============================================================

ADVANCED LIBRARY MANAGEMENT SYSTEM - REPORT

============================================================

Author: Aarya Gupta

Institution: KIET Group of Institutions

Program: B.Tech – Computer Science (AI/ML)

GitHub: https://github.com/your-username

------------------------------------------------------------

PROJECT OVERVIEW

------------------------------------------------------------

This project extends a relational SQL-based Library Management System with advanced data handling.

It focuses on core data modeling, insertions, many-to-many relationships, views, triggers, and reporting.

Each component is modular, relational, and scalable for educational or production environments.

------------------------------------------------------------

TASK BREAKDOWN

------------------------------------------------------------

1. DEFINE SCHEMA: BOOKS, AUTHORS, MEMBERS, LOANS

------------------------------------------------------------

Tables and relationships:

• Authors

- AuthorID (PK)

- Name

• Categories

- CategoryID (PK)

- Name

• Books

- BookID (PK)

- Title

- AuthorID (FK) → Authors

- CategoryID (FK) → Categories

• Members

- MemberID (PK)

- Name

- Email

• Loans

- LoanID (PK)

- BookID (FK) → Books

- MemberID (FK) → Members

- LoanDate

- ReturnDate

• Staff

- StaffID (PK)

- Name

- Role

Constraints:

- Foreign keys with cascading or SET NULL behavior

- Primary keys on all tables

- NOT NULL, DEFAULT constraints as needed

------------------------------------------------------------

2. INSERT TEST DATA FOR EACH ENTITY

------------------------------------------------------------

- 25–50 entries added per table

- Realistic data: names, book titles, categories, emails, roles

- NULL and default values used where allowed

- Loan dates designed to test overdue logic

------------------------------------------------------------

3. HANDLE MANY-TO-MANY RELATIONSHIPS WITH BRIDGE TABLES

------------------------------------------------------------

To support many-to-many (e.g., books with multiple authors), a bridge table like this can be used:

CREATE TABLE BookAuthors (

BookID INT,

AuthorID INT,

PRIMARY KEY (BookID, AuthorID),

FOREIGN KEY (BookID) REFERENCES Books(BookID),

FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)

);

This maintains normalized design and referential integrity.

------------------------------------------------------------

4. CREATE VIEWS FOR BORROWED AND OVERDUE BOOKS

------------------------------------------------------------

`BorrowedBooks\_View`

- Lists currently borrowed books with member info

`OverdueBooks\_View`

- Shows loans not returned in 30+ days:

SELECT \* FROM Loans

WHERE ReturnDate IS NULL AND LoanDate < DATE\_SUB(CURDATE(), INTERVAL 30 DAY);

Other useful views:

- `Books\_Never\_Borrowed\_View`

- `Loan\_History\_Per\_Member`

- `Books\_By\_Category`

- `Most\_Borrowed\_Books\_View`

Views simplify frequent queries and promote reusability.

------------------------------------------------------------

5. WRITE TRIGGERS FOR DUE-DATE NOTIFICATIONS

------------------------------------------------------------

Sample trigger that detects overdue loans:

CREATE TRIGGER DueDate\_Notifier

AFTER INSERT ON Loans

FOR EACH ROW

BEGIN

IF NEW.ReturnDate IS NULL AND NEW.LoanDate < CURDATE() - INTERVAL 30 DAY THEN

INSERT INTO Notifications(MemberID, Message)

VALUES (NEW.MemberID, ' Your loan is overdue!');

END IF;

END;

Other possible triggers:

- Auto-set default ReturnDate

- Audit loan extensions

------------------------------------------------------------

6. WRITE REPORTS USING AGGREGATION AND JOINS

------------------------------------------------------------

Reports created:

• Most Borrowed Books:

SELECT BookID, COUNT(\*) FROM Loans GROUP BY BookID ORDER BY COUNT(\*) DESC;

• Member Loan Counts:

SELECT MemberID, COUNT(\*) FROM Loans GROUP BY MemberID;

• Category-Wise Book Count:

SELECT c.Name, COUNT(\*) FROM Books b

JOIN Categories c ON b.CategoryID = c.CategoryID GROUP BY c.Name;

• Recent Loans (last 30 days):

SELECT \* FROM Loans WHERE LoanDate >= DATE\_SUB(CURDATE(), INTERVAL 30 DAY);

• Books Never Borrowed:

SELECT \* FROM Books WHERE BookID NOT IN (SELECT BookID FROM Loans);

Used techniques:

- GROUP BY, JOIN, HAVING

- Aggregate functions: COUNT, MAX, AVG

- Subqueries and DATE filters

------------------------------------------------------------

DATASET & FILE STRUCTURE

------------------------------------------------------------

| File Name | Purpose |

|----------------------------|---------------------------------|

| Library-DB-Schema-1.sql | Table creation |

| Library-DB-2.sql | Initial inserts |

| Library-DB-3.sql | Member and loan data |

| Library-DB-4.sql | Updates, deletes |

| task7.sql | Views created |

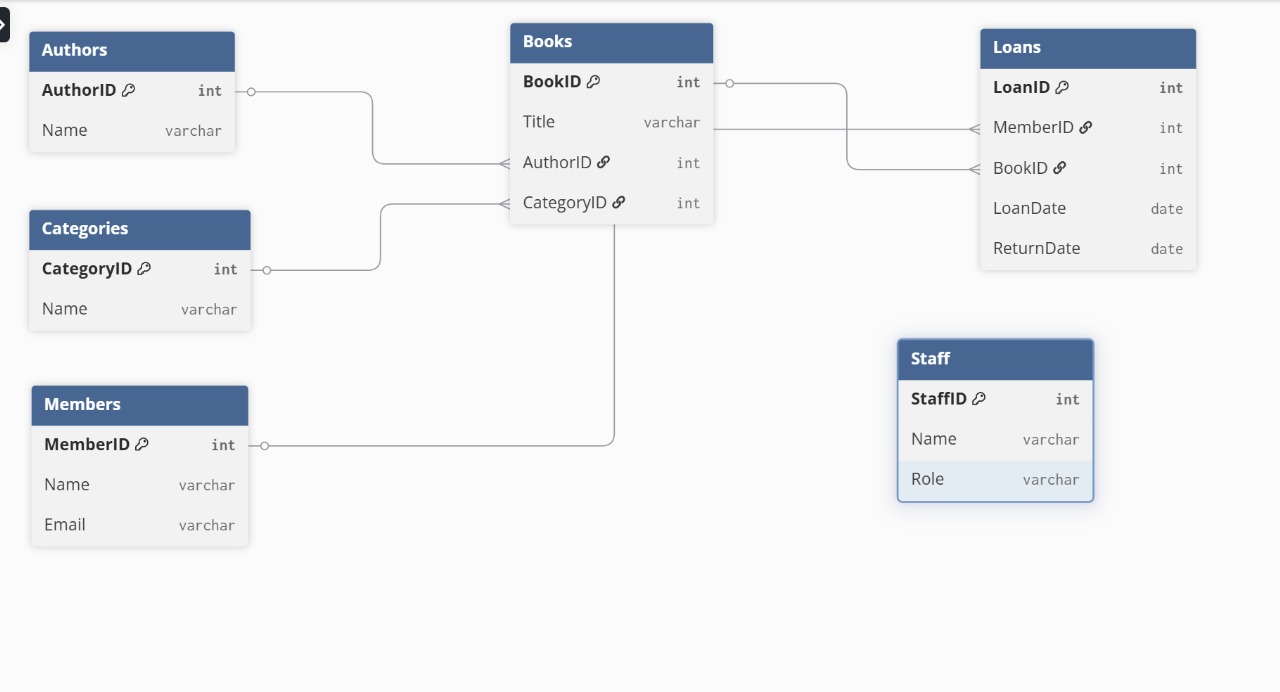
| README\_Task7.md | Task-specific documentation |

| ER\_Diagram.jpg / png | Visual schema |

------------------------------------------------------------

ER Diagram

------------------------------------------------------------



------------------------------------------------------------

CONCLUSION

------------------------------------------------------------

This project delivers a complete SQL-based library backend with:

- Normalized schema

- Consistent and realistic data

- Many-to-many handling

- Views for modular reporting

- Trigger-based automation

- Rich analytical reports

Prepared for real-world extension or integration into a full-stack app.

============================================================