Final Report: The University Library Management System

Hongbo Li, Ruiqi Li,

Bachelor of Computer Science, UNB Saint John

CS 1103: Introduction to Databases

Professor: Jong-Kyou Kim, PhD

April 23, 2025

Abstract

This project aims to simulate and build a university library management system to make book borrowing and returning more efficient and the organization process more convenient. The system uses SQLite to design a relational database and integrates with a Java-based graphical interface through JDBC. The core functions include user and book management, loan tracking, and book classification. During the development process, we encountered some challenges in how to maintain data consistency and improve query performance. This project enabled our team to accumulate practical experience in the actual application of database structure and application development.

Introduction

Some tasks in a library, such as tracking books and users, can be tedious and inefficient if done manually. After looking for some library management systems, this project for a university library management system aims to improve the overall workflow and resource management of the library by automating common library operations such as managing users, tracking borrowed books, and updating the availability of books. SQLite was used to handle the data, and a simple and easy-to-use interface was built using Java. Database concepts such as schema design, SQL operations (DDL and DML), and JDBC integration were applied throughout the process to connect the interface with the database backend. The result is a more streamlined and reliable library management system.

The System

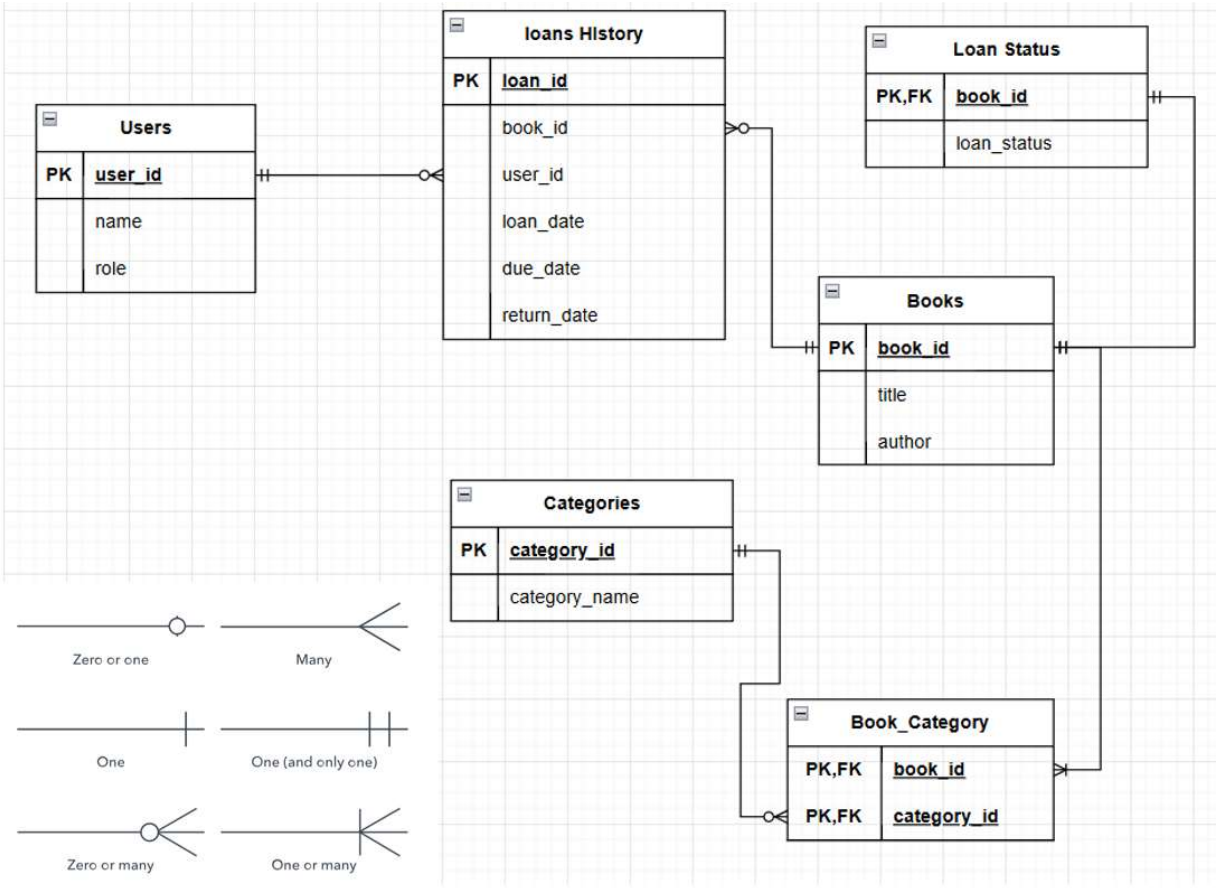
Requirements Satisfied:

1. Database Design (include ER Diagram, DDL statements, Insert statements)

The system utilizes a relational database schema with SQLite, designed to enforce data integrity and support efficient querying.

- Key tables include:
    - Users: Tracks user details (user_id, name, role).
    - Books: Stores book metadata (book_id, title, author).
    - Categories: Defines book classifications (category_id, category_name).
    - Book_Category: Manages many-to-many relationships between books and categories.
    - Loans_History: Records borrowing history (loan_date, due_date, return_date).
    - Loans_Status: Tracks real-time availability (loan_status).

- Referential Integrity:
    - Foreign keys (e.g., Loans_History.user_id) enforce relationships.
    - Foreign keys with ON DELETE CASCADE ensure automatic cleanup of related records (e.g., deleting a book removes its entries in Loans_History).
    - CHECK constraints validate domain-specific rules (e.g., role IN ('Student', 'Professor', 'Administrator')).



Appendix(ER-Diagram)

- DDL/DML Statements (Complete Schema.txt file on GitHub)
  - Write DDL statements for entities and relationships.

    The two images correspond to 2 entities and a relationship statement example respectively

```
CREATE TABLE Books
(
    book_id     INTEGER PRIMARY KEY,
    title       VARCHAR(255),
    author      VARCHAR(100)
);
```

```
CREATE TABLE Book_Category
(
    book_id         INTEGER,
    category_id     INTEGER,
    PRIMARY KEY (book_id, category_id),
    FOREIGN KEY (book_id) REFERENCES Books(book_id) ON DELETE CASCADE,
    FOREIGN KEY (category_id) REFERENCES Categories(category_id) ON DELETE CASCADE
);
```

- Sample insert statements for each table (both entity and the relationship)

```
INSERT INTO Users (user_id, name, role) VALUES (8, 'Heidi', 'Student')

INSERT INTO Book_Category (book_id, category_id) VALUES (2, 3), (2, 4), (2, 6)
```

2. JDBC Integration
    - The Java application connects to SQLite via JDBC and executes DDL/DML operations:
        - Database Connection:

```
public MethodHouse(String dbPath) throws SQLException
{
    conn = DriverManager.getConnection("jdbc:sqlite:" + dbPath);
}
```

        - Schema Creation: Executes the DDL statements on initialization to create tables.

3. Core Function Implementation

To meet the moderate and challenge requirements outlined in the project guidelines, we extended the system's capabilities by implementing seven core functionalities that integrate JDBC-driven database operations with a JavaFX GUI. The University Library Management System combines search operations (e.g., borrowing records, keyword-based availability checks, and category filtering) with administrative data management tasks (e.g., adding books, users, categories, and loans), fulfilling both functional and technical objectives of the project.

- Search Functions
  - i) Search User Borrowing Records
    - a. Purpose: Retrieve a user's complete loan history with dates and return status.
    - b. SQL Query:

```
1 SELECT Books.title, Loans_History.loan_date, Loans_History.return_date
2 FROM Loans_History
3 JOIN Books ON Loans_History.book_id = Books.book_id
4 WHERE user_id = ?;
```

    - c. Java: Uses Prepared Statement to prevent SQL injection.

      Validates user ID input to ensure it is an integer.

  - ii) Search Books by Keyword (Check Availability)
    - a. Purpose: Search books by title keyword and display real-time availability.
    - b. SQL Query:

```
7 SELECT Books.title, Loans_Status.loan_status
8 FROM Books
9 JOIN Loans_Status ON Books.book_id = Loans_Status.book_id
0 WHERE LOWER(Books.title) LIKE ?;
```

c. Java: Case-insensitive search using LOWER() and wildcards (%keyword%)

iii) List Books by Category

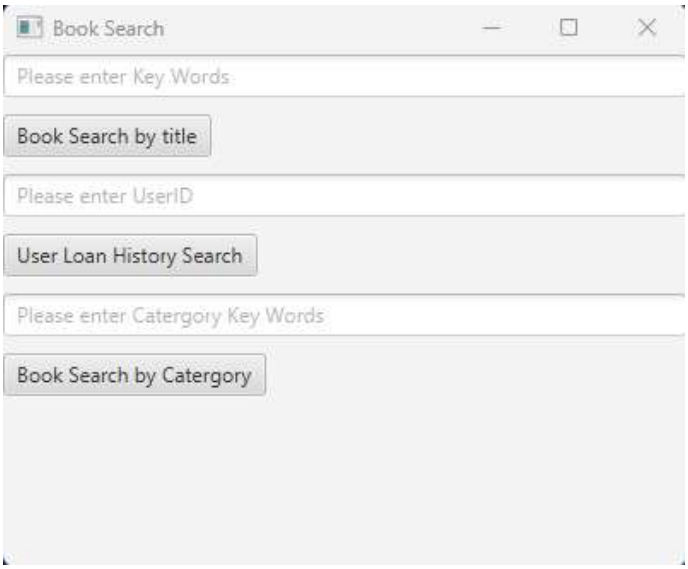
a. Purpose: Filter books by category keyword.

b. SQL Query:

```
SELECT Books.title, Categories.category_name
FROM Books
JOIN Book_Category ON Books.book_id = Book_Category.book_id
JOIN Categories ON Book_Category.category_id = Categories.category_id
WHERE LOWER(category_name) LIKE ?;
```

c. Java: Double JOIN to resolve many-to-many relationships.

iv) GUI:



The search interface (MainFXApp.java) enables efficient data retrieval:

- Keyword Search:

"Book Search by title": A text field accepts keywords (e.g., "introduction"), executing a case-insensitive search across Books.title and returning titles with real-time availability from Loans_Status.

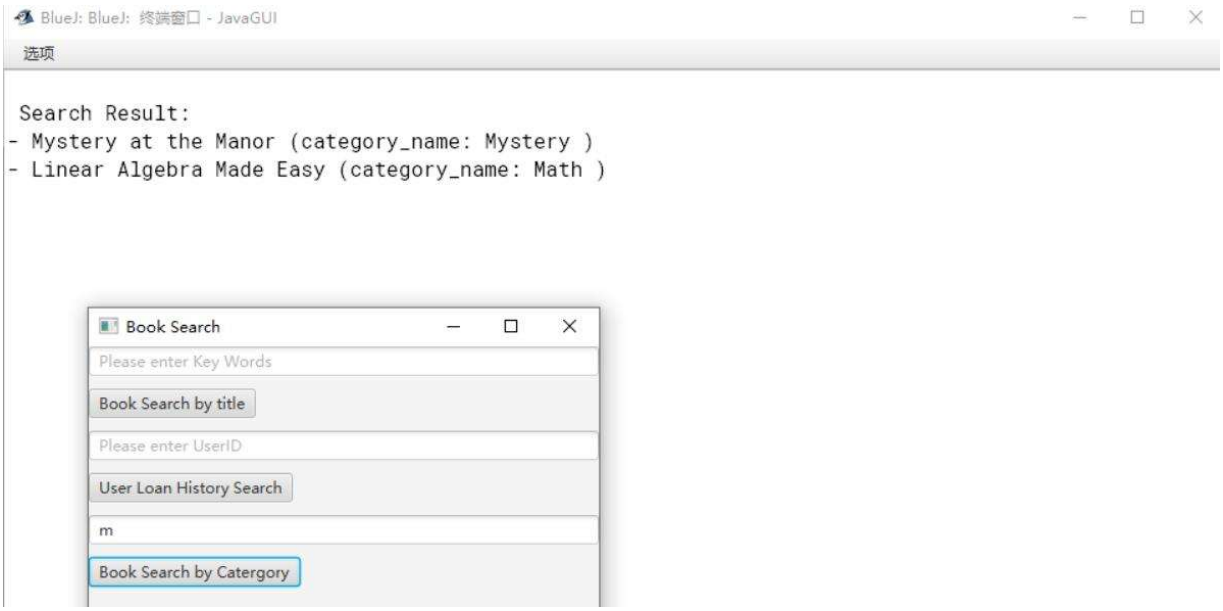
- User Loan History:

"User Loan History Search": Requires a valid User ID to fetch borrowing records from Loans_History, displaying loan dates, return dates, and unreturned statuses.

- Category Filtering:

"Book Search by Category": Accepts category keywords and lists books linked to matching categories via Book_Category and Categories tables.

e.g.



- Administrative Functions

  i) Add Book

    - Purpose: Insert a new book into Books, assign it to a category in Book_Category, and set its default availability in Loans_Status.

  ii) Add Category

    - Purpose: Create a new category in the Categories table.

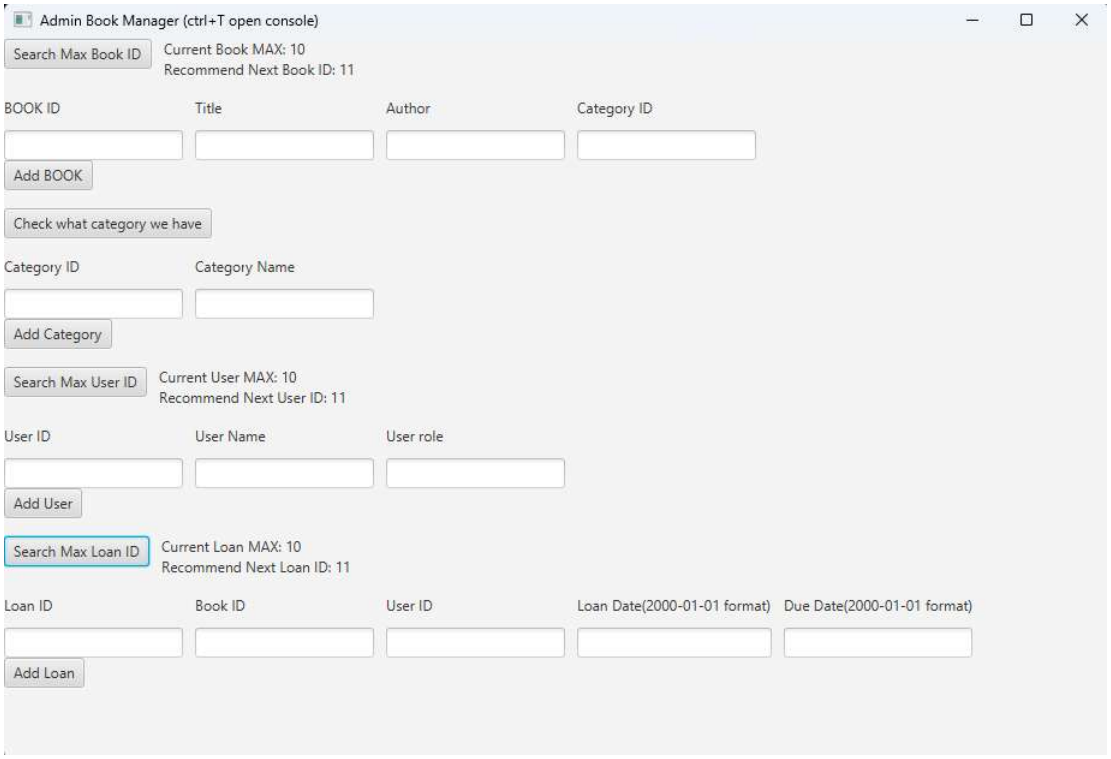
  iii) Add User

    - Purpose: Register a new user in the Users table.

  iv) Add Loan

    - Purpose: Record a new loan in Loans_History and update the book's status to Loaned.

  v) GUI:



The administrative interface (AdminApp.java) provides tools for managing library resources and users:

- ID Management:

  Buttons like "Search Max Book ID" dynamically retrieve the highest current ID (e.g., 10) and recommend the next available ID (e.g., 11) to prevent conflicts.

- Data Entry Forms:
  - Add Book:

    Input fields for Book ID, Title, Author, and Category ID. The "Add BOOK" button triggers transactional inserts into Books, Book_Category, and Loans_Status tables.
  - Add Category:

    Fields for Category ID and Category Name. Validates uniqueness and updates the Categories table.
  - Add User:

    Inputs for User ID, Name, and Role, with role validation (Student, Professor, Administrator).
  - Add Loan:

    Captures Loan ID, Book ID, User ID, Loan Date, and Due Date. Ensures book availability and updates Loans_History and Loans_Status.

Conclusion

Through the development of the University Library Management System, we integrated seven core features into a JavaFX application using SQLite and JDBC. In this process, we learned to design normalized database tables (e.g., *Users*, *Books*, *Loans_History*), apply foreign key constraints (ON DELETE CASCADE), and manage transactions (COMMIT/ROLLBACK, setAutoCommit(false)) to ensure data integrity. We practiced writing complex SQL queries with multi-table JOINs, optimized performance using indexes, and secured operations through Prepared Statement. The JavaFX GUI supports input validation, dynamic data display, and reusable components that enhance usability. This project emphasizes the importance of combining sound database design with user-friendly, reliable application development.

Bibliography

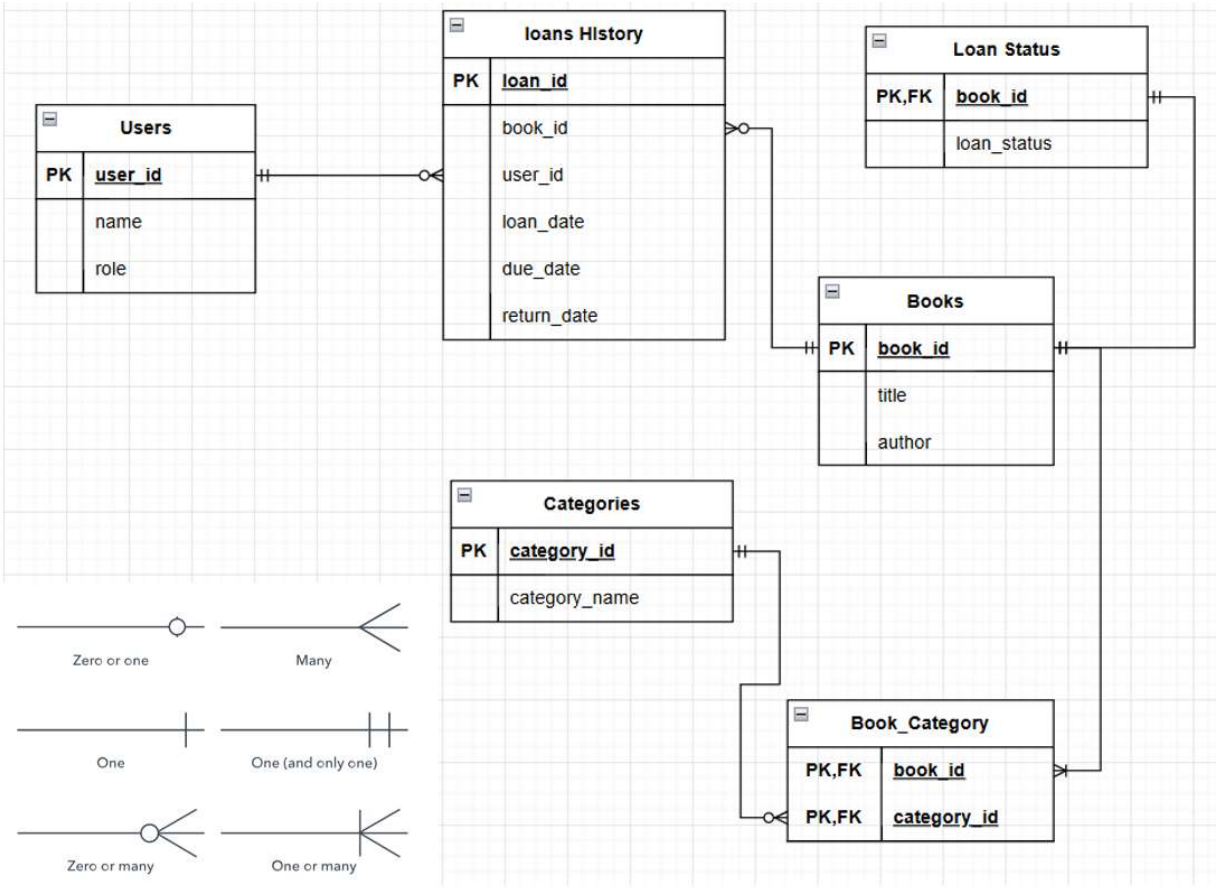
SQLite Online, Java/ JavaFX

The Contents of CS1103 Labs

Appendix

The source codes all in the GitHub: Hongbo-L-i/CS1103-Project

ER-Diagram: Same image as front


Users
PK
user_id
name
role
loans History
PK
loan_id
book_id
user_id
loan_date
due_date
return_date
Loan Status
PK,FK
book_id
loan_status
Books
PK
book_id
title
author
Categories
PK
category_id
category_name
Book_Category
PK,FK
book_id
PK,FK
category_id
Zero or one
Many
One
One (and only one)
Zero or many
One or many