Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Создание приложения для управления базой данных городской транспортной системы

Студент ИУ7-65Б _____ **М. Д.Коротыч**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ **К. А. Кивва**
(Подпись, дата) (И.О.Фамилия)

2022 г.

РЕФЕРАТ

Расчётно-пояснительная записка 40 с., 17 рис., 3 табл., 23 ист.

В работе представлена разработка базы данных городской транспортной системы и приложения для её манипуляции.

Проведена формализация задачи и определён требуемый функционал. Проведён анализ предметной области и описана структура базы данных. Создана и заполнена база данных. Спроектировано приложение для доступа к базе данных. Разработан графический интерфейс приложения. Проведено исследование по зависимости времени ответа база данных от различных типов индексации.

КЛЮЧЕВЫЕ СЛОВА: база данных, PostgreSQL, общественный транспорт, Visual Studio Code, Rust.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Транспортная система является главным элементом современного города, её функционирование во многом определяет удобство жизни городского населения. Общественный транспорт играет в ней немаловажную роль, его удобство и доступность достигается за счёт различной пешеходной и дорожной инфраструктуры. Именно благодаря остановкам, расписаниям, спланированным маршрутам общественный транспорт мобилен, гибок и эффективен.

В XXI веке со стремительным развитием электронно-вычислительной техники — такой как компьютеры и смартфоны — появилась возможность узнать, где в определённый момент едет автобус, или какой маршрут нужен, чтобы доехать до места назначения.

Цель данной работы — реализовать базу данных системы городского общественного транспорта и приложение для удобного просмотра и манипулирования (редактирования, обновления).

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) провести анализ предметной области;
2) формализовать задание, определить необходимый функционал;
3) провести анализ СУБД;
4) описать структуру базы данных, включая объекты, из которых она состоит;
5) создать и заполнить БД;
6) спроектировать и реализовать приложение для доступа к БД;
7) исследовать производительность доступа к базе данных.

1 Аналитический раздел

1.1 Формализация задачи

Главная особенность городского общественного транспорта — наличие маршрута регулярных перевозок. Это предназначенный для осуществления перевозок пассажиров и багажа по расписаниям путь следования транспортных средств от начального остановочного пункта через промежуточные остановочные пункты до конечного остановочного пункта, которые определены в установленном порядке [1].

В крупных городах общественный наземный транспорт представлен 2 видами: безрельсовый маршрутный и рельсовый. К безрельсовому транспорту обычно относят автобусы и троллейбусы. К рельсовому же — трамваи.

Как уже отмечалось ранее, невозможно должное функционирование городского общественного транспорта без соответствующей инфраструктуры. В первую очередь к ней относится **остановка общественного транспорта** — общественное место остановки транспортных средств по маршруту регулярных перевозок, оборудованное для посадки, высадки пассажиров и ожидания транспортных средств. Движение общественного транспорта регламентируется **графиком** (расписанием). Основные исходные данные для составления графика — время оборота по маршруту и количество машин на маршруте. Время оборота по маршруту зависит от протяжённости маршрута, частоты расположения остановок, перекрёстков. **Оплата проезда** обычно формируется по таким же принципам.

1.2 Формализация данных

База данных должна хранить информацию о:

- остановке и расписании приходящих на неё маршрутов;
- различных типах транспорта (автобус, троллейбус, трамвай);
- тарифах на проезд.

Таблица 1 — Категории и сведения о данных в базе

Категория	**Сведения**
Расписание	Время прибытия, идентификатор конечной остановки, номер маршрута, максимальная цена за проезд, ходит ли маршрут по выходным дням
Тариф	Стоимость, номер маршрута, идентификатор начальной и конечной остановки, время суток по прибытии
Остановка	Идентификатор остановки, названии, адрес, остановка по требованию, год установки, наличие электрической сети и рельс
Конкретный транспорт	Номер маршрута, идентификаторы начальной и конечной остановки, тип, дата запуска маршрута

ER-диаграмма модели системы общественного транспорта в нотации Чена представлена на рисунке 1.

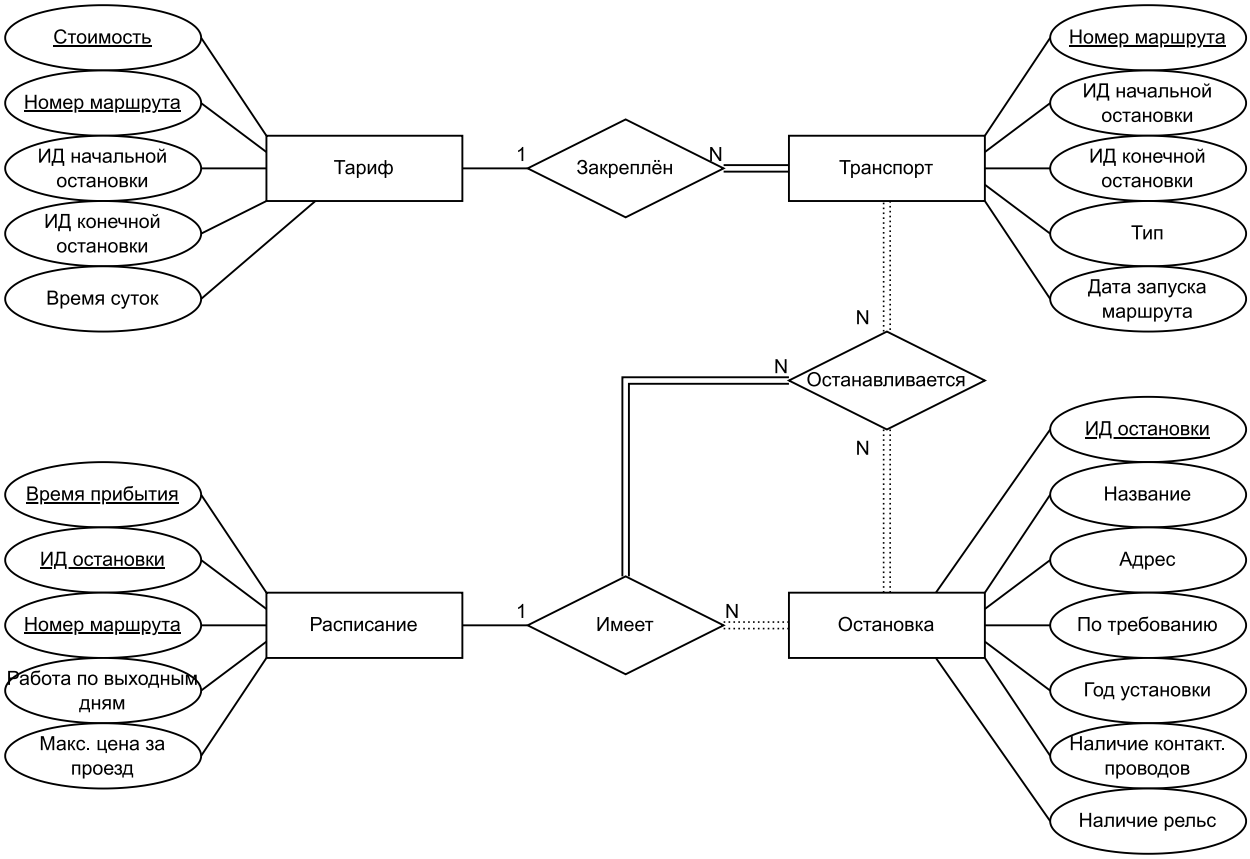

Рисунок 1 — ER-диаграмма

1.3 Типы пользователей

Для удобства необходимо разделить пользователей на разные категории (представлены в таблице 2).

Таблица 2 — Типы пользователей и их функционал

Тип пользователя	**Права доступа**
«Пассажир»	Просмотр информации о маршрутах, остановках, расписаниях и тарифах
«Диспетчер»	Возможность смотреть, а также изменять информацию об остановках, расписаниях, маршрутах и их тарифах
«Администратор»	Кроме всех прав доступа диспетчера может удалять и создавать новые таблицы

Модель взаимодействия пассажира с программой представлена на рисунке 2.

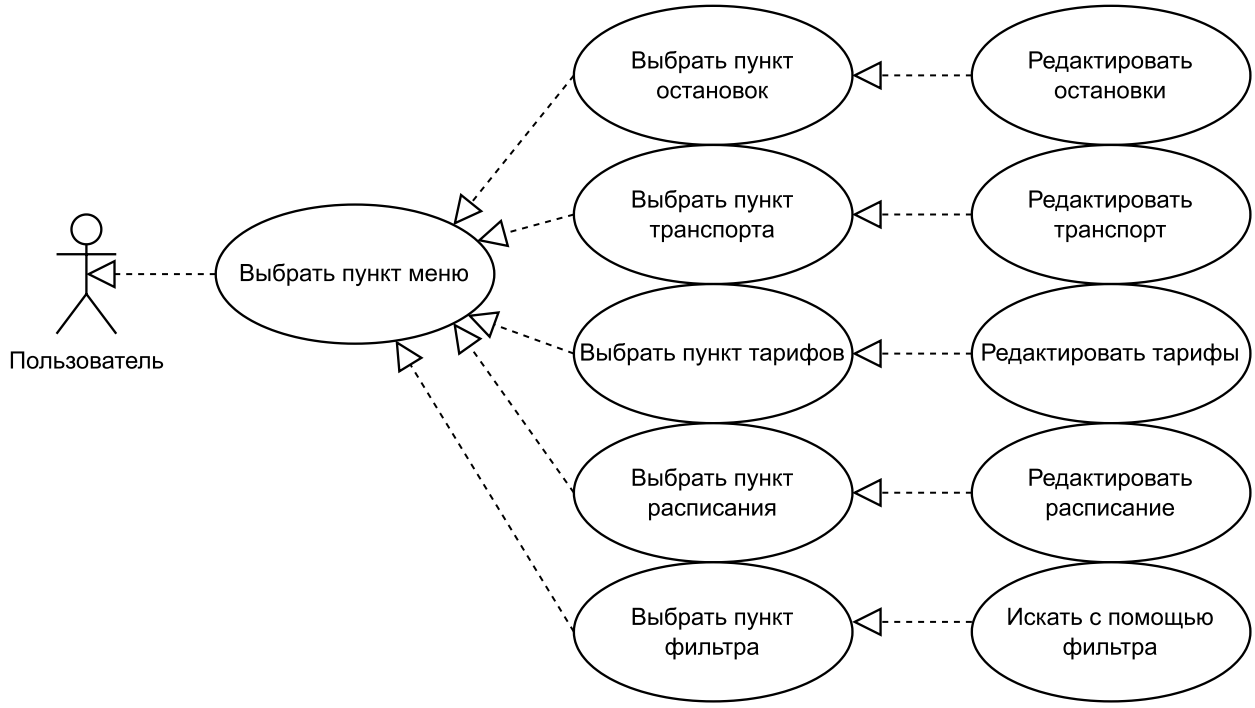

Рисунок 2 — Диаграмма прецедентов

1.4 Обзор моделей хранения данных и СУБД

1.4.1 Классификация модели данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных [2].

Существует 3 основных типа моделей организации данных:

- дореляционная:
  - иерархическая;
  - сетевая;
- реляционная;
- постреляционная:
  - объектно-ориентированная;
  - документно-ориентированная.

В иерархической модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок (показана на рис. 3).

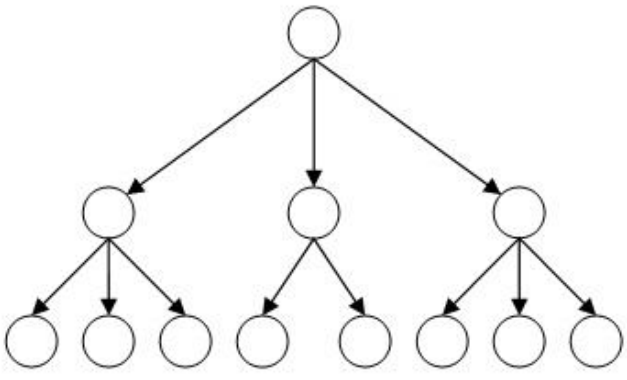

Рисунок 3 — Структура иерархической модели данных

Преобразование связи типа «один ко многим» между предком и потомком осуществляется практически автоматически в том случае, если потомок имеет одного предка. Сегмент со стороны связи «много» становится потомком, а сегмент со стороны «один» становится предком.

Однако у иерархической модели есть существенный недостаток — ситуация, в которой потомок в связи имеет не одного, а двух и более предков. Так как подобное положение является невозможным для иерархической модели, то отражаемая структура данных нуждается в преобразованиях, которые сводятся к замене одного дерева, например, двумя (если имеется два предка). В результате такого преобразования в базе данных появляется избыточность, так как единственно возможный выход из этой ситуации — дублирование данных.

В сетевой модели данных, в отличии от иерархической, у потомка может иметься любое число предков. Сетевая БД состоит из набора экземпляров определённого типа записи и набора экземпляров определённого типа связей между этими записями (рис. 4).

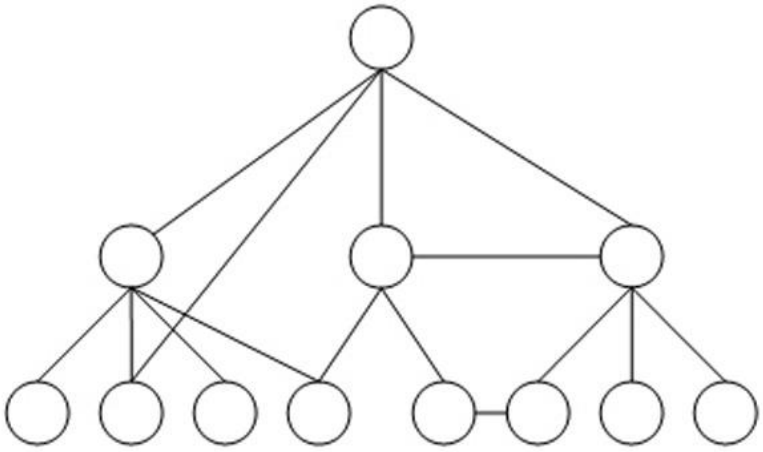

Рисунок 4 — Структура сетевой модели данных

Главным недостатком сетевой модели данных являются жёсткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения.

Иначе говоря, если будет необходимо изменить структуру данных, то нужно будет изменять и приложение.

Реляционная модель данных является совокупностью данных и состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк (записей) и столбцов (полей). На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений (структура на рис. 5). За счёт возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

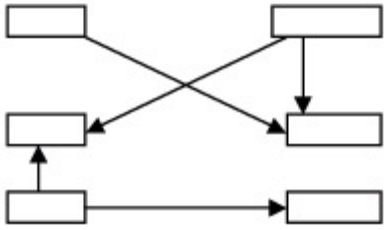

Рисунок 5 — Структура реляционной модели данных

Реляционная модель является удобной и наиболее широко используемой формой представления данных.

Объектно-ориентированная модель данных — модель данных, в которой сами данные моделируются в виде объектов, их атрибутов, методов и классов. К числу обязательных критериев этой модели данных относятся, например: поддержка инкапсуляции, классов, наследования, перегрузка и т. д.

Хотя ОО-подход и представляет более совершенные средства для отображения реального мира, чем реляционная модель: более естественное представление данных разными уровнями абстракции, определение новых типов данных и операций с ними, — у ОО-модели есть ряд недостатков. К ним относят отсутствие непроцедурных средств извлечения объектов из базы, явный процедурный код для ограничений целостности.

В отличие от остальных баз данных, документо-ориентированные оперируют «документами» — наборами атрибутов (ключ и соответствующее ему значение). Значения могут быть в свою очередь вложенными документами или массивами. Документы одного типа могут иметь общие атрибуты, а могут и не иметь — отсутствует жёстко заданная схема.

Преимущества ДО-модели данных:

- лучшая по сравнению с реляционными базами данных производительность при индексировании больших объёмов данных и большом количестве запросов на чтение;
- легче масштабируются в сравнении с SQL-решениями;
- децентрализированы.

Существенным минусом такой модели данных является отсутствие транзакционной логики и контроля целостности в большинстве реализаций — её необходимо реализовывать в логике приложения.

Была выбрана реляционная модель данных; необходима реализация отношения «многие ко многим» (транспорт и тарифы, транспорт и его остановки, остановки и расписания), что легче всего позволяют сделать именно реляционные базы данных — в то время как дореляционные модели позволяют осуществить лишь связь «один ко многим», постреляционные оперируют

понятиями документов и графов, что совершенно не соответствует городской системе общественного транспорта.

1.5 Выводы из аналитического раздела

В данном разделе была проанализирована предметная область и поставленная задача, рассмотрены различные способы её решения. Был проведён обзор разных моделей данных и осуществлён их выбор. Были определены основные требования к базе данных городской транспортной системы, её ролевая модель. Продемонстрирована модель «сущность-связь» в нотации Чена и диаграмма прецедентов.

2 Конструкторский раздел

2.1 Проектирование базы данных

2.1.1 Таблицы

База данных должна хранить рассмотренные в таблице 1 данные. Для этого можно выделить следующие таблицы:

- таблица тарифов `fare`;
- таблица транспорта `transport`;
- таблица остановок `transport_stop`;
- таблица расписания `timetable`.

Таблица `fare` должна содержать в себе информацию о тарифах и их особенностях:

- `root_number` — номер маршрута. Входит в первичный ключ.
- `price` — цена за проезд, установленная по тарифу. Входит в первичный ключ;
- `start_id` — уникальный идентификатор начальной остановки маршрута;
- `stop_id` — уникальный идентификатор конечной остановки маршрута;
- day_time — часы работы маршрута. Примеры: «с 8 утра до 23 вечера», «круглосуточно»;

Таблица `transport` должна хранить информацию о маршрутах городского транспорта, в том числи и о:

- `root_number` — номер соответствующего маршрута как уникальный идентификатор;
- `start_id` — уникальный идентификатор начальной остановки маршрута;
- `stop_id` — уникальный идентификатор конечной остановки маршрута;

- transport_type — тип городского транспорта. Возможные варианты: «трамвай», «троллейбус», «автобус», «маршрутка», «электробус».
- entry_date — дата введения маршрута в эксплуатацию.

Таблица transport_stop содержит данные об остановках городского общественного транспорта:

- id — уникальный идентификатор остановки;
- name — название остановки;
- address — адрес остановки;
- request_stop — является ли остановка таковой «по требованию»;
- install_year — год установки;
- electricity — факт наличия контактной сети для троллейбуса;
- rails — факт наличия рельс для трамвая.

Таблица timetable представляет собой расписание маршрутов:

- timing — время прибытия на остановку. Входит в первичный ключ;
- transport_stop_id — уникальный идентификатор остановки соответствующего маршрута. Входит в первичный ключ;
- root — номер маршрута. Входит в первичный ключ;
- weekends — ходит ли маршрут по выходным дням;
- max_price — максимальная цена за проезд.

Таблица tr_trst реализует отношение «многие ко многим» между таблицами transport_stop и transport. Обладает следующими полями, которые вместе образуют первичный ключ:

- root_number — номер соответствующего маршрута;
- transport_stop_id — уникальный идентификатор остановки.

2.1.2 Ограничения

Для корректной работы программы необходимо наложить некоторые ограничения рассматриваемой предметной области (система общественного транспорта):

- `fare` и `timetable` — цена за проезд и максимальное значение цены за проезд должны быть неотрицательными;
- `transport_stop` — идентификатор остановки должен быть строго положительным;
- `transport` — троллейбус не может останавливаться на остановке без контактного провода, а трамвай — на остановке без рельса.

Код создания таблиц и их ограничений представлен в приложении А.

Схема базы данных представлена на рисунке 6.

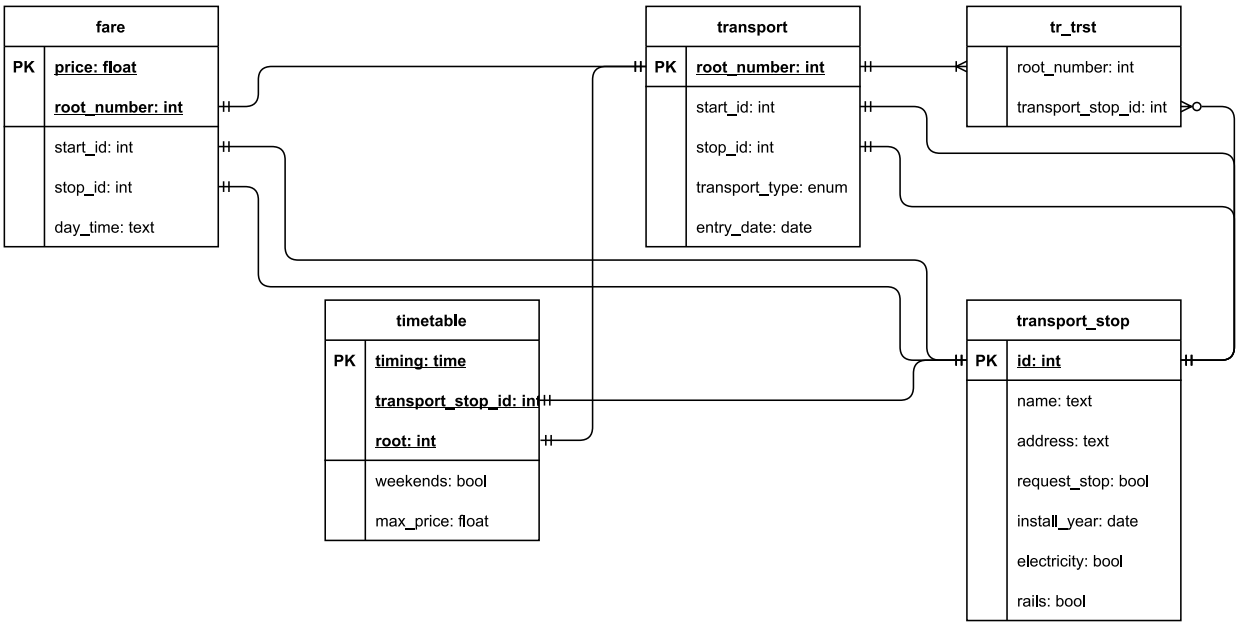

Рисунок 6 — Схема разработанной БД

2.1.3 Функции и триггеры

Для корректной работы таблицы необходимо создать несколько функций и триггеров.

Функция `time_diff`, вычисляющую разницу времени прихода транспорта со средним временем прихода транспорта на остановку по расписанию. Входные аргументы: время прихода транспорта на остановку.

Возвращаемое значение: разница между средним временем по расписанию и временем прихода транспорта к остановке.

Функция `all_roles` возвращает таблицу из всех зарегистрированных ролей в базе данных.

Триггеры `check_transport`, `check_fare` и их функции — `check_transport()` и `check_fare()` проверяют соответствие типа остановки и приходящего на неё транспорта: на остановку без рельсов или контактного провода не может прийти трамвай или троллейбус соответственно.

Исходный код функций и триггеров представлен в приложении Б.

2.2 Проектирование программы

Для успешной реализации поставленной задачи программа должна предоставлять следующие возможности:

- авторизация;
- вывод списка всех тарифов (также с фильтрацией по характеристикам);
- вывод списка всех расписаний (также с фильтрацией по характеристикам);
- вывод списка всех транспортных остановок (также с фильтрацией по характеристикам);
- вывод списка всех маршрутов (также с фильтрацией по характеристикам).
- поиск данных по фильтру.

Помимо вышеупомянутых возможностей, «диспетчер» может:

- добавлять маршрут, тариф, запись расписания или остановку;
- удалять маршрут, тариф, запись расписания или остановку;
- изменять маршрут, тариф, запись расписания или остановку.

2.3 Проектирование приложения

Дабы отделить бизнес-логику от её представления конечному пользователю и использовать код повторно реализован паттерн MVC.

MVC (Model–View–Controller, т. е. «Модель–Представление–Контроллер») — схема разделения данных и бизнес-логике по трём отдельным компонентам [2] (представлены на рисунке 7):

- *Модель* представляет данные приложения и реагирует на изменения контроллера;
- *Контроллер* реагирует на действия пользователя через представление, уведомляя модель об изменениях;
- *Представление* отображает модель пользователю.

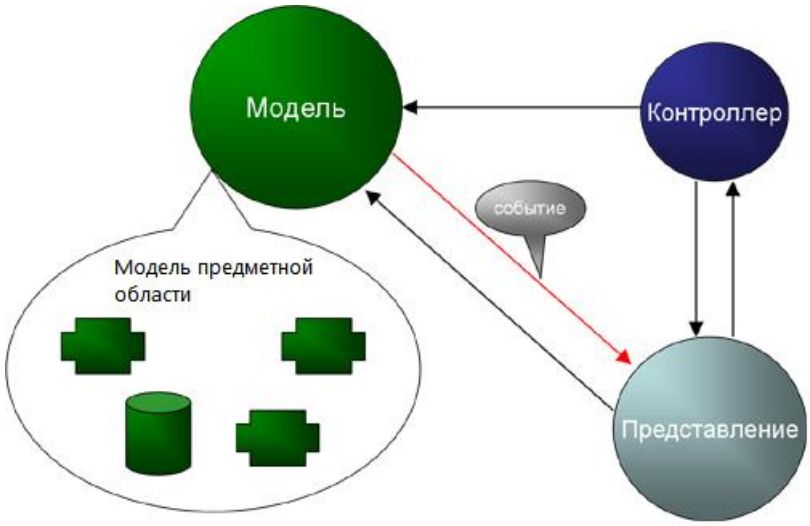

Рисунок 7 — Схема MVC

Компонент «модель» состоит из 5 сущностей:

- `Client` — сущность пользователя приложения;
- `Fare` — сущность, представляющая тарифы;
- `Timetable` — сущность, представляющая расписание;
- `Transport` — сущность, представляющая транспорт;
- `TransportStop` — сущность, представляющая остановку транспорта.

Хранящиеся в этих сущностях поля дублируют соответствующие поля таблиц из БД, за исключением логина и пароля пользователя, которые не хранятся на стороне клиента.

На рисунке 8 представлена UML-схема взаимодействия компонентов приложения между собой.

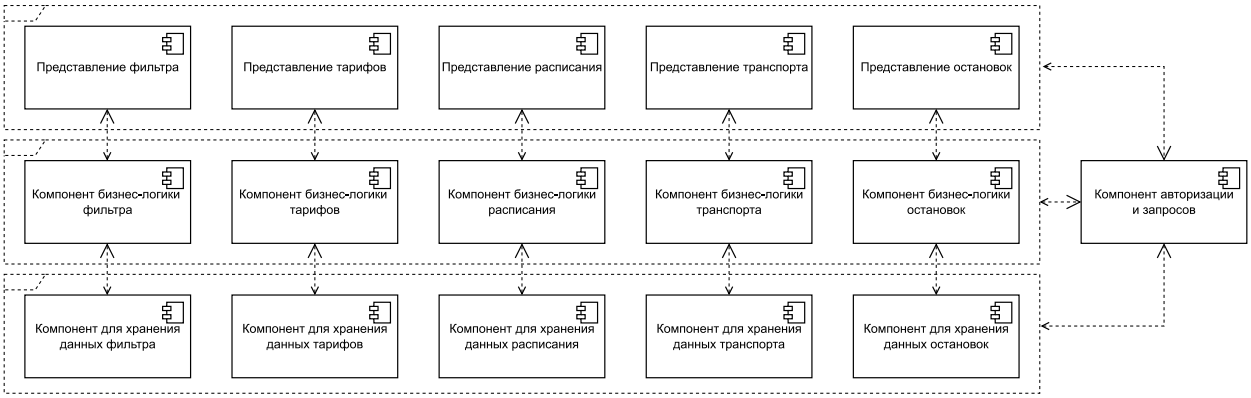

Рисунок 8 — Схема взаимодействия компонентов

2.5 Вывод

В разделе представлено проектирование базы данных и приложения — схема базы данных общественного транспорта, требования по хранению данных и обработке данных в программе, UML-диаграмма её компонентов. Продемонстрированы методы организации потока данных в программе. Перечислены функции, триггеры и таблицы для корректного функционирования приложения и БД.

3 Технологический раздел

3.1 Популярные реляционные СУБД и их основные функции

Система управления базами данных (сокр. СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [3].

Основными функциями СУБД являются:

- управление данными во внешней памяти;
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД.

В данном подразделе рассмотрены популярные реляционные СУБД, которые могут быть использованы для реализации хранения в разрабатываемом программном продукте.

3.1.1 PostgreSQL

PostgreSQL [4] — это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных [5].

PostgreSQL предоставляет транзакции со свойствами атомарности, согласованности, изоляции, долговечности (ACID [6]), автоматически обновляемые представления, материализованные представления, триггеры, внешние ключи и хранимые процедуры. Данная СУБД предназначена для обработки ряда рабочих нагрузок, от отдельных компьютеров до хранилищ данных или веб-сервисов с множеством одновременных пользователей.

Рассматриваемая СУБД управляет параллелизмом с помощью технологии управления многоверсионным параллелизмом — MVCC [7]. Эта технология даёт каждой транзакции «снимок» текущего состояния базы данных, позволяя вносить изменения, не затрагивая другие транзакции. Это в значительной

степени устраняет необходимость в блокировках чтения [8] и гарантирует, что база данных поддерживает принципы ACID.

3.1.2 Oracle Database

Oracle Database [9] — объектно-реляционная система управления базами данных компании Oracle [10]. На данный момент эта СУБД является самой популярной в мире [11].

Все транзакции Oracle Database обладают свойствами ACID, поддерживает триггеры, внешние ключи и хранимые процедуры. Данная СУБД подходит для разнообразных рабочих нагрузок и может использоваться практически в любых задачах. Особенностью Oracle Database является быстрая работа с большими массивами данных.

Oracle Database может использовать один или более методов параллелизма. Сюда входят механизмы блокировки для гарантии монопольного использования таблицы одной транзакцией, методы временных меток, которые разрешают сериализацию транзакций и планирование транзакций на основе проверки достоверности.

3.1.3 MySQL

MySQL [12] — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle.

Рассматриваемая СУБД имеет два основных движка хранения данных: InnoDB [13] и myISAM [14]. Движок InnoDB полностью совместим с принципами ACID, в отличии от движка myISAM. СУБД MySQL подходит для использования при разработке веб-приложений, что объясняется очень тесной интеграцией с популярными языками PHP [15] и Perl [16].

Реализация параллелизма в СУБД MySQL реализована с помощью механизма блокировок, который обеспечивает одновременный доступ к данным.

3.1.4 Вывод

Для решения задачи была выбрана СУБД PostgreSQL, потому что данная СУБД имеет поддержку языка plpython3u [17], который упрощает процесс интеграции базы данных в разрабатываемое приложение. Кроме того,

PostgreSQL проста в развёртывании, имеет подробную документацию на русском языке, множество встроенных языковых средств, не имеющих аналога в других SQL-подобных средах и языках. Стоит отметить встроенный пулер запросов, инкрементальное резервное копирование и оптимизированное секционирование таблиц.

3.2 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования был выбран Rust, т. к.:

- это быстрый и безопасный язык, обладающий уникальной моделью управления памятью [18];
- он имеет подробную и исчерпывающую документацию с примерами и комментариями [19];
- для этого языка имеется модуль для работы с СУБД PostgreSQL — postgres [20].

Средой разработки была выбрана Visual Studio Code:

- она бесплатна в использовании;
- имеет множество возможностей для написания и отладки кода;
- поддерживает множество языков программирования, включая Rust.

3.3 Структура и состав модулей

Программа состоит из 16 модулей:

- `client` — модуль пользователя приложения. Также отвечает за установку соединения с базой данных;
- `fare_view`, `fare_controller`, `fare_model` — модули представления, логики и модели сущности тарифов;
- `timetable_view`, `timetable_controller`, `timetable_model` — модули представления, логики и модели сущности расписания;
- `transport_view`, `transport_controller`, `transport_model` — модули представления, логики и модели сущности транспорта;

- `transportstop_view`, `transportstop_controller`, `transportstop_model` — модули представления, логики и модели сущности остановки транспорта.

3.4 Интерфейс программы

3.4.1 Вход в систему

На рисунке 9 представлен экран входа в программу. Пользователю необходимо выбрать роль диспетчера, администратора или пассажира. Роли диспетчера и администратора защищены паролем. Для пассажира поле ввода пароля недоступно. Единственная кнопка «Выбрать» устанавливает соединение с базой данных городской транспортной системы по выбранной роли.

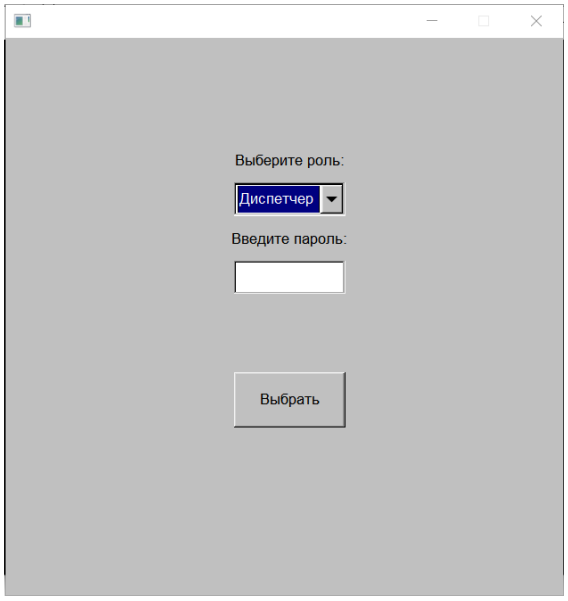

Рисунок 9 — Экран входа

3.4.2 Меню

Экран меню представлен на рисунке 10. В меню находятся следующие пункты:

- *Редактирование тарифов* — добавление, удаление, изменение записей о существующих тарифах в базе данных;
- *Редактирование транспорта* — добавление, удаление, изменение записей о существующих маршрутах в базе данных;
- *Редактирование остановок* — добавление, удаление, изменение записей о существующих остановках в базе данных;

- *Редактирование расписания* — добавление, удаление, изменение записей о расписании в базе данных;
- *Фильтр* — опциональный поиск записей в базе данных.

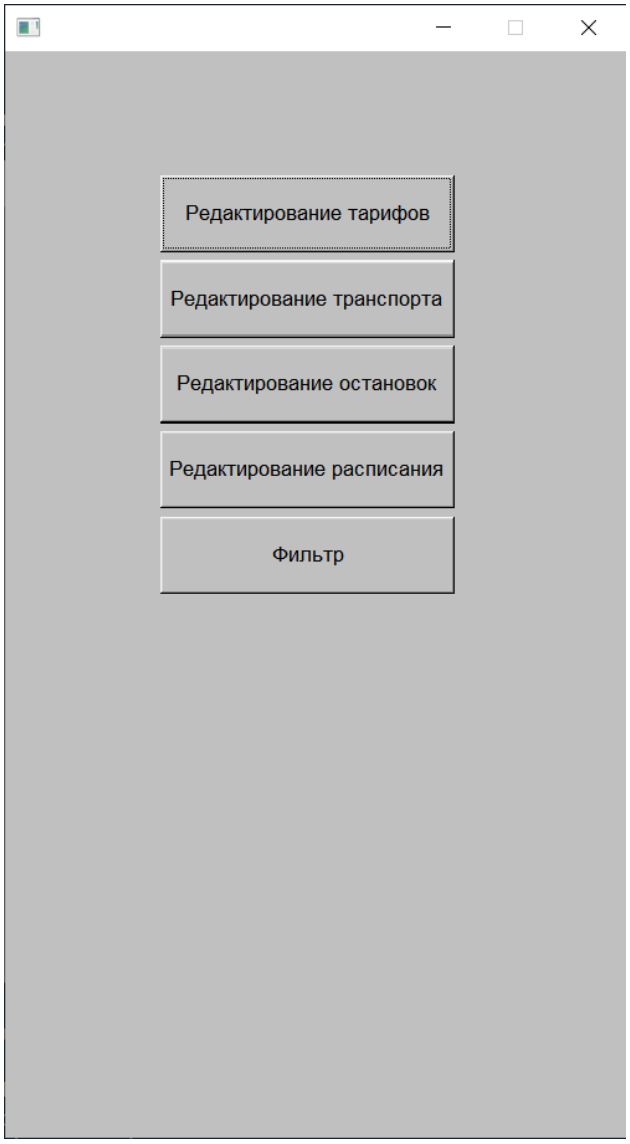

Рисунок 10 — Меню

Все пункты кроме «Фильтра» недоступны для пассажира.

3.4.3 Пункты редактирования записей

Для того чтобы обеспечить удобство ориентирования пользователя по программе, окна редактирования данных в основном единообразны. Дальнейшей интерфейс будет продемонстрирован на примере изменения тарифов.

На рисунке 11 окно изменения записей о тарифах. Оно включает в себя таблицу тарифов, формы ввода изменяемых данных, а также кнопки для добавления и удаления новых.

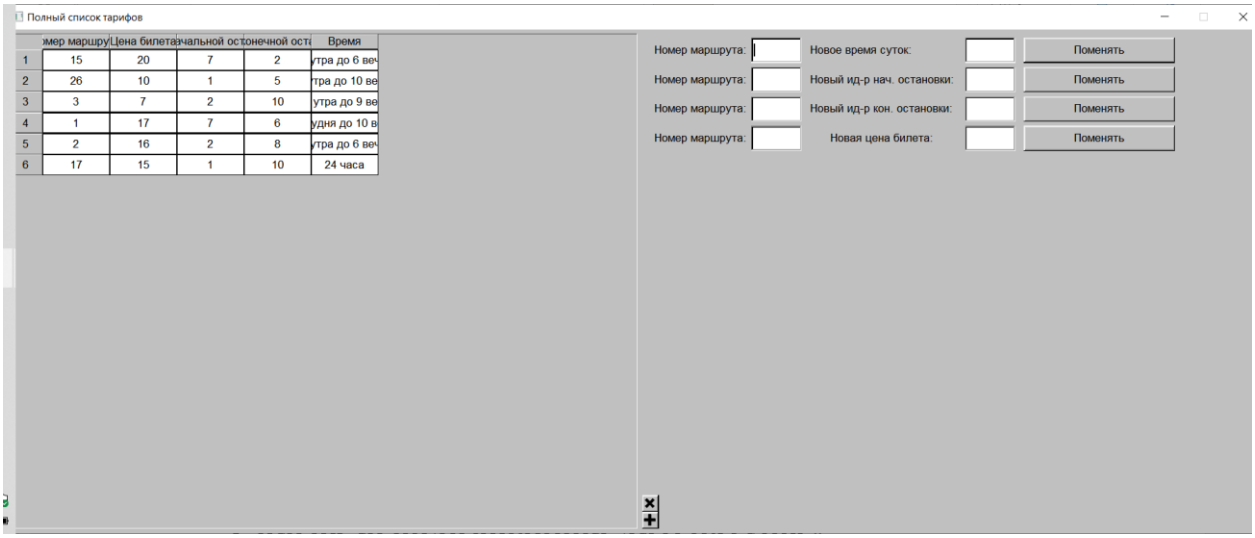

Рисунок 11 — Полный список тарифов и формы для изменения

3.4.4 Добавление и удаление записей

Рисунок 12 демонстрирует окно удаления данных из БД. Оно выглядит почти одинаково для всех 4 таблиц, поэтому для краткости будет приведено только окно удаления тарифа. Чтобы удалить какую-либо запись в БД надо ввести её первичные ключи — здесь это цена билета и номер маршрута. После нажатия кнопки удаления в правом нижнем углу окна запись удаляется из базы данных.

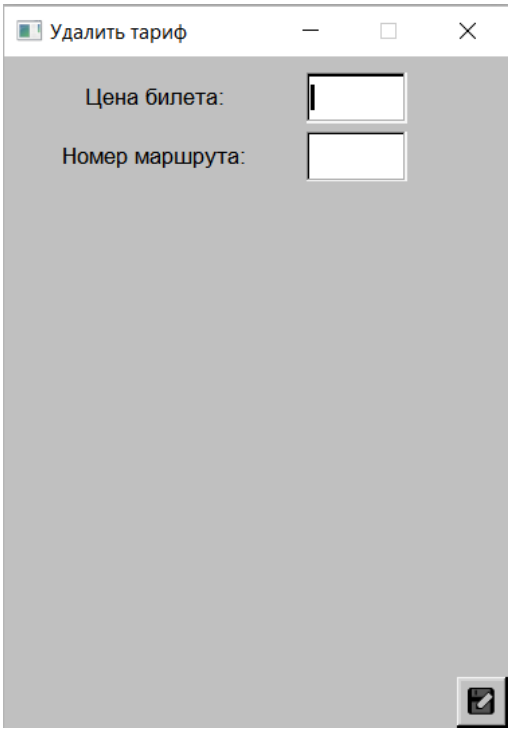

Рисунок 12 — Удаление записи

На рисунке 13 изображено окно добавления новой записи (здесь, нового тарифа). Чтобы добавить новую запись, например, о тарифе в БД, нужно ввести цену билета, номер маршрута, закреплённого за ним, верные идентификаторы начальной и конечной остановок и время суток, в течение которого тариф будет действительным.

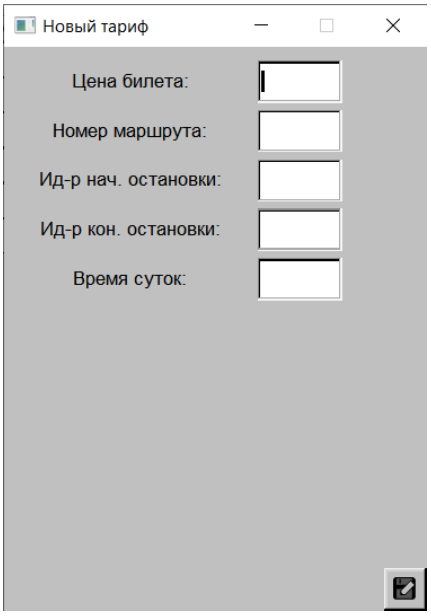

Рисунок 13 — Окно добавления записи (на примере тарифа)

3.4.5 Поиск записей

На рисунке 14 окно поиска по базе данных. Оно разделено на 4 секции, соответствующим 4 таблицам, по которым будет производиться поиск.

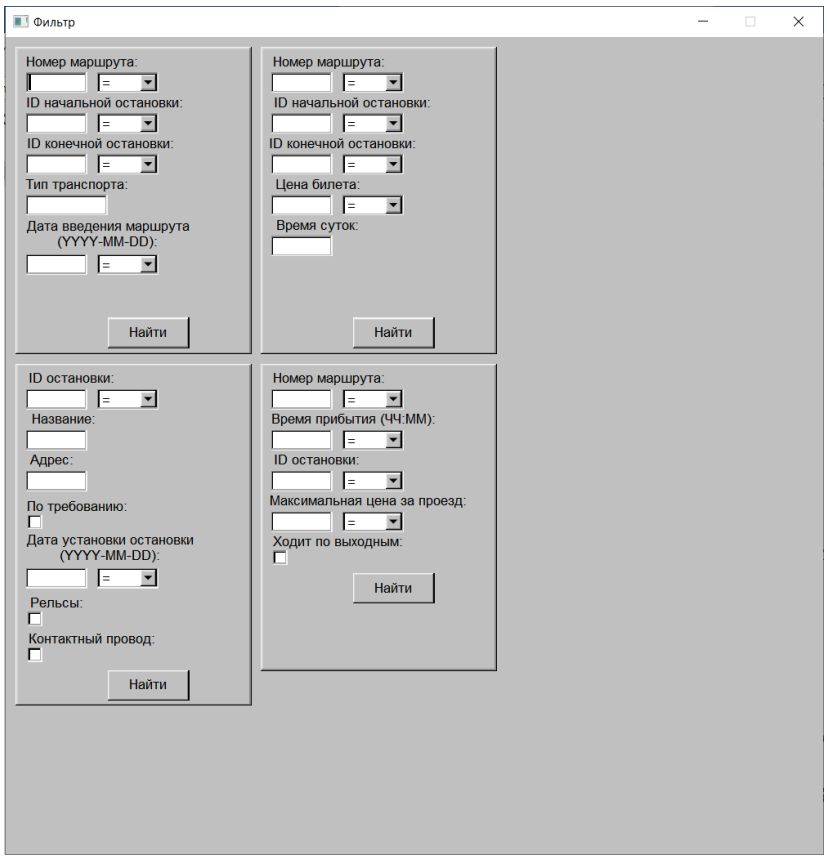

Рисунок 14 — Поиск записей в БД

Не все поля обязательны для заполнения. Так, не заполнив ни одно поле, пользователь увидит все записи в результирующей таблице, а если будет заполнено только, к примеру, поле «Номер маршрута», то выведутся только маршруты с таким номером как на рисунке 15.

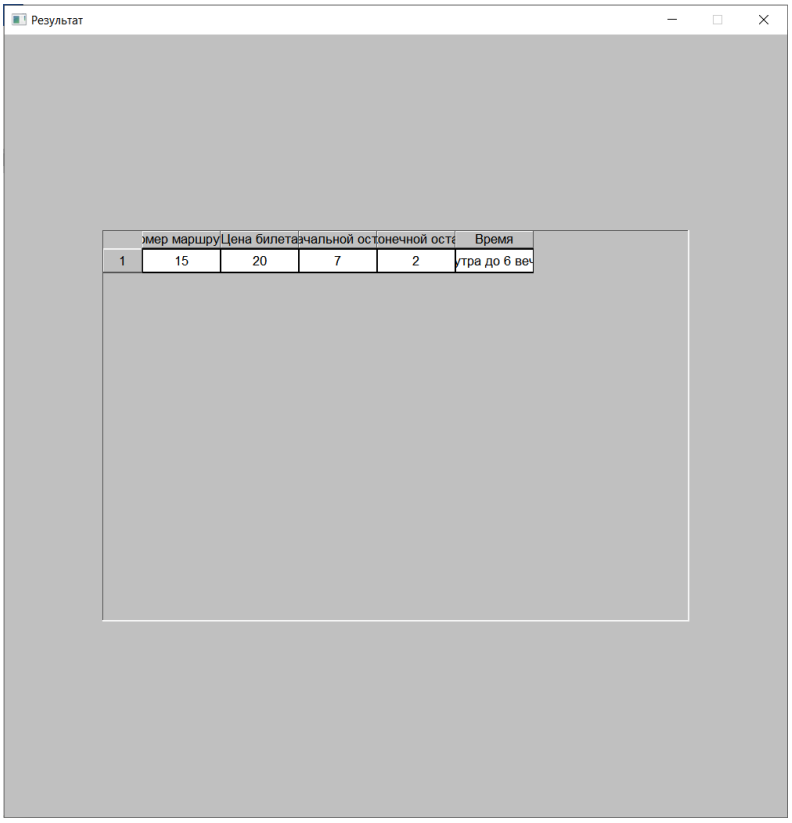

Рисунок 15 — Результирующая таблица

Также для некоторых полей, которые соотносятся с полями базы данных (числового, временного типа или типа даты), предусмотрено ранжирование «больше», «меньше», «больше или равно», «меньше или равно» и «равно». Последний «знак» задаётся по умолчанию.

3.5 Вывод

В данном разделе был обоснован выбор СУБД (PostgreSQL), языка программирования клиентского приложения (Rust), среды разработки ПО (Visual Studio Code), рассмотрены структура и состав программных модулей. Продемонстрирован интерфейс клиентского приложения; он основан на графических окнах и кнопках. Это позволит значительно облегчить и упростить взаимодействие пользователя с приложением: не надо вводить запросы к СУБД вручную или заучивать командную строку.

4 Экспериментальный раздел

4.1 Технические характеристики устройства

Компьютер, на котором ставился эксперимент, обладает следующими техническими характеристиками:

- операционная система: Windows 10 Домашняя для одного языка 21H2, x64 [21];
- центральный процессор: Intel® Core™ i7-8550U 1.80ГГц 1.99 ГГц [22];
- количество ядер: 4;
- количество потоков: 8;
- объём оперативного запоминающего устройства: 8 ГБ.

Для проведения эксперимента из оперативной памяти были выгружены все программы, кроме СУБД и среды выполнения программы.

4.2 Исследование зависимости времени выполнения запроса от индекса

Суть эксперимента заключается в наблюдении зависимости времени ответа базы данных от наличия индексации и количества строк в одной из таблиц БД.

Различные индексы используются для увеличения производительности СУБД и ускорения поиска по базе. В данном эксперименте были использованы индексы на основе B-дерева и хэша. Первый эффективен в большинстве случаев, т. е. на тех данных, которые можно каким-либо образом упорядочить (использую отношения «больше», «меньше», «больше или равно» и т. д.). Хэш-индекс же эффективен, когда столбец участвует в сравнении с оператором «равно». [23]

Поскольку ко всем четырём таблицам пользователь обращается примерно с одинаковой интенсивностью (в следствие интерфейса программы), то проводить эксперимент с какой-то определённой таблицей не принципиально. Поэтому зависимость будет продемонстрирована, к примеру, на таблице `Fare` (листинг 1). Её результат продемонстрирован в таблице 3, на рисунках 16 и 17.

Листинг 1: Тестируемый запрос

```
select root, max_price
from timetable
where max_price > all
(
      select max_price
      from fare
      where start_id = 123
);
```

Таблица 3 — Зависимость ответа БД от индексации и кол-ва строк

Кол-во строк	Время ответа без индексации, мс	Время ответа с индексацией b-деревом, мс	Время ответа с хэш-индексацией, мс
5	0.650	0.078	0.072
10	2.089	0.086	0.094
50	6.233	0.149	0.139
100	11.851	0.224	0.209
500	65.209	0.957	0.778
1000	155.330	1.520	1.330

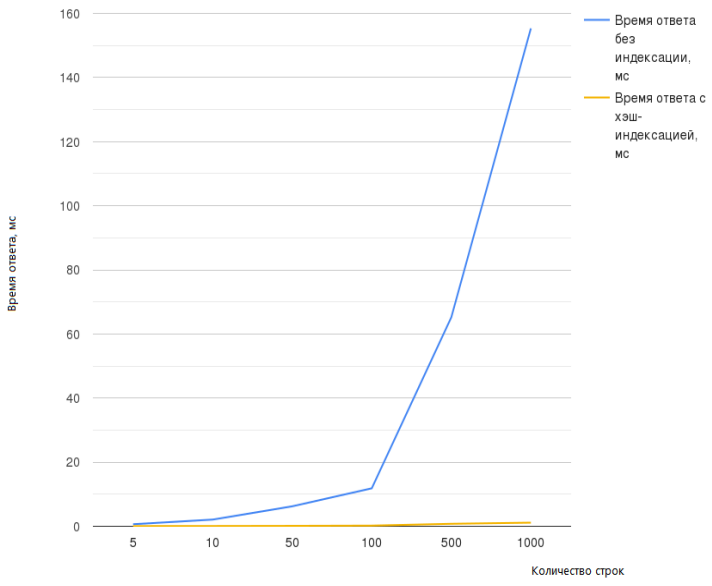

Рисунок 16 — График зависимости

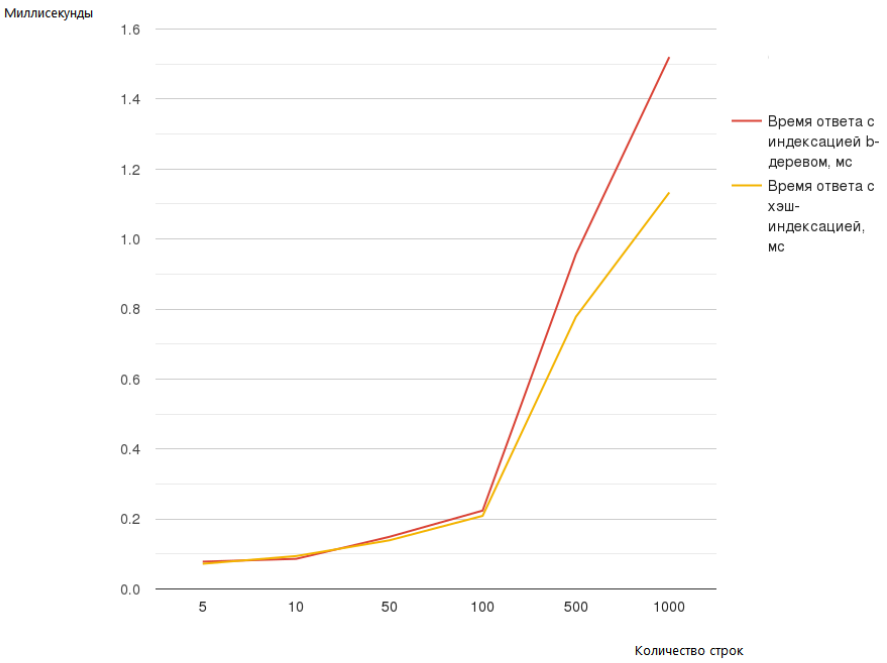

Рисунок 17 — График зависимости (увеличенный)

Таблица и график показывают, что с увеличением количества строк таблицы время ответа базы данных экспоненциально увеличивается без использования индекса. Однако индексирование позволяет сократить время на поиск нужных записей. Например, для таблицы из 1000 строк индексация b-деревом даёт сокращение времени в 150 миллисекунд, а хэшем — в 170 миллисекунд. Небольшой проигрыш по времени b-дерева объясняется тем, что этот алгоритм предназначен более для проверки диапазонов данных, в то время как в тестируемом запросе использовалось только отношение «равно».

ЗАКЛЮЧЕНИЕ

Была проанализирована поставленная задача реализации базы данных городской транспортной системы и приложения к ней; формализована предметная область — система общественного транспорта и сопутствующая ей инфраструктура. Были рассмотрены разные модели данных. Выбрана реляционная модель данных, так как она наиболее точно соответствует структуре выбранной предметной области.

Было проведено проектирование базы данных и приложения: составлены схема базы данных общественного транспорта, требования по хранению данных и обработке данных в программе, приложение было разработано с применением паттерна MVC (модель-представление-контроллер).

Был обоснован выбор СУБД, языка программирования клиентского приложения, рассмотрены структура и состав программных модулей. Была реализована разработанная программа, создана база данных и заполнена правдоподобными данными. Также продемонстрирован интерфейс самого приложения.

В ходе выполнения экспериментальной части было установлено, что с помощью индексов можно значительно увеличить скорость выполнения запросов в базе данных.

В качестве вариантов дальнейшего развития разработанной системы можно рассмотреть следующее: разработка функций удаления и создания таблиц, улучшение и оптимизация графического интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Устав автомобильного транспорта и городского наземного электрического транспорта [Текст]: Федеральный закон от 08.11.2007 №259-ФЗ (ред. от 02.07.2021) (с изм. и доп., вступ. в силу с 01.03.2022)
2. Model-View-Controller в .Net. [Электронный ресурс] Режим доступа: http://rsdn.org/article/patterns/modelviewpresenter.xml (дата обращения: 21.05.2022)
3. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006
4. PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql
5. PostgreSQL: вчера, сегодня, завтра [Электронный ресурс]. Режим доступа: https://postgrespro.ru/blog/media/17768
6. Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions
7. Documentation: 12: 13.1. Introduction - PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/12/mvcc-intro.html
8. Применение блокировок чтения/записи | IBM [Электронный ресурс]. Режим доступа: https://www.ibm.com/docs/ru/aix/7.2?topic=programming-using-readwrite-locks
9. SQL Language | Oracle [Электронный ресурс]. Режим доступа: https://www.oracle.com/database/technologies/appdev/sql.html
10. Oracle | Integrated Cloud Applications and Platform Services [Электронный ресурс]. Режим доступа: https://www.oracle.com/index.html
11. DB-Engines Ranking [Электронный ресурс]. Режим доступа: https://db-engines.com/en/ranking
12. MySQL Database Service is a fully managed database service to deploy cloud-native applications. [Электронный ресурс]. Режим доступа: https://www.mysql.com/

13. MySQL Reference Manual 8.0: The InnoDB Storage Enginee [Электронный ресурс]. Режим доступа: https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html
14. MySQL Reference Manual 16.2: The MyISAM Storage Engine. [Электронный ресурс]. Режим доступа: https://dev.mysql.com/doc/refman/8.0/en/myisam-storage-engine.html
15. PHP: Hypertext Preprocessor [Электронный ресурс]. Режим доступа: https://www.php.net/
16. The Perl Programming Language [Электронный ресурс]. Режим доступа: https://www.perl.org/
17. PostgreSQL: Документация: 9.6: 44.1. Python 2 и Python 3. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql/9.6/plpython-python23
18. Rust Programming Language. [Электронный ресурс]. Режим доступа: https://www.rust-lang.org/
19. The Rust Programming Language — Rust Programming Language. [Электронный ресурс]. Режим доступа: https://www.rust-lang.org/book
20. postgres — crates.io: Rust Package Registry. [Электронный ресурс]. Режим доступа: https://crates.io/crates/postgres
21. Скачать Windows 10. [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10
22. Intel Core i78550U Processor 8M Cache up to 4.00 GHz Спецификации продукции. [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i78550u-processor-8m-cache-up-to-4-00-ghz.html
23. Postgres Pro Standard : Документация: 14: 11.2. Типы индексов : Компания Postgres Professional. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgrespro/14/indexes-types

ПРИЛОЖЕНИЕ А

В листинге 2 приведён DDL-код создания таблиц и их ограничений базы данных.

Листинг 2: Создание таблиц и ограничений

```
CREATE TABLE public.transport_stop (
     id serial4 NOT NULL,
     "name" text NULL,
     address text NULL,
     request_stop bool NULL,
     install_year date NULL,
     electricity bool NULL,
     rails bool NULL,
     CONSTRAINT transport_stop_pkey PRIMARY KEY (id)
);
CREATE TABLE public.transport (
     root_number int4 NOT NULL,
     start_id int4 NOT NULL,
     stop_id int4 NULL,
     transport_type text NULL,
     entry_date date NULL,
     CONSTRAINT transport_pkey PRIMARY KEY (root_number),
     CONSTRAINT transport_start_id_fkey FOREIGN KEY (start_id)
REFERENCES public.transport_stop(id),
     CONSTRAINT transport_stop_id_fkey FOREIGN KEY (stop_id)
REFERENCES public.transport_stop(id)
);
CREATE TABLE public.fare (
     root_number int4 NOT NULL,
     price float8 NOT NULL,
     start_id int4 NOT NULL,
     stop_id int4 NOT NULL,
     day_time text NULL,
     CONSTRAINT fare_pkey PRIMARY KEY (price, root_number),
     CONSTRAINT fare_root_number_fkey FOREIGN KEY (root_number)
REFERENCES public.transport(root_number),
     CONSTRAINT fare_start_id_fkey FOREIGN KEY (start_id)
REFERENCES public.transport_stop(id),
     CONSTRAINT fare_stop_id_fkey FOREIGN KEY (stop_id)
REFERENCES public.transport_stop(id)
);
CREATE TABLE public.timetable (
     timing time NOT NULL,
     transport_stop_id int4 NULL,
     root int4 NULL,
     weekends bool NOT NULL,
     max_price float8 NULL,
```

```
    CONSTRAINT timetable_pkey PRIMARY KEY (timing),
    CONSTRAINT timetable_root_fkey FOREIGN KEY (root) REFERENCES
public.transport(root_number),
    CONSTRAINT timetable_transport_stop_id_fkey FOREIGN KEY
(transport_stop_id) REFERENCES public.transport_stop(id)
);
CREATE TABLE public.tr_trst (
    root_number int4 NOT NULL,
    transport_stop_id int4 NULL,
    CONSTRAINT rn_rn FOREIGN KEY (root_number) REFERENCES
public.transport(root_number) ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT tsi_tsi FOREIGN KEY (transport_stop_id)
REFERENCES public.transport_stop(id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

ПРИЛОЖЕНИЕ Б

В листинге 3 приведён код функций, разработанных в ходе выполнения работы.

Листинг 3: Необходимые функции

```
create or replace function public.all_roles()
 returns table(role_name text)
 language sql
as $function$
     select usename as role_name
from pg_catalog.pg_user
order by role_name desc;
$function$;

create or replace function public.time_diff(x time without time
zone) returns time without time zone language sql as $function$
                  select x -
                  (
                       select avg(timing)
from timetable
                  )
              $function$
;
create or replace function public.check_fare() returns trigger
language plpgsql as $function$
     begin
          if (select rails from transport_stop where id =
NEW.start_id) = false and new.transport_type = 'трамвай' then
raise exception 'Трамвайная остановка должна иметь рельсы для
хождения трамвая.';
elseif (select electricity from transport_stop where id =
NEW.start_id) = false and new.transport_type = 'троллейбус' then
raise exception 'Троллейбусная остановка должна иметь контактные
провода для хождения троллейбуса.'; end if;
if (select rails from transport_stop where id = NEW.stop_id) =
false and new.transport_type = 'трамвай' then raise exception
'Трамвайная остановка должна иметь рельсы для хождения трамвая.';
elseif (select electricity from transport_stop where id =
NEW.stop_id) = false and new.transport_type = 'троллейбус' then
raise exception 'Троллейбусная остановка должна иметь контактные
провода для хождения троллейбуса.'; end if;
return new;
end;
$function$;
create trigger check_fare before insert or update on public.fare
for each row execute function check_fare();
```

```
create or replace function public.check_transport() returns
trigger language plpgsql as $function$
     begin
          if (select rails from transport_stop where id =
NEW.start_id) = false and new.transport_type = 'трамвай' then
raise exception 'Трамвайная остановка должна иметь рельсы для
хождения трамвая.';
elseif (select electricity from transport_stop where id =
NEW.start_id) = false and new.transport_type = 'троллейбус' then
raise exception 'Троллейбусная остановка должна иметь контактные
провода для хождения троллейбуса.'; end if;
if (select rails rom transport_stop here id = NEW.stop_id) =
false and new.transport_type = 'трамвай' then raise exception
'Трамвайная остановка должна иметь рельсы для хождения трамвая.';
elseif (select electricity from transport_stop where id =
NEW.stop_id) = false and new.transport_type = 'троллейбус' then
raise exception 'Троллейбусная остановка должна иметь контактные
провода для хождения троллейбуса.'; end if;
return new; end;
$function$;


create trigger check_transport before insert or update on
public.transport for each row execute function check_transport();


CREATE OR REPLACE FUNCTION public.check_timetable() RETURNS
trigger LANGUAGE plpgsql AS $function$
     begin
          if (select rails from transport_stop where id =
NEW.transport_stop_id) = false and (select transport_type from
transport
where root_number = NEW.root) = 'трамвай' then  raise exception
'Трамвайная остановка должна иметь рельсы для хождения трамвая.';

elseif (select electricity from transport_stop where id =
NEW.transport_stop_id) = false and (select transport_type from
transport
where root_number = NEW.root) = 'троллейбус' then raise exception
'Троллейбусная остановка должна иметь контактные провода для
хождения троллейбуса.'; end if; return new; end;
$function$;
```

ПРИЛОЖЕНИЕ В

В листинге 3 представлен код ролевой модели базы данных.

Листинг 3: Выделенные роли

```
CREATE ROLE passenger WITH
     NOSUPERUSER
     NOCREATEDB
     NOCREATEROLE
     NOINHERIT
     LOGIN
     NOREPLICATION
     BYPASSRLS
     CONNECTION LIMIT -1;
GRANT SELECT ON TABLE public.fare TO passenger;
GRANT USAGE ON SCHEMA public TO passenger;
GRANT SELECT ON TABLE public.timetable TO passenger;
GRANT SELECT ON TABLE public.transport TO passenger;
GRANT SELECT ON TABLE public.transport_stop TO passenger;


CREATE ROLE "operator" WITH
     NOSUPERUSER
     NOCREATEDB
     NOCREATEROLE
     NOINHERIT
     LOGIN
     NOREPLICATION
     BYPASSRLS
     CONNECTION LIMIT -1;
GRANT UPDATE, INSERT, SELECT, DELETE ON TABLE public.fare TO
"operator";
GRANT USAGE ON SCHEMA public TO "operator";
GRANT UPDATE, INSERT, SELECT, DELETE ON TABLE public.timetable TO
"operator";
GRANT UPDATE, INSERT, SELECT, DELETE ON TABLE public.transport TO
"operator";
GRANT UPDATE, INSERT, SELECT, DELETE ON TABLE
public.transport_stop TO "operator";


CREATE ROLE administrator WITH
     NOSUPERUSER
     NOCREATEDB
     NOCREATEROLE
     NOINHERIT
     LOGIN
     NOREPLICATION
     NOBYPASSRLS
     CONNECTION LIMIT -1;
```

```
GRANT REFERENCES, UPDATE, INSERT, TRUNCATE, SELECT, DELETE,
TRIGGER ON TABLE public.fare TO administrator;
GRANT REFERENCES, UPDATE, INSERT, TRUNCATE, SELECT, DELETE,
TRIGGER ON TABLE public.timetable TO administrator;
GRANT REFERENCES, UPDATE, INSERT, TRUNCATE, SELECT, DELETE,
TRIGGER ON TABLE public.transport TO administrator;
GRANT REFERENCES, UPDATE, INSERT, TRUNCATE, SELECT, DELETE,
TRIGGER ON TABLE public.transport_stop TO administrator;
```