**Library Management System Database Design**

Here are the **functional** and **non-functional** requirements for a **Library Management System**:

**Functional Requirements**

1. **User Registration & Authentication**

* Register new members and staff
* Provide login/logout functionality for staff and members

1. **Book Management**

* Add, update, delete, and search books
* Manage book details including title, author, publisher, category, and available copies
* Categorize books by genre, subject, and other classification schemes Track book condition and availability status

1. **Member Management**

* Add, update, and delete member records
* View borrowing history of members
* Manage member account status (active, suspended, expired)

1. **Loan Management**

* Issue books to members
* Process book returns and update return dates
* Calculate overdue fines when applicable
* Set and enforce due dates for returns

1. **Staff Management**

* Add and manage staff users with different roles (e.g., admin, librarian)
* Track book acquisitions and removals
* Support periodic inventory checks
* Record book conditions and damage reports
* Manage book locations within the library

1. **Search Functionality**

* Search books by title, author, ISBN, or category
* Filter members or loans based on various criteria (e.g., overdue books)

1. **Notifications**

* Send due date reminders to members
* Alert staff about overdue books
* Notify members when reserved books become available
* Generate notifications about fine payments

1. **Reports**

Generate reports for:

* Issued books
* Overdue books
* Books by category
* Member activity

**Non-Functional Requirements**

1. **Performance**

* Support concurrent users (multiple librarians and members accessing the system simultaneously)
* Provide fast response time for search and report generation

1. **Scalability**

* Handle increasing numbers of books, members, and transactions without performance degradation

1. **Security**

* Implement secure login with hashed passwords
* Enforce role-based access control (only authorized staff can issue/return books)
* Prevent SQL injection and XSS attacks

1. **Availability**

* Ensure system availability during working hours or 24/7 (if online)

1. **Maintainability**

* Follow clean architecture principles to make the system easy to update or extend

1. **Usability**

* Provide a simple and user-friendly interface for both staff and members

1. **Backup and Recovery**

* Implement regular data backup procedures Support ability to recover data in case of system failure

1. **Data Integrity**

* Prevent duplicate records
* Ensure all references (e.g., foreign keys) remain valid

1. **ERD/Schema Design Overview**

**Entities:**

* **Books** (BookID, Title, ISBN, AuthorID, PublisherID, CategoryID, CopiesAvailable, Condition, Location)
* **Authors** (AuthorID, Name, Bio)
* **Publishers** (PublisherID, Name, ContactInfo)
* **Categories** (CategoryID, CategoryName, Description)
* **Members** (MemberID, FullName, Email, Address, Phone, Status [active/suspended/expired], JoinDate, PasswordHash)
* **Staff** (StaffID, FullName, Email, Role [admin/librarian], PasswordHash, JoinDate, Phone)
* **Loans** (LoanID, BookID, MemberID, IssueDate, DueDate, ReturnDate, FineAmount)
* **Reservations** (ReservationID, BookID, MemberID, ReservationDate, Status [pending/notified/cancelled])
* **Notifications** (NotificationID, MemberID, Type [due\_reminder/fine/reservation], Message, DateSent, IsRead)
* **BookInventoryLogs** (LogID, BookID, StaffID, ActionType [added/removed/damaged/relocated], ActionDate, Notes)

**Relationships:**

* **Books** → **Authors**: Many-to-One
* **Books** → **Publishers**: Many-to-One
* **Books** → **Categories**: Many-to-One
* **Loans** → **Books**: Many-to-One
* **Loans** → **Members**: Many-to-One
* **Reservations** → **Books**: Many-to-One
* **Reservations** → **Members**: Many-to-One
* **Notifications** → **Members**: Many-to-One
* **BookInventoryLogs** → **Books**: Many-to-One
* **BookInventoryLogs** → **Staff**: Many-to-One
* **Loans** → **Staff** : Many-to-One