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: Тогушов В.А.

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.

Учебная задача:

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 в соответствии со следующей структурой:

Customers — информация о клиентах:

CustomerID (int, PK) — уникальный идентификатор клиента.

FirstName (varchar) — имя клиента.

LastName (varchar) — фамилия клиента.

Email (varchar) — адрес электронной почты клиента.

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.

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.

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.

Индивидуальное задание:

Выполнить все пункты раздела «Учебная задача», оформить отчёт, представить результаты выполнения лабораторной работы к защите.

Сформируем SQL-запросы для создания таблиц в СУБД Postgres с информацией о клиентах, заказах и деталей заказов (Рисунок 1).

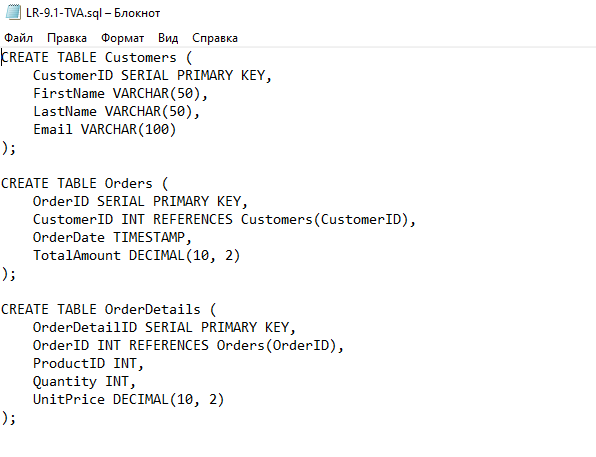


Рисунок 1 – Создание таблиц

Теперь обернем базу данных в docker-контейнер с помощью файла docker-compose.yml (Рисунок 2).

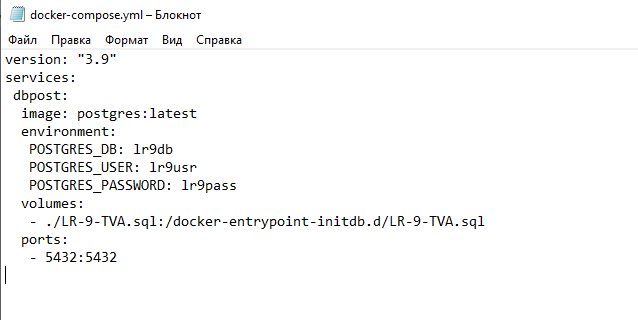


Рисунок 2 – Файл «docker-compose.yml»

Теперь запустим контейнер с помощью команды: docker-compose up.

Далее перейдем в DBeaver и подключимся к созданной базе данных. Приступим к заполнению таблиц клиентов, заказов и деталей заказов, для каждой около 20 строк.

Добавлять данные в таблицу можно, как перейдя в раздел «Данные», выбранной таблицы и через «Добавить запись» внести необходимые значения, так и с помощью SQL-запроса.

Напишем запрос на добавление данных в таблицу customers (Рисунок 3).

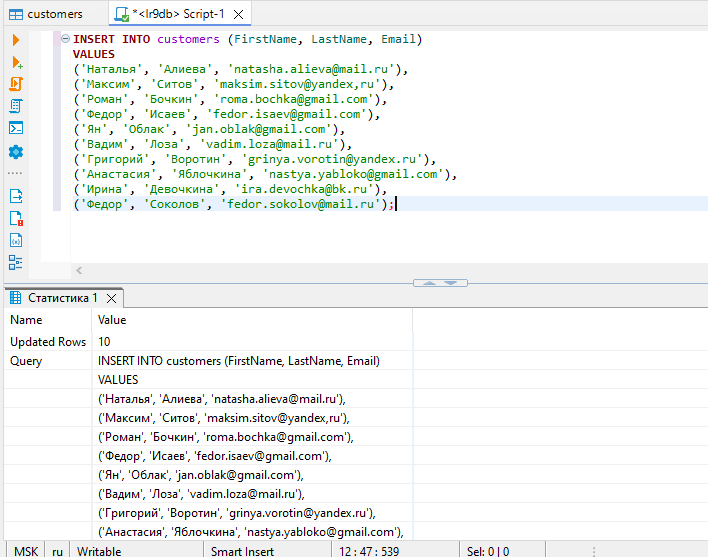


Рисунок 3 – Запрос на добавление данных в таблицу

Посмотрим, как добавились значения в таблицу в разделе «Данные» (Рисунок 4).

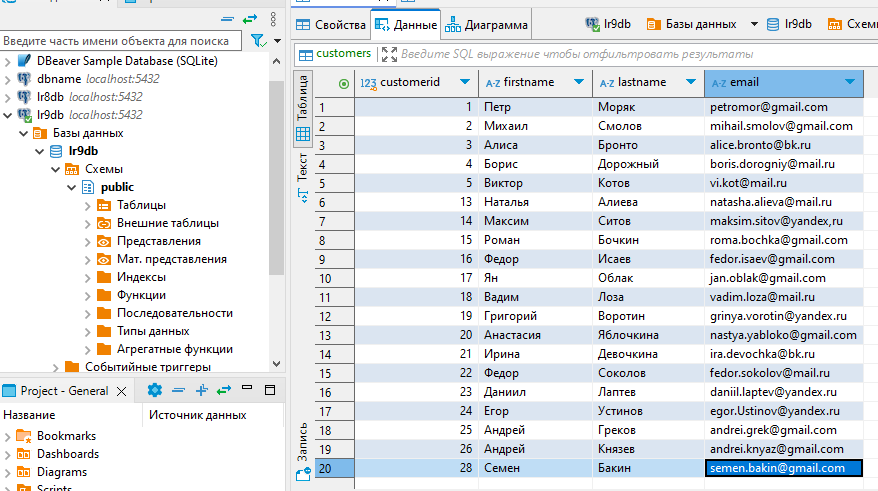


Рисунок 4 – Добавленные данные в таблицу Customers

Аналогично можем заполнить и остальные таблицы (Рисунок 5-6).

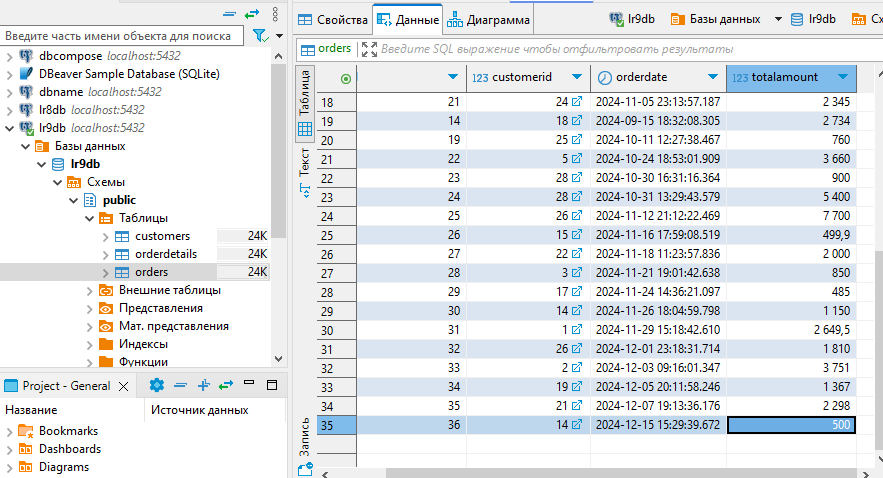


Рисунок 5 – Таблица Orders

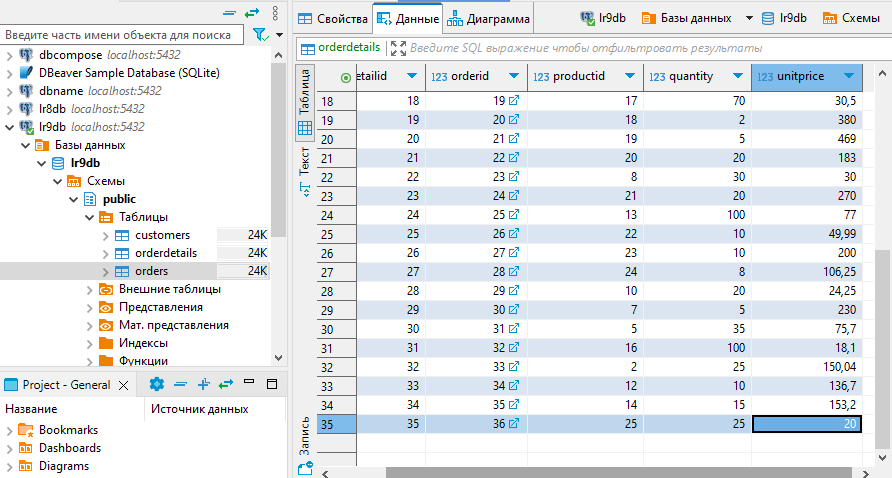


Рисунок 6 – Таблица OrderDetails

Теперь приступим к формированию запросов к нашей базе данных

Сделаем 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. Также мы сразу отсортируем полученный список по убыванию суммарной стоимости заказов клиента (Рисунок 7).

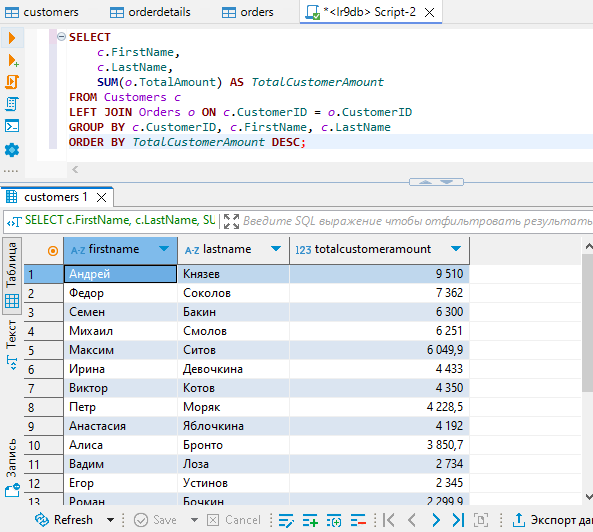


Рисунок 7 – Запрос на суммарную стоимость заказов каждого клиента

Далее добавим столбец со средней суммарной стоимостью заказов (Рисунок 8).

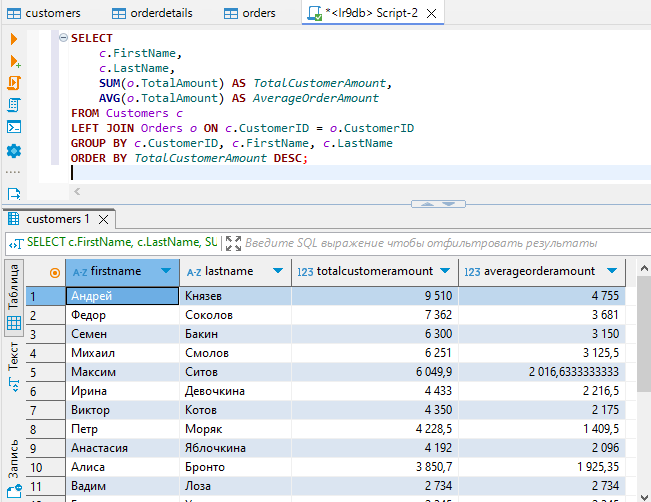


Рисунок 8 – Добавление столбца со средней стоимостью заказов каждого клиента

Выведем клиента (список клиентов) с наибольшей суммарной стоимостью заказов (имя, фамилия, стоимость) (Рисунок 9).

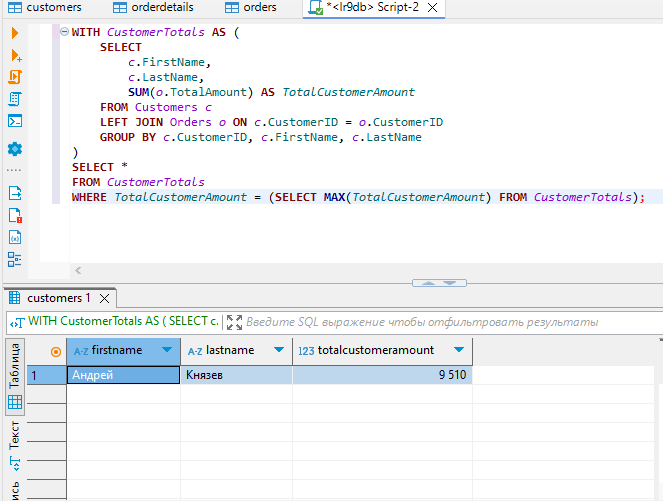


Рисунок 9 – Запрос на вывод клиента с наибольшей стоимостью заказов

Для каждого клиента с наибольшей суммарной стоимостью заказов выведем список его заказов (номер заказа и стоимость) в порядке возрастания стоимости заказа (Рисунок 10).

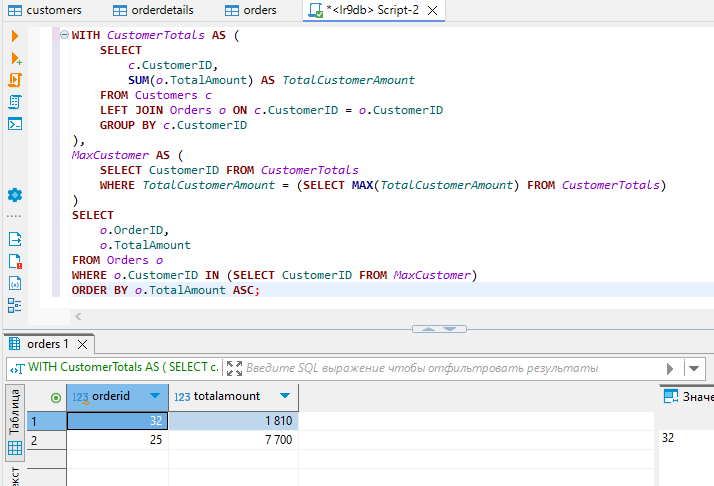


Рисунок 10 – Вывод заказов для каждого клиента с наибольшей их стоимостью

Выведем только тех клиентов, у которых суммарная стоимость заказов превышает среднюю суммарную стоимость заказов клиентов (имя, фамилия, суммарная стоимость заказов клиента, средняя стоимость заказа) (Рисунок 11).

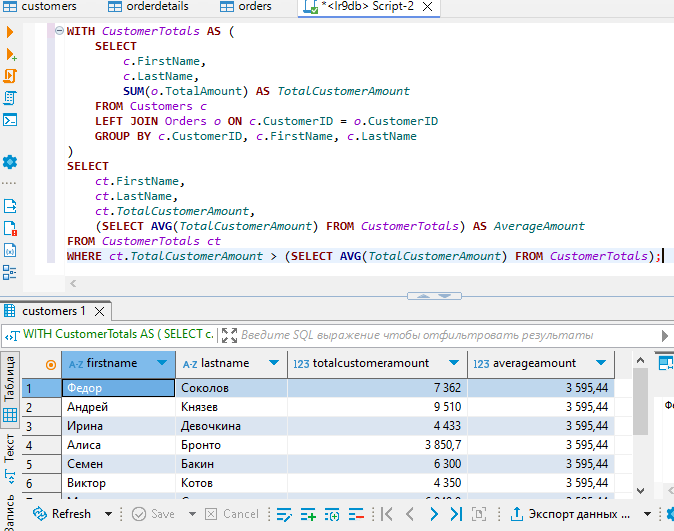


Рисунок 11 – Вывод клиентов, у которых суммарная стоимость заказов превышает среднюю

Сохраним SQL-запросы в соответствующем файле со скриптами SQL.

Сформулируем задачу, аналогичную тренировочной, на основе базы данных, разработанной в предыдущей лабораторной работе в соответствии с индивидуальным заданием.

Для начала заполним таблицы нашей базы данных (Рисунок 12-16).

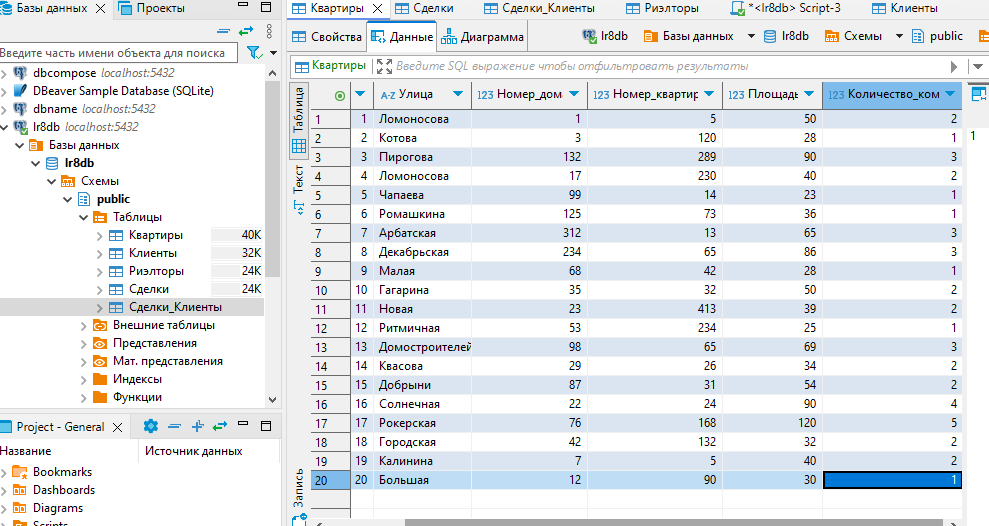


Рисунок 12 – Таблица «Квартиры»

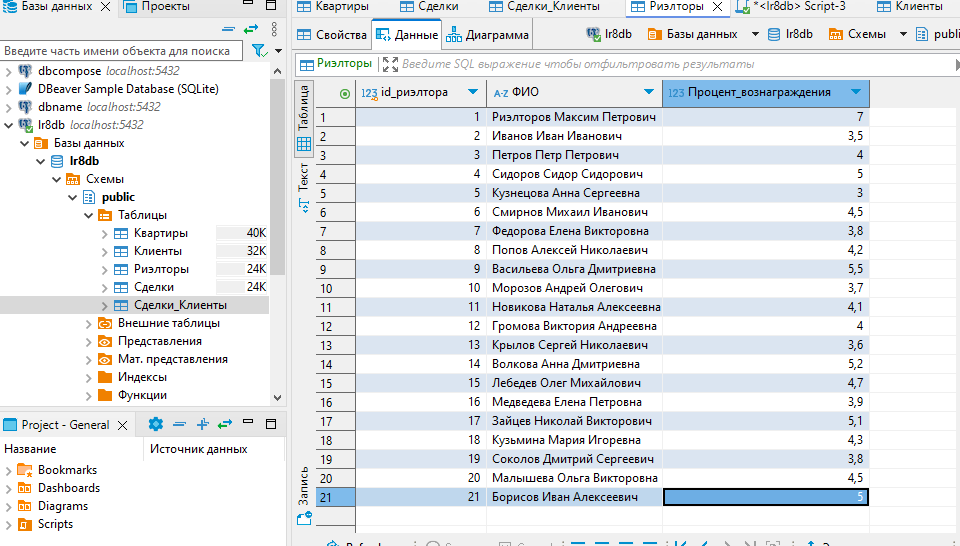


Рисунок 13 – Таблица «Риэлторы»

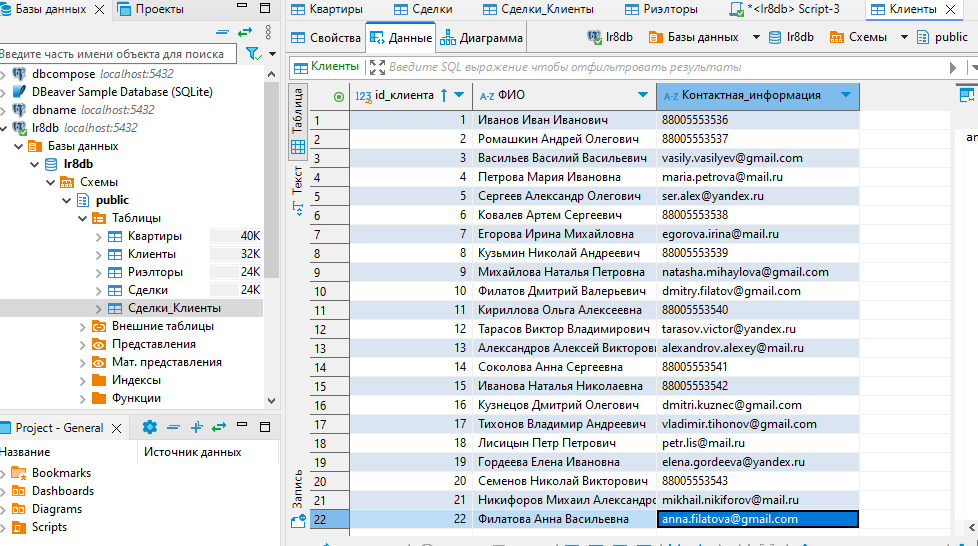


Рисунок 14 – Таблица «Клиенты»

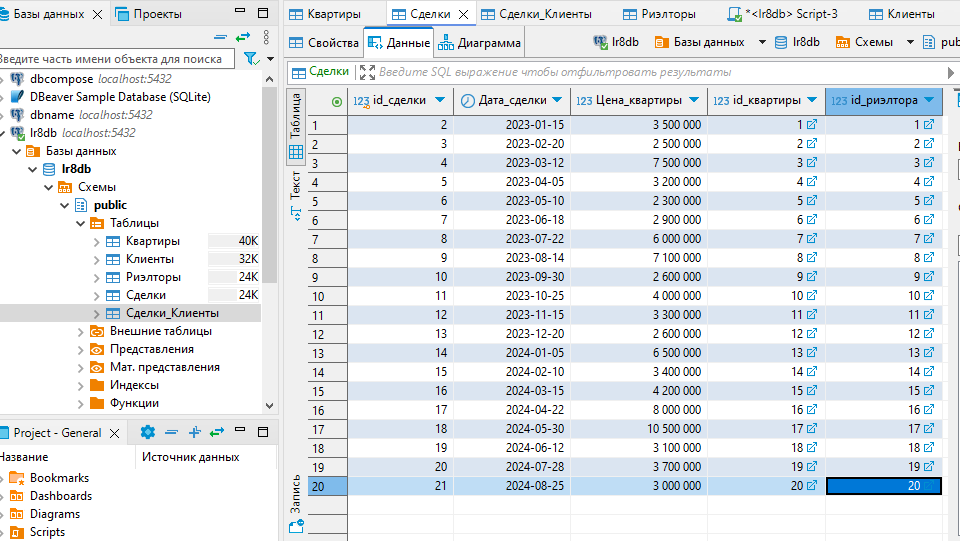


Рисунок 15 – Таблица «Сделки»

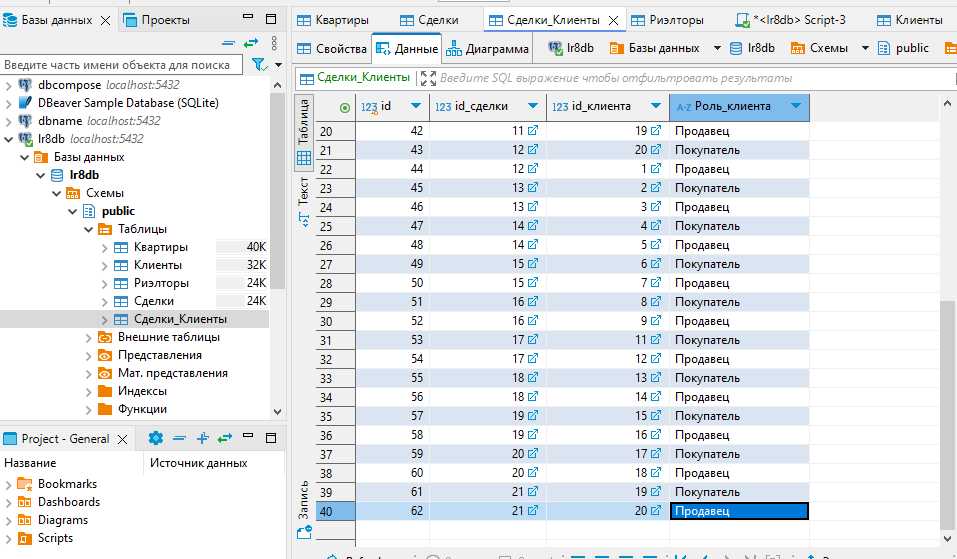


Рисунок 16 – Таблица «Сделки\_Клиенты»

Начнем с запроса, который возвращает список клиентов с их именами, контактной информацией и общей суммой сделок, в которых они участвуют, отсортированной по убыванию (Рисунок 17).

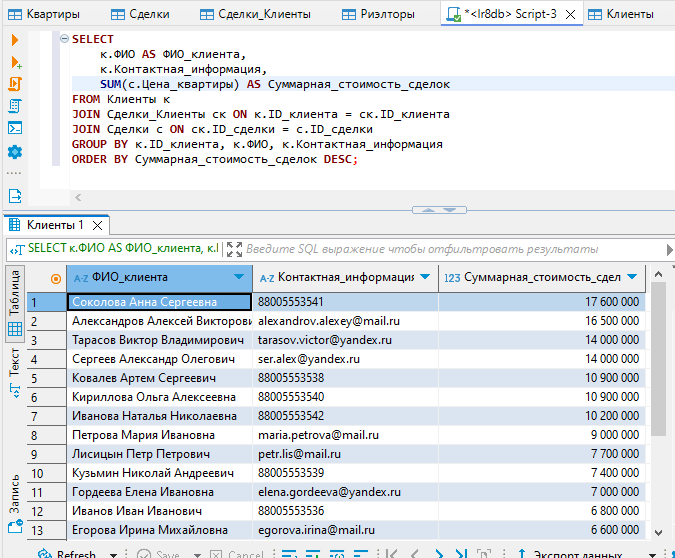


Рисунок 17 – Запрос на общую сумму сделок каждого клиента

Следующий запрос, помимо суммарной стоимости сделок, также вычисляет для каждого клиента среднюю стоимость сделки, в которой он участвует (Рисунок 18).

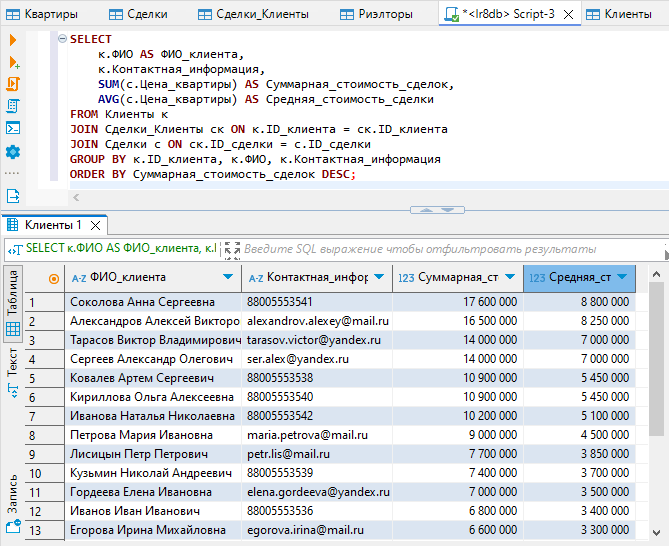


Рисунок 18 – Запрос на добавление столбца со средней стоимостью сделок

Выполним запрос, который находит клиента, или клиентов с наибольшей общей суммой сделок (Рисунок 19).

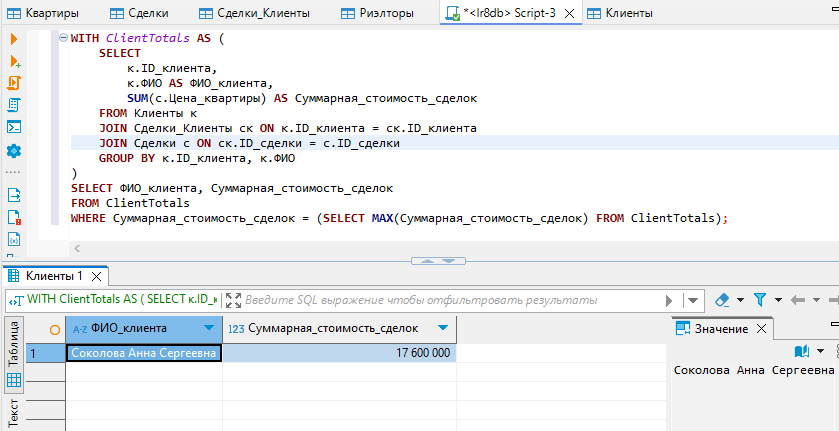


Рисунок 19 – Запрос на поиск клиентов с наибольшей общей суммой покупок

Также, для найденных клиентов с наибольшей общей суммой по сделкам, напишем запрос, который возвращает сделки этих клиентов с максимальной суммарной стоимостью, отсортированные по возрастанию цены квартиры (Рисунок 20).

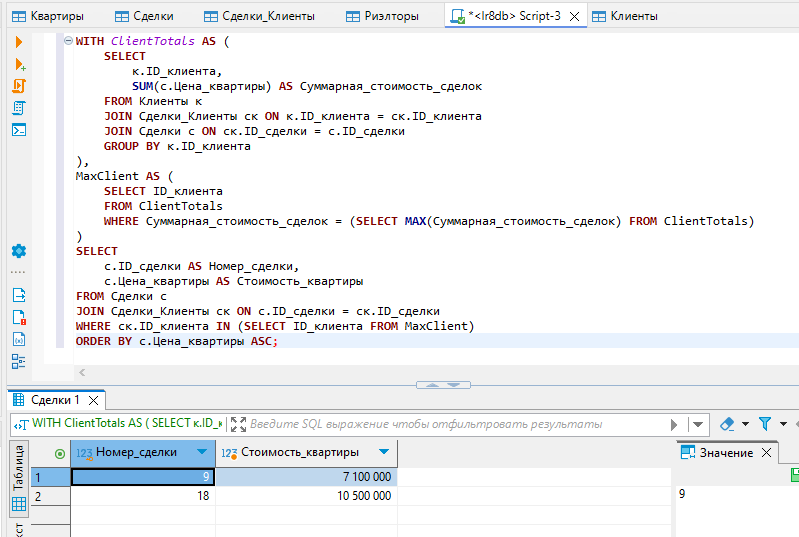


Рисунок 20 – Запрос, возвращающий сделки клиентов с максимальной общей суммой сделок

Риэлтор, оформивший сделку купли-продажи, получает комиссионное вознаграждение, которое вычисляется как:

Цена квартиры х Процент вознаграждения.

Сформируем запрос, который возвращает список риелторов с подсчитанными суммарными комиссионными вознаграждениями (Рисунок 21).

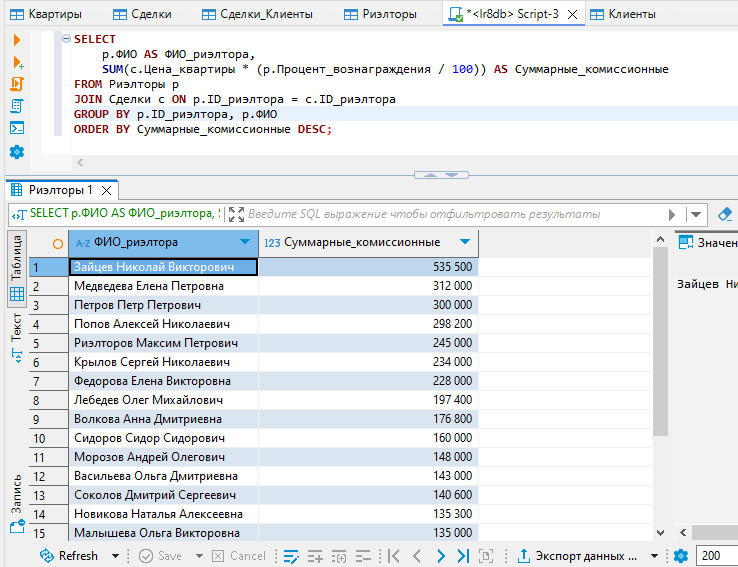


Рисунок 21 – Запрос на вывод списка риэлторов с суммарными комиссионными вознаграждениями

Также сохраним SQL-запросы по индивидуальному заданию в соответствующем файле со скриптами SQL и сделаем описание с комментариями.

Контрольные вопросы:

1) Что такое подзапрос?

Подзапрос в SQL – это SELECT-запрос, вложенный в другой запрос или подзапрос.

Он используется, когда нужно использовать результат выполнения одного запроса в следующем запросе.

2) Как заполнить таблицу данными?

Для заполнения таблицы данными в SQL используется оператор INSERT INTO.

Синтаксис:

INSERT INTO имя\_таблицы (поле1, поле2, ...) VALUES (значение\_поля\_1, значение\_поля\_2, ...);

3) Агрегатные функции. Основные сведения.

Агрегатные функции – это специализированные функции, которые выполняют вычисления над группами переменных и возвращают единый результат.

Основные агрегатные функции в языке SQL:

1. COUNT. Считает количество строк в таблице или не-NULL значений полей, которые выбрал запрос.

2. SUM. Вычисляет арифметическую сумму всех выбранных значений данного поля.

3. AVG. Вычисляет среднее значение столбца.

4. MAX. Возвращает наибольшее из всех выбранных значений данного поля.

5. MIN. Возвращает наименьшее из всех выбранных значений данного поля.

Агрегатные функции используются для построения сложных запросов, отчётов и визуализации анализа данных в различных приложениях.

4) Как объединить таблицы в запросе?

Для объединения таблиц в запросе можно использовать команду UNION. Она объединяет данные из нескольких таблиц в одну при выборке. При объединении количество столбцов во всех таблицах должно совпадать, иначе будет ошибка.

Синтаксис команды UNION с удалением дублей:

SELECT \* FROM имя\_таблицы1 WHERE условие UNION SELECT \* FROM имя\_таблицы2 WHERE условие.

Синтаксис команды UNION без удаления дублей:

SELECT \* FROM имя\_таблицы1 WHERE условие UNION ALL SELECT \* FROM имя\_таблицы2 WHERE условие.

5) Способы объединения таблиц. Краткая характеристика.

Некоторые способы объединения таблиц в SQL и их краткая характеристика:

1. Inner Join. Возвращает только те строки, которые имеют совпадения в обеих объединённых таблицах. Например, можно объединить таблицы сотрудников employees и отделов departments, чтобы создать набор результатов, показывающий название отдела для каждого сотрудника.

2. Outer Join. Это расширение Inner Join, которое возвращает строки, даже если они не имеют связанных строк в объединённой таблице. Существует три типа Outer Join: left outer join, right outer join и full outer join.

3. Cross Join. Это соединение без условия соединения. Каждая строка одной таблицы объединяется с каждой строкой другой таблицы. Такой тип набора результатов называется декартовым или перекрёстным произведением. Например, cross join между таблицами сотрудников employees и отделов departments вернёт набор результатов с одной строкой для каждой возможной комбинации «сотрудники/отделы».

6) План запроса. Краткое описание.

План выполнения запроса (или план запроса) – это детализированное описание последовательности операций, которые СУБД (система управления базами данных) выполняет для получения результата SQL-запроса. Он показывает, как данные извлекаются из таблиц, какие индексы используются, как выполняются соединения и сортировки, и другие аспекты выполнения запроса.

7) Оптимизация запроса. Краткое описание.

Оптимизация запросов – это 1) функция СУБД, осуществляющая поиск оптимального плана выполнения запросов из всех возможных для заданного запроса, 2) процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса.

8) Оконные функции. Краткое описание.

Оконные функции в SQL (от англ. window function) – это особый класс функций, позволяющий производить вычисления по определённым группам строк в базе данных. При этом они не объединяют строки в одну, а возвращают столько же, сколько было на входе.

Эти функции удобно использовать для отчётов, анализа данных, финансового моделирования и других задач, где нужно видеть результаты в контексте исходных данных.

9) Команда и синтаксис для группировки данных.

Команда для группировки данных в SQL – GROUP BY. Она позволяет группировать строки в таблице по одному или нескольким столбцам.

Синтаксис команды:

SELECT ИМЕНА\_СТОЛБЦОВ FROM ИМЯ\_ТАБЛИЦЫ [WHERE УСЛОВИЕ] GROUP BY ИМЕНА\_СТОЛБЦОВ

10) Сортировка данных. Краткое описание.

Сортировка данных в SQL позволяет расположить строки таблицы в указанном порядке по выбранному столбцу.

Для этого используется оператор ORDER BY. После него указывается поле, по которому производится сортировка.

По умолчанию команда ORDER BY выполняет сортировку по возрастанию. Чтобы управлять направлением сортировки вручную, после имени столбца указывается ключевое слово ASC (по возрастанию) или DESC (по убыванию).

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

WHERE используется для фильтрации записей из таблицы. После служебного слова WHERE указываются условия выбора строк, которые будут помещены в результирующую таблицу. Условия применяются к строкам исходной таблицы до группировки данных.

HAVING применяется после группировки данных. Основная функция HAVING – фильтрация групп на основе уже вычисленных агрегатных значений. Условия, определённые в HAVING, применяются к уже сформированным группам данных, что даёт возможность провести более детальный анализ.

Таким образом, WHERE исключает строки до того, как выполняется группировка, а HAVING — после неё. Поэтому в HAVING могут использоваться агрегатные функции, в отличие от WHERE, где агрегатные функции недопустимы.