Project Report: Simple Library Management System (LMS)

Attribute	**Detail**
Project Title	Simple Library Management System (LMS)
Primary Goal	To implement a functional, menu-driven command-line tool for tracking library inventory and book status using a persistent SQLite database.
Technical Stack	Python 3.x, SQLite 3
Date	November 2025

1. Introduction and Objectives

The Simple Library Management System is a lightweight application designed to manage basic library operations. The core objective was to demonstrate proficiency in:

1. Establishing persistent data storage using the Python sqlite3 module.
2. Implementing fundamental database operations (CRUD).
3. Enforcing business logic, specifically around the availability and status of books.
4. Providing a clean, interactive command-line interface (CLI) for users.

2. System Architecture

The application is designed with a modular, layered structure to ensure separation of concerns between data handling, business rules, and the user interface. This structure makes the code easier to maintain, test, and debug.

Shutterstock
The system is logically divided into three main modules:

2.1 Database Manager (Data Access Layer)

This module contains low-level functions (add_book, get_all_books, update_book_status, delete_book) responsible for direct communication with the library.db file. It handles connecting to the database, executing SQL queries, committing transactions, and managing database-specific errors (e.g., sqlite3.IntegrityError).

2.2 Library Logic (Business Layer)

This module contains the system's core rules and decision-making logic (check_out_book, return_book). It acts as an intermediary, calling the Data Access Layer functions after validating the request against current book status.

2.3 User Interface and Workflow

This module handles all user interaction, including displaying the main menu, accepting input, validating non-database input (like ensuring Book IDs are integers), and formatting the output (list_all_books). It drives the main application loop (run_system).

3. Database Design

The system uses a single table, books, within the library.db SQLite file.

Table Schema: books

Column Name	**Data Type**	**Constraint**	**Description**
id	INTEGER	PRIMARY KEY	Unique identifier for the book record.
title	TEXT	NOT NULL	The book's title.
author	TEXT	NOT NULL	The book's author.
isbn	TEXT	UNIQUE, NOT NULL	International Standard Book Number, enforced as unique to prevent duplicate records.
status	TEXT	NOT NULL	Current loan status (Available or Checked Out).

The schema design prioritizes data integrity by using NOT NULL constraints on required fields and a UNIQUE constraint on the isbn field.

4. Key Implementation Details

4.1 CRUD Operations

- **Create (add_book):** Safely inserts a new book. Crucially, it uses a try...except sqlite3.IntegrityError block to inform the user if the provided ISBN already exists, preventing database corruption and duplicate entries.
- **Read (get_all_books):** Fetches all records for inventory display.
- **Update (update_book_status):** A specialized update function focused solely on changing the status field, which is essential for the library logic.
- **Delete (delete_book):** Removes a book based on its primary key (id).

4.2 Business Logic

- **check_out_book(book_id):** Before executing the update, this function performs a read operation to check the book's status. If the status is *not* Available, the checkout is blocked, and an informative message is returned, upholding library rules.
- **return_book(book_id):** Similarly, this function ensures the book is currently Checked Out before marking it Available, preventing logical errors like trying to return a book that is already on the shelf.

4.3 Initialization and Persistence

The script initializes the database and table structure immediately upon execution using create_table(). The insert_sample_books() function ensures a user has pre-populated data to work with, enhancing the user experience on the first run, without overwriting data on subsequent runs.

5. Conclusion and Future Recommendations

The current LMS successfully meets all requirements for a basic book management system. It showcases robust use of SQLite, clean separation of code modules, and appropriate error handling.

Recommendations for Future Enhancement

1. **User/Borrower Tracking:** Extend the database to include a borrowers table and link it to the books table. When checking out a book, the system should record *who* borrowed the book and *when*.
2. **Search Functionality:** Implement a function to search books by title, author, or partial ISBN, rather than relying solely on viewing the entire list.
3. **Input Validation:** Implement stricter validation for the ISBN format (e.g., using a regex pattern) to ensure data quality before attempting database insertion.

Comprehensive Project Report: Simple Library Management System (LMS)

Attribute	Detail
Project Title	Simple Library Management System (LMS)

Primary Goal	To implement a functional, menu-driven command-line tool for tracking library inventory and book status using a persistent SQLite database.
Technical Stack	Python 3.x, SQLite 3
Date	November 2025

1. Abstract

The Simple Library Management System (LMS) is a proof-of-concept application built in Python, utilizing the standard `sqlite3` library for durable, file-based data storage. The application provides a command-line interface (CLI) that enables users to manage books through core library functions: adding new inventory, checking out, returning, viewing, and deleting records. The system is designed with a clear focus on data integrity, enforcing the uniqueness of book ISBNs and controlling book status changes based on predefined business rules.

2. System Architecture and Design

The LMS follows a three-tiered modular design approach to ensure maintainability and separation of concerns. This structure isolates the data handling from the application logic and the user interface.

2.1 Layered Structure

2.1.1 Data Access Layer (Database Manager)

This is the lowest layer, directly interacting with the `library.db` file. Functions in this layer, such as `add_book` and `update_book_status`, are responsible only for executing SQL commands. They handle the connection, cursor management, transaction commitment, and reporting database-specific exceptions (e.g., `sqlite3.IntegrityError`).

2.1.2 Business Logic Layer (Library Logic)

This middle layer defines the rules of the library. Functions like `check_out_book` and `return_book` manage the flow of data and enforce rules (e.g., "A book can only be checked out if its status is 'Available'"). It calls the Data Access Layer to modify the database only after validating the request.

2.1.3 Presentation Layer (User Interface)

The top layer, managed by functions like `run_system` and `display_menu`, handles all interaction with the user. It is responsible for accepting input, presenting formatted output (e.g., the tabulated list of books), and navigating the menu-driven workflow.

2.2 Data Flow Diagram

The primary flow for a status change (e.g., checking out a book) is sequential:

1. **User Input:** The `run_system` loop receives a menu choice (e.g., '3' for Checkout).
2. **Input Handling:** `handle_checkout_return` validates the input as an integer ID.
3. **Business Logic:** `check_out_book` queries the database for the book's status.
4. **Decision:** If the status is 'Available', it calls the Data Access Layer.
5. **Data Access:** `update_book_status` executes the `UPDATE` SQL command.
6. **Confirmation:** A success or failure message is returned through the layers to the user.

3. Database Design

The system uses a persistent SQLite database named `library.db`. The core inventory data is stored in the single `books`table.

3.1 Table Schema: `books`

Column Name	Data Type	Constraint	Description
`id`	`INTEGER`	`PRIMARY KEY`	Unique identifier for the book record.
`title`	`TEXT`	`NOT NULL`	The book's title.
`author`	`TEXT`	`NOT NULL`	The book's author.
`isbn`	`TEXT`	`UNIQUE`, `NOT NULL`	International Standard Book Number, enforced as unique to prevent duplicate records.
`status`	`TEXT`	`NOT NULL`	Current loan status (`Available` or `Checked Out`).

The use of `UNIQUE` on the ISBN is critical for data quality, ensuring every physical or digital title is registered only once.

4. Testing and Validation Plan

A crucial part of any system development is testing. Since this is a CLI application, a manual testing plan is appropriate.

4.1 Test Cases for Critical Functions

Test Case ID	Feature	Input/Action	Expected Result	Pass/Fail
TC-1.1	Add Book (Success)	Title: "Galaxy", Author: "K. Lee", ISBN: "1112223334445"	Success message; Book appears in View All (2) with 'Available' status.	
TC-1.2	Add Book (Fail - Duplicate ISBN)	Title: "Book B", Author: "X", **ISBN: "9780451524935"** (from sample data)	Error message: "Book could not be added. ISBN might already exist."	
TC-2.1	Checkout (Success)	Choose option 3; Enter an ID with status **'Available'**.	Status changes to 'Checked Out'; success message shown.	
TC-2.2	Checkout (Fail - Unavailable)	Choose option 3; Enter the ID from TC-2.1 again.	Blocked message: "Book '...' is already **Checked Out**."	
TC-3.1	Return (Success)	Choose option 4; Enter the ID from TC-2.1 (which is 'Checked Out').	Status changes to 'Available'; success message shown.	
TC-3.2	Return (Fail - Invalid ID)	Choose option 4; Enter a non-existent ID (e.g., `999`).	Error message: "Book ID not found."	

TC-4.1	Delete (Success)	Choose option 5; Enter the ID created in TC-1.1.	Success message; Book is removed from View All (2) list.

5. Conclusion and Future Recommendations

The Library Management System is a robust and functional command-line tool that successfully demonstrates object-oriented principles, database interaction, and exception handling in Python.

Recommendations for Future Enhancement

1. **User/Borrower Tracking: HIGH PRIORITY.** Implement a separate table for `borrowers` and link it to the `books`table. This would require updating the `check_out_book` function to record the borrower's ID and the date of checkout.
2. **Search Functionality:** Enhance the `list_all_books` function with search parameters to allow users to filter the inventory by partial Title, Author, or ISBN.
3. **Input Validation:** Implement stricter front-end validation (e.g., checking if ISBN is exactly 13 digits) before attempting database operations, reducing unnecessary database load and providing clearer user feedback.
4. **Date Tracking:** Add a `checkout_date` and a calculated `due_date` field to introduce late return logic.

Appendix A: Core SQL and Python Mapping

This section illustrates how Python functions map to the underlying SQL commands.