DATABASE TESTING

SECTION - 4

MID-TERM GROUP PROJECT

Professor : Kiran Talwar

**1) Roles and responsibilities of Team Member**

|  |  |  |  |
| --- | --- | --- | --- |
| Group Number: 04 | | | |
| **Team Members** | | **Student Ids** | **Roles/Responsibility** |
|  | Shivani Shaileshkumar Varu | 8941914 | Typescript Interface Development, Containerize Project with docker |
|  |  |  | Database Schema Design |
|  |  |  | Sample Data Creation  (DDL, DML, CRUD) |
|  |  |  | SQL Queries Implementation |
|  |  |  | Readme File Creation |

**2) Clearly title and identify the tables required and what type each of the attribute is**

**Database Design:**

|  |  |  |
| --- | --- | --- |
| **Table Name** | | |
| Attribute | Type | Description |
|  |  |  |
|  |  |  |

**DDL Command**

**CREATE TABLE:**

**Books Table**

CREATE TABLE Books (

book\_id SERIAL PRIMARY KEY,

title VARCHAR(200) NOT NULL,

genre VARCHAR(50) NOT NULL,

author\_id INT NOT NULL,

publisher\_id INT NOT NULL,

publication\_date DATE NOT NULL,

price DECIMAL(10, 2) NOT NULL,

rating DECIMAL(3, 2),

format VARCHAR(20) NOT NULL CHECK (format IN ('physical', 'ebook', 'audiobook')),

FOREIGN KEY (author\_id) REFERENCES Authors(author\_id),

FOREIGN KEY (publisher\_id) REFERENCES Publishers(publisher\_id)

);

**DML Command**

**INSERTING DATA INTO TABLE:**

**Books Table**

INSERT INTO Books (title, genre, author\_id, publisher\_id, publication\_date, price, rating, format) VALUES

('The Namesake', 'Literary Fiction', 1, 1, '2023-09-01', 14.99, 4.2, 'physical'),

('Unaccustomed Earth', 'Literary Fiction', 1, 1, '2022-05-01', 15.99, 4.3, 'ebook'),

('Interpreter of Maladies', 'Literary Fiction', 1, 1, '2021-03-01', 13.99, 4.4, 'audiobook'),

('The Lowland', 'Literary Fiction', 1, 1, '2020-07-01', 16.99, 4.5, 'physical'),

('The God of Small Things', 'Thriller', 2, 2, '2023-06-01', 12.99, 4.5, 'ebook'),

('The Ministry of Utmost Happiness', 'Thriller', 2, 2, '2022-11-01', 14.99, 4.2, 'physical'),

('Capitalism: A Ghost Story', 'Comedy', 2, 2, '2021-08-01', 12.99, 4.0, 'ebook'),

('Field Notes on Democracy', 'Comedy', 2, 2, '2020-02-01', 13.99, 4.3, 'audiobook'),

('The Immortals of Meluha', 'Crime', 7, 7, '2023-02-01', 14.99, 4.5, 'physical'),

('The Secret of the Nagas', 'Crime', 7, 7, '2022-06-01', 15.99, 4.6, 'ebook'),

('The Oath of the Vayuputras', 'Crime', 7, 7, '2021-12-01', 16.99, 4.4, 'audiobook'),

('Scion of Ikshvaku', 'Crime', 7, 7, '2020-05-01', 13.99, 4.7, 'physical'),-- Chitra Banerjee Divakaruni (Mythology)

('The Palace of Illusions', 'Mythology', 5, 5, '2023-02-01', 15.99, 4.4, 'ebook'),

('The Forest of Enchantments', 'Mythology', 5, 5, '2022-03-01', 14.99, 4.3, 'physical'),

('Before We Visit the Goddess', 'Mythology', 5, 5, '2021-07-01', 13.99, 4.2, 'audiobook'),

('The Last Queen', 'Mythology', 5, 5, '2020-10-01', 16.99, 4.1, 'physical');

**SQL Queries for CRUD Operations**

**Create a New Book**

INSERT INTO Books (title, genre, author\_id, publisher\_id, publication\_date, price, rating, format)

VALUES ('The Last Archive', 'Historical Fiction', 4, 2, '2023-09-15', 21.99, 4.7, 'ebook');

**Read a Book by ID**

SELECT \* FROM Books WHERE book\_id = 1;

**Update a Book**

UPDATE Books SET title = 'Updated Title', genre = 'Updated Genre', price = 15.99 WHERE book\_id = 1;

**Delete a Book**

DELETE FROM Books WHERE book\_id = 1;

**SQL QUERIES:**

**Power writers (authors) with more than X books in the same genre published within the last X years**

SELECT

b.author\_id,

a.name,

b.genre,

COUNT(\*) AS book\_count

FROM

Books b

JOIN

Authors a ON b.author\_id = a.author\_id

WHERE

b.publication\_date >= CURRENT\_DATE - INTERVAL '5 years'

GROUP BY

b.author\_id, a.name, b.genre

HAVING

COUNT(\*) > 3;

**Loyal Customers who have spent more than X dollars in the last year**

SELECT c.name, o.order\_date, SUM(o.total\_amount) AS total\_spent

FROM Customers c

JOIN Orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_date >= DATE\_TRUNC('year', NOW()) - INTERVAL '5 year'

GROUP BY c.customer\_id, c.name, o.order\_date

HAVING SUM(o.total\_amount) > 58;

**Well Reviewed Books that have a better user rating than average**

SELECT

b.title,

b.rating,

r.comment AS review

FROM

Books b

JOIN

Reviews r ON b.book\_id = r.book\_id

WHERE

b.rating > (SELECT AVG(rating) FROM Books);

**The most popular genre by sales:**

SELECT

b.genre,

SUM(oi.quantity) AS total\_sold

FROM

Books b

JOIN

Order\_Items oi ON b.book\_id = oi.book\_id

GROUP BY

b.genre

ORDER BY

total\_sold DESC

LIMIT 1;

**The 10 most recent posted reviews by Customers:**

SELECT

r.review\_id,

r.comment,

r.review\_date,

c.name AS customer\_name,

b.title AS book\_title

FROM

Reviews r

JOIN

Customers c ON r.customer\_id = c.customer\_id

JOIN

Books b ON r.book\_id = b.book\_id

ORDER BY

r.review\_date DESC

LIMIT 10;

**Typescript Interface**

**BookService.ts**

* **Purpose**: This file contains the BookService class which encapsulates all database interactions related to the Books table. It provides methods to create, read, update, and delete books (CRUD operations).
* **Details**: This class abstracts the database layer, making it easier to manage database queries and maintain the code. It interacts with the PostgreSQL database using the pg library.  
    
  import { Pool } from 'pg';

interface Book {

book\_id?: number;

title: string;

author\_id: number;

publisher\_id: number;

genre: string;

format: string;

price: number;

publication\_date: string;

rating: number;

}

class BookService {

private pool: Pool;

constructor() {

this.pool = new Pool({

user: process.env.DB\_USER || 'postgres',

host: process.env.DB\_HOST || 'localhost',

database: process.env.DB\_NAME || 'online\_bookstore\_midterm\_dbt',

password: process.env.DB\_PASSWORD || 'password',

port: Number(process.env.DB\_PORT) || 5432,

});

}

async createBook(book: Book): Promise<Book> {

const result = await this.pool.query(

`INSERT INTO Books (title, author\_id, publisher\_id, genre, format, price, publication\_date, rating)

VALUES ($1, $2, $3, $4, $5, $6, $7, $8) RETURNING \*`,

[

book.title,

book.author\_id,

book.publisher\_id,

book.genre,

book.format,

book.price,

book.publication\_date,

book.rating,

]

);

return result.rows[0];

}

async getBookById(book\_id: number): Promise<Book | null> {

const result = await this.pool.query(

`SELECT \* FROM Books WHERE book\_id = $1`,

[book\_id]

);

return result.rows[0] || null;

}

async updateBook(book: Book): Promise<Book> {

const result = await this.pool.query(

`UPDATE Books SET title = $1, author\_id = $2, publisher\_id = $3, genre = $4, format = $5, price = $6, publication\_date = $7, rating = $8

WHERE book\_id = $9 RETURNING \*`,

[

book.title,

book.author\_id,

book.publisher\_id,

book.genre,

book.format,

book.price,

book.publication\_date,

book.rating,

book.book\_id,

]

);

return result.rows[0];

}

async deleteBook(book\_id: number): Promise<void> {

await this.pool.query(

`DELETE FROM Books WHERE book\_id = $1`,

[book\_id]

);

}

}

export default BookService;

**index. ts:**

* **Purpose**: This file sets up and runs the Express server. It defines the API endpoints that the frontend or other clients will interact with.
* **Details**: The server listens on a specified port and routes requests to the appropriate methods in the BookService class. It handles incoming HTTP requests, calls the corresponding service methods, and sends back HTTP responses.

import express from 'express';

import BookService from './BookService';

const app = express();

const port = 3000; // Ensure the port is set to 3000

const bookService = new BookService();

app.use(express.json());

app.get("/", (req, res) => {

res.send("Welcome to the Online Bookstore API");

});

app.post("/books", async (req, res) => {

try {

const newBook = await bookService.createBook(req.body);

res.status(201).json(newBook);

} catch (error: any) {

res.status(500).json({ error: error.message });

}

});

app.get("/books/:id", async (req, res) => {

try {

const book = await bookService.getBookById(parseInt(req.params.id, 10));

if (book) {

res.json(book);

} else {

res.status(404).json({ message: "Book not found" });

}

} catch (error: any) {

res.status(500).json({ error: error.message });

}

});

app.put("/books/:id", async (req, res) => {

try {

const book = { ...req.body, book\_id: parseInt(req.params.id, 10) };

const updatedBook = await bookService.updateBook(book);

res.json(updatedBook);

} catch (error: any) {

res.status(500).json({ error: error.message });

}

});

app.delete("/books/:id", async (req, res) => {

try {

await bookService.deleteBook(parseInt(req.params.id, 10));

res.status(204).send();

} catch (error: any) {

res.status(500).json({ error: error.message });

}

});

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

**Test.ts file :**

* **Purpose**: This file is used for running test scripts to perform CRUD operations on the Books table. It helps in verifying that the methods in the BookService class work as expected.
* **Details**: The test script initializes an instance of BookService and performs a sequence of operations such as creating a new book, reading the book by ID, updating the book, and deleting the book. It logs the results of each operation to the console.

import BookService from "./BookService";

// Initialize BookService

const bookService = new BookService();

async function testCRUDOperations() {

// Create a new book

const newBook = {

title: 'New Book',

author\_id: 1,

publisher\_id: 1,

genre: 'Fiction',

format: 'physical' as 'physical' | 'ebook' | 'audiobook',

price: 19.99,

publication\_date: '2023-06-01',

rating: 4.5

};

const createdBook = await bookService.createBook(newBook);

console.log('Created Book:', createdBook);

// Read the book by ID

const bookId = createdBook.book\_id;

if (bookId !== undefined) {

const readBook = await bookService.getBookById(bookId);

console.log('Read Book:', readBook);

if (readBook) {

// Update the book

readBook.price = 21.99;

const updatedBook = await bookService.updateBook(readBook);

console.log('Updated Book:', updatedBook);

// Delete the book

await bookService.deleteBook(bookId);

console.log('Deleted Book:', bookId);

}

}

}

testCRUDOperations().catch(console.error);