2023

Project Report on Software for Library Management

Computer Science Project

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

Introduction

This application aims to streamline the process of issuing and returning books in a library. Its goal is to easily keep track of library transactions and display them with ease. It also allows to update information as well as create and delete data from the database.

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.

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.

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 records interactions between patrons and books, including issuing and returning. It helps

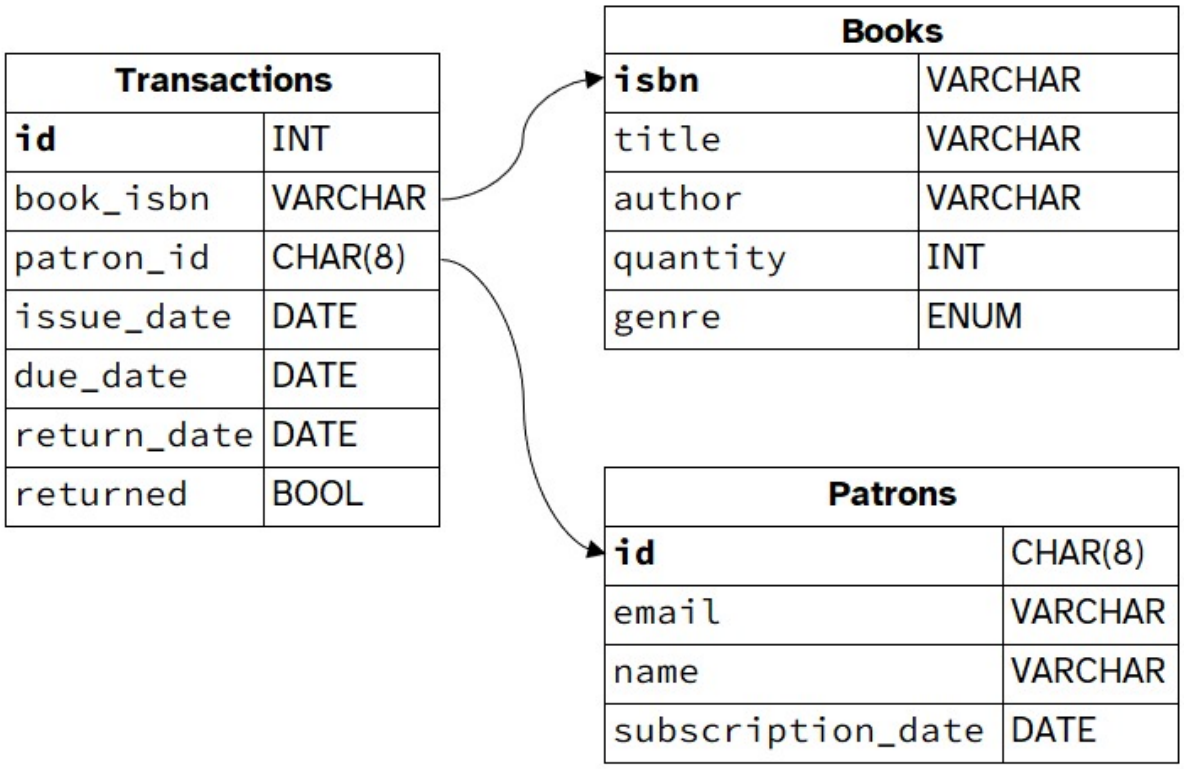

Figure 1: Database Entity Relationship diagram

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).

Code

data.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


# 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,))
        db.commit
    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):
    if ValidateISBN(ISBN) == False:
        print("INVALID ISBN NUMBER!!")
        return 1
    Search = SearchBookByISBN(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) ---> 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
    query = "UPDATE books SET quantity = quantity - 1 WHERE isbn = %s AND
quantity >= 1"
    cursor.execute(query, (ISBN,))
    if cursor.rowcount == 1:
        db.commit()

        issue = "INSERT INTO transactions (book_isbn, patron_id, issue_date,
return_date) VALUES (%s, %s, CURDATE(), DATE(CURDATE() + 7))"
        cursor.execute(issue, (ISBN, ID))
        db.commit()
        print("Book issued successfully!")
    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, Subcription_Date):
    newpatron = (
        "INSERT INTO patrons (id, email, name, subscription_date) VALUES(%s,
%s,%s,%s)"
    )
    cursor.execute(
        newpatron,
        (
            ID,
            Email,
            Patron_Name,
            Subcription_Date,
        ),
    )
    db.commit()


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

```
def EditPatron(ID):
    Search = SearchPatronByID(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 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 * FROM transactions"
    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 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
);
```