DBMS Mini Project

Report

Title - LIBRARY MANAGEMENT SYSTEM

PROJECT Members:

Kandukuru Udaya Sankar-RA1811003010**456**

Janga Sai Chandra-RA1811003010**479**

Library Management System in Python

### Project Prerequisites:

* tkinter
* pillow
* pymysql

### Description of Project Files:

* **main.py -** which does function call to all other python files.
* **AddBook.py -** To add the book.
* **ViewBooks.py -** To View the list of books in the library.
* **DeleteBook.py -** To Delete a book from library.
* **IssueBook.py –** To Issue a book from library.
* **ReturnBook.py -** To Return a book to the library.

### Description of Tables:

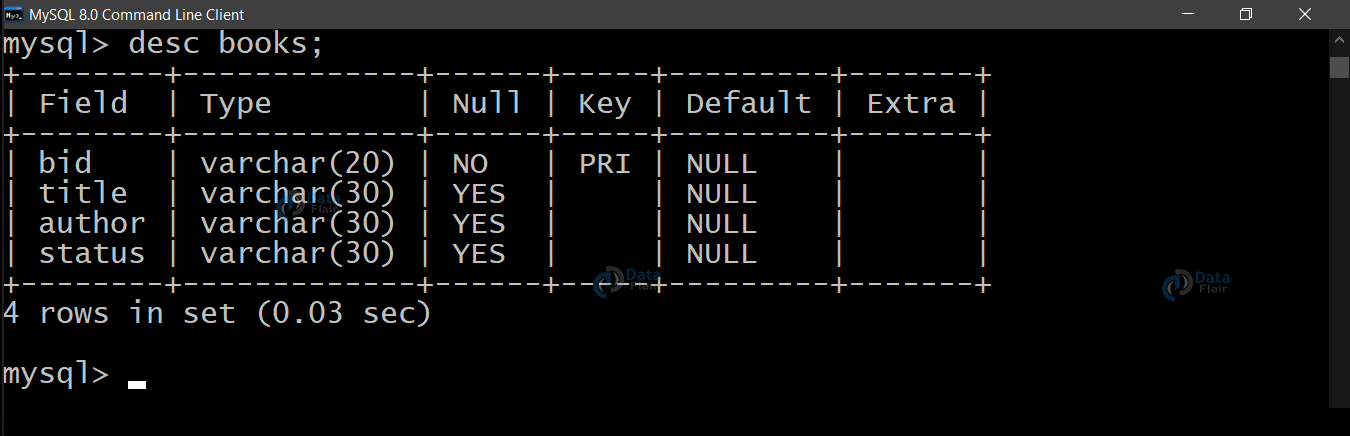
**Create Tables:**

create database db;

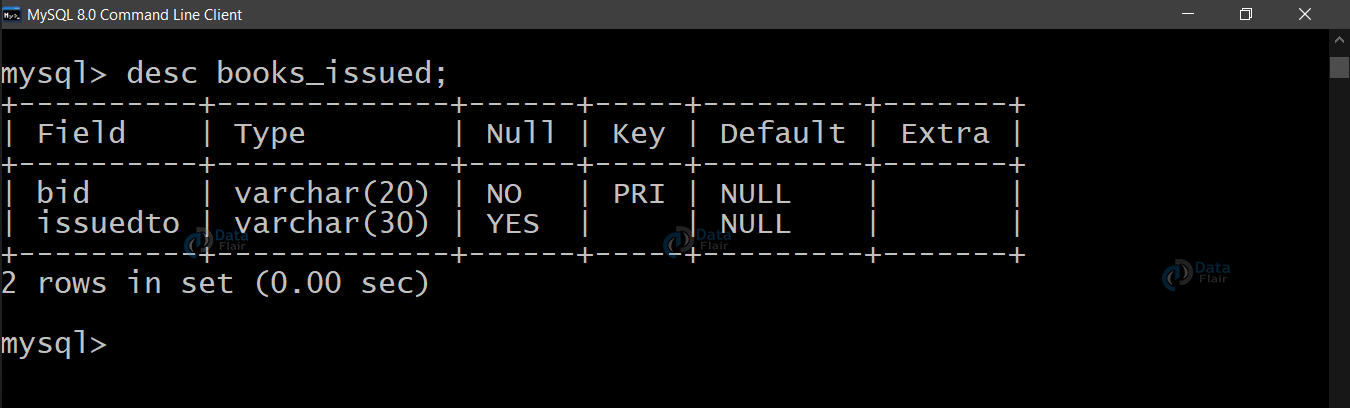
create table books(bid varchar(20) primary key, title varchar(30), author varchar(30), status varchar(30));

create table books\_issued(bid varchar(20) primary key, issuedto varchar(30));

**1)books:**



**2)issued\_books:**



### Library Management Project module Explanation:

**1)main.py:**

**(Importing the modules)**

To use the Tkinter we need to import the Tkinter module. We have imported each file so that we can make function calls from our main file. Now we will connect to the server with the correct credentials associated with the MySql server installed on our system. 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.

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.

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**.And at the last add buttons to the window frame.

**2)AddBook.py:**

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.

The function bookRegister() executes an SQL command to insert data into the table and commit the changes. 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.

insertBooks = “insert into “+bookTable+” values (‘”+bid+”‘,'”+title+”‘,'”+author+”‘,'”+status+”‘)”

We put this code in a try-except block in order to handle the situation effectively.Now, we execute the **insertBooks** command by **.execute()** method associated with cur. We commit the changes by **.commit()** method associated with con as discussed above.We use **messagebox()** function to acknowledge the user of success or failure.

The function addBook() 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.

**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**

We declare certain variables as global in order to use them in the **bookRegister()** function.

We create another window and connect to the MySql server and pass on the cursor control to **cur**. It means that whatever we want to write on the MySql shell, we will do it through **cur**. In order to **commit(write changes in the table)** the changes we will use **con**.

We draw the canvas and paint it with a good color. I have used **Orange**, you can use any color you like.

We create and place the **headingLabel** inside the **headingFrame1** and give the title as “**Add Books**”.

The **labelFrame** basically creates a **black box** (in our case) to accommodate the input fields to fetch the book details.

We create and place a **Label** on our black box which displays the text ‘**Book ID**:’. Just after **Label** is placed we create and place the **Entry** box acting as our input field.

Similarly, we create and place input fields associated with labels: Title, Author, and Status.

In the end, we create a button to **Finally.!!** submit the details given by the user and a button to exit just in case the user did not intend to enter any details in library management.

As an action of a click on the **SubmitBtn**, we call the **bookRegister()** function to insert the details into the **books** table.

**3)ViewBooks.py:**

The function view() in our library project creates a window for displaying the records in the table. We create a new window to display the list of books and their status.

Just like we did in the previous file, we create a **headingFrame** and title it with a Label to display ‘**View Books**’.

Again we create a black box to accommodate the records returned by the **getBooks** query.

We manually display the name of the columns associated with our books table.We execute the query stored in **getBooks** using **cur.execute()** display each record one by one as a Label. We place each **Label** You can notice that each time a record is displayed the value of **y** increases by **‘0.1’**.

In order to handle any discrepancies, we place this code in a try-except block.In the end, we create and place a button, **quitBtn** to exit from the current window of the library project.

**4)DeleteBook.py:**

The function deletebook() primarily checks if the bid (book id) exists in the book table and if it does, it executes the necessary command to remove it. We store the SQL query to delete the record in **deleteSql**. After that, we execute this command using **cur.execute()**.

In case someone loses a book, we should delete that book from the **issueTable** in order to prevent discrepancies in the future. Hence we store the SQl query to delete the same record from the IssueTable in **deleteIssue**. We execute it along with the **deleteSql**.

The function delete() creates a window for accommodating a text input field. We fetch details of a book from the user and then call **deleteBook()** function to delete the book record from the table. Firstly, we create a new window and accommodate a **headingFrame** followed by creating the **labelFrame** to create and place a big black box.

This black box accommodates a **Label** and an **Entry** text field to take input of the Book ID.

After the black box, we create and place a **Submit** and **Quit** button associated with the name SubmitBtn and **quitBtn** respectively. A click on the Submit button triggers the **deleteBook()** function.

Moreover, we declare some variables as **global**, in order to provide their access in the **deleteBook()** function.

**5)IssueBook.py:**

In the function issue() primarily, we fetch the desired book ID and Issuer’s name and store it into **bid** and **issueto** respectively.

After that we store all the Book ID from the **books** table in **allBid** by executing the SQL query stored in **extractBid**.

We check for the existence of the desired **bid** in **allBid**. If it exists and is available, we set the **status** as **True**.

If the book is available we update the **books\_issued** table with the book id (**bid**) and Issuer’s name (**issueto**) and update the **books** table by changing the status of the issued book to ‘**issued**’.

In the function issueBook() we create and place a **headingFrame** and two input fields for taking input of the desired books’ ID and issuers’ name. After which we create and add two buttons named **issueBtn** and **quitBtn** to facilitate submission of our issue request and closing the present window of library management system respectively.

**6)ReturnBook.py:**

In the function returnn(),this function is similar to the **issue()** function we designed for **issueBook.py**.

In this library management system project, we fetch the desired book ID and store it into **bid**.

After that we store all the Book IDs from the **books\_issued** table in **allBid** by executing the SQL query stored in **extractBid**.

We check for the existence of the desired **bid** in **allBid**. Also, we check the status of the same book and if it is ‘**issued**’, we set the **status** as **True**.

Then, we remove the record from **books\_issued** table and update the **books** table by changing the status of the issued book to ‘**avail**’.

In the function returnBook() we create and place a **headingFrame** and an input field for taking input of the books’ ID. Then, we create and add two buttons named **SubmitBtn** and **quitBtn** to facilitate submission of our return request and closing the present window respectively.

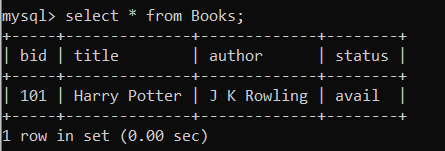
### Front End Tool Used:

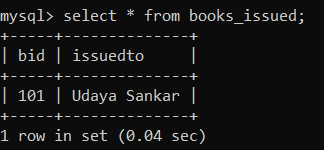
* **Tkinter**

### Backend:

* **Python**
* **MySQL Community Server**

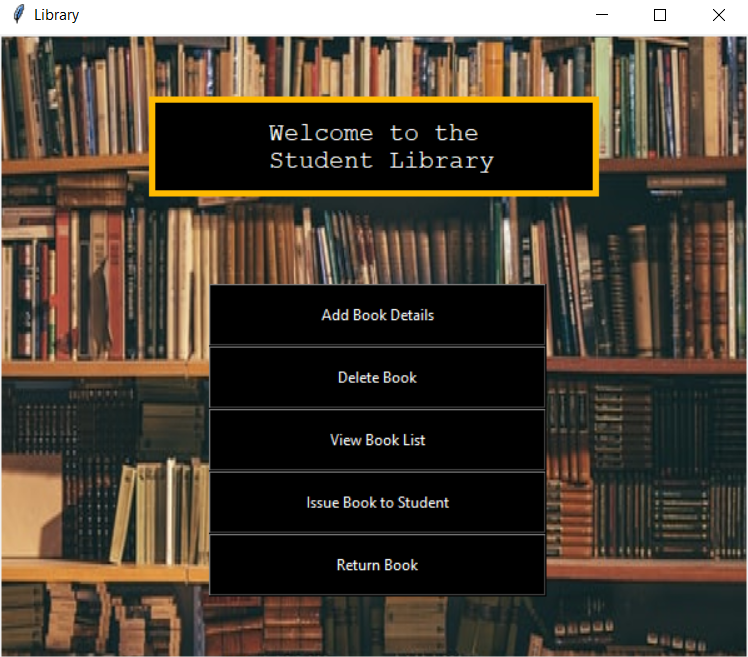
Sample Entries:

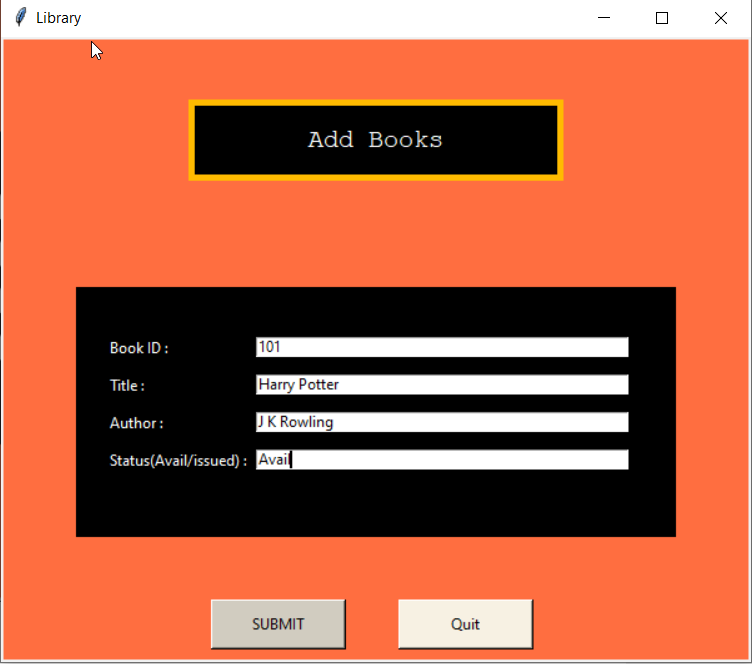
****

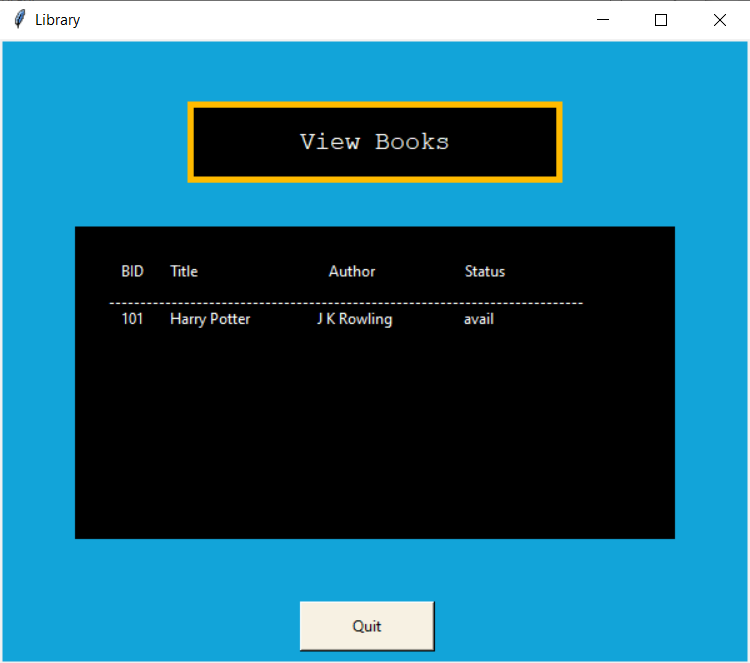
****

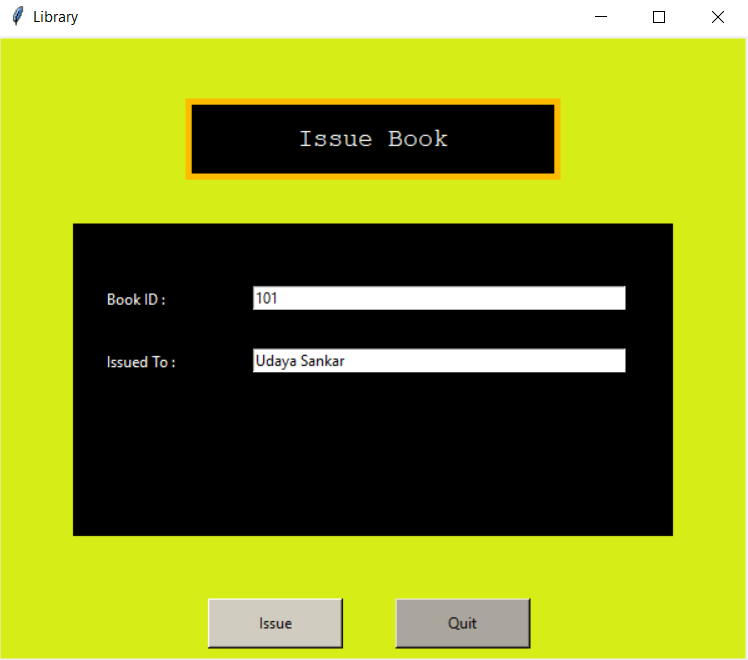
Screenshots:

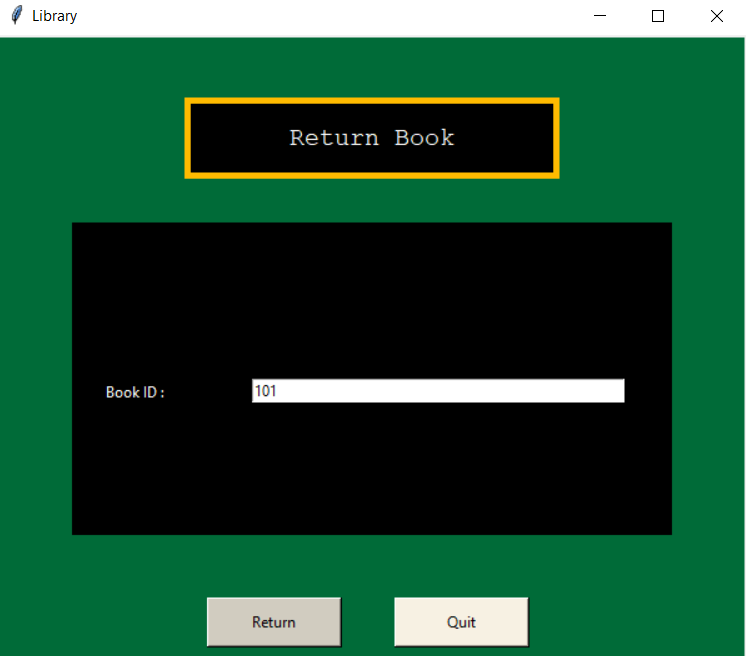
**Front End:**

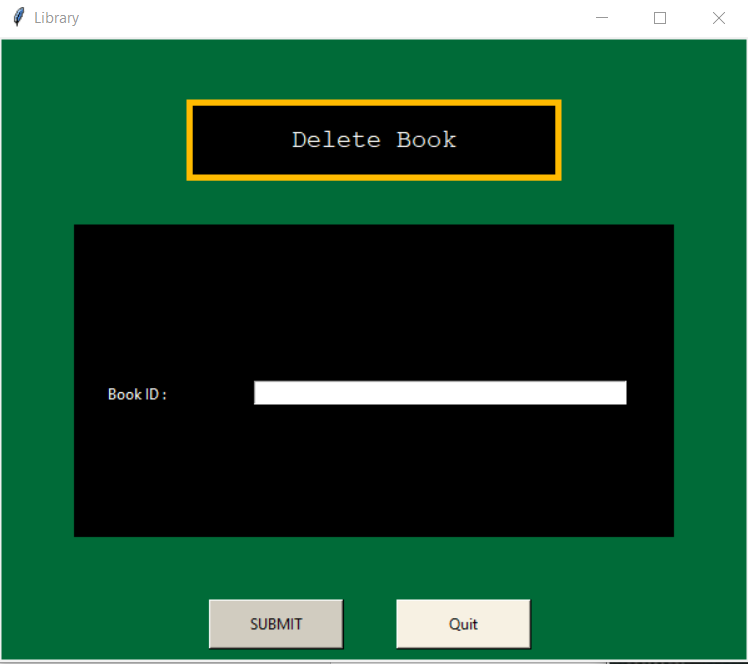
****

****

****

****

****

****

Conclusion:

Therefore, we have successfully designed a Library management system using python and tkinter with a decent UI. In order to make things easy, we divided the various tasks into different python files. The name of the files are main.py which is the main file that uses other files which are addbook.py,viewbooks.py,deletebook.py,issuebook.py,returnbook.py.The library management system keeps track of the books present in the library.We can use this to add, delete, view books and we can also see whether the book is issued and if the book is returned.