Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

*К защите допустить:*

И.О. Заведующего кафедрой информатики

\_\_\_\_\_\_\_\_\_\_\_\_\_С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**АГРЕГАТОР АВТОБУСНЫХ МАРШРУТОВ И БИЛЕТОВ**

БГУИР КП 1-40 04 01 012 ПЗ

Студент Д. С. Кончик

Руководитель В. С. Плиска

Минск 2024

СОДЕРЖАНИЕ

[Введение 5](#_Toc184767161)

[1 Обзор существующих аналогов 6](#_Toc184767162)

[1.1 Атлас 6](#_Toc184767163)

[1.2 Яндекс.Транспорт 7](#_Toc184767164)

[1.3 Infobus 9](#_Toc184767165)

[2 Функциональные требования и инструменты разработки 11](#_Toc184767166)

[2.1 Функциональные требования к системе 11](#_Toc184767167)

[2.2 Выбор СУБД 12](#_Toc184767168)

[3 Проектирование базы данных 21](#_Toc184767173)

[3.1 Инфологическая модель 21](#_Toc184767174)

[3.2 Даталогическая модель 23](#_Toc184767175)

[3.3 Модель миграции данных 28](#_Toc184767176)

[4 Разработка базы данных 30](#_Toc184767177)

[4.1 Физическая модель базы данных 30](#_Toc184767178)

[4.2 Создание таблиц 31](#_Toc184767179)

[4.3 Оптимизация производительности 33](#_Toc184767180)

[4.4 Триггеры и процедуры 34](#_Toc184767181)

[4.5 Запросы 35](#_Toc184767182)

[5 Тестирование работоспособности базы данных 37](#_Toc184767183)

[Заключение 39](#_Toc184767184)

[Список литературных источников 40](#_Toc184767185)

[Приложение А (обязательное) Листинг кода 41](#_Toc184767186)

[Приложение Б (обязательное) Схема базы данных 52](#_Toc184767187)

[Приложение В (обязательное) Ведомость документов 53](#_Toc184767188)

Введение

В современных условиях цифровизации транспорта особое значение приобретает удобство и доступность информации о маршрутах транспорта, а также возможность быстрой и надежной покупки билетов. Несмотря на существование различных приложений и сервисов, многие из них либо не охватывают все возможные маршруты, либо не обеспечивают удобного интерфейса для покупки билетов. Это создает неудобства для пассажиров и снижает эффективность использования общественного транспорта.

Цель данной курсовой работы — разработка агрегатора автобусных маршрутов и билетов, который будет представлять собой информационную систему с базой данных, обеспечивающей централизованное хранение, обработку и предоставление информации о расписаниях, маршрутах и возможностях покупки билетов. Такая система будет востребована пассажирами, желающими оперативно планировать поездки, а также транспортными компаниями, стремящимися повысить эффективность своих услуг.

В рамках данной работы будут выполнены следующие задачи:

1. Анализ существующих аналогов и выявление их достоинств и недостатков.
2. Формирование функциональных требований и выбор подходящих инструментов для разработки.
3. Проектирование базы данных, включая разработку инфологической и даталогической моделей.
4. Создание физической базы данных с использованием современных технологий и инструментов.
5. Проведение тестирования для оценки работоспособности системы.

Важной частью разработки является проектирование базы данных, которая должна обеспечивать надежное хранение данных, минимизацию ошибок и высокую производительность при выполнении запросов.

Результаты данной работы могут быть использованы в транспортной отрасли для создания сервисов, улучшающих пользовательский опыт, сокращающих время на планирование маршрутов и покупку билетов, а также повышающих общую привлекательность общественного транспорта.

1. Обзор существующих аналогов

На белорусском рынке функционируют различные сервисы и приложения, предоставляющие информацию о маршрутах общественного транспорта и предлагающие функции по покупке билетов. Эти системы различаются по своей функциональности, доступности и пользовательскому опыту. Наибольшую популярность среди пользователей приобрели такие решения, как «Атлас», «Яндекс.Транспорт» и «Infobus». Эти сервисы обеспечивают доступ к данным о городских и междугородних маршрутах, а также предлагают функции, упрощающие процесс планирования поездок. Однако каждый из них обладает как преимуществами, так и недостатками, которые влияют на удобство использования и уровень удовлетворенности пользователей. Их анализ позволяет выявить ключевые аспекты, которые требуют улучшений, и сформировать требования для создания новой системы, способной интегрировать лучшие черты существующих решений.

* 1. Атлас

«Атлас» — это белорусский сервис, предназначенный для предоставления информации о междугородних и международных автобусных маршрутах, а также для покупки билетов. Приложение ориентировано исключительно на дальние маршруты и не включает информацию о городском транспорте, что определяет его узкую специализацию [1].

Функциональность:

1. Отображение расписаний междугородних и международных автобусных рейсов.
2. Возможность поиска маршрутов по заданным параметрам (город отправления, город прибытия, дата).
3. Покупка билетов онлайн с интеграцией локальных платежных систем, таких как ЕРИП.
4. Информация о перевозчиках и условиях поездки (например, количество остановок и время в пути).

Преимущества:

1. Удобство для дальних поездок. Приложение охватывает практически все междугородние и международные маршруты, что делает его удобным инструментом для планирования поездок внутри Беларуси и за ее пределы.
2. Интеграция с местными платежными системами. Возможность оплаты через ЕРИП позволяет белорусским пользователям быстро и удобно покупать билеты.
3. Простота использования. Интерфейс приложения упрощает поиск маршрутов и процесс покупки билетов.
4. Надежность информации. Сервис предоставляет актуальные расписания и данные о перевозчиках, что повышает доверие пользователей.

Недостатки:

1. Отсутствие данных о городском транспорте. Приложение ориентировано исключительно на дальние маршруты и не предоставляет информации о городских автобусах, маршрутных такси или троллейбусах.
2. Устаревший интерфейс. Дизайн приложения не соответствует современным требованиям, что делает его менее привлекательным для пользователей.
3. Нет данных в реальном времени. Приложение не предоставляет информацию о текущем местоположении автобусов, изменениях маршрутов или задержках, что затрудняет использование в случаях отклонений от расписания.
4. Ограниченность мультимодальных маршрутов. Пользователи не могут комбинировать различные виды транспорта (например, поезд и автобус) для построения комплексных маршрутов.

Таким образом, «Атлас» представляет собой надежное решение для междугородних и международных автобусных маршрутов, но его ограниченность в функционале и отсутствие данных в реальном времени снижают удобство использования. При разработке нового сервиса стоит учесть положительные аспекты «Атласа», такие как интеграция с локальными платежными системами и простота поиска маршрутов, а также устранить его недостатки путем добавления данных о городском транспорте, визуализации маршрутов на карте и функций мониторинга движения автобусов в реальном времени.

* 1. Яндекс.Транспорт

«Яндекс.Транспорт» — это приложение, разработанное компанией «Яндекс», предназначенное для предоставления информации о городском общественном транспорте в режиме реального времени. Оно активно используется в Беларуси, особенно в крупных городах, таких как Минск. Основное назначение сервиса — помочь пользователям планировать поездки на общественном транспорте с учетом актуального положения транспортных средств [2].

Функциональность:

1. Отображение маршрутов городского транспорта (автобусы, троллейбусы, трамваи, маршрутные такси).
2. Информация о реальном местоположении транспорта с использованием GPS-данных.
3. Прогнозирование времени прибытия транспорта на остановку.
4. Визуализация маршрутов на карте, включая остановки и конечные точки.
5. Поиск маршрутов с пересадками.

Преимущества:

1. Информация в реальном времени. Приложение показывает текущее местоположение транспортных средств, что позволяет пользователям избегать длительного ожидания на остановках.
2. Удобный интерфейс. Простота использования и интуитивная навигация делают сервис доступным для широкого круга пользователей.
3. Отображение маршрутов на карте. Карта с маршрутом и остановками помогает пользователям визуализировать путь и правильно оценить расстояние между точками.
4. Поддержка городского транспорта. Широкое покрытие включает автобусы, троллейбусы и маршрутки, что делает приложение универсальным для жителей и гостей города.
5. Адаптация для крупных городов. Приложение особенно хорошо работает в городах с большим количеством маршрутов и высокой плотностью транспорта, таких как Минск.

Недостатки:

1. Нет функции покупки билетов. Приложение предоставляет только информацию о маршрутах и расписаниях, но не позволяет приобретать билеты на транспорт.
2. Отсутствие междугородних маршрутов. Сервис ориентирован исключительно на городской транспорт и не включает информацию о междугородних или международных рейсах.
3. Зависимость от GPS-данных. В случаях сбоя GPS или отсутствия сигнала данные о местоположении транспорта могут быть недостоверными или отсутствовать вовсе.
4. Проблемы с региональным охватом. Приложение в основном поддерживает крупные города, такие как Минск и областные центры, но для небольших городов и поселков его функциональность ограничена.

«Яндекс.Транспорт» является мощным инструментом для городских пассажиров, предоставляя удобный доступ к информации о маршрутах и времени прибытия транспорта. Однако отсутствие функций, связанных с покупкой билетов и интеграцией междугородних маршрутов, ограничивает его применение. При разработке новой системы можно учесть преимущества «Яндекс.Транспорт», такие как данные в реальном времени, визуализация маршрутов на карте и удобный интерфейс, а также устранить его недостатки путем добавления функций для работы с междугородним транспортом и интеграции с системами оплаты билетов.

* 1. Infobus

«Infobus» — это международная платформа, предоставляющая информацию о междугородных и международных автобусных маршрутах. В Беларуси сервис пользуется популярностью среди тех, кто планирует поездки по стране и за ее пределы. «Infobus» позволяет пользователям искать и бронировать билеты на автобусные рейсы, предоставляя информацию о маршрутах, расписаниях и перевозчиках.

Функциональность:

1. Поиск междугородних и международных автобусных маршрутов с возможностью указания точки отправления и назначения.
2. Информация о расписаниях и стоимости билетов на различные маршруты.
3. Возможность бронирования и покупки билетов через интернет, включая выбор места в автобусе.
4. Интеграция с платежными системами для оплаты билетов.
5. Сравнение предложений разных перевозчиков по ценам и условиям поездки.
6. Отзывы пользователей о перевозчиках и рейсах.

Преимущества:

1. Широкий выбор маршрутов. Infobus охватывает не только белорусские междугородные маршруты, но и международные автобусные рейсы, что делает его удобным сервисом для дальних поездок, как по Беларуси, так и за рубеж.
2. Удобство поиска и бронирования билетов. Платформа позволяет пользователям искать маршруты, сравнивать цены и бронировать билеты на выбранный рейс, что значительно упрощает процесс планирования поездок.
3. Интеграция с платежными системами. Возможность оплаты билетов через популярные международные и локальные платежные системы, такие как банковские карты, PayPal и другие. Это дает пользователям гибкость в выборе метода оплаты.
4. Прозрачность условий. Пользователи могут ознакомиться с условиями поездки, маршрутами и временем в пути до момента покупки, что помогает избежать неприятных сюрпризов. Также можно читать отзывы о перевозчиках, что позволяет сделать осознанный выбор.

Недостатки:

1. Отсутствие информации о городском транспорте. «Infobus» ориентирован исключительно на междугородний и международный транспорт, в отличие от сервисов, таких как «Яндекс.Транспорт», которые предоставляют информацию о городских маршрутах.
2. Ограниченная доступность в некоторых регионах. Несмотря на широкое покрытие маршрутов, сервис может не охватывать все малые города и населенные пункты, что ограничивает его доступность для жителей сельских регионов.
3. Отсутствие информации в реальном времени. В отличие от сервисов, которые предоставляют информацию о текущем местоположении транспорта, «Infobus» не имеет функции отслеживания автобусов в реальном времени, что может быть неудобно при изменениях расписания или задержках.
4. Усложненная навигация. Несмотря на наличие удобного поиска маршрутов, интерфейс платформы может быть сложным для пользователей, не привыкших к онлайн-бронированию, особенно для людей старшего возраста.

«Infobus» является надежным и удобным сервисом для планирования междугородних и международных поездок на автобусах, предоставляя пользователю возможность легко найти маршруты, сравнить цены и забронировать билеты. Однако его функциональность ограничена только дальними рейсами и не включает в себя информацию о городском транспорте или данные о текущем положении транспорта. При разработке нового агрегатора можно учесть сильные стороны «Infobus», такие как простота бронирования и широкий выбор маршрутов, а также устранить его недостатки путем добавления данных в реальном времени и информации о городских маршрутах.

1. Функциональные требования и инструменты разработки
   1. Функциональные требования к системе

Основной задачей разрабатываемого агрегатора автобусных маршрутов и билетов является создание эффективного и удобного сервиса для пользователей, позволяющего искать, бронировать и оплачивать билеты на междугородные и международные автобусные рейсы. Для реализации этого необходимо определить ключевые функциональные требования, которые будут обеспечивать необходимую функциональность системы.

Основные функциональные требования к приложению:

1. Поиск маршрутов и рейсов. Пользователи должны иметь возможность искать доступные автобусные маршруты по точке отправления и назначения, а также по времени отправления и прибытия. Необходимо реализовать фильтрацию маршрутов по различным параметрам, таким как стоимость, время в пути, перевозчик и другие.
2. Бронирование билетов. Система должна позволять пользователю выбрать и забронировать билеты на автобус, с возможностью выбора места в автобусе. Ожидается, что пользователи смогут просматривать доступные места и бронировать их через онлайн-платформу.
3. Оплата билетов. Система должна поддерживать онлайн-оплату с использованием различных методов, таких как банковские карты, электронные кошельки и другие способы оплаты. Поддержка валютных операций для международных маршрутов.
4. Управление учетными записями пользователей. Пользователи должны иметь возможность регистрироваться в системе, редактировать свои данные и просматривать историю своих поездок. В систему должны быть встроены механизмы аутентификации и авторизации, включая возможность восстановления пароля и подтверждения учетной записи.
5. Система лояльности. Разработка системы лояльности, в которой пользователи смогут накапливать баллы за покупки билетов и использовать их для получения скидок. Применение различных уровней лояльности с уникальными скидками и бонусами для пользователей, в зависимости от накопленных баллов.
6. Система отзывов и рейтингов. Пользователи должны иметь возможность оставлять отзывы о поездках и перевозчиках, а также оценивать качество обслуживания. Система должна включать рейтинг перевозчиков и маршрутов на основе пользовательских отзывов.
7. Поддержка нескольких языков. Для обеспечения удобства пользователей из разных стран, система должна поддерживать несколько языков, включая русский, белорусский и английский.
8. Интеграция с внешними сервисами. Возможность интеграции с другими сервисами для получения актуальной информации о рейсах и ценах на билеты, а также для работы с дополнительными функциями, такими как маршруты городского транспорта, а также с внешними платежными системами.
9. Адаптивность интерфейса. Система должна поддерживать адаптивный интерфейс, который будет корректно отображаться на мобильных устройствах, планшетах и ПК, обеспечивая удобное взаимодействие с сервисом.

Таким образом, для эффективной работы агрегатора автобусных маршрутов и билетов необходимо обеспечить комплексную функциональность, включающую поиск и фильтрацию маршрутов, возможность бронирования и оплаты билетов, управление учетными записями пользователей и систему лояльности. Важным аспектом является также внедрение системы отзывов и рейтингов, поддержка нескольких языков и интеграция с внешними сервисами. Все эти требования направлены на создание удобного и высокоэффективного сервиса для пользователей, который будет предоставлять актуальную информацию, обеспечивать удобство бронирования и оплаты билетов, а также поддерживать пользователей в процессе взаимодействия с системой.

* 1. Выбор СУБД

Для реализации проекта агрегатора автобусных маршрутов и билетов был выбран подход, основанный на использовании реляционной базы данных (СУБД), так как предполагается работа с большими объемами структурированных данных, которые логично представлять в виде таблиц, связанных между собой. Реляционная модель данных идеально подходит для хранения информации о пользователях, маршрутах, транзакциях и бронированиях, поскольку она обеспечивает четкую организацию данных и позволяет эффективно их обрабатывать [4].

Требования к СУБД:

1. Поддержка работы с большими объемами данных. В системе будет храниться большое количество данных о пользователях, рейсах, маршрутах и транзакциях. Важно, чтобы СУБД обеспечивала высокую производительность при работе с большими объемами информации, а также быстрое выполнение операций вставки, обновления и удаления данных.
2. Поддержка транзакций. Транзакции играют важную роль в системе, особенно в части бронирования и оплаты билетов. Важно, чтобы СУБД гарантировала целостность данных в случае ошибок или сбоев, а также поддерживала механизмы отката транзакций. Система должна удовлетворять ACID-свойствам (атомарность, согласованность, изолированность, долговечность) для обеспечения надежности работы с финансовыми операциями.
3. Высокая производительность при выполнении сложных запросов. Важной частью системы является возможность быстрого поиска и фильтрации маршрутов по различным параметрам: времени отправления, стоимости, перевозчику и т.д. Необходимо, чтобы СУБД обеспечивала высокую производительность при выполнении сложных запросов с множеством условий.
4. Поддержка масштабируемости и расширяемости. Система должна быть готова к масштабированию, так как в будущем может потребоваться поддержка большего количества пользователей, маршрутов и дополнительных функциональностей. Важно, чтобы СУБД позволяла эффективно увеличивать объемы данных без потери производительности.
5. Возможности для интеграции с внешними сервисами. В системе будет интеграция с внешними сервисами, например, для осуществления платежей или получения актуальной информации о рейсах. Поэтому СУБД должна поддерживать возможность подключения и взаимодействия с внешними API и системами.

Для реализации проекта агрегатора автобусных маршрутов и билетов были рассмотрены несколько популярных реляционных СУБД. Каждый из этих вариантов обладает своими преимуществами и недостатками, которые следует учитывать при выборе оптимального решения для системы. В процессе выбора СУБД учитывались такие факторы, как поддержка работы с большими объемами данных, транзакциями, производительность и масштабируемость. Ниже подробно рассматриваются четыре СУБД, которые были анализированы для данного проекта.

* + 1. PostgreSQL

PostgreSQL — это одна из самых мощных и гибких реляционных СУБД с открытым исходным кодом. Ее история начинается в 1986 году, когда профессор Майкл Стоунбрейкер из Университета Калифорнии в Беркли начал разработку системы под названием POSTGRES, которая была предназначена для исследования теоретических концепций в области реляционных баз данных. В 1996 году POSTGRES был переименован в PostgreSQL, и с тех пор эта СУБД развивается как мощный инструмент с открытым исходным кодом, который активно используется для решения задач, связанных с хранением и обработкой больших объемов данных [5].

Преимущества:

1. Высокая производительность при работе с большими объемами данных и сложными запросами. PostgreSQL предоставляет отличную производительность при работе с большими объемами данных и сложными запросами. Это достигается за счет использования индексации, оптимизации запросов и эффективной работы с большими таблицами. В контексте проекта агрегатора автобусных маршрутов и билетов, где необходимо обрабатывать большое количество маршрутов, транзакций и запросов, PostgreSQL обеспечит быструю и стабильную работу даже при высоких нагрузках.
2. PostgreSQL полностью поддерживает транзакции и гарантирует соблюдение ACID-свойств. Это особенно важно для системы бронирования билетов, где каждая транзакция должна быть выполнена точно и корректно, чтобы избежать ошибок в данных, например, при одновременной попытке забронировать одно и то же место двумя пользователями. С помощью транзакций обеспечивается целостность данных, что делает систему надежной и безопасной.
3. Возможность работы с сложными типами данных. PostgreSQL поддерживает работу с широким спектром сложных типов данных, включая JSON, массивы, хранимые процедуры и пользовательские типы данных. Это позволяет эффективно хранить и манипулировать неструктурированными данными, такими как информация о маршрутах, местах в автобусах и прочее. В контексте агрегатора, это может быть полезно для динамичного хранения данных, таких как расписания, информация о бронированиях и другие параметры, которые могут варьироваться.
4. Поддержка расширений для географических данных и полнотекстового поиска. PostgreSQL имеет встроенную поддержку расширений, таких как PostGIS, для работы с географическими данными. Это расширение идеально подходит для систем, которые работают с картами и маршрутами, как в случае с агрегатором автобусных маршрутов. Возможность выполнения полнотекстового поиска также позволяет значительно улучшить функциональность поиска по маршрутам и маршрутам перевозчиков, что делает систему более удобной для пользователей.
5. Бесплатная и открытая лицензия. PostgreSQL распространяется по лицензии Open Source, что означает отсутствие платы за лицензию и возможность свободного использования, модификации и распространения кода. Это снижает затраты на лицензирование, что особенно важно для стартапов и компаний с ограниченными бюджетами. Бесплатность PostgreSQL делает его доступным для широкого круга пользователей и предоставляет большую гибкость в разработке системы.

Недостатки:

1. Требует высокой квалификации для настройки и оптимизации. PostgreSQL — это мощная СУБД, которая требует определенных знаний и навыков для оптимальной настройки и эксплуатации. Например, необходимо умело настраивать индексы, параллельные запросы и систему репликации для достижения наилучших результатов. Без достаточного опыта в настройке и управлении СУБД возможны проблемы с производительностью и надежностью системы.
2. Сложность развертывания и настройки по сравнению с другими СУБД. В отличие от более простых СУБД, таких как MySQL или SQLite, развертывание PostgreSQL может быть более сложным. Это связано с множеством настроек, параметров и возможностей, которые нужно учитывать при первоначальной настройке базы данных. Настройка масштабируемости, репликации и резервного копирования также может потребовать значительных усилий и времени для достижения нужного уровня эффективности и безопасности.

Таким образом, несмотря на более сложное развертывание и необходимость в высококвалифицированных специалистах для настройки, PostgreSQL представляет собой отличное решение для проектов, которые требуют обработки больших объемов данных и сложных операций, таких как агрегатор автобусных маршрутов и билетов.

* + 1. MySQL

MySQL — это одна из самых популярных и широко используемых реляционных систем управления базами данных с открытым исходным кодом. История MySQL началась в 1994 году, когда шведская компания MySQL AB, основанная Михаэлем Видениусом и Альфом Ларссоном, начала разрабатывать эту СУБД как решение для работы с большими объемами данных, ориентированное на интернет-приложения. В 2008 году компанию MySQL AB приобрела корпорация Sun Microsystems, а спустя два года — Oracle Corporation. Несмотря на смену владельцев, MySQL продолжает оставаться одной из самых популярных СУБД в мире [6].

Преимущества:

1. Простота в использовании и настройке. MySQL известен своей простотой в установке и настройке. Это особенно важно для разработчиков, которым необходимо быстро развернуть систему и начать работать с базой данных. В отличие от более сложных СУБД, таких как PostgreSQL, MySQL не требует глубокой настройки и специфичных знаний для начальной конфигурации. Это делает его хорошим выбором для стартапов и небольших проектов, где важна скорость развертывания.
2. Хорошая поддержка репликации и масштабируемости. MySQL предлагает надежную систему репликации, которая позволяет создавать резервные копии данных и эффективно распределять нагрузку между несколькими серверами. Это критически важно для обеспечения высокой доступности и масштабируемости системы. Репликация помогает в создании системы с высокой отказоустойчивостью, что также улучшает производительность при высоких нагрузках. В контексте агрегатора автобусных маршрутов и билетов, MySQL может эффективно справляться с ростом числа пользователей и объемом данных, обеспечивая бесперебойную работу при увеличении нагрузки.
3. Широкая поддержка сообщества и разработчиков. MySQL имеет большое и активное сообщество разработчиков, что облегчает решение проблем и ускоряет процесс поиска решений для возникающих вопросов. Большое количество готовых библиотек, инструментов и документации позволяет быстро адаптироваться к работе с базой данных. Это снижает затраты на поддержку и делает систему более доступной для разработчиков с разным уровнем подготовки.

Недостатки:

1. Меньшая производительность для работы с большими объемами данных по сравнению с PostgreSQL. Несмотря на хорошую производительность MySQL, он уступает PostgreSQL в работе с большими объемами данных и сложными запросами. PostgreSQL лучше подходит для обработки больших наборов данных, сложных выборок и транзакций, что делает его более подходящим для систем, работающих с крупными базами данных и требующих высокой производительности. В случае с агрегатором автобусных маршрутов, где необходимо эффективно обрабатывать множество рейсов и бронирований, MySQL может проявить себя хуже при выполнении сложных запросов, таких как поиск по нескольким критериям или расчет стоимости.
2. Ограниченная поддержка сложных типов данных и некоторых расширений. MySQL имеет ограниченную поддержку для работы с более сложными типами данных, такими как JSON или массивы, по сравнению с PostgreSQL. Также в MySQL отсутствует поддержка ряда расширений, таких как PostGIS для географических данных или полнотекстового поиска на уровне PostgreSQL. Для проекта агрегатора автобусных маршрутов и билетов это может стать ограничением, если потребуется хранить и обрабатывать данные, которые не подходят для традиционных реляционных типов данных (например, географическая информация о маршрутах или данные в формате JSON).

Таким образом, MySQL представляет собой отличное решение для простых и средних по сложности проектов, где важна высокая доступность, масштабируемость и быстрая настройка. Однако для более сложных проектов, таких как агрегатор автобусных маршрутов и билетов, где требуется высокая производительность при работе с большими объемами данных и поддержка сложных типов данных, PostgreSQL может быть более подходящим выбором.

* + 1. Oracle Database

Oracle Database — это одна из самых мощных и широко используемых коммерческих реляционных систем управления базами данных, разработанная компанией Oracle Corporation. История Oracle началась в 1977 году, когда Ларри Эллисон, Боб Минтер и Эд Оутс основали компанию Relational Software Inc., которая позже была переименована в Oracle Corporation. В 1979 году Oracle представила первую коммерческую реляционную СУБД, которая использовала язык запросов SQL (Structured Query Language). С тех пор Oracle стал лидером на рынке корпоративных СУБД, с особым акцентом на высокую производительность, надежность и масштабируемость, что сделало Oracle Database стандартом для крупных организаций и финансовых учреждений [7].

Преимущества:

1. Высокая производительность и отличная масштабируемость. Oracle Database предоставляет высокую производительность, особенно для крупных корпоративных приложений, работающих с большими объемами данных. Система способна обрабатывать огромные нагрузки и работать с большими базами данных, что делает ее идеальной для систем, которые требуют высокой скорости обработки данных при выполнении сложных транзакций и запросов. Это преимущество особенно важно для крупных агрегаторов с большим количеством пользователей и сложной структурой данных.
2. Oracle Database известна своей надежной поддержкой транзакций и соблюдением ACID-свойств, что гарантирует целостность данных. Это особенно важно в контексте системы бронирования, где важно обеспечить, чтобы данные о бронированиях и оплатах были точными и не были потеряны при сбоях системы. Также поддержка репликации данных позволяет создавать высокодоступные решения, обеспечивая бесперебойную работу системы.
3. Широкий спектр функциональных возможностей для бизнеса. Oracle Database предоставляет множество инструментов для бизнес-анализа, работы с большими данными, машинного обучения и аналитики. Это делает систему универсальным инструментом для предприятий, которые могут использовать эти функции для углубленного анализа данных и построения бизнес-отчетности. Для проекта агрегатора автобусных маршрутов это может быть полезно, если система будет расширяться с внедрением аналитики для прогнозирования спроса на билеты или изучения поведения пользователей.

Недостатки:

1. Высокая стоимость лицензирования. Одним из главных недостатков Oracle Database является высокая стоимость лицензирования, что делает ее дорогим решением для малых и средних бизнесов. Стоимость лицензий и ежегодной поддержки может быть значительным бременем для стартапов и небольших компаний, особенно если они не используют все возможности СУБД. В контексте проекта агрегатора автобусных маршрутов и билетов это может означать большие затраты на эксплуатацию системы, которые могут быть не оправданы для небольшого бизнеса.
2. Требует значительных ресурсов для развертывания и эксплуатации. Развертывание и эксплуатация Oracle Database требуют значительных вычислительных и человеческих ресурсов. Это включает в себя не только покупку и установку серверов, но и поддержку на протяжении всего срока эксплуатации. Сложная настройка и оптимизация базы данных могут потребовать высококвалифицированных специалистов, что увеличивает затраты на техническую поддержку. Для небольших проектов это может быть чрезмерным, особенно если задачи можно решить с помощью более легких и дешевых СУБД.
3. Не является бесплатной. В отличие от PostgreSQL или MySQL, Oracle Database является платной СУБД с ограниченной бесплатной версией. Стоимость лицензирования может быть высока, что увеличивает затраты на проект и делает решение менее привлекательным для стартапов и малых компаний. Для агрегации автобусных маршрутов и билетов, где высокая стоимость лицензии может не быть оправдана функциональностью, доступной в других СУБД, использование Oracle Database может быть нецелесообразным.

Таким образом, несмотря на многочисленные преимущества Oracle Database, такие как высокая производительность, поддержка транзакций и масштабируемость, для проекта агрегатора автобусных маршрутов, ориентированного на малый и средний бизнес, эта СУБД может быть слишком дорогой и ресурсоемкой. В данном случае выбор более легковесной и менее затратной СУБД, такой как PostgreSQL или MySQL, может быть более оправдан.

* + 1. SQLite

SQLite — это легковесная реляционная система управления базами данных, которая была впервые выпущена в 2000 году. Ее создал Ричард Хипп, который разработал SQLite как встраиваемую СУБД, ориентированную на простоту использования и компактность. SQLite была спроектирована так, чтобы быть максимально легкой, быстро внедряемой и без необходимости в отдельном сервере баз данных [8].

Преимущества:

1. Простота интеграции и использования. SQLite — это серверная СУБД, которая не требует настройки отдельного серверного компонента. Это делает ее особенно удобной для интеграции в приложения. Для небольших проектов или мобильных приложений SQLite является отличным выбором, так как она не требует особых усилий для развертывания, что значительно ускоряет процесс разработки. Важно отметить, что база данных SQLite реализуется в виде одного файла, что упрощает ее использование и переносимость.
2. Легковесность и возможность использования на мобильных устройствах. SQLite занимает мало места на диске и использует минимальные ресурсы для работы, что делает ее отличным выбором для мобильных приложений и небольших проектов. Эта СУБД активно используется в мобильных операционных системах (например, в Android и iOS), что подтверждает ее легковесность и способность работать в условиях ограниченных ресурсов. Для простых систем бронирования и учета, которые не требуют сложных операций с большими объемами данных, SQLite может быть вполне достаточна.

Недостатки:

1. Ограниченная масштабируемость и производительность для работы с большими объемами данных. SQLite не предназначена для работы с высокими нагрузками и большими объемами данных. Она не обладает возможностями масштабирования, которые присутствуют в более сложных СУБД, таких как PostgreSQL или MySQL. Это ограничение проявляется в снижении производительности при увеличении количества пользователей или данных. Для систем с интенсивным использованием, таких как агрегатор автобусных маршрутов, где необходимо обрабатывать множество транзакций и запросов, SQLite может не быть достаточно эффективной, особенно когда речь идет о масштабировании системы.
2. Отсутствие полноценной поддержки транзакций на уровне предприятия. Несмотря на то, что SQLite поддерживает транзакции, она не предназначена для использования в крупных предприятиях с высокими требованиями к целостности и надежности данных. SQLite ограничена в функционале управления параллельными транзакциями и в обработке больших потоков данных. Это может быть критическим недостатком для систем, таких как бронирование билетов и обработка платежей, где требуется высокая степень целостности данных и возможности для обеспечения надежности при многопользовательской работе

На основе анализа различных СУБД было принято решение использовать PostgreSQL для реализации базы данных агрегатора автобусных маршрутов и билетов. Это решение обусловлено высокой производительностью PostgreSQL, поддержкой транзакций, гибкостью работы с данными и расширениями, а также бесплатной лицензией. Эти характеристики обеспечат эффективную работу системы с большими объемами данных и сложными запросами, а также гарантируют целостность и надежность при обработке операций с билетами и оплатой.

1. Проектирование базы данных

Проектирование базы данных является важным этапом в разработке системы агрегатора автобусных маршрутов и билетов, так как правильная структура данных определяет эффективность работы системы, обеспечивая быструю обработку запросов, целостность и безопасность данных. В рамках этого раздела будут рассмотрены этапы проектирования, включая инфологическую и даталогическую модели, модель миграции данных, а также требования к входным данным и форматы их хранения.

* 1. Инфологическая модель

Инфологическая модель — это этап проектирования базы данных, на котором происходит детальное описание структуры данных, их взаимосвязей и организационных единиц системы. В рамках этого этапа создается концептуальная модель, отражающая основные объекты и связи между ними. Инфологическая модель не зависит от конкретной реализации в СУБД, она служит основой для дальнейшего проектирования и реализации базы данных, позволяя понять, как данные будут структурированы и взаимодействовать в системе.

В данной работе инфологическая модель базы данных отражает структуру и связи между сущностями, необходимыми для разработки агрегатора автобусных маршрутов и билетов. Основными сущностями являются пользователи, маршруты, билеты, транзакции лояльности и различные связанные с ними данные. Важно, чтобы модель была гибкой и могла эффективно обрабатывать запросы, касающиеся маршрутов, бронирований, лояльности и поддержки пользователей [9].

Инфологическая модель базы данных включает в себя следующие сущности и их атрибуты:

1. Пользователь (User). Атрибуты: имя пользователя, хеш пароля, Дата регистрации, роль (связь с сущностью «Роль»), уровень лояльности (связь с сущностью «Уровень лояльности»).
2. Роль (Role). Атрибуты: название роли, описание роли.
3. Уровень лояльности (Loyalty Level). Атрибуты: название уровня, требуемые баллы, процент скидки, описание уровня.
4. Профиль пользователя (User Profile). Атрибуты: имя, фамилия, электронная почта, телефонный номер, дата рождения, адрес.
5. Баллы лояльности (Loyalty Points). Атрибуты: количество баллов, дата последнего обновления.
6. Транзакция лояльности (Loyalty Transaction). Атрибуты: дата транзакции, количество баллов, описание, тип транзакции (связь с сущностью «Тип транзакции»).
7. Тип транзакции лояльности (Loyalty Transaction Type). Атрибуты: название типа, описание.
8. Местоположение (Location). Атрибуты: название местоположения.
9. Тип транспорта (Transport Type). Атрибуты: название типа транспорта, описание.
10. Перевозчик (Carrier). Атрибуты: название перевозчика, юридический адрес, дата регистрации, электронная почта, телефонный номер.
11. Транспорт (Transport). Атрибуты: модель, цвет, вместимость, перевозчик (связь с сущностью «Перевозчик»).
12. Удобства транспорта (Transport Feature). Атрибуты: название удобства, описание.
13. Связь между транспортом и удобствами (Transport Feature Link). Атрибуты: дата добавления, статус активности.
14. Маршрут (Route). Атрибуты: цена, расстояние, ожидаемое время в пути, общее количество мест, доступные места, время отправления, время прибытия, местоположение отправления (связь с сущностью «Местоположение»), местоположение прибытия (связь с сущностью «Местоположение»), транспорт (связь с сущностью «Транспорт»), водитель (связь с сущностью «Пользователь»).
15. Статус билета (Ticket Status). Атрибуты: название статуса, описание статуса.
16. Бронирование билета (Ticket Booking). Атрибуты: количество забронированных мест, статус бронирования (связь с сущностью «Статус билета»), пользователь (связь с сущностью «Пользователь»), маршрут (связь с сущностью «Маршрут»).
17. Оценка билета (Ticket Rating). Атрибуты: рейтинг, комментарий, дата создания, бронирование билета (связь с сущностью «Бронирование билета»).
18. Статус запроса в поддержку (Support Request Status). Атрибуты: название статуса, описание.
19. Запрос в поддержку (Support Request). Атрибуты: дата создания, дата обновления, тема запроса, описание, пользователь (связь с сущностью «Пользователь»), статус запроса (связь с сущностью «Статус запроса в поддержку»), бронирование билета (связь с сущностью «Бронирование билета»).
20. Сообщение из чата поддержки (Support Message). Атрибуты: сообщение, дата отправки, отправитель (связь с сущностью «Пользователь»), запрос в поддержку (связь с сущностью «Запрос в поддержку»).
21. Часто задаваемые вопросы (FAQ). Атрибуты: вопрос, ответ, категория, дата создания, дата обновления.
22. Уровень логирования (Log Level). Атрибуты: название уровня, описание.
23. Конфиденциальные логи (Sensitive Log). Атрибуты: сообщение лога, дата создания, уровень логирования (связь с сущностью «Уровень логирования»).
24. Статус уведомления (Notification Status). Атрибуты: название статуса, описание.
25. Уведомления (Notifications). Атрибуты: сообщение, дата создания, дата отправки, статус уведомления (связь с сущностью «Статус уведомления»).

Таким образом, инфологическая модель для системы агрегатора автобусных маршрутов и билетов предоставляет понятное описание сущностей и их связей. Эта модель не включает технических аспектов хранения данных, но служит основой для последующего проектирования и реализации базы данных в выбранной СУБД.

* 1. Даталогическая модель

Даталогическая модель описывает структуру данных на уровне базы данных, в том числе типы данных, ограничения и связи между таблицами. Она отвечает за определение точных типов данных для каждого атрибута сущности, установление ограничений на значения и обеспечение целостности данных. В данном разделе представлена детальная спецификация всех сущностей, их атрибутов и типов данных, которые будут использоваться в базе данных системы. Модель строится на основе инфологической модели, уточняя типы данных и устанавливая точные форматы хранения информации для эффективного функционирования системы [10].

Описание таблиц и их атрибутов:

1. Роль (Role):

* id (SERIAL) — уникальный идентификатор роли;
* name (VARCHAR(50)) — имя роли, должно быть уникальным;
* description (TEXT) — описание роли.

1. Уровень лояльности (Loyalty Level):

* id (SERIAL) — уникальный идентификатор уровня лояльности;
* level\_name (VARCHAR(50)) — название уровня, должно быть уникальным;
* required\_points (INT) — количество баллов, необходимых для достижения этого уровня.;
* discount\_percentage (DECIMAL(5, 2)) — процент скидки, который предоставляется на этом уровне;
* description (TEXT) — описание уровня лояльности.

1. Пользователь (User):

* id (SERIAL) — уникальный идентификатор пользователя;
* username (VARCHAR(50)) — имя пользователя, уникальное;
* password\_hash (TEXT) — хеш пароля;
* registration\_date (TIMESTAMP) — дата регистрации пользователя;
* role\_id (INT) — внешнй ключ, ссылающийся на сущность «Роль»;
* current\_loyalty\_level\_id (INT) — внешний ключ, ссылающийся на сущность «Уровень лояльности».

1. Профиль пользователя (User Profile):

* id (SERIAL) — уникальный идентификатор профиля;
* first\_name (VARCHAR(50)) — имя пользователя;
* last\_name (VARCHAR(50)) — фамилия пользователя;
* email (VARCHAR(100)) — электронная почта, уникальная;
* phone\_number (VARCHAR(15)) — телефонный номер, уникальный;
* birth\_date (DATE) — дата рождения;
* address (TEXT) — адрес пользователя;
* user\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь».

1. Баллы лояльности (Loyalty Points):

* id (SERIAL) — уникальный идентификатор записи о баллах;
* points (INT) — количество баллов;
* last\_updated (TIMESTAMP) — дата последнего обновления баллов;
* user\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь».

1. Тип транзакции лояльности (Loyalty Transaction Type):

* id (SERIAL) — уникальный идентификатор типа транзакции;
* type\_name (VARCHAR(20)) — название типа транзакции, уникальное;
* description (TEXT) — описание типа транзакции;

1. Транзакция лояльности (Loyalty Transaction):

* id (SERIAL) — уникальный идентификатор транзакции;
* transaction\_date (TIMESTAMP) — дата транзакции;
* points (INT) — количество баллов, добавленных или вычтенных;
* description (TEXT) — описание транзакции;
* transaction\_type\_id (INT) — внешний ключ, ссылающийся на сущность «Тип транзакции»;
* user\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь».

1. Местоположение (Location):

* id (SERIAL) — уникальный идентификатор местоположения;
* name (VARCHAR(100)) — название местоположения, уникальное.

1. Тип транспорта (Transport Type):

* id (SERIAL) — уникальный идентификатор типа транспорта;
* name (VARCHAR(50)) — название типа транспорта, уникальное;
* description (TEXT) — описание типа транспорта.

1. Перевозчик (Carrier):

* id (SERIAL) — уникальный идентификатор перевозчика;
* name (VARCHAR(100)) — название перевозчика;
* legal\_address (TEXT) — юридический адрес перевозчика;
* registration\_date (TIMESTAMP) — дата регистрации перевозчика;
* email (VARCHAR(100)) — электронная почта, уникальная;
* phone\_number (VARCHAR(15)) — телефонный номер, уникальный.

1. Транспорт (Transport):

* id (SERIAL) — уникальный идентификатор транспорта;
* transport\_type\_id (INT) — внешний ключ, ссылающийся на сущность «Тип транспорта»;
* model (VARCHAR(100)) — модель транспорта;
* color (VARCHAR(30)) — цвет транспорта;
* capacity (INT) — вместимость транспорта;
* carrier\_id (INT) — внешний ключ, ссылающийся на сущность «Перевозчик».

1. Удобства транспорта (Transport Feature):

* id (SERIAL) — уникальный идентификатор удобства;
* feature\_name (VARCHAR(100)) — название удобства;
* description (TEXT) — описание удобства.

1. Связь между транспортом и удобствами (Transport Feature Link):

* id (SERIAL) — уникальный идентификатор связи;
* added\_at (TIMESTAMP) — дата добавления удобства в транспорт;
* is\_active (BOOLEAN) — статус активности связи;
* transport\_id (INT) — внешний ключ, ссылающийся на сущность «Транспорт»;
* transport\_feature\_id (INT) — внешний ключ, ссылающийся на сущность «Удобства транспорта».

1. Маршрут (Route):

* id (SERIAL) — уникальный идентификатор маршрута;
* price (DECIMAL(10, 2)) — цена маршрут;
* distance\_km (DECIMAL(10, 2)) — расстояние маршрута;
* estimated\_time\_minutes (INT) — ориентировочное время в пути (в минутах);
* total\_seats (INT) — общее количество мест;
* available\_seats (INT) — доступные места;
* departure\_time (TIMESTAMP) — время отправления;
* arrival\_time (TIMESTAMP) — время прибытия;
* departure\_location\_id (INT) — внешний ключ, ссылающийся на сущность «Местоположение»;
* arrival\_location\_id (INT) — внешний ключ, ссылающийся на сущность «Местоположение»;
* transport\_id (INT) — внешний ключ, ссылающийся на сущность «Транспорт»;
* driver\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь».

1. Статус билета (Ticket Status):

* id (SERIAL) — уникальный идентификатор статуса билета;
* status\_name (VARCHAR(50)) — название статуса;
* description (TEXT) — описание статуса.

1. Бронирование билета (Ticket Booking):

* id (SERIAL) — уникальный идентификатор бронирования;
* seats\_booked (INT) — количество забронированных мест;
* user\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь»;
* route\_id (INT) — внешний ключ, ссылающийся на сущность «Маршрут»;
* status\_id (INT) — внешний ключ, ссылающийся на сущность «Статус билета».

1. Оценка билета (Ticket Rating):

* id (SERIAL) — уникальный идентификатор оценки;
* rating (INT) — рейтинг билета (от 1 до 5);
* comment (TEXT) — комментарий к билету;
* created\_at (TIMESTAMP) — дата создания;
* ticket\_booking\_id (INT) — внешний ключ, ссылающийся на сущность «Бронирование билета».

1. Статус запроса в поддержку (Support Request Status):

* id (SERIAL) — уникальный идентификатор статуса запроса;
* status\_name (VARCHAR(50)) — название статуса запроса;
* description (TEXT) — описание статуса.

1. Запрос в поддержку (Support Request):

* id (SERIAL) — уникальный идентификатор запроса;
* created\_at (TIMESTAMP) — дата создания запроса;
* updated\_at (TIMESTAMP) — дата последнего обновления;
* subject (VARCHAR(100)) — тема запроса;
* description (TEXT) — описание запроса;
* user\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь»;
* status\_id (INT) — внешний ключ, ссылающийся на сущность «Статус запроса в поддержку»;
* ticket\_booking\_id (INT) — внешний ключ, ссылающийся на сущность «Бронирование билета».

1. Сообщение из чата поддержки (Support Message):

* id (SERIAL) — уникальный идентификатор сообщения;
* message (TEXT) — текст сообщения;
* sent\_at (TIMESTAMP) — дата отправки сообщения;
* sender\_id (INT) — внешний ключ, ссылающийся на сущность «Пользователь»;
* support\_request\_id (INT) — внешний ключ, ссылающийся на сущность «Запрос в поддержку».

1. Часто задаваемые вопросы (FAQ):

* id (SERIAL) — уникальный идентификатор вопроса;
* question (TEXT) — вопрос;
* answer (TEXT) — ответ на вопрос;
* category (VARCHAR(50)) — категория вопроса;
* created\_at (TIMESTAMP) — дата создания;
* updated\_at (TIMESTAMP) — дата последнего обновления.

1. Уровень логирования (Log Level):

* id (SERIAL) — уникальный идентификатор уровня логирования;
* level\_name (VARCHAR(50)) — название уровня логирования;
* description (TEXT) — описание уровня.

1. Конфиденциальные логи (Sensitive Log):

* id (SERIAL) — уникальный идентификатор лога;
* log\_message (TEXT) — сообщение лога;
* created\_at (TIMESTAMP) — дата создания лога.
* log\_level\_id (INT) — внешний ключ, ссылающийся на сущность «Уровень логирования».

1. Статус уведомления (Notification Status):

* id (SERIAL) — уникальный идентификатор статуса уведомления;
* status\_name (VARCHAR(50)) — название статуса уведомления;
* description (TEXT) — описание статуса;

1. Уведомления (Notifications):

* id (SERIAL) — уникальный идентификатор уведомления;
* message (TEXT) — сообщение уведомления;
* created\_at (TIMESTAMP) — дата создания уведомления;
* sent\_at (TIMESTAMP) — дата отправки уведомления;
* status\_id (INT) — внешний ключ, ссылающийся на сущность «Статус уведомления».

Разработка даталогической модели является важным этапом проектирования базы данных, так как она обеспечивает основу для дальнейшей реализации системы. В данном разделе была представлена детальная структура данных, где для каждой сущности были определены атрибуты с точными типами данных и ограничениями, а также установлены связи между таблицами. Это позволяет обеспечить целостность и согласованность данных, предотвращая ошибки при взаимодействии с базой данных. Такая модель является основой для создания физической базы данных и дальнейшего развития системы. Правильное определение типов данных и отношений между сущностями значительно повышает производительность работы с данными и делает систему более надежной.

* 1. Модель миграции данных

В процессе реализации системы возможны ситуации, когда потребуется перенос данных из существующих источников (например, старой базы данных) в новую систему. Это может происходить по причине перехода на новую версию системы, интеграции с другими системами или улучшения структуры данных. Модель миграции данных для данного проекта будет включать в себя подготовку данных, трансформацию их в нужный формат и перенос в целевую систему, что гарантирует соответствие всех данных новым стандартам и структурам.

Миграция данных в данном контексте будет включать несколько ключевых этапов:

1. Анализ текущих данных. На первом этапе необходимо понять структуру текущих данных и определить, какие данные будут мигрированы. Для этого нужно провести анализ всех таблиц, которые будут переноситься, и их связи.
2. Извлечение данных. На этом этапе данные извлекаются из старой базы данных. Для каждой таблицы будут созданы отдельные запросы на извлечение данных. Извлечение может происходить с учетом определенных фильтров или периодичности. Также могут быть использованы инструменты для миграции данных, такие как Talend или Apache Nifi, для автоматизации процесса.
3. Трансформация данных. После извлечения данных необходимо привести их к требуемому формату. Например, трансформация может включать: преобразование типов данных, преобразование значений, маппинг сущностей между системами.
4. Загрузка данных в целевую систему. После того как данные будут извлечены и преобразованы, их необходимо загрузить в новую систему. Это будет включать выполнение SQL-запросов на вставку данных в целевые таблицы. Если данные поступают из разных источников, может быть использована автоматизация для загрузки их в соответствующие таблицы.
5. Пост-миграционный анализ и валидация. После загрузки данных важно провести валидацию на целостность и корректность: проверка наличия всех записей, сравнение количества записей в исходной и целевой таблице, проверка связей между таблицами (например, между маршрутами и перевозчиками или пользователями и их транзакциями).
6. Мониторинг и оптимизация. После миграции данных необходимо мониторить целевую систему, чтобы убедиться в ее стабильной работе. Оптимизация может включать индексирование таблиц, настройку кеширования и других механизмов для повышения производительности.

Миграция данных — это неотъемлемая часть любого проекта, связанного с изменением структуры или обновлением базы данных. Для успешной миграции данных необходимо внимательно продумать все  
этапы — от анализа текущих данных до загрузки и проверки целостности в новой системе. Правильная реализация модели миграции данных обеспечит бесперебойную работу системы, сохранность и доступность данных, что критически важно для пользователей и бизнеса.

1. Разработка базы данных

Разработка базы данных – это центральный элемент построения системы, обеспечивающий основу для хранения, обработки и анализа данных. На данном этапе проектирования осуществляется переход от логической модели данных, представляющей концептуальное описание системы, к физической модели, реализующей это описание на уровне реляционной базы данных.

Процесс разработки базы данных включает несколько этапов: проектирование структуры таблиц, реализация связей между ними, настройка ограничений целостности данных, создание индексов для ускорения запросов, разработка триггеров и хранимых процедур для автоматизации операций, а также внедрение механизмов защиты данных. В рамках системы агрегатора маршрутов база данных играет ключевую роль, так как содержит всю необходимую информацию о пользователях, маршрутах, билетах, точках отправления и прибытия, транспорте и водителях.

Разработка велась с использованием системы управления базами данных (СУБД) PostgreSQL, которая была выбрана за её высокую производительность, широкие возможности работы с реляционными данными, а также встроенные механизмы масштабирования и обеспечения безопасности.

* 1. Физическая модель базы данных

Физическая модель представляет собой описание структуры данных, которое реализуется в виде таблиц и связей между ними. Она учитывает требования к производительности системы, предотвращению избыточности данных, а также обеспечению согласованности и целостности.

В системе агрегатора предусмотрены следующие основные таблицы:

1. Таблица пользователей (user) – содержит данные для аутентификации пользователей, их роли и текущего уровня лояльности.
2. Таблица профилей пользователей (user\_profile) – включает персональные данные, такие как имя, фамилия, email и телефон.
3. Таблица маршрутов (route) – хранит данные о рейсах, их стоимости, длительности и доступности мест (рисунок 4.1).
4. Таблица точек отправления и прибытия (location) – содержит адреса и координаты остановок.
5. Таблица транспортных средств (transport) – описывает автобусы, их вместимость и характеристики.
6. Таблица водителей (driver) – хранит информацию о водителях, закрепленных за маршрутами.

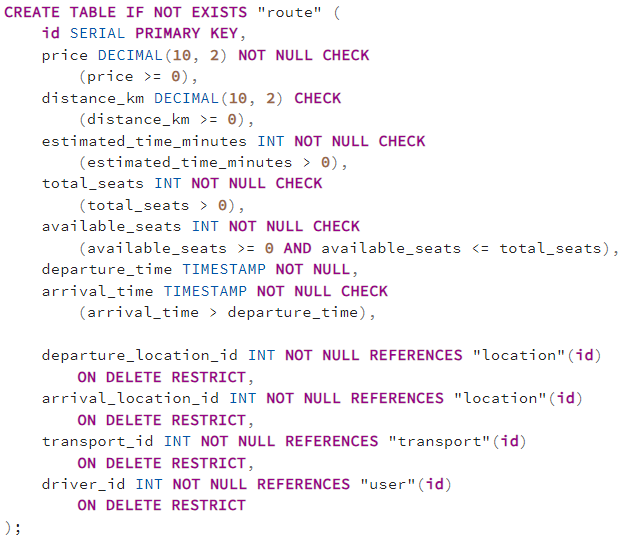


Рисунок 4.1 – Пример физической реализации таблицы маршрутов

* 1. Создание таблиц

Создание таблиц является ключевым шагом в процессе реализации физической модели базы данных. На данном этапе описывается структура каждой таблицы: её поля, их типы данных, ограничения, а также связи с другими таблицами.

Каждое поле таблицы имеет определенный тип данных. Например, числовые значения, такие как стоимость или расстояние, реализуются через тип NUMERIC, строки – через VARCHAR или TEXT, а даты и время – через TIMESTAMP. Для повышения производительности и целостности данных были также настроены ограничения на уровне таблиц:

1. Поля с обязательными значениями помечены как NOT NULL, что исключает возможность хранения записей с пустыми критически важными данными.
2. Первичные ключи (PRIMARY KEY) используются для идентификации каждой строки, обеспечивая уникальность каждой записи в пределах таблицы.
3. Внешние ключи (FOREIGN KEY) обеспечивают связь между таблицами, что позволяет поддерживать логические отношения между сущностями базы данных.
4. Уникальные ограничения (UNIQUE) предотвращают дублирование данных, гарантируя, что определенные столбцы будут содержать только уникальные значения.

На рисунке 4.2 приведен пример создания таблицы для хранения данных о заявках в службу поддержки. Она содержит уникальный идентификатор заявки, информацию о пользователе, отправившем заявку, тему и описание проблемы, статус обработки, а также временные метки создания и последнего обновления.

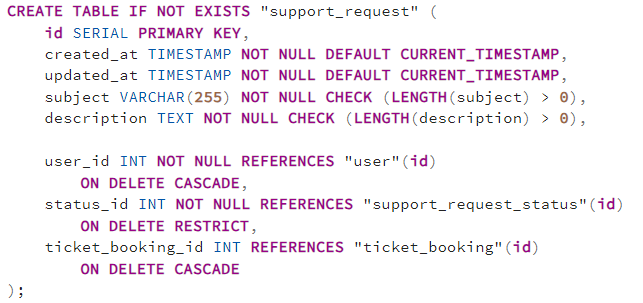


Рисунок 4.2 – Таблица запросов в поддержку

Создание таблиц является важным этапом разработки базы данных, так как правильная структура данных обеспечивает эффективное хранение и обработку информации в процессе работы системы. В листинге А.1 представлен код для создания всех таблиц базы данных, используемых в проекте.

* 1. Оптимизация производительности

Для обеспечения высокой производительности системы и быстрого отклика на запросы пользователей в базе данных были внедрены индексы. Индексы играют важную роль в оптимизации работы с большими объемами данных, позволяя существенно сократить время, необходимое для выполнения операций выборки, поиска, сортировки и фильтрации

Без использования индексов, при выполнении запросов к базе данных, система должна просматривать все строки таблицы, что значительно увеличивает время выполнения операций. Индексы позволяют ускорить этот процесс, создавая структуры данных, которые позволяют базе данных находить нужные записи значительно быстрее. В зависимости от типа индекса и структуры данных, поиск может быть выполнен за время, пропорциональное логарифму от количества строк, что значительно быстрее линейного поиска.

Кроме того, индексы играют ключевую роль при соединении таблиц. Когда нужно выполнить объединение данных из разных таблиц, индексы помогают быстро находить соответствующие записи, что значительно повышает скорость выполнения сложных запросов с множеством соединений.

В проекте использованы следующие типы индексов:

1. Индексы типа B-tree (B-дерево) — применяются для операций сортировки и фильтрации по числовым и строковым данным. Эти индексы эффективны для диапазонных запросов и часто используются в таких полях, как дата регистрации, время отправления и другие, где требуется сортировка или фильтрация по значениям.
2. Hash-индексы — используются для точного поиска по уникальным значениям. Они оптимальны для быстрого поиска по полям с уникальными значениями, таким как имя пользователя или электронная почта, где важен точный поиск без диапазонных операций.
3. Индексы GIN (Generalized Inverted Index) — предназначены для работы с текстовыми данными и позволяют эффективно искать по подстрокам или фрагментам текста. Эти индексы применяются для полей с большими текстовыми данными, такими как описание запросов, где требуется быстрый поиск по ключевым словам.

Таким образом, внедрение индексов в базу данных не только ускоряет выполнение базовых операций, но и способствует улучшению общей производительности системы, обеспечивая эффективное использование ресурсов при работе с большими объемами данных. Скрипт для создания индексов представлен в листинге А.2.

* 1. Триггеры и процедуры

В данном разделе описаны разработанные триггеры и процедуры, обеспечивающие автоматизацию операций в базе данных, контроль целостности данных, а также повышение уровня удобства и производительности системы. Код создания триггеров и процедур представлен в листинге А.3. В проекте реализованы следующие ключевые механизмы:

1. Автообновление доступных мест в маршруте. Для обеспечения актуальности данных о доступных местах в маршрутах при бронировании или отмене билетов был создан триггер *trg\_update\_available\_seats*, связанный с таблицей *ticket\_booking*. Функция *update\_available\_seats*, вызываемая этим триггером, автоматически уменьшает или увеличивает количество доступных мест в зависимости от операции (*INSERT* или *DELETE*). Такой подход исключает необходимость ручного обновления информации о маршрутах, минимизируя риск несоответствия данных.
2. Автообновление уровня лояльности пользователя. Для автоматического пересчета уровня лояльности пользователя в соответствии с накопленными баллами реализован триггер *trg\_update\_loyalty\_level*, связанный с обновлением таблицы *loyalty\_points*. Функция *update\_loyalty\_level* определяет соответствующий уровень лояльности и обновляет информацию в таблице *user*. Это позволяет поддерживать актуальность данных без дополнительных действий со стороны администратора системы.
3. Логирование изменений статуса запросов в поддержку. Для фиксации изменений статусов в таблице *support\_request* разработан триггер *trg\_log\_support\_request\_changes*. Функция *log\_support\_request\_changes* добавляет соответствующую запись в таблицу *sensitive\_log* с указанием идентификатора запроса, нового статуса и времени изменения. Это обеспечивает прозрачность и контроль за процессом обработки запросов пользователей.
4. Автоматическое создание транзакции лояльности. С целью автоматизации учета бонусных баллов при бронировании билетов был создан триггер *trg\_create\_loyalty\_transaction*. Функция *create\_loyalty\_transaction* добавляет запись в таблицу *loyalty\_transaction*, а также обновляет текущий баланс баллов пользователя в таблице *loyalty\_points*. Это облегчает учет бонусов и снижает нагрузку на операторов системы.
5. Проверка доступности мест перед бронированием. Для предотвращения превышения доступных мест на маршруте перед добавлением нового бронирования используется триггер *trg\_check\_seat\_availability*. Функция *check\_seat\_availability* проверяет наличие достаточного количества свободных мест и выбрасывает исключение, если требуемое количество мест недоступно. Такой механизм повышает надежность системы и предотвращает ошибки при обработке заказов.
6. Автоудаление старых логов. Для оптимизации объема данных и поддержания актуальности логов реализована процедура *cleanup\_old\_logs*, которая удаляет записи из таблицы *sensitive\_log*, созданные более 90 дней назад. Запуск данной процедуры может быть автоматизирован с использованием инструментов, таких как *pgAgent* или *cron*, что обеспечивает регулярное обслуживание базы данных без вмешательства администратора.

Каждый из описанных механизмов интегрирован в архитектуру системы, обеспечивая автоматизацию и снижение вероятности ошибок при работе с данными. Эти триггеры и процедуры повышают надежность базы данных и упрощают её администрирование.

* 1. Запросы

Этот раздел посвящен описанию ключевых запросов, разработанных для реализации функциональных возможностей системы. Запросы позволяют эффективно извлекать данные из базы для решения различных задач, таких как анализ пользовательской активности, управление бронированиями, отслеживание работы службы поддержки и другие. Они обеспечивают быстрый доступ к необходимой информации, упрощая работу с базой данных и повышая удобство использования системы.

В проекте реализованы следующие запросы:

1. Информация о пользователях с деталями их лояльности и ролями. Этот запрос извлекает данные о пользователях, включая их имя, электронную почту, текущий уровень лояльности, процент скидки и роль в системе. Он упрощает анализ данных о пользователях и их статусе.
2. История транзакций лояльности пользователя. Запрос отображает историю всех транзакций лояльности, совершенных пользователем. Результаты включают тип транзакции, количество начисленных или списанных баллов, дату и описание. Это удобно для отслеживания динамики начислений бонусных баллов.
3. Заказы билетов для определенного маршрута. Запрос предоставляет данные о бронированиях билетов для заданного маршрута. Включает информацию о пользователях, количестве забронированных мест и статусе билета. Это полезно для анализа заполненности маршрута.
4. Оценки пользователей для маршрутов. Данный запрос собирает оценки и комментарии пользователей о маршрутах. Он позволяет анализировать обратную связь и повышать качество предоставляемых услуг.
5. Сводка по транспортным средствам и их удобствам. Запрос предоставляет информацию о транспортных средствах, используемых в системе, и связанных с ними удобствах, включая их описание и модель транспорта.
6. Статистика по заявкам в поддержку. Этот запрос извлекает данные о заявках в поддержку, их статусах, темах, датах создания и пользователях, которые их отправили. Используется для мониторинга и анализа работы службы поддержки.
7. Пользователи с запросами в поддержку и сообщениями. Запрос объединяет информацию о запросах в поддержку и связанных с ними сообщениях, предоставляя данные о пользователях, отправивших запросы, и их деталях.
8. Статистика по нотификациям. Запрос извлекает информацию о нотификациях, их статусах, времени создания и отправки. Это позволяет отслеживать эффективность доставки уведомлений.
9. Полная информация по маршрутам. Запрос объединяет данные о маршрутах, транспортных средствах, отправных и конечных точках маршрута, а также информацию о водителе. Используется для анализа характеристик маршрутов и транспорта.

Эти запросы охватывают ключевые аспекты функциональности системы, предоставляя гибкие инструменты для анализа данных и принятия решений. Код запросов представлен в листинге А.4.

1. Тестирование работоспособности базы данных

В данном разделе представлены результаты тестирования базы данных, направленные на проверку корректности работы скриптов, созданных для реализации бизнес-логики. Все тесты были выполнены с использованием SQL-запросов, ошибки или успешное выполнение которых зафиксированы на скриншотах.

Тесты на соблюдение ограничений:

1. Проверка уникальности значений в таблице *user\_profile*. Для тестирования был выполнен запрос с повторяющимся значением *email.* Запрос завершится с ошибкой из-за нарушения ограничения уникальности (рисунок 5.1).

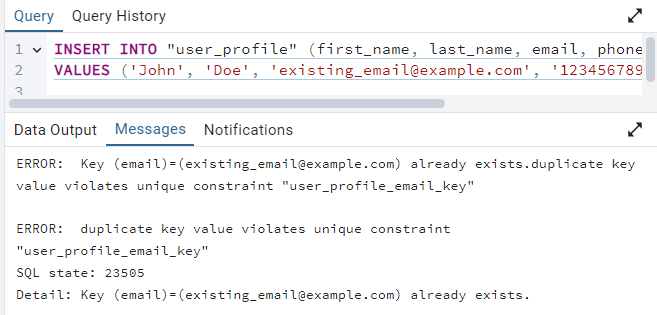


Рисунок 5.1 – Ошибка проверки уникальности

1. Проверка ссылочной целостности. Для тестирования работы внешнего ключа, который ссылается на таблицу *role*, был выполнен запрос на удаление записи из таблицы *role*, на которую ссылается поле *role\_id* в таблице *user*. Результат представлен на рисунке 5.2.

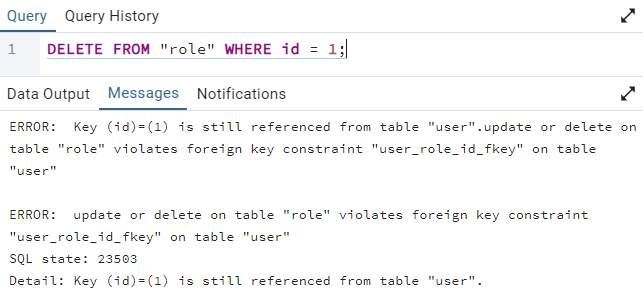


Рисунок 5.2 – Проверка ссылочной целостности

1. Проверка *CHECK* ограничения. Для проверки работы ограничения, запрещающего отрицательные значения для поля *price* таблицы *route*, был выполнен запрос, представленный на рисунке 5.3.

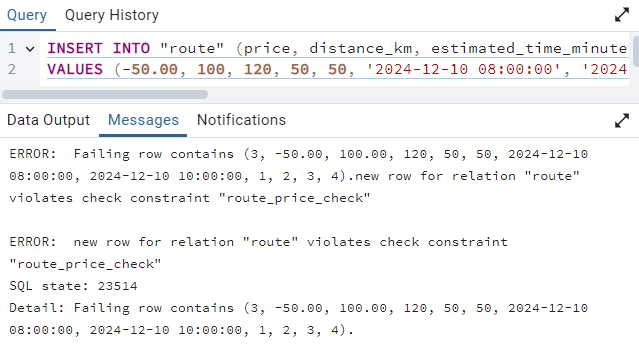


Рисунок 5.3 – Проверка ограничения

Для тестирования триггера для обновления доступных мест при бронировании был выполнен запрос на добавление бронирования пользователем двух билетов. В результате число доступных мест уменьшилось на 2 после выполнения команды *INSERT* (рисунок 5.4).

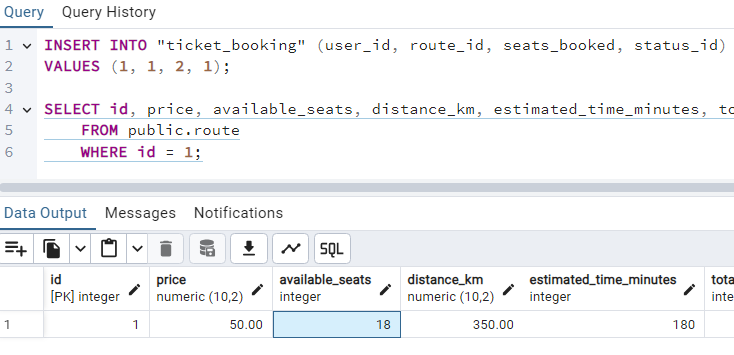


Рисунок 5.4 – Работа триггера на бронирование

Заключение

В процессе разработки системы агрегатора автобусных маршрутов и билетов был создан проект базы данных, обеспечивающий централизованное хранение, обработку и предоставление информации о маршрутах, расписаниях и возможностях покупки билетов. Важной частью разработки стало проектирование базы данных с использованием современных технологий и инструментов. Система базы данных была спроектирована с учетом всех требований к целостности, безопасности и производительности данных.

Проектирование базы данных было выполнено в несколько этапов: сначала были созданы инфологическая и даталогическая модели, что позволило точно определить структуру и взаимосвязи данных. После этого была разработана физическая модель базы данных с использованием современных технологий и инструментов. Созданная система обладает возможностью быстрого и эффективного извлечения данных, а также гарантирует минимизацию ошибок и высокую производительность при обработке запросов.

Также был проведен ряд тестов для оценки работоспособности базы данных, в том числе на ограничения, аномалии, а также тесты для процедур и триггеров. Тестирование подтвердило, что система успешно справляется с поставленными задачами, такими как создание, обновление и удаление записей, а также выполнение сложных запросов. Все тесты показали корректность работы системы в различных сценариях, что подтверждает высокую надежность разработанной базы данных.

Результаты данной работы могут найти практическое применение в транспортной отрасли. Созданный агрегатор маршрутов и билетов предлагает значительное улучшение в области удобства планирования поездок и покупки билетов, а также в повышении эффективности работы транспортных компаний. Это позволит пассажирам оперативно получать информацию о доступных маршрутах и забронировать билеты, что сокращает время на подготовку поездки и делает использование общественного транспорта более удобным и доступным. Система также способствует повышению привлекательности транспортных услуг, что в условиях цифровизации становится важным аспектом для развития отрасли.

Список литературных источников

1. Атлас [Электронный ресурс]. – Режим доступа: https://atlasbus.by/. – Дата доступа: 12.10.2024.
2. Яндекс Карты [Электронный ресурс]. – Режим доступа: https://yandex.by/maps. – Дата доступа: 12.10.2024.
3. Infobus [Электронный ресурс]. – Режим доступа: https://infobus.by/. – Дата доступа: 12.10.2024.
4. Коннолли, Т. Базы данных : проектирование, реализация и сопровождение. Теория и практика, 2-е изд. / Т. Коннолли, К. Бегг,   
   А. Страчан. – Пер. с англ. — М. : Издательский дом «Вильямс»,   
   2021. – 1120 с.
5. Новиков, Б.А. Основы технологий баз данных / Б.А. Новиков, Е. А. Горшкова, Н. Г. Графеева. – Москва : ДМК Пресс, 2020. – 582 с.
6. Дюбуа, П. MySQL. Сборник рецептов. / П. Дюбуа, Е. А. Горшкова, Н. Г. Графеева. – Москва : Символ-Плюс, 2017. – 491 с.
7. Том, К. Oracle для профессионалов / К. Том, Е. А. Горшкова, Н. Г. Графеева. – СПб. : ООО «ДиаСофтЮП», 2019. – 672 с.
8. Руководство по SQLite [Электронный ресурс]. – Режим доступа: https://metanit.com/sql/sqlite/. – Дата доступа: 28.11.2024.
9. Стружкин, Н. П. Базы данных: проектирование. Практикум : учебное пособие для вузов / Н. П. Стружкин, В. В. Годин. — Москва : Издательство Юрайт, 2024. — 291 с.
10. Осипов Д. Технологии проектирования баз данных. – М.: ДМК Пресс, 2019. – 498 с.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода

Листинг А.1 – Файл *ddl.tables.sql*

DROP SCHEMA public CASCADE;

CREATE SCHEMA public;

GRANT ALL ON SCHEMA public TO public;

-- Таблица ролей пользователей

CREATE TABLE IF NOT EXISTS "role" (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL UNIQUE,

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица уровня лояльности

CREATE TABLE IF NOT EXISTS "loyalty\_level" (

id SERIAL PRIMARY KEY,

level\_name VARCHAR(50) NOT NULL UNIQUE,

required\_points INT NOT NULL CHECK (required\_points >= 0),

discount\_percentage DECIMAL(5, 2) NOT NULL CHECK (discount\_percentage >= 0 AND discount\_percentage <= 50),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица пользователей

CREATE TABLE IF NOT EXISTS "user" (

id SERIAL PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(username) >= 3),

password\_hash TEXT NOT NULL CHECK (LENGTH(password\_hash) >= 60),

registration\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

role\_id INT NOT NULL REFERENCES "role"(id) ON DELETE RESTRICT,

current\_loyalty\_level\_id INT NOT NULL REFERENCES "loyalty\_level"(id) ON DELETE RESTRICT

);

-- Таблица профиля пользователя

CREATE TABLE IF NOT EXISTS "user\_profile" (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL CHECK (LENGTH(first\_name) > 0),

last\_name VARCHAR(50) NOT NULL CHECK (LENGTH(last\_name) > 0),

email VARCHAR(100) NOT NULL UNIQUE CHECK (POSITION('@' IN email) > 0),

phone\_number VARCHAR(15) NOT NULL UNIQUE CHECK (LENGTH(phone\_number) >= 10),

birth\_date DATE NOT NULL CHECK (birth\_date <= CURRENT\_DATE),

address TEXT CHECK (LENGTH(address) <= 500),

user\_id INT NOT NULL UNIQUE REFERENCES "user"(id) ON DELETE CASCADE

);

-- Таблица баллов лояльности

CREATE TABLE IF NOT EXISTS "loyalty\_points" (

id SERIAL PRIMARY KEY,

points INT NOT NULL DEFAULT 0 CHECK (points >= 0),

last\_updated TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

user\_id INT NOT NULL REFERENCES "user"(id) ON DELETE CASCADE

);

-- Таблица типов транзакции лояльности

CREATE TABLE IF NOT EXISTS "loyalty\_transaction\_type" (

id SERIAL PRIMARY KEY,

type\_name VARCHAR(20) NOT NULL UNIQUE CHECK (LENGTH(type\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица записей о транзакциях лояльности

CREATE TABLE IF NOT EXISTS "loyalty\_transaction" (

id SERIAL PRIMARY KEY,

transaction\_date TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

points INT NOT NULL CHECK (points <> 0),

description TEXT CHECK (LENGTH(description) <= 500),

transaction\_type\_id INT NOT NULL REFERENCES "loyalty\_transaction\_type"(id) ON DELETE RESTRICT,

user\_id INT NOT NULL REFERENCES "user"(id) ON DELETE CASCADE

);

-- Таблица отправной/конечной точки

CREATE TABLE IF NOT EXISTS "location" (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE CHECK (LENGTH(name) > 0)

);

-- Таблица типа транспорта

CREATE TABLE IF NOT EXISTS "transport\_type" (

id SERIAL PRIMARY KEY,

name VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица перевозчиков

CREATE TABLE IF NOT EXISTS "carrier" (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL CHECK (LENGTH(name) > 0),

legal\_address TEXT CHECK (LENGTH(legal\_address) <= 500),

registration\_date TIMESTAMP NOT NULL CHECK (registration\_date <= CURRENT\_TIMESTAMP),

email VARCHAR(100) NOT NULL UNIQUE CHECK (POSITION('@' IN email) > 0),

phone\_number VARCHAR(15) NOT NULL UNIQUE CHECK (LENGTH(phone\_number) >= 10)

);

-- Таблица транспорта

CREATE TABLE IF NOT EXISTS "transport" (

id SERIAL PRIMARY KEY,

model VARCHAR(100) CHECK (LENGTH(model) > 0),

color VARCHAR(30),

capacity INT NOT NULL CHECK (capacity > 0),

transport\_type\_id INT NOT NULL REFERENCES "transport\_type"(id) ON DELETE RESTRICT,

carrier\_id INT NOT NULL REFERENCES "carrier"(id) ON DELETE RESTRICT

);

-- Таблица удобств транспорта

CREATE TABLE IF NOT EXISTS "transport\_feature" (

id SERIAL PRIMARY KEY,

feature\_name VARCHAR(100) NOT NULL CHECK (LENGTH(feature\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица связи транспорта и удобств

CREATE TABLE IF NOT EXISTS "transport\_feature\_link" (

id SERIAL PRIMARY KEY,

added\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

is\_active BOOLEAN NOT NULL DEFAULT TRUE,

transport\_id INT NOT NULL REFERENCES "transport"(id) ON DELETE CASCADE,

transport\_feature\_id INT NOT NULL REFERENCES "transport\_feature"(id) ON DELETE CASCADE,

UNIQUE (transport\_id, transport\_feature\_id)

);

-- Таблица для маршрута

CREATE TABLE IF NOT EXISTS "route" (

id SERIAL PRIMARY KEY,

price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),

distance\_km DECIMAL(10, 2) CHECK (distance\_km >= 0),

estimated\_time\_minutes INT NOT NULL CHECK (estimated\_time\_minutes > 0),

total\_seats INT NOT NULL CHECK (total\_seats > 0),

available\_seats INT NOT NULL CHECK (available\_seats >= 0 AND available\_seats <= total\_seats),

departure\_time TIMESTAMP NOT NULL,

arrival\_time TIMESTAMP NOT NULL CHECK (arrival\_time > departure\_time),

departure\_location\_id INT NOT NULL REFERENCES "location"(id) ON DELETE RESTRICT,

arrival\_location\_id INT NOT NULL REFERENCES "location"(id) ON DELETE RESTRICT,

transport\_id INT NOT NULL REFERENCES "transport"(id) ON DELETE RESTRICT,

driver\_id INT NOT NULL REFERENCES "user"(id) ON DELETE RESTRICT

);

-- Таблица статусов билетов

CREATE TABLE IF NOT EXISTS "ticket\_status" (

id SERIAL PRIMARY KEY,

status\_name VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(status\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица бронирования билетов

CREATE TABLE IF NOT EXISTS "ticket\_booking" (

id SERIAL PRIMARY KEY,

seats\_booked INT NOT NULL CHECK (seats\_booked > 0),

user\_id INT NOT NULL REFERENCES "user"(id) ON DELETE CASCADE,

route\_id INT NOT NULL REFERENCES "route"(id) ON DELETE CASCADE,

status\_id INT NOT NULL REFERENCES "ticket\_status"(id) ON DELETE RESTRICT,

UNIQUE (user\_id, route\_id)

);

-- Таблица отзывов пользователя по билету

CREATE TABLE IF NOT EXISTS "ticket\_rating" (

id SERIAL PRIMARY KEY,

rating INT CHECK (rating >= 1 AND rating <= 5) NOT NULL,

comment TEXT CHECK (LENGTH(comment) <= 1000),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

ticket\_booking\_id INT NOT NULL UNIQUE REFERENCES "ticket\_booking"(id) ON DELETE CASCADE

);

-- Таблица для статуса запроса в поддержку

CREATE TABLE IF NOT EXISTS "support\_request\_status" (

id SERIAL PRIMARY KEY,

status\_name VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(status\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица для запроса в поддержку

CREATE TABLE IF NOT EXISTS "support\_request" (

id SERIAL PRIMARY KEY,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

subject VARCHAR(255) NOT NULL CHECK (LENGTH(subject) > 0),

description TEXT NOT NULL CHECK (LENGTH(description) > 0),

user\_id INT NOT NULL REFERENCES "user"(id) ON DELETE CASCADE,

status\_id INT NOT NULL REFERENCES "support\_request\_status"(id) ON DELETE RESTRICT,

ticket\_booking\_id INT REFERENCES "ticket\_booking"(id) ON DELETE CASCADE

);

-- Таблица сообщения из чата поддержки

CREATE TABLE IF NOT EXISTS "support\_message" (

id SERIAL PRIMARY KEY,

message TEXT NOT NULL CHECK (LENGTH(message) > 0),

sent\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

sender\_id INT NOT NULL REFERENCES "user"(id) ON DELETE RESTRICT,

support\_request\_id INT NOT NULL REFERENCES "support\_request"(id) ON DELETE CASCADE

);

-- Таблица для часто задаваемых вопросов

CREATE TABLE IF NOT EXISTS "faq" (

id SERIAL PRIMARY KEY,

question TEXT NOT NULL CHECK (LENGTH(question) > 0 AND LENGTH(question) <= 1000),

answer TEXT NOT NULL CHECK (LENGTH(answer) > 0 AND LENGTH(answer) <= 2000),

category VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(category) > 0),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

-- Таблица уровней логирования

CREATE TABLE IF NOT EXISTS "log\_level" (

id SERIAL PRIMARY KEY,

level\_name VARCHAR(20) NOT NULL UNIQUE CHECK (LENGTH(level\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица для хранения конфиденциальных логов

CREATE TABLE IF NOT EXISTS "sensitive\_log" (

id SERIAL PRIMARY KEY,

log\_message TEXT NOT NULL CHECK (LENGTH(log\_message) > 0),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

log\_level\_id INT NOT NULL REFERENCES "log\_level"(id) ON DELETE RESTRICT

);

-- Таблица статусов нотификаций

CREATE TABLE IF NOT EXISTS "notification\_status" (

id SERIAL PRIMARY KEY,

status\_name VARCHAR(50) NOT NULL UNIQUE CHECK (LENGTH(status\_name) > 0),

description TEXT CHECK (LENGTH(description) <= 500)

);

-- Таблица нотификаций

CREATE TABLE IF NOT EXISTS "notifications" (

id SERIAL PRIMARY KEY,

message TEXT NOT NULL CHECK (LENGTH(message) > 0 AND LENGTH(message) <= 1000),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

sent\_at TIMESTAMP,

status\_id INT NOT NULL REFERENCES "notification\_status"(id) ON DELETE RESTRICT

);

Листинг А.2 – Файл *ddl.indexes.sql*

-- Индексы для таблицы "user"

CREATE INDEX IF NOT EXISTS idx\_user\_username ON "user" USING hash (username);

CREATE INDEX IF NOT EXISTS idx\_user\_registration\_date ON "user" USING btree (registration\_date);

-- Индексы для таблицы "user\_profile"

CREATE INDEX IF NOT EXISTS idx\_user\_profile\_email ON "user\_profile" USING hash (email);

CREATE INDEX IF NOT EXISTS idx\_user\_profile\_phone\_number ON "user\_profile" USING hash (phone\_number);

CREATE INDEX IF NOT EXISTS idx\_user\_profile\_last\_name ON "user\_profile" USING btree (last\_name);

-- Индексы для таблицы "loyalty\_points"

CREATE INDEX IF NOT EXISTS idx\_loyalty\_points\_user\_id ON "loyalty\_points" USING btree (user\_id);

CREATE INDEX IF NOT EXISTS idx\_loyalty\_points\_last\_updated ON "loyalty\_points" USING btree (last\_updated);

-- Индексы для таблицы "route"

CREATE INDEX IF NOT EXISTS idx\_route\_departure\_time ON "route" USING btree (departure\_time);

CREATE INDEX IF NOT EXISTS idx\_route\_arrival\_time ON "route" USING btree (arrival\_time);

CREATE INDEX IF NOT EXISTS idx\_route\_departure\_location\_id ON "route" USING btree (departure\_location\_id);

CREATE INDEX IF NOT EXISTS idx\_route\_arrival\_location\_id ON "route" USING btree (arrival\_location\_id);

CREATE INDEX IF NOT EXISTS idx\_route\_price ON "route" USING btree (price);

CREATE INDEX IF NOT EXISTS idx\_route\_transport\_id ON "route" USING btree (transport\_id);

CREATE INDEX IF NOT EXISTS idx\_route\_driver\_id ON "route" USING btree (driver\_id);

-- Индексы для таблицы "ticket\_booking"

CREATE INDEX IF NOT EXISTS idx\_ticket\_booking\_user\_id ON "ticket\_booking" USING btree (user\_id);

CREATE INDEX IF NOT EXISTS idx\_ticket\_booking\_route\_id ON "ticket\_booking" USING btree (route\_id);

CREATE INDEX IF NOT EXISTS idx\_ticket\_booking\_status\_id ON "ticket\_booking" USING btree (status\_id);

-- Индексы для таблицы "support\_request"

CREATE INDEX IF NOT EXISTS idx\_support\_request\_user\_id ON "support\_request" USING btree (user\_id);

CREATE INDEX IF NOT EXISTS idx\_support\_request\_created\_at ON "support\_request" USING btree (created\_at);

CREATE INDEX IF NOT EXISTS idx\_support\_request\_status\_id ON "support\_request" USING btree (status\_id);

CREATE INDEX IF NOT EXISTS idx\_support\_request\_ticket\_booking\_id ON "support\_request" USING btree (ticket\_booking\_id);

-- Индексы для таблицы "support\_message"

CREATE INDEX IF NOT EXISTS idx\_support\_message\_support\_request\_id ON "support\_message" USING btree (support\_request\_id);

CREATE INDEX IF NOT EXISTS idx\_support\_message\_sender\_id ON "support\_message" USING btree (sender\_id);

-- Индексы для таблицы "transport"

CREATE INDEX IF NOT EXISTS idx\_transport\_transport\_type\_id ON "transport" USING btree (transport\_type\_id);

CREATE INDEX IF NOT EXISTS idx\_transport\_carrier\_id ON "transport" USING btree (carrier\_id);

CREATE INDEX IF NOT EXISTS idx\_transport\_capacity ON "transport" USING btree (capacity);

-- Индексы для таблицы "transport\_feature\_link"

CREATE INDEX IF NOT EXISTS idx\_transport\_feature\_link\_transport\_id ON "transport\_feature\_link" USING btree (transport\_id);

CREATE INDEX IF NOT EXISTS idx\_transport\_feature\_link\_feature\_id ON "transport\_feature\_link" USING btree (transport\_feature\_id);

-- Индексы для таблицы "notifications"

CREATE INDEX IF NOT EXISTS idx\_notifications\_sent\_at ON "notifications" USING btree (sent\_at);

CREATE INDEX IF NOT EXISTS idx\_notifications\_status\_id ON "notifications" USING btree (status\_id);

-- Индексы для таблицы "sensitive\_log"

CREATE INDEX IF NOT EXISTS idx\_sensitive\_log\_created\_at ON "sensitive\_log" USING btree (created\_at);

CREATE INDEX IF NOT EXISTS idx\_sensitive\_log\_log\_level\_id ON "sensitive\_log" USING btree (log\_level\_id);

-- Индексы для поиска по тексту

CREATE EXTENSION IF NOT EXISTS pg\_trgm;

CREATE INDEX IF NOT EXISTS idx\_support\_request\_description\_gin ON "support\_request" USING gin (description gin\_trgm\_ops);

Листинг А.3 – Файл *ddl.functions-triggers.sql*

-- Функция для обновления количества доступных мест

CREATE OR REPLACE FUNCTION update\_available\_seats()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

UPDATE route

SET available\_seats = available\_seats - NEW.seats\_booked

WHERE id = NEW.route\_id;

ELSIF TG\_OP = 'DELETE' THEN

UPDATE route

SET available\_seats = available\_seats + OLD.seats\_booked

WHERE id = OLD.route\_id;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Триггер для вызова функции обновления доступных мест

CREATE TRIGGER trg\_update\_available\_seats

AFTER INSERT OR DELETE ON ticket\_booking

FOR EACH ROW

EXECUTE FUNCTION update\_available\_seats();

-- Функция для обновления уровня лояльности пользователя

CREATE OR REPLACE FUNCTION update\_loyalty\_level()

RETURNS TRIGGER AS $$

DECLARE

new\_level\_id INT;

BEGIN

SELECT id INTO new\_level\_id

FROM loyalty\_level

WHERE required\_points <= (SELECT points FROM loyalty\_points WHERE user\_id = NEW.user\_id)

ORDER BY required\_points DESC

LIMIT 1;

UPDATE "user"

SET current\_loyalty\_level\_id = new\_level\_id

WHERE id = NEW.user\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер для вызова функции обновления уровня лояльности

CREATE TRIGGER trg\_update\_loyalty\_level

AFTER UPDATE ON loyalty\_points

FOR EACH ROW

EXECUTE FUNCTION update\_loyalty\_level();

-- Функция для логирования изменений в таблице support\_request

CREATE OR REPLACE FUNCTION log\_support\_request\_changes()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO sensitive\_log (log\_message, log\_level\_id, created\_at)

VALUES (

'Support request ID ' || NEW.id || ' status changed to ' || NEW.status\_id,

1, -- Assuming 1 corresponds to "INFO"

CURRENT\_TIMESTAMP

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер для логирования изменений статуса запроса в поддержку

CREATE TRIGGER trg\_log\_support\_request\_changes

AFTER UPDATE ON support\_request

FOR EACH ROW

WHEN (OLD.status\_id IS DISTINCT FROM NEW.status\_id)

EXECUTE FUNCTION log\_support\_request\_changes();

-- Функция для создания записи транзакции лояльности

CREATE OR REPLACE FUNCTION create\_loyalty\_transaction()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO loyalty\_transaction (transaction\_date, points, description, transaction\_type\_id, user\_id)

VALUES (CURRENT\_TIMESTAMP, NEW.seats\_booked \* 10, 'Booking points', 1, NEW.user\_id); -- Assuming 1 is the ID for "Purchase"

UPDATE loyalty\_points

SET points = points + (NEW.seats\_booked \* 10), last\_updated = CURRENT\_TIMESTAMP

WHERE user\_id = NEW.user\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер для создания транзакции лояльности при бронировании билета

CREATE TRIGGER trg\_create\_loyalty\_transaction

AFTER INSERT ON ticket\_booking

FOR EACH ROW

EXECUTE FUNCTION create\_loyalty\_transaction();

-- Функция для проверки доступности мест в маршруте перед бронированием

CREATE OR REPLACE FUNCTION check\_seat\_availability()

RETURNS TRIGGER AS $$

BEGIN

IF (SELECT available\_seats FROM route WHERE id = NEW.route\_id) < NEW.seats\_booked THEN

RAISE EXCEPTION 'Not enough available seats for route ID %', NEW.route\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Триггер для проверки доступности мест при добавлении бронирования

CREATE TRIGGER trg\_check\_seat\_availability

BEFORE INSERT ON ticket\_booking

FOR EACH ROW

EXECUTE FUNCTION check\_seat\_availability();

-- Функция для удаления логов старше 90 дней

CREATE OR REPLACE FUNCTION cleanup\_old\_logs()

RETURNS VOID AS $$

BEGIN

DELETE FROM sensitive\_log

WHERE created\_at < CURRENT\_DATE - INTERVAL '90 days';

END;

$$ LANGUAGE plpgsql;

Листинг А.4– Файл *dml.select-pool.sql*

-- 1. Информация о пользователях с деталями их лояльности и ролями

SELECT

u.id AS user\_id,

u.username,

u\_profile.first\_name,

u\_profile.last\_name,

u\_profile.email,

r.name AS role,

ll.level\_name AS loyalty\_level,

ll.discount\_percentage

FROM

"user" u

JOIN

"user\_profile" u\_profile ON u.id = u\_profile.user\_id

JOIN

"role" r ON u.role\_id = r.id

JOIN

"loyalty\_level" ll ON u.current\_loyalty\_level\_id = ll.id

ORDER BY

u.username;

-- 2. История транзакций лояльности пользователя

SELECT

lt.transaction\_date,

lt.points,

lt.description AS transaction\_description,

ltt.type\_name AS transaction\_type,

u.username AS user\_name

FROM

"loyalty\_transaction" lt

JOIN

"user" u ON lt.user\_id = u.id

JOIN

"loyalty\_transaction\_type" ltt ON lt.transaction\_type\_id = ltt.id

ORDER BY

lt.transaction\_date DESC;

-- 3. Заказы билетов для определенного маршрута

SELECT

tb.id AS booking\_id,

u.username,

tb.seats\_booked,

ts.status\_name AS ticket\_status

FROM

"ticket\_booking" tb

JOIN

"user" u ON tb.user\_id = u.id

JOIN

"ticket\_status" ts ON tb.status\_id = ts.id

JOIN

"route" r ON tb.route\_id = r.id

WHERE

r.id = 1; -- Замените на ID маршрута

-- 4. Оценки пользователей для маршрутов

SELECT

r.id AS route\_id,

r.departure\_time,

r.arrival\_time,

tr.rating,

tr.comment,

u.username AS reviewer

FROM

"ticket\_rating" tr

JOIN

"ticket\_booking" tb ON tr.ticket\_booking\_id = tb.id

JOIN

"route" r ON tb.route\_id = r.id

JOIN

"user" u ON tb.user\_id = u.id

ORDER BY

r.departure\_time, tr.created\_at DESC;

-- 5. Сводка по транспортным средствам и их удобствам

SELECT

t.model,

t.color,

tf.feature\_name,

tf.description AS feature\_description

FROM

"transport" t

JOIN

"transport\_feature\_link" tfl ON t.id = tfl.transport\_id

JOIN

"transport\_feature" tf ON tfl.transport\_feature\_id = tf.id

WHERE

t.carrier\_id = 1 -- Замените на ID перевозчика

ORDER BY

t.model, tf.feature\_name;

-- 6. Статистика по заявкам в поддержку

создал запрос.

SELECT

sr.id AS support\_request\_id,

sr.subject,

sr.created\_at AS request\_date,

sr.status\_id,

srs.status\_name AS request\_status,

u.username AS user\_name

FROM

"support\_request" sr

JOIN

"support\_request\_status" srs ON sr.status\_id = srs.id

JOIN

"user" u ON sr.user\_id = u.id

ORDER BY

sr.created\_at DESC;

-- 7. Пользователи с запросами в поддержку и сообщениями

SELECT

u.username AS user\_name,

sr.subject,

sm.message AS support\_message,

sm.sent\_at AS message\_sent\_at

FROM

"support\_request" sr

JOIN

"support\_message" sm ON sr.id = sm.support\_request\_id

JOIN

"user" u ON sr.user\_id = u.id

ORDER BY

sm.sent\_at DESC;

-- 8. Статистика по нотификациям

SELECT

n.message,

n.created\_at AS notification\_created,

n.sent\_at AS notification\_sent,

ns.status\_name AS notification\_status

FROM

"notifications" n

JOIN

"notification\_status" ns ON n.status\_id = ns.id

ORDER BY

n.sent\_at DESC;

-- 9. Полная информация по маршрутам

SELECT

r.id AS route\_id,

r.departure\_time,

r.arrival\_time,

r.price,

r.distance\_km,

r.estimated\_time\_minutes,

dl.name AS departure\_location,

al.name AS arrival\_location,

t.model AS transport\_model,

t.color AS transport\_color,

u.username AS driver

FROM

"route" r

JOIN

"location" dl ON r.departure\_location\_id = dl.id

JOIN

"location" al ON r.arrival\_location\_id = al.id

JOIN

"transport" t ON r.transport\_id = t.id

JOIN

"user" u ON r.driver\_id = u.id

ORDER BY

r.departure\_time;

ПРИЛОЖЕНИЕ Б  
(обязательное)  
Схема базы данных

ПРИЛОЖЕНИЕ В  
(обязательное)  
Ведомость документов