2023

Project Report on Software for Library Management

Computer Science Project

Dipam Sen **Arghya Kumar**

Table of Contents

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At.

Thus, we'll use the `mysql` module

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 information about book issuance

Code

The complete source code for the application as well as this document is accessible on the GitHub repository for the project: https://github.com/dipamsen/LibraryManager

`data.py`

```python
import mysql.connector
from tabulate import tabulate
import random

db = mysql.connector.connect(
    user="root", host="localhost", passwd="libraryroot123",
database="library"
)
cursor = db.cursor()
cursor.execute("USE library")


# Raise error in SQL even though there is no error
def RaiseSQLError(query, values=None):
    cursor.execute(query, values)
    result = cursor.fetchall()
    if cursor.rowcount == 0:
        raise ValueError("No matching records found.")
    return result


# check for valid ISBN number
def ValidISBN(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 ReplaceBooks(oISBN, nquantity, nTITLE, nAUTHOR, nISBN):
    if ValidISBN(oISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    try:
        deletebook = "DELETE FROM books WHERE ISBN= %s"
        RaiseSQLError(deletebook, (oISBN,))
    except ValueError:
        db.rollback()
        return 1
    newbooks = "INSERT INTO books (quantity,title,author,isbn) VALUES(%s,
%s,%s,%s)"
    cursor.execute(newbooks, (nquantity, nTITLE, nAUTHOR, nISBN))
    db.commit()


# Add Books
def AddBooks(nquantity, nTITLE, nAUTHOR, nISBN):
    newbooks = "INSERT INTO books(quantity,title,author,isbn) VALUES(%s,
%s,%s,%s)"
    cursor.execute(newbooks, (nquantity, nTITLE, nAUTHOR, nISBN))
    db.commit()


# Remove Books
def RemoveBooks(ISBN):
    if ValidISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    try:
        deletebook = "DELETE FROM books WHERE isbn= %s;"
        RaiseSQLError(deletebook, (ISBN,))
    except ValueError:
        db.rollback()
        return 1
    db.commit()


# Searching Books
# By ISBN
def SearchBook_ISBN(ISBN):
    if ValidISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    try:
        SearchBook = "SELECT * FROM books WHERE isbn= %s;"
        return RaiseSQLError(SearchBook, (ISBN,))
    except ValueError:
        db.rollback()
        return 1


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


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


genres = ["Fiction", "Non Fiction"]


# By Genre
def SearchBook_Genre(Genre):
    if Genre not in genres:
        print("INVALID GENRE!!")
        return 1
    try:
        SearchBook = "SELECT * FROM books WHERE genre LIKE %s"
        return RaiseSQLError(SearchBook, (Genre,))
    except ValueError:
        db.rollback()
        return 1


def Editbooks(ISBN):
    if ValidISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    Search = SearchBook_ISBN(ISBN)
    if Search == 1:
        return 1
    else:
        print("OPTIONS : ")
        print("(1) ---> ISBN number")
        print("(2) ---> Book Author")
        print("(3) ---> Book Title")
        print("(4) ---> Quantity")
        print("(5) ---> Remove book")
        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 ISBN NUMBER OF BOOK : ")
                if ISBN != id and ValidISBN(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:
                RemoveBooks(ISBN)


# Issue Books to Patron and updating the return status
def Issue_Books(ISBN, Unique_ID):
    if ValidISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    if SearchBook_ISBN(ISBN) == 1:
        print("ISBN number not found. Check and Try Again!")
        return 1
    if SearchPatron_ID(Unique_ID) == 1:
        print("Patron ID not found. Check and Try Again!")
        return 1
    query = "UPDATE books SET quantity = quantity - 1 WHERE isbn = %s AND
quantity >= 1"
    cursor.execute(query, (ISBN,))
    if cursor.rowcount == 1:
        db.commit()

        id = random.randint(10000000, 99999999)
        issue = "INSERT INTO issues (id, book_isbn, patron_id, issue_date,
return_date) VALUES (%s, %s, %s, CURDATE(), DATE(CURDATE() + 7))"
        cursor.execute(issue, (id, ISBN, Unique_ID))

        print("Book issued successfully!")
        returnstatus = "UPDATE patrons SET return_status = FALSE WHERE id
= %s"
        cursor.execute(returnstatus, (Unique_ID,))
        db.commit()
    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 return_books(ISBN, Unique_ID):
    if ValidISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    if SearchBook_ISBN(ISBN) == 1:
        print("ISBN number not found. Check and Try Again!")
        return 1
    if SearchPatron_ID(Unique_ID) == 1:
        print("Patron ID not found. Check and Try Again!")
        return 1
    query = "UPDATE books SET quantity = quantity + 1 WHERE ISBN = %s"
    cursor.execute(query, (ISBN,))
    db.commit()

    # TODO: update "returned" inside issues table
    print("Book returned successfully!")


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


def RemovePatron(Unique_ID):
    try:
        deletepatron = "DELETE FROM patrons WHERE id= %s;"
        RaiseSQLError(deletepatron, (Unique_ID,))
    except ValueError:
        db.rollback()
        return 1
    db.commit()


def EditPatron(Unique_ID):
    Search = SearchPatron_ID(Unique_ID)
    if Search == 1:
        return 1
    else:
        print("OPTIONS : ")
        print("(1) ---> Patron ID number")
        print("(2) ---> Patron Email")
        print("(3) ---> Patron Name")
        print("(4) ---> RENEW Patron Subcription 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 Unique_ID != id and len(id) == 8:
                    query = "UPDATE patron SET id= %s WHERE id=%s"
                    cursor.execute(
                        query,
                        (
                            id,
                            Unique_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,
                            Unique_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,
                        Unique_ID,
                    ),
                )
                db.commit()
            elif option == 4:
                query = "UPDATE patrons SET subscription_date=DATE(NOW())
WHERE id=%s"
                cursor.execute(query, (Unique_ID,))
                db.commit()
            elif option == 5:
                RemovePatron(Unique_ID)


def SearchPatron_ID(Unique_ID):
    try:
        SearchPatron = "SELECT * FROM patrons WHERE id= %s"
        return RaiseSQLError(SearchPatron, (Unique_ID,))
    except ValueError:
        db.rollback()
        return 1


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


def Checkouts():
    query = "SELECT * FROM issues"
    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 Patron():
    query = 'SELECT id "ID", name "Name", email "Email ID",
subscription_date "Subscribed on", IF(return_status, "Yes", "No")
"Returned?" FROM patrons'
    cursor.execute(query)
    columns = [desc[0] for desc in cursor.description]
    data = cursor.fetchall()
    print(tabulate(data, headers=columns, tablefmt="pretty"))


def GenerateBookReport():
    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"))


# Issue_Books("9780060838676", "dgiidgsd")


# AddPatron("29839329", "python@main.com", "Python", "2021-05-01")
# RemovePatron("dgiidgsd")
# Editbooks("9780060838676")
# EditPatron("29839329")
```

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
);

INSERT INTO
    books (quantity, title, author, isbn)
VALUES
    (
        3,
        "The Sun Also Rises",
        "Ernest Hemingway",
        "9780743297332"
    ),
    (
        2,
        "To Kill A Mockingbird",
        "Harper Lee",
        "9780099549482"
    ),
    (
        2,
        "Their Eyes Were Watching God",
        "Zora Neale Hurston",
        "9780060838676"
    );

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,
    return_status BOOL NOT NULL DEFAULT TRUE,
    CONSTRAINT id_length CHECK (LENGTH(TRIM(id)) = 8),
    CONSTRAINT valid_email CHECK (email LIKE '%@%.%')
);

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

gui.py

```
import tkinter as tk
import tkinter.font as tkFont

screens = []


def goto(screen):
    for i in screens:
        i.pack_forget()
```

```python
    screen.pack()


app = tk.Tk()
font = tkFont.Font(family="Helvetica", size=12, weight="normal")
app.title("Library Management System")
app.geometry("500x500")
app.configure(bg="light blue")
app.option_add("*Font", font)

homescrn = tk.Frame(app)
homescrn.config(background="SystemWindow")
# homescrn.attributes("-transparent", True)
screens.append(homescrn)
homescrn.pack()


label = tk.Label(homescrn, text="Library Management System",
font=("Arial", 20))
label.pack(pady=10)

app.mainloop()
```

`main.py`

```python
# Library Manager
# Author: Dipam Sen and Arghya Kumar

print("Welcome to the Library Manager")
```