**CSA0593**

**DATABASE MANAGEMENT SYSTEM**

**K. SWETHA**

**192311404**

**ASSIGNMENT – 5**

**Design a database for a book library system with books, authors, members, and borrow history.  
  
-Model tables for books, authors, library members, and borrowing records.**

**-Write stored procedures for borrowing books, returning books, and managing late fees.**

**-Implement triggers to automatically update book availability and member borrowing limits.**

**-Write SQL queries to analyse popular authors, member borrowing trends, and late return pattern**

Designing a database for a **book library system** involves modeling tables for **books**, **authors**, **members**, and **borrow history**. These tables should allow efficient management of books, tracking borrow activities, storing information about authors, and maintaining member details.

1. **Database Schema**

#### ****1.1 Books Table****

Stores information about the books in the library.

CREATE TABLE Books (

BookID INT PRIMARY KEY AUTO\_INCREMENT

Title VARCHAR(255) NOT NULL,

Genre VARCHAR(100),

PublishedYear INT,

ISBN VARCHAR(20) UNIQUE

AuthorID INT,

AvailableCopies INT,

TotalCopies INT,

FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID) ON DELETE SET NULL

);

#### ****1.2 Authors Table****

Stores information about the authors of books.

CREATE TABLE Authors (

AuthorID INT PRIMARY KEY AUTO\_INCREMENT, FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

BirthYear INT,

DeathYear INT,

Biography TEXT

);

#### ****1.3 Members Table****

Stores information about library members (borrowers).

CREATE TABLE Members (

MemberID INT PRIMARY KEY AUTO\_INCREMENT FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

Email VARCHAR(255) UNIQUE,

Phone VARCHAR(20),

Address VARCHAR(255),

MembershipDate DATETIME DEFAULT CURRENT\_TIMESTAMP

);

#### ****1.4 Borrow History Table****

Tracks the borrowing history of members, including the borrow date and return date.

CREATE TABLE BorrowHistory (

BorrowID INT PRIMARY KEY AUTO\_INCREMENT MemberID INT,

BookID INT,

BorrowDate DATETIME DEFAULT CURRENT\_TIMESTAMP

ReturnDate DATETIME,

DueDate DATETIME,

FOREIGN KEY (MemberID) REFERENCES Members(MemberID),

FOREIGN KEY (BookID) REFERENCES Books(BookID)

);

### **2. Database Relationships**

* **Books ↔ Authors**: A **book** is written by one **author**, but an **author** can write many books, creating a **one-to-many** relationship between **Authors** and **Books**.
* **Members ↔ BorrowHistory**: A **member** can borrow multiple **books** over time, creating a **one-to-many** relationship between **Members** and **BorrowHistory**.
* **Books ↔ BorrowHistory**: A **book** can be borrowed multiple times, and each borrow transaction is logged in the **BorrowHistory** table, creating a **one-to-many** relationship between **Books** and **BorrowHistory**.

**-Write stored procedures for borrowing books, returning books, and managing late fees**:

To implement the operations for **borrowing books**, **returning books**, and **managing late fees**, we will create stored procedures that handle these actions. The stored procedures will update relevant tables, such as Books, BorrowingRecords, and Members, as well as calculate and apply late fees when necessary.

**1. Borrow Book**

This stored procedure will allow a member to borrow a book, update the BorrowingRecords table, and decrement the AvailableCopies in the Books table.

**1.1 Borrow Book Stored Procedure**

DELIMITER $$

CREATE PROCEDURE BorrowBook (

IN p\_MemberID INT,

IN p\_BookID INT

)

BEGIN

DECLARE available INT;

SELECT AvailableCopies INTO available FROM Books WHERE BookID = p\_BookID;

IF available > 0 THEN

INSERT INTO BorrowingRecords (MemberID, BookID, DueDate)

VALUES (p\_MemberID, p\_BookID, DATE\_ADD(CURRENT\_DATE, INTERVAL 14 DAY));

UPDATE Books

SET AvailableCopies = AvailableCopies - 1

WHERE BookID = p\_BookID;

SELECT 'Book borrowed successfully' AS Message;

ELSE

SELECT 'Book is not available' AS Message;

END IF;

END $$

DELIMITER ;

* **Input**: Member ID (p\_MemberID), Book ID (p\_BookID).
* **Process**:
  + Checks if the book has available copies.
  + If available, records the borrowing transaction in the BorrowingRecords table, sets the due date, and decrements the available copies in the Books table.
  + If the book is unavailable, a message is returned.

### **2. Return Book**

This stored procedure will handle the return of a book by a member. It will update the BorrowingRecords table and increment the AvailableCopies in the Books table. If the book is returned late, the late fee will be calculated.

#### ****2.1 Return Book Stored Procedure****

DELIMITER $$

CREATE PROCEDURE ReturnBook (

IN p\_BorrowID INT

)

BEGIN

DECLARE due\_date DATETIME;

DECLARE return\_date DATETIME;

DECLARE late\_fee DECIMAL(10,2);

DECLARE days\_late INT;

SELECT DueDate, CURRENT\_DATE INTO due\_date, return\_date

FROM BorrowingRecords

WHERE BorrowID = p\_BorrowID;

IF return\_date IS NOT NULL THEN

SELECT 'Book has already been returned.' AS Message;

LEAVE proc;

END IF;

IF return\_date > due\_date THEN

SET days\_late = DATEDIFF(return\_date, due\_date);

SET late\_fee = days\_late \* 1.00; -- Assuming $1 per day late fee

ELSE

SET late\_fee = 0;

END IF;

UPDATE BorrowingRecords

SET ReturnDate = return\_date, LateFee = late\_fee

WHERE BorrowID = p\_BorrowID;

UPDATE Books

SET AvailableCopies = AvailableCopies + 1

WHERE BookID = (SELECT BookID FROM BorrowingRecords WHERE BorrowID = p\_BorrowID);

UPDATE Members

SET Balance = Balance + late\_fee

WHERE MemberID = (SELECT MemberID FROM BorrowingRecords WHERE BorrowID = p\_BorrowID);

SELECT 'Book returned successfully' AS Message, late\_fee AS LateFee;

END $$

DELIMITER ;

 **Input**: Borrow ID (p\_BorrowID).

 **Process**:

* Fetches the due date and current return date from BorrowingRecords.
* If the book is returned after the due date, it calculates the late fee based on the number of days late.
* Updates the BorrowingRecords table with the return date and late fee.
* Increments the AvailableCopies of the book in the Books table.
* Updates the member’s balance in the Members table to reflect the late fee.
* Returns a message with the late fee (if any).

### **3. Manage Late Fees**

This stored procedure will calculate and apply late fees for overdue books that have not yet been returned. It runs periodically (e.g., daily) to ensure that overdue books are penalized.

#### ****3.1 Manage Late Fees Stored Procedure****

DELIMITER $$

CREATE PROCEDURE ManageLateFees ()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE borrow\_id INT;

DECLARE due\_date DATETIME;

DECLARE current\_date DATETIME;

DECLARE late\_fee DECIMAL(10,2);

DECLARE member\_id INT;

DECLARE days\_late INT;

DECLARE overdue\_books CURSOR FOR

SELECT BorrowID, DueDate, MemberID FROM BorrowingRecords

WHERE ReturnDate IS NULL AND DueDate < CURRENT\_DATE;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN overdue\_books;

read\_loop: LOOP

FETCH overdue\_books INTO borrow\_id, due\_date, member\_id;

IF done THEN

LEAVE read\_loop;

END IF;

SET current\_date = CURRENT\_DATE;

SET days\_late = DATEDIFF(current\_date, due\_date);

SET late\_fee = days\_late \* 1.00; -- Assuming $1 per day late fee

UPDATE BorrowingRecords

SET LateFee = late\_fee

WHERE BorrowID = borrow\_id;

UPDATE Members

SET Balance = Balance + late\_fee

WHERE MemberID = member\_id;

END LOOP;

CLOSE overdue\_books;

SELECT 'Late fees applied successfully.' AS Message;

END $$

DELIMITER ;

* **Input**: None.
* **Process**:
  + Loops through all overdue books (books that are not yet returned and are past their due date).
  + Calculates the late fee based on the number of days overdue.
  + Updates the BorrowingRecords table with the calculated late fee.
  + Updates the member's balance in the Members table to reflect the late fee.

**-Implement triggers to automatically update book availability and member borrowing limits:**

To implement **triggers** that automatically update **book availability** and **member borrowing limits**, we'll need two primary triggers:

1. **Trigger to Update Book Availability**: This trigger will automatically update the number of available copies of a book whenever a book is borrowed or returned.
2. **Trigger to Update Member Borrowing Limits**: This trigger will check and update a member's borrowing limit whenever they borrow or return a book. This ensures that a member cannot exceed their borrowing limit.

Let's implement these triggers:

### **1. Trigger to Update Book Availability**

This trigger will automatically update the AvailableCopies in the Books table when a book is borrowed or returned.

* When a book is borrowed, the number of available copies decreases by 1.
* When a book is returned, the number of available copies increases by 1.

#### ****Trigger for Borrowing Books****

#### DELIMITER $$

#### CREATE TRIGGER UpdateBookAvailability\_Borrow

#### AFTER INSERT ON BorrowingRecords

#### FOR EACH ROW

#### BEGIN

#### UPDATE Books

#### SET AvailableCopies = AvailableCopies - 1

#### WHERE BookID = NEW.BookID;

#### END $$

#### DELIMITER ;

* **Trigger Type**: AFTER INSERT on the BorrowingRecords table.
* **Action**: When a new borrowing record is added (i.e., a member borrows a book), the AvailableCopies in the Books table is decreased by 1.
* **Benefit**: This ensures that the availability of books is updated automatically whenever a book is borrowed.

#### ****1.2 Trigger for Returning Books****

#### DELIMITER $$

#### CREATE TRIGGER UpdateBookAvailability\_Return

#### AFTER UPDATE ON BorrowingRecords

#### FOR EACH ROW

#### BEGIN

#### IF OLD.ReturnDate IS NULL AND NEW.ReturnDate IS NOT NULL THEN

#### UPDATE Books

#### SET AvailableCopies = AvailableCopies + 1

#### WHERE BookID = NEW.BookID;

#### END IF;

#### END $$

#### DELIMITER ;

 **Trigger Type**: AFTER UPDATE on the BorrowingRecords table.

 **Action**: When the ReturnDate of a borrowing record is updated (indicating a book is returned), the AvailableCopies in the Books table is increased by 1.

####  **Benefit**: This ensures that the availability of books is updated automatically whenever a book is returned.

### **2. Trigger to Update Member Borrowing Limits**

This trigger will ensure that members cannot exceed their borrowing limit. We will assume that there is a column MaxBorrowLimit in the Members table which defines the maximum number of books a member can borrow at one time.

* When a new borrowing record is inserted, the trigger will check if the member has exceeded their borrowing limit.
* If the borrowing limit is exceeded, the trigger will cancel the borrowing action by raising an error.

#### ****2.1 Trigger for Borrowing Books (Borrow Limit Check)****

#### DELIMITER $$

#### CREATE TRIGGER CheckMemberBorrowLimit

#### BEFORE INSERT ON BorrowingRecords

#### FOR EACH ROW

#### BEGIN

#### DECLARE current\_borrowed INT;

#### DECLARE max\_limit INT;

#### SELECT COUNT(\*) INTO current\_borrowed

#### FROM BorrowingRecords

#### WHERE MemberID = NEW.MemberID AND ReturnDate IS NULL;

#### SELECT MaxBorrowLimit INTO max\_limit

#### FROM Members

#### WHERE MemberID = NEW.MemberID;

#### IF current\_borrowed >= max\_limit THEN

#### SIGNAL SQLSTATE '45000'

#### SET MESSAGE\_TEXT = 'Member has exceeded their borrowing limit.';

#### END IF;

#### END $$

#### DELIMITER ;

* **Trigger Type**: BEFORE INSERT on the BorrowingRecords table.
* **Action**:
  + It checks the number of books currently borrowed by the member (i.e., those with a NULL ReturnDate).
  + It compares the number of books borrowed with the MaxBorrowLimit of the member.
  + If the member has already reached or exceeded the borrowing limit, it raises an error (SIGNAL SQLSTATE '45000'), which will prevent the borrowing record from being inserted.
* **Benefit**: This ensures that a member cannot borrow more books than their allowed borrowing limit.

### **3. Summary of Triggers**

1. **UpdateBookAvailability\_Borrow**: Automatically updates the book availability when a book is borrowed. The available copies decrease by 1 each time a book is borrowed.
2. **UpdateBookAvailability\_Return**: Automatically updates the book availability when a book is returned. The available copies increase by 1 each time a book is returned.
3. **CheckMemberBorrowLimit**: Prevents a member from borrowing more books than their maximum allowed borrowing limit by checking their current borrow count before allowing a new borrow.

**-Write SQL queries to analyse popular authors, member borrowing trends, and late return pattern:**

To analyze **popular authors**, **member borrowing trends**, and **late return patterns**, we need to write SQL queries that provide insights from the library system's data. Below are the SQL queries for each analysis.

**1. Popular Authors Analysis**

This query will identify the most popular authors based on the number of books borrowed. It will join the Books, Authors, and BorrowingRecords tables to count how many times books by each author were borrowed.

**1.1 Query for Popular Authors**

SELECT a.AuthorID, a.AuthorName, COUNT(br.BorrowID) AS BorrowCount

FROM Authors a

JOIN Books b ON a.AuthorID = b.AuthorID

JOIN BorrowingRecords br ON b.BookID = br.BookID

WHERE br.ReturnDate IS NOT NULL -- Only count books that have been borrowed and returned

GROUP BY a.AuthorID, a.AuthorName

ORDER BY BorrowCount DESC

LIMIT 10;

**Explanation**:

* **JOIN**s the Authors, Books, and BorrowingRecords tables to get the borrowing data for each author.
* **COUNT(br.BorrowID)** counts the number of borrowings for each author.
* The query filters for books that have been **borrowed and returned** (WHERE br.ReturnDate IS NOT NULL).
* The results are ordered by the count of borrows, and the top 10 authors are returned.

### **2. Member Borrowing Trends**

This query will provide insights into the borrowing trends of members, such as the number of books borrowed per member over a given period.

#### ****2.1 Query for Member Borrowing Trends****

SELECT m.MemberID, m.MemberName, COUNT(br.BorrowID) AS TotalBooksBorrowed,

DATE\_FORMAT(br.BorrowDate, '%Y-%m') AS BorrowMonth

FROM Members m

JOIN BorrowingRecords br ON m.MemberID = br.MemberID

WHERE br.ReturnDate IS NULL

GROUP BY m.MemberID, m.MemberName, BorrowMonth

ORDER BY BorrowMonth DESC, TotalBooksBorrowed DESC;

* **Explanation**:
  + **JOIN**s the Members and BorrowingRecords tables to get the borrowing data.
  + **COUNT(br.BorrowID)** counts the total number of books borrowed by each member.
  + **DATE\_FORMAT(br.BorrowDate, '%Y-%m')** groups the borrowings by month (you can adjust this to yearly or weekly as needed).
  + The results are ordered by the month and total books borrowed, showing trends over time.

### **3. Late Return Patterns**

This query will analyze the late return patterns by identifying which members frequently return books after the due date. It calculates the number of days late for each return and groups the results by member.

#### ****3.1 Query for Late Return Patterns****

SELECT m.MemberID, m.MemberName, COUNT(br.BorrowID) AS LateReturns,

AVG(DATEDIFF(br.ReturnDate, br.DueDate)) AS AvgDaysLate

FROM Members m

JOIN BorrowingRecords br ON m.MemberID = br.MemberID

WHERE br.ReturnDate IS NOT NULL AND br.ReturnDate > br.DueDate

GROUP BY m.MemberID, m.MemberName

ORDER BY LateReturns DESC, AvgDaysLate DESC;

**Explanation**:

* **JOIN**s the Members and BorrowingRecords tables to get the return information.
* **DATEDIFF(br.ReturnDate, br.DueDate)** calculates the number of days a book is late.
* **AVG(DATEDIFF(...))** calculates the average number of days late per member.
* **COUNT(br.BorrowID)** counts the number of late returns per member.
* The query is filtered to only include books that were returned **late** (WHERE br.ReturnDate > br.DueDate).
* Results are ordered by the total number of late returns and the average days late.

### **4. Additional Analysis (Borrowing Frequency by Month)**

If you want to analyze borrowing frequency over time (e.g., the number of borrowings per month), you can use this query:

#### ****4.1 Query for Borrowing Frequency by Month****

SELECT DATE\_FORMAT(br.BorrowDate, '%Y-%m') AS BorrowMonth,

COUNT(br.BorrowID) AS TotalBorrows

FROM BorrowingRecords br

WHERE br.ReturnDate IS NULL -- Only count current borrowings

GROUP BY BorrowMonth

ORDER BY BorrowMonth DESC;

* **Explanation**:
  + **DATE\_FORMAT(br.BorrowDate, '%Y-%m')** groups borrowings by month.
  + **COUNT(br.BorrowID)** counts the total number of borrowings in each month.
  + The query is filtered for borrowings that are currently active (i.e., not yet returned).

**CONCLUSION:**

The database for the book library system effectively tracks key aspects of library operations, including book availability, member borrowing behavior, and late return patterns. By implementing triggers to automatically update book availability and member borrowing limits, the system ensures accurate real-time data. SQL queries for analyzing popular authors, member borrowing trends, and late return patterns provide valuable insights for library management, enabling better decision-making. These features contribute to a more efficient and user-friendly library system, helping to optimize resource allocation, monitor member activity, and improve overall library service quality.