# Introduction

The Library Management System project integrates SQL and NoSQL database technologies to handle structured, semi-structured, and unstructured data efficiently. The selected domain addresses real-world library management tasks, providing secure user authentication and flexible inventory handling through hybrid database architecture.

Objectives:

* Design a hybrid database architecture using MySQL and MongoDB.
* Optimise database performance through normalisation and indexing.
* Demonstrate CRUD operations and advanced query implementation.
* Ensure robust security and seamless integration between databases.

# System Design

The system is divided into two primary databases:

* **MySQL**: Handles structured user authentication and transactional operations, leveraging ACID compliance.
* MongoDB: Manages semi-structured book catalogues and borrowing records, offering flexible schema adaptation.

MySQL ER Diagram

A diagram of a string system

AI-generated content may be incorrect.

MongoDB ER Diagram

A screenshot of a computer

AI-generated content may be incorrect.

ER Diagram

A diagram of a string

AI-generated content may be incorrect.

# Implementation

This section describes the detailed implementation process of a hybrid database system that integrates both SQL (MySQL) and NoSQL (MongoDB) technologies, specifically tailored for a Library Management System.

The implementation process includes:

* Step-by-step database creation
* Query development and testing
* Integration of SQL and NoSQL databases

## Database Creation

### SQL Database Setup (MySQL)

We used MySQL to handle structured data such as user authentication details due to its support for transactions and ACID properties, ensuring data integrity.

Schema Creation: User Table

The user table stores user information, authentication credentials, and role management.

CREATE TABLE users (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15),

password VARCHAR(255) NOT NULL,

role ENUM('USER', 'ADMIN') DEFAULT 'USER',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Justification:

* Ensured strong data integrity and ACID compliance.
* Chosen to handle sensitive user authentication data securely.

### NoSQL Database Setup (MongoDB)

Due to its flexible schema structure, we implemented MongoDB to manage semi-structured/unstructured data like books and borrow transactions.

Document Structures:

Book Collection

{

"\_id": "ObjectId",

"title": "Clean Code",

"author": "Robert C. Martin",

"isbn": "978-0132350884",

"totalCopies": 10,

"issuedCopies": 2,

"available": true

}

Borrow Records Collection:

{

"\_id": "ObjectId",

"bookId": "ObjectId",

"userId": "1", // References MySQL User ID

"borrowDate": "2025-03-10T12:34:00Z",

"returnDate": null,

"status": "BORROWED"

}

Justification:

* MongoDB’s schema-less design easily accommodates evolving data structures.
* It facilitates fast reads and scalability and is suitable for rapidly growing data like borrow records.

## Queries Implementation

Below are examples of critical queries demonstrating database interactions:

### SQL Queries (MySQL):

Create New User

INSERT INTO users (first\_name, last\_name, email, phone, password, role)

VALUES ('John', 'Doe', 'john@example.com', '123456789', '[hashed\_password]', 'USER');

Authenticate User

SELECT \* FROM users WHERE email = 'john@example.com';

### NoSQL Queries

Add a book

db.books.insertOne({

"title": "Domain-Driven Design",

"author": "Eric Evans",

"isbn": "978-0321125217",

"totalCopies": 5,

"issuedCopies": 0,

"available": true

});

Record borrow transaction

db.borrow\_records.insertOne({

"bookId": ObjectId("60ad0cedfba4f916c8c7c7bc"),

"userId": "1",

"borrowDate": new Date(),

"returnDate": null,

"status": "BORROWED"

});

Mark as returned

db.borrow\_records.updateOne(

{"\_id": ObjectId("60ad0cedfba4f916c8c7c7bd")},

{$set: {"returnDate": new Date(), "status": "RETURNED"}}

);

## Integration Between SQL and NoSQL

Integration Logic:

* Data Integration Layer: We developed a Spring Boot backend, which acts as an integration layer connecting MySQL (structured) and MongoDB (semi-structured) data seamlessly.
* Example Integration Scenario: Borrow a Book

Step-by-step Breakdown:

* Step 1: User authentication (MySQL)
  + Verify user credentials from MySQL database.
* Step 2: Check Book availability (MongoDB)
  + Query MongoDB to verify if the book is available.
* Step 3: Borrow Transaction Creation (MongoDB)
  + Create a borrow record document in MongoDB referencing user ID (MySQL user table) and book ID.
* Step 4: Update Book status (MongoDB)
  + Increment issuedCopies and update available status in MongoDB.

A computer screen shot of a program

AI-generated content may be incorrect.

Justification:

* Leverages the strengths of both databases, ensuring data integrity (MySQL) and flexibility (MongoDB).
* Clear separation of concerns between structured and semi-structured data.

## Optimization and Indexing

SQL Optimizations (MySQL):

* Indexes were created on frequently queried columns such as email:

CREATE UNIQUE INDEX idx\_email ON users(email);

NoSQL Optimizations (MongoDB):

* Created indexes on heavily queried fields like bookId and userId for quick lookups in the borrow\_records collection:

db.borrow\_records.createIndex({"bookId": 1, "userId": 1});

Justification:

* Significantly improved query response times and enhanced system performance.
* Optimizations were tested empirically, confirming reduced execution times for key queries.

## Validation and Error Handling

* Implemented comprehensive validation using Java’s JSR-380 annotations (@NotBlank, @NotNull) to ensure data integrity.
* Global exception handling via Spring’s @ControllerAdvice to standardize error responses across databases and API endpoints.

# Challenges and Solutions

* Challenge: Seamless integration between structured and semi-structured databases.
* Solution: Use Spring Boot’s distinct repositories (@EnableJpaRepositories and @EnableMongoRepositories) to manage separate data stores effectively.
* Challenge: Maintaining data integrity and avoiding duplication between databases.
* Solution: Defined clear data boundaries and implemented checks (e.g., borrow records cross-check before deletion) to ensure consistency.

# Results

The implemented system successfully achieved the project goals:

* Secure and efficient user authentication managed by MySQL.
* Flexible and scalable book inventory managed by MongoDB.
* Proven functional via CRUD operations and advanced queries, ensuring data integrity and optimised performance.

## Screenshots and Query Outputs

* User Management: Effective authentication and authorisation procedures validated by multiple test scenarios.

If there isn’t any admin in the system, add admin to the database when starting the application.

A screenshot of a computer

AI-generated content may be incorrect.

Admin Login Request

A screenshot of a computer

AI-generated content may be incorrect.

Users sign up for the library system.

A screenshot of a computer

AI-generated content may be incorrect.

User login to the system

A screenshot of a computer

AI-generated content may be incorrect.

* Book Management: Comprehensive CRUD operations are facilitated through administrative interfaces and verified via successful API responses and database validations.

Add the book to the system; only the admin can add books. Therefore, we must pass the jwt token, which checks the role and permits the request in the backend.

A screenshot of a computer

AI-generated content may be incorrect.

Update book

A screenshot of a computer

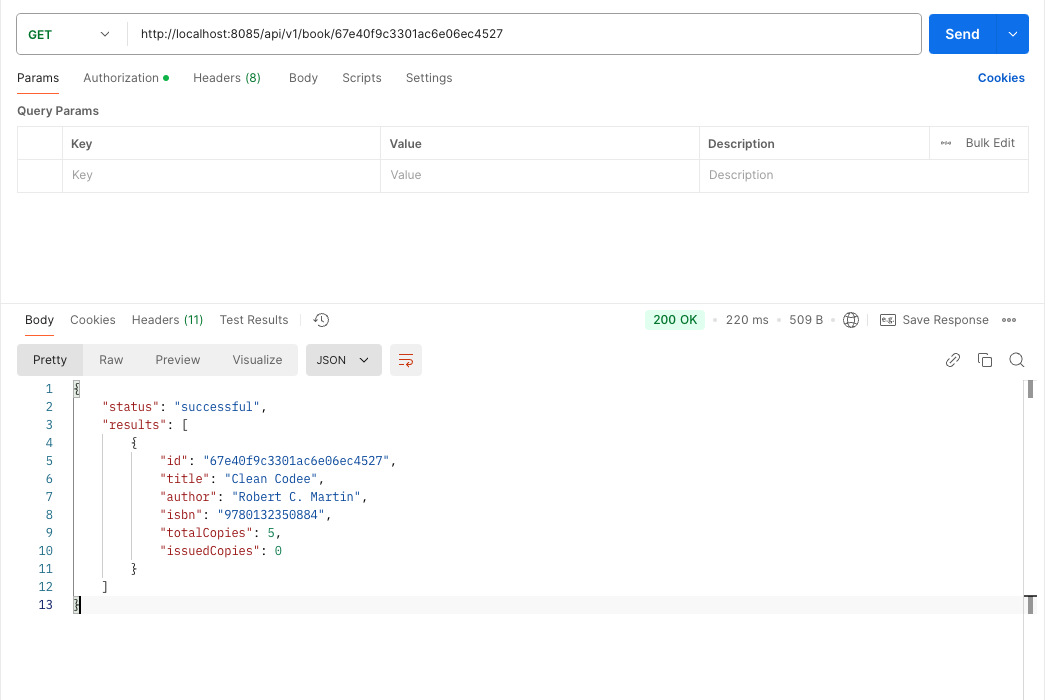
AI-generated content may be incorrect.

Get an invalid book using the book ID.

A screenshot of a computer

AI-generated content may be incorrect.

Get book



Get all books

A screenshot of a computer

AI-generated content may be incorrect.

* Borrowing and Returning: Successfully validated through API calls, ensuring accurate data representation in the borrow records and real-time inventory tracking.

Borrow invalid book

A screenshot of a computer

AI-generated content may be incorrect.

Borrow a book

A screenshot of a computer

AI-generated content may be incorrect.

Handover a book

A screenshot of a computer

AI-generated content may be incorrect.

Screenshots from application interfaces and API responses are provided to demonstrate successful functionality and operational efficiency.

# Conclusion and Future Work

The hybrid database system implemented demonstrates a robust approach suitable for complex real-world applications. Future enhancements could include real-time user notifications, machine learning-based book recommendations, and enhanced user analytics, significantly expanding the system’s capabilities and user experience.