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-231: Никулин В.С.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2023

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

**Задачи работы:**

1) Изучить синтаксис SQL-запросов в СУБД Postgres.

2) Изучить основы формирования SQL-запросов, включая такие вопросы, как: подзапросы, агрегатные функции, объединение таблиц.

3) Изучить способы оптимизации запросов, план запроса, оконные функции.

4) Сформировать SQL-запросы для создания таблиц в СУБД Postgres с соответствии со следующей структурой:

a) Customers — информация о клиентах:

• CustomerID (int, PK) — уникальный идентификатор клиента.

• FirstName (varchar) — имя клиента.

• LastName (varchar) — фамилия клиента.

• Email (varchar) — адрес электронной почты клиента.

b) Orders — информация о заказах:

• OrderID (int, PK) — уникальный идентификатор заказа.

• CustomerID (int) — идентификатор клиента, совершившего заказ.

• OrderDate (datetime) — дата и время создания заказа.

• TotalAmount (decimal) — общая стоимость заказа.

c) OrderDetails — информация о деталях заказов:

• OrderDetailID (int, PK) — уникальный идентификатор детали заказа.

• OrderID (int) — идентификатор заказа.

• ProductID (int) — идентификатор продукта в заказе.

• Quantity (int) — количество продуктов в заказе.

• UnitPrice (decimal) — цена за единицу продукта.

5) При необходимости создать дополнительные атрибуты и таблицы.

6) Наполнить базу данных данными (таблицы по 20 – 50 строк).

7) Сформировать SQL-запрос, который возвращает список клиентов (имя и фамилия) и суммарную стоимость заказов каждого клиента (должны отобразиться столбцы с именем, фамилией и стоимостью). То есть все заказы каждого клиента надо просуммировать.

*Примечание: при формировании SQL запросов необходимо:*

*• использовать подзапросы, агрегатные функции и объединения таблиц (там, где нужно);*

*• оптимизировать запрос, если это возможно.*

8) Отсортировать с полученный список по убыванию суммарной стоимости заказов клиента.

9) Добавить столбец со средней суммарной стоимостью заказов.

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

11) Для каждого клиента с наибольшей суммарной стоимостью заказов вывести список его заказов (номер заказа и стоимость) в порядке возрастания стоимости заказа.

12) Вывести только тех клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов (имя, фамилия, суммарная стоимость заказов клиента, средняя стоимость заказа).

13) Сохранить SQL-запросы в соответствующем файле со скриптами SQL.

14) Сформулировать задачу, аналогичную изложенной выше (в пунктах 4 – 12), на основе базы данных, разработанной в предыдущей лабораторной работе в соответствии с индивидуальным заданием. Для своей задачи разрешается создать новую базу данных либо дополнить уже имеющуюся.

15) Сохранить SQL-запросы для решения задачи согласно индивидуальному заданию в соответствующем файле со скриптами SQL.

16) Описать каждый запрос (что делает запрос, из каких элементов (составляющих) состоит запрос, роль этих составляющих) (как для тренировочной задачи, так и для задачи в соответствии с индивидуальным заданием) в форме комментариев в файле со скриптами.

17) Обернуть созданную базу данных в docker-контейнер (файл docker-compose.yml).

18) Подготовить отчёт о проделанной работе.

19) Отчётные материалы загрузить в репозиторий Git и отправить ссылку на ваш репозиторий на платформе github на почту преподавателю. Репозиторий должен быть публичным.

**Ход работы:**

Для начала, были сформированы SQL-запросы для создания таблиц в СУБД Postgres в соответствии со структурой описанной в 4-м пункте учебной задачи. Структура представлена тремя таблицами: Customers (Клиенты), Orders (Заказы) и OrderDetails (Детальная информация заказов). Название таблиц и их полей были переведены и записаны на русском языке, для удобства выполнения лабораторной работы. Также, к этим таблицам была добавлена 4-я таблица – таблица «Товары», т.к. таблица OrderDetails (Детальная информация заказов) содержала внешний ключ ProductID (идентификатор продукта/товара), а соответствующей таблицы не было.

По итогу, получились 4 таблицы:

1) Клиенты (ID клиента, Фамилия, Имя, Email);

2) Заказы (ID заказа, ID клиента, Дата заказа, Общая стоимость заказа);

3) Товары (ID товара, Наименование товара);

4) Детальная информация заказов (ID детальной информации заказа, ID заказа, ID товара, Количество товаров в заказе, Цена на единицу товара).

Соответствующие скрипты SQL-запросов на создание этих 4-х таблиц показаны на рисунке 1. Далее, эти запросы были выполнены в программе DBeaver (рисунок 2).

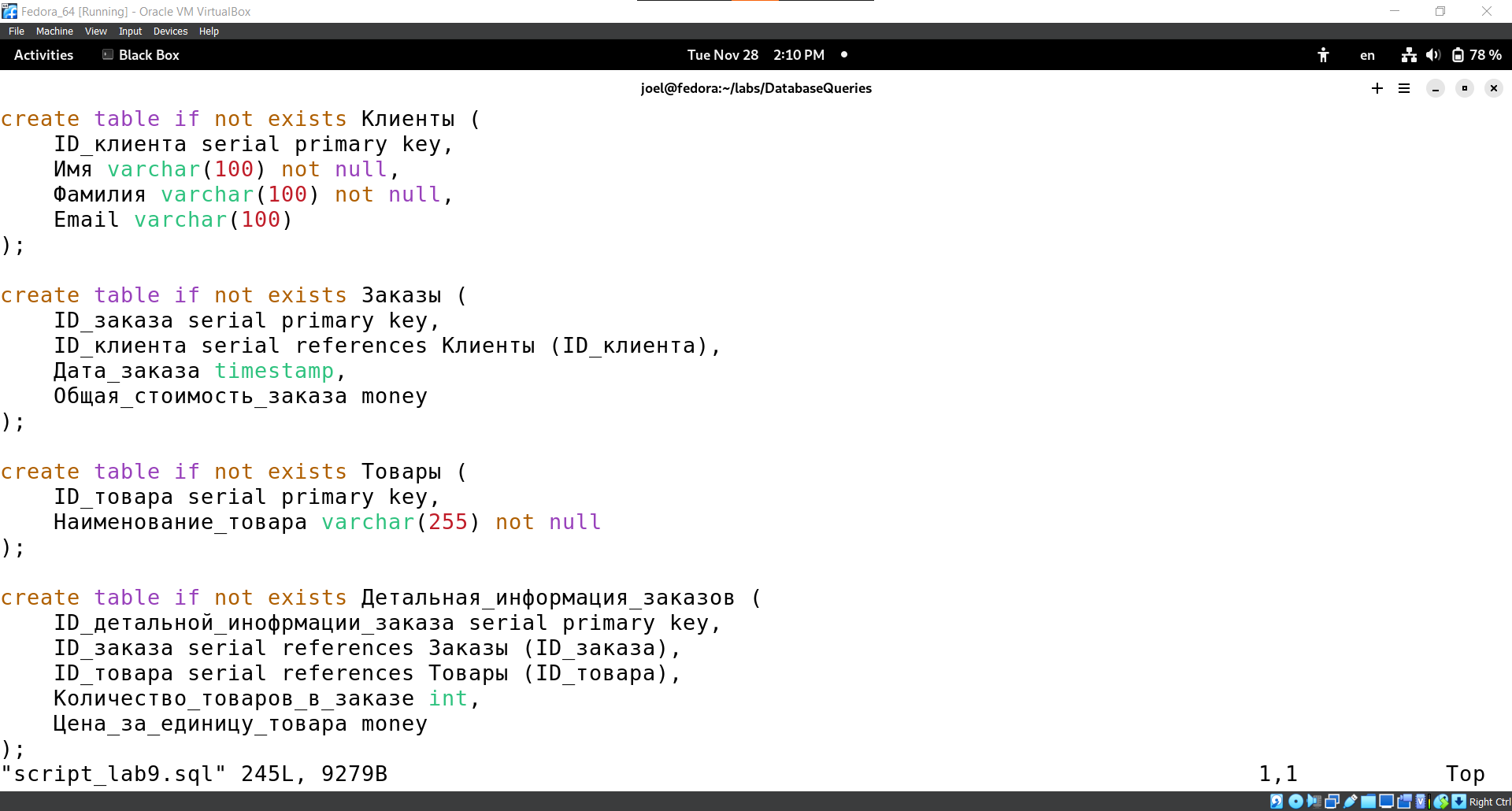


Рисунок 1 - Сформированные SQL-запросы для создания таблиц  
из 4-го пункта учебной задачи

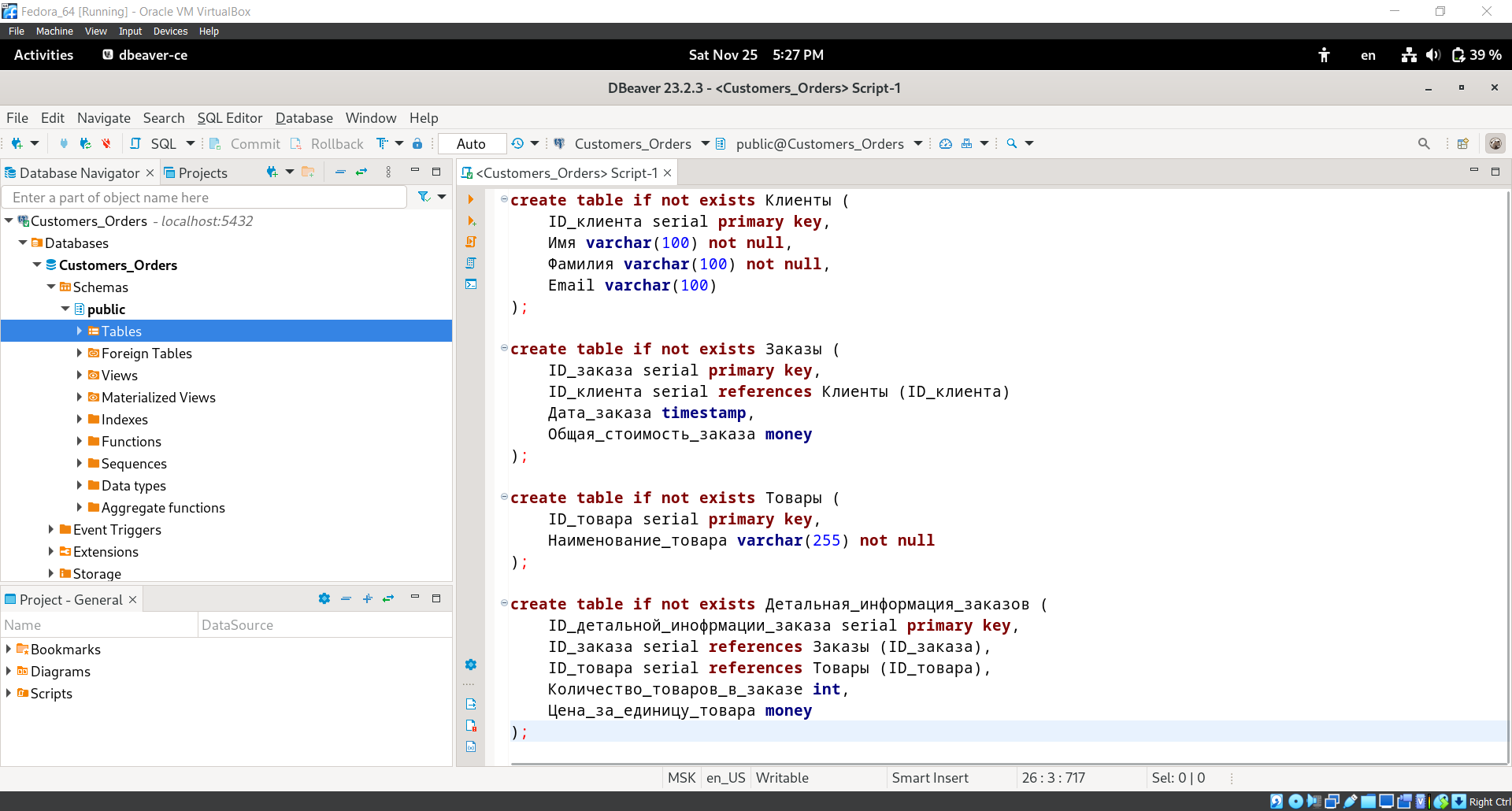


Рисунок 2 - Выполнение запросов на создание таблиц в DBeaver

Далее, созданные таблицы необходимо было заполнить данными от 20 до 50 строк. Для этого, для каждой таблицы был сформирован SQL-запрос на внесение данных в таблицу. Далее, каждый из этих запросов был так же выполнен в DBeaver. Заполнение таблицы «Товары» показано на рисунке 3. Заполнение таблицы «Клиенты» показано на рисунке 4. Заполнение таблицы «Заказы» показано на рисунке 5. Заполнение таблицы «Детальная информация заказов» показано на рисунке 6.

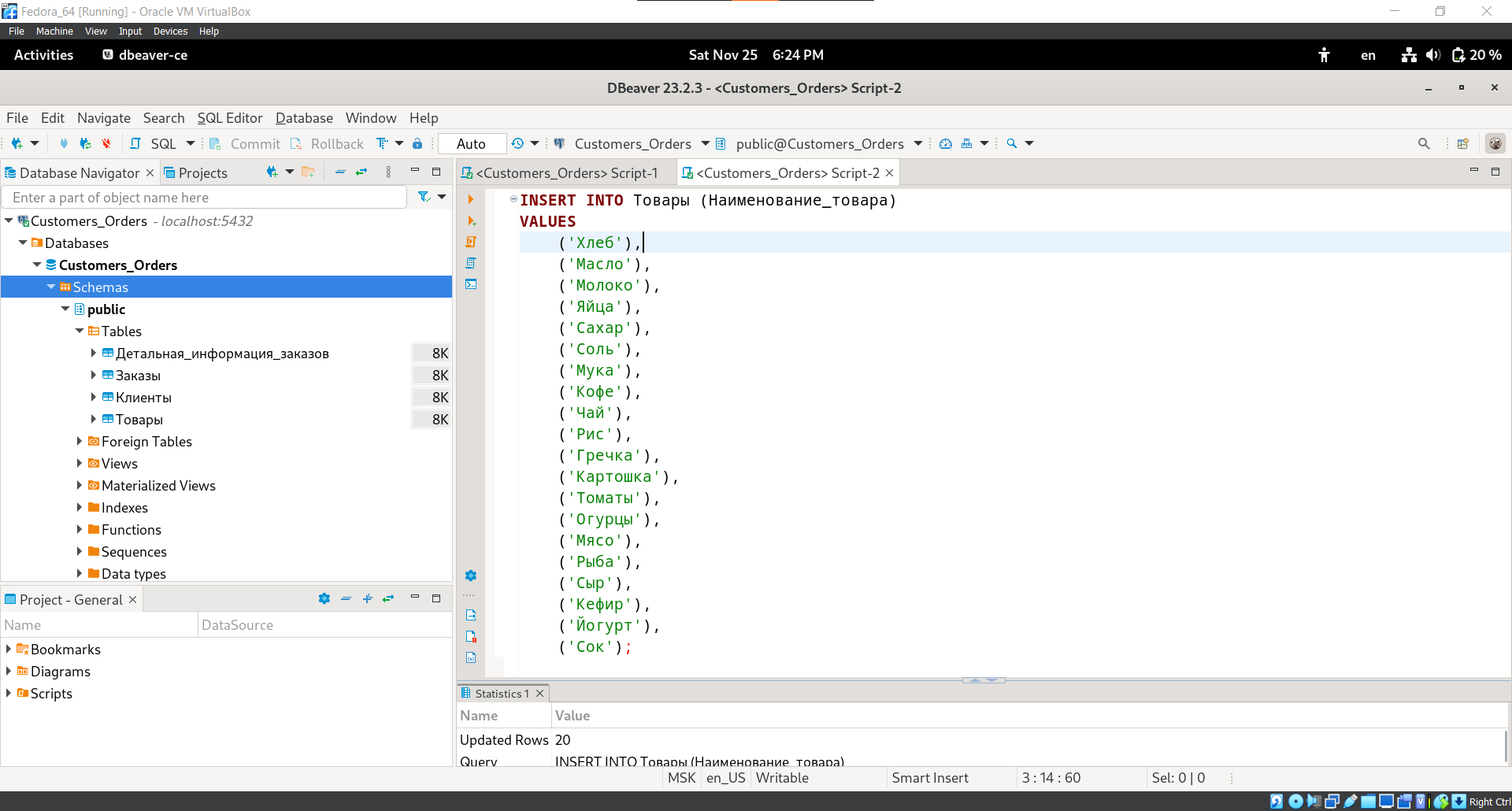


Рисунок 3 - Заполнение таблицы «Товары»

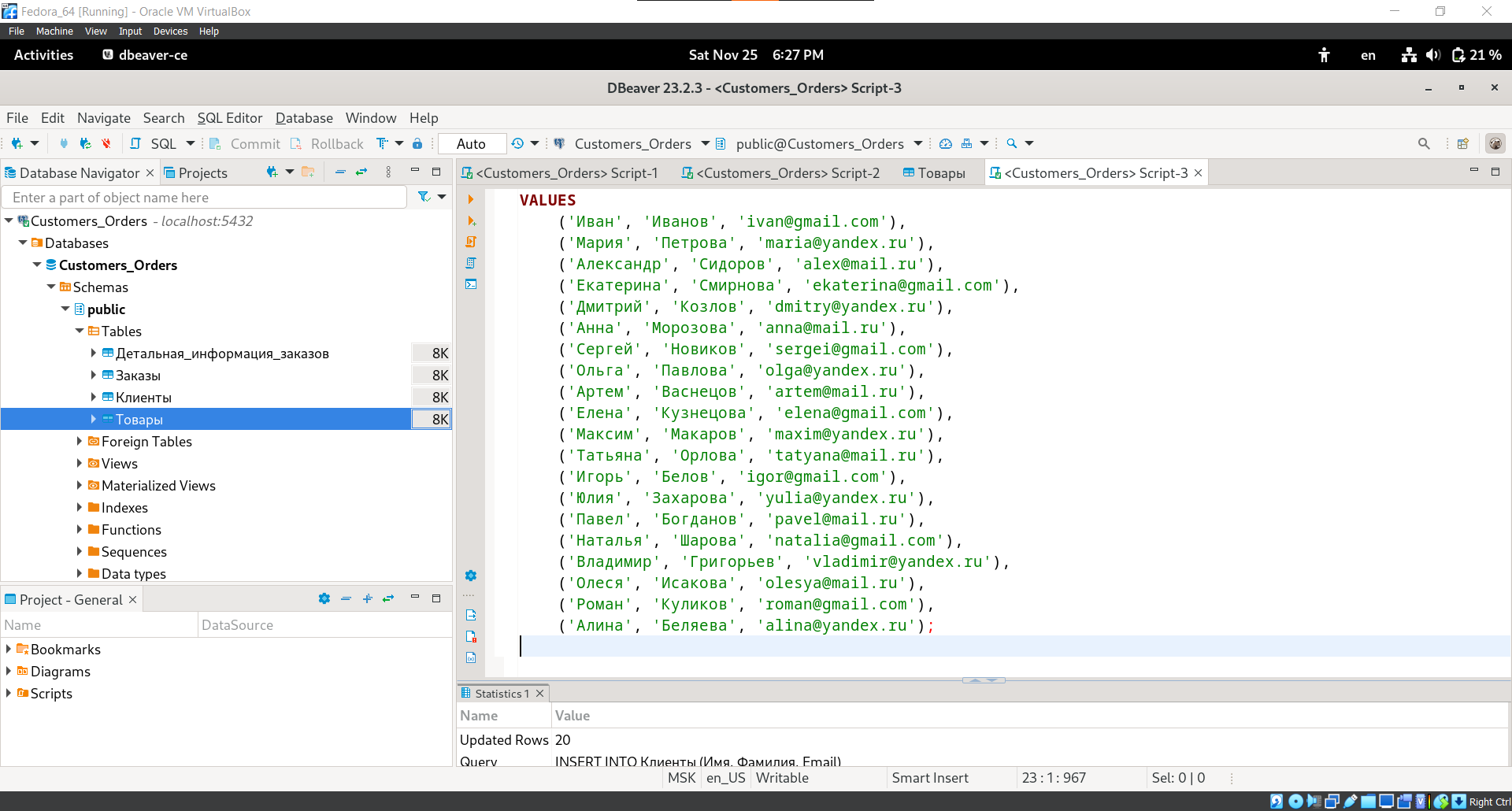


Рисунок 4 - Заполнение таблицы «Клиенты»

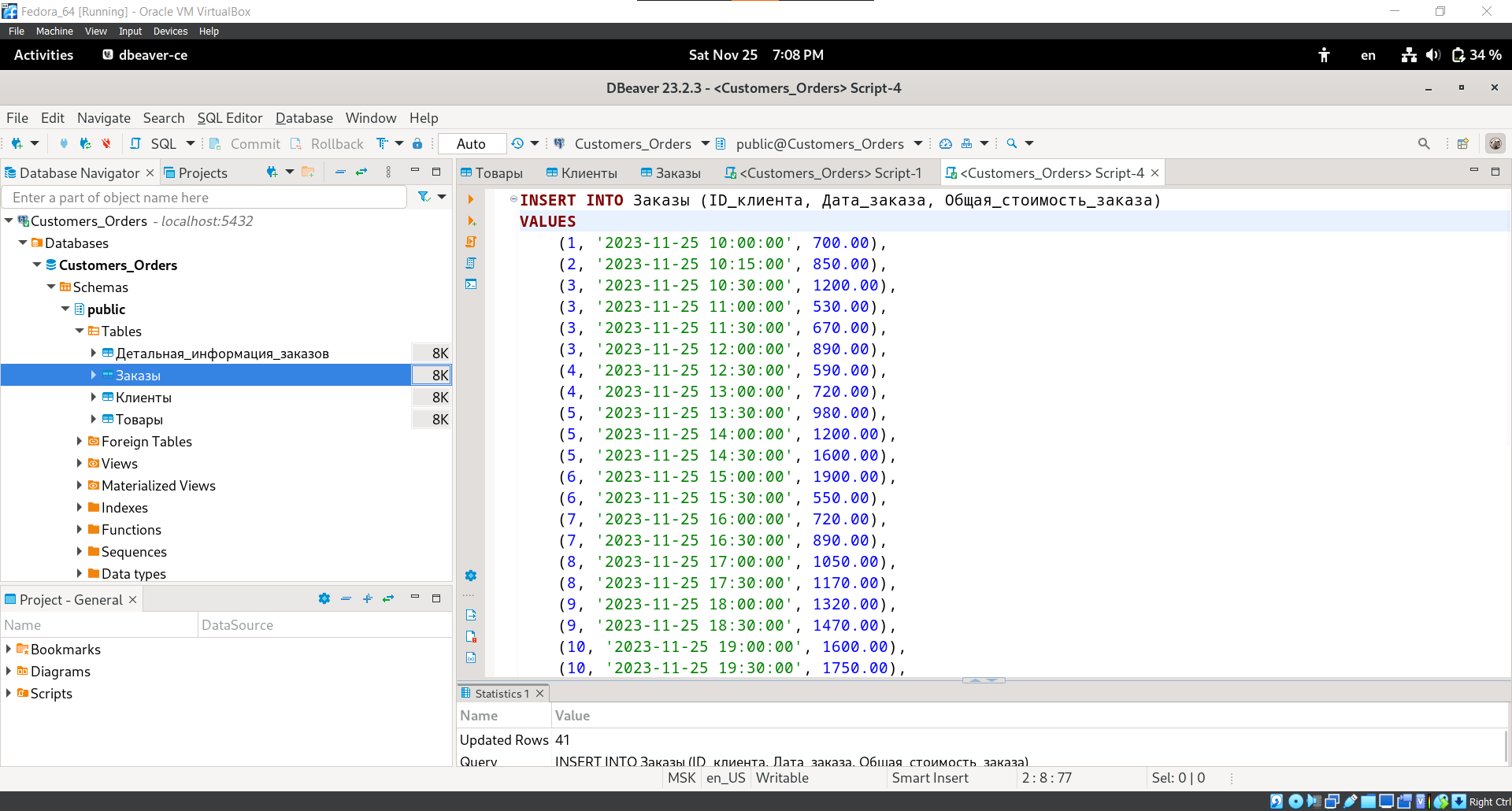


Рисунок 5 - Заполнение таблицы «Заказы»

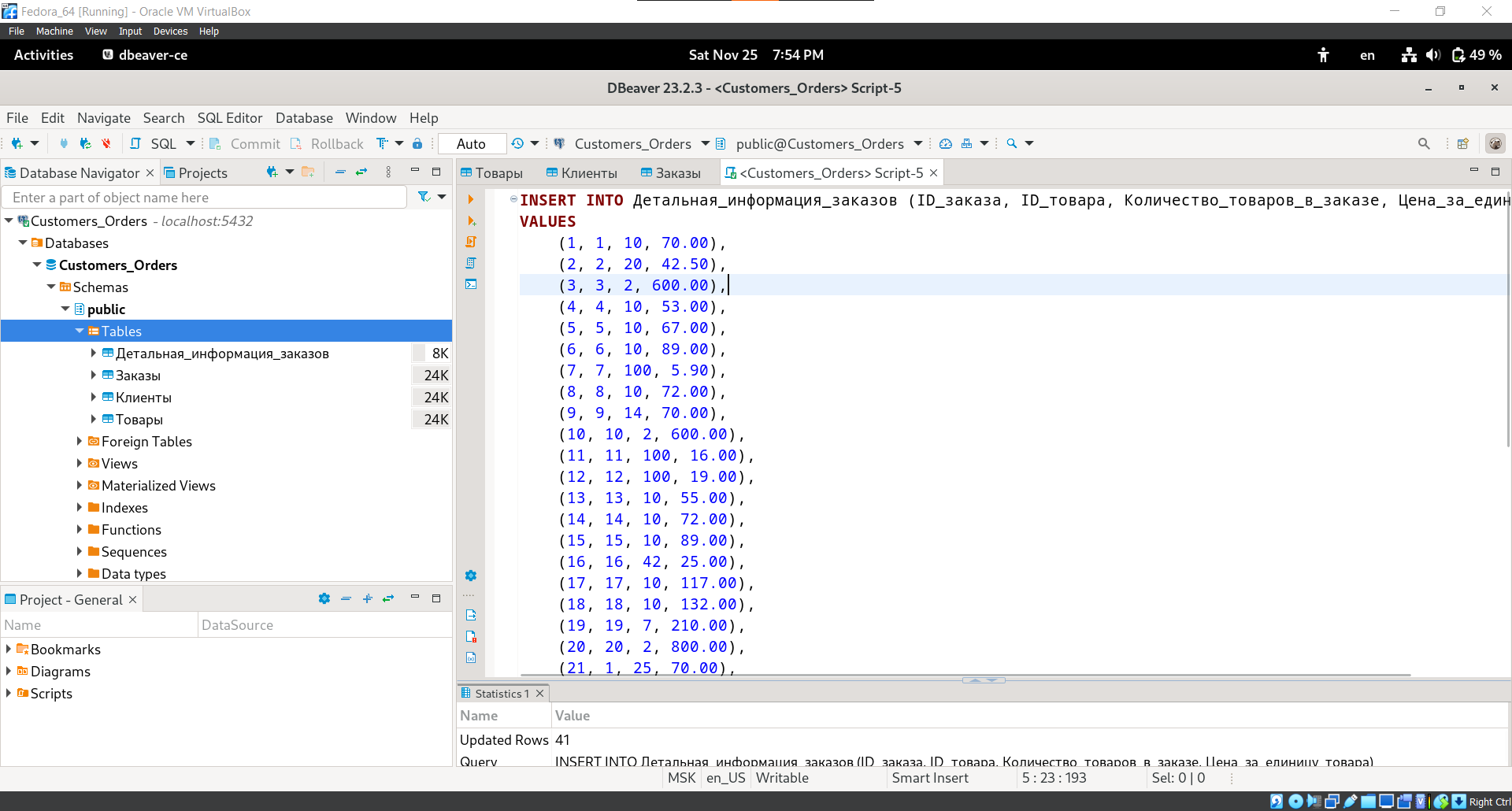


Рисунок 6 - Заполнение таблицы «Детальная информация заказов»

Далее, был сформирован SQL-запрос, который возвращает список клиентов (имя и фамилия) и суммарную стоимость заказов каждого клиента. По итогу, при выполнении данного запроса, отображаются столбцы с именем, фамилией и стоимостью (рисунок 7).

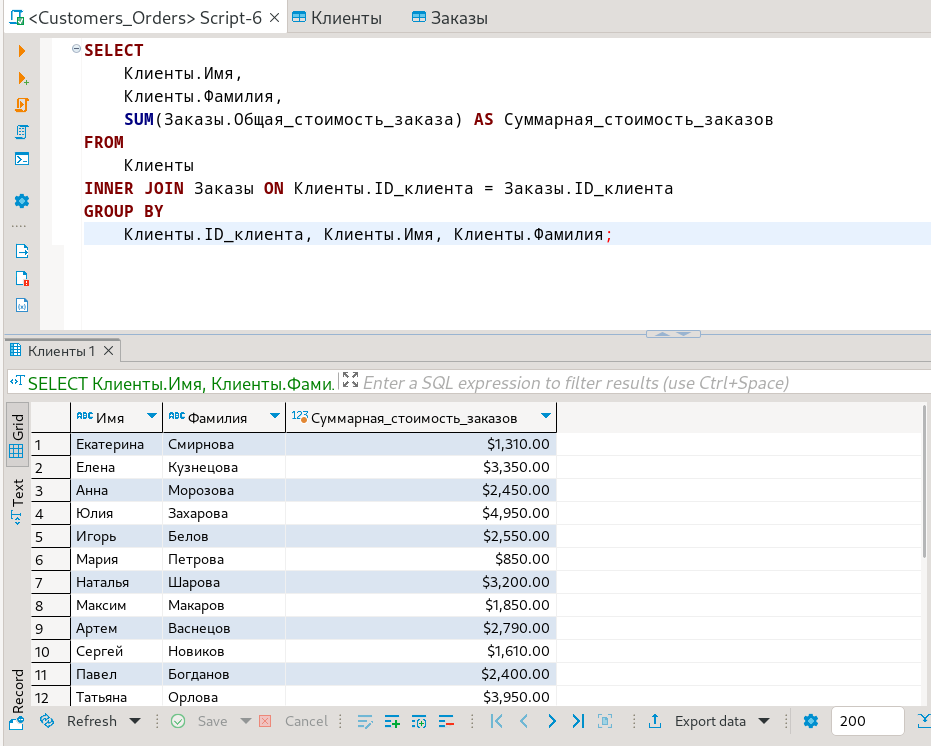


Рисунок 7 - Запрос на вывод списка клиентов с суммарной стоимостью заказов  
для каждого клиента

Запрос из рисунка 7 достаточно прост. После ключевого слова «SELECT» указываются поля, которые необходимо отобразить: фамилия, имя и суммарная стоимость всех заказов (агрегатная функция SUM вычислит это значение, а ключевое слово «AS» присвоит данному полю заголовок «Суммарная стоимость заказов»).  
Далее, ключевое слово «FROM» указывает из какой таблицы будут взяты данные, а далее, используется ключевое слово «INNER JOIN» (внутреннее объединение  
- возвратит только те строки, где есть совпадение по указанным ключам в обеих таблицах), которое объединяет таблицу «Клиенты» с таблицей «Заказы» по полю «ID\_клиента» и уже из этой «объединённой» таблицы по итогу будут браться данные. Далее, указано ключевое слово «GROUP BY», которое группирует всех клиентов по их ID, Имени и Фамилии. Делается это для того, чтобы агрегатная функция «SUM», считала суммарную стоимость заказа отдельно для каждого клиента, а не для всех клиентов разом.

Далее, в соответствии с заданием, необходимо было отсортировать полученный список клиентов по убыванию суммарной стоимости заказов клиента. Для этого в конец существующего запроса было добавлено ключевое слово «ORDER BY», где далее было указано поле, по которому производится сортировка, а также указывается направление сортировки - по убыванию (DESC) или по возрастанию (ASC). Результат выполнения обновлённого запроса показан на рисунке 8.

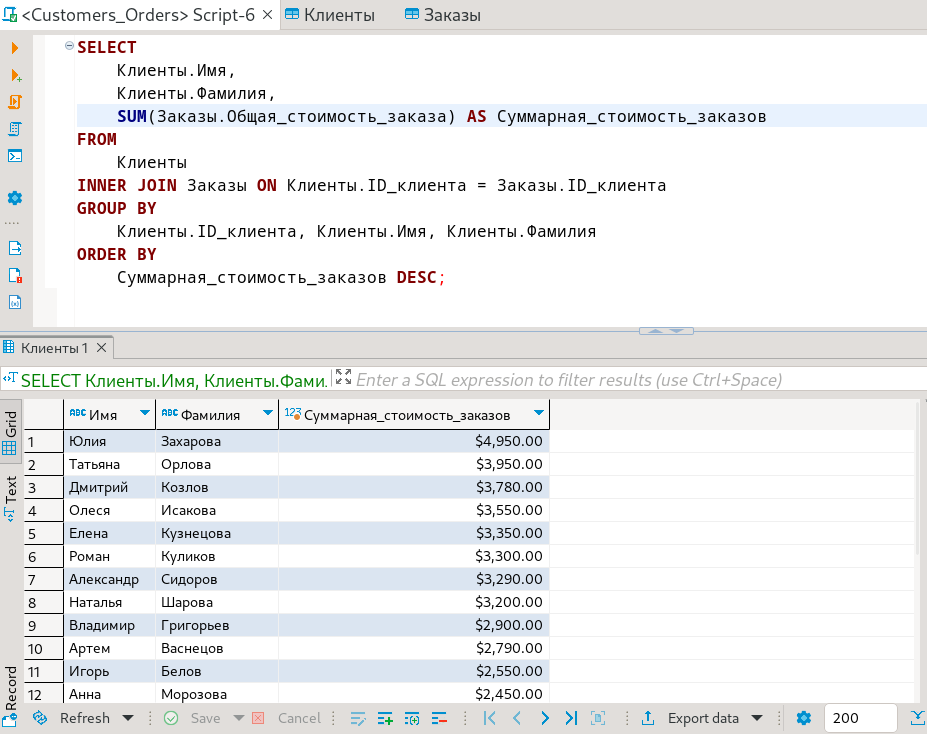


Рисунок 8 - Запрос на вывод списка клиентов с суммарной стоимостью заказов  
для каждого клиента с сортировкой по убыванию стоимости заказов клиента

Далее, необходимо было дополнительно вывести столбец со средней суммарной стоимостью заказа для каждого клиента. Для этого, в существующий запрос была добавлена агрегатная функция «AVG» как 4-е поле оператора SELECT. Также, так как поле «Общая стоимость заказа» имеет тип данных «money», то для корректной работы функции AVG необходимо явно указать, что использовать это поле нужно как тип данных «NUMERIC». Результат выполнения обновлённого запроса показан на рисунке 9.

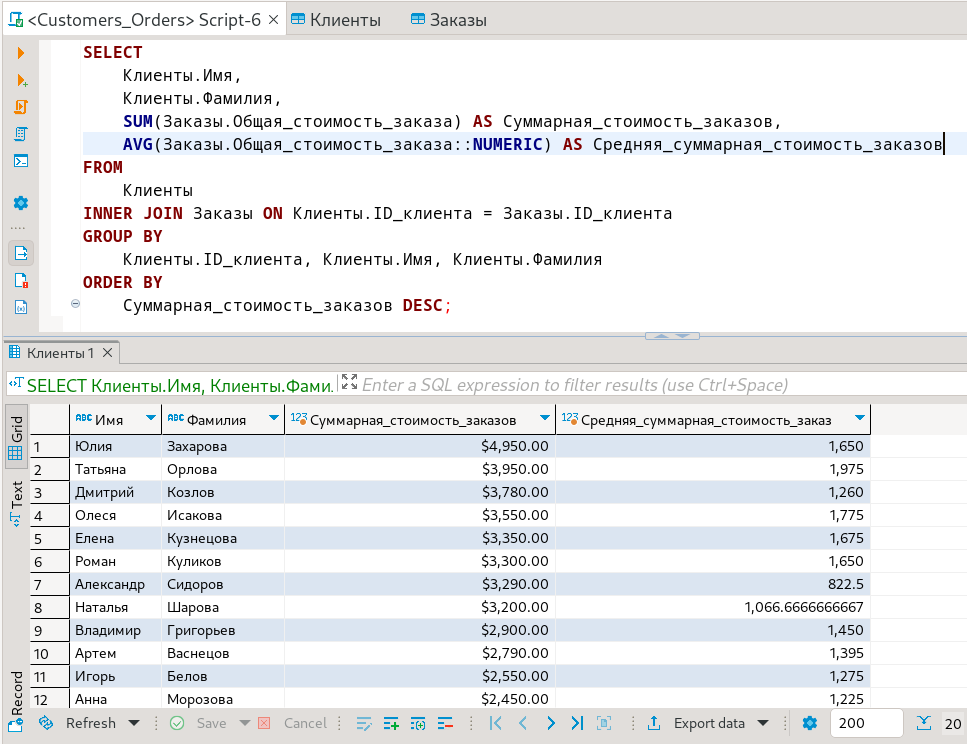


Рисунок 9 - Запрос на вывод списка клиентов с суммарной стоимостью заказов и средней суммарной стоимостью заказа для каждого клиента  
с сортировкой по убыванию стоимости заказов клиента

Далее, по заданию, необходимо было сформировать запрос, который выводил бы клиента (список клиентов) с наибольшей суммарной стоимостью заказов  
(имя, фамилия, стоимость). Для этой задачи был сформирован запрос, который изображён на рисунке 10.

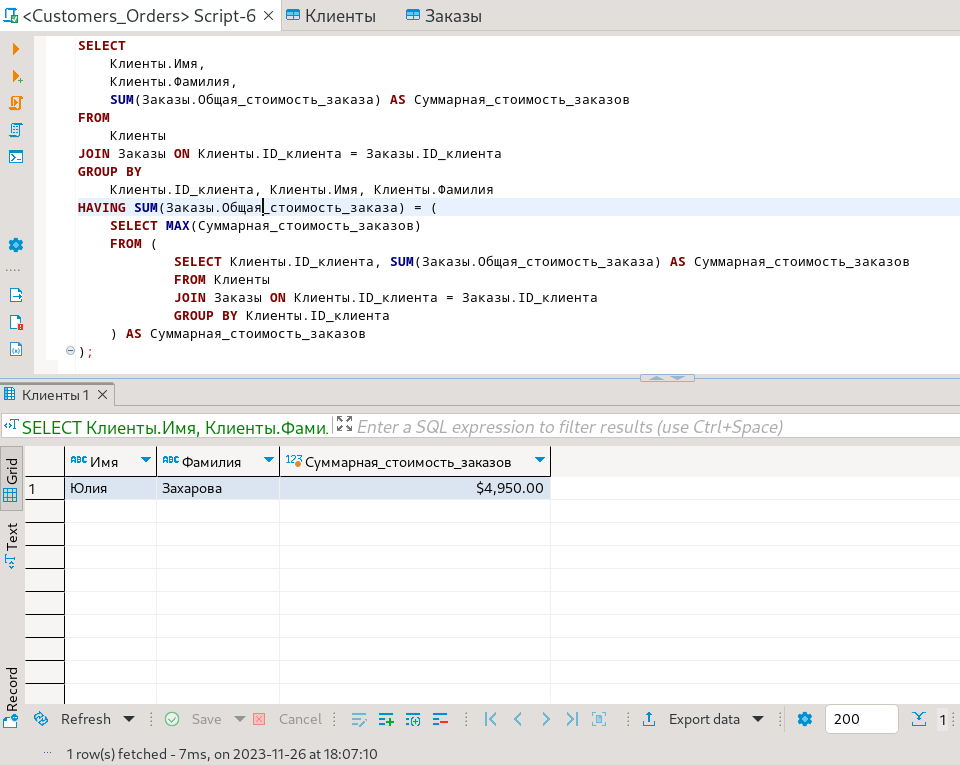


Рисунок 10 - Запрос на вывод клиента с наибольшей суммарной стоимостью заказа

Запрос с рисунка 10 был получен путём модернизации запроса с рисунка 9. Для начала была убрана сортировка (ORDER BY) и агрегатная функция «AVG», т.к. по заданию сортировка и среднее значение суммарной стоимости заказа не требуется. Далее, в конец запроса было добавлено ключевое слово «HAVING», которое выполнит фильтрацию клиентов, которые были сгруппированы по критериям из  
«GROUP BY». В левой части от знака «равно» оператора HAVING рассчитывается суммарная стоимость заказа конкретного клиента, которое потом будет сравниваться с максимумом.

Так как максимальное значение может быть только одно, его необходимо сначала рассчитать. Для этого, к оператору HAVING был сформулирован подзапрос, который вычисляет максимальную суммарную стоимость заказов среди всех клиентов, с помощью функции «MAX». Стоит отметить, что данный подзапрос содержит в себе ещё один – вложенный подзапрос, который выполняет агрегацию данных по клиентам, находит суммарную стоимость заказов для каждого клиента. Этот «вложенный»  
подзапрос используется в операторе FROM с псевдонимом  
«Суммарная стоимость заказов». Далее, после выполнения подзапросов в правой части от знака «равно» оператора HAVING будет максимальное значение суммарной стоимости заказа, по которому оператор HAVING отфильтрует только те строки с клиентами, где суммарная стоимость заказов равна этому максимальному значению.

По итогу, при выполнении данного запроса, был получен лишь один клиент – Юлия Захарова с суммарной стоимостью заказов в 4,950$.

Далее, по заданию необходимо было для каждого клиента с наибольшей суммарной стоимостью заказов вывести список его заказов (номер заказа и стоимость) в порядке возрастания стоимости заказа. Для этого был сформулирован SQL-запрос, который изображён на рисунке 11.

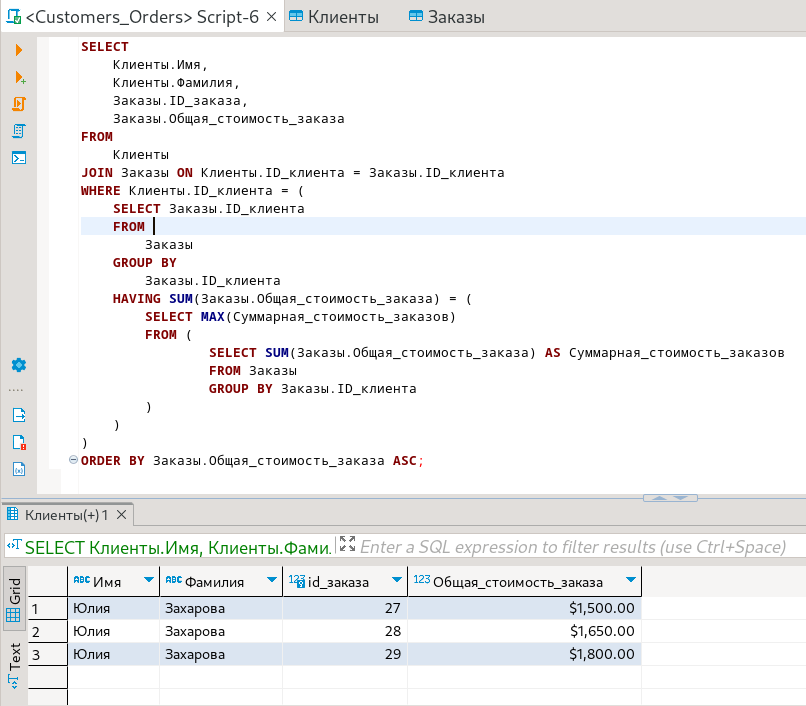


Рисунок 11 - Для клиента с наибольшей суммарной стоимостью заказа  
выводится список его заказов (номер заказа и стоимость)  
в порядке возрастания стоимости заказа

Запрос с рисунка 11 выводит имя, фамилию клиента с наибольшей (максимальной) суммарной стоимостью заказов, номера его заказов и общую стоимость каждого заказа. Соответственно, эти поля и были указаны в операторе SELECT. Данные брались из объединения таблиц Клиенты и Заказы по полю ID\_клиента. Также, необходимо из всех клиентов выполнить фильтрацию только по тому клиенту, который имеет наибольшую суммарную стоимость заказов. Для этого, использовался оператор «WHERE», который сравнит ID\_клиента с тем ID, который возвращает подзапрос.  
А подзапрос выполняет поиск клиента с наибольшей суммарной стоимостью заказов из таблицы «Заказы», где клиенты сгруппированы по ID, а оператор HAVING выполнит аналогичную фильтрацию по группам клиентов, как и в предыдущем запросе с рисунка 10. Далее. Оператор ORDER BY отсортирует список в порядке возрастания стоимости заказа.

Далее, по заданию необходимо было вывести только тех клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов (имя, фамилия, суммарная стоимость заказов клиента, средняя стоимость заказа). Для этого, был сформулирован SQL-запрос, который показан на рисунке 12. Стоит отметить важную деталь, фильтрация клиентов проводилась по средней суммарной стоимости заказов среди всем клиентов (1265.83…), но запрос с рисунка 12 выводит также и среднюю суммарную стоимость заказа для каждого клиента для большей наглядности работы запроса.

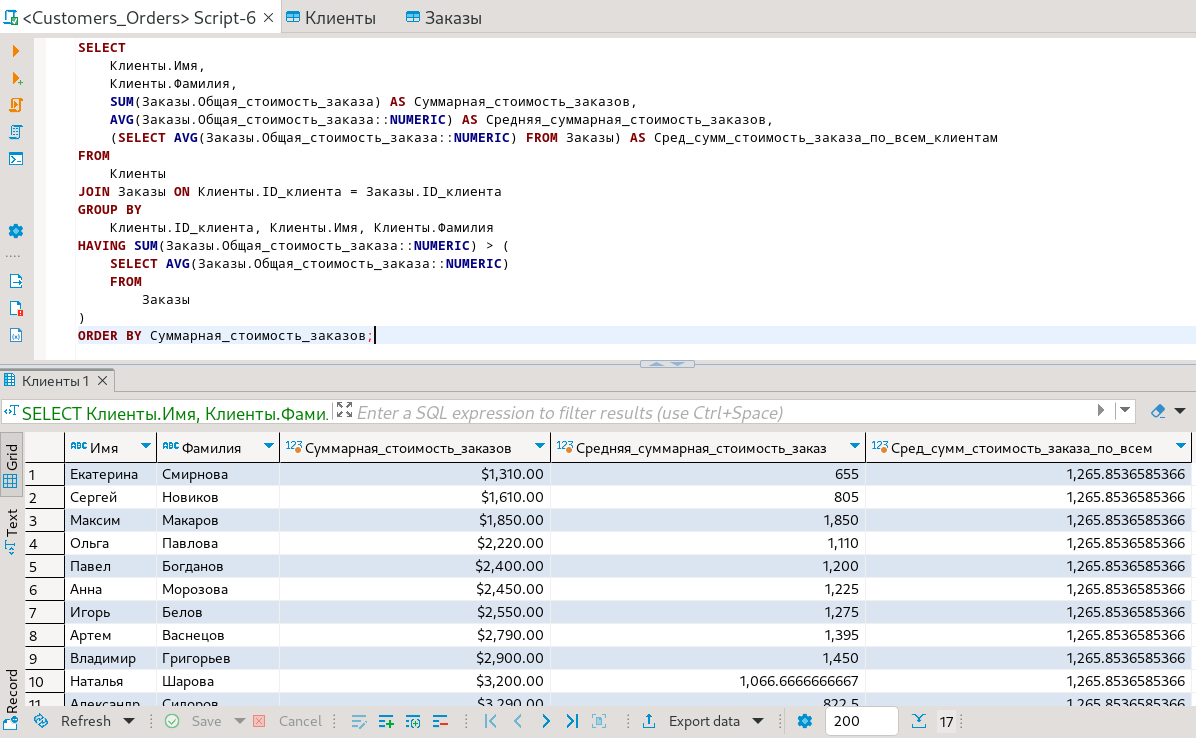


Рисунок 12 - Клиенты, чья суммарная стоимость заказов превышает среднюю  
суммарную стоимость заказов клиентов

Запрос с рисунка 12 состоит из оператора SELECT, где перечислены выводимые поля: Имя, Фамилия, Суммарная стоимость заказов одного конкретного клиента, Средняя суммарная стоимость заказов одного конкретного клиента, и подзапрос, который выведет Среднюю суммарную стоимость заказа среди всех клиентов (по этому значению и происходит фильтрация клиентов). Далее, указаны из каких таблиц брать данные (объединение Клиенты и Заказы по полю ID\_клиента). Далее, перечислены поля для группировки клиентов (фамилия, имя, ID). Далее, в операторе HAVING выполняется фильтрация клиентов по суммарной стоимости заказов. Если данное значение будет больше, чем среднее значение по всем клиентам (получено в подзапросе), то клиент отобразится в итоговом списке. Далее, оператор ORDER BY отсортирует список по возрастанию Суммарной стоимости заказов клиента.

Все SQL-запросы, перечисленные выше, были сохранены в файле «script\_lab9.sql» в порядке своего исполнения.

Далее, по заданию необходимо было сформулировать задачу, аналогичную изложенной выше (в пунктах 4 – 12 учебной задачи), на основе базы данных, разработанной в предыдущей лабораторной работе в соответствии с индивидуальным заданием. Для своей задачи разрешается создать новую базу данных либо дополнить уже имеющуюся.

Для выполнения этой задачи, использовалась база данных, из прошлой лабораторной работы – «Оптовые продажи». Уже имелось несколько SQL-запросов на создание 5-ти таблиц данной БД. Эти запросы были выполнены, и далее необходимо было заполнить БД данными.

Для начала была заполнена таблица «Единицы измерения» (рисунок 13). Далее, была заполнена таблица «Товары» (рисунок 14). Далее, была заполнена таблица «Продавцы» (рисунок 15). Далее, была заполнена таблица «Товары\_Продавцы»  
(рисунок 16). Далее, была заполнена таблица «Продажи» (рисунок 17).

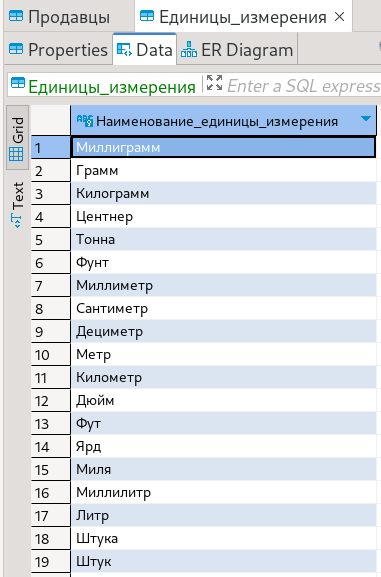


Рисунок 13 - Заполненная таблица «Единицы измерения»

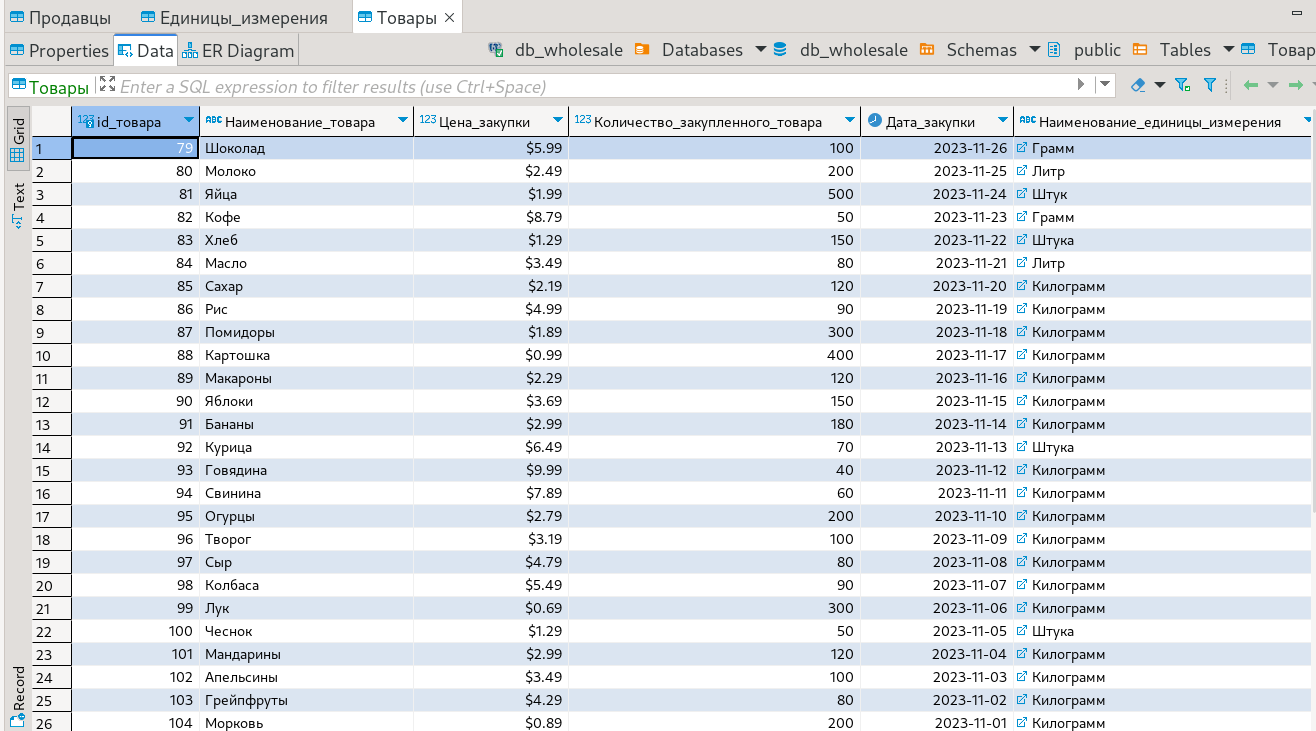


Рисунок 14 - Заполненная таблица «Товары»

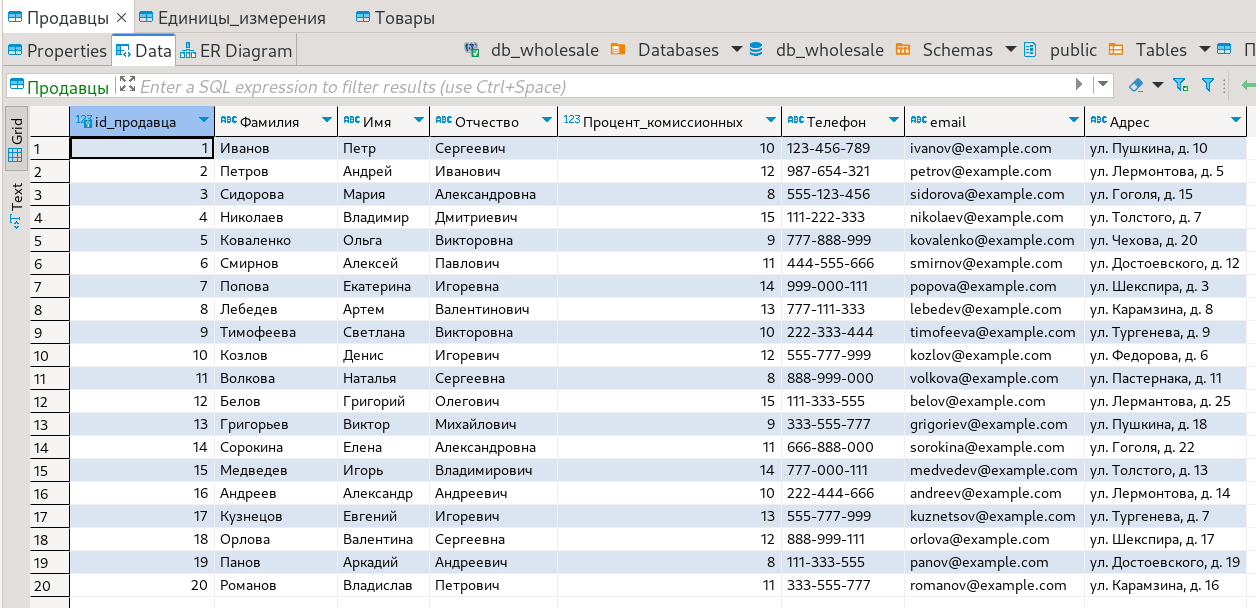


Рисунок 15 - Заполненная таблица «Продавцы»

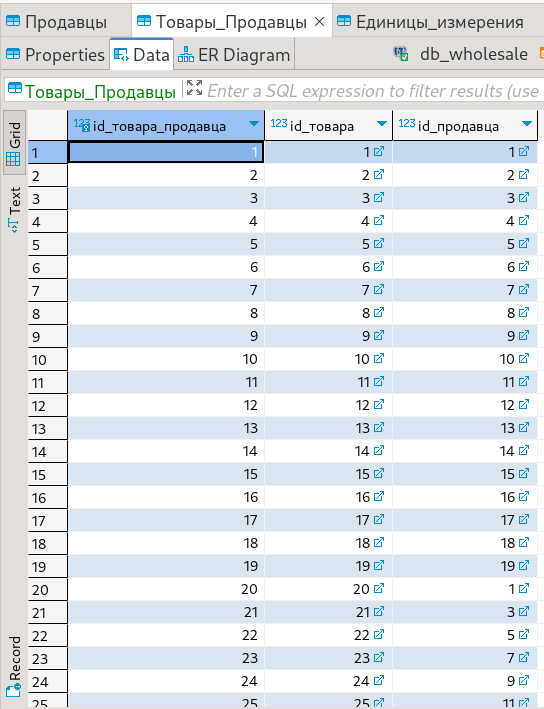


Рисунок 16 - Заполненная таблица «Товары\_Продавцы»

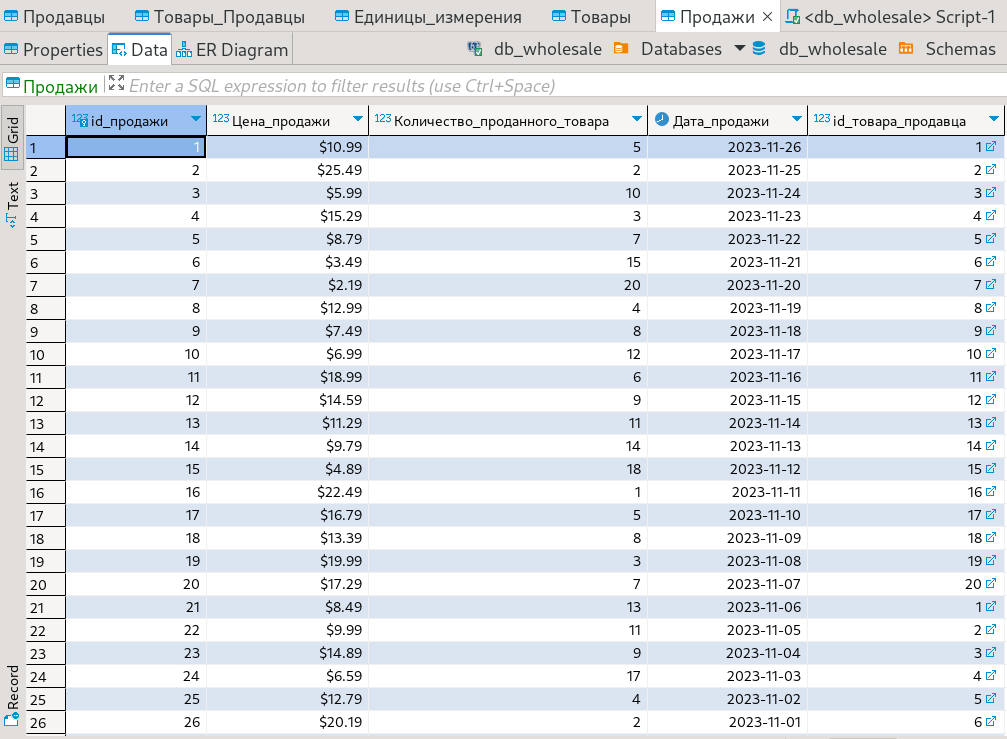


Рисунок 17 - Заполненная таблица «Продажи»

Далее, для данной базы данных была сформулирована следующая задача:   
вывести продавцов (фамилия, имя, отчество, суммарное число продаж продавца), у которых суммарное число продаж превышает средний показатель по всем продавцам. Также, необходимо вывести размер общей выручки, которую получит фирма с продажи товаров, а также вывести размер денежного вознаграждения, которое получит продавец от этой выручки в соответствии с его процентом комиссионных  
(общая выручка фирмы, доход продавца). Полученный список необходимо отсортировать по убыванию общей выручки фирмы.

Для получения необходимой информации в соответствии со сформулированной задачей, был разработан SQL-запрос, который представлен на рисунке 18.  
Результат выполнения данного запроса представлен на рисунке 19.

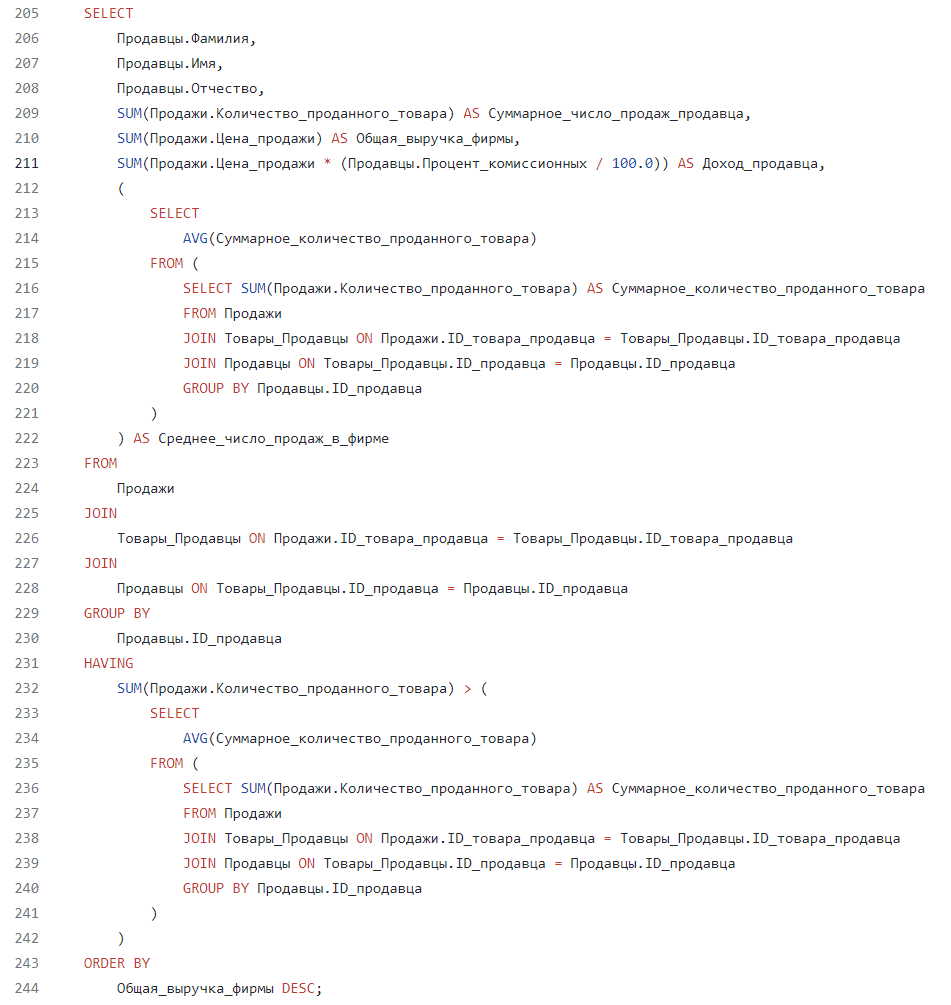


Рисунок 18 - Полный текст SQL-запроса для индивидуальной задачи

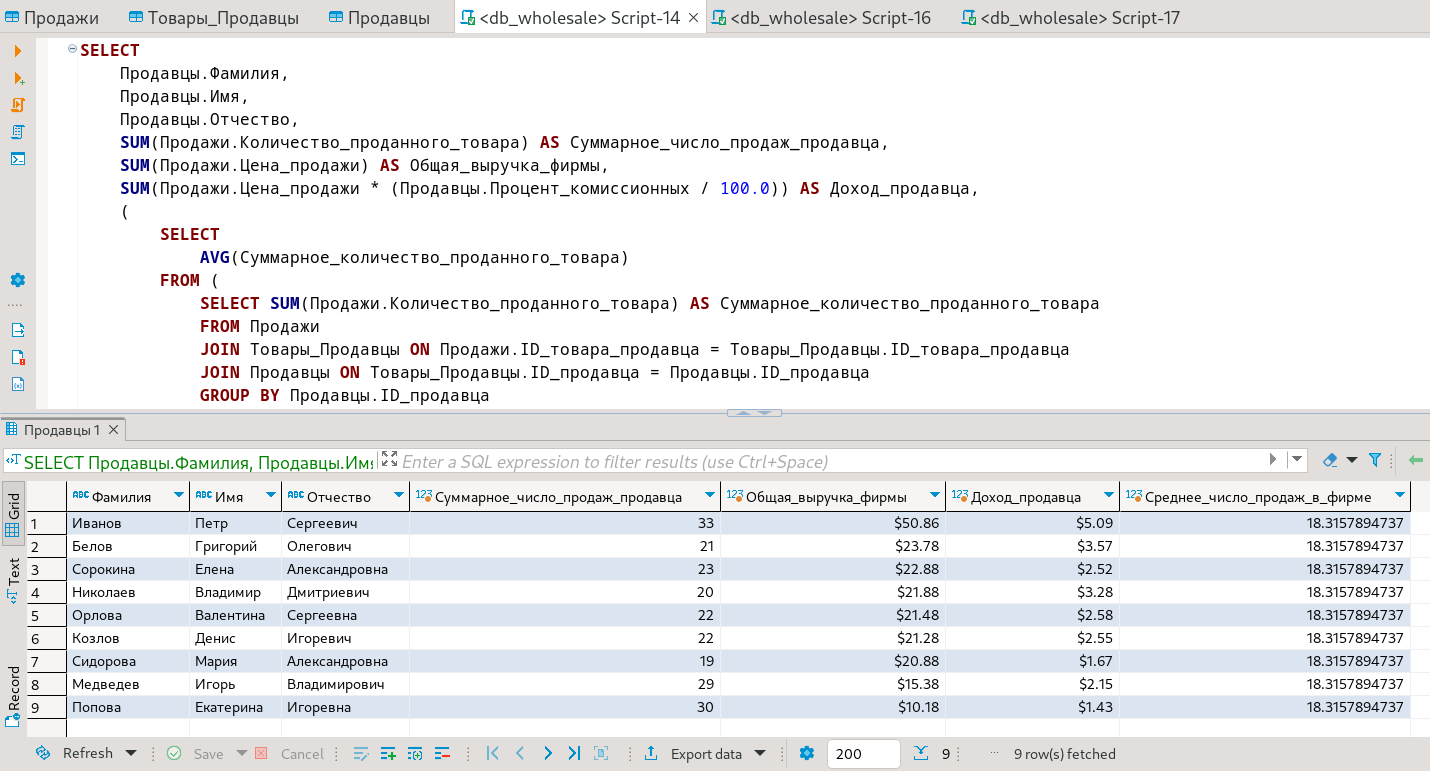


Рисунок 19 - Результат выполнения SQL-запроса для индивидуальной задачи

Запрос с рисунка 18 и рисунка 19 начинается с оператора SELECT, где перечисляются поля фамилии, имени и отчества продавца. Также там перечислены три агрегатные функции SUM, которые рассчитывают, соответственно, «Суммарное число продаж», «Общую выручку фирмы» и «Доход продавца». Далее, следом, идёт подзапрос, который дополнительно, для наглядности, выведет среднее число продаж по всем продавцам (функция AVG рассчитает это значение на основе списка с числом продаж для каждого продавца).

Оператор SELECT делает эту выборку из объединения таблиц «Продажи»,  
«Товары\_Продавцы» и «Продавцы». Далее, в операторе GROUP BY указаны категории, по которым будут работать вышеописанные функции SUM, а именно, эти функции рассчитают свои значения для каждого отдельного продавца (поле ID\_продавца).

Далее, полученный список будет отфильтрован оператором HAVING, те продавцы, у которых суммарное число продаж меньше среднего по фирме, будут отсеяны. Далее, оператор ORDER BY отсортирует полученный список по полю «Общая выручка фирмы».

Далее, в соответствии с заданием, все SQL-запросы для индивидуальной задачи были сохранены в отдельном файле – «script\_lab8.sql».

При выполнении лабораторной работы, в файле «docker-compose.yml»  
(рисунок 20) были обёрнуты две отдельные базы данных. Первая используется для основной задачи лабораторной работы, а вторая для индивидуального задания. В каждой из БД, при запуске контейнера, выполнятся соответствующие скрипты с SQL-запросами («script\_lab9.sql» и «script\_lab8.sql») в файле инициализации БД («init.sql»). Каждая из БД имеет свой порт и при подключении это нужно учитывать.

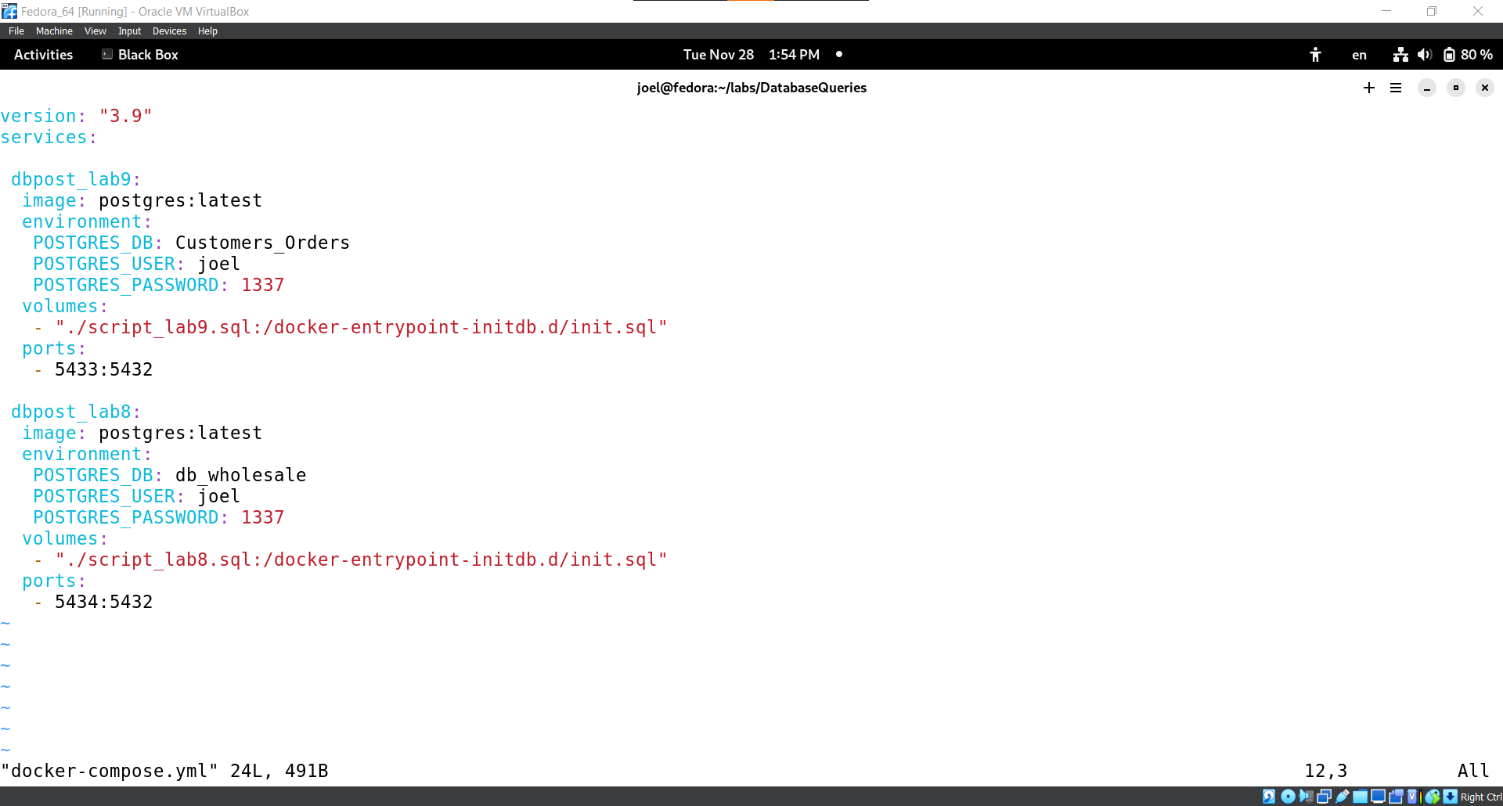


Рисунок 20 - Файл «docker-compose.yml»

После исполнения команды «docker compose up -d» в терминале, в Docker Desktop контейнер будет запущен и появится возможность подключения одновременно к двум базам данных (рисунок 21).

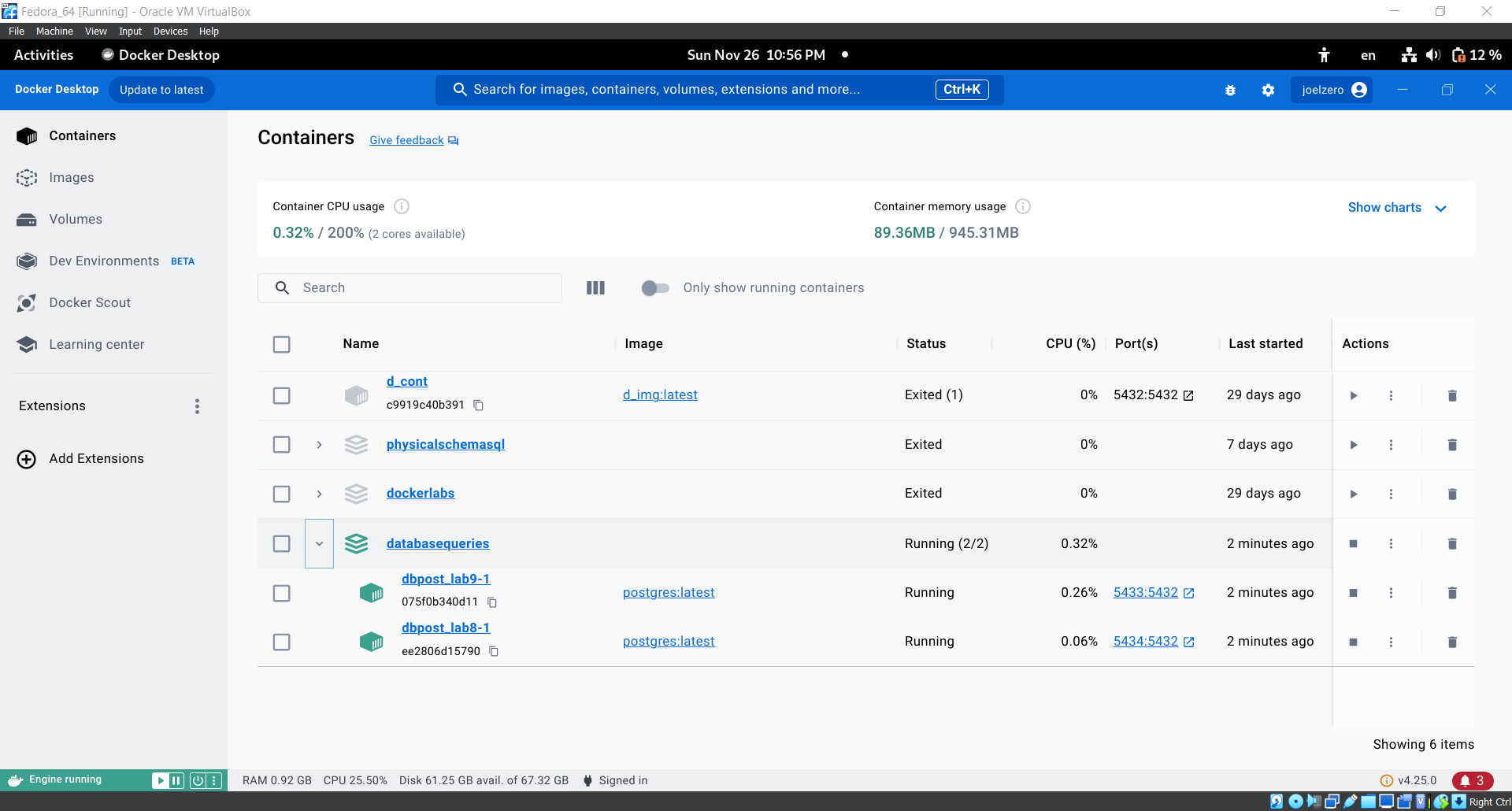


Рисунок 21 - Запуск контейнера в Docker Desktop

**Ответы на контрольные вопросы:**

Вопрос 1. Что такое подзапрос?

Ответ. Подзапрос – это запрос, который включен внутри другого запроса. Он может использоваться в различных частях SQL, таких как SELECT, FROM, WHERE, и так далее. Идея заключается в том, чтобы выполнить один запрос (подзапрос) и использовать его результат в другом запросе.

Вопрос 2. Как заполнить таблицу данными?

Ответ. Заполнение таблицы данными можно выполнить с использованием оператора INSERT. Пример простого запроса для вставки данных в таблицу:  
«INSERT INTO имя\_таблицы (столбец1, столбец2, столбец3) VALUES (значение1, значение2, значение3);».

Вопрос 3. Агрегатные функции. Основные сведения.

Ответ. Агрегатные функции используются для выполнения вычислений по набору значений и возвращают единый результат. Эти функции обычно используются в комбинации с оператором GROUP BY для агрегации данных по группам.  
Агрегатными функциями являются такие функция как COUNT (вернёт количество строк в результате запроса), SUM (возвращает сумму значений в указанном столбце), AVG (среднее значение в указанном столбце), MIN (минимальное значение из столбца), MAX (максимальное значение из столбца).

Вопрос 4. Как объединить таблицы в запросе?

Ответ. Для объединения таблиц в запросе используют оператор JOIN.

Вопрос 5. Способы объединения таблиц. Краткая характеристика.

Ответ. Оператор JOIN предоставляет несколько способ своего использования. Варианты использования данного оператора представлены на рисунке 22.

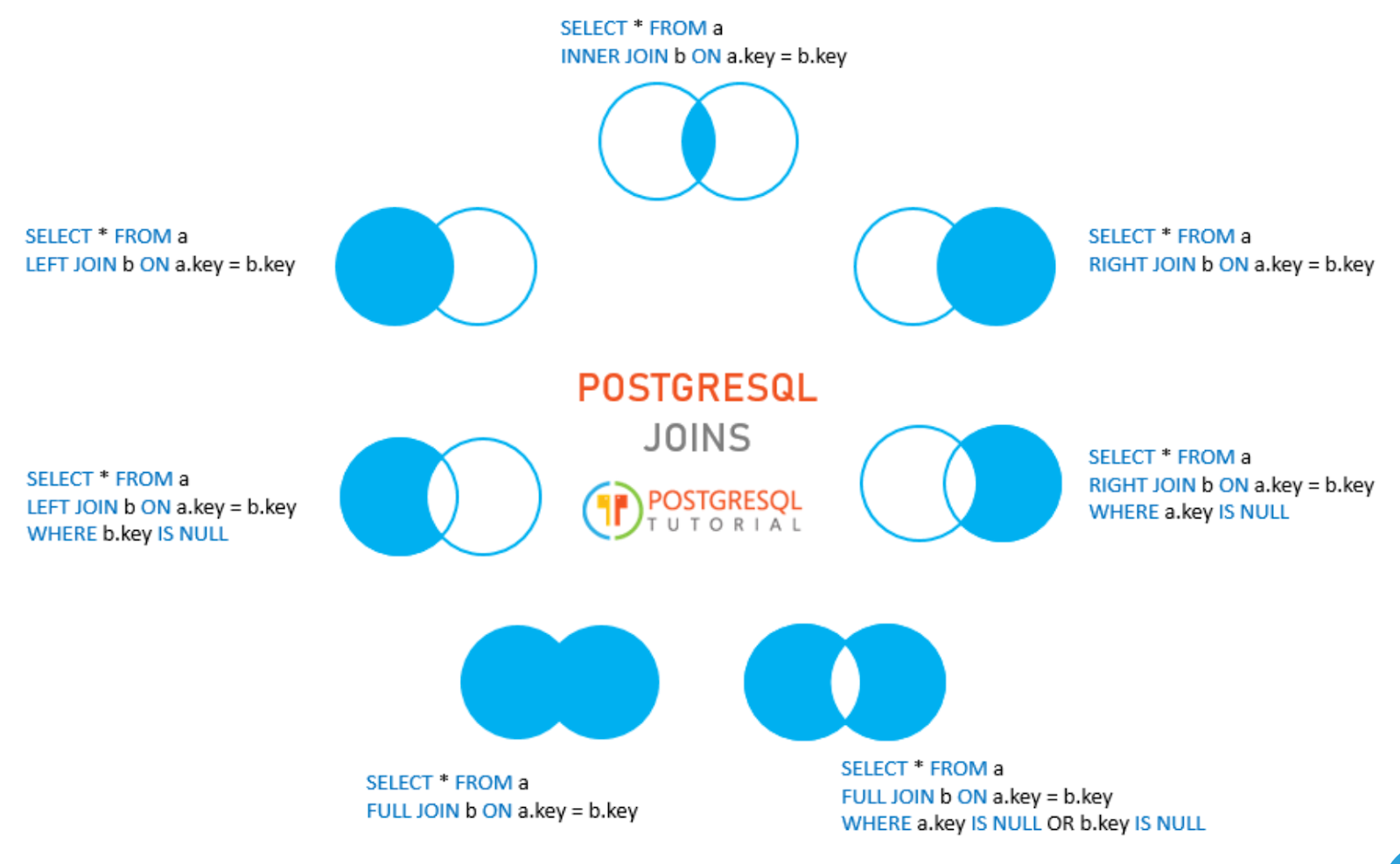


Рисунок 22 – Варианты использования оператора JOIN

Вопрос 6. План запроса. Краткое описание.

Ответ. План запроса (Query Execution Plan) представляет собой последовательность шагов, которые оптимизатор запросов выбирает для выполнения SQL-запроса. Оптимизатор оценивает различные способы выполнения запроса и выбирает наилучший, учитывая структуру таблиц, индексы и другие факторы.

Краткое описание элементов плана запроса:

1. Seq Scan (Последовательное сканирование): Полное сканирование таблицы по порядку.
2. Index Scan (Индексное сканирование): Использование индекса для быстрого поиска данных.
3. Hash Join (Хэш-объединение): Объединение таблиц с использованием хэш-функций.
4. Nested Loop (Вложенный цикл): Вложенный цикл, используемый для объединения таблиц.
5. Sort (Сортировка): Сортировка данных.
6. Aggregate (Агрегация): Вычисление агрегатных функций, таких как COUNT, SUM, AVG.
7. Bitmap Index Scan (Сканирование индекса по битовой карте): используется при наличии нескольких индексов.

План запроса помогает понять, как система будет извлекать и обрабатывать данные, что помочь в оптимизации производительности запроса. План запроса можно просмотреть, с помощью команды EXPLAIN перед SQL-запросом.

Вопрос 7. Оптимизация запроса. Краткое описание.

Ответ. Оптимизация запроса – это процесс улучшения производительности SQL-запроса путем изменения его структуры, использования индексов, выбора оптимальных алгоритмов выполнения и других методов.

Несколько стратегий оптимизации запросов:

1. Использование индексов.

Необходимо убедится, что таблицы имеют соответствующие индексы для  
ускорения поиска данных. Индексы могут быть созданы на столбцах, которые часто используются в условиях WHERE или JOIN.

1. Правильное использование инструкции WHERE.

Необходимо использовать наилучшие условия в инструкции WHERE, чтобы ограничить объем данных до минимума еще до выполнения других операций.

1. Ограничение выбираемых столбцов.

Необходимо выбирать только те столбцы, которые действительно нужны для результата, чтобы уменьшить объем данных, передаваемых между сервером и  
клиентом.

1. Правильное использование JOIN.

Необходимо наилучший вариант оператора JOIN в зависимости от структуры данных и требований запроса.

1. Оптимизация подзапросов.

Подзапросы могут быть медленными, поэтому необходимо писать их  
эффективно или же рассмотреть возможность объединения данных в один запрос.

1. Использование EXPLAIN.

Можно использовать команду EXPLAIN, чтобы посмотреть план выполнения запроса и идентифицировать места для улучшения.

1. Статистика и анализ использования ресурсов.

Можно использовать инструменты для анализа статистики выполнения  
запросов и выявления узких мест.

Вопрос 8. Оконные функции. Краткое описание.

Ответ. Оконные функции, как и агрегатные функции, работают с множеством строк, называемым рамкой окна. В отличие от агрегатных функций, оконные функции возвращают единственное значение для каждой строки рассматриваемого запроса. Окно определяется с использованием оператора OVER. Оно позволяет задать окно на основе конкретного столбца, подобно GROUP BY в случае агрегатных  
функций.

Вопрос 9. Команда и синтаксис для группировки данных.

Ответ. Группировка данных выполняется с использованием оператора GROUP BY в сочетании с агрегатными функциями. Синтаксис для использования данной команды:

SELECT column1, aggregate\_function(column2)

FROM table\_name

GROUP BY column1;

Вопрос 10. Сортировка данных. Краткое описание.

Ответ. Сортировка данных выполняется с использованием оператора ORDER BY в конце запроса. Оператор ORDER BY позволяет упорядочить результаты запроса по указанным столбцам в порядке возрастания (по умолчанию, ASC) или убывания (DESC).

Вопрос 11. Условия для выборки данных (WHERE, HAVING). Краткое  
описание.

Ответ. Условия для выборки данных могут быть заданы с использованием оператора WHERE и, в случае агрегированных данных, оператора HAVING.

Оператор WHERE используется для фильтрации строк на основе определенного условия и выполняется до работы оператора GROUP BY. Это нужно для того, чтобы можно было оперировать как можно меньшим количеством данных и сэкономить ресурсы сервера и время пользователя.

Оператор HAVING используется для фильтрации результатов агрегированных функций (например, COUNT, SUM, AVG) и выполняется после оператора  
GROUP BY, когда уже можно накладывать условия на результаты агрегатных  
функций.

**Вывод:** в ходе лабораторной работы был изучен синтаксис и основные  
команды для формирования запросов к базе данных. Также, был освоен процесс  
формирования SQL-запросов.