

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  
\_\_\_\_\_. \_\_\_\_\_. 2023  
В. С. Плиска

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
к курсовому проекту  
на тему:  
**«БАЗА ДАННЫХ ДЛЯ ТОРГОВОЙ ПЛОЩАДКИ»**

БГУИР КП 1-40 04 01 011 ПЗ

Выполнил студент группы 053505  
Слуцкий Никита Сергеевич

\_\_\_\_\_  
(подпись студента)  
Курсовой проект представлен на  
проверку \_\_\_\_\_. \_\_\_\_\_. 2023  
\_\_\_\_\_  
(подпись студента)

Минск 2023

# СОДЕРЖАНИЕ

Введение.....	3
1 Обзор существующих аналогов.....	4
1.1 Интернет-портал «Каталог Онлайнер».....	4
1.2 Торговая площадка «Яндекс. Маркет».....	5
1.3 Интернет-магазин «Wildberries».....	6
1.4 Формирование требований.....	6
2 Формирование функциональных требований и выбор инструментов.....	8
2.1 Формирование функциональных требований.....	8
2.2 Выбор инструментов разработки.....	9
2.3 Обоснование выбора.....	13
3 Проектирование базы данных.....	14
3.1 Концептуальная ER диаграмма.....	15
3.2 IDEF1X диаграмма.....	15
3.3 Физическая схема базы данных.....	16
3.4 Нормализация базы данных.....	17
4 Разработка базы данных.....	18
4.1 Создание проекта в Supabase.....	19
4.2 Разработка DDL-скриптов для создания таблиц.....	20
4.3 Разграничение прав доступа.....	22
4.4 Создание некоторых процедур, функций и триггеров.....	23
4.5 Создание индексов.....	25
4.6 Тестирование базы данных.....	26
Заключение.....	27
Список используемой литературы.....	28
Приложение А (обязательное) Листинг кода.....	30
Приложение Б (обязательное) Концептуальная диаграмма.....	36
Приложение В (обязательное) Логическая модель.....	37
Приложение Г (обязательное) Физическая модель.....	38
Приложение Д (обязательное) Ведомость документов.....	39

## ВВЕДЕНИЕ

В настоящее время электронная коммерция стала неотъемлемой частью современного бизнеса, и интернет-магазины становятся все более важным каналом продаж для предприятий различных масштабов. С ростом популярности онлайн-торговли важно иметь эффективную и хорошо спроектированную базу данных, которая обеспечивает хранение, управление и доступ к информации о товарах, клиентах, заказах и других аспектах торговли в электронной среде.

Данный курсовой проект направлен на создание и исследование базы данных упрощенного интернет-магазина, которая превращает концепцию онлайн-торговли в управляемую и интуитивно понятную систему. В основу проектирования базы данных положены принципы гибкости, расширяемости, связанные с хранением и обработкой информации о продуктах, клиентах, заказах, оплате, доставке и других аспектах.

В качестве контекста в будущем может быть использовано предполагаемое приложение, которое будет реализовано на основе созданной базы данных, обеспечивая удобство для покупателей и эффективное управление информацией для администраторов и владельцев магазина.

Целями данного курсового проекта ставятся:

- разработать базу данных для упрощенного интернет-магазина товаров с использованием выбранных технологий;
- использовать не менее 25 сущностей в разрабатываемой базе данных;
- нормализовать базу данных до третьей нормальной формы;
- создать набор используемых триггеров, процедур;
- оформить пояснительную записку к курсовому проекту в соответствии со стандартом предприятия университета.

# 1 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

## 1.1 Интернет-портал «Каталог Онлайнер»

Onliner.by [1] – это белорусский сайт, который включает в себя СМИ, платформу для размещения товаров и услуг, а также форум.

«Каталог Онлайнер» – это раздел сайта, где представлены различные товары и услуги. Здесь можно найти информацию о ценах, скидках, а также воспользоваться услугой оплаты частями. Это удобный инструмент для выбора и покупки различных товаров с доставкой по Беларуси.

На рисунке 1.1 приведено изображение главной страницы каталога.

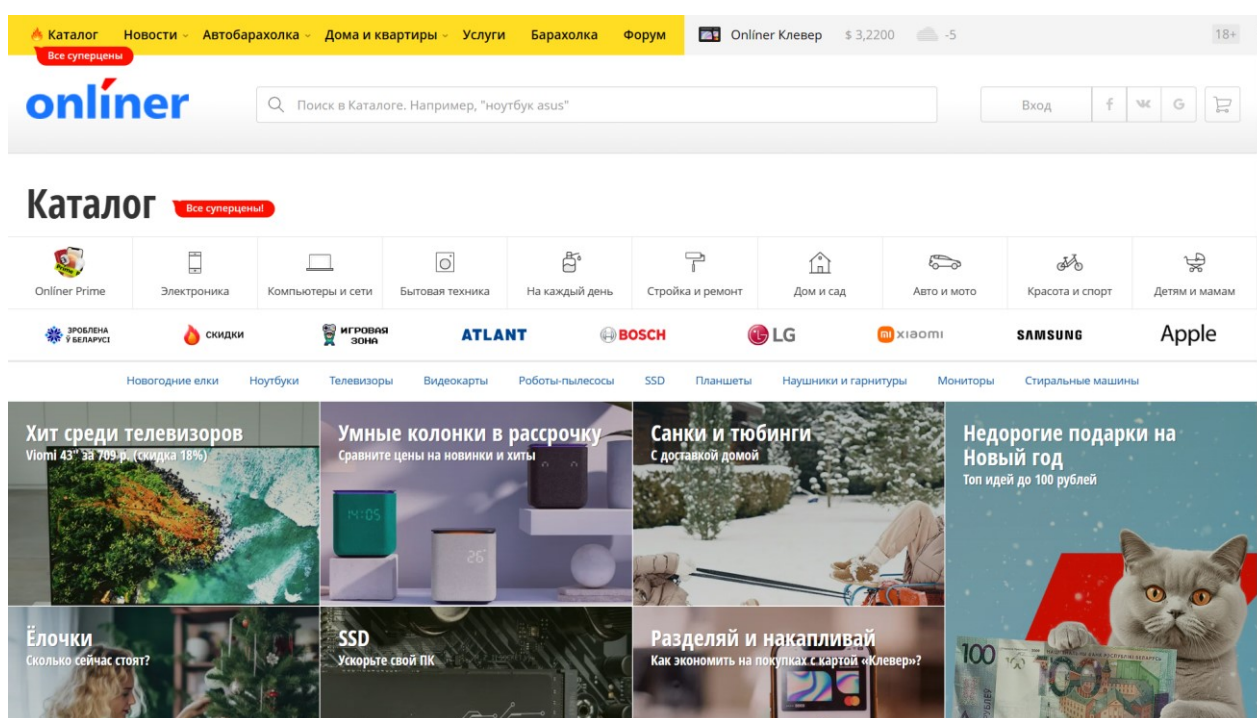


Рисунок 1.1 – Главная страница приложения

Сайт предлагает пользователям искать товары в разных категориях, сравнивать товары по характеристикам и ценам от разных магазинов, а также производить оплату и заказ непосредственно через сайт. В одном из разделов сайта можно размещать объявления по продаже единиц товара, бывших в употреблении.

Магазинам данный каталог предоставляет возможность размещать свои предложения на площадке, а также покупать места на показ рекламы.

Таким образом это полноценное и большое решение по решению задач продаж очень широкого спектра товаров через интернет. Сайт является одним из самых посещаемых ресурсов в Беларуси.

## 1.2 Торговая площадка «Яндекс. Маркет»

«Яндекс. Маркет» [2] – электронная торговая площадка, сервис для покупки товаров. Пользователь «Маркета» может просматривать и покупать товары из различных категорий, сравнивать их характеристики и цены, читать и оставлять отзывы и обзоры на товары, задавать вопросы другим посетителям сайта, магазинам и производителям. Сервис берет на себя хранение товаров, обработку и доставку заказов и общение с покупателями. Приложение «Яндекс. Маркет» доступно для iOS и Android.

На рисунке 1.2 приведено изображение главной страницы описываемого сайта.

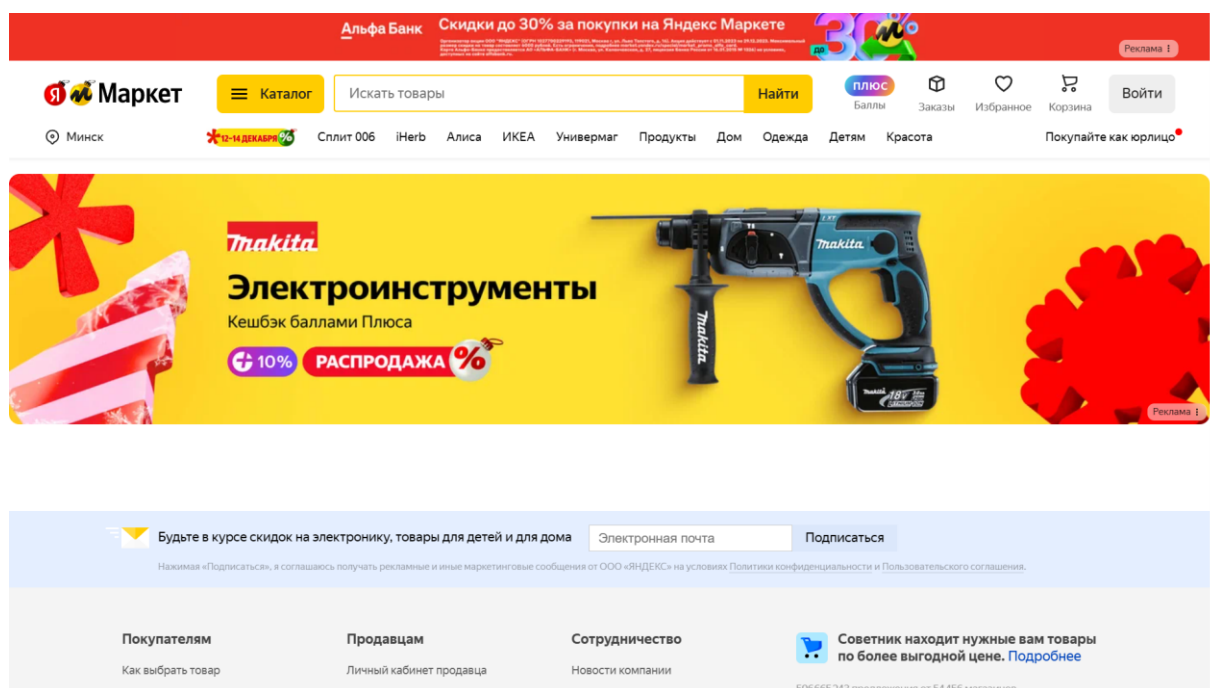


Рисунок 1.2 – Главная страница сервиса «Яндекс. Маркет»

«Яндекс. Маркет» – единая торговая площадка, где продавцы размещают товары, а покупатели сравнивают и выбирают то, что им нужно – от электроники до одежды. Площадка доступна независимо от того, есть ли у магазина сайт.

### 1.3 Интернет-магазин «Wildberries»

Интернет-магазин «Wildberries» [3] – международный интернет-магазин одежды, обуви, электроники, детских товаров, товаров для дома и других товаров.

География присутствия компании Беларусь, Россию и другие страны. Ежедневно через магазин оформляется 4 миллиона заказов. Площадка основана в 2003 году и управляется ООО «Вайлдберриз» со штаб-квартирой в Москве. На рисунке 1.3 приведено изображение одной из страниц сайта.

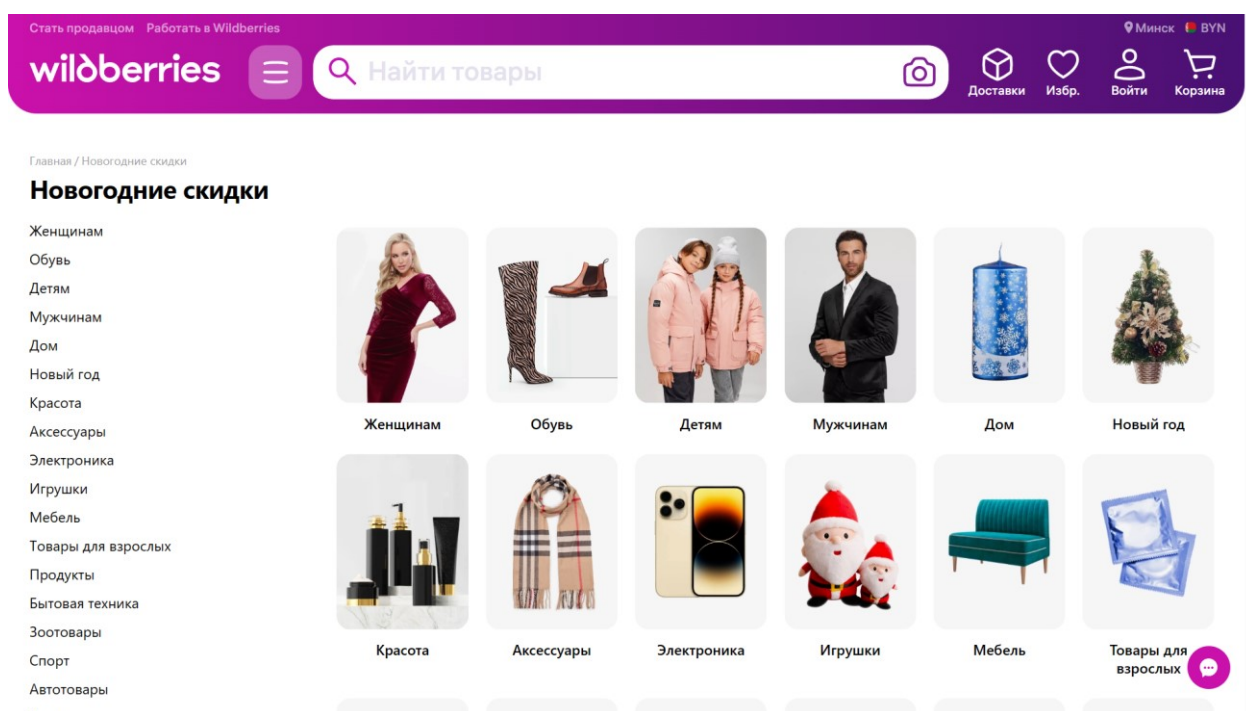


Рисунок 1.3 – Одна из страниц торговой площадки «Wildberries»

Бизнес-модель предприятия характеризуют как онлайн-гипермаркет, магазин универсального формата или торговая площадка с товарами компаний-партнёров. Компания напрямую сотрудничает с производителями одежды и официальными поставщиками товаров. Они самостоятельно формируют ассортимент своих товаров в интернет-магазине и розничные цены, а площадка зарабатывает на комиссии по итогам продаж.

### 1.4 Формирование требований

Исходя из разбора существующих аналогов торговых площадок, к разрабатываемой базе данных упрощённой торговой площадке можно выдвинуть некоторые требования.

Площадка должна в разной степени поддерживать следующие возможности:

- регистрация магазинов;
- формирование цен на товар у разных магазинов;
- написание комментариев и отзывов;
- оформление заказов;
- разграничение прав доступа для пользователей;
- поддержка встраивания рекламы на сайт;
- наличие раздела с новостями и купонами.

При последующей возможной разработке приложения под разрабатываемую в рамках настоящего курсового проекта базу данных необходимо сделать упор также на:

- доступность интерфейса с разных платформ;
- валидацию данных пользователей при совершении платежей, оформлении заказа, регистрации.
- правильное разграничение доступа к разделам или страницам сайта;
- безопасность при работе с данными пользователей;
- удобные интерфейсы для взаимодействия системой не только для клиентов, а также других типов пользователей: администраторов, курьеров и представителей магазинов.

В рамках данного проекта разрабатывается упрощённая модель базы данных для заданной предметной области. Эта модель может быть предварительной и не учитывать всех потребностей предметной области.

## **2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ И ВЫБОР ИНСТРУМЕНТОВ**

### **2.1 Формирование функциональных требований**

Функциональное требование – это заявление о том, как должна вести себя система. Он определяет, что система должна делать, чтобы удовлетворить потребности или ожидания пользователя. Функциональные требования можно рассматривать как функции, которые обнаруживает пользователь. Они отличаются от нефункциональных требований, которые определяют, как система должна работать внутри. Ниже изложены основные функциональные требования к приложению, которое можно будет разработать на основе проектируемой в рамках настоящего курсового проекта базы данных.

1 Просмотр списка товаров. Пользователь должен иметь возможность просматривать товары в разных категориях и подкатегориях. Товары можно фильтровать по производителю и линейке, а также сортировать, например, по средней цене от магазинов.

2 Регистрация и вход в приложение. Пользователю должно быть предоставлено несколько вариантов входа в приложения для улучшения опыта использования.

3 Оформление заказа. У пользователя должна быть возможность составить заказ, возможно, состоящий из нескольких позиций и оформить его. Каждый магазин определяет свои возможные способы оплаты и доставки. Заказ может объединить в себе лишь товары из одного магазина. При этом если есть необходимость заказать товары из разных магазинов, необходимо будет сделать несколько заказов.

4 Просмотр информации. Пункт включает в себя просмотр информации разного типа:

- описание товара;
- информация о поставщиках и линейках;
- сведения о магазинах, их способах оплаты и доставки, пунктах самовывоза;
- сведения об актуальных купонах и акциях;
- новости портала;
- отзывы к товарам;
- форумы и комментарии;
- часто задаваемые вопросы.



5 Журналирование действий. Основные виды действий должны заноситься в таблицу журналирования.

6 Права пользователей с ролью администратора. Они должны включать редактирование списка товаров без привязки к магазинам и конкретным ценам, добавление новых рекламных объектов в реестре отображаемой рекламы, изменение списка купонов, а также регистрацию новых магазинов и их представителей.

7 Права представителей магазинов. Представители магазинов при использовании приложением относятся к пользователям с отдельной соответствующей ролью. Они могут редактировать товары, выставленные их магазином на разрабатываемой торговой площадке.

8 Права пользователей с ролью курьеров. В будущем планируется расширение базы данных и приложения с возможностью отслеживания истории доставок, самовывозов, а также с введением системы рейтинга для курьеров и обычных клиентов.

## **2.2 Выбор инструментов разработки**

Исходя из описанных выше функциональных требований системы и имеющегося опыта разработки, выбор сделан в пользу проектирования с использованием реляционных систем управления базой данных. В настоящее время одними из самых используемых реляционных систем управления базой данных являются PostgreSQL и Oracle Database. Также набирают популярность комплексы, включающие внутри себя базу данных и многие вспомогательные инструменты и сервисы, такие, как, например, Supabase. Упомянутые технологии описаны ниже.

**2.2.1 Система управления базами данных PostgreSQL [4].** PostgreSQL – свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многие из возможностей стандарта SQL. Имеет открытый исходный код и является альтернативой коммерческим базам данных.

Данная система управления базами данных позволяет гибко управлять базами данных. С ее помощью можно создавать, модифицировать или удалять записи, отправлять транзакцию – набор из нескольких последовательных запросов на особом языке запросов SQL.

Система управления базами данных PostgreSQL нужна для:

– гибкого доступа к базам данных, их организации и хранения;

- управления записями в базах данных: создания, редактирования и удаления, обновления версий и так далее;
- просмотра нужной информации из базы по запросу, например для ее отправки на сайт или в интерфейс приложения;
- отправки транзакций, последовательных запросов, собранных в подобие скрипта;
- настройки и контроля доступа к той или иной информации, группировки пользователей по уровню прав;
- контроля версий и организации одновременного доступа к базе из разных источников так, чтобы предотвратить сбои;
- защиты информации от возможных утечек и потерь;
- контроля состояния базы в целом.

Иногда PostgreSQL называют бесплатным аналогом Oracle Database. Обе системы адаптированы под большие проекты и высокую нагрузку, но по-разному хранят данные.

Среди преимуществ PostgreSQL разработчики выделяют следующие пункты:

1 Объектно-реляционная модель. Традиционно популярные системы управления базами данных – реляционные. Это значит, что данные, которые в них хранятся, представляются в виде записей, связанных друг с другом отношениями. Получаются связанные списки, которые могут иметь между собой те или иные отношения, – так и образуется таблица.

2 Поддержка множества типов данных. Еще одна особенность PostgreSQL – поддержка большого количества типов записи информации. Это не только стандартные целочисленные значения, числа с плавающей точкой, строки и логические значения, но и денежный, геометрический, перечисляемый, бинарный и другие типы. PostgreSQL поддерживает битовые строки и сетевые адреса, массивы данных, в том числе многомерные, композитные типы и другие сложные структуры. В ней есть поддержка XML, JSON и NoSQL-баз.

3 Работа с большими объемами. В большинстве систем управления базами данных, рассчитанных на средние и небольшие проекты, есть ограничения по объему базы и количеству записей в ней. В PostgreSQL ограничений нет.

4 Поддержка сложных запросов. PostgreSQL работает со сложными, составными запросами. Система справляется с задачами разбора и выполнения трудоемких операций, которые подразумевают и чтение, и запись, и

валидацию одновременно. Она медленнее аналогов, если речь заходит только о чтении, но в других аспектах превосходит конкурентов.

5 Написание функций на нескольких языках. В PostgreSQL можно писать собственные функции – пользовательские блоки кода, которые выполняют те или иные действия. Эта возможность есть практически в любых системах управления базами данных, но PostgreSQL поддерживает больше языков, чем аналоги. Кроме стандартного SQL, в PostgreSQL можно писать на C и C++, Java, Python, PHP, Lua и Ruby. Он поддерживает V8 – один из движков JavaScript, поэтому JS тоже можно использовать.

6 Одновременная модификация базы. Важная особенность PostgreSQL – возможность одновременного доступа к базе с нескольких устройств. В системе управления базами данных реализована клиент-серверная архитектура, когда база данных хранится на сервере, а доступ к ней осуществляется с клиентских компьютеров. Так, например, реализуются разнообразные сайты. Одна из возможных сложностей – ситуация, когда несколько человек одновременно модифицируют базу и нужно избежать конфликтов.

7 Высокая мощность и широкая функциональность. PostgreSQL – бесплатная система управления базами данных с открытым исходным кодом, которая рассчитана на работу с объемными и сложными проектами. Она мощная, производительная, способна эффективно работать с большими массивами данных.

**2.2.2 Oracle Database [5]** – это объектно-реляционная система управления базами данных от компании Oracle. Она используется для создания структуры новой базы, ее наполнения, редактирования содержимого и отображения информации.

Данная система объединяет в себе две модели хранения информации: объектно-ориентированную и реляционную. Реляционная модель представляется как набор отношений между записями. Одни данные связаны с другими – так формируется база. Визуально ее можно представить как двумерную таблицу; математически – как модель, построенную на отношениях. Объектно-ориентированная модель воспринимает данные как объект. У объекта есть атрибуты, которые описывают его свойства, и методы – они нужны для взаимодействия с другими объектами. Каждый объект принадлежит к классу – это понятие можно представить как «схему» объекта. Объектно-ориентированный подход используется во многих языках программирования и упоминается во многих наших статьях. А в базах данных он нужен для работы с данными, у которых сложная структура.

Oracle Database работает и с объектно-ориентированной, и с реляционной моделью.

Система работает по принципу «клиент – сервер». Это означает, что ее основная часть размещается на сервере, там же, где и база данных. Человек работает с интерфейсом приложения-клиента. Клиентская часть управляет только пересылкой и получением информации от сервиса.

Достоинства такого подхода – в высоком уровне безопасности и легком доступе для клиентов. Клиент-серверная организация разгружает сеть и снимает вычислительную нагрузку с клиентских компьютеров. А вот сервер для такой системы управления базами данных должен быть мощным.

Информация в системе хранится в отдельных базах – экземплярах базы данных. Это не физические, а логические понятия, которые состоят из процессов и оперативной памяти. Все содержимое одного экземпляра имеет единую системную глобальную область – часть оперативной памяти, с которой работает.

Внутри экземпляров расположены логические пространства, которые называются табличными – tablespaces. Табличные пространства содержат компоненты данных – как файлы в папках.

Данная система управления базами данных состоит из одного или нескольких инстансов и программного обеспечения, которое ими управляет. Система поддерживает работу с независимыми базами в рамках одного инстанса. Она может работать и с мультиарендной архитектурой, где множественными клиентами управляет один экземпляр приложения. В Oracle поддерживаются кластеризация и секционирование – физическое разделение элементов баз данных без потери доступа.

**2.2.3 Supabase [6]** – это реляционная база данных на основе тех же технологий, что лежат в PostgreSQL – одной из самых популярных и надежных баз данных в мире.

Supabase – бесплатный аналог Firebase [7], полифункциональная платформа, объединяющая в себе несколько важных программных решений и упрощающая их реализацию до предельно примитивного уровня, чтобы разработчики могли добавлять в свои приложения или сайты такие функции, как:

- авторизация;
- хранилище файлов;
- обновление содержания в приложении в реальном времени.

Разработчики данного сервиса требуют оплату использования только при достижении количества запросов определенных значений. То есть на

этапе разработки оплачивать базу данных не придется, все возможности Supabase можно тестировать самостоятельно.

Supabase идет в комплекте с собственным хранилищем файлов, которое можно подключать к базе данных. Например, можно добавить в Supabase файлы изображений, еще на этапе их загрузки в базу данных создавать специальные ссылки и закреплять их за статьями, комментариями, профилями в таблицах Supabase. Таким образом, можно привязывать файлы из хранилища к записям из базы данных, создавая бесшовную систему. Разработчики создали достаточный набор готовых команд для управления файлами.

Важное преимущество Supabase – встроенная функция авторизации, реализованная практически на надёжном уровне.

Supabase – универсальный продукт, включающий в себя много компонентов, необходимых для создания полноценных приложений.

### **2.3 Обоснование выбора**

Выбрана реляционная модель баз данных. Реляционная модель предлагает следующие преимущества:

- структурированность данных и декларативный язык запросов;
- целостность данных и связей;
- гибкость в запросах и аналитике;
- нормализация и избыточность;
- масштабируемость;
- поддержка транзакций.

Исходя из вышеописанного, решено использовать реляционную модель данных для обеспечения структурированности, эффективности запросов и обеспечения целостности данных.

В ходе разработки учебного приложения целесообразно использовать PostgreSQL, потому что мощности Oracle Database избыточны. Supabase представляет собой обёртку над PostgreSQL с дополнительными удобными возможностями для разработки, например, механизмы аутентификации и поддержания сессии.

По описанным причинам выбор сделан в пользу Supabase. Это современный инструмент, позволяющий быстро и удобно вести разработку базы данных описанного разрабатываемого учебного приложения, также предоставляющий дополнительные возможности, которые могут быть удобны и полезны при последующей разработке самого приложения, которое будет использовать данную базу данных.

### 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

В деловой или личной сфере часто приходится работать с данными из различных источников, каждый из которых связан с определенным видом деятельности. В настоящее время благодаря огромным возможностям компьютеров, которые связаны с хранением и обработкой больших массивов информации компьютер применяется для решения широкого круга задач буквально во всех сферах человеческой деятельности. Одновременно с развитием компьютерной техники развивалась и теория баз данных, которые представляют собой наборы взаимосвязанных данных о некоторой предметной области. Такие наборы имеют определенную структуру и постоянно хранятся в памяти компьютера.

В результате развития концепций баз данных выделены три уровня представления информации: инфологический, даталогический и физический. На каждом уровне проводится структуризация информации таким образом, чтобы на третьем уровне информация могла быть представлена в виде структур данных, реализуемых в памяти электронно-вычислительной машины. На инфологическом уровне определяется какая информация о предметной области будет храниться и обрабатываться в компьютере, а в результате исследования предметной области строится ее инфологическая модель, которая описывается в терминах классов объектов и их взаимодействий. В инфологической модели информация представляется вне зависимости от того, что представляют собой данные и какие технические средства будут использоваться в дальнейшем для ее хранения и обработки.

Даталогическая и физическая модели непосредственно реализуются в системах управления базами данных, а физическая модель в свою очередь определяет структуру хранения данных на физических носителях. Цель инфологического проектирования заключается в представлении семантики (смысла) предметной области. Для описания предметной области наиболее часто используется ER-модель. ER-диаграмма модели имеет лексикографическую структуру и включает в себя текст и элементы графики. На практике инфологическая (семантическая) модель используется на первой стадии проектирования базы данных. При этом в терминах ER-модели описывается концептуальная схема базы данных, которая затем преобразуется к реляционной или другой схеме.

### 3.1 Концептуальная ER диаграмма

Концептуальная модель данных [8] – схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы. Менее масштабные системы могут обойтись и без этой модели. В этом случае можно сразу переходить к логической модели. В нотации Чена используются следующие правила и обозначения для проектирования для ER-диаграмм:

- сущность или объект обозначаются прямоугольником;
- отношения обозначаются ромбом;
- атрибуты объектов, обозначаются овалом;
- связь сущности и отношения обозначается прямой линией со стрелкой;
- необязательная связь обозначается пунктирной линией;
- мощная связь обозначается двойной линией;
- каждый атрибут может быть связан с одним объектом.

В приложении Б приведена концептуальная схема разрабатываемого продукта.

### 3.2 IDEF1X диаграмма

IDEF1X [9] используется для формирования графических представлений информационных моделей, которые отражают структуру и семантику информации внутри среды или системы.

IDEF1X позволяет строить семантические модели данных, которые могут служить для поддержки управления данными как ресурсом, интеграции информационных систем и построения компьютерных баз данных. Этот стандарт является частью семейства языков моделирования IDEF в области программной инженерии. Данный метод моделирования используется для моделирования данных стандартным, последовательным и предсказуемым образом, чтобы управлять ими как ресурсом. Он может быть использован в проектах, требующих стандартных средств определения и анализа ресурсов данных внутри организации.

Основные цели стандарта IDEF1X заключаются в том, чтобы предоставить:

- средство для полного понимания и анализа информационных ресурсов организаций;
- общее средство представления и передачи сложности данных;

- методики общего представления данных, необходимых для работы предприятия;
- средства для определения независимого от приложения представления данных;
- методики получения интегрированного определения данных из существующих ресурсов данных.

В приложении В приведена IDEF1X диаграмма разрабатываемого приложения.

### **3.3 Физическая схема базы данных**

Схема базы данных [10] – это структура и организация базы данных, которая определяет ее таблицы, поля, связи, ограничения и типы данных. Он служит образцом для организации данных и доступа к ним, предоставляя план действий для разработчиков, администраторов и пользователей при работе с базой данных. Схема необходима для эффективного и действенного управления данными, что приводит к повышению производительности базы данных и упрощению обслуживания.

Понимая структуру и организацию базы данных через ее схему, разработчик получает информацию о том, как информация хранится, доступна и обрабатывается в базе данных. Эти знания позволяют более эффективно взаимодействовать с данными, оптимизировать запросы, повышать целостность данных и обеспечивать соответствие базы данных требованиям приложения.

Отношения определяют взаимосвязь между таблицами, определяя, как данные в одной таблице связаны с данными в другой. В схеме базы данных существует три основных типа отношений:

- один-к-одному: объект в одной таблице связан с одним объектом в другой таблице;
- один-ко-многим: объект в одной таблице связан с несколькими объектами в другой таблице;
- многие-ко-многим: несколько объектов в одной таблице связаны с несколькими объектами в другой таблице.

Отношения имеют решающее значение для структуры и функций схемы, поскольку они обеспечивают эффективные средства получения данных и управления ими.

В приложении Г приведена физическая схема разрабатываемой базы данных с уже раскрытыми отношениями «многие-ко-многим».



### 3.4 Нормализация базы данных

Нормальная форма [11] – свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации.

Ниже приведены описания трёх первых нормальных форм.

1 Первая нормальная форма. Переменная отношения находится в первой нормальной форме тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов. В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в этой нормальной форме.

2 Вторая нормальная форма. Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут неприводимо зависит от её потенциального ключа. Функционально полная зависимость означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.

3 Третья нормальная форма. Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

## 4 РАЗРАБОТКА БАЗЫ ДАННЫХ

Используемый инструмент, Supabase, – это реляционная база данных, использующая SQL-синтаксис. SQL – это язык, ориентированный на взаимодействие с базами данных. Он позволяет создавать специальные команды, которые могут считывать, изменять, добавлять или удалять встроенные в базу данных объекты.

В рамках написания хранимых процедур понадобится расширение возможностей стандартного SQL – PL/pgSQL [12]. PL/pgSQL – процедурное расширение языка SQL, используемое в системе управления базами данных PostgreSQL. Этот язык предназначен для написания функций, триггеров и правил и обладает следующими особенностями:

- добавляет управляющие конструкции к стандарту SQL;
- допускает сложные вычисления;
- может использовать все объекты базы данных, определенные пользователем;
- является простым в использовании.

Стандартный SQL используется в PostgreSQL и других реляционных базах данных как основной язык для создания запросов. Он переносим и прост, как для изучения, так и для использования. Каждая такая конструкция языка выполняется сервером отдельно. Это значит, что клиентское приложение должно отправлять каждый запрос серверу, получить его результат, определенным образом согласно логике приложения обработать его, посылать следующий запрос и так далее. В случае, если клиент и сервер базы данных расположены на разных машинах, это может привести к нежелательному увеличению задержек и объема пересылаемых от клиента серверу и наоборот данных.

При использовании PL/pgSQL появляется возможность сгруппировать запросы и вычислительные блоки в единую конструкцию, которая будет размещаться и выполняться на сервере, а клиент будет отправлять запрос на её выполнение и получать результат, минуя все промежуточные пересылки данных назад и вперед, что в большинстве случаев позитивно сказывается на производительности. Также функциональность анонимных блоков позволяет писать запросы не на SQL, а на любом существующем процедурном языке сервера, в том числе PL/pgSQL, без создания хранимых функций на сервере системы управления базами данных. Функции, написанные на PL/pgSQL, могут принимать в качестве аргумента и возвращать как результат значения любого скалярного или составного типа, допустимые для сервера базы данных, включая определенные пользователем, строковые типы и записи.

## 4.1 Создание проекта в Supabase

На рисунке 4.1 приведено изображение формы создания нового проекта в панели управления Supabase.

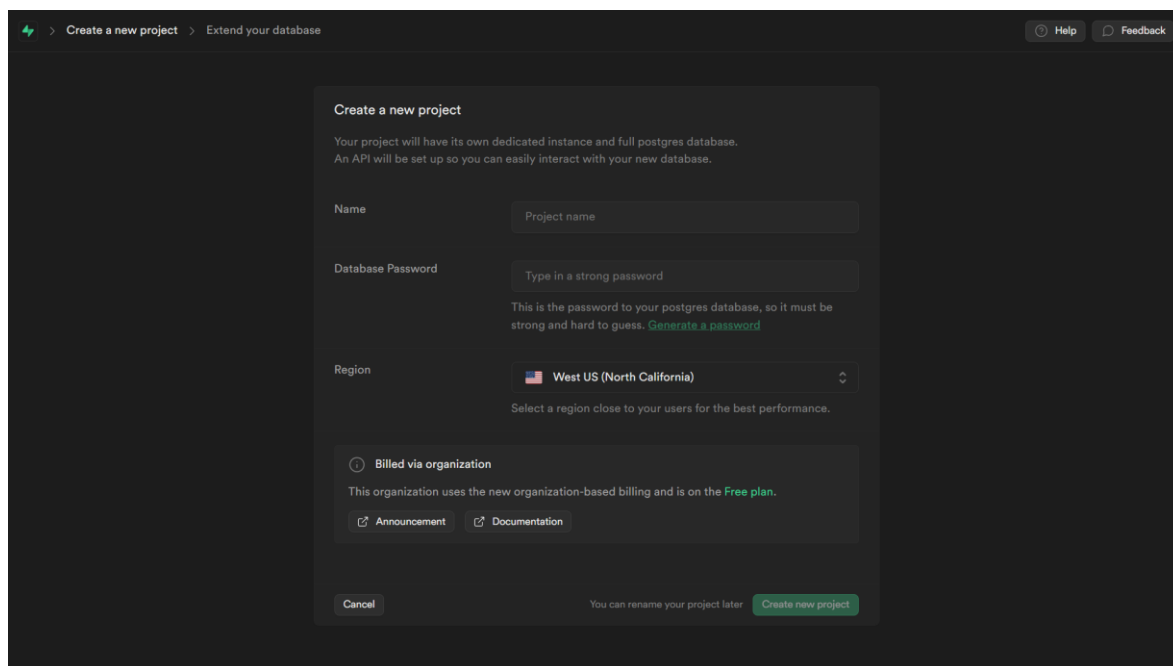
The image shows a dark-themed web interface for creating a new Supabase project. At the top, there's a breadcrumb trail: 'Create a new project' > 'Extend your database'. In the top right corner, there are links for 'Help' and 'Feedback'. The main form is titled 'Create a new project' and includes the following elements: a sub-header stating 'Your project will have its own dedicated instance and full postgres database. An API will be set up so you can easily interact with your new database.'; a 'Name' field with a placeholder 'Project name'; a 'Database Password' field with a placeholder 'Type in a strong password' and a note below it stating 'This is the password to your postgres database, so it must be strong and hard to guess. Generate a password'; a 'Region' dropdown menu currently showing 'West US (North California)' with a note below it saying 'Select a region close to your users for the best performance.'; a section titled 'Billed via organization' with a note 'This organization uses the new organization-based billing and is on the Free plan.' and links for 'Announcement' and 'Documentation'; and at the bottom, a 'Cancel' button, a note 'You can rename your project later', and a green 'Create new project' button.

Рисунок 4.1 – Создание нового проекта в панели управления Supabase

На данном этапе можно ввести название проекта, пароль для доступа и предпочитаемый регион для сервера базы данных. После успешного создания в списке проектов отображается созданный проект.

Бесплатная версия Supabase предоставляет возможность создать только два проекта. В рамках одной организации это может быть основная база данных и копия базы данных для тестирования. В рамках данного курсового проекта приложение не создаётся, поэтому клонирование базы данных для тестов опускается.

Был создан тестовый аккаунт для разработки учебного проекта внутри Supabase.

На рисунке 4.2 приведено изображение страницы с управлением проектами в Supabase.

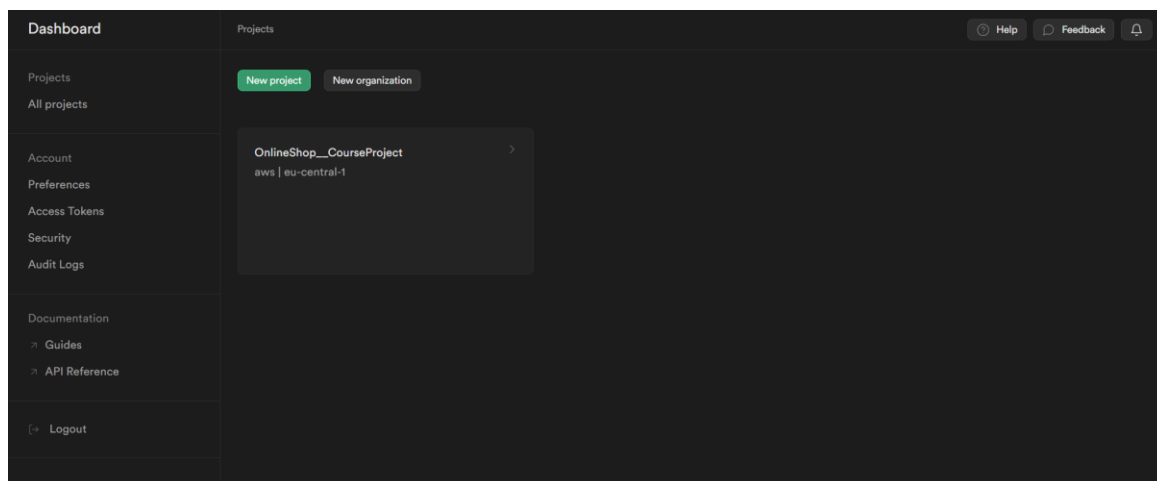


Рисунок 4.2 – Раздел управления проектами в Supabase

Чтобы посмотреть, как можно взаимодействовать с таблицами программно из своего приложения, необходимо открыть раздел API в нижней части боковой панели интерфейса Supabase. Там указан список команд и для подключения к базе данных из сторонних проектов, и для управления информацией в базе данных.

## 4.2 Разработка DDL-скриптов для создания таблиц

DDL – это подмножество языка SQL, используемое для определения структуры базы данных и ее объектов, таких как таблицы, представления, индексы и процедуры. DDL Операторы используются для создания, изменения и удаления объектов базы данных, включая таблицы, представления, индексы и хранимые процедуры. Некоторые из наиболее распространенных DDL операторы включают:

- CREATE;
- ALTER;
- DROP;
- TRUNCATE.

Для создания таблиц есть две опции: редактор таблиц и редактор SQL. В рамках разработки базы данных для приложения будет использоваться второй вариант – редактор SQL. Для разработки триггеров и процедур единственным вариантом остаётся редактор SQL. На рисунке 4.3 приведено изображение редактора.

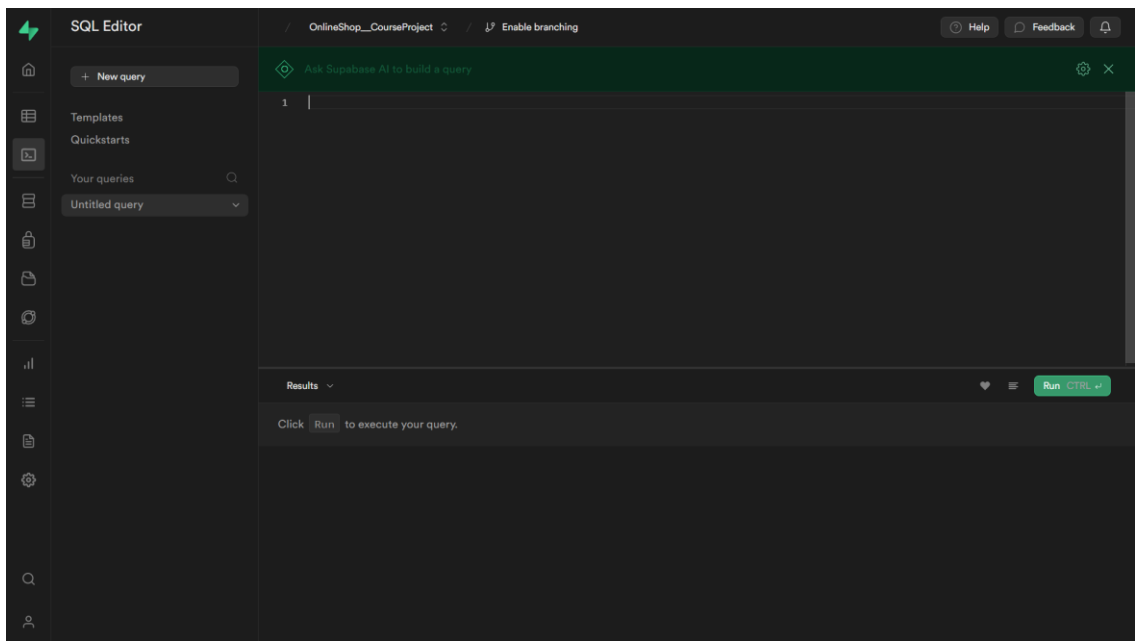


Рисунок 4.3 – Редактор SQL в проекте Supabase

В данном редакторе можно писать запросы SQL, а также код на процедурном языке PL/pgSQL.

На рисунке 4.4 изображён пример запроса для создания одной из наиболее объёмных таблиц в рамках разработки базы данных для торговой площадки. Эта таблица содержит в себе ссылку на товар, зафиксированную стоимость, количество единиц товара и ссылку на сам заказ, к которому эта группа товаров принадлежит.

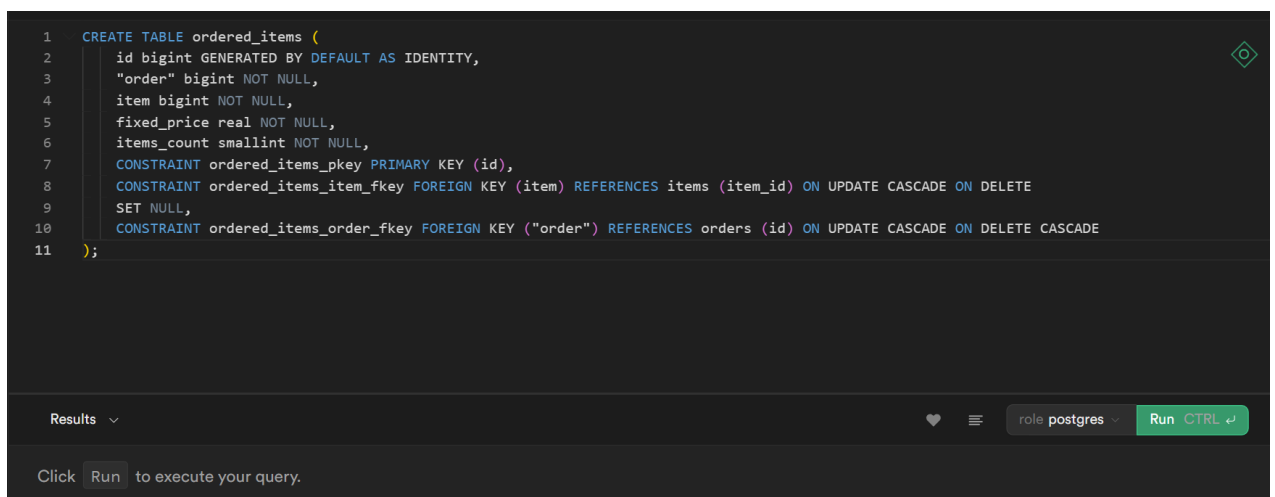


Рисунок 4.4 – Создание таблиц для заказанных единиц товара

Для создания таблиц применяется команда `CREATE TABLE`, после которой указывается название таблицы. Также с этой командой можно использовать ряд операторов, которые определяют столбцы таблицы и их атрибуты. После названия таблицы в скобках перечисляется спецификация для всех столбцов. Необязательные предложения ограничений задают ограничения, которым должны удовлетворять добавляемые или изменяемые строки, чтобы операция добавления или изменения была выполнена успешно.

Редактор предлагает подсветку синтаксиса, что помогает при написании длинных запросов.

### 4.3 Разграничение прав доступа

Механизм `row level security` [13] позволяет реализовать разграничение доступа к данным средствами базы данных прозрачно для работающих с ней приложений. Даже если злоумышленник получил прямой доступ к базе, например под учетной записью владельца схемы с данными, RLS может не дать ему увидеть защищенную информацию. Политики RLS позволяют убирать строки из выборки целиком или скрывать значения столбцов для строк, к которым пользователь не имеет доступа. В этом отличие от обычного управления правами в базе данных, которые можно выдать только на объект целиком. При выполнении любого запроса к базе планировщик проверяет, есть ли для этих таблиц политики доступа.

Команда `CREATE POLICY` создаёт новую политику защиты на уровне строк для таблицы. Политика даёт разрешение на выборку, добавление, изменение или удаление строк, удовлетворяющих соответствующему выражению политики. Существующие строки таблицы проверяются по выражению, указанному в `USING`, тогда как строки, которые могут быть созданы командами `INSERT` или `UPDATE` проверяются по выражению, указанному в `WITH CHECK`. Когда выражение `USING` истинно для заданной строки, эта строка видна пользователю, а если ложно или выдаёт `NULL`, строка не видна. Когда выражение `WITH CHECK` истинно для заданной строки, эта строка добавляется или изменяется, а если ложно или выдаёт `NULL`, происходит ошибка. Пример создания политики доступа для разрабатываемой базы данных представлен на рисунке 4.5.

```
1 CREATE POLICY "Users can read their own orders" auth.uid() = id ON "public"."orders"  
2 AS PERMISSIVE FOR SELECT  
3 TO authenticated  
4 USING (auth.uid() = client)  
5 |
```

Рисунок 4.5 – Политика доступа для таблицы

Принимая во внимание объём базы данных, создание полного необходимого набора политик доступа для полноценного приложения опускается. В приложении А для примера приведены созданные политики для некоторых таблиц.

#### 4.4 Создание некоторых процедур, функций и триггеров

Хранимая процедура [14] – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных. Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры могут возвращать множества результатов, то есть результаты запроса SELECT. Такие множества результатов могут обрабатываться, используя курсоры, другими хранимыми процедурами, возвращая указатель результирующего множества, либо же приложениями. Хранимые процедуры могут также содержать объявленные переменные для обработки данных и курсоров, которые позволяют организовать цикл по нескольким строкам в таблице. Стандарт SQL предоставляет для работы выражения IF, LOOP, REPEAT, CASE и многие другие. Хранимые процедуры могут принимать переменные, возвращать результаты или изменять переменные и возвращать их, в зависимости от того, где переменная объявлена.

На рисунке 4.6 приведен пример создания функции для регистрации клиента в приложении.

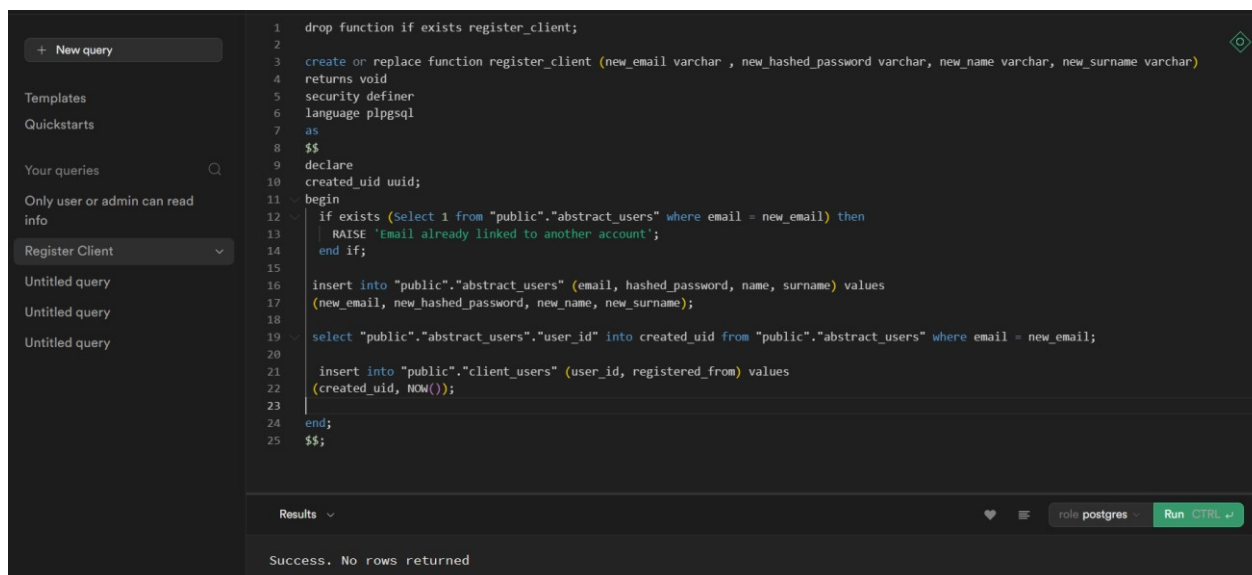


Рисунок 4.6 – Функция регистрации клиента

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением, удалением строки в заданной таблице, или изменением данных в определённом столбце заданной таблицы реляционной базы данных.

Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Момент запуска триггера определяется с помощью ключевых слов BEFORE или AFTER. В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись. Некоторые системы управления базами данных накладывают ограничения на операторы, которые могут быть использованы в триггере.

В некоторых серверах триггеры могут вызываться не для каждой модифицируемой записи, а один раз на изменение таблицы. Такие триггеры называются табличными.

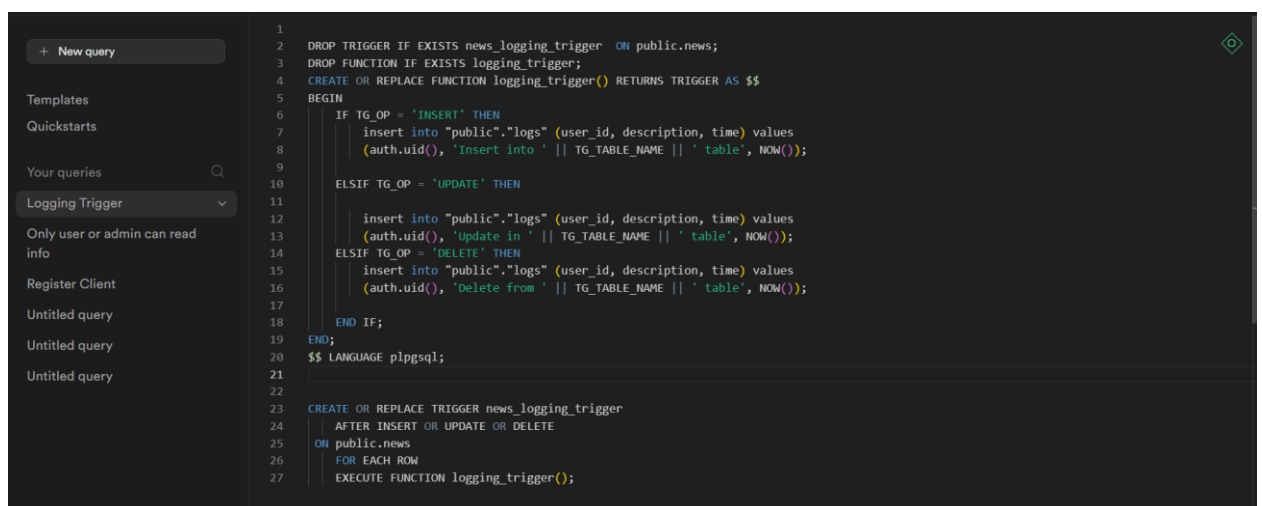
С увеличением размеров приложения возрастает необходимость во всё более удобных триггерах. В рамках разработки базы данных для упрощённой торговой площадки для первой версии приложения создан триггер,



отвечающий за общее журналирование действий. Такой триггер можно привязать к любой таблице, так как он записывает только тип и дату действия. При необходимости хранить более подробные данные по каждой таблице можно создавать отдельные триггеры.

Триггер `logging_trigger` демонстрирует подход, когда один триггер можно привязывать к абсолютно любой таблице. При этом если расширить немного кодовую базу триггера и его реальную пользу, можно также в описание действия заносить сериализованные изменения в изменённой таблице, например, в формате JSON. Система управления базами данных PostgreSQL имеет встроенные инструменты для работы с таким форматом.

На рисунке 4.7 приведен пример создания универсального триггера для журналирования событий и пример привязки такого триггера к таблице новостей в разрабатываемой базе данных.



```
1 DROP TRIGGER IF EXISTS news_logging_trigger ON public.news;
2 DROP FUNCTION IF EXISTS logging_trigger;
3 CREATE OR REPLACE FUNCTION logging_trigger() RETURNS TRIGGER AS $$
4 BEGIN
5     IF TG_OP = 'INSERT' THEN
6         insert into "public"."logs" (user_id, description, time) values
7             (auth.uid(), 'Insert into ' || TG_TABLE_NAME || ' table', NOW());
8     ELSIF TG_OP = 'UPDATE' THEN
9         insert into "public"."logs" (user_id, description, time) values
10            (auth.uid(), 'Update in ' || TG_TABLE_NAME || ' table', NOW());
11     ELSIF TG_OP = 'DELETE' THEN
12         insert into "public"."logs" (user_id, description, time) values
13            (auth.uid(), 'Delete from ' || TG_TABLE_NAME || ' table', NOW());
14     END IF;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE OR REPLACE TRIGGER news_logging_trigger
19 AFTER INSERT OR UPDATE OR DELETE
20 ON public.news
21 FOR EACH ROW
22 EXECUTE FUNCTION logging_trigger();
```

Рисунок 4.7 – Создание и ассоциация триггера с таблицей

Этот триггер также можно привязать к любой другой таблице для подобных записей о действиях над таблицей общего вида.

## 4.5 Создание индексов

Индекс [15] – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие

строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск – например, сбалансированного дерева.

Необходимость создания индексов может вызвать медленная работа запросов при выборке данных. Показать это может только тестирование на больших объёмах и показания при использовании реального приложения с реальными данными.

#### **4.6 Тестирование базы данных**

Тестирование базы данных [16] является важной частью процесса обеспечения качества программного обеспечения. В данной статье мы рассмотрим основные аспекты тестирования баз данных, а также предложим подходы и методики для эффективного выполнения этой задачи.

Выделяют следующие виды тестирования баз данных:

- тестирование структуры данных;
- тестирование манипуляции данными;
- тестирование хранимых процедур и функций;
- тестирование интеграции с приложениями;
- тестирование производительности;
- тестирование безопасности.

Тестирование можно производить с помощью тестирующих фреймворков, таких, как Jest [17]. Тестирование может проводиться на тестовой базе данных, которую клонируют от основной.

Ввиду того, что в рамках данного курсового проекта разрабатывается только база данных без приложения и сопутствующих ему тестов, написание тестов опускается.

## ЗАКЛЮЧЕНИЕ

В заключение данного курсового проекта по проектированию базы данных для торговой площадки и интернет-магазина следует отметить значительное значение, которое базы данных имеют в современном электронном бизнесе. Решение о наиболее подходящей структуре и организации базы данных должно быть принято с учетом множества аспектов, включая функциональность, безопасность, производительность и масштабируемость.

На протяжении данного проекта были изучены современные требования к разработке интернет-магазинов или торговых площадок, а также рассмотрены специфические потребности в управлении информацией о продуктах, пользователях, заказах и других аспектах внутренней деятельности торговых площадок. В результате чего были выявлены ключевые функциональные и нефункциональные требования к базе данных, которые влияют как на архитектуру базы данных, так и на основные процессы бизнеса.

Проектирование базы данных для интернет-магазина или торговой площадки является сложной задачей, которая требует глубокого понимания бизнес-потребностей, требований пользователей и технических аспектов хранения и обработки данных. Принято решение использовать реляционную модель данных для обеспечения структурированности, эффективности запросов и обеспечения целостности данных. Были изучены и применены передовые методы проектирования, а также обеспечения безопасности и надежности нашей базы данных. Во время проектирования базы данных и названий сущностей были использованы подходы по именованию сущностей из общепризнанной конвенции по именованию сущностей в базах данных [18].

Данный курсовой проект является важным этапом в понимании и практическом применении концепций проектирования баз данных для торговой площадки и интернет-магазина, и позволяет углубить знания о процессах управления информацией и данных в электронной коммерции.

Цели курсового проекта можно считать достигнутыми.

## СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

- [1] Сервис «Каталог Онлайнер» [Электронный ресурс]. – Режим доступа: <https://catalog.onliner.by/>. – Дата доступа: 10.10.2023.
- [2] Сайт «Яндекс. Маркет» [Электронный ресурс]. – Режим доступа: <https://market.yandex.by/>. – Дата доступа: 11.10.2023.
- [3] Торговая площадка «Wildberries» [Электронный ресурс]. – Режим доступа: <https://www.wildberries.by/>. – Дата доступа: 12.10.2023.
- [4] Система управления базами данных PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/>. – Дата доступа: 20.10.2023.
- [5] Система управления базами данных Oracle [Электронный ресурс]. – Режим доступа: <https://www.oracle.com/database/>. – Дата доступа: 20.10.2023.
- [6] Облачная база данных Supabase [Электронный ресурс]. – Режим доступа: <https://supabase.com/>. – Дата доступа: 20.10.2023.
- [7] Облачная база данных Firebase [Электронный ресурс]. – Режим доступа: <https://firebase.google.com/>. – Дата доступа: 20.10.2023.
- [8] Концептуальная ER диаграмма [Электронный ресурс]. – Режим доступа: <https://miro.com/ru/diagramming/what-is-an-er-diagram/>. – Дата доступа: 28.10.2023.
- [9] Диаграмма IDEF1X [Электронный ресурс]. – Режим доступа: <https://www.cfin.ru/vernikov/idef/idef1x.shtml>. – Дата доступа: 28.10.2023.
- [10] Физическая схема базы данных [Электронный ресурс]. – Режим доступа: <https://askanydifference.com/ru/difference-between-logical-database-model-and-physical-database-model/>. – Дата доступа: 28.10.2023.
- [11] Нормальные формы базы данных [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/254773/>. – Дата доступа: 30.10.2023.
- [12] Процедурный язык PL/pgSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/current/plpgsql.html>. – Дата доступа: 01.11.2023.
- [13] Механизм блокировок на уровне строк [Электронный ресурс]. – Режим доступа: <https://supabase.com/docs/guides/auth/row-level-security>. – Дата доступа: 10.11.2023.
- [14] Хранимые функции и процедуры [Электронный ресурс]. – Режим доступа: <https://www.postgresqltutorial.com/postgresql-plpgsql/>. – Дата доступа: 15.11.2023.
- [15] Индексы базы данных [Электронный ресурс]. – Режим доступа: [https://www.tutorialspoint.com/postgresql/postgresql\\_indexes.htm](https://www.tutorialspoint.com/postgresql/postgresql_indexes.htm). – Дата доступа: 15.11.2023.

[16] Виды тестирования базы данных [Электронный ресурс]. – Режим доступа: <https://sky.pro/media/kak-testirovat-bazy-dannyh/>. – Дата доступа: 25.11.2023.

[17] Фреймворк для тестирования Jest [Электронный ресурс]. – Режим доступа: <https://jestjs.io/>. – Дата доступа: 01.12.2023.

[18] Соглашения по именованию в SQL [Электронный ресурс]. – Режим доступа: <https://www.sqlstyle.guide/#naming-conventions>. – Дата доступа: 01.12.2023.

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Листинг кода

```
CREATE TABLE abstract_users (  
    user_id uuid NOT NULL default gen_random_uuid (),  
    name character varying NOT NULL,  
    surname character varying NOT NULL,  
    CONSTRAINT abstract_users_pkey PRIMARY KEY (user_id)  
);  
CREATE TABLE user_settings (  
    user_id uuid NOT NULL,  
    theme character varying NOT NULL default 'light' :: character varying,  
    language character varying NOT NULL default 'EN' :: character varying,  
    CONSTRAINT user_settings_pkey PRIMARY KEY (user_id),  
    CONSTRAINT user_settings_user_id_fkey FOREIGN KEY (user_id) REFERENCES abstract_users  
(user_id) ON UPDATE CASCADE ON DELETE CASCADE  
);  
CREATE TABLE logs (  
    log_id bigint GENERATED BY DEFAULT AS IDENTITY,  
    user_id uuid NULL,  
    description character varying NULL,  
    CONSTRAINT logs_pkey PRIMARY KEY (log_id),  
    CONSTRAINT logs_user_id_fkey FOREIGN KEY (user_id) REFERENCES abstract_users (user_id) ON  
UPDATE CASCADE ON DELETE  
SET  
    NULL  
);  
CREATE TABLE courier_users (  
    user_id uuid NOT NULL,  
    rating smallint NOT NULL,  
    CONSTRAINT courier_users_pkey PRIMARY KEY (user_id),  
    CONSTRAINT courier_users_user_id_fkey FOREIGN KEY (user_id) REFERENCES abstract_users  
(user_id) ON UPDATE CASCADE ON DELETE CASCADE  
);  
CREATE TABLE admin_users (  
    user_id uuid NOT NULL,  
    role character varying NOT NULL,  
    CONSTRAINT admin_users_pkey PRIMARY KEY (user_id),  
    CONSTRAINT admin_users_user_id_fkey FOREIGN KEY (user_id) REFERENCES abstract_users  
(user_id)  
);  
CREATE TABLE news (  
    news_id uuid NOT NULL default gen_random_uuid (),  
    title character varying NOT NULL,  
    description text NOT NULL,  
    date date NOT NULL,  
    posted_by uuid NULL,  
    CONSTRAINT news_pkey PRIMARY KEY (news_id),  
    CONSTRAINT news_posted_by_fkey FOREIGN KEY (posted_by) REFERENCES admin_users  
(user_id) ON UPDATE CASCADE ON DELETE
```

```

        SET
        NULL
    );
CREATE TABLE advertisement_banners (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    author uuid NULL,
    date_created date NOT NULL,
    showing_term date NOT NULL,
    banner_link character varying NOT NULL,
    CONSTRAINT advertisement_banners_pkey PRIMARY KEY (id),
    CONSTRAINT advertisement_banners_author_fkey FOREIGN KEY (author) REFERENCES
admin_users (user_id) ON UPDATE CASCADE ON DELETE
    SET
    NULL
);
CREATE TABLE client_users (
    user_id uuid NOT NULL,
    registered_from date NOT NULL,
    CONSTRAINT client_users_pkey PRIMARY KEY (user_id),
    CONSTRAINT client_users_user_id_fkey FOREIGN KEY (user_id) REFERENCES abstract_users
(user_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE shops (
    shop_id integer GENERATED BY DEFAULT AS IDENTITY,
    name character varying NOT NULL,
    official_organization_name character varying NOT NULL,
    registration_number character varying NOT NULL,
    description text NOT NULL,
    CONSTRAINT shop_pkey PRIMARY KEY (shop_id)
);
CREATE TABLE shop_pick_up_points (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    address character varying NOT NULL,
    shop_id integer NOT NULL,
    operating_mode character varying NOT NULL,
    CONSTRAINT shop_pick_up_points_pkey PRIMARY KEY (id),
    CONSTRAINT shop_pick_up_points_shop_id_fkey FOREIGN KEY (shop_id) REFERENCES shops
(shop_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE shop_representative_users (
    user_id uuid NOT NULL,
    shop_id integer NOT NULL,
    CONSTRAINT shop_representative_users_pkey PRIMARY KEY (user_id),
    CONSTRAINT shop_representative_users_shop_id_fkey FOREIGN KEY (shop_id) REFERENCES
shops (shop_id) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT shop_representative_users_user_id_fkey FOREIGN KEY (user_id) REFERENCES
abstract_users (user_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE shipping_methods (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    name character varying NOT NULL,
    description character varying NOT NULL,
    CONSTRAINT shipping_methods_pkey PRIMARY KEY (id)

```

```

);
CREATE TABLE shop_shipping_methods (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    shipping_method smallint NOT NULL,
    shop integer NOT NULL,
    CONSTRAINT shop_shipping_methods_pkey PRIMARY KEY (id),
    CONSTRAINT shop_shipping_methods_shipping_method_fkey FOREIGN KEY (shipping_method)
REFERENCES shipping_methods (id) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT shop_shipping_methods_shop_fkey FOREIGN KEY (shop) REFERENCES shops
(shop_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE payment_approaches (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    name character varying NOT NULL,
    description character varying NOT NULL,
    CONSTRAINT payment_approaches_pkey PRIMARY KEY (id)
);
CREATE TABLE shop_payment_approaches (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    shop integer NOT NULL,
    payment_approach bigint NOT NULL,
    CONSTRAINT shop_payment_approaches_pkey PRIMARY KEY (id),
    CONSTRAINT shop_payment_approaches_payment_approach_fkey FOREIGN KEY
(payment_approach) REFERENCES payment_approaches (id) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT shop_payment_approaches_shop_fkey FOREIGN KEY (shop) REFERENCES shops
(shop_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE forums (
    forum_id bigint GENERATED BY DEFAULT AS IDENTITY,
    topic character varying NOT NULL,
    description text NULL,
    CONSTRAINT forums_pkey PRIMARY KEY (forum_id)
);
CREATE TABLE forum_comments (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    forum_id bigint NOT NULL,
    author uuid NOT NULL,
    date timestamp without time zone NOT NULL,
    text text NOT NULL,
    CONSTRAINT forum_comments_pkey PRIMARY KEY (id),
    CONSTRAINT forum_comments_author_fkey FOREIGN KEY (author) REFERENCES abstract_users
(user_id) ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT forum_comments_forum_id_fkey FOREIGN KEY (forum_id) REFERENCES forums
(forum_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE coupons (
    coupon_id integer GENERATED BY DEFAULT AS IDENTITY,
    title character varying NOT NULL,
    description character varying NOT NULL,
    sale_amount smallint NOT NULL,
    validity_period date NOT NULL,
    CONSTRAINT coupons_pkey PRIMARY KEY (coupon_id)
);

```



```

CREATE TABLE frequently_asked_questions (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    question character varying NOT NULL,
    answer text NOT NULL,
    CONSTRAINT frequently_asked_questions_pkey PRIMARY KEY (id)
);
CREATE TABLE categories (
    category_id smallint GENERATED BY DEFAULT AS IDENTITY,
    title character varying NOT NULL,
    CONSTRAINT categories_pkey PRIMARY KEY (category_id)
);
CREATE TABLE subcategories (
    subcategory_id integer GENERATED BY DEFAULT AS IDENTITY,
    parent_category smallint NOT NULL,
    title character varying NOT NULL,
    CONSTRAINT subcategories_pkey PRIMARY KEY (subcategory_id),
    CONSTRAINT subcategories_parent_category_fkey FOREIGN KEY (parent_category) REFERENCES
categories (category_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE manufacturers (
    manufacturer_id integer GENERATED BY DEFAULT AS IDENTITY,
    name character varying NOT NULL,
    info text NULL,
    CONSTRAINT manufacturers_pkey PRIMARY KEY (manufacturer_id)
);
CREATE TABLE model_lines (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    manufacturer integer NOT NULL,
    name character varying NULL,
    info text NOT NULL,
    CONSTRAINT model_lines_pkey PRIMARY KEY (id),
    CONSTRAINT model_lines_manufacturer_fkey FOREIGN KEY (manufacturer) REFERENCES
manufacturers (manufacturer_id) ON UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE items (
    item_id bigint GENERATED BY DEFAULT AS IDENTITY,
    title character varying NOT NULL,
    subcategory integer NULL,
    model_line bigint NULL,
    description text NOT NULL,
    CONSTRAINT items_pkey PRIMARY KEY (item_id),
    CONSTRAINT items_model_line_fkey FOREIGN KEY (model_line) REFERENCES model_lines (id)
ON UPDATE CASCADE ON DELETE
SET NULL,
    CONSTRAINT items_subcategory_fkey FOREIGN KEY (subcategory) REFERENCES subcategories
(subcategory_id) ON UPDATE CASCADE ON DELETE
SET NULL
);
CREATE TABLE priced_items (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    price real NOT NULL,
    shop integer NOT NULL,
    item bigint NOT NULL,

```

```

        CONSTRAINT priced_items_pkey PRIMARY KEY (id),
        CONSTRAINT priced_items_item_fkey FOREIGN KEY (item) REFERENCES items (item_id) ON
UPDATE CASCADE ON DELETE CASCADE,
        CONSTRAINT priced_items_shop_fkey FOREIGN KEY (shop) REFERENCES shops (shop_id) ON
UPDATE CASCADE ON DELETE CASCADE
    );
CREATE TABLE orders (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    date timestamp with time zone NOT NULL default now(),
    shop integer NULL,
    client uuid NULL,
    CONSTRAINT orders_pkey PRIMARY KEY (id),
    CONSTRAINT orders_client_fkey FOREIGN KEY (client) REFERENCES client_users (user_id) ON
UPDATE CASCADE ON DELETE
SET NULL,
        CONSTRAINT orders_shop_fkey FOREIGN KEY (shop) REFERENCES shops (shop_id) ON
UPDATE CASCADE ON DELETE
SET NULL
    );
CREATE TABLE ordered_items (
    id bigint GENERATED BY DEFAULT AS IDENTITY,
    "order" bigint NOT NULL,
    item bigint NOT NULL,
    fixed_price real NOT NULL,
    items_count smallint NOT NULL,
    CONSTRAINT ordered_items_pkey PRIMARY KEY (id),
    CONSTRAINT ordered_items_item_fkey FOREIGN KEY (item) REFERENCES items (item_id) ON
UPDATE CASCADE ON DELETE
SET NULL,
        CONSTRAINT ordered_items_order_fkey FOREIGN KEY ("order") REFERENCES orders (id) ON
UPDATE CASCADE ON DELETE CASCADE
    );
CREATE TABLE reviews (
    review_id bigint GENERATED BY DEFAULT AS IDENTITY,
    text text NOT NULL,
    item bigint NOT NULL,
    author uuid NULL,
    CONSTRAINT reviews_pkey PRIMARY KEY (review_id),
    CONSTRAINT reviews_author_fkey FOREIGN KEY (author) REFERENCES client_users (user_id) ON
UPDATE CASCADE ON DELETE
SET NULL,
        CONSTRAINT reviews_item_fkey FOREIGN KEY (item) REFERENCES items (item_id) ON UPDATE
CASCADE ON DELETE CASCADE
    );
CREATE POLICY "Enable insert to Reviews for authenticated users only" ON "public"."reviews"
AS PERMISSIVE FOR INSERT
TO authenticated
WITH CHECK (true);
CREATE POLICY "Users can read their own ordersauth.uid() = id" ON "public"."orders"
AS PERMISSIVE FOR SELECT
TO authenticated
USING (auth.uid() = client);
CREATE POLICY "User can view only his info" ON "public"."client_users"

```

```

AS PERMISSIVE FOR SELECT
TO authenticated
USING (auth.uid() = user_id);
CREATE POLICY "Courier has permission to his row only" ON "public"."courier_users"
AS PERMISSIVE FOR SELECT
TO authenticated
USING (auth.uid() = user_id);
CREATE POLICY "Only admins can insert values" ON "public"."news"
AS PERMISSIVE FOR INSERT
TO authenticated
WITH CHECK (auth.uid() in (select user_id from admin_users));
drop function if exists register_client;
create or replace function register_client (new_email varchar , new_hashed_password varchar, new_name
varchar, new_surname varchar)
returns void
security definer
language plpgsql
as
$$
declare
created_uid uuid;
begin
if exists (Select 1 from "public"."abstract_users" where email = new_email) then
    RAISE 'Email already linked to another account';
end if;
insert into "public"."abstract_users" (email, hashed_password, name, surname) values (new_email,
new_hashed_password, new_name, new_surname);
select "public"."abstract_users"."user_id" into created_uid from "public"."abstract_users" where email =
new_email;
insert into "public"."client_users" (user_id, registered_from) values (created_uid, NOW());
end;
$$;
DROP TRIGGER IF EXISTS news_logging_trigger ON public.news;
DROP FUNCTION IF EXISTS logging_trigger;
CREATE OR REPLACE FUNCTION logging_trigger() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        insert into "public"."logs" (user_id, description, time) values
        (auth.uid(), 'Insert into ' || TG_TABLE_NAME || ' table', NOW());
    ELSIF TG_OP = 'UPDATE' THEN
        insert into "public"."logs" (user_id, description, time) values
        (auth.uid(), 'Update in ' || TG_TABLE_NAME || ' table', NOW());
    ELSIF TG_OP = 'DELETE' THEN
        insert into "public"."logs" (user_id, description, time) values
        (auth.uid(), 'Delete from ' || TG_TABLE_NAME || ' table', NOW());
    END IF;
END;
$$ LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER news_logging_trigger
AFTER INSERT OR UPDATE OR DELETE
ON public.news
FOR EACH ROW
EXECUTE FUNCTION logging_trigger();

```

**ПРИЛОЖЕНИЕ Б**  
**(обязательное)**  
**Концептуальная схема**

**ПРИЛОЖЕНИЕ В**  
**(обязательное)**  
**Логическая модель**

**ПРИЛОЖЕНИЕ Г**  
**(обязательное)**  
**Физическая модель**

**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Ведомость документов**