Project Report

## About the Project:

Introduction:

The Library Management System (LMS) is a comprehensive software solution designed to streamline library operations, enhance user experience, and provide efficient management of library resources. The system is tailored to meet the needs of libraries in various settings, including educational institutions, public libraries, and corporate libraries. By leveraging technology, the LMS aims to automate routine tasks, facilitate easy access to information, and empower librarians with tools for effective resource management.

Background:

Traditional library management involves manual processes for cataloging, circulation, and patron management, leading to inefficiencies and limitations in service delivery. With the advent of digitalization and advancements in information technology, there arises a need for modernizing library operations to meet the evolving needs of users. The Library Management System addresses these challenges by offering a centralized platform for managing library resources, tracking circulation, and providing seamless access to a wide range of materials.

Features and Services:

* Catalog Management: The LMS allows librarians to create and maintain a comprehensive catalog of library resources, including books, journals, multimedia, and digital assets. Cataloging features enable easy organization, classification, and retrieval of materials.
* User Management: Patrons can register and borrow books.
* Circulation Management: The system automates circulation processes, including check-in, check-out. Librarians can efficiently track the movement of books across users and branches.
* Search and Discovery: A user-friendly search interface enables patrons to explore the library catalog, search for specific titles and discover related materials.
* Reporting and Analytics: Users/Admin have access to reporting tools and analytics dashboards to monitor library usage, track circulation trends, and generate insights for collection development and resource allocation.

The created application implements all the above features. For simplicity, security details are not considered, which means, login-logout workflow and sql injection checks are not implemented.

The application is also designed for the purpose of both admins and users, and it has options to choose for each of these roles.

Conclusion:

The Library Management System offers a modern, user-centric approach to library operations, enabling libraries to deliver high-quality services, improve resource accessibility, and foster a culture of lifelong learning. By embracing digital transformation and leveraging innovative technologies, libraries can enhance their relevance, efficiency, and impact in the digital age.

# Part A: ER Diagram:

A diagram of a computer

Description automatically generated

The Entity-Relationship (E-R) diagram for the Library Management System (LMS) reflects the structure and relationships between various entities involved in the system. Here's how the diagram is produced and how it reflects the described application:

Entity Sets and Relationships:

* Writer: Stores information about authors including their name, gender, and qualification.
* Publisher: Contains details about book publishers such as their name, city, and ZIP code.
* Books: Manages records of books including their name, ISBN, writer ID, and publisher ID.
* Branch: Stores information about library branches, including their city and street.
* Reader: Holds data about library readers, including their name, email, phone number, city, and ZIP code.
* Loaned: Tracks loaned books, linking reader IDs and book IDs to branch IDs, loan dates, and return dates.

The “Publisher-Books” and “Writer-Books” relationships have been combined into the Books relations since they are many-one.

Producing the Diagram:

The E-R diagram is created using modeling tools or software that allows for the visualization of entities, attributes, and relationships.

Each entity is represented as a rectangular box, with its attributes listed inside.

Relationships between entities are depicted using lines connecting related entities, with labels indicating the nature of the relationship (e.g., one-to-many, many-to-many).

Reflection of the Application:

The E-R diagram reflects the core functionalities and data structures of the Library Management System. It showcases how writers, publishers, books, branches, and readers are interconnected within the system.

The relationships depicted in the diagram illustrate how users interact with books, borrow them from library branches, and how books are associated with their respective authors and publishers.

# Part B: Relational Schema

Creating the schemas from the above-mentioned ER diagram

- Writer (ID, Name, Gender, Qualification)

- Publisher (ID, Name, City, Zip)

- Books (ID, Name, ISBN, Write\_ID, Publisher\_ID)

- Branch (ID, City, Street)

- Reader (ID, Name, Email, Phone, City, Zip)

- Loaned (Reader\_ID, Book\_ID, Branch\_ID, Loan\_Date, Return\_Date)

**A diagram of a data flow

Description automatically generated**

**Functional Dependencies:**

Here are the list of functional dependencies that exist within the above tables. These are the functional dependencies that are expected to be held in real-world scenario.

#### Books:

ID → Name, Writer\_ID, Publisher\_ID

In this context, the "ID" attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

Writer:

ID → Name, Gender, Qualification

In this context, the "ID" attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

Publisher:

ID → Name, City, Zip

In this context, the "ID" attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

Branch:

ID → City, Street

Here, the "ID" attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

Reader:

ID → Name, Email, Phone, City, Zip

Here, the "ID" attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

Loaned:

(Reader\_ID, Book\_ID ) → Branch\_ID, Loan\_Date, Return\_Date

Here, the Reader\_ID, Book\_ID attribute serves as the super key, ensuring that the database is in Boyce-Codd Normal Form (BCNF), thereby maintaining optimal structural integrity.

In the above table, the Branch\_ID is not part of the composite primary key because each loan entry (Reader\_ID, Book\_ID) is associated with a specific branch through the Branch\_ID attribute. However, including Branch\_ID as part of the composite primary key would imply that each loan record is uniquely identified by the combination of Reader\_ID, Book\_ID, and Branch\_ID, which may not be accurate.

Including Branch\_ID in the primary key would suggest that the same book borrowed by the same reader from different branches constitutes different loan entries, which is not the desired behavior.

Instead, Branch\_ID serves as a foreign key in the Loaned table, establishing a relationship with the Branch table and indicating the branch from which the book was borrowed. This design allows for flexibility in tracking loans across different branches while maintaining accurate data representation in the Loaned table.

Now let us check if the above tables are in third normal form.

The third normal form (3NF) requires that a relation is in 2NF and that no non-prime attribute is transitively dependent on the primary key. In simpler terms, it means that all attributes in a table must depend only on the primary key and nothing else.

Books:

The attributes Name, Writer\_ID, and Publisher\_ID are dependent on the primary key ID.

All non-prime attributes (Name, Writer\_ID, Publisher\_ID) are directly dependent on the primary key.

Therefore, Books is in 3NF.

#### Writer:

The attributes Name, Gender, and Qualification are dependent on the primary key ID.

All non-prime attributes (Name, Gender, Qualification) are directly dependent on the primary key.

Therefore, Writer is in 3NF.

#### Publisher:

The attributes Name, City, and Zip are dependent on the primary key ID.

All non-prime attributes (Name, City, Zip) are directly dependent on the primary key.

Therefore, Publisher is in 3NF.

#### Branch:

The attributes City and Street are dependent on the primary key ID.

All non-prime attributes (City, Street) are directly dependent on the primary key.

Therefore, Branch is in 3NF.

#### Reader:

The attributes Name, Email, Phone, City, and Zip are dependent on the primary key ID.

All non-prime attributes (Name, Email, Phone, City, Zip) are directly dependent on the primary key.

Therefore, Reader is in 3NF.

#### Loaned:

The attributes Branch\_ID, Loan\_Date, and Return\_Date are dependent on the composite primary key (Reader\_ID, Book\_ID).

All non-prime attributes (Branch\_ID, Loan\_Date, Return\_Date) are directly dependent on the composite primary key (Reader\_ID, Book\_ID).

Therefore, Loaned is in 3NF.

In conclusion, all tables in the given schema appear to satisfy the conditions of the third normal form (3NF).

# Part C: Creating the Database using SQL

To create the database schema for my application, I used MySQL Workbench, which provides a user-friendly interface for designing and managing databases using SQL. Here's a brief overview of the steps I followed:

Designing the Schema: I first created the tables based on the relational schema outlined in Part B of the project. Using MySQL Workbench's visual design tools, I created each table, specifying the attributes, data types, and relationships between tables as per the schema.

Defining Relationships: MySQL Workbench allows for easy establishment of relationships between tables using foreign key constraints. I ensured that each relationship between tables was accurately represented in the schema.

Generating Data: For populating the database, I wrote a program in Python to generate both artificial made up data and real data sourced from Kaggle. For the artificial data, I created records conforming to the schema specifications, ensuring that the data was of a realistic size to allow for meaningful experimentation with the database. For the real data sourced from Kaggle, I transformed the datasets into files of records conforming to my database schema and MySQL's load format using excel.

Key attributes such as IDs (e.g., Writer\_ID, Publisher\_ID) are designated as primary keys in their respective tables. This ensures that each record in the table has a unique identifier, preventing duplication and maintaining data integrity. Unique constraints may also be applied to other attributes where necessary to enforce uniqueness.

When I generate datasets involving multiple relations, I ensure that joins between these relations are meaningful and accurate. This requires me to carefully map foreign keys to their corresponding primary keys and establish relationships that accurately reflect the real-world connections between entities. By doing so, I ensure that the data integrity is maintained, and the relationships within the database accurately represent the relationships in the real world.

Loading Data: Once the data files were ready, I inserted the tuples into the corresponding relations using the insert command. Please find the sql scripts used for this purpose in the github repo.

Verifying Data: After loading the data, I verified its correctness by running SQL queries to retrieve sample records from each table and comparing them with the original datasets.

MySQL Workbench provided a convenient and efficient platform for creating the database schema, populating the data, and ensuring the integrity and accuracy of the database for further development and analysis of my application.

I used the following SQL queries to create the tables in the MySQL database.

Publisher Table

create table Publisher

(

ID integer,

NAME varchar(100) not null,

CITY varchar(50),

ZIP varchar(5),

PRIMARY KEY(ID)

-- CONSTRAINT "Publisher\_ID\_PK" PRIMARY KEY ("ID") ENABLE,

-- CONSTRAINT "Publisher\_Zip\_CHK" CHECK ( "ZIP" like '\_\_\_\_\_') ENABLE

);

Writer Table

create table Writer

(

ID integer,

Name varchar(100) not null,

Age numeric(10),

Gender char(1) not null,

Qualification varchar(50),

PRIMARY KEY(ID)

-- constraint "Gender\_che" check (Gender in ('M', 'F')),

-- constraint "Author\_ID\_PK" primary key (ID)

);

Books Table

create table Books

(

ID integer,

Name varchar(100),

ISBN varchar(30) not null,

Writer\_ID integer,

Publisher\_ID integer,

No\_of\_Books numeric(10, 0),

PRIMARY KEY(ID),

FOREIGN KEY(Publisher\_ID) REFERENCES Publisher(ID) ON DELETE CASCADE,

FOREIGN KEY(Writer\_ID) REFERENCES Writer(ID) ON DELETE CASCADE

);

Branch Table

create table Branch

(

ID integer,

Street varchar(50),

City varchar(50) not null,

PRIMARY KEY(ID)

-- constraint "Branch\_ID\_PK" primary key (ID),

-- constraint "Branch\_ID\_CHK" check (ID like '\_\_\_')

);

Reader Table

create table Reader

(

ID integer,

Name varchar(100) not null,

Email varchar(100),

Phone varchar(11),

City varchar(50),

Zip varchar(4),

PRIMARY KEY(ID)

);

Loaned Table

create table Loaned

(

Reader\_ID integer not null,

Book\_ID integer not null,

Branch\_ID integer not null,

Issue\_Date date,

Return\_Date date,

PRIMARY KEY(Reader\_ID, Book\_ID),

FOREIGN KEY(Reader\_ID) REFERENCES Reader(ID) ON DELETE CASCADE,

FOREIGN KEY(Book\_ID) REFERENCES Books(ID) ON DELETE CASCADE,

FOREIGN KEY(Branch\_ID) REFERENCES Branch(ID) ON DELETE CASCADE

);

The CSV files used for data tables are attached in the repo, these contain real and synthetic data tuples.

# Part D: Developing a User Interface

As I developed the database and web application system for the library management system, I leveraged Flask, a Python web framework, to create the backend logic and serve dynamic web pages. Flask provides a lightweight yet powerful platform for building web applications, allowing me to focus on implementing the core functionality of the library management system while ensuring scalability and maintainability.

To interact with the MySQL database, I utilized the Python MySQL Connector, which is a standardized database driver for connecting Python applications to MySQL databases. This connector enables seamless communication between the Flask application and the MySQL database, facilitating the execution of SQL queries, retrieval of data, and manipulation of database records. The Flask application is structured according to the Model-View-Controller (MVC) architecture, where:

With Flask and the Python MySQL Connector, I was able to seamlessly integrate the frontend and backend components of the library management system, providing users with a smooth and intuitive experience. Users can access the system through their web browsers, perform various tasks such as searching for books, borrowing materials, and accessing library analytics, all while benefiting from the robustness and reliability of the Flask framework and MySQL database.

The user interface offers a range of functions to meet the needs of both patrons and administrators:

For Borrowing:

* Search for a Book: Patrons can search for books using keywords, titles, authors, or ISBN numbers. This feature allows users to quickly find the books they are interested in borrowing.
* Borrow a Book: Once users find the desired book, they can borrow it by selecting the appropriate option. This initiates the borrowing process, updating the database to reflect the loan transaction.
* Library Dashboard: Users have access to a dashboard that provides insights into library activities, such as the most borrowed books, popular genres. This feature offers valuable information for both patrons and administrators to understand library usage trends.

For Administration:

* Insert a New Record: Administrators can add new records to the database, such as books, authors, publishers, or users. This functionality allows for the expansion and maintenance of the library catalog.
* Run SQL Query: Administrators can execute custom SQL queries directly within the application. This advanced feature provides flexibility for data analysis, reporting, and system management.

These functions are implemented in SQL through a series of queries executed by the flask web application. For example, when a user searches for a book, the application executes a SQL query to retrieve relevant records from the database based on the search criteria. Similarly, borrowing a book involves updating the loan status of the book in the database through SQL queries.

Overall, the user interface of the library management system aims to provide an intuitive and efficient experience for users, allowing them to perform various tasks seamlessly while ensuring the integrity and security of the underlying database.

# Project source code

The complete project code is available in a github repo: <https://github.com/ParimiHarsha/LibraryManagementSystem>

# Discussion

Developing a database application, such as the library management system, was an enriching experience that presented various challenges and learning opportunities. Here are some key aspects of my experience:

Database Design Challenges:

One of the initial hurdles I faced was designing the database schema to accurately represent the relationships between different entities, such as books, authors, readers, and branches. Ensuring data integrity and normalization while accommodating complex relationships required careful planning and analysis.

To overcome this challenge, I conducted thorough research on database design principles and consulted resources such as textbooks and online tutorials. I also iteratively refined the schema based on feedback and testing to ensure it met the requirements of the library management system.

Implementation Complexity:

Implementing the backend logic and frontend interface of the application posed challenges, especially in integrating the Flask framework with the MySQL database and creating dynamic web pages.

To address these challenges, I broke down the development process into smaller tasks and tackled them incrementally. I leveraged the extensive documentation and community support available for Flask and MySQL to troubleshoot issues and find solutions.

Throughout the project, I embraced a growth mindset and viewed challenges as opportunities for learning and improvement. I actively sought feedback from peers and mentors, followed online forums and communities, and experimented with new technologies and techniques to expand my skills.

By embracing continuous learning and remaining adaptable in the face of challenges, I gained valuable insights into database management, web development, and software engineering practices. I also honed my problem-solving skills and developed a deeper appreciation for the intricacies of building robust and scalable database applications.

In summary, developing a database application like the library management system was a rewarding journey that involved overcoming challenges, acquiring new skills, and continuously refining my approach to software development.