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

Факультет Информационных технологий

Кафедра Информационных систем и технологий

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема «Реализация базы данных пиццерии с использованием технологии системы e-mail уведомления о событиях в базе данных»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кунцов С. Ю.

подпись, дата

**Руководитель**

Доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Блинова Е. А.

должность, учен. степень, ученое звание подпись, дата

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

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

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ Блинова Е. А.

подпись дата инициалы и фамилия

Минск 2024

**Содержание**

[Введение 4](#_Toc167349081)

[1. Аналитический обзор литературы по теме проекта 6](#_Toc167349082)

[1.1 Сравнительный анализ теоретических и эвристических методов решения поставленных задач 6](#_Toc167349083)

[1.2 Аналитический обзор аналогов 6](#_Toc167349084)

[1.2.1 Аналог «Пицца лисицца» 6](#_Toc167349085)

[Как и у любого подобного сайта, у автомобильного салона есть свои недостатки. Один из таких недостатков связан с ошибками при просмотре каталога меню. Например, пользователи могут столкнуться с проблемами понимания что они должны выбрать. 8](#_Toc167349086)

[1.2.2 Аналог «Додо пицца» 9](#_Toc167349087)

[1.3 Изучение требований, определение вариантов использования 9](#_Toc167349088)

[1.3.1 Определение основных функциональных требований к базе данных 9](#_Toc167349089)

[1.3.2 Определение нефункциональных требований к базе данных 10](#_Toc167349090)

[1.3.3 Определение вариантов использования 10](#_Toc167349091)

[1.4 Вывод 11](#_Toc167349092)

[2. Разработка архитектуры проекта 12](#_Toc167349093)

[2.1 Обобщенная структура управления приложением 12](#_Toc167349094)

[2.3 Роли и пользователи 12](#_Toc167349095)

[2.3 Описание информационных объектов и ограничений целостности 13](#_Toc167349096)

[2.3.1 Таблица Customer 14](#_Toc167349097)

[2.3.2 Таблица Manager 14](#_Toc167349098)

[2.3.3 Таблица Seller 15](#_Toc167349099)

[2.3.4 Таблица Courier 15](#_Toc167349100)

[2.3.5 Таблица Order 16](#_Toc167349101)

[2.3.6 Таблица Order\_details 16](#_Toc167349102)

[2.3.7 Таблица Basket 16](#_Toc167349103)

[2.3.8 Таблица Product\_property 17](#_Toc167349104)

[2.3.9 Таблица REVIEW 17](#_Toc167349105)

[2.3.10 Таблица Product 18](#_Toc167349106)

[2.3.11 Таблица Product\_category 18](#_Toc167349107)

[2.4 Вывод 18](#_Toc167349108)

[3. Разработка объектов базы данных 19](#_Toc167349109)

[3.1 Создание необходимых объектов 19](#_Toc167349110)

[3.2 Функции 19](#_Toc167349111)

[3.3 Процедуры 21](#_Toc167349112)

[3.4 Триггеры 22](#_Toc167349113)

[3.5 Индексы 22](#_Toc167349114)

[3.6 Вывод 23](#_Toc167349115)

[4.Описание процедур экспорта и импорта. 24](#_Toc167349116)

[4.1 Процедура экспорта данных 24](#_Toc167349117)

[4.2 Процедура импорта данных 25](#_Toc167349118)

[4.3 Вывод 25](#_Toc167349119)

[5. Тестирование производительности базы данных. 26](#_Toc167349120)

[5.1 Заполнение таблицы 100000 строк 26](#_Toc167349121)

[5.2 Тестирование производительности по таблице REVIEWS. 26](#_Toc167349122)

[5.3 Вывод 30](#_Toc167349123)

[6. Описание технологии и ее применение в базе данных 31](#_Toc167349124)

[6.1 Технология «Разработка системы мониторинга за состоянием базы данных» 31](#_Toc167349125)

[6.2 Графическое средство для администрирования pgAdmin 4 31](#_Toc167349126)

[6.3 Представление pg\_stat\_statements 34](#_Toc167349127)

[6.4 Вывод 37](#_Toc167349128)

[7. Руководство пользователя 38](#_Toc167349129)

[7.1 Инструкция по эксплуатации для пользователя «User» 38](#_Toc167349130)

[7.2 Инструкция по эксплуатации для пользователя «Manager» 39](#_Toc167349131)

[7.3 Инструкция по эксплуатации для пользователя «Mechanic» 40](#_Toc167349132)

[7.4 Инструкция по эксплуатации для пользователя «Administrator» 41](#_Toc167349133)

[7.4 Вывод 42](#_Toc167349134)

[Заключение 43](#_Toc167349135)

[Список использованных литературных источников 44](#_Toc167349136)

[Приложение 45](#_Toc167349137)

[Приложение А 45](#_Toc167349138)

[Приложение Б 49](#_Toc167349139)

[Приложение В 59](#_Toc167349140)

[Приложение Г 84](#_Toc167349141)

Введение

В современном мире пиццерии становятся все более популярными и востребованными местами, где люди могут насладиться вкусной пиццей и провести время с семьей и друзьями. Если рассматривать эту отрасль с точки зрения предпринимателей и владельцев пиццерий, то возникает необходимость в эффективной системе управления и автоматизации процессов, связанных с заказами, доставкой и учетом продуктов.

Целью данного проекта было разработка реляционной базы данных для пиццерии. Эта база данных предназначалась для обеспечения пиццерии функциями управления заказами, инвентарем, доставкой и другими операциями.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных. В качестве СУБД для базы данных была выбрана Oracle, в связи с ее высокой производительностью и надежностью.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-сервера с базой данных.

Для обеспечения работы приложения использовались ASP.NET Core в связке с WPF.

Основные требования к приложению:

* Регистрация и авторизация пользователя.
* Составление заказа.
* Выбор услуги.
* Поиск по услугам и товарам.
* Просмотр каталога (меню).
* Просмотр корзины и статуса заказа.
* Просмотр истории заказов.
* Составление отзыва к заказу.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

1. Аналитический обзор литературы по теме проекта

1.1 Сравнительный анализ теоретических и эвристических методов решения поставленных задач

В решении данной задачи существует множество методов, которые можно классифицировать по нескольким параметрам:

1. Производительность: эффективность выполнения запросов и оптимальное использование ресурсов.

2. Надежность: способность системы работать без сбоев, а также ее возможность восстановления после сбоя.

3. Гибкость: способность системы адаптироваться к изменяющимся требованиям и удовлетворять различным потребностям пользователей.

4. Расширяемость: возможность системы масштабироваться и добавлять новые функциональности.

В литературе активно обсуждаются различные модели данных, включая реляционную, иерархическую и сетевую модели. В контексте реализации базы данных для салона автомобилей, реляционная модель является наиболее предпочтительной. Она позволяет эффективно описывать связи между различными элементами информационной системы, такими как автомобили, клиенты, заказы, запчасти и прочее.

В литературе также описываются различные методы обработки данных, включая SQL-запросы и NoSQL-базы данных. В контексте данной темы SQL-запросы являются предпочтительными, так как они хорошо подходят для организации транзакций, которые важны для системы управления заказами, клиентами и другими аспектами работы салона автомобилей.

1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к базе данных.

Пиццерии предлагают широкий ассортимент разнообразных продуктов. Клиенты могут выбрать пиццы, дессерты, и т.д, соответствующие их предпочтениям и вкусам.

1.2.1 Аналог «Пицца лисицца»

«Пицца лисицца» включает в себя ряд таблиц, которые хранят информацию о различных аспектах. Некоторые из них:

Таблица "Пиццы": содержит информацию о доступных пиццах, такую как название, состав, размеры, цена, описание и фотографии.

Таблица "Заказы": хранит информацию о заказах клиентов, включая дату и время заказа, детали пиццы, связанные с заказом, адрес доставки, цену, статус заказа и контактную информацию клиента.

Таблица "Клиенты": содержит информацию о клиентах, включая их персональные данные, адреса доставки, историю заказов и предпочтения (например, любимые вкусы пиццы).

Таблица "Отзывы": содержит информацию о отзывах, оставленных клиентами о своем опыте с "PizzaFox" и качестве пиццы.

Таблица "Курьеры": хранит информацию о курьерах, которые осуществляют доставку заказов, включая их персональные данные, контактную информацию и статус занятости.

В салонах автомобилей данные хранятся в различных таблицах в различных форматах, включая структурированные таблицы, текстовые файлы, аудио- и видеозаписи, фотографии и прочие форматы. Для доступа к таблицам используются специализированные программные интерфейсы и приложения, которые обеспечивают управление и обработку информации на различных этапах работы салона автомобилей.

В качестве аналога рассмотрим сайт «Пицца лисицца» [1], который представлен на рисунке 1.1.



Рисунок 1.1 – Сайт «Пицца лисицца»

Данный сайт позволяет из любой точки города заказать понравившиеся блюда, а также получить всю необходимую информацию о них. Пример представлен на рисунке 1.2.

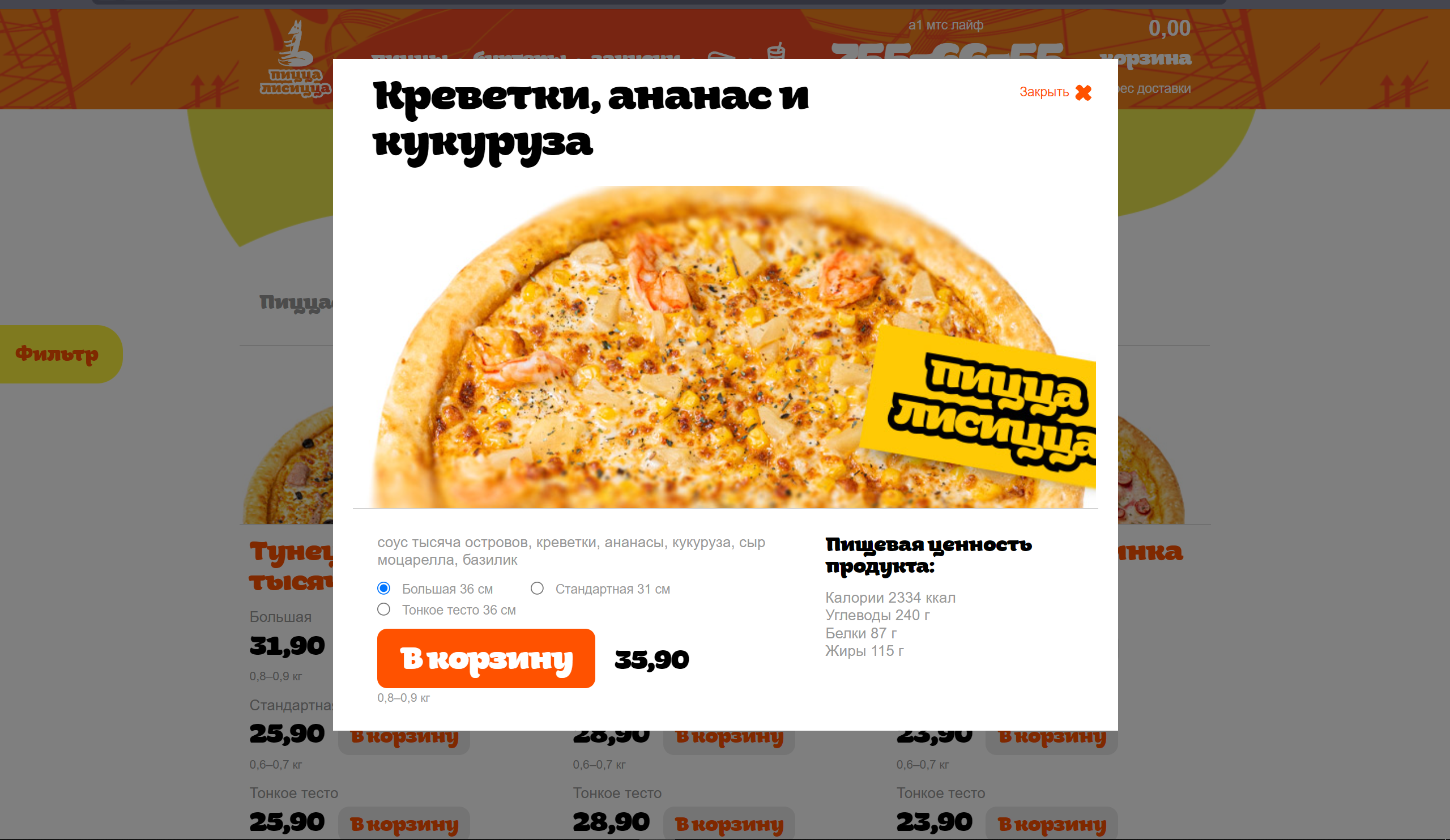


Рисунок 1.2 – Выбор пиццы на сайте «Пицца лисицца»

Также сайт позволяет увидеть доступные рестораны. Пример на рисунке 1.3.

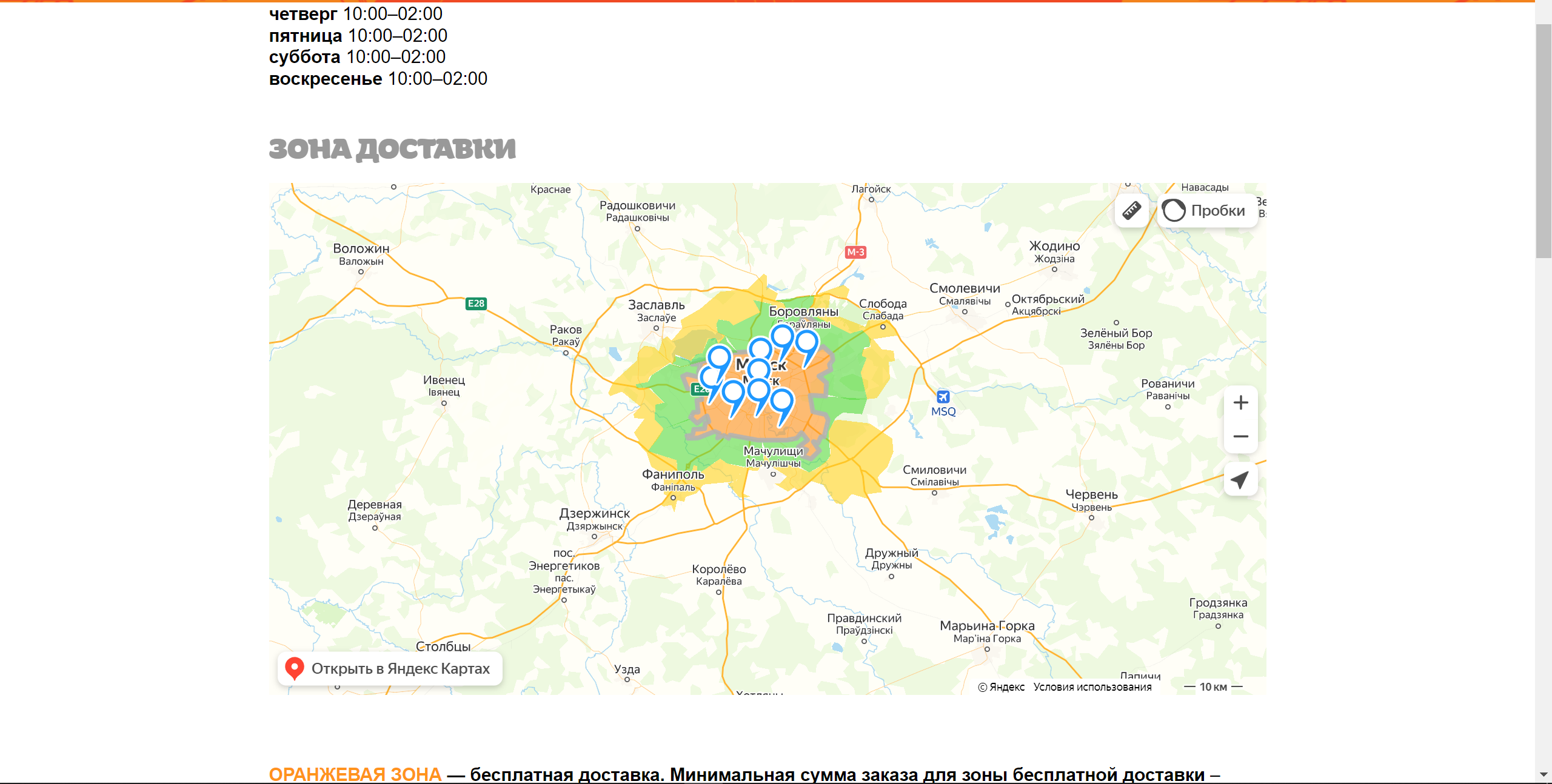


Рисунок 1.3 – Расположение ресторанов «Пицца лисицца»

Как и у любого подобного сайта, у автомобильного салона есть свои недостатки. Один из таких недостатков связан с ошибками при просмотре каталога меню. Например, пользователи могут столкнуться с проблемами понимания что они должны выбрать.

1.2.2 Аналог «Додо пицца»

Еще одним аналогом является сайт « Додо пицца » [2], предоставляющий услуги по покупке автомобилей на территории Москвы. Сайт продемонстрирован на рисунке 1.4.



Рисунок 1.4 – Выбор пиццы в «Додо пицца»

Плюсом данного аналога является предоставление обширной информации о продукте, включая ингредиенты и калории, а также простоту взаимодействия клиента с сайтом.

1.3 Изучение требований, определение вариантов использования

Одним из важных этапов в разработке проекта является анализ требований и определение потенциальных сценариев использования. Требования к проекту могут быть разделены на функциональные – связанные с основными функциями проекта, и нефункциональные – связанные с качественными характеристиками проекта.

1.3.1 Определение основных функциональных требований к базе данных

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к разработке базы данных в данном курсовом проекте. База данных должна обеспечивать возможность выполнения перечисленных ниже функций:

1. Хранение информации о продуктах, заказах, менеджерах, клиентах, продавцах, курьерах и отзывах.
2. Просмотр информации о всех доступных продуктах.
3. Сортировка продуктов по названию и цене.
4. Создание корзины на основе выбранных продуктов и их свойств.
5. Оформление заказов на основе корзины клиента.
6. Отслеживание статуса заказов и возможность его отменить.
7. Просмотр истории покупок.
8. Оставление отзывов о пиццерии.
9. Добавление и обновление информации о продуктах.
10. Удаление продуктов из базы данных.
11. Работа с заказами, изменение их статусов от имен продавцов и курьеров.
12. Просмотр доходности за месяц о продажах.
13. Добавления и удаление персонала салона автомобилей.
14. Управление авторизацией работников пиццерии.
15. Управление ценами на продукты.
16. Получение уведомлений об изменениях продуктов в базе данных.

1.3.2 Определение нефункциональных требований к базе данных

В качестве нефункциональных требований, можно выделить:

1. Производительность: База данных должна иметь высокую производительность и обеспечивать быстрый доступ к данным для обеспечения эффективной работы вокзала.
2. Надежность: База данных должна быть надежной и стабильной, обеспечивать целостность данных и иметь возможность быстрого восстановления после сбоев.
3. Масштабируемость: База данных должна быть масштабируемой и гибкой, чтобы обеспечивать эффективное управление растущим объемом данных вокзала.
4. Удобство использования: База данных должна быть удобной и простой в использовании для обеспечения эффективной работы пользователей.

1.3.3 Определение вариантов использования

Вариантами использования базы данных являются:

1. Просмотр информации о доступных автомобилях и запчастях.
2. Поиск автомобилей и запчастей по марке и модели.
3. Оформление заказов на автомобили и запчасти.
4. Отслеживание статуса заказов и даты заказа.
5. Просмотр истории покупок автомобилей и запчастей, а также просмотр истории обслуживания.
6. Оставление отзывов о компании.
7. Запись на техническое обслуживание автомобилей.
8. Добавление и обновление информации об автомобилях и запчастях.
9. Удаление автомобилей и запчастей из базы данных.
10. Работа с заявками на техническое обслуживание.
11. Просмотр отчетности об доступных автомобилях и запчастях.
12. Добавления и удаление персонала салона автомобилей.
13. Управление ценами автомобилей и запчастей.

1.4 Вывод

В ходе исследования были рассмотрены два аналога: сайты «Пицца лисиццы» и «Додо пицца». Эти ресурсы представляют собой примеры успешного применения баз данных для управления пиццерией и обеспечения удобного доступа к информации для клиентов.

2. Разработка архитектуры проекта

2.1 Обобщенная структура управления приложением

Пользовательский интерфейс взаимодействует с бизнес-логикой, которая, в свою очередь, взаимодействует со слоем доступа к данным. Слой доступа к данным осуществляет запросы к базе данных PostgreSQL и получает результаты, которые затем передаются обратно в бизнес-логику для обработки. База данных содержит данные, с которыми работает приложение.

Обобщенная структура управления приложением представлена на рисунке 2.1.

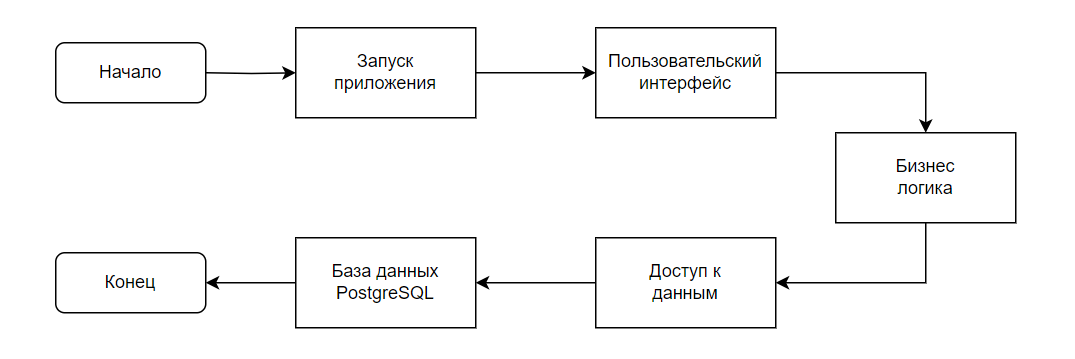


Рисунок 2.1 – Обобщенная структура управления приложением

Первым делом приложение должно установить соединение с БД, чтобы получить доступ к данным. Для этого в приложении на C# WPF используются паттерны UnitOfWork в связке с Repository и Singleton для хранения строк подлкючения к базе данных и манипуляций ими.

После установления соединения с БД, приложение может выполнять запросы к Postgresql. Для выполнения запросов используется объект библиотеки Npgsql, необходимый для работы PostgreSQL с C# WPF.

2.3 Роли и пользователи

Роль — это поименованный набор привилегий. Роли нужны для упрощения процесса управления доступом и безопасности в базе данных.

Пользователь — это учетная запись, которая позволяет конкретному человеку или приложению получать доступ к базе данных. Присвоение пользователю ролей и профилей безопасности позволяет определить его права доступа и ограничения при работе с базой данных.

При выполнении команды CREATE USER в PostgreSQL создается пользователь базы данных, а с помощью CREATE ROLE создается роль.

Пример создания роли и пользователя приведено на листинге 2.1.

|  |
| --- |
| create user manager with login; |

Листинг 2.1 – Создание роли «Менеджер»

Были разработаны следующие роли:

− Пользователь;

− Менеджер;

− Продавец;

− Курьер;

* Роль для авторизации;

− Программист.

Пример создания пользователя Менеджер с необходимыми разрешениями на выполнение процедур и функций представлен в листинге 2.2.

|  |
| --- |
| CREATE ROLE MANAGER\_ROLE;  CREATE USER MANAGER WITH LOGIN;  GRANT MANAGER\_ROLE TO MANAGER; GRANT USAGE ON SCHEMA procedures TO manager\_role;  GRANT EXECUTE ON PROCEDURE procedures.*add\_product*(varchar, text, bytea, double precision, boolean, varchar, data.product\_property[]) TO manager\_role; GRANT EXECUTE ON FUNCTION procedures.*add\_product*(varchar, text, bytea, double precision, boolean, varchar, OUT int, OUT varchar) TO manager\_role; GRANT EXECUTE ON FUNCTION procedures.*get\_user\_role*(uuid) TO manager\_role; GRANT EXECUTE ON FUNCTION procedures.*get\_products\_count*() TO manager\_role; GRANT EXECUTE ON FUNCTION procedures.*get\_paged\_data*(int, int) TO manager\_role; GRANT EXECUTE ON FUNCTION procedures.*add\_property\_for\_product*(product\_id\_ int, name varchar, value varchar) TO manager\_role; GRANT EXECUTE ON PROCEDURE procedures.*delete\_product*(int) TO manager\_role; |
|  |

Листинг 2.2 – Создание пользователя «Менеджер»

2.3 Описание информационных объектов и ограничений целостности

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

Структуру базы данных, ограничения целостности, связи и поля можно увидеть в приложении A и на рисунке 2.4.

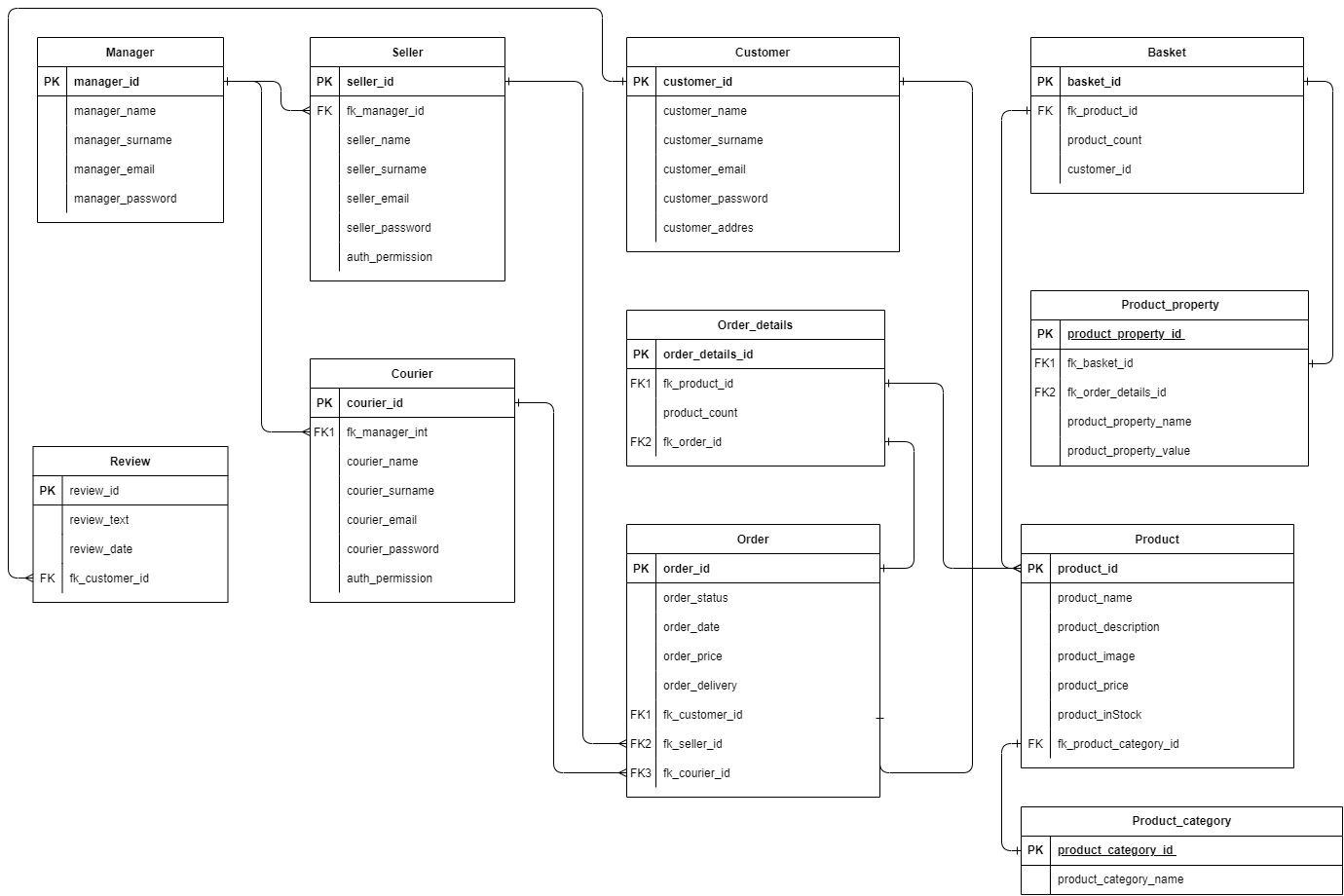


Рисунок 2.2 – Структура базы данных

2.3.1 Таблица Customer

Данная таблица используется для хранения информации о клиентах. В её состав входят следующие столбцы:

− customer\_id: хранит уникальный идентификатор каждого клиента. Тип данных – UUID.

− customer\_ surname: хранит фамилию клиента. Тип данных – VARCHAR.

− customer\_name: хранит имя клиента. Тип данных – VARCHAR.

− customer\_email: хранит почту клиента. Тип данных – VARCHAR.

− customer\_password: хранит пароль почты клиента. Тип данных – VARCHAR.

− customer\_address: хранит полный адрес клиента. Тип данных – VARCHAR.

Данная таблица содержит ограничение первичного ключа для столбца customer\_id, а также все столбцы должны содержать какое-либо значение, а также почта должна быть уникальными.

2.3.2 Таблица Manager

Данная таблица используется для хранения информации о менеджерах. В её состав входят следующие столбцы:

− manager\_id: хранит уникальный идентификатор каждого менеджера. Тип данных–UUID.

− manager\_surname: хранит фамилию менеджера. Тип данных – VARCHAR.

− manager\_name: хранит имя менеджера. Тип данных – VARCHAR.

− manager\_email: хранит почту менеджера. Тип данных – VARCHAR.

− manager\_password: хранит пароль почты менеджера. Тип данных – VARCHAR.

Данная таблица содержит ограничение первичного ключа для столбца manager\_id, а также все столбцы должны содержать какое-либо значение, а также почта должна быть уникальна.

2.3.3 Таблица Seller

Данная таблица используется для хранения информации о продавцах. В её состав входят следующие столбцы:

− seller\_id: хранит уникальный идентификатор каждого продавца. Тип данных – UUID.

− seller\_ surname: хранит фамилию продавца. Тип данных – VARCHAR.

− seller\_name: хранит имя продавца. Тип данных – VARCHAR.

− seller\_email: хранит почту продавца. Тип данных – VARCHAR.

− seller\_password: хранит пароль почты продавца. Тип данных – VARCHAR.

− auth\_permission: хранит статус возможности авторизации. Тип данных –BOOL.

* fk\_manager\_id: хранит уникальный идентификатор связки продавца и менеджера.

Данная таблица содержит ограничение первичного ключа для столбца seller\_id, и внешний ключ для столбца fk\_manager\_id, ссылающийся на соответствующий столбец в таблице MANAGERсоответственно, а также все столбцы должны содержать какое-либо значение, а также почта должна быть уникальна.

2.3.4 Таблица Courier

Данная таблица используется для хранения информации о курьерах. В её состав входят следующие столбцы:

− сourier\_id: хранит уникальный идентификатор каждого курьера. Тип данных – UUID.

− сourier\_ surname: хранит фамилию курьера. Тип данных – VARCHAR.

− сourier\_name: хранит имя курьера. Тип данных – VARCHAR.

− сourier\_email: хранит почту курьера. Тип данных – VARCHAR.

− сourier\_password: хранит пароль почты курьера. Тип данных – VARCHAR.

− auth\_permission: хранит статус возможности авторизации. Тип данных –BOOL.

fk\_manager\_id: хранит уникальный идентификатор связки продавца и курьера. Тип данных –UUID.

Данная таблица содержит ограничение первичного ключа для столбца сourier\_id, и внешний ключ для столбца fk\_manager\_id, ссылающийся на соответствующий столбец в таблице MANAGERсоответственно, а также все столбцы должны содержать какое-либо значение, а также почта должна быть уникальна.

2.3.5 Таблица Order

Данная таблица используется для хранения информации о заказах. В её состав входят следующие столбцы:

− order\_id: хранит уникальный идентификатор каждого заказа. Тип данных −UUID.

− order\_status: хранит статус заказа. Тип данных − VARCHAR.

− order\_date: : хранит дату заказа. Тип данных - DATE.

− order\_price: хранит цену заказа. Тип данных – DOUBLE PRECISION.

− order\_delivery: хранит статус доставки заказа. Тип данных − BOOLEN.

− fk\_customer\_id : хранит уникальный идентификатор связки заказа и клиента. Тип данных –UUID.

− fk\_seller\_id : хранит уникальный идентификатор связки заказа и продавца. Тип данных –UUID.

−fk\_courier\_id  хранит уникальный идентификатор связки заказа и курьера. Тип данных –UUID.

Таблица содержит ограничения первичного ключа для столбца order\_id и внешние ключи для столбцов fk\_customer\_id, fk\_seller\_id  и fk\_courier\_id, ссылающиеся на соответствующие столбцы в таблицах CUSTOMER, SELLER и COURIER соответственно, а также все столбцы должны содержать какое-либо значение.

2.3.6 Таблица Order\_details

Данная таблица используется для хранения деталей заказа о продукте в заказе. В её состав входят следующие столбцы:

− order\_details\_id: хранит уникальный идентификатор каждой детали заказа. Тип данных - SERIAL.

− fk\_product\_id : хранит уникальный идентификатор связки деталей заказа и продукта. Тип данных - SERIAL.

− product\_count : хранит количество продукта в заказе. Тип данных - INT.

− fk\_order\_id : хранит уникальный идентификатор связки деталей заказа и заказа. Тип данных –UUID.

Таблица содержит ограничения первичного ключа для столбца order\_details\_id и внешние ключи для столбцов fk\_product\_id , fk\_order\_id ссылающиеся на соответствующие столбцы в таблицах PRODUCT, ORDER соответственно, а также все столбцы должны содержать какое-либо значение.

2.3.7 Таблица Basket

Данная таблица используется для хранения информации о корзинах пользователей. В её состав входят следующие столбцы:

− **basket\_id**: хранит уникальный идентификатор каждого заказа запчастей. Тип данных − SERIAL.

− fk\_product\_id : хранит уникальный идентификатор продукта из корзины. Тип данных − SERIAL.

− customer\_id : хранит уникальный идентификатор клиента. Тип данных − UUID.

− product\_count : хранит количество заказанных продуктов. Тип данных − INT.

Таблица содержит ограничения первичного ключа для столбца **basket\_id** и внешние ключ для столбца fk\_product\_id , ссылающийся на соответствующий столбкеу в таблицеPRODUCT, а также все столбцы должны содержать какое-либо значение.

2.3.8 Таблица Product\_property

Данная таблица используется для хранения информации о свойствах заказаоного продукта. В её состав входят следующие столбцы:

− product\_property\_id: хранит уникальный идентификатор каждого листа обслуживания. Тип данных − SERIAL.

− fk\_basket\_id : хранит уникальный идентификатор корзины. Тип данных − SERIAL.

− fk\_order\_details\_id : хранит уникальный идентификатор деталей заказа. Тип данных − SERIAL.

− product\_property\_name : хранит название свойства. Тип данных − VARCHAR.

− product\_property\_value : хранит значение свойства. Тип данных − VARCHAR.

Таблица содержит ограничения первичного ключа для столбца product\_property\_id и внешние ключи для столбцов fk\_basket\_id , fk\_order\_details\_id , ссылающиеся на соответствующие столбцы в таблицах BASKET, ORDER\_DETAILS соответственно, а также все столбцы, должны содержать какое-либо значение.

2.3.9 Таблица REVIEW

Данная таблица используется для хранения отзывов клиентов. В её состав входят следующие столбцы:

− review\_id : хранит уникальный идентификатор каждого отзыва. Тип данных − SERIAL.

− fk\_customer\_id : хранит уникальный идентификатор клиента, оставившего отзыв. Тип данных −UUID.

− review\_text : хранит текст отзыва. Тип данных − VARCHAR.

− review\_date: хранит дату отзыва. Тип данных − DATE.

Таблица содержит ограничения первичного ключа для столбца review\_id  и внешний ключ для столбца fk\_customer\_id , ссылающийся на customer\_id  в таблице CUSTOMER, а также все столбцы должны содержать какое-либо значение.

2.3.10 Таблица Product

Данная таблица используется для хранения информации о продктах. В её состав входят следующие столбцы:

− product\_id: хранит уникальный кажлого продукта. Тип данных − SERIAL.

− product\_name : хранит название продукта. Тип данных − VARCHAR.

− product\_description : хранит описание продукта. Тип данных − VARCHAR.

− product\_image: хранит изображение продукта. Тип данных − BYTEA.

− product\_price : хранит цену продукта. Тип данных – DOUBLE PRECISION.

− product\_inStock: хранит статус наличия продукта. Тип данных − BOOLEN.

− fk\_product\_category\_id: хранит уникальный идентификатор категории продукта. Тип данных −SERIAL.

Таблица содержит ограничения первичного ключа для столбца product\_id  и внешний ключ для столбца fk\_product\_category\_id, ссылающийся на product\_category\_id в таблице Product\_category, а также все столбцы должны содержать какое-либо значение.

2.3.11 Таблица Product\_category

Данная таблица используется для хранения информации о категориях продукта. В её состав входят следующие столбцы:

− product\_category\_id: хранит уникальный идентификатор каждой категории. Тип данных − SERIAL.

− product\_category\_name : хранит название категории продукта. Тип данных − VARCHAR.

Таблица содержит ограничение первичного ключа для столбца product\_category\_id, а также все столбцы должны содержать какое-либо значение.

2.4 Вывод

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

3. Разработка объектов базы данных

3.1 Создание необходимых объектов

Для реализации базы данных салона автомобилей необходимо создать набор объектов базы данных, которые представляют собой таблицы, индексы, ограничения, хранимые процедуры и. Все эти объекты должны быть созданы в соответствии с требованиями.

В проектируемой базе данных были созданы 11 таблиц, описание которых приведено выше.

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

3.2 Функции

Функция в PostgreSQL – это объект базы данных, который возвращает значение на основе переданных в нее аргументов. Функции могут быть созданы с помощью оператора CREATE OR REPLACE FUNCTION. Для данного проекта были разработаны следующие функции:

* add\_in\_basket2: Добавляет продукт в корзину по UUID клиента, ID продукта и количеству
* add\_manager: Добавляет менеджера с переданными данными (имя, фамилия, телефон, адрес)
* add\_order: Добавляет заказ по UUID клиента и флагу наличия
* add\_product: Добавляет продукт с переданными данными (название, описание, изображение, цена, наличие, категория, свойства)
* add\_product2: Добавляет продукт с переданными данными (название, описание, изображение, цена, наличие, категория)
* add\_product\_category: Добавляет категорию продукта с переданным названием
* add\_product\_to\_basket: Добавляет продукт в корзину по UUID клиента, ID продукта и количеству
* add\_property\_for\_product2: Добавляет свойство продукта по ID продукта, названию и значению свойства
* calculate\_monthly\_revenue: Рассчитывает месячный доход за указанный период времени
* get\_all\_customers: Возвращает список всех клиентов
* get\_all\_orders: Возвращает список всех заказов
* get\_all\_products\_preview: Возвращает предварительный просмотр всех продуктов
* get\_all\_products\_preview\_from\_view: Возвращает предварительный просмотр всех продуктов из представления
* get\_all\_reviews: Возвращает все отзывы по UUID продукта
* get\_authorized\_employees: Возвращает список всех авторизованных сотрудников
* get\_courier\_orders: Возвращает список всех заказов курьеров
* get\_current\_user: Возвращает информацию о текущем пользователе по его UUID
* get\_customer\_basket2: Возвращает корзину клиента по его UUID
* get\_customer\_basket\_property: Возвращает свойства продукта корзины по ID продукта
* get\_customer\_basket\_product: Возвращает продукт корзины клиента по UUID клиента и ID продукта
* get\_customer\_order: Возвращает заказ клиента по его UUID
* get\_product\_info: Возвращает информацию о продукте по его ID
* get\_paged\_data: Возвращает постраничные данные по номеру страницы и количеству записей на странице
* get\_paged\_data\_instock: Возвращает постраничные данные о наличии по номеру страницы и количеству записей на странице
* get\_paged\_data\_instock (with filters): Возвращает постраничные данные о наличии с фильтрами по номеру страницы, количеству записей на странице, категории и названию
* get\_products\_count: Возвращает количество всех продуктов
* get\_products\_count\_instock: Возвращает количество всех продуктов в наличии
* get\_unauthorized\_employees: Возвращает список всех неавторизованных сотрудников
* get\_user\_role: Возвращает роль пользователя по его UUID
* login: Выполняет вход пользователя по переданным данным (логин, пароль) и возвращает UUID, имя, фамилию и роль
* signup: Регистрирует нового пользователя по переданным данным и возвращает UUID и имя пользователя

Пример реализации функции, которая выводит все заказы конкретного клиента – листинг 3.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION procedures.*get\_customer\_order*(customer\_id uuid)  RETURNS SETOF data.all\_orders\_without\_property\_view  LANGUAGE plpgsql  SECURITY DEFINER SET SEARCH\_PATH = procedures AS $$ BEGIN  RETURN QUERY  SELECT \*  FROM (SELECT DISTINCT order\_id,  order\_status,  order\_date,  order\_price,  order\_delivery,  fk\_customer\_id,  product\_name,  product\_image,  product\_price,  product\_count  FROM data.all\_orders\_view  WHERE fk\_customer\_id = customer\_id  AND order\_status IN ('Отменен', 'Передан в доставку', 'В пути', 'Готовится', 'Оформлен',  'Завершен(Доставлен)', 'Завершен(Выдан)')) AS subquery  ORDER BY CASE  WHEN order\_status = 'Оформлен' THEN 1  WHEN order\_status = 'Готовится' THEN 2  WHEN order\_status = 'Готовится' THEN 2  WHEN order\_status = 'Передан в доставку' THEN 3  WHEN order\_status = 'В пути' THEN 4  WHEN order\_status = 'Завершен(Доставлен)' THEN 5  WHEN order\_status = 'Завершен(Выдан)' THEN 6  WHEN order\_status = 'Отменен' THEN 7  ELSE 8  END;   RETURN; END; $$; |
|  |

Листинг 3.1 – Функция получения заказов машин конкретного клиента

Здесь мы создаем функцию get\_orders\_cars\_by\_manager\_id, которая принимает число (уникальный номер менеджера) и возвращает таблицу с информацией о его заказах машин.

3.3 Процедуры

Процедура в PostgreSQL – это объект базы данных, который представляет собой набор SQL-инструкций, которые могут быть вызваны для выполнения определенной задачи. Процедуры могут быть созданы с помощью оператора CREATE PROCEDURE. Для данного проекта были разработаны следующие процедуры:

* accept\_order: Принимает заказ по переданному UUID заказа и UUID клиента
* add\_courier: Добавляет курьера с переданными данными (имя, фамилия, телефон, адрес)
* add\_customer: Добавляет клиента с переданными данными (имя, фамилия, email, телефон, адрес)
* add\_in\_basket2: Добавляет продукт в корзину по UUID клиента, ID продукта и количеству
* add\_manager: Добавляет менеджера с переданными данными (имя, фамилия, телефон, адрес)
* add\_order: Добавляет заказ по UUID клиента и флагу наличия
* add\_product: Добавляет продукт с переданными данными (название, описание, изображение, цена, наличие, категория, свойства)
* add\_product2: Добавляет продукт с переданными данными (название, описание, изображение, цена, наличие, категория)
* add\_product\_category: Добавляет категорию продукта с переданным названием
* cancel\_order: Отменяет заказ по его UUID
* complete\_deliver\_order: Завершает доставку заказа по UUID заказа и UUID курьера
* complete\_order: Завершает заказ по UUID заказа и UUID клиента
* deauthorize\_employee: Деавторизует сотрудника по его UUID
* delete\_employee: Удаляет сотрудника по его UUID
* delete\_from\_basket: Удаляет продукт из корзины по ID продукта
* delete\_product: Удаляет продукт по его ID
* delete\_product\_category: Удаляет категорию продукта по названию категории
* delete\_review: Удаляет отзыв по его ID
* delete\_review\_from\_customer: Удаляет отзыв от клиента по ID отзыва и UUID клиента
* deliver\_order: Доставляет заказ по UUID заказа и UUID курьера
* get\_products\_count: Возвращает количество всех продуктов
* get\_products\_count\_instock: Возвращает количество всех продуктов в наличии
* get\_unauthorized\_employees: Возвращает список всех неавторизованных сотрудников
* get\_user\_role: Возвращает роль пользователя по его UUID
* login: Выполняет вход пользователя по переданным данным (логин, пароль) и возвращает UUID, имя, фамилию и роль
* signup: Регистрирует нового пользователя по переданным данным и возвращает UUID и имя пользователя

Пример определения процедуры, которая добавляет новый заказ машины – листинг 3.2.

|  |
| --- |
|  |
| CREATE OR REPLACE PROCEDURE procedures.*cancel\_order*(order\_id\_ uuid)  LANGUAGE plpgsql  SECURITY DEFINER SET SEARCH\_PATH = procedures AS $$ DECLARE  status varchar;  BEGIN  SELECT order\_status  INTO status  FROM data."order"  WHERE order\_id = order\_id\_;  IF status = 'Оформлен' THEN  UPDATE data."order"  SET  order\_status = 'Отменен'  WHERE order\_id = order\_id\_;  RAISE NOTICE 'Статус заказа % изменен на "Отменен"', order\_id\_;  ELSE  RAISE NOTICE 'Невозможно изменить статус заказа %, так как текущий статус не является "Оформлен"', order\_id\_;  END IF;  IF NOT FOUND THEN  RAISE NOTICE 'Заказ с id % не найден.', order\_id\_;  END IF;  END;  $$; |

Листинг 3.2 – Процедура отмены заказа

В сумме, при разработке курсового проекта, было создано 32 процедуры для следующих целей:

1. Добавление данных в таблицы.
2. Удаление данных из таблиц.
3. Изменение данных в таблицах.
4. Экспорт и импорт таблицы в формат JSON.

Весь перечень созданных процедур представлен в Приложении Б.

3.4 Триггеры

Триггер в PostgreSQL – это объект базы данных, который автоматически запускается при определенном событии в базе данных, таком как вставка, обновление или удаление данных. Триггеры могут быть созданы с помощью оператора CREATE TRIGGER. Для данного проекта были разработаны следующие триггеры:

send\_email\_trigger\_insert: вызывается при создании новой записи в продукта

send\_email\_trigger\_update: вызывается при изменении новой записи в продуктах

Пример определения триггера, который срабатывает при добавлении записи в таблицу – листинг 3.3.

|  |
| --- |
| CREATE OR REPLACE TRIGGER send\_email\_trigger\_insert  AFTER INSERT  ON data.product  FOR EACH ROW  EXECUTE FUNCTION procedures.send\_email\_insert\_trigger\_function('For the PRODUCT table, ADDING an object with an id: '); |
|  |
|  |
|  |

Листинг 3.3 – Созданный в проекте индекс

3.5 Индексы

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

Индексы могут быть созданы с помощь оператора CREATE INDEX. Созданные индексы представлены на листинге ниже.

|  |
| --- |
| CREATE index idx\_product\_price on data.product(product\_price); |

Листинг 3.4 – Созданный в проекте индекс

3.6 Вывод

В данном разделе были разработаны и созданы объекты для взаимодействия с базой данных, такие как функции, процедуры, триггеры, индексы.

4.Описание процедур экспорта и импорта.

JSON — это формат данных, предназначенный для хранения и передачи информации в структурированном виде ключ – значение.

* 1. Процедура экспорта данных

Пример функии записи в json виден на листинге 4.1

|  |
| --- |
| CREATE OR REPLACE FUNCTION generate\_json() |
| RETURNS VOID AS $$ |
| BEGIN |
| EXECUTE format('COPY ( |
| SELECT to\_json(seller) FROM data.seller |
| ) TO %L', 'C:\ seller.json'); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг 4.1 – Функция записи в JSON файл

4.2 Процедура импорта данных

Функция импорта из JSON используется для таблицы «REVIEWS» для добавления новых отзывов из вне базы данных. Данная процедура представлена в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_review\_from\_json() AS $$ |
| BEGIN |
| CREATE TEMP TABLE temp\_reviews ( |
| data JSONB |
| ); |
|  |
| COPY temp\_reviews(data) FROM 'C:/reviews.json'; |
|  |
| INSERT INTO reviews (customer\_id, date, rating, review) |
| SELECT |
| (data->>'CUSTOMER\_ID')::INT, |
| (data->>'DATE')::DATE, |
| (data->>'RATING')::INT, |
| (data->>'REVIEW')::VARCHAR(1000) |
| FROM temp\_reviews; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг 4.2 – Процедура импорта из JSON

4.3 Вывод

В данном разделе были описаны примеры разработанных процедур, функция и триггеров импорта и экспорта данных для таблиц «ORDERCAR», «ORDERSPAREPART» и «REVIEWS». Также были продемонстрированы листинги кода процедур.

5. Тестирование производительности базы данных.

Производительность относится к производительности пользователей, систем или сред. Эффективность может характеризовать производительность, но также учитывает затраты на программирование, обработку данных пользователями и системой, затраты на приобретение и обслуживание аппаратного и программного обеспечения, соответствие требованиям и безопасность, устойчивость.

Так же стоит отметить, что производительность и результативность нелегко поддаются количественной оценке. Не существует единого определения, применимого к каждому контексту. Часто даже не ясно, как это измерить производительность или даже прирост производительности. В этом параграфе представлены меры по оптимизации производительности, направленные на минимизацию затрат процессорного времени, реального времени.

5.1 Заполнение таблицы 100000 строк

Для заполнения таблицы 100000 строк был написан скрипт SQL. Листинг скрипта приведен ниже.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE generate\_products()  LANGUAGE plpgsql  AS  $$  DECLARE  i INTEGER := 41000;  num\_properties INTEGER;  image\_value bytea; -- Переменная для хранения значения изображения  result RECORD;  BEGIN  -- Извлекаем значение изображения из другой таблицы  SELECT product\_image INTO image\_value  FROM data.product  WHERE product\_name = 'Product 8';  WHILE i <= 50000  LOOP  num\_properties := floor(random() \* 3) + 1; -- Генерация случайного числа свойств от 1 до 3  SELECT \* INTO result FROM procedures.add\_product2(  'Product ' || i::varchar,  'Description of product ' || i::varchar,  image\_value, -- Используем значение изображения из другой таблицы  round(random() \* 100),  CASE round(random())::INTEGER WHEN 1 THEN TRUE ELSE FALSE END,  'Пицца'::varchar  );  -- Делаем что-то с результатом, если нужно  RAISE NOTICE 'ID продукта: %, Ошибка: %', result.id, result.error;  i := i + 1;  END LOOP;  END;  $$; |
|  |

Листинг 3.5 – Код скрипта вставки 100000 строк в таблицу Products

5.2 Тестирование производительности на таблице Product

Ниже на рисунке 5.1 представлен запрос до создание и применения индекса.

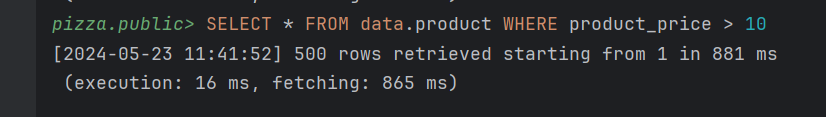


Рисунок 5.1 – Результат запроса без индекса

Затем я применяю индекс и новый результат виден на рисунке 5.2. Как видно время изменилось с 881 до 120ms



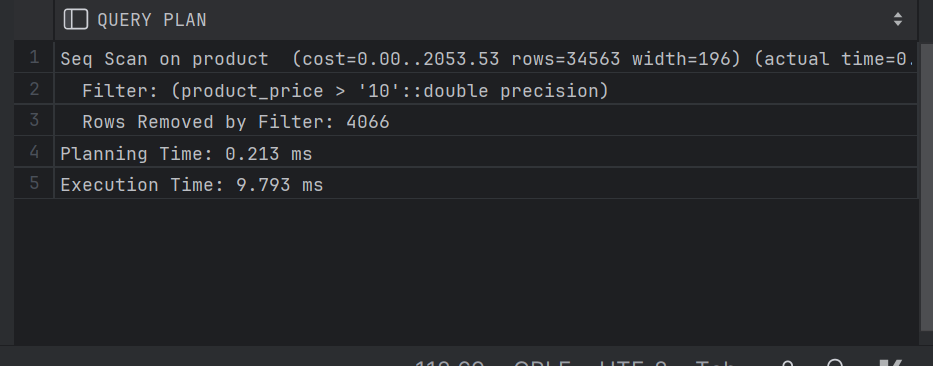


Рисунок 5.2 – Результат запроса индекса



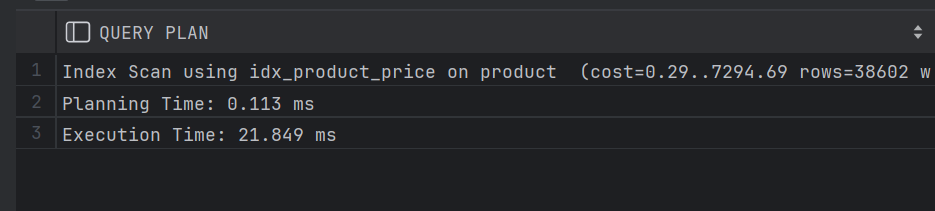


Рисунок 5.3 – Результат запроса c индекса на Order By

5.3 Вывод

Тестирование производительности базы данных на примере таблицы «Products» в PostgreSQL показало улучшение времени выполнения запросов после добавления индексов.

Результаты тестирования подтверждают, что создание индексов над таблицами в PostgreSQL существенно улучшает производительность запросов. Особенно заметно снижение времени выполнения запросов с использованием JOIN-операций и условий ORDER BY.

6. Описание технологии и ее применение в базе данных

6.1 Технология «Система e-mail уведомлений о событиях в базе данных»

Технология email уведомлений помогает админимтраторам базы данной, или тем кто с ней работает, знать когда и что происходит, не утруждая себя постоянным мониторингом. Эта функциональность позволяет оперативно информировать заинтересованных пользователей о важных событиях или обновлениях, происходящих в базе данных.

Для подключения библиотеки нужно расположить скомпилированные файлы расширения в нужные директории Postgresql.

Далее вам нужно скачать его себе на сервер, что видно на листинге 6.1.

|  |
| --- |
| CREATE EXTENSION send\_email\_v2 SCHEMA procedures;  CREATE EXTENSION pg\_background SCHEMA procedures; |

Листинг 6.1 – Установка библиотеки

6.2 Вызов библиотеки в Sql.

Далее самый просто вызов функции библиотеки видел на листинге 6.2.

|  |
| --- |
| SELECT procedures.*send\_email\_v2*('kuncovs19@gmail.com', 'kuncovs1.0@gmail.com', 'Notification', 'Notification from pizza database'); |

Листинг 6.2 – Вызов функции с помощью select.

Так как библиотеке нужно время на выполнение запроса, мы модем обернуть запрос в функцию pg\_background\_launch, дабы выполнять все на фоне и не останавливать поток выполнения. Этаа функция видна на листинге 6.3.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE procedures.*send\_email\_procedure*(  p\_subject text,  p\_message text )  LANGUAGE plpgsql AS $$ DECLARE  sql text; BEGIN  RAISE NOTICE 'Result.';  RAISE NOTICE '1. %', p\_subject;  RAISE NOTICE '2. %', p\_message;   sql := 'SELECT send\_email\_v2(' || *quote\_literal*('kuncovs19@gmail.com') || ', ' || *quote\_literal*('kuncovs1.0@gmail.com') || ', ' ||  *quote\_literal*(p\_subject) || ', ' || *quote\_literal*(p\_message) || ')';   RAISE NOTICE '%', sql;   PERFORM procedures.*pg\_background\_launch*(sql); END; $$; |

Листинг 6.3 – Использование pg\_background\_launch.

6.3 Механизм триггеров и функций.

|  |
| --- |
| CREATE OR REPLACE TRIGGER send\_email\_trigger\_insert  AFTER INSERT  ON data.product  FOR EACH ROW  EXECUTE FUNCTION procedures.send\_email\_insert\_trigger\_function('For the PRODUCT table, ADDING an object with an id: '); |

Листинг 6.4 – Структура Триггера

|  |
| --- |
| CREATE OR REPLACE FUNCTION procedures.*send\_email\_insert\_trigger\_function*()  RETURNS TRIGGER  LANGUAGE plpgsql   SECURITY DEFINER SET SEARCH\_PATH = procedures AS $$ DECLARE  trigger\_text text;  trigger\_text\_result text; BEGIN  trigger\_text := TG\_ARGV[0];  trigger\_text\_result := trigger\_text || *quote\_literal*(NEW.product\_id) || '.';   RAISE NOTICE '##################: %', 1;  RAISE NOTICE 'trigger\_text: %', trigger\_text;  RAISE NOTICE 'last\_inserted\_object: %', *quote\_literal*(NEW.product\_id);  RAISE NOTICE 'trigger\_text\_result: %', trigger\_text\_result;   CALL *send\_email\_procedure*('Notification!', trigger\_text\_result);   RETURN NEW; END; $$; |

Листинг 6.5 – Функция вызываемая триггером.

6.4 Програмный код библиотеки e-mail уведомлений.

Для написания кода самой библиотеки я использовал язык C для отправки curl запрос на сервер сервиса Gmail. Главный компилируемый файл представлен на листинге 6.6.

|  |
| --- |
| #include "postgres.h"  #include "postgres\_ext.h"  #include "send\_email.h"  #include "utils/builtins.h"  #include "utils/bytea.h"  #include "utils/expandeddatum.h"  #include "lib/stringinfo.h"  PG\_MODULE\_MAGIC;  PG\_FUNCTION\_INFO\_V1(send\_email);  struct upload\_status {  size\_t bytes\_read;  const char\* data;  };  static size\_t payload\_source(char\* ptr, size\_t size, size\_t nmemb, void\* userp) {  struct upload\_status\* upload\_ctx = (struct upload\_status\*)userp;  const char\* body = upload\_ctx->data;  if ((size == 0) || (nmemb == 0) || ((size \* nmemb) < 1)) {  return 0;  }  const char\* data = body + upload\_ctx->bytes\_read;  int len = strlen(body) - upload\_ctx->bytes\_read;  if (len > size)  len = size;  memcpy(ptr, data, len);  upload\_ctx->bytes\_read += len;  return len;  }  Datum send\_email(PG\_FUNCTION\_ARGS) {  text\* from\_mail = PG\_GETARG\_TEXT\_PP(0);  text\* to\_mail = PG\_GETARG\_TEXT\_PP(1);  text\* subject = PG\_GETARG\_TEXT\_PP(2);  text\* body = PG\_GETARG\_TEXT\_PP(3);  CURL\* curl;  CURLcode res = CURLE\_OK;  struct curl\_slist\* recipients = NULL;  struct upload\_status upload\_ctx = { 0 };  upload\_ctx.data = VARDATA(body);  curl\_global\_init(CURL\_GLOBAL\_DEFAULT);  curl = curl\_easy\_init();  if (curl) {  /\* Set username and password \*/  curl\_easy\_setopt(curl, CURLOPT\_USERNAME, "gmass");  curl\_easy\_setopt(curl, CURLOPT\_PASSWORD, "39bce97d-0c42-4179-8cf2-8aad14091ede");  curl\_easy\_setopt(curl, CURLOPT\_URL, "smtp://smtp.gmass.co:587");  curl\_easy\_setopt(curl, CURLOPT\_USE\_SSL, (long)CURLUSESSL\_ALL);  char from\_mail\_str[MAXIMUM\_ALIGNOF];  snprintf(from\_mail\_str, sizeof(from\_mail\_str), "From: %s\r\n", text\_to\_cstring(from\_mail));  curl\_easy\_setopt(curl, CURLOPT\_MAIL\_FROM, from\_mail\_str);  char to\_mail\_str[MAXIMUM\_ALIGNOF];  snprintf(to\_mail\_str, sizeof(to\_mail\_str), "To: %s\r\n", text\_to\_cstring(to\_mail));  recipients = curl\_slist\_append(recipients, to\_mail\_str);  recipients = curl\_slist\_append(recipients, "<info@example.com>");  curl\_easy\_setopt(curl, CURLOPT\_MAIL\_RCPT, recipients);  char subject\_str[MAXIMUM\_ALIGNOF];  snprintf(subject\_str, sizeof(subject\_str), "Subject: %s\r\n", text\_to\_cstring(subject));  curl\_easy\_setopt(curl, CURLOPT\_READFUNCTION, payload\_source);  curl\_easy\_setopt(curl, CURLOPT\_READDATA, &upload\_ctx);  curl\_easy\_setopt(curl, CURLOPT\_UPLOAD, 1L);  res = curl\_easy\_perform(curl);  /\* Check for errors \*/  if (res != CURLE\_OK)  elog(ERROR, "curl\_easy\_perform() failed: %s\n", curl\_easy\_strerror(res));  /\* Always cleanup \*/  curl\_easy\_cleanup(curl);  }  curl\_global\_cleanup();  PG\_RETURN\_INT32(res);  } |

Листинг 6.6 – Демонстрация С кода библиотеки.

6.4 Вывод

Технология "Система e-mail уведомлений о событиях в базе данных" в PostgreSQL предоставляет удобный способ отправки электронных уведомлений при возникновении определенных событий или изменений в базе данных. Она позволяет оперативно информировать заинтересованных пользователей о важных событиях и обновлениях, что может быть полезно для мониторинга и управления базой данных. При использовании этой технологии необходимо учитывать безопасность и правильно настроить параметры сервера для отправки электронных писем.

7. Руководство пользователя

Для взаимодействия с базой данных было разработано функциональное приложения с предоставлением доступа к окнам пользователя, менеджера, механика и администратора. Данное приложение использует для отображения информации разработанных таблиц и взаимодействием с базой данных процедуры, функции и триггеры, разрешения на выполнение которых выданы в соответствии с разработанными требованиями и бизнес-логики.

7.1 Инструкция по эксплуатации для пользователя «Customer»

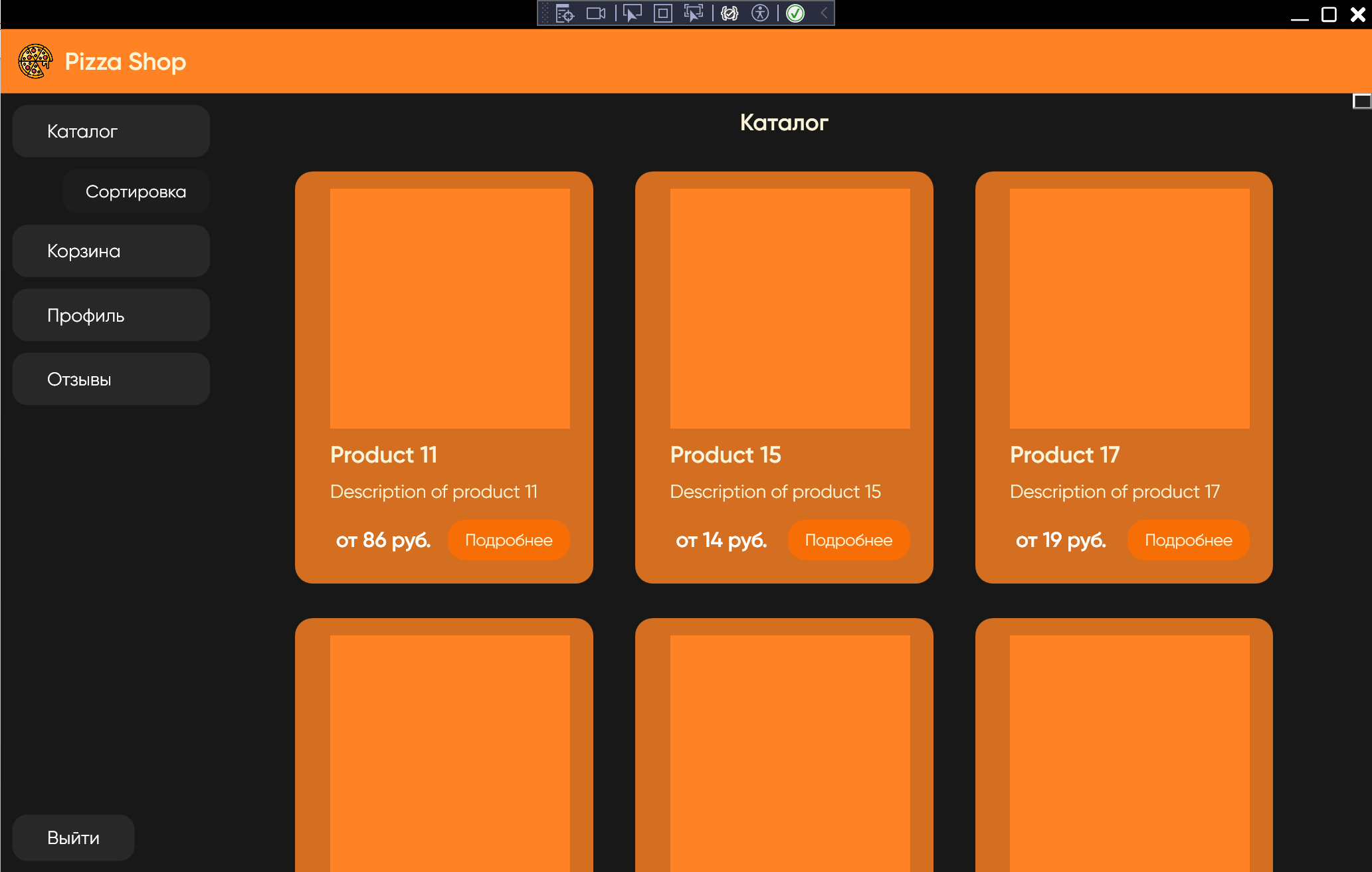


Рисунок 7.1 – Окно взаимодействия пользователя

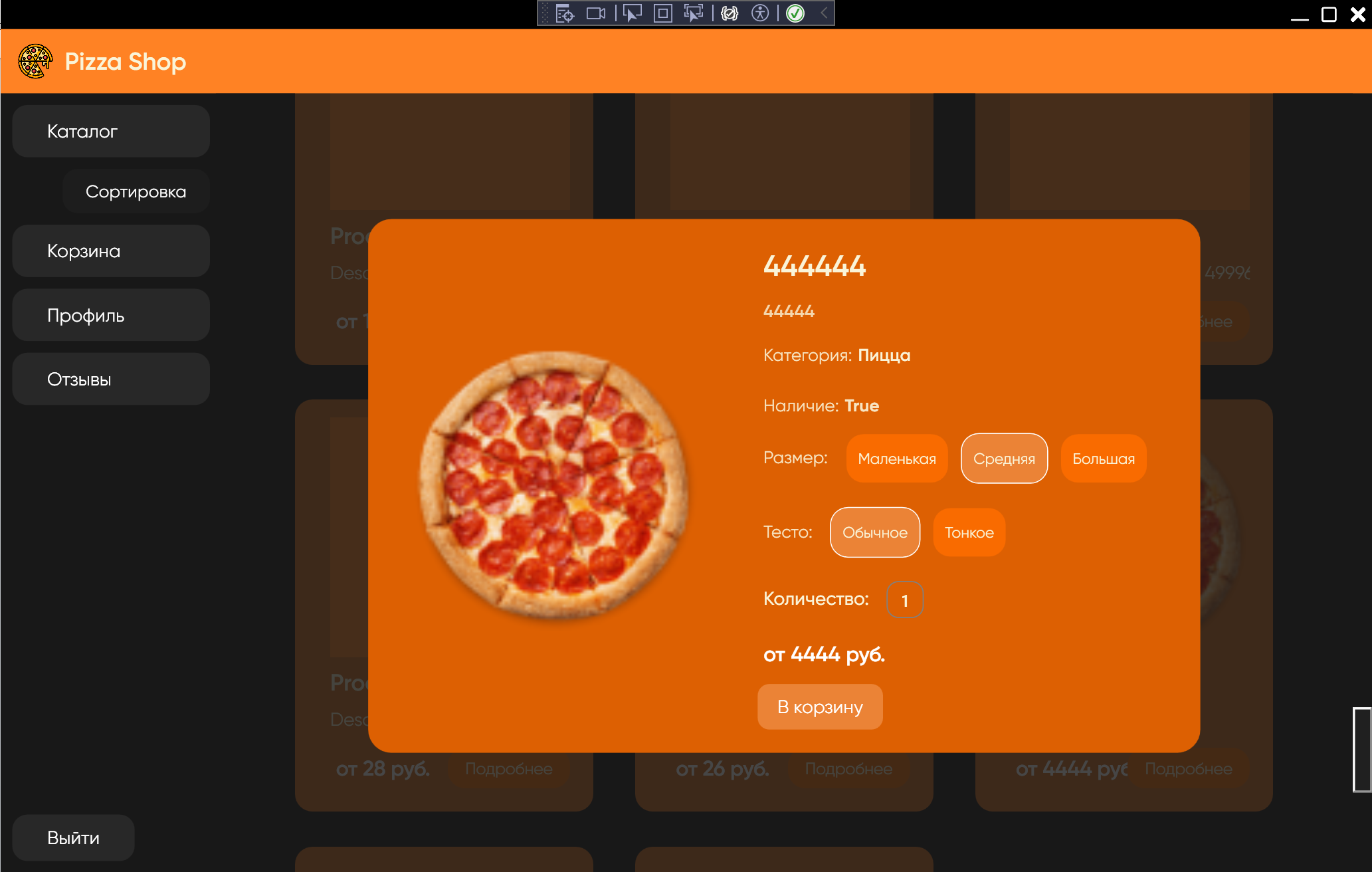


Рисунок 7.2 – Окно взаимодействия пользователя с конкретным продуктом

На приведенном интерфейсе программы для пользователя автосалона отображаются следующие варианты использования:

Просмотр информации:

− Просмотреть информацию о каталоге: Возможность просмотра подробной информации с возможность добавление в корзину.

Сортировка:

−Сортировка продуктов по нужным критерями от большего к меньшему.

Отзывы:

− Просмотр отзывов: Просмотре отзывово оставленных собой, а также и другими пользователями.

Заказы и корзина:

− Добавить продукт в корзину: возможность выбрать сразу несколько товаров с разными свойствами и количеством.

− Сделать заказ: возможность оформить заказ на покупку выбранной корзины с возможностью выбор доставки.

Просмотр профиля:

− Просмотреть профиль и свой краткую информацию, а также изменить адрес доставки.

История заказов:

− Просмотреть историю заказов продуктов: возможность просмотреть историю всех своих заказов из корзины.

История обслуживания:

Эти варианты использования позволяют пользователю пиццерии выполнять основные действия, связанные с выбором, составление и оформлением заказа, с последующей возможностью просмотреть историю своих заказов.

7.2 Инструкция по эксплуатации для пользователя «Manager»

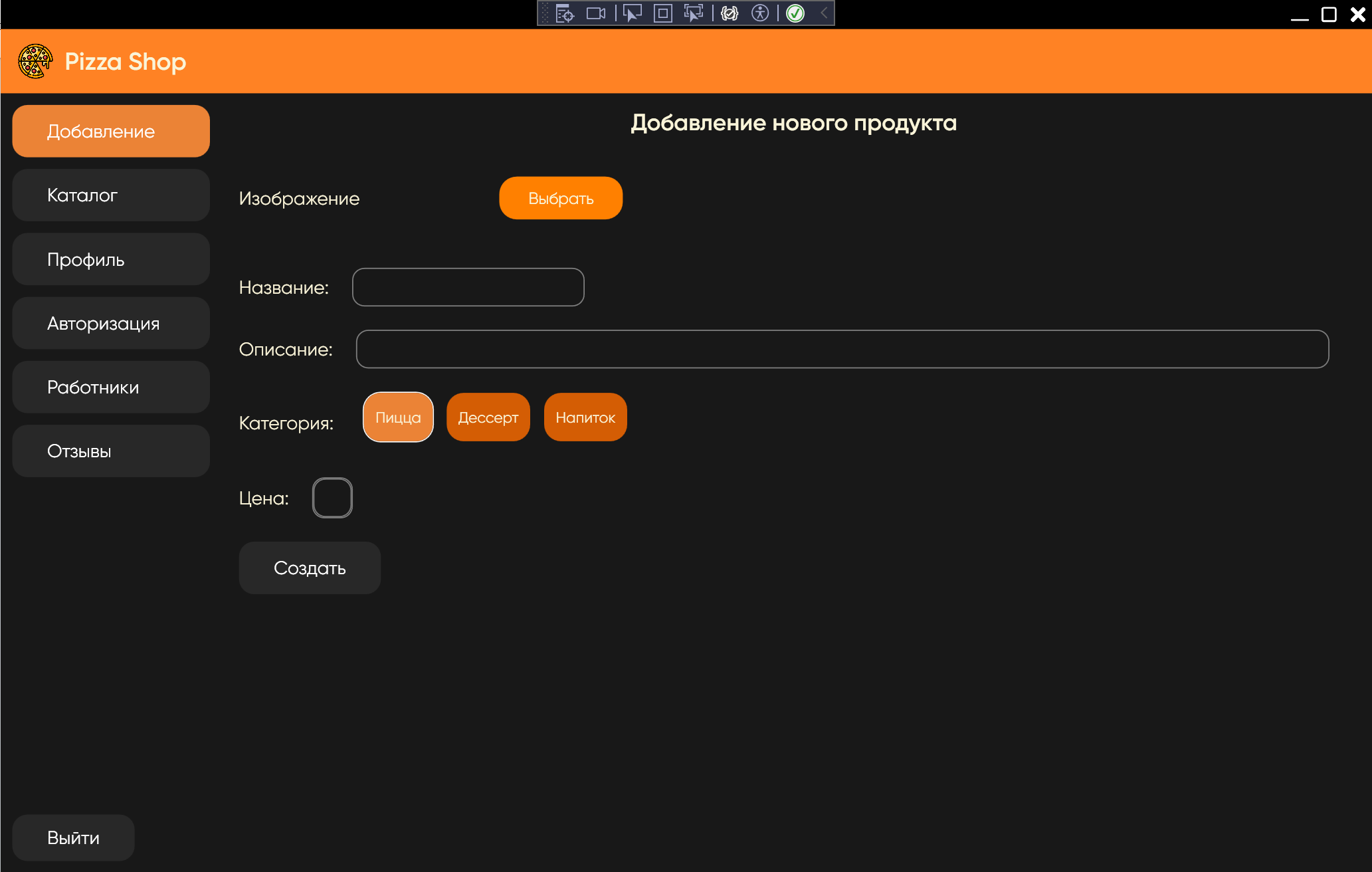


Рисунок 7.2 – Окно взаимодействия пользователя

На приведенном интерфейсе программы для управления менеджером автосалона отображаются следующие варианты использования:

Создание нового продукта:

* Выбрать название, описание, изобрадение, категорию и цену.

Управление каталогом и продуктами:

− Удалить продукт.

− Обновить продукт: редактирование информации об уже существующем продукте.

Управление работниками:

− Авторизация работника: Дать возможность новому сотруднику официально начать работать.

− Удалить работника

* Разлогинить работника: временно забрать возможность входа в приложение.

Управление отзывами:

* Просмотр всех отзывов.
* Удаление любого отзыва.

Эти варианты использования позволяют пользователю менеджеру управлять различными пиццерии, включая управление каталогом, своими работниками.

7.3 Инструкция по эксплуатации для пользователя «Seller» и «Courier»

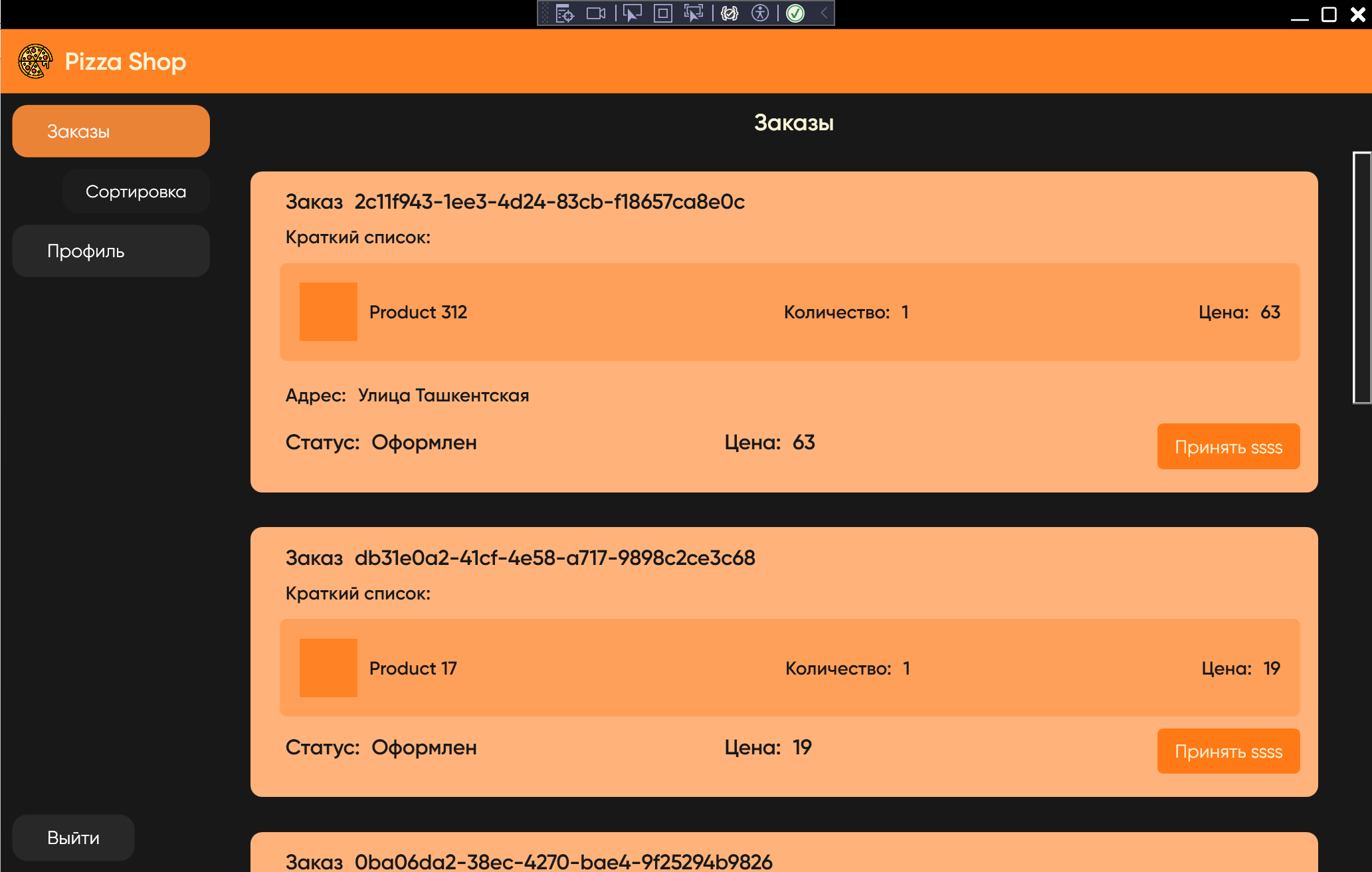


Рисунок 7.3 – Окно взаимодействия Продавца

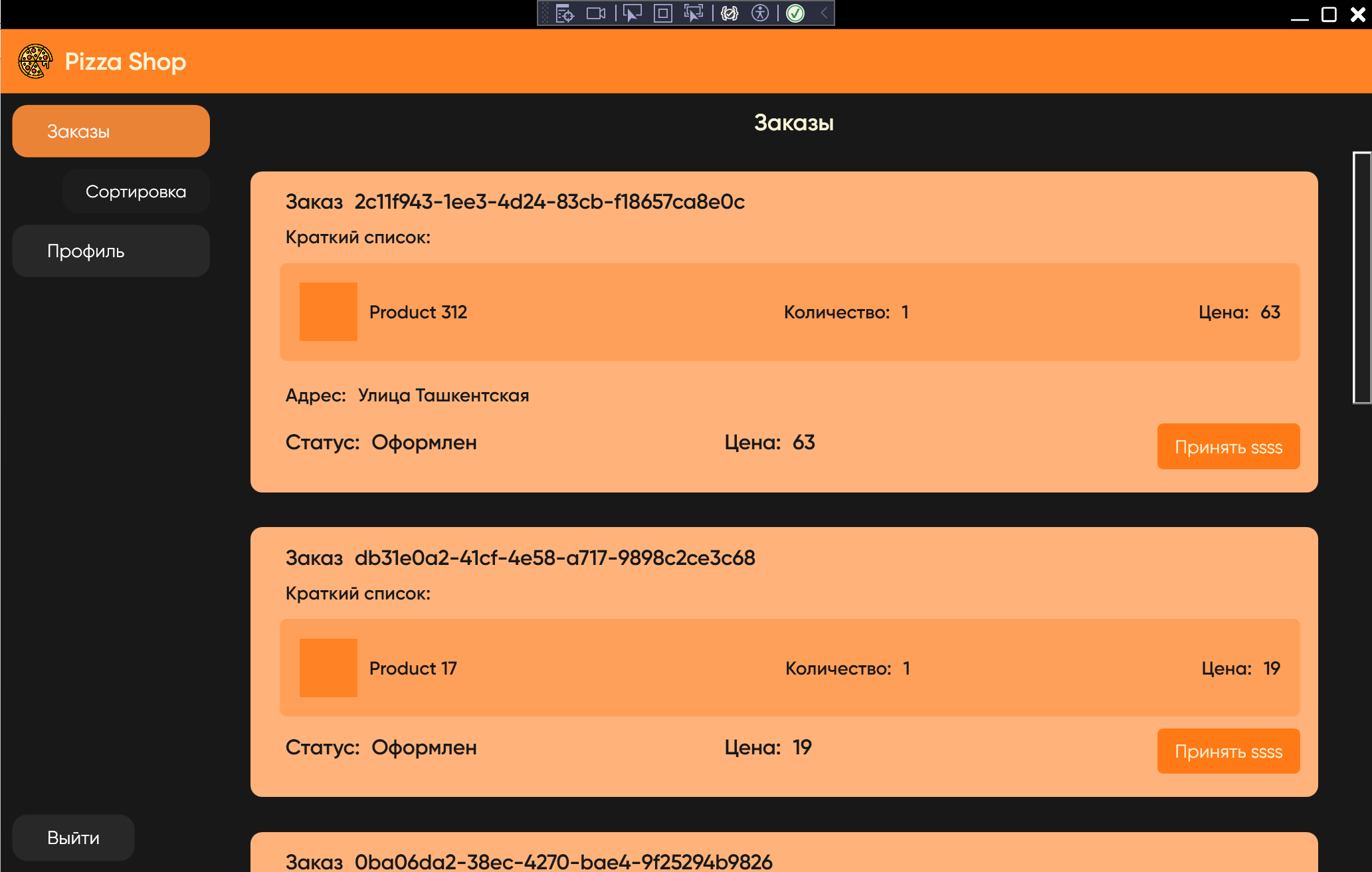


Рисунок 7.3 – Окно взаимодействия Курьера

На приведенном интерфейсе программы отображаются следующие варианты использования:

Просмотр заказов продавца:

* Просмотреть заказы.
* Принять, передать курьеру для доставки.
* Приготовить и выдать заказ.

Просмотр заказов курьера:

* Просмотреть заказы.
* Принять и доставить заказы.

Эти варианты использования позволяют механику эффективно управлять своими задачами, контролировать процесс выполнения работ, и менять статусы заказов.

7.4 Вывод

Данный раздел описывает функционал приложения, включающий в себя возможности для различных категорий пользователей. В приложении предусмотрены разнообразные функции, позволяющие эффективно управлять заказами, катологом, персоналом, а также обеспечивать взаимодействие с базой данных через процедуры, функции и триггеры.

Заключение

В ходе исследования и разработки курсового проекта были рассмотрены и проанализированы успешные аналоги сайтов автосалонов «Пицца лисицца» и «Додо пицца», которые продемонстрировали важность использования баз данных для управления пиццерией и предоставления удобного доступа к информации для клиентов. Анализ функционала этих сайтов показал, что ключевыми элементами являются таблицы, содержащие информацию о продуктах и заказах.

На основе этого анализа была разработана архитектура проекта, описано взаимодействие приложения с базой данных. В проекте была описана структура базы данных вместе с ограничениями целостности, а также разработаны и созданы объекты для взаимодействия с базой данных, такие как функции, процедуры, триггеры и индексы.

Примеры разработанных процедур, функций и триггеров, включая процессы импорта и экспорта.

Тестирование производительности базы данных на примере таблицы «Products» в PostgreSQL показало значительное улучшение времени выполнения запросов после добавления индексов. Результаты тестирования подтвердили, что создание индексов существенно улучшает производительность запросов, особенно при условий ORDER BY.

Для обеспечения удобности и легкости в использовании базы данных была разработана система e-mail уведомлений о событиях с базе данных.

Таким образом, разработанное приложение предоставляет разнообразные функции для различных категорий пользователей, включая клиентов, менеджеров, продавцов и курьеров. Оно позволяет эффективно управлять заказами, каталогом и персоналом, обеспечивая надежное и производительное взаимодействие с базой данных.

Список использованных литературных источников

1. Пицца лисицца [Электронный ресурс] / Режим доступа: https://pzz.by/ – Дата доступа: 20.03.2024.
2. Додо пицца [Электронный ресурс] / Режим доступа: https://dodopizza.by/minsk/ – Дата доступа: 20.03.2024.
3. Лузанов П. В., Рогов, Лёвшин. Postgres 15. Первое знакомство. 2023.
4. Лесовский А. В. Мониторинг PostgreSQL. М.: Бумба, 2024. 247 с. ISBN 978-5-907754-42-3.
5. Домбровская Г., Новиков Б., Бейликова А. Оптимизация запросов в PostgreSQL / пер. с англ. Д. А. Беликова. М.: ДМК Пресс, 2022. 278 с.: ил.
6. Advanced SQL with SAS®. Cary, NC, USA: SAS Institute Inc., 2022. ISBN 978-1-955977-88-3 (Web PDF)
7. Официальный сайт PostgreSQL [Электронный ресурс] / Справочник Режим доступа: https://www.postgresql.org/ – Дата доступа 20.03.2024.
8. Официальный сайт pgAdmin 4 [Электронный ресурс] / Справочник Режим доступа: https://www.pgadmin.org/docs/ – Дата доступа 20.03.2024

1. [Postgres Auditing in 150 lines of SQL (zone-www-dot-com-git-fix-ai-guide-supabase.vercel.app)](https://zone-www-dot-com-git-fix-ai-guide-supabase.vercel.app/blog/postgres-audit) / Режим доступа: <https://zone-www-dot-com-git-fix-ai-guide-supabase.versel.app/blog/postgres-audit> – Дата доступа: 15.04.2024.

Приложение

Приложение А

Структура базы данных

|  |
| --- |
| CREATE TABLE MECHANIC( |
| ID SERIAL NOT NULL UNIQUE, |
| SECONDNAME VARCHAR(20) NOT NULL, |
| FIRSTNAME VARCHAR(20) NOT NULL, |
| THIRDNAME VARCHAR(20) NOT NULL, |
| MAIL VARCHAR(40) NOT NULL UNIQUE, |
| PHONE VARCHAR(16) NOT NULL UNIQUE, |
| PRIMARY KEY (ID) |
| ); |

Листинг А.1 – Скрипт создания таблицы MECHANIC

|  |
| --- |
| CREATE TABLE CUSTOMER( |
| ID SERIAL NOT NULL UNIQUE, |
| SECONDNAME VARCHAR(20) NOT NULL, |
| FIRSTNAME VARCHAR(20) NOT NULL, |
| THIRDNAME VARCHAR(20) NOT NULL, |
| MAIL VARCHAR(40) NOT NULL UNIQUE, |
| PHONE VARCHAR(16) NOT NULL UNIQUE, |
| COUNTRY VARCHAR(50) NOT NULL, |
| ADDRESS VARCHAR(100) NOT NULL, |
| REQUISITES VARCHAR(16) NOT NULL, |
| PRIMARY KEY (ID) |
| ); |

Листинг А.2 – Скрипт создания таблицы CUSTOMER

|  |
| --- |
| CREATE TABLE MANAGER( |
| ID SERIAL NOT NULL UNIQUE, |
| SECONDNAME VARCHAR(20) NOT NULL, |
| FIRSTNAME VARCHAR(20) NOT NULL, |
| THIRDNAME VARCHAR(20) NOT NULL, |
| MAIL VARCHAR(40) NOT NULL UNIQUE, |
| PHONE VARCHAR(16) NOT NULL UNIQUE, |
| PRIMARY KEY (ID) |
| ); |

Листинг А.3 – Скрипт создания таблицы MANAGER

|  |
| --- |
| CREATE TABLE CARS( |
| ID SERIAL NOT NULL UNIQUE, |
| LABEL\_MODEL\_ID SERIAL NOT NULL, |
| YEAR INT NOT NULL, |
| MILEAGE NUMERIC NOT NULL, |
| ENGINETYPE VARCHAR(20) NOT NULL, |
| ENGINECAPACITY NUMERIC NOT NULL, |
| POWER INT NOT NULL, |
| PRICE MONEY NOT NULL, |
| DESCRIPTION VARCHAR(1000) NOT NULL, |
| STATUS BOOLEAN, |
| PRIMARY KEY (ID), |
| FOREIGN KEY (LABEL\_MODEL\_ID) REFERENCES LABEL\_MODEL(ID) |
| ); |

Листинг А.4 – Скрипт создания таблицы CARS

|  |
| --- |
| CREATE TABLE LABELS( |
| ID SERIAL NOT NULL UNIQUE, |
| LABEL VARCHAR(30) NOT NULL, |
| COUNTRY VARCHAR(50) NOT NULL, |
| PRIMARY KEY (ID) |
| ); |

Листинг А.5 – Скрипт создания таблицы LABELS

|  |
| --- |
| CREATE TABLE MODEL( |
| ID SERIAL NOT NULL UNIQUE, |
| MODEL VARCHAR(30) NOT NULL, |
| PRIMARY KEY (ID) |
| ); |

Листинг А.6 – Скрипт создания таблицы MODEL

|  |
| --- |
| CREATE TABLE LABEL\_MODEL( |
| ID SERIAL NOT NULL UNIQUE, |
| LABEL\_ID SERIAL NOT NULL, |
| MODEL\_ID SERIAL NOT NULL, |
| PRIMARY KEY (ID), |
| FOREIGN KEY (LABEL\_ID) REFERENCES LABELS(ID), |
| FOREIGN KEY (MODEL\_ID) REFERENCES MODEL(ID) |
| ); |

Листинг А.7 – Скрипт создания таблицы LABEL\_MODEL

|  |
| --- |
| CREATE TABLE ORDERCAR( |
| ID SERIAL NOT NULL UNIQUE, |
| CUSTOMER\_ID SERIAL NOT NULL, |
| MANAGER\_ID SERIAL NOT NULL, |
| DATE DATE NOT NULL, |
| CAR\_ID SERIAL NOT NULL, |
| STATUS BOOLEAN, |
| COMMENT VARCHAR(1000), |
| PRIMARY KEY (ID), |
| FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMER(ID), |
| FOREIGN KEY (MANAGER\_ID) REFERENCES MANAGER(ID), |
| FOREIGN KEY (CAR\_ID) REFERENCES CARS(ID) |
| ); |

Листинг А.8 – Скрипт создания таблицы ORDERCAR

|  |
| --- |
| CREATE TABLE SPAREPARTS( |
| ID SERIAL NOT NULL UNIQUE, |
| LABEL VARCHAR(50), |
| LABEL\_MODEL\_ID SERIAL NOT NULL, |
| DESCRIPTION VARCHAR(1000) NOT NULL, |
| QUANTITY INT NOT NULL, |
| PRICE MONEY NOT NULL, |
| STATUS BOOLEAN, |
| FOREIGN KEY (LABEL\_MODEL\_ID) REFERENCES LABEL\_MODEL(ID) |
| ); |

Листинг А.9 – Скрипт создания таблицы SPAREPARTS

|  |
| --- |
| CREATE TABLE ORDERSPAREPARTS( |
| ID SERIAL NOT NULL UNIQUE, |
| CUSTOMER\_ID SERIAL NOT NULL, |
| SPAREPART\_ID SERIAL NOT NULL, |
| DATE DATE NOT NULL, |
| QUANTITY INT NOT NULL, |
| STATUS BOOLEAN, |
| PRIMARY KEY (ID), |
| FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMER(ID), |
| FOREIGN KEY (SPAREPART\_ID) REFERENCES SPAREPARTS(ID) |
| ); |

Листинг А.10 – Скрипт создания таблицы ORDERSPAREPARTS

|  |
| --- |
| CREATE TABLE SERVICESHEET( |
| ID SERIAL NOT NULL UNIQUE, |
| CUSTOMER\_ID SERIAL NOT NULL, |
| MECHANIC\_ID SERIAL NOT NULL, |
| DATE DATE NOT NULL, |
| LABEL\_MODEL\_ID SERIAL NOT NULL, |
| PROBLEMDESCRIPTION VARCHAR(1000) NOT NULL, |
| PRICE MONEY, |
| STATUS BOOLEAN, |
| PRIMARY KEY (ID), |
| FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMER(ID), |
| FOREIGN KEY (MECHANIC\_ID) REFERENCES MECHANIC(ID), |
| FOREIGN KEY (LABEL\_MODEL\_ID) REFERENCES LABEL\_MODEL(ID) |
| ); |

Листинг А.11 – Скрипт создания таблицы SERVICESHEET

|  |
| --- |
| CREATE TABLE REVIEWS( |
| ID SERIAL NOT NULL UNIQUE, |
| CUSTOMER\_ID SERIAL NOT NULL, |
| DATE DATE NOT NULL, |
| RATING INT NOT NULL, |
| REVIEW VARCHAR(1000) NOT NULL, |
| PRIMARY KEY (ID), |
| FOREIGN KEY (CUSTOMER\_ID) REFERENCES CUSTOMER(ID) |
| ); |

Листинг А.12 – Скрипт создания таблицы REVIEWS

|  |
| --- |
| CREATE TABLE REPORTS\_CAR ( |
| CAR\_ID INT, |
| CAR\_LABEL VARCHAR(30), |
| CAR\_MODEL VARCHAR(30), |
| CAR\_YEAR INT, |
| CAR\_MILEAGE NUMERIC, |
| CAR\_ENGINETYPE VARCHAR(20), |
| CAR\_ENGINECAPACITY NUMERIC, |
| CAR\_POWER INT, |
| CAR\_PRICE MONEY, |
| TOTAL\_SALES INT, |
| TOTAL\_PRICE MONEY |
| ); |

Листинг А.13 – Скрипт создания таблицы ROPORTS\_CAR

|  |
| --- |
| CREATE TABLE REPORTS\_SPAREPART ( |
| SPAREPART\_ID INT, |
| SPAREPART\_LABEL VARCHAR(50), |
| LABEL VARCHAR(30), |
| MODEL VARCHAR(30), |
| SPAREPART\_QUANTITY INT, |
| SPAREPART\_PRICE MONEY, |
| TOTAL\_SALES INT, |
| TOTAL\_PRICE MONEY |
| ); |

Листинг А.14 – Скрипт создания таблицы REPORTS\_SPAREPART

Приложение Б

Листинги процедур

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_car( |
| new\_label\_model\_id NUMERIC, |
| new\_year INT, |
| new\_mileage NUMERIC, |
| new\_enginetype VARCHAR(20), |
| new\_enginecapacity NUMERIC, |
| new\_power INT, |
| new\_price MONEY, |
| new\_description VARCHAR(1000), |
| new\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO CARS (label\_model\_id, year, mileage, enginetype, enginecapacity, power, price, description, status) |
| VALUES (new\_label\_model\_id, new\_year, new\_mileage, new\_enginetype, new\_enginecapacity, new\_power, new\_price, new\_description, new\_status); |
|  |
| RAISE NOTICE 'Машина добавлена успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при добавлении машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.1 – Скрипт создания процедуры add\_car

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_car( |
| car\_id NUMERIC, |
| new\_label\_model\_id NUMERIC, |
| new\_year INT, |
| new\_mileage NUMERIC, |
| new\_enginetype VARCHAR(20), |
| new\_enginecapacity NUMERIC, |
| new\_power INT, |
| new\_description VARCHAR(1000), |
| new\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE CARS |
| SET label\_model\_id = new\_label\_model\_id, |
| year = new\_year, |
| mileage = new\_mileage, |
| enginetype = new\_enginetype, |
| enginecapacity = new\_enginecapacity, |
| power = new\_power, |
| description = new\_description, |
| status = new\_status |
| WHERE id = car\_id; |
|  |
| RAISE NOTICE 'Машина обновлена успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.2 – Скрипт создания процедуры update\_car

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_car(car\_id NUMERIC, new\_status BOOLEAN) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE CARS |
| SET status = new\_status |
| WHERE id = car\_id; |
|  |
| RAISE NOTICE 'Статус машины обновлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении статуса машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б3 – Скрипт создания процедуры delete\_car

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_sparepart( |
| new\_label VARCHAR(50), |
| new\_label\_model\_id NUMERIC, |
| new\_description VARCHAR(1000), |
| new\_quantity INT, |
| new\_price MONEY, |
| new\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO SPAREPARTS (label, label\_model\_id, description, quantity, price, status) |
| VALUES (new\_label, new\_label\_model\_id, new\_description, new\_quantity, new\_price, new\_status); |
|  |
| RAISE NOTICE 'Запчасть добавлена успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при добавлении запчасти: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.4 – Скрипт создания процедуры add\_sparepart

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_sparepart( |
| sparepart\_id NUMERIC, |
| new\_label VARCHAR(50), |
| new\_label\_model\_id NUMERIC, |
| new\_description VARCHAR(1000), |
| new\_quantity INT, |
| new\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE SPAREPARTS |
| SET label = new\_label, |
| label\_model\_id = new\_label\_model\_id, |
| description = new\_description, |
| quantity = new\_quantity, |
| status = new\_status |
| WHERE id = update\_sparepart.sparepart\_id; |
|  |
| RAISE NOTICE 'Запчасть обновлена успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении запчасти: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.5 – Скрипт создания процедуры update\_sparepart

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_sparepart(sparepart\_id NUMERIC, new\_quantity INT, new\_status BOOLEAN) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE SPAREPARTS |
| SET status = new\_status, quantity = new\_quantity |
| WHERE id = delete\_sparepart.sparepart\_id; |
|  |
| RAISE NOTICE 'Статус запчасти обновлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении статуса запчасти: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |
| Листинг Б.6 – Скрипт создания процедуры delete\_sparepart |
| CREATE OR REPLACE PROCEDURE add\_customer( |
| new\_secondname VARCHAR(20), |
| new\_firstname VARCHAR(20), |
| new\_thirdname VARCHAR(20), |
| new\_mail VARCHAR(40), |
| new\_phone VARCHAR(16), |
| new\_country VARCHAR(50), |
| new\_address VARCHAR(100), |
| new\_requisites VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO CUSTOMER ( secondname, firstname, thirdname, mail, phone, country, address, requisites) |
| VALUES (new\_secondname, new\_firstname, new\_thirdname, new\_mail, new\_phone, new\_country, new\_address, new\_requisites); |
|  |
| RAISE NOTICE 'Покупатель добавлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при добавлении покупателя: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.7 – Скрипт создания процедуры add\_customer

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_customer( |
| customer\_id NUMERIC, |
| new\_secondname VARCHAR(20), |
| new\_firstname VARCHAR(20), |
| new\_thirdname VARCHAR(20), |
| new\_mail VARCHAR(40), |
| new\_phone VARCHAR(16), |
| new\_country VARCHAR(50), |
| new\_address VARCHAR(100), |
| new\_requisites VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE CUSTOMER |
| SET id = customer\_id, |
| secondname = new\_secondname, |
| firstname = new\_firstname, |
| thirdname = new\_thirdname, |
| mail = new\_mail, |
| phone = new\_phone, |
| country = new\_country, |
| address = new\_address, |
| requisites = new\_requisites |
| WHERE id = update\_customer.customer\_id; |
|  |
| RAISE NOTICE 'Покупатель обновлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении покупателя: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.8 – Скрипт создания процедуры update\_customer

|  |
| --- |
| CREATE OR REPLACE PROCEDURE change\_status\_ordercar(ordercar\_id NUMERIC, new\_status BOOLEAN) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE ORDERCAR |
| SET status = new\_status |
| WHERE id = ordercar\_id; |
|  |
| RAISE NOTICE 'Статус заказа машины обновлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении статуса заказа машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.9 – Скрипт создания процедуры change\_status\_ordercar

|  |
| --- |
| CREATE OR REPLACE PROCEDURE change\_status\_orderspareparts(ordersparepart\_id NUMERIC, new\_status BOOLEAN) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE ORDERSPAREPARTS |
| SET status = new\_status |
| WHERE id = ordersparepart\_id; |
|  |
| RAISE NOTICE 'Статус заказа запчасти обновлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при обновлении статуса заказа машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.10 – Скрипт создания процедуры change\_status\_orderspareparts

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_ordercar( |
| order\_customer\_id INTEGER, |
| order\_manager\_id INTEGER, |
| order\_date DATE, |
| order\_car\_id INTEGER, |
| order\_status BOOLEAN, |
| order\_comment VARCHAR(1000) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO ORDERCAR (CUSTOMER\_ID, MANAGER\_ID, DATE, CAR\_ID, STATUS, COMMENT) |
| VALUES (order\_customer\_id, order\_manager\_id, order\_date, order\_car\_id, order\_status, order\_comment); |
| RAISE NOTICE 'Заказ машины создан успешно.'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при создании заказа машины: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.11 – Скрипт создания процедуры add\_ordercar

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_ordersparepart( |
| order\_customer\_id INTEGER, |
| order\_sparepart\_id INTEGER, |
| order\_date DATE, |
| order\_quantity INT, |
| order\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO ORDERSPAREPARTS (CUSTOMER\_ID, SPAREPART\_ID, DATE, QUANTITY, STATUS) |
| VALUES (order\_customer\_id, order\_sparepart\_id, order\_date, order\_quantity, order\_status); |
| RAISE NOTICE 'Заказ запчасти создан успешно.'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при создании заказа запчасти: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.12 – Скрипт создания процедуры add\_ordersparepart

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_review( |
| cust\_id INT, |
| review\_date DATE, |
| review\_rating INT, |
| review\_text VARCHAR(1000) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO REVIEWS (CUSTOMER\_ID, DATE, RATING, REVIEW) |
| VALUES (cust\_id, review\_date, review\_rating, review\_text); |
| RAISE NOTICE 'Отзыв оставлен успешно'; |
| EXCEPTION |
| WHEN OTHERS THEN |
| RAISE EXCEPTION 'Ошибка при добавлении отзыва: %', SQLERRM; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.13 – Скрипт создания процедуры add\_review

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_servicesheet( |
| p\_customer\_id NUMERIC, |
| p\_mechanic\_id NUMERIC, |
| p\_date DATE, |
| p\_label\_model\_id NUMERIC, |
| p\_problem\_description VARCHAR(1000), |
| p\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO SERVICESHEET (CUSTOMER\_ID, MECHANIC\_ID, DATE, LABEL\_MODEL\_ID, PROBLEMDESCRIPTION, STATUS) |
| VALUES (p\_customer\_id, p\_mechanic\_id, p\_date, p\_label\_model\_id, p\_problem\_description, p\_status); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.14 – Скрипт создания процедуры add\_servicesheet

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_service\_sheet( |
| service\_sheet\_id INT, |
| service\_sheet\_description VARCHAR(1000), |
| service\_sheet\_price MONEY, |
| service\_sheet\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE SERVICESHEET |
| SET |
| PROBLEMDESCRIPTION = service\_sheet\_description, |
| PRICE = service\_sheet\_price, |
| STATUS = service\_sheet\_status |
| WHERE ID = service\_sheet\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.15 – Скрипт создания процедуры update\_service\_sheet

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_sparepart\_quantity(part\_id INT, reduce\_quantity INT) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| IF NOT EXISTS (SELECT 1 FROM SPAREPARTS WHERE ID = part\_id) THEN |
| RAISE EXCEPTION 'Запчасть с ID % не существует', part\_id; |
| END IF; |
|  |
| IF (SELECT QUANTITY FROM SPAREPARTS WHERE ID = part\_id) < reduce\_quantity THEN |
| RAISE EXCEPTION 'Недостаточное количество для запчасти с ID %', part\_id; |
| END IF; |
|  |
| UPDATE SPAREPARTS |
| SET |
| QUANTITY = QUANTITY - reduce\_quantity, |
| STATUS = CASE WHEN QUANTITY - reduce\_quantity = 0 THEN FALSE ELSE STATUS END |
| WHERE ID = part\_id; |
|  |
| RAISE NOTICE 'Количество для запчасти с ID % уменьшено на %', part\_id, reduce\_quantity; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.16 – Скрипт создания процедуры update\_sparepart\_quantity

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_manager( |
| new\_secondname VARCHAR(20), |
| new\_firstname VARCHAR(20), |
| new\_thirdname VARCHAR(20), |
| new\_mail VARCHAR(40), |
| new\_phone VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO manager (secondname, firstname, thirdname, mail, phone) |
| VALUES (new\_secondname, new\_firstname, new\_thirdname, new\_mail, new\_phone); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.17 – Скрипт создания процедуры add\_manager

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_manager(manager\_id INT) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| DELETE FROM MANAGER |
| WHERE ID = manager\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.18 – Скрипт создания процедуры delete\_manager

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_mechanic( |
| new\_secondname VARCHAR(20), |
| new\_firstname VARCHAR(20), |
| new\_thirdname VARCHAR(20), |
| new\_mail VARCHAR(40), |
| new\_phone VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| INSERT INTO mechanic (secondname, firstname, thirdname, mail, phone) |
| VALUES (new\_secondname, new\_firstname, new\_thirdname, new\_mail, new\_phone); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.19 – Скрипт создания процедуры add\_mechanic

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_mechanic(mechanic\_id INT) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| DELETE FROM MECHANIC |
| WHERE ID = mechanic\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.20 – Скрипт создания процедуры delete\_mechanic

|  |
| --- |
| CREATE OR REPLACE PROCEDURE change\_car\_price(car\_id INT, car\_price MONEY) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE cars SET price = car\_price WHERE id = car\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.21 – Скрипт создания процедуры change\_car\_price

|  |
| --- |
| CREATE OR REPLACE PROCEDURE change\_sparepart\_price(sparepart\_id INT, sparepart\_price MONEY) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| UPDATE spareparts SET price = sparepart\_price WHERE id = sparepart\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.22 – Скрипт создания процедуры change\_sparepart\_price

|  |
| --- |
| CREATE OR REPLACE PROCEDURE add\_review\_from\_json() AS $$ |
| BEGIN |
| CREATE TEMP TABLE temp\_reviews ( |
| data JSONB |
| ); |
|  |
| COPY temp\_reviews(data) FROM 'D:/reviews.json'; |
|  |
| INSERT INTO reviews (customer\_id, date, rating, review) |
| SELECT |
| (data->>'CUSTOMER\_ID')::INT, |
| (data->>'DATE')::DATE, |
| (data->>'RATING')::INT, |
| (data->>'REVIEW')::VARCHAR(1000) |
| FROM temp\_reviews; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг Б.23 – Скрипт создания процедуры add\_review\_from\_json

Приложение В

Листинги функций

|  |
| --- |
| CREATE OR REPLACE FUNCTION calculate\_total\_price(start\_date DATE, end\_date DATE) |
| RETURNS MONEY |
| SECURITY DEFINER |
| AS $$ |
| DECLARE |
| total\_price MONEY := 0; |
| BEGIN |
| IF start\_date IS NULL AND end\_date IS NULL THEN |
| SELECT SUM(c.PRICE) INTO total\_price |
| FROM ORDERCAR o |
| JOIN CARS c ON o.CAR\_ID = c.ID |
| WHERE o.STATUS = true; |
| ELSIF start\_date IS NULL THEN |
| SELECT SUM(c.PRICE) INTO total\_price |
| FROM ORDERCAR o |
| JOIN CARS c ON o.CAR\_ID = c.ID |
| WHERE o.DATE <= end\_date |
| AND o.STATUS = true; |
| ELSIF end\_date IS NULL THEN |
| SELECT SUM(c.PRICE) INTO total\_price |
| FROM ORDERCAR o |
| JOIN CARS c ON o.CAR\_ID = c.ID |
| WHERE o.DATE >= start\_date |
| AND o.STATUS = true; |
| ELSE |
| SELECT SUM(c.PRICE) INTO total\_price |
| FROM ORDERCAR o |
| JOIN CARS c ON o.CAR\_ID = c.ID |
| WHERE o.DATE BETWEEN start\_date AND end\_date |
| AND o.STATUS = true; |
| END IF; |
|  |
| RETURN total\_price; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.1 – Скрипт создания функции calculate\_total\_price

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_available\_cars() |
| RETURNS TABLE ( |
| car\_id INT, |
| label VARCHAR(30), |
| model VARCHAR(30), |
| year INT, |
| mileage NUMERIC, |
| engine\_type VARCHAR(20), |
| engine\_capacity NUMERIC, |
| power INT, |
| price MONEY, |
| description VARCHAR(1000) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| DECLARE |
| total\_cars\_count INT; |
| BEGIN |
| SELECT COUNT(\*) INTO total\_cars\_count |
| FROM cars |
| WHERE status = true; |
|  |
| RAISE NOTICE 'Доступно машин: %', total\_cars\_count; |
|  |
| RETURN QUERY |
| SELECT |
| c.id AS car\_id, |
| l.label AS label, |
| m.model AS model, |
| c.year, |
| c.mileage, |
| c.enginetype AS engine\_type, |
| c.enginecapacity AS engine\_capacity, |
| c.power, |
| c.price, |
| c.description |
| FROM |
| cars c |
| INNER JOIN label\_model lm ON c.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.status = true; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.2 – Скрипт создания функции get\_available\_cars

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_available\_customers() |
| RETURNS TABLE ( |
| customer\_id INT, |
| secondname VARCHAR(20), |
| firstname VARCHAR(20), |
| thirdname VARCHAR(20), |
| mail VARCHAR(40), |
| phone VARCHAR(16), |
| country VARCHAR(50), |
| address VARCHAR(100) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| DECLARE |
| total\_customers\_count INT; |
| BEGIN |
| SELECT COUNT(\*) |
| INTO total\_customers\_count |
| FROM customer; |
| RAISE NOTICE 'Всего клиентов: %', total\_customers\_count; |
|  |
| RETURN QUERY |
| SELECT |
| c.id AS customer\_id, |
| c.secondname, |
| c.firstname, |
| c.thirdname, |
| c.mail, |
| c.phone, |
| c.country, |
| c.address |
| FROM customer c; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.3 – Скрипт создания функции get\_available\_customers

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_available\_spareparts() |
| RETURNS TABLE ( |
| sparepart\_id INT, |
| label\_sparepart VARCHAR(50), |
| label VARCHAR(30), |
| model VARCHAR(30), |
| description VARCHAR(1000), |
| price MONEY, |
| quantity INT |
| ) |
| SECURITY DEFINER |
| AS $$ |
| DECLARE |
| total\_spareparts\_count INT; |
| BEGIN |
| SELECT COUNT(\*) INTO total\_spareparts\_count |
| FROM spareparts |
| WHERE status = true; |
|  |
| RAISE NOTICE 'Доступно запчастей: %', total\_spareparts\_count; |
|  |
| RETURN QUERY |
| SELECT |
| s.id AS sparepart\_id, |
| s.label AS name, |
| l.label, |
| m.model, |
| s.description, |
| s.price, |
| s.quantity |
| FROM |
| spareparts s |
| INNER JOIN label\_model lm ON s.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| s.status = true; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.4 – Скрипт создания функции get\_available\_spareparts

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_label\_model() |
| RETURNS TABLE ( |
| label\_model\_id INT, |
| label\_model\_label VARCHAR(30), |
| label\_model\_model VARCHAR(30) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT lm.id AS label\_model\_id, l.label, m.model |
| FROM label\_model lm |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| ORDER BY l.label, m.model; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.5 – Скрипт создания функции get\_all\_label\_model

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_car() |
| RETURNS TABLE ( |
| car\_id INT, |
| label\_model\_label VARCHAR(30), |
| label\_model\_model VARCHAR(30), |
| car\_year INT, |
| car\_status BOOLEAN, |
| car\_price MONEY |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT c.id, l.label, m.model, c.year, c.status, c.price |
| FROM cars c |
| INNER JOIN label\_model lm ON c.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| ORDER BY l.label, m.model; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.6 – Скрипт создания функции get\_all\_car

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_customer() |
| RETURNS TABLE ( |
| customer\_id INT, |
| customer\_secondname VARCHAR(20), |
| customer\_firstname VARCHAR(20), |
| customer\_thirdname VARCHAR(20), |
| customer\_mail VARCHAR(40), |
| customer\_phone VARCHAR(16), |
| customer\_country VARCHAR(50), |
| customer\_address VARCHAR(100), |
| customer\_requisites VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT id, secondname, firstname, thirdname, mail, phone, country, address, requisites |
| FROM customer |
| ORDER BY secondname; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.7 – Скрипт создания функции get\_all\_customer

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_manager() |
| RETURNS TABLE ( |
| manager\_id INT, |
| manager\_secondname VARCHAR(20), |
| manager\_firstname VARCHAR(20), |
| manager\_thirdname VARCHAR(20) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT id, secondname, firstname, thirdname FROM manager; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.8 – Скрипт создания функции get\_all\_manager

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_orders\_cars\_by\_manager\_id(managers\_id INT) |
| RETURNS TABLE ( |
| order\_id INT, |
| secondname\_manager VARCHAR(20), |
| label\_car VARCHAR(30), |
| model\_car VARCHAR(30), |
| year\_car INT, |
| secondname\_customer VARCHAR(20), |
| status\_car BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT o.id, m.secondname, l.label, mo.model, c.year, mc.secondname, o.status |
| FROM ORDERCAR o |
| INNER JOIN MANAGER m ON o.manager\_id = m.id |
| INNER JOIN CARS c ON o.car\_id = c.id |
| INNER JOIN LABEL\_MODEL lm ON c.label\_model\_id = lm.id |
| INNER JOIN LABELS l ON lm.label\_id = l.id |
| INNER JOIN MODEL mo ON lm.model\_id = mo.id |
| INNER JOIN CUSTOMER mc ON o.customer\_id = mc.id |
| WHERE o.manager\_id = managers\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.9 – Скрипт создания функции get\_orders\_cars\_by\_manager\_id

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_orders\_spare\_parts\_by\_customer\_id(customers\_id INT) |
| RETURNS TABLE ( |
| order\_id INT, |
| secondname\_customer VARCHAR(20), |
| sparepart\_label VARCHAR(50), |
| label VARCHAR(30), |
| model VARCHAR(30), |
| status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT os.id, c.secondname, sp.label, l.label, m.model, os.status |
| FROM ORDERSPAREPARTS os |
| INNER JOIN CUSTOMER c ON os.customer\_id = c.id |
| INNER JOIN SPAREPARTS sp ON os.sparepart\_id = sp.id |
| INNER JOIN LABEL\_MODEL lm ON sp.label\_model\_id = lm.id |
| INNER JOIN LABELS l ON lm.label\_id = l.id |
| INNER JOIN MODEL m ON lm.model\_id = m.id |
| WHERE c.id = customers\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.10 – Скрипт создания функции get\_orders\_spare\_parts\_by\_customer\_id

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_sparepart() |
| RETURNS TABLE ( |
| sparepart\_id INT, |
| sparepart\_label VARCHAR(50), |
| label\_model\_label VARCHAR(30), |
| label\_model\_model VARCHAR(30), |
| sparepart\_quantity INT, |
| sparepart\_status BOOLEAN, |
| sparepart\_description VARCHAR(1000), |
| label\_model\_id INT, |
| sparepart\_price MONEY |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT s.id, s.label, l.label, m.model, s.quantity, s.status, s.description, s.label\_model\_id, s.price |
| FROM spareparts s |
| INNER JOIN label\_model lm ON s.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| ORDER BY s.label; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.11 – Скрипт создания функции get\_all\_sparepart

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_manager() |
| RETURNS TABLE ( |
| manager\_id INT, |
| manager\_secondname VARCHAR(20), |
| manager\_firstname VARCHAR(20), |
| manager\_thirdname VARCHAR(20) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT id, secondname, firstname, thirdname FROM manager; |
| END; |
| $$ LANGUAGE plpgsql; |
|  |
| CREATE OR REPLACE FUNCTION get\_customer\_by\_phone(phone\_number VARCHAR(16)) |
| RETURNS TABLE ( |
| id INT, |
| secondname VARCHAR(20), |
| firstname VARCHAR(20), |
| thirdname VARCHAR(20), |
| mail VARCHAR(40), |
| phone VARCHAR(16), |
| country VARCHAR(50), |
| address VARCHAR(100), |
| requisites VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT \* |
| FROM CUSTOMER |
| WHERE CUSTOMER.PHONE = phone\_number; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.12 – Скрипт создания функции get\_all\_manager

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_quantity\_spareparts(order\_id INT) |
| RETURNS TABLE ( |
| available\_quantity INT |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT s.quantity FROM SPAREPARTS s WHERE s.id = order\_id; |
| END; |
| $$ LANGUAGE plpgsql; |
|  |
| CREATE OR REPLACE FUNCTION get\_all\_mechanic() |
| RETURNS TABLE ( |
| mechanic\_id INT, |
| mechanic\_secondname VARCHAR(20), |
| mechanic\_firstname VARCHAR(20), |
| mechanic\_thirdname VARCHAR(20) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT id, secondname, firstname, thirdname FROM mechanic; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.13 – Скрипт создания функции check\_quantity\_sparepart

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_date\_status(cust\_id NUMERIC) |
| RETURNS TABLE ( |
| label VARCHAR(30), |
| model VARCHAR(30), |
| year INT, |
| price MONEY, |
| date DATE, |
| status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| l.label, |
| m.model, |
| c.year, |
| c.price, |
| o.date, |
| o.status |
| FROM |
| ordercar o |
| INNER JOIN cars c ON o.car\_id = c.id |
| INNER JOIN label\_model lm ON c.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| o.customer\_id = cust\_id; |
| END; |
| $$ LANGUAGE plpgsql; |
| Листинг В.14 – Скрипт создания функции get\_date\_status |
| CREATE OR REPLACE FUNCTION get\_date\_status\_spare(cust\_id NUMERIC) |
| RETURNS TABLE ( |
| label\_sparepart VARCHAR(50), |
| label VARCHAR(50), |
| model VARCHAR(30), |
| quantity INT, |
| price MONEY, |
| date DATE, |
| status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| sp.label, |
| l.label, |
| m.model, |
| os.quantity, |
| sp.price \* os.quantity AS price, |
| os.date, |
| os.status |
| FROM |
| orderspareparts os |
| INNER JOIN spareparts sp ON os.sparepart\_id = sp.id |
| INNER JOIN label\_model lm ON sp.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| os.customer\_id = cust\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.15 – Скрипт создания функции get\_date\_status\_spare

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_history\_orders\_cars(cust\_id NUMERIC) |
| RETURNS TABLE ( |
| label VARCHAR(30), |
| model VARCHAR(30), |
| year INT, |
| price MONEY, |
| date DATE |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| l.label, |
| m.model, |
| c.year, |
| c.price, |
| o.date |
| FROM |
| ordercar o |
| INNER JOIN cars c ON o.car\_id = c.id |
| INNER JOIN label\_model lm ON c.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| o.customer\_id = cust\_id AND o.status = true; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.16 – Скрипт создания функции get\_history\_orders\_car

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_history\_orders\_spareparts(cust\_id NUMERIC) |
| RETURNS TABLE ( |
| label\_sparepart VARCHAR(50), |
| label VARCHAR(50), |
| model VARCHAR(30), |
| quantity INT, |
| price MONEY, |
| date DATE |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| sp.label, |
| l.label, |
| m.model, |
| os.quantity, |
| sp.price \* os.quantity AS price, |
| os.date |
| FROM |
| orderspareparts os |
| INNER JOIN spareparts sp ON os.sparepart\_id = sp.id |
| INNER JOIN label\_model lm ON sp.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE |
| os.customer\_id = cust\_id AND os.status = true; |
| END; |
| $$ LANGUAGE plpgsql; |
| Листинг В.17 – Скрипт создания функции get\_history\_orders\_spareparts |
| CREATE OR REPLACE FUNCTION get\_servicesheet(cust\_id NUMERIC) |
| RETURNS TABLE ( |
| label VARCHAR(30), |
| model VARCHAR(30), |
| mechanic\_secondname VARCHAR(20), |
| mechanic\_firstname VARCHAR(20), |
| mechanic\_thirdname VARCHAR(20), |
| date DATE, |
| problem\_description VARCHAR(1000), |
| price MONEY, |
| status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT l.label, |
| m.model, |
| mech.secondname, |
| mech.firstname, |
| mech.thirdname, |
| s.date, |
| s.problemdescription, |
| s.price, |
| s.status |
| FROM SERVICESHEET s |
| INNER JOIN LABEL\_MODEL lm ON s.label\_model\_id = lm.id |
| INNER JOIN LABELS l ON lm.label\_id = l.id |
| INNER JOIN MODEL m ON lm.model\_id = m.id |
| INNER JOIN MECHANIC mech ON s.mechanic\_id = mech.id |
| WHERE s.customer\_id = cust\_id; |
| END; |
| $$ |
| LANGUAGE plpgsql; |

Листинг В.18 – Скрипт создания функции get\_servicesheet

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_reviews() |
| RETURNS TABLE ( |
| review\_id INT, |
| customer\_firstname VARCHAR(20), |
| review\_date DATE, |
| review\_rating INT, |
| review\_review VARCHAR(1000) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT r.id, |
| c.firstname, |
| r.date, |
| r.rating, |
| r.review |
| FROM reviews r |
| INNER JOIN customer c ON r.customer\_id = c.id |
| ORDER BY r.date DESC |
| LIMIT 30; |
| END; |
| $$ |
| LANGUAGE plpgsql; |

Листинг В.19 – Скрипт создания функции get\_all\_reviews

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_labels() |
| RETURNS TABLE ( |
| label\_label\_id INT, |
| label\_label VARCHAR(30), |
| label\_country VARCHAR(50) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT id, label, country FROM labels; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.20 – Скрипт создания функции get\_labels

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_all\_models\_by\_label(l\_id INT) |
| RETURNS TABLE ( |
| id INT, |
| model\_model\_id INT, |
| model\_model VARCHAR(30) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT lm.id, m.id, m.model |
| FROM LABEL\_MODEL lm |
| JOIN MODEL m ON lm.MODEL\_ID = m.ID |
| WHERE lm.LABEL\_ID = l\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.21 – Скрипт создания функции get\_all\_models\_by\_label

|  |
| --- |
| CREATE OR REPLACE FUNCTION search\_cars( |
| search\_year INT, |
| search\_label\_id INT, |
| search\_label\_model\_id INT |
| ) RETURNS TABLE ( |
| p\_id INT, |
| p\_label VARCHAR(30), |
| p\_model VARCHAR(30), |
| p\_year INT, |
| p\_mileage NUMERIC, |
| p\_engine\_type VARCHAR(20), |
| p\_engine\_capacity NUMERIC, |
| p\_power INT, |
| p\_price MONEY, |
| p\_description VARCHAR(1000), |
| p\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| IF search\_year IS NOT NULL AND search\_label\_id IS NOT NULL AND search\_label\_model\_id IS NOT NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.year = search\_year AND |
| c.label\_model\_id = search\_label\_model\_id AND |
| lm.label\_id = search\_label\_id; |
| ELSIF search\_year IS NOT NULL AND search\_label\_id IS NOT NULL AND search\_label\_model\_id IS NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.year = search\_year AND |
| lm.label\_id = search\_label\_id; |
| ELSIF search\_year IS NULL AND search\_label\_id IS NOT NULL AND search\_label\_model\_id IS NOT NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.label\_model\_id = search\_label\_model\_id AND |
| lm.label\_id = search\_label\_id; |
| ELSIF search\_year IS NOT NULL AND search\_label\_id IS NULL AND search\_label\_model\_id IS NOT NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.year = search\_year AND |
| c.label\_model\_id = search\_label\_model\_id; |
| ELSIF search\_year IS NOT NULL AND search\_label\_id IS NULL AND search\_label\_model\_id IS NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| c.year = search\_year; |
| ELSIF search\_year IS NULL AND search\_label\_id IS NOT NULL AND search\_label\_model\_id IS NULL THEN |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| lm.label\_id = search\_label\_id; |
| ELSE |
| RETURN QUERY |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| c.description, |
| c.status |
| FROM |
| cars c |
| JOIN label\_model lm ON c.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id; |
| END IF; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.22 – Скрипт создания функции search\_cars

|  |
| --- |
| CREATE OR REPLACE FUNCTION search\_spareparts( |
| search\_label\_id INT, |
| search\_label\_model\_id INT DEFAULT NULL |
| ) RETURNS TABLE ( |
| p\_id INT, |
| p\_label VARCHAR(30), |
| p\_llabel VARCHAR(30), |
| p\_model VARCHAR(30), |
| p\_description VARCHAR(1000), |
| p\_quantity INT, |
| p\_price MONEY, |
| p\_status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| IF search\_label\_model\_id IS NOT NULL THEN |
| RETURN QUERY |
| SELECT |
| s.id, |
| s.label, |
| l.label, |
| m.model, |
| s.description, |
| s.quantity, |
| s.price, |
| s.status |
| FROM |
| spareparts s |
| JOIN label\_model lm ON s.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| lm.label\_id = search\_label\_id AND |
| lm.model\_id = search\_label\_model\_id; |
| ELSE |
| RETURN QUERY |
| SELECT |
| s.id, |
| s.label, |
| l.label, |
| m.model, |
| s.description, |
| s.quantity, |
| s.price, |
| s.status |
| FROM |
| spareparts s |
| JOIN label\_model lm ON s.label\_model\_id = lm.id |
| JOIN labels l ON lm.label\_id = l.id |
| JOIN model m ON lm.model\_id = m.id |
| WHERE |
| lm.label\_id = search\_label\_id; |
| END IF; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.23 – Скрипт создания функции search\_spareparts

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_service\_sheet(mech\_id INT) |
| RETURNS TABLE ( |
| id INT, |
| label VARCHAR(30), |
| model VARCHAR(30), |
| customer\_id INT, |
| customer\_secondname VARCHAR(20), |
| customer\_firstname VARCHAR(20), |
| price MONEY, |
| date DATE, |
| mechanic\_secondname VARCHAR(20), |
| mechanic\_firstname VARCHAR(20), |
| description VARCHAR(1000), |
| status BOOLEAN |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| ss.id, |
| lb.LABEL, |
| md.MODEL, |
| c.id, |
| c.SECONDNAME, |
| c.FIRSTNAME, |
| ss.PRICE, |
| ss.DATE, |
| m.SECONDNAME, |
| m.FIRSTNAME, |
| ss.problemdescription, |
| ss.STATUS |
| FROM |
| SERVICESHEET ss |
| JOIN CUSTOMER c ON ss.CUSTOMER\_ID = c.ID |
| JOIN MECHANIC m ON ss.MECHANIC\_ID = m.ID |
| JOIN LABEL\_MODEL lm ON ss.LABEL\_MODEL\_ID = lm.ID |
| JOIN LABELS lb ON lm.LABEL\_ID = lb.ID |
| JOIN MODEL md ON lm.MODEL\_ID = md.ID |
| WHERE |
| ss.MECHANIC\_ID = mech\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.24 – Скрипт создания функции get\_service\_sheet

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_customer\_full\_names(mech\_id INT) |
| RETURNS TABLE ( |
| id INT, |
| secondname VARCHAR(20), |
| firstname VARCHAR(20), |
| thirdnmae VARCHAR(20) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT |
| c.id, |
| c.secondname, |
| c.firstname, |
| c.thirdname |
| FROM servicesheet ss |
| INNER JOIN customer c ON ss.customer\_id = c.id |
| WHERE ss.mechanic\_id = mech\_id; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.25 – Скрипт создания функции get\_customer\_full\_names

|  |
| --- |
| CREATE OR REPLACE FUNCTION get\_mechanic\_by\_phone(phone\_number VARCHAR) |
| RETURNS TABLE ( |
| mech\_id INT, |
| secondname VARCHAR(20), |
| firstname VARCHAR(20), |
| thirdname VARCHAR(20), |
| mail VARCHAR(40), |
| phone VARCHAR(16) |
| ) |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT m.id, m.secondname, m.firstname, m.thirdname, m.mail, m.phone |
| FROM mechanic m |
| WHERE m.phone = phone\_number; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.26 – Скрипт создания функции get\_mechanic\_by\_phone

|  |
| --- |
| CREATE OR REPLACE FUNCTION create\_report\_car() |
| RETURNS TRIGGER |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| DELETE FROM reports\_car; |
|  |
| INSERT INTO reports\_car (car\_id, car\_label, car\_model, car\_year, |
| car\_mileage, car\_enginetype, car\_enginecapacity, |
| car\_power, car\_price, total\_sales, total\_price) |
| SELECT |
| c.id, |
| l.label, |
| m.model, |
| c.year, |
| c.mileage, |
| c.enginetype, |
| c.enginecapacity, |
| c.power, |
| c.price, |
| COUNT(o.id) AS total\_orders, |
| SUM(c.price) AS total\_revenue |
| FROM ordercar o |
| INNER JOIN cars c ON o.car\_id = c.id |
| INNER JOIN label\_model lm ON c.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE o.status = true |
| GROUP BY |
| c.id, l.label, m.model; |
|  |
| RETURN NULL; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.27 – Скрипт создания функции create\_report\_car

|  |
| --- |
| CREATE OR REPLACE FUNCTION generate\_sales\_report\_car() |
| RETURNS VOID AS $$ |
| BEGIN |
| EXECUTE format('COPY ( |
| SELECT to\_json(reports\_car) FROM reports\_car |
| ) TO %L', 'D:\reports\_car.json'); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.28 – Скрипт создания функции generate\_sales\_report\_car

|  |
| --- |
| CREATE OR REPLACE FUNCTION create\_report\_sparepart() |
| RETURNS TRIGGER |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| DELETE FROM reports\_sparepart; |
|  |
| INSERT INTO reports\_sparepart (sparepart\_id, sparepart\_label, label, model, |
| sparepart\_quantity, sparepart\_price, total\_sales, |
| total\_price) |
| SELECT |
| s.id, |
| s.label, |
| l.label, |
| m.model, |
| o.quantity, |
| s.price \* o.quantity AS price, |
| COUNT(o.id) AS total\_orders, |
| SUM(s.price) AS total\_revenue |
| FROM orderspareparts o |
| INNER JOIN spareparts s ON s.id = o.sparepart\_id |
| INNER JOIN label\_model lm ON s.label\_model\_id = lm.id |
| INNER JOIN labels l ON lm.label\_id = l.id |
| INNER JOIN model m ON lm.model\_id = m.id |
| WHERE o.status = true |
| GROUP BY |
| s.id, s.label, l.label, m.model, o.quantity, s.price; |
|  |
| RETURN NULL; |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.29 – Скрипт создания функции create\_report\_sparepart

|  |
| --- |
| CREATE OR REPLACE FUNCTION generate\_sales\_report\_sparepart() |
| RETURNS VOID AS $$ |
| BEGIN |
| EXECUTE format('COPY ( |
| SELECT to\_json(reports\_sparepart) FROM reports\_sparepart |
| ) TO %L', 'D:\reports\_sparepart.json'); |
| END; |
| $$ LANGUAGE plpgsql; |

Листинг В.30 – Скрипт создания функции generate\_sales\_report\_sparepart

Приложение Г

Листинги триггеров

|  |
| --- |
| CREATE TRIGGER update\_reports\_car\_trigger |
| AFTER INSERT ON ordercar |
| FOR EACH STATEMENT |
| EXECUTE FUNCTION create\_report\_car(); |

Листинг Г.1 – Скрипт создания триггера update\_reports\_car\_trigger

|  |
| --- |
| CREATE TRIGGER update\_reports\_sparepart\_trigger |
| AFTER INSERT ON orderspareparts |
| FOR EACH STATEMENT |
| EXECUTE FUNCTION create\_report\_sparepart(); |

Листинг Г.2 – Скрипт создания триггера update\_reports\_sparepart\_trigger