Library Management System – DB Project Part 2

Repository Name: Library Management System - DB Project Part2

Goal: Elevate your database from functional to production-ready. In this phase, you'll improve performance, enable advanced reporting, implement automation, and ensure transactional integrity — just like a backend engineer in a real-world system.

➢ What's Required in Your Repository

Structure your GitHub repo with the following files:

File	Purpose
SELECT Queries	The script of queries
Indexes.sql	Index creation scripts to optimize query performance
Views.sql	Logical views for frontend data access
Functions.sql	User-defined functions for encapsulated logic
StoredProcedures.sql	Stored procedures for system automation
Triggers.sql	Triggers for enforcing business rules
Aggregations.sql	Basic real-world aggregation queries
AdvancedAggregations.sql	Advanced aggregation tasks using HAVING, subqueries
Transactions.sql	Transaction scripts for atomic operations
README.md	Brief description of each file, learning reflections (optional)

SELECT Queries – Think Like a Frontend API Imagine the following queries are API endpoints the frontend will call:

- GET /loans/overdue → List all overdue loans with member name, book title, due date
- GET /books/unavailable → List books not available
- GET /members/top-borrowers → Members who borrowed >2 books
- GET /books/:id/ratings → Show average rating per book
- GET /libraries/:id/genres → Count books by genre
- GET /members/inactive → List members with no loans
- GET /payments/summary → Total fine paid per member
- GET /reviews → Reviews with member and book info
- GET /books/popular → List top 3 books by number of times they were loaned
- GET /members/:id/history → Retrieve full loan history of a specific member including book title, loan & return dates

- GET /books/:id/reviews → Show all reviews for a book with member name and comments
- GET /libraries/:id/staff → List all staff working in a given library
- GET /books/price-range?min=5&max=15 → Show books whose prices fall within a given range
- GET /loans/active → List all currently active loans (not yet returned) with member and book info
- GET /members/with-fines → List members who have paid any fine
- GET /books/never-reviewed → List books that have never been reviewed
- GET /members/:id/loan-history →Show a member's loan history with book titles and loan status.
- GET /members/inactive →List all members who have never borrowed any book.
- GET /books/never-loaned → List books that were never loaned.
- GET /payments →List all payments with member name and book title.
- GET /loans/overdue→ List all overdue loans with member and book details.
- GET /books/:id/loan-count → Show how many times a book has been loaned.
- GET /members/:id/fines → Get total fines paid by a member across all loans.
- GET /libraries/:id/book-stats → Show count of available and unavailable books in a library.
- GET /reviews/top-rated → Return books with more than 5 reviews and average rating > 4.5.

1. Indexing Strategy – Performance Optimization

Apply indexes to speed up commonly-used queries:

Library Table

- Non-clustered on Name → Search by name
- Non-clustered on Location → Filter by location

Book Table

- Clustered on LibraryID, ISBN → Lookup by book in specific library
- Non-clustered on Genre → Filter by genre

Loan Table

- Non-clustered on MemberID → Loan history
- Non-clustered on Status → Filter by status
- Composite index on BookID, LoanDate, ReturnDate → Optimize overdue checks

2. Views – Frontend Integration Support

View Name	Description
ViewPopularBooks	Books with average rating > 4.5 + total loans
ViewMemberLoanSummary	Member loan count + total fines paid
ViewAvailableBooks	Available books grouped by genre, ordered by price
ViewLoanStatusSummary	Loan stats (issued, returned, overdue) per library
ViewPaymentOverview	Payment info with member, book, and status

3. Functions – Reusable Logic

Function	Purpose
GetBookAverageRating(BookID)	Returns average rating of a book
GetNextAvailableBook(Genre, Title, LibraryID)	Fetches the next available book
CalculateLibraryOccupancyRate(LibraryID)	Returns % of books currently issued
fn_GetMemberLoanCount	Return the total number of loans made by a given member.
fn_GetLateReturnDays	Return the number of late days for a loan (0 if not late).
fn_ListAvailableBooksByLibrary	Returns a table of available books from a specific library.
fn_GetTopRatedBooks	Returns books with average rating ≥ 4.5
fn_FormatMemberName	Returns the full name formatted as "LastName, FirstName"

- **Reflect:** Where would such functions be used in a frontend (e.g., member profile, book search, admin analytics)?

4. Stored Procedures – Backend Automation

Procedure	Purpose
sp_MarkBookUnavailable(BookID)	Updates availability after issuing
sp_UpdateLoanStatus()	Checks dates and updates loan statuses
sp_RankMembersByFines()	Ranks members by total fines paid

5. Triggers – Real-Time Business Logic

Trigger	Purpose
trg_UpdateBookAvailability	After new loan → set book to unavailable
trg_CalculateLibraryRevenue	After new payment → update library revenue
trg_LoanDateValidation	Prevents invalid return dates on insert

6. Aggregation Functions – Dashboard Reports

Tasks simulate admin dashboards:

- Total fines per member
- Most active libraries (by loan count)
- Avg book price per genre
- Top 3 most reviewed books
- Library revenue report
- Member activity summary (loan + fines)

7. Advanced Aggregations – Analytical Insight

Includes:

- HAVING for filtering aggregates
- Subqueries for complex logic (e.g., max price per genre)
- Occupancy rate calculations
- Members with loans but no fine
- Genres with high average ratings

8. Transactions – Ensuring Consistency

Real-world transactional flows:

- Borrowing a book (loan insert + update availability)
- Returning a book (update status, return date, availability)
- Registering a payment (with validation)
- Batch loan insert with rollback on failure

10. Developer Reflection (Optional in README.md)

Encourage trainees to reflect:

- What part was hardest and why?
- Which concept helped them think like a backend dev?
- How would they test this if it were a live web app?