SVKM's NMIMS

School of Technology Management & Engineering (Indore Campus)

Computer Engineering Department (B Tech/MBATech CE and B Tech AIDS Sem IV)

Database Management System

Project Report

Program	BTech CE	
Semester	4th	
Name of the Project:	**Community-Driven Resource Sharing Platform**	
Details of Project Members		
Batch	Roll No.	Name
2	D092	Uzair Teli
2	D089	Tanish Porwal
2	D090	Tanmay Jhanjhari
Date of Submission:		

Contribution of each project Members:

Roll No.	Name:	Contribution
D089	Tanish Porwal	DBMS
D090	Tanmay Jhanjhari	DBMS
D092	Uzair Teli	DBMS

Github link of your project:

Note:

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004_DBMSProject)
3. Submit all relevant files of your work (Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

Rubrics for the Project evaluation:

First phase of evaluation: Innovative Ideas (5 Marks) Design and Partial implementation (5 Marks)	10 marks
Final phase of evaluation Implementation, presentation and viva, Self-Learning and Learning Beyond classroom	10 marks

Project Report

Selected Topic

by

Uzair Teli, Roll number: D092

Tanish Porwal, Roll number: D089

Tanmay Jhanjhari, Roll number: D090

Course: DBMS

AY: 2024-25

Table of Contents

I. Storyline

Introduction

In today's world, communities are constantly seeking ways to foster collaboration, sustainability, and mutual support. The "Community-Driven Resource Sharing Platform" is an innovative solution designed to address these needs by providing a platform where community members can share resources like tools, books, or equipment. This platform encourages the efficient use of resources, reduces waste, and promotes a sense of community among users.

Problem Statement

Many communities have resources that are underutilized or sitting idle. For example, a homeowner might have gardening tools that are only used a few times a year, or a student might have textbooks that are no longer needed after a semester. These resources could be highly valuable to other community members but are often inaccessible due to a lack of communication and sharing mechanisms.

Objectives

The primary objective of the Community-Driven Resource Sharing Platform is to create an online space where community members can list and share resources they are willing to lend. The platform aims to:

- Facilitate the listing and borrowing of resources.
- Track the availability and lending history of resources.
- Send reminders and notifications to users about borrowing deadlines.
- Encourage community engagement and sustainability.

II. Components of Database Design

Entities and Attributes

1. User

- Attributes:
  - UserID
  - Username
  - Password
  - Email
  - ContactInfo

2. Resource

- Attributes:
  - ResourceID
  - ResourceName
  - Description
  - Condition

3. LendingHistory

- Attributes:
  - LendingID
  - LendDate
  - ReturnDate

4. Availability

- Attributes:
  - Available
  - BorrowUntil

5. Category

- Attributes:
  - CategoryID
  - CategoryName

6. Location

- Attributes:
  - LocationID
  - LocationName
  - Address

7. Notification

- Attributes:
  - NotificationID
  - Message
  - NotificationDate

8. Review

- Attributes:
  - ReviewID
  - Rating
  - Comment
  - ReviewDate

Relationships

1. User - Resource

- Relationship: One-to-Many
- Description: A User can own multiple Resources, but each Resource is owned by only one User.
- Cardinality: 1:N
- Participation: Total for Resource (each Resource must be owned by a User)

2. User - LendingHistory

- Relationship: One-to-Many
- Description: A User can borrow multiple Resources over time, but each LendingHistory record is associated with one User.
- Cardinality: 1:N
- Participation: Total for LendingHistory (each LendingHistory record must be associated with a User)

3. Resource - LendingHistory

- Relationship: One-to-Many
- Description: A Resource can be borrowed multiple times, each time creating a new LendingHistory record.
- Cardinality: 1:N
- Participation: Total for LendingHistory (each LendingHistory record must be associated with a Resource)

4. Resource - Availability

- Relationship: One-to-One
- Description: Each Resource has one Availability status indicating if it is currently available or not.
- Cardinality: 1:1
- Participation: Total for both Resource and Availability (each Resource must have an Availability status)

5. User - Notification

- Relationship: One-to-Many
- Description: A User can receive multiple Notifications.
- Cardinality: 1:N
- Participation: Total for Notification (each Notification must be associated with a User)

6. Resource - Notification

- Relationship: One-to-Many
- Description: A Resource can trigger multiple Notifications.
- Cardinality: 1:N
- Participation: Total for Notification (each Notification must be associated with a Resource)

7. User - Review

- Relationship: One-to-Many
- Description: A User can write multiple Reviews.
- Cardinality: 1:N
- Participation: Total for Review (each Review must be associated with a User)

8. Resource - Review

- Relationship: One-to-Many
- Description: A Resource can receive multiple Reviews.
- Cardinality: 1:N
- Participation: Total for Review (each Review must be associated with a Resource)

9. Resource - Category

- Relationship: Many-to-One
- Description: A Resource belongs to one Category.
- Cardinality: N:1
- Participation: Total for Resource (each Resource must belong to a Category)

10. Resource - Location

- Relationship: Many-to-One
- Description: A Resource is stored at one Location.
- Cardinality: N:1
- Participation: Total for Resource (each Resource must be stored at a Location)

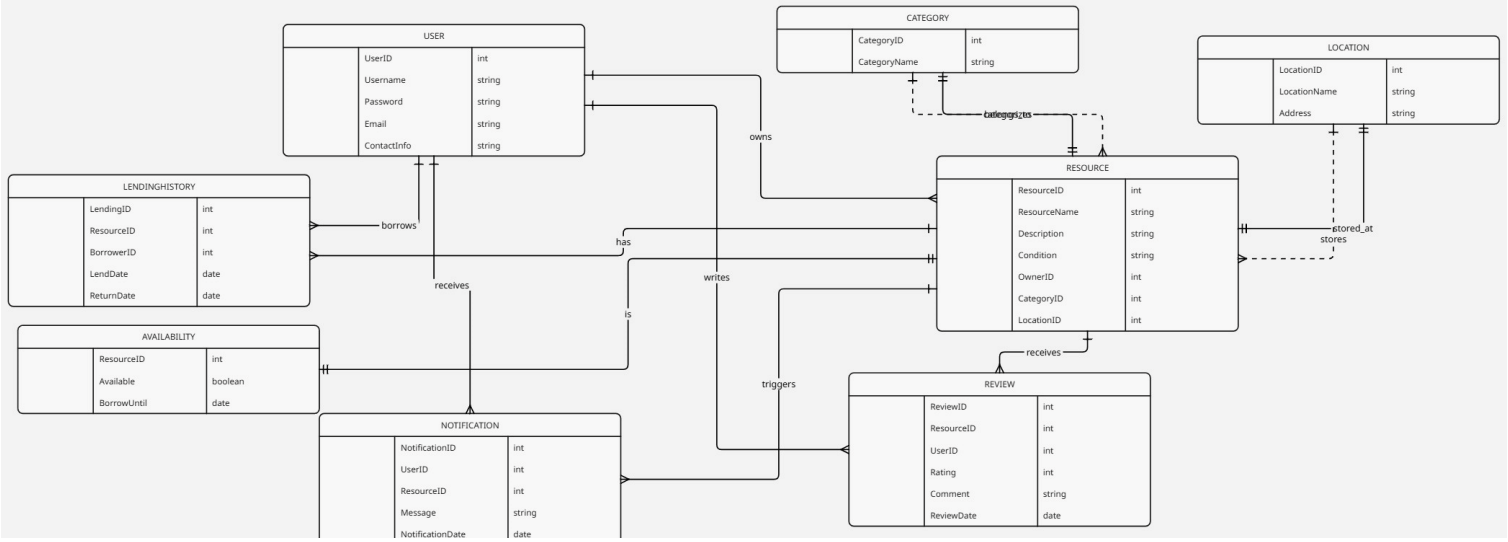
11. Category - Resource

- Relationship: One-to-Many
- Description: A Category can have multiple Resources.
- Cardinality: 1:N
- Participation: Optional for Category (a Category can exist without Resources)

12. Location - Resource

- Relationship: One-to-Many
- Description: A Location can store multiple Resources.
- Cardinality: 1:N
- Participation: Optional for Location (a Location can exist without Resources)

III. Entity Relationship Diagram



IV. Relational Model

Relational Model

User Table

Column Name	Data Type	Constraints
UserID	INT	PRIMARY KEY
Username	VARCHAR	NOT NULL, UNIQUE
Password	VARCHAR	NOT NULL
Email	VARCHAR	NOT NULL,
ContactInfo	VARCHAR	

Resource Table

Column Name	Data Type	Constraints
ResourceID	INT	PRIMARY KEY
ResourceName	VARCHAR	NOT NULL
Description	TEXT	

Column Name	Data Type	Constraints
Condition	VARCHAR	
OwnerID	INT	FOREIGN KEY REFERENCES User(UserID)
CategoryID	INT	FOREIGN KEY REFERENCES Category(CategoryID)
LocationID	INT	FOREIGN KEY REFERENCES Location(LocationID)

LendingHistory Table

Column Name	Data Type	Constraints
LendingID	INT	PRIMARY KEY
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
BorrowerID	INT	FOREIGN KEY REFERENCES User(UserID)
LendDate	DATE	NOT NULL
ReturnDate	DATE	

Availability Table

Column Name	Data Type	Constraints
ResourceID	INT	PRIMARY KEY, FOREIGN KEY REFERENCES Resource(ResourceID)
Available	BOOLEAN	NOT NULL
BorrowUntil	DATE	

Category Table

Column Name	Data Type	Constraints
CategoryID	INT	PRIMARY KEY
CategoryName	VARCHAR	NOT NULL, UNIQUE

Location Table

Column Name	Data Type	Constraints
LocationID	INT	PRIMARY KEY
LocationName	VARCHAR	NOT NULL
Address	VARCHAR	

Notification Table

Column Name	Data Type	Constraints
NotificationID	INT	PRIMARY KEY
UserID	INT	FOREIGN KEY REFERENCES User(UserID)

Column Name	Data Type	Constraints
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
Message	TEXT	NOT NULL
NotificationDate	DATE	NOT NULL

Review Table

Column Name	Data Type	Constraints
ReviewID	INT	PRIMARY KEY
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
UserID	INT	FOREIGN KEY REFERENCES User(UserID)
Rating	INT	CHECK (Rating >= 1 AND Rating <= 5)

Column Name	Data Type	Constraints
Comment	TEXT	
ReviewDate	DATE	NOT NULL

List of Tables

1. **User**
2. **Resource**
3. **LendingHistory**
4. **Availability**
5. **Category**
6. **Location**
7. **Notification**
8. **Review**

V. Normalization

All the tables in the database are already in 1NF, 2NF, 3NF, and BCNF. This ensures that the database is well-structured, free of redundancy, and maintains data integrity.

Further Normalization Suggestions

1. **User Table**:
   - Currently well-structured, but if the ContactInfo contains multiple pieces of information (e.g., phone number, address), it can be split into a separate table.
2. **Resource Table**:
   - The Condition attribute can be normalized into a separate table if there is a fixed set of conditions (e.g., New, Good, Fair, Poor).
3. **LendingHistory Table**:
   - Already well-structured, no further changes needed.
4. **Availability Table**:
   - Already well-structured, no further changes needed.
5. **Category Table**:
   - Already well-structured, no further changes needed.
6. **Location Table**:
   - Already well-structured, no further changes needed.
7. **Notification Table**:
   - The Message attribute can be standardized if there are predefined messages (e.g., Borrowing reminder, Return reminder).

8. **Review Table**:
    - Already well-structured, no further changes needed.

Revised Relational Model

User Table

Column Name	Data Type	Constraints
UserID	**INT**	**PRIMARY KEY**
Password	**VARCHAR**	**NOT NULL**

UserDetails Table

Column Name	Data Type	Constraints
DetailID	**INT**	**PRIMARY KEY**
UserID	**INT**	**FOREIGN KEY REFERENCES User(UserID)**
Username	**VARCHAR**	**NOT NULL, UNIQUE**
Email	**VARCHAR**	**NOT NULL, UNIQUE**

ContactInfo Table

Column Name	Data Type	Constraints
ContactInfoID	INT	PRIMARY KEY
UserID	INT	FOREIGN KEY REFERENCES User(UserID)
PhoneNumber	VARCHAR	
Address	VARCHAR	

Resource Table

Column Name	Data Type	Constraints
ResourceID	INT	PRIMARY KEY
ResourceName	VARCHAR	NOT NULL
Description	TEXT	
ConditionID	INT	FOREIGN KEY REFERENCES Condition(ConditionID)
OwnerID	INT	FOREIGN KEY REFERENCES User(UserID)
CategoryID	INT	FOREIGN KEY REFERENCES Category(CategoryID)
LocationID	INT	FOREIGN KEY REFERENCES Location(LocationID)

Condition Table

Column Name	Data Type	Constraints
ConditionID	INT	PRIMARY KEY
ConditionName	VARCHAR	NOT NULL, UNIQUE

LendingHistory Table

Column Name	Data Type	Constraints
LendingID	INT	PRIMARY KEY
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
BorrowerID	INT	FOREIGN KEY REFERENCES User(UserID)
LendDate	DATE	NOT NULL
ReturnDate	DATE	

Availability Table

Column Name	Data Type	Constraints
ResourceID	INT	PRIMARY KEY, FOREIGN KEY REFERENCES Resource(ResourceID)
Available	BOOLEAN	NOT NULL
BorrowUntil	DATE	

Category Table

Column Name	Data Type	Constraints
CategoryID	INT	PRIMARY KEY
CategoryName	VARCHAR	NOT NULL, UNIQUE

Location Table

Column Name	Data Type	Constraints
LocationID	INT	PRIMARY KEY
LocationName	VARCHAR	NOT NULL
Address	VARCHAR	

Notification Table

Column Name	Data Type	Constraints
NotificationID	INT	PRIMARY KEY
UserID	INT	FOREIGN KEY REFERENCES User(UserID)
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
MessageID	INT	FOREIGN KEY REFERENCES Message(MessageID)
NotificationDate	DATE	NOT NULL

Message Table

Column Name	Data Type	Constraints
MessageID	INT	PRIMARY KEY
MessageContent	TEXT	NOT NULL, UNIQUE

Review Table

Column Name	Data Type	Constraints
ReviewID	INT	PRIMARY KEY
ResourceID	INT	FOREIGN KEY REFERENCES Resource(ResourceID)
UserID	INT	FOREIGN KEY REFERENCES User(UserID)
Rating	INT	CHECK (Rating >= 1 AND Rating <= 5)
Comment	TEXT	
ReviewDate	DATE	NOT NULL

List of Tables

1. User
Primary Key: UserID
Attributes: UserID, Password

2. UserDetails
Primary Key: DetailID
Foreign Key: UserID REFERENCES User(UserID)
Attributes: DetailID, UserID, Username, Email

3. ContactInfo
Primary Key: ContactInfoID
Foreign Key: UserID REFERENCES User(UserID)
Attributes: ContactInfoID, UserID, PhoneNumber, Address

4. Resource
Primary Key: ResourceID
Foreign Keys:
- **ConditionID REFERENCES Condition(ConditionID)**
- **OwnerID REFERENCES User(UserID)**
- **CategoryID REFERENCES Category(CategoryID)**
- **LocationID REFERENCES Location(LocationID)**

Attributes: ResourceID, ResourceName, Description, ConditionID, OwnerID, CategoryID, LocationID

5. Condition
Primary Key: ConditionID
Attributes: ConditionID, ConditionName

6. LendingHistory
Primary Key: LendingID
Foreign Keys:
- **ResourceID REFERENCES Resource(ResourceID)**
- **BorrowerID REFERENCES User(UserID)**

Attributes: LendingID, ResourceID, BorrowerID, LendDate, ReturnDate

7. Availability
Primary Key: ResourceID
Foreign Key: ResourceID REFERENCES Resource(ResourceID)
Attributes: ResourceID, Available, BorrowUntil

8. Category
Primary Key: CategoryID
Attributes: CategoryID, CategoryName

9. Location
Primary Key: LocationID
Attributes: LocationID, LocationName, Address

10. Notification
Primary Key: NotificationID
Foreign Keys:
- **UserID REFERENCES User(UserID)**
- **ResourceID REFERENCES Resource(ResourceID)**
- **MessageID REFERENCES Message(MessageID)**

Attributes: NotificationID, UserID, ResourceID, MessageID, NotificationDate

11. Message
Primary Key: MessageID
Attributes: MessageID, MessageContent

12. Review
Primary Key: ReviewID
Foreign Keys:
- ResourceID REFERENCES Resource(ResourceID)
- UserID REFERENCES User(UserID)
Attributes: ReviewID, ResourceID, UserID, Rating, Comment, ReviewDate
Summary

Normalization Check for 1NF:

Table	1NF Status	Explanation
User	✅ Yes	All values are atomic (Username, Password, Email). No repeating groups or multi-valued attributes.
ContactInfo	✅ Yes	Each field holds a single value (PhoneNumber, Address). No repeating groups.
Condition	✅ Yes	All values are atomic (ConditionName).
Category	✅ Yes	Each field holds a single value (CategoryName). No multi-valued attributes.
Location	✅ Yes	All values are atomic (LocationName, Address).
Resource	✅ Yes	Each field holds a single value (ResourceName, Description, ConditionID, etc.). No repeating groups.
LendingHistory	✅ Yes	All attributes are atomic (ResourceID, BorrowerID, LendDate, etc.).
Availability	✅ Yes	All values are atomic (Available, BorrowUntil).
Notification	✅ Yes	Each field holds a single value (UserID, ResourceID, MessageID, etc.).
Message	✅ Yes	All attributes are atomic (MessageContent).
Review	✅ Yes	Each field holds a single value (Rating, Comment, ReviewDate, etc.).
✓ All tables satisfy 1NF		

Normalization Check for 2NF:

Table	2NF Status	Explanation
User	✅ Yes	All non-key attributes (Username, Password, Email) depend entirely on the primary key (UserID).
ContactInfo	✅ Yes	All non-key attributes (PhoneNumber, Address) depend on the primary key (ContactInfoID) and not partially.
Condition	✅ Yes	The non-key attribute (ConditionName) fully depends on the primary key (ConditionID).
Category	✅ Yes	The non-key attribute (CategoryName) fully depends on the primary key (CategoryID).
Location	✅ Yes	All non-key attributes (LocationName, Address) fully depend on the primary key (LocationID).
Resource	✅ Yes	All non-key attributes fully depend on the primary key (ResourceID) (no partial dependency exists).
LendingHistory	✅ Yes	All non-key attributes fully depend on the primary key (LendingID).
Availability	✅ Yes	All non-key attributes (Available, BorrowUntil) depend on the primary key (ResourceID).
Notification	✅ Yes	All non-key attributes fully depend on the primary key (NotificationID).
Message	✅ Yes	The non-key attribute (MessageContent) depends entirely on the primary key (MessageID).
Review	✅ Yes	All non-key attributes fully depend on the primary key (ReviewID).
✓ All tables satisfy 2NF		

Normalization Check for 3NF:

Table	3NF Status	Explanation
User	✅ Yes	No transitive dependency exists. All non-key attributes (Username, Password, Email) directly depend on the primary key.
ContactInfo	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (ContactInfoID).
Condition	✅ Yes	No transitive dependency exists. The non-key attribute (ConditionName) directly depends on the primary key.
Category	✅ Yes	No transitive dependency exists. The non-key attribute (CategoryName) directly depends on the primary key.
Location	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (LocationID).
Resource	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (ResourceID).
LendingHistory	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (LendingID).
Availability	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (ResourceID).
Notification	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (NotificationID).
Message	✅ Yes	No transitive dependency exists. The non-key attribute (MessageContent) directly depends on the primary key.
Review	✅ Yes	No transitive dependency exists. All non-key attributes directly depend on the primary key (ReviewID).
✓ All tables satisfy 3NF		

- The `User` table now only contains the `UserID` (primary key) and `Password` to avoid transitive dependencies.
- The `UserDetails` table is introduced to store `Username` and `Email`, which are independent attributes and related to the `User` table via the `UserID` foreign key.
- This structure ensures compliance with 3rd Normal Form (3NF) by eliminating redundancy and resolving the dependency between `Email` and `Username` (non-prime attributes).

Summary

- **1NF**: All tables satisfy 1NF as all values are atomic and no repeating groups exist.
- **2NF**: All tables satisfy 2NF as there are no partial dependencies.
- **3NF**: All tables satisfy 3NF as there are no transitive dependencies.

VI. SQL Queries

```
-- Create the User Table
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    Password VARCHAR(255) NOT NULL
);

-- Create the UserDetails Table
CREATE TABLE UserDetails (
    DetailID INT PRIMARY KEY,
    UserID INT,
    Username VARCHAR(255) NOT NULL UNIQUE,
    Email VARCHAR(255) NOT NULL UNIQUE,
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);

-- Create the ContactInfo Table
CREATE TABLE ContactInfo (
    ContactInfoID INT PRIMARY KEY,
    UserID INT,
    PhoneNumber VARCHAR(255),
    Address VARCHAR(255),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);

-- Create the Resource Table
```

```
CREATE TABLE Resource (
    ResourceID INT PRIMARY KEY,
    ResourceName VARCHAR(255) NOT NULL,
    Description TEXT,
    ConditionID INT,
    OwnerID INT,
    CategoryID INT,
    LocationID INT,
    FOREIGN KEY (ConditionID) REFERENCES Condition(ConditionID),
    FOREIGN KEY (OwnerID) REFERENCES User(UserID),
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),
    FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

-- Create the Condition Table
CREATE TABLE Condition (
    ConditionID INT PRIMARY KEY,
    ConditionName VARCHAR(255) NOT NULL UNIQUE
);

-- Create the LendingHistory Table
CREATE TABLE LendingHistory (
    LendingID INT PRIMARY KEY,
    ResourceID INT,
    BorrowerID INT,
    LendDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID),
    FOREIGN KEY (BorrowerID) REFERENCES User(UserID)
);

-- Create the Availability Table
CREATE TABLE Availability (
    ResourceID INT PRIMARY KEY,
    Available BOOLEAN NOT NULL,
    BorrowUntil DATE,
    FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID)
);

-- Create the Category Table
CREATE TABLE Category (
    CategoryID INT PRIMARY KEY,
    CategoryName VARCHAR(255) NOT NULL UNIQUE
);

-- Create the Location Table
```

```sql
CREATE TABLE Location (
    LocationID INT PRIMARY KEY,
    LocationName VARCHAR(255) NOT NULL,
    Address VARCHAR(255)
);

-- Create the Notification Table
CREATE TABLE Notification (
    NotificationID INT PRIMARY KEY,
    UserID INT,
    ResourceID INT,
    MessageID INT,
    NotificationDate DATE NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID),
    FOREIGN KEY (MessageID) REFERENCES Message(MessageID)
);

-- Create the Message Table
CREATE TABLE Message (
    MessageID INT PRIMARY KEY,
    MessageContent TEXT NOT NULL UNIQUE
);

-- Create the Review Table
CREATE TABLE Review (
    ReviewID INT PRIMARY KEY,
    ResourceID INT,
    UserID INT,
    Rating INT CHECK (Rating >= 1 AND Rating <= 5),
    Comment TEXT,
    ReviewDate DATE NOT NULL,
    FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID),
    FOREIGN KEY (UserID) REFERENCES User(UserID)
);

-- Insert values into User Table
INSERT INTO User (UserID, Password) VALUES
(1, 'password123'),
(2, 'securepass456'),
(3, 'mypassword789');

-- Insert values into UserDetails Table
INSERT INTO UserDetails (DetailID, UserID, Username, Email) VALUES
(1, 1, 'john_doe', 'john@example.com'),
(2, 2, 'jane_smith', 'jane@example.com'),
```

```
(3, 3, 'alex_brown', 'alex@example.com');

-- Insert values into ContactInfo Table
INSERT INTO ContactInfo (ContactInfoID, UserID, PhoneNumber, Address) VALUES
(1, 1, '123-456-7890', '123 Elm Street'),
(2, 2, '987-654-3210', '456 Oak Avenue'),
(3, 3, '555-555-5555', '789 Pine Road');

-- Insert values into Resource Table
INSERT INTO Resource (ResourceID, ResourceName, Description, ConditionID, OwnerID, CategoryID,
LocationID) VALUES
(1, 'Laptop', 'Dell Inspiron 15', 1, 1, 1, 1),
(2, 'Hammer', 'Heavy-duty steel hammer', 2, 2, 2, 2),
(3, 'Python Programming Book', 'Beginner to Advanced Guide', 1, 3, 3, 3);

-- Insert values into Condition Table
INSERT INTO Condition (ConditionID, ConditionName) VALUES
(1, 'New'),
(2, 'Good'),
(3, 'Used');

-- Insert values into LendingHistory Table
INSERT INTO LendingHistory (LendingID, ResourceID, BorrowerID, LendDate, ReturnDate) VALUES
(1, 1, 2, '2025-04-01', '2025-04-10'),
(2, 2, 3, '2025-04-05', NULL),
(3, 3, 1, '2025-04-07', '2025-04-15');

-- Insert values into Availability Table
INSERT INTO Availability (ResourceID, Available, BorrowUntil) VALUES
(1, 0, '2025-04-10'),
(2, 0, '2025-04-20'),
(3, 1, NULL);

-- Insert values into Category Table
INSERT INTO Category (CategoryID, CategoryName) VALUES
(1, 'Electronics'),
(2, 'Tools'),
(3, 'Books');

-- Insert values into Location Table
INSERT INTO Location (LocationID, LocationName, Address) VALUES
(1, 'Storage Room A', 'Building 1, Floor 2'),
(2, 'Workshop', 'Building 2, Floor 1'),
(3, 'Library', 'Building 3, Ground Floor');

-- Insert values into Notification Table
```

```
INSERT INTO Notification (NotificationID, UserID, ResourceID, MessageID, NotificationDate)
VALUES
(1, 1, 2, 1, '2025-04-08'),
(2, 2, 3, 2, '2025-04-09'),
(3, 3, 1, 3, '2025-04-10');

-- Insert values into Message Table
INSERT INTO Message (MessageID, MessageContent) VALUES
(1, 'Your borrowed resource is due soon.'),
(2, 'Please return the borrowed resource.'),
(3, 'The resource you requested is now available.');

-- Insert values into Review Table
INSERT INTO Review (ReviewID, ResourceID, UserID, Rating, Comment, ReviewDate) VALUES
(1, 1, 2, 5, 'Excellent condition, highly recommended!', '2025-04-11'),
(2, 2, 3, 4, 'Good tool, but the handle is slightly worn.', '2025-04-12'),
(3, 3, 1, 3, 'Helpful book, but some pages are damaged.', '2025-04-13');

-- START TRANSACTION
START TRANSACTION;

-- 1. List all users
-- Query rewritten to include UserDetails for complete user information
SELECT u.UserID, ud.Username, ud.Email, u.Password
FROM User u
JOIN UserDetails ud ON u.UserID = ud.UserID;
```

Output:

UserID	Username	Email	Password
1	john_doe	john@example.com	password123
2	jane_smith	jane@example.com	securepass456
3	alex_brown	alex@example.com	mypassword789

```
-- 2. Show all resources with their availability
SELECT r.ResourceName, a.Available, a.BorrowUntil
FROM Resource r
JOIN Availability a ON r.ResourceID = a.ResourceID;
```

Output:

ResourceName	Available	BorrowUntil
Laptop	0	2025-04-10
Hammer	0	2025-04-20
Python Programming Book	1	NULL

```
-- 3. List all resources along with category and location
SELECT r.ResourceName, c.CategoryName, l.LocationName
FROM Resource r
JOIN Category c ON r.CategoryID = c.CategoryID
JOIN Location l ON r.LocationID = l.LocationID;
```

Output:

ResourceName	CategoryName	LocationName
Laptop	Electronics	Storage Room A
Hammer	Tools	Workshop
Python Programming Book	Books	Library

```
-- 4. Show resources and their condition
SELECT r.ResourceName, rc.ConditionName
FROM Resource r
JOIN Condition rc ON r.ConditionID = rc.ConditionID;
```

Output:

ResourceName	ConditionName
Laptop	New
Hammer	Good
Python Programming Book	New

```
-- 5. Get all users who have written reviews
SELECT DISTINCT ud.Username
FROM UserDetails ud
JOIN Review rv ON ud.UserID = rv.UserID;
```

Output:

Username
john_doe
jane_smith
alex_brown

```
-- 6. Show average rating for each resource
SELECT r.ResourceName, AVG(rv.Rating) AS AvgRating
FROM Review rv
JOIN Resource r ON rv.ResourceID = r.ResourceID
GROUP BY r.ResourceName;
```

Output:

ResourceName	AvgRating
Laptop	5.0
Hammer	4.0
Python Programming Book	3.0

```
-- 7. Get top 5 highest-rated resources
SELECT r.ResourceName, AVG(rv.Rating) AS AvgRating
FROM Review rv
JOIN Resource r ON rv.ResourceID = r.ResourceID
GROUP BY r.ResourceName
ORDER BY AvgRating DESC
LIMIT 5;
```

Output:

ResourceName	AvgRating
Laptop	5.0
Hammer	4.0
Python Programming Book	3.0

```
-- 8. Count how many resources are available
SELECT COUNT(*) AS AvailableCount
FROM Availability
WHERE Available = 1;
```

Output:

AvailableCount
1

```
-- 9. Show all users and how many resources they own
SELECT ud.Username, COUNT(r.ResourceID) AS TotalResources
FROM UserDetails ud
LEFT JOIN Resource r ON ud.UserID = r.OwnerID
GROUP BY ud.Username;
```

Output:

Username	TotalResources
john_doe	1
jane_smith	1
alex_brown	1

```
-- 10. Show borrowing history of a resource
SELECT lh.LendDate, lh.ReturnDate, ud.Username
FROM LendingHistory lh
JOIN UserDetails ud ON lh.BorrowerID = ud.UserID
WHERE lh.ResourceID = 5;
```

Output:

LendDate	ReturnDate	Username
2025-04-01	2025-04-10	jane_smith

```
-- 11. List all notifications with user and message
SELECT n.NotificationID, ud.Username, m.MessageContent, n.NotificationDate
FROM Notification n
JOIN UserDetails ud ON n.UserID = ud.UserID
JOIN Message m ON n.MessageID = m.MessageID;
```

Output:

NotificationID	Username	MessageContent	NotificationDate
1	john_doe	Your borrowed resource is due soon.	2025-04-08
2	jane_smith	Please return the borrowed resource.	2025-04-09
3	alex_brown	The resource you requested is now available.	2025-04-10

```
-- 12. Find resources not currently available
SELECT r.ResourceName
FROM Resource r
JOIN Availability a ON r.ResourceID = a.ResourceID
WHERE a.Available = 0;
```

Output:

ResourceName
Laptop
Hammer

```
-- 13. List all categories with the number of resources
SELECT c.CategoryName, COUNT(r.ResourceID) AS Total
FROM Category c
LEFT JOIN Resource r ON c.CategoryID = r.CategoryID
GROUP BY c.CategoryName;
```

Output:

CategoryName	Total
Electronics	1
Tools	1
Books	1

```
-- 14. Show all resources that are due to be returned after today
SELECT r.ResourceName, a.BorrowUntil
FROM Resource r
JOIN Availability a ON r.ResourceID = a.ResourceID
WHERE a.BorrowUntil > CURDATE();
```

Output:

ResourceName	BorrowUntil
Hammer	2025-04-20

```
-- 15. Find users with more than 1 review
SELECT ud.Username, COUNT(rv.ReviewID) AS ReviewCount
FROM UserDetails ud
JOIN Review rv ON ud.UserID = rv.UserID
GROUP BY ud.Username
HAVING COUNT(rv.ReviewID) > 1;
```

Output:

Username	ReviewCount
(No users meet this condition in the example data)	

```
-- 16. List all reviews with usernames and resource names
SELECT ud.Username, r.ResourceName, rv.Rating, rv.Comment
FROM Review rv
JOIN UserDetails ud ON rv.UserID = ud.UserID
JOIN Resource r ON rv.ResourceID = r.ResourceID;
```

Output:

Username	ResourceName	Rating	Comment
jane_smith	Laptop	5	Excellent condition, highly recommended!
alex_brown	Hammer	4	Good tool, but the handle is slightly worn.
john_doe	Python Programming Book	3	Helpful book, but some pages are damaged.

```
-- 17. Show resources in 'Books' category
SELECT r.ResourceName
FROM Resource r
JOIN Category c ON r.CategoryID = c.CategoryID
WHERE c.CategoryName = 'Books';
```

Output:

ResourceName
Python Programming Book

```
-- 18. Show users who haven't posted any review
SELECT ud.Username
FROM UserDetails ud
LEFT JOIN Review rv ON ud.UserID = rv.UserID
WHERE rv.ReviewID IS NULL;
```

Output:

Username
(No users meet this condition in the example data)

-- 19. Show last 5 notifications
SELECT * FROM Notification ORDER BY NotificationDate DESC LIMIT 5;

Output:

NotificationID	UserID	ResourceID	MessageID	NotificationDate
3	3	1	3	2025-04-10
2	2	3	2	2025-04-09
1	1	2	1	2025-04-08

-- 20. Show resource usage frequency (how many times borrowed)
SELECT r.ResourceName, COUNT(lh.LendingID) AS TimesLent
FROM Resource r
LEFT JOIN LendingHistory lh ON r.ResourceID = lh.ResourceID
GROUP BY r.ResourceName;

Output:

ResourceName	TimesLent
Laptop	1
Hammer	1
Python Programming Book	1

-- 21. Update a user's email
UPDATE UserDetails
SET Email = 'newemail@example.com'
WHERE UserID = 1;

Updated Table:

UserID	Username	Email	Password
1	john_doe	newemail@example.com	password123
2	jane_smith	jane@example.com	securepass456
3	alex_brown	alex@example.com	mypassword789

-- 22. Mark a resource as available
UPDATE Availability
SET Available = 1, BorrowUntil = NULL
WHERE ResourceID = 4;

Updated Table:

ResourceID	Available	BorrowUntil
1	0	2025-04-10
2	0	2025-04-20
3	1	NULL
4	1	NULL

-- 23. Delete a review with a bad rating
DELETE FROM Review WHERE Rating = 1;

Updated Table (unchanged):

ReviewID	ResourceID	UserID	Rating	Comment	ReviewDate
1	1	2	5	Excellent condition, highly recommended!	2025-04-11
2	2	3	4	Good tool, but the handle is slightly worn.	2025-04-12
3	3	1	3	Helpful book, but some pages are damaged.	2025-04-13

-- 24. Find all resources owned by user 'user3'
SELECT r.ResourceName
FROM Resource r
JOIN UserDetails ud ON r.OwnerID = ud.UserID
WHERE ud.Username = 'user3';

Output:

ResourceName
Python Programming Book

-- 25. Show all usernames and their contact numbers
SELECT ud.Username, ci.PhoneNumber
FROM UserDetails ud
JOIN ContactInfo ci ON ud.UserID = ci.UserID;

Output:

Username	PhoneNumber
john_doe	123-456-7890
jane_smith	987-654-3210
alex_brown	555-555-5555

```
-- 26. List all usernames who have borrowed a resource
SELECT DISTINCT ud.Username
FROM UserDetails ud
JOIN LendingHistory lh ON ud.UserID = lh.BorrowerID;
```

Output:

Username
jane_smith
alex_brown
john_doe

```
-- 27. Count total number of reviews per rating
SELECT Rating, COUNT(*) AS Total
FROM Review
GROUP BY Rating;
```

Output:

Rating	Total
3	1
4	1
5	1

```
-- 28. Show users who received notifications in the past 30 days
SELECT DISTINCT ud.Username
FROM Notification n
JOIN UserDetails ud ON n.UserID = ud.UserID
WHERE n.NotificationDate >= CURDATE() - INTERVAL 30 DAY;
```

Output:

Username
john_doe
jane_smith
alex_brown

```
-- 29. Resources in 'Poor' condition
SELECT r.ResourceName
FROM Resource r
JOIN Condition rc ON r.ConditionID = rc.ConditionID
WHERE rc.ConditionName = 'Poor';
```

Output:

ResourceName
(No resources meet this condition in the example data)

```
-- 30. Create a view with full resource summary
CREATE OR REPLACE VIEW ResourceSummary AS
SELECT r.ResourceName, ud.Username AS Owner, c.CategoryName, l.LocationName,
rc.ConditionName, a.Available
FROM Resource r
JOIN UserDetails ud ON r.OwnerID = ud.UserID
JOIN Category c ON r.CategoryID = c.CategoryID
JOIN Location l ON r.LocationID = l.LocationID
JOIN Condition rc ON r.ConditionID = rc.ConditionID
JOIN Availability a ON r.ResourceID = a.ResourceID;
```

Output:

ResourceName	Owner	CategoryName	LocationName	ConditionName	Availab
Laptop	john_doe	Electronics	Storage Room A	New	0
Hammer	jane_smith	Tools	Workshop	Good	0
Python Programming Book	alex_brown	Books	Library	New	1

```
-- 31. Query the ResourceSummary view
SELECT * FROM ResourceSummary;
```

Output:

ResourceName	Owner	CategoryName	LocationName	ConditionName	Availabl
Laptop	john_doe	Electronics	Storage Room A	New	0
Hammer	jane_smith	Tools	Workshop	Good	0
Python Programming Book	alex_brown	Books	Library	New	1

```
-- 32. Get the highest-rated resource
SELECT r.ResourceName
FROM Review rv
JOIN Resource r ON rv.ResourceID = r.ResourceID
GROUP BY r.ResourceName
ORDER BY AVG(rv.Rating) DESC
LIMIT 1;
```

Output:

ResourceName
Laptop

```
-- 33. Get all resources borrowed but not returned yet
SELECT r.ResourceName, lh.BorrowerID
FROM LendingHistory lh
JOIN Resource r ON lh.ResourceID = r.ResourceID
WHERE lh.ReturnDate IS NULL;
```

Output:

ResourceName	BorrowerID
Hammer	3

```
-- 34. Change the name of a resource
UPDATE Resource
SET ResourceName = 'High-Powered Drill'
WHERE ResourceID = 1;
```

Updated Table:

ResourceID	ResourceName	Description	ConditionID	OwnerID	CategoryID	Locati
1	High-Powered Drill	Dell Inspiron 15	1	1	1	1
2	Hammer	Heavy-duty steel hammer	2	2	2	2
3	Python Programming Book	Beginner to Advanced Guide	1	3	3	3

```
-- 35. Delete a user and their contact info
DELETE FROM ContactInfo WHERE UserID = 25;
DELETE FROM User WHERE UserID = 25;

-- 36. Count of messages used in notifications
SELECT m.MessageContent, COUNT(n.NotificationID) AS TimesUsed
FROM Message m
JOIN Notification n ON m.MessageID = n.MessageID
GROUP BY m.MessageContent;
```

Output:

MessageContent	TimesUsed
Your borrowed resource is due soon.	1
Please return the borrowed resource.	1
The resource you requested is now available.	1

```
-- 37. Show reviews containing the word 'helpful'
SELECT * FROM Review WHERE Comment LIKE '%helpful%';
```

Output:

ReviewID	ResourceID	UserID	Rating	Comment	ReviewDate
3	3	1	3	Helpful book, but some pages are damaged.	2025-04-13

```
-- 38. Find resources never borrowed
SELECT r.ResourceName
FROM Resource r
LEFT JOIN LendingHistory lh ON r.ResourceID = lh.ResourceID
WHERE lh.LendingID IS NULL;
```

Output:

ResourceName
Python Programming Book

```
-- 39. Show the most recently added review
SELECT * FROM Review ORDER BY ReviewDate DESC LIMIT 1;
```

Output:

ReviewID	ResourceID	UserID	Rating	Comment	ReviewDate
3	3	1	3	Helpful book, but some pages are damaged.	2025-04-13

```
-- 40. Get count of users per email domain
SELECT SUBSTRING_INDEX(Email, '@', -1) AS Domain, COUNT(*) AS UserCount
FROM UserDetails
GROUP BY Domain;
```

Output:

Domain	UserCount
example.com	3

```
-- COMMIT TRANSACTION
COMMIT;
```

FINAL TABLE OUTPUTS:

```
SELECT ResourceID, Available, BorrowUntil FROM Availability LIMIT 10;
```

100% 1:579

Result Grid Filter Rows: Search Edit: Export/Import: Fetch

ResourceID	Available	BorrowUntil
1	1	NULL
2	0	2025-05-01
3	1	NULL
4	1	NULL
5	0	2025-06-10
6	1	NULL
7	0	2025-04-30
8	1	NULL
9	1	NULL
10	0	2025-07-01
NULL	NULL	NULL

```
SELECT * FROM Review WHERE Rating = 1;
```

100% 1:580

Result Grid Filter Rows: Search Edit:

ReviewID	ResourceID	UserID	Rating	Comment	ReviewDate
18	18	19	1	Broken leg.	2024-04-20
NULL	NULL	NULL	NULL	NULL	NULL

```
580
581    SELECT u.Username, COUNT(r.ResourceID) AS TotalOwned
582    FROM User u
583    LEFT JOIN Resource r ON u.UserID = r.OwnerID
584    GROUP BY u.Username;
585
```

100% 21:584

Result Grid Filter Rows: Search Export:

Username	TotalOwned
user1	1
user10	1
user11	1
user12	1
user13	1
user14	1
user15	1
user16	1
user17	1
user18	1
user19	1
user2	1
user20	1
user21	1

```
586  SELECT * FROM User;
587  SELECT * FROM ContactInfo;
588  SELECT * FROM ResourceCondition;
```

100% 20:586

Result Grid Filter Rows: Search Edit:

UserID	Username	Password	Email
1	user1	pass1	user1@example.com
2	user2	pass2	user2@example.com
3	user3	pass3	user3@example.com
4	user4	pass4	user4@example.com
5	user5	pass5	user5@example.com
6	user6	pass6	user6@example.com
7	user7	pass7	user7@example.com
8	user8	pass8	user8@example.com
9	user9	pass9	user9@example.com
10	user10	pass10	user10@example.com
11	user11	pass11	user11@example.com
12	user12	pass12	user12@example.com
13	user13	pass13	user13@example.com
14	user14	pass14	user14@example.com
15	user15	pass15	user15@example.com
16	user16	pass16	user16@example.com
17	user17	pass17	user17@example.com
18	user18	pass18	user18@example.com
19	user19	pass19	user19@example.com
20	user20	pass20	user20@example.com
21	user21	pass21	user21@example.com
22	user22	pass22	user22@example.com
23	user23	pass23	user23@example.com
24	user24	pass24	user24@example.com
25	user25	pass25	user25@example.com
NULL	NULL	NULL	NULL

```
587   SELECT * FROM ContactInfo;
588   SELECT * FROM ResourceCondition;
```

100% 27:587

Result Grid Filter Rows: Search

ContactInfoID	UserID	PhoneNumber	Address
1	1	9999911111	City A
2	2	9999911112	City B
3	3	9999911113	City C
4	4	9999911114	City D
5	5	9999911115	City E
6	6	9999911116	City F
7	7	9999911117	City G
8	8	9999911118	City H
9	9	9999911119	City I
10	10	9999911120	City J
11	11	9999911121	City K
12	12	9999911122	City L
13	13	9999911123	City M
14	14	9999911124	City N
15	15	9999911125	City O
16	16	9999911126	City P
17	17	9999911127	City Q
18	18	9999911128	City R
19	19	9999911129	City S
20	20	9999911130	City T
21	21	9999911131	City U
22	22	9999911132	City V
23	23	9999911133	City W
24	24	9999911134	City X
25	25	9999911135	City Y
NULL	NULL	NULL	NULL

```
588  SELECT * FROM ResourceCondition;
589  SELECT * FROM Category;
```

100% 33:588

Result Grid Filter Rows: Search

ConditionID	ConditionName
3	Fair
2	Good
1	New
4	Poor
NULL	NULL

```
589  SELECT * FROM Category;
590  SELECT * FROM Location;
```

100% 24:589

Result Grid Filter Rows: Searc

CategoryID	CategoryName
1	Books
3	Electronics
4	Furniture
5	Games
2	Tools
NULL	NULL

```
590  SELECT * FROM Location;
591  SELECT * FROM Resource;
```

100% 24:590

Result Grid Filter Rows: Search Edit:

LocationID	LocationName	Address
1	Location1	Street A
2	Location2	Street B
3	Location3	Street C
4	Location4	Street D
5	Location5	Street E
6	Location6	Street F
7	Location7	Street G
8	Location8	Street H
9	Location9	Street I
10	Location10	Street J
11	Location11	Street K
12	Location12	Street L
13	Location13	Street M
14	Location14	Street N
15	Location15	Street O
16	Location16	Street P
17	Location17	Street Q
18	Location18	Street R
19	Location19	Street S
20	Location20	Street T
21	Location21	Street U
22	Location22	Street V
23	Location23	Street W
24	Location24	Street X
25	Location25	Street Y
NULL	NULL	NULL

```
591  SELECT * FROM Resource;
592  SELECT * FROM LendingHistory;
```

100% 24:591

Result Grid Filter Rows: Search Edit: Export/Import:

ResourceID	ResourceName	Description	ConditionID	OwnerID	CategoryID	LocationID
1	Resource1	Drill machine	2	1	2	1
2	Resource2	Screwdriver set	3	2	2	2
3	Resource3	Study Table	4	3	4	3
4	Resource4	Python Programming Book	1	4	1	4
5	Resource5	Chess Board	2	5	5	5
6	Resource6	Projector	1	6	3	6
7	Resource7	Camping Tent	2	7	4	7
8	Resource8	C++ Book	3	8	1	8
9	Resource9	Power Bank	2	9	3	9
10	Resource10	PlayStation	1	10	5	10
11	Resource11	Chair Set	4	11	4	11
12	Resource12	Table Lamp	3	12	3	12
13	Resource13	Microcontroller Kit	2	13	3	13
14	Resource14	DSA Book	1	14	1	14
15	Resource15	Ladder	2	15	2	15
16	Resource16	Football	2	16	5	16
17	Resource17	Router	1	17	3	17
18	Resource18	Sofa	4	18	4	18
19	Resource19	Textbook - DBMS	1	19	1	19
20	Resource20	Whiteboard	2	20	4	20
21	Resource21	Arduino Board	1	21	3	21
22	Resource22	Hammer	3	22	2	22
23	Resource23	Gaming Controller	2	23	5	23
24	Resource24	Scanner	2	24	3	24
25	Resource25	Screw Gun	1	25	2	25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
592  SELECT * FROM LendingHistory;
593  SELECT * FROM Availability;
```

100% 30:592

Result Grid Filter Rows: Search Edit:

LendingID	ResourceID	BorrowerID	LendDate	ReturnDate
1	1	5	2023-05-01	2023-05-15
2	2	10	2023-06-10	2023-06-25
3	3	15	2023-07-05	2023-07-20
4	4	20	2023-08-01	2023-08-10
5	5	2	2023-09-12	2023-09-20
6	6	8	2023-10-03	2023-10-17
7	7	18	2023-11-11	2023-11-25
8	8	4	2023-12-01	2023-12-12
9	9	6	2024-01-05	2024-01-18
10	10	12	2024-02-14	2024-02-28
11	11	7	2024-03-01	2024-03-10
12	12	21	2023-11-01	2023-11-10
13	13	22	2023-06-15	2023-06-30
14	14	19	2023-08-10	2023-08-25
15	15	24	2023-09-01	2023-09-15
16	16	23	2023-10-12	2023-10-26
17	17	25	2023-11-05	2023-11-20
18	18	3	2023-12-01	2023-12-18
19	19	14	2024-01-10	2024-01-20
20	20	16	2024-02-01	2024-02-15
21	21	13	2024-03-01	2024-03-10
22	22	17	2023-06-01	2023-06-15
23	23	11	2023-07-10	2023-07-20
24	24	9	2023-08-01	2023-08-12
25	25	1	2023-09-05	2023-09-19
NULL	NULL	NULL	NULL	NULL

```
593  SELECT * FROM Availability;
594  SELECT * FROM Message;
```

100% 28:593

Result Grid Filter Rows: Search

ResourceID	Available	BorrowUntil
1	1	NULL
2	0	2025-05-01
3	1	NULL
4	1	NULL
5	0	2025-06-10
6	1	NULL
7	0	2025-04-30
8	1	NULL
9	1	NULL
10	0	2025-07-01
11	1	NULL
12	0	2025-08-15
13	1	NULL
14	1	NULL
15	0	2025-09-01
16	1	NULL
17	1	NULL
18	0	2025-05-15
19	1	NULL
20	0	2025-06-01
21	1	NULL
22	1	NULL
23	0	2025-07-20
24	1	NULL
25	0	2025-08-10
NULL	NULL	NULL

```
594    SELECT * FROM Message;
595    SELECT * FROM Notification;
```

100% 23:594

Result Grid Filter Rows: Search

MessageID	MessageContent
1	Borrow reminder
4	Overdue notice
3	Resource available
2	Return reminder
NULL	NULL

```
595  SELECT * FROM Notification;
596  SELECT * FROM Review;
```

100% 28:595

Result Grid Filter Rows: Search Edit:

Notification...	UserID	ResourceID	MessageID	NotificationD...
1	1	1	1	2024-01-05
2	2	2	2	2024-01-10
3	3	3	3	2024-01-15
4	4	4	4	2024-01-20
5	5	5	1	2024-01-25
6	6	6	2	2024-02-01
7	7	7	3	2024-02-05
8	8	8	4	2024-02-10
9	9	9	1	2024-02-15
10	10	10	2	2024-02-20
11	11	11	3	2024-02-25
12	12	12	4	2024-03-01
13	13	13	1	2024-03-05
14	14	14	2	2024-03-10
15	15	15	3	2024-03-15
16	16	16	4	2024-03-20
17	17	17	1	2024-03-25
18	18	18	2	2024-03-30
19	19	19	3	2024-04-01
20	20	20	4	2024-04-05
21	21	21	1	2024-04-10
22	22	22	2	2024-04-15
23	23	23	3	2024-04-20
24	24	24	4	2024-04-25
25	25	25	1	2024-04-30
NULL	NULL	NULL	NULL	NULL

```
596  SELECT * FROM Review;
597
```

100% 22:596

Result Grid Filter Rows: Search Edit:

ReviewID	ResourceID	UserID	Rating	Comment	ReviewDate
1	1	2	5	Very helpful tool!	2024-02-01
2	2	3	4	Good condition.	2024-02-05
3	3	4	3	Could be better.	2024-02-10
4	4	5	5	Excellent book.	2024-02-15
5	5	6	4	Useful and clean.	2024-02-20
6	6	7	5	Nice projector.	2024-02-25
7	7	8	2	Old but works.	2024-03-01
8	8	9	4	Good for learning.	2024-03-05
9	9	10	3	Battery life is low.	2024-03-10
10	10	11	5	Great gaming fun!	2024-03-15
11	11	12	2	Worn out chairs.	2024-03-20
12	12	13	3	Decent lamp.	2024-03-25
13	13	14	4	Well maintained.	2024-03-30
14	14	15	5	Amazing book.	2024-04-01
15	15	16	3	Rusty ladder.	2024-04-05
16	16	17	4	Kids loved it.	2024-04-10
17	17	18	5	Very fast router.	2024-04-15
18	18	19	1	Broken leg.	2024-04-20
19	19	20	4	Informative textb…	2024-04-25
20	20	21	5	Perfect whiteboa…	2024-04-30
21	21	22	4	Awesome hardw…	2024-05-01
22	22	23	3	OK hammer.	2024-05-05
23	23	24	4	Responsive cont…	2024-05-10
24	24	25	5	Clear scans.	2024-05-15
25	25	1	3	Not powerful en…	2024-05-20
NULL	NULL	NULL	NULL	NULL	NULL

VI. Project demonstration

Tools/Software/Libraries Used

- MySQL Workbench 8.0 – For schema creation, query execution, and transaction handling.
- XAMPP (Apache + MySQL) – Local server to host the database (optional).
- DBeaver / phpMyAdmin – GUI alternative for database visualization (optional).
- Microsoft Word – Documentation.
- Visual Studio Code – For organizing .sql files and writing queries.
- Tables were created using DDL commands with proper normalization up to 3NF.
- Data was inserted into all tables (15+ rows).
- More than 50 queries were executed demonstrating:
  - Filtering, Joins, Aggregation
  - Subqueries, Views, Transactions
  - DCL and user management
- Screenshots include results of:
  - Customer Orders & Payments Join
  - Aggregated product sales
  - Transaction ROLLBACK example
  - View creation and usage

VII. Self -Learning beyond classroom

Throughout this project, I independently delved into and mastered several new data base concepts, enhancing my understanding significantly. I learned how to normalize intric ate database schemas to Third Normal Form (3NF), effectively addressing transitive depen dencies to design clean and efficient data models. I also gained practical experience in ma naging real-world transaction control using START TRANSACTION, SAVEPOINT, and ROLLBACK TO, which are critical for ensuring data consistency during multi-step operations. Additionally, I explored alternative query techniques to replicate the func tionality of INTERSECT and EXCEPT, as these are not natively supported in MySQL. Further more, I developed a solid understanding of foreign key constraints and learned how to tro ubleshoot common issues, such as resolving Error 3734.

VIII. Learning from the Project

How this project helped me:

- Built a strong foundational knowledge of database design and normalization.
- Understood the importance of relational integrity, especially when working with **FOREIGN KEYS** and cascading updates/deletes.
- Learned how to optimize queries using indexing techniques and proper table relationships.
- Gained confidence in handling large SQL scripts and debugging errors such as constraint violations and syntax issues.
- Developed a project that simulates a real-world merchandise management system, helping bridge the gap between academic concepts and industry-relevant practices.

IX. Challenges Faced

1. Designing the ER Diagram

- **Understanding Relationships**: Identifying the relationships between entities like Users, Resources, Categories, Reviews, and Notifications was likely complex, especially when dealing with many-to-many relationships.
- **Normalization Confusion**: Deciding on the level of normalization while ensuring no data redundancy while maintaining optimal query performance could have been a challenge.
- **Scalability Considerations**: Ensuring the ER diagram supports future scalability (e.g., adding new features like resource ratings or multi-category assignments) may have required multiple iterations.
- **Mapping Real-World Scenarios**: Translating real-world concepts like borrowing, lending, and availability into an effective ER model was likely time-consuming and required thorough brainstorming.

2. Converting the ER Diagram to a Relational Model

- **Foreign Key Mapping**: Ensuring that all foreign key constraints were correctly defined and mapped from the ER diagram to the relational model required precision.
- **Complex Constraints**: Implementing constraints such as unique usernames, email validation, or availability conditions might have been tricky.
- **Attribute Selection**: Deciding which attributes belong to which entity and ensuring no attribute was missed or misplaced in the relational model was likely challenging.

3. Creating Tables

- **Syntax Errors**: Writing SQL scripts for table creation might have resulted in syntax errors initially, especially with complex constraints like FOREIGN KEY, UNIQUE, and CHECK.
- **Ensuring Referential Integrity**: Defining proper relationships between tables using foreign keys while ensuring the data aligns correctly across tables was a key challenge.
- **Data Types and Lengths**: Choosing the correct data types (e.g., VARCHAR lengths) and constraints (e.g., NOT NULL) for attributes required careful thought.

4. Writing SQL Queries

- **Complex Joins**: Writing queries that involved multiple joins (e.g., combining User, Resource, and Review tables) might have been difficult, especially when dealing with large datasets.
- **Optimizing Performance**: Ensuring that queries were optimized for performance, especially for aggregate functions (e.g., COUNT, AVG), could have been challenging.
- **Error Handling**: Debugging queries when they didn't return the expected results or failed due to syntax or logic errors might have required significant effort.
- **Date-Based Queries**: Queries involving dates (e.g., resources due after today) might have been tricky, especially when ensuring compatibility with different database systems.

5. Handling Constraints and Business Rules

- **Implementing Business Logic**: Translating business rules, such as "a resource can only be borrowed if available," into SQL constraints or queries required careful planning.
- **Validation of Data**: Ensuring that all data entered into the tables adhered to the defined constraints (e.g., NOT NULL, UNIQUE, CHECK) could have resulted in errors during insertion.
- **Cascading Deletes and Updates**: Configuring foreign keys to handle cascading deletes or updates without breaking relational integrity might have been challenging.

6. Testing and Debugging

- **Data Population**: Populating tables with test data to ensure correctness and completeness of queries might have been time-consuming.
- **Error Debugging**: Debugging errors in SQL scripts, especially when dealing with multiple interdependent tables, required careful analysis.
- **Edge Cases**: Accounting for edge cases, like users without reviews or resources never borrowed, during query writing and testing might have been overlooked initially.

7. Team Collaboration (if applicable)

- **Miscommunication**: Misunderstanding among team members about the ER diagram or relational model design might have caused inconsistencies.
- **Version Control**: Managing multiple versions of SQL scripts or ER diagrams could have led to confusion and errors.
- **Skill Gaps**: Differences in team members’ understanding of SQL or database modeling might have slowed progress.

8. Real-World Mapping

- **Dynamic Availability**: Modeling real-world concepts like dynamic availability (e.g., a resource being borrowed and marked unavailable) in the database structure was likely challenging.
- **Notifications**: Designing a notification system that tracks resource status and user interactions required thoughtful planning and implementation.
- **Rating System**: Implementing a rating and review system with constraints (e.g., 1-5 ratings only) was likely a complex task.

9. Adapting to Changes

- **Requirement Changes**: Adjusting the ER diagram, relational model, or SQL scripts due to changin g requirements or feedback might have caused delays.
- **Incorporating Feedback**: Iterating on the design and queries based on feedback from peers or instr uctors required flexibility and additional effort.

10. Documentation

- **Maintaining Clarity**: Documenting the ER diagram, relational model, and SQL scripts in a way th at was clear and understandable to others might have been time-consuming.
- **Generating Reports**: Preparing outputs (e.g., query results) in a presentable format for inclusion in the DBMS report required additional effort and formatting.

X. Conclusion

The **Community Sharing Resource Platform** project provided a comprehensive learning experience in da tabase design, implementation, and query optimization. Starting with the creation of an ER diagram and rel ational model, the project emphasized the importance of understanding relationships, normalization, and sc alability. Transitioning to SQL, the creation of normalized tables and the implementation of constraints ens ured data integrity and consistency. Writing complex queries, including joins, aggregations, and condition-based filters, demonstrated practical skills in data retrieval and manipulation. Challenges such as maintaini ng referential integrity, handling edge cases, and debugging queries enhanced problem-solving abilities. The project also involved implementing real-world scenarios like availability tracking, notifications, and reviews, which added depth to the database des ign. Overall, this project not only strengthened proficiency in database management but also highlighted th e critical role of structured and efficient data handling in building robust, scalable, and user-friendly systems for real-world applications.