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 салона автомобилей с использованием средств мониторинга за состоянием базы данных»

**Исполнитель**

студент 2 курса 7 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов М. О.

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** 6](#_Toc167246299)

[1. Аналитический обзор литературы по теме проекта 7](#_Toc167246300)

[1.1 Сравнительный анализ теоретических и эвристических методов решения поставленных задач 7](#_Toc167246301)

[1.2 Аналитический обзор аналогов 7](#_Toc167246302)

[1.2.1 Аналог «JetCar» 7](#_Toc167246303)

[Как и у любого подобного сайта, у автомобильного салона есть свои недостатки. Один из таких недостатков связан с ошибками при просмотре каталога автомобилей. Например, пользователи могут столкнуться с проблемами отображения фотографий из-за ошибок в данных базы данных. 9](#_Toc167246304)

[1.2.2 Аналог «Royal Motors» 9](#_Toc167246305)

[1.3 Изучение требований, определение вариантов использования 10](#_Toc167246306)

[1.3.1 Определение основных функциональных требований к базе данных 10](#_Toc167246307)

[1.3.2 Определение нефункциональных требований к базе данных 11](#_Toc167246308)

[1.3.3 Определение вариантов использования 11](#_Toc167246309)

[1.4 Вывод 12](#_Toc167246310)

[2. Разработка архитектуры проекта 13](#_Toc167246311)

[2.1 Обобщенная структура управления приложением 13](#_Toc167246312)

[2.2 Диаграммы UML, взаимосвязь всех компонентов 13](#_Toc167246313)

[2.3 Роли и пользователи 14](#_Toc167246314)

[2.4 Описание информационных объектов и ограничений целостности 15](#_Toc167246315)

[2.4.1 Таблица Customer 16](#_Toc167246316)

[2.4.2 Таблица Manager 16](#_Toc167246317)

[2.4.3 Таблица Mechanic 16](#_Toc167246318)

[2.4.4 Таблица CARS 17](#_Toc167246319)

[2.4.5 Таблица ORDERCAR 17](#_Toc167246320)

[2.4.6 Таблица SPAREPARTS 18](#_Toc167246321)

[2.4.7 Таблица ORDERSPAREPARTS 18](#_Toc167246322)

[2.4.8 Таблица SERVICESHEET 18](#_Toc167246323)

[2.4.9 Таблица REVIEWS 19](#_Toc167246324)

[2.4.10 Таблица LABELS 19](#_Toc167246325)

[2.4.11 Таблица MODEL 19](#_Toc167246326)

[2.4.12 Таблица LABEL\_MODEL 20](#_Toc167246327)

[2.5 Вывод 20](#_Toc167246328)

[3. Разработка объектов базы данных 21](#_Toc167246329)

[3.1 Создание необходимых объектов 21](#_Toc167246330)

[3.2 Функции 21](#_Toc167246331)

[3.3 Процедуры 23](#_Toc167246332)

[3.4 Триггеры 24](#_Toc167246333)

[3.5 Индексы 24](#_Toc167246334)

[3.6 Вывод 25](#_Toc167246335)

[4.Описание процедур экспорта и импорта. 26](#_Toc167246336)

[4.1 Процедура экспорта данных 26](#_Toc167246337)

[4.2 Процедура импорта данных 27](#_Toc167246338)

[4.3 Вывод 27](#_Toc167246339)

[5. Тестирование производительности базы данных. 28](#_Toc167246340)

[5.1 Заполнение таблицы 100000 строк 28](#_Toc167246341)

[5.2 Тестирование производительности по таблице REVIEWS. 28](#_Toc167246342)

[5.3 Вывод 32](#_Toc167246343)

[6. Описание технологии и ее применение в базе данных 33](#_Toc167246344)

[6.1 Технология «Разработка системы мониторинга за состоянием базы данных» 33](#_Toc167246345)

[6.2 Графическое средство для администрирования pgAdmin 4 33](#_Toc167246346)

[6.3 Представление pg\_stat\_statements 36](#_Toc167246347)

[6.4 Вывод 39](#_Toc167246348)

[7. Руководство пользователя 40](#_Toc167246349)

[7.1 Инструкция по эксплуатации для пользователя «User» 40](#_Toc167246350)

[7.2 Инструкция по эксплуатации для пользователя «Manager» 41](#_Toc167246351)

[7.3 Инструкция по эксплуатации для пользователя «Mechanic» 42](#_Toc167246352)

[7.4 Инструкция по эксплуатации для пользователя «Administrator» 43](#_Toc167246353)

[7.4 Вывод 44](#_Toc167246354)

[Заключение 45](#_Toc167246355)

[Список использованных литературных источников 46](#_Toc167246356)

[Приложение 47](#_Toc167246357)

[Приложение А 47](#_Toc167246358)

[Приложение Б 51](#_Toc167246359)

[Приложение В 60](#_Toc167246360)

[Приложение Г 85](#_Toc167246361)

Введение

Задачи проекта включают анализ литературы, изучение требований, определение вариантов использования базы данных, проектирование модели данных, описание информационных объектов и ограничений, создание необходимых элементов базы данных, импорт и экспорт данных, описание выбранной технологии, тестирование производительности и формирование выводов. Выбор правильной системы управления базами данных является сложной задачей из-за множества предложений на рынке, включая коммерческие СУБД и базы данных с открытым исходным кодом и коммерческой поддержкой.

База данных играет ключевую роль в автомобильном салоне, обеспечивая централизованное хранение всей необходимой информации, что позволяет быстро и точно получать данные о наличии автомобилей, их технических характеристиках, ценах и истории продаж. Это особенно важно для улучшения качества обслуживания клиентов, так как менеджеры салона могут оперативно предоставлять покупателям актуальную информацию и помогать им в выборе подходящего автомобиля. Кроме того, база данных помогает в управлении запасами, что позволяет оптимизировать процесс заказа и хранения автомобилей и запчастей.

Эффективная система управления базой данных также способствует более точному анализу продаж и поведения клиентов, что позволяет салону разрабатывать стратегические планы и улучшать маркетинговые кампании. Система мониторинга производительности базы данных обеспечивает своевременное выявление и устранение потенциальных проблем, что минимизирует время простоя и повышает надежность работы салона. В результате, хорошо спроектированная и реализованная база данных становится важным инструментом для повышения конкурентоспособности и успеха автомобильного салона на рынке.

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

В современном мире салоны автомобилей становятся все более популярными. Одной из главных задач таких салонов является надежная продажа и обслуживание автомобилей, чтобы удовлетворить потребности и ожидания клиентов.

Салоны автомобилей предлагают широкий ассортимент автомобилей различных марок и моделей. Клиенты могут выбрать автомобиль, соответствующий их предпочтениям и потребностям.

Кроме продажи автомобилей, салоны также предоставляют услуги по обслуживанию и ремонту. Механики проводят техническое обслуживание, диагностику и ремонт автомобилей. Салоны также предлагают запчасти, чтобы клиенты могли настроить свои автомобили под свои потребности и предпочтения.

1.2.1 Аналог «JetCar»

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

Таблица "Автомобили": содержит информацию о доступных автомобилях, такую как модель, марка, год выпуска, цена, характеристики и фотографии.

Таблица "Заказы": хранит информацию о заказах клиентов, включая дату заказа, детали автомобиля, связанные с заказом, цену, скидки и статус заказа.

Таблица "Менеджеры": содержит информацию о менеджерах, которые управляют различными аспектами работы салона.

Таблица "Клиенты": содержит информацию о клиентах, включая их персональные данные, контактную информацию, историю покупок и предпочтения.

Таблица "Отзывы": содержит информацию о отзывах, оставленных клиентами о своем опыте салона и его услугами.

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 «JetCar» [1], который представлен на рисунке 1.1.



Рисунок 1.1 – Сайт «JetCar»

Данный сайт позволяет из любой точки забронировать необходимый автомобиль, а также получить всю необходимую информацию о нем. Пример представлен на рисунке 1.2.

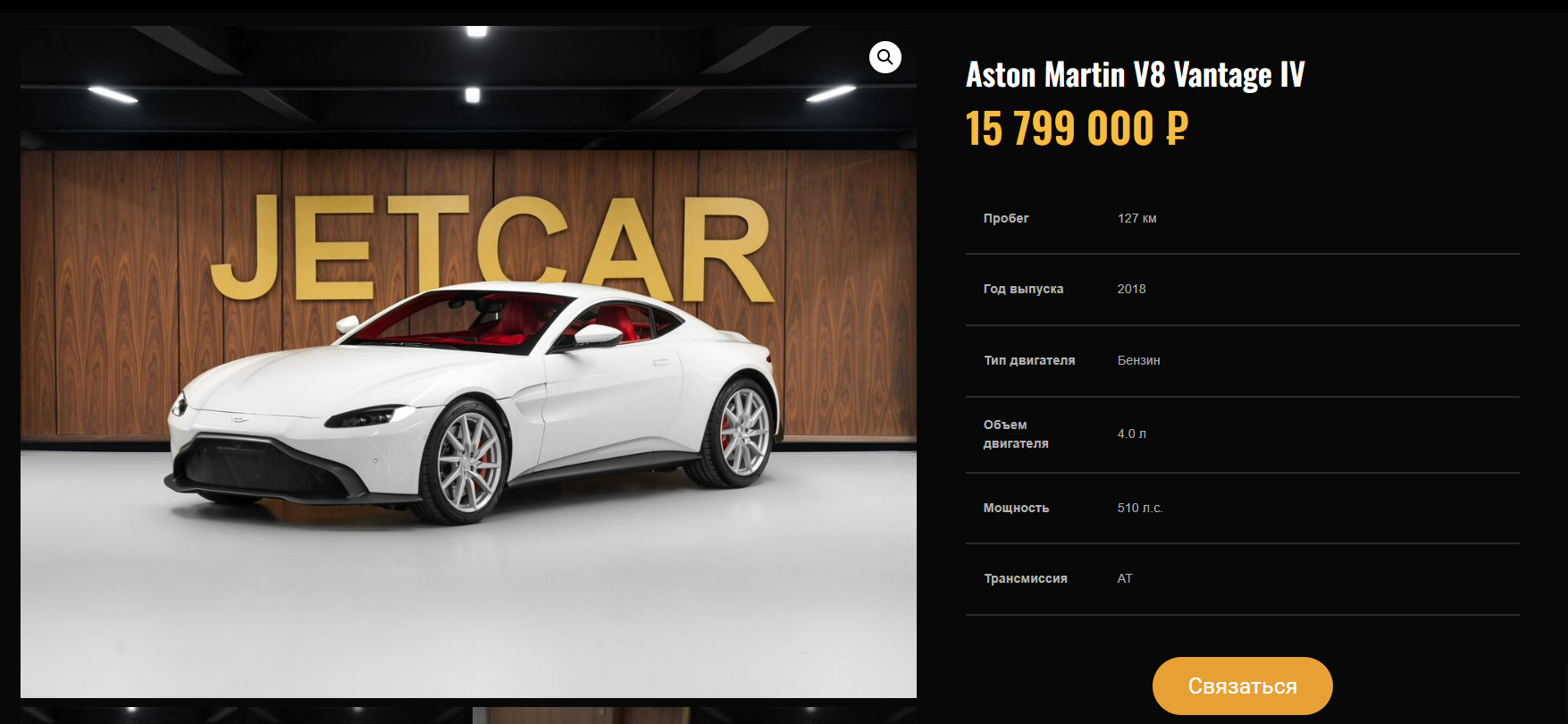


Рисунок 1.2 – Выбор автомобиля на сайте «JetCar»

Также сайт позволяет оставить отзыв. Пример на рисунке 1.3.



Рисунок 1.3 – Отзывы на сайте «JetCar»

Как и у любого подобного сайта, у автомобильного салона есть свои недостатки. Один из таких недостатков связан с ошибками при просмотре каталога автомобилей. Например, пользователи могут столкнуться с проблемами отображения фотографий из-за ошибок в данных базы данных.

1.2.2 Аналог «Royal Motors»

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



Рисунок 1.4 – Выбор автомобиля «Royal Motors»

Из минусов, можно отметить, также проблему отображения медиа файлов.

Плюсом данного аналога является предоставление обширной информации об автомобиле и простоту взаимодействия клиента с сайтом для заполнения персональных данных.

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

В ходе изучения требований было выявлено, что база данных салона автомобилей должна включать информацию об автомобилях, запчастях, менеджерах, клиентах, механиках, заказах, отзывах, записях на обслуживание. Были определены следующие варианты использования:

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

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 «JetCar» и «Royal Motors». Эти ресурсы представляют собой примеры успешного применения баз данных для управления автосалонами и обеспечения удобного доступа к информации для клиентов.

Анализ функционала сайтов «JetCar» и «Royal Motors» показал, что ключевыми элементами являются таблицы, содержащие информацию об автомобилях, заказах, менеджерах, клиентах и отзывах.

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.

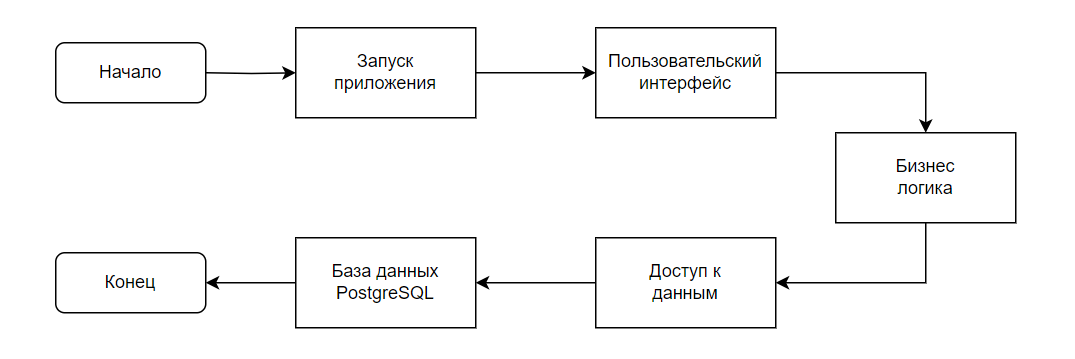


Рисунок 2.1 – Обобщенная структура управления приложением

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

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

2.2 Диаграммы UML, взаимосвязь всех компонентов

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

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

Сама диаграмма представлена на рисунке 2.2.

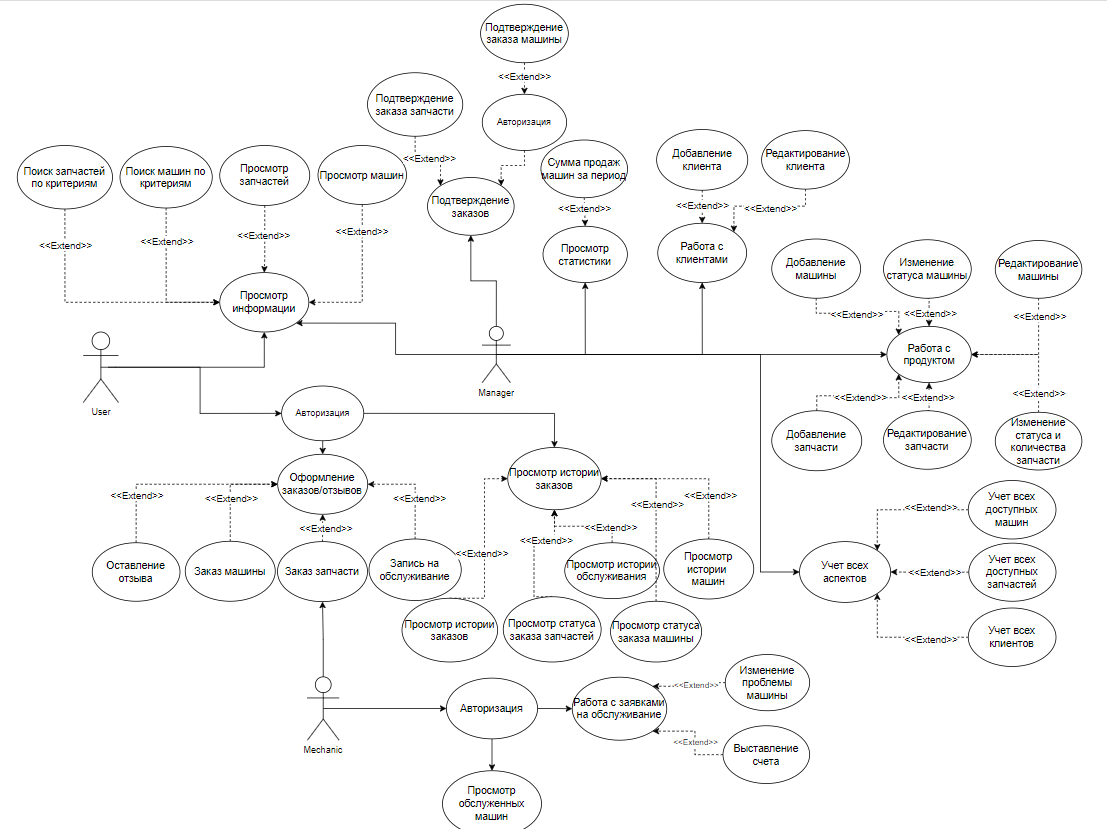


Рисунок 2.2 – Диаграмма вариантов использования

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, и автоматически создается роль с тем же названием. В PostgreSQL пользователь и роль совпадают по имени и используются взаимозаменяемо.

Пример создания роли и пользователя приведено на листинге 2.1.

|  |
| --- |
| create user manager with password '111' |

Листинг 2.1 – Создание роли «Менеджер»

Были разработаны следующие роли:

− Пользователь;

− Менеджер;

− Механик;

− Администратор;

− Программист.

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

|  |
| --- |
| CREATE USER MANAGER PASSWORD '111';  GRANT EXECUTE ON PROCEDURE add\_car(NUMERIC, NUMERIC, INT, NUMERIC, VARCHAR(20), NUMERIC, INT, VARCHAR(200), BOOLEAN) TO MANAGER |
| GRANT EXECUTE ON PROCEDURE update\_car(NUMERIC, NUMERIC, INT, NUMERIC, VARCHAR(20), NUMERIC, INT, VARCHAR(200), BOOLEAN) TO MANAGER |
| GRANT EXECUTE ON FUNCTION calculate\_total\_price(DATE, DATE) TO MANAGER |
| GRANT EXECUTE ON FUNCTION get\_available\_cars() TO MANAGER |

Листинг 2.2 – Создание пользователя «Менеджер»

2.4 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.

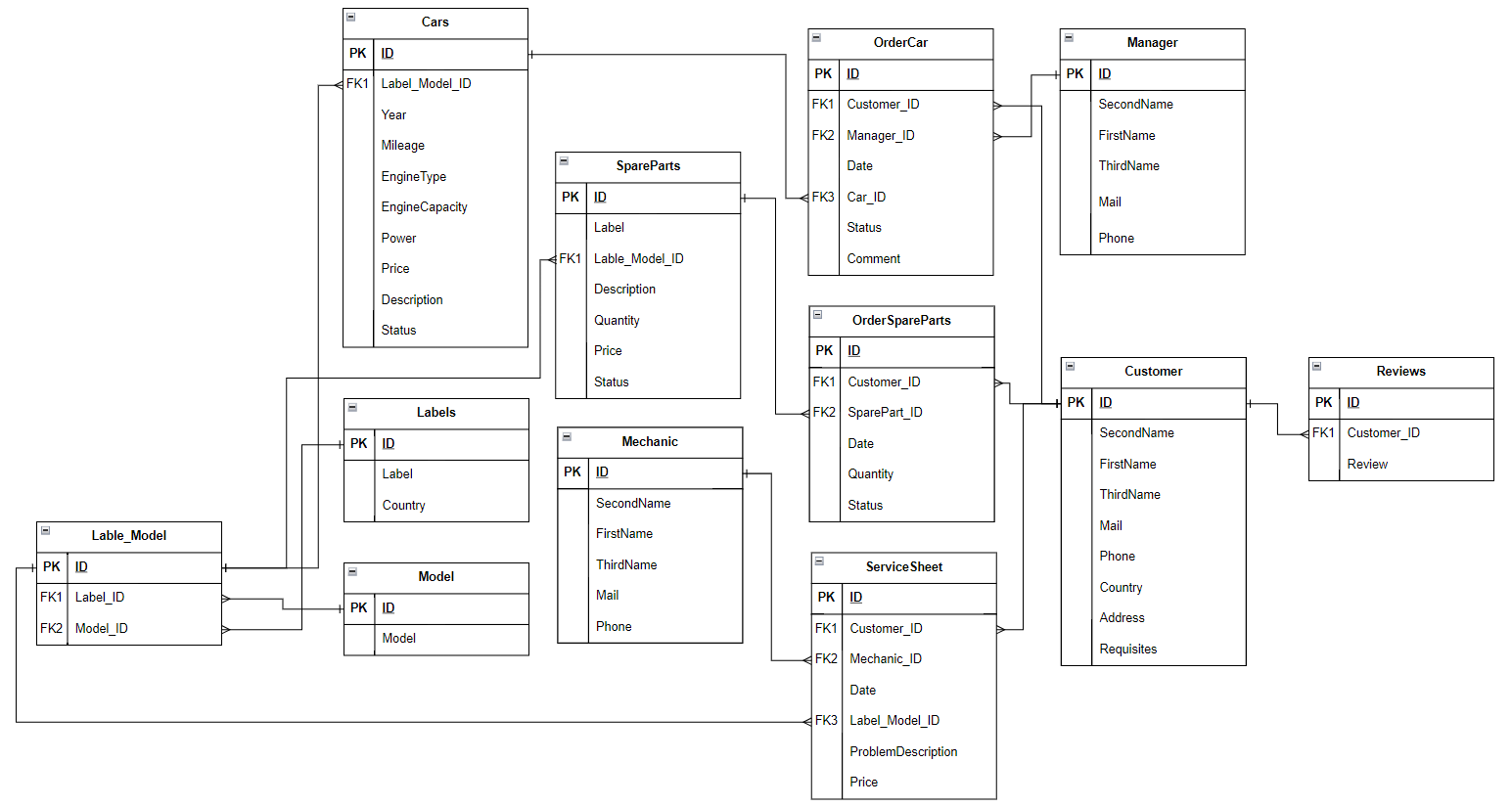


Рисунок 2.4 – Структура базы данных

2.4.1 Таблица Customer

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

− ID: хранит уникальный идентификатор каждого клиента. Тип данных – SERIAL.

− SECONDNAME: хранит фамилию клиента. Тип данных – VARCHAR.

− FIRSTNAME: хранит имя клиента. Тип данных – VARCHAR.

− THIRDNAME: хранит отчество клиента. Тип данных – VARCHAR.

− MAIL: хранит почту клиента. Тип данных – VARCHAR.

− PHONE: хранит номер телефона клиента. Тип данных – VARCHAR.

− COUNTRY: хранит страну проживания клиента. Тип данных – VARCHAR.

− ADDRESS: хранит полный адрес клиента. Тип данных – VARCHAR.

− REQUISITES: хранит реквизиты клиента для оплаты. Тип данных – VARCHAR.

Данная таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение, а также номер телефона и почта должны быть уникальными.

2.4.2 Таблица Manager

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

− ID: хранит уникальный идентификатор каждого менеджера. Тип данных– SERIAL.

− SECONDNAME: хранит фамилию менеджера. Тип данных – VARCHAR.

− FIRSTNAME: хранит имя менеджера. Тип данных – VARCHAR.

− THIRDNAME: хранит отчество менеджера. Тип данных – VARCHAR.

− MAIL: хранит почту менеджера. Тип данных – VARCHAR.

Данная таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение, а также номер телефона и почта должны быть уникальными.

2.4.3 Таблица Mechanic

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

− ID: хранит уникальный идентификатор каждого механика. Тип данных – SERIAL.

− SECONDNAME: хранит фамилию механика. Тип данных – VARCHAR.

− FIRSTNAME: хранит имя механика. Тип данных – VARCHAR.

− THIRDNAME: хранит отчество механика. Тип данных – VARCHAR.

− MAIL: хранит почту механика. Тип данных – VARCHAR.

− PHONE: хранит номер телефона механика. Тип данных – VARCHAR.

Данная таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение, а также номер телефона и почта должны быть уникальными.

2.4.4 Таблица CARS

Данная таблица используется для хранения информации об автомобилях. В её состав входят следующие столбцы:

− ID: хранит уникальный идентификатор каждого автомобиля. Тип данных - SERIAL.

− LABEL\_MODEL\_ID: хранит уникальный идентификатор связки марки и модели автомобиля. Тип данных - SERIAL.

− YEAR: хранит год выпуска автомобиля. Тип данных − INT.

− MILEAGE: хранит пробег автомобиля. Тип данных − NUMERIC.

− ENGINETYPE: хранит тип двигателя автомобиля. Тип данных − VARCHAR.

− ENGINECAPACITY: хранит объём двигателя автомобиля. Тип данных − NUMERIC.

− POWER: хранит мощность автомобиля в лошадиных силах. Тип данных − INT.

− PRICE: хранит цену автомобиля. Тип данных − MONEY.

− DESCRIPTION: хранит описание автомобиля. Тип данных − VARCHAR.

− STATUS: хранит статус автомобиля. Тип данных − BOOLEAN.

Таблица содержит ограничения первичного ключа для столбца ID и внешний ключ для столбца LABEL\_MODEL\_ID, ссылающийся на ID в таблице LABEL\_MODEL, а также все столбцы должны содержать какое-либо значение.

2.4.5 Таблица ORDERCAR

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

− ID: хранит уникальный идентификатор каждого заказа. Тип данных − SERIAL.

− CUSTOMER\_ID: хранит уникальный идентификатор клиента, сделавшего заказ. Тип данных − SERIAL.

− MANAGER\_ID: хранит уникальный идентификатор менеджера. Тип данных − SERIAL.

− DATE: хранит дату заказа. Тип данных - DATE.

− CAR\_ID: хранит уникальный идентификатор автомобиля, который был заказан. Тип данных − SERIAL.

− STATUS: хранит статус заказа (выполнен или нет). Тип данных − BOOLEAN.

− COMMENT: хранит комментарий к заказу. Тип данных – VARCHAR.

Таблица содержит ограничения первичного ключа для столбца ID и внешние ключи для столбцов CUSTOMER\_ID, MANAGER\_ID и CAR\_ID, ссылающиеся на соответствующие столбцы в таблицах CUSTOMER, MANAGER и CARS соответственно, а также все столбцы, кроме COMMENT, должны содержать какое-либо значение.

2.4.6 Таблица SPAREPARTS

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

− ID: хранит уникальный идентификатор каждой запчасти. Тип данных - SERIAL.

− LABEL: хранит название запчасти. Тип данных - VARCHAR.

− LABEL\_MODEL\_ID: хранит уникальный идентификатор связки марки и модели автомобиля, к которой относится запчасть. Тип данных - SERIAL.

− DESCRIPTION: хранит описание запчасти. Тип данных - VARCHAR.

− QUANTITY: хранит количество запчастей на складе. Тип данных - INT.

− PRICE: хранит цену запчасти. Тип данных - MONEY.

− STATUS: хранит статус доступности запчасти. Тип данных - BOOLEAN.

Таблица содержит ограничение первичного ключа для столбца ID и внешний ключ для столбца LABEL\_MODEL\_ID, ссылающийся на ID в таблице LABEL\_MODEL, а также все столбцы должны содержать какое-либо значение.

2.4.7 Таблица ORDERSPAREPARTS

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

− ID: хранит уникальный идентификатор каждого заказа запчастей. Тип данных − SERIAL.

− CUSTOMER\_ID: хранит уникальный идентификатор клиента, сделавшего заказ. Тип данных − SERIAL.

− SPAREPART\_ID: хранит уникальный идентификатор запчасти, которая была заказана. Тип данных − SERIAL.

− DATE: хранит дату заказа. Тип данных − DATE.

− QUANTITY: хранит количество заказанных запчастей. Тип данных − INT.

− STATUS: хранит статус заказа. Тип данных − BOOLEAN.

Таблица содержит ограничения первичного ключа для столбца ID и внешние ключи для столбцов CUSTOMER\_ID и SPAREPART\_ID, ссылающиеся на соответствующие столбцы в таблицах CUSTOMER и SPAREPARTS соответственно, а также все столбцы должны содержать какое-либо значение.

2.4.8 Таблица SERVICESHEET

Данная таблица используется для хранения информации о техническом обслуживании автомобилей. В её состав входят следующие столбцы:

− ID: хранит уникальный идентификатор каждого листа обслуживания. Тип данных − SERIAL.

− CUSTOMER\_ID: хранит уникальный идентификатор клиента, заказавшего обслуживание. Тип данных − SERIAL.

− MECHANIC\_ID: хранит уникальный идентификатор механика, проводившего обслуживание. Тип данных − SERIAL.

− DATE: хранит дату обслуживания. Тип данных − DATE.

− LABEL\_MODEL\_ID: хранит уникальный идентификатор связки марки и модели автомобиля, который обслуживается. Тип данных − SERIAL.

− PROBLEMDESCRIPTION: хранит описание проблемы, указанной клиентом. Тип данных − VARCHAR.

− PRICE: хранит стоимость обслуживания. Тип данных − MONEY.

Таблица содержит ограничения первичного ключа для столбца ID и внешние ключи для столбцов CUSTOMER\_ID, MECHANIC\_ID и LABEL\_MODEL\_ID, ссылающиеся на соответствующие столбцы в таблицах CUSTOMER, MECHANIC и LABEL\_MODEL соответственно, а также все столбцы, кроме PRICE, должны содержать какое-либо значение.

2.4.9 Таблица REVIEWS

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

− ID: хранит уникальный идентификатор каждого отзыва. Тип данных − SERIAL.

− CUSTOMER\_ID: хранит уникальный идентификатор клиента, оставившего отзыв. Тип данных − SERIAL.

− REVIEW: хранит текст отзыва. Тип данных − VARCHAR.

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

2.4.10 Таблица LABELS

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

− ID: хранит уникальный идентификатор каждой марки. Тип данных − SERIAL.

− LABEL: хранит название марки автомобиля. Тип данных − VARCHAR.

− COUNTRY: хранит название страны-производителя марки. Тип данных − VARCHAR.

Таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение.

2.4.11 Таблица MODEL

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

− ID: хранит уникальный идентификатор каждой модели. Тип данных − SERIAL.

− MODEL: хранит название модели автомобиля. Тип данных − VARCHAR.

Таблица содержит ограничение первичного ключа для столбца ID, а также все столбцы должны содержать какое-либо значение.

2.4.12 Таблица LABEL\_MODEL

Данная таблица используется для установления связи между марками и моделями автомобилей. В её состав входят следующие столбцы:

− ID: хранит уникальный идентификатор каждой связки марка-модель. Тип данных − SERIAL.

− LABEL\_ID: хранит уникальный идентификатор марки автомобиля. Тип данных − SERIAL.

− MODEL\_ID: хранит уникальный идентификатор модели автомобиля. Тип данных − SERIAL.

Таблица содержит ограничения первичного ключа для столбца ID и внешние ключи для столбцов LABEL\_ID и MODEL\_ID, ссылающиеся на столбцы ID в таблицах LABELS и MODEL соответственно, а также все столбцы должны содержать какое-либо значение.

2.5 Вывод

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

3. Разработка 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 12 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 FUNCTION. Для данного проекта были разработаны следующие функции:

– calculate\_total\_price: расчет суммы проданных машин за период времени

– get\_available\_cars: вывод всех доступных для покупки машин

– get\_available\_customers: вывод всех существующих клиентов

– get\_available\_spareparts: вывод всех доступных для покупки запчастей

– get\_all\_label\_model: вывод всех марок и моделей

– get\_all\_car: вывод всех машин

– get\_all\_customer: вывод всех клиентов

– get\_all\_manager: вывод всех менеджеров

– get\_orders\_cars\_by\_manager\_id: вывод всех машин у которых есть id менеджера

– get\_orders\_spare\_parts\_by\_customer\_id: вывод заказанных запчастей конкретным клиентом

– get\_all\_sparepart: вывод всех запчастей

– get\_customer\_by\_phone: авторизация клиента по номеру телефона

– check\_quantity\_spareparts: проверка количества запчастей

– get\_all\_mechanic: вывод всех механиков

– get\_date\_status: вывод даты и статуса заказа автомобиля

– get\_date\_status\_spare: вывод даты и статуса заказа запчасти

– get\_history\_orders\_cars: вывод всей истории заказов машин конкретным клиентом

– get\_history\_orders\_spareparts: вывод всей истории заказов запчастей конкретным клиентом

– get\_servicesheet: вывод всех записей на обслуживание для конкретного клиента

– get\_all\_reviews: вывод последних 30 отзывав по дате

– get\_labels: вывод всех марок

– get\_all\_models\_by\_label: вывод всех моделей для конкретной марки

– search\_cars: поиск машин по критериям

– search\_spareparts: поиск запчастей по критериям

– get\_service\_sheet: вывод всех записей на обслуживание для конкретного механика

– get\_customer\_full\_names: вывод имен всех клиентов

– get\_mechanic\_by\_phone: авторизация механика по номеру телефона

– create\_report\_car: заполнение таблицы для дальнейшего отчета о проданных автомобилях

– create\_report\_sparepart: заполнение таблицы для дальнейшего отчета о проданных запчастях

– generate\_sales\_report\_car: создание отчета о проданных автомобилях в JSON формат

– generate\_sales\_report\_sparepart: создание отчета о проданных запчастях в JSON формат

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

|  |
| --- |
| 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 |
| ) |
| LANGUAGE plpgsql |
| SECURITY DEFINER |
| AS $$ |
| BEGIN |
| RETURN QUERY |
| SELECT o.id, m.secondname, l.label, mo.model, c.year, mc.secondname, o.status |
| FROM ORDERCAR o |
| INNER JOIN MANAGER m ON o.manager\_id = m.id |
| INNER JOIN CARS c ON o.car\_id = c.id |
| INNER JOIN LABEL\_MODEL lm ON c.label\_model\_id = lm.id |
| INNER JOIN LABELS l ON lm.label\_id = l.id |
| INNER JOIN MODEL mo ON lm.model\_id = mo.id |
| INNER JOIN CUSTOMER mc ON o.customer\_id = mc.id |
| WHERE o.manager\_id = managers\_id; |
| END; |
| $$; |

Листинг 3.1 – Функция получения заказов машин конкретного менеджера

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:

– add\_car: создание нового автомобиля

– update\_car: обновление существующего автомобиля

– delete\_car: изменение статуса доступности автомобиля для покупки

– add\_sparepart: создание новой запчасти

– update\_sparepart: обновление существующей запчасти

– delete\_sparepart: изменение статуса доступности запчасти для покупки

– add\_customer: создание нового клиента

– update\_customer: обновление существующего клиента

– change\_status\_ordercar: подтверждение заказа автомобиля

– change\_status\_orderspareparts: подтверждение заказа запчасти

– add\_ordercar: создание нового заказа автомобиля

– add\_ordersparepart: создание нового заказа запчасти

– add\_review: создание нового отзыва

– add\_servicesheet: создание новой записи на обслуживание автомобиля

– update\_service\_sheet: изменение статуса записи на обслуживание и проблемы

– update\_sparepart\_quantity: изъятие запчастей из наличия

– add\_manager: создание нового менеджера

– delete\_manager: удаление существующего менеджера

– add\_mechanic: создание нового механика

– delete\_mechanic: удаление существующего механика

– change\_car\_price: изменение цены автомобиля

– change\_sparepart\_price: изменение цены запчасти

– add\_review\_from\_json: извлечение данных из JSON и запись в таблицу

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

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

Листинг 3.2 – Процедура создания нового заказа машины

В сумме, при разработке курсового проекта, было создано 22 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:

update\_reports\_car\_trigger: вызывается при создании новой записи в заказанных автомобилях

update\_reports\_sparepart\_trigger: вызывается при создании новой записи в заказанных запчастях

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

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

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\_reviews\_customer\_rating\_date ON reviews (customer\_id, rating, date) |

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.

4.1 Процедура экспорта данных

Функция экспорта используется для таблицы «ORDERCAR» и «ORDERSPAREPART» для создания отчета о проданных автомобилях и запчастях.

Для этого были разработаны функции, триггеры и таблицы для экспорта. Их представление отображено в листингах 4.1, 4.2 и 4.3 и приложениях Б и В.

|  |
| --- |
| 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 4.1 – Функция обновления данных проданных автомобилей

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

Листинг 4.2 – Триггер, срабатывающий на добавление записей

|  |
| --- |
| 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 4.3 – 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 4.4.

|  |
| --- |
| 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 4.4 – 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.

|  |
| --- |
| DO $$ |
| DECLARE |
| i INT; |
| random\_customer\_id INT; |
| random\_date TIMESTAMP; |
| random\_rating INT; |
| random\_review TEXT; |
| BEGIN |
| FOR i IN 1..100000 LOOP |
| SELECT id INTO random\_customer\_id FROM customer ORDER BY RANDOM() LIMIT 1; |
|  |
| random\_date := NOW() - INTERVAL '2 years' \* RANDOM(); |
|  |
| random\_rating := FLOOR(RANDOM() \* 5 + 1)::INT; |
|  |
| random\_review := 'Отзыв ' || i; |
|  |
| INSERT INTO reviews (customer\_id, date, rating, review) |
| VALUES (random\_customer\_id, random\_date, random\_rating, random\_review); |
| END LOOP; |
| END $$; |

Листинг 3.5 – Код скрипта вставки 100000 строк в таблицу REVIEWS

5.2 Тестирование производительности по таблице REVIEWS.

В PostgreSQL оптимизация запросов в основном заключается в построение индексов над таблицами.

Для тестирования производительности были добавлены 100 000 записей в таблицу REVIEWS.

Ниже на рисунке 5.1 представлен результат простого Select-запроса к таблице до добавления индекса. Время выполнения составляет 112 мс.

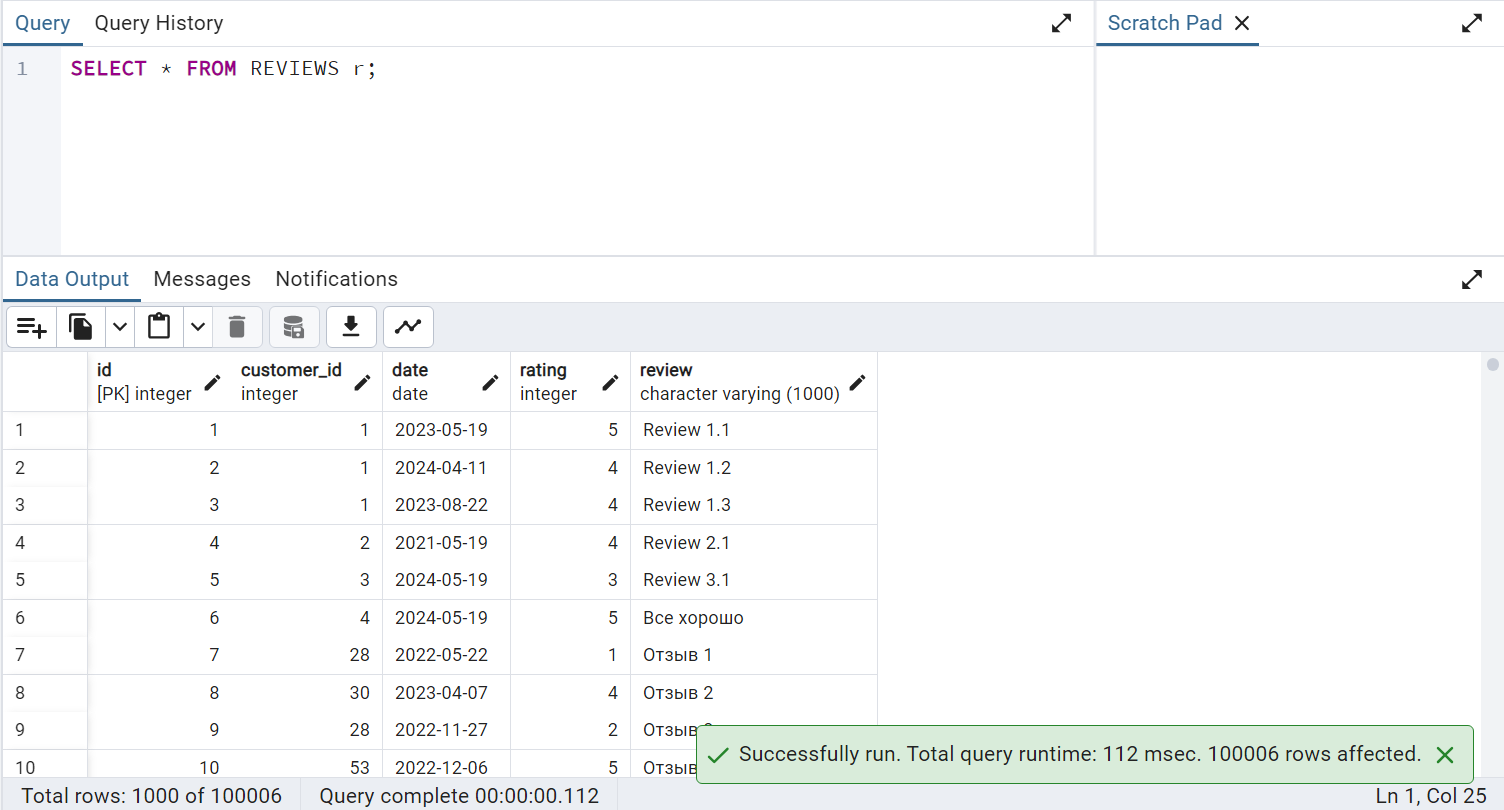


Рисунок 5.1 – Результат Select-запроса к таблице без индекса

На рисунке 4.2 представлен результат Select-запроса к таблице после добавления индекса. Время выполнения уже составляет 88 мс.

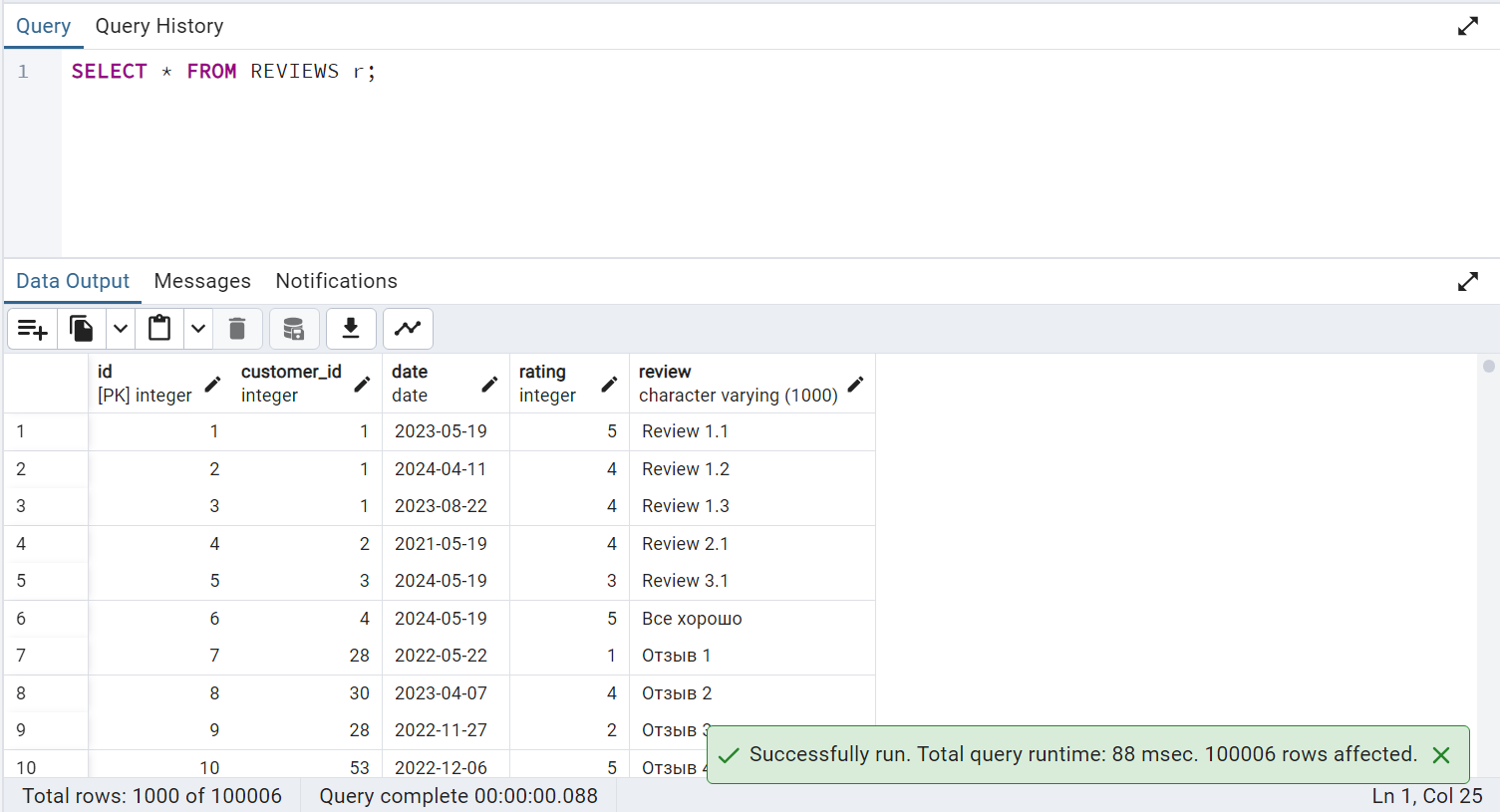
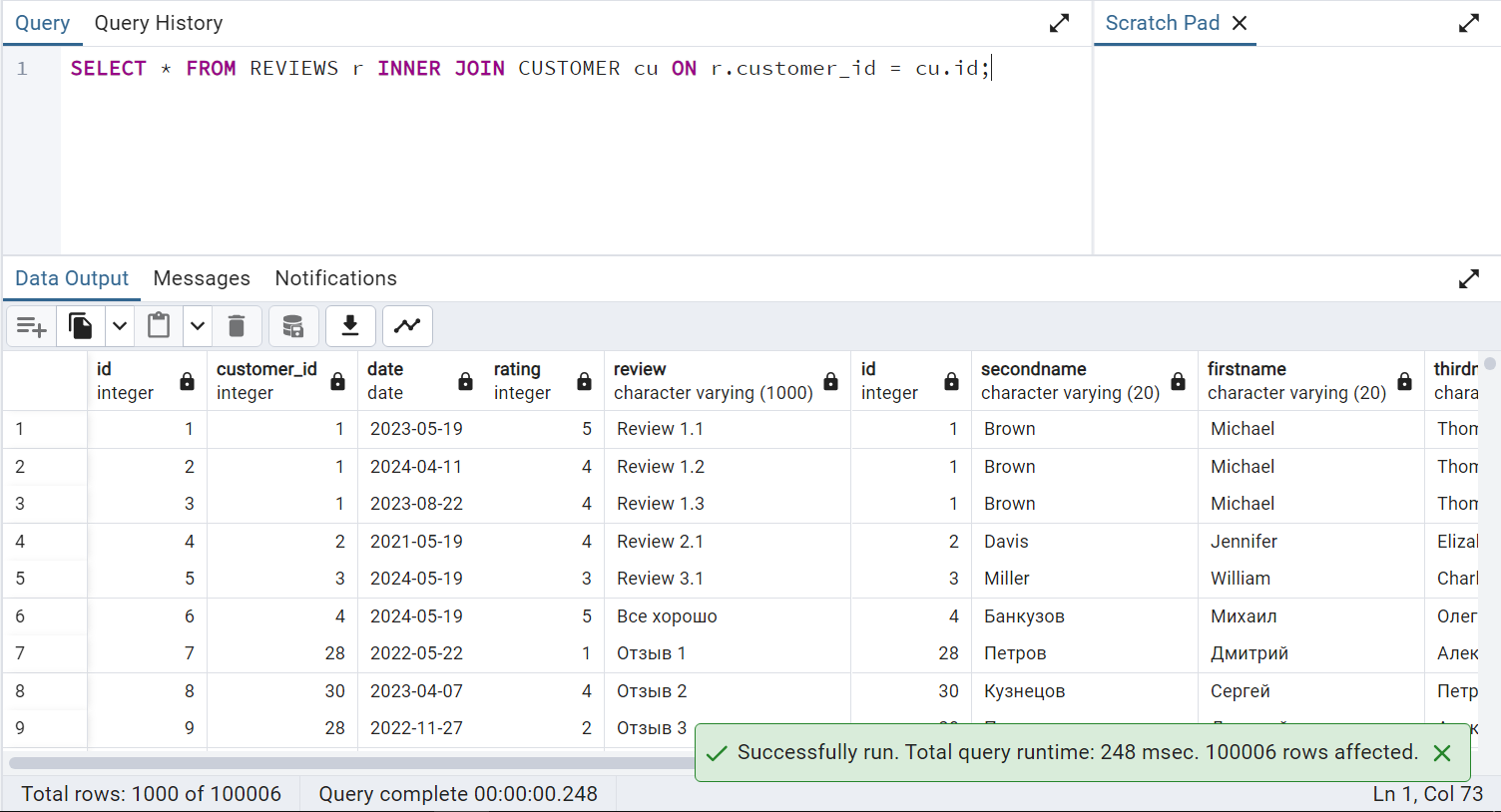


Рисунок 5.2 – Результат Select-запроса к таблице с индексом

На рисунке 5.3 представлен результат Select-запроса с использованием INNER JOIN до добавления индекса. Время выполнения составляет 248 мс.

Рисунок 5.3 – Результат Select-запроса к таблице без индекса

На рисунке 5.4 представлен результат Select-запроса с использованием INNER JOIN после добавления индекса. Время выполнения уже составляет 206 мс.

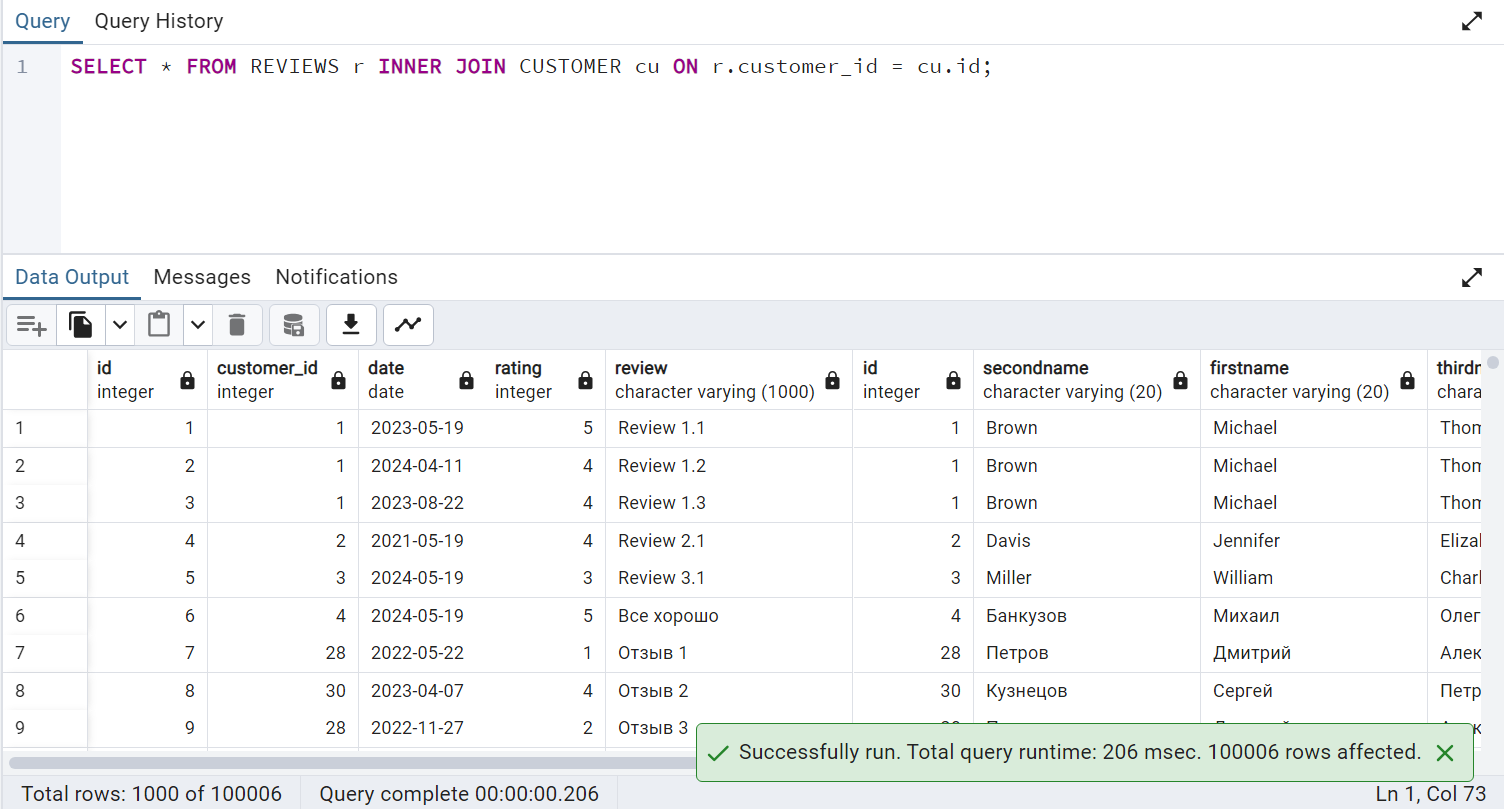


Рисунок 5.2 – Результат Select-запроса к таблице с индексом

На рисунке 5.5 представлен результат Select-запроса с использованием INNER JOIN и условием ORDER BY до добавления индекса. Время выполнения составляет 351 мс.

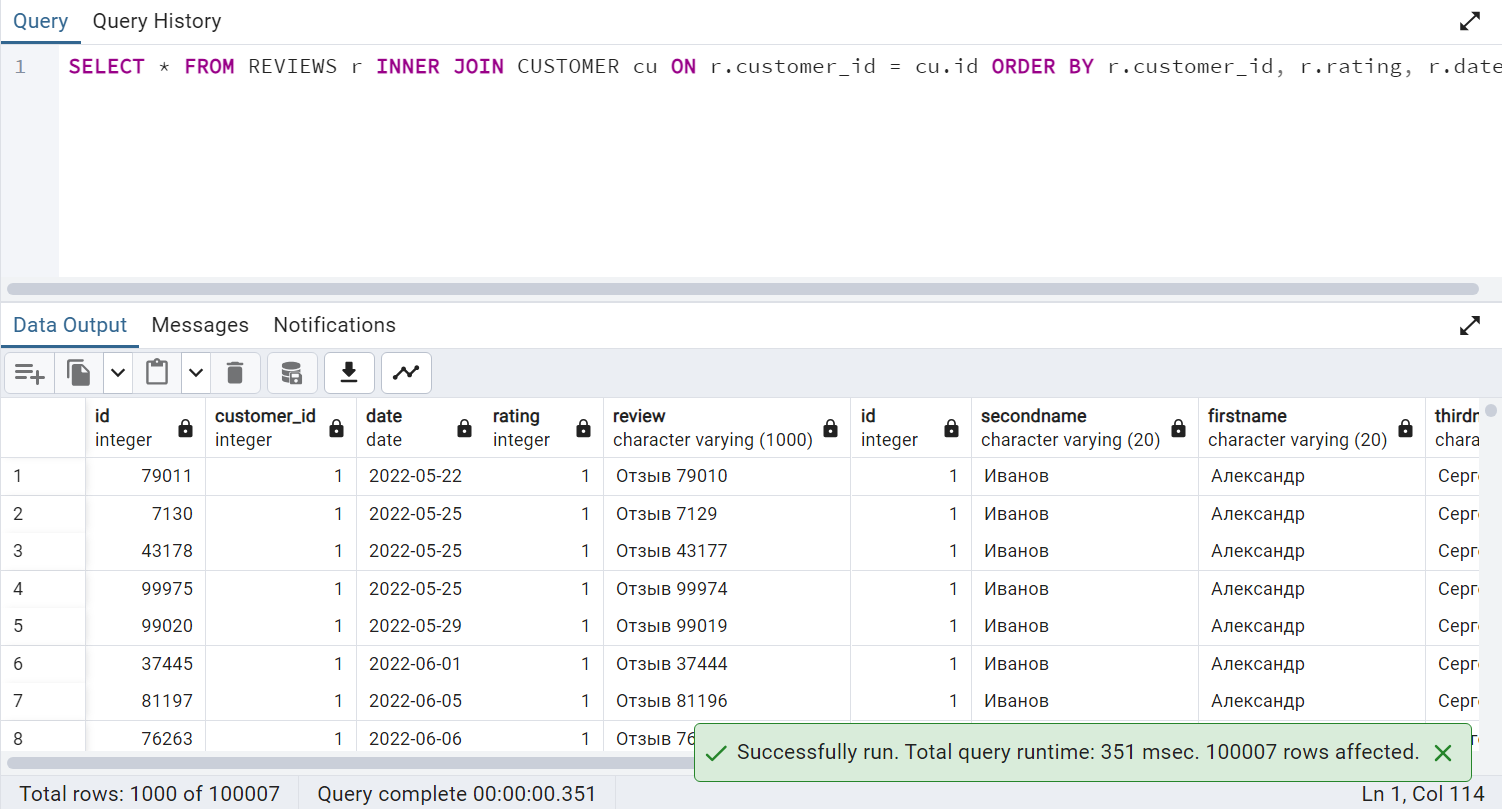


Рисунок 5.5 – Результат Select-запроса к таблице без индекса

На рисунке 5.6 представлен результат Select-запроса с использованием INNER JOIN и условием ORDER BY после добавления индекса. Время выполнения уже составляет 224 мс.

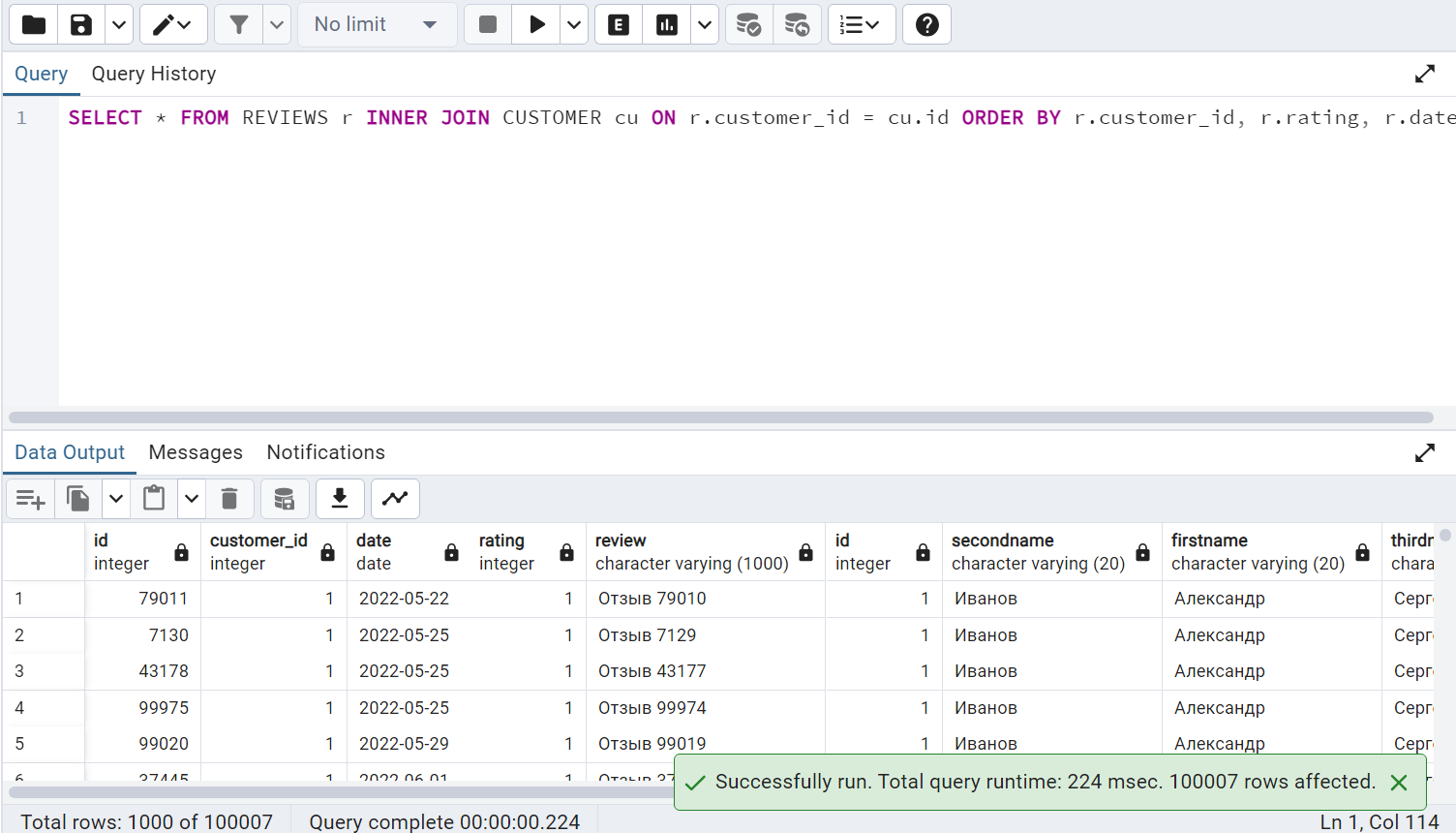


Рисунок 5.6 – Результат Select-запроса к таблице с индексом

5.3 Вывод

Тестирование производительности базы данных на примере таблицы «REVIEWS» 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 «Разработка системы мониторинга за состоянием базы данных»

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

Для мониторинга состояния базы данных PostgreSQL в данном проекте используется pgAdmin 4. pgAdmin — популярное графическое средство для администрирования PostgreSQL. Программа упрощает основные задачи администрирования, отображает объекты баз данных, позволяет выполнять запросы SQL.

Представление pg\_stat\_statements является одним из основных источников статистики по всем запросам, которые выполняются в СУБД. Представление служит интерфейсом к одноименному расширению, которое отслеживает стадии выполнения запросов и хранит накопленную статистику. Расширение сохраняет статистику по всем успешно выполненным запросам. В случаях, когда запрос завершился ошибкой (или был прерван администратором), статистика о нем не будет сохранена. После установки расширения сбор статистики начинается автоматически.

Для установки представления statements необходимо внести изменения в файл конфигурации PostgreSQL postgresql.conf и перезагрузить сервер. Данные изменения представлены в листинге 6.1.

|  |
| --- |
| shared\_preload\_libraries = 'pg\_stat\_statements' # (change requires restart) |

Листинг 6.1 – Изменения файла конфигурации

После чего можно установить представление, которое начнет собирать статистику состояния СУБД автоматически. Создание представления представлено в листинге 6.2.

|  |
| --- |
| CREATE EXTENSION pg\_stat\_statements |

Листинг 6.2 – Создание представления pg\_stat\_statements

6.2 Графическое средство для администрирования pgAdmin 4

pgAdmin 4 предоставляет несколько вариантов мониторинга с различными критериями, такими как Dashboard General, Dashboard System Statistics, Statistics.

Dashboard General представлен на рисунке 6.1. На нем отображаются различные графики, такие как Database session, Transactions per second, Tuples in, Tuples out и Block I/O.

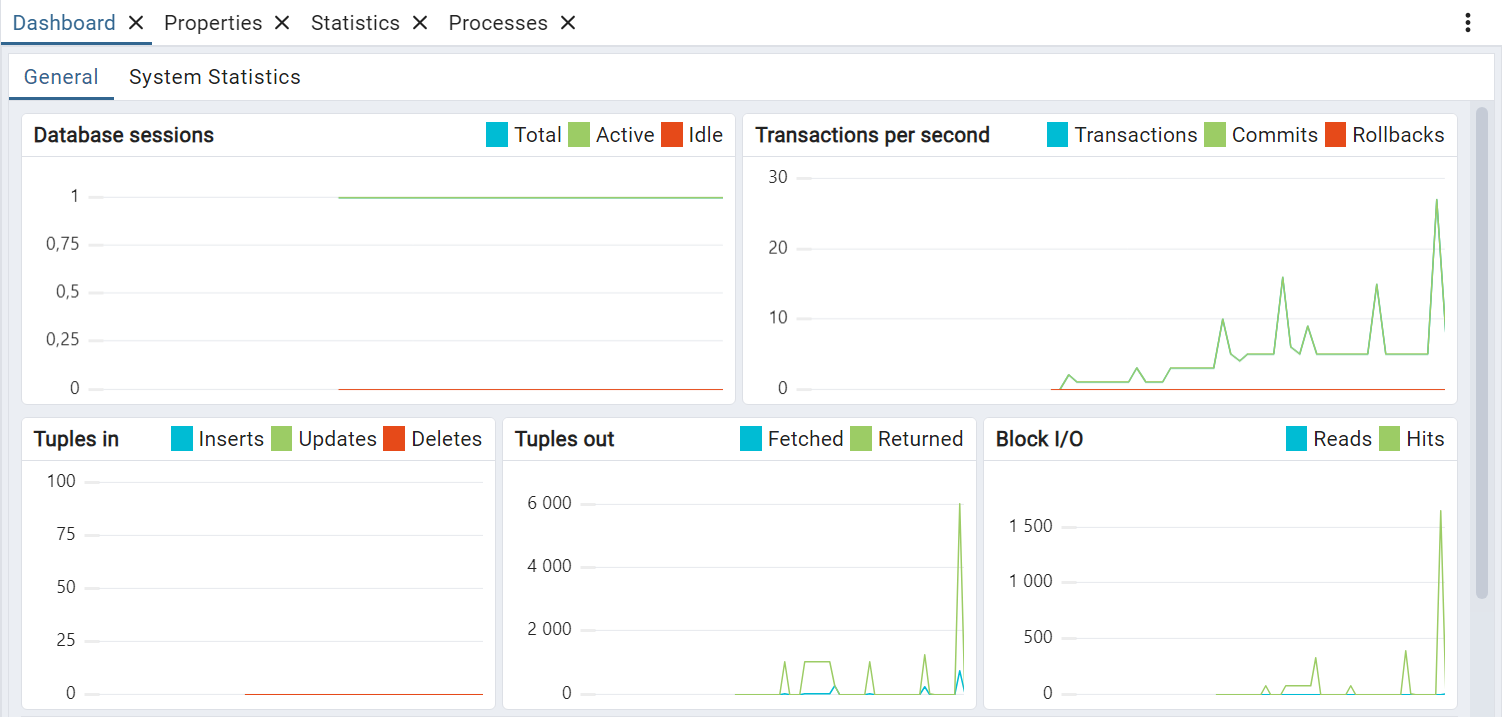


Рисунок 6.1 – Приборная панель pgAdmin

Database session – этот график отображает количество активных сессий базы данных во времени. График идет вверх, когда увеличивается количество подключений к базе данных и идет вниз, когда сессии закрываются или завершаются.

Transactions per second – этот график показывает количество транзакций, обрабатываемых базой данных в секунду. График идет вверх, когда увеличивается количество выполняемых транзакций и идет вниз, когда количество транзакций уменьшается.

Tuples in – этот график показывает количество вставленных и обновленных кортежей (строк) в базу данных. График идет вверх, когда в базу данных вставляется или обновляется большое количество данных и идет вниз, когда активность по вставке и обновлению данных снижается.

Tuples out – этот график показывает количество извлеченных кортежей (строк) из базы данных. График идет вверх, когда выполняется много операций чтения данных, например, при выполнении сложных запросов, отчетов или выборок данных пользователями и идет вниз, когда количество операций чтения данных уменьшается.

Block I/O – этот график отображает количество операций ввода-вывода блоков, выполняемых базой данных. График идет вверх, когда увеличивается активность чтения и записи данных на диске и идет вниз, когда активность ввода-вывода блоков снижается, что может быть связано с уменьшением рабочей нагрузки или улучшением производительности за счет кэширования данных.

Так же pgAdmin 4 предоставляет возможность мониторинга за состоянием системы с помощью Dashboard System Statistics, что представлено на рисунках 6.2 и 6.3.

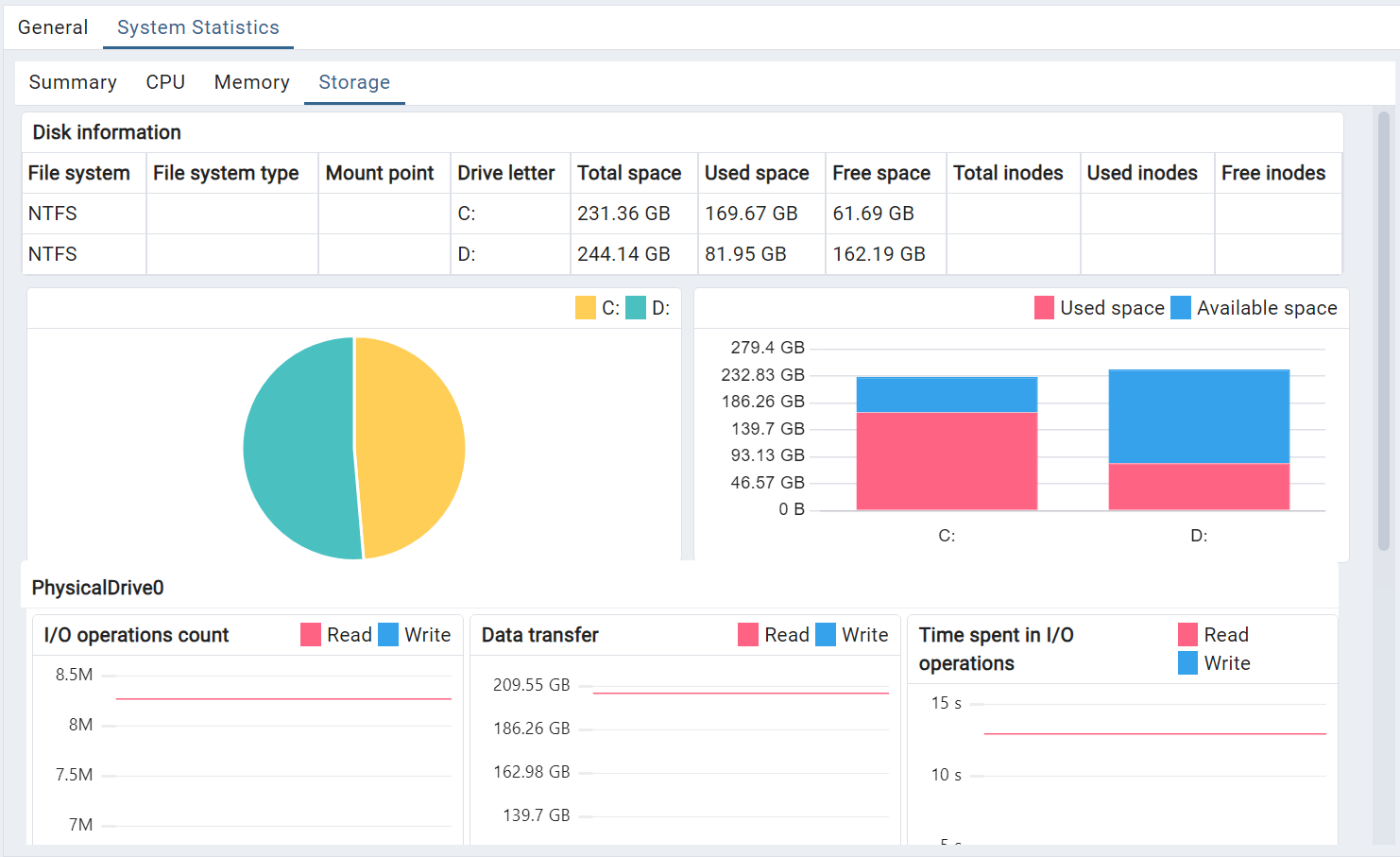


Рисунок 6.2 – Отслеживание состояния постоянной памяти

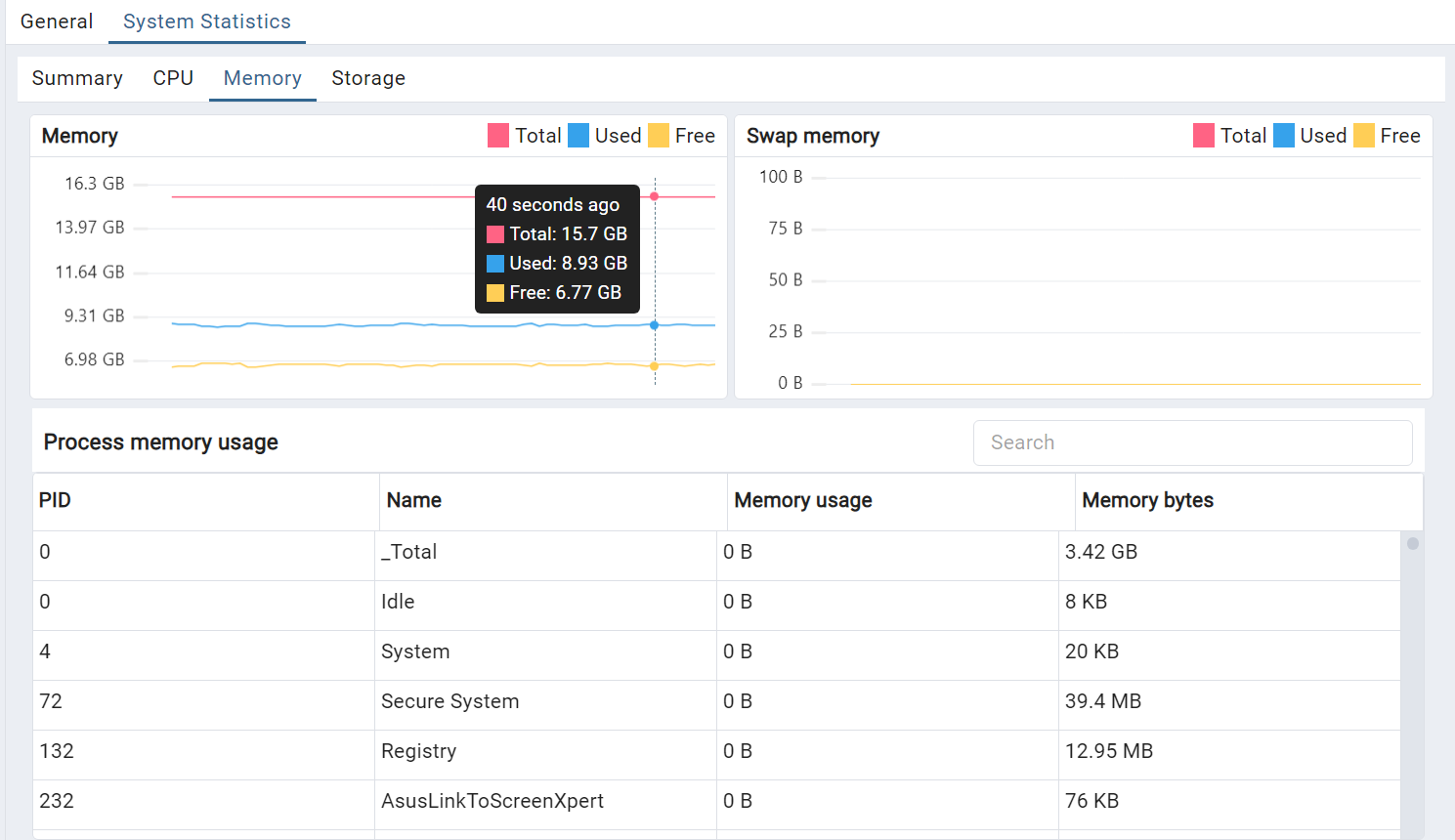


Рисунок 6.3 – Отслеживание состояния оперативной памяти

Так же не менее важным является хранение и отображение общих сведений о базе данных, что pgAdmin предоставляет в Statistics. Окно Statistics представлено на рисунке 6.4.

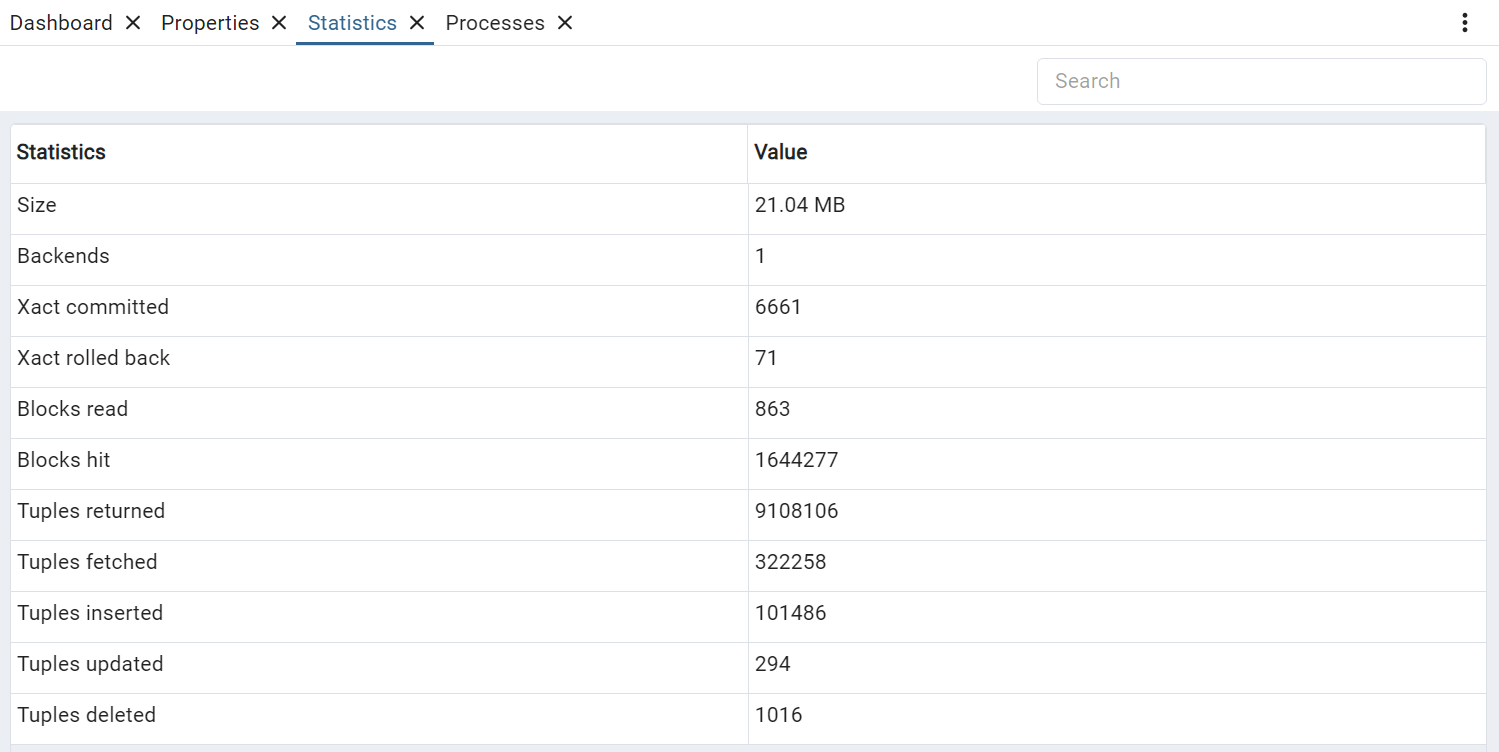


Рисунок 6.4 – Отслеживание общих сведений базы данных

– Поле Size отвечает и отображает общий размер базы данных, которая располагается в постоянной памяти.

– Поле Backends отображает количество активных подключений к базе данных.  
 – Поле Xact committed отображает общее количество успешно завершенных транзакций.

– Поле Xact rolled back общее количество откатанных транзакций.

– Поле Blocks read общее количество прочитанных блоков с диска.

– Поле Blocks hit общее количество обращений к блокам данных в памяти (кэш).

– Поле Tuples returned общее количество возвращенных кортежей (строк) при запросах.

– Поле Tuples fetched общее количество извлеченных кортежей (строк) из базы данных.

– Поле Tuples inserted общее количество вставленных кортежей (строк).

– Поле Tuples updated общее количество обновленных кортежей (строк).

– Поле Tuples deleted количество удаленных кортежей (строк).

6.3 Представление pg\_stat\_statements

Для работы с представлением pg\_stat\_statements в данном проекте будет так же использоваться pgAdmin. Для этого необходимо создать Query Tool. Query Tool – инструмент для взаимодействия с базами данных PostgreSQL. Он предназначен для выполнения SQL-запросов и анализа результатов, а также позволяет строить графики на основе выполненного запроса. Демонстрация работы Query Tool и pg\_stat\_statements представлена на рисунках 6.5 и 6.6.

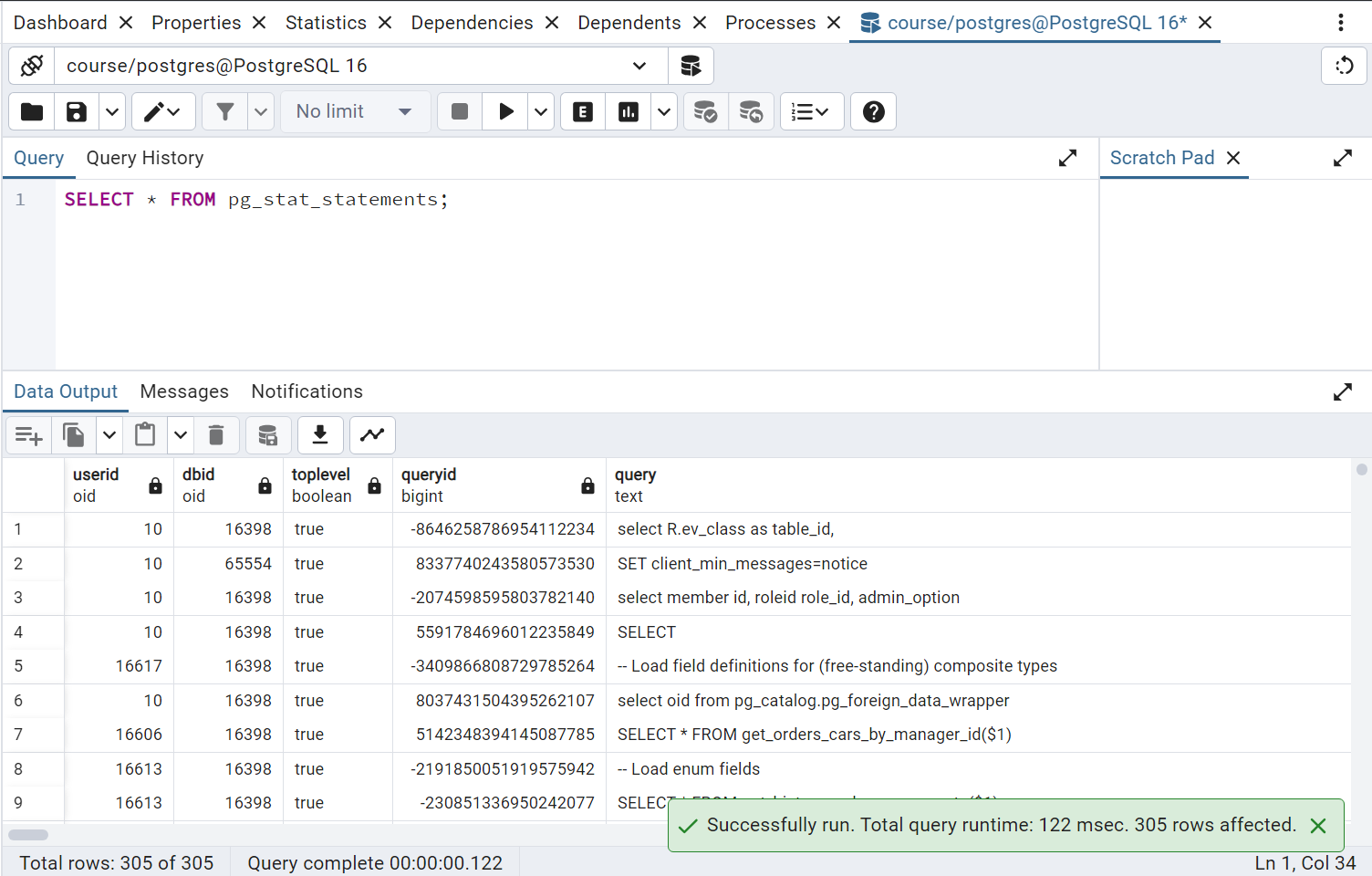


Рисунок 6.5 – Получение всех данных из pg\_stat\_statements

На рисунке 6.6 для примера был построен график зависимости userid (OID пользователя, выполнявшего оператор) от rows (общее число строк, полученных или затронутых оператором).

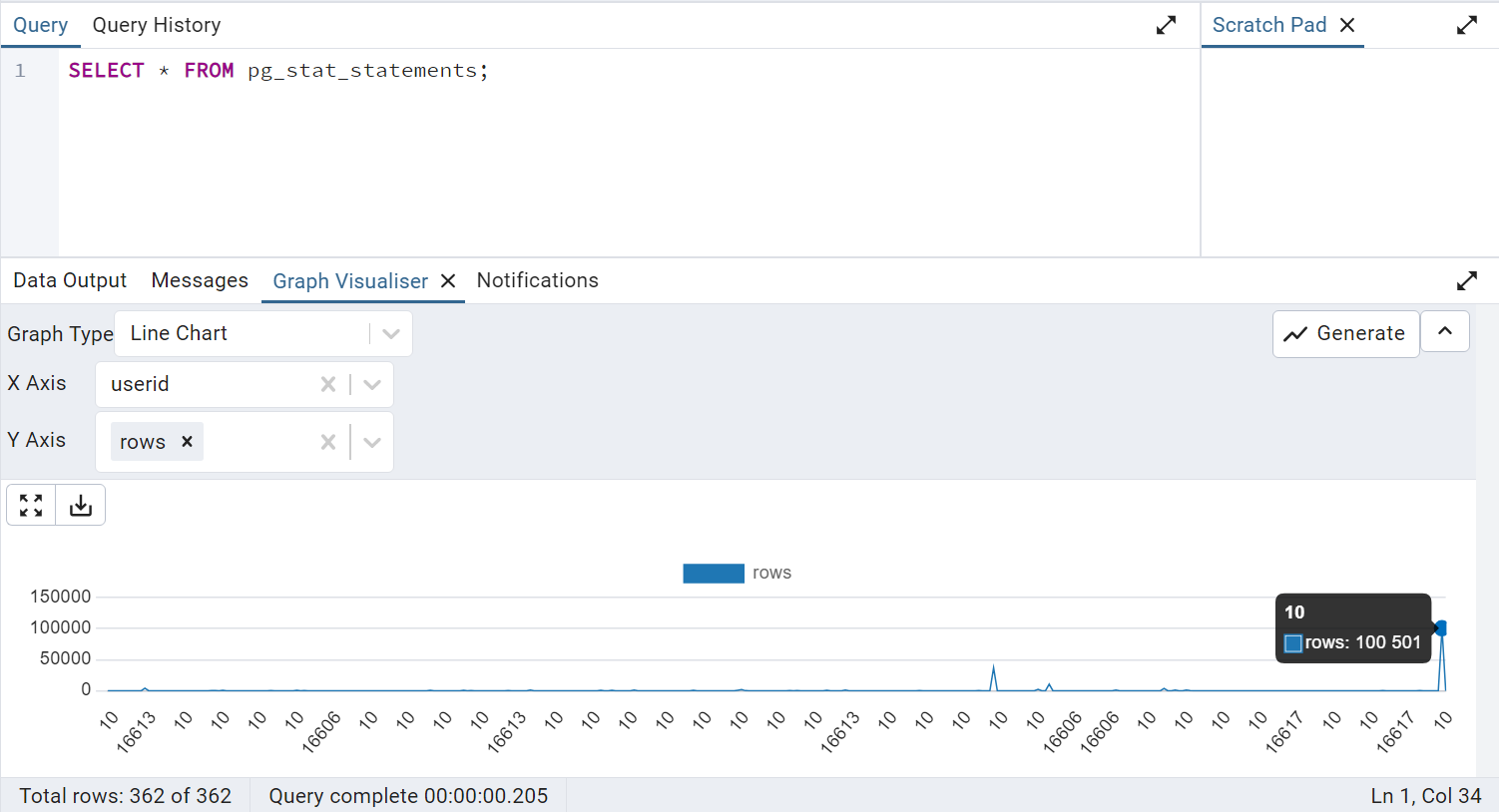


Рисунок 6.6 – Построение графика

Одними из важных полей pg\_stat\_statements являются:

– userid: OID пользователя, выполнявшего оператор

– dbid: OID базы данных, в которой выполнялся оператор

– query: текст, представляющий оператор

– total\_plan\_time: общее время, затраченное на планирование этого оператора в миллисекундах

– calls: число выполнений

– total\_exec\_time: общее время, затраченное на выполнение оператора, в миллисекундах

– rows: общее число строк, полученных или затронутых оператором

Также pgAdmin предоставляет возможность построения различных типов графиков, в том числе в виде диаграмм. Данный тип представлен на рисунке 6.7.

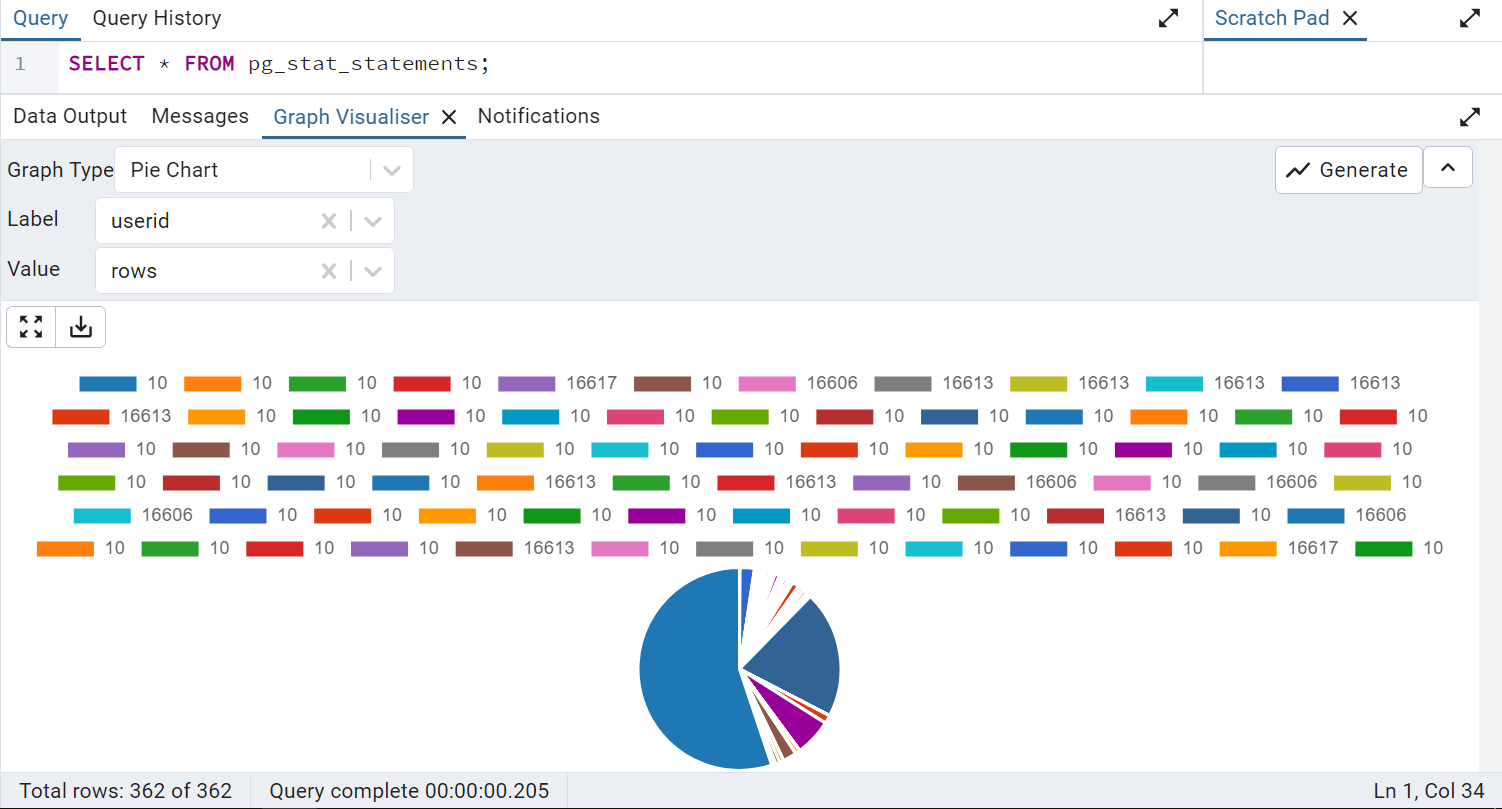


Рисунок 6.7 – Построение диаграммы

И, конечно, можно использовать различные арифметические функции, которые будут объединять различный столбцы и высчитывать более точные данные по затраченному времени и количеству вызовов. Пример такого запроса, который выводит текст запроса, а также количество его вызовов представлен в листинге 6.3, а также его график на рисунке 6.8.

|  |
| --- |
| SELECT |
| sum(calls) AS total\_calls, |
| Left (query, 64) AS query\_trunc |
| FROM pg\_stat\_statements |
| GROUP BY query |
| ORDER BY sum(calls) DESC; |

Листинг 6.3 – Пример запроса

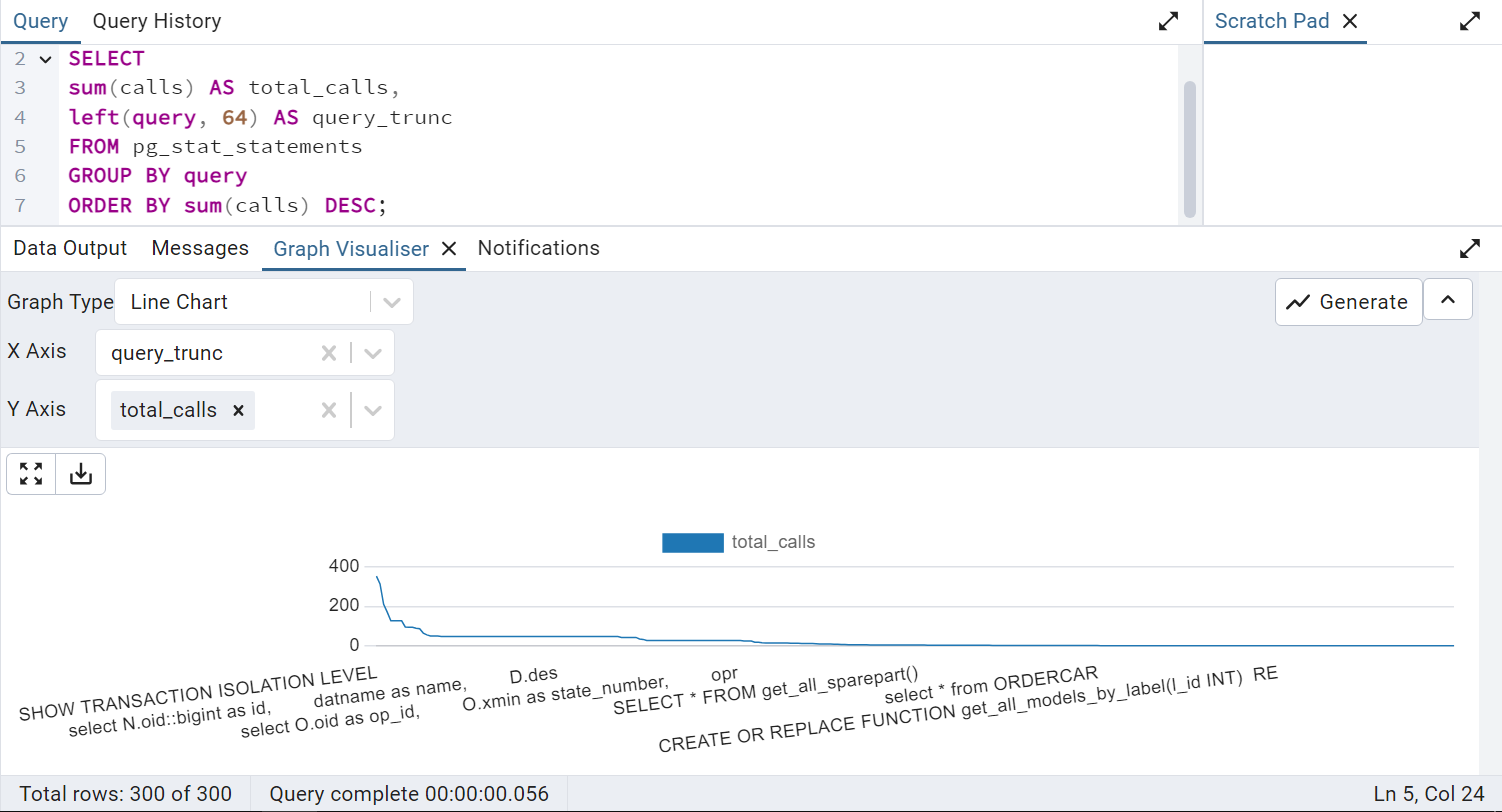


Рисунок 6.8 – Построение графика

6.4 Вывод

Технология мониторинга состояния СУБД PostgreSQL используется для обеспечения высокой производительности и надежности базы данных. В данном проекте применяется pgAdmin 4, популярное графическое средство для администрирования PostgreSQL, которое упрощает задачи мониторинга и анализа работы базы данных в реальном времени. Для сбора и анализа статистики запросов используется представление pg\_stat\_statements, которое отслеживает и сохраняет статистику выполнения запросов, предоставляя ценные данные для оптимизации работы СУБД.

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 «User»

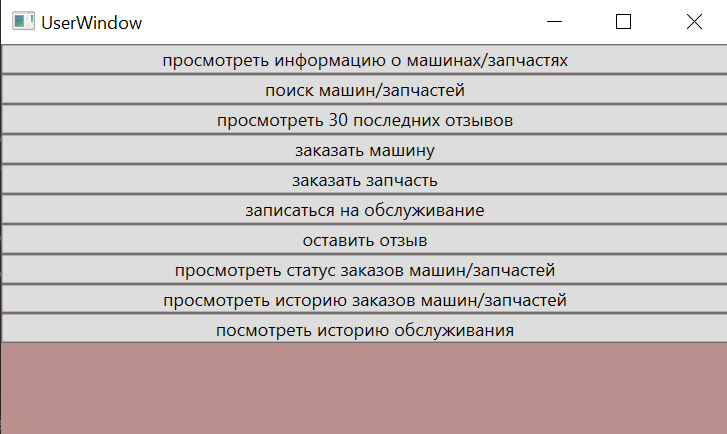


Рисунок 7.1 – Окно взаимодействия пользователя

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

Просмотр информации:

− Просмотреть информацию о машинах/запчастях: Возможность просмотра подробной информации о доступных автомобилях и запчастях.

Поиск:

− Поиск машин/запчастей: функция поиска нужных автомобилей или запчастей по критериям.

Отзывы:

− Просмотреть 30 последних отзывов: просмотр последних 30 отзывов, оставленных другими пользователями.

Заказы:

− Заказать машину: возможность оформить заказ на покупку автомобиля.

− Заказать запчасть: возможность оформить заказ на покупку запчасти.

Запись на обслуживание:

− Записаться на обслуживание: возможность записаться на обслуживание машины.

Оставить отзыв:

− Оставить отзыв: возможность оставить отзыв о автосалоне.

Просмотр статусов:

− Просмотреть статус заказов машин/запчастей: возможность проверить текущий статус и дату своих заказов на автомобили и запчасти.

История заказов:

− Просмотреть историю заказов машин/запчастей: возможность просмотреть историю всех своих заказов на автомобили и запчасти.

История обслуживания:

− Посмотреть историю обслуживания: возможность просмотреть историю всех своих обращений в сервисный центр для обслуживания автомобиля.

Эти варианты использования позволяют пользователю системы автосалона выполнять основные действия, связанные с покупкой, обслуживанием и отслеживанием машин и запчастей, а также взаимодействовать с системой посредством отзывов и проверки статусов заказов.

7.2 Инструкция по эксплуатации для пользователя «Manager»

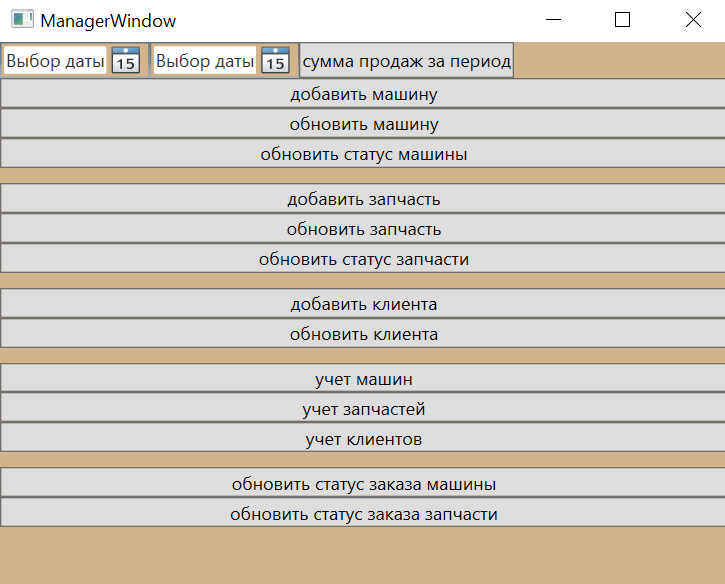


Рисунок 7.2 – Окно взаимодействия пользователя

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

Выбор даты:

− Выбор даты (левая кнопка): выбор начальной даты для определения периода.

− Выбор даты (правая кнопка): выбор конечной даты для определения периода.

− Сумма продаж автомобилей за период: подсчет и отображение общей суммы продаж за выбранный период времени.

Управление машинами:

− Добавить машину: ввод информации и добавление нового автомобиля в базу данных.

− Обновить машину: редактирование информации об уже существующем автомобиле.

− Обновить статус машины: изменение статуса автомобиля.

Управление запчастями:

− Добавить запчасть: ввод информации и добавление новой запчасти в базу данных.

− Обновить запчасть: редактирование информации о уже существующей запчасти.

− Обновить статус запчасти: изменение статуса запчасти.

Управление клиентами:

− Добавить клиента: ввод информации и добавление нового клиента в базу данных.

− Обновить клиента: редактирование информации о уже существующем клиенте.

Учет данных:

− Учет машин: просмотр данных о всех автомобилях.

− Учет запчастей: просмотр данных о всех запчастях.

− Учет клиентов: просмотр данных о всех клиентах.

Обновление статусов заказов:

− Обновить статус заказа машины: изменение статуса заказа на автомобиль.

− Обновить статус заказа запчасти: изменение статуса заказа на запчасть.

Эти варианты использования позволяют пользователю менеджеру управлять различными аспектами автосалона, включая управление инвентарем (машины и запчасти), клиентами, а также отслеживание и управление продажами и заказами.

7.3 Инструкция по эксплуатации для пользователя «Mechanic»

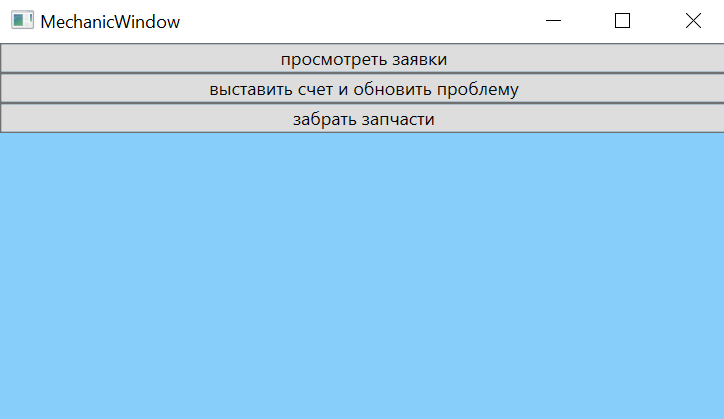


Рисунок 7.3 – Окно взаимодействия пользователя

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

Просмотр заявок:

− Просмотреть заявки: возможность просматривать поступившие заявки на обслуживание автомобилей.

Выставление счета и обновление проблемы:

− Выставить счет и обновить проблему: возможность выставить счет за выполненные работы и обновить информацию о решении проблемы, указанной в заявке.

Получение запчастей:

− Забрать запчасти: возможность получить запчасти, необходимые для выполнения ремонта или обслуживания автомобиля.

Эти варианты использования позволяют механику эффективно управлять своими задачами, контролировать процесс выполнения работ, выставлять счета за выполненные услуги и получать необходимые запчасти для ремонта.

7.4 Инструкция по эксплуатации для пользователя «Administrator»

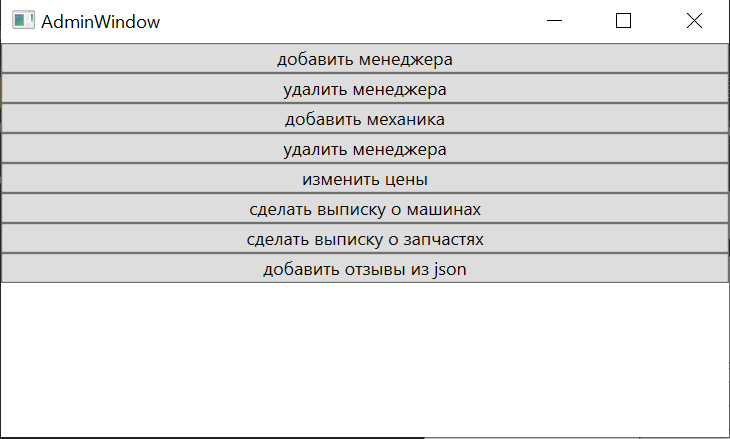


Рисунок 7.4 – Окно взаимодействия пользователя

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

Управление персоналом:

− Добавить менеджера: возможность создать нового менеджера.

− Удалить менеджера: возможность удалить уже существующего менеджера.

− Добавить механика: возможность создать нового механика.

− Удалить механика: возможность уделить уже существующего механика.

Управление ценами:

− Изменить цену: возможность изменять цены на любые автомобили и запчасти.

Оформление отчетов:

− Сделать выписку о машинах: возможность создать отчет о проданных автомобилях в формат JSON.

− Сделать выписку о запчастях: возможность создать отчет о проданных запчастях в формат JSON.

Добавить отзывы:

− Добавить отзывы из JSON: возможность добавить отзывы из файла JSON.

Эти варианты использования позволяют механику эффективно управлять своими задачами, контролировать процесс выполнения работ, выставлять счета за выполненные услуги и получать необходимые запчасти для ремонта.

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, инвентарем, клиентами и 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 «JetCar» и «Royal Motors», которые продемонстрировали важность использования баз данных для управления автосалонами и предоставления удобного доступа к информации для клиентов. Анализ функционала этих сайтов показал, что ключевыми элементами являются таблицы, содержащие информацию об автомобилях, заказах, менеджерах, клиентах и отзывах.

На основе этого анализа была разработана архитектура проекта, описано взаимодействие приложения с базой данных, составлена диаграмма вариантов использования. 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 данных для таблиц «ORDERCAR», «ORDERSPAREPART» и «REVIEWS», были продемонстрированы с помощью листингов кода. Эти элементы обеспечили эффективное управление данными и улучшили взаимодействие с базой данных.

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

Для обеспечения высокой производительности и надежности базы данных была использована технология мониторинга состояния СУБД PostgreSQL. В проекте применялось графическое средство pgAdmin 4, которое упрощает задачи мониторинга и анализа работы базы данных в реальном времени. Для сбора и анализа статистики запросов использовалось представление pg\_stat\_statements, которое предоставляет ценные данные для оптимизации работы СУБД.

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

Список использованных литературных источников

1. JetCar [Электронный ресурс] / Режим доступа: https://jetcar24.ru/ – Дата доступа: 20.03.2024.
2. SoundClick [Электронный ресурс] / Режим доступа: https://royalmotors.ru/ – 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

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