Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Модели данных и системы управления базами данных»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.В.Давыдчик |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему

**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ИНТЕРНЕТ-МАГАЗИНА**

**ЭЛЕКТРОНИКИ**

БГУИР КП 1-40 04 01

|  |  |
| --- | --- |
|  | Выполнил студент группы 153501  Власенко Тимофей Павлович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**СОДЕРЖАНИЕ**

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc182755270)

[1.1 Обзор веб-сайта *Onliner* 5](#_Toc182755271)

[2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 8](#_Toc182755272)

[2.1 Функциональные требования к разрабатываемой базе данных 8](#_Toc182755273)

[2.2 Анализ существующих подходов к разработке баз данных 9](#_Toc182755274)

[2.3 Выбор базы данных для разработки предметной области 12](#_Toc182755275)

[3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 14](#_Toc182755276)

[3.1 Инфологическая модель 14](#_Toc182755277)

[3.2 Разработка даталогической модели базы данных 14](#_Toc182755278)

[4 РАЗРАБОТКА БАЗЫ ДАННЫХ 15](#_Toc182755279)

[4.1 Создание исходных таблиц, индексов и ограничений 15](#_Toc182755280)

[4.2 Создание хранимых процедур 15](#_Toc182755281)

[4.3 Реализация триггеров 15](#_Toc182755282)

[ЗАКЛЮЧЕНИЕ 16](#_Toc182755283)

[СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ 17](#_Toc182755284)

[ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программы 18](#_Toc182755285)

[ПРИЛОЖЕНИЕ Б (обязательное) Схема базы данных 19](#_Toc182755286)

**ВВЕДЕНИЕ**

Интернет-магазины стали неотъемлемой частью современной торговли и повседневной жизни. Они предоставляют пользователям удобный способ приобретать товары, экономя время и ресурсы. Среди разнообразия товаров, доступных онлайн, электроника занимает одно из ведущих мест благодаря постоянному спросу и технологическому прогрессу. С ростом популярности онлайн-покупок увеличивается и необходимость в эффективных системах, которые позволяют организовать, управлять и оптимизировать процессы продажи электроники.

Цифровые платформы для интернет-магазинов электроники помогают пользователям быстро находить нужные товары, сравнивать характеристики и цены, оформлять заказы и отслеживать их доставку. Они также предоставляют владельцам магазинов удобные инструменты для управления ассортиментом, аналитики и взаимодействия с клиентами. Однако многие существующие решения не полностью удовлетворяют запросы пользователей, такие как сложная навигация, недостаточная персонализация или ограниченные возможности аналитики. Это подчеркивает актуальность разработки более функциональных и удобных систем.

Целью данной курсовой работы является создание концепции и прототипа программного обеспечения для интернет-магазина электроники. Разрабатываемое ПО позволит пользователям легко находить и приобретать электронику, обеспечит гибкие функции фильтрации и сортировки, а также предоставит владельцам магазина инструменты для оптимизации управления и повышения качества обслуживания клиентов. Для достижения поставленной цели необходимо решить следующие задачи:

Подведем основные цели на данный курсовой проект:

– провести анализ предметной области и изучить существующие аналоги;

– описать основные сущности проектируемой базы данных и связи между ними;

– разработать структуру базы данных, включая индексы для оптимизации поиска;

– создать триггеры и хранимые процедуры для обеспечения корректной работы системы;

– разработать программное обеспечение, использующее данные, которые хранятся в созданной базе данных.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Современные платформы для интернет-магазинов являются популярным инструментом для поиска и покупки товаров. Среди множества приложений и сервисов можно выделить один из наиболее популярных веб-сайтов ­– *Onliner*.

## Обзор веб-сайта *Onliner*

*Onliner*.*by* (стилизуется как *Onlíner*) – белорусский сайт, включающий в себя СМИ, маркетплейс товаров и услуг, а также форум. На рисунке 1.1 представлен графический интерфейс главной страницы веб-сайта:

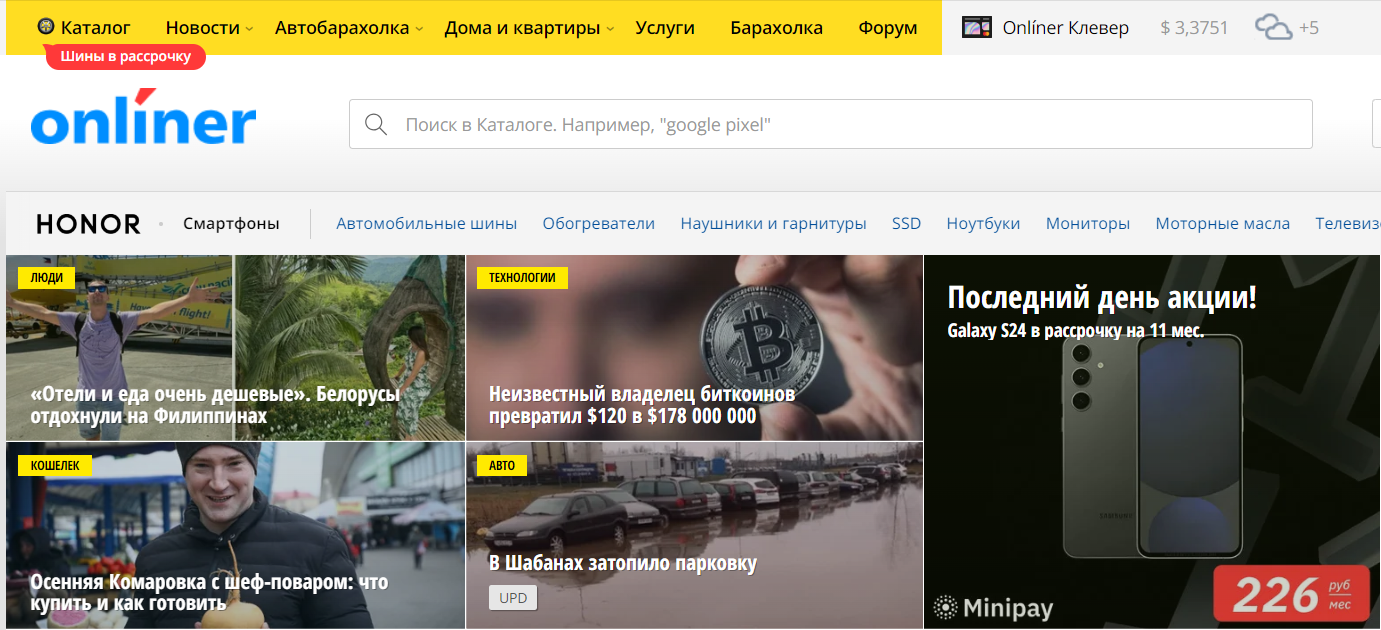


Рисунок 1.1 – Графический интерфейс главной страницы *Onliner*

*Onliner* предлагает пользователям мощный инструмент поиска и фильтрации, который позволяет находить товары по множеству параметров, таких как производитель, год выпуска, технические характеристики, рейтинг и другие.

Пример фильтров и параметров для поиска представлен на рисунке 1.2:

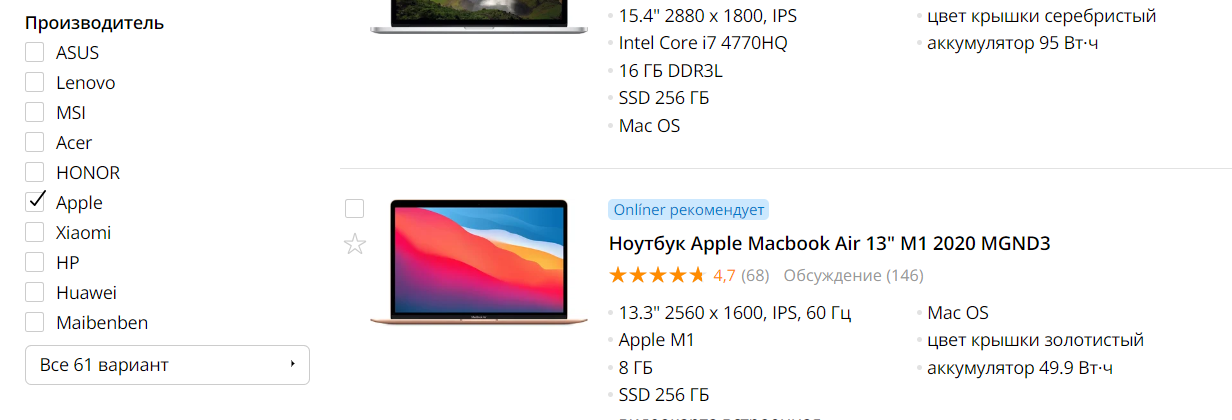


Рисунок 1.2 – Фрагмент списка ноутбуков на *Onliner*

Каждая единица контента на *Onliner* представлена в виде подробной страницы, где пользователи могут найти следующую информацию: краткое описание характеристик и составляющих товара, изображения, отзывы, стоимость, список продавцов.

Эта структура страницы позволяет быстро получить представление о любом товаре. Карточка также часто включает трейлеры, галерею изображений и ссылки на похожие фильмы.

На рисунке 1.3 продемонстрирован пример такой страницы для ноутбука:

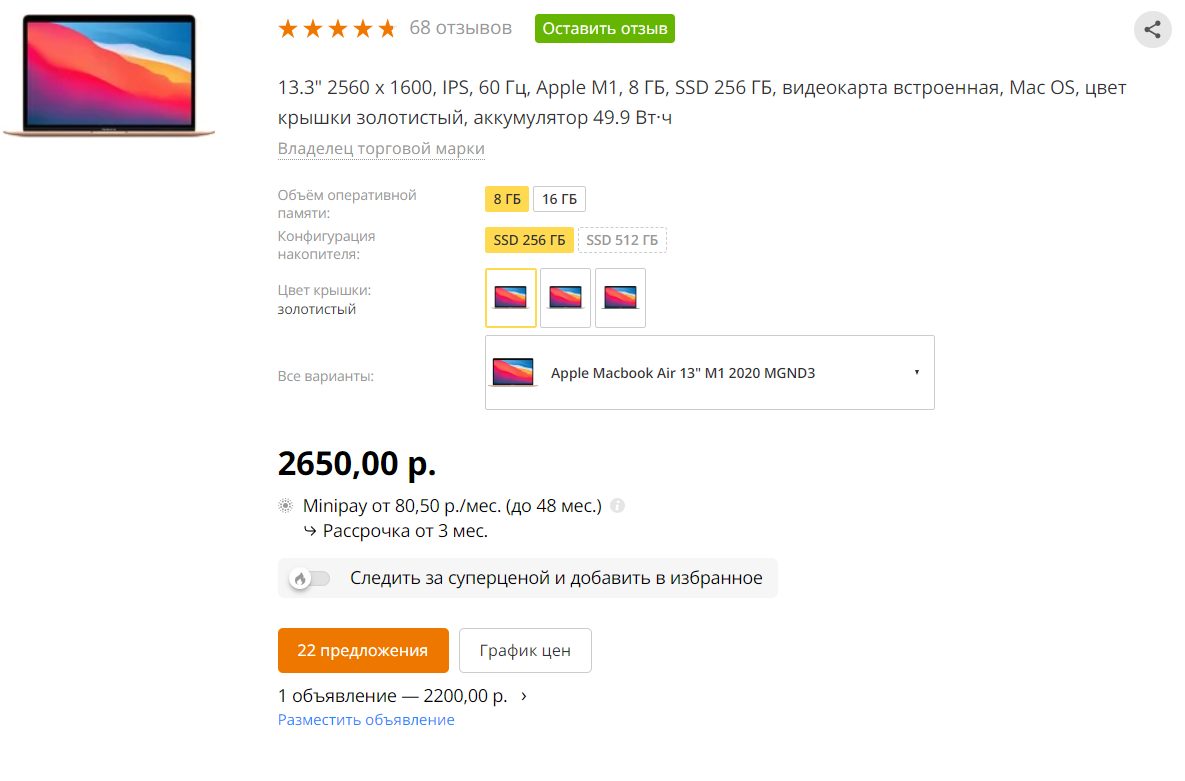


Рисунок 1.3 – Карточка с описанием выбранного ноутбука

*Onliner* предоставляет пользователям возможность оценивать товары и оставлять отзывы.

На рисунке 1.4 представлена страница с отзывами пользователей на выбранный ноутбук:

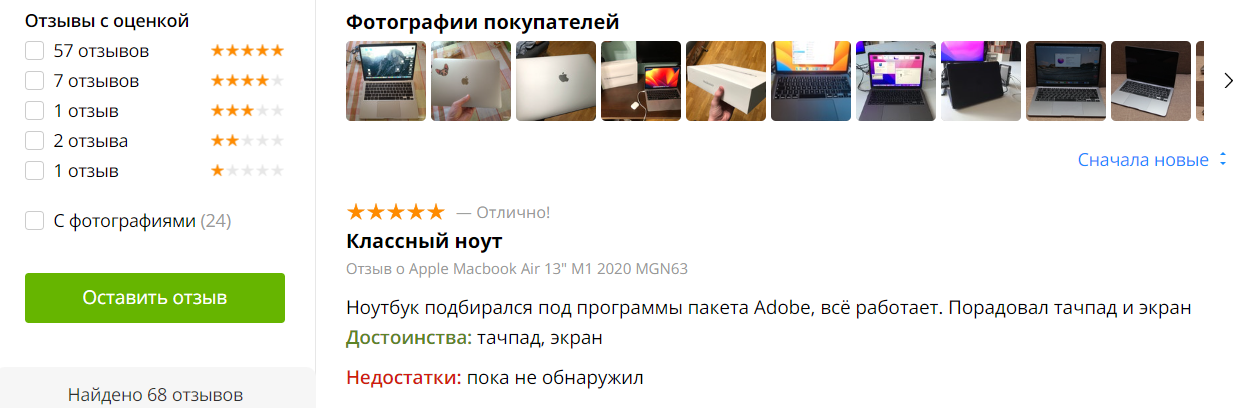


Рисунок 1.4 – Страница с отзывами на выбранный ноутбук

*Onliner* обладает рядом преимуществ, которые сделали его одной из ведущих платформ в своей области:

1 Широкий охват информации. Представлены товары различных видов и категорий.

2 Пользовательские рейтинги и отзывы.

3 Обширная база данных.

Однако *Onliner* также имеет некоторые недостатки:

1 Помимо маркетплейса представлено множество других возможностей, ненужных для интернет-магазина.

2 Отсутствие специализации на конкретных товарах.

*Onliner* является одним из самых известных маркетплейсов на белорусском рынке. Однако для создания полнофункциональной интерент магазина может потребоваться более персонализированное решение, которое позволит пользователям эффективнее совершать поиск и покупки необходимой техники.

# 2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## 2.1 Функциональные требования к разрабатываемой базе данных

На основе анализа существующих аналогов выдвинем функциональные требования к разрабатываемому программному средству:

1 Авторизация пользователя. Пользователи должны иметь возможность авторизоваться в системе, используя свои учетные данные, и получать доступ к расширенным функциям в зависимости от своей роли.

2 Управление пользователями (CRUD). Система должна позволять создание, чтение, обновление и удаление учетных записей пользователей, обеспечивая гибкость и контроль для администраторов.

3 Ролевая система. Необходимо реализовать систему ролей, где предусмотрены роли администратора и обычного пользователя. Разграничение прав доступа позволит обеспечить безопасность и функциональность системы.

4 История действий пользователей. Система должна вести журнал всех действий пользователей для отслеживания истории взаимодействий. Пользователи смогут просматривать свои действия, а администраторы — действия всех пользователей.

После авторизации обычные пользователи должны иметь следующие возможности:

1 Возможность находить товары по категориям.

2 Получение информации о товаре.

3 Возможность оставить отзыв.

4 Создания заказа.

Пользователи с ролью администратора будут иметь доступ к расширенным функциям:

1 Доступ к функциям пользователей. Возможность выполнять все те же действия, что и обычные пользователи.

2 Просмотр заказов.

3 Добавления новых сотрудников.

Эти требования обеспечат надежное управление данными, удобство использования для пользователей и функциональные возможности для администраторов при поддержке контроля доступа и безопасности системы.

## 2.2 Анализ существующих подходов к разработке баз данных

В современной разработке баз данных выделяют два основных типа: SQL и NoSQL. Их отличие связано не только с использованием SQL-запросов, но и с принципами организации данных. SQL базы данных, основанные на реляционной модели, представляют данные в виде таблиц с четко определенными связями, что обеспечивает целостность данных и удобную обработку с помощью SQL. NoSQL базы данных, напротив, не придерживаются реляционной теории и могут хранить данные в различных форматах (документы, графы, колонки и др.), что позволяет гибко адаптироваться к задачам масштабируемых систем.

Для лучшего понимания структуры и подвидов этих баз данных обратимся к рисунку 2.1, который иллюстрирует основные категории SQL и NoSQL баз.

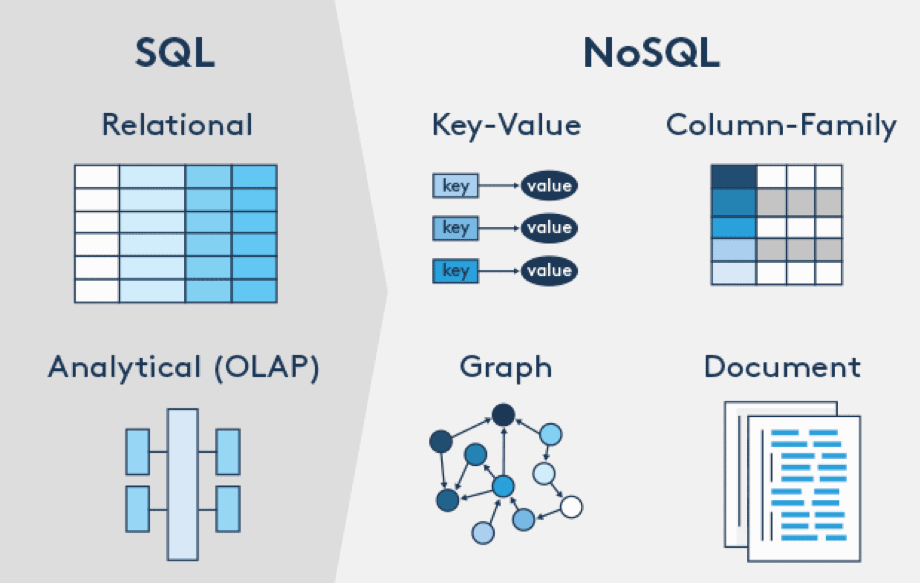


Рисунок 2.1 – Основные виды SQL и NoSQL баз данных

Далее рассмотрим более подробно основные отличия реляционных и нереляционных баз данных.

**2.2.1 Реляционные базы данных**

Реляционные базы данных – это базы данных, которые используют табличную структуру для хранения и обработки данных. В них информация организована в виде таблиц, состоящих из строк и столбцов, где каждая строка представляет собой запись с уникальным ключом. Столбцы содержат атрибуты, описывающие характеристики записей. Связи между таблицами устанавливаются с помощью ключей, что поддерживает целостность и согласованность данных.

Для управления реляционными базами данных применяется стандартный язык SQL, позволяющий выполнять различные операции с данными: выборку, добавление, обновление, удаление, объединение, агрегацию и другие. SQL также поддерживает транзакции – неделимые блоки операций, которые подчиняются принципам ACID:

1 Атомарность (Atomicity). Транзакция выполняется как единое целое, «всё или ничего». Это значит, что либо все операции в транзакции выполняются успешно, либо ни одна из них не выполняется. Если какая-то операция в транзакции не удается, происходит откат, и база данных возвращается к состоянию до начала транзакции. Например, при переводе денег между счетами атомарность гарантирует, что деньги спишутся с одного счета только в том случае, если они зачислятся на другой.

2 Согласованность (Consistency). После завершения транзакции база данных должна перейти из одного согласованного состояния в другое. Согласованность подразумевает соблюдение всех установленных правил и ограничений. Например, если для определённого столбца указано ограничение на уникальные значения, транзакция, нарушающая это правило, не будет завершена.

3 Изолированность (Isolation). Параллельные транзакции не должны влиять друг на друга, обеспечивая независимость их выполнения. Изолированность гарантирует, что одна транзакция не сможет «увидеть» изменения, вносимые другой, пока та не завершится. Существуют разные уровни изолированности, такие как «чтение с повторной фиксацией» и «серийная изолированность», которые регулируют степень изолированности транзакций.

4 Долговечность (Durability). После успешного завершения транзакции ее результаты сохраняются в базе данных даже в случае сбоя системы. Это означает, что изменения в данных, внесенные транзакцией, сохраняются в долгосрочной перспективе и остаются доступными после завершения транзакции.

**2.2.3 Нереляционные базы данных**

NoSQL – это обширная категория систем управления базами данных, которые не соответствуют традиционной реляционной модели. Основные типы NoSQL баз данных включают:

– документо-ориентированные базы данных;

– базы данных ключ-значение;

– графовые базы данных;

– колоночные базы данных.

Далее данные типы будут рассмотрены более подробно.

Документо-ориентированные базы данных хранят информацию в виде документов, которые имеют свою структуру и могут включать различные типы данных. Документы группируются в коллекции, которые могут иметь разные схемы. Для работы с такими базами данных используются специализированные языки запросов, позволяющие обращаться к данным по их атрибутам.

MongoDB – одна из самых известных и мощных документо-ориентированных баз данных, поддерживающая различные форматы документов, такие как JSON, BSON и XML. MongoDB характеризуется высокой производительностью, масштабируемостью и гибкостью, а также предоставляет множество функций для работы с данными, таких как агрегация, индексация, шардирование и репликация.

Далее перейдем к базам данных ключ-значение. Эти базы данных хранят данные в виде пар ключ-значение, где ключ служит уникальным идентификатором, а значение может быть любым типом данных. Базы данных ключ-значение обеспечивают быстрый доступ к данным по ключу, однако не поддерживают сложные запросы и связи между данными.

Redis – популярная база данных, которая хранит данные в оперативной памяти и поддерживает различные типы значений, такие как строки, списки и множества. Redis также предлагает функции, такие как транзакции, репликация и кэширование.

Графовые базы данных хранят данные в виде узлов и ребер, представляющих сущности и связи между ними. Они подходят для моделирования сложных сетей и отношений, таких как социальные сети и рекомендательные системы. Для работы с графовыми базами данных используются специализированные языки запросов, которые позволяют искать пути и паттерны в графах.

Графовые базы данных могут быть разделены на два основных типа: базы данных свойственных графов, где узлы и ребра имеют атрибуты, и базы данных знаний, где узлы и ребра имеют семантические метки. Примеры баз данных свойственных графов: Neo4j и ArangoDB.

Колоночные базы данных представляют собой один из типов NoSQL систем управления базами данных, которые хранят данные в виде колонок, а не строк. Это подход позволяет эффективно организовывать и обрабатывать большие объемы данных, особенно в сценариях, требующих высокой производительности при выполнении аналитических запросов. Колоночные базы данных оптимизированы для работы с большими наборами данных и часто используются в аналитических приложениях, бизнес-аналитике и хранилищах данных.

Основные характеристики колоночных баз данных включают в себя:

1 Хранение по колонкам. Данные хранятся в виде отдельных колонок, что позволяет эффективно считывать только необходимые данные без необходимости загружать всю строку. Это значительно повышает производительность при выполнении агрегирующих запросов и выборок по определённым колонкам.

2 Оптимизация для чтения. Колоночные базы данных идеально подходят для сценариев, где требуется высокая скорость обработки запросов и анализ данных. Это достигается благодаря тому, что данные одной колонки хранятся вместе, что позволяет лучше использовать кэш и сократить время доступа к данным.

3 Сжатие данных. Поскольку данные в колонках часто имеют одинаковый тип, это позволяет применять более эффективные алгоритмы сжатия, что приводит к уменьшению объема хранимых данных и экономии дискового пространства.

4 Гибкость схемы. Колоночные базы данных могут поддерживать динамические схемы, что позволяет добавлять новые колонки без необходимости изменения всей структуры базы данных.

Apache Cassandra – это распределенная колоночная база данных, предназначенная для обработки больших объемов данных. Cassandra обеспечивает высокую доступность и отказоустойчивость благодаря дублированию данных на нескольких узлах. Она идеально подходит для работы с веб-приложениями и IoT-решениями.

Таким образом, NoSQL базы данных предлагают гибкость, масштабируемость и производительность, что делает их подходящими для работы с большими объемами данных и динамически меняющимися схемами.

## 2.3 Выбор базы данных для разработки предметной области

Исходя из анализа подходов, представленных в предыдущем разделе, можно заключить, что предметная область для магазина электроники будет реализована с использованием реляционных баз данных. Это объясняется тем, что в данной области четко определены атрибуты, сущности и связи, которые удобно отображать в виде таблиц и ключей в рамках реляционной модели.

Для обеспечения корректной работы системы необходимо гарантировать целостность и согласованность данных, что особенно важно для избежания ошибок при создании заказов.

В качестве базы данных для разработки была выбрана PostgreSQL.

Преимущества PostgreSQL заключаются в следующем:

1 Эффективный механизм обработки запросов. PostgreSQL использует продвинутые алгоритмы, что позволяет оптимально работать с данными.

2 Поддержка параллелизма. Это означает, что несколько запросов могут выполняться одновременно, что существенно увеличивает общую производительность.

3 Масштабируемость. PostgreSQL может масштабироваться как вертикально, увеличивая ресурсы на одном сервере, так и горизонтально, распределяя нагрузку между несколькими серверами.

Система также подходит для различных типов приложений, включая:

1 Корпоративные решения. PostgreSQL идеальна для крупных организаций, которым необходимы высокая производительность и надежность.

2 Веб-приложения. Эта СУБД часто используется для разработки масштабируемых и надежных веб-приложений.

3 Интернет-магазины. PostgreSQL может эффективно обрабатывать транзакции и управлять товарными запасами и клиентами.

PostgreSQL обеспечивает высокий уровень безопасности данных, предлагая разнообразные методы аутентификации, такие как пароли, сертификаты и двухфакторная аутентификация. Она также поддерживает ролевые и объектно-ориентированные схемы авторизации, что позволяет администраторам контролировать доступ пользователей к данным и функциям. База данных включает методы шифрования, как для хранения, так и для передачи данных, защищая информацию от несанкционированного доступа.

Таким образом, PostgreSQL была выбрана благодаря своей высокой производительности, масштабируемости и богатым возможностям безопасности.

# 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

## 3.1 Концептуальная модель базы данных

Проектирование базы данных было начато с создания концептуальной модели. Она включает в себя основные сущности и связи, не содержит подробных сведений об атрибутах и часто используется на этапе планирования. Концептуальная модель представлена на рисунке 3.1.

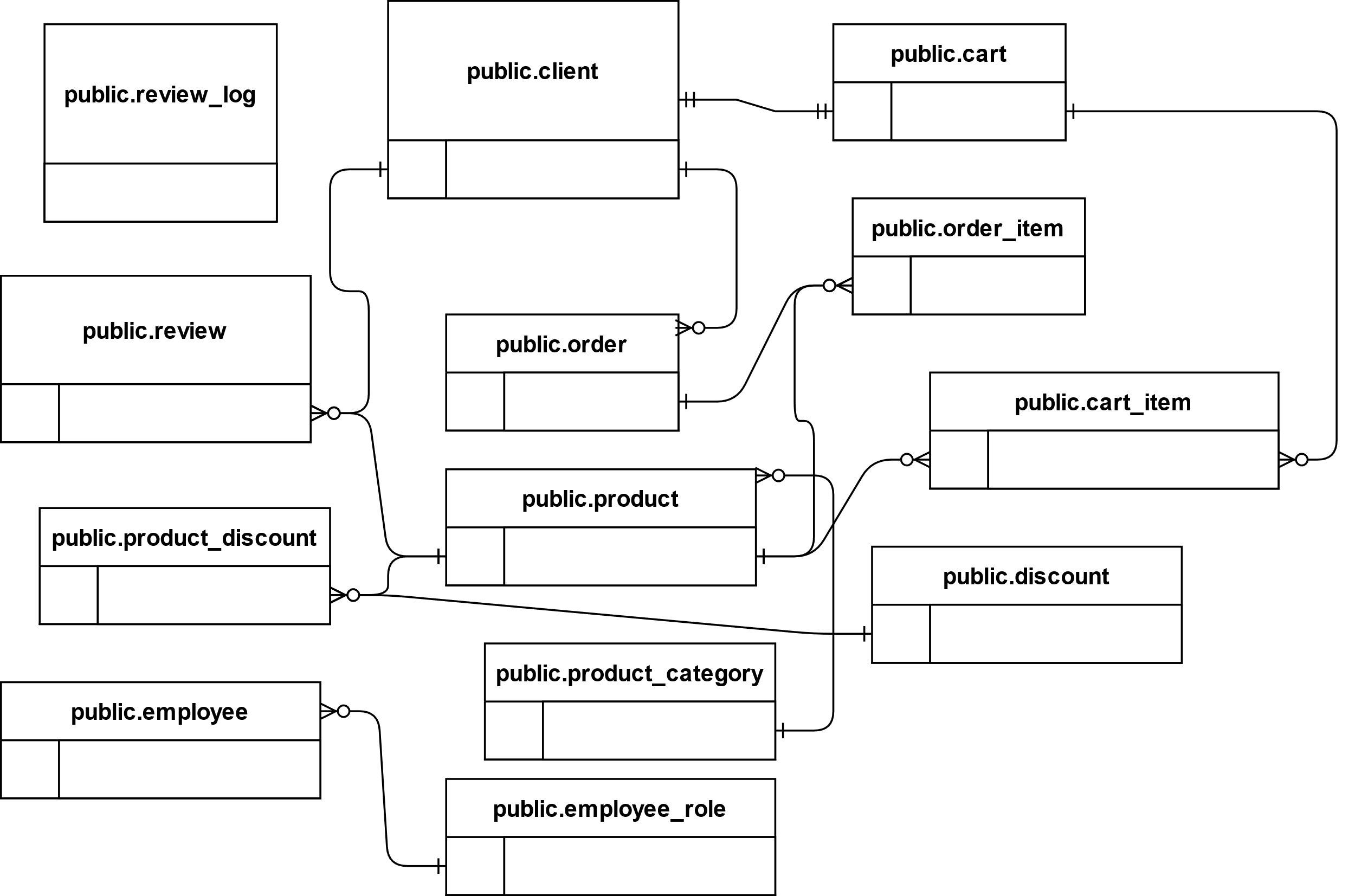


Рисунок 3.1 – Концептуальная модель базы данных

Описываемая база данных включает в себя следующие сущности:

1. **Товар**:

Таблица: product.

1. **Категория товара**:

Таблица: product\_category.

1. **Сотрудник**:

Таблица: employee.

1. **Роль сотрудника**:

Таблица: employee\_role.

1. **Клиент**:

Таблица: client.

1. **Заказ**:

Таблица: order.

1. **Элемент заказа**:

Таблица: order\_item.

1. **Корзина**:

Таблица: cart.

1. **Элемент корзины**:

Таблица: cart\_item.

1. **Отзыв**:

Таблица: review.

1. **Акция**:

Таблица: discount.

Таким образом, в данном разделе представлена концептуальная модель базы данных, служащая основой для построения логической модели.

## 3.2 Логическая модель базы данных

На основе концептуальной модели была спроектирована логическая модель базы данных, которая включает в себя все сущности, атрибуты, ключи и связи, которые представляют бизнес-информацию и определяют бизнес-правила. Логическая модель представлена на рисунке 3.2:

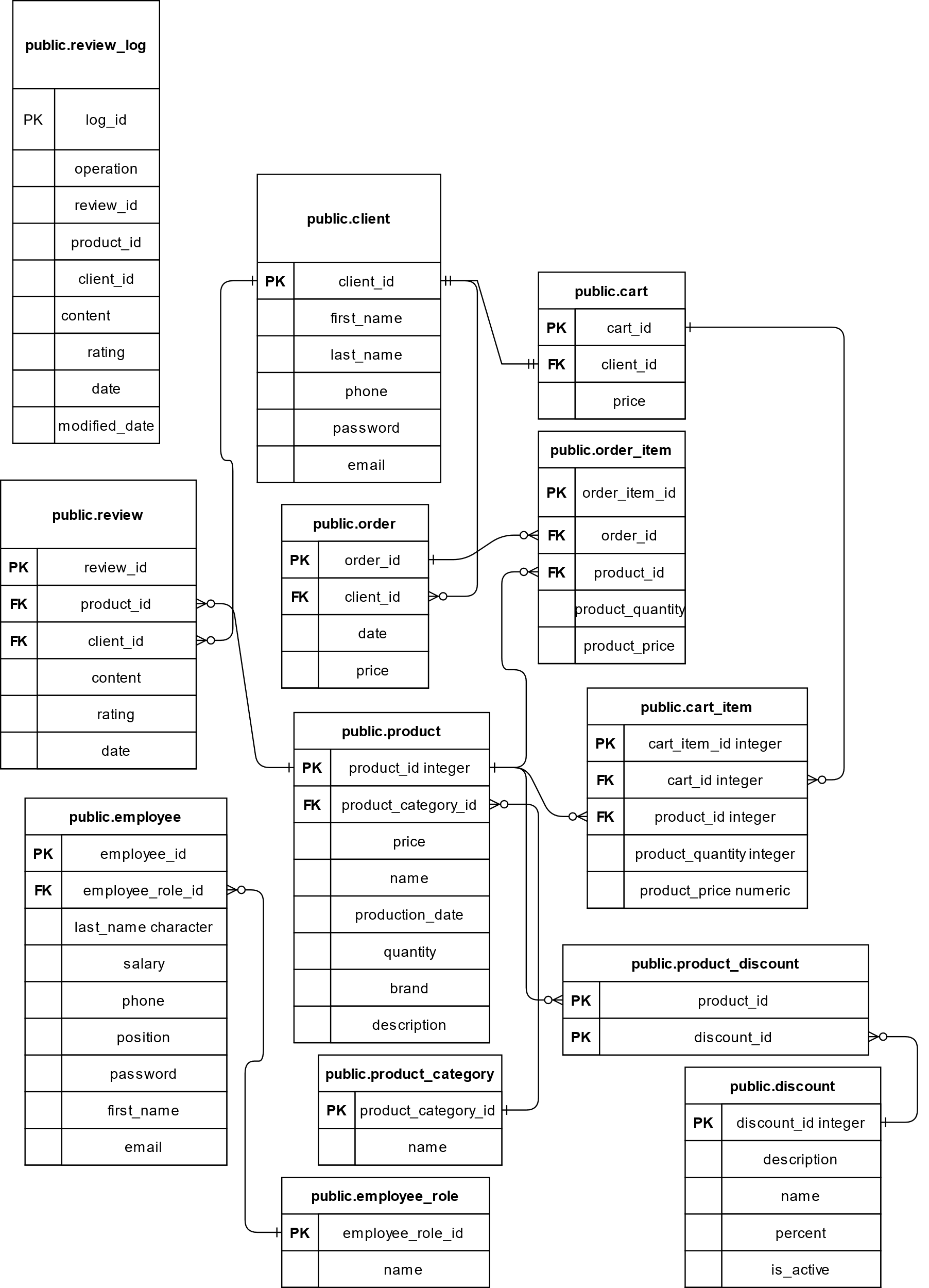


Рисунок 3.2 – Логическая модель базы данных

Подробное описание сущностей:

1. **Товар**:

Таблица: product.

Поля:

– price (цена);

– name (название);

– production\_date (дата производства);

– quantity (количество);

– brand (бренд);

– description (описание).

Связи:

– один ко многим с order\_item;

– один ко многим с review;

– один ко многим с cart\_item;

– многие к одному с product\_category;

– многие ко многим с discount.

1. **Категория товара**:

Таблица: product\_category.

Поля:

– name (имя).

Связи:

– один ко многим с product.

1. **Сотрудник**:

Таблица: employee.

Поля:

– first\_name (имя);

– last\_name (фамилия);

– salary (зарплата);

– phone (телефон);

– position (должность);

– password (пароль);

– email (электронная почта).

Связи:

– многик к одному с employee\_role.

1. **Роль сотрудника**:

Таблица: employee\_role.

Поля:

– name (имя).

Связи:

– один ко многим с employee.

1. **Клиент**:

Таблица: client.

Поля:

– name (имя);

– phone (телефон);

– password (пароль);

– email (электронная почта);

Связи:

– один ко многим с review;

– один к одному с cart;

– один ко многим с order.

1. **Заказ**:

Таблица: order.

Поля:

– date (дата);

– price (стоимость).

Связи:

– многие к одному с client;

– один ко многим с order\_item.

1. **Элемент заказа**:

Таблица: order\_item.

Поля:

– product\_quantity (количество товара);

– product\_price (стоимость товара).

Связи:

– многие к одному с order;

– многие к одному с product.

1. **Корзина**:

Таблица: cart.

Поля:

– price (стоимость).

Связи:

– один к одному с client;

– один ко многим с cart\_item.

1. **Элемент корзины**:

Таблица: cart\_item.

Поля:

– product\_quantity (количество товара);

– product\_price (стоимость товара).

Связи:

– многие к одному с cart;

– многие к одному с product.

1. **Отзыв**:

Таблица: review.

Поля:

– content (содержимое отзыва);

– rating (рейтинг, оценка);

– date (дата отзыва).

Связи:

– многие к одному с client;

– многие к одному с product.

1. **Акция**:

Таблица: discount.

Поля:

– description (описание);

– name (имя);

– percent (процент);

– is\_active (активна или нет).

Связи:

– многие ко многим с product.

Таким образом, в данном разделе была представлена и описана логическая модель базы данных.

## 3.3 Физическая модель базы данных

На основе логической модели была разработана физическая модель базы данных, которая включает в себя все необходимые таблицы, столбцы, связи, свойства базы данных для физической реализации баз данных. Производительность базы данных, стратегии индексации, физическое хранилище являются важными параметрами физической модели. Физическая модель представлена на рисунке 3.3.

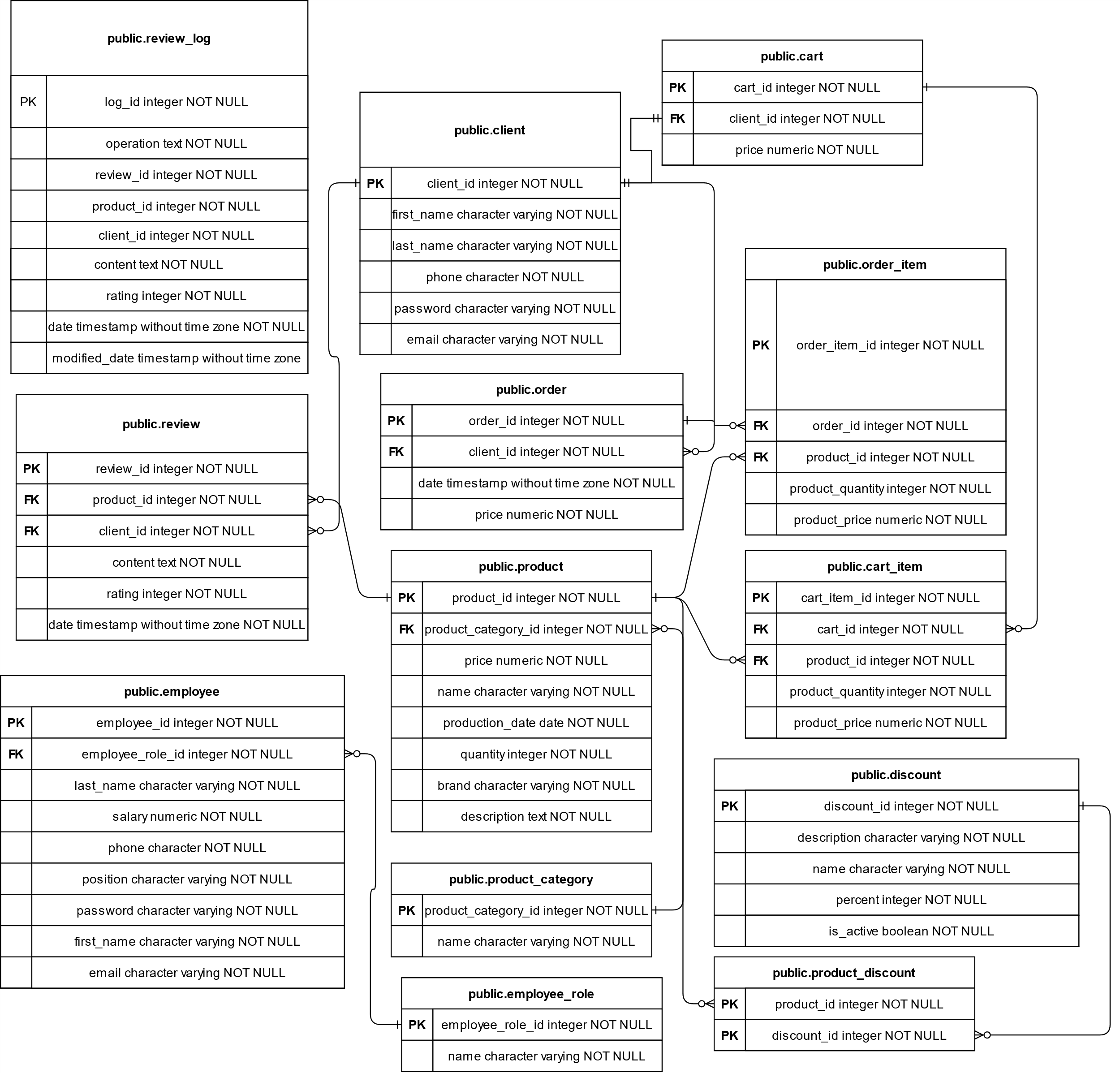


Рисунок 3.3 – Физическая модель базы данных

В данном разделе была рассмотрена физическая модель базы данных.

# 4 РАЗРАБОТКА БАЗЫ ДАННЫХ

## 4.1 Создание исходных таблиц, индексов и ограничений

## 4.2 Создание хранимых процедур

## 4.3 Реализация триггеров

# ЗАКЛЮЧЕНИЕ

Результатом работы данного курсового проекта является база данных для интернет-магазина электроники.

Для разработки была выбрана реляционная СУБД PostgreSQL, обладающая своими преимуществами перед известными конкурентами. Как итог можно сказать, что цели данной курсовой работы достигнуты.

# СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода программы

SET TIMEZONE TO 'UTC';

SET check\_function\_bodies = false;

SET search\_path = pg\_catalog;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.cart.cart\_id

CREATE SEQUENCE public.cart\_cart\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.cart\_cart\_id\_seq OWNER TO postgres;

CREATE TABLE public.cart (

cart\_id integer DEFAULT nextval('public.cart\_cart\_id\_seq'::regclass) NOT NULL,

client\_id integer NOT NULL,

price numeric(7,2) NOT NULL

);

ALTER TABLE public.cart OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.cart\_item.cart\_item\_id

CREATE SEQUENCE public.cart\_item\_cart\_item\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.cart\_item\_cart\_item\_id\_seq OWNER TO postgres;

CREATE TABLE public.cart\_item (

cart\_item\_id integer DEFAULT nextval('public.cart\_item\_cart\_item\_id\_seq'::regclass) NOT NULL,

cart\_id integer NOT NULL,

product\_id integer NOT NULL,

product\_quantity integer NOT NULL,

product\_price numeric(7,2) NOT NULL

);

ALTER TABLE public.cart\_item OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.client.client\_id

CREATE SEQUENCE public.client\_client\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.client\_client\_id\_seq OWNER TO postgres;

CREATE TABLE public.client (

client\_id integer DEFAULT nextval('public.client\_client\_id\_seq'::regclass) NOT NULL,

first\_name character varying(30) NOT NULL,

last\_name character varying(30) NOT NULL,

phone character(13) NOT NULL,

password character varying(20) NOT NULL,

email character varying(30) NOT NULL

);

ALTER TABLE public.client OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.discount.discount\_id

CREATE SEQUENCE public.discount\_discount\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.discount\_discount\_id\_seq OWNER TO postgres;

CREATE TABLE public.discount (

discount\_id integer DEFAULT nextval('public.discount\_discount\_id\_seq'::regclass) NOT NULL,

description character varying(50) NOT NULL,

name character varying(30) NOT NULL,

percent integer NOT NULL,

is\_active boolean NOT NULL

);

ALTER TABLE public.discount OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.employee.employee\_id

CREATE SEQUENCE public.employee\_employee\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.employee\_employee\_id\_seq OWNER TO postgres;

CREATE TABLE public.employee (

employee\_id integer DEFAULT nextval('public.employee\_employee\_id\_seq'::regclass) NOT NULL,

employee\_role\_id integer NOT NULL,

first\_name character varying(30) NOT NULL,

last\_name character varying(30) NOT NULL,

salary numeric(7,2) NOT NULL,

phone character(13) NOT NULL,

"position" character varying(30) NOT NULL,

password character varying(20) NOT NULL,

email character varying(30) NOT NULL

);

ALTER TABLE public.employee OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.employee\_role.employee\_role\_id

CREATE SEQUENCE public.employee\_role\_employee\_role\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.employee\_role\_employee\_role\_id\_seq OWNER TO postgres;

CREATE TABLE public.employee\_role (

employee\_role\_id integer DEFAULT nextval('public.employee\_role\_employee\_role\_id\_seq'::regclass) NOT NULL,

name character varying(30) NOT NULL

);

ALTER TABLE public.employee\_role OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.order\_item.order\_item\_id

CREATE SEQUENCE public.order\_item\_order\_item\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.order\_item\_order\_item\_id\_seq OWNER TO postgres;

CREATE TABLE public.order\_item (

order\_item\_id integer DEFAULT nextval('public.order\_item\_order\_item\_id\_seq'::regclass) NOT NULL,

order\_id integer NOT NULL,

product\_id integer NOT NULL,

product\_quantity integer NOT NULL,

product\_price numeric(7,2) NOT NULL

);

ALTER TABLE public.order\_item OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public."order".order\_id

CREATE SEQUENCE public.order\_order\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.order\_order\_id\_seq OWNER TO postgres;

CREATE TABLE public."order" (

order\_id integer DEFAULT nextval('public.order\_order\_id\_seq'::regclass) NOT NULL,

client\_id integer NOT NULL,

date timestamp without time zone DEFAULT CURRENT\_TIMESTAMP NOT NULL,

price numeric(7,2) NOT NULL

);

ALTER TABLE public."order" OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.product\_category.product\_category\_id

CREATE SEQUENCE public.product\_category\_product\_category\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.product\_category\_product\_category\_id\_seq OWNER TO postgres;

CREATE TABLE public.product\_category (

product\_category\_id integer DEFAULT nextval('public.product\_category\_product\_category\_id\_seq'::regclass) NOT NULL,

name character varying(30) NOT NULL

);

ALTER TABLE public.product\_category OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.product.product\_id

CREATE SEQUENCE public.product\_product\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.product\_product\_id\_seq OWNER TO postgres;

CREATE TABLE public.product (

product\_id integer DEFAULT nextval('public.product\_product\_id\_seq'::regclass) NOT NULL,

product\_category\_id integer NOT NULL,

price numeric(7,2) NOT NULL,

name character varying(30) NOT NULL,

production\_date date NOT NULL,

quantity integer NOT NULL,

brand character varying(30) NOT NULL,

description text NOT NULL

);

ALTER TABLE public.product OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.review\_log.log\_id

CREATE SEQUENCE public.review\_log\_log\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.review\_log\_log\_id\_seq OWNER TO postgres;

CREATE TABLE public.review\_log (

log\_id integer DEFAULT nextval('public.review\_log\_log\_id\_seq'::regclass) NOT NULL,

operation text NOT NULL,

review\_id integer NOT NULL,

product\_id integer NOT NULL,

client\_id integer NOT NULL,

content text NOT NULL,

rating integer NOT NULL,

date timestamp without time zone NOT NULL,

modified\_date timestamp without time zone DEFAULT CURRENT\_TIMESTAMP

);

ALTER TABLE public.review\_log OWNER TO postgres;

-- DEPCY: This SEQUENCE is a dependency of COLUMN: public.review.review\_id

CREATE SEQUENCE public.review\_review\_id\_seq

AS integer

START WITH 1

INCREMENT BY 1

NO MAXVALUE

NO MINVALUE

CACHE 1;

ALTER SEQUENCE public.review\_review\_id\_seq OWNER TO postgres;

CREATE TABLE public.review (

review\_id integer DEFAULT nextval('public.review\_review\_id\_seq'::regclass) NOT NULL,

product\_id integer NOT NULL,

client\_id integer NOT NULL,

content text NOT NULL,

rating integer NOT NULL,

date timestamp without time zone DEFAULT CURRENT\_TIMESTAMP NOT NULL

);

ALTER TABLE public.review OWNER TO postgres;

CREATE TABLE public.product\_discount (

product\_id integer NOT NULL,

discount\_id integer NOT NULL

);

ALTER TABLE public.product\_discount OWNER TO postgres;

CREATE TABLE public.total\_discount\_percent (

"coalesce" bigint

);

ALTER TABLE public.total\_discount\_percent OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.copy\_cart\_items\_to\_order\_items() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE cur\_cart\_id INT;

BEGIN

SELECT cart\_id INTO cur\_cart\_id

FROM cart

WHERE client\_id = NEW.client\_id;

INSERT INTO order\_item (

order\_id,

product\_id,

product\_quantity,

product\_price

)

SELECT NEW.order\_id,

product\_id,

product\_quantity,

product\_price

FROM cart\_item

WHERE cart\_id = cur\_cart\_id;

DELETE FROM cart\_item

WHERE cart\_id = cur\_cart\_id;

UPDATE cart

SET price = 0

WHERE client\_id = NEW.client\_id;

RETURN NULL;

END;

$$;

ALTER FUNCTION public.copy\_cart\_items\_to\_order\_items() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.create\_cart\_for\_new\_client() RETURNS trigger

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO cart (client\_id, price)

VALUES (NEW.client\_id, 0.0);

RETURN NULL;

END;

$$;

ALTER FUNCTION public.create\_cart\_for\_new\_client() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_cart\_item\_by\_id(p\_cart\_item\_id integer) RETURNS TABLE(cart\_item\_id integer, cart\_id integer, product\_id integer, product\_quantity integer, product\_price numeric, client\_id integer, price numeric, product\_category\_id integer, name character varying, production\_date date, quantity integer, brand character varying, description text)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT ci.cart\_item\_id,

ci.cart\_id,

ci.product\_id,

ci.product\_quantity,

ci.product\_price,

c.client\_id,

c.price,

p.product\_category\_id,

p.name,

p.production\_date,

p.quantity,

p.brand,

p.description

FROM cart\_item ci

JOIN cart c ON ci.cart\_id = c.cart\_id

JOIN product p ON ci.product\_id = p.product\_id

WHERE ci.cart\_item\_id = p\_cart\_item\_id;

END;

$$;

ALTER FUNCTION public.get\_cart\_item\_by\_id(p\_cart\_item\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_cart\_items\_by\_cart\_id(p\_cart\_id integer) RETURNS TABLE(cart\_item\_id integer, cart\_id integer, product\_id integer, product\_quantity integer, product\_price numeric, client\_id integer, price numeric, product\_category\_id integer, name character varying, production\_date date, quantity integer, brand character varying, description text)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT ci.cart\_item\_id,

ci.cart\_id,

ci.product\_id,

ci.product\_quantity,

ci.product\_price,

c.client\_id,

c.price,

p.product\_category\_id,

p.name,

p.production\_date,

p.quantity,

p.brand,

p.description

FROM cart\_item ci

JOIN cart c ON ci.cart\_id = c.cart\_id

JOIN product p ON ci.product\_id = p.product\_id

WHERE ci.cart\_id = p\_cart\_id;

END;

$$;

ALTER FUNCTION public.get\_cart\_items\_by\_cart\_id(p\_cart\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_discount\_by\_id(p\_discount\_id integer) RETURNS TABLE(discount\_id integer, description character varying, name character varying, percent integer, is\_active boolean)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT d.discount\_id,

d.description,

d.name,

d.percent,

d.is\_active

FROM discount d

WHERE d.discount\_id = p\_discount\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_discount\_by\_id(p\_discount\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_discounts() RETURNS TABLE(discount\_id integer, description character varying, name character varying, percent integer, is\_active boolean)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT d.discount\_id,

d.description,

d.name,

d.percent,

d.is\_active

FROM discount d;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_discounts() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_employee\_with\_role(p\_email text, p\_password text) RETURNS TABLE(employee\_id integer, employee\_role\_id integer, first\_name character varying, last\_name character varying, salary numeric, phone character, "position" character varying, password character varying, email character varying, name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT e.employee\_id,

e.employee\_role\_id,

e.first\_name,

e.last\_name,

e.salary,

e.phone,

e.position,

e.password,

e.email,

r.name

FROM employee AS e

INNER JOIN employee\_role AS r ON e.employee\_role\_id = r.employee\_role\_id

WHERE e.email = p\_email

AND e.password = p\_password;

END;

$$;

ALTER FUNCTION public.get\_employee\_with\_role(p\_email text, p\_password text) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_orders() RETURNS TABLE(order\_id integer, client\_id integer, date timestamp without time zone, price numeric, first\_name character varying, last\_name character varying, phone character, email character varying, password character varying, product\_id integer, quantity integer, product\_price numeric, name character varying, product\_category\_id integer, product\_category\_name character varying, brand character varying, description text)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT o.order\_id,

o.client\_id,

o.date,

o.price,

c.first\_name,

c.last\_name,

c.phone,

c.email,

c.password,

oi.product\_id,

oi.product\_quantity,

oi.product\_price,

p.name,

p.product\_category\_id,

pc.name,

p.brand,

p.description

FROM "order" o

JOIN "client" c ON o.client\_id = c.client\_id

JOIN "order\_item" oi ON o.order\_id = oi.order\_id

JOIN "product" p ON oi.product\_id = p.product\_id

JOIN "product\_category" pc ON p.product\_category\_id = pc.product\_category\_id;

END;

$$;

ALTER FUNCTION public.get\_orders() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_product\_with\_category\_by\_id(p\_product\_id integer) RETURNS TABLE(product\_id integer, product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text, category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT p.product\_id,

p.product\_category\_id,

p.price,

p.name,

p.production\_date,

p.quantity,

p.brand,

p.description,

pc.name AS category\_name

FROM product p

INNER JOIN product\_category pc ON pc.product\_category\_id = p.product\_category\_id

WHERE p.product\_id = p\_product\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_product\_with\_category\_by\_id(p\_product\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_products\_with\_categories() RETURNS TABLE(product\_id integer, product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text, category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT p.product\_id,

p.product\_category\_id,

p.price,

p.name,

p.production\_date,

p.quantity,

p.brand,

p.description,

pc.name AS category\_name

FROM product p

INNER JOIN product\_category pc ON pc.product\_category\_id = p.product\_category\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_products\_with\_categories() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_products\_with\_categories\_by\_category\_id(p\_product\_category\_id integer) RETURNS TABLE(product\_id integer, product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text, category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT p.product\_id,

p.product\_category\_id,

p.price,

p.name,

p.production\_date,

p.quantity,

p.brand,

p.description,

pc.name AS category\_name

FROM product p

INNER JOIN product\_category pc ON pc.product\_category\_id = p.product\_category\_id

WHERE p.product\_category\_id = p\_product\_category\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_products\_with\_categories\_by\_category\_id(p\_product\_category\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_review\_by\_id(p\_review\_id integer) RETURNS TABLE(review\_id integer, product\_id integer, client\_id integer, content text, rating integer, date timestamp without time zone, client\_first\_name character varying, client\_last\_name character varying, client\_phone character, client\_email character varying, product\_name character varying, product\_price numeric, product\_category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT r.review\_id,

r.product\_id,

r.client\_id,

r.content,

r.rating,

r.date,

c.first\_name AS client\_first\_name,

c.last\_name AS client\_last\_name,

c.phone AS client\_phone,

c.email AS client\_email,

p.name AS product\_name,

p.price AS product\_price,

pc.name AS product\_category\_name

FROM review r

INNER JOIN client c ON r.client\_id = c.client\_id

INNER JOIN product p ON r.product\_id = p.product\_id

INNER JOIN product\_category pc ON p.product\_category\_id = pc.product\_category\_id

WHERE r.review\_id = p\_review\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_review\_by\_id(p\_review\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_reviews() RETURNS TABLE(review\_id integer, product\_id integer, client\_id integer, content text, rating integer, date timestamp without time zone, client\_first\_name character varying, client\_last\_name character varying, client\_phone character, client\_email character varying, product\_name character varying, product\_price numeric, product\_category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT r.review\_id,

r.product\_id,

r.client\_id,

r.content,

r.rating,

r.date,

c.first\_name AS client\_first\_name,

c.last\_name AS client\_last\_name,

c.phone AS client\_phone,

c.email AS client\_email,

p.name AS product\_name,

p.price AS product\_price,

pc.name AS product\_category\_name

FROM review r

INNER JOIN client c ON r.client\_id = c.client\_id

INNER JOIN product p ON r.product\_id = p.product\_id

INNER JOIN product\_category pc ON p.product\_category\_id = pc.product\_category\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_reviews() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.get\_reviews\_by\_product\_id(p\_product\_id integer) RETURNS TABLE(review\_id integer, product\_id integer, client\_id integer, content text, rating integer, date timestamp without time zone, client\_first\_name character varying, client\_last\_name character varying, client\_phone character, client\_email character varying, product\_name character varying, product\_price numeric, product\_category\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN RETURN QUERY

SELECT r.review\_id,

r.product\_id,

r.client\_id,

r.content,

r.rating,

r.date,

c.first\_name AS client\_first\_name,

c.last\_name AS client\_last\_name,

c.phone AS client\_phone,

c.email AS client\_email,

p.name AS product\_name,

p.price AS product\_price,

pc.name AS product\_category\_name

FROM review r

INNER JOIN client c ON r.client\_id = c.client\_id

INNER JOIN product p ON r.product\_id = p.product\_id

INNER JOIN product\_category pc ON p.product\_category\_id = pc.product\_category\_id

WHERE p.product\_id = p\_product\_id;

RETURN;

END;

$$;

ALTER FUNCTION public.get\_reviews\_by\_product\_id(p\_product\_id integer) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.insert\_discount(p\_description character varying, p\_name character varying, p\_percent integer, p\_is\_active boolean) RETURNS integer

LANGUAGE plpgsql

AS $$

DECLARE r\_discount\_id INT;

BEGIN

INSERT INTO discount (description, name, percent, is\_active)

VALUES (p\_description, p\_name, p\_percent, p\_is\_active)

RETURNING discount\_id INTO r\_discount\_id;

RETURN r\_discount\_id;

END;

$$;

ALTER FUNCTION public.insert\_discount(p\_description character varying, p\_name character varying, p\_percent integer, p\_is\_active boolean) OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.review\_trigger\_function() RETURNS trigger

LANGUAGE plpgsql

AS $$ BEGIN IF (TG\_OP = 'INSERT') THEN

INSERT INTO review\_log (

operation,

review\_id,

product\_id,

client\_id,

content,

rating,

"date"

)

VALUES (

'INSERT',

NEW.review\_id,

NEW.product\_id,

NEW.client\_id,

NEW.content,

NEW.rating,

NEW.date

);

ELSIF (TG\_OP = 'UPDATE') THEN

INSERT INTO review\_log (

operation,

review\_id,

product\_id,

client\_id,

content,

rating,

"date"

)

VALUES (

'UPDATE',

NEW.review\_id,

NEW.product\_id,

NEW.client\_id,

NEW.content,

NEW.rating,

NEW.date

);

ELSIF (TG\_OP = 'DELETE') THEN

INSERT INTO review\_log (

operation,

review\_id,

product\_id,

client\_id,

content,

rating,

"date"

)

VALUES (

'DELETE',

OLD.review\_id,

OLD.product\_id,

OLD.client\_id,

OLD.content,

OLD.rating,

OLD.date

);

END IF;

RETURN NEW;

END;

$$;

ALTER FUNCTION public.review\_trigger\_function() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.update\_cart\_price() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE total\_discount\_percent numeric;

BEGIN PERFORM update\_product\_quantity(OLD, NEW, TG\_OP);

IF TG\_OP = 'DELETE' THEN

UPDATE cart

SET price = price - OLD.product\_price \* OLD.product\_quantity

WHERE cart\_id = OLD.cart\_id;

ELSIF TG\_OP = 'INSERT' THEN

SELECT COALESCE(SUM(d.percent), 0) INTO total\_discount\_percent

FROM discount d

JOIN product\_discount pd ON pd.discount\_id = d.discount\_id

WHERE pd.product\_id = NEW.product\_id

AND d.is\_active = TRUE;

total\_discount\_percent = total\_discount\_percent % 50;

NEW.product\_price = NEW.product\_price \* (1 - total\_discount\_percent / 100);

UPDATE cart\_item

SET product\_price = NEW.product\_price

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

UPDATE cart

SET price = price + NEW.product\_price \* NEW.product\_quantity

WHERE cart\_id = NEW.cart\_id;

ELSIF TG\_OP = 'UPDATE' THEN

UPDATE cart

SET price = price + (NEW.product\_quantity - OLD.product\_quantity) \* NEW.product\_price

WHERE cart\_id = NEW.cart\_id;

END IF;

RETURN NULL;

END;

$$;

ALTER FUNCTION public.update\_cart\_price() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.update\_product\_quantity() RETURNS trigger

LANGUAGE plpgsql

AS $$ BEGIN IF TG\_OP = 'INSERT' THEN

UPDATE product

SET quantity = quantity - NEW.product\_quantity

WHERE product\_id = NEW.product\_id;

ELSIF TG\_OP = 'DELETE' THEN

UPDATE product

SET quantity = quantity + OLD.product\_quantity

WHERE product\_id = OLD.product\_id;

END IF;

RETURN NULL;

END;

$$;

ALTER FUNCTION public.update\_product\_quantity() OWNER TO postgres;

CREATE OR REPLACE FUNCTION public.update\_product\_quantity(old anyelement, new anyelement, tg\_op text) RETURNS void

LANGUAGE plpgsql

AS $$ BEGIN IF TG\_OP = 'INSERT' THEN

UPDATE product

SET quantity = quantity - NEW.product\_quantity

WHERE product\_id = NEW.product\_id;

ELSIF TG\_OP = 'DELETE' THEN

UPDATE product

SET quantity = quantity + OLD.product\_quantity

WHERE product\_id = OLD.product\_id;

ELSIF TG\_OP = 'UPDATE' THEN

UPDATE product

SET quantity = quantity - (NEW.product\_quantity - OLD.product\_quantity)

WHERE product\_id = OLD.product\_id;

END IF;

RETURN;

END;

$$;

ALTER FUNCTION public.update\_product\_quantity(old anyelement, new anyelement, tg\_op text) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.delete\_cart\_item\_by\_id(p\_cart\_item\_id integer)

LANGUAGE plpgsql

AS $$ BEGIN

DELETE FROM cart\_item

WHERE cart\_item\_id = p\_cart\_item\_id;

END;

$$;

ALTER PROCEDURE public.delete\_cart\_item\_by\_id(p\_cart\_item\_id integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.delete\_category\_by\_id(category\_id integer)

LANGUAGE plpgsql

AS $$ BEGIN

DELETE FROM product\_category

WHERE product\_category\_id = category\_id;

END;

$$;

ALTER PROCEDURE public.delete\_category\_by\_id(category\_id integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.delete\_discount\_by\_id(p\_discount\_id integer)

LANGUAGE plpgsql

AS $$ BEGIN

DELETE FROM discount

WHERE discount\_id = p\_discount\_id;

END;

$$;

ALTER PROCEDURE public.delete\_discount\_by\_id(p\_discount\_id integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.delete\_product\_by\_id(p\_product\_id integer)

LANGUAGE plpgsql

AS $$ BEGIN

DELETE FROM product

WHERE product\_id = p\_product\_id;

END;

$$;

ALTER PROCEDURE public.delete\_product\_by\_id(p\_product\_id integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.delete\_review(p\_review\_id integer)

LANGUAGE plpgsql

AS $$ BEGIN

DELETE FROM review

WHERE review\_id = p\_review\_id;

END;

$$;

ALTER PROCEDURE public.delete\_review(p\_review\_id integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_cart\_item(p\_cart\_id integer, p\_product\_id integer, p\_product\_quantity integer, p\_product\_price numeric)

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO cart\_item (

cart\_id,

product\_id,

product\_quantity,

product\_price

)

VALUES (

p\_cart\_id,

p\_product\_id,

p\_product\_quantity,

p\_product\_price

);

END;

$$;

ALTER PROCEDURE public.insert\_cart\_item(p\_cart\_id integer, p\_product\_id integer, p\_product\_quantity integer, p\_product\_price numeric) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_category(p\_name character varying)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO product\_category (name)

VALUES (p\_name);

END;

$$;

ALTER PROCEDURE public.insert\_category(p\_name character varying) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_client(p\_first\_name character varying, p\_last\_name character varying, p\_phone character, p\_password character varying, p\_email character varying)

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO "client" (

"first\_name",

"last\_name",

"phone",

"password",

"email"

)

VALUES (

p\_first\_name,

p\_last\_name,

p\_phone,

p\_password,

p\_email

);

END;

$$;

ALTER PROCEDURE public.insert\_client(p\_first\_name character varying, p\_last\_name character varying, p\_phone character, p\_password character varying, p\_email character varying) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_employee(p\_employee\_role\_id integer, p\_first\_name character varying, p\_last\_name character varying, p\_salary numeric, p\_phone character, p\_position character varying, p\_password character varying, p\_email character varying)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO "employee" (

"employee\_role\_id",

"first\_name",

"last\_name",

"salary",

"phone",

"position",

"password",

"email"

)

VALUES (

p\_employee\_role\_id,

p\_first\_name,

p\_last\_name,

p\_salary,

p\_phone,

p\_position,

p\_password,

p\_email

);

END;

$$;

ALTER PROCEDURE public.insert\_employee(p\_employee\_role\_id integer, p\_first\_name character varying, p\_last\_name character varying, p\_salary numeric, p\_phone character, p\_position character varying, p\_password character varying, p\_email character varying) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_order(p\_client\_id integer, p\_price numeric)

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO "order" ("client\_id", "price")

VALUES (p\_client\_id, p\_price);

END;

$$;

ALTER PROCEDURE public.insert\_order(p\_client\_id integer, p\_price numeric) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_product(product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text)

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO product (

product\_category\_id,

price,

name,

production\_date,

quantity,

brand,

description

)

VALUES (

product\_category\_id,

price,

name,

production\_date,

quantity,

brand,

description

);

END;

$$;

ALTER PROCEDURE public.insert\_product(product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_product\_discounts(p\_discount\_id integer, p\_product\_ids integer[])

LANGUAGE plpgsql

AS $$

DECLARE

i INT;

BEGIN

FOR i IN 1..array\_length(p\_product\_ids, 1)

LOOP

INSERT INTO product\_discount (product\_id, discount\_id)

VALUES (p\_product\_ids[i], p\_discount\_id);

END LOOP;

END;

$$;

ALTER PROCEDURE public.insert\_product\_discounts(p\_discount\_id integer, p\_product\_ids integer[]) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert\_review(p\_product\_id integer, p\_client\_id integer, p\_content text, p\_rating integer)

LANGUAGE plpgsql

AS $$ BEGIN

INSERT INTO review (product\_id, client\_id, content, rating)

VALUES (p\_product\_id, p\_client\_id, p\_content, p\_rating);

END;

$$;

ALTER PROCEDURE public.insert\_review(p\_product\_id integer, p\_client\_id integer, p\_content text, p\_rating integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.update\_cart\_item\_product\_quantity(p\_cart\_item\_id integer, p\_product\_quantity integer)

LANGUAGE plpgsql

AS $$ BEGIN

UPDATE cart\_item

SET product\_quantity = p\_product\_quantity

WHERE cart\_item\_id = p\_cart\_item\_id;

END;

$$;

ALTER PROCEDURE public.update\_cart\_item\_product\_quantity(p\_cart\_item\_id integer, p\_product\_quantity integer) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.update\_category(id integer, new\_name character varying)

LANGUAGE plpgsql

AS $$ BEGIN

UPDATE product\_category

SET name = new\_name

WHERE product\_category\_id = id;

END;

$$;

ALTER PROCEDURE public.update\_category(id integer, new\_name character varying) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.update\_discount(p\_discount\_id integer, p\_description character varying, p\_name character varying, p\_percent integer, p\_is\_active boolean)

LANGUAGE plpgsql

AS $$ BEGIN

UPDATE discount

SET description = p\_description,

name = p\_name,

percent = p\_percent,

is\_active = p\_is\_active

WHERE discount\_id = p\_discount\_id;

END;

$$;

ALTER PROCEDURE public.update\_discount(p\_discount\_id integer, p\_description character varying, p\_name character varying, p\_percent integer, p\_is\_active boolean) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.update\_product(p\_product\_id integer, product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text)

LANGUAGE plpgsql

AS $$ BEGIN

UPDATE product

SET product\_category\_id = update\_product.product\_category\_id,

price = update\_product.price,

name = update\_product.name,

production\_date = update\_product.production\_date,

quantity = update\_product.quantity,

brand = update\_product.brand,

description = update\_product.description

WHERE product\_id = p\_product\_id;

END;

$$;

ALTER PROCEDURE public.update\_product(p\_product\_id integer, product\_category\_id integer, price numeric, name character varying, production\_date date, quantity integer, brand character varying, description text) OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.update\_review(p\_review\_id integer, p\_content text, p\_rating integer)

LANGUAGE plpgsql

AS $$ BEGIN

UPDATE review

SET content = p\_content,

rating = p\_rating

WHERE review\_id = p\_review\_id;

END;

$$;

ALTER PROCEDURE public.update\_review(p\_review\_id integer, p\_content text, p\_rating integer) OWNER TO postgres;

CREATE INDEX idx\_cart\_client\_id ON public.cart USING btree (client\_id);

CREATE INDEX idx\_cart\_item\_cart\_id ON public.cart\_item USING btree (cart\_id);

CREATE INDEX idx\_cart\_item\_product\_id ON public.cart\_item USING btree (product\_id);

CREATE INDEX idx\_client\_email ON public.client USING btree (email);

CREATE INDEX idx\_discount\_percent ON public.discount USING btree (percent);

CREATE INDEX idx\_employee\_email ON public.employee USING btree (email);

CREATE INDEX idx\_employee\_employee\_role\_id ON public.employee USING btree (employee\_role\_id);

CREATE INDEX idx\_order\_client\_id ON public."order" USING btree (client\_id);

CREATE INDEX idx\_order\_date ON public."order" USING btree (date);

CREATE INDEX idx\_order\_item\_order\_id ON public.order\_item USING btree (order\_id);

CREATE INDEX idx\_order\_item\_product\_id ON public.order\_item USING btree (product\_id);

CREATE INDEX idx\_product\_brand ON public.product USING btree (brand);

CREATE INDEX idx\_product\_name ON public.product USING btree (name);

CREATE INDEX idx\_product\_product\_category\_id ON public.product USING btree (product\_category\_id);

CREATE INDEX idx\_product\_category\_name ON public.product\_category USING btree (name);

CREATE INDEX idx\_product\_discount\_discount\_id ON public.product\_discount USING btree (discount\_id);

CREATE INDEX idx\_product\_discount\_product\_id ON public.product\_discount USING btree (product\_id);

CREATE INDEX idx\_review\_client\_id ON public.review USING btree (client\_id);

CREATE INDEX idx\_review\_product\_id ON public.review USING btree (product\_id);

-- DEPCY: This CONSTRAINT is a dependency of CONSTRAINT: public.cart.cart\_client\_id\_fkey

ALTER TABLE public.client

ADD CONSTRAINT client\_pkey PRIMARY KEY (client\_id);

ALTER TABLE public.cart

ADD CONSTRAINT cart\_client\_id\_fkey FOREIGN KEY (client\_id) REFERENCES public.client(client\_id) ON DELETE CASCADE;

ALTER TABLE public.cart

ADD CONSTRAINT cart\_client\_id\_key UNIQUE (client\_id);

ALTER TABLE public.cart

ADD CONSTRAINT cart\_pkey PRIMARY KEY (cart\_id);

ALTER TABLE public.cart

ADD CONSTRAINT non\_negative\_price CHECK ((price >= (0)::numeric));

ALTER TABLE public.cart\_item

ADD CONSTRAINT cart\_item\_cart\_id\_fkey FOREIGN KEY (cart\_id) REFERENCES public.cart(cart\_id) ON DELETE CASCADE;

ALTER TABLE public.cart\_item

ADD CONSTRAINT cart\_item\_pkey PRIMARY KEY (cart\_item\_id);

-- DEPCY: This CONSTRAINT is a dependency of CONSTRAINT: public.cart\_item.cart\_item\_product\_id\_fkey

ALTER TABLE public.product

ADD CONSTRAINT product\_pkey PRIMARY KEY (product\_id);

ALTER TABLE public.cart\_item

ADD CONSTRAINT cart\_item\_product\_id\_fkey FOREIGN KEY (product\_id) REFERENCES public.product(product\_id) ON DELETE RESTRICT;

ALTER TABLE public.cart\_item

ADD CONSTRAINT positive\_product\_price CHECK ((product\_price > (0)::numeric));

ALTER TABLE public.cart\_item

ADD CONSTRAINT positive\_product\_quantity CHECK ((product\_quantity > 0));

ALTER TABLE public.client

ADD CONSTRAINT client\_email\_key UNIQUE (email);

ALTER TABLE public.discount

ADD CONSTRAINT discount\_percent\_value CHECK (((percent > 0) AND (percent < 100)));

ALTER TABLE public.discount

ADD CONSTRAINT discount\_pkey PRIMARY KEY (discount\_id);

-- DEPCY: This CONSTRAINT is a dependency of CONSTRAINT: public.employee.employee\_employee\_role\_id\_fkey

ALTER TABLE public.employee\_role

ADD CONSTRAINT employee\_role\_pkey PRIMARY KEY (employee\_role\_id);

ALTER TABLE public.employee

ADD CONSTRAINT employee\_employee\_role\_id\_fkey FOREIGN KEY (employee\_role\_id) REFERENCES public.employee\_role(employee\_role\_id) ON DELETE RESTRICT;

ALTER TABLE public.employee

ADD CONSTRAINT employee\_pkey PRIMARY KEY (employee\_id);

ALTER TABLE public.employee

ADD CONSTRAINT positive\_salary CHECK ((salary > (0)::numeric));

ALTER TABLE public."order"

ADD CONSTRAINT order\_client\_id\_fkey FOREIGN KEY (client\_id) REFERENCES public.client(client\_id) ON DELETE CASCADE;

ALTER TABLE public."order"

ADD CONSTRAINT order\_pkey PRIMARY KEY (order\_id);

ALTER TABLE public."order"

ADD CONSTRAINT positive\_price CHECK ((price > (0)::numeric));

ALTER TABLE public.order\_item

ADD CONSTRAINT order\_item\_order\_id\_fkey FOREIGN KEY (order\_id) REFERENCES public."order"(order\_id) ON DELETE CASCADE;

ALTER TABLE public.order\_item

ADD CONSTRAINT order\_item\_pkey PRIMARY KEY (order\_item\_id);

ALTER TABLE public.order\_item

ADD CONSTRAINT order\_item\_product\_id\_fkey FOREIGN KEY (product\_id) REFERENCES public.product(product\_id) ON DELETE RESTRICT;

ALTER TABLE public.order\_item

ADD CONSTRAINT positive\_product\_price CHECK ((product\_price > (0)::numeric));

ALTER TABLE public.order\_item

ADD CONSTRAINT positive\_product\_quantity CHECK ((product\_quantity > 0));

ALTER TABLE public.product

ADD CONSTRAINT non\_negative\_quantity CHECK ((quantity >= 0));

ALTER TABLE public.product

ADD CONSTRAINT positive\_price CHECK ((price > (0)::numeric));

-- DEPCY: This CONSTRAINT is a dependency of CONSTRAINT: public.product.product\_product\_category\_id\_fkey

ALTER TABLE public.product\_category

ADD CONSTRAINT product\_category\_pkey PRIMARY KEY (product\_category\_id);

ALTER TABLE public.product

ADD CONSTRAINT product\_product\_category\_id\_fkey FOREIGN KEY (product\_category\_id) REFERENCES public.product\_category(product\_category\_id) ON DELETE RESTRICT;

ALTER TABLE public.product\_discount

ADD CONSTRAINT product\_discount\_discount\_id\_fkey FOREIGN KEY (discount\_id) REFERENCES public.discount(discount\_id) ON DELETE CASCADE;

ALTER TABLE public.product\_discount

ADD CONSTRAINT product\_discount\_pkey PRIMARY KEY (product\_id, discount\_id);

ALTER TABLE public.product\_discount

ADD CONSTRAINT product\_discount\_product\_id\_fkey FOREIGN KEY (product\_id) REFERENCES public.product(product\_id) ON DELETE CASCADE;

ALTER TABLE public.review

ADD CONSTRAINT one\_to\_five\_rating\_range CHECK (((rating > 0) AND (rating < 6)));

ALTER TABLE public.review

ADD CONSTRAINT review\_client\_id\_fkey FOREIGN KEY (client\_id) REFERENCES public.client(client\_id) ON DELETE CASCADE;

ALTER TABLE public.review

ADD CONSTRAINT review\_pkey PRIMARY KEY (review\_id);

ALTER TABLE public.review

ADD CONSTRAINT review\_product\_id\_fkey FOREIGN KEY (product\_id) REFERENCES public.product(product\_id) ON DELETE CASCADE;

ALTER TABLE public.review\_log

ADD CONSTRAINT review\_log\_pkey PRIMARY KEY (log\_id);

CREATE VIEW public.vw\_employee\_with\_role AS

SELECT e.employee\_id,

e.first\_name,

e.last\_name,

e.salary,

e.phone,

e."position",

e.password,

e.email,

r.name

FROM (public.employee e

JOIN public.employee\_role r ON ((e.employee\_role\_id = r.employee\_role\_id)));

ALTER VIEW public.vw\_employee\_with\_role OWNER TO postgres;

CREATE TRIGGER after\_delete\_cart\_item\_trigger

AFTER DELETE ON public.cart\_item

FOR EACH ROW

EXECUTE PROCEDURE public.update\_cart\_price();

CREATE TRIGGER after\_insert\_cart\_item\_trigger

AFTER INSERT ON public.cart\_item

FOR EACH ROW

EXECUTE PROCEDURE public.update\_cart\_price();

CREATE TRIGGER after\_update\_cart\_item\_trigger

AFTER UPDATE ON public.cart\_item

FOR EACH ROW

EXECUTE PROCEDURE public.update\_cart\_price();

CREATE TRIGGER insert\_cart\_for\_new\_client

AFTER INSERT ON public.client

FOR EACH ROW

EXECUTE PROCEDURE public.create\_cart\_for\_new\_client();

CREATE TRIGGER copy\_cart\_items\_to\_order\_items\_trigger

AFTER INSERT ON public."order"

FOR EACH ROW

EXECUTE PROCEDURE public.copy\_cart\_items\_to\_order\_items();

CREATE TRIGGER review\_trigger

AFTER INSERT OR UPDATE OR DELETE ON public.review

FOR EACH ROW

EXECUTE PROCEDURE public.review\_trigger\_function();

ALTER SEQUENCE public.cart\_cart\_id\_seq

OWNED BY public.cart.cart\_id;

ALTER SEQUENCE public.cart\_item\_cart\_item\_id\_seq

OWNED BY public.cart\_item.cart\_item\_id;

ALTER SEQUENCE public.client\_client\_id\_seq

OWNED BY public.client.client\_id;

ALTER SEQUENCE public.discount\_discount\_id\_seq

OWNED BY public.discount.discount\_id;

ALTER SEQUENCE public.employee\_employee\_id\_seq

OWNED BY public.employee.employee\_id;

ALTER SEQUENCE public.employee\_role\_employee\_role\_id\_seq

OWNED BY public.employee\_role.employee\_role\_id;

ALTER SEQUENCE public.order\_item\_order\_item\_id\_seq

OWNED BY public.order\_item.order\_item\_id;

ALTER SEQUENCE public.order\_order\_id\_seq

OWNED BY public."order".order\_id;

ALTER SEQUENCE public.product\_category\_product\_category\_id\_seq

OWNED BY public.product\_category.product\_category\_id;

ALTER SEQUENCE public.product\_product\_id\_seq

OWNED BY public.product.product\_id;

ALTER SEQUENCE public.review\_log\_log\_id\_seq

OWNED BY public.review\_log.log\_id;

ALTER SEQUENCE public.review\_review\_id\_seq

OWNED BY public.review.review\_id;

ПРИЛОЖЕНИЕ Б  
(обязательное)  
Схема базы данных