PADMA SESHADRI BALA BHAVAN

SENIOR SECONDARY SCHOOL,

NUNGAMBAKKAM, CHENNAI – 600034

Library Management System

User Manual & Project Documentation

Jhanvi Shankar

XII B

BONAFIDE CERTIFICATE

Certified to be the bonafide record of work done by Master / Miss .....JHANVI SHANKAR..... of Class .....XII 'B'..... in PADMA SESHADRI BALA BHAVAN SR. SEC. SCHOOL, CHENNAI.

During the year .....2025-26.....

Date ..... P.G.T. In .....

Chennai

Submitted for All - India Senior Secondary Practical held in ..PADMA SESHADRI BALA BHAVAN SR. SECONDARY SCHOOL at ..LAKE AREA, First Street,Chennai.

Date ..... Examiner

Seal

INDEX

Acknowledgement

I would like to express my sincere gratitude to all those who helped me complete this project successfully.

I am deeply grateful to my Computer Science teacher, Mrs Kala.K for her invaluable guidance, continuous support, and encouragement throughout the development of this Library Management System. Her expertise in Python programming and database management was instrumental in shaping this project.

I would like to thank my school Principal , Mrs.S.Vasanthi for providing the necessary infrastructure and resources, including access to computer lab facilities and MySQL database software, which were essential for developing and testing this application.

Case Study

Background

Libraries face significant challenges in managing their book inventory and member borrowing activities manually. Traditional paper-based systems are:

- Time-consuming and error-prone
- Difficult to track overdue books and defaulters
- Lack real-time availability information
- Cannot generate quick reports
- Prone to data loss and duplication

Problem Statement

Design and develop a computerized Library Management System that can:

1. Maintain a digital catalog of books with multiple copies
2. Manage member registration and profiles
3. Track book issuing and returns with due dates
4. Prevent issuing books to members with overdue items
5. Generate reports on transactions and defaulters
6. Provide role-based access for administrators and members
7. Ensure data integrity and maintain audit trails

Proposed Solution

A desktop application built using Python and MySQL that provides:

For Library Administrators:

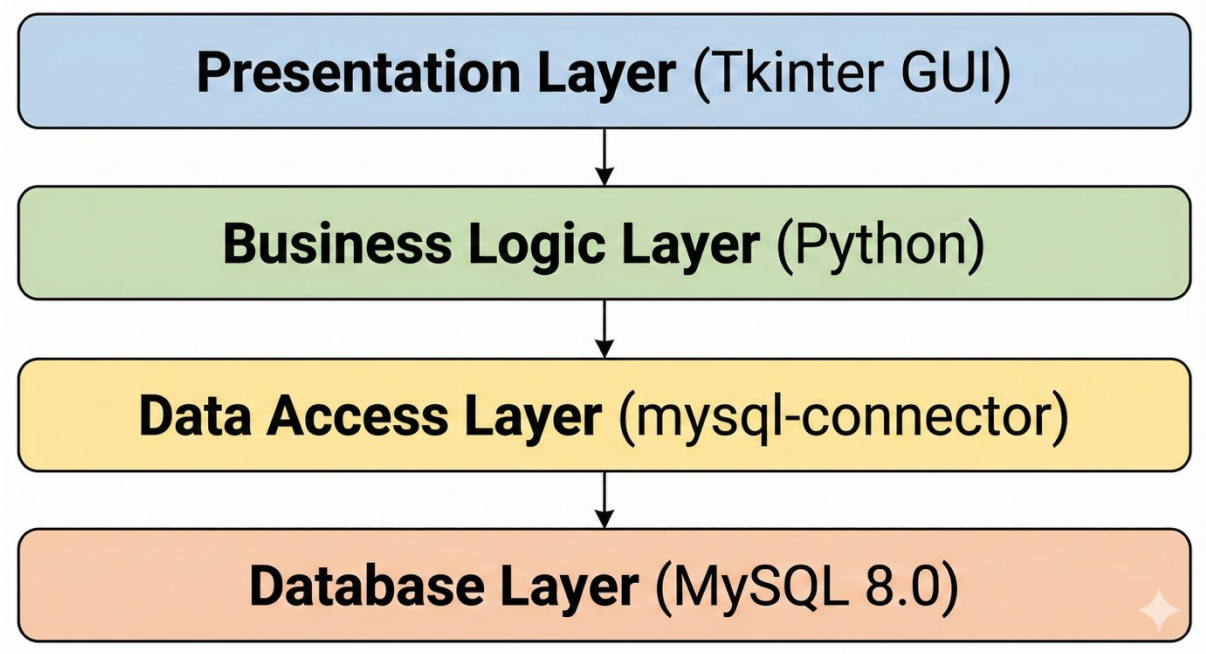
- Centralized dashboard
- Quick user registration (Admin/Member types) - Efficient book catalog management (1-5 copies per book)

- Book issuing with automatic validations like:
    - Check for overdue books (blocks issuing)
    - Enforce borrowing limit (3 books maximum)
    - Calculate due dates automatically (15 days)
- Simple return processing
- Soft delete functionality (preserve audit trail) –
- Generate real-time reports:
    - Weekly transaction reports
    - Defaulters list with overdue days
- Complete user directory

For Library Members:

- Personal dashboard showing borrowed books
- Visual overdue indicators (red highlighting)
- Warning legend for overdue items
- Searchable catalog of available books
- Due date tracking

Technical Architecture



System Benefits

For Library Administrators:

- Reduced manual workload
- Eliminates paper-based record keeping
- Instant access to member and book information
- Quick identification of defaulters
- Automated due date tracking
- Error-free transaction recording

For Library Members:

- Self-service book search capability
- Clear visibility of borrowed books and due dates
- Visual alerts for overdue items

MySQL Table Structure

Entity Relationship Diagram (ERD)

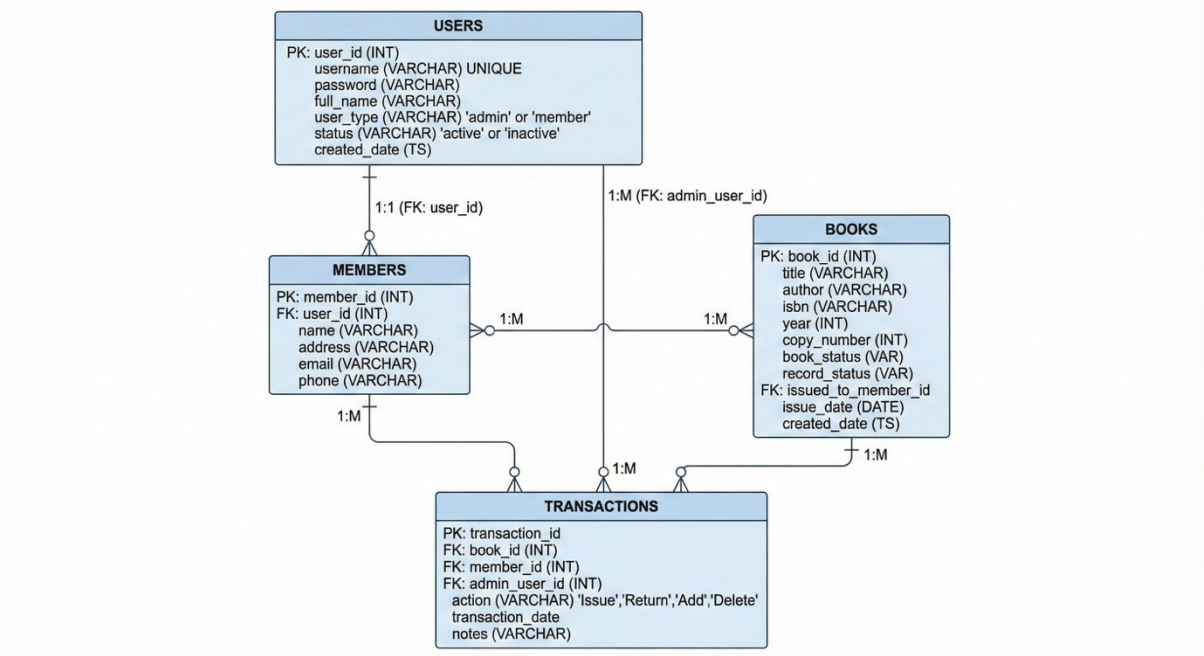


Table Definitions

Table: users

Purpose: Store profile information for users of the Library Management System

Field	Type	Constraints	Description
user_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for user
username	VARCHAR(50)	UNIQUE, NOT NULL	Login username
password	VARCHAR(50)	NOT NULL	Plain text password (educational purpose)
full_name	VARCHAR(100)	NOT NULL	User's full name
user_type	VARCHAR(20)	NOT NULL	'admin' or 'member'
status	VARCHAR(20)	DEFAULT 'active'	'active' or 'inactive'
created_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Account creation timestamp

Sample Data:

INSERT INTO users VALUES

(1, 'admin', 'admin123', 'Administrator', 'admin', 'active', '2024-10-01 10:00:00'),

(2, 'librarian', 'lib123', 'Librarian Name', 'admin', 'active', '2024-10-01 10:05:00'),

(3, 'priya', 'priya123', 'Priya Sharma', 'member', 'active', '2024-10-02 11:30:00'),

(4, 'rahul', 'rahul123', 'Rahul Kumar', 'member', 'active', '2024-10-02 14:20:00');

Table: members

Purpose: Store additional profile information for member-type

Field	Type	Constraints	Description
member_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique member identifier
user_id	INT	FOREIGN KEY → users(user_id)	Link to user account
name	VARCHAR(100)	NOT NULL	Member's full name
address	VARCHAR(200)	NULL allowed	Residential address
email	VARCHAR(100)	NULL allowed	Email address
phone	VARCHAR(15)	NULL allowed	Contact number

Sample Data:

INSERT INTO members VALUES

(1, 3, 'Priya Sharma', '123 MG Road, Delhi', 'priya@email.com', '9876543210'),

(2, 4, 'Rahul Kumar', '456 Park Street, Mumbai', 'rahul@email.com', '9123456780');

Table: books

Purpose: Store book catalog with multiple copy support and tracking information

Field	Type	Constraints	Description
book_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique book identifier
title	VARCHAR(200)	NOT NULL	Book title
author	VARCHAR(100)	NOT NULL	Book author
isbn	VARCHAR(20)	NOT NULL	ISBN number
year	INT	NOT NULL	Publication year
copy_number	INT	NOT NULL	Copy number (1, 2, 3...)
book_status	VARCHAR(20)	DEFAULT 'New'	'New', 'Issued', 'Returned'
record_status	VARCHAR(20)	DEFAULT 'Active'	'Active', 'Deleted'
issued_to_member_id	INT	FOREIGN KEY, NULL allowed	Currently issued to which member
issue_date	DATE	NULL allowed	Date when book was issued
created_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When book was added

Rules:

- Combination of (title, author, copy_number) should be unique

- book_status: 'New' (never issued), 'Issued' (currently out), 'Returned' (was issued, now available)
- record_status: 'Active' (visible), 'Deleted' (soft deleted, hidden from active view)
- issued_to_member_id is NULL when book is available
- issue_date is NULL when book is available

Sample Data:

INSERT INTO books VALUES

(1, 'Train to Pakistan', 'Khushwant Singh', '978-0143065883', 1956, 1, 'New', 'Active', NULL, NULL, '2024-10-03 09:00:00'),

(2, 'Train to Pakistan', 'Khushwant Singh', '978-0143065883', 1956, 2, 'Issued', 'Active', 1, '2024-12-01', '2024-10-03 09:00:00'),

Table: transactions

Purpose: Maintain complete audit trail of all library

operations

Field	Type	Constraints	Description
transaction_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique transaction identifier
book_id	INT	FOREIGN KEY, NULL allowed	Identifies which book is involved
member_id	INT	FOREIGN KEY, NULL allowed	Identifies which member is involved (for Issue)
admin_user_id	INT	FOREIGN KEY, NULL allowed	Identifies which admin performed the action
action	VARCHAR(20)	NOT NULL	'Issue', 'Return', 'Add', or 'Delete'
transaction_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of when the action occurred
notes	VARCHAR(200)	NULL allowed	Additional details (e.g., book title)

Rules:

- 'Issue': Book issued to member (member_id filled, admin_user_id filled)
- 'Return': Book returned by member (member_id filled, admin_user_id filled)
- 'Add': New book added to catalog (book_id filled, admin_user_id filled)
- 'Delete': Book soft-deleted (book_id filled, admin_user_id filled)

Sample Data:

INSERT INTO transactions VALUES

(1, 1, NULL, 1, 'Add', '2024-10-03 09:00:00', 'Train to Pakistan by Khushwant Singh (Copy
1)'),

(2, 2, 1, 1, 'Issue', '2024-12-01 10:30:00', 'Train to Pakistan by Khushwant Singh'),

(3, 3, 1, 1, 'Return', '2024-12-10 14:20:00', 'The God of Small Things by Arundhati Roy');

Relationships Summary

From Table	To Table	Type	Foreign Key	Description
users	members	1:1	members.user_id	Each member has one login
members	books	1:M	books.issued_to_member_id	Member can borrow multiple books
books	transactions	1:M	transactions.book_id	Each book has transaction history
members	transactions	1:M	transactions.member_id	Member’s borrowing history
users	transactions	1:M	transactions.admin_user_id	Admin’s action history

Algorithm

Login Authentication Algorithm

Name of the function: authenticate_user(username, password)

Input: username (string), password (string)

Output: Success (redirect to dashboard) or Error message

Steps:

1. START
2. GET username and password from input fields
3. TRIM whitespace from both inputs
4. IF username is empty OR password is empty THEN
   DISPLAY "Enter username and password" RETURN
5. CONNECT to database
6. EXECUTE query: "SELECT user_id, full_name, user_type, status
   FROM users
   WHERE username = ? AND password = ?"
7. FETCH result
8. CLOSE database connection
9. IF result is NULL THEN
   DISPLAY "Invalid username or password" RETURN
10. IF result.status != 'active' THEN

DISPLAY "Account is inactive. Please contact administrator."
RETURN

11. EXTRACT user_id, user_name, user_type from result

12. DISPLAY "Welcome, {user_name}!"
13. CLOSE login window
14. IF user_type == 'admin' THEN
    OPEN admin.py with user_id parameter
    ELSE
    OPEN member.py with user_id parameter
15. END

Add Book Algorithm

Name of the function: add_book(title, author, isbn, year, copies)

Input: title (string), author (string), isbn (string), year (int), copies (int)

Output: Success message or Error message

Steps:

1. START
2. GET all form field values and TRIM whitespace
3. IF any field is empty THEN
   DISPLAY "All fields are required!"
   RETURN
4. TRY to convert year and copies to integers CATCH ValueError:
   DISPLAY "Year and copies must be valid
   numbers" RETURN
5. IF copies < 1 THEN
   DISPLAY "Number of copies must be at
   least 1" RETURN
6. IF copies > 5 THEN

DISPLAY "Number of copies cannot be more

than 5" RETURN

7. CONNECT to database
8. EXECUTE query: "SELECT title, author FROM books

WHERE title = ? AND author = ? LIMIT 1"

9. IF duplicate found THEN

DISPLAY "Book '{title}' by {author} already exists in the library!"

CLOSE

connection

RETURN

10. FOR i = 1 TO copies DO

INSERT INTO books (title, author, isbn, year, copy_number,

book_status, record_status)

VALUES (title, author, isbn, year, i, 'New', 'Active')

GET last_insert_id as book_id

INSERT INTO transactions (book_id, admin_user_id, action, notes)

VALUES (book_id, current_admin_id, 'Add',

'{title} by

{author} (Copy {i})')

END FOR

11. COMMIT transaction
12. CLOSE connection
13. DISPLAY "{copies} cop{'y' if copies==1 else 'ies'} of '{title}' added successfully!"
14. CLEAR all form fields
15. END

Issue Book Algorithm

Name of the function: issue_book(title, author, member_username)

Input: title (string), author (string), member_username (string)

Output: Success with due date or Error message

Steps:

1. START
2. IF member_username is empty THEN
   DISPLAY "Please enter a member
   username" RETURN
3. CONNECT to database
4. CHECK if book is deleted:
   EXECUTE "SELECT record_status FROM books
   WHERE title=? AND author=? LIMIT 1"

IF record_status == 'Deleted' THEN
DISPLAY "Cannot issue deleted books"
CLOSE connection
RETURN

5. CHECK available copies:
   COUNT available = "SELECT COUNT(*) FROM books
   WHERE title=? AND author=?
   AND book_status IN ('New', 'Returned')
   AND record_status='Active'"
   IF available == 0 THEN
   DISPLAY "No copies available"

CLOSE connection
RETURN

6. VALIDATE member:
   EXECUTE "SELECT user_id FROM users
   WHERE username=? AND user_type='member'"
   IF user not found THEN

DISPLAY "Username not found"

CLOSE connection

RETURN

7. GET member_id:

EXECUTE "SELECT member_id FROM members WHERE user_id=?"

IF member not found THEN

DISPLAY "Member not found"

CLOSE connection

RETURN

8. PRIORITY CHECK 1 - OVERDUE BOOKS:

CALL get_overdue_books(member_id)

IF overdue_books list is NOT empty

THEN FORMAT overdue_list =

"\n".join(overdue_books) DISPLAY

"Cannot issue book to

'{member_username}'.

Member has overdue books:

{overdue_list}

Please return overdue books first."

CLOSE connection

RETURN

9. PRIORITY CHECK 2 - BORROWING LIMIT:

COUNT current_books = "SELECT COUNT(*) FROM books

WHERE issued_to_member_id=?

AND book_status='Issued'"

IF current_books >= 3 THEN

DISPLAY "Member '{member_username}' already has 3 books issued.

Maximum borrowing limit reached."

CLOSE connection

RETURN

10. FIND available book:

    EXECUTE "SELECT book_id

    FROM books WHERE

    title=? AND author=?

        AND book_status IN ('New', 'Returned')

    AND record_status='Active' LIMIT 1"

    GET book_id

11. SET issue_date = current_date

    UPDATE books SET book_status='Issued',

    issued_to_member_id=member_id, issue_date=issue_date

    WHERE book_id=book_id

12. INSERT INTO transactions (book_id, member_id, admin_user_id, action, notes)

    VALUES (book_id, member_id, current_admin_id, 'Issue', '{title} by {author}')

13. COMMIT transaction

14. CALCULATE due_date = issue_date + 15 days

15. DISPLAY "Book issued!

    Issue: {issue_date in

    dd/mm/yyyy}

    Due: {due_date in

    dd/mm/yyyy}"

16. REFRESH catalog display

17. CLOSE connection

18. END

Helper function algorithm

Name of the helper Function: get_overdue_books(member_id)

Input: member_id (int)

Output: List of overdue book descriptions

Steps:

1. GET today = current_date
2. EXECUTE "SELECT title, author, issue_date FROM books

   WHERE issued_to_member_id=? AND book_status='Issued'"
3. INITIALIZE overdue_books = empty list
4. FOR each issued_book DO

   CALCULATE due_date = issue_date + 15 days

   IF due_date < today THEN

   CALCULATE days_overdue = (today - due_date).days

   ADD "{title} by {author} ({days_overdue} days overdue)" to overdue_books

   END FOR
5. RETURN overdue_books

Return Book Algorithm

Name of the function: return_book(book_id)

Input: book_id (int)

Output: Success message or Error message

Steps:

1. START
2. GET book status from selected row
3. IF book_status != 'Issued' THEN

   DISPLAY "This copy is

   not issued" RETURN
4. CONNECT to database
5. GET member_id:

   EXECUTE "SELECT issued_to_member_id FROM books WHERE book_id=?"
6. UPDATE books SET book_status='Returned',

issued_to_member_id=NULL,

issue_date=NULL

WHERE book_id=book_id

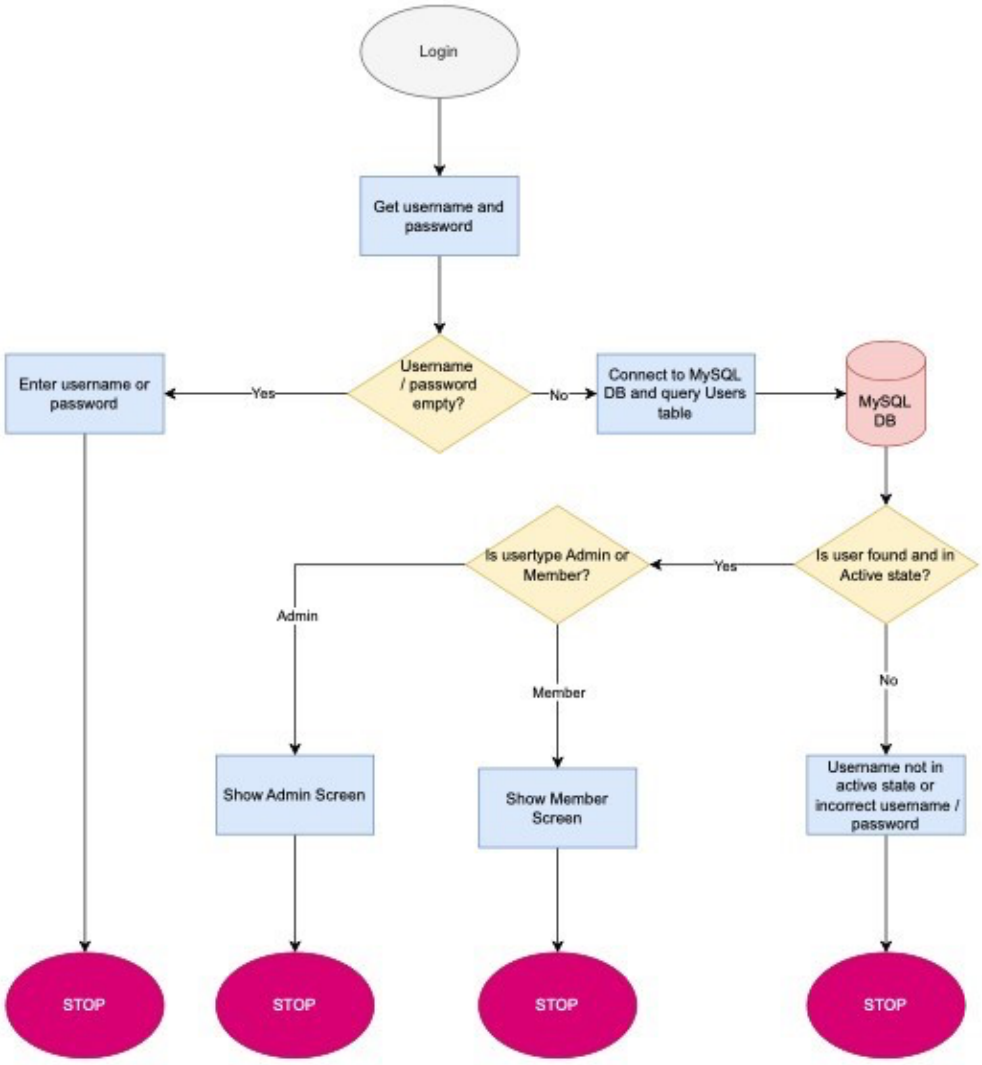
7. INSERT INTO transactions (book_id, member_id, admin_user_id, action, notes)

VALUES (book_id, member_id, current_admin_id, 'Return','{title} by {author}')

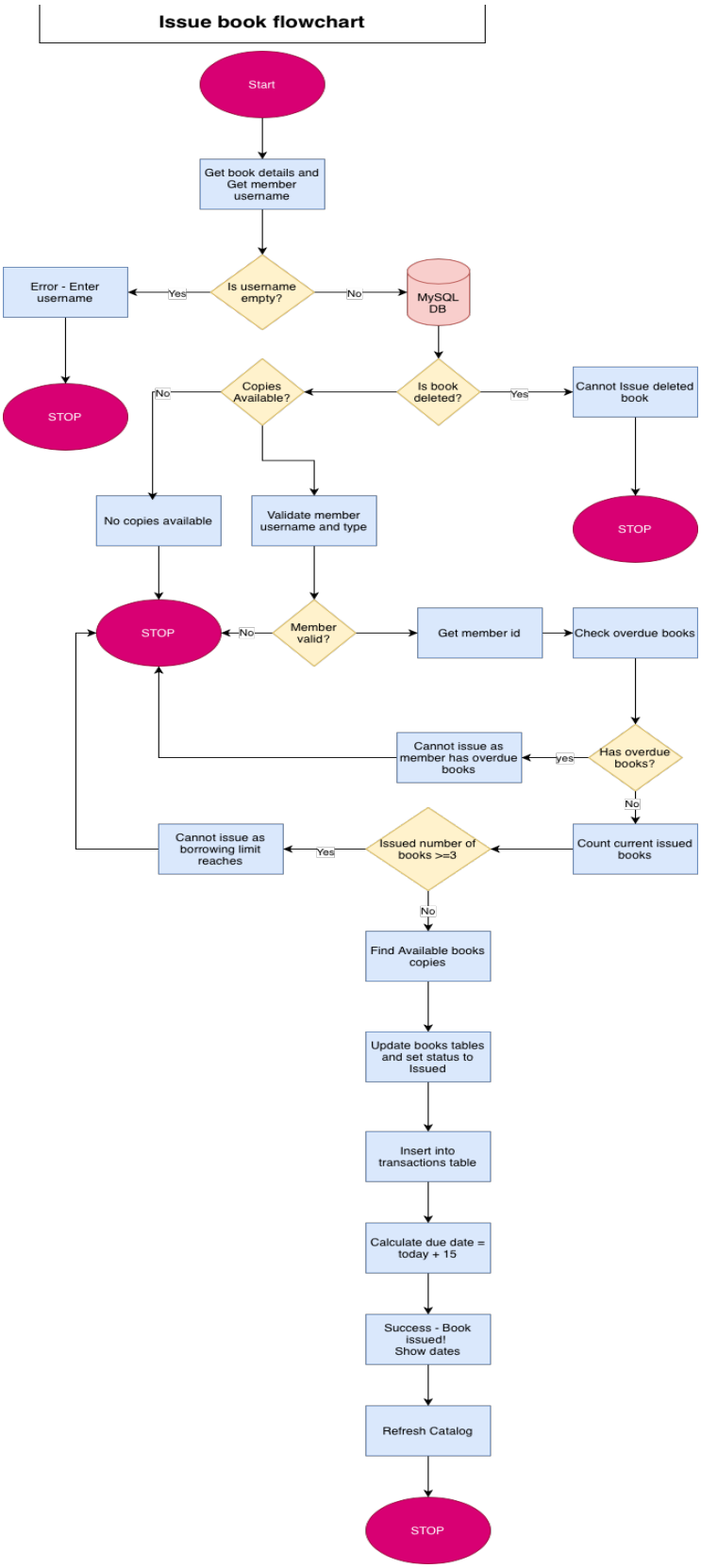
8. COMMIT transaction
9. CLOSE connection
10. DISPLAY "Book returned!"
11. REFRESH both copies list and main catalog
12. END

Flowchart

Login Process Flowchart



Issue Book Process Flowchart



Program Code

Main Program Files

This section contains the core Python code for the Library Management System.

login.py - Login Screen

Purpose: Authentication and role-based

redirection

Key Functions:

- `login()`: Validates credentials and opens appropriate dashboard

Code:

```
# Import libraries
import tkinter as tk
from tkinter import messagebox
import mysql.connector
import subprocess
import sys
from PIL import Image, ImageTk
from config import db_config

def login(event=None):
    # Get input values
    username = username_entry.get().strip()
    password = password_entry.get().strip()

    # Validate input
    if not username or not password:
        messagebox.showerror("Error", "Enter username and password")
        return

    # Database connection
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor()

    # Query user credentials
    query = """SELECT user_id, full_name, user_type, status
            FROM users WHERE username=%s AND password=%s"""
    cursor.execute(query, (username, password))
    result = cursor.fetchone()
    conn.close()
```

```
    # Process login result
    if result:
        if result[3] == 'active':
            user_id = result[0]
            user_name = result[1]
            user_type = result[2]

            messagebox.showinfo("Success", f"Welcome, {user_name}!")
            root.destroy()

            # Launch appropriate portal
            if user_type == 'admin':
                subprocess.Popen([sys.executable, 'admin.py', str(user_id)])
            else:
                subprocess.Popen([sys.executable, 'member.py', str(user_id)])
        else:
            messagebox.showerror("Error",
                "Account is inactive. Contact administrator.")
    else:
        messagebox.showerror("Error", "Invalid username or password")

# Main window setup
root = tk.Tk()
root.title("Library Management System")
root.geometry("800x600")
root.resizable(False, False)

# Background image
bg_image = Image.open("images/library.jpeg").resize((800, 600))
bg_photo = ImageTk.PhotoImage(bg_image)
canvas = tk.Canvas(root, width=800, height=600)
canvas.pack()
canvas.create_image(0, 0, image=bg_photo, anchor=tk.NW)

# Login frame
frame = tk.Frame(canvas, bg="white", relief=tk.RAISED, bd=2)
canvas.create_window(400, 300, window=frame, width=350, height=280)

# Title
tk.Label(frame, text="Library Management System",
         font=("Arial", 16, "bold"), bg="white", fg="#2196F3").pack(pady=15)

# Username
tk.Label(frame, text="Username", font=("Arial", 11, "bold"),
         bg="white", fg="#333").pack(pady=5)
username_entry = tk.Entry(frame, font=("Arial", 11), width=25)
username_entry.pack(pady=5)

# Password
```

```
tk.Label(frame, text="Password", font=("Arial", 11, "bold"),
        bg="white", fg="#333").pack(pady=5)
password_entry = tk.Entry(frame, font=("Arial", 11), width=25, show="*")
password_entry.pack(pady=5)

# Login button
tk.Button(frame, text="Login", font=("Arial", 11, "bold"),
         width=15, command=login).pack(pady=15)

# Enter key binding
password_entry.bind('<Return>', login)

# Run application
root.mainloop()
```

admin.py - Admin Dashboard

Purpose: Main control panel for administrators

Key Functions:

- `get_overdue_books(member_id, cur)`: Helper to check overdue books
- `create_list_window(title, size, columns, widths)`: Helper for report windows
- `issue_book()`: Issue book with validation
- `delete_book()`: Soft delete with validation
- `defaulters_list()`: Generate overdue report
- `view_users()`: Display all users

```
# Admin Dashboard - Library Management System

import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
import mysql.connector
import subprocess
import sys
from datetime import datetime, timedelta
from PIL import Image, ImageTk
from config import db_config
```

```
admin_user_id = int(sys.argv[1]) if len(sys.argv) > 1 else None

# Window tracking
open_windows = {
    'add_member': None,
    'add_book': None,
    'search_catalog': None,
    'weekly_report': None,
    'defaulters_list': None,
    'view_users': None
}

def get_conn():
    return mysql.connector.connect(**db_config)

def get_overdue_books(member_id, cur):
    """Get overdue books for a member"""
    today = datetime.now().date()
    query = """SELECT title, author, issue_date FROM books
               WHERE issued_to_member_id = %s AND book_status = 'Issued'"""
    cur.execute(query, (member_id,))
    issued_books = cur.fetchall()
    overdue_books = []
    for book in issued_books:
        due_date = book[2] + timedelta(days=15)
        if due_date < today:
            days_overdue = (today - due_date).days
            overdue_books.append(
                f"{book[0]} by {book[1]} ({days_overdue} days overdue)")
    return overdue_books

def create_list_window(title, size, columns, widths):
    """Create standard list window"""
    win = tk.Toplevel(root)
    win.title(title)
    win.geometry(size)
    tk.Label(win, text=title, font=("Arial", 16, "bold")).pack(pady=10)
```

```
    tree = ttk.Treeview(win, columns=columns, show='headings', height=18)
    for i, col in enumerate(columns):
        tree.heading(col, text=col)
        tree.column(col, width=widths[i])
    tree.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

    return win, tree

def add_user():
    if open_windows['add_member'] is not None and \
       open_windows['add_member'].poll() is None:
        return
    open_windows['add_member'] = subprocess.Popen(
        [sys.executable, 'add_member.py'])

def add_book():
    if open_windows['add_book'] is not None and \
       open_windows['add_book'].poll() is None:
        return
    open_windows['add_book'] = subprocess.Popen(
        [sys.executable, 'add_book.py', str(admin_user_id)])

def weekly_report():
    if open_windows['weekly_report'] is not None and \
       open_windows['weekly_report'].winfo_exists():
        open_windows['weekly_report'].lift()
        return

    cols = ('Date', 'Action', 'Book', 'Member', 'Notes')
    widths = [100, 80, 80, 120, 360]
    win, tree = create_list_window(
        "Weekly Library Report", "800x500", cols, widths)
    open_windows['weekly_report'] = win

    conn = get_conn()
    cur = conn.cursor()
    week_ago = datetime.now() - timedelta(days=7)
    query = """SELECT action, book_id, member_id, transaction_date,
               notes, admin_user_id FROM transactions
```

```
            WHERE transaction_date >= %s ORDER BY transaction_date DESC"""
    cur.execute(query, (week_ago,))
    transactions = cur.fetchall()

    for trans in transactions:
        date_str = trans[3].strftime('%d/%m/%Y') if trans[3] else 'N/A'
        action = trans[0]

        member_name = 'N/A'
        if action == 'Add' or action == 'Delete':
            if trans[5]:
                cur.execute("SELECT username FROM users WHERE user_id = %s",
                        (trans[5],))
                admin = cur.fetchone()
                if admin:
                    member_name = admin[0]
        elif action == 'Return':
            if trans[5]:
                cur.execute("SELECT username FROM users WHERE user_id = %s",
                        (trans[5],))
                admin = cur.fetchone()
                if admin:
                    member_name = admin[0]
        else:
            if trans[2]:
                cur.execute("""SELECT username FROM users u
                            JOIN members m ON u.user_id = m.user_id
                            WHERE m.member_id = %s""", (trans[2],))
                member = cur.fetchone()
                if member:
                    member_name = member[0]

        tree.insert('', tk.END, values=(
            date_str, trans[0], trans[1], member_name, trans[4] or ''))
    conn.close()

def defaulters_list():
    if open_windows['defaulters_list'] is not None and \
       open_windows['defaulters_list'].winfo_exists():
```

```
    open_windows['defaulters_list'].lift()
    return

cols = ('Member Name', 'Email', 'Phone', 'Book Title',
        'Author', 'Due Date', 'Days Overdue')
widths = [120, 150, 100, 180, 120, 90, 100]
win, tree = create_list_window(
    "Defaulters List - Overdue Books", "900x500", cols, widths)
open_windows['defaulters_list'] = win

conn = get_conn()
cur = conn.cursor()
today = datetime.now().date()

query = """SELECT b.title, b.author, b.issue_date, m.name,
           m.email, m.phone, b.book_id FROM books b
           JOIN members m ON b.issued_to_member_id = m.member_id
           WHERE b.book_status = 'Issued' AND b.issue_date IS NOT NULL"""
cur.execute(query)
issued_books = cur.fetchall()

defaulters_found = False
for book in issued_books:
    title, author, issue_date, member_name, email, phone, book_id = book
    due_date = issue_date + timedelta(days=15)

    if due_date < today:
        defaulters_found = True
        days_overdue = (today - due_date).days
        due_date_str = due_date.strftime('%d/%m/%Y')
        tree.insert('', tk.END, values=(
            member_name, email or 'N/A', phone or 'N/A',
            title, author, due_date_str, days_overdue))

if not defaulters_found:
    tk.Label(win, text="No overdue books found!",
             font=("Arial", 12), fg="green").pack(pady=20)

conn.close()
```

```
def view_users():
    if open_windows['view_users'] is not None and \
       open_windows['view_users'].winfo_exists():
        open_windows['view_users'].lift()
        return

    cols = ('Username', 'User Type', 'Date Added', 'Email', 'Phone')
    widths = [150, 100, 150, 250, 150]
    win, tree = create_list_window("All Users", "900x500", cols, widths)
    open_windows['view_users'] = win

    conn = get_conn()
    cur = conn.cursor()

    query = """SELECT u.username, u.user_type, u.created_date,
               m.email, m.phone FROM users u
               LEFT JOIN members m ON u.user_id = m.user_id
               ORDER BY u.created_date DESC"""
    cur.execute(query)
    users = cur.fetchall()

    for user in users:
        username, user_type, created_date, email, phone = user
        date_str = created_date.strftime('%d/%m/%Y %H:%M') \
            if created_date else 'N/A'
        tree.insert('', tk.END, values=(
            username, user_type.title(), date_str,
            email or 'N/A', phone or 'N/A'))

    conn.close()

def search_catalog():
    if open_windows['search_catalog'] is not None and \
       open_windows['search_catalog'].winfo_exists():
        open_windows['search_catalog'].lift()
        return

    win = tk.Toplevel(root)
```

```
win.title("Search Catalog")
win.geometry("900x500")
open_windows['search_catalog'] = win

search_frame = tk.Frame(win)
search_frame.pack(pady=5)
tk.Label(search_frame, text="Search:",
        font=("Arial", 11)).pack(side=tk.LEFT, padx=5)
search_entry = tk.Entry(search_frame, font=("Arial", 11), width=30)
search_entry.pack(side=tk.LEFT, padx=5)
tk.Label(search_frame, text="Filter:",
        font=("Arial", 11)).pack(side=tk.LEFT, padx=5)
filter_var = tk.StringVar(value="Active")
filter_dropdown = tk.OptionMenu(search_frame, filter_var,
                                "All", "Active", "Deleted")
filter_dropdown.config(font=("Arial", 10))
filter_dropdown.pack(side=tk.LEFT, padx=5)

cols = ('Title', 'Author', 'Year', 'Total', 'Available', 'Action')
tree = ttk.Treeview(win, columns=cols, show='headings', height=15)
widths = [250, 200, 60, 80, 80, 180]
for i, col in enumerate(cols):
    tree.heading(col, text=col)
    tree.column(col, width=widths[i])
tree.pack(pady=10, fill=tk.BOTH, expand=True)

popup_window = None

def search():
    for row in tree.get_children():
        tree.delete(row)
    conn = get_conn()
    cur = conn.cursor()
    search_term = '%' + search_entry.get() + '%'
    filter_status = filter_var.get()

    if filter_status == "All":
        query = """SELECT DISTINCT title, author, year, record_status
                FROM books WHERE title LIKE %s OR author LIKE %s
```

```
            OR year LIKE %s"""
        cur.execute(query, (search_term, search_term, search_term))
    else:
        query = """SELECT DISTINCT title, author, year, record_status
                FROM books WHERE (title LIKE %s OR author LIKE %s
                OR year LIKE %s) AND record_status=%s"""
        cur.execute(query, (search_term, search_term,
                    search_term, filter_status))
    books = cur.fetchall()

    for book in books:
        title, author, rec_status = book[0], book[1], book[3]
        if filter_status == "All":
            cur.execute("SELECT COUNT(*) FROM books WHERE title=%s AND author=%s",
                    (title, author))
            total = cur.fetchone()[0]
            cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                    AND author=%s AND book_status IN ('Returned', 'New')
                    AND record_status='Active'""", (title, author))
            available = cur.fetchone()[0]
        else:
            cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                    AND author=%s AND record_status=%s""",
                    (title, author, filter_status))
            total = cur.fetchone()[0]
            if filter_status == "Deleted":
                available = 0
            else:
                cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                        AND author=%s AND book_status IN ('Returned', 'New')
                        AND record_status=%s""", (title, author, filter_status))
                available = cur.fetchone()[0]

        status_suffix = f" [{rec_status}]" \
            if filter_status == "All" and rec_status == "Deleted" else ""
        if rec_status == "Deleted" or filter_status == "Deleted":
            action_text = "Deleted Book"
        else:
            action_text = "Double-click to view copies"
```

```
            tree.insert('', tk.END, values=(
                book[0] + status_suffix, book[1], book[2],
                total, available, action_text))
        conn.close()

    def show_copies(_):
        nonlocal popup_window

        if popup_window and popup_window.winfo_exists():
            popup_window.destroy()

        sel = tree.selection()
        if not sel:
            return
        item_values = tree.item(sel[0])['values']

        if item_values[5] == "Deleted Book":
            return

        title = item_values[0].replace(" [Deleted]", "")
        author = item_values[1]

        popup = tk.Toplevel(win)
        popup.title(f"{title} - All Copies")
        popup.geometry("900x500")
        popup_window = popup

        tk.Label(popup, text=f"{title} by {author}",
                 font=("Arial", 14, "bold"), pady=10).pack()

        copy_cols = ('Book ID', 'Copy#', 'Status',
                     'Issued To', 'Issue Date', 'Due Date')
        copy_tree = ttk.Treeview(popup, columns=copy_cols,
                                 show='headings', height=12)
        copy_widths = [80, 70, 90, 200, 120, 120]
        for i, col in enumerate(copy_cols):
            copy_tree.heading(col, text=col)
            copy_tree.column(col, width=copy_widths[i])
        copy_tree.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)
```

```
def refresh_copies():
    for row in copy_tree.get_children():
        copy_tree.delete(row)

    conn = get_conn()
    cur = conn.cursor()
    query = """SELECT book_id, copy_number, book_status,
            issued_to_member_id, issue_date, record_status
            FROM books WHERE title=%s AND author=%s"""
    cur.execute(query, (title, author))
    copies = cur.fetchall()

    for copy in copies:
        issued_to = 'N/A'
        issue_date_str = 'N/A'
        due_date_str = 'N/A'
        book_status = copy[2] if copy[5] == 'Active' else 'Deleted'
        if copy[3]:
            cur.execute("""SELECT name, username FROM members m
                    JOIN users u ON m.user_id = u.user_id
                    WHERE member_id = %s""", (copy[3],))
            member = cur.fetchone()
            if member:
                issued_to = f"{member[0]} ({member[1]})"
        if copy[4]:
            issue_date_str = copy[4].strftime('%d/%m/%Y')
            due_date = copy[4] + timedelta(days=15)
            due_date_str = due_date.strftime('%d/%m/%Y')
        copy_tree.insert('', tk.END, values=(
            copy[0], copy[1], book_status, issued_to,
            issue_date_str, due_date_str))
    conn.close()

refresh_copies()

def return_selected():
    sel_copy = copy_tree.selection()
    if not sel_copy:
```

```
            messagebox.showwarning("Warning", "Select a copy to return")
            return
        book_id = copy_tree.item(sel_copy[0])['values'][0]
        status = copy_tree.item(sel_copy[0])['values'][2]
        if status != 'Issued':
            messagebox.showerror("Error", "This copy is not issued")
            return
        conn = get_conn()
        cur = conn.cursor()
        cur.execute("SELECT issued_to_member_id FROM books WHERE book_id=%s",
                    (book_id,))
        member_id = cur.fetchone()[0]
        cur.execute("""UPDATE books SET book_status='Returned',
                    issued_to_member_id=NULL, issue_date=NULL
                    WHERE book_id=%s""", (book_id,))
        cur.execute("""INSERT INTO transactions
                    (book_id, member_id, admin_user_id, action, notes)
                    VALUES (%s, %s, %s, 'Return', %s)""",
                    (book_id, member_id, admin_user_id, f"{title} by {author}"))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "Book returned!")
        refresh_copies()
        search()

    button_frame = tk.Frame(popup)
    button_frame.pack(side=tk.BOTTOM, pady=15)
    tk.Button(button_frame, text="Return Selected Copy",
              command=return_selected, font=("Arial", 11, "bold"),
              width=25).pack()

tree.bind('<Double-1>', show_copies)

def issue_book():
    sel = tree.selection()
    if not sel:
        messagebox.showwarning("Warning", "Select a book first")
        return
```

```
item_values = tree.item(sel[0])['values']
title = item_values[0].replace(" [Deleted]", "")
author = item_values[1]
available = item_values[4]

conn = get_conn()
cur = conn.cursor()
cur.execute("""SELECT record_status FROM books
            WHERE title=%s AND author=%s LIMIT 1""", (title, author))
book_rec = cur.fetchone()
if book_rec and book_rec[0] == 'Deleted':
    conn.close()
    messagebox.showerror("Error", "Cannot issue deleted books")
    return

if available == 0:
    conn.close()
    messagebox.showerror("Error", "No copies available")
    return

member_username = simpledialog.askstring("Issue Book",
                                 "Enter member username:")
if member_username:
    cur.execute("""SELECT user_id FROM users WHERE username = %s
                AND user_type = 'member'""", (member_username,))
    user = cur.fetchone()
    if user:
        cur.execute("SELECT member_id FROM members WHERE user_id = %s",
                    (user[0],))
        member = cur.fetchone()
        if member:
            # Priority check: overdue books
            overdue_books = get_overdue_books(member[0], cur)
            if overdue_books:
                conn.close()
                overdue_list = "\n".join(overdue_books)
                messagebox.showerror("Error",
                    f"Cannot issue book to '{member_username}'.\n\n"
                    f"Member has overdue books:\n{overdue_list}\n\n"
```

```
            f"Please return overdue books first.")
        return

    # Check borrowing limit
    cur.execute("""SELECT COUNT(*) FROM books
                WHERE issued_to_member_id = %s
                AND book_status = 'Issued'""", (member[0],))
    current_books = cur.fetchone()[0]
    if current_books >= 3:
        conn.close()
        messagebox.showerror("Error",
            f"Member '{member_username}' already has 3 books.\n"
            f"Maximum borrowing limit reached.")
        return

    cur.execute("""SELECT book_id FROM books WHERE title=%s
                AND author=%s AND book_status IN ('Returned', 'New')
                AND record_status='Active' LIMIT 1""",
              (title, author))
    available_book = cur.fetchone()
    if available_book:
        book_id = available_book[0]
        issue_date = datetime.now().date()
        cur.execute("""UPDATE books SET book_status='Issued',
                    issued_to_member_id=%s, issue_date=%s
                    WHERE book_id=%s""",
                  (member[0], issue_date, book_id))
        cur.execute("""INSERT INTO transactions
                    (book_id, member_id, admin_user_id, action, notes)
                    VALUES (%s, %s, %s, 'Issue', %s)""",
                  (book_id, member[0], admin_user_id,
                   f"{title} by {author}"))
        conn.commit()
        due_date = issue_date + timedelta(days=15)
        messagebox.showinfo("Success",
            f"Book issued!\n"
            f"Issue: {issue_date.strftime('%d/%m/%Y')}\n"
            f"Due: {due_date.strftime('%d/%m/%Y')}")
        search()
```

```
                else:
                    messagebox.showerror("Error", "No available copy found")
            else:
                messagebox.showerror("Error", "Member not found")
        else:
            messagebox.showerror("Error", "Username not found")
        conn.close()

def delete_book():
    sel = tree.selection()
    if not sel:
        messagebox.showwarning("Warning", "Select a book first")
        return

    item_values = tree.item(sel[0])['values']
    title = item_values[0].replace(" [Deleted]", "")
    author = item_values[1]

    conn = get_conn()
    cur = conn.cursor()

    cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                AND author=%s AND record_status='Deleted'""",
            (title, author))
    deleted_count = cur.fetchone()[0]
    cur.execute("SELECT COUNT(*) FROM books WHERE title=%s AND author=%s",
            (title, author))
    total_count = cur.fetchone()[0]

    if deleted_count > 0 and deleted_count == total_count:
        conn.close()
        messagebox.showerror("Error",
            f"Cannot delete '{title}' by {author}.\n\n"
            f"This book is already deleted.")
        return

    cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                AND author=%s AND book_status='Issued'""", (title, author))
    issued_count = cur.fetchone()[0]
```

```
    if issued_count > 0:
        conn.close()
        copy_text = "y is" if issued_count == 1 else "ies are"
        messagebox.showerror("Error",
            f"Cannot delete '{title}' by {author}.\n\n"
            f"{issued_count} cop{copy_text} currently issued.\n"
            f"Please return all copies before deleting.")
        return

    confirm = messagebox.askyesno("Delete Book",
        f"Delete '{title}' by {author}?\n\n"
        f"All copies will be marked deleted.")
    if confirm:
        cur.execute("""UPDATE books SET record_status='Deleted'
                    WHERE title=%s AND author=%s""", (title, author))
        cur.execute("""SELECT book_id FROM books WHERE title=%s
                    AND author=%s LIMIT 1""", (title, author))
        book = cur.fetchone()
        if book:
            cur.execute("""INSERT INTO transactions
                        (book_id, member_id, admin_user_id, action, notes)
                        VALUES (%s, NULL, %s, 'Delete', %s)""",
                    (book[0], admin_user_id, f"{title} by {author}"))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "Book deleted!")
        search()

button_frame = tk.Frame(win)
button_frame.pack(pady=10)
tk.Button(button_frame, text="Search", command=search,
          font=("Arial", 11, "bold"), width=15).pack(side=tk.LEFT, padx=5)
tk.Button(button_frame, text="Issue Book", command=issue_book,
          font=("Arial", 11, "bold"), width=15).pack(side=tk.LEFT, padx=5)
tk.Button(button_frame, text="Delete Book", command=delete_book,
          font=("Arial", 11, "bold"), width=15).pack(side=tk.LEFT, padx=5)
search()
```

```
# Main window
root = tk.Tk()
root.title("Library Management System - Admin")
root.geometry("800x600")
root.resizable(False, False)

# Center window
root.update_idletasks()
x = (root.winfo_screenwidth() // 2) - 400
y = (root.winfo_screenheight() // 2) - 300
root.geometry(f'800x600+{x}+{y}')

# Header
header = tk.Frame(root, bg="#2196F3", height=80)
header.pack(fill=tk.X)

# Admin icon
try:
    admin_icon_img = Image.open("images/admin_portal.png").resize((40, 40))
    admin_icon_photo = ImageTk.PhotoImage(admin_icon_img)
    tk.Label(header, image=admin_icon_photo,
             bg="#2196F3").place(x=230, y=20)
    header.admin_icon = admin_icon_photo
except:
    pass

tk.Label(header, text="Admin Dashboard", font=("Arial", 18, "bold"),
         bg="#2196F3", fg="white").place(x=280, y=25)
tk.Button(header, text="Logout", font=("Arial", 10),
          command=root.destroy).place(x=700, y=25)

content = tk.Frame(root, bg="#f5f5f5")
content.pack(fill=tk.BOTH, expand=True, padx=40, pady=40)

button_container = tk.Frame(content, bg="#f5f5f5")
button_container.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

# Button icons
button_icons = {}
```

```
icon_files = {
    "Add Member": "images/add_member.png",
    "Add Book": "images/add_books.png",
    "View Users": "images/view_users.png",
    "Search Catalog": "images/search_catalog.png",
    "Weekly Reports": "images/weekly_report.png",
    "Defaulters List": "images/defaulters_list.png"
}

for name, file in icon_files.items():
    try:
        img = Image.open(file).resize((50, 50))
        button_icons[name] = ImageTk.PhotoImage(img)
    except:
        button_icons[name] = None

buttons = [("Add Member", add_user), ("Add Book", add_book),
           ("View Users", view_users), ("Search Catalog", search_catalog),
           ("Weekly Reports", weekly_report), ("Defaulters List", defaulters_list)]

# 3x2 button grid
for i, (text, cmd) in enumerate(buttons):
    btn_frame = tk.Frame(button_container, bg="#f5f5f5")
    btn_frame.grid(row=i//3, column=i%3, padx=20, pady=20)

    if button_icons.get(text):
        tk.Label(btn_frame, image=button_icons[text],
                 bg="#f5f5f5").pack(side=tk.TOP, pady=5)

    tk.Button(btn_frame, text=text, font=("Arial", 12, "bold"),
              width=20, height=2, command=cmd).pack(side=tk.TOP)

root.button_icons = button_icons

root.mainloop()
```

member.py - Member Dashboard

Purpose: Member portal with overdue indicators

Key Functions:

- `refresh_borrowed()`: Display borrowed books with red highlighting
- `search_books()`: Search available books

Code

```
# Member Portal - Library Management System



import tkinter as tk
from tkinter import ttk
import mysql.connector
import sys
from datetime import datetime, timedelta
from PIL import Image, ImageTk
from config import db_config

user_id = int(sys.argv[1]) if len(sys.argv) > 1 else 1

def get_conn():
    return mysql.connector.connect(**db_config)

def refresh_borrowed():
    for row in borrowed_tree.get_children():
        borrowed_tree.delete(row)
    conn = get_conn()
    cur = conn.cursor()
    cur.execute("SELECT member_id FROM members WHERE user_id=%s", (user_id,))
    member = cur.fetchone()
    has_overdue = False
    if member:
        query = """SELECT book_id, title, author, year, issue_date
                FROM books WHERE issued_to_member_id=%s
                AND book_status='Issued'"""
        cur.execute(query, (member[0],))
        today = datetime.now().date()
        for book in cur.fetchall():
            issue_date_str = 'N/A'
            due_date_str = 'N/A'
            is_overdue = False
            if book[4]:
                issue_date_str = book[4].strftime('%d/%m/%Y')
                due_date = book[4] + timedelta(days=15)
```

```
            due_date_str = due_date.strftime('%d/%m/%Y')
            if due_date < today:
                is_overdue = True
                has_overdue = True

        item_id = borrowed_tree.insert('', tk.END, values=(
            book[0], book[1], book[2], book[3],
            issue_date_str, due_date_str))
        if is_overdue:
            borrowed_tree.item(item_id, tags=('overdue',))

    borrowed_tree.tag_configure('overdue', background='#ffcccc')

    if has_overdue:
        overdue_legend.pack(pady=5)
    else:
        overdue_legend.pack_forget()

    conn.close()

def search_books():
    for row in search_tree.get_children():
        search_tree.delete(row)
    conn = get_conn()
    cur = conn.cursor()
    search_term = '%' + search_entry.get() + '%'

    query = """SELECT DISTINCT title, author, year FROM books
            WHERE (title LIKE %s OR author LIKE %s OR year LIKE %s)
            AND book_status IN ('Returned', 'New')
            AND record_status='Active'"""
    cur.execute(query, (search_term, search_term, search_term))
    books = cur.fetchall()

    for book in books:
        title, author, year = book[0], book[1], book[2]
        cur.execute("""SELECT COUNT(*) FROM books WHERE title=%s
                AND author=%s AND book_status IN ('Returned', 'New')
                AND record_status='Active'""", (title, author))
        available = cur.fetchone()[0]

        if available > 0:
            search_tree.insert('', tk.END, values=(
                title, author, year, available))

    conn.close()
```

```
# Main window
root = tk.Tk()
root.title("Library Management System - Member")
root.geometry("800x600")
root.resizable(False, False)

# Center window
root.update_idletasks()
x = (root.winfo_screenwidth() // 2) - 400
y = (root.winfo_screenheight() // 2) - 300
root.geometry(f'800x600+{x}+{y}')

# Header
header = tk.Frame(root, bg="#4CAF50", height=70)
header.pack(fill=tk.X)

# Member icon
try:
    member_icon_img = Image.open("images/member_portal.png").resize((40, 40))
    member_icon_photo = ImageTk.PhotoImage(member_icon_img)
    tk.Label(header, image=member_icon_photo,
             bg="#4CAF50").place(x=280, y=15)
    header.member_icon = member_icon_photo
except:
    pass

tk.Label(header, text="Member Portal", font=("Arial", 18, "bold"),
         bg="#4CAF50", fg="white").place(x=330, y=20)
tk.Button(header, text="Logout", font=("Arial", 10),
          command=root.destroy).place(x=700, y=20)

# Main container
main = tk.Frame(root, bg="#f5f5f5")
main.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Borrowed books section
borrowed_frame = tk.Frame(main, bg="white", relief=tk.RAISED, bd=1)
borrowed_frame.pack(fill=tk.BOTH, expand=True, pady=(0, 5))

tk.Label(borrowed_frame, text="My Borrowed Books",
         font=("Arial", 14, "bold"), bg="white").pack(pady=10)

cols = ('ID', 'Title', 'Author', 'Year', 'Issue Date', 'Due Date')
borrowed_tree = ttk.Treeview(borrowed_frame, columns=cols,
                             show='headings', height=5)
```

```
widths = [50, 200, 150, 80, 100, 100]
for i, col in enumerate(cols):
    borrowed_tree.heading(col, text=col)
    borrowed_tree.column(col, width=widths[i])
borrowed_tree.pack(pady=5, padx=10, fill=tk.BOTH, expand=True)

# Overdue legend
overdue_legend = tk.Label(borrowed_frame,
    text="⚠ Red highlighted books are overdue. Please contact admin.",
    font=("Arial", 10, "bold"), bg="#ffcccc", fg="#cc0000",
    relief=tk.SOLID, bd=1, padx=10, pady=5)

# Search section
search_frame = tk.Frame(main, bg="white", relief=tk.RAISED, bd=1)
search_frame.pack(fill=tk.BOTH, expand=True)

tk.Label(search_frame, text="Search Available Books",
        font=("Arial", 14, "bold"), bg="white").pack(pady=10)

search_bar = tk.Frame(search_frame, bg="white")
search_bar.pack()
tk.Label(search_bar, text="Search:", bg="white").pack(side=tk.LEFT, padx=5)
search_entry = tk.Entry(search_bar, font=("Arial", 11), width=35)
search_entry.pack(side=tk.LEFT, padx=5)
tk.Button(search_bar, text="Search",
         command=search_books).pack(side=tk.LEFT, padx=5)

search_cols = ('Title', 'Author', 'Year', 'Available Copies')
search_tree = ttk.Treeview(search_frame, columns=search_cols,
                           show='headings', height=8)
search_widths = [280, 200, 80, 110]
for i, col in enumerate(search_cols):
    search_tree.heading(col, text=col)
    search_tree.column(col, width=search_widths[i])
search_tree.pack(pady=10, padx=10, fill=tk.BOTH, expand=True)

tk.Label(search_frame, text="Contact admin to issue books",
        font=("Arial", 10), bg="white", fg="#666").pack(pady=10)

refresh_borrowed()
search_books()

root.mainloop()
```

Add_member.py - Add User Form

Purpose: Create new Admin or Member accounts

Key Features

- User Type dropdown (Member/Admin)
- Right-aligned labels
- Conditional member profile creation

Code

```
# Add Member - Library Management System


import tkinter as tk
from tkinter import messagebox, ttk
import mysql.connector
from config import db_config

def save():
    username = username_entry.get().strip()
    password = password_entry.get().strip()
    name = name_entry.get().strip()
    user_type = user_type_var.get()
    address = address_entry.get().strip()
    email = email_entry.get().strip()
    phone = phone_entry.get().strip()

    if not username or not password or not name:
        messagebox.showerror("Error",
            "Username, Password and Full Name are required fields")
        return

    try:
        conn = mysql.connector.connect(**db_config)
        cur = conn.cursor()

        cur.execute("SELECT username FROM users WHERE username = %s",
                (username,))
        if cur.fetchone():
            messagebox.showerror("Error",
                f"Username '{username}' already exists!\n"
                f"Please choose a different username.")
            conn.close()
            return
```

```
        query = """INSERT INTO users
                (username, password, full_name, user_type, status)
                VALUES (%s, %s, %s, %s, 'active')"""
        cur.execute(query, (username, password, name, user_type.lower()))
        user_id = cur.lastrowid

        if user_type.lower() == 'member':
            query = """INSERT INTO members
                    (user_id, name, address, email, phone)
                    VALUES (%s, %s, %s, %s, %s)"""
            cur.execute(query, (user_id, name, address, email, phone))
            member_id = cur.lastrowid
            success_msg = f"Member added successfully!\nMember ID: {member_id}"
        else:
            success_msg = f"Admin user '{username}' added successfully!"

        conn.commit()
        conn.close()

        messagebox.showinfo("Success", success_msg)
        reset()
    except Exception as e:
        messagebox.showerror("Error", f"Failed to add user:\n{e}")

def reset():
    username_entry.delete(0, tk.END)
    password_entry.delete(0, tk.END)
    name_entry.delete(0, tk.END)
    user_type_var.set("Member")
    address_entry.delete(0, tk.END)
    email_entry.delete(0, tk.END)
    phone_entry.delete(0, tk.END)

# Main window
root = tk.Tk()
root.title("Add User")
root.geometry("800x600")
root.resizable(False, False)

# Form
frame = tk.Frame(root)
frame.place(relx=0.5, rely=0.5, anchor="center")

tk.Label(frame, text="Username: *", anchor="e",
        width=15).grid(row=0, column=0, padx=5, pady=5, sticky="e")
```

```
username_entry = tk.Entry(frame, width=25)
username_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(frame, text="Password: *", anchor="e",
        width=15).grid(row=1, column=0, padx=5, pady=5, sticky="e")
password_entry = tk.Entry(frame, show="*", width=25)
password_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(frame, text="Full Name: *", anchor="e",
        width=15).grid(row=2, column=0, padx=5, pady=5, sticky="e")
name_entry = tk.Entry(frame, width=25)
name_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(frame, text="User Type:", anchor="e",
        width=15).grid(row=3, column=0, padx=5, pady=5, sticky="e")
user_type_var = tk.StringVar(value="Member")
user_type_dropdown = ttk.Combobox(frame, textvariable=user_type_var,
    values=["Member", "Admin"], state="readonly", width=22)
user_type_dropdown.grid(row=3, column=1, padx=5, pady=5)

tk.Label(frame, text="Address:", anchor="e",
        width=15).grid(row=4, column=0, padx=5, pady=5, sticky="e")
address_entry = tk.Entry(frame, width=25)
address_entry.grid(row=4, column=1, padx=5, pady=5)

tk.Label(frame, text="Email:", anchor="e",
        width=15).grid(row=5, column=0, padx=5, pady=5, sticky="e")
email_entry = tk.Entry(frame, width=25)
email_entry.grid(row=5, column=1, padx=5, pady=5)

tk.Label(frame, text="Phone:", anchor="e",
        width=15).grid(row=6, column=0, padx=5, pady=5, sticky="e")
phone_entry = tk.Entry(frame, width=25)
phone_entry.grid(row=6, column=1, padx=5, pady=5)

# Buttons
button_frame = tk.Frame(frame)
button_frame.grid(row=7, column=0, columnspan=2, pady=10)
tk.Button(button_frame, text="Save", command=save,
         width=12).pack(side=tk.LEFT, padx=5)
tk.Button(button_frame, text="Reset", command=reset,
         width=12).pack(side=tk.LEFT, padx=5)

root.mainloop()
```

add_book.py - Add Book

Purpose: Add books to catalog (1-5 copies)

Key Validation:

- All fields required
- Year and copies must be integers
- Copies range: 1-5
- Duplicate prevention (title + author)

Code

```
# Add Book - Library Management System

import tkinter as tk
from tkinter import messagebox
import mysql.connector
import sys
from config import db_config

admin_user_id = int(sys.argv[1]) if len(sys.argv) > 1 else None

def save():
    title = title_entry.get().strip()
    author = author_entry.get().strip()
    isbn = isbn_entry.get().strip()
    year = year_entry.get().strip()
    copies = copies_entry.get().strip()

    if not all([title, author, isbn, year, copies]):
        messagebox.showerror("Error", "All fields are required!")
        return

    try:
        year = int(year)
        copies = int(copies)
        if copies < 1:
            messagebox.showerror("Error",
                "Number of copies must be at least 1")
            return
        if copies > 5:
            messagebox.showerror("Error",
                "Number of copies cannot be more than 5")
            return
    except ValueError:
```

```
        messagebox.showerror("Error",
            "Year and copies must be valid numbers")
        return

    try:
        conn = mysql.connector.connect(**db_config)
        cur = conn.cursor()

        cur.execute("""SELECT title, author FROM books
                    WHERE title = %s AND author = %s LIMIT 1""",
                (title, author))
        existing_book = cur.fetchone()
        if existing_book:
            messagebox.showerror("Error",
                f"Book '{title}' by {author} already exists!\n"
                f"Use a different title or check existing copies.")
            conn.close()
            return

        for i in range(1, copies + 1):
            query = """INSERT INTO books
                    (title, author, isbn, year, copy_number,
                     book_status, record_status)
                    VALUES (%s, %s, %s, %s, %s, 'New', 'Active')"""
            cur.execute(query, (title, author, isbn, year, i))
            book_id = cur.lastrowid

            query = """INSERT INTO transactions
                    (book_id, member_id, admin_user_id, action, notes)
                    VALUES (%s, NULL, %s, 'Add', %s)"""
            cur.execute(query, (book_id, admin_user_id,
                        f"{title} by {author} (Copy {i})"))

        conn.commit()
        conn.close()

        copy_text = "y" if copies == 1 else "ies"
        messagebox.showinfo("Success",
            f"{copies} cop{copy_text} of '{title}' added successfully!")
        reset()
    except Exception as e:
        messagebox.showerror("Error", f"Failed to add book:\n{e}")

def reset():
    title_entry.delete(0, tk.END)
    author_entry.delete(0, tk.END)
```

```
    isbn_entry.delete(0, tk.END)
    year_entry.delete(0, tk.END)
    copies_entry.delete(0, tk.END)

# Main window
root = tk.Tk()
root.title("Add Book")
root.geometry("800x600")
root.resizable(False, False)

# Form
frame = tk.Frame(root)
frame.place(relx=0.5, rely=0.5, anchor="center")

tk.Label(frame, text="Book Title: *", anchor="e",
        width=20).grid(row=0, column=0, padx=5, pady=5, sticky="e")
title_entry = tk.Entry(frame, width=30)
title_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(frame, text="Author: *", anchor="e",
        width=20).grid(row=1, column=0, padx=5, pady=5, sticky="e")
author_entry = tk.Entry(frame, width=30)
author_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(frame, text="ISBN Number: *", anchor="e",
        width=20).grid(row=2, column=0, padx=5, pady=5, sticky="e")
isbn_entry = tk.Entry(frame, width=30)
isbn_entry.grid(row=2, column=1, padx=5, pady=5)

tk.Label(frame, text="Publication Year: *", anchor="e",
        width=20).grid(row=3, column=0, padx=5, pady=5, sticky="e")
year_entry = tk.Entry(frame, width=30)
year_entry.grid(row=3, column=1, padx=5, pady=5)

tk.Label(frame, text="Number of Copies: *", anchor="e",
        width=20).grid(row=4, column=0, padx=5, pady=5, sticky="e")
copies_entry = tk.Entry(frame, width=30)
copies_entry.grid(row=4, column=1, padx=5, pady=5)

# Buttons
button_frame = tk.Frame(frame)
button_frame.grid(row=5, column=0, columnspan=2, pady=10)
tk.Button(button_frame, text="Save", command=save,
         width=12).pack(side=tk.LEFT, padx=5)
tk.Button(button_frame, text="Reset", command=reset,
         width=12).pack(side=tk.LEFT, padx=5)
```

```
root.mainloop()
```

Configuration Files

config_template.py

```
# Database Configuration Template
# Copy this file as 'config.py' and update with your MySQL credentials
# Database connection settings

db_config = {

    'host': 'localhost',        # MySQL server address (usually localhost)
    'user': 'root',             # MySQL username
    'password': 'your_password', # MYSQL PASSWORD HERE
    'database': 'library_management_system'  # Database name
}
# Instructions:
# 1. Copy this file and rename it to 'config.py'
# 2. Replace 'your_password' with your actual MySQL password
# 3. Save the file
# 4. All Python files will automatically use these settings
```

requirements.txt

```
mysql-connector-python==8.0.33

Pillow==9.5.0
```

Database Setup Script

setup_database_final.py

Purpose: Create database and populate with initial data

```
# Database Setup - Library Management System

import mysql.connector
from mysql.connector import Error
from datetime import datetime, timedelta

def create_connection(host, user, password):
    try:
        connection = mysql.connector.connect(
            host=host, user=user, password=password,
```

```
            database='library_management_system')
        if connection.is_connected():
            print("✓ Successfully connected to MySQL")
            return connection
    except Error as e:
        print(f"✗ Error: {e}")
        return None

def drop_all_tables(connection):
    try:
        cursor = connection.cursor()
        print("\nDropping tables...")
        cursor.execute("DROP TABLE IF EXISTS transactions")
        cursor.execute("DROP TABLE IF EXISTS books")
        cursor.execute("DROP TABLE IF EXISTS members")
        cursor.execute("DROP TABLE IF EXISTS users")
        connection.commit()
        cursor.close()
        print("✓ Tables dropped")
    except Error as e:
        print(f"✗ Error: {e}")

def create_tables(connection):
    try:
        cursor = connection.cursor()
        print("\nCreating tables...")

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                user_id INT AUTO_INCREMENT PRIMARY KEY,
                username VARCHAR(50) UNIQUE NOT NULL,
                password VARCHAR(100) NOT NULL,
                full_name VARCHAR(100) NOT NULL,
                user_type VARCHAR(20) NOT NULL,
                status VARCHAR(20) DEFAULT 'active',
                created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        """)

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS members (
                member_id INT AUTO_INCREMENT PRIMARY KEY,
                user_id INT NOT NULL,
                name VARCHAR(100) NOT NULL,
                address VARCHAR(200),
```

```
        email VARCHAR(100),
        phone VARCHAR(15),
        created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON DELETE CASCADE
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS books (
        book_id INT AUTO_INCREMENT PRIMARY KEY,
        title VARCHAR(200) NOT NULL,
        author VARCHAR(100) NOT NULL,
        isbn VARCHAR(20) DEFAULT NULL,
        year INT NOT NULL,
        copy_number INT DEFAULT 1,
        book_status VARCHAR(20) DEFAULT 'Returned',
        record_status VARCHAR(20) DEFAULT 'Active',
        issued_to_member_id INT DEFAULT NULL,
        issue_date DATE DEFAULT NULL,
        created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (issued_to_member_id)
        REFERENCES members(member_id) ON DELETE SET NULL
    )
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        transaction_id INT AUTO_INCREMENT PRIMARY KEY,
        book_id INT NOT NULL,
        member_id INT DEFAULT NULL,
        admin_user_id INT DEFAULT NULL,
        action VARCHAR(20) NOT NULL,
        transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        notes VARCHAR(200) DEFAULT NULL,
        FOREIGN KEY (book_id) REFERENCES books(book_id)
        ON DELETE CASCADE,
        FOREIGN KEY (member_id) REFERENCES members(member_id)
        ON DELETE SET NULL,
        FOREIGN KEY (admin_user_id) REFERENCES users(user_id)
        ON DELETE SET NULL
    )
""")

connection.commit()
cursor.close()
```

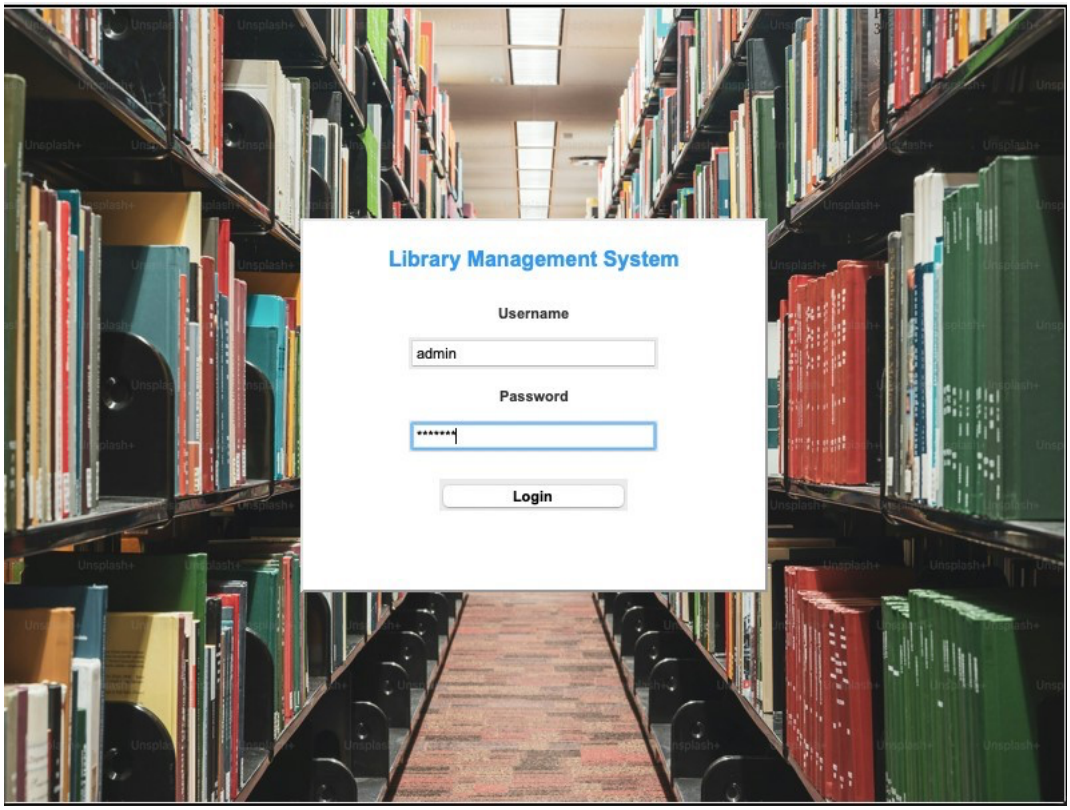
```
        print("✓ Tables created")
    except Error as e:
        print(f"✗ Error: {e}")

def insert_sample_data(connection):
    try:
        cursor = connection.cursor()

        print("\nInserting users...")
        users = [
            ('admin', 'admin123', 'Admin User', 'admin', 'active'),
            ('priya', 'priya123', 'Priya Sharma', 'member', 'active'),
            ('rahul', 'rahul123', 'Rahul Kumar', 'member', 'active')
        ]
        query = """INSERT INTO users
                   (username, password, full_name, user_type, status)
                   VALUES (%s, %s, %s, %s, %s)"""
        cursor.executemany(query, users)

        print("Inserting members...")
        members = [
            (2, 'Priya Sharma', '123 MG Road, Mumbai',
             'priya@email.com', '9876543210'),
            (3, 'Rahul Kumar', '45 Park Street, Delhi',
             'rahul@email.com', '9876543211')
        ]
        query = """INSERT INTO members
                   (user_id, name, address, email, phone)
                   VALUES (%s, %s, %s, %s, %s)"""
        cursor.executemany(query, members)

        print("Inserting books...")
        today = datetime.now().date()
        books = [
            ('The God of Small Things', 'Arundhati Roy',
             '978-0143028574', 1997, 1, 'Returned', 'Active', None, None),
            ('The God of Small Things', 'Arundhati Roy',
             '978-0143028574', 1997, 2, 'Returned', 'Active', None, None),
            ('Midnight\'s Children', 'Salman Rushdie',
             '978-0099511892', 1981, 1, 'Returned', 'Active', None, None),
            ('The White Tiger', 'Aravind Adiga',
             '978-1416562603', 2008, 1, 'Issued', 'Active', 1, today),
            ('A Suitable Boy', 'Vikram Seth',
             '978-0143065937', 1993, 1, 'Returned', 'Active', None, None),
            ('The Namesake', 'Jhumpa Lahiri',
             '978-0618485222', 2003, 1, 'Returned', 'Active', None, None),
```

```
    ('Train to Pakistan', 'Khushwant Singh',
     '978-0143065883', 1956, 1, 'Issued', 'Active', 2,
     today - timedelta(days=5)),
    ('Train to Pakistan', 'Khushwant Singh',
     '978-0143065883', 1956, 2, 'Returned', 'Active', None, None),
    ('The Guide', 'R.K. Narayan',
     '978-0143039648', 1958, 1, 'New', 'Active', None, None),
    ('Malgudi Days', 'R.K. Narayan',
     '978-0143039655', 1943, 1, 'New', 'Active', None, None),
    ('2 States', 'Chetan Bhagat',
     '978-8129115300', 2009, 1, 'New', 'Active', None, None),
    ('2 States', 'Chetan Bhagat',
     '978-8129115300', 2009, 2, 'New', 'Active', None, None),
    ('Five Point Someone', 'Chetan Bhagat',
     '978-8129104177', 2004, 1, 'New', 'Active', None, None),
    ('The Immortals of Meluha', 'Amish Tripathi',
     '978-9380658742', 2010, 1, 'New', 'Active', None, None),
    ('The Secret of the Nagas', 'Amish Tripathi',
     '978-9380658698', 2011, 1, 'New', 'Active', None, None),
    ('Wings of Fire', 'A.P.J. Abdul Kalam',
     '978-8173711466', 1999, 1, 'New', 'Active', None, None),
    ('Wings of Fire', 'A.P.J. Abdul Kalam',
     '978-8173711466', 1999, 2, 'New', 'Active', None, None),
    ('My Experiments with Truth', 'Mahatma Gandhi',
     '978-0486245935', 1927, 1, 'New', 'Active', None, None),
    ('Discovery of India', 'Jawaharlal Nehru',
     '978-0143031055', 1946, 1, 'New', 'Active', None, None),
    ('The Blue Umbrella', 'Ruskin Bond',
     '978-0143333654', 1980, 1, 'New', 'Active', None, None),
    ('The Room on the Roof', 'Ruskin Bond',
     '978-0143333654', 1956, 1, 'New', 'Active', None, None),
    ('The Krishna Key', 'Ashwin Sanghi',
     '978-9382618287', 2012, 1, 'New', 'Active', None, None),
    ('Gitanjali', 'Rabindranath Tagore',
     '978-8129116673', 1910, 1, 'New', 'Active', None, None),
    ('The Bhagavad Gita', 'Eknath Easwaran',
     '978-1586380199', 1985, 1, 'New', 'Active', None, None),
    ('Shantaram', 'Gregory David Roberts',
     '978-0312330538', 2003, 1, 'New', 'Active', None, None)
]
query = """INSERT INTO books
          (title, author, isbn, year, copy_number, book_status,
           record_status, issued_to_member_id, issue_date)
          VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"""
cursor.executemany(query, books)
```

```
        connection.commit()
        cursor.close()
        print(f"✓ Data inserted ({len(books)} books)")
    except Error as e:
        print(f"✗ Error: {e}")

def main():
    print("="*60)
    print("Library Management System - Database Setup")
    print("="*60)

    host = input("Host (localhost): ").strip() or "localhost"
    user = input("Username (root): ").strip() or "root"
    password = input("Password: ").strip()

    connection = create_connection(host, user, password)
    if connection:
        drop_all_tables(connection)
        create_tables(connection)
        insert_sample_data(connection)

        print("\n" + "="*60)
        print("✓ Setup complete!")
        print("="*60)
        print("\nLogin Credentials:")
        print("Admin: admin/admin123")
        print("Members: priya/priya123, rahul/rahul123")
        print()
        connection.close()

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

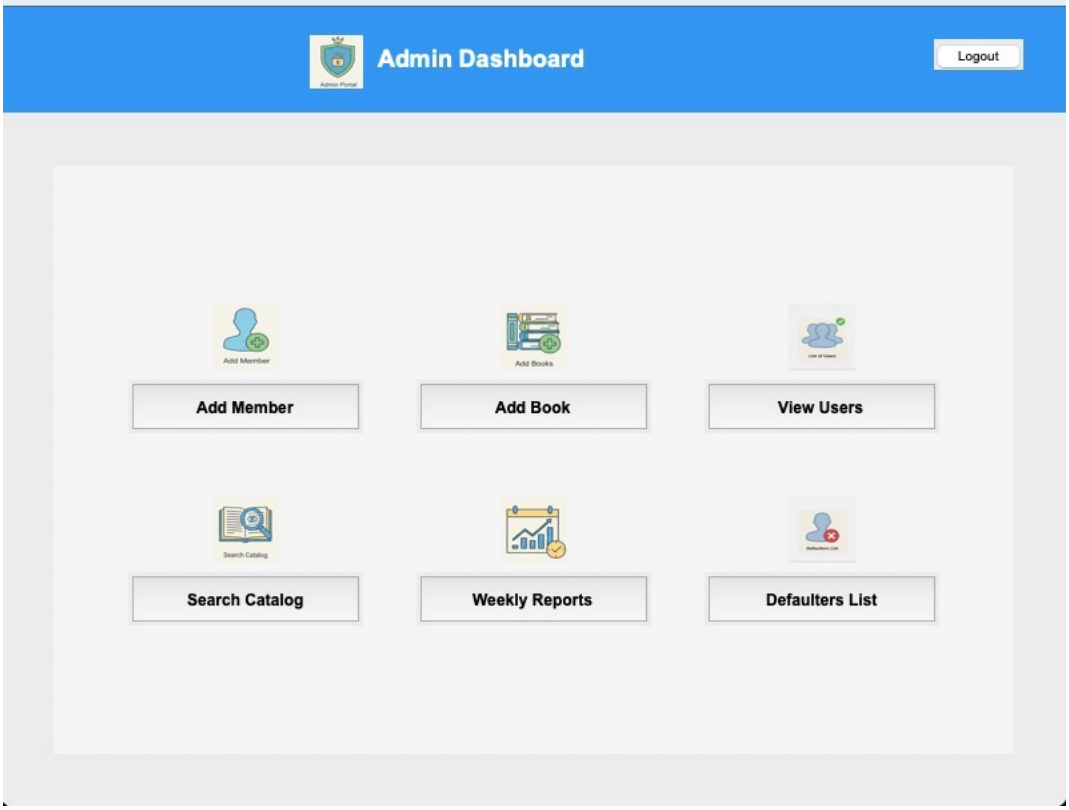
Sample Screenshots

Login Screen



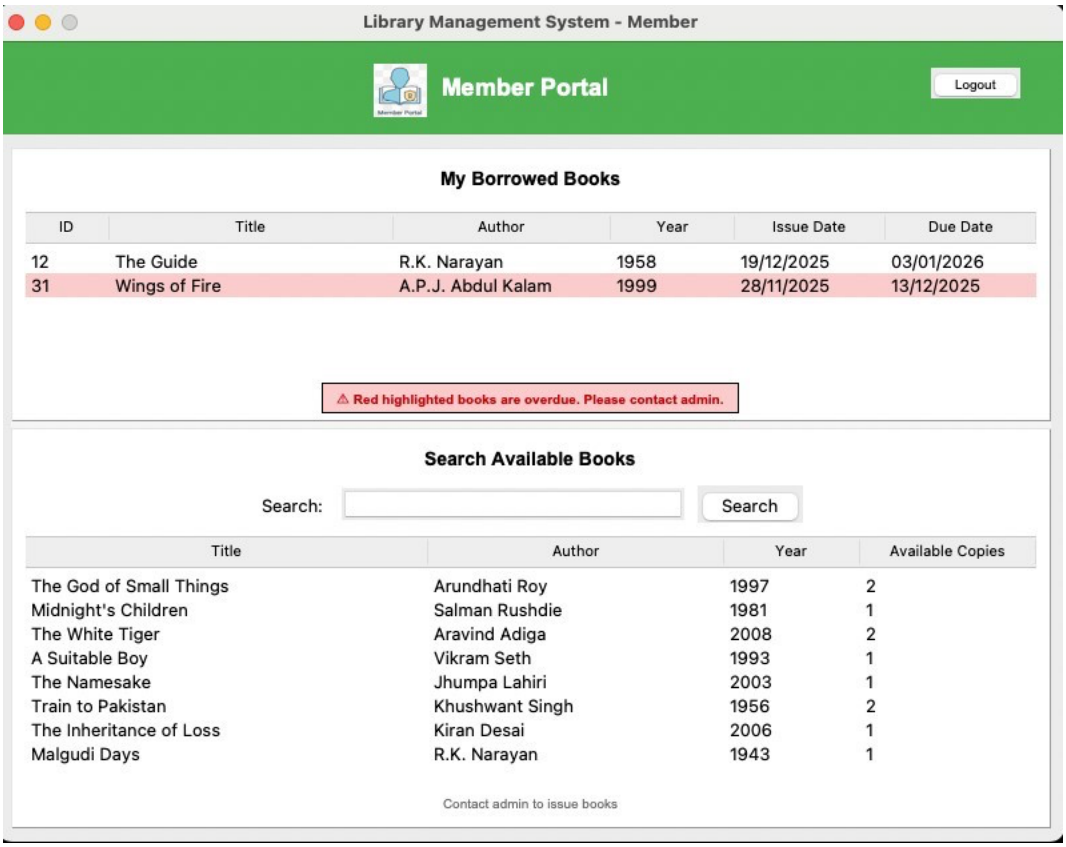
- Library background image (800x600 pixels)
- Centered white login box
- Username and Password input fields
- Login button
- Title: "Library Management System"

Admin Dashboard



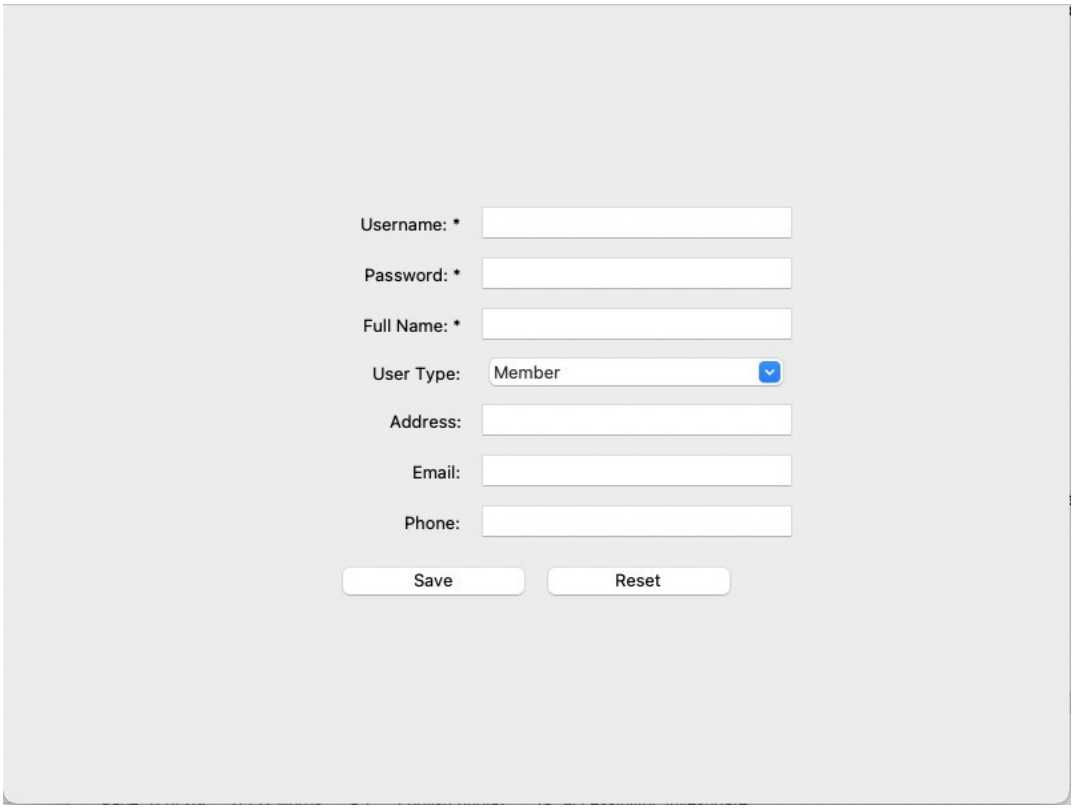
- Blue header with admin portal icon (40x40) next to "Admin Dashboard" title
- Logout button positioned in top-right corner - 6 buttons arranged in 3x2 grid layout
- Row 1: Add Member | Add Book | View Users
- Row 2: Search Catalog | Weekly Reports | Defaulters List
- Each button displays a 50x50 icon above the button text

Member Portal



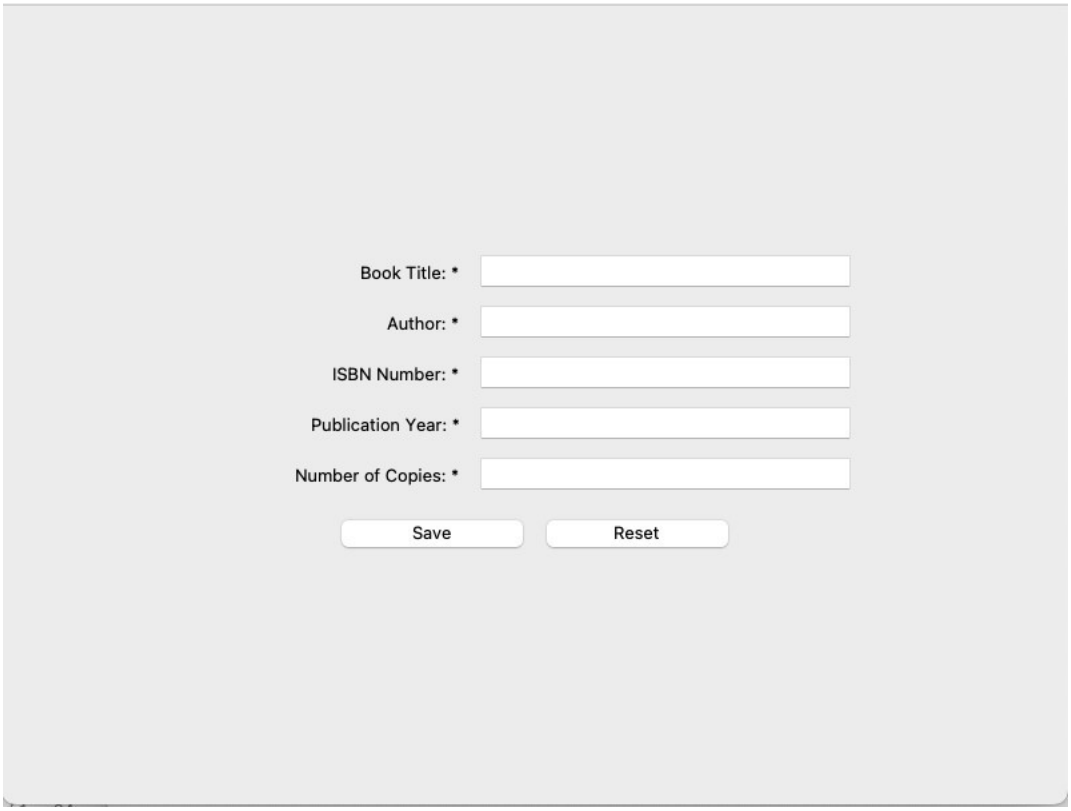
- Green header with member portal icon (40x40) next to "Member Portal" title
- Logout button in top-right corner
- My Borrowed Books (top section): Shows books currently borrowed with ID, Title, Author, Year, Issue Date, and Due Date
- Search Available Books (bottom section): Allows members to search the catalog
- Note at bottom: "Contact admin to issue books"

Add Member Form



- Right-aligned labels (15 characters wide)
- Form fields:
    - o Username: * (required)
    - o Password: * (required, masked input)
    - o Full Name: * (required)
    - o User Type: Dropdown with "Member" and "Admin" options (default: Member)
    - o Address: (optional) o Email: (optional) o Phone: (optional)
- Centered Save and Reset buttons

Add Book Form



- Right-aligned labels (20 characters wide)
- All fields marked with asterisk (*) indicating required:
    - Book Title: *
    - Author: *
    - ISBN Number: *
    - Publication Year: *
    - Number of Copies: * (accepts 1-5)
- Centered Save and Reset buttons

Application Limitations

Security Limitations

1. Password Storage

- **Issue**: Passwords stored in plain text in database
- **Risk**: Database breach would expose all passwords

2. No Session Management

- **Issue**: User credentials not validated beyond login. No timeout or auto-logout feature
- **Risk**: If application left open, anyone can access it using the logged in userID

Functional Limitations

1. Book Copy Limit

- Restriction: Maximum 5 copies per book
- Reason: UI design and validation logic constraint

2. Borrowing Limit

- Fixed: Maximum 3 books per member
- Hardcoded: Not configurable without code changes

3. Due Date Period

- Fixed: Exactly 15 days from issue date
- Hardcoded: Not configurable without code changes
- Missing: Different due date periods for different book types (reference, fiction)

4. No Fine Calculation

- Missing: Automatic fine calculation for overdue books
- Manual fine tracking required

Data Limitations

1. No Book Categories/Genres

- Missing: Classification system (Fiction, Non-Fiction, Reference, magazines etc.)
- Cannot filter or search by category

2. Limited Book Information

- Basic catalog only
- Missing: Publisher, edition, language, page count, summary

Reporting Limitations

1. Limited Report Type

Available: Only weekly transactions and defaulters list **Following are not available**:

- Monthly/yearly reports
- Most borrowed books - Most active members
- Book utilization statistics
- Genre-wise reports

2. No Export Functionality

- **Missing**: Cannot export reports to PDF, Excel, or CSV
- Manual data copying required for external use

3. No Charts/Graphs

- **Missing**: Visual representation of data
- Difficult to spot trends for data analysis

Scope for Improvement

1. Password Hashing

Feature: Password hashing using bcrypt library

Benefit: Protect user credentials in the database even if database compromised

2. Session Management

Feature: Auto-logout after inactivity (15-30 minutes)

Benefit: Prevent unauthorized access from unattended terminals

3.Email Notifications

Features:

- Due date reminder (2 days before)
- Overdue notification
- Book availability notification (for reservations)

Benefit : Serves as a reminder to return books borrowed

Note: smtplib can be used for email sending

4. SMS Notifications

Feature: Using Twilio API or similar service , SMS can be sent for overdue books **Benefit**: Higher reach than email

5. Advanced Search

Feature : Following filters can be included for search:

- By ISBN
- By year range (e.g., 1990-2000)
- By category/genre (requires new field)
- By availability status

Benefit: Faster book discovery

6. Graphical dashboard

Feature : Visual representation of reports like the following:

- Books issued vs returned (line chart)
- Category-wise distribution (pie chart)
- Monthly trends (bar chart)
- Overdue books over time (area chart)

Benefit : Provides a better view for data analysis

Bibliography / References

Books

1. Computer Science with Python" (Class XII) by Sumita Arora
2. "Learning MySQL" (2nd Edition) by Seyed M.M. Tahaghoghi, Hugh E. Williams
3. "Tkinter GUI Application Development" by Bhaskar Chaudhary

Websites

1. **Python and SQL programming** :

   Website : https://docs.python.org/3/

2. **Tkinter Documentation**

   Website: https://docs.python.org/3/library/tkinter.html

3. **MySQL 8.0 Reference Manual**

   Website: https://dev.mysql.com/doc/refman/8.0/en/ mysql-connector-**python Documentation**

   Website: https://dev.mysql.com/doc/connector-python/en/