Assignment 7.1

Given: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Business Scenario: Hotel Management

Entities:

- Guest
- Hotel
- Reservation
- Department
- Staff
- Room

Attributes:

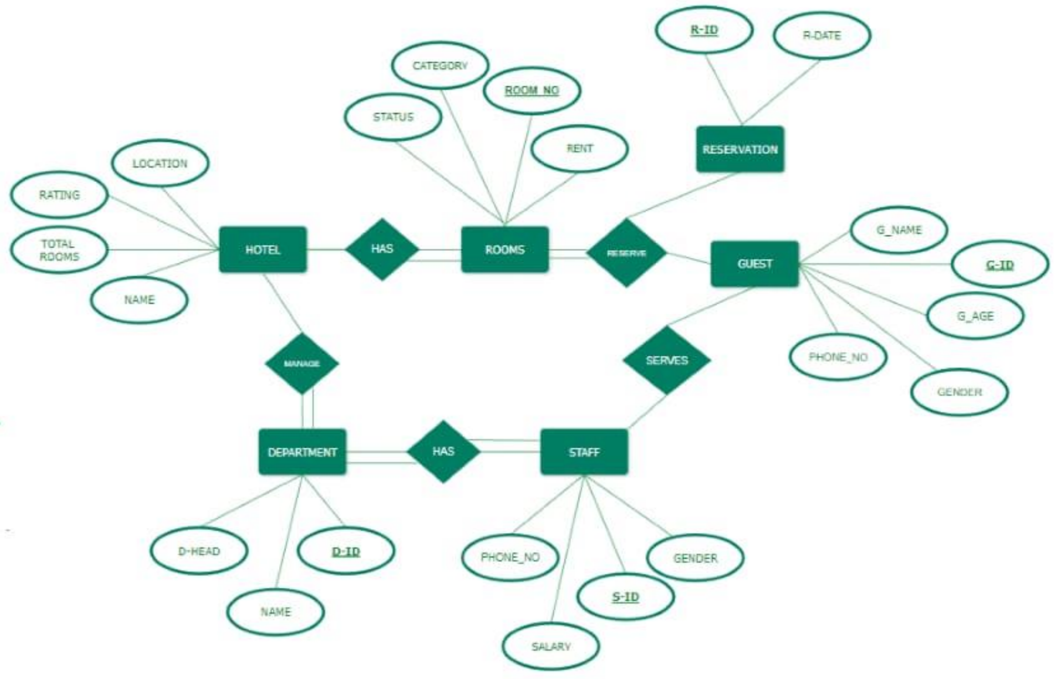
1. **Guest:**
    - Guest_ID
    - Name
    - Contact Information
    - Nationality
    - Gender
    - Reservation_History
2. **Hotel**:
    - Hotel_ID
    - Name
    - Location
    - Number of Rooms
    - Rating
    - Contact Information
3. **Reservation:**
    - Reservation_ID
    - Check-in Date
    - Check-out Date
4. **Department:**
    - Department_ID
    - D_Head
    - D_Role
    - Staff-Count
    - Contact Information
5. **Staff:**
    - Staff_ID
    - Name
    - Age
    - Contact Information
    - Salary

6. **Room:**
    - Room_No.
    - Category
    - Rent
    - Status

<u>**Mapping Relationships and Cardinalities**</u>

Hotel to Room (1:N):
One hotel can have many rooms, but each room belongs to only one hotel.
The relationship between the hotel and its rooms is a one-to-many relationship.
Guest to Reservation (1:N):
A guest can make multiple reservations, but each reservation is made by only one guest.
There is a one-to-many relationship between the guest and reservations.
Reservation to Room (1:1):
A reservation is for one room, but each of the rooms can be reserved multiple times.
This is a one-to-one relationship between reservation and room.
Staff to Department (N:1):
Each staff member works in one department, but each department can have multiple staff members.
For eg. housekeeping staff can also work with restaurant staff or vice versa
There is a many-to-one relationship, as many staff members can belong to one department, but each staff member is associated with only one department.
Hotel and Department (1:N):
- Each hotel has multiple departments, such as Front Desk, Housekeeping, Food and Beverage, Maintenance, and Management.
- Each department operates within a specific hotel.
- This is a one-to-many relationship, as one hotel can have multiple departments, but each department belongs to only one hotel.

ER Diagram for Hotel Management



Assignment 7.2

Given: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.

Tables:

1. **Book**
    - BookID (Primary Key)
    - Title
    - Author
    - Genre
    - PublicationYear
    - ISBN (Unique)

2. **Member**

    - MemberID (Primary Key)
    - Name
    - Address
    - Phone
    - Email (Unique)

3. **Borrowing**

    - BorrowingID (Primary Key)
    - BookID (Foreign Key from Book)
    - MemberID (Foreign Key from Member)
    - BorrowDate
    - ReturnDate
    - Status (e.g., "Borrowed", "Returned")

Constraints:

- All primary key fields (BookID, MemberID, BorrowingID) cannot be NULL.
- ISBN in the Book table is UNIQUE.
- Email in the Member table is UNIQUE.
- BorrowDate must be before ReturnDate.
- Status in the Borrowing table can only have predefined values like "Borrowed", "Returned".

Database Schema:

```
CREATE TABLE Book (

   BookID INT PRIMARY KEY,

   Title VARCHAR(255) NOT NULL,

   Author VARCHAR(255) NOT NULL,

   Genre VARCHAR(100),
   PublicationYear INT,
   ISBN VARCHAR(13) UNIQUE
);

CREATE TABLE Member (
   MemberID INT PRIMARY KEY,
   Name VARCHAR(100) NOT NULL,
   Address VARCHAR(255),
   Phone VARCHAR(15),
   Email VARCHAR(255) UNIQUE
);

CREATE TABLE Borrowing (
   BorrowingID INT PRIMARY KEY,
   BookID INT,
   MemberID INT,
   BorrowDate DATE NOT NULL,
   ReturnDate DATE,
   Status VARCHAR(20) CHECK (Status IN ('Borrowed', 'Returned')),
   FOREIGN KEY (BookID) REFERENCES Book(BookID),
   FOREIGN KEY (MemberID) REFERENCES Member(MemberID),
   CHECK (BorrowDate < ReturnDate)
);
```

Assignment 7.3

Given: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

ACID properties are the golden rules that ensure the reliability and consistency of data in database transactions. Imagine a transaction as a complex recipe with multiple steps. ACID properties make sure the recipe is followed flawlessly, and the final dish (data) is always perfect, even if unexpected things happen in the kitchen (database).

1. **Atomicity:** All or nothing! This property guarantees that a transaction acts as a single unit. Either all the changes within the transaction are successfully applied, or none are. It's like following a recipe exactly. If you run out of sugar halfway through making a cake, you would not add flour and bake a half-baked mess. You would start over with a fresh batch of ingredients (undo the transaction).
2. **Consistency:** This property ensures that a transaction transforms the database from one valid state to another. It's like following a recipe that results in a delicious dish, not a burnt offering. Transactions maintain the predefined data integrity rules of the database.
3. **Isolation:** Even with multiple chefs (concurrent transactions) in the kitchen, their dishes (data modifications) don't interfere with each other. This property ensures that concurrent transactions are executed independently and don't see intermediate changes from other transactions until they are committed. It's like having separate workstations for each chef; they can work on their recipes without affecting each other's ingredients.
4. **Durability:** Once a chef serves a dish (commits a transaction), it stays served. This property guarantees that successful transactions are permanently stored, even in case of system failures. The database ensures the changes are written to permanent storage, not just temporary memory.

SQL statements to simulate a transaction that includes locking :

Scenario: Transferring money between two accounts (AccountID is the primary key):

```
START TRANSACTION;  -- Begin the transaction

-- Lock both accounts to prevent conflicts (pessimistic locking)

SELECT * FROM Account WHERE AccountID = 1 FOR UPDATE;

SELECT * FROM Account WHERE AccountID = 2 FOR UPDATE;



-- Update balance (assuming sufficient funds in account 1)

UPDATE Account SET Balance = Balance - 100 WHERE AccountID = 1;

UPDATE Account SET Balance = Balance + 100 WHERE AccountID = 2;

COMMIT;  -- Commit the transaction, making changes permanent
```

Isolation Levels:

- **READ UNCOMMITTED (RU):** Transactions see uncommitted changes from others, leading to dirty reads (seeing inconsistent data).
- **READ COMMITTED (RC):** Transactions only see committed changes from others, avoiding dirty reads but potentially encountering non-repeatable reads (seeing data that has since been modified by another transaction).
- **REPEATABLE READ (RR):** Transactions don't see changes from other transactions that haven't been committed yet, ensuring both consistent reads and repeatable reads.
- **SERIALIZABLE (SERIAL):** Transactions are executed one at a time, ensuring the highest level of isolation but potentially impacting performance.

Assignment 7.4

Given: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

SQL statements for creating the database and tables, modifying structures, and dropping a table:

1. **Create Database:**

```
CREATE DATABASE Library;
```

2. **Create Tables:**

Members Table:

```
CREATE TABLE Members (

MemberID INT PRIMARY KEY AUTO_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(20)
);
```

Books Table:

```
CREATE TABLE Books (

  BookID INT PRIMARY KEY AUTO_INCREMENT,

  Title VARCHAR(255) NOT NULL,

  Author VARCHAR(100) NOT NULL,

  ISBN VARCHAR(13),

  PublicationYear INT,

  Genre VARCHAR(50)

);
```

Borrowings Table (tracks who borrows which book and when):

```
CREATE TABLE Borrowings (

  BorrowingID INT PRIMARY KEY AUTO_INCREMENT,
  MemberID INT NOT NULL,

  BookID INT NOT NULL,

  BorrowedDate DATE NOT NULL,

  ReturnedDate DATE,

  FOREIGN KEY (MemberID) REFERENCES Members(MemberID),

  FOREIGN KEY (BookID) REFERENCES Books(BookID)

);
```

3. **Modify Table Structure (Adding a Due Date to Borrowings):**

```
ALTER TABLE Borrowings
ADD DueDate DATE;
```

4. **Drop a Redundant Table (assuming you decide the ISBN in Books is enough to uniquely identify a book):**

```
DROP TABLE ISBNs;  -- Replace 'ISBNs' with the actual redundant table name
```

Assignment 7.5

Given: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyze the impact on query execution.

Creating and Analyzing an Index for Improved Query Performance:

Let's consider a table named Customers with columns for CustomerID (primary key), Name, Email, and City. We'll focus on creating an index on the City column and analyse the impact on querying by city.

1. **Creating an Index:**
   CREATE INDEX idx_city ON Customers(City);
   --This statement creates an index named idx_city on the City column of the Customers table. An index acts like a pre-sorted catalog for the specific column, allowing for faster retrieval of data based on that column's values.

2. **Improved Query Performance:**

   Imagine searching for all customers from a particular city, say "New York". Without an index, the database engine would need to scan through every row in the Customers table, comparing each City value to "New York." This can be very slow for large tables.

   However, with the index, the database can efficiently navigate the pre-sorted list of city values in the index. It quickly locates entries for "New York" and then retrieves the corresponding customer data from the main table. This significantly reduces the time required to execute the query.

3. **Dropping the Index and Analysing Impact:**

   DROP INDEX idx_city;

   --This statement removes the idx_city index from the Customers table. Now, if we run the same query to find customers from "New York", the database will revert to the slower method of scanning the entire City column.

4. Observing the Performance Difference:

To analyze the impact, you can use tools provided by your database management system to compare query execution times. Running the same query with and without the index will showcase the significant performance improvement gained by using an index for specific search criteria.

Assignment 7.6

Given: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

1. Creating a New User and Granting Privileges:

```
-- Create a new database user

CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';



-- Grant SELECT, INSERT, UPDATE privileges on a specific database to the user

GRANT SELECT, INSERT, UPDATE ON dbname.* TO 'newuser'@'localhost';
```

2. **Revoking Certain Privileges:**

```
-- Revoke INSERT and UPDATE privileges on a specific table from the user
REVOKE INSERT, UPDATE ON dbname.tablename FROM 'newuser'@'localhost';
```

3. **Dropping the User:**

```
-- Drop the user
DROP USER 'newuser'@'localhost';
--This removes the user from the database system
```

Assignment 7.7

Given: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

1. **INSERT new records into the library tables:**

```
-- Insert a new book record
INSERT INTO Book (BookID, Title, Author, Genre, PublicationYear, ISBN)
VALUES (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925, '9780743273565');

-- Insert a new member record
INSERT INTO Member (MemberID, Name, Address, Phone, Email)
VALUES (1, 'John Doe', '123 Main St', '555-1234', 'john@example.com');
```

2. **UPDATE existing records with new information:**

```
-- Update the address of a member
UPDATE Member
SET Address = '456 Elm St'
WHERE MemberID = 1;
```

3. **DELETE records based on specific criteria:**

```
-- Delete a book record
DELETE FROM Book
WHERE BookID = 1;
```

4. **BULK INSERT operations to load data from an external source:**

```
-- Bulk insert data from a CSV file into the Book table
LOAD DATA INFILE '/path/to/file.csv'
INTO TABLE Book
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(Title, Author, Genre, PublicationYear, ISBN);
```