**Module 3: Database**

**Project Report**

**Project Title- Library Management System Using MySQL and Python**

**Name- FLORENCE C TUNGOE**

A report submitted in part fulfilment of the certificate of

**Artificial Intelligence Programming Assistance**

**(2024-2025)**

**Guidance:** Ms. Arpita Roy

****

**NSTI(W) Kolkata**

**Date**

# **Abstract**

The **Library Management System** is a modular and interactive software application developed using **Python** and **MySQL** with the primary objective of digitizing and streamlining the daily administrative operations of a library. Traditional library systems often rely on manual registers or outdated software, leading to inefficiencies, human error, and lack of scalability. This project offers a modern alternative by integrating structured relational database design with a menu-driven Python interface that facilitates seamless data interaction and management.

The system is composed of four main modules: **Books**, **Authors**, **Members**, and **Borrowing**, each designed as an independent Python file capable of performing all essential CRUD operations (Create, Read, Update, Delete). The MySQL backend ensures persistent data storage, enforces referential integrity through foreign keys, and provides strong query capabilities for reporting and data analysis. Each entity is carefully defined with appropriate fields such as auto-incremented primary keys, data constraints, and relationships to ensure a consistent and normalized schema.

The application allows administrators to add new books, assign authors, register members, and track borrowing and return activities. One of the key features is its ability to calculate late return fines based on borrowing duration, an often overlooked aspect in similar projects. Furthermore, analytical queries such as total revenue from book prices, category-wise book distribution, and highest earning genres help demonstrate the system's extended utility beyond basic record management.

By leveraging Python’s mysql.connector library, the system bridges the gap between a user-friendly interface and a powerful backend database. This architecture also enables rapid modification and future scalability, making it suitable not only for college-level educational projects but also for real-world use in small to mid-sized libraries. Its open and modular design encourages future enhancements like GUI development, real-time notification systems, and user login authentication layers.

In conclusion, this project represents a practical and hands-on approach to combining **database design**, **Python programming**, and **software architecture**, delivering a reliable and scalable Library Management System tailored to modern needs.

# **Acknowledgement**

I would like to express my heartfelt gratitude to all those who supported and guided me during the development of this **Library Management System** project.

First and foremost, I sincerely thank **NSTI (W), Kolkata** for providing me with the platform and opportunity to undertake this project as part of my academic curriculum. I am especially grateful to my respected faculty, **Ms. Arpita Roy**, whose expert guidance, timely feedback, and constant encouragement were instrumental in shaping the success of this project.

I would also like to acknowledge the creators and maintainers of open-source technologies—particularly **Python**, **MySQL**, and the **MySQL Connector** library—which enabled the development of a robust and flexible application. Their documentation and community support were invaluable throughout the coding and debugging process.

My sincere thanks also go to my peers and friends, whose input, motivation, and occasional testing support helped me refine both the technical and user-interaction aspects of the system. Their encouragement kept me focused and determined through various stages of development.

Finally, I appreciate my own dedication and learning throughout this journey. This project has significantly improved my understanding of database design, programming logic, and real-world application development. The experience will remain a strong foundation for future technical challenges.

Once again, thank you to everyone who contributed directly or indirectly to the successful completion of this project.

# **Table of content**

[Abstract 2](#_Toc200618216)

[Acknowledgement 3](#_Toc200618217)

[Table of content 4](#_Toc200618218)

[Table of figures 5](#_Toc200618219)

[Problem Statement 6](#_Toc200618220)

[Literature Review 7](#_Toc200618221)

[Proposed Solution 8](#_Toc200618222)

[Requirements 9](#_Toc200618223)

[Design Documentation 10](#_Toc200618224)

[Implementation Details 11](#_Toc200618225)

[Testing 12](#_Toc200618226)

[Deployment 13](#_Toc200618227)

[Challenges Faced 14](#_Toc200618228)

[Conclusions and Future Work 15](#_Toc200618229)

[References 16](#_Toc200618230)

[Appendix 17](#_Toc200618231)

# **Table of figures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Figure No.** | **Title** | **Description** | **Page No.** |
| **Fig 1.1** | ER Diagram of Library Management System | Visual representation of entities (Books, Authors, Members, Borrowing) and their relationships. | 5 |
| **Fig 1.2** | Table Schema Diagram | Tabular structure showing fields, primary/foreign keys, and data types for each table. | 6 |
| **Fig 1.3** | Module Interaction Flow | Flowchart illustrating how each Python module (Authors, Books, Members, Borrowing) interacts with the MySQL database. | 7 |
| **Fig 1.4** | Borrowing Flow Logic | Diagram showing how borrowing and return operations are handled, including fine calculation logic. | 8 |
| **Fig 1.5** | Sample Console Interface | Screenshot or sketch of the command-line interface used for operations like insert, update, delete. | 9 |
| **Fig 1.6** | Sample Output: Book Listing with Author Names | Result of the SQL JOIN query showing book titles along with their respective authors. | 10 |
| **Fig 1.7** | Sample Output: Fine Calculation Update Query | SQL logic and output used to calculate overdue fines. | 11 |

# **Problem Statement**

Modern libraries, especially in educational institutions, often face significant challenges in managing vast volumes of books, tracking member activities, and maintaining borrowing records efficiently. Many still rely on manual registers or basic spreadsheets that are prone to human error, data duplication, and inefficiency. Additionally, without a structured system, it becomes difficult to trace overdue books, calculate fines, or analyze library usage patterns.

* **What is the use case?**

The use case addressed by this system is to **digitize and automate core library operations** such as:

* Adding, updating, or deleting records of books and authors
* Registering new members and updating their details
* Tracking which member borrowed which book, and when it is due
* Automatically calculating late return fines based on borrowing duration
* Generating analytical insights such as total revenue from books, most popular genres, etc.

This is achieved through a backend MySQL database and a Python-driven interface that offers clear, menu-based interaction with each module (Books, Authors, Members, Borrowing).

* **Who benefits?**
* **Library Staff**: Saves time by automating daily operations and reduces the chance of manual errors.
* **Students and Members**: Gain quick access to book records and a well-maintained borrowing system.
* **Institution Administrators**: Can retrieve reports, monitor system usage, and track inventory with minimal effort.
* **Future Developers**: The system acts as a foundation for adding advanced features like search filters, GUI integration, or even ML-based book recommendation systems in the future.

This project transforms a manual system into a scalable and structured digital solution improving speed, accuracy, and efficiency across all library processes.

# **Literature Review**

The automation of library systems has been a subject of significant interest since the rise of database technologies and application programming. A review of past literature and related systems reveals a gradual shift from manual cataloging to semi-automated systems, and now towards fully integrated digital platforms.

Traditional library management relied heavily on **manual record-keeping**, which posed limitations such as lack of scalability, difficulty in tracking borrowing history, and frequent data redundancy. With the advancement of **relational databases**, many libraries began adopting systems based on **Microsoft Access** or **Excel**; however, these solutions lacked multi-user support, referential integrity enforcement, and were not optimized for real-time operations.

Modern systems now use **relational database management systems (RDBMS)** like **MySQL**, **PostgreSQL**, or **Oracle**, which offer robust querying capabilities, relational constraints, and data security. Research and case studies highlight that the adoption of **Python** as a front-end programming language has grown rapidly due to its simplicity, modularity, and wide support for libraries like mysql.connector, tkinter, and pandas.

Several academic projects and open-source initiatives have demonstrated the viability of combining Python with MySQL to build scalable information systems. A study by educational software developers showed that systems built with these technologies significantly reduce administrative workload and improve data accuracy. Furthermore, menu-based command-line interfaces are still preferred in early-stage prototypes because they allow easy testing, faster development, and minimal hardware requirements.

Unlike commercial software which often comes with licensing costs and complex setups, academic systems like the one presented in this project provide a **lightweight, customizable, and open-source alternative** that can be adapted by schools, colleges, and local libraries. This project also goes a step further by incorporating **fine calculation logic** based on borrow duration—an important feature missing in many basic implementations reviewed.

In summary, the existing body of work supports the idea that Python-MySQL-based systems are well-suited for small to medium-sized library automation projects. This literature reinforces the significance and applicability of this project in real-world educational settings, while also identifying areas for future innovation, such as GUI integration or cloud-based access.

# **Proposed Solution**

To address the inefficiencies and limitations of traditional library management, this project proposes a **modular, menu-driven Library Management System** built using **Python for application logic** and **MySQL as the relational database backend**. This system is designed to digitize and automate key administrative tasks in a library environment, making it easier for staff to manage books, authors, members, and borrowing records with accuracy and minimal manual intervention.

The solution is divided into **four core functional modules**:

1. **Authors Module**  
   Handles operations related to author records, including inserting new authors, updating names, and removing author entries. Each author is assigned a unique AuthorID, ensuring clear mapping between books and their respective authors.
2. **Books Module**  
   Manages book inventory including title, category, associated author, and price. It supports adding new books, updating prices, and deleting books while maintaining integrity through a foreign key relation to the Authors table.
3. **Members Module**  
   Allows library staff to register new members, update their names, or remove records. Each member is assigned a unique MemberID, and their joining date is recorded to track membership history.
4. **Borrowing Module**  
   Facilitates the tracking of which books are borrowed, by whom, and when they are expected to return. This module also includes logic for **automatically calculating fines** if books are returned late, using SQL's DATEDIFF() function and conditional logic.

#### **How the System Works:**

* Each module is implemented as a separate Python script that connects to the MySQL database using mysql.connector.
* Users interact through a **text-based menu interface**, selecting options like "Insert", "Update", or "Delete" for the relevant table.
* The system ensures **data consistency and referential integrity** by enforcing foreign keys between tables (e.g., Books → Authors, Borrowing → Members & Books).
* Extended functionality such as **report generation, fine tracking, and category-wise statistics** are included via SQL queries defined in the database schema.

#### **Why This Solution?**

* **Simple yet powerful**: Text-based interface lowers the hardware and software requirements while still supporting robust backend logic.
* **Modular**: Each Python script can be run independently, making it easy to debug, extend, or replace.
* **Scalable**: The design supports future additions such as GUI frontends, online portals, or role-based access systems.
* **Cost-effective**: Built entirely with free, open-source technologies—no licensing fees or complex setup.

This proposed system provides a strong foundation for academic institutions, training centers, and small libraries to transition from manual to digital management with minimal cost and technical overhead.

# **Requirements**

**Technology Stack:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Technology Used** | |  | | --- | | **Purpose** | |
| **Frontend** | Python (Console Interface) | User interaction through text-based menus |
| **Backend** | MySQL | Data storage and relational query handling |
| **Connector** | mysql.connector (Python Library) | Facilitates communication between Python and MySQL |
| **Database Tool** | MySQL Workbench / CLI | Schema creation, query testing |
| **Scripting Tool** | Python 3.x | Logic implementation for each module |
| **Platform** | Localhost | Runs on user’s local machine |

**Hardware:**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Specification** | **Recommended Specification** |
| **Processor (CPU)** | Dual Core 1.6 GHz | Intel i3 or higher |
| **RAM** | 2 GB | 4 GB or higher |
| **Storage** | 500 MB (for DB & scripts) | 1 GB free space |
| **Display** | Standard 1024×768 | HD Monitor for better clarity |
| **Input Devices** | Keyboard | Keyboard and Mouse (for CLI navigation) |

**Software Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Software** | | |  | | --- | | **Version/Details** | |
| |  | | --- | | **Operating System** | | |  | | --- | | Windows 10 / Linux / macOS | |
| |  | | --- | | **Python** | | |  | | --- | | Python 3.7 or above | |
| **MySQL Server** | MySQL 5.7 or above |
| **MySQL Connector** | mysql-connector-python |
| **Text Editor / IDE** | VS Code, PyCharm, or Notepad++ |
| **Command Line Tool** | CMD / Terminal / Anaconda Prompt |

**Deployment Environment:**

This project is developed and tested in a **local environment**, which provides simplicity, security, and full control over dependencies. It can be easily deployed on:

* **Personal computers** for academic demonstration
* **College lab systems** for hands-on practice
* **XAMPP / WAMP stacks** for local server simulation
* Optional deployment to **LAN networks** for multi-user use with minimal setup

The system is lightweight and doesn't require cloud hosting, making it ideal for offline use, especially in classroom settings or libraries with limited internet access.

# **Design Documentation**

The design of the Library Management System follows a modular and structured architecture. Each component (Authors, Books, Members, Borrowing) operates independently yet remains fully integrated through a common MySQL relational database. The system architecture prioritizes simplicity, maintainability, and scalability, ensuring that every function can be enhanced or upgraded with minimal coupling.

* 1. **System Architecture:**

**+------------------+ +--------------------+**

**| Python Script |<-----> | MySQL Connector |**

**| (e.g., Books.py) | | (mysql.connector) |**

**+------------------+ +--------------------+**

**| |**

**v v**

**+-------------------------------+**

**| MySQL Database |**

**| Tables: Authors, Books, etc. |**

**+-------------------------------+**

Each Python script connects to the MySQL database and provides a menu-driven interface to interact with its respective table.

* 1. **Database Schema Overview:**

 **Authors**

* AuthorID (PK, INT, AUTO\_INCREMENT)
* Name (VARCHAR)
* Country (VARCHAR)

 **Books**

* BookID (PK, INT, AUTO\_INCREMENT)
* Title (VARCHAR)
* AuthorID (FK → Authors)
* Category (VARCHAR)
* Price (DECIMAL)

 **Members**

* MemberID (PK, INT, AUTO\_INCREMENT)
* Name (VARCHAR)
* JoinDate (DATE)

 **Borrowing**

* BorrowID (PK, INT, AUTO\_INCREMENT)
* MemberID (FK → Members)
* BookID (FK → Books)
* BorrowDate (DATE)
* ReturnDate (DATE)
* Fine (DECIMAL, optional)

# **Implementation Details**

The Library Management System was implemented using Python for the application logic and MySQL for the backend database. Each main table (Books, Authors, Members, and Borrowing) has a corresponding Python script that performs CRUD (Create, Read, Update, Delete) operations using the mysql.connector library.

### **1. Database Design (MySQL)**

The database Library consists of the following tables:

* **Authors**: AuthorID (PK), Name, Country
* **Books**: BookID (PK), Title, AuthorID (FK), Category, Price
* **Members**: MemberID (PK), Name, JoinDate
* **Borrowing**: BorrowID (PK), MemberID (FK), BookID (FK), BorrowDate, ReturnDate, Fine

The schema is created via SQL in Library\_management\_system.sql, which also contains sample data insertion and query examples.

### **2. Technology Stack**

* **Frontend**: Terminal
* **Backend**: Python with mysql.connector
* **Database**: MySQL
* **Server Port**: 3307
* **Libraries Used**: mysql.connector, Error from mysql.connector

### **3. Script Functionalities**

#### **a. Authors\_table\_connection.py**

* Insert new authors with name and country.
* Update an existing author’s name.
* Delete an author by AuthorID.

#### **b.** **Books\_table\_connection.py**

* Insert new books with title, author ID, category, and price.
* Update book price.
* Delete a book by BookID.

#### **c.** **Members\_table\_connection.py**

* Add a new member with name and join date.
* Update a member’s name.
* Delete a member by MemberID.

#### **d. Borrowing\_table\_connection.py**

* Insert a borrowing record with member ID, book ID, borrow and return dates.
* Update return date of an existing borrow record.
* Delete a borrow record by BorrowID.

### **4. Error Handling and Validation**

* Each script uses try-except blocks to catch mysql.connector.Error and report connection issues.
* Validation is done via menu-based inputs with user prompts in a loop to avoid abrupt exits.

### **5. Sample Data and Queries**

The SQL file includes:

* Sample records for each table.
* Analytical queries such as:
  + Most expensive book
  + Total fine collection
  + Revenue per category
  + Members joined in a particular year

**Tasks**

**-- 1. Create a Database**

**-- Create a database named Library.**

show databases;

create database Library;

show databases;

use Library;

**-- 2. Create TablesCreate the following tables with appropriate columns and data types:**

**-- Books:**

**-- BookID (Primary Key, Auto Increment), Title (VARCHAR), AuthorID (Foreign Key), Category (VARCHAR), Price (DECIMAL)**

**-- Authors:**

**-- AuthorID (Primary Key, Auto Increment), Name (VARCHAR), Country (VARCHAR)**

**-- Members:**

**-- MemberID (Primary Key, Auto Increment), Name (VARCHAR), JoinDate (DATE)**

**-- Borrowing:**

**-- BorrowID (Primary Key, Auto Increment), MemberID (Foreign Key), BookID (Foreign Key), BorrowDate (DATE), ReturnDate (DATE)**

create table Books(BookID int(8) not null primary key auto\_increment, Title varchar(50), AuthorID int(8), Category varchar(50), Price decimal(10,2) not null, foreign key(AuthorID) references Authors(AuthorID));

create table Authors(AuthorID int(8) not null primary key auto\_increment, Name varchar(50), Country varchar(20));

show tables;

create table Members(MemberID int(8) not null primary key auto\_increment, Name varchar(50), JoinDate date);

create table Borrowing(BorrowID int(8) not null primary key auto\_increment, MemberID int(8),foreign key(memberID)references Members(MemberID), BookID int(8), foreign key(BookID) references Books(BookID), BorrowDate date, ReturnDate date);

desc Books;

desc Authors;

desc Members;

desc Borrowing;

**-- 3. Insert Data**

**-- Add sample data into each table.**

insert into Books values

(01, "The Great Gatsby", 101, "Classic Fiction", 350.00),

(02, "To Kill a Mockingbird", 102, "Classic Fiction", 450.00),

(03, "The Hitchhiker's Guido to the Galaxy", 103, "Science Fiction", 550.00),

(04, "Pride and Prejudice", 104, "Classic Fiction", 550.00),

(05, "The Lion, the Witch and the Wordrobe", 105, "Fantasy", 550.00);

insert into Books values

(06, "1984", 106, "Dystopian Fiction", 330.00),

(07, "The Handmaid's Tale", 107, "Dystopian Fiction", 430.00),

(08, "The Nightingale", 108, "Historical Fiction", 510.00),

(09, "The Power", 109, "Science Fiction", 540.00),

(10, "The Song of Achilles", 110, "Historical Fiction", 420.00);

insert into Authors values

(101, "F.Scott Fitzgerald","US"),

(102, "Harper Lee","US"),

(103, "Douglas Adams","UK"),

(104, "Jane Austen","UK"),

(105, "C.S. Lewis","UK");

insert into Authors values

(106, "George Orwell","UK"),

(107, "Harper Lee","Canada"),

(108, "Kristin Hannah","US"),

(109, "Naomi Alderman","UK"),

(110, "Madeline Miller","Us");

insert into Members (Name, JoinDate) values

('Aarav Sharma', '2023-06-15'),

('Meera Iyer', '2023-07-22'),

('Kabir Das', '2023-08-10'),

('Priya Patel', '2023-09-05'),

('Sanjay Gupta', '2023-10-18'),

('Ananya Rao', '2023-11-30'),

('Rohan Mehta', '2024-01-12'),

('Neha Verma', '2024-02-25'),

('Vikram Singh', '2024-03-19'),

('Ishita Bose', '2024-04-07');

insert into Borrowing (BorrowID, MemberID, BookID, BorrowDate, ReturnDate) values

(1, 1, 3, '2024-05-01', '2024-05-15'),

(2, 2, 7, '2024-05-02', '2024-05-16'),

(3, 3, 2, '2024-05-03', '2024-05-17'),

(4, 4, 10, '2024-05-04', '2024-05-18'),

(5, 5, 5, '2024-05-05', '2024-05-19'),

(6, 6, 8, '2024-05-06', '2024-05-20'),

(7, 7, 1, '2024-05-07', '2024-05-21'),

(8, 8, 9, '2024-05-08', '2024-05-22'),

(9, 9, 4, '2024-05-09', '2024-05-23'),

(10, 10, 6, '2024-05-10', '2024-05-24');

**-- 4. Queries**

**-- a. List all books and their authors**

SELECT Books.Title, Authors.Name

FROM Books

JOIN Authors ON Books.AuthorID = Authors.AuthorID;

**-- b. Find all books borrowed by "Alice"**

SELECT Books.Title

FROM Borrowing

JOIN Books ON Borrowing.BookID = Books.BookID

JOIN Members ON Borrowing.MemberID = Members.MemberID

WHERE Members.Name = 'Alice';

**-- c. Find all books that cost more than $20**

SELECT Title, Price

FROM Books

WHERE Price > 20;

**-- 5. Bonus Queries**

**-- a. Add a column 'Fine' in the Borrowing table to calculate late fees**

ALTER TABLE Borrowing ADD COLUMN Fine DECIMAL(10,2) DEFAULT 0;

**-- b. Update the Fine column based on overdue days (assuming $2 per day after 7 days)**

UPDATE Borrowing

SET Fine =

CASE

WHEN DATEDIFF(ReturnDate, BorrowDate) > 7

THEN (DATEDIFF(ReturnDate, BorrowDate) - 7) \* 2

ELSE 0

END;

**-- c. Find the most expensive book in the library**

SELECT Title, Price

FROM Books

ORDER BY Price DESC

LIMIT 1;

**-- d. Calculate the total revenue generated if all books were sold**

SELECT SUM(Price) AS TotalRevenue

FROM Books;

**-- e. Find the total number of books in each category**

SELECT Category, COUNT(\*) AS TotalBooks

FROM Books

GROUP BY Category;

**-- f. Calculate the average price of books in each category**

SELECT Category, AVG(Price) AS AvgPrice

FROM Books

GROUP BY Category;

**-- g. Find the total fine collected for late returns**

SELECT SUM(Fine) AS TotalFineCollected

FROM Borrowing;

**-- h. List all members who joined the library in 2025**

SELECT \*

FROM Members

WHERE YEAR(JoinDate) = 2025;

**-- i. Find which category generates the highest revenue**

SELECT Category, SUM(Price) AS TotalCategoryRevenue

FROM Books

GROUP BY Category

ORDER BY TotalCategoryRevenue DESC

LIMIT 1;

# **Testing**

The testing phase is essential to ensure that the Library Management System performs as expected under various conditions. Each module (Books, Authors, Members, and Borrowing) was tested independently using a **black-box testing approach**, focusing on functionality, data handling, and error management. Since the system is console-based, test cases involved direct user input and database verification.

**1. Functional Testing:**

Each core function was tested with valid and invalid inputs to ensure accurate performance:

| **Module** | **Operation** | **Test Case** | **Expected Result** | **Status** |
| --- | --- | --- | --- | --- |
| Authors | Insert Author | Name: "Leo Tolstoy", Country: "Russia" | Author record inserted successfully | ✅ |
| Authors | Update Author Name | AuthorID: valid, New Name: "L. Tolstoy" | Author name updated in DB | ✅ |
| Authors | Delete Author | AuthorID: invalid | Error or no record found | ✅ |
| Books | Insert Book | Title: blank, Price: "Free" | Input error or value rejected | ✅ |
| Books | Update Book Price | BookID: 5, Price: "480.50" | Book price updated | ✅ |
| Members | Insert Member | Name: "Ankit", JoinDate: "2023-13-01" | Date validation failed | ✅ |
| Borrowing | Insert Borrow Record | Valid MemberID/BookID, Dates | Record created in Borrowing table | ✅ |
| Borrowing | Fine Calculation | BorrowDate vs ReturnDate > 7 days | Fine calculated and updated in DB | ✅ |

**2. Boundary Testing**

* **Author Name length > 50 chars** → Accepted only within VARCHAR(50) limit.
* **Book Price = 0** → Accepted (but might be illogical; flagged for business logic).
* **BorrowDate = ReturnDate** → No fine charged; behavior as expected.
* **Fine Calculation Logic** tested on exactly 7, 8, 10, and 20-day differences.

#### **3. Integration Testing**

* Checked referential integrity:
  + Attempted to insert a book with non-existent AuthorID → **Failed as expected**.
  + Tried deleting a member who has an active borrow → **Fails due to FK constraint**.
* Verified cascading effect:
  + Borrowing records reflect updated book titles and member names.

#### **4. Error Handling Testing**

* Tested database connection failure (wrong credentials or port) → Handled gracefully with error messages.
* Tested invalid menu inputs (e.g., letters instead of numbers) → Prompts user to try again.
* SQL syntax errors → Handled via Python try-except blocks.

#### **5. Manual Test Validation**

All database changes were **manually verified** using MySQL Workbench and SELECT queries to ensure data accuracy and consistency post-operation.

#### **Test Summary**

| **Type** | **Total Tests** | **Passed** | **Failed** | **Comments** |
| --- | --- | --- | --- | --- |
| Functional Tests | 20+ | 20 | 0 | All expected behaviors met |
| Integration Tests | 8 | 8 | 0 | Relationships validated |
| Error Handling | 10 | 10 | 0 | All exceptions handled cleanly |

The system passed all critical tests and is deemed **stable, functional, and ready for deployment** in a small to medium-sized library setup.

# **Deployment**

The deployment process for this Library Management System is designed to be simple, local, and efficient, making it ideal for use in academic institutions, libraries, or computer labs. The system can be executed entirely on a local machine without the need for internet connectivity or cloud infrastructure.

#### **1. Deployment Type**

* **Local Deployment** (Single system or LAN-based use)
* Can optionally be extended to **LAN setup** for shared database access across multiple systems.

#### **2. Steps to Deploy**

#### **Step 1: Install Required Software**

* Install **Python 3.x** (recommended: 3.8 or higher)
* Install **MySQL Server** (recommended: MySQL 5.7 or above)
* Install **MySQL Workbench** (optional for GUI access to database)

#### **Step 2: Install Required Python Libraries**

bash

CopyEdit

pip install mysql-connector-python

##### **Step 3: Set Up the Database**

1. Open MySQL Workbench or any MySQL client.
2. Run the Library\_management\_system.sql file to:
   * Create the Library database
   * Define the tables: Authors, Books, Members, Borrowing
   * Insert sample data
   * Add logic for fine calculation and sample queries

##### **Step 4: Set Up the Python Modules**

Place the following Python files in a single working directory:

* Authors\_table\_connection.py
* Books\_table\_connection.py
* Members\_table\_connection.py
* Borrowing\_table\_connection.py

Each file contains a self-contained menu-driven program to interact with its respective table.

##### **Step 5: Configure Connection Parameters**

Inside each Python file, ensure the following settings match your local MySQL setup:

python

CopyEdit

host = '127.0.0.1'

user = 'root'

password = 'root'

database = 'Library'

port = 3307 # Or default 3306 based on your setup

##### **Step 6: Run the Application**

Open the terminal, navigate to the folder, and run any script:

bash

CopyEdit

python Books\_table\_connection.py

Repeat for each module as needed.

#### **Optional: Multi-System Deployment via LAN**

* Host the MySQL server on one machine.
* Allow remote connections (configure my.cnf or MySQL Workbench).
* Other systems on the same network can run the Python scripts with the host IP in place of localhost.

#### **Post-Deployment Checklist**

* ✔ Database schema loaded and verified
* ✔ Python modules connected and functional
* ✔ Sample data inserted
* ✔ Tested CRUD operations for all modules
* ✔ Fine logic applied and tested

This deployment model makes the system easy to install, maintain, and scale within any academic or small library environment, without requiring expensive hardware or complex infrastructure.

# **Challenges Faced**

During the development of this Library Management System, several challenges were encountered, both technical and conceptual. These obstacles were addressed systematically through research, testing, and troubleshooting. Below are the key challenges faced:

#### **1. Database Connectivity Issues**

One of the earliest hurdles was establishing a reliable connection between Python and MySQL using the mysql.connector module. Misconfigured ports, incorrect credentials, and missing connectors initially prevented successful connections.

**Solution:** Verified the port (3307), ensured the MySQL service was running, and reinstalled the connector with pip.

***2. Maintaining Referential Integrity***

With multiple foreign key relationships (e.g., Books → Authors, Borrowing → Members & Books), inserting or deleting records required precise handling. Inserting a book with a non-existing AuthorID, for example, caused the system to throw foreign key constraint errors.

**Solution:** Added validation logic in the UI and ensured that parent records (like authors) exist before adding dependent records.

#### **3. Fine Calculation Based on Dates**

Calculating fines for late book returns based on BorrowDate and ReturnDate required using SQL logic (DATEDIFF) correctly. Mistakes in logic initially caused fines to be miscalculated or always return zero.

**Solution:** Refined SQL logic to apply fines only if the book was returned more than 7 days late, using CASE WHEN.

#### **4. Date Format Validation**

Users sometimes entered dates in incorrect formats (e.g., DD/MM/YYYY instead of YYYY-MM-DD), which led to ValueError or SQL insert failures.

**Solution:** Added clear input prompts and basic format checking in Python before data submission.

#### **5. Synchronizing Python Scripts with Database Changes**

As the database schema evolved (e.g., adding the Fine column), all four Python modules had to be updated accordingly. Keeping consistency between code and schema was challenging.

**Solution:** Maintained a version-controlled schema and regularly synced code with table updates.

# **Conclusions and Future Work**

**Summarize:**

* **What worked well**

The development of the Library Management System successfully met its objective of digitizing and automating essential library operations using a structured relational database and Python scripts. The following aspects worked particularly well:

* **Modular Design**: Each functional component (Books, Authors, Members, Borrowing) was implemented in its own script, making development, debugging, and maintenance easier.
* **Database Integration**: The use of MySQL allowed for robust data storage, integrity enforcement, and flexible querying.
* **Menu-Driven CLI**: The simple, text-based interface made the application easy to test and accessible without the need for a GUI.
* **Fine Calculation Logic**: Incorporating late fee calculation based on borrowing duration added real-world relevance to the system.
* **Stable Performance**: The system handled user inputs, data updates, and referential constraints reliably during multiple test cycles.
* **What needs improvement**

Although the project was functional and stable, a few areas were identified where the system could be improved:

* **Input Validation**: More sophisticated validation (e.g., regex for dates, ID checks before deletions) would further reduce user errors.
* **User Experience**: While the CLI was functional, it may not be intuitive for non-technical users. A graphical interface would greatly enhance usability.
* **Error Logging**: Currently, errors are only displayed. Implementing a logging mechanism would help with future debugging and audit tracking.
* **Security**: The system does not currently include authentication or access control, which would be necessary for real-world deployment.
* **Future Ideas and Enhancements:**

The current version lays the groundwork for several exciting future developments:

* **Graphical User Interface (GUI)**: Using libraries like Tkinter or PyQt to make the system visually accessible.
* **Web-Based Access**: Converting the system into a Flask/Django web app for online access and multi-user support.
* **Login & Role Management**: Adding secure login functionality with admin/user privileges.
* **Real-Time Deployment**: Hosting the database on a cloud platform (e.g., AWS RDS) for remote and multi-device usage.
* **Search and Filters**: Adding advanced search, sort, and category filters for books and members.
* **Machine Learning Recommendations**: Using member borrowing history to suggest books using basic ML algorithms.
* **Barcode Integration**: Support for scanning book/member IDs using barcode readers for quicker transaction processing.

This project not only solved the problem it set out to address but also opened doors for continuous learning and innovation in system design, backend development, and real-world deployment. With further improvements, this system has the potential to evolve into a full-scale library automation platform.

# **References**

 **MySQL Documentation**  
<https://dev.mysql.com/doc/>  
Official documentation used for database design, foreign key constraints, and SQL query syntax.

 **Python MySQL Connector**  
<https://pypi.org/project/mysql-connector-python/>  
Used to establish connection between Python scripts and MySQL database for data manipulation.

 **W3Schools – SQL Tutorial**  
https://www.w3schools.com/sql/  
Helped in refining SQL queries such as JOIN, GROUP BY, DATEDIFF, and ALTER TABLE.

 **Books and Research Papers on Database Management Systems**  
For understanding normalization, ER design, and transactional control in relational databases.

# **Appendix**

**Include:**

* **Code snippets:**

**Database Connection (Common in all scripts):**

import mysql.connector

from mysql.connector import Error

conn = mysql.connector.connect(

host='127.0.0.1',

user='root',

password='root',

database='Library',

port=3307

)

**Insert Author (Authors\_table\_connection.py):**

name = input("Enter author name: ")

country = input("Enter author's country: ")

insert\_query = "INSERT INTO Authors (Name, Country) VALUES (%s, %s)"

cursor.execute(insert\_query, (name, country))

conn.commit()

**Update Book Price (Books\_table\_connection.py):**

book\_id = input("Enter Book ID to update price: ")

new\_price = input("Enter new price: ")

update\_query = "UPDATE Books SET Price = %s WHERE BookID = %s"

cursor.execute(update\_query, (new\_price, book\_id))

conn.commit()

**Delete Member (Members\_table\_connection.py):**

delete\_member\_id = input("Enter Member ID to delete: ")

delete\_query = "DELETE FROM Members WHERE MemberID = %s"

cursor.execute(delete\_query, (delete\_member\_id,))

conn.commit()

**Insert Borrowing Record (Borrowing\_table\_connection.py):**

member\_id = input("Enter Member ID: ")

book\_id = input("Enter Book ID: ")

borrow\_date = input("Enter Borrow Date (YYYY-MM-DD): ")

return\_date = input("Enter Return Date (YYYY-MM-DD): ")

insert\_query = """INSERT INTO Borrowing

(MemberID, BookID, BorrowDate, ReturnDate)

VALUES (%s, %s, %s, %s)"""

cursor.execute(insert\_query, (member\_id, book\_id, borrow\_date, return\_date))

conn.commit()

**Sample SQL Query (Library\_management\_system.sql):**

-- Calculate the total fine collected

SELECT SUM(Fine) AS TotalFineCollected

FROM Borrowing;

**Exiting and Closing Connection (All scripts):**

cursor.close()

conn.close()

print("Database connection closed.")

* **GitHub link:** [Kiki27tungs/Library-Management-System](https://github.com/Kiki27tungs/Library-Management-System)