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

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных каршеринга с применением технологий мультимедийных типов данных в БД»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А.Ободов

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Харланович

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Харланович

подпись дата инициалы и фамилия

**Содержание**

[Введение 5](#_Toc134952400)

[1 Анализ требований к программному средству 5](#_Toc134952401)

[1.1 Аналитический обзор аналогов 5](#_Toc134952402)

[1.1.1 Аналог Hello 5](#_Toc134952403)

[1.1.2 Аналог Anytime 5](#_Toc134952404)

[1.2 Разработка функциональных требований, определение вариантов использования 5](#_Toc134952405)

[1.3 Вывод 5](#_Toc134952406)

[2 Разработка архитектуры проекта 5](#_Toc134952407)

[2.1 Обобщенная структура управлением приложения 5](#_Toc134952408)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 5](#_Toc134952409)

[3 Разработка модели базы данных 5](#_Toc134952410)

[3.1 Создание необходимых объектов 5](#_Toc134952411)

[3.1.1 Представления базы данных 5](#_Toc134952412)

[3.1.2 Индексы базы данных 5](#_Toc134952413)

[3.1.3 Триггеры базы данных 5](#_Toc134952414)

[3.2 Описание используемой технологии 5](#_Toc134952415)

[3.3 Вывод 5](#_Toc134952416)

[4 Установка, настройка и использование Oracle DB 19c 5](#_Toc134952417)

[4.1 Установка Oracle 5](#_Toc134952418)

[4.2 Создание таблиц 5](#_Toc134952419)

[4.3 Создание ролей для разграничения 5](#_Toc134952420)

[4.4 Создание пакетов процедур для базы данных 5](#_Toc134952421)

[4.4.1 Выборка данных из таблиц 5](#_Toc134952422)

[4.4.2 Выборка данных по поисковому запросу 5](#_Toc134952423)

[4.4.3 Заполнение таблиц 100 000 строк 5](#_Toc134952424)

[4.4.4 Добавление данных в таблицы 5](#_Toc134952425)

[4.4.5 Удаление данных в таблицы 5](#_Toc134952426)

[4.4.6 Изменение данных в таблицы 5](#_Toc134952427)

[4.7 Дополнительные функции 5](#_Toc134952428)

[4.5 Описание процедур экспорта и импорта 5](#_Toc134952429)

[4.6 Вывод 5](#_Toc134952430)

[5 Тестирование 5](#_Toc134952431)

[5.1 Тестирование производительности базы данных 5](#_Toc134952432)

[5.2 Вывод 5](#_Toc134952433)

[6 Руководство по использованию программного средства 5](#_Toc134952434)

[6.1 Руководство пользователя 5](#_Toc134952435)

[6.2 Установка приложения 5](#_Toc134952436)

[6.3 Вывод 5](#_Toc134952437)

[Заключение 5](#_Toc134952438)

[Список литературных источников 5](#_Toc134952439)

[Приложение А 5](#_Toc134952440)

# Введение

Цель данной работы заключается в создании реляционной базы данных для каршеринга, которая обеспечивает пользователя доступом к имеющимся на ней автомобилям. В рамках работы также необходимо разработать соответствующее приложение, которое будет демонстрировать функциональность базы данных и обеспечивать клиентов доступом к различным маркам автомобилей.

База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Oracle, поскольку эта система обладает высокой надежностью и производительностью, а также позволяет работать с большим объёмом данных.

Для гарантированной безопасности пользователей приложения в моей курсовой работе применяется метод хэширования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении изображений.

Основные требования к приложению:

* Реализация ролей администратора, менеджера и пользователя.
* Предоставление каталога и поиск автомобилей по названию.
* Фильтрация автомобилей по кузову и популярности.
* Добавление автомобилей менеджером.
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Каршеринг - это услуга по аренде автомобиля на короткий срок, которая позволяет пользователям бронировать автомобиль в аренду на несколько часов или дней и возвращать его в заданное время и место. Каршеринг может быть организован как с использованием специальных автомобилей, которые стоят на стоянках в городе, так и с использованием автомобилей частных владельцев, которые разрешают арендовать свои автомобили через специальные приложения или платформы.

Из плюсов можно отметить, что каршеринг позволяет людям экономить деньги на покупке и содержании личного автомобиля, а также снижает загруженность дорог и уменьшает выбросы вредных веществ в атмосферу. Более того пользователи могут бронировать транспорт в любое время онлайн.

### 1.1.1 Аналог Hello

Hello [1] - это популярный каршеринг, который позволяет пользователям бронировать автомобили различных марок и кузова онлайн. Каршеринг Hello предоставляет гибкую и удобную альтернативу общественному транспорту и такси. Пример интерфейса данного сервиса представлен на рисунке 1.1.

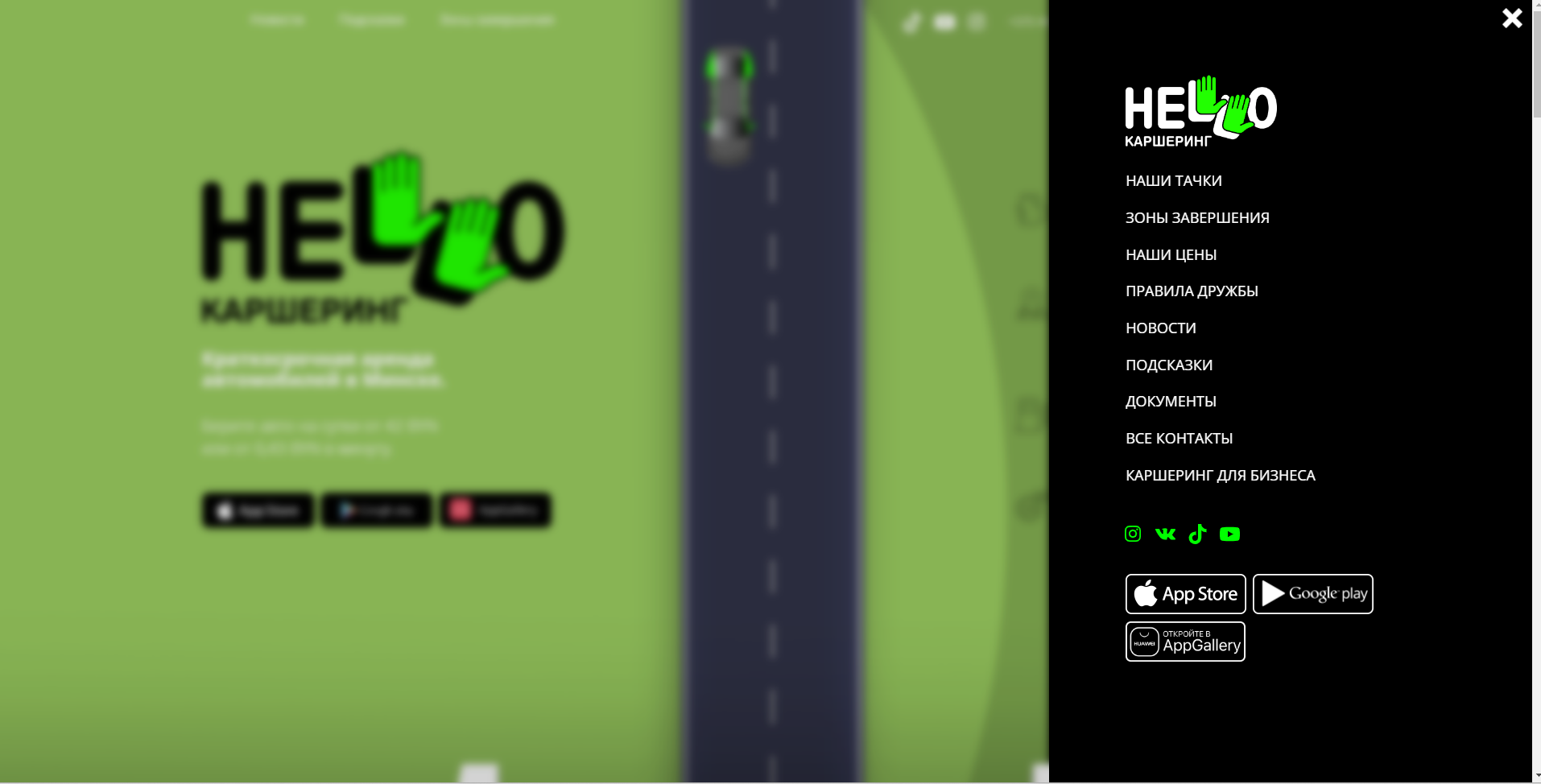


Рисунок 1.1 – Интерфейс сервиса Hello

Одной из основных функций Hello является подбор нужного автомобиля. Пользователи могут выбирать нужный автомобиль по его техническим характеристикам и цене. Кроме того, Hello предлагает мобильное приложение, в котором можно отследить забронированный автомобиль.

Пользователи могут выбрать понравившейся автомобиль выбрать оплату сразу или после истечения выбранного срока, заправка осуществляется за счёт сервиса, входит в стоимость, можно бронировать автомобиль как по расстоянию, так и по времени .

Другая функция Hello - это быстрая регистрация и интеграция. Для того чтобы стать клиентом сервиса достаточно ввести свой мобильный номер телефона. Также регистрация доступна через почту и пароль.

Hello предоставляет мобильное приложение, которое позволяет пользователям отслеживать информацию о своих забронированных автомобилях. Они могут узнать о местоположении автомобиля, его статусе и времени окончания бронирования.

Кроме основных функций, Hello также предлагает ряд дополнительных возможностей, чтобы сделать процесс использования автомобиля более удобным и безопасным. Пользователи могут открыть и запустить автомобиль, используя мобильное приложение Hello в качестве виртуального ключа. Это удобно, так как нет необходимости носить с собой физический ключ и можно легко поделиться доступом к автомобилю с другими пользователями. Hello предоставляет круглосуточную службу поддержки, доступную через мобильное приложение или по телефону. Пользователи могут обратиться за помощью или задать вопросы в любое время, связанные с бронированием, использованием автомобиля или другими вопросами.

Hello использует несколько баз данных в своей архитектуре, включая PostgreSQL и Redis – высокопроизводительная базой данных с открытым исходным кодом, ориентированная на хранение и обработку данных в оперативной памяти. Она может использоваться для кэширования часто запрашиваемых данных в системе каршеринга, улучшая производительность и отзывчивость. Служит для хранения метаданных автомобилей, включая информацию об технических характеристиках, моделях, состояниях автомобилей и т.д. PostgreSQL используется для хранения информации о пользовательских аккаунтах и настроек, поскольку предоставляет высокую надёжность. Redis позволяет совершать обработку потоков данных, позволяется взаимодействовать между элементами Hello, такими как поисковая система, упорядочивание по критериям т.д.

В общем, Hello использует надёжные и высокопроизводительные базы данных для обеспечения быстрого доступа к большому ассортименту автомобилей и обработки потоковых данных для предоставления разделов каталога.

### 1.1.2 Аналог Anytime

Anytime [2] - это популярная платформа каршеринга, которая предоставляет удобные и гибкие возможности аренды автомобилей. Anytime позволяет пользователям арендовать автомобили на определенное время и возвращать их по окончании использования. Сервис начинал свою работу с 20 машин. Сейчас в каршеринге насчитывается более 570 машин Он позволяет пользователям бронировать автомобили в любое удобное время, чтобы не тратить средства на покупку личного авто.

Anytime предлагает большой ассортимент автомобилей в различном кузове. Сервис также имеет категории авто «Хит» и «Популярные».

Пользователи могут отслеживать свои автомобили и просматривать сколько времени осталось до завершения аренды.

Сервис Anytime предлагает удобные опции получения и возврата автомобилей. Пользователи могут забрать автомобиль из специально оборудованных станций каршеринга, расположенных в различных частях города. После использования автомобиля его можно вернуть на любую доступную станцию.

Anytime стремится предоставить удобный и доступный способ аренды автомобилей, предоставляя пользователям большой выбор, гибкость и простоту использования.

Интерфейс Anytime схож с интерфейсом Hello и предлагает удобный выбор автомобиля, указывая его технические характеристики. Также сервис предлагает свои авто для различных бизнес-компаний. Пример интерфейса данного сервиса представлен на рисунке 1.2.

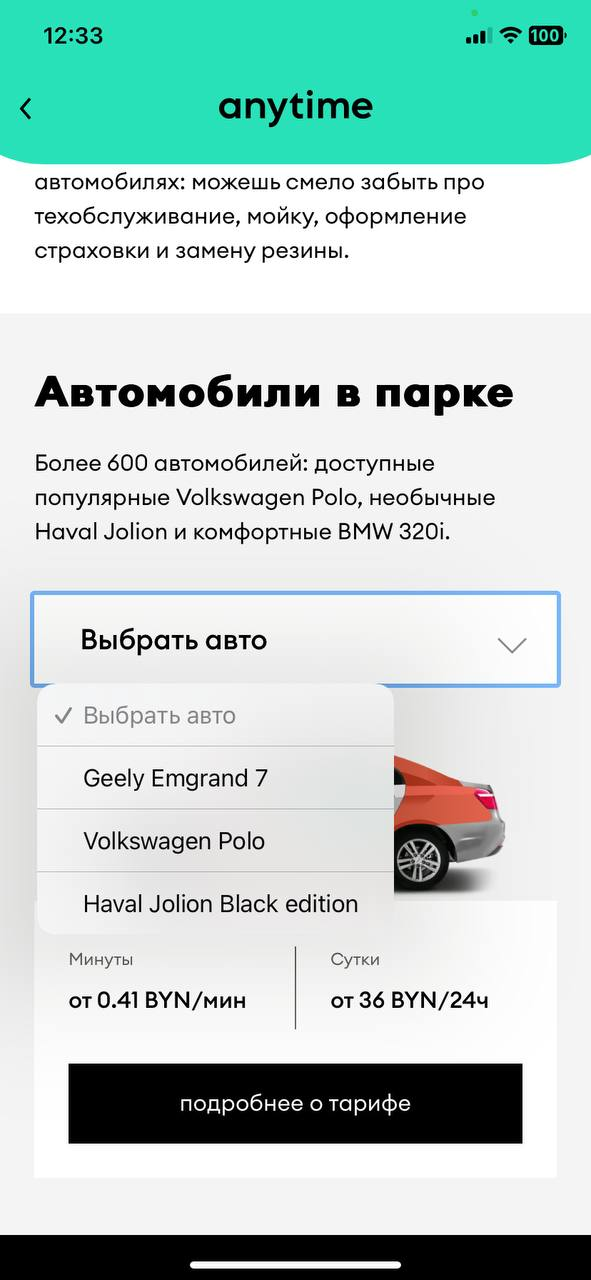


Рисунок 1.2 – Интерфейс сервиса Anytime

Anytime использует несколько баз данных в своей архитектуре, включая MongoDB и Apache Cassandra – то распределенная NoSQL база данных, способная обрабатывать большие объемы данных и обеспечивать высокую доступность. Она может быть полезной для хранения данных о прокате автомобилей, пользовательских аккаунтах и других сущностях в системе каршеринга.

MongoDB служит для хранения метаданных автомобилей, включая информацию о заказах, бронировании и т.д. Она хранит данные в формате JSON-подобных документов. Она обеспечивает гибкость и масштабируемость, что может быть полезно для хранения и обработки данных в системе каршеринга.

В общей сложности, Anytime использует объёмные и структурированные базы данных для надежного обеспечения доступа к данным и обработки этих данных для стабильной работы сервиса.

## 1.2 Разработка функциональных требований, определение вариантов использования

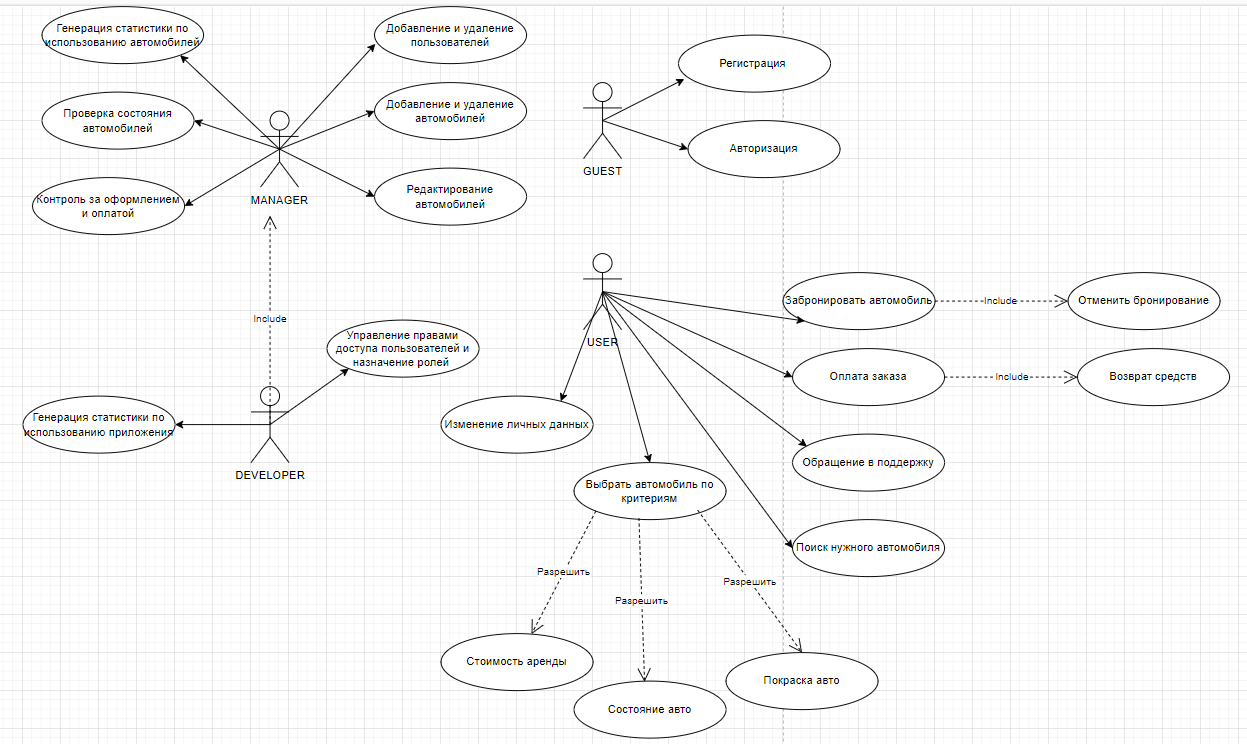
Функциональные требования базы данных определяют, как информационная система должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя определение способа хранения и организации данных, механизмы поиска и выборки информации, методы обновления данных и механизмы защиты данных. Система должна обеспечивать механизмы аутентификации пользователей, чтобы убедиться в их подлинности перед предоставлением доступа к данным. Это может включать функции, такие как регистрация пользователей, проверка учетных данных и предоставление соответствующих разрешений на доступ в зависимости от роли пользователя. Кроме того, функциональные требования могут определять взаимодействие базы данных с другими системами и программными компонентами. Например, в случае каршеринга функциональные требования могут включать функции для хранения информации об автомобилях и клиентах, выбора автомобилей на основе различных категорий и критериев, а также функции для обработки платежей и отслеживания состояния автомобилей.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Роли пользователя — это набор прав, которые пользователь может получить в системе. Роли обычно назначаются пользователям в зависимости от их позиции, роли или ответственности в организации. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Guest;
* User;
* Developer;
* Manager.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.3.

Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности бронирования авто. После регистрации пользователь становится пользователем User.

После регистрации роль пользователя User получает возможность бронировать автомобиль или отменять бронирование, оплачивать бронирование или отменять заказ. Совершать поиск нужного автомобиля как по названию, так и по критериям таким как цена, состояние и окрашивание. Кроме того, есть возможность обратиться в поддержку за консультацией, а также изменять свои данные.

Роль администратора Developer заключается в управлении сервисом и мониторинге действий пользователей. Администратор имеет доступ к просмотру информации о всех пользователях, в том числе их действиях. При необходимости администратор может редактировать информацию автомобилей и пользователей, добавлять новые, удалять старые. Также может управлять правами доступа и назначать поли пользователей. Также роль Developer получает возможность совершать мониторинг использования приложения.

Роль Manager заключается в управлении автопарком и мониторинге состояния автомобилей. Manager также имеет доступ к просмотру информации о всех пользователях и автомобилях. При необходимости Manager может редактировать информацию о автомобилях и пользователях, например, изменять категории у автомобилей. Помимо этого, роль Manager получает возможность совершать мониторинг использования автомобилей.

В данном разделе были определены роли пользователей и разработаны варианты использования системы в зависимости от этих ролей. Гость может только просматривать информацию, пользователь получает доступ к функциям системы, таким как бронирование автомобилей, выбор по критериям , менеджер имеет права на управление сервисом и редактирование информации о пользователях и автомобилях, разработчик же имеет все привилегии и способствует контролю стабильности сервиса

## 1.3 Вывод

Итого, был проведен аналитический обзор аналогов приложений каршеринга и сервисов, которые уже существуют на рынке. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Также были определены функциональные требования базы данных, а также роли пользователей и варианты использования системы в зависимости от этих ролей. Была разработана UML-диаграмма, на которой отображены основные функции, которые доступны для каждой из ролей пользователей.

1. Разработка архитектуры проекта

## Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для каршеринга может входить возможность быстрого и удобного выбора автомобилей для последующего бронирование, разделение автомобилей на критерии кузова, цен, популярности. Также пользователи должны иметь возможность просматривать каталог автомобилей, обращаться за консультацией в поддержку. Для удобства пользователей также должна быть предусмотрена возможность просмотра истории платежей.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) - это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

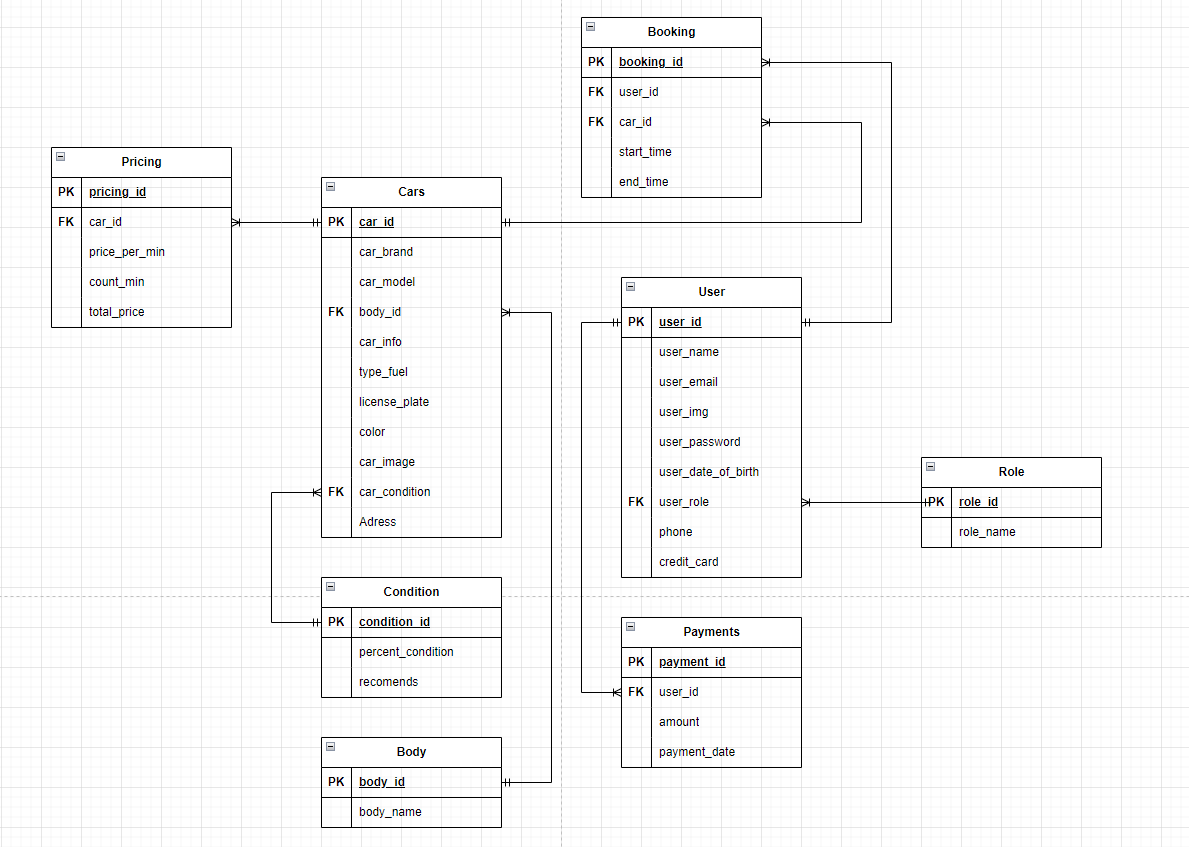


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному". Например, таблица Users связана с таблицами Booking, Payments через внешние ключи user\_id. Также видно, что таблица Role связана с таблицей Users, а таблица Cars связана с таблицами Condition, Pricing, Body,Booking.

**2.3 Описание информационных объектов**

Для реализации базы данных было разработано 8 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Role, User, Booking, Payments, Cars, Condition, Body, Pricing. Ниже мы разберем каждую из них более подробно.

Таблица Role представляет собой перечень ролей, которые могут иметь пользователи. Имеющиеся поля: role\_id (идентификатор роли, типа NUMBER(10), первичный ключ), role\_name (текстовое название роли, тип VARCHAR2(50) NOT NULL).

Таблица Users содержит информацию о пользователях. Имеющиеся поля: user\_id (идентификатор пользователя, типа NUMBER(10), первичный ключ), user\_name (имя пользователя, тип VARCHAR2(50) NOT NULL), user\_img (изображение пользователя, тип BLOB), user\_email (адрес электронной почты пользователя, тип VARCHAR2(50) NOT NULL UNIQUE), user\_password (пароль пользователя, тип VARCHAR2(128)), user\_date\_of\_birth (дата рождения пользователя, тип DATE), user\_role (идентификатор роли пользователя, тип NUMBER(10), внешний ключ ссылающийся на role\_id в таблице Role), phone (мобильный телефон пользователя, тип VARCHAR2(20) NOT NULL UNIQUE), credit\_card(валютная карта пользователя, тип VARCHAR2(20) NOT NULL UNIQUE).

Таблица Cars содержит информацию о автомобилях. Имеющиеся поля: cars\_id (идентификатор автомобиля, типа NUMBER(10), первичный ключ), card\_brand (марка автомобиля, тип VARCHAR(20) NOT NULL), card\_model (модель автомобиля, тип VARCHAR(30) NOT NULL), body\_id (уникальный идентификатор кузова автомобиля, тип NUMBER(10) NOT NULL , внешний ключ ссылающийся на body\_id в таблице Body), car\_info (описание автомобиля, тип CLOB NOT NULL), type\_fuel (тип двигателя автомобиля, тип VARCHAR(20) NOT NULL), license\_plate (регистрационный номер автомобиля, тип VARCHAR2(20) NOT NULL UNIQUE), color( цвет автомобиля, тип VARCHAR(20) NOT NULL), car\_image(фото автомобиля, тип BLOB NOT NULL), car\_condition (уникальный идентификатор состояния автомобиля, тип NUMBER(10) NOT NULL, внешний ключ ссылающейся на condition\_id в таблице Condition, Adress (адрес местонахождения автомобиля тип VARCHAR2(50) ).

Таблица Body содержит информацию об типе кузова автомобиля. Имеющиеся поля: body\_id (идентификатор типа кузова, типа NUMBER(10), первичный ключ), body\_name (название типа кузова, тип VARCHAR(20) NOT NULL UNIQUE).

Таблица Condition содержит информацию о состоянии автомобиля. Имеющиеся поля: condition\_id (идентификатор состояния автомобиля, типа NUMBER, первичный ключ), percent\_condition (процентный показатель состояния автомобиля, тип NUMBER(5,2) NOT NULL), recommends (рекомендации по ремонту или утилизации автомобиля, тип VARCHAR2(50) NOT NULL).

Таблица Pricing содержит информацию о стоимости автомобилей. Имеющиеся поля: pricing\_id (идентификатор стоимости, типа NUMBER(10), первичный ключ), car\_id (идентификатор автомобиля, тип NUMBER(10) NOT NULL, внешний ключ ссылающийся на car\_id в таблице Cars), price\_per\_min ( стоимость аренды за одну минуту, тип NUMBER(10,2) NOT NULL), count\_min (количество арендованных минут, тип NUMBER(10)), total\_price (полная стоимость аренды автомобиля, тип NUMBER(10)).

Таблица Booking содержит информацию о заказах автомобилей. Имеющиеся поля: booking\_id (идентификатор заказа, типа NUMBER(10), первичный ключ), user\_id (идентификатор пользователя, заказавшего автомобиль, тип NUMBER(10), внешний ключ ссылающийся на user\_id в таблице Users) и car\_id (идентификатор автомобиля, тип NUMBER(10)), start\_time(время начало заказа, тип DATE), end\_time(время окончания заказа, тип DATE).

Таблица Payments содержит платежи пользователей. Имеющиеся поля: payment\_id (идентификатор платежей, типа NUMBER(10), первичный ключ), user\_id (идентификатор пользователя, сделавшего платёж, тип NUMBER(10), внешний ключ ссылающийся на user\_id в таблице Users) и amount (показывает количество платежей пользователя, тип NUMBER(30)), payment\_date(дата совершения платежа, тип DATE).

**2.4 Вывод**

Проектирование архитектуры проекта необходимо для определения структуры и функциональности приложения. Общая структура управления приложением позволяет определить, какие компоненты требуются для реализации приложения и как они должны взаимодействовать друг с другом.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении и как они будут храниться и обрабатываться. В данном проекте были описаны объекты, такие как участники, роли, типы кузова автомобилей, автомобили, состояние автомобилей, платежи, заказы и цены на автомобили.

В общем, проектирование архитектуры проекта является важным этапом в процессе разработки приложения, поскольку оно позволяет определить необходимые компоненты и информационные объекты, которые помогут создать функциональное и эффективное приложение.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для каршеринга необходимо создать несколько таблиц, которые будут хранить данные о пользователях, автомобилях, фотографиях, платежах и заказах.

Для каршеринга необходимо создать следующие таблицы:

Role - таблица, содержащая роли пользователей. В этой таблице будут храниться записи с ролями, такими как администратор, модератор, пользователь и т.д.

Users - таблица, содержащая данные о пользователе. В этой таблице будут храниться информация о пользователе, такая как имя, фотография, электронная почта, дата рождения и т.д.

Body - таблица, содержащая типы кузова авто. В этой таблице будут храниться записи о типах кузова автомобилей.

Cars - таблица, содержащая информацию об автомобилях. В этой таблице будут храниться данные об автомобилях, такие как название, модель, изображение, регистрационный номер, тип топлива и т.д.

Booking - таблица, содержащая отношение между пользователем и автомобилями. В этой таблице будут храниться записи бронирования автомобилей пользователями.

Condition - таблица, содержащая информацию о состоянии автомобилей. В этой таблице будут храниться данные о том, как скоро нуждается автомобиль в обслуживании.

Pricing - таблица, содержащая информацию о ценах автомобилей. В этой таблице будут храниться данные о стоимости аренды автомобиля на промежуток времени.

Payments - таблица, содержащая информацию о платежах пользователей. В этой таблице будут храниться записи о тех пользователях, которые совершают платежи.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, можно создать индексы на столбцах user\_email в таблице Users, чтобы быстро находить информацию о пользователе по его электронной почте.

Для более удобной работы с базой данных можно создать несколько функций и процедур. Например, можно создать функцию для поиска и сортировки автомобилей по критериям, таким как тип кузова, цена, название и т.д. Также можно создать процедуру для добавления новых автомобилей в базу данных.

### 3.1.1 Представления базы данных

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы пять представлений:

* all\_info\_user, которое объединяет данные таблиц Users и Role;
* all\_info\_car, которое содержит информацию об автомобилях, их типе кузова и состояние каждого автомобиля;
* pricing\_car, которое содержит информацию о всех ценах автомобилей;
* all\_payments, которое содержит информацию о всех платежах пользователей;
* all\_booking, которое содержит информацию о всех бронированиях автомобилей.

Представление all\_info\_user было создано для того, чтобы получить полную информацию о пользователях и их ролях, объединив данные из двух таблиц. Оно будет на листинге 3.1. Остальные представления будут аналогичны, только будут работать с другими таблицами.

|  |
| --- |
| CREATE OR REPLACE VIEW ALL\_INFO\_USER AS  SELECT USERS.USER\_ID, USER\_NAME, USER\_EMAIL, USER\_IMG, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, PHONE, USER\_PASSPORT, CREDIT\_CARD, ROLE.ROLE\_NAME FROM USERS  JOIN ROLE ON USERS.USER\_ROLE = ROLE.ROLE\_ID; |

Листинг 3.1 – Представление all\_info\_track

Представление all\_info\_car было создано для того, чтобы получить полную информацию о автомобилях, их типах кузова и состоянии, объединив данные из трёх таблиц Cars, Body и Condition. Оно будет на листинге 3.2. Остальные представления будут аналогичны, только будут работать с другими таблицами.

|  |
| --- |
| CREATE OR REPLACE VIEW ALL\_INFO\_USER AS  SELECT USERS.USER\_ID, USER\_NAME, USER\_EMAIL, USER\_IMG, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, PHONE, USER\_PASSPORT, CREDIT\_CARD, ROLE.ROLE\_NAME FROM USERS  JOIN ROLE ON USERS.USER\_ROLE = ROLE.ROLE\_ID; |

Листинг 3.2 – Представление all\_info\_car

Представление pricing\_car было создано для того, чтобы получить информацию о ценах автомобилей, объединив данные из таблиц Сars и Pricing.

Представление all\_payments было создано для того, чтобы получить информацию о платежах, совершёнными пользователями, объединив две таблицы Payments и Users. Помимо это можно увидеть дату совершения платежа и количество платежей заданного пользователя.

Представление all\_booking было создано для того, чтобы получить информацию о бронировании автомобилей пользователями, объединив две таблицы Booking и Users. Помимо это можно увидеть дату начала и окончания бронирования.

### 3.1.2 Индексы базы данных

Индекс — объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Для того, чтобы быстро находить автомобили по их модели и пользователей по их уникальному значению, были созданы два индекса: IDX\_MODEL на таблице CARS по полю CAR\_MODEL и IDX\_USER\_ID на таблице USERS по полю USER\_ID. Сами индексы представлены на листинге 3.4.

|  |
| --- |
| CREATE INDEX IDX\_USER\_NAME ON USERS(USER\_NAME);  CREATE INDEX IDX\_MODEL ON CARS(CAR\_MODEL);  CREATE INDEX IDX\_USER\_ID ON USERS(USER\_ID);  CREATE INDEX IDX\_CONDITION\_ID ON CONDITION(CONDITION\_ID);  CREATE INDEX IDX\_PAYMENT\_USER\_ID ON PAYMENTS(USER\_ID);  CREATE INDEX IDX\_PRICING\_CAR\_ID ON PRICING(CAR\_ID);  CREATE INDEX IDX\_BOOKING\_USER\_ID ON BOOKING(USER\_ID);  CREATE INDEX IDX\_BOOKING\_CAR\_ID ON BOOKING(CAR\_ID); |

Листинг 3.4 – Индексы базы данных

Кроме индексов IDX\_MODEL и IDX\_USER\_ID, в базе данных также присутствуют индексы на полях USER\_NAME, CONDITION\_ID и CAR\_ID таблиц USERS, CONDITION и BOOKING соответственно.

Индекс IDX\_CONDITION на таблице CONDITION был создан для ускорения поиска состояния автомобиля по его ID.

Индекс IDX\_PAYMENT\_USER\_ID на таблице PAYMENTS может использоваться для быстрого поиска платежей по ID пользователя.

Индекс IDX\_PRICING\_CAR\_ID на таблице PRICING был создан для ускорения поиска цен автомобилей по их ID.

Индекс IDX\_BOOKING\_USER\_ID на таблице BOOKING был создан для ускорения поиска бронирования пользователей по их ID.

Индекс IDX\_BOOKING\_CAR\_ID на таблице BOOKING был создан для ускорения поиска бронирования автомобилей по их ID.

В целом, использование индексов позволяет существенно ускорить операции поиска, сортировки и фильтрации данных в базе данных, особенно в случае большого объема данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах, поэтому необходимо сбалансировать количество и тип индексов для оптимальной производительности базы данных.

### 3.1.3 Триггеры базы данных

Триггер базы данных — это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для обеспечения целостности данных и контроля доступа к данным, а также для автоматической обработки данных при выполнении определенных операций в таблице.

Первый триггер, созданный в таблице Users, проверяет, что пользователь является совершеннолетним, его возраст больше 17 лет. Если возраст нового пользователя меньше 18 лет, то триггер генерирует исключение с сообщением об ошибке "User must be at least 18 years old". При создании триггера он применяется автоматически ко всем новым данным. Однако, что исключить применение триггера можно выполнить команду переключения триггера на значение ''Disable'' Скрипт триггера будет представлен на листинге 3.5.

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_DATE\_OF\_BIRTH  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  BEGIN  IF MONTHS\_BETWEEN(SYSDATE, :NEW.USER\_DATE\_OF\_BIRTH) < 216 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'USER MUST BE AT LEAST 18 YEARS OLD');  END IF;  END;  ALTER TRIGGER CHECK\_DATE\_OF\_BIRTH DISABLE; |

Листинг 3.5 – Скрипт триггера check\_date\_of\_birth

Второй триггер проверяет, что длина пароля нового пользователя не менее 8 символов. Если длина пароля меньше 8 символов, то триггер генерирует исключение с сообщением об ошибке "Password must be at least 8 characters long". При создании триггера он применяется автоматически ко всем новым данным. Также за счёт того, что я использую технологию шифрования пароля, его нужно для начало дешифровать.

Скрипт триггера будет представлен на листинге 3.6.

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_PASSWORD\_LENGTH  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  DECLARE  DECRYPTED\_PASSWORD VARCHAR2(128);  BEGIN  DECRYPTED\_PASSWORD:= DECRYPT\_PASSWORD(:NEW.USER\_PASSWORD);  IF LENGTH(DECRYPTED\_PASSWORD) < 8 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'PASSWORD MUST CONTAIN AT LEAST 8 CHARACTERS');  END IF;  END; |

Листинг 3.6 – Скрипт триггера check\_password\_length

Третий триггер проверяет, поле user\_gmail на домен электронной почты. Если адрес электронной почты не содержит домен, то триггер генерирует исключение с сообщением об ошибке "Unvalid email domain". При создании триггера он применяется автоматически ко всем новым данным.Скрипт триггера будет представлен на листинге 3.7.

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_EMAIL\_DOMAIN  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  BEGIN  IF NOT REGEXP\_LIKE(:NEW.USER\_EMAIL, '(^| )[^@]+@(gmail\.com|yandex\.ru|mail\.ru)( |$)') THEN  RAISE\_APPLICATION\_ERROR(-20001, 'INVALID EMAIL DOMAIN');  END IF;  END; |

Листинг 3.7 – Скрипт триггера check\_email\_domain

Таким образом, все триггеры выполняют валидацию данных перед их вставкой или обновлением в таблице Users, обеспечивая целостность данных

## 3.2 Описание используемой технологии

В данной базе данных используется мультимедийность для хранения изображений пользователей и автомобилей. Например, в таблице Users в столбце user\_img хранится изображение профиля пользователя в формате BLOB. В таблице Cars в столбце car\_image также хранятся изображения в формате BLOB.Таким образом, использование мультимедийности в базе данных позволяет хранить различные мультимедийные данные, необходимые для работы каршеринга.

Для того, чтобы изображения отображались корректно нам нужно инициализировать две переменные, такие как l\_blob типа BLOB для хранения двоичных данных и l\_bfile типа BFILE, которая предоставляет ссылку на внешний двоичный файл на файловой системе. Помимо этого, мы создаем директорию, указывая нужный нам путь. Используя функцию Bfilename() находим наше изображение в заданной директории переводим формат изображения в бинарный вид С помощью встроенной процедуры ''Dbms\_lob.open'' открываем наш Bfile. Создаем временный Blob объект с помощью процедуры ''Dbms\_lob.createtemporary''. Производим загрузку данных из BFILE в BLOB c использованием процедуры ''Dbms\_lob.loadfromfile''. После закрываем наш бинарный файл. Процедура ''Dbms\_lob.getlength'' используется для получения длины бинарного файла. Cкрипт на Node.js для конвертации загруженного файла в бинарный формат представлен на листинге 3.8.

|  |
| --- |
| CREATE DIRECTORY MY\_IMAGE\_DIR AS'D:\OracleBD\oradata\CARSH\IMG\_DIR';  l\_blob BLOB;  l\_bfile BFILE := BFILENAME('MY\_IMAGE\_DIR', IN\_USER\_IMG);  DBMS\_LOB.OPEN(l\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(l\_blob, TRUE); DBMS\_LOB.LOADFROMFILE(l\_blob, l\_bfile, DBMS\_LOB.GETLENGTH(l\_bfile));  DBMS\_LOB.CLOSE(l\_bfile); |

Листинг 3.8 – Cкрипт конвертации файла в формат BLOB

Таким образом, была рассмотрена тема хранения мультимедийных данных в базах данных

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для каршеринга. Tакже была описана использованная технология мультимедийности для хранения музыкальных файлов и изображений в формате BLOB. Использование мультимедийности в базе данных позволяет хранить различные мультимедийные данные, необходимые для работы каршеринга.

4 Установка, настройка и использование Oracle DB 19c

## 4.1 Установка Oracle

Oracle DB (Oracle Database) - это реляционная база данных, разработанная и распространяемая компанией Oracle Corporation. Она является одной из наиболее популярных и широко используемых систем управления базами данных (СУБД) в мире. Предоставляет мощные возможности для организации, хранения, управления и извлечения данных. Она поддерживает стандартный язык запросов SQL (Structured Query Language) и предоставляет различные функциональные возможности, такие как транзакционная целостность, обработка многопользовательских запросов, поддержка распределенных систем и масштабируемость.

После установки Oracle DB была создана база данных с названием Carsh, которая будет использоваться в дальнейшем для хранения данных каршеринга.

## 4.2 Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличные пространства.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE TS\_USER  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_USER.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_CARS  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_CARS.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_BOOKING  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_BOOKING.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_PAYMENTS  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_PAYMENTS.dbf'  SIZE 100M  AUTOEXTEND ON; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим восемь основных таблиц: Role, Users, Cars, Body, Condition, Pricing, Payments, Booking.

Таблицы Role, Users будут храниться в табличном пространстве TS\_USER, таблицы Cars, Body и Condition - в табличном пространстве TS\_CARS, таблица Booking - в табличном пространстве TS\_BOOKING, а таблица Payments - в табличном пространстве TS\_PATMENTS.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у каждого пользователя может быть множество платежей, которые он оплатил. Для этого мы добавим в таблицу Payments внешний ключ (FOREIGN KEY) на таблицу Users, который будет указывать на идентификатор пользователя.

Другой тип связи - это связь "многие ко многим" (many-to-many), которая определяет отношение между множеством записей в одной таблице и множеством записей в другой таблице. Например, у каждого автомобиля может быть множество бронирований, и каждое бронирование может содержать множество пользователей. Для этого мы создадим таблицу Booking, которая будет содержать внешние ключи на таблицы Cars и Users. Скрипт создание таблицы User будет представлен на листинге 4.2.

|  |
| --- |
| CREATE TABLE USERS (  USER\_ID NUMBER(10) PRIMARY KEY,  USER\_NAME VARCHAR2(50) NOT NULL,  USER\_IMG BLOB,  USER\_EMAIL VARCHAR2(50) UNIQUE NOT NULL,  USER\_PASSWORD VARCHAR2(128) NOT NULL,  USER\_DATE\_OF\_BIRTH DATE,  USER\_ROLE NUMBER(10),  PHONE VARCHAR2(20) NOT NULL UNIQUE,  CREDIT\_CARD VARCHAR2(20) NOT NULL UNIQUE,  CONSTRAINT FK\_USER\_ROLE FOREIGN KEY (USER\_ROLE) REFERENCES ROLE(ROLE\_ID)  ) TABLESPACE TS\_USER; |

Листинг 4.2 – Cкрипт создание таблицы User

Таким образом, было описано создание табличных пространств для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы четыре табличных пространства: TS\_USER, TS\_CARS, TS\_BOOKING и TS\_PAYMENTS, в каждом из которых будут храниться соответствующие таблицы.

## Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Будут созданы три роли для разграничения доступа к базе данных: user\_role, manager\_role и developer\_role. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли user\_role и manager\_role являются основными, в то время как роль developer role будет иметь более широкий набор привилегий.

Роль developer\_role имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли developer \_role можно увидеть на листинге 4.3.

|  |
| --- |
| CREATE ROLE RL\_DEVELOPER  GRANT ALL PRIVILEGES TO RL\_DEVELOPER;  GRANT CONNECT TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON USERS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON ROLE TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON BOOKING TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON PAYMENTS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON CARS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON BODY TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON CONDITION TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON PRICING TO RL\_DEVELOPER; |

Листинг 4.3 – Привилегии, выданные роли rl\_developer

Роль rl\_manager имеет права на выполнение различных функций и процедур, связанных с управлением пользователями и автомобилями в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления автомобилей, а также обновления информации о пользователях. Выданные привилегии роли manager\_role можно увидеть на листинге 4.4.

|  |
| --- |
| GRANT CONNECT TO RL\_MANAGER;  GRANT CREATE SESSION TO RL\_MANAGER;  GRANT CREATE ANY TABLE TO RL\_MANAGER;  GRANT CREATE VIEW TO RL\_MANAGER;  GRANT CREATE TRIGGER TO RL\_MANAGER;  GRANT SELECT ANY TABLE TO RL\_MANAGER;  GRANT CREATE PROCEDURE TO RL\_MANAGER;  GRANT CREATE TABLESPACE TO RL\_MANAGER;  GRANT DROP ANY TABLE TO RL\_MANAGER;  GRANT DROP ANY VIEW TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON DEV.USERS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON BOOKING TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON PAYMENTS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE, DELETE ON DEV.CARS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE, DELETE ON DEV.PRICING TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON BODY TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON CONDITION TO RL\_MANAGER;  GRANT EXECUTE ON REGISTER TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERBYID TO RL\_MANAGER;  GRANT EXECUTE ON GETCARS TO RL\_MANAGER;  GRANT EXECUTE ON GETBOOKINGBYUSERID TO RL\_MANAGER;  GRANT EXECUTE ON SEARCH\_CARS\_BY\_PRICE TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERBYID TO RL\_MANAGER;  GRANT EXECUTE ON ADD\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERS TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_PAYMENT TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_BOOKING TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_USER TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_BODY\_DEFAULT TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_BOOKING TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_CONDITION TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_PAYMENTS TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_CAR TO PL\_MANAGER;  GRANT EXECUTE ON UPDATE\_USER TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_USER\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON DECTYPT\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON ENCRYPT\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON SORT\_CARS\_BY\_CONDITION TO RL\_MANAGER; |

Листинг 4.4 – Привилегии, выданные роли rl\_manager

Роль user может добавлять совершать платежи и бронирование автомобилей, поиск автомобилей по критериям, сортировка автомобилей. Кроме того, user имеет возможность изменять свой профиль и пароль. Пользователь может просматривать профили других пользователей, искать автомобили по названию или по цене и просматривать информацию об автомобиля, такую как бренд, модель, тип кузова, состояние и т.д. Выданные привилегии роли user\_role можно увидеть на листинге 4.5.

|  |
| --- |
| GRANT CONNECT TO RL\_USER;  GRANT CREATE SESSION TO RL\_USER;  GRANT EXECUTE ON REGISTER TO RL\_USER;  GRANT EXECUTE ON LOGIN TO RL\_USER;  GRANT EXECUTE ON UPDATE\_USER TO RL\_USER;  GRANT EXECUTE ON SORT\_CARS\_BY\_CONDITION TO RL\_USER;  GRANT EXECUTE ON UPDATE\_USER\_PASSWORD TO RL\_USER;  GRANT EXECUTE ON GETUSERBYID TO RL\_USER;  GRANT EXECUTE ON SEARCH\_CARS\_BY\_PRICE TO RL\_USER;  GRANT EXECUTE ON DELETE\_BOOKING TO RL\_USER;  GRANT EXECUTE ON INSERT\_BOOKING TO RL\_USER;  GRANT EXECUTE ON GETCARBYID TO RL\_USER;  GRANT EXECUTE ON GETCARNAMEBYPRICINGID TO RL\_USER;  GRANT EXECUTE ON GETCARS TO RL\_USER;  GRANT EXECUTE ON GETTOTALPRICEBYCARID TO RL\_USER;  GRANT SELECT, UPDATE ON USERS TO RL\_USER;  GRANT SELECT ON ROLE TO RL\_USER;  GRANT SELECT ON CARS TO RL\_USER;  GRANT SELECT ON PRICING TO RL\_USER;  GRANT SELECT ON BODY TO RL\_USER;  GRANT SELECT ON CONDITION TO RL\_USER;  GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKING TO RL\_USER;  GRANT SELECT, INSERT, UPDATE ON PAYMENTS TO RL\_USER; |

Листинг 4.5 – Привилегии, выданные роли user\_role

Таким образом, были созданы три роли для ограничения доступа к базе данных: rl\_user, rl\_manager и rl\_developer. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль rl\_developer имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль rl\_user имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию и совершать бронирование и платежи. Роль rl\_manager имеет средний уровень привилегий и может выполнять функции, связанные с управлением пользователями и автомобилями в базе данных.

## 4.4 Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Добавление данных в таблицы;
5. Удаление данных из таблиц;
6. Изменение данных в таблицах;
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

### 4.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: GetUsers, GetUserById, GetCars, GetTotalPriceByCarId, GetBookingByUserId, GetCarById, GetCarNameByPricingID, GetUserNameByPaymentDate. Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

GetUsers, GetCars - функции для выборки списка пользователей и автомобилей.

GetUserById, GetBookingByUserId, GetCarById- функции для получения информации об определенном пользователе, автомобиле или бронировании.

GetCarNameByPricingID. – функция для получения полного названия автомобиля по конкретному значению цены

GetTotalPriceByCarId – функция для получения полной стоимости аренды.

GetUserNameByPaymentDate – функция для получения имени пользователя по дате его платежа.

На листинге 4.6 будет функция GetCars, которая предназначена для выборки всех автомобилей из таблицы CARS.

|  |
| --- |
| create or replace NONEDITIONABLE FUNCTION GETCARS  RETURN SYS\_REFCURSOR  AS  car\_cursor SYS\_REFCURSOR;  BEGIN  OPEN car\_cursor FOR  SELECT C.CAR\_ID, C.CAR\_BRAND, C.CAR\_MODEL, C.BODY\_ID, C.CAR\_INFO,  C.TYPE\_FUEL, C.LICENSE\_PLATE, C.COLOR, C.CAR\_IMAGE, C.CAR\_CONDITION  FROM CARS C;  RETURN car\_cursor;  END; |

Листинг 4.6 – Функция GetCars

Все остальные функции и процедуры будут аналогичны, также предназначены для выборки данных из различных таблиц базы данных.

### 4.4.2 Выборка данных по поисковому запросу

Для поиска определенного автомобиля по цене, можно использовать функцию search\_cars\_by\_price. Данная функция принимает на вход поисковый запрос в виде значения максимальной и минимальной цены и возвращает таблицу с данными о найденных автомобилях, цена которая расположена в указанном диапазоне. Функция представлена на листинге 4.7.

|  |
| --- |
| CREATE OR REPLACE FUNCTION SEARCH\_CARS\_BY\_PRICE (  p\_min\_price NUMBER,  p\_max\_price NUMBER  ) RETURN SYS\_REFCURSOR AS  v\_result SYS\_REFCURSOR;  BEGIN  OPEN v\_result FOR  SELECT c.car\_id, c.car\_brand, c.car\_model, c.body\_id, c.car\_info, c.type\_fuel,  c.license\_plate, c.color, c.car\_image, c.car\_condition, p.price\_per\_min,  p.count\_min, p.total\_price  FROM cars c  JOIN pricing p ON c.car\_id = p.car\_id  WHERE p.price\_per\_min BETWEEN p\_min\_price AND p\_max\_price;  RETURN v\_result;  END; |

Листинг 4.7 – Функция search\_cars\_by\_price

Для поиска используется оператор BETWEEN, который позволяет выполнить поиск в указанном диапазоне. Поисковый запрос в виде цифр передается в функцию в качестве аргументов p\_min\_price. И p\_max\_price. Затем возвращается таблица с данными о найденных автомобилях.

### 4.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы BODY была разработана функция INSERT\_BODY, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.8.

|  |
| --- |
| create or replace PROCEDURE INSERT\_BODY AS  I PLS\_INTEGER := 5;  BEGIN  -- Вставляем новые записи  WHILE I <= 100005 LOOP  INSERT INTO BODY (body\_id, body\_name) VALUES (I, 'BODY ' || I);  I := I + 1;  END LOOP;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END; |

Листинг 4.8 – Функция заполнения таблицы GENRE

Процедура INSERT\_BODY была создана для заполнения таблицы BODY 100000 строками. В теле функции используется цикл WHILE, который проходит по значениям от 5 до 100005 и для каждого значения выполняет вставку новой строки в таблицу BODY с именем 'BODY' и порядковым номером из цикла. Процедура не не принимает аргументов. Для выполнения процедуры необходимо выполнить CALL INSERT\_BODY.

### 4.4.4 Добавление данных в таблицы

Были разработаны следующие процедуры и функции для добавления новых строк в основные таблицы базы данных: Register, Add\_car, Insert\_booking, Insert\_condition, Insert\_body\_default, Insert\_payments, Insert\_role, Total\_price. Ниже будут описание каждой функции или процедуры.

Были разработаны следующие функции и процедуры для работы с базой данных каршеринга:

Register - функция, которая добавляет нового пользователя в таблицу Users. Принимает значения для полей in\_user\_id, in\_user\_name, in\_user\_email, in\_user\_password, in\_user\_date\_of\_birth, in\_user\_role\_id и in\_user\_img, in\_user\_role, in\_phone, passport, in\_credit\_card.

Login - функция, которая проверяет правильность ввода пароля для пользователя по указанному email. Принимает значения in\_user\_email и in\_user\_password.

Add\_car – функция, которая добавляет новый автомобиль в таблицу Cars. Принимает значения для полей in\_car\_id, in\_car\_brand, in\_car\_model, in\_body\_id, in\_car\_info, in\_type\_fuel, in\_license\_plate, in\_color, in\_car\_img, in\_car\_condition.

Insert\_booking - процедура, которая добавляет запись о бронировании автомобиля пользователем в таблицу Booking. Принимает значения in\_booking\_id, in\_user\_id, in\_car\_id, in\_start\_time, in\_end\_time.

Insert\_condition - процедура, которая добавляет запись о состоянии автомобилей в таблицу Condition. Принимает значения p\_condition\_id, p\_percent\_condition и p\_recommends.

Insert\_payments - функция, которая создает новый платёж для определенного пользователя в таблице Paymentst. Принимает значения in\_payment\_id, in\_user\_id, in\_payment\_date.

Insert\_body\_default - процедура, которая добавляет основные типы кузова в таблицу Body. Принимает значения p\_body\_id и p\_body\_name.

Insert\_role - процедура, которая добавляет новую роль пользователя Role. Принимает значения p\_role\_id и p\_role\_name.

Total\_price – функция, которая считает полную стоимость аренды авто и вставляет запись цены конкретного автомобиля в таблицу Pricing.

На листинге 4.9 будет функция Add\_car, которая добавляет автомобиль в таблицу Cars.

|  |
| --- |
| create or replace FUNCTION ADD\_CAR(  IN\_CAR\_ID IN NUMBER,  IN\_CAR\_BRAND IN VARCHAR2,  IN\_CAR\_MODEL IN VARCHAR2,  IN\_BODY\_ID IN NUMBER,  IN\_CAR\_INFO IN VARCHAR2,  IN\_TYPE\_FUEL IN VARCHAR,  IN\_LICENSE\_PLATE IN VARCHAR2,  IN\_COLOR IN VARCHAR2,  IN\_CAR\_IMG IN VARCHAR2,  IN\_CAR\_CONDITION IN NUMBER,  IN\_CAR\_ADRESS IN VARCHAR2  )  RETURN NUMBER  IS  car\_img BLOB;  car\_bfile BFILE := BFILENAME('MY\_IMAGE\_DIR', IN\_CAR\_IMG);  CAR\_ID NUMBER;  PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN  DBMS\_LOB.OPEN(car\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(car\_img, TRUE);  DBMS\_LOB.LOADFROMFILE(car\_img,car\_bfile, DBMS\_LOB.GETLENGTH(car\_bfile));  DBMS\_LOB.CLOSE(car\_bfile);  INSERT INTO CARS(CAR\_ID,CAR\_BRAND,CAR\_MODEL, BODY\_ID, CAR\_INFO, TYPE\_FUEL, LICENSE\_PLATE, COLOR, CAR\_IMAGE,CAR\_CONDITION, Adress )  VALUES (IN\_CAR\_ID, IN\_CAR\_BRAND, IN\_CAR\_MODEL,IN\_BODY\_ID, TO\_CLOB(IN\_CAR\_INFO),IN\_TYPE\_FUEL, IN\_LICENSE\_PLATE, IN\_COLOR, car\_img, IN\_CAR\_CONDITION, IN\_CAR\_ADRESS)  RETURNING CARS.CAR\_ID INTO CAR\_ID;  COMMIT;  RETURN CAR\_ID;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |
|  |

Листинг 4.9 – Функция заполнения таблицы Register

Все остальные функции и процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.5 Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры: DELETE\_USER, DELETE\_BOOKING, DELETE\_CAR, DELETE\_PAYMENT, DELETE\_BODY. Ниже будут описание каждой функции или процедуры.

DELETE\_USER - удаляет пользователя из таблицы User, а также удаляет все записи из таблицы Booking и Payments, в которых присутствует идентификатор удаляемого пользователя.

DELETE\_BOOKING - удаляет бронирование из таблицы Booking.

DELETE\_CAR - удаляет автомобиль из таблицы Cars, а также удаляет все записи из таблицы Booking и Payments, в которых присутствует идентификатор удаляемого автомобиля.

DELETE\_PAYMENT - удаляет платёж из таблицы Payments.

DELETE\_BODY - удаляет запись о типе кузова из таблицы Body.

На листинге 4.10 будет процедура DELETE\_CAR.

|  |
| --- |
| create or replace PROCEDURE DELETE\_CAR(IN\_CAR\_ID NUMBER)  AS  BEGIN  DELETE FROM BOOKING WHERE CAR\_ID = IN\_CAR\_ID;  DELETE FROM PRICING WHERE CAR\_ID = IN\_CAR\_ID;  DELETE FROM CARS WHERE CAR\_ID = IN\_CAR\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('CAR deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating car: ' || SQLERRM);  END; |

Листинг 4.10 – Процедура для удаления car

Все остальные процедуры будут аналогичны, также предназначены для удаления соответствующих данных из основных таблиц базы данных

### 4.4.6 Изменение данных в таблицы

Для изменение данных в базе данных были созданы следующие процедуры: UPDATE\_USER, UPDATE\_USER\_PASSWORD, UPDATE\_CAR, UPDATE\_PRICING, UPDATE\_BOOKING. Ниже будут описание каждой функции или процедуры.

UPDATE\_USER - обновляет данные пользователя в таблице Users (имя пользователя, дата рождения, изображение профиля и т.д).

UPDATE\_USER\_PASSWORD - обновляет пароль пользователя в таблице Users.

UPDATE\_СAR - обновляет данные автомобиля в таблице Car (бренд автомобиля, изображение автомобиля, модель автомобиля и т.д.).

UPDATE\_PRICING - обновляет цены на автомобили в таблице Pricing.

UPDATE\_BOOKING - обновляет данные бронирования в таблице Booking.

На листинге 4.11 будет процедура UPDATE\_USER.

|  |
| --- |
| create or replace PROCEDURE UPDATE\_USER(  in\_user\_id IN NUMBER,  in\_user\_name IN VARCHAR2,  in\_user\_email IN VARCHAR2,  in\_user\_img IN VARCHAR2,  in\_user\_date\_of\_birth IN DATE,  in\_user\_phone IN VARCHAR2,  in\_user\_passport IN VARCHAR2,  in\_user\_credit\_card IN VARCHAR2  )  AS  l\_blob BLOB;  l\_bfile BFILE;  BEGIN  IF in\_user\_img is not null then  l\_bfile:= BFILENAME('MY\_IMAGE\_DIR', in\_user\_img);  DBMS\_LOB.OPEN(l\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(l\_blob, TRUE);  DBMS\_LOB.LOADFROMFILE(l\_blob, l\_bfile, DBMS\_LOB.GETLENGTH(l\_bfile));  DBMS\_LOB.CLOSE(l\_bfile);  UPDATE USERS  SET USER\_NAME = in\_user\_name,  USER\_EMAIL = in\_user\_email,  USER\_IMG = l\_blob,  USER\_DATE\_OF\_BIRTH = in\_user\_date\_of\_birth,  PHONE = in\_user\_phone,  USER\_PASSPORT = in\_user\_passport,  CREDIT\_CARD = in\_user\_credit\_card  WHERE USER\_ID = in\_user\_id;  else  UPDATE USERS  SET USER\_NAME = in\_user\_name,  USER\_EMAIL = in\_user\_email,  USER\_IMG = null,  USER\_DATE\_OF\_BIRTH = in\_user\_date\_of\_birth,  PHONE = in\_user\_phone,  USER\_PASSPORT = in\_user\_passport,  CREDIT\_CARD = in\_user\_credit\_card  WHERE USER\_ID = in\_user\_id;  END IF;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('User updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating user: ' || SQLERRM);  END; |

Листинг 4.11 – Процедура для обновления пользователя

Все остальные процедуры будут аналогичны, также предназначены для изменения соответствующих данных в основных таблицах базы данных

### 4.7 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Функция encrypt\_password позволяет зашифровать пароль с помощью ключа, что может повысить безопасность хранения паролей пользователей в базе данных. Используется шифрование DES, для того чтобы можно было выбрать любую размерность ключа. Функция представлена на листинге 4.12.

|  |
| --- |
| create or replace FUNCTION ENCRYPT\_PASSWORD(  PASSWORD IN VARCHAR2  ) RETURN VARCHAR2  IS  CIPHER\_TEXT RAW(32767);  KEY RAW(16);  BEGIN  KEY := '1616161616161616';  CIPHER\_TEXT := DBMS\_CRYPTO.ENCRYPT(  SRC => UTL\_I18N.STRING\_TO\_RAW(PASSWORD, 'AL32UTF8'),  TYP => DBMS\_CRYPTO.ENCRYPT\_DES + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  KEY => KEY  );  RETURN RAWTOHEX(CIPHER\_TEXT);  END; |

Листинг 4.12 – Функция encrypt\_password

Функция decrypt\_password позволяет расшифровать зашифрованный пароль, что может быть полезно для проверки правильности введенного пароля при аутентификации. Функция представлена на листинге 4.13.

|  |
| --- |
| create or replace FUNCTION DECRYPT\_PASSWORD(  CIPHER\_TEXT IN VARCHAR2  ) RETURN VARCHAR2  IS  PASSWORD VARCHAR2(32767);  KEY RAW(16);  BEGIN  KEY := '1616161616161616';  PASSWORD := UTL\_I18N.RAW\_TO\_CHAR(  DBMS\_CRYPTO.DECRYPT(  SRC => HEXTORAW(CIPHER\_TEXT),  TYP => DBMS\_CRYPTO.ENCRYPT\_DES + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  KEY => KEY  ),  'AL32UTF8'  );  RETURN PASSWORD;  END; |

Листинг 4.13 – Функция decrypt\_password

В целом, эти функции могут быть полезны для повышения безопасности хранения паролей пользователей.

## 4.5 Описание процедур экспорта и импорта

База данных имеет возможность экспортировать и импортировать данные для таблицы Users в формате XML. Это может быть полезно в случае необходимости переноса данных на другой сервер или резервного копирования данных.

Для экспорта данных в формате XML была создана процедура EXPORT\_USERS\_TO\_XML\_FILE, которая принимает имя файла. Процедура представлена на листинге 4.15.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE EXPORT\_USERS\_TO\_XML\_FILE(FILE\_PATH IN VARCHAR2) IS  XML\_DATA XMLTYPE;  XML\_FILE UTL\_FILE.FILE\_TYPE;  BEGIN  SELECT XMLELEMENT("USERS", XMLAGG(XMLELEMENT("USER",  XMLFOREST(user\_id, user\_name, user\_email, user\_password, user\_date\_of\_birth,phone, user\_passport, credit\_card ))))  INTO XML\_DATA  FROM USERS;  XML\_FILE := UTL\_FILE.FOPEN('IMPORT\_EXPORT', FILE\_PATH, 'W', 32767);  UTL\_FILE.PUT\_LINE(XML\_FILE, '<?xml version="1.0" encoding="UTF-8"?>');  UTL\_FILE.PUT\_LINE(XML\_FILE, XML\_DATA.GETCLOBVAL());  UTL\_FILE.FCLOSE(XML\_FILE);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('Данные пользователей не найдены.');  WHEN UTL\_FILE.INVALID\_PATH THEN  DBMS\_OUTPUT.PUT\_LINE('Неверный путь к файлу.');  WHEN UTL\_FILE.WRITE\_ERROR THEN  DBMS\_OUTPUT.PUT\_LINE('Ошибка записи в файл.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE DIRECTORY IMPORT\_EXPORT AS 'D:\OracleBD\oradata\CARSH';  CALL EXPORT\_USERS\_TO\_XML\_FILE('USERS.xml'); |

Листинг 4.15 – Функция EXPORT\_USERS\_TO\_XML\_FILE

Для импорта данных из файла в формате XML была создана функция IMPORT\_USERS\_FROM\_XML\_FILE которая принимает имя файла и импортирует данные из файла в таблицу Users. Функция представлена на листинге 4.16.

|  |
| --- |
| create or replace NONEDITIONABLE FUNCTION IMPORT\_USERS\_FROM\_XML\_FILE(FILE\_PATH IN VARCHAR2)  RETURN SYS\_REFCURSOR AS  XML\_DATA XMLTYPE;  USER\_NAME VARCHAR2(255);  USER\_EMAIL VARCHAR2(255);  USER\_PASSWORD VARCHAR2(255);  USER\_DATE\_OF\_BIRTH DATE;  USER\_PASSPORT VARCHAR2(255);  CREDIT\_CARD VARCHAR2(255);  CURSOR USER\_CURSOR IS  SELECT \*  FROM XMLTABLE('/USERS/USER'  PASSING XML\_DATA  COLUMNS USER\_NAME VARCHAR2(255) PATH 'USER\_NAME',  USER\_EMAIL VARCHAR2(255) PATH 'USER\_EMAIL',  USER\_PASSWORD VARCHAR2(255) PATH 'USER\_PASSWORD',  USER\_DATE\_OF\_BIRTH DATE PATH 'USER\_DATE\_OF\_BIRTH',  USER\_PASSPORT VARCHAR2(255) PATH 'USER\_PASSPORT',  CREDIT\_CARD VARCHAR2(255) PATH 'CREDIT\_CARD');  RESULT SYS\_REFCURSOR;  BEGIN  DELETE FROM TEMP\_USERS;  SELECT XMLTYPE(BFILENAME('IMPORT\_EXPORT', FILE\_PATH), NLS\_CHARSET\_ID('UTF8'))  INTO XML\_DATA  FROM DUAL;  OPEN USER\_CURSOR;  LOOP  FETCH USER\_CURSOR INTO USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD;  EXIT WHEN USER\_CURSOR%NOTFOUND;  INSERT INTO TEMP\_USERS (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD)  VALUES (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD);  END LOOP;  CLOSE USER\_CURSOR;  OPEN RESULT FOR  SELECT USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD  FROM TEMP\_USERS;  RETURN RESULT;  END; |

Листинг 4.16 – Функция IMPORT\_USERS\_TO\_XML\_FILE

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в формате XML, что делает управление базой данных более удобным и эффективным.

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования Oracle DB. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и пакетов процедур для выполнения различных операций с данными.

Были также представлены процедуры экспорта и импорта данных в формате Xml, а также были проведены тесты производительности базы данных. В результате тестирования было установлено, что база данных Oracle DB обладает высокой производительностью и способна быстро обрабатывать запросы на получение данных.

Итак, можно сделать вывод, что Oracle DB является быстрой и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка Oracle DB, а также оптимизация запросов, позволят обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

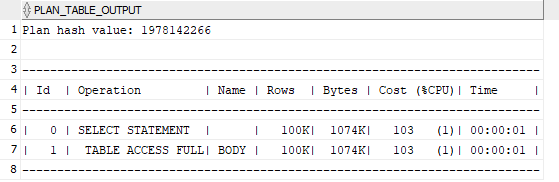
Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица Body, содержащая больше всего данных. Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| ANALYZE TABLE BODY COMPUTE STATISTICS;  EXEC DBMS\_STATS.GATHER\_TABLE\_STATS('DEV', 'BODY');  EXPLAIN PLAN FOR SELECT BODY\_NAME FROM BODY; |

Листинг 5.1 – Запрос к таблице genre

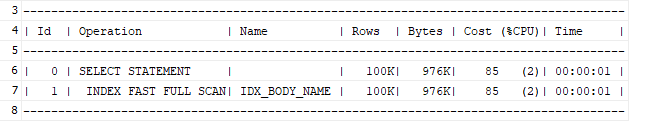
Запрос к таблице Body показывает, что его стоимость выполнения с Cost равным 103 говорит о том, что выполнение запроса требует значительных затрат времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составляет 1 с. Результаты запроса будут представлены на рисунке 5.2.



а б

Рисунок 5.2 – Результат выполнения запроса (а – стоимость выполнения запроса, б – время выполнения запроса)

Для ускорения данного процесса можно создать индекс на поле body\_name, так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3

а б

Рисунок 5.3 – Результат выполнения запроса (а – стоимость выполнения запроса, б – время выполнения запроса)

После создания индекса на поле body\_name запрос к таблице Body имеет стоимость выполнения с Cost равным 85, что является значительным улучшением по сравнению с предыдущим запросом. Результаты запроса также будут представлены на рисунке 5.3.

Тестирование показало, что создание индекса на поле body\_name значительно улучшило производительность запроса к таблице Body. Стоимость выполнения запроса уменьшилась с 103 до 85, что говорит о том, что сканирование таблицы и применение фильтра стали занимать меньше времени и ресурсов. Время выполнения запроса сократилось с 187 мс до 163 мс, что может показаться незначительным улучшением, но при работе с большими объемами данных каждый миллисекунд может иметь значение. Поэтому создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

## 5.2 Вывод

В данном разделе было рассмотрено важное понятие тестирования производительности базы данных. Для проведения тестирования была выбрана таблица Body с обширным объемом данных. Проведенный тест показал, что создание индекса на поле body\_name существенно улучшило производительность запроса к таблице Body, сократив время выполнения запроса и снизив стоимость его выполнения. Вывод заключается в том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных, что особенно важно при работе с большими объемами данных.

6 Руководство по использованию программного средства

## 6.1 Руководство пользователя

При открытии приложения каршеринга пользователю будет представлена главная страница, где можно ознакомиться с каталогом автомобилей. Для неавторизованных пользователей не доступен просмотр контента. Однако, все изменится после регистрации. Для того, чтобы зарегистрироваться в системе, пользователь может перейти на страницу регистрации. Он заполняет свои данные после чего запускается функция REGISTER. Если пользователь меньше 18 лет то сработает триггер CHECK\_DATE\_OF\_BIRTH. Если пользовать введет пароль меньше 8 символов то сработает триггер CHECK\_PASSWORD\_LENGTH. Если пользователь введет неверный домен электронной почты то сработает триггер CHECK\_EMAIL\_DOMAIN.

Если пользователь уже зарегистрирован в системе, ему нужно будет перейти на страницу входа в систему. На этой странице пользователь сможет ввести свой логин и пароль, после чего сработает функция Login. При успешной авторизации пользователь получает роль USER и пользователю будет доступен каталог автомобилей и сработает функция GETCARS. Пользователю доступен выбор и поиск автомобилей по критериям функции GETCARBYID, GETCARNAMEBYPRICING, GETTOTALBRICEBYCARID, SEARCH\_CARS\_BY\_PRICE, SORT\_CARS\_BY\_CONDITION.

Пользователь может зайти в личный кабинет и изменить свои личные данные. Например пользователь меняет дату рождения и срабатывает процедура UPDATE\_USER. Если пользователь решил поменять пароль от учётной записи срабатывает процедура UPDATE\_USER\_PASSWORD.

Когда пользователь определился с выбором автомобиля и нажимает кнопку ''Забронировать'' а после ''Оплатить'' то срабатывают две процедуры INSERT\_BOOKING и INSERT\_PAYMENTS. Также пользователь может отменить свое бронирование и тогда срабатывает процедура DELETE\_BOOKING.

Если пользователь зашёл под ролью manager, то ему предоставляется больше привилегий. При переходе на страницу ''Добавление авто'' и заполнении формы данных автомобиля срабатывает функции INSERT\_BODY, INSERT\_CONDITON, ADD\_CAR. Когда пользователь вводит цену автомобиля срабатывает функция TOTAL\_PRICE. Также при нажатии кнопки ''Редактировании авто'' менеджер заполняет данные которые нужно изменить и срабатывает процедура UPDATE\_CAR. Если пользователь изменяет стоимость автомобиля срабатывает процедура UPDATE\_PRICING. Если менеджер нажмёт кнопку ''Удаление авто'' срабатывает процедура DELETE\_CAR.

Когда пользователь с ролью User бронирует автомобиль то чтобы отследить бронирования пользователя менеджер заходит на страницу ''История бронирований'' и срабатывает функция GETBOOKINGBYUSERID. Тут менеджер может отследить всех пользователей, которые забронировали автомобили.

Когда пользователь с ролью User совершает платёж то, чтобы предоставить пользователю автомобиль менеджер заходит на страницу ''История платежей'' и срабатывает функция GETUSERNAMEBYPAYMENT\_DATE. Тут менеджер может отсортировать пользователей по дате платежа. Тогда срабатывает функция SORT\_PAYMENTS\_BY\_DATE.

Когда пользователь авторизуется под ролью developer ему доступен весь функционал ролей user и manager. Однако он имеет все привилегии сервиса. Он может отслеживать пароли пользователей для помощи пользователям в авторизации. Тогда срабатывают функции ENCRYPT\_PASSWORD и DECRYPT\_PASSWORD. Разработчик может добавлять новый функционал, редактировать и удалять существующий.

## 6.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, которая соединяет базу данных и приложение. Для этого необходимо запустить скрипт, который настроит соединение с базой данных и запустит сервер.

2. Запустить приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером. Для этого необходимо запустить команду для сборки и запуска приложения.

После выполнения этих шагов приложение будет полностью готово к работе и пользователь сможет начать использовать его функционал.

## 6.3 Вывод

В данном разделе были рассмотрены функциональные возможности приложения, а также права доступа для пользователей с различными ролями. Было показано, что наша платформа позволяет пользователям создавать автомобили, искать и сортировать их по цене, состоянию. Также изменять свой профиль и пароль.

Кроме того, менеджер имеет доступ к управлению пользователями, автомобилями, платежами и бронированиями пользователей. Для запуска приложения необходимо запустить серверную часть, которая соединяет базу данных и приложение.

Разработчик имеет доступ ко всему функционалу приложения.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление данными. В данной работе была поставлена задача разработки базы данных для каршеринга с использованием технологии применения мультимедийных типов данных в СУБД ORACLE.

В процессе выполнения работы были использованы различные объекты, включая таблицы, триггеры и функции, чтобы обеспечить структурированное хранение данных и своевременный доступ к ним. В результате, цель работы была успешно достигнута, и база данных готова к использованию.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формате XML, что обеспечило удобство использования и управления данными.

Одной из ключевых особенностей разработанной базы данных является технология хранения мультимедийных данных, что позволяет эффективно управлять медиа-файлами в проекте. Также использована технология шифрования паролей пользователей.

В целом, результаты работы говорят о том, что база данных полностью соответствует поставленным требованиям и может успешно использоваться в рамках каршеринга.

# Список литературных источников

1. Hello [Электронный ресурс] / Режим доступа: https://hello.by/ – Дата доступа: 01.05.2023.
2. Anytime [Электронный ресурс] / Режим доступа: https://any-time.by/– Дата доступа: 01.05.2023.
3. Coder Lessons «Хранимая процедура и функции» [Электронный ресурс] / Режим доступа: https://coderlessons.com/tutorials/bazy-dannykh/uchebnik-pl-sql/14-khranimaia-protsedura-i-funktsii– Дата доступа: 01.05.2023
4. Stackoverflow.com «Форум» [Электронный ресурс] / Режим доступа: https://stackoverflow.com/questions/tagged/oracle – Дата доступа: 01.05.2023.
5. Habr «10 приёмов работы с Oracle» [Электронный ресурс] / Режим доступа: https://habr.com/ru/companies/sberbank/articles/519358 – Дата доступа: 01.05.2023.
6. Oracle Patches «Автономные транзакции» [Электронный ресурс] / Режим доступа: https://oracle-patches.com/db/sql – Дата доступа: 01.05.2023.
7. Oracle «7 Модели форматов» [Электронный ресурс] / Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/21/sqlqr/Format-Models.html – Дата доступа: 01.05.2023.

# Приложение А

|  |
| --- |
| ------------------------TABLESPACE-----------------------------  CREATE TABLESPACE TS\_USER  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_USER.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_CARS  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_CARS.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_BOOKING  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_BOOKING.dbf'  SIZE 100M  AUTOEXTEND ON;  CREATE TABLESPACE TS\_PAYMENTS  DATAFILE 'D:\OracleBD\oradata\CARSH\TS\_PAYMENTS.dbf'  SIZE 100M  AUTOEXTEND ON;  -------------------- CREATE TABLES----------------------  ---------------------TABLE USERS -------------------------  CREATE TABLE USERS (  user\_id NUMBER(10) PRIMARY KEY,  user\_name VARCHAR2(50) NOT NULL,  user\_email VARCHAR2(50) NOT NULL UNIQUE,  user\_img BlOB,  user\_password VARCHAR2(128) NOT NULL,  user\_date\_of\_birth DATE,  user\_role NUMBER(10),  phone VARCHAR2(20) NOT NULL UNIQUE,  user\_passport VARCHAR2(20) NOT NULL UNIQUE,  credit\_card VARCHAR2(20) NOT NULL UNIQUE,  CONSTRAINT FK\_USER\_ROLE FOREIGN KEY (USER\_ROLE) REFERENCES ROLE (ROLE\_ID)  )TABLESPACE TS\_USER;  ---------------------TABLE ROLE --------------------------  CREATE TABLE ROLE(  role\_id NUMBER(10) PRIMARY KEY,  role\_name VARCHAR2(50) NOT NULL  )TABLESPACE TS\_USER;  ---------------------TABLE BOOKING -----------------------  CREATE TABLE BOOKING(  booking\_id NUMBER(10) PRIMARY KEY,  user\_id NUMBER(10),  car\_id NUMBER(10),  start\_time DATE NOT NULL,  end\_time DATE NOT NULL,  CONSTRAINT FK\_BOOKING\_USER FOREIGN KEY (USER\_ID) REFERENCES USERS (USER\_ID),  CONSTRAINT FK\_BOOKING\_CARS FOREIGN KEY (CAR\_ID) REFERENCES CARS(CAR\_ID),  CONSTRAINT CK\_BOOKING\_TIME CHECK (end\_time > start\_time)  )TABLESPACE TS\_BOOKING;  ---------------------TABLE PAYMENTS --------------------------------  CREATE TABLE PAYMENTS(  payment\_id NUMBER(10) PRIMARY KEY,  user\_id NUMBER(10),  amount NUMBER(10, 2) NOT NULL,  payment\_date DATE,  CONSTRAINT FK\_PAYMENTS\_USER FOREIGN KEY (USER\_ID) REFERENCES USERS(USER\_ID)  )TABLESPACE TS\_PAYMENTS;  ---------------------TABLE CARS ------------------------------------  CREATE TABLE CARS(  car\_id NUMBER(10) PRIMARY KEY,  car\_brand VARCHAR2(20) NOT NULL,  car\_model VARCHAR2(30) NOT NULL,  body\_id NUMBER(10) NOT NULL,  car\_info CLOB NOT NULL,  type\_fuel VARCHAR2(20) NOT NULL,  license\_plate VARCHAR2(20) NOT NULL UNIQUE,  color VARCHAR2(20) NOT NULL,  car\_image BLOB NOT NULL,  car\_condition NUMBER(10) NOT NULL,  address VARCHAR(50)  CONSTRAINT FK\_CARS\_BODY FOREIGN KEY (BODY\_ID) REFERENCES BODY (BODY\_ID),  CONSTRAINT FK\_CARS\_CONDITION FOREIGN KEY (CAR\_CONDITION) REFERENCES CONDITION(CONDITION\_ID)  )TABLESPACE TS\_CARS;  ---------------------TABLE BODY ------------------------------------  CREATE TABLE BODY(  body\_id NUMBER(10) PRIMARY KEY,  body\_name VARCHAR2(20) NOT NULL  )TABLESPACE TS\_CARS;  ---------------------TABLE CONDITION -------------------------------  CREATE TABLE CONDITION(  condition\_id NUMBER(10) PRIMARY KEY,  percent\_condition NUMBER(5,2) NOT NULL CHECK ((percent\_condition>=0) and (percent\_condition<=100)),  recomends VARCHAR2(50) NOT NULL  )TABLESPACE TS\_CARS;  DELETE FROM CONDITION WHERE CONDITION\_ID=4;  ---------------------TABLE PRICING ---------------------------------  CREATE TABLE PRICING(  pricing\_id NUMBER(10) PRIMARY KEY,  car\_id NUMBER(10) NOT NULL,  price\_per\_min NUMBER(10,2) NOT NULL,  count\_min NUMBER(10) NOT NULL,  total\_price NUMBER(10) NOT NULL,  CONSTRAINT FK\_PRICING\_CARS FOREIGN KEY(CAR\_ID) REFERENCES CARS(CAR\_ID)  )TABLESPACE TS\_CARS; |

Листинг А.1 – Скрипты создания таблиц

|  |
| --- |
| ---------------------CREATE ROLES ----------------------------------  CREATE ROLE RL\_DEVELOPER;  CREATE ROLE RL\_USER;  CREATE ROLE RL\_MANAGER;  SELECT \* FROM dba\_roles;  SELECT \*  FROM DBA\_TAB\_PRIVS  WHERE GRANTEE = 'RL\_DEVELOPER';  GRANT UNLIMITED TABLESPACE TO RL\_DEVELOPER;  SELECT \*  FROM DBA\_SYS\_PRIVS  WHERE GRANTEE = 'RL\_DEVELOPER';  ---------------------ROLE DEVELOPER --------------------------------  GRANT CONNECT TO RL\_DEVELOPER;  GRANT CREATE SESSION TO RL\_DEVELOPER;  GRANT CREATE ANY TABLE TO RL\_DEVELOPER;  GRANT CREATE VIEW TO RL\_DEVELOPER;  GRANT CREATE TRIGGER TO RL\_DEVELOPER;  GRANT CREATE PROCEDURE TO RL\_DEVELOPER;  GRANT CREATE TABLESPACE TO RL\_DEVELOPER;  GRANT DROP ANY TABLE TO RL\_DEVELOPER;  GRANT DROP ANY VIEW TO RL\_DEVELOPER;    GRANT ALL PRIVILEGES TO RL\_DEVELOPER;  GRANT EXECUTE ON UTL\_I18NTO TO RL\_DEVELOPER;  GRANT EXECUTE ON ENCRYPT\_PASSWORD TO RL\_DEVELOPER;  GRANT EXECUTE ON DECRYPT\_PASSWORD TO RL\_DEVELOPER;  GRANT EXECUTE ON DECRYPT\_PASSWORD TO RL\_DEVELOPER;  GRANT EXECUTE ON REGISTER TO RL\_DEVELOPER;  GRANT EXECUTE ON LOGIN TO RL\_DEVELOPER;  GRANT EXECUTE ON TOTAL\_PRICE TO RL\_DEVELOPER;  GRANT EXECUTE ON ADD\_CAR TO RL\_DEVELOPER;  GRANT EXECUTE ON INSERT\_BODY\_DEFAUL TO RL\_DEVELOPER;  GRANT EXECUTE ON INSERT\_CONDITION TO RL\_DEVELOPER;  GRANT EXECUTE ON INSERT\_BODY TO RL\_DEVELOPER;  GRANT EXECUTE ON INSERT\_PAYMENTS TO RL\_DEVELOPER;  GRANT EXECUTE ON INSERT\_BOOKING TO RL\_DEVELOPER;  GRANT EXECUTE ON UPDATE\_USER TO RL\_DEVELOPER;  GRANT EXECUTE ON UPDATE\_USER\_PASSWORD TO RL\_DEVELOPER;  GRANT EXECUTE ON UPDATE\_CAR TO RL\_DEVELOPER;  GRANT EXECUTE ON UPDATE\_PRICING TO RL\_DEVELOPER;  GRANT EXECUTE ON DELETE\_USER TO RL\_DEVELOPER;  GRANT EXECUTE ON DELETE\_CAR TO RL\_DEVELOPER;  GRANT EXECUTE ON DELETE\_PAYMENT TO RL\_DEVELOPER;  GRANT EXECUTE ON DELETE\_BOOKING TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON USERS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON ROLE TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON BOOKING TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON PAYMENTS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON CARS TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON BODY TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON CONDITION TO RL\_DEVELOPER;  GRANT ALL PRIVILEGES ON PRICING TO RL\_DEVELOPER;  ---------------------MANAGER\_ROLE ----------------------------------  GRANT CONNECT TO RL\_MANAGER;  GRANT CREATE SESSION TO RL\_MANAGER;  GRANT CREATE ANY TABLE TO RL\_MANAGER;  GRANT CREATE VIEW TO RL\_MANAGER;  GRANT CREATE TRIGGER TO RL\_MANAGER;  GRANT SELECT ANY TABLE TO RL\_MANAGER;  GRANT CREATE PROCEDURE TO RL\_MANAGER;  GRANT CREATE TABLESPACE TO RL\_MANAGER;  GRANT DROP ANY TABLE TO RL\_MANAGER;  GRANT DROP ANY VIEW TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON DEV.USERS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON BOOKING TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON PAYMENTS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE, DELETE ON DEV.CARS TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE, DELETE ON DEV.PRICING TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON BODY TO RL\_MANAGER;  GRANT SELECT, INSERT, UPDATE ON CONDITION TO RL\_MANAGER;  GRANT EXECUTE ON REGISTER TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERBYID TO RL\_MANAGER;  GRANT EXECUTE ON GETCARS TO RL\_MANAGER;  GRANT EXECUTE ON GETBOOKINGBYUSERID TO RL\_MANAGER;  GRANT EXECUTE ON SEARCH\_CARS\_BY\_PRICE TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERBYID TO RL\_MANAGER;  GRANT EXECUTE ON ADD\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON GETUSERS TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_CAR TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_PAYMENT TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_BOOKING TO RL\_MANAGER;  GRANT EXECUTE ON DELETE\_USER TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_BODY\_DEFAULT TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_BOOKING TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_CONDITION TO RL\_MANAGER;  GRANT EXECUTE ON INSERT\_PAYMENTS TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_CAR TO PL\_MANAGER;  GRANT EXECUTE ON UPDATE\_USER TO RL\_MANAGER;  GRANT EXECUTE ON UPDATE\_USER\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON DECTYPT\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON ENCRYPT\_PASSWORD TO RL\_MANAGER;  GRANT EXECUTE ON SORT\_CARS\_BY\_CONDITION TO RL\_MANAGER;  ---------------------USER\_ROLE -------------------------------------  GRANT CONNECT TO RL\_USER;  GRANT CREATE SESSION TO RL\_USER;  GRANT EXECUTE ON REGISTER TO RL\_USER;  GRANT EXECUTE ON LOGIN TO RL\_USER;  GRANT EXECUTE ON UPDATE\_USER TO RL\_USER;  GRANT EXECUTE ON SORT\_CARS\_BY\_CONDITION TO RL\_USER;  GRANT EXECUTE ON UPDATE\_USER\_PASSWORD TO RL\_USER;  GRANT EXECUTE ON GETUSERBYID TO RL\_USER;  GRANT EXECUTE ON SEARCH\_CARS\_BY\_PRICE TO RL\_USER;  GRANT EXECUTE ON DELETE\_BOOKING TO RL\_USER;  GRANT EXECUTE ON INSERT\_BOOKING TO RL\_USER;  GRANT EXECUTE ON GETCARBYID TO RL\_USER;  GRANT EXECUTE ON GETCARNAMEBYPRICINGID TO RL\_USER;  GRANT EXECUTE ON GETCARS TO RL\_USER;  GRANT EXECUTE ON GETTOTALPRICEBYCARID TO RL\_USER;  GRANT SELECT, UPDATE ON USERS TO RL\_USER;  GRANT SELECT ON ROLE TO RL\_USER;  GRANT SELECT ON CARS TO RL\_USER;  GRANT SELECT ON PRICING TO RL\_USER;  GRANT SELECT ON BODY TO RL\_USER;  GRANT SELECT ON CONDITION TO RL\_USER;  GRANT SELECT, INSERT, UPDATE, DELETE ON BOOKING TO RL\_USER;  GRANT SELECT, INSERT, UPDATE ON PAYMENTS TO RL\_USER;  --------------------------------------------------  ----------------CREATE USER -----------------------  ---------------------------------------------------  CREATE USER DEV IDENTIFIED BY p123;  GRANT RL\_DEVELOPER TO DEV;  CREATE USER USER\_1 IDENTIFIED BY u123;  GRANT RL\_USER TO USER\_1;  CREATE USER MANAGER\_1 IDENTIFIED BY m123;  GRANT RL\_MANAGER TO MANAGER\_1;  GRANT EXECUTE ON DBMS\_CRYPTO TO DEV;  GRANT UNLIMITED TABLESPACE TO DEV;  GRANT EXECUTE ON DBMS\_CRYPTO TO MANAGER\_1;  GRANT SELECT, INSERT, UPDATE, DELETE ON DEV.USERS TO MANAGER\_1;  GRANT SELECT ANY TABLE TO MANAGER\_1;  GRANT UNLIMITED TABLESPACE TO MANAGER\_1;  GRANT CONNECT, CREATE SESSION TO USER\_1; |

Листинг А.2 – Скрипты создание ролей и пользователей

|  |
| --- |
| -------------------------------FUNCTIONS----------------------------  ----------------------------ENCRYPT\_PASSWORD------------------------  CREATE OR REPLACE FUNCTION ENCRYPT\_PASSWORD(  PASSWORD IN VARCHAR2  ) RETURN VARCHAR2  IS  CIPHER\_TEXT RAW(32767);  KEY RAW(16);  BEGIN  KEY := '1616161616161616';  CIPHER\_TEXT := DBMS\_CRYPTO.ENCRYPT(  SRC => UTL\_I18N.STRING\_TO\_RAW(PASSWORD, 'AL32UTF8'),  TYP => DBMS\_CRYPTO.ENCRYPT\_DES + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  KEY => KEY  );  RETURN RAWTOHEX(CIPHER\_TEXT);  END;  SELECT ENCRYPT\_PASSWORD('Pasha111') FROM DUAL;  DROP FUNCTION ENCRYPT\_PASSWORD;  ----------------------------DECRYPT\_PASSWORD------------------------  CREATE OR REPLACE FUNCTION DECRYPT\_PASSWORD(  CIPHER\_TEXT IN VARCHAR2  ) RETURN VARCHAR2  IS  PASSWORD VARCHAR2(32767);  KEY RAW(16);  BEGIN  KEY := '1616161616161616';  PASSWORD := UTL\_I18N.RAW\_TO\_CHAR(  DBMS\_CRYPTO.DECRYPT(  SRC => HEXTORAW(CIPHER\_TEXT),  TYP => DBMS\_CRYPTO.ENCRYPT\_DES + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  KEY => KEY  ),  'AL32UTF8'  );  RETURN PASSWORD;  END;  SELECT DECRYPT\_PASSWORD('A40686440675EEAD') FROM DUAL;  DROP FUNCTION DECRYPT\_PASSWORD;  CREATE DIRECTORY MY\_IMAGE\_DIR AS 'D:\OracleBD\oradata\CARSH\IMG\_DIR';  DROP DIRECTORY MY\_IMAGE\_DIR;  ----------------------------REGISTER-------------------------  CREATE OR REPLACE FUNCTION REGISTER(  IN\_USER\_ID IN NUMBER,  IN\_USER\_NAME IN VARCHAR2,  IN\_USER\_EMAIL IN VARCHAR2,  IN\_USER\_IMG IN VARCHAR2,  IN\_USER\_PASSWORD IN VARCHAR2,  IN\_USER\_DATE\_OF\_BIRTH IN DATE,  IN\_USER\_ROLE\_ID IN NUMBER,  IN\_USER\_PHONE IN VARCHAR2,  IN\_USER\_PASSPORT IN VARCHAR2,  IN\_USER\_CREDIT\_CARD IN VARCHAR2  )  RETURN NUMBER  IS  l\_blob BLOB;  l\_bfile BFILE := BFILENAME('MY\_IMAGE\_DIR', IN\_USER\_IMG);  ENCRYPTED\_PASSWORD VARCHAR2(128) := ENCRYPT\_PASSWORD(IN\_USER\_PASSWORD);  USER\_ID NUMBER;  PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN  DBMS\_LOB.OPEN(l\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(l\_blob, TRUE);  DBMS\_LOB.LOADFROMFILE(l\_blob, l\_bfile, DBMS\_LOB.GETLENGTH(l\_bfile));  DBMS\_LOB.CLOSE(l\_bfile);  INSERT INTO DEV.USERS(USER\_ID,USER\_NAME, USER\_EMAIL, USER\_IMG, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_ROLE, PHONE, USER\_PASSPORT, CREDIT\_CARD )  VALUES (IN\_USER\_ID, IN\_USER\_NAME, IN\_USER\_EMAIL, l\_blob, ENCRYPTED\_PASSWORD, IN\_USER\_DATE\_OF\_BIRTH, IN\_USER\_ROLE\_ID, IN\_USER\_PHONE, IN\_USER\_PASSPORT, IN\_USER\_CREDIT\_CARD)  RETURNING DEV.USERS.USER\_ID INTO USER\_ID;  DBMS\_OUTPUT.PUT\_LINE('User registred successfully.');  COMMIT;  RETURN USER\_ID;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating user: ' || SQLERRM);  RAISE;    END;  SELECT DEV.REGISTER(10, 'Daria', 'Darilolila23456@gmail.com','10.jpg', 'Dari765Dar', '12-12-99', 1, '+3752945625233', 'KB2256744245', '2222-1770-4000-7655') FROM DUAL;  SELECT \* from DEV.USERS;  DELETE from DEV.USERS where USER\_ID=2;  DROP FUNCTION REGISTER;  ----------------------------LOGIN-------------------------  CREATE OR REPLACE FUNCTION LOGIN(IN\_USER\_EMAIL VARCHAR2, IN\_USER\_PASSWORD VARCHAR2)  RETURN NUMBER  IS  STORED\_PASSWORD VARCHAR2(32767);  BEGIN  SELECT USER\_PASSWORD INTO STORED\_PASSWORD FROM USERS WHERE USER\_EMAIL = IN\_USER\_EMAIL;  IF STORED\_PASSWORD IS NULL THEN  RETURN 0;  ELSIF STORED\_PASSWORD = ENCRYPT\_PASSWORD(IN\_USER\_PASSWORD) THEN  RETURN 1;  DBMS\_OUTPUT.PUT\_LINE('User login successfully.');  ELSE  RETURN 0;  DBMS\_OUTPUT.PUT\_LINE('Error user: ' || SQLERRM);  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN 0;  DBMS\_OUTPUT.PUT\_LINE('Error user: ' || SQLERRM);  END;  SELECT LOGIN('Pashok12.obodov@gmail.com', 'ADMIN111') FROM DUAL;  DROP FUNCTION DEV.LOGIN;  ----------------------------TOTAL\_PRICE-------------------------  CREATE OR REPLACE FUNCTION TOTAL\_PRICE(  in\_pricing\_id NUMBER,  in\_car\_id NUMBER,  in\_price\_per\_min NUMBER,  in\_count\_min NUMBER  )  RETURN NUMBER  IS  PRICE NUMBER;  PRICING\_ID NUMBER;  PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN  PRICE := in\_price\_per\_min \* in\_count\_min;  INSERT INTO PRICING(pricing\_id, car\_id, price\_per\_min, count\_min, total\_price)  VALUES (in\_pricing\_id, in\_car\_id, in\_price\_per\_min, in\_count\_min, PRICE)  RETURNING PRICING.pricing\_id INTO pricing\_id;  DBMS\_OUTPUT.PUT\_LINE('Insered table and calculate total\_price successfully.');  COMMIT;  RETURN pricing\_id;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error insered: ' || SQLERRM);  RAISE;    END;  SELECT TOTAL\_PRICE(10, 10, 11, 30) FROM DUAL;  SELECT \* FROM PRICING;  DELETE FROM DEV.PRICING WHERE PRICING\_ID=1;  DROP FUNCTION TOTAL\_PRICE;  CREATE OR REPLACE FUNCTION ADD\_CAR(  IN\_CAR\_ID IN NUMBER,  IN\_CAR\_BRAND IN VARCHAR2,  IN\_CAR\_MODEL IN VARCHAR2,  IN\_BODY\_ID IN NUMBER,  IN\_CAR\_INFO IN VARCHAR2,  IN\_TYPE\_FUEL IN VARCHAR,  IN\_LICENSE\_PLATE IN VARCHAR2,  IN\_COLOR IN VARCHAR2,  IN\_CAR\_IMG IN VARCHAR2,  IN\_CAR\_CONDITION IN NUMBER,  IN\_CAR\_Adress VARCHAR2  )  RETURN NUMBER  IS  car\_img BLOB;  car\_bfile BFILE := BFILENAME('MY\_IMAGE\_DIR', IN\_CAR\_IMG);  CAR\_ID NUMBER;  PRAGMA AUTONOMOUS\_TRANSACTION;  BEGIN  DBMS\_LOB.OPEN(car\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(car\_img, TRUE);  DBMS\_LOB.LOADFROMFILE(car\_img,car\_bfile, DBMS\_LOB.GETLENGTH(car\_bfile));  DBMS\_LOB.CLOSE(car\_bfile);  INSERT INTO CARS(CAR\_ID,CAR\_BRAND,CAR\_MODEL, BODY\_ID, CAR\_INFO, TYPE\_FUEL, LICENSE\_PLATE, COLOR, CAR\_IMAGE,CAR\_CONDITION, Adress )  VALUES (IN\_CAR\_ID, IN\_CAR\_BRAND, IN\_CAR\_MODEL,IN\_BODY\_ID, TO\_CLOB(IN\_CAR\_INFO),IN\_TYPE\_FUEL, IN\_LICENSE\_PLATE, IN\_COLOR, car\_img, IN\_CAR\_CONDITION, IN\_CAR\_Adress)  RETURNING CARS.CAR\_ID INTO CAR\_ID;  DBMS\_OUTPUT.PUT\_LINE('Car added successfully.');  COMMIT;  RETURN CAR\_ID;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error car: ' || SQLERRM);  RAISE;    END;    SELECT DEV.ADD\_CAR(1, 'BMW', 'M5', 2, 'BMW M5 - воплощение гармонии между элегантностью и мощью. Это автомобиль, созданный для тех, кто требует непревзойденного сочетания стиля, комфорта и невероятной производительности.', 'бензин', 'IA004-7', 'чёрный', 'car1.jpg', 4) FROM DUAL;  SELECT \* from DEV.CARS;  DELETE FROM CARS WHERE CAR\_ID=1;  DROP FUNCTION ADD\_CAR;  -----------------------------COUNT\_BOOKINGS-------------------------  CREATE OR REPLACE FUNCTION COUNT\_BOOKINGS(  in\_user\_id IN NUMBER  ) RETURN NUMBER  AS  v\_count NUMBER;  BEGIN  SELECT COUNT(\*) INTO v\_count  FROM BOOKING  WHERE user\_id = in\_user\_id;  RETURN v\_count;  END; |

Листинг А.3 – Скрипты создания дополнительных функций

|  |
| --- |
| -------------------------------CREATE-------------------------------  -------------------------------INSERT ROLE--------------------------  create or replace PROCEDURE INSERT\_ROLE(  p\_role\_id IN NUMBER,  p\_role\_name IN VARCHAR2  )  AS  BEGIN  INSERT INTO ROLE(role\_id, role\_name)  VALUES (p\_role\_id, p\_role\_name);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END;  CALL INSERT\_ROLE(1, 'admin');  -------------------------------INSERT\_BOOKING-----------------------  create or replace PROCEDURE INSERT\_BOOKING(  in\_booking\_id IN NUMBER,  in\_user\_id IN NUMBER,  in\_car\_id IN NUMBER,  in\_start\_time IN DATE,  in\_end\_time IN DATE  )  AS  v\_count\_user\_bookings NUMBER;  v\_count\_car\_bookings NUMBER;  BEGIN  -- Проверка, что пользователь не забронировал другую машину в тот же день  SELECT COUNT(\*) INTO v\_count\_user\_bookings  FROM BOOKING  WHERE user\_id = in\_user\_id  AND TRUNC(start\_time) = TRUNC(in\_start\_time);  IF v\_count\_user\_bookings > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'User already has a booking on the same day.');  END IF;  -- Проверка, что машина не забронирована другим пользователем в то же время  SELECT COUNT(\*) INTO v\_count\_car\_bookings  FROM BOOKING  WHERE car\_id = in\_car\_id  AND ( (in\_start\_time >= start\_time AND in\_start\_time < end\_time)  OR (in\_end\_time > start\_time AND in\_end\_time <= end\_time)  OR (in\_start\_time <= start\_time AND in\_end\_time >= end\_time));  IF v\_count\_car\_bookings > 0 THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Car is already booked during the specified time.');  END IF;  -- Вставка записи бронирования  INSERT INTO BOOKING(booking\_id, user\_id, car\_id, start\_time, end\_time)  VALUES (in\_booking\_id, in\_user\_id, in\_car\_id, in\_start\_time, in\_end\_time);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END;  CALL INSERT\_BOOKING(6, 2, 1, TO\_DATE('2023-10-05', 'YYYY-MM-DD'), TO\_DATE('2023-10-07', 'YYYY-MM-DD'));  SELECT \* FROM BOOKING;  -------------------------------INSERT\_CONDITION---------------------  create or replace PROCEDURE INSERT\_CONDITION(  p\_condition\_id IN NUMBER,  p\_percent\_condition IN NUMBER,  p\_recommends IN VARCHAR2  )  AS  BEGIN  INSERT INTO CONDITION(condition\_id, percent\_condition, recomends)  VALUES (p\_condition\_id, p\_percent\_condition,  CASE  WHEN p\_percent\_condition BETWEEN 0 AND 20 THEN 'Repair needed now'  WHEN p\_percent\_condition BETWEEN 21 AND 50 THEN 'Repair needed urgently'  WHEN p\_percent\_condition BETWEEN 51 AND 70 THEN 'Repair needed soon'  WHEN p\_percent\_condition BETWEEN 71 AND 100 THEN 'No repair needed'  ELSE 'Invalid condition percentage'  END  );  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END;  CALL INSERT\_CONDITION(1, 75, '');  --------------------------INSERT\_BODY\_DEFAUL-------------------  create or replace PROCEDURE INSERT\_BODY\_DEFAUL(  p\_body\_id IN NUMBER,  p\_body\_name IN VARCHAR2  )  AS  l\_count NUMBER;  BEGIN  -- Проверяем наличие существующей записи с заданным body\_id  SELECT COUNT(\*) INTO l\_count  FROM BODY  WHERE body\_id = p\_body\_id;  IF l\_count > 0 THEN  -- Если запись уже существует, генерируем ошибку  RAISE\_APPLICATION\_ERROR(-20001, 'Record with body\_id ' || p\_body\_id || ' already exists.');  ELSE  -- Вставляем новую запись  INSERT INTO BODY(body\_id, body\_name)  VALUES (p\_body\_id, p\_body\_name);  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END;  CALL INSERT\_BODY\_DEFAUL(1, 'Sedan');  ---------------------------INSERT\_PAYMENTS----------------------  CREATE OR REPLACE PROCEDURE INSERT\_PAYMENTS(  in\_payment\_id IN NUMBER,  in\_user\_id IN NUMBER,  in\_payment\_date IN DATE  )  AS  v\_total\_price NUMBER(10, 2);  v\_start\_time DATE;  v\_end\_time DATE;  v\_duration\_min NUMBER;  v\_calculated\_amount NUMBER(10, 2);  BEGIN  SELECT total\_price, start\_time, end\_time  INTO v\_total\_price, v\_start\_time, v\_end\_time  FROM PRICING  JOIN BOOKING ON PRICING.car\_id = BOOKING.car\_id  WHERE BOOKING.user\_id = in\_user\_id;  v\_duration\_min := ROUND((v\_end\_time - v\_start\_time) \* 24 \* 60);  v\_calculated\_amount := COUNT\_BOOKINGS(in\_user\_id);  IF v\_calculated\_amount > 0 AND in\_payment\_date > v\_start\_time AND in\_payment\_date < v\_end\_time THEN  INSERT INTO PAYMENTS(payment\_id, user\_id, amount, payment\_date)  VALUES (in\_payment\_id, in\_user\_id, v\_calculated\_amount, in\_payment\_date);  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20001, 'Invalid amount or payment date.');  END IF;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  CALL INSERT\_PAYMENTS(2, 2, TO\_DATE('2023-05-13 09:00:00', 'YYYY-MM-DD HH24:MI:SS'));  SELECT \* FROM PAYMENTS;  --------------------------DELETE\_USER-------------------------  create or replace PROCEDURE DELETE\_USER(IN\_USER\_ID NUMBER)  AS  BEGIN  DELETE FROM BOOKING WHERE USER\_ID = IN\_USER\_ID;  DELETE FROM PAYMENTS WHERE USER\_ID = IN\_USER\_ID;  DELETE FROM USERS WHERE USER\_ID = IN\_USER\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('USER deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleted user: ' || SQLERRM);  END;  -------------------------- DELETE\_PAYMENT----------------------  create or replace PROCEDURE DELETE\_PAYMENT(IN\_PAYMENT\_ID NUMBER)  AS  BEGIN  DELETE FROM PAYMENTS WHERE payment\_id = IN\_PAYMENT\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Payment deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleted payment: ' || SQLERRM);  END;  CAll DELETE\_PAYMENT(2);  -------------------------DELETE\_BOOKING----------------------------  create or replace PROCEDURE DELETE\_BOOKING(IN\_BOOKING\_ID NUMBER)  AS  BEGIN  DELETE FROM BOOKING WHERE booking\_id = IN\_BOOKING\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Booking deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleted booking: ' || SQLERRM);  END;  CALL DEV.DELETE\_BOOKING(1);  SELECT \* FROM DEV.CONDITION ORDER BY CONDITION\_ID;  --------------------------DELETE\_CAR----------------------------  create or replace PROCEDURE DELETE\_CAR(IN\_CAR\_ID NUMBER)  AS  BEGIN  DELETE FROM BOOKING WHERE CAR\_ID = IN\_CAR\_ID;  DELETE FROM PRICING WHERE CAR\_ID = IN\_CAR\_ID;  DELETE FROM CARS WHERE CAR\_ID = IN\_CAR\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('CAR deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleted car: ' || SQLERRM);  END;  CALL DELETE\_CAR(1);  --------------------------DELETE\_BODY----------------------------  create or replace PROCEDURE DELETE\_BODY(IN\_BODY\_ID NUMBER)  AS  BEGIN  DELETE FROM BODY WHERE BODY\_ID = IN\_BODY\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('BODY deleted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error deleted body: ' || SQLERRM);  END;  Call DELETE\_BODY(33);  -------------------------update\_booking----------------------------  create or replace PROCEDURE update\_booking(  p\_booking\_id IN NUMBER,  p\_start\_time IN DATE,  p\_end\_time IN DATE  )  IS  BEGIN  UPDATE BOOKING  SET start\_time = p\_start\_time,  end\_time = p\_end\_time  WHERE booking\_id = p\_booking\_id;  DBMS\_OUTPUT.PUT\_LINE('Booking update success.');  COMMIT; -- Optional, if you want to commit the changes immediately  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('Booking not found.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);  ROLLBACK; -- Optional, if you want to rollback the changes in case of an error  END;  call update\_booking(1, TO\_DATE('2023-10-05', 'YYYY-MM-DD'), TO\_DATE('2023-10-07', 'YYYY-MM-DD'));  -------------------------UPDATE\_CAR----------------------------  create or replace PROCEDURE UPDATE\_CAR(  in\_car\_id IN NUMBER,  in\_car\_brand IN VARCHAR2,  in\_car\_model IN VARCHAR2,  in\_body\_id IN NUMBER,  in\_car\_info IN VARCHAR2,  in\_type\_fuel IN VARCHAR2,  in\_license\_plate IN VARCHAR2,  in\_color IN VARCHAR2,  in\_car\_img IN VARCHAR2,  in\_car\_condition IN NUMBER,  in\_car\_adress VARCHAR,  )  AS  car\_img BLOB;  car\_bfile BFILE := BFILENAME('MY\_IMAGE\_DIR', in\_car\_img);  BEGIN  DBMS\_LOB.OPEN(car\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(car\_img, TRUE);  DBMS\_LOB.LOADFROMFILE(car\_img, car\_bfile, DBMS\_LOB.GETLENGTH(car\_bfile));  DBMS\_LOB.CLOSE(car\_bfile);  UPDATE CARS  SET  CAR\_BRAND = in\_car\_brand,  CAR\_MODEL = in\_car\_model,  BODY\_ID = in\_body\_id,  CAR\_INFO = TO\_CLOB(in\_car\_info),  TYPE\_FUEL = in\_type\_fuel,  LICENSE\_PLATE = in\_license\_plate,  COLOR = in\_color,  CAR\_IMAGE = car\_img,  CAR\_CONDITION = in\_car\_condition,  Adress = in\_car\_adress  WHERE  CAR\_ID = in\_car\_id;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Car updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating car: ' || SQLERRM);  END;  CALL UPDATE\_CAR(1, 'INFINITI', 'FX-35', 1, 'Отличный вариант', 'бензин', 'ER234-5', 'белый', 'car2.jpg', 3);  -------------------------UPDATE\_PRICING----------------------------  create or replace PROCEDURE UPDATE\_PRICING(  in\_pricing\_id IN NUMBER,  in\_price\_per\_min IN NUMBER,  in\_count\_min IN NUMBER  )  AS total NUMBER;  BEGIN  total :=in\_price\_per\_min \* in\_count\_min;  UPDATE PRICING SET  price\_per\_min = in\_price\_per\_min,  count\_min = in\_count\_min,  total\_price = total  Where pricing\_id = in\_pricing\_id;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Pricing updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating car: ' || SQLERRM);  END;  CALL UPDATE\_PRICING(1, 3, 30);  -------------------------UPDATE\_USER----------------------------  create or replace PROCEDURE UPDATE\_USER(  in\_user\_id IN NUMBER,  in\_user\_name IN VARCHAR2,  in\_user\_email IN VARCHAR2,  in\_user\_img IN VARCHAR2,  in\_user\_date\_of\_birth IN DATE,  in\_user\_phone IN VARCHAR2,  in\_user\_passport IN VARCHAR2,  in\_user\_credit\_card IN VARCHAR2  )  AS  l\_blob BLOB;  l\_bfile BFILE;  BEGIN  IF in\_user\_img is not null then  l\_bfile:= BFILENAME('MY\_IMAGE\_DIR', in\_user\_img);  DBMS\_LOB.OPEN(l\_bfile, DBMS\_LOB.LOB\_READONLY);  DBMS\_LOB.CREATETEMPORARY(l\_blob, TRUE);  DBMS\_LOB.LOADFROMFILE(l\_blob, l\_bfile, DBMS\_LOB.GETLENGTH(l\_bfile));  DBMS\_LOB.CLOSE(l\_bfile);  UPDATE USERS  SET USER\_NAME = in\_user\_name,  USER\_EMAIL = in\_user\_email,  USER\_IMG = l\_blob,  USER\_DATE\_OF\_BIRTH = in\_user\_date\_of\_birth,  PHONE = in\_user\_phone,  USER\_PASSPORT = in\_user\_passport,  CREDIT\_CARD = in\_user\_credit\_card  WHERE USER\_ID = in\_user\_id;  else  UPDATE USERS  SET USER\_NAME = in\_user\_name,  USER\_EMAIL = in\_user\_email,  USER\_IMG = null,  USER\_DATE\_OF\_BIRTH = in\_user\_date\_of\_birth,  PHONE = in\_user\_phone,  USER\_PASSPORT = in\_user\_passport,  CREDIT\_CARD = in\_user\_credit\_card  WHERE USER\_ID = in\_user\_id;  END IF;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('User updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating user: ' || SQLERRM);  END;  CALL UPDATE\_USER(1, 'Admin' , 'fhewhfew@gmail.com', '2.jpg', '20-09-99', '+375446532789', 'KB32552316', '4255-1903-5678-4587');  -------------------------UPDATE\_USER\_PASSWORD-----------------------  create or replace PROCEDURE UPDATE\_USER\_PASSWORD(  IN\_USER\_ID NUMBER,  IN\_USER\_PASSWORD VARCHAR2  )  AS  ENCRYPTED\_PASSWORD  VARCHAR2(32767) := ENCRYPT\_PASSWORD(IN\_USER\_PASSWORD);  BEGIN  UPDATE USERS  SET USER\_PASSWORD = ENCRYPTED\_PASSWORD  WHERE USER\_ID = IN\_USER\_ID;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('User password updated successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error updating user password: ' || SQLERRM);  END;  CALL UPDATE\_USER\_PASSWORD(2, 'efwhfewhhef'); |

Листинг А.4 – Скрипты создание функций CRUD

|  |
| --- |
| ------------------------------View----------------------------  ---------------------------ALL\_INFO\_USER------------------------  CREATE OR REPLACE VIEW ALL\_INFO\_USER AS  SELECT USERS.USER\_ID, USER\_NAME, USER\_EMAIL, USER\_IMG,  USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, PHONE, USER\_PASSPORT, CREDIT\_CARD, ROLE.ROLE\_NAME  FROM USERS  JOIN ROLE ON USERS.USER\_ROLE = ROLE.ROLE\_ID;  SELECT \* FROM DEV.ALL\_INFO\_USER;  DROP VIEW ALL\_INFO\_USER;  -------------------------------ALL\_INFO\_CAR-------------------------  CREATE OR REPLACE VIEW ALL\_INFO\_CAR AS  SELECT C.car\_id, C.car\_brand, C.car\_model, C.car\_info, C.type\_fuel, C.license\_plate, C.Adress  C.color , C.car\_image, B.body\_name, CO.percent\_condition, CO.recomends  FROM CARS C  JOIN BODY B ON C.body\_id = B. body\_id  JOIN CONDITION CO ON C.car\_condition=CO.condition\_id;  SELECT \* FROM ALL\_INFO\_CAR;  DROP VIEW ALL\_INFO\_CAR;  ------------------------------PRICING\_CAR---------------------------  CREATE OR REPLACE VIEW PRICING\_CAR AS  SELECT C.car\_brand, C.car\_model, C.car\_image, C.car\_info, C.license\_plate,  C.color,P.price\_per\_min, P.count\_min, P.total\_price  FROM PRICING P  JOIN CARS C ON P.car\_id = C.car\_id;  SELECT \* FROM PRICING\_CAR;  DROP VIEW PRICING\_CAR;  ------------------------------ALL\_PAYMENTS--------------------------  CREATE OR REPLACE VIEW ALL\_PAYMENTS AS  SELECT U.user\_name, U.user\_img, U.credit\_card, U.user\_email, P.amount, P.payment\_date  FROM PAYMENTS P  JOIN USERS U ON P.user\_id = U.user\_id;  SELECT \* FROM ALL\_PAYMENTS;  DROP VIEW ALL\_PAYMENTS;  ------------------------------ALL\_BOOKING---------------------------  CREATE OR REPLACE VIEW ALL\_BOOKING AS  SELECT U.user\_name, U.user\_img, U.credit\_card, U.user\_email,C.car\_brand, C.car\_model, C.car\_info, C.car\_image, B.start\_time, B.end\_time  FROM BOOKING B  JOIN USERS U ON B.user\_id = U.user\_id  JOIN CARS C ON B.car\_id=C.car\_id;  SELECT \* FROM ALL\_BOOKING;  DROP VIEW ALL\_BOOKING; |

Листинг А.5 – Скрипты создание представлений

|  |
| --- |
| -------------------------------TRIGGERS-----------------------------  -----------------------------CHECK\_DATE\_OF\_BIRTH--------------------  CREATE OR REPLACE TRIGGER CHECK\_DATE\_OF\_BIRTH  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  BEGIN  IF MONTHS\_BETWEEN(SYSDATE, :NEW.USER\_DATE\_OF\_BIRTH) < 216 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'USER MUST BE AT LEAST 18 YEARS OLD');  END IF;  END;  ALTER TRIGGER CHECK\_DATE\_OF\_BIRTH ENABLE;  ALTER TRIGGER CHECK\_DATE\_OF\_BIRTH DISABLE;  ----------------------------CHECK\_PASSWORD\_LENGTH-------------------  CREATE OR REPLACE TRIGGER CHECK\_PASSWORD\_LENGTH  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  DECLARE  decrypted\_password VARCHAR2(128);  BEGIN  decrypted\_password := DECRYPT\_PASSWORD(:NEW.USER\_PASSWORD);    IF LENGTH(decrypted\_password) < 8 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'PASSWORD MUST CONTAIN AT LEAST 8 CHARACTERS');  END IF;  END;  ALTER TRIGGER CHECK\_PASSWORD\_LENGTH ENABLE;  ALTER TRIGGER CHECK\_PASSWORD\_LENGTH DISABLE;  ---------------------------CHECK\_EMAIL\_DOMAIN-----------------------  CREATE OR REPLACE TRIGGER CHECK\_EMAIL\_DOMAIN  BEFORE INSERT OR UPDATE ON USERS  FOR EACH ROW  BEGIN  IF NOT REGEXP\_LIKE(:NEW.USER\_EMAIL, '(^| )[^@]+@(gmail\.com|yandex\.ru|mail\.ru)( |$)') THEN  RAISE\_APPLICATION\_ERROR(-20001, 'INVALID EMAIL DOMAIN');  END IF;  END;  ALTER TRIGGER CHECK\_EMAIL\_DOMAIN ENABLE;  ALTER TRIGGER CHECK\_EMAIL\_DOMAIN DISABLE; |

Листинг А.6 – Скрипты создания триггеров

|  |
| --- |
| -------------------------------XML----------------------------------  -----------------------------EXPORT---------------------------------  CREATE OR REPLACE PROCEDURE EXPORT\_USERS\_TO\_XML\_FILE(FILE\_PATH IN VARCHAR2) IS  XML\_DATA XMLTYPE;  XML\_FILE UTL\_FILE.FILE\_TYPE;  BEGIN  SELECT XMLELEMENT("USERS", XMLAGG(XMLELEMENT("USER",  XMLFOREST(user\_id, user\_name, user\_email, user\_password, user\_date\_of\_birth,phone, user\_passport, credit\_card ))))  INTO XML\_DATA  FROM USERS;  XML\_FILE := UTL\_FILE.FOPEN('IMPORT\_EXPORT', FILE\_PATH, 'W', 32767);  UTL\_FILE.PUT\_LINE(XML\_FILE, '<?xml version="1.0" encoding="UTF-8"?>');  UTL\_FILE.PUT\_LINE(XML\_FILE, XML\_DATA.GETCLOBVAL());  UTL\_FILE.FCLOSE(XML\_FILE);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('Данные пользователей не найдены.');  WHEN UTL\_FILE.INVALID\_PATH THEN  DBMS\_OUTPUT.PUT\_LINE('Неверный путь к файлу.');  WHEN UTL\_FILE.WRITE\_ERROR THEN  DBMS\_OUTPUT.PUT\_LINE('Ошибка записи в файл.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE DIRECTORY IMPORT\_EXPORT AS 'D:\OracleBD\oradata\CARSH';  CALL EXPORT\_USERS\_TO\_XML\_FILE('USERS.xml');  -----------------------------IMPORT---------------------------------  CREATE OR REPLACE FUNCTION IMPORT\_USERS\_FROM\_XML\_FILE(FILE\_PATH IN VARCHAR2)  RETURN SYS\_REFCURSOR AS  XML\_DATA XMLTYPE;  USER\_NAME VARCHAR2(255);  USER\_EMAIL VARCHAR2(255);  USER\_PASSWORD VARCHAR2(255);  USER\_DATE\_OF\_BIRTH DATE;  USER\_PASSPORT VARCHAR2(255);  CREDIT\_CARD VARCHAR2(255);  CURSOR USER\_CURSOR IS  SELECT \*  FROM XMLTABLE('/USERS/USER'  PASSING XML\_DATA  COLUMNS USER\_NAME VARCHAR2(255) PATH 'USER\_NAME',  USER\_EMAIL VARCHAR2(255) PATH 'USER\_EMAIL',  USER\_PASSWORD VARCHAR2(255) PATH 'USER\_PASSWORD',  USER\_DATE\_OF\_BIRTH DATE PATH 'USER\_DATE\_OF\_BIRTH',  USER\_PASSPORT VARCHAR2(255) PATH 'USER\_PASSPORT',  CREDIT\_CARD VARCHAR2(255) PATH 'CREDIT\_CARD');  RESULT SYS\_REFCURSOR;  BEGIN  DELETE FROM TEMP\_USERS;  SELECT XMLTYPE(BFILENAME('IMPORT\_EXPORT', FILE\_PATH), NLS\_CHARSET\_ID('UTF8'))  INTO XML\_DATA  FROM DUAL;  OPEN USER\_CURSOR;  LOOP  FETCH USER\_CURSOR INTO USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD;  EXIT WHEN USER\_CURSOR%NOTFOUND;  INSERT INTO TEMP\_USERS (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD)  VALUES (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD);  END LOOP;  CLOSE USER\_CURSOR;  OPEN RESULT FOR  SELECT USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD  FROM TEMP\_USERS;  RETURN RESULT;  END;  DECLARE  RESULT\_CURSOR SYS\_REFCURSOR;  USER\_NAME VARCHAR2(255);  USER\_EMAIL VARCHAR2(255);  USER\_PASSWORD VARCHAR2(255);  USER\_DATE\_OF\_BIRTH DATE;  USER\_PASSPORT VARCHAR2(255);  CREDIT\_CARD VARCHAR2(255);  BEGIN  RESULT\_CURSOR := IMPORT\_USERS\_FROM\_XML\_FILE('USERS.xml');  LOOP  FETCH RESULT\_CURSOR INTO USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_PASSPORT, CREDIT\_CARD;  EXIT WHEN RESULT\_CURSOR%NOTFOUND;  END LOOP;  CLOSE RESULT\_CURSOR;  END; |

Листинг А.7 – Скрипты для импорта и экспорта

|  |
| --- |
| ------------------------------------GET-----------------------------  ----------------------------------GETUSERS--------------------------  CREATE OR REPLACE FUNCTION GETUSERS  RETURN SYS\_REFCURSOR  AS  user\_cursor SYS\_REFCURSOR;  BEGIN  OPEN user\_cursor FOR  SELECT U.USER\_ID, U.USER\_NAME, U.USER\_IMG, U.USER\_EMAIL, U.USER\_PASSWORD,  U.USER\_DATE\_OF\_BIRTH, R.ROLE\_NAME  FROM USERS U  JOIN ROLE R ON U.USER\_ROLE = R.ROLE\_ID  ORDER BY U.USER\_ID;    RETURN user\_cursor;  END;  VAR user\_result REFCURSOR  EXECUTE :user\_result := DEV.GETUSERS;  PRINT user\_result;  ------------------------------GETUSERBYID-----------------------  CREATE OR REPLACE FUNCTION GETUSERBYID(USERID NUMBER)  RETURN SYS\_REFCURSOR  AS  user\_cursor SYS\_REFCURSOR;  BEGIN  OPEN user\_cursor FOR  SELECT U.USER\_ID, U.USER\_NAME, U.USER\_IMG, U.USER\_EMAIL, U.USER\_PASSWORD,  U.USER\_DATE\_OF\_BIRTH, R.ROLE\_NAME  FROM USERS U  JOIN ROLE R ON U.USER\_ROLE = R.ROLE\_ID  WHERE U.USER\_ID = USERID;    RETURN user\_cursor;  END;  VAR user\_result REFCURSOR;  EXECUTE :user\_result := DEV.GETUSERBYID(2);  PRINT user\_result;  -------------------------------GETCARS---------------------------  CREATE OR REPLACE FUNCTION GETCARS  RETURN SYS\_REFCURSOR  AS  car\_cursor SYS\_REFCURSOR;  BEGIN  OPEN car\_cursor FOR  SELECT C.CAR\_ID, C.CAR\_BRAND, C.CAR\_MODEL, C.BODY\_ID, C.CAR\_INFO, C.TYPE\_FUEL, C.LICENSE\_PLATE, C.COLOR, C.CAR\_IMAGE, C.CAR\_CONDITION, C.Adress  FROM CARS C;    RETURN car\_cursor;  END;  VAR car\_result REFCURSOR;  EXECUTE :car\_result := GETCARS;  PRINT car\_result;  ------------------------------GETCARBYID------------------------  CREATE OR REPLACE FUNCTION GETCARBYID(CARID NUMBER)  RETURN SYS\_REFCURSOR  AS  car\_cursor SYS\_REFCURSOR;  BEGIN  OPEN car\_cursor FOR  SELECT C.CAR\_ID, C.CAR\_BRAND, C.CAR\_MODEL, C.BODY\_ID, C.CAR\_INFO,C.TYPE\_FUEL, C.LICENSE\_PLATE, C.COLOR, C.CAR\_IMAGE, C.CAR\_CONDITION, C.Adress  FROM CARS C  WHERE C.CAR\_ID = CARID;    RETURN car\_cursor;  END;  VAR car\_result REFCURSOR;  EXECUTE :car\_result := GETCARBYID(1);  PRINT car\_result;  ---------------------------GETCARNAMEBYPRICINGID-------------  CREATE OR REPLACE FUNCTION GETCARNAMEBYPRICINGID(PRICEID NUMBER)  RETURN VARCHAR2  AS  car\_name VARCHAR2(50);  BEGIN  SELECT C.CAR\_BRAND || ' ' || C.CAR\_MODEL  INTO car\_name  FROM CARS C  JOIN PRICING P ON C.CAR\_ID = P.CAR\_ID  WHERE P.PRICING\_ID = PRICEID;    RETURN car\_name;  END;  DECLARE  car\_name VARCHAR2(50);  BEGIN  car\_name := GETCARNAMEBYPRICINGID(1);  DBMS\_OUTPUT.PUT\_LINE('Car Name: ' || car\_name);  END;  -------------------------GETTOTALPRICEBYCARID--------------  CREATE OR REPLACE FUNCTION GETTOTALPRICEBYCARID(CARID NUMBER)  RETURN NUMBER  AS  total\_price NUMBER(10,2);  BEGIN  SELECT SUM(P.TOTAL\_PRICE)  INTO total\_price  FROM PRICING P  WHERE P.CAR\_ID = CARID;    RETURN total\_price;  END;  DECLARE  car\_id NUMBER(10);  total\_price NUMBER(10,2);  BEGIN  car\_id := 1;  total\_price := GETTOTALPRICEBYCARID(car\_id);  DBMS\_OUTPUT.PUT\_LINE('Total Price for Car ID ' || car\_id || ': ' || total\_price);  END;  ----------------------GETCONDITIONBYLICENSEPLATE--------  CREATE OR REPLACE FUNCTION GETCONDITIONBYLICENSEPLATE(LICENSEPLATE VARCHAR2)  RETURN VARCHAR2  AS  car\_condition VARCHAR2(50);  BEGIN  SELECT C.RECOMENDS  INTO car\_condition  FROM CARS A  JOIN PRICING P ON A.CAR\_ID = P.CAR\_ID  JOIN CONDITION C ON P.CAR\_CONDITION = C.CONDITION\_ID  WHERE A.LICENSE\_PLATE = LICENSEPLATE;    RETURN car\_condition;  END;  --------------------------GETUSERNAMEBYPAYMENT\_DATE---------  CREATE OR REPLACE FUNCTION GETUSERNAMEBYPAYMENT\_DATE(PAYMENTDATE DATE)  RETURN VARCHAR2  AS  user\_name VARCHAR2(255);  BEGIN  SELECT U.USER\_NAME  INTO user\_name  FROM USERS U  JOIN PAYMENTS P ON U.USER\_ID = P.USER\_ID  WHERE P.PAYMENT\_DATE = PAYMENTDATE;    RETURN user\_name;  END;  DECLARE  payment\_date DATE;  user\_name VARCHAR2(255);  BEGIN  payment\_date := TO\_DATE('2023-05-20', 'YYYY-MM-DD');  user\_name := GETUSERNAMEBYPAYMENT\_DATE(payment\_date);  DBMS\_OUTPUT.PUT\_LINE('User Name for Payment Date ' || payment\_date || ': ' || user\_name);  END;  --------------------------GETBOOKINGBYUSERID----------------  CREATE OR REPLACE FUNCTION GETBOOKINGBYUSERID(USERID NUMBER)  RETURN SYS\_REFCURSOR  AS  cur SYS\_REFCURSOR;  BEGIN  OPEN cur FOR  SELECT B.BOOKING\_ID, B.START\_TIME, B.END\_TIME, C.CAR\_BRAND, C.CAR\_MODEL  FROM BOOKING B  JOIN CARS C ON B.CAR\_ID = C.CAR\_ID  WHERE B.USER\_ID = USERID;    RETURN cur;  END;  DECLARE  user\_id NUMBER(10);  booking\_cur SYS\_REFCURSOR;  booking\_id NUMBER(10);  start\_time DATE;  end\_time DATE;  car\_brand VARCHAR2(20);  car\_model VARCHAR2(30);  BEGIN  user\_id := 2;  booking\_cur := GETBOOKINGBYUSERID(user\_id);    LOOP  FETCH booking\_cur INTO booking\_id, start\_time, end\_time, car\_brand, car\_model;  EXIT WHEN booking\_cur%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE('Booking ID: ' || booking\_id || ', Start Time: ' || start\_time || ', End Time: ' || end\_time || ', Car: ' || car\_brand || ' ' || car\_model);  END LOOP;    CLOSE booking\_cur;  END; |

Листинг А.8 – Скрипты для получения данных

|  |
| --- |
| -----------------------------SEARCHES-------------------------  ---------------------------SEARCH\_CARS\_BY\_PRICE--------------  CREATE OR REPLACE FUNCTION SEARCH\_CARS\_BY\_PRICE (  p\_min\_price NUMBER,  p\_max\_price NUMBER  ) RETURN SYS\_REFCURSOR AS  v\_result SYS\_REFCURSOR;  BEGIN  OPEN v\_result FOR  SELECT c.car\_id, c.car\_brand, c.car\_model, c.body\_id, c.car\_info, c.type\_fuel, c.Adress  c.license\_plate, c.color, c.car\_image, c.car\_condition, p.price\_per\_min,  p.count\_min, p.total\_price  FROM cars c  JOIN pricing p ON c.car\_id = p.car\_id  WHERE p.price\_per\_min BETWEEN p\_min\_price AND p\_max\_price;  RETURN v\_result;  END;  DECLARE  v\_cursor SYS\_REFCURSOR;  v\_car\_id NUMBER(10);  v\_car\_brand VARCHAR2(20);  v\_car\_model VARCHAR2(30);  v\_body\_id NUMBER(10);  v\_car\_info CLOB;  v\_type\_fuel VARCHAR2(20);  v\_license\_plate VARCHAR2(20);  v\_color VARCHAR2(20);  v\_car\_image BLOB;  v\_car\_condition NUMBER(10);  v\_price\_per\_min NUMBER(10,2);  v\_count\_min NUMBER(10);  v\_total\_price NUMBER(10);  v\_adress VARCHAR2(50);  BEGIN  v\_cursor := SEARCH\_CARS\_BY\_PRICE(p\_min\_price => 1, p\_max\_price => 50);  LOOP  FETCH v\_cursor INTO v\_car\_id, v\_car\_brand, v\_car\_model, v\_body\_id, v\_car\_info, v\_type\_fuel, v\_adress  v\_license\_plate, v\_color, v\_car\_image, v\_car\_condition, v\_price\_per\_min,  v\_count\_min, v\_total\_price;  EXIT WHEN v\_cursor%NOTFOUND;  -- Process the retrieved values as needed  DBMS\_OUTPUT.PUT\_LINE('Car ID: ' || v\_car\_id);  DBMS\_OUTPUT.PUT\_LINE('Car Brand: ' || v\_car\_brand);  DBMS\_OUTPUT.PUT\_LINE('Car Model: ' || v\_car\_model);  DBMS\_OUTPUT.PUT\_LINE('Car Adress: ' || v\_adress);  -- Output other retrieved columns  END LOOP;  CLOSE v\_cursor;  END;  -------------------------GET\_SEARCH\_USER\_BY\_NAME-----------  CREATE OR REPLACE FUNCTION GET\_SEARCH\_USER\_BY\_NAME (  p\_user\_name VARCHAR2  ) RETURN SYS\_REFCURSOR AS  v\_result SYS\_REFCURSOR;  BEGIN  OPEN v\_result FOR  SELECT user\_id, user\_name, user\_email, user\_img, user\_password, user\_date\_of\_birth, user\_role, phone, user\_passport, credit\_card  FROM users  WHERE user\_name LIKE '%' || p\_user\_name || '%';  RETURN v\_result;  END;  DECLARE  v\_cursor SYS\_REFCURSOR;  v\_user\_id NUMBER(10);  v\_user\_name VARCHAR2(50);  v\_user\_email VARCHAR2(50);  v\_user\_img BLOB;  v\_user\_password VARCHAR2(128);  v\_user\_date\_of\_birth DATE;  v\_user\_role NUMBER(10);  v\_phone VARCHAR2(20);  v\_user\_passport VARCHAR2(20);  v\_credit\_card VARCHAR2(20);  BEGIN  v\_cursor := GET\_SEARCH\_USER\_BY\_NAME(p\_user\_name => 'Pavel');  LOOP  FETCH v\_cursor INTO v\_user\_id, v\_user\_name, v\_user\_email, v\_user\_img, v\_user\_password,  v\_user\_date\_of\_birth, v\_user\_role, v\_phone, v\_user\_passport, v\_credit\_card;  EXIT WHEN v\_cursor%NOTFOUND;  -- Process the retrieved values as needed  DBMS\_OUTPUT.PUT\_LINE('User ID: ' || v\_user\_id);  DBMS\_OUTPUT.PUT\_LINE('User Name: ' || v\_user\_name);  DBMS\_OUTPUT.PUT\_LINE('User Email: ' || v\_user\_email);  -- Output other retrieved columns  END LOOP;  CLOSE v\_cursor;  END; |

Листинг А.9– Скрипты для поиска данных

|  |
| --- |
| --------------------------INSERT\_BODY--------------------------  create or replace PROCEDURE INSERT\_BODY AS  I PLS\_INTEGER := 5;  BEGIN  -- Вставляем новые записи  WHILE I <= 100005 LOOP  INSERT INTO BODY (body\_id, body\_name) VALUES (I, 'BODY ' || I);  I := I + 1;  END LOOP;  COMMIT;  DBMS\_OUTPUT.PUT\_LINE('Data inserted successfully.');  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  DBMS\_OUTPUT.PUT\_LINE('Error inserting data: ' || SQLERRM);  END;  BEGIN  INSERT\_BODY;  END;  SELECT \* FROM BODY ORDER BY BODY\_ID;  DELETE FROM BODY WHERE BODY\_NAME LIKE '%BODY%';  EXPLAIN PLAN FOR  SELECT BODY\_NAME  FROM BODY;  SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);  SET TIMING ON  SELECT \* FROM Body ;  ANALYZE TABLE BODY COMPUTE STATISTICS;  EXEC DBMS\_STATS.GATHER\_TABLE\_STATS('DEV', 'BODY'); |

Листинг А.10– Скрипты заполнение таблиц