

Amicus International School, Bharuch

PROJECT REPORT ON

SOFTWARE FOR

Library Management

Computer Science Project

Created By: **Dipam Sen**
Arghya Kumar
Manya Kansara

Certificate

This is to certify that **Dipam Sen** of Class XII-B, have successfully completed the investigatory project on the topic **Software for Library Management** under the guidance of Mrs. Arpita Bhoi during the academic year 2023-24 in the partial fulfillment of Practical Examination conducted by CBSE.

Signature of
Internal Examiner

Signature of
Principal

Signature of
External Examiner

Acknowledgements

I extend my heartfelt gratitude to everyone who has contributed to the completion of this Project.

I am sincerely grateful to our principal Mrs. Thresiamma Pappachan and my CS teacher Mrs. Arpita Bhoi. Their unwavering support and guidance have been instrumental in its completion.

Lastly, I express my sincere gratitude to my parents and family for their unwavering support.

To all those mentioned above and many others who have directly or indirectly contributed, your support has been truly invaluable in my growth as a learner.

- Dipam Sen

Index

Certificate	2
Acknowledgements	3
Index	4
Introduction	5
Technologies Used	5
Functional Requirements	5
Database Design	6
Books	6
Patrons	7
Transactions	7
Functional Decomposition	8
Control Flow of the Application	10
Issuing a Book	11
Source Code	12
main.py	12
database.py	20
db.sql	29
Screenshots	30
Bibliography	31

Introduction

The Library Manager is designed to facilitate the process of book issuance and returns in a library environment. This application offers a solution for tracking transactions, simplifying administrative tasks, and presenting data in a clear manner. Users can efficiently manage the database by updating, creating, and removing records. The goal is to make the application more accessible by streamlining library operations and transaction management.

Technologies Used

This project uses Python as the primary programming language for the frontend application. The project uses an MySQL database for the backend, for efficient data storage and retrieval.

Functional Requirements

The following are the functional requirements for a library management system:

- To list and search books by criteria
- To add, remove and update books
- To view information about patrons
- To issue books and return books
- To view and filter information about book issuance

Database Design

The database for our application is managed by MySQL - it comprises of 3 tables: **Books**, **Patrons**, and **Transactions**. The ERD (Entity Relationship Diagram) shows the various relations and fields in the database (See Figure 1).

Books

The **Books** table stores information on books, storing isbn, title, author, quantity and genre.

- isbn: The International Standard Book Number uniquely identifying each book.
- title: The title of the book.
- author: The author of the book.
- quantity: The number of available copies of the book.
- genre: The genre of the book.

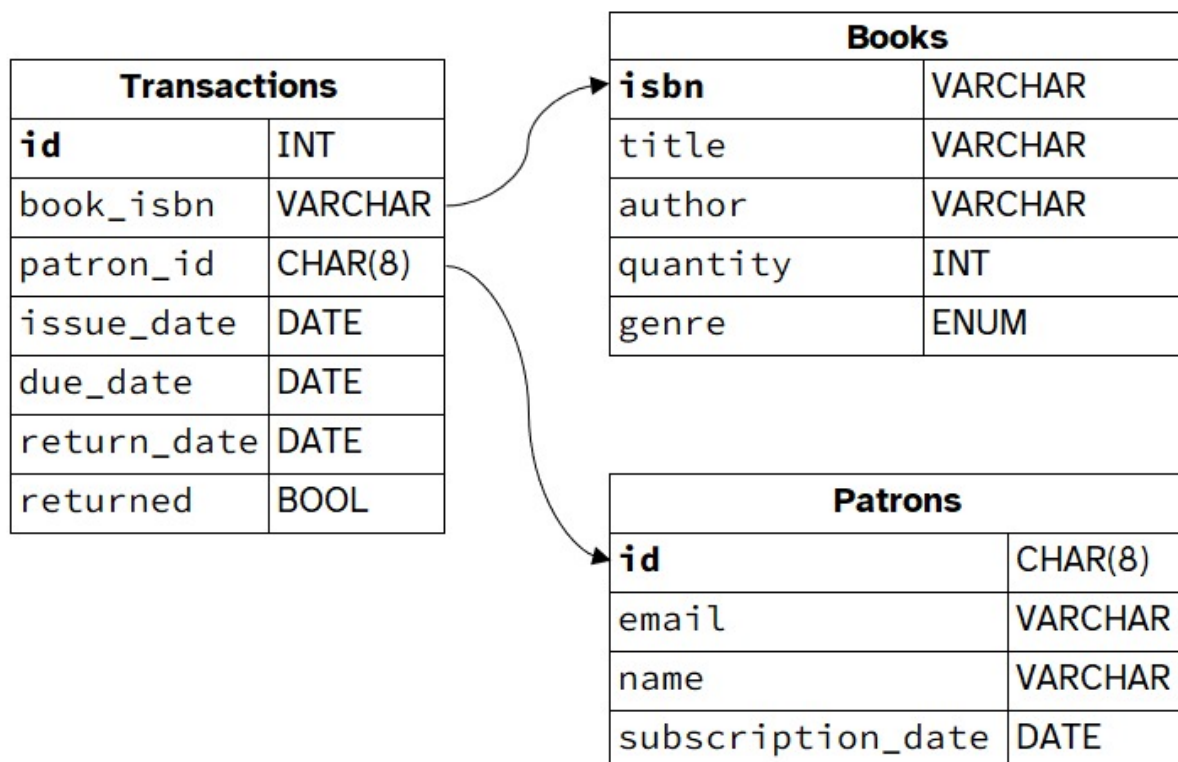


Figure 1: Database Entity Relationship diagram

Patrons

The **Patrons** table stores user info, about library patrons who can borrow and return books.

- **id**: A unique identifier for each patron.
- **email**: The email address of the patron.
- **name**: The name of the patron.
- **subscription_date**: The date when the patron subscribed to the library.

Transactions

The **Transactions** table keeps track of all issuings, with their date of issue and date of return. It helps manage the circulation of books, tracking their status and due dates, and allowing for reporting and analysis.

- **id**: A unique identifier for each transaction.
- **book_isbn**: A foreign key referencing the “books” table, specifying the ISBN of the book involved.
- **patron_id**: A foreign key referencing the “patrons” table, identifying the patron associated.
- **issue_date**: The date when the book was issued.
- **due_date**: The date when the book is expected to be returned.
- **return_date**: The date when the book was actually returned (if returned).
- **returned**: A boolean indicating whether the book has been returned (default is false).

Functional Decomposition

The application is decomposed into smaller components, which include:

- functionality for querying the database
- general utility functions
- the main program (driver)

All database queries are isolated inside a separate file, **database.py**.

```
1 # database.py
2
3 def IssueBook(isbn, patron_id):
4     ...
5
6 def ReturnBook(isbn, patron_id):
7     ...
8
9 def ViewTransactions():
10     ...
11
12 ...
```

Listing 1: Function decomposition of database functions

These functions use the `mysql.connector` library to connect to the MySQL database and query the database.

The main driver code is present inside `main.py`, which uses an interesting strategy to dynamically manage the menu-based interface for admin functions. Instead of hardcoding separate sections for each menu option, the code defines a list containing the display text of the option and its corresponding function.


```

1 # main.py
2 actions = [
3     {
4         "text": "Issue Books",
5         "value": db.IssueBook
6     },
7     {
8         "text": "Return Books",
9         "value": db.ReturnBook
10    },
11    {
12        "text": "View Transactions",
13        "value": db.ViewTransactions
14    },
15    ...
16 ]

```

Listing 2: Mapping menu options to database functions

During runtime, the program continuously presents the user with the menu choices, and upon selection, it uses the **inspect** module to determine the parameters required by the chosen function. It then dynamically prompts the user for the necessary input, eliminating the need for repetitive code for user input handling.

For example, in case of the `ReturnBook` function, the **inspect.getfullargspec** can determine its parameters - `isbn` and `patron_id`. Using this, the driver code can dynamically create user inputs requesting for these values, and then call the function, showing the output.

```

1 # main.py
2 choice = {"text": "Return Books", "value":
  db.ReturnBook} # choice selected by the user
3
4 # the function to execute
5 func = choice["value"]
6
7 # gets list of arguments to the function
8 # ["isbn", "patron_id"]
9 args = inspect.getfullargspec(func).args
10
11 params = []
12 for arg in args:
13     params.append(input(f"Enter {arg}: "))
14
15 # call the function with the provided arguments
16 func(*params)

```

Listing 3: Dynamically determining function parameters

This strategy offers a flexible and organized way to interact with various admin functions while maintaining clarity and reducing the overall complexity of the driver code.

Control Flow of the Application

Initially on running the app, it provides with two options — to use the application as an **Admin** or an **User** (Patron). The major functionality of an app is only accessible to the admin for obvious reasons, and the user can only perform a few READ operations.

The admin operations are:

- Add/Remove/Edit a Book

- Add/Remove/Edit a Patron
- View Books/Patrons
- Search Books
- Issue/Return a Book
- View all Transactions
- View pending Transactions

Most of these functions just perform simple CRUD SQL queries and return the output (if any). The custom functionality which is implemented specifically for this project is the Issue/Return workflow.

Issuing a Book

The `IssueBook()` function takes in two arguments - `isbn` and `patron_id`. The following database commands take place:

Source Code

main.py

```
from tkinter import simpledialog
import time
import database as db
from tabulate import tabulate

def validate_date(date):
    # yyyy-mm-dd
    try:
        time.strptime(date, "%Y-%m-%d")
        return True
    except ValueError:
        return False

print("Welcome to the Library Manager!!")
print()
print("ACCESS TO : ")
print("(1) ADMIN")
print("(2) USER")
print()
choice = int(input("Enter your choice: "))
print()

if choice >= 1 and choice <= 2:
    if choice == 1:
        pwd = simpledialog.askstring("Password", "Enter your password:",
show="*")
        if pwd == "admin123123" or pwd == "libraryroot123":
            print("Recognised as admin...")
            print("Accessing administrative functions.....")
            print()
            time.sleep(2)
            while True:
                print("\t(1) Book functions")
                print("\t(2) Patron functions")
                print("\t(3) Transactions and Returns")
                print("\t(4) Exit")

                acc = int(input("\tPLEASE ENTER THE OPTION NUMBER : "))
                if acc >= 1 and acc <= 4:
                    if acc == 1:
                        print("\tAccessing.....")
                        time.sleep(2)

                        while True: #
                            print("\t\t**** BOOK FUNCTIONS ****")
```

```

print("\t\t OPTIONS : ")
print("\t\t(1) Add a book")
print("\t\t(2) Remove a book")
print("\t\t(3) Update a book")
print("\t\t(4) Issue a Book")
print("\t\t(5) Return a Book")
print("\t\t(6) Search a Book")
print("\t\t(7) View all books")
print("\t\t(8) Go back to main menu")
print("\t\t(9) Exit")

opt = int(input("\t\tPLEASE ENTER THE OPTION NUMBER : "))
if opt >= 1 and opt <= 9:
    if opt == 1:
        nISBN = input("\t\tISBN : ")
        nTITLE = input("\t\tTITLE : ")
        nAUTHOR = input("\t\tAUTHOR : ")
        nGENRE = input("\t\tGENRE (Fiction/Non-Fiction): ")
        Quantity = int(input("\t\tQUANTITY : "))
        db.AddBook(Quantity, nTITLE, nAUTHOR, nISBN, nGENRE)
        break
    elif opt == 2:
        ISBN = input("\t\tISBN : ")
        db.RemoveBook(ISBN)
        break
    elif opt == 3:
        ISBN = input("\t\tISBN : ")
        db.EditBook(ISBN)
        break
    elif opt == 4:
        ISBN = input("\t\tISBN : ")
        ID = input("\t\tID of the patron: ")
        db.IssueBook(ISBN, ID)
        break
    elif opt == 5:
        ISBN = input("\t\tISBN : ")
        ID = input("\t\tID of the patron: ")
        db.ReturnBook(ISBN, ID)
    elif opt == 6:
        while True:
            print("\t\t\t OPTIONS :")
            print("\t\t\t(1) By ISBN number")
            print("\t\t\t(2) By Author")
            print("\t\t\t(3) By Title")
            print("\t\t\t(4) By Genre")
            print("\t\t\t(5) Return to previous menu")
            print("\t\t\t(6) Exit")

            x = int(
                input(
                    "\t\t\tPLEASE ENTER THE OPTION NUMBER : "

```

```

    )
)
if x >= 1 and x <= 6:
    if x == 1:
        ISBN = input("\t\t\t\tISBN : ")
        res = db.SearchBookByISBN(ISBN)
        if res != 1:
            data, columns = res
            print(
                tabulate(
                    data,
                    headers=columns,
                    tablefmt="pretty",
                )
            )
        else:
            print("\t\t\t\tBook not found!")
            break
    elif x == 2:
        Author = input("\t\t\t\tAuthor : ")
        res = db.SearchBookByAuthor(Author)
        if res != 1:
            data, columns = res
            print(
                tabulate(
                    data,
                    headers=columns,
                    tablefmt="pretty",
                )
            )
        else:
            print("\t\t\t\tBook not found!")
            break
    elif x == 3:
        Title = input("\t\t\t\tTitle : ")
        res = db.SearchBookByTitle(Title)
        if res != 1:
            data, columns = res
            print(
                tabulate(
                    data,
                    headers=columns,
                    tablefmt="pretty",
                )
            )
        else:
            print("\t\t\t\tBook not found!")
            break
    elif x == 4:
        Genre = input("\t\t\t\tGenre : ")
        res = db.SearchBookByGenre(Genre)

```

```

        if res != 1:
            data, columns = res
            print(
                tabulate(
                    data,
                    headers=columns,
                    tablefmt="pretty",
                )
            )
        else:
            print("\t\t\t\tBook not found!")
            break
    elif x == 5:
        break
    elif x == 6:
        exit()
    else:
        print("\t\t\t\tIncorrect option entered!")
elif opt == 7:
    db.ViewBooks()
elif opt == 8:
    break # \t ##### Command executed
elif opt == 9:
    exit()
else:
    print("\t\t\t\tIncorrect option entered!")
elif acc == 2:
    print("\tAccessing.....")
    time.sleep(2)

while True:
    print("\t\t\t\t**** PATRON FUNCTIONS ****")
    print("\t\t\t\tOPTIONS : ")
    print("\t\t\t\t(1) Add Patron")
    print("\t\t\t\t(2) Remove Patron")
    print("\t\t\t\t(3) Update Patron")
    print("\t\t\t\t(4) Search Patron")
    print("\t\t\t\t(5) View Patrons")
    print("\t\t\t\t(6) Go back to main menu")
    print("\t\t\t\t(7) Exit")
    opt = int(input("\t\t\t\tPLEASE ENTER THE OPTION NUMBER : "))
    if opt >= 1 and opt <= 7:
        if opt == 1:
            ID = input("\t\t\t\tID : ")
            Email = input("\t\t\t\tEmail : ")
            Patron_Name = input("\t\t\t\tPatron Name : ")
            Subcription_Date = input(
                "\t\t\t\tEnter Date(YYYY-MM-DD) : "
            )
            if not validate_date(Subcription_Date):
                print("\t\t\t\t\tIncorrect date entered!")

```



```

        print("\t\t\t\tPatron not found!")
        break
    elif x == 3:
        break
    elif x == 4:
        exit()
    else:
        print("Incorrect option entered!")
elif opt == 5:
    db.ViewPatrons()
elif opt == 6:
    break
elif opt == 7:
    exit()
else:
    print("Incorrect option entered!")
elif acc == 3:
    print("Accessing.....")
    time.sleep(2)

while True:
    print("\t\t**** TRANSACTIONS ****")
    print("\t\tOPTIONS : ")
    print("\t\t(1) View Transactions")
    print("\t\t(2) View all pending returns")
    print("\t\t(3) Return to previous menu")
    print("\t\t(4) Exit")

    opt = int(input("\t\tPLEASE ENTER THE OPTION NUMBER : "))
    if opt >= 1 and opt <= 4:
        if opt == 1:
            db.ViewTransactions()
        elif opt == 2:
            db.ViewTransactionsPending()
        elif opt == 3:
            break
        elif opt == 4:
            exit()
    else:
        print("Incorrect option entered!")
elif acc == 4:
    exit()
else:
    print("Incorrect option entered!")
else:
    print("Wrong Password !!")
    exit()
elif choice == 2:
    ID = simplifiedialog.askstring("ID", "Enter your Patron ID:")
    if db.SearchPatronByID(ID) == 1:
        print("Patron ID not found !!")

```

```

    exit()
else:
    print("Recognised as our registered patron....")
    print("Accessing patron functions.....")
    print()
    time.sleep(2)
    while True:
        print("\tOPTIONS :")
        print("\t(1) Search a book")
        print("\t(2) View all books")
        print("\t(3) View my issued books")
        print("\t(4) Exit")
        acc = int(input("\tPLEASE ENTER THE OPTION NUMBER : "))
        if acc >= 1 and acc <= 4:
            if acc == 1:
                while True:
                    print("\t\tOPTIONS :")
                    print("\t\t(1) By ISBN number")
                    print("\t\t(2) By Author")
                    print("\t\t(3) By Title")
                    print("\t\t(4) By Genre")
                    print("\t\t(5) Return to previous menu")
                    print("\t\t(6) Exit")
                    x = int(input("\t\t\tPLEASE ENTER THE OPTION NUMBER : "))
                    if x >= 1 and x <= 6:
                        if x == 1:
                            ISBN = input("\t\t\tISBN : ")
                            res = db.SearchBookByISBN(ISBN)
                            if res != 1:
                                data, columns = res
                                print(
                                    tabulate(
                                        data, headers=columns, tablefmt="psql"
                                    )
                                )
                            else:
                                print("\t\t\t\tBook not found!")
                                break
                        elif x == 2:
                            Author = input("\t\t\tAuthor : ")
                            res = db.SearchBookByAuthor(Author)
                            if res != 1:
                                data, columns = res
                                print(
                                    tabulate(
                                        data, headers=columns, tablefmt="pretty"
                                    )
                                )
                            else:
                                print("\t\t\t\tBook not found!")
                                break

```

```

elif x == 3:
    Title = input("\t\t\tTitle : ")
    res = db.SearchBookByTitle(Title)
    if res != 1:
        data, columns = res
        print(
            tabulate(
                data, headers=columns, tablefmt="pretty"
            )
        )
    else:
        print("\t\t\t\tBook not found!")
        break
elif x == 4:
    Genre = input("\t\t\tGenre : ")
    res = db.SearchBookByGenre(Genre)
    if res != 1:
        data, columns = res
        print(
            tabulate(
                data, headers=columns, tablefmt="pretty"
            )
        )
    else:
        db.SearchBookByAuthor(Author)
elif x == 5:
    break
elif x == 6:
    exit()
elif acc == 2:
    db.ViewBooks()
elif (
    acc == 3
):

```

```

#####
query = 'SELECT b.title "Book", p.name "Issued by", t.issue_date
"Issued On", t.due_date "Due Date", IFNULL(t.return_date, \'Not Returned\')
\'Returned On\' FROM transactions t JOIN books b ON t.book_isbn = b.isbn JOIN
patrons p ON p.id = t.patron_id WHERE t.patron_id = %s'

```

```

res = db.Query(query, (ID,))
if res != 1:
    data, columns = res
    print(
        tabulate(
            data,
            headers=columns,
            tablefmt="pretty",
            stralign="center",
        )
    )
else:

```

```

        print("\t\t\t\t\tNo books issued!")
    elif acc == 4:
        exit()
    else:
        print("Incorrect option entered!")
else:
    print("Incorrect option entered!")
    exit()

```

database.py

```

import mysql.connector
from tabulate import tabulate

db = mysql.connector.connect(
    user="root", host="localhost", passwd="password", database="library"
)
cursor = db.cursor()
cursor.execute("USE library")

# Try to execute SQL command
def TrySQLCommand(query, values=None):
    cursor.execute(query, values)
    result = cursor.fetchall()
    if cursor.rowcount == 0:
        raise ValueError("No matching records found.")
    return result, [desc[0] for desc in cursor.description]

# check for valid ISBN number
def ValidateISBN(isbn):
    # Remove hyphens and spaces
    isbn = isbn.replace("-", "").replace(" ", "")

    # ISBN-10
    if len(isbn) == 10:
        if not isbn[-1].isdigit():
            return False
        if isbn[-1].upper() == "X":
            isbn_sum = (
                sum(int(digit) * (i + 1) for i, digit in enumerate(isbn[:-1])) + 10
            )
        else:
            isbn_sum = sum(int(digit) * (i + 1) for i, digit in enumerate(isbn))
        return isbn_sum % 11 == 0

    # ISBN-13
    elif len(isbn) == 13:
        if not isbn.isdigit():
            return False

```

```

        isbn_sum = sum(
            int(digit) * (1 if i % 2 == 0 else 3) for i, digit in enumerate(isbn)
        )
        return isbn_sum % 10 == 0

    return False

# Edit Books Table Functions
# Replace Books
def ReplaceBook(oISBN, nQuantity, nTITLE, nAUTHOR, nISBN, nGenre):
    if ValidateISBN(nISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    try:
        deletebook = "DELETE FROM books WHERE ISBN= %s"
        cursor.execute(deletebook, (oISBN,))
    except ValueError:
        db.rollback()
        return 1
    newbooks = "INSERT INTO books (quantity, title, author, isbn, genre)
VALUES(%s, %s, %s, %s, %s)"
    cursor.execute(newbooks, (nQuantity, nTITLE, nAUTHOR, nISBN, nGenre))
    db.commit()

# Add Books
def AddBook(nQuantity, nTITLE, nAUTHOR, nISBN, nGenre):
    if ValidateISBN(nISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    try:
        newbooks = "INSERT INTO books(quantity, title, author, isbn, genre)
VALUES(%s, %s, %s, %s, %s)"
        cursor.execute(newbooks, (nQuantity, nTITLE, nAUTHOR, nISBN, nGenre))
        db.commit()
    except ValueError:
        db.rollback()
        return 1

# Remove Books
def RemoveBook(ISBN):
    try:
        deletebook = "DELETE FROM books WHERE isbn= %s;"
        TrySQLCommand(deletebook, (ISBN,))
        db.commit()
    except ValueError:
        print("ISBN number not found. Check and Try Again!")
        db.rollback()
        return 1

```

```

# Searching Books
# By ISBN
def SearchBookByISBN(ISBN):
    try:
        SearchBook = "SELECT * FROM books WHERE isbn= %s;"
        return TrySQLCommand(SearchBook, (ISBN,))
    except ValueError:
        db.rollback()
        return 1

# By Author
def SearchBookByAuthor(Author):
    try:
        SearchBook = "SELECT * FROM books WHERE author LIKE %s"
        return TrySQLCommand(SearchBook, ("% " + Author + "%",))
    except ValueError:
        db.rollback()
        return 1

# By Title
def SearchBookByTitle(Title):
    try:
        SearchBook = "SELECT * FROM books WHERE title LIKE %s"
        return TrySQLCommand(SearchBook, ("% " + Title + "%",))
    except ValueError:
        db.rollback()
        return 1

# By Genre
def SearchBookByGenre(Genre):
    try:
        SearchBook = "SELECT * FROM books WHERE genre LIKE %s"
        return TrySQLCommand(SearchBook, (Genre,))
    except ValueError:
        db.rollback()
        return 1

def EditBook(ISBN):
    Search = SearchBookByISBN(ISBN)
    if Search == 1:
        return 1
    else:
        print(
            tabulate(
                Search,

```

```

        headers=["ISBN", "Title", "Author", "Quantity", "Genre"],
        tablefmt="pretty",
    )
)
print("OPTIONS : ")
print("(1) ---> ISBN number")
print("(2) ---> Book Author")
print("(3) ---> Book Title")
print("(4) ---> Quantity")
print("(5) ---> Genre")
print("(6) ---> Remove book")
option = int(input("Enter OPTION NUMBER of what you want to edit : "))
if option >= 1 and option <= 6:
    if option == 1:
        id = input("ENTER THE NEW ISBN NUMBER OF BOOK : ")
        if ISBN != id and ValidateISBN(id):
            query = "UPDATE books SET isbn= %s WHERE isbn=%s"
            cursor.execute(
                query,
                (
                    id,
                    ISBN,
                ),
            )
            db.commit()
        else:
            print("Failed to edit! Try again!")
            db.rollback()
            return 1
    elif option == 2:
        Author = input("ENTER THE NEW AUTHOR NAME OF THE BOOK : ")
        cursor.execute(
            "UPDATE books SET author=%s WHERE isbn=%s",
            (
                Author,
                ISBN,
            ),
        )
        db.commit()
    elif option == 3:
        Title = input("ENTER THE NEW TITLE OF BOOK : ")
        cursor.execute(
            "UPDATE books SET title=%s WHERE isbn=%s",
            (
                Title,
                ISBN,
            ),
        )
        db.commit()
    elif option == 4:
        quantity = input("ENTER THE QUANTITY OF BOOK AVAILABLE : ")

```

```

        query = "UPDATE books SET quantity= %s WHERE isbn=%s"
        cursor.execute(
            query,
            (
                quantity,
                ISBN,
            ),
        )
        db.commit()
    elif option == 5:
        genre = input("ENTER THE NEW GENRE OF BOOK : ")
        query = "UPDATE books SET genre= %s WHERE isbn=%s"
        cursor.execute(
            query,
            (
                genre,
                ISBN,
            ),
        )
        db.commit()
    elif option == 6:
        RemoveBook(ISBN)

# Issue Books to Patron and updating the return status
def IssueBook(ISBN, ID):
    if ValidateISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    if SearchBookByISBN(ISBN) == 1:
        print("ISBN number not found. Check and Try Again!")
        return 1
    if SearchPatronByID(ID) == 1:
        print("Patron ID not found. Check and Try Again!")
        return 1
    # check if patron has less than 3 unreturned books
    check1 = (
        "SELECT COUNT(*) FROM transactions WHERE patron_id = %s AND returned IS FALSE"
    )
    cursor.execute(check1, (ID,))
    if cursor.fetchone()[0] >= 3:
        print(
            "PATRON HAS ALREADY ISSUED 3 BOOKS! \nPLEASE RETURN A BOOK AND TRY AGAIN!"
        )
    )
    return 1

# check if patron currently has this book
check2 = "SELECT * FROM transactions WHERE patron_id = %s AND book_isbn = %s AND returned IS FALSE"

```



```

cursor.execute(check2, (ID, ISBN))
if cursor.rowcount > 0:
    print("PATRON HAS ALREADY ISSUED THIS BOOK!")
    return 1

query = "UPDATE books SET quantity = quantity - 1 WHERE isbn = %s AND
quantity >= 1"
cursor.execute(query, (ISBN,))
if cursor.rowcount == 1:
    try:
        issue = "INSERT INTO transactions (book_isbn, patron_id, issue_date,
due_date) VALUES (%s, %s, CURDATE(), DATE(CURDATE() + 7))"
        cursor.execute(issue, (ISBN, ID))
        db.commit()
        print("Book issued successfully!")
    except:
        db.rollback()
        print("Book issue failed!")
        return 1
else:
    db.rollback()
    print(
        "BOOK ISSUE COULD NOT BE COMPLETED AS BOOK OUT OF STOCK! \nPLEASE
CHOOSE ANOTHER BOOK AND TRY AGAIN!"
    )
    return 1

def ReturnBook(ISBN, ID):
    if ValidateISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    if SearchBookByISBN(ISBN) == 1:
        print("ISBN number not found. Check and Try Again!")
        return 1
    if SearchPatronByID(ID) == 1:
        print("Patron ID not found. Check and Try Again!")
        return 1
    trans = "UPDATE transactions SET return_date = CURDATE(), returned = TRUE
WHERE book_isbn = %s AND patron_id = %s AND return_date IS NULL AND returned
= FALSE"
    cursor.execute(trans, (ISBN, ID))
    if cursor.rowcount == 0:
        print("No book issued to this patron with this ISBN number!")
        return 1

    db.commit()

    query = "UPDATE books SET quantity = quantity + 1 WHERE ISBN = %s"
    cursor.execute(query, (ISBN,))
    db.commit()

```

```

print("Book returned successfully!")

def AddPatron(ID, Email, Patron_Name, Subscription_Date):
    newpatron = (
        "INSERT INTO patrons (id, email, name, subscription_date) VALUES(%s,%s,%s,%s)"
    )
    cursor.execute(
        newpatron,
        (
            ID,
            Email,
            Patron_Name,
            Subscription_Date,
        ),
    )
    db.commit()

def RemovePatron(ID):
    try:
        delet Patron = "DELETE FROM patrons WHERE id= %s;"
        TrySQLCommand(delet Patron, (ID,))
    except ValueError:
        db.rollback()
    return 1
    db.commit()

def EditPatron(ID):
    Search = SearchPatronByID(ID)
    if Search == 1:
        return 1
    else:
        print(
            tabulate(
                Search,
                headers=["ID", "Email", "Name", "Subscription Date"],
                tablefmt="pretty",
            )
        )
        print("OPTIONS : ")
        print("(1) ---> Patron ID number")
        print("(2) ---> Patron Email")
        print("(3) ---> Patron Name")
        print("(4) ---> Renew Patron Subscription Date")
        print("(5) ---> Remove Patron")
        option = int(input("Enter OPTION NUMBER of what you want to edit : "))
        if option >= 1 and option <= 5:

```

```

if option == 1:
    id = input("ENTER THE NEW 8-DIGIT UNIQUE ID OF PATRON : ")
    if ID != id and len(id) == 8:
        query = "UPDATE patron SET id= %s WHERE id=%s"
        cursor.execute(
            query,
            (
                id,
                ID,
            ),
        )
        db.commit()
    else:
        print("Failed to edit! Try again!")
        db.rollback()
        return 1
elif option == 2:
    Email = input("ENTER THE NEW EMAIL OF PATRON : ")
    if "@" in Email and "." in Email:
        cursor.execute(
            "UPDATE patron SET email= %s WHERE id= %s",
            (
                Email,
                ID,
            ),
        )
        db.commit()
    else:
        print("Enter a VALID EMAIL and Try Again!")
        db.rollback()
        return 1
elif option == 3:
    Name = input("ENTER THE NEW NAME OF PATRON : ")
    cursor.execute(
        "UPDATE patrons SET name= %s WHERE id= %s",
        (
            Name,
            ID,
        ),
    )
    db.commit()
elif option == 4:
    query = "UPDATE patrons SET subscription_date=DATE(NOW()) WHERE
id=%s"
    cursor.execute(query, (ID,))
    db.commit()
elif option == 5:
    RemovePatron(ID)

def SearchPatronByID(ID):

```

```

try:
    SearchPatron = "SELECT * FROM patrons WHERE id= %s"
    return TrySQLCommand(SearchPatron, (ID,))
except ValueError:
    db.rollback()
    return 1

def SearchPatronByName(Patron_Name):
    try:
        SearchPatron = "SELECT * FROM patrons WHERE name LIKE %s"
        return TrySQLCommand(SearchPatron, ("% " + Patron_Name + "%",))
    except ValueError:
        db.rollback()
        return 1

def ViewTransactions():
    query = 'SELECT t.id "ID", b.title "Book", p.name "Issued By", t.issue_date
    "Issued On", t.due_date "Due On", IFNULL(t.return_date, "Not Returned")
    "Returned On" FROM transactions t JOIN books b ON t.book_isbn = b.isbn JOIN
    patrons p ON t.patron_id = p.id ORDER BY t.book_isbn, t.issue_date;'
    cursor.execute(query)
    # Fetch column names
    columns = [desc[0] for desc in cursor.description]
    # Fetch data
    data = cursor.fetchall()
    # Print tabulated data
    print(tabulate(data, headers=columns, tablefmt="pretty"))

def ViewTransactionsPending():
    query = "SELECT * FROM transactions WHERE returned = false"
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    data = cursor.fetchall()
    print(tabulate(data, headers=columns, tablefmt="pretty"))

def ViewPatrons():
    query = """SELECT patrons.id "ID", name "Name", email "Email ID",
    subscription_date "Subscribed on", COUNT(transactions.id) "Unreturned Books"
    FROM patrons
    LEFT JOIN transactions ON patrons.id = transactions.patron_id AND
    transactions.returned = false
    GROUP BY patrons.id, email, name, subscription_date;"""
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    data = cursor.fetchall()
    print(tabulate(data, headers=columns, tablefmt="pretty"))

```

```

def ViewBooks():
    query = "SELECT * FROM books"
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    data = cursor.fetchall()
    print(tabulate(data, headers=columns, tablefmt="pretty"))

def Query(query, values=None):
    cursor.execute(query, values)
    data = cursor.fetchall()
    return data, [desc[0] for desc in cursor.description]

```

db.sql

```

DROP DATABASE IF EXISTS library;

CREATE DATABASE library;

USE library;

CREATE TABLE books (
    isbn VARCHAR(13) NOT NULL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR(50) NOT NULL,
    quantity INT NOT NULL,
    genre ENUM("Fiction", "Non-Fiction") NOT NULL
);

CREATE TABLE patrons (
    id CHAR(8) NOT NULL PRIMARY KEY,
    email VARCHAR(50) NOT NULL,
    name VARCHAR(30) NOT NULL,
    subscription_date DATE NOT NULL,
    CONSTRAINT id_length CHECK (LENGTH(TRIM(id)) = 8),
    CONSTRAINT valid_email CHECK (email LIKE '%@%.%')
);

CREATE TABLE transactions (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    book_isbn VARCHAR(13) NOT NULL,
    patron_id CHAR(8) NOT NULL,
    issue_date DATE NOT NULL,
    due_date DATE NOT NULL,
    return_date DATE,
    returned BOOLEAN NOT NULL DEFAULT FALSE,
    FOREIGN KEY (book_isbn) REFERENCES books(isbn) ON DELETE CASCADE,
    FOREIGN KEY (patron_id) REFERENCES patrons(id) ON DELETE CASCADE
);

```

Screenshots

Bibliography

- [1] “MySQL 8.1 Reference.” <https://dev.mysql.com/doc/refman/8.1/en/>
- [2] “MySQL Connector/Python Developer's Guide.” <https://dev.mysql.com/doc/connector-python/en/>