можем получить список всех пациентов из таблицы:

# SELECT COUNT(\*)

# FROM Patients;

# Также, при помощи агрегации и группировки, мы можем получить количество приемов для каждого врача:

# SELECT Doctor\_ID, COUNT(\*) AS AppointmentCount

# FROM Appointments

# GROUP BY Doctor\_ID;

# Чтобы объединить таблицы, можно использовать INNER JOIN или LEFT JOIN. Они отличаются по возвращаемым строкам. INNER JOINТ Возвращает только те строки, для которых существуют соответствующие значения в обеих таблицах. LEFT JOIN возвращает все строки из левой таблицы (FROM), и соответствующие строки из правой таблицы (JOIN).

# Рассмотрим пример использования INNER JOIN, чтобы получить информацию о приемах и соответствующих пациентах:

# SELECT Appointments.Appointment\_ID, Appointments.Doctor\_ID, Appointments.Patient\_ID, Patients.Last\_Name, Patients.First\_Name

# FROM Appointments

# INNER JOIN Patients ON Appointments.Patient\_ID = Patients.Patient\_ID;

# При помощи LEFT JOIN получим всех врачей и информацию о приемах, если они есть:

# SELECT Doctors.Doctor\_ID, Doctors.Full\_Name, Appointments.Appointment\_ID, Appointments.Appointment\_Date

# FROM Doctors

# LEFT JOIN Appointments ON Doctors.Doctor\_ID = Appointments.Doctor\_ID;

# Способы оптимизации запросов, план запроса, оконные функции

# Способы оптимизации запросов в SQL:

# Индексы: Создание индексов на столбцах, используемых в условиях WHERE и JOIN, может значительно ускорить выполнение запросов.

# Правильное использование инструкций: Использование правильных инструкций SQL, таких как JOIN, вместо подзапросов, может улучшить производительность.

# Оптимизация запроса: Рефакторинг запроса, уменьшение объема данных, которые нужно извлечь, может сократить время выполнения запроса.

# Избегание использования "SELECT ": Выбор только необходимых столбцов, а не всех, может уменьшить объем передаваемых данных.

# План запроса:

# План выполнения запроса (query execution plan): Это план, предоставляемый СУБД, который описывает порядок, в котором будут выполнены различные операции, чтобы получить результат запроса. Может включать в себя сканирование таблиц, использование индексов, объединения и сортировку данных.

# Оконные функции:

# Оконные функции (window functions): Это функции, которые выполняются в пределах "окна" результатов запроса, определенного определенными условиями, такими как OVER (PARTITION BY ...). Они могут использоваться для вычисления агрегатов или аналитических функций, применяемых к определенным частям результирующего набора. Примеры оконных функций: ROW\_NUMBER(), SUM(), AVG(), и т. д.

# Сформировать SQL-запросы для создания таблиц в СУБД Postgres

# -- Таблица Customers

# CREATE TABLE Customers (

# CustomerID SERIAL PRIMARY KEY,

# FirstName VARCHAR(255),

# LastName VARCHAR(255),

# Email VARCHAR(255)

# );

# -- Таблица Orders

# CREATE TABLE Orders (

# OrderID SERIAL PRIMARY KEY,

# CustomerID INTEGER REFERENCES Customers(CustomerID),

# OrderDate TIMESTAMP,

# TotalAmount DECIMAL(10, 2)

# );

# -- Таблица OrderDetails

# CREATE TABLE OrderDetails (

# OrderDetailID SERIAL PRIMARY KEY,

# OrderID INTEGER REFERENCES Orders(OrderID),

# ProductID INTEGER,

# Quantity INTEGER,

# UnitPrice DECIMAL(10, 2)

# );

# Отредактируем файл docker-compose.yml, изменим имя базы данных и проверим правильность путей для скрипта инициализации. Отредактированный файл отображен на рисунке 1.

# 

# Рисунок 1 – Файл docker-compose.yml

# Далее откроем файл скриптов sql и внесем туда данные запросы. Итоговый файл изображен на рисунке 2.

# 

# Рисунок 2 – Файл init.sql

# 2 Наполнить базу данных данными

# Для начала запустим базу данных через контейнер, как делали это в предыдущих лабораторных работах и подключимся к ней, как это показано на рисунке 3.

# 

# Рисунок 3 – Запуск и подключение к бд

# После входа в базу данных, мы можем выполнять sql команды. Выполним команду INSERT, чтобы заполнить таблицу Customers данными. Выполнение команды отображено на рисунке 4.

# 

# Рисунок 4 – Заполнение таблицы данными

# Теперь проверим, внеслись ли изменения в таблицу. Для этого введем команду SELECT \* FROM Customers;. Выполнение показано на рисунке 5.

# 

# Рисунок 5 – Успешно внесенные данные

# Выполним заполнение таблиц Orders и OrderDetails аналогичным образом. Заполнение таблицы для Orders отображено на рисунке 5. Для OrderDeatils на рисунке 6.

# 

# Рисунок 5 – Заполнение таблицы Orders

# 

# Рисунок 6 – Заполнение таблицы OrderDetails

# 3 Написать ряд запросов к базе данных для отработки навыка формирования SQL-запросов

# 3.1 Сформировать SQL-запросы

# Составим и выполним запрос. Используем SELECT для выбора столбцов с именем и фамилией клиента, а также суммарной стоимости заказов. Используем COALESCE для предотвращение вывода нулевых значений, если заказы будут отсутствовать в таблице. Оператор FROM указывает, откуда берутся данные. В данном случае используется таблица Customers (переименованная как c) и происходит объединение с таблицей Orders (переименованной как o) через LEFT JOIN, чтобы включить все клиентов, даже если у них нет заказов. GROUP BY группирует строки по именам и фамилиям клиентов, что позволяет применять агрегатные функции к каждой группе. ORDER BY сортирует результаты по фамилии и имени клиента. Итоговый запрос отображен на рисунке 7. Результат вывода запроса отображен на рисунке 8.

# 

# Рисунок 7 – Запрос согласно заданию

# 

# Рисунок 8 – Результат работы запроса

# Отсортируем полученный список по убыванию суммарной стоимости заказов клиента, для этого используем ORDER BY с параметром DESC. Получивший запрос:

# 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.FirstName, c.LastName

# ORDER BY

# TotalOrderAmount DESC, c.LastName, c.FirstName;

# Результат работы запроса представлен на рисунке 8.

# 

# Рисунок 8 – Сортировка по убыванию

# Теперь добавим столбец со средней суммарной стоимостью всех заказов для каждого покупателя. Запрос будет выглядеть следующим образом:

# SELECT

# c.FirstName,

# c.LastName,

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

# COALESCE(ROUND(AVG(o.TotalAmount), 2), 0) AS AvgOrderAmount

# FROM

# Customers c

# LEFT JOIN

# Orders o ON c.CustomerID = o.CustomerID

# GROUP BY

# c.FirstName, c.LastName

# ORDER BY

# TotalOrderAmount DESC, c.LastName, c.FirstName;

# Результат запроса отображен на рисунке 9.

# 

# Рисунок 9 – Средняя стоимость заказов

# Теперь выведем клиента с наибольшей суммарной стоимостью заказов, для этого воспользуемся оконной функцией ROW NUMBER для нумерования клиентов в порядке убывания суммарной стоимости заказов и выбора первой записи. Итоговый запрос:

# WITH RankedCustomers AS (

# SELECT

# c.FirstName,

# c.LastName,

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

# ROW\_NUMBER() OVER (ORDER BY COALESCE(SUM(o.TotalAmount), 0) DESC) AS Rank

# FROM

# Customers c

# LEFT JOIN

# Orders o ON c.CustomerID = o.CustomerID

# GROUP BY

# c.FirstName, c.LastName

# )

# SELECT

# FirstName,

# LastName,

# TotalOrderAmount

# FROM

# RankedCustomers

# WHERE

# Rank = 1;

# Результат выполнения запроса отображен на рисунке 10.

# 

# Рисунок 10 – Выполнение запроса

# Теперь сделаем запрос, который выведет список заказов клиента с наибольшей суммарной стоимостью заказов.

# Чтобы вывести для клиента список его заказов в порядке возрастания, мы можем воспользоваться функцией RANK, чтобы пронумеровать заказы внутри клиента. Итоговый запрос:

# WITH RankedOrders AS (

# SELECT

# c.CustomerID,

# c.FirstName,

# c.LastName,

# o.OrderID,

# o.TotalAmount,

# RANK() OVER (PARTITION BY c.CustomerID ORDER BY o.TotalAmount) AS OrderRank

# FROM

# Customers c

# LEFT JOIN

# Orders o ON c.CustomerID = o.CustomerID

# ),

# TopCustomers 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

# ORDER BY

# TotalOrderAmount DESC

# LIMIT 1

# )

# SELECT

# ro.CustomerID,

# ro.FirstName,

# ro.LastName,

# ro.OrderID,

# ro.TotalAmount

# FROM

# TopCustomers tc

# JOIN

# RankedOrders ro ON tc.CustomerID = ro.CustomerID

# ORDER BY

# ro.TotalAmount;

# Результат запроса показан на рисунке 11.

# 

# Рисунок 11 – Результат запроса

# Теперь выведем клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость всех заказов клиентов.

# Для этого создадим такой запрос:

# 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

# ),

# AverageOrderAmount AS (

# SELECT

# AVG(TotalOrderAmount) AS AvgOrderAmount

# FROM

# CustomerOrderTotals

# )

# SELECT

# cot.CustomerID,

# cot.FirstName,

# cot.LastName,

# cot.TotalOrderAmount,

# aoa.AvgOrderAmount

# FROM

# CustomerOrderTotals cot

# JOIN

# AverageOrderAmount aoa ON 1 = 1

# WHERE

# cot.TotalOrderAmount > aoa.AvgOrderAmount

# ORDER BY

# cot.TotalOrderAmount DESC, cot.LastName, cot.FirstName;

# В данном запросе сначала вычисляет суммарная стоимость заказов для каждого клиента, потом определяется средняя стоимость заказов всех клиентов. Затем запрос выбирает только тех, у кого суммарная стоимость заказов превышает среднюю стоимость заказа. Результат выполнения запроса показан на рисунке 12.

# 

# Рисунок 12 – Результат выполнения запроса

# 3.2 Сохранение запросов в init файл

# Сохраним все запросы из пунктов 2 и 3.1 в файл init.sql. Получившийся файл отображен на рисунке 13.

# 

# Рисунок 13 – файл init\_lr9\_1.sql

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

# 4.1 Сформулировать задачу, аналогичную изложенной выше

# Разработаем задачу, которую необходимо будет решить для базы данных созданной в результате выполнения лабораторной работы номер 8.

# Была разработана база данных медицинских приёмов в поликлинике, поэтому выведем Id приёма, дату приёма пациента, стоимость приёма, данные о пациенте, принимающего врача и специальность врача.

# Создадим для этого запрос:

# SELECT

# R.Receipt\_ID,

# A.Appointment\_Date,

# A.Cost,

# P.Last\_Name AS Patient\_Last\_Name,

# P.First\_Name AS Patient\_First\_Name,

# D.Full\_Name AS Doctor\_Name,

# D.Specialty

# FROM

# Receipts R

# JOIN

# Appointments A ON R.Appointment\_ID = A.Appointment\_ID

# JOIN

# Doctors D ON A.Doctor\_ID = D.Doctor\_ID

# JOIN

# Patients P ON A.Patient\_ID = P.Patient\_ID;

# Результат выполнения запроса отображен на рисунке 14.

# 

# Рисунок 14 – Выполнение задания по запросу

Вывод: в процессе выполнения работы было освоено формирование SQL-запросов и проектирование физической структуры базы данных.

Контрольные вопросы:

1. Что такое подзапрос?

Подзапрос - это запрос, вложенный в другой запрос. Он выполняется внутри основного запроса и может возвращать набор данных, который затем используется в условиях, операторах SELECT, FROM, и других частях главного запроса.

1. Как заполнить таблицу данными?

Для заполнения таблицы данными используется оператор INSERT INTO.

1. Агрегатные функции. Основные сведения.

Агрегатные функции выполняют вычисления на наборе данных и возвращают единственное значение. Некоторые основные агрегатные функции включают COUNT (подсчет), SUM (сумма), AVG (среднее значение), MIN (минимум), MAX (максимум).

1. Как объединить таблицы в запросе?

Для объединения таблиц в запросе используется оператор JOIN. Он позволяет соединять строки из двух или более таблиц на основе определенного условия, обычно связи между столбцами в этих таблицах.

1. Способы объединения таблиц. Краткая характеристика.?

Основные способы объединения таблиц: INNER JOIN (внутреннее объединение, возвращает строки, соответствующие условию), LEFT JOIN (левое объединение, возвращает все строки из левой таблицы и соответствующие строки из правой), RIGHT JOIN (правое объединение, возвращает все строки из правой таблицы и соответствующие строки из левой), FULL JOIN (полное объединение, возвращает строки, если они соответствуют условию в любой из таблиц).

1. План запроса. Краткое описание?

План запроса - это последовательность операций, которую база данных выполняет для выполнения запроса. Он включает в себя информацию о том, какие индексы будут использованы, какие операции объединения или сортировки будут выполнены, и т.д. План запроса помогает оптимизировать выполнение запросов.

1. Оптимизация запроса. Краткое описание.

Оптимизация запроса - это процесс улучшения производительности запроса путем выбора эффективных индексов, переписывания запроса, уменьшения объема данных, которые нужно обработать, и других методов. Цель - ускорить выполнение запроса и снизить нагрузку на базу данных.

1. Оконные функции. Краткое описание.

Оконные функции (window functions) в SQL позволяют выполнять вычисления на "окне" или диапазоне строк, связанных с текущей строкой. Примеры включают ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), и другие.

1. Команда и синтаксис для группировки данных.

SELECT column1, column2, aggregate\_function(column3)

FROM table

GROUP BY column1, column2;

1. Сортировка данных. Краткое описание.

Сортировка данных в SQL выполняется с использованием оператора ORDER BY. Он указывает порядок сортировки для результатов запроса.

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

Условия выборки данных в SQL определяются с использованием операторов WHERE (для фильтрации строк при запросе) и HAVING (для фильтрации групп при использовании GROUP BY). Условия WHERE применяются перед группировкой (если она есть), а HAVING - после группировки.