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 интернет-магазина электроники с использованием технологии полнотекстового поиска»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г. А. Хващевский

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](#_Toc135187219)

[1 Анализ требований к программному средству 5](#_Toc135187220)

[1.1 Аналитический обзор аналогов 5](#_Toc135187221)

[1.1.1 Аналог “5 Элемент” 5](#_Toc135187222)

[1.1.2 Аналог “7745.by” 6](#_Toc135187223)

[1.2 Разработка функциональных требований, определение вариантов использования 6](#_Toc135187224)

[В данном разделе были разделены возможности пользования услугами сервиса между различными ролями в целях избежания конфликта между пользователями в использовании услуг. 8](#_Toc135187225)

[1.3 Вывод 8](#_Toc135187226)

[2 Разработка архитектуры проекта 9](#_Toc135187227)

[2.1 Обобщенная структура управлением приложения 9](#_Toc135187228)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 9](#_Toc135187229)

[3 Разработка модели базы данных 12](#_Toc135187230)

[3.1 Создание необходимых объектов 12](#_Toc135187231)

[3.1.1 Представления базы данных 12](#_Toc135187232)

[3.1.2 Индексы базы данных 13](#_Toc135187233)

[3.1.3 Триггеры базы данных 14](#_Toc135187234)

[3.2 Описание используемой технологии 15](#_Toc135187235)

[3.3 Вывод 16](#_Toc135187236)

[4 Установка, настройка и использование PosgtreSQL 15 18](#_Toc135187237)

[4.1 Установка PostgreSQL 18](#_Toc135187238)

[4.2 Создание таблиц 18](#_Toc135187239)

[4.3 Создание ролей для разграничения 19](#_Toc135187240)

[4.4 Создание пакетов процедур для базы данных 20](#_Toc135187241)

[4.5 Описание процедур экспорта и импорта 24](#_Toc135187242)

[4.6 Вывод 26](#_Toc135187243)

[5 Тестирование 27](#_Toc135187244)

[5.1 Тестирование производительности базы данных 27](#_Toc135187245)

[5.2 Вывод 28](#_Toc135187246)

[6 Руководство по использованию программного средства 29](#_Toc135187247)

[6.1 Руководство пользователя 29](#_Toc135187248)

[6.2 Установка приложения 31](#_Toc135187249)

[6.3 Вывод 31](#_Toc135187250)

[Заключение 32](#_Toc135187251)

[Список литературных источников 33](#_Toc135187252)

[Приложение А 34](#_Toc135187253)

# Введение

Цель курсовой работы заключается в создании реляционной базы данных для интренет-магазина. База данных нужна для обеспечения пользователей взаимодействием с интерфейсом интернет-сервиса.

В рамках курсового проекта база данных будет использована для хранения информации о пользователях и товарах, организации данных, а также хранения функций и процедур, с помощью которых пользователь сможет взаимодействовать с интернет-сервисом. Для курсового проекта было принято решения воспользоваться СУБД Postgres SQL, так как эта СУБД наиболее популярна среди пользователей, а так же обладает высокой производительностью, приятным интерфейсом и надежностью.

Для гарантированной безопасности пользователей приложения в моей курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных.

Основные требования к базе данных:

* Реализация ролей администратора, менеджера и пользователя;
* поиск товаров по названию;
* поиск пользователей по имени;
* реализация логики работы с корзиной;
* использование функций и процедур для работы с базой данных.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Самое главное в проекте – тематика, ведь отталкиваясь от тематики, разрабатывается проект. Чтобы проект получился удобным в использовании и смог конкурировать, нужно проанализировать аналоги, ведь именно с аналогами и будет конкурировать проект в будущем.

Тематика проекта – интернет-магазин электроники. В настоящее время интернет-магазин превосходит обычные магазины практически во всех аспектах. К примеру, людям не надо выходить из дома и ехать в магазин, чтобы выбрать устройства, когда можно все это сделать из любого устройства с выходом в интернет. В интернет-магазинах можно сравнивать товары, основываясь на отзывах, ценах, характеристиках, что мы не сможем сделать у полок магазинов так же быстро и просто.

### 1.1.1 Аналог “5 Элемент”

5 Элемент [1] – один из самых популярных интернет-сервисов по продажам электроники. В сервисе можно искать товары по названию, сортировать по ценам, категориям, фирмам и т.д. Пример интерфейса данного сервиса представлен на рисунке 1.1.

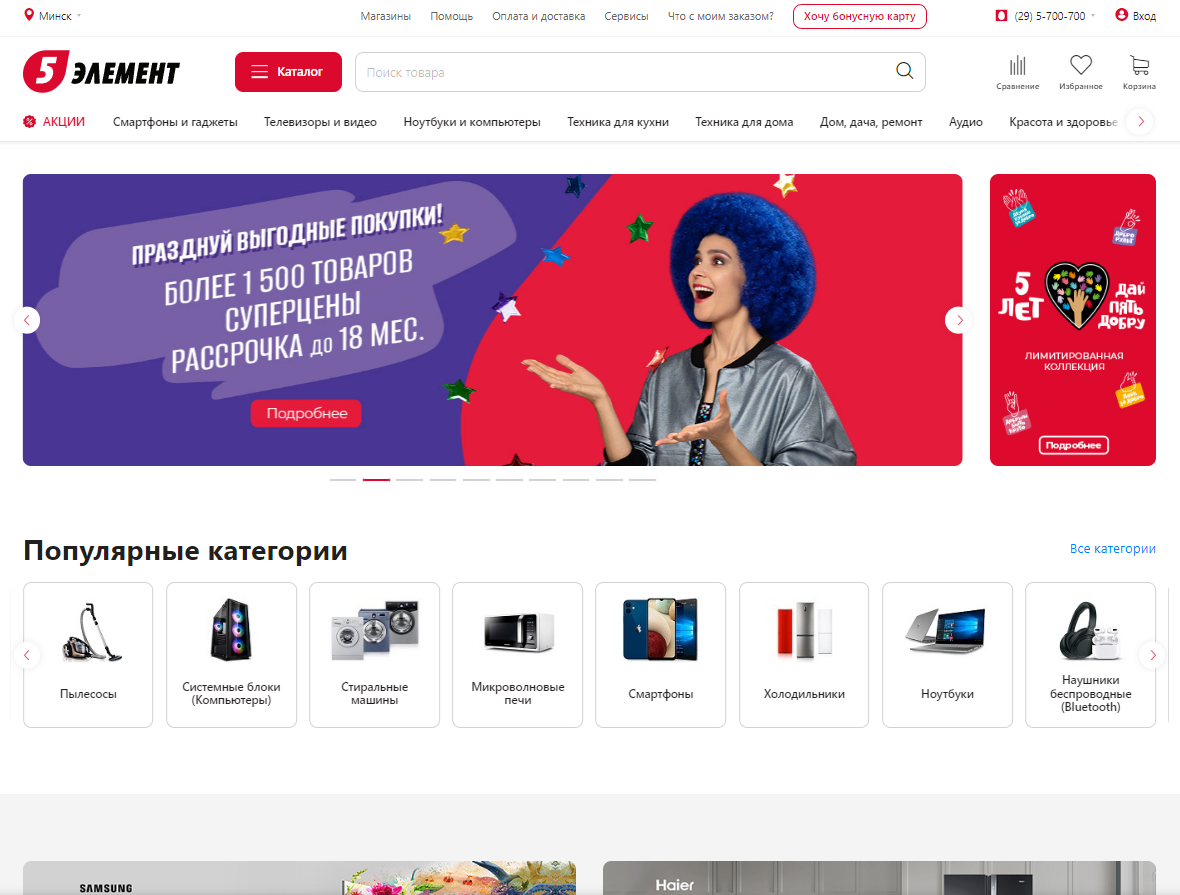


Рисунок 1.1 – Интерфейс сервиса “5 Элемент”

Главные возможности сервиса являются: поиск по названию, поиск по категории, сортировка по цене, сортировка по рейтингу, сортировка по дате добавления. Так же в сервисе возможно добавлять товары в корзину, изменять их количество и заказывать на дом, указывая данные для доставки.

Пользователи могут оставлять отзывы товарам, чтобы будущие покупатели, не уверенные в выборе товаров, могли узнать опыт пользования товара от других пользователей.

### 1.1.2 Аналог “7745.by”

7745.by [2] – еще один популярный интернет-сервис, которая позволяет пользователям заказывать товары через интернет с различных устройств с доступом в интернет.

Пользователи могут добавлять товар в корзину, отслеживать статус заказа, переписываться в онлайн со службой поддержки, что очень полезно при непонимании в некоторых местах сервиса, где пользователь бы мог потеряться или если пользователю нужна консультация по различным услугам сервиса. Пример интерфейса данного сервиса представлен на рисунке 1.2.

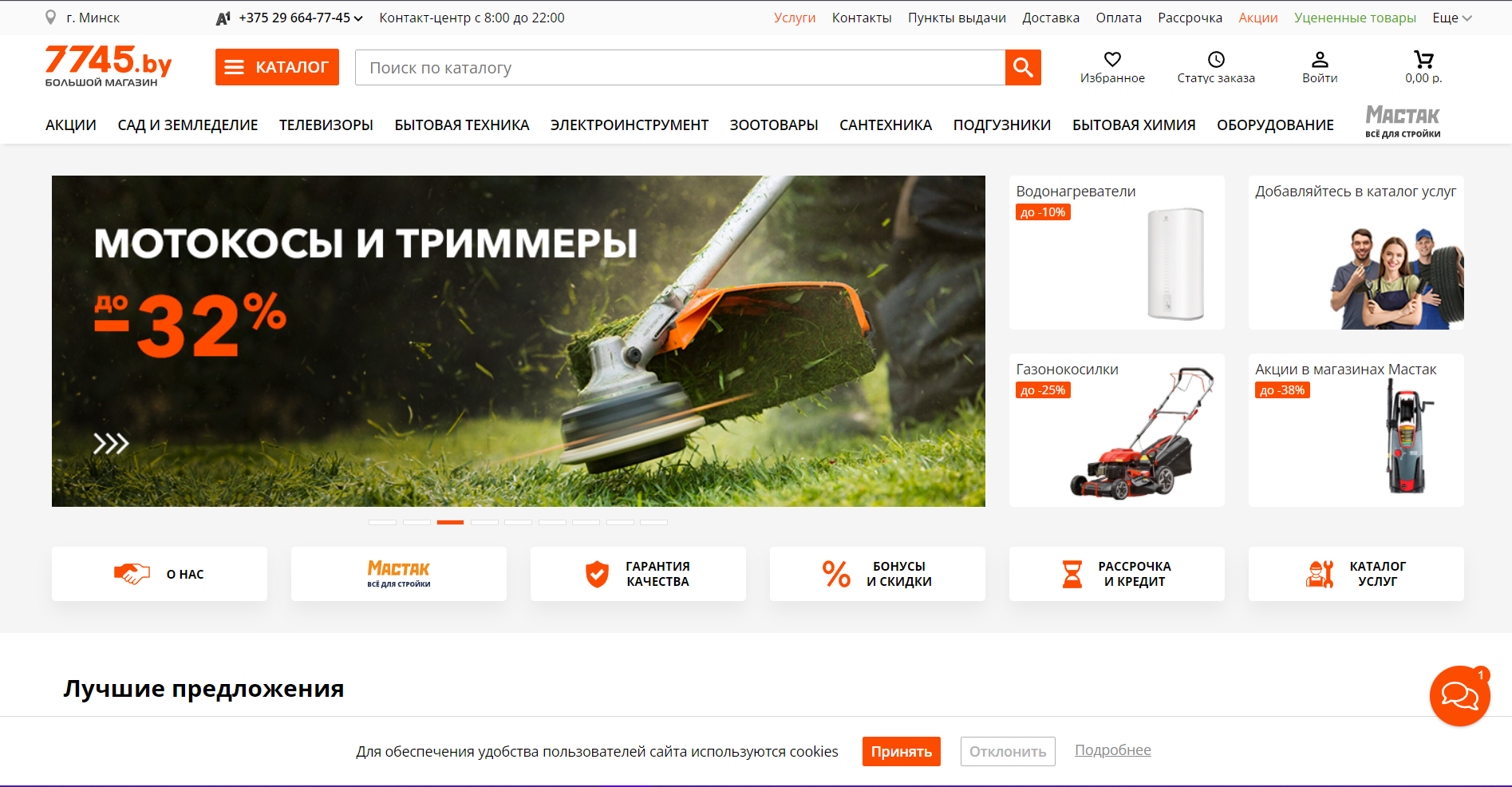


Рисунок 1.2 – Интерфейс сервиса “7745.by”

В данном разделе были проведены аналитические обзоры двух популярных интернет-сервисов по продажам электроники – “5 элемент” и “7745.by”. На основе обзора аналогов будут созданы свои пользовательские функции и услуги.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования – это необходимая к выполнению часть проекта, которая описывает то, как будет обрабатывать запросы и хранить данные база данных, как будет происходить поиск и выборка данных, а так же их защита. К примеру, в интернет-магазине должен быть реализован поиск по названию товара, поиск пользователей по имени, сортировка товара по цене, оценкам, а также выборка по категориям.

Варианты использования описывают возможные сценарии использования сервиса. На основе сценариев добавляют роли, которые как раз таки и отвечают за различные сценарии использования. После составления ролей будут создаваться функции и процедуры для каждой роли соответственно. Чтобы было проще придумывать роли и функции под них, будет создана UML-диаграмма, опираясь на которую, будет разрабатываться логика сервиса.

Роль в базе данных – набор прав, которые предоставляются получившему роль пользователю. Благодаря роли пользователь может получать доступ к различным функциям в системе. Так же роли разграничивают доступы к функциям различным пользователям, чтобы в дальнейшем не возникало конфликтов использования. В проекте было решено создать четыре роли:

* Guest;
* User;
* Manager;
* Administrator

Далее на основе ролей была разработана UML-диаграмма, описывающая возможности каждой роли. Варианты использование изображена на рисунке 1.3.



Рисунок 1.3 – UML диаграмма вариантов использования

При заходе в приложение, пользователь получит роль Guest, возможности которой просмотр каталога, регистрация, вход. После входа или регистрации Guest пользователь получит роль User.

Роль User получает возможность работы с корзиной, оценивания товара и заказа товара. В корзине User может добавлять и удалять товар. При оценивании товара User может оставлять оценку и отзыв товару.

Роль Manager заключается в управлении каталогом товаров. Под управлением подразумевается изменять характеристики товара, удалять или добавлять товар, добавлять или удалять категории.

Роль Administrator разработана для управления ролями, товарами и пользователями. Administrator имеет право на изменение структуры базы данных, логики использования функций и процедур.

## В данном разделе были разделены возможности пользования услугами сервиса между различными ролями в целях избежания конфликта между пользователями в использовании услуг.

## 1.3 Вывод

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

В ходе раздела был проведен анализ площадок, которые будут составлять конкуренцию, чтобы

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 прослушивания музыки может входить возможность быстрого и удобного поиска музыкальных композиций, исполнителей и альбомов по различным параметрам, таким как жанр, год выпуска, рейтинг и т.д. Также пользователи должны иметь возможность проигрывать музыкальные композиции на сайте без необходимости скачивания или установки специального программного обеспечения. Для удобства пользователей также должна быть предусмотрена возможность создания собственных плейлистов с возможностью настройки порядка воспроизведения.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных — это графическое представление таблиц базы данных с визуальным отображением связей между таблицами по внешним ключам. Диаграмма базы данных будет представлена на рисунке 2.1.

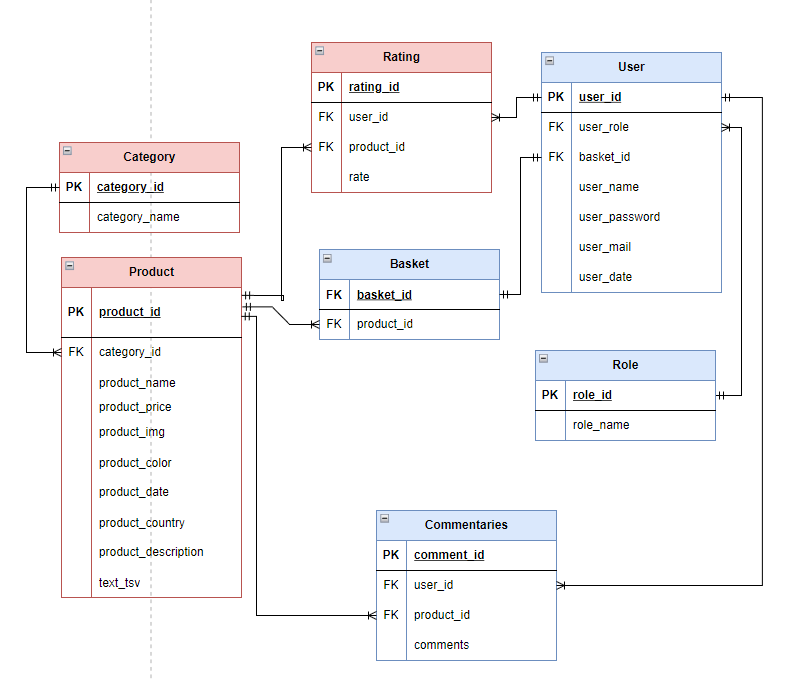


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному". Например, таблица Users связана с таблицами Rating, Basket и Role через внешние ключи user\_id, basket\_id, user\_role соответственно. Таблица Product связана с таблицами Category, Basket, Rating через внешниий ключ category\_id, и первичный ключ product\_id.

**2.3 Описание информационных объектов**

Для реализации базы данных было разработано 7 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Role, Users, Category, Basket, Rating, Product, Commentaries.

Таблица Role содержит ID роли и соответствующее имя роли. Содержит поля: role\_id (идентификатор роли, типа SERIAL, первичный ключ), role\_name (текстовое название роли, тип VARCHAR(100)).

Таблица Users содержит информацию о пользователях. Содержит поля: user\_id (идентификатор пользователя, типа SERIAL, первичный ключ), user\_role (идентификатор роли пользователя, тип INTEGER, внешний ключ ссылающийся на role\_id в таблице Role), user\_name (имя пользователя, тип VARCHAR(100)), user\_mail (адрес электронной почты пользователя, тип VARCHAR(100)), user\_password (пароль пользователя, тип VARCHAR(30)), user\_date (дата рождения пользователя, тип DATE.

Таблица Category содержит информацию о категории товара. Содержит поля: category\_id (идентификатор категории, типа SERIAL, первичный ключ), category \_name (название категории, тип VARCHAR(100)).

Таблица Commentaries содержит отзывы к товарам. Содержит поля: comment\_id (идентификатор категории, типа SERIAL, первичный ключ), user\_id (идентификатор пользователя, тип integer, внешний ключ, ссылающийся на user\_id в таблице Users), product\_id (идентификатор товара, тип integer, внешний ключ, ссылающийся на product \_id в таблице Products), comments(текст комментария, тип varchar(300)).

Таблица Product содержит информацию о товарах. Имеющиеся поля: product\_id (идентификатор товара, типа SERIAL, первичный ключ), product \_name (название товара, тип VARCHAR(100)), product \_date (дата размещения товара, тип DATE, product \_image (изображение товара, тип VARCHAR(100)), product\_country (страна изготовитель, тип VARCHAR(100)), product\_color (цвет товара, тип VARCHAR(100)), category\_id(идентификатор категории товара, тип SERIAL, внешний ключ ссылающийся на category\_id, в таблице Category), product\_price(цена товара, тип INTEGER).

Таблица Rating содержит информацию о рейтинге пользователей для треков. Имеющиеся поля: rating\_id (идентификатор рейтинга, типа SERIAL, первичный ключ), user\_id (идентификатор пользователя, который оценил трек, тип INTEGER, внешний ключ ссылающийся на user\_id в таблице Users), track\_id (идентификатор трека, который был оценен, тип INTEGER, внешний ключ ссылающийся на track\_id в таблице Product), rate (оценка, которую поставил пользователь, тип INTEGER).

Таблица Backet содержит информацию о корзине пользователя. Имеющиеся поля: backet\_id(идентификатор корзины, тип SERIAL, , внешний ключ ссылающийся на basket\_id в таблице Users), product\_id(идентификатор товара, тип SERIAL, внешний ключ ссылающийся на product\_id в таблице Product).

**2.4 Вывод**

Архитектура проекта показывает структуру приложения, а так же функциональность, которую пользователи смогут использовать в дальнейшем. Сначала была разобрана обобщенная структура управления приложением, благодаря которой были определены компоненты, которые необходимы для правильной работы проекта.

Диаграмма базы данных была разработана для построения логики связывания данных между таблицами. В ходе построение диаграммы было описано, где и как будут храниться данные пользователей и товаров.

В ходе раздела были выяснены детали хранения данных и то, как данные будут взаимодействовать между собой в проекте.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

В интернет-магазине будет несколько таблиц, хранящие данные о товарах и пользователях, а именно:

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

Users - таблица, содержащая данные о пользователе. В этой таблице будут храниться информация о пользователе, такая как имя, электронная почта, дата рождения, идентификатор корзины, идентификатор роли.

Category - таблица, содержащая категории товаров. В этой таблице будут храниться идентификатор категории и имя.

Product - таблица, содержащая информацию о товарах. В этой таблице будут храниться данные о имени товара, цене, стране-производителе, цвете, фото.

Basket - таблица, являющаяся хранилищем товаров, которые хочет заказать пользователь, а именно идентификатор товара и идентификатор пользователя.

Commentaries - таблица, являющаяся хранилищем комментариев к определенным товарам.

Rating - таблица, содержащая информацию об оценках товара пользователей.

Далее для того, чтобы пользователи могли пользоваться сервисом, не вникая в логику работы, будут созданы процедуры и функции, вызов которых будет приводить к выводу нужной информации или ее изменению.

Так же, чтобы созданные объекты работали быстрее, будут созданы индексы на столбцах, к которым пользователи чаще всего обращаются. К примеру, наиболее частый запрос к базе данных в рамках интернет-магазина — это поиск товара по названию. Соответственно нужно создать индекс на столбец product\_name таблицы Product.

### 3.1.1 Представления базы данных

Представление — это виртуальная таблица, содержимое которой определяется запросом. Как и таблица, представление состоит из ряда именованных столбцов и строк данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно. Данные таблиц, к которым обращаются – не меняются.

В данном проекте были созданы три представления:

* User\_info, которое объединяет данные таблиц Users и Role;
* Product\_info, которое содержит информацию о треках и среднюю оценку каждого трека;
* Best\_products, которое содержит информацию о 10-ти лучших товарах.

Представление User\_info выводит всю информацию о пользователях, связывая таблицы Users и Role, с помощью объединения join. Представление представлено на листинге 3.1.

|  |
| --- |
| CREATE VIEW USER\_INFO AS  SELECT u.user\_name,  u.user\_mail,  u.user\_date,  u.user\_password,  r.role\_name  FROM users u  JOIN roles r ON u.user\_role = r.role\_id; |

Листинг 3.1 – Представление User\_info

Представление Product\_info выводит всю информацию о товарах, включая их рейтинг. Для этого были объединены таблицы Product и Rating.

Представление Best\_products выводит информацию о 10-ти лучших товара по оценкам пользователей, объединяя таблицы Product и Rating.

### 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 товары по их названию, был создан индекс INDEX\_PRODUCT\_NAME на таблице Product по полю product\_name , представленый на листинге 3.2.

|  |
| --- |
| create index INDEX\_PRODUCT\_NAME on Product(product\_name); |

Листинг 3.2 – Индекс базы данных

Так же были созданы и другие индексы, которые описаны далее.

Индекс INDEX\_USER\_NAME на таблице USERS был создан для ускорения поиска пользователя по его Имени.

Индекс INDEX\_CATEGORY\_ID на таблице CATEGORY может использоваться для быстрого поиска категории товара по их ID.

Индексы нужны для более быстрого поиска большого объема данных, ведь благодаря им базе данных не нужно будет проходиться по всей таблице, когда есть индекс определенного столбца, по которому база данных и будет проводить поиск. Но и слишком большое количество индексов замедлит работу базы данных, ведь тогда понадобится для изменения или добавления данных в таблицу, если в ней установлен индекс. Поэтому при создании индексов нужно учитывать, как и пользу от его создания, так и проблемы при его эксплуатации.

### 3.1.3 Триггеры базы данных

Триггер базы данных — это объект базы данных, который срабатывает автоматически при определенных событиях. Пользователь не может напрямую вызвать триггер. Обычно триггер срабатывает при добавлении, изменении, удалении данных из таблицы. Триггеры очень полезны в случаях, когда нужно что-либо контролировать или обезопасить.

Триггер TR\_DATE срабатывает, когда при добавлении нового пользователя дата рождения оказывается несуществующей датой. При ошибке триггер генерирует исключение “Такой даты не существует”. Скрипт триггера представлен на листинге 3.3.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_TR\_DATE() RETURNS TRIGGER AS $$  BEGIN  IF NEW.user\_date >= CURRENT\_DATE THEN  RAISE EXCEPTION 'Такой даты не существует';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE or replace TRIGGER TR\_DATE  BEFORE INSERT OR UPDATE  ON Users  FOR EACH ROW EXECUTE FUNCTION FUNC\_FOR\_TR\_DATE(); |

Листинг 3.3 – Скрипт триггера TR\_DATE

Второй триггер проверяет, что длина пароля нового пользователя не менее 6 символов. Если длина пароля меньше 6 символов, то триггер генерирует исключение с сообщением об ошибке "Password must be at least 6 characters long". Триггер создается с помощью функции check\_password\_length(), которая также возвращает значение NEW при выполнении вставки или обновления.

Триггер TR\_PASSWORD срабатывает, когда при добавлении нового пользователя длина пароля оказывается меньше 8 символов. При ошибке триггер генерирует исключение “Пароль должаен содержать не менее 8 символов”. Скрипт триггера представлен на листинге 3.4.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_TR\_PASSWORD() RETURNS TRIGGER AS $$  BEGIN  IF CHAR\_LENGTH(NEW.user\_password) < 8 THEN  RAISE EXCEPTION 'Пароль должаен содержат не менее 8 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_PASSWORD  BEFORE INSERT OR UPDATE ON Users  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_TR\_PASSWORD(); |

Листинг 3.4 – Скрип триггера check\_password\_length

Триггер TR\_RATE срабатывает, когда пользователь ставит оценку товара выше или ниже возможной. При срабатывании триггер выводит исключение “Оцените в пределах от 0 до 10”. Скрипт триггера представлен на листинге 3.5.

|  |
| --- |
| CREATE FUNCTION FUNC\_FOR\_TR\_RATE() RETURNS TRIGGER AS $$  BEGIN  IF NEW.rate > 10 or NEW.rate < 0 THEN  RAISE EXCEPTION 'Оцените в пределах от 0 до 10';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE or replace TRIGGER TR\_RATE  BEFORE INSERT OR UPDATE  ON Rating  FOR EACH ROW EXECUTE FUNCTION FUNC\_FOR\_TR\_RATE(); |

Листинг 3.5 – Скрип триггера TR\_RATE

С помощью триггеров была организована проверка валидации некоторых введенных данных, что помогает пользователю быстрее исправлять ошибки, которые он допустил.

## 3.2 Описание используемой технологии

В базе данных была выбрана технология полнотекстового поиска. В Postgre полнотекстовый поиск состоит из трех компонентов: анализатор, словарь, конфигуратор. Анализатор разделяет документ за фрагменты и классифицирует их, к примеру название устройства “ Samsung Galaxy A20” будет фрагментирован так: “'a20':3 'galaxi':2 'samsung':1 'смартфон':5 'эт':4”. Причем в Postgre конфигуратор работает по заранее созданной логике и администратору не требуется придумывать логику фрагментирования. Словари преобразуют фрагменты в лексемы, приводит их к нормализованному виду и удаляют стоп-слова, которые содержатся во всех документах и по ним делать поиск бессмысленно. В Postgres есть стандартные словари для многих языков. Конфигуратор задает связку анализаторов набором словарей, которые будут обрабатывать выделенные фрагменты, для каждого фрагмента конфигуратор будет создавать набор словарей. Фрагмент последовательно проходит через все словари, и если найдено стоп-слово или фрагмент не найден, то фрагмент не будет учитываться при индексации поиска.

В нашем случае было решено применить полнотекстовый поиск к названию товара, так как зачастую название товара состоит из нескольких слов, и пользователь может не помнить полностью название и будет искать товар по словам, которые помнит.

Для реализации технологии был создан столбец text\_tsv в таблице Product, который будет хранить в себе данные названий товаров, приведенные к типу tsvector, по которому и будет вестись поиск соответствующих товаров.

Функция, реализующая полнотекстовый поиск представлена на листинге 3.5.

|  |
| --- |
| CREATE or replace FUNCTION SEARCH\_PRODUCT\_BY\_NAME(product\_name\_arg varchar(100))  RETURNS TABLE(  product\_name varchar(100),  product\_price integer,  product\_color varchar(100),  product\_country varchar(100),  product\_date date  ) as $$  begin  RETURN QUERY  SELECT Product.product\_name,Product.product\_price,  Product.product\_color, Product.product\_country,  Product.product\_date FROM Product  where  Product.product\_name like '%'|| product\_name\_arg || '%' or  Product.text\_tsv @@ plainto\_tsquery('russian', product\_name\_arg);  end;  $$ language PLPGSQL; |

Листинг 3.6 – Cкрипт полнотекстового поиска

С помощью столбца text\_tsv можно найти товары даже если неправильно написано название. Это значительно упрощает поиск товаров.

## 3.3 Вывод

В разделе были созданы объекты, облегчающие работу с базой данных. Были созданы триггеры, которые отслеживают правильность введенных данных. Была создана технология полнотекстового поиска, которая дает возможность быстро и точно находить товар. С помощью индексов была ускорена работа базы данных при поиске определенных данных. Представления дали возможность быстро обращаться к определенным данным без написания больших запросов. Была проведена оптимизация и ускорение работы базы данных.

4 Установка, настройка и использование PosgtreSQL 15

## 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 HiTech, которая будет использоваться в дальнейшем для хранения данных музыкального приложения.

## 4.2 Создание таблиц

Создание таблиц – самая важная часть при конструировании базы данных. Здесь устанавливаются связи и логика хранения всех данных сервиса. Для того, чтобы таблицы хранились в файловой системе и была быстрая возможность к ним обратиться, были созданы табличные пространства, соединяющие таблицы в файловой системе. Скрипт для создания табличных пространств представлен на листинге 4.1.

|  |
| --- |
| create tablespace TS\_USER  location 'D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\Tablespaces\TS\_USER';  create tablespace TS\_PRODUCT  location 'D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\Tablespaces\TS\_PRODUCT'; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим шесть основных таблиц: Role, Users, Category, Rating,Product, Basket.

Таблицы Role, Users и Basket будут храниться в табличном пространстве TS\_USER, таблицы Product, Category и Rating - в табличном пространстве TS\_PRODUCT.

В базе данных будут созданы связи между таблицами к доступу информации с нескольких таблиц сразу и дальнейшего объединения в результирующий ответ. К примеру связь FK\_USERS\_BASKET связывает таблицы Users и Basket по столбцу basket\_id. Пример создания таблицы с связью представлен в листинге 4.2

|  |
| --- |
| CREATE TABLE USERS (  USER\_ID SERIAL PRIMARY KEY,  USER\_NAME VARCHAR(255) NOT NULL,  USER\_IMG BYTEA,  USER\_EMAIL VARCHAR(255) UNIQUE NOT NULL,  USER\_PASSWORD VARCHAR(255) NOT NULL,  USER\_DATE\_OF\_BIRTH DATE,  USER\_ROLE\_ID INTEGER,  CONSTRAINT FK\_USER\_ROLE FOREIGN KEY (USER\_ROLE\_ID) REFERENCES ROLE(ROLE\_ID)  ) TABLESPACE TS\_USER; |

Листинг 4.2 – Cкрипт создание таблицы Users

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, TS\_PRODUCT, в каждом из которых будут храниться таблицы.

## Создание ролей для разграничения

В разделе будут описаны создание и использование ролей в базе данных для разграничения возможностей использования процедур и функций. Разграничение создается для безопасности использования базы данных, чтобы посторонние лица не могли изменить структуру базы данных.

Роль role\_root является главной в базе данных. Эта роль имеет доступ ко всем функциям и процедурам, которые созданы в базе данных. Выданные привилегии роли programmer\_role можно увидеть на листинге 4.3.

|  |
| --- |
| grant connect on database "HiTech" to role\_root;  grant all on schema public TO role\_root;  grant all privileges on database "HiTech" to role\_root;  grant all privileges on tablespace TS\_USER to role\_root;  grant all privileges on tablespace TS\_PRODUCT to role\_root; |

Листинг 4.3 – Привилегии, выданные роли role\_root

Роль role\_manager создана для управления каталогом проекта. Ей выданы права на редактирование, добавление, удаление категорий и товаров в базе данных. Выданные привилегии роли manager\_role можно увидеть на листинге 4.4.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION add\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_product TO role\_manager;  GRANT EXECUTE ON FUNCTION registration TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_product TO role\_manager;  GRANT EXECUTE ON FUNCTION add\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_user TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_password TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_user TO role\_manager;  GRANT SELECT, INSERT, UPDATE ON TABLE Users TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Product TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Basket TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Category TO role\_manager;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_id\_seq TO role\_manager;  GRANT USAGE, SELECT ON SEQUENCE product\_product\_id\_seq TO role\_manager; |

Листинг 4.4 – Привилегии, выданные роли manager\_role

Роль User это роль зарегистрированного пользователя. Из возможностей роли это просмотр каталога, работа с корзиной, поиск определенной информации. Выданные привилегии роли user\_role можно увидеть на листинге 4.5.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION registration TO role\_user;  GRANT EXECUTE ON FUNCTION signin TO role\_user;  GRANT EXECUTE ON PROCEDURE add\_rate TO role\_user;  GRANT EXECUTE ON PROCEDURE add\_product\_from\_basket TO role\_user;  GRANT EXECUTE ON PROCEDURE udpate\_user TO role\_user;  GRANT EXECUTE ON PROCEDURE update\_password TO role\_user;  GRANT EXECUTE ON PROCEDURE delete\_from\_basket TO role\_user;  GRANT EXECUTE ON FUNCTION search\_product\_by\_name TO role\_user;  GRANT EXECUTE ON FUNCTION search\_product\_by\_category TO role\_user;  GRANT EXECUTE ON FUNCTION search\_user\_by\_name TO role\_user;  GRANT EXECUTE ON FUNCTION CHECK\_BASKET\_FUNC TO role\_user;  GRANT SELECT, INSERT, UPDATE ON TABLE Users TO role\_user;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Rating TO role\_user;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Basket TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE basket\_basket\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE basket\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_rating\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_user\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_role\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_basket\_seq TO role\_user; |

Листинг 4.5 – Привилегии, выданные роли user\_role

Таким образом, были созданы три роли для ограничения доступа к базе данных: role\_user, role\_root и role\_manager. Каждой роли были выданы свои права на определенные функции и процедуры в базе данных.

## 4.4 Создание пакетов процедур для базы данных

Процедуры и функции в базе данных используются для того, чтобы можно было быстро обратиться к данным без написания каждым разом запроса, ведь можно быстро обратиться по имени к функции или процедуре и передать ей соответствующие аргументы поиска, редактирования или удаления данных.

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.

Выбор использования процедуры или функции зависит от контекста использования. К примеру процедуры используются чаще для изменения данных в таблицах, а функции для вычисления данных с возвратом итога.

Для поиска определенного пользователя по имени была создана функция SEARCH\_PRODUCT\_BY\_NAME. Данная функция принимает на вход поисковый запрос в виде имени пользователя и возвращает всю информацию о нем. Функция представлена на листинге 4.6.

|  |
| --- |
| CREATE or replace FUNCTION SEARCH\_USER\_BY\_NAME(user\_name\_arg varchar(100))  RETURNS TABLE(  user\_name varchar(100),  user\_mail varchar(100),  user\_date date  ) as $$  declare  begin  RETURN QUERY  SELECT USER\_INFO.user\_name,USER\_INFO.user\_mail,  USER\_INFO.user\_date FROM USER\_INFO  where USER\_INFO.user\_name ilike '%'|| user\_name\_arg || '%';  end;  $$ language PLPGSQL; |

Листинг 4.6 – Функция SEARCH\_PRODUCT\_BY\_NAME

Для поиска используется оператор ILIKE, который позволяет выполнить поиск без регистра. Функция в качестве аргумента принимает имя пользователя и возвращает таблицу с данными о пользователе.

Для заполнения таблицы CATEGORY была разработана функция INSERT\_GENRES, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.7.

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_CATEGORY()  RETURNS VOID AS $$  DECLARE  I INTEGER := 5;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Category (category\_name,category\_id) VALUES ('CATEGORY ' || I,I);  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.7 – Функция заполнения таблицы Category

Функция INSERT\_CATEGORY была создана для заполнения таблицы CATEGORY 100000 строками. Для выполнения функции нужно выполнить запрос SELECT INSERT\_CATEGORY().

Для добавления данных в таблицы были созданы следующие функции:

ADD\_PRODUCT – добавить товар;

REGISTRATION – регистрация пользователя;

ADD\_PRODUCT\_TO\_BASKET – добавить товар в корзину;

ADD\_RATING – добавить оценку товару.

Для редактирование данных в таблице были созданы следующие процедуры и функции:

UPDATE\_USER– обновить данные пользователя;

UPDATE\_PASSWORD – обновить пароль;

UPDATE\_PRODUCT– обновить данные товара;

ADD\_RATING – добавить оценку товару.

Для удаления данных из таблицы были созданы следующие процедуры и функции:

DELETE\_USER– удалить пользователя;

DELETE\_FROM\_BASKET– удалить товар из корзины;

DELETE\_PRODUCT– удалить товар из магазина;

ADD\_RATING – добавить оценку товару.

Для вывода данных из таблицы были созданы следующие процедуры и функции:

AVG\_RATE– вывести рейтинг товара;

SEARCH\_USER\_BY\_NAME– поиск пользователя по его имени;

SEARCH\_PRODUCT\_BY\_NAME– поиск товара по названию;

SEARCH\_PRODUCT\_BY\_CATEGORY– вывести товары определенной категории;

CHECK\_BASKET\_FUNC – вывести товары корзины пользователя.

На листинге 4.8 будет функция ADD\_RPODUCT, которая добавляет товар в каталог.

|  |
| --- |
| CREATE OR REPLACE FUNCTION ADD\_PRODUCT(category\_id\_arg INTEGER,product\_name\_arg VARCHAR(100),product\_price\_arg INTEGER,  product\_img\_arg VARCHAR(100), product\_color\_arg VARCHAR(100),product\_country\_arg VARCHAR(100),product\_description\_arg varchar(200))  RETURNS INTEGER  AS $$  DECLARE  product\_id\_add INTEGER;  product\_date\_arg date;  BEGIN  product\_date\_arg = current\_date;  INSERT INTO Product(category\_id,product\_name,product\_price,product\_img ,product\_color, product\_date, product\_country,product\_description)  VALUES(category\_id\_arg,product\_name\_arg,product\_price\_arg,product\_img\_arg,product\_color\_arg,product\_date\_arg,product\_country\_arg,product\_description\_arg)  RETURNING Product.product\_id into product\_id\_add;  RETURN product\_id\_add;  END;  $$ LANGUAGE PLPGSQL;  select ADD\_PRODUCT(2,'Samsung Galaxy A20',2000,'A20.img','Черный','Южная Корея','Это смартфон');  drop function ADD\_PRODUCT;  select \* from Product; |

Листинг 4.8 – Функция заполнения таблицы Product

На листинге 4.9 будет представлен скрипт редактирования товара.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_PRODUCT(  product\_id\_arg INTEGER,  category\_id\_arg integer,  product\_name\_arg varchar(100),  product\_price\_arg INTEGER,  product\_img\_arg varchar(100),  product\_color\_arg varchar(100),  product\_date\_arg date,  product\_country\_arg varchar(100),  product\_descryption\_arg varchar(200)  ) LANGUAGE PLPGSQL AS $$  BEGIN  IF product\_id\_arg IS null THEN  UPDATE Product  SET  category\_id = category\_id\_arg,  product\_name=product\_name\_arg,  product\_price=product\_price\_arg,  product\_img=product\_img\_arg,  product\_color=product\_color\_arg,  product\_date=product\_date\_arg,  product\_country=product\_country\_arg,  product\_descryption=product\_descryption\_arg  WHERE Product.product\_id = product\_id\_arg;  END IF;  END;  $$; |

Листинг 4.9 – Функция редактирования таблицы Product

А так же удаление данных из таблицы Product будет представлено в листинге 4.10.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DELETE\_PRODUCT( product\_id\_arg INTEGER) LANGUAGE PLPGSQL AS $$  BEGIN  DELETE FROM Product WHERE Product.product\_id = product\_id\_arg;  DELETE FROM Basket WHERE Basket.product\_id = product\_id\_arg;  END;  $$;  call DELETE\_PRODUCT(2); |

Листинг 4.10 – Функция удаления данных из таблицы Product

После удаления товара из таблицы Product, так же удаляется товар из таблицы Basket, чтобы не получилось ситуации, в которой у пользователя в корзине был бы несуществующий товар.

## 4.5 Описание процедур экспорта и импорта

База данных имеет возможность экспортировать и импортировать данные для таблицы Users в формате XML. Данные функции полезны для сохранения данных в отдельных местах, если нужна будет возможность восстановления.

Для экспорта данных в формате XML была создана функция EXPORT\_USERS, которая принимает имя файла. Функция представлена на листинге 4.11.

|  |
| --- |
| CREATE EXTENSION adminpack;  CREATE EXTENSION IF NOT EXISTS file\_fdw;  CREATE OR REPLACE FUNCTION export\_users(  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('Users', 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;  $BODY$; |

Листинг 4.11 – Функция EXPORT\_USERS

Для импорта данных из файла в формате XML была создана процедура IMPORT\_USERS которая принимает имя файла и импортирует данные из файла в таблицу Users. Функция представлена на листинге 4.18.

|  |
| --- |
| CREATE OR REPLACE FUNCTION import\_data\_from\_xml(filename TEXT) RETURNS VOID AS $$  DECLARE  xml\_data TEXT;  BEGIN  -- Читаем XML данные из файла  xml\_data := pg\_read\_file(filename, 0, 1000);  -- Парсим XML данные и выполняем импорт  WITH xml\_rows AS (  SELECT unnest(xpath('/root/row', xmlparse(content xml\_data))) AS xml\_row  )  INSERT INTO Users (user\_id, user\_name,user\_password,user\_date,basket\_id,user\_role,user\_mail)  SELECT  (xpath('/row/user\_id/text()', xml\_row))[1]::text::integer AS user\_id,  (xpath('/row/user\_name/text()', xml\_row))[1]::TEXT AS user\_name,  (xpath('/row/user\_password/text()', xml\_row))[1]::TEXT AS user\_password,  (xpath('/row/user\_date/text()', xml\_row))[1]::text::date AS user\_date,  (xpath('/row/basket\_id/text()', xml\_row))[1]::text::int AS basket\_id,  (xpath('/row/user\_role/text()', xml\_row))[1]::text::int AS user\_role,  (xpath('/row/user\_mail/text()', xml\_row))[1]::TEXT AS user\_mail  FROM  xml\_rows;  -- Выводим сообщение об успешном импорте  RAISE NOTICE 'Данные импортированы из файла %', filename;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.12 – Функция IMPORT\_USERS

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 разделе были разработаны основные функции, процедуры, с помощью которых будет возможна работа с базой данных без обращений к запросам. Экспорт и импорт данных в XML формат дадут возможность сохранения данных в надежном месте для обеспечения надежности хранения данных. В качестве СУБД была выбрана PostgreSQL, которая оказалась очень удобной и надежной в использовании. Были созданы роли, разграничивающие возможность пользователей во избежание конфликтов. Это все помогло оптимизировать базу данных и сделать пользование ею более удобным.

5 Тестирование

## 5.1 Тестирование производительности базы данных

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

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

|  |
| --- |
| EXPLAIN ANALYZE SELECT Category\_id FROM category WHERE Category\_id = 5;  create index catname on category(category\_id); |

Листинг 5.1 – Запрос к таблице genre

Запрос к таблице genre показывает, что его стоимость выполнения с Total Time равным 1.172мс говорит о том, что время выполнения будет большим при огромном количестве запросов и в 10 раз больше планируемого времени. Результаты запроса будут представлены на рисунке 5.2.

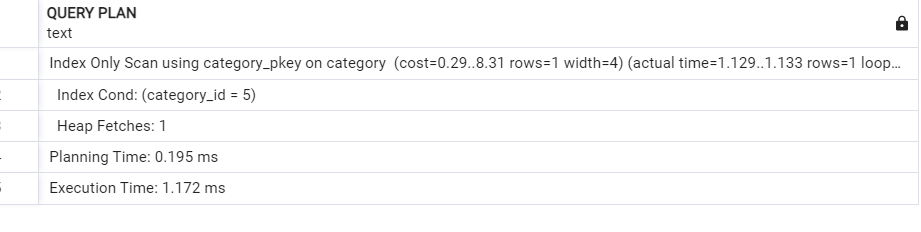


Рисунок 5.2 – Результат выполнения запроса

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

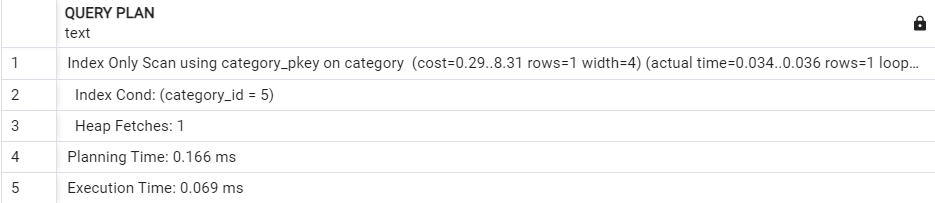


Рисунок 5.3 – Результат выполнения запроса

После создания индекса время сократилось до 0.166мс, что в 10 раз быстрее предыдущего запроса.

Тестирование показало, что создание индекса на поле category\_id значительно улучшило производительность запроса к таблице Category. Время выполнения оказалось в 10 раз быстрее запроса без индекса. Может в рамках одного запроса 1 миллисекунда покажется малым временем, но в констексте крупного проекта с огромным количеством пользователей это очень хороший показатель. Поэтому создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

## 5.2 Вывод

В разделе было проведено тестирование базы данных на самой заполненной таблице Category. Результат показал, что время выполнения запроса оказалось в 10 раз быстрее планируемого. Благодаря создания индекса по соответствующему столбцу поиск данных с 1.172мс сократился до 0.166мс, что ускорило поиск в 10 раз. На основе этого можно сделать вывод, что создание индексов по столбцам, к которым чаще всего обращаются или содержащее огромное количество данных даст очень большое преимущество во времени выполнения запроса.

6 Руководство по использованию программного средства

## 6.1 Руководство пользователя

Как только пользователь откроет приложение, нужно будет зарегистрироваться или войти, введя данные. Если пользователь зайдет как гость, то получит доступ только к просмотру каталога, а если зайдет как зарегистрированный пользователь, то сможет добавлять товары в корзину и пользоваться корзиной. Страница регистрации показана в рисунке 6.1.

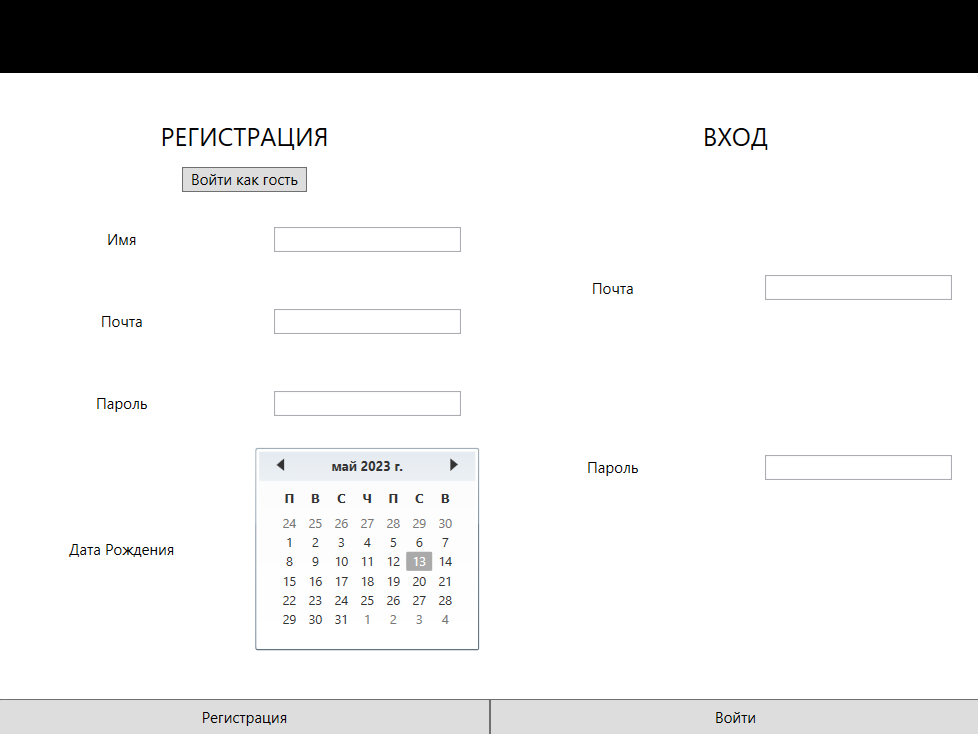


Рисунок 6.1 – Главная страница

После регистрации или авторизации, пользователь переходит на главную страницу со списком товаров. Пользователь может добавлять товары в корзину и перейти в страницу с товарами, которые пользователь добавил в корзину. На главной странице есть возможность поиска товара по названию и по категории. Главная страница представлена в рисунке 6.2

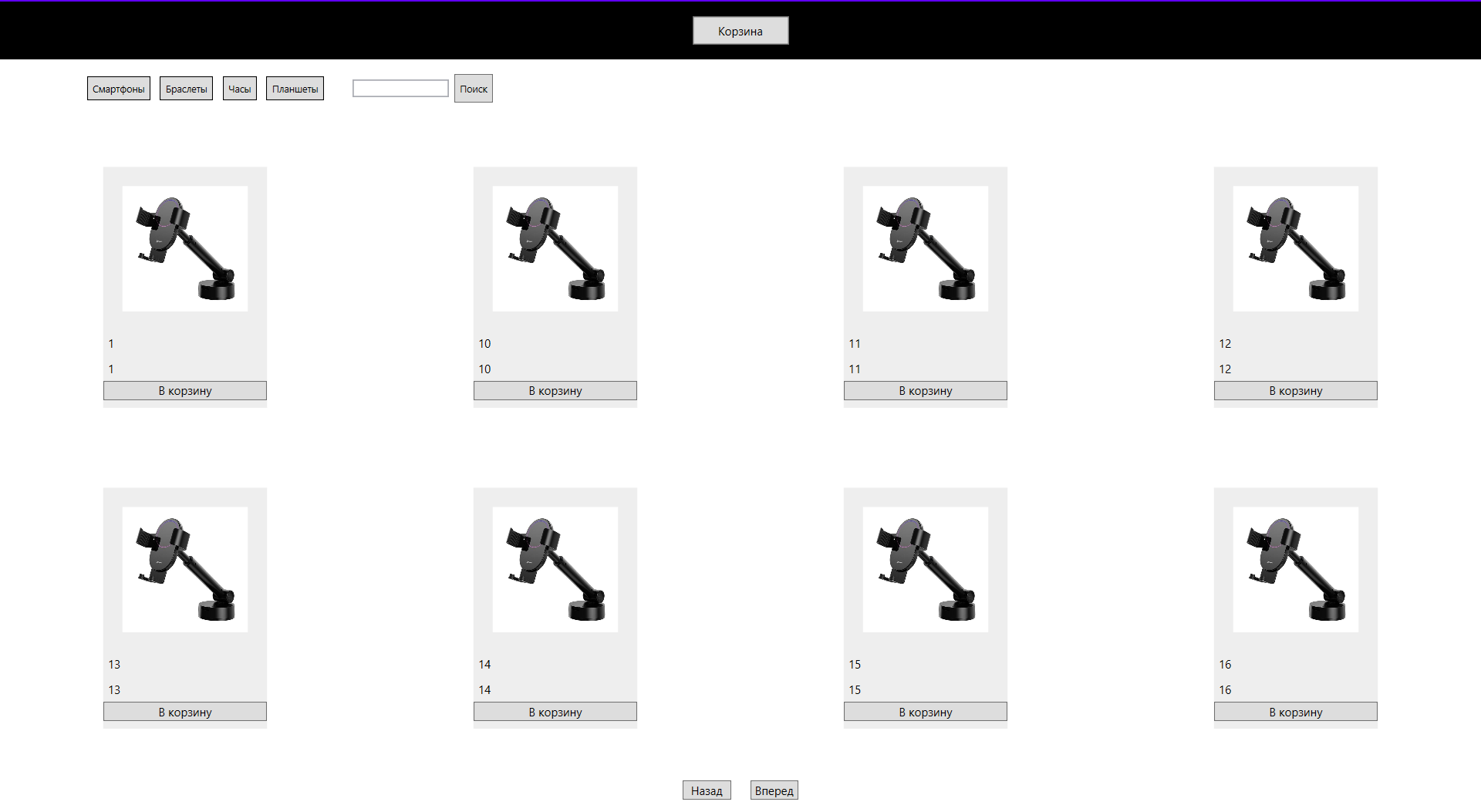


Рисунок 6.2 – Главная страница

Чтобы зайти как администратор или менеджер, нужно ввести во вкладке “Войти” данные администратора или менеджера соответственно. На странице администратора есть возможность добавлять, удалять, редактировать список товаров и пользователей. На странице менеджера есть возможность управления только каталогом товаров. Рисунок страницы администратора представлен на рисунке 6.3.



Рисунок 6.3 – Страница администратора

Чтобы перейти в страницу с корзиной, пользователю следует нажать на кнопку “Корзина” вверху главной страницы. Страница с корзиной представлена на рисунке 6.4.

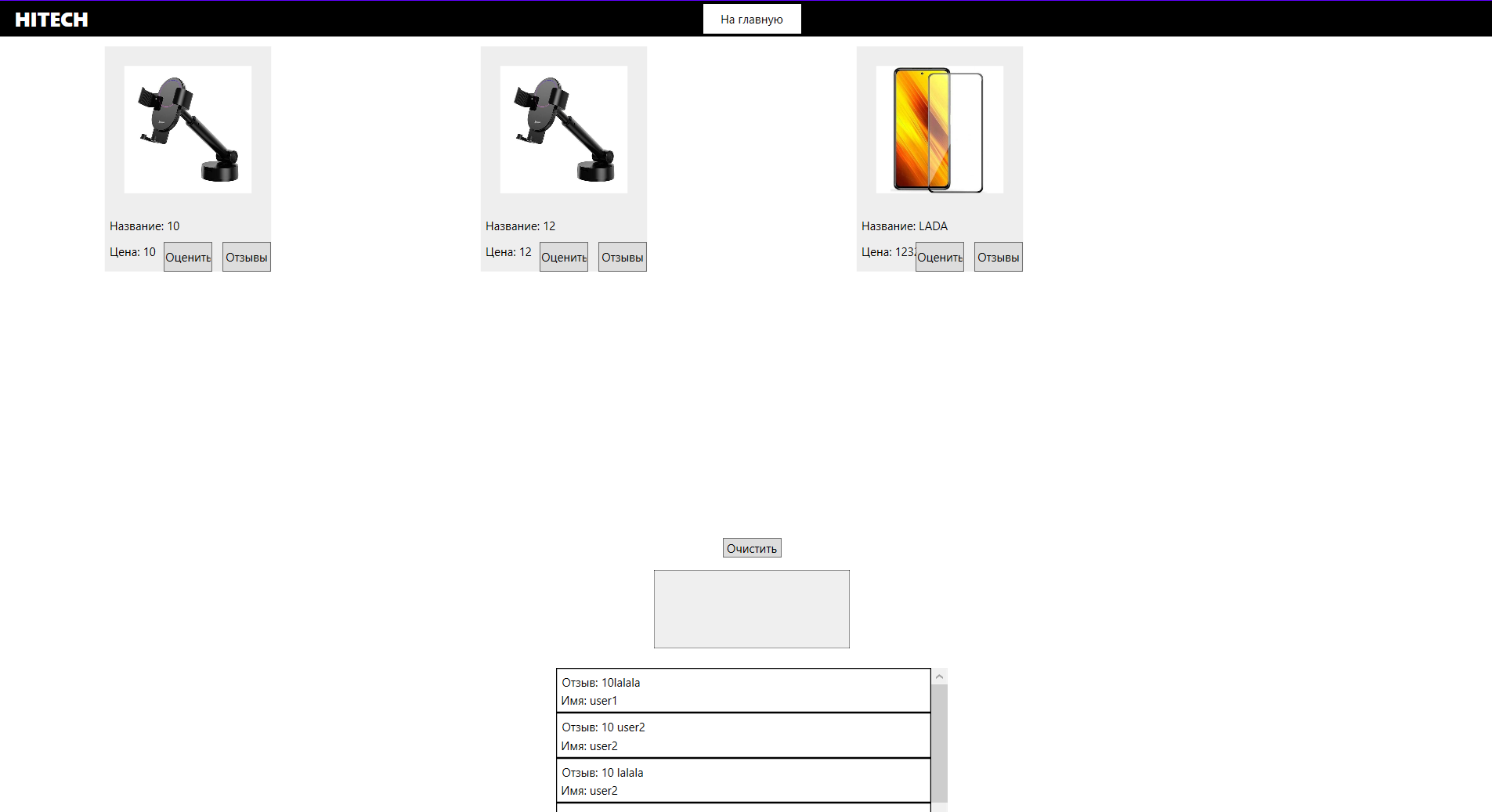


Рисунок 6.4 – Страница с корзиной

На странице с корзиной так же можно просматривать отзывы к товарам и оценивать их с помощью текстового поля. Кнопка “Очистить” очищает корзину у соответствующего пользователя

## 6.2 Установка приложения

Для запуска приложение необходимо запустить HiTech.exe файл. Подключение к базе данных проходит автоматически.

## 6.3 Вывод

В разделе было описано руководство пользователя, а также основные функции приложения для работы с товарами и пользователями в интернет-магазине. Приложение было написано на языке C# и оно работает на базе фреймворка WPF. Пользователь может регистрироваться или входить, искать товары по имени или категории, добавлять товары в корзине. Администратор может изменять данные товаров и пользователей. Менеджер может изменять только данные товаров. Приложение возможно расширять различными функциями в дальнейшем, чтобы оно было более конкурентным на рынке.

# Заключение

База данных – основа любого проекта, в котором нужно будет хранить информацию и выводить ее куда-либо. В проекте была поставлена задача разработки базы данных для интернет-магазина электроники.

В качестве СУБД была выбрана PostgreSQL, которая всегда славилась надежностью и удобностью в использовании.

Для удобного использования базы данных были разработаны процедуры и функции, которые облегчают получать, редактировать и удалять данные. Пользователям, администраторам и менеджерам не нужно будет вникать в логику запроса каждый раз, когда им нужно будет обращаться к данным таблиц.

Технологией был выбран полнотекстовый поиск, который обеспечивает поиск по названию товара, даже если запрос не очень точный и пользователь помнить части от названия товара.

Было разработано приложение на языке C# и на базе фреймворка WPF, которое позволяет визуально показать работу функций и процедур базы данных.

Разработанная база данных может быть использована для небольших интернет-магазинов, где нет огромного количества ролей, каждой из которых нужно выдавать свои права и обязанности. В дальнейшем база данных может расширяться под нужды владельца для более комфортного использования.

# Список литературных источников

1 5 элемент [Электронный ресурс] / Режим доступа: https://5element.by – Дата доступа: 20.04.2023.

2 7745.by [Электронный ресурс] / Режим доступа: https://7745.by – Дата доступа: 20.04.2023.

3 PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 19.04.2023.

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

5 Habr.com [Электронный ресурс] / Режим доступа: https://habr.com/ru– Дата доступа: 27.04.2023.

# Приложение А

|  |
| --- |
| --Создание табличных пространств--  create tablespace TS\_USER  location 'D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\Tablespaces\TS\_USER';  create tablespace TS\_PRODUCT  location 'D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\Tablespaces\TS\_PRODUCT';  --Создание таблиц--  alter table Users alter column USER\_DATE type date  using to\_date(user\_date,'YYYY-MM-DD');  create table Users(  user\_id serial primary key,  user\_role serial,  user\_name varchar(30) not null,  basket\_id serial,  user\_password varchar(100) not null,  user\_mail varchar(100) unique not null,  user\_date varchar(100) not null,  constraint FK\_USERS\_BASKET foreign key (basket\_id) references Basket(basket\_id),  constraint FK\_USERS\_ROLES foreign key (user\_role) references Roles(role\_id)  ) tablespace TS\_USER;  drop table Users cascade;  drop table Basket cascade;  create table Basket(  basket\_id serial ,  product\_id serial ,  constraint FK\_BASKET\_PRODUCT foreign key (product\_id) references Product(product\_id)  ) tablespace TS\_USER;  drop table Basket;  create table Roles(  role\_id serial primary key,  role\_name varchar(100) not null  ) tablespace TS\_USER;  insert into Roles values(1,'root\_role');  insert into Roles values(2,'mamager\_role');  insert into Roles values(3,'user\_role');  create table Commentaries(comment\_id serial primary key,  user\_id integer not null,  product\_id integer not null,  comment\_text varchar(300),  constraint FK\_COMM\_PRODUCT foreign key (product\_id) references Product(product\_id),  constraint FK\_COMM\_USER foreign key (user\_id) references Users(user\_id)  ) tablespace TS\_PRODUCT;    create table Product(  product\_id serial primary key,  category\_id serial,  product\_name varchar(100) unique not null,  product\_price serial not null,  product\_img varchar(100),  product\_color varchar(100) not null,  product\_date date not null,  product\_country varchar(100) not null,  product\_description varchar(200) not null,  product\_comments varchar(100)  constraint FK\_PRODUCT\_CATEGORY foreign key (category\_id) references Category(category\_id)  ) tablespace TS\_PRODUCT;  create index INDEX\_PRODUCT\_NAME\_LOWER on Product(product\_name);  drop index INDEX\_TSV\_APHORISM;  create index if not exists INDEX\_TSV\_APHORISM ON Product USING gin(text\_tsv);  drop function SEARCH\_PRODUCT\_BY\_NAME;  explain analyze select SEARCH\_PRODUCT\_BY\_NAME('am');  EXPLAIN ANALYZE SELECT Category\_id FROM category WHERE Category\_id = 5;  create index catname on category(category\_id);  EXPLAIN ANALYZE SELECT product\_name FROM Product WHERE lower(product\_name) ILIKE '%Samsung%';  create table Category(  category\_id serial primary key,  category\_name varchar(50) not null  ) tablespace TS\_PRODUCT;  insert into Category values(1,'Смартфон');  insert into Category values(2,'Часы');  insert into Category values(3,'Планшет');  insert into Category values(4,'Браслет');  ALTER TABLE Product ADD COLUMN text\_tsv tsvector;  UPDATE Product SET text\_tsv = to\_tsvector('russian', product\_name)  || to\_tsvector('russian', product\_description);  select text\_tsv from Product;  select \* from Category;  create table Rating(  rating\_id serial primary key,  user\_id serial not\_null,  product\_id serial not null,  rate integer not null,  constraint FK\_RATING\_USER foreign key (user\_id) references Users(user\_id),  constraint FK\_RATING\_PRODUCT foreign key (product\_id) references Product(product\_id)  ) tablespace TS\_PRODUCT; |

Листинг 1 – Скрипты создания таблиц и табличных пространств

|  |
| --- |
| --Создание ролей--  create role role\_root;  create role role\_user;  create role role\_manager;  create role role\_guest;  --Создание пользователей--  create user root\_user1 password 'root';  grant role\_root to root\_user1;  create user user1 password '12345';  grant role\_user to user1;  create user manager\_user1 password '12345';  grant role\_manager to manager\_user1;  create user guest\_user1 password '12345';  grant role\_guest to guest\_user1;  --Выдача привелегий role\_guest--  GRANT EXECUTE ON FUNCTION search\_product\_by\_name TO role\_guest;  GRANT EXECUTE ON FUNCTION search\_product\_by\_category TO role\_guest;  GRANT EXECUTE ON FUNCTION REGISTRATION TO role\_guest;  GRANT insert,select ON TABLE Users to role\_guest;  GRANT insert,select ON TABLE Basket to role\_guest;  select \* from Users;  grant role\_guest to guest\_user1;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_id\_seq TO role\_guest;  GRANT USAGE, SELECT ON SEQUENCE users\_basket\_id\_seq TO role\_guest;  GRANT EXECUTE ON FUNCTION FUNC\_FOR\_TR\_DATE TO role\_guest;  --Выдача привелегий role\_root--  grant connect on database "HiTech" to role\_root;  GRANT ALL ON SCHEMA public TO role\_root;  grant all privileges on database "HiTech" to role\_root;  grant all privileges on tablespace TS\_USER to role\_root;  grant all privileges on tablespace TS\_PRODUCT to role\_root;  select \* from users;  --Выдача привелегий role\_user--  GRANT EXECUTE ON FUNCTION REGISTRATION TO role\_user;  GRANT EXECUTE ON FUNCTION signin TO role\_user;  GRANT EXECUTE ON PROCEDURE add\_rating TO role\_user;  GRANT EXECUTE ON PROCEDURE ADD\_PRODUCT\_TO\_BASKET TO role\_user;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER TO role\_user;  GRANT EXECUTE ON PROCEDURE update\_password TO role\_user;  GRANT EXECUTE ON PROCEDURE delete\_from\_basket TO role\_user;  GRANT EXECUTE ON PROCEDURE ADD\_COMMENT TO role\_user;  GRANT EXECUTE ON FUNCTION search\_product\_by\_name TO role\_user;  GRANT EXECUTE ON FUNCTION search\_product\_by\_category TO role\_user;  GRANT EXECUTE ON FUNCTION search\_user\_by\_name TO role\_user;  GRANT EXECUTE ON FUNCTION CHECK\_BASKET\_FUNC TO role\_user;  grant select on PRODUCT\_INFO TO role\_user;  grant select on Users TO role\_user;  GRANT SELECT, INSERT, UPDATE ON TABLE Users TO role\_user;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Rating TO role\_user;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Rating TO role\_user;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Commentaries TO role\_user;  GRANT SELECT ON TABLE PRODUCT TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE basket\_basket\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE basket\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_rating\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_user\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE rating\_product\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_id\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_role\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_basket\_seq TO role\_user;  GRANT USAGE, SELECT ON SEQUENCE commentaries\_comment\_id\_seq TO role\_user;  -----role\_manager-----  GRANT EXECUTE ON FUNCTION add\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_product TO role\_manager;  GRANT EXECUTE ON FUNCTION REGISTRATION TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_product TO role\_manager;  GRANT EXECUTE ON FUNCTION add\_product TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_user TO role\_manager;  GRANT EXECUTE ON PROCEDURE update\_password TO role\_manager;  GRANT EXECUTE ON PROCEDURE delete\_user TO role\_manager;  GRANT SELECT, INSERT, UPDATE ON TABLE Users TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Product TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Basket TO role\_manager;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Category TO role\_manager;  GRANT USAGE, SELECT ON SEQUENCE users\_user\_id\_seq TO role\_manager;  GRANT USAGE, SELECT ON SEQUENCE product\_product\_id\_seq TO role\_manager; |

Листинг 2 – Скрипты создание пользователей и ролей

|  |
| --- |
| --Создание функции регистрации--  CREATE OR REPLACE FUNCTION REGISTRATION(  user\_name\_arg varchar(30),  user\_mail\_arg varchar(100),  user\_password\_arg varchar(100),  user\_date\_arg date    )  RETURNS integer  AS $$  DECLARE  USER\_ID INTEGER;  BEGIN  INSERT INTO Users(user\_name , user\_mail , user\_password, user\_date , user\_role)  VALUES (user\_name\_arg, user\_mail\_arg, user\_password\_arg, user\_date\_arg, 3)  RETURNING Users.user\_id,Users.basket\_id INTO USER\_ID ;  INSERT INTO Basket(basket\_id, product\_id) values(USER\_ID, 5);  RETURN USER\_ID ;  END;  $$ LANGUAGE PLPGSQL;  drop function REGISTRATION;  SELECT REGISTRATION('administrator', 'administrator@gmail.com', 'administrator2005', '2005-03-14'); --триггеры пароля и даты  delete from Basket;  select \* from Commentaries ;  --Создание функции входа--  CREATE OR REPLACE FUNCTION SIGNIN(user\_mail\_arg VARCHAR(100), user\_password\_arg VARCHAR(100))  RETURNS integer  AS $$  DECLARE  role\_id varchar(100) = ( select user\_role from Users where user\_mail = user\_mail\_arg);  passw varchar(100);  BEGIN  SELECT user\_password INTO passw FROM Users WHERE user\_mail = user\_mail\_arg ;    IF passw IS NULL THEN  raise exception 'Почта не верна';  elseif passw = user\_password\_arg then  return role\_id;  else raise exception ' Пароль не верен';  END IF;    RETURN role\_id;  END;  $$ LANGUAGE PLPGSQL;  drop FUNCTION SIGNIN;  SELECT SIGNIN('manager@gmail.com', 'manager2005');  --Создание функции добавления товара--  CREATE OR REPLACE FUNCTION ADD\_PRODUCT(category\_id\_arg INTEGER,product\_name\_arg VARCHAR(100),product\_price\_arg INTEGER,  product\_img\_arg VARCHAR(100), product\_color\_arg VARCHAR(100),product\_country\_arg VARCHAR(100),product\_description\_arg varchar(200))  RETURNS INTEGER  AS $$  DECLARE  product\_id\_add INTEGER;  product\_date\_arg date;  BEGIN  product\_date\_arg = current\_date;  INSERT INTO Product(category\_id,product\_name,product\_price,product\_img ,product\_color, product\_date, product\_country,product\_description)  VALUES(category\_id\_arg,product\_name\_arg,product\_price\_arg,product\_img\_arg,product\_color\_arg,product\_date\_arg,product\_country\_arg,product\_description\_arg)  RETURNING Product.product\_id into product\_id\_add;  RETURN product\_id\_add;  UPDATE Product SET text\_tsv = to\_tsvector('russian', product\_name)  || to\_tsvector('russian', product\_description);  END;  $$ LANGUAGE PLPGSQL;  select ADD\_PRODUCT(2,'Samsung Galaxy A20',2000,'A20.img','Черный','Южная Корея','Это смартфон');  --Создание функции просмотра отзывов товару--  create or replace function SELECT\_COMMENTS( product\_id\_arg integer)  returns table(user\_name varchar(100),comments\_text varchar(100))  LANGUAGE PLPGSQL AS $$  BEGIN  RETURN QUERY  SELECT u.user\_name, c.comment\_text  FROM Users u  join Commentaries c on u.user\_id = c.user\_id where c.product\_id =product\_id\_arg ;  END;  $$;  select \* from SELECT\_COMMENTS(18);  --Создание функции просмотра товаров в корзине--  CREATE OR REPLACE FUNCTION CHECK\_BASKET\_FUNC(basket\_id\_arg integer) returns table  (product\_id integer,product\_name varchar(100),product\_price integer, product\_img varchar(100))  LANGUAGE PLPGSQL AS $$  BEGIN  IF basket\_id\_arg IS NOT NULL THEN  RETURN QUERY  SELECT b.product\_id, p.product\_name, p.product\_price, p.product\_img  FROM Basket b join Product p on b.product\_id = p.product\_id where basket\_id = basket\_id\_arg;  END IF;  END;  $$;  select \* from CHECK\_BASKET\_FUNC(48) order by product\_name offset 0 rows fetch next 8 rows only;  --Создание функции вывода рейтинга товара--  CREATE OR REPLACE FUNCTION AVG\_RATE(product\_id INTEGER) RETURNS NUMERIC AS $$  DECLARE  rating INTEGER;  count\_ratings INTEGER;  BEGIN  SELECT SUM(rate), COUNT(\*) INTO rating, count\_ratings  FROM Rating WHERE Rating.product\_id = $1;  IF count\_ratings = 0 THEN RETURN 0;  END IF;  RETURN rating :: NUMERIC / count\_ratings;  END;$$  language PLPGSQL;    CREATE or replace FUNCTION SEARCH\_USER\_BY\_NAME(user\_name\_arg varchar(100))  RETURNS TABLE(  user\_name varchar(100),  user\_mail varchar(100),  user\_date date  ) as $$  declare  begin  RETURN QUERY  SELECT USER\_INFO.user\_name,USER\_INFO.user\_mail,  USER\_INFO.user\_date FROM USER\_INFO  where USER\_INFO.user\_name ilike '%'|| user\_name\_arg || '%';  end;  $$ language PLPGSQL;  --Создание функции поиска товара по названию--  CREATE or replace FUNCTION SEARCH\_PRODUCT\_BY\_NAME(product\_name\_arg varchar(100))  RETURNS TABLE(  product\_name varchar(100),  product\_price integer,  product\_color varchar(100),  product\_country varchar(100),  product\_date date,  product\_img varchar(100)  ) as $$  begin  RETURN QUERY  SELECT Product.product\_name,Product.product\_price,  Product.product\_color, Product.product\_country,  Product.product\_date,Product.product\_img FROM Product  where  Product.product\_name ilike '%'|| product\_name\_arg || '%' or  Product.text\_tsv @@ plainto\_tsquery('russian', product\_name\_arg);  end;  $$ language PLPGSQL;  drop function SEARCH\_PRODUCT\_BY\_NAME;  select \* from SEARCH\_PRODUCT\_BY\_NAME('Am');  --Создание функции поиска товара по категории--  CREATE or replace FUNCTION SEARCH\_PRODUCT\_BY\_CATEGORY(category\_name\_arg varchar(100))  RETURNS TABLE(  product\_name varchar(100),  product\_price integer,  product\_color varchar(100),  product\_country varchar(100),  product\_date date,  category\_name varchar(100),  product\_img varchar(100)  ) as $$  begin  RETURN QUERY  SELECT PRODUCT\_INFO.product\_name,PRODUCT\_INFO.product\_price,  PRODUCT\_INFO.product\_color, PRODUCT\_INFO.product\_country,  PRODUCT\_INFO.product\_date, PRODUCT\_INFO.category\_name, PRODUCT\_INFO.product\_img FROM PRODUCT\_INFO  where PRODUCT\_INFO.category\_name ilike '%'|| category\_name\_arg || '%';  end;  $$ language PLPGSQL; |

Листинг 3 – Скрипты создания функций

|  |
| --- |
| --Представление с информацией о товаре--  CREATE VIEW PRODUCT\_INFO AS  SELECT p.product\_name,  p.product\_price,  c.category\_name,  p.product\_color,  p.product\_date,  p.product\_country,  p.product\_img,  p.product\_description,  avg\_rate(p.product\_id) AS avg\_rating  FROM product p  JOIN category c ON p.category\_id = c.category\_id;  drop view PRODUCT\_INFO;  select \* from PRODUCT\_INFO;  select \* from Users;      --Представление с информацией о пользователе--  CREATE VIEW USER\_INFO AS  SELECT u.user\_id,  u.user\_name,  u.user\_mail,  u.user\_date,  u.user\_password,  r.role\_name  FROM users u  JOIN roles r ON u.user\_role = r.role\_id;  select  --Представление с информацией лучших продуктов--  CREATE VIEW BEST\_PRODUCTS  AS  SELECT p.product\_name,  p.product\_price,  c.category\_name,  p.product\_color,  p.product\_date,  p.product\_country,  p.product\_description,  avg\_rate(p.product\_id) AS avg\_rating  FROM product p  JOIN category c ON p.category\_id = c.category\_id  ORDER BY avg\_rating DESC LIMIT 10;  select \* from BEST\_PRODUCTS; |

Листинг 4 – Скрипты создания представлений

|  |
| --- |
| --Создание триггера, реагирующего на дату рождения позже настоящей--  CREATE or replace FUNCTION FUNC\_FOR\_TR\_DATE() RETURNS TRIGGER AS $$  BEGIN  IF NEW.user\_date >= CURRENT\_DATE THEN  RAISE EXCEPTION 'Такой даты не существует';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE or replace TRIGGER TR\_DATE  BEFORE INSERT OR UPDATE  ON Users  FOR EACH ROW EXECUTE FUNCTION FUNC\_FOR\_TR\_DATE();  drop FUNCTION FUNC\_FOR\_TR\_DATE() cascade;  drop FUNCTION FUNC\_FOR\_TR\_DATE() cascade;  drop trigger TR\_DATE cascade;  select \* from Users;  --Создание триггера, реагирующий на валидацию пароля--  CREATE or replace FUNCTION FUNC\_FOR\_TR\_PASSWORD() RETURNS TRIGGER AS $$  BEGIN  IF char\_LENGTH(NEW.user\_password) < 8 THEN  RAISE EXCEPTION 'Пароль должен содержать не менее 8 символов';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE TRIGGER TR\_PASSWORD  BEFORE INSERT OR UPDATE ON Users  FOR EACH ROW  EXECUTE FUNCTION FUNC\_FOR\_TR\_PASSWORD();  --Создание триггера, реагирующего на оценивание товара--  CREATE FUNCTION FUNC\_FOR\_TR\_RATE() RETURNS TRIGGER AS $$  BEGIN  IF NEW.rate > 10 or NEW.rate < 0 THEN  RAISE EXCEPTION 'Оцените в пределах от 0 до 10';  END IF;  RETURN NEW;  END;  $$ LANGUAGE PLPGSQL;  CREATE or replace TRIGGER TR\_RATE  BEFORE INSERT OR UPDATE  ON Rating  FOR EACH ROW EXECUTE FUNCTION FUNC\_FOR\_TR\_RATE(); |

Листинг 5 – Скрипт создания триггеров

|  |
| --- |
| --------------Экспорт XML----------------  ------------------------------------------------  CREATE EXTENSION adminpack;  CREATE EXTENSION IF NOT EXISTS file\_fdw;  CREATE OR REPLACE FUNCTION export\_users(  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('Users', 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;  $BODY$;  SELECT export\_users('D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\XML\USERS.XML');  --------------ИМПОРТ-----------------------  CREATE OR REPLACE FUNCTION import\_data\_from\_xml(filename TEXT) RETURNS VOID AS $$  DECLARE  xml\_data TEXT;  BEGIN  -- Читаем XML данные из файла  xml\_data := pg\_read\_file(filename, 0, 1000);  -- Парсим XML данные и выполняем импорт  WITH xml\_rows AS (  SELECT unnest(xpath('/root/row', xmlparse(content xml\_data))) AS xml\_row  )  INSERT INTO Users (user\_id, user\_name,user\_password,user\_date,basket\_id,user\_role,user\_mail)  SELECT  (xpath('/row/user\_id/text()', xml\_row))[1]::text::integer AS user\_id,  (xpath('/row/user\_name/text()', xml\_row))[1]::TEXT AS user\_name,  (xpath('/row/user\_password/text()', xml\_row))[1]::TEXT AS user\_password,  (xpath('/row/user\_date/text()', xml\_row))[1]::text::date AS user\_date,  (xpath('/row/basket\_id/text()', xml\_row))[1]::text::int AS basket\_id,  (xpath('/row/user\_role/text()', xml\_row))[1]::text::int AS user\_role,  (xpath('/row/user\_mail/text()', xml\_row))[1]::TEXT AS user\_mail  FROM  xml\_rows;  -- Выводим сообщение об успешном импорте  RAISE NOTICE 'Данные импортированы из файла %', filename;  END;  $$ LANGUAGE plpgsql;  ------------------------------------------------  SELECT import\_data\_from\_xml('D:\MYDOCUMENTS\BGTU\DB\_KURS\HiTech\XML\USERS.XML'); |

Листинг 6 – Скрипт для импорта и экспорта XML файлов

|  |
| --- |
| --ЗАПОЛНЕНИЕ 100000 СТРОК----  CREATE OR REPLACE FUNCTION INSERT\_CATEGORY()  RETURNS VOID AS $$  DECLARE  I INTEGER := 5;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO Category (category\_name,category\_id) VALUES ('CATEGORY ' || I,I);  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL;  select INSERT\_CATEGORY();  explain select category\_name from Category ;  create index INDEX\_CATEGORY\_NAME on Category(category\_name) ;  drop index INDEX\_CATEGORY\_NAME; |

Листинг 7 – Скрипт заполнения таблицы 100000 строками