Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА ИУ7 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных маркетплейса

Студент	**ИУ7-65Б**		**Равашдех**
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			**Романова Т.Н.**
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка 61 с., 8 рис., 22 таблиц, 13 источников, 1 приложение.

БАЗА ДАННЫХ, СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ, РЕЛЯЦИОННАЯ БАЗА ДАННЫХ, МАРКЕТПЛЕЙС.

Цель работы — разработка программного обеспечения маркетплейса, которое позволит продавцам размещать предложения о продаже товаров, а покупателям находить интересующие товары и покупать их.

Был проведен анализ предметоной области, сравнительный анализ существующих решений и анализ существующих баз данных. Была формализована задача и требования к разрабатываемой базе данных и приложению. Была проведена формализация данных и ролей в базе данных и приложении. Было проведено описание сущностей базы данных, описание атрибутов, типов данных и ограничений, проектирование базы данных, описание ролевой модели. В данном разделе был проведен выбор средств реализации приложения, языка программирования, системы управления базой данных. Были приведены архитектура приложения, листинги кода для создания таблиц базы данных и листинги взаимодействия приложения с базой данных. Исследование продемонстрировало, что оптимизация запросов за счёт использования индексов — с учётом структуры запросов и особенностей данных — позволяет сократить время их выполнения.

Содержание

ВВЕДЕНИЕ

Маркетплейс — это онлайн-платформа, основная задача которой заключается в предоставлении продавцам удобных инструментов для размещения и продвижения своих товаров, а покупателям — единый интерфейс для поиска, выбора и приобретения продукции. В отличие от традиционных интернет-магазинов, маркетплейс не ограничивается товарами одного поставщика, что обеспечивает широкое разнообразие ассортимента и конкурентные цены.

Актуальность маркетплейсов обусловлена стремительным развитием электронной коммерции и цифровых сервисов. Современные пользователи всё чаще совершают покупки в интернете, и спрос на универсальные торговые платформы постоянно растёт. Маркетплейсы становятся важным элементом экономической экосистемы, так как упрощают взаимодействие между продавцами и покупателями, способствуют развитию малого и среднего бизнеса, а также формируют новые бизнес-модели.

Функционирование маркетплейса невозможно без хранения и обработки больших объёмов информации. В базе данных маркетплейса хранятся сведения о товарах, пердложениях, пользователях, заказах и отзывах.

Цель курсовой работы — разработка программного обеспечения маркетплейса, которое позволит продавцам размещать предложения о продаже товаров, а покупателям находить интересующие товары и покупать их.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1) провести анализ предметной области и существующих баз данных;
2) спроектировать базу данных и программное обеспечение;
3) реализовать спроектированные базу данных и программное обеспечение;
4) провести исследование разработанной базы данных.

1 Аналитический раздел

В этом разделе будет проведен анализ предметоной области, сравнительный анализ существующих решений и анализ существующих баз данных. Будет формализована задача и требования к разрабатываемой базе данных и приложению. Будет проведена формализация данных и ролей в базе данных и приложении.

1.1 Анализ предметной области

При планировании покупки некоторого товара, пользователи перебирают множество вариантов товара, просматривают множество сайтов магазинов, чтобы найти самое выгодное предложение того самого товара.

Однако можно не тратить время на перебор всех вариантов и комбинаций продавцов-товаров, а воспользоваться маркетплейсом, где все можно найти все товары от различных продавцов, выбрать необходимый, а затем получить список предложений из разных магазинов. Все удобно собрано в одном месте.

1.2 Анализ существующих решений

На российском рынке выделяются несколько крупных маркетплейсов: *Wildberries* [3], *Ozon* [4], *Яндекс Маркет* [5] и *МегаМаркет* [6]. Несмотря на общую цель — предоставить пользователю возможность найти и купить товар, решения различаются по функциональным возможностям.

Для анализа выбраны следующие ключевые критерии:

— **Поиск товаров (группировка по товарам)** — пользователь видит список уникальных товаров, а все предложения от разных продавцов скрыты внутри карточки товара.

— **Поиск предложений (вывод всех предложений сразу)** — пользователь видит сразу все доступные предложения конкретного товара от разных продавцов, без группировки по товарам.

— **Добавление в избранное** — возможность сохранять товары или предложения для последующего просмотра.

— **Написание отзывов** — возможность оставлять оценки и комментарии к товарам или предложениям.

— **Рекомендации** — система персональных или тематических рекомендаций товаров на основе истории просмотров или покупок.

Результаты сравнения представлены в таблице 1.1.

Таблица 1.1 — Сравнение российских маркетплейсов по ключевым функциям

Критерий	**Wildberries**	**Ozon**	**Яндекс Маркет**	**МегаМаркет**
Поиск товаров (группировка по товарам)	Нет	Нет	Есть	Нет
Поиск предложений (вывод всех предложений сразу)	Есть	Есть	Есть	Есть
Добавление в избранное	Есть	Есть	Есть	Есть
Написание отзывов	Есть	Есть	Есть	Есть
Рекомендации	Есть	Есть	Есть	Есть

Таким образом, даже при схожем базовом функционале различия проявляются в способе отображения товаров и предложений.

1.3 Формализация задачи

Необходимо спроектировать и разработать приложение и базу данных для взаимодействия с товарами и предложениями.

Покупатель должен иметь возможность просматривать товары, предложения, свои заказы, отзывы, избранные товары.

Продавец должен иметь возможность просматривать, добавлять и изменять свои предложения и магазины.

Администратор должен иметь возможность добавлять товары и изменять информацию о товарах, предложениях и магазинах.

1.4 Формализация данных

База данных для маркетплейса должна содержать информацию о покупателях, продавцах, магазинах, товарах, предложениях, заказах, отзывах и избранных.

На рисунке 1.1 приведена диаграмма сущность-связь базы данных маркетплейса в нотации Чена.

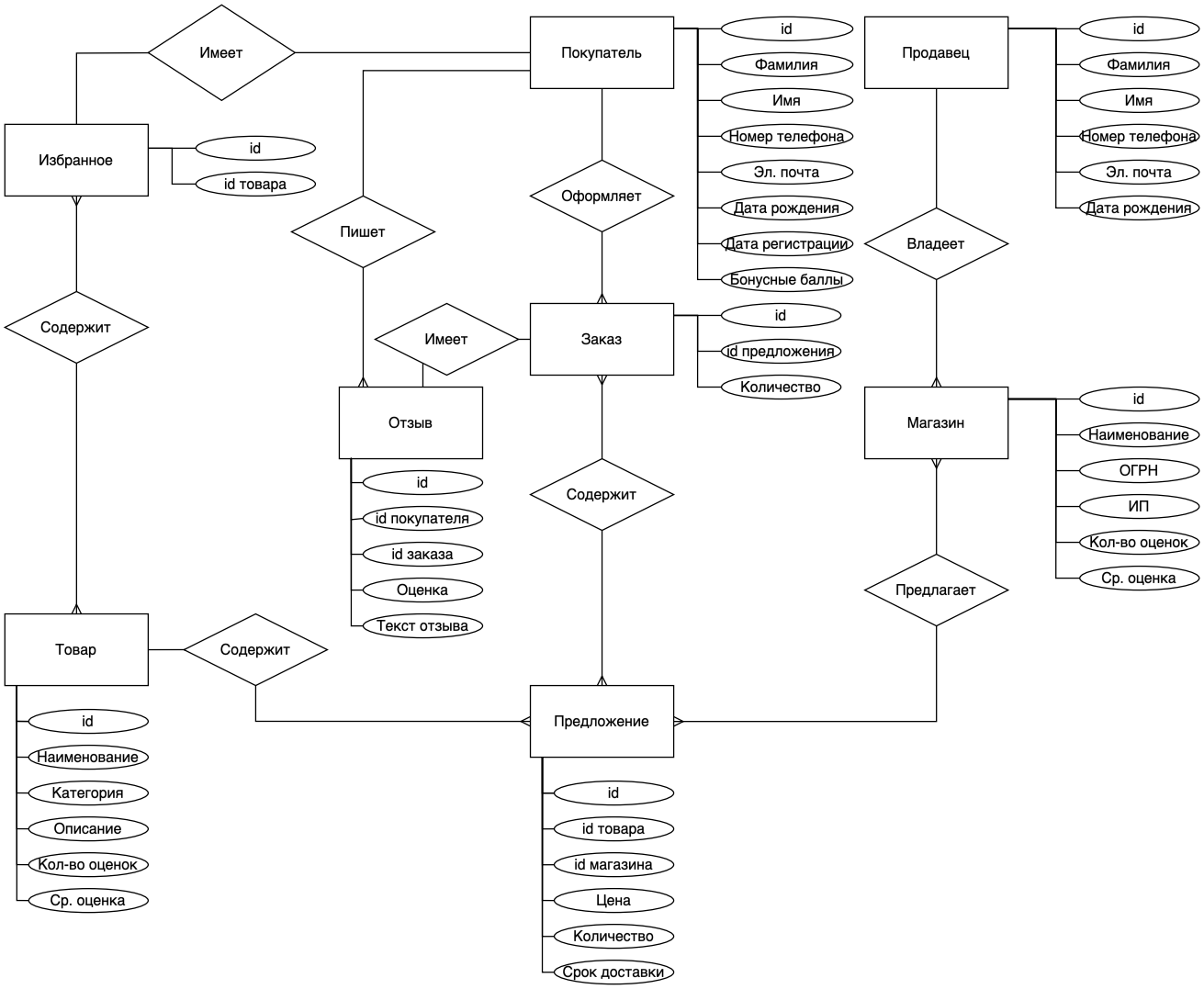

Рисунок 1.1 — Диаграмма сущность-связь базы данных маркетплейса в нотации Чена

Данная диаграмма описывает сущности базы данных, их атрибуты и связи между этими сущностями.

1.5 Формализация ролей

Для взаимодействия с приложением маркетплейса было выделено четыре роли пользователя: гость, покупатель, продавец и администратор.

Гость (неавторизованный пользователь) является начальным состоянием пользователя в приложении. Гость имеет возможность зарегистрироваться, то есть создать покупателя или продавца, и войти в качестве покупателя, продавца или администратора.

На рисунке 1.2 представлена диаграмма вариантов использования для роли гость.

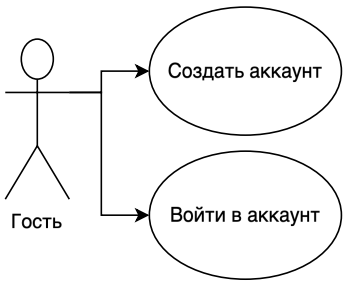

Рисунок 1.2 — Диаграмма вариантов использования для роли Гость

Покупатель имеет возможность просматривать товары, предложения, свои заказы, отзывы, избранные товары, добавлять приложение в заказ, оплачивать заказы, добавлять и удалять товар из изранного, писать и удалять отзывы.

На рисунке 1.3 представлена диаграмма вариантов использования для роли покупатель.

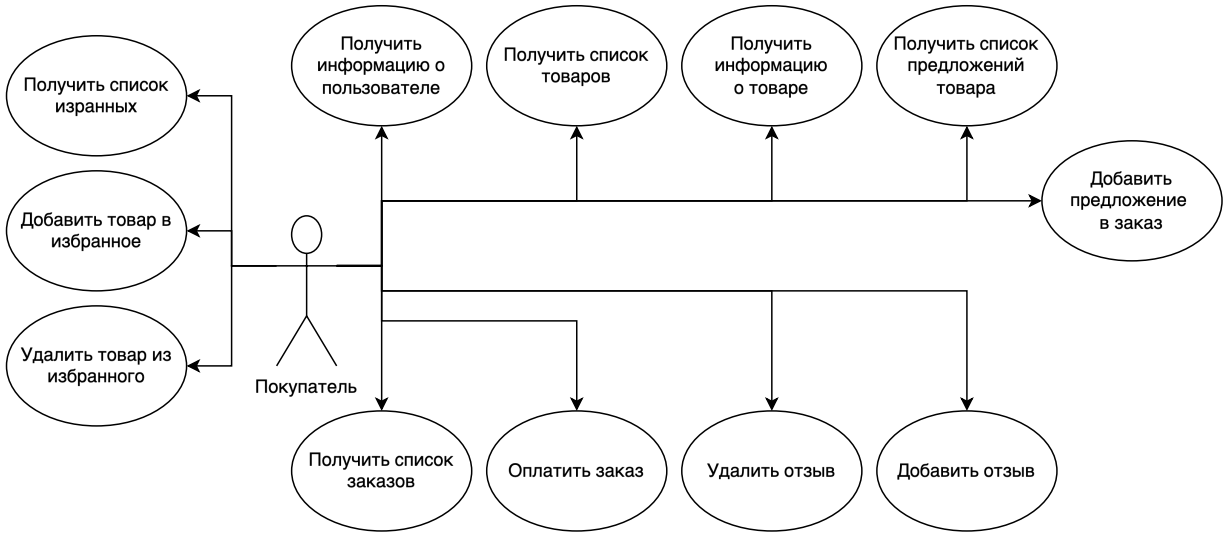

Рисунок 1.3 — Диаграмма вариантов использования для роли Покупатель

Продавец имеет возможность просматривать, добавлять и изменять свои предложения и магазины.

На рисунке 1.4 представлена диаграмма вариантов использования для роли продавец.

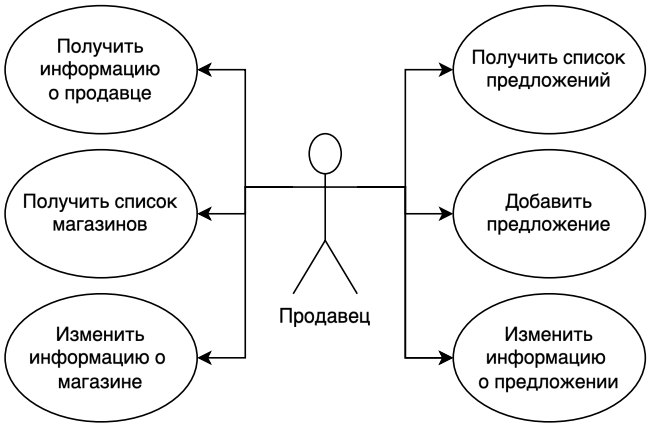

Рисунок 1.4 — Диаграмма вариантов использования для роли Продавец

Администратор имеет возможность добавлять товары и изменять информацию о товарах, предложениях и магазинах.

На рисунке 1.5 представлена диаграмма вариантов использования для роли администратор.

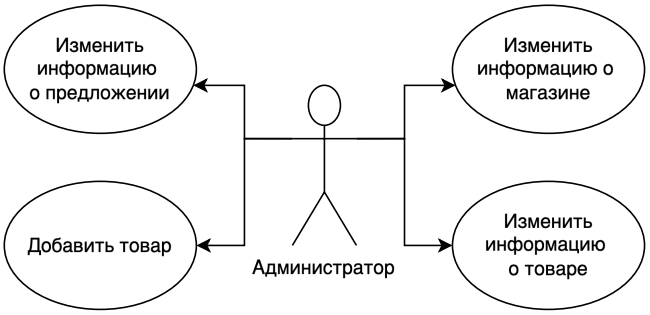

Рисунок 1.5 — Диаграмма вариантов использования для роли Администратор

1.6 Анализ существующих баз данных

1.6.1 Иерархическая модель

Иерархическая модель описывает данные в форме иерархических древовидных структур, где каждая иерархия представляет собой совокупность связанных записей. Унифицированного стандарта языка для этой модели не существует. Наиболее известным языком манипулирования данными является DL/1, используемый в системе IMS. Данная система занимала лидирующее положение на рынке СУБД более 20 лет (с 1965 по 1985 гг.), а её язык DL/1 в течение длительного времени выступал фактическим промышленным стандартом [1].

1.6.2 Сетевая модель

Сетевая модель представляет данные в виде типов записей и поддерживает связь «один-ко-многим» (set type) с помощью указателей. Известна как модель CODASYL DBTG и использует язык обработки данных «запись за записью», встроенный в язык программирования-хост. Язык манипулирования данными (DML) для сетевой модели предложен в DBTG Report (1971) как расширение COBOL [1].

1.6.3 Реляционная модель

Реляционная модель представляет базу данных как совокупность отношений и основывается на следующих условиях [2]:

— **Структурный аспект.** Данные в базе воспринимаются пользователем исключительно как таблицы.

— **Аспект целостности.** Эти таблицы удовлетворяют определённым условиям целостности (которые будут рассмотрены в конце раздела).

— **Аспект обработки.** В распоряжении пользователя имеются операторы манипулирования таблицами (например, предназначенные для поиска данных), которые генерируют новые таблицы на основании уже имеющихся и среди которых есть, по крайней мере, операторы *сокращения* (restrict), *проекции* (project) и *объединения* (join).

1.6.4 Постреляционная модель

Постреляционные модели — системы с более общим подходом к данным, чем реляционная модель. Они не ограничены принципом информации Кодда и допускают хранение данных не только в виде отношений. Некоторые расширяют реляционные системы поддержкой графов (например, GraphDB [7]), другие комбинируют реляционные и дореляционные возможности (PICK, MUMPS). Альтернативный подход — модель ресурсного пространства (RSM) на основе многомерной классификации.

1.6.5 Анализ моделей баз данных

Для проведения анализа сформулируем следующие критерии, выполнение которых необходимо для выполнения поставленых задач:

1) Данные описываются стандартизированным языком запросов;
2) Есть универсальная модель связей;
3) Логика не зависит от физического хранения.

Результаты сравнения моделей баз данных указаны в таблице 1.2.

Таблица 1.2 — Сравнение моделей баз данных по ключевым критериям

Модель БД	**Критерий 1**	**Критерий 2**	**Критерий 3**
Иерархическая	Нет	Нет	Нет
Сетевая	Нет	Частично	Нет
Реляционная	Да	Да	Да
Постреляционная	Частично	Да	Да

В ходе сравнения была выбрана база данных с реляционной моделью, так как она удовлетворяет перечисленным критериям.

Вывод

В аналитическом разделе был проведен анализ предметоной области, сравнительный анализ существующих решений и анализ существующих баз данных. Была формализована задача и требования к разрабатываемой базе данных и приложению. Была проведена формализация данных и ролей в базе данных и приложении.

2 Конструкторский раздел

В данном разделе будет проведено описание сущностей базы данных, описание атрибутов, типов данных и ограничений, проектирование базы данных, описание ролевой модели.

2.1 Описание сущностей базы данных

В таблицах 2.1-2.8 приведено описание сущностей баз данных.

Таблица 2.1 — Описание сущности «Покупатель»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
Фамилия	surname	Строковый	Обязателен; не пустая строка
Имя	name	Строковый	Обязателен; не пустая строка
Номер телефона	phone	Строковый	Обязателен; уникален; формат телефонного номера
Эл. почта	email	Строковый	Обязателен; уникален; корректный формат email
Дата рождения	birthdate	Дата	Может быть пустым
Дата регистрации	registration_date	Дата	Обязателен; по умолчанию — текущая дата
Бонусные баллы	bonus	Целочисленный	По умолчанию 0

Таблица 2.2 — Описание сущности «Продавец»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
Фамилия	surname	Строковый	Обязателен; не пустая строка
Имя	name	Строковый	Обязателен; не пустая строка
Номер телефона	phone	Строковый	Уникален; формат телефонного номера
Эл. почта	email	Строковый	Уникален; корректный формат email
Дата рождения	birthdate	Дата	Может быть пустым

Таблица 2.3 — Описание сущности «Магазин»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
Наименование	name	Строковый	Обязателен; уникален; не пустая строка
ОГРН	ogrn	Строковый	Уникален; обязательное поле
ИП	ip	Строковый	Может быть пустым
Количество оценок	rating_count	Целочисленный	По умолчанию 0
Средняя оценка	rating_avg	Числовой (дробный)	По умолчанию 0.0; диапазон 0–5

Таблица 2.4 — Описание сущности «Товар»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
Наименование	name	Строковый	Обязателен; не пустая строка
Категория	category	Строковый	Обязателен
Описание	description	Текстовый	Может быть пустым
Количество оценок	rating_count	Целочисленный	По умолчанию 0
Средняя оценка	rating_avg	Числовой (дробный)	По умолчанию 0.0; диапазон 0–5

Таблица 2.5 — Описание сущности «Предложение»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
ID товара	product_id	UUID	Внешний ключ на «Товар»
ID магазина	store_id	UUID	Внешний ключ на «Магазин»
Цена	price	Числовой (дробный)	Обязателен; >0
Количество	quantity	Целочисленный	Обязателен; >=0
Срок доставки	delivery_time	Целочисленный	Количество дней; может быть пустым

Таблица 2.6 — Описание сущности «Заказ»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
ID предложения	offer_id	UUID	Внешний ключ на «Предложение»
Количество	quantity	Целочисленный	Обязателен; >0

Таблица 2.7 — Описание сущности «Избранное»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
ID покупателя	customer_id	UUID	Внешний ключ на «Покупатель»
ID товара	product_id	UUID	Внешний ключ на «Товар»

Таблица 2.8 — Описание сущности «Отзыв»

Атрибут	**Значение**	**Тип данных**	**Описание**
Идентификатор	id	UUID	Первичный ключ
ID покупателя	customer_id	UUID	Внешний ключ на «Покупатель»
ID товара	product_id	UUID	Внешний ключ на «Товар»
Оценка	rating	Числовой	Обязательна; диапазон 0–5
Текст отзыва	text	Текстовый	Может быть пустым; длина ограничена

2.2 Диаграмма проектируемой базы данных

На рисунке 2.1 приведена диаграмма сущность-связь проектируемой базы данных в нотации Чена.

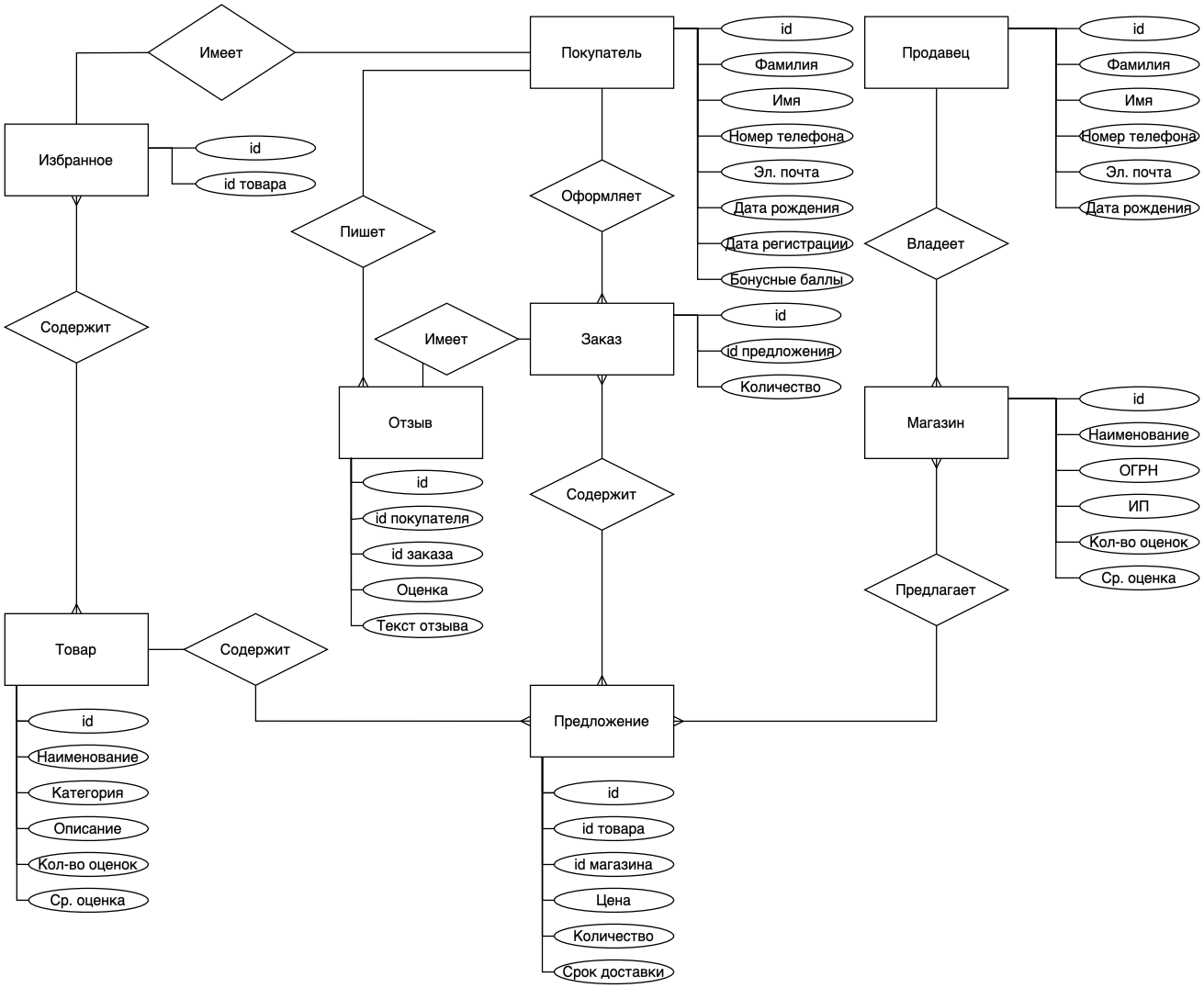

Рисунок 2.1 — Диаграмма сущность-связь проектируемой базы данных в нотации Чена

2.3 Описание проектируемой ролевой модели на уровне БД

Определены следующие роли:

1) **Пользователь** (marketplace_user). Роль позволяет просматривать данные таблиц товаров, предложений, магазинов. Внесение изменений недоступно.

2) **Менеджер** (marketplace_manager). Эта роль включает в себя все возможности пользователя. Дополнительно менеджер получает право на просмотр и редактирование всех таблиц базы данных, а также добавление товаров.

3) **Администратор** (marketplace_admin). Эта роль обладает полным доступом и не имеет ограничений.

Вывод

В данном разделе было проведено описание сущностей базы данных, описание атрибутов, типов данных и ограничений, проектирование базы данных, описание ролевой модели.

3 Технологический раздел

В данном разделе будет проведен выбор средств реализации приложения, языка программирования, системы управления базой данных. Будет приведена архитектура приложения, листинги кода для создания таблиц базы данных и листинги взаимодействия приложения с базой данных.

3.1 Выбор средств реализации

3.1.1 Выбор языка программирования

В качестве языка программирования был выбран C# [9] по следующим причинам:

1) наличие библиотек Npgsql [11] и Dapper [12] для взаимодействия с PostgreSQL [10];
2) наличие библиотеки Serilog для логирования;
3) наличие библиотеки Xunit для тестирования.

3.1.2 Выбор системы управления базами данных

Наиболее популярными реляционными системами управления базами данных являются [13]:

1) PostgreSQL;
2) MySQL;
3) MariaDB;
4) SQLite;
5) Oracle Database;
6) Microsoft SQL Server.

Для выбора реляционной системы управления базами данных воспользуемся следующими критериями:

1) Есть возможность создавать собственные типы данных и операторы.
2) СУБД имеет неограниченную лицензию.

Результаты сравнения реляционных систем управления базами данных по перечисленным критериям указаны в таблице 3.1.

Таблица 3.1 — Сравнение СУБД

Модель БД	**Критерий 1**	**Критерий 2**
PostgreSQL	Да	Да
MySQL	Нет	Нет
MariaDB	Нет	Нет
SQLite	Нет	Да
Oracle Database	Да	Нет
Microsoft SQL Server	Да	Нет

В результате анализа для реализации поставленной задачи была выбрана система управления базами данных **PostgreSQL**, потому что она удовлетворяет перечисленным критериям.

3.2 Архитектура приложения

На рисунке 3.1 приведена подробная UML-диаграмма архитектуры приложения с разбиением по слоям согласно чистой архитектуре.

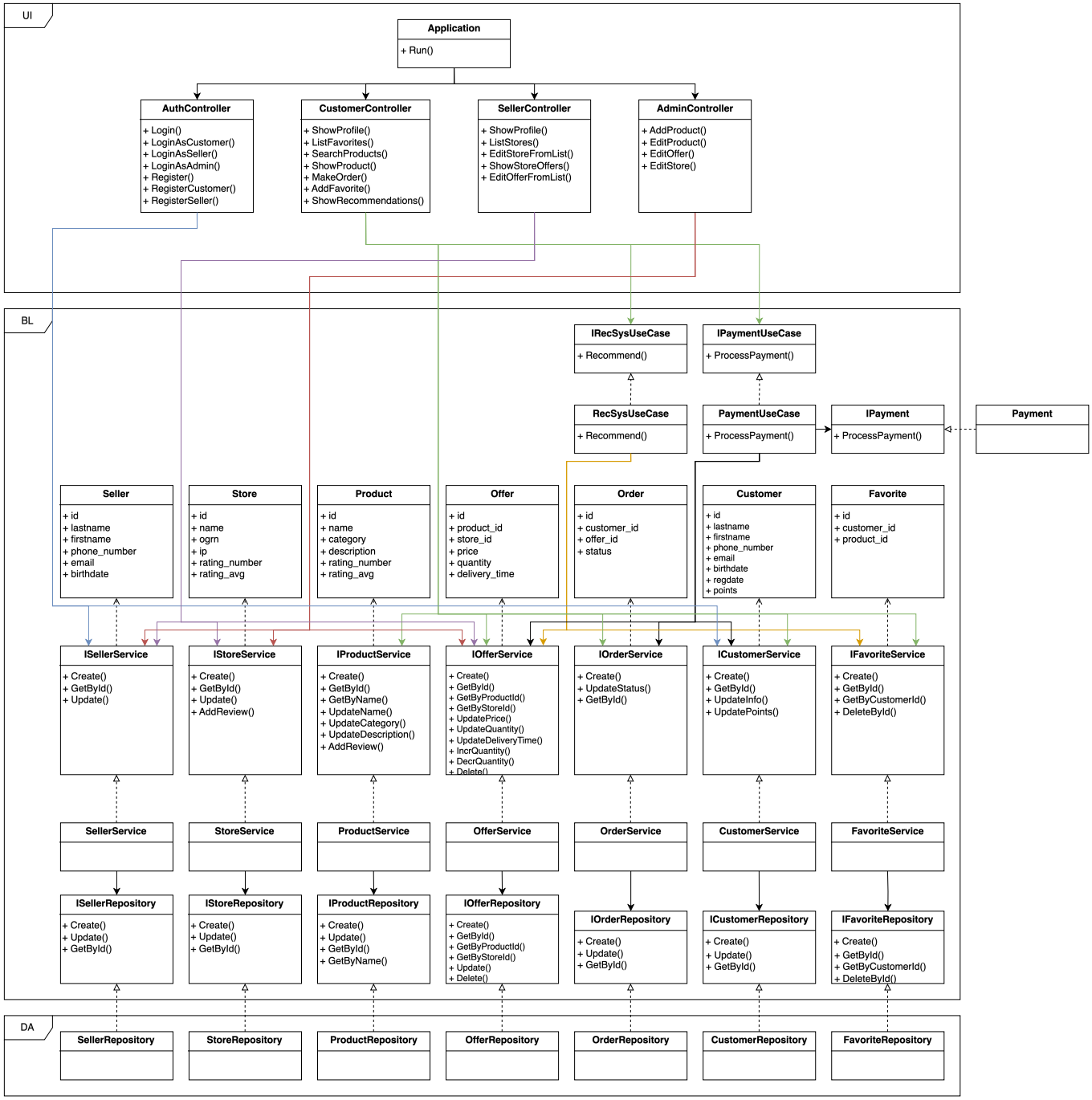

Рисунок 3.1 — UML-диаграмма архитектуры приложения

3.3 Реализация базы данных

В листингах 3.1-3.8 представлена реализация сущностей БД с использованием SQL.

Листинг 3.1 — Таблица customers

```
CREATE TABLE public.customers (
id uuid NOT NULL,
first_name varchar(100) NOT NULL,
last_name varchar(100) NOT NULL,
phone varchar(20) NULL,
email varchar(100) NULL,
birth_date date NOT NULL,
registered_at timestamp NOT NULL,
points int4 NOT NULL,
CONSTRAINT customers_pkey PRIMARY KEY (id)
);
```

Листинг 3.2 — Таблица favorites

```
CREATE TABLE public.favorites (
id uuid NOT NULL,
customer_id uuid NOT NULL,
product_id uuid NOT NULL,
CONSTRAINT favorites_pkey PRIMARY KEY (id)
);

ALTER TABLE public.favorites ADD CONSTRAINT
   favorites_customer_id_fkey
FOREIGN KEY (customer_id) REFERENCES public.customers(id);

ALTER TABLE public.favorites ADD CONSTRAINT
   favorites_product_id_fkey
FOREIGN KEY (product_id) REFERENCES public.products(id);
```

Листинг 3.3 — Таблица offers

```
CREATE TABLE public.offers (
id uuid NOT NULL,
product_id uuid NOT NULL,
store_id uuid NOT NULL,
price numeric(18, 2) NOT NULL,
quantity int4 NOT NULL,
delivery_time int4 NOT NULL,
CONSTRAINT offers_pkey PRIMARY KEY (id)
);

ALTER TABLE public.offers ADD CONSTRAINT offers_product_id_fkey
FOREIGN KEY (product_id) REFERENCES public.products(id);

ALTER TABLE public.offers ADD CONSTRAINT offers_store_id_fkey
FOREIGN KEY (store_id) REFERENCES public.stores(id);
```

Листинг 3.4 — Таблица orders

```
CREATE TABLE public.orders (
id uuid NOT NULL,
customer_id uuid NOT NULL,
offer_id uuid NOT NULL,
quantity int4 NOT NULL,
status varchar(20) NOT NULL,
CONSTRAINT orders_pkey PRIMARY KEY (id)
);

ALTER TABLE public.orders ADD CONSTRAINT
   orders_customer_id_fkey
FOREIGN KEY (customer_id) REFERENCES public.customers(id);

ALTER TABLE public.orders ADD CONSTRAINT orders_offer_id_fkey
FOREIGN KEY (offer_id) REFERENCES public.offers(id);
```

Листинг 3.5 — Таблица products

```
CREATE TABLE public.products (
id uuid NOT NULL,
"name" varchar(200) NOT NULL,
category varchar(100) NULL,
description text NULL,
rating_count int4 DEFAULT 0 NULL,
avg_rating float8 DEFAULT 0.0 NULL,
CONSTRAINT products_pkey PRIMARY KEY (id)
);
```

Листинг 3.6 — Таблица reviews

```
CREATE TABLE public.reviews (
id uuid NOT NULL,
customer_id uuid NOT NULL,
order_id uuid NOT NULL,
review_text text NOT NULL,
rating int4 DEFAULT 0 NOT NULL,
CONSTRAINT reviews_pkey PRIMARY KEY (id)
);

ALTER TABLE public.reviews ADD CONSTRAINT
   reviews_customer_id_fkey
FOREIGN KEY (customer_id) REFERENCES public.customers(id) ON
   DELETE CASCADE;

ALTER TABLE public.reviews ADD CONSTRAINT reviews_order_id_fkey
FOREIGN KEY (order_id) REFERENCES public.orders(id) ON DELETE
   CASCADE;
```

Листинг 3.7 — Таблица sellers

```
CREATE TABLE public.sellers (
id uuid NOT NULL,
first_name varchar(100) NOT NULL,
last_name varchar(100) NOT NULL,
phone varchar(20) NULL,
email varchar(100) NULL,
birth_date date NOT NULL,
CONSTRAINT sellers_pkey PRIMARY KEY (id)
);
```

Листинг 3.8 — Таблица stores

```
CREATE TABLE public.stores (
id uuid NOT NULL,
owner_seller_id uuid NOT NULL,
"name" varchar(200) NOT NULL,
ogrn varchar(50) NULL,
rating_count int4 DEFAULT 0 NULL,
avg_rating float8 DEFAULT 0.0 NULL,
CONSTRAINT stores_pkey PRIMARY KEY (id)
);

ALTER TABLE public.stores ADD CONSTRAINT
   stores_owner_seller_id_fkey
FOREIGN KEY (owner_seller_id) REFERENCES public.sellers(id);
```

В листинге 3.9 представлена реализация ролевой модели базы данных.

Листинг 3.9 — Реализация ролевой модели в PostgreSQL

```
CREATE ROLE marketplace_user NOINHERIT;
CREATE ROLE marketplace_manager NOINHERIT;
CREATE ROLE marketplace_admin NOINHERIT;

GRANT SELECT ON products TO marketplace_user;
GRANT SELECT ON offers TO marketplace_user;
GRANT SELECT ON stores TO marketplace_user;

GRANT marketplace_user TO marketplace_manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON products TO
   marketplace_manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON offers TO
   marketplace_manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON stores TO
   marketplace_manager;

GRANT marketplace_manager TO marketplace_admin;
GRANT ALL PRIVILEGES ON DATABASE marketplace_db TO
   marketplace_admin;
```

3.4 Программный интерфейс доступа

Для доступа к базе данных использовались библиотеки Npgsql и Dapper. В листингах 3.10-3.17 представлена реализация репозиториев для доступа к данным.

Листинг 3.10 — CustomerRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class CustomerRepository : ICustomerRepository
{
  private readonly NpgsqlConnection _connection;

  public CustomerRepository(DapperContext context)
  {
    _connection = context.Connection;
  }

  public Customer Create(Customer customer)
  {
    var sql = @"
    INSERT INTO customers (id, first_name, last_name, phone,
       email, birth_date, registered_at, points)
    VALUES (@Id, @FirstName, @LastName, @Phone, @Email,
       @BirthDate, @RegisteredAt, @Points)";
    _connection.Execute(sql, customer);
    return customer;
  }

  public Customer? GetById(CustomerId customerId)
  {
    var sql = @"
    SELECT id,
    first_name AS FirstName,
```

```
        last_name AS LastName,
        phone AS Phone,
        email AS Email,
        birth_date AS BirthDate,
        registered_at AS RegisteredAt,
        points AS Points
        FROM customers
        WHERE id = @Id";
        return _connection.QuerySingleOrDefault<Customer>(sql, new
            { Id = customerId.Id });
    }

    public Customer? GetByEmail(string email)
    {
        var sql = @"
        SELECT id,
        first_name AS FirstName,
        last_name AS LastName,
        phone AS Phone,
        email AS Email,
        birth_date AS BirthDate,
        registered_at AS RegisteredAt,
        points AS Points
        FROM customers
        WHERE email = @email";
        return _connection.QuerySingleOrDefault<Customer>(sql, new
            { email });
    }

    public Customer Update(Customer customer)
    {
        var sql = @"
        UPDATE customers
        SET first_name=@FirstName,
        last_name=@LastName,
        phone=@Phone,
        email=@Email,
        birth_date=@BirthDate,
        points=@Points
        WHERE id=@Id";
```

```
        var rows = _connection.Execute(sql, customer);
        if (rows == 0)
        throw new RepositoryException($"Customer with id {customer.
            Id} not found");
        return customer;
    }
}
```

Листинг 3.11 — FavoriteRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class FavoriteRepository : IFavoriteRepository
{
    private readonly NpgsqlConnection _connection;

    public FavoriteRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Favorite Create(Favorite favorite)
    {
        var sql = @"
        INSERT INTO favorites (id, customer_id, product_id)
        VALUES (@Id, @CustomerId, @ProductId)";
        _connection.Execute(sql, favorite);
        return favorite;
    }

    public Favorite? GetById(FavoriteId favoriteId)
    {
        var sql = @"
        SELECT id,
        customer_id AS CustomerId,
```

```
    product_id AS ProductId
    FROM favorites
    WHERE id=@Id";
    return _connection.QuerySingleOrDefault<Favorite>(sql, new
        { Id = favoriteId.Id });
}

public List<Favorite> GetByCustomerId(CustomerId customerId)
{
    var sql = @"
    SELECT id,
    customer_id AS CustomerId,
    product_id AS ProductId
    FROM favorites
    WHERE customer_id = @CustomerId";
    return _connection.Query<Favorite>(sql, new { CustomerId =
        customerId.Id }).ToList();
}

public List<FavoriteDetailsDto>
    GetFavoriteDetailsByCustomerId(CustomerId customerId)
{
    var sql = @"
    SELECT f.id as FavoriteId,
    f.customer_id as CustomerId,
    p.id as ProductId,
    p.name as ProductName,
    p.description as ProductDescription
    FROM favorites f
    JOIN products p ON f.product_id = p.id
    WHERE f.customer_id = @CustomerId";
    return _connection.Query<FavoriteDetailsDto>(sql, new {
        CustomerId = customerId.Id }).ToList();
}

public Favorite? DeleteById(FavoriteId favoriteId)
{
    var favorite = GetById(favoriteId);
    if (favorite == null) return null;
    var sql = "DELETE FROM favorites WHERE id=@Id";
```

```
        _connection.Execute(sql, new { Id = favoriteId.Id });
        return favorite;
    }
}
```

Листинг 3.12 — OfferRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class OfferRepository : IOfferRepository
{
    private readonly NpgsqlConnection _connection;

    public OfferRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Offer Create(Offer offer)
    {
        var sql = @"
        INSERT INTO offers (id, product_id, store_id, price,
            quantity, delivery_time)
        VALUES (@Id, @ProductId, @StoreId, @Price, @Quantity,
            @DeliveryTime)";
        _connection.Execute(sql, offer);
        return offer;
    }

    public Offer? GetById(OfferId offerId)
    {
        var sql = @"
        SELECT id,
        product_id AS ProductId,
        store_id AS StoreId,
```

```
        price AS Price,
        quantity AS Quantity,
        delivery_time AS DeliveryTime
        FROM offers
        WHERE id=@Id";
        return _connection.QuerySingleOrDefault<Offer>(sql, new {
            Id = offerId.Id });
    }

    public List<Offer> GetByProductId(ProductId productId)
    {
        var sql = @"
        SELECT id,
        product_id AS ProductId,
        store_id AS StoreId,
        price AS Price,
        quantity AS Quantity,
        delivery_time AS DeliveryTime
        FROM offers
        WHERE product_id=@ProductId";
        return _connection.Query<Offer>(sql, new { ProductId =
            productId.Id }).ToList();
    }

    public List<Offer> GetByStoreId(StoreId storeId)
    {
        var sql = @"
        SELECT id,
        product_id AS ProductId,
        store_id AS StoreId,
        price AS Price,
        quantity AS Quantity,
        delivery_time AS DeliveryTime
        FROM offers
        WHERE store_id=@StoreId";
        return _connection.Query<Offer>(sql, new { StoreId =
            storeId.Id }).ToList();
    }

    public Offer Update(Offer offer)
```

```
    {
        var sql = @"
        UPDATE offers
        SET price=@Price,
        quantity=@Quantity,
        delivery_time=@DeliveryTime
        WHERE id=@Id";
        var rows = _connection.Execute(sql, offer);
        if (rows == 0)
        throw new RepositoryException($"Offer with id {offer.Id}
            not found");
        return offer;
    }

    public Offer? DeleteById(OfferId offerId)
    {
        var offer = GetById(offerId);
        if (offer == null) return null;
        var sql = "DELETE FROM offers WHERE id=@Id";
        _connection.Execute(sql, new { Id = offerId.Id });
        return offer;
    }
}
```

Листинг 3.13 — OrderRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class OrderRepository : IOrderRepository
{
    private readonly NpgsqlConnection _connection;

    public OrderRepository(DapperContext context)
    {
        _connection = context.Connection;
```

```
    }

    public Order Create(Order order)
    {
      var sql = @"
      INSERT INTO orders (id, customer_id, offer_id, quantity,
         status)
      VALUES (@Id, @CustomerId, @OfferId, @Quantity, @Status)";
      _connection.Execute(sql, order);
      return order;
    }

    public Order? GetById(OrderId orderId)
    {
      var sql = @"
      SELECT id,
      customer_id AS CustomerId,
      offer_id AS OfferId,
      quantity AS Quantity,
      status AS Status
      FROM orders
      WHERE id=@Id";
      return _connection.QuerySingleOrDefault<Order>(sql, new {
         Id = orderId.Id });
    }

    public List<Order> GetByCustomerId(CustomerId customerId)
    {
      var sql = @"
      SELECT id,
      customer_id AS CustomerId,
      offer_id AS OfferId,
      quantity AS Quantity,
      status AS Status
      FROM orders
      WHERE customer_id=@Id";
      return _connection.Query<Order>(sql, new { Id = customerId.
         Id }).ToList();
    }
```

```
public Order Update(Order order)
{
  var sql = @"
  UPDATE orders
  SET quantity=@Quantity,
  status=@Status
  WHERE id=@Id";
  var rows = _connection.Execute(sql, order);
  if (rows == 0)
  throw new RepositoryException($"Order with id {order.Id}
     not found");
  return order;
}

public List<OrderDetailsDto> GetOrderDetailsByCustomerId(
   CustomerId customerId)
{
  var sql = @"
  SELECT
  o.id AS OrderId,
  o.status AS Status,
  o.quantity AS Quantity,

  p.id AS ProductId,
  p.name AS ProductName,
  p.description AS ProductDescription,

  ofr.id AS OfferId,
  ofr.price AS OfferPrice,
  ofr.quantity AS OfferQuantity,

  s.id AS StoreId,
  s.name AS StoreName,

  r.id AS ReviewId,
  r.rating AS Rating,
  r.review_text AS ReviewText

  FROM orders o
  JOIN offers ofr ON o.offer_id = ofr.id
```

```
        JOIN products p ON ofr.product_id = p.id
        JOIN stores s ON ofr.store_id = s.id
        LEFT JOIN reviews r ON o.id = r.order_id
        WHERE o.customer_id = @CustomerId";
        return _connection.Query<OrderDetailsDto>(sql, new {
            CustomerId = customerId.Id }).ToList();
    }
}
```

Листинг 3.14 — ProductRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class ProductRepository : IProductRepository
{
    private readonly NpgsqlConnection _connection;

    public ProductRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Product Create(Product product)
    {
        var sql = @"
        INSERT INTO products (id, name, category, description,
            rating_count, avg_rating)
        VALUES (@Id, @Name, @Category, @Description, @RatingCount,
            @AvgRating)";
        _connection.Execute(sql, product);
        return product;
    }

    public Product? GetById(ProductId productId)
    {
```

```
        var sql = @"
        SELECT id,
        name AS Name,
        category AS Category,
        description AS Description,
        rating_count AS RatingCount,
        avg_rating AS AvgRating
        FROM products
        WHERE id=@Id";
        return _connection.QuerySingleOrDefault<Product>(sql, new {
            Id = productId.Id });
    }

    public List<Product> GetByName(string name)
    {
        var sql = @"
        SELECT id,
        name AS Name,
        category AS Category,
        description AS Description,
        rating_count AS RatingCount,
        avg_rating AS AvgRating
        FROM products
        WHERE name ILIKE @Name";
        return _connection.Query<Product>(sql, new { Name = $"%{
            name}%" }).ToList();
    }

    public Product Update(Product product)
    {
        var sql = @"
        UPDATE products
        SET name=@Name,
        category=@Category,
        description=@Description,
        rating_count=@RatingCount,
        avg_rating=@AvgRating
        WHERE id=@Id";
        var rows = _connection.Execute(sql, product);
        if (rows == 0)
```

```
            throw new RepositoryException($"Product with id {product.Id
                } not found");
            return product;
        }
    }
```

Листинг 3.15 — ReviewRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class ReviewRepository : IReviewRepository
{
    private readonly NpgsqlConnection _connection;

    public ReviewRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Review Create(Review review)
    {
        var sql = @"
        INSERT INTO reviews (id, customer_id, order_id, rating,
            review_text)
        VALUES (@Id, @CustomerId, @OrderId, @Rating, @ReviewText)";
        _connection.Execute(sql, review);
        return review;
    }

    public Review? GetById(ReviewId reviewId)
    {
        var sql = @"
        SELECT id,
        customer_id AS CustomerId,
        order_id AS OrderId,
```

```
        rating AS Rating,
        review_text AS ReviewText
        FROM reviews
        WHERE id = @Id";
        return _connection.QuerySingleOrDefault<Review>(sql, new {
            Id = reviewId.Id });
    }

    public List<Review> GetByCustomerId(CustomerId customerId)
    {
        var sql = @"
        SELECT id,
        customer_id AS CustomerId,
        order_id AS OrderId,
        rating AS Rating,
        review_text AS ReviewText
        FROM reviews
        WHERE customer_id = @CustomerId";
        return _connection.Query<Review>(sql, new { CustomerId =
            customerId.Id }).ToList();
    }

    public Review? GetByOrderId(OrderId orderId)
    {
        var sql = @"
        SELECT id,
        customer_id AS CustomerId,
        order_id AS OrderId,
        rating AS Rating,
        review_text AS ReviewText
        FROM reviews
        WHERE order_id = @OrderId";
        return _connection.QuerySingleOrDefault<Review>(sql, new {
            OrderId = orderId.Id });
    }

    public Review? DeleteById(ReviewId reviewId)
    {
        var review = GetById(reviewId);
        if (review == null) return null;
```

```
        var sql = "DELETE FROM reviews WHERE id=@Id";
        _connection.Execute(sql, new { Id = reviewId.Id });
        return review;
    }
}
```

Листинг 3.16 — SellerRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class SellerRepository : ISellerRepository
{
    private readonly NpgsqlConnection _connection;

    public SellerRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Seller Create(Seller seller)
    {
        var sql = @"
        INSERT INTO sellers (id, first_name, last_name, phone,
            email, birth_date)
        VALUES (@Id, @FirstName, @LastName, @Phone, @Email,
            @BirthDate)";
        _connection.Execute(sql, seller);
        return seller;
    }

    public Seller? GetById(SellerId sellerId)
    {
        var sql = @"
        SELECT id,
        first_name AS FirstName,
```

```
        last_name AS LastName,
        phone AS Phone,
        email AS Email,
        birth_date AS BirthDate
        FROM sellers
        WHERE id=@Id";
        return _connection.QuerySingleOrDefault<Seller>(sql, new {
            Id = sellerId.Id });
    }

    public Seller? GetByEmail(string email)
    {
        var sql = @"
        SELECT id,
        first_name AS FirstName,
        last_name AS LastName,
        phone AS Phone,
        email AS Email,
        birth_date AS BirthDate
        FROM sellers
        WHERE email=@email";
        return _connection.QuerySingleOrDefault<Seller>(sql, new {
            email });
    }

    public Seller Update(Seller seller)
    {
        var sql = @"
        UPDATE sellers
        SET first_name=@FirstName,
        last_name=@LastName,
        phone=@Phone,
        email=@Email,
        birth_date=@BirthDate
        WHERE id=@Id";
        var rows = _connection.Execute(sql, seller);
        if (rows == 0)
        throw new RepositoryException($"Seller with id {seller.Id}
            not found");
        return seller;
```

```
    }
}
```

Листинг 3.17 — StoreRepository

```
using Dapper;
using Domain.Models;
using Domain.OutputPorts;
using Domain;
using Npgsql;

namespace DataAccess.Repositories;

public class StoreRepository : IStoreRepository
{
    private readonly NpgsqlConnection _connection;

    public StoreRepository(DapperContext context)
    {
        _connection = context.Connection;
    }

    public Store Create(Store store)
    {
        var sql = @"
        INSERT INTO stores (id, owner_seller_id, name, ogrn,
            rating_count, avg_rating)
        VALUES (@Id, @OwnerSellerId, @Name, @Ogrn, @RatingCount,
            @AvgRating)";
        _connection.Execute(sql, store);
        return store;
    }

    public Store? GetById(StoreId storeId)
    {
        var sql = @"
        SELECT id,
        owner_seller_id AS OwnerSellerId,
        name AS Name,
        ogrn AS Ogrn,
        rating_count AS RatingCount,
```

```
        avg_rating AS AvgRating
        FROM stores
        WHERE id=@Id";
        return _connection.QuerySingleOrDefault<Store>(sql, new {
            Id = storeId.Id });
    }

    public List<Store> GetBySellerId(SellerId sellerId)
    {
        var sql = @"
        SELECT id,
        owner_seller_id AS OwnerSellerId,
        name AS Name,
        ogrn AS Ogrn,
        rating_count AS RatingCount,
        avg_rating AS AvgRating
        FROM stores
        WHERE owner_seller_id = @SellerId";
        return _connection.Query<Store>(sql, new { SellerId =
            sellerId.Id }).ToList();
    }

    public Store Update(Store store)
    {
        var sql = @"
        UPDATE stores
        SET name=@Name,
        ogrn=@Ogrn,
        rating_count=@RatingCount,
        avg_rating=@AvgRating
        WHERE id=@Id";
        var rows = _connection.Execute(sql, store);
        if (rows == 0)
        throw new RepositoryException($"Store with id {store.Id}
            not found");
        return store;
    }
}
```

Листинг 3.18 — DapperContext

```
using Npgsql;
using Dapper;

public class DapperContext : IDisposable
{
  private readonly string _connectionString;
  private readonly NpgsqlConnection _connection;

  public DapperContext(string connectionString)
  {
    _connectionString = connectionString;
    _connection = new NpgsqlConnection(_connectionString);
    _connection.Open();
  }

  public NpgsqlConnection Connection => _connection;

  public void Dispose()
  {
    _connection.Close();
    _connection.Dispose();
  }
}
```

3.5 Тестирование

В таблицах 3.2-3.11 приведены описание тестов и результаты тестирования.

Таблица 3.2 — Тесты сервиса клиентов

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание клиента	Клиент успешно создается в репозитории	Клиент создан, FirstName = Ivan, LastName = Ivanov
2	Получение клиента по Id	Возвращается существующий клиент	Клиент успешно возвращен
3	Получение несуществующего клиента по Id	Бросается исключение IdNotFoundException	Исключение IdNotFoundException сгенерировано
4	Начисление баллов клиенту	Баллы клиента увеличиваются на указанное значение	Баллы увеличены, Points = 15

Таблица 3.3 — Тесты сервиса избранного

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание избранного	Объект добавлен в репозиторий	Создано избранное, CustomerId и ProductId совпадают
2	Получение избранного по Id	Возвращается существующий объект	Объект успешно возвращен
3	Получение несуществующего объекта	Бросается IdNotFoundException	Исключение сгенерировано
4	Получение избранного по Id клиента	Возвращается список объектов этого клиента	Список содержит 2 объекта с правильным CustomerId
5	Удаление существующего объекта	Объект удален и возвращен	Объект успешно удален
6	Удаление несуществующего объекта	Бросается IdNotFoundException	Исключение сгенерировано

Таблица 3.4 — Тесты сервиса предложений (OfferService)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание предложения	Предложение добавлено в репозиторий	Предложение создано с правильными ProductId, StoreId, Price, Quantity
2	Получение существующего предложения по Id	Предложение возвращается	Предложение успешно возвращено
3	Получение несуществующего предложения	Бросается IdNotFoundException	Исключение сгенерировано
4	Получение предложений по ProductId	Возвращается список предложений	Список содержит 2 предложения с верным ProductId
5	Получение предложений по StoreId	Возвращается список предложений	Список содержит 2 предложения с верным StoreId
6	Обновление цены	Цена обновляется	Цена обновлена на 99
7	Обновление количества	Количество обновляется	Количество обновлено на 42
8	Обновление времени доставки	Время доставки обновляется	Время доставки обновлено на 7
9	Увеличение количества	Количество увеличивается	Количество увеличено на 3
10	Уменьшение количества	Количество уменьшается	Количество уменьшено на 3
11	Удаление существующего предложения	Предложение удалено и возвращено	Предложение успешно удалено
12	Удаление несуществующего предложения	Бросается IdNotFoundException	Исключение сгенерировано

Таблица 3.5 — Тесты сервиса заказов (OrderService)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание заказа	Заказ успешно добавлен	Заказ создан с правильными CustomerId, ProductId и Status
2	Получение заказа по Id	Возвращается существующий заказ	Заказ успешно возвращен
3	Получение несуществующего заказа	Бросается IdNotFoundException	Исключение сгенерировано
4	Обновление статуса заказа	Статус заказа изменен	Статус изменен на 'PAID'
5	Удаление заказа	Заказ удален и возвращен	Заказ успешно удален

Таблица 3.6 — Тесты сервиса платежей (PaymentUseCase)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание платежа	Платеж успешно добавлен	Платеж создан с правильной суммой и OrderId
2	Получение платежа по Id	Возвращается существующий платеж	Платеж успешно возвращен
3	Попытка оплаты несуществующего заказа	Бросается IdNotFoundException	Исключение сгенерировано
4	Обновление статуса платежа	Статус платежа изменен	Статус изменен на 'COMPLETED'

Таблица 3.7 — Тесты сервиса продуктов (ProductService)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание продукта	Продукт добавлен в репозиторий	Продукт создан с правильным названием и категорией
2	Получение продукта по Id	Возвращается существующий продукт	Продукт успешно возвращен
3	Получение несуществующего продукта	Бросается IdNotFoundException	Исключение сгенерировано
4	Обновление названия продукта	Название продукта изменено	Название изменено на 'New Product Name'
5	Обновление категории продукта	Категория продукта изменена	Категория изменена на 'Electronics'
6	Удаление продукта	Продукт удален и возвращен	Продукт успешно удален

Таблица 3.8 — Тесты сервиса рекомендаций (RecSysUseCase)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Получение рекомендаций для клиента	Возвращается список рекомендуемых продуктов	Список содержит 5 продуктов, соответствующих интересам клиента
2	Попытка получения рекомендаций для несуществующего клиента	Бросается IdNotFoundException	Исключение сгенерировано

Таблица 3.9 — Тесты сервиса продавцов (SellerService)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание продавца	Продавец успешно добавлен	Продавец создан с правильным Name и StoreId
2	Получение продавца по Id	Возвращается существующий продавец	Продавец успешно возвращен
3	Обновление информации о продавце	Информация обновлена	Имя и контакты изменены
4	Удаление продавца	Продавец удален	Продавец успешно удален

Таблица 3.10 — Тесты сервиса магазинов (StoreService)

№	Описание теста	Ожидаемый результат	Полученный результат
1	Создание магазина	Магазин успешно добавлен	Магазин создан с правильным Name и Address
2	Получение магазина по Id	Возвращается существующий магазин	Магазин успешно возвращен
3	Обновление информации о магазине	Информация обновлена	Адрес и контакты изменены
4	Удаление магазина	Магазин удален	Магазин успешно удален

Таблица 3.11 — Тесты репозиториев базы данных

№	Описание теста	Ожидаемый результат	Полученный результат
1	Проверка сохранения сущности в репозитории	Сущность добавлена	Сущность присутствует в базе
2	Получение сущности по Id	Сущность возвращена	Сущность успешно получена
3	Попытка получения несуществующей сущности	Бросается исключение	Исключение сгенерировано
4	Обновление сущности	Сущность обновлена	Данные изменены корректно
5	Удаление сущности	Сущность удалена	Сущность удалена из базы

Вывод

В данном разделе был проведен выбор средств реализации приложения, языка программирования, системы управления базой данных. Были приведены архитектура приложения, листинги кода для создания таблиц базы данных и листинги взаимодействия приложения с базой данных.

4 Исследовательский раздел

4.1 Технические характеристики

Характеристики устройства, на котором выполнялись замеры [8]:

1) операционная система — macOS Sonoma 14.1 (23B2073);
2) процессор — Apple M3;
3) оперативная память — 16 Гб.

4.2 Цель исследования

Целью исследования является сравнение времени выполнения запроса *GetOfferDetailsByProductId* с индексом *idx_offers_product_id* и без него в зависимости от количества строк в таблице *offers*.

4.3 Результаты исследования

В таблице 4.1 представлены результаты сравнения врмени выполнения запроса с индексом и без него в зависимости от количества строк в таблице *offers*.

Таблица 4.1 — Сравнение времени выполнения запроса с индексом и без него

Количество строк	Без индекса (мс)	С индексом (мс)
100	1.2	1.1
200	2.3	1.9
300	3.5	2.7
400	4.8	3.5
500	6.1	4.2
600	7.5	4.9
700	8.9	5.4
800	10.3	6.0
900	11.8	6.6
1000	13.5	7.2

На рисунке 4.1 представлены графики зависимости времени врмени выполнения запроса с индексом и без него в зависимости от количества строк в таблице *offers*.

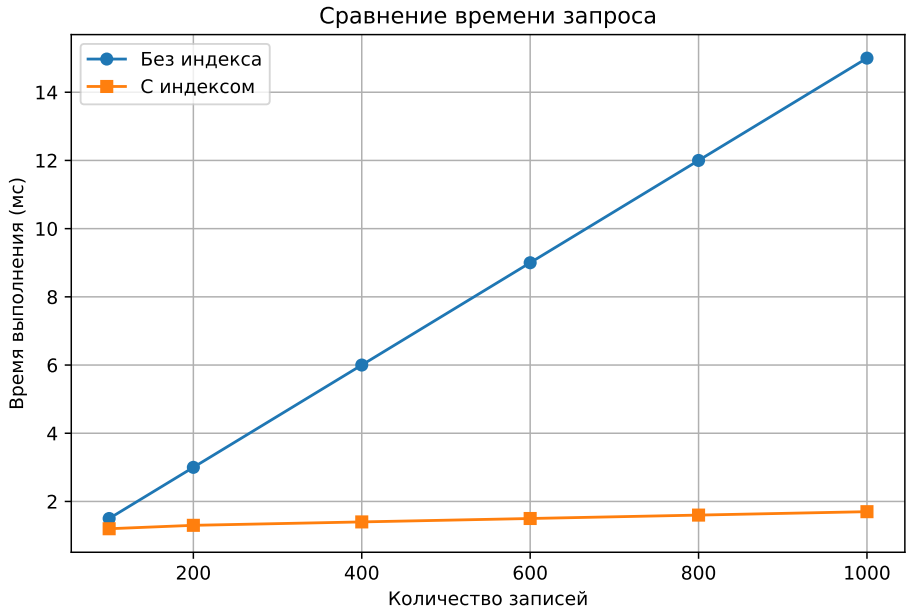

Рисунок 4.1 — Графики зависимости времени выполнения запроса с индексом и без него от количества строк

Вывод

Из проведенного исследования следует, что наличие индекса (по умолчанию *B-tree*) подходящего под частый запрос (запрос предложений — один из основных запросов маркетплейса) уменьшает время выполнения запроса относительно запроса без индекса (обычный последовательный поиск *sequential scan*), тем самым ускоряя работу приложения и повышая отзывчивость для пользователя, что улучшает его пользовательский опыт.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы поставленная цель была достигнута: разработано программное обеспечение маркетплейса, которое позволяет продавцам размещать предложения о продаже товаров, а покупателям находить интересующие товары и покупать их.

Были выполнены следующие задачи:

1) проведен анализ предметной области и существующих баз данных;
2) спроектированы база данных и программное обеспечение;
3) реализованы спроектированные база данных и программное обеспечение;
4) проведено исследование разработанной базы данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ramez Elmasri, Shamkant B. Navathe «FUNDAMENTALS OF Database Systems». 7th edition. — Pearson, 2016. — 1273 с.

2. Date C.J. «An Introduction to Database Systems». 8th edition. — Addison-Wesley, 2004. — 1024 p.

3. Wildberries — официальный сайт маркетплейса [Электронный ресурс]. Режим доступа: https://www.wildberries.ru/ (дата обращения 07.04.2025).

4. Ozon — официальный сайт маркетплейса [Электронный ресурс]. Режим доступа: https://www.ozon.ru/ (дата обращения 07.04.2025).

5. Яндекс Маркет — официальный сайт маркетплейса [Электронный ресурс]. Режим доступа: https://market.yandex.ru/ (дата обращения 07.04.2025).

6. МегаМаркет — официальный сайт маркетплейса [Электронный ресурс]. Режим доступа: https://megamarket.ru/ (дата обращения 07.04.2025).

7. GraphDB — официальный сайт [Электронный ресурс]. Режим доступа: https://graphdb.ontotext.com/ (дата обращения 06.06.2025).

8. Технические характеристики MacBook Pro [Электронный ресурс]. Режим доступа: https://support.apple.com/en-by/117736 (дата обращения 11.12.24)

9. Microsoft Docs. C# Language Documentation [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата обращения 12.05.2025).

10. PostgreSQL Global Development Group. PostgreSQL Documentation [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/ (дата обращения 15.05.2025).

11. Npgsql — .NET Data Provider for PostgreSQL [Электронный ресурс]. Режим доступа: https://www.npgsql.org/ (дата обращения 15.05.2025).

12. Dapper — A simple object mapper for .NET [Электронный ресурс]. Режим доступа: https://github.com/DapperLib/Dapper (дата обращения 15.05.2025).

13. DB-Engines Ranking [Электронный ресурс]. Режим доступа: https://db-engines.com/en/ (дата обращения 06.06.2025).

Приложение А

Презентация состоит из _ слайдов.