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

Факультет информационных технологий

Кафедра информационных систем и технологий

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных для магазина телефонов с использованием технологии полнотекстового поиска»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П. С. Синяк

подпись, дата

**Руководитель**

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

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

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

дата, подпись

Курсовой проект защищен с оценкой

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

подпись дата инициалы и фамилия

Минск 2022

**Содержание**

[Введение 4](#_Toc134919185)

[1 Анализ требований к программному средству 5](#_Toc134919186)

[1.1 Аналитический обзор аналогов 5](#_Toc134919187)

[1.1.1 Аналог MobiStore 5](#_Toc134919188)

[1.1.2 Аналог Samsung 7](#_Toc134919189)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc134919190)

[1.3 Вывод 9](#_Toc134919191)

[2 Разработка архитектуры проекта 10](#_Toc134919192)

[2.1 Обобщенная структура управлением приложения 10](#_Toc134919193)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc134919194)

[3 Разработка модели базы данных 13](#_Toc134919195)

[3.1 Создание необходимых объектов 13](#_Toc134919196)

[3.1.1 Представления базы данных 14](#_Toc134919197)

[3.1.2 Индексы базы данных 15](#_Toc134919198)

[3.1.3 Триггеры базы данных 15](#_Toc134919199)

[3.2 Описание используемой технологии 17](#_Toc134919200)

[3.3 Вывод 18](#_Toc134919201)

[4 Установка, настройка и использование PosgtreSQL 14.5 19](#_Toc134919202)

[4.1 Установка PostgreSQL 19](#_Toc134919203)

[4.2 Создание таблиц 19](#_Toc134919204)

[4.3 Создание ролей для разграничения 20](#_Toc134919205)

[4.4 Создание пакетов процедур для базы данных 22](#_Toc134919206)

[4.4.1 Выборка данных из таблиц 22](#_Toc134919207)

[4.4.2 Выборка данных по поисковому запросу 23](#_Toc134919208)

[4.4.3 Заполнение таблиц 100 000 строк 24](#_Toc134919209)

[4.4.4 Добавление данных в таблицы 25](#_Toc134919210)

[4.4.5 Удаление данных из таблиц 26](#_Toc134919211)

[4.4.6 Изменение данных в таблицы 26](#_Toc134919212)

[4.4.7 Дополнительные функции 27](#_Toc134919213)

[4.5 Описание процедур экспорта и импорта 28](#_Toc134919214)

[4.6 Вывод 29](#_Toc134919215)

[5 Тестирование 30](#_Toc134919216)

[5.1 Тестирование производительности базы данных 30](#_Toc134919217)

[5.2 Вывод 31](#_Toc134919218)

[6 Руководство по использованию программного средства 32](#_Toc134919219)

[6.1 Руководство пользователя 32](#_Toc134919220)

[6.2 Вывод 33](#_Toc134919221)

[Заключение 34](#_Toc134919222)

[Список литературных источников 35](#_Toc134919223)

[**Приложение А** 36](#_Toc134919224)

# Введение

Целью данной курсовой работы является разработка базы данных для магазина телефонов. В рамках данной работы будут рассмотрены основные аспекты работы магазина по продаже телефонов, такие как ассортимент товаров, заказы, клиенты, и создана база данных, позволяющая управлять этими аспектами.

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

Задачи работы включают в себя анализ требований к базе данных, разработку структуры базы данных, создание модели данных, реализацию базы данных и ее тестирование.

Требования к курсовой работе включают в себя выполнение следующих шагов:

* Анализ требований к базе данных магазина телефонов.
* Разработка структуры базы данных, включая ее таблицы, связи и ограничения.
* Создание модели данных для магазина телефонов, опираясь на требования, определенные на предыдущих этапах.
* Реализация базы данных на СУБД PostgreSQL.
* Тестирование базы данных.

Также для обеспечения функциональности приложения используются технологии полнотекстового поиска.

Результатом данной работы должна быть полнофункциональная база данных магазина по продаже телефонов, которая позволит ознакомится с товаром в полной мере, осуществить заказ, а также выбрать дополнительную комплектацию для товара.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Трудно представить в наше время человека, который не использует телефон. К тому же сейчас телефон используют не только для звонков, но и для работы, развлечений, покупок. Помимо этого, телефон предоставляет прямой доступ к интернету, без которого сейчас трудно представить свою жизнь. Именно поэтому на сегодняшний день имеется огромный ассортимент телефонов.

Как было сказано существует большое разнообразие телефонов на данный момент и существует огромные компании по их производству. Таким образом есть основные компании по производству телефонов, которые и борются за рынок. Именно поэтому сайты, которые могли бы предоставить все разнообразие телефонов и ознакомить клиента с товаром сейчас пользуются большим спросом.

### 1.1.1 Аналог MobiStore

MobiStore [1] – это сайт по продаже различной техники, но основная их специализация это продажа смартфонов и различных аксессуаров для них. Пример интерфейса данного сервиса представлен на рисунке 1.1.

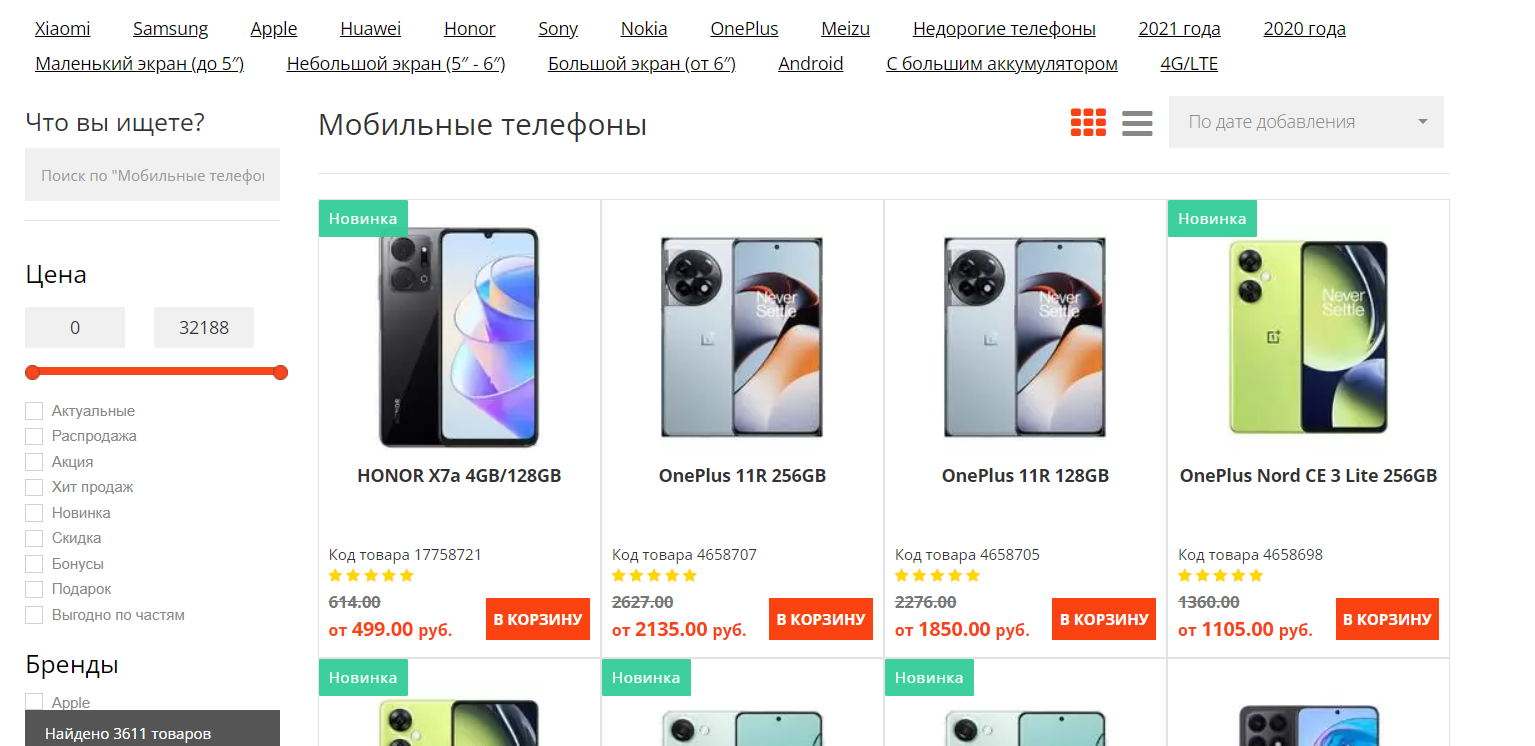


Рисунок 1.1 – Страница с каталогом товаров сервиса MobiStore

Сервис предоставляет возможность отсортировать и отфильтровать ассортимент товаров по различным параметрам. Например, по бренду, дате выхода, операционной системе, цене и так далее. Пример инструментов по фильтрации и сортировке представлен на рисунке 1.2.

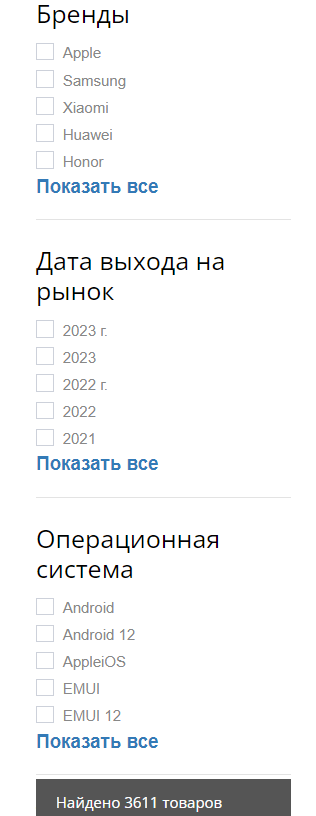


Рисунок 1.1 – Пример инструментов для фильтрации

Пользователи могут ознакомится с товаром. Посмотреть все его характеристики, его изображения с различных ракурсов, добавить товар для сравнения, выбрать цвет, выбрать аксессуары и добавить в корзину. Пример страницы с информацией о товаре представлена на рисунке 1.3.

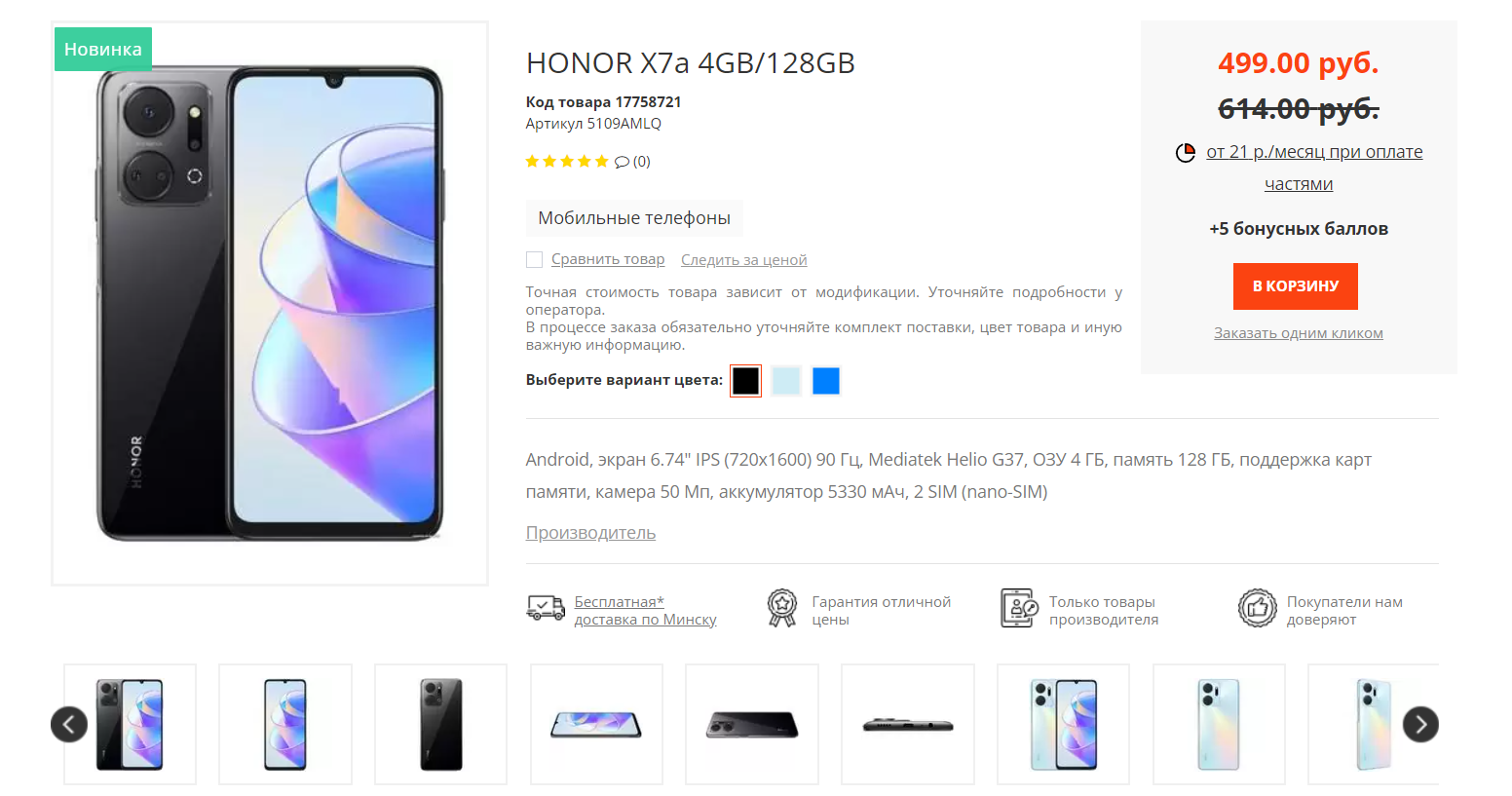


Рисунок 1.3 – Страница с информацией о товаре

После добавления товара в корзину, можно либо продолжить ознакомления с товарами, либо перейти в корзину для оформления заказа. Пример оформления корзины представлен на рисунке 1.4.

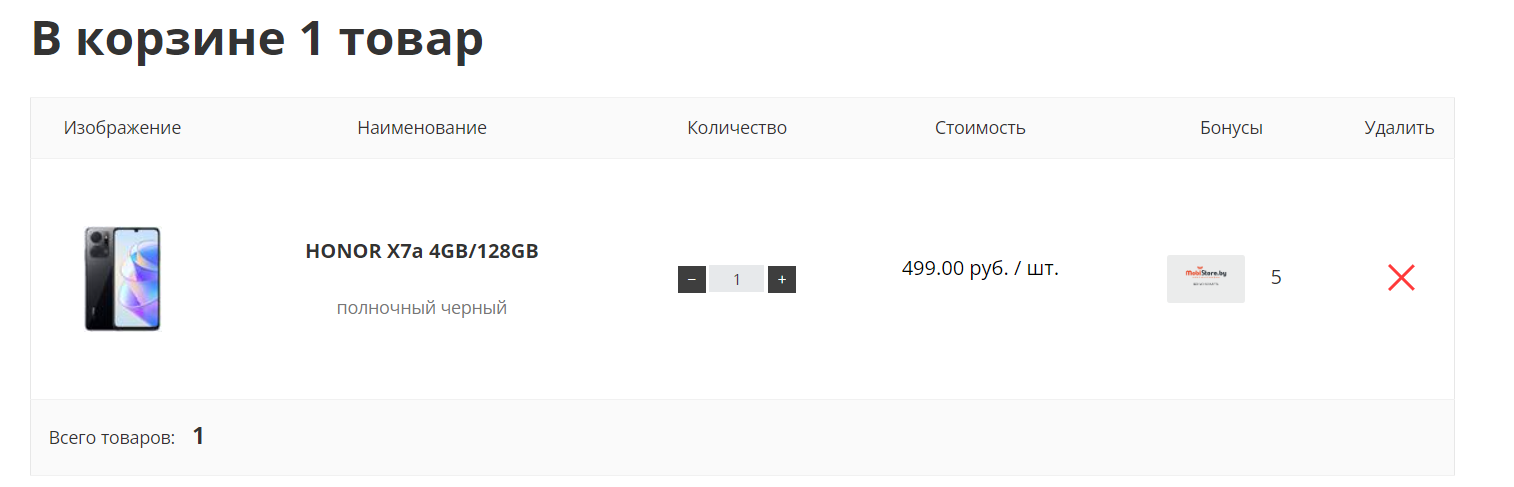


Рисунок 1.4 – Пример оформления корзины

Таким образом был рассмотрен сайт по продаже телефонов различных брендов.

### 1.1.2 Аналог Samsung

Samsung [2] – это одна из самых больших компаний по продаже техники, в частности смартфонов. Они также предоставляют возможно выбрать аксессуары для телефона только уже своего бренда. Пример интерфейса данного сервиса представлен на рисунке1.5.

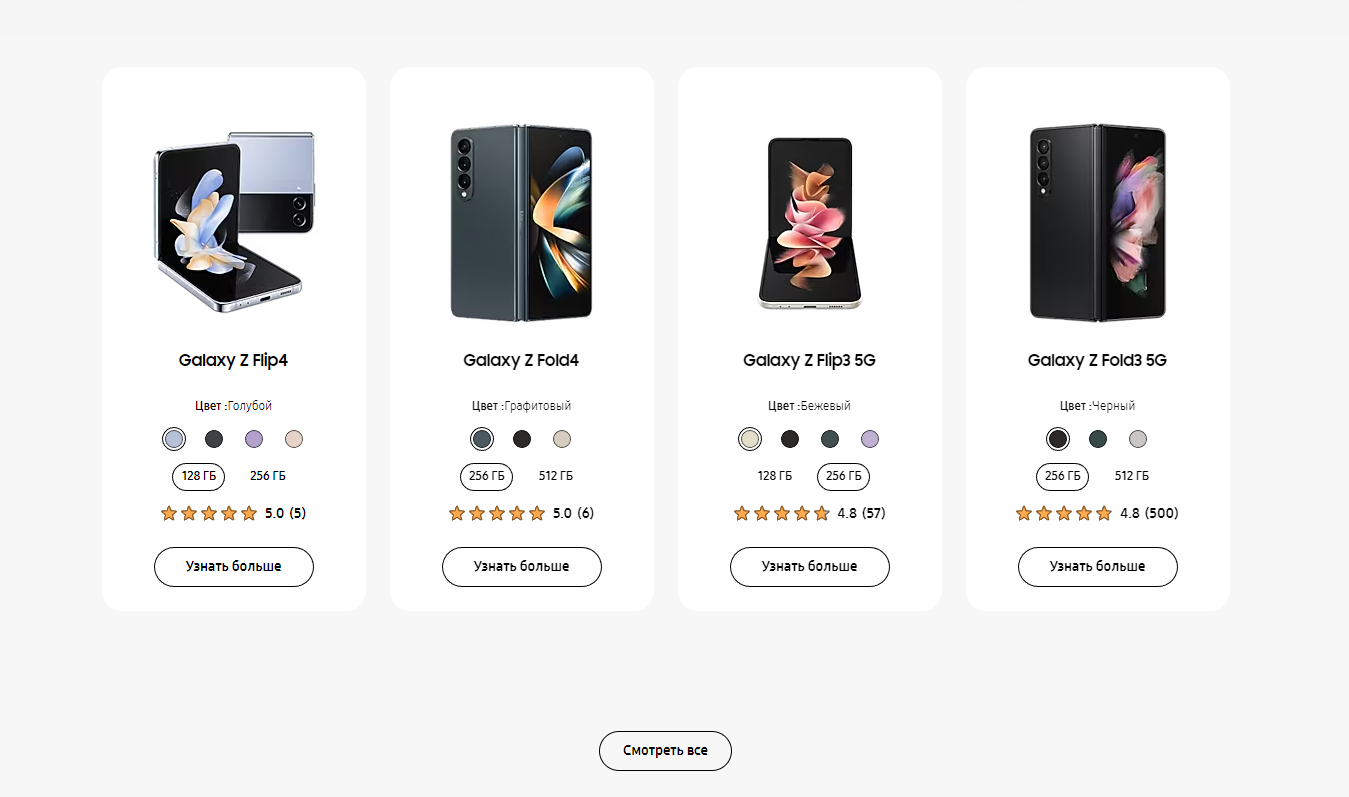


Рисунок 1.5 – Пример интерфейса сервиса Sumsung

На сервисе также можно ознакомится с товаром, правда из-за того, что это уже сервис производителя, то и оформление каждого продукта выглядит как его презентация. Также можно рассмотреть товаров с различных ракурсов, поменять цвет и добавить товар в корзину.

В данном разделе были проведены аналитические обзоры двух популярных музыкальных сервисов - MobiStore и Samsung. Оба сервиса предоставляют пользователям возможность ознакомится с товаром и совершить заказ.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для магазина телефонов функциональные требования могут включать в себя функции для хранения информации о смартфонах и пользователях, поиска телефонов по категориям и критериям, добавление товара в корзину.

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

Роли пользователя — это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Client.
* Manager.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.6.

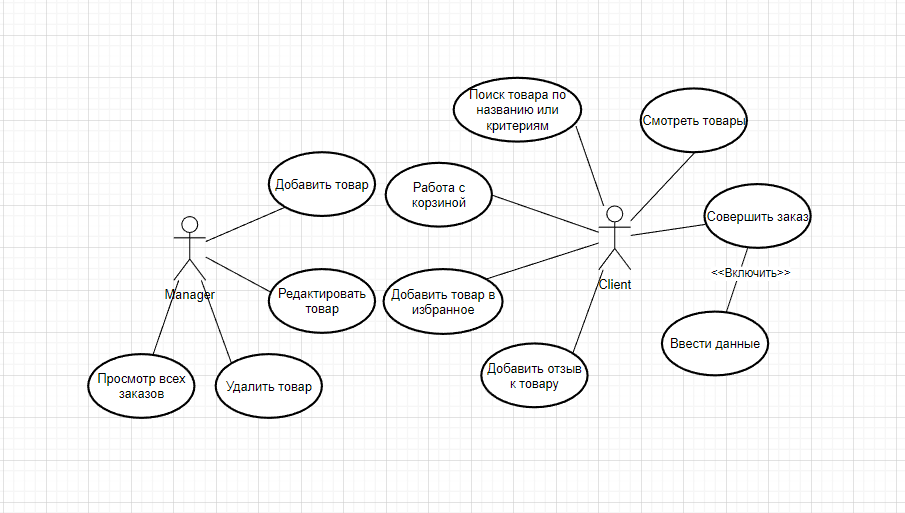


Рисунок 1.6 – UML диаграмма вариантов использования

Роль Client получает возможность просматривать товары, осуществлять поиск товаров по названию или критериям, добавлять товар в избранное и оставить отзыв к товару. Помимо этого, пользователь может добавить товар в корзину и совершить заказ, что включает в себя заполнение всех необходимых данных о пользователе.

Роль Manager заключается в управлении каталогом товаров. Менеджер может добавлять новый товар, удалять товар и изменять товар. Помимо этого, он сможет посмотреть все уже выполненные заказы для анализа работы магазина.

В данном разделе были определены роли пользователей и разработаны варианты использования системы в зависимости от этих ролей. Пользователь получает доступ к функциям системы, таким как просмотр товаров, добавление в корзину и совершение заказа, а менеджер может изменять каталог товаров, добавляя, удаляя или изменяя информацию.

## 1.3 Вывод

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

1. Разработка архитектуры проекта

## 2.1 Обобщенная структура управлением приложения

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

В функциональность приложения для продажи телефонов может входить возможность быстрого и удобного поиска определенной модели телефона по параметрам, таким как бренд, год выпуска, операционная система и т.д. Также пользователи должны иметь возможность добавлять товары в корзину и взаимодействовать с ней. Также пользователи смогут добавлять товары в избранное для того, чтобы пока что отложить покупку товара.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

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

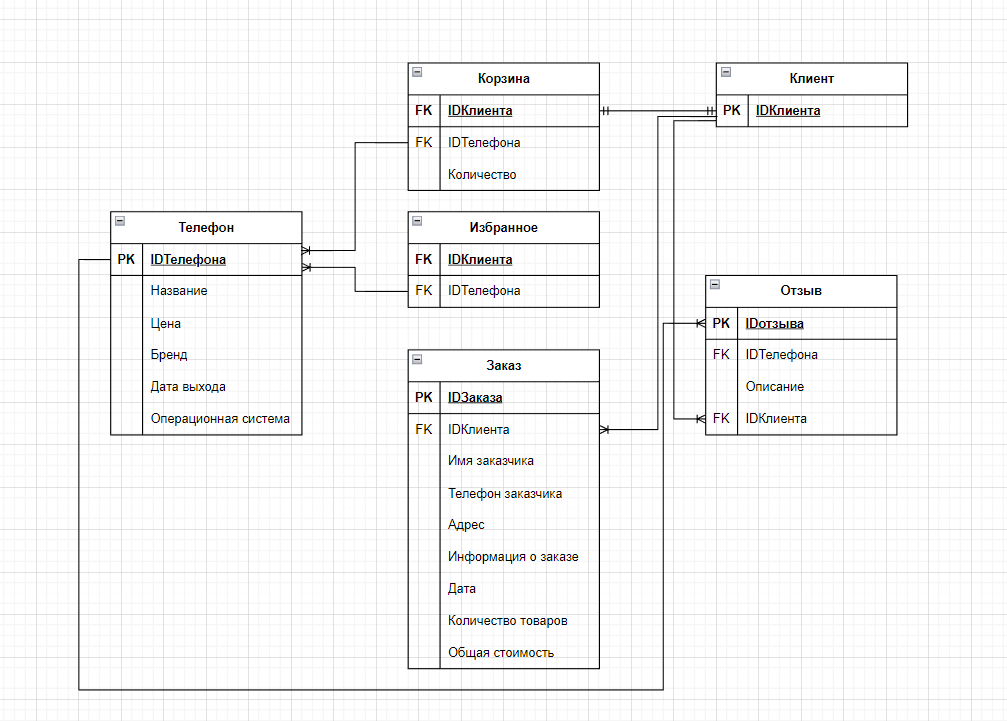


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному". Например, таблица Корзина связана с таблицей Клиент через внешний ключ IDКлиента, а таблица Телефон связана с таблицей Корзина через внешний ключ IDТелефона.

**2.3 Описание информационных объектов**

Для реализации базы данных было разработано 6 таблицы. В структуру схемы базы данных для проекта входят следующие таблицы: Телефон, Корзина, Клиент, Заказ, Избранное и Отзывы. Ниже будет описание про каждую из них более подробно.

Таблица Client содержит определенную корзину. Имеющиеся поля: client\_id (идентификатор пользователя, типа SERIAL, первичный ключ).

Таблица Mobile содержит информацию о телефонах. Имеющиеся поля: id\_mobile (идентификатор телефона, типа SERIAL, первичный ключ), mobile\_name (модель телефона, типа varchar(100)), mobile\_price (стоимость телефона типа int), mobile\_brend (название бренда типа varchar(100)), mobile\_os (название операционной системы телефона типа varchar(100)), mobile\_date (дата выхода на рынок типа date).

Таблица Basket содержит информацию о товарах, которые добавил пользователь. Имеющиеся поля: id\_mobile (идентификатор модели телеофона, тип SERIAL, внешний ключ ссылающийся на id\_mobile в таблице Mobile), counter (количество товаров одного типа, типа INT), client\_id (идентификатор пользователя, которому принадлежит данная корзина, тип SERIAL, внешний ключ ссылающийся на client\_id в таблице Client).

Таблица Ordert содержит информацию о заказе. Имеющиеся поля: id\_ordert (идентификатор заказа, типа SERIAL, первичный ключ), client\_id(идентификатор пользователя, тип SERIAL, внешний ключ, ссылающийся на client\_id в таблице Client), client\_name (имя клиента, тип varchar(100)), client\_phone (телефон заказчика, тип varchar(20)), date\_order(дата совершения заказа, тип TIMESTAMP), address\_order(информация о заказа, тип text), all\_count(общее количество товаров в заказе, тип int), all\_price(общая стоимость всех товаров в заказе, тип int).

Таблица Favorites содержит информацию о понравившихся товарах, которые пользователь добавил в избранное. Имеющиеся поля: id\_mobile(идентификатор телефона, тип SERIAL, внешний ключ ссылающийся на таблицу Mobile), client\_id(идентификатор пользователя, тип SERIAL, внешний ключ ссылающийся на таблицу Client).

Таблица Reviews содержит информацию о отзывах к телефону. Имеющиеся поля: id\_review(идентификатор отзыва, тип SERIAL, первичный ключ), id\_mobile(идентификатор телефона, тип SERIAL, внешний ключ ссылающийся на таблицу Mobile), client\_id(идентификатор пользователя, тип SERIAL, внешний ключ ссылающийся на таблицу Client), description(отзыв о самом продукте, тип text)

**2.4 Вывод**

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться. В данном проекте были описаны объекты, такие как пользователи, телефоны, корзина и заказы.

В целом, разработка архитектуры проекта является важным шагом в процессе разработки приложения, так как это позволяет определить необходимые компоненты и информационные объекты, которые помогут создать функциональное и эффективное приложение.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для магазина телефонов необходимо создать несколько таблиц, которые будут хранить данные о телефонах и заказах.

Для магазина телефонов необходимо создать следующие таблицы:

Client – таблица содержит идентификатор клиента. Так как на сервисе будет отсутствовать возможно регистрации, то данная таблица помогает привязать корзину к пользователю

Mobile – таблица содержащая информацию о телефонах. В этой таблице будет хранится вся информация, которая касается телефонов, например, бренд, название модели, операционная система и т.д.

Basket - таблица, содержащая информацию о телефонах, которые добавил пользователь. В этой таблице будет храниться идентификатор телефона, которого добавил клиент, идентификатор самого клиента, для связи корзины с клиентом и количество товара одного типа, для того, чтобы просто увеличивать счетчик товаров в случае добавления двух одинаковых телефонов.

Ordert - таблица, содержащая информацию о заказах. Эта таблица хранит в себе информацию после совершения оплаты. Сюда входит идентификатор самой таблицы заказов, идентификатор клиента, имя и телефон клиента, так как регистрация на сервисе отсутствует, то краткая информация о пользователе будет заполняться именно в момент оплаты, дата совершения заказа, адрес заказа и информация о заказе, то есть перечисление названий моделей телефонов, которые были в корзине. Помимо этого, есть информация о количестве товаров в заказе и итоговой стоимости для того, чтобы можно было осуществить анализ работы магазина.

Favorites – таблица, содержащая информацию о телефонах, которые пользователю понравились или которые просто решили отложить. Сюда входит идентификатор самого телефона и идентификатор пользователя для связи самоой таблицы избранное с клиентом.

Reviews – таблица, содержащая отзыв клиента о телефоне. Сюда входит идентификатор самого отзыва, идентификатор телефона, к которому этот отзыв относится, идентификатор клиента, который оставил данных отзыв и текст самого отзыва. Стоит отметить, что под одной и той же моделью телефона, пользователь сможет оставить только один отзыв.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, можно создать индексы на столбцах mobile\_name в таблице Mobile, чтобы быстро находить информацию о телефоне по его названию.

Для более удобной работы с базой данных можно создать несколько функций и процедур. Например, можно создать функцию для поиска телефонов по заданным критериям, таким как бренд, дата выхода на рынок, операционная система и т.д. Также можно создать процедуру для добавления новых товаров в базу данных

### 3.1.1 Представления базы данных

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

В данном проекте были созданы два представления:

* mobile\_info, которое содержит информацию о телефонах;
* info\_orders, которое содержит информацию о всех заказах.

Представление mobile\_info было создано для того, чтобы получить полную информацию о всем ассортименте товаров. Оно будет на листинге 3.1.

|  |
| --- |
| CREATE VIEW MOBILE\_INFO AS  SELECT p.mobile\_name,  p.mobile\_price,  p.mobile\_brend,  p.mobile\_os,  p.mobile\_date  FROM MOBILE p |

Листинг 3.1 – Представление MOBILE\_INFO

Представление ordert\_info было создано для того, чтобы получить информацию заказах. Оно будет на листинге 3.2.

|  |
| --- |
| CREATE OR REPLACE VIEW public.ordert\_info  AS  SELECT o.id\_ordert,  o.client\_id,  o.client\_name,  o.client\_phone,  o.date\_order,  o.address\_order,  o.information\_order,  o.all\_count,  o.all\_price  FROM ordert o; |

Листинг 3.2 – Представление MOBILE\_INFO

Именно по этому представлению можно будет проанализировать работу сервиса, то есть вычислить количество проданных товаров и выручку.

### 3.1.2 Индексы базы данных

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

Для того, чтобы быстро находить телефоны по их названию, был создан индекс idx\_mob на таблице Mobile по полю mobile\_name. Сами индексы представлены на листинге 3.3.

|  |
| --- |
| CREATE INDEX idx\_mob ON Mobile(mobile\_name);  CREATE INDEX idx\_mob\_id ON Mobile(id\_mobile);  CREATE INDEX idx\_ id\_ordert ON Ordert(id\_ordert);  create index if not exists idx\_tsv\_mobile\_name on Mobile using gin(text\_tsv) |

Листинг 3.3 – Индексы базы данных

Кроме индексов idx\_mob в базе данных также присутствуют индексы на полях id\_mobile, и id\_ordert таблиц Mobile и Ordert соответственно.

Индекс id\_mobile на таблице Mobile был создан для ускорения работы с другими таблицами, так как он используется как внешний ключ у многих таблиц, так как данный сервис связан именно с телефонами.

Индекс id\_ordert на таблице Ordert может использоваться для быстрого поиска определенного заказа по его ID.

Индекс idx\_tsv\_mobile\_name необходим для ускорения поиска телефона, используя технологию полнотекстного поиска.

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

### 3.1.3 Триггеры базы данных

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

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

Первый триггер, созданный в таблице Ordert, проверяет, чтобы номер телефона был равен 13 символам. Если длина не соответствует, то выводится исключения с текстом “Телефон должен содержать 13 символов”. Триггер создается с помощью функции FUNC\_FOR\_custom\_phone (), которая возвращает значение NEW (новые значения строк в таблице) при выполнении вставки или обновления. Скрипт триггера будет представлен на листинге 3.4.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_custom\_phone() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.client\_phone) < 13 or CHAR\_LENGTH(NEW.client\_phone)>13 THEN  RAISE EXCEPTION 'Телефон должен содеражть 13 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_custom\_phone  BEFORE INSERT OR UPDATE ON Ordert  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_custom\_phone(); |

Листинг 3.4 – Скрипт триггера TR\_custom\_phone

Второй триггер проверяет, что длина имени ложна быть не меньше 2 сиволов. Если длина имени меньше, то выводится ошибка “Имя должно содержать минимум 2 символа”. Триггер создается с помощью функции FUNC\_FOR\_custom\_name(), которая также возвращает значение NEW при выполнении вставки или обновления. Скрипт триггера будет представлен на листинге 3.5.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_custom\_phone() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.client\_phone) < 13 or CHAR\_LENGTH(NEW.client\_phone)>13 THEN  RAISE EXCEPTION 'Телефон должен содеражть 13 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_custom\_phone  BEFORE INSERT OR UPDATE ON Ordert  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_custom\_phone(); |

Листинг 3.5 – Скрип триггера c TR\_custom\_name

Ещё один триггер будет затрагивать поле address\_order в таблице Ordert. Так как нужно проверить на правильность написания адреса. Если длина адреса будет меньше 5 символов, то вызовется ошибка с сообщением “ Адрес должен быть больше 4 символов”. Триггер создается с помощью функции FUNC\_FOR\_address\_order (), которая также возвращает значение NEW при выполнении вставки или обновления. Скрипт триггера будет представлен на листинге 3.6.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_address\_order() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.address\_order) < 5 THEN  RAISE EXCEPTION 'Адрес должен быть больше 4 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_address\_order  BEFORE INSERT OR UPDATE ON Ordert  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_address\_order(); |

Листинг 3.6 – Скрип триггера c TR\_address\_order

Таким образом, оба триггера выполняют валидацию данных перед их вставкой или обновлением в таблице Ordert, обеспечивая целостность данных.

## 3.2 Описание используемой технологии

В данной базе данных используется технология полнотекстового поиска. Для полнотекстового поиска в PostgreSQL предусмотрены специальные типы данных: tsvector и tsquery.

Тип tsvector — представляет документ в виде, оптимизированном для текстового поиска. По сути это нормализованная строка, по которой будет производиться поиск. Под нормализацией следует понимать такие процедуры как выкидывание стоп-слов, вырезание окончаний слов, приведение к нижнему регистру и т.д. Для нормализации строки используется процедура to\_tsvector. Поэтому для таблицы Mobile следует добавить столбец text\_tsv и заполнить его. Пример нормализации представлен в листинге 3.7.

|  |
| --- |
| alter table Mobile add column text\_tsv tsvector  update Mobile set text\_tsv = to\_tsvector('russian',mobile\_name) |

Листинг 3.7 – Пример нормализации

Тип tsquery – используется для представления запросов поиска. Для преобразования поисковых запросов используется процедура plainto\_tsquery. Пример использование данной процедуры приведен в листинге 3.8.

|  |
| --- |
| select MOBILE\_INFO.mobile\_name,MOBILE\_INFO.mobile\_price,  MOBILE\_INFO.mobile\_brend, MOBILE\_INFO.mobile\_os,  MOBILE\_INFO.mobile\_date FROM MOBILE\_INFO where MOBILE\_INFO.text\_tsv @@  plainto\_tsquery('russian',mobile\_name\_arg); |

Листинг 3.8 – Пример процедуры plainto\_tsquery

Для того, чтобы поиск осуществлялся быстрее следует поставить индекс вида gin, так как он рассчитан на случай, когда чаще выполняется поиск чем запись. Пример создания индекса представлен в листинге 3.9.

|  |
| --- |
| create index if not exists idx\_tsv\_mobile\_name on Mobile using gin(text\_tsv) |

Листинг 3.9 – Пример инедекса idx\_tsv\_mobile

Таким образом, мы применили технологию полнотекстового поиска на столбец mobile\_name. Теперь искать телефон по названию станет возможно и без знания точного его названия.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для магазина телефонов. Также была описана использованная технология полнотекстового поиска. Идея полнотекстового поиска состоит в том, чтобы затратить время на обработку документа один раз и сохранить время при поиске, использовать специальные программы-словари для нормализации слов, чтобы не заботиться, например, о формах слов, учитывать информацию о важности различных атрибутов документа и положения слова из запроса в документе для ранжирования найденных документов.

4 Установка, настройка и использование PosgtreSQL 14.5

## 4.1 Установка PostgreSQL

PostgreSQL - это мощная реляционная база данных с открытым исходным кодом, которая предлагает множество функций, таких как многопоточность, транзакционность, контроль целостности данных, масштабируемость и многое другое. PostgreSQL широко используется в коммерческих и научных проектах, а также веб-приложениях и мобильных приложениях.

После установки PostgreSQL на сервер, была произведена конфигурация сервера для оптимальной работы с базой данных. Затем была создана база данных с названием Kurshach, которая будет использоваться в дальнейшем для хранения данных магазина телефонов.

## 4.2 Создание таблиц

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

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

|  |
| --- |
| create tablespace TS\_USER  location 'D:\BDKURSACH\TS\_USER';  create tablespace TS\_TOVAR  location 'D:\BDKURSACH\TS\_TOVAR'; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим шесть основных таблиц: Mobile, Client, Basket, Ordert, Favorites, Review.

Таблицы Client, Basket, Favorites будут храниться в табличном пространстве TS\_USER, таблица Ordert, Mobile, Review - в табличном пространстве TS\_TOVAR.

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

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

Другой тип связи - это связь "один к одному" (one-to-one), которая определяет отношение между множеством записи в одной таблице к записи в другой таблице. Например, у каждого пользователя может быть только одна корзина, и каждая корзина может принадлежать только одному пользователю. Для этого мы добавим в таблицу Client внешний ключ на таблицу Basket. Скрипт создание таблицы Mobile будет представлен на листинге 4.2.

|  |
| --- |
| create table Mobile(  id\_mobile serial primary key,  mobile\_name varchar(100) not null,  mobile\_price int not null check (mobile\_price>0),  mobile\_brend varchar(100),  mobile\_os varchar(100) not null,  mobile\_date date not null  ) tablespace TS\_TOVAR; |

Листинг 4.2 – Cкрипт создание таблицы Mobile

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

## Создание ролей для разграничения

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

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

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

|  |
| --- |
| grant connect on database "Kursach" to RoleDevelop;  GRANT ALL ON SCHEMA public TO RoleDevelop;  grant all privileges on database "Kursach" to RoleDevelop;  grant all privileges on tablespace TS\_USER to RoleDevelop;  grant all privileges on tablespace TS\_TOVAR to RoleDevelop; |

Листинг 4.3 – Привилегии, выданные роли RoleDevelop

Роль RoleManager имеет права на выполнение различных функций и процедур, связанных с управлением таблицы с телефонами в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления телефонов. Выданные привилегии роли RoleManager можно увидеть на листинге 4.4.

|  |
| --- |
| GRANT EXECUTE ON PROCEDURE add\_mobile TO RoleManager;  GRANT EXECUTE ON PROCEDURE update\_mobile TO RoleManager;  GRANT EXECUTE ON PROCEDURE delete\_mobile TO RoleManager;  GRANT EXECUTE ON FUNCTION ANALIZE TO RoleManager; |

Листинг 4.4 – Привилегии, выданные роли RoleManager

Роль RoleClient может просматривать таблицу телефонов. Кроме того, RoleClient имеет возможность добавлять товар в корзину и совершить заказ. Выданные привилегии роли RoleClient можно увидеть на листинге 4.5.

|  |
| --- |
| GRANT EXECUTE ON PROCEDURE add\_mobile\_to\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE add\_mobile\_to\_favourites TO RoleClient;  GRANT EXECUTE ON PROCEDURE add\_review\_to\_mobile TO RoleClient;  GRANT EXECUTE ON PROCEDURE delete\_mobile\_count\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE delete\_mobile\_from\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE purchaise TO RoleClient;  GRANT EXECUTE ON FUNCTION search\_mobile\_by\_brend TO RoleClient;  GRANT EXECUTE ON FUNCTION search\_mobile\_by\_name TO RoleClient;  GRANT EXECUTE ON FUNCTION show\_mobile TO RoleClient; |

Листинг 4.5 – Привилегии, выданные роли RoleClient

Таким образом, были созданы три роли для ограничения доступа к базе данных: RoleClient, RoleManager и RoleDevelop. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль RoleDevelop имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль RoleClient имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию и добавлять товар в корзину. Роль RoleManager имеет средний уровень привилегий и может выполнять функции, связанные с управлением товарами в базе данных.

## 4.4 Создание пакетов процедур для базы данных

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

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

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

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

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

### 4.4.1 Выборка данных из таблиц

Так как самая главная таблица – это Mobile, то должна быть функция для вывода всей информации из этой таблицы.

На листинге 4.6 будет функция SHOW\_MOBILE, которая предназначена для выборки всех телефонов из таблицы Mobile.

|  |
| --- |
| CREATE OR REPLACE FUNCTION SHOW\_MOBILE()  RETURNS TABLE(  id\_mobile int,  mobile\_name varchar(100),  mobile\_price integer,  mobile\_brend varchar(100),  mobile\_os varchar(100),  mobile\_date date  ) as $$  begin  RETURN QUERY  SELECT \* FROM Mobile;  end  $$ language PLPGSQL; |

Листинг 4.6 – Функция SHOW\_MOBILE

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

### 4.4.2 Выборка данных по поисковому запросу

Для поиска определенной модели телефона SEARCH\_MOBILE\_BY\_NAME. Данная функция принимает на вход поисковый запрос в виде текста и возвращает таблицу с данными о найденных телефонах. Функция представлена на листинге 4.7.

|  |
| --- |
| CREATE or replace FUNCTION SEARCH\_MOBILE\_BY\_NAME(mobile\_name\_arg varchar(100))  RETURNS TABLE(  mobile\_name varchar(100),  mobile\_price integer,  mobile\_brend varchar(100),  mobile\_os varchar(100),  mobile\_date date  ) as $$  begin  RETURN QUERY  select MOBILE\_INFO.mobile\_name,MOBILE\_INFO.mobile\_price,  MOBILE\_INFO.mobile\_brend, MOBILE\_INFO.mobile\_os,  MOBILE\_INFO.mobile\_date FROM MOBILE\_INFO where MOBILE\_INFO.text\_tsv @@  plainto\_tsquery('russian',mobile\_name\_arg);  end;  $$ language PLPGSQL; |

Листинг 4.7 – Функция SEARCH\_MOBILE\_BY\_NAME

Для поиска используется технология полнотекстного поиска. Поисковый запрос в виде текста передается в функцию в качестве аргумента mobile\_name\_arg. Затем возвращается таблица с данными о найденных телефонах.

Для поиска по брендам используется ещё одна функция SEARCH\_MOBILE\_BY\_BREND. Здесь уже поиск осуществляется с помощью оператора like. Функция представлена в листинге 4.8.

|  |
| --- |
| CREATE or replace FUNCTION SEARCH\_MOBILE\_BY\_BREND(mobile\_brend\_arg varchar(100))  RETURNS TABLE(  mobile\_name varchar(100),  mobile\_price integer,  mobile\_brend varchar(100),  mobile\_os varchar(100),  mobile\_date date  ) as $$  begin  RETURN QUERY  SELECT MOBILE\_INFO.mobile\_name,MOBILE\_INFO.mobile\_price,  MOBILE\_INFO.mobile\_brend, MOBILE\_INFO.mobile\_os,  MOBILE\_INFO.mobile\_date FROM MOBILE\_INFO  where MOBILE\_INFO.mobile\_brend like '%'|| mobile\_brend\_arg || '%';  end;  $$ language PLPGSQL; |

Листинг 4.7 – Функция SEARCH\_MOBILE\_BY\_BREND

Таким образом, были рассмотрены функции по поиску телефонов. Осуществить поиск можно либо по названию, либо по бренду.

### 4.4.3 Заполнение таблиц 100 000 строк

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

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_MOBILE()  RETURNS VOID AS $$  DECLARE  I INTEGER := 1;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Mobile (mobile\_name,mobile\_price,mobile\_brend,mobile\_os,  mobile\_date) values('SUWU',12312,'SUMSUNG','ANDROID','1923-03-22');  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.8 – Функция заполнения таблицы GENRE

Функция INSERT\_ MOBILE была создана для заполнения таблицы Mobile 100000 строками. В теле функции используется цикл WHILE, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу Mobile. Функция не возвращает значение и не принимает аргументов. Для выполнения функции необходимо выполнить SELECT INSERT\_ MOBILE ().

### 4.4.4 Добавление данных в таблицы

Были разработаны две процедуры для заполнения таблицы Mobile(ADD\_MOBILE) и для заполнения таблицы Basket(ADD\_MOBILE\_TO\_BASKET).

На листинге 4.9 будет функция ADD\_MOBILE, которая добавляет телефон в таблицу Mobile.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ADD\_MOBILE(mobile\_name\_arg varchar(100),mobile\_price\_arg int,mobile\_brend\_arg varchar(100),  mobile\_os\_arg varchar(100), mobile\_date\_arg date)  AS $$  DECLARE  mobile\_id\_add INTEGER;  mobile\_date\_arg date;  BEGIN  mobile\_date\_arg = current\_date;  INSERT INTO Mobile(mobile\_name,mobile\_price,mobile\_brend,mobile\_os ,mobile\_date)  VALUES(mobile\_name\_arg,mobile\_price\_arg,mobile\_brend\_arg,mobile\_os\_arg,mobile\_date\_arg);  update Mobile set text\_tsv = to\_tsvector('russian',mobile\_name);  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.9 – Функция добавления телефона

В процедуре добавления товара в корзину будет иметь значение именно поле количество, так как если добавляется телефон с тем же идентификатором, то будет увеличиваться именно это поле. Процедура представлена в листинге 4.10.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE ADD\_MOBILE\_TO\_BASKET(  mobile\_id\_arg INTEGER,  client\_id\_arg int  )  LANGUAGE PLPGSQL  AS $$  declare hh int;  BEGIN  hh = (select count(id\_mobile) from Basket where id\_mobile = mobile\_id\_arg);  if hh = 0 then  INSERT INTO Basket  VALUES (mobile\_id\_arg,client\_id\_arg,1);  elseif hh > 0 then  update Basket set counter = counter + 1 where id\_mobile = mobile\_id\_arg;  END if;  end;  $$; |

Листинг 4.9 – Функция заполнения таблицы Register

Все остальные процедуры будут похожи и предназначены для заполнения других таблиц.

### 4.4.5 Удаление данных из таблиц

Для удаления товара из таблиц были разработаны две процедуры: процедура по удалению телефона полностью из магазина(DELETE\_MOBILE) и процедура, которая удаляет телефон только из корзины(DELETE\_MOBILE\_FROM\_BASKET)

На листинге 4.11 будет процедура DELETE\_MOBILE.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_MOBILE( mobile\_id\_arg INTEGER) LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Basket WHERE Basket.id\_mobile = mobile\_id\_arg;  DELETE FROM Mobile WHERE Mobile.id\_mobile = mobile\_id\_arg;  END;  $$; |

Листинг 4.11 – Процедура для удаления Mobile

Процедура DELETE\_MOBILE\_FROM\_BASKET представлена в листинге 4.12.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_MOBILE\_FROM\_BASKET( mobile\_id\_arg INTEGER) LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Basket WHERE Basket.id\_mobile = mobile\_id\_arg;  END;  $$; |

Листинг 4.12 – Процедура для удаления Basket

### 4.4.6 Изменение данных в таблицы

Для изменения данных в таблицах была разработана функция, которая изменяет информацию о телефонах в таблице Mobile(UPDATE\_MOBILE).

На листинге 4.13 будет процедура UPDATE\_MOBILE.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_MOBILE(  id\_mobile\_arg int,  mobile\_name\_arg varchar(100),  mobile\_price\_arg int,  mobile\_brend\_arg varchar(100),  mobile\_os\_arg varchar(100),  mobile\_date\_arg date  ) LANGUAGE PLPGSQL AS $$  BEGIN  UPDATE Mobile  SET  mobile\_name = mobile\_name\_arg,  mobile\_price=mobile\_price\_arg,  mobile\_brend=mobile\_brend\_arg,  mobile\_os=mobile\_os\_arg,  mobile\_date=mobile\_date\_arg  WHERE Mobile.id\_mobile = id\_mobile\_arg;  END;  $$; |

Листинг 4.13 – Процедура для обновления товара

Таким образом была разработана процедура по изменению товара в таблице Mobile.

### 4.4.7 Дополнительные функции

Была также разработана функция по ANALIZE, которая выводит статистику работы магазина. Здесь будет выводится общее количество проданных телефонов и их общая стоимость. Функция представлена на листинге 4.14.

|  |
| --- |
| CREATE or replace FUNCTION ANALIZE()  RETURNS TABLE(  Все\_проданные\_товары bigint,  Общая\_выручка bigint  ) as $$  declare cont text;  begin  RETURN QUERY  select sum(all\_count), sum(all\_price) from ordert\_info;  end;  $$ language PLPGSQL; |

Листинг 4.14 – Функция ANALIZE

Функция использует представление ordert\_info, чтобы получить информацию о заказах.

## 4.5 Описание процедур экспорта и импорта

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

Для экспорта данных в формате XML была создана функция EXPORT\_ORDER\_TO\_XML\_FILE, которая принимает имя файла. Функция представлена на листинге 4.15.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.export\_orders\_to\_xml\_file(  file\_path text)  RETURNS void  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  xml\_data xml;  xml\_doc text;  BEGIN  SELECT table\_to\_xml('Mobile', true, false, '') INTO xml\_data;  xml\_doc := format('<?xml version="1.0" encoding="UTF-8"?>%s', xml\_data::text);  PERFORM pg\_file\_write(file\_path, xml\_doc, true);  END;  $BODY$;  ALTER FUNCTION public.export\_orders\_to\_xml\_file(text)  OWNER TO roledevelop; |

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

Для импорта данных из файла в формате XML была создана процедура IMPORT\_XML\_ORDERT которая принимает имя файла и импортирует данные из файла в таблицу Ordert. Функция представлена на листинге 4.16.

|  |
| --- |
| CREATE OR REPLACE FUNCTION import\_data\_from\_xml()  RETURNS VOID AS  $$  DECLARE  xml\_data XML;  BEGIN  SELECT XMLPARSE(DOCUMENT convert\_from(pg\_read\_binary\_file('D:/BDKURSACH/export.xml'), 'UTF8'))  INTO xml\_data;  END;  $$  LANGUAGE plpgsql; |

Листинг 4.16 – Функция import\_data\_from\_xml

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

## Вывод

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

Были также представлены процедуры экспорта и импорта данных в формате XML.

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

5 Тестирование

## 5.1 Тестирование производительности базы данных

Ключевым фактором для эффективности управленческих и коммерческих приложений является производительность базы данных. При медленном поиске или записи данных производительность приложения может значительно упасть. Для выявления причины плохой производительности необходимо проводить количественные измерения и определить узкие места производительности.

В PostgreSQL оптимизация запросом в основном заключается в построение индексов над таблицами, и изменением плана запроса.

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

Ниже на рисунке 5.1 представлен результат статистического анализа и компиляции PostgreSQL до добавления индекса.

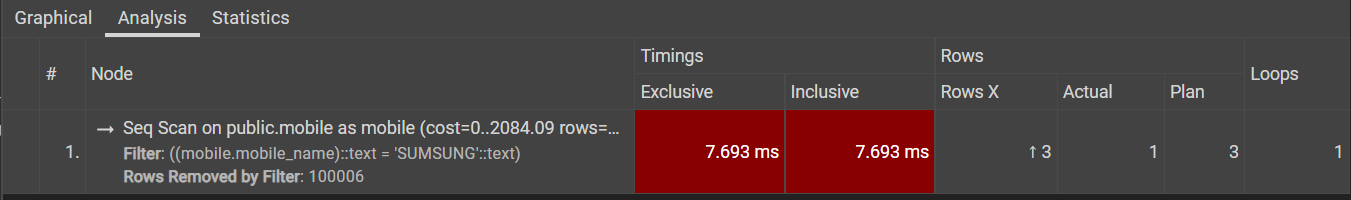


Рисунок 5.1 – Статический анализ перед созданием индекса покрытия

Как можно увидеть, затраченное время составляет примерно 6 с половиной милисекунд. Для оптимизации поиска был добавлен индекс покрытия, код которого представлен на листинге 5.1.

|  |
| --- |
| CREATE INDEX idx\_mob ON Mobile(mobile\_name); |

Листинг 5.1 – Индекс покрытия для таблицы Mobile

Ниже, на рисунке 5.2 представлен результат статического анализа и компиляции PostgreSQL после создания индекса покрытия.

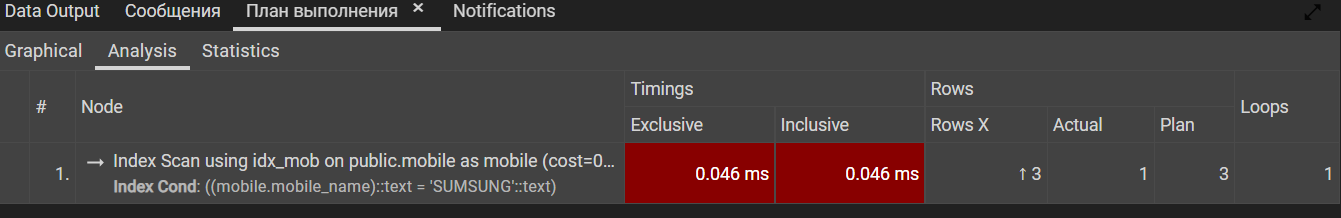


Рисунок 5.2 – Статический анализ после создания индекса покрытия

По результатам статистического анализа можно сказать, что построение индексов в данной таблице необходимо, так как во втором случае общая стоимость запроса уменьшилась.

## 5.2 Вывод

Для проведения тестирования в данном разделе была выбрана таблица Mobile с самым большим количеством строк в базе данных. Проведенный тест показал, что создание индекса на поле mobile\_name в разы повысило производительность запроса к таблице Mobile, сократив время выполнения запроса. Можно сделать вывод, что создание индексов на полях, по которым постоянно выполняются запросы, повышает производительность базы данных.

6 Руководство по использованию программного средства

## 6.1 Руководство пользователя

При заходе на сервис магазина телефонов пользователь автоматически получает идентификатор клиента. Так как пользователи не часто любят регистрироваться на подобных сайтах, где им нужно просто что-то заказать. После получения идентификатора у клиента уже будет собственная корзина и список избранное. При выходе из сервиса и обратном входе вся информация останется, так как при первом входе на сервис уже будет создаваться сессия, то есть уникальный идентификатор, который присваивается пользователю при первом взаимодействии с сайтом.

Далее клиент захочет ознакомиться с перечнем товаров. Для этого он должен вызвать функцию show\_mobile. Таким образом будут выведены все товары, которые предоставляет сервис. Если пользователь уже знает, зачем он пришел, то он вызвать функцию поиска по названию SEARCH\_MOBILE\_BY\_NAME. Помимо этого, он может отфильтровать товары по брендам, вызвав функцию SEARCH\_MOBILE\_BY\_BREND. Таким образом он ознакомиться с телефонами определенного бренда.

Если пользователю понравился телефон, и он хочет его добавить в корзину, то он выполняет процедуру add\_mobile\_to\_basket. Если же клиента захочет добавить ещё один точно такой же телефон, то в корзине просто увеличиться количество данного товара в корзине. Для просмотра содержимого корзины, клиент вызывает функцию SHOW\_BASKET. После этого клиент может взаимодействовать с корзиной. Если товаров одного типа несколько, то клиент может уменьшить количество данного товара в корзине, вызвав процедуру delete\_mobile\_count\_basket. Таким образом, если количество данного товара больше одного, то оно уменьшится на единицу, если же количество данного товара уже единица, то данную процедуру клиент вызвать не сможет. Поэтому если клиент захочет полностью удалить телефон из корзины, то он должен вызвать процедуру delete\_mobile\_from\_basket. Помимо этого, пользователь может добавить телефон в избранное (add\_mobile\_to\_favourites), посмотреть список избранных товаров (SHOW\_FAVOURITES), удалить телефон из списка избранных (DELETE\_MOBILE\_FROM\_FAVOURITES), оставить отзыв к товару (add\_review\_to\_mobile), но только один отзыв к одному товару. Ну и самая главная для сервиса процедура, которая может быть выполнена клиентом – это покупка (purchaise). В ходе данной процедуры, будет заполняться таблица заказов, а также очищаться корзина клиента.

Также на сервисе присутствует роль manager. В его перечень возможностей входит: добавление товара(add\_mobile), удаление товара(delete\_mobile), обновление товара(update\_mobile), ознакомление с товаром (show\_mobile). Также менеджер может ознакомиться со статистикой работы магазина вызвав функцию ANALIZE. Данная функция выведет все проданные товары и их общая стоимость за все заказы.

## 6.2 Вывод

В этом разделе был рассмотрен функционал приложения, доступный различным пользователям. Продемонстрировано отличие пользователей с различными ролями. Было показано, что пользователь может манипулировать с товарами, а именно добавлять их в корзину, в избранное, оставлять отзывы, а также осуществить покупку. Менеджер же имеет возможность манипулировать с таблицей телефоны для добавления, удаления и изменения товара. Помимо этого, он может ознакомиться со статистикой работы магазина.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для магазина телефонов с использованием технологии полнотекстового поиска в СУБД PostgreSQL.

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

Были реализованы процедуры для импорта и экспорта данных в формате XML, что обеспечило удобство использования и управления данными.

Одной из ключевых особенностей разработанной базы данных является технология полнотекстового поиска данных, что позволяет эффективно осуществлять поиск.

Также было проведено тестирование, по результату которого было выяснено, что добавление индексов в данные таблица положительно сказывается на скорость выполнения запросов.

Также было разработано руководство пользователя.

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

# Список литературных источников

1. Борис Новиков Основы технологий баз данных. Издание Litres, 2022. 233 с.

2. MobiStore [Электронный ресурс] / Режим доступа: https:// mobistore.by – Дата доступа: 14.04.2023.

3. Sumsung [Электронный ресурс] / Режим доступа: https://www.samsung.com/ – Дата доступа: 15.04.2023.

4. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.04.2023.

5.Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 18.04.2023.

6. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 18.04.2023.

**Приложение А**

|  |
| --- |
| --Roles--  create role RoleManager;  create role RoleClient;  create role RoleDevelop;  --Users--  create user Manager password '123';  grant RoleManager to Manager;  create user Client password '123';  grant RoleManager to Client;  create user Develop password '123';  grant RoleDevelop to Develop;  --Prev to Manager--  GRANT EXECUTE ON PROCEDURE add\_mobile TO RoleManager;  GRANT EXECUTE ON PROCEDURE update\_mobile TO RoleManager;  GRANT EXECUTE ON PROCEDURE delete\_mobile TO RoleManager;  GRANT EXECUTE ON FUNCTION ANALIZE TO RoleManager;  --Prev to Develop--  grant connect on database "Kursach" to RoleDevelop;  GRANT ALL ON SCHEMA public TO RoleDevelop;  grant all privileges on database "Kursach" to RoleDevelop;  grant all privileges on tablespace TS\_USER to RoleDevelop;  grant all privileges on tablespace TS\_TOVAR to RoleDevelop;  --Prev to Client--  GRANT EXECUTE ON PROCEDURE add\_mobile\_to\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE add\_mobile\_to\_favourites TO RoleClient;  GRANT EXECUTE ON PROCEDURE add\_review\_to\_mobile TO RoleClient;  GRANT EXECUTE ON PROCEDURE delete\_mobile\_count\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE delete\_mobile\_from\_basket TO RoleClient;  GRANT EXECUTE ON PROCEDURE purchaise TO RoleClient;  GRANT EXECUTE ON FUNCTION search\_mobile\_by\_brend TO RoleClient;  GRANT EXECUTE ON FUNCTION search\_mobile\_by\_name TO RoleClient;  GRANT EXECUTE ON FUNCTION show\_mobile TO RoleClient;  --Tablespaces--  create tablespace TS\_USER  location 'D:\BDKURSACH\TS\_USER';  create tablespace TS\_TOVAR  location 'D:\BDKURSACH\TS\_TOVAR';  --Tables--  create table Client(  client\_id serial primary key  ) tablespace TS\_USER;  create table Basket(  id\_mobile serial,  counter int check (counter=0 or counter >0),  client\_id serial,  constraint FK\_Basket\_Mobile foreign key (id\_mobile) references Mobile(id\_mobile),  constraint FK\_Basket\_Client foreign key (client\_id) references Client(client\_id)  ) tablespace TS\_USER;  create table Mobile(  id\_mobile serial primary key,  mobile\_name varchar(100) not null,  mobile\_price int not null check (mobile\_price>0),  mobile\_brend varchar(100),  mobile\_os varchar(100) not null,  mobile\_date date not null  ) tablespace TS\_TOVAR;  create table Ordert(  id\_ordert serial primary key,  client\_id serial not null,  client\_name varchar(100) not null,  client\_phone varchar(20) not null,  date\_order TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  address\_order varchar(100) not null,  information\_order text not null,  all\_count int not null,  all\_price int not null,  constraint FK\_Ordert\_Client foreign key (client\_id) references Client(client\_id)  ) tablespace TS\_TOVAR;  create table Favorites(  id\_mobile serial not null,  client\_id serial not null,  constraint FK\_Favorites\_Mobile foreign key (id\_mobile) references Mobile(id\_mobile),  constraint FK\_Favorites\_Client foreign key (client\_id) references Client(client\_id)  ) tablespace TS\_USER;  create table Reviews(  id\_review serial primary key,  id\_mobile serial not null,  client\_id serial not null,  description text not null,  constraint FK\_Reviews\_Mobile foreign key (id\_mobile) references Mobile(id\_mobile),  constraint FK\_Reviews\_Client foreign key (client\_id) references Client(client\_id)  ) tablespace TS\_TOVAR;  CREATE INDEX idx\_mob ON Mobile(mobile\_name);  --Создание функции добавления товара--  CREATE OR REPLACE PROCEDURE ADD\_MOBILE(mobile\_name\_arg varchar(100),mobile\_price\_arg int,mobile\_brend\_arg varchar(100),  mobile\_os\_arg varchar(100), mobile\_date\_arg date)  AS $$  DECLARE  mobile\_id\_add INTEGER;  mobile\_date\_arg date;  BEGIN  mobile\_date\_arg = current\_date;  INSERT INTO Mobile(mobile\_name,mobile\_price,mobile\_brend,mobile\_os ,mobile\_date)  VALUES(mobile\_name\_arg,mobile\_price\_arg,mobile\_brend\_arg,mobile\_os\_arg,mobile\_date\_arg);  update Mobile set text\_tsv = to\_tsvector('russian',mobile\_name);  END;  $$ LANGUAGE PLPGSQL;  call ADD\_MOBILE('dfsd',12312,'SUMSUNG','ANDROID','1923-03-22');  drop function ADD\_MOBILE;  select \* from Mobile;  --Создание функции для просмотра телефонов--  CREATE OR REPLACE FUNCTION SHOW\_MOBILE()  RETURNS TABLE(  id\_mobile\_arg int,  mobile\_name\_arg varchar(100),  mobile\_price\_arg integer,  mobile\_brend\_arg varchar(100),  mobile\_os\_arg varchar(100),  mobile\_date\_arg date  ) as $$  begin  RETURN QUERY  SELECT id\_mobile,mobile\_name,mobile\_price,mobile\_brend,mobile\_os,mobile\_date FROM Mobile;  end  $$ language PLPGSQL;  drop function SHOW\_MOBILE  select SHOW\_MOBILE()  --Создание функции для просмотра корзины--  CREATE OR REPLACE FUNCTION SHOW\_BASKET(client\_id\_arg int)  RETURNS TABLE(  id\_mobile\_arg int,  client\_id\_a int,  counter int  ) as $$  begin  RETURN QUERY  SELECT \* FROM Basket where client\_id = client\_id\_arg;  end  $$ language PLPGSQL;  drop function SHOW\_BASKET  select \* from SHOW\_BASKET(1)  --Создание функции для просмотра избранное--  CREATE OR REPLACE FUNCTION SHOW\_FAVOURITES(client\_id\_arg int)  RETURNS TABLE(  id\_mobile\_arg int,  client\_id\_a int  ) as $$  begin  RETURN QUERY  SELECT \* FROM Favorites where client\_id = client\_id\_arg;  end  $$ language PLPGSQL;  drop function SHOW\_FAVOURITES  select \* from SHOW\_FAVOURITES(1)  --Создание функции добавления товара в корзину--  CREATE OR REPLACE PROCEDURE ADD\_MOBILE\_TO\_BASKET(  mobile\_id\_arg INTEGER,  client\_id\_arg int  )  LANGUAGE PLPGSQL  AS $$  declare hh int;  BEGIN  hh = (select count(id\_mobile) from Basket where id\_mobile = mobile\_id\_arg);  if hh = 0 then  INSERT INTO Basket  VALUES (mobile\_id\_arg,client\_id\_arg,1);  elseif hh > 0 then  update Basket set counter = counter + 1 where id\_mobile = mobile\_id\_arg;  END if;  end;  $$;  select \* from Mobile  call ADD\_MOBILE\_TO\_BASKET(2,1)  --Создание функции обновления товара--  CREATE OR REPLACE PROCEDURE UPDATE\_MOBILE(  id\_mobile\_arg int,  mobile\_name\_arg varchar(100),  mobile\_price\_arg int,  mobile\_brend\_arg varchar(100),  mobile\_os\_arg varchar(100),  mobile\_date\_arg date  ) LANGUAGE PLPGSQL AS $$  BEGIN  UPDATE Mobile  SET  mobile\_name = mobile\_name\_arg,  mobile\_price=mobile\_price\_arg,  mobile\_brend=mobile\_brend\_arg,  mobile\_os=mobile\_os\_arg,  mobile\_date=mobile\_date\_arg  WHERE Mobile.id\_mobile = id\_mobile\_arg;  END;  $$;  call UPDATE\_MOBILE(1,'dfssd',777,'SUMSUNG','ANDROID','1923-03-22')  --Создание функции удаления товара из магазина--  CREATE OR REPLACE PROCEDURE DELETE\_MOBILE( mobile\_id\_arg INTEGER) LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Basket WHERE Basket.id\_mobile = mobile\_id\_arg;  DELETE FROM Mobile WHERE Mobile.id\_mobile = mobile\_id\_arg;  END;  $$;  call DELETE\_MOBILE(2);  select \* from Mobile;  --Удаление товара из корзины--  CREATE OR REPLACE PROCEDURE DELETE\_MOBILE\_FROM\_BASKET( mobile\_id\_arg INTEGER)  LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Basket WHERE id\_mobile = mobile\_id\_arg;  END;  $$;  --Удаление товара из избранных--  CREATE OR REPLACE PROCEDURE DELETE\_MOBILE\_FROM\_FAVOURITES(mobile\_id\_arg INTEGER,  client\_id\_arg int)  LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Favorites WHERE id\_mobile = mobile\_id\_arg  and client\_id = client\_id\_arg;  END;  $$;  --Уменьшение количесвтва товаров в корзине--  CREATE OR REPLACE PROCEDURE DELETE\_MOBILE\_COUNT\_BASKET(mobile\_id\_arg INTEGER)  LANGUAGE PLPGSQL AS $$  declare hh int;  BEGIN  hh = (select count(id\_mobile) from Basket where id\_mobile = mobile\_id\_arg);  if hh > 1 then  update Basket set counter = counter - 1 where id\_mobile = mobile\_id\_arg;  elseif then  RAISE EXCEPTION 'Ошибка';  end if;  END;  $$;  call DELETE\_MOBILE\_COUNT\_BASKET(3)  select \* from Basket  --Создание функции поиска товара по названию--  create index if not exists idx\_tsv\_mobile\_name on Mobile using gin(text\_tsv)  alter table Mobile add column text\_tsv tsvector  update Mobile set text\_tsv = to\_tsvector('russian',mobile\_name)  select \* from Mobile  drop function SEARCH\_MOBILE\_BY\_NAME  CREATE or replace FUNCTION SEARCH\_MOBILE\_BY\_NAME(mobile\_name\_arg varchar(100))  RETURNS TABLE(  mobile\_name varchar(100),  mobile\_price integer,  mobile\_brend varchar(100),  mobile\_os varchar(100),  mobile\_date date  ) as $$  begin  RETURN QUERY  select MOBILE\_INFO.mobile\_name,MOBILE\_INFO.mobile\_price,  MOBILE\_INFO.mobile\_brend, MOBILE\_INFO.mobile\_os,  MOBILE\_INFO.mobile\_date FROM MOBILE\_INFO where MOBILE\_INFO.text\_tsv @@  plainto\_tsquery('russian',mobile\_name\_arg);  end;  $$ language PLPGSQL;  drop function SEARCH\_PRODUCT\_BY\_NAME;  select \* from SEARCH\_MOBILE\_BY\_NAME('о Самсунгах');  select \* from Mobile;  --Создание функции добавления отзыва телефону--  CREATE OR REPLACE PROCEDURE public.add\_review\_to\_mobile(  IN mobile\_id\_arg integer,  IN client\_id\_arg integer,  IN description\_arg character varying)  LANGUAGE 'plpgsql'  AS $$  declare hh int;  BEGIN  hh = (select count(id\_mobile) from Reviews where client\_id = client\_id\_arg);  if hh = 0 then  INSERT INTO Reviews  VALUES (mobile\_id\_arg,client\_id\_arg,description\_arg);  elseif hh > 0 then  RAISE EXCEPTION 'Ошибка: нельзя добавить больше 1 отзыва к товару';  END if;  end;  $$;  --Создание функции добавленяи телефона в избранное--  CREATE OR REPLACE PROCEDURE public.add\_mobile\_to\_favourites(  IN mobile\_id\_arg integer,  IN client\_id\_arg integer)  LANGUAGE 'plpgsql'  AS $BODY$  declare hh int;  BEGIN  hh = (select count(id\_mobile) from Favorites where id\_mobile = mobile\_id\_arg);  if hh = 0 then  INSERT INTO Favorites  VALUES (mobile\_id\_arg,client\_id\_arg);  elseif hh > 0 then  RAISE EXCEPTION 'Ошибка: товар уже добавлен в избранное';  END if;  end;  $BODY$;  --Создание функции офрмления заказа--  CREATE OR REPLACE PROCEDURE PURCHAISE(  id\_client\_arg INTEGER,  client\_name\_arg varchar(100),  client\_phone\_arg varchar(20),  address\_order\_arg TEXT)  LANGUAGE plpgsql  AS $$  DECLARE  information\_f1 TEXT;  information\_f2 TEXT;  all\_count int;  all\_price int;  BEGIN  SELECT string\_agg(mobile\_name || '(' || counter || ')', ', ') INTO information\_f1  FROM Basket join Mobile on Basket.id\_mobile = Mobile.id\_mobile  WHERE client\_id = id\_client\_arg and counter > 1;  SELECT string\_agg(mobile\_name, ', ') INTO information\_f2 FROM Basket  join Mobile on Basket.id\_mobile = Mobile.id\_mobile  WHERE client\_id = id\_client\_arg and counter = 1;  all\_count = (select sum(counter) from Basket);  all\_price = (select sum(mobile\_price\*counter) from Basket join  Mobile on Basket.id\_mobile = Mobile.id\_mobile);  INSERT INTO Ordert (client\_id, client\_name, client\_phone, address\_order,  information\_order, all\_count, all\_price)  VALUES (id\_client\_arg,  client\_name\_arg,  client\_phone\_arg,  address\_order\_arg,  concat\_ws(', ',information\_f1, information\_f2),  all\_count,  all\_price);  DELETE FROM Basket WHERE client\_id = id\_client\_arg;  END;  $$;  select \* from Basket  call ADD\_MOBILE\_TO\_BASKET(2,1)  call PURCHAISE(1,'Павел','+343245324332','Г Минск')  select \* from Ordert  --Создание функции поиска товара по категории--  CREATE or replace FUNCTION SEARCH\_MOBILE\_BY\_BREND(mobile\_brend\_arg varchar(100))  RETURNS TABLE(  mobile\_name varchar(100),  mobile\_price integer,  mobile\_brend varchar(100),  mobile\_os varchar(100),  mobile\_date date  ) as $$  begin  RETURN QUERY  SELECT MOBILE\_INFO.mobile\_name,MOBILE\_INFO.mobile\_price,  MOBILE\_INFO.mobile\_brend, MOBILE\_INFO.mobile\_os,  MOBILE\_INFO.mobile\_date FROM MOBILE\_INFO  where MOBILE\_INFO.mobile\_brend like '%'|| mobile\_brend\_arg || '%';  end;  $$ language PLPGSQL;  drop function SEARCH\_PRODUCT\_BY\_CATEGORY;  select SEARCH\_MOBILE\_BY\_BREND('SUMSUNG');  select \* from Product;  select \* from Product\_info    --Создание функции для анализа заказов--  CREATE or replace FUNCTION ANALIZE()  RETURNS TABLE(  Все\_проданные\_товары bigint,  Общая\_выручка bigint  ) as $$  declare cont text;  begin  RETURN QUERY  select sum(all\_count), sum(all\_price) from ordert\_info;  end;  $$ language PLPGSQL;  select \* from ANALIZE()  drop function ANALIZE  --Создание функции для заполнения 100000 строк--  CREATE OR REPLACE FUNCTION INSERT\_MOBILE()  RETURNS VOID AS $$  DECLARE  I INTEGER := 1;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Mobile (mobile\_name,mobile\_price,mobile\_brend,mobile\_os,  mobile\_date) values('SUWU',I,'SUMSUNG','ANDROID','1923-03-22');  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL;  select INSERT\_MOBILE()  select \* from Mobile where mobile\_name='SUMSUNG'  ---Процедура входа на сервис---  CREATE OR REPLACE PROCEDURE ENTERING()  AS $$  DECLARE  BEGIN  INSERT INTO Client default values;  END;  $$ LANGUAGE PLPGSQL;  call ENTERING()  -------------Индексы--------  CREATE INDEX idx\_mob ON Mobile(mobile\_name);  CREATE INDEX idx\_mob\_id ON Mobile(id\_mobile);  CREATE INDEX idx\_id\_ordert ON Ordert(id\_ordert);  create index if not exists idx\_tsv\_mobile\_name on Mobile using gin(text\_tsv)  drop INDEX idx\_mob  -----------------------------------  -------------ПРЕДСТАВЛЕНИЯ---------  -----------------------------------    --Представление с информацией о товаре--  CREATE VIEW MOBILE\_INFO AS  SELECT p.mobile\_name,  p.mobile\_price,  p.mobile\_brend,  p.mobile\_os,  p.mobile\_date,  p.text\_tsv  FROM MOBILE p  drop view MOBILE\_INFO;  select \* from MOBILE\_INFO;  --Представление с информацией о заказах--  CREATE OR REPLACE VIEW public.ordert\_info  AS  SELECT o.id\_ordert,  o.client\_id,  o.client\_name,  o.client\_phone,  o.date\_order,  o.address\_order,  o.information\_order,  o.all\_count,  o.all\_price  FROM ordert o;      ------------- Триггеры -----------------  ----------------------------------------  ----------------------------------------  --Создание триггера, реагирующего на корректный ввод имени--  CREATE FUNCTION FUNC\_FOR\_custom\_name() RETURNS TRIGGER AS $$  BEGIN  IF NEW.client\_name < 2 THEN  RAISE EXCEPTION 'Имя должно содержать минимум 2 символа';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE or replace TRIGGER TR\_custom\_name  BEFORE INSERT OR UPDATE  ON Ordert  FOR EACH ROW EXECUTE FUNCTION FUNC\_FOR\_custom\_name();  --Создание триггера, реагирующий на корректность номера телефона--  CREATE FUNCTION FUNC\_FOR\_custom\_phone() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.client\_phone) < 13 or CHAR\_LENGTH(NEW.client\_phone)>13 THEN  RAISE EXCEPTION 'Телефон должен содеражть 13 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_custom\_phone  BEFORE INSERT OR UPDATE ON Ordert  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_custom\_phone();  --Создание триггера, реагирующий на корректность адреса--  CREATE FUNCTION FUNC\_FOR\_address\_order() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.address\_order) < 5 THEN  RAISE EXCEPTION 'Адрес должен быть больше 4 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_address\_order  BEFORE INSERT OR UPDATE ON Ordert  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_address\_order();    ----Технология----  select \* from Ordert, Mobile where information\_order @@  plainto\_tsquery('russian',Mobile.mobile\_name)  CREATE EXTENSION adminpack;  CREATE EXTENSION IF NOT EXISTS file\_fdw;  --------------— Функция экспорта данных в XML  CREATE OR REPLACE FUNCTION public.export\_orders\_to\_xml\_file(  file\_path text)  RETURNS void  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  xml\_data xml;  xml\_doc text;  BEGIN  SELECT table\_to\_xml('Mobile', true, false, '') INTO xml\_data;  xml\_doc := format('<?xml version="1.0" encoding="UTF-8"?>%s', xml\_data::text);  PERFORM pg\_file\_write(file\_path, xml\_doc, true);  END;  $BODY$;  ALTER FUNCTION public.export\_orders\_to\_xml\_file(text)  OWNER TO roledevelop;  SELECT export\_orders\_to\_xml\_file('D:/BDKURSACH/export.xml');  --------------— Функция импорта данных из XML  CREATE OR REPLACE FUNCTION import\_data\_from\_xml()  RETURNS VOID AS  $$  DECLARE  xml\_data XML;  BEGIN  SELECT XMLPARSE(DOCUMENT convert\_from(pg\_read\_binary\_file('D:/BDKURSACH/export.xml'), 'UTF8'))  INTO xml\_data;  END;  $$  LANGUAGE plpgsql;  SELECT import\_data\_from\_xml(); |