MINI PROJECT

ON

**LIBRARY MANAGEMENT SYSTEM**

Submitted by

**PRANEETH WARAN K**

**NC.SC.U4CSE24222**

For

23CSE111- Object Oriented Programming

II Semester

B.Tech. CSE

School of Computing

Amrita Vishwa Vidyapeetham, Nagercoil

**si.no. chaptername pg.no**

1. **Introduction 1**
2. **Problem statement 4**
3. **Objectives 5**
4. **MODULES OF PROJECT 7**
5. **CODE 10**
6. **OUTPUT SCREENSHOTS 23**
7. **UML DIAGRAM 27**
8. **APPLICATION OF PROJECT 31**
9. **LIMITATIONS OF PROJECT 33**
10. **BIBLIOGRAPHY 34**
11. **GITHUB LINK OF THE PROJECT 35**

**INTRODUCTION**

**Project Overview**

* The Library Management System is a software application designed to manage the daily operations of a library.
* It automates tasks such as adding books, issuing and returning books, calculating fines, and maintaining user and book records.

**Purpose of the System**

* To simplify and streamline library operations.
* To reduce manual errors and improve efficiency.
* To offer users quick access to book borrowing and return features.

**Target Users**

* Library staff (admin/manager) for adding and updating books.
* Students or members for borrowing, returning, and checking fines.

**1**

**Key Features**

* Add, update, and view books.
* Borrow and return books with automatic return date calculation.
* Fine calculation for late returns.
* User-friendly console-based interface.

**Technologies Used**

* Frontend: Java Console-based UI
* Backend: Java (Core Java, JDBC)
* Database: MySQL

**Motivation**

* Traditional library systems are time-consuming and prone to errors.
* This system offers a digital, more reliable alternative that saves time and resources.

**2**

**Scalability**

* Can be extended with a graphical interface or web portal.
* Additional features like user authentication, book search filters, and book availability tracking can be added later.

**3**

**PROBLEM STATEMENT**

Due the following issues I made the solution to overcome these problems in the upcoming days

In many educational institutions and public libraries, the management of book records, issue/return processes, and fine calculations is often done manually or through outdated systems. This manual approach is not only time-consuming but also prone to human error, leading to misplaced books, incorrect fine records, and inefficient tracking of borrowed materials.

There is a pressing need for a system that can automate these library operations efficiently. The lack of a centralized and computerized system makes it difficult for users to borrow or return books quickly and for library staff to manage records accurately.

Therefore, a robust, user-friendly **Library Management System** is required to streamline the process of managing book inventory, handling borrowing/return transactions, and automating fine calculations, ensuring efficiency, reliability, and accuracy in library operations.

**4**

**OBJECTIVES**

**Automate Library Operations**To develop a system that automates key tasks such as adding books, updating book details, issuing and returning books, and calculating fines.

**Improve Accuracy and Efficiency**To minimize manual errors and reduce the time taken for managing library tasks through a computerized solution.

**Simplify Book Borrowing and Returning**  
To provide an easy-to-use interface for users to borrow and return books with predefined borrowing periods and return dates.

**Enable Fine Calculation for Late Returns**  
To implement an automatic fine calculation feature based on the number of days a book is returned late.

**5**

**Ensure Data Storage and Retrieval**  
To securely store book and user records in a MySQL database, allowing easy retrieval and updates when needed.

**User-Friendly Console Interface**  
To create a simple and interactive console-based menu system for managing the library without requiring technical expertise.

**Build a Scalable Foundation**  
To design the system in a way that it can be extended in the future with features like user login, book availability checking, search functionality, or a graphical/web interface.

**6**

**MODULES IN JAVA(AS FILE)**

**PROJECT NAME:** MINI PRJECT

**PACKAGE NAME:** LIBRARY MANAGEMENT SYSTEM

**OTHER MODULES:**

1. **Book Management Module(file as bookmanager.java)**
   * Add new books with details like title, author, year, ISBN, and username.
   * Update existing book records.
   * Store and retrieve book data from the database.
2. **User Management Module(file as main.java)**
   * Handles the basic user interaction (via console input).
   * Accepts user information for borrowing and returning books.
   * Tracks which user has borrowed which book.

7

3.**Borrow & Return Module(file as book.java)**

* + Allows users to borrow books with a default borrowing period of 1 month.
  + Manages return of books and records actual return dates.
  + Prevents duplicate or invalid borrowing/return entries.

4.**Fine Calculation Module(file as bookmain.java)**

* + Automatically calculates fines for overdue book returns.
  + Uses a fixed fine rate (e.g., $5 per day late).
  + Displays fine details per book and total fine per user.

8

1. **Database Management Module(file as dbconnection.java )**
   * Connects the application to a MySQL database using JDBC.
   * Executes SQL operations (INSERT, UPDATE, SELECT) for various modules.
   * Ensures proper database connection handling and error catching.

6.**Console Menu Module**

* + Provides a user-friendly console interface for interaction.
  + Displays options such as Add Book, Borrow, Return, Check Fine, Update Book, and Exit.
  + Controls flow and input validation.

**9**

**Code**

**package LibraryManagementSystem;**

**import java.sql.\*;**

**public class DBConnection {**

**private static final String *URL* = "jdbc:mysql://localhost:3306/library\_db";**

**private static final String *USER* = "root";**

**private static final String *PASSWORD* = "praneeth$$6381851557"; // Change if your MySQL has a password**

**public static Connection getConnection() {**

**try {**

**return DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);**

**} catch (SQLException e) {**

**System.*out*.println("DB Connection failed: " + e.getMessage());**

**return null;**

**}**

**}**

**}**

**public class BookManager {**

**public static void addBook(Book book) {**

**Connection conn = DBConnection.*getConnection*();**

**if (conn == null) return;**

**10**

**try {**

**String sql = "INSERT INTO books (title, author, year, isbn, username) VALUES (?, ?, ?, ?, ?)";**

**PreparedStatement stmt = conn.prepareStatement(sql);**

**stmt.setString(1, book.getTitle());**

**stmt.setString(2, book.getAuthor());**

**stmt.setInt(3, book.getYear());**

**stmt.setString(4, book.getIsbn());**

**stmt.setString(5, book.getUsername());**

**stmt.executeUpdate();**

**System.*out*.println("Book added successfully.");**

**stmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void updateBook(int bookId, String newTitle, int newId) {**

**Connection conn = DBConnection.*getConnection*();**

**if (conn == null) return;**

**11**

**try {**

**String query = "UPDATE books SET ";**

**boolean hasTitle = !newTitle.isEmpty();**

**boolean hasId = newId != -1;**

**if (hasTitle) query += "title = ?";**

**if (hasId) {**

**if (hasTitle) query += ", ";**

**query += "id = ?";**

**}**

**query += " WHERE id = ?";**

**PreparedStatement stmt = conn.prepareStatement(query);**

**int index = 1;**

**if (hasTitle) stmt.setString(index++, newTitle);**

**if (hasId) stmt.setInt(index++, newId);**

**stmt.setInt(index, bookId);**

**int rows = stmt.executeUpdate();**

**System.*out*.println(rows > 0 ? "Book updated." : "Book ID not found.");**

**stmt.close();**

**conn.close();**

**12**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void borrowBook(String username, String isbn) {**

**Connection conn = DBConnection.*getConnection*();**

**if (conn == null) return;**

**try {**

**LocalDate borrowDate = LocalDate.*now*();**

**LocalDate returnDate = borrowDate.plusMonths(1);**

**String sql = "INSERT INTO borrowed\_books (username, isbn, borrow\_date, return\_date) VALUES (?, ?, ?, ?)";**

**PreparedStatement stmt = conn.prepareStatement(sql);**

**stmt.setString(1, username);**

**stmt.setString(2, isbn);**

**stmt.setDate(3, Date.*valueOf*(borrowDate));**

**stmt.setDate(4, Date.*valueOf*(returnDate));**

**stmt.executeUpdate();**

**System.*out*.println("Book borrowed! Return by: " + returnDate);**

**stmt.close();**

**conn.close();**

**13**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void returnBook(String username, String isbn) {**

**Connection conn = DBConnection.*getConnection*();**

**if (conn == null) return;**

**try {**

**LocalDate today = LocalDate.*now*();**

**String sql = "UPDATE borrowed\_books SET actual\_return\_date = ? WHERE username = ? AND isbn = ? AND actual\_return\_date IS NULL";**

**PreparedStatement stmt = conn.prepareStatement(sql);**

**stmt.setDate(1, Date.*valueOf*(today));**

**stmt.setString(2, username);**

**stmt.setString(3, isbn);**

**int updated = stmt.executeUpdate();**

**System.*out*.println(updated > 0 ? "Book returned." : "No record found.");**

**stmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**14**

**}**

**}**

**public static void calculateFine(String username) {**

**Connection conn = DBConnection.*getConnection*();**

**if (conn == null) return;**

**try {**

**String sql = "SELECT isbn, return\_date, actual\_return\_date FROM borrowed\_books WHERE username = ?";**

**PreparedStatement stmt = conn.prepareStatement(sql);**

**stmt.setString(1, username);**

**ResultSet rs = stmt.executeQuery();**

**int totalFine = 0;**

**while (rs.next()) {**

**Date returnDate = rs.getDate("return\_date");**

**Date actualReturn = rs.getDate("actual\_return\_date");**

**if (actualReturn != null && actualReturn.after(returnDate)) {**

**long daysLate = ChronoUnit.*DAYS*.between(returnDate.toLocalDate(), actualReturn.toLocalDate());**

**int fine = (int) daysLate \* 5;**

**totalFine += fine;**

**15**

**System.*out*.println("Book ISBN: " + rs.getString("isbn") + " | Days Late: " + daysLate + " | Fine: $" + fine);**

**}**

**}**

**System.*out*.println("Total fine for user '" + username + "': $" + totalFine);**

**rs.close();**

**stmt.close();**

**conn.close();**

**} catch (SQLException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**public class Book {**

**private String title;**

**private String author;**

**private int year;**

**private String isbn;**

**private String username;**

**16**

**public Book(String title, String author, int year, String isbn, String username) {**

**this.title = title;**

**this.author = author;**

**this.year = year;**

**this.isbn = isbn;**

**this.username = username;**

**}**

**public String getTitle() { return title; }**

**public String getAuthor() { return author; }**

**public int getYear() { return year; }**

**public String getIsbn() { return isbn; }**

**public String getUsername() { return username; }**

**}**

**public class BookMain {**

**public static void main(String[] args) {**

**BookManager bm = new BookManager();**

**bm.getAllBooks();**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**Scanner sc = new Scanner(System.*in*);**

**int choice;**

**17**

**do {**

**System.*out*.println("\n1. Add Book");**

**System.*out*.println("2. Check Fine");**

**System.*out*.println("3. Update Book");**

**System.*out*.println("4. Borrow Book");**

**System.*out*.println("5. Return Book");**

**System.*out*.println("6. Exit");**

**System.*out*.print("Enter your choice: ");**

**choice = sc.nextInt();**

**sc.nextLine(); // clear buffer**

**switch (choice) {**

**case 1:**

**System.*out*.print("Title: ");**

**String title = sc.nextLine();**

**System.*out*.print("Author: ");**

**String author = sc.nextLine();**

**System.*out*.print("Year: ");**

**int year = sc.nextInt();**

**sc.nextLine();**

**System.*out*.print("ISBN: ");**

**String isbn = sc.nextLine();**

**System.*out*.print("Username: ");**

**String username = sc.nextLine();**

**Book book = new Book(title, author, year, isbn, username);**

**BookManager.*addBook*(book);**

**break;**

**18**

**case 2:**

**System.*out*.print("Enter username: ");**

**username = sc.nextLine();**

**BookManager.*calculateFine*(username);**

**break;**

**case 3:**

**System.*out*.print("Enter book ID to update: ");**

**int bookId = sc.nextInt();**

**sc.nextLine();**

**System.*out*.print("Enter new Title (leave blank to skip): ");**

**String newTitle = sc.nextLine();**

**System.*out*.print("Enter new ID (or -1 to skip): ");**

**int newId = sc.nextInt();**

**sc.nextLine();**

**BookManager.*updateBook*(bookId, newTitle, newId);**

**break;**

**case 4:**

**System.*out*.print("Enter your username: ");**

**username = sc.nextLine();**

**System.*out*.print("Enter ISBN of book: ");**

**isbn = sc.nextLine();**

**BookManager.*borrowBook*(username, isbn);**

**break;**

**19**

**case 5:**

**System.*out*.print("Enter your username: ");**

**username = sc.nextLine();**

**System.*out*.print("Enter ISBN of book: ");**

**isbn = sc.nextLine();**

**BookManager.*returnBook*(username, isbn);**

**break;**

**case 6:**

**System.*out*.println("Exiting system. Goodbye!");**

**break;**

**default:**

**System.*out*.println("Invalid choice. Try again.");**

**}**

**} while (choice != 6);**

**sc.close();**

**}**

**}**

**20**

**code for sql (database):**

CREATE TABLE books (

book\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

author VARCHAR(100),

publisher VARCHAR(100),

quantity INT

);

CREATE TABLE members (

member\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

phone VARCHAR(15)

);

**21**

CREATE TABLE transactions (

trans\_id INT AUTO\_INCREMENT PRIMARY KEY,

book\_id INT,

member\_id INT,

issue\_date DATE,

return\_date DATE,

returned BOOLEAN DEFAULT FALSE,

FOREIGN KEY (book\_id) REFERENCES books(book\_id),

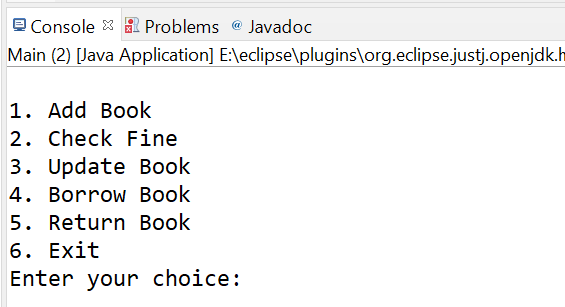
FOREIGN KEY (member\_id) REFERENCES members(member\_id)

);

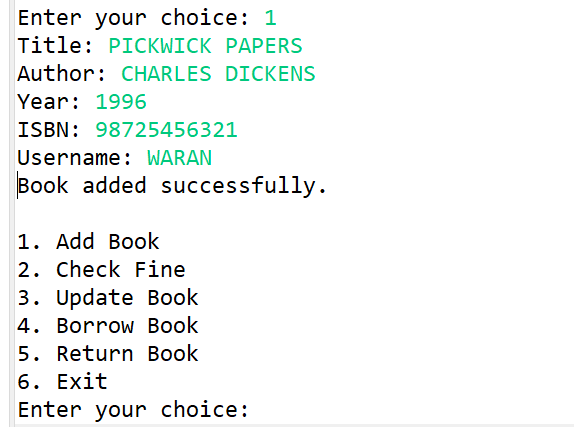
**22**

**Output:**

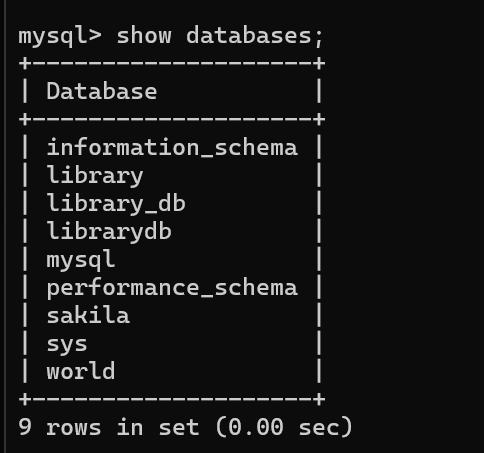
Java console output:



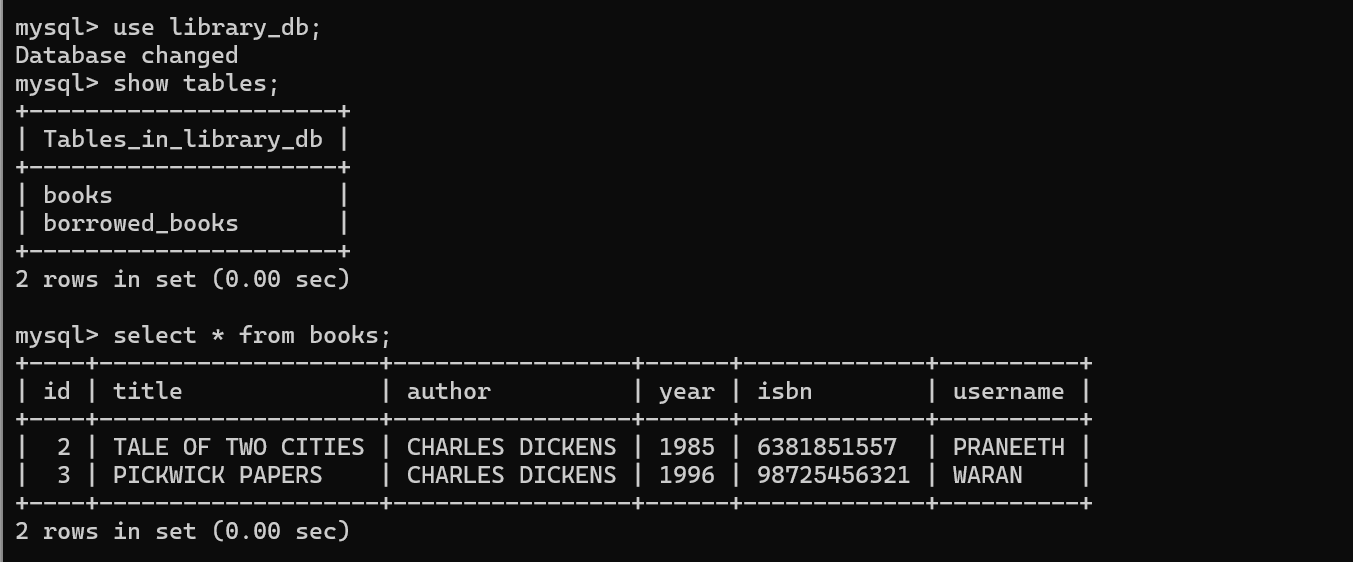
ADDING BOOK IN CONSOLE

****

**23**

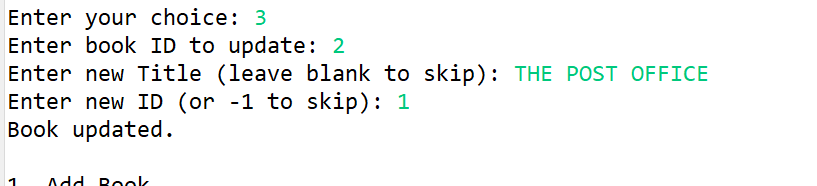
****

BOOKS STORED IN DATABASE

****

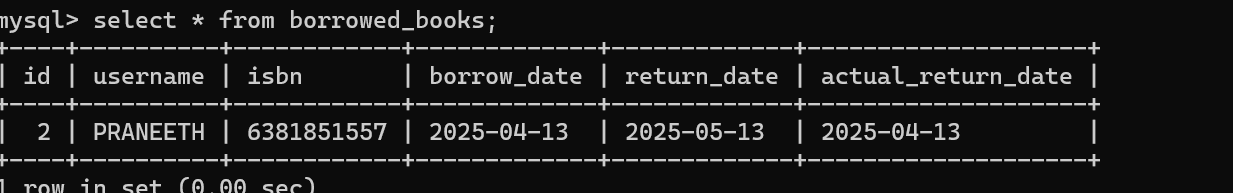
**24**

**UPDATE BOOKS**

****

****

**BORROWED BOOKS BEFORE UPDATION:**

****

RETURN BOOKS

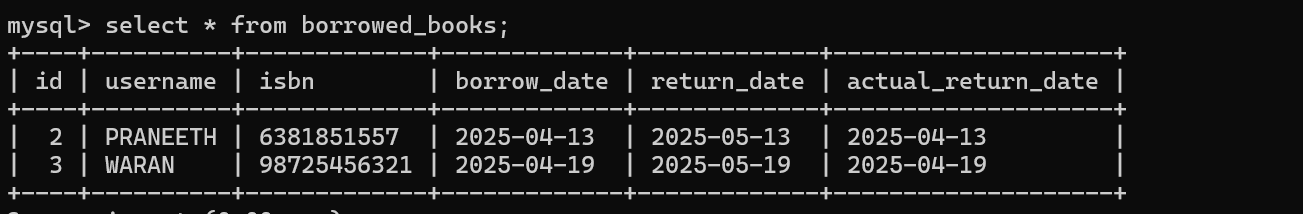
****

****

**25**

**IN THE ABOVE EXAMPLE I HAVE THE BOOK ON SAME DATE SO THE FINE AMOUNT IS 0.BY DEFAULT I HAVE GIVE THE DEFAULT RETURN DATE AS ONE MONTH FROM GETTING THE BOOKS IF NOT THE FINE AMOUNT IS RS.5 PER DAY.**

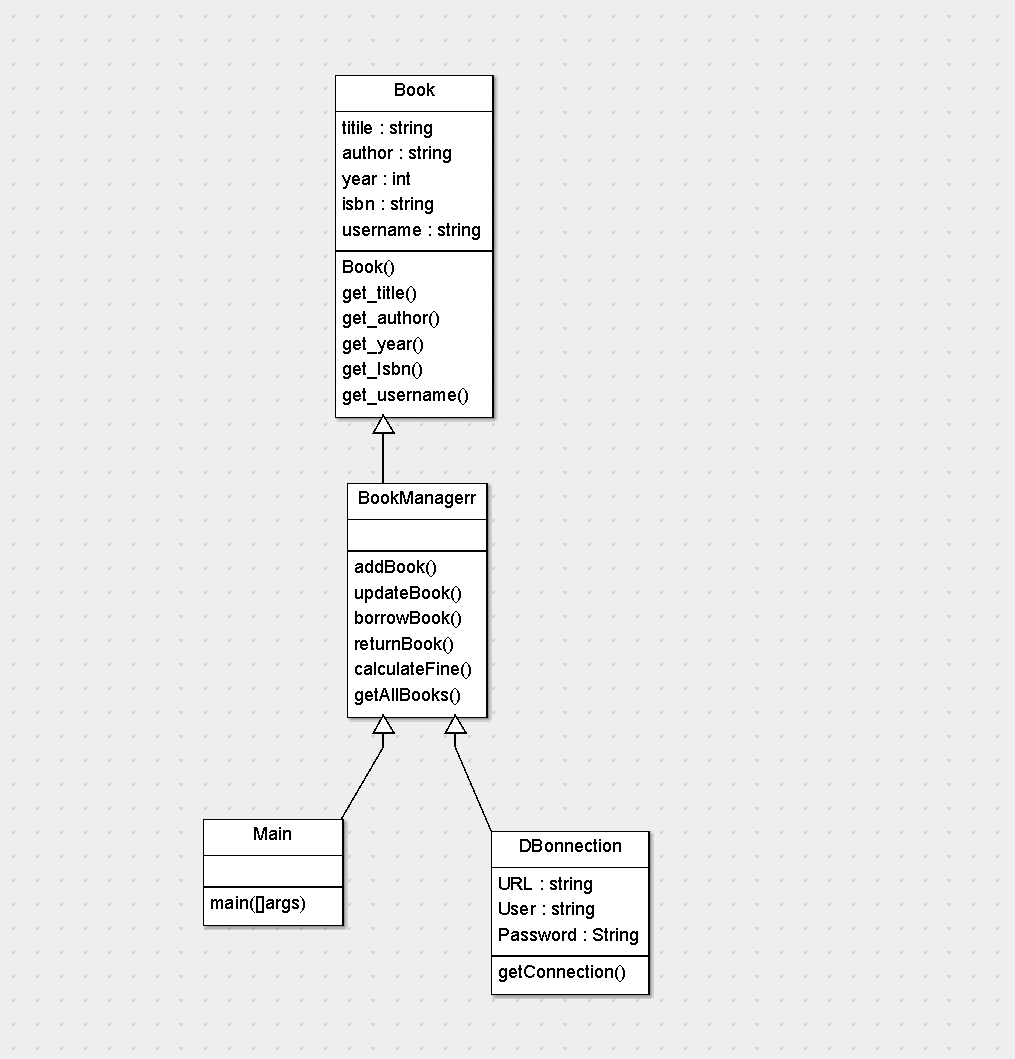
**THE RESULT IN DATABASE:**

****

**26**

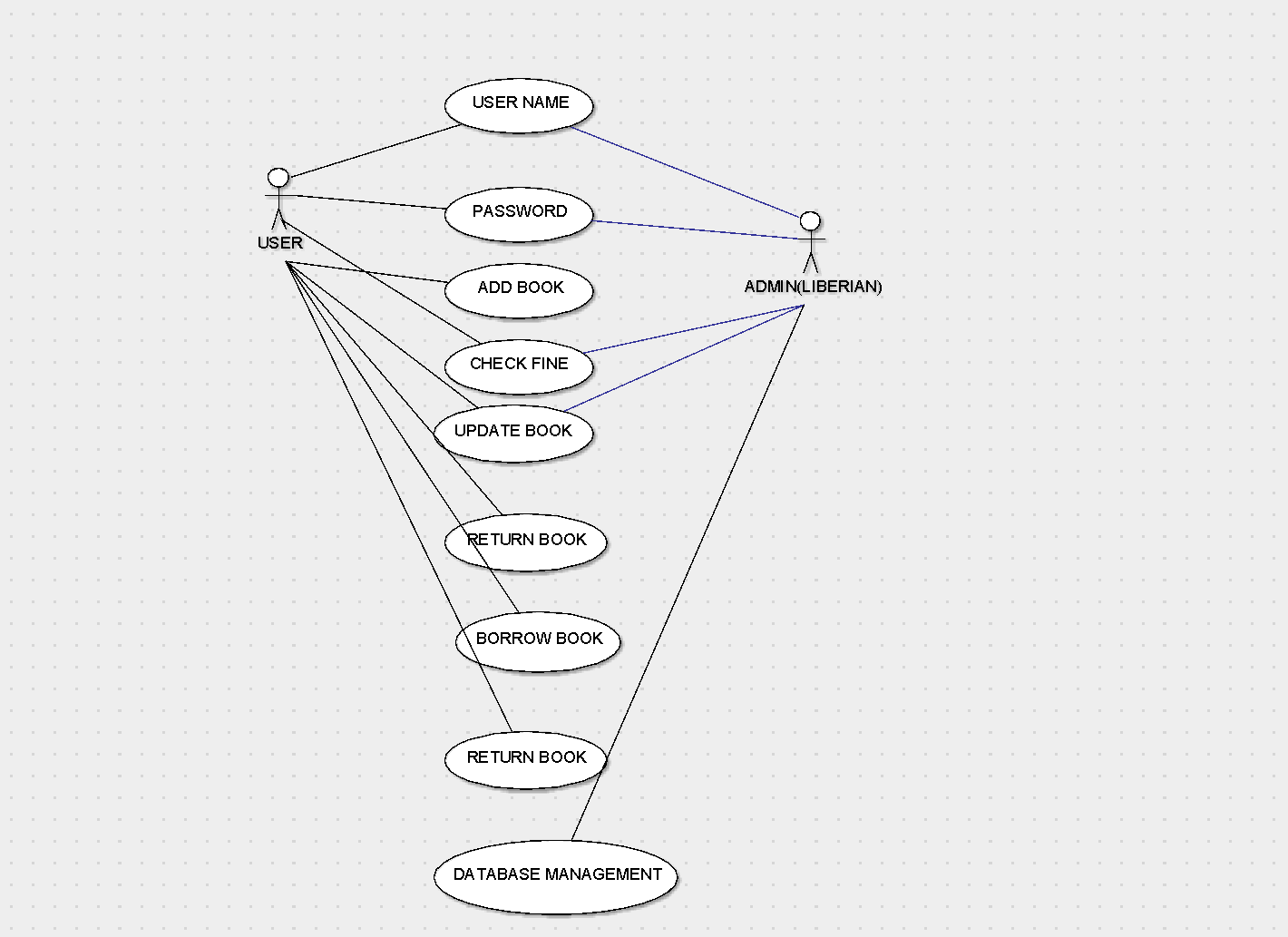
**UML DIAGRAM:**

**1.CLASS DIAGRAM:**

****

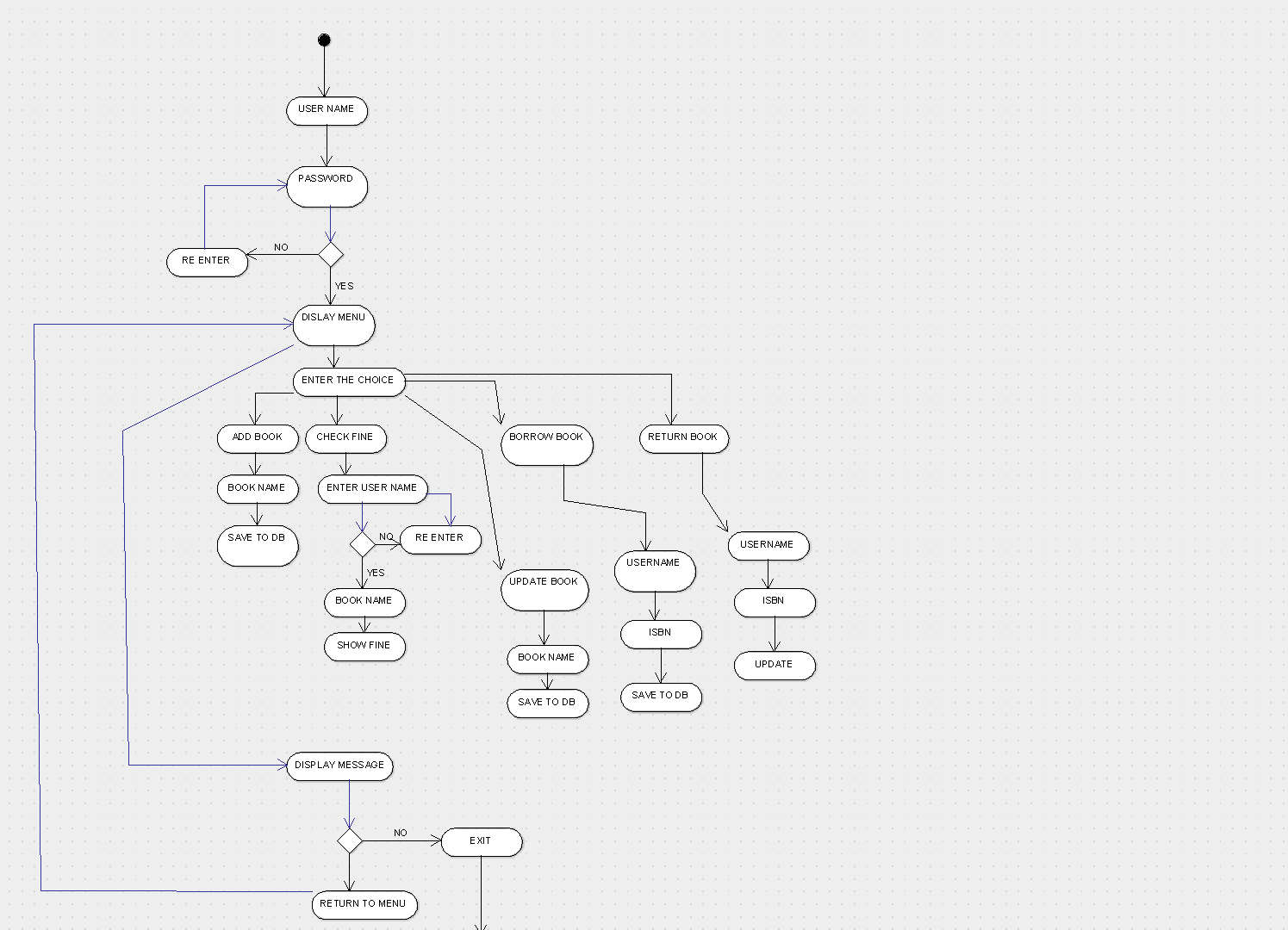
**27**

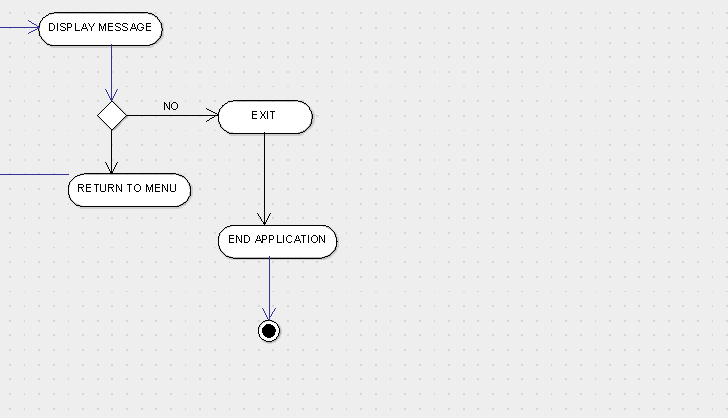
**2.USECASE DIAGAM**

****

**28**

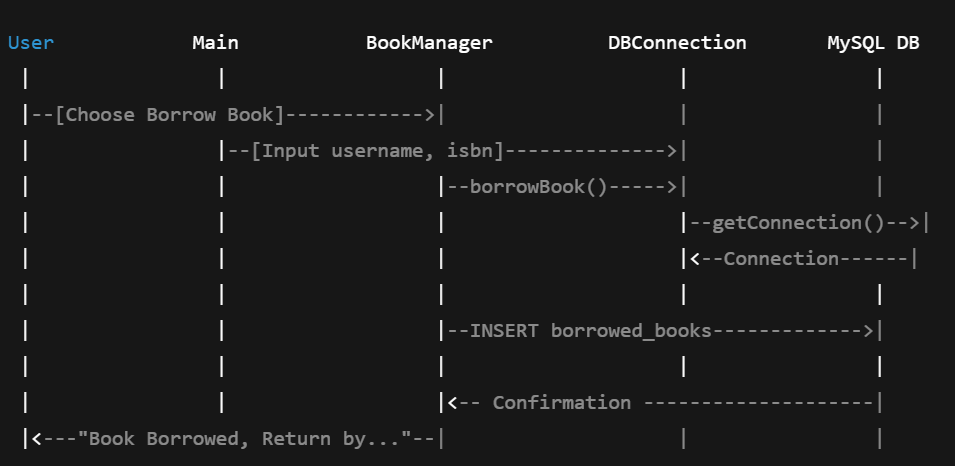
**3.ACTIVITY DIAGRAM**

****

****

**29**

**4.SEQUENCE DIAGRAM**

****

**30**

**APPLICATION OF THE PROJECT**

1. Educational Institutions
   * Used in schools, colleges, and universities to manage large volumes of books and track student borrowing records efficiently.
2. Public Libraries
   * Helps public libraries provide better service by streamlining the book lending and return process, reducing wait times, and keeping accurate records**.**
3. Corporate Libraries
   * Companies with internal libraries can use this system to manage technical books, journals, or training material issued to employees.
4. Private Book Clubs or Communities
   * Useful for managing shared book collections in reading groups or local clubs.

31

1. Digital Libraries
   * Forms the backend logic for managing digital or e-book lending systems when extended with a user interface and online access.
2. Research Centers & Laboratories
   * Keeps track of reference materials, manuals, and publications issued to researchers and staff**.**
3. Foundation for Advanced Systems
   * Acts as a base for building more complex systems like:
     + Web-based Library Portals
     + Mobile Library Apps
     + RFID-based Smart Libraries

**32**

**LIMITATINS OF THE PROJECT**

No Reporting or Analytics

* **There are no reports or dashboards to view borrowing trends, user history, or book popularity.**

No Concurrency Handling

* **If multiple users were to access the system simultaneously (e.g., in a networked setup), data inconsistencies may occur since there’s no concurrency control.**

Hardcoded DB Connection

* **Database credentials are hardcoded, which is not secure or scalable in a real-world deployment.**

Fixed Borrow Period & Fine

* **The return period and fine are hardcoded (1 month, $5/day), with no flexibility for different rules per book/user.**

No Mobile or Remote Access

* The system is not accessible over the internet or via mobile devices.

**33**

**Bibliography**

**Books & Study Material**

* ***Head First Java* by Kathy Sierra & Bert Bates**
* ***Java: The Complete Reference* by Herbert Schildt**
* ***Database System Concepts* by Abraham Silberschatz, Henry Korth, and S. Sudarshan**

**Websites & Online Resources**

* [**https://docs.oracle.com/javase/**](https://docs.oracle.com/javase/) **– Official Java Documentation**
* [**https://dev.mysql.com/doc/**](https://dev.mysql.com/doc/) **– MySQL Documentation**
* [**https://www.geeksforgeeks.org/**](https://www.geeksforgeeks.org/) **– Java, JDBC, and SQL tutorials**
* [**https://stackoverflow.com/**](https://stackoverflow.com/) **– Programming Q&A and problem solving**
* **https://www.w3schools.com/sql/ – SQL basics and examples**

**Software & Tools Used**

* **MySQL Server**
* **MySQL Workbench**
* **Java JDK (Java Development Kit)**
* **Eclipse IDE**

**34**

**GITHUB LINK:**

**35**