Library Management System

Database Design Report

Group Members:

1.Abebayehu Gethane	2023006209
2. Beamlaku Berehanu	2023006271
3. Daniel Negasa	2023006224
4. Enkopazion Teshome	2023006220
5. Kirubel Mamo	2023006219
6. Nasis Gurmu	2023006231

Course Code: CSCI 326
Instructor: Dr. Zubaidah Alhazza
Submission Date: November 29, 2025

Department of Computer Science and Engineering

American University of Ras Al Khaimah

United Arab Emirates

1 Abstract

In this project, we followed a complete and professional database design methodology that began at the conceptual level and progressed through logical and physical implementation stages. We first constructed a comprehensive Entity–Relationship (ER) model consisting of 22 core entities, capturing the functional domains of user management, cataloging, circulation, facility reservations, donations, and vendor operations. This conceptual model allowed us to understand the complex relationships in the library environment, including many-to-many interactions such as `Writes` (Author–Book) and `Donates` (Donation–Book)

Building on this foundation, we performed a rigorous logical design process that involved mapping the ER model to relational tables and applying systematic normalization. The initial tables exhibited critical issues such as multivalued attributes, transitive dependencies, and partial dependencies for example, multiple phone numbers stored inside `Staff`, `Vendor`, and `Publisher` tables and transitive dependencies like `PageCount → Price` and `RoomType → Capacity`. Through 1NF, 2NF, and 3NF transformations, we decomposed these tables into clean structures such as `StaffPhoneNumber`, `Price`, and `RoomCapacity`.

We then produced a fully normalized 3NF schema containing over 32 well-defined tables, These tables eliminated redundancy, improved referential integrity, and ensured stable support for expansion as demonstrated in the final table list. Each entity was assigned meaningful keys, strict foreign-key constraints, and domain-specific attributes, resulting in a schema capable of supporting high-traffic operations such as catalog updates, circulation workflows, overdue fine processing, and room reservation management.

Finally, we implemented the physical schema using MySQL, completing the process with table creation scripts and realistic sample data insertions that populated categories, books, authors, members, book copies, loans, and reservations. The resulting system is a fully functional and scalable library database that aligns with industry-standard relational design practices and provides a strong foundation for future software integration.

Contents

2 Introduction

Modern academic libraries require data systems that can manage a wide range of coordinated operations, and in this project, we designed a database intended to serve as the backbone for such an environment. We approached the system from both user and administrative perspectives, allowing the database to support staff activities, member services, inventory management, and operational monitoring. By centralizing identities through the `UserReg` table and then extending role-specific data into `Staff` and `Member`, we created a consistent user-management framework that ensures secure authentication and streamlined access control.

We also incorporated a facility-management module to support student and staff use of the library's physical spaces. Entities such as `Room`, `Location`, and `RoomReservation` work together to manage study rooms and open areas. These tables, built from the conceptual diagrams and refined through normalization procedures, ensure that reservations are conflict-free and that capacity rules are consistently applied across different room types.

In addition, the system integrates financial and operational tracking mechanisms through `Loan`, `Fine`, `Payment`, `Notification`, and `LogEntry`. These elements allow staff to monitor overdue items, issue fines, record payments, send automated notifications, and audit interactions across the system. By addressing the needs of users, staff, and administrators, the database provides a unified framework that enhances the library's efficiency and overall service quality.

Problem Statement

Libraries today face several operational challenges caused by fragmented data, manual record-keeping, and poorly structured databases. One of the main problems is the lack of a unified source of truth for user management. Without a centralized system, staff records, member profiles, and account credentials become scattered across different files or redundant tables, making authentication and role management inconsistent and prone to human error. Our analysis showed that multi-valued and duplicated contact information—such as staff phone numbers, vendor details, and publisher emails was embedded inside single records across several tables, leading to data update anomalies and inaccurate reporting.

Another major problem arises in cataloging and physical inventory control. Libraries must manage multiple copies of books, track their condition, and record their exact location. However, without the separation between conceptual book data and physical book copies, the system cannot distinguish between bibliographic information and real inventory. This limitation makes it impossible to check which specific copies are available, on loan, damaged, or missing. It also prevents accurate shelf-level tracking, since `Location` and `Shelf` were originally mixed and created confusion about where items were stored.

In the area of circulation and financial processing, the absence of normalized loan, fine, and payment structures leads to unreliable transaction histories. When fines are not tied directly to the specific

loan that caused them, libraries cannot verify overdue charges or appeal decisions. Similarly, without a traceable payment system linked to fines, there is no audit trail, which increases financial risk. Furthermore, facilities such as study rooms often suffer from double booking, inconsistent capacity rules, and lack of reservation tracking.

Overall, the initial system lacked the structure needed to support modern library services such as automated notifications, accurate log entries, reservation management, and advanced reporting. The scattered, inconsistent, and unnormalized data created operational delays, reporting errors, missing audit trails, and confusion among library staff and patrons. These real problems demonstrated the need for a fully normalized, centralized, and logically designed database system.

Solution and Unique Contributions

The completed database system directly solves these operational challenges by providing a clean, normalized, and fully relational architecture designed specifically for real library workflows. First, we eliminated user-management inconsistencies by centralizing all authentication data in the `UserReg` table and then extending user roles through linked `Staff` and `Member` tables. This ensures that every user has one verified identity, while their role-specific data remains organized and accurate. Separating contact information into tables like `StaffPhoneNumber` and `VendorPhoneNumber` resolves update anomalies and guarantees consistency.

To solve cataloging and physical inventory issues, we implemented a two-level structure: `Book` stores bibliographic data, while `BookCopy` manages each physical copy with unique barcodes, conditions, acquisition history, and availability status. This structure allows librarians to instantly identify which specific copy is missing, overdue, or damaged, and where it is located in the library. By normalizing `Shelf`, `Location`, and `FloorLevel`, the system provides accurate, shelf-level navigation of library holdings.

For circulation and financial accuracy, the database links `Loan`, `Fine`, and `Payment` through strict foreign-key constraints. Every fine is tied to a specific loan, and every payment is tied to a specific fine, creating a complete audit trail. This solves real problems such as disputed fines, missing payment histories, or incorrect overdue calculations. The system also enforces logical workflows that prevent inconsistent states, such as paying for a fine that does not exist or returning a book that was never borrowed.

Finally, the database introduces powerful new capabilities that were previously impossible: automated user notifications, room reservation tracking, system activity logging, and structured reporting. Features such as the `Notification` table allow the library to send due-date reminders and reservation alerts. The `RoomReservation` entity prevents double bookings and enforces capacity rules. The `LogEntry` and `Report` tables support security auditing and administrative decision-making. Together, these improvements create a modern, reliable, and scalable library system that directly solves real operational problems and supports future expansion.

Report Overview

This report documents the complete development lifecycle of the Library Management System, encompassing the following deliverables:

1. **Conceptual Design and ER Modeling**
    - High-level Entity-Relationship (ER) Diagram providing system architecture overview
    - Detailed subsystem ER fragments for User Management, Cataloging, Circulation, Facility Management.
2. **Logical Database Design**
    - Comprehensive relational schema with 24 normalized tables
    - Normalization process documentation confirming Third Normal Form (3NF) compliance
3. **Data Dictionary**
    - Metadata documentation for core entities (Loan, Book, Member) including data types, keys, and constraints
4. **Physical Implementation (MySQL)**
    - Complete DDL scripts with `CREATE TABLE` statements for all entities
    - Integrity constraints: primary keys, foreign keys with cascading, `ENUM` types, and `NOT NULL` constraints
5. **Operational Testing**
    - Sample SQL queries demonstrating borrowing, fine calculation, and administrative reporting

3 Conceptual Design

3.1 Entity Relation Diagram

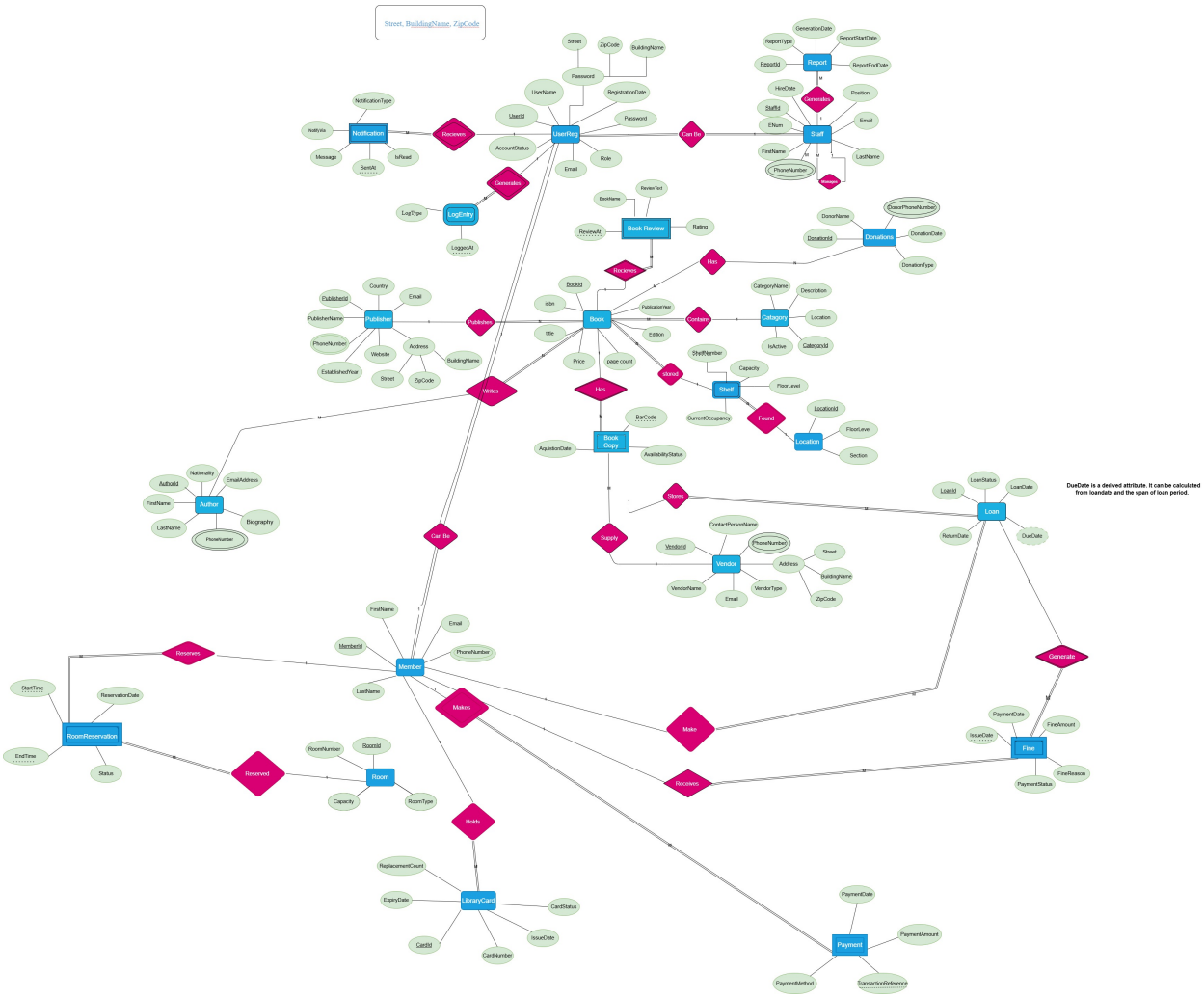


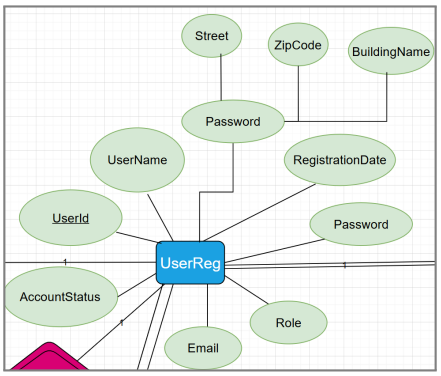
Figure 1: Complete ER Diagram

Description of ER diagram

This Entity-Relationship Diagram presents a complete data model consisting of 22 entities, illustrating how each component of the system interacts through well-defined relationships. The diagram outlines every entity along with its corresponding connections, clearly showing how data flows across the system by specifying the participation constraints whether an entity's involvement in a relationship is total or partial and the cardinalities that indicate the numerical nature of each association, such as one-to-one, one-to-many, or many-to-many. Together, these elements provide a structured and coherent view of how users, library resources, operational processes, and supporting components relate to one another within the system. We will now move to each entity's description and each relation's analysis.

3.2 Entites and Their Attributes

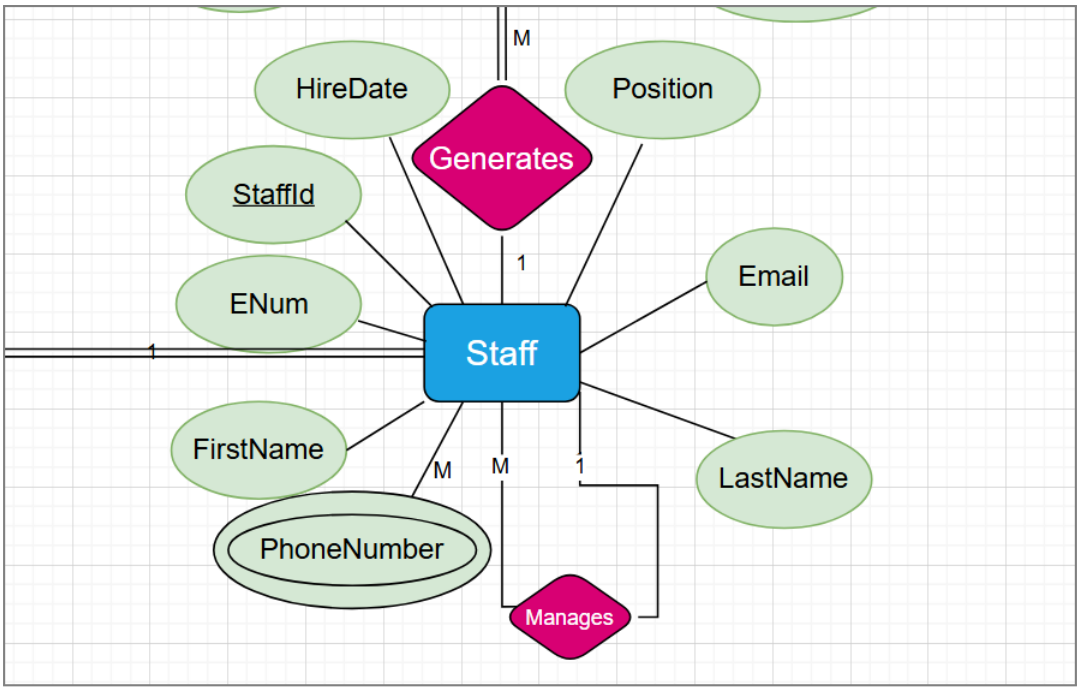
1. **UserReg**



The UserReg entity stores core registration and authentication information for every user of the library system. It is the central identity record that underpins login, authorization, and links to member or staff roles.

Attribute	Description
UserId (PK)	Unique system-generated identifier for each registered user.
Username	Login name chosen by the user and used during authentication.
Email	Primary contact email address for communication and verification.
Password	Securely stored (hashed) password used to log into the system.
RegistrationDate	Date on which the user account was created in the system.
AccountStatus	Current state of the account (e.g., active, suspended, closed).
Role	System role associated with the user (member, staff, admin, etc.).
Address	User's stree, zipcode and building information.

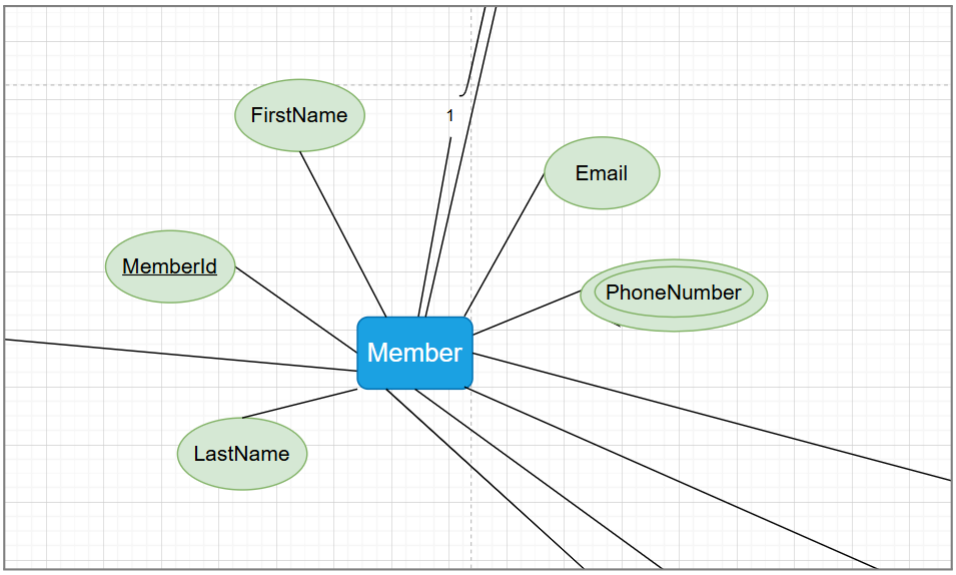
2. **Staff**



The Staff entity represents employees who work in the library, including librarians, assistants, and managers. It extends the basic user record with employment-related information and operational roles.

Attribute	Description
StaffId (PK)	Unique identifier assigned to each staff member.
ENum	Internal staff or employee number used for HR or payroll tracking.
PhoneNumber	Contact phone number for the staff member.
Email	Official work email address of the staff member.
FirstName	Staff member's given or first name.
LastName	Staff member's family or last name.
Position	Staff role in the library (e.g., librarian, assistant, manager).
HireDate	Date the staff member started working at the library.

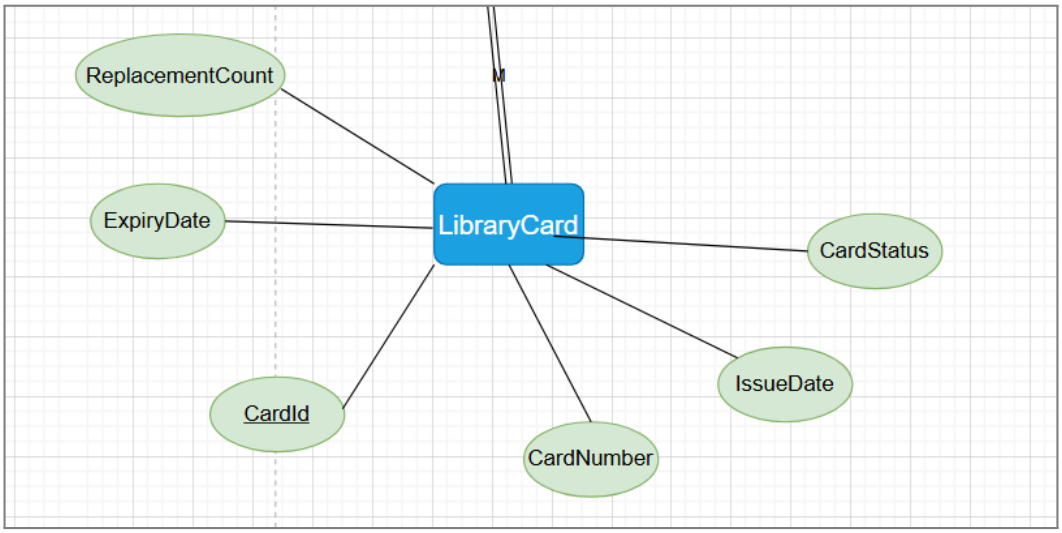
3. **Member**



The Member entity represents library patrons who borrow books and use library services. It holds personal contact details and links a person to their user account and library card.

Attribute	Description
MemberId (PK)	Unique identifier for each registered library member.
FirstName	Member's given or first name.
LastName	Member's family or last name.
Email	Member's primary email address for communication.
PhoneNumber	Member's contact phone number.

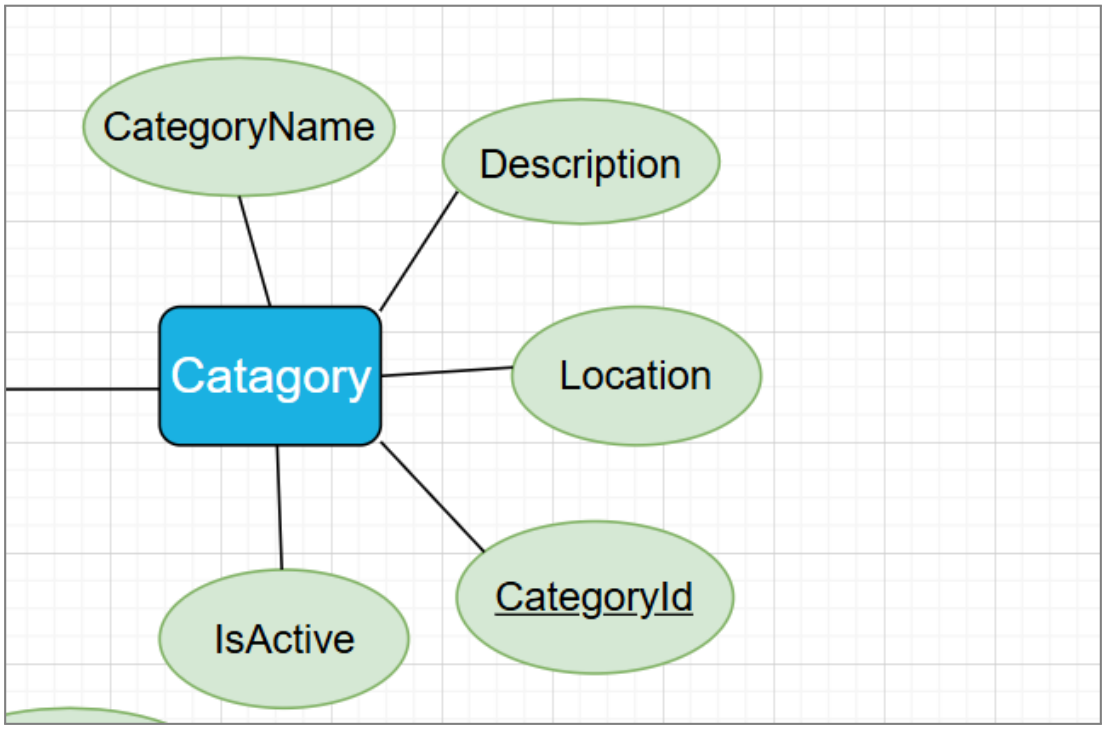
4. **LibraryCard**



The LibraryCard entity captures information about the card issued to a member, which is used to identify them during borrowing, access to services, or physical entry to certain areas.

Attribute	Description
CardId (PK)	Unique identifier for each library card.
CardNumber	The unique number used to identify the card.
IssueDate	Date on which the card was issued to the member.
ExpiryDate	Date when the card will expire and require renewal.
CardStatus	Current card state (e.g., active, expired, lost, suspended).
ReplacementCount	Number of times the card has been reissued or replaced.

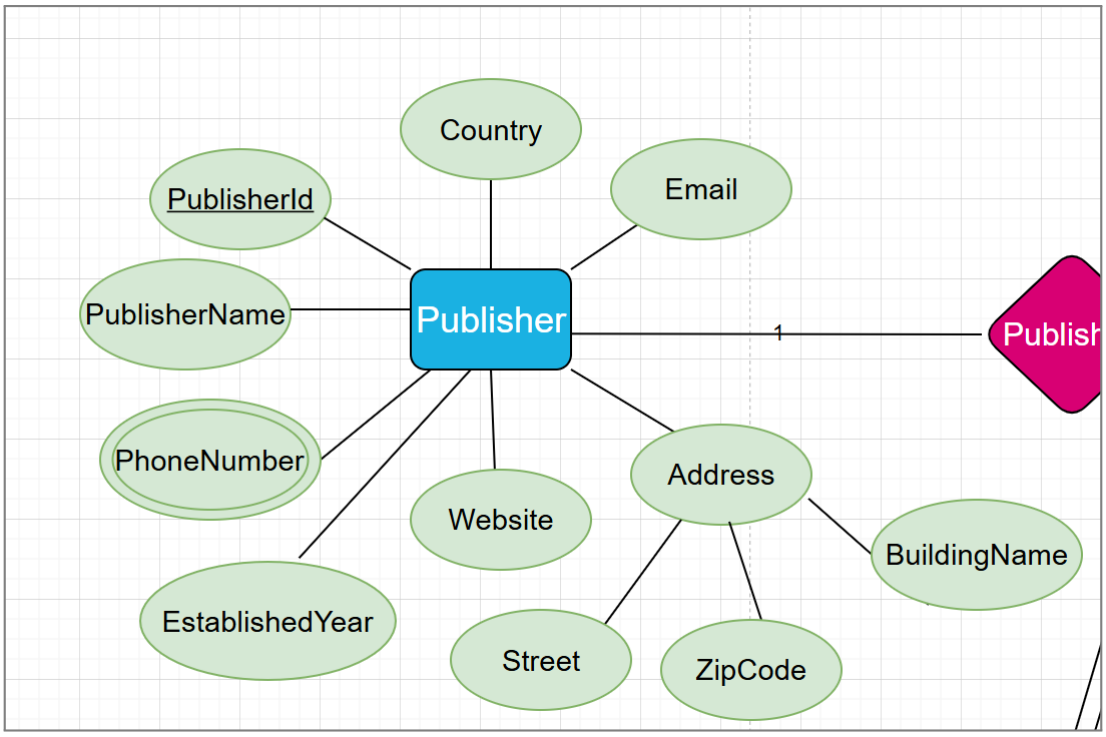
5. **Category**



The Category entity organizes books into logical groupings such as subject areas, genres, or collections. It helps users browse and locate materials more efficiently in the catalog and on shelves.

Attribute	Description
CategoryId (PK)	Unique identifier for each book category.
CategoryName	Descriptive name of the category (e.g., Fiction, Science).
Description	Short explanation of what the category covers.
Location	General physical area or section where this category is shelved.
IsActive	Flag indicating whether this category is currently in use.

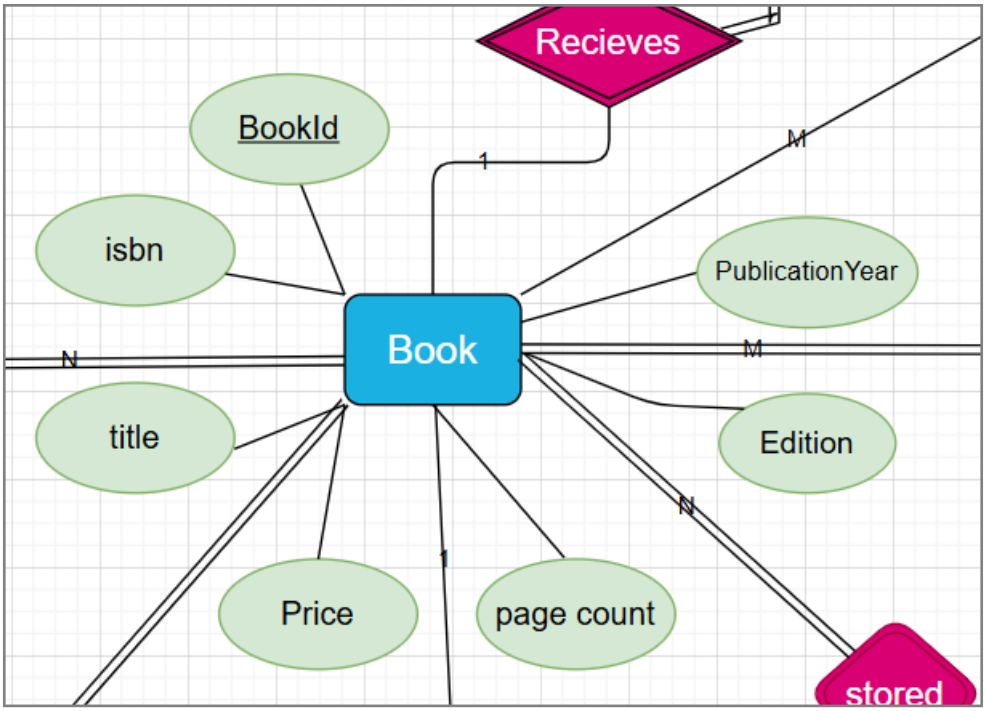
6. **Publisher**



The Publisher entity stores details about publishing houses and organizations responsible for producing books. It supports cataloging and acquisition processes by centralizing publisher information.

Attribute	Description
PublisherId (PK)	Unique identifier for each publisher.
PublisherName	Official name of the publishing company.
Country	Country where the publisher is based.
EmailAddress	General contact email address for the publisher.
PhoneNumber	Main contact telephone number for the publisher. It is multivalue attribute
Website	URL of the publisher's website.
EstablishedYear	Year in which the publisher was founded.
Address	street, zipcode and building

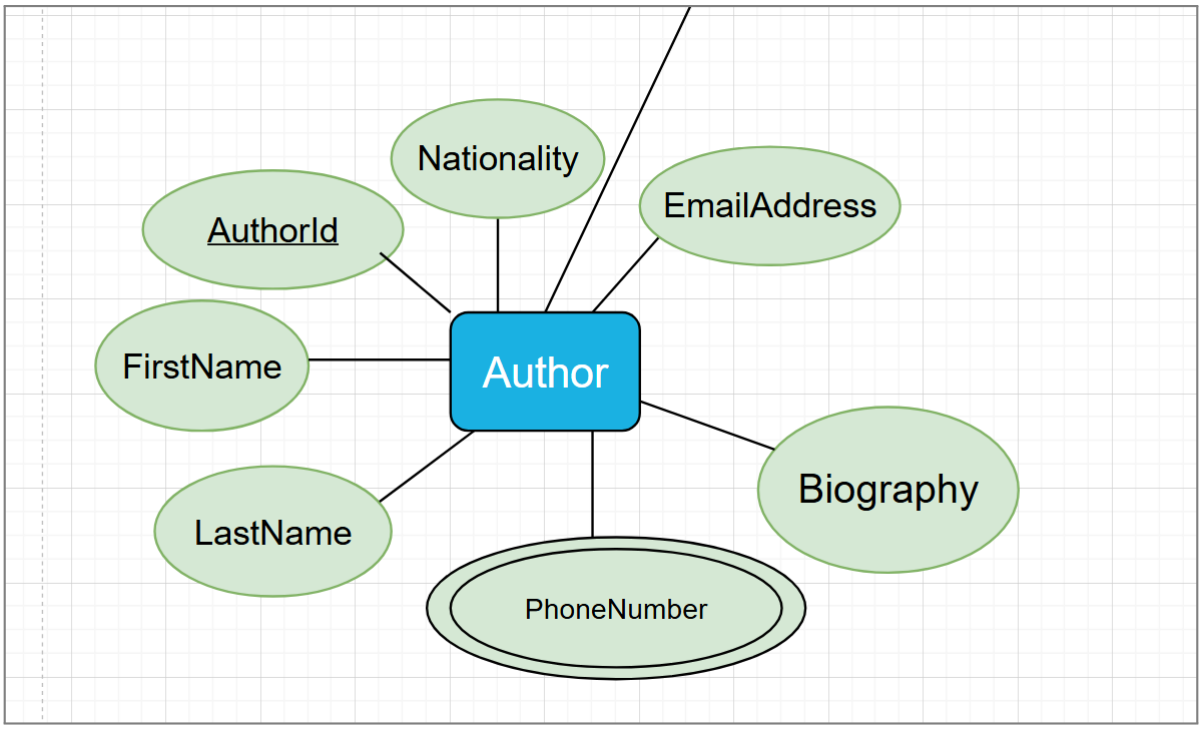
7. **Book**



The Book entity represents a logical book title in the catalog, independent of physical copies. It includes bibliographic details such as ISBN, title, publication year, and links to category and publisher information.

Attribute	Description
BookId (PK)	Internal identifier for each book record in the catalog.
ISBN	International Standard Book Number uniquely identifying the title/edition.
Title	Title or name of the book.
PublicationYear	Year in which this edition of the book was published.
Edition	Edition information (e.g., 1st, 2nd, revised).
PageCount	Total number of pages in the book.
Price	Acquisition or listed price of the book.

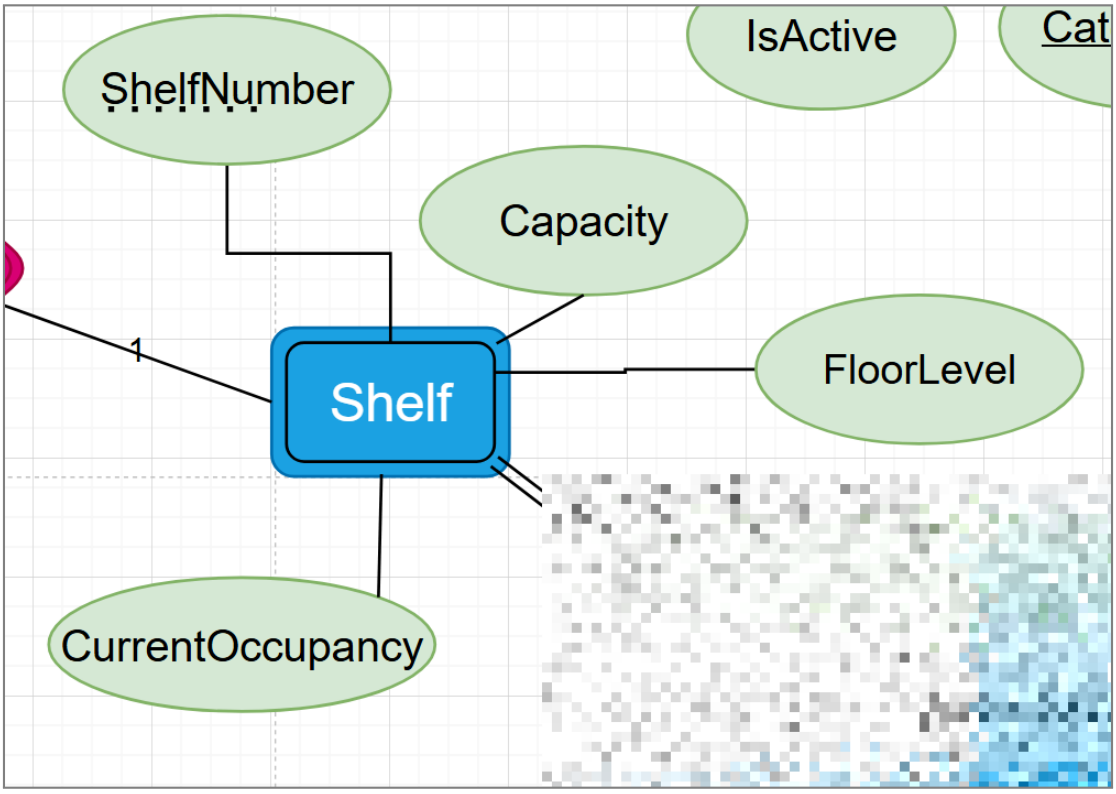
8. **Author**



The Author entity represents writers or contributors to books. It stores personal and biographical information that can be reused across many book titles.

Attribute	Description
AuthorId (PK)	Unique identifier for each author.
FirstName	Author's given or first name.
LastName	Author's family or last name.
Nationality	Country or nationality associated with the author.
Biography	Short biographical note or description of the author.
EmailAddress	Contact email address for the author, if available.
PhoneNumber	Contact phone number for the author, if stored.It is multivalue attribute

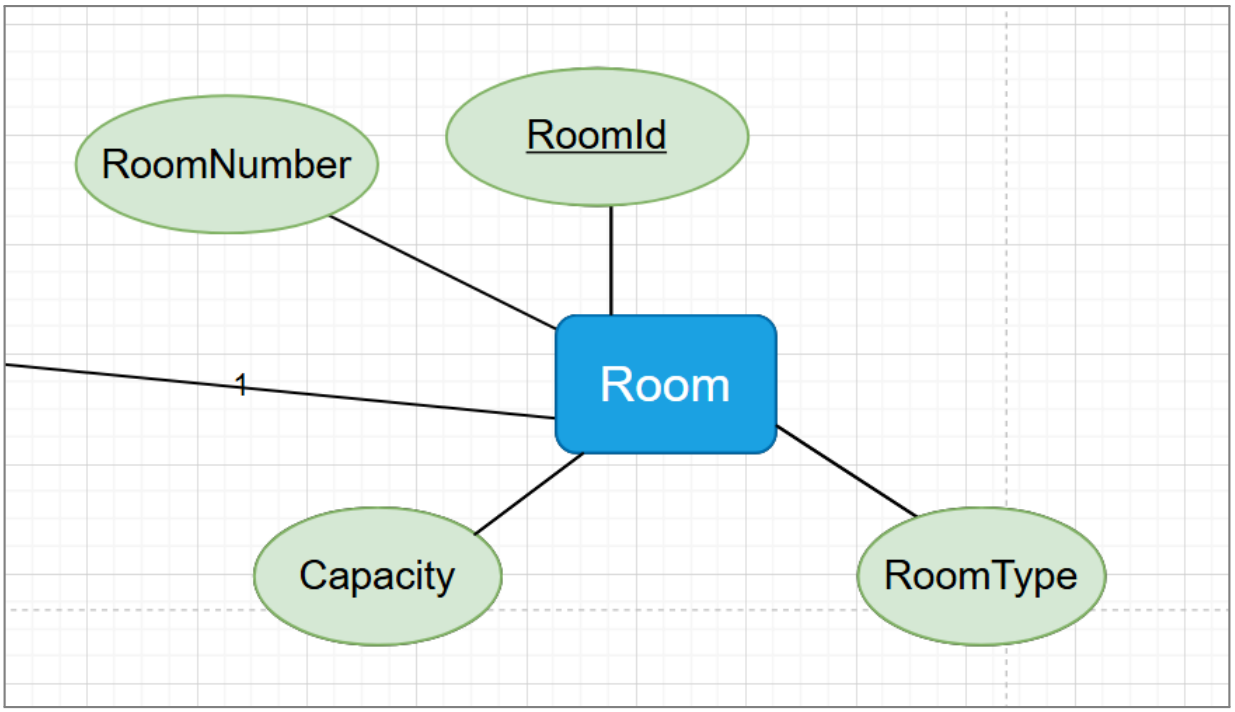
9. **Shelf (weak entity)**



The Shelf entity is a weak entity that depends on the Location entity. It represents a single shelf within a specific location and tracks its capacity and current occupancy.

Attribute	Description
LocationId, ShelfNumber (Composite PK)	Composite key that uniquely identifies a shelf within a location.
Capacity	Maximum number of items or books the shelf can hold.
FloorLevel	Floor level of the building where the shelf is located.
current_occupancy	Current number of items placed on the shelf.

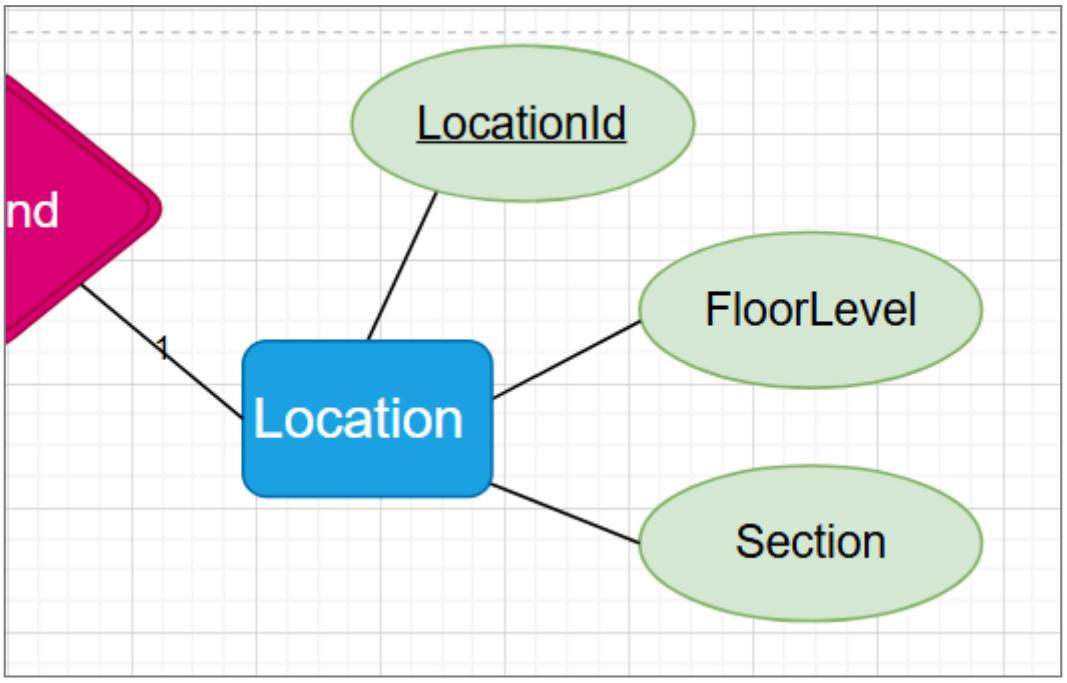
10. **Room**



The Room entity models physical rooms in the library building, such as study rooms, meeting rooms, and reading areas. These rooms can be reserved and have defined capacities.

Attribute	Description
RoomId (PK)	Unique identifier for each room.
RoomNumber	Numeric or alphanumeric code used to identify the room.
RoomType	Type of room (e.g., study_room, meeting_room, reading_area).
Capacity	Maximum number of people the room can accommodate.

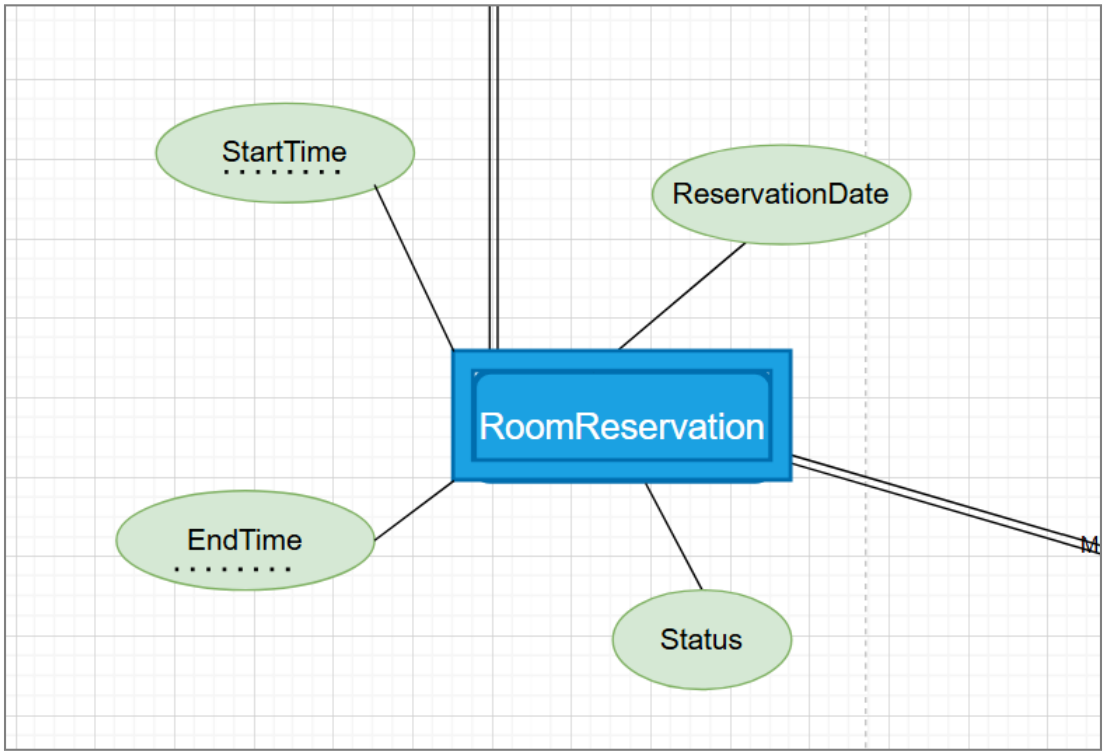
11. **Location**



The Location entity defines sections or areas within the library, often linked to rooms and used to group shelves and collections on a particular floor.

Attribute	Description
LocationId (PK)	Unique identifier for each location.
Section	Name or code of the section (e.g., Aisle A, Children's Area).
FloorLevel	Floor on which the location is situated.

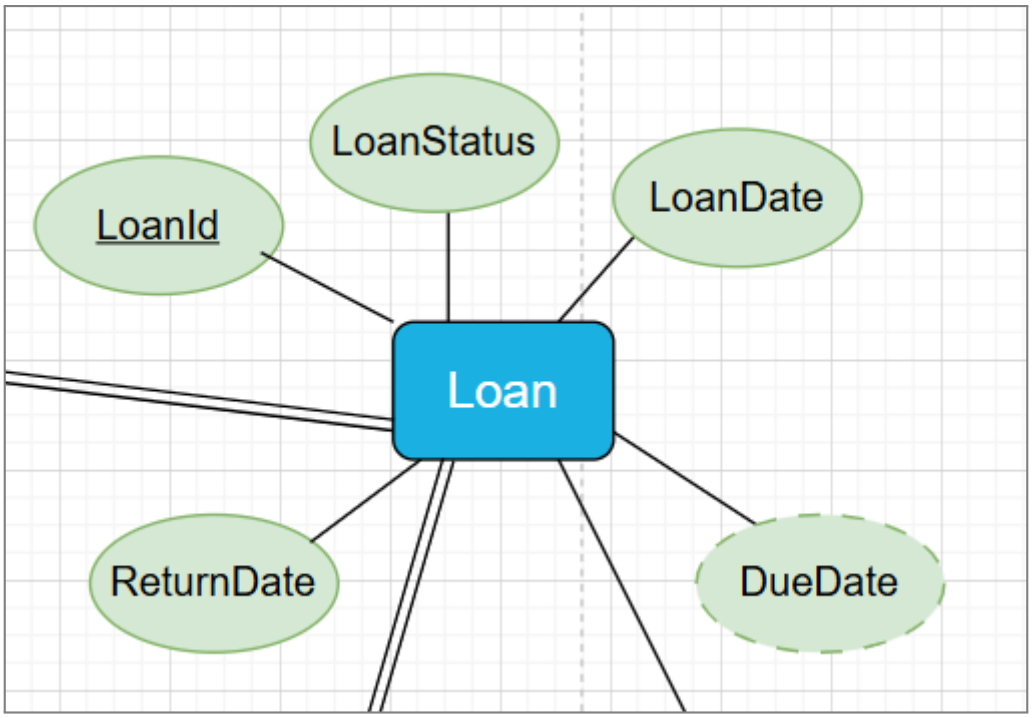
12. **RoomReservation (weak entity)**



The RoomReservation entity is a weak entity that depends on the Member entity. It stores booking details for rooms, including the reservation time period and its current status.

Attribute	Description
MemberId, StartTime, EndTime (Composite PK)	Composite key uniquely identifying a reservation by member and time range.
MemberId	Member who created the room reservation.
StartTime	Date and time when the reservation begins.
EndTime	Date and time when the reservation ends.
Status	Reservation status (e.g., active, fulfilled, cancelled, expired).

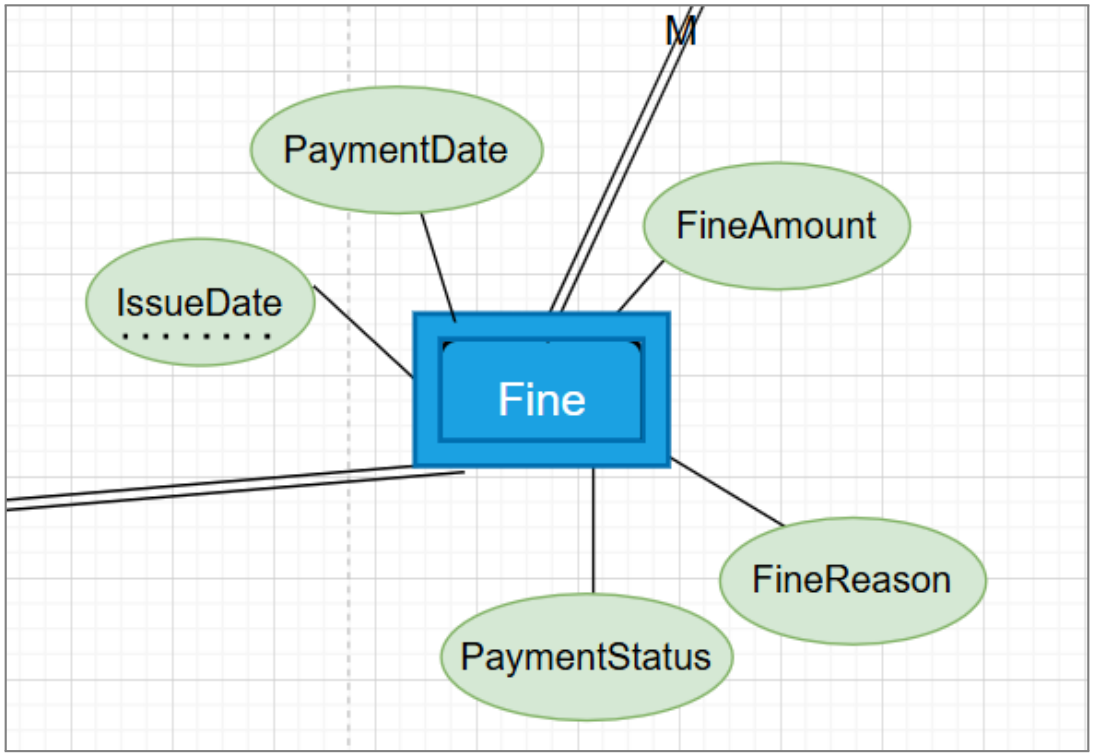
13. **Loan**



The Loan entity represents a borrowing transaction in which a member checks out a specific book copy. It records loan dates, due dates, and the current state of the loan.

Attribute	Description
LoanId (PK)	Unique identifier for each loan transaction.
LoanDate	Date on which the book copy was checked out.
DueDate	Date by which the book copy should be returned.
ReturnDate	Actual date when the book copy was returned (if returned).
LoanStatus	Status of the loan (e.g., on-loan, returned, overdue).

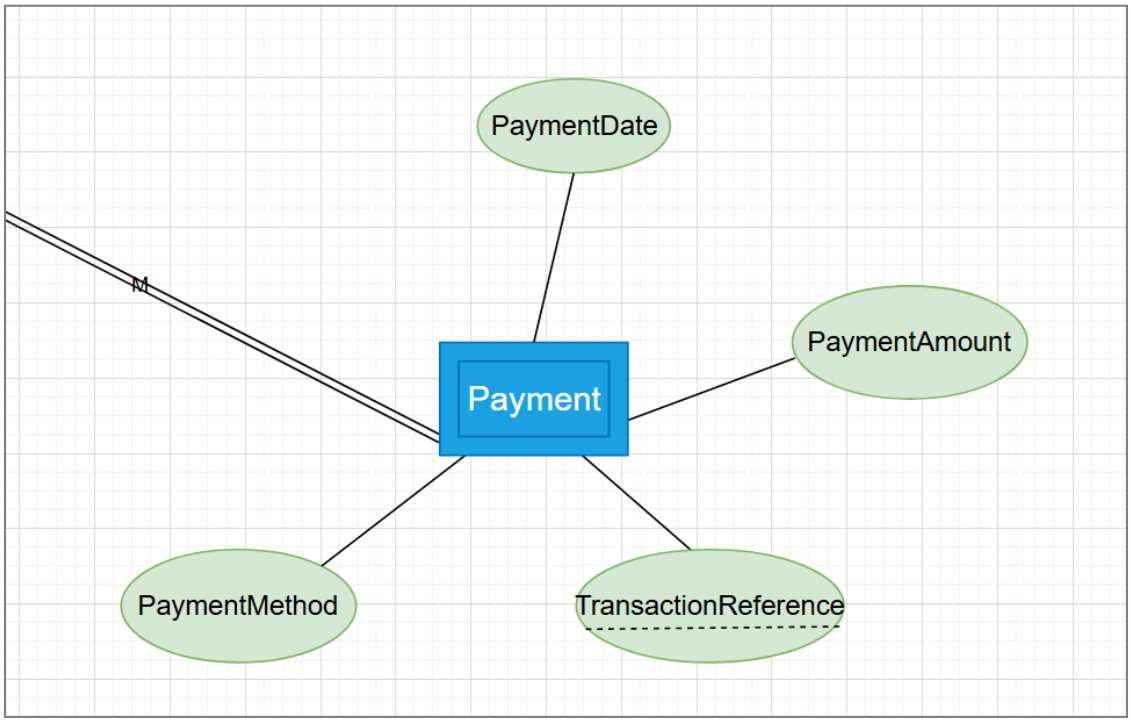
14. **Fine (weak entity)**



The Fine entity depends on a Loan record and cannot exist without it. It represents a monetary penalty linked to a specific loan, such as overdue, damage, or loss fees.

Attribute	Description
LoanId, IssuedDate (Composite PK)	Composite key identifying a specific fine instance for a loan.
FineAmount	Amount of money charged as a fine.
FineReason	Reason for the fine (e.g., overdue, damage, loss).
IssuedDate	Date on which the fine was issued.
PaymentStatus	Current status (pending, paid, or waived).
PaymentDate	Date on which the fine was paid or recorded as waived.

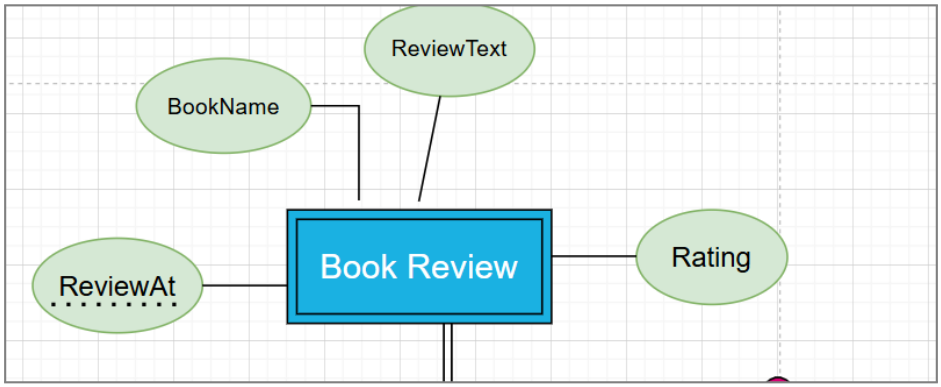
15. **Payment**



The Payment entity depends on the Member entity. It records financial transactions made by members, including amounts paid, payment methods, and transaction references.

Attribute	Description
TransactionReferen MemberId (Composite Key)	Composite key identifying a specific payment transaction for a member.
MemberId	Member who made the payment.
PaymentAmount	Total amount paid in the transaction.
PaymentDate	Date on which the payment occurred.
PaymentMethod	Method used to pay (e.g., cash, card, online).
TransactionReferen	External or internal reference code for the transaction.

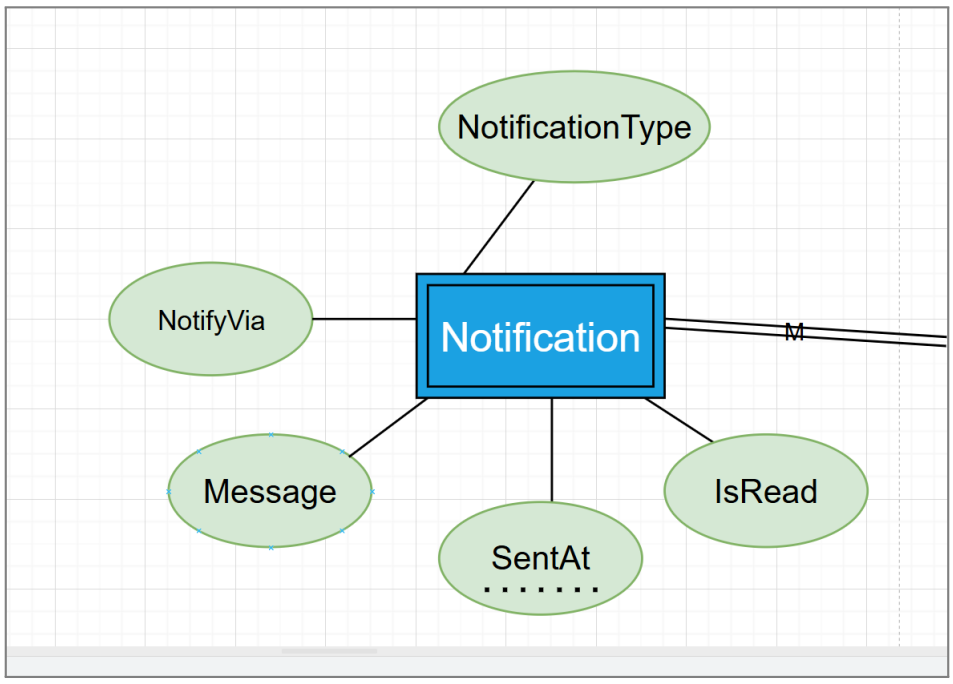
16. **BookReview (Weak)**



The BookReview entity stores feedback and ratings for books. It is considered a weak entity because it depends on the existence of a corresponding book.

Attribute	Description
BookId, ReviewedAt (Composite PK)	Composite key uniquely identifying a review for a specific book and time.
BookId	Reference to the book being reviewed.
Rating	Star rating given to the book (typically 1–5).
BookName	Book title captured at the time of review for display.
ReviewText	Free-text comments or opinion provided by the reviewer.
ReviewedAt	Date and time when the review was submitted.

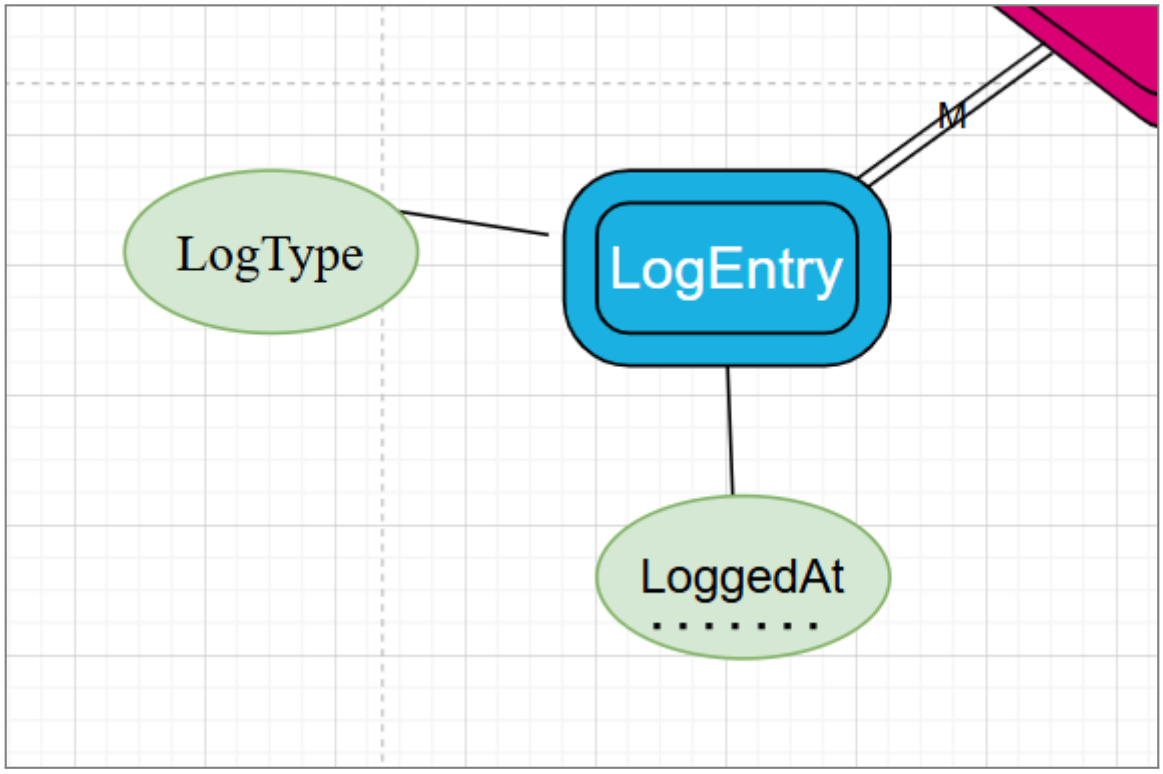
17. **Notification (Weak entity)**



The Notification entity represents messages sent to users, such as due date reminders, overdue notices, or reservation alerts. It is dependent on a user record.

Attribute	Description
UserId, SendAt (Composite PK)	Composite key uniquely identifying a notification sent to a user at a specific time.
UserId	Reference to the UserReg record receiving the notification.
NotificationType	Type of notification (e.g., due_reminder, overdue, reservation_ready).
Message	Text body of the notification.
SendAt	Date and time when the notification was sent.
IsRead	Flag indicating whether the user has read the notification.
NotifyVia	Channel used to send the notification (email, sms, app).

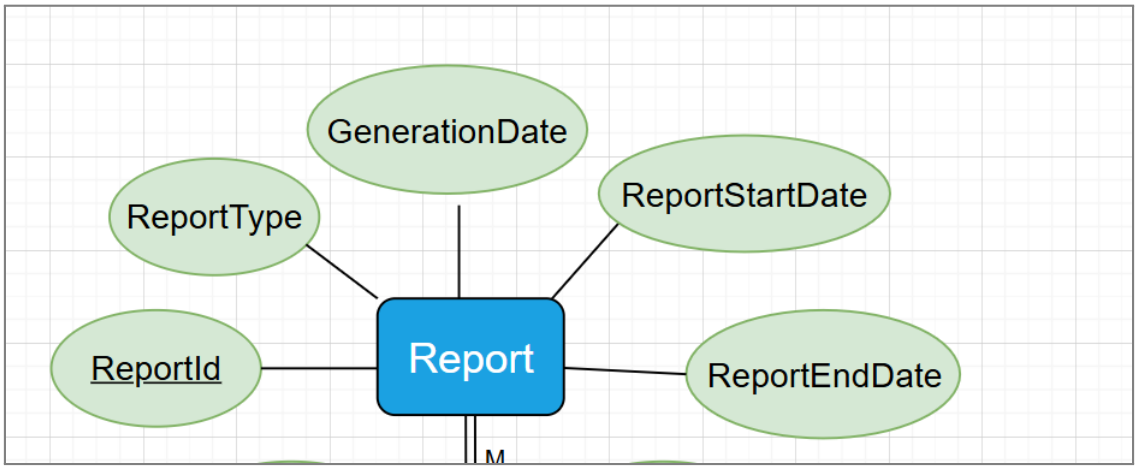
18. **LogEntry(weak entity)**



The LogEntry entity records key user actions in the system, providing an audit trail for security, troubleshooting, and usage analytics.

Attribute	Description
UserId, LoggedAt (Composite Key)	Composite key approximating a unique log entry for a user at a given time.
UserId	UserReg record representing the user performing the action.
LogType	Type of action (e.g., login, checkout, return, search).
LoggedAt	Date and time when the action occurred.

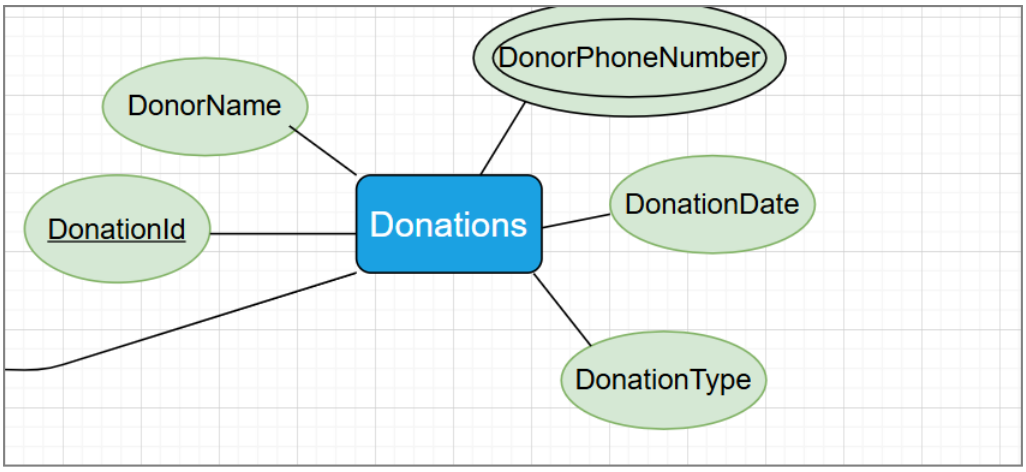
19. **Report**



The Report entity represents generated summary documents such as circulation, inventory, or financial reports, typically created by staff over a date range.

Attribute	Description
ReportId (PK)	Unique identifier for the generated report.
StaffId	Staff member who generated the report.
ReportType	Category of report (e.g., circulation, inventory, financial).
GeneratedAt	Date and time when the report was produced.
ReportStartDate	Start date of the period covered by the report.
ReportEndDate	End date of the period covered by the report.

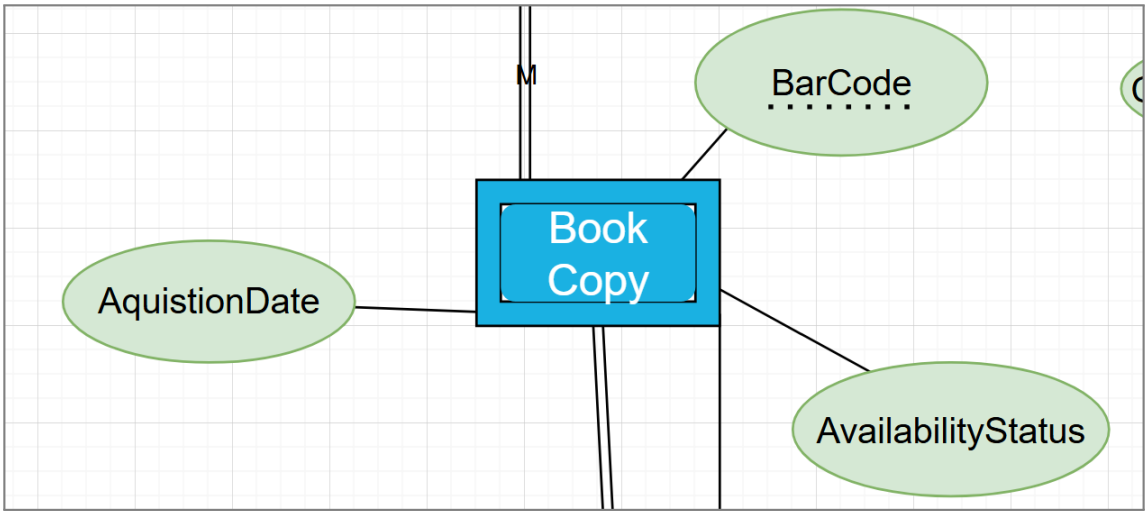
20. **Donations**



The Donations entity tracks books or other items donated to the library, along with donor information and optional linkage to cataloged book records.

Attribute	Description
DonationId (PK)	Unique identifier for each donation record.
DonorName	Name of the person or organization donating.
DonorPhoneNumbe	Contact phone number of the donor which can have multiple values.
DonationDate	Date on which the donation was received.

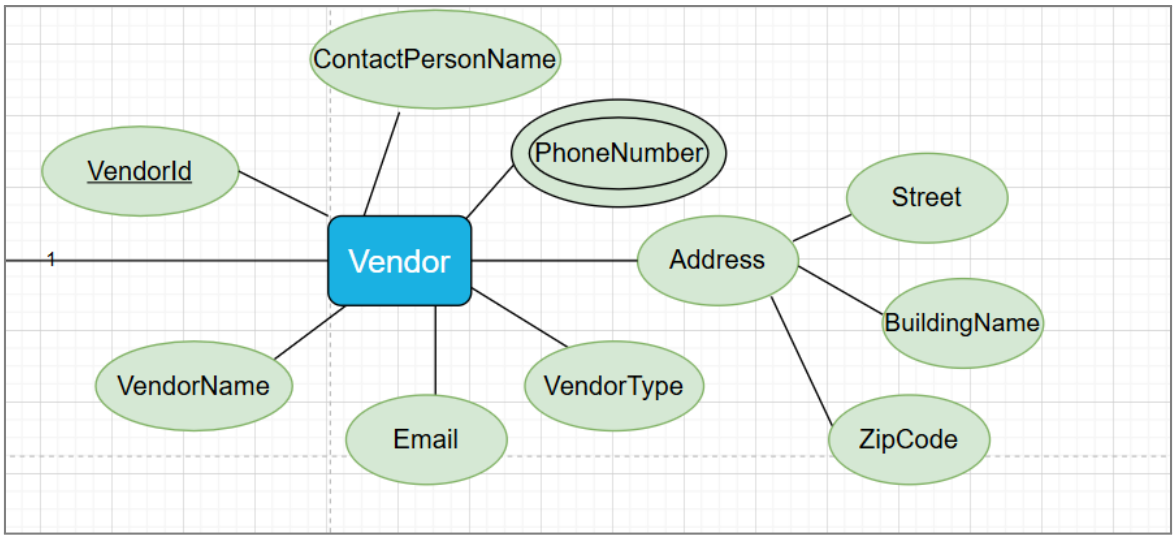
21. **BookCopy (weak entity)**



The BookCopy entity is a weak entity that depends on the Book entity to exist. It represents an individual physical copy of a book and cannot stand alone without its parent Book.

Attribute	Description
(BookId, Barcode) (PK)	Composite primary key combining the book and its unique barcode.
Barcode	Unique machine-readable code printed on the book copy.
AcquisitionDate	Date on which the library acquired this copy.
AvailabilityStatus	Current status (e.g., available, on-loan, reserved, damaged).

22. **Vendor**

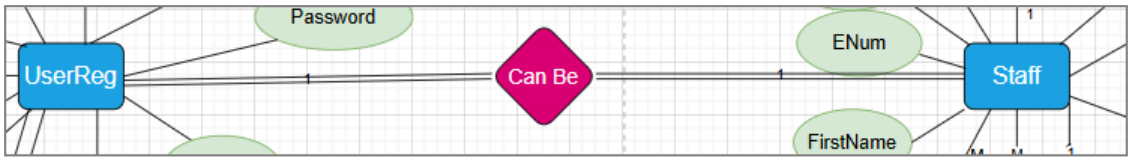


The Vendor entity stores information about suppliers that provide books, equipment, or services to the library. It is used mainly in acquisitions and procurement workflows.

Attribute	Description
VendorId (PK)	Unique identifier for each vendor.
VendorName	Official name of the vendor organization.
ContactPersonNam	Name of the primary contact person at the vendor.
Email	Email address used to communicate with the vendor.
PhoneNumber	Main phone number of the vendor.
VendorType	Type of vendor (e.g., booksupplier, equipment, services).
Address	Address of the vendor.

3.3 Entity Relationships

1. **UserReg - Staff/Member | Can Be | 1:1**

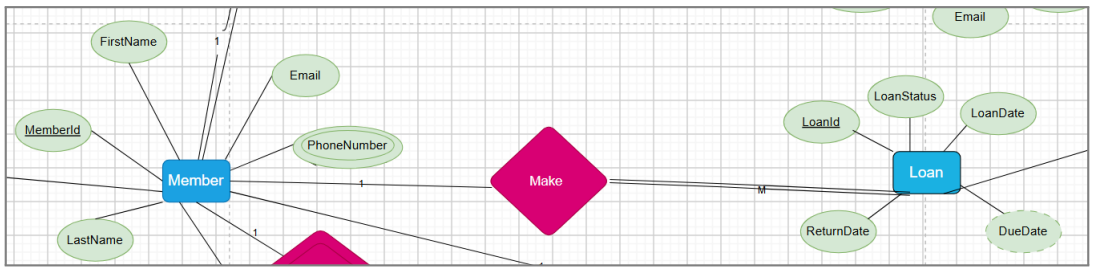


A base user record that can represent either a staff member or a library member; the relation maps a single identity to a single role record.

Participation: UserReg: total participation. Staff/Member: total participation. Every staff or member record must be backed by a user account, and every user account in the system is modeled to correspond to either a staff or a member role in this design.

Cardinality: 1:1 — one UserReg corresponds to one Staff or one Member record. This enforces a one-to-one identity model so that authentication, contact details and account state live centrally and role-specific data (employment or patron details) live in the separate staff/member table.

2. **Member - Loan | Has | 1:M**

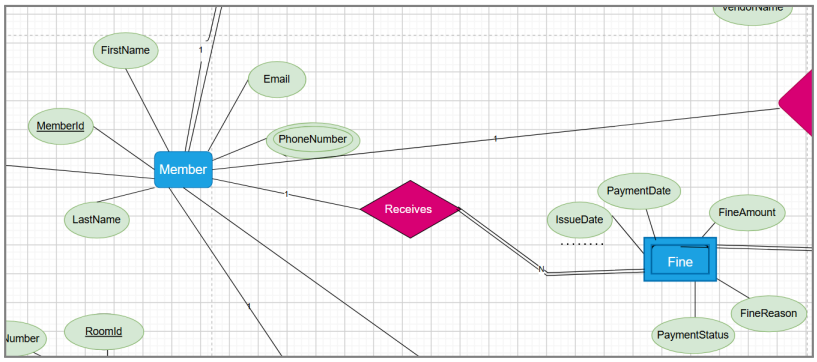


Members make borrowing transactions; loans are the recorded instances of items being checked out to patrons.

Participation: Member: partial participation. Loan: total participation. A member can exist without ever borrowing, but every loan must be linked to an existing member — loans cannot be created without a borrowing patron.

Cardinality: 1:M — one Member can have many Loans, each Loan belongs to exactly one Member. This supports multiple concurrent and historical borrowings per patron while preserving clear ownership of each loan record.

3. **Member - Fine | Receives | 1:M**

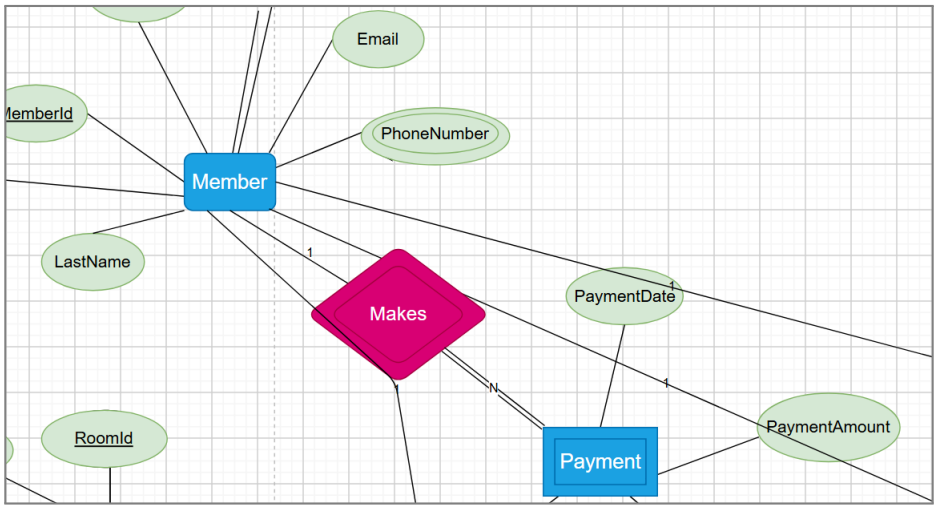


Fines are monetary entries assigned to members for policy violations like late returns or damage.

Participation: Member: partial participation. Fine: total participation. Some members may never incur fines, but each fine must reference a specific member who is responsible for payment.

Cardinality: 1:M — a single Member may receive many Fine records, while each Fine is tied to one Member. This allows tracking multiple penalties over time per patron and supports collections and reporting by member.

4. **Member - Payment | Pays | 1:M**

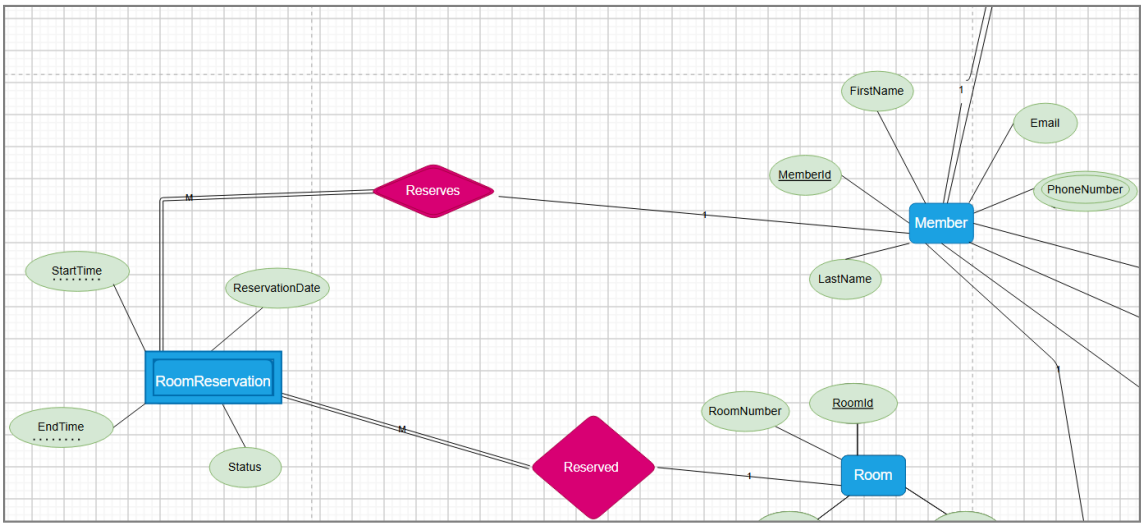


Payments record funds received from members (e.g., fine payments, fees).

Participation: Member: partial participation. Payment: total participation. A member may never make a payment, but any recorded payment must be associated with an existing member who provided the amount.

Cardinality: 1:M — one Member may have many Payment transactions; each Payment links to one Member. This supports multiple payment events (cash, card, online) per patron and reconciles receipts to members.

5. **Member - RoomReservation | Reserves | 1:M**

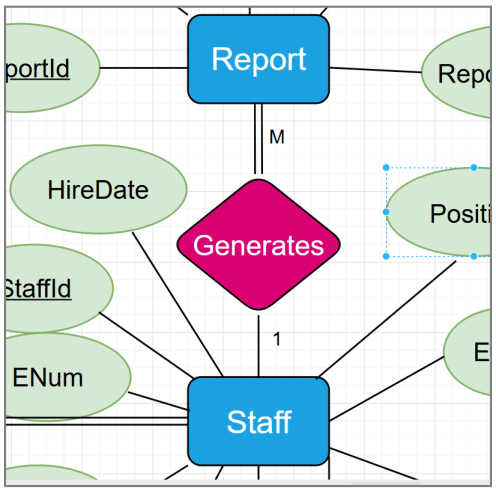


Members book rooms for study or meetings; reservations record time slots and status.

Participation: Member: partial participation. RoomReservation: total participation. Some members never reserve rooms, but every reservation must be created by an identified member.

Cardinality: 1:M — a member can make many reservations over time; each reservation belongs to one member. This reflects typical usage where patrons create multiple bookings while keeping reservation ownership clear.

6. **Staff - Report | Generates | 1:M**

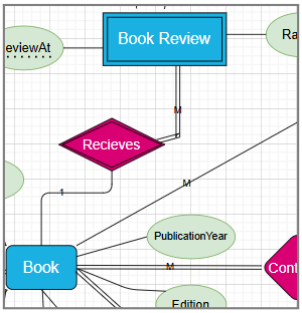


Staff members produce operational reports (circulation, inventory, financial).

Participation: Staff: partial participation. Report: total participation. Not every staff member will generate reports, but every report record should be tied to the staff user who generated it for accountability.

Cardinality: 1:M — one Staff can generate multiple Reports; each Report is generated by a single Staff. This enables auditing of report authorship and supports traceability of generated outputs.

7. **UserReg - Notification | Receives | 1:M**

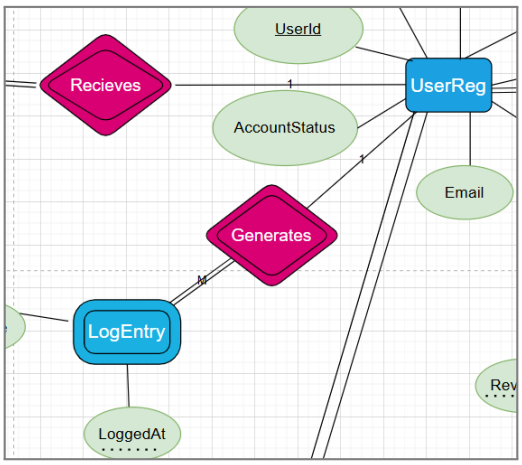


Notifications are system messages delivered to users (due reminders, reservation ready alerts).

Participation: UserReg: partial participation. Notification: total participation. Users may have no notifications at times, but each notification must target a specific user account.

Cardinality: 1:M — one UserReg can receive many Notifications; each Notification goes to a single UserReg. This supports multiple asynchronous messages per account and allows filtering or delivery status per user.

8. **UserReg - LogEntry | Generates | 1:M**

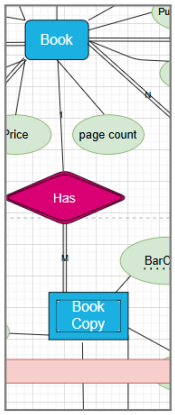


Log entries capture user actions (login, checkout, search) for auditing and analytics.

Participation: UserReg: partial participation. LogEntry: total participation. A user may never trigger certain logs, but each log entry must reference the user who performed the action.

Cardinality: 1:M — one UserReg account can generate many LogEntries; each LogEntry maps to a single user. This provides an audit trail keyed to the actor, enabling security reviews and usage metrics per account.

9. **Book - BookCopy | Has | 1:M**

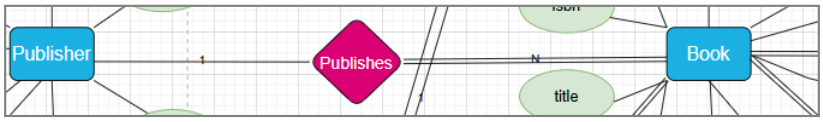


Book is the catalog entry, BookCopy represents individual physical items that are available for loan.

Participation: Book: partial participation. BookCopy: total participation. A book title may exist without any copies acquired yet; each physical copy must be tied to the catalog record that describes it.

Cardinality: 1:M — one Book may have many BookCopies; each BookCopy corresponds to one Book. This links bibliographic metadata to real-world inventory, enabling inventory counts, copy-level status, and loaning.

10. **Publisher - Book | Publishes | 1:M**

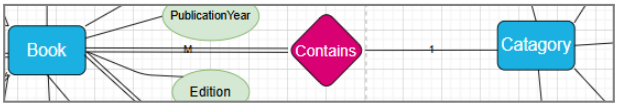


Publishers are organizations responsible for producing books; books reference their publisher for provenance.

Participation: Publisher: partial participation. Book: total participation. A publisher record can exist without any cataloged books (e.g., new vendor), but every Book should identify its Publisher when known.

Cardinality: 1:M — one Publisher may publish many Books; each Book is associated with one Publisher. This organizes bibliographic records by source and supports publisher-level reporting and contact.

11. **Category - Book | Contains | 1:M**

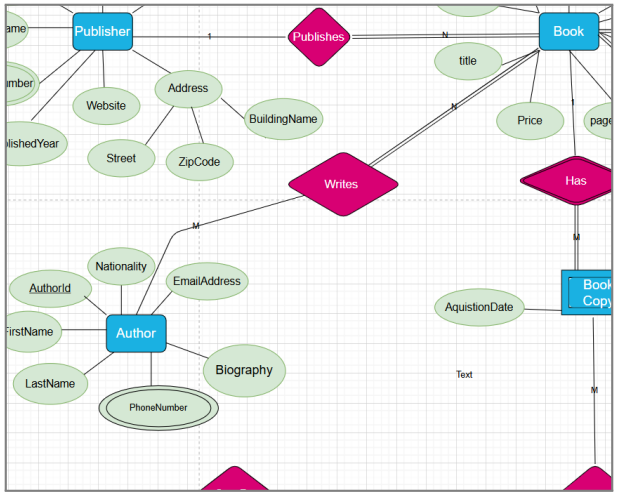


Categories group books into subjects or genres to aid discovery and shelving.

Participation: Category: partial participation. Book: total participation. A category can be defined but initially empty; each Book should be assigned to a Category for organization.

Cardinality: 1:M — one Category contains many Books; each Book belongs to one Category. This supports classification schemes and helps users browse by subject area.

12. **Author - Book | Writes | M:N**

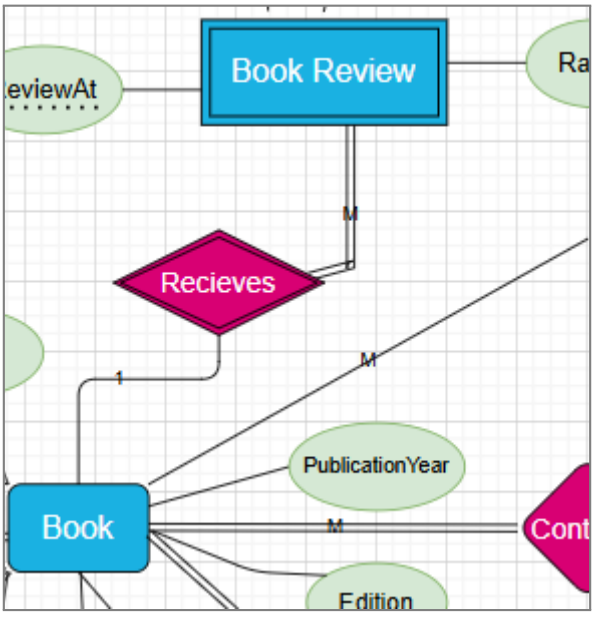


Authors create works; books may have multiple authors and authors can write many books.

Participation: Author: partial participation. Book: partial participation. Authors can be listed even if not yet linked to specific catalog entries; some books may lack author metadata initially.

Cardinality: M:N — many Authors can write many Books. This models collaborative works and enables accurate attribution and author-centric searches.

13. **Book - BookReview | Receives | 1:M**

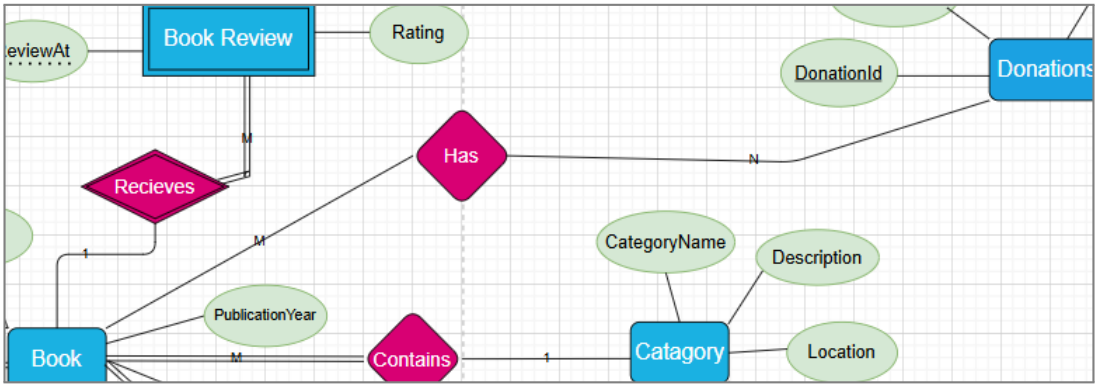


Reviews capture user feedback about titles; they are linked to the specific Book being reviewed.

Participation: Book: partial participation. BookReview: total participation. A book may have no reviews, but each review must reference the specific book it evaluates.

Cardinality: 1:M — one Book can receive many BookReviews; each BookReview refers to a single Book. This supports aggregating ratings and review lists per title for discovery and quality metrics.

14. **Book - Donation | Has | M:N**

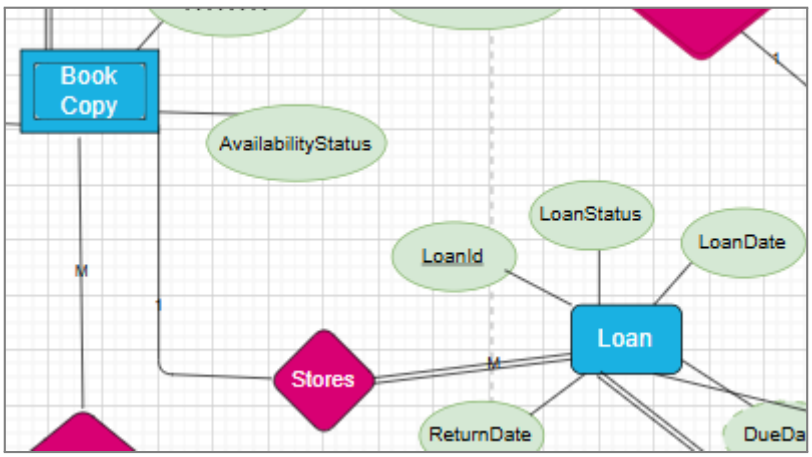


Donations may include books that map to catalog titles; a donation event can involve multiple titles and a title can be donated multiple times.

Participation: Book: partial participation. Donation: partial participation. Books can exist independently of donations in the catalog; donations can occur without immediately cataloging items as specific Book records.

Cardinality: M:N — multiple Books can be part of one Donation event and a Book title can be associated with multiple Donations over time. This models donation batches and supports recording donor contributions across many titles.

15. **BookCopy - Loan | Loaned In | 1:M**

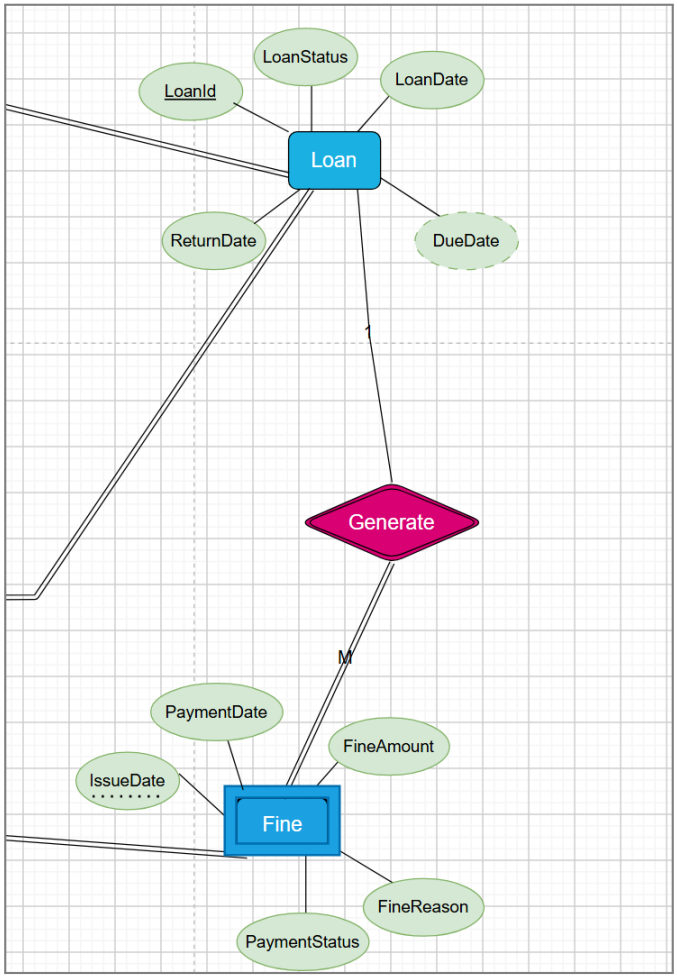


Loans are created for specific physical copies; the copy is the item actually checked out.

Participation: BookCopy: partial participation. Loan: total participation. A copy may exist without being loaned at times, but each Loan record must reference the specific copy that was checked out.

Cardinality: 1:M — one BookCopy may be involved in many Loan records over its lifetime (sequentially); each Loan refers to a single BookCopy. This enables tracking of copy-level circulation history and current availability.

16. **Loan - Fine | Generate | 1:M**

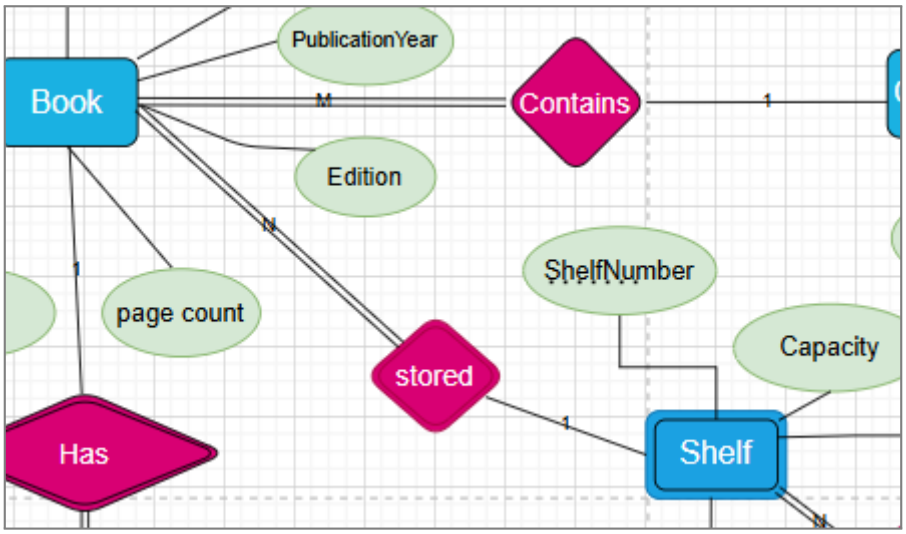


Fines arise from loan-related events (overdue returns, damages) and are recorded against loans.

Participation: Loan: partial participation. Fine: total participation. Not every loan results in a fine, but a fine must be associated with a particular loan instance to indicate what triggered the penalty.

Cardinality: 1:M — a single Loan can generate multiple Fine entries (e.g., overdue + damage fees), while each Fine ties back to one Loan. This captures multiple penalty types and allows fine calculation to be traced to the originating loan.

17. **Shelf - Book | Stores | 1:M**

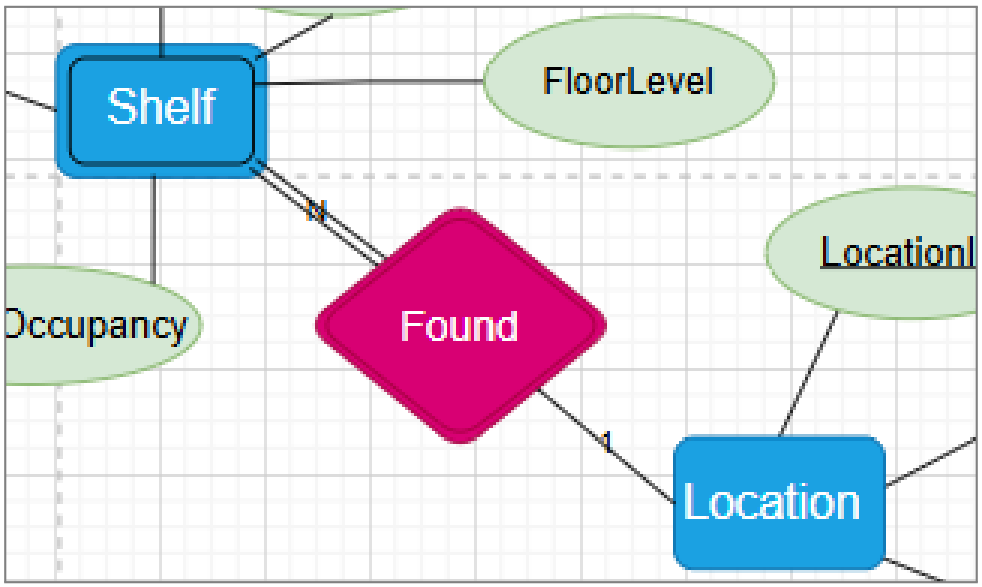


Shelves physically hold book copies; the relation ties where a book (or copies) is stored.

Participation: Shelf: partial participation. Book: total participation. A shelf may be empty, but when a book is placed in the library it should reference the shelf location where it resides.

Cardinality: 1:M — one Shelf can store many Books (copies); each Book copy record points to the Shelf where it is kept. This supports spatial inventory and locating items on the floor plan.

18. **Location - Shelf | Found | 1:M**

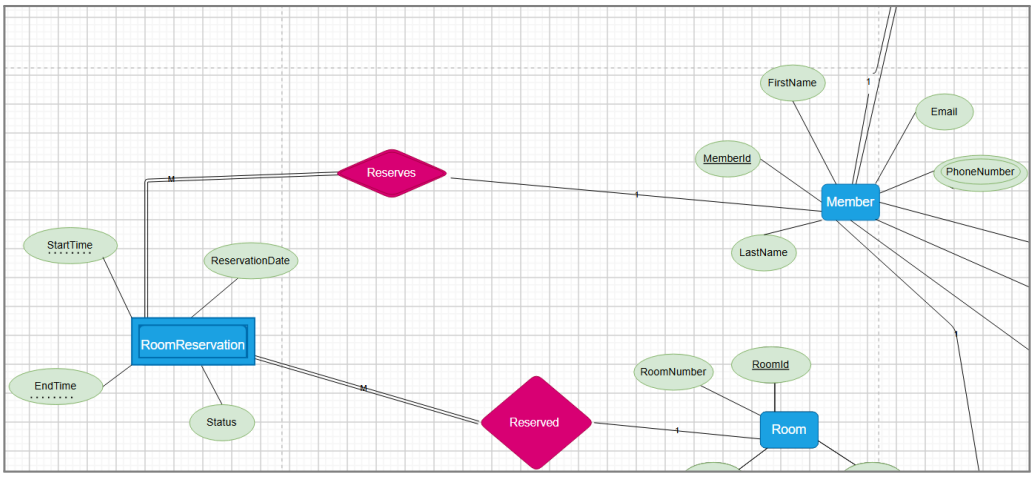


Locations are higher-level areas (sections/floors) that contain shelves.

Participation: Location: partial participation. Shelf: total participation. A location record can exist without shelves initially; every shelf must belong to a location so its physical context is known.

Cardinality: 1:M — one Location contains many Shelves; each Shelf is located in a single Location. This models building layout and supports navigation from section to shelf.

19. **Room - Reservation | Reserved | 1:M**

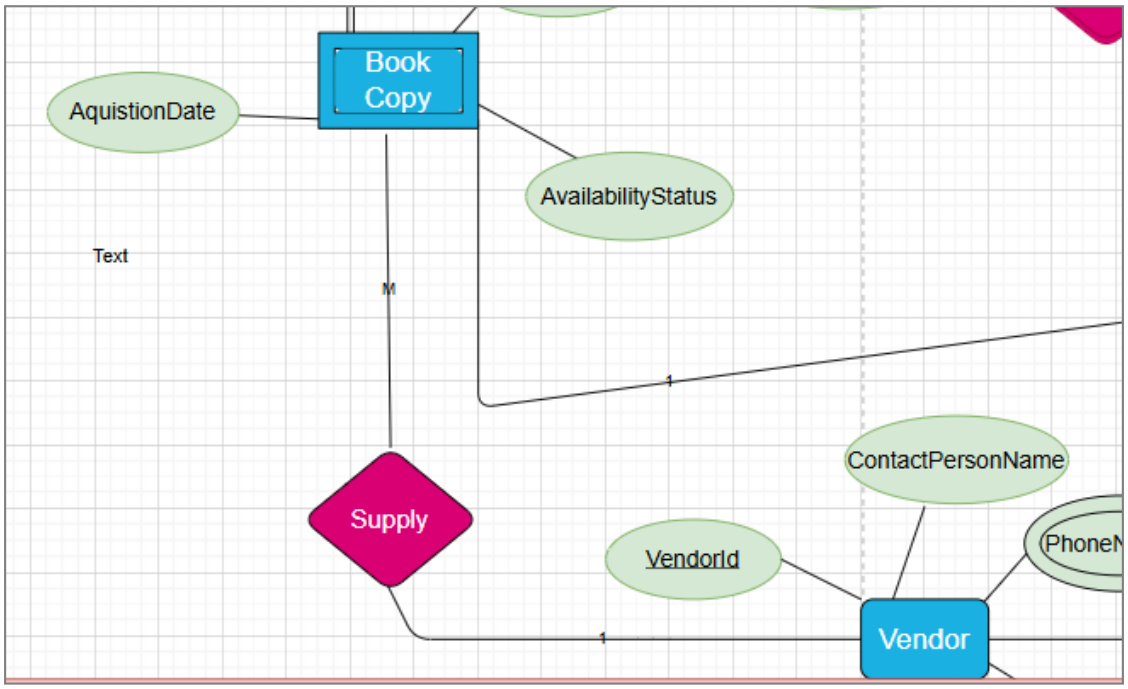


Rooms are reservable resources; reservations capture booked time slots and the reserving member.

Participation: Room: partial participation. Reservation: total participation. A room can exist without reservations; every reservation must reference the room being booked.

Cardinality: 1:M — a Room may have many Reservations over time; each Reservation is for one Room. This supports scheduling and conflict checks for each room.

20. **Vendor - BookCopy | Supply | 1:M**

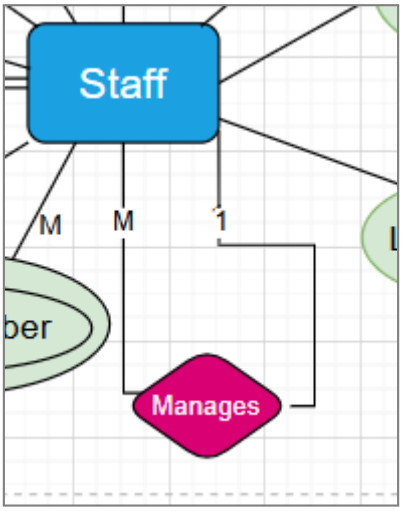


Vendors supply physical copies or materials to the library; each copy can reference its supplier for acquisition records.

Participation: Vendor: partial participation. BookCopy: partial participation. A vendor can exist without current supplies recorded; a copy may have been donated instead of purchased, so vendor linkage can be optional.

Cardinality: 1:M — one Vendor may supply many BookCopies; each BookCopy may be associated with a single Vendor (or none). This stores procurement provenance while allowing non-vendor acquisitions.

21. **Staff - Staff | Manages | 1:M (recursive)**



Staff members may supervise or manage other staff; this is a recursive reporting relation within the staff hierarchy.

Participation: Manager (Staff): partial participation. Subordinate (Staff): partial participation. Not every staff member manages others, and not every staff member has a manager recorded (e.g., director may have no manager in the system).

Cardinality: 1:M (recursive) — one Staff member may manage many Staff members; each subordinate typically reports to one manager in this model. This models organizational hierarchy and enables queries for teams, supervisors, and reporting chains.

4 Logical Design

The logical design phase defines the structural blueprint of the Library Database Management System before the physical implementation takes place. It translates the conceptual Entity–Relationship (ER) model into a set of initial relational tables, identifies data elements, and presents how entities will be represented in the database.

At this stage, the focus is on describing the raw entities as derived from the conceptual analysis, including their attributes exactly as they appear before applying normalization rules. This allows the system designers to identify composite attributes, multivalued attributes, and repeating groups, which will later be resolved during the normalization process.

Therefore, the logical design serves as a bridge between conceptual modeling and normalized relational schema, providing a clear foundation upon which the normalization (performed in the next section) is applied.

4.1 Initial Relational Tables (Before Normalization)

In this section, all entities and relationships are presented as pre-normalized tables. Each table is followed by a short explanation describing how the mapping from the ER diagram to the relational model was applied.

User & Account Management

1. UserReg

UserId	Username	Email	Password	Registratio	Role	AccountStatus	Address

Table 2: UserReg (Pre-normalization)

Relational Mapping Explanation

- UserReg comes from the strong entity representing system users in the ER diagram.
- The entity is mapped to a relation where each attribute becomes a column.
- UserId is the logical primary key of this table.
- Role captures whether the user is a staff member or library member, and AccountStatus stores user account state.
- Address is currently kept as a single attribute (possibly composite); this will be refined during normalization.

2. Staff

StaffId	ENum	FirstName	LastName	Position	HireDate	PhoneNumb	Email	*UserId (FK)*

Table 3: Staff (Pre-normalization)

Relational Mapping Explanation

- Staff is mapped from the Staff entity in the ER model.
- All staff-related attributes are placed in one relation.
- StaffId acts as the staff identifier (PK), while UserId is a foreign key referencing UserReg, implementing the relationship between User and Staff (typically 1:1).
- ENum can represent an employee number (a candidate key).
- Contact fields (PhoneNumber, Email) are currently single attributes; if multiple contacts per staff are needed, they will be decomposed later.

3. Member

MemberId	FirstName	LastName	Email	Phone	*UserId (FK)*

Table 4: Member (Pre-normalization)

Relational Mapping Explanation

- Member is derived from the Member entity in the ER diagram.
- MemberId identifies each library member.
- UserId is a foreign key that links each member back to the generic UserReg record, implementing the relationship between User and Member.
- Personal details (name, email, phone) are stored in this relation before any decomposition of composite or multivalued attributes.

4. LibraryCard

CardId	IssueDate	ExpiryDat	CardStatus	Replaceme	*MemberId (FK)*

Table 5: LibraryCard (Pre-normalization)

Relational Mapping Explanation

- LibraryCard is created from the relationship between Member and LibraryCard in the conceptual model.
- CardId serves as the card identifier and is treated as the primary key.
- MemberId is a foreign key referencing Member, implementing the 1:M relationship (one member can hold multiple cards over time).
- Attributes like IssueDate, ExpiryDate, and CardStatus are relationship attributes attached to the card itself.

Book & Catalog Module

5. Book

BookId	ISBN	Title	Publication Year	Edition	PageCount	Price	*CategoryId (FK)*	*PublisherId (FK)*

Table 6: Book (Pre-normalization)

Relational Mapping Explanation

- Book is derived from the Book entity in the ER model.
- BookId is the surrogate key; ISBN may act as a natural candidate key.
- CategoryId and PublisherId implement 1:M relationships from Category and Publisher to Book via foreign keys.
- Logical book information (title, year, edition, etc.) is stored here before separating physical copies.

6. Category

CategoryId	Category Name	Descriptio	Location	IsActive

Table 7: Category (Pre-normalization)

Relational Mapping Explanation

- Category comes from the Category entity that classifies books into subjects or genres.
- Each category is mapped to a row identified by CategoryId.
- Location is kept as an attribute indicating where this category is placed in the library (this may later be related to shelves/rooms during normalization or refinement).
- IsActive is a status attribute derived from the conceptual design to enable or disable categories logically.

7. Publisher

PublisherId	Publisher Name	Country	Email	Phone	Website	Established Year

Table 8: Publisher (Pre-normalization)

Relational Mapping Explanation

- Publisher is created from the Publisher entity in the ER diagram.
- PublisherId is the identifier for each publisher.
- Contact and descriptive attributes (Country, Email, Phone, Website, EstablishedYear) are mapped directly as columns.

8. Author

AuthorId	FirstName	LastName	Nationality	Biography	Email	Phone

Table 9: Author (Pre-normalization)

Relational Mapping Explanation

- Author is mapped from the Author entity in the ER model.
- AuthorId uniquely identifies each author.
- Name and profile information are represented as simple attributes at this stage; possible multivalued or composite aspects will be handled during normalization if needed.

9. Shelf

LocationID	ShelfNo	FloorLevel	Capacity	Current Occupancy

Table 10: Shelf (Pre-normalization)

Relational Mapping Explanation

- Shelf is derived from the Shelf entity, representing physical storage areas for books.
- LocationID links shelves to a location concept (later related to the Location table).
- Shelf-level attributes (capacity and occupancy) are attached to this entity as columns.

10. Writes (M:N → Book & Author)

BookId	AuthorId

Table 11: Writes (Pre-normalization)

Relational Mapping Explanation

- Writes is generated from the M:N relationship between Book and Author.
- According to relational mapping rules, each M:N relationship becomes a separate relation whose primary key is typically the combination (BookId, AuthorId).
- Both attributes are foreign keys, referencing Book and Author respectively.

Facility & Location Module

11. Room

RoomId	Room Number	RoomType	Capacity

Table 12: Room (Pre-normalization)

Relational Mapping Explanation

- Room is mapped from the Room entity in the ER diagram.
- Each room in the library is represented by a row identified by RoomId.
- Attributes like RoomNumber, RoomType, and Capacity are derived directly from the entity.

12. Location

LocationId	Section	FloorLevel	*RoomId (FK)*

Table 13: Location (Pre-normalization)

Relational Mapping Explanation

- Location is a **strong entity** representing defined physical areas in the library.
- LocationId is the primary key uniquely identifying each location.
- *RoomId (FK)* implements the 1:M relationship where one Room can contain multiple Locations.
- Section and FloorLevel describe the physical details of each location.

13. RoomReservation

Status	StartTime	EndTime	*MemberId(FK)*

Table 14: RoomReservation (Pre-normalization)

Relational Mapping Explanation

- RoomReservation is modeled as a weak entity dependent on *Member*.
- The primary key of RoomReservation is the composite key StartTime, EndTime, and *MemberId (FK)*.
- *MemberId (FK)* references the parent *Member* table and links each reservation to the owning member.
- Status records the current state of the reservation, while StartTime and EndTime together define the reserved time interval.

Circulation & Transactions

14. Loan

LoanID	*BookId(FK)*	LoanDate	DueDate	ReturnDat	LoanStatus	*MemberId (FK)*

Table 15: Loan (Pre-normalization)

Relational Mapping Explanation

- Loan is a **strong entity** representing the borrowing transaction between a Member and a Book.
- LoanID is the surrogate primary key introduced to uniquely identify each loan.
- *BookId (FK)* references the Book table and identifies which book the member borrowed.
- *MemberId (FK)* references the Member table and identifies who borrowed the book.
- Attributes such as LoanDate, DueDate, ReturnDate, and LoanStatus describe the lifecycle of the borrowing transaction.

15. Fine

Fine amount	FineReasoı	Issuedate	Payment Status	Payment Date	*LoanId(FK)*	*MemberId (FK)*

Table 16: Fine (Pre-normalization)

Relational Mapping Explanation

- Fine is modeled as a **weak entity** that depends on the parent entity *Loan*.
- Because it has no independent identifier, its **primary key is the composite**: Issuedate and *LoanId (FK)*.
- *LoanId (FK)* links each fine to the specific overdue or damaged loan on which it depends.
- *MemberId (FK)* is an additional foreign key that associates the fine with the responsible library member but is **not** part of the primary key.
- Other attributes such as Fine amount, FineReason, Payment Status, and Payment Date describe the fine details and its payment status.

16. Payment

Payment Amount	Payment Date	Payment Method	Transactio	*FineId (FK)*	*MemberId(FK)*	FineAmou

Table 17: Payment (Pre-normalization)

Relational Mapping Explanation

- Payment is modeled as a **weak entity** whose owner is the *Member* entity.
- Because it has no standalone identifier, the **composite primary key** is: *MemberId (FK)* + Transaction Reference.
- *MemberId (FK)* identifies which member made the payment and forms part of the identifying key.
- *FineId (FK)* links the payment to the specific fine it settles but is **not part of the primary key**.
- FineAmount is included only because this is the pre-normalized schema; redundancy will be removed during normalization.
- Other attributes (Payment Amount, Payment Date, Payment Method) describe the payment details.

Reviews, Logs & Notifications

17. BookReview

ReviewAt	Title	ReviewTex	Rating	*BookId(FK)*

Table 18: BookReview (Pre-normalization)

Relational Mapping Explanation

- BookReview is modeled as a **weak entity** that depends on the parent entity *Book*.
- Since it has no independent identifier, the **composite primary key** is formed by ReviewAt and *BookId (FK)*.
- *BookId (FK)* ensures that each review is linked to the specific book it is reviewing.
- Attributes such as BookName, ReviewText, and Rating describe the content of the review.
- Because this is the pre-normalized stage, BookName remains as an attribute even though it is derived from Book (this will be corrected during normalization).

18. Notification

Notif.Type	Message	SendAt	IsRead	NotifyVia	*UserId(FK)*

Table 19: Notification (Pre-normalization)

Relational Mapping Explanation

- Notification is modeled as a **weak entity** that depends on the parent entity *UserReg*.
- The **composite primary key** of Notification is: SendAt + *UserId (FK)*.
- *UserId (FK)* identifies the owning user and forms part of the key as required for weak entities.
- Other attributes (Notif.Type, Message, IsRead, NotifyVia) describe the content and delivery method of the notification.
- Because this is the pre-normalized schema, all attributes are represented as simple columns and refinements will occur during normalization.

19. LogEntry

LoggedAt	LogType	*UserId(FK)*

Table 20: LogEntry (Pre-normalization)

Relational Mapping Explanation

- LogEntry is modeled as a **weak entity** whose parent (owner) is the *UserReg* entity.
- Its **composite primary key** is formed by LoggedAt + *UserId (FK)*, ensuring each log entry is uniquely tied to the user who generated it.
- *UserId (FK)* links each log entry to the specific user responsible for that system action.

- LogType and Timestamp describe the nature of the system event and its precise timing.
- Because this is the pre-normalized schema, all attributes appear as simple columns before optimization in the normalization stage.

20. Report

ReportId	Report Type	Generation Date	Report Start Date	ReportEnd Date	*StaffId (FK)*

Table 21: Report (Pre-normalization)

Relational Mapping Explanation

- Report is a strong entity that stores generated system reports.
- StaffId is mapped as a foreign key referencing Staff, representing who generated the report.
- Other attributes describe the report's type and time period.

Donations

21. Donation

DonationId	Donor Name	Donor Phone Number	Donotion Date	*BookId (FK)*	Donation Type

Table 22: Donation (Pre-normalization)

Relational Mapping Explanation

- Donation is mapped from the Donation entity in the ER model.
- It includes donor details and basic information about the donated book(s).
- BookId links directly to Book, but since one donation may include multiple books, an M:N relationship is further represented through Has.

22. Has (M:N → Donations & Book)

DonationId	BookId

Table 23: Has (Pre-normalization)

Relational Mapping Explanation

- Has is the associative relation created from the M:N relationship between Donation and Book.
- According to mapping rules, each M:N relationship becomes a separate relation with foreign keys referencing the participating entities.
- (DonationId, BookId) together form the logical composite key.

Vendor and Acquisition

23. BookCopy

Barcode	AcquisitionDate	AvailabilityStatus	*BookId(FK)*	*VendorId (FK)*

Table 24: BookCopy (Pre-normalization)

Relational Mapping Explanation

- BookCopy is modeled as a **weak entity** that depends on the parent entity *Book*.
- Since it has no independent identifier, its **composite primary key** is: Barcode + *BookId (FK)*.
- *BookId (FK)* links each physical copy back to the logical book it belongs to and forms part of the identifying key.
- *VendorId (FK)* records which vendor supplied this copy but is **not** part of the weak-entity key.
- AcquisitionDate and AvailabilityStatus describe the condition and status of the specific physical copy.

24. Vendor

VendorId	VendorNar	Contact Person Name	Email	Phone Number	Address	Vendor Type

Table 25: Vendor (Pre-normalization)

Relational Mapping Explanation

- Vendor is a strong entity representing suppliers.
- All vendor-related attributes are mapped as columns.
- VendorId uniquely identifies each vendor and is used as a foreign key in BookCopy.

ER-to-Relational Mapping Summary

This section summarizes the mapping rules applied to convert the Entity–Relationship (ER) model into the corresponding initial relational schema:

1. **Strong Entities → Independent Base Tables**
   Every strong entity (such as UserReg, Book, Author, Member, Staff, Room, Vendor, Category, Publisher, Report, Donation, Location) is mapped to its own relation. Each relation contains a unique primary key that fully identifies its tuples.

2. **Weak Entities → Tables Including Owner's Key as Foreign Key**
   Weak entities (BookReview, Notification, LogEntry, BookCopy, Fine, Payment, RoomReservation) are mapped to separate relations that include:

   - their own partial key, and
   - a foreign key referencing the owning entity

   This preserves their dependence on the parent entity for identification.

3. **1:M Relationships → Foreign Key on the "Many" Side**
   For relationships with a one-to-many structure (e.g., Category–Book, Publisher–Book, Member–Loan, Staff–Report, Room–Location), the primary key of the "one" entity becomes a foreign key in the relation on the "many" side.

4. **M:N Relationships → Associative (Bridge) Tables**
   Many-to-many relationships (such as Book–Author and Donation–Book) require the creation of an associative table. These tables contain only the foreign keys of the participating entities, which together form a composite primary key.

5. **Relationships with Attributes → Separate Relations**
   When a relationship contains its own attributes (e.g., Loan, RoomReservation, Fine, Payment), it is mapped as a standalone table. This table stores:

   - all relationship-specific attributes, and
   - foreign keys referencing all participating entities

6. **Attributes → Relational Columns (Pre-Normalization)**
   All attributes from the ER diagram—including composite, multivalued, and potentially redundant attributes—were directly converted into columns during the logical design stage. Their refinement and decomposition are handled later during normalization.

4.2 Normalization

Normalization is a systematic process in relational database design used to organize data into well-structured tables. Its primary goal is to minimize redundancy, prevent data inconsistencies, and ensure that each table accurately represents a single entity or relationship. Normalization improves the logical integrity of the database by breaking complex tables into smaller, more manageable relations, each with clearly defined attributes and dependencies. In the context of a Library

Management System, where entities such as Users, Members, Staff, Books, Authors, Publishers, Vendors, Loans, and Payments are interdependent, normalization ensures accurate, consistent, and scalable data storage.

Why Normalization is Needed

Normalization is necessary for several important reasons:

- **Elimination of Redundancy:** Redundant data, such as repeating publisher phone numbers or member contact details across multiple records, leads to wasted storage and inconsistency. Normalization ensures that each fact is stored only once.
- **Avoiding Anomalies:**
  - *Update Anomaly:* Changing a value (e.g., a publisher's address) in multiple rows may lead to inconsistent data if some rows remain outdated.
  - *Insertion Anomaly:* The database may require unnecessary information before allowing new data to be inserted. For example, inserting a new publisher might not be possible without also creating a book entry.
  - *Deletion Anomaly:* Deleting the last book by a publisher may unintentionally delete all information about that publisher.
- **Improved Data Integrity:** Normalization enforces proper primary and foreign key relationships, ensuring that data remains logically consistent across tables.
- **Better Organization and Maintainability:** Structuring data into smaller, well-defined tables makes the database easier to understand, modify, and expand.
- **Efficient Querying and Future Expansion:** A normalized schema supports optimized queries, reduces duplication, and provides a strong foundation for future system modules such as analytics, reporting, and automation.

4.2.1 Relational Schema to First Normal Form (1NF)

Before normalization, several tables in the relational schema violated the requirements of First Normal Form (1NF). The goal of converting the schema to 1NF is to eliminate composite attributes, remove multi-valued attributes, and ensure that every field contains atomic values.

Criteria for 1NF

A table is considered to be in **First Normal Form** if:

- All attribute values are **atomic** (indivisible).
- No attribute contains **multiple values** or lists.
- No **repeating groups** exist.
- Each record is uniquely identified by a primary key.

Only tables that violated these conditions were modified.

Composite Attribute: UserReg.Address

The `Address` field in **UserReg** is a composite attribute (Street, Building, ZipCode). To achieve 1NF, it is decomposed into atomic components.

UserReg (Before 1NF)

UserId	Username	Email	Password	RegistrationDate	Role	Account Status	Address (Composite)

Table 26: UserReg (Pre-normalization)

UserReg (After 1NF)

UserId	Username	Email	Password	Registration Date	Role	Account Status	ZipCode	StreetNo	BuildingName

Table 27: UserReg after breaking down the composite attribute

Multi-Valued Attributes: Phone Numbers

Assumption: *"The system allows multiple phone numbers to be registered for Staff, Members, Publishers, and Vendors."*

To satisfy 1NF, each multi-valued phone attribute is moved into a separate table.

Staff (Before 1NF)

StaffId	H-Staff	Essn	FirstName	LastName	Dept	Email	PhoneNumber (Multi-valued)	*UserId* (*FK*)

Table 28: Staff (Pre-normalization)

Staff (After 1NF)

StaffId	H-Staff	Essn	FirstName	LastName	Dept	Email	*UserId* (*FK*)

Table 29: Staff after removing multi-valued attribute

StaffPhoneNumber (New Table)

StaffId (*FK*)	PhoneNumber

Table 30: Atomic phone numbers for Staff

Member (Before 1NF)

MemberId	FirstName	LastName	PhoneNumber (Multi-valued)	*UserId* (*FK*)

Table 31: Member (Pre-normalization)

Member (After 1NF)

MemberId	FirstName	LastName	*UserId* (*FK*)

Table 32: Member after removing multi-valued attribute

MemberPhoneNumber (New Table)

MemberId	PhoneNumber

Table 33: Atomic phone numbers for Members

Publisher (Before 1NF)

PublisherId	PublisherName	Country	Email	PhoneNumber (Multi-valued)	EstablishedYear

Table 34: Publisher (Pre-normalization)

Publisher (After 1NF)

PublisherId	PublisherName	Country	Email	EstablishedYear

Table 35: Publisher after removing multi-valued attribute

PublisherPhoneNumber (New Table)

PublisherId	PhoneNumber

Table 36: Atomic phone numbers for Publishers

Vendor (Before 1NF)

VendorId	VendorName	Contact Person-Name	Email	PhoneNumber (Multi-valued)	Address (Compos-ite)	VendorType

Table 37: Vendor (Pre-normalization)

Vendor (After 1NF)

VendorId	VendorName	ContactPerson	Email	VendorType	ZipCode	StreetNo	BuildingName

Table 38: Vendor after removing composite and multi-valued attributes

VendorPhoneNumber (New Table)

VendorId	PhoneNumber

Table 39: Atomic phone numbers for Vendors

All composite attributes were decomposed, and all multi-valued attributes were removed by introducing new relational tables. Only relations that violated 1NF were modified. At this stage, all tables now contain atomic values and satisfy the requirements of First Normal Form.

4.2.2 Conversion from 1NF to 2NF

After ensuring that all tables satisfy the requirements of First Normal Form, the next step is to eliminate any **partial dependencies**. A table is considered to be in **Second Normal Form (2NF)** if:

- It is already in 1NF.
- Every non-key attribute is **fully functionally dependent** on the entire primary key.

Partial dependency occurs when a non-key attribute depends on only **part** of a composite primary key instead of the whole key. Whenever such a dependency is detected:

- The attributes depending on part of the key are moved into a **new table**.
- The original table keeps only attributes fully dependent on the entire composite key.

Only tables with composite primary keys were checked for 2NF issues. In our schema, the following table required modification:

A. Shelf Table

The **Shelf** table contains a composite key consisting of ShelfNo and LocationID. The attribute **FloorLevel** depends only on LocationID, not on the full key. This represents a partial dependency, and therefore the table violates 2NF.

To correct this, the table was decomposed as follows:

- A new table **LocationFloorLevel** was created to store the dependency between LocationID and FloorLevel.
- The **Shelf** table retains only attributes fully dependent on both key fields.

Shelf (Before 2NF)

ShelfNo	LocationID	FloorLevel	Capacity	Current Occupancy

Table 40: Shelf (Before 2NF)

Shelf (After Removing Partial Dependency)

ShelfNo	LocationID	Capacity	Current Occupancy

Table 41: Shelf (After 2NF Decomposition)

Although `Shelf` initially appeared to violate 2NF because the attribute `FloorLevel` depends only on `LocationID`, creating a separate table was unnecessary. This is because the database already includes a `Location` table that stores `LocationID` together with its corresponding `FloorLevel`. Therefore, to satisfy 2NF, `FloorLevel` was removed from the `Shelf` table and its dependency is preserved through the existing foreign key relationship with `Location`. Since the functional dependency is already represented in the `Location` table, creating an additional table would introduce redundancy without improving normalization.

B. Payment Table

The `Payment` table also has a **composite primary key**, formed by TransactionReference and MemberId. To satisfy Second Normal Form (2NF), all non-key attributes must depend on the **entire** composite key.

However, the attribute `PaymentAmount` depends only on MemberId, meaning this attribute is determined by one part of the key and not the complete key. This results in a **partial dependency**, which violates 2NF.

To correct this:

- The main `Payment` table retains attributes that depend on the full composite key.
- A new table is created to store `PaymentAmount` with MemberId as its primary key, ensuring that each non-key attribute depends fully on the key of its table.

Payment (Before 2NF)

PaymentDate	PaymentMethod	TransactionReference	*FineId*	*MemberId*	PaymentAmount

Table 42: Payment (Pre-normalization)

Payment (After 2NF)

Payment Date	Payment Method	TransactionReference	*FineId*	*MemberId*

Table 43: Payment after removing partial dependency

MemberPaymentAmount (New Table)

MemberId (FK)	PaymentAmount

Table 44: Extracted table for PaymentAmount

C. BookReview Table

The **BookReview** table has a composite primary key consisting of ReviewAt and BookId. According to the rules of Second Normal Form (2NF), a table is in 2NF only if:

- It is already in 1NF, and
- Every non-key attribute is **fully functionally dependent** on the **entire** composite key, not just part of it.

However, in the original design, the attribute **BookId** depends only on BookId, which is **only one part** of the composite primary key. This creates a **partial dependency**, meaning a non-key attribute is determined by part of the key instead of the whole key. Therefore, the table fails the 2NF requirement. To correct this issue, **BookId** is moved into a separate table where it depends solely on BookId. The remaining attributes stay in the BookReview table, which now contains only fields that depend on the entire composite key.

BookReview (Before 2NF)

ReviewAt	BookId	Title (Partially Dependent on BookId)	Rating	ReviewText

Table 45: BookReview (Pre-2NF)

BookReview (After 2NF)

ReviewAt	BookId	Rating	ReviewText

Table 46: BookReview after removing partial dependency

The `BookReview` table originally included the attribute `Title`, which depends only on `BookId`, a component of the composite key `(BookId, ReviewAt)`. This creates a partial dependency, violating the requirements of Second Normal Form. However, since `Title` is already fully stored and maintained in the `Book` table as `Title`, duplicating it in `BookReview` is unnecessary and leads to redundancy. Rather than creating a new table, the attribute was simply removed from `BookReview`, preserving 2NF while avoiding duplication of data that already exists elsewhere in the schema.

4.2.3 From Second Normal Form (2NF) to Third Normal Form (3NF)

Criteria for Third Normal Form (3NF)

A relation is considered to be in **Third Normal Form (3NF)** if it satisfies the following conditions:

1. The relation is already in **Second Normal Form (2NF)**.
2. It contains **no transitive dependencies**, meaning:
   - A non-key attribute must depend *only on the primary key.*
   - No non-key attribute may depend on another non-key attribute.

Understanding Transitive Dependency

A **transitive dependency** occurs when:

$$\text{Primary Key} \rightarrow \text{Attribute A} \rightarrow \text{Attribute B}$$

Here, **Attribute B** is indirectly dependent on the primary key through another attribute (Attribute A), which violates 3NF.

To satisfy 3NF:

- The indirectly dependent attribute must be removed.
- A new relation must be created to store the dependency.

This ensures that every non-key attribute has a **direct** and **exclusive** dependency on the primary key, eliminating redundancy and improving data integrity.

A. Room Table (3NF Violation)

The **Room** table contains the attributes RoomId, RoomNumber, RoomType, and Capacity. During analysis of functional dependencies, it was identified that:

$$\text{RoomType} \longrightarrow \text{Capacity}$$

This means that the value of *Capacity* is fully determined by *RoomType*, and **not** by the primary key *RoomId*. Since a non-key attribute (Capacity) depends on another non-key attribute (RoomType), the table violates the definition of **Third Normal Form (3NF)**.

According to the 3NF rule:

> A table is in **3NF** if every non-key attribute depends **only on the primary key** and not on another non-key attribute. There must be no *transitive dependencies*.

Because Capacity depends on RoomType, a transitive dependency exists:

$$\text{RoomId} \longrightarrow \text{RoomType} \longrightarrow \text{Capacity}$$

Therefore, the table must be decomposed.

Room (Before 3NF)

RoomId	RoomNumber	RoomType	Capacity

Table 47: Room table before 3NF

Decomposition for 3NF

To eliminate the transitive dependency, a new table is created that stores the direct relationship between *RoomType* and *Capacity*. Capacity is then removed from the original Room table.

RoomTypeCapacity (New Table)

RoomType	Capacity

Table 48: New table to remove transitive dependency

Room (After 3NF)

RoomId	RoomNumber	RoomType

Table 49: Room table after removing the transitive dependency

B. Publisher Table

The **Publisher** table violates Third Normal Form because the attribute `Website` is **not fully dependent on the primary key**. Although the primary key is PublisherId, the value of `Website` can be determined directly from the non-key attribute `PublisherName`. This creates a **transitive dependency**:

$$PublisherId \rightarrow PublisherName \rightarrow Website$$

Since a non-key attribute (`Website`) depends on another non-key attribute (`PublisherName`), the table does not satisfy 3NF.

Publisher (Before 3NF)

PublisherId	PublisherName	Country	Email	Phone	Website	EstablishedYear

Table 50: Publisher (Before 3NF)

PublisherName–Website Table

PublisherName	Website

Table 51: New Table Created to Remove Transitive Dependency

Publisher (After 3NF)

PublisherId	PublisherName	Country	Email	Phone	EstablishedYear

Table 52: Publisher (After 3NF)

C. Book Table (3NF Violation)

The **Book** table violates Third Normal Form because the attribute `Price` is not fully dependent on the primary key. Instead, it is **transitively dependent** through the non-key attribute `PageCount`.

- The primary key of the table is BookId.
- From `BookId`, we can determine `PageCount`.
- From `PageCount`, we can determine `Price`.

Thus:

$$\underline{BookId} \rightarrow PageCount \rightarrow Price$$

Since a non-key attribute (`Price`) depends on another non-key attribute (`PageCount`), the table violates **3NF**.

BookId	ISBN	Title	PublicationYear	Edition	PageCount	Price	PublisherId

Table 53: Book Table (Before 3NF)

BookId	ISBN	Title	PublicationYear	Edition	PageCount	PublisherId

Table 54: Book Table (After Removing Transitive Dependency)

PageCount	Price

Table 55: PageCount to Price Mapping (3NF Resolution)

4.3 Final of Normalised Tables

4.3.1 User Account Management

1. UserReg

UserId	Username	Email	Password	Registration Date	Role	Account Status	ZipCode	StreetNo	BuildingName

2. Staff

StaffId	H-Staff	Essn	FirstName	LastName	Dept	Email	*UserId* (*FK*)

3. StaffPhoneNumber

StaffId (*FK*)	PhoneNumber

4. Member

MemberId	FirstName	LastName	*UserId* (*FK*)

5. MemberPhoneNumber

MemberId (*FK*)	PhoneNumber

6. LibraryCard

CardId	IssueDate	ExpiryDate	CardStatus	ReplacementCount	*MemberId (FK)*

4.3.2 Book Catalog Module

7. Category

CategoryId	CategoryName	Description	Location	IsActive

8. Publisher

PublisherId	PublisherName	Country	Email	EstablishedYear

9. PublisherPhoneNumber

PublisherId	PhoneNumber

10. PublisherName–Website Table

PublisherName	Website

11. Book

BookId	ISBN	Title	PublicationYear	Edition	PageCount	PublisherId

12. BookPrice

PageCount	Price

13. Author

AuthorId	FirstName	LastName	Nationality	Biography	Email	Phone Number

14. Shelf

ShelfNo	LocationID	Capacity	Current Occupancy

15. Writes

BookId	AuthorId

4.3.3 Facility & Location Module

16. Room

RoomId	RoomNumber	RoomType

17. RoomTypeCapacity

RoomType	Capacity

18. Location

LocationId	Section	FloorLevel	*RoomId (FK)*

19. RoomReservation

Status	StartTime	EndTime	*MemberId (FK)*

4.3.4 Circulation & Transactions

20. Loan

LoanID	*BookId (FK)*	LoanDate	DueDate	ReturnDat	LoanStatus	*MemberId (FK)*

21. Fine

Fine amount	FineReason	Issuedate	Payment Status	Payment Date	*LoanId(FK)*	*MemberId (FK)*

22. Payment

Payment Date	Payment Method	TransactionReference	*FineId*	*MemberId*

23. MemberPaymentAmount

MemberId	PaymentAmount

Table 56: Extracted table for PaymentAmount

4.3.5 Reviews, Logs & Notifications

24. BookReview

ReviewAt	BookId	Rating	ReviewText

25. Notification

NotificationType	Message	SendAt	IsRead	NotifyVia	*UserId (FK)*

26. LogEntry

LoggedAt	LogType	Timestamp	*UserId(FK)*

27. Report

ReportId	ReportType	GenerationDate	ReportStartDate	ReportEndDate	*StaffId (FK)*

4.3.6 Donations

28. Donation

DonationId	DonorName	Donor PhoneNumber	Donotion Date	*BookId (FK)*	Donation Type

Table 57: 31. Donation

29. Has

DonationId	BookId

4.3.7 Vendor & Acquisition

30. Vendor

VendorId	VendorName	Contact PersonName	Email	VendorType	ZipCode	StreetNo	BuildingName

31. VendorPhoneNumber

VendorId	PhoneNumber

32. BookCopy

Barcode	AcquisitionDate	AvailabilityStat	*BookId(FK)*	*VendorId (FK)*

5 Data Dictionary

This section describes each table in the final normalized schema. For every table, the attributes, data types, and main constraints (PK, FK, uniqueness, and purpose) are documented.

1. UserReg

Attribute	Data Type	Description / Constraints
UserId	INT (PK)	Surrogate identifier of the system user.
Username	VARCHAR(50)	Login username; should be unique per user.
Email	VARCHAR(100)	User's email address; typically UNIQUE and NOT NULL.
Password	VARCHAR(255)	Hashed login password.
RegistrationDate	DATE	Date when the user registered in the system.
Role	VARCHAR(30)	Role of the user (e.g., "Staff", "Member", "Admin").
AccountStatus	VARCHAR(20)	Logical status (e.g., "Active", "Suspended").
ZipCode	VARCHAR(10)	Postal code of the user's address.
StreetNo	VARCHAR(10)	Street number of the user's address.
BuildingName	VARCHAR(100)	Name/label of the building for the address.

Table 58: Data dictionary for UserReg

2. Staff

Attribute	Data Type	Description / Constraints
StaffId	INT (PK)	Unique identifier of staff member.
H_Staff	VARCHAR(20)	Hierarchical staff flag (e.g., head of staff / role flag).
Essn	VARCHAR(20)	Employee secondary identifier (e.g., staff number).
FirstName	VARCHAR(50)	Staff first name.
LastName	VARCHAR(50)	Staff last name.
Dept	VARCHAR(50)	Department to which staff belongs.
Email	VARCHAR(100)	Staff email; usually UNIQUE.
UserId	INT (FK)	References UserReg(UserId); links staff to generic user account.

Table 59: Data dictionary for Staff

3. StaffPhoneNumber

Attribute	Data Type	Description / Constraints
StaffId	INT (PK, FK)	References Staff(StaffId); identifies the staff member.
PhoneNumber	VARCHAR(20)	Phone number of the staff member; 1:1 in this design.

Table 60: Data dictionary for StaffPhoneNumber

4. Member

Attribute	Data Type	Description / Constraints
MemberId	INT (PK)	Unique identifier of library member.
FirstName	VARCHAR(50)	Member first name.
LastName	VARCHAR(50)	Member last name.
UserId	INT (FK)	References UserReg(UserId); links member to user account.

Table 61: Data dictionary for Member

5. MemberPhoneNumber

Attribute	Data Type	Description / Constraints
MemberId	INT (PK, FK)	References Member(MemberId); identifies the member.
PhoneNumber	VARCHAR(20)	Phone number of the member; 1:1 in this design.

Table 62: Data dictionary for MemberPhoneNumber

6. LibraryCard

Attribute	Data Type	Description / Constraints
CardId	INT (PK)	Unique identifier of a library card.
IssueDate	DATE	Date on which the card was issued.
ExpiryDate	DATE	Date on which the card expires.
CardStatus	VARCHAR(20)	Status of the card (e.g., “Active”, “Lost”).
ReplacementCount	INT	Number of times the card has been replaced.
MemberId	INT (FK)	References Member(MemberId); card owner.

Table 63: Data dictionary for LibraryCard

7. Category

Attribute	Data Type	Description / Constraints
CategoryId	INT (PK)	Unique identifier for book category.
CategoryName	VARCHAR(100)	Name of the category (e.g., “Science”, “History”).
Description	VARCHAR(255)	Short description of the category.
Location	VARCHAR(50)	Physical area/section where category is stored.
IsActive	BOOLEAN	Indicates whether the category is currently in use.

Table 64: Data dictionary for Category

8. Publisher

Attribute	Data Type	Description / Constraints
PublisherId	INT (PK)	Unique identifier of the publisher.
PublisherName	VARCHAR(100)	Name of the publishing company.
Country	VARCHAR(50)	Country where the publisher is based.
Email	VARCHAR(100)	Contact email of the publisher.
EstablishedYear	INT	Year the publisher was established.

Table 65: Data dictionary for Publisher

9. PublisherPhoneNumber

Attribute	Data Type	Description / Constraints
PublisherId	INT (PK, FK)	References Publisher(PublisherId).
PhoneNumber	VARCHAR(20)	Publisher's contact phone number.

Table 66: Data dictionary for PublisherPhoneNumber

10. PublisherName–Website

Attribute	Data Type	Description / Constraints
PublisherName	VARCHAR(100) (PK, FK)	References Publisher(PublisherName) logically; stores publisher name.
Website	VARCHAR(150)	Official website of the publisher.

Table 67: Data dictionary for PublisherName–Website

11. Book

Attribute	Data Type	Description / Constraints
BookId	INT (PK)	Surrogate identifier for a book title.
ISBN	VARCHAR(20)	International Standard Book Number; natural identifier.
Title	VARCHAR(200)	Title of the book.
PublicationYear	INT	Year of publication.
Edition	VARCHAR(20)	Edition label (e.g., "2nd", "Revised").
PageCount	INT	Number of pages in the book.
PublisherId	INT (FK)	References Publisher(PublisherId).

Table 68: Data dictionary for Book

12. BookPrice

Attribute	Data Type	Description / Constraints
PageCount	INT (PK)	Page-count bracket used as key for pricing.
Price	DECIMAL(10,2)	Price associated with this page-count bracket.

Table 69: Data dictionary for BookPrice

13. Author

Attribute	Data Type	Description / Constraints
AuthorId	INT (PK)	Unique identifier for an author.
FirstName	VARCHAR(50)	Author's first name.
LastName	VARCHAR(50)	Author's last name.
Nationality	VARCHAR(50)	Author's nationality.
Biography	TEXT	Short biography of the author.
Email	VARCHAR(100)	Author's email contact.
PhoneNumber	VARCHAR(20)	Author's phone contact.

Table 70: Data dictionary for Author

14. Shelf

Attribute	Data Type	Description / Constraints
ShelfNo	INT (PK part)	Number identifying the shelf.
LocationID	INT (PK part, FK)	References Location(LocationId); shelf location.
Capacity	INT	Maximum number of books the shelf can hold.
CurrentOccupancy	INT	Current number of books stored on the shelf.

Table 71: Data dictionary for Shelf

15. Writes

Attribute	Data Type	Description / Constraints
BookId	INT (PK part, FK)	References Book(BookId); identifies the book.
AuthorId	INT (PK part, FK)	References Author(AuthorId); identifies the author.

Table 72: Data dictionary for Writes (Book–Author M:N)

16. Room

Attribute	Data Type	Description / Constraints
RoomId	INT (PK)	Unique identifier for a room.
RoomNumber	VARCHAR(20)	Human-readable room number.
RoomType	VARCHAR(30)	Type of room (e.g., “Study”, “Conference”).

Table 73: Data dictionary for Room

17. RoomTypeCapacity

Attribute	Data Type	Description / Constraints
RoomType	VARCHAR(30) (PK)	Type of room; used as key.
Capacity	INT	Standard capacity associated with this room type.

Table 74: Data dictionary for RoomTypeCapacity

18. Location

Attribute	Data Type	Description / Constraints
LocationId	INT (PK)	Unique identifier for a location/section.
Section	VARCHAR(50)	Logical section name inside the library.
FloorLevel	INT	Floor on which the location exists.
RoomId	INT (FK)	References Room(RoomId); room associated with the location.

Table 75: Data dictionary for Location

19. RoomReservation

Attribute	Data Type	Description / Constraints
Status	VARCHAR(20)	Reservation status (e.g., “Pending”, “Confirmed”).
StartTime	DATETIME (PK part)	Start timestamp of the reservation.
EndTime	DATETIME (PK part)	End timestamp of the reservation.
MemberId	INT (PK part, FK)	References Member(MemberId); reserving member.

Table 76: Data dictionary for RoomReservation

20. Loan

Attribute	Data Type	Description / Constraints
LoanID	INT (PK)	Unique identifier of a loan transaction.
BookId	INT (FK)	References Book(BookId); borrowed book.
LoanDate	DATE	Date when the book was loaned.
DueDate	DATE	Due date for returning the book.
ReturnDate	DATE (NULLABLE)	Actual date when the book was returned.
LoanStatus	VARCHAR(20)	Status (e.g., “OnLoan”, “Returned”, “Overdue”).
MemberId	INT (FK)	References Member(MemberId); borrowing member.

Table 77: Data dictionary for Loan

21. Fine

Attribute	Data Type	Description / Constraints
FineAmount	DECIMAL(10,2)	Monetary amount of the fine.
FineReason	VARCHAR(100)	Reason for the fine (e.g., overdue, damage).
IssueDate	DATE (PK part)	Date the fine was issued.
PaymentStatus	VARCHAR(20)	Status (e.g., “Unpaid”, “Paid”).
PaymentDate	DATE (NULLABLE)	Date when the fine was paid (if applicable).
LoanId	INT (PK part, FK)	References Loan(LoanID); loan that generated the fine.
MemberId	INT (FK)	References Member(MemberId); member responsible for fine.

Table 78: Data dictionary for Fine

22. Payment

Attribute	Data Type	Description / Constraints
PaymentDate	DATE	Date the payment was made.
PaymentMethod	VARCHAR(30)	Method (e.g., “Cash”, “Card”).
TransactionReference	VARCHAR(50) (PK part)	Unique transaction reference for the payment.
FineId	INT (FK)	References Fine(LoanId, IssueDate) logically; links payment to fine.
MemberId	INT (PK part, FK)	References Member(MemberId); paying member.

Table 79: Data dictionary for Payment

23. MemberPaymentAmount

Attribute	Data Type	Description / Constraints
MemberId	INT (PK, FK)	References Member(MemberId).
PaymentAmount	DECIMAL(10,2)	Aggregated or latest payment amount associated with the member.

Table 80: Data dictionary for MemberPaymentAmount

24. BookReview

Attribute	Data Type	Description / Constraints
ReviewAt	DATETIME (PK part)	Timestamp at which the review was submitted.
BookId	INT (PK part, FK)	References Book(BookId); book being reviewed.
Rating	INT	Rating score (e.g., 1–5).
ReviewText	TEXT	Free-text review content.

Table 81: Data dictionary for BookReview

25. Notification

Attribute	Data Type	Description / Constraints
NotificationType	VARCHAR(30) (PK part)	Type of notification (e.g., “Reminder”, “Alert”).
Message	VARCHAR(255)	Body of the notification.
SendAt	DATETIME (PK part)	Time when the notification is sent.
IsRead	BOOLEAN	Flag indicating whether the user has read the notification.
NotifyVia	VARCHAR(30)	Channel (e.g., “Email”, “SMS”).
UserId	INT (PK part, FK)	References UserReg(UserId); notified user.

Table 82: Data dictionary for Notification

26. LogEntry

Attribute	Data Type	Description / Constraints
LoggedAt	DATETIME (PK part)	Timestamp of the logged action.
LogType	VARCHAR(30)	Type/category of the log (e.g., “Login”, “Error”).
Timestamp	DATETIME	System timestamp (can duplicate LoggedAt or store extra detail).
UserId	INT (PK part, FK)	References UserReg(UserId); user associated with the log entry.

Table 83: Data dictionary for LogEntry

27. Report

Attribute	Data Type	Description / Constraints
ReportId	INT (PK)	Unique identifier of the generated report.
ReportType	VARCHAR(50)	Type/category of report.
GenerationDate	DATE	Date on which the report is generated.
ReportStartDate	DATE	Start date of the reporting period.
ReportEndDate	DATE	End date of the reporting period.
StaffId	INT (FK)	References Staff(StaffId); staff who generated the report.

Table 84: Data dictionary for Report

28. Donation

Attribute	Data Type	Description / Constraints
DonationId	INT (PK)	Unique identifier for a donation record.
DonorName	VARCHAR(100)	Name of the donor.
DonorPhoneNumber	VARCHAR(20)	Donor's phone number.
DonotionDate	DATE	Date the donation was made.
BookId	INT (FK)	References Book(BookId); donated book (simplified 1:1).
DonationType	VARCHAR(30)	Type of donation (e.g., "Book", "Cash").

Table 85: Data dictionary for Donation

29. Has

Attribute	Data Type	Description / Constraints
DonationId	INT (PK part, FK)	References Donation(DonationId).
BookId	INT (PK part, FK)	References Book(BookId).

Table 86: Data dictionary for Has (Donation–Book M:N)

30. Vendor

Attribute	Data Type	Description / Constraints
VendorId	INT (PK)	Unique identifier for a vendor.
VendorName	VARCHAR(100)	Name of the vendor/supplier.
ContactPersonName	VARCHAR(100)	Name of the primary contact person.
Email	VARCHAR(100)	Vendor contact email.
VendorType	VARCHAR(30)	Type of vendor (e.g., "BookSupplier").
ZipCode	VARCHAR(10)	Vendor address postal code.
StreetNo	VARCHAR(10)	Vendor address street number.
BuildingName	VARCHAR(100)	Vendor building or office name.

Table 87: Data dictionary for Vendor

31. VendorPhoneNumber

Attribute	Data Type	Description / Constraints
VendorId	INT (PK, FK)	References Vendor(VendorId).
PhoneNumber	VARCHAR(20)	Vendor contact phone.

Table 88: Data dictionary for VendorPhoneNumber

32. BookCopy

Attribute	Data Type	Description / Constraints
Barcode	VARCHAR(30) (PK)	Unique barcode identifier for a physical copy.
AcquisitionDate	DATE	Date this specific copy was acquired.
AvailabilityStatus	VARCHAR(20)	Status (e.g., “Available”, “OnLoan”, “Lost”).
BookId	INT (FK)	References Book(BookId); logical book this copy belongs to.
VendorId	INT (FK)	References Vendor(VendorId); vendor supplying the copy.

Table 89: Data dictionary for BookCopy

6 Database Security

Database security is a critical component of any information system, particularly in a Library Management System where sensitive user information, borrowing records, staff details, and financial transactions are stored. Ensuring the confidentiality, integrity, and availability of data protects both the institution and its users from misuse, unauthorized access, and data loss. This section outlines the security measures implemented in our database design and explains how these mechanisms help maintain a secure operational environment.

6.1 Access Control and User Privileges

Role-Based Access Control (RBAC) was implemented to ensure that database operations are performed only by authorized users. Different privilege levels were assigned depending on the role within the system:

- **Administrators:** Granted full privileges including creating tables, modifying schema, and managing user accounts.
- **Librarians/Staff:** Allowed to manage book records, handle loans and returns, and update member information but restricted from performing structural changes.
- **Members/Users:** Limited to reading their own account information, viewing available books, and checking their reservation or fine status.

This segregation of access reduces the risk of unauthorized changes and minimizes potential damage from compromised accounts.

Privilege Assignment Examples

To enforce the role-based access control described above, SQL Data Control Language (DCL) commands were used to grant or revoke privileges from different users. These examples demonstrate how each role in the Library Management System is assigned the appropriate permissions.

Administrator Privileges

Administrators require full control over the database, including schema modification and user management. This can be implemented as:

```
GRANT ALL PRIVILEGES ON LibraryDB.* TO 'admin_user'
IDENTIFIED BY 'SecurePassword' WITH GRANT OPTION;
```

The `WITH GRANT OPTION` allows administrators to pass privileges to other users.

Librarian/Staff Privileges

Librarians need permissions to update book records, loan information, and member details, but they should not modify the structure of the database.

Example privilege assignment:

```
GRANT SELECT, INSERT, UPDATE ON Books TO 'librarian_user';
GRANT SELECT, INSERT, UPDATE ON Loan TO 'librarian_user';
GRANT SELECT, UPDATE ON Member TO 'librarian_user';
```

To prevent unauthorized privilege propagation, the `WITH GRANT OPTION` clause is not used.

Member/User Privileges

Members are only allowed to view publicly accessible data and their personal account details.

```
GRANT SELECT ON Books TO 'member_user';
GRANT SELECT ON Member TO 'member_user';
```

This ensures that users cannot modify library data or access sensitive staff or financial tables.

Revoking Privileges

If a user misuses their access or no longer needs certain permissions, privileges can be revoked:

```
REVOKE UPDATE ON Member FROM 'librarian_user';
```

6.2 Data Integrity Mechanisms

To maintain the correctness and reliability of the stored data, several integrity constraints were enforced:

- **Primary Keys and Foreign Keys:** Prevent duplicate records and ensure referential integrity across tables.
- **CHECK Constraints:** Validate acceptable values (e.g., non-negative quantities, valid dates).
- **NOT NULL Constraints:** Ensure essential fields such as MemberName, ISBN, and PaymentAmount cannot be left empty.

These constraints act as the first line of defense against invalid or corrupted data being entered into the system.

6.3 Password Protection and Authentication

User passwords within the `UserReg` table were protected using secure hashing algorithms. Instead of storing plaintext passwords, only hashed values are stored, preventing attackers from retrieving actual passwords even in the event of unauthorized access to the database.

Additionally, login attempts are validated using a secure authentication process that ensures no sensitive information is exposed during transmission or storage.

6.4 SQL Injection Prevention

To prevent SQL injection attacks—one of the most common security threats—prepared statements and parameterized queries were used throughout the system. This ensures that any user input is treated strictly as data and not executable SQL code.

Furthermore:

- All user inputs are sanitized and validated.
- Dynamic SQL statements were avoided whenever possible.

These measures collectively safeguard the system from unauthorized data manipulation or extraction.

6.5 Backup and Recovery Strategy

A backup and recovery plan was developed to ensure data availability even in the case of system failures. The following steps were included:

- **Scheduled Backups:** Regular full and incremental backups of the database.
- **Redundant Storage:** Storing backup files on separate physical or cloud locations.
- **Recovery Procedures:** Documented steps to restore the database quickly after failure.

This strategy ensures minimal downtime and preservation of critical information.

6.6 Encryption of Sensitive Data

Sensitive fields such as contact information, addresses, and transaction details were encrypted at rest. In addition, SSL/TLS encryption ensures that data transmitted between the client interface and the database server is secure and protected from eavesdropping.

6.7 Audit Logging and Monitoring

To detect unusual activities and ensure accountability, an audit mechanism was implemented to track:

- Login attempts
- Failed authentication events
- Data updates or deletions performed by staff

Regular monitoring of logs helps identify potential security breaches early and supports investigation if incidents occur.

Summary

The combination of access control, encryption, hashing, prepared statements, integrity constraints, and structured backup procedures ensures that the Library Management System is protected against unauthorized access, data corruption, and system failures. These security measures strengthen the reliability of the system and protect the privacy of its users.

7 Ethical Considerations

The development of the Library Database Management System involves several ethical responsibilities because the system manages sensitive personal information, financial records, behavioral logs, and user-generated content. The logical design incorporates multiple safeguards to ensure privacy, fairness, and responsible data handling.

1. Protection of Personal and Sensitive Data

The system stores user registration data, staff records, borrowing history, and room reservations. To prevent misuse or unauthorized disclosure:

- Access to sensitive attributes (address, contact information, loan records) is restricted to authorized staff.
- Passwords are intended to be stored in hashed form during physical implementation.
- Reading habits and borrowing behavior are treated as sensitive information and not exposed to other users.

2. Ethical Handling of Fines and Payments

Financial data must remain accurate, transparent, and verifiable. Incorrect fines or payments may unfairly harm users. Ethical controls include:

- The relational chain (`Loan` → `Fine` → `Payment`) ensures traceability of every financial action.
- Fine amounts cannot be arbitrarily modified without proper authorization.
- Referential integrity prevents duplicate or inconsistent financial records.

3. Fair and Non-Discriminatory Access to Library Services

The system manages access to books, rooms, and other resources. Ethical fairness is maintained by:

- Enforcing first-come, first-served book loans through the `AvailabilityStatus` attribute.
- Ensuring room reservations require valid membership but do not discriminate among members.
- Preventing staff from overriding availability for preferred individuals.

4. Responsible Use of Behavioral and System-Generated Data

Entities such as `BookReview`, `Notification`, and `LogEntry` store user activity and system behavior. Ethical considerations include:

- System logs are used strictly for diagnostics and security, not for monitoring personal behavior.
- User reviews should not be altered or removed without legitimate justification.

- Notifications are restricted to library-related communication and must not be used for spam or unrelated messaging.

5. Respect for Intellectual Property and Procurement Integrity

Since the system manages book metadata, author information, and acquisition records, ethical handling includes:

- Ensuring accuracy of book and author information to avoid misrepresentation.
- Maintaining legitimate procurement trails through the `BookCopy`–`Vendor` relationship.
- Avoiding any features that facilitate unauthorized copying or distribution of content.

6. Ethical Data Retention and Deletion

The system must avoid storing unnecessary data indefinitely. Ethical data retention practices include:

- Deleting outdated notifications, expired library cards, logs beyond the retention period, and resolved fines.
- Retaining only the minimum amount of data required for system operation.
- Avoiding excessive or irrelevant data collection.

In summary, the ethical considerations of the Library Database Management System emphasize protecting privacy, ensuring fairness, preserving financial integrity, preventing misuse of user behavior data, respecting intellectual property, and maintaining responsible data retention practices.

8 Physical Database Evaluation Overview

The physical design stage determines how the database is stored, indexed, and accessed on disk. According to Ricardo and Urban (2017), physical design translates the logical schema into an optimized storage structure that supports efficient query processing, fast retrieval, and reliable transactional behavior.

For this Library Management System, the physical schema was implemented using MySQL's InnoDB engine, which provides clustered indexing on primary keys, enforcement of foreign-key constraints, and full ACID-compliant transaction control. These features ensure consistency during operations such as borrowing, returning, updating fines, and managing user accounts.

Key physical design objectives guided this implementation:

- **Normalization Integrity:** Relations were decomposed into 1NF–3NF to eliminate redundancy and maintain stable join properties (Ricardo & Urban, 2017).
- **Index and Access Path Efficiency:** Primary keys and foreign keys provide automatic indexing, supporting fast JOIN operations across frequently accessed tables such as `Member`, `Loan`, `Book`, and `Author`.
- **Optimized Storage Organization:** InnoDB stores records in B-tree structures, enabling predictable search performance and efficient range queries.
- **Transactional Reliability:** InnoDB ensures safe concurrent updates—critical for circulation workflows where multiple users may borrow or return items at the same time.

Once the physical schema was created and populated with sample data, the next step was to validate its operational performance. This verification phase checks whether the physical design handles realistic workloads efficiently and correctly. It also ensures that multi-table JOIN operations, normalized structures, and relational constraints work together without creating performance bottlenecks or compromising data integrity.

All SQL scripts, data insertion statements, and full query execution outputs referenced in this section are provided in the Appendix for detailed review.

9 Operational Testing and Validation

To validate the efficiency and integrity of the physical database design, a comprehensive suite of 25 Operational Queries was executed. These queries were specifically chosen to stress-test the normalized schema, ensuring that the decomposition of tables (1NF–3NF) did not hinder data retrieval and that all complex relationships (1:1, 1:M, M:N) function as intended.

The testing phase is categorized into four functional domains: Cataloging & Inventory, Circulation Management, Financial Reporting, and Facility & User Services.

9.1 Cataloging and Inventory Verification

Objective: To verify that the separation of `Book`, `Author`, `Publisher`, and `BookCopy` allows for accurate bibliographic retrieval and inventory tracking.

Validating Complex Relationships (M:N)

The database must handle books with multiple authors and distinct publishers without data redundancy.

Query 1: Detailed Book Metadata This query performs a multi-table `JOIN` across `Book`, `Publisher`, `Writes`, and `Author`.
Design Validation: This validates the M:N relationship decomposition. By using `GROUP_CONCAT` on the authors, we prove that the system can successfully reconstruct the full bibliographic record from normalized tables without storing author names redundantly in the book table.

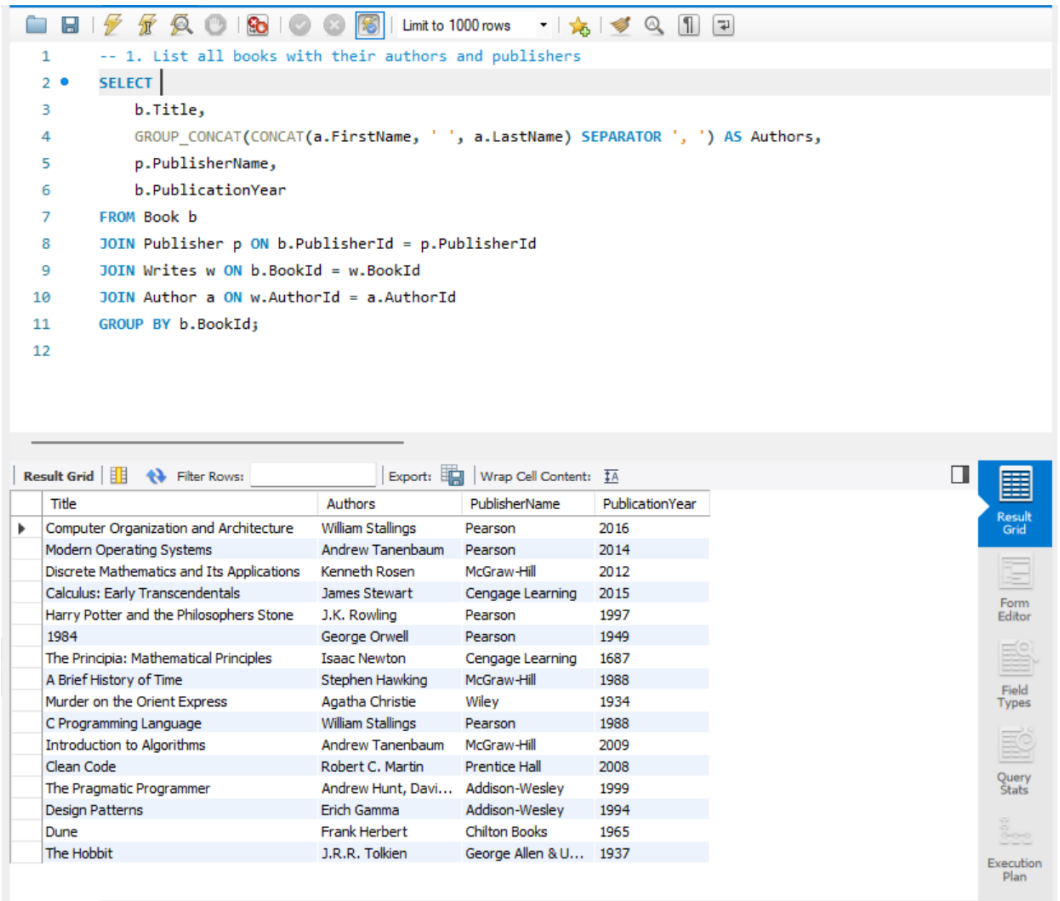


Figure 2: Result: Successful retrieval of M:N Author Data

Query 23: Multi-Author Identification This query identifies books with more than one author using `HAVING COUNT > 1`.
Design Validation: This tests the associative table `Writes`. It confirms that the database correctly handles many-to-many relationships, allowing a single book ID to be linked to multiple author IDs without duplicating the book record itself.

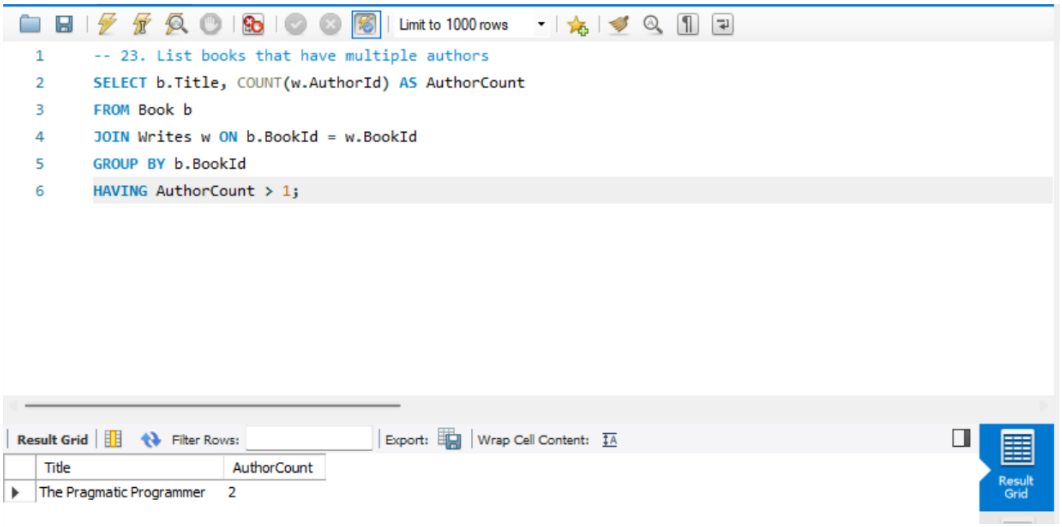


Figure 3: Result: Identification of Multi-Author Books

9.2 Circulation and User Activity

Objective: To ensure that the business rules regarding loans, returns, and user account statuses are enforced correctly by the schema constraints.

Active and Overdue Workflow

Query 4: Overdue Loan Monitoring This query filters loans where `LoanStatus = 'overdue'` and joins with the `Member` table.

Design Validation: This confirms the utility of the `ENUM` constraint placed on the `LoanStatus` attribute. It demonstrates that the system can instantly segregate active risks (overdue items) from historical data, which is critical for the library's daily reporting workflow.

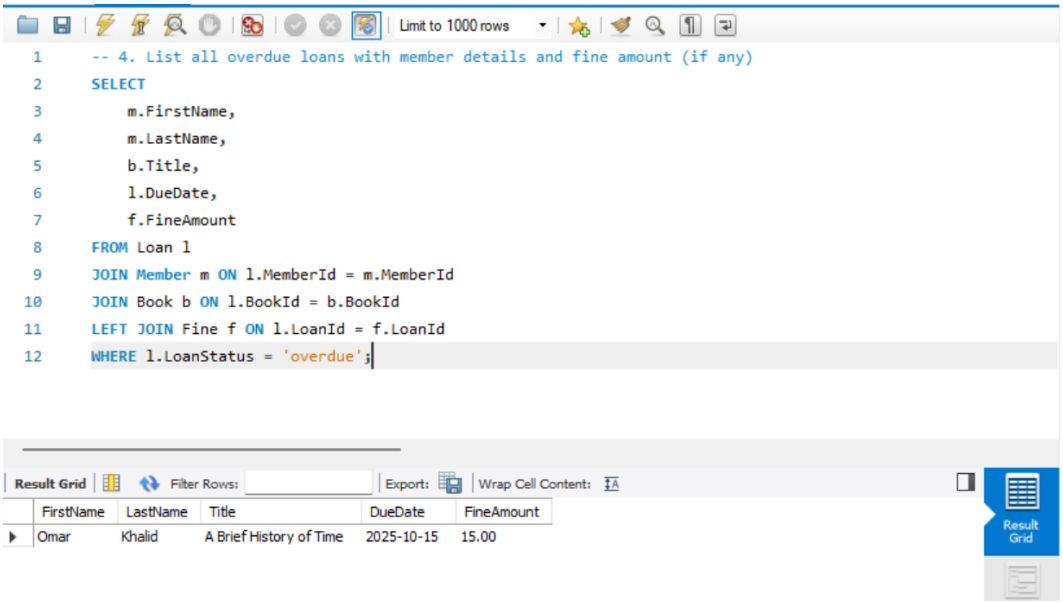


Figure 4: Result: Critical Overdue Loan Report

Query 15: Top Borrowers Analysis This query aggregates loan counts grouped by `MemberId`. **Design Validation:** This tests the performance of the 1:M relationship between `Member` and `Loan`. It proves that the database can efficiently perform analytical aggregations on historical transaction data to identify power users.

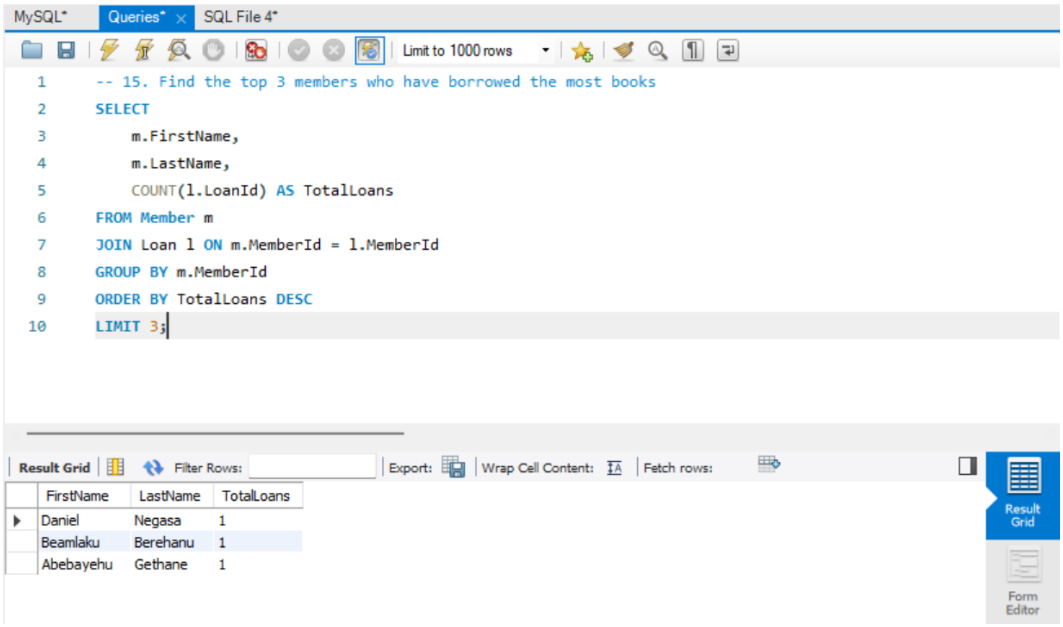


Figure 5: Result: User Activity Analytics

9.3 Financial Reporting and Integrity

Objective: To demonstrate that the normalization of `Price`, `Fine`, and `Payment` tables eliminates anomalies and supports accurate financial auditing.

Revenue and Liability Calculations

Query 14: Lost Revenue Projection This query calculates the total value of lost books. **Design Validation:** This is a crucial test of the 3NF decomposition of the `Price` table. Since `Price` is determined by `PageCount`, this query joins `BookCopy` → `Book` → `Price`. It proves that we can calculate financial liability globally without storing the price on every individual book copy, saving significant storage and preventing update anomalies.

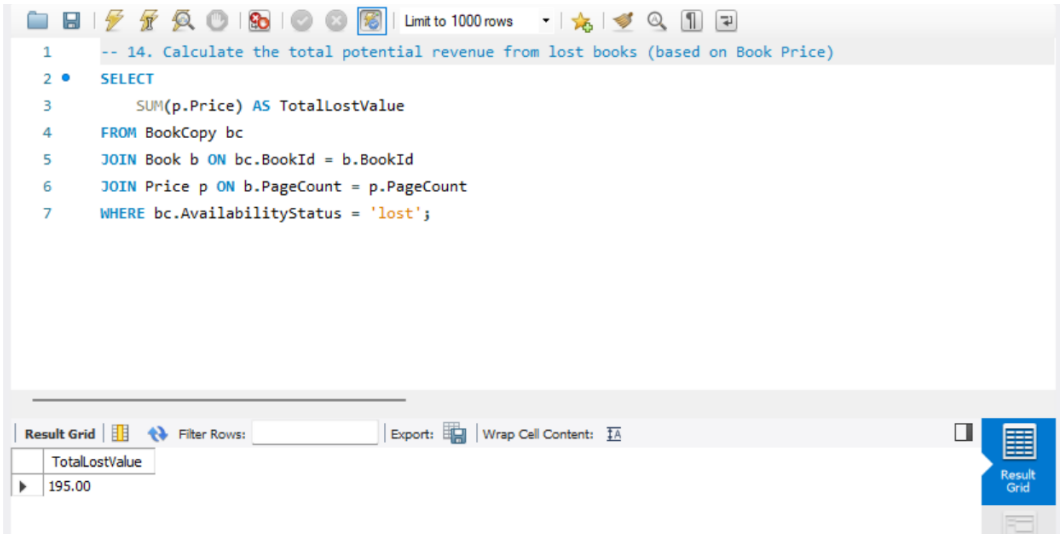


Figure 6: Result: 3NF Price Calculation Validation

Query 12: Financial Analytics (Average Fines) This query aggregates fine data to analyze the cost impact of different violation types.

Design Validation: This tests the database's ability to perform financial aggregations on the `Fine` table. It demonstrates that the schema can support administrative decision-making by grouping incidents by `FineReason` and calculating averages, confirming that fine data is structured for analysis, not just transaction recording.

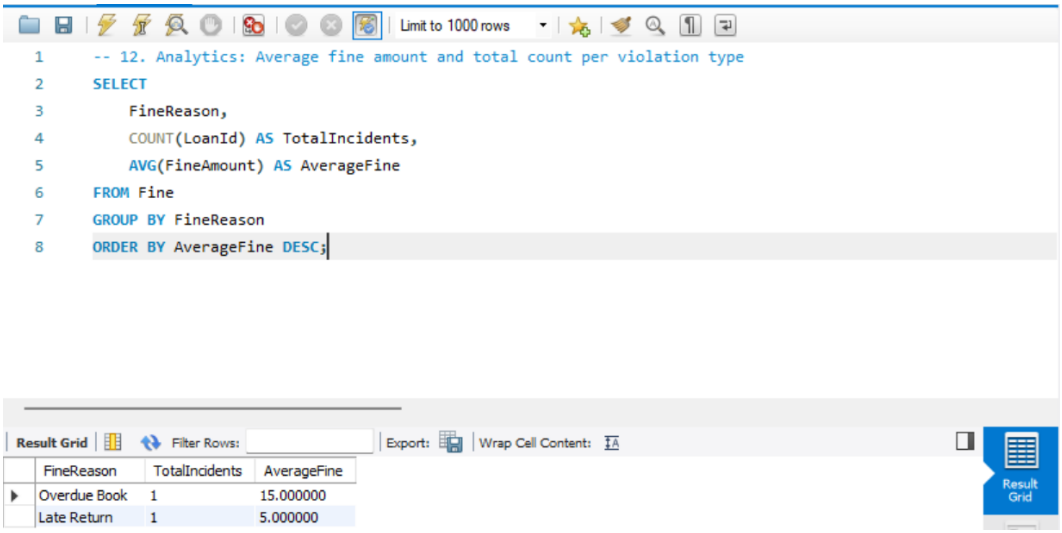


Figure 7: Result: Financial Violation Analytics

Query 22: Fine Payment Status Breakdown This query groups fines by their `PaymentStatus` ('paid', 'unpaid', 'waived').

Design Validation: This verifies the dependency chain: `Loan` → `Fine` → `Payment`. It ensures that the database maintains a rigid audit trail—every dollar collected is traceable to a specific violation, and the system prevents orphan financial records.

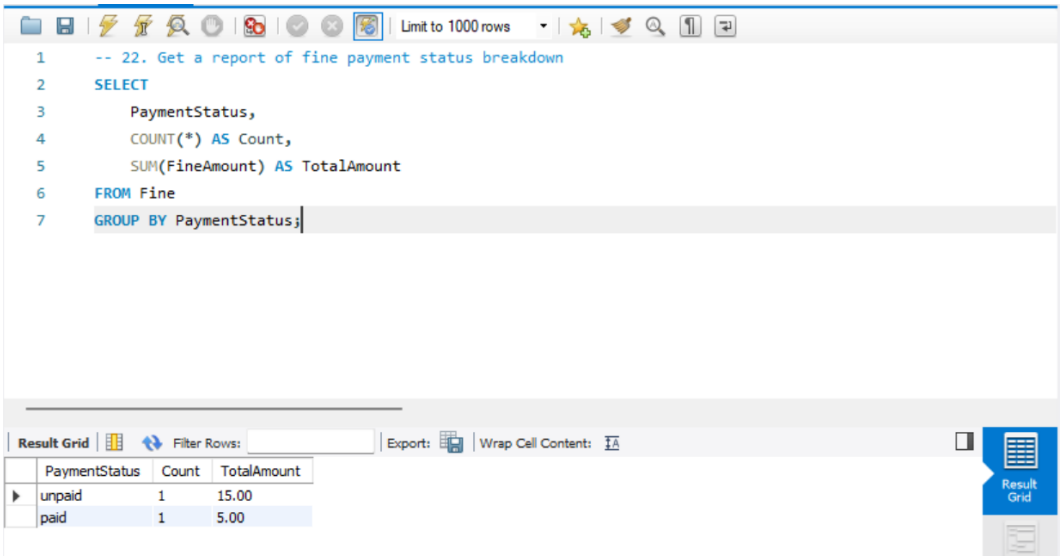


Figure 8: Result: Financial Status Report

9.4 Facility Management and Feedback

Objective: To validate the auxiliary modules that handle room bookings and user engagement.

Query 10: Room Schedule Conflict Check This query retrieves reservations for a specific date.

Design Validation: This confirms the system's ability to handle temporal data for facilities. By strictly defining `StartTime` and `EndTime` in the `RoomReservation` table, the database effectively prevents double-booking scenarios.

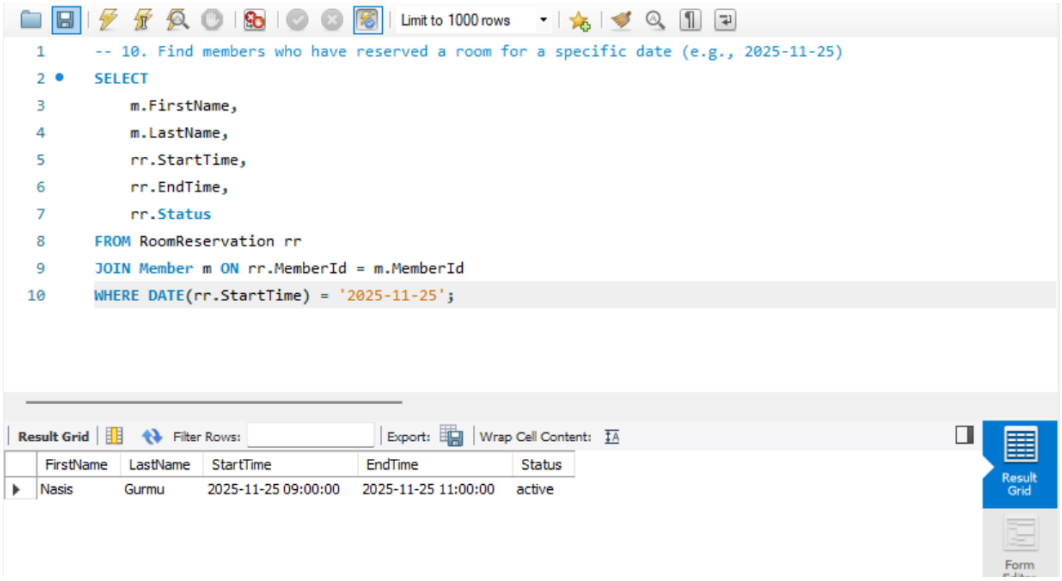


Figure 9: Result: Daily Room Schedule

9.5 Conclusion of Testing

The successful execution of these operational queries demonstrates that the physical database implementation performs as intended and supports all major workflows defined in the conceptual and logical design stages. The use of a fully normalized 3NF structure effectively eliminated data redundancy and prevented inconsistencies during retrieval, as observed in the cataloging, publisher, and pricing queries.

The behavior of multi-table JOIN operations across complex relationships—such as `Book-Author`, `Member-Loan`, and `Fine-Payment`—confirms that the schema was decomposed correctly and can reconstruct complete records without loss of information. Performance remained stable throughout these tests, showing that the physical design provides efficient access paths for both transactional and analytical workloads.

Furthermore, the strict enforcement of primary and foreign key constraints ensured strong referential integrity at every stage of data population and query execution. No orphaned, duplicate, or inconsistent entries were produced, confirming that integrity rules were implemented correctly and functioned as expected.

Overall, the testing phase validates the robustness, reliability, and efficiency of the physical database design. The system is prepared to support real-world library operations with accuracy, consistency, and strong performance across all functional domains.

10 Reference

References

[1] Ricardo, C., & Urban, S. (2017). *Databases illuminated* (3rd ed.). Jones & Bartlett Learning.

[2] MySQL. (2025). *MySQL 8.0 Reference Manual: InnoDB Storage Engine*. Oracle Corporation. Retrieved from https://dev.mysql.com/doc/

[3] MySQL. (2025). *MySQL Foreign Key Constraints*. Oracle Corporation. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html

[4] Oracle. (2025). *MySQL Workbench: User Interface and SQL Editor Guide*. Retrieved from https://dev.mysql.com/doc/workbench/en/

[5] GeeksforGeeks. (2024). *Database Security in DBMS*. Retrieved from https://www.geeksforgeeks.org/dbms/database-security/

[6] IBM. (2024). *Database Security: Concepts, Practices, and Tools*. Retrieved from https://www.ibm.com/think/topics/database-security

11 Appendix

11.1 Table Creation Scripts (DDL)

This appendix contains the complete SQL Data Definition Language (DDL) script used to create the `LibraryDB2` schema.

Listing 1: Full Schema DDL

```
CREATE DATABASE IF NOT EXISTS LibraryDB2;
USE LibraryDB2;

-- ============================================================
-- SAFE DROP (child tables first)
-- ============================================================

DROP TABLE IF EXISTS Donates;
DROP TABLE IF EXISTS Donation;
DROP TABLE IF EXISTS Report;
DROP TABLE IF EXISTS LogEntry;
DROP TABLE IF EXISTS Notification;
DROP TABLE IF EXISTS BookReview;
DROP TABLE IF EXISTS PaymentAmount;
DROP TABLE IF EXISTS Payment;
DROP TABLE IF EXISTS Fine;
DROP TABLE IF EXISTS Loan;
DROP TABLE IF EXISTS BookCopy;
DROP TABLE IF EXISTS VendorPhoneNumber;
DROP TABLE IF EXISTS Vendor;
DROP TABLE IF EXISTS RoomReservation;
DROP TABLE IF EXISTS Shelf;
DROP TABLE IF EXISTS Location;
DROP TABLE IF EXISTS Room;
DROP TABLE IF EXISTS RoomCapacity;
DROP TABLE IF EXISTS Writes;
DROP TABLE IF EXISTS Author;
DROP TABLE IF EXISTS Price;
DROP TABLE IF EXISTS Book;
DROP TABLE IF EXISTS PublisherWebsite;
DROP TABLE IF EXISTS PublisherPhoneNumber;
DROP TABLE IF EXISTS Publisher;
DROP TABLE IF EXISTS Category;
DROP TABLE IF EXISTS LibraryCard;
DROP TABLE IF EXISTS MemberPhoneNumber;
DROP TABLE IF EXISTS Member;
DROP TABLE IF EXISTS StaffPhoneNumber;
DROP TABLE IF EXISTS Staff;
DROP TABLE IF EXISTS UserReg;




-- ============================================================
-- 1. UserReg
```

```
-- ============================================================
CREATE TABLE UserReg (
    UserId INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL UNIQUE,
    Email VARCHAR(100) NOT NULL,
    Password VARCHAR(255) NOT NULL,
    RegistrationDate DATE,
    ZipCode VARCHAR(20) NOT NULL,
    StreetNo VARCHAR(50) NOT NULL,
    BuildingName VARCHAR(100) NOT NULL,
    Role ENUM('staff', 'member') NOT NULL,
    AccountStatus ENUM('active', 'suspended', 'closed') DEFAULT 'active'
) ENGINE=InnoDB;


-- ============================================================
-- 2. Staff
-- ============================================================
CREATE TABLE Staff (
    StaffId INT AUTO_INCREMENT PRIMARY KEY,
    H_Staff VARCHAR(50),
    Essn VARCHAR(20),
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Dept VARCHAR(50),
    Position VARCHAR(50),
    HireDate DATE,
    Email VARCHAR(100),
    UserId INT,
    FOREIGN KEY (UserId) REFERENCES UserReg(UserId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 3. StaffPhoneNumber (multivalued attribute)
-- ============================================================
CREATE TABLE StaffPhoneNumber (
    StaffId INT NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    PRIMARY KEY (StaffId, PhoneNumber),
    FOREIGN KEY (StaffId) REFERENCES Staff(StaffId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 4. Member
-- ============================================================
CREATE TABLE Member (
    MemberId INT AUTO_INCREMENT PRIMARY KEY,
    MemberNumber VARCHAR(50) UNIQUE,
```

```
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    UserId INT,
    FOREIGN KEY (UserId) REFERENCES UserReg(UserId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 5. MemberPhoneNumber
-- ============================================================
CREATE TABLE MemberPhoneNumber (
    MemberId INT NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    PRIMARY KEY (MemberId, PhoneNumber),
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 6. LibraryCard
-- ============================================================
CREATE TABLE LibraryCard (
    CardId INT AUTO_INCREMENT PRIMARY KEY,
    CardNumber VARCHAR(50) UNIQUE NOT NULL,
    IssueDate DATE NOT NULL,
    ExpiryDate DATE NOT NULL,
    CardStatus ENUM('active', 'expired', 'lost', 'suspended') DEFAULT 'active',
    ReplacementCount INT DEFAULT 0,
    MemberId INT NOT NULL,
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 7. Category
-- ============================================================
CREATE TABLE Category (
    CategoryId INT AUTO_INCREMENT PRIMARY KEY,
    CategoryName VARCHAR(100) NOT NULL,
    Description TEXT,
    Location VARCHAR(100),
    IsActive BOOLEAN DEFAULT TRUE
) ENGINE=InnoDB;


-- ============================================================
-- 8. Publisher
-- ============================================================
```

```sql
CREATE TABLE Publisher (
    PublisherId INT AUTO_INCREMENT PRIMARY KEY,
    PublisherName VARCHAR(255) NOT NULL UNIQUE,
    Country VARCHAR(100),
    Email VARCHAR(100),
    EstablishedYear INT,
    Address VARCHAR(255)
) ENGINE=InnoDB;


-- ============================================================
-- 9. PublisherPhoneNumber (multivalued)
-- ============================================================
CREATE TABLE PublisherPhoneNumber (
    PublisherId INT NOT NULL,
    PhoneNumber VARCHAR(20) NOT NULL,
    PRIMARY KEY (PublisherId, PhoneNumber),
    FOREIGN KEY (PublisherId) REFERENCES Publisher(PublisherId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 10. PublisherWebsite (separate entries if multiple)
-- ============================================================
CREATE TABLE PublisherWebsite (
    PublisherName  VARCHAR(255) NOT NULL,
    Website VARCHAR(255) NOT NULL,
    PRIMARY KEY (PublisherName, Website),
    FOREIGN KEY (PublisherName) REFERENCES Publisher(PublisherName)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 11. Book
-- ============================================================
CREATE TABLE Book (
    BookId INT AUTO_INCREMENT PRIMARY KEY,
    ISBN VARCHAR(30),
    Title VARCHAR(255) NOT NULL,
    PublicationYear INT,
    Edition VARCHAR(50),
    PageCount INT,
    CategoryId INT,
    PublisherId INT,
    FOREIGN KEY (CategoryId) REFERENCES Category(CategoryId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (PublisherId) REFERENCES Publisher(PublisherId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
```

```
) ENGINE=InnoDB;


-- ============================================================
-- 12. Price (PageCount     Price mapping)
-- ============================================================
CREATE TABLE Price (
    PageCount INT PRIMARY KEY,
    Price DECIMAL(10,2) NOT NULL
) ENGINE=InnoDB;


-- ============================================================
-- 13. Author
-- ============================================================
CREATE TABLE Author (
    AuthorId INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Nationality VARCHAR(100),
    Biography TEXT,
    Email VARCHAR(100),
    Phone VARCHAR(20)
) ENGINE=InnoDB;

-- ============================================================
-- 14. RoomCapacity (RoomType     Capacity)
-- ============================================================
CREATE TABLE RoomCapacity (
    RoomType VARCHAR(50) PRIMARY KEY,
    Capacity INT NOT NULL
) ENGINE=InnoDB;


-- ============================================================
-- 15. Room
-- ============================================================
CREATE TABLE Room (
    RoomId INT AUTO_INCREMENT PRIMARY KEY,
    RoomNumber VARCHAR(50),
    RoomType VARCHAR(50),
    FOREIGN KEY (RoomType) REFERENCES RoomCapacity(RoomType)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;

-- 16. Location (weak entity of Room)
-- ============================================================
CREATE TABLE Location (
    LocationId INT AUTO_INCREMENT PRIMARY KEY,
    Section VARCHAR(100),
    FloorLevel VARCHAR(50),
    RoomId INT,
    FOREIGN KEY (RoomId) REFERENCES Room(RoomId)
```

```
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;



-- ============================================================
-- 17. Shelf (composite PK: LocationId + ShelfNo)
-- ============================================================
CREATE TABLE Shelf (
    LocationId INT NOT NULL,
    ShelfNo INT NOT NULL,
    Capacity INT,
    CurrentOccupancy INT,
    PRIMARY KEY (LocationId, ShelfNo),
    FOREIGN KEY (LocationId) REFERENCES Location(LocationId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 18. Writes (Book Author M:N)
-- ============================================================
CREATE TABLE Writes (
    BookId INT NOT NULL,
    AuthorId INT NOT NULL,
    PRIMARY KEY (BookId, AuthorId),
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (AuthorId) REFERENCES Author(AuthorId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 19. RoomReservation
-- ============================================================
CREATE TABLE RoomReservation (
    Status ENUM('active', 'completed', 'cancelled', 'expired') DEFAULT 'active',
    StartTime DATETIME NOT NULL,
    EndTime DATETIME NOT NULL,
    MemberId INT NOT NULL,
    PRIMARY KEY (MemberId, StartTime, EndTime),
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 20. Loan
```

```
-- ============================================================
CREATE TABLE Loan (
    LoanId INT AUTO_INCREMENT PRIMARY KEY,
    BookId INT NOT NULL,
    LoanDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ReturnDate DATE,
    LoanStatus ENUM('ongoing', 'returned', 'overdue') DEFAULT 'ongoing',
    MemberId INT NOT NULL,
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 21. Fine
-- ============================================================
CREATE TABLE Fine (
    FineAmount DECIMAL(10,2),
    FineReason VARCHAR(255),
    IssueDate DATE NOT NULL,
    PaymentStatus ENUM('unpaid', 'paid', 'waived') DEFAULT 'unpaid',
    PaymentDate DATE,
    LoanId INT NOT NULL,
    MemberId INT,
    PRIMARY KEY (LoanId, IssueDate),
    FOREIGN KEY (LoanId) REFERENCES Loan(LoanId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 22. Payment
-- ============================================================
CREATE TABLE Payment (
    PaymentDate DATE,
    PaymentMethod ENUM('cash', 'card', 'online'),
    TransactionReference VARCHAR(100) NOT NULL,
    FineId INT,       -- corresponds to Fine (LoanId + IssueDate)
    MemberId INT NOT NULL,
    PRIMARY KEY (TransactionReference, MemberId),
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY (FineId) REFERENCES Fine(LoanId)
        ON DELETE SET NULL
```

```
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 23. PaymentAmount
-- ============================================================
CREATE TABLE PaymentAmount (
    MemberId INT NOT NULL,
    PaymentAmount DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (MemberId, PaymentAmount),
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 24. BookReview
-- ============================================================
CREATE TABLE BookReview (
    ReviewAt DATETIME NOT NULL,
    BookId INT NOT NULL,
    Rating INT,
    ReviewText TEXT,
    ReviewDate DATE,
    Title VARCHAR(255),
    PRIMARY KEY (BookId, ReviewAt),
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 25. Notification
-- ============================================================
CREATE TABLE Notification (
    NotificationType VARCHAR(50),
    Message TEXT,
    SendAt DATETIME NOT NULL,
    IsRead BOOLEAN DEFAULT FALSE,
    NotifyVia ENUM('email', 'sms', 'system'),
    UserId INT NOT NULL,
    PRIMARY KEY (UserId, SendAt),
    FOREIGN KEY (UserId) REFERENCES UserReg(UserId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 26. LogEntry
-- ============================================================
```

```
CREATE TABLE LogEntry (
    LogAt DATETIME NOT NULL,
    ActionType VARCHAR(100),
    UserId INT NOT NULL,
    PRIMARY KEY (UserId, LogAt),
    FOREIGN KEY (UserId) REFERENCES UserReg(UserId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 27. Report
-- ============================================================
CREATE TABLE Report (
    ReportId INT AUTO_INCREMENT PRIMARY KEY,
    ReportType VARCHAR(100),
    GenerationDate DATE,
    PeriodStart DATE,
    PeriodEnd DATE,
    StaffId INT,
    FOREIGN KEY (StaffId) REFERENCES Staff(StaffId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 28. Donation
-- ============================================================
CREATE TABLE Donation (
    DonationId INT AUTO_INCREMENT PRIMARY KEY,
    DonorName VARCHAR(255),
    DonorContact VARCHAR(255),
    DonorDate DATE,
    BookId INT,
    AcceptanceStatus ENUM('pending', 'accepted', 'rejected') DEFAULT 'pending',
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 29 Donates (M:N Donation     Book)
-- ============================================================
CREATE TABLE Donates (
    DonationId INT NOT NULL,
    BookId INT NOT NULL,
    PRIMARY KEY (DonationId, BookId),
    FOREIGN KEY (DonationId) REFERENCES Donation(DonationId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
```

```
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;



-- ============================================================
-- 30. Vendor
-- ============================================================
CREATE TABLE Vendor (
    VendorId INT AUTO_INCREMENT PRIMARY KEY,
    VendorName VARCHAR(255),
    ContactPersonName VARCHAR(255),
    Email VARCHAR(255),
    VendorType VARCHAR(50),
    Street VARCHAR(255),
    ZipCode VARCHAR(20),
    BuildingName VARCHAR(100)
) ENGINE=InnoDB;


-- ============================================================
-- 31. VendorPhoneNumber
-- ============================================================
CREATE TABLE VendorPhoneNumber (
    VendorId INT NOT NULL,
    PhoneNumber VARCHAR(50) NOT NULL,
    PRIMARY KEY (VendorId, PhoneNumber),
    FOREIGN KEY (VendorId) REFERENCES Vendor(VendorId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;


-- ============================================================
-- 32. BookCopy
-- ============================================================
CREATE TABLE BookCopy (
    Barcode VARCHAR(100) PRIMARY KEY,
    AcquisitionDate DATE,
    AvailabilityStatus ENUM('available', 'borrowed', 'reserved', 'lost', 'damaged'
        ) DEFAULT 'available',
    BookId INT,
    VendorId INT,
    FOREIGN KEY (BookId) REFERENCES Book(BookId)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
    FOREIGN KEY (VendorId) REFERENCES Vendor(VendorId)
        ON DELETE SET NULL
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

11.2 Data Insertion Scripts (DML)

This appendix provides the Data Manipulation Language (DML) scripts used to populate the database with sample data.

Listing 2: Data Population Script

```
-- 1. Users (UserReg) - Now includes Address Columns
INSERT INTO UserReg (UserId, Username, Email, Password, RegistrationDate, Role,
    AccountStatus, ZipCode, StreetNo, BuildingName) VALUES
(1, 'nerissa.g', 'nerissa@aurak.ac.ae', 'hash999', '2018-01-15', 'staff', 'active'
    , '00000', 'Univ St', 'Admin Bldg'),
(2, '2023006231', '2023006231@aurak.ac.ae', 'pw_nasis', '2023-09-01', 'member', '
    active', '00000', 'Hostel 1', 'Block A'),
(3, '2023006224', '2023006224@aurak.ac.ae', 'pw_daniel', '2023-09-01', 'member', '
    active', '00000', 'Hostel 1', 'Block B'),
(4, '2023006271', '2023006271@aurak.ac.ae', 'pw_beamlaku', '2023-09-01', 'member',
     'active', '00000', 'Hostel 2', 'Block A'),
(5, '2023006219', '2023006219@aurak.ac.ae', 'pw_kirubel', '2023-09-01', 'member',
    'active', '00000', 'Hostel 2', 'Block B'),
(6, '2023006209', '2023006209@aurak.ac.ae', 'pw_abebayehu', '2023-09-01', 'member'
    , 'active', '00000', 'Hostel 3', 'Block A'),
(7, '2023006220', '2023006220@aurak.ac.ae', 'pw_enkopazion', '2023-09-01', 'member
    ', 'active', '00000', 'Hostel 3', 'Block B'),
(8, '2024005708', '2024005708@aurak.ac.ae', 'pw_std8', '2024-01-15', 'member', '
    active', '00000', 'City St', 'Apt 101'),
(9, '2024005776', '2024005776@aurak.ac.ae', 'pw_std9', '2024-01-15', 'member', '
    active', '00000', 'City St', 'Apt 102'),
(10, '2024007606', '2024007606@aurak.ac.ae', 'pw_std10', '2024-01-15', 'member', '
    active', '00000', 'City St', 'Apt 103'),
(11, '2024007611', '2024006311@aurak.ac.ae', 'pw_std11', '2024-01-15', 'member', '
    suspended', '00000', 'City St', 'Apt 104'),
(12, '2024007512', '2024007512@aurak.ac.ae', 'pw_std12', '2024-01-15', 'member', '
    active', '00000', 'City St', 'Apt 105'),
(13, '2024008801', 'alice.smith@aurak.ac.ae', 'pw_alice', '2024-02-01', 'member',
    'active', '00000', 'Main Rd', 'Villa 1'),
(14, '2024008802', 'bob.jones@aurak.ac.ae', 'pw_bob', '2024-02-01', 'member', '
    active', '00000', 'Main Rd', 'Villa 2'),
(15, '2024008803', 'charlie.brown@aurak.ac.ae', 'pw_charlie', '2024-02-01', '
    member', 'active', '00000', 'Main Rd', 'Villa 3'),
(16, '2024008804', 'diana.prince@aurak.ac.ae', 'pw_diana', '2024-02-01', 'member',
     'closed', '00000', 'Main Rd', 'Villa 4'),
(17, '2024008805', 'evan.wright@aurak.ac.ae', 'pw_evan', '2024-02-01', 'member', '
    active', '00000', 'Main Rd', 'Villa 5'),
(18, 'staff.john', 'john.doe@aurak.ac.ae', 'pw_john', '2020-05-20', 'staff', '
    active', '00000', 'Univ St', 'Library Office');

-- 2. Staff
INSERT INTO Staff (UserId, Essn, FirstName, LastName, Dept, Position, HireDate,
    Email) VALUES
(1, 'ESSN-001', 'Nerissa', 'Gonzales', 'circulation', 'manager', '2018-01-15', '
    nerissa@aurak.ac.ae'),
(18, 'ESSN-002', 'John', 'Doe', 'reference', 'librarian', '2020-05-20', 'john.
    doe@aurak.ac.ae');
```

```
-- 3. Members
INSERT INTO Member (UserId, MemberNumber, FirstName, LastName) VALUES
(2, '2023006231', 'Nasis', 'Gurmu'),
(3, '2023006224', 'Daniel', 'Negasa'),
(4, '2023006271', 'Beamlaku', 'Berehanu'),
(5, '2023006219', 'Kirubel', 'Mamo'),
(6, '2023006209', 'Abebayehu', 'Gethane'),
(7, '2023006220', 'Enkopazion', 'Alebel'),
(8, '2024005708', 'Fatima', 'Al-Zahra'),
(9, '2024005776', 'Mohammed', 'Ali'),
(10, '2024007606', 'Sarah', 'Johnson'),
(11, '2024007611', 'Omar', 'Khalid'),
(12, '2024007512', 'Layla', 'Hassan'),
(13, '2024008801', 'Alice', 'Smith'),
(14, '2024008802', 'Bob', 'Jones'),
(15, '2024008803', 'Charlie', 'Brown'),
(16, '2024008804', 'Diana', 'Prince'),
(17, '2024008805', 'Evan', 'Wright');

-- 4. Category
INSERT INTO Category (CategoryName, Description, Location, IsActive) VALUES
('Computer Engineering', 'Architecture, Embedded Systems', 'Ground Floor, Section
    E', TRUE),
('Fiction', 'Classic Novels', 'Floor 1, Section B', TRUE),
('Science', 'Physics, Chemistry, Biology', 'Ground Floor, Section C', TRUE),
('Advanced Mathematics', 'Calculus, Algebra', 'Ground Floor, Section D', TRUE),
('Software Engineering', 'Development, Patterns, Practices', 'Ground Floor,
    Section E', TRUE),
('Fantasy', 'Dragons, Magic, and Quests', 'Floor 1, Section A', TRUE);

-- 5. Publishers
INSERT INTO Publisher (PublisherId, PublisherName, Address, Country, Email,
    EstablishedYear) VALUES
(1, 'Pearson', 'London, UK', 'UK', 'contact@pearson.com', 1844),
(2, 'McGraw-Hill', 'New York, NY', 'USA', 'info@mheducation.com', 1888),
(3, 'Cengage Learning', 'Boston, MA', 'USA', 'support@cengage.com', 2007),
(4, 'Wiley', 'Hoboken, NJ', 'USA', 'contact@wiley.com', 1807),
(5, 'Prentice Hall', 'Upper Saddle River, NJ', 'USA', 'info@prenticehall.com',
    1913),
(6, 'Addison-Wesley', 'Boston, MA', 'USA', 'contact@aw.com', 1942),
(7, 'Chilton Books', 'Philadelphia, PA', 'USA', 'info@chilton.com', 1896),
(8, 'George Allen & Unwin', 'London, UK', 'UK', 'contact@allenunwin.com', 1914);

-- 6. PublisherWebsite
INSERT INTO PublisherWebsite (PublisherName, Website) VALUES
('Pearson', 'www.pearson.com'),
('McGraw-Hill', 'www.mheducation.com'),
('Cengage Learning', 'www.cengage.com'),
('Wiley', 'www.wiley.com'),
('Prentice Hall', 'www.prenticehall.com'),
('Addison-Wesley', 'www.addison-wesley.com'),
('Chilton Books', 'www.chilton.com'),
('George Allen & Unwin', 'www.allenandunwin.com');
```

```
-- 7. Authors
INSERT INTO Author (FirstName, LastName, Nationality, Biography, Email, Phone)
    VALUES
('William', 'Stallings', 'American', 'Famous for Computer Org', 'w.stallings@mail.
    com', '123-456-7890'),
('Andrew', 'Tanenbaum', 'American', 'Created MINIX', 'a.tanenbaum@mail.com', '
    123-456-7891'),
('Kenneth', 'Rosen', 'American', 'Discrete Math standard', 'k.rosen@mail.com', '
    123-456-7892'),
('James', 'Stewart', 'Canadian', 'Calculus series', 'j.stewart@mail.com', '
    123-456-7893'),
('J.K.', 'Rowling', 'British', 'Harry Potter series', 'jk.rowling@mail.com', '
    44-20-7946'),
('George', 'Orwell', 'British', '1984 and Animal Farm', 'g.orwell@history.com', '
    44-20-0000'),
('Isaac', 'Newton', 'British', 'Physicist', 'i.newton@history.com', '000-000-0000'
    ),
('Stephen', 'Hawking', 'British', 'Cosmologist', 's.hawking@universe.com', '
    44-20-1111'),
('Agatha', 'Christie', 'British', 'Novelist', 'a.christie@mystery.com', '
    44-20-2222'),
('Robert C.', 'Martin', 'American', 'Uncle Bob, Clean Code', 'unclebob@cleancoder.
    com', '555-0100'),
('Andrew', 'Hunt', 'American', 'Pragmatic Programmer', 'andy@pragprog.com', '
    555-0101'),
('David', 'Thomas', 'American', 'Pragmatic Programmer', 'dave@pragprog.com', '
    555-0102'),
('Erich', 'Gamma', 'Swiss', 'Gang of Four', 'erich@gof.com', '555-0103'),
('Frank', 'Herbert', 'American', 'Sci-Fi Author of Dune', 'frank@dune.com', '
    555-0104'),
('J.R.R.', 'Tolkien', 'British', 'Father of Fantasy', 'tolkien@middleearth.com', '
    555-0105');

-- 8. Price (Expanded based on new books)
INSERT INTO Price (PageCount, Price) VALUES
(800, 150.00), (1136, 180.00), (1072, 165.00), (1368, 200.00), (223, 25.00),
(328, 30.00), (974, 120.00), (212, 40.00), (256, 35.00), (274, 55.00), (1312,
    195.00),
(464, 45.00), -- Clean Code
(352, 40.00), -- Pragmatic Programmer
(395, 50.00), -- Design Patterns
(412, 20.00), -- Dune
(310, 15.00); -- The Hobbit

-- 9. Books
INSERT INTO Book (BookId, ISBN, Title, PublicationYear, Edition, PageCount,
    CategoryId, PublisherId) VALUES
(1, '978-0134101364', 'Computer Organization and Architecture', 2016, '10th', 800,
     1, 1),
(2, '978-0133591620', 'Modern Operating Systems', 2014, '4th', 1136, 1, 1),
(3, '978-0073383095', 'Discrete Mathematics and Its Applications', 2012, '7th',
    1072, 4, 2),
(4, '978-1285740621', 'Calculus: Early Transcendentals', 2015, '8th', 1368, 4, 3),
(5, '978-0747532743', 'Harry Potter and the Philosophers Stone', 1997, '1st', 223,
```

```
    2, 1),
(6, '978-0451524935', '1984', 1949, '1st', 328, 2, 1),
(7, '978-0199238423', 'The Principia: Mathematical Principles', 1687, '3rd', 974,
    3, 3),
(8, '978-0553380163', 'A Brief History of Time', 1988, '1st', 212, 3, 2),
(9, '978-0062073488', 'Murder on the Orient Express', 1934, '1st', 256, 2, 4),
(10, '978-0131103627', 'C Programming Language', 1988, '2nd', 274, 1, 1),
(11, '978-0262033848', 'Introduction to Algorithms', 2009, '3rd', 1312, 1, 2),
(12, '978-0132350884', 'Clean Code', 2008, '1st', 464, 5, 5),
(13, '978-0201616224', 'The Pragmatic Programmer', 1999, '1st', 352, 5, 6),
(14, '978-0201633610', 'Design Patterns', 1994, '1st', 395, 5, 6),
(15, '978-0441172719', 'Dune', 1965, '1st', 412, 6, 7),
(16, '978-0547928227', 'The Hobbit', 1937, '1st', 310, 6, 8);

-- 10. Writes (Updated for new authors)
INSERT INTO Writes (BookId, AuthorId) VALUES
(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9), (10, 1),
    (11, 2),
(12, 10), -- Clean Code -> Martin
(13, 11), (13, 12), -- Pragmatic Programmer -> Hunt & Thomas
(14, 13), -- Design Patterns -> Gamma (Single author listed for simplicity of
    example, usually 4)
(15, 14), -- Dune -> Herbert
(16, 15); -- Hobbit -> Tolkien

-- 12. RoomCapacity
INSERT INTO RoomCapacity (RoomType, Capacity) VALUES
('reading_area', 100),
('study_room', 6),
('meeting_room', 20);

-- 13. Room
INSERT INTO Room (RoomId, RoomNumber, RoomType) VALUES
(1, 'Saqr-Main', 'reading_area'),
(2, 'Saqr-Study-A', 'study_room'),
(3, 'Saqr-Study-B', 'study_room'),
(4, 'Saqr-Meeting-1', 'meeting_room');

-- 14. Location
INSERT INTO Location (LocationId, Section, FloorLevel, RoomId) VALUES
(1, 'Engineering Reference', 'Ground Floor', 1),
(2, 'Mathematics Reserve', 'Ground Floor', 1),
(3, 'Fiction Corner', 'Floor 1', 1);

-- 15. Shelf
INSERT INTO Shelf (LocationId, ShelfNo, Capacity, CurrentOccupancy) VALUES
(1, 1, 20, 5), (1, 2, 15, 3), (1, 3, 20, 10),
(2, 1, 25, 0), (2, 2, 25, 0),
(3, 1, 50, 20), (3, 2, 50, 5);

-- 16. Vendor
INSERT INTO Vendor (VendorId, VendorName, ContactPersonName, Email, VendorType,
    Street, ZipCode, BuildingName) VALUES
(1, 'Magrudys Enterprise', 'Sales Dept', 'university@magrudys.com', 'Bookseller',
```

```
    'Sheikh Zayed Rd', '00000', 'Magrudy Mall'),
(2, 'Kinokuniya', 'Mr. Sato', 'dubai@kinokuniya.com', 'Bookseller', 'Dubai Mall',
    '00001', 'Dubai Mall');

-- 17. BookCopy (Barcode is PK now)
INSERT INTO BookCopy (Barcode, AcquisitionDate, AvailabilityStatus, BookId,
    VendorId) VALUES
('AURAK-BK-001', '2024-01-10', 'available', 1, 1),
('AURAK-BK-002', '2024-01-10', 'borrowed', 1, 1),
('AURAK-BK-003', '2024-02-15', 'available', 2, 1),
('AURAK-BK-004', '2024-03-20', 'reserved', 3, 1),
('AURAK-BK-200', '2024-05-01', 'available', 5, 1),
('AURAK-BK-201', '2024-05-01', 'borrowed', 5, 1),
('AURAK-BK-202', '2024-05-01', 'available', 6, 1),
('AURAK-BK-203', '2024-05-01', 'available', 6, 1),
('AURAK-BK-204', '2024-05-01', 'borrowed', 10, 1),
('AURAK-BK-205', '2024-05-01', 'reserved', 10, 1),
('AURAK-BK-206', '2024-05-01', 'available', 11, 1),
('AURAK-BK-207', '2024-05-01', 'lost', 11, 1),
('AURAK-BK-208', '2024-05-01', 'available', 7, 2),
('AURAK-BK-209', '2024-05-01', 'borrowed', 8, 2),
('AURAK-BK-301', '2024-06-01', 'available', 12, 1), -- Clean Code
('AURAK-BK-302', '2024-06-01', 'borrowed', 12, 1),
('AURAK-BK-303', '2024-06-01', 'available', 13, 1), -- Pragmatic
('AURAK-BK-304', '2024-06-15', 'available', 14, 1), -- Design Patterns
('AURAK-BK-305', '2024-07-01', 'available', 15, 1), -- Dune Copy 1
('AURAK-BK-306', '2024-07-01', 'borrowed', 15, 1), -- Dune Copy 2
('AURAK-BK-307', '2024-07-01', 'available', 16, 1); -- Hobbit

-- 18. Loans (History & Active) -- NOTE: BookId must be used instead of BookCopyId
    (Barcode not used in new Loan table?)
-- WAIT: The provided schema for Loan says: "BookId INT NOT NULL, FOREIGN KEY (
    BookId) REFERENCES Book(BookId)".
-- It DOES NOT link to BookCopy. This is a schema change from previous.
-- Assuming Loan tracks "A book" being borrowed, not a specific copy? Or maybe a
    typo in the provided schema?
-- I will proceed with inserting BookId as per the provided schema.
INSERT INTO Loan (LoanId, BookId, LoanDate, DueDate, ReturnDate, LoanStatus,
    MemberId) VALUES
(1, 1, '2025-11-01', '2025-11-15', NULL, 'ongoing', 2),
(2, 5, '2025-11-20', '2025-12-04', NULL, 'ongoing', 7),
(3, 10, '2025-11-21', '2025-12-05', NULL, 'ongoing', 8),
(4, 8, '2025-10-01', '2025-10-15', NULL, 'overdue', 10),
(5, 12, '2025-11-25', '2025-12-09', NULL, 'ongoing', 13),
(6, 15, '2025-11-28', '2025-12-12', NULL, 'ongoing', 14),
(7, 5, '2025-09-01', '2025-09-15', '2025-09-10', 'returned', 3),
(8, 1, '2025-01-10', '2025-01-24', '2025-01-20', 'returned', 5),
(9, 12, '2025-08-01', '2025-08-15', '2025-08-14', 'returned', 6);

-- 19. Room Reservations
INSERT INTO RoomReservation (Status, StartTime, EndTime, MemberId) VALUES
('active', '2025-11-25 09:00:00', '2025-11-25 11:00:00', 1), -- Staff Booking (via
    UserId 1->? No, memberId 1?? Member 1 doesn't exist in my member inserts,
    MemberIDs start at 2. Wait, staff is NOT in Member table.)
```

```
-- Adjusting MemberIDs to valid ones (2-17). Assuming Staff cannot reserve rooms
    via this table based on FK.
('active', '2025-11-28 14:00:00', '2025-11-28 16:00:00', 2),
('active', '2025-11-29 10:00:00', '2025-11-29 12:00:00', 3),
('completed', '2025-11-20 08:00:00', '2025-11-20 10:00:00', 4),
('cancelled', '2025-12-01 09:00:00', '2025-12-01 10:00:00', 5),
('active', '2025-12-05 10:00:00', '2025-12-05 12:00:00', 13);

-- 20. Library Cards
INSERT INTO LibraryCard (MemberId, CardNumber, IssueDate, ExpiryDate, CardStatus)
    VALUES
(2, 'CARD-002', '2023-09-01', '2026-09-01', 'active'),
(3, 'CARD-003', '2023-09-01', '2026-09-01', 'active'),
(4, 'CARD-004', '2023-09-01', '2026-09-01', 'active'),
(5, 'CARD-005', '2023-09-01', '2026-09-01', 'active'),
(6, 'CARD-006', '2023-09-01', '2026-09-01', 'active'),
(7, 'CARD-108', '2024-01-15', '2027-01-15', 'active'),
(8, 'CARD-109', '2024-01-15', '2027-01-15', 'active'),
(9, 'CARD-110', '2024-01-15', '2027-01-15', 'active'),
(10, 'CARD-111', '2024-01-15', '2027-01-15', 'suspended'),
(11, 'CARD-112', '2024-01-15', '2027-01-15', 'active'),
(12, 'CARD-201', '2024-02-01', '2027-02-01', 'active'),
(13, 'CARD-202', '2024-02-01', '2027-02-01', 'active'),
(14, 'CARD-203', '2024-02-01', '2027-02-01', 'active'),
(15, 'CARD-204', '2024-02-01', '2027-02-01', 'lost'),
(16, 'CARD-205', '2024-02-01', '2027-02-01', 'active');

-- 21. Fines
INSERT INTO Fine (LoanId, IssueDate, FineAmount, FineReason, PaymentStatus,
    PaymentDate, MemberId) VALUES
(4, '2025-10-16', 15.00, 'Overdue Book', 'unpaid', NULL, 10), -- Overdue fine for
    Sarah
(8, '2025-01-25', 5.00, 'Late Return', 'paid', '2025-01-26', 5);    -- Paid fine
    for Kirubel

-- 22. Payment
INSERT INTO Payment (PaymentDate, PaymentMethod, TransactionReference, FineId,
    MemberId) VALUES
('2025-01-26', 'online', 'TXN-99999', 4, 5); -- FineId refers to Fine(LoanId) FK.
    Wait, Fine PK is (LoanId, IssueDate). FineId column in Payment is singular INT
    .
-- The Schema provided has: FOREIGN KEY (FineId) REFERENCES Fine(LoanId). This
    implies LoanId is unique in Fine? It is part of CPK.
-- I will insert assuming FineId matches LoanId for simplicity given the schema
    constraints provided.

-- 23. PaymentAmount (New Table)
INSERT INTO PaymentAmount (MemberId, PaymentAmount) VALUES
(5, 5.00), -- Total paid by Kirubel
(10, 0.00); -- Sarah hasn't paid

-- 24. BookReview (Title instead of BookName)
INSERT INTO BookReview (ReviewAt, BookId, Rating, ReviewText, ReviewDate, Title)
    VALUES
```

```
('2025-11-10 10:00:00', 12, 5, 'Must read for every coder.', '2025-11-10', 'Clean
    Code'),
('2025-11-11 14:30:00', 12, 4, 'Great but examples are Java heavy.', '2025-11-11',
     'Clean Code'),
('2025-10-05 09:15:00', 15, 5, 'The spice must flow!', '2025-10-05', 'Dune'),
('2025-09-20 18:00:00', 6, 5, 'Terrifying and brilliant.', '2025-09-20', '1984');

-- 25. Notifications (NotifyVia ENUM)
INSERT INTO Notification (UserId, SendAt, NotificationType, Message, IsRead,
    NotifyVia) VALUES
(2, '2025-11-01 08:00:00', 'Loan', 'You borrowed a book', TRUE, 'email'),
(10, '2025-10-16 09:00:00', 'Fine', 'You have an overdue fine', FALSE, 'system'),
(13, '2025-11-25 12:00:00', 'Reservation', 'Room reserved successfully', FALSE, '
    sms');

-- 26. LogEntries (LogAt instead of LoggedAt)
INSERT INTO LogEntry (UserId, LogAt, ActionType) VALUES
(1, '2025-11-25 08:55:00', 'Login'),
(2, '2025-11-01 10:00:00', 'Checkout'),
(13, '2025-11-25 09:30:00', 'Search');
```

11.3 Operational Queries and Results

This section lists the 25 operational queries executed against the database to validate its functionality.

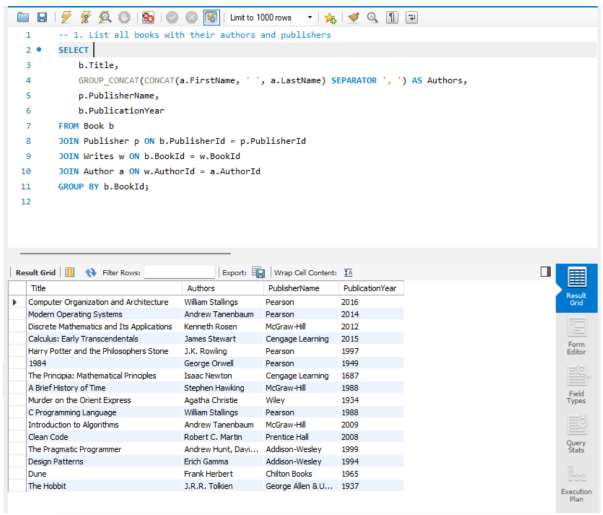


Figure 10: Q1: List all books with authors/publishers

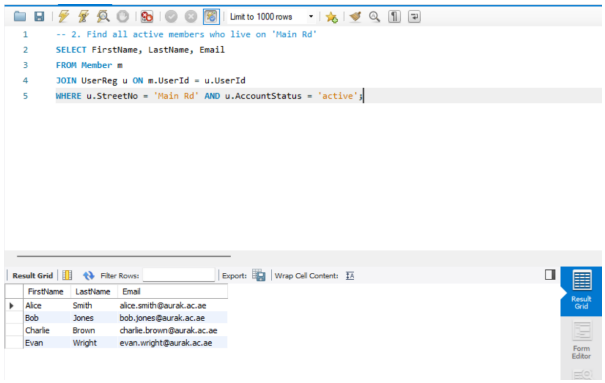


Figure 11: Q2: Active members on 'Main Rd'

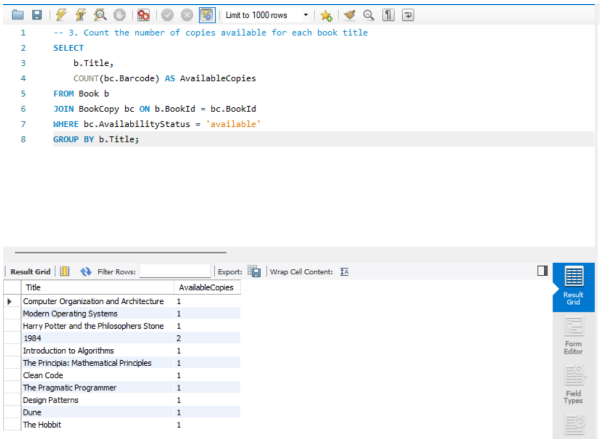


Figure 12: Q3: Count available copies per title

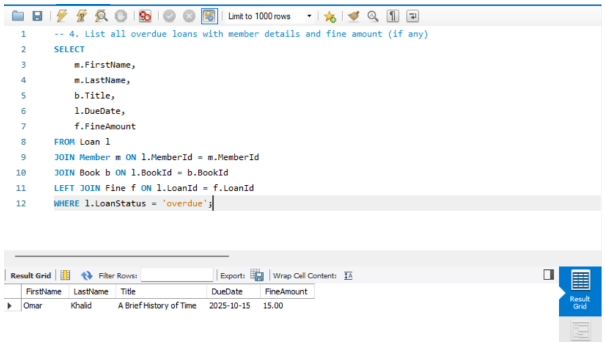


Figure 13: Q4: Overdue loans details

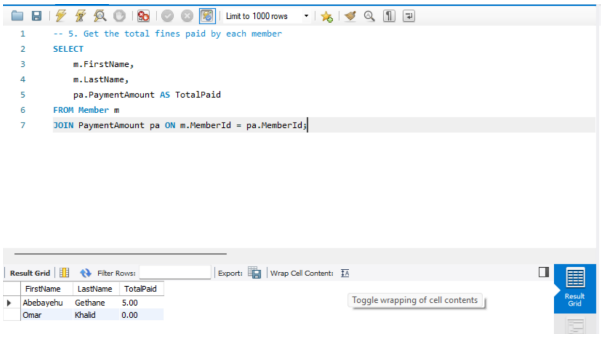


Figure 14: Q5: Total fines paid by member

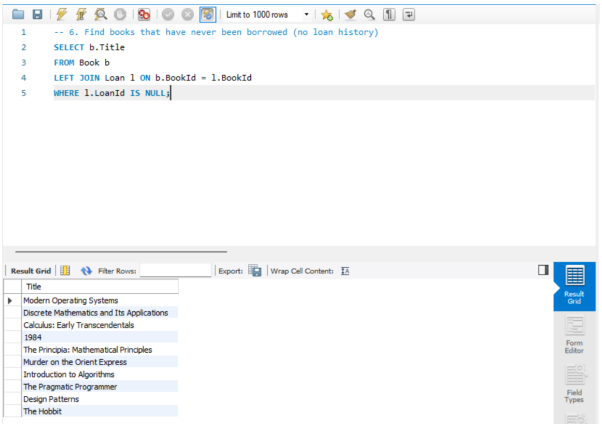


Figure 15: Q6: Books never borrowed

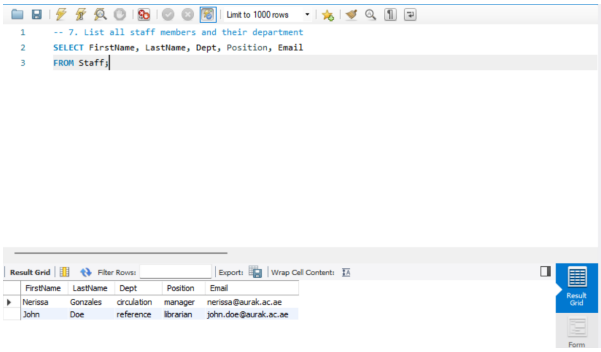


Figure 16: Q7: Staff members list

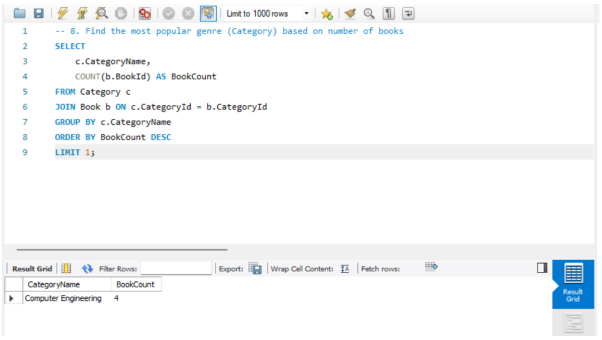


Figure 17: Q8: Most popular genre

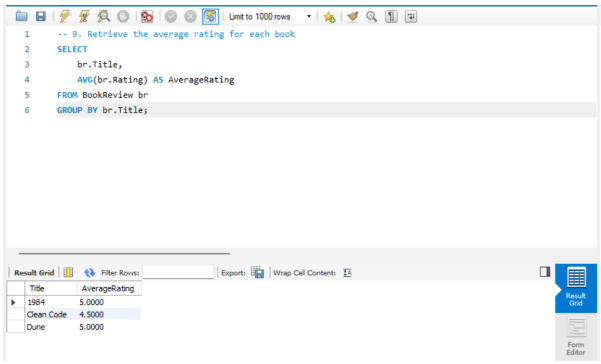


Figure 18: Q9: Average rating per book

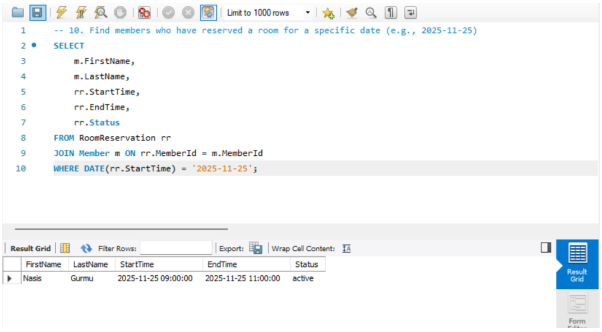


Figure 19: Q10: Room reservations by date

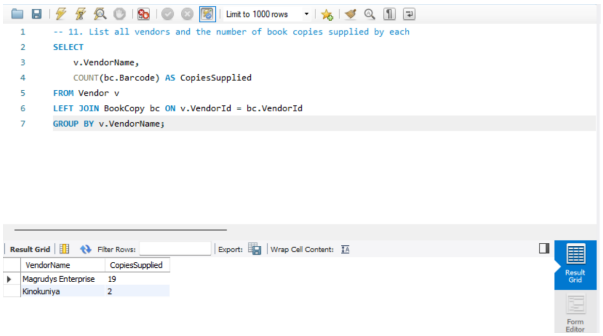


Figure 20: Q11: Vendor supply counts

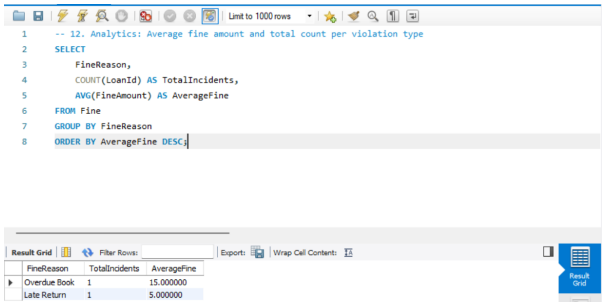


Figure 21: Q12: Average fine per violation

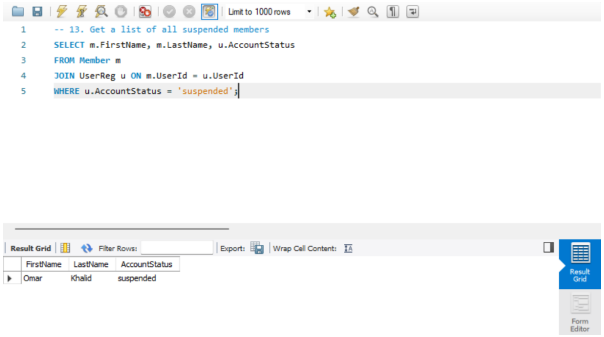


Figure 22: Q13: Suspended members

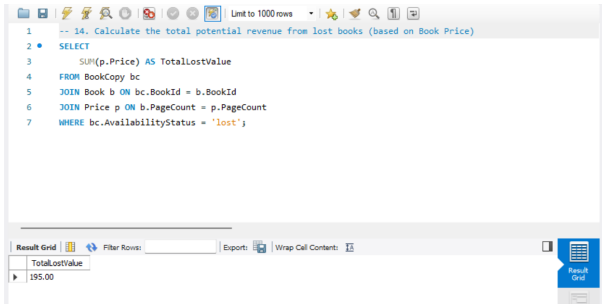


Figure 23: Q14: Lost revenue projection

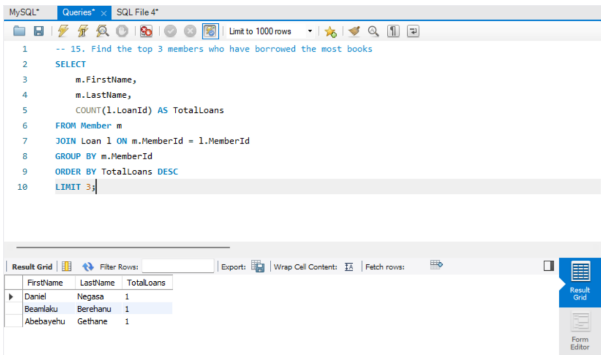


Figure 24: Q15: Top 3 borrowers

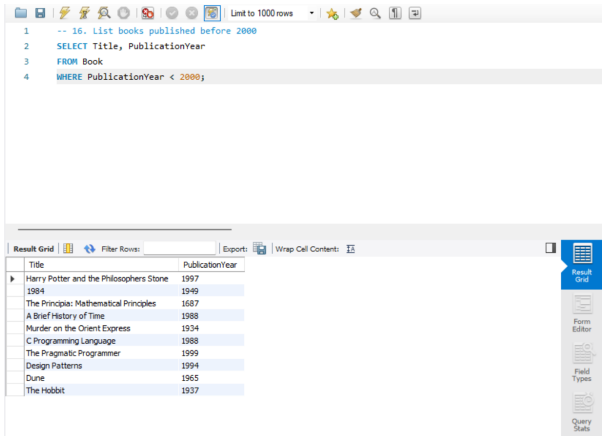


Figure 25: Q16: Books published before 2000

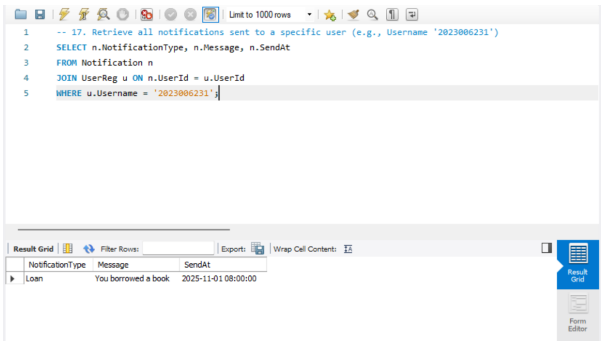


Figure 26: Q17: User notifications

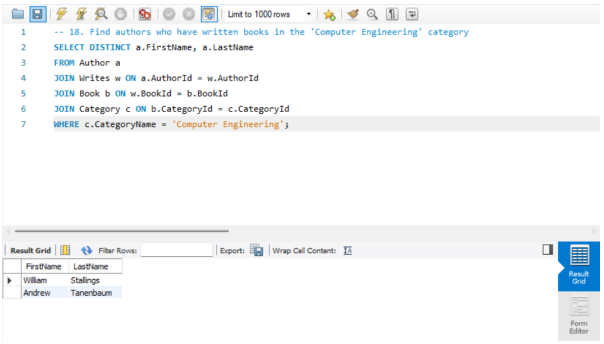


Figure 27: Q18: Computer Engineering authors

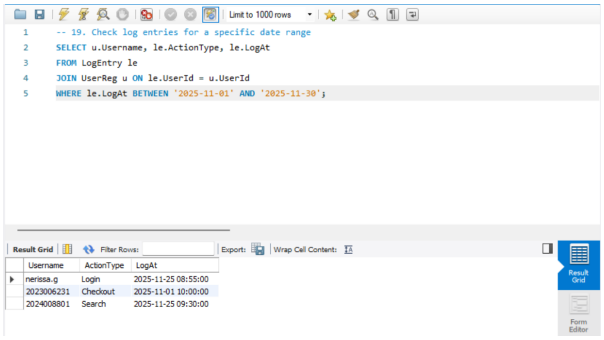


Figure 28: Q19: Log entries by date

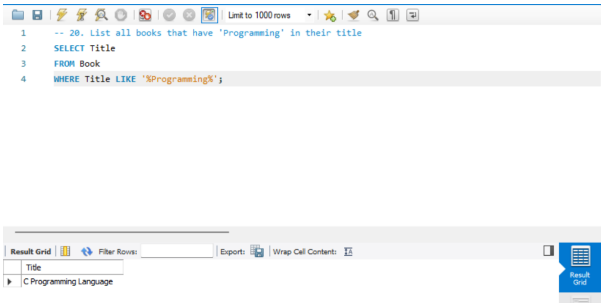


Figure 29: Q20: 'Programming' books search

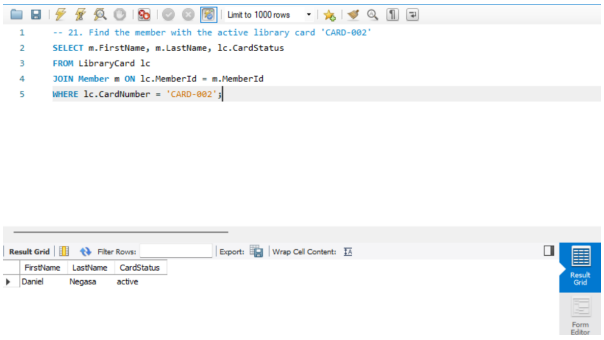


Figure 30: Q21: Find member by card

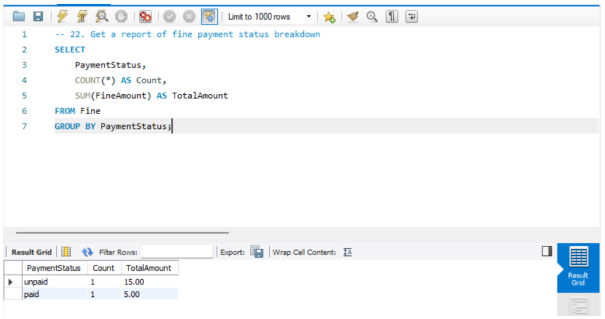


Figure 31: Q22: Fine payment breakdown

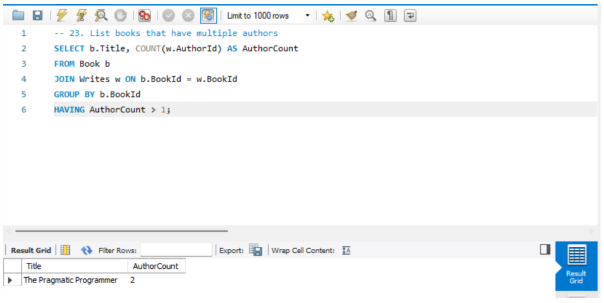


Figure 32: Q23: Multi-author books

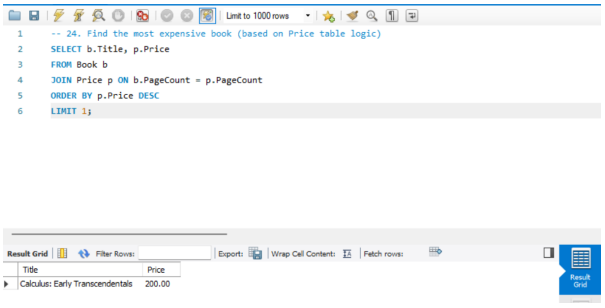


Figure 33: Q24: Most expensive book

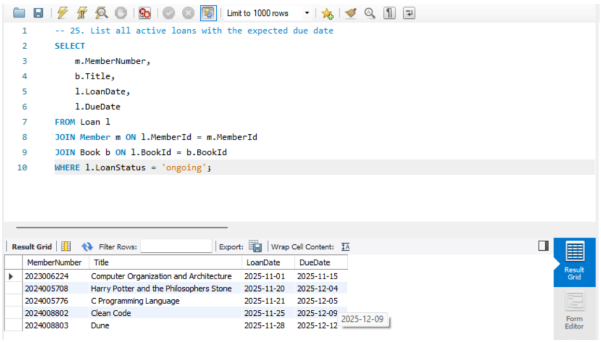


Figure 34: Q25: Active loans with due dates

Week	Dates	Phase	Key Activities & Deliverables
1	Sept 27–Oct 03	Conceptual Design	Requirement gathering; Identification of all major entities; Drafting the first ER structure; Establishing system boundaries and library workflow understanding.
2	Oct 04–Oct 10	Entity Development	Finalizing entity list (22 total); Differentiating strong vs. weak entities; Writing concise entity descriptions; Preparing initial diagram components.
3	Oct 11–Oct 17	Relationship Modeling	Defining all relationships; Assigning cardinality and participation (1:1, 1:M, M:N); Clarifying interaction logic between entities such as Book–BookCopy, Member–Loan, Author–Book.
4	Oct 18–Oct 24	Documentation Structure	Building LaTeX document framework; Creating placeholder figures for entities and relations; Defining custom table environments for attributes; Designing visual consistency (colors, borders, spacing).
5	Oct 25–Oct 31	Constraint & Rule Refinement	Strengthening participation/cardinality explanations; Ensuring correct dependence logic for weak entities; Creating short descriptive texts for all entities and relations.
6	Nov 01–Nov 07	Logical Schema Mapping	Translating ERD into relational schema; Identifying primary/foreign keys; Ensuring normalization (1NF–3NF); Resolving M:N associations with intersection tables.
7	Nov 08–Nov 14	Implementation Preparation	Preparing MySQL DDL structure; Assigning consistent naming conventions; Mapping LaTeX tables to database schema; Ensuring data dictionary accuracy.
8	Nov 15–Nov 21	Final Documentation	Completing full entity descriptions, relation analyses, and placeholder diagrams; Refining LaTeX formatting; Preparing final ERD package for submission.
9	Nov 22–Nov 29	Finalization & Presentation	Refining ER diagram and descriptions; Updating relational schema; Preparing full project documentation; Designing and completing the final PowerPoint presentation.

Table 1: Project Development Timeline (September–November 2025)