МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность: 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных «Интернет-магазин» с реализацией технологии репликации данных между серверами баз данных

Выполнил студент: Ткачёв Антон Алексеевич

(Ф.И.О.)

Руководитель проекта: ассистент Копыток Д.В.

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

Заведующий кафедрой: к.т.н., доц. Смелов В.В.

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

Консультант: ассистент Копыток Д.В.

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

Нормоконтролер: ассистент Копыток Д.В.

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 3](#_Toc121274402)

[1 Постановка задачи 4](#_Toc121274403)

[1.1 База данных 4](#_Toc121274404)

[1.2 Используемая технология 4](#_Toc121274405)

[2 Разработка модели базы данных 5](#_Toc121274406)

[3 Разработка необходимых объектов 6](#_Toc121274407)

[3.1 Таблицы 6](#_Toc121274408)

[3.2 Процедуры 7](#_Toc121274409)

[3.3 Функции 7](#_Toc121274410)

[3.4 Индексы 8](#_Toc121274411)

[3.5 Пользователи 8](#_Toc121274412)

[3.5 Материальные представления 8](#_Toc121274413)

[4 Описание процесса импорта и экспорта 9](#_Toc121274414)

[5. Тестирование производительности 10](#_Toc121274415)

[5.1 Тестирование индексов на выборке 10](#_Toc121274416)

[5.2 Сравнительный анализ результатов тестирования 11](#_Toc121274417)

[6. Описание технологий 12](#_Toc121274418)

[6.1 Потоковая репликация баз данных 12](#_Toc121274419)

[6.2 Создание распределенных баз данных 13](#_Toc121274420)

[6.3 Логирование и аудит действий 14](#_Toc121274421)

[6.4 Резервное копирование и восстановление 15](#_Toc121274422)

[6.5 Загрузка байтовых файлов 15](#_Toc121274423)

[7 Инструкция пользователя 16](#_Toc121274424)

[Заключение 20](#_Toc121274425)

[Приложение А: Скрипт для создания основных таблиц 22](#_Toc121274426)

[Приложение Б: Конфигурация docker-compose 24](#_Toc121274427)

[Приложение В: Скрипты для инициализаций БД 26](#_Toc121274428)

[Приложение Г: Конфигурация базы данных «Postgres 14» 27](#_Toc121274429)

[Приложение Д: Скрипт для импорта данных в JSON 29](#_Toc121274430)

[Приложение Е: Скрипт для экспорта данных в JSON 31](#_Toc121274431)

[Приложение Ж: Скрипт для экспорта данных в SQL 32](#_Toc121274432)

[Приложение З: Запросы сравнения оптимизации 33](#_Toc121274433)

Введение

Данные всегда играли важную роль в жизни людей и в разные времена их хранили по-разному. В наше время всё больше информации хранится в цифровом виде и для хранения такой информации отлично подходят различные базы данных. Использование базы данных упрощает управление и хранение данных. Одним из примеров успешного внедрения баз данных является использование их различными интернет-магазинами.

На сегодняшний день, самой распространённой является реляционная модель структурирования данных. Реляционная база данных — это база данных, основанная на реляционной модели данных. Она подразумевает под собой описание каждой сущности набором атрибутов, которые хранятся в соответствующих, для этих сущностей, таблицах.

Целью данной курсовой работы является разработка реляционной базы данных «Интернет-магазина». База данных должна содержать необходимые данные о пользователях, товарах, заказах, комментариях, категориях, а также хранить и шифровать данные о паролях.

В качестве базы данных была выбрана «PostgreSQL 14», в связи с ее производительностью и надежностью.

Необходимо создать основные объекты, такие как таблицы, пользователи, роли, индексы. В базе данных должна быть реализована возможность экспорта и импорта данных в таблицы базы данных. Также после создание объектов и добавления данных в таблицы, нужно протестировать базу данных на производительность, вследствие чего добавить необходимые индексы для таблиц. В качестве технологии для курсового проекта была выбрана технологии репликации данных между серверами баз данных.

1. Постановка задачи

1.1 База данных

В соответствии с заданием целью курсового проекта является разработка базы данных для интернет-магазина и объектов базы данных. Была выбрана база данных «PostgreSQL», которая имеет множество технологий для обеспечения безопасности, надежности и распределения нагрузки на серверы и оборудование, что позволяет обеспечить быструю обработку данных и повысить надёжность всей системы в целом для любых задач.

В базе данных «PostgreSQL 14» имеется множество объектов для проектирования базы данных. Основные объекты базы данных «PostgreSQL 14»:

* таблицы, представления и синонимы — используются для хранения данных и доступа к ним.
* синонимы обеспечивают альтернативные имена объектов базы данных.
* индексы — используются для ускорения доступа к данным.
* последовательности — используются для генерации чисел для различных целей.
* триггеры и ограничения целостности — используются для поддержания допустимости вводимых данных.
* привилегии, роли— используются для управления доступом к базе данных и ее использование.

## **1.2 Используемая технология**

В процессе выбора технологии была рассмотрена технология «Репликация данных между серверами СУБД», которая повышает надёжность. Была выбрана потоковая репликация, так как сервер может работать с меньшой задержкой, чем при трансляции файлов.

1. Разработка модели базы данных

Одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие, составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя.

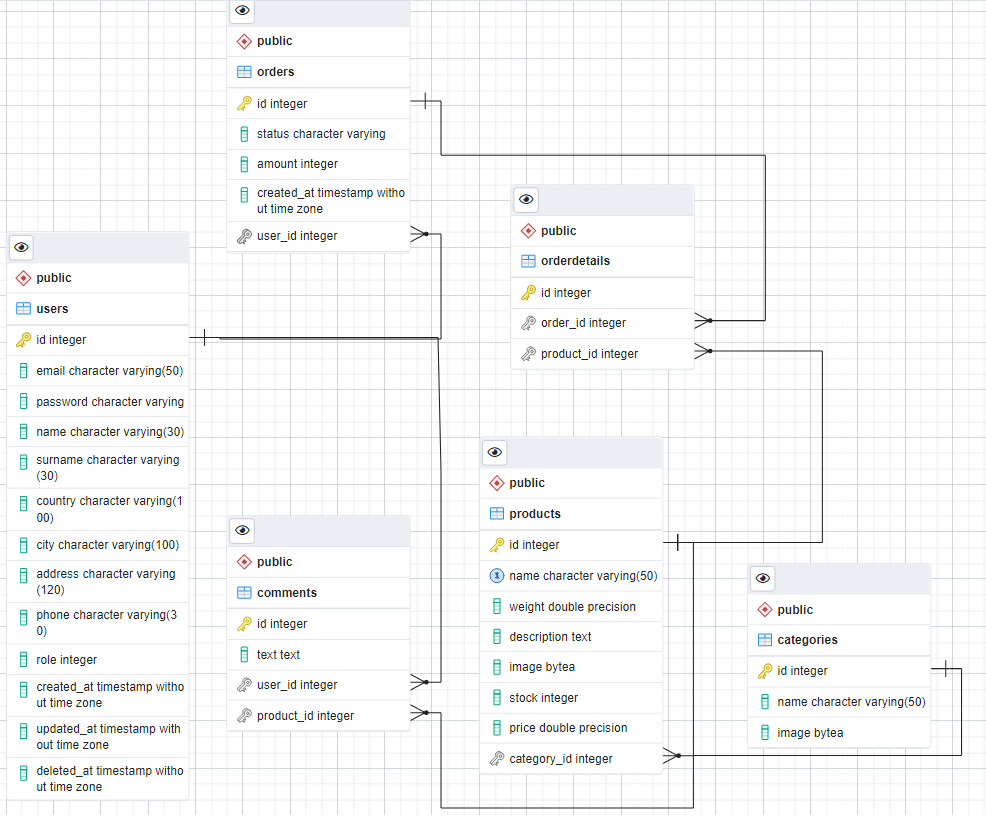


Рисунок 2.1 — Структурная схема базы данных

1. Разработка необходимых объектов

При разработке курсового проекта понадобились следующие объекты:

* Таблицы;
* Хранимые процедуры;
* Хранимые функции;
* Индексы;
* Роли;
* Пользователи;
* Материализованные представления.
  1. Таблицы

Таблицы являются неотъемлемой частью любой реляционной базы данных. Для реализации базы данных «Интернет-магазин» было разработано 6 таблиц, код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица USERS – пользователи:

* id – идентификатор;
* email – электронная почта;
* password – пароль;
* name – имя;
* surname – фамилия;
* country – страна;
* city – город;
* address – адрес;
* phone – телефон;
* role – роль;
* created\_at – дата регистрации;
* updated\_at – дата обнавления;
* deleted\_at – дата удаления

Таблица PRODUCTS – продукты:

* id – идентификатор;
* name – имя;
* weight – вес;
* description – краткое описание;
* image – основное изображение;
* stock – количество;
* price – цена;
* category\_id – категория;

Таблица CATEGORIES – категории:

* id– идентификатор;
* name– имя;
* image– изображение;

Таблица COMMENTS – комментарии:

* id– идентификатор;
* text– комментарий;
* product\_id– идентификатор продукта;
* user\_id– идентификатор пользователя;

Таблица ORDERS – заказы пользователя:

* id– идентификатор заказа;
* status– статус;
* created\_at– дата;
* amount– общая стоимость;
* user\_id– идентификатор пользователя;

Таблица ORDERDETAILS – товары для заказа:

* id– идентификатор
* order\_id– номер заказа
* product\_id– номер товара
  1. Процедуры

Процедура представляет собой модуль, выполняющий одно или несколько действий. Поскольку вызов процедуры в SQL является отдельным исполняемым оператором, блок кода SQL может состоять только из вызова процедуры. Процедуры относятся к числу ключевых компонентов модульного кода, обеспечивающих оптимизацию и повторное использование программной логики.

Для реализации базы данных «Интернет-магазина» было разработано 9 процедур, которые реализуют функции CRUD.

Процедуры, связанные c манипулированием данных в таблицах:

* add\_category — добавление категории;
* add\_comment — добавление комментария;
* add\_person — добавление пользователя;
* add\_product — добавление продукта;
* add\_category\_id\_to\_product — добавление категории для товара
* make\_order — совершение заказа;
* add\_stock\_product — добавление количества продукта;
* change\_status\_order — изменение статуса заказа;
* change\_stock\_in\_product — изменение количества товара;
* change\_comment\_text — изменить текст комментария;
* chage\_user\_email — изменить email пользователя
* denied\_order — отклонение заказа;
* delete\_comment — удаление комментария;

3.3 Функции

SQL-функции выполняют произвольный список операторов SQL и возвращают результат последнего запроса в списке.

Функции, связанные c манипулированием данных в таблицах:

Функции, для получения данных из таблиц:

* get\_all\_comments— получить все комментарии;
* get\_one\_comment — получить комментарий по id;
* get\_all\_category — получить все категории;
* get\_one\_category — получить одну категорию по id;
* get\_all\_products — получить все товары;
* get\_one\_product — получить товар по name;
* get\_all\_orders — получить все заказы;
* get\_one\_order — получить заказ по id;

3.4 Индексы

Индекс это структура базы данных, используемая сервером для быстрого поиска строки в таблице.

Индексы базы данных:

* ix\_categories\_name — индекс имен в категориях;
* ix\_users\_email — индекс почт у пользователей;
* ix\_users\_phone — индекс телефонов у пользователей;

3.5 Пользователи

Для данного решения было добавлено три пользователи с разграниченным доступом.

Первый пользователь «postgres», который имеет доступ ко всем объектам базы данных «vsy», а также является администратором всего кластера баз данных.

Второй пользователь «staff» доступны только операции чтения в базе данных.

Третий пользователь «repluser», которому разрешен доступ только подключение к реплицированным базам данных на других экземплярах.

3.5 Материальные представления

Материализованное представление — физический объект базы данных, содержащий результат выполнения запроса. Позволяет многократно ускорить выполнение запроса обращающихся к большому количеству записей, позволяя за секунды выполнять запросы.

Материализованные базы данных:

– order\_and\_user – материализованное представление пользователей и из заказов

1. Описание процесса импорта и экспорта

База данных обычно имеет не самостоятельную ценность, является частью информационной системы. Независимо от того, как устроена эта система, на противоположном от БД конце находится интерфейс взаимодействия с пользователем, и задача программиста предоставить простой и понятный способ работы с хранящимися в БД данными и объектами.

При всей своей отлаженности и очевидности, классический способ хранения и представления объектов развитой структуры имеет и вполне определенные недостатки и может вызывать проблемы, с которыми сталкивался любой разработчик, пытавшийся реализовать таким способом достаточно сложную систему. В некоторых ситуациях, решить эти проблемы позволяет хранение объектов в виде JSON.

Было создано 3 скрипта позволяющие совершать экспорты и импорты в JSON и SQL. В приложении Д, Е, Ж представлен код скриптов. Для создания фейковых данных использовался язык программирования Python и библиотеки Faker. С помощью их создавались фикстуры.

# 5. Тестирование производительности

Для тестирования производительности были выбраны потенциальные поля для индексации, в приложении З набор запросов для оптимизации. Так как индекс увеличивает производительность при выборке, но уменьшает время для добавление, обновления, удаления, то запросы были созданы покрыть все возможные варианты.

## **5.1 Тестирование индексов на выборке**

Таблица 5.1 – Количество строк данных в таблицах

|  |  |
| --- | --- |
| Таблица | Строк |
| users | 500000 |
| categories | 8 |

Для тестирования будет выполнен поиск пользователя по названию почты и телефону, так как эти строки для поиска и сравнения должны занять без индекса достаточно большое время. Таблица categories намного меньше, чем таблица users, поэтому будет проведен анализ по эффективности индексов на объеме данных.

Таблица 5.2 – Результаты тестирования таблиц

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Без индекса | | С индексом | |
| № запроса | Планирование, мс | Запуск, мс | Планирование, мс | Запуск, мс |
| 1 | 1.772 | 0.088 | 2.897 | 0.037 |
| 2 | 0.071 | 6.031 | 2.197 | 0.049 |
| 3 | 0.074 | 1.961 | 0.103 | 0.072 |
| 4 | 0.031 | 10.325 | 0.044 | 0.857 |
| 5 | 0.033 | 6.095 | 0.028 | 1.198 |

## **5.2 Сравнительный анализ результатов тестирования**

Для построения более объективной и понятной модели анализа была создана гистограмма времени выполнения запросов при индексации и без неё.

График 5.1 – Cравнения общего выполнения плана запроса

Исходя из графика тестирования можно сделать вывод о том, что создание индексов для таблиц могут значительно уменьшить цену запроса до с среднем на 400%. При меньших объемах данных польза от индексов не так, большая. Использовались некластеризованные индексы.

# 6. Описание технологий

В курсовом проекте реализованы данные технологии:

* Потоковая репликация баз данных
* Создание распределенных баз данных
* Логирование и аудит действий
* Резервное копирование и восстановление
* Загрузка байтовых файлов

## **6.1 Потоковая репликация баз данных**

Репликация — это дублирование данных, когда данные с одного сервера полностью повторяются на других. Приложения пишут данные в одну базу данных «PostgreSQL», а изменения автоматически синхронизируются на другие базы.  
Репликация используется для достижения двух целей:

* Повышение отказоустойчивости. Если один из серверов выйдет из строя, то остальные продолжат работу.
* Повышение производительности. Распределение данных по серверам в разных частях страны или мира повышает скорость доступа к данным для местных пользователей.

Потоковая репликация - это репликация, при которой от основного сервера «PostgreSQL» на реплики передается WAL. И каждая реплика затем по этому журналу изменяет свои данные. Для настройки такой репликации все серверы должны быть одной версии, работать на одной ОС и архитектуре.

Для осуществления репликации необходима два свободных хоста, на которых мы должны установить «PostgreSQL» базу данных. Использовал технологию docker контейнеров организованные в compose. Затем необходима с конфигурировать файлы pg\_hba.conf и postgresql.conf.

pg\_hba.conf является файлом конфигурации прослушиваемых IP-адресов и методы подлинности подключения.

postgresql.conf является главным файлом конфигурации инстанса базы данных на хосте. В файле указывается параметры не обходимые для primary и standby баз данных.

На standby базе данных необходима запустить утилиту с определенными параметрами для осуществления репликации между базами данных.

Листинг скриптов создания, инициализации и начала репликации представлены в приложении Б, Д, В.

## **6.2 Создание распределенных баз данных**

Создание распределенных баз данных, может осуществляется разными вариантами. Мною была выбрана технология docker compose для осуществления в пределах LAN и docker swarm для осуществления во внешней сети.

Docker – это открытая платформа для разработки, доставки и эксплуатации приложений, позволяет легко и просто разворачивать изображения приложений на локальном хосте.

Docker compose – это средство для определения и запуска приложений Docker с несколькими контейнерами.

Docker swarm – это встроенная система оркестровки Docker для масштабирования контейнеров в кластере физических машин. Несколько независимых клиентов, на которых работает Docker Engine, объединяют свои ресурсы, образуя рой.

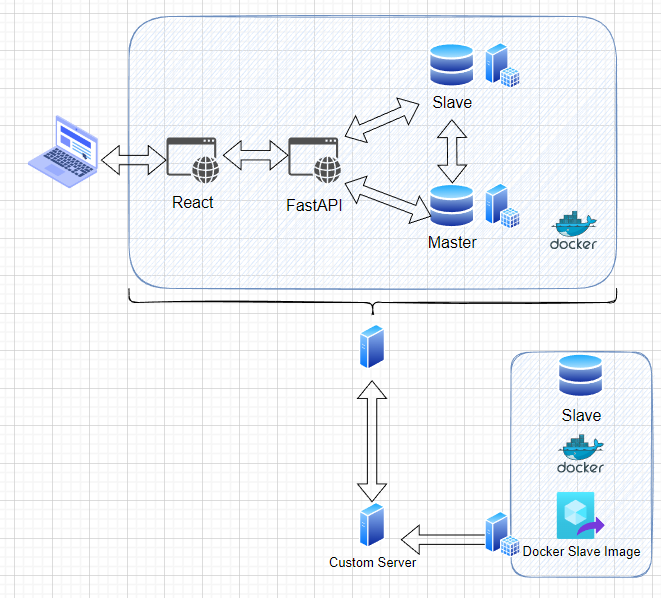
****

Рис. 6.1 – Схема распределенной архитектуры

На рисунке 6.1 изображена текущая архитектура системы, где главный хост хранит Master(Primary) и Slave(Standby) базы данных с клиент-серверными приложениями на одном хосте.

Другой хост подключенный к кластеру физических машин способен подключится к главной базе данных и осуществлять репликацию из него. Также используется технология Docker Image для осуществления легко использования созданной Standby базы данной на другом хосте.

Так архитектура имеет несколько плюсов:

* Возможность испытывать большие нагрузки
* Простота шардинга базы данных
* Безопасность

Также архитектура имеет несколько минусов:

* Большая стоимость
* Сложность управления

Описание создания архитектуры в приложении Б.

## **6.3 Логирование и аудит действий**

Логирование – это запись логов, структурирование и перемещение их в отдельные файлы для быстрого доступа к ним.

Механизм реализации логов и аудитов в базе данных «PostgreSQL» встроен. Сама база данных предоставляет механизм логов, который по умолчанию отключен. Для включения логов необходимо было сконфигурировать файл postgresql.conf и в разделе логов и репортов, поставить нужные значения в приложении Г. На рисунке 6.2 изображен пример логов и аудитов.

|  |
| --- |
| 2022-12-01 16:27:29.917 GMT [76] 172.28.0.1(40790) [unknown]:[unknown]/[unknown]LOG: connection received: host=172.28.0.1 port=40790  2022-12-01 16:27:29.938 GMT [76] 172.28.0.1(40790) postgres:vsy/[unknown]LOG: connection authenticated: identity="postgres" method=scram-sha-256 (/etc/postgresql/pg\_hba.conf:103)  2022-12-01 16:27:29.938 GMT [76] 172.28.0.1(40790) postgres:vsy/[unknown]LOG: connection authorized: user=postgres database=vsy application\_name=pgAdmin 4 - DB:vsy SSL enabled (protocol=TLSv1.3, cipher=TLS\_AES\_256\_GCM\_SHA384, bits=256)  2022-12-01 16:27:29.952 GMT [75] 172.28.0.1(40788) postgres:vsy/pgAdmin 4 - DB:vsyLOG: disconnection: session time: 0:00:01.850 user=postgres database=vsy host=172.28.0.1 port=40788  2022-12-01 16:27:29.952 GMT [76] 172.28.0.1(40790) postgres:vsy/pgAdmin 4 - DB:vsyLOG: statement: SET DateStyle=ISO; SET client\_min\_messages=notice; SELECT set\_config('bytea\_output','hex',false) FROM pg\_settings WHERE name = 'bytea\_output'; SET client\_encoding='UNICODE';  2022-12-01 16:27:29.955 GMT [76] 172.28.0.1(40790) postgres:vsy/pgAdmin 4 - DB:vsyLOG: statement: SELECT version()--verbose --column-inserts" postgres |

Рис. 6.1 – Пример логов в «PostgreSQL»

## **6.4 Резервное копирование и восстановление**

Резервное копирование и восстановление — это создание и хранение копий данных, которые можно использовать для восстановления служб организации в случае сбоя первичных данных из-за отключения электричества, атаки программ-вымогателей и прочих нештатных ситуаций. Резервные копии позволяют восстановить состояние систем на более ранний момент, предшествующий потере или повреждению данных, для восстановления служб.

Для осуществления резервного копирования и восстановления на хост был установлена утилита crontab. Утилита использует встроенный планировщик задач в дистрибутивах Linux, crontab же является более удобной конфигурацией в виде таблиц с расписанием запуска скриптов и программ, оформленная в специальном формате, который умеет считывать компьютер. Для каждого пользователя системы создается отдельный crontab-файл со своим расписанием. На рисунке 6.2 предоставлены задачи для осуществления резервной копии в JSON и SQL формат с помощью скриптов импорта и экспорта.

|  |
| --- |
| 0 \* \* \* \* /etc/postgresql/scripts/dump\_json\_db.sh  0 \* \* \* \* /etc/postgresql/scripts/dump\_sql.sh |

Рис. 6.2 – Конфигурационный файл crontab планировщика

## **6.5 Загрузка байтовых файлов**

Байтовый файл - представляют собой последовательность октетов (байт). В «PostgreSQL» предоставляет особый тип данных, как bytea. Тип bytea поддерживает два формата ввода и вывода: «шестнадцатеричный» и традиционный для PostgreSQL формат «спецпоследовательностей». Входные данные принимаются в обоих форматах.

Была реализован «шестнадцатеричный» формат ввода и вывода байтовых данных, так как данные формат совместим со множеством внешних приложений и протоколов, к тому же обычно преобразуется быстрее, поэтому предпочтительнее использовать его.

В «шестнадцатеричном» формате двоичные данные кодируются двумя шестнадцатеричными цифрами на байт, при этом первая цифра соответствует старшим 4 битам. К полученной строке добавляется префикс \x, на рисунке 6.3 предоставлен пример представления строки.

|  |
| --- |
| select \* from categories limit 1;  id | name | image  1 | armchairs | \xffd8ffe000104a46494600010101012c012c0000--more-- |

Рис. 6.3 – Выполнение запроса хранящие байтовое поле

## **7 Инструкция пользователя**

Для данного программного решения было написано клиент-серверное приложение для тестирования и использования базы данных. Фронтенд приложение было написано с помощью веб-фреймворка React. Бэкенд приложение было написано с помощью веб-фреймворка FastAPI. Данные для приложения берутся из данной базы данных и передается последовательно к клиенту через приложения.

Регистрация пользователя осуществляется с помощью советующего окна, который представлен на рисунке 7.1.

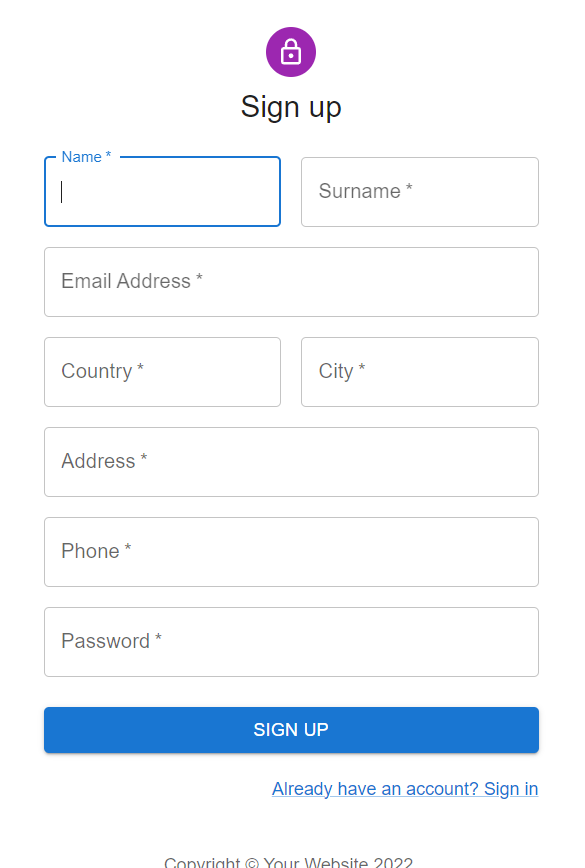


Рисунок 7.1 Окно регистрации

Когда заполнены все соответствующие поля, то пользователь использую кнопку регистрация, после этого вызывается бэкенд приложение, которое проверяет не существует ли пользователь с такой почтой. Если пользователя еще нет, то происходит создание пользователя в базу данных. Далее пользователь может перейти в авторизации.

Для авторизации используется окно, которое представлено на рисунке 7.2. В данном окне располагается два поля: электронная почта и пароль. Далее пользователь может войти в приложение.

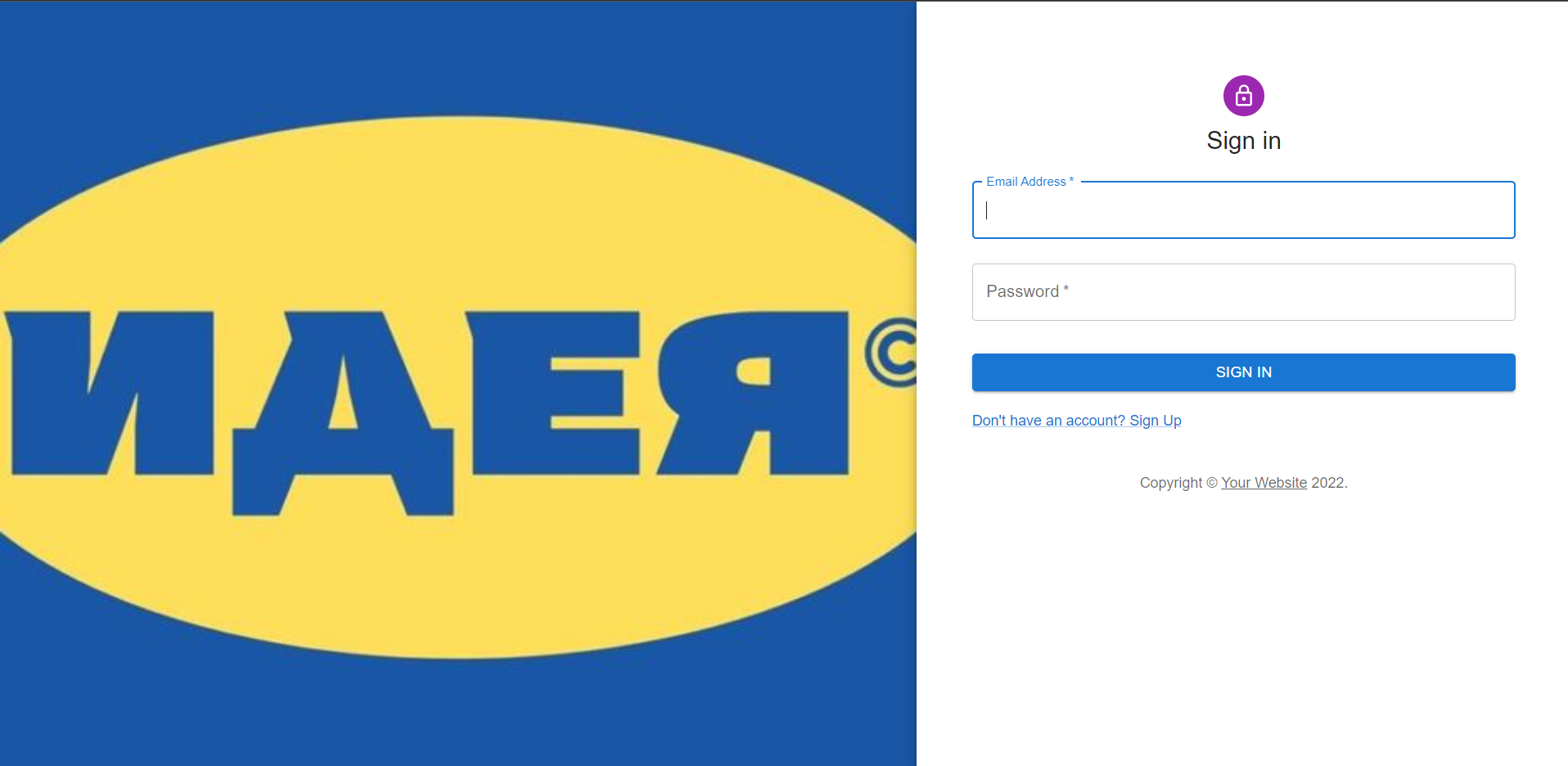


Рисунок 7.2 Окно авторизации

Клиент переходит на главную страницу магазина, где предоставляются категории, которые предоставляет магазин. Окно представлено на рисунке 7.3.

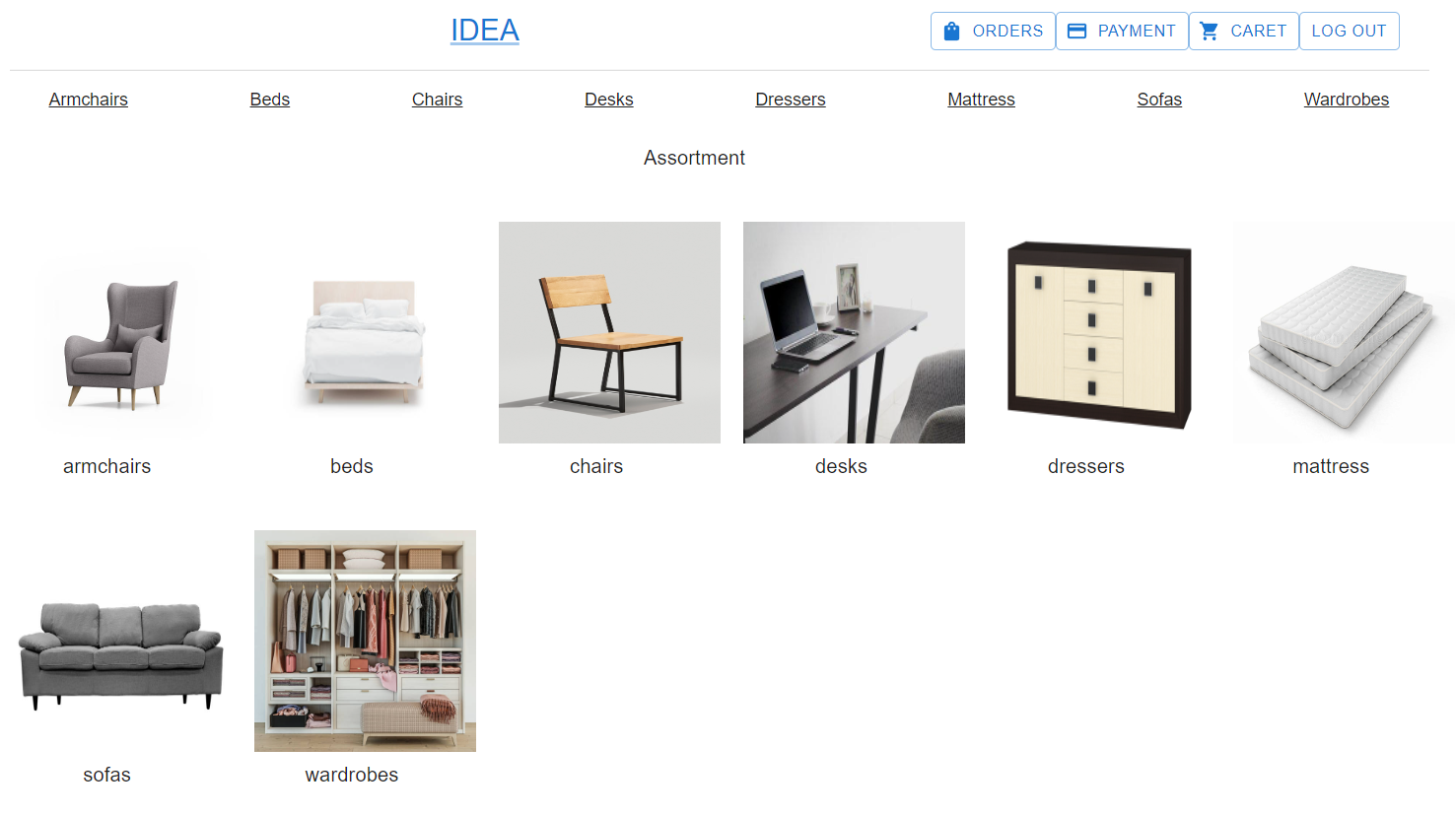


Рисунок 7.3 Главное окно

Пользователь может совершать покупки перейдя по одной из категорий и выбрав товар и его количество. Окно представлено на рисунке 7.4.

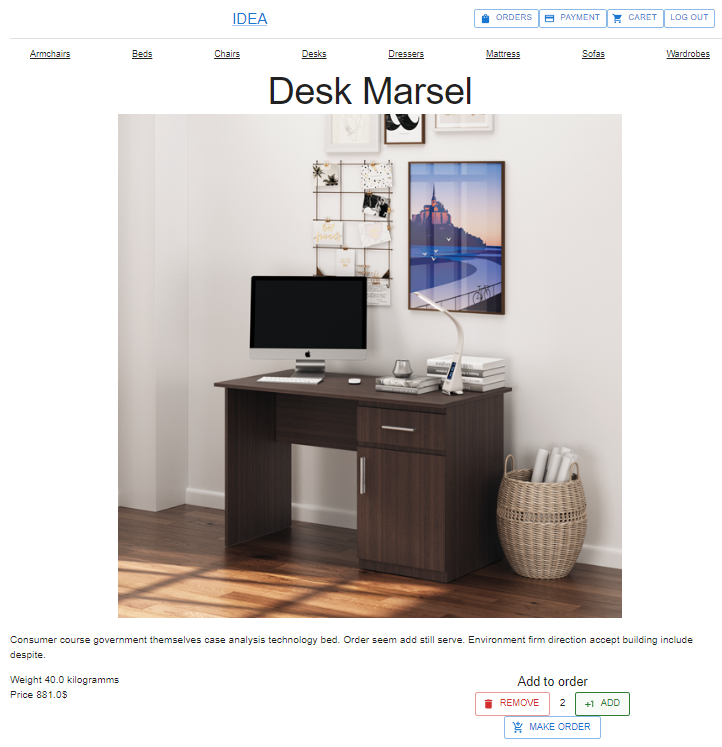


Рисунок 7.4 Окно товара

После нажатия кнопки, товар попадает в корзину клиента. Где клиент уточняет количество товара и подтверждает заказ. Окно представлено на рисунке 7.5.

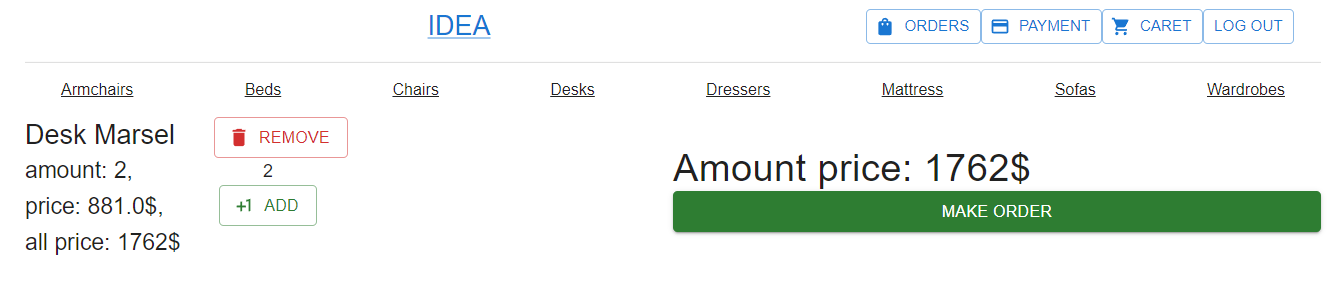


Рисунок 7.5 Окно корзины

После подтверждения заказа, клиент может выбрать товары к оплате и введя банковские данные. Окно представлено на рисунке 7.6.

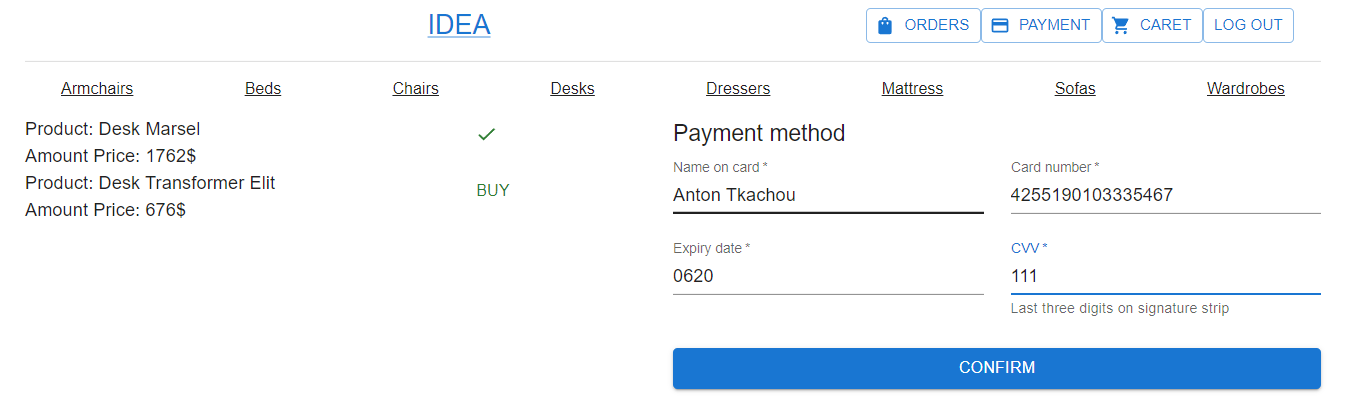


Рисунок 7.6 Окно оплаты

После совершения оплаты, клиент может посмотреть статус своего заказа. Окно представлено на рисунке 7.7.

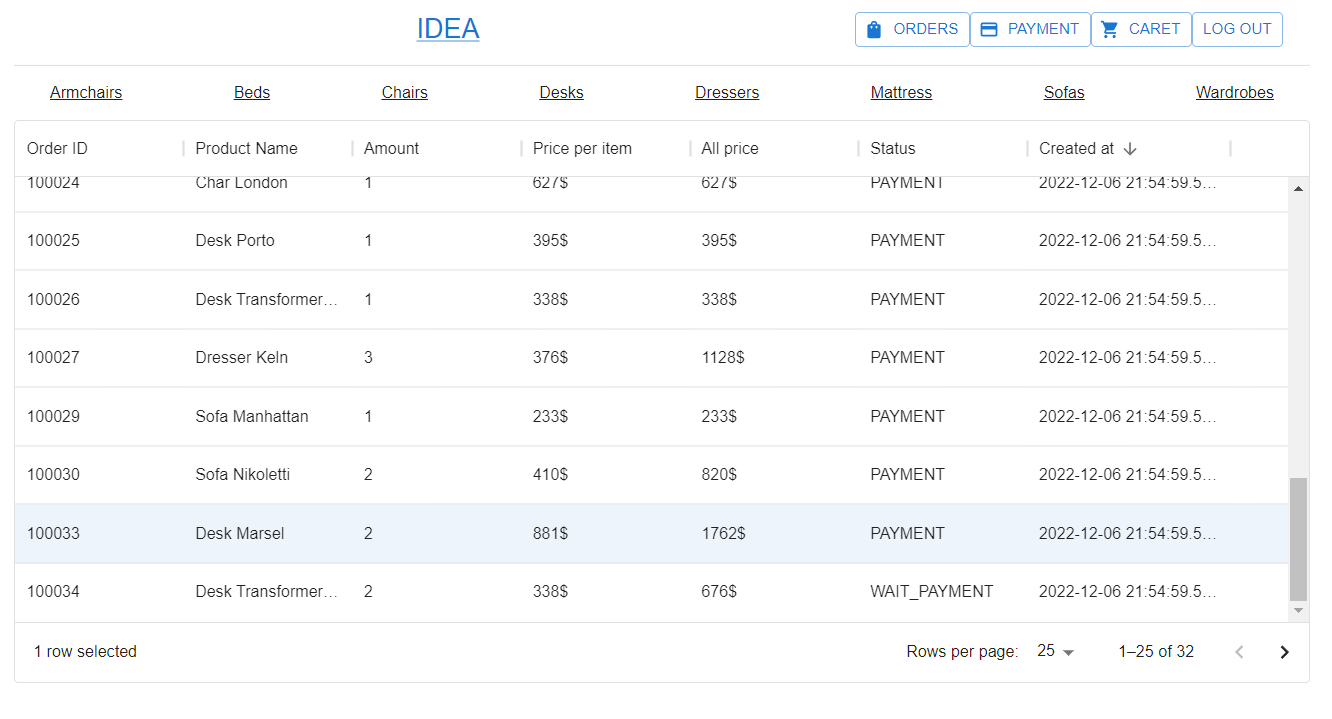


Рисунок 7.7 Окно статусов заказов

Также же есть роль администратора интернет-магазина. Для этого существует отдельный сайт. Админ подтверждает приход оплаты и изменяет статус заказа. Окно представлено на рисунке 7.8.

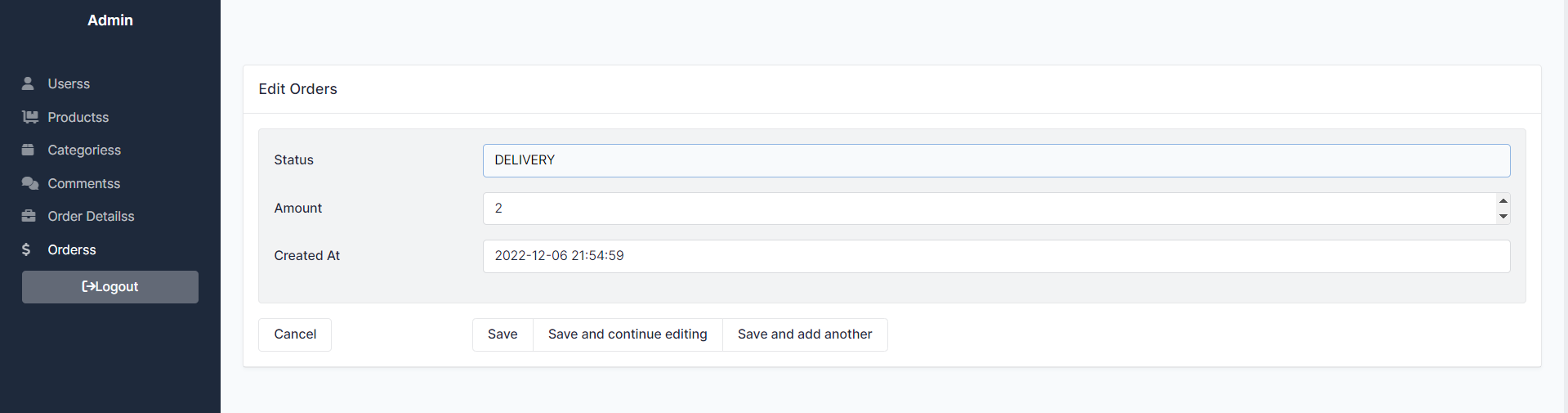


Рисунок 7.8 Окно подтверждения заказа

Заключение

В ходе выполнения данной работы подробно были закреплены знания для проектирования баз данных, с учётом большого количества хранимых параметров, учётом множества связей и использование процедур, функций для осуществления взаимодействий с таблицами. Было проведено тестирование производительности, в котором анализ времени выполнения.

Основой целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением. Приложение было разработано с использованием технологий:

* Python(фреймворк FastAPI)
* React

Разработанная база данных отвечает всем требованиям предметной области, ее таблицы созданной отвечают требованиям нормализации, что позволяет обеспечить целостность и непротиворечивость информации. База данных была разработана с помощью «PostgreSQL 14».

Помимо этого, была изучены и применены при разработке приложения технологии потоковой репликации данных в базе данных, аудит и логирование действий над базой данной, создание распределенной базы данной, вставка мультимедийных данных, резервное копирование данных.

Благодаря проведенной работе, была создана база данных, которая в дальнейшем может быть использована на проект и также её можно легко дорабатывать и расширять её возможности.

При разработке базы данных:

* создано 6 таблиц;
* создано 14 хранимые процедуры;
* создано 8 функций;
* создан индекс, ускоряющий аутентификации клиент;
* создано 3 пользователя базы данных;
* создана распределенная база данных;
* создано резервное копирование;
* подключено аудит и логирование.

В соответствии с полученным результатом работы можно сделать вывод, что разработанная БД работает верно, а требования технического задания выполнены в полном объёме.

Список используемой литературы

1. All things SQL[Электронный ресурс] – Режим доступа: <https://blogs.oracle.com/>. Дата обращения: 10.10.2022.
2. PostgreSQL Tutorials and Solutions[Электронный ресурс] – Режим доступа: https://www.postgresql.r2schools.com/. Дата обращения: 15.10.2022.
3. PostgreSQL documention[Электронный ресурс] – Режим доступа: https://www.postgresql.org/docs/14/index.html. Дата обращения: 30.10.2022.
4. Настройка репликации PostgreSQL в контейнерах Docker[Электронный ресурс] – Режим доступа: https://www.dmosk.ru/miniinstruktions.php?mini=postgresql-replication-docker. Дата обращения: 20.10.2022.

# Приложение А: Скрипт для создания основных таблиц

|  |
| --- |
| CREATE TABLE categories (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  image BYTEA NOT NULL,  PRIMARY KEY (id),  CONSTRAINT categories\_name\_gt\_0 CHECK (length(name)>0)  );  CREATE UNIQUE INDEX ix\_categories\_name ON categories (name);  CREATE TABLE users (  id SERIAL NOT NULL,  email VARCHAR(50) NOT NULL,  password VARCHAR NOT NULL,  name VARCHAR(30) NOT NULL,  surname VARCHAR(30) NOT NULL,  country VARCHAR(100),  city VARCHAR(100),  address VARCHAR(120),  phone VARCHAR(30) NOT NULL,  role INTEGER NOT NULL,  created\_at TIMESTAMP WITHOUT TIME ZONE NOT NULL,  updated\_at TIMESTAMP WITHOUT TIME ZONE,  deleted\_at TIMESTAMP WITHOUT TIME ZONE,  PRIMARY KEY (id),  CONSTRAINT users\_name\_gt\_0 CHECK (length(name)>0),  CONSTRAINT users\_surname\_gt\_0 CHECK (length(surname)>0)  );  CREATE UNIQUE INDEX ix\_users\_email ON users (email);  CREATE UNIQUE INDEX ix\_users\_phone ON users (phone);  CREATE TABLE products (  id SERIAL NOT NULL,  name VARCHAR(50) NOT NULL,  weight FLOAT,  description TEXT,  image BYTEA,  stock INTEGER NOT NULL,  price FLOAT NOT NULL, |

Продолжение листинга создания таблиц.

|  |
| --- |
| category\_id INTEGER,  PRIMARY KEY (id),  CONSTRAINT products\_weight\_gt\_0 CHECK (weight>0),  CONSTRAINT products\_stock\_nature\_or\_0 CHECK (stock>-1),  CONSTRAINT products\_price\_gt\_0 CHECK (price>0),  UNIQUE (name),  FOREIGN KEY(category\_id) REFERENCES categories (id)  );  category\_id INTEGER,  PRIMARY KEY (id),  CONSTRAINT products\_weight\_gt\_0 CHECK (weight>0),  CONSTRAINT products\_stock\_nature\_or\_0 CHECK (stock>-1),  CONSTRAINT products\_price\_gt\_0 CHECK (price>0),  UNIQUE (name),  FOREIGN KEY(category\_id) REFERENCES categories (id)  );  CREATE TABLE comments (  id SERIAL NOT NULL,  text TEXT NOT NULL,  user\_id INTEGER,  product\_id INTEGER,  PRIMARY KEY (id),  FOREIGN KEY(user\_id) REFERENCES users (id),  FOREIGN KEY(product\_id) REFERENCES products (id)  );  CREATE TABLE orderdetails (  id SERIAL NOT NULL,  order\_id INTEGER,  product\_id INTEGER,  PRIMARY KEY (id),  FOREIGN KEY(order\_id) REFERENCES orders (id),  FOREIGN KEY(product\_id) REFERENCES products (id)  ); |

Окончание листинга для создания таблиц.

# Приложение Б: Конфигурация docker-compose

|  |
| --- |
| version: "3.8"  services:  db-primary:  build:  context: .  dockerfile: ./primary/primary.Dockerfile  container\_name: "db-primary"  env\_file: "./primary/primary.env"  ports:  - 5432:5432  # - 192.168.0.12:15432:5432  networks:  db-network:  ipv4\_address: 172.28.0.2  volumes:  - ./primary/primary-db-data:/var/lib/postgresql/data  - ./primary/primary.pg\_hba.conf:/etc/postgresql/pg\_hba.conf  ./primary/primary.postgresql.conf:/etc/postgresql/postgresql.conf  - ./primary/scripts/init.sh:/docker-entrypoint-initdb.d/init.sh  - ./primary/scripts/:/etc/postgresql/scripts/  - ./primary/.pgpass:/var/lib/postgresql/.pgpass  - ./primary/logs/:/var/log/postgresql/  command: postgres -c config\_file=/etc/postgresql/postgresql.conf -c hba\_file=/etc/postgresql/pg\_hba.conf  healthcheck:  test: ["CMD-SHELL", "pg\_isready -U postgres -d postgres"]  interval: 10s  timeout: 5s  retries: 5  start\_period: 10s  restart: always  db-standby:  build:  context: .  dockerfile: ./standby/standby.Dockerfile  container\_name: "db-standby"  ports:  - 15433:5432  # - 192.168.0.12:15433:5432  networks: |

Продолжение листинга конфигурации docker-compose.

|  |
| --- |
| db-network:  ipv4\_address: 172.28.0.3  env\_file: "./standby/standby.env"  volumes:  - ./standby/standby-db-data:/var/lib/postgresql/data  - ./standby/standby-db-repl:/var/lib/postgresql/repl  - ./standby/.pgpass:/var/lib/postgresql/.pgpass  - ./standby/standby.pg\_hba.conf:/etc/postgresql/pg\_hba.conf  ./standby/standby.postgresql.conf:/etc/postgresql/postgresql.conf  - ./standby/scripts/init.sh:/docker-entrypoint-initdb.d/init.sh  - ./standby/scripts/:/etc/postgresql/scripts/  command: postgres -c config\_file=/etc/postgresql/postgresql.conf -c hba\_file=/etc/postgresql/pg\_hba.conf  healthcheck:  test: ["CMD-SHELL", "pg\_isready -U postgres -d postgres"]  interval: 10s  timeout: 5s  retries: 5  start\_period: 10s  restart: always  depends\_on:  - db-primary  networks:  db-network:  driver: bridge  name: db-network  ipam:  config:  - subnet: 172.28.0.0/16  gateway: 172.28.0.1 |

Окончание листинга конфигурации docker-compose.

# Приложение В: Скрипты для инициализаций БД

Скрипт для «Primary database»

|  |
| --- |
| #!/bin/bash  set -e  chmod 0600 /var/lib/postgresql/.pgpass  export PGPASSFILE='/var/lib/postgresql/.pgpass'  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "$POSTGRES\_DB" <<-EOSQL  CREATE DATABASE $EXAMPLE\_DB;  CREATE USER repluser WITH REPLICATION ENCRYPTED PASSWORD '$REPL\_USER\_PASS' LOGIN;  CREATE USER staff WITH NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN REPLICATION ENCRYPTED PASSWORD 'staff';  GRANT CONNECT, TEMPORARY ON DATABASE vsy TO staff;  GRANT pg\_read\_all\_data TO staff;  SELECT pg\_reload\_conf();  EOSQL |

Скрипт для «Standby database»

|  |
| --- |
| #!/bin/bash  set -e  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "$POSTGRES\_DB" <<-EOSQL  SELECT pg\_reload\_conf();  EOSQL |

# Приложение Г: Конфигурация базы данных «Postgres 14»

Файл конфигурации primary.postgresql.conf

|  |
| --- |
| # - Connection Settings -  listen\_addresses = '\*'  port = 5432 # (change requires restart)  max\_connections = 100 # (change requires restart)  # - SSL –  ssl = on  ssl\_cert\_file = '/var/lib/postgresql/server.crt'  ssl\_key\_file = '/var/lib/postgresql/server.key'  # - Memory –  shared\_buffers = 128MB # min 128kB  work\_mem = 4MB # min 64kB  maintenance\_work\_mem = 100MB # min 1MB  # - Asynchronous Behavior –  effective\_io\_concurrency = 200 # 1-1000; 0 disables prefetching  max\_worker\_processes = 8 # (change requires restart)  max\_parallel\_workers\_per\_gather =4 # taken from max\_parallel\_workers  # - Settings –  wal\_level = replica # minimal, replica, or logical  wal\_buffers = 16MB # min 32kB, -1 sets based on shared\_buffers  checkpoint\_completion\_target = 0.7 # checkpoint target duration, 0.0 - 1.0  max\_wal\_size = 4GB  min\_wal\_size = 1GB  # - Sending Servers –  max\_wal\_senders = 10 # max number of walsender processes  max\_replication\_slots = 10 # max number of replication slots  wal\_keep\_size = 126 # in megabytes; 0 disables  track\_commit\_timestamp =on # collect timestamp of transaction commit  # - Primary Server -  hot\_standby = on # "off" disallows queries during recovery  hot\_standby\_feedback = on # send info from standby to prevent  # - Planner Cost Constants -  random\_page\_cost = 1.1 # same scale as above  effective\_cache\_size = 4GB  # - Other Planner Options –  default\_statistics\_target = 100 # range 1-10000  # - Where to Log –  logging\_collector = on # Enable capturing of stderr and csvlog  log\_directory = '/var/log/postgresql' # directory where log files are written  log\_file\_mode = 0777 # creation mode for log files  log\_connections = on  log\_disconnections = on  log\_hostname = on  log\_line\_prefix = '%m [%p] %r %u:%d/%a'  log\_statement = 'all' # none, ddl, mod, all  # - Statement Behavior –  row\_security = off |

Файл конфигурации primary.pg\_hba.conf

|  |
| --- |
| # TYPE DATABASE USER ADDRESS METHOD  # "local" is for Unix domain socket connections only  local all all trust  # IPv4 local connections:  host all all 127.0.0.1/32 trust  # IPv6 local connections:  host all all ::1/128 trust  # Allow replication connections from localhost, by a user with the  # replication privilege.  local replication all trust  host replication all 127.0.0.1/32 trust  host replication all ::1/128 trust  host replication repluser 172.28.0.0/16 md5  host replication repluser 198.168.0.0/24 md5 # LAN CHANGED!  host replication repluser 0.0.0.0/0 md5  host all all all scram-sha-256 |

# Приложение Д: Скрипт для импорта данных в JSON

|  |
| --- |
| #!/bin/bash  set -e  myArray=( "users" "categories" "products" "orders" "orderdetails" "comments" )  for str in ${myArray[@]}; do  echo $str  jq -c '.[]' /etc/postgresql/scripts/json\_data/$str.json > /etc/postgresql/scripts/json\_data/${str}\_valid.json  # psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "$EXAMPLE\_DB" <<-EOSQL  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  DROP TABLE IF EXISTS temp;  CREATE TABLE temp (data jsonb);  \COPY temp (data) FROM 'json\_data/${str}\_valid.json';  EOSQL  if [ "$str" == "users" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  INSERT INTO users(email, password, name, surname, country, city, address, phone, role, created\_at, updated\_at)  SELECT data->>'email', data->>'password', data->>'name', data->>'surname', data->>'country', data->>'city', data->>'address',  data->>'phone', (data->>'role')::integer, (data->>'created\_at')::timestamp, (data->>'updated\_at')::timestamp FROM temp;  EOSQL  elif [ "$str" == "categories" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  INSERT INTO categories (name, image) SELECT data->>'name', (decode(data->>'image', 'hex'))::bytea FROM temp;  EOSQL  elif [ "$str" == "products" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL |

Продолжение листинга скрипта импорта.

|  |
| --- |
| INSERT INTO products (name, weight, description, image, stock, price, category\_id)  SELECT data->>'name', (data->>'weight')::double precision, data->>'description', (decode(data->>'image', 'hex'))::bytea, (data->>'stock')::integer,  (data->>'price')::double precision, (data->>'category\_id')::integer FROM temp;  EOSQL  elif [ "$str" == "comments" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  INSERT INTO comments (text, user\_id, product\_id)  SELECT data->>'text', (data->>'user\_id')::integer, (data->>'product\_id')::integer  FROM temp;  EOSQL  elif [ "$str" == "orders" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  INSERT INTO orders (status, amount, created\_at, user\_id)  SELECT data->>'status', (data->>'amount')::integer, (data->>'created\_at')::timestamp, (data->>'user\_id')::integer  FROM temp;  EOSQL  elif [ "$str" == "orderdetails" ]; then  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  INSERT INTO orderdetails (order\_id, product\_id)  SELECT (data->>'order\_id')::integer, (data->>'product\_id')::integer  FROM temp;  EOSQL  fi  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "vsy" <<-EOSQL  DROP TABLE temp;  EOSQL  done |

Окончание листинга скрипта импорта.

# Приложение Е: Скрипт для экспорта данных в JSON

|  |
| --- |
| #!/bin/bash  set -e  dump\_name=$(date +%Y\_%m\_%d\_%H\_%M)  myArray=("categories" "comments" "orderdetails" "orders" "products" "users")  mkdir -p /etc/postgresql/scripts/dumps/  for str in ${myArray[@]}; do  echo $str  psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" --dbname "$EXAMPLE\_DB" <<-EOSQL  \t  \a  \o /etc/postgresql/scripts/dumps/${str}\_${dump\_name}.json  SELECT array\_to\_json(array\_agg(row\_to\_json(t))) FROM $str t;  EOSQL  done |

# Приложение Ж: Скрипт для экспорта данных в SQL

|  |
| --- |
| #!/bin/bash  set -e  dump\_name=$(date +%Y\_%m\_%d\_%H\_%M)  mkdir -p /etc/postgresql/scripts/dumps/  touch /etc/postgresql/scripts/dumps/$dump\_name.sql  chown postgres:postgres /etc/postgresql/scripts/dumps/$dump\_name.sql  chown postgres:postgres /etc/postgresql/scripts/dumps/  chmod 0777 /etc/postgresql/scripts/dumps/  su -c "pg\_dump --host=172.28.0.2 --username=postgres --dbname=$EXAMPLE\_DB -w --file=/etc/postgresql/scripts/dumps/$dump\_name.sql --format=p --if-exists --create --clean --verbose --column-inserts" postgres |

# Приложение З: Запросы сравнения оптимизации

|  |
| --- |
| -- 1  EXPLAIN ANALYZE VERBOSE SELECT \* FROM categories WHERE name = 'dressers';  -- 2  EXPLAIN ANALYZE VERBOSE SELECT \* FROM users WHERE email = 'nfranco@example.com';  -- 3  EXPLAIN ANALYZE VERBOSE SELECT \* FROM users WHERE phone = '+1-455-858-3617';  -- 4  BEGIN;  EXPLAIN ANALYZE VERBOSE INSERT INTO users( email, password,  name, surname, country, city, address, phone, role, created\_at ) VALUES (  'mytest@gmaill.com', '1111', 'Anton', 'Tkachou', 'Belarus', 'Minsk',  'Belaruskaia 21', '+3132134131', 1, current\_timestamp );  ROLLBACK;  -- 5  BEGIN;  EXPLAIN ANALYZE VERBOSE INSERT INTO categories(name, image)  VALUES ('door', 'dasdadasd');  ROLLBACK; |