Table of Contents

Introduction

A **Mini Library Management System** is a software application designed to simplify the organization and management of library resources. Libraries, whether in schools, colleges, or small communities, often face challenges in tracking books, managing member records, and recording borrowing or return transactions. This system aims to provide a user-friendly solution that automates these processes, reducing manual effort and minimizing errors.

Developed in **Python**, the system leverages the simplicity and versatility of the language to create an efficient and easy-to-use application. It allows library staff or users to perform key tasks such as adding new books, issuing books to members, returning books, and checking the availability of resources. By storing data digitally, it ensures quick access to information, improves accuracy in record-keeping, and saves time.

The **Mini Library Management System** is ideal for small-scale libraries where a full-fledged enterprise system may not be necessary. Its design focuses on clarity, simplicity, and functionality, making it accessible even to users with minimal technical expertise. Overall, this system represents a step toward modernizing library management and enhancing the experience of both librarians and library users.

Operation.Py

```python
# operations.py — Core logic for Library Management System

books = {}
members = []
genres = ("Fiction", "Non-Fiction", "Sci-Fi", "Thriller", "Romance", "Biography")

def add_book(isbn, title, author, genre, total_copies):  3 usages  AbubakarrJalloh200 *
    if isbn in books:
        print("Book already exists.")
        return
    if genre not in genres:
        print("Invalid genre. Please choose from:", genres)
        return
    books[isbn] = {
        "title": title,
        "author": author,
        "genre": genre,
        "total_copies": total_copies,
        "borrowed": 0
    }
    print(f"Book '{title}' added successfully.")

def add_member(member_id, name, email):  3 usages  AbubakarrJalloh200 *
    if any(m["member_id"] == member_id for m in members):
        print("Member already exists.")
        return
    members.append({
        "member_id": member_id,
        "name": name,
        "email": email,
        "borrowed_books": []
    })
    print(f"Member '{name}' added successfully.")

def borrow_book(member_id, isbn):  3 usages  AbubakarrJalloh200 *
    member = next((m for m in members if m["member_id"] == member_id), None)
    if not member:
        print("Invalid member ID.")
        return
    if isbn not in books:
        print("Book not found.")
        return
```

```
    if books[isbn]["borrowed"] >= books[isbn]["total_copies"]:
        print("No copies available.")
        return
    if isbn in member["borrowed_books"]:
        print("Member already borrowed this book.")
        return

    member["borrowed_books"].append(isbn)
    books[isbn]["borrowed"] += 1
    print(f"Book '{books[isbn]['title']}' borrowed by {member['name']}.")

def return_book(member_id, isbn):  2 usages  AbubakarrJalloh200 *
    """
    Return a borrowed book to the library.
    Checks that the member and book exist and the member has actually borrowed it.
    """
    member = next((m for m in members if m["member_id"] == member_id), None)
    if not member:
        print("Invalid member ID.")
        return
    if isbn not in books:
        print("Book not found.")
        return
    if isbn not in member["borrowed_books"]:
        print("Book not borrowed by this member.")
        return

    member["borrowed_books"].remove(isbn)
    books[isbn]["borrowed"] = max(books[isbn]["borrowed"] - 1, 0)
    print(f"Book '{books[isbn]['title']}' returned by {member['name']} successfully.")

def update_book(isbn, **kwargs):  2 usages  AbubakarrJalloh200 *
    if isbn in books:
        books[isbn].update(kwargs)
        print(f"Book '{isbn}' updated successfully.")
    else:
        print("Book not found.")
```

```
def update_member(member_id, **kwargs):  1 usage  AbubakarrJalloh200 *
    for m in members:
        if m["member_id"] == member_id:
            m.update(kwargs)
            print(f"Member '{member_id}' updated successfully.")
            return
    print("Member not found.")

def delete_book(isbn):  2 usages  AbubakarrJalloh200 *
    if isbn in books:
        del books[isbn]
        print(f"Book '{isbn}' deleted successfully.")
    else:
        print("Book not found.")

def delete_member(member_id):  1 usage  AbubakarrJalloh200 *
    global members
    before_count = len(members)
    members = [m for m in members if m["member_id"] != member_id]
    if len(members) < before_count:
        print(f"Member '{member_id}' deleted successfully.")
    else:
        print("Member not found.")
```

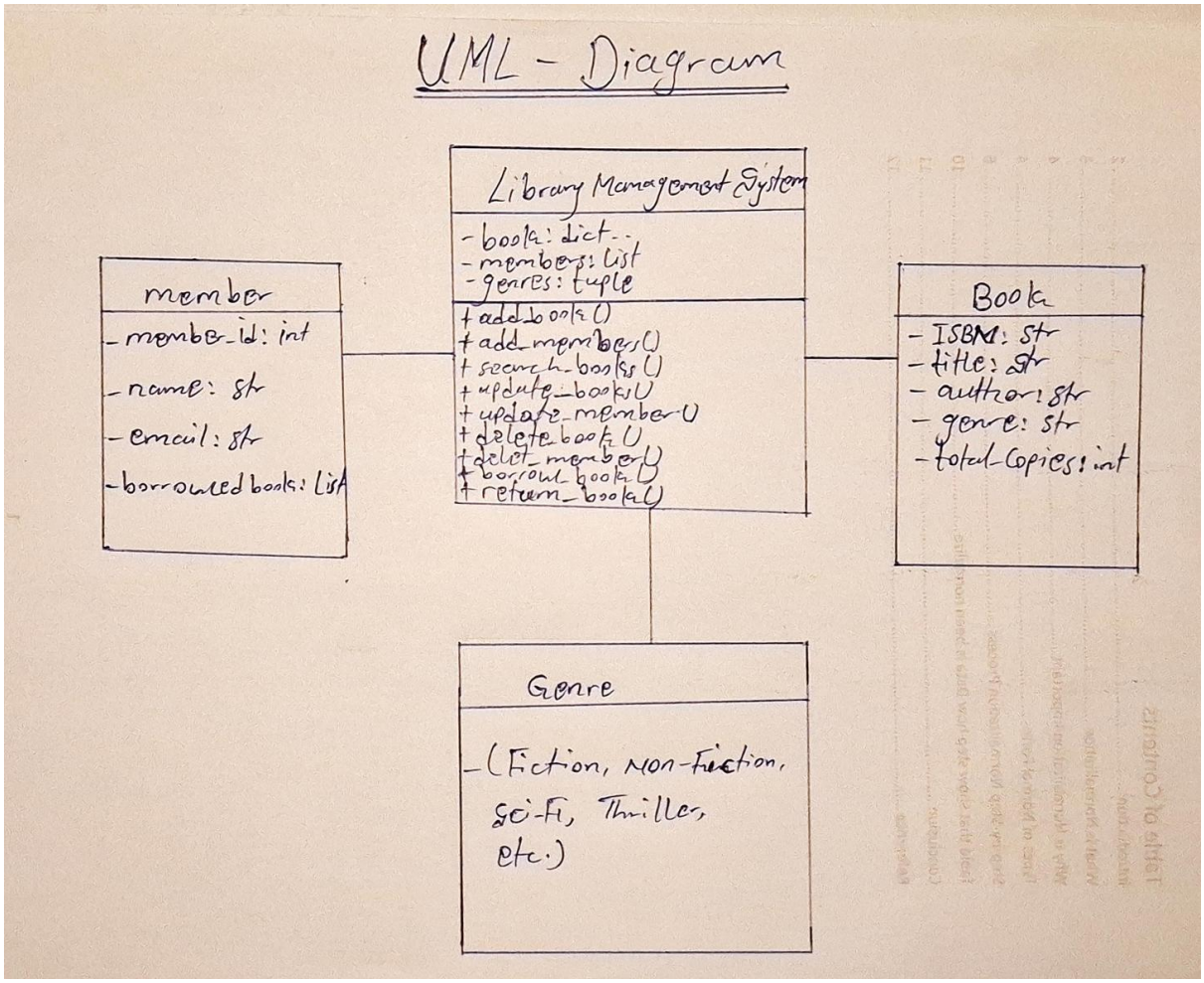
Demo.Py

```python
# demo.py — Demonstrates how the system works

from operations import *

print("===== LIBRARY SYSTEM DEMO =====")

# Add books
add_book(isbn: "111", title: "1984", author: "George Orwell", genre: "Fiction", total_copies: 3)
add_book(isbn: "222", title: "The Martian", author: "Andy Weir", genre: "Sci-Fi", total_copies: 2)

# Add members
add_member(member_id: 1, name: "Alice", email: "alice@example.com")
add_member(member_id: 2, name: "Bob", email: "bob@example.com")

# Borrow & Return
print("\n--- Borrowing Books ---")
borrow_book(member_id: 1, isbn: "111")
borrow_book(member_id: 1, isbn: "222")

print("\n--- Returning Book ---")
return_book(member_id: 1, isbn: "111")

# Update info
print("\n--- Updating Info ---")
update_book(isbn: "222", total_copies=5)
update_member(member_id: 2, email="bob.new@example.com")

# Delete
print("\n--- Deleting ---")
delete_book("111")
delete_member(2)

print("\n--- Final Data ---")
print("Books:", books)
print("Members:", members)
```

Test.Py

```python
# tests.py — Unit testing using assert

from operations import *

# Reset
books.clear()
members.clear()

# Test add functions
add_book(isbn: "123", title: "Book A", author: "Author A", genre: "Fiction", total_copies: 3)
assert "123" in books, "Book not added."

add_member(member_id: 1, name: "John Doe", email: "john@example.com")
assert any(m["member_id"] == 1 for m in members), "Member not added."

# Test borrowing
borrow_book(member_id: 1, isbn: "123")
assert "123" in members[0]["borrowed_books"], "Book not borrowed."

# Test returning
return_book(member_id: 1, isbn: "123")
assert "123" not in members[0]["borrowed_books"], "Book not returned."

# Test updating
update_book(isbn: "123", total_copies=5)
assert books["123"]["total_copies"] == 5, "Book not updated."

# Test deleting
delete_book("123")
assert "123" not in books, "Book not deleted."

print("✅ All tests passed successfully!")
```

UML Diagram



Design Rationale

Introduction

In designing the Mini Library System, the choice of data structures—dictionaries, lists, and tuples—was based on the need for efficient data access, storage, and manipulation.

Use of Dictionary

Dictionaries were used to store books and user information because they provide **fast access by keys**.

- Each book is uniquely identified by its ISBN, making a dictionary ideal.

- Each user has a unique ID, allowing quick lookup and updates.

Use of List

Lists were used for collections where order matters or duplicates are allowed:

- borrowed_books for a user is a list because a user may borrow multiple books.
- Lists allow easy iteration to display borrowed books or check membership.

Use of Tuple

Tuples were used to store **immutable data**, such as author information or book genres:

- Book details that do not change frequently can be stored as tuples for **safety and efficiency**.

Conclusion

The Library Management System project demonstrates how Python's core data structures **dictionaries, lists, and tuples**—can be effectively used to design a simple yet functional management tool. The system successfully performs key operations such as adding, updating, deleting, borrowing, and returning books while maintaining accurate records of members and inventory.

Through the implementation of modular functions, UML design, and unit testing, the project promotes **clarity, maintainability, and reusability** of code. The use of test scripts ensures that each component functions correctly, while the demo file provides a practical example of real-world operation.

Overall, this project highlights the importance of **structured programming, data integrity, and user interaction** in software development. It provides a foundational understanding of how real library systems operate and sets the stage for further enhancement—such as database integration, GUI design, or online access management.