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: Власов И.В.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

*Цель лабораторной 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 выполнения лабораторной работы

Для тренировочного задания создадим файл docker-compose.yml. Адаптируем его под тренировочное задание, изменим название файла init.sql на LR\_9\_tren.sql и строку, ссылающуюся на него. В файле LR\_9\_tren.sql напишем SQL-скрипты для создания таблиц и заполнения их значениями. Содержимое файла LR\_9\_tren.sql представлено ниже:

-- Создание таблицы "Customers"

CREATE TABLE Customers (

CustomerID SERIAL PRIMARY KEY,

FirstName VARCHAR(255) NOT NULL,

LastName VARCHAR(255) NOT NULL,

Email VARCHAR(255) NOT NULL

);

-- Создание таблицы "Orders"

CREATE TABLE Orders (

OrderID SERIAL PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate TIMESTAMP NOT NULL,

TotalAmount DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));

-- Создание таблицы "OrderDetails"

CREATE TABLE OrderDetails (

OrderDetailID SERIAL PRIMARY KEY,

OrderID INT NOT NULL,

ProductID INT NOT NULL,

Quantity INT NOT NULL,

UnitPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

Заполнение таблиц значениями:

INSERT INTO Customers (FirstName, LastName, Email) VALUES

('Ivan', 'Ivanov', 'ivan.ivanov@example.com'),

('Petr', 'Petrov', 'petr.petrov@example.com'),

('Sofia', 'Sokolova', 'sofia.sokolova@example.com'),

('Dmitry', 'Smirnov', 'dmitry.smirnov@example.com'),

('Olga', 'Orlova', 'olga.orlova@example.com'),

('Anna', 'Anisimova', 'anna.anisimova@example.com'),

('Nikolay', 'Nikolaev', 'nikolay.nikolaev@example.com'),

('Maria', 'Morozova', 'maria.morozova@example.com'),

('Yulia', 'Yakovleva', 'yulia.yakovleva@example.com'),

('Sergey', 'Sergeev', 'sergey.sergeev@example.com'),

('Elena', 'Egorova', 'elena.egorova@example.com'),

('Alexey', 'Alekseev', 'alexey.alekseev@example.com'),

('Ekaterina', 'Ershova', 'ekaterina.ershova@example.com'),

('Andrey', 'Antonov', 'andrey.antonov@example.com'),

('Viktoria', 'Volkova', 'viktoria.volkova@example.com'),

('Oleg', 'Ozerov', 'oleg.ozerov@example.com'),

('Irina', 'Ivanova', 'irina.ivanova@example.com'),

('Mikhail', 'Mikhailov', 'mikhail.mikhailov@example.com'),

('Svetlana', 'Svetlova', 'svetlana.svetlova@example.com'),

('Igor', 'Ignatiev', 'igor.ignatiev@example.com');

INSERT INTO Orders (CustomerID, OrderDate, TotalAmount) VALUES

(1, '2024-12-01 10:00:00', 100.00),

(2, '2024-12-02 11:00:00', 150.00),

(3, '2024-12-03 12:00:00', 200.00),

(4, '2024-12-04 13:00:00', 250.00),

(5, '2024-12-05 14:00:00', 300.00),

(6, '2024-12-06 15:00:00', 350.00),

(7, '2024-12-07 16:00:00', 400.00),

(8, '2024-12-08 17:00:00', 450.00),

(9, '2024-12-09 18:00:00', 500.00),

(10, '2024-12-10 19:00:00', 550.00),

(11, '2024-12-11 20:00:00', 600.00),

(12, '2024-12-12 21:00:00', 650.00),

(13, '2024-12-13 22:00:00', 700.00),

(14, '2024-12-14 23:00:00', 750.00),

(15, '2024-12-15 08:00:00', 800.00),

(16, '2024-12-16 09:00:00', 850.00),

(17, '2024-12-17 10:30:00', 900.00),

(18, '2024-12-18 11:30:00', 950.00),

(19, '2024-12-19 12:30:00', 1000.00),

(20, '2024-12-20 13:30:00', 1050.00);

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

(1, 101, 2, 50.00), (2, 102, 3, 50.00), (3, 103, 4, 50.00), (4, 104, 5, 50.00), (5, 105, 6, 50.00), (6, 106, 7, 50.00), (7, 107, 8, 50.00), (8, 108, 9, 50.00), (9, 109, 10, 50.00), (10, 110, 11, 50.00), (11, 111, 12, 50.00), (12, 112, 13, 50.00), (13, 113, 14, 50.00), (14, 114, 15, 50.00), (15, 115, 16, 50.00), (16, 116, 17, 50.00), (17, 117, 18, 50.00), (18, 118, 19, 50.00), (19, 119, 20, 50.00), (20, 120, 21, 50.00);

Таким образом, при запуске контейнера будет сразу создана заполненная база данных. Проверим созданные таблицы (Рисунок 1).

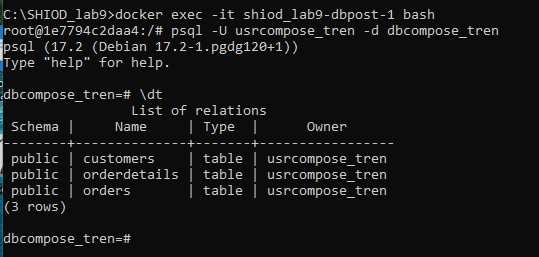


Рисунок 1 – Таблицы тренировочного задания

Приступаем к созданию запросов к базе данных:

Сформировать SQL-запрос для списка клиентов и суммарной стоимости заказов (Рисунок 2).

SELECT

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.FirstName, c.LastName;



Рисунок 2 - Список клиентов и суммарная стоимость заказов

Отсортировать список по убыванию суммарной стоимости заказов клиента (Рисунок 3).

SELECT

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.FirstName, c.LastName

ORDER BY

TotalOrderAmount DESC;



Рисунок 3 - Список по убыванию суммарной стоимости заказов клиента

Добавить столбец со средней суммарной стоимостью заказов (Рисунок 4).

SELECT

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount,

AVG(SUM(o.TotalAmount)) OVER () AS AverageOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.FirstName, c.LastName

ORDER BY

TotalOrderAmount DESC;

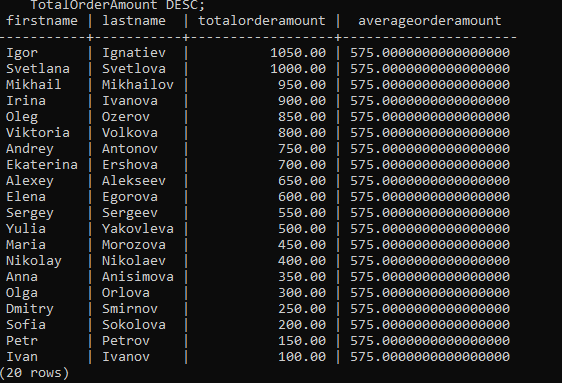


Рисунок 4 - Столбец со средней суммарной стоимостью заказов

Вывести клиента с наибольшей суммарной стоимостью заказов (Рисунок 5).

SELECT

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.FirstName, c.LastName

ORDER BY

TotalOrderAmount DESC

LIMIT 1;

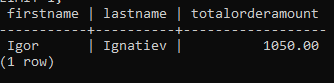


Рисунок 5 - Клиент с наибольшей суммарной стоимостью заказов

Вывести список заказов клиента с наибольшей суммарной стоимостью заказов (Рисунок 6).

WITH TopCustomer AS (

SELECT

c.CustomerID,

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.CustomerID, c.FirstName, c.LastName

ORDER BY

TotalOrderAmount DESC

LIMIT 1

)

SELECT

o.OrderID,

o.TotalAmount

FROM

Orders o

JOIN

TopCustomer tc ON o.CustomerID = tc.CustomerID

ORDER BY

o.TotalAmount ASC;

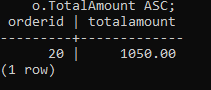


Рисунок 6 - Список заказов клиента с наибольшей суммарной стоимостью заказов

Вывести клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов (Рисунок 7).

WITH CustomerTotals AS (

SELECT

c.FirstName,

c.LastName,

SUM(o.TotalAmount) AS TotalOrderAmount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.FirstName, c.LastName

),

AverageTotal AS (

SELECT

AVG(TotalOrderAmount) AS AverageOrderAmount

FROM

CustomerTotals

)

SELECT

ct.FirstName,

ct.LastName,

ct.TotalOrderAmount,

at.AverageOrderAmount

FROM

CustomerTotals ct,

AverageTotal at

WHERE

ct.TotalOrderAmount > at.AverageOrderAmount;

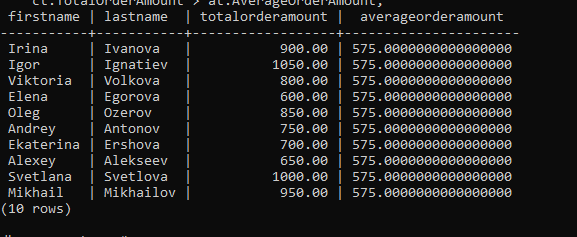


Рисунок 7 - Клиенты, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов

Сформируем похожие запросы для ранее созданной базы данных «Учет подписки на периодические печатные издания». Обернем ее в docker-контейнер и при помощи sql-скрипта пересоздадим ее с заполненными значениями. Запустим его при помощи следующих команд:

docker exec -it shiod\_lab6-dbpost-1 bash

psql -U usrcompose -d dbcompose

Содержимое файла LR\_8.sql представлено ниже:

-- Создание таблицы "Издания"

CREATE TABLE Издания (

индекс\_издания SERIAL PRIMARY KEY,

название\_издания VARCHAR(255) NOT NULL,

вид\_издания VARCHAR(50) NOT NULL,

цена\_1\_экземпляра DECIMAL(10, 2) NOT NULL

);

-- Создание таблицы "Получатели"

CREATE TABLE Получатели (

идентификатор\_получателя SERIAL PRIMARY KEY,

фио\_подписчика VARCHAR(255) NOT NULL,

улица VARCHAR(255) NOT NULL,

номер\_дома VARCHAR(10) NOT NULL,

номер\_квартиры VARCHAR(10) NOT NULL

);

-- Создание таблицы "Подписки"

CREATE TABLE Подписки (

идентификатор\_подписки SERIAL PRIMARY KEY,

идентификатор\_получателя INT,

индекс\_издания INT,

дата\_начала\_подписки DATE NOT NULL,

срок\_подписки INT NOT NULL,

стоимость\_подписки DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (идентификатор\_получателя) REFERENCES Получатели(идентификатор\_получателя),

FOREIGN KEY (индекс\_издания) REFERENCES Издания(индекс\_издания)

);

-- Создание таблицы "Доставка"

CREATE TABLE Доставка (

идентификатор\_доставки SERIAL PRIMARY KEY,

идентификатор\_подписки INT,

дата\_доставки DATE NOT NULL,

стоимость\_доставки DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (идентификатор\_подписки) REFERENCES Подписки(идентификатор\_подписки)

);

INSERT INTO Издания (название\_издания, вид\_издания, цена\_1\_экземпляра) VALUES

('Газета 1', 'Газета', 50.00),

('Газета 2', 'Газета', 60.00),

('Газета 3', 'Газета', 55.00),

('Газета 4', 'Газета', 40.00),

('Газета 5', 'Газета', 45.00),

('Журнал 1', 'Журнал', 100.00),

('Журнал 2', 'Журнал', 120.00),

('Журнал 3', 'Журнал', 110.00),

('Журнал 4', 'Журнал', 130.00),

('Журнал 5', 'Журнал', 115.00),

('Журнал 6', 'Журнал', 125.00),

('Журнал 7', 'Журнал', 135.00),

('Журнал 8', 'Журнал', 140.00),

('Журнал 9', 'Журнал', 150.00),

('Журнал 10', 'Журнал', 160.00),

('Журнал 11', 'Журнал', 170.00),

('Газета 6', 'Газета', 70.00),

('Газета 7', 'Газета', 80.00),

('Газета 8', 'Газета', 65.00),

('Газета 9', 'Газета', 75.00);

INSERT INTO Получатели (фио\_подписчика, улица, номер\_дома, номер\_квартиры) VALUES

('Иванов Иван Иванович', 'ул. Ленина', '1', '1'),

('Петров Петр Петрович', 'ул. Гагарина', '2', '2'),

('Сидоров Сидор Сидорович', 'ул. Пушкина', '3', '3'),

('Кузнецов Николай Николаевич', 'ул. Мира', '4', '4'),

('Смирнов Алексей Алексеевич', 'ул. Советская', '5', '5'),

('Васильев Василий Васильевич', 'ул. Победы', '6', '6'),

('Морозов Дмитрий Дмитриевич', 'ул. Жукова', '7', '7'),

('Федоров Федор Федорович', 'ул. Калинина', '8', '8'),

('Ковалев Евгений Евгеньевич', 'ул. Кузнецова', '9', '9'),

('Волков Владимир Владимирович', 'ул. Гагарина', '10', '10'),

('Егоров Егор Егорович', 'ул. Ленина', '11', '11'),

('Захаров Захар Захарович', 'ул. Мира', '12', '12'),

('Лебедев Леонид Леонидович', 'ул. Советская', '13', '13'),

('Медведев Михаил Михайлович', 'ул. Победы', '14', '14'),

('Николаев Николай Николаевич', 'ул. Жукова', '15', '15'),

('Павлов Павел Павлович', 'ул. Калинина', '16', '16'),

('Романов Роман Романович', 'ул. Кузнецова', '17', '17'),

('Семёнов Семён Семёнович', 'ул. Гагарина', '18', '18'),

('Титов Тимофей Тимофеевич', 'ул. Ленина', '19', '19'),

('Ушаков Ульян Ульянович', 'ул. Мира', '20', '20');

INSERT INTO Подписки (идентификатор\_получателя, индекс\_издания, дата\_начала\_подписки, срок\_подписки, стоимость\_подписки) VALUES

(1, 1, '2024-01-01', 12, 600.00),

(2, 2, '2024-02-01', 6, 360.00),

(3, 3, '2024-03-01', 3, 165.00),

(4, 4, '2024-04-01', 1, 40.00),

(5, 5, '2024-05-01', 12, 540.00),

(6, 6, '2024-06-01', 12, 1200.00),

(7, 7, '2024-07-01', 6, 720.00),

(8, 8, '2024-08-01', 3, 330.00),

(9, 9, '2024-09-01', 1, 130.00),

(10, 10, '2024-10-01', 12, 1380.00),

(11, 11, '2024-11-01', 6, 1020.00),

(12, 12, '2024-12-01', 3, 375.00),

(13, 13, '2024-01-15', 1, 150.00),

(14, 14, '2024-02-15', 12, 1680.00),

(15, 15, '2024-03-15', 6, 1020.00),

(16, 16, '2024-04-15', 3, 375.00),

(17, 17, '2024-05-15', 1, 125.00),

(18, 18, '2024-06-15', 12, 1500.00),

(19, 19, '2024-07-15', 6, 900.00),

(20, 20, '2024-08-15', 3, 225.00);

INSERT INTO Доставка (идентификатор\_подписки, дата\_доставки, стоимость\_доставки) VALUES

(1, '2024-02-01', 6.00),

(2, '2024-03-01', 3.60),

(3, '2024-04-01', 1.65),

(4, '2024-05-01', 0.40),

(5, '2024-06-01', 5.40),

(6, '2024-07-01', 12.00),

(7, '2024-08-01', 7.20),

(8, '2024-09-01', 3.30),

(9, '2024-10-01', 1.30),

(10, '2024-11-01', 13.80),

(11, '2024-12-01', 10.20),

(12, '2025-01-01', 3.75),

(13, '2025-02-01', 1.50),

(14, '2025-03-01', 16.80),

(15, '2025-04-01', 10.20),

(16, '2025-05-01', 3.75),

(17, '2025-06-01', 1.25),

(18, '2025-07-01', 15.00),

(19, '2025-08-01', 9.00),

(20, '2025-09-01', 2.25);

Сформируем похожие запросы для базы из индивидуального задания:

Сформировать SQL-запрос, который возвращает список получателей (имя и фамилия) и суммарную стоимость подписок каждого получателя

sql

SELECT

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.фио\_подписчика;

Отсортировать полученный список по убыванию суммарной стоимости подписок получателя

sql

SELECT

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.фио\_подписчика

ORDER BY

TotalSubscriptionCost DESC;

Добавить столбец со средней суммарной стоимостью подписок

sql

SELECT

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost,

AVG(SUM(s.стоимость\_подписки)) OVER () AS AverageSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.фио\_подписчика

ORDER BY

TotalSubscriptionCost DESC;

Вывести получателя с наибольшей суммарной стоимостью подписок

sql

SELECT

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.фио\_подписчика

ORDER BY

TotalSubscriptionCost DESC

LIMIT 1;

Вывести список подписок получателя с наибольшей суммарной стоимостью подписок

sql

WITH TopRecipient AS (

SELECT

p.идентификатор\_получателя,

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.идентификатор\_получателя, p.фио\_подписчика

ORDER BY

TotalSubscriptionCost DESC

LIMIT 1

)

SELECT

s.идентификатор\_подписки,

s.стоимость\_подписки

FROM

Подписки s

JOIN

TopRecipient tr ON s.идентификатор\_получателя = tr.идентификатор\_получателя

ORDER BY

s.стоимость\_подписки ASC;

Вывести только тех получателей, у которых суммарная стоимость подписок превышает среднюю суммарную стоимость подписок

sql

WITH RecipientTotals AS (

SELECT

p.фио\_подписчика,

SUM(s.стоимость\_подписки) AS TotalSubscriptionCost

FROM

Получатели p

JOIN

Подписки s ON p.идентификатор\_получателя = s.идентификатор\_получателя

GROUP BY

p.фио\_подписчика

),

AverageTotal AS (

SELECT

AVG(TotalSubscriptionCost) AS AverageSubscriptionCost

FROM

RecipientTotals

)

SELECT

rt.фио\_подписчика,

rt.TotalSubscriptionCost,

at.AverageSubscriptionCost

FROM

RecipientTotals rt,

AverageTotal at

WHERE

rt.TotalSubscriptionCost > at.AverageSubscriptionCost;

Ссылка на репозиторий: https://github.com/IgorVlasov1/SHIOD\_LAB9.

Ответы на контрольные вопросы:

1. Подзапрос (или вложенный запрос) – это запрос, который находится внутри другого SQL-запроса. Подзапросы используются для получения промежуточных данных, которые затем используются в основном запросе. Подзапросы могут находиться в разделе SELECT, WHERE или FROM главного запроса.
2. Для заполнения таблицы данными используется команда INSERT INTO. Можно вставить одну или несколько строк за один раз.
3. Агрегатные функции в SQL используются для выполнения вычислений над набором значений и возврата одного значения. Основные агрегатные функции включают: COUNT(): возвращает количество строк. SUM(): возвращает сумму значений. AVG(): возвращает среднее значение. MAX(): возвращает максимальное значение. MIN(): возвращает минимальное значение.
4. Таблицы можно объединять с использованием команд JOIN. Основные типы объединений включают INNER JOIN, LEFT JOIN, RIGHT JOIN и FULL OUTER JOIN.
5. INNER JOIN: возвращает строки, которые имеют соответствия в обеих таблицах. LEFT JOIN (или LEFT OUTER JOIN): возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если соответствий нет, возвращаются NULL значения. RIGHT JOIN (или RIGHT OUTER JOIN): возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если соответствий нет, возвращаются NULL значения. FULL OUTER JOIN: возвращает все строки, когда есть соответствие в одной из таблиц.
6. План запроса — это план выполнения SQL-запроса, показывающий, как база данных собирается извлечь данные. Он включает информацию о порядке выполнения операций, использовании индексов, типах соединений и других параметрах.
7. Оптимизация запроса — это процесс улучшения производительности SQL-запроса. Основные методы включают: использование индексов. Оптимизация схемы базы данных. Переписывание запросов для уменьшения количества операций. Использование агрегатных функций и подзапросов.
8. Оконные функции используются для выполнения вычислений по набору строк, связанных с текущей строкой. Они позволяют выполнять операции, такие как суммирование, ранжирование и другие, без группировки строк.
9. Для группировки данных используется команда GROUP BY. Она группирует строки по указанным столбцам.
10. Для сортировки данных используется команда ORDER BY. Она сортирует результаты запроса по указанным столбцам в порядке возрастания (ASC) или убывания (DESC).
11. WHERE: используется для фильтрации строк на уровне таблиц. Условие применяется до группировки данных. HAVING: используется для фильтрации результатов после группировки. Применяется к агрегатным функциям.