**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ(МИИТ)**

**Институт управления и цифровых технологий**

Кафедра «Кафедра «Вычислительные системы, сети и информационная безопасность»

**Отчет по КУРСОВой РАБОТе**

**ПО Дисциплине «БАЗЫ ДАННЫХ»**

**НА ТЕМУ:** **«Расписание пригородных поездов»**

Выполнил:   
студент группы УИБ-311

Москаленко Виталий Александрович

Проверили:

Доцент Давыдовский М. А.

Ст.преп. Новиков А.И.

**МОСКВА 2024**

**СОДЕРЖАНИЕ**

[1. ЗАДАНИЕ 3](#_Toc183308262)

[2. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc183308263)

[3. ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИЯХ IDEF1X 7](#_Toc183308264)

[4. ОПИСАНИЕ СХЕМЫ БАЗЫ ДАННЫХ НА ЯЗЫКЕ SQL 12](#_Toc183308265)

[5. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ЗАПРОСОВ НА ЯЗЫКЕ SQL 17](#_Toc183308266)

[6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПО РАБОТЕ С ПРИЛОЖЕНИЕМ 56](#_Toc183308267)

[7. ИСХОДНЫЙ КОД ПРОГРАММЫ 60](#_Toc183308268)

# **ЗАДАНИЕ**

Разработать приложение на одном из языков высокого уровня (C++, C#, Java, Python и др), которое позволяет работать с базой данных.

Приложение должно иметь визуальный интерфейс, позволяющий вводить, корректировать и удалять информацию из таблиц базы данных. Приложение может быть как локальным, так и веб-приложением.

Приложение должно выполнять поисковые запросы, которые прописаны в задании.

Схема базы данных должна быть спроектирована по алгоритмам проектирования реляционной модели базы данных (алгоритмы: Фэджина, Делобеля-Кейси, Бернштейна).

Для реализации проекта выбирается схема реляционной базы данных, полученная по одному из алгоритмов проектирования.

При выполнении курсовой работы используется такая же система управления базами данных как при выполнении лабораторных работ.

# **ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

**19. Расписание движения пригородных поездов.**

В терминах ER-модели описать предметную область информационно-

справочной системы сведений о расписании движения пригородных поездов

России.

Пригородное расписание вводится ежегодно в последнее воскресенье

мая. Регулярно (не позднее чем за 10 дней) в расписание работниками подразделениями ОАО РЖД вносятся изменения, которые обусловлены проводимыми ремонтами объектов инфраструктуры ж.д. транспорта (пути на станциях и перегонах, платформы и т.д.) и другими причинами. Пригородные поезда совершают рейсы (поездки) по конкретным ниткам графика движения с шеститысячной или семитысячной нумерацией от заданной станции отправления (начальная) до станции назначения (конечная), представляющие собой название маршрута. Для каждого рейса поезда кроме номера поезда, совпадающей с номером нитки графика, имеется общий маршрут, характеризующийся станцией отправления и станцией назначения.

Для всех промежуточных станций маршрута расписания (даже тех, где нет остановки для посадки/высадки пассажиров), а также начального и конечного пункта задаются времена прибытия и отправления с точностью до секунд. Так как очень часто маршрут расписания изменяется (отменяются или назначаются новые остановки, например, из-за перехода с летнего на зимнее расписание), то предусматривается для каждого рейса несколько вариантов расписания, каждый из которых отличается временами прохождения остановочных пунктов и/или набором остановочных пунктов (из-за удлинения или укорочения маршрутов).

Каждый из остановочных пунктов варианта маршрута расписания принадлежит отраслевому классификатору раздельных пунктов (станций) ЦНСИ ОАО РЖД. Каждая из станций принадлежит свой железной дороге из справочника дороги. Имеют список остановочных пунктов маршрут расписания. Хождение поездов в конкретные даты определяется календарем, где для каждого поезда и даты годового сезона (от последнего воскресенья мая текущего года включительно до последнего воскресенья мая следующего года исключительно) задается ненулевым вариантом расписания. Каждый рейс обслуживается своей компанией перевозчиком, например, Аэроэкспрес, ЦППК и другие.

Система должна содержать информацию о:

* номерах ниток и станциях отправления/назначения, категории (общий/скоростной), перевозчике, регулярность хождения (по будням/выходным/ежедневно), год графика всех поездов
* станциях отправления/назначения и год графика всех вариантов расписания
* номерах ниток, времени проследования, пройденном километраже по прибытию/отправления, признаком разрешения посадки/высадки 20 для пассажиров для каждого из остановочного пункта варианта расписания
* календаре хождения поезда в заданную дату
* станциях (коды, название, тип, признак действующей станции, принадлежность дороги)
* железных дорогах РФ, стран СНГ и Балтии (код, полные и
* сокращенные названия, принадлежность государству)
* перевозчиках (код, полные и сокращенные названия,
* принадлежность государству)

Система должна эффективно выдавать ответы на следующие

запросы:

1. Выдать список наименований всех платформ (тип раздельного пункта - 7) Москвской ж.д. (код дороги - 17)
2. Выдать список наименований всех раздельных пунктов заданного варианта маршрута (по коду варианта)
3. Выдать повторяющиеся имена раздельных пунктов на Московской дороге
4. Выдать список всех поездов заданного направления (четность номера поезда) и начальной станции отправления поезда
5. Выдать список всех поездов, останавливающихся для посадки/высадки пассажиров на заданной станции A
6. Выдать список всех поездов, делающих техническую остановку (без посадки/высадки пассажиров) на заданной станции A
7. Выдать список поездов, на которых можно проехать от станции А до станции B
8. Выдать список поездов, прибывающих на станцию А, в заданном направлении в промежутке времени [t1, t2] в заданную дату
9. Выдать поезда, которые следуют со всеми остановками для посадки/высадки
10. Выдать расписание поездов для заданного маршрута (по станции отправления и назначения) в табличной форме, как это делается стандартном расписании (слева – станции, вверху – поезда, элемент таблицы – время отправления со станции)
11. Поиск оптимального по времени поезда, следующего от станции станции В в заданном промежутке времени [t1, t2]
12. Изменить тип станции в классификаторе станций
13. Добавить нового перевозчика
14. Удалить из БД расписания всю устаревшую информацию

# **ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИЯХ IDEF1X**

Для создания реляционной базы в СУБД PostgreSQL была использована ER-модель в нотации IDEF1X, изображённая на рисунке 1.

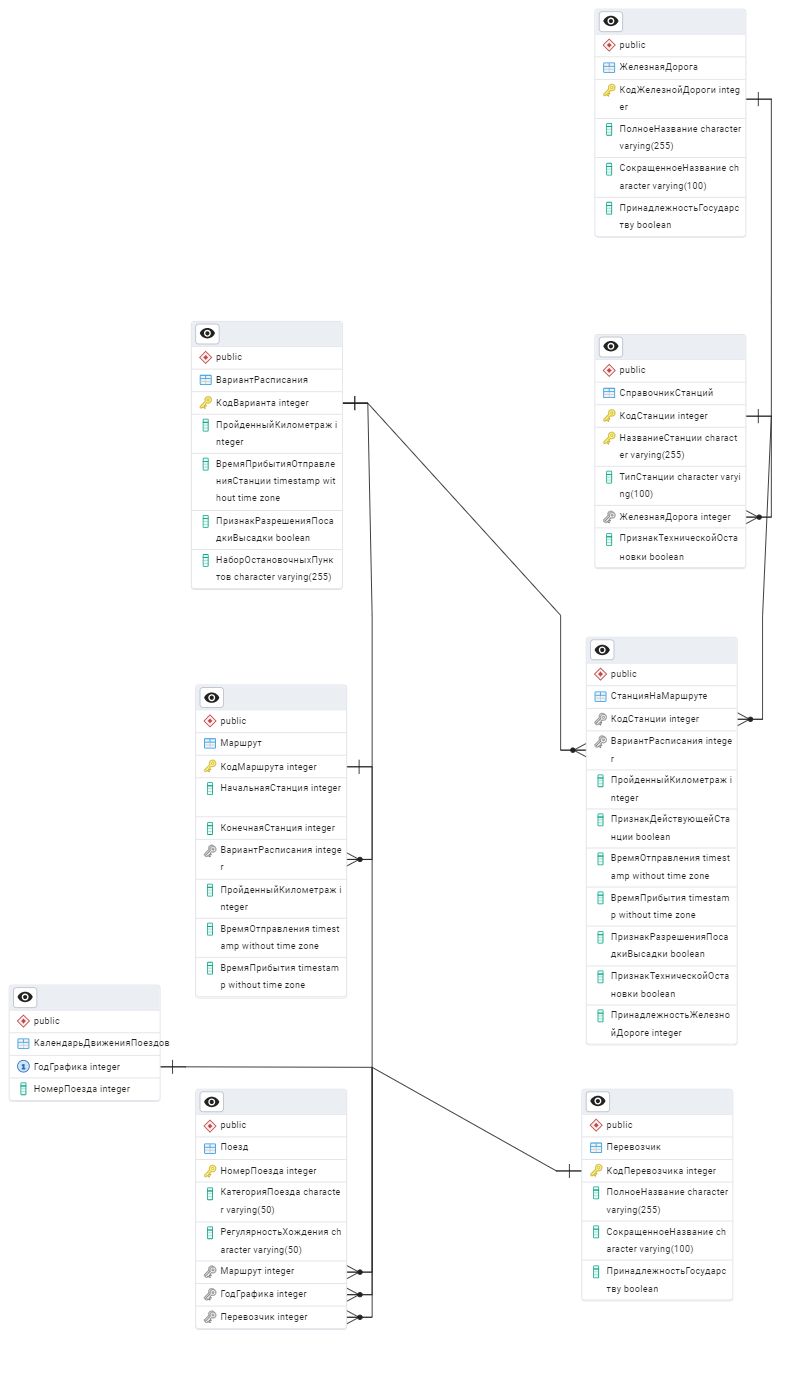


Рисунок 1 – ER-модель в нотации IDEF1X

**Описание сущностей**

**ВариантРасписания**

Сущность описывает варианты расписания движения поездов. Основные атрибуты:

* КодВарианта (integer): Уникальный идентификатор варианта расписания. Первичный ключ.
* ПройденныйКилометраж (integer): Указывает расстояние, пройденное поездом в рамках данного варианта расписания.
* ВремяПрибытияОтправленияСтанции (timestamp): Время прибытия или отправления на станции.
* ПризнакРазрешенияПосадкиВысадки (boolean): Указывает, разрешена ли посадка/высадка пассажиров.
* НаборОстановочныхПунктов (character varying): Список остановочных пунктов в рамках расписания.

**ЖелезнаяДорога**

Сущность хранит информацию о железных дорогах. Основные атрибуты:

* КодЖелезнойДороги (integer): Уникальный идентификатор железной дороги. Первичный ключ.
* ПолноеНазвание (character varying): Полное название железной дороги.
* СокращенноеНазвание (character varying): Сокращённое название железной дороги.
* ПринадлежностьГосударству (boolean): Признак государственной принадлежности.

**КалендарьДвиженияПоездов**

Сущность хранит информацию о годах графиков движения поездов. Основные атрибуты:

* ГодГрафика (integer): Год действия графика. Уникальный ключ.
* НомерПоезда (integer): Номер поезда, связанный с графиком.

**Маршрут**

Сущность описывает маршруты поездов. Основные атрибуты:

* КодМаршрута (integer): Уникальный идентификатор маршрута. Первичный ключ.
* НачальнаяСтанция (integer): Код начальной станции.
* КонечнаяСтанция (integer): Код конечной станции.
* ВариантРасписания (integer): Ссылка на вариант расписания (внешний ключ к ВариантРасписания).
* ПройденныйКилометраж (integer): Расстояние, пройденное по маршруту.
* ВремяОтправления (timestamp): Время отправления с начальной станции.
* ВремяПрибытия (timestamp): Время прибытия на конечную станцию.

**Перевозчик**

Сущность хранит информацию о транспортных компаниях. Основные атрибуты:

* КодПеревозчика (integer): Уникальный идентификатор перевозчика. Первичный ключ.
* ПолноеНазвание (character varying): Полное название перевозчика.
* СокращенноеНазвание (character varying): Сокращённое название перевозчика.
* ПринадлежностьГосударству (boolean): Признак государственной принадлежности перевозчика.

**Поезд**

Сущность описывает поезда. Основные атрибуты:

* НомерПоезда (integer): Уникальный номер поезда. Первичный ключ.
* КатегорияПоезда (character varying): Категория поезда (например, пассажирский, грузовой).
* РегулярностьХождения (character varying): Характеристика регулярности движения поезда.
* Маршрут (integer): Ссылка на маршрут (внешний ключ к Маршрут).
* ГодГрафика (integer): Ссылка на год графика движения (внешний ключ к КалендарьДвиженияПоездов).
* Перевозчик (integer): Ссылка на перевозчика (внешний ключ к Перевозчик).

**СправочникСтанций**

Сущность содержит информацию о станциях. Основные атрибуты:

* КодСтанции (integer): Уникальный идентификатор станции. Уникальный и первичный ключ.
* НазваниеСтанции (character varying): Название станции. Обязательный атрибут.
* ТипСтанции (character varying): Тип станции (например, узловая, промежуточная).
* ЖелезнаяДорога (integer): Ссылка на железную дорогу (внешний ключ к ЖелезнаяДорога).
* ПризнакТехническойОстановки (boolean): Указывает, является ли станция технической.

**СтанцияНаМаршруте**

Сущность описывает станции, входящие в маршрут. Основные атрибуты:

* КодСтанции (integer): Ссылка на станцию (внешний ключ к СправочникСтанций).
* ВариантРасписания (integer): Ссылка на вариант расписания (внешний ключ к ВариантРасписания).
* ПройденныйКилометраж (integer): Расстояние до станции в рамках маршрута.
* ПризнакДействующейСтанции (boolean): Указывает, действует ли станция.
* ВремяОтправления (timestamp): Время отправления с данной станции.
* ВремяПрибытия (timestamp): Время прибытия на данную станцию.
* ПризнакРазрешенияПосадкиВысадки (boolean): Указывает, разрешена ли посадка/высадка.
* ПризнакТехническойОстановки (boolean): Указывает, является ли остановка технической.
* ПринадлежностьЖелезнойДороге (integer): Ссылка на железную дорогу (внешний ключ к ЖелезнаяДорога).

# **ОПИСАНИЕ СХЕМЫ БАЗЫ ДАННЫХ НА ЯЗЫКЕ SQL**

BEGIN;

CREATE TABLE IF NOT EXISTS public."ВариантРасписания"

(

"КодВарианта" integer NOT NULL,

"ПройденныйКилометраж" integer,

"ВремяПрибытияОтправленияСтанции" timestamp without time zone,

"ПризнакРазрешенияПосадкиВысадки" boolean,

"НаборОстановочныхПунктов" character varying(255) COLLATE pg\_catalog."default",

CONSTRAINT "ВариантРасписания\_pkey" PRIMARY KEY ("КодВарианта")

);

CREATE TABLE IF NOT EXISTS public."ЖелезнаяДорога"

(

"КодЖелезнойДороги" integer NOT NULL,

"ПолноеНазвание" character varying(255) COLLATE pg\_catalog."default",

"СокращенноеНазвание" character varying(100) COLLATE pg\_catalog."default",

"ПринадлежностьГосударству" boolean,

CONSTRAINT "ЖелезнаяДорога\_pkey" PRIMARY KEY ("КодЖелезнойДороги")

);

CREATE TABLE IF NOT EXISTS public."КалендарьДвиженияПоездов"

(

"ГодГрафика" integer NOT NULL,

"НомерПоезда" integer,

CONSTRAINT "КалендарьДвиженияПоездов\_ГодГраф" UNIQUE ("ГодГрафика")

);

CREATE TABLE IF NOT EXISTS public."Маршрут"

(

"КодМаршрута" integer NOT NULL,

"НачальнаяСтанция" integer,

"КонечнаяСтанция" integer,

"ВариантРасписания" integer NOT NULL,

"ПройденныйКилометраж" integer,

"ВремяОтправления" timestamp without time zone,

"ВремяПрибытия" timestamp without time zone,

CONSTRAINT "Маршрут\_pkey" PRIMARY KEY ("КодМаршрута")

);

CREATE TABLE IF NOT EXISTS public."Перевозчик"

(

"КодПеревозчика" integer NOT NULL,

"ПолноеНазвание" character varying(255) COLLATE pg\_catalog."default",

"СокращенноеНазвание" character varying(100) COLLATE pg\_catalog."default",

"ПринадлежностьГосударству" boolean,

CONSTRAINT "Перевозчик\_pkey" PRIMARY KEY ("КодПеревозчика")

);

CREATE TABLE IF NOT EXISTS public."Поезд"

(

"НомерПоезда" integer NOT NULL,

"КатегорияПоезда" character varying(50) COLLATE pg\_catalog."default",

"РегулярностьХождения" character varying(50) COLLATE pg\_catalog."default",

"Маршрут" integer,

"ГодГрафика" integer,

"Перевозчик" integer,

CONSTRAINT "Поезд\_pkey" PRIMARY KEY ("НомерПоезда")

);

CREATE TABLE IF NOT EXISTS public."СправочникСтанций"

(

"КодСтанции" integer NOT NULL,

"НазваниеСтанции" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"ТипСтанции" character varying(100) COLLATE pg\_catalog."default",

"ЖелезнаяДорога" integer,

"ПризнакТехническойОстановки" boolean,

CONSTRAINT "СправочникСтанций\_pkey" PRIMARY KEY ("КодСтанции", "НазваниеСтанции"),

CONSTRAINT "unique\_КодСтанции" UNIQUE ("КодСтанции")

);

CREATE TABLE IF NOT EXISTS public."СтанцияНаМаршруте"

(

"КодСтанции" integer,

"ВариантРасписания" integer,

"ПройденныйКилометраж" integer,

"ПризнакДействующейСтанции" boolean,

"ВремяОтправления" timestamp without time zone,

"ВремяПрибытия" timestamp without time zone,

"ПризнакРазрешенияПосадкиВысадки" boolean,

"ПризнакТехническойОстановки" boolean,

"ПринадлежностьЖелезнойДороге" integer

);

ALTER TABLE IF EXISTS public."Маршрут"

ADD CONSTRAINT "Маршрут\_ВариантРасписания\_fkey" FOREIGN KEY ("ВариантРасписания")

REFERENCES public."ВариантРасписания" ("КодВарианта") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."Поезд"

ADD CONSTRAINT "Поезд\_ГодГрафика\_fkey" FOREIGN KEY ("ГодГрафика")

REFERENCES public."КалендарьДвиженияПоездов" ("ГодГрафика") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."Поезд"

ADD CONSTRAINT "Поезд\_Маршрут\_fkey" FOREIGN KEY ("Маршрут")

REFERENCES public."Маршрут" ("КодМаршрута") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."Поезд"

ADD CONSTRAINT "Поезд\_Перевозчик\_fkey" FOREIGN KEY ("Перевозчик")

REFERENCES public."Перевозчик" ("КодПеревозчика") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."СправочникСтанций"

ADD CONSTRAINT "СправочникСтан\_ЖелезнаяДорога\_fkey" FOREIGN KEY ("ЖелезнаяДорога")

REFERENCES public."ЖелезнаяДорога" ("КодЖелезнойДороги") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."СтанцияНаМаршруте"

ADD CONSTRAINT "СтанцияНаМаршр\_ВариантРасписа\_fkey" FOREIGN KEY ("ВариантРасписания")

REFERENCES public."ВариантРасписания" ("КодВарианта") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

ALTER TABLE IF EXISTS public."СтанцияНаМаршруте"

ADD CONSTRAINT "СтанцияНаМаршруте\_КодСтанции\_fkey" FOREIGN KEY ("КодСтанции")

REFERENCES public."СправочникСтанций" ("КодСтанции") MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE;

END;

# **ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ЗАПРОСОВ НА ЯЗЫКЕ SQL**

Чтобы просмотреть сохранённые данные в таблицах, можно воспользоваться оператором SELECT. Самый простой пример использования оператора SELECT – выдача содержимого таблицы целиком:

SELECT \* FROM <имя таблицы>

Cодержимое таблиц, полученное при помощи запроса SELECT представлено на рисунках:

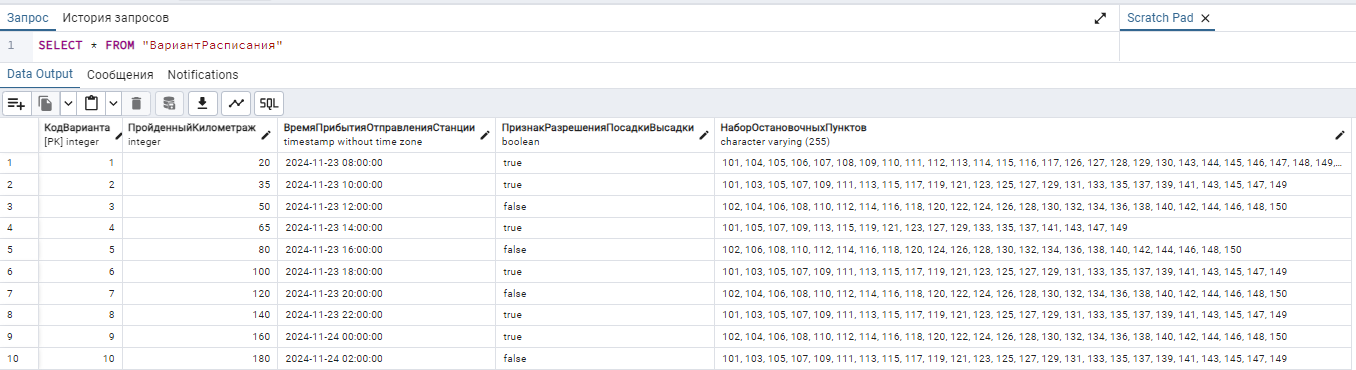


Рисунок 3 – Результат запроса SELECT для таблицы ВариантРасписания

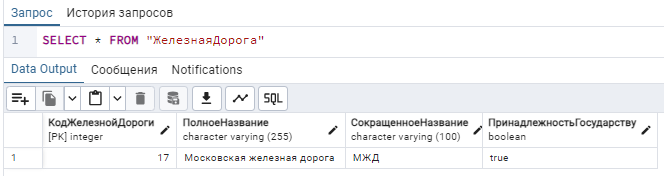


Рисунок 4 – Результат запроса SELECT для таблицы ЖелезнаяДорога

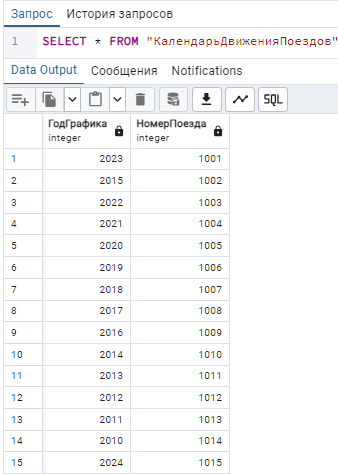


Рисунок 5 – Результат запроса SELECT для таблицы КалендарьДвиженияПоездов

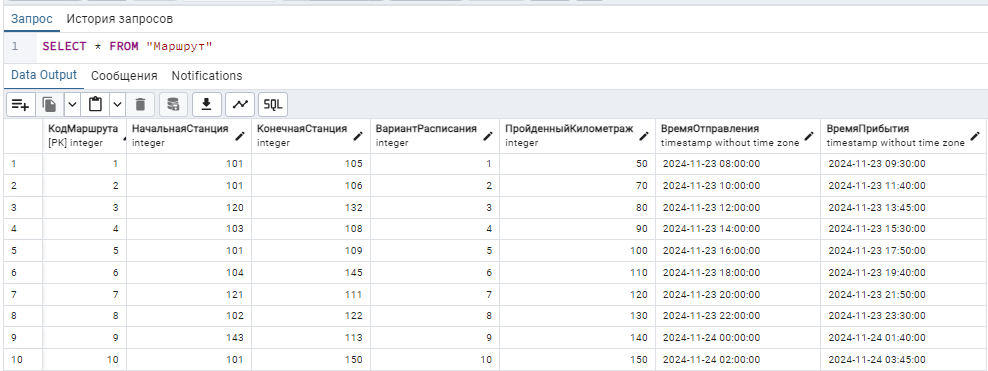


Рисунок 6 – Результат запроса SELECT для таблицы Маршрут

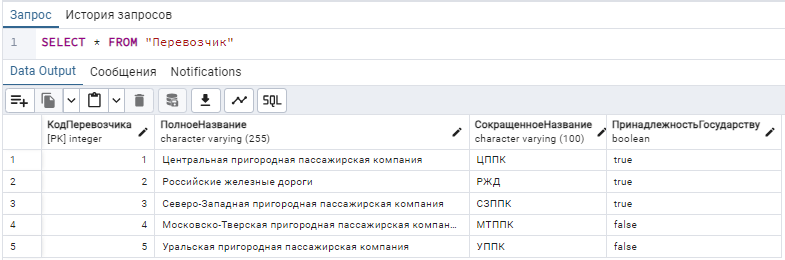


Рисунок 7 – Результат запроса SELECT для таблицы Перевозчик

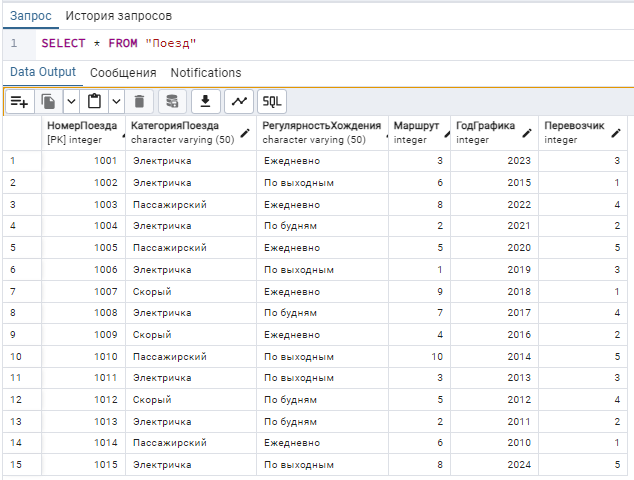
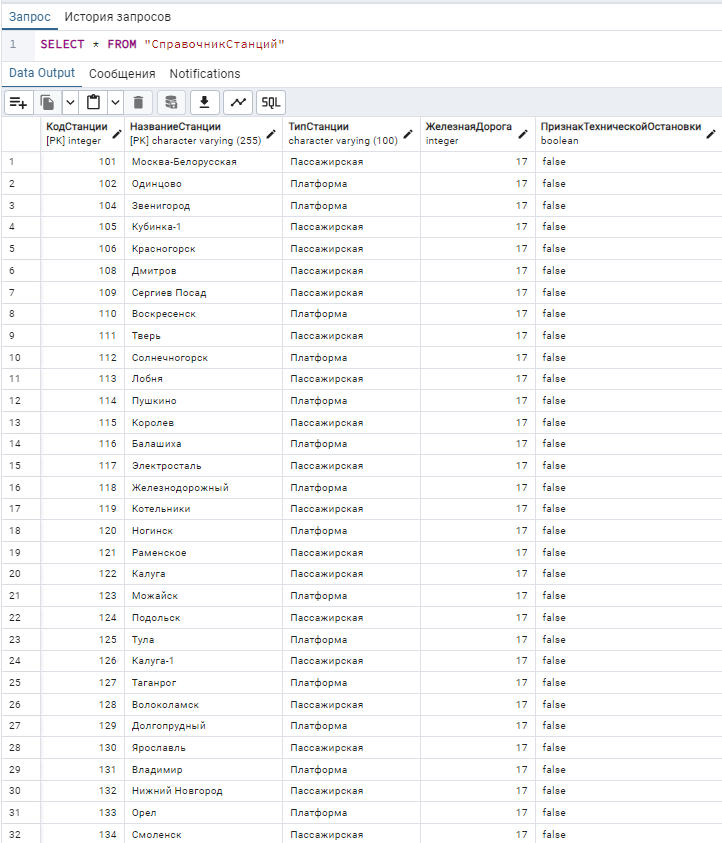


Рисунок 8 – Результат запроса SELECT для таблицы Поезд



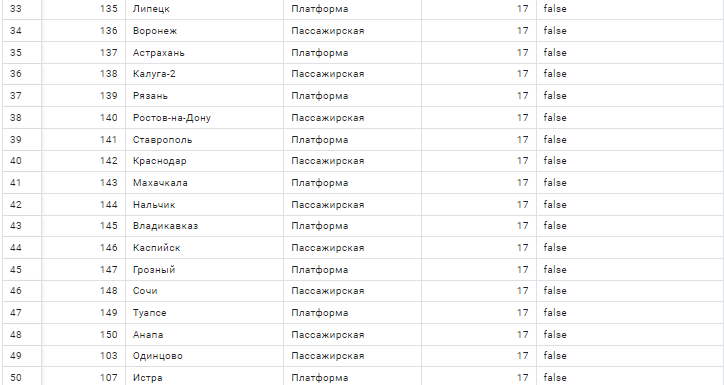
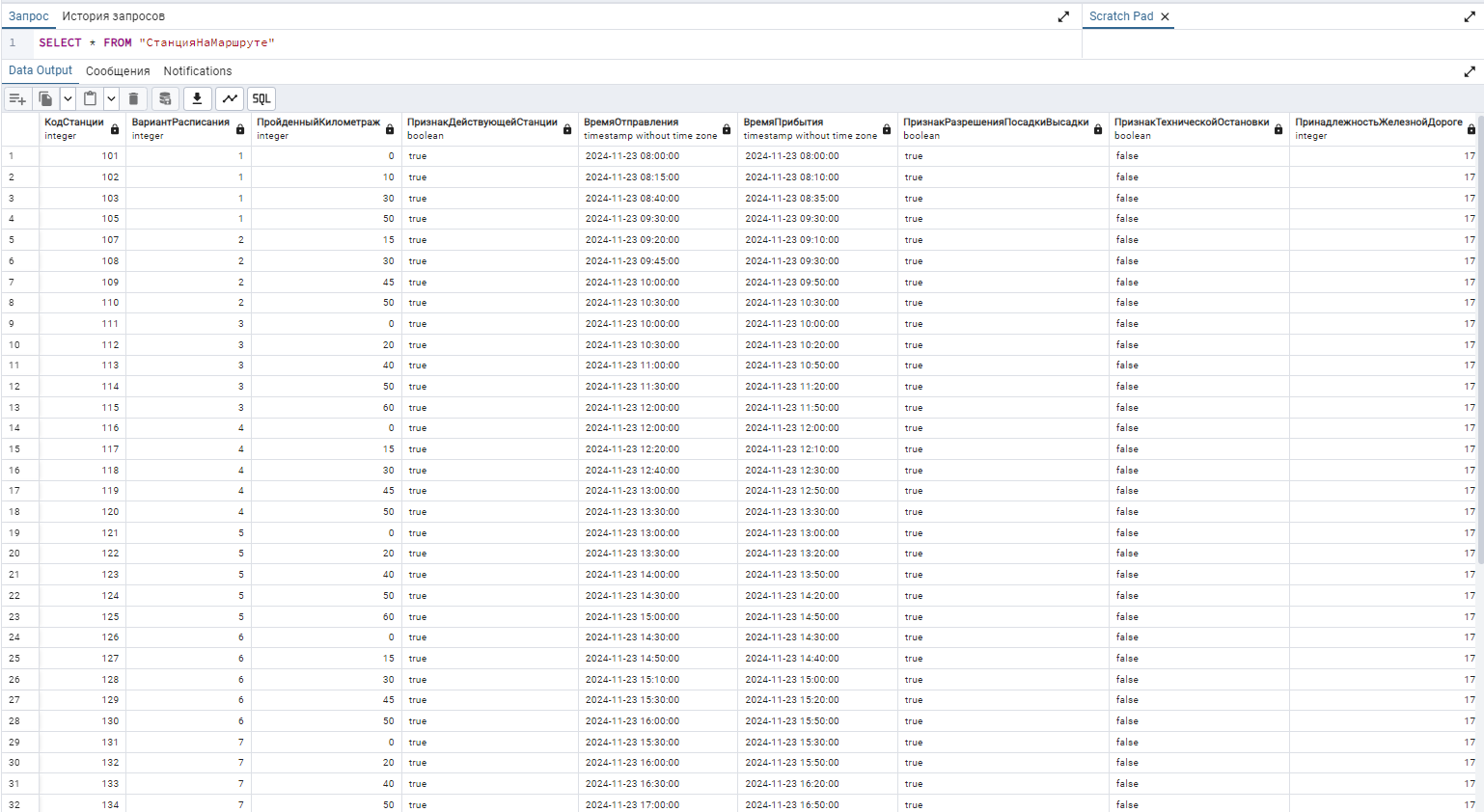


Рисунок 9 – Результат запроса SELECT для таблицы СправочникСтанций



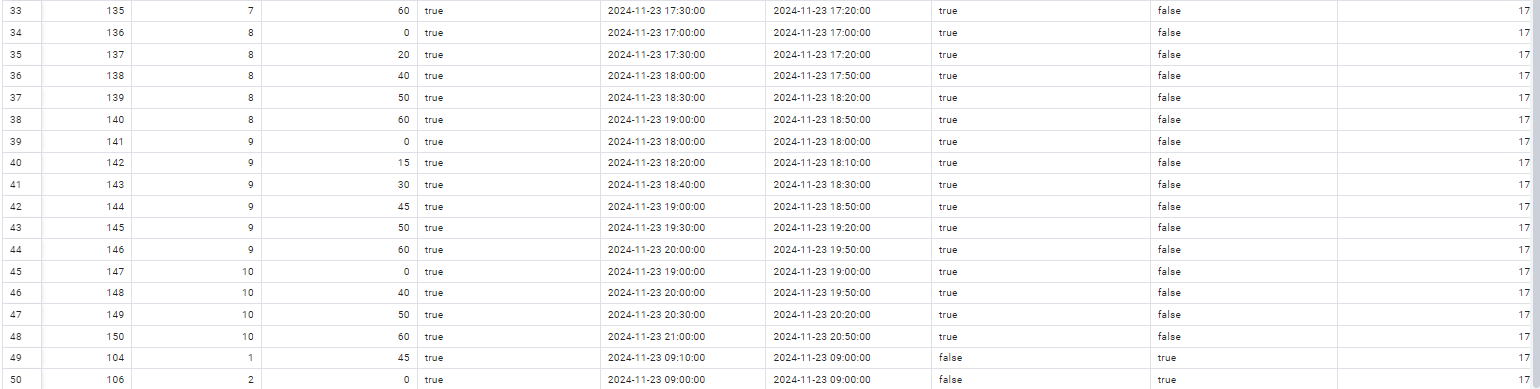


Рисунок 10 – Результат запроса SELECT для таблицы СтанцияНаМаршруте

**Описание запросов для взаимодействия с базой данных**

**Добавление записей в таблицы**

Для вставки данных в таблицы используется следующий SQL-запрос:

INSERT INTO public."{table\_name\_eng}" ({", ".join(quoted\_columns)}) VALUES ({placeholders});

Описание:

1. INSERT INTO public."{table\_name\_eng}": Команда для добавления новой строки в указанную таблицу.
   * "{table\_name\_eng}" - наименование таблицы, в которую вносятся данные.
2. ({", ".join(quoted\_columns)}): Указаны имена столбцов, в которые записываются значения.
   * quoted\_columns - перечень названий столбцов, оформленных в кавычках. Это необходимо для случаев, когда в именах используются пробелы, специфический регистр символов или зарезервированные слова.
3. VALUES ({placeholders}): Указываются значения для перечисленных выше столбцов.
   * placeholders - это заполнители, такие как %s, $1, $2, применяемые при параметризованной передаче данных.

**Обновление данных**

Для изменения уже существующих записей используется запрос следующего вида:

UPDATE public."{table\_name\_eng}" SET {placeholders} WHERE "{primary\_key\_column}" = %s;

Описание:

1. UPDATE public."{table\_name\_eng}": Команда для изменения записей в указанной таблице.
   * "{table\_name\_eng}" - таблица, где производится обновление данных.
2. SET {placeholders}: Здесь указываются столбцы, подлежащие изменению, и их новые значения.
   * placeholders - это пары вида "column\_name" = %s, перечисленные через запятую.
   * Пример: Если placeholders = "name" = %s, "price" = %s, то запрос заменяет значения в столбцах name и price.
3. WHERE "{primary\_key\_column}" = %s: Условие, задающее, какая строка подлежит обновлению.
   * "{primary\_key\_column}" - имя столбца, выполняющего роль уникального идентификатора записи.
   * %s - параметр, который заменяется конкретным значением.

**Удаление данных**

Для удаления строк, соответствующих определённым условиям, используется следующий запрос:

DELETE FROM public."{table\_name\_eng}" WHERE "{key\_to\_delete}" = %s;

Описание:

1. DELETE FROM public."{table\_name\_eng}": Удаляет строки из указанной таблицы.
   * "{table\_name\_eng}" - таблица, из которой удаляются записи.
2. WHERE "{key\_to\_delete}" = %s: Условие для поиска строки, подлежащей удалению.
   * "{key\_to\_delete}" - столбец, который используется для фильтрации.
   * %s - параметр, заменяемый значением, определяющим строку.

**Запросы из предметной области**

**1. Выдать список наименований всех платформ (тип раздельного пункта - 7) Москвской ж.д. (код дороги - 17)**

SELECT "НазваниеСтанции", "ТипСтанции", "ЖелезнаяДорога"  
FROM public."СправочникСтанций"  
WHERE "ТипСтанции" = 'Платформа' AND "ЖелезнаяДорога" = 17;

Описание:

1. SELECT "НазваниеСтанции", "ТипСтанции", "ЖелезнаяДорога": Указывает, какие столбцы будут включены в результат выборки.
   * "НазваниеСтанции" - название станции.
   * "ТипСтанции" - тип станции, например, платформа, вокзал и т. д.
   * "ЖелезнаяДорога" - идентификатор железной дороги, к которой относится станция.
2. FROM public."СправочникСтанций": Указывает таблицу, из которой выполняется выборка данных.
   * "СправочникСтанций" - справочная таблица, содержащая данные о станциях.
3. WHERE "ТипСтанции" = 'Платформа' AND "ЖелезнаяДорога" = 17: Задаёт условия для фильтрации данных.
   * "ТипСтанции" = 'Платформа' - фильтрует записи, оставляя только станции, тип которых равен 'Платформа'.
   * AND "ЖелезнаяДорога" = 17 - добавляет дополнительное условие, оставляя только те записи, где идентификатор железной дороги равен 17.

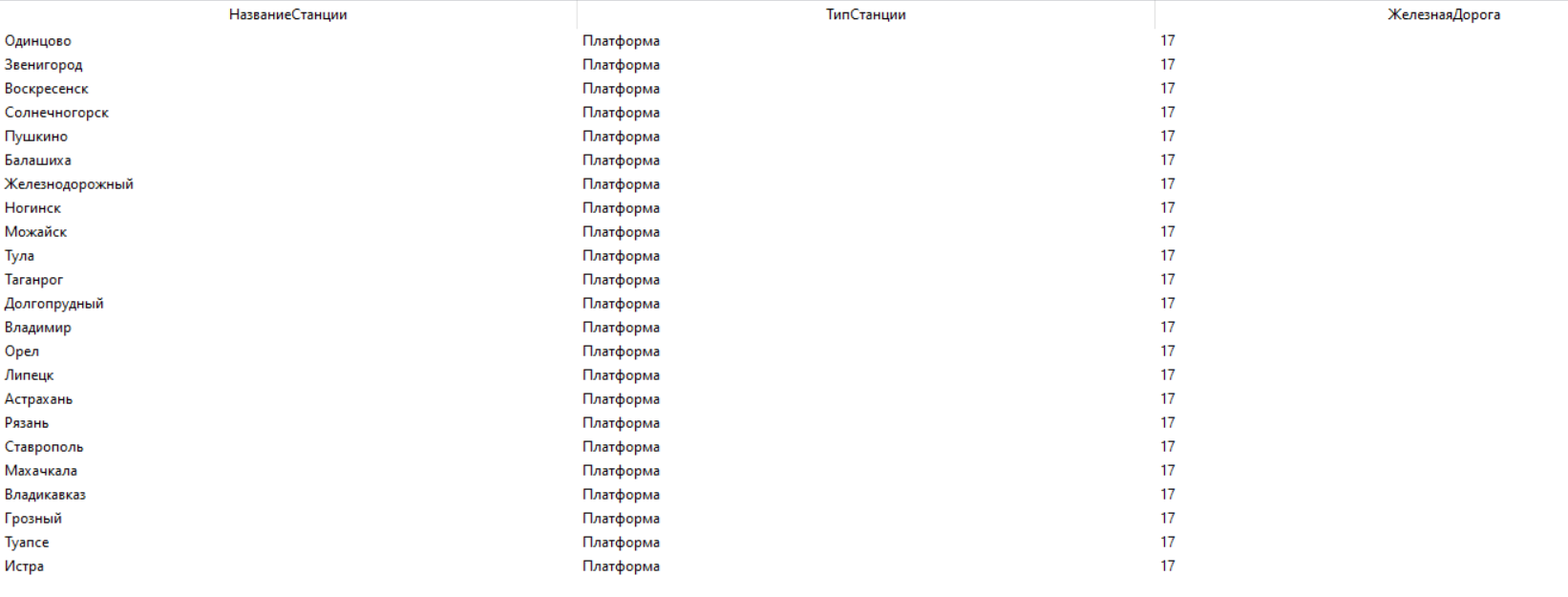


Рисунок 11 – Результат первого дополнительного запроса из предметной области

**2. Выдать список наименований всех раздельных пунктов заданного**

**варианта маршрута (по коду варианта)**

SELECT s."НазваниеСтанции", m."КодМаршрута"  
FROM public."СправочникСтанций" s  
JOIN public."СтанцияНаМаршруте" sn ON s."КодСтанции" = sn."КодСтанции"  
JOIN public."Маршрут" m ON sn."ВариантРасписания" = m."ВариантРасписания"  
WHERE m."КодМаршрута" = %s;

Описание:

1. SELECT s."НазваниеСтанции", m."КодМаршрута"  
   Указывает, какие столбцы будут включены в результирующую таблицу:
   * s."НазваниеСтанции" - название станции, извлекается из таблицы "СправочникСтанций".
   * m."КодМаршрута" - код маршрута, извлекается из таблицы "Маршрут".
2. FROM public."СправочникСтанций" s  
   Указывает основную таблицу для выборки.
   * "СправочникСтанций" - таблица, содержащая справочную информацию о станциях.
   * s - псевдоним для таблицы, который упрощает запись при обращении к её столбцам.
3. JOIN public."СтанцияНаМаршруте" sn ON s."КодСтанции" = sn."КодСтанции"  
   Осуществляет соединение таблицы "СправочникСтанций" с таблицей "СтанцияНаМаршруте".
   * Связь выполняется по полю "КодСтанции", где значения из обеих таблиц должны совпадать.
4. JOIN public."Маршрут" m ON sn."ВариантРасписания" = m."ВариантРасписания"  
   Осуществляет соединение таблицы "СтанцияНаМаршруте" с таблицей "Маршрут".
   * Связь выполняется по полю "ВариантРасписания", где значения из обеих таблиц должны совпадать.
5. WHERE m."КодМаршрута" = %s  
   Задаёт условие фильтрации данных:
   * Только строки, где "КодМаршрута" равен переданному параметру %s, будут включены в результат.

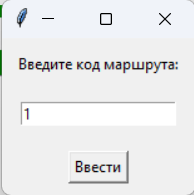


Рисунок 12 – Окно для ввода дополнительных параметров в запрос



Рисунок 13 – Результат второго дополнительного запроса из предметной области

**3. Выдать повторяющиеся имена раздельных пунктов на Московской**

**дороге**

SELECT   
 s."НазваниеСтанции",   
COUNT(\*) AS "Количество"  
FROM   
public."СправочникСтанций" s  
JOIN   
public."ЖелезнаяДорога" j ON s."ЖелезнаяДорога" = j."КодЖелезнойДороги"  
WHERE   
j."ПолноеНазвание" LIKE '%Московская железная дорога%' -- Фильтруем по названию железной дороги  
GROUP BY   
s."НазваниеСтанции"  
HAVING   
 COUNT(\*) > 1;

Описание:

1. SELECT s."НазваниеСтанции", COUNT(\*) AS "Количество"  
   Указывает, какие данные нужно извлечь:
   * s."НазваниеСтанции" - название станции из таблицы "СправочникСтанций".
   * COUNT(\*) AS "Количество" - подсчёт числа записей для каждой станции.
2. FROM public."СправочникСтанций" s  
   Указывает основную таблицу для выборки:
   * "СправочникСтанций" - таблица, содержащая данные о станциях.
   * s - псевдоним для таблицы.
3. JOIN public."ЖелезнаяДорога" j ON s."ЖелезнаяДорога" = j."КодЖелезнойДороги"  
   Осуществляет соединение таблицы "СправочникСтанций" с таблицей "ЖелезнаяДорога".
   * Связь выполняется через поле "ЖелезнаяДорога" таблицы "СправочникСтанций" и поле "КодЖелезнойДороги" таблицы "ЖелезнаяДорога".
4. WHERE j."ПолноеНазвание" LIKE '%Московская железная дорога%'  
   Фильтрует данные так, чтобы в результат попали только записи, где поле "ПолноеНазвание" в таблице "ЖелезнаяДорога" содержит текст "Московская железная дорога".
5. GROUP BY s."НазваниеСтанции"  
   Группирует данные по полю "НазваниеСтанции", чтобы подсчитать количество записей для каждой станции.
6. HAVING COUNT(\*) > 1  
   Фильтрует результаты группировки, оставляя только те станции, у которых количество записей превышает 1.

 Рисунок 14 – Результат третьего дополнительного запроса из предметной области

**4. Выдать список всех поездов заданного направления (четность номера поезда) и начальной станции отправления поезда**

SELECT   
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 s."НазваниеСтанции" AS "НачальнаяСтанция",  
 m."ВремяОтправления",  
 m."ВремяПрибытия"  
FROM   
 public."Поезд" p  
JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
JOIN   
 public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
WHERE   
 p."НомерПоезда" % 2 = 0   
ORDER BY   
 p."НомерПоезда";

Описание:

1. SELECT  
   Указывает, какие данные нужно извлечь:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда из таблицы "Поезд".
   * s."НазваниеСтанции" AS "НачальнаяСтанция" - название начальной станции из таблицы "СправочникСтанций". Используется псевдоним AS "НачальнаяСтанция" для переименования столбца.
   * m."ВремяОтправления" - время отправления из таблицы "Маршрут".
   * m."ВремяПрибытия" - время прибытия из таблицы "Маршрут".
2. FROM public."Поезд" p  
   Указывает основную таблицу для выборки:
   * "Поезд" - таблица, содержащая информацию о поездах.
   * p - псевдоним для таблицы.
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Осуществляет соединение таблицы "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" таблицы "Поезд" и поле "КодМаршрута" таблицы "Маршрут".
   * Это позволяет получить информацию о маршруте для каждого поезда.
4. JOIN public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
   Осуществляет соединение таблицы "Маршрут" с таблицей "СправочникСтанций":
   * Связь выполняется через поле "НачальнаяСтанция" таблицы "Маршрут" и поле "КодСтанции" таблицы "СправочникСтанций".
   * Это позволяет получить название начальной станции для каждого маршрута.
5. WHERE p."НомерПоезда" % 2 = 0  
   Фильтрует данные так, чтобы в результат попали только поезда с чётными номерами:
   * "НомерПоезда" % 2 - операция вычисления остатка от деления номера поезда на 2.
   * = 0 - условие для выбора чётных чисел.
6. ORDER BY p."НомерПоезда"  
   Сортирует результат по номеру поезда в возрастающем порядке.

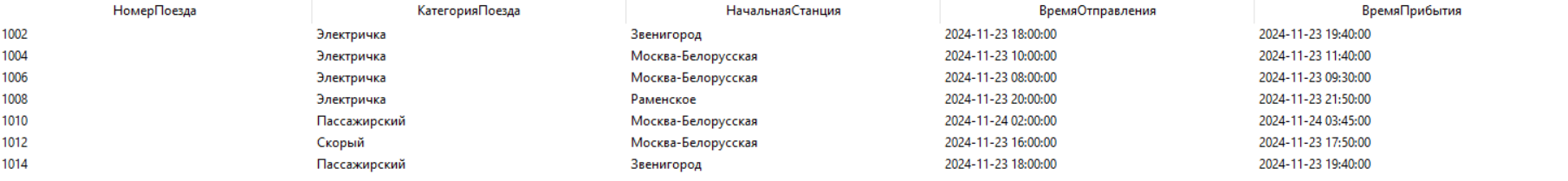


Рисунок 15 – Результат четвертого дополнительного запроса из предметной области

**5. Выдать список всех поездов, останавливающихся для посадки/высадки пассажиров на заданной станции A**

SELECT   
 p."НомерПоезда",   
 p."КатегорияПоезда",   
 s."НазваниеСтанции",   
 snm."ВремяПрибытия",   
 snm."ВремяОтправления",   
 snm."ПризнакРазрешенияПосадкиВысадки"   
FROM   
 public."Поезд" p   
JOIN   
 public."Маршрут" m   
 ON p."Маршрут" = m."КодМаршрута"   
JOIN   
 public."СтанцияНаМаршруте" snm   
 ON m."ВариантРасписания" = snm."ВариантРасписания"   
JOIN   
 public."СправочникСтанций" s   
 ON snm."КодСтанции" = s."КодСтанции"   
WHERE   
 s."НазваниеСтанции" = %s   
 AND snm."ПризнакРазрешенияПосадкиВысадки" = TRUE   
ORDER BY   
 p."НомерПоезда", snm."ВремяПрибытия";

**Описание:**

1. SELECT  
   Указывает поля, которые будут выбраны в результате:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда (например, пассажирский, скорый).
   * s."НазваниеСтанции" - название станции из таблицы "СправочникСтанций".
   * snm."ВремяПрибытия" - время прибытия поезда на станцию из таблицы "СтанцияНаМаршруте".
   * snm."ВремяОтправления" - время отправления поезда со станции из таблицы "СтанцияНаМаршруте".
   * snm."ПризнакРазрешенияПосадкиВысадки" - флаг, показывающий, разрешена ли посадка/высадка на этой станции (логическое значение).
2. FROM public."Поезд" p  
   Указывает основную таблицу для выборки:
   * "Поезд" содержит информацию о поездах.
   * p - псевдоним для таблицы.
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" в таблице "Поезд" и поле "КодМаршрута" в таблице "Маршрут".
   * Это позволяет получить данные о маршрутах, связанных с поездами.
4. JOIN public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
   Соединяет таблицу "Маршрут" с таблицей "СтанцияНаМаршруте":
   * Связь выполняется через поле "ВариантРасписания" в обеих таблицах.
   * Это позволяет получить данные о станциях, включённых в маршрут.
5. JOIN public."СправочникСтанций" s ON snm."КодСтанции" = s."КодСтанции"  
   Соединяет таблицу "СтанцияНаМаршруте" с таблицей "СправочникСтанций":
   * Связь выполняется через поле "КодСтанции" в обеих таблицах.
   * Это позволяет получить названия станций по их коду.
6. WHERE  
   Фильтрует данные по двум условиям:
   * s."НазваниеСтанции" = %s - выбирает записи только для указанной станции. Параметр %s заменяется реальным названием станции.
   * snm."ПризнакРазрешенияПосадкиВысадки" = TRUE - выбирает записи, где разрешена посадка/высадка пассажиров.
7. ORDER BY  
   Упорядочивает результат:
   * Сначала по полю p."НомерПоезда" в возрастающем порядке.
   * Затем по полю snm."ВремяПрибытия".

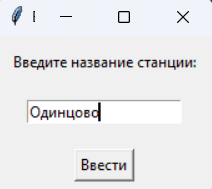
  
Рисунок 16 – Окно для ввода дополнительных параметров в запрос



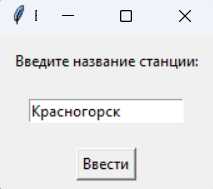
Рисунок 17 – Результат пятого дополнительного запроса из предметной области

**6. Выдать список всех поездов, делающих техническую остановку (без посадки/высадки пассажиров) на заданной станции A**

SELECT   
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 s."НазваниеСтанции",  
 snm."ВремяПрибытия",  
 snm."ВремяОтправления",  
 snm."ПризнакТехническойОстановки",  
 snm."ПризнакРазрешенияПосадкиВысадки"  
FROM   
 public."Поезд" p  
JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
JOIN   
 public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
JOIN   
 public."СправочникСтанций" s ON snm."КодСтанции" = s."КодСтанции"  
WHERE   
 s."НазваниеСтанции" = %s  
 AND snm."ПризнакТехническойОстановки" = TRUE  
 AND snm."ПризнакРазрешенияПосадкиВысадки" = FALSE;

**Описание:**

1. SELECT  
   Указывает поля, которые будут выбраны в результате:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда (например, пассажирский, грузовой).
   * p."РегулярностьХождения" - информация о регулярности движения поезда (ежедневный, по определённым дням и т. д.).
   * s."НазваниеСтанции" - название станции из таблицы "СправочникСтанций".
   * snm."ВремяПрибытия" - время прибытия поезда на станцию из таблицы "СтанцияНаМаршруте".
   * snm."ВремяОтправления" - время отправления поезда со станции.
   * snm."ПризнакТехническойОстановки" - флаг, указывающий, что остановка носит технический характер.
   * snm."ПризнакРазрешенияПосадкиВысадки" - флаг, указывающий, разрешена ли посадка или высадка на данной станции.
2. FROM public."Поезд" p  
   Указывает основную таблицу для выборки:
   * "Поезд" содержит информацию о поездах.
   * p - псевдоним для таблицы.
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" в таблице "Поезд" и поле "КодМаршрута" в таблице "Маршрут".
4. JOIN public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
   Соединяет таблицу "Маршрут" с таблицей "СтанцияНаМаршруте":
   * Связь осуществляется через поле "ВариантРасписания" в обеих таблицах.
   * Это позволяет получить данные о станциях, связанных с маршрутами.
5. JOIN public."СправочникСтанций" s ON snm."КодСтанции" = s."КодСтанции"  
   Соединяет таблицу "СтанцияНаМаршруте" с таблицей "СправочникСтанций":
   * Связь осуществляется через поле "КодСтанции".
   * Это позволяет извлечь название станции.
6. WHERE  
   Указывает условия фильтрации записей:
   * s."НазваниеСтанции" = %s - выбирает записи для указанной станции. Параметр %s заменяется реальным названием станции.
   * snm."ПризнакТехническойОстановки" = TRUE - выбирает записи, где остановка является технической.
   * snm."ПризнакРазрешенияПосадкиВысадки" = FALSE - выбирает записи, где посадка и высадка запрещены.

  
Рисунок 18 – Окно для ввода дополнительных параметров в запрос

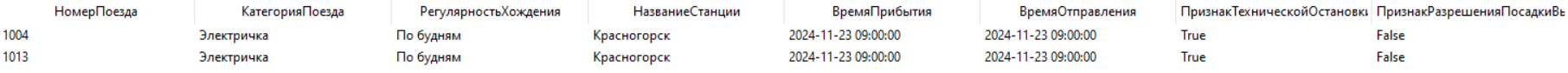


Рисунок 19 – Результат шестого дополнительного запроса из предметной области

**7. Выдать список поездов, на которых можно проехать от станции А до станции B**

SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция"  
FROM   
 public."Поезд" p  
JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
WHERE   
 m."НачальнаяСтанция" = %s  
 AND m."КонечнаяСтанция" = %s  
ORDER BY   
 p."НомерПоезда";

**Описание:**

1. SELECT DISTINCT  
   Указывает поля, которые будут выбраны в результате, при этом исключая дубликаты:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда (например, пассажирский, экспресс).
   * p."РегулярностьХождения" - регулярность движения поезда (например, ежедневно, по определённым дням недели).
   * m."НачальнаяСтанция" - начальная станция маршрута из таблицы "Маршрут".
   * m."КонечнаяСтанция" - конечная станция маршрута из таблицы "Маршрут".
2. FROM public."Поезд" p  
   Указывает основную таблицу для выборки:
   * "Поезд" - таблица с информацией о поездах.
   * p - псевдоним для таблицы.
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" в таблице "Поезд" и поле "КодМаршрута" в таблице "Маршрут".
   * Это позволяет получить информацию о маршруте, который обслуживает конкретный поезд.
4. WHERE  
   Указывает условия фильтрации записей:
   * m."НачальнаяСтанция" = %s - выбирает маршруты, где начальная станция совпадает с указанной. Параметр %s будет заменён на фактическое значение начальной станции.
   * m."КонечнаяСтанция" = %s - выбирает маршруты, где конечная станция совпадает с указанной. Параметр %s будет заменён на фактическое значение конечной станции.
5. ORDER BY p."НомерПоезда"  
   Сортирует результаты по номеру поезда в порядке возрастания.

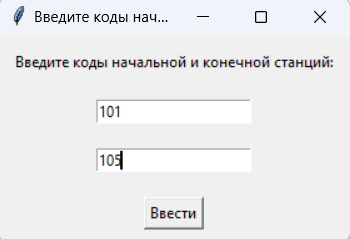


Рисунок 20 – Окно для ввода дополнительных параметров в запрос



Рисунок 21 – Результат седьмого дополнительного запроса из предметной области

**8. Выдать список поездов, прибывающих на станцию А, в заданном направлении в промежутке времени [t1, t2] в заданную дату**

SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."ВремяПрибытия",  
 m."ВремяОтправления",  
 cd."ГодГрафика",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция"  
FROM   
 public."Поезд" p  
JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
JOIN   
 public."КалендарьДвиженияПоездов" cd ON p."ГодГрафика" = cd."ГодГрафика"  
JOIN   
 public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
WHERE   
 s."НазваниеСтанции" = %s  
 AND cd."ГодГрафика" = %s  
 AND m."ВремяПрибытия" BETWEEN %s AND %s  
ORDER BY   
 m."ВремяПрибытия";

**Описание:**

1. SELECT DISTINCT  
   Указывает поля, которые будут выбраны в результате, при этом исключая дубликаты:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда (например, пассажирский, экспресс).
   * p."РегулярностьХождения" - регулярность движения поезда.
   * m."ВремяПрибытия" - время прибытия поезда на станцию.
   * m."ВремяОтправления" - время отправления поезда с начальной станции.
   * cd."ГодГрафика" - год графика движения поезда.
   * m."НачальнаяСтанция" - начальная станция маршрута.
   * m."КонечнаяСтанция" - конечная станция маршрута.
2. FROM public."Поезд" p  
   Указывает основную таблицу для выборки:
   * "Поезд" - таблица с информацией о поездах.
   * p - псевдоним для таблицы.
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" в таблице "Поезд" и поле "КодМаршрута" в таблице "Маршрут".
   * Это позволяет получить информацию о маршруте, который обслуживает конкретный поезд.
4. JOIN public."КалендарьДвиженияПоездов" cd ON p."ГодГрафика" = cd."ГодГрафика"  
   Соединяет таблицу "Поезд" с таблицей "КалендарьДвиженияПоездов":
   * Связь выполняется через поле "ГодГрафика" в обеих таблицах.
   * Это позволяет получить информацию о годе графика поезда.
5. JOIN public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
   Соединяет таблицу "Маршрут" с таблицей "СправочникСтанций":
   * Связь выполняется через поле "НачальнаяСтанция" в таблице "Маршрут" и поле "КодСтанции" в таблице "СправочникСтанций".
   * Это позволяет получить информацию о начальной станции маршрута.
6. WHERE  
   Указывает условия фильтрации записей:
   * s."НазваниеСтанции" = %s - выбирает маршруты, где начальная станция совпадает с указанной. Параметр %s будет заменён на фактическое значение названия станции.
   * cd."ГодГрафика" = %s - выбирает только записи для указанного года графика. Параметр %s будет заменён на фактическое значение года.
   * m."ВремяПрибытия" BETWEEN %s AND %s - выбирает поезда, чье время прибытия находится в указанном промежутке. Параметры %s будут заменены на фактические значения времени.
7. ORDER BY m."ВремяПрибытия"  
   Сортирует результаты по времени прибытия в порядке возрастания.

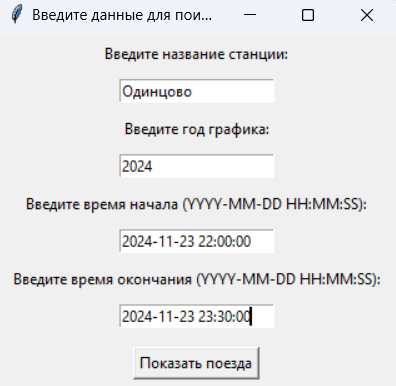


Рисунок 22 – Окно для ввода дополнительных параметров в запрос



Рисунок 23 – Результат восьмого дополнительного запроса из предметной области

**9. Выдать поезда, которые следуют со всеми остановками для посадки/высадки**

SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция",  
 snm."ПризнакРазрешенияПосадкиВысадки"  
FROM   
 public."Поезд" p  
JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
JOIN   
 public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
WHERE   
 snm."ПризнакРазрешенияПосадкиВысадки" = TRUE  
 AND NOT EXISTS (  
 SELECT 1  
 FROM public."СтанцияНаМаршруте" s  
 WHERE s."ВариантРасписания" = snm."ВариантРасписания"  
 AND s."ПризнакРазрешенияПосадкиВысадки" = FALSE  
 )  
ORDER BY   
 p."НомерПоезда";

**Описание:**

1. SELECT DISTINCT  
   Указывает поля, которые будут выбраны в результат, исключая дубликаты:
   * p."НомерПоезда" - номер поезда из таблицы "Поезд".
   * p."КатегорияПоезда" - категория поезда (например, пассажирский, экспресс).
   * p."РегулярностьХождения" - регулярность хождения поезда.
   * m."НачальнаяСтанция" - начальная станция маршрута.
   * m."КонечнаяСтанция" - конечная станция маршрута.
   * snm."ПризнакРазрешенияПосадкиВысадки" - признак разрешения посадки и высадки на станции.
2. FROM public."Поезд" p  
   Основная таблица для выборки:
   * "Поезд" - таблица, содержащая информацию о поездах.
   * p - псевдоним для таблицы "Поезд".
3. JOIN public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Поезд" с таблицей "Маршрут":
   * Связь выполняется через поле "Маршрут" в таблице "Поезд" и поле "КодМаршрута" в таблице "Маршрут".
   * Это позволяет получить информацию о маршруте для каждого поезда.
4. JOIN public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
   Соединяет таблицу "Маршрут" с таблицей "СтанцияНаМаршруте":
   * Связь выполняется через поле "ВариантРасписания" в обеих таблицах.
   * Это позволяет получить информацию о станциях на маршруте, включая информацию о разрешении на посадку/высадку пассажиров.
5. WHERE  
   Указывает условия фильтрации записей:
   * snm."ПризнакРазрешенияПосадкиВысадки" = TRUE - выбирает только те станции, на которых разрешена посадка и высадка.
   * AND NOT EXISTS - исключает те маршруты, где для этого же варианта расписания существует хотя бы одна станция с запретом на посадку/высадку.
     + В подзапросе:
       - s."ВариантРасписания" = snm."ВариантРасписания" - связывает таблицу "СтанцияНаМаршруте" с подзапросом по варианту расписания.
       - s."ПризнакРазрешенияПосадкиВысадки" = FALSE - исключает маршруты с запретом посадки/высадки хотя бы на одной из станций.
6. ORDER BY p."НомерПоезда"  
   Сортирует результаты по номеру поезда в порядке возрастания.

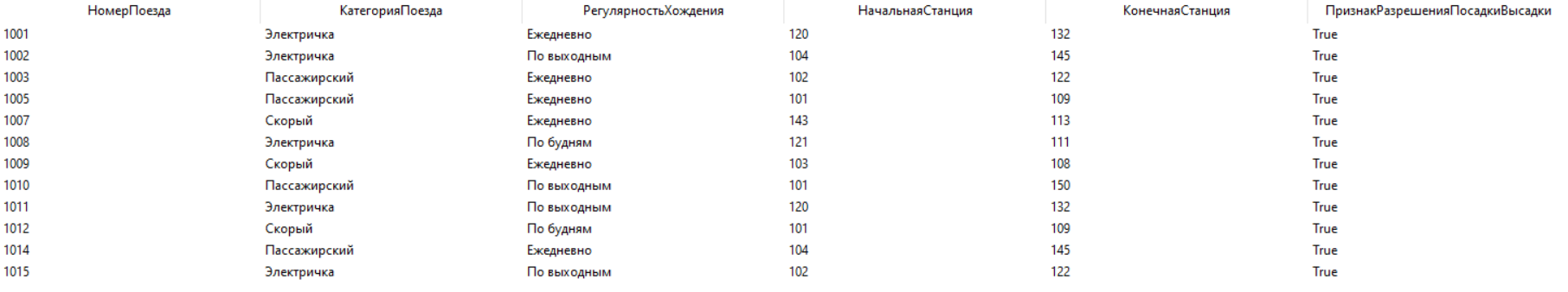


Рисунок 24 – Результат девятого дополнительного запроса из предметной области

**10. Выдать расписание поездов для заданного маршрута (по станции отправления и назначения) в табличной форме, как это делается стандартном расписании (слева – станции, вверху – поезда, элемент таблицы – время отправления со станции)**

SELECT DISTINCT  
 s."НазваниеСтанции" AS "Станция",  
 p."НомерПоезда" AS "Номер поезда",  
 p."КатегорияПоезда" AS "Категория поезда",  
 sm."ВремяОтправления" AS "Время отправления"  
FROM   
 public."СтанцияНаМаршруте" sm  
JOIN   
 public."Маршрут" m ON sm."ВариантРасписания" = m."ВариантРасписания"  
JOIN   
 public."СправочникСтанций" s ON sm."КодСтанции" = s."КодСтанции"  
JOIN   
 public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
WHERE   
 m."НачальнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND m."КонечнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
ORDER BY   
 p."НомерПоезда", sm."ВремяОтправления";

**Описание:**

1. SELECT DISTINCT  
   Запрашивает уникальные записи по следующим полям:
   * s."НазваниеСтанции" AS "Станция" - название станции.
   * p."НомерПоезда" AS "Номер поезда" - номер поезда.
   * p."КатегорияПоезда" AS "Категория поезда" - категория поезда (например, пассажирский, экспресс).
   * sm."ВремяОтправления" AS "Время отправления" - время отправления поезда с данной станции.
2. FROM public."СтанцияНаМаршруте" sm  
   Основная таблица, содержащая информацию о станциях на маршруте:
   * "СтанцияНаМаршруте" - таблица, в которой указаны станции, через которые проходит маршрут поезда.
   * sm - псевдоним для этой таблицы.
3. JOIN public."Маршрут" m ON sm."ВариантРасписания" = m."ВариантРасписания"  
   Соединяет таблицу "СтанцияНаМаршруте" с таблицей "Маршрут":
   * Связь через поле "ВариантРасписания".
   * Это позволяет получить информацию о маршруте, связанном с каждой станцией на маршруте.
4. JOIN public."СправочникСтанций" s ON sm."КодСтанции" = s."КодСтанции"  
   Соединяет таблицу "СтанцияНаМаршруте" с таблицей "СправочникСтанций":
   * Связь через поле "КодСтанции".
   * Это позволяет получить название каждой станции на маршруте.
5. JOIN public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу "Маршрут" с таблицей "Поезд":
   * Связь через поле "Маршрут" в таблице "Поезд" и "КодМаршрута" в таблице "Маршрут".
   * Это позволяет получить информацию о поездах, соответствующих маршруту.
6. WHERE  
   Указывает условия фильтрации записей:
   * m."НачальнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s) - начальная станция маршрута должна быть равна указанной станции.
   * m."КонечнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s) - конечная станция маршрута должна быть равна указанной станции.
   * Подзапросы SELECT "КодСтанции" ... получают код станции, который будет использован в качестве фильтра для начальной и конечной станции маршрута.
7. ORDER BY  
   Сортирует результаты по следующим полям:
   * p."НомерПоезда" - номер поезда в порядке возрастания.
   * sm."ВремяОтправления" - время отправления поезда с каждой станции в порядке возрастания.

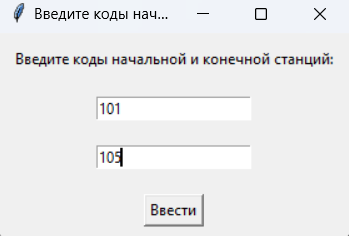


Рисунок 25 – Окно для ввода дополнительных параметров в запрос

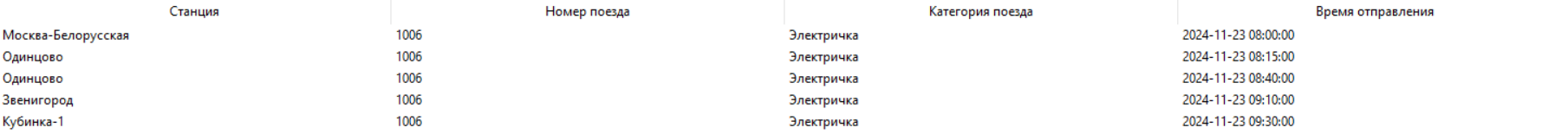


Рисунок 26 – Результат десятого дополнительного запроса из предметной области

**11. Поиск оптимального по времени поезда, следующего от станции А к станции В в заданном промежутке времени [t1, t2]**

SELECT   
 p."НомерПоезда" AS "Номер поезда",  
 p."КатегорияПоезда" AS "Категория поезда",  
 sm\_start."ВремяОтправления" AS "Время отправления",  
 sm\_end."ВремяПрибытия" AS "Время прибытия",  
 sm\_start."ПройденныйКилометраж" AS "Километраж от А",  
 sm\_end."ПройденныйКилометраж" AS "Километраж до В"  
FROM   
 public."СтанцияНаМаршруте" sm\_start  
JOIN   
 public."СтанцияНаМаршруте" sm\_end ON sm\_start."ВариантРасписания" = sm\_end."ВариантРасписания"  
JOIN   
 public."Маршрут" m ON sm\_start."ВариантРасписания" = m."ВариантРасписания"  
JOIN   
 public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
WHERE   
 sm\_start."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND sm\_end."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND sm\_start."ПройденныйКилометраж" < sm\_end."ПройденныйКилометраж"  
 AND sm\_start."ВремяОтправления" BETWEEN %s AND %s  
ORDER BY   
 sm\_start."ВремяОтправления" ASC  
LIMIT 1;

**Описание:**

1. SELECT  
   Запрашивает следующие поля:
   * p."НомерПоезда" AS "Номер поезда" - номер поезда.
   * p."КатегорияПоезда" AS "Категория поезда" - категория поезда (например, пассажирский, экспресс).
   * sm\_start."ВремяОтправления" AS "Время отправления" - время отправления поезда с начальной станции.
   * sm\_end."ВремяПрибытия" AS "Время прибытия" - время прибытия поезда на конечную станцию.
   * sm\_start."ПройденныйКилометраж" AS "Километраж от А" - километрож на начальной станции.
   * sm\_end."ПройденныйКилометраж" AS "Километраж до В" - километрож на конечной станции.
2. FROM public."СтанцияНаМаршруте" sm\_start  
   Основная таблица, в которой указаны данные о станциях на маршруте поезда для начальной станции:
   * "СтанцияНаМаршруте" - таблица с данными о станциях на маршруте.
   * sm\_start - псевдоним для таблицы начальной станции.
3. JOIN public."СтанцияНаМаршруте" sm\_end ON sm\_start."ВариантРасписания" = sm\_end."ВариантРасписания"  
   Соединяет таблицу начальной станции с таблицей конечной станции на маршруте:
   * Связь через поле "ВариантРасписания".
   * Это позволяет сопоставить начальную и конечную станции на маршруте.
4. JOIN public."Маршрут" m ON sm\_start."ВариантРасписания" = m."ВариантРасписания"  
   Соединяет таблицы станций с таблицей маршрута:
   * Связь через поле "ВариантРасписания".
   * Это позволяет получить данные о маршруте, соответствующие станциям.
5. JOIN public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
   Соединяет таблицу маршрутов с таблицей поездов:
   * Связь через поле "Маршрут" в таблице поездов и "КодМаршрута" в таблице маршрутов.
   * Это позволяет получить информацию о поезде для данного маршрута.
6. WHERE  
   Указывает условия фильтрации записей:
   * sm\_start."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s) - начальная станция маршрута должна быть равна указанной станции.
   * sm\_end."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s) - конечная станция маршрута должна быть равна указанной станции.
   * sm\_start."ПройденныйКилометраж" < sm\_end."ПройденныйКилометраж" - километрож на начальной станции должен быть меньше, чем на конечной.
   * sm\_start."ВремяОтправления" BETWEEN %s AND %s - время отправления поезда с начальной станции должно быть в пределах указанного промежутка.
7. ORDER BY  
   Сортирует результаты:
   * sm\_start."ВремяОтправления" ASC - по времени отправления с начальной станции в порядке возрастания.
8. LIMIT 1  
   Ограничивает количество возвращаемых строк только одной записью.

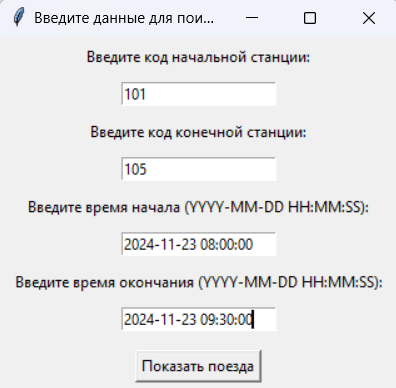


Рисунок 27 – Окно для ввода дополнительных параметров в запрос



Рисунок 28 – Результат одиннадцатого дополнительного запроса из предметной области

**12. Изменить тип станции в классификаторе станций**

UPDATE public."СправочникСтанций"  
SET "ТипСтанции" = %s  
WHERE "НазваниеСтанции" = %s  
RETURNING \*;

**Описание:**

1. UPDATE public."СправочникСтанций"  
   Запрос выполняет обновление данных в таблице "СправочникСтанций", которая хранит информацию о станциях.
2. SET "ТипСтанции" = %s  
   В данном запросе обновляется поле "ТипСтанции" на значение, переданное как параметр %s. Это может быть, например, тип станции, такой как "пассажирская", "грузовая" и т. д.
3. WHERE "НазваниеСтанции" = %s  
   Обновление будет применяться только к строкам, где поле "НазваниеСтанции" соответствует значению, переданному как второй параметр %s. Это означает, что будет обновлён тип станции для конкретной станции с указанным названием.
4. RETURNING \*  
   После выполнения обновления, запрос возвращает все поля обновлённых строк. Это позволяет увидеть все данные, которые были изменены в таблице. Возвращаемые данные будут включать все поля из строки, включая "ТипСтанции", которое было изменено.

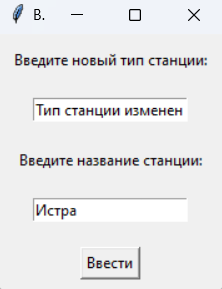


Рисунок 29 – Окно для ввода дополнительных параметров в запрос



Рисунок 30 – Результат двенадцатого дополнительного запроса из предметной области

**13. Добавить нового перевозчика**

INSERT INTO public."Перевозчик" ("КодПеревозчика", "ПолноеНазвание", "СокращенноеНазвание", "ПринадлежностьГосударству")  
VALUES (%s, %s, %s, %s)  
RETURNING \*;

**Описание:**

1. INSERT INTO public."Перевозчик"  
   Запрос выполняет вставку данных в таблицу "Перевозчик", которая, вероятно, хранит информацию о различных перевозчиках.
2. ("КодПеревозчика", "ПолноеНазвание", "СокращенноеНазвание", "ПринадлежностьГосударству")  
   Это список столбцов таблицы, в которые будут вставлены значения. Столбцы следующие:
   * "КодПеревозчика" - уникальный идентификатор перевозчика.
   * "ПолноеНазвание" - полное название перевозчика.
   * "СокращенноеНазвание" - сокращённое название перевозчика.
   * "ПринадлежностьГосударству" - информация о принадлежности перевозчика государству (например, государственная или частная собственность).
3. VALUES (%s, %s, %s, %s)  
   Это место для вставки значений в указанные столбцы. Вместо %s будут подставлены параметры. Эти параметры будут переданы в запрос при его выполнении, например:
   * первый %s - значение для поля "КодПеревозчика".
   * второй %s - значение для поля "ПолноеНазвание".
   * третий %s - значение для поля "СокращенноеНазвание".
   * четвёртый %s - значение для поля "ПринадлежностьГосударству".
4. RETURNING \*  
   Этот оператор указывает, что после выполнения операции вставки запрос должен вернуть все столбцы вставленной строки. Это полезно, чтобы подтвердить вставку и получить полные данные, включая сгенерированный уникальный идентификатор (если он присутствует в базе данных, например, для "КодПеревозчика").

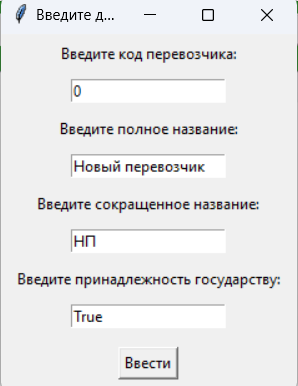


Рисунок 31 – Окно для ввода дополнительных параметров в запрос



Рисунок 32 – Результат тринадцатого дополнительного запроса из предметной области

**14. Удалить из БД расписания всю устаревшую информацию прошедшего графика (по году графика)**

DELETE FROM public."Поезд"  
WHERE "ГодГрафика" = %s  
RETURNING \*;

DELETE FROM public."КалендарьДвиженияПоездов"  
WHERE "ГодГрафика" = %s;

1. Операция удаления из таблицы "Поезд"

DELETE FROM public."Поезд"

WHERE "ГодГрафика" = %s

RETURNING \*;

* DELETE FROM public."Поезд" - удаляет строки из таблицы "Поезд", которая содержит информацию о поездах.
* WHERE "ГодГрафика" = %s - условие удаления: строки будут удаляться, если значение столбца "ГодГрафика" совпадает с переданным параметром %s. Этот параметр будет заменён на конкретный год графика при выполнении запроса.
* RETURNING \* - возвращает все столбцы для каждой удалённой строки. Это позволяет получить информацию о всех поездах, которые были удалены из таблицы. Такой подход полезен для проверки и получения данных о том, какие именно строки были удалены.

2. Операция удаления из таблицы "КалендарьДвиженияПоездов"

DELETE FROM public."КалендарьДвиженияПоездов"

WHERE "ГодГрафика" = %s;

* DELETE FROM public."КалендарьДвиженияПоездов" - удаляет строки из таблицы "КалендарьДвиженияПоездов", которая, вероятно, содержит данные о календарях движения поездов (например, расписания поезда по годам).
* WHERE "ГодГрафика" = %s - условие удаления: строки будут удаляться, если значение столбца "ГодГрафика" совпадает с переданным параметром %s. Этот параметр также будет заменён на конкретный год графика при выполнении запроса.

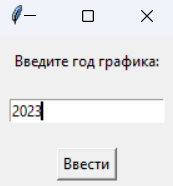


Рисунок 33 – Окно для ввода дополнительных параметров в запрос



Рисунок 34 – Результат четырнадцатого дополнительного запроса из предметной области

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПО РАБОТЕ С ПРИЛОЖЕНИЕМ**

Программа предусматривает работу с базой данных, в первом окне можно выбрать любую таблицу и нажать кнопку загрузить, после чего выбранная таблица отобразиться в программе:

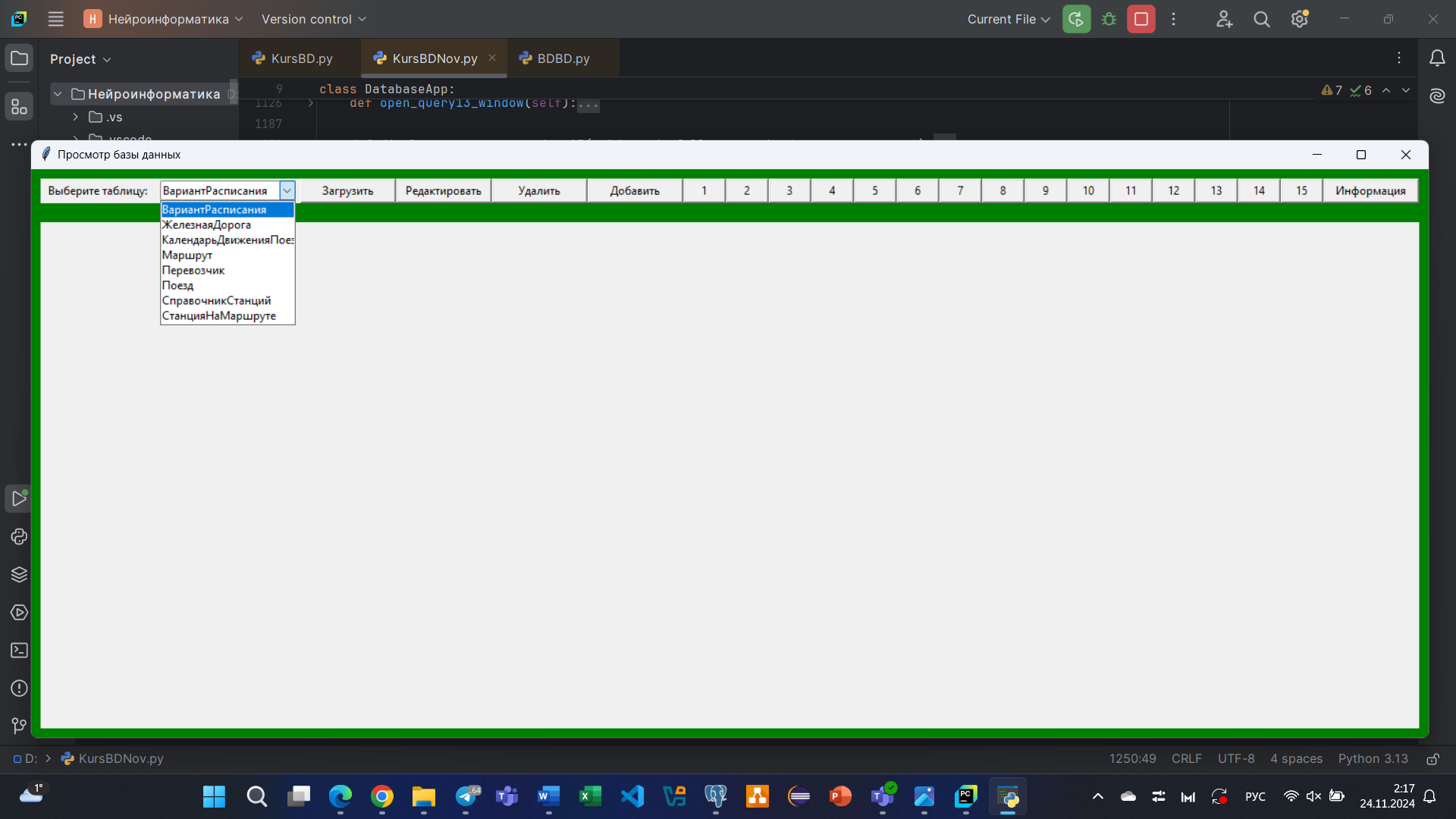


Рисунок 35 – Окно программы с выбором необходимой таблицы

Пример отображения таблицы Перевозчик:

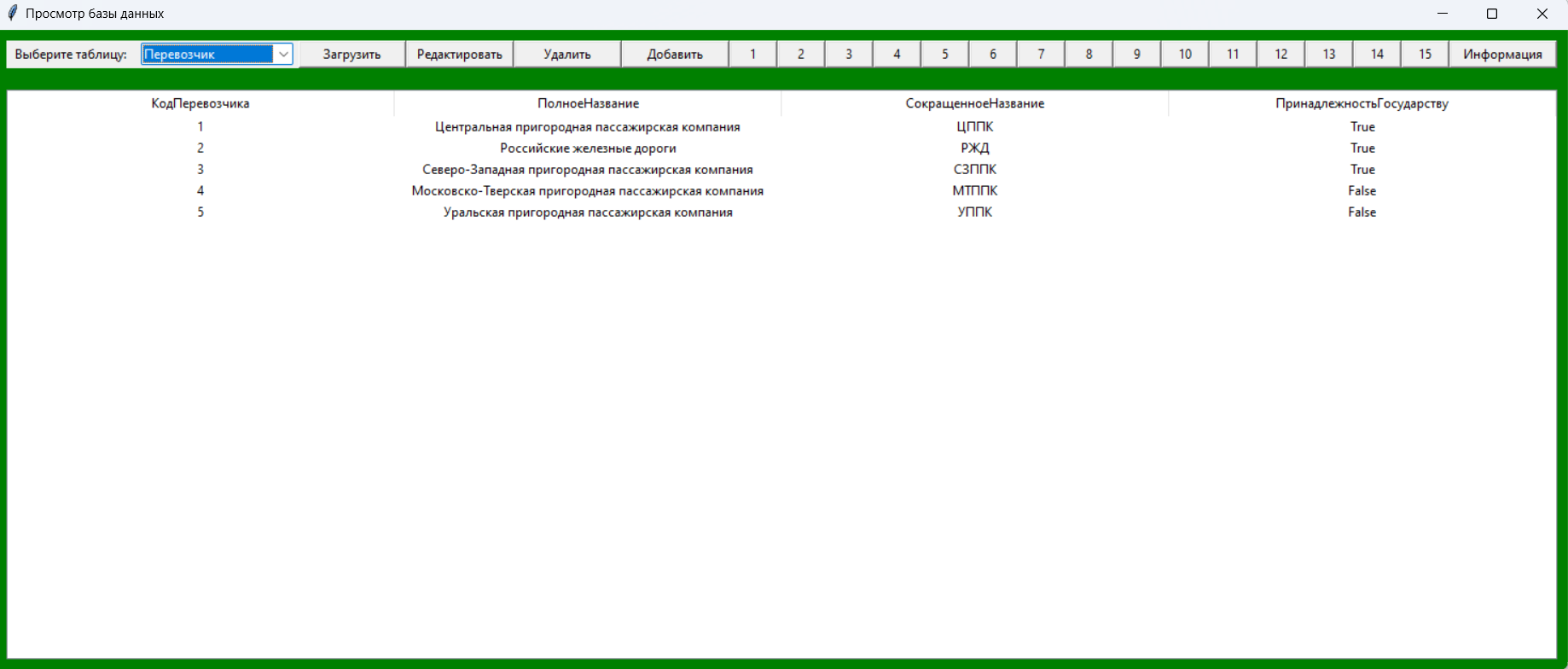


Рисунок 36 – Таблица Перевозчик

В каждую таблицу можно добавить данные, нажав на кнопку Добавить:

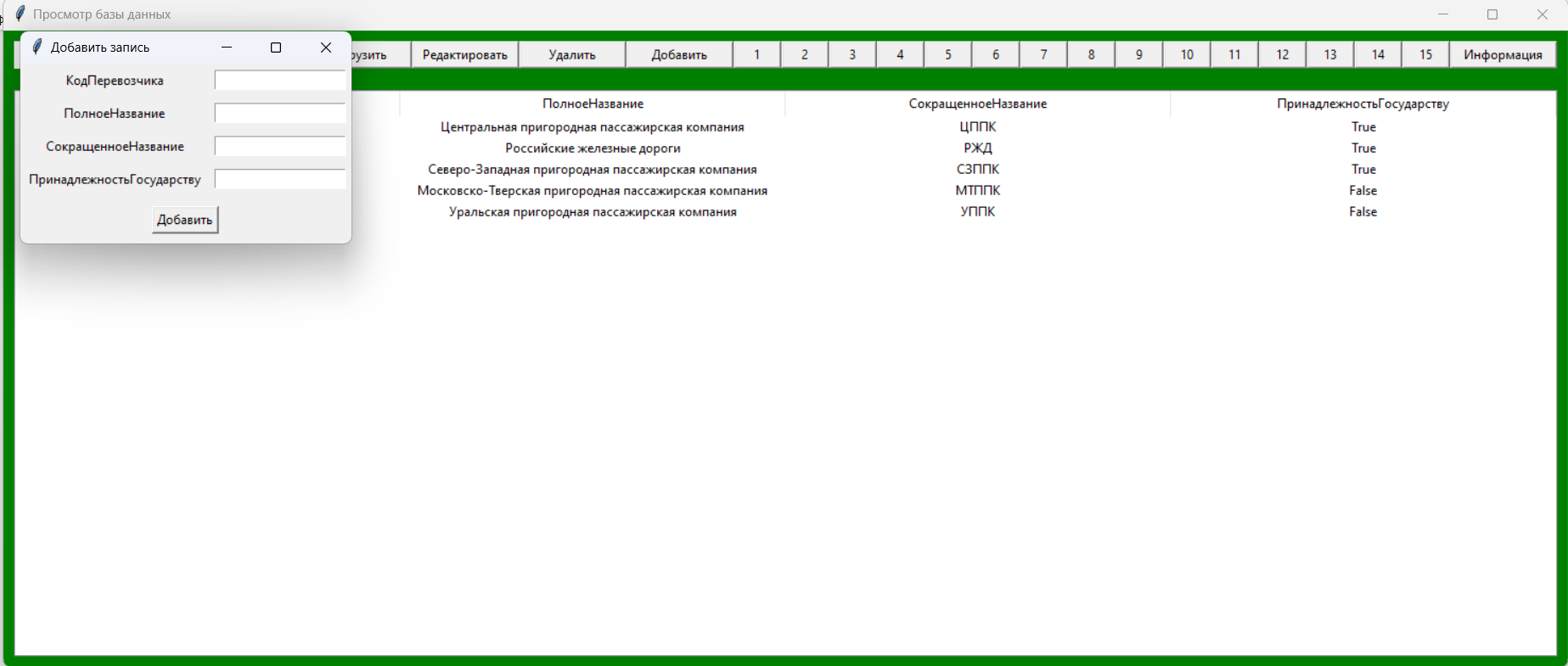


Рисунок 37– Окно добавления

Из каждой таблицы можно удалить строку по нажатию кнопку Удалить, предварительно выбрав строчку, программа спросит, уверен ли пользователь удалить запись :

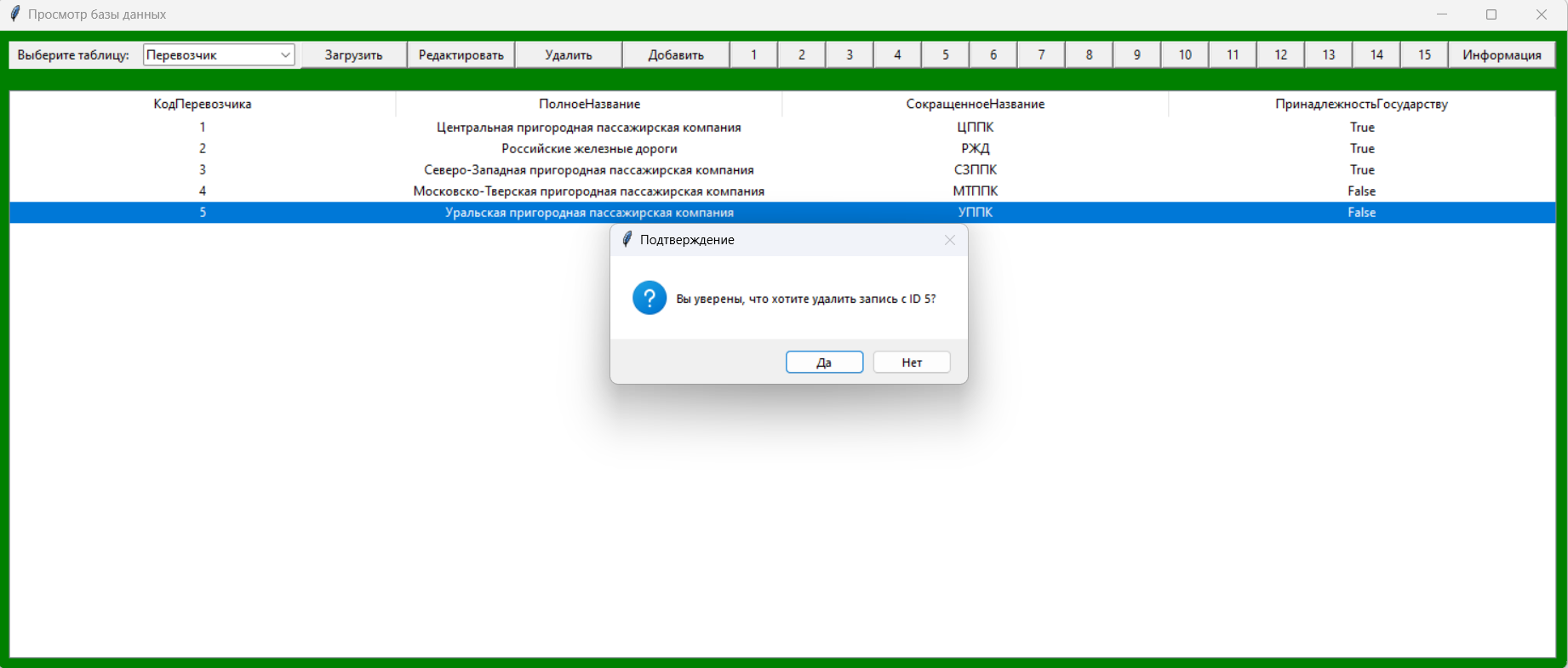


Рисунок 38 – Окно удаления

В каждой таблице можно редактировать данные по нажатию кнопки Редактировать, предварительно выбрав строку:

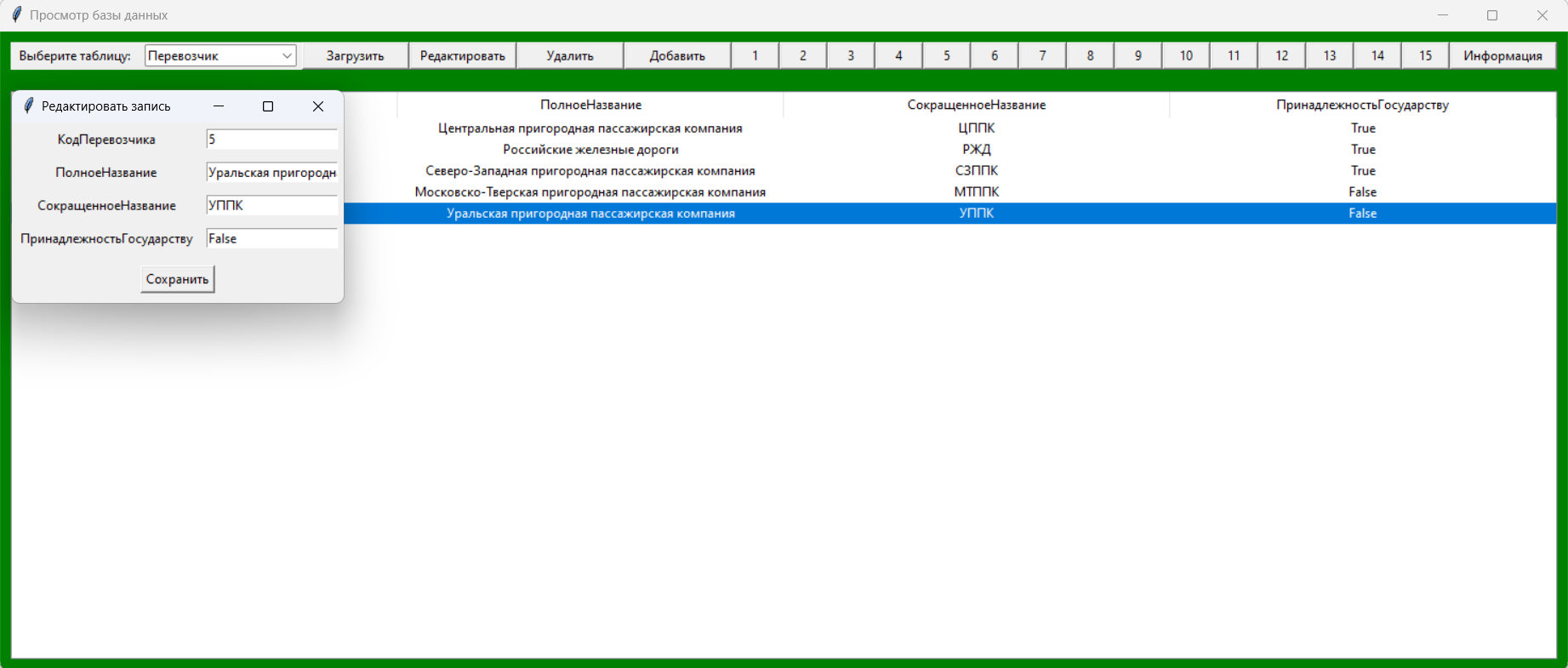


Рисунок 39 – Окно редактирования

Программа предоставляет возможность выполнения 14 дополнительных запросов из предметной области, для этого в программе есть кнопки с 1 по 14. Информацию о назначении каждой кнопки можно узнать, нажав на кнопку Информация:

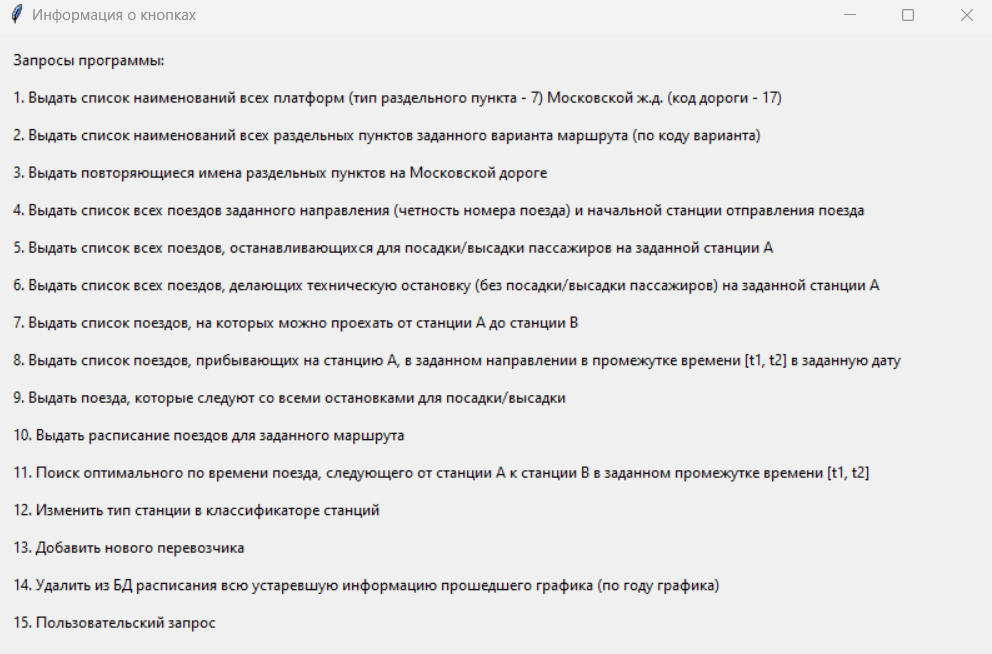


Рисунок 40 – Окно информации о запросах

Также в программе реализована возможность ввода пользовательского запроса - кнопка с номером 15. При нажатии на нее открывается текстовый файл BDfile.txt, в котором пользователь может ввести свой запрос. Далее необходимо сохранить введенный запрос нажатием сочетания клавиш Ctrl+S или любым другим способом и обязательно закрыть текстовый документ с помощью крестика в правом углу окна:

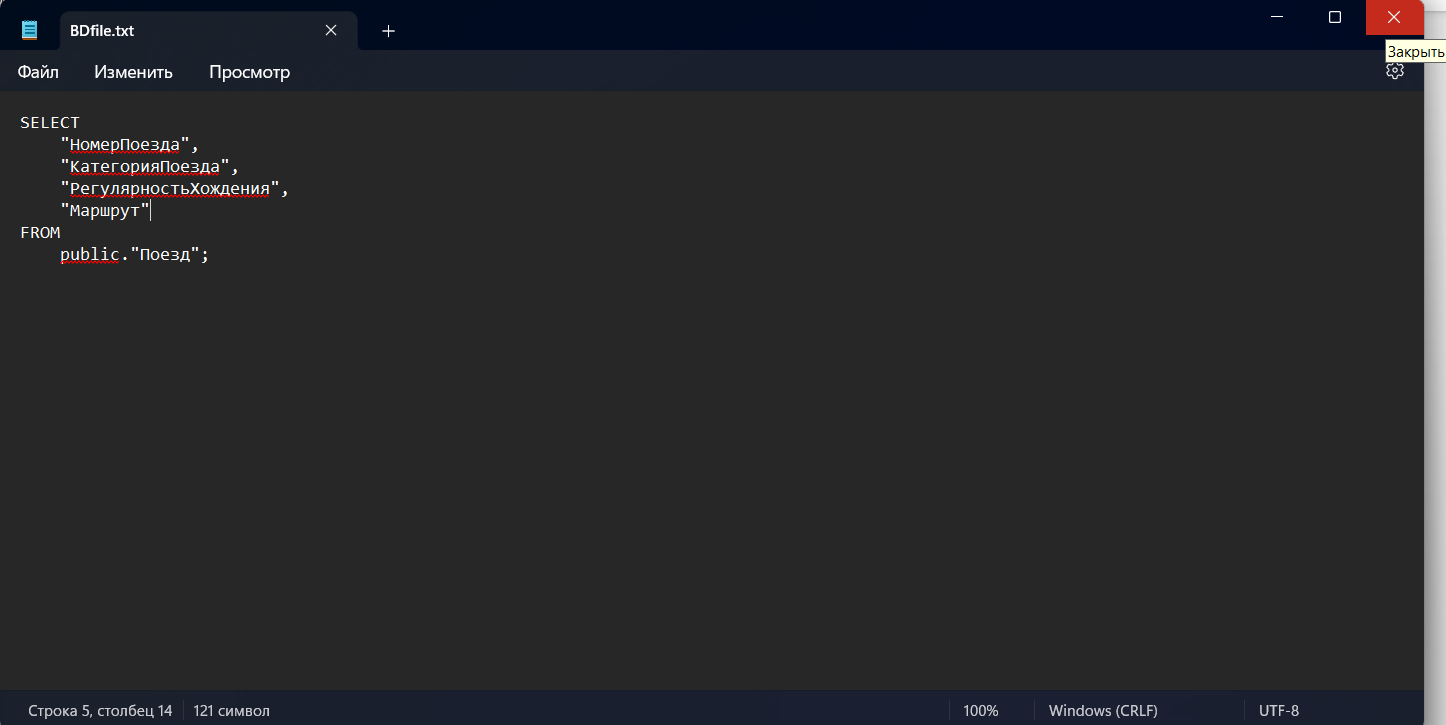


Рисунок 41 – Текстовый файл для ввода пользовательского запроса

При выборе некоторых дополнительных запросов из предметной области могут понадобится дополнительные данные для исполнения запроса, их нужно ввести в всплывающее окно, которое появляется после нажатия книпки запроса:

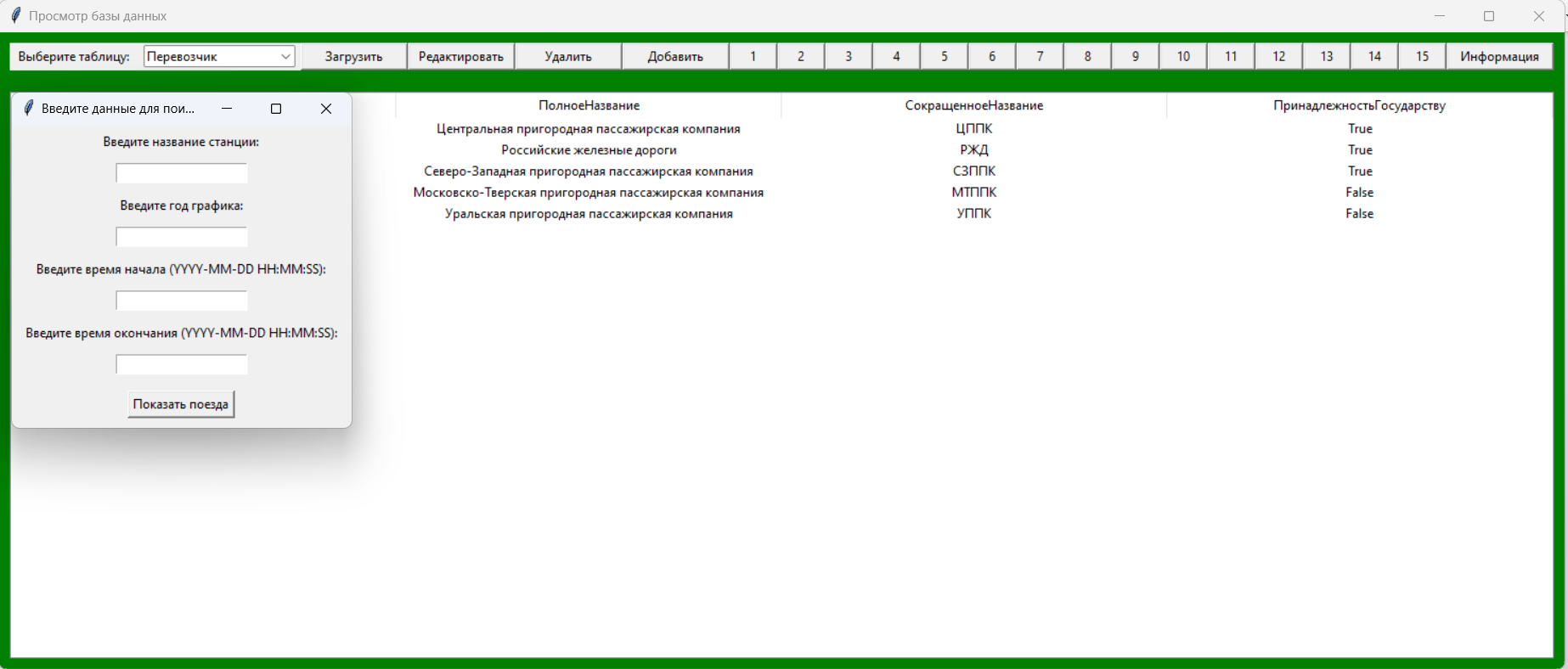


Рисунок 42 – Окно для ввода дополнительных параметров в запрос

# **ИСХОДНЫЙ КОД ПРОГРАММЫ**

import tkinter as tk  
from tkinter import ttk, messagebox, simpledialog  
import psycopg  
import subprocess  
import time  
import os  
from PIL import Image, ImageTk  
  
class DatabaseApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Просмотр базы данных")  
 self.root.geometry("1474x600")  
 root.configure(bg="green") # Установка цвета фона окна  
  
 self.frame\_controls = tk.Frame(self.root)  
 self.frame\_controls.pack(side=tk.TOP, fill=tk.X, padx=10, pady=10)  
  
 self.frame\_table = tk.Frame(self.root)  
 self.frame\_table.pack(side=tk.TOP, fill=tk.BOTH, expand=True, padx=10, pady=10)  
  
 self.label\_table = tk.Label(self.frame\_controls, text="Выберите таблицу:")  
 self.label\_table.pack(side=tk.LEFT, padx=5)  
  
 self.tables = [  
 "ВариантРасписания",  
 "ЖелезнаяДорога",  
 "КалендарьДвиженияПоездов",  
 "Маршрут",  
 "Перевозчик",  
 "Поезд",  
 "СправочникСтанций",  
 "СтанцияНаМаршруте"  
 ]  
  
 self.table\_var = tk.StringVar(value=self.tables[0])  
 self.table\_menu = ttk.Combobox(self.frame\_controls, textvariable=self.table\_var, values=self.tables, state="readonly")  
 self.table\_menu.pack(side=tk.LEFT, padx=5)  
  
 button\_width = 13   
  
 self.button\_load = tk.Button(self.frame\_controls, text="Загрузить", command=self.load\_table, width=button\_width)  
 self.button\_load.pack(side=tk.LEFT)  
  
 self.button\_edit = tk.Button(self.frame\_controls, text="Редактировать", command=self.edit\_record, width=button\_width)  
 self.button\_edit.pack(side=tk.LEFT)  
  
 self.button\_delete = tk.Button(self.frame\_controls, text="Удалить", command=self.delete\_record, width=button\_width)  
 self.button\_delete.pack(side=tk.LEFT)  
  
 self.button\_add = tk.Button(self.frame\_controls, text="Добавить", command=self.add\_record, width=button\_width)  
 self.button\_add.pack(side=tk.LEFT)  
  
 self.button\_custom\_query1 = tk.Button(self.frame\_controls, text="1", command=self.display\_custom\_query\_data1, width=5)  
 self.button\_custom\_query1.pack(side=tk.LEFT)  
  
 self.button\_custom\_query2 = tk.Button(self.frame\_controls, text="2", command=self.open\_query2\_window, width=5)  
 self.button\_custom\_query2.pack(side=tk.LEFT)  
  
 self.button\_custom\_query3 = tk.Button(self.frame\_controls, text="3", command=self.display\_custom\_query\_data3, width=5)  
 self.button\_custom\_query3.pack(side=tk.LEFT)  
  
 self.button\_custom\_query4 = tk.Button(self.frame\_controls, text="4", command=self.display\_custom\_query\_data4, width=5)  
 self.button\_custom\_query4.pack(side=tk.LEFT)  
  
 self.button\_custom\_query5 = tk.Button(self.frame\_controls, text="5", command=self.open\_query5\_window, width=5)  
 self.button\_custom\_query5.pack(side=tk.LEFT)  
  
 self.button\_custom\_query6 = tk.Button(self.frame\_controls, text="6", command=self.open\_query6\_window, width=5)  
 self.button\_custom\_query6.pack(side=tk.LEFT)  
  
 self.button\_custom\_query7 = tk.Button(self.frame\_controls, text="7", command=self.open\_query7\_window, width=5)  
 self.button\_custom\_query7.pack(side=tk.LEFT)  
  
 self.button\_custom\_query8 = tk.Button(self.frame\_controls, text="8", command=self.open\_query8\_window, width=5)  
 self.button\_custom\_query8.pack(side=tk.LEFT)  
  
 self.button\_custom\_query9 = tk.Button(self.frame\_controls, text="9", command=self.display\_custom\_query\_data9, width=5)  
 self.button\_custom\_query9.pack(side=tk.LEFT)  
  
 self.button\_custom\_query10 = tk.Button(self.frame\_controls, text="10", command=self.open\_query10\_window, width=5)  
 self.button\_custom\_query10.pack(side=tk.LEFT)  
  
 self.button\_custom\_query11 = tk.Button(self.frame\_controls, text="11", command=self.open\_query11\_window, width=5)  
 self.button\_custom\_query11.pack(side=tk.LEFT)  
  
 self.button\_custom\_query12 = tk.Button(self.frame\_controls, text="12", command=self.open\_query12\_window, width=5)  
 self.button\_custom\_query12.pack(side=tk.LEFT)  
  
 self.button\_custom\_query13 = tk.Button(self.frame\_controls, text="13", command=self.open\_query13\_window, width=5)  
 self.button\_custom\_query13.pack(side=tk.LEFT)  
  
 self.button\_custom\_query14 = tk.Button(self.frame\_controls, text="14", command=self.open\_query14\_window, width=5)  
 self.button\_custom\_query14.pack(side=tk.LEFT)  
  
 self.button\_custom\_query15 = tk.Button(self.frame\_controls, text="15", command=self.execute\_custom\_query\_from\_file, width=5)  
 self.button\_custom\_query15.pack(side=tk.LEFT)  
  
 self.info\_button = tk.Button(self.frame\_controls, text="Информация",command=self.show\_info\_window, width=button\_width)  
 self.info\_button.pack(side=tk.LEFT)  
  
 def connect\_to\_db(self):  
 try:  
 conn = psycopg.connect(  
 dbname="BDNew",  
 user="postgres",  
 password="12345",  
 host="localhost",  
 port="5432"  
 )  
 return conn  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка подключения", f"Не удалось подключиться к базе данных: {e}")  
 return None  
  
 def load\_table\_data(self, table\_name):  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(f"SELECT \* FROM public.\"{table\_name}\"")  
 rows = cursor.fetchall()  
 colnames = [desc[0] for desc in cursor.description]  
 return colnames, rows  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось загрузить данные из таблицы {table\_name}: {e}")  
 return None, None  
 finally:  
 conn.close()  
  
 def display\_data(self, table\_name):  
 columns, rows = self.load\_table\_data(table\_name)  
 if columns is None or rows is None:  
 return  
  
 self.clear\_table\_frame()  
 self.current\_table = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 self.current\_table.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 self.current\_table.heading(col, text=col)  
 self.current\_table.column(col, width=150, anchor = "center")  
  
 for row in rows:  
 self.current\_table.insert("", tk.END, values=row)  
  
 self.current\_columns = columns   
  
 def load\_table(self):  
 table\_name = self.table\_var.get()  
 self.display\_data(table\_name)  
  
 def edit\_record(self):  
 if not self.current\_table or not self.current\_table.selection():  
 messagebox.showwarning("Выбор записи", "Пожалуйста, выберите запись для редактирования.")  
 return  
  
 selected\_item = self.current\_table.selection()[0]   
 record = self.current\_table.item(selected\_item)["values"]  
  
 edit\_window = tk.Toplevel(self.root)  
 edit\_window.title(f"Редактировать запись")  
  
 entries = []  
 for i, value in enumerate(record):  
 label = tk.Label(edit\_window, text=self.current\_columns[i])  
 label.grid(row=i, column=0, padx=5, pady=5)  
 entry = tk.Entry(edit\_window)  
 entry.insert(0, value)  
 entry.grid(row=i, column=1, padx=5, pady=5)  
 entries.append(entry)  
  
 def save\_record():  
 new\_values = [entry.get() for entry in entries]  
 set\_clause = ", ".join([f"{col} = %s" for col in self.current\_columns])   
 params = new\_values + [record[0]]   
  
 try:  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
 cursor = conn.cursor()  
 query = f"UPDATE public.\"{self.table\_var.get()}\" SET {set\_clause} WHERE {self.current\_columns[0]} = %s"  
 cursor.execute(query, params)  
 conn.commit()  
 messagebox.showinfo("Успех", "Запись обновлена успешно")  
 conn.close()  
 edit\_window.destroy()  
 self.display\_data(self.table\_var.get())   
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось обновить запись: {e}")  
  
 save\_button = tk.Button(edit\_window, text="Сохранить", command=save\_record)  
 save\_button.grid(row=len(record), column=0, columnspan=2, pady=10)  
  
 def delete\_record(self):  
 if not self.current\_table or not self.current\_table.selection():  
 messagebox.showwarning("Выбор записи", "Пожалуйста, выберите запись для удаления.")  
 return  
  
 selected\_item = self.current\_table.selection()[0]  
 record = self.current\_table.item(selected\_item)["values"]  
 record\_id = record[0]  
  
 confirm = messagebox.askyesno("Подтверждение", f"Вы уверены, что хотите удалить запись с ID {record\_id}?")  
 if confirm:  
 try:  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
 cursor = conn.cursor()  
 query = f"DELETE FROM public.\"{self.table\_var.get()}\" WHERE {self.current\_columns[0]} = %s"  
 cursor.execute(query, (record\_id,))  
 conn.commit()  
 messagebox.showinfo("Успех", "Запись удалена успешно")  
 conn.close()  
 self.display\_data(self.table\_var.get())   
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось удалить запись: {e}")  
  
 def add\_record(self):  
 if not self.current\_columns:  
 messagebox.showwarning("Ошибка", "Таблица не загружена. Пожалуйста, загрузите таблицу.")  
 return  
  
 add\_window = tk.Toplevel(self.root)  
 add\_window.title(f"Добавить запись")  
  
 entries = []  
 for col in self.current\_columns:  
 label = tk.Label(add\_window, text=col)  
 label.grid(row=len(entries), column=0, padx=5, pady=5)  
 entry = tk.Entry(add\_window)  
 entry.grid(row=len(entries), column=1, padx=5, pady=5)  
 entries.append(entry)  
  
 def save\_new\_record():  
 new\_values = [entry.get() for entry in entries]  
 placeholders = ", ".join(["%s"] \* len(self.current\_columns))  
 query = f"INSERT INTO public.\"{self.table\_var.get()}\" ({', '.join(self.current\_columns)}) VALUES ({placeholders})"  
  
 try:  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
 cursor = conn.cursor()  
 cursor.execute(query, new\_values)  
 conn.commit()  
 messagebox.showinfo("Успех", "Запись добавлена успешно")  
 conn.close()  
 add\_window.destroy()  
 self.display\_data(self.table\_var.get   
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось добавить запись: {e}")  
  
 save\_button = tk.Button(add\_window, text="Добавить", command=save\_new\_record)  
 save\_button.grid(row=len(self.current\_columns), column=0, columnspan=2, pady=10)  
  
 def clear\_table\_frame(self):  
 for widget in self.frame\_table.winfo\_children():  
 widget.destroy()  
  
 def display\_custom\_query\_data1(self):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT "НазваниеСтанции", "ТипСтанции", "ЖелезнаяДорога"  
 FROM public."СправочникСтанций"  
 WHERE "ТипСтанции" = 'Платформа' AND "ЖелезнаяДорога" = 17;  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query)  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query2\_window(self):  
 query2\_window = tk.Toplevel(self.root)  
 query2\_window.title("Введите код маршрута")  
  
 label = tk.Label(query2\_window, text="Введите код маршрута:")  
 label.pack(padx=10, pady=10)  
  
 entry = tk.Entry(query2\_window)  
 entry.pack(padx=10, pady=10)  
  
 def execute\_query():  
 route\_code = entry.get().strip()  
 if not route\_code.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректный код маршрута.")  
 return  
  
 # Передаем код маршрута в метод display\_custom\_query\_data2  
 self.display\_custom\_query\_data2(route\_code)  
 query2\_window.destroy()  
  
 button = tk.Button(query2\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data2(self, route\_code):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT s."НазваниеСтанции", m."КодМаршрута"  
 FROM public."СправочникСтанций" s  
 JOIN public."СтанцияНаМаршруте" sn ON s."КодСтанции" = sn."КодСтанции"  
 JOIN public."Маршрут" m ON sn."ВариантРасписания" = m."ВариантРасписания"  
 WHERE m."КодМаршрута" = %s;  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (route\_code,))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def display\_custom\_query\_data3(self):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT   
 s."НазваниеСтанции",   
 COUNT(\*) AS "Количество"  
 FROM   
 public."СправочникСтанций" s  
 JOIN   
 public."ЖелезнаяДорога" j ON s."ЖелезнаяДорога" = j."КодЖелезнойДороги"  
 WHERE   
 j."ПолноеНазвание" LIKE '%Московская железная дорога%' -- Фильтруем по названию железной дороги  
 GROUP BY   
 s."НазваниеСтанции"  
 HAVING   
 COUNT(\*) > 1; -- Выбираем только те станции, которые встречаются более одного раза  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query)  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def display\_custom\_query\_data4(self):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT   
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 s."НазваниеСтанции" AS "НачальнаяСтанция",  
 m."ВремяОтправления",  
 m."ВремяПрибытия"  
 FROM   
 public."Поезд" p  
 JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
 JOIN   
 public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
 WHERE   
 p."НомерПоезда" % 2 = 0 -- Только четные номера поездов  
 ORDER BY   
 p."НомерПоезда";  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query)  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query5\_window(self):  
 query5\_window = tk.Toplevel(self.root)  
 query5\_window.title("Введите название станции:")  
  
 label = tk.Label(query5\_window, text="Введите название станции:")  
 label.pack(padx=10, pady=10)  
  
 entry = tk.Entry(query5\_window)  
 entry.pack(padx=10, pady=10)  
  
 def execute\_query():  
 station\_name = entry.get().strip()  
 if not station\_name:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректный код маршрута.")  
 return  
  
 # Передаем код маршрута в метод display\_custom\_query\_data2  
 self.display\_custom\_query\_data5(station\_name)  
 query5\_window.destroy()  
  
 button = tk.Button(query5\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data5(self, station\_name):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT   
 p."НомерПоезда",   
 p."КатегорияПоезда",   
 s."НазваниеСтанции",   
 snm."ВремяПрибытия",   
 snm."ВремяОтправления",   
 snm."ПризнакРазрешенияПосадкиВысадки"   
 FROM   
 public."Поезд" p   
 JOIN   
 public."Маршрут" m   
 ON p."Маршрут" = m."КодМаршрута"   
 JOIN   
 public."СтанцияНаМаршруте" snm   
 ON m."ВариантРасписания" = snm."ВариантРасписания"   
 JOIN   
 public."СправочникСтанций" s   
 ON snm."КодСтанции" = s."КодСтанции"   
 WHERE   
 s."НазваниеСтанции" = %s   
 AND snm."ПризнакРазрешенияПосадкиВысадки" = TRUE   
 ORDER BY   
 p."НомерПоезда", snm."ВремяПрибытия";  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name,))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query6\_window(self):  
 query6\_window = tk.Toplevel(self.root)  
 query6\_window.title("Введите название станции:")  
  
 label = tk.Label(query6\_window, text="Введите название станции:")  
 label.pack(padx=10, pady=10)  
  
 entry = tk.Entry(query6\_window)  
 entry.pack(padx=10, pady=10)  
  
 def execute\_query():  
 station\_name1 = entry.get().strip()  
 if not station\_name1:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректный код маршрута.")  
 return  
  
 self.display\_custom\_query\_data6(station\_name1)  
 query6\_window.destroy()  
  
 button = tk.Button(query6\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data6(self, station\_name1):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT   
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 s."НазваниеСтанции",  
 snm."ВремяПрибытия",  
 snm."ВремяОтправления",  
 snm."ПризнакТехническойОстановки",  
 snm."ПризнакРазрешенияПосадкиВысадки"  
 FROM   
 public."Поезд" p  
 JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
 JOIN   
 public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
 JOIN   
 public."СправочникСтанций" s ON snm."КодСтанции" = s."КодСтанции"  
 WHERE   
 s."НазваниеСтанции" = %s  
 AND snm."ПризнакТехническойОстановки" = TRUE  
 AND snm."ПризнакРазрешенияПосадкиВысадки" = FALSE;  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name1,))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query7\_window(self):  
 query7\_window = tk.Toplevel(self.root) # Исправлено на query7\_window  
 query7\_window.title("Введите коды начальной и конечной станций:")  
  
 label = tk.Label(query7\_window, text="Введите коды начальной и конечной станций:")  
 label.pack(padx=10, pady=10)  
  
 entry1 = tk.Entry(query7\_window)  
 entry1.pack(padx=10, pady=10)  
  
 entry2 = tk.Entry(query7\_window)  
 entry2.pack(padx=10, pady=10)  
  
 def execute\_query():  
 station\_name1 = entry1.get().strip()  
 station\_name2 = entry2.get().strip()  
  
 if not station\_name1 or not station\_name2:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите начальную и конечную станции.")  
 return  
  
 self.display\_custom\_query\_data7(station\_name1, station\_name2)  
 query7\_window.destroy()  
  
 button = tk.Button(query7\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data7(self, station\_name1, station\_name2):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция"  
 FROM   
 public."Поезд" p  
 JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
 WHERE   
 m."НачальнаяСтанция" = %s  
 AND m."КонечнаяСтанция" = %s  
 ORDER BY   
 p."НомерПоезда";  
 """   
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name1, station\_name2))   
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query8\_window(self):  
 query8\_window = tk.Toplevel(self.root)  
 query8\_window.title("Введите данные для поиска поездов:")  
  
 label\_station = tk.Label(query8\_window, text="Введите название станции:")  
 label\_station.pack(padx=10, pady=5)  
  
 entry\_station = tk.Entry(query8\_window)  
 entry\_station.pack(padx=10, pady=5)  
  
 label\_date = tk.Label(query8\_window, text="Введите год графика:")  
 label\_date.pack(padx=10, pady=5)  
  
 entry\_date = tk.Entry(query8\_window)  
 entry\_date.pack(padx=10, pady=5)  
  
 label\_time\_start = tk.Label(query8\_window, text="Введите время начала (YYYY-MM-DD HH:MM:SS):")  
 label\_time\_start.pack(padx=10, pady=5)  
  
 entry\_time\_start = tk.Entry(query8\_window)  
 entry\_time\_start.pack(padx=10, pady=5)  
  
 label\_time\_end = tk.Label(query8\_window, text="Введите время окончания (YYYY-MM-DD HH:MM:SS):")  
 label\_time\_end.pack(padx=10, pady=5)  
  
 entry\_time\_end = tk.Entry(query8\_window)  
 entry\_time\_end.pack(padx=10, pady=5)  
  
 def execute\_query():  
 # Получаем данные из полей ввода  
 station\_name = entry\_station.get().strip()  
 date = entry\_date.get().strip()  
 time\_start = entry\_time\_start.get().strip()  
 time\_end = entry\_time\_end.get().strip()  
  
 if not station\_name:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную станцию.")  
 return  
  
 if not date.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную дату (YYYY).")  
 return  
  
 if not time\_start or not time\_end:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректное время.")  
 return  
  
 self.display\_custom\_query\_data8(station\_name, date, time\_start, time\_end)  
  
 query8\_window.destroy()  
  
 button = tk.Button(query8\_window, text="Показать поезда", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data8(self, station\_name, date, time\_start, time\_end):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."ВремяПрибытия",  
 m."ВремяОтправления",  
 cd."ГодГрафика",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция"  
 FROM   
 public."Поезд" p  
 JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
 JOIN   
 public."КалендарьДвиженияПоездов" cd ON p."ГодГрафика" = cd."ГодГрафика"  
 JOIN   
 public."СправочникСтанций" s ON m."НачальнаяСтанция" = s."КодСтанции"  
 WHERE   
 s."НазваниеСтанции" = %s  
 AND cd."ГодГрафика" = %s  
 AND m."ВремяПрибытия" BETWEEN %s AND %s  
 ORDER BY   
 m."ВремяПрибытия";  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name, date, time\_start, time\_end))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def display\_custom\_query\_data9(self):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT DISTINCT  
 p."НомерПоезда",  
 p."КатегорияПоезда",  
 p."РегулярностьХождения",  
 m."НачальнаяСтанция",  
 m."КонечнаяСтанция",  
 snm."ПризнакРазрешенияПосадкиВысадки"  
 FROM   
 public."Поезд" p  
 JOIN   
 public."Маршрут" m ON p."Маршрут" = m."КодМаршрута"  
 JOIN   
 public."СтанцияНаМаршруте" snm ON m."ВариантРасписания" = snm."ВариантРасписания"  
 WHERE   
 snm."ПризнакРазрешенияПосадкиВысадки" = TRUE  
 AND NOT EXISTS (  
 SELECT 1  
 FROM public."СтанцияНаМаршруте" s  
 WHERE s."ВариантРасписания" = snm."ВариантРасписания"  
 AND s."ПризнакРазрешенияПосадкиВысадки" = FALSE  
 )  
 ORDER BY   
 p."НомерПоезда";  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query)  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query10\_window(self):  
 query10\_window = tk.Toplevel(self.root) # Исправлено на query7\_window  
 query10\_window.title("Введите коды начальной и конечной станций:")  
  
 label = tk.Label(query10\_window, text="Введите коды начальной и конечной станций:")  
 label.pack(padx=10, pady=10)  
  
 entry1 = tk.Entry(query10\_window)  
 entry1.pack(padx=10, pady=10)  
  
 entry2 = tk.Entry(query10\_window)  
 entry2.pack(padx=10, pady=10)  
  
 def execute\_query():  
 station\_name1 = entry1.get().strip()  
 station\_name2 = entry2.get().strip()  
  
 if not station\_name1 or not station\_name2:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите начальную и конечную станции.")  
 return  
  
 self.display\_custom\_query\_data10(station\_name1, station\_name2)  
 query10\_window.destroy()  
  
 button = tk.Button(query10\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data10(self, station\_name1, station\_name2):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT DISTINCT  
 s."НазваниеСтанции" AS "Станция",  
 p."НомерПоезда" AS "Номер поезда",  
 p."КатегорияПоезда" AS "Категория поезда",  
 sm."ВремяОтправления" AS "Время отправления"  
 FROM   
 public."СтанцияНаМаршруте" sm  
 JOIN   
 public."Маршрут" m ON sm."ВариантРасписания" = m."ВариантРасписания"  
 JOIN   
 public."СправочникСтанций" s ON sm."КодСтанции" = s."КодСтанции"  
 JOIN   
 public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
 WHERE   
 m."НачальнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND m."КонечнаяСтанция" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 ORDER BY   
 p."НомерПоезда", sm."ВремяОтправления";  
 """ # Исправлено на %s для обоих параметров  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name1, station\_name2)) # Все параметры передаем в одном кортеже  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query11\_window(self):  
 query11\_window = tk.Toplevel(self.root)  
 query11\_window.title("Введите данные для поиска поездов:")  
  
 label\_station1 = tk.Label(query11\_window, text="Введите код начальной станции:")  
 label\_station1.pack(padx=10, pady=5)  
  
 entry\_station1 = tk.Entry(query11\_window)  
 entry\_station1.pack(padx=10, pady=5)  
  
 label\_station2 = tk.Label(query11\_window, text="Введите код конечной станции:")  
 label\_station2.pack(padx=10, pady=5)  
  
 entry\_station2 = tk.Entry(query11\_window)  
 entry\_station2.pack(padx=10, pady=5)  
  
 label\_time\_start = tk.Label(query11\_window, text="Введите время начала (YYYY-MM-DD HH:MM:SS):")  
 label\_time\_start.pack(padx=10, pady=5)  
  
 entry\_time\_start = tk.Entry(query11\_window)  
 entry\_time\_start.pack(padx=10, pady=5)  
  
 label\_time\_end = tk.Label(query11\_window, text="Введите время окончания (YYYY-MM-DD HH:MM:SS):")  
 label\_time\_end.pack(padx=10, pady=5)  
  
 entry\_time\_end = tk.Entry(query11\_window)  
 entry\_time\_end.pack(padx=10, pady=5)  
  
 def execute\_query():  
 # Получаем данные из полей ввода  
 station\_name1 = entry\_station1.get().strip()  
 station\_name2 = entry\_station2.get().strip()  
 time\_start = entry\_time\_start.get().strip()  
 time\_end = entry\_time\_end.get().strip()  
  
 if not station\_name1.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную станцию.")  
 return  
  
 if not station\_name2.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную станцию.")  
 return  
  
 if not time\_start or not time\_end:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректное время.")  
 return  
  
 self.display\_custom\_query\_data11(station\_name1, station\_name2, time\_start, time\_end)  
  
 query11\_window.destroy()  
  
 button = tk.Button(query11\_window, text="Показать поезда", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data11(self, station\_name1, station\_name2, time\_start, time\_end):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 SELECT   
 p."НомерПоезда" AS "Номер поезда",  
 p."КатегорияПоезда" AS "Категория поезда",  
 sm\_start."ВремяОтправления" AS "Время отправления",  
 sm\_end."ВремяПрибытия" AS "Время прибытия",  
 sm\_start."ПройденныйКилометраж" AS "Километраж от А",  
 sm\_end."ПройденныйКилометраж" AS "Километраж до В"  
 FROM   
 public."СтанцияНаМаршруте" sm\_start  
 JOIN   
 public."СтанцияНаМаршруте" sm\_end ON sm\_start."ВариантРасписания" = sm\_end."ВариантРасписания"  
 JOIN   
 public."Маршрут" m ON sm\_start."ВариантРасписания" = m."ВариантРасписания"  
 JOIN   
 public."Поезд" p ON p."Маршрут" = m."КодМаршрута"  
 WHERE   
 sm\_start."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND sm\_end."КодСтанции" = (SELECT "КодСтанции" FROM public."СправочникСтанций" WHERE "КодСтанции" = %s)  
 AND sm\_start."ПройденныйКилометраж" < sm\_end."ПройденныйКилометраж"  
 AND sm\_start."ВремяОтправления" BETWEEN %s AND %s  
 ORDER BY   
 sm\_start."ВремяОтправления" ASC  
 LIMIT 1;  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_name1, station\_name2, time\_start, time\_end))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query12\_window(self):  
 query12\_window = tk.Toplevel(self.root) # Исправлено на query7\_window  
 query12\_window.title("Введите тип станции и ее название:")  
  
 label1 = tk.Label(query12\_window, text="Введите новый тип станции:")  
 label1.pack(padx=10, pady=10)  
  
 entry1 = tk.Entry(query12\_window)  
 entry1.pack(padx=10, pady=10)  
  
 label2 = tk.Label(query12\_window, text="Введите название станции:")  
 label2.pack(padx=10, pady=10)  
  
 entry2 = tk.Entry(query12\_window)  
 entry2.pack(padx=10, pady=10)  
  
 def execute\_query():  
 station\_type = entry1.get().strip()  
 station\_name = entry2.get().strip()  
  
 # Проверка на пустой ввод  
 if not station\_type or not station\_name:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите начальную и конечную станции.")  
 return  
  
 self.display\_custom\_query\_data12(station\_type, station\_name)  
 query12\_window.destroy()  
  
 button = tk.Button(query12\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data12(self, station\_type, station\_name):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 UPDATE public."СправочникСтанций"  
 SET "ТипСтанции" = %s  
 WHERE "НазваниеСтанции" = %s  
 RETURNING \*;  
 """ # Исправлено на %s для обоих параметров  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (station\_type, station\_name)) # Все параметры передаем в одном кортеже  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 conn.commit()  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query13\_window(self):  
 # Создаем новое окно для ввода данных  
 query13\_window = tk.Toplevel(self.root)  
 query13\_window.title("Введите данные для добавления нового перевозчика:")  
  
 label\_cod = tk.Label(query13\_window, text="Введите код перевозчика:")  
 label\_cod.pack(padx=10, pady=5)  
  
 entry\_cod = tk.Entry(query13\_window)  
 entry\_cod.pack(padx=10, pady=5)  
  
 label\_full\_name = tk.Label(query13\_window, text="Введите полное название:")  
 label\_full\_name.pack(padx=10, pady=5)  
  
 entry\_full\_name = tk.Entry(query13\_window)  
 entry\_full\_name.pack(padx=10, pady=5)  
  
 label\_short\_name = tk.Label(query13\_window, text="Введите сокращенное название:")  
 label\_short\_name.pack(padx=10, pady=5)  
  
 entry\_short\_name = tk.Entry(query13\_window)  
 entry\_short\_name.pack(padx=10, pady=5)  
  
 label\_goverment = tk.Label(query13\_window, text="Введите принадлежность государству:")  
 label\_goverment.pack(padx=10, pady=5)  
  
 entry\_goverment = tk.Entry(query13\_window)  
 entry\_goverment.pack(padx=10, pady=5)  
  
 def execute\_query():  
 # Получаем данные из полей ввода  
 cod = entry\_cod.get().strip()  
 full\_name = entry\_full\_name.get().strip()  
 short\_name = entry\_short\_name.get().strip()  
 goverment = entry\_goverment.get().strip()  
  
 if not cod.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную станцию.")  
 return  
  
 if not full\_name or not short\_name:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректное время.")  
 return  
  
 if not goverment:  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректную станцию.")  
 return  
  
 self.display\_custom\_query\_data13(cod, full\_name, short\_name, goverment)  
  
 query13\_window.destroy()  
  
 button = tk.Button(query13\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data13(self, cod, full\_name, short\_name, goverment):  
  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 query = """  
 INSERT INTO public."Перевозчик" ("КодПеревозчика", "ПолноеНазвание", "СокращенноеНазвание", "ПринадлежностьГосударству")  
 VALUES (%s, %s, %s, %s)  
 RETURNING \*;  
 """  
  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query, (cod, full\_name, short\_name, goverment))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 conn.commit()  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def open\_query14\_window(self):  
 query14\_window = tk.Toplevel(self.root)  
 query14\_window.title("Введите год графика:")  
  
 label = tk.Label(query14\_window, text="Введите год графика:")  
 label.pack(padx=10, pady=10)  
  
 entry = tk.Entry(query14\_window)  
 entry.pack(padx=10, pady=10)  
  
 def execute\_query():  
 year = entry.get().strip()  
  
 if not year.isdigit():  
 messagebox.showwarning("Неверный ввод", "Пожалуйста, введите корректный год графика.")  
 return  
  
 self.display\_custom\_query\_data14(year)  
 query14\_window.destroy()  
  
 button = tk.Button(query14\_window, text="Ввести", command=execute\_query)  
 button.pack(padx=10, pady=10)  
  
 def display\_custom\_query\_data14(self, year):  
 self.clear\_table\_frame()  
  
 conn = self.connect\_to\_db()  
 if not conn:  
 return  
  
 try:  
 cursor = conn.cursor()  
  
 cursor.execute("""  
 DELETE FROM public."Поезд"  
 WHERE "ГодГрафика" = %s  
 RETURNING \*;  
 """, (year,))  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 cursor.execute("""  
 DELETE FROM public."КалендарьДвиженияПоездов"  
 WHERE "ГодГрафика" = %s;  
 """, (year,))  
  
 conn.commit()  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def execute\_custom\_query\_from\_file(self):  
 file\_path = "D:/BDfile.txt" # Замените на нужный путь  
  
 try:  
 process = subprocess.Popen(['notepad', file\_path])  
  
 process.communicate()  
  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 query = file.read().strip()  
  
 if query:  
 self.execute\_query(query)   
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Произошла ошибка при чтении файла: {e}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")  
  
 def execute\_query(self, query):  
 conn = self.connect\_to\_db()  
 if conn:  
 try:  
 cursor = conn.cursor()  
 cursor.execute(query)  
 rows = cursor.fetchall()  
 columns = [desc[0] for desc in cursor.description]  
  
 conn.commit()  
  
 self.clear\_table\_frame()  
  
 tree = ttk.Treeview(self.frame\_table, columns=columns, show="headings")  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 for col in columns:  
 tree.heading(col, text=col)  
 tree.column(col, width=150)  
  
 for row in rows:  
 tree.insert("", tk.END, values=row)  
  
 except psycopg.Error as e:  
 messagebox.showerror("Ошибка выполнения запроса", f"Не удалось выполнить запрос: {e}")  
 finally:  
 conn.close()  
  
 def show\_info\_window(self):  
 info\_window = tk.Toplevel()  
 info\_window.title("Информация о кнопках")  
 info\_window.geometry("800x500")  
  
 info\_text = (  
 "Запросы программы:\n\n"  
 "1. Выдать список наименований всех платформ (тип раздельного пункта - 7) Московской ж.д. (код дороги - 17)\n\n"  
 "2. Выдать список наименований всех раздельных пунктов заданного варианта маршрута (по коду варианта)\n\n"  
 "3. Выдать повторяющиеся имена раздельных пунктов на Московской дороге\n\n"  
 "4. Выдать список всех поездов заданного направления (четность номера поезда) и начальной станции отправления поезда\n\n"  
 "5. Выдать список всех поездов, останавливающихся для посадки/высадки пассажиров на заданной станции A\n\n"  
 "6. Выдать список всех поездов, делающих техническую остановку (без посадки/высадки пассажиров) на заданной станции A\n\n"  
 "7. Выдать список поездов, на которых можно проехать от станции А до станции B\n\n"  
 "8. Выдать список поездов, прибывающих на станцию А, в заданном направлении в промежутке времени [t1, t2] в заданную дату\n\n"  
 "9. Выдать поезда, которые следуют со всеми остановками для посадки/высадки\n\n"  
 "10. Выдать расписание поездов для заданного маршрута\n\n"  
 "11. Поиск оптимального по времени поезда, следующего от станции А к станции В в заданном промежутке времени [t1, t2]\n\n"  
 "12. Изменить тип станции в классификаторе станций\n\n"  
 "13. Добавить нового перевозчика\n\n"  
 "14. Удалить из БД расписания всю устаревшую информацию прошедшего графика (по году графика)\n\n"  
 "15. Пользовательский запрос\n"  
 )  
  
 info\_label = tk.Label(info\_window, text=info\_text, justify=tk.LEFT, wraplength=750, anchor="nw")  
 info\_label.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = DatabaseApp(root)  
 root.mainloop()