Library Booking System: Database

Week 5 Final Report

Database CS A

Made by:

Klara Ahinta Daniswara	**24/532751/PA/22531**
Daffa Aryza Pasya	**24/532884/PA/22549**
Shadiq Arif Aryasatya	**24/536702/PA/22763**

DEPARTEMEN ILMU KOMPUTER dan ELEKTRONIKA

FAKULTAS MATEMATIKA dan ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

YOGYAKARTA

2025

Tables of Content

1. Introduction

1.1 Background

This database is designed to support the core operational processes of a library management system, including book cataloging, borrowing activities, member management, staff operations, and fine handling. The overall design follows relational database principles by modeling real-world objects and activities, enforcing data integrity through constraints, and reducing redundancy through normalization. The design process begins with a conceptual entity–relationship diagram and is later translated into a logical relational schema and physical SQL implementation.

1.2 Problem Statement

The core issue addressed by this project was the lack of an organized, reliable, and scalable method for managing library operations. Manual book recording leads to misplaced records, repeated entries, difficulty tracking active borrowings, and a high chance of human error. As the number of books and members grows, searching for titles or identifying overdue items becomes slow. A structured database system was needed to store information consistently, maintain borrowing history, and ensure that library data can be retrieved and updated accurately.

1.3 Objectives

The main objective of the project was to transform a conceptual ERD into a fully working system. The aim was to design a clean database schema, implement it in SQL, then build a backend capable of CRUD operations for books, members, staff, and borrowing activities. Another objective was to create a frontend interface so that users could interact with the system visually, rather than directly accessing the database. The final intention was to produce a basic but functional digital library system which demonstrates how structured data flows from design, to code, to application level.

1.4 Scope and Limitations

The system currently includes core features such as book catalog management, physical book copy tracking, member registration, borrowing and returning functions, and fine recording structures. The implementation is still early-stage: the user interface is simple, fines are structurally supported but not fully automated, and some usability improvements were postponed

due to time constraints. The project focuses on learning database implementation rather than achieving a production-ready application.

2. Literature Review

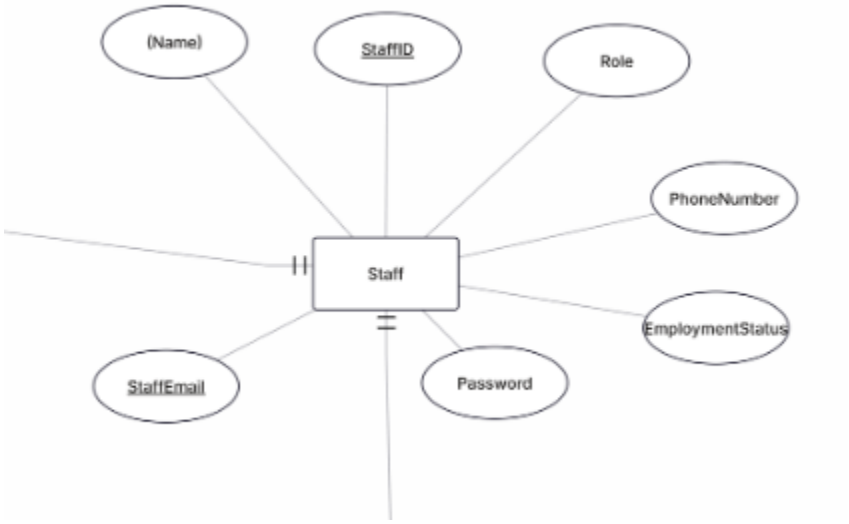
The entity–relationship diagram represents the conceptual view of the system and focuses on identifying the main entities, their attributes, and the relationships between them without emphasizing implementation details. In the ERD, the Book entity represents books available in the library and includes attributes such as ISBN, title, genre, publication year, publisher, author, availability, and book identifier. At the conceptual level, these attributes are grouped together to simplify understanding of the domain, even though some of them describe bibliographic information while others relate to physical copies. ISBN serves as the identifying attribute for books in the ERD because it uniquely distinguishes one book edition from another in the real world.

2.1 Background Theory

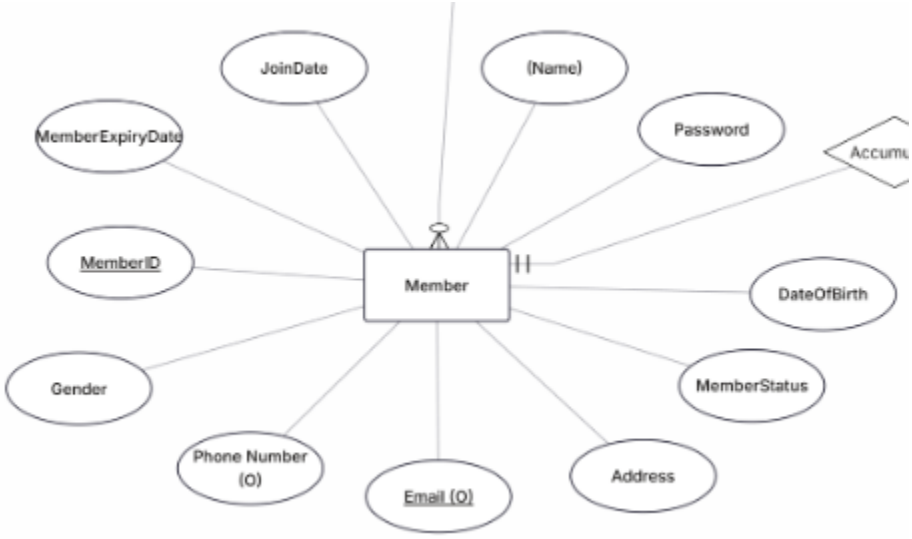
A relational database organizes data into tables linked by keys. Its main purpose is to reduce redundancy while maintaining integrity between related data. The ERD models entities and relationships conceptually before any table is created. Once the ERD is validated, it is converted into a relational schema where each table contains only attributes relevant to its function. Normalization ensures that updates do not cause anomalies. Third Normal Form naturally appeared during design because attributes were grouped by dependency on primary keys, and fields unrelated to the entity were moved into new tables.

The ERD also models the Staff entity, which represents library employees responsible for managing books, assisting members, and handling fines. Staff attributes include staff identifier,

name, email, password, role, phone number, and employment status. Staff participates in relationships with books through management activities and with fines through administrative handling, reflecting their operational responsibilities in the library.

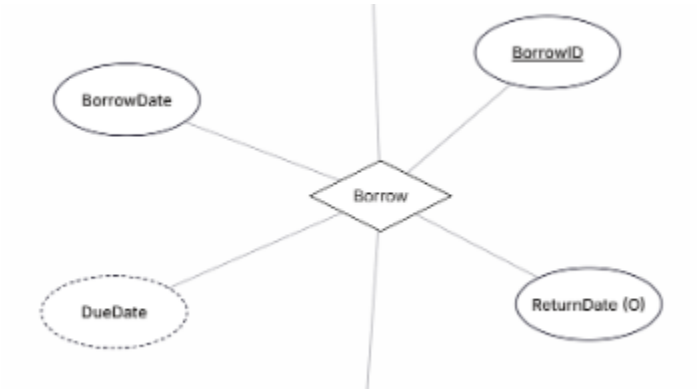


The Member entity represents registered library users who are allowed to borrow books. Member attributes include member identifier, name, password, contact information, address, gender, membership status, join date, expiry date, and date of birth. This entity is independent of borrowing activities because a member's personal and membership data must persist regardless of how often or whether they borrow books.



Borrowing activity is represented in the ERD as a Borrow relationship between Member and Book. This relationship includes attributes such as borrow date, due date, return date, and borrow identifier. Conceptually, this models the fact that members borrow books over time and that each borrowing event has its own properties. The ERD captures borrowing as an interaction rather

than a standalone object, which is appropriate at the conceptual level.



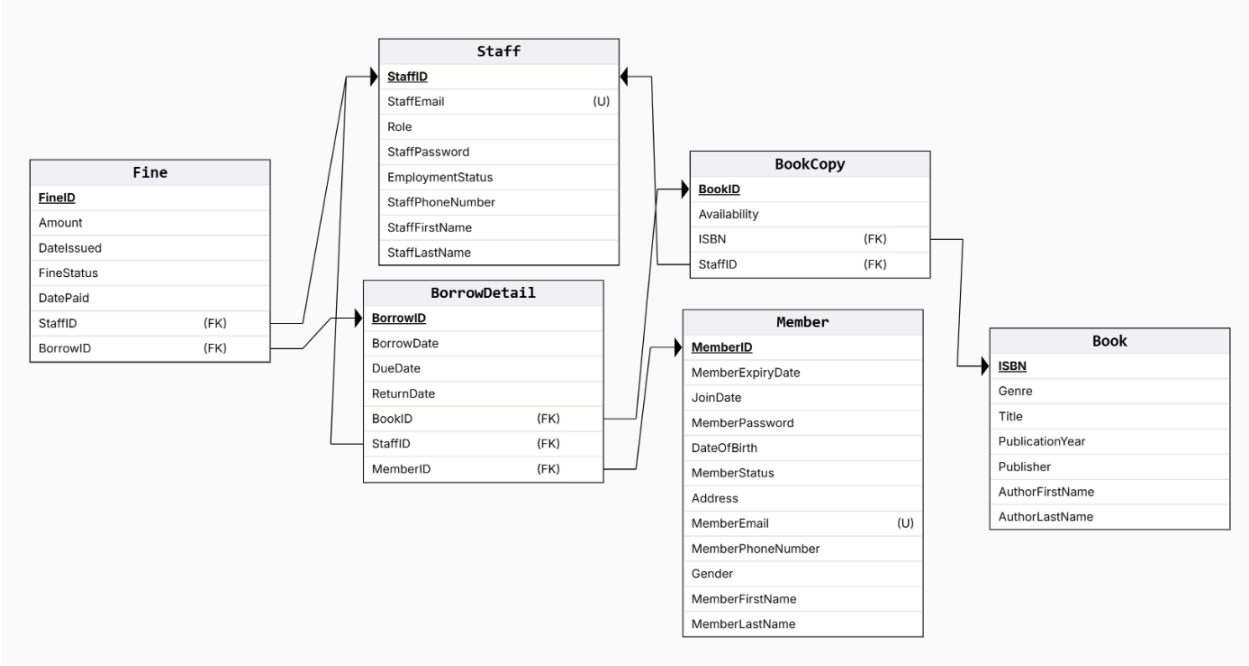
The Fine entity represents penalties imposed on members due to borrowing violations such as late returns. It includes attributes such as fine identifier, amount, date issued, fine status, and optional payment date. The ERD shows that fines are associated with members and handled by staff, reflecting real-world accountability and administrative processes.

2.2 Comparison to Existing Systems

Traditional library workflows rely on manual logging or spreadsheet storage. Such systems are prone to duplicate entries and inconsistent records. Digital library systems automate these tasks, allowing quick searching, faster borrowing registration, and reliable returns management. Our project follows this modern approach using SQL as a structured storage foundation and an API layer to interact with data programmatically.

3. System Analysis & Design

When translating the ERD into a relational schema, the design is refined to comply with relational database constraints and normalization rules. At this stage, implementation-oriented decisions are introduced. One of the most important refinements is the separation of physical book copies into their own table. While the ERD conceptually groups availability and book identifiers within Book, the relational schema introduces a BookCopy table to properly model multiple physical copies of the same book. This separation avoids redundancy and allows each copy to be uniquely tracked.



In the relational schema, the Book table stores only bibliographic information such as ISBN, title, genre, publication year, publisher, and author names. ISBN is chosen as the primary key because it is a natural, globally unique identifier for book editions. This ensures that book information is stored only once, regardless of how many copies exist.

The BookCopy table represents individual physical copies of books. Each copy is identified by a unique BookID, which serves as the primary key. This is necessary because multiple copies of the same ISBN may exist and must be distinguished individually. BookCopy includes an availability attribute to track whether a copy is available, borrowed, or unavailable, and it references the Book table through a foreign key on ISBN. BookCopy also references Staff to record which staff member registered or manages the copy.

The Member table in the relational schema closely follows the ERD but is implemented with a system-generated primary key, MemberID, to ensure stable and efficient references. Email is

constrained to be unique to prevent duplicate member accounts. Optional attributes such as email and phone number are nullable to reflect real-world conditions where not all members provide complete contact information.

The Staff table is implemented similarly, with StaffID as the primary key and StaffEmail as a unique attribute to support authentication. Attributes related to role, employment status, password, and contact information are included to support administrative and operational requirements.

The borrowing process is implemented as the BorrowDetail table, which corresponds to the Borrow relationship in the ERD. This table exists because borrowing represents a many-to-many relationship between members and book copies when viewed over time. A member can borrow many book copies, and a book copy can be borrowed multiple times throughout its lifetime. Relational databases cannot represent many-to-many relationships directly, so BorrowDetail acts as an associative table. BorrowID is used as the primary key to uniquely identify each borrowing transaction, while foreign keys reference the associated book copy, member, and staff member involved in the transaction. Borrow date and due date are mandatory, while return date is optional to represent ongoing borrowings.

The Fine table implements the Fine entity from the ERD. It exists as a separate table because not all borrow transactions result in fines, and fines have their own lifecycle independent of borrowing. FineID serves as the primary key, ensuring each fine record is uniquely identifiable. Foreign keys reference BorrowDetail and Staff, ensuring that every fine is associated with a valid borrowing transaction and handled by an authorized staff member. Attributes such as amount, date issued, fine status, and optional payment date fully capture the state of a fine.

The relational schema enforces referential integrity through foreign key constraints, preventing invalid or orphaned records. For example, a borrow record cannot exist without a valid member, book copy, and staff member, and a fine cannot exist without a corresponding borrow transaction. The schema satisfies Third Normal Form by ensuring that each table represents a single concept, that all non-key attributes depend on the whole primary key, and that there are no transitive dependencies.

The SQL implementation directly reflects this relational schema. Tables are created in an order that respects foreign key dependencies, beginning with independent entities and followed by dependent tables. Primary key and unique constraints ensure data uniqueness, while enumerated types restrict attribute values to predefined sets, enforcing business rules and improving consistency. Nullable fields are used intentionally to represent optional information and incomplete processes, such as unpaid fines or unreturned books.

3.1 Requirement Analysis

The system required the ability to store and edit book information, track multiple copies of the same ISBN, register new members, record borrow dates and return dates, and differentiate active and inactive accounts. It also needed authentication fields for staff, although full login functionality was not implemented. Non-functional expectations included reliable database constraints, fast access for small data sets, maintainability through clear schema separation, and data integrity enforced through foreign keys.

3.2 System Architecture

The architecture is straightforward: MySQL functions as the data layer, storing persistent information. The backend, written in Java with Spring Boot, communicates with the database through repositories and exposes data through REST endpoints. The frontend, built using React, interacts with the backend through HTTP requests and renders results to the user. The backend runs on port 8080 and responds to API calls, while the frontend runs locally at port 3000 as the client interface.

3.3 Algorithm Design

The borrowing algorithm inserts a new record into BorrowDetail linking a member with a book copy and sets the borrow and due dates. Returning updates the same record by inserting a return date. Availability of book copies is toggled depending on whether a book is borrowed or returned. The search algorithm filters by keyword or ISBN, retrieving matching book records through SQL queries and returning them through API responses.

4. Implementation Section

4.1 Code Structure Summary

The backend source directory is divided into controllers, services, and repositories. Controllers receive requests and route them to services. Services contain core logic for create, read, update, and delete operations. Repositories communicate directly with the database through JPA. The resources folder stores configuration files like application.properties. The frontend is kept separate, containing pages, components, and utility code that sends fetch requests to the backend. The project structure makes debugging easier because responsibilities are well-separated.

```SQL
CREATE TABLE Book
(
  Genre VARCHAR(255) NOT NULL,
  ISBN INT NOT NULL,
  Title VARCHAR(255) NOT NULL,
  PublicationYear VARCHAR(255) NOT NULL,
  Publisher VARCHAR(255) NOT NULL,
  AuthorFirstName VARCHAR(255) NOT NULL,
  AuthorLastName VARCHAR(255) NOT NULL,
  PRIMARY KEY (ISBN),
  UNIQUE (ISBN)
);

CREATE TABLE Member
(
  MemberID INT NOT NULL,
  MemberExpiryDate DATE NOT NULL,
  JoinDate DATE NOT NULL,
  Password VARCHAR(255) NOT NULL,
  DateOfBirth DATE NOT NULL,
  MemberStatus ENUM('Active', 'Not Active') NOT NULL,
  Address VARCHAR(255) NOT NULL,
  Email VARCHAR(255),
  PhoneNumber VARCHAR(30),
  Gender ENUM('Male', 'Female') NOT NULL,
  MemberFirstName VARCHAR(255) NOT NULL,
  MemberLastName VARCHAR(255) NOT NULL,
  PRIMARY KEY (MemberID),
  UNIQUE (Email)
);
```

```
CREATE TABLE Staff
(
  StaffID INT NOT NULL,
  StaffEmail VARCHAR(255) NOT NULL,
  Role VARCHAR(255) NOT NULL,
  Password VARCHAR(255) NOT NULL,
  EmploymentStatus ENUM('Active', 'Not Active') NOT NULL,
  PhoneNumber VARCHAR(30) NOT NULL,
  StaffFirstName VARCHAR(255) NOT NULL,
  StaffLastName VARCHAR(255) NOT NULL,
  PRIMARY KEY (StaffID),
  UNIQUE (StaffEmail)
);

CREATE TABLE BookCopy
(
  BookID INT NOT NULL,
  Availability ENUM('Available', 'Borrowed', 'Unavailable') NOT NULL,
  ISBN INT NOT NULL,
  StaffID INT NOT NULL,
  PRIMARY KEY (BookID),
  FOREIGN KEY (ISBN) REFERENCES Book(ISBN),
  FOREIGN KEY (StaffID) REFERENCES Staff(StaffID)
);

CREATE TABLE BorrowDetail
(
  BorrowID INT NOT NULL,
  BorrowDate DATE NOT NULL,
  DueDate DATE NOT NULL,
  ReturnDate DATE,
  BookID INT NOT NULL,
  StaffID INT NOT NULL,
  MemberID INT NOT NULL,
  PRIMARY KEY (BorrowID),
  FOREIGN KEY (BookID) REFERENCES BookCopy(BookID),
  FOREIGN KEY (StaffID) REFERENCES Staff(StaffID),
  FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
);

CREATE TABLE Fine
(
  FineID INT NOT NULL,
  Amount INT NOT NULL,
```

```
  DateIssued DATE NOT NULL,
  FineStatus ENUM('Paid', 'Not Paid') NOT NULL,
  DatePaid DATE,
  StaffID INT NOT NULL,
  BorrowID INT NOT NULL,
  PRIMARY KEY (FineID),
  FOREIGN KEY (StaffID) REFERENCES Staff(StaffID),
  FOREIGN KEY (BorrowID) REFERENCES BorrowDetail(BorrowID)
);
```

4.2 Integration Summary

Integration involved running the backend first and ensuring the database connection was active. The frontend was started afterwards using Node so it could fetch data from the backend API. Most CRUD routes were tested using HTTP requests before connecting UI elements, making debugging simpler. Once both servers ran simultaneously, the UI displayed book information and allowed interactions like adding or viewing records.

4.3 Challenges and Solutions

During local deployment and integration tests, several technical constraints appeared. One noticeable issue was the handling of ISBN values. In reality, ISBN consists of thirteen digits and often contains hyphens. However, implementing thirteen-digit ISBN with formatting proved difficult during development, especially when inserting book data through SQL and the API, because input validation and formatting were not yet supported. To avoid insert failures, ISBN was temporarily stored as an integer with eleven digits without separators. The system remains functional, but this limitation prevents insertion of true ISBN-13 values unless converted or reformatted. This is recorded as one of the design compromises made under time pressure.

Another uncertainty arose regarding Maven. The backend runs using Spring Boot, which normally manages dependencies through Maven automatically. In testing, the system worked because the device already had Maven installed beforehand. It is unclear whether fresh machines require manual Maven installation, or whether the bundled wrapper would suffice. This creates ambiguity in deployment instructions, suggesting that installation documentation needs refinement before production use.

5. Testing & Results

5.1 Test Report

Setting up the project revealed that the system only became fully operational after creating at least one staff account manually in the database. Without an existing staff user, login attempts failed and the frontend could not progress beyond the authentication screen. After inserting a sample staff record and confirming its visibility through http://localhost:8080/staff, the login process succeeded. This confirmed that backend-frontend interaction is functional and that staff bootstrapping is a necessary step for first-time setup. These findings emphasize that installation and run instructions should explicitly guide new users through initial database seeding.

5.2 Discussion of Results

Based on the integration and debugging process, the system is able to run as intended. The backend responds correctly through API routes, CRUD operations function normally, and the frontend can be accessed through localhost:3000 after login. Successful authentication and dashboard access confirm that the database, backend, and frontend are communicating properly. The only minor limitations discovered during testing involve ISBN formatting and dependency clarity. Since ISBN is stored as an integer with eleven digits, inserting full ISBN-13 values is not always straightforward and often requires removing separators before input. Additionally, although the system worked on a machine that already had Maven installed, it is still uncertain whether a clean device would require manual Maven installation. Despite these small gaps, the system operates correctly and fulfills the core borrowing-tracking workflow.

6. Conclusion and Future Work

Overall, the project reached a functional state. The website can run locally, the database schema supports the borrowing system end-to-end, CRUD operations execute normally, and integration between backend and frontend is proven during testing. The system behaves according to design, with only two technical notes carried forward: ISBN handling is simplified to an integer format rather than full ISBN-13, and dependency setup (especially Maven) still needs more clarity for first-time installations. These issues do not stop the system from operating, but they are areas that can be refined for a more polished and user-friendly deployment process.

6.1 Lessons Learned

Initially, database design felt abstract, but translating ERD and schema into code demonstrated how structure becomes functional logic. The project highlighted the importance of planning UI/UX early instead of designing interface last. We learned how backend and frontend talk to each other, and how SQL tables enforce behavior at data level. A working website emerging from an ERD was the clearest learning outcome.

6.2 Future Work

The next improvement is developing a polished interface. Assigning one team member to UI/UX would reduce workflow confusion. Fine calculation could be automated rather than manually assigned.

Future development should also focus on improving ISBN support by converting the field into a string-based format that can store full ISBN-13 values along with hyphen structure. Documentation for installation should also be improved, ensuring that developers know whether Maven needs to be installed manually or whether a wrapper can handle it automatically. Clearer onboarding instructions and automated setup would make the system easier for new contributors to run without confusion.