A group of people in a library

Description automatically generated



**[2024-2025]**

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 External Examiner

Head of the department

(Department of Computer Science)

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

|  |  |
| --- | --- |
| **Content** | **Page No.** |
| 1. Introduction | - |
| * + Problem Definition | 05 |
| * + Python Introduction | 06 |
| * + MySQL Introduction | 07 |
| 1. About Project | 08 |
| 1. System Requirements | - |
| * Software Requirements | 09 |
| * Hardware Requirements | 09 |
| 1. Table design | 10 |
| 1. Preview Of Program | 12 |
| 1. Conclusion | 22 |
| 1. References | - |
| * Bibliography | 23 |
| * Webliography | 23 |
| 1. Appendix (Source Code) |  |
| * Database Code | 24 |
| * Main Project Code | 30 |

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](https://en.wikipedia.org/wiki/Guido_van_Rossum) [Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), which was inspired by [SETL](https://en.wikipedia.org/wiki/SETL), capable of [exception](https://en.wikipedia.org/wiki/Exception_handling) [handling](https://en.wikipedia.org/wiki/Exception_handling) (from the start plus new capabilities in Python 3.11) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) 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](https://en.wikipedia.org/wiki/Benevolent_dictator_for_life) [dictator for life](https://en.wikipedia.org/wiki/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.](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) [Object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of their features support functional programming and aspect-oriented [programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including [meta-programming](https://en.wikipedia.org/wiki/Metaprogramming) and [meta-objects](https://en.wikipedia.org/wiki/Metaobject)). Many other paradigms are supported via extensions, including [design by contract.](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing) and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management.](https://en.wikipedia.org/wiki/Memory_management) It uses dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

A close up of a logo

Description automatically generated

* SQL INTRODUCTION

**SQL HISTORY**: SQL was initially developed at [IBM](https://en.wikipedia.org/wiki/IBM) by [Donald D.](https://en.wikipedia.org/wiki/Donald_D._Chamberlin) [Chamberlin](https://en.wikipedia.org/wiki/Donald_D._Chamberlin) and [Raymond F. Boyce](https://en.wikipedia.org/wiki/Raymond_F._Boyce) after learning about the relational model from [Edgar F.](https://en.wikipedia.org/wiki/Edgar_F._Codd) [Codd](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/IBM_System_R), which a group at [IBM San Jose](https://en.wikipedia.org/wiki/IBM_Almaden_Research_Center) [Research Laboratory](https://en.wikipedia.org/wiki/IBM_Almaden_Research_Center) 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, they began work on a sequel to SQUARE. The name SEQUEL was later changed to SQL (dropping the vowels) because "SEQUEL" was a [trademark](https://en.wikipedia.org/wiki/Trademark) of the [UK-based](https://en.wikipedia.org/wiki/United_Kingdom) [Hawker](https://en.wikipedia.org/wiki/Hawker_Siddeley) [Siddeley](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Domain-specific_language) used in programming and designed for managing data held in a [relational database management](https://en.wikipedia.org/wiki/Relational_database_management_system) [system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS), or for stream processing in a [relational data stream management](https://en.wikipedia.org/wiki/Relational_data_stream_management_system) [system](https://en.wikipedia.org/wiki/Relational_data_stream_management_system) (RDSMS). It is particularly useful in handling [structured data,](https://en.wikipedia.org/wiki/Data_model) i.e. data incorporating relations among entities and variables.

SQL offers two main advantages over older read–write [APIs](https://en.wikipedia.org/wiki/API) such as [ISAM](https://en.wikipedia.org/wiki/ISAM) or [VSAM](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Database_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

A black screen with white text

Description automatically generated

* BOOK AUTHORS TABLE

A black screen with white text

Description automatically generated

* BOOK COPIES TABLE

A black screen with white text

Description automatically generated

* BOOK LOANS TABLE

A black screen with white text

Description automatically generated

* BORROWER TABLE

A screen shot of a computer screen

Description automatically generated

* LIBRARIAN TABLE

A black screen with white text

Description automatically generated

PREVIEW OF PROGRAM

--------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

Description automatically generated

--------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

Description automatically generated

--------------------------------------------------------------------------------------------------------------------------

A screenshot of a computer

Description automatically generated

--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------

A computer screen shot of a computer screen

Description automatically generated

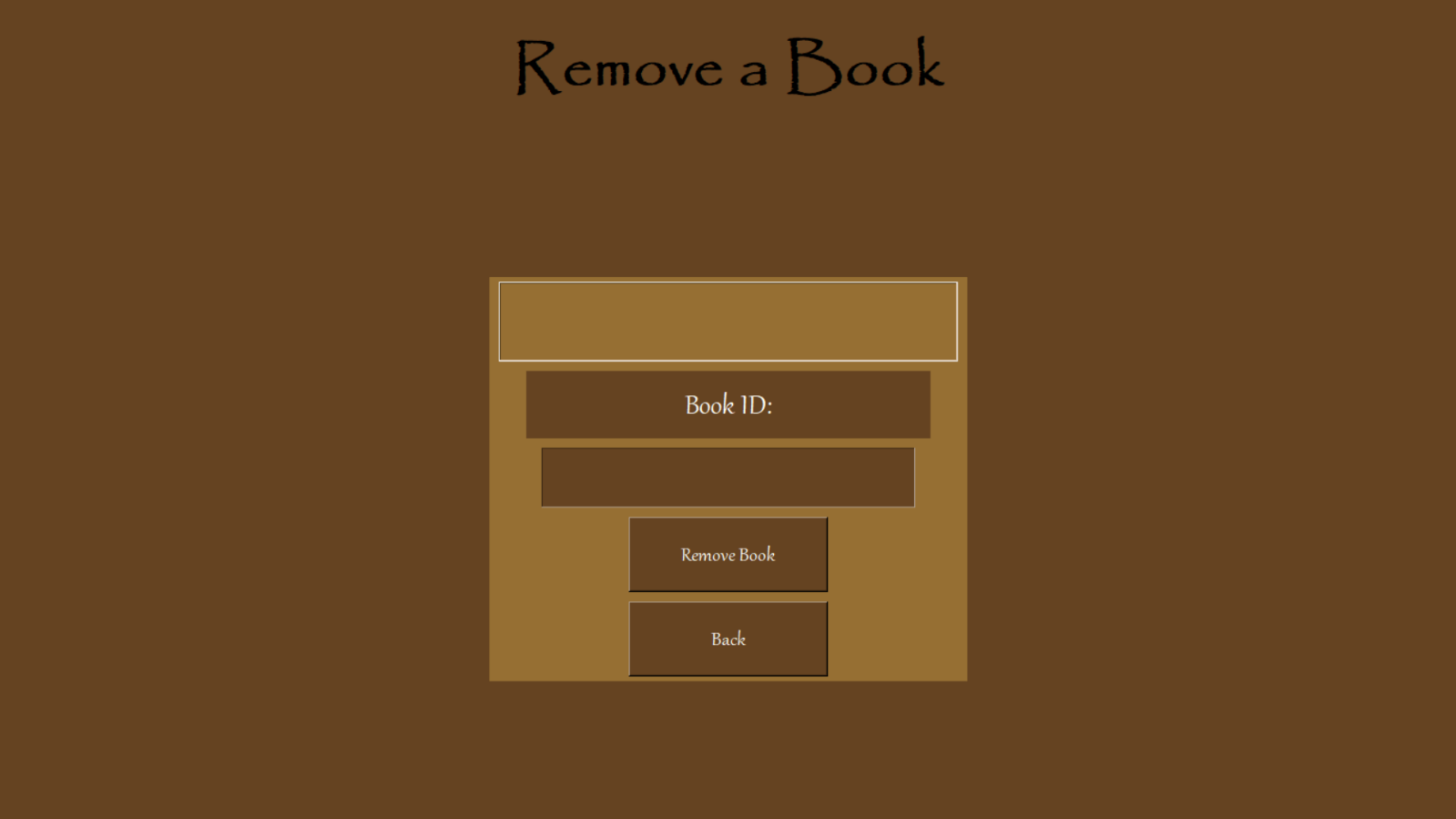
--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



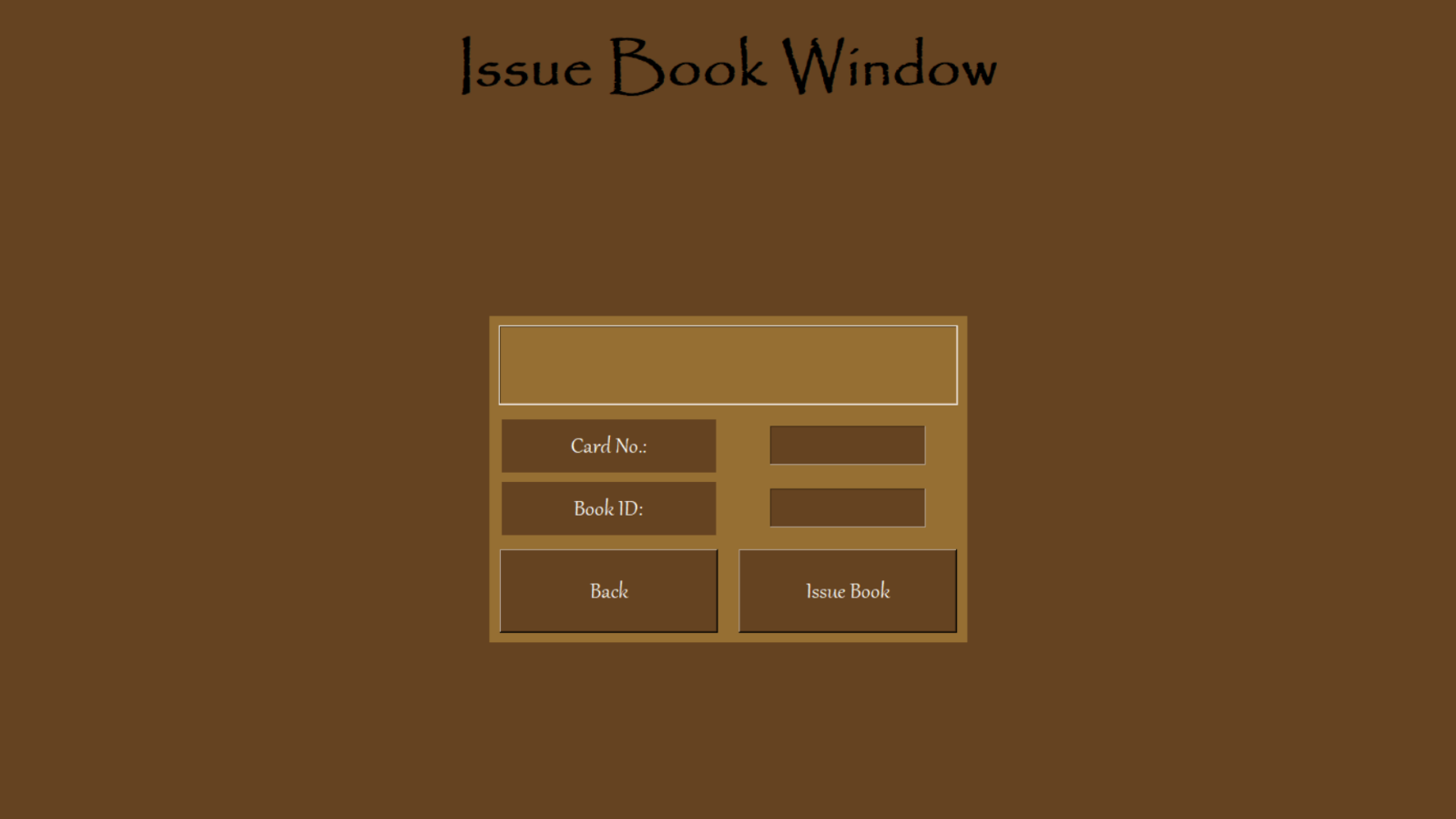
--------------------------------------------------------------------------------------------------------------------------



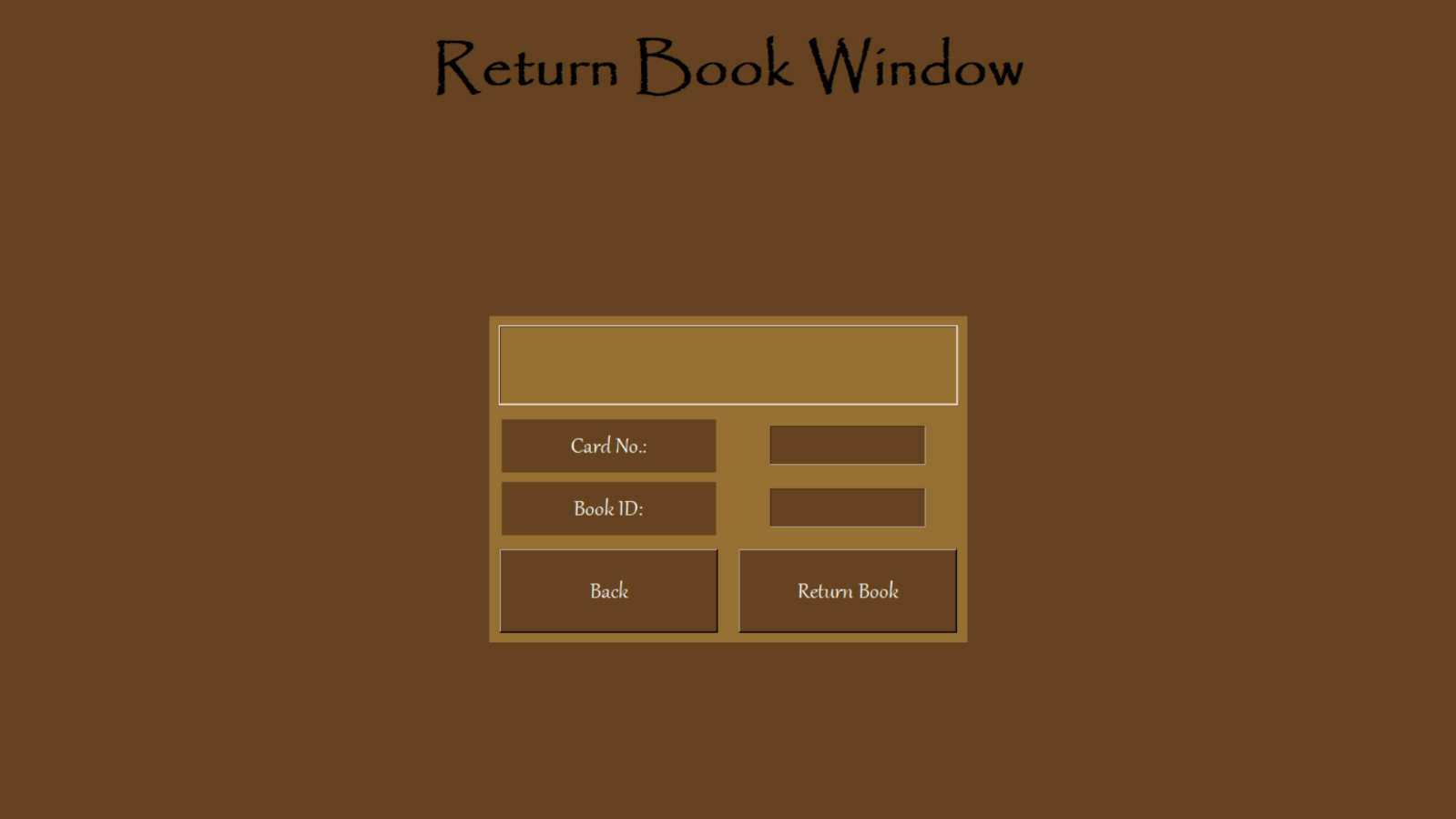
--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



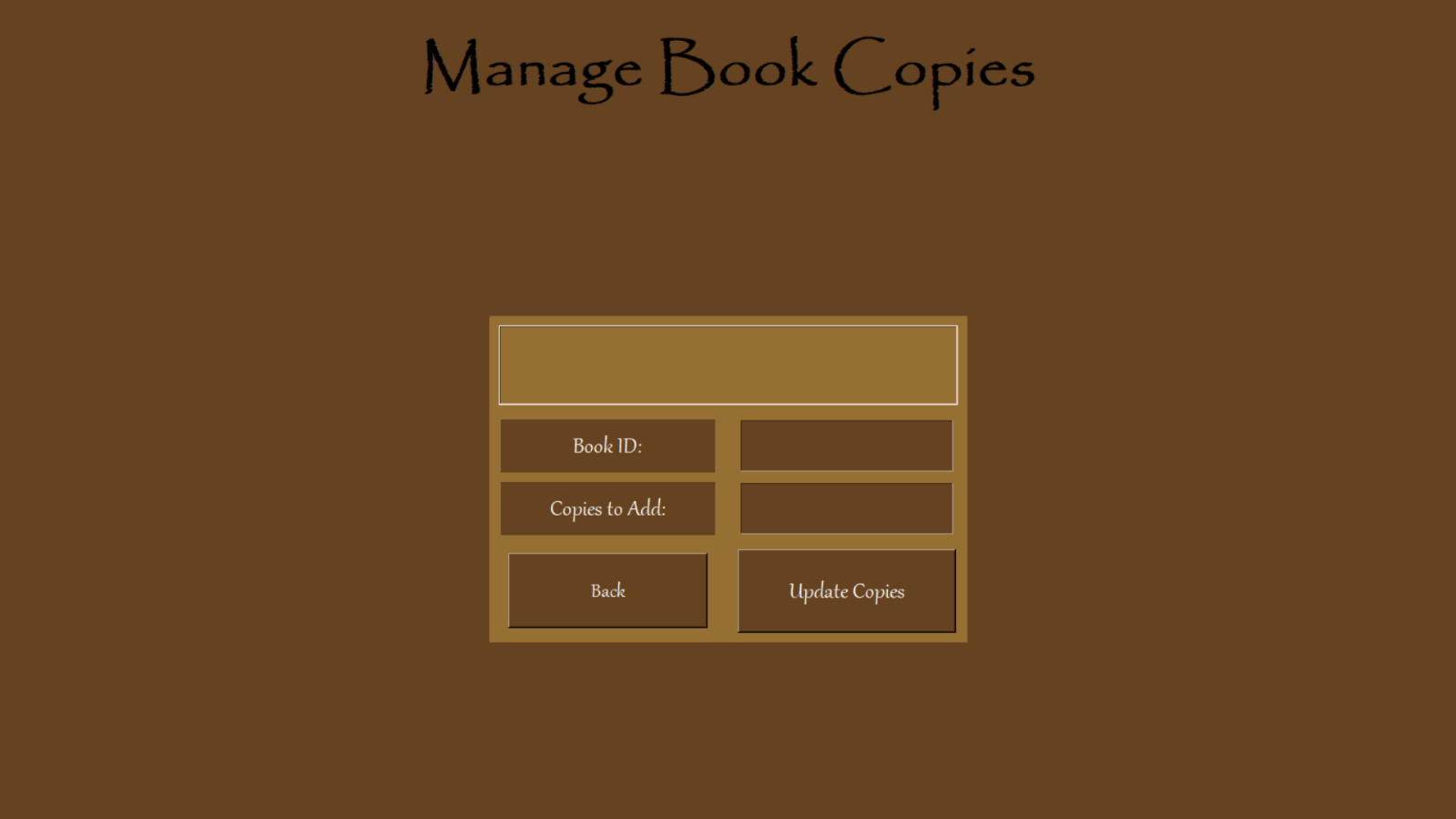
--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------------------------------------



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()