[2024-2025]

LIBRARY MANAGEMENT

With Python & MySQL

Class - XII

BY-

Yuvraj Singh

Madhav Gupta

Ashwinder

TO-

Benoie Mathew

CERTIFICATE

This is to certify that *Yuvraj Singh* of class XII With Team Members *Ashwinder and Madhav Gupta* has successfully completed the project of Library Management System according to CBSE guidelines under my guidance and supervision during the academic year 2024-2025

Mr. Benoie Mathew

Head of the department

(Department of Computer Science)

External Examiner

ACKNOWLEDGEMENT

I would like to express my special thanks to my Informatics Practice teacher **Mr. Benoie Mathew**, who gave me the golden opportunity to do this wonderful project of Informatics Practices on the topic **"Library Management"** and helped me a lot in completing this project. I came to know so many things and I'm genuinely thankful to him. I am also deeply grateful to my principal, **Ms. Nidhi Jain**, for her constant encouragement and for allowing me to use the school's advanced computer labs, which greatly facilitated my work on this project. Secondly, I would like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

Board Roll No:

- 13681658 (*Yuvraj Singh*)
- 13680719 (*Ashwinder*)
- 13681694 (*Madhav Gupta*)

INDEX

Introduction

- PROBLEM DEFINITION

Libraries are essential institutions that provide access to knowledge and resources, serving as a cornerstone for education and information dissemination. However, managing library operations effectively remains a significant challenge, especially in large-scale or resource-constrained environments.

Traditional methods of maintaining library records—such as manually tracking books, borrowers, and transactions—are often labor-intensive and prone to errors. These inefficiencies can result in mismanagement of resources, difficulties in locating specific books, and delays in tracking borrowed items or overdue returns. Moreover, without a proper system in place, maintaining transparency and accountability becomes a daunting task.

As the demand for library services grows, manual processes struggle to keep up. Librarians may face difficulties such as:

- Inconsistent or inaccurate record-keeping.
- Limited visibility into the availability of books and overdue returns.
- Challenges in managing a growing inventory and user base.
- Lack of security in accessing sensitive information, leading to potential misuse or data loss.

Furthermore, without a structured system, borrowers may face inconvenience in accessing resources or understanding their responsibilities, such as return deadlines or overdue fines.

These issues highlight the urgent need for a streamlined approach to library management, one that minimizes errors, optimizes resource utilization, and enhances both librarian and user experience.

- PYTHON INTRODUCTION

PYTHON HISTORY: Python was conceived in the late 1980s by Guido Van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC programming language, which was inspired by SETL, capable of exception handling (from the start plus new capabilities in Python 3.11) and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "benevolent dictator for life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member Steering Council to lead the project.

PYTHON: Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect-oriented programming (including meta-programming and meta-objects). Many other paradigms aresupported via extensions, including design by contract. and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It uses dynamic name resolution (late binding), which binds method and variable names during program execution.

- SQL INTRODUCTION

SQL HISTORY: SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce after learning about the relational model from Edgar F. Codd in the early 1970s. This version, initially called SEQUEL (Structured English Query Language), was designed to manipulate and retrieve data stored in IBM's original quasirelational database management system, System R, which a group at IBM San Jose Research Laboratory had developed during the 1970s.

Chamberlin and Boyce's first attempt at a relational database language was SQUARE (Specifying Queries in A Relational Environment), but it was difficult to use due to subscript/superscript notation. After moving to the San Jose Research Laboratory in 1973, theybegan work on a sequel to SQUARE. The name SEQUEL was later changed to SQL (dropping the vowels) because "SEQUEL" was a trademark of the UK-based Hawker Siddeley Dynamics Engineering Limited company. The label SQL later became the acronym for Structured Query Language.

SQL : **Structured Query Language**, abbreviated as **SQL** is a domain-specific language usedin programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

SQL offers two main advantages over older read-write APIs such as ISAM or VSAM. Firstly, it introduced the concept of accessing many records with one single command. Secondly, it eliminates the need to specify how to reach a record, e.g. with or without an index.

ABOUT PROJECT

This is a **Library Management System** designed to assist librarians in managing library operations with greater efficiency and ease. The system is tailored to function like a specialized operating system for library management, optimized to run on low hardware requirements compared to other solutions.

KEY FEATURES:

1. **Security**:
    - Login authentication with password protection to ensure data safety.
    - A history log to keep track of activities for transparency and monitoring.
2. **Core Functionalities**:
    - **Book Management**: Add, search, issue, return, and remove books.
    - **Inventory Control**: Manage book copies and view all books in the system.
    - **Transaction Tracking**: View books that have not been returned and access fine details for overdue returns.
3. **User Management**:
    - Search, add, and delete borrowers and librarians efficiently.

This system is a comprehensive solution for libraries to streamline their daily operations, ensure data integrity, and provide a user-friendly interface for librarians to manage resources effectively.

SYSTEM REQUIREMENTS

- SOFTWARE REQUIREMENTS
    - **Operating System:**
        - Windows, macOS, or Linux (any modern version).
    - **Python Version:**
        - Python 3.8 or above.
    - **Required Python Libraries:**
        - tkinter: For building GUI applications.
        - mysql-connector-python: To connect and interact with the MySQL database.
    - **Database:**
        - MySQL Server (e.g., MySQL 8.0 or higher).
        - Properly configured MySQL user with access rights (default user in code: root, password: root).
- HARDWARE REQUIREMENTS
    - **Processor:**
        - Minimum: Dual-core processor.
        - Recommended: Quad-core or better for smooth multitasking.
    - **RAM:**
        - Minimum: 4 GB.
        - Recommended: 8 GB or more.
    - **Storage:**
        - Minimum: 500 MB for Python, MySQL, and the project files.
        - Additional storage for the database depending on the data size.
    - **Display:**
        - Resolution: 1024x768 or higher (required for tkinter GUIs).

TABLE DESIGN

- BOOK TABLE

```
mysql> desc book;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| Book_ID | int         | NO   | PRI | NULL    | auto_increment |
| Title   | varchar(255) | NO   | UNI | NULL    |                |
| Genre   | varchar(20)  | NO   |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

- BOOK AUTHORS TABLE

```
mysql> desc book_authors;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| Book_ID     | int          | NO   | PRI | NULL    |       |
| Author_Name | varchar(255) | NO   | PRI | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

- BOOK COPIES TABLE

```
mysql> desc book_copies;
+--------------+------+------+-----+---------+-------+
| Field        | Type | Null | Key | Default | Extra |
+--------------+------+------+-----+---------+-------+
| Book_ID      | int  | NO   | PRI | NULL    |       |
| No_Of_Copies | int  | NO   |     | NULL    |       |
+--------------+------+------+-----+---------+-------+
```

- BOOK LOANS TABLE

```
mysql> desc book_loans;
+----------+------+------+-----+---------+-------+
| Field    | Type | Null | Key | Default | Extra |
+----------+------+------+-----+---------+-------+
| Book_ID  | int  | NO   | PRI | NULL    |       |
| Card_No  | int  | NO   | PRI | NULL    |       |
| Date_Out | date | NO   | PRI | NULL    |       |
| Due_Date | date | YES  |     | NULL    |       |
+----------+------+------+-----+---------+-------+
```

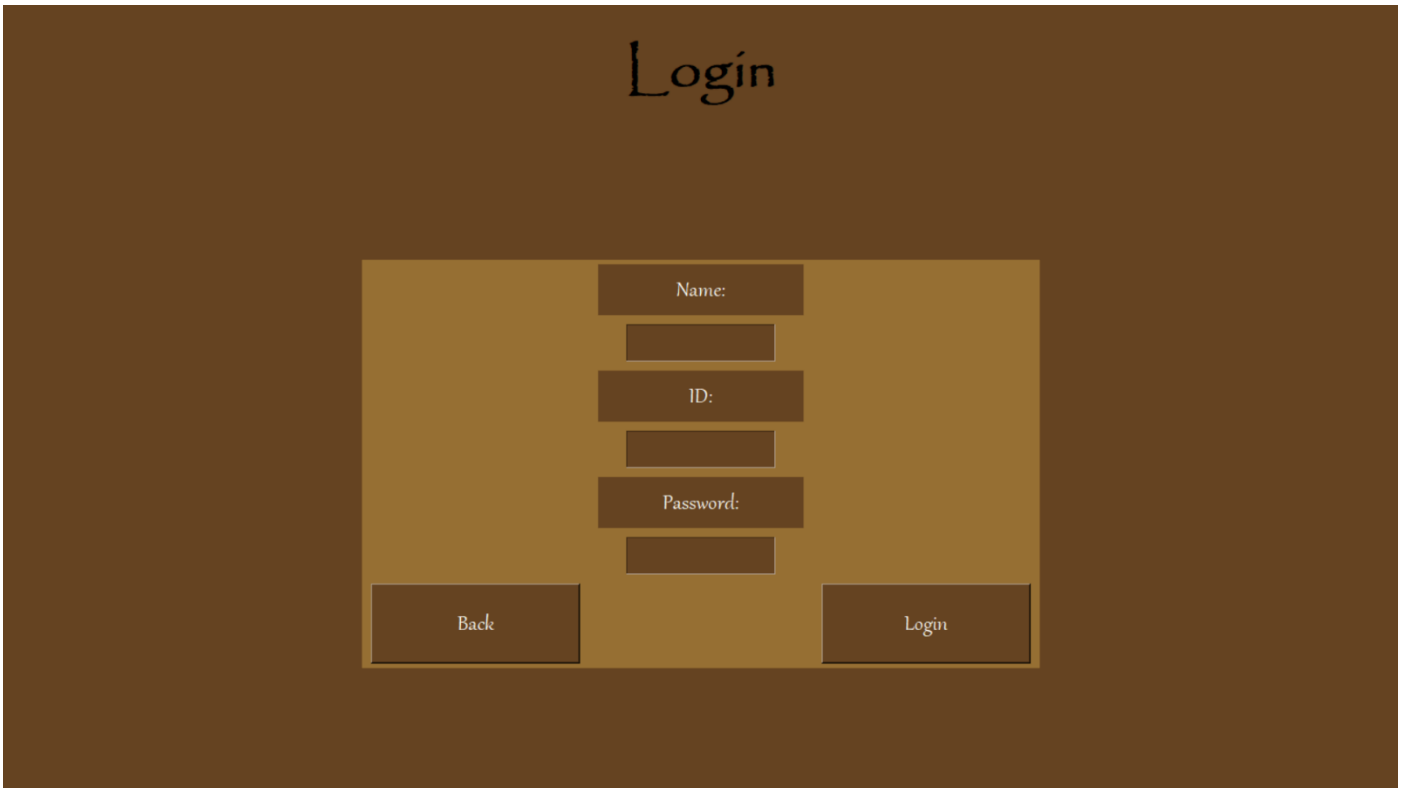
- BORROWER TABLE

```
mysql> desc borrower;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| Card_No | int         | NO   | PRI | NULL    | auto_increment |
| Name    | varchar(55) | NO   |     | NULL    |                |
| Address | text        | YES  |     | NULL    |                |
| Email   | varchar(255)| YES  | UNI | NULL    |                |
| Phone   | varchar(10) | NO   | UNI | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

- LIBRARIAN TABLE

```
mysql> desc librarian;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| Librarian_ID| int         | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(255)| NO   |     | NULL    |                |
| Address     | text        | YES  |     | NULL    |                |
| Email       | varchar(255)| YES  | UNI | NULL    |                |
| Phone       | varchar(15) | NO   | UNI | NULL    |                |
| Password    | varchar(255)| NO   |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

PREVIEW OF PROGRAM

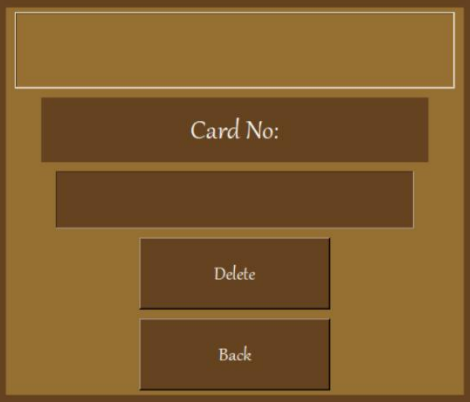

Login
Name:
ID:
Password:
Back
Login


Librarian Window
Search Borrower
View All Books
Search Book
Books Not Returned
Search Librarian
Add Borrower
Add Book
Issue Book
Manage Book Copies
Add Librarian
Delete Borrower
Remove Book
Return Book
View Fine Details
Delete Librarian
Back

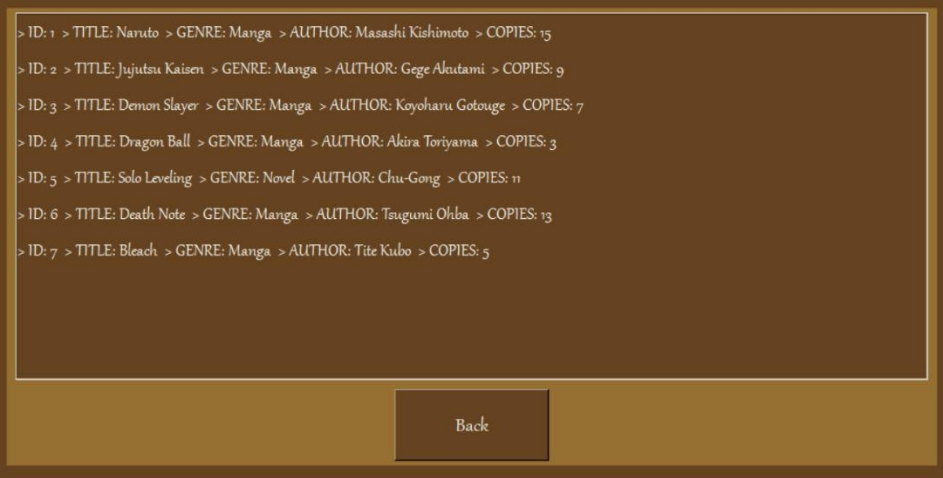

Search Borrower Window
Borrower Name:
Search
Back


Add New Borrower
Name:
Address:
Phone No.:
Email:
Back
Sign Up

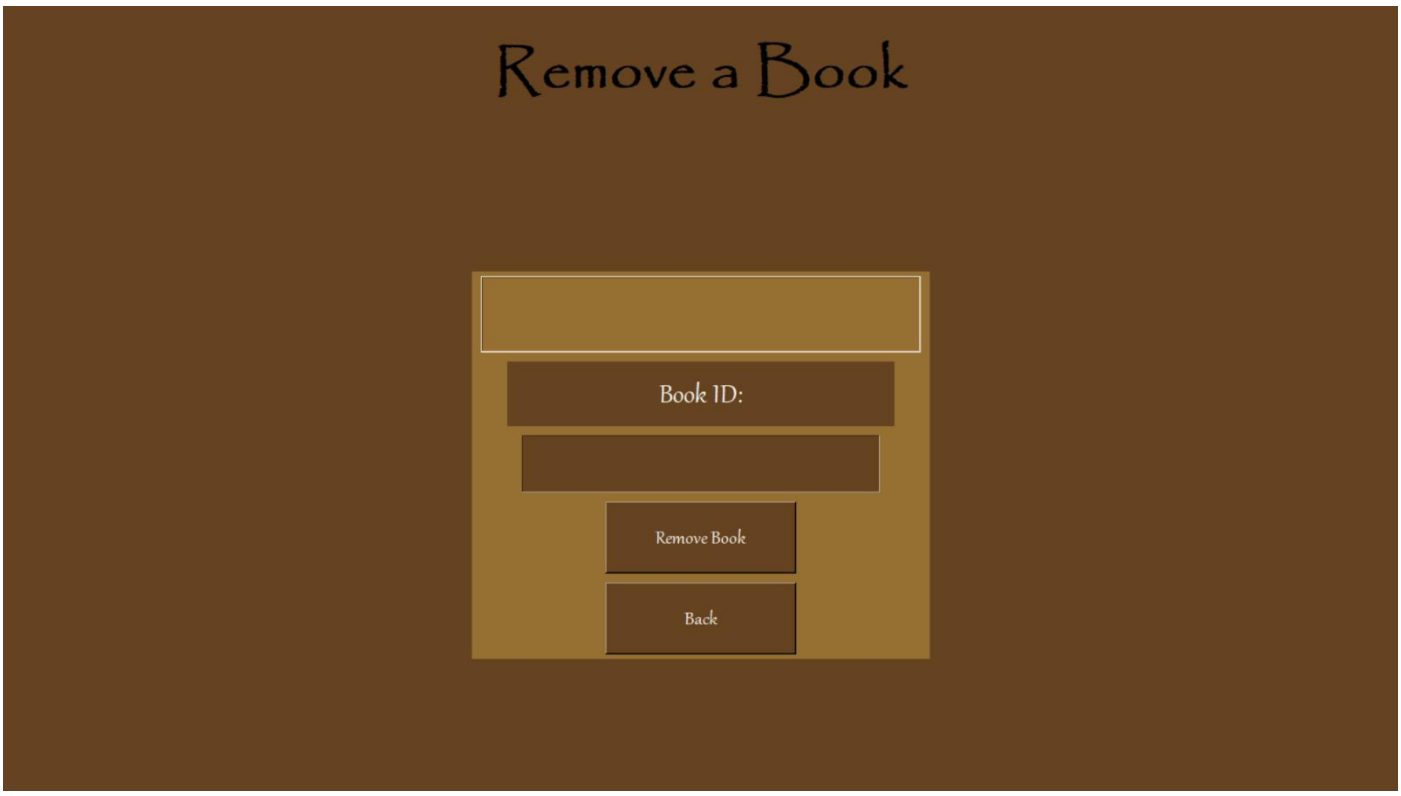
Delete Borrower Window



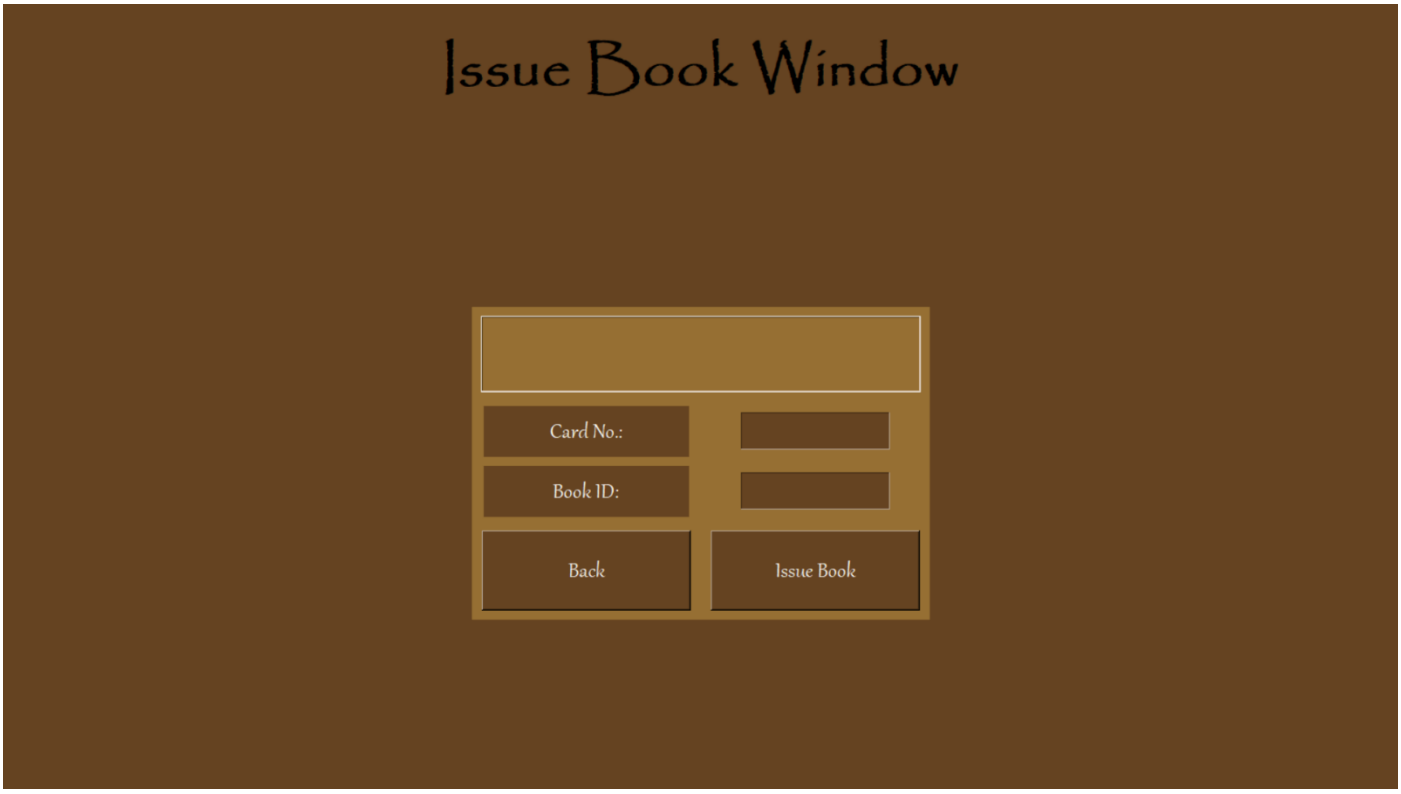
View All Books

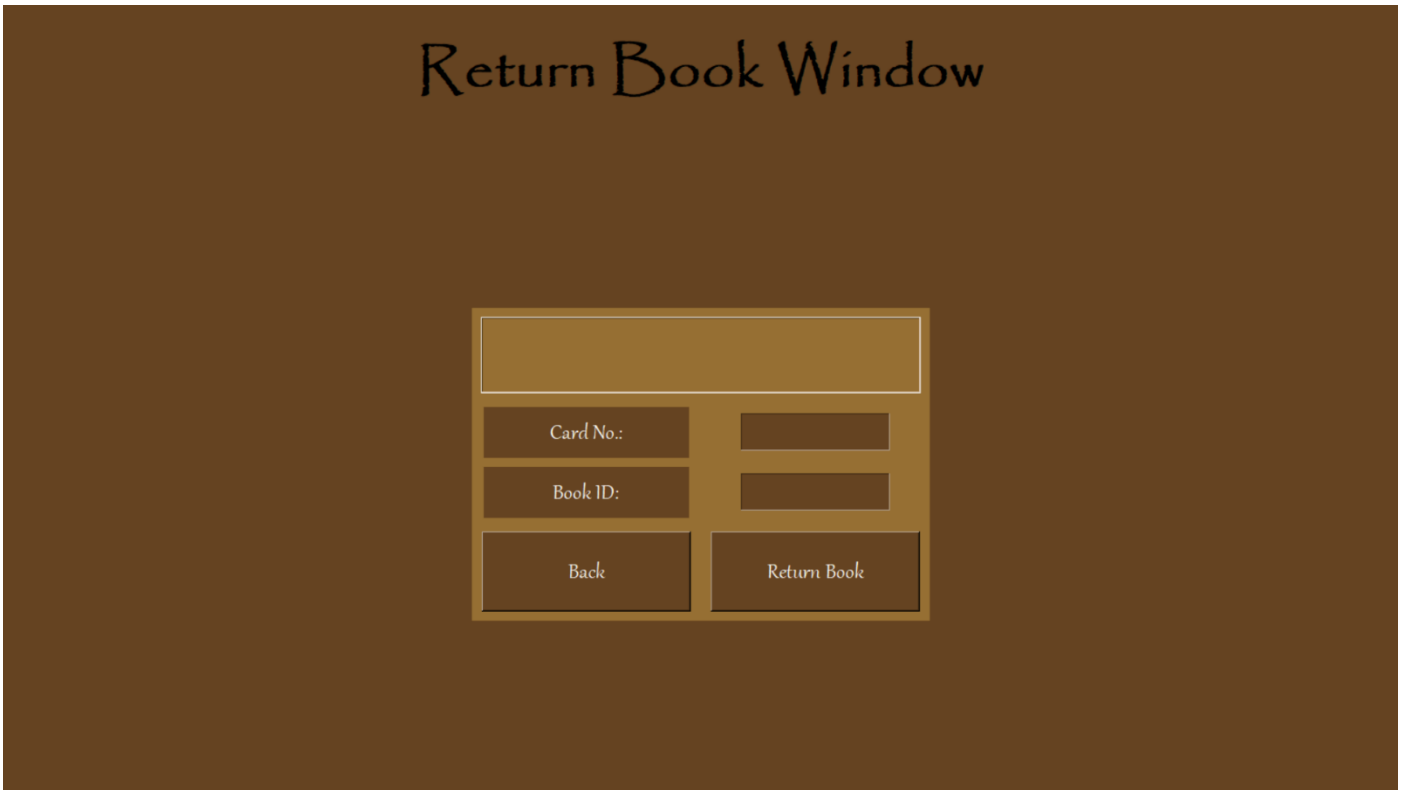



Add New Book
Title:
Author:
Genre:
Copies:
Back
Add Book

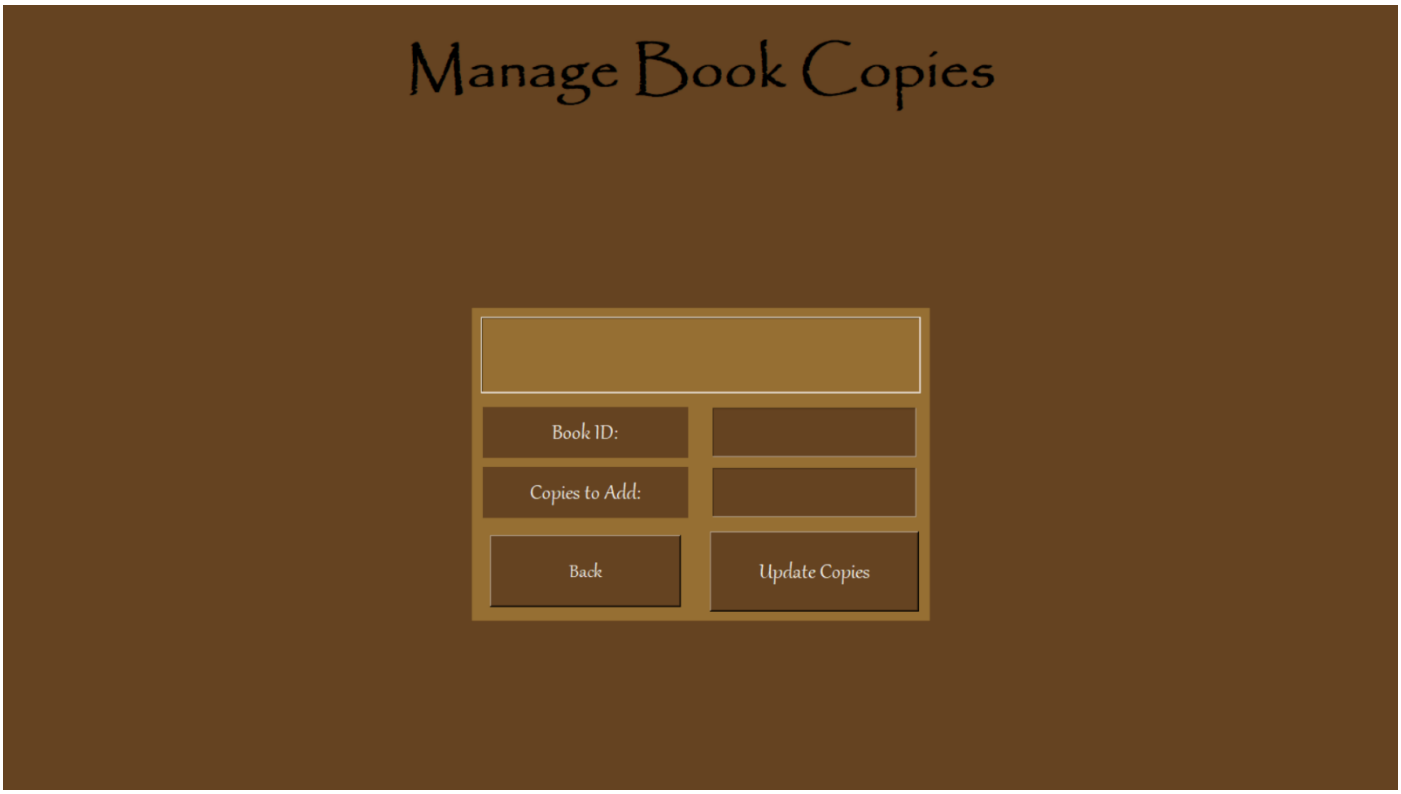

Remove a Book
Book ID:
Remove Book
Back


Search Book Window
Search Book:
Search By:
Title
Back
Search Book


Issue Book Window
Card No.:
Book ID:
Back
Issue Book


Return Book Window
Card No.:
Book ID:
Back
Return Book

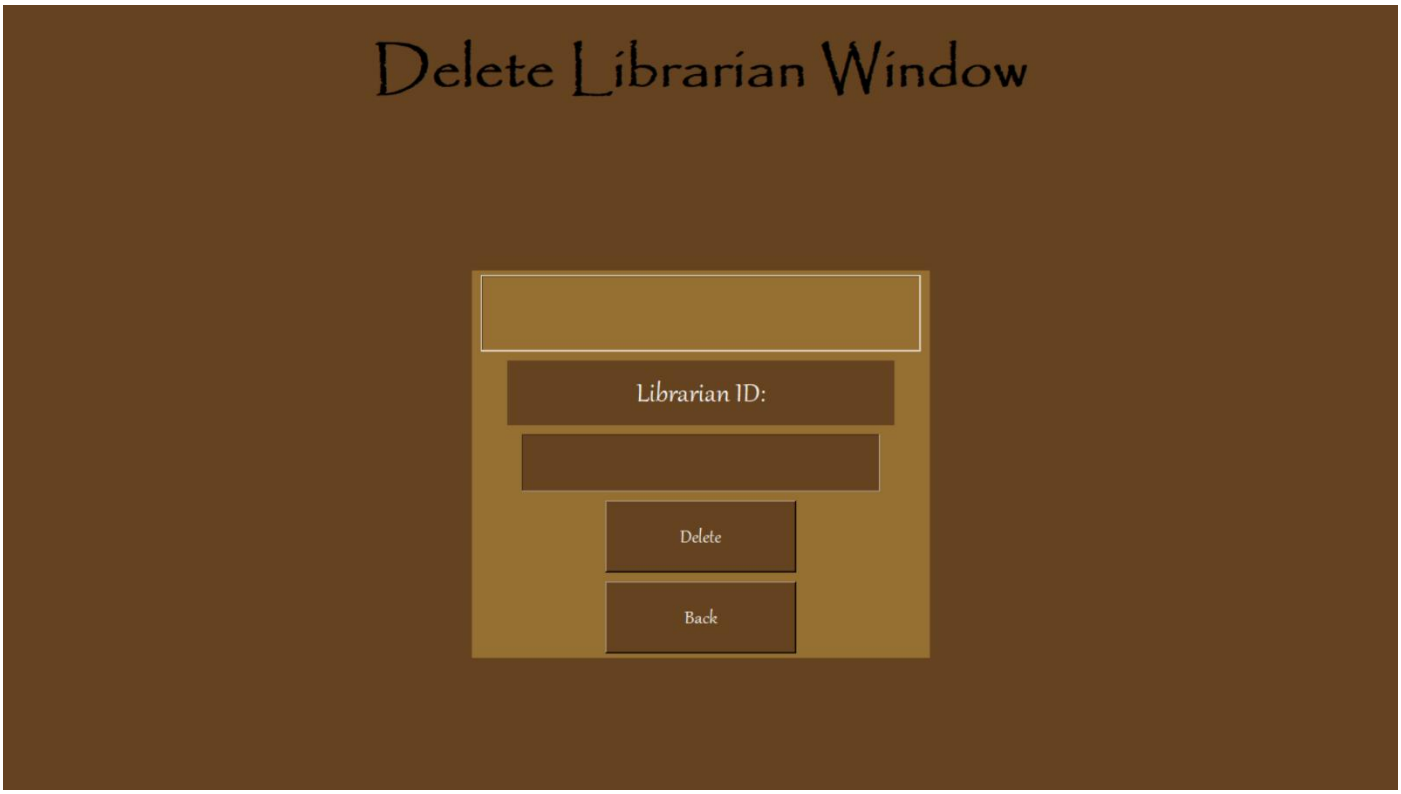

Books Not Returned
Book ID: 1, Title: Naruto, Borrower ID: 1, Name: Palki, Issued: 2025-01-09, Due: 2025-01-23
Book ID: 3, Title: Demon Slayer, Borrower ID: 2, Name: Dhaval, Issued: 2024-10-18, Due: 2024-11-01
Book ID: 4, Title: Dragon Ball, Borrower ID: 3, Name: Raghav, Issued: 2024-11-06, Due: 2024-11-20
Back


Manage Book Copies
Book ID:
Copies to Add:
Back
Update Copies


Fine Management
Borrower: Dhaval (2), Book: Demon Slayer, Overdue Days: 35, Fine: ₹175
Borrower: Raghav (3), Book: Dragon Ball, Overdue Days: 16, Fine: ₹80
Back


Search Librarian Window
Librarian Name:
Search
Back


Add New Librarian
Name:
Address:
Phone No.:
Email:
Password:
Back
Sign Up


Delete Librarian Window
Librarian ID:
Delete
Back

CONCLUSION

The **Library Management System** project successfully addresses the challenges of managing library operations through a user-friendly interface built with **Python** and **MySQL**. The system ensures efficient book inventory management, secure user authentication, and accurate transaction tracking, simplifying tasks for librarians while improving the user experience. This project has enhanced our skills in database management, GUI design, and collaborative problem-solving.

Future Scope:

1. Integration of cloud-based storage for enhanced accessibility and scalability.
2. Development of a mobile application for better user convenience.
3. Implementation of advanced features like book recommendation systems using machine learning.
4. Multi-language support to cater to diverse user bases.

This system lays the foundation for further advancements to meet the evolving needs of library management.

REFERENCES

- BIBLIOGRAPHY
  - Informatics Practices (Sumita Arora)
  - Informatics Practices (N.C.E.R.T.)

- WEBLIOGRAPHY
  - Chat GPT
  - Wikipedia

APPENDIX - 1 (database code)

```
# Imports
from tkinter import messagebox
import mysql.connector as sqlcon
from mysql.connector import Error

My_Sql_Password = 'yuvraj' # Replace with MySQL password
Do_You_Want_To_Delete_Old_Database = 'yes' # Type 'yes' or 'no'

''' SQL Connection Codes '''

# Function To Create A Connection To MySQL Server
def connect_SQ():
    try:
        connection = sqlcon.connect(host='localhost',user='root',password = My_Sql_Password)
        return connection
    except Error as e:
        if e.errno == 1062:  # MySQL error code for duplicate entry
            print("Duplicate Entry Found")
        else:
            print(f"Error: {e}")
        return None

def connect_db():
    try:
        connection = sqlcon.connect(
            host='localhost',
            user='root',
            password = My_Sql_Password,
            database = "Library_Management"
    )
        return connection
    except Error as e:
        if e.errno == 1062:  # MySQL error code for duplicate entry
            print("Duplicate Entry Found")
        else:
            messagebox.showerror("Database Error", f"Error connecting to the database: {e}")
        return None

# Function To Execute SQL Queries
def execute_query(connection, query, values=None):
    cursor = connection.cursor()
    try:
        if values:
            cursor.executemany(query, values)
        else:
            cursor.execute(query)
        connection.commit()
```

```
        print("Query executed successfully")
    except Error as e:
        if e.errno == 1062:  # MySQL error code for duplicate entry
            print("Duplicate Entry Found")
        else:
            print(f"Error: {e}")

''' SQL Commands '''

# Drop Library_Management Database
if Do_You_Want_To_Delete_Old_Database == 'yes':
    drop_db_query = "DROP DATABASE Library_Management"

# Create Library_Management Database
create_db_query = "CREATE DATABASE IF NOT EXISTS Library_Management"

# Create Books Table
create_books_table_query = """
CREATE TABLE IF NOT EXISTS Book (
    Book_ID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255) UNIQUE NOT NULL,
    Genre VARCHAR(20) NOT NULL
)
"""

# Create Book Copies Table
create_book_copies_table_query = """
CREATE TABLE IF NOT EXISTS Book_Copies (
    Book_ID INT PRIMARY KEY,
    No_Of_Copies INT NOT NULL,
    FOREIGN KEY (Book_ID) REFERENCES BOOK(Book_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
)
"""

# Create Book Authors Table
create_book_authors_table_query = """
CREATE TABLE IF NOT EXISTS Book_Authors (
    Book_ID INT NOT NULL,
    Author_Name VARCHAR(255) NOT NULL,
    PRIMARY KEY (Book_ID, Author_Name),
    FOREIGN KEY (Book_ID) REFERENCES BOOK(Book_ID)
        ON DELETE CASCADE ON UPDATE CASCADE
)
"""

# Create Borrower Table
create_borrower_table_query = """
CREATE TABLE IF NOT EXISTS Borrower (
    Card_No INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(55) NOT NULL,
    Address TEXT,
```

```
    Email VARCHAR(255) UNIQUE ,
    Phone VARCHAR(10) UNIQUE NOT NULL
)
"""

# Create Book Loan Table
create_book_loans_table_query = """
CREATE TABLE IF NOT EXISTS Book_Loans (
    Book_ID INT NOT NULL,
    Card_No INT NOT NULL,
    Date_Out DATE NOT NULL,
    Due_Date DATE,
    PRIMARY KEY (Book_ID, Card_No, Date_Out),
    FOREIGN KEY (Book_ID) REFERENCES BOOK(Book_ID)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (Card_No) REFERENCES BORROWER(Card_No)
        ON DELETE CASCADE ON UPDATE CASCADE
)
"""

# Create Librarian Table
create_librarian_table_query = """
CREATE TABLE IF NOT EXISTS Librarian (
    Librarian_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Address TEXT,
    Email VARCHAR(255) UNIQUE ,
    Phone VARCHAR(15) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL
)
"""

# Inserting in Book
insert_book_query = """
INSERT INTO Book (Title, Genre)
VALUES (%s, %s)
"""
book_values = [
    ("Naruto", "Manga"),
    ("Jujutsu Kaisen", "Manga"),
    ("Demon Slayer", "Manga"),
    ("Dragon Ball", "Manga"),
    ("Solo Leveling", "Novel"),
    ("Death Note", "Manga"),
    ("Bleach", "Manga")
]

# Inserting in Book Copies
insert_book_copies_query = """
INSERT INTO Book_Copies (Book_ID, No_Of_Copies)
VALUES (%s, %s)
"""
```

```
book_copies_values = [
    (1, 15),  # For Book_ID 1
    (2, 9),
    (3, 7),
    (4, 3),
    (5, 11),
    (6, 13),
    (7, 5)
]

# Inserting in Book Authors
insert_book_authors_query = """
INSERT INTO Book_Authors (Book_ID, Author_Name)
VALUES (%s, %s)
"""
book_authors_values = [
    (1, "Masashi Kishimoto"),
    (2, "Gege Akutami"),
    (3, "Koyoharu Gotouge"),
    (4, "Akira Toriyama"),
    (5, "Chu-Gong"),
    (6, "Tsugumi Ohba"),
    (7, "Tite Kubo")
]

# Inserting in Borrower Table
insert_borrower_query = """
INSERT INTO Borrower (Name, Address, Email, Phone)
VALUES (%s, %s, %s, %s)
"""
borrower_values = [
    ("Palki", "221 Basant St", "Palki@email.com","7973484399"),
    ("Dhaval", "123 Main St", "Dhaval@email.com", "9876543210"),
    ("Raghav", "456 Elm St", "Raghav@email.com", "9876543211")
]

# Inserting in Book Loans
insert_book_loans_query = """
INSERT INTO Book_Loans (Book_ID, Card_No, Date_Out, Due_Date)
VALUES (%s, %s, %s, %s)
"""
book_loans_values = [
    (3, 2, '2024-10-18', '2024-11-01'),  # Dhaval borrowing "Demon Slayer"
    (4, 3, '2024-11-06', '2024-11-20'),   # Raghav borrowing "Dragon Ball"
    # (1, 1, '2025-01-09', '2025-01-23') # Palki borrowing "Naruto"
]

# Inserting in Librarian Table
insert_librarian_query = """
INSERT INTO Librarian (Name, Address, Email, Phone, Password)
VALUES (%s, %s, %s, %s, %s)
"""
```

```
librarian_values = [
    ("1", "1 Basi St", "One@email.com", "1234567892", "1"),
    ("Yuvraj", "220 Basant St", "Yuvraj@email.com", "1234567893", "y"),
    ("Ashwinder", "321 Oak St", "Ashwinder@email.com", "1234567894", "a"),
    ("Madhav", "789 Pine St", "Madhav@email.com", "1234567895", "m"),
]

# Main SQL Script
connection = connect_SQ()

if connection:
    try:
        if Do_You_Want_To_Delete_Old_Database == 'yes':
            # Drop the Library_Management database
            execute_query(connection, drop_db_query)

        # Create the Library_Management database
        execute_query(connection, create_db_query)

        # Use the Library_Management database
        connection.database = "Library_Management"

        # Create The Books Table
        execute_query(connection, create_books_table_query)

        # Create the Book_Copies table
        execute_query(connection, create_book_copies_table_query)

        # Create the Book_Copies table
        execute_query(connection, create_book_authors_table_query)

        # Create the Borrower table
        execute_query(connection, create_borrower_table_query)

        # Create the Book_Loans table
        execute_query(connection, create_book_loans_table_query)

        # Create the Librarian table
        execute_query(connection, create_librarian_table_query)

        # Insert book data
        execute_query(connection, insert_book_query, book_values)

        # Insert book copies data
        execute_query(connection, insert_book_copies_query, book_copies_values)

        # Insert book authors data
        execute_query(connection, insert_book_authors_query, book_authors_values)

        # Insert borrower data
        execute_query(connection, insert_borrower_query, borrower_values)
```

```
        # Insert book loans data
        execute_query(connection, insert_book_loans_query, book_loans_values)

        # Insert librarian data
        execute_query(connection, insert_librarian_query, librarian_values)

    finally:
        # Close the connection after the insertions
        connection.close()
```

APPENDIX - 2 (main project code)

```
# Imports
import tkinter as tk
from tkinter import messagebox
import datetime
import csv
import os
import mysql.connector as sqlcon
from mysql.connector import Error

# Some Pre Defined Values
Desired_Folder = "F:\\Code Playground\\Library Management"  # Replace With Your desired Folder Where You Want To
Keep"Activity_Log.csv" File
My_Sql_Password = 'yuvraj' # Replace with MySQL password
Logged_In_User = 'Self' # Don't Change

''' Colors '''

Color_1 = '#654321' # Dark Brown
Color_2 = '#FCFBF4' # Cream White
Color_3 = '#966F33' # Tree Brown
Color_4 = '#333333' # Dark Gray
Color_5 = '#000000' # Pure Black

''' SQL Connection Codes '''

# Function To Create A Connection To MySQL Server
def connect_db():
    try:
        connection = sqlcon.connect(
            host='localhost',
            user='root',
            password = My_Sql_Password,
            database = "Library_Management"
    )
        return connection
    except Error as e:
        if e.errno == 1062:  # MySQL error code for duplicate entry
            print("Duplicate Entry Found")
        else:
            messagebox.showerror("Database Error", f"Error connecting to the database: {e}")
        return None

# Function To Execute An Update Query
def execute_update(query, values=None):
    with connect_db() as mydb:
        mycursor = mydb.cursor()
        try:
```

```
            mycursor.execute(query, values)
            mydb.commit()
        except sqlcon.Error as e:
            if e.errno == 1062:  # MySQL error code for duplicate entry
                print("Duplicate Entry Found")
            else:
                messagebox.showerror("Database Error", f"Error connecting to the database: {e}")

# Function To Execute And Fetch Results
def execute_fetch_results(query, values=None):
    with connect_db() as mydb:
        mycursor = mydb.cursor()
        try:
            mycursor.execute(query, values)
            results = mycursor.fetchall()
            return results
        except sqlcon.Error as e:
            if e.errno == 1062:  # MySQL error code for duplicate entry
                print("Duplicate Entry Found")
            else:
                messagebox.showerror("Database Error", f"Error connecting to the database: {e}")
            return None

''' GUIs '''

# GUI entry point
def main_menu():
    Main_Window = tk.Tk()
    Main_Window.title("Library Management System")
    Main_Window.attributes('-fullscreen', True)

    button_frame = Box(Main_Window,"Welcome to Public Library")

    # Login Button
    Login_btn = tk.Button(button_frame, text="Login",font=("Gabriola", 26),bg=Color_1, fg=Color_2, width=20,
command=lambda: login())
    Login_btn.grid(row=1, column=1, padx=10, pady=5)

    # Signup Button
    Signup_btn = tk.Button(button_frame, text="Sign Up",font=("Gabriola", 26),bg=Color_1, fg=Color_2, width=20,
command=lambda: signup('Librarian'))
    Signup_btn.grid(row=2, column=1, padx=10, pady=5)

    # Back Button
    Back_btn = tk.Button(button_frame, text="Back",font=("Gabriola", 26),bg=Color_1, fg=Color_2, width=20,
command=lambda: Main_Window.destroy() )
    Back_btn.grid(row=3, column=1, padx=10, pady=5)

    Main_Window.mainloop()

# GUI for login
def login():
```

```
    login_page = tk.Toplevel()
    login_page.title("Login")
    login_page.attributes('-fullscreen', True)

    button_frame = Box(login_page,"Login")

    # Name
    Name = tk.Label(button_frame, text="Name:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Name.grid(row=3, column=2, padx=10, pady=5)

    entry_name = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_name.grid(row=4, column=2, padx=10, pady=5)

    # ID
    Id = tk.Label(button_frame, text="ID:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Id.grid(row=5, column=2, padx=10, pady=5)

    entry_id = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_id.grid(row=6, column=2, padx=10, pady=5)

    # Password
    Passw = tk.Label(button_frame, text="Password:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Passw.grid(row=7, column=2, padx=10, pady=5)

    entry_password = tk.Entry(button_frame, show="*",font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_password.grid(row=8, column=2, padx=10, pady=5)

    # Buttons
    Back = tk.Button(button_frame, text="Back",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20 ,
command=lambda: login_page.destroy())
    Back.grid(row=9, column=1, padx=10, pady=5)

    login = tk.Button(button_frame, text="Login",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20
,command=lambda: validate_login(entry_name.get(), entry_id.get(), entry_password.get(), login_page))
    login.grid(row=9, column=3, padx=10, pady=5)

# GUI for signup
def signup(Role):
    signup_page = tk.Toplevel()
    signup_page.title("Sign Up")
    signup_page.attributes('-fullscreen', True)

    if Role == 'Borrower' :
        button_frame = Box(signup_page,"Add New Borrower")

        # Signup Button
        SignUp = tk.Button(button_frame, text="Sign Up",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20 ,
command=lambda: insert(Role, entry_name.get(), entry_Address.get(), entry_phone.get(), entry_email.get(), "none",
result_listbox))
        SignUp.grid(row=7, column=2, padx=10, pady=5)

    elif Role == 'Librarian' :
```

```
        button_frame = Box(signup_page,"Add New Librarian")

        # Password
        Passw = tk.Label(button_frame, text="Password:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
        Passw.grid(row=6, column=1, padx=10, pady=5)

        entry_password = tk.Entry(button_frame, show="*",font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
        entry_password.grid(row=6, column=2, padx=10, pady=5)

        # Signup Button
        SignUp = tk.Button(button_frame, text="Sign Up",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20
,command=lambda: insert(Role, entry_name.get(), entry_Address.get(), entry_phone.get(), entry_email.get(),
ntry_password.get(), result_listbox))
        SignUp.grid(row=7, column=2, padx=10, pady=5)

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, columnspan=2, padx=10, pady=5)

    # Name
    Name = tk.Label(button_frame, text="Name:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Name.grid(row=2, column=1, padx=10, pady=5)

    entry_name = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_name.grid(row=2, column=2, padx=10, pady=5)

    # Address
    Address = tk.Label(button_frame, text="Address:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Address.grid(row=3, column=1, padx=10, pady=5)

    entry_Address = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_Address.grid(row=3, column=2, padx=10, pady=5)

    # Phone No.
    Phone = tk.Label(button_frame, text="Phone No.:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Phone.grid(row=4, column=1, padx=10, pady=5)

    entry_phone = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_phone.grid(row=4, column=2, padx=10, pady=5)

    # Email
    Email = tk.Label(button_frame, text="Email:",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20)
    Email.grid(row=5, column=1, padx=10, pady=5)

    entry_email = tk.Entry(button_frame,font=("Gabriola", 15),bg=Color_1, fg=Color_2, width=20)
    entry_email.grid(row=5, column=2, padx=10, pady=5)

    # Back Button
    Back = tk.Button(button_frame, text="Back",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20 ,
command=lambda: signup_page.destroy())
    Back.grid(row=7, column=1, padx=10, pady=5)
```

```
# GUI For Librarian Options
def librarian_options():
    librarian_window = tk.Toplevel()
    librarian_window.title("Librarian Options")
    librarian_window.attributes('-fullscreen', True)

    button_frame = Box(librarian_window,"Librarian Window")


    # Search Borrower Button
    searchBorrower = tk.Button(button_frame, text="Search Borrower",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: search_Borrower(librarian_window))
    searchBorrower.grid(row=1, column=1, padx=10, pady=5)

    # Add Borrower Button
    addBorrower = tk.Button(button_frame, text="Add Borrower",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: signup('Borrower'))
    addBorrower.grid(row=2, column=1, padx=10, pady=5)

    # Delete Borrower Button
    deleteBorrower = tk.Button(button_frame, text="Delete Borrower",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: delete_Borrower(librarian_window))
    deleteBorrower.grid(row=3, column=1, padx=10, pady=5)


    # View All Books Button
    viewallbooks = tk.Button(button_frame, text="View All Books",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: view_all_books(librarian_window))
    viewallbooks.grid(row=1, column=2, padx=10, pady=5)

    # Add Book Button
    addbook = tk.Button(button_frame, text="Add Book",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: add_book(librarian_window))
    addbook.grid(row=2, column=2, padx=10, pady=5)

    # Remove Book Button
    removebook = tk.Button(button_frame, text="Remove Book",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: remove_book(librarian_window))
    removebook.grid(row=3, column=2, padx=10, pady=5)


    # Search Book Button
    searchbook = tk.Button(button_frame, text="Search Book",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: search_book(librarian_window))
    searchbook.grid(row=1, column=3, padx=10, pady=5)

    # Issue Book Button
    issuebook = tk.Button(button_frame, text="Issue Book",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: issue_book(librarian_window))
    issuebook.grid(row=2, column=3, padx=10, pady=5)

    # Return Book Button
```

```
    returnbook = tk.Button(button_frame, text="Return Book",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20,
command=lambda: return_book(librarian_window))
    returnbook.grid(row=3, column=3, padx=10, pady=5)


    # View Issued Books Button
    viewissuedbooks = tk.Button(button_frame, text="Books Not Returned",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: view_issued_books(librarian_window))
    viewissuedbooks.grid(row=1, column=4, padx=10, pady=5)

    # Manage Book Copies Button
    managebookcopies = tk.Button(button_frame, text="Manage Book Copies",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: manage_book_copies(librarian_window))
    managebookcopies.grid(row=2, column=4, padx=10, pady=5)

    # Fine Details Button
    finedetails = tk.Button(button_frame, text="View Fine Details",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: fine_manager(librarian_window))
    finedetails.grid(row=3, column=4, padx=10, pady=5)


    # Search Librarian Button
    searchlibrarian = tk.Button(button_frame, text="Search Librarian",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: search_librarian(librarian_window))
    searchlibrarian.grid(row=1, column=5, padx=10, pady=5)

    # Add Librarian Button
    addlibrarian = tk.Button(button_frame, text="Add Librarian",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: signup('Librarian'))
    addlibrarian.grid(row=2, column=5, padx=10, pady=5)

    # Delete Librarian Button
    deletelibrarian = tk.Button(button_frame, text="Delete Librarian",font=("Gabriola", 20),bg=Color_1, fg=Color_2,
width=20, command=lambda: delete_librarian(librarian_window))
    deletelibrarian.grid(row=3, column=5, padx=10, pady=5)


    # Back Button
    Back = tk.Button(button_frame, text="Back",font=("Gabriola", 20),bg=Color_1, fg=Color_2, width=20, command=lambda:
librarian_window.destroy())
    Back.grid(row=4, column=3, padx=10, pady=5)

# GUI For Searching Borrower
def search_Borrower(page):
    search_borrower_window = tk.Toplevel()
    search_borrower_window.title("Search Borrower")
    search_borrower_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(search_borrower_window, "Search Borrower Window")

    # Label
```

```
    label_search = tk.Label(button_frame, text="Borrower Name:", font=("Gabriola", 18), bg=Color_1, fg=Color_2)
    label_search.grid(row=1, column=1, padx=10, pady=5)

    # Entry
    entry_search = tk.Entry(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2)
    entry_search.grid(row=2, column=1, padx=10, pady=5)

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2, width=80)
    result_listbox.grid(row=3, column=1, padx=10, pady=5)

    # Buttons
    search_button = tk.Button(button_frame, text="Search", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: search_action(result_listbox, entry_search.get(), 'Borrower'))
    search_button.grid(row=4, column=1, padx=10, pady=5)

    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(search_borrower_window,librarian_options))
    back_button.grid(row=5, column=1, padx=10, pady=5)

# GUI For Deleting Borrower
def delete_Borrower(page):
    delete_borrower_window = tk.Toplevel()
    delete_borrower_window.title("Delete Borrower")
    delete_borrower_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(delete_borrower_window, "Delete Borrower Window")

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, padx=10, pady=5)

    # Label
    label_card_no = tk.Label(button_frame, text="Card No:", font=("Gabriola", 26), bg=Color_1, fg=Color_2, width=30)
    label_card_no.grid(row=2, column=1, padx=10, pady=5)

    # Entry
    entry_card_no = tk.Entry(button_frame, font=("Gabriola", 24), bg=Color_1, fg=Color_2, width=30)
    entry_card_no.grid(row=3, column=1, padx=10, pady=5)

    # Buttons
    delete_button = tk.Button(button_frame, text="Delete", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: delete_action(entry_card_no.get(), result_listbox, 'Borrower'))
    delete_button.grid(row=4, column=1, padx=10, pady=5)

    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(delete_borrower_window,librarian_options))
    back_button.grid(row=5, column=1, padx=10, pady=5)

# GUI for viewing all books
def view_all_books(page):
```

```
    View_All_Books_window = tk.Toplevel()
    View_All_Books_window.title("View All Books")
    View_All_Books_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(View_All_Books_window,"View All Books")

    # listbox
    result_listbox = tk.Listbox(button_frame,font=("Gabriola", 16),bg=Color_1, fg=Color_2, width=125)
    result_listbox.grid(row=1, column=1, padx=10, pady=5)

    view_all_books_action(result_listbox)

    # Back
    Back = tk.Button(button_frame, text="Back",font=("Gabriola", 18),bg=Color_1, fg=Color_2, width=16, command=lambda:
back(View_All_Books_window,librarian_options))
    Back.grid(row=2, column=1, padx=10, pady=5)

# GUI for Adding a Book
def add_book(page):
    add_book_window = tk.Toplevel()
    add_book_window.title("Add Book")
    add_book_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(add_book_window, "Add New Book")

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, columnspan=2, padx=10, pady=5)

    # Title Entry
    title = tk.Label(button_frame, text="Title:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    title.grid(row=2, column=1, padx=10, pady=5)

    entry_title = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_title.grid(row=2, column=2, padx=10, pady=5)

    # Author Entry
    author = tk.Label(button_frame, text="Author:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    author.grid(row=3, column=1, padx=10, pady=5)

    entry_author = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_author.grid(row=3, column=2, padx=10, pady=5)

    # Genre Entry
    genre = tk.Label(button_frame, text="Genre:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    genre.grid(row=4, column=1, padx=10, pady=5)

    entry_genre = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_genre.grid(row=4, column=2, padx=10, pady=5)
```

```
    # Copies Entry
    copies = tk.Label(button_frame, text="Copies:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    copies.grid(row=5, column=1, padx=10, pady=5)

    entry_copies = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_copies.grid(row=5, column=2, padx=10, pady=5)

    # Back Button
    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(add_book_window,librarian_options))
    back_button.grid(row=6, column=1, padx=10, pady=5)

    # Add Book Button
    add_button = tk.Button(button_frame, text="Add Book", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20,
command=lambda: add_book_action(entry_title.get(), entry_author.get(), entry_genre.get(), entry_copies.get(),
result_listbox))
    add_button.grid(row=6, column=2, padx=10, pady=5)

# GUI For Deleting A Book
def remove_book(page):
    remove_book_window = tk.Toplevel()
    remove_book_window.title("Remove Book")
    remove_book_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(remove_book_window, "Remove a Book")

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, padx=10, pady=5)

    # Book ID Label
    bookid = tk.Label(button_frame, text="Book ID:", font=("Gabriola", 26), bg=Color_1, fg=Color_2, width=30)
    bookid.grid(row=2, column=1, padx=10, pady=5)

    # Entry
    entry_book_id = tk.Entry(button_frame, font=("Gabriola", 24), bg=Color_1, fg=Color_2, width=30)
    entry_book_id.grid(row=3, column=1, padx=10, pady=5)

    # Buttons
    delete_button = tk.Button(button_frame, text="Remove Book", font=("Gabriola", 18), bg=Color_1, fg=Color_2,
width=20, command=lambda: remove_book_action(entry_book_id.get(), result_listbox))
    delete_button.grid(row=4, column=1, padx=10, pady=5)

    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(remove_book_window,librarian_options))
    back_button.grid(row=5, column=1, padx=10, pady=5)

# GUI For Searching Books
def search_book(page):
    search_book_window = tk.Toplevel()
    search_book_window.title("Search Book")
```

```
    search_book_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(search_book_window, "Search Book Window")

    # Label
    label_search = tk.Label(button_frame, text="Search Book:", font=("Gabriola", 18), bg=Color_3, fg=Color_2,
width=18)
    label_search.grid(row=1, column=1, padx=10, pady=5)

    # Typing Box
    entry_search = tk.Entry(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2, width=26)
    entry_search.grid(row=1, column=2, padx=10, pady=5)

    # Dropdown Menu for Search Criteria
    label_criteria = tk.Label(button_frame, text="Search By:", font=("Gabriola", 18), bg=Color_3, fg=Color_2,
width=18)
    label_criteria.grid(row=1, column=3, padx=10, pady=5)

    # Dropdown variable and menu
    search_by = tk.StringVar(value="Title")  # Default value
    dropdown_menu = tk.OptionMenu(button_frame, search_by, "Title", "Genre", "Author")
    dropdown_menu.config(font=("Gabriola", 16), bg=Color_1, fg=Color_2, width=20)
    dropdown_menu.grid(row=1, column=4, padx=10, pady=5)

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2, width=125)
    result_listbox.grid(row=2, column=1, columnspan=4, padx=10, pady=5)

    # Search Button
    search_book = tk.Button(button_frame, text="Search Book", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=18,
        command=lambda: search_action(result_listbox, entry_search.get(), search_by.get()))
    search_book.grid(row=3, column=3, columnspan=2, padx=10, pady=5)

    # Back Button
    Back = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=16,
command=lambda: back(search_book_window, librarian_options))
    Back.grid(row=3, column=1, columnspan=2, padx=10, pady=5)

    search_book_window.mainloop()

# GUI For Issuing A Book
def issue_book(page):
    issue_book_window = tk.Toplevel()
    issue_book_window.title("Issue Book")
    issue_book_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(issue_book_window, "Issue Book Window")

    # Listbox to display messages
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
```

```
    result_listbox.grid(row=1, column=1, columnspan=2, padx=10, pady=10)

    # Borrower ID Label and Entry
    borrower = tk.Label(button_frame, text="Card No.:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    borrower.grid(row=2, column=1, padx=10, pady=5)

    entry_borrower_id = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_borrower_id.grid(row=2, column=2, padx=10, pady=5)

    # Book ID Label and Entry
    book = tk.Label(button_frame, text="Book ID:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    book.grid(row=3, column=1, padx=10, pady=5)

    entry_book_id = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_book_id.grid(row=3, column=2, padx=10, pady=5)

    # Buttons
    issue_book_bt = tk.Button(button_frame, text="Issue Book", font=("Gabriola", 20), bg=Color_1, fg=Color_2,
width=20, command=lambda: issue_book_action(entry_borrower_id.get(), entry_book_id.get(), result_listbox))
    issue_book_bt.grid(row=4, column=2, padx=10, pady=10)

    back_bt = tk.Button(button_frame, text="Back", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(issue_book_window, librarian_options))
    back_bt.grid(row=4, column=1, padx=10, pady=10)

# GUI For Returning A Book
def return_book(page):
    return_book_window = tk.Toplevel()  # Use Toplevel instead of Tk to create a new window
    return_book_window.title("Return Book")
    return_book_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(return_book_window,"Return Book Window")

    # Listbox to display messages
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, columnspan=2, padx=10, pady=10)

    # Borrower ID Label and Entry
    borrower = tk.Label(button_frame, text="Card No.:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    borrower.grid(row=2, column=1, padx=10, pady=5)

    entry_borrower_id = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_borrower_id.grid(row=2, column=2, padx=10, pady=5)

    # Book ID Label and Entry
    book = tk.Label(button_frame, text="Book ID:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    book.grid(row=3, column=1, padx=10, pady=5)

    entry_book_id = tk.Entry(button_frame, font=("Gabriola", 15), bg=Color_1, fg=Color_2, width=20)
    entry_book_id.grid(row=3, column=2, padx=10, pady=5)
```

```
    # return Book
    return_book_bt = tk.Button(button_frame, text="Return Book", font=("Gabriola", 20), bg=Color_1, fg=Color_2,
width=20, command=lambda: return_book_action(entry_borrower_id.get(), entry_book_id.get(), result_listbox))
    return_book_bt.grid(row=4, column=2, padx=10, pady=10)

    back_bt = tk.Button(button_frame, text="Back", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(return_book_window, librarian_options))
    back_bt.grid(row=4, column=1, padx=10, pady=10)

# GUI For Viewing Issued Books
def view_issued_books(page):
    view_issued_window = tk.Toplevel()
    view_issued_window.title("Books Not Returned")
    view_issued_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(view_issued_window, "Books Not Returned")

    # Listbox to display issued books
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=10, width=125, bg=Color_1, fg=Color_2)
    result_listbox.grid(row=1, column=1, padx=10, pady=10)

    # Fetch and display all issued books
    view_all_issued_books(result_listbox)

    # Back button
    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2,
width=20,command=lambda: back(view_issued_window, librarian_options))
    back_button.grid(row=2, column=1, padx=10, pady=10)

    view_issued_window.mainloop()

# GUI For To Add Copies
def manage_book_copies(page):
    manage_copies_window = tk.Toplevel()
    manage_copies_window.title("Manage Book Copies")
    manage_copies_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(manage_copies_window, "Manage Book Copies")

    # Listbox to show update results
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, columnspan=2, padx=10, pady=10)

    # Book ID Label and Entry
    book_id_label = tk.Label(button_frame, text="Book ID:", font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    book_id_label.grid(row=2, column=1, padx=10, pady=5)

    entry_book_id = tk.Entry(button_frame, font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    entry_book_id.grid(row=2, column=2, padx=10, pady=5)
```

```
    # Copies to Add Label and Entry
    copies_label = tk.Label(button_frame, text="Copies to Add:", font=("Gabriola", 20), bg=Color_1, fg=Color_2,
width=20)
    copies_label.grid(row=3, column=1, padx=10, pady=5)

    entry_copies = tk.Entry(button_frame, font=("Gabriola", 20), bg=Color_1, fg=Color_2, width=20)
    entry_copies.grid(row=3, column=2, padx=10, pady=5)

    # Back button
    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2,
width=20,command=lambda: back(manage_copies_window, librarian_options))
    back_button.grid(row=4, column=1, padx=10, pady=10)

    # Button to update book copies
    update_button = tk.Button(button_frame, text="Update Copies", font=("Gabriola", 20), bg=Color_1, fg=Color_2,
width=20, command=lambda: manage_book_copies_action(entry_book_id.get(), int(entry_copies.get()), result_listbox))
    update_button.grid(row=4, column=2, padx=10, pady=10)

# GUI For Calculating Fine
def fine_manager(page):
    fine_manager_window = tk.Toplevel()
    fine_manager_window.title("Fine Manager")
    fine_manager_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(fine_manager_window, "Fine Management")

    # Listbox to display fines
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=10, width=80, bg=Color_1, fg=Color_2)
    result_listbox.grid(row=1, column=1, padx=10, pady=10)

    # Calculate fines
    calculate_fines(result_listbox)

    # Back button
    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(fine_manager_window, librarian_options))
    back_button.grid(row=2, column=1, padx=10, pady=10)

    fine_manager_window.mainloop()

# GUI For Searching Librarian
def search_librarian(page):
    search_librarian_window = tk.Toplevel()
    search_librarian_window.title("Search Librarian")
    search_librarian_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(search_librarian_window, "Search Librarian Window")

    # Label
    label_search = tk.Label(button_frame, text="Librarian Name:", font=("Gabriola", 18), bg=Color_1, fg=Color_2)
```

```
    label_search.grid(row=1, column=1, padx=10, pady=5)

    # Entry
    entry_search = tk.Entry(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2)
    entry_search.grid(row=2, column=1, padx=10, pady=5)

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), bg=Color_1, fg=Color_2, width=80)
    result_listbox.grid(row=3, column=1, padx=10, pady=5)

    # Buttons
    search_button = tk.Button(button_frame, text="Search", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: search_action(result_listbox, entry_search.get(), 'Librarian'))
    search_button.grid(row=4, column=1, padx=10, pady=5)

    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(search_librarian_window,librarian_options))
    back_button.grid(row=5, column=1, padx=10, pady=5)

# GUI For Deleting A Librarian
def delete_librarian(page):
    delete_librarian_window = tk.Toplevel()
    delete_librarian_window.title("Delete Librarian")
    delete_librarian_window.attributes('-fullscreen', True)
    page.destroy()

    button_frame = Box(delete_librarian_window, "Delete Librarian Window")

    # Listbox
    result_listbox = tk.Listbox(button_frame, font=("Gabriola", 16), height=2, width=60, bg=Color_3, fg=Color_4)
    result_listbox.grid(row=1, column=1, padx=10, pady=5)

    # Label
    label_card_no = tk.Label(button_frame, text="Librarian ID:", font=("Gabriola", 26), bg=Color_1, fg=Color_2,
width=30)
    label_card_no.grid(row=2, column=1, padx=10, pady=5)

    # Entry
    entry_card_no = tk.Entry(button_frame, font=("Gabriola", 24), bg=Color_1, fg=Color_2, width=30)
    entry_card_no.grid(row=3, column=1, padx=10, pady=5)

    # Buttons
    delete_button = tk.Button(button_frame, text="Delete", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: delete_action(entry_card_no.get(), result_listbox, 'Librarian'))
    delete_button.grid(row=4, column=1, padx=10, pady=5)

    back_button = tk.Button(button_frame, text="Back", font=("Gabriola", 18), bg=Color_1, fg=Color_2, width=20,
command=lambda: back(delete_librarian_window,librarian_options))
    back_button.grid(row=5, column=1, padx=10, pady=5)

''' Simple BackEnd '''
```

```
# Create a frame
def Box(Main,Title):
    # Create a frame for layout
    frame = tk.Frame(Main, bg=Color_1)
    frame.pack(expand=True, fill='both')

    # Title label
    title_label = tk.Label(frame, text=Title, font=("Papyrus", 48, "bold"), fg=Color_5, bg=Color_1)
    title_label.pack(pady=20)

    # Create a frame for buttons
    button_frame = tk.Frame(frame, bg=Color_3)
    button_frame.pack(expand=True)

    return button_frame

# Function To Update The Result Display
def update_result_display(listbox, Statement):

    if Statement:
        listbox.insert(tk.END,Statement)
    else:
        listbox.insert(tk.END, "Nothing To Display")

# Function To Show A Message
def show_message(title, message):
    messagebox.showinfo(title, message)

# Function To Destory current Window And Go To Previous Window
def back(C_window,P_window):
    C_window.destroy()
    P_window()

''' The BackEnd '''

# Function to validate login
def validate_login(Name, ID, Password, Page):
    global Logged_In_User

    if Name and ID and Password:
        try:
            mydb = connect_db()
            mycursor = mydb.cursor()

            sql = "SELECT * FROM Librarian WHERE Name = %s AND Librarian_ID = %s AND Password = %s"

            val = (Name, ID, Password)

            mycursor.execute(sql, val)
            result = mycursor.fetchone()

            if result:
```

```
                Logged_In_User = Name
                log_activity(Logged_In_User, "Logged In", f"Librarian ID: {ID}")

                messagebox.showinfo("Success", "Login successful")
                librarian_options()

            else:
                messagebox.showerror("Error", "Invalid credentials")
        except sqlcon.Error as e:
            messagebox.showerror("Error", f"Database error: {e}")
        finally:
            mycursor.close()
            mydb.close()
    else:
        messagebox.showerror("Error", "Please fill in all fields.")
    Page.destroy()

# Function To Insert Values To Database
def insert(Table, Name, Address, Phone_No, Email, Password, listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results
    global Logged_In_User

    if Table == 'Borrower':
        # Validate Name and Phone Number
        if not Name or not Phone_No:
            Statement = "Please Fill In Name And Phone No."
            update_result_display(listbox, Statement)
            return

    elif Table == 'Librarian':
        # Validate Name, Phone Number and Password
        if not Name or not Phone_No or not Password:
            Statement = "Please Fill In Name, Phone No. and Password"
            update_result_display(listbox, Statement)
            return

    # Check if Phone Number is 10 digits
    if not Phone_No.isdigit() or len(Phone_No) != 10:
        Statement = "Error: Phone number must be exactly 10 digits."
        update_result_display(listbox, Statement)
        return

    try:
        mydb = connect_db()
        mycursor = mydb.cursor()

        if Table == 'Borrower':
            sql = "INSERT INTO Borrower (Name, Address, Phone, Email) VALUES (%s, %s, %s, %s)"
            val = (Name, Address, Phone_No, Email)

        elif Table == 'Librarian':
            sql = "INSERT INTO Librarian (Name, Address, Phone, Email, Password) VALUES (%s, %s, %s, %s, %s)"
```

```
            val = (Name, Address, Phone_No, Email, Password)

        mycursor.execute(sql, val)
        mydb.commit()

        id = mycursor.lastrowid

        log_activity(Logged_In_User, "Sign Up", f"Name:{Name} Librarian ID: {id}")

        Statement = f"Success! Your ID is [{id}]."
        update_result_display(listbox, Statement)

    except sqlcon.Error as e:
        Statement = f"Error: Database error: {e}"
        update_result_display(listbox, Statement)

    finally:
        mycursor.close()
        mydb.close()


def search_action(listbox, Entry, what_to_search):
    listbox.delete(0, tk.END)  # Clear Previous Results

    if Entry:
        if what_to_search == 'Title':
            sql = """
            SELECT
                Book.Book_ID,
                Book.Title,
                Book.Genre,
                Book_Authors.Author_Name,
                Book_Copies.No_Of_Copies
            FROM
                Book
            LEFT JOIN
                Book_Authors ON Book.Book_ID = Book_Authors.Book_ID
            LEFT JOIN
                Book_Copies ON Book.Book_ID = Book_Copies.Book_ID
            WHERE
                Title LIKE %s;
            """
        elif what_to_search == 'Genre':
            sql = """
            SELECT
                Book.Book_ID,
                Book.Title,
                Book.Genre,
                Book_Authors.Author_Name,
                Book_Copies.No_Of_Copies
            FROM
                Book
```

```
        LEFT JOIN
            Book_Authors ON Book.Book_ID = Book_Authors.Book_ID
        LEFT JOIN
            Book_Copies ON Book.Book_ID = Book_Copies.Book_ID
        WHERE
            Book.Genre LIKE %s;
        """
    elif what_to_search == 'Author':
        sql = """
        SELECT
            Book.Book_ID,
            Book.Title,
            Book.Genre,
            Book_Authors.Author_Name,
            Book_Copies.No_Of_Copies
        FROM
            Book
        LEFT JOIN
            Book_Authors ON Book.Book_ID = Book_Authors.Book_ID
        LEFT JOIN
            Book_Copies ON Book.Book_ID = Book_Copies.Book_ID
        WHERE
            Book_Authors.Author_Name LIKE %s;
        """
    elif what_to_search == 'Borrower':
        sql = "SELECT * FROM Borrower WHERE Name LIKE %s"
    elif what_to_search == 'Librarian':
        sql = "SELECT * FROM Librarian WHERE Name LIKE %s"
    else:
        update_result_display(listbox, "Invalid search criterion.")
        return

    values = ("%" + Entry + "%",)
    results = execute_fetch_results(sql, values)

    if results:
        # Iterate over results and display them in the listbox
        for row in results:
            if what_to_search in ['Title', 'Genre', 'Author']:
                Statement = (
                    f"> ID: {row[0]} \n"
                    f"> TITLE: {row[1]} \n"
                    f"> GENRE: {row[2]} \n"
                    f"> AUTHOR: {row[3]} \n"
                    f"> COPIES: {row[4]}"
                )
            elif what_to_search == 'Borrower':
                Statement = (
                    f"> Card No: {row[0]} \n"
                    f"> Name: {row[1]} \n"
                    f"> Address: {row[2]} \n"
                    f"> Email: {row[3]} \n"
```

```
                        f"> Phone: {row[4]}"
                    )
                elif what_to_search == 'Librarian':
                    Statement = (
                        f"> Librarian ID: {row[0]} \n"
                        f"> Name: {row[1]} \n"
                        f"> Address: {row[2]} \n"
                        f"> Email: {row[3]} \n"
                        f"> Phone: {row[4]}"
                    )
                else:
                    continue

                # Insert the statement into the listbox
                listbox.insert(tk.END, Statement)
        else:
            # Handle no results found
            listbox.insert(tk.END, "Nothing found matching the search criteria.")
    else:
        # Handle empty search entry
        listbox.insert(tk.END, "Please fill the search box.")

# Function to delete a borrower
def delete_action(id, listbox, role):
    listbox.delete(0, tk.END)  # Clear Previous Results

    if id:
        # SQL queries to fetch the name before deletion
        if role == 'Borrower':
            fetch_sql = "SELECT Name FROM Borrower WHERE Card_No = %s"
            delete_sql = "DELETE FROM Borrower WHERE Card_No = %s"
        elif role == 'Librarian':
            fetch_sql = "SELECT Name FROM Librarian WHERE Librarian_ID = %s"
            delete_sql = "DELETE FROM Librarian WHERE Librarian_ID = %s"
        else:
            update_result_display(listbox, "Invalid role specified.")
            return

        values = (id,)

        try:
            # Connect to the database
            connection = connect_db()
            cursor = connection.cursor()

            # Fetch the name
            cursor.execute(fetch_sql, values)
            result = cursor.fetchone()

            if result:
                name = result[0]  # Get the name from the result
                # Proceed to delete
```

```
                cursor.execute(delete_sql, values)
                connection.commit()

                if cursor.rowcount > 0:
                    # Display name and ID in the success message
                    statement = f"Success: {role} '{name}' with ID {id} deleted successfully."
                else:
                    statement = f"Error: Failed to delete {role} with ID {id}."
            else:
                # No matching record found
                statement = f"Error: {role} with ID {id} not found."

            update_result_display(listbox, statement)
        except sqlcon.Error as e:
            # Handle database errors
            statement = f"Error: Database error: {e}"
            update_result_display(listbox, statement)
        finally:
            # Close the cursor and connection
            cursor.close()
            connection.close()
    else:
        # ID not provided
        statement = "Error: Please provide a valid ID."
        update_result_display(listbox, statement)

# Function To View All Books
def view_all_books_action(listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results

    sql = """
    SELECT
        Book.Book_ID,
        Book.Title,
        Book.Genre,
        Book_Authors.Author_Name,
        Book_Copies.No_Of_Copies
    FROM
        Book
    LEFT JOIN
        Book_Authors ON Book.Book_ID = Book_Authors.Book_ID
    LEFT JOIN
        Book_Copies ON Book.Book_ID = Book_Copies.Book_ID;
    """

    try:
        results = execute_fetch_results(sql)

        if results:
            for book in results:
                statement = f"> ID: {book[0]} \n > TITLE: {book[1]} \n > GENRE: {book[2]} \n > AUTHOR: {book[3]} \n >
COPIES: {book[4]}\r\n"
```

```
                update_result_display(listbox, statement)
        else:
            statement = "No books found in the library."
            update_result_display(listbox, statement)

    except sqlcon.Error as e:
        statement = f"Error: Database error: {e}"
        update_result_display(listbox, statement)

# Function to Add a Book to the Database
def add_book_action(title, author, genre, copy, listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results

    if title and genre and author and copy:
        try:
            # Connect to the database
            mydb = connect_db()
            mycursor = mydb.cursor()

            # Step 1: Insert into Book table
            sql_1 = "INSERT INTO Book (Title, Genre) VALUES (%s, %s)"
            val_1 = (title, genre)
            mycursor.execute(sql_1, val_1)
            mydb.commit()

            # Get the last inserted Book_ID
            book_id = mycursor.lastrowid

            # Step 2: Insert into Book_Copies table
            sql_2 = "INSERT INTO Book_Copies (Book_ID, No_Of_Copies) VALUES (%s, %s)"
            val_2 = (book_id, copy)  # Use Book_ID as a foreign key
            mycursor.execute(sql_2, val_2)
            mydb.commit()

            # Step 3: Insert into Book_Authors table
            sql_3 = "INSERT INTO Book_Authors (Book_ID, Author_Name) VALUES (%s, %s)"
            val_3 = (book_id, author)  # Use Book_ID as a foreign key
            mycursor.execute(sql_3, val_3)
            mydb.commit()

            # Success message
            statement = f"Book added successfully! Book ID is {book_id}"
            update_result_display(listbox, statement)
        except sqlcon.Error as e:
            # Error handling
            statement = f"Error: Database error: {e}"
            update_result_display(listbox, statement)
        finally:
            # Close cursor and connection
            mycursor.close()
            mydb.close()
    else:
```

```
        # Validation error
        statement = "Error", "Please fill in all fields."
        update_result_display(listbox, statement)

# Function To Remove A Book
def remove_book_action(book_id, listbox):
    if book_id:
        try:
            # SQL query to delete a book based on its ID
            sql = "DELETE FROM Book WHERE Book_ID = %s"
            values = (book_id,)

            # Execute the query
            connection = connect_db()
            cursor = connection.cursor()
            cursor.execute(sql, values)
            connection.commit()

            if cursor.rowcount > 0:
                # Book successfully deleted
                statement = f"Success: Book with ID {book_id} has been removed."
            else:
                # Book not found
                statement = f"Info: No book found with ID {book_id}."

            # Update the listbox with the result
            update_result_display(listbox, statement)
        except sqlcon.Error as e:
            # Handle database errors
            statement = f"Error: Database error: {e}"
            update_result_display(listbox, statement)
        finally:
            # Close the cursor and the connection
            cursor.close()
            connection.close()
    else:
        # Validation error if Book ID is not provided
        statement = "Error: Please enter a valid Book ID."
        update_result_display(listbox, statement)

# Function To Issue A Book
def issue_book_action(borrower_id, book_id, listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results
    global Logged_In_User

    if borrower_id and book_id:
        try:
            # Fetch borrower details
            borrower_query = "SELECT Name FROM Borrower WHERE Card_No = %s"
            borrower_result = execute_fetch_results(borrower_query, (borrower_id,))

            # Fetch book details
```

```
            book_query = "SELECT Title FROM Book WHERE Book_ID = %s"
            book_result = execute_fetch_results(book_query, (book_id,))

            if borrower_result and book_result:
                borrower_name = borrower_result[0][0]
                book_title = book_result[0][0]

                # Check available copies
                copies_query = "SELECT No_Of_Copies FROM Book_Copies WHERE Book_ID = %s"
                copies_result = execute_fetch_results(copies_query, (book_id,))

                if copies_result and copies_result[0][0] > 0:  # Ensure copies are available
                    # Issue the book
                    issue_query = "INSERT INTO Book_Loans (Book_ID, Card_No, Date_Out, Due_Date) VALUES (%s, %s,
CURDATE(), CURDATE() + INTERVAL 14 DAY)"
                    execute_update(issue_query, (book_id, borrower_id))

                    # Update available copies
                    update_copies_query = "UPDATE Book_Copies SET No_Of_Copies = No_Of_Copies - 1 WHERE Book_ID = %s"
                    execute_update(update_copies_query, (book_id,))

                    # Success message with names
                    statement = f"Success: '{book_title}' has been issued to {borrower_name}."
                    log_activity(Logged_In_User, "Issued Book", f"Borrower: {borrower_name}, Book Name: {book_title}")
                else:
                    # No copies available
                    statement = f"Error: No available copies of '{book_title}'."
            else:
                # Borrower or book not found
                statement = "Error: Borrower or Book not found."
        except sqlcon.Error as e:
            # Handle database errors
            statement = f"Error: Database error: {e}"
    else:
        # Missing input
        statement = "Error: Please provide both Borrower ID and Book ID."

    # Update the listbox with the result
    update_result_display(listbox, statement)

# Function To Return A Book
def return_book_action(borrower_id, book_id, listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results
    global Logged_In_User

    if borrower_id and book_id:
        try:
            # Fetch borrower details
            borrower_query = "SELECT Name FROM Borrower WHERE Card_No = %s"
            borrower_result = execute_fetch_results(borrower_query, (borrower_id,))

            # Fetch book details
```

```
            book_query = "SELECT Title FROM Book WHERE Book_ID = %s"
            book_result = execute_fetch_results(book_query, (book_id,))

            if borrower_result and book_result:
                borrower_name = borrower_result[0][0]
                book_title = book_result[0][0]

                # Check if the book is issued to the borrower
                issued_query = "SELECT * FROM Book_Loans WHERE Book_ID = %s AND Card_No = %s"
                issued_result = execute_fetch_results(issued_query, (book_id, borrower_id))

                if issued_result:
                    # Return the book
                    delete_loan_query = "DELETE FROM Book_Loans WHERE Book_ID = %s AND Card_No = %s"
                    execute_update(delete_loan_query, (book_id, borrower_id))

                    # Update available copies
                    update_copies_query = "UPDATE Book_Copies SET No_Of_Copies = No_Of_Copies + 1 WHERE Book_ID = %s"
                    execute_update(update_copies_query, (book_id,))

                    # Success message with names
                    statement = f"Success: '{book_title}' has been returned by {borrower_name}."
                    log_activity(Logged_In_User, "Returned Book", f"Borrower: {borrower_name}, Book Name:
{book_title}")
                else:
                    # Book not issued to the borrower
                    statement = f"Error: '{book_title}' is not currently issued to {borrower_name}."
            else:
                # Borrower or book not found
                statement = "Error: Borrower or Book not found."
        except sqlcon.Error as e:
            # Handle database errors
            statement = f"Error: Database error: {e}"
    else:
        # Missing input
        statement = "Error: Please provide both Borrower ID and Book ID."

    # Update the listbox with the result
    update_result_display(listbox, statement)

# Function To View All Issued Books
def view_all_issued_books(listbox):
    sql = """
    SELECT
        Book_Loans.Book_ID,
        Book.Title,
        Borrower.Card_No,
        Borrower.Name,
        Book_Loans.Date_Out,
        Book_Loans.Due_Date
    FROM
        Book_Loans
```

```
    INNER JOIN
        Book ON Book_Loans.Book_ID = Book.Book_ID
    INNER JOIN
        Borrower ON Book_Loans.Card_No = Borrower.Card_No;
    """
    try:
        # Fetch results
        results = execute_fetch_results(sql)

        # Clear listbox
        listbox.delete(0, tk.END)

        if results:
            for row in results:
                statement = f"Book ID: {row[0]}, Title: {row[1]}, Borrower ID: {row[2]}, Name: {row[3]}, Issued:
{row[4]}, Due: {row[5]}"
                listbox.insert(tk.END, statement)
        else:
            listbox.insert(tk.END, "No issued books found.")
    except sqlcon.Error as e:
        listbox.insert(tk.END, f"Error: Database error: {e}")

# Function To Manage Book Copies
def manage_book_copies_action(book_id, copies_to_add, listbox):
    listbox.delete(0, tk.END)  # Clear Previous Results

    if book_id and copies_to_add:
        try:
            # Update the number of copies
            sql = "UPDATE Book_Copies SET No_Of_Copies = No_Of_Copies + %s WHERE Book_ID = %s"
            values = (copies_to_add, book_id)

            connection = connect_db()
            cursor = connection.cursor()
            cursor.execute(sql, values)
            connection.commit()

            if cursor.rowcount > 0:
                statement = f"Success: Updated copies for Book ID {book_id}. Added {copies_to_add} copies."
            else:
                statement = f"Error: Book ID {book_id} not found."
            update_result_display(listbox, statement)
        except sqlcon.Error as e:
            statement = f"Error: Database error: {e}"
            update_result_display(listbox, statement)
        finally:
            cursor.close()
            connection.close()
    else:
        statement = "Error: Please provide both Book ID and number of copies."
        update_result_display(listbox, statement)
```

```
# Function To Manage Fine
def calculate_fines(listbox):
    sql = """
    SELECT
        Borrower.Card_No,
        Borrower.Name,
        Book.Title,
        DATEDIFF(CURDATE(), Book_Loans.Due_Date) AS Overdue_Days
    FROM
        Book_Loans
    INNER JOIN
        Borrower ON Book_Loans.Card_No = Borrower.Card_No
    INNER JOIN
        Book ON Book_Loans.Book_ID = Book.Book_ID
    WHERE
        Book_Loans.Due_Date < CURDATE();
    """
    try:
        results = execute_fetch_results(sql)

        # Clear listbox
        listbox.delete(0, tk.END)

        if results:
            for row in results:
                overdue_days = row[3]
                fine = overdue_days * 5  # Example: 5 currency units per day
                statement = f"Borrower: {row[1]} ({row[0]}), Book: {row[2]}, Overdue Days: {overdue_days}, Fine:
₹{fine}"
                listbox.insert(tk.END, statement)
        else:
            listbox.insert(tk.END, "No overdue books found.")
    except sqlcon.Error as e:
        listbox.insert(tk.END, f"Error: Database error: {e}")

# Function to log activities
def log_activity(user_name, action, details=""):
    # Get the current timestamp
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Set the desired directory and file name
    os.makedirs(Desired_Folder, exist_ok=True)  # Ensure the folder exists
    log_file_name = os.path.join(Desired_Folder, "Activity_Log.csv")

    # Check if the file exists
    file_exists = os.path.exists(log_file_name)

    # Write the log entry to the CSV file
    with open(log_file_name, "a", newline="") as log_file:
        csv_writer = csv.writer(log_file)

        # Write headers if file doesn't exist
```

```
        if not file_exists:
            csv_writer.writerow(["Timestamp", "User", "Action", "Details"])

        # Write the log entry
        csv_writer.writerow([timestamp, user_name, action, details])

    # print(f"Log written to: {log_file_name}")

# Initiation Of The Code #

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