МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных для виртуальной торговой площадки c использованием технологии репликации»

Выполнил студент Д. В. Круглик

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Содержание

[Введение 5](#_Toc167082581)

[1 Аналитический обзор литературы по теме проекта 6](#_Toc167082582)

[1.1 Постановка задачи 6](#_Toc167082583)

[1.1.1 Аналог Ozon 6](#_Toc167082584)

[1.1.2 Аналог Aliexpress 7](#_Toc167082585)

[1.1.3 Аналог Wildberries 8](#_Toc167082586)

[1.2 Функциональные требования 9](#_Toc167082587)

[1.3. Определение вариантов использования 10](#_Toc167082588)

[1.4 Вывод 11](#_Toc167082589)

[2 Анализ и проектирование архитектуры проекта. 12](#_Toc167082590)

[2.1 UML схема базы данных 12](#_Toc167082591)

[2.2 Описание информационных объектов и ограничений целостности 12](#_Toc167082592)

[2.3 Вывод 14](#_Toc167082593)

[3 Проектирование и разработка базы данных 15](#_Toc167082594)

[3.1 Табличные пространства базы данных 15](#_Toc167082595)

[3.2 Создание основных ролей и пользователей базы данных 15](#_Toc167082596)

[3.3 Вывод 17](#_Toc167082597)

[4 Разработка необходимых объектов 18](#_Toc167082598)

[4.1 Создание таблиц 18](#_Toc167082599)

[4.2 Создание триггеров 19](#_Toc167082600)

[4.3 Создание индексов 19](#_Toc167082601)

[4.4 Создание представлений 20](#_Toc167082602)

[4.5 Создание процедур 21](#_Toc167082603)

[4.6 Создание функций 22](#_Toc167082604)

[4.7 Вывод 23](#_Toc167082605)

[5 Описание процедур экспорта и импорта 24](#_Toc167082606)

[5.1 Экспорт данных в JSON 24](#_Toc167082607)

[5.2 Импорт данных из JSON 24](#_Toc167082608)

[5.3 Вывод 25](#_Toc167082609)

[6 Использование технологии репликации 26](#_Toc167082610)

[6.1 Настройка репликации 27](#_Toc167082611)

[6.2 Проверка работы репликации 33](#_Toc167082612)

[6.3 Вывод 34](#_Toc167082613)

[7 Тестирование производительности базы данных 35](#_Toc167082614)

[7.1 Тестирование производительности 35](#_Toc167082615)

[7.2 Вывод 36](#_Toc167082616)

[Заключение 38](#_Toc167082617)

[Список используемых источников 39](#_Toc167082618)

[Приложение А Листинг создания таблиц 40](#_Toc167082619)

[Приложение Б Листинг создания триггеров 42](#_Toc167082620)

[Приложение В Листинг процедур 44](#_Toc167082621)

[Приложение Г Листинг функций 53](#_Toc167082622)

# Введение

Цель данного проекта состоит в создании реляционной базы данных для интернет торговой площадки, обеспечивая доступ пользователю ко всем товарам, имеющимся в нем. Также требуется использовать подходящую технологию репликации базы данных для увеличения надежности и безопасности.

База данных представляет собой совокупность взаимосвязанных данных, обычно сохраняемых в электронном виде в компьютерной системе. Она используется для хранения, организации и управления как структурированными, так и неструктурированными данными. Реляционная база данных является наиболее распространенной формой организации, где данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка – кортеж или запись.

В данном проекте для управления базой данных была выбрана СУБД SQL Server Management Studio благодаря ее высокой надежности и производительности, что обеспечивает эффективное хранение, обработку и управление данными.

Для обеспечения стабильности и надежности приложения используется технология репликации, которая предполагает наличие дополнительных баз данных. Эта технология позволяет хранить актуальные данные на других бд, которые в свою очередь тоже явлюятся самостоятельными, обеспечивает синхронизицию с основной базой данных в реальном времени, а также изменения вступаю в силу в обе стороны, что увеличивает надежность сервиса в целом в случае сбоя.

Пояснительная записка содержит информацию о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1. Аналитический обзор литературы по теме проекта

## Постановка задачи

Сегодня интернет-магазины играют ключевую роль в нашей повседневной жизни, обеспечивая возможность приобретения разнообразных товаров и услуг без выхода из дома. В связи с этим, при создании базы данных для интернет-магазина необходимо основательно изучить и объективно оценить функциональные возможности, используемые технологии и прочие аспекты.

После проведения анализа требуется создать конкретный список функциональных требований к базе данных и разработать диаграмму использования. Основная задача проекта заключается в построении архитектуры приложения с интуитивно понятным интерфейсом для каждого пользователя, а также в создании базы данных и тестировании готового продукта. Основные требования к реализации включают:

* Определение ролей (покупатель, продавец);
* Добавление и удаление товаров на площадке (для продавцов);
* Поиск товаров по категориям и наименованию (для покупателей);
* Просмотр заказанных товаров (для покупателей);
* Просмотр выставленных товаров (для продавцов);
* Оформление заказа (Пользователь);
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

### 1.1.1 Аналог Ozon

Ozon [1] является одной из ведущих платформ для цифровой и физической дистрибуции товаров и услуг. Основным преимуществом этой платформы является ее удобство и простота использования. Пример интерфейса данного сервиса представлен на рисунке 1.1.

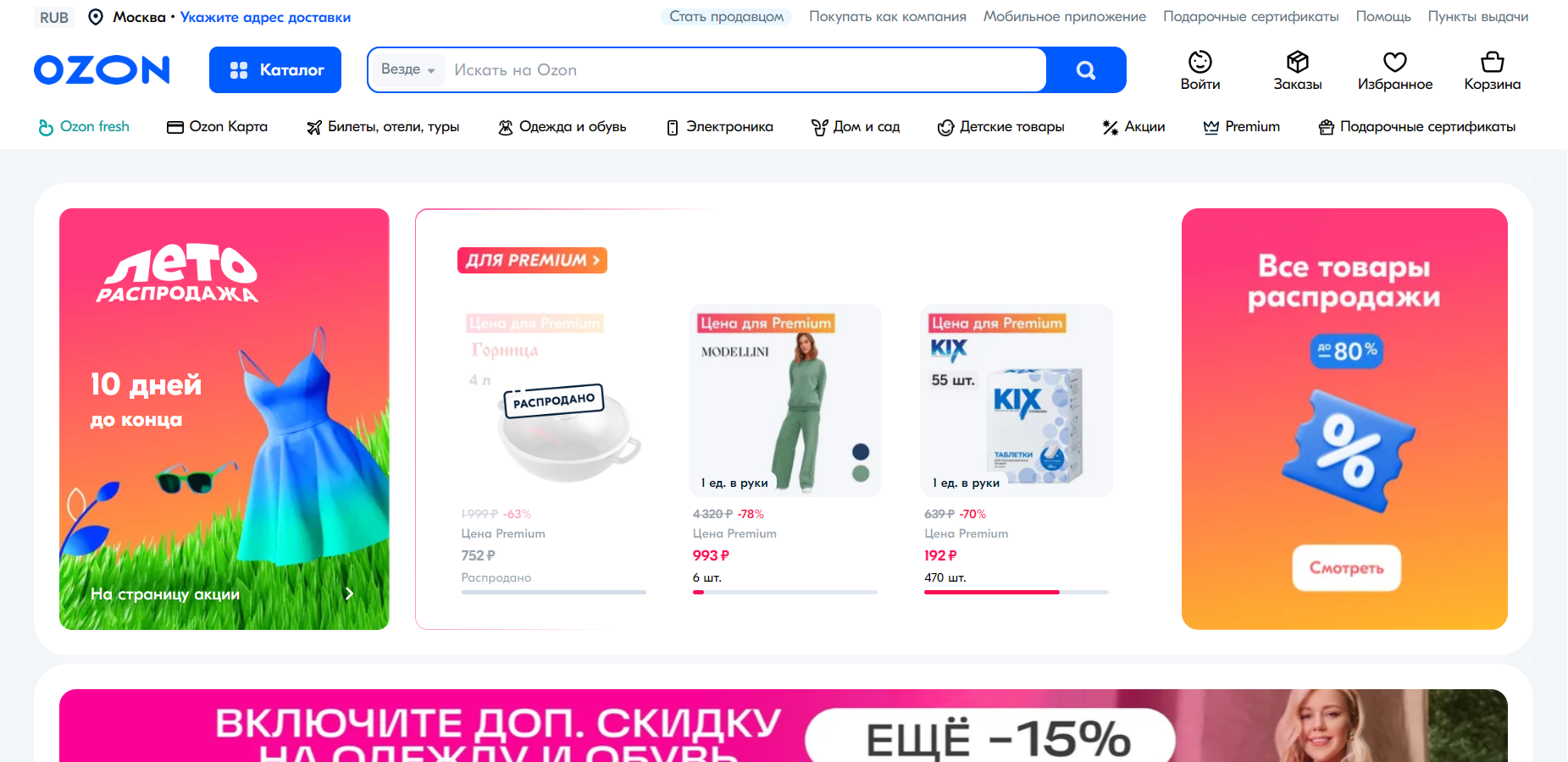


Рисунок 1.1 – Интерфейс сервиса Ozon

Ключевая функциональность платформы Ozon – это доступ к широкому ассортименту товаров. Пользователи могут легко найти товары по категориям, брендам, цене или популярности. Кроме того, Ozon предоставляет персонализированные рекомендации на основе истории покупок пользователя, а также список товаров, рекомендованных другими покупателями и экспертами платформы.

Пользователи могут создавать собственные списки желаемых товаров и делиться ими с друзьями или сохранять товары для последующего приобретения. Ozon также предоставляет пользователям обширную информацию о каждом товаре, включая описание, отзывы пользователей и технические характеристики.

Платформа использует множество баз данных в своей архитектуре, включая системы управления базами данных для хранения информации о товарах, продавцах, отзывах и т.д. Эти системы также используются для хранения информации о пользовательских аккаунтах, заказах и настройках.

В общем, Ozon использует масштабируемые и высокопроизводительные базы данных для обеспечения быстрого доступа к огромному объему информации о товарах и обработки данных для предоставления персонализированных рекомендаций и других функций платформы.

### 1.1.2 Аналог Aliexpress

AliExpress [2] — это международная платформа электронной коммерции. Она предоставляет широкий ассортимент товаров, предлагаемых различными продавцами из Китая и других стран, и ориентирована на глобальный рынок. Платформа известна своими низкими ценами и частыми распродажами, что делает товары еще более доступными для покупателей по всему миру. Пример интерфейса данного сервиса представлен на рисунке 1.2.

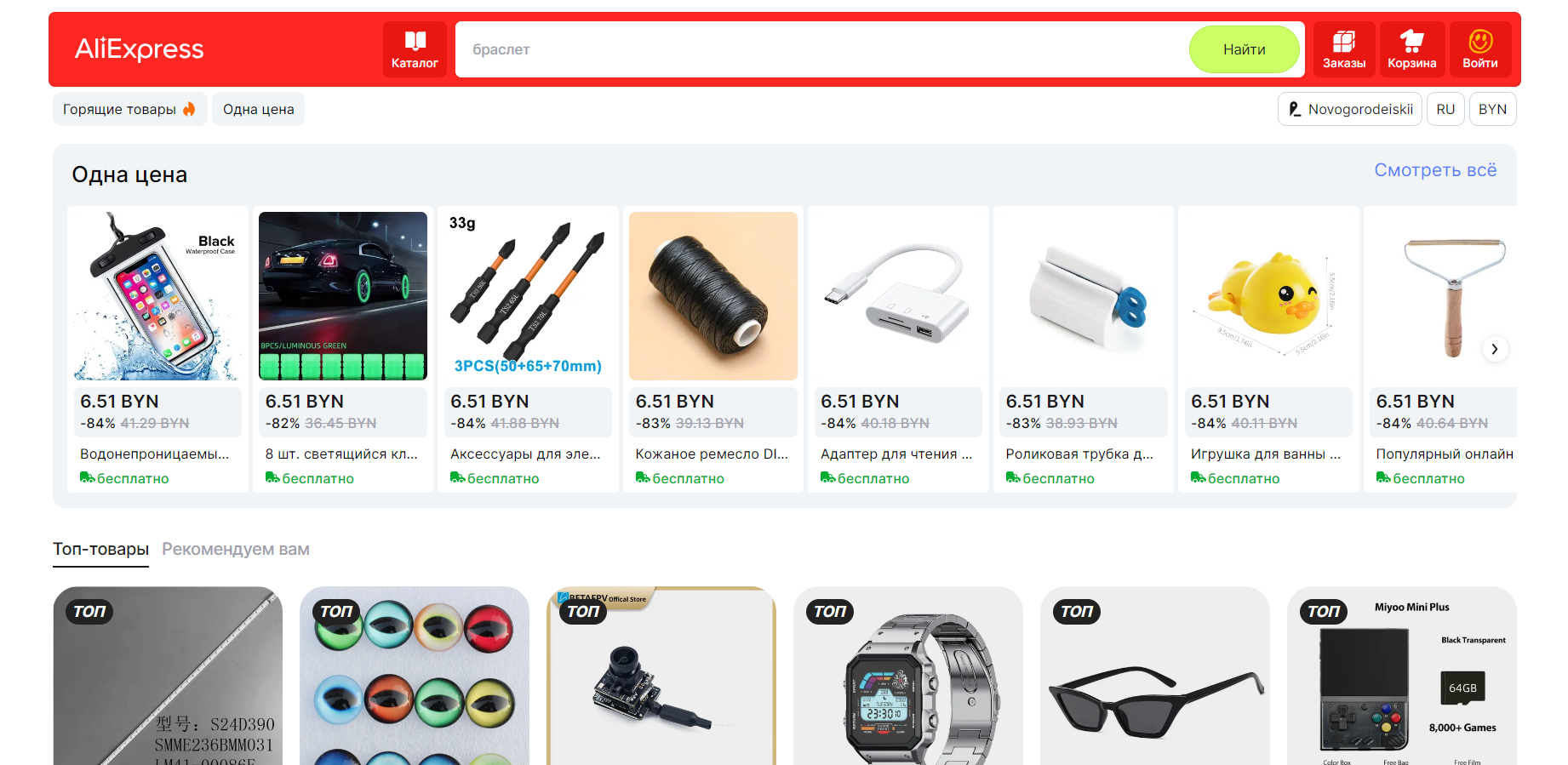


Рисунок 1.2 – Интерфейс сервиса Aliexpress

Одной из ключевых характеристик AliExpress является огромный ассортимент товаров, включающий электронику, моду, товары для дома, игрушки и многое другое. Продавцами могут быть как производители, так и реселлеры. Платформа работает в более чем 200 странах и регионах, предлагая интерфейс на множестве языков, таких как английский, испанский, французский, русский и другие, что делает ее доступной для международной аудитории.

База данных AliExpress играет критически важную роль в функционировании платформы. Она обеспечивает хранение, управление и быстрый доступ к огромному количеству данных, связанных с товарами, пользователями, заказами и транзакциями. Ежедневно на платформу добавляются новые продукты, заказы и отзывы, что требует масштабируемой и производительной базы данных.

Типы данных, с которыми работает база данных AliExpress, включают продуктовые данные (информация о товарах, описания, изображения, цены, запасы), пользовательские данные (профили пользователей, адреса доставки, платежные данные), транзакционные данные (заказы, платежи, статусы доставки) и отзывы с рейтингами. Для обеспечения производительности и масштабируемости AliExpress, вероятно, использует комбинацию реляционных и нереляционных баз данных.

Таким образом, AliExpress является одной из крупнейших мировых платформ электронной коммерции, успех которой в значительной мере зависит от эффективного управления ее базой данных. Использование передовых технологий позволяет платформе обеспечивать бесперебойную работу, безопасность и аналитические возможности, поддерживая конкурентоспособность и высокое качество обслуживания пользователей по всему миру.

### 1.1.3 Аналог Wildberries

Wildberries ещё один крупнейший интернет-магазин, который предлагает широкий ассортимент товаров, начиная от одежды и обуви, заканчивая электроникой и продуктами питания. Пример интерфейса данного сервиса представлен на рисунке 1.3.

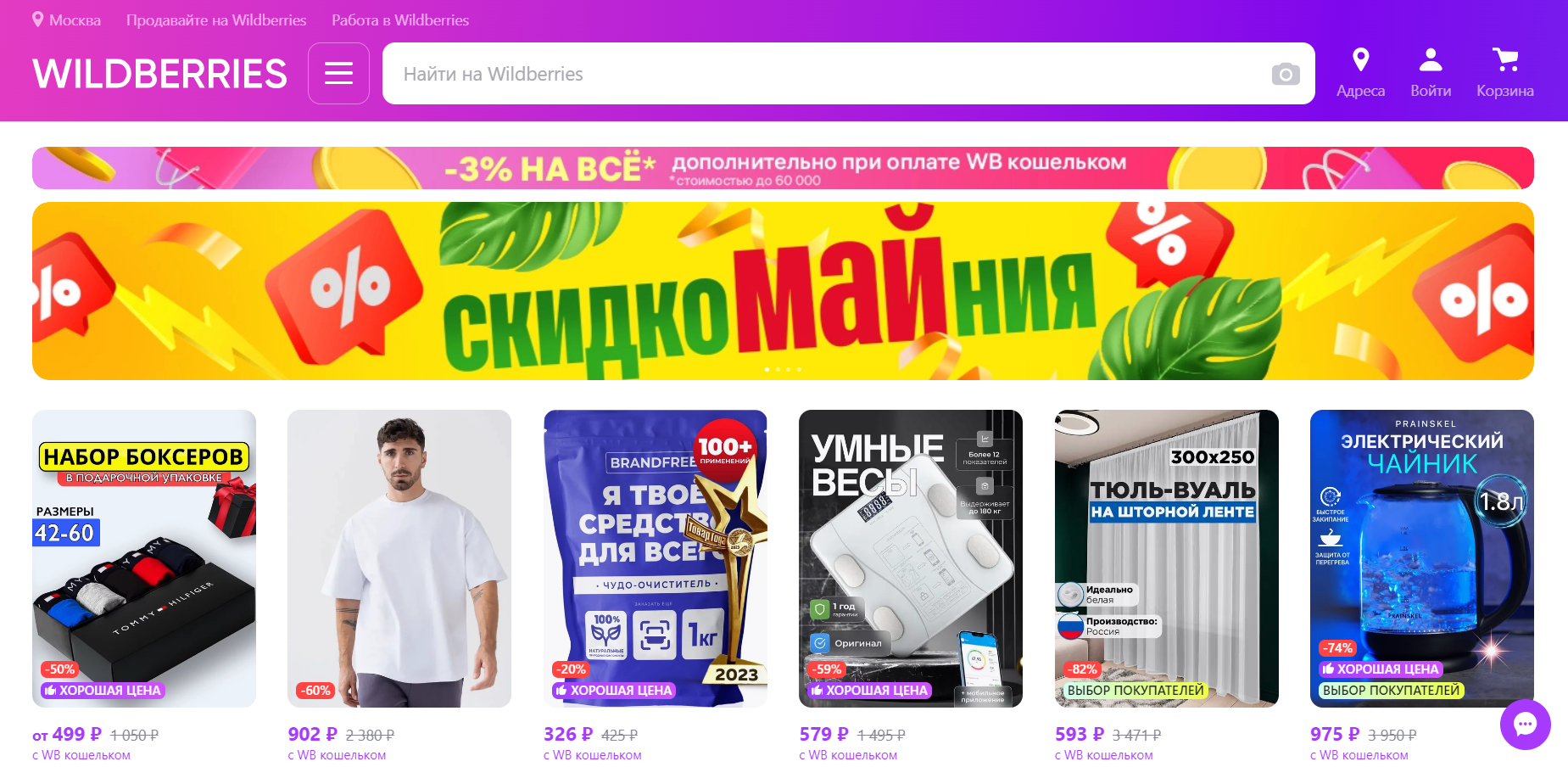


Рисунок 1.3 – Интерфейс сервиса Wildberries

Wildberries демонстрирует значительный рост благодаря своей стратегии, ориентированной на максимальное удобство и доступность для пользователей. Компания инвестирует в собственную логистическую инфраструктуру, что позволяет ей обеспечивать быструю и надежную доставку. Развитие сети пунктов самовывоза и складских комплексов по всей стране также способствует повышению уровня обслуживания клиентов.

Одним из ключевых факторов успеха Wildberries является использование современных технологий и обширной базы данных. Система управления данными позволяет эффективно анализировать предпочтения клиентов и предлагать персонализированные рекомендации, что повышает уровень удовлетворенности покупателей и стимулирует повторные покупки. Мощные аналитические инструменты помогают компании прогнозировать спрос и оптимизировать товарные запасы, минимизируя издержки и улучшая логистику.

Система управления базами данных Wildberries построена на основе современных технологий, обеспечивающих высокую производительность и безопасность. Использование облачных решений позволяет масштабировать ресурсы в зависимости от потребностей, что особенно важно в пиковые периоды продаж, такие как сезонные распродажи или праздничные акции.

Таким образом, успех Wildberries объясняется сочетанием стратегического подхода к развитию логистики и инфраструктуры, использования передовых технологий для анализа данных и персонализации, а также гибкой и масштабируемой платформы, поддерживающей множество продавцов и широкий ассортимент товаров.

## 1.2 Функциональные требования

Функциональные требования базы данных определяют способы обработки данных и предоставления пользователю определенной роли необходимой функциональности. Это включает в себя указание на способы хранения и структурирования данных, методы поиска и выборки данных, процессы обновления информации и механизмы защиты данных. Кроме того, такие требования могут охватывать интеграцию базы данных с другими системами и программным обеспечением. Для маркетплейса функциональные требования могут включать функции для хранения информации о товарах, возможность поиска товаров по различным категориям и критериям, создание и удаление товаров, а также функции оценки товаров.

По результатам рассмотренных аналогов основные задачи для базы данных интернет торговой площадки:

* разработать процедуры и функции для обработки информации и действий пользователя;
* разработать функционал оценки товаров;
* разработать функционал для оформления заказа покупателем;
* разработать функционал удаления товара.
* разработать функционал поиска товаров.

## 1.3. Определение вариантов использования

Кроме функциональных требований, необходимо также установить роли пользователей и их сценарии взаимодействия с системой. Эти сценарии описывают, каким образом пользователи будут взаимодействовать с системой в зависимости от своих ролей. Обычно такие сценарии представляются с помощью UML диаграмм, которые наглядно демонстрируют взаимодействие пользователей с системой.

В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Гость
* Покупатель
* Продавец

На основе предоставленного списка ролей необходимо построить варианты использования. Варианты использования изображены на рисунке 1.4.

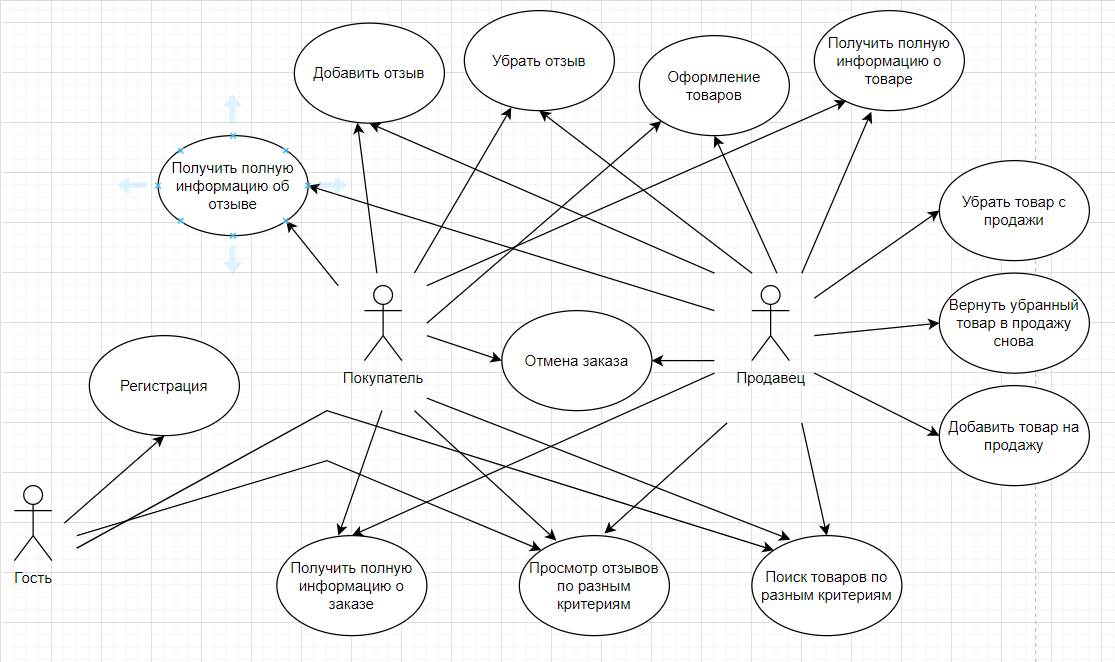


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступен только просмотр информации и самих товаров без возможности их покупки. После регистрации пользователь становится зарегистрированным покупателем.

Роль Покупатель получает возможность осуществления поиска товаров по различным критериям, покупке, отмене заказов, а также к написанию и удалению отзывов о товаре.

Роль Продавец будет иметь те же возможности, но кроме этого сможет добавлять новые товары на площадку, а также удалять или возвращать обратно на продажу уже существующие товары.

## 1.4 Вывод

Проведен аналитический обзор аналогов виртуальных маркетплейсов, существующих на рынке, для определения основных характеристик и функциональных возможностей, необходимых в разрабатываемой базе данных. Этот обзор помог выявить требования к базе данных и роли пользователей, а также варианты использования приложения в зависимости от этих ролей.

На основе анализа были выделены функциональные требования базы данных. Это включает в себя хранение информации о товарах, их описаниях, ценах, наличии на складе, а также информацию о пользователях, заказах и их статусах.

Для удобства проектирования и понимания системы была разработана UML-диаграмма, на которой отображены основные функции, доступные для каждой из ролей пользователей. Например, гость имеет доступ лишь к просмотру товаров и отзывов, а также возможности регистрации, в то время как обычные пользователи могут осуществлять поиск, просмотр информации о продуктах и совершать заказы.

Этот аналитический подход позволил учесть разнообразные потребности пользователей и создать базу данных, которая будет эффективно поддерживать функционирование интернет-магазина, обеспечивая удобство использования и надежность операций.

1. Анализ и проектирование архитектуры проекта.

## 2.1 UML схема базы данных

Схема базы данных представляет собой логическую структуру всей реляционной базы данных или ее части. Она может существовать как в виде графического отображения, так и в виде набора формул, определяющих ее структуру. Эти формулы выражаются с помощью языков описания данных, таких как SQL. Как часть словаря данных, схема иллюстрирует взаимосвязи между сущностями базы данных, включая таблицы, представления, хранимые процедуры и другие элементы. Схема базы данных представлена на рисунке 2.1.

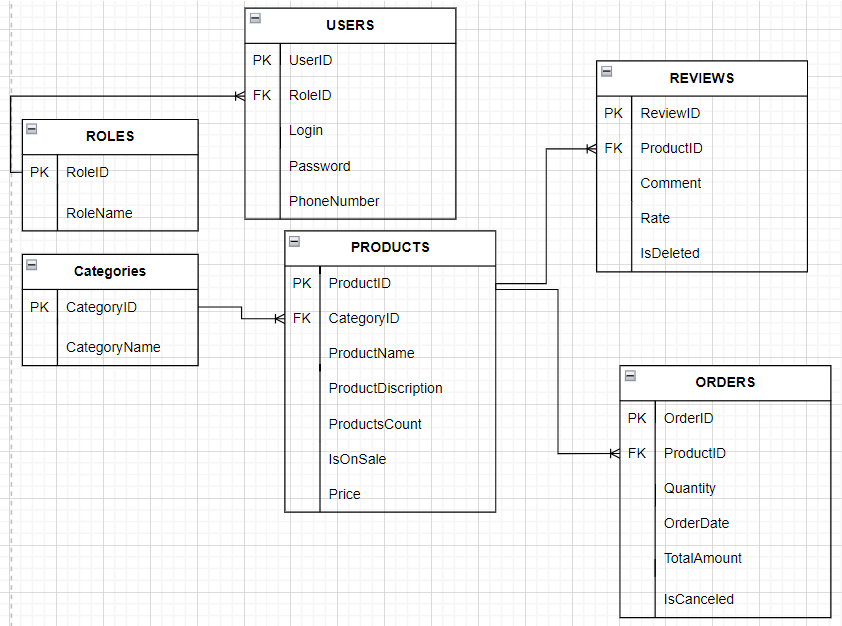


Рисунок 2.1 – Схема базы данных

Таким образом, схема показывает связи между таблицами и полями, а также типы отношений между ними, такие как связи "один-ко-многим", которые считаются самыми распространёнными среди всех. Например, таблица PRODUCTS связана с таблицами ORDERS и REVIEWS через внешние ключи ProductID. В этой базе данных используется лишь один вид связи, так как он является самым оптимальным.

## 2.2 Описание информационных объектов и ограничений целостности

Для реализации базы данных было разработано 6 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: ROLES, USERS, REVIEWS, PRODUCTS, CATEGORIES и ORDERS. Ниже приведено описание подробное описание каждой таблицы, включающее типы данных столбцов и ограничения целостности. Описание представлено в таблице 2.2.

Таблица 2.2 − Описание ограничений целостности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ROLES | RoleID | PK | Идентификатор роли | INTEGER |
| RoleName |  | Название роли | CHAR(15) |
| USERS | UserID | PK | Идентификатор пользователя | INTEGER |
| RoleID | FK | Идентификатор роли | INTEGER |
| Login |  | Логин пользователя | VARCHAR(35) |
| Password |  | Пароль пользователя | CHAR(16) |
| PhoneNumber |  | Телефонный номер пользователя | CHAR(20) |
| CATEGORIES | CategoryID | PK | Идентификатор категории | INTEGER |
| CategoryName |  | Название категории | CHAR(25) |
| PRODUCTS | ProductID | PK | Идентификатор товара | INTEGER |
| CategoryID | FK | Название товара | INTEGER |
| ProductName |  | Стоимость товара | VARCHAR(40) |
| ProductDistription |  | Описание товара | VARCHAR(300) |
| ProductCount |  | Количество товаров | INT |
| IsOnSale |  | В продаже или нет | BIT |
| Price |  | Цена товара | MONEY |
| REVIEWS | ReviewID | PK | Идентификатор отзыва | INTEGER |
| ProductID | FL | Идентификатор продукта | INTEGER |
| Comment |  | Комментарий пользователя | VARCHAR(500) |
| Rate |  | Оценка от 1 до 5 | INTEGER |
| IsDeleted |  | Удалён ли комментарий | BIT |
| ORDERS | OrderID | PK | Идентификатор заказа | INTEGER |
| ProductID | FK | Идентификатор товара | INTEGER |
| Quantity |  | Количество товаров | INTEGER |
| OrderDate |  | Дата заказа товаров | DATETIME |
| TotalAmount |  | Итоговая стоимость | MONEY |
| IsCanceled |  | Отменен ли товар или нет | BIT |

Таким образом, была определена структура таблиц и их ограничений.

## 2.3 Вывод

Анализ и проектирование модели базы данных являются критически важными этапами разработки информационной системы, закладывающими фундамент, от которого зависят эффективность, масштабируемость и надежность всей системы. На данном этапе была разработана UML-схема базы данных, которая наглядно демонстрирует структуру и взаимосвязи её компонентов, таких как таблицы, поля и связи между ними. Эта схема включает в себя таблицы, поля и связи между ними, что наглядно демонстрирует взаимодействие таблиц, типы данных в полях и отношения между сущностями. Схема содержит все основные компоненты базы данных, что упрощает понимание её структуры и способствует дальнейшей разработке и поддержке системы.

Структура базы данных состоит из шести таблиц, каждая из которых выполняет свою функцию и содержит определенные данные. Таблицы включают в себя: ROLES (содержит информацию о ролях пользователей в системе), USERS (хранит данные о пользователях системы), CATEGORIES (содержит категории товаров), PRODUCTS (хранит информацию о товарах), REVIEWS (содержит отзывы о товарах) и ORDERS (хранит данные о заказах). Таблицы связаны друг с другом через внешние ключи, что обеспечивает целостность данных и позволяет эффективно управлять информацией.

Разработанная UML-схема базы данных и структура таблиц обеспечивают четкую организацию данных и поддерживают целостность и управляемость информацией, что является основой для создания надежной и масштабируемой системы. Внимательное проектирование и анализ позволяют предусмотреть возможные сложности и минимизировать риски, связанные с разработкой и эксплуатацией базы данных.

# 3 Проектирование и разработка базы данных

## 3.1 Табличные пространства базы данных

Для виртуальной торговой площадки потребуется создать базу данных с файловой группой.

Файловая группа PRIMARY необходима для хранения данных таблиц, процедур, функций, триггеров, представлений и индексов. Пример скрипта создания табличного файловой группы представлена в листинге 3.1.

|  |
| --- |
| create database MarketPlace  ON  PRIMARY  (  NAME = MarketPlace\_PM,  FILENAME = 'D:\учёба\2 курс 4-й сем\Курсач\MarketPlace\_PM.mdf',  SIZE = 10MB,  MAXSIZE = UNLIMITED,  FILEGROWTH = 10%  )  LOG ON  (  NAME = MarketPlace\_LOG,  FILENAME = 'D:\учёба\2 курс 4-й сем\Курсач\MarketPlace\_LOG.ldf',  SIZE = 50MB,  MAXSIZE = 100MB,  FILEGROWTH = 5MB  ); |

Листинг 3.1 – Файловая группа

Таким образом, разработчик не имеет лимита на размещение данных в созданном табличном пространстве.

## 3.2 Создание основных ролей и пользователей базы данных

Для проекта было выделено несколько основных ролей, участвующих в разработке базы данных для интернет-магазина видеоигр с ролями и привилегиями для пользователей и ограничениями доступа к базе данных.

В нем определены такие пользователи, как Seller и Customer, каждый из которых имеет свои привилегии.

Seller получает доступ практически ко всем процедурам, за исключением системных, так как он может выполнять процедуры как обычного покупателя, так и продавца, в то время как Customer получает доступ лишь к своим процедурам. Пример роли покупателя показана в листинге 3.1.

|  |
| --- |
| CREATE ROLE Customer;  ALTER ROLE Customer ADD MEMBER CustomerUser  GRANT EXECUTE ON GetRatesByProductIDAndSortingPROC TO Customer;  GRANT EXECUTE ON GetFullReviewPROC TO Customer;  GRANT EXECUTE ON DeleteReview TO Customer;  GRANT EXECUTE ON AddReview TO Customer;  GRANT EXECUTE ON AddOrder TO Customer;  GRANT EXECUTE ON CancelOrder TO Customer;  GRANT EXECUTE ON GetFullOrderPROC TO Customer;  GRANT EXECUTE ON GetProductByCategoryNamePROC TO Customer;  GRANT EXECUTE ON GetProductsByNamePROC TO Customer;  GRANT EXECUTE ON GetProductsByNameAndPriceBetweenPROC TO Customer;  GRANT EXECUTE ON GetProductsByNameAndPriceHigherPROC TO Customer;  GRANT EXECUTE ON GetProductsByNameAndPriceLowerPROC TO Customer;  DENY EXECUTE ON AddProduct TO Customer;  DENY EXECUTE ON RemoveProductFromSale TO Customer;  DENY EXECUTE ON ReturnProductForSale TO Customer; |

Листинг 3.1 – Роль Customer

Seller имеет доступ как к процедурам покупателя, так и возможность добавлять свои товары, а также изменять их состояние. Пример создания роли продавца отображен на листинге 3.2.

|  |
| --- |
| CREATE ROLE Seller  ALTER ROLE Seller ADD MEMBER SellerUser  GRANT EXECUTE ON AddProduct TO Seller;  GRANT EXECUTE ON RemoveProductFromSale TO Seller;  GRANT EXECUTE ON ReturnProductForSale TO Seller;  GRANT EXECUTE ON GetRatesByProductIDAndSortingPROC TO Seller;  GRANT EXECUTE ON GetFullReviewPROC TO Seller;  GRANT EXECUTE ON DeleteReview TO Seller;  GRANT EXECUTE ON AddReview TO Seller;  GRANT EXECUTE ON AddOrder TO Seller;  GRANT EXECUTE ON CancelOrder TO Seller;  GRANT EXECUTE ON GetFullOrderPROC TO Seller;  GRANT EXECUTE ON GetProductByCategoryNamePROC TO Seller;  GRANT EXECUTE ON GetProductsByNamePROC TO Seller;  GRANT EXECUTE ON GetProductsByNameAndPriceBetweenPROC TO Seller;  GRANT EXECUTE ON GetProductsByNameAndPriceHigherPROC TO Seller;  GRANT EXECUTE ON GetProductsByNameAndPriceLowerPROC TO Seller; |

Листинг 3.2 – Роль Seller

В целом, была создана безопасная и эффективная база данных для интернет-магазина, где различные пользователи имеют различные уровни доступа и привилегии в соответствии с их ролями и функциями.

## 3.3 Вывод

Проект базы данных для маркетплейса включает создание файловой группы, предназначенной для хранения информации о пользователях, продуктах и т.д.

В рамках создания базы данных также определены роли и пользователи. Основные роли Seller и Customer имеют различные роли с различными привилегиями и выполняют собственные задачи.

В целом, создана безопасная и эффективная база данных, в которой различные пользователи имеют различные уровни доступа и привилегии в соответствии с их ролями и функциями.

4 Разработка необходимых объектов

## 4.1 Создание таблиц

В рамках проекта необходимо создать 6 таблиц, в которых будут храниться данные о пользователях, товарах и т. д.

Для интернет маркетплейса были созданы следующие таблицы.

ROLES – таблица, содержащая роли пользователей. Здесь хранится информация о ролях, которые могут быть присвоены пользователям. Каждая роль имеет уникальный идентификатор (RoleID) и название (RoleName).

USERS – это таблица, где хранятся данные о пользователях. Здесь содержится информация, такая как идентификатор пользователя (UserID), его имя (Login), номер телефона (PhoneNumber), пароля (Password). Также в таблице присутствует связь с таблицей ROLES через поле USER\_RoleID, определяющее роль пользователя.

Categories – таблица, предназначенная для хранения категорий товаров. Здесь содержатся различные категории, каждая из которых имеет название (CategoryName) и уникальный идентификатор.

PRODUCTS – таблица, содержащая информацию о товарах. Каждый товар имеет уникальный идентификатор (ProductID), название (ProductName), цену (PRICE) и прочую информацию.

REVIEWS – таблица, хранящая в себе всю информаци об отзывах. У каждого отзыва есть идентификатор (ReviewID), комментарий (Comment) и оценка (Rate).

ORDERS в свою очередь хранит информацию о заказах, также имеет свой идентификатор (OrderID), и такие данные как: дата заказа (OrderDate), количество (Quantity) и прочую информацию.

Связь между таблицами PRODUCTS, CATEGORIES, ORDERS и REVIEWS реализована через столбцы ProductID, позволяющие организовать связь "один ко многим".

При создании таблицы указывается ее название, перечисляются поля, ограничения целостности, а также табличное пространство, в котором будут храниться данные таблицы. Пример создания таблицы будет на листинге 4.1.

|  |
| --- |
| create table PRODUCTS  (  ProductID int identity(1,1) constraint Product\_ID\_PK primary key,  CategoryID int constraint CategoryID\_FK foreign key references CATEGORIES(CategoryID),  ProductName varchar(40),  ProductDiscription varchar(300),  ProductsCount integer,  IsOnSale bit default 1,  Price money  ) |

Листинг 4.1 – Создание таблицы PRODUCTS

Таблицы являются главной частью базы данных. Все последующие созданные объекты будут использовать данные базовых таблиц.

## 4.2 Создание триггеров

Триггер (trigger) в базе данных представляет собой особый вид процедур, которые срабатывают по запускающему их событию. В рамках данного проекта использовано два триггера для реагирования на некоторые, например, при вставке данных в таблице ORDERS: ChangeProductsQuantity и ExportJson. Каждый триггер является after-триггером, срабатывающим при вставке и рассматривает каждую измененную строку по отдельности. Пример создания триггера ChangeProductsQuantity, который изменяет количество товаров на складе после оформления заказа, представлен в листинге 4.2.

|  |
| --- |
| CREATE TRIGGER ChangeProductsQuantity  ON ORDERS  AFTER INSERT  AS  BEGIN  DECLARE @QuantityTaken INT = (Select Quantity from inserted)  DECLARE @QuantityWithoutTaken INT = (Select ProductsCount from ProductDetails where ProductID = (select ProductID from inserted))  Update ProductDetails  SET ProductsCount = @QuantityWithoutTaken - @QuantityTaken  END |

Листинг 4.2 – Создание триггера ChangeProductsQuantity

Использование триггеров сделало возможным создание срабатыванием на определенные действия, что стало неплохим решением сохранения целостности данных.

## 4.3 Создание индексов

Индекс представляет собой объект базы данных, который используется для ускорения поиска данных. Последовательный поиск данных может занимать много времени при большом количестве записей в таблице. 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, так как они имеют оптимизированную структуру для поиска.

Для того, чтобы быстро находить данные по их названию или описанию, были созданы индексы:

Индексы Index\_ProductByName, Index\_ProductByШВ, Index\_ProductByCategory, Index\_ProductByPrice, Index\_ProductByIsOnSale, Index\_OrderByID, Index\_OrderByIsCanceled, Index\_CategoryByID, Index\_CategoryByName, Index\_ReviewByID, Index\_ReviewByRate, ndex\_ReviewByProductID используются оптимизации поиска по названию объектов в соответствующих таблицах.

Пример создания индекса приведен в листинге 4.3.

|  |
| --- |
| CREATE NONCLUSTERED INDEX Index\_CategoryByName  ON CATEGORIES (CategoryName); |

Листинг 4.3 – Индекс Index\_CategoryName

Использование индексов ускорило работу некоторых запросов к таблицам, что позитивно сказалось на опыте работы с базой данных.

## 4.4 Создание представлений

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. Представления дают возможность пользователям просматривать содержимое таблиц без непосредственного обращения к таблицам, а также позволяют скрыть некоторые конфиденциальные данные, либо, наоборот, выдать больше данных, чем позволяют исходные таблицы.

В данном проекте было решено создать представления для каждой из 6 таблиц. Данное решение позволит пользователям не ограничиваться получением данных только лишь из процедур или функций, но и даст им возможность использовать собственные select-запросы к представлениям. Названия всех представлений содержат слово "Details", а также название исходной таблицы. Ниже приведен список некоторых представлений:

* RolesDetails содержит информацию о всех ролях.
* UsersDetails содержит информацию о всех пользователях.
* ProductsDetails содержит информацию о товарах.
* CategoriesDetails содержит информацию о категориях.
* OrdersDetails содержит информацию связанную с заказами.
* ReviewsDetails – содержит информацию об отзывах.

Пример представления показан в листинге 4.4.

|  |
| --- |
| CREATE VIEW ProductDetails AS  SELECT ProductID, ProductName, CategoryID, ProductsCount, Price, IsOnSale, ProductDiscription  FROM PRODUCTS; USERS  ; |

Листинг 4.4 – Представление ProductsDetails

Использование представлений позволило скрыть от пользователей внутреннюю структуру таблиц в базе данных, что повысило ее безопасность.

## 4.5 Создание процедур

Процедура в T-SQL – это модуль подпрограммы, состоящий из группы операторов T-SQL, которые можно вызывать по имени. Каждая процедура в T-SQL имеет собственное уникальное имя, по которому к ней можно обращаться и вызывать. Этот модуль подпрограммы в Oracle хранится как объект базы данных. Пример процедуры представлен в листинге 4.5.

|  |
| --- |
| CREATE PROCEDURE AddProduct  @ProductName VARCHAR(40),  @Quantity INT,  @CategoryName CHAR(25),  @ProductDescription VARCHAR(300),  @Price MONEY  AS  BEGIN  IF (TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL  AND TRY\_CAST(@Quantity AS INT) IS NOT NULL  AND TRY\_CAST(@CategoryName AS CHAR(25)) IS NOT NULL  AND TRY\_CAST(@ProductDescription AS VARCHAR(300)) IS NOT NULL  AND TRY\_CAST(@Price AS MONEY) IS NOT NULL)  BEGIN  IF @Quantity > 0  BEGIN  IF NOT EXISTS (SELECT 1 FROM ProductDetails WITH (INDEX(Index\_ProductByName)) WHERE ProductName = @ProductName)  BEGIN  DECLARE @CategoryID INT;  SET @CategoryID = (SELECT CategoryID FROM CategoryDetails WITH (INDEX(Index\_CategoryByName)) WHERE CategoryName = @CategoryName);    IF @CategoryID IS NOT NULL  BEGIN  INSERT INTO PRODUCTS (ProductName, ProductsCount, CategoryID, ProductDiscription, Price)  VALUES (@ProductName, @Quantity, @CategoryID, @ProductDescription, @Price);  PRINT 'Товар был добавлен успешно. ID вашего товара: ' + CAST(SCOPE\_IDENTITY() AS VARCHAR(10));  END  ELSE  BEGIN  PRINT 'Такой категории не существует.';  END  END  ELSE  BEGIN  PRINT 'Такой же товар уже существует.';  END  END  ELSE  BEGIN  PRINT 'Количество товаров не может быть отрицательным или равняться нулю.';  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.';  END  END; |

Листинг 4.5 – Процедура AddProduct

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.

## 4.6 Создание функций

Функция в T-SQL – это модуль подпрограммы, состоящий из группы операторов T-SQL, которые можно вызывать по имени. В отличие от процедур, функции должны заканчиваться оператором RETURN, возвращая вычисленное значение к точке вызова функции. Каждая функция в T-SQL имеет собственное уникальное имя, по которому к ней можно обращаться и вызывать. Этот модуль подпрограммы в SQL хранится как объект базы данных. Пример функции вычисления полной стоимости заказа приведен в листинге 4.6.

|  |
| --- |
| CREATE FUNCTION totalAmount  (  @ProductName VARCHAR(40),  @Quantity INT  )  RETURNS MONEY  AS  BEGIN  DECLARE @Price MONEY;  DECLARE @Result INT;  SET @Price = (select Price from ProductDetails  with (INDEX(Index\_ProductSearch))  where ProductName = @ProductName);  SET @Result = @Price \* @Quantity;  RETURN @Result;  END; |

Листинг 4.6 – Функция totalAmount

Данная функция используется в процедуре AddOrder для подсчета итоговой стоимости заказа, результат продолжает использоваться в процедуре.

## 4.7 Вывод

В рамках данного раздела базы данных были разработаны и созданы различные объекты, предназначенные для эффективного хранения, обработки и обеспечения безопасности данных интернет торговой площадки. Среди ключевых элементов используются таблицы для хранения различных типов данных, таких как информация о продуктах, заказах, пользователях и других аспектах бизнеса.

Для обеспечения целостности данных были созданы триггеры, которые активируются при вставке в определенных таблицах.

Для обеспечения безопасного доступа пользователей к данным были разработаны представления, которые предоставляют удобный интерфейс для работы с данными, фильтруя или скрывая конфиденциальную информацию в зависимости от прав доступа пользователя.

Для управления изменениями данных, вносимыми пользователями, были разработаны хранимые процедуры и функции. Они предоставляют удобные интерфейсы для выполнения операций над данными, таких как добавление новых продуктов, обновление информации о заказах и управление аккаунтами пользователей, не требуя напрямую взаимодействовать с исходными таблицами базы данных.

Для организации и группировки логически связанных объектов были созданы пакеты. Они позволяют упорядочить и сгруппировать хранимые процедуры, функции и другие элементы базы данных под единым названием, что облегчает их использование и поддержку.

В целом, эти объекты и механизмы созданы для обеспечения эффективного управления данными интернет-магазина, обеспечивая их целостность, безопасность и удобство использования для пользователей.

5 Описание процедур экспорта и импорта

## 5.1 Экспорт данных в JSON

Для экспорта данных в формат JSON был разработан триггер. Он срабатывает когда в таблицу ORDERS вносятся новые заказы, сразу же после этого происходит перевод записи в JSON вид и в дальнейшем через встроенную процедуру вызова утилиты происходит запись в соотвествующий файл. Код данного триггера представлен в листинге 5.1.

|  |
| --- |
| CREATE TRIGGER ExportJson  ON ORDERS  AFTER INSERT  AS  BEGIN  DECLARE @JsonData NVARCHAR(MAX)  SET @JsonData = (Select \* from inserted FOR JSON AUTO)    DECLARE @Command NVARCHAR(MAX)  DECLARE @FilePath NVARCHAR(MAX) = 'D:\учёба\2 курс 4-й сем\Курсач\OrdersData.json'  SET @Command = 'echo ' + @JsonData + ' > "' + @FilePath + '"';  EXEC xp\_cmdshell @Command;  END |

Листинг 5.1 – Триггер ExportJson

## 5.2 Импорт данных из JSON

Для импорта JSON-данных в базу данных была создана процедура, использующая встроенные SQL типы данных и функции для работы с форматом JSON. Среди таких типов и функций можно выделить OPENJSON() и OPENWORKSET(). Для импортирования полученных данных в таблицы используется конструкция INSERT INTO, позволяющая обновить существующие или добавить новые строки в таблицу. Пример процедуры GetImportDataJson, позволяющей получить данные из файла в таблицу ImportedDataJson, представлен в листинге 5.3.

|  |
| --- |
| CREATE PROCEDURE GetImportDataJson  AS  BEGIN  DECLARE @Json NVARCHAR(MAX);  SELECT @Json = BulkColumn  FROM OPENROWSET (BULK 'D:\учёба\2 курс 4-й сем\Курсач\OrdersData.json', SINGLE\_CLOB) as j;  INSERT INTO ImportedDataJson (OrderID, ProductID, OrderDate, TotalAmount, IsCanceled)  SELECT OrderID, ProductID, OrderDate, TotalAmount, IsCanceled  FROM OPENJSON(@Json)  WITH (  OrderId int,  ProductID int,  OrderDate datetime,  TotalAmount money,  IsCanceled bit  );  Select \* FROM ImportedDataJson  END; |

Листинг 5.3 – Процедура GetImportDataJson

## 5.3 Вывод

В рамках данного раздела были созданы триггеры и процедуры, позволяющие экспортировать и импортировать данные из таблиц в формат JSON. Подобная работа с данными может использоваться для резервного сохранения данных, их модификации вне базы данных и многих других сценариях. К тому же, выбор популярного, простого и широко используемого формата JSON позволяет использовать экспортированные из базы данные во многих современных приложениях.

6 Использование технологии репликации

Репликация в SQL — это технология, которая позволяет копировать и распространять данные и объекты базы данных с одной базы данных на другую и синхронизировать между ними изменения для обеспечения согласованности данных. Основные цели репликации включают в себя распределение нагрузки на несколько серверов, обеспечение высокой доступности данных, создание резервных копий и поддержку удаленных баз данных.

Виды репликации в SQL

1. Снимковая репликация (Snapshot Replication):

- Делает снимок данных на определенный момент времени и копирует его на целевой сервер.

- Используется для данных, которые изменяются редко или для начальной загрузки других видов репликации.

2. Транзакционная репликация (Transactional Replication):

- Постоянно отслеживает изменения данных и передает их на целевые серверы в реальном времени или почти в реальном времени.

- Подходит для систем, где важна высокая консистентность данных и минимальная задержка.

3. Репликация слияния (Merge Replication):

- Позволяет инициализировать и изменять данные на разных серверах независимо друг от друга, а затем синхронизировать изменения.

- Используется в случаях, когда необходима автономная работа различных узлов с последующим объединением данных.

В нашем проекте мы будем использовать репликацию слияния, так как она позволяет распределить нагрузку на несколько серверов а также увеличить надежность базы данных.

Основные компоненты репликации

1. Публикация (Publication):Набор данных или объектов базы данных, которые будут реплицироваться. Публикуется на сервере-издателе (Publisher).

2. Издатель (Publisher): Сервер базы данных, на котором создается публикация. Распространяет данные и изменения другим серверам.

3. Подписка (Subscription): Определяет, какие публикации и как они будут получены целевым сервером. Может быть "push" (данные отправляются издателем) или "pull" (данные запрашиваются подписчиком).

4. Подписчик (Subscriber): Сервер базы данных, который получает публикации и применяет их к своей базе данных. Может подписываться на несколько публикаций от различных издателей.

5. Дистрибьютор (Distributor): Сервер, который управляет хранением и распространением данных. Может быть отдельным сервером или выполнять функции издателя.

## 6.1 Настройка репликации

Настройка репликации требует некоторых особых настроек основной базы. Как было сказано ранее, в репликации разделяют дистрибьютер, подписку и издателя. Для начала нужно настроить дистрибьютер в файловой группе, которая в будущем будет издателем.

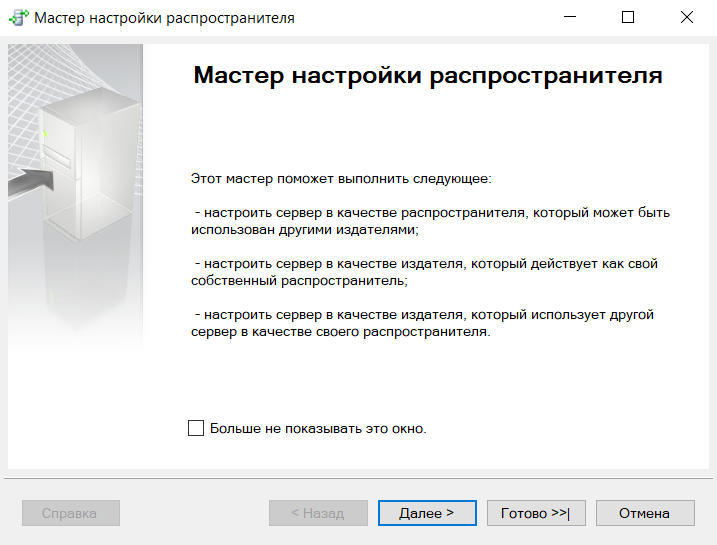


Рисунок 6.1 – Настройка дистрибьютера

В настройке нет ничего сложного, в результате мы смогли его настроить. Далее нужно настроить публикацию.

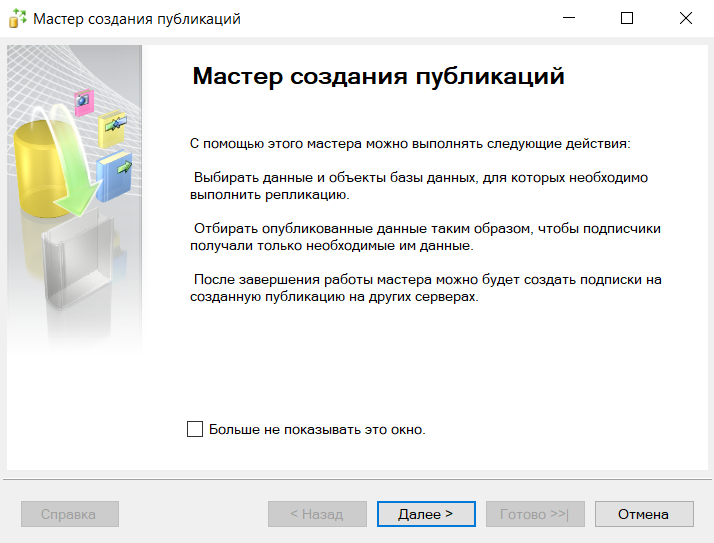


Рисунок 6.2 – Настройка публикации

Далее нужно указать сервер и файловую группу, откуда будет производиться снимок данных и выбираем тип репликации, в нашем случае это репликация слиянием. После этого выбираем объекты, которые мы будем передавать, в нашем случае это таблицы, выбираем все. После этого нужно настроить агент моментальных снимков, который будет фиксировать изменения данных в базе данных и после отправлять полученные снимки подписчикам.

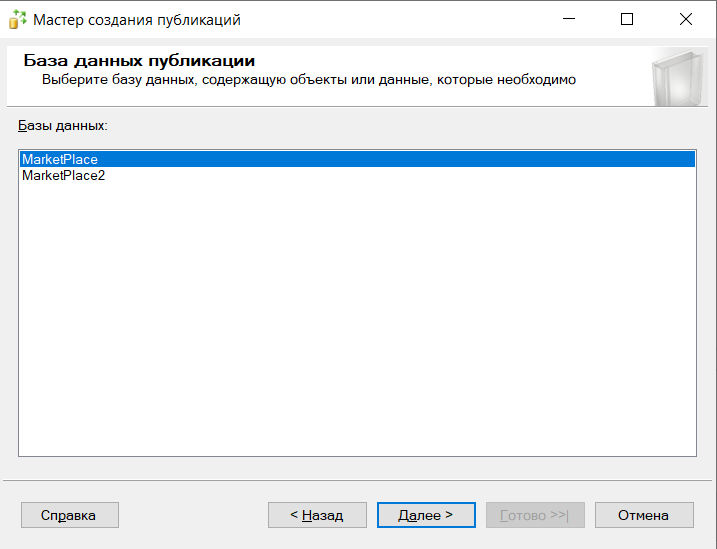


Рисунок 6.3 – Выбор базы данных

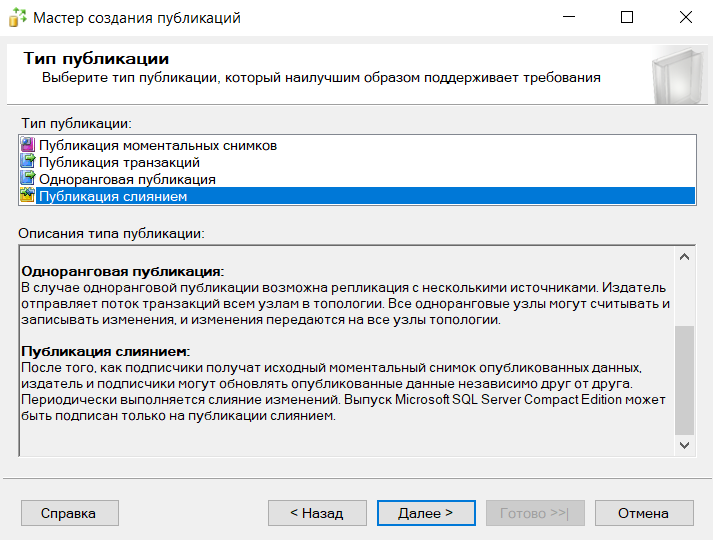


Рисунок 6.4 – Выбор типа публикации

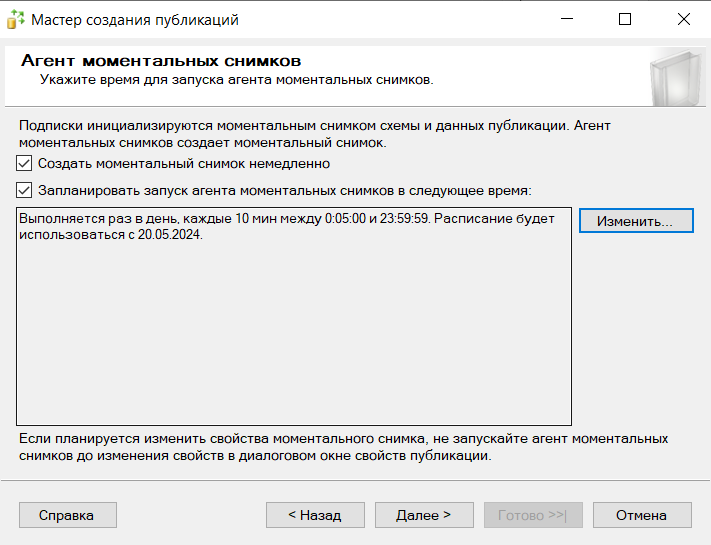


Рисунок 6.5 – Выбор плана частоты создания снимков

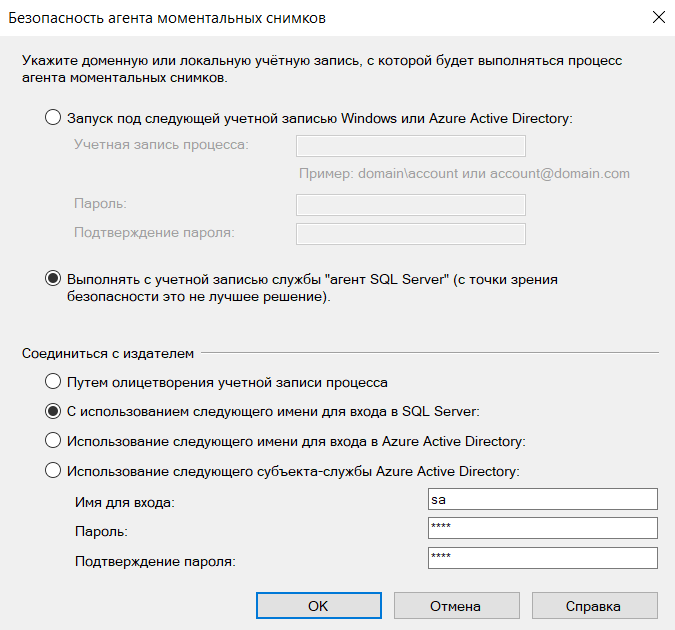


Рисунок 6.6 – Настройка агента моментальных снимков

После необходимо создать подписчика, там мы уже выбираем базу данных, на которую будут приходить необходимые данные, которые будут вступать сразу же в силу после получения. Также необходимо настроить агент слияния.

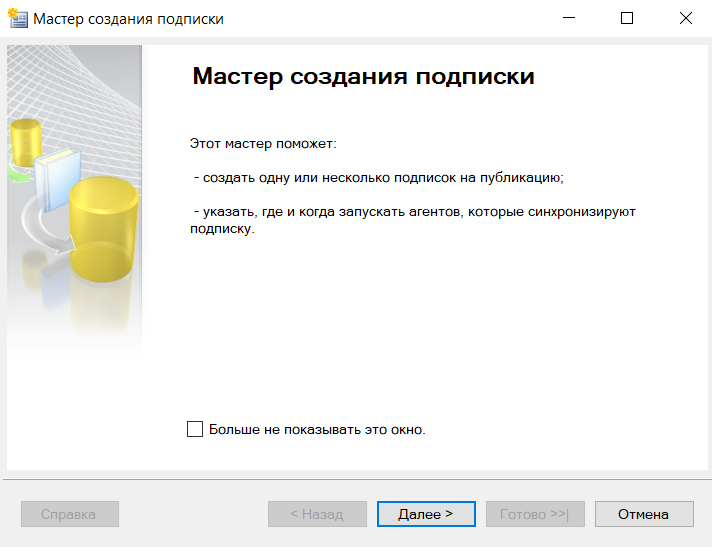


Рисунок 6.7 – Настройка подписки

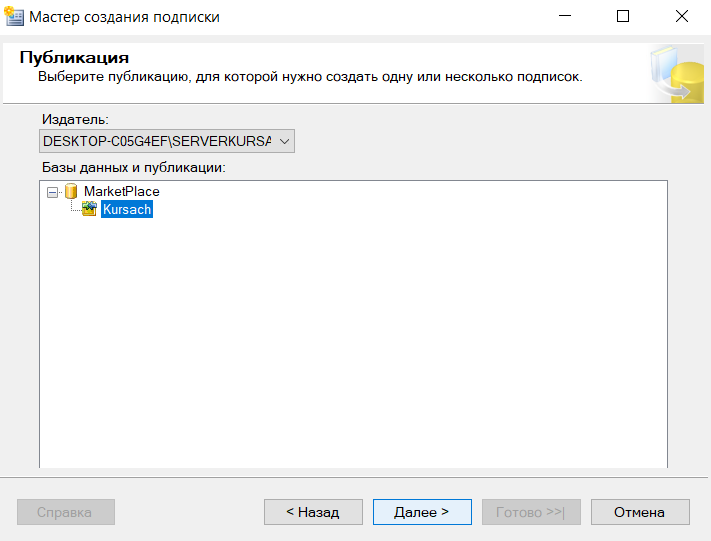


Рисунок 6.8 – Выбор сервера и базы данных для подписки

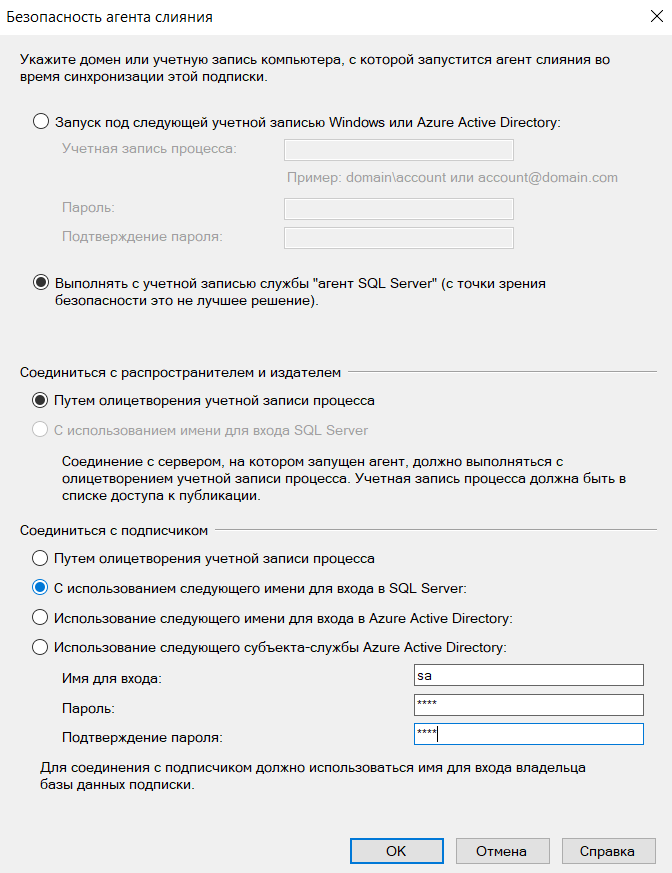


Рисунок 6.9 – Настройка агента слияния

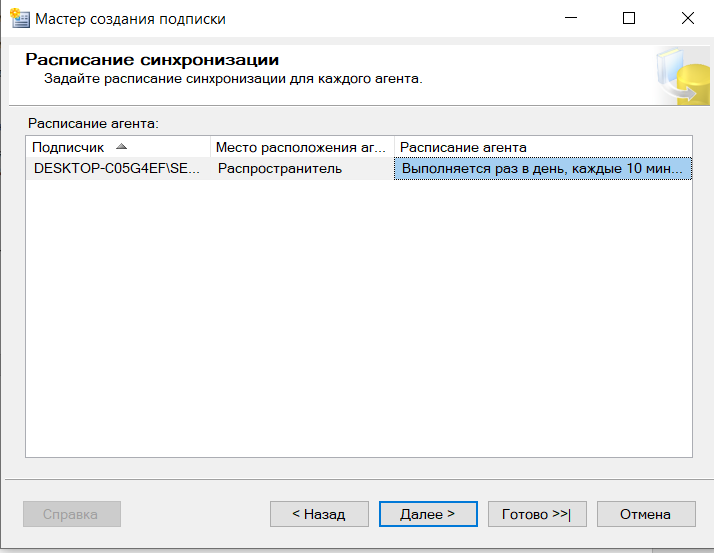


Рисунок 6.10 – Настройка расписания слияния

## 6.2 Проверка работы репликации

Для проверки работы репликации можно выполнить select-запрос к любой таблице на обоих базах данных. Результат запроса для основного сервера представлен на рисунке 6.11.

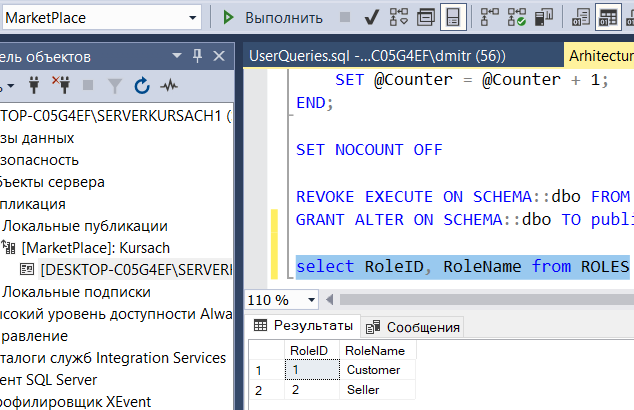


Рисунок 6.11 – Выполнение запроса на основном сервере

Результат выполнения аналогичного select-запроса на другом сервере представлен на рисунке 6.12.

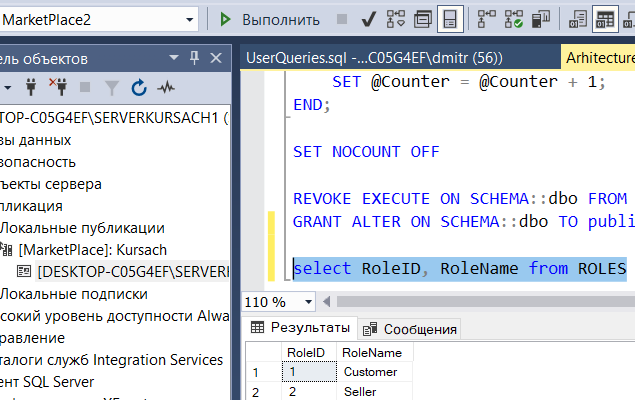


Рисунок 6.4 – Выполнение запроса на резервном сервере

Из этого можно сделать вывод, что настройка репликации прошла успешно.

## 6.3 Вывод

Репликация данных в SQL — это мощный инструмент для повышения производительности, доступности и отказоустойчивости баз данных. Выбор подходящего вида репликации и тщательное планирование её реализации позволяют эффективно использовать эту технологию в различных сценариях бизнеса.

7 Тестирование производительности базы данных

## 7.1 Тестирование производительности

Для тестирования производительности базы данных было решено заполнить таблицу PRODUCTS. Цикл while представленная в листинге 7.1, при выполнении добавит в таблицу PRODUCTS 100000 строк. Время выполнения данной процедуры представлено на рисунке 7.2.

|  |
| --- |
| DECLARE @Counter INT = 1;  WHILE @Counter <= 100000  BEGIN  INSERT INTO PRODUCTS (CategoryID, ProductName, ProductDiscription, ProductsCount, Price)  VALUES (1, 'TestProduct' + CAST(@Counter AS VARCHAR(MAX)), 'Description for Test Product', 100, CAST(@Counter AS MONEY));  SET @Counter = @Counter + 1;  END; |

Листинг 7.1 – Цикл while для заполнения 100000 строк в таблице PRODUCTS



Рисунок 7.1 – Время выполнения цикла

Результат выборки всех данных из таблицы представлен на рисунке 7.2.

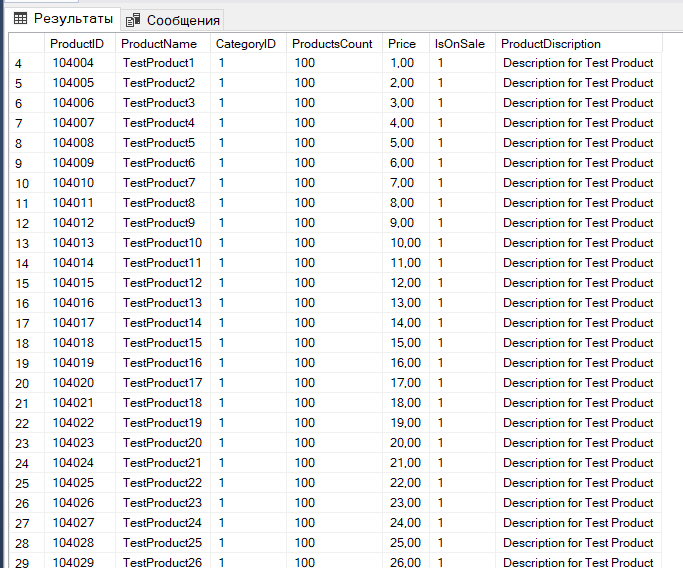


Рисунок 7.3 – Выборка всех данных из таблицы

Время выполнения запроса к таблице без использования дополнительных индексов представлено на рисунке 7.3.



Рисунок 7.4 – Время выполнения select-запроса

Время выполнение аналогичного запроса после создания индексов для столбцов данной таблицы представлено на рисунке 7.4.



Рисунок 7.4 – Время выполнения select-запроса

Как можно заметить, после создания индексов выполнение запросов заметно ускорилось, что говорит о важности индексов при работе с базой данных.

## 7.2 Вывод

В рамках данного раздела была протестирована производительность базы данных. Для этого в одну из таблиц было вставлено 100,000 записей и проведены замеры времени выполнения одинаковых запросов с использованием дополнительных индексов и без них. По результатам тестирования выяснилось следующее: база данных стабильно работает при большом количестве записей в ней и использование индексов полезно при работе с большими объемами данных.

Добавление индексов значительно сократило время выполнения запросов, что подтверждает их важность для оптимизации работы базы данных. Эти результаты демонстрируют необходимость и эффективность использования индексов для обеспечения высокой производительности системы при увеличении объема данных.

Таким образом, исследование показало, что индексы являются неотъемлемым инструментом для поддержания и улучшения производительности базы данных в условиях роста объема информации. Эти данные подчеркивают важность правильного проектирования структуры базы данных с учетом возможного увеличения объема данных, а также необходимость регулярного анализа и оптимизации производительности с использованием индексов.

Кроме того, результаты тестирования подчеркивают важность стратегического подхода к управлению базой данных. Это включает в себя не только создание и использование индексов, но и регулярное мониторинг производительности, своевременную оптимизацию запросов и обновление структуры базы данных в соответствии с растущими требованиями и объемами данных. Таким образом, правильное управление базой данных и использование индексов играют ключевую роль в обеспечении ее эффективной и стабильной работы в долгосрочной перспективе.

# Заключение

В результате работы была успешно реализована база данных для интернет торговой площадки, включающая управление информацией о пользователях, продуктах, категориях, заказах, оценках товаров и другими аспектами. Эта база данных охватывает все ключевые элементы, необходимые для полноценного функционирования интернет-магазина, обеспечивая надежное хранение и оперативное управление данными.

Были использованы различные возможности базы данных Microsoft SQL Server, начиная от простых операций вставки и удаления до сложных триггеров и процедур. Применение широкого спектра функций SQL позволило создать эффективную и гибкую систему управления данными. Были разработаны триггеры для автоматизации определённых задач и процедуры для выполнения сложных операций с данными.

Важным аспектом проекта стало обеспечение безопасности данных с помощью применения технологии репликации. Она обеспечила высокий уровень отказоустойчивости и непрерывность работы базы данных.

Была разработана иерархия процедур управления данными, позволяющая каждому пользователю получать доступ только к необходимым данным. Разработка иерархии прав доступа позволила точно распределить роли и обязанности между пользователями системы, обеспечивая их доступ только к тем данным, которые необходимы для выполнения их задач.

Проведено тестирование производительности базы данных на большом объеме данных, что позволило оптимизировать ее структуру и проанализировать планы запросов к таблицам. Тестирование на высоких нагрузках помогло выявить и устранить узкие места в структуре базы данных, что значительно повысило её производительность и эффективность обработки запросов.

Использование технологий базы данных в этом проекте значительно упростило и ускорило процессы управления данными магазина. Благодаря современным технологиям, интегрированным в проект, удалось существенно сократить время обработки данных и улучшить общую производительность системы, что в конечном итоге повысило удовлетворённость пользователей и эффективность работы интернет-магазина.

# Список используемых источников

1. Репликация SQL Server [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/sql/relational-databases/replication/sql-server-replication?view=sql-server-ver16. – Дата доступа: 05.03.2024.
2. Работа с JSON в SQL [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver16. – Дата доступа: 12.03.2024.
3. Руководство по архитектуре и разработке индексов в SQL Server [Электронный ресурс]. – Режим доступа:

https://learn.microsoft.com/ru-ru/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver16. – Дата доступа: 20.03.2024.

1. Определение триггеров [Электронный ресурс]. – Режим доступа: https://metanit.com/sql/sqlserver/12.1.php. – Дата доступа: 01.04.2024.

# Приложение А Листинг создания таблиц

|  |
| --- |
| create table USERS  (  UserID int identity(1,1) constraint UserID\_PK primary key,  Login varchar(35),  PhoneNumber varchar(20),  RoleID int constraint RoleID\_USERS\_FK foreign key references ROLES(RoleID),  Password char(16)  )  create table ROLES  (  RoleID int identity(1,1) constraint RoleID\_PK primary key,  RoleName varchar(MAX),  )  create table CATEGORIES  (  CategoryID int identity(1,1) constraint CategoryID\_PK primary key,  CategoryName char(25)  )  create table PRODUCTS  (  ProductID int identity(1,1) constraint Product\_ID\_PK primary key,  CategoryID int constraint CategoryID\_FK foreign key references CATEGORIES(CategoryID),  ProductName varchar(40),  ProductDiscription varchar(300),  ProductsCount integer,  IsOnSale bit default 1,  Price money  )  create table REVIEWS  (  ReviewID int identity(1,1) constraint ReviewID\_PK primary key,  ProductID int constraint ProdutID\_FK foreign key references PRODUCTS(ProductID),  Comment varchar(500),  Rate int constraint RateCheck check(Rate between 1 and 5),  IsDeleted BIT default 0  )  create table Orders  (  OrderId int identity(1,1) constraint OrderID\_PK primary key,  ProductID int constraint ProdutID\_ORDERS\_FK foreign key references PRODUCTS(ProductID),  Quantity INT,  OrderDate datetime,  TotalAmount money,  IsCanceled bit default 0  ) |

# Приложение Б Листинг создания триггеров

|  |
| --- |
| CREATE TRIGGER ExportJson  ON ORDERS  AFTER INSERT  AS  BEGIN  DECLARE @JsonData NVARCHAR(MAX)  SET @JsonData = (Select \* from inserted FOR JSON AUTO)    DECLARE @Command NVARCHAR(MAX)  DECLARE @FilePath NVARCHAR(MAX) = 'D:\учёба\2 курс 4-й сем\Курсач\OrdersData.json'  SET @Command = 'echo ' + @JsonData + ' > "' + @FilePath + '"';  EXEC xp\_cmdshell @Command;  END  CREATE TRIGGER ChangeProductsQuantity  ON ORDERS  AFTER INSERT  AS  BEGIN  DECLARE @QuantityTaken INT = (Select Quantity from inserted)  DECLARE @QuantityWithoutTaken INT = (Select ProductsCount from ProductDetails where ProductID = (select ProductID from inserted))  Update ProductDetails  SET ProductsCount = @QuantityWithoutTaken - @QuantityTaken  END |

# Приложение В Листинг процедур

|  |
| --- |
| CREATE PROCEDURE AddOrder  @ProductName VARCHAR(40),  @Quantity INT  AS  BEGIN  IF (TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL AND TRY\_CAST(@Quantity AS INT) IS NOT NULL AND @Quantity >= 0)  BEGIN  DECLARE @ProductID INT;  SET @ProductID = (SELECT ProductID FROM ProductDetails WITH (INDEX(Index\_ProductByName)) WHERE ProductName = @ProductName);  IF @ProductID IS NOT NULL  BEGIN  IF EXISTS (SELECT 1 FROM ProductDetails WITH (INDEX(Index\_ProductID)) WHERE ProductID = @ProductID)  BEGIN  IF (SELECT IsOnSale FROM ProductDetails WITH (INDEX(Index\_ProductByID)) WHERE ProductID = @ProductID) = 1  BEGIN  DECLARE @AvailableQuantity INT;  SET @AvailableQuantity = ISNULL((SELECT ProductsCount FROM ProductDetails WITH (INDEX(Index\_ProductByID)) WHERE ProductID = @ProductID), 0);  IF @AvailableQuantity > 0  BEGIN  DECLARE @TotalAmount MONEY;  DECLARE @OrderDate DATETIME;  SET @OrderDate = GETDATE();  SET @TotalAmount = dbo.totalAmount(@ProductName, @ProductID);  INSERT INTO ORDERS (OrderDate, ProductID, TotalAmount, Quantity) VALUES (@OrderDate, @ProductID, @TotalAmount, @Quantity);  DECLARE @OrderID INT;  SET @OrderID = SCOPE\_IDENTITY();  PRINT 'Заказ оформлен успешно. Номер вашего заказа: ' + CAST(@OrderID AS VARCHAR(10));  END  ELSE  BEGIN  PRINT 'Товаров нет на складе.';  END  END  ELSE  BEGIN  PRINT 'Товар не находится в продаже.';  END  END  ELSE  BEGIN  PRINT 'Такого товара не существует.';  END  END  ELSE  BEGIN  PRINT 'Такого товара не существует.';  END  END  ELSE  BEGIN  PRINT 'Один или оба параметра введены неверно.';  END  END;  CREATE PROCEDURE cancelOrder  @OrderID int  AS  BEGIN  IF TRY\_CAST(@OrderID AS INT) IS NOT NULL  BEGIN  IF EXISTS (Select 1 from ORDERS with (INDEX(Index\_OrderByID)) where OrderID = @OrderID AND IsCanceled = 0)  BEGIN  Update ORDERS  Set IsCanceled = 1 where OrderID = @OrderID;  PRINT 'Заказ был успешно отменен.'  END  ELSE  BEGIN  PRINT 'Такого заказа не существует или он уже был отменен.';  END  END  ELSE  BEGIN  PRINT 'Вы ввели неверное значение.'  END  END;  CREATE PROCEDURE GetFullOrderPROC  @OrderID INT  AS  BEGIN  IF(TRY\_CAST(@OrderID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from OrderDetails with (INDEX(Index\_OrderByID)) where OrderID = @OrderID)  BEGIN  Select \* from GetFullOrder(@OrderID)  PRINT 'Заказ был найден успешно.'  END  ELSE  BEGIN  PRINT 'Такого товара не существует.'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE AddProduct  @ProductName VARCHAR(40),  @Quantity INT,  @CategoryName CHAR(25),  @ProductDescription VARCHAR(300),  @Price MONEY  AS  BEGIN  IF (TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL  AND TRY\_CAST(@Quantity AS INT) IS NOT NULL  AND TRY\_CAST(@CategoryName AS CHAR(25)) IS NOT NULL  AND TRY\_CAST(@ProductDescription AS VARCHAR(300)) IS NOT NULL  AND TRY\_CAST(@Price AS MONEY) IS NOT NULL)  BEGIN  IF @Quantity > 0  BEGIN  IF NOT EXISTS (SELECT 1 FROM ProductDetails WITH (INDEX(Index\_ProductByName)) WHERE ProductName = @ProductName)  BEGIN  DECLARE @CategoryID INT;  SET @CategoryID = (SELECT CategoryID FROM CategoryDetails WITH (INDEX(Index\_CategoryByName)) WHERE CategoryName = @CategoryName);    IF @CategoryID IS NOT NULL  BEGIN  INSERT INTO PRODUCTS (ProductName, ProductsCount, CategoryID, ProductDiscription, Price)  VALUES (@ProductName, @Quantity, @CategoryID, @ProductDescription, @Price);  PRINT 'Товар был добавлен успешно. ID вашего товара: ' + CAST(SCOPE\_IDENTITY() AS VARCHAR(10));  END  ELSE  BEGIN  PRINT 'Такой категории не существует.';  END  END  ELSE  BEGIN  PRINT 'Такой же товар уже существует.';  END  END  ELSE  BEGIN  PRINT 'Количество товаров не может быть отрицательным или равняться нулю.';  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.';  END  END;  CREATE PROCEDURE RemoveProductFromSale  @ProductID INT  AS  BEGIN  IF(TRY\_CAST(@ProductID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ProductDetails with (INDEX(Index\_ProductByID)) where ProductID = @ProductID and IsOnSale = 0)  BEGIN  Update PRODUCTS  set IsOnSale = 1 where ProductID = @ProductID  PRINT 'Товар успешно снят с продажи.'  END  ELSE  BEGIN  PRINT 'Такого товара не существует или он уже был снят с продажи.'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE ReturnProductForSale  @ProductID INT  AS  BEGIN  IF(TRY\_CAST(@ProductID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ProductDetails with (INDEX(Index\_ProductByID)) where ProductID = @ProductID and IsOnSale = 1)  BEGIN  Update PRODUCTS  set IsOnSale = 1 where ProductID = @ProductID  PRINT 'Товар теперь снова в продаже.'  END  ELSE  BEGIN  PRINT 'Такого товара не существует или он уже находится в продаже.'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE GetFullProductPROC  @ProductID INT  AS  BEGIN  IF(TRY\_CAST(@ProductID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ProductDetails with (INDEX(Index\_ProductByID)) where ProductID = @ProductID)  BEGIN  Select \* from GetFullProduct(@ProductID);  PRINT 'Товар был найден успешно.'  END  ELSE  BEGIN  PRINT 'Такого товара не существует.'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE AddReview  @ProductID INT,  @Rate INT,  @Comment VARCHAR(500)  AS  BEGIN  IF (TRY\_CAST(@ProductID AS INT) IS NOT NULL AND TRY\_CAST(@Rate AS INT) IS NOT NULL AND TRY\_CAST(@Comment AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ProductDetails with (INDEX(Index\_ProductByID)) where ProductID = @ProductID)  BEGIN  IF @Rate < 1 or @Rate > 5  BEGIN  INSERT INTO REVIEWS(ProductID, Rate, Comment) values  (@ProductID, @Rate, @Comment)  PRINT 'Ваш отзыв был опубликован успешно. ID вашего отзыва: ' + SCOPE\_IDENTITY();  END  BEGIN  PRINT 'Отзыв не может иметь оценки меньше 1 или больше 5.'  END  END  ELSE  BEGIN  PRINT 'Такого товара не существует.'  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.'  END  END  CREATE PROCEDURE DeleteReview  @ReviewID INT  AS  BEGIN  IF (TRY\_CAST(@ReviewID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ReviewDetails with(INDEX(Index\_ReviewByID)) where ReviewID = @ReviewID and IsDeleted = 0)  BEGIN  Update REVIEWS  Set IsDeleted = 1 where ReviewID = @ReviewID  END  ELSE  BEGIN  PRINT 'Такого отзыва не существует или он уже был удален.'  END  END  ELSE  BEGIN  PRINT 'Параметр введён неверно.'  END  END  CREATE PROCEDURE GetFullReviewPROC  @ReviewID INT  AS  BEGIN  IF(TRY\_CAST(@ReviewID AS INT) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from ReviewDetails with (INDEX(Index\_ReviewByID)) where ProductID = @ReviewID)  BEGIN  Select \* from GetFullReview(@ReviewID);  PRINT 'Отзыв был найден успешно.'  END  ELSE  BEGIN  PRINT 'Такого отзыва не существует'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE GetProductByCategoryNamePROC  @CategoryName CHAR(25)  AS  BEGIN  IF(TRY\_CAST(@CategoryName AS CHAR(25)) IS NOT NULL)  BEGIN  IF EXISTS (Select 1 from CategoryDetails with (INDEX(Index\_CategoryByName)) where CategoryName = @CategoryName)  BEGIN  Select \* from GetProductsByCategory(@CategoryName)  PRINT 'Товары были успешно найдены.'  END  ELSE  BEGIN  PRINT 'Такой категории не существует.'  END  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE GetProductsByNamePROC  @ProductName VARCHAR(40)  AS  BEGIN  IF(TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL)  BEGIN  Select \* from GetProductsByName(@ProductName)  PRINT 'Поиск произошёл успешно.'  END  ELSE  BEGIN  PRINT 'Параметр введен неверно.'  END  END  CREATE PROCEDURE GetProductsByNameAndPriceBetweenPROC  @ProductName VARCHAR(40),  @MinPrice MONEY,  @MaxPrice MONEY  AS  BEGIN  IF(TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL AND  TRY\_CAST(@MinPrice AS MONEY) IS NOT NULL AND  TRY\_CAST(@MaxPrice AS MONEY) IS NOT NULL)  BEGIN  Select \* from GetProductsByNameAndPriceBetween(@ProductName, @MinPrice, @MaxPrice)  PRINT 'Поиск произошёл успешно.'  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.'  END  END  CREATE PROCEDURE GetProductsByNameAndPriceHigherPROC  @ProductName VARCHAR(40),  @MinPrice MONEY,  @SortingMode CHAR(5)  AS  BEGIN  IF(TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL  AND TRY\_CAST(@MinPrice AS MONEY) IS NOT NULL  AND TRY\_CAST(@SortingMode AS CHAR(5)) IS NOT NULL)  BEGIN  IF (@SortingMode = 'ASC' or @SortingMode = 'DESC')  BEGIN  Select \* from GetProductsByNameAndPriceHigher(@ProductName, @MinPrice, @SortingMode)  PRINT 'Поиск произошёл успешно.'  END  ELSE  BEGIN  PRINT 'Сортировка может принимать только значения "ASC" или "DESC"'  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.'  END  END  CREATE PROCEDURE GetProductsByNameAndPriceLowerPROC  @ProductName VARCHAR(40),  @MaxPrice MONEY,  @SortingMode CHAR(5)  AS  BEGIN  IF(TRY\_CAST(@ProductName AS VARCHAR(40)) IS NOT NULL  AND TRY\_CAST(@MaxPrice AS MONEY) IS NOT NULL  AND TRY\_CAST(@SortingMode AS CHAR(5)) IS NOT NULL)  BEGIN  IF (@SortingMode = 'ASC' or @SortingMode = 'DESC')  BEGIN  Select \* from GetProductsByNameAndPriceLower(@ProductName, @MaxPrice, @SortingMode)  PRINT 'Поиск произошёл успешно.'  END  ELSE  BEGIN  PRINT 'Сортировка может принимать только значения "ASC" или "DESC"'  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.'  END  END  CREATE PROCEDURE GetRatesByProductIDAndSortingPROC  @ProductID VARCHAR(40),  @SortingMode CHAR(5)  AS  BEGIN  IF(TRY\_CAST(@ProductID AS INT) IS NOT NULL AND TRY\_CAST(@SortingMode AS CHAR(5)) IS NOT NULL)  BEGIN  Select \* from GetRatesByProductIDAndSorting(@ProductID, @SortingMode)  PRINT 'Поиск произошел успешно.'  END  ELSE  BEGIN  PRINT 'Один или оба параметра введены неверно.'  END  END  CREATE PROCEDURE Registration  @UserName VARCHAR(25),  @PhoneNumber VARCHAR(20),  @Password CHAR(16),  @UserRole VARCHAR(MAX)  AS  BEGIN  IF(TRY\_CAST(@UserName AS VARCHAR(25)) IS NOT NULL  AND TRY\_CAST(@PhoneNumber AS CHAR(20)) IS NOT NULL  AND TRY\_CAST(@Password AS CHAR(16)) IS NOT NULL  AND TRY\_CAST(@UserRole AS CHAR(15)) IS NOT NULL)  BEGIN  IF @PhoneNumber LIKE '+375(\_\_)\_\_\_-\_\_-\_\_'  AND SUBSTRING(@PhoneNumber, 6, 2) LIKE '[0-9][0-9]'  AND SUBSTRING(@PhoneNumber, 9, 3) LIKE '[0-9][0-9][0-9]'  AND SUBSTRING(@PhoneNumber, 13, 2) LIKE '[0-9][0-9]'  AND SUBSTRING(@PhoneNumber, 16, 2) LIKE '[0-9][0-9]'  BEGIN  DECLARE @Role VARCHAR(MAX);  IF (@UserRole = 'Customer')  BEGIN  SET @Role = @UserRole  END  ELSE IF (@UserRole = 'Seller')  BEGIN  SET @Role = @UserRole  END  ELSE  BEGIN  PRINT 'Такой роли не существует.'  END    DECLARE @CurrentUser NVARCHAR(128)  SET @CurrentUser = SUSER\_NAME();  DECLARE @SQL NVARCHAR(400);  REVERT;  DECLARE @RoleID INT  SET @RoleID = (Select RoleID from RolesDetails where RoleName = @UserRole)    Insert into UsersDeails (Login, PhoneNumber, Password, RoleID) values (@UserName, @PhoneNumber, @Password, @RoleID)  PRINT 'Регистрация прошла успешно.'  END  ELSE  BEGIN  PRINT 'Формат телефонного номера неверен.'  END  END  ELSE  BEGIN  PRINT 'Один или несколько параметров введены неверно.'  END  END |

# Приложение Г Листинг функций

|  |
| --- |
| CREATE FUNCTION totalAmount  (  @ProductName VARCHAR(40),  @Quantity INT  )  RETURNS MONEY  AS  BEGIN  DECLARE @Price MONEY;  DECLARE @Result INT;  SET @Price = (select Price from ProductDetails  with (INDEX(Index\_ProductSearch))  where ProductName = @ProductName);  SET @Result = @Price \* @Quantity;  RETURN @Result;  END;  CREATE FUNCTION GetFullProduct  (  @ProductID INT  )  RETURNS TABLE  AS  RETURN(  Select \* from ProductDetails with(INDEX(Index\_ProductByID)) where ProductID = @ProductID  )  CREATE FUNCTION GetFullOrder  (  @OrderID INT  )  RETURNS TABLE  AS  RETURN(  Select OrderId, ProductName, OrderDate, TotalAmount, IsCanceled, Quantity from OrderDetails  inner join ProductDetails on ProductDetails.ProductID = OrderDetails.ProductID where OrderID = @OrderID  )  CREATE FUNCTION GetFullReview  (  @ReviewID INT  )  RETURNS TABLE  AS  RETURN(  Select ReviewID, ProductName, Comment, Rate from ReviewDetails  inner join ProductDetails on ProductDetails.ProductID = ReviewDetails.ProductID where ReviewID = @ReviewID  )  CREATE FUNCTION GetProductsByCategory  (  @CategoryName CHAR(25)  )  RETURNS TABLE  AS  RETURN(  Select ProductName, ProductDiscription, ProductsCount, IsOnSale, Price from ProductDetails  inner join CategoryDetails on CategoryDetails.CategoryID = ProductDetails.CategoryID where CategoryName = @CategoryName  )  CREATE FUNCTION GetProductsByName  (  @ProductName VARCHAR(40)  )  RETURNS TABLE  AS  RETURN(  Select ProductName, CategoryName, ProductDiscription, ProductsCount, IsOnSale, Price from ProductDetails  inner join CategoryDetails on CategoryDetails.CategoryID = ProductDetails.CategoryID  where ProductName LIKE '%' + @ProductName + '%'  )  CREATE FUNCTION GetProductsByNameAndPriceBetween  (  @ProductName VARCHAR(40),  @MinPrice MONEY,  @MaxPrice MONEY  )  RETURNS TABLE  AS  RETURN(  Select ProductName, CategoryName, ProductDiscription, ProductsCount, IsOnSale, Price from ProductDetails  inner join CategoryDetails on CategoryDetails.CategoryID = ProductDetails.CategoryID  where ProductName LIKE '%' + @ProductName + '%' and Price between @MinPrice and @MaxPrice  )  CREATE FUNCTION GetProductsByNameAndPriceHigher  (  @ProductName VARCHAR(40),  @MinPrice MONEY,  @SortingMode CHAR(5)  )  RETURNS TABLE  AS  RETURN(  Select top 100 percent ProductName, CategoryName, ProductDiscription, ProductsCount, IsOnSale, Price from ProductDetails  inner join CategoryDetails on CategoryDetails.CategoryID = ProductDetails.CategoryID  where ProductName LIKE '%' + @ProductName + '%' and Price > @MinPrice  order by  case  when @SortingMode = 'DESC' then Price  else NULL  end desc,  case  when @SortingMode = 'ASC' then Price  else NULL  end asc  )  CREATE FUNCTION GetProductsByNameAndPriceLower  (  @ProductName VARCHAR(40),  @MaxPrice MONEY,  @SortingMode CHAR(5)  )  RETURNS TABLE  AS  RETURN(  Select top 100 percent ProductName, CategoryName, ProductDiscription, ProductsCount, IsOnSale, Price from ProductDetails  inner join CategoryDetails on CategoryDetails.CategoryID = ProductDetails.CategoryID  where ProductName LIKE '%' + @ProductName + '%' and Price < @MaxPrice  order by  case  when @SortingMode = 'DESC' then Price  else NULL  end desc,  case  when @SortingMode = 'ASC' then Price  else NULL  end asc  )  CREATE FUNCTION GetRatesByProductIDAndSorting  (  @ProductID INT,  @SortingMode CHAR(5)  )  RETURNS TABLE  AS  RETURN(  Select Rate, ReviewID, ProductName, Comment from ReviewDetails  inner join ProductDetails on ProductDetails.ProductID = ReviewDetails.ProductID  where ReviewDetails.ProductID = @ProductID  order by  case  when @SortingMode = 'DESC' then Price  else NULL  end desc,  case  when @SortingMode = 'ASC' then Price  else NULL  end asc  ) |