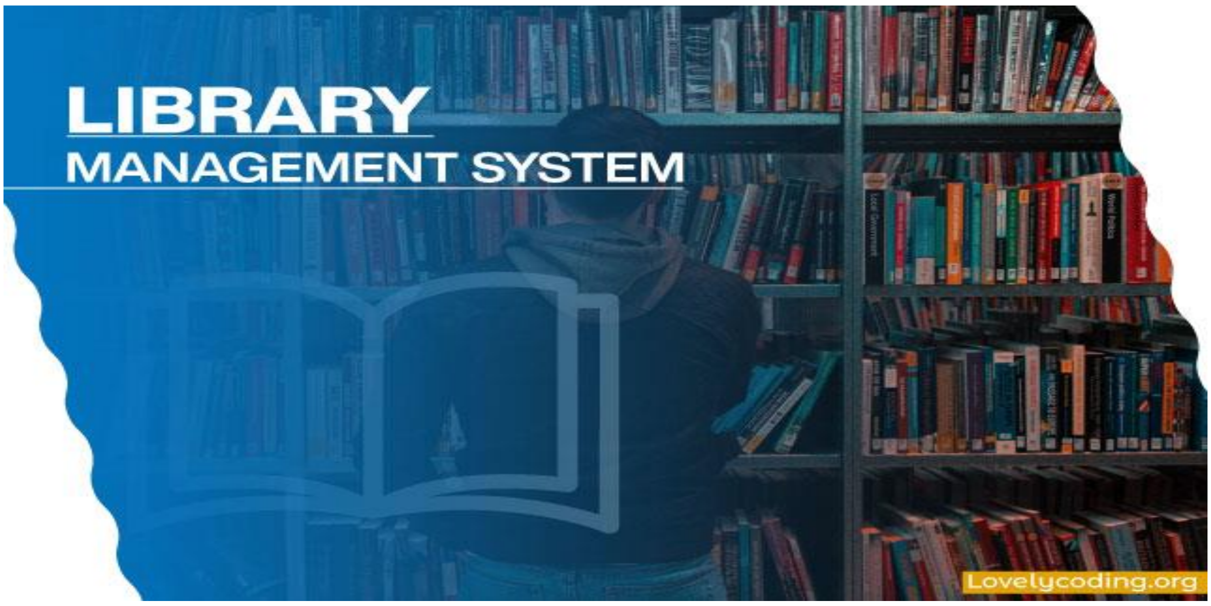
Employee ID: SP 11213
Employee Name: Nidhin Nandakumar
Project Name: Library Management System



Project Background

The Library Management System (LMS) project is aimed at providing an efficient and user-friendly solution to automate and streamline the operations of a modern library. In today's digital age, libraries face new challenges and opportunities in managing their collections, improving patron services, and adapting to the changing landscape of information management. This project addresses these challenges and leverages technology to enhance the library experience.

Project Objective

The primary objective of the Library Management System (LMS) project is to design, develop, and implement a comprehensive software solution that enhances the efficiency and effectiveness of library operations. The system aims to modernize and streamline the management of library resources, improve patron services, and provide data-driven insights for library administration.

The project seeks to achieve the following specific goals:

Automation and Cataloging: Implement a robust cataloging system (using the "Book," "Author," "Publisher," and "Genre" tables) to efficiently manage and track library resources. This includes automating the entry and maintenance of book records, author details, publisher information, and genre categorization.

Patron Services: Enhance patron services by creating a user-friendly interface for patrons to search and access library resources. The system (utilizing the "Patron," "Checkout," "Reservation," and

"Library_Card" tables) will enable patrons to borrow, return, and reserve items, and provide them with essential information, such as due dates and fine tracking.

Fine Management: Implement a fine tracking system (utilizing the "Fine" table) to accurately calculate and manage fines for overdue materials. Patrons will be informed about accrued fines, and library staff can efficiently handle fine collection and record-keeping.

Staff Management: Create a user management system (with the "Staff" table) to support library staff in their roles. Staff members will be able to access and manage the system securely and efficiently.

Activities:
Creating Tables using Queries

Creating Tables

Book Table

```
CREATE TABLE Book (
    ISBN VARCHAR(20) PRIMARY KEY,
    Title VARCHAR(255),
    Author_ID INT,
    Patron_ID INT,
    Publisher_ID INT,
    Genre_ID INT,
    Total_Copies INT
   );
```

```
Insertion of value example:

INSERT INTO Book (ISBN, Title, Author_ID, Publisher_ID, Genre_ID, Total_Copies)

VALUES('ISBN001', 'The Secret Garden', 1, 1, 1, 5)
```

Patron Table

```
CREATE TABLE Patron (
    Patron_ID INT PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Address TEXT,
    Phone_Number VARCHAR(15),
    Email VARCHAR(100)
);
```

```
Insertion of value example:

INSERT INTO Patron (Patron_ID, First_Name, Last_Name,Address,Phone_Number,Email,)
VALUES
(100, 'Roger', 'Williams', '123 Main St, City, State, Zip', '555-345-6789',
'roger.williams@example.com')
```

Checkout Table

```
CREATE TABLE Checkout (
    Checkout_ID INT PRIMARY KEY,
    Patron_ID INT,
    ISBN VARCHAR(20),
    Checkout_Date DATE,
    Due_Date DATE,
    Return_Date DATE

);
```

Insertion of value example:

```
INSERT INTO Checkout (Checkout_ID, Patron_ID, ISBN, Checkout_Date, Due_Date,
Return_Date)
VALUES
(1, 100, 'ISBN001', '2023-04-01', '2023-04-15', '2023-04-10'),
```

Author Table

```
CREATE TABLE Author (
    Author_ID INT PRIMARY KEY,
    Author_Name VARCHAR(100)
);
```

Insertion of value example:

```
INSERT INTO Author (Author_ID, Author_Name)

VALUES (1, 'Jane Austen')
```

Publisher Table

```
CREATE TABLE Publisher (
    Publisher_ID INT PRIMARY KEY,
    Publisher_Name VARCHAR(100)
);
```

Insertion of value example:

```
INSERT INTO Publisher (Publisher_ID, Publisher_Name)

VALUES (1, 'Penguin Books')
```

Genre Table

```
CREATE TABLE Genre (
    Genre_ID INT PRIMARY KEY,
    Genre_Name VARCHAR(50)
);
```

Insertion of value example:

```
INSERT INTO Genre (Genre_ID, Genre_Name)

VALUES (1, 'Classics')
```

Staff Table

```
CREATE TABLE Staff (
    Staff_ID INT PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50)

);
```

```
Insertion of value example:
```

```
INSERT INTO Staff (Staff_ID, First_Name, Last_Name)
VALUES
(1, 'John', 'Smith')
```

Reservation Table

```
CREATE TABLE Reservation (
    Reservation_ID INT PRIMARY KEY,
    Patron_ID INT,
    ISBN VARCHAR(20),
    Reservation_Date DATE

);
```

```
Insertion of value example:
```

```
INSERT INTO Reservation (Reservation_ID, Patron_ID, ISBN, Reservation_Date)
VALUES
(1, 100, 'ISBN001', '2023-03-01')
```

Library Table

```
CREATE TABLE Library_Card (
    Card_ID INT PRIMARY KEY,
    Patron_ID INT,
    Issue_Date DATE,
    Staff_ID INT,
    Expiration_Date DATE
);
```

```
Insertion of value example:
```

```
INSERT INTO Library_Card (Card_ID, Patron_ID, Issue_Date,Staff_ID,Expiration_Date)
VALUES
(1, 100, '2023-04-01',1, '2024-04-01')
```

Fine Table

```
CREATE TABLE Fine (
    Fine_ID INT PRIMARY KEY,
    Patron_ID INT,
    Amount DECIMAL(10, 2)
);
```

```
Insertion of value example:
```

```
INSERT INTO Fine (Fine_ID, Patron_ID, Amount)
VALUES
(1, 100, 5.00)
```

Transaction Table

```
CREATE TABLE Transactions (
    Transaction_ID INT PRIMARY KEY,
    Patron_ID INT,
    ISBN VARCHAR(20),
    Transaction_Date DATE,
    Transaction_Type VARCHAR(50)

);
```

Insertion of value example:

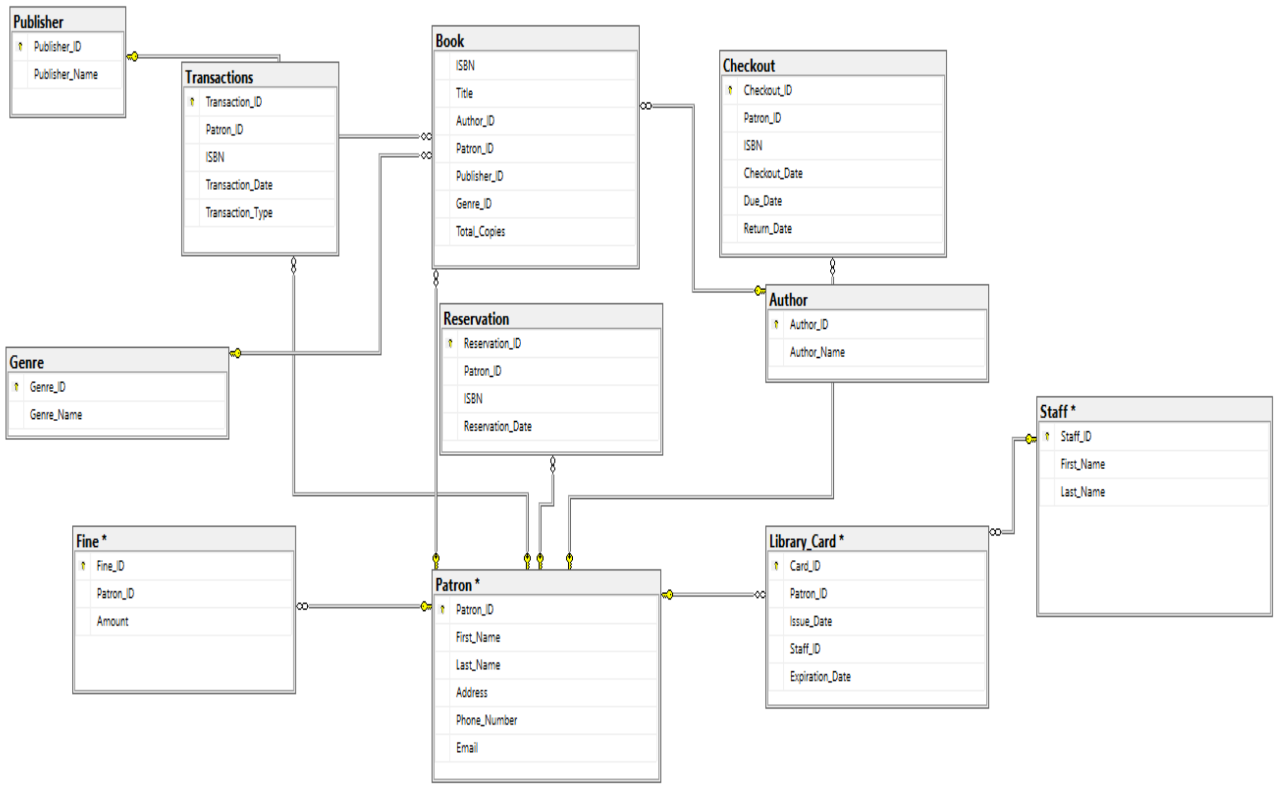
```
INSERT INTO Transactions (Transaction_ID, Patron_ID, ISBN,Transaction_Date,
Transaction_Type) VALUES (1, 101, 'ISBN001', '2023-10-10', 'Check Out')
```

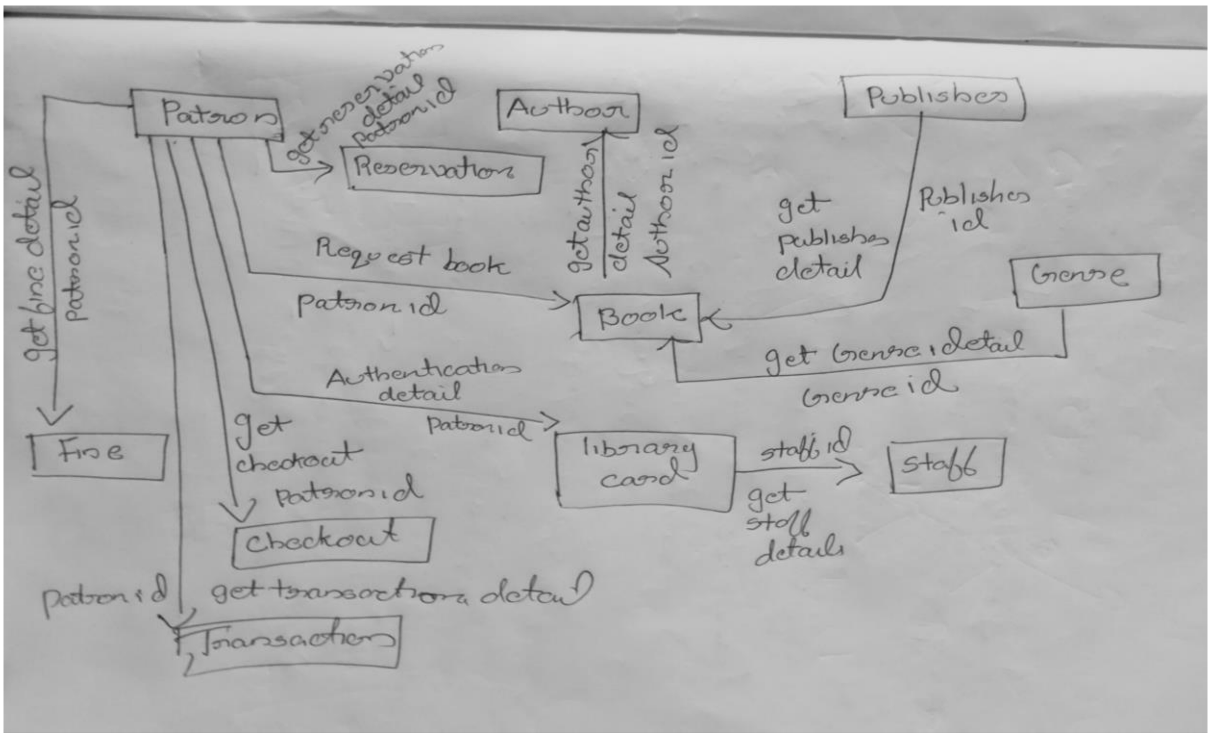
For setting foreign key here is the example of foreign key set in Library_Card table from table Patron.

```
FOREIGN KEY (Patron_ID) REFERENCES Patron(Patron_ID)
```

This is how the foreign keys are set in the corresponding tables as per schema diagram.

Schema Diagram and the Data-Flow Diagram





Entering Data in tables using queries and triggers

Here the trigger is used to calculate fine amount in the fine table on adding data in the checkout table ,if the book is delayed to return.

```
CREATE TRIGGER CalculateFine
ON Checkout
AFTER INSERT
AS
BEGIN
    UPDATE Fine
    SET Amount = DATEDIFF(DAY, i.Due_Date, i.Return_Date) * 5
    FROM Fine
    JOIN Inserted AS i ON Fine.Patron_ID = i.Patron_ID
    WHERE i.Return_Date > i.Due_Date;
END;
```

Here is an example how value inserted to checkout table so that the trigger will be invoked.

```
insert into checkout values(26,103,'ISBN022','3-16-2023','4-08-2023','4-16-2023');
```

or

Here is an example for the update query that can be used to update the fine values

```
UPDATE Fine
SET Amount = DATEDIFF(DAY, Checkout.Due_Date, Checkout.Return_Date) * 5
FROM Fine
JOIN Checkout ON Fine.Patron_ID = Checkout.Patron_ID
WHERE Checkout.Return_Date > Checkout.Due_Date;
```

Creating Views

View 1

Here the View named CheckoutDetails is created to list the name of book,their patrons and their following checkout dates.

```
CREATE VIEW CheckoutDetails AS
SELECT Book.Title, Patron.First_Name, Patron.Last_Name, Checkout.Checkout_Date
FROM Book
INNER JOIN Checkout ON Book.ISBN = Checkout.ISBN
INNER JOIN Patron ON Checkout.Patron_ID = Patron.Patron_ID;

SELECT * FROM CheckoutDetails;
```

View 2

Here the view named AuthorInfo shows authors and the books they've written in a single, easy-to-access list. It simplifies the process of finding out which author wrote which books.

```
CREATE VIEW AuthorInfo AS
SELECT A.Author_ID, A.Author_Name,
       B.ISBN, B.Title
FROM Author A
JOIN Book B ON A.Author_ID = B.Author_ID;
```

```
select * from AuthorInfo
```

View 3

This view "PatronTransactions" view displays transaction details, including transaction ID, date, type, along with patron and book information, making it easy to track transactions associated with specific patrons and books.

```
CREATE VIEW PatronTransactions AS
SELECT T.Transaction_ID, T.Transaction_Date, T.Transaction_Type,
       P.Patron_ID, P.First_Name, P.Last_Name,
       B.ISBN, B.Title
FROM Transactions T
JOIN Patron P ON T.Patron_ID = P.Patron_ID
JOIN Book B ON T.ISBN = B.ISBN;

select * from PatronTransactions
```

Reports using queries

Query to List all the books checked out by patrons(Users) with library cards issued by Library Staff Named 'John Smith':

```
SELECT Book.ISBN, Book.Title, Checkout.Checkout_Date, Checkout.Due_Date,
Checkout.Return_Date
FROM Book
INNER JOIN Checkout ON Book.ISBN = Checkout.ISBN
INNER JOIN Library_Card ON Checkout.Patron_ID = Library_Card.Patron_ID
INNER JOIN Staff ON Library_Card.Staff_ID = Staff.Staff_ID
WHERE Staff.First_Name = 'John' AND Staff.Last_Name = 'Smith';
```

Query to calculate the total fine amount for each patron and find the patron's with the highest total fine in descending order of the fine amount.

```
SELECT Patron.Patron_ID, Patron.First_Name, Patron.Last_Name, SUM(Fine.Amount) AS
Total_Fines
FROM Patron
LEFT JOIN Fine ON Patron.Patron_ID = Fine.Patron_ID
GROUP BY Patron.Patron_ID, Patron.First_Name, Patron.Last_Name
ORDER BY Total_Fines DESC
```

Query to find the books that were checked out and returned late, along with the number of days they were overdue:

```
SELECT Book.ISBN, Book.Title, Checkout.Checkout_Date, Checkout.Due_Date,
Checkout.Return_Date,
       DATEDIFF(day, Checkout.Due_Date, Checkout.Return_Date) AS Days_Overdue
FROM Book
INNER JOIN Checkout ON Book.ISBN = Checkout.ISBN
WHERE Checkout.Return_Date > Checkout.Due_Date;
```

Query to calculate the average fine amount for each patron who has checked out books more than once, ordered by average fine :

```
SELECT Fine.Patron_ID, Patron.First_Name, Patron.Last_Name, AVG(Fine.Amount) AS
Avg_Fine
FROM Patron
INNER JOIN Fine ON Patron.Patron_ID = Fine.Patron_ID
INNER JOIN Checkout ON Patron.Patron_ID = Checkout.Patron_ID
GROUP BY Fine.Patron_ID, Patron.First_Name, Patron.Last_Name
HAVING COUNT(Checkout.Checkout_ID) > 1
ORDER BY Avg_Fine;
```

Query used to calculate and display the total fines collected each month, breaking down the results by year and month:

```
SELECT
    YEAR(Transactions.Transaction_Date) AS Collection_Year,
    MONTH(Transactions.Transaction_Date) AS Collection_Month,
    SUM(Fine.Amount) AS Total_Fines
FROM Transactions
INNER JOIN Fine ON Transactions.Patron_ID = Fine.Patron_ID
GROUP BY YEAR(Transactions.Transaction_Date), MONTH(Transactions.Transaction_Date)
ORDER BY YEAR(Transactions.Transaction_Date), MONTH(Transactions.Transaction_Date);
```

Query to find the most popular genre based on the total number of book copies:

```
SELECT Genre.Genre_Name, SUM(Book.Total_Copies) AS Total_Copies
FROM Book
INNER JOIN Genre ON Book.Genre_ID = Genre.Genre_ID
GROUP BY Genre.Genre_Name
ORDER BY Total_Copies DESC
```

Query to find the number of books that are not reserved:

```
SELECT COUNT(*) AS NumberOfBooksNotReserved
FROM Book b
LEFT JOIN Reservation r ON b.ISBN = r.ISBN
WHERE r.Reservation_ID IS NULL;
```

Query to find all reservations made by a specific patron with their reservation dates:

```
SELECT Book.Title, Reservation.Reservation_Date
FROM Book
JOIN Reservation ON Book.ISBN = Reservation.ISBN
WHERE Reservation.Patron_ID = 120;
```

Query to list all transactions (checkouts, returns, etc.) for a specific patron within a date range:

```
SELECT ISBN, Transaction_Date, Transaction_Type
FROM Transactions
WHERE Patron_ID = 100
AND Transaction_Date BETWEEN '2023-01-01' AND '2023-12-31';
```

Query used to retrieve the latest transaction date for each transaction type in the library database while including the associated patron's first name, patron's last name, and book title:

```
SELECT Transactions.Transaction_Type, MAX(Transactions.Transaction_Date) AS
Latest_Transaction_Date,
       Patron.First_Name, Patron.Last_Name, Book.Title
FROM Transactions
JOIN Patron ON Transactions.Patron_ID = Patron.Patron_ID
JOIN Book ON Transactions.ISBN = Book.ISBN
GROUP BY Transactions.Transaction_Type, Patron.First_Name, Patron.Last_Name,
Book.Title;
```

Query to retrieve all transactions (of a specific type) for a group of patrons:

```
SELECT Patron.First_Name, Patron.Last_Name, Transactions.Transaction_Type,
Transactions.Transaction_Date
FROM Patron
JOIN Transactions ON Patron.Patron_ID = Transactions.Patron_ID
WHERE Patron.Patron_ID IN (100, 101, 102)
AND Transactions.Transaction_Type = 'Check Out';
```

Query to find all patrons who have checked out books from a specific list of genres:

```
SELECT DISTINCT Patron.First_Name, Patron.Last_Name
FROM Patron
JOIN Checkout ON Patron.Patron_ID = Checkout.Patron_ID
JOIN Book ON Checkout.ISBN = Book.ISBN
JOIN Genre ON Book.Genre_ID = Genre.Genre_ID
WHERE Genre.Genre_Name IN ('Mystery', 'Science Fiction', 'Romance');
```

Query to find the average number of days books are checked out by genre:

```
SELECT Genre.Genre_Name, AVG(DATEDIFF(day, Checkout.Checkout_Date,
Checkout.Return_Date)) AS Avg_Days_Checked_Out
FROM Genre
JOIN Book ON Genre.Genre_ID = Book.Genre_ID
JOIN Checkout ON Book.ISBN = Checkout.ISBN
GROUP BY Genre.Genre_Name;
```

Query to find the patrons who have both checked out and reserved books:

```
SELECT DISTINCT Patron.Patron_ID, Patron.First_Name, Patron.Last_Name
FROM Patron
WHERE Patron.Patron_ID IN (
    SELECT Patron_ID FROM Checkout
    INTERSECT
    SELECT Patron_ID FROM Reservation
);
```

Query to list all the staff members and the number of library cards they have issued:

```
SELECT Staff.Staff_ID, Staff.First_Name, Staff.Last_Name, COUNT(Library_Card.Card_ID)
AS Cards_Issued
FROM Staff
LEFT JOIN Library_Card ON Staff.Staff_ID = Library_Card.Staff_ID
GROUP BY Staff.Staff_ID, Staff.First_Name, Staff.Last_Name;
```