

Case Study

ON

Library Management System

*NAME: Ankit*

*SECTION: 3(A)*

*UID: 24BCA10448*

*SUBJECT: Object Oriented Programming Language*

*SUBMITTED TO: Miss. Jyoti Rani*

**1. Introduction**

A **Library Management System (LMS)** is a software application designed to manage the daily operations of a library efficiently. It automates tasks such as book cataloging, member registration, issue and return tracking, and fine calculation.  
The goal is to provide a **centralized, digital system** that reduces manual work, prevents data loss, and improves user experience for both **librarians** and **members**.

**2. Background and Problem Statement**

**Traditional libraries often rely on manual record-keeping, using registers or spreadsheets to track:**

* Book inventories
* Member information
* Borrowing and return details
* Fine calculations

**This manual system leads to:**

* Human errors (e.g., wrong entries, duplicate records)
* Difficulty in searching books
* Time-consuming issue/return process
* Poor tracking of overdue books and fines

**3. Objectives**

The main objectives of the LMS are:

1. To automate library operations like book issuance, return, and cataloging.
2. To provide quick access to books using search and filter functionalities.
3. To maintain accurate records of members, books, and transactions.
4. To generate reports (e.g., books issued, overdue books, fine collection).
5. To allow both **admin** (librarians) and **users** (students/faculty) to interact through a user-friendly interface.

**4. Stakeholders**

| **Stakeholder** | **Role** | **Description** |
| --- | --- | --- |
| **Librarian** | Admin | Manages all book records, member details, and transactions. |
| **Student/Faculty** | User | Searches, borrows, and returns books; checks availability and due dates. |
| **System Administrator** | IT Support | Ensures smooth operation, backups, and security updates. |

1. **System Requirements**

**Functional Requirements**

* Book Management: Add, update, delete, and search books by title, author, or ISBN.
* Member Management: Register new members and update personal details.
* Transaction Management: Handle issue and return of books; track due dates and fines.
* Fine Calculation: Automatically calculate fines for overdue books.
* Reports Generation: Create reports on library usage, inventory, and fines collected.

**Non-Functional Requirements**

* Usability: Simple and intuitive interface for all users.
* Reliability: System should handle multiple users concurrently without data loss.
* Security: Role-based access (admin vs. user); password protection.
* Scalability: Ability to handle increasing number of users and books.
* Performance: Quick search and response time under heavy loads.

**6. System Design**

**Architecture Overview**

**A three-tier architecture is used:**

1. Presentation Layer: User Interface (web or desktop application)
2. Application Layer: Business logic (issue/return, fine calculation, reports)
3. Database Layer: Stores data in relational databases like MySQL or PostgreSQL

Entity-Relationship (ER) Diagram

**Entities:**

* Book (Book\_ID, Title, Author, ISBN, Category, Availability)
* Member (Member\_ID, Name, Email, Contact\_No, Membership\_Date)
* Transaction (Transaction\_ID, Book\_ID, Member\_ID, Issue\_Date, Return\_Date, Fine)

**Relationships:**

* A Member can issue multiple Books.
* A Book can be issued to one Member at a time.
* Each Transaction records a unique issue-return event.

**7. Use Case Diagram**

**Actors:** Librarian, Member  
**Main Use Cases:**

* Librarian: Manage books, manage members, issue/return books, view reports
* Member: Search books, check availability, view issued books, renew or return books

**8. Implementation Highlights**

* **Search Optimization:** Full-text search and filters (author, category).
* **Fine Calculation Logic:** Fine = (Return Date – Due Date) × ₹5 per day.
* **Email Notifications:** Automated reminders for due or overdue books.
* **Dashboard:** Shows total books, members, books issued, and pending returns.

**9. Testing**

**Testing Methods:**

* Unit Testing: Test individual modules (book search, fine calculation).
* Integration Testing: Verify that modules work together correctly.
* User Acceptance Testing (UAT): Validate that the system meets user needs.

**10. Results**

After implementation, the LMS achieved:

* **70% reduction** in time spent on issuing and returning books.
* **Zero data loss** due to digital record storage.
* **Higher accuracy** in fine calculations and report generation.
* **Improved accessibility** for users through online book search.

**11. Conclusion**

The **Library Management System** successfully automates the traditional library processes, offering reliability, efficiency, and scalability. It not only improves the librarian’s workflow but also enhances the overall user experience for students and faculty.