МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

КУРСОВАЯ РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Старший преподаватель		В. В. Боженко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ АВТОСАЛОНА

по дисциплине: БАЗЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4217		У. А. Мазориев
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Отсутствие централизованной информационной системы приводит к потерям времени, неточностям в отчетах и усложнению контроля за текущей деятельностью. Поэтому разработка и внедрение информационной системы (ИС), позволяющей эффективно управлять всеми аспектами работы автосалона, является важной и актуальной задачей.

Целью данной курсовой работы является разработка полноценной информационной системы для автосалона средствами PostgreSQL. В рамках проекта предполагается создание базы данных, обеспечивающей хранение информации о клиентах, автомобилях, заказах, консультантах по продажам, а также построение взаимосвязей между всеми ключевыми объектами предметной области.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области, выявив ключевые сущности и процессы;
- определить основные сущности и связи между ними;
- разработать даталогическую модели базы данных;
- реализовать физическую модель в выбранной СУБД (PostgreSQL);
- наполнить базу тестовыми данными;
- реализовать примеры SQL-запросов к базе данных, демонстрирующие её функциональность;
- провести тестирование работоспособности базы данных;
- сделать выводы по результатам работы.

В качестве программного обеспечения выбрана реляционная система управления базами данных PostgreSQL, которая предоставляет широкие возможности для создания надежных, масштабируемых и производительных информационных систем. PostgreSQL поддерживает стандарты SQL, имеет развитую систему типов данных, триггеры, процедуры, представления и множество других функций, что делает её идеальным инструментом для решения задач подобного рода.

Таким образом, в рамках курсовой работы будет спроектирована и реализована база данных, способная обеспечить эффективную информационную поддержку бизнес-процессов автосалона, улучшить качество обслуживания клиентов и упростить управленческий контроль.

1 Анализ предметной области

1.1 Описание предметной области

Автомобильная торговля - это одна из наиболее динамично развивающихся отраслей, требующая постоянного контроля за товарными запасами, персоналом, клиентской базой и продажами. Автосалон - это не только место реализации транспортных средств, но и полноценное предприятие, включающее в себя взаимодействие с поставщиками, клиентами, ведение технической документации, бухгалтерский и складской учет.

Одной из ключевых задач успешной работы автосалона является эффективное управление информацией о клиентах, автомобилях, моделях и брендах, а также процессами продаж и консультаций. Для решения этой задачи необходима автоматизация, позволяющая ускорить обработку заказов, снизить количество ошибок и обеспечить надежное хранение данных.

1.2 Анализ сущностей и связей между ними

Были выделены основные сущности, сведения о которых будут храниться в базе данных. Сущность в базе данных - это любой объект в базе данных, который можно выделить исходя из сути предметной области, для которой разрабатывается эта база данных [1]. Список сущностей приведен в таблице 1.

Таблица 1 - Список сущностей

№	Название	Описание
1	clients	Клиенты автосалона
2	brands	Бренды автомобилей
3	models	Модели автомобилей
4	countries	Страны-производители
5	cars	Автомобили
6	technical_specs	Технические характеристики
7	sales_consultants	Консультанты по продажам
8	orders	Заказы клиентов

Также были выделены атрибуты базы данных. Значения атрибутов описывают элементы сущности [3]. Список атрибутов и их описание для каждой сущности приведены в таблицах 2–9.

Таблица 2 - Список атрибутов сущности «clients»

Название	Тип	NaN	Описание
id_client	SERIAL	Нет	Идентификационный номер клиента
surname	VARCHAR(50)	Нет	Фамилия клиента
name	VARCHAR(50)	Нет	Имя клиента
patronymic	VARCHAR(50)	Да	Отчество клиента
passport_series	CHAR(4)	Нет	Серия паспорта
passport_number	CHAR(6)	Нет	Номер паспорта
address	VARCHAR(255)	Нет	Адрес проживания
phone_number	VARCHAR(20)	Нет	Номер телефона
Date_birthday	DATE	Нет	Дата рождения

Таблица 3 - Список атрибутов сущности «brands»

Название	Тип	NaN	Описание
id_brand	SERIAL	Нет	Идентификационный номер бренда
name	VARCHAR(50)	Нет	Название бренда

Таблица 4 - Список атрибутов сущности «models»

Название	Тип	NaN	Описание
id_model	SERIAL	Нет	Идентификационный номер модели
id_brand	INT	Нет	Идентификатор бренда
name	VARCHAR(50)	Нет	Название модели

Таблица 5 - Список атрибутов сущности «countries»

Название	Тип	NaN	Описание
id_country	SERIAL	Нет	Идентификационный номер страны
name	VARCHAR(50)	Нет	Название страны

Таблица 6 - Список атрибутов сущности «cars»

Название	Тип	NaN	Описание
id_car	SERIAL	Нет	Идентификационный номер автомобиля

id_model	INT	Нет	Идентификатор модели
id_country	INT	Нет	Идентификатор страны
price	DECIMAL(10,2)	Нет	Цена автомобиля
quantity_in_stock	INT	Нет	Количество в наличии
stock_status	VARCHAR(20)	Нет	Статус наличия
arrival_date	DATE	Да	Дата поступления

Таблица 7 - Список атрибутов сущности «technical_specs»

Название	Тип	NaN	Описание
id_spec	SERIAL	Нет	Идентификатор характеристик
id_car	INT	Нет	Идентификатор автомобиля
body_type	VARCHAR(50)	Нет	Тип кузова
doors_count	INT	Нет	Количество дверей
seats_count	INT	Нет	Количество сидений
engine_type	VARCHAR(50)	Нет	Тип двигателя
engine_position	VARCHAR(50)	Нет	Положение двигателя
engine_capacity	DECIMAL(4,2)	Нет	Объём двигателя
engine_power	INT	Нет	Мощность двигателя
Options	TEXT	Да	Дополнительные опции
drive_wheels	VARCHAR(20)	Нет	Привод
fuel_consumption	DECIMAL(4,2)	Нет	Расход топлива
Acceleration	DECIMAL(4,2)	Нет	Разгон до 100 км/ч

Таблица 8 - Список атрибутов сущности «sales_consultants»

Название	Тип	NaN	Описание
id_consultant	SERIAL	Нет	Идентификатор консультанта
surname	VARCHAR(50)	Нет	Фамилия
name	VARCHAR(50)	Нет	Имя
patronymic	VARCHAR(50)	Да	Отчество
passport_series	CHAR(4)	Нет	Серия паспорта
passport_number	CHAR(6)	Нет	Номер паспорта
phone_number	VARCHAR(20)	Нет	Номер телефона
salary	DECIMAL(10,2)	Нет	Зарплата

Таблица 9 - Список атрибутов сущности «orders»

Название	Тип	NaN	Описание
id_order	SERIAL	Нет	Идентификатор заказа
id_client	INT	Нет	Идентификатор клиента
id_car	INT	Нет	Идентификатор автомобиля
id_consultant	INT	Да	Идентификатор консультанта
order_date	DATE	Нет	Дата заказа
delivery	VARCHAR(10)	Нет	Доставка
payment_type	VARCHAR(20)	Нет	Способ оплаты

Связи между сущностями делятся на три типа по множественности [2]:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

Связь «один-к-одному» означает, что экземпляр одной сущности связан только с одним экземпляром другой сущности.

Связь «один-ко-многим» означает, что один экземпляр сущности, расположенный слева по связи, может быть связан с несколькими экземплярами сущности, расположенными справа по связи.

Связь «многие-ко-многим» означает, что один экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и наоборот, один экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности.
Тип связи между сущностями в таблицы один-ко-многим:

- Один бренд может соответствовать нескольким моделям (brands - models)
- Одна модель может соответствовать нескольким автомобилям (models - cars)
- Одна страна может быть связана с несколькими автомобилями (countries -cars)
- Один клиент может совершить несколько заказов (clients - orders)
- Один консультант может быть связан с несколькими заказами
- (sales_consultants - orders)
- Один автомобиль может быть продан в нескольких заказах (cars - orders)
- Один автомобиль связан с одной записью технических характеристик (cars -
- technical_specs), связь реализована как один-к-одному через внешний ключ с

- уникальностью на id_car (при необходимости)

1.3 Нормализация отношений

Нормализация отношений (таблиц) — одна из основополагающих частей теории реляционных баз данных. Нормализация имеет своей целью избавиться от избыточности в отношениях и модифицировать их структуру таким образом, чтобы процесс работы с ними не был обременён различными посторонними сложностями. При игнорировании такого подхода эффективность проектирования стремительно снижается [3].

Существует несколько нормальных форм (НФ), каждая из которых предъявляет определённые требования к структуре отношений в базе данных. Наиболее часто используется нормализация до третьей нормальной формы (3НФ), как оптимальный баланс между сложностью и эффективностью работы с данными.

Требования к 1НФ

- все атрибуты атомарны (т.е. неделимы),
- в таблице отсутствуют повторяющиеся группы.

Все таблицы в разработанной базе данных удовлетворяют 1НФ. Каждое поле содержит одно значение, и все записи в таблицах уникальны.

Требования к 2НФ

- оно находится в 1НФ;
- каждый неключевой атрибут полностью функционально зависит от первичного ключа.

Во всех таблицах используется одинарный первичный ключ (например, id_client, id_model), поэтому отсутствуют частичные зависимости. Все неключевые поля зависят от первичного ключа таблицы.

Требования к 3НФ

- оно находится во 2НФ;
- в нём отсутствуют транзитивные зависимости между неключевыми атрибутами.

В таблице cars атрибут stock_status зависит от quantity_in_stock, но благодаря использованию ограничения CHECK в определении таблицы, логика связи этих полей сохранена без нарушения требований 3НФ.

Также все остальные неключевые атрибуты функционально зависят только от первичного ключа, и между ними нет транзитивных зависимостей.

1.4 Создание схемы данных

Была создана ER-модель данной базы данных. Модель сущностных отношений (ER Modeling) – это графический подход к проектированию базы данных. Это модель данных высокого уровня, которая определяет элементы данных и их взаимосвязь для определенной программной системы [5].

ER-модель созданной базы данных представлена на рисунке 1.

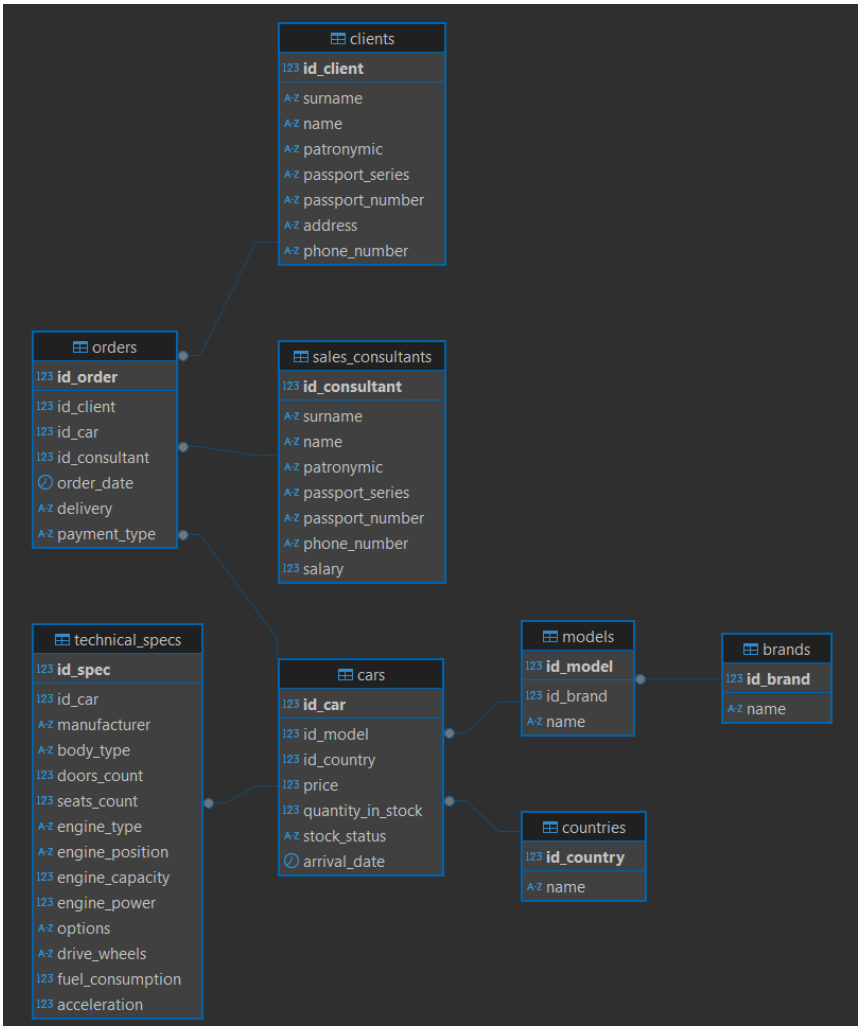


Рисунок 1 – ER-диаграмма базы данных

После создания ER-диаграммы все таблицы заполняются искусственно сгенерированными данными. Для этого пишется скрипт на Python с использованием библиотеки Faker, который генерирует запросы на вставку значений в таблицы, сам скрипт представлен в приложении А, а скриншоты заполненных таблиц в приложении Б на рисунках 54 - 61.

2 Разработка базы данных

2.1 Составление запросов к базе данных

После создания и заполнения базы данных были составлены запросы, используемые для извлечения необходимых данных из таблицы.

Запрос №1: Выдать информацию о наличии автомобилей определенной марки и модели (марка и модель – входные параметры);

Для выполнения данного запроса объединяются таблицы models, cars, brands, technical_specs для этого используется inner_join (внутреннее соединение) после выборка сортируются по первым двум полям для читаемости. В качестве фильтра выступают поля марки и Модели. У одной марки и модели может быть несколько комплектаций, поэтому было принято решение выводить также столбцы с ценой и опциями определенной комплектации. Код запросы представлен в листинге 1. Результат выполнения запросы представлен на рисунке 2. В результате выводятся информация о наличии машин определенной марки и модели в разных комплектациях.

Листинг 1 – Запрос №1

```
select
          b.name as "Марка"
          ,m.name as "Модель"
          ,c.stock_status as "Наличие"
          ,c.price as "Цена"
          ,ts.options as "Опции"
from models m
inner join cars c on c.id_model = m.id_model
inner join brands b on b.id_brand = m.id_brand
inner join technical_specs ts on ts.id_car = c.id_car
where b.name = 'Audi' and m.name = 'A6'
order by 1 asc, 2 asc
```

	Марка	Модель	Наличие	Цена	Опции
1	Audi	A6	да	2 699 206,01	Круиз-контроль, Камера заднего вида
2	Audi	A6	да	3 124 176,29	Система безопасности вождения, Мультимедийная система с поддержкой Apple Ca
3	Audi	A6	да	3 124 176,29	Навигация, Электрические сиденья, Передние и задние датчики парковки, Подогре
4	Audi	A6	да	3 124 176,29	Круиз-контроль, Передние и задние датчики парковки, Система стабилизации, Кож
5	Audi	A6	да	3 124 176,29	Навигация, Bluetooth, Автономное вождение, Климат-контроль
6	Audi	A6	да	3 124 176,29	Автономное вождение, Камера заднего вида, Проекционный экран, Система стаби

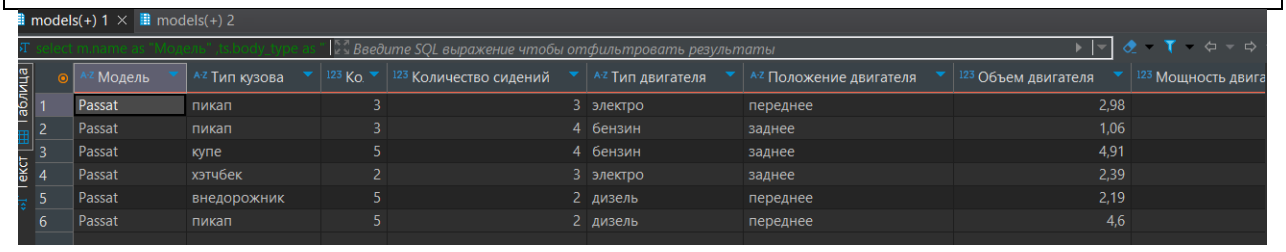
Рисунок 2 – Результат выполнения запроса №1

Запрос №2: Выдать технические данные заданной модели (модель – входной параметр);

Для выполнения данного запроса выбираются нужные поля из таблиц «models», «technical_specs» для этого используется внутренние соединение, которое соединяется таблицы «models», «technical_specs» через таблицу «cars» далее данные фильтруются по определенной модели, сортировка по возрастанию. Код запроса представлен в листинге 2. Результат выполнения запроса представлен на рисунке 3. В результате выводятся технические характеристики для модели в разных комплектациях.

Листинг 2 – Запрос №2

```
select
          m.name as "Модель"
          ,ts.body_type as "Тип кузова"
          ,ts.doors_count as "Количество дверей"
          ,ts.seats_count "Количество сидений"
          ,ts.engine_type as "Тип двигателя"
          ,ts.engine_position as "Положение двигателя"
          ,ts.engine_capacity as "Объем двигателя"
          ,ts.engine_power as "Мощность двигателя"
          ,ts."options" as "Дополнительные опции"
          ,ts.drive_wheels as "Привод"
          ,ts.fuel_consumption as "Расход топлива"
          ,ts.acceleration as "Разгон до 100км/ч"
from models m
inner join cars c on c.id_model = m.id_model
inner join technical_specs ts on ts.id_car = c.id_car
where m.name = 'Passat'
```



	Модель	Тип кузова	Ко	Количество сидений	Тип двигателя	Положение двигателя	Объем двигателя	Мощность двига
1	Passat	пикап	3	3	электро	переднее	2,98	
2	Passat	пикап	3	4	бензин	заднее	1,06	
3	Passat	купе	5	4	бензин	заднее	4,91	
4	Passat	хэтчбек	2	3	электро	заднее	2,39	
5	Passat	внедорожник	5	2	дизель	переднее	2,19	
6	Passat	пикап	5	2	дизель	переднее	4,6	

Рисунок 3 - Результат выполнения запроса №2

Запрос №3: Количество проданных моделей определенной марки с подсчетом общей выручки по каждому модельному ряду (марка – входной параметр).

Для выполнения данного запроса выбираются нужные поля из таблиц «orders», «cars», «models», «brands», которые соединяются через внутреннее соединение. Таблицы «cars», «models» и «brands» связываются через соответствующие идентификаторы. Затем данные фильтруются по определенной марке, в данном случае «Kia». Для подсчета общей выручки используется функция sum(), а количество проданных автомобилей подсчитывается с помощью функции count(). Данные группируются по марке автомобиля. Код запроса представлен в листинге 3. Результат выполнения запроса представлен на рисунке 4. В результате выводится количество проданных автомобилей и общая выручка для марки «Kia».

Листинг 3 – Запрос №3

```
SELECT
    b.name AS "Марка",
    COUNT(o.id_order) AS "Количество проданных автомобилей",
    SUM(c.price) AS "Общая выручка"
FROM
    orders o
INNER JOIN cars c ON c.id_car = o.id_car
INNER JOIN models m ON c.id_model = m.id_model
INNER JOIN brands b ON b.id_brand = m.id_brand
WHERE b.name = 'Kia'
GROUP BY b.name;
```

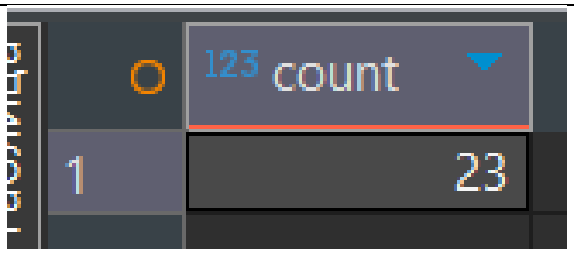


Рисунок 4 - Результат выполнения запроса №3

Запрос №4: Средняя сумма продаж авто за определенный промежуток времени.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders» и «cars», которые соединяются через внутреннее соединение. Затем данные фильтруются по диапазону дат с помощью оператора BETWEEN. Для вычисления средней суммы продаж

используется функция AVG(). Результат округляется до двух знаков после запятой с помощью функции ROUND(). Код запроса представлен в листинге 4. Результат выполнения запроса представлен на рисунке 5. В результате выводится средняя цена автомобилей, проданных в заданный промежуток времени.

Листинг 4 – Запрос №4

```
SELECT
     ROUND(AVG(c.price), 2) AS "Средняя сумма продаж"
FROM
     orders o
INNER JOIN cars c ON c.id_car = o.id_car
WHERE o.order_date BETWEEN '2024-01-01' AND '2024-12-31';
```

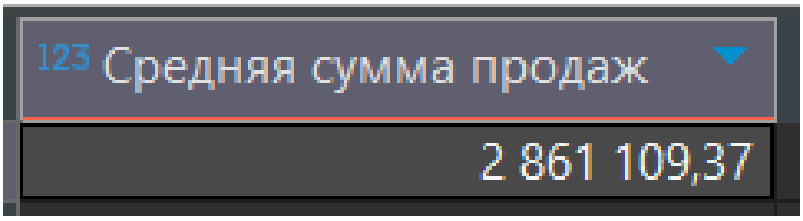


Рисунок 5 – Результат выполнения запроса №4

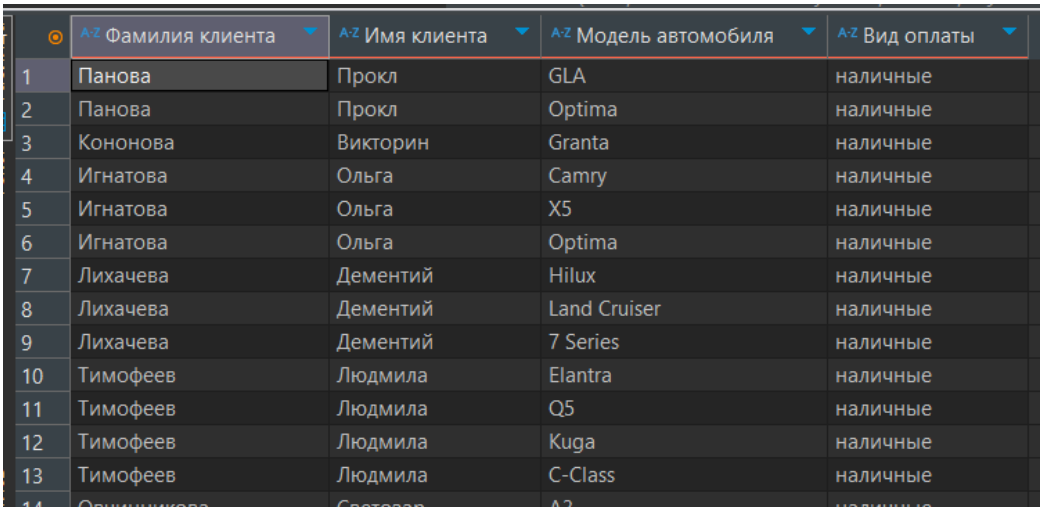
Запрос №5: Выдать списки клиентов и автомобилей по виду оплаты.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders», «clients», «cars» и «models», которые соединяются через внутреннее соединение. Данные фильтруются по виду оплаты, например, 'наличные'. В результате выводятся фамилия и имя клиента, модель автомобиля, а также вид оплаты. Код запроса представлен в листинге 5. Результат выполнения запроса представлен на рисунке 6. В результате выводится список клиентов и их автомобилей с указанным видом оплаты.

Листинг 5 – Запрос №5

```
SELECT
    cl.surname AS "Фамилия клиента",
    cl.name AS "Имя клиента",
    m.name AS "Модель автомобиля",
    o.payment_type AS "Вид оплаты"
FROM
```

```
    orders o
INNER JOIN clients cl ON cl.id_client = o.id_client
INNER JOIN cars c ON c.id_car = o.id_car
INNER JOIN models m ON m.id_model = c.id_model
WHERE o.payment_type = 'наличные';
```



	Фамилия клиента	Имя клиента	Модель автомобиля	Вид оплаты
1	Панова	Прокл	GLA	наличные
2	Панова	Прокл	Optima	наличные
3	Кононова	Викторин	Granta	наличные
4	Игнатова	Ольга	Camry	наличные
5	Игнатова	Ольга	X5	наличные
6	Игнатова	Ольга	Optima	наличные
7	Лихачева	Дементий	Hilux	наличные
8	Лихачева	Дементий	Land Cruiser	наличные
9	Лихачева	Дементий	7 Series	наличные
10	Тимофеев	Людмила	Elantra	наличные
11	Тимофеев	Людмила	Q5	наличные
12	Тимофеев	Людмила	Kuga	наличные
13	Тимофеев	Людмила	C-Class	наличные

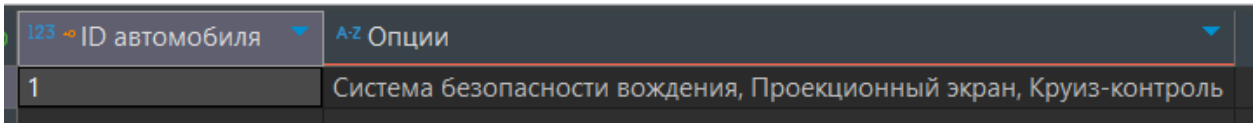
Рисунок 6 – Результат выполнения запроса №5

Запрос №6: Информация об опциях данного автомобиля.

Для выполнения данного запроса выбираются нужные поля из таблиц «cars» и «technical_specs», которые соединяются через внутреннее соединение. Данные фильтруются по идентификатору автомобиля. В результате выводится список опций для выбранного автомобиля. Код запроса представлен в листинге 6. Результат выполнения запроса представлен на рисунке 7. В результате выводятся опции для конкретного автомобиля.

Листинг 6 – Запрос №6

```
SELECT
    c.id_car AS "ID автомобиля",
    ts.options AS "Опции"
FROM
    cars c
INNER JOIN technical_specs ts ON ts.id_car = c.id_car
WHERE c.id_car = 1;
```



ID автомобиля	Опции
1	Система безопасности вождения, Проекционный экран, Круиз-контроль

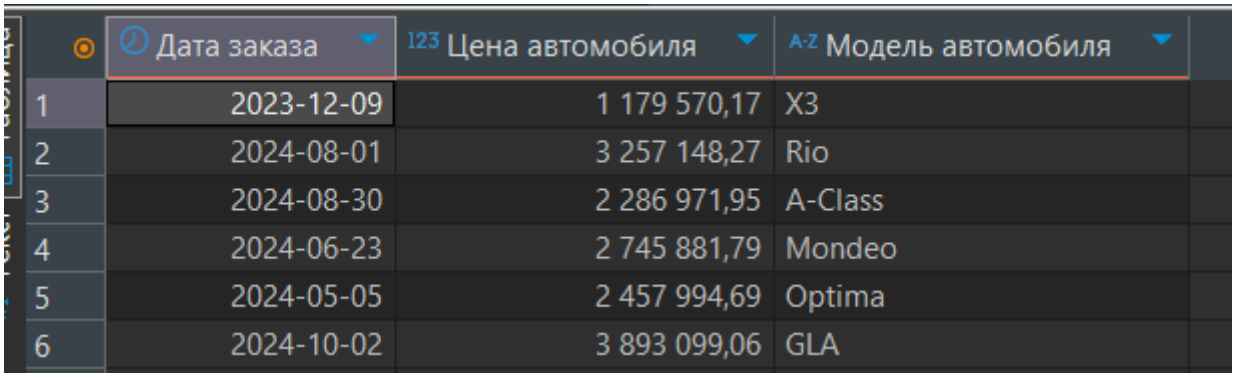
Рисунок 7 – Результат выполнения запроса №6

Запрос №7: Информация обо всех покупках данного клиента.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders», «cars» и «models», которые соединяются через внутреннее соединение. Данные фильтруются по идентификатору клиента. В результате выводятся дата заказа, цена автомобиля и модель для всех покупок данного клиента. Код запроса представлен в листинге 7. Результат выполнения запроса представлен на рисунке 8. В результате выводится информация обо всех покупках клиента.

Листинг 7 – Запрос №7

```
SELECT
    o.order_date AS "Дата заказа",
    c.price AS "Цена автомобиля",
    m.name AS "Модель автомобиля"
FROM
    orders o
INNER JOIN cars c ON c.id_car = o.id_car
INNER JOIN models m ON m.id_model = c.id_model
WHERE o.id_client = 1;
```



	Дата заказа	Цена автомобиля	Модель автомобиля
1	2023-12-09	1 179 570,17	X3
2	2024-08-01	3 257 148,27	Rio
3	2024-08-30	2 286 971,95	A-Class
4	2024-06-23	2 745 881,79	Mondeo
5	2024-05-05	2 457 994,69	Optima
6	2024-10-02	3 893 099,06	GLA

Рисунок 8– Результат выполнения запроса №7

Запрос №8: Найти всех клиентов, которые заказали автомобили с мощностью двигателя более «x» л.с. и расходом топлива меньше «y» литров на 100 км.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders», «clients», «cars», «models» и «technical_specs», которые соединяются через внутреннее соединение. Данные фильтруются по мощности двигателя и расходу топлива, где параметры мощности и расхода задаются как входные значения. В результате выводятся фамилия и имя клиента, модель автомобиля и дата заказа. Код запроса представлен в листинге 8. Результат выполнения запроса представлен на рисунке 9. В результате выводятся клиенты, которые заказали автомобили с заданными характеристиками.

Листинг 8 – Запрос №8

```
SELECT
    cl.surname AS "Фамилия клиента",
    cl.name AS "Имя клиента",
    m.name AS "Модель автомобиля",
    o.order_date AS "Дата заказа"
FROM
    orders o
INNER JOIN clients cl ON cl.id_client = o.id_client
INNER JOIN cars c ON c.id_car = o.id_car
INNER JOIN models m ON m.id_model = c.id_model
INNER JOIN technical_specs ts ON ts.id_car = c.id_car
WHERE ts.engine_power > 150
AND ts.fuel_consumption < 8.0
```

	Фамилия клиента	Имя клиента	Модель автомобиля	Дата заказа
1	Панова	Прокл	Mondeo	2024-06-23
2	Кононова	Викторин	Kuga	2024-11-08
3	Осипова	Аркадий	Jetta	2025-02-22
4	Осипова	Аркадий	Jetta	2025-02-22
5	Тимофеев	Людмила	Kuga	2023-11-25
6	Тимофеев	Людмила	Corolla	2024-09-23
7	Кириллов	Изяслав	X-Trail	2025-03-20
8	Овчинникова	Светозар	A3	2023-09-20
9	Овчинникова	Светозар	Santa Fe	2023-12-09
10	Юдина	Добромысл	Optima	2024-05-23

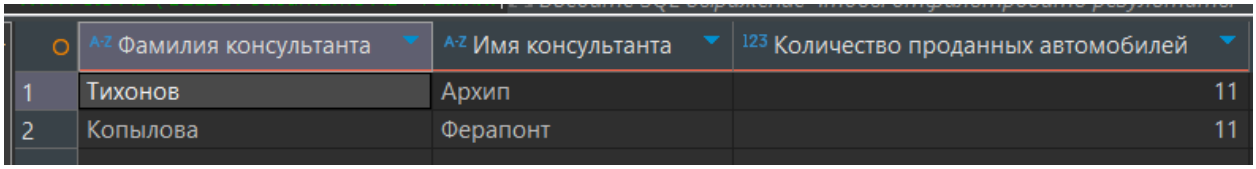
Рисунок 9 – Результат выполнения запроса №8

Запрос №9: Найти продавцов-консультантов, которые продали наибольшее количество автомобилей в разрезе двух дат.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders» и «sales_consultants», которые соединяются через внутреннее соединение. С помощью временной таблицы (WITH) выполняется подсчет количества проданных автомобилей каждым консультантом, затем данные ранжируются по количеству продаж. В результате выводятся продавцы-консультанты, которые продали наибольшее количество автомобилей в заданный промежуток времени. Код запроса представлен в листинге 9. Результат выполнения запроса представлен на рисунке 10. В результате выводится информация о консультанте с наибольшими продажами.

Листинг 9 – Запрос №9

```
WITH cte AS
(
    SELECT
        sc.surname AS "Фамилия консультанта",
        sc.name AS "Имя консультанта",
        COUNT(o.id_order) AS "Количество проданных автомобилей",
        RANK() OVER (ORDER BY COUNT(o.id_order) DESC) AS "rank"
    FROM orders o
    INNER JOIN sales_consultants sc ON sc.id_consultant =
o.id_consultant
    WHERE o.order_date BETWEEN '2024-01-01' AND '2024-12-31'
    GROUP BY sc.id_consultant
)
SELECT
    cte.*
FROM
    cte
WHERE cte."rank" = 1;
```



	Фамилия консультанта	Имя консультанта	Количество проданных автомобилей
1	Тихонов	Архип	11
2	Копылова	Ферапонт	11

Рисунок 10 – Результат выполнения запроса №9

Запрос №10: Выдать все заказы, которые были осуществлены в заданный отчетный период, в которых участвуют автомобили определенной марки.

Для выполнения данного запроса выбираются нужные поля из таблиц «orders», «cars», «models» и «brands», которые соединяются через внутреннее соединение. Данные фильтруются по заданному периоду времени и марке автомобиля. В результате выводятся все заказы, которые были сделаны на автомобили выбранной марки в указанный временной промежуток. Код запроса представлен в листинге 10. Результат выполнения запроса представлен на рисунке 11. В результате выводятся заказы на автомобили конкретной марки в заданный период.

Листинг 10 – Запрос №10

```
SELECT
    o.id_order AS "ID заказа",
    o.order_date AS "Дата заказа",
    b.name AS "Марка автомобиля"
FROM
    orders o
INNER JOIN cars c ON c.id_car = o.id_car
INNER JOIN models m ON c.id_model = m.id_model
INNER JOIN brands b ON b.id_brand = m.id_brand
WHERE o.order_date BETWEEN '2024-01-01' AND '2024-12-31'
AND b.name = 'Audi';
```

id_order	Дата заказа	Марка машины	Модель автомобиля
22	2024-11-19	BMW	Land Cruiser
53	2024-12-24	BMW	Optima
120	2024-07-07	BMW	Optima
126	2024-05-09	BMW	Optima
133	2024-08-30	BMW	A-Class
142	2024-01-23	BMW	Land Cruiser

Рисунок 11 – Результат выполнения запроса №10

2.2 Анализ разрешений и запретов на операции с табличными данными для различных пользователей

Для более безопасной работы БД создаются пользователи. Привилегии

пользователям должны быть предоставлены строго только те, что действительно необходимы. Пользователи и описание их прав приведены в таблице 10. Код для создания пользователей представлен в листинге 11. Результат выполнения кода для создания пользователей представлен на рисунке 12.

Таблица 10 - Разрешения и запреты на операции с табличными данными для различных пользователей

Роль	Описание прав	Разрешенные операции	Запрещенные операции
Продавец-консультант	Работает с клиентами, записывает информацию о клиенте, оформляет покупку.	SELECT, INSERT, UPDATE в таблицах клиентов и заказов	DELETE
Экономист	Анализирует финансовую деятельность предприятия	SELECT в таблицах заказов, автомобилей	INSERT, UPDATE, DELETE
Директор	Осуществляет общий контроль деятельности предприятия	SELECT, UPDATE во всех таблицах, доступ к аналитическим данным	INSERT, `DELETE
Администратор БД	Имеет полное право доступа к БД, отвечает за её работу.	Все	Нет ограничений

Листинг 11 – Создание пользователей

```
CREATE USER seller WITH PASSWORD 'seller';
GRANT SELECT, INSERT, UPDATE ON clients TO seller;
GRANT SELECT, INSERT, UPDATE ON orders TO seller;
REVOKE DELETE ON clients, orders FROM seller;
```

```
CREATE USER economist WITH PASSWORD 'economist'
GRANT SELECT ON cars, orders, cars TO economist;
REVOKE INSERT, UPDATE, DELETE ON cars, orders, cars FROM economist;
CREATE USER director WITH PASSWORD 'director';

GRANT SELECT, UPDATE ON ALL TABLES IN SCHEMA auto TO director;
REVOKE INSERT, DELETE ON ALL TABLES IN SCHEMA auto FROM director;

CREATE USER admin WITH PASSWORD 'admin';
GRANT ALL PRIVILEGES ON DATABASE students TO admin;
```

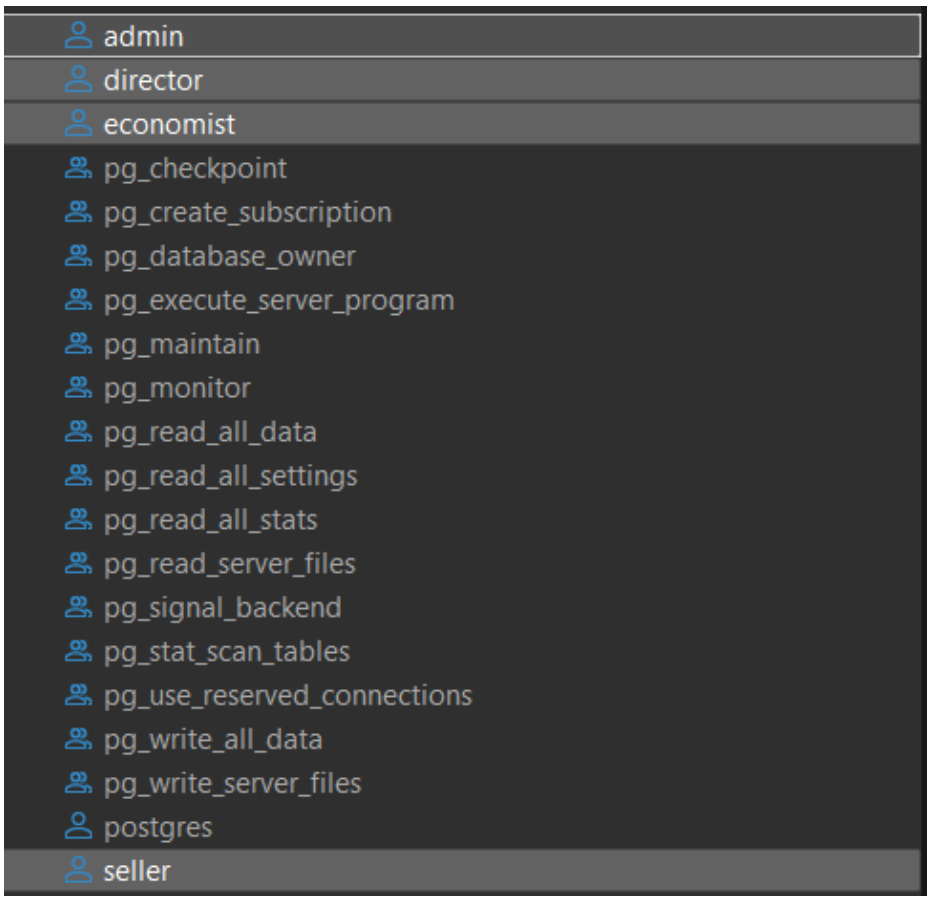


Рисунок 12 – Созданные роли

3 Проектирование пользовательского интерфейса

3.1 Создание хранимых процедур

Все запросы были реализованы через функции.

Функции в PostgreSQL - блоки кода, которые выполняют определенные действия и могут возвращать результаты, такие как данные, числовые значения или же просто изменять состояние базы данных (например, выполнять обновления).

Основные отличия от написания запроса заключается в том, что в функции нужно объявить входные и выходные параметры, указать язык написаний

Функция №1: Выдать информацию о наличии автомобилей определенной марки и модели (марка и модель – входные параметры);

Для выполнения данного запроса через функцию создается функция get_car_availability, которая принимает на вход текстовые не пустые значения марки и модели автомобиля, если значения NULL или пустая строка, то функция вызывает исключение. Функция не чувствительна к регистру написания. В качестве выходного параметра выступает таблица, в которой 5 полей. Если машина не найдена, то функции выводит сообщение об этом. Код функции представлен в листинге 12. Результат выполнения запроса представлен на рисунках 13 - 15.

Листинг 12 – Функция get_car_availability()

```
CREATE OR REPLACE FUNCTION get_car_availability(p_brand TEXT,
p_model TEXT)
RETURNS TABLE (
    Марка TEXT,
    Модель TEXT,
    Наличие TEXT,
    Цена NUMERIC,
    Опции TEXT
) AS $$
BEGIN
    IF p_brand IS NULL OR p_model IS NULL OR p_brand = '' OR
p_model = '' THEN
        RAISE EXCEPTION 'Марка и модель не могут быть пустыми';
    END IF;
```

```
    FOR Марка, Модель, Наличие, Цена, Опции IN
        SELECT b.name, m.name, c.stock_status, c.price,
ts.options
        FROM models m
        INNER JOIN cars c ON c.id_model = m.id_model
        INNER JOIN brands b ON b.id_brand = m.id_brand
        INNER JOIN technical_specs ts ON ts.id_car = c.id_car
        WHERE lower(b.name) = lower(p_brand) AND lower(m.name) =
lower(p_model)
        ORDER BY 1, 2
    LOOP
        RETURN NEXT;
    END LOOP;

    IF NOT FOUND THEN
        RETURN QUERY
            SELECT 'Автомобиля с такой маркой и моделью нет в
наличии'::TEXT,
                   NULL::TEXT, NULL::TEXT, NULL::NUMERIC,
NULL::TEXT;
    END IF;

    RETURN;
END;
$$ LANGUAGE plpgsql;
```

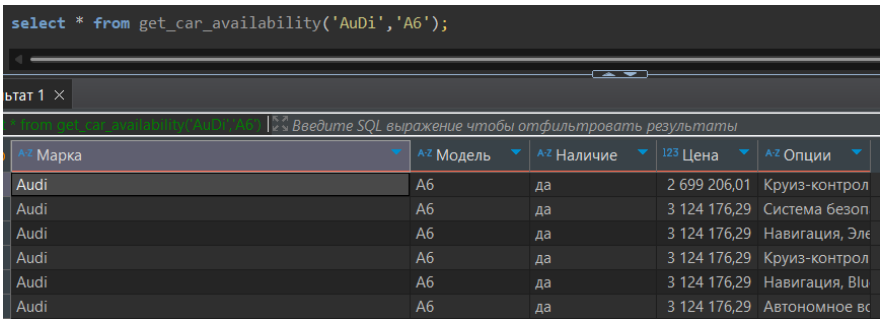


Рисунок 13 – Результат работы функции нечувствительной к регистру

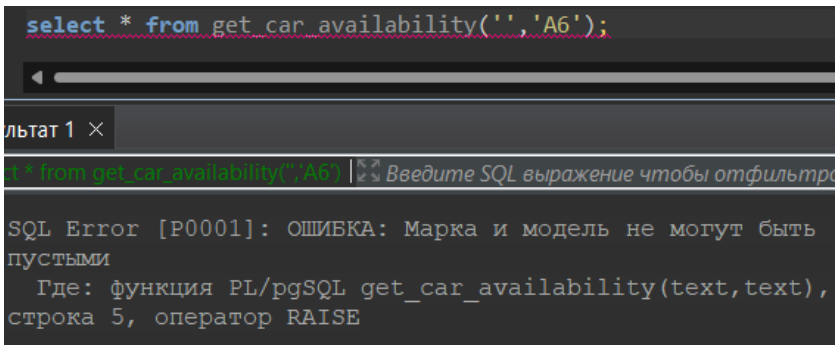


Рисунок 14 – Проверка на пустые строки

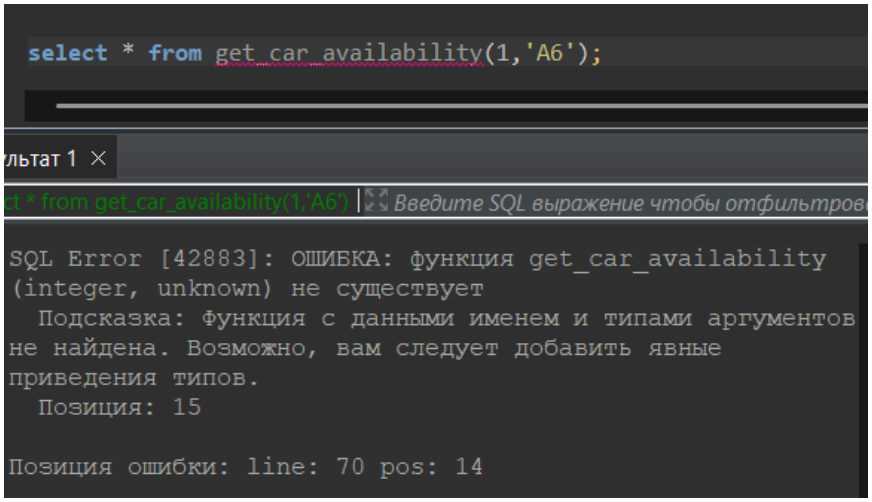


Рисунок 15 – Ошибка при числовых значениях

В дальнейшем исключения было принято обрабатывать в Python.

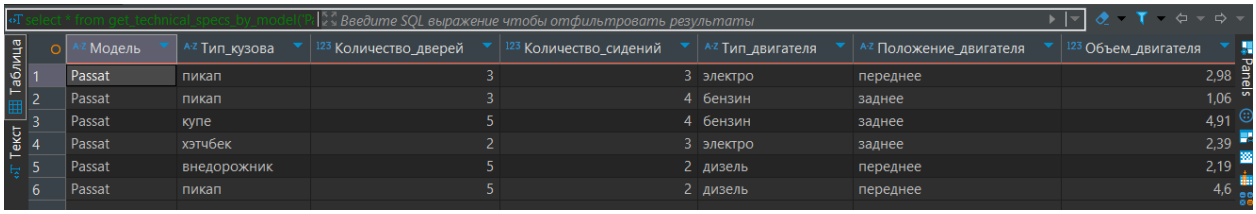
Функция №2: Выдать технические данные заданной модели (модель – входной параметр);

Для выполнения данного запроса была создана функция get_technical_specs_by_model, которая принимает на вход один параметр типа TEXT — название модели автомобиля. Функция возвращает таблицу с 12 полями, содержащими информацию о технических характеристиках данной модели, включая тип кузова, количество дверей и сидений, параметры двигателя, тип привода, расход топлива и ускорение до 100 км/ч.

Функция реализована на языке PL/pgSQL и использует внутренние соединения между таблицами models, cars и technical_specs, чтобы получить соответствующие данные. Поиск выполняется строго по введенному названию модели (без приведения к нижнему регистру), то есть чувствителен к регистру символов. Код функции представлен в листинге 13. Результат выполнения запроса представлен на рисунке 16.

Листинг 13 – Функция get_technical_specs_by_model

```
CREATE OR REPLACE FUNCTION get_technical_specs_by_model(p_model
TEXT)
RETURNS TABLE (
    Модель TEXT,
    Тип_кузова TEXT,
    Количество_дверей INT,
    Количество_сидений INT,
    Тип_двигателя TEXT,
    Положение_двигателя TEXT,
    Объем_двигателя NUMERIC,
    Мощность_двигателя INT,
    Дополнительные_опции TEXT,
    Привод TEXT,
    Расход_топлива NUMERIC,
    Разгон_до_100 NUMERIC
) AS $$
SELECT
    m.name, ts.body_type, ts.doors_count, ts.seats_count,
    ts.engine_type, ts.engine_position, ts.engine_capacity,
    ts.engine_power, ts.options, ts.drive_wheels,
    ts.fuel_consumption, ts.acceleration
FROM models m
JOIN cars c ON c.id_model = m.id_model
JOIN technical_specs ts ON ts.id_car = c.id_car
WHERE m.name = p_model;
$$ LANGUAGE SQL;
```



	Модель	Тип_кузова	Количество_дверей	Количество_сидений	Тип_двигателя	Положение_двигателя	Объем_двигателя
1	Passat	пикап	3	3	электро	переднее	2,98
2	Passat	пикап	3	4	бензин	заднее	1,06
3	Passat	купе	5	4	бензин	заднее	4,91
4	Passat	хэтчбек	2	3	электро	заднее	2,39
5	Passat	внедорожник	5	2	дизель	переднее	2,19
6	Passat	пикап	5	2	дизель	переднее	4,6

Рисунок 16 – Вызов функция get_technical_specs_by_model

Функция №3: Количество проданных моделей определённой марки с подсчётом общей выручки по каждому модельному ряду (марка – входной параметр);

Для выполнения данного запроса через функцию создается функция get_sold_model_count_and_revenue, которая принимает на вход название марки автомобиля (параметр brand_name). В случае, если передано пустое значение или NULL, функция вызывает исключение. Функция нечувствительна к регистру и выполняет подсчет количества проданных автомобилей каждой модели этой марки и их общей выручки. В качестве выходного параметра возвращается таблица, состоящая из трёх полей: модели, количества проданных автомобилей и общей выручки по каждому модельному ряду. Если автомобилей данной марки нет в базе данных, то функция возвращает пустую таблицу со значением 0. Код функции представлен в листинге 14. Результат выполнения запроса представлен на рисунках 17 - 18.

Листинг 14 – Функция get_sold_model_count()

```
CREATE OR REPLACE FUNCTION get_sold_model_count(brand_name TEXT)
RETURNS INT AS $$
DECLARE
    total INT;
BEGIN
    SELECT COUNT(b.name) INTO total
    FROM orders o
    INNER JOIN cars c ON c.id_car = o.id_car
    INNER JOIN models m ON c.id_model = m.id_model
    INNER JOIN brands b ON b.id_brand = m.id_brand
    WHERE b.name = brand_name;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

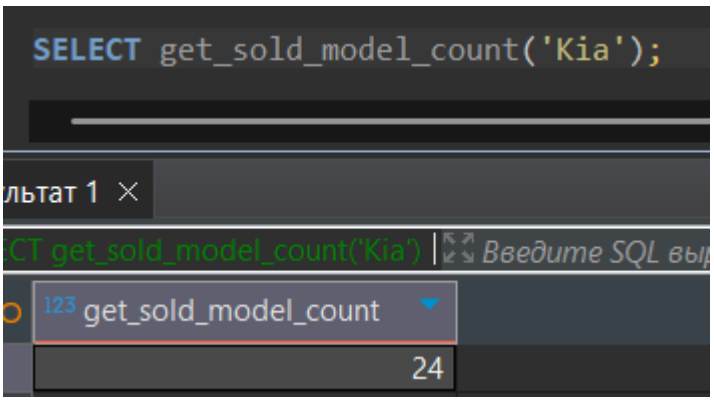


Рисунок 17 – Функция get_sold_model_count()

Функция №4: Средняя сумма продаж авто за определённый промежуток времени (даты – входной параметр);

Для выполнения данного запроса через функцию создается функция get_average_sale_amount, которая принимает два входных параметра: start_date и end_date, определяющие диапазон дат, в котором нужно подсчитать среднюю сумму продаж автомобилей. В случае, если хотя бы один из параметров является NULL, функция вызывает исключение. В качестве выходного параметра функция возвращает среднюю сумму продаж автомобилей за указанный промежуток времени. Если в указанный период не было продаж, функция возвращает значение NULL. Код функции представлен в листинге 15. Результат выполнения запроса представлен на рисунке 18.

Листинг 15 – Функция get_average_sale_amount()

```
CREATE OR REPLACE FUNCTION get_average_sale_amount(start_date
DATE, end_date DATE)
RETURNS NUMERIC AS $$
DECLARE
    avg_amount NUMERIC;
BEGIN
    SELECT ROUND(AVG(c.price), 2) INTO avg_amount
    FROM orders o
    INNER JOIN cars c ON c.id_car = o.id_car
    WHERE o.order_date BETWEEN start_date AND end_date;
    RETURN avg_amount;
END;
$$ LANGUAGE plpgsql;
```

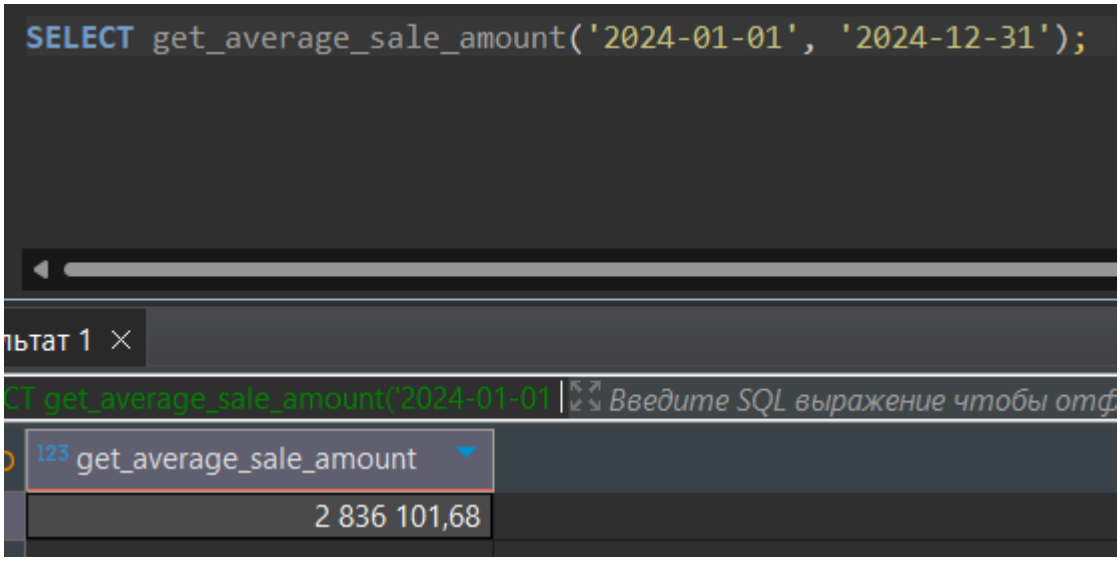


Рисунок 18 – Функция get_average_sale_amount()

Функция №5. Выдать списки клиентов и автомобилей по виду оплаты (вид оплаты – входной параметр);

Для выполнения данного запроса через функцию создается функция get_clients_by_payment, которая принимает на вход параметр payment, определяющий вид оплаты (например, "Cash" или "Credit Card"). Функция возвращает таблицу, в которой содержатся фамилия и имя клиента, модель автомобиля и вид оплаты, который использовался для покупки. Если в базе данных нет клиентов, использующих указанный вид оплаты, функция возвращает пустую таблицу. Код функции представлен в листинге 16. Результат выполнения запроса представлен на рисунке 19.

Листинг 16 – Функция get_clients_by_payment()

```
CREATE OR REPLACE FUNCTION get_clients_by_payment(payment TEXT)
RETURNS TABLE (
    Фамилия VARCHAR(50),
    Имя VARCHAR(50),
    Модель VARCHAR(50),
    Вид_оплаты VARCHAR(50)
) AS $$
BEGIN
    RETURN QUERY
    SELECT cl.surname, cl.name, m.name, o.payment_type
```

```
    FROM orders o
    INNER JOIN clients cl ON cl.id_client = o.id_client
    INNER JOIN cars c ON c.id_car = o.id_car
    INNER JOIN models m ON m.id_model = c.id_model
    WHERE o.payment_type = payment;
END;
$$ LANGUAGE plpgsql;
```

	Фамилия	Имя	Модель	Вид_оплаты
1	Кононова	Викторин	Kuga	кредит
2	Игнатова	Ольга	Passat	кредит
3	Игнатова	Ольга	Corolla	кредит
4	Осипова	Аркадий	A4	кредит
5	Тимофеев	Людмила	Corolla	кредит
6	Тимофеев	Людмила	X5	кредит
7	Тимофеев	Людмила	Granta	кредит
8	Кириллов	Изяслав	X-Trail	кредит
9	Овчинникова	Светозар	E-Class	кредит
10	Овчинникова	Светозар	Hilux	кредит

Рисунок 19 - Функция get_clients_by_payment()

Функция №6: Информация об опциях данного автомобиля (автомобиль – входной параметр);

Для выполнения данного запроса через функцию создается функция get_car_options, которая принимает на вход параметр car_id — уникальный идентификатор автомобиля. Функция возвращает информацию об опциях, доступных для данного автомобиля, в виде таблицы с двумя полями: ID_автомобиля (идентификатор автомобиля) и Опции (перечень доступных опций). Если автомобиль не найден, функция вернет пустую таблицу. Код функции представлен в листинге 17. Результат выполнения запроса представлен на рисунке 20.

Листинг 17 – Функция get_car_options()

```
CREATE OR REPLACE FUNCTION get_car_options(car_id INT)
RETURNS TABLE (
    ID_автомобиля INT,
```

```
    Опции TEXT
) AS $$
BEGIN
    RETURN QUERY
    SELECT c.id_car, ts.options
    FROM cars c
    INNER JOIN technical_specs ts ON ts.id_car = c.id_car
    WHERE c.id_car = car_id;
END;
$$ LANGUAGE plpgsql;
```

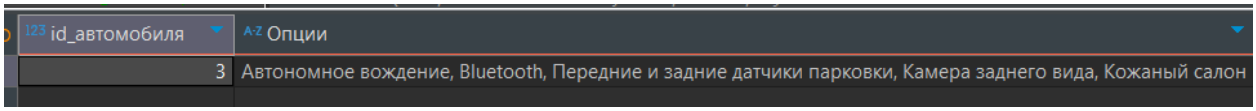


Рисунок 20 – Функция get_car_options()

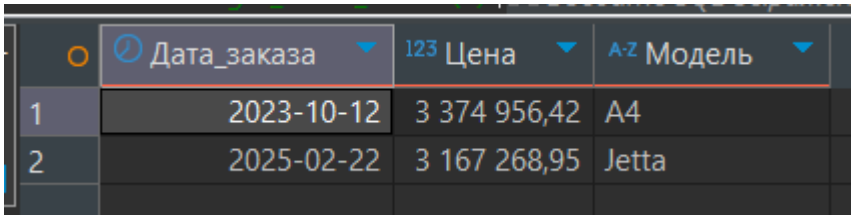
Функция №7: Информация обо всех покупках данного клиента (клиент – входной параметр);

Для выполнения данного запроса через функцию создается функция get_client_orders, которая принимает на вход параметр client_id — уникальный идентификатор клиента. Функция возвращает таблицу с информацией о всех покупках данного клиента: дата заказа, цена автомобиля и модель автомобиля. Если клиент не имеет заказов, функция вернет пустую таблицу. Код функции представлен в листинге 18. Результат выполнения запроса представлен на рисунке 21.

Листинг 18 – Функция get_client_orders()

```
CREATE OR REPLACE FUNCTION get_client_orders(client_id INT)
RETURNS TABLE (
    Дата_заказа DATE,
    Цена NUMERIC,
    Модель VARCHAR(50)  -- <- тип, соответствующий models.name
) AS $$
BEGIN
    RETURN QUERY
    SELECT o.order_date, c.price, m.name
    FROM orders o
```

```
    INNER JOIN cars c ON c.id_car = o.id_car
    INNER JOIN models m ON m.id_model = c.id_model
    WHERE o.id_client = client_id;
END;
$$ LANGUAGE plpgsql;
```



	Дата_заказа	Цена	Модель
1	2023-10-12	3 374 956,42	A4
2	2025-02-22	3 167 268,95	Jetta

Рисунок 21 - Функция get_client_orders()

Функция №8: Найти всех клиентов, которые заказали автомобили с мощностью двигателя более «x» л.с. и расходом топлива меньше «y» литров на 100 км, с указанием их имени, фамилии, модели автомобиля и даты заказа. (x, y – входные параметры).

Для выполнения данного запроса через функцию создается функция get_clients_by_power_and_consumption, которая принимает два параметра: min_power — минимальная мощность двигателя в л.с. и max_consumption — максимальный расход топлива на 100 км в литрах. Функция возвращает таблицу с фамилией, именем клиента, моделью автомобиля и датой заказа всех клиентов, заказавших автомобили с характеристиками, соответствующими указанным параметрам. Если таких клиентов нет, функция возвращает пустую таблицу. Код функции представлен в листинге 19. Результат выполнения запроса представлен на рисунке 22.

Листинг 19 – Функция get_clients_by_power_and_consumption()

```
CREATE OR REPLACE FUNCTION
get_clients_by_power_and_consumption(min_power NUMERIC,
max_consumption NUMERIC)
RETURNS TABLE (
    Фамилия TEXT,
    Имя TEXT,
    Модель TEXT,
    Дата_заказа DATE
) AS $$
BEGIN
```

```
    RETURN QUERY
    SELECT    cl.surname::TEXT,    cl.name::TEXT,    m.name::TEXT,
o.order_date
    FROM orders o
    INNER JOIN clients cl ON cl.id_client = o.id_client
    INNER JOIN cars c ON c.id_car = o.id_car
    INNER JOIN models m ON m.id_model = c.id_model
    INNER JOIN technical_specs ts ON ts.id_car = c.id_car
    WHERE ts.engine_power > min_power AND ts.fuel_consumption <
max_consumption;
END;
$$ LANGUAGE plpgsql;
```

	Фамилия	Имя	Модель	Дата_заказа
1	Овчинникова	Светозар	Santa Fe	2023-12-09
2	Константинова	Анисим	Tucson	2024-12-06
3	Константинова	Клавдий	Kuga	2023-08-05
4	Юдина	Добромысл	Optima	2024-05-23
5	Шубин	Евфросиния	RAV4	2023-11-18
6	Шубин	Евфросиния	RAV4	2023-11-18
7	Кононова	Викторин	Kuga	2024-11-08
8	Константинова	Анисим	Mondeo	2025-02-19
9	Осипова	Аркадий	Jetta	2025-02-22

Рисунок 22 - Функция get_clients_by_power_and_consumption()

Функция №9: Найти продавцов-консультантов, которые продали наибольшее количество автомобилей в разрезе двух дат, с указанием их имени, фамилии и количества проданных автомобилей. (Дата начала отчетного периода, Дата конца отчетного периода – входные параметры)

Для выполнения данного запроса создается функция get_top_sellers, которая принимает два входных параметра: start_date и end_date — дату начала и дату окончания отчетного периода. Функция возвращает информацию о продавцах-консультантах, которые продали наибольшее количество автомобилей в данном периоде, с указанием их фамилии, имени и количества проданных автомобилей. Внутренний запрос использует оконную

функцию RANK(), чтобы отсортировать консультантов по количеству продаж и выбрать тех, кто продал больше всего автомобилей. Если таких консультантов несколько, будет возвращен только первый по рангу. Код функции представлен в листинге 20. Результат выполнения запроса представлен на рисунке 23.

Листинг 20 – Функция get_top_sellers()

```
CREATE OR REPLACE FUNCTION get_top_sellers(start_date DATE,
end_date DATE)
RETURNS TABLE (
    Фамилия TEXT,
    Имя TEXT,
    Количество_продаж INT
) AS $$
BEGIN
    RETURN QUERY
    WITH cte AS (
        SELECT sc.surname::TEXT, sc.name::TEXT,
COUNT(o.id_order)::INTEGER AS total,
               RANK() OVER (ORDER BY COUNT(o.id_order) DESC) AS
rank
        FROM orders o
        INNER JOIN sales_consultants sc ON sc.id_consultant =
o.id_consultant
        WHERE o.order_date BETWEEN start_date AND end_date
        GROUP BY sc.id_consultant, sc.surname, sc.name
    )
    SELECT surname, name, total
    FROM cte
    WHERE rank = 1;
END;
$$ LANGUAGE plpgsql;
```

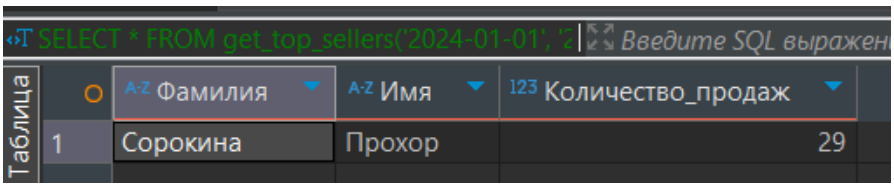


Рисунок 23 - Функция get_top_sellers()

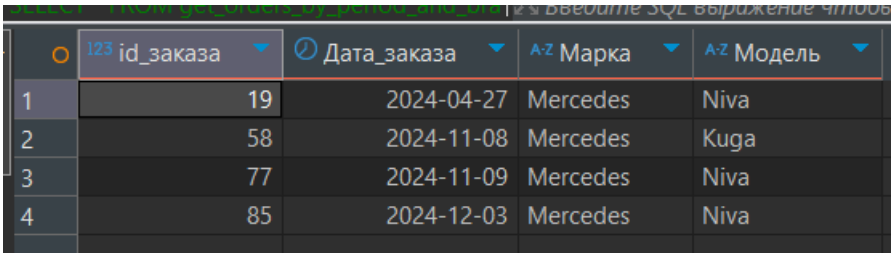
Функция №10: Выдать все заказы, которые были осуществлены в заданный отчетный период, в которых участвуют определенная марка машины (Дата начала отчетного периода, Дата конца отчетного периода, марка машины – входные параметры).

Для выполнения данного запроса создается функция get_orders_by_period_and_brand, которая принимает три входных параметра: start_date, end_date — даты начала и окончания отчетного периода, и brand — марку автомобиля. Функция возвращает таблицу с информацией о заказах, которые были выполнены в указанном периоде и касаются указанной марки автомобиля. Результат включает ID заказа, дату заказа, марку автомобиля и модель. Код функции представлен в листинге 21. Результат выполнения запроса представлен на рисунке 24.

Листинг 21 – Функция get_orders_by_period_and_brand()

```
CREATE OR REPLACE FUNCTION
get_orders_by_period_and_brand(start_date DATE, end_date DATE,
brand TEXT)
RETURNS TABLE (
    ID_заказа INT,
    Дата_заказа DATE,
    Марка TEXT,
    Модель TEXT
) AS $$
BEGIN
    RETURN QUERY
    SELECT o.id_order, o.order_date, b.name::TEXT, m.name::TEXT
    FROM orders o
    INNER JOIN cars c ON c.id_car = o.id_car
    INNER JOIN models m ON m.id_model = c.id_model
    INNER JOIN brands b ON b.id_brand = m.id_brand
    WHERE o.order_date BETWEEN start_date AND end_date
```

```
    AND b.name = brand;
END;
$$ LANGUAGE plpgsql;
```



	id_заказа	Дата_заказа	Марка	Модель
1	19	2024-04-27	Mercedes	Niva
2	58	2024-11-08	Mercedes	Kuga
3	77	2024-11-09	Mercedes	Niva
4	85	2024-12-03	Mercedes	Niva

Рисунок 24 - Функция get_orders_by_period_and_brand()

3.2 Разработка триггеров

Триггер в PostgreSQL - это объект базы данных, который автоматически выполняет заданное действие в ответ на изменение данных в таблице или представлении [4]. Триггеры позволяют автоматизировать процесс обработки событий, таких как вставка, обновление или удаление данных.

Типы триггеров:

1. AFTER - триггер, который выполняется после того, как операция (INSERT, UPDATE, DELETE) была выполнена над строками таблицы. Обычно используется для выполнения операций, которые не изменяют данные (например, логирование, уведомления).
2. BEFORE - триггер, который выполняется до того, как операция (INSERT, UPDATE, DELETE) будет выполнена. Он может быть использован для модификации данных перед их вставкой или обновлением.
3. INSERT - триггер, который срабатывает, когда выполняется операция вставки (INSERT) новой строки в таблицу.
4. UPDATE - триггер, который срабатывает при изменении существующих данных в таблице с помощью операции UPDATE.
5. DELETE - триггер, который срабатывает, когда строка удаляется из таблицы с помощью операции DELETE

Триггер №1: Проверка автомобиля на доступность перед добавлением нового заказа в таблицу orders.

Проверяет существует ли автомобиль с указанным id. Если автомобиль найден получает его текущий статус наличия и количество на складе. Если статус нет, выбрасывается исключение. Если статус да и количество на складе больше нуля, выполняется уменьшение количества на складе на 1. Если после уменьшения количество становится равным нулю, статус наличия меняется на «нет». Если автомобиль с заданным идентификатором не найден, выбрасывается исключение с сообщением об ошибке. Программный код триггера представлен в листинге 22. Результат работы триггера показан на рисунках 25 – 29.

Листинг 22 – Триггер trigger_check_car_availability()

```
CREATE OR REPLACE FUNCTION check_car_availability()
RETURNS TRIGGER AS $$
DECLARE
    car_stock_status VARCHAR(20);
    car_quantity_in_stock INT;
BEGIN
    IF EXISTS (SELECT 1 FROM cars WHERE id_car = NEW.id_car) THEN
        SELECT stock_status, quantity_in_stock
        INTO car_stock_status, car_quantity_in_stock
        FROM cars
        WHERE id_car = NEW.id_car;

        IF car_stock_status = 'нет' THEN
            RAISE EXCEPTION 'Автомобиль с id % не доступен для
заказа (наличие = "нет")', NEW.id_car;
        ELSIF car_stock_status = 'да' AND car_quantity_in_stock >
0 THEN
            UPDATE cars
            SET quantity_in_stock = quantity_in_stock - 1,
                stock_status = CASE
                                    WHEN quantity_in_stock - 1 = 0
THEN 'нет'
                                    ELSE 'да'
                                END
            WHERE id_car = NEW.id_car;
```

```
        ELSIF car_stock_status = 'да' AND car_quantity_in_stock =
0 THEN
            UPDATE cars
            SET stock_status = 'нет', status = 'не в наличии'
            WHERE id_car = NEW.id_car;
        END IF;
    ELSE
        RAISE EXCEPTION 'Автомобиль с id % не найден', NEW.id_car;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_check_car_availability
BEFORE INSERT ON orders
FOR EACH ROW
EXECUTE FUNCTION check_car_availability();
```

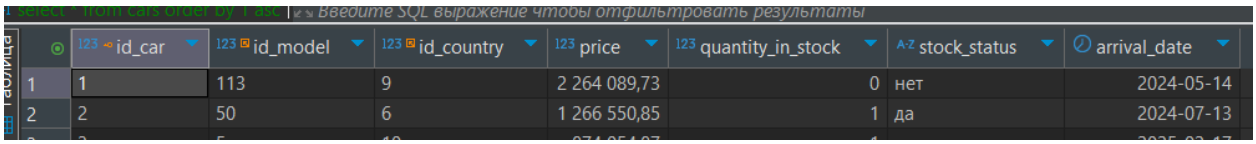


Рисунок 25 – Таблица cars до создание заказа

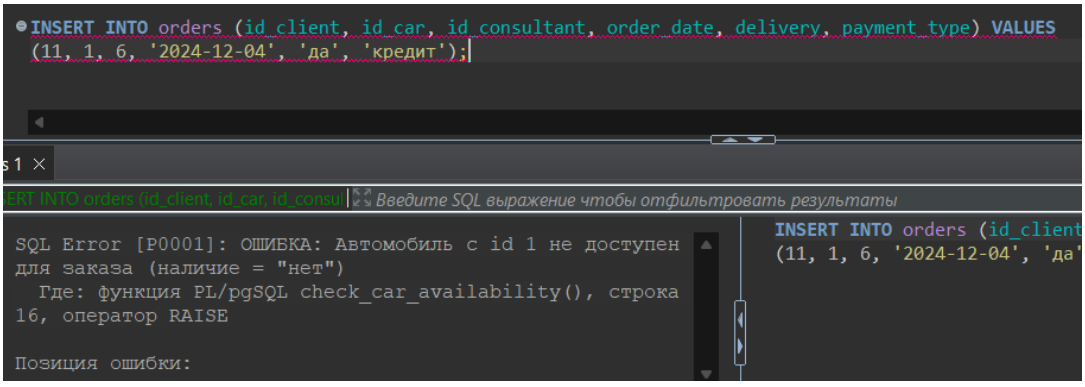


Рисунок 26 – Ошибка при вставке автомобиля с id_car = 1

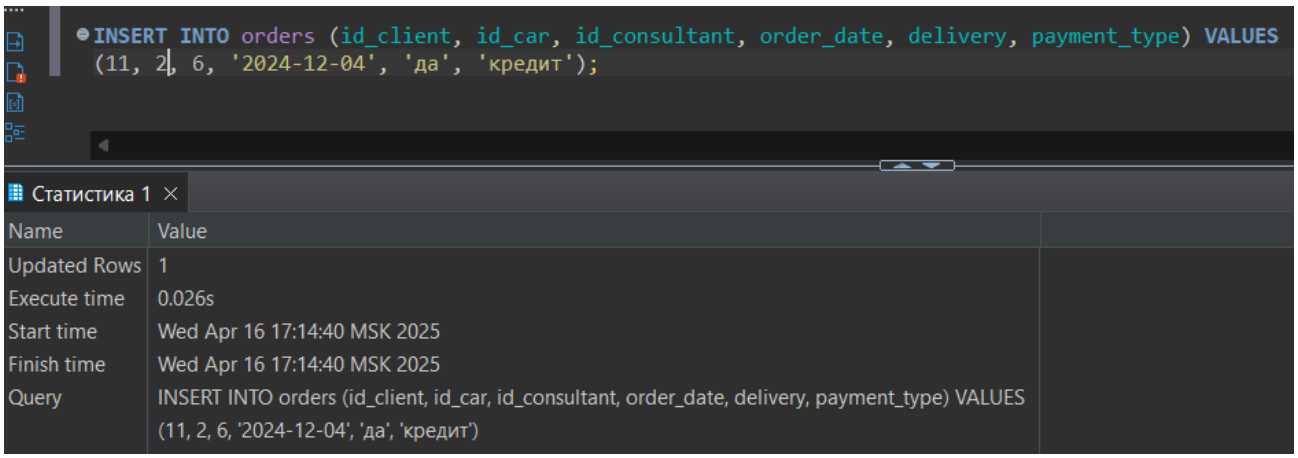


Рисунок 27 – Создание заказа с автомобилем с id_car = 2

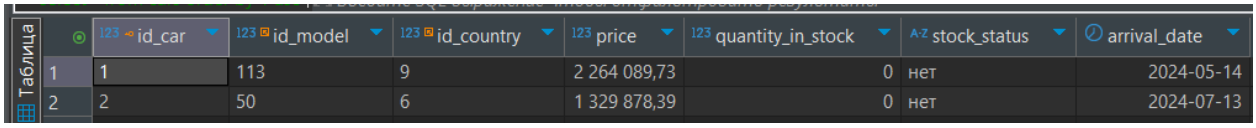


Рисунок 28 – Таблица cars после вставки

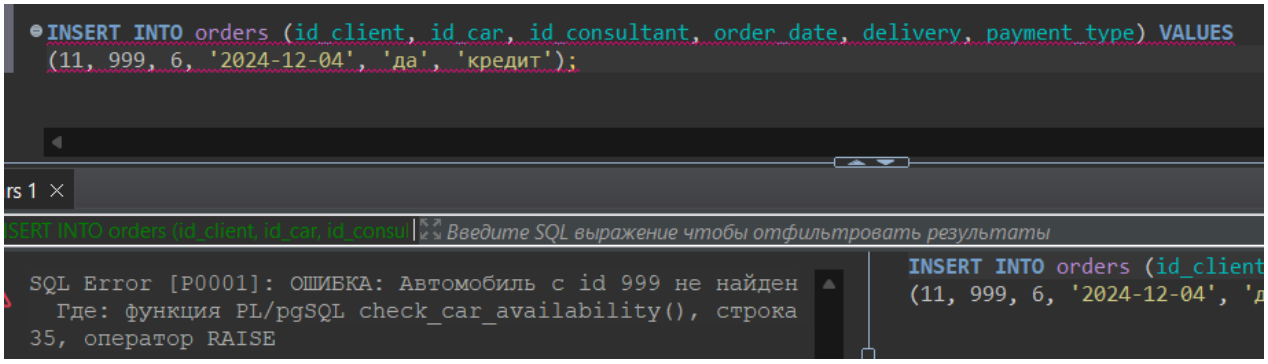


Рисунок 29 – Ошибка при вставке автомобиля с несуществующим id_car

Триггер №2: Обновление информации о наличии автомобиля при отмене заказа.

Срабатывает после изменения значения поля order_status в таблице orders. Если статус заказа меняется на «отменен», происходит увеличение количества автомобилей на складе на 1 по соответствующему идентификатору автомобиля. При этом значение поля stock_status также обновляется на «да», что означает, что автомобиль снова доступен для заказа. Программный код триггера представлен в листинге 23. Результат работы триггера показан на рисунке 30.

Листинг 23 – Триггер trigger_update_car_stock_on_order_cancellation()

```
CREATE OR REPLACE TRIGGER trigger_check_car_availability
BEFORE INSERT ON orders
FOR EACH ROW
```

```
EXECUTE FUNCTION check_car_availability();

CREATE OR REPLACE FUNCTION
update_car_stock_on_order_cancellation()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.order_status = 'отменен' THEN
        UPDATE cars
        SET quantity_in_stock = quantity_in_stock + 1,
            stock_status = 'да'
        WHERE id_car = NEW.id_car;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER
trigger_update_car_stock_on_order_cancellation
AFTER UPDATE OF order_status ON orders
FOR EACH ROW
WHEN (NEW.order_status = 'отменен')
EXECUTE FUNCTION update_car_stock_on_order_cancellation();
```

Для тестирования триггера обновим значение order_status у заказа с id_car = 2 из предыдущего примера, после обновления в таблице cars у id_car = 2 обновится значения наличия товара.

id_car	id_model	id_country	price	quantity_in_stock	stock_status	arrival_date
1	113	9	2 264 089,73	0	нет	2024-05-14
2	50	6	1 396 372,31	1	да	2024-07-13

Рисунок 30 – Таблица cars после отмены заказа

Триггер №3–5. Логирования операций с таблицами orders, clients, cars.

Предварительно создана таблица Logs. Фиксирует все изменения, происходящие с таблицами orders, clients, cars. При добавлении новой записи в таблицу в журнал (logs) записываются тип операции INSERT, имя таблицы, идентификатор записи, новые значения и имя пользователя. При обновлении фиксируются тип операции UPDATE, идентификатор

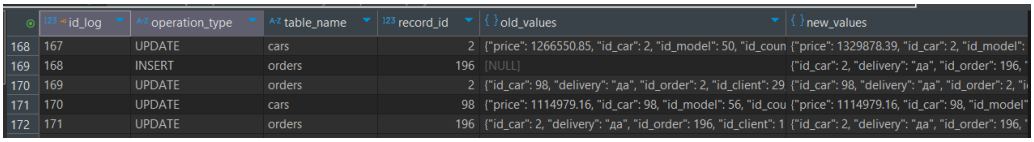
записи, старые и новые значения. При удалении - тип операции DELETE, идентификатор удалённой записи и её значения до удаления. Программный код триггера представлен в листинге 24. Результаты работы триггера отображены на рисунке 31.

Листинг 24 – Триггеры для логирования

```
CREATE TABLE if not exists logs (
    id_log SERIAL PRIMARY KEY,
    operation_type VARCHAR(10) NOT NULL CHECK (operation_type IN
('INSERT', 'UPDATE', 'DELETE')),
    table_name VARCHAR(50) NOT NULL,
    record_id INT NOT NULL,
    old_values JSONB,
    new_values JSONB,
    operation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    logged_by VARCHAR(50) NOT NULL
);

CREATE OR REPLACE FUNCTION log_orders_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
new_values, logged_by)
        VALUES ('INSERT', 'orders', NEW.id_order, to_jsonb(NEW),
current_user);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, new_values, logged_by)
        VALUES ('UPDATE', 'orders', OLD.id_order, to_jsonb(OLD),
to_jsonb(NEW), current_user);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, logged_by)
```

```
        VALUES ('DELETE', 'orders', OLD.id_order, to_jsonb(OLD),
current_user);
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER trigger_log_orders
AFTER INSERT OR UPDATE OR DELETE ON orders
FOR EACH ROW
EXECUTE FUNCTION log_orders_trigger_function();

CREATE OR REPLACE FUNCTION log_clients_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
new_values, logged_by)
        VALUES     ('INSERT',     'clients',     NEW.id_client,
to_jsonb(NEW), current_user);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, new_values, logged_by)
        VALUES     ('UPDATE',     'clients',     OLD.id_client,
to_jsonb(OLD), to_jsonb(NEW), current_user);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, logged_by)
        VALUES ('DELETE', 'clients', OLD.id_client,
to_jsonb(OLD), current_user);
        RETURN OLD;
    END IF;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER trigger_log_clients
AFTER INSERT OR UPDATE OR DELETE ON clients
FOR EACH ROW
EXECUTE FUNCTION log_clients_trigger_function();

CREATE OR REPLACE FUNCTION log_cars_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
new_values, logged_by)
        VALUES ('INSERT', 'cars', NEW.id_car, to_jsonb(NEW),
current_user);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, new_values, logged_by)
        VALUES ('UPDATE', 'cars', OLD.id_car, to_jsonb(OLD),
to_jsonb(NEW), current_user);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO logs (operation_type, table_name, record_id,
old_values, logged_by)
        VALUES ('DELETE', 'cars', OLD.id_car, to_jsonb(OLD),
current_user);
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE or replace TRIGGER trigger_log_cars
```



	id_log	operation_type	table_name	record_id	old_values	new_values
168	167	UPDATE	cars	2	{"price": 1266550.85, "id_car": 2, "id_model": 50, "id_coun	{"price": 1329878.39, "id_car": 2, "id_model":
169	168	INSERT	orders	196	[NULL]	{"id_car": 2, "delivery": "да", "id_order": 196, "
170	169	UPDATE	orders	2	{"id_car": 98, "delivery": "да", "id_order": 2, "id_client": 29	{"id_car": 98, "delivery": "да", "id_order": 2, "i
171	170	UPDATE	cars	98	{"price": 1114979.16, "id_car": 98, "id_model": 56, "id_cou	{"price": 1114979.16, "id_car": 98, "id_model"
172	171	UPDATE	orders	196	{"id_car": 2, "delivery": "да", "id_order": 196, "id_client": 1	{"id_car": 2, "delivery": "да", "id_order": 196, "

Рисунок 31 – Таблица logs после изменения данных в таблицах

Триггер №6: Проверка корректности даты рождения клиента при добавлении или обновлении данных в таблице clients.

Проверяет, что значение даты рождения клиента находится в допустимом диапазоне - не ранее 1 января 1900 года и не позднее текущей даты. Если дата рождения выходит за указанные пределы, выбрасывается исключение с сообщением об ошибке. Также выполняется проверка на возраст: клиент должен быть не младше 18 лет на момент внесения записи. В противном случае также генерируется исключение. Программный код триггера представлен в листинге 25. Результат работы триггера показан на рисунках 32 – 34.

Листинг 32 – Триггер trigger_check_birth_date()

```
CREATE OR REPLACE FUNCTION check_birth_date()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.birth_date < '1900-01-01'::DATE OR NEW.birth_date >
CURRENT_DATE THEN
        RAISE EXCEPTION 'Дата рождения должна быть в пределах от
1900-01-01 до сегодняшней даты';
    END IF;

    IF NEW.birth_date > CURRENT_DATE - INTERVAL '18 years' THEN
        RAISE EXCEPTION 'Клиент должен быть не младше 18 лет';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_check_birth_date
BEFORE INSERT OR UPDATE ON clients
```

```
FOR EACH ROW
EXECUTE FUNCTION check_birth_date();
```

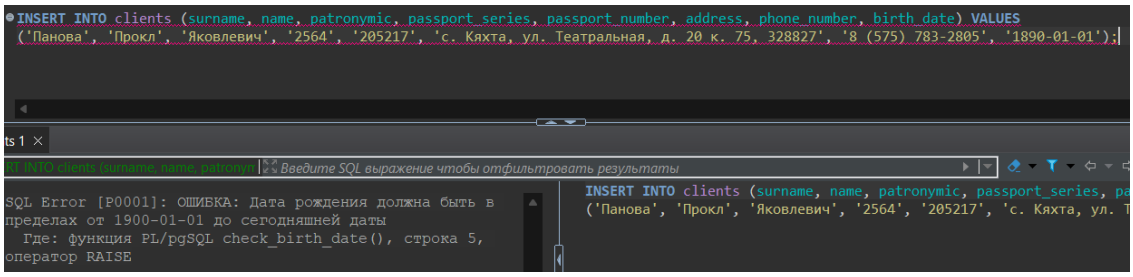


Рисунок 32 – Работа триггера при дате до 1900 года

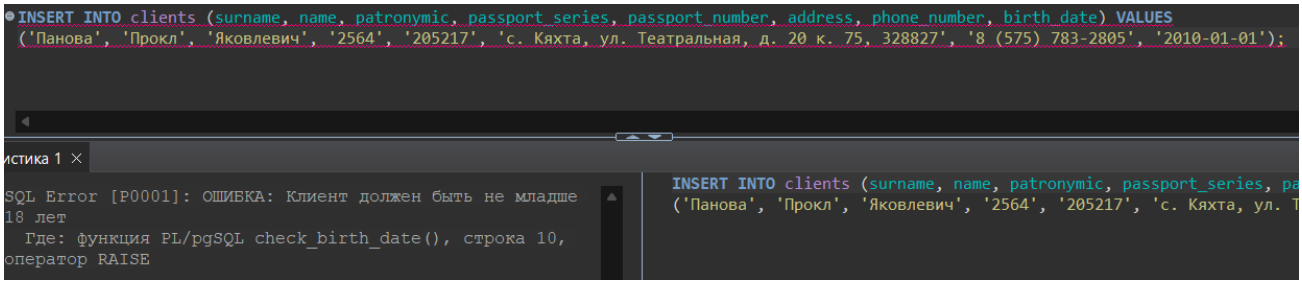


Рисунок 33 – Работа триггера при возрасте до 18 лет

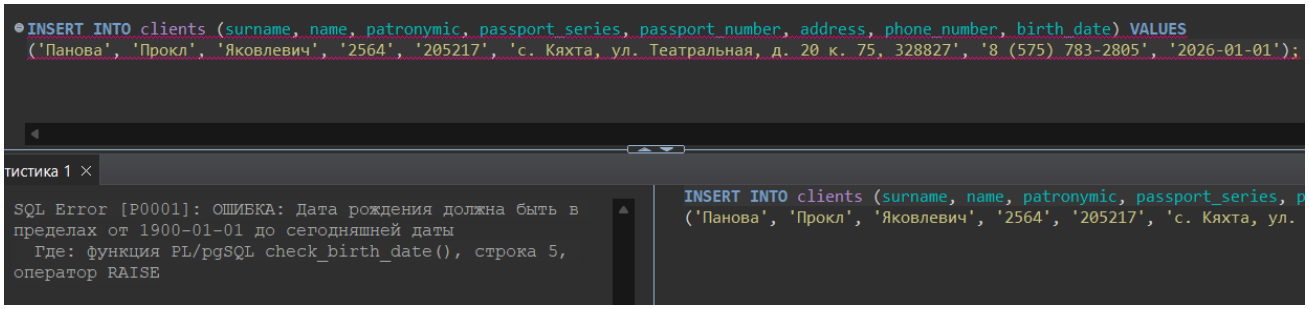


Рисунок 34 – Работа триггера при дате из будущего

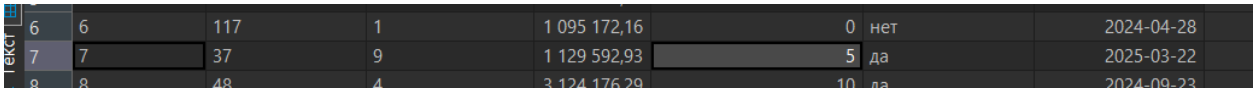
Триггер №7. Автоматическое увеличение цены автомобиля при обновлении информации о количестве на складе.

Срабатывает перед обновлением записи в таблице cars, если новое количество автомобилей на складе (quantity_in_stock) становится меньше или равно 1. В случае выполнения условия цена автомобиля (price) автоматически увеличивается на 5%. Это позволяет учитывать дефицит товара и повышать цену на редкие позиции. Программный код триггера приведён в листинге 33. Результат работы триггера демонстрируется на рисунках 35 - 36.

Листинг 33 – Триггер increase_car_price()

```
CREATE OR REPLACE FUNCTION increase_car_price()
RETURNS TRIGGER AS $$
```

```
BEGIN
    IF NEW.quantity_in_stock <= 1 THEN
        NEW.price = NEW.price * 1.05;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_increase_car_price
BEFORE UPDATE ON cars
FOR EACH ROW
WHEN (NEW.quantity_in_stock <= 1)
EXECUTE FUNCTION increase_car_price();
```



6	6	117	1	1 095 172,16	0	нет	2024-04-28
7	7	37	9	1 129 592,93	5	да	2025-03-22

Рисунок 35 - Стоимость автомобиля с car_id когда количество на складе = 5

6	6	117	1	1 095 172,16	0	нет	2024-04-28
7	7	37	9	1 186 072,58	1	да	2025-03-22
8	8	48	4	3 124 176,29	10	да	2024-09-23

Рисунок 36 – Стоимость после изменения количества на складе

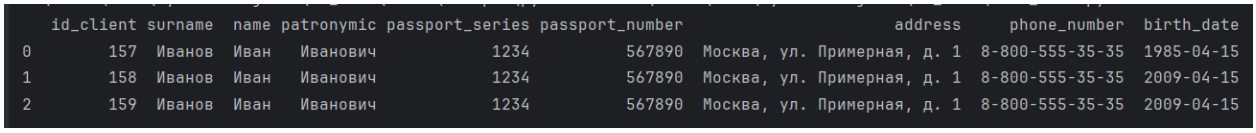
3.3 Обработка и визуализация данных

Для визуализации данных был использован язык программирования Python. Чтобы установить соединение, используется connect() из модуля psycopg2 [7]. Для визуализации данных используется библиотека matplotlib [6]. Программный код соединения приведен в листинге 34. Результат работы демонстрируется на рисунке 37.

Листинг 34 – Соединения с СУБД Postgre

```
import psycopg2
import pandas as pd
import warnings as wn
wn.filterwarnings('ignore')

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
pd.set_option('display.max_colwidth', None)
pd.set_option('display.width', 0)

conn = psycopg2.connect(database="students",
                                user="postgres",
                                password="postgres",
                                host="127.0.0.1",
                                port="5432")
with conn.cursor() as cursor:
    cursor.execute("SET search_path TO auto;")
query = "SELECT * FROM clients;"
df = pd.read_sql_query(query, conn)
print(df.head(3))
conn.close()
```



	id_client	surname	name	patronymic	passport_series	passport_number	address	phone_number	birth_date
0	157	Иванов	Иван	Иванович	1234	567890	Москва, ул. Примерная, д. 1	8-800-555-35-35	1985-04-15
1	158	Иванов	Иван	Иванович	1234	567890	Москва, ул. Примерная, д. 1	8-800-555-35-35	2009-04-15
2	159	Иванов	Иван	Иванович	1234	567890	Москва, ул. Примерная, д. 1	8-800-555-35-35	2009-04-15

Рисунок 37 – Чтение данных через Python

Визуализация 1: Распределение клиентов по возрастам и гендерам. По данному графику видно, что наибольшая долю клиентов составляет возраст 30–50 лет, доля женщин и мужчин минимально отличается. Данная визуализация будет полезно для экономиста и директора магазина, с помощью нее можно определить топ клиентов по возрасту. Программный код представлен в листинге 35. Визуализация показана на рисунке 38.

Листинг 35 – Функция для построения визуализации 1

```
query = "SELECT *, EXTRACT(YEAR FROM AGE(CURRENT_DATE, birth_date))::int AS age FROM clients;"
df = pd.read_sql_query(query, conn)
conn.close()

if 'gender' not in df.columns:
    import numpy as np
    np.random.seed(42)
```

```
    df['gender'] = np.random.choice(['М', 'Ж'], size=len(df))

bins = [0, 19, 29, 39, 49, 59, 69, 79, 89, 100]
labels = ['<20', '20-29', '30-39', '40-49', '50-59', '60-69',
'70-79', '80-89', '90+']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels,
right=True)

grouped = df.groupby(['age_group',
'gender']).size().unstack(fill_value=0)

grouped.plot(kind='bar', figsize=(12, 6), width=0.8,
color=['cornflowerblue', 'lightpink'])

plt.title('Распределение клиентов по возрастным группам и полу')
plt.xlabel('Возрастная группа')
plt.ylabel('Количество клиентов')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.legend(title='Пол')
plt.tight_layout()
plt.show()
```

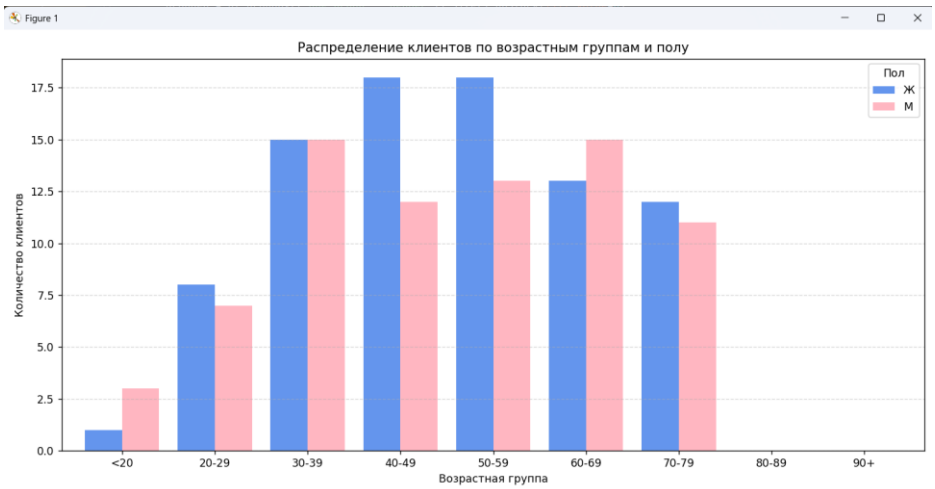


Рисунок 38 – Визуализация 1

Визуализация 2: Количество заказов по маркам автомобилей. По данному графику видно, что самая продаваемая марка это Audi. Данная визуализация будет полезна директору магазину, с помощью нее можно оценить спрос на продукцию. Программный код представлен в листинге 36. Визуализация показана на рисунке 39.

Листинг 36 - Функция для построения визуализации 2

```
def vis2():
    query = """
    SELECT b.name, COUNT(*) as total_orders
    FROM orders o
    JOIN cars c on o.id_car = c.id_car
    JOIN models m on c.id_model = m.id_model
    JOIN brands b on b.id_brand = m.id_brand
    GROUP BY b.name
    ORDER BY total_orders DESC;
    """

    df_orders = pd.read_sql_query(query, conn)

    plt.figure(figsize=(12, 6))
    sns.barplot(data=df_orders, x='name', y='total_orders',
palette='crest')
    plt.title("Количество заказов по маркам автомобилей",
fontsize=14)
    plt.xlabel("Марка автомобиля")
    plt.ylabel("Количество заказов")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

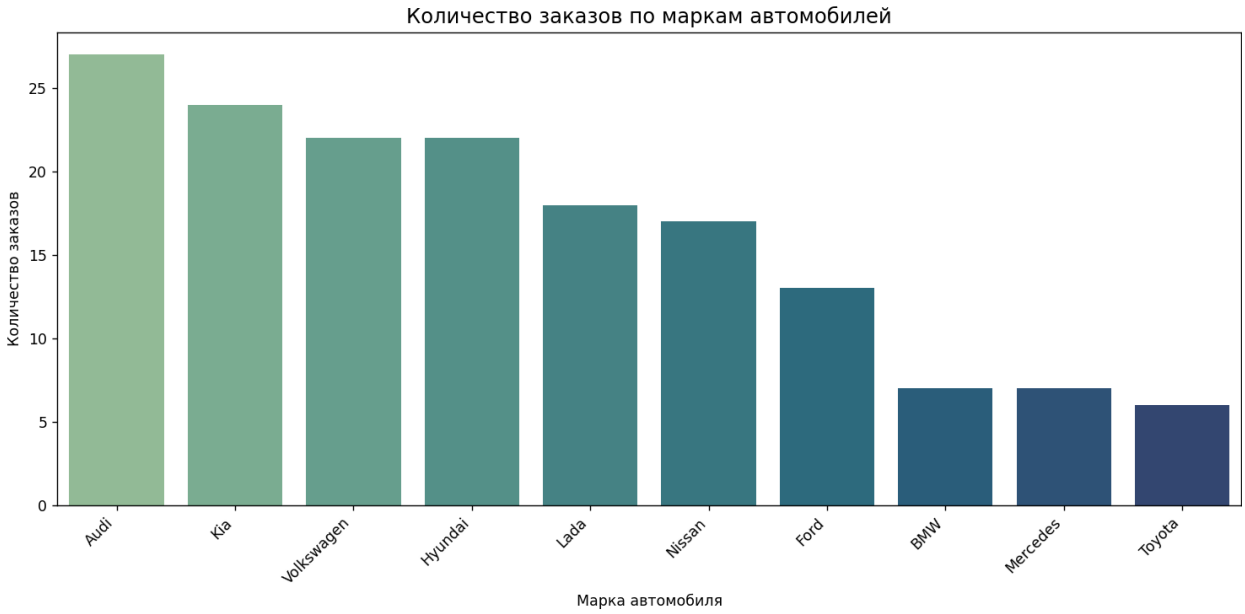


Рисунок 39 – Визуализация 2

Визуализация 3: Количество заказов по модели определённой марки. По данному графику видно, какие модели определенное марки были проданы. Данная визуализация будет полезна для директора магазина, с помощью нее можно определить спрос на модель определённой марки. Программный код представлен в листинге 37. Визуализация показана на рисунке 40.

Листинг 37 – Функция для построения визуализации 3

```
def vis3(marka):
    query = f"""
    SELECT m.name, COUNT(*) as total_orders
    FROM orders o
    JOIN cars c on o.id_car = c.id_car
    JOIN models m on c.id_model = m.id_model
    JOIN brands b on b.id_brand = m.id_brand where lower(b.name)
= lower('{marka}')
    GROUP BY m.name
    ORDER BY total_orders DESC;
    """

    df_orders = pd.read_sql_query(query, conn)
```

```
    plt.figure(figsize=(12, 6))
    sns.barplot(data=df_orders, x='name', y='total_orders',
palette='crest')
    plt.title(f"Модели {marka}", fontsize=14)
    plt.xlabel("Марка автомобиля")
    plt.ylabel("Количество заказов")
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

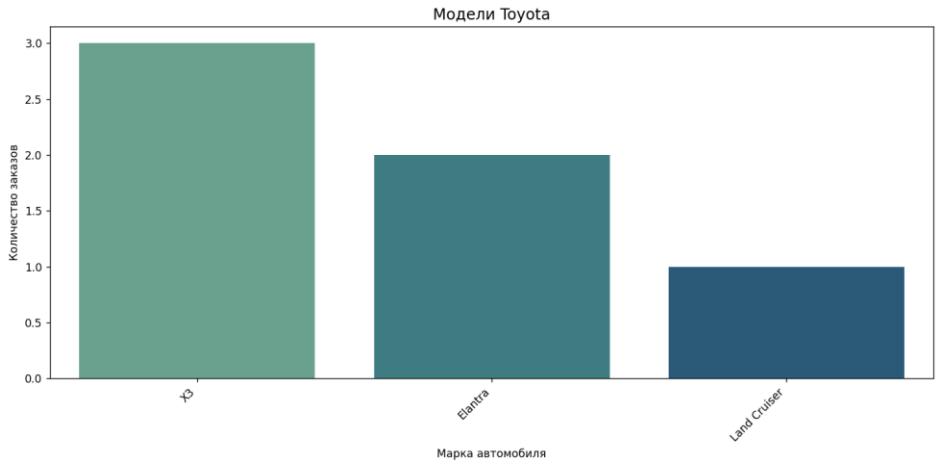


Рисунок 40 – Визуализация 3

Визуализация 4: Сумма заказов за определённый год. По данному графику видно, что пик продаж приходится на декабрь. Данная визуализация полезна экономисту и директору. Программный код представлен в листинге 38. Визуализация показана на рисунке 41.

Листинг 38 - Функция для построения визуализации 4

```
def vis4(year):
    if  2025 >= year >= 2000:
        query = f"""
        SELECT TO_CHAR(order_date, 'TMMonth') AS month,
SUM(c.price) AS revenue
        FROM orders o
        inner join cars c on c.id_car = o.id_car
        where EXTRACT(YEAR FROM order_date) = {year}
        GROUP BY month
```

```
        ORDER BY month;
        """

        df_revenue = pd.read_sql_query(query, conn)

        # Визуализация
        plt.figure(figsize=(12, 6))
        sns.lineplot(data=df_revenue, x='month', y='revenue',
marker='o', linewidth=2, color='green')
        plt.title(f"Общая сумма заказов по месяцам за {year}
год", fontsize=14)
        plt.xlabel("Месяц")
        plt.ylabel("Сумма заказов, ₽")
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.tight_layout()
        plt.show()
    else:
        print("Введите корректное значение (2000 - 2025 год)")
```

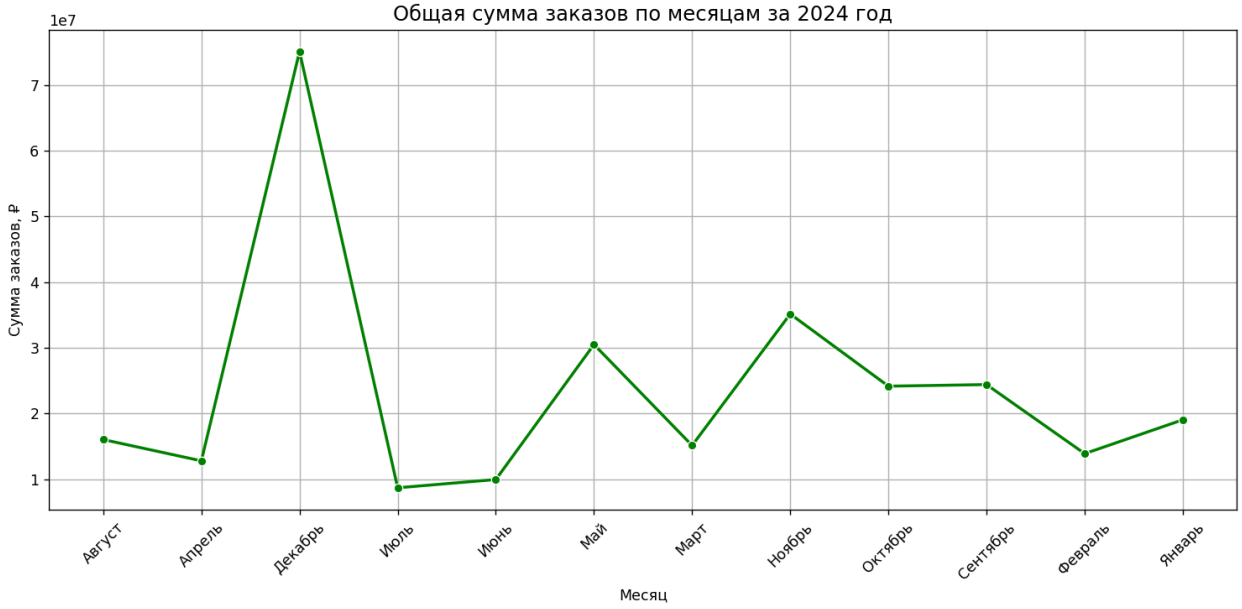


Рисунок 41 – Визуализация 4

Визуализация 5: Сводная информация по консультантом. Программный код представлен в листинге 39. Данная визуализация полезна для директора магазина. Визуализация показана на рисунках 42-43.

Листинг 39 - Функция для построения визуализации 5.1, 5.2.

```
def vis5(option):
    if option == 1:
        query = """
        SELECT
            sc.surname || ' ' || sc.name || ' ' || sc.patronymic
AS full_name,
            SUM(c.price) AS total_sales
        FROM auto.sales_consultants sc
        JOIN auto.orders o ON sc.id_consultant = o.id_consultant
        JOIN auto.cars c ON o.id_car = c.id_car
        WHERE EXTRACT(YEAR FROM o.order_date) = 2025
        GROUP BY full_name
        ORDER BY total_sales DESC;
        """
        title = "Сумма заказов по консультантам (2025 год)"
        value_column = "total_sales"

    elif option == 2:
        query = """
        SELECT
            sc.surname || ' ' || sc.name || ' ' || sc.patronymic
AS full_name,
            COUNT(o.id_order) AS order_count
        FROM auto.sales_consultants sc
        JOIN auto.orders o ON sc.id_consultant = o.id_consultant
        WHERE EXTRACT(YEAR FROM o.order_date) = 2025
        GROUP BY full_name
        ORDER BY order_count DESC;
        """
        title = "Количество заказов по консультантам (2025 год)"
```

```
        value_column = "order_count"

    else:
        print("Неверный параметр. Введите 1 или 2.")
        return

    df = pd.read_sql_query(query, conn)


    plt.figure(figsize=(10, 6))
    sns.barplot(data=df, x=value_column, y="full_name",
palette="viridis")
    plt.title(title)
    plt.xlabel("Значение")
    plt.ylabel("Консультант")
    plt.tight_layout()
    plt.show()
```

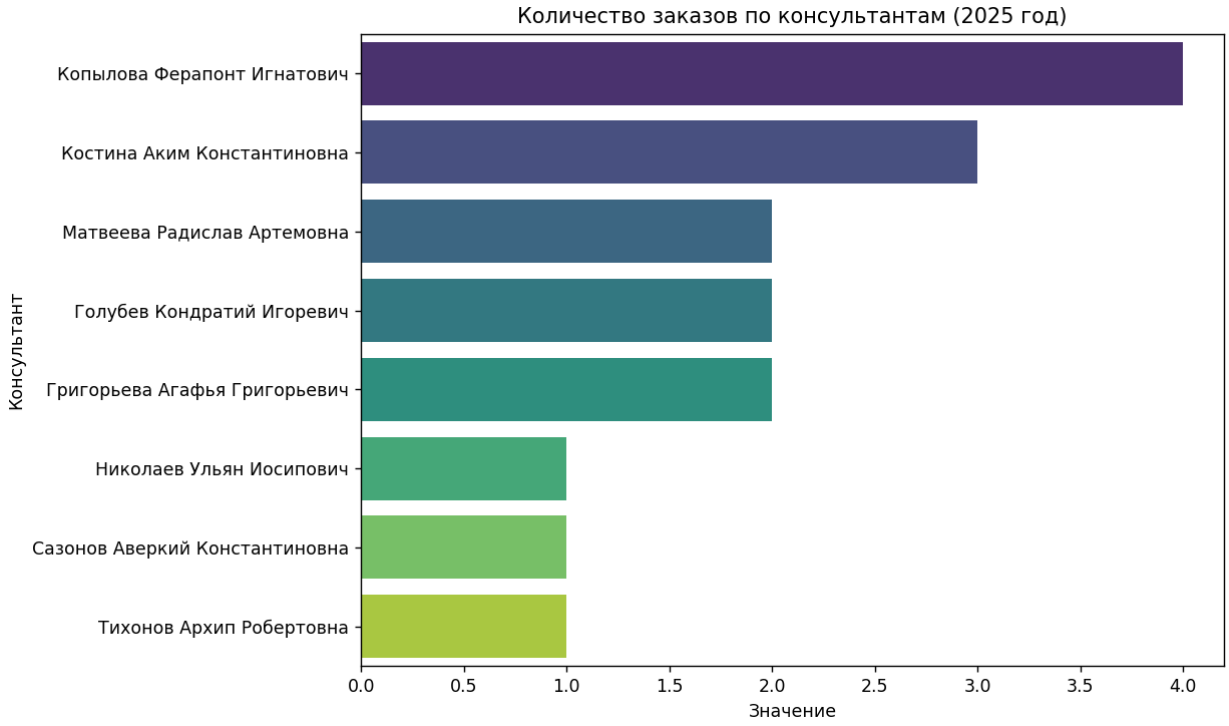


Рисунок 42 – Визуализация 5.1

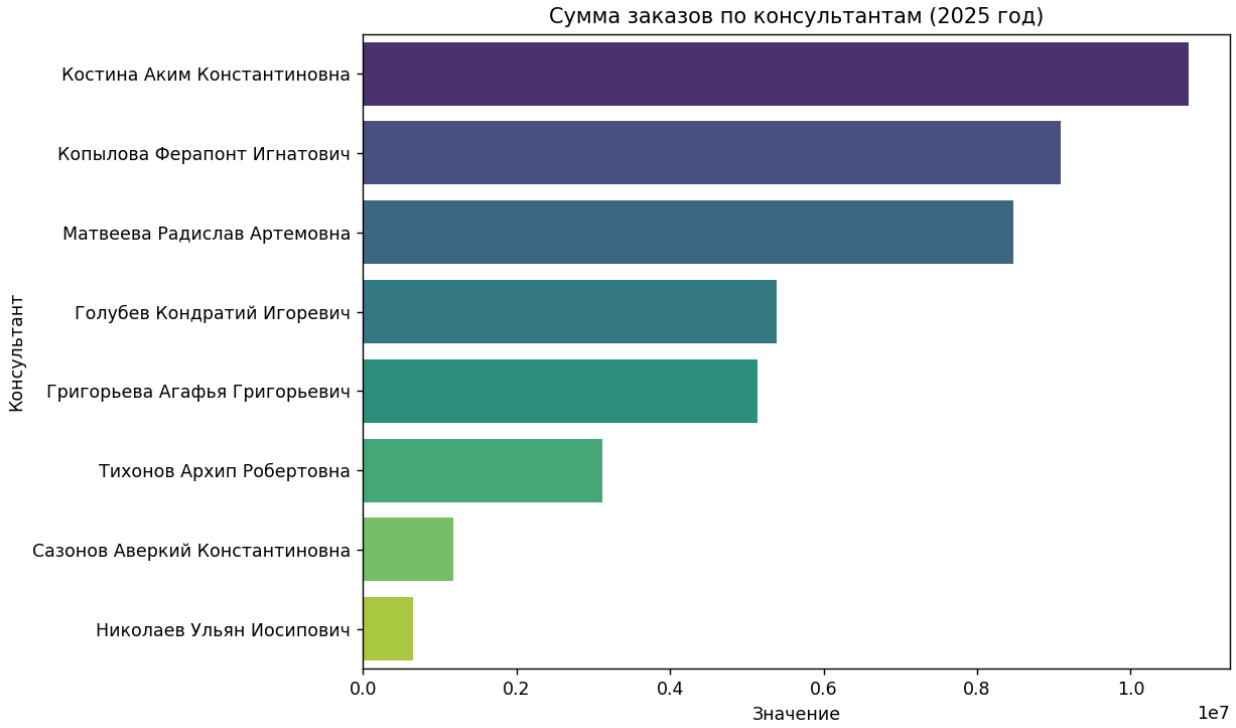


Рисунок 43 – Визуализация 5.2

4 Тестирование разработанной ИС

Для разработки пользовательского интерфейса был использован модуль Tkinter. Tkinter является кроссплатформенным графическим интерфейсом для Python, который работает с библиотекой Tk [5]. Этот модуль предоставляет элементы графического интерфейса (GUI - Graphical User Interface), с помощью которых можно создавать различные приложения.

Интерфейс приложения разделен на 4 файла. Каждый файл отвечает за своё функциональное окно. Первое окно – окно ввода. Данное окно отвечает за подключение пользователя к БД с определёнными правами. Окно содержит поля для ввода названия роли и пароля, а также кнопку подтверждения ввода. При отсутствии пользователя в БД выводится оосбщение об ошибке. Полный код файла, содержащего код окна входа, находится в приложении В.

Алгоритм работает таким образом, что изначально создаётся основное окно, в нём определяются поля для ввода и кнопки. Для кнопки ввода написана функция login, данная функция получает данные из полей ввода при помощи которых выполняется подключение к БД. При успешном подключении выполняется функция open_main_window в которую передаётся пароль и роль пользователя, для последующего подключения к БД, также в

функцию передаётся основное окно предложения. При неудачной попытке входы выводится сообщение об ошибке. На рисунках 44 – 46 показана работа окна ввода.

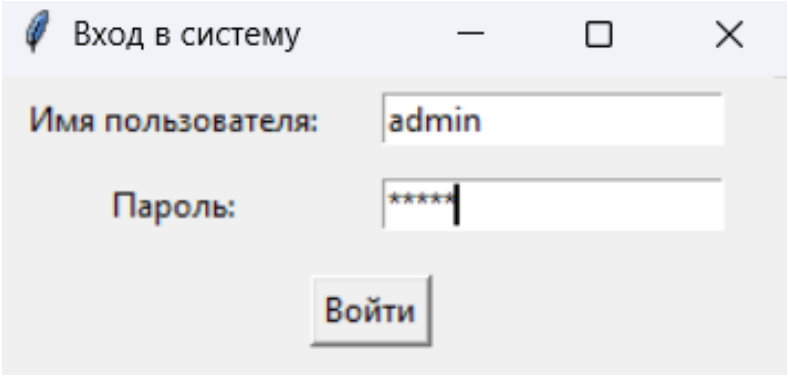


Рисунок 44 – Окно ввода

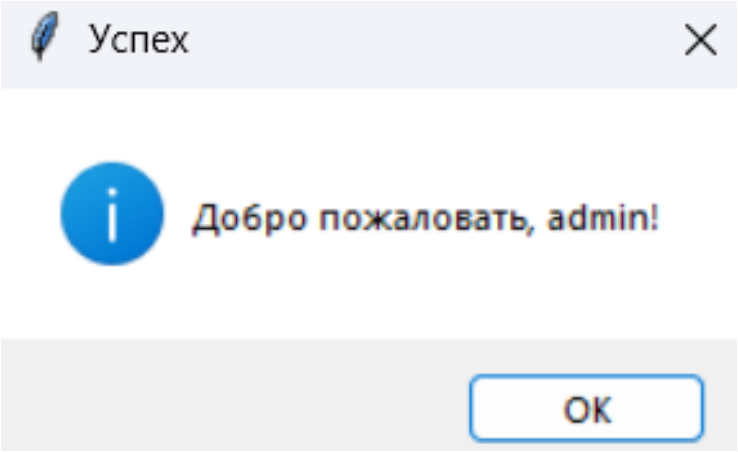


Рисунок 45 – Успешный вход под ролью администратора

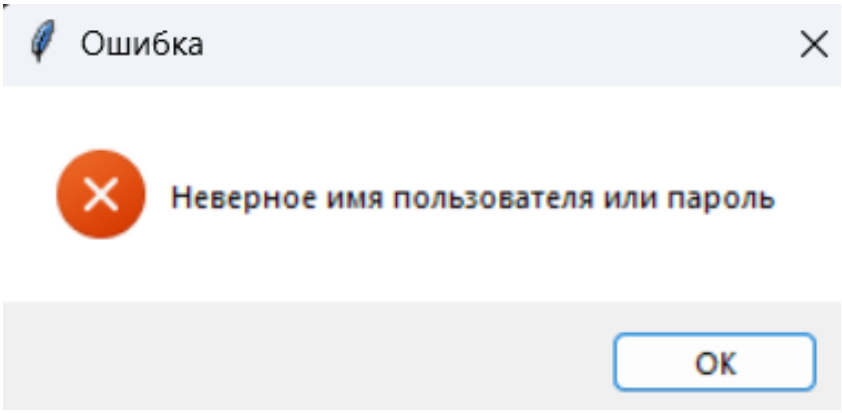


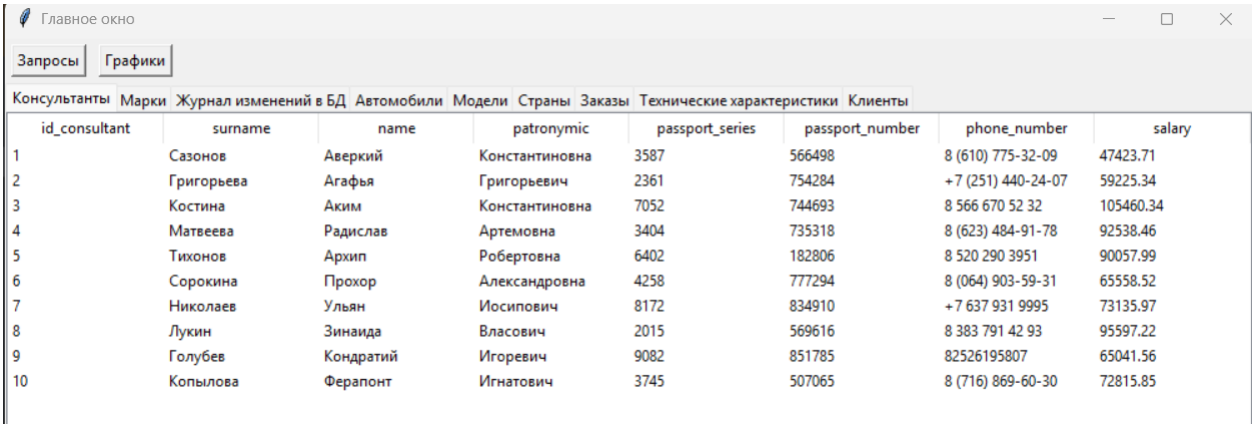
Рисунок 46 – Сообщение об ошибке

После прохождения авторизации открывается основное окно. Данное окно содержит отображение всех доступных пользователю таблиц, а также кнопки для перехода к выполнению запросов и создания графиков. Код файла, содержащего основное окно, представлен в Приложении Г.

Основное окно представлено ввиде функции open_main_window. Получив на вход переменную, содержащую главное окно приложения, функция перестраивает окно, добавляя в него таблицу, список доступных таблиц и кнопки: «Запросы», «Графики».

Данные таблиц получаются при помощи переподключения к БД, и создания запроса к ней.

На рисунке 47 представлено основное окно для пользователя admin, а на рисунке 48 для economist.



id_consultant	surname	name	patronymic	passport_series	passport_number	phone_number	salary
1	Сазонов	Аверкий	Константиновна	3587	566498	8 (610) 775-32-09	47423.71
2	Григорьева	Агафья	Григорьевич	2361	754284	+7 (251) 440-24-07	59225.34
3	Костина	Аким	Константиновна	7052	744693	8 566 670 52 32	105460.34
4	Матвеева	Радислав	Артемовна	3404	735318	8 (623) 484-91-78	92538.46
5	Тихонов	Архип	Робертовна	6402	182806	8 520 290 3951	90057.99
6	Сорокина	Прохор	Александровна	4258	777294	8 (064) 903-59-31	65558.52
7	Николаев	Ульян	Иосипович	8172	834910	+7 637 931 9995	73135.97
8	Лукин	Зинаида	Власович	2015	569616	8 383 791 42 93	95597.22
9	Голубев	Кондратий	Игоревич	9082	851785	82526195807	65041.56
10	Копылова	Ферапонт	Игнатович	3745	507065	8 (716) 869-60-30	72815.85

Рисунок 47 – Вид главного окна для администратора

id_car	id_model	id_country	price	quantity_in_stock	stock_status	arrival_date
4	46	5	1661308.93	7	да	2024-04-11
8	48	4	3124176.29	10	да	2024-09-23
9	126	3	2583753.81	2	да	2024-07-18
10	125	2	2903674.81	20	да	2024-06-26
11	24	1	1878100.88	17	да	2025-02-06
12	1	2	3374956.42	3	да	2024-08-31
13	25	4	4561935.10	12	да	2024-10-29
14	7	4	1632617.61	17	да	2024-12-22
15	141	2	1281595.27	1	да	2024-05-14

Рисунок 48 – Вид главного окна для экономиста

При нажатии на кнопку «Запросы» выводится новое окно, содержащее все ранее созданные запросы. Для каждого запроса была написана отдельная функция. Каждая из функций выполняет подключении к БД, после чего выполняет запрос, заданный ей запрос. В зависимости от данных, полученных в запросе, функция выводит разные окна. Для запросов с параметрами было создано окно ввода. Код файла, содержащего окно запросов, представлен в Приложении Д. На рисунках 49 – 52 показана работа окна с запросами.

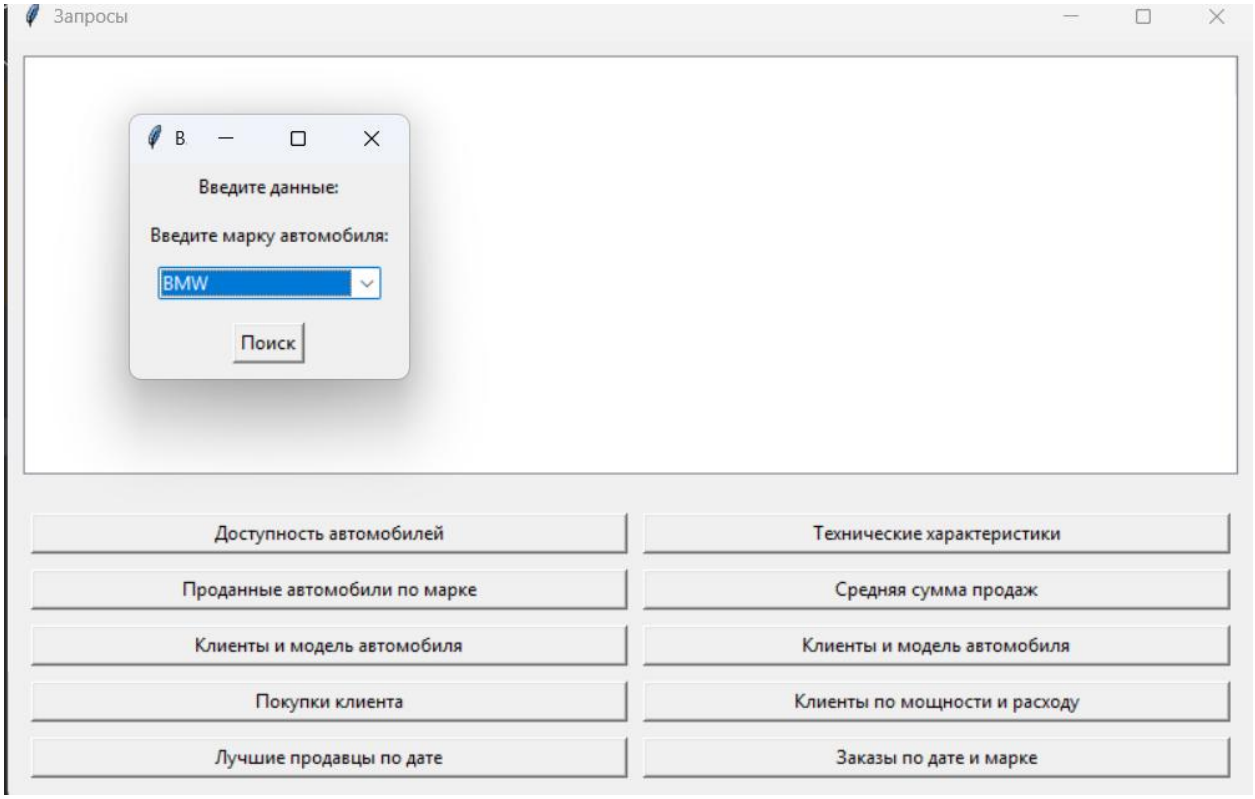


Рисунок 49 – Выпадающий список для выбора марки автомобиля

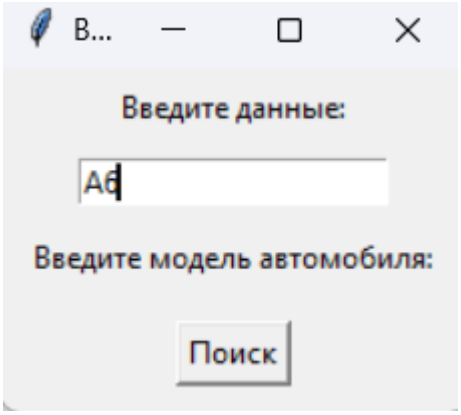
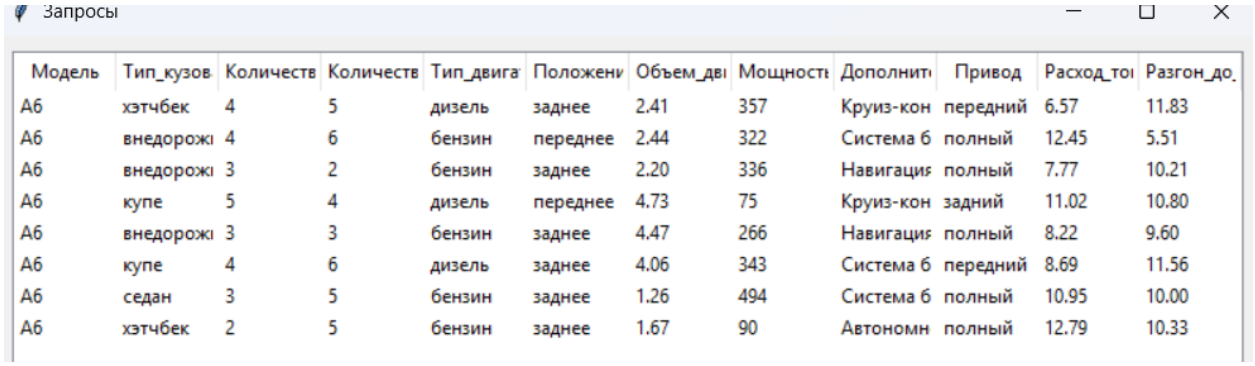


Рисунок 50 – Вводимое значение для выбора модели автомобиля



Модель	Тип_кузов	Количеств	Количеств	Тип_двига	Положени	Объем_дв	Мощност	Дополнит	Привод	Расход_то	Разгон_до
A6	хэтчбек	4	5	дизель	заднее	2.41	357	Круиз-кон	передний	6.57	11.83
A6	внедорожн	4	6	бензин	переднее	2.44	322	Система б	полный	12.45	5.51
A6	внедорожн	3	2	бензин	заднее	2.20	336	Навигация	полный	7.77	10.21
A6	купе	5	4	дизель	переднее	4.73	75	Круиз-кон	задний	11.02	10.80
A6	внедорожн	3	3	бензин	заднее	4.47	266	Навигация	полный	8.22	9.60
A6	купе	4	6	дизель	заднее	4.06	343	Система б	передний	8.69	11.56
A6	седан	3	5	бензин	заднее	1.26	494	Система б	полный	10.95	10.00
A6	хэтчбек	2	5	бензин	заднее	1.67	90	Автономн	полный	12.79	10.33

Рисунок 51 – Результат запроса для выдачи технических характеристик в зависимости от модели автомобиля

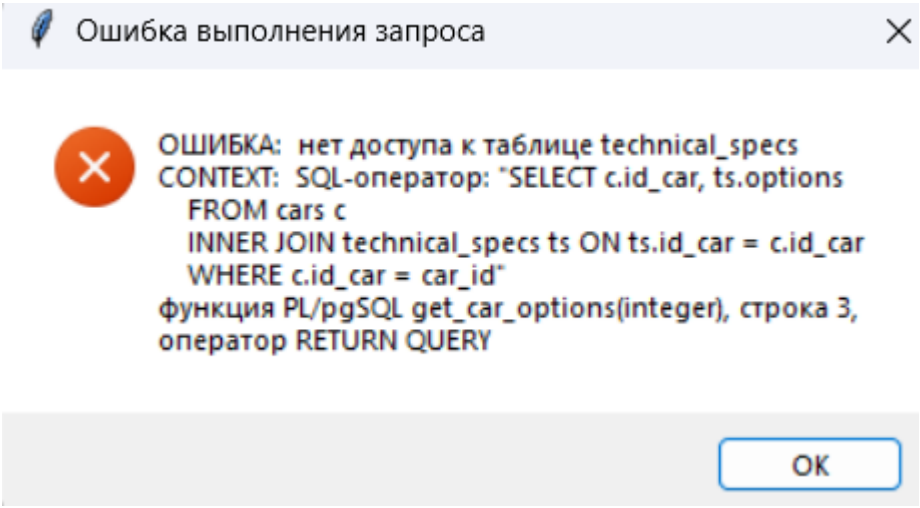


Рисунок 52 – Ограничения доступа

При нажатии в основном окне на кнопку «Графики» открывается окно с выбором графика. В данном окне находится три кнопки. Каждая кнопка выводит отдельное окно со свои графиком, для каждой кнопки была написана своя функция графика, также для получения данных для графика были созданы SQL запросы. Код файла, содержащего окно графиков, представлен в Приложении Ж. На рисунке 53 представлено окно графиков. При нажатии в зависимости от кнопки строятся графики, которые были показаны на рисунках 38 – 43.

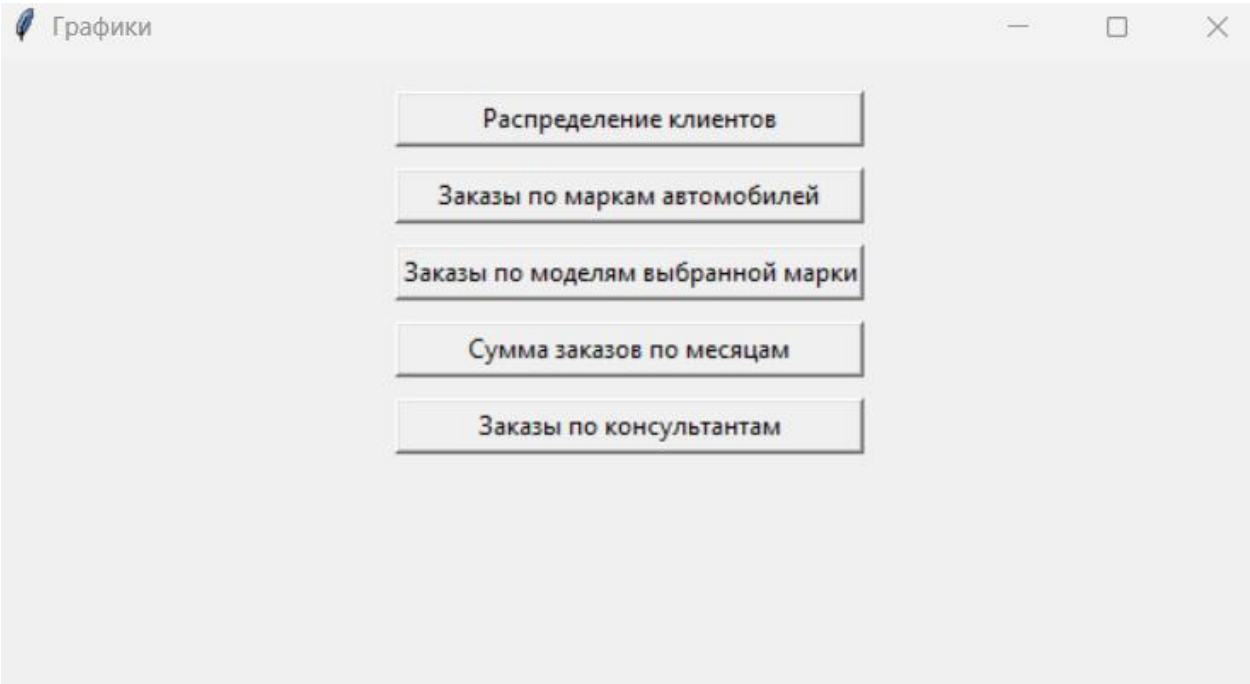


Рисунок 53 – Окно графиков

ЗАКЛЮЧЕНИЕ

В процессе разработки информационной системы для управления финансовыми операциями в автомобильном салоне были достигнуты значительные результаты.

Анализируя предметную область, мы выявили ключевые потребности и требования к системе. Результатом этого анализа стала разработка комплексной базы данных, охватывающей сущности и связи между ними. Эта база данных спроектирована для эффективного управления всеми аспектами деятельности автомобильного салона: от учета товара и заказов до отслеживания финансовых транзакций.

Информационная система была успешно реализована с использованием Postgre в качестве системы управления базами данных и Python для создания пользовательского интерфейса. Важным элементом системы стали триггеры, обеспечивающие проверку целостности данных, и права доступа, обеспечивающие отслеживание деятельности пользователей.

Было разработано 10 запросов, который далее были реализованы в функциях, что позволяет пользователям получать полезную информацию о состоянии автомобильного салона и эффективно управлять ее ресурсами.

Для визуализации работы системы были созданы инструменты, позволяющие строить графики различных типов, что помогает анализировать и прогнозировать работу автомобильного салона.

Результатом работы стала полноценная информационная система, обеспечивающая удобство и эффективность работы с данными автомобильного салона.

В дальнейшем при масштабировании данных, для быстрых принятий бизнес-решений может быть реализована BI-система, которая будет показывать работу автосалона в целом и отдельные показатели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Metanit: Основы проектирования баз данных, URL: https://metanit.com/sql/tutorial/1.1.php (дата обращения 18.04.2025).
2. PostgrePro: Документация к PostgreSQL 17.4, URL: https://postgrespro.ru/docs/postgresql/17/ (дата обращения 18.04.2025).
3. Хабр: Нормализация отношений. Шесть нормальных форм, URL: https://habr.com/ru/articles/254773/ (дата обращения 19.04.2025)
4. Хабр: Триггеры в PostgreSQL: основы, URL: https://habr.com/ru/companies/otus/articles/857396/ (дата обращения 21.04.2025)
5. Selectel : Библиотека Tkinter в Python, URL: https://selectel.ru/blog/tutorials/tkinter-library-in-python/ (дата обращения 21.04.2025)
6. Matplotlib: 3.10.1 documentation, URL: https://matplotlib.org/stable/index.html (дата обращения 21.04.2025)
7. Psycopg: PostgreSQL database adapter for Python, URL: https://www.psycopg.org/docs/ (дата обращения 21.04.2025)
8. Seaborn: statistical data visualization, URL: https://seaborn.pydata.org/ (дата обращения 21.04.2025)
9. Metanit: Руководство по PostgreSQL, URL: https://metanit.com/sql/postgresql/ (дата обращения 21.04.2025)
10. Яндекс Практикум: СУБД PostgreSQL: почему её стоит выбрать для работы с данными и как установить, URL: https://practicum.yandex.ru/blog/chto-takoe-subd-postgresql/ (дата обращения 21.04.2025)
11. Хабр: Чем PostgreSQL лучше других SQL баз данных с открытым исходным кодом. Часть 1, URL: https://habr.com/ru/articles/282764/ (дата обращения 21.04.2025)
12. Selectel: Что такое PostgreSQL, URL: https://selectel.ru/blog/postgresql/ (дата обращения 21.04.2025)
13. Python 3 для начинающих: Исключения в python. Конструкция try - except для обработки исключений, URL: https://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html (дата обращения 21.04.2025)
14. Skillfactory: Инкапсуляция, URL: https://blog.skillfactory.ru/glossary/inkapsulyacziya/ (дата обращения 21.04.2025)

15. Metanit: Руководство по языку программирования Python, URL: https://metanit.com/python/tutorial/ (дата обращения 21.04.2025)
16. Wikipedia: Commit (SQL), URL: https://ru.wikipedia.org/wiki/Commit_(SQL) (дата обращения 21.04.2025)
17. GeekBrains: Атрибуты данных: разбираемся в деталях, URL: https://gb.ru/blog/atributy-dannyh/ (дата обращения 21.04.2025)
18. Skillbox: Пишем десктоп-приложение на Python с помощью Tkinter, URL: https://skillbox.ru/media/code/pishem-desktopprilozhenie-na-python-s-pomoshchyu-tkinter/ (дата обращения 21.04.2025)
19. Metanit: Окно приложения, URL: https://metanit.com/python/tkinter/1.2.php (дата (дата обращения 21.04.2025)
20. Metanit: Введение в виджеты. Tk и ttk, URL: https://metanit.com/python/tkinter/2.1.php (дата обращения 21.04.2025)
21. Metanit: Позиционирование. Pack, URL: https://metanit.com/python/tkinter/2.4.php (дата (дата обращения 21.04.2025)

ПРИЛОЖЕНИЕ А. КОД ФАЙЛА ДЛЯ ЗАПОЛНЕНИЯ БД

```
from faker import Faker
import random

fake = Faker('ru_RU')

car_brands = ['Toyota', 'BMW', 'Lada', 'Hyundai', 'Mercedes',
'Kia', 'Volkswagen', 'Audi', 'Ford', 'Nissan']
car_models = {
'Toyota': ['Camry', 'Corolla', 'Land Cruiser', 'RAV4', 'Hilux'],
'BMW': ['X5', '3 Series', '7 Series', 'X3', '5 Series'],
'Lada': ['Granta', 'Vesta', 'Niva', 'XRay'],
'Hyundai': ['Sonata', 'Elantra', 'Tucson', 'Santa Fe'],
'Mercedes': ['S-Class', 'E-Class', 'A-Class', 'GLA', 'C-Class'],
'Kia': ['Sportage', 'Optima', 'Sorento', 'Rio', 'Ceed'],
'Volkswagen': ['Golf', 'Passat', 'Tiguan', 'Jetta', 'Polo'],
'Audi': ['A3', 'A4', 'Q7', 'Q5', 'A6'],
'Ford': ['Focus', 'Fiesta', 'Mondeo', 'Kuga', 'Explorer'],
'Nissan': ['Qashqai', 'X-Trail', 'Murano', 'Pathfinder', 'Juke']
}
car_countries = ['Япония', 'Германия', 'Россия', 'Южная Корея',
'США', 'Франция', 'Италия', 'Китай', 'Индия', 'Мексика']    #
Определение переменной
body_types = ['седан', 'внедорожник', 'купе', 'пикап', 'хэтчбек']
engine_types = ['бензин', 'дизель', 'электро']
drive_wheels = ['передний', 'задний', 'полный']
options_list = [
'Климат-контроль', 'Система стабилизации', 'Передние и задние
датчики парковки',
'Кожаный салон', 'Круиз-контроль', 'Система безопасности
вождения', 'Навигация',
'Система бесключевого доступа', 'Проекционный экран', 'Подогрев
сидений',
```

```
'Камера заднего вида', 'Автономное вождение', 'Электрические
сиденья', 'Bluetooth',
'Мультимедийная система с поддержкой Apple CarPlay и Android Auto'
]


def insert_block(table, columns, values):
return f"INSERT INTO {table} ({', '.join(columns)}) VALUES\n" +
",\n".join(values) + ";"

def generate_clients(n):
values = []
for _ in range(n):
values.append(f"('{fake.last_name()}',     '{fake.first_name()}',
'{fake.middle_name()}', "
f"'{fake.random_number(digits=4, fix_len=True)}', "
f"'{fake.random_number(digits=6, fix_len=True)}', "
f"'{fake.address().replace(chr(10),           ',           ')}',
'{fake.phone_number()}')")
return insert_block("clients", [
"surname", "name", "patronymic", "passport_series",
"passport_number", "address", "phone_number"
], values)

def generate_brands(n):
values = [f"('{brand}')" for brand in car_brands[:n]]
return insert_block("brands", ["name"], values)

def generate_models(n, brand_range):
values = []
for _ in range(n):
brand_id = random.randint(1, brand_range)
brand = random.choice(car_brands[:brand_range])
model = random.choice(car_models[brand])
```

```
values.append(f"({brand_id}, '{model}')")
return insert_block("models", ["id_brand", "name"], values)

def generate_countries(n):
values = [f"('{country}')" for country in car_countries[:n]]
return insert_block("countries", ["name"], values)

def generate_cars(n, model_range, country_range):
statuses = ['да', 'нет', 'ожидание']
values = []
for _ in range(n):
quantity = random.randint(0, 20)
status = 'да' if quantity > 0 else 'нет'
values.append(f"({random.randint(1,                model_range)},
{random.randint(1, country_range)}, "
f"{round(random.uniform(500000, 5000000), 2)}, {quantity}, "
f"'{status}',                '{fake.date_between(start_date='-1y',
end_date='today')}')")
return insert_block("cars", [
"id_model",    "id_country",    "price",    "quantity_in_stock",
"stock_status", "arrival_date"
], values)

def generate_technical_specs(n, car_range):
values = []
for _ in range(n):
car_id = random.randint(1, car_range)
brand = random.choice(car_brands)
body_type = random.choice(body_types)
doors_count = random.randint(2, 5)
seats_count = random.randint(2, 7)
engine_type = random.choice(engine_types)
engine_position = random.choice(['переднее', 'заднее'])
engine_capacity = round(random.uniform(1.0, 5.0), 2)  # Литры
```

```
engine_power = random.randint(70, 500)  # Лошадиные силы

options = random.sample(options_list, random.randint(2, 5))
options_str = ', '.join(options)

drive_wheels_choice = random.choice(drive_wheels)
fuel_consumption = round(random.uniform(5.0, 15.0), 2)
acceleration = round(random.uniform(5.0, 12.0), 2)

values.append(f"({car_id},       '{body_type}',    {doors_count},
{seats_count}, "
f"'{engine_type}',     '{engine_position}',     {engine_capacity},
{engine_power}, "
f"'{options_str}',  '{drive_wheels_choice}',  {fuel_consumption},
{acceleration})")

return insert_block("technical_specs", [
"id_car",  "body_type", "doors_count", "seats_count",
"engine_type",        "engine_position",        "engine_capacity",
"engine_power",
"options", "drive_wheels", "fuel_consumption", "acceleration"
], values)

def generate_sales_consultants(n):
values = []
for _ in range(n):
values.append(f"('{fake.last_name()}',     '{fake.first_name()}',
'{fake.middle_name()}', "
f"'{fake.random_number(digits=4, fix_len=True)}', "
f"'{fake.random_number(digits=6, fix_len=True)}', "
f"'{fake.phone_number()}', {round(random.uniform(40000, 120000),
2)})")
return insert_block("sales_consultants", [
```

```
"surname", "name", "patronymic", "passport_series",
"passport_number",
"phone_number", "salary"
], values)

def generate_orders(n, client_range, car_range,
consultant_range):
values = []
for _ in range(n):
values.append(f"({random.randint(1, client_range)},
{random.randint(1, car_range)}, "
f"{random.randint(1, consultant_range)},
'{fake.date_between(start_date='-2y', end_date='today')}', "
f"'{random.choice(['да', 'нет'])}', "
f"'{random.choice(['перечисление', 'наличные', 'кредит'])}')")
return insert_block("orders", [
"id_client", "id_car", "id_consultant", "order_date", "delivery",
"payment_type"
], values)

# Пример использования
if __name__ == "__main__":
n = 150
print(generate_clients(n))
print('')
print(generate_brands(10))
print('')
print(generate_models(150, brand_range=10))
print('')
print(generate_countries(10))
print('')
print(generate_cars(150, model_range=150, country_range=10))
print('')
print(generate_technical_specs(150, car_range=150))
```

```
print('')
print(generate_sales_consultants(10))
print('')
print(generate_orders(150,    client_range=40,    car_range=150,
consultant_range=10))
```

ПРИЛОЖЕНИЕ Б. НАПОЛНЕНИЕ ТАБЛИЦ БАЗЫ ДАННЫХ

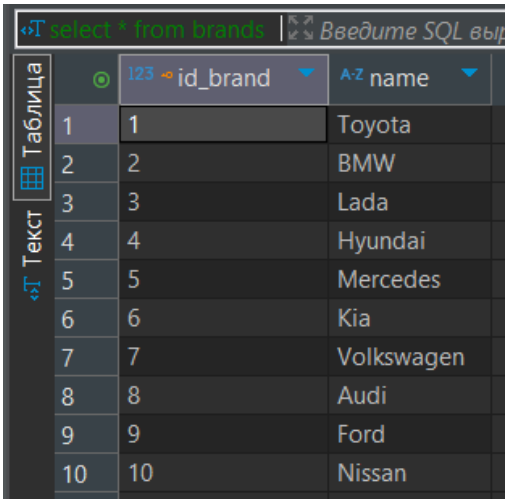


Рисунок 54 – Таблица «brands»

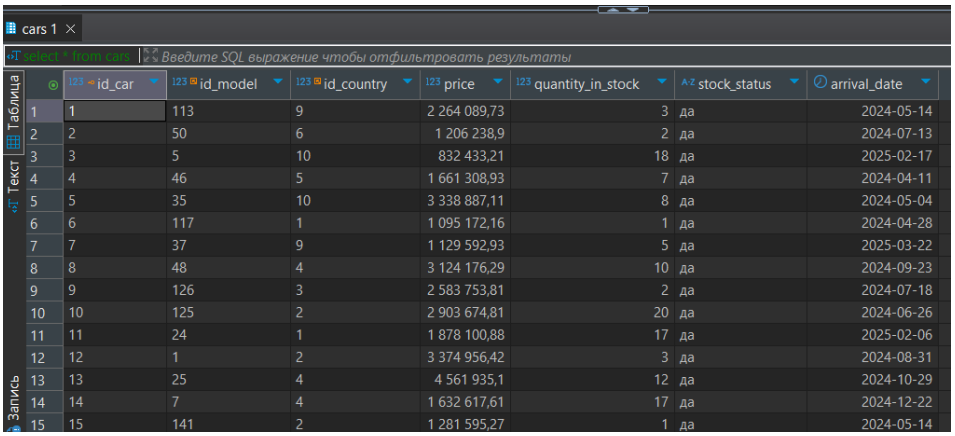


Рисунок 55 – Таблица «cars»

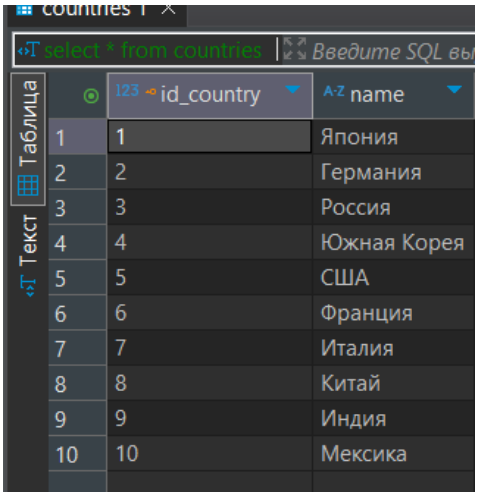


Рисунок 56 – Таблица «countries»

clients 1

select * from clients | Введите SQL выражение чтобы отфильтровать результаты

	id_client	surname	name	patronymic	passport_series	passport_number	address
1	1	Панова	Прокл	Яковлевич	2564	205217	с. Кяхта, ул. Театральная, д. 20 к. 75, 328827
2	2	Кононова	Викторин	Андреевич	5159	235173	п. Белоярский, алл. Чайковского, д. 746 к. 764, 582567
3	3	Игнатова	Ольга	Васильевич	9199	800685	клх Осташков, алл. Депутатская, д. 878 стр. 8/7, 161802
4	4	Лихачева	Дементий	Ааронович	6227	721722	клх Чайковский, алл. Московская, д. 42 к. 2, 195041
5	5	Осипова	Аркадий	Аскольдовна	5302	386891	ст. Ельня, пр. Путевой, д. 6/1 стр. 14, 263886
6	6	Тимофеев	Людмила	Кузьминична	1111	261874	ст. Ачинск, ш. Производственное, д. 8/4, 678143
7	7	Кириллов	Изяслав	Игоревич	4687	137535	п. Кяхта, бул. Геологический, д. 1 к. 97, 549310
8	8	Овчинникова	Светозар	Давидович	1377	731445	с. Новороссийск, ш. Новое, д. 3 стр. 12, 402933
9	9	Юдина	Добромысл	Авдеевич	3699	563854	к. Неплюевка, наб. Васильковая, д. 940, 458949
10	10	Мишина	Боян	Олеговна	9329	969333	ст. Кабанск, ул. Победы, д. 4 к. 103, 216686
11	11	Максимова	Елизавета	Максимовна	1222	942688	д. Таганрог, бул. Седова, д. 7 к. 9/3, 058845
12	12	Веселова	Симон	Геннадиевна	1111	405260	д. Ростов-на-Дону, ул. Хвойная, д. 54, 143631
13	13	Гаврилова	Афиноген	Георгиевна	9521	278305	д. Арсеньев, пер. Высотный, д. 19 стр. 353, 829942
14	14	Васильев	Иванна	Власовна	3299	738924	п. Адлерск, ул. Луговая, д. 7/1, 278805

Рисунок 57 – Таблица «clients»

models 1

select * from models | Введите SQL выражение чтобы

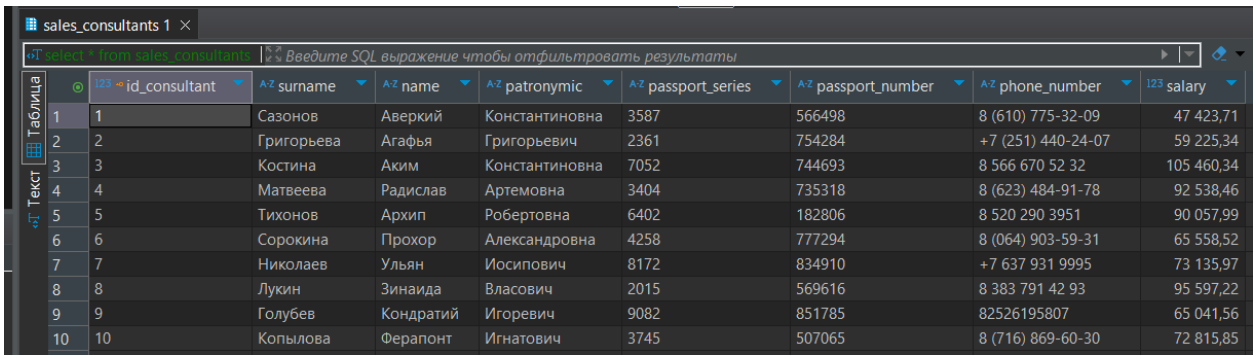
	id_model	id_brand	name
1	1	8	A4
2	2	4	A4
3	3	7	7 Series
4	4	3	Tucson
5	5	2	Optima
6	6	3	Optima
7	7	5	X5
8	8	1	Kuga
9	9	4	Juke
10	10	1	A6
11	11	2	Ceed
12	12	3	E-Class
13	13	6	Mondeo

Рисунок 58 – Таблица «models»

select * from orders | Введите SQL выражение чтобы отфильтровать результаты

	id_order	id_client	id_car	id_consultant	order_date	delivery	payment_type
1	1	11	30	6	2024-12-04	да	кредит
2	2	29	98	5	2024-06-24	да	перечисление
3	3	13	40	6	2023-04-11	да	кредит
4	4	8	27	4	2023-09-20	нет	наличные
5	5	16	28	10	2024-05-08	нет	наличные
6	6	1	130	8	2024-10-02	нет	наличные
7	7	3	109	6	2024-04-05	да	наличные
8	8	1	105	8	2024-06-23	нет	перечисление
9	9	37	23	6	2024-01-23	да	перечисление
10	10	19	60	5	2023-10-05	да	перечисление
11	11	1	67	9	2024-08-01	да	кредит
12	12	6	37	6	2024-01-23	нет	наличные
13	13	37	132	4	2024-09-11	нет	наличные
14	14	25	70	8	2023-08-31	да	кредит

Рисунок 59 – Таблица «orders»



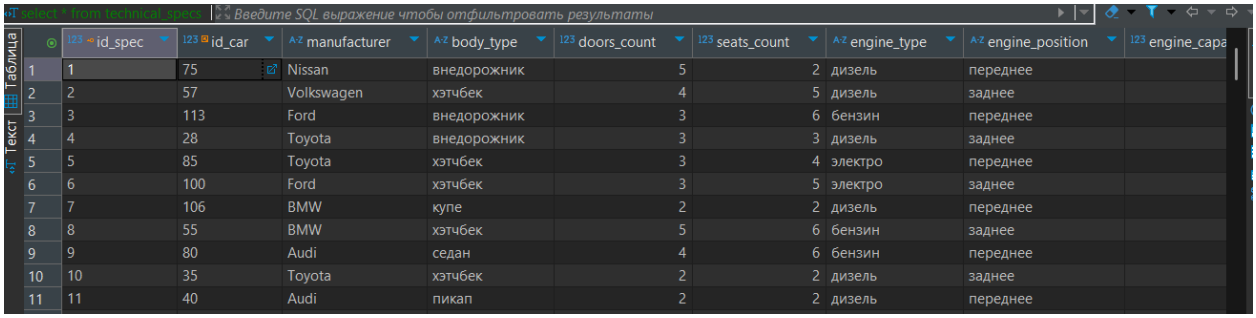
sales_consultants 1

select * from sales_consultants | Введите SQL выражение чтобы отфильтровать результаты

Таблица | Текст

	id_consultant	surname	name	patronymic	passport_series	passport_number	phone_number	salary
1	1	Сазонов	Аверкий	Константиновна	3587	566498	8 (610) 775-32-09	47 423,71
2	2	Григорьева	Агафья	Григорьевич	2361	754284	+7 (251) 440-24-07	59 225,34
3	3	Костина	Аким	Константиновна	7052	744693	8 566 670 52 32	105 460,34
4	4	Матвеева	Радислав	Артемовна	3404	735318	8 (623) 484-91-78	92 538,46
5	5	Тихонов	Архип	Робертовна	6402	182806	8 520 290 3951	90 057,99
6	6	Сорокина	Прохор	Александровна	4258	777294	8 (064) 903-59-31	65 558,52
7	7	Николаев	Ульян	Иосипович	8172	834910	+7 637 931 9995	73 135,97
8	8	Лукин	Зинаида	Власович	2015	569616	8 383 791 42 93	95 597,22
9	9	Голубев	Кондратий	Игоревич	9082	851785	82526195807	65 041,56
10	10	Копылова	Ферапонт	Игнатович	3745	507065	8 (716) 869-60-30	72 815,85

Рисунок 60 – Таблица «sales_consultants»



select * from technical_specs | Введите SQL выражение чтобы отфильтровать результаты

Таблица | Текст

	id_spec	id_car	manufacturer	body_type	doors_count	seats_count	engine_type	engine_position	engine_capa
1	1	75	Nissan	внедорожник	5	2	дизель	переднее	
2	2	57	Volkswagen	хэтчбек	4	5	дизель	заднее	
3	3	113	Ford	внедорожник	3	6	бензин	переднее	
4	4	28	Toyota	внедорожник	3	3	дизель	заднее	
5	5	85	Toyota	хэтчбек	3	4	электро	переднее	
6	6	100	Ford	хэтчбек	3	5	электро	заднее	
7	7	106	BMW	купе	2	2	дизель	переднее	
8	8	55	BMW	хэтчбек	5	6	бензин	заднее	
9	9	80	Audi	седан	4	6	бензин	переднее	
10	10	35	Toyota	хэтчбек	2	2	дизель	заднее	
11	11	40	Audi	пикап	2	2	дизель	переднее	

Рисунок 61 – Таблица «technical_specs»

ПРИЛОЖЕНИЕ В. КОД ФАЙЛА ОКНА ВВОДА

```
import tkinter as tk
from tkinter import messagebox
import psycopg2
from psycopg2 import OperationalError
from main_window import open_main_window  # Убедись, что такой
файл и функция есть

def login():
    username = entry_username.get()
    password = entry_password.get()

    if not username or not password:
        messagebox.showwarning("Внимание", "Пожалуйста, введите
имя пользователя и пароль.")
        return

    try:
        conn = psycopg2.connect(
            database="students",
            user=username,
            password=password,
            host="127.0.0.1",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")
        cursor.close()
        conn.close()

        messagebox.showinfo("Успех", f"Добро пожаловать,
{username}!")

        root.withdraw()  # Скрыть окно авторизации
```

```
        open_main_window(root, username, password)

    except OperationalError as e:
        if "password authentication failed" in str(e):
            messagebox.showerror("Ошибка входа", "Неверное имя
пользователя или пароль.")
        else:
            messagebox.showerror("Ошибка подключения", str(e))

    except Exception as e:
        messagebox.showerror("Ошибка", f"Неверное имя
пользователя или пароль")

# Главное окно входа
root = tk.Tk()
root.title("Вход в систему")

tk.Label(root, text="Имя пользователя:").grid(row=0, column=0,
padx=10, pady=5)
entry_username = tk.Entry(root)
entry_username.grid(row=0, column=1, padx=10, pady=5)

tk.Label(root, text="Пароль:").grid(row=1, column=0, padx=10,
pady=5)
entry_password = tk.Entry(root, show="*")
entry_password.grid(row=1, column=1, padx=10, pady=5)

tk.Button(root, text="Войти", command=login).grid(row=2,
column=0, columnspan=2, pady=10)

root.mainloop()
```

ПРИЛОЖЕНИЕ Г. КОД ФАЙЛА ОСНОВНОГО ОКНА

```
import tkinter as tk
from tkinter import ttk, messagebox
import psycopg2
from querys import open_query_window
from graph import open_graph_window

def open_main_window(login_window, role, password):
    login_window.destroy()  # Закрываем окно входа

    main_window = tk.Tk()
    main_window.title("Главное окно")
    main_window.geometry("1000x700")

    # Верхняя панель с кнопками
    button_frame = tk.Frame(main_window)
    button_frame.pack(side="top", fill="x")

    query_button = tk.Button(button_frame, text="Запросы",
command=lambda: open_query_window(role, password))
    query_button.pack(side="left", padx=5, pady=5)

    chart_button = tk.Button(button_frame, text="Графики",
command=lambda: open_graph_window(role, password))
    chart_button.pack(side="left", padx=5, pady=5)

    # Подключение к базе данных
    try:
        conn = psycopg2.connect(
            database="students",
            user=role,
            password=password,
            host="127.0.0.1",
            port="5432"
```

```
        )
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")

        cursor.execute("""
            SELECT table_name
            FROM information_schema.tables
            WHERE table_schema = 'auto' AND table_type = 'BASE
TABLE';
        """)
        tables = cursor.fetchall()

        custom_table_names = {
            "clients": "Клиенты",
            "orders": "Заказы",
            "cars": "Автомобили",
            "sales_consultants": "Консультанты",
            "brands" : "Марки",
            "models" : "Модели",
            "countries" : "Страны",
            "technical_specs" : "Технические характеристики",
            "logs" : "Журнал изменений в БД"
        }

        notebook = ttk.Notebook(main_window)

        for table in tables:
            table_name = table[0]
            custom_name = custom_table_names.get(table_name,
table_name)

            table_frame = tk.Frame(notebook)
            notebook.add(table_frame, text=custom_name)
```

```
            cursor.execute(f"SELECT * FROM {table_name}")
            rows = cursor.fetchall()

            if not rows:
                tk.Label(table_frame, text="Нет данных в
таблице").pack()
                continue

            tree = ttk.Treeview(table_frame)
            tree["columns"] = [f"col{i}" for i in
range(len(rows[0]))]
            tree["show"] = "headings"

            for i, col in enumerate(cursor.description):
                tree.heading(f"col{i}", text=col[0])
                tree.column(f"col{i}", width=120)

            for row in rows:
                tree.insert("", "end", values=row)

            tree.pack(expand=True, fill="both")

        notebook.pack(expand=True, fill="both")

    except Exception as e:
        messagebox.showerror("Ошибка подключения", str(e))

    main_window.mainloop()
```

ПРИЛОЖЕНИЕ Д. КОД ФАЙЛА ОКНА ЗАПРОСОВ

```
import tkinter as tk
from tkinter import ttk, messagebox
import psycopg2
import datetime

def is_valid_date(date_string):
    try:
        datetime.datetime.strptime(date_string, "%Y-%m-%d")
        return True
    except ValueError:
        return False

def is_float(value):
    try:
        float(value)
        return True
    except ValueError:
        return False

def check_not_empty(*args):
    for value in args:
        if not value.strip():
            return False
    return True




def open_query_window(role, password):
    query_window = tk.Toplevel()
    query_window.title("Запросы")
    query_window.geometry("800x400")  # Уменьшаем высоту окна

    result_frame = tk.Frame(query_window)
```

```
    result_frame.pack(expand=True, fill="both", padx=10,
pady=10)

    tree = ttk.Treeview(result_frame)
    tree.pack(expand=True, fill="both")

    def run_query(sql, description):
        try:
            conn = psycopg2.connect(
                database="students",
                user=role,
                password=password,
                host="127.0.0.1",
                port="5432"
            )
            cursor = conn.cursor()
            cursor.execute("SET search_path TO auto;")
            cursor.execute(sql)
            rows = cursor.fetchall()

            tree.delete(*tree.get_children())
            tree["columns"] = [f"col{i}" for i in
range(len(rows[0]))] if rows else []
            tree["show"] = "headings"

            for i, col in enumerate(cursor.description):
                tree.heading(f"col{i}", text=col[0])
                tree.column(f"col{i}", width=120)

            for row in rows:
                tree.insert("", "end", values=row)

            cursor.close()
            conn.close()
```

```
        except Exception as e:
            messagebox.showerror("Ошибка выполнения запроса",
str(e))

    def open_input_window(query_type):
        input_window = tk.Toplevel(query_window)
        input_window.title("Ввод данных")

        # Поле для ввода модели или даты
        tk.Label(input_window, text="Введите
данные:").pack(padx=10, pady=5)

        if query_type == "get_sold_model_count":
            tk.Label(input_window, text="Введите марку
автомобиля:").pack(padx=10, pady=5)
            list_brands = ["Kia", "BMW", "Audi", "Hyundai",
"Ford", "Mercedes", "Lada", "Nissan", "Toyota", "Volkswagen"]
            brand1 = ttk.Combobox(input_window,
values=list_brands, state="readonly")
            brand1.pack(padx=10, pady=5)
            def on_submit():
                try:
                    brand = brand1.get()
                    if not brand:
                        messagebox.showerror("Ошибка", "Марка не
может быть пустой")
                        return
                    sql_query = f"SELECT * from
get_sold_model_count('{brand}');"
                    run_query(sql_query, f"Количество проданных
автомобилей марки {brand}")
                    input_window.destroy()
                except Exception as e:
```

```
                    messagebox.showerror("Ошибка выполнения
запроса", str(e))


        elif query_type == "get_average_sale_amount":
            start_date_entry = tk.Entry(input_window)
            end_date_entry = tk.Entry(input_window)
            start_date_entry.pack(padx=10, pady=5)
            end_date_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите начальную дату
(YYYY-MM-DD):").pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите конечную дату
(YYYY-MM-DD):").pack(padx=10, pady=5)

            def on_submit():
                try:
                    start_date = start_date_entry.get()
                    end_date = end_date_entry.get()
                    if not start_date or not end_date:
                        messagebox.showerror("Ошибка", "Дата не
может быть пустой")
                        return
                    sql_query = f"SELECT * from
get_average_sale_amount('{start_date}', '{end_date}');"
                    run_query(sql_query, f"Средняя сумма продаж
с {start_date} по {end_date}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка выполнения
запроса", str(e))


        elif query_type == "get_clients_by_payment":
            payment = tk.Entry(input_window)
```

```
            payment.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите тип оплаты
(кредит, наличные, перечисление):").pack(padx=10, pady=5)

            def on_submit():
                try:
                    payment1 = payment.get()
                    if not payment1:
                        messagebox.showerror("Ошибка", "Введеное
неверное значение")
                        return
                    sql_query = f"SELECT * from
get_clients_by_payment(lower('{payment1}'));"
                    run_query(sql_query, f"Список клиентов и
автомобилей по виду оплаты {payment1}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка выполнения
запроса", str(e))


        elif query_type == "get_car_options":
            car_id = tk.Entry(input_window)
            car_id.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите id автомобиля
(таблица Автомобили):").pack(padx=10, pady=5)

            def on_submit():
                try:
                    car_id1 = car_id.get()
                    if not car_id1:
                        messagebox.showerror("Ошибка", "Введено
неверное значение")
                        return
```

```
                    sql_query = f"SELECT
get_car_options({car_id1});"
                    run_query(sql_query, f"Информация об опциях
автомобиля {car_id1}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка выполнения
запроса", str(e))


        elif query_type == "get_car_availability":
            # Добавляем поле для ввода марки и модели для
доступности
            brand_entry = tk.Entry(input_window)
            model_entry = tk.Entry(input_window)
            brand_entry.pack(padx=10, pady=5)
            model_entry.pack(padx=10, pady=5)

            tk.Label(input_window, text="Введите марку
автомобиля:").pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите модель
автомобиля:").pack(padx=10, pady=5)

            def on_submit():
                try:
                    brand = brand_entry.get()
                    model = model_entry.get()
                    if not brand or not model:
                        messagebox.showerror("Ошибка", "Марка и
модель не могут быть пустыми")
                        return
                    sql_query = f"SELECT * FROM
get_car_availability(lower('{brand}'), lower('{model}'));"
                    run_query(sql_query, f"Доступность {brand}
```

```
{model}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка выполнения
запроса", str(e))


        elif query_type == "get_technical_specs_by_model":
            model_entry = tk.Entry(input_window)
            model_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите модель
автомобиля:").pack(padx=10, pady=5)
            def on_submit():
                model = model_entry.get()
                if not model:
                    messagebox.showerror("Ошибка", "Модель не
может быть пустой")
                    return
                try:
                    sql_query = f"SELECT * FROM
get_technical_specs_by_model('{model}');"
                    run_query(sql_query, f"Технические
характеристики для {model}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка запроса",
str(e))


        elif query_type == "get_client_orders":
            client_entry = tk.Entry(input_window)
            client_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите ID
клиента:").pack(padx=10, pady=5)
```

```
            def on_submit():
                client_id = client_entry.get()
                if not client_id.isdigit():
                    messagebox.showerror("Ошибка", "ID клиента
должен быть числом")
                    return
                try:
                    sql_query = f"SELECT * FROM
get_client_orders({client_id});"
                    run_query(sql_query, f"Покупки клиента
{client_id}")
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка запроса",
str(e))

        elif query_type ==
"get_clients_by_power_and_consumption":
            power_entry = tk.Entry(input_window)
            consumption_entry = tk.Entry(input_window)
            power_entry.pack(padx=10, pady=5)
            consumption_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Минимальная мощность
двигателя:").pack(padx=10, pady=5)
            tk.Label(input_window, text="Максимальный расход
топлива:").pack(padx=10, pady=5)
            def on_submit():
                min_power = power_entry.get()
                max_consumption = consumption_entry.get()
                try:
                    float(min_power)
                    float(max_consumption)
                    sql_query = f"SELECT * FROM
get_clients_by_power_and_consumption({min_power},
```

```
{max_consumption});"
                    run_query(sql_query, f"Клиенты с мощностью >
{min_power} и расходом < {max_consumption}")
                    input_window.destroy()
                except ValueError:
                    messagebox.showerror("Ошибка", "Введите
числовые значения")
                except Exception as e:
                    messagebox.showerror("Ошибка запроса",
str(e))


        elif query_type == "get_top_sellers":
            start_date_entry = tk.Entry(input_window)
            end_date_entry = tk.Entry(input_window)
            start_date_entry.pack(padx=10, pady=5)
            end_date_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите начальную дату
(YYYY-MM-DD):").pack(padx=10, pady=5)
            tk.Label(input_window, text="Введите конечную дату
(YYYY-MM-DD):").pack(padx=10, pady=5)

            def on_submit():
                start_date = start_date_entry.get()
                end_date = end_date_entry.get()
                if not start_date or not end_date:
                    messagebox.showerror("Ошибка", "Обе даты
должны быть заполнены")
                    return
                try:
                    sql_query = f"SELECT * FROM
get_top_sellers('{start_date}', '{end_date}');"
                    run_query(sql_query, f"Лучшие продавцы с
{start_date} по {end_date}")
```

```
                    input_window.destroy()
                except Exception as e:
                    messagebox.showerror("Ошибка запроса",
str(e))


        elif query_type == "get_orders_by_period_and_brand":
            start_date_entry = tk.Entry(input_window)
            end_date_entry = tk.Entry(input_window)
            brand_entry = tk.Entry(input_window)
            start_date_entry.pack(padx=10, pady=5)
            end_date_entry.pack(padx=10, pady=5)
            brand_entry.pack(padx=10, pady=5)
            tk.Label(input_window, text="Начальная дата (YYYY-
MM-DD):").pack(padx=10, pady=5)
            tk.Label(input_window, text="Конечная дата (YYYY-MM-
DD):").pack(padx=10, pady=5)
            tk.Label(input_window, text="Марка
автомобиля:").pack(padx=10, pady=5)

            def on_submit():
                start_date = start_date_entry.get()
                end_date = end_date_entry.get()
                brand = brand_entry.get()
                if not brand or not start_date or not end_date:
                    messagebox.showerror("Ошибка", "Все поля
должны быть заполнены")
                    return
                try:

                    sql_query = f"SELECT * FROM
get_orders_by_period_and_brand('{start_date}', '{end_date}',
'{brand}');"
                    run_query(sql_query, f"Заказы по марке
```

```
{brand} с {start_date} по {end_date}")
                    input_window.destroy()

                except Exception as e:
                    messagebox.showerror("Ошибка запроса",
str(e))

        # Кнопка для отправки данных
        submit_button = tk.Button(input_window, text="Поиск",
command=on_submit)
        submit_button.pack(pady=10)

    # Кнопки запросов
    button_frame = tk.Frame(query_window)
    button_frame.pack(fill="x", padx=10, pady=10)

    # Расположение кнопок в два ряда
    btn1 = tk.Button(button_frame, text="Доступность
автомобилей",
                     command=lambda:
open_input_window("get_car_availability"))
    btn2 = tk.Button(button_frame, text="Технические
характеристики",
                     command=lambda:
open_input_window("get_technical_specs_by_model"))
    btn3 = tk.Button(button_frame, text="Проданные автомобили по
марке",
                     command=lambda:
open_input_window("get_sold_model_count"))
    btn4 = tk.Button(button_frame, text="Средняя сумма продаж",
                     command=lambda:
open_input_window("get_average_sale_amount"))
    btn5 = tk.Button(button_frame, text="Клиенты и модель
автомобиля",
```

```
                    command=lambda:
open_input_window("get_clients_by_payment"))
    btn6 = tk.Button(button_frame, text="Клиенты и модель
автомобиля",
                    command=lambda:
open_input_window("get_car_options"))
    btn7 = tk.Button(button_frame, text="Покупки клиента",
                    command=lambda:
open_input_window("get_client_orders"))
    btn8 = tk.Button(button_frame, text="Клиенты по мощности и
расходу",
                    command=lambda:
open_input_window("get_clients_by_power_and_consumption"))
    btn9 = tk.Button(button_frame, text="Лучшие продавцы по
дате",
                    command=lambda:
open_input_window("get_top_sellers"))
    btn10 = tk.Button(button_frame, text="Заказы по дате и
марке",
                     command=lambda:
open_input_window("get_orders_by_period_and_brand"))

    btn1.grid(row=0, column=0, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn2.grid(row=0, column=1, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn3.grid(row=1, column=0, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn4.grid(row=1, column=1, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn5.grid(row=2, column=0, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn6.grid(row=2, column=1, padx=5, pady=5, sticky="ew",
ipadx=20)
```

```
    btn7.grid(row=3, column=0, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn8.grid(row=3, column=1, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn9.grid(row=4, column=0, padx=5, pady=5, sticky="ew",
ipadx=20)
    btn10.grid(row=4, column=1, padx=5, pady=5, sticky="ew",
ipadx=20)

    # Устанавливаем одинаковую ширину для кнопок
    button_frame.grid_columnconfigure(0, weight=1)
    button_frame.grid_columnconfigure(1, weight=1)
```

ПРИЛОЖЕНИЕ Е. КОД ФАЙЛА ОКНА С ГРАФИКАМИ

```
import tkinter as tk
from tkinter import messagebox, simpledialog
import psycopg2
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wn
wn.filterwarnings('ignore')

def vis1(role, password):
    try:
        print(role)
        conn = psycopg2.connect(
            database="students",
            user=role,
            password=password,
            host="127.0.0.1",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")

        query = "SELECT *, EXTRACT(YEAR FROM AGE(CURRENT_DATE,
birth_date))::int AS age FROM clients;"
        df = pd.read_sql_query(query, conn)
        conn.close()

        if 'gender' not in df.columns:
            import numpy as np
            np.random.seed(42)
            df['gender'] = np.random.choice(['М', 'Ж'],
size=len(df))
```

```
        bins = [0, 19, 29, 39, 49, 59, 69, 79, 89, 100]
        labels = ['<20', '20-29', '30-39', '40-49', '50-59',
'60-69', '70-79', '80-89', '90+']
        df['age_group'] = pd.cut(df['age'], bins=bins,
labels=labels, right=True)

        grouped = df.groupby(['age_group',
'gender']).size().unstack(fill_value=0)

        grouped.plot(kind='bar', figsize=(12, 6), width=0.8,
color=['cornflowerblue', 'lightpink'])

        plt.title('Распределение клиентов по возрастным группам
и полу')
        plt.xlabel('Возрастная группа')
        plt.ylabel('Количество клиентов')
        plt.xticks(rotation=0)
        plt.grid(axis='y', linestyle='--', alpha=0.5)
        plt.legend(title='Пол')
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def vis2(role, password):
    try:
        conn = psycopg2.connect(
            database="students",
            user=role,
            password=password,
            host="127.0.0.1",
            port="5432"
        )
```

```
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")

        query = """
        SELECT b.name, COUNT(*) as total_orders
        FROM orders o
        JOIN cars c on o.id_car = c.id_car
        JOIN models m on c.id_model = m.id_model
        JOIN brands b on b.id_brand = m.id_brand
        GROUP BY b.name
        ORDER BY total_orders DESC;
        """
        df_orders = pd.read_sql_query(query, conn)
        conn.close()

        plt.figure(figsize=(12, 6))
        sns.barplot(data=df_orders, x='name', y='total_orders',
palette='crest')
        plt.title("Количество заказов по маркам автомобилей",
fontsize=14)
        plt.xlabel("Марка автомобиля")
        plt.ylabel("Количество заказов")
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def vis3(role, password):
    # Запрашиваем марку автомобиля у пользователя
    marka = simpledialog.askstring("Введите марку", "Введите
марку автомобиля:")
```

```
    if not marka:
        messagebox.showwarning("Ошибка", "Марка не была
введена!")
        return

    try:
        conn = psycopg2.connect(
            database="students",
            user=role,
            password=password,
            host="127.0.0.1",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")

        # SQL-запрос для получения данных по марке
        query = f"""
        SELECT m.name, COUNT(*) as total_orders
        FROM orders o
        JOIN cars c on o.id_car = c.id_car
        JOIN models m on c.id_model = m.id_model
        JOIN brands b on b.id_brand = m.id_brand
        WHERE lower(b.name) = lower('{marka}')
        GROUP BY m.name
        ORDER BY total_orders DESC;
        """

        # Выполняем запрос и загружаем данные в DataFrame
        df_orders = pd.read_sql_query(query, conn)

        # Закрываем соединение с базой данных
        conn.close()
```

```
        if df_orders.empty:
            messagebox.showinfo("Информация", f"Нет данных по
марке '{marka}'.")
            return

        # Строим график
        plt.figure(figsize=(12, 6))
        sns.barplot(data=df_orders, x='name', y='total_orders',
palette='crest')
        plt.title(f"Модели автомобилей марки {marka}",
fontsize=14)
        plt.xlabel("Модели")
        plt.ylabel("Количество заказов")
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def vis4(role, password):
    # Запрашиваем год у пользователя
    year = simpledialog.askinteger("Введите год", "Введите год
для анализа (2000-2025):")

    if not year or not (2000 <= year <= 2025):
        messagebox.showwarning("Ошибка", "Введите корректный год
(2000-2025).")
        return

    try:
        conn = psycopg2.connect(
            database="students",
            user=role,
```

```
            password=password,
            host="127.0.0.1",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("SET search_path TO auto;")

        # SQL-запрос для получения данных по сумме заказов по
месяцам
        query = f"""
        SELECT TO_CHAR(order_date, 'TMMonth') AS month,
SUM(c.price) AS revenue
        FROM orders o
        INNER JOIN cars c ON c.id_car = o.id_car
        WHERE EXTRACT(YEAR FROM order_date) = {year}
        GROUP BY month
        ORDER BY month;
        """

        # Выполняем запрос и загружаем данные в DataFrame
        df_revenue = pd.read_sql_query(query, conn)

        # Закрываем соединение с базой данных
        conn.close()

        if df_revenue.empty:
            messagebox.showinfo("Информация", f"Нет данных для
года {year}.")
            return

        # Строим график
        plt.figure(figsize=(12, 6))
        sns.lineplot(data=df_revenue, x='month', y='revenue',
marker='o', linewidth=2, color='green')
```

```
        plt.title(f"Общая сумма заказов по месяцам за {year}
год", fontsize=14)
        plt.xlabel("Месяц")
        plt.ylabel("Сумма заказов, ₽")
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def vis5(role, password):
    # Запрашиваем параметр (1 или 2) для выбора типа анализа
    option = simpledialog.askinteger("Выберите опцию", "Введите
1 для анализа по сумме заказов, 2 для анализа по количеству
заказов:")

    if option == 1:
        query = """
        SELECT
            sc.surname || ' ' || sc.name || ' ' || sc.patronymic
AS full_name,
            SUM(c.price) AS total_sales
        FROM auto.sales_consultants sc
        JOIN auto.orders o ON sc.id_consultant = o.id_consultant
        JOIN auto.cars c ON o.id_car = c.id_car
        WHERE EXTRACT(YEAR FROM o.order_date) = 2025
        GROUP BY full_name
        ORDER BY total_sales DESC;
        """
        title = "Сумма заказов по консультантам (2025 год)"
        value_column = "total_sales"
```

```
    elif option == 2:
        query = """
        SELECT
            sc.surname || ' ' || sc.name || ' ' || sc.patronymic
AS full_name,
            COUNT(o.id_order) AS order_count
        FROM auto.sales_consultants sc
        JOIN auto.orders o ON sc.id_consultant = o.id_consultant
        WHERE EXTRACT(YEAR FROM o.order_date) = 2025
        GROUP BY full_name
        ORDER BY order_count DESC;
        """
        title = "Количество заказов по консультантам (2025 год)"
        value_column = "order_count"

    else:
        messagebox.showwarning("Ошибка", "Неверный параметр.
Введите 1 или 2.")
        return

    try:
        conn = psycopg2.connect(
            database="students",
            user=role,
            password=password,
            host="127.0.0.1",
            port="5432"
        )

        # Выполняем запрос и загружаем данные в DataFrame
        df = pd.read_sql_query(query, conn)

        # Закрываем соединение с базой данных
        conn.close()
```

```
        if df.empty:
            messagebox.showinfo("Информация", "Нет данных для
выбранного параметра.")
            return

        # Строим график
        plt.figure(figsize=(10, 6))
        sns.barplot(data=df, x=value_column, y="full_name",
palette="viridis")
        plt.title(title)
        plt.xlabel("Значение")
        plt.ylabel("Консультант")
        plt.tight_layout()
        plt.show()

    except Exception as e:
        messagebox.showerror("Ошибка", str(e))


def open_graph_window(role, password):
    graph_window = tk.Toplevel()
    graph_window.title("Графики")
    graph_window.geometry("600x300")

    button_frame = tk.Frame(graph_window)
    button_frame.pack(padx=10, pady=10, fill="both",
expand=True)

    button_vis1 = tk.Button(button_frame, text="Распределение
клиентов", command=lambda: vis1(role, password), width=30)
    button_vis1.pack(pady=5)

    button_vis2 = tk.Button(button_frame, text="Заказы по маркам
```

```
автомобилей", command=lambda: vis2(role, password), width=30)
    button_vis2.pack(pady=5)

    button_vis3 = tk.Button(button_frame, text="Заказы по
моделям выбранной марки", command=lambda: vis3(role, password),
width=30)
    button_vis3.pack(pady=5)

    button_vis4 = tk.Button(button_frame, text="Сумма заказов по
месяцам", command=lambda: vis4(role, password), width=30)
    button_vis4.pack(pady=5)

    button_vis5 = tk.Button(button_frame, text="Заказы по
консультантам", command=lambda: vis5(role, password), width=30)
    button_vis5.pack(pady=5)

    graph_window.mainloop()
```