

Python Project Report on Book Management System



School of computer science and Engineering

Course code: INT213

Topic: Design a Books Management System in LPU using python.

Project Number: 19

Section: K21KA

Team Members Details

| | | | |
|-------------------------------|------------|---|---------------|
| L. Ganesh Kumar Reddy | - 12109343 | - | Roll no. : 02 |
| Siddamsetti Revanth Gupta | - 12109471 | - | Roll no. : 25 |
| Nedunuri Naga Satya Sai Akhil | - 12109450 | - | Roll no. : 33 |

Submitted to: Dr. Deepika Ghai

Table of Content

| | |
|---|----|
| Introduction..... | 2 |
| Objectives..... | 3 |
| Design..... | 4 |
| Widgets used in project..... | 5 |
| Geometry Management..... | 6 |
| Description of Modules | 7 |
| Functions in Book management system..... | 8 |
| Source code..... | 9 |
| Snapshots of GUI..... | 14 |
| Applications of the Book management system..... | 15 |
| Conclusion..... | 16 |
| References | |

INTRODUCTION

The book management system software helps in reducing operational costs. Managing a books manually is labor intensive and an immense amount of paperwork is involved. An automated system reduces the need for manpower and stationery. This leads to lower operational costs. The system saves time for both the user and the librarian. With just a click the user can search for the books available in the book. The librarian can answer queries with ease regarding the availability of books. Adding, removing, or editing the database is a simple process. Adding new members or cancelling memberships can be done with ease.

Stock checking and verification of books in the book can be done within a few hours. The automated system saves a considerable amount of time as opposed to the manual system. The book management system software makes the book a smart one by organizing the books systematically by author, title, and subject. This enables users to search for books quickly and effortlessly.

Students need access to authentic information. An advanced organized book is an integral part of any educational institution. In this digital age, a web-based book management system would be ideal for students who can access the book's database on their smartphones.

OBJECTIVES

The main objective of the book management system is to allow the user to enter and store data in the database easily and effectively. Its main purpose is to maintain the details of books and book members of different libraries. The other purpose is to maintain easy circulation between clients and the libraries. Users can also search for the book in the different sections of the book. The other features of the book management system are to allow the user to add the details to the database, delete the book record, show borrower details, update borrower details, delete a record, record a book detail, and delete a book detail.

User-Friendly System:

- Simple and easy to use.
- Online and offline storage of data.
- Automatically updates and backups data.
- Flexible and can be fully configurable.

Cost Effective:

- Eliminate the need for extensive paperwork.
- Maintenance overheads and operation costs are reduced.
- Eliminates the need for manual entries.
- Makes the database error-free and accurate.

Increased Member Engagement.

- Easily accessible from anywhere and at any time.
- Easy access from smartphones and tablets.
- Reliable and secure operations.

Design

Main Library used in this Project:

Tkinter:

Tkinter is the standard GUI book for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using Tkinter is an easy task

Basic code for GUI using Tkinter in Python:

```
import tkinter as tk
window = tk.Tk()
greeting = tk.Label(text = "Hello, Tkinter") # adding widget label to print text.
greeting.pack() # .pack() method is used to add label to windows
```

Output of the code is shown below:



Tkinter Widgets

Tkinter is Python's standard GUI (Graphical User Interface) package. Tkinter provides us with a variety of common GUI elements which we can use to build out interfaces – such as buttons, menus, and various kinds of entry fields and display areas.

Widgets Used in the project

Label: The Label is used to specify the container box where we can place the text or images. This widget is used to provide a message to the user about other widgets used in the python application. There are various options that can be specified to configure the text or the part of the text shown on the Label. The syntax to use the Label is given below.

Syntax: w = Label (master, options)

Below is the Source code in the project where **Labels** are used

```
l1 = Label(window,text="Book Name",bg="light blue",width=12)
l1.grid(row=0,column=0)

l2 = Label(window, text="Author",bg="light blue",width=12)
l2.grid(row=0,column=2)

l3 = Label(window, text="Quantity",bg="light blue",width=12)
l3.grid(row=1,column=0)

l4 = Label(window, text="Book Id",bg="light blue",width=12)
l4.grid(row=1,column=2)
```

Button: The button widget is used to add various types of buttons to the python application. Python allows us to configure the look of the button according to our requirements. Various options can be set or reset depending on the requirements. We can also associate a method or function with a button which is called when the button is pressed. The syntax to use the button widget is given below.

Syntax: w = Button (parent, options)

Below is the Source code in the project where **buttons** are used

```
b1 =Button(window, text= "View All", width=12, command=view_command,bg="light
blue")
b1.grid(row=2, column=3)

b2 =Button(window, text= "Search Book", width=12,
command=search_command,bg="light blue")
b2.grid(row=3, column=3)

b3 =Button(window, text= "Add Book", width=12, command=add_book,bg="light
blue")
b3.grid(row=4, column=3)

b4 =Button(window, text= "Update", width=12, command=update_book,bg="light
blue")
```

```

b4.grid(row=5, column=3)

b5 =Button(window, text= "Delete", width=12, command=delete_book,bg="light
blue")
b5.grid(row=6, column=3)

b6 =Button(window, text= "Close", width=12, command=window.destroy,bg="light
blue")
b6.grid(row=7, column=3)

```

Entry: The Entry widget is used to provide the single line text-box to the user to accept a value from the user. We can use the Entry widget to accept the text strings from the user. It can only be used for one line of text from the user. For multiple lines of text, we must use the text widget. The syntax to use the Entry widget is given below.

Syntax: w = Entry (parent, options)

Bg: It is used to set the normal background color.

```

e1 = Entry(window, textvariable= title_text)
e1.grid(row=0, column=1)

author_text = StringVar()
e2 = Entry(window, textvariable= author_text)
e2.grid(row=0, column=3)

year_text = StringVar()
e3 = Entry(window, textvariable= year_text)
e3.grid(row=1, column=1)

isbn_text = StringVar()
e4 = Entry(window, textvariable= isbn_text)
e4.grid(row=1, column=3)

```

ListBox: It is used to display different types of items. These items must be of the same type of font and having the same font color. The items must also be of Text type. The user can select one or more items from the given list according to the requirement.

Syntax: List = Listbox(root, bg, fg, bd, height, width, font)

Below is the Source code in the project where **Listbox** are used

```
list1 = Listbox(window, height=25, width=45,bg="#FAF165",bd="3px", font =  
"Helvetica")  
list1.grid(row=2, column =0, rowspan=6, columnspan=2)
```

Scrollbar: The scrollbar widget is used to scroll down the content of the other widgets like listbox, text, and canvas. However, we can also create horizontal scrollbars for the Entry widget.

Syntax: w = Scrollbar (top, options)

```
sb1 = Scrollbar(window)  
sb1.grid(row=2, column=2 ,rowspan = 6)
```

Geometry Management

Grid () Method: The grid () geometry manager organizes the widgets in tabular form. We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or rowspan(height) of a widget. This is a more organized way to place the widget into the python application. The syntax to use the grid () is given below.

Syntax: widget.grid(options)

A list of all options that can be passed inside the grid() method is which are used in a project is given below.

1. **Row:** The row number in which the widget is to be placed and the top-most row is represented by 0.
2. **Rowspan:** The height of the widget is the number of rows up to which the widget is expanded.
3. **Column:** The column number in which the widget is to be placed. The leftmost column is represented by 0.
4. **Columnspan:** The width of the widget. It represents the number of columns up to which, the column is expanded.

Description of Modules

There are 4 modules in this project:

1. **Bookstore.py:** This module is like the front end of this project where all the widgets, Tkinter functions are used.

2. **Backend.py:** In this module, all the functions are declared, which are used in bookstore.py and it acts like a bridge for Bookstore.py and database.py modules.
3. **Database.py:** It consists of database functions in python, which collect data from the user through GUI and update the table in a database.
4. **Books.db:** Books.db is the database module where all the information of books is stored in this database in the table book

Functions in Book management system

View All: This button shows all the books present in the database along with the Book name, Book Id, Author, and Quantity.

```
def view():  
    conn =sqlite3.connect("books.db")  
    cur= conn.cursor()  
    cur.execute("SELECT * FROM book")  
    rows =cur.fetchall()  
    conn.close()
```

When a user presses the view all button the cursor goes to the database table and runs the command and returns all the rows present in the table.

Command: “SELECT * FROM book”

Search Book: To search the book entered by the user, we need a unique attribute to identify them i.e, title in our table, and return the rows whose title is equal to the user-entered book name.

```
def search(title="", author="", year="", isbn=""):  
    conn =sqlite3.connect("books.db")  
    cur= conn.cursor()  
    cur.execute("SELECT * FROM book WHERE title=? OR author=? OR year=? OR  
isbn=?",(title,author,year,isbn))  
    rows =cur.fetchall()  
    conn.close()  
    return rows
```

Add book: This button inserts the values entered by the user in GUI into the database.

Command: “INSERT INTO book VALUES(NULL, ?,?,?,?)”

```
def insert(title, author, year, isbn):  
    conn =sqlite3.connect("books.db")  
    cur= conn.cursor()  
    cur.execute("INSERT INTO book VALUES(NULL, ?,?,?,?)",(Name,author,quantity,Id))
```



```
conn.commit()
conn.close()
```

Update: It is used to modify the data of books like the number of books present in the library, based on name or author or quantity or id.

Command: "UPDATE book SET title=?, author=?, year=?, isbn=? WHERE id=?"

```
def update(id, title, author, year, isbn ):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("UPDATE book SET name=?, author=?, qquantity=?, id=? WHERE id=?", (name,
author, quantity, id))
    conn.commit()
    conn.close()
```

Delete: To delete the book entered by the user, we need a unique attribute to identify them i.e, book Id in our table, and return the rows whose title is equal to the user-entered book name and delete them.

Command: "Delete from book where id=?"

```
def delete(id):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("DELETE FROM book WHERE id=?", (id,))
    conn.commit()
    conn.close()
```

Source code

Code for bookstore.py module

```
from tkinter import *
import backend
window = Tk()

def get_selected_row(event):
    try:
        global select_tup
        index=list1.curselection()[0]
        select_tup = list1.get(index)
        e1.delete(0,END)
        e1.insert(END, select_tup[1])
        e2.delete(0,END)
```

```

        e2.insert(END, select_tup[2])
        e3.delete(0,END)
        e3.insert(END, select_tup[3])
        e4.delete(0,END)
        e4.insert(END, select_tup[4])
    except IndexError:
        pass

def view_command():
    list1.delete(0,END)
    for row in backend.view():
        list1.insert(END, row)

def search_command():
    list1.delete(0,END)
    for row in backend.search(title_text.get(),author_text.get(),year_text.get(),
isbn_text.get()):
        list1.insert(END,row)

def add_book():
    backend.insert(title_text.get(),author_text.get(),year_text.get(), isbn_text.get())
    list1.delete(0,END)
    list1.insert(END,(title_text.get(),author_text.get(),year_text.get(), isbn_text.get()))

def delete_book():
    backend.delete(select_tup[0])

def update_book():
    backend.update(select_tup[0], title_text.get(),author_text.get(),year_text.get(),
isbn_text.get())

window.title("Book Management system")
window.geometry("650x650")
window.config(bg="#FABF65")

l1 = Label(window,text="Book Name",bg="light blue",width=12)
l1.grid(row=0,column=0)

l2 = Label(window, text="Author",bg="light blue",width=12)
l2.grid(row=0,column=2)

l3 = Label(window, text="Quantity",bg="light blue",width=12)
l3.grid(row=1,column=0)

l4 = Label(window, text="Book Id",bg="light blue",width=12)
l4.grid(row=1,column=2)

title_text = StringVar()
e1 = Entry(window, textvariable= title_text)
e1.grid(row=0, column=1)

```

```

author_text = StringVar()
e2 = Entry(window, textvariable= author_text)
e2.grid(row=0, column=3)

year_text = StringVar()
e3 = Entry(window, textvariable= year_text)
e3.grid(row=1, column=1)

isbn_text = StringVar()
e4 = Entry(window, textvariable= isbn_text)
e4.grid(row=1, column=3)

list1 = Listbox(window, height=20, width=30,bg="#FAF165",bd="3px", font = "Helvetica")
list1.grid(row=2, column =0, rowspan=6, columnspan=2)
list1.bind("<<ListboxSelect>>", get_selected_row)

sb1 =Scrollbar(window)
sb1.grid(row=2, column=2 ,rowspan = 6)

list1.configure(yscrollcommand=sb1.set)
sb1.configure(command=list1.yview)

b1 =Button(window, text= "View All", width=12, command=view_command,bg="light blue")
b1.grid(row=2, column=3)

b2 =Button(window, text= "Search Book", width=12, command=search_command,bg="light blue")
b2.grid(row=3, column=3)

b3 =Button(window, text= "Add Book", width=12, command=add_book,bg="light blue")
b3.grid(row=4, column=3)

b4 =Button(window, text= "Update", width=12, command=update_book,bg="light blue")
b4.grid(row=5, column=3)

b5 =Button(window, text= "Delete", width=12, command=delete_book,bg="light blue")
b5.grid(row=6, column=3)

b6 =Button(window, text= "Close", width=12, command=window.destroy,bg="light blue")
b6.grid(row=7, column=3)
window.mainloop()

```

Code for backend.py module

```
import sqlite3

def connect():
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS book (id INTEGER PRIMARY KEY, title text, author text, year integer, isbn integer) ")
    conn.commit()
    conn.close()

def insert(title, author, year, isbn):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("INSERT INTO book VALUES(NULL, ?,?,?,?)", (title,author,year,isbn))
    conn.commit()
    conn.close()

def view():
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("SELECT * FROM book")
    rows =cur.fetchall()
    conn.close()
    return rows

def search(title="", author="", year="", isbn=""):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("SELECT * FROM book WHERE title=? OR author=? OR year=? OR isbn=?", (title,author,year,isbn))
    rows =cur.fetchall()
    conn.close()
    return rows

def delete(id):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("DELETE FROM book WHERE id=?", (id,))
    conn.commit()
    conn.close()

def update(id, title, author, year,isbn ):
    conn =sqlite3.connect("books.db")
    cur= conn.cursor()
    cur.execute("UPDATE book SET title=?, author=?, year=?, isbn=? WHERE id=?", (title, author, year, isbn,id))
    conn.commit()
    conn.close()

connect()
```

Code for database.py module

```
import sqlite3

class Database:

    def __init__(self, db):
        self.conn =sqlite3.connect(db)
        self.cur= self.conn.cursor()
        self.cur.execute("CREATE TABLE IF NOT EXISTS books1 (id INTEGER PRIMARY KEY, title
text, author text, quantity integer, isbn integer) ")
        self.conn.commit()

    def insert(self,title, author, year, isbn):
        self.cur.execute("INSERT INTO books1 VALUES(NULL, ?,?,?,?)",
(title,author,year,isbn))
        self.conn.commit()

    def view(self):
        self.cur.execute("SELECT * FROM books1")
        rows =self.cur.fetchall()
        return rows

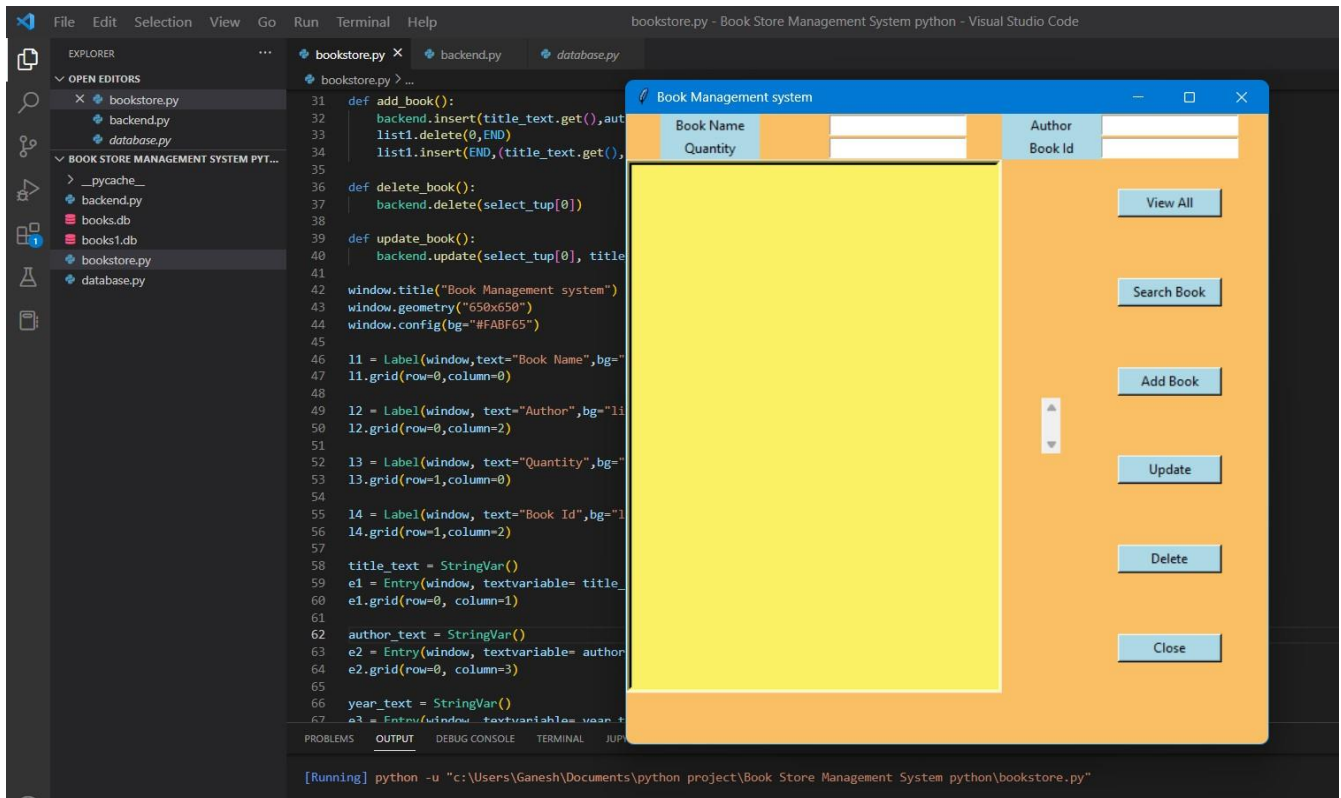
    def search(self,title="", author="", year="", isbn=""):
        self.cur.execute("SELECT * FROM books1 WHERE title=? OR author=? OR year=? OR
isbn=?",(title,author,year,isbn))
        rows = self.cur.fetchall()
        return rows

    def delete(self,id):
        self.cur.execute("DELETE FROM books1 WHERE id=?",(id,))
        self.conn.commit()

    def update(self,id, title, author, year,isbn ):
        self.cur.execute("UPDATE books1 SET title=?, author=?, year=?, isbn=? WHERE
id=?",(title, author, year, isbn,id))
        self.conn.commit()

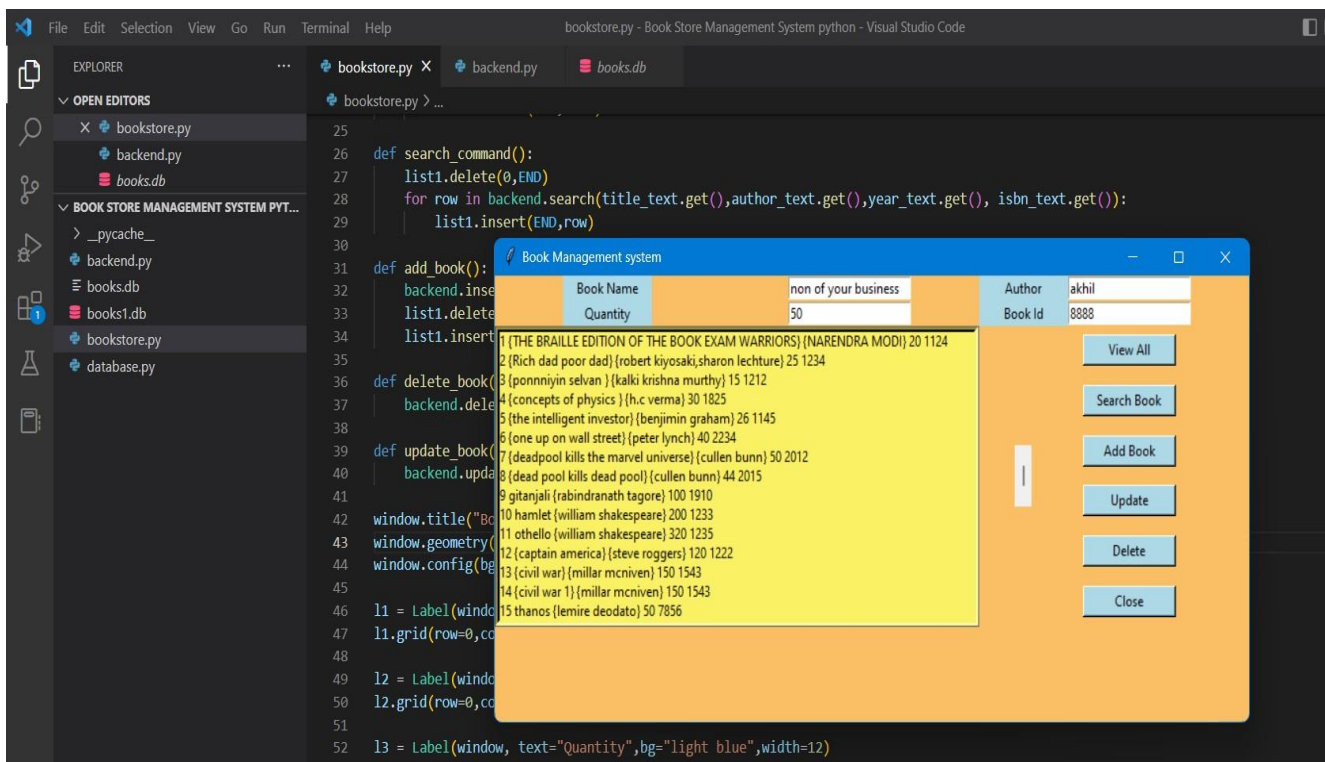
    def __del__(self):
        self.conn.close()
```

Snapshots of GUI

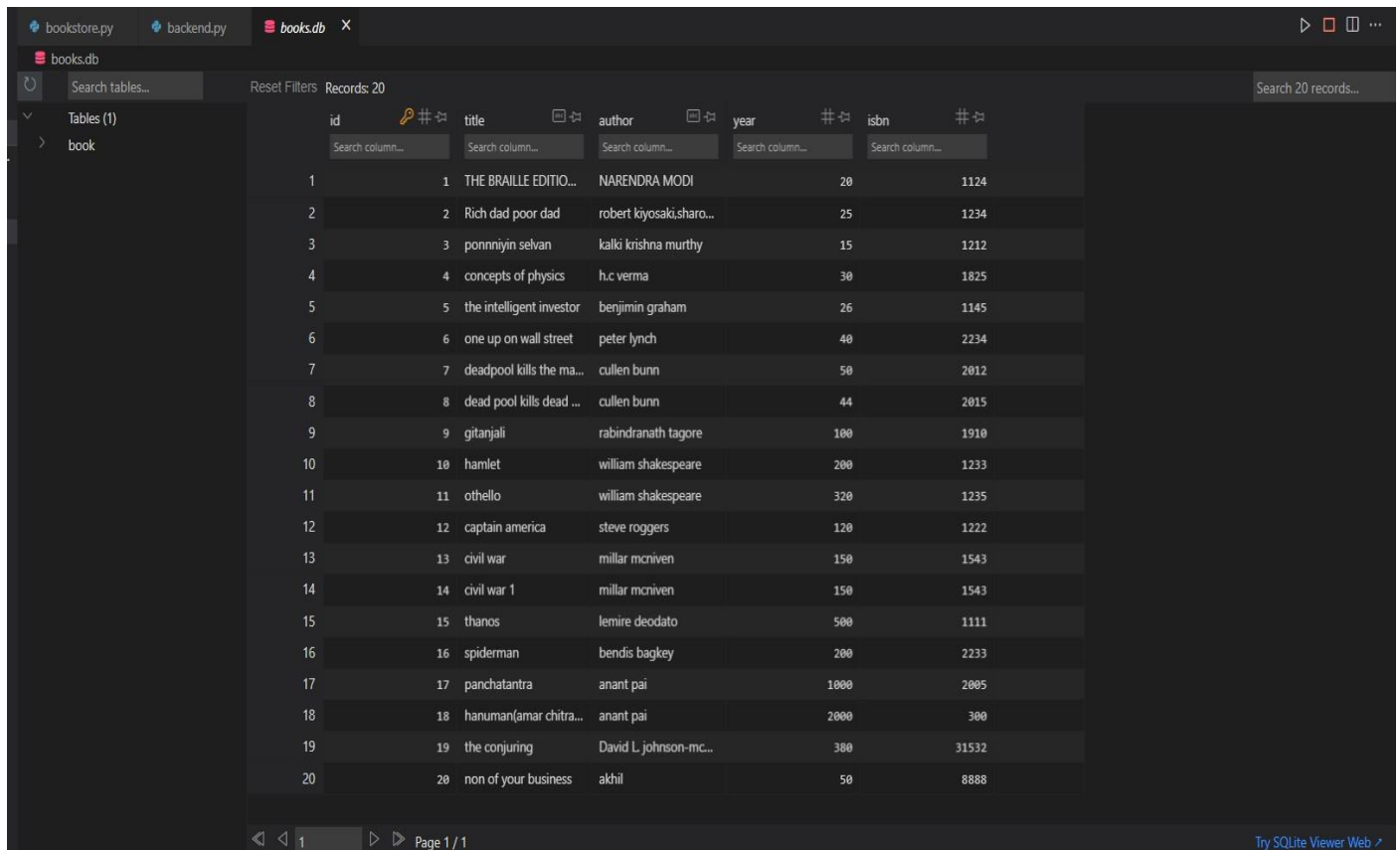


After entering the data of books into above blanks

The GUI



The updated database table in books.db



| id | title | author | year | isbn |
|----|--------------------------|--------------------------|------|-------|
| 1 | THE BRAILLE EDITIO... | NARENDRA MODI | 20 | 1124 |
| 2 | Rich dad poor dad | robert kiyosaki.sharo... | 25 | 1234 |
| 3 | ponnniyin selvan | kalki krishna murthy | 15 | 1212 |
| 4 | concepts of physics | h.c verma | 30 | 1825 |
| 5 | the intelligent investor | benjamin graham | 26 | 1145 |
| 6 | one up on wall street | peter lynch | 40 | 2234 |
| 7 | deadpool kills the ma... | cullen bunn | 50 | 2012 |
| 8 | dead pool kills dead ... | cullen bunn | 44 | 2015 |
| 9 | gitanjali | rabindranath tagore | 100 | 1910 |
| 10 | hamlet | william shakespeare | 200 | 1233 |
| 11 | othello | william shakespeare | 320 | 1235 |
| 12 | captain america | steve rogers | 120 | 1222 |
| 13 | civil war | millar mcniven | 150 | 1543 |
| 14 | civil war 1 | millar mcniven | 150 | 1543 |
| 15 | thanos | lemire deodato | 500 | 1111 |
| 16 | spiderman | bendis bagkey | 200 | 2233 |
| 17 | panchatantra | anant pai | 1000 | 2005 |
| 18 | hanuman(amar chitra... | anant pai | 2000 | 300 |
| 19 | the conjuring | David L.johnson-mc... | 300 | 31532 |
| 20 | non of your business | akhil | 50 | 8888 |

Applications of the Book management system

The **Book management system** is designed to contribute well-management of library functions. It offers ease to perform day-to-day library operations electronically.

Some applications are:

- Simple and easy to operate.
- Increase librarian efficiencies.
- Mobile access, anytime, anywhere.
- Search, add, update, and view library materials online.
- Helps to manage library functions constructively.
- Saves time and reduces overhead.
- Reduce the library's operating cost.
- Customized reports for better management.
- Remove manual processes to issue books and maintain records.

Conclusion

Working on this project helped us learn about python in more detail, it also made us realize how important and how useful it is, working on PIP modules, Tkinter, and other libraries, we learned about them in more depth, how to work on Graphical User Interfaces (GUI) on python using Tkinter. This project even helped us understand how to create a book management system and how the different modules work. Using this software users can easily keep records of books and do a variety of tasks.

References

- <https://www.javatpoint.com/>
- <https://www.geeksforgeeks.org/>
- <https://www.w3schools.com/>
- <https://www.tutorialspoint.com/>
- <https://stackoverflow.com/>

These are some websites used for solving problems and used for collecting information