Design Rationale – Library Management System

The main goal of this project was to build a Library Management System using only basic Python features, without any databases or frameworks. To make it simple but still realistic, I used three core data structures: a dictionary for books, a list of dictionaries for members, and a tuple for genres.

Why I Choose These Data Structures

Books - Dictionary

Each book has a unique ISBN number, so using a dictionary made the most sense. It allows me to access a book instantly just by using its ISBN as a key. This makes searching, updating, or borrowing a book very fast compared to scanning through a list. It also automatically prevents duplicate entries.

Members - List of Dictionaries

For members, I used a list, where each member is stored as a dictionary. I chose a list because the number of members is relatively small, and looping through them is simple. Storing each member as a dictionary makes it easy to attach details like name, email, and borrowed books.

This structure is also future proof, because if I later want to add phone numbers or fines, I can just add more keys to each member dictionary.

Genres - Tuple

Genres don't change often, so I stored them inside a tuple to make them constant. Using a tuple instead of a list prevents accidental edits and acts like a small "database" of allowed categories when adding a book.

Function Design

All features such as adding books, borrowing, and returning are separate functions. I did this to keep things organized and easy to test. Every function handles only one job, which follows the Single Responsibility Principle meaning it's easier to fix or improve things later.

Example:

- add_book() only adds books.
- borrow_book() only deals with borrowing.
- return_book() handles only returning.

Testing

To make sure everything works correctly, I used assert statements in tests.py. Instead of manually checking outputs, the program itself verifies whether things like duplicate books or failed borrow attempts are correctly handled. This makes the system more reliable and less dependent on guesswork.

Finally

Overall, the combination of dictionary + list + tuple gave me a balance of speed, simplicity, and control. Even though this is a basic system, the structure is solid enough that it could later be upgraded into a database or even a website. The modular function design also means that I can plug a GUI or API on top of it in the future without rewriting everything.

```python
# operations.py
# Library Management System Functions
# Author: Osman Sheriff

# ---------- Data Structures ----------
books = {}
members = []
genres = ("Fiction", "Non-Fiction", "Sci-Fi", "Education")

# ---------- CRUD + Borrow/Return ----------

def add_book(books, isbn, title, author, genre, total_copies, genres):
    """Add a new book if ISBN is unique and genre is valid."""
    if isbn in books:
        return "Book already exists."
    if genre not in genres:
        return "Invalid genre."
    books[isbn] = {
        "title": title,
        "author": author,
        "genre": genre,
        "total_copies": total_copies,
        "borrowed": 0
    }
    return "Book added successfully."


def search_book(books, title):
    """Search a book by title."""
    for isbn, info in books.items():
        if info["title"].lower() == title.lower():
            return isbn, info
    return None


def update_book(books, isbn, title=None, author=None, total_copies=None):
    """Update book details."""
    if isbn not in books:
        return "Book not found."
    if title:
        books[isbn]["title"] = title
    if author:
        books[isbn]["author"] = author
    if total_copies:
        books[isbn]["total_copies"] = total_copies
    return "Book updated successfully."


def delete_book(books, isbn):
    """Delete a book if it exists."""
    if isbn in books:
        del books[isbn]
        return "Book deleted successfully."
    return "Book not found."


def add_member(members, member_id, name, email):
```

```
    """Add a new library member."""
    for m in members:
        if m["member_id"] == member_id:
            return "Member already exists."
    members.append({"member_id": member_id, "name": name, "email": email,
"borrowed_books": []})
    return "Member added successfully."


def delete_member(members, member_id):
    """Delete a member only if no borrowed items."""
    for m in members:
        if m["member_id"] == member_id:
            if m["borrowed_books"]:
                return "Member still has borrowed books."
            members.remove(m)
            return "Member deleted successfully."
    return "Member not found."


def borrow_book(books, members, member_id, isbn):
    """Allow a member to borrow a book if available."""
    for m in members:
        if m["member_id"] == member_id:
            if len(m["borrowed_books"]) >= 3:
                return "Member has already borrowed 3 books."
            if isbn not in books:
                return "Book not found."
            book = books[isbn]
            if book["borrowed"] >= book["total_copies"]:
                return "No copies left."
            book["borrowed"] += 1
            m["borrowed_books"].append(isbn)
            return "Book borrowed successfully."
    return "Member not found."


def return_book(books, members, member_id, isbn):
    """Allow a member to return a borrowed book."""
    for m in members:
        if m["member_id"] == member_id:
            if isbn not in m["borrowed_books"]:
                return "This book was not borrowed."
            m["borrowed_books"].remove(isbn)
            books[isbn]["borrowed"] -= 1
            return "Book returned successfully."
    return "Member not found."
```

```python
# demo.py
from operations import *

print("=== DEMO: Library Management System ===\n")

print(add_book(books, "001", "Python Basics", "Osman sheriff", "Fiction", 3,
genres))
print(add_book(books, "002", "Networking 101", "Alpha Sheriff", "Education",
2, genres))

print(add_member(members, 1, "Selwyn", "selwyn@gmail.com"))
print(add_member(members, 2, "Solomon", "solomon@gmail.com"))

print(borrow_book(books, members, 1, "001"))
print(borrow_book(books, members, 1, "002"))
print(return_book(books, members, 1, "001"))

print(delete_book(books, "002"))
print(delete_member(members, 2))
```

```python
# tests.py
from operations import *

def run_tests():
    # Reset data
    test_books = {}
    test_members = []
    test_genres = ("Fiction", "Non-Fiction")

    # 1. Add book
    result = add_book(test_books, "111", "Python 101", "Osman", "Fiction", 3,
test_genres)
    assert result == "Book added successfully."

    # 2. Add duplicate book
    result = add_book(test_books, "111", "Python 101", "Osman", "Fiction", 3,
test_genres)
    assert result == "Book already exists."

    # 3. Add member
    result = add_member(test_members, 1, "Selwyn", "selwyn@mail.com")
    assert result == "Member added successfully."

    # 4. Borrow book
    result = borrow_book(test_books, test_members, 1, "111")
    assert result == "Book borrowed successfully."

    # 5. Borrow when no copies left
    test_books["111"]["borrowed"] = 3
    result = borrow_book(test_books, test_members, 1, "111")
    assert result == "No copies left."

    print("All tests passed successfully!")

if __name__ == "__main__":
    run_tests()
```