Library Management System - DBMS

Project Report

PROJECT TITLE: Library Management System

TEAM DETAILS:

NAMES	SRN
VINAYKUMAR	PES2UG23CS693
VINAY S KATNUR	PES2UG23CS692

2. Abstract

This Library Management System automates library operations including book inventory management, member registrations, issue/return tracking, and fine calculation. The backend uses MySQL with triggers, stored procedures, and functions to enforce business rules. The frontend is a Python Tkinter GUI with role-based access (Admin, Librarian, Viewer).

3. User Requirement Specification

Purpose:

The system aims to simplify library workflows—maintain catalogues, manage members and librarians, handle the issue/return process, calculate and record fines, and maintain audit logs for deletions. It reduces manual errors and speeds up day-to-day tasks.

Scope:

Covers book/publisher/author management, member and librarian management, issuing and returning books, fine management, reports (aggregate/join/nested queries), triggers to maintain stock, and logging deletions for audit purposes.

Detailed Description

Entities:

- **Book**: Attributes such as Book ID, Title, Author, Publisher, Year, Copies.
- **Author**: Contains Author ID, Name, Email.
- **Publisher**: Publisher ID, Name, Address.
- **Member**: Member ID, Name, Email, Phone, Join Date.
- **Librarian**: Librarian ID, Name, Email, Phone.
- **Issue**: Issue ID, Issue Date, Due Date, Return Date, Book ID, Member ID.
- **Fine**: Fine ID, Amount, Status, Issue ID.

Relationships:

As shown in the ER diagram:

- **Book – Author** (1:M)
- **Book – Publisher** (M:1)
- **Member – Issue** (1:M)
- **Book – Issue** (1:M)
- **Issue – Fine** (1:1)

Database automation:

From the SQL code:

- Trigger decreases book copies when issued.
- Trigger increases book copies when returned.
- Logs deleted books and members.

Frontend GUI:

Your Tkinter application has separate tabs for:

- Publishers
- Books
- Members
- Librarians
- Issue
- Fine
- Reports (Functions + Queries)

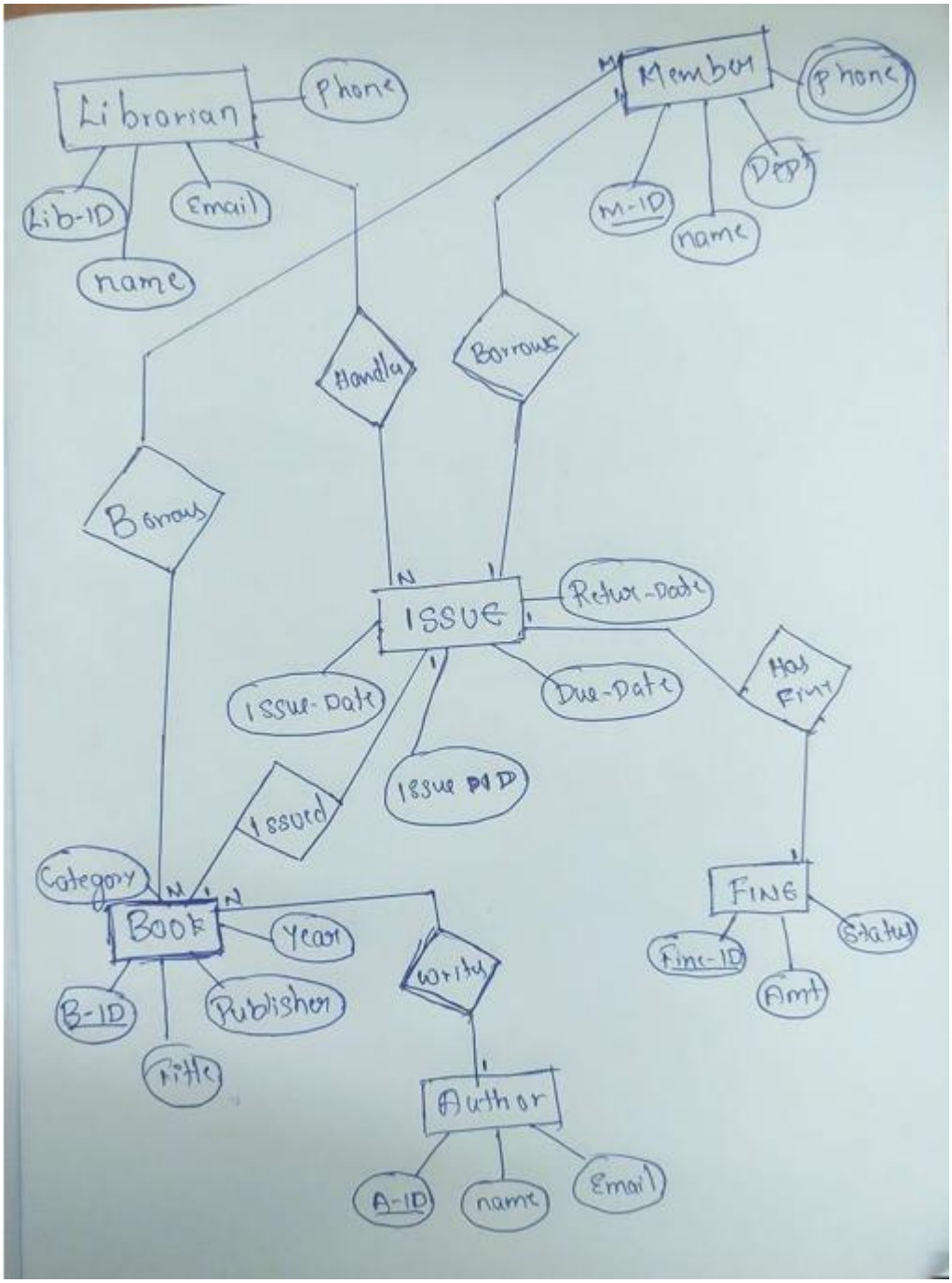
Functional Requirements (List)

FR1 - User Login System: Role-based login: Admin, Librarian, Viewer with access restrictions.
FR2 - Manage Publishers: Add/View/Delete publishers (name, address).
FR3 - Manage Books: Add/View/Delete books; automatic stock updates via triggers.
FR4 - Manage Members: Add/Update/Delete members; track join dates and contact info.
FR5 - Manage Librarians: Add/Delete librarians, maintain contact details.
FR6 - Issue Books: Record book issues; decrement available copies automatically.
FR7 - Return Books: Record returns; increment available copies automatically.
FR8 - Fine Management: Record fines and mark payments via stored procedures.
FR9 - Reports: Functions to get total books, total members and aggregate/join/nested query reports.
FR10 - Audit Logging: Log deleted books and members into audit tables for review.

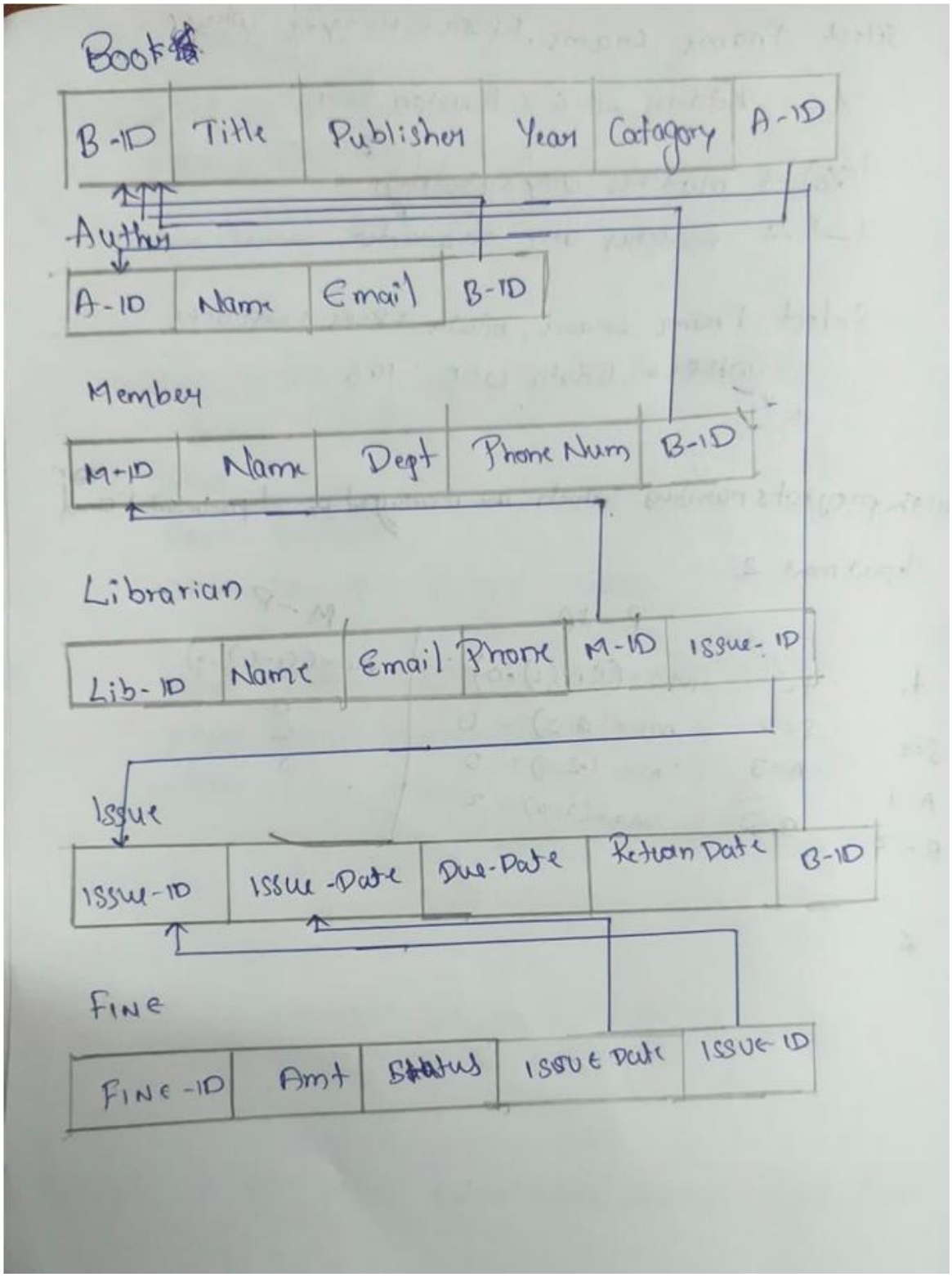
4. Software / Tools Used

- MySQL (database server)
- Python 3.x
- mysql-connector-python
- Tkinter (GUI)
- python-docx
- Git / GitHub

5. ER Diagram


Librarian
Phone
Lib-ID
Email
name
Member
Phone
M-ID
name
Dept
Handle
Borrows
Borrows
ISSUE
Return-Date
Issue-Date
Due-Date
Has Fine
Issued
Issue PID
Category
Book
Year
B-ID
Publisher
Title
Written
Author
A-ID
name
Email
FINE
Fine-ID
Amt
Status

6. Relational Schema



7. DDL Commands

```
--  PUBLISHERS TABLE
CREATE TABLE Publishers (
   publisher_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL UNIQUE,
   address VARCHAR(200)
);

INSERT INTO Publishers (name, address) VALUES
('Pearson Education', 'New Delhi, India'),
('O'Reilly Media', 'Sebastopol, USA'),
('McGraw Hill', 'Bangalore, India'),
('Oxford University Press', 'Oxford, UK'),
('Springer Nature', 'Berlin, Germany'),
('vinaykatnur', 'bengaluru');

--  BOOKS TABLE
CREATE TABLE Books (
   book_id INT PRIMARY KEY AUTO_INCREMENT,
   title VARCHAR(150) NOT NULL,
   author VARCHAR(100) NOT NULL,
   publisher_id INT,
   year_published YEAR CHECK (year_published >= 1800),
   available_copies INT DEFAULT 1 CHECK (available_copies >= 0),

   CONSTRAINT fk_publisher FOREIGN KEY (publisher_id)
      REFERENCES Publishers(publisher_id)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);

INSERT INTO Books (title, author, publisher_id, year_published, available_copies) VALUES
('Database System Concepts', 'Abraham Silberschatz', 3, 2019, 5),
('Learning Python', 'Mark Lutz', 2, 2021, 3),
('Operating System Concepts', 'Peter Galvin', 3, 2018, 4),
('Artificial Intelligence: A Modern Approach', 'Stuart Russell', 1, 2020, 2),
('Computer Networks', 'Andrew Tanenbaum', 4, 2017, 6);
```

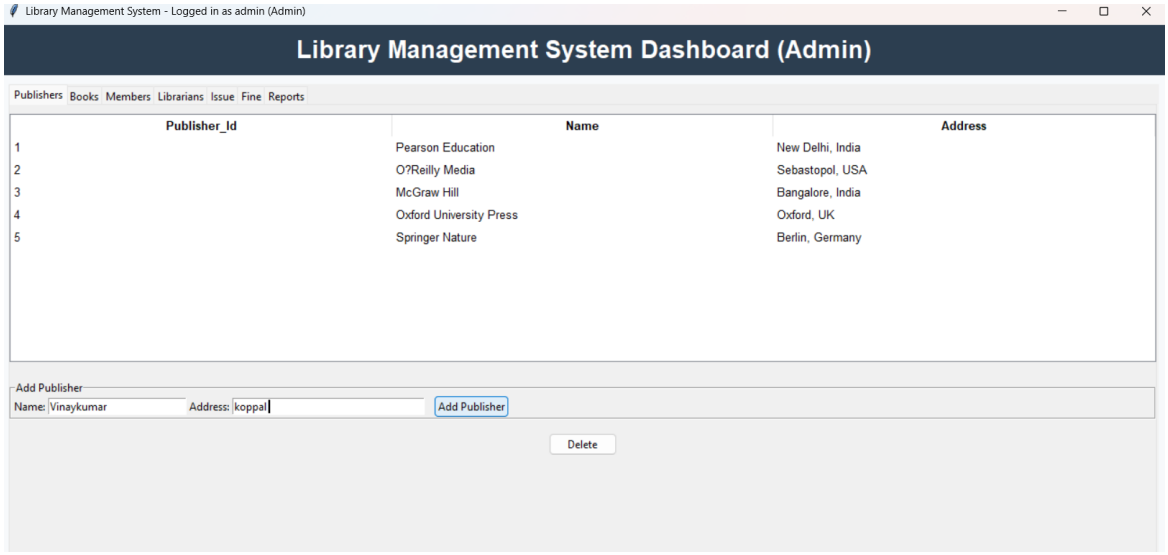
```
--  MEMBERS TABLE
CREATE TABLE Members (
   member_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE,
   phone VARCHAR(15),
   join_date DATE DEFAULT (CURRENT_DATE)
);

INSERT INTO Members (name, email, phone, join_date) VALUES
('Ravi Kumar', 'ravi.kumar@example.com', '9876543210', '2023-01-10'),
('Sneha Reddy', 'sneha.reddy@example.com', '9876501234', '2023-02-14'),
('Aman Verma', 'aman.verma@example.com', '9123456780', '2023-03-20'),
('Priya Nair', 'priya.nair@example.com', '9345678123', '2023-04-05'),
('Karthik Shetty', 'karthik.shetty@example.com', '9456789012', '2023-05-18'),
('vikram', 'vikram@gmail.com', '7897897890', '2025-11-11');

--  LIBRARIAN TABLE
CREATE TABLE Librarian (
   librarian_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE,
   phone VARCHAR(15)
);

INSERT INTO Librarian (name, email, phone) VALUES
('Anita Sharma', 'anita.sharma@library.com', '9871112222'),
('Rahul Menon', 'rahul.menon@library.com', '9873334444');

--  ISSUE TABLE
CREATE TABLE Issue (
   issue_id INT PRIMARY KEY AUTO_INCREMENT,
   book_id INT NOT NULL,
   member_id INT NOT NULL,
   librarian_id INT,
   issue_date DATE DEFAULT (CURRENT_DATE),
   return_date DATE,

   CONSTRAINT fk_book FOREIGN KEY (book_id)
      REFERENCES Books(book_id)
      ON DELETE CASCADE
```

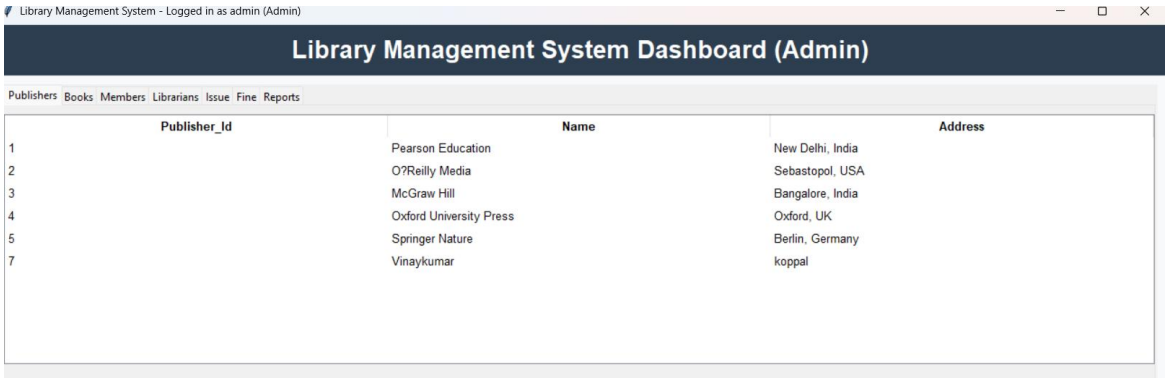
```
    ON UPDATE CASCADE,

  CONSTRAINT fk_member FOREIGN KEY (member_id)
    REFERENCES Members(member_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,

  CONSTRAINT fk_librarian FOREIGN KEY (librarian_id)
    REFERENCES Librarian(librarian_id)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);

-- Sample issue inserts
INSERT INTO Issue (book_id, member_id, librarian_id, issue_date) VALUES
(1, 1, 1, '2025-11-10'),
(2, 2, 2, '2025-11-09'),
(3, 3, 1, '2025-11-08');

-- Return update
UPDATE Issue
SET return_date = '2025-11-11'
WHERE issue_id = 1;

--  FINE TABLE
CREATE TABLE Fine (
  fine_id INT PRIMARY KEY AUTO_INCREMENT,
  issue_id INT NOT NULL,
  amount DECIMAL(8,2) CHECK (amount >= 0),
  status VARCHAR(20) DEFAULT 'Unpaid' CHECK (status IN ('Paid','Unpaid')),

  CONSTRAINT fk_issue FOREIGN KEY (issue_id)
    REFERENCES Issue(issue_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

INSERT INTO Fine (issue_id, amount, status) VALUES
(1, 100, 'Unpaid'),
(2, 50, 'Paid'),
(3, 150, 'Unpaid');
```

```
--  DELETED BOOK LOG TABLE
CREATE TABLE DeletedBooksLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   book_id INT,
   title VARCHAR(150),
   deleted_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- DELETED MEMBER LOG TABLE
CREATE TABLE DeletedMembersLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   member_id INT,
   name VARCHAR(100),
   deleted_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


--  USERS TABLE
CREATE TABLE IF NOT EXISTS Users (
   user_id INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) UNIQUE NOT NULL,
   password VARCHAR(100) NOT NULL,
   role ENUM('Admin', 'Librarian', 'Viewer') DEFAULT 'Librarian'
);

INSERT INTO Users (username, password, role) VALUES
('admin', 'admin123', 'Admin'),
('anita', 'lib123', 'Librarian'),
('viewer', 'view123', 'Viewer');
```

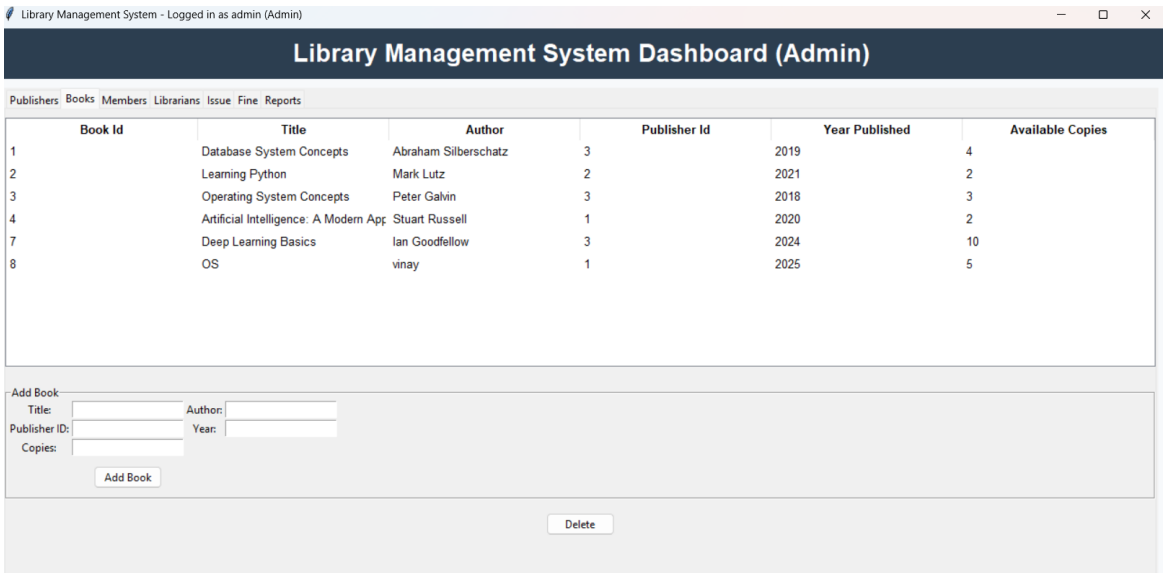
8. CRUD Operation Screenshots

CREATE



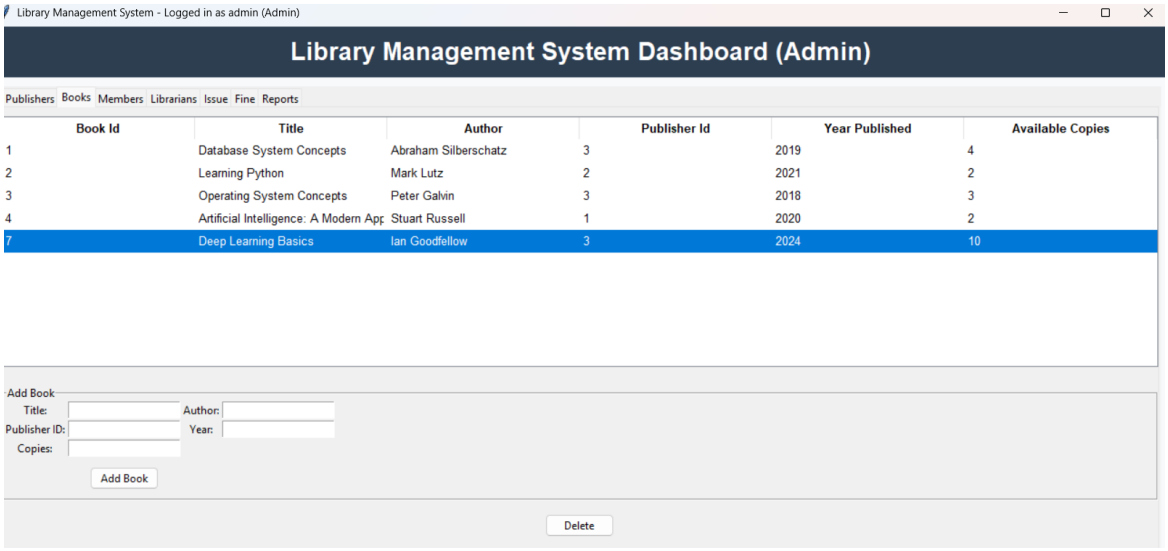


READ

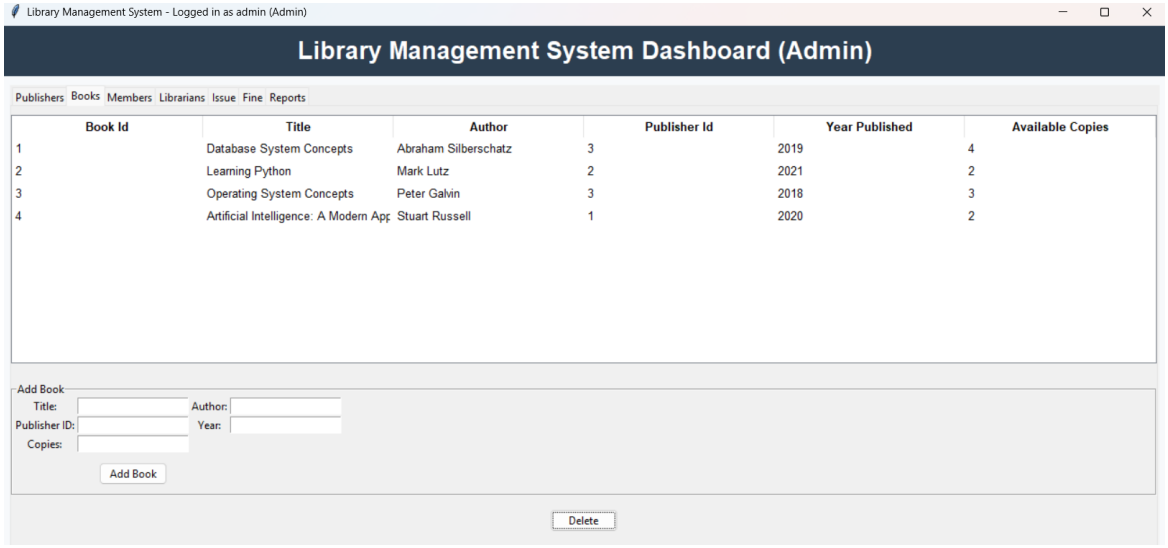


DELETE

Before:



After:



9. Application Functionalities & GUI Screenshots

Below is the complete list of system functionalities.

FR1: User Login System

Users must log in using username and password.

Different roles: Admin, Librarian, Viewer.

Admin has full access; Viewer has read-only access.

FR2: Manage Publishers

Add, view, and delete publishers.

Stores name and address.

FR3: Manage Books

Add books using stored procedure.

Delete books (Admin only).

View all books with publisher details.

Automatic decrease/increase copies via triggers.

FR4: Manage Members

Add new members using stored procedure.

Edit and update member details.

Delete members (Admin only).

FR5: Manage Librarians

Add and delete librarian records.

Stores contact information.

FR6: Issue Books

Records book issue with date, member, librarians.

Trigger trg_after_issue_insert reduces available copies.

FR7: Return Books

Update return date.

Trigger trg_after_return_update increases available copies.

FR8: Fine Management

Add fine entries.

Use stored procedure RecordFinePayment to mark fine as paid.

FR9: Reports Module

Show output of SQL functions (total books, total members).

Show aggregate, join and nested queries.

FR10: Audit Logging

Logs deleted books into DeletedBooksLog.

Logs deleted members into DeletedMembersLog.

FR11: Stored Procedures

AddNewBook

AddNewMember

UpdateMemberDetails

DeleteBook

DeleteMember

RecordFinePayment

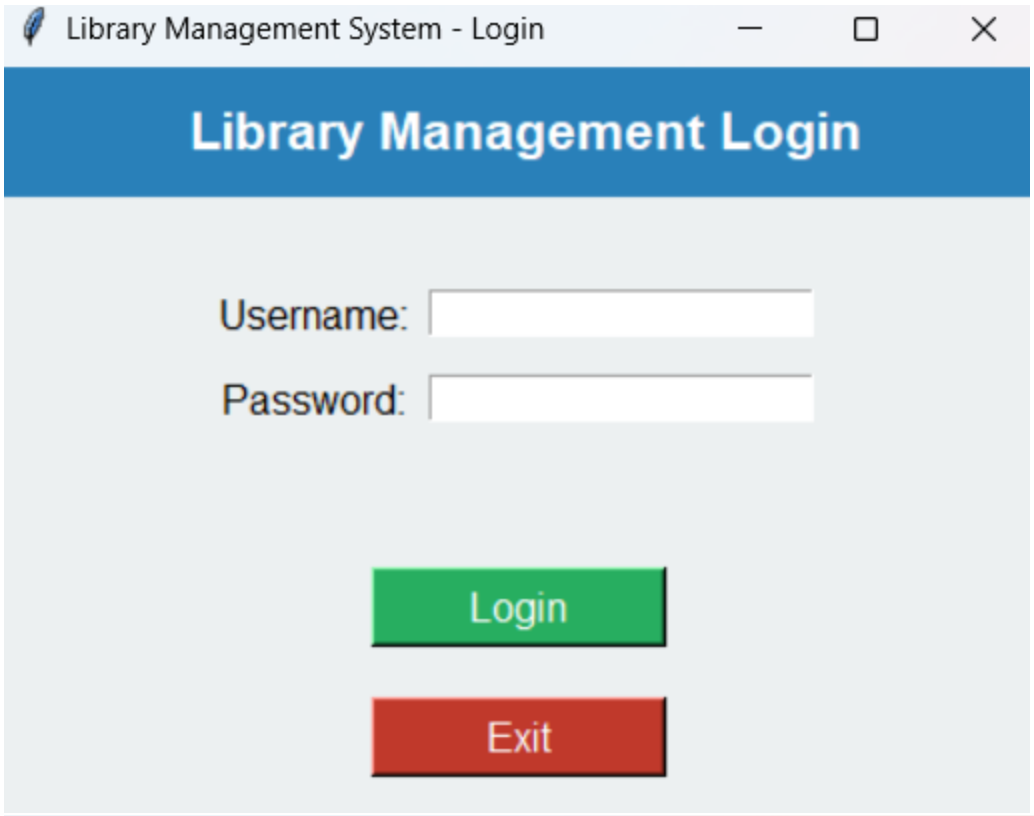
FR12: SQL Functions

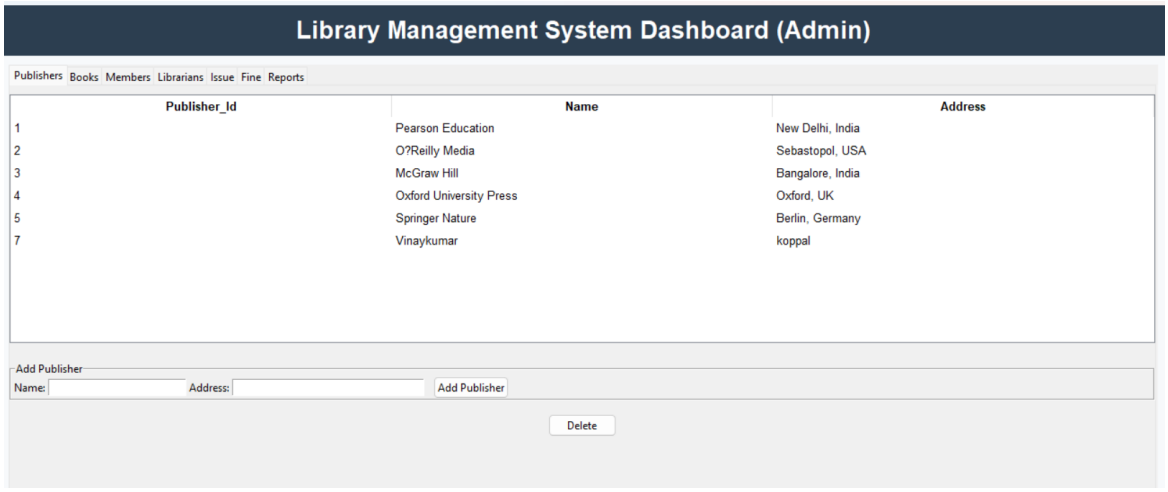
GetTotalFine

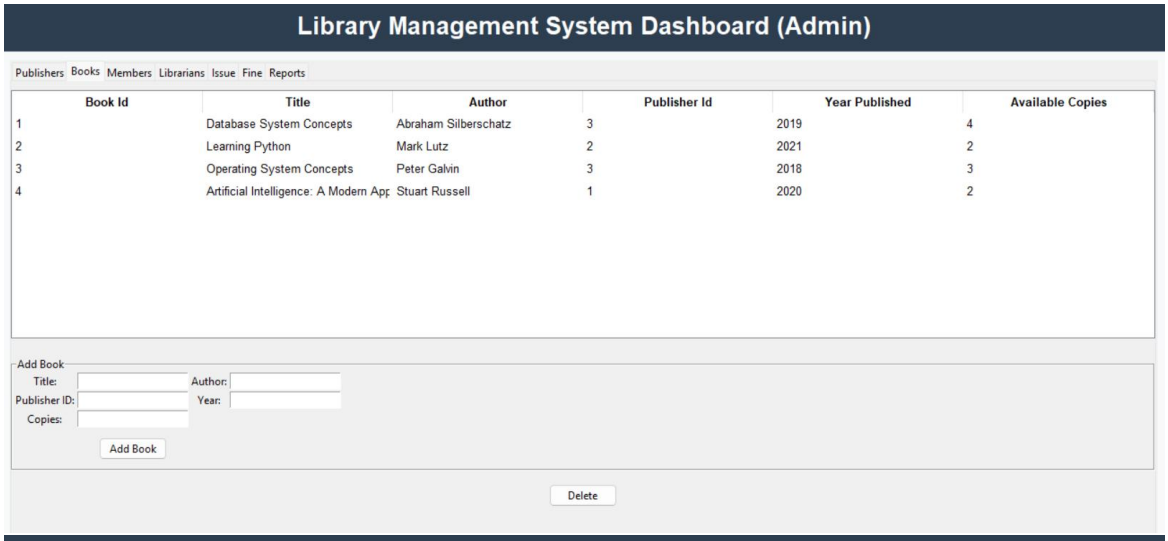
GetAvailableBooks

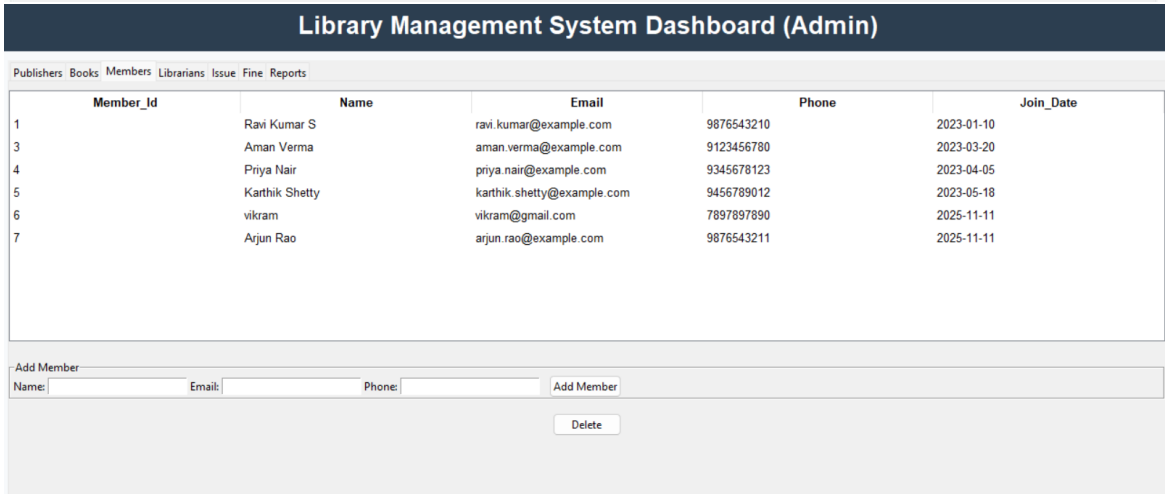
GetMemberCount

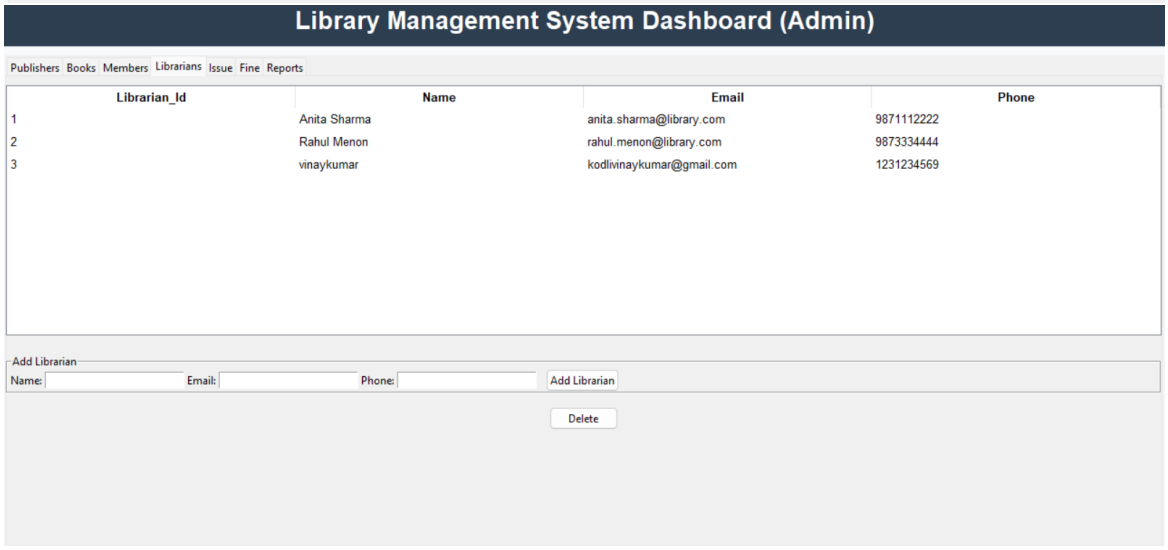
GetBooksByPublisher


Library Management System - Login
Library Management Login
Username:
Password:
Login
Exit


Library Management System Dashboard (Admin)
Publishers
Books
Members
Librarians
Issue
Fine
Reports
Publisher_Id
Name
Address
1
Pearson Education
New Delhi, India
2
O?Reilly Media
Sebastopol, USA
3
McGraw Hill
Bangalore, India
4
Oxford University Press
Oxford, UK
5
Springer Nature
Berlin, Germany
7
Vinaykumar
koppal
Add Publisher
Name:
Address:
Add Publisher
Delete


Library Management System Dashboard (Admin)
Publishers Books Members Librarians Issue Fine Reports
Book Id Title Author Publisher Id Year Published Available Copies
1 Database System Concepts Abraham Silberschatz 3 2019 4
2 Learning Python Mark Lutz 2 2021 2
3 Operating System Concepts Peter Galvin 3 2018 3
4 Artificial Intelligence: A Modern Apr Stuart Russell 1 2020 2
Add Book
Title:
Author:
Publisher ID:
Year:
Copies:
Add Book
Delete


Library Management System Dashboard (Admin)
Publishers Books Members Librarians Issue Fine Reports
Member_Id Name Email Phone Join_Date
1 Ravi Kumar S ravi.kumar@example.com 9876543210 2023-01-10
3 Aman Verma aman.verma@example.com 9123456780 2023-03-20
4 Priya Nair priya.nair@example.com 9345678123 2023-04-05
5 Karthik Shetty karthik.shetty@example.com 9456789012 2023-05-18
6 vikram vikram@gmail.com 7897897890 2025-11-11
7 Arjun Rao arjun.rao@example.com 9876543211 2025-11-11
Add Member
Name:
Email:
Phone:
Add Member
Delete


Library Management System Dashboard (Admin)
Publishers Books Members Librarians Issue Fine Reports
Librarian_Id Name Email Phone
1 Anita Sharma anita.sharma@library.com 9871112222
2 Rahul Menon rahul.menon@library.com 9873334444
3 vinaykumar kodlivinaykumar@gmail.com 1231234569
Add Librarian
Name:
Email:
Phone:
Add Librarian
Delete

Library Management System Dashboard (Admin)

Publishers | Books | Members | Librarians | Issue | Fine | Reports

Issue Id	Book Id	Member Id	Librarian Id	Issue Date	Return Date
1	2	4	3	2025-11-11	2025-11-11
2	1	1	1	2025-11-10	None
4	3	3	1	2025-11-08	None

Add Issue

Book ID: Member ID:

Librarian ID: Issue Date:

Return Date:

Add Issue

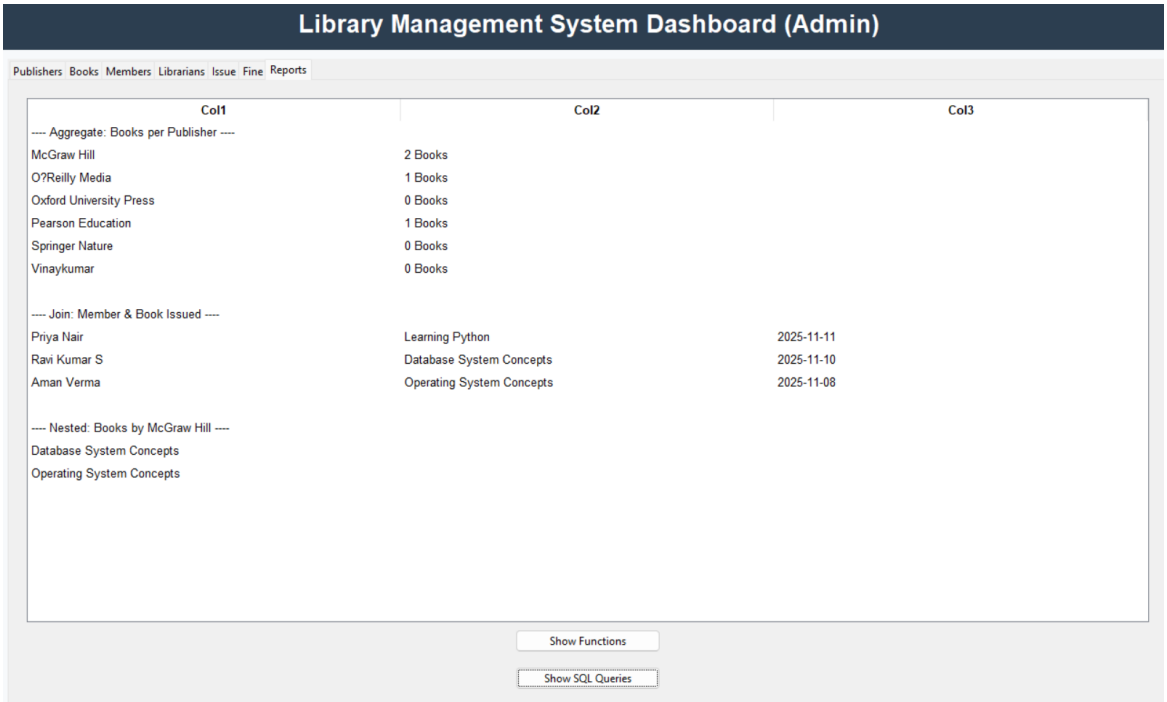
Library Management System Dashboard (Admin)

Publishers | Books | Members | Librarians | Issue | Fine | Reports

Fine_Id	Issue_Id	Amount	Status
1	1	10.00	unpaid
2	1	100.00	Unpaid
3	2	50.00	Paid

Add Fine

Issue ID: Amount: Status: Add Fine



10. Triggers / Procedures / Functions

```
--  ALL TRIGGERS (FULL SET)

DELIMITER $$

-- Trigger 1: Reduce available copies when a book is issued
CREATE TRIGGER trg_after_issue_insert
AFTER INSERT ON Issue
FOR EACH ROW
BEGIN
   UPDATE Books
   SET available_copies = available_copies - 1
   WHERE book_id = NEW.book_id;
END$$

-- Trigger 2: Increase available copies when a book is returned
CREATE TRIGGER trg_after_return_update
AFTER UPDATE ON Issue
FOR EACH ROW
```

```
BEGIN
   IF NEW.return_date IS NOT NULL
     AND (OLD.return_date IS NULL OR OLD.return_date <> NEW.return_date) THEN
      UPDATE Books
      SET available_copies = available_copies + 1
      WHERE book_id = NEW.book_id;
   END IF;
END$$

-- Log deleted books
CREATE TRIGGER trg_after_book_delete
AFTER DELETE ON Books
FOR EACH ROW
BEGIN
   INSERT INTO DeletedBooksLog (book_id, title)
   VALUES (OLD.book_id, OLD.title);
END$$

-- Log deleted members
CREATE TRIGGER trg_after_member_delete
AFTER DELETE ON Members
FOR EACH ROW
BEGIN
   INSERT INTO DeletedMembersLog (member_id, name)
   VALUES (OLD.member_id, OLD.name);
END$$

DELIMITER ;

--  ALL STORED PROCEDURES

DELIMITER $$

-- Add new book
CREATE PROCEDURE AddNewBook(
   IN p_title VARCHAR(150),
   IN p_author VARCHAR(100),
   IN p_publisher_id INT,
   IN p_year YEAR,
   IN p_copies INT
)
```

```
BEGIN
   INSERT INTO Books (title, author, publisher_id, year_published, available_copies)
   VALUES (p_title, p_author, p_publisher_id, p_year, p_copies);
END$$

-- Add new member
CREATE PROCEDURE AddNewMember(
   IN p_name VARCHAR(100),
   IN p_email VARCHAR(100),
   IN p_phone VARCHAR(15)
)
BEGIN
   INSERT INTO Members (name, email, phone)
   VALUES (p_name, p_email, p_phone);
END$$

-- Update member details
CREATE PROCEDURE UpdateMemberDetails(
   IN p_member_id INT,
   IN p_name VARCHAR(100),
   IN p_email VARCHAR(100),
   IN p_phone VARCHAR(15)
)
BEGIN
   UPDATE Members
   SET name = p_name,
      email = p_email,
      phone = p_phone
   WHERE member_id = p_member_id;
END$$

-- Delete book
CREATE PROCEDURE DeleteBook(IN p_book_id INT)
BEGIN
   DELETE FROM Books WHERE book_id = p_book_id;
END$$

-- Delete member
CREATE PROCEDURE DeleteMember(IN p_member_id INT)
BEGIN
   DELETE FROM Members WHERE member_id = p_member_id;
```

```
END$$

-- Mark fine as paid
CREATE PROCEDURE RecordFinePayment(IN p_fine_id INT)
BEGIN
   UPDATE Fine
   SET status = 'Paid'
   WHERE fine_id = p_fine_id;
END$$

DELIMITER ;

--  ALL FUNCTIONS


DELIMITER $$

-- Total fine for a member
CREATE FUNCTION GetTotalFine(p_member_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
   DECLARE total DECIMAL(10,2);
   SELECT SUM(f.amount)
   INTO total
   FROM Fine f
   JOIN Issue i ON f.issue_id = i.issue_id
   WHERE i.member_id = p_member_id;

   RETURN IFNULL(total, 0.00);
END$$

-- Total available books
CREATE FUNCTION GetAvailableBooks()
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE total INT;
   SELECT SUM(available_copies) INTO total FROM Books;
   RETURN total;
END$$
```

```
-- Total members count
CREATE FUNCTION GetMemberCount()
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE cnt INT;
   SELECT COUNT(*) INTO cnt FROM Members;
   RETURN cnt;
END$$

-- Number of books by a publisher
CREATE FUNCTION GetBooksByPublisher(p_publisher_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE count_books INT;
   SELECT COUNT(*) INTO count_books FROM Books WHERE publisher_id =
p_publisher_id;
   RETURN count_books;
END$$

DELIMITER ;

--  ALL SAMPLE SQL QUERIES

--  Aggregate Query: Count books per publisher
SELECT p.name AS Publisher, COUNT(b.book_id) AS TotalBooks
FROM Publishers p
LEFT JOIN Books b ON p.publisher_id = b.publisher_id
GROUP BY p.name;

--  Join Query: Show members and the books they have issued
SELECT m.name AS MemberName, b.title AS BookTitle,
      i.issue_date, i.return_date
FROM Members m
JOIN Issue i ON m.member_id = i.member_id
JOIN Books b ON i.book_id = b.book_id;

--  Nested Query: Books published by McGraw Hill
SELECT title
```

```
FROM Books
WHERE publisher_id IN (
   SELECT publisher_id
   FROM Publishers
   WHERE name = 'McGraw Hill'
);
```

11. Code Snippets (Summarized)

A. Invoking Stored Procedures (from Python GUI)

```
 ---- Add New Book (Stored Procedure: AddNewBook) ----
def add_book():
   if role == "Viewer":
      return messagebox.showwarning("Access Denied", "Viewers
cannot add data.")

   conn = connect_db()
   cur = conn.cursor()

   cur.callproc("AddNewBook", (
      b_title.get(),
      b_author.get(),
      b_pubid.get(),
      b_year.get(),
      b_copies.get()
   ))

   conn.commit()
   conn.close()
   load_table(book_table, "SELECT * FROM Books")


# ---- Add New Member (Stored Procedure: AddNewMember) ----
def add_member():
   if role == "Viewer":
      return messagebox.showwarning("Access Denied", "Viewers
cannot add data.")

   conn = connect_db()
```

```
    cur = conn.cursor()

    cur.callproc("AddNewMember", (
        m_name.get(),
        m_email.get(),
        m_phone.get()
    ))

    conn.commit()
    conn.close()
    load_table(mem_table, "SELECT * FROM Members")


# ---- Update Member Details (Stored Procedure:
UpdateMemberDetails) ----
def update_member_details(member_id, name, email, phone):
    conn = connect_db()
    cur = conn.cursor()

    cur.callproc("UpdateMemberDetails", (
        member_id,
        name,
        email,
        phone
    ))

    conn.commit()
    conn.close()


# ---- Delete Book (Stored Procedure: DeleteBook) ----
def delete_book(book_id):
    conn = connect_db()
    cur = conn.cursor()

    cur.callproc("DeleteBook", (book_id,))

    conn.commit()
    conn.close()
```

```
 ---- Delete Member (Stored Procedure: DeleteMember) ----
def delete_member(member_id):
    conn = connect_db()
    cur = conn.cursor()

    cur.callproc("DeleteMember", (member_id,))

    conn.commit()
    conn.close()


# ---- Record Fine Payment (Stored Procedure: RecordFinePayment) --
--
def record_fine_payment(fine_id):
    conn = connect_db()
    cur = conn.cursor()

    cur.callproc("RecordFinePayment", (fine_id,))

    conn.commit()
    conn.close()
```

B. Invoking SQL Functions from Python

```
def show_functions():
    conn = connect_db()
    cur = conn.cursor()

    # Calling MySQL functions GetAvailableBooks and
GetMemberCount
    cur.execute("SELECT GetAvailableBooks(), GetMemberCount();")
    result = cur.fetchone()

    tree.delete(*tree.get_children())
    tree.insert("", "end", values=("Total Available Books", result[0]))
    tree.insert("", "end", values=("Total Members", result[1]))

    conn.close()
```

```
 ---- Example: Calling Join, Aggregate, Nested Queries ----
def show_queries():
    conn = connect_db()
    cur = conn.cursor()

    # Aggregate: Count books per publisher
    cur.execute("""
        SELECT p.name, COUNT(b.book_id)
        FROM Publishers p LEFT JOIN Books b
        ON p.publisher_id = b.publisher_id
        GROUP BY p.name;
    """)
    agg = cur.fetchall()

    # Join: Members & issued books
    cur.execute("""
        SELECT m.name, b.title, i.issue_date
        FROM Members m
        JOIN Issue i ON m.member_id = i.member_id
        JOIN Books b ON i.book_id = b.book_id;
    """)
    join_data = cur.fetchall()

    # Nested Query: Books by McGraw Hill
    cur.execute("""
        SELECT title FROM Books
        WHERE publisher_id IN (
            SELECT publisher_id FROM Publishers WHERE name='McGraw
Hill'
        );
    """)
    nested = cur.fetchall()

    conn.close()
```

C. Trigger Execution

Trigger Name: trg_after_issue_insert

• Executes AFTER a new record is inserted into Issue table.
• Purpose: Reduce the available book copies when a book is issued.

```
-- Trigger Code
CREATE TRIGGER trg_after_issue_insert
AFTER INSERT ON Issue
FOR EACH ROW
BEGIN
   UPDATE Books
   SET available_copies = available_copies - 1
   WHERE book_id = NEW.book_id;
END;

-- ❗ Trigger Execution (Firing Example)
INSERT INTO Issue (book_id, member_id, librarian_id, issue_date)
VALUES (1, 1, 1, '2025-11-10');

-- ✓ Effect of Trigger
-- Book with book_id = 1 will have available_copies reduced by 1.
```

Trigger Name: trg_after_return_update

• Executes AFTER a record in Issue table is updated.
• Purpose: Increase available copies when a book is returned.

```
-- Trigger Code
CREATE TRIGGER trg_after_return_update
AFTER UPDATE ON Issue
FOR EACH ROW
BEGIN
   IF NEW.return_date IS NOT NULL
      AND (OLD.return_date IS NULL OR OLD.return_date <>
NEW.return_date) THEN
       UPDATE Books
```

```
      SET available_copies = available_copies + 1
      WHERE book_id = NEW.book_id;
   END IF;
END;

--  Trigger Execution (Firing Example)
UPDATE Issue
SET return_date = '2025-11-11'
WHERE issue_id = 1;

--  Effect of Trigger
-- Book copies are increased by 1 because the book was returned.
```

Trigger Name: trg_after_book_delete

- Executes AFTER a book record is deleted.
- Purpose: Log deleted book details into DeletedBooksLog table.

```
-- Trigger Code
CREATE TRIGGER trg_after_book_delete
AFTER DELETE ON Books
FOR EACH ROW
BEGIN
   INSERT INTO DeletedBooksLog (book_id, title)
   VALUES (OLD.book_id, OLD.title);
END;

--  Trigger Execution (Firing Example)
DELETE FROM Books WHERE book_id = 5;

--  Effect of Trigger
-- A new log entry is created in DeletedBooksLog.
```

Trigger Name: trg_after_member_delete

- Executes AFTER a member record is deleted.
- Purpose: Log deleted member information.

```
-- Trigger Code
CREATE TRIGGER trg_after_member_delete
AFTER DELETE ON Members
FOR EACH ROW
BEGIN
   INSERT INTO DeletedMembersLog (member_id, name)
   VALUES (OLD.member_id, OLD.name);
END;

--  Trigger Execution (Firing Example)
DELETE FROM Members WHERE member_id = 2;

-- Effect of Trigger
-- Deleted member is added to DeletedMembersLog.
```

12. SQL QUERIES

```
-- ============================
-- DATABASE
-- ============================
CREATE DATABASE IF NOT EXISTS LibraryDB;
USE LibraryDB;

-- ============================
-- TABLES (DDL)
-- ============================
CREATE TABLE Publishers (
   publisher_id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100) NOT NULL UNIQUE,
   address VARCHAR(200)
);

CREATE TABLE Books (
   book_id INT PRIMARY KEY AUTO_INCREMENT,
   title VARCHAR(150) NOT NULL,
   author VARCHAR(100) NOT NULL,
   publisher_id INT,
   year_published YEAR CHECK (year_published >= 1800),
```

```
    available_copies INT DEFAULT 1 CHECK (available_copies >= 0),
    CONSTRAINT fk_publisher FOREIGN KEY (publisher_id)
        REFERENCES Publishers(publisher_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);

CREATE TABLE Members (
    member_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    join_date DATE DEFAULT (CURRENT_DATE)
);

CREATE TABLE Librarian (
    librarian_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15)
);

CREATE TABLE Issue (
    issue_id INT PRIMARY KEY AUTO_INCREMENT,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    librarian_id INT,
    issue_date DATE DEFAULT (CURRENT_DATE),
    return_date DATE,
    CONSTRAINT fk_book FOREIGN KEY (book_id)
        REFERENCES Books(book_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_member FOREIGN KEY (member_id)
        REFERENCES Members(member_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_librarian FOREIGN KEY (librarian_id)
        REFERENCES Librarian(librarian_id)
        ON DELETE SET NULL
        ON UPDATE CASCADE
```

```
);

CREATE TABLE Fine (
   fine_id INT PRIMARY KEY AUTO_INCREMENT,
   issue_id INT NOT NULL,
   amount DECIMAL(8,2) CHECK (amount >= 0),
   status VARCHAR(20) DEFAULT 'Unpaid' CHECK (status IN ('Paid','Unpaid')),
   CONSTRAINT fk_issue FOREIGN KEY (issue_id)
      REFERENCES Issue(issue_id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);

CREATE TABLE DeletedBooksLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   book_id INT,
   title VARCHAR(150),
   deleted_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE DeletedMembersLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   member_id INT,
   name VARCHAR(100),
   deleted_on TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS Users (
   user_id INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) UNIQUE NOT NULL,
   password VARCHAR(100) NOT NULL,
   role ENUM('Admin', 'Librarian', 'Viewer') DEFAULT 'Librarian'
);

-- ============================
-- INSERT STATEMENTS (DML)
-- ============================
INSERT INTO Publishers (name, address) VALUES
('Pearson Education', 'New Delhi, India'),
('O'Reilly Media', 'Sebastopol, USA'),
('McGraw Hill', 'Bangalore, India'),
```

```
('Oxford University Press', 'Oxford, UK'),
('Springer Nature', 'Berlin, Germany'),
('vinaykatnur', 'bengaluru');

INSERT INTO Books (title, author, publisher_id, year_published, available_copies) VALUES
('Database System Concepts', 'Abraham Silberschatz', 3, 2019, 5),
('Learning Python', 'Mark Lutz', 2, 2021, 3),
('Operating System Concepts', 'Peter Galvin', 3, 2018, 4),
('Artificial Intelligence: A Modern Approach', 'Stuart Russell', 1, 2020, 2),
('Computer Networks', 'Andrew Tanenbaum', 4, 2017, 6);

INSERT INTO Members (name, email, phone, join_date) VALUES
('Ravi Kumar', 'ravi.kumar@example.com', '9876543210', '2023-01-10'),
('Sneha Reddy', 'sneha.reddy@example.com', '9876501234', '2023-02-14'),
('Aman Verma', 'aman.verma@example.com', '9123456780', '2023-03-20'),
('Priya Nair', 'priya.nair@example.com', '9345678123', '2023-04-05'),
('Karthik Shetty', 'karthik.shetty@example.com', '9456789012', '2023-05-18'),
('vikram', 'vikram@gmail.com', '7897897890', '2025-11-11');

INSERT INTO Librarian (name, email, phone) VALUES
('Anita Sharma', 'anita.sharma@library.com', '9871112222'),
('Rahul Menon', 'rahul.menon@library.com', '9873334444');

INSERT INTO Issue (book_id, member_id, librarian_id, issue_date) VALUES
(1, 1, 1, '2025-11-10'),
(2, 2, 2, '2025-11-09'),
(3, 3, 1, '2025-11-08');

UPDATE Issue SET return_date = '2025-11-11' WHERE issue_id = 1;

INSERT INTO Fine (issue_id, amount, status) VALUES
(1, 100, 'Unpaid'),
(2, 50, 'Paid'),
(3, 150, 'Unpaid');

INSERT INTO Users (username, password, role) VALUES
('admin', 'admin123', 'Admin'),
('anita', 'lib123', 'Librarian'),
('viewer', 'view123', 'Viewer');

-- ============================
```

```
-- TRIGGERS
-- ============================
DELIMITER $$

CREATE TRIGGER trg_after_issue_insert
AFTER INSERT ON Issue
FOR EACH ROW
BEGIN
   UPDATE Books
   SET available_copies = available_copies - 1
   WHERE book_id = NEW.book_id;
END$$

CREATE TRIGGER trg_after_return_update
AFTER UPDATE ON Issue
FOR EACH ROW
BEGIN
   IF NEW.return_date IS NOT NULL AND (OLD.return_date IS NULL OR OLD.return_date
<> NEW.return_date) THEN
      UPDATE Books
      SET available_copies = available_copies + 1
      WHERE book_id = NEW.book_id;
   END IF;
END$$

CREATE TRIGGER trg_after_book_delete
AFTER DELETE ON Books
FOR EACH ROW
BEGIN
   INSERT INTO DeletedBooksLog (book_id, title)
   VALUES (OLD.book_id, OLD.title);
END$$

CREATE TRIGGER trg_after_member_delete
AFTER DELETE ON Members
FOR EACH ROW
BEGIN
   INSERT INTO DeletedMembersLog (member_id, name)
   VALUES (OLD.member_id, OLD.name);
END$$
```

```
DELIMITER ;

-- ============================
-- STORED PROCEDURES
-- ============================
DELIMITER $$

CREATE PROCEDURE AddNewBook(
   IN p_title VARCHAR(150),
   IN p_author VARCHAR(100),
   IN p_publisher_id INT,
   IN p_year YEAR,
   IN p_copies INT
)
BEGIN
   INSERT INTO Books (title, author, publisher_id, year_published, available_copies)
   VALUES (p_title, p_author, p_publisher_id, p_year, p_copies);
END$$

CREATE PROCEDURE AddNewMember(
   IN p_name VARCHAR(100),
   IN p_email VARCHAR(100),
   IN p_phone VARCHAR(15)
)
BEGIN
   INSERT INTO Members (name, email, phone)
   VALUES (p_name, p_email, p_phone);
END$$

CREATE PROCEDURE UpdateMemberDetails(
   IN p_member_id INT,
   IN p_name VARCHAR(100),
   IN p_email VARCHAR(100),
   IN p_phone VARCHAR(15)
)
BEGIN
   UPDATE Members
   SET name = p_name,
      email = p_email,
      phone = p_phone
   WHERE member_id = p_member_id;
```

```
END$$

CREATE PROCEDURE DeleteBook(IN p_book_id INT)
BEGIN
   DELETE FROM Books WHERE book_id = p_book_id;
END$$

CREATE PROCEDURE DeleteMember(IN p_member_id INT)
BEGIN
   DELETE FROM Members WHERE member_id = p_member_id;
END$$

CREATE PROCEDURE RecordFinePayment(IN p_fine_id INT)
BEGIN
   UPDATE Fine
   SET status = 'Paid'
   WHERE fine_id = p_fine_id;
END$$

DELIMITER ;

-- ============================
-- FUNCTIONS
-- ============================
DELIMITER $$

CREATE FUNCTION GetTotalFine(p_member_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
   DECLARE total DECIMAL(10,2);
   SELECT SUM(f.amount)
   INTO total
   FROM Fine f
   JOIN Issue i ON f.issue_id = i.issue_id
   WHERE i.member_id = p_member_id;
   RETURN IFNULL(total, 0.00);
END$$

CREATE FUNCTION GetAvailableBooks()
RETURNS INT
```

```
DETERMINISTIC
BEGIN
   DECLARE total INT;
   SELECT SUM(available_copies) INTO total FROM Books;
   RETURN total;
END$$

CREATE FUNCTION GetMemberCount()
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE cnt INT;
   SELECT COUNT(*) INTO cnt FROM Members;
   RETURN cnt;
END$$

CREATE FUNCTION GetBooksByPublisher(p_publisher_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE count_books INT;
   SELECT COUNT(*) INTO count_books FROM Books WHERE publisher_id =
p_publisher_id;
   RETURN count_books;
END$$

DELIMITER ;

-- ============================
-- QUERIES (AGGREGATE / JOIN / NESTED)
-- ============================
SELECT p.name AS Publisher, COUNT(b.book_id) AS TotalBooks
FROM Publishers p
LEFT JOIN Books b ON p.publisher_id = b.publisher_id
GROUP BY p.name;

SELECT m.name AS MemberName, b.title AS BookTitle, i.issue_date, i.return_date
FROM Members m
JOIN Issue i ON m.member_id = i.member_id
JOIN Books b ON i.book_id = b.book_id;
```

```
SELECT title
FROM Books
WHERE publisher_id IN (
   SELECT publisher_id
   FROM Publishers
   WHERE name = 'McGraw Hill'
);
```

13. GitHub Repository Link

https://github.com/Vinaykumar1311/Library-Management-System