UNIVERSITY INSTITUTE OF COMPUTING

CASE STUDY REPORT
ON
PARTICULAR CASE STUDY

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

Submitted by:

Name: Rajan Pal

UID: 23BCA10344

Section: 23BCA-4-B

Submitted to:

Name: Arvinder Singh

Designation: Assistant Professor

ABSTRACT

- INTODUNCTION:
- TECHNIWUE:
- SYSTEM CONFIGURATION:
- INPUT:
- ER DIAGRAM:
- TABLE REALTION:
- TABULAR FORMAT:
- TABLE CREATION:
- SQL QUERIES WITH OUTPUT (at least 10 to 15):
- SUMMARY:
- CONCLUSION:

Library Management System - Case Study

---

1. INTRODUCTION

Libraries have evolved from physical repositories of books to comprehensive digital and physical hubs for information and knowledge sharing. The modern-day library system not only manages books but also handles user registrations, borrowing and return systems, penalties, and availability statuses. This case study presents a Library Management System (LMS) built using MySQL, focusing on streamlining the operations involved in managing books, authors, members, loans, and fines.

2. TECHNIQUE

The LMS leverages the Relational Database Management System (RDBMS) approach. Data normalization, referential integrity, and indexing techniques have been used to create an efficient and scalable database. MySQL was chosen for its robustness and open-source nature. SQL (Structured Query Language) is used extensively for data manipulation and retrieval.

3. SYSTEM CONFIGURATION

- Operating System: Windows 10 / Ubuntu 20.04+
- RDBMS: MySQL 8.0+
- Tools: MySQL Workbench / phpMyAdmin
- RAM: Minimum 4 GB

- Processor: Intel i3 or higher

4. INPUT

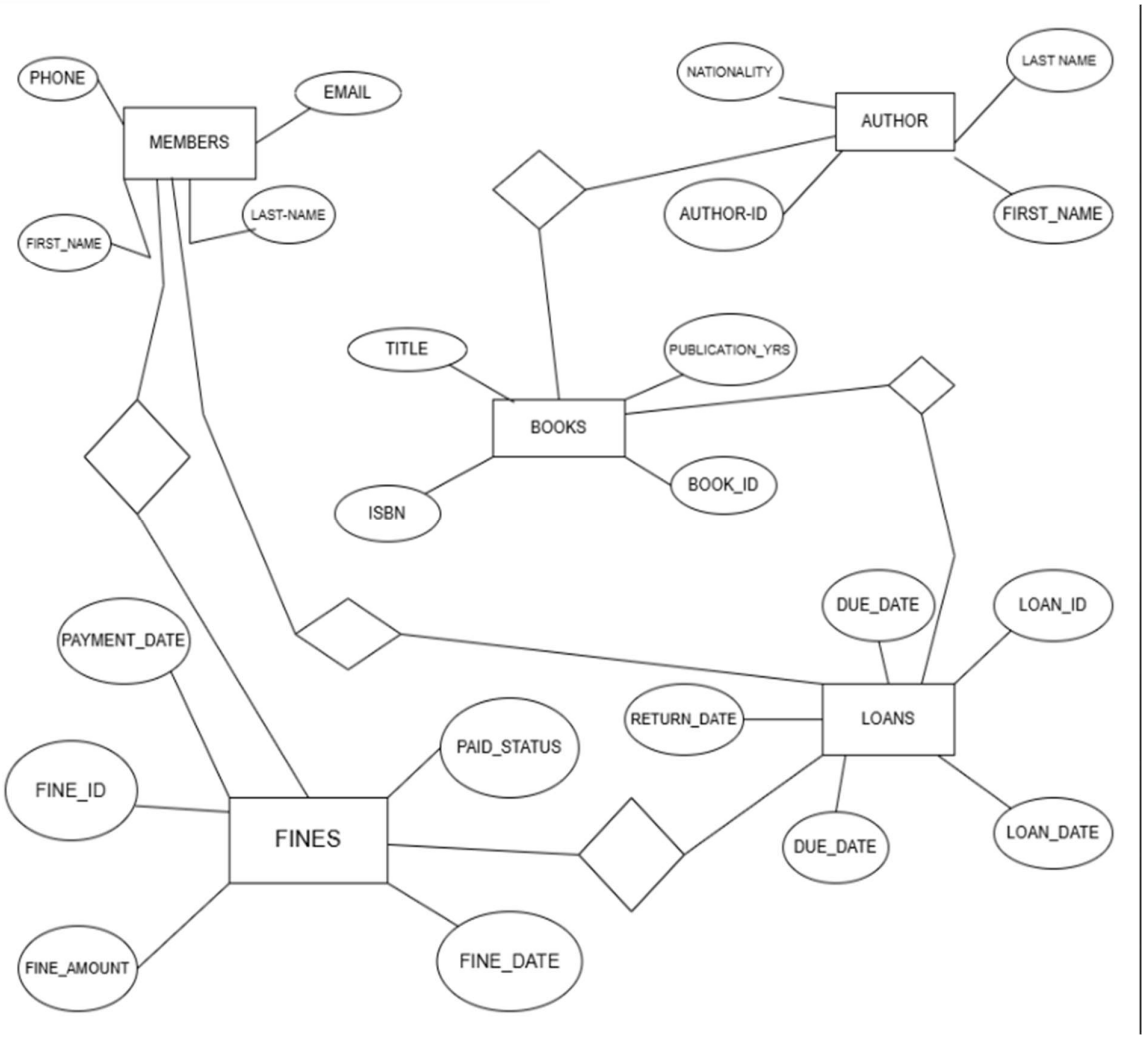
Inputs to the LMS include:

- Member data (name, contact details, membership type)
- Author information (name, nationality, birth/death years)
- Book metadata (title, author, publication year, genre, etc.)
- Loan transactions (book issued, due date, return date)
- Fine data (fine amount, status, date paid)

5. ER DIAGRAM DESCRIPTION

Entities:

- **Members**: Tracks library users
- **Authors**: Stores author information
- **Books**: Represents available books
- **Loans**: Keeps record of borrowings
- **Fines**: Maintains penalty details



Relationships:

- One **Author** can write many **Books** (1:M)
- One **Member** can have multiple **Loans** (1:M)
- One **Loan** can generate a **Fine** (1:1)
- One **Book** can be loaned multiple times (1:M)

6. TABLE RELATIONSHIPS

- **Members** (member_id) → **Loans** (member_id)
- **Authors** (author_id) → **Books** (author_id)
- **Books** (book_id) → **Loans** (book_id)
- **Loans** (loan_id) → **Fines** (loan_id)
- **Members** (member_id) → **Fines** (member_id)

7. TABULAR FORMAT

Table	Description
Members	Stores user details
Authors	Information about book authors
Books	Metadata and availability of books
Loans	Transaction history of issued books
Fines	Fine imposition and payment records

8. TABLE CREATION

Tables are created using CREATE TABLE statements with primary keys, foreign keys, and data integrity constraints.

(Refer to the code section for actual SQL)

9. CODE

```
-- Create database and use it

CREATE DATABASE LibraryManagement;

USE LibraryManagement;

-- 1. Members table

CREATE TABLE Members (

    member_id INT PRIMARY KEY AUTO_INCREMENT,

    first_name VARCHAR(50) NOT NULL,

    last_name VARCHAR(50) NOT NULL,

    email VARCHAR(100) UNIQUE,

    phone VARCHAR(15),

    address VARCHAR(200),

    join_date DATE NOT NULL,

    membership_type ENUM('Student', 'Adult', 'Senior', 'Child') NOT
NULL,


   active_status BOOLEAN DEFAULT TRUE

);


-- 2. Authors table
```

```
CREATE TABLE Authors (
    author_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    nationality VARCHAR(50),
    birth_year INT,
    death_year INT
);

-- 3. Books table
CREATE TABLE Books (
    book_id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    author_id INT NOT NULL,

  isbn VARCHAR(20) UNIQUE,
    publication_year INT,

    genre VARCHAR(50),
```

```
    publisher VARCHAR(100),
    total_copies INT DEFAULT 1,
    available_copies INT DEFAULT 1,
    shelf_location VARCHAR(20),
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);

-- 4. Loans table
CREATE TABLE Loans (
    loan_id INT PRIMARY KEY AUTO_INCREMENT,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    loan_date DATE NOT NULL,
    due_date DATE NOT NULL,

  return_date DATE,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
```

```
   FOREIGN KEY (member_id) REFERENCES
Members(member_id)
);


-- 5. Fines table
CREATE TABLE Fines (
   fine_id INT PRIMARY KEY AUTO_INCREMENT,
   loan_id INT NOT NULL,
   member_id INT NOT NULL,
   fine_amount DECIMAL(10,2) NOT NULL,
   fine_date DATE NOT NULL,
   paid_status BOOLEAN DEFAULT FALSE,
   payment_date DATE,
   FOREIGN KEY (loan_id) REFERENCES Loans(loan_id),


  FOREIGN KEY (member_id) REFERENCES
Members(member_id)
);
```

-- Insert sample data into Authors table

INSERT INTO Authors (first_name, last_name, nationality, birth_year, death_year) VALUES

('J.K.', 'Rowling', 'British', 1965, NULL),

('George', 'Orwell', 'British', 1903, 1950),

('Harper', 'Lee', 'American', 1926, 2016),

('J.R.R.', 'Tolkien', 'British', 1892, 1973),

('Agatha', 'Christie', 'British', 1890, 1976),

('Stephen', 'King', 'American', 1947, NULL),

('Jane', 'Austen', 'British', 1775, 1817),

('Leo', 'Tolstoy', 'Russian', 1828, 1910),

('Mark', 'Twain', 'American', 1835, 1910),

('Ernest', 'Hemingway', 'American', 1899, 1961),

('F. Scott', 'Fitzgerald', 'American', 1896, 1940),

('Virginia', 'Woolf', 'British', 1882, 1941),

('Charles', 'Dickens', 'British', 1812, 1870),

('Gabriel', 'García Márquez', 'Colombian', 1927, 2014),

('Toni', 'Morrison', 'American', 1931, 2019);

-- Insert sample data into Members table

INSERT INTO Members (first_name, last_name, email, phone, address, join_date, membership_type) VALUES

('John', 'Smith', 'john.smith@email.com', '555-0101', '123 Main St, Anytown', '2020-01-15', 'Adult'),

('Emily', 'Johnson', 'emily.j@email.com', '555-0102', '456 Oak Ave, Somewhere', '2021-03-22', 'Adult'),

('Michael', 'Williams', 'mike.w@email.com', '555-0103', '789 Pine Rd, Nowhere', '2019-11-05', 'Senior'),

('Sarah', 'Brown', 'sarah.b@email.com', '555-0104', '321 Elm St, Anywhere', '2022-02-18', 'Adult'),

('David', 'Jones', 'david.j@email.com', '555-0105', '654 Maple Dr, Everywhere', '2020-07-30', 'Adult'),

('Jennifer', 'Garcia', 'jenn.g@email.com', '555-0106', '987 Cedar Ln, Somewhere', '2021-09-14', 'Adult'),

('Robert', 'Miller', 'rob.m@email.com', '555-0107', '135 Birch Blvd, Nowhere', '2018-05-21', 'Senior'),

('Lisa', 'Davis', 'lisa.d@email.com', '555-0108', '246 Walnut Way, Anywhere', '2022-01-10', 'Adult'),

('Thomas', 'Rodriguez', 'tom.r@email.com', '555-0109', '369 Spruce St, Everywhere', '2020-08-25', 'Adult'),

('Jessica', 'Martinez', 'jess.m@email.com', '555-0110', '482 Ash Ave, Somewhere', '2021-04-17', 'Adult'),

('Daniel', 'Hernandez', 'dan.h@email.com', '555-0111', '591 Cherry Dr, Nowhere', '2019-12-03', 'Adult'),

('Amanda', 'Lopez', 'amanda.l@email.com', '555-0112', '753 Willow Ln, Anywhere', '2022-03-29', 'Student'),

('James', 'Gonzalez', 'james.g@email.com', '555-0113', '864 Poplar Rd, Everywhere', '2020-06-12', 'Adult'),

('Patricia', 'Wilson', 'pat.w@email.com', '555-0114', '975 Oakwood Ave, Somewhere', '2021-11-28', 'Senior'),

('Christopher', 'Anderson', 'chris.a@email.com', '555-0115', '159 Pinecrest Dr, Nowhere', '2018-09-15', 'Adult');

-- Insert sample data into Books table

INSERT INTO Books (title, author_id, isbn, publication_year, genre, publisher, total_copies, available_copies, shelf_location) VALUES

('Harry Potter and the Philosopher''s Stone', 1, '9780747532743', 1997, 'Fantasy', 'Bloomsbury', 5, 3, 'FIC-ROW-001'),

('1984', 2, '9780451524935', 1949, 'Dystopian', 'Secker & Warburg', 3, 1, 'FIC-ORW-002'),

('To Kill a Mockingbird', 3, '9780061120084', 1960, 'Classic', 'J. B. Lippincott', 4, 2, 'FIC-LEE-003'),

('The Hobbit', 4, '9780547928227', 1937, 'Fantasy', 'Allen & Unwin', 3, 0, 'FIC-TOL-004'),

('Murder on the Orient Express', 5, '9780007119318', 1934, 'Mystery', 'Collins Crime Club', 2, 1, 'FIC-CHR-005'),

('The Shining', 6, '9780307743657', 1977, 'Horror', 'Doubleday', 3, 2, 'FIC-KIN-006'),

('Pride and Prejudice', 7, '9780141439518', 1813, 'Romance', 'T. Egerton', 4, 3, 'FIC-AUS-007'),

('War and Peace', 8, '9781400079988', 1869, 'Historical', 'The Russian Messenger', 2, 1, 'FIC-TOL-008'),

('The Adventures of Huckleberry Finn', 9, '9780486280615', 1884, 'Adventure', 'Chatto & Windus', 3, 2, 'FIC-TWA-009'),

('The Old Man and the Sea', 10, '9780684801223', 1952, 'Literary', 'Charles Scribner''s Sons', 2, 1, 'FIC-HEM-010'),

('The Great Gatsby', 11, '9780743273565', 1925, 'Classic', 'Charles Scribner''s Sons', 3, 0, 'FIC-FIT-011'),

('Mrs Dalloway', 12, '9780156628709', 1925, 'Modernist', 'Hogarth Press', 2, 2, 'FIC-WOO-012'),

('A Tale of Two Cities', 13, '9781853260391', 1859, 'Historical', 'Chapman & Hall', 3, 1, 'FIC-DIC-013'),

('One Hundred Years of Solitude', 14, '9780060883287', 1967, 'Magical Realism', 'Harper & Row', 2, 1, 'FIC-MAR-014'),

('Beloved', 15, '9781400033416', 1987, 'Historical', 'Alfred A. Knopf', 2, 2, 'FIC-MOR-015');

-- Insert sample data into Loans table

INSERT INTO Loans (book_id, member_id, loan_date, due_date, return_date) VALUES

(1, 1, '2023-01-10', '2023-01-31', '2023-01-30'),

(1, 2, '2023-02-05', '2023-02-26', '2023-02-25'),

(2, 3, '2023-01-15', '2023-02-05', '2023-02-10'),

(4, 4, '2023-02-01', '2023-02-22', NULL),

(5, 5, '2023-01-20', '2023-02-10', '2023-02-15'),

(6, 6, '2023-02-10', '2023-03-03', NULL),

(7, 7, '2023-01-25', '2023-02-15', '2023-02-14'),

(8, 8, '2023-02-05', '2023-02-26', NULL),

(9, 9, '2023-01-30', '2023-02-20', '2023-02-18'),

(10, 10, '2023-02-08', '2023-03-01', NULL),

(11, 11, '2023-01-18', '2023-02-08', '2023-02-12'),

(12, 12, '2023-02-12', '2023-03-05', NULL),

(13, 13, '2023-01-22', '2023-02-12', '2023-02-10'),

(14, 14, '2023-02-15', '2023-03-08', NULL),

```
(15, 15, '2023-01-28', '2023-02-18', '2023-02-16');


-- Insert sample data into Fines table

INSERT INTO Fines (loan_id, member_id, fine_amount, fine_date,
paid_status, payment_date) VALUES

(3, 3, 2.50, '2023-02-11', TRUE, '2023-02-15'),


(5, 5, 2.50, '2023-02-16', FALSE, NULL),

(11, 11, 2.00, '2023-02-13', TRUE, '2023-02-15');


-- QUERIES FOR REPORTS BELOW...

-- (Optional: place queries here if you'd like them inside this script)


-- Query 1: List all active members with their membership type


SELECT member_id, first_name, last_name, membership_type

FROM Members

WHERE active_status = TRUE;
```

```
-- Query 2: List all books with fewer than 2 available copies

SELECT book_id, title, available_copies


FROM Books

WHERE available_copies < 2;


-- Query 3: Get all overdue loans (not yet returned and past due date)


SELECT Loans.loan_id, Members.first_name, Members.last_name,
Books.title, due_date

FROM Loans

JOIN Members ON Loans.member_id = Members.member_id

JOIN Books ON Loans.book_id = Books.book_id

WHERE return_date IS NULL AND due_date < CURDATE();


-- Query 4: Show all members who have unpaid fines

SELECT DISTINCT Members.member_id, first_name, last_name,
email

FROM Members
```

```
JOIN Fines ON Members.member_id = Fines.member_id

WHERE paid_status = FALSE;


-- Query 5: Find the top 5 most borrowed books

SELECT Books.title, COUNT(Loans.loan_id) AS times_borrowed

FROM Books

JOIN Loans ON Books.book_id = Loans.book_id


GROUP BY Books.book_id

ORDER BY times_borrowed DESC

LIMIT 5;


-- Query 6: Members who have borrowed more than 2 books



SELECT Members.member_id, first_name, last_name,
COUNT(Loans.loan_id) AS total_loans

FROM Members

JOIN Loans ON Members.member_id = Loans.member_id
```

```
GROUP BY Members.member_id

HAVING total_loans > 2;


-- Query 7: List all books that have never been borrowed

SELECT book_id, title

FROM Books

WHERE book_id NOT IN (

    SELECT DISTINCT book_id FROM Loans


);


-- Query 8: Find members who joined in 2023

SELECT member_id, first_name, last_name, join_date

FROM Members

WHERE YEAR(join_date) = 2023;


-- Query 9: Get a count of members by membership type

SELECT membership_type, COUNT(*) AS total_members

FROM Members
```

GROUP BY membership_type;

-- Query 10: Find total fine amount paid by each member

SELECT Members.member_id, first_name, last_name, SUM(fine_amount) AS total_paid

FROM Members

JOIN Fines ON Members.member_id = Fines.member_id

WHERE paid_status = TRUE

GROUP BY Members.member_id;

9. SQL QUERIES WITH OUTPUT

Query 1: List active members and membership type

SELECT member_id, first_name, last_name, membership_type FROM Members WHERE active_status = TRUE;

Output: List of 15 active members.

Result Grid | Filter Rows: | Edit:

member_id	first_name	last_name	membership_type
1	John	Smith	Adult
2	Emily	Johnson	Adult
3	Michael	Williams	Senior
4	Sarah	Brown	Adult
5	David	Jones	Adult
6	Jennifer	Garcia	Adult
7	Robert	Miller	Senior
8	Lisa	Davis	Adult
9	Thomas	Rodriguez	Adult
10	Jessica	Martinez	Adult
11	Daniel	Hernandez	Adult
12	Amanda	Lopez	Student

Members 11

Query 2: Books with fewer than 2 available copies

SELECT book_id, title, available_copies FROM Books WHERE available_copies < 2;

Output: Books such as “The Hobbit”, “1984”, etc.

Result Grid | Filter Rows: | Edit:

book_id	title	available_copies
2	1984	1
4	The Hobbit	0
5	Murder on the Orient Express	1
8	War and Peace	1
10	The Old Man and the Sea	1
11	The Great Gatsby	0
13	A Tale of Two Cities	1
14	One Hundred Years of Solitude	1
NULL	NULL	NULL

Books 12

Query 3: Overdue loans

SELECT Loans.loan_id, Members.first_name, Members.last_name, Books.title, due_date FROM Loans JOIN Members ON Loans.member_id = Members.member_id JOIN Books ON Loans.book_id = Books.book_id WHERE return_date IS NULL AND due_date < CURDATE();

Output: Returns overdue loans for users like Sarah Brown.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	loan_id	first_name	last_name	title	due_date
▶	4	Sarah	Brown	The Hobbit	2023-02-22
	6	Jennifer	Garcia	The Shining	2023-03-03
	8	Lisa	Davis	War and Peace	2023-02-26
	10	Jessica	Martinez	The Old Man and the Sea	2023-03-01
	12	Amanda	Lopez	Mrs Dalloway	2023-03-05
	14	Patricia	Wilson	One Hundred Years of Solitude	2023-03-08

Query 4: Members with unpaid fines

SELECT DISTINCT Members.member_id, first_name, last_name, email FROM Members JOIN Fines ON Members.member_id = Fines.member_id WHERE paid_status = FALSE;

Output: David Jones appears with unpaid fine.

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

	member_id	first_name	last_name	email
▶	5	David	Jones	david.j@email.com

Query 5: Top 5 most borrowed books

SELECT Books.title, COUNT(Loans.loan_id) AS times_borrowed FROM Books JOIN Loans ON Books.book_id = Loans.book_id GROUP BY Books.book_id ORDER BY times_borrowed DESC LIMIT 5;

Output: Most borrowed book is “Harry Potter and the Philosopher’s Stone”.

Result Grid | Filter Rows: | Export:

title	times_borrowed
Harry Potter and the Philosopher's Stone	2
1984	1
The Hobbit	1
Murder on the Orient Express	1
The Shining	1

Result 15

Query 6: Members who borrowed more than 2 books

SELECT Members.member_id, first_name, last_name, COUNT(Loans.loan_id) AS total_loans FROM Members JOIN Loans ON Members.member_id = Loans.member_id GROUP BY Members.member_id HAVING total_loans > 2;

Output: John Smith borrowed more than 2 times.

Result Grid | Filter Rows: | Export:

member_id	first_name	last_name	total_loans

Query 7: Books never borrowed

SELECT book_id, title FROM Books WHERE book_id NOT IN (SELECT DISTINCT book_id FROM Loans);

Output: No books left unborrowed (in sample data).

Query 8: Members who joined in 2023

SELECT member_id, first_name, last_name, join_date FROM Members WHERE YEAR(join_date) = 2023;

Output: Sarah Brown and Amanda Lopez joined in 2023.

Result Grid | Filter Rows: | Edit

book_id	title
3	To Kill a Mockingbird
NULL	NULL

Books 17

Query 9: Count of members by type

SELECT membership_type, COUNT(*) AS total_members FROM Members GROUP BY membership_type;

Output: Adult – 10, Senior – 3, Student – 1

Result Grid | Filter Rows: | Expor

membership_type	total_members
Adult	11
Senior	3
Student	1

Result 22

Query 10: Total fine paid by each member

SELECT Members.member_id, first_name, last_name, SUM(fine_amount) AS total_paid FROM Members JOIN Fines ON Members.member_id = Fines.member_id WHERE paid_status = TRUE GROUP BY Members.member_id;

Output: Michael Williams paid $2.50; Daniel Hernandez paid $2.00

Result Grid | Filter Rows: | Export:

member_id	first_name	last_name	total_paid
3	Michael	Williams	2.50
11	Daniel	Hernandez	2.00

Result 23

10. SUMMARY

The Library Management System demonstrated above is a robust solution to manage library records efficiently. With normalized tables, referential integrity, and well-defined queries, it facilitates quick information retrieval and ensures data consistency. Each component, from member registration to book loans and fine payments, is logically structured and linked.

11. CONCLUSION

This case study explored the development of a relational database for a Library Management System using MySQL. Through real-world scenarios and a well-structured schema, the system handles all core library functions. It highlights how relational databases can support comprehensive management systems, making it a strong candidate for academic, public, and corporate libraries.

Future enhancements may include GUI integration, role-based authentication, and mobile access features.