**Group Project**

**Class: CS 470 – Section 001- Database System**

**Instructor: Dr. Anjum Razzaque**

**Date: 04/28/25**

**Members: Kinshuk Mangal, Derrick Eason, Timothy Layrea, Jin Zezhou**

**Contributions:**

**Kinshuk Mangal:**

* Contributed to the overall database design, ERD creation, SQL development (tables, views, triggers), and backend integration using Flask.
* Assisted with frontend development, connecting HTML, Bootstrap, and JavaScript to Python routes, and participated in system testing and deployment.

**Derrick Eason:**

* Supported database table creation and dummy data insertion for initial system setup.
* Assisted in functional testing of reservation processes and fine calculations.

**Timothy Layrea:**

* Contributed to the development of SQL views supporting different user roles.
* Participated in organizing external and internal view documentation for thesystem.

**Jin Zezhou:**

* Assisted with frontend page testing for members and librarians.
* Helped with documentation formatting and identifying system workflow improvements.

**Introduction**

### **Organization and Structure of the Library Database System**

Our database system mimics the functions and operations of a **routine library**, designed to efficiently manage books, track borrowing activities, handle transactions, and maintain user records for a university library. The system ensures that books can be borrowed, reserved, and returned in a structured manner while keeping track of due dates, fines, and book conditions. Different stakeholders (Library Members, Librarians, and Administrators) have defined roles and permissions to ensure smooth library operations.

The database consists of **seven key entities**, each serving a crucial function in managing library resources. The relationships between these entities define how books are borrowed, reserved, and maintained.

## **Description of Entities and Their Attributes**

### **LibraryBranches (Library Locations)**

* Represents different physical branches of the university library where books are stored.
* **Attributes**:
  + **BranchID (PK)** – Unique identifier for each branch.
  + **BranchName** – Name of the library branch.
  + **Address** – Location of the branch.
  + **ContactNumber** – Phone number of the branch.

**Relationship**:

* 1:M with Books (Each branch has multiple books).
* 1:M with Librarians (Each branch has multiple librarians).

### **Books (Library Collection)**

* Stores information about physical books and digital resources available in the library.
* **Attributes**:
  + **BookID (PK)** – Unique identifier for each book.
  + **Title** – Book title.
  + **ISBN** – Unique book number.
  + **Author** – Author(s) of the book.
  + **Publisher** – Name of the book publisher.
  + **PublicationYear** – Year the book was published.
  + **Genre** – Category of the book (e.g., Fiction, Science).
  + **BookType** – Format of the book (Physical, Ebook, Research Paper).
  + **ShelfLocation** – Physical shelf location (NULL for ebooks).
  + **BranchID (FK referring to BranchID in LibraryBranches)** – References the branch where the book is stored.
  + **TotalCopies** – Number of copies available.
  + **AvailableCopies** – Copies that can be borrowed.
  + **FileFormat** – Digital format (PDF, EPUB, etc.) for ebooks.
  + **AccessURL** – URL to access digital resources (NULL for physical books).

**Relationship**:

* M:N with LibraryMembers via Loans (A book can be borrowed multiple times).
* M:N with LibraryMembers via Transactions (A book can be reserved).
* M:1 with LibraryBranches (A book belongs to one branch).

### **LibraryMembers (Registered Users)**

* Represents students, faculty, or visitors who use the library.
* **Attributes:**
  + **MemberID (PK)** – Unique identifier for each member.
  + **FirstName, LastName** – Member’s name.
  + **DateOfBirth** – Ensures members are at least 18 years old.
  + **Email, Phone** – Contact details (must be unique).
  + **Address** – Member’s residential address.
  + **MembershipType** – Specifies if the member is a Student, Faculty, or Visitor.
  + **MembershipStartDate** – Auto-filled with registration date.
  + **OutstandingFines** – Keeps track of unpaid fines.
  + **CurrentReservations** – Number of books currently reserved.

**Relationship:**

* M:N with Books via Loans (A member can borrow multiple books).
* M:N with Books via Transactions (A member can reserve multiple books).

### **Loans (Tracking Borrowed Books)**

* Tracks which member borrowed which book, including due dates and book conditions.
* **Attributes:**
  + **LoanID (PK)** – Unique identifier for each loan.
  + **BookID ((FK) referring to BookID from Books)** – The book being borrowed.
  + **MemberID ((FK) referring to MemberID from LibraryMembers)** – The member borrowing the book.
  + **LoanDate** – Date when the book was borrowed.
  + **DueDate** – Date the book is due for return.
  + **ReturnDat**e – Date when the book was returned (NULL if not returned).
  + **Status** – Indicates Borrowed, Returned, Overdue.
  + **FineAmount** – Fine applied if overdue.
  + **ReturnCondition** – Indicates Good, Damaged, or Lost.
  + **DamageFee** – Fee charged if the book is damaged/lost.

**Relationship**:

* M:N with Books and LibraryMembers (Each loan links a book and a member).

### **Transactions (Reservations & Fine Payments)**

* Handles book reservations and fine payments made by members.
* **Attributes:**
  + **TransactionID (PK)** – Unique transaction identifier.
  + **MemberID ((FK) referring to MemberID from LibraryMembers)** – Member making the transaction.
  + **BookID ((FK) referring to BookID from Books)** – Book being reserved.
  + **TransactionType** – Specifies Reservation or Fine Payment.
  + **TransactionDate** – When the transaction was made.
  + **Amount** – The fine amount paid (if applicable).
  + **Status** – Indicates Pending, Approved, Completed, or Cancelled.
  + **LibrarianID ((FK) referring to LibrarianID from Librarians)** – References the librarian who processed the transaction.

**Relationship**:

* M:N with Books and LibraryMembers (Each transaction involves a book and a member).
* 1:M with Librarians (A librarian processes multiple transactions).

### **Librarians (Library Staff & Managers)**

* Represents staff responsible for managing books and transactions.
* **Attributes:**
  + **LibrarianID (PK)** – Unique identifier for each librarian.
  + **FirstName, LastName** – Librarian’s name.
  + **Email, Phone** – Contact details.
  + **Role** – Specifies if they are Admin, Staff, or Assistant.
  + **HireDate** – Date of employment.
  + **Salary** – Monthly salary.
  + **BranchID ((FK) referring to BranchID from LibraryBranches)** – The branch they work in.

**Relationship:**

* 1:M with LibraryBranches (Each branch has multiple librarians).
* 1:M with Transactions (Each transaction is processed by one librarian).

**AuditLog**

* Saves logs of all important activities in the DB according to the different users and their privileges.
* **Attributes:**

**LogID (PK):** Unique identifier for each log

**UserID ((FK) referring to LibrarianID/MemberID in Librarians/LibraryMembers):** Connects the log to the user

**UserRole:** Defines the User type

**ActionType**: Defines the action type for log records

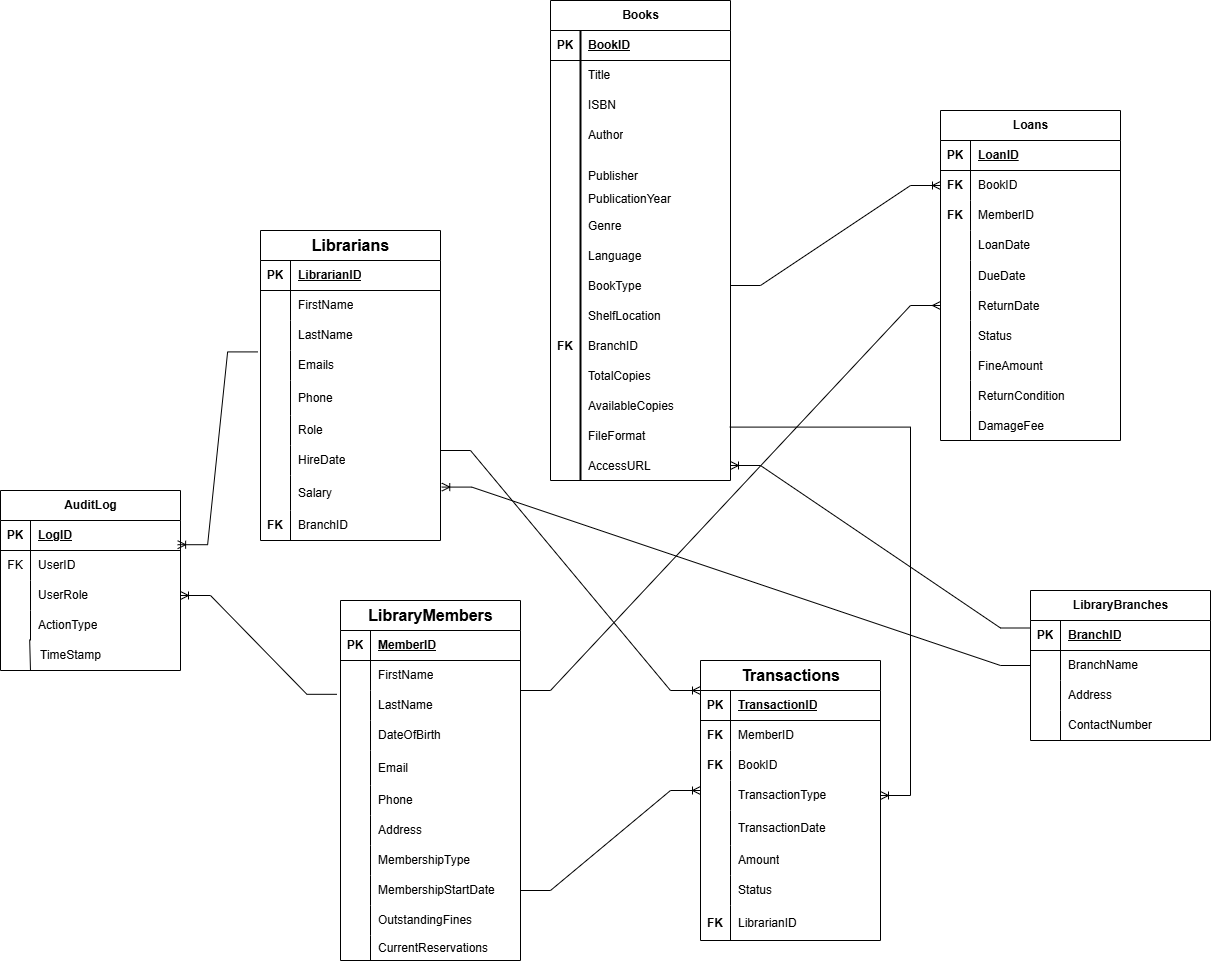
**TimeStamp**: Datetime of the recorded log

**Relationship:**

* One to many relation between LibraryMembers/Librarians as one person can have multiple audit logs, but 1 auditlog, has just 1 person on it.

## **ER Diagram (Entity-Relationship Diagram)**

Here is the ER diagram which visually represents the entities and relationships in our library database system:



## **Functionality of the DB based on the ERD**

Our library database is designed to efficiently store and manage physical and digital books, members, book loans, and transactions. This enables the customer to borrow, reserve, and return books while also applying fines if there are delays on return. The database allows Librarians the access to oversee loans, transactions, and book inventory ensuring operations with ease. Administrators have full access handling user roles, system modification, and reporting to maintain the integrity of the database.

1. **Conceptual View:**

* Represents the entire database structure at a high level, independent of any specific DBMS or underlying physical structure
* It focuses on entities, attributes, and relationships without details of physical storage

### **In our database, we include 7 entities:**

### Books

### LibraryMembers

### Loans

### Transactions

### Librarians

### LibraryBranches

* AuditLog

In our conceptual view, we talk about 2 main things:

* Authorization for different stakeholders
* Business rules/constraints

**Authorization levels**

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Access Level** | **Examples of Tasks** |
| Library Members | Read-Only | Can search books, view available copies, and their loan history. |
| Librarians | Read & Write | Can add/remove books, approve loans, and process fines |
| Administrators | Full Access | Can manage users, modify database schema, and monitor system logs. |

**Business rules/constraints**

In the library database system, authorization ensures that different stakeholders have different access levels based on their authorization level. The system follows Role-Based Access Control (RBAC), dividing users into **Library Members**, **Librarians**, and **Administrators**, each with distinct permissions. Library Members have **read-only access** to books and their own transactions, allowing them to search for books, make reservations, and update personal information, but they cannot modify book records or access other members' data. Librarians have **moderate access (read & write)**, enabling them to manage book inventory, process loans and returns, approve or reject reservations, and view member loan histories while being restricted from modifying the database schema. Administrators have **full control,** with permissions to modify system settings, manage users, alter table structures, and oversee transactions and reports.

Security measures such as authentication (requiring users to log in), row-level security (restricting members from accessing others' transactions), and audit logging (tracking key operations like book loans and fine payments) further enforce **data integrity**. These authorization policies ensure efficient, secure, and role-specific access within the library system while preventing unauthorized modifications or access to sensitive data.

**Here are the business rules/constraints for each entity in the DB:**

## **Books Table Constraints**

Each Book must have a unique identifier (BookID).

* Primary Key: Ensures no duplicate books exist.

Each Book must belong to exactly one Library Branch.

* Foreign Key (BranchID) links books to branches.

Total copies of a book must be at least 1.

* CHECK (TotalCopies >= 1)

Available copies cannot exceed total copies.

* CHECK (AvailableCopies >= 0 AND AvailableCopies <= TotalCopies)

Digital books (eBooks, Research Papers) must not have a Shelf Location.

* If BookType = 'Ebook', then ShelfLocation = NULL.

## **LibraryMembers Table Constraints**

Each Library Member must have a unique MemberID.

* Primary Key: Ensures every member has a distinct identifier.

Email and Phone Number must be unique.

* No duplicate email or phone numbers allowed.

Library Members can’t have a non-null date-of-birth.

* DateOfBirth DATE NOT NULL

Membership Type must be ‘Student’, ‘Faculty’, or ‘Visitor’.

Outstanding fines cannot be negative.

* CHECK (OutstandingFines >= 0)

A member cannot have more than 5 active reservations.

* CHECK (CurrentReservations <= 5)

Members with outstanding fines cannot borrow books.

* Before borrowing, OutstandingFines must be 0.

Phone numbers must start with + (international) or 0 (local).

* CHECK (Phone LIKE '+%' OR Phone LIKE '0%')

## **Loans Table Constraints**

Each Loan must be linked to exactly one Member and one Book.

* Foreign Keys (BookID, MemberID) enforce this.

A book cannot be borrowed if no copies are available.

* Before inserting a loan, AvailableCopies > 0.

A member cannot borrow more than 5 books at a time.

* Before inserting a loan, CurrentLoans <= 5.

A loan must have a Due Date later than the Loan Date.

* CHECK (DueDate > LoanDate)

Late returns result in a fine.

* FineAmount = DailyRate \* (ReturnDate - DueDate) (Trigger)

If a book is marked as “Lost,” a damage fee must be applied.

* CHECK (DamageFee > 0) IF ReturnCondition = 'Lost'

## **Transactions Table Constraints**

Each Transaction must be linked to a single Book and a single Member.

* Foreign Keys (BookID, MemberID).

A Librarian must approve each reservation or fine payment.

* Foreign Key (LibrarianID) ensures oversight.

Transaction Type must be ‘Reservation’ or ‘Fine Payment’.

* Uses ENUM.

## **Librarians Table Constraints**

Each Librarian must have a unique LibrarianID.

* Primary Key ensures uniqueness.

Each Librarian is assigned to exactly one Branch.

* Foreign Key (BranchID).

A Librarian can process multiple Transactions.

* Foreign Key (LibrarianID in Transactions).

## **LibraryBranches Table Constraints**

Each Library Branch must have a unique BranchID.

* Primary Key.

Each Branch must have at least one assigned Librarian.

* EXISTS (SELECT 1 FROM Librarians WHERE Librarians.BranchID = LibraryBranches.BranchID)

**AuditLog Table Constraints**

* Each AuditLog record must have a unique LogID.  
   (Primary Key constraint.)
* UserID must reference a valid MemberID or LibrarianID.  
   (Foreign Key constraint ensuring no orphan records.)
* UserRole must be one of: 'Member', 'Librarian', or 'Admin'.  
   (CHECK constraint to enforce valid user types.)
* ActionType must be limited to allowed operations such as 'Borrow', 'Return', 'Reserve', 'Login', and 'Fine Payment'.  
   (CHECK constraint on operation types.)
* TimeStamp must auto-record the current system time when a new log entry is created.  
   (DEFAULT CURRENT\_TIMESTAMP.)

### **Why These Constraints Matter**

These constraints are necessary because they ensure data integrity (preventing duplication and providing valid references), prevent incorrect transactions (e.g., borrowing unavailable books), enforce business rules (like maximum number of books borrowed at a time), and maintain referential integrity (making sure FKs enforce valid connections across other tables).

2) **External View**

The external view of a database represents how different users (stakeholders) interact with the system. **Instead of exposing the entire database**, external views create customized "virtual tables" (SQL Views) that **only show relevant data to specific user roles.**

#### **Importance of SQL views**:

They **prevent unauthorized access to sensitive data** **by limiting what each user can see**. Users only see the data relevant to their role, making it easier to work with. Views also protect original tables by allowing users to access data without modifying the underlying structure. Views filter out unnecessary columns or rows, improving query efficiency and data readability.

## **Stakeholder-Based Views for the Library Database**

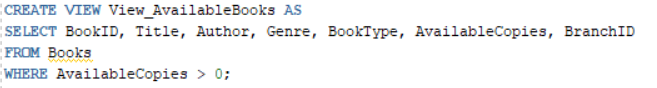
Each stakeholder (Library Member, Librarian, Administrator) has specific tasks, requiring different views to access only the data they need.

### **Views for Library Members (Read-only privilege)**

**Purpose**: Allow members to browse books, track their borrowings, reservations, and fines.

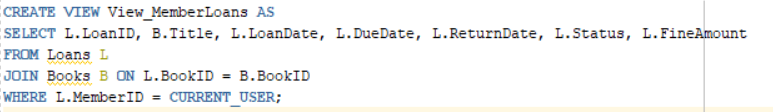
#### **View\_AvailableBooks**

* Shows only available books that members can borrow.
* Filters out books with AvailableCopies = 0 to avoid confusion.
* Includes book title, author, genre, and branch location.



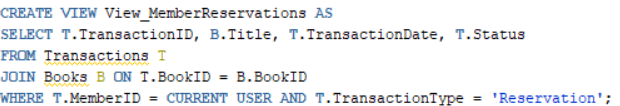
#### **View\_MemberLoans**

* Displays current and past borrowed books for the logged-in member.
* Shows due dates, return status, and any overdue fines.
* Filters records to show only books borrowed by the logged-in member.



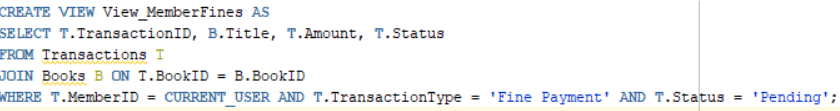
#### **View\_MemberReservations**

* Lists books the member has reserved and their status (Pending, Approved, Cancelled).
* Includes reservation date and expected availability.



#### **View\_MemberFines**

* Shows any outstanding fines the member needs to pay.
* Includes fine amount, due date, and payment status.

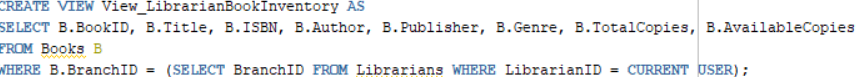


### **Views for Librarians (Read-Write Privilege)**

Purpose: Allow librarians to manage book loans, reservations, and transactions.

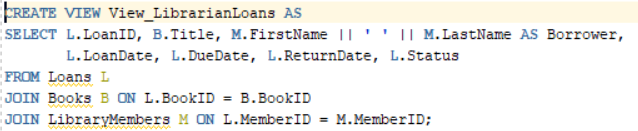
#### **View\_LibrarianBookInventory**

* Displays all books in the library, including branch location, total copies, and available copies.
* Helps librarians track book stock and restocking needs.



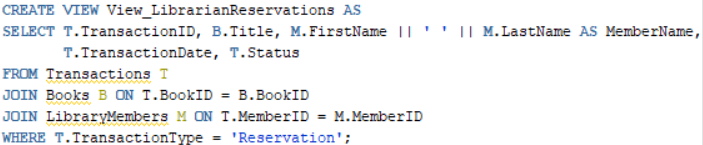
#### **View\_LibrarianLoans**

* Shows all currently borrowed books, the member who borrowed them, and due dates.
* Allows librarians to track overdue books.



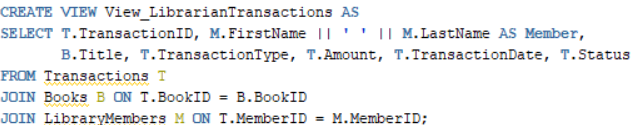
#### **View\_LibrarianReservations**

* Displays all book reservations with reservation status (Pending, Approved, Cancelled).
* Allows librarians to approve or reject reservations.



#### **View\_LibrarianTransactions**

* Shows all financial transactions handled by the librarian, including fine payments and reservation fees.

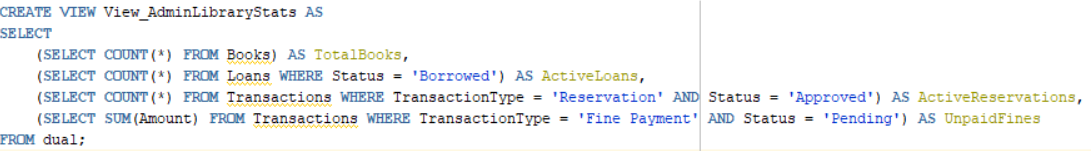


### **Views for Administrators (Full Access)**

Purpose: Provide administrators full system oversight, including user activity, book statistics, and fine reports.

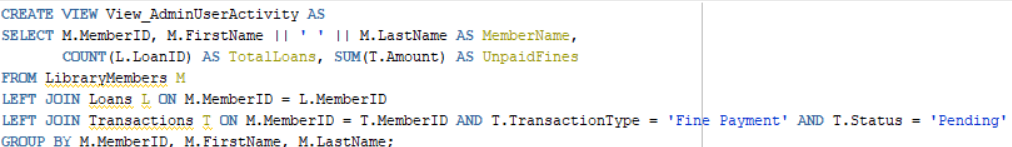
#### **View\_AdminLibraryStats**

* Provides a summary of total books, loans, reservations, and fines across all branches.
* Helps administrators monitor library usage and plan improvements.



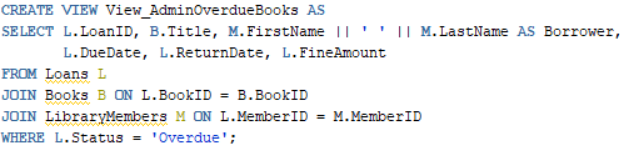
#### **View\_AdminUserActivity**

* Shows all library members and their borrowing history.
* Helps in analyzing user engagement and detecting inactive accounts.



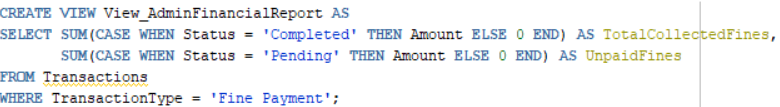
#### **View\_AdminOverdueBooks**

* Lists all overdue books and outstanding fines.
* Helps administrators track unreturned books and enforce penalties.



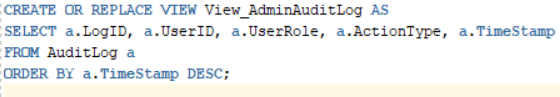
#### **View\_AdminFinancialReport**

* Generates a financial summary of fines collected and pending payments.
* Used for budgeting and financial planning.



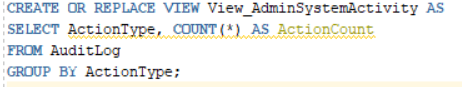
**View\_AdminAuditLog**

* Allows administrators to view all recorded user activities (borrowings, returns, logins, fine payments) from the AuditLog.
* Helps administrators monitor system activity and track user behavior historically.



**View\_AdminSystemActivity**

* Aggregates and summarizes system actions over time (e.g., number of borrows, returns, fines).
* Supports administrative reporting on system usage patterns and library engagement metrics.



3) **Internal View**

The Internal View of the library database focuses on how data is **physically stored, indexed, and optimized** for performance. This level is **hidden** from users and focuses on efficiency, indexing, **constraints**, and **metadata** that improve database performance.

**Key Components of the Internal View:**

### **Database Schema (Physical Representation of Tables)**

* Defines how data is stored in the DBMS, including tables, data types, and constraints.
* Ensures referential integrity using Primary Keys (PKs) and Foreign Keys (FKs).

### **Indexing (Performance Optimization**)

* Creates indexes on frequently queried columns to speed up searches (e.g., searching for books by title or author).
* Reduces query execution time.

### **Data Storage & Optimization**

* Determines how the DBMS physically stores tables, relationships, and constraints.
* Ensures efficient storage allocation for large datasets (e.g., book records, loan history).

### **Constraints & Referential Integrity**

* Enforces business rules at the DBMS level.
* Prevents invalid data entry (e.g., ensuring a book has at least one copy before being listed).

### **Data Dictionary (Metadata Documentation)**

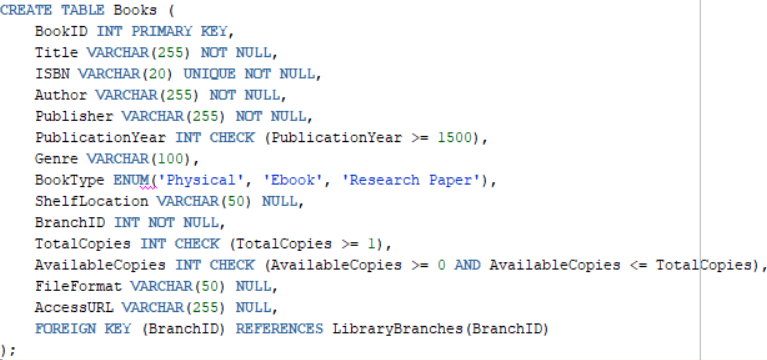
* Stores metadata about the database structure, including:
  + Table names
  + Column names & data types
  + Constraints
  + Relationships between tables
* Helps users and applications understand the database structure.

**Internal view pertaining to our Library DB:**

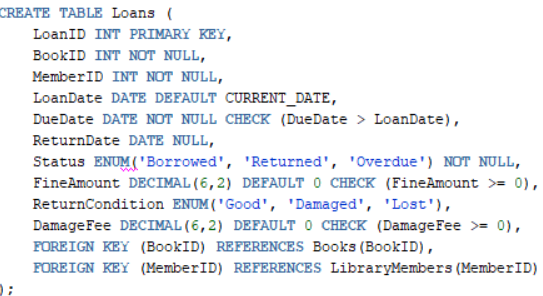
# **Database Schema (Physical Representation)**

This represents the actual database structure, including tables, attributes, data types, constraints, and keys.

### **Books Table**



### **Loans Table**



All other tables follow similar definitions.

# **Indexing (Performance Optimization)**

Indexes help speed up searches, sorting, and filtering by reducing the number of scanned rows.

### **Essential Indexes:**

**Index for Fast Book Searches by Title**



**Index for Searching Books by Author**



**Index for Looking Up Loans by MemberID**



**Index for Fast Transactions Lookups**



**Index for Fast Retrieval of Overdue Books**



# **Data Storage & Optimization**

* Data is stored in tables and indexed to optimize queries.
* DBMS handles physical storage automatically, but partitioning and indexing strategies can improve performance.
* Tables with large records (Loans, Transactions) should be indexed for quick lookups.

**Storage Considerations:**  
Frequent Queries: Books and Loans are queried the most → Indexing improves speed.  
Historical Data: Old Transactions & Loans may be archived to reduce database load.

# **Constraints & Referential Integrity**

To enforce business rules and prevent invalid data, constraints are applied to tables.

### **Examples of Essential Constraints:**

**Ensure Books Have at Least One Copy**



**Loans Must Have a Valid Due Date**



**Fine Amount Cannot Be Negative**



**Foreign Key to Ensure Transactions Are Linked to Existing Members & Books**



**Data Dictionary (Metadata Documentation)**

A data dictionary contains metadata about the database structure.

### **Examples of Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Column Name** | **Data Type** | **Constraint & Description** |
| Books | BookID | INT | PRIMARY KEY, Unique Identifier for each book |
| Books | Title | VARCHAR(255) | NOT NULL, tile of the book |
| Books | ISBN | VARCHAR(20) | UNIQUE, NOT NULL, International identifier of book |
| LibraryMembers | MemberID | INT | PRIMARY KEY, Unique identifier for each member |
| LibraryMembers | FirstName | VARCHAR(100) | NOT NULL, member’s first name |
| LibraryMembers | LastName | VARCHAR(100) | NOT NULL, member’s last name |
| Loans | BookID | INT | FOREIGN KEY REFERENCES Books(BookID), Loaned book |
| Loans | MemberID | INT | FOREIGN KEY REFERENCES LibraryMembers(MemberID), Borrowing member |
| Loans | DueDate | DATE | CHECK (DueDate > LoanDate), Expected return date |
| Transactions | Amount | DECIMAL (6,2) | CHECK (>=0), Amount for fines or reservations |
| Transactions | Status | ENUM('Pending', 'Completed', 'Cancelled') | Transaction Status |
| Transactions | LibrarianID | INT | FOREIGN KEY REFERENCES Librarians(LibrarianID), Processed by |
| Librarians | HireDate | DATE | DEFAULT CURRENT\_DATE, Date of employment |
| Librarians | Salary | DECIMAL (8,2) | CHECK (>=0), Librarian salary |
| Librarians | BranchID | INT | FOREIGN KEY REFERENCES LibraryBranches(BranchID), Work location |
| LibraryBranches | BranchName | VARCHAR (255) | NOT NULL, Name of library branch |
| LibraryBranches | Address | VARCHAR (255) | NOT NULL, Branch location |
| AuditLog | LogID | NUMBER | Primary Key. Unique identifier for each audit log record. |
| AuditLog | UserID | NUMBER | Foreign Key. References MemberID in LibraryMembers or LibrarianID in Librarians. Must exist. |
| AuditLog | UserRole | VARCHAR(20) | CHECK constraint. Must be 'Member', 'Librarian', or 'Admin'. |
| AuditLog | ActionType | VARCHAR(20) | CHECK constraint. Must be one of: 'Borrow', 'Return', 'Reserve', 'Login', 'Fine Payment', 'Add Book', 'Delete Member', 'Modify Book'. |
| AuditLog | TimeStamp | TIMESTAMP | Automatically records the date and time when the log entry is created (default CURRENT\_TIMESTAMP). |

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| Table | Books | Stores book details including author, genre, and availability |
| Table | LibraryMembers | Contains information about registered library users |
| Table | Loans | Tracks borrowed books and their due dates." |
| View | View\_AvailableBooks | Shows books available for borrowing |
| View | View\_MemberLoans | Displays books borrowed by a specific member |
| View | View\_MemberFines | Displays outstanding fines of a member. |
| Index | idx\_books\_title | Dense, Clustered - Speeds up book searches by title |
| Index | idx\_books\_author | Dense, Non-Clustered - Optimizes searches for books by author |
| Index | idx\_loans\_member | Sparse, Clustered - Quickens lookup of member loans. |

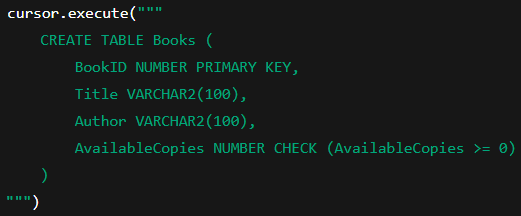
### **How the Data Dictionary Helps:**

* Documents tables, attributes, and constraints.
* Useful for developers, DBAs, and applications.
* Prevents errors by defining relationships and rules.

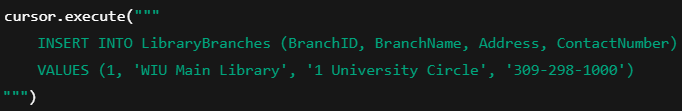
**Implementation in SQL (Examples):**

In our project, we integrated SQL database management directly into Python applications by executing SQL commands through Python scripts using the oracledb module. This allowed dynamic database creation, data manipulation, and system testing in an automated environment.

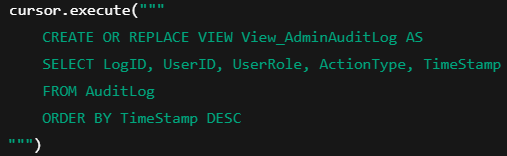
* To create tables, we embedded SQL Data Definition Language (DDL) commands inside Python and executed them through a cursor object. For example, the Books table was created in Python as follows:



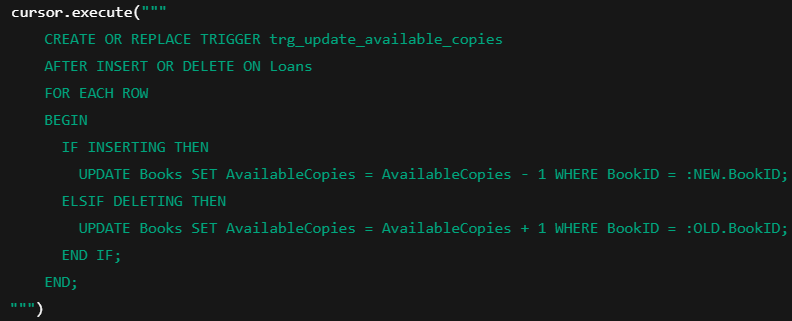
* For data insertion, we used SQL INSERT statements embedded in Python to populate tables with dummy values. For instance, adding a new library branch dynamically was done using:



* Views were also programmatically created and tested through Python scripts. For example, the administrator audit view was created using:



* Finally, we utilized Python to deploy and verify database triggers, ensuring automatic enforcement of business rules. For example, the trigger trg\_update\_available\_copies, which updates the number of available copies when a book is borrowed or returned, was created and validated through Python:



**Integration of Backend and Frontend Using Flask, HTML, Bootstrap, and JavaScript:**

In this project, we successfully integrated the backend database operations with a dynamic frontend interface by leveraging Flask, Python .py files, HTML templates, Bootstrap, and JavaScript. Flask served as the web framework to handle HTTP requests and responses, connect to the Oracle database through Python, and route users to different pages based on their actions. Each major functionality, such as logging in, viewing available books, reserving books, and accessing administrator dashboards, was mapped to specific .py routes in Flask applications (e.g., member.py, librarian.py, admin.py).

The frontend was built using HTML files structured as Flask templates, where placeholders (using Jinja2 templating syntax like {{ variable }}) dynamically displayed data retrieved from the database. Bootstrap was incorporated to ensure responsive and professional styling for pages such as the login screen, dashboards, book listings, and system management pages. It provided ready-to-use components like navigation bars, cards, buttons, and modals, ensuring both usability and visual appeal.

JavaScript was used selectively to enhance user interactions on the frontend. For instance, JavaScript handled real-time QR code scanning, form validations, and dynamic content updates without requiring full page reloads. Custom JavaScript scripts worked alongside Bootstrap components to provide interactivity such as flashing notification alerts, live updating of book inventories, and dynamic page redirection after certain actions like reservations or returns.

Flask acted as the glue between the frontend and backend by binding HTML templates to Python functions that processed SQL queries and returned results to users in real time. This architecture allowed the system to maintain role-based access control (different dashboards for members, librarians, and administrators) while keeping the application modular and scalable. Through the combination of Flask, HTML, Bootstrap, JavaScript, and SQL integration in Python, we successfully built a full-stack, web-based library management system that supports efficient operations, user engagement, and system transparency.

**Challenges Encountered:**

* Backend–Database Communication: Managing database transactions reliably through Python Flask, especially during concurrent actions like borrowing and reserving books.
* Session Management Across Roles: Maintaining distinct member, librarian, and admin sessions while enforcing role-based access control through Flask routing and decorators.
* Real-Time Features Integration: Implementing live QR code scanning using JavaScript while ensuring browser compatibility and secure data transfer to Flask routes.
* Efficient Activity Logging: Designing the AuditLog structure to record all critical user activities without degrading database performance.
* Debugging and Modularization: Troubleshooting across multiple layers (frontend, backend, database) and maintaining a modular, scalable codebase.

**Future Work and Research Opportunities:**

* Real-Time Notifications: Implement WebSocket (e.g., Flask-SocketIO) to push instant updates for reservations, fines, and system messages without page reloads.
* Enhanced Authentication: Introduce two-factor authentication (2FA) for administrators and librarians to further secure access.
* Dynamic Frontend Development: Upgrade from static HTML/Bootstrap to dynamic frameworks like ReactJS or VueJS for more responsive dashboards.
* Advanced User Behavior Analytics: Expand the AuditLog to capture device, location, and browser data for future predictive analytics and usage research.
* Cloud Deployment and Scalability: Prepare the system for cloud hosting (AWS, Azure) and automate database backups and failover support to make it production-ready.

## 