

**Частное учреждение профессионального образования**

**«Высшая школа предпринимательства»**

**(ЧУПО «ВШП»)**

КУРСОВОЙ ПРОЕКТ

«Разработка базы данных для интернет-магазина одежды»

Выполнил:

студент 3-го курса специальности

09.02.07 «Информационные системы и программирование»

Айдарбеков Адахан Таалайбекович

подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

преподаватель дисциплины,

преподаватель ЧУПО «ВШП»,

к.ф.н. Ткачев П.С.

оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оглавление**

[Введение 2](#_Toc169609406)

[Глава 1. Анализ требований и проектирование логической модели базы данных 6](#_Toc169609407)

[1.1. Анализ требований к базе данных интернет-магазина одежды 6](#_Toc169609408)

[1.2. Анализ связей между сущностями 8](#_Toc169609409)

[1.3. Разработка схемы базы данных с использованием нотации Crow’s 11](#_Toc169609410)

[1.4. Администрирование базы данных 12](#_Toc169609411)

[Глава 2. Проектирование физической модели базы данных и реализация 14](#_Toc169609412)

[2.1. Проектирование физической модели базы данных 14](#_Toc169609413)

[2.2 Создание ролей и предоставление привилегий 21](#_Toc169609414)

[2.3. Типовые запросы 22](#_Toc169609415)

[2.4. Транзакции 24](#_Toc169609416)

[2.5. Триггеры 26](#_Toc169609417)

[2.6. Хранимые процедуры 27](#_Toc169609418)

[2.7. Локальные переменные 29](#_Toc169609419)

[2.8. Представления 29](#_Toc169609420)

[2.9. Условия 30](#_Toc169609421)

[2.10. Обработчики исключений 30](#_Toc169609422)

[2.11. Пользовательские функции 30](#_Toc169609423)

[Заключение 31](#_Toc169609424)

[Список источников 32](#_Toc169609425)

[Приложение 1. Условия 33](#_Toc169609426)

[Приложение 2. Пользовательские функции 33](#_Toc169609427)

[Приложениие 3. Антиплагиат 34](#_Toc169609428)

[Приложение 4. QR код для репозитория 34](#_Toc169609429)

# **Введение**

В рамках курсовой работы предлагается разработать базу данных для интернет-магазина одежды. Актуальность выбранной темы обусловлена ростом популярности онлайн-покупок и конкуренции на рынке, что требует создания эффективной системы для обработки и хранения данных. В настоящее время рынок интернет-торговли (бытовой техникой, одеждой и т.д.) является одним из наиболее быстрорастущих направлений. Покупатели все чаще предпочитают приобретать товары онлайн, так как это просто, удобно, экономит время и позволяет выбрать нужные вещи. В основной теме курсовой работы, в качестве выбранного направления для продажи рассматривается именно одежда. На самом деле под предлогом “одежда” имеются в виду абсолютно все товары, которые люди носят на себе (обувь, рюкзаки и т.д.). Именно эти товары являются одним из наиболее продаваемых и востребованных в онлайн-магазинах. Согласно статистике, объем продаж одежды через интернет ежегодно растет примерно до 25%.

Еще одним не менее важным аспектом, который явно показывает актуальность разработки базы данных для онлайн-магазинов, является рост конкуренции на онлайн-рынке. При высокой конкуренции, качество и скорость обслуживания клиентов, а также безопасность и надежность хранения их личных данных становятся ключевыми моментами, которые с высокой долей вероятности могут повлиять на успех бизнеса. Для успешной работы нашего магазина необходимо обеспечить высокую производительность и надежное хранение крупного объема данных, связанных с клиентами, товарами, заказами и другими сущностями (таблицами).

Также, база данных кроме обеспечения хранения и обработки данных, должна предоставлять возможности для анализа о покупательском поведении клиентов, чтобы выявлять тренды и предпочтения, формировать выгодные предложения и акции, которые в будущем поспособствуют увеличению продаж.

Кроме того, разработка базы данных для интернет-магазина одежды является важной задачей для бизнеса по следующим причинам:

1. Повышение эффективности работы магазина: благодаря возможности управления данными, сотрудники магазина могут быстро и точно обрабатывать заказы, контролировать наличие товаров на складе и вести учет продаж.

2. Улучшение качества обслуживания клиентов: с помощью базы данных можно хранить информацию о клиентах, их предпочтениях и в целом совершенных ими действиях, что позволяет найти индивидуальный подход для каждого покупателя, обслуживать и предлагать клиентам товары и услуги, соответствующие их требованиям.

3. Рост продаж: Об этом уже упоминалось выше, но все-таки стоит еще раз подчеркнуть этот пункт, так как для бизнеса это очень важно.

4. Минимизация задержек при работе: благодаря автоматизации многих бизнес-процессов, таких как учет запасов, обработка заказов, ведение отчетности, сокращается количество повторяемых операций и уменьшаются задержки, что удобно и для самих сотрудников.

В итоге, с уверенностью можно сказать, что разработка базы данных для интернет-магазина является не только актуальной, но и важной задачей для бизнеса, так как она способствует улучшению качества обслуживания клиентов, росту продаж и многому другому. [1]

**Цель:**

Целью данной работы является разработка базы данных для интернет-магазина одежды. Для достижения этой цели необходимо выполнить нижеперечисленные задачи.

**Задачи:**

1. Анализ требований к базе данных интернет-магазина одежды.

2. Проектирование логической модели базы данных.

3. Проектирование физической модели базы данных.

4. Реализация базы данных с использованием системы управления базами данных (СУБД).

5. Наполнение базы данных тестовыми данными.

Объектом исследования в данной работе является база данных для интернет-магазина одежды.

Предметом исследования является системный анализ и проектирование баз данных.

**Методы исследования**

В качестве основных методов исследования, использованных в данной работе, можно называть:

* Метод анализа - изучение требований к проектированию модели базы данных для интернет-магазина.
* Метод абстрагирования - создание схемы и принципа работы разрабатываемой базы данных.
* Метод структурных аналогий - создание базы данных при помощи общих частей, наблюдаемых в существующих базах данных.

# **Глава 1. Анализ требований и проектирование логической модели базы данных**

## **Анализ требований к базе данных интернет-магазина одежды**

При разработке базы данных в первую очередь необходимо провести анализ требований и спроектировать логическую модель базы данных. Этот процесс включает понимание того, какие задачи должна решать база данных и какие результаты ожидаются в конце разработки. Например, при создании базы данных для интернет-магазина одежды необходимо учитывать хранение информации о товарах. Эти товары будут представлены сущностями (таблицами) в базе данных, каждая из которых будет иметь свои атрибуты (столбцы), такие как название, описание, цена, размер и т.д. Помимо товаров, база данных будет содержать и другие сущности, такие как таблица "Пользователь" (клиенты), которые будут совершать покупки. Каждая из этих таблиц также будет иметь свои столбцы, данные которых будут храниться в базе данных. В ходе работы необходимо определить взаимосвязи между таблицами и их функционирование для удовлетворения потребностей бизнеса.

При проектировании базы данных необходимо учитывать возможность добавления новых данных или изменения существующих в будущем. Поэтому важно разработать гибкую и масштабируемую базу данных. Рассмотрим ситуацию, в которой планируется добавление новых данных: база данных должна быть способна адаптироваться к этим изменениям без затруднений. Кроме того, она должна обеспечивать высокую производительность даже при значительных нагрузках, связанных с ростом клиентской базы. Таким образом, гибкость и масштабируемость базы данных являются ключевыми аспектами её проектирования. [2], [3]

Следующим важным шагом является определение взаимодействия клиентов с базой данных. При наличии веб-приложения с профилями пользователей необходимо реализовать функциональность редактирования профилей и добавления новых данных. Также важно обеспечить удобство для клиентов, разработав фильтрацию и поиск по определённым критериям, а также пагинацию для отображения товаров на нескольких страницах.

Не менее важным аспектом является хранение и обработка мультимедийных файлов, таких как изображения, видео и аудиозаписи. Следует предусмотреть возможность хранения изображений товаров в высоком качестве, а также функции просмотра и увеличения этих изображений. Хранение метаданных о мультимедийных файлах, таких как название, размер, формат, описание, позволит эффективно управлять данными и быстро предоставлять их покупателям.

Все вышеперечисленные аспекты являются основными при создании базы данных для интернет-магазина одежды. Следует переходить к следующим этапам разработки, которые включают проектирование физической модели базы данных и её реализацию.

В целом, разработка логической модели базы данных в основном связана с теоретическими аспектами. Разработка физической модели базы данных, которая будет рассмотрена в следующей главе, фокусируется на практических аспектах и реализации задуманного. Разделение этих этапов необходимо для полного понимания процесса разработки и предотвращения возможных вопросов в будущем. [4]

## **Анализ связей между сущностями**

Анализ связей между сущностями является важной частью при проектировании логической модели базы данных. Он позволяет определить, как сущности (таблицы) связаны друг с другом и позволяет выбрать наиболее подходящую связь.

В базах данных существует три основных способа связи между сущностями:

1. Один ко многим (one-to-many) - способ связи, при котором одна сущность может быть связана с многими другими сущностями. Например, у одного покупателя может быть несколько отзывов, а один отзыв может принадлежать одному пользователю.

2. Многие ко многим (many-to-many) - способ связи, при котором многие сущности одного типа могут быть связаны с многими сущностями другого типа. Например, у одного автора может быть несколько книг, также у одной книги может быть несколько авторов.

3. Один к одному (one-to-one) - способ связи, при котором одна сущность может быть связана только с одной другой сущностью. Например, у одного пользователя может быть только один паспорт и наоборот, один паспорт может принадлежать только одному владельцу.

При анализе связей между сущностями необходимо учитывать количество объектов одной сущности, которые могут быть связаны с объектами другой сущности. Это называется кардинальностью связи

Также необходимо учитывать направленность связи, то есть определить, какая сущность является владельцем связи, а какая зависимой. Например, в связи "один ко многим" между сущностями "категория товаров" и "товар" категория товаров является владельцем связи, а товар - зависимой сущностью. [5]

Благодаря анализу связей между сущностями можно создать четкую и понятную структуру базы данных, которая будет эффективно работать и обеспечивать нужный функционал для интернет-магазина

Кроме того, при разработке базы данных необходимо убедиться, что все данные корректны и согласованы между собой. Этот процесс называется обеспечением целостности данных.

Сущностная целостность (entity integrity) гарантирует, что каждая сущность имеет уникальный идентификатор и что все атрибуты сущности не могут быть NULL, если это не предусмотрено моделью данных.

Ссылочная целостность (referential integrity) гарантирует, что все связи между сущностями корректны и что нет ссылок на несуществующие экземпляры сущностей. [5]

При проектировании базы данных необходимо учитывать производительность и масштабируемость. В эти аспекты могут быть включены: оптимизации запросов к базе данных, выбор подходящей СУБД (система управления базами данных), декомпозиция таблиц и другие аспекты.

Важно также учитывать требования к безопасности данных. Это может включать шифрование данных, использование протоколов безопасной передачи данных, контроль доступа к базе данных и другие методы защиты. Разработка базы данных требует значительных усилий, но с использованием правильных инструментов можно облегчить процесс и достичь желаемого результата.

Еще одним важным аспектом разработки логической модели базы данных является нормализация. Нормализация — это процесс разбиения таблиц на более мелкие и более управляемые таблицы с целью устранения избыточности данных и обеспечения целостности данных. Существует несколько нормальных форм, каждая из которых имеет свои требования к структуре таблиц.

1. Первая нормальная форма требует, чтобы каждая таблица имела уникальный первичный ключ и чтобы все атрибуты таблицы были атомарными, то есть неделимыми.

2. Вторая нормальная форма требует, чтобы все неключевые столбцы зависели от всего первичного ключа, а не только от его части.

3. Третья нормальная форма требует, чтобы все неключевые столбцы не зависели друг от друга. Неключевые столбцы - это те столбцы, которые не используются для уникальной идентификации записи в таблице. Например, в таблице ‘Пользователи’ столбец c id пользователя является ключевым столбцом, так как используется для уникальной идентификации записей, а другие столбцы - нет.

Нормализация помогает избежать избыточности данных, обеспечить целостность данных и упростить модель данных. Однако в некоторых случаях нормализация может привести к снижению производительности, поэтому необходимо найти баланс между нормализацией и производительностью. [10]

## **1.3. Разработка схемы базы данных с использованием нотации Crow’s foot**

Нотация “Crow’s foot” (“Воронья лапа”) - это графическая нотация, используемая для создания схем баз данных. Данная нотация была разработана в 1976 году Гордоном Эверестом и в настоящее время люди продолжают широко использовать ее. Нотация – это система обозначения.

“Воронья лапа” использует графические символы для обозначения связей между таблицами и столбцами для определения отношений между ними. Таблицы представлены в виде прямоугольников, а столбцы – в виде овалов, связи представлены в виде линий, выполняющих роль соединения таблиц. В целом, существует несколько типов отношений, которые можно посмотреть в виде “Вороньей лапы”:

Один ко одному (1:1)

Один ко многим(1:М)

Многие ко многим (M)

Назначение этих связей было рассмотрено в предыдущих разделах, поэтому повторное обсуждение не требуется. Для каждого типа связей используются уникальные графические символы. Например, для обозначения связи “один ко многим” применяется стрелка, указывающая от объекта “один” к объекту “многие”. Эта нотация также предоставляет инструменты для представления правил целостности данных и ограничений, таких как первичные и внешние ключи.

Популярность и частая используемость данной нотации заключается в том, что графические иллюстрации часто способствуют лучшему пониманию структуры данных. Нотация “Воронья лапка” позволяет визуализировать структуру базы данных, обеспечивая наглядное представление её компонентов и их взаимосвязей. [6]

## **1.4. Администрирование базы данных**

Администрирование базы данных является неотъемлемой частью разработки, а также поддержки любого приложения или системы, использующей базу данных. Ниже будут приведены некоторые ключевые моменты, связанные с администрированием данных:

**Настройка СУБД:** Настройка параметров СУБД является одной из важных частей для обеспечения работы базы данных. В этот процесс включены настройка количества соединений, размера буфера обмена, конфигурация кэширования и множество других параметров, которые могут влиять на производительность базы данных.

**Резервное копирование:** Этот аспект является критически важным, так как позволяет не потерять все данные и застраховать себя от различных ситуаций. Необходимо регулярно тестировать и проверять восстановление данных из резервной копии для обеспечения безопасности данных в будущем.

**Мониторинг:** Мониторинг включает анализ и сбор информации о состоянии базы данных. Мониторинг производительности базы данных: эта часть является необходимой для обеспечения постоянной работы приложения, которая минимизирует риски появления сбоев в системе. Также используется для мониторинга ЦП, памяти, времени отклика запросов.

**Управление данными:** Этот аспект администрирования базы данных включает следующие задачи:

1. Очистка данных: Необходимо регулярно проводить очистку данных, чтобы удалять дубликаты, исправлять ошибки и устаревшие данные. Это помогает сохранить базу данных точной и эффективной.

2. Архивирование данных: Архивирование данных является процессом перемещения старых или редко используемых данных из активной базы данных в архивное хранилище. Это может помочь улучшить производительность и сократить расходы на хранение данных, так как она будет содержать меньшее количество данных, которые необходимо обрабатывать. Также этот процесс может помочь сократить расходы на хранение данных, по причине того, что архивные хранилища как правило дешевле, нежели активные базы данных.

Если представить администрирование баз данных (БД) и систем управления базами данных (СУБД) в кратком виде, можно выделить следующие аспекты:

Администрирование БД:

* Создание объектов базы данных
* Разработка структуры системы безопасности
* Контроль целостности данных
* Отслеживание производительности

СУБД:

* Важность установки и обновления версий
* Запуск и установка служб СУБД
* Настройка СУБД
* Управление учетными записями пользователей
* Создание и модификация базы данных

Были разобраны основные моменты администрирования баз данных, которые в будущем обязательно понадобятся в работе. [7]

# **Глава 2. Проектирование физической модели базы данных и реализация**

## **2.1. Проектирование физической модели базы данных**

Прежде чем начать писать о разработке физической модели базы данных, стоит рассмотреть выбор системы управления базами данных(СУБД). В качестве СУБД был выбран PostgreSQL. Эта система не имеет значительных отличий от MySQL (кроме синтаксиса). PostgreSQL при работе с Python (Django ORM) предоставляет некоторые дополнительные функции, что тоже стало одним из факторов выбора, так как в будущем будет полезно знать об этих предоставляемых функциях.

Начало проектирования базы данных. [2]

Создание базы данных:

CREATE DATABASE shop\_db;

После создания базы данных, необходимо перейти к созданию таблиц, которые будут разработаны:

1. Таблица “Product”. Данная таблица хранит информацию о продуктах. Начнем описание полей таблицы и типов данных, которые будут использованы:

Поле [id] - хранит идентификатор продукта, тип данных SERIAL означает, что идентификатор будет автоматически инкрементироваться. Это поле является PRIMARY KEY (первичный ключ). В будущем на это поле будут ссылаться другие таблицы с помощью внешнего ключа (FOREIGN KEY). Обычно принято использовать такие значения как: NOT NULL (не может иметь пустое значение) и указать тип данных INTEGER, но в случае использования PostgreSQL это не требуется, так как SERIAL автоматически устанавливает эти значения.

[title] - название продукта, тип данных varchar (255), NOT NULL (не может иметь пустое значение).

[description] - описание продукта, тип данных TEXT, не имеет ограниченного количества символов.

[price] - DECIMAL(10, 2) - используется для указания точных значений, в нашем случае цена будет состоять из 8 чисел до запятой и 2 после.

[quantity] - количество товаров, тип данных INT, значение по умолчанию = 0.

[images] - изображения продуктов, varchar для указания пути к картинке.

[category\_id] - внешний ключ, который ссылается на таблицу “Category”.

SQL скрипт для создания таблицы “Product”:

CREATE TABLE Product (

id SERIAL PRIMARY KEY,

title varchar(255) NOT NULL,

description TEXT,

price DECIMAL(10, 2) NOT NULL,

quantity INT CHECK(quantity > 0) NOT NULL,

images varchar(255) NOT NULL,

category\_id INT REFERENCES Category(id));

ПРИМЕЧАНИЕ!!

На этом этапе специально максимально подробно описано каждое поле, даже самое простое. На следующих этапах знакомые нам уже поля не будут описываться так подробно, так как мы уже с ними ознакомлены.

2. Таблица “Category”. Данная таблица хранит информацию о категориях товара.

[id] - идентификатор категории.

[name] - название категории

[description] - описание категории (необязательное поле, но для ясности, что к чему, будет полезно знать).

SQL скрипт для создания таблицы “Category”:

CREATE TABLE Category (

id SERIAL PRIMARY KEY,

name VARCHAR(255),

description TEXT

);

3. Таблица “Role”. Данная таблица хранит информацию о ролях пользователей. Не был использован ENUM, хотя это тоже возможно, но было решено создать отдельную таблицу для хранения дополнительной информации и обеспечения гибкости управления ролями.

Поля те же самые, что и были описаны в таблице “Сategory”, но в данном случае хранится информация о роли пользователя

SQL скрипт для создания таблицы “Role”:

CREATE TABLE Role (

id SERIAL PRIMARY KEY,

name varchar(255) NOT NULL,

description TEXT

);

4. Таблица “Users”. Данная таблица хранит информацию о пользователях.

Будут описаны только незнакомые поля. Все введенные данные о пользователе будут необходимы при регистрации и авторизации.

[username] - имя пользователя, тип данных varchar.

[email] - email пользователя

[password] - Пароль пользователя, хеширование пароля еще не реализовано и будет выполнено на backend части с использованием специализированных фреймворков.

[first\_name], [last\_name] - имя и фамилия пользователя, оба поля имеют тип данных varchar.

[role\_id] - Foreign Key, который ссылается на таблицу “Role”.

SQL скрипт для создания таблицы “Users”:

CREATE TABLE Users (

id SERIAL PRIMARY KEY,

username varchar(255) NOT NULL,

email varchar(255) UNIQUE NOT NULL,

password varchar(255) NOT NULL,

first\_name varchar(255) NOT NULL,

images varchar(255),

last\_name varchar(255) NOT NULL,

role\_id INT REFERENCES Role(id)

);

5. Таблица “Orders”. Хранит информацию о заказах (кем и когда было создано и какой статус у заказа)

[status] - Статус заказа, тип данных varchar.

[created\_at] - Дата создания, тип данных Timestamp.

[updated\_at] - Дата обновления, тип данных Timestamp.

[user\_id] - Внешний ключ, который ссылается на таблицу “Users”.

SQL скрипт для создания таблицы “Orders”:

CREATE TABLE Orders (

id SERIAL PRIMARY KEY,

status varchar(255) NOT NULL,

created\_at TIMESTAMP NOT NULL,

updated\_at TIMESTAMP NOT NULL,

user\_id INT REFERENCES Users(id)

);

6. Таблица “OrderDetails”. Хранит информацию о деталях заказа (какой товар был заказан, количество, цена)

[order\_id] - Foreign Key, который ссылается на таблицу “Orders”

[product\_id] - Foreign Key, который ссылается на таблицу “Product”

SQL скрипт для создании таблицы OrdersDetails

CREATE TABLE OrderDetails (

id SERIAL PRIMARY KEY,

quantity INT CHECK(quantity > 0),

price DECIMAL(10, 2),

order\_id INT REFERENCES Orders(id),

product\_id INT REFERENCES Product(id)

);

7. Таблица “Review”. Данная таблица нужна для хранения информации об отзывах на товары.

[rate] - Отзыв, тип данных Float, для указания точного значения отзыва. В ином случае можно было бы выбрать Decimal. Оценка должна варьироваться от 0 до 5.

[comment] - комментарий к отзыву.

[user\_id] - Foreign Key, который ссылается на таблицу “Users”

[product\_id] - Foreign Key, который ссылается на таблицу “Product”

SQL скрипт для создания таблицы “Review”:

CREATE TABLE Review (

id SERIAL PRIMARY KEY,

rate FLOAT CHECK(rate >= 0 AND rate <= 5) NOT NULL,

comment TEXT,

created\_at TIMESTAMP NOT NULL,

user\_id INT REFERENCES Users(id),

product\_id INT REFERENCES Product(id));

8. Таблица “Cart” (корзина). Эта таблица хранит в себе информацию о корзине (когда и кем было создано, дата и время последнего обновления).

[user\_id] - Foreign Key, который ссылается на таблицу “Users”.

CREATE TABLE Cart (

Id SERIAL PRIMARY KEY,

user\_id INT REFERENCES Users(id),

created\_at TIMESTAMP NOT NULL,

updated\_at TIMESTAMP NOT NULL);

9. Таблица “CartDetails”, таблица, которая хранит в себе информацию о деталях корзины (товар, количество товаров в корзине, цена)

[product\_id] - Foreign Key, который ссылается на таблицу “Product”

[cart\_id] - Foreign Key, который ссылается на таблицу “Cart”

CREATE TABLE CartDetails (

id SERIAL PRIMARY KEY,

quantity INT CHECK(quantity > 0),

price DECIMAL(10, 2),

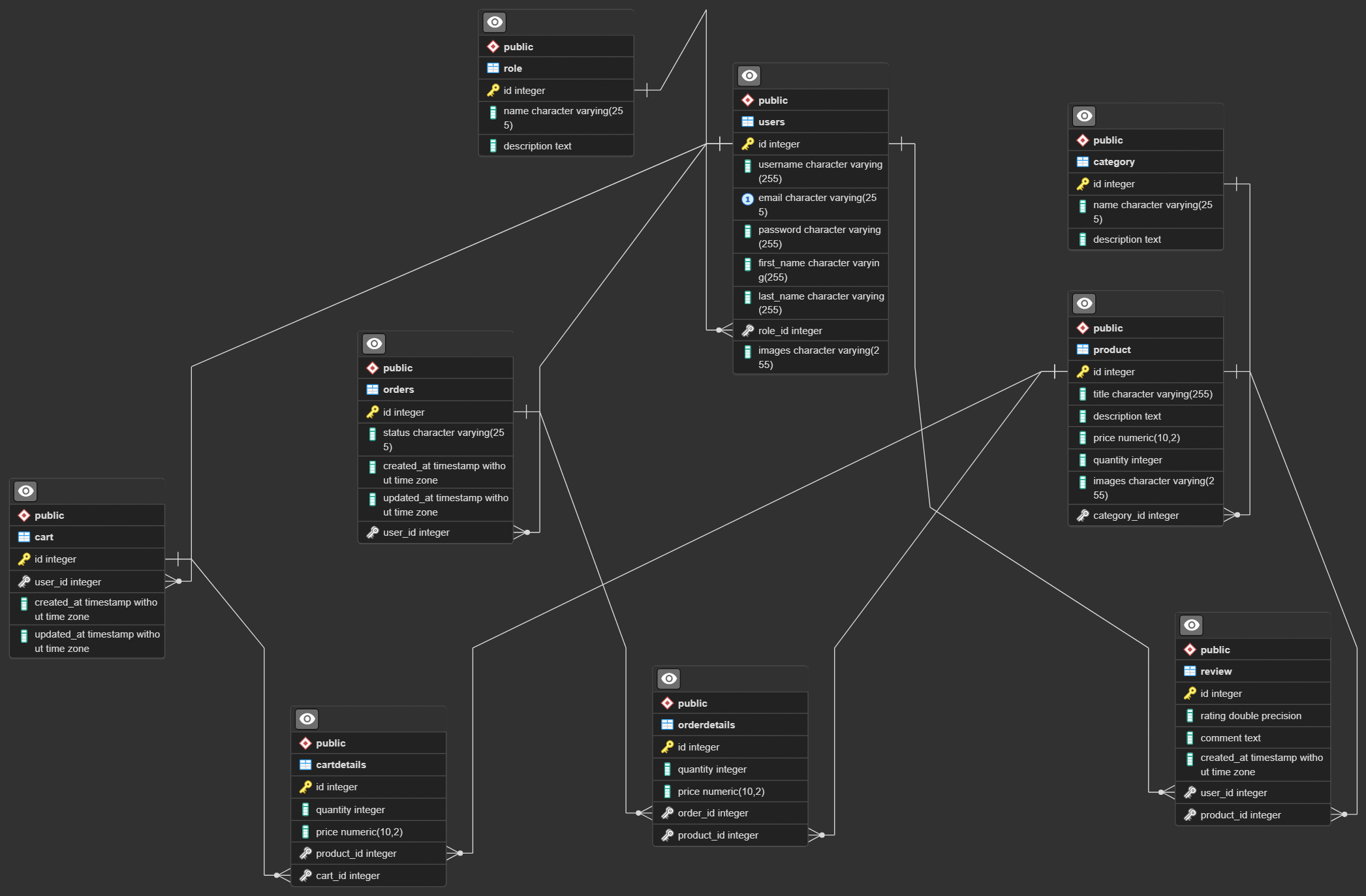
product\_id INT REFERENCES Product(id),

cart\_id INT REFERENCES Cart(id)

);

В итоге, мы создаем базу данных из 9 таблиц. ERD схема для данных таблиц будет предоставлена позже. [2]

ERD - это графическое представление структуры базы данных, который предоставляет возможность детально рассмотреть и изучить все сущности и связи между ними. Сущности(таблицы) в ERD-схеме представляются в виде прямоугольников с названиями этих сущностей. Атрибуты (столбцы) - это свойства наших сущностей, они могут быть уникальными или неуникальными, а также иметь разные типы данных. В теории, читая все эти определения, может казаться, что ничего не понятно, поэтому лучше все рассматривать с практической части, так как такой подход больше позволяет вникать в суть. На следующей странице представлено изображение ERD моей базы данных, демонстрирующее ее структуру и взаимосвязи. [8]

  **Рис 1.**

В данной картинке мы имеем 9 таблиц связанных между собой линиями и элементами похожими на вороньи лапки. В теоретической части мы обсудили нотацию “Crow’s foot” (вороньи лапки), которая уже использована в нашей ERD-схеме для обозначения связей.

Как видно из нашего ERD, между всеми таблицами реализована связь “один ко многим”. Это значит, что одна запись из одной таблицы может иметь несколько записей во второй таблице, но одна запись из второй таблицы может иметь только одну запись из первой.

Примеры: Отношение “Users к Role” (one-to-many): у одной роли может быть несколько пользователей, но у одного пользователя может быть только одна роль. Разберем отношение немного сложнее. На данном изображении, отношение таблицы “Cart” к таблице “CartDetails” - (one-to-many): у одной корзины может быть несколько позиций (записей) из “CartDetails”, но у одной позиции может быть только одна корзина.

## **2.2 Создание ролей и предоставление привилегий**

Создание ролей является важным аспектом безопасности при разработке базы данных, так как позволяет контролировать доступ пользователей к определенным ресурсам и их возможности совершать какие-либо действия. Например, можно создать роль со значением ‘read-only’, в которой пользователь может только читать данные и не более. В разработанной базе данных, были созданы 3 роли: администратор, менеджер, покупатель. Всем трем были предоставлены разные привилегии над таблицами. Ниже будет рассмотрен пример создания ролей и предоставления привилегий. [11]

Создание ролей:

CREATE ROLE admin WITH LOGIN PASSWORD '\*\*\*\*\*\*';

CREATE ROLE customer WITH LOGIN PASSWORD '\*\*\*\*\*\*';

CREATE ROLE manager WITH LOGIN PASSWORD '\*\*\*\*\*\*\*';

Вместо \* каждому установлен свой пароль.

Предоставление доступа администратору:

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE Product TO admin;

...

Администратору были предоставлены только те привилегии, которые необходимы для работы над всеми таблицами. Решено не предоставлять права для CREATE, ALTER, DROP, TRUNCATE. Такие права остаются за суперпользователем

Покупателю:

...

GRANT SELECT ON TABLE Product TO customer;

GRANT SELECT ON TABLE Category TO customer;

...

Покупателю были предоставлены права только на выборку данных из таблиц “Product” и “Category”.

Менеджер может выполнять действия SELECT, INSERT, UPDATE, DELETE для таблиц “Product” и “Category”. Для других таблиц менеджер имеет следующие привилегии:

...

GRANT SELECT, UPDATE ON TABLE Orders TO manager;

GRANT SELECT, UPDATE ON TABLE OrderDetails TO manager;

GRANT SELECT ON TABLE Cart TO manager;

GRANT SELECT ON TABLE CartDetails TO manager;

## **2.3. Типовые запросы**

Типовые запросы - это запросы, которые используются для выведения определенной информации из базы данных. Для моей базы данных были написаны 8 типовых запросов. Все запросы рассматриваться не будут, так как их много, будут разобраны только самые интересные.

1. Вывод общей суммы заказов за определенный период. В этом запросе используется SELECT для извлечения данных, агрегатная функция SUM для подсчета общей суммы, все эти данные извлекаются из таблицы “OrderDetails”. Дату и время созданного заказа необходимо взять из таблицы “Orders” с помощью подзапроса. В подзапросе извлекается id заказа который был создан в период от 4 апреля до 7 апреля, после чего срабатывает основной запрос:

SELECT SUM(quantity \* price) as total\_sum

FROM OrderDetails

WHERE order\_id IN (SELECT id FROM Orders WHERE created\_at BETWEEN '2024-04-01' AND '2024-04-07');

2. Вывод всех товаров в корзине пользователя. Извлекаются все товары и количество товаров из корзины с помощью SELECT, таблицы Product и “CartDetails” связываются с помощью JOIN. Связываться они будут по “Primary Key” в таблице Product и “Foreign Key” в таблице “CartDetails”. Для того, чтобы узнать корзину пользователя, необходимо с помощью WHERE получить “cart\_id” равный 1, который связан с определенным пользователем

SELECT Product.\*, CartDetails.quantity

FROM CartDetails

JOIN Product ON Product.id = CartDetails.product\_id

WHERE CartDetails.cart\_id = 1;

3. Вывод списка товаров, которые не были заказаны за определенный период. Извлекаются все товары с таблицы “Product”, где будут извлекаться все идентификаторы, которые не совпадают со значением внутри списка и созданные в период от 16 до 31 марта.

IN - условие, в котором будем получать все совпадающие со списком объекты.

NOT IN - условие, в котором будем получать все не совпадающие со списком объекты.

SELECT \* FROM Product

WHERE id NOT IN (SELECT product\_id FROM OrderDetails

WHERE order\_id IN(SELECT id FROM Orders WHERE created\_at

BETWEEN '2024-03-16' AND '2024-03-31'));

4. Вывод списка пользователей, сделавших заказ в определенный период. Извлекаются все пользователи с помощью SELECT и используя команды WHERE и IN получаются именно те пользователи, которые совпадают указанным требованиям внутри скобок. В нашем случае необходимо вывести пользователей, сделавших заказ с 1 по 7 апреля.

SELECT \* FROM Users WHERE id IN (SELECT DISTINCT user\_id FROM Orders WHERE created\_at BETWEEN '2024-04-01' AND '2024-04-07');

## **2.4. Транзакции**

Транзакции - предоставляют собой операции, которые выполняются как единое целое. Эти операции используются для обеспечения согласованности и целостности базы данных.

Транзакции имеют такие свойства как:

Атомарность (Atomicity) – транзакция либо выполняется полностью, либо не выполняется вообще.

Согласованность(Consistency) - транзакция переводит базу данных из одного согласованного состояния в другое.

Изоляция(Isolation) - транзакция должна выполнятся независимо от других транзакций выполняющихся в это же время.

Долговечность(Durabilty) - изменения, которые были сделаны транзакцией должны быть сохранены в базе данных и в случае сбоя или системного отказа не должны быть потеряны. [9]

Рассмотрим пример транзакции, выполненной для моей базы данных. Для краткости будем рассматривать только одну транзакцию.

1. Создание заказа и добавление товаров в него. DO $$ является частью синтаксиса PostgreSQL для объявления блоков кода. DECLARE используется для объявления переменной “order\_id”, которая будет хранить id нового заказа. BEGIN - начало транзакции. Создаем заказ для таблицы “Orders”, возвращаем id этого заказа и присваиваем “order\_id”, затем добавляем нашу переменную и данные для “OrderDetails”. [12]

DO $$

DECLARE

order\_id INTEGER;

BEGIN

-- Создание нового заказа

INSERT INTO Orders (status, user\_id, created\_at, updated\_at)

VALUES ('Не оплачено', 1, NOW(), NOW())

RETURNING id INTO order\_id;

-- Добавление товаров в заказ

INSERT INTO OrderDetails (order\_id, product\_id, quantity, price)

VALUES

(order\_id, 1, 2, 1299.00),

(order\_id, 2, 1, 5000.00);

COMMIT;

END;

$$;

## **2.5. Триггеры**

Триггеры - это процедуры, которые автоматически срабатывают в ответ на определенные события в базе данных. В качестве событий могут выступать удаление, создание, обновление данных в таблицах.

1. Обновление количества товаров в корзине после добавления или удаления товаров. Сначала создается функция, которая не принимает параметры. RETURNS TRIGGER – указывает на то, что данная функция возвращает триггер. $$ - начало блока кода. Объявляется переменная. Выводим общее количество товаров из таблицы “CartDetails”, где id корзины равен новому значению “cart\_id” и результат присваивается переменной. Обновляется таблица, изменяя количество товаров и возвращается новое значение. Создается триггер, который будет срабатывать после вставки или удаления данных для каждой строки. В конце вызывается функция. [13]

CREATE OR REPLACE FUNCTION update\_product\_quantity\_in\_cart()

RETURNS TRIGGER AS $$

DECLARE

new\_product\_quantity INTEGER;

BEGIN

SELECT SUM(cd.quantity) INTO new\_product\_quantity

FROM CartDetails AS cd

WHERE cd.cart\_id = NEW.cart\_id;

UPDATE CartDetails SET quantity = new\_product\_quantity

WHERE cart\_id = NEW.cart\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_quantity

AFTER INSERT OR DELETE ON CartDetails

FOR EACH ROW

EXECUTE FUNCTION update\_product\_quantity\_in\_cart()

## **2.6. Хранимые процедуры**

Хранимые процедуры имеют возможность содержать в себе готовый код, хранящиеся в базе данных, которую при необходимости можно вызвать. Также они позволяют выполнять сложные операции с данными. Хранимые процедуры похожи на функции, могут также принимать параметры и возвращать значения, но основное различие между ними в том, что хранимые процедуры могут изменять данные в базе данных, выполняя такие операции INSERT, DELETE, UPDATE. Функции же возвращают результаты вычислений и не могут изменять данные. Есть еще ряд преимуществ использования хранимых процедур, но пора уже перейти к практике. В разработанной мною базе данных 3 хранимых функций (процедуры), но рассмотрим только одну и самую интересную из них. Перед тем как приступить, нужно обратить внимание на то, что в PostgreSQL хоть и есть команда CREATE PROCEDURE, но ее на практике часто не используют. И в целом, вместо хранимых процедур используют хранимые функции, которые могут выполнять те же самые задачи. [14]

1. Создается хранимая функция “add\_to\_cart”, которая в качестве параметров имеет “p\_user\_id” – id пользователя, “p\_product\_id” – id товара и “p\_quantity” - количество товара. RETURNS VOID - никакого значения функция возвращать не будет. $$ - начало блока кода.

Добавление товаров в корзину:

CREATE OR REPLACE FUNCTION add\_to\_cart(p\_user\_id INTEGER, p\_product\_id INTEGER, p\_quantity INTEGER)

RETURNS VOID AS $$

. . .

2. С помощью DECLARE объявляются переменные “cart\_id” и product\_price. Первая переменная будет хранить в себе id корзины, а вторая цену товара.

BEGIN - начало работы. Получаем id из корзины, где “user\_id” = текущему пользователю, который будет передавать аргументы в параметры, затем результат присваивается переменной “cart\_id”. Если корзины не существует, создается новая.

DECLARE

cart\_id INTEGER;

product\_price DECIMAL(10, 2);

BEGIN

SELECT id INTO cart\_id FROM Cart WHERE user\_id = p\_user\_id;

IF cart\_id IS NULL THEN

INSERT INTO Cart(user\_id, created\_at, updated\_at)

VALUES (p\_user\_id, NOW(), NOW()) RETURNING id INTO cart\_id;

END IF;

...

3. Получаем цену товара, где id нашего товара = id товару, который мы получим из параметра. После этого создается корзину. Сохраняются изменения, завершается хранимая функция и закрывается блок кода.

SELECT price INTO product\_price FROM Product WHERE id = p\_product\_id;

INSERT INTO CartDetails (quantity, price, product\_id, cart\_id)

VALUES (p\_quantity, product\_price, p\_product\_id, cart\_id);

COMMIT;

END;

$$ LANGUAGE plpgsql;

## **2.7. Локальные переменные**

Локальные переменные - используются для хранения временных значений, применяемых внутри блока хранимых процедур, триггеров и функций. Работают такие переменные только внутри блока кода, после завершения выполнения кода все удаляется. При разработке базы данных было создано 7 локальных переменных (“total\_price”, “product\_count”, “user\_role”, “rating”, “comment”, “user\_id”, “product\_id”), для выполнения определенных действий внутри блока кода. [15]

## **2.8. Представления**

Представления позволяют создавать таблицы, на основе результатов запроса. В моей базе данных создано 3 представления. Рассмотрим 1 из них.

1. Вывод всех заказов пользователя. Извлекаются данные из всех столбцов таблицы “Orders”, а также имя пользователя и количество товаров в заказе. Затем соединяются таблицы “Orders” и “OrderDetails” для получения данных о заказе. Затем Product с “OrderDetails” с целью получения информации о товаре. “Users” с “Orders” для получения данных о пользователе, которому принадлежит определенное количество заказов. [16]

CREATE VIEW user\_orders AS

SELECT

o.id, o.user\_id, o.created\_at, o.updated\_at, o.status,

p.id AS product\_id, p.title, p.price,

u.username,

od.quantity

FROM Orders AS o

JOIN OrderDetails AS od ON od.order\_id = o.id

JOIN Product AS p ON od.product\_id = p.id

JOIN Users AS u ON u.id = o.user\_id;

## **2.9. Условия**

Условия в SQL предназначены для получения данных в запросе, которые соответствуют определенным критериям, после чего с полученными данными позволяют выполнять разные операции. В процессе написания условий, были созданы новые транзакции, существующие не были использованы. В разработанной базе данных реализованы две транзакции, с условиями внутри блока кода. В первой транзакции проверяется наличие товаров на складе перед добавлением их в заказ, во второй проверяется наличие товара на складе перед добавлением его в корзину. В первой транзакции, если на складе есть товар, создается заказ, иначе получаем ошибку: ‘Недостаточно товаров на складе’. Во второй транзакции такая же логика, если товар существует, создается корзина, иначе генерируется ошибка.

см. Приложение 1

## **2.10. Обработчики исключений**

Обработчики исключений в SQL используются для того, чтобы отслеживать и обрабатывать ошибки, которые могут возникнуть во время выполнения кода. В рамках разработки базы данных для интернет-магазина, были написаны две функции с обработчиками исключений. В первом случае, мы получим исключение при отсутствии товара на складе, а во втором, при добавлении отзыва, если такой отзыв уже существует.

## **2.11. Пользовательские функции**

Это функции созданные пользователем для выполнения определенных задач, которые могут принимать аргументы и возвращать значения, а также выполнять сложные вычисления. В моей базе данных разработано две пользовательские функции. Первая для расчета общей суммы заказа, вторая для расчета среднего рейтинга. Но рассмотрим только одну из них. [14]

см. Приложение 2

# **Заключение**

В итоге - в процессе работы была разработана база данных для интернет-магазина одежды, которая содержит необходимые свойства для полноценной работы. Решены задачи, необходимые для достижения поставленной цели. Для проверки работоспособности базы данных, были добавлены тестовые данные, в ходе которых были получены ожидаемые результаты. База данных спроектирована хорошо, для учебных целей, дополнительной практики, получения опыта и знаний, она подходит, но при этом приходит понимание того, что этого недостаточно, так как бизнес-логика на этом не заканчивается. К примеру, как минимум должна быть добавлена таблица с данными об оплате, так как для покупки, пользователи должны платить деньги, но пока из-за недостаточности знаний и практики (временно), реализовать не получилось. Не все разработано идеально, но определенным ожиданиям соответствует. Еще многое нужно узнать и продолжать получать еще больше информации в этой области.

Есть потенциальные возможности для развития и улучшения бизнес-процессов этой базы данных, а также при возможности использовать уже на реальной практике. В плане безопасности базы данных были созданы отдельные роли для каждого пользователя, чтобы определенный пользователь имел право совершать какие-либо действия, которые были предоставлены ему. Такой подход защищает от непредвиденных действий злоумышленников.

Таким образом, можно заявить, что в процессе разработки базы данных была освежена память многими забытыми темами, что оказалось полезным. Благодаря этому был получен ценный багаж знаний и практический опыт.

# **Список источников**

1. Актуальность интернет-магазинов в наше время [Электронный ресурс] / - <https://www.ds77.ru/news/1063845/> / <https://www.cossa.ru/synergy_academy/308100/>

2. Анализ требований и проектирование модели базы данных [Электронный ресурс] / - <https://bseu.by/it/tohod/lekcii4_6.htm>

3. Пример базы данных интернет-магазина [Электронный ресурс] / - <https://fabric.inc/blog/commerce/ecommerce-database-design-example>

4. Отличия между физической и логической моделью базы данных [Электронный ресурс] / - <https://aws.amazon.com/ru/compare/the-difference-between-logical-and-physical-data-model/>

5. Связи между таблицами и целостность базы данных [Электронный ресурс] / - <https://habr.com/ru/articles/488054/> https://www.astera.com/ru/type/blog/data-integrity-in-a-database/

6. Нотация “Crow’s foot” [Электронный ресурс] / - <https://vertabelo.com/blog/crow-s-foot-notation/>

7. Администрирование базы данных [Электронный ресурс] / - <https://bseu.by/it/tohod/lekcii9.htm> / <https://ppt-online.org/157132>

8. Что такое ERD [Электронный ресурс] / - <https://www.lucidchart.com/pages/ru/erd-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0>

9. Что такое ACID [Электронный ресурс] / - <https://habr.com/ru/articles/555920/>

10. Нормализация данных и нормальные формы [Электронный ресурс] / - [https://www.kaznu.kz/content/files/news/folder](https://www.kaznu.kz/content/files/news/folder22759/%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%205%20.pdf)

11. Создание ролей и предоставление привилегий. [Электронный ресурс] / - <https://postgrespro.ru/docs/postgrespro/10/sql-createrole>

12. Транзакции [Электронный ресурс] - <https://postgrespro.ru/docs/postgrespro/10/tutorial-transactions>

13. Триггеры. Основная информация: Глава "Практика — добавление информации в две таблицы" [Электронный ресурс] / - <https://timeweb.cloud/tutorials/postgresql/postgresql-triggery-sozdanie-udalenie-primery>

14. - Хранимые процедуры и пользовательские функции [Электронный ресурс] / - <https://postgrespro.ru/docs/postgrespro/9.5/sql-createfunction>

15. - Локальные переменные [Электронный ресурс] / - <https://sky.pro/wiki/sql/obyavlenie-i-ispolzovanie-peremennykh-v-postgre-sql-8-3/>

16. Представления [Электронный ресурс] - <https://postgrespro.ru/docs/postgrespro/10/sql-createview>

# **Приложение 1.** Условия. Проверка наличия товара на складе перед добавлением его в заказ:

DO $$

DECLARE

product\_quantity INTEGER;

BEGIN

SELECT quantity INTO product\_quantity FROM Product WHERE id = 1;

IF product\_quantity >= 1 THEN

INSERT INTO OrderDetails (quantity, price, order\_id, product\_id)

VALUES (2, 1299.00, 2, 1);

UPDATE Product SET quantity = quantity - 2 WHERE id1;

ELSE

RAISE NOTICE 'Недостаточно товаров на складе';

END IF;

COMMIT;

END $$;

# **Приложение 2.** Пользовательские функции. Создание функции для расчета общей суммы заказа:

CREATE OR REPLACE FUNCTION calculate\_total\_price(order\_id INTEGER)

RETURNS DECIMAL(10, 2) AS $$

DECLARE

total\_price DECIMAL(10, 2);

BEGIN

SELECT SUM(od.quantity \* od.price) INTO total\_price

FROM OrderDetails AS od

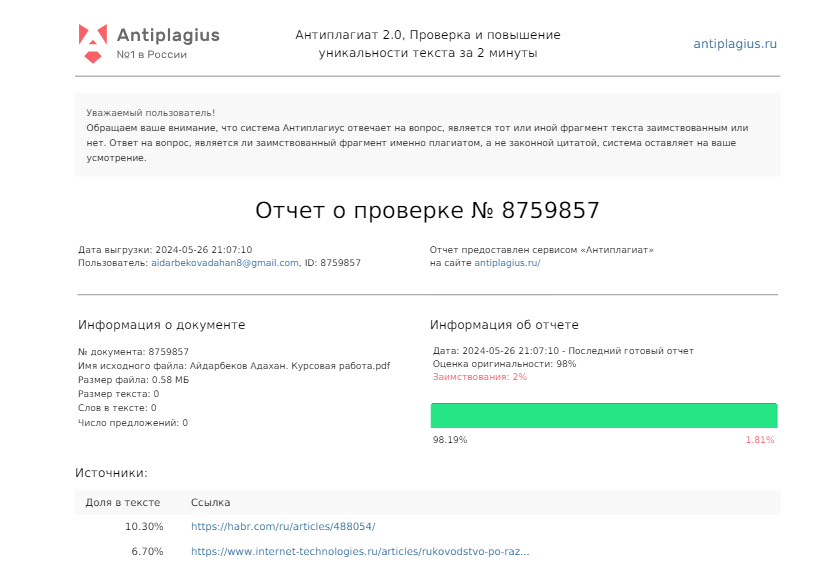
WHERE od.order\_id = order\_id;

RETURN total\_price;

END;

$$ LANGUAGE plpgsql;

# **Приложениие 3. Антиплагиат**



# **Приложение 4. QR код для репозитория**



Ссылка на репозиторий: <https://github.com/adeqoou/pgdb_work>