# LIBRARY MANAGEMENT SYSTEM

**Group-17**

**Chandan Byanna Venkatesh – 02167738 – 05**
**China Subba Rao Koduri – 02190693 – 17**
**Abhishek Reddy Surkanti – 02187989 – 43**

## Problem Statement

In traditional library systems, managing books, users, and transactions manually can be time-consuming and error-prone. The lack of an automated, efficient system often leads to difficulties in tracking books, maintaining user records, and managing borrow and return transactions. This project aims to address these challenges by developing a Library Management System that provides an easy-to-use interface for library staff and users. The system should streamline library operations, reduce manual errors, and enhance the overall management of library resources, ensuring better control over books and user transactions while promoting an organized digital library environment.

**Technologies Used**

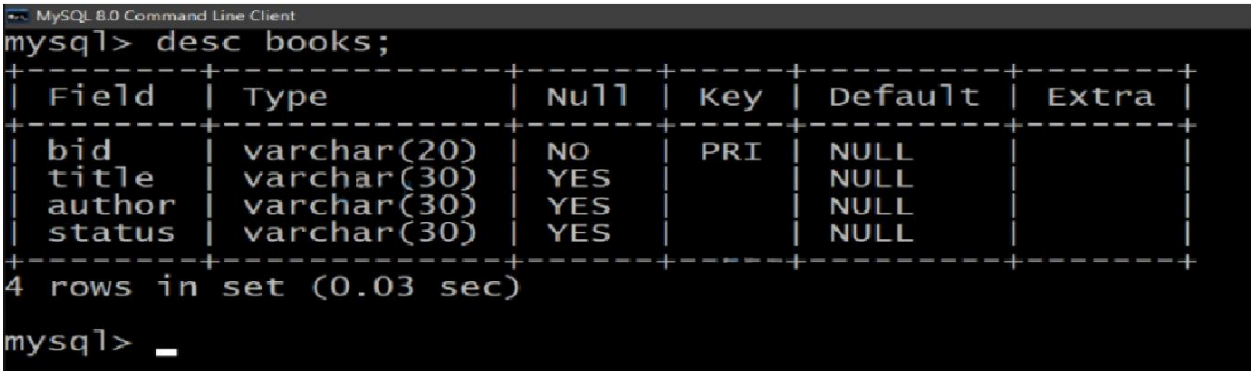
- **Python**: Backend scripting and logic control
- **Tkinter**: GUI development for user interaction
- **MySQL**: Persistent storage and data management
- **PyMySQL**: Database connector for executing SQL commands
- **Pillow (PIL)**: Used for handling images in the GUI

# Description of Tables

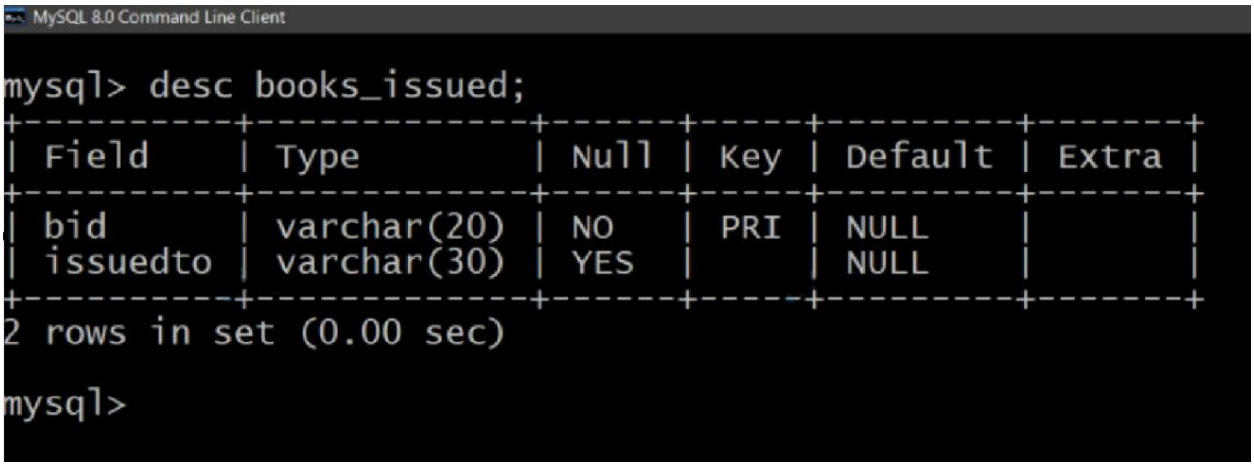
# Create Tables

```
1.  create database db;
2.
3.  create table books(bid varchar(20) primary key, title varchar(30), author varchar(30), status
    varchar(30));
4.
5.  create table books_issued(bid varchar(20) primary key, issuedto varchar(30));
```

### 1. Books



**2. issued_books**




# Library Management Project Code

Let's start the detailed discussion of each and every file of our library management system python project:

## 1. main.py

### 1.1. Importing the Modules

To use the Tkinter we need to import the Tkinter module. As stated above, we have imported each file so that we can make function calls from our main file.

**Code:**

```
1.  from tkinter import *
2.  from PIL import ImageTk,Image #PIL -> Pillow
3.  import pymysql
4.  from tkinter import messagebox
5.  from AddBook import *
6.  from DeleteBook import *
7.  from ViewBooks import *
8.  from IssueBook import *
```

### 1.2. Connecting to the MySql server

Now we will connect to the server with the correct credentials associated with the MySql server installed on our system.

**Code:**

```
mypass = "root" #use your own password
mydatabase="db" #The database name

con = pymysql.connect (host="localhost",user="root",password=mypass,database=mydatabase)
#root is the username here

cur = con.cursor() #cur -> cursor
```

### 1.3. Designing the Window

Now we will design the project window and add a background image. Make sure to keep the image in the same directory as the project is in order to avoid discrepancies.

**Code:**

```
root = Tk()
root.title("Library")
root.minsize(width=400,height=400)
root.geometry("600x500")
```

**Explanation:**

The above code sets the title of the library project window as ‘Library’. When you run the above code, you will get a window.

### 1.4. Adding a Background Image

### Code:

```
same=True
n=0.25

# Adding a background image
background_image =Image.open("lib.jpg")
[imageSizeWidth, imageSizeHeight] = background_image.size

newImageSizeWidth = int(imageSizeWidth*n)
if same:
    newImageSizeHeight = int(imageSizeHeight*n)
else:
    newImageSizeHeight = int(imageSizeHeight/n)

background_image =
background_image.resize((newImageSizeWidth,newImageSizeHeight),Image.ANTIALIAS)
img = ImageTk.PhotoImage(background_image)
Canvas1 = Canvas(root)
Canvas1.create_image(300,340,image = img)
Canvas1.config(bg="white",width = newImageSizeWidth, height = newImageSizeHeight)
Canvas1.pack(expand=True,fill=BOTH)
```

### Explanation:

We store our image in **background_image** with the help of **.open()** method. We fetch the image dimensions and adjust the image size according to our window size.

**newImageHeight** and **newImageWidth** contains the adjusted image dimensions.

Now we resize the image using **.resize()** method using the new dimensions.The **.PhotoImage()** method is used to display images (either grayscale or true color images) in labels, buttons, canvases, and text widgets.

We create the image on the **canvas1** using **.create_image()** method. We use **.pack()** method to organize widgets in blocks before placing them in the parent widget.

### 1.5. Setting up the Head Frame

### Code:

```
1.  headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
2.  headingFrame1.place(relx=0.2,rely=0.1,relwidth=0.6,relheight=0.16)
3.  headingLabel = Label(headingFrame1, text="Welcome to \n DataFlair Library", bg='black',
    fg='white', font=('Courier',15))
4.  headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)
```

### Explanation:

We create a frame that will hold our Label wiz **headingLabel**. We increase the size and alter the font by passing one more parameter in the **Label** method wiz **font**.

### 1.6. Adding the Buttons

### Code:

```
1.  btn1 = Button(root,text="Add Book Details",bg='black', fg='white', command=addBook)
2.  btn1.place(relx=0.28,rely=0.4, relwidth=0.45,relheight=0.1)
3.
4.  btn2 = Button(root,text="Delete Book",bg='black', fg='white', command=delete)
5.  btn2.place(relx=0.28,rely=0.5, relwidth=0.45,relheight=0.1)
6.
7.  btn3 = Button(root,text="View Book List",bg='black', fg='white', command=View)
8.  btn3.place(relx=0.28,rely=0.6, relwidth=0.45,relheight=0.1)
9.
10. btn4 = Button(root,text="Issue Book to Student",bg='black', fg='white', command = issueBook)
11. btn4.place(relx=0.28,rely=0.7, relwidth=0.45,relheight=0.1)
12.
13. btn5 = Button(root,text="Return Book",bg='black', fg='white', command = returnBook)
14. btn5.place(relx=0.28,rely=0.8, relwidth=0.45,relheight=0.1)
15. root.mainloop()
```

### Explanation:

The above code adds buttons to the window frame.

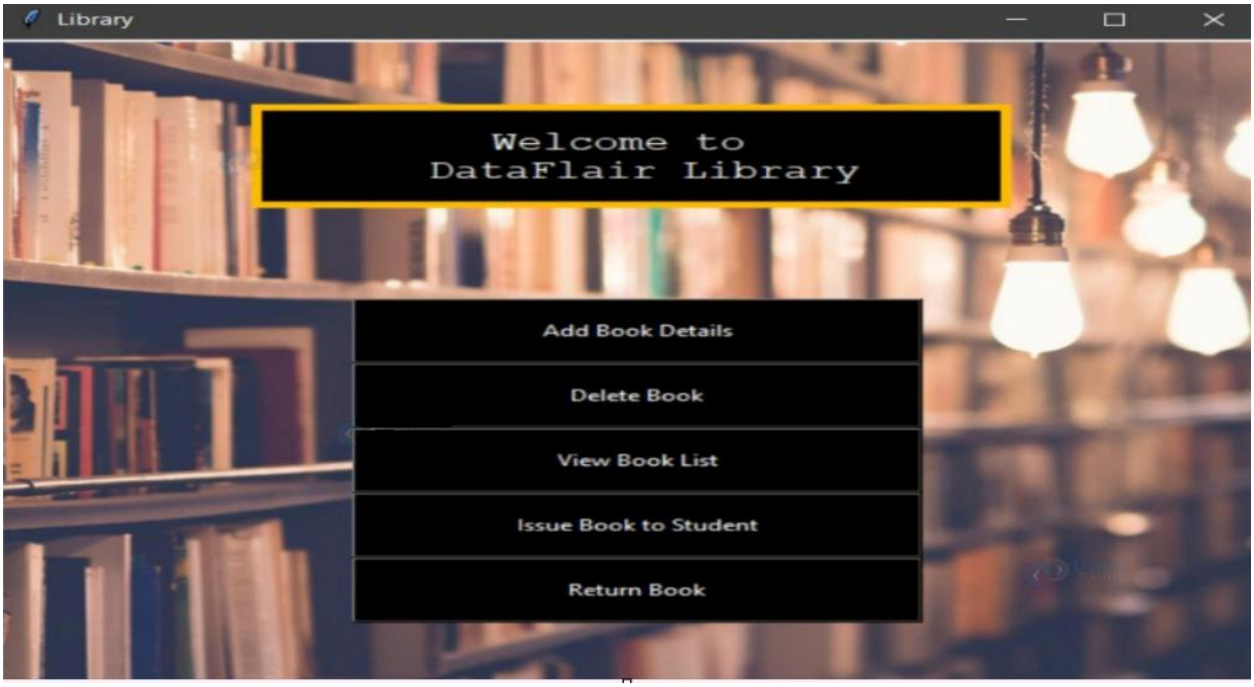
**AddBook Details :**

btn1 stores the button created on **root** with **text = 'AddBook Details'**. As soon as someone clicks this button, we call the function addBook defined in the **AddBook.py**. We call a function by specifying the **command** parameter equal to the **name of the function**.

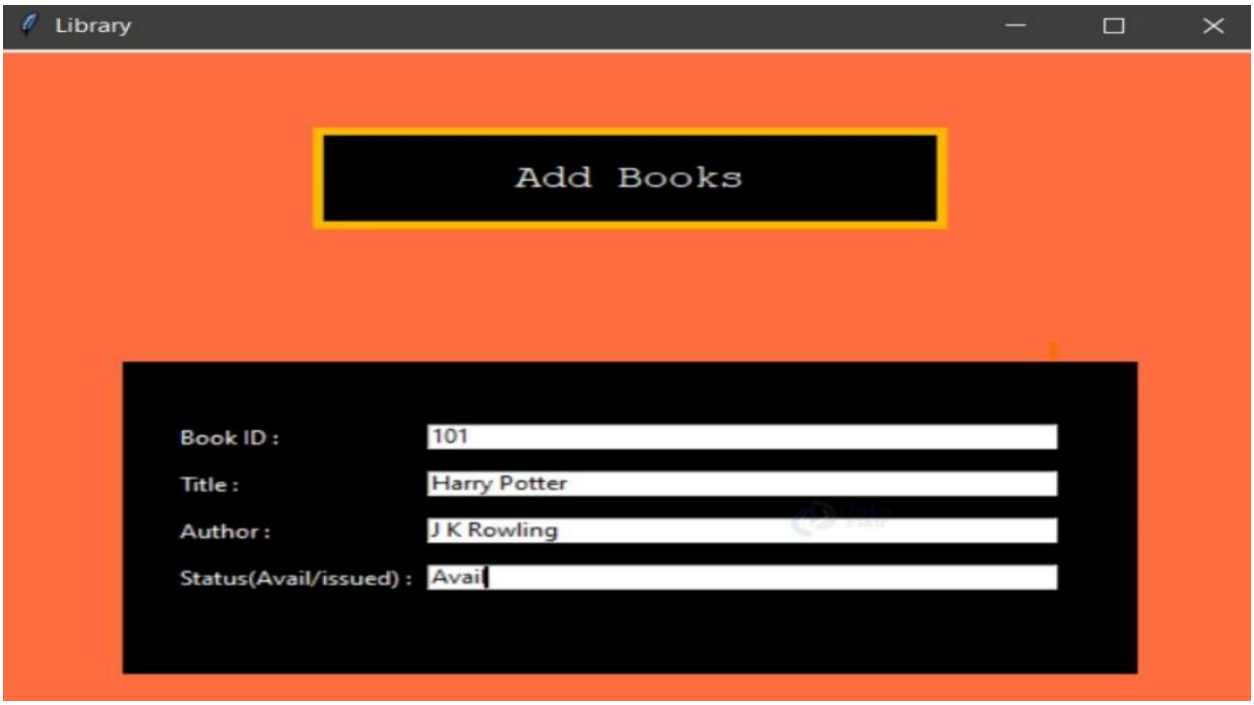
We place this button using the **.place()** method by defining the position as well as dimensions of the button.

Similarly, we define other buttons using the **Button** method and keep placing them by making minor changes in the **rely** parameter. You can notice that we are increasing it by 0.1 every time we define a new button.

**Output:**



2. AddBook.py




### 2.1. Importing the necessary modules

**Code:**

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

### 2.2. Function – bookRegister()

This function executes an SQL command to insert data into the table and commit the changes.

**Code:**

```
def bookRegister():

    bid = bookInfo1.get()
    title = bookInfo2.get()
    author = bookInfo3.get()
    status = bookInfo4.get()
    status = status.lower()

    insertBooks = "insert into "+bookTable+" values
('"+bid+"','"+title+"','"+author+"','"+status+"')"
    try:
        cur.execute(insertBooks)
        con.commit()
        messagebox.showinfo('Success',"Book added successfully")
    except:
        messagebox.showinfo("Error","Can't add data into Database")

    print(bid)
    print(title)
    print(author)
    print(status)
    root.destroy()
```

**Explanation:**

We fetch the data in the input field by **.get()** method. Hence after fetching each of the data fields value we are ready to execute an SQL command to insert the data.

### 2.3. Function – addBook()

This function connects to the MySql server and creates a window for accommodating new text fields. We fetch details of a new book from the user and then call **bookRegister()** function to register the books into the table.

**Code:**

```
def addBook():

    global bookInfo1 ,bookInfo2, bookInfo3, bookInfo4, Canvas1, con, cur, bookTable, root

    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")


    mypass = "root"
    mydatabase="db"

    con = pymysql.connect( host="localhost",user="root",password=mypass,database=mydatabase)
    cur = con.cursor()

    # Enter Table Names here
    bookTable = "books" # Book Table

    Canvas1 = Canvas(root)

    Canvas1.config(bg="#ff6e40")
    Canvas1.pack(expand=True,fill=BOTH)

    headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
    headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)

    headingLabel = Label(headingFrame1, text="Add Books", bg='black', fg='white', font=
('Courier',15))
    headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)


    labelFrame = Frame(root,bg='black')
    labelFrame.place(relx=0.1,rely=0.4,relwidth=0.8,relheight=0.4)

    # Book ID
    lb1 = Label(labelFrame,text="Book ID : ", bg='black', fg='white')
    lb1.place(relx=0.05,rely=0.2, relheight=0.08)

   bookInfo1 = Entry(labelFrame)
   bookInfo1.place(relx=0.3,rely=0.2, relwidth=0.62, relheight=0.08)

   # Title
   lb2 = Label(labelFrame,text="Title : ", bg='black', fg='white')
   lb2.place(relx=0.05,rely=0.35, relheight=0.08)

   bookInfo2 = Entry(labelFrame)
   bookInfo2.place(relx=0.3,rely=0.35, relwidth=0.62, relheight=0.08)

   # Book Author
   lb3 = Label(labelFrame,text="Author : ", bg='black', fg='white')
   lb3.place(relx=0.05,rely=0.50, relheight=0.08)

   bookInfo3 = Entry(labelFrame)
   bookInfo3.place(relx=0.3,rely=0.50, relwidth=0.62, relheight=0.08)

   # Book Status
   lb4 = Label(labelFrame,text="Status(Avail/issued) : ", bg='black', fg='white')
   lb4.place(relx=0.05,rely=0.65, relheight=0.08)

   bookInfo4 = Entry(labelFrame)
   bookInfo4.place(relx=0.3,rely=0.65, relwidth=0.62, relheight=0.08)

   #Submit Button
   SubmitBtn = Button(root,text="SUBMIT",bg='#d1ccc0', fg='black',command=bookRegister)
   SubmitBtn.place(relx=0.28,rely=0.9, relwidth=0.18,relheight=0.08)

   quitBtn = Button(root,text="Quit",bg='#f7f1e3', fg='black',       command=root.destroy)
   quitBtn.place(relx=0.53,rely=0.9, relwidth=0.18,relheight=0.08)

   root.mainloop()
```

## Variables

**bookInfo1** – contains book ID
**bookInfo2** – contains Title of the book
**bookInfo3** – contains Author of the book
**bookInfo4** – contains status of the book (available or issued)
**Con** – MySql console

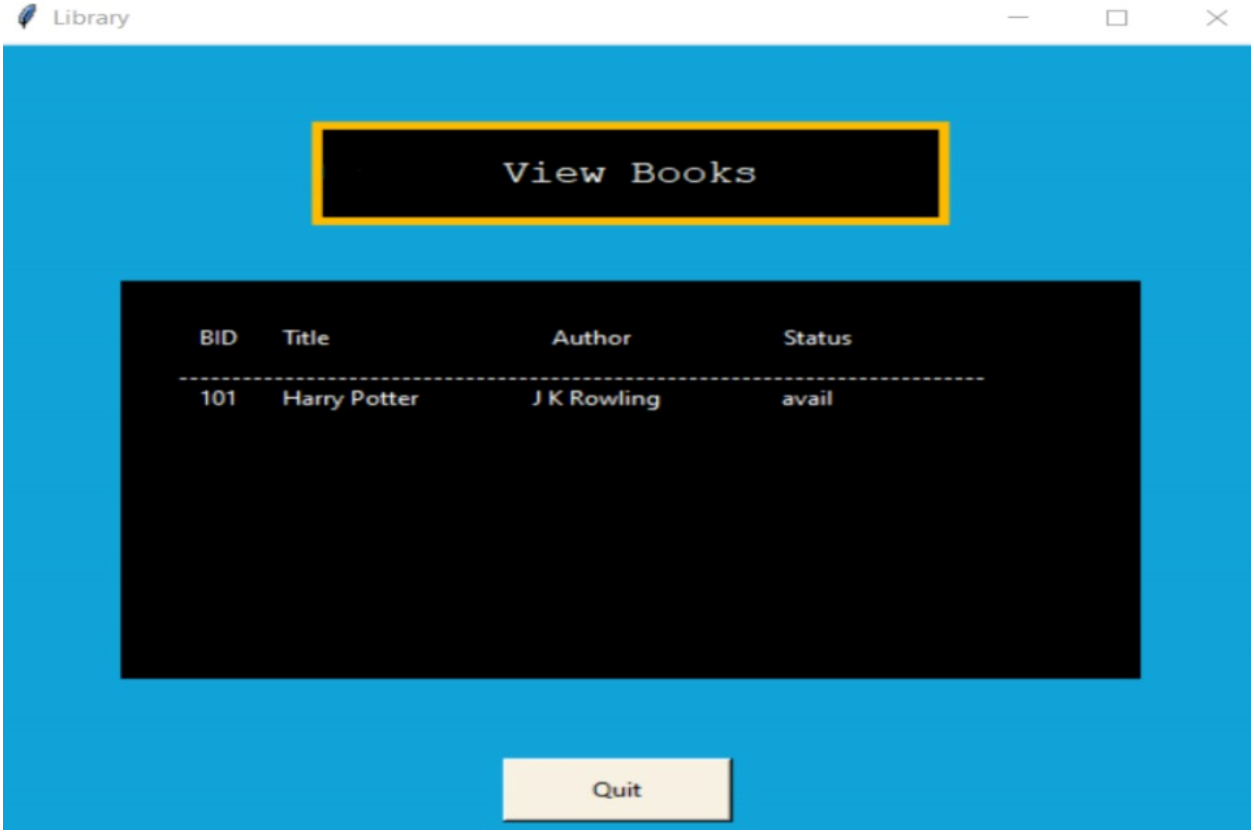
**Cur** – cursor of the console

**Buttons**

**Submit** – to commit the changes

**Exit**

## 3. ViewBooks.py




### 3.1. Importing the necessary modules

**Code:**

```
1.  from tkinter import *
2.  from PIL import ImageTk,Image
3.  from tkinter import messagebox
4.  import pymysql
```

## 3.2. Connection to MySql server

## Code:

```
mypass = "root"
mydatabase="db"

con = pymysql.connect( host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()

# Enter Table Names here
bookTable = "books"
```

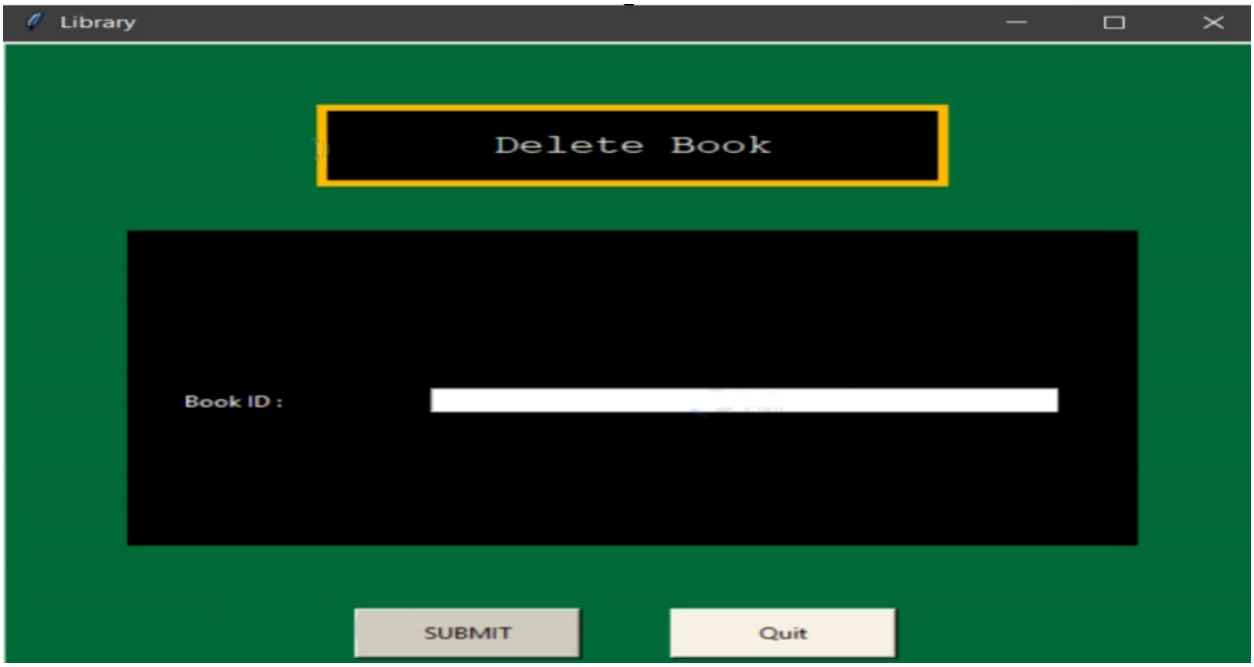
## 3.3. Function – View()

This function in our library project creates a window for displaying the records in the table.

## Code:

```
def View():

    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")

    Canvas1 = Canvas(root)
    Canvas1.config(bg="#12a4d9")
    Canvas1.pack(expand=True,fill=BOTH)

    headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
    headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)

    headingLabel = Label(headingFrame1, text="View Books", bg='black', fg='white', font =
('Courier',15))

    headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)

    labelFrame = Frame(root,bg='black')
    labelFrame.place(relx=0.1,rely=0.3,relwidth=0.8,relheight=0.5)
    y = 0.25

    Label(labelFrame, text="%-10s%-40s%-30s%-20s"%('BID','Title','Author','Status'),
    bg='black',fg='white').place(relx=0.07,rely=0.1)
    Label(labelFrame, text = "----------------------------------------------------------------------
----------",bg='black',fg='white').place (relx=0.05,rely=0.2)
    getBooks = "select * from "+bookTable
    try:
        cur.execute(getBooks)
        con.commit()

```

4. DeleteBook.py




## 4.1. Importing the necessary modules

## Code:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

## 4.2. Connect to MySql Server

## Code:

```
mypass = "root"
mydatabase="db"

con = pymysql.connect (host="localhost",user="root",password=mypass,database=mydatabase)
cur = con.cursor()

# Enter Table Names here
issueTable = "books_issued"
bookTable = "books" #Book Table
```

### 4.3. Function – deleteBook()

This function  checks if the bid (book id) exists in the book table and if it does, it executes the necessary command to remove it.

### Code:

```
def deleteBook():

    bid = bookInfo1.get()

    deleteSql = "delete from "+bookTable+" where bid = '"+bid+"'"
    deleteIssue = "delete from "+issueTable+" where bid = '"+bid+"'"

    try:
        cur.execute(deleteSql)
        con.commit()
        cur.execute(deleteIssue)
        con.commit()

        messagebox.showinfo('Success',"Book Record Deleted Successfully")

    except:
        messagebox.showinfo("Please check Book ID")

    print(bid)

    bookInfo1.delete(0, END)
    root.destroy()
```

5. IssueBook.py

## 5.1. Importing the necessary modules

## Code:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

## 5.2. Connecting to the MySql server

## Code:

```
mypass = "root"
mydatabase="db"

con = pymysql.connect(host="localhost",user="root", password=mypass,database=mydatabase)
cur = con.cursor()


# Enter Table Names here
issueTable = "books_issued"
bookTable = "books"

allBid = [] #To store all the Book ID's
```

## 5.3. Function – issue()

## Code:

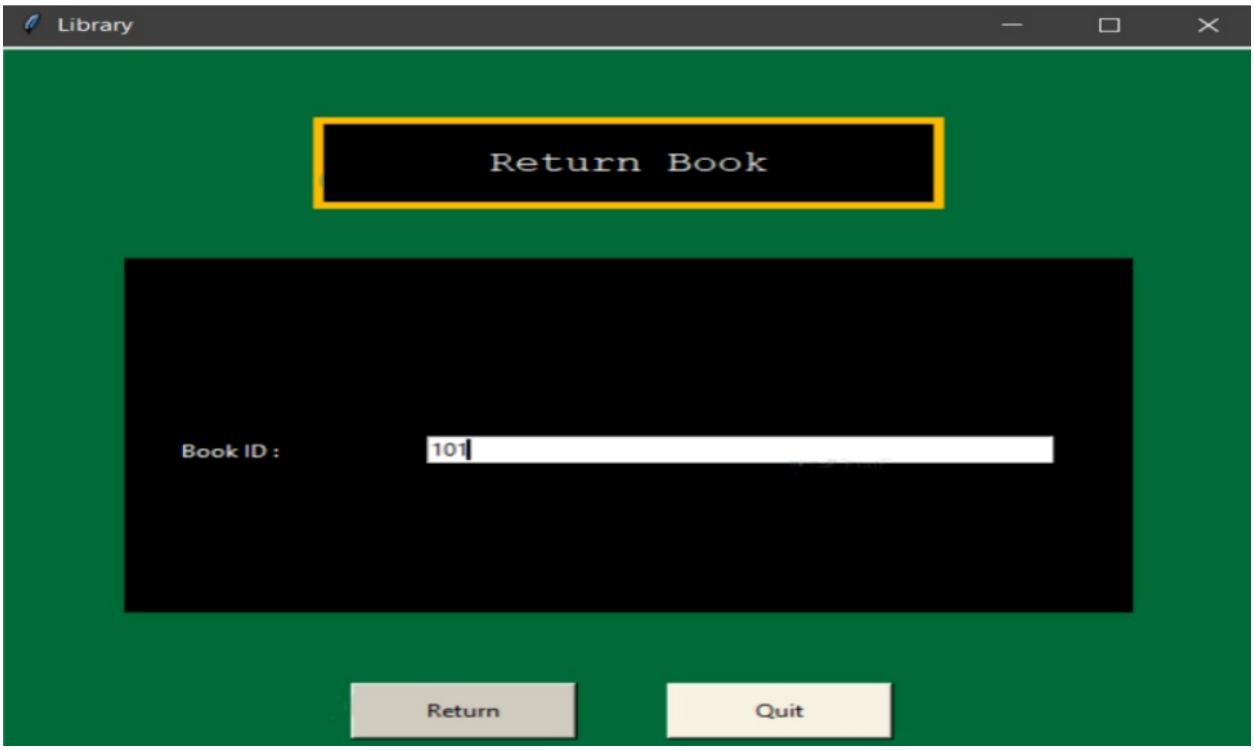
```
def issue():

    global issueBtn,labelFrame,lb1,inf1,inf2,quitBtn,root,Canvas1,status

    bid = inf1.get()
    issueto = inf2.get()

    issueBtn.destroy()
    labelFrame.destroy()
    lb1.destroy()
    inf1.destroy()
    inf2.destroy()


    extractBid = "select bid from "+bookTable
    try:
        cur.execute(extractBid)
        con.commit()
        for i in cur:
            allBid.append(i[0])

        if bid in allBid:
            checkAvail = "select status from "+bookTable+" where bid = '"+bid+"'"
            cur.execute(checkAvail)
            con.commit()
            for i in cur:
                check = i[0]
```

```
                if check == 'avail':
                    status = True
                else:
                    status = False

            else:
                messagebox.showinfo("Error","Book ID not present")
        except:
            messagebox.showinfo("Error","Can't fetch Book IDs")

        issueSql = "insert into "+issueTable+" values ('"+bid+"','"+issueto+"')"
        show = "select * from "+issueTable

        updateStatus = "update "+bookTable+" set status = 'issued' where bid = '"+bid+"'"
        try:
            if bid in allBid and status == True:
                cur.execute(issueSql)
                con.commit()
                cur.execute(updateStatus)
                con.commit()
                messagebox.showinfo('Success',"Book Issued Successfully")
                root.destroy()
            else:
                allBid.clear()
                messagebox.showinfo('Message',"Book Already Issued")
                root.destroy()
                return
        except:
            messagebox.showinfo("Search Error","The value entered is wrong, Try again")

        print(bid)
        print(issueto)

        allBid.clear()
```

## 5.4. Function – issueBook()Code:

```
def issueBook():

    global issueBtn,labelFrame,lb1,inf1,inf2,quitBtn,root,Canvas1,status

    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")

    Canvas1 = Canvas(root)
    Canvas1.config(bg="#D6ED17")
    Canvas1.pack(expand=True,fill=BOTH)

    headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
    headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)

    headingLabel = Label(headingFrame1, text="Issue Book", bg='black', fg='white', font=
('Courier',15))
    headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)

    labelFrame = Frame(root,bg='black')
    labelFrame.place(relx=0.1,rely=0.3,relwidth=0.8,relheight=0.5)

    # Book ID
    lb1 = Label(labelFrame,text="Book ID : ", bg='black', fg='white')
    lb1.place(relx=0.05,rely=0.2)

    inf1 = Entry(labelFrame)
    inf1.place(relx=0.3,rely=0.2, relwidth=0.62)

    # Issued To Student name
    lb2 = Label(labelFrame,text="Issued To : ", bg='black', fg='white')
    lb2.place(relx=0.05,rely=0.4)

    inf2 = Entry(labelFrame)
    inf2.place(relx=0.3,rely=0.4, relwidth=0.62)


    #Issue Button
    issueBtn = Button(root,text="Issue",bg='#d1ccc0', fg='black',command=issue)
    issueBtn.place(relx=0.28,rely=0.9, relwidth=0.18,relheight=0.08)
```

# 6. ReturnBook.py




## 6.1. Importing the necessary modules

## Code:

```
from tkinter import *
from PIL import ImageTk,Image
from tkinter import messagebox
import pymysql
```

## 6.2. Connecting to the MySql serverCode:

```
mypass = "root"
mydatabase="db"

con = pymysql.connect(host="localhost",user="root", password=mypass,database=mydatabase)
cur = con.cursor()

# Enter Table Names here
issueTable = "books_issued"
bookTable = "books"

allBid = []  #To store all the Book ID's
```

## 6.3. Function – return()

## Code:

```
def returnn():

    global SubmitBtn,labelFrame,lb1,bookInfo1,quitBtn,root,Canvas1,status

    bid = bookInfo1.get()

    extractBid = "select bid from "+issueTable
    try:
        cur.execute(extractBid)
        con.commit()
        for i in cur:
            allBid.append(i[0])

        if bid in allBid:
            checkAvail = "select status from "+bookTable+" where bid = '"+bid+"'"
            cur.execute(checkAvail)
            con.commit()
            for i in cur:
                check = i[0]

            if check == 'issued':
                status = True
            else:
                status = False

        else:
            messagebox.showinfo("Error","Book ID not present")
    except:
        messagebox.showinfo("Error","Can't fetch Book IDs")


    issueSql = "delete from "+issueTable+" where bid = '"+bid+"'"

    print(bid in allBid)
    print(status)
    updateStatus = "update "+bookTable+" set status = 'avail' where bid = '"+bid+"'"
    try:
        if bid in allBid and status == True:
            cur.execute(issueSql)
            con.commit()
            cur.execute(updateStatus)
            con.commit()
            messagebox.showinfo('Success',"Book Returned Successfully")
        else:
            allBid.clear()
            messagebox.showinfo('Message',"Please check the book ID")
            root.destroy()
            return
    except:
        messagebox.showinfo("Search Error","The value entered is wrong, Try again")


    allBid.clear()
    root.destroy()
```

Explanation – Used to return the books to library and keep a track

## 6.4. Function – returnBook()

## Code:

```
def returnBook():

    global bookInfo1,SubmitBtn,quitBtn,Canvas1,con,cur,root,labelFrame, lb1

    root = Tk()
    root.title("Library")
    root.minsize(width=400,height=400)
    root.geometry("600x500")


    Canvas1 = Canvas(root)

    Canvas1.config(bg="#006B38")
    Canvas1.pack(expand=True,fill=BOTH)

    headingFrame1 = Frame(root,bg="#FFBB00",bd=5)
    headingFrame1.place(relx=0.25,rely=0.1,relwidth=0.5,relheight=0.13)

    headingLabel = Label(headingFrame1, text="Return Book", bg='black', fg='white', font=
('Courier',15))
    headingLabel.place(relx=0,rely=0, relwidth=1, relheight=1)

    labelFrame = Frame(root,bg='black')
    labelFrame.place(relx=0.1,rely=0.3,relwidth=0.8,relheight=0.5)

    # Book ID to Delete
    lb1 = Label(labelFrame,text="Book ID : ", bg='black', fg='white')
    lb1.place(relx=0.05,rely=0.5)

    #Submit Button
    SubmitBtn = Button(root,text="Return",bg='#d1ccc0', fg='black',command=returnn)
    SubmitBtn.place(relx=0.28,rely=0.9, relwidth=0.18,relheight=0.08)

    quitBtn = Button(root,text="Quit",bg='#f7f1e3', fg='black', command=root.destroy)
    quitBtn.place(relx=0.53,rely=0.9, relwidth=0.18,relheight=0.08)

    root.mainloop()
```

Explanation : This function is used to return the books back to the library

# Summary

The Library Management System (LMS) is a comprehensive software solution developed to streamline and automate the day-to-day operations of a library. Designed using **Python for backend logic**, **Tkinter for the graphical user interface (GUI)**, and **MySQL as the core database engine**, this project reflects a practical implementation of database-driven application design within an academic context.

## Purpose and Motivation

Traditional libraries often rely on manual procedures to maintain book inventories and user records, which are prone to inefficiencies and errors. This project addresses those challenges by delivering a system that centralizes book and transaction data, enforces data integrity, and provides a user-friendly interface for both staff and users. The aim is to **increase accuracy, reduce operational time**, and **enhance the accessibility** of information.

## System Architecture

The system follows a **modular and layered architecture**, where each function—such as adding, viewing, deleting, issuing, or returning a book—is encapsulated in a dedicated Python script. The frontend is built using Tkinter, which provides all interface elements like windows, forms, and buttons. The backend interacts with a MySQL database using the `pymysql` connector, allowing for dynamic data retrieval and manipulation through SQL queries.

Key components of the architecture include:

- **AddBook.py**: Handles insertion of new book records into the database.
- **ViewBooks.py**: Retrieves and displays all book details from the `books` table.
- **DeleteBook.py**: Enables deletion of book records from the system.
- **IssueBook.py**: Updates book status and logs issuance in the `books_issued` table.
- **ReturnBook.py**: Manages book returns and restores book availability.

## Database Design

The database schema is structured around the following core tables:

- **books**: Stores details such as Book ID, Title, Author, and Status.
- **books_issued**: Logs which books are issued to which users along with timestamps.
- **login (optional)**: Holds user credentials for system access.

The use of primary keys, foreign keys, and composite keys ensures **data consistency**, supports **relational integrity**, and allows for **efficient joins and queries**.

## Features and Functionalities

The system offers the following core functionalities:

- **Add Books**: Allows librarians to register new books in the catalog.
- **View Books**: Displays all registered books with their details and availability.
- **Delete Books**: Removes outdated or lost books from the system.
- **Issue Books**: Assigns books to users and updates the inventory.
- **Return Books**: Processes book returns and restores availability.
- **Real-Time Data Interaction**: All actions are immediately reflected in the MySQL database.
- **Error Handling**: Ensures the user is informed of invalid inputs or failed transactions.

## Benefits of the System

- **Automation** of routine tasks improves productivity.
- **Reduction of errors** due to manual handling.
- **Improved access and organization** of book and transaction data.
- **Scalable database structure** supports future enhancements.
- **Cross-platform compatibility** due to Python's portability.

## Scope for Future Work

The system serves as a functional prototype with scope for real-world deployment in school, college, or small community libraries. Future upgrades may include:

- User registration and account management
- Integration with barcode scanners or RFID
- Automated fine calculation
- Notification systems for due dates and reservations
- Cloud database support for centralized access

## Conclusion

The Library Management System project demonstrates the practical application of database concepts, GUI development, and Python programming to solve a real-world problem. It merges academic principles with real-time interaction, providing a strong foundation for learning and extending into professional library solutions. Through this project, the team has gained hands-on experience in designing, coding, and integrating multi-tier systems and handling data-driven operations effectively.