Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ
по лабораторной работе № 3
на тему
«Разработка NoSQL базы данных и спецификаций прикладной программы»

Студент: М.А. Ходосевич

Преподаватель: А.И. Крюков

МИНСК 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Темой данной лабораторной работы являются освоение прикладного интерфейса СУБД BerkeleyDB, разработка конвертора базы данных PostgreSQL в набор баз данных Berkeley DB и адаптация спецификаций приложения.

В ходе работы нужно научиться преобразовывать реляционные базы данных (PostgreSQL) в формат ключ-значение (Berkeley DB). А также освоить процесс сериализации и десериализации данных для хранения в нереляционной базе данных и выполнить адаптацию существующих спецификаций приложения для работы с Berkeley DB.

Так как с установкой Berkeley DB возникли определенные трудности, в частности проблемы с компиляцией файлов и назначения переменных среды, было решено использовать другую базу данных. Вместо Berkeley DB будет использована MongoDB, которая также является нереляционной и содержит значения вида «ключ-значение».

1 СОЗДАНИЕ UML-ДИАГРАММЫ

Исходное задание: Создать концептуальную модель организации «Прокат видеодисков» и представить сущности и связи в виде UML-диаграммы.

Концептуальная UML-диаграмма представлена на рисунке 1.

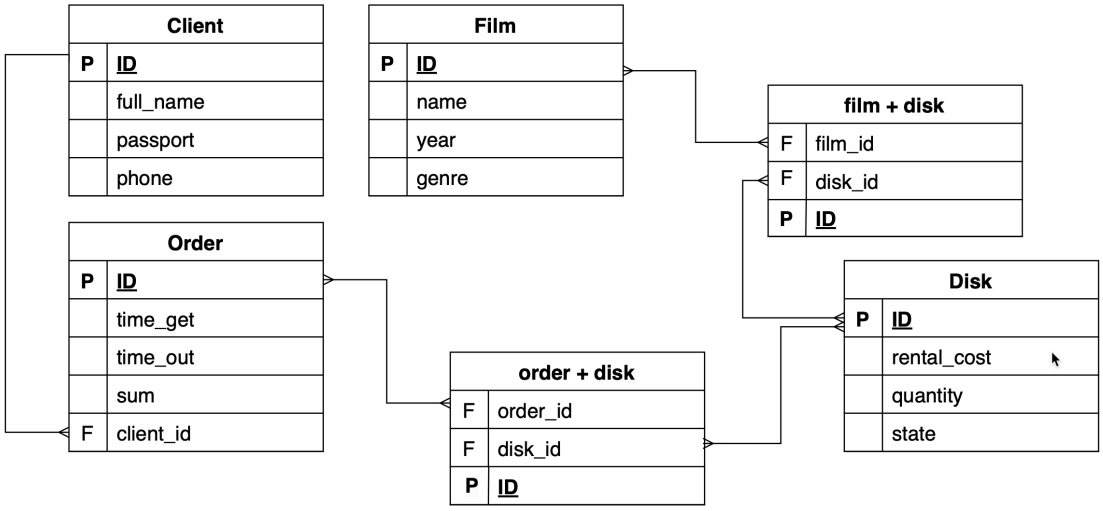

Рисунок 1.1 – UML-диаграмма

1.1 Предметная область

Предметная область – «Прокат видеодисков». Модели по типу «клиент-продавец». Предоставляемая услуга – прокат видеодисков.

1.2 Типы объектов

Для модели «Прокат видеодисков» было выделено 4 типов объектов

1) «Заказ» – заказ, который совершает клиент.

2) «Диск» – содержит информацию о конкретных дисках (копиях фильмов), находящихся в прокате.

3) «Фильм» – описывает информацию о фильмах, доступных для проката.

4) «Клиент» – человек, арендующий диск.

1.3 Атрибуты объектов

Сущность «Film» содержит атрибуты: name, year, genre;
Сущность «Disk» содержит атрибуты: rental_cost, quantity, state
Сущность «Director» содержит атрибуты: full_name, email

Сущность «Client» содержит атрибуты: full_name, passport, phone

1.4 Типы связей

Для модели «Прокат видеодисков» можно выделить следующие связи:

1 – «film-disk», описывает фильм, который записан на диск. Мощность связи «многие-ко-многим».

2 – «disk-order», описывает, какие диски взяты в аренду. Мощность связи «многие-ко-многим».

3 – «client-order», описывает заказ, который сделал клиент. Мощность связи «один-ко-многим».

2 ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Техтребования содержат принципы построения взаимодействия клиент-серверного приложения в рамках работы с базой данных, но оторвано от конкретной реализации будь то Postgres или BearkleyDB.

Техтребования подразделяются на требования для серверного приложения и требования для интерфейса клиентского приложения.

2.1 Серверное приложение

1) Серверное приложение для реализации соединения с базой данный MongoDB будет написано на языке NodeJS.

2) Должны быть предусмотрены CRUD операции для всех таблиц из UML-диаграммы.

3) Серверное приложением должно представлять из себя REST API сервер.

4) Серверные операции должны быть описаны обще, для дальнейшего масштабирования и наследования.

5) В серверном приложении должны быть описаны все используемые сущности базы данных.

6) Приложение должно быть оптимизированным.

2.2 Клиентское приложение. Интерфейс

Клиентское приложение должно иметь дополнительный функционал: приложение должно предоставлять интерфейс для взаимодействия как с PostgreSQL, так и с Redis. Также должна быть возможность произвести преобразование данных из PostgreSQL в Redis.

1) Клиентское приложение должно быть написано в SPA, для обеспечения быстродействия и реактивности. Использовать один из популярных фреймворков.

2) Интерфейс приложения должен отвечать принципам UI/UX. Дизайн должен быть удобен, понятен и однозначен.

3) Взаимодействие с серверным приложением должно происходить через REST API.

4) Приложение должно иметь минималистичный дизайн.

5) Приложение должно быть оптимизированным.

3 РАЗРАБОТКА КОНВЕРТОРА БАЗЫ ДАННЫХ

Приложение должно иметь механизм преобразования данных из Postgre в наборы базы данных Mongo. Данный функционал реализован следующим образом:

1) Сервер имеет дополнительный API, предоставляющий метод для получения данных из PostgreSQL и занесения их в Mongo.

2) Сервер проверяет, есть ли запись из Postgre в Mongo:

а) данные есть и они актуальны – они пропускаются;

б) данные есть и они не актуальны, или данных вовсе нет – они заносятся в Mongo.

Пример SQL-запроса для получения данных приведен ниже.

```
async function transferTableToCollection(pgTable, mongoCollection) {
  try {
    const pgClient = await pgPool.connect();
    const result = await pgClient.query(`SELECT * FROM ${pgTable}`);
    const rows = result.rows;
    pgClient.release();

    const collection = mongoDb.collection(mongoCollection);
    await collection.deleteMany({});
    await collection.insertMany(rows);

       console.log(`Таблица ${pgTable} успешно перенесена в коллекцию
${mongoCollection}`);
  } catch (err) {
    console.error(`Ошибка при переносе таблицы ${pgTable}:`, err);
  }
}
```

Полный код для конвертора базы данных с PastgreSQL на MongoDB на языке Node.js приведен ниже.

```
require('dotenv').config();
const { Pool } = require('pg');
const { MongoClient } = require('mongodb');

const pgPool = new Pool({
  user: process.env.PG_USER,
  host: process.env.PG_HOST,
  database: process.env.PG_DATABASE,
  password: process.env.PG_PASSWORD,
  port: process.env.PG_PORT,
});

const mongoClient = new MongoClient(process.env.MONGO_URI);
let mongoDb;

async function transferTableToCollection(pgTable, mongoCollection) {
  try {
    const pgClient = await pgPool.connect();
    const result = await pgClient.query(`SELECT * FROM ${pgTable}`);
    const rows = result.rows;
    pgClient.release();
```

```
    const collection = mongoDb.collection(mongoCollection);
    await collection.deleteMany({});
    await collection.insertMany(rows);

       console.log(`Таблица ${pgTable} успешно перенесена в коллекцию
${mongoCollection}`);
  } catch (err) {
    console.error(`Ошибка при переносе таблицы ${pgTable}:`, err);
  }
}

async function convertPostgresToMongo() {
  try {
    await mongoClient.connect();
    mongoDb = mongoClient.db('cafebar');

    await transferTableToCollection('film', 'films');
    await transferTableToCollection('client', 'clients');
    await transferTableToCollection('"order"', 'orders');
    await transferTableToCollection('disk', 'disks');
  } finally {
    await pgPool.end();
    await mongoClient.close();
    console.log('Конвертация завершена и соединения закрыты.');
  }
}

convertPostgresToMongo().catch(console.error);
```

Далее будут приведены преимущества и недостатки обеих баз данных.

Преимущества Postgre:

- PostgreSQL поддерживает SQL, транзакции и ACID-совместимость, что делает её удобной для сложных запросов и надёжных операций с данными.

- PostgreSQL поддерживает пользовательские функции, хранимые процедуры, индексы и типы данных, что позволяет адаптировать её под разнообразные задачи.

- PostgreSQL может справляться с большими нагрузками при правильной настройке и оптимизации.

Недостатки Postgre:

- По сравнению с некоторыми другими базами данных PostgreSQL требует больше усилий по настройке.

- PostgreSQL потребляет больше памяти и процессорных ресурсов, особенно на высоких нагрузках.

- PostgreSQL имеет ограниченную поддержку NoSQL-функционала, хотя и поддерживает JSON и hstore.

Преимущества MongoDB:

- MongoDB часто используется как встроенное хранилище для приложений, так как не требует отдельного сервера и хорошо интегрируется с языками высокого уровня.

- Поддерживает высокую производительность на простых операциях с ключами и значениями, особенно при минимальном объёме данных.

- Поддержка транзакций и ACID, что обеспечивает надёжность операций.

Недостатки MongoDB:

- MongoDB не поддерживает SQL, что усложняет её использование в проектах, требующих сложных реляционных запросов.
- Меньше возможностей для настройки индексов, типов данных и расширений, чем у PostgreSQL.

ЗАКЛЮЧЕНИЕ

В результате работы была разработана UML-диаграмма для базы данных. После чего были написаны технические требования для серверной и клиентской частей будущего приложения. После чего был разработан конвертор на языке Node.js для конвертации базы данных PostgreSQL в нереляционную базу данных MongoBD.