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: «Развёртывание СУБД Postgres с использованием средств автоматизации развёртывания и управления приложениями»

Выполнил работу студент группы мИИВТ-241: Ульянов А.Г.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Задачи лабораторной работы:

• 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.

1. Для создания таблиц в СУБД PostgreSQL в соответствии с указанной структурой, можно использовать следующие SQL-запросы.

SQL-запросы для создания таблиц

-- Создание таблицы Customers

CREATE TABLE IF NOT EXISTS Customers (

CustomerID SERIAL PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL клиента

);

-- Создание таблицы Orders

CREATE TABLE IF NOT EXISTS Orders (

OrderID SERIAL PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

TotalAmount DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE

);

1. IF NOT EXISTS – Проверяет существуют ли таблицы с данным названием, если таблица существует, SQL не выполняет команду создания.
2. FOREIGN KEY (CustomerID): Эта часть указывает, что столбец CustomerID в текущей таблице является внешним ключом. Внешний ключ используется для создания связи между двумя таблицами.
3. REFERENCES Customers(CustomerID): Эта часть указывает, что внешний ключ CustomerID ссылается на столбец CustomerID в таблице Customers. Это означает, что значения в столбце CustomerID текущей таблицы должны соответствовать значениям в столбце CustomerID таблицы Customers.
4. ON DELETE CASCADE: Эта часть определяет поведение при удалении записей. В данном случае, если запись в таблице Customers с определенным CustomerID будет удалена, все записи в текущей таблице, которые ссылаются на этот CustomerID, также будут автоматически удалены. Это помогает поддерживать целостность данных.

-- Создание таблицы OrderDetails

CREATE TABLE IF NOT EXISTS OrderDetails (

OrderDetailID SERIAL PRIMARY KEY,

OrderID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL CHECK (Quantity > 0),

UnitPrice DECIMAL(10, 2) NOT NULL,

UnitPrice DECIMAL(10, 2): Эта часть указывает, что столбец UnitPrice будет хранить значения с плавающей запятой (тип данных DECIMAL). В данном случае, DECIMAL(10, 2) означает, что общее количество цифр, которые могут быть сохранены, составляет 10, из которых 2 цифры будут после десятичной точки.

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE CASCADE

);

2. Наполнение базы данных данными

Теперь давайте добавим данные в таблицы. Я приведу примеры вставки данных для каждой таблицы.

-- Наполнение таблицы Customers

INSERT INTO Customers (FirstName, LastName, Email) VALUES

('John', 'Doe', 'john.doe@example.com'),

('Jane', 'Smith', 'jane.smith@example.com'),

('Alice', 'Johnson', 'alice.johnson@example.com'),

('Bob', 'Brown', 'bob.brown@example.com'),

('Charlie', 'Davis', 'charlie.davis@example.com');

-- Наполнение таблицы Orders

INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES

(1, '2023-10-01 10:30:00', 25.00),

(2, '2023-10-02 12:15:00', 30.50),

(3, '2023-10-03 14:45:00', 15.00),

(4, '2023-10-04 09:00:00', 42.75),

(5, '2023-10-05 11:20:00', 22.50);

-- Наполнение таблицы OrderDetails

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

(1, 1, 1, 10.00),

(1, 2, 1, 15.00),

(2, 2, 2, 15.50),

(3, 1, 1, 10.00),

(4, 3, 1, 20.00),

(4, 4, 2, 5.75),

(5, 5, 3, 7.50);

1. SERIALиспользуется для автоматической генерации уникальных идентификаторов.

2) Для заполнения полей ФИО использовался сайт для генерации ФИО.



Рисунок 1 – Пример заполненной таблицы для Customers

Вывод: SELECT\*FROM customers

3. Выполнение запросов SQL.

1) Сформировать SQL-запрос, который возвращает список клиентов (имя и фамилия) и суммарную стоимость заказов каждого клиента (должны отобразиться столбцы с именем, фамилией и стоимостью).

Код:

SELECT Cus.FirstName, Cus.LastNAME, (SELECT sum(Ord.TotalAmount)

FROM Orders Ord

WHERE Ord.CustomerID = Cus.CustomerID) AS amount,

--Добавим среднюю стоимость

(SELECT AVG(Ord.TotalAmount) FROM Orders ord WHERE Ord.CustomerID = CUS.CustomerID) AS AVGSUM

FROM Customers Cus

Where Cus.CustomerID IN (SELECT DISTINCT CustomerID FROM Orders)

Order by amount DESC;

Объяснение:

1) Выбираем два поля: FirstName и LastNAME из таблицы Customers, которая имеет псевдоним Cus. Это означает, получим имена и фамилии клиентов.

2) Суммируем TotalAmount из таблицы Orders, которая имеет псевдоним Ord. Условие WHERE Ord.CustomerID = Cus.CustomerID связывает заказы с конкретным клиентом, позволяет получить общую сумму заказов для каждого клиента. Результат этого подзапроса будет иметь псевдоним amount.

3) Указываем, что основная таблица, из которой извлекаем данные, — это Customers, и даем ей псевдоним Cus.

4) Фильтруем клиентов, оставляя только тех, у кого есть хотя бы один заказ. Подзапрос SELECT DISTINCT CustomerID FROM Orders возвращает уникальные идентификаторы клиентов, которые сделали заказы. Условие IN проверяет, что CustomerID клиента находится в этом наборе.

5) Добавляем еще один подзапрос, который вычисляет среднюю стоимость заказов для каждого клиента. Условие WHERE Ord.CustomerID = CUS.CustomerID также связывает заказы с конкретным клиентом. Результат этого подзапроса будет иметь псевдоним AVGSUM.

6) Сортируем результаты по полю amount в порядке убывания. Это означает, что клиенты с наибольшей суммой заказов будут показаны первыми.



Рисунок 2 – Результат

2) Вывести клиента (список клиентов) с наибольшей суммарной стоимостью заказов (имя, фамилия, стоимость).

Код:

SELECT FirstName, LastNAME, sum(TotalAmount) as sumamount

FROM Customers JOIN Orders On customers.customerid = orders.customerid

GROUP BY FirstName, LastNAME

HAVING sum(TotalAmount) = (SELECT MAX(sumamount)

From (Select sum(totalAmount) AS sumamount FROM orders GROUP By CustomerID))

Order by sumamount DESC;

1) Выбираем поля: FirstName, LastNAME и сумму всех TotalAmount (сумма заказов) для каждого клиента. Сумма будет иметь псевдоним sumamount.

2) Указываем, что данные извлекаются из таблицы Customers, которая соединяется с таблицей Orders по полю customerid. Получаем информацию о клиентах вместе с их заказами.

3) Группируем результаты по FirstName и LastNAME. Это необходимо, чтобы суммировать значения TotalAmount для каждого уникального клиента.

4) Используем оператор HAVING, который фильтрует группы, созданные с помощью GROUP BY. Оставляем только те группы, для которых сумма заказов равна максимальной сумме заказов, вычисленной в подзапросе.

5) Вычисляем сумму для каждого клиента, группируем по CustomerID. Затем берем максимальную сумму из этих значений. Возвращаем одно значение, то есть максимальную сумму заказов среди всех клиентов.

6) Сортируем результаты по sumamount в порядке убывания.

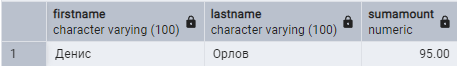


Рисунок 2 - Результат

3) Вывести список клиентов с наибольшей суммой стоимости заказов СПИСОК ЗАКАЗОВ (нормер заказа и стоимость).

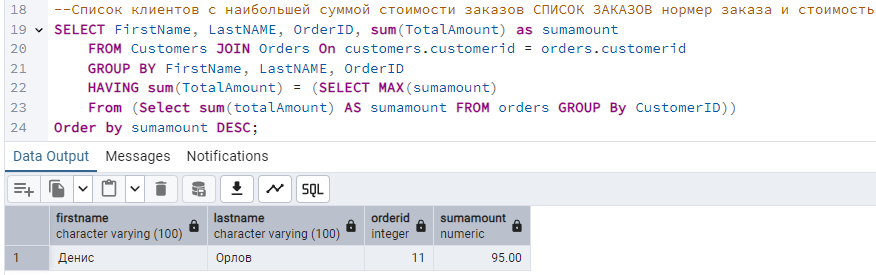


Рисунок 3 – Результат и код

4)Вывести клиентов, у которых общая сумма заказов превышает среднюю общую сумму заказов всех клиентов.

Код:

SELECT firstname, lastname, SUM(totalamount) AS sumtotalamount

FROM customers JOIN orders ON customers.customerid = orders.customerid

GROUP BY firstname, lastname

HAVING SUM(totalamount) > (SELECT avg(totalamount) FROM orders) ORDER BY sumtotalamount;

1) Выбираем firstname, lastname и сумму всех заказов для каждого клиента. Результат суммы будет иметь псевдоним sumtotalamount.

2) Извлекаем данные из таблицы customers, соединяя ее с таблицей orders по полю customerid, чтобы получить информацию о клиентах и их заказах.

3) Группируем результаты по имени и фамилии клиентов, чтобы сложить суммы заказов для каждого уникального клиента.

4) Используем оператор HAVING для фильтрации групп. Оставляем только тех клиентов, у которых сумма заказов больше средней суммы заказов по всем записям в таблице orders.

5) Сортируем результаты по sumtotalamount в порядке возрастания.

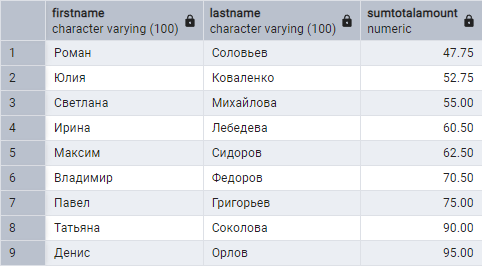


Рисунок 4 – Результат

4. Заполним базу данных из 8 лабораторной работы.

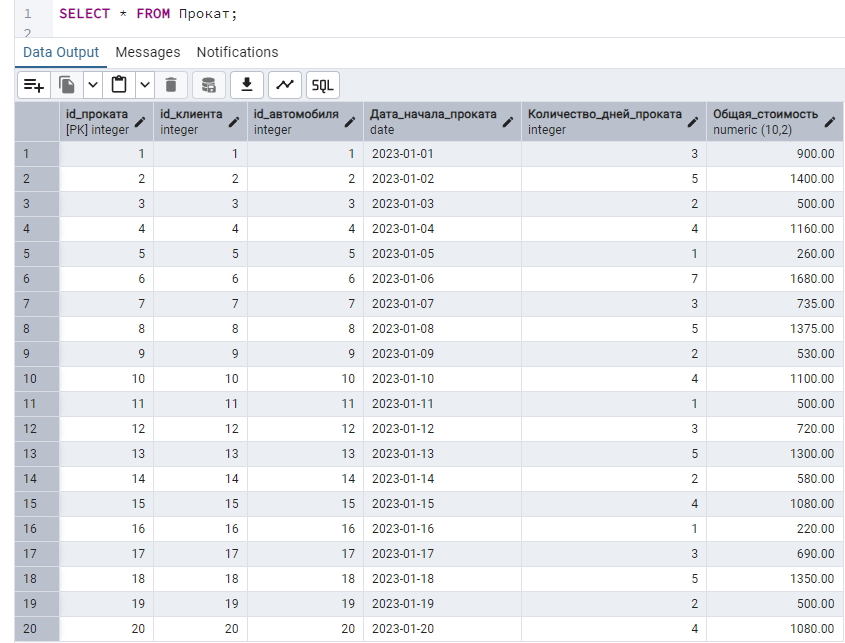


Рисунок 5 – Заполненная таблица ПРОКАТ.

Рисунок 5, показывает пример заполнения таблицы ПРОКАТ, так же приведен код для вывода таблицы.

5. Запросы

1)

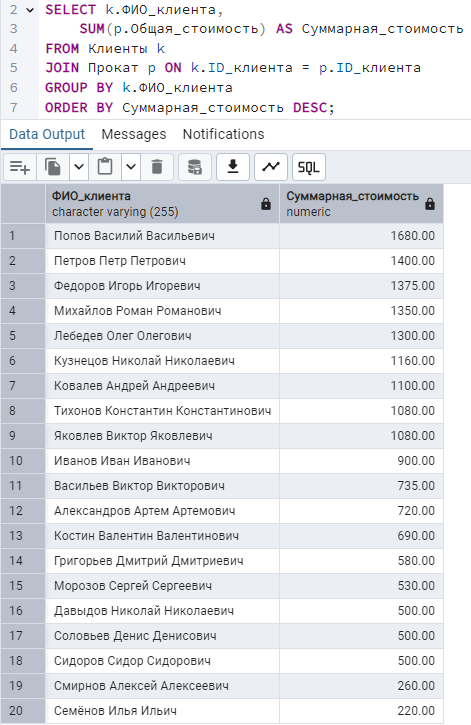


Рисунок 6 – Результат и код.

Запрос выбирает полное ФИО клиента и суммарную стоимостьиз двух таблиц: Клиенты и Прокат.

Далее используется оператор JOIN, чтобы соединить таблицы Клиенты и Прокат по полю ID\_клиента. Это означает, что для каждого клиента будут выбраны только те записи из таблицы Прокат, которые соответствуют этому клиенту. С помощью функции SUM вычисляем общую стоимость всех прокатов для каждого клиента. Результат это Суммарная\_стоимость. Данные группируются, что позволяет получить суммарную стоимость для каждого клиента. Результаты сортируются по суммарной стоимости в порядке убывания.

2) Список клиентов с наибольшей суммой.

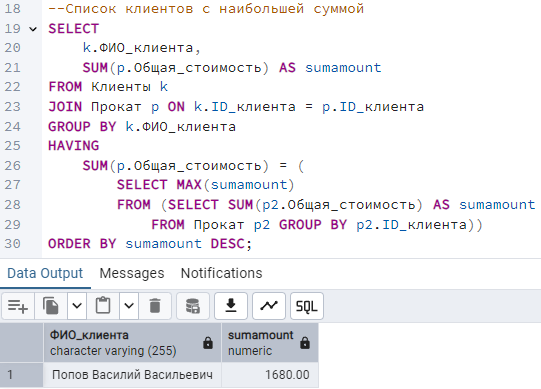


Рисунок 7 – Результат и код

Запрос выбирает полное ФИО клиента и суммарную стоимость из таблиц Клиенты и Прокат. Используем оператор JOIN, чтобы соединить таблицы Клиенты и Прокат по полю ID\_клиента. С помощью функции SUM вычисляется общая стоимость всех прокатов для каждого клиента.

Данные группируются по ФИО. Используется условие HAVING, чтобы выбрать только тех клиентов, у которых суммарная стоимость равна максимальной суммарной стоимости среди всех клиентов. Результаты сортируются по суммарной стоимости в порядке убывания.

3) Список клиентов с наибольшей суммой стоимости Аренды авто.

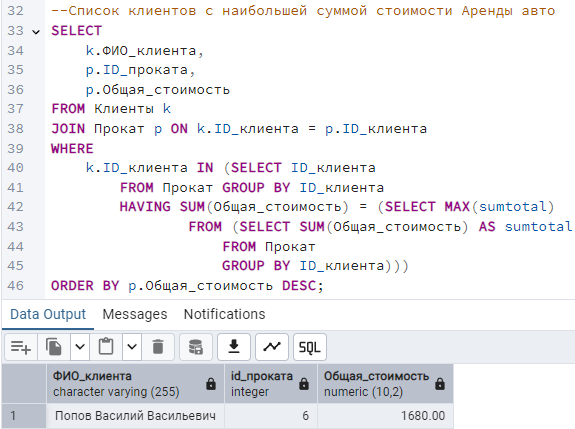


Рисунок 8 – Результат и код

Извлекаем ФИО, id проката и общая стоимость проката. Таблица "Клиенты" соединяется с таблицей "Прокат" по полю id клиента. Условие WHERE используется для определения клиентов, чья общая стоимость прокатов равна максимальной по всем клиентам. Результаты сортируются по общей стоимости проката в порядке убывания.

4) Список клиентов, которые потратили больше среднего на прокаты, вместе с их общими затратами, отсортированными по убыванию суммы заказов.

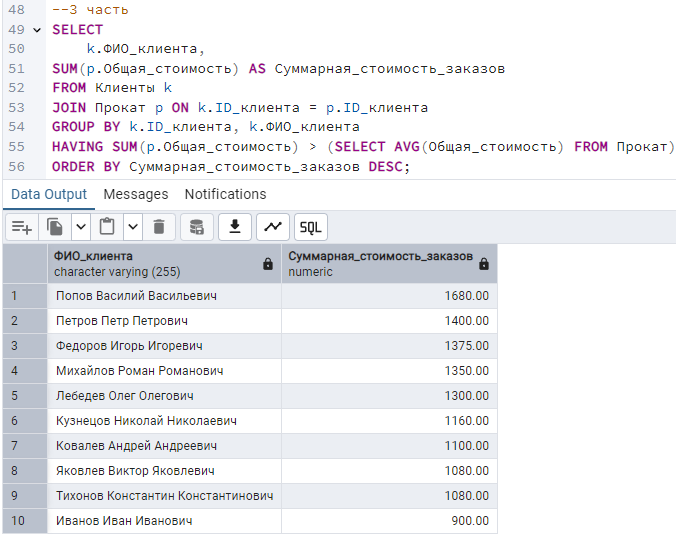


Рисунок 9 - Результат т код

Извлекается ФИО и сумма их общих затрат на прокаты. Таблица Клиенты объединяется с таблицей Прокат по полю id клиента. Данные группируются по id клиента и ФИО клиента, что позволяет подсчитать общую стоимость для каждого клиента. Используется условие HAVING для определения клиентов, чья суммарная стоимость заказов превышает среднее значение общей стоимости по всем прокатам. Результаты сортируются по суммарной стоимости заказов в порядке убывания.

6. Создание yml обертки.

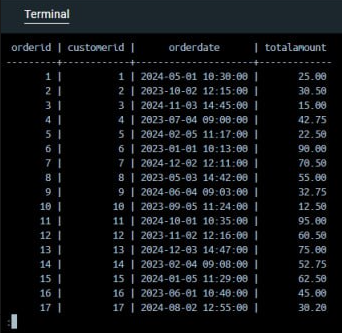


Рисунок 10 – Пример таблицы «OrderS»

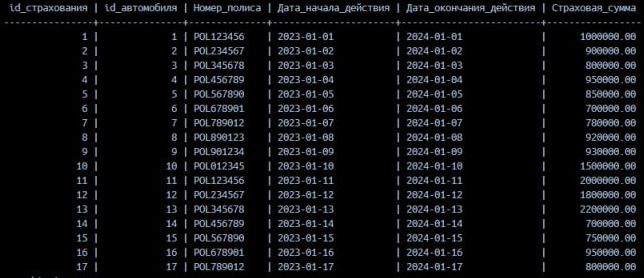


Рисунок 11 – Пример таблицы «Страхование»

Создаем файл docker-composse.yml. В файле указываем версию формата docker. Далее необходимо указать раздел Services, который будет показывать запущенные сервисы. Далее указываем имя сервиса; указываем, что будет использоваться Postgresql из образа Docker; environment: этот сервис задает переменные окружения для настройки базы данных:

POSTGRES\_DB: Имя базы данных;

POSTGRES\_USER: Имя пользователя для доступа к базе данных.

POSTGRES\_PASSWORD: Пароль для указанного пользователя.

volumes: Этот раздел определяет тома для хранения данных.

1) docker-compose up –d - команда запускает контейнеры, определенные в файле docker-compose.yml, в фоновом режиме (опция -d означает "detached mode"). Если контейнеры еще не созданы, они будут созданы автоматически. Если они уже существуют, они будут запущены заново.

2) docker exec -it games-dbpost-1 psql -U usrcompose -d dbcompose – эта команда выполняет команду внутри работающего контейнера Docker.

docker exec позволяет запускать команды в уже запущенном контейнере.

-it — это комбинация флагов, где -i означает "interactive", а -t — "pseudo-TTY", что позволяет вам взаимодействовать с командной строкой контейнера.

games-dbpost-1 — это имя контейнера, в котором вы хотите выполнить команду.

3) psql -U usrcompose -d dbcompose — это команда, которая запускает клиент PostgreSQL psql, используя пользователя usrcompose для подключения к базе данных dbcompose.

4) \dt. Данная команда, используемая в клиенте psql, которая отображает список всех таблиц в текущей базе данных. Показывает информацию о таблицах, таких как их имена и схемы.

5) SELECT \* FROM Атопрокат – Выведет всю информацию из указанной талбицы.