

# Library Borrowing Tracker

Final Report dedicated for Database Group Project



Group 7

## Group Members:

Farrel Tsaqif Anindyo (24/536735/PA/22769)  
Faiq Athaya Urumsah (24/546678/PA/23211)  
Akhtar Gadang Abimanyu (24/532855/PA/22547)

December 20, 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Problem Background . . . . .	2
1.2	Objectives . . . . .	2
1.3	Users and Use Cases . . . . .	2
<b>2</b>	<b>Database Design</b>	<b>2</b>
2.1	Entity Relationship Diagram (ERD) . . . . .	2
2.2	Entity Explanations . . . . .	3
2.3	Normalization Steps . . . . .	3
<b>3</b>	<b>Database Implementation</b>	<b>4</b>
3.1	Relational Schema . . . . .	4
3.2	Key Constraints and SQL . . . . .	4
<b>4</b>	<b>Application Implementation</b>	<b>6</b>
4.1	Borrower Dashboard . . . . .	6
4.2	Librarian Dashboard (Approvals) . . . . .	6
4.3	Return Process . . . . .	7
4.3.1	Librarian Dashboard . . . . .	7
4.3.2	Borrower Dashboard . . . . .	8
<b>5</b>	<b>Testing and Results</b>	<b>10</b>
5.1	User Registration (CREATE) . . . . .	10
5.2	Get Books (READ) . . . . .	13
5.3	Borrowing Flow (UPDATE) . . . . .	14
5.4	Deleting a Book and its History (DELETE) . . . . .	15
<b>6</b>	<b>Conclusion and Reflection</b>	<b>15</b>
<b>7</b>	<b>References</b>	<b>16</b>

# 1 Introduction

## 1.1 Problem Background

Many traditional libraries still rely on manual logging methods or outdated spreadsheet systems to manage book loans. This manual approach leads to significant issues, including duplicate records, lost data, difficulty in tracking overdue books, and a general lack of visibility regarding book availability. Librarians often struggle to manage inventory efficiently, while borrowers lack a convenient way to check book status or history.

## 1.2 Objectives

The primary objective of the Library Borrowing Tracker is to digitize and automate core library operations. The specific goals are:

- To create a centralized relational database for storing book, user, and transaction data.
- To provide an interface for Borrowers to browse catalogs and request loans.
- To provide an interface for Librarians to manage inventory, approve requests, and process returns.
- To ensure data integrity through proper database normalization and constraints.

## 1.3 Users and Use Cases

The system is designed for two distinct user roles:

- **Borrower (Student/User):** Can register, log in, view available books, request to borrow books, and view their active loan status.
- **Librarian (Admin):** Can add new books, remove books, approve borrowing requests (setting loan duration), and process book returns.

# 2 Database Design

## 2.1 Entity Relationship Diagram (ERD)

The conceptual design of the database focuses on the relationships between Borrowers, Books, and Librarians.

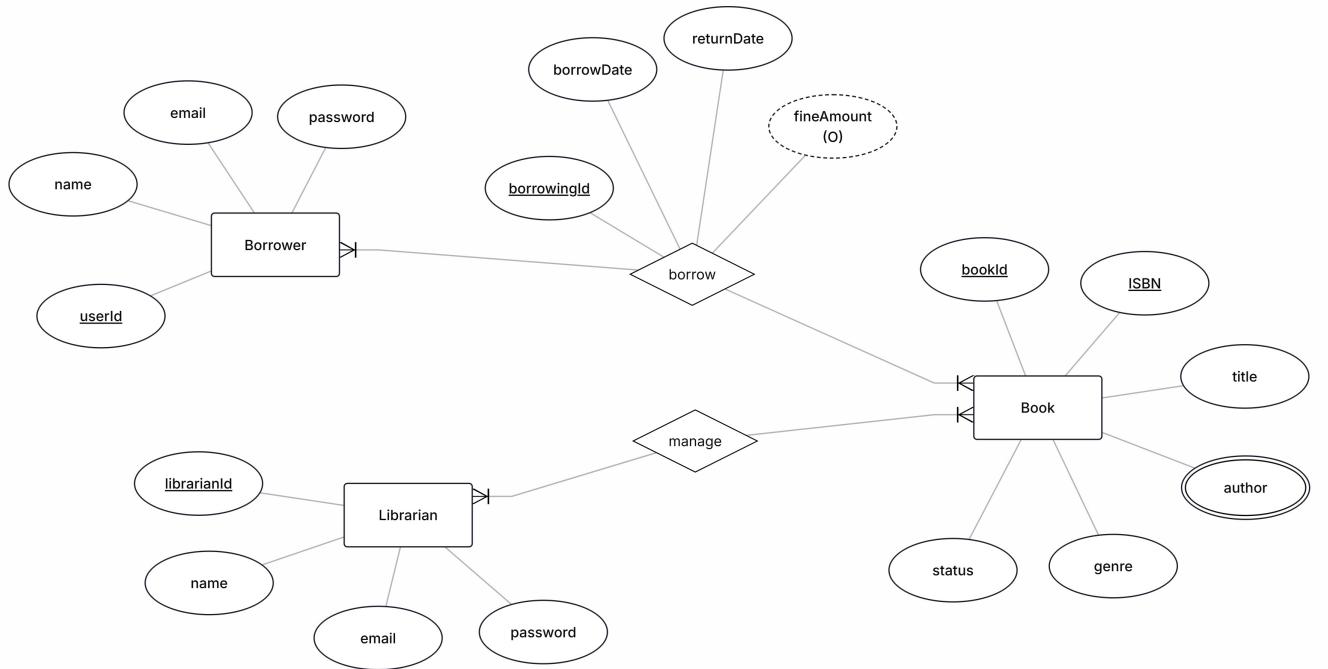


Figure 1: Entity Relationship Diagram (ERD)

## 2.2 Entity Explanations

- **Borrower**: Stores user credentials. Attributes: userID (PK), **borrowerName**, **borrowerEmail**, **borrowerPass**.
- **Book**: Represents the library inventory. Attributes: **bookID** (PK), **ISBN**, **bookTitle**, **bookGenre**, **bookStatus** (e.g., AVAILABLE, BORROWED).
- **Librarian**: Stores admin credentials. Attributes: **librarianID** (PK), **libName**, **libEmail**, **libPass**.
- **Borrowing (Transaction)**: An associative entity tracking the loan process. Attributes: **borrowID** (PK), **borrowDate**, **returnDate**, **status** (PENDING, APPROVED, RETURNED), **fineAmount**.

## 2.3 Normalization Steps

To ensure the database is efficient and free of redundancy:

- **1NF (First Normal Form)**: All attributes contain atomic values (e.g., no multiple authors in a single cell). Each record is unique.
- **2NF (Second Normal Form)**: All non-key attributes are fully dependent on the primary key. We separated "Borrowing History" into its own table so that transaction data doesn't clutter the User or Book tables.

- **3NF (Third Normal Form):** Transitive dependencies were removed. For example, book details are tied strictly to bookID, not to the borrower who currently has the book.

## 3 Database Implementation

### 3.1 Relational Schema

The physical implementation of the database in MySQL follows this schema structure.

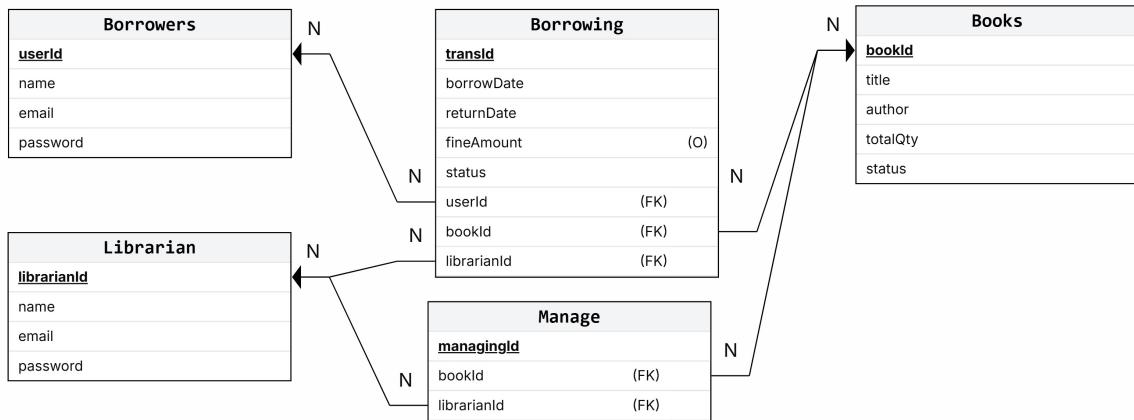


Figure 2: Relational Schema

### 3.2 Key Constraints and SQL

The system utilizes Primary Keys (PK) for unique identification and Foreign Keys (FK) to link the Borrow table to Borrower and Book.

**Example SQL for Table Creation:**

```

1 CREATE DATABASE librarydb;
2 USE librarydb;
3
4 CREATE TABLE Borrower (
5     userID INT NOTNULL AUTO_INCREMENT PRIMARY KEY ,
6     borrowerName VARCHAR(100) NOT NULL ,
7     borrowerEmail VARCHAR(100) NOT NULL UNIQUE ,
8     borrowerPass VARCHAR(255) NOT NULL
9 );
10

```

```

11 CREATE TABLE Librarian (
12     librarianID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
13     librarianName VARCHAR(100) NOT NULL,
14     librarianEmail VARCHAR(100) NOT NULL UNIQUE,
15     librarianPass VARCHAR(255) NOT NULL
16 );
17
18 CREATE TABLE Book (
19     bookID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
20     bookISBN VARCHAR(20) NOT NULL,
21     bookTitle VARCHAR(255) NOT NULL,
22     bookAuthor VARCHAR(255) NOT NULL,
23     bookGenre VARCHAR(100) NOT NULL,
24     bookStatus ENUM('available', 'pending', 'borrowed') DEFAULT 'available',
25     totalCopies INT DEFAULT 1,
26     availableCopies INT DEFAULT 1
27 );
28
29 CREATE TABLE Borrowing (
30     borrowID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
31     borrowDate DATE NOT NULL,
32     returnDate DATE NOT NULL,
33     fine DECIMAL(10,2) DEFAULT 0.00,
34     bookID INT,
35     userID INT,
36     librarianID INT,
37     dueDate DATE,
38     status ENUM('pending', 'approved', 'return_requested', 'returned')
39         DEFAULT 'pending',
40     FOREIGN KEY (bookID) REFERENCES Book(bookID),
41     FOREIGN KEY (userID) REFERENCES Borrower(userID) ON DELETE CASCADE,
42     FOREIGN KEY (librarianID) REFERENCES Librarian(librarianID) ON DELETE SET NULL
43 );

```

Listing 1: Creating the Borrow Transaction Table

**Stored Procedures/Logic:** When a book is requested, the system inserts a record into `borrowing_history` with a status of 'pending'. When approved by a librarian, the books table updates the `bookStatus` to 'UNAVAILABLE'.

**Explanation:** The provided SQL script successfully establishes the physical schema for the Library Borrowing Tracker by creating four normalized tables: Borrower, Librarian, Book, and Borrowing. The Borrower and Librarian tables manage user and staff credentials, while the Book table tracks inventory, including the crucial `availableCopies` and the book's `bookStatus` (available, pending, or borrowed). The central Borrowing ta-

ble connects these entities using Foreign Keys (bookID, userID, librarianID), maintaining the transaction details such as borrowDate, dueDate, fine amount, and the current status of the loan (pending, approved, return requested, or returned), with constraints ensuring data integrity during deletion, such as setting the associated librarianID to NULL if a librarian is removed.

## 4 Application Implementation

The application uses a Node.js/Express backend connected to a MySQL database, with a frontend interface for user interaction.

### 4.1 Borrower Dashboard

Borrowers can view the catalog and see the status of their requests.

- **Logic:** The user clicks "Borrow," which sends a POST request to the API. The status initially shows as "PENDING."

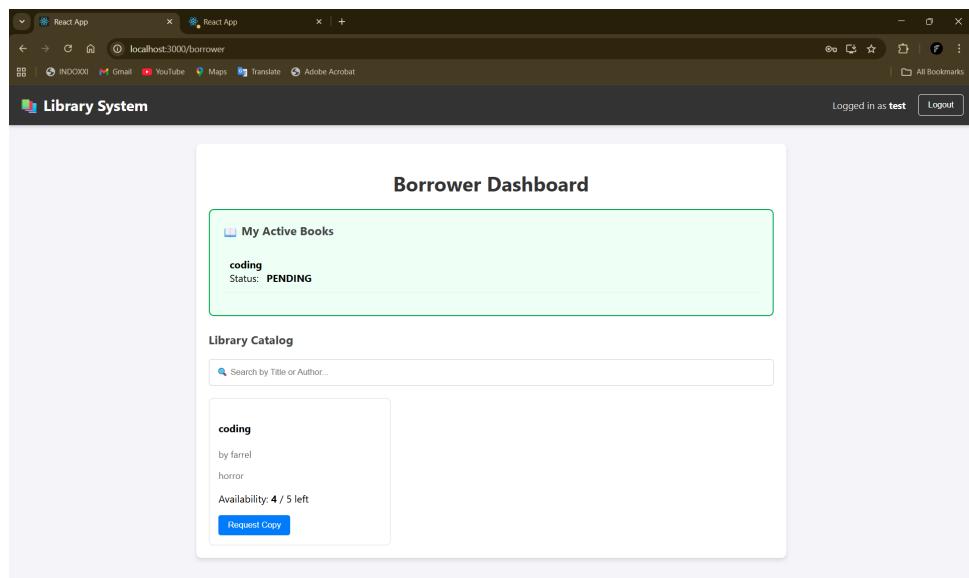


Figure 3: Borrower Dashboard showing a Pending Request

### 4.2 Librarian Dashboard (Approvals)

Librarians view a list of pending requests. They can approve a request and set the borrowing duration (e.g., 7 days).

- **Logic:** Upon approval, the `returnDate` is calculated, and the book status is updated in the database.

The screenshot shows a 'Librarian Dashboard' interface. At the top, there's a header 'Librarian Dashboard'. Below it is a section titled 'Tasks' with a bell icon. A table lists a single task: 'User' (test), 'Book' (coding), and 'Action' (a green button labeled 'Approve').

Figure 4: Librarian Approval Interface

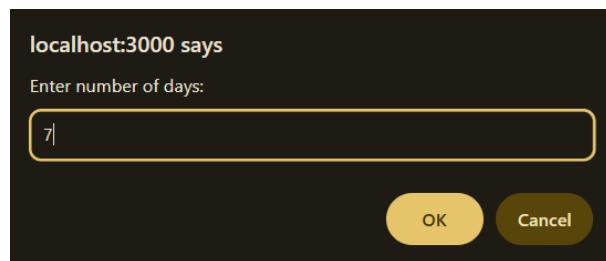


Figure 5: calculate the returnDate

### 4.3 Return Process

There are two ways to return the book. First, the librarian can enter the borrowID to process a return. Second, the borrower can request the return through their dashboard, and then the librarian can approve the request. This updates the transaction to "returned" and makes the book "AVAILABLE" again in the catalog. If the 'returnDate' is past the 'dueDate', the fine will be calculated.

#### 4.3.1 Librarian Dashboard

- Librarian Return Dashboard:

The screenshot shows the 'Librarian Dashboard'. At the top, there's a header 'Librarian Dashboard'. Below it is a section titled 'Tasks' with a bell icon, which displays the message 'No pending tasks.' A second section titled 'Quick Return' contains a 'Borrow ID' input field and a blue 'Return' button.

Figure 6: Librarian Dashboard

	borrowID	borrowDate	returnDate	fine	bookID	userID	librarianID	dueDate	status
▶	1	2025-12-03	2025-12-24	70.00	1	1	1	2025-12-10	returned
	2	2025-12-24	NULL	0.00	1	1	1	2025-12-31	approved

Figure 7: Check the borrowID

- Processing 'Return' by entering the borrowID:

A screenshot of a web-based application interface titled 'Quick Return'. It features a search bar containing the number '2' and a blue 'Return' button to its right.

Figure 8: Enter borrowID to 'Return'

- After the book is returned, the database will be updated

A screenshot of a 'All Books' page. It shows a search bar with placeholder text 'Search by Title, Author, or ISBN...'. Below the search bar, there is a list item for a book titled 'coding' by 'farrel'. The list item indicates 'Copies: 5 / 5' and includes 'Edit' and 'Delete' buttons.

Figure 9: Number of copies will be increased by 1 after returning

#### 4.3.2 Borrower Dashboard

- Borrower Return Dashboard:

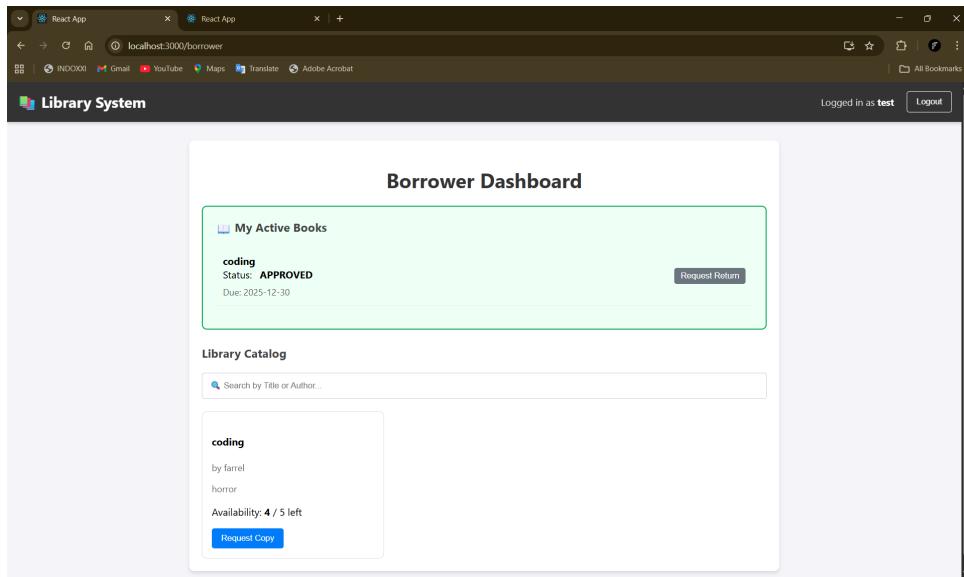


Figure 10: Borrower Dashboard

- Processing 'Return' by request the return

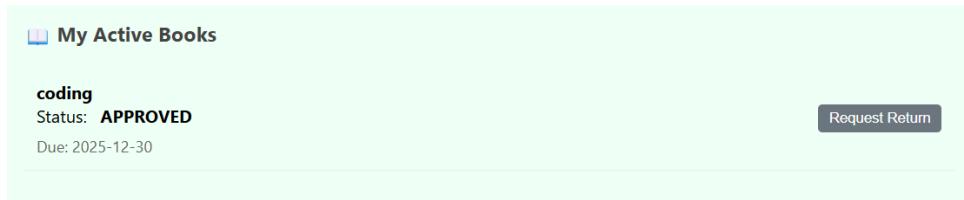


Figure 11: request the return

- Approve the return request

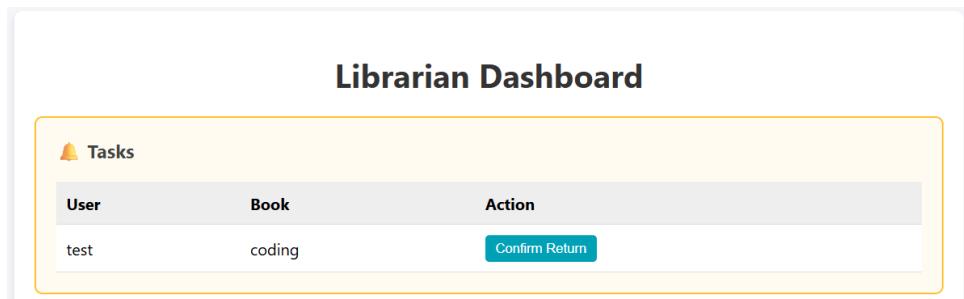


Figure 12: Librarian approves the return request

## 5 Testing and Results

We performed CRUD (Create, Read, Update, Delete) testing using Postman to verify the backend API before integrating it with the frontend.

### 5.1 User Registration (CREATE)

Tested the POST /borrower/register endpoint.

- **Input:** JSON containing name, email, and password.
- **Result:** 200 OK, "Borrower registered."

The screenshot shows the Postman interface for a POST request to `http://localhost:5000/borrower/register`. The request body is a JSON object with fields: `borrowerName: "Test User"`, `borrowerEmail: "test@example.com"`, and `borrowerPass: "password123"`. The response body is a JSON object with a single field: `message: "Borrower registered"`. Below the requests, a table displays the database results for the registered user, showing columns: userID, borrowerName, borrowerEmail, and borrowerPass. The table has one row with values: 1, Test User, test@example.com, and a hashed password.

	userID	borrowerName	borrowerEmail	borrowerPass
▶	1	Test User	test@example.com	\$2b\$10\$d/kIBgwInEY6KHZQ4BbetB2hiqQISp...
*	HULL	HULL	HULL	HULL

Figure 13: API Testing - Register Borrower

- **Register Librarian and Login:** A librarian, "Boss," was registered and logged in to acquire an authorization token required for privileged actions.

The screenshot shows the Postman interface for testing an API endpoint at `http://localhost:5000/librarian/register`. The request method is `POST`. The `Body` tab is selected, showing a JSON payload:

```

1  {
2    "librarianName": "Boss",
3    "librarianEmail": "boss@lib.com",
4    "librarianPass": "123"
5  }

```

The response status is `200 OK`, and the response body is:

```

1  {
2    "message": "Librarian registered"
3  }

```

A table below shows the registered librarian data:

	librarianID	librarianName	librarianEmail	librarianPass
▶	1	Boss	boss@lib.com	\$2b\$10\$Pq0BjdhFJ3dd5u/bG9p.6uKlrZCo0LwsT...
*	HULL	NULL	NULL	HULL

Figure 14: API Testing - Register Librarian

The screenshot shows the Postman interface for testing an API endpoint at `http://localhost:5000/librarian/login`. The request method is `POST`. The `Body` tab is selected, showing a JSON payload:

```

1  {
2    "librarianName": "Boss",
3    "librarianEmail": "boss@lib.com",
4    "librarianPass": "123"
5  }

```

The response status is `200 OK`, and the response body is:

```

1  {
2    "message": "Login success",
3    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6M5w1cm9eZSI6ImxgYnJhm1hbisInhdCi6MTc2NDI1NTExNSeiZXhiIjoxNzY0MjgzOTc1fQ.8q0078UMyeysXeotsZZ9Os83Tj6XRMyygiuhBY-HWMy"
4  }

```

Figure 15: API Testing - Librarian Login

- **Add Book:** A book titled "Database Introduction" was successfully added using the librarian's token.

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/books`. The 'Authorization' tab is selected, showing 'Bearer Token' and a placeholder for the token value. A note below states: 'The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.'

Figure 16: API Testing - Librarian's token to add book

The screenshot shows the Postman interface with a POST request to `http://localhost:5000/books`. The 'Body' tab is selected, showing a JSON payload:

```

1  {
2    "bookISBN": "000-001",
3    "bookTitle": "Database Introduction",
4    "bookGenre": "Education"
5  }

```

The response status is 200 OK. The response body is:

```

1  {
2    "message": "Book added"
3  }

```

A table at the bottom shows the database entry:

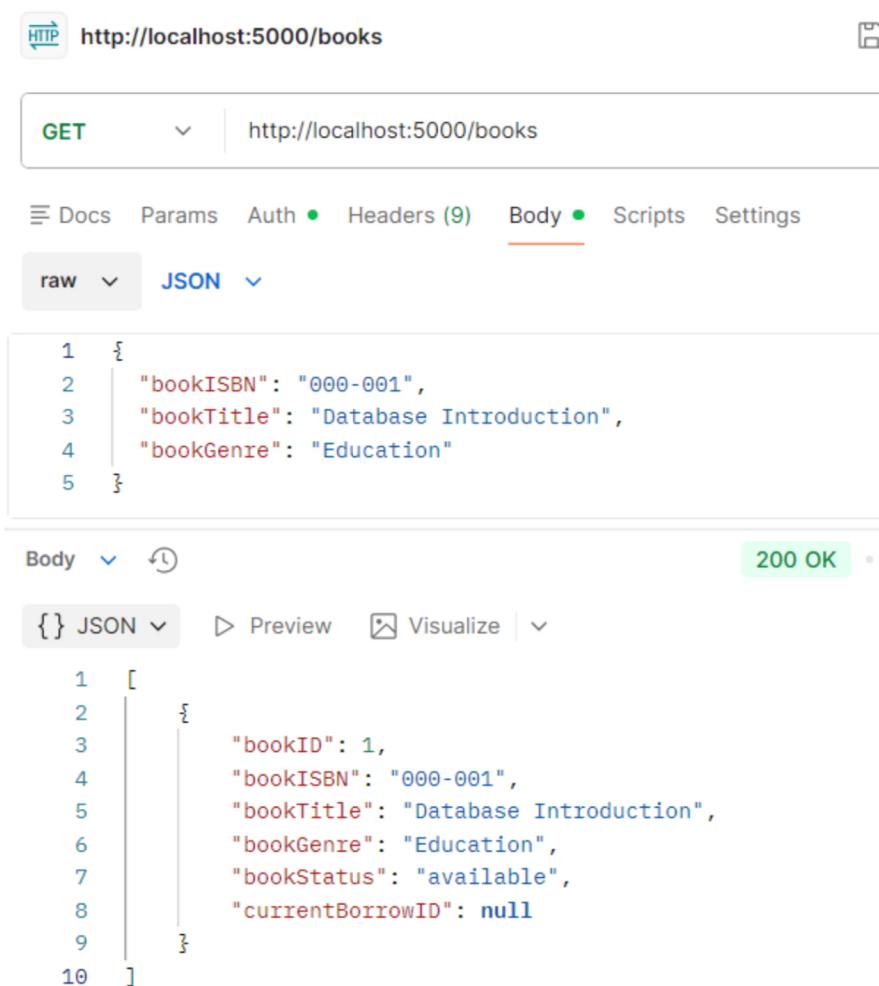
	bookID	bookISBN	bookTitle	bookGenre	bookStatus
▶	1	000-001	Database Introduction	Education	available
*	NULL	NULL	NULL	NULL	NULL

Figure 17: API Testing - Add book

## 5.2 Get Books (READ)

This operation tests the system's ability to retrieve and display current library inventory.

- **Endpoint:** GET `http://localhost:5000/books`.
- **Result:** A 200 OK response was received, returning a JSON array containing the newly added book, confirming it is in the database. The output showed:
  - bookID: 1
  - bookTitle: "Database Introduction"
  - bookStatus: "available"



The screenshot shows the Postman application interface. At the top, there is a header bar with icons for HTTP, URL (`http://localhost:5000/books`), and a document icon. Below the header, the main interface has a search bar with `GET` selected and the URL `http://localhost:5000/books`. Underneath the search bar, there are tabs for `Docs`, `Params`, `Auth`, `Headers (9)`, `Body` (which is currently selected), `Scripts`, and `Settings`. The `Body` tab has two dropdowns: `raw` and `JSON`, with `JSON` selected. The JSON response body is displayed in a code editor-like area:

```
1  {
2    "bookISBN": "000-001",
3    "bookTitle": "Database Introduction",
4    "bookGenre": "Education"
5 }
```

Below the JSON response, there is a status indicator showing `200 OK`. At the bottom of the interface, there are buttons for `{}`, `JSON`, `Preview`, `Visualize`, and a refresh icon.

Figure 18: Read books information

### 5.3 Borrowing Flow (UPDATE)

Tested the lifecycle of a book loan.

- **Request:** POST /borrow → Message: "Request sent."

The screenshot shows the Postman interface for a POST request to `http://localhost:5000/borrow`. The request body is JSON, containing `{"bookID": 1}`. The response status is 200 OK, and the message is "Request sent".

```
1 {  
2   |   "bookID": 1  
3 }  
  
1 {  
2   |   "message": "Request sent"  
3 }
```

Figure 19: API Testing - Borrow request sent

- **Approve:** POST /borrow/approve → Input: borrowID, days. → Message: "Approved."

The screenshot shows the Postman interface for a POST request to `http://localhost:5000/borrow/approve`. The request body is JSON, containing `{"borrowID": 1, "days": 7}`. The response status is 200 OK, and the message is "Approved". Below the interface, a table shows the updated state of the book record.

	bookID	bookISBN	bookTitle	bookGenre	bookStatus
▶	1	000-001	Database Introduction	Education	borrowed
●	HULL	HULL	HULL	HULL	HULL

Figure 20: API Testing - Borrow request approved

## 5.4 Deleting a Book and its History (DELETE)

This operation tests the deletion functionality, ensuring relational integrity via constraints.

- Endpoint: DELETE `http://localhost:5000/books/1` (requires the librarian's Bearer Token).
- Logic: The API endpoint handles the deletion of the book from the Book table and, due to the foreign key constraint on the Borrowing table, it also deletes the associated borrowing history.
- Result: A 200 OK response was returned with the message: "Book and history deleted".
- Verification: Post-deletion, a check confirmed that the book record (ID 1) no longer exists in the database.

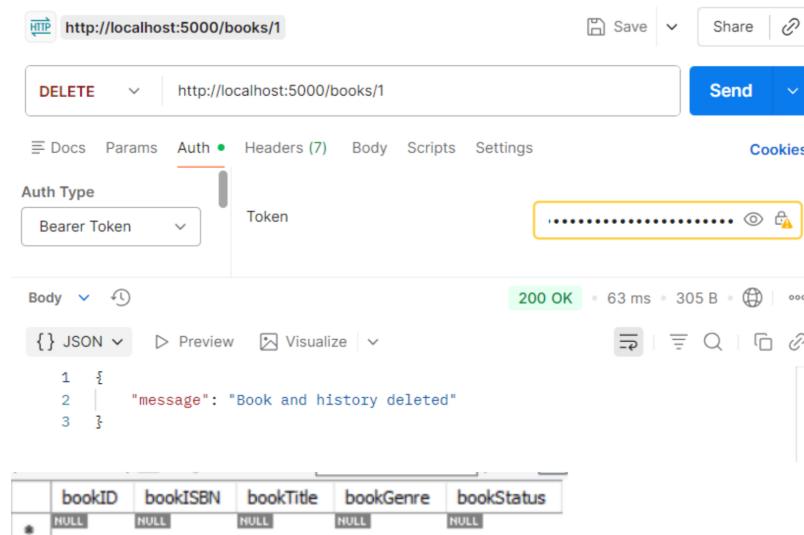


Figure 21: API Testing - Book and history successfully deleted

## 6 Conclusion and Reflection

This project successfully implemented a functional Library Borrowing Tracker for conventional libraries. We learned valuable lessons in integrating a relational MySQL database with a Node.js backend, specifically regarding the handling of asynchronous queries and managing Foreign Key constraints (e.g., ensuring a book cannot be deleted if it is currently borrowed).

**Challenges & Improvements:** One challenge was synchronizing the status updates between the Books table and the Borrowing table to ensure the catalog accurately reflects availability. In the future, we could improve the system by adding automated fine calculations based on the `returnDate` and implementing email notifications for overdue books. Overall, the system meets the core objectives of digitizing the library's workflow.

## 7 References

- [1] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. New York, NY, USA: McGraw-Hill Education, 2020.