LIBRARY MANAGEMENT SYSTEM

Python & SQL

Submitted By :

Rineesh M S
Muhammed Nihal
Muhammed Sinan
Athulya

LIBRARY MANAGEMENT SYSTEM

1. INTRODUCTION

1.1 Purpose

The Library Management System (LMS) is designed to manage books, members, and borrowing/returning operations. It allows librarians to add, update, and remove books while enabling members to search for books, borrow, and return them efficiently.

1.2 Features

Librarian Functions:

- - Register/Login as a librarian
- - Add, update, and delete books
- - View book list

Member Functions:

- - Register/Login as a member
- - View book list
- - Borrow books

1.3 Requirements

Programming Language: Python

Database: MySQL

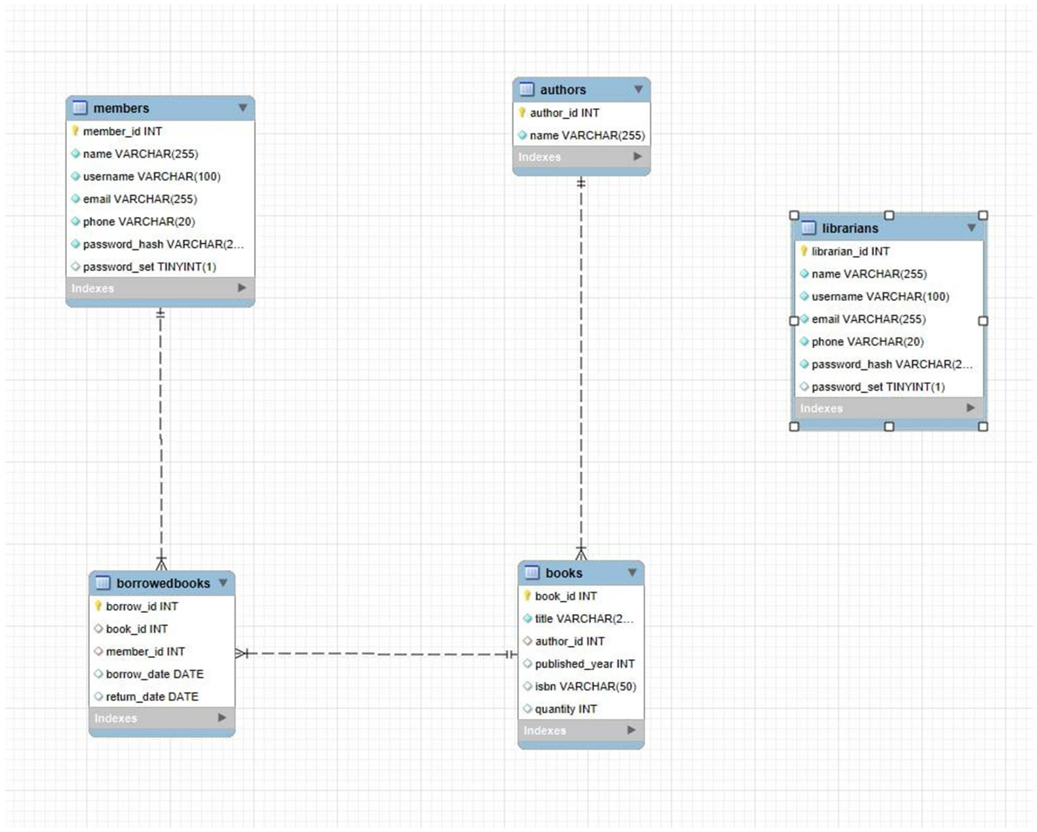
Libraries Used:

- - mysql.connector (Database connection)
- - bcrypt (Password hashing)
- - getpass (Secure password input)

2. DATABASE DESIGN

2.1 Tables

- - Librarians Table
- - Members Table
- - Authors Table
- - Books Table
- - BorrowedBooks Table



3. SYSTEM FUNCTIONALITY

3.1 User Registration & Authentication

1. Register New User (Librarian/Member)

   - User enters name, username, email, phone number, and password.

   - Password is hashed using bcrypt before storing.

   - Data is stored in either the Librarians or Members table.

2. User Login

   - User enters username and password.

   - The system fetches the hashed password from the database.

   - Password is verified using bcrypt.

   - On successful login, the appropriate menu (Librarian/Member) is displayed.

5. SECURITY FEATURES

- - Password Hashing: Uses bcrypt to store passwords securely.
- - Authentication System: Ensures only authorized users can access librarian or member functionalities.

- - Input Validation: Prevents SQL injection and invalid inputs.

6. FUTURE ENHANCEMENTS

- - Implement book return functionality.
- - Add a fine system for late returns.
- - Include email notifications for due dates.
- - Enhance UI with a web-based interface.

7. CONCLUSION

The Library Management System provides a structured approach to managing library operations efficiently. With role-based access, book tracking, and borrowing functionalities, it ensures smooth library management.

Library Management System - Code Implementation

SQL Database Schema

```
CREATE DATABASE LibraryDB;
USE LibraryDB;

CREATE TABLE Authors (
  author_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE Books (
  book_id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(255) NOT NULL,
  author_id INT,
  published_year INT CHECK (published_year BETWEEN 1500 AND
YEAR(CURDATE())),
  isbn VARCHAR(50) UNIQUE NOT NULL,
  quantity INT DEFAULT 1 CHECK (quantity >= 0),
  FOREIGN KEY (author_id) REFERENCES Authors(author_id) ON DELETE CASCADE
);

CREATE TABLE Members (
  member_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  username VARCHAR(100) UNIQUE NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  phone VARCHAR(20) UNIQUE NOT NULL CHECK (phone REGEXP '^[0-9]{10,15}$'),
  password_hash VARCHAR(255) NOT NULL,
  password_set BOOLEAN DEFAULT FALSE
);

CREATE TABLE Librarians (
  librarian_id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  username VARCHAR(100) UNIQUE NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  phone VARCHAR(20) UNIQUE NOT NULL CHECK (phone REGEXP '^[0-9]{10,15}$'),
  password_hash VARCHAR(255) NOT NULL,
  password_set BOOLEAN DEFAULT FALSE
);

CREATE TABLE BorrowedBooks (
  borrow_id INT PRIMARY KEY AUTO_INCREMENT,
  book_id INT,
  member_id INT,
  borrow_date DATE DEFAULT (CURDATE()),
  return_date DATE NULL,
  status ENUM('borrowed', 'returned') DEFAULT 'borrowed',
  FOREIGN KEY (book_id) REFERENCES Books(book_id) ON DELETE CASCADE,
  FOREIGN KEY (member_id) REFERENCES Members(member_id) ON DELETE
CASCADE
);
```

Python Code for Library Management System

```
import mysql.connector
import bcrypt
from getpass import getpass

# Database Connection
def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",  # Change if needed
        password="root123",  # Change if needed
        database="LibraryDB"
    )

# User Registration
def register_user(user_type):
    conn = connect_db()
    cursor = conn.cursor()

    print(f"\nRegister as {user_type.capitalize()}")

    name = input("Enter Full Name: ")
    username = input("Enter Username: ")
    email = input("Enter Email: ")
    phone = input("Enter Phone: ")

    while True:
        password = getpass("Enter Password: ")
        confirm_password = getpass("Confirm Password: ")
        if password == confirm_password:
            break
        print("\nPasswords do not match. Try again.")

    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    table = "Librarians" if user_type == "librarian" else "Members"
    cursor.execute(f"""
        INSERT INTO {table} (name, username, email, phone, password_hash,
password_set)
        VALUES (%s, %s, %s, %s, %s, TRUE)
    """, (name, username, email, phone, hashed_password.decode('utf-8')))

    conn.commit()
    print(f"\n{user_type.capitalize()} Registered Successfully!")

    cursor.close()
    conn.close()

# Login System
def login():
    conn = connect_db()
    cursor = conn.cursor()

    while True:
        print("\nLibrary Management System")
        print("1. Login as Librarian")
        print("2. Login as Member")
        print("3. Register New User")
        print("4. Exit")

        choice = input("\nEnter choice (1/2/3/4): ")

        if choice == "3":
            user_type = input("\nRegister as Librarian or Member? (librarian/member):
").lower()
            if user_type in ["librarian", "member"]:
                register_user(user_type)
            else:
                print("\nInvalid choice. Please enter 'librarian' or 'member'.")
            continue

        if choice == "4":
            print("\nExiting... Goodbye!")
            break

        if choice not in ["1", "2", "3", "4"]:
            print("\nInvalid choice. Please enter a number from 1 to 4.")
            continue

        username = input("\nEnter Username: ")
        password = getpass("Enter Password: ")

        table = "Librarians" if choice == "1" else "Members"
        id_column = "librarian_id" if choice == "1" else "member_id"
        cursor.execute(f"SELECT {id_column}, password_hash FROM {table} WHERE
username=%s", (username,))
        user = cursor.fetchone()

        if user and bcrypt.checkpw(password.encode('utf-8'), user[1].encode('utf-8')):
            print("\nLogin Successful")
            if choice == "1":
                librarian_menu()
            else:
                member_menu(user[0])
        else:
            print("\nInvalid Credentials. Please try again.")
    cursor.close()
    conn.close()

# Librarian Menu
def librarian_menu():
    while True:
        print("\nLibrarian Menu:")
        print("1. Add Book")
        print("2. Update Book")
        print("3. Delete Book")
        print("4. View Books")
        print("5. Logout")

        choice = input("\nEnter choice: ")

        if choice == "1":
            add_book()
        elif choice == "2":
            update_book()
        elif choice == "3":
            delete_book()
        elif choice == "4":
            view_books()
        elif choice == "5":
            print("\nLogging out...\n")
            break
        else:
            print("\nInvalid choice. Please enter a number from 1 to 5.")

# Member Menu
def member_menu(member_id):
    while True:
        print("\nMember Menu:")
        print("1. View Books")
        print("2. Borrow Book")
        print("3. View Borrowed Books")
        print("4. Return Book")  # New option for returning a book
        print("5. Logout")

        choice = input("\nEnter choice: ")

        if choice == "1":
            view_books()
        elif choice == "2":
            borrow_book(member_id)
        elif choice == "3":
            view_borrowed_books(member_id)
        elif choice == "4":
            return_book(member_id)  # Call the function to return a book
        elif choice == "5":
            print("\nLogging out...\n")
            break
        else:
            print("\nInvalid choice. Please enter a number from 1 to 5.")

# Add Book
def add_book():
    conn = connect_db()
    cursor = conn.cursor()

    title = input("\nEnter Book Title: ")
    author = input("Enter Author Name: ")
    year = input("Enter Published Year: ")
    quantity = input("Enter Quantity: ")

    cursor.execute("SELECT author_id FROM Authors WHERE name=%s", (author,))
    author_data = cursor.fetchone()

    if not author_data:
        cursor.execute("INSERT INTO Authors (name) VALUES (%s)", (author,))
        conn.commit()
        cursor.execute("SELECT LAST_INSERT_ID()")
        author_id = cursor.fetchone()[0]
    else:
        author_id = author_data[0]

    cursor.execute("INSERT INTO Books (title, author_id, published_year, quantity)
VALUES (%s, %s, %s, %s)",
                   (title, author_id, year, quantity))
    conn.commit()
    print("\nBook Added Successfully")
    cursor.close()
    conn.close()

# Update Book
def update_book():
    conn = connect_db()
    cursor = conn.cursor()

    book_id = input("\nEnter Book ID to Update: ")
    title = input("Enter New Title: ")
    year = input("Enter New Published Year: ")
    quantity = input("Enter New Quantity: ")

    cursor.execute("UPDATE Books SET title=%s, published_year=%s, quantity=%s
WHERE book_id=%s",
                   (title, year, quantity, book_id))
    conn.commit()

    print("\nBook Updated Successfully")
    cursor.close()
    conn.close()

# Delete Book
def delete_book():
    conn = connect_db()
    cursor = conn.cursor()

    book_id = input("\nEnter Book ID to Delete: ")
    cursor.execute("DELETE FROM Books WHERE book_id=%s", (book_id,))
    conn.commit()

    print("\nBook Deleted Successfully")
    cursor.close()
    conn.close()

# View Books
def view_books():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT b.book_id, b.title, a.name, b.published_year, b.quantity FROM
Books b JOIN Authors a ON b.author_id = a.author_id")
    books = cursor.fetchall()

    print("\nAvailable Books:\n")
    print("ID  | Title | Author | Year | Quantity")
    print("-" * 50)

    for book in books:
        print(f"{book[0]}  | {book[1]} | {book[2]} | {book[3]} | {book[4]}")

    cursor.close()
    conn.close()

# Borrow Book
def borrow_book(member_id):
    conn = connect_db()
    cursor = conn.cursor()

    while True:
        book_id = input("\nEnter Book ID to Borrow (or type 'back' to return): ")

        if book_id.lower() == "back":
            return

        cursor.execute("SELECT quantity FROM Books WHERE book_id=%s", (book_id,))
        book_data = cursor.fetchone()

        if book_data and book_data[0] > 0:
            cursor.execute("UPDATE Books SET quantity = quantity - 1 WHERE book_id=%s",
(book_id,))
            cursor.execute("INSERT INTO BorrowedBooks (book_id, member_id) VALUES
(%s, %s)", (book_id, member_id))
            conn.commit()
            print("\nBook Borrowed Successfully")
            break
        else:
            print("\nBook Not Available or Invalid Book ID. Try again.")

# View Borrowed Books for a Member
def view_borrowed_books(member_id):
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT b.book_id, b.title, a.name AS author, bb.borrowed_date, bb.return_date
        FROM BorrowedBooks bb
        JOIN Books b ON bb.book_id = b.book_id
        JOIN Authors a ON b.author_id = a.author_id
        WHERE bb.member_id = %s
    """, (member_id,))

    borrowed_books = cursor.fetchall()
    if borrowed_books:
        print("\nBorrowed Books:")
        print("ID  | Title | Author | Borrowed Date | Return Date")
        print("-" * 70)
        for book in borrowed_books:
            print(f"{book[0]}  | {book[1]} | {book[2]} | {book[3]} | {book[4] if book[4] else
'Not Returned'}")
    else:
        print("\nNo borrowed books found.")

    cursor.close()
    conn.close()

# Return Book
def return_book(member_id):
    conn = connect_db()
    cursor = conn.cursor()

    # Get list of borrowed books by the member
    cursor.execute("""
        SELECT bb.borrowed_id, b.book_id, b.title
        FROM BorrowedBooks bb
        JOIN Books b ON bb.book_id = b.book_id
        WHERE bb.member_id = %s AND bb.return_date IS NULL
    """, (member_id,))

    borrowed_books = cursor.fetchall()

    if borrowed_books:
        print("\nBorrowed Books:")
        print("ID  | Book Title")
        print("-" * 30)
        for book in borrowed_books:
            print(f"{book[0]}  | {book[2]}")

        # Let the member choose a book to return
        borrowed_id = input("\nEnter the ID of the book you want to return (or type 'back'
to go back): ")

        if borrowed_id.lower() == 'back':
            return

        cursor.execute("SELECT * FROM BorrowedBooks WHERE borrowed_id=%s AND
member_id=%s AND return_date IS NULL", (borrowed_id, member_id))
        book_to_return = cursor.fetchone()
        if book_to_return:
            # Update return_date to current date
            cursor.execute("UPDATE BorrowedBooks SET return_date = CURDATE() WHERE
borrowed_id = %s", (borrowed_id,))

            # Update the book's quantity
            cursor.execute("UPDATE Books SET quantity = quantity + 1 WHERE book_id =
%s", (book_to_return[1],))

            conn.commit()
            print("\nBook Returned Successfully.")
        else:
            print("\nInvalid selection. Please try again.")
    else:
        print("\nNo borrowed books to return.")

    cursor.close()
    conn.close()

if __name__ == "__main__":
    login()
```