МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема «Реализация базы данных железнодорожного вокзала с использованием средств мониторинга состояния СУБД»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Тимошенко

подпись, дата

**Руководитель**

Ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кантарович В. С.

должность, учен. степень, ученое звание подпись, дата

Допущен к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ Кантарович В. С.

подпись дата инициалы и фамилия

Минск 2023

**Содержание**

[Введение 6](#_Toc135084487)

[1. Аналитический обзор литературы по теме проекта 7](#_Toc135084488)

[1.1 Сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих по данной тематике технических научных решений 7](#_Toc135084489)

[1.2 Аналитический обзор аналогов 8](#_Toc135084490)

[1.2.1 Аналог «Белорусская железная дорога» 8](#_Toc135084491)

[1.2.2 Аналог «Intercity.pl» 10](#_Toc135084492)

[1.3 Изучение требований, определение вариантов использования 10](#_Toc135084493)

[1.3.1 Определение основных функциональных требований к базе данных 10](#_Toc135084494)

[1.3.2 Определение нефункциональных требований к базе данных 11](#_Toc135084495)

[1.3.3 Определение вариантов использования 11](#_Toc135084496)

[1.4 Вывод 12](#_Toc135084497)

[2. Разработка архитектуры проекта 13](#_Toc135084498)

[2.1 Обобщенная структура управления приложением 13](#_Toc135084499)

[2.2 Диаграммы UML, взаимосвязь всех компонентов 14](#_Toc135084500)

[2.3 Описание информационых объектов и ограничений целостности 15](#_Toc135084501)

[2.3.1 Таблица Stations 15](#_Toc135084502)

[2.3.2 Таблица Routes 16](#_Toc135084503)

[2.3.3 Таблица Schedule 16](#_Toc135084504)

[2.3.4 Таблица Trains 16](#_Toc135084505)

[2.3.5 Таблица Vans 17](#_Toc135084506)

[2.3.6 Таблица Passengers 17](#_Toc135084507)

[2.3.7 Таблица Payments 17](#_Toc135084508)

[2.3.8 Таблица Tickets 17](#_Toc135084509)

[2.3.9 Таблица Stations\_routes 18](#_Toc135084510)

[2.4 Вывод 18](#_Toc135084511)

[3. Разработка модели базы данных 19](#_Toc135084512)

[3.1 Создание необходимых объектов 19](#_Toc135084513)

[3.1.1 Функции 19](#_Toc135084514)

[3.1.2 Процедуры 20](#_Toc135084515)

[3.1.3 Представления 21](#_Toc135084516)

[3.1.4 Триггеры 21](#_Toc135084517)

[3.1.5 Пользовательские типы данных 23](#_Toc135084518)

[3.1.6 Индексы 23](#_Toc135084519)

[3.2 Заполнение таблицы 100000 строк 23](#_Toc135084520)

[3.3 Описание технологии 25](#_Toc135084521)

[3.4 Вывод 26](#_Toc135084522)

[4. Установка, настройка и использование Oracle 19c 27](#_Toc135084523)

[4.1 Создание ролей для разграничения доступа 27](#_Toc135084524)

[4.2 Описание процедур импорта и экспорта 28](#_Toc135084525)

[4.3 Тестирование производительности базы данных 29](#_Toc135084526)

[4.4 Вывод 31](#_Toc135084527)

[5. Тестирование, проверка работоспособности и анализ полученных результатов. 32](#_Toc135084528)

[5.1. Тестирование клиентской части. 32](#_Toc135084529)

[5.2. Тестирование части менеджера 33](#_Toc135084530)

[5.3. Вывод. 35](#_Toc135084531)

[6. Руководство по использованию программного продукта 36](#_Toc135084532)

[6.1 Установка приложения 36](#_Toc135084533)

[6.2 Работа с приложением 36](#_Toc135084534)

[6.3.1 Сторона менеджера 36](#_Toc135084535)

[6.3.2 Сторона пользователя 38](#_Toc135084536)

[6.3. Вывод 39](#_Toc135084537)

[Заключение 40](#_Toc135084538)

[Список использованных литературных источников 41](#_Toc135084539)

[Приложение 42](#_Toc135084540)

[Приложение А 42](#_Toc135084541)

[Приложение Б 44](#_Toc135084542)

[Приложение В 55](#_Toc135084543)

**Введение**

В нашем современном мире имеется огромный спрос пользователей на хранилища информации, также называемые базами данных, которые могли бы постоянно работать и позволять удобно пользоваться хранимой инфромацией, также необходимо учитывать необходимость в использовании средств мониторинга состояния хранилища данных. Особенно важным становится использование баз данных в железнодорожной отрасли для обработки и хранения данных о состоянии железнодорожных вокзалов и поездов.

В связи с актуальностью данной работы целью курсового проекта является разработка и реализация базы данных Oracle и интерфейса для неё.

К задачам курсового проекта относится: аналитический обзор литературы по теме проекта, изучение требований; определение вариантов использования; анализ и проектирование модели данных; описание информационных объектов и ограничений целостности; создание необходимых объектов; импорт и экспорт данных; описание требуемой технологии; тестирование производительности; формирование вывода по каждому разделу; заключение, включающее вывод по проделанной работе.

Для базы данных обычно требуется комплексное программное обеспечение, которое называется системой управления базами данных (СУБД). СУБД служит интерфейсом между базой данных и пользователями или программами, предоставляя пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией. СУБД обеспечивает контроль и управление данными, позволяя выполнять различные административные операции, такие как мониторинг производительности, настройка, а также резервное копирование и восстановление.

Спектр применения систем управления базами данных на сегодняшний день практически необъятен – базы данных используются в интернете, в производстве, в промышленности, в маркетинге, в мобильных устройствах, в финансовой и банковской сферах, на телевидении, в телекоммуникациях и рекламе.

1. Аналитический обзор литературы по теме проекта

1.1 Сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих по данной тематике технических научных решений

Существует множество методов решения этой задачи, которые могут быть классифицированы по следующим параметрам:

1. Производительность: скорость выполнения запросов и эффективность использования ресурсов.
2. Надежность: способность системы работать без сбоев и сбоев, а также ее способность восстановиться после сбоя.
3. Гибкость: способность системы адаптироваться к изменениям требований и удовлетворять различным потребностям пользователей.
4. Расширяемость: возможность системы расширяться и добавлять новые функциональные возможности.

В теоретической литературе активно обсуждаются различные модели данных, в том числе реляционная, иерархическая и сетевая модели. Однако в контексте реализации базы данных железнодорожного вокзала реляционная модель является наиболее предпочтительной, так как она хорошо подходит для описания связей между различными элементами информационной системы, такими как станции, поезда, расписания, билеты и т.д.

Реляционная модель базы данных – это стандарт, который используется для описания и управления данными в большинстве современных баз данных. Для реализации базы данных железнодорожного вокзала можно использовать различные СУБД, такие как Oracle, Microsoft SQL Server, PostgreSQL и другие.

В литературе также описываются различные методы обработки данных, такие как SQL-запросы и NoSQL-базы данных. В контексте данной темы SQL-запросы являются более предпочтительными, так как они хорошо подходят для организации транзакций, которые важны для системы бронирования билетов и работы с расписаниями.

Для реализации базы данных железнодорожного вокзала с использованием средств мониторинга состояния СУБД можно использовать специализированные программные средства, такие как системы мониторинга и управления базами данных. Одним из примеров такой системы является Nagios, которая позволяет мониторить состояние базы данных, отслеживать ошибки и предупреждать об их возникновении.

Другой подход - использование инструментов профилирования кода, которые позволяют выявить узкие места в работе приложения и оптимизировать его.

Среди существующих технических решений можно выделить информационную систему "Железнодорожный транспорт", разработанную компанией "ООО Интерконтиненталь". Система предназначена для автоматизации работы железнодорожного транспорта и включает в себя модуль управления железнодорожными станциями и вокзалами, а также модуль мониторинга и управления базами данных.

Другим примером является система управления железнодорожным транспортом "ТРАНСДИЗЕЛЬ", разработанная ООО "Трансдизель". Система включает в себя модули управления поездами, грузами, вагонами и другими ресурсами железнодорожного транспорта. В системе используется СУБД PostgreSQL, а для мониторинга состояния базы данных применяется средство мониторинга pgBadger.

1.2 Аналитический обзор аналогов

В современном мире, когда скорость и точность играют важную роль, транспортные компании становятся все более популярными. Одной из главных задач таких компаний является надежная доставка клиентов в места назначения.

1.2.1 Аналог «Белорусская железная дорога»

Белорусская железная дорога (БЧ) использует комплексную информационную систему (КИС) "Единая транспортная информационно-телекоммуникационная система" (ЕТИС) для управления информацией о поездах, расписании, билетах, грузах и других аспектах железнодорожного транспорта.

КИС ЕТИС включает в себя ряд баз данных, которые хранят информацию о различных аспектах железнодорожной деятельности. Некоторые из них:

База данных "Поезда": содержит информацию о поездах, их маршрутах, времени отправления и прибытия, количество вагонов и другие данные, необходимые для управления движением поездов.

База данных "Расписание": содержит информацию о расписании движения поездов, включая время отправления и прибытия, остановки и другие детали.

База данных "Билеты": содержит информацию о продаже и бронировании билетов, включая номера мест, стоимость билетов, данные о пассажирах и другие детали.

База данных "Грузы": содержит информацию о грузах, их маршрутах, весе, объеме, стоимости и других деталях, необходимых для управления грузовыми перевозками.

База данных "Техническое обслуживание": содержит информацию о техническом состоянии поездов, их ремонте и техническом обслуживании.

База данных "Персонал": содержит информацию о сотрудниках железнодорожной компании, их квалификации, должностях, зарплатах и других деталях.

Данные в этих базах хранятся в различных форматах, включая структурированные данные в виде таблиц, текстовые файлы, аудио- и видеозаписи, фотографии и другие форматы. Доступ к базам данных осуществляется через специализированные программные интерфейсы и приложения, которые обеспечивают управление и обработку информации на различных этапах железнодорожной деятельности.

В качестве аналога рассмотрим приложение белорусской железной дороги [1], которое представлено на рисунке 1.1.

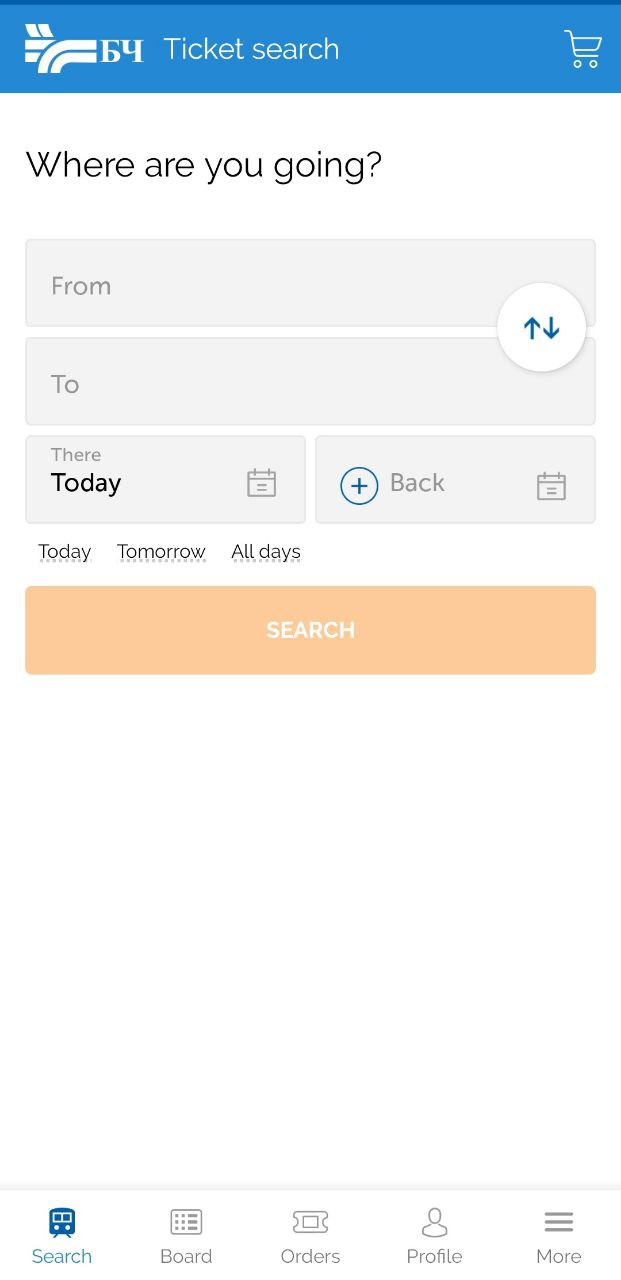


Рисунок 1.1 – Приложение «БЧ. Мой поезд»

Данное приложение позволяет из любой точки страны, где имеется доступ в интернет, забронировать билет на поезд. Пример представлен на рисунке 1.2.

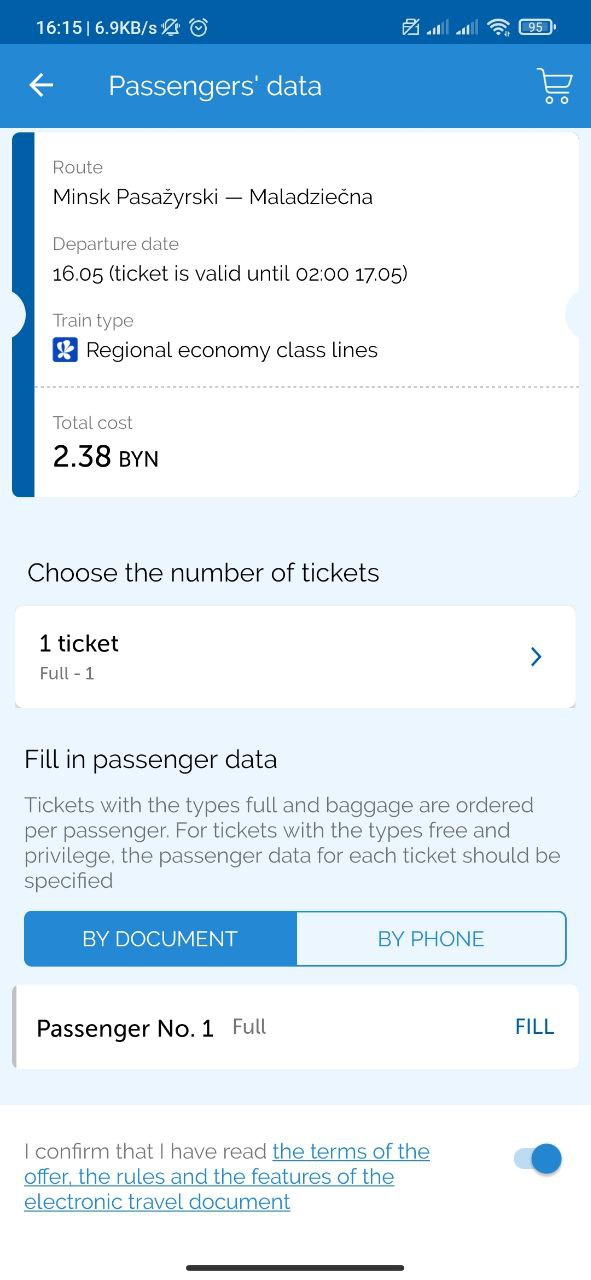


Рисунок 1.2 – Бронирование билета в приложении «БЧ. Мой поезд»

Также приложение позволяет просмотреть расписание рейсов онлайн. Пример на рисунке 1.3.

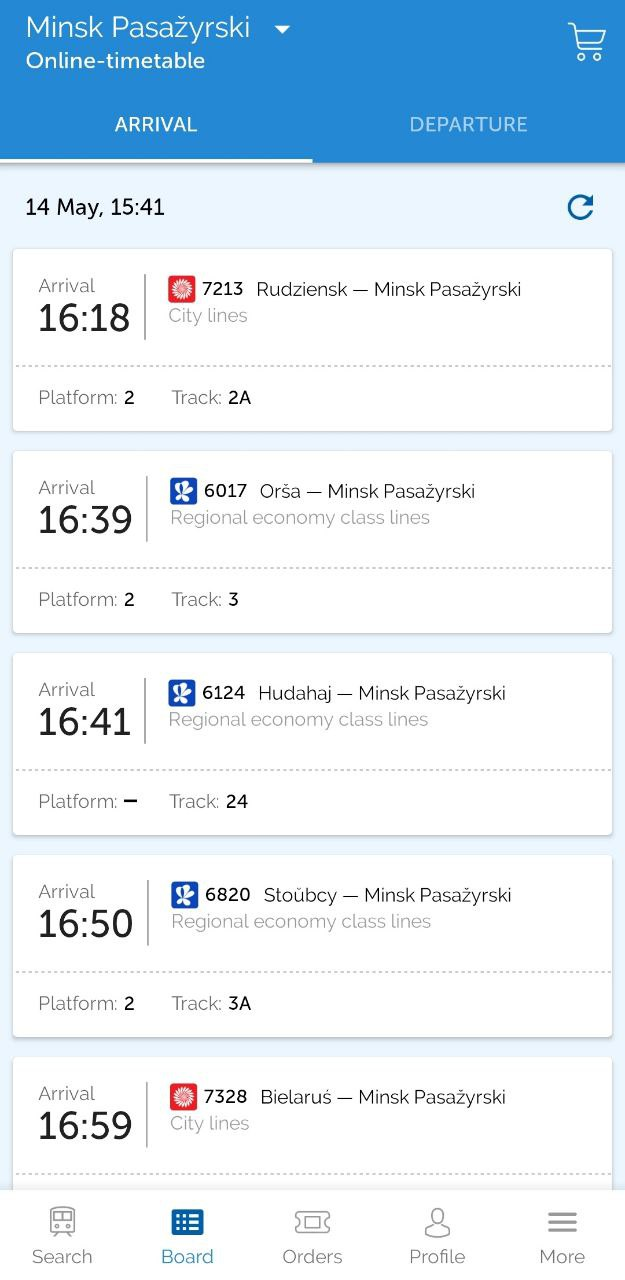


Рисунок 1.3 – Расписание рейсов в приложении «БЧ. Мой поезд»

Однако, как и любое подобное приложение, у него есть минусы. Одним из таких минусов можно назвать ошибки при бронировании билетов. У некоторых пользователей возникают проблемы с бронированием билетов через приложение. Например, система может выдавать ошибки при оплате или не отображать доступные для бронирования билеты.

1.2.2 Аналог «Intercity.pl»

Еще одним аналогом является сайт «Intercity.pl» [2], предоставляющий услуги по бронированию билетов на территории Польши. Сайт продемонстрирован на рисунке 1.4.

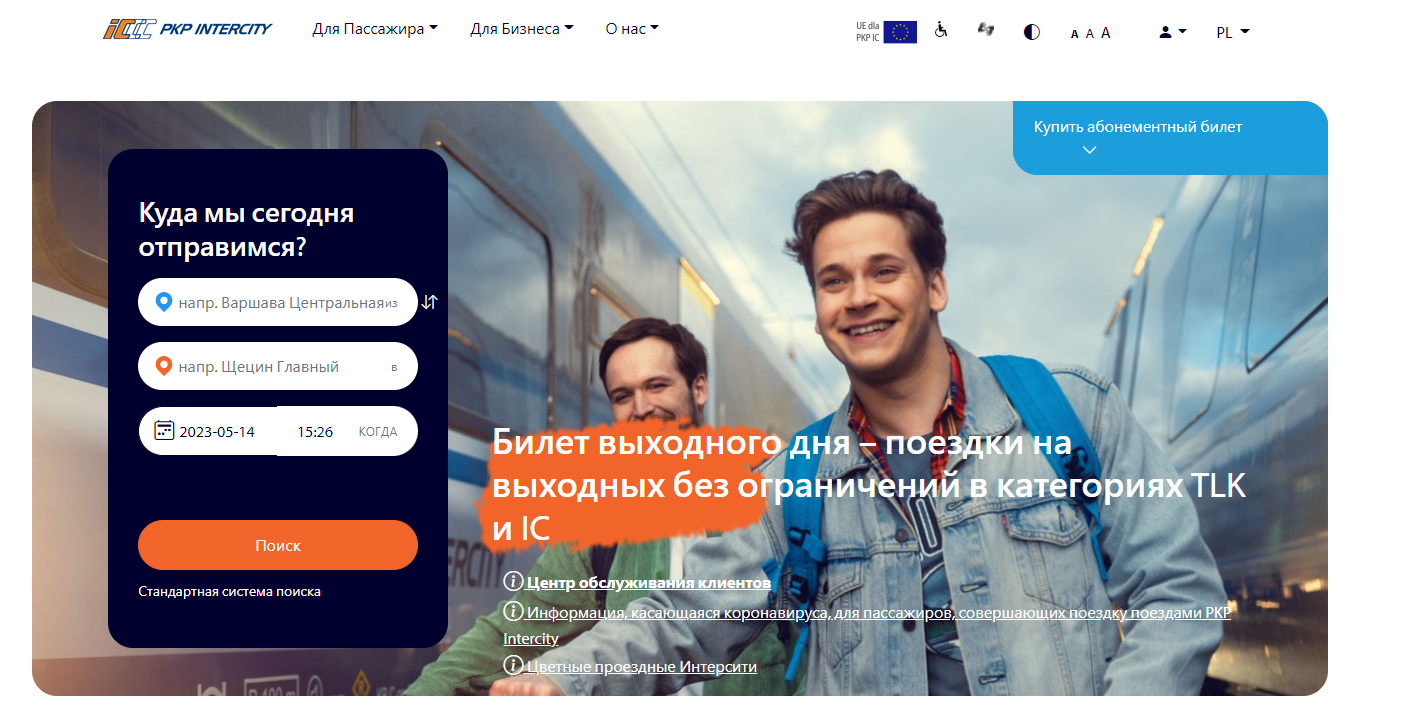


Рисунок 1.4 – Сайт «Intercity.pl»

Из минусов, можно отметить, наличие только возможности поиска билетов и бронирования билетов.

1.3 Изучение требований, определение вариантов использования

Важным этапом разработки проекта является изучение требований и определение возможных вариантов использования. Требования к проекту могут быть как функциональными, то есть связанными с основными функциями проекта, так и нефункциональными, то есть связанными с качественными характеристиками проекта.

1.3.1 Определение основных функциональных требований к базе данных

В ходе изучения требований было выявлено, что база данных железнодорожного вокзала должна включать информацию о поездах, вагонах, расписании движения, билетах и о пассажирах. Были определены следующие варианты использования:

1. Хранение информации о расписании прибытия и отправления поездов на определенную дату и время.
2. Предоставление информации о расписании движения поездов пассажирам.
3. Возможность просмотра и редактирования информации о поездах, включая название, маршрут, время отправления и прибытия, количество свободных мест.
4. Хранение информации о билетах на поезда, включая стоимость, маршрут, дату и время отправления, место в вагоне.
5. Разработка простого приложения для взаимодействия с базой данных.
6. Ведение учета проданных билетов и свободных мест на каждый поезд.
7. Хранение информации о пассажирах, включая имя, фамилию, номер паспорта и предоставляемые льготы.
8. Обеспечение защиты данных и доступа только авторизованных пользователей к базе данных.
9. Мониторинг состояния базы данных и обнаружение проблем в работе СУБД, включая информацию о возникающих ошибках и неполадках, о времени отклика системы, а также о загрузке процессора и объеме свободного места на диске.

1.3.2 Определение нефункциональных требований к базе данных

В качестве нефункциональных требований, можно выделить:

1. Безопасность: База данных должна обеспечивать безопасность данных и защиту от несанкционированного доступа, в том числе использование механизмов авторизации и аутентификации.
2. Производительность: База данных должна иметь высокую производительность и обеспечивать быстрый доступ к данным для обеспечения эффективной работы вокзала.
3. Надежность: База данных должна быть надежной и стабильной, обеспечивать целостность данных и иметь возможность быстрого восстановления после сбоев.
4. Масштабируемость: База данных должна быть масштабируемой и гибкой, чтобы обеспечивать эффективное управление растущим объемом данных вокзала.
5. Удобство использования: База данных должна быть удобной и простой в использовании для обеспечения эффективной работы пользователей.

1.3.3 Определение вариантов использования

Вариантами использования базы данных являются:

1. Авторизация пользователей или менеджеров.
2. Просмотр расписания поездов.
3. Просмотр информации о наличии мест на поездах.
4. Бронирование места на поезд.
5. Создание расписания со стороны менеджера.
6. Просмотр оформленных билетов.
7. Администрирование со стороны менеджера.

1.4 Вывод

В данном разделе проведен аналитический обзор литературы по теме проекта, а также осуществлен сравнительный анализ теоретических и эвристических методов решения поставленных задач и существующих технических научных решений.

Также был проведен аналитический обзор аналогов, включающий два конкретных примера аналогов: «Белорусская железная дорога» и «Intercity.pl». Это позволило определить преимущества и недостатки существующих решений и использовать эту информацию при разработке нового проекта.

Важным этапом является изучение требований, определение основных функциональных и нефункциональных требований к базе данных и вариантов использования. Это позволило определить ключевые аспекты, которые необходимо учитывать при проектировании и разработке проекта.

2. Разработка архитектуры проекта

2.1 Обобщенная структура управления приложением

Пользовательский интерфейс взаимодействует с бизнес-логикой, которая в свою очередь взаимодействует со слоем доступа к данным. Слой доступа к данным выполняет запросы к базе данных Oracle и возвращает результаты обратно в бизнес-логику, которая обрабатывает эти результаты и возвращает их обратно в пользовательский интерфейс. База данных содержит данные, с которыми приложение работает.

Обобщенная структура управления приложением представлена на рисунке 2.1.

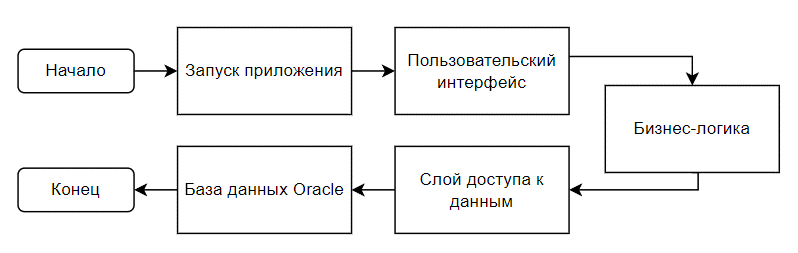


Рисунок 2.1 – Обобщенная структура управления приложением

Первым делом приложение должно установить соединение с БД, чтобы получить доступ к данным. Для этого в приложении на C# WPF используется объект класса OracleConnection, который содержит информацию о сервере БД, имя БД и учетные данные пользователя.

После установления соединения с БД, приложение может выполнять запросы к БД. Для выполнения запросов используется объект класса OracleCommand. В основном в приложении используются динамические запросы.

Следущим этапом приложение получает результаты в виде объектов класса OracleDataAdapter или OracleDataReader. OracleDataReader используется для получения одной строки результата за раз, а OracleDataAdapter - для получения целого набора данных. Для обеспечения безопасного доступа к базе данных были использованы хранимые процедуры и способ привязки параметров к OracleCommand. Полученные результаты используются в самом приложении для отображения или взаимодействия с ними.

После выполнения запросов и обработки результатов, приложение должно закрыть соединение с БД, чтобы освободить ресурсы сервера БД.

В целом, для работы с Oracle Database в приложении на C# WPF используется набор классов из пространства имен Oracle.ManagedDataAccess.Client. Они предоставляют различные функции для работы с БД, такие как создание соединения, выполнение запросов и обработка результатов. Однако, для работы с Oracle Database, также необходимо установить дополнительные компоненты, такие как Oracle Data Provider for .NET (ODP.NET).

2.2 Диаграммы UML, взаимосвязь всех компонентов

Диаграммы UML могут помочь визуализировать взаимосвязь всех компонентов в системе базы, а также описать функциональность системы и ее взаимодействие с внешними пользователями или другими системами.

Диаграмма вариантов использования была создана для описания функциональности системы базы данных железнодорожного вокзала и ее взаимодействия с внешними пользователями. На диаграмме показаны актеры, варианты использования и связи между ними.

Актерами на диаграмме являются пользователи и менеджеры. Варианты использования включают бронирование билетов, просмотр расписания поездов, изменение информации о поездах и т.д.

Сама диаграмма представлена на рисунке 2.1.

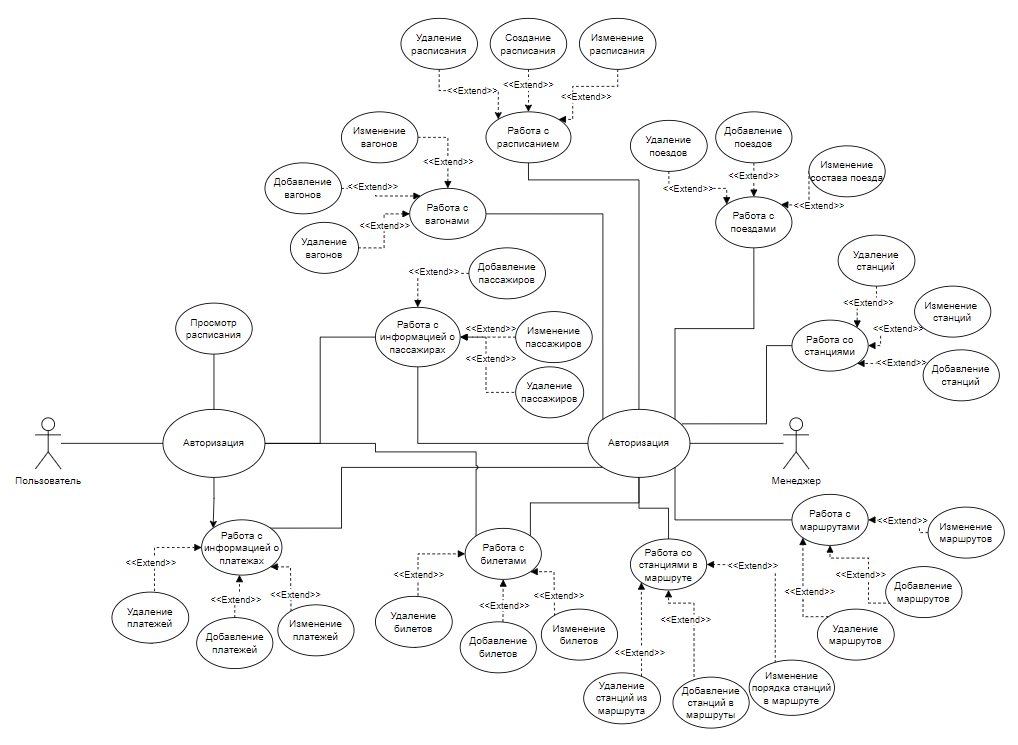


Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма вариантов использования помогает лучше понять функциональность системы базы данных железнодорожного вокзала и ее взаимодействие с внешними пользователями, что может быть полезным при проектировании и разработке системы.

2.3 Описание информационых объектов и ограничений целостности

Для разработки веб-приложения в рамках курсового проекта понадобилась база данных с необходимой конфигурацией сущностей для хранения всей необходимой информации.

Структуру базу данных, ограничения целостности, связи и поля можно увидеть в приложении A и на рисунке 2.2.

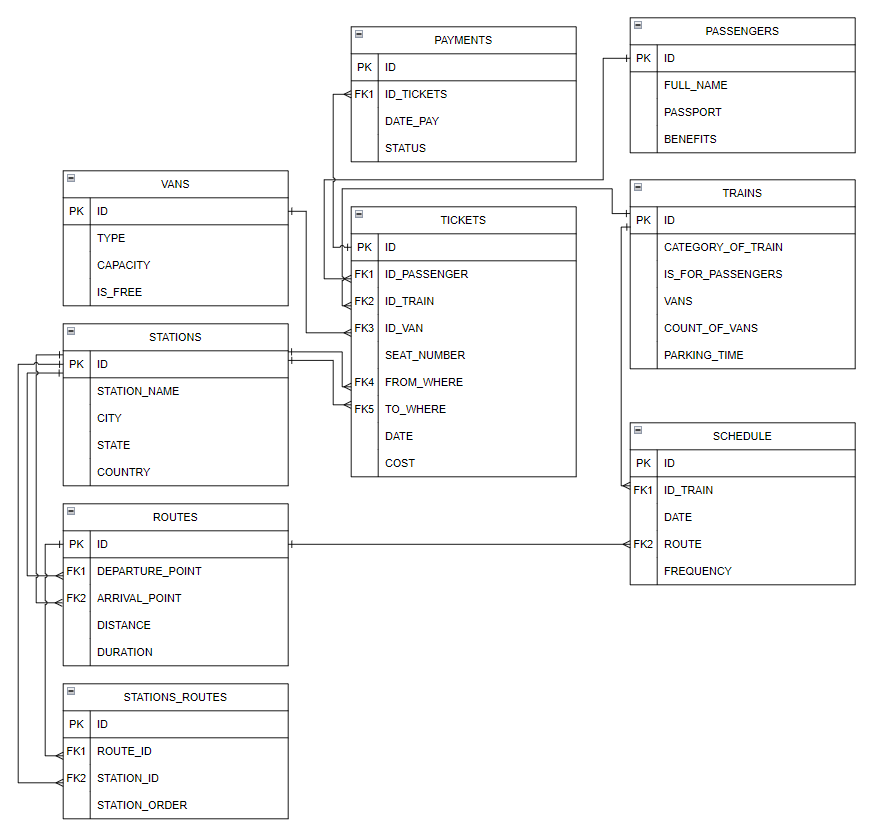


Рисунок 2.2 – Структура базы данных

В базе данных существует 9 таблиц, каждая из которых выполняет свою функцию.

2.3.1 Таблица Stations

Данная таблица используется для хранения информации о станциях остановки составов.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой точки остановки.
* Cтолбец STATION\_NAME. Хранит название станции.
* Cтолбец CITY. Хранит название города, в котором расположена станция.
* Cтолбец STATE. Хранит название региона, в котором расположена станция.
* Cтолбец COUNTRY. Хранит название страны, в котором расположена станция.

Данная таблица содержит ограничение перичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение.

2.3.2 Таблица Routes

Данная таблица используется для хранения информации о маршрутах передвижения составов.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого маршрута.
* Cтолбец DEPARTURE\_POINT. Хранит ID начальной точки.
* Cтолбец ARRIVAL\_POINT. Хранит ID конечной точки.
* Cтолбец DISTANCE. Хранит расстояние, проходящее составом.
* Cтолбец DURATION. Хранит время, проводимое в пути.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также для столбцов DEPARTURE\_POINT и ARRIVAL\_POINT существуют ограничения целостности по внешнему ключу к таблице STATIONS.

2.3.3 Таблица Schedule

Данная таблица используется для хранения информации о расписании передвижения составов.

В её состав входят следущие столбцы:

* Cтолбец ID. Хранит уникальный идентификатор каждого маршрута.
* Cтолбец ID\_TRAIN. Хранит ID поезда, зарезервированного для маршрута.
* Cтолбец DATE. Хранит дату начала выдвижения состава по маршруту.
* Cтолбец ROUTE. Хранит ID маршрута передвижения состава.
* Cтолбец FREQUENCY. Хранит частоту использования данной ячейки расписания.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также для столбцов ID\_TRAIN и ROUTE существуют ограничения целостности по внешнему ключу к таблице TRAINS и ROUTES соответсвенно.

Также таблица включает ограничение check на столбец FREQUENCY.

2.3.4 Таблица Trains

Данная таблица используется для хранения информации о поездах(составах).

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого состава.
* Cтолбец CATEGORY\_OF\_TRAIN. Хранит категорию данного состава.
* Cтолбец IS\_FOR\_PASSENGERS. Хранит число, обозначающее возможность использования данного состава для перевозки пассажиров.
* Cтолбец VANS. Хранит массив ID вагонов, из которых состоит поезд.
* Cтолбец COUNTS\_OF\_VANS. Хранит количество вагонов в составе.
* Cтолбец PARKING\_TIME. Хранит время стоянки состава.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также таблица включает ограничение check на столбец IS\_FOR\_PASSENGERS.

2.3.5 Таблица Vans

Данная таблица используется для хранения информации о вагонах.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого вагона.
* Cтолбец TYPE. Хранит тип вагона.
* Cтолбец CAPACITY. Хранит вместимость вагона.
* Столбец IS\_FREE. Хранит число, указывающее, прикреплен ли вагон к составу.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также таблица включает ограничение check на столбец IS\_FREE.

2.3.6 Таблица Passengers

Данная таблица используется для хранения информации о пассажирах.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого пассажира.
* Cтолбец FULLNAME. Хранит ФИО пассажира.
* Cтолбец PASSPORT. Хранит паспортные данные пассажира.
* Cтолбец BENEFITS. Хранит льготы, применяемые к пассажиру.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также таблица включает ограничение check на столбец BENEFITS.

2.3.7 Таблица Payments

Данная таблица используется для хранения информации о платежах.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого платежа.
* Cтолбец ID\_TICKET. Хранит ID билета, по которому произошла оплата.
* Cтолбец DATE\_PAY. Хранит дату оплаты.
* Cтолбец STATUS. Хранит статус оплаты.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также для столбцов ID\_TICKET существует ограничение целостности по внешнему ключу к таблице TICKETS.

Также таблица включает ограничение check на столбец STATUS.

2.3.8 Таблица Tickets

Данная таблица используется для хранения информации о билетах.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждого билета.
* Cтолбец ID\_PASSENGER. Хранит ID пассажира, которому был выдан билет.
* Cтолбец ID\_TRAIN. Хранит ID поезда, на который забронирован билет.
* Cтолбец ID\_VAN. Хранит ID вагона, в котором находится арендованное место.
* Cтолбец SEAT\_NUMBER. Хранит номер арендованного места.
* Cтолбец FROM\_WHERE. Хранит место посадки на поезд.
* Cтолбец TO\_WHERE. Хранит место выхода из поезда.
* Cтолбец DATE. Хранит дату поездки.
* Cтолбец COST. Хранит стоимость билета.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также для столбцов ID\_PASSENGER, ID\_TRAIN и ID\_VAN существуют ограничения целостности по внешнему ключу к таблице PASSENGERS, TRAINS и VANS соответсвенно.

2.3.9 Таблица Stations\_routes

Данная таблица используется для хранения информации о порядке следования станций в маршрутах.

В её состав входят следущие столбцы:

* Столбец ID. Хранит уникальный идентификатор каждой станции в маршруте.
* Cтолбец ROUTE\_ID. Хранит ID маршрута, в котором привязана станция.
* Cтолбец STATION\_ID. Хранит ID станции, которая используется в маршруте.
* Cтолбец STATION\_ORDER. Хранит номер станции в маршруте.

Данная таблица содержит ограничение перичного ключа для столбца ID.

Также для столбцов ROUTE\_ID, STATION\_ID существуют ограничения целостности по внешнему ключу к таблице ROUTES, STATIONS соответственно.

2.4 Вывод

В данном разделе была разработана архитектура проекта, описано взаимодействие приложения с базой данных, составлена диаграмма вариантов использования. Также была описана структура базы данных вместе с ограничениями целостности.

3. Разработка модели базы данных

Модель базы данных – это структура, которая определяет, как данные будут храниться, организовываться и управляться в базе данных. Она описывает типы данных, связи между таблицами, ограничения целостности данных и другие аспекты, необходимые для правильного функционирования базы данных.

Правильная модель базы данных обеспечивает эффективное хранение и управление данными, а также обеспечивает защиту данных и целостность базы данных.

3.1 Создание необходимых объектов

Для реализации базы данных железнодорожного вокзала необходимо создать набор объектов базы данных, которые представляют собой таблицы, представления, индексы, ограничения и хранимые процедуры. Все эти объекты должны быть созданы в соответствии с требованиями к производительности, безопасности и надежности.

В проектируемой базе данных были созданы 9 таблиц, описание которых приведено выше.

Для взаимодействия с этими таблицами были разработаны такие объекты базы данных, как функции, процедуры, представления и триггеры, а также пользовательский тип данных и индексы.

3.1.1 Функции

Функция в Oracle – это объект базы данных, который возвращает значение на основе переданных в нее аргументов. Функции могут быть созданы с помощью оператора CREATE FUNCTION. Пример функции, которая удаляет из массива VARRAY определенное значение – листинг 3.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION *delete\_entry*(p\_record IN VANS\_COMPOSITION, p\_val IN NUMBER) RETURN VANS\_COMPOSITION IS  v\_ret VANS\_COMPOSITION := VANS\_COMPOSITION(); BEGIN  FOR n IN p\_record.FIRST..p\_record.LAST LOOP  IF p\_record(n) != p\_val THEN  v\_ret.EXTEND;  v\_ret(v\_ret.LAST) := p\_record(n);  END IF;  END LOOP;  RETURN v\_ret; END; |

Листинг 3.1 – Функция работы с массивом

Здесь мы создаем функцию delete\_entry, которая принимает массив чисел и одно число и возвращает новый массив без этого числа.

3.1.2 Процедуры

Процедура в Oracle – это объект базы данных, который представляет собой набор SQL-инструкций, которые могут быть вызваны для выполнения определенной задачи. Процедуры могут быть созданы с помощью оператора CREATE PROCEDURE. Пример определения процедуры, которая обновляет вместимость вагонов, по прошествии даты поездки в билете – листинг 3.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *UPDATE\_VANS\_CAPACITY* AS BEGIN DECLARE  CURSOR cur IS SELECT ID\_VAN FROM TICKETS WHERE TICKETS."DATE" < *SYSDATE*;  TYPE id\_tab\_type IS TABLE OF TICKETS.ID\_VAN%TYPE INDEX BY PLS\_INTEGER;  id\_tab id\_tab\_type; BEGIN  OPEN cur;  LOOP  FETCH cur BULK COLLECT INTO id\_tab LIMIT 1000;  EXIT WHEN id\_tab.COUNT = 0;  FORALL i IN 1..id\_tab.COUNT  UPDATE VANS SET CAPACITY = CAPACITY + 1  WHERE ID = id\_tab(i);  END LOOP;  CLOSE cur; END; end; |

Листинг 3.2 – Процедура UPDATE\_VANS\_CAPACITY

Данный код представляет собой определение процедуры UPDATE\_VANS\_CAPACITY, которая использует курсор для выборки идентификаторов вагонов из таблицы TICKETS и обновляет информацию о вместимости вагонов в таблице VANS.

Процедура позволяет эффективно обновлять информацию о вместимости вагонов на железнодорожных станциях, используя массовое обновление с помощью курсора и массива.

В сумме, при разработке курсового проекта, было создано 35 процедур для следующих целей:

1. Добавление данных в таблицы.
2. Удаление данных из таблиц.
3. Изменение данных в таблицах.
4. Экспорт и импорт таблицы в формат JSON.
5. Изменение данных при срабатывании триггеров.
6. Работа с представлениями.

Весь перечень созданных процедур представлен в Приложении Б.

3.1.3 Представления

Представление в Oracle – это объект базы данных, который представляет собой виртуальную таблицу, основанную на результатах выполнения SQL-запроса. Представления могут быть созданы с помощью оператора CREATE VIEW. Пример определения представления, которое используется для отображения расписания – листинг 3.3.

|  |
| --- |
| CREATE OR REPLACE VIEW TAKE\_SCHEDULE AS  SELECT SCHEDULE."ID", ID\_TRAIN, TRAINS.CATEGORY\_OF\_TRAIN,  ROUTES.DEPARTURE\_POINT, ROUTES.ARRIVAL\_POINT,  ROUTES.DISTANCE, ROUTES.DURATION,  "DATE", FREQUENCY, TRAINS.IS\_FOR\_PASSENGERS  FROM SCHEDULE  JOIN TRAINS ON SCHEDULE.ID\_TRAIN = TRAINS.ID  JOIN ROUTES ON SCHEDULE.ROUTE = ROUTES.ID; |

Листинг 3.3 – Представление TAKE\_SCHEDULE

Данный код создает представление, которое объединяет таблицы "SCHEDULE", "TRAINS" и "ROUTES", чтобы получить информацию о графике движения поездов. В результате выполнения данного кода можно получить доступ к перечню поездов, их категории, маршруту, дате, частоте и другой информации, связанной с графиком движения.

Также было создано еще одно представление для удобного отображения билетов, а также быстрого взаимодействия с ними, листинг 3.4.

|  |
| --- |
| CREATE OR REPLACE VIEW TAKE\_TICKET AS  SELECT TICKETS.ID, ID\_PASSENGER, ID\_TRAIN, ID\_VAN, SEAT\_NUMBER, FROM\_WHERE, TO\_WHERE, "DATE", COST, DATE\_PAY, STATUS FROM TICKETS  JOIN PAYMENTS ON TICKETS.ID = PAYMENTS.ID\_TICKET; |

Листинг 3.4 – Представление TAKE\_TICKET

3.1.4 Триггеры

Триггер в Oracle – это объект базы данных, который автоматически запускается при определенном событии в базе данных, таком как вставка, обновление или удаление данных. Триггеры могут быть созданы с помощью оператора CREATE TRIGGER. Пример определения триггера, который проверяет вагоны в поезде – листинг 3.5.

|  |
| --- |
| CREATE OR REPLACE TRIGGER TRAINS\_INSERT\_UPDATE\_TRIGGER  BEFORE INSERT OR UPDATE ON TRAINS  FOR EACH ROW  DECLARE  vans\_in VANS\_COMPOSITION := VANS\_COMPOSITION();  vans\_out VANS\_COMPOSITION := VANS\_COMPOSITION();  vansfree NUMBER(1, 0);  BEGIN  IF INSERTING THEN  vans\_in := :NEW.VANS;  vans\_out := :NEW.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  IF vansfree = 0 THEN  vans\_out := DELETE\_ENTRY(vans\_out, vans\_in(I));  ELSE  UPDATE VANS SET IS\_FREE = 0 WHERE ID = vans\_in(I);  END IF;  END LOOP;  :NEW.COUNT\_OF\_VANS := vans\_out.COUNT;  :NEW.VANS := vans\_out;  ELSIF UPDATING('VANS') THEN  vans\_in := :OLD.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  UPDATE VANS SET IS\_FREE = 1 WHERE ID = vans\_in(I);  END LOOP;  vans\_in := :NEW.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  IF vansfree = 0 THEN  vans\_out := DELETE\_ENTRY(vans\_out, vans\_in(I));  ELSE  UPDATE VANS SET IS\_FREE = 0 WHERE ID = vans\_in(I);  END IF;  END LOOP;  :NEW.VANS := vans\_out;  :NEW.COUNT\_OF\_VANS := vans\_out.COUNT;  END IF;  END; |

Листинг 3.5 – Триггер TRAINS\_INSERT\_UPDATE\_TRIGGER

Данный код представляет собой триггер, который запускается перед операциями INSERT или UPDATE на таблице "TRAINS". Он обходит массив VARAAY VANS\_COMPOSITION для поиска плохих данные (повторяющихся и недоступных значений), при нахождении выполняется удаление данных из массива.

В результате выполнения кода, массив значений будет содержать только уникальные и разрешенные значения.

Всего было разработано 6 триггеров для решения следующих задач:

1. Обновление вместимости вагонов.
2. Проверка вводимых данных.

3.1.5 Пользовательские типы данных

Для хранения номеров вагонов, прикрепленных к определенному составу, был разработан пользовательский тип данных на основе массива varray, листинг 3.6.

|  |
| --- |
| CREATE OR REPLACE TYPE VANS\_COMPOSITION AS VARRAY(15) OF NUMBER(14,0); |

Листинг 3.6 – Пользовательский тип VANS\_COMPOSITION

3.1.6 Индексы

Для оптимизации времени, затрачиваемого на выполнение запросов к таблицам, были разработаны индексы.

Индексы могут быть созданы с помощь оператора CREATE INDEX. Cозданные индекса представлены на листинге ниже.

|  |
| --- |
| CREATE INDEX id\_train\_date\_idx ON schedule (id\_train, 'date');  CREATE INDEX type\_is\_free\_idx ON vans (type, is\_free);  CREATE INDEX DATE\_SORT\_INDEX ON TICKETS("DATE" DESC); |

Листинг 3.7 – Созданные в проекте индексы

Исключая индексы, автоматически создаваемые к первичным ключам таблиц, было создано 3 индекса, для самых частоиспользуемых запросов.

3.2 Заполнение таблицы 100000 строк

Для заполнения таблицы 100000 строк был написан скрипт на языке программирования JavaScript. Листинг скрипта приведен ниже.

|  |
| --- |
| const oracledb = require('oracledb');  async function ToVANS() {  let connection;  try {  connection = await oracledb.getConnection({  user: 'MANAGER',  password: 'MANAGER\_PASS',  connectString: 'localhost:1521/xe'  });  // Определяем название таблицы  const tableName = 'VANS';  const col = [  "TYPE",  "CAPACITY",  "IS\_FREE"  ]  const values = [  ":value1",  ":value2",  ":value3"  ]  // Генерируем случайные данные  /\*  Грузовые вагоны  Пассажирские вагоны  Вагоны-цистерны  Платформы для перевозки контейнеров  Лесовозы  Термосы для перевозки продуктов  Вагоны для перевозки автомобилей  \*/  const data = [];  function SwitchRand(rand) {  switch(rand) {  case 1: return "Грузовой";  break;  case 2: return "Пассажирский";  break;  case 3: return "Цистерна";  break;  case 4: return "Контейнеры";  break;  case 5: return "Лесовоз";  break;  case 6: return "Продукты";  break;  case 7: return "Автомобили";  break;  }  }  for (let i = 0; i < 100000; i++) {  const row = [  SwitchRand(Math.floor(Math.random() \* 6 + 1)),  Number(60),  Number(1)  ];  data.push(row);  }  // Формируем SQL-запрос для вставки данных  const query = `INSERT INTO ${tableName} (${col[0]}, ${col[1]}, ${col[2]}) VALUES (${values[0]}, ${values[1]}, ${values[2]})`;  for (let i = 0; i < data.length; i++) {  // Выполняем запрос с помощью массива значений  const result = await connection.execute(query, [data[i][0], data[i][1], data[i][2]]);  console.log(`Inserted ${result.rowsAffected} rows into the table`);  }  await connection.commit();  } catch (err) {  console.error(err);  } finally {  if (connection) {  try {  await connection.close();  } catch (err) {  console.error(err);  }  }  }  } |

Листинг 3.8 – Код скрипта вставки 100000 строк в таблицу VANS

**3.3 Описание технологии**

Мониторинг состояния СУБД вляется важным инструментом для обеспечения высокой производительности и надежности базы данных. Он позволяет анализировать работу базы данных в реальном времени и выявлять проблемы, которые могут влиять на ее работу.

В данном проекте используется Oracle Enterprise Manager (OEM).

Oracle Enterprise Manager (OEM) - это инструмент управления базами данных Oracle, который позволяет централизованно управлять всеми аспектами базы данных, включая мониторинг, настройку, управление и автоматическое уведомление о проблемах. OEM является мощным инструментом для управления базами данных, который может быть использован как для управления одной базой данных, так и для управления несколькими базами данных.

Основные функции Oracle Enterprise Manager включают в себя:

1. Мониторинг производительности: OEM позволяет мониторить производительность базы данных в реальном времени и предоставляет детальную информацию о производительности, включая информацию о блокировках, процессах, сессиях и других параметрах.
2. Автоматическое уведомление о проблемах: OEM позволяет настроить уведомления о проблемах в базе данных, такие как ошибка в журнале аудита, превышение пороговых значений производительности и другие. Уведомления могут быть отправлены через электронную почту, SMS или другие способы.
3. Управление настройками: OEM позволяет настраивать параметры базы данных, включая параметры памяти, параметры сети и другие.
4. Управление безопасностью: OEM позволяет управлять безопасностью базы данных, включая настройку параметров аутентификации и авторизации, мониторинг защиты данных и другие функции.
5. Управление резервными копиями: OEM позволяет создавать и управлять резервными копиями базы данных, включая настройку резервного копирования и восстановления данных.
6. Управление версиями: OEM позволяет управлять версиями базы данных и обновлять ее до последней версии.

Для реализации Oracle Enterprise Manager необходимо установить и настроить специальный сервер, который будет управлять базами данных. После установки сервера необходимо настроить подключение к базе данных и настроить параметры мониторинга и уведомлений.

Одной из главных функций OEM является мониторинг производительности базы данных. Он позволяет анализировать данные о нагрузке на базу данных, использовании ресурсов и производительности запросов – рисунок 3.9.

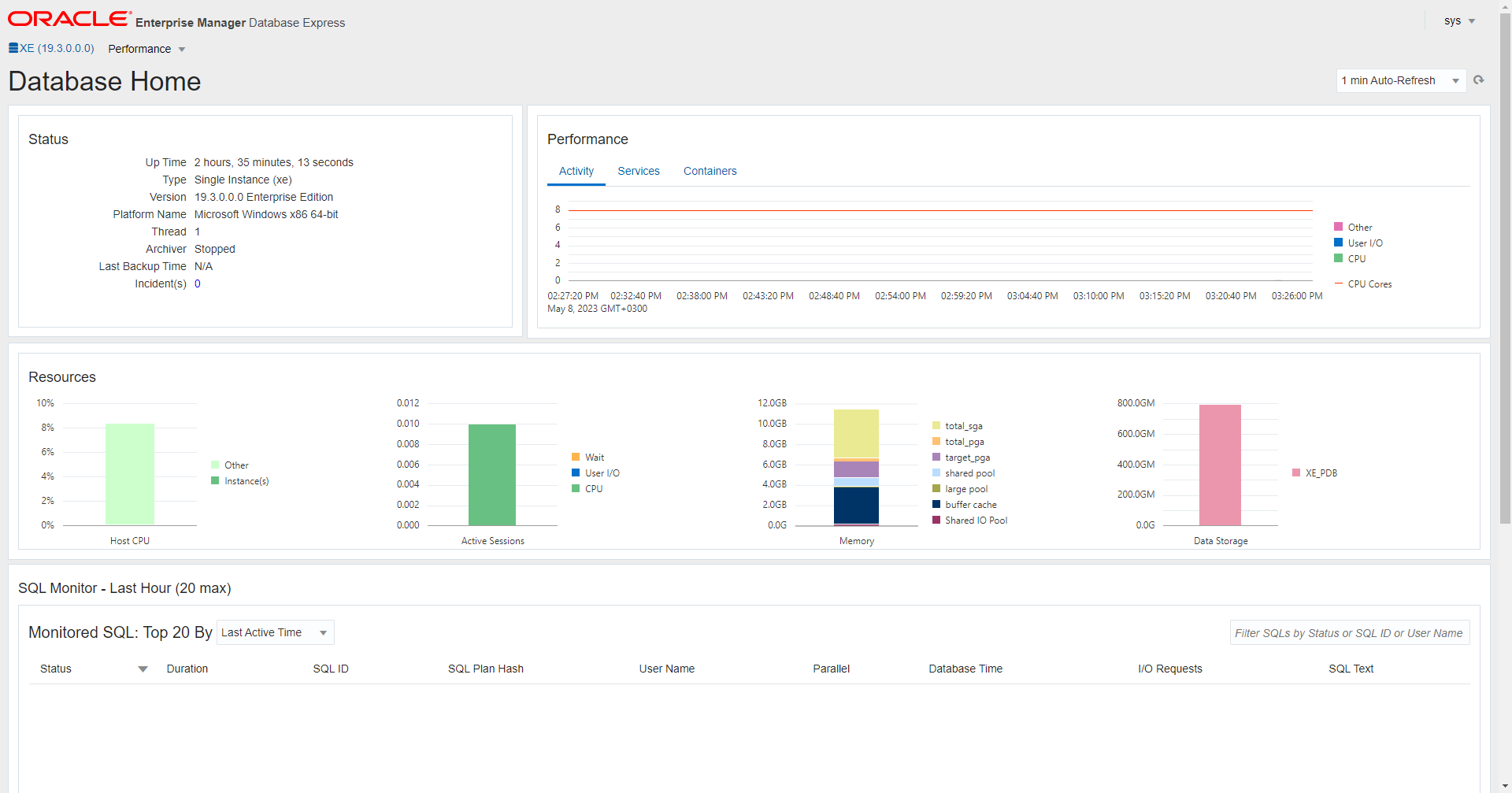


Рисунок 3.9 – Oracle Enterprise Manager

Данное ПО доступно из любого браузера по ссылке, настраиваемой администратором, что значительно упрощает доступ для мониторинга состояния базы данных.

Кроме того, OEM обеспечивает механизмы автоматического обнаружения и устранения проблем в работе базы данных. Он позволяет анализировать журналы ошибок, настраивать систему резервного копирования, мониторить доступность базы данных и многое другое.

Таким образом, использование Oracle Enterprise Manager в проекте обеспечивает высокую производительность и надежность базы данных, а также облегчает ее администрирование и управление.

**3.4 Вывод**

В данном разделе была разработана модель данных, была составлена UML диаграмма базы данных, а также описана каждая таблица в диаграмме. Были описаны виды разработанных объектов базы данных. Также были описаны процедуры импорта и экспорта из JSON. Была рассмотрена используемая технология, которая значительно упрощает работу по администрированию базы данных.

4. Установка, настройка и использование Oracle 19c

Oracle Database – это мощная реляционная СУБД, разработанная компанией Oracle Corporation. Она обеспечивает надежное хранение и управление структурированными данными, а также обладает высокой производительностью и масштабируемостью.

Oracle Database имеет широкий набор функций и возможностей, таких как многопоточная обработка, распределенная обработка данных, механизмы безопасности и аутентификации, индексы, триггеры, процедуры, функции и многое другое. Она используется для управления транзакционными данными, аналитическими данными, приложениями и бизнес-процессами, и является одним из основных компонентов многих крупных предприятий и организаций во всем мире.

Oracle Database поддерживает различные операционные системы и аппаратные платформы, что делает ее универсальным решением для любых задач, связанных с управлением данными. Она также обладает мощными инструментами для администрирования и мониторинга баз данных, что делает ее идеальным выбором для профессионалов в области баз данных.

4.1 Создание ролей для разграничения доступа

Роли в Oracle позволяют управлять безопасностью базы данных, ограничивая доступ к конфиденциальным данным и защищая данные от несанкционированного доступа.

Кроме того, использование ролей позволяет упростить управление пользователями и привилегиями, так как все права и разрешения, связанные с определенной задачей, могут быть объединены в одну роль. Это уменьшает количество необходимых для управления пользователями и упрощает процесс назначения привилегий и разрешений.

Для выполнения этих целей были разработаны две роли: менеджера и пользователей. Первым будет рассмотрена роль менеджера.

В обязаности менеджера входит полное управление базой данных, следовательно у него будут все привелегии для созданных объектов. Листинг создания роли менеджера представлен ниже.

|  |
| --- |
| GRANT CREATE SESSION,  CREATE TABLE,  CREATE VIEW,  CREATE PROCEDURE,  CREATE TRIGGER TO RLMANAGERCORE;  CREATE USER MANAGER IDENTIFIED BY MANAGER\_PASS TEMPORARY TABLESPACE TS\_COURSEWORK\_TEMP DEFAULT TABLESPACE TS\_COURSEWORK QUOTA UNLIMITED ON TS\_COURSEWORK ACCOUNT UNLOCK;  GRANT RLMANAGERCORE TO MANAGER;  CREATE PROFILE PMANAGERCORE LIMIT FAILED\_LOGIN\_ATTEMPTS 3 PASSWORD\_LIFE\_TIME UNLIMITED PASSWORD\_GRACE\_TIME UNLIMITED PASSWORD\_LOCK\_TIME UNLIMITED; ALTER USER MANAGER PROFILE PMANAGERCORE; |

Листинг 4.1 – Создание роли менеджера

Для роли пользователя же были выданы только необходимые привилегии, листинг 4.2.

|  |
| --- |
| GRANT CREATE SESSION TO RLUSERCORE;  GRANT SELECT ON TAKE\_SCHEDULE TO RLUSERCORE;  GRANT SELECT, INSERT, UPDATE ON TAKE\_TICKET TO RLUSERCORE;  GRANT EXECUTE ON INSERT\_TAKE\_TICKET TO RLUSERCORE; GRANT EXECUTE ON UPDATE\_TAKE\_TICKET TO RLUSERCORE; GRANT EXECUTE ON DELETE\_TAKE\_TICKET TO RLUSERCORE; GRANT EXECUTE ON INSERT\_PAYMENTS TO RLUSERCORE;  GRANT EXECUTE ON UPDATE\_PAYMENTS TO RLUSERCORE; GRANT EXECUTE ON INSERT\_PASSENGERS TO RLUSERCORE; GRANT EXECUTE ON UPDATE\_PASSENGERS TO RLUSERCORE; GRANT EXECUTE ON DELETE\_PASSENGERS TO RLUSERCORE;  CREATE USER USERS IDENTIFIED BY USERS\_PASS TEMPORARY TABLESPACE TS\_COURSEWORK\_TEMP DEFAULT TABLESPACE TS\_COURSEWORK QUOTA UNLIMITED ON TS\_COURSEWORK ACCOUNT UNLOCK; GRANT RLUSERCORE TO USERS; CREATE PROFILE PUSERCORE LIMIT FAILED\_LOGIN\_ATTEMPTS 5 PASSWORD\_LIFE\_TIME 90 PASSWORD\_GRACE\_TIME 7; ALTER USER USERS PROFILE PUSERCORE; |

Листинг 4.2 – создание роли пользователя

4.2 Описание процедур импорта и экспорта

Процедуры импорта и экспорта являются важными инструментами для перемещения данных между различными базами данных или для создания резервных копий базы данных.

JSON (JavaScript Object Notation) – это легкий формат обмена данными, который используется для представления структурированных данных в виде пар "имя/значение". Он широко используется в веб-приложениях и мобильных приложениях, а также в базах данных.

Использование формата JSON для импорта и экспорта данных предоставляет удобный и гибкий способ обмена данными между различными базами данных или веб-приложениями. Он позволяет представлять структурированные данные в виде пар "имя/значение" и легко преобразовывать данные в различные форматы, такие как XML или CSV.

Процедуры импорта и экспорта данных могут быть полезны для создания резервных копий базы данных, для переноса данных между различными окружениями или для обмена данными с другими приложениями. Они также обеспечивают удобный способ загрузки данных из внешних источников и экспорта данных для анализа и обработки в других приложениях – листинг 4.7.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE TO\_JSON(return\_out out int)  IS BEGIN  DECLARE  output\_file UTL\_FILE.FILE\_TYPE;  json\_data CLOB;  CURSOR vans\_cursor IS  SELECT JSON\_OBJECT(  'type' VALUE TYPE,  'capacity' VALUE CAPACITY,  'is\_free' VALUE IS\_FREE  ) AS json\_data  FROM VANS  WHERE ROWNUM < 10;  BEGIN  output\_file := UTL\_FILE.FOPEN('UTL\_DIR', 'VANS.JSON', 'W');  FOR van\_rec IN vans\_cursor LOOP  json\_data := van\_rec.json\_data;  UTL\_FILE.PUT\_LINE(output\_file, json\_data);  END LOOP;  UTL\_FILE.FCLOSE(output\_file);  return\_out := 1;  END;  end; |

Листинг 4.7 – Пример процедуры экспорта из базы данных

Процедура производит экспорт первых 9 строк из таблицы VANS в файл VANS.JSON.

В целом, процедуры импорта и экспорта данных являются важными инструментами для обеспечения эффективной работы базы данных и обеспечения ее надежности.

4.3 Тестирование производительности базы данных

Тестирование производительности базы данных является важной частью процесса разработки приложений.

Тестирование производительности базы данных – это процесс измерения и анализа производительности базы данных при выполнении различных операций и запросов. Это позволяет убедиться, что база данных способна обрабатывать запросы пользователей с достаточной скоростью и производительностью, чтобы поддерживать высокую нагрузку.

Целью тестирования производительности является выявление проблем, которые могут замедлять работу базы данных и/или приложений, использующих эту базу данных.

Первым делом протестируем скорость выполнения запроса в таблице со 100000 строк, рисунок 4.1.



Рисунок 4.1 – Скорость выполнения запроса без использования индекса

Для того, чтобы уменьшить это число, создадим комбинированный-индекс, по столбцам type и is\_free.



Рисунок 4.2 – Скорость выполнения запроса с использованием комбинированного индекса

Также посмотрим скорость выполнения запроса для таблицы schedule, рисунок 4.3.



Рисунок 4.3 – скорость выполнения запроса без индекса

Теперь же добавим комбинированный индекс по столбцам id\_train и date. Скорость выполнения показана на рисунке 4.4.



Рисунок 4.4 – скорость выполнения запроса с комбинированным индексом

Аналогично для таблицы tickets, рисунок 4.5.



Рисунок 4.5 – скорость выполнения запроса без индексов

Теперь же добавим индекс сортировки по столбцу даты, рисунок 4.6.



Рисунок 4.6 – скорость выполнения запроса с индексом сортировки

4.4 Вывод

В данном разделе были рассмотрены созданные роли, а также приведены их листинги. Также было проведено тестирование скорости выполнения запросов по некоторым таблицам и столбцам.

5. Тестирование, проверка работоспособности и анализ полученных результатов.

5.1. Тестирование клиентской части.

Для клиентской части были написаны процедуры работы с данными из базы данных. А именно добавления, изменения и удаления данных. Процедура INSERT\_TICKETS, которая используется для вставки значений в таблицу TICKETS, рисунок 5.1.





Рисунок 5.1 – Процедура INSERT\_TICKETS

Далее процедура INSERT\_PASSENGERS, которая используется для вставки значений в таблицу PASSENGERS, рисунок 5.2.





Рисунок 5.2 – Процедура INSERT\_PASSENGERS

Далее процедура INSERT\_PAYMENTS, которая используется для вставки значений в таблицу PAYMENTS, рисунок 5.3.





Рисунок 5.3 – Процедура INSERT\_PAYMENTS

Также после вставки (или обновления) строки в таблицу PAYMENTS, должен выполнятся триггер для изменения количества свободных мест в вагоне, связанным с оплаченным билетом, рисунок 5.4.

|  |
| --- |
| а |
| б |

Рисунок 5.4 – Результат применения триггера: а – до, б – после

После этого можно проверить представление TAKE\_SCHEDULE, вывод данных продемонстрирован на рисунке 5.5.

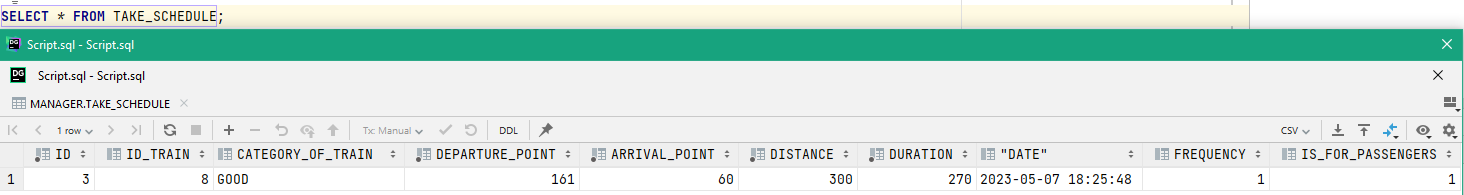


Рисунок 5.5 – представление TAKE\_SCHEDULE

Проверка представления TAKE\_TICKET, рисунок 5.6.



Рисунок 5.6 – Представление TAKE\_TICKET

А также процедуры INSERT\_TAKE\_TICKET для него, рисунок 5.7.





Рисунок 5.7 – Процедура INSERT\_TAKE\_TICKET

5.2. Тестирование части менеджера

Для начала тестирования части менеджера необходимо дозаполнить таблицы данными. Процедура INSERT\_VANS продемонстрирована на рисунке 5.6.





Рисунок 5.6 – Процедура INSERT\_VANS

Далее процедура INSERT\_TRAINS, которая используется для вставки значений в таблицу TRAINS, рисунок 5.7.





Рисунок 5.7 – Процедура INSERT\_TRAINS

Далее процедура INSERT\_ROUTES, которая используется для вставки значений в таблицу ROUTES, рисунок 5.8.





Рисунок 5.8 – Процедура INSERT\_ROUTES

Далее процедура INSERT\_STATIONS, которая используется для вставки значений в таблицу STATIONS, рисунок 5.9.





Рисунок 5.9 – Процедура INSERT\_STATIONS

Далее процедура INSERT\_STATIONS\_ROUTES, которая используется для вставки значений в таблицу STATION\_ROUTES, рисунок 5.10.





Рисунок 5.10 – Процедура INSERT\_STATIONS\_ROUTES

Далее процедура INSERT\_SCHEDULE, которая используется для вставки значений в таблицу SCHEDULE, рисунок 5.11.





Рисунок 5.11 – Процедура INSERT\_SCHEDULE

Аналогичны этим процедурам, процедуры изменения и удаления из базы данных.

Далее протестируем работоспособность триггера TRAINS\_INSERT\_UPDATE\_TRIGGER, рисунок 5.12.



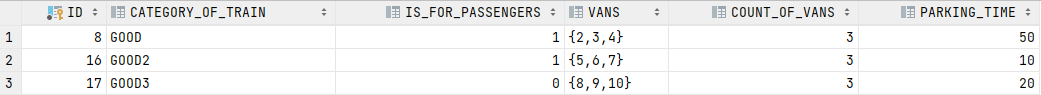
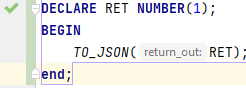


Рисунок 5.12 – Проверка триггера TRAINS\_INSERT\_UPDATE\_TRIGGER

Далее протестируем работоспособность процедур экспорта и импорта данных из JSON, рисунок 5.13, 5.14.



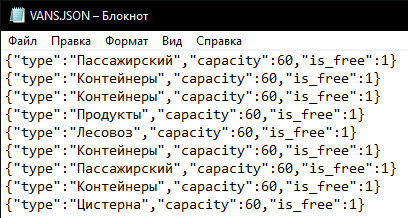


Рисунок 5.13 – Проверка процедуры экспорта

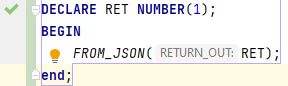




Рисунок 5.14 – Проверка процедуры импорта

5.3. Вывод.

В данном разделе были протестированы возможности клиентов и менеджеров. По итогам тестирования можно сказать, что все работает корректно.

6. Руководство по использованию программного продукта

6.1 Установка приложения

Как таковой установки у приложения нет, нужно просто запустить exe-файл и можно сразу же работать с приложением.

6.2 Работа с приложением

Первое, что видит пользователь приложения – это страница авторизации. Она представлена на рисунке 6.1.

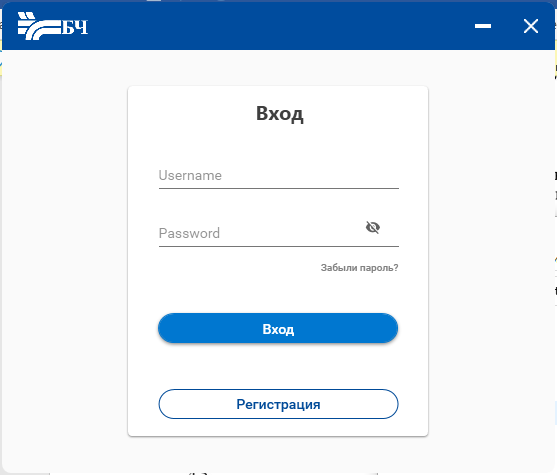


Рисунок 6.1 – Страница авторизации

Для дальнейшей работы с приложением, на этой странице необходимо ввести данные для входа. После чего открывается главная страница с надписью «Приветсвуем в приложении БЧ!».

6.3.1 Сторона менеджера

Слева расположено меню, со страницами для перехода, также, в зависимости от того, кто вошел в систему (пользователь или менеджер), оно отличается, для менеджера добавляется возможность перехода на страницу с панелью администратора, рисунок 6.2.

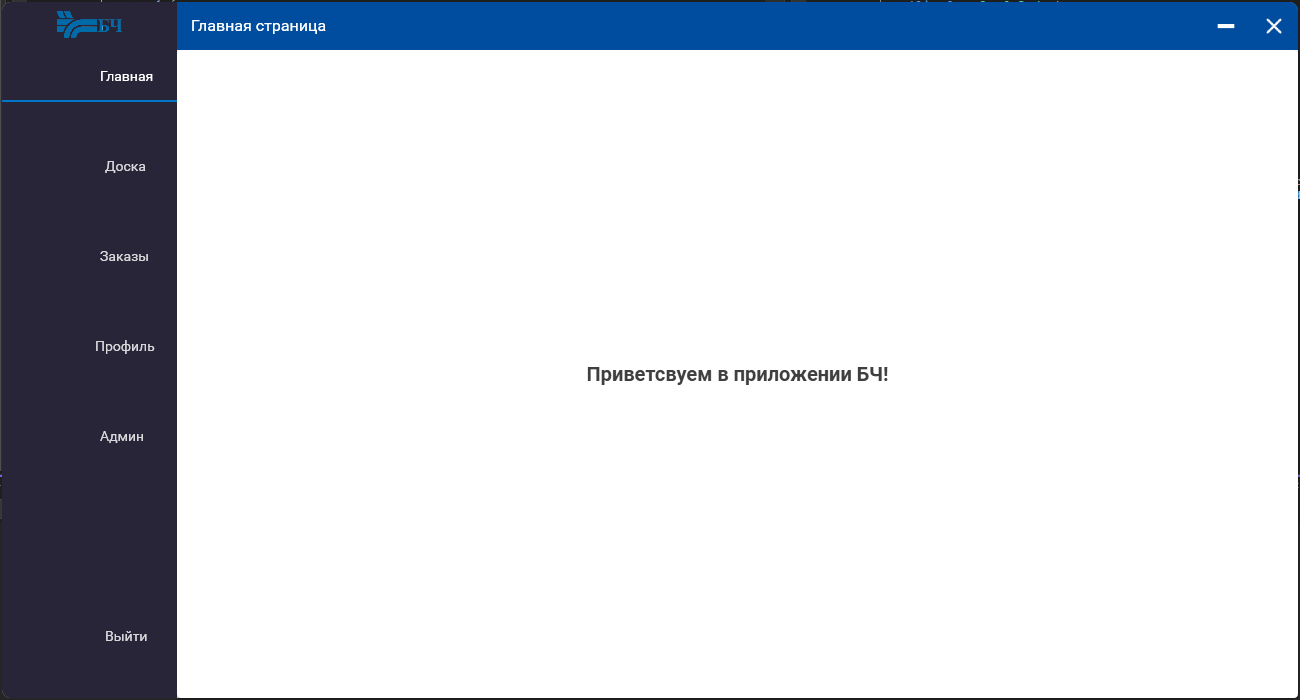


Рисунок 6.2 – Главная страница приложения (менеджер)

Сама панель администратора, представляет собой меню выбора таблицы для непосредственного добавления, редактирования или удаления столбцов у выбранной таблицы и стрелочки для перехода по страницам таблицы, рисунок 6.3.

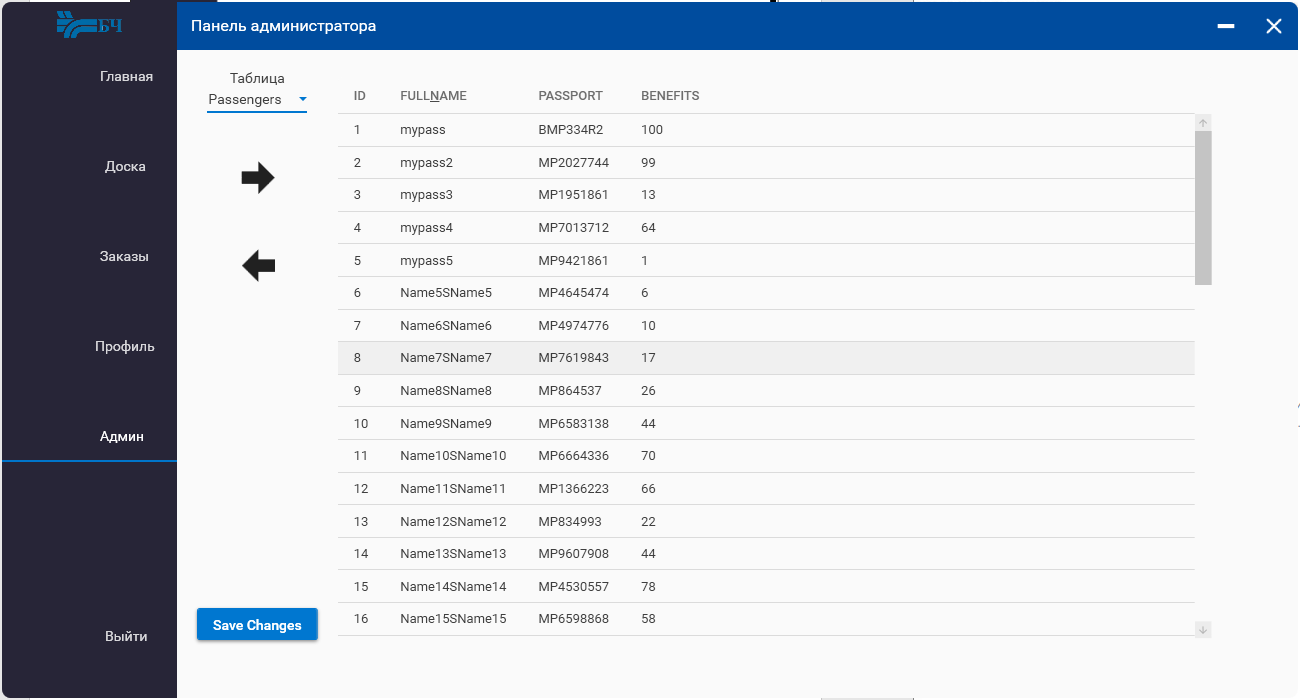


Рисунок 6.3 – Страница «Панель администратора»

На этом отличия менеджера от пользователя, в приложении, заканчиваются. Также администратор может работать с приложением также как и обычный пользователь.

6.3.2 Сторона пользователя

Что видит пользователь при входе в приложение, рисунок 6.4.

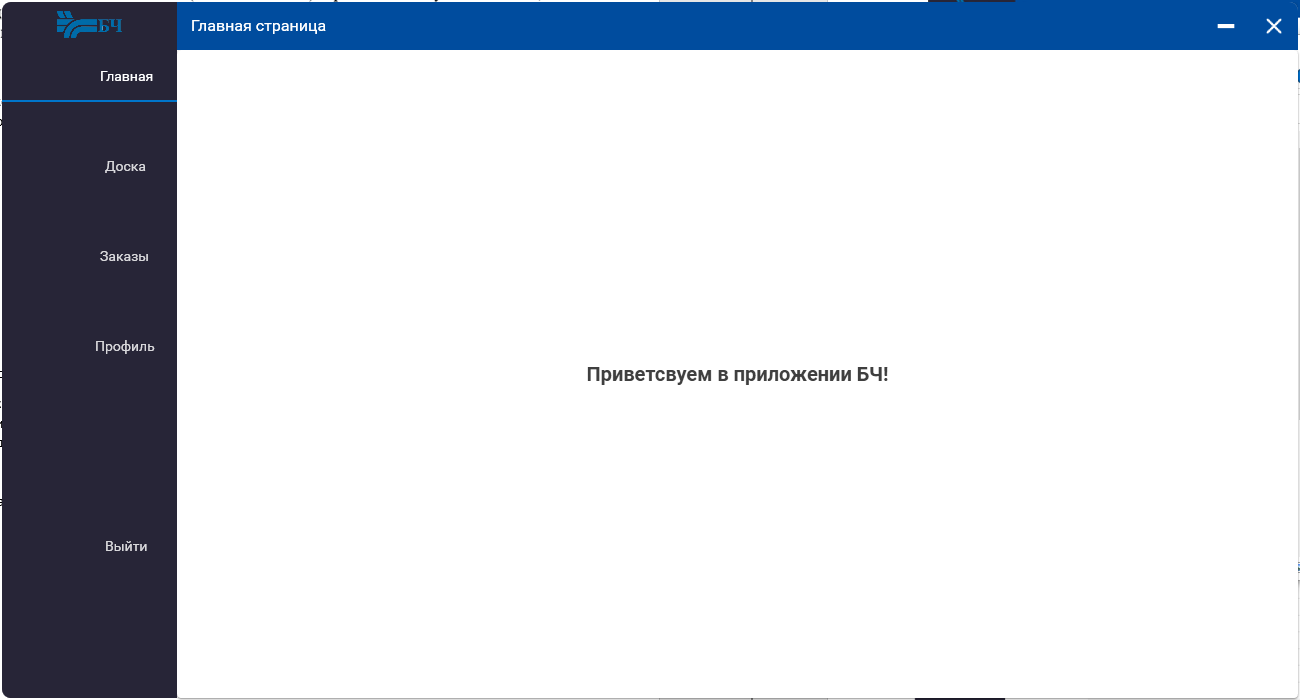


Рисунок 6.4 – Главная страница приложения (пользователь)

У пользователя же есть возможность посмотреть доску расписаний, а также воспользоваться сортировкой по предложенным столбцам расписания, для обеспечения удобства поиска нужной ячейки расписания, рисунок 6.5.

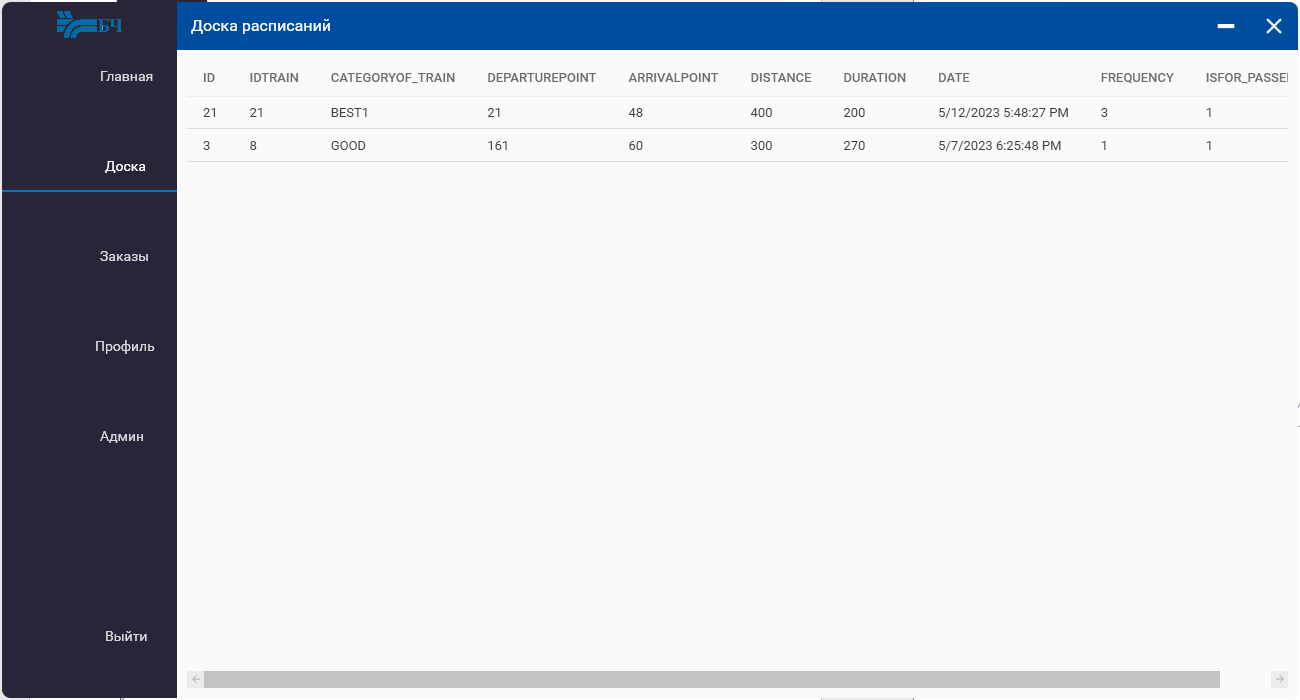


Рисунок 6.5 – Страница «Доска расписаний»

Также у пользователя есть возможность забронировать, на странице «Заказы», рисунок 6.6.

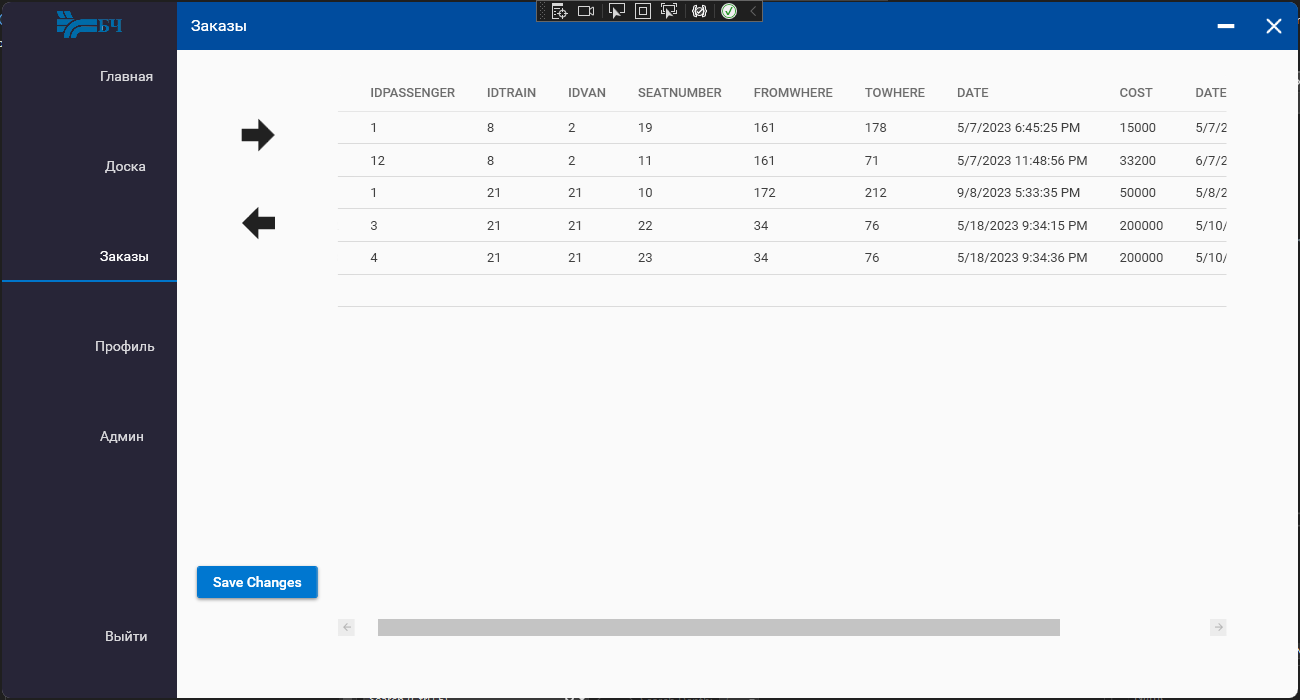


Рисунок 6.6 – Страница «Заказы»

Кроме этого, есть возможность редактировать и удалять уже имеющиеся брони билетов.

6.3. Вывод

Было составлено четкое и понятное руководство пользователя для работы с приложением. Были рассмотрены базовые функции приложения, а также предоставлены скриншоты.

Заключение

В данной работе была реализована база данных железнодорожного вокзала с использованием СУБД Oracle и средств мониторинга состояния. Целью работы было создание эффективной и надежной базы данных, которая позволит эффективно управлять данными о поездах, рейсах, билетах и пассажирах.

В рамках работы были поставлены следующие задачи: разработка модели базы данных, создание необходимых объектов (таблиц, представлений, индексов, ограничений), реализация процедур импорта и экспорта данных, тестирование производительности и разработка руководства пользователя.

Была разработана модель базы данных, состоящая из таблиц, связанных между собой отношениями. В модели были учтены основные аспекты работы железнодорожного вокзала, такие как рейсы, поезда, билеты, пассажиры, станции и т.д.

На основе разработанной модели были созданы необходимые объекты базы данных, такие как таблицы, представления, индексы и ограничения. Были использованы средства мониторинга состояния СУБД Oracle для оптимизации работы базы данных.

Важным этапом работы было тестирование производительности базы данных. Были проведены тесты выполнения запросов. Результаты тестирования показали полную работоспособность базы данных.

Было разработано руководство пользователя, которое содержит инструкции по использованию приложения базы данных железнодорожного вокзала. В нем описаны основные функции и возможности базы данных.

Как итог, работа позволила создать удобную в использовании и гибкую в управлении базу данных, которая может быть использована для управления данными о железнодорожных рейсах и пассажирах, а также может быть адаптирована под конкретные потребности пользователей.

Список использованных литературных источников

1. БЧ. Мой поезд [Электронный ресурс] / Режим доступа:

https://play.google.com/store/apps/details?id=by.rw.client/ – Дата доступа 08.03.2023.

1. Intercity.pl [Электронный ресурс] / Режим доступа:

https://www.intercity.pl/ – Дата доступа 09.03.2023.

1. Жуков А. А. Мониторинг баз данных СПб.: БХВ-Петербург, 2010. 320 с.
2. Development of Railway Station Management Information System Based on Internet of Things / J. Lin [и др.] // IEEE Access. 2019. №1. С. 25-29.
3. Форум StackOverflow [Электронный ресурс] / Справочник Режим доступа:

https://ru.stackoverflow.com/ – Дата доступа 02.04.2023.

1. Оффициальный сайт Oracle [Электронный ресурс] / Справочник Режим доступа:

https://www.oracle.com/ – Дата доступа 6.04.2023.

1. Оффициальная документация Oracle [Электронный ресурс] / Справочник Режим доступа:

https://docs.oracle.com/en/ – Дата доступа 12.04.2023.

1. Информационный портал Oracle-patches [Электронный ресурс] / Справочник Режим доступа:

https://oracle-patches.com/ – Дата доступа 18.04.2023.

1. Статья по Oracle Enterprice manager [Электронный ресурс] / Справочник Режим доступа:

https://habr.com/ru/companies/tinkoff/articles/525436/ – Дата доступа 20.04.2023.

1. Информационный портал Oracle-dba [Электронный ресурс] / Справочник Режим доступа:

https://oracle-dba.ru/docs – Дата доступа 24.04.2023.

1. Документация по Oracle PL/SQL [Электронный ресурс] / Справочник Режим доступа:

https://oracleplsql.ru/ – Дата доступа 26.04.2023.

1. Оффициальный блог Oracle [Электронный ресурс] / Справочник Режим доступа:

https://blogs.oracle.com/ – Дата доступа 01.05.2023.

1. Статья по коллекциям в Oracle [Электронный ресурс] / Справочник Режим доступа:

https://habr.com/ru/articles/254355/ – Дата доступа 04.05.2023.

Приложение

Приложение А

Структура базы данных

|  |
| --- |
| CREATE TABLE SCHEDULE (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ID\_TRAIN NUMBER(14, 0),  "DATE" DATE,  ROUTE NUMBER(14,0),  FREQUENCY NUMBER(1) CHECK (FREQUENCY IN (1,2,3,4)),  CONSTRAINT NUMBER\_OF\_TRAIN\_FK FOREIGN KEY (ID\_TRAIN) REFERENCES TRAINS("ID"),  CONSTRAINT SROUTE\_FK FOREIGN KEY (ROUTE) REFERENCES ROUTES("ID") ); |

Листинг А.1 – Скрипт создания таблицы SCHEDULE

|  |
| --- |
| CREATE TABLE TRAINS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  CATEGORY\_OF\_TRAIN NVARCHAR2(15),  IS\_FOR\_PASSENGERS Number(1) CHECK ( IS\_FOR\_PASSENGERS IN (0,1)),  VANS VANS\_COMPOSITION,  COUNT\_OF\_VANS NUMBER(2),  PARKING\_TIME NUMBER(8,0) ); |

Листинг А.2 – Скрипт создания таблицы TRAINS

|  |
| --- |
| CREATE TABLE VANS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  "TYPE" NVARCHAR2(15),  "CAPACITY" NUMBER(4,0),  IS\_FREE NUMBER(1,0) CHECK ( IS\_FREE IN (0, 1) ) ); |

Листинг А.3 – Скрипт создания таблицы VANS

|  |
| --- |
| CREATE TABLE PASSENGERS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  FULL\_NAME NVARCHAR2(50),  PASSPORT NVARCHAR2(9),  BENEFITS NUMBER(3,0) CHECK (BENEFITS < 101) ); |

Листинг А.4 – Скрипт создания таблицы PASSENGERS

|  |
| --- |
| CREATE TABLE TICKETS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ID\_PASSENGER NUMBER(14,0),  ID\_TRAIN NUMBER(14,0),  ID\_VAN NUMBER(14,0),  SEAT\_NUMBER NUMBER(4,0),  FROM\_WHERE NUMBER(14, 0),  TO\_WHERE NUMBER(14, 0),  "DATE" DATE,  COST NUMBER(8,0),  CONSTRAINT ID\_PASSENGER\_FK FOREIGN KEY (ID\_PASSENGER) REFERENCES PASSENGERS("ID"),  CONSTRAINT ID\_TRAIN\_FK FOREIGN KEY (ID\_TRAIN) REFERENCES TRAINS("ID"),  CONSTRAINT ID\_VAN\_FK FOREIGN KEY (ID\_VAN) REFERENCES VANS("ID"),  CONSTRAINT FROM\_WHERE\_FK FOREIGN KEY (FROM\_WHERE) REFERENCES STATIONS("ID"),  CONSTRAINT TO\_WHERE\_FK FOREIGN KEY (TO\_WHERE) REFERENCES STATIONS("ID") ); |

Листинг А.5 – Скрипт создания таблицы TICKETS

|  |
| --- |
| CREATE TABLE PAYMENTS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ID\_TICKET NUMBER(14,0),  DATE\_PAY DATE,  STATUS CHAR(1) CHECK ( STATUS IN ('S', 'R', 'W')),  CONSTRAINT ID\_TICKET\_FK FOREIGN KEY (ID\_TICKET) REFERENCES TICKETS("ID") ); |

Листинг А.6 – Скрипт создания таблицы PAYMENTS

|  |
| --- |
| CREATE TABLE ROUTES (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  DEPARTURE\_POINT NUMBER(14,0) NOT NULL,  ARRIVAL\_POINT NUMBER(14,0) NOT NULL,  DISTANCE NUMBER NOT NULL,  DURATION NUMBER NOT NULL,  CONSTRAINT DEPARTURE\_POINT\_FK FOREIGN KEY (DEPARTURE\_POINT) REFERENCES STATIONS(ID),  CONSTRAINT ARRIVAL\_POINT\_FK FOREIGN KEY (ARRIVAL\_POINT) REFERENCES STATIONS(ID) ); |

Листинг А.7 – Скрипт создания таблицы ROUTES

|  |
| --- |
| CREATE TABLE STATIONS (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  STATION\_NAME NVARCHAR2(30) NOT NULL,  CITY NVARCHAR2(30) NOT NULL,  STATE NVARCHAR2(30) NOT NULL,  COUNTRY NVARCHAR2(30) NOT NULL ); |

Листинг А.8 – Скрипт создания таблицы STATIONS

|  |
| --- |
| CREATE TABLE STATIONS\_ROUTES (  "ID" NUMBER(14,0) GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  ROUTE\_ID NUMBER(14,0) NOT NULL,  STATION\_ID NUMBER(14,0) NOT NULL,  STATION\_ORDER NUMBER NOT NULL,  CONSTRAINT STATION\_ID\_FK FOREIGN KEY (STATION\_ID) REFERENCES STATIONS(ID),  CONSTRAINT ROUTE\_FK FOREIGN KEY (ROUTE\_ID) REFERENCES ROUTES(ID) ); |

Листинг А.9 – Скрипт создания таблицы STATIONS\_ROUTES

Приложение Б

Листинги процедур

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *UPDATE\_VANS\_CAPACITY* AS BEGIN DECLARE  CURSOR cur IS SELECT ID\_VAN FROM TICKETS WHERE TICKETS."DATE" < *SYSDATE*;  TYPE id\_tab\_type IS TABLE OF TICKETS.ID\_VAN%TYPE INDEX BY PLS\_INTEGER;  id\_tab id\_tab\_type; BEGIN  OPEN cur;  LOOP  FETCH cur BULK COLLECT INTO id\_tab LIMIT 1000;  EXIT WHEN id\_tab.COUNT = 0;  FORALL i IN 1..id\_tab.COUNT  UPDATE VANS SET CAPACITY = CAPACITY + 1  WHERE ID = id\_tab(i);  END LOOP;  CLOSE cur; END; end; |

Листинг Б.1 – Скрипт создания процедуры UPDATE\_VANS\_CAPACITY

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_PASSENGERS(  FULLNAME IN NVARCHAR2,  PASSPOR IN NVARCHAR2,  BENEFIT IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.PASSENGERS(FULL\_NAME, PASSPORT, BENEFITS) VALUES (FULLNAME, PASSPOR, BENEFIT);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.2 – Скрипт создания процедуры INSERT\_PASSENGERS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_PASSENGERS(  ID\_IN IN NUMBER,  FULLNAME IN NVARCHAR2,  PASSPOR IN NVARCHAR2,  BENEFIT IN NUMBER)  IS  BEGIN  UPDATE MANAGER.PASSENGERS SET FULL\_NAME = FULLNAME, PASSPORT = PASSPOR,  BENEFITS = BENEFIT WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.3 – Скрипт создания процедуры UPDATE\_PASSENGERS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_PASSENGERS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.PASSENGERS WHERE ID = ID\_IN;  end; |

Листинг Б.4 – Скрипт создания процедуры DELETE\_PASSENGERS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_PAYMENTS(  IDTICKET IN NUMBER,  DATEPAY IN DATE,  STAT IN CHAR)  IS  BEGIN  INSERT INTO MANAGER.PAYMENTS(ID\_TICKET, DATE\_PAY, STATUS) VALUES (IDTICKET, DATEPAY, STAT);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  END; |

Листинг Б.5 – Скрипт создания процедуры INSERT\_PAYMENTS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_PAYMENTS(  ID\_IN IN NUMBER,  IDTICKET IN NUMBER,  DATEPAY IN DATE,  STAT IN CHAR)  IS  BEGIN  UPDATE MANAGER.PAYMENTS SET ID\_TICKET = IDTICKET, DATE\_PAY = DATEPAY,  STATUS = STAT WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.6 – Скрипт создания процедуры UPDATE\_PAYMENTS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_PAYMENTS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.PAYMENTS WHERE ID = ID\_IN;  end; |

Листинг Б.7 – Скрипт создания процедуры DELETE\_PAYMENTS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_ROUTES(  DEPARTURE\_POINT\_ID IN NUMBER,  ARRIVAL\_POINT\_ID IN NUMBER,  DISTANCE\_IN IN NUMBER,  DURATION\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.ROUTES(DEPARTURE\_POINT, ARRIVAL\_POINT, DISTANCE, DURATION)  VALUES (DEPARTURE\_POINT\_ID, ARRIVAL\_POINT\_ID, DISTANCE\_IN, DURATION\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.8 – Скрипт создания процедуры INSERT\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_ROUTES(  ID\_IN IN NUMBER,  DEPARTURE\_POINT\_ID IN NUMBER,  ARRIVAL\_POINT\_ID IN NUMBER,  DISTANCE\_IN IN NUMBER,  DURATION\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.ROUTES SET DEPARTURE\_POINT = DEPARTURE\_POINT\_ID, ARRIVAL\_POINT = ARRIVAL\_POINT\_ID,  DISTANCE = DISTANCE\_IN, DURATION = DURATION\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.9 – Скрипт создания процедуры UPDATE\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_ROUTES(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.ROUTES WHERE ID = ID\_IN;  end; |

Листинг Б.10 – Скрипт создания процедуры DELETE\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_SCHEDULE(  ID\_TRAIN\_IN IN NUMBER,  DATE\_IN IN DATE,  ROUTE\_ID IN NUMBER,  FREQUENCY\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.SCHEDULE(ID\_TRAIN, "DATE", ROUTE, FREQUENCY)  VALUES (ID\_TRAIN\_IN, DATE\_IN, ROUTE\_ID, FREQUENCY\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.11 – Скрипт создания процедуры INSERT\_SCHEDULE

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_SCHEDULE(  ID\_IN IN NUMBER,  ID\_TRAIN\_IN IN NUMBER,  DATE\_IN IN DATE,  ROUTE\_ID IN NUMBER,  FREQUENCY\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.SCHEDULE SET ID\_TRAIN = ID\_TRAIN\_IN, "DATE" = DATE\_IN,  ROUTE = ROUTE\_ID, FREQUENCY = FREQUENCY\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.12 – Скрипт создания процедуры UPDATE\_SCHEDULE

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_SCHEDULE(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.SCHEDULE WHERE ID = ID\_IN;  end; |

Листинг Б.13 – Скрипт создания процедуры DELETE\_SCHEDULE

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_STATIONS(  STATION\_NAME\_IN IN NVARCHAR2,  CITY\_IN IN NVARCHAR2,  STATE\_IN IN NVARCHAR2,  COUNTRY\_IN IN NVARCHAR2)  IS  BEGIN  INSERT INTO MANAGER.STATIONS(STATION\_NAME, CITY, STATE, COUNTRY)  VALUES (STATION\_NAME\_IN, CITY\_IN, STATE\_IN, COUNTRY\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.14 – Скрипт создания процедуры INSERT\_STATIONS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_STATIONS(  ID\_IN IN NUMBER,  STATION\_NAME\_IN IN NVARCHAR2,  CITY\_IN IN NVARCHAR2,  STATE\_IN IN NVARCHAR2,  COUNTRY\_IN IN NVARCHAR2)  IS  BEGIN  UPDATE MANAGER.STATIONS SET STATION\_NAME = STATION\_NAME\_IN, CITY = CITY\_IN,  STATE = STATE\_IN, COUNTRY = COUNTRY\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.15 – Скрипт создания процедуры UPDATE\_STATIONS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_STATIONS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.STATIONS WHERE ID = ID\_IN;  end; |

Листинг Б.16 – Скрипт создания процедуры DELETE\_STATIONS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_STATIONS\_ROUTES(  ROUTE\_ID\_IN IN NUMBER,  STATION\_ID\_IN IN NUMBER,  STATION\_ORDER\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.STATIONS\_ROUTES(ROUTE\_ID, STATION\_ID, STATION\_ORDER)  VALUES (ROUTE\_ID\_IN, STATION\_ID\_IN, STATION\_ORDER\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.17 – Скрипт создания процедуры INSERT\_STATIONS\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_STATIONS\_ROUTES(  ID\_IN IN NUMBER,  ROUTE\_ID\_IN IN NUMBER,  STATION\_ID\_IN IN NUMBER,  STATION\_ORDER\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.STATIONS\_ROUTES SET ROUTE\_ID = ROUTE\_ID\_IN, STATION\_ID = STATION\_ID\_IN,  STATION\_ORDER = STATION\_ORDER\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.18 – Скрипт создания процедуры UPDATE\_STATIONS\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_STATIONS\_ROUTES(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.STATIONS\_ROUTES WHERE ID = ID\_IN;  end; |

Листинг Б.19 – Скрипт создания процедуры DELETE\_STATIONS\_ROUTES

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_TICKETS(  ID\_PASSENGER\_IN IN NUMBER,  ID\_TRAIN\_IN IN NUMBER,  ID\_VAN\_IN IN NUMBER,  SEAT\_NUMBER\_IN IN NUMBER,  FROM\_WHERE\_IN IN NUMBER,  TO\_WHERE\_IN IN NUMBER,  DATE\_IN IN DATE,  COST\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.TICKETS(ID\_PASSENGER, ID\_TRAIN, ID\_VAN, SEAT\_NUMBER, FROM\_WHERE, TO\_WHERE, "DATE", COST)  VALUES (ID\_PASSENGER\_IN, ID\_TRAIN\_IN, ID\_VAN\_IN, SEAT\_NUMBER\_IN, FROM\_WHERE\_IN, TO\_WHERE\_IN, DATE\_IN, COST\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.20 – Скрипт создания процедуры INSERT\_TICKETS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_TICKETS(  ID\_IN IN NUMBER,  ID\_PASSENGER\_IN IN NUMBER,  ID\_TRAIN\_IN IN NUMBER,  ID\_VAN\_IN IN NUMBER,  SEAT\_NUMBER\_IN IN NUMBER,  FROM\_WHERE\_IN IN NUMBER,  TO\_WHERE\_IN IN NUMBER,  DATE\_IN IN DATE,  COST\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.TICKETS SET ID\_PASSENGER = ID\_PASSENGER\_IN, ID\_TRAIN = ID\_TRAIN\_IN,  ID\_VAN = ID\_VAN\_IN, SEAT\_NUMBER = SEAT\_NUMBER\_IN, FROM\_WHERE = FROM\_WHERE\_IN,  TO\_WHERE = TO\_WHERE\_IN, "DATE" = DATE\_IN, COST = COST\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.21 – Скрипт создания процедуры UPDATE\_TICKETS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_TICKETS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.TICKETS WHERE ID = ID\_IN;  end; |

Листинг Б.22 – Скрипт создания процедуры DELETE\_TICKETS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_TRAINS(  CATEGORY\_OF\_TRAIN\_IN IN NVARCHAR2,  IS\_FOR\_PASSENGERS\_IN IN NUMBER,  VANS\_IN IN VANS\_COMPOSITION,  COUNT\_OF\_VANS\_IN IN NUMBER,  PARKING\_TIME\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.TRAINS(CATEGORY\_OF\_TRAIN, IS\_FOR\_PASSENGERS, VANS, COUNT\_OF\_VANS, PARKING\_TIME)  VALUES (CATEGORY\_OF\_TRAIN\_IN, IS\_FOR\_PASSENGERS\_IN, VANS\_IN, COUNT\_OF\_VANS\_IN, PARKING\_TIME\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.23 – Скрипт создания процедуры INSERT\_TRAINS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_TRAINS(  ID\_IN IN NUMBER,  CATEGORY\_OF\_TRAIN\_IN IN NVARCHAR2,  IS\_FOR\_PASSENGERS\_IN IN NUMBER,  VANS\_IN IN VANS\_COMPOSITION,  COUNT\_OF\_VANS\_IN IN NUMBER,  PARKING\_TIME\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.TRAINS SET CATEGORY\_OF\_TRAIN = CATEGORY\_OF\_TRAIN\_IN, IS\_FOR\_PASSENGERS = IS\_FOR\_PASSENGERS\_IN,  VANS = VANS\_IN, COUNT\_OF\_VANS = COUNT\_OF\_VANS\_IN, PARKING\_TIME = PARKING\_TIME\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.24 – Скрипт создания процедуры UPDATE\_TRAINS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_TRAINS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.TRAINS WHERE ID = ID\_IN;  end; |

Листинг Б.25 – Скрипт создания процедуры DELETE\_TRAINS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_VANS(  TYPE\_IN IN NVARCHAR2,  CAPACITY\_IN IN NUMBER,  IS\_FREE\_IN IN NUMBER)  IS  BEGIN  INSERT INTO MANAGER.VANS("TYPE", "CAPACITY", IS\_FREE)  VALUES (TYPE\_IN, CAPACITY\_IN, IS\_FREE\_IN);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.26 – Скрипт создания процедуры INSERT\_VANS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_VANS(  ID\_IN IN NUMBER,  TYPE\_IN IN NVARCHAR2,  CAPACITY\_IN IN NUMBER,  IS\_FREE\_IN IN NUMBER)  IS  BEGIN  UPDATE MANAGER.VANS SET "TYPE" = TYPE\_IN, CAPACITY = CAPACITY\_IN,  IS\_FREE = IS\_FREE\_IN WHERE ID = ID\_IN;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.27 – Скрипт создания процедуры UPDATE\_VANS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_VANS(  ID\_IN NUMBER  )  IS BEGIN  DELETE FROM MANAGER.VANS WHERE ID = ID\_IN;  end; |

Листинг Б.28 – Скрипт создания процедуры DELETE\_VANS

|  |
| --- |
| CREATE OR REPLACE PROCEDURE INSERT\_TAKE\_TICKET(  p\_id\_passenger IN NUMBER,  p\_id\_train IN NUMBER,  p\_id\_van IN NUMBER,  p\_seat\_number IN NUMBER,  p\_from\_where IN VARCHAR2,  p\_to\_where IN VARCHAR2,  p\_date IN DATE,  p\_cost IN NUMBER,  p\_date\_pay IN DATE,  p\_status IN CHAR  )  IS  ID\_TICK NUMBER;  BEGIN  INSERT INTO MANAGER.TICKETS (ID\_PASSENGER, ID\_TRAIN, ID\_VAN, SEAT\_NUMBER, FROM\_WHERE, TO\_WHERE, "DATE", COST)  VALUES (p\_id\_passenger, p\_id\_train, p\_id\_van, p\_seat\_number, p\_from\_where, p\_to\_where, p\_date, p\_cost);  SELECT ID INTO ID\_TICK  FROM MANAGER.TICKETS  WHERE ID\_PASSENGER = p\_id\_passenger  AND ID\_TRAIN = p\_id\_train  AND ID\_VAN = p\_id\_van  AND SEAT\_NUMBER = p\_seat\_number  AND FROM\_WHERE = p\_from\_where  AND TO\_WHERE = p\_to\_where  AND "DATE" = p\_date  AND COST = p\_cost;  INSERT INTO MANAGER.PAYMENTS (ID\_TICKET, DATE\_PAY, STATUS)  VALUES (ID\_TICK, p\_date\_pay, p\_status);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.29 – Скрипт создания процедуры INSERT\_TAKE\_TICKET

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_TAKE\_TICKET(  p\_id IN NUMBER,  p\_id\_passenger IN NUMBER,  p\_id\_train IN NUMBER,  p\_id\_van IN NUMBER,  p\_seat\_number IN NUMBER,  p\_from\_where IN VARCHAR2,  p\_to\_where IN VARCHAR2,  p\_date IN DATE,  p\_cost IN NUMBER,  p\_date\_pay IN DATE,  p\_status IN VARCHAR2  )  IS  BEGIN  UPDATE MANAGER.TICKETS SET  ID\_PASSENGER = p\_id\_passenger,  ID\_TRAIN = p\_id\_train,  ID\_VAN = p\_id\_van,  SEAT\_NUMBER = p\_seat\_number,  FROM\_WHERE = p\_from\_where,  TO\_WHERE = p\_to\_where,  "DATE" = p\_date,  COST = p\_cost  WHERE ID = p\_id;  UPDATE MANAGER.PAYMENTS SET  DATE\_PAY = p\_date\_pay,  STATUS = p\_status  WHERE ID\_TICKET = p\_id;  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг Б.30 – Скрипт создания процедуры UPDATE\_TAKE\_TICKET

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_TAKE\_TICKET(  p\_id IN NUMBER  )  IS BEGIN  DELETE FROM PAYMENTS WHERE ID\_TICKET = p\_id;  DELETE FROM TICKETS WHERE ID = p\_id;  end; |

Листинг Б.31 – Скрипт создания процедуры DELETE\_TAKE\_TICKET

|  |
| --- |
| CREATE OR REPLACE PROCEDURE TO\_JSON(return\_out out int)  IS BEGIN  DECLARE  output\_file UTL\_FILE.FILE\_TYPE;  json\_data CLOB;  CURSOR vans\_cursor IS  SELECT JSON\_OBJECT(  'type' VALUE TYPE,  'capacity' VALUE CAPACITY,  'is\_free' VALUE IS\_FREE  ) AS json\_data  FROM VANS  WHERE ROWNUM < 10;  BEGIN  output\_file := UTL\_FILE.FOPEN('UTL\_DIR', 'VANS.JSON', 'W');  FOR van\_rec IN vans\_cursor LOOP  json\_data := van\_rec.json\_data;  UTL\_FILE.PUT\_LINE(output\_file, json\_data);  END LOOP;  UTL\_FILE.FCLOSE(output\_file);  return\_out := 1;  END;  end; |

Листинг Б.32 – Скрипт создания процедуры TO\_JSON

|  |
| --- |
| CREATE OR REPLACE PROCEDURE FROM\_JSON(RETURN\_OUT OUT NUMBER)  AS BEGIN  INSERT INTO VANS("TYPE", CAPACITY, IS\_FREE)  SELECT "type", capacity, is\_free  FROM JSON\_TABLE(BFILENAME('UTL\_DIR', 'VANS.JSON'), '$[\*]'  COLUMNS (  "type" NUMBER PATH '$.type',  capacity VARCHAR2(50) PATH '$.capacity',  is\_free NUMBER PATH '$.is\_free'  )  );  RETURN\_OUT := 1;  end; |

Листинг Б.33 – Скрипт создания процедуры FROM\_JSON

Приложение В

Листинги триггеров

|  |
| --- |
| CREATE OR REPLACE TRIGGER increase\_van\_capacity  AFTER INSERT OR UPDATE ON TICKETS  BEGIN  UPDATE\_VANS\_CAPACITY();  END; |

Листинг В.1 – Скрипт создания триггера INCREASE\_VAN\_CAPACITY

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_VAN  BEFORE INSERT OR UPDATE ON TICKETS  FOR EACH ROW  DECLARE VANS VANS\_COMPOSITION := VANS\_COMPOSITION();  BOOL NUMBER(1,0) := 0;  BEGIN  SELECT VANS INTO VANS FROM TRAINS WHERE ID = :NEW.ID\_TRAIN;  FOR I IN VANS.FIRST .. VANS.LAST LOOP  IF VANS(I) = :NEW.ID\_VAN THEN  BOOL := 1;  END IF;  END LOOP;  IF BOOL = 0 THEN  :NEW.ID\_VAN := NULL;  RAISE\_APPLICATION\_ERROR(-20001, 'VAN NOT IN TRAIN');  end if;  end; |

Листинг В.2 – Скрипт создания триггера CHECK\_VAN

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_SEAT  BEFORE INSERT OR UPDATE ON TICKETS  FOR EACH ROW  DECLARE FREE NUMBER;  BEGIN  IF :NEW.SEAT\_NUMBER != :OLD.SEAT\_NUMBER OR INSERTING  SELECT COUNT(\*) INTO FREE FROM TICKETS WHERE SEAT\_NUMBER = :NEW.SEAT\_NUMBER AND ID\_VAN = :NEW.ID\_VAN;  IF FREE > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001,'THIS PLACE ALREADY USED');  end if;  end if;  end; |

Листинг В.3 – Скрипт создания триггера CHECK\_SEAT

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_ROUTE  BEFORE INSERT OR UPDATE ON TICKETS  FOR EACH ROW  DECLARE FROM\_BOOL NUMBER;  TO\_BOOL NUMBER;  BEGIN  SELECT COUNT(\*) INTO FROM\_BOOL FROM ROUTES  JOIN STATIONS\_ROUTES ON ROUTES.ID = STATIONS\_ROUTES.ROUTE\_ID  JOIN SCHEDULE S on ROUTES.ID = S.ROUTE  WHERE STATION\_ID = :NEW.FROM\_WHERE AND S.ID\_TRAIN = :NEW.ID\_TRAIN;  SELECT COUNT(\*) INTO TO\_BOOL FROM ROUTES  JOIN STATIONS\_ROUTES ON ROUTES.ID = STATIONS\_ROUTES.ROUTE\_ID  JOIN SCHEDULE S on ROUTES.ID = S.ROUTE  WHERE STATION\_ID = :NEW.TO\_WHERE AND S.ID\_TRAIN = :NEW.ID\_TRAIN;  IF FROM\_BOOL = 0 OR TO\_BOOL = 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'DOES NOT EXIST THIS ROUTE');  end if;  end; |

Листинг В.4 – Скрипт создания триггера CHECK\_ROUTE

|  |
| --- |
| CREATE OR REPLACE TRIGGER update\_van\_capacity  BEFORE INSERT OR UPDATE ON PAYMENTS  FOR EACH ROW  WHEN (NEW.STATUS = 'S')  DECLARE  VAN\_CAPACITY NUMBER;  BEGIN  -- Получаем вместимость вагона из таблицы VANS  SELECT CAPACITY INTO VAN\_CAPACITY  FROM VANS  WHERE ID = (SELECT ID\_VAN FROM TICKETS WHERE ID = :NEW.ID\_TICKET);  -- Проверяем, достигнута ли максимальная вместимость вагона  IF VAN\_CAPACITY = 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'The capacity of the van is zero. Cannot insert a new row into the PAYMENTS table.');  END IF;  -- Уменьшаем значение столбца CAPACITY в таблице VANS  UPDATE VANS  SET CAPACITY = CAPACITY - 1  WHERE ID = (SELECT ID\_VAN FROM TICKETS WHERE ID = :NEW.ID\_TICKET);  END; |

Листинг В.5 – Скрипт создания триггера UPDATE\_VAN\_CAPACITY

|  |
| --- |
| CREATE OR REPLACE TRIGGER TRAINS\_INSERT\_UPDATE\_TRIGGER  BEFORE INSERT OR UPDATE ON TRAINS  FOR EACH ROW  DECLARE  vans\_in VANS\_COMPOSITION := VANS\_COMPOSITION();  vans\_out VANS\_COMPOSITION := VANS\_COMPOSITION();  vansfree NUMBER(1, 0);  BEGIN  IF INSERTING THEN  vans\_in := :NEW.VANS;  vans\_out := :NEW.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  IF vansfree = 0 THEN  vans\_out := DELETE\_ENTRY(vans\_out, vans\_in(I));  ELSE  UPDATE VANS SET IS\_FREE = 0 WHERE ID = vans\_in(I);  END IF;  END LOOP;  :NEW.COUNT\_OF\_VANS := vans\_out.COUNT;  :NEW.VANS := vans\_out;  ELSIF UPDATING('VANS') THEN  vans\_in := :OLD.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  UPDATE VANS SET IS\_FREE = 1 WHERE ID = vans\_in(I);  END LOOP;  vans\_in := :NEW.VANS;  FOR I IN vans\_in.FIRST .. vans\_in.LAST LOOP  SELECT IS\_FREE INTO vansfree FROM VANS WHERE ID = vans\_in(I);  IF vansfree = 0 THEN  vans\_out := DELETE\_ENTRY(vans\_out, vans\_in(I));  ELSE  UPDATE VANS SET IS\_FREE = 0 WHERE ID = vans\_in(I);  END IF;  END LOOP;  :NEW.VANS := vans\_out;  :NEW.COUNT\_OF\_VANS := vans\_out.COUNT;  END IF;  END; |

Листинг В.6 – Скрипт создания триггера TRAINS\_INSERT\_UPDATE\_TRIGGER