# Assignment 1:

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.

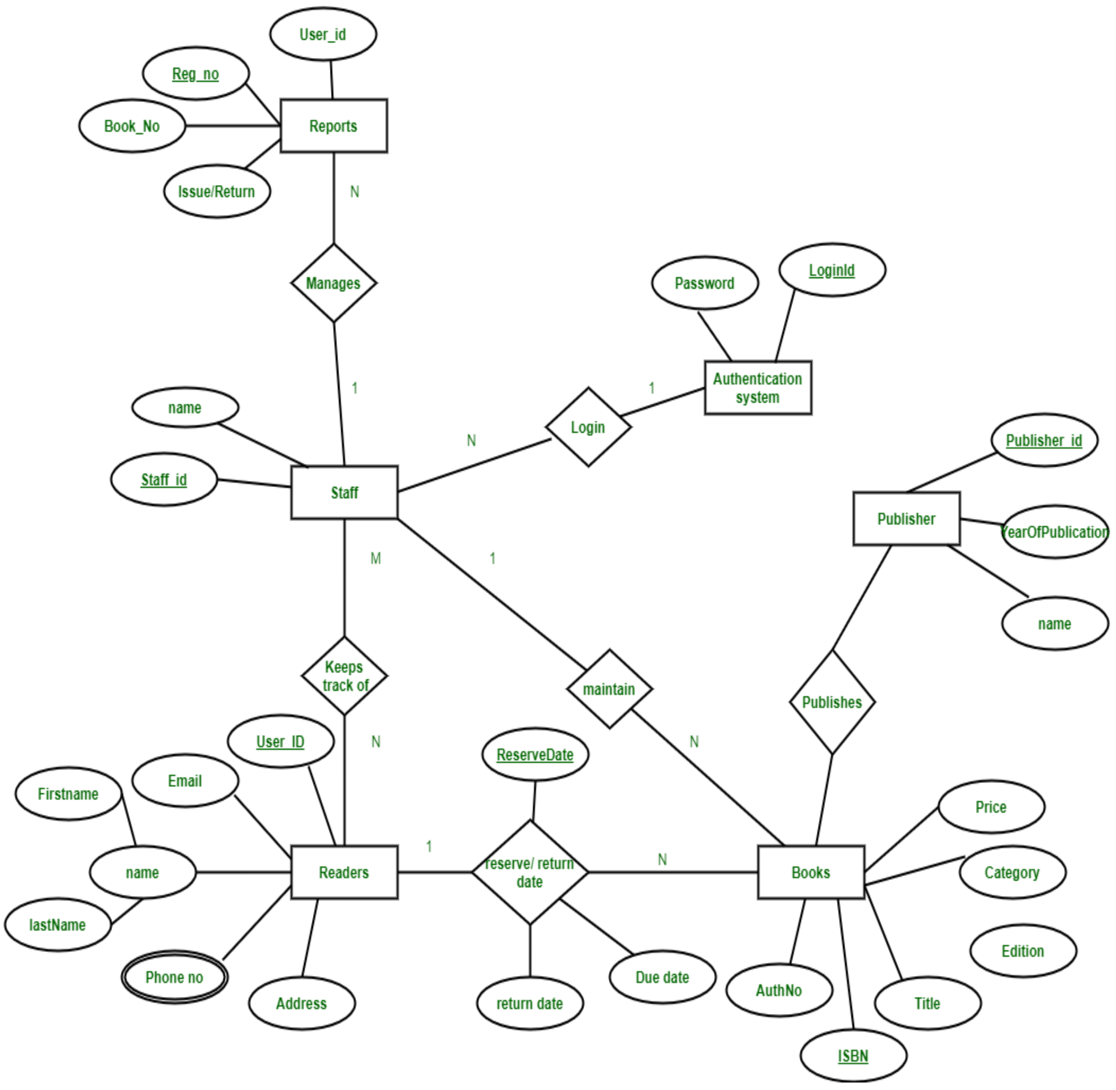
**Scenario: Library Management System**

**Entities:**

**Book**: Contains information about each book available in the library.
**Attributes:** Book_ID (Primary Key), Title, Author, ISBN, Genre
**Member:** Represents the library members who can borrow books.
**Attributes:** Member_ID (Primary Key), Name, Address, Email, Phone
**Borrowing**: Records the borrowing activity, showing which member borrowed which book and when.
**Attributes:** Borrowing_ID (Primary Key), Member_ID (Foreign Key), Book_ID (Foreign Key), Borrow_Date, Return_Date

1. **let's define the relationships:**
   - Book to Borrowing (1-to-Many): A book can be borrowed by many members, but at any given time, it can be borrowed by only one member.
   - Member to Borrowing (1-to-Many): A member can borrow multiple books, but each borrowing record belongs to only one member.

2. **let's ensure normalization up to the third normal form (3NF):**
   - **First Normal Form (1NF):** Each attribute should contain only atomic values. All attributes are atomic in our scenario.
   - **Second Normal Form (2NF)**: No partial dependencies. Each non-key attribute must depend on the entire primary key. We don't have composite primary keys, so this condition is automatically met.
   - **Third Normal Form (3NF):** No transitive dependencies. Non-key attributes should not depend on other non-key attributes. In our scenario, there are no transitive dependencies.



**In this ER diagram:**

The primary keys are underlined.

- The relationships are depicted by lines connecting the entities.
- The cardinality (1-to-Many) is shown using the "1" and "n" notation.

This ER diagram is normalized up to the third normal form and represents the entities, relationships, attributes, and cardinality in the library management system scenario.

## Assignment 3:

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.

The ACID properties of a transaction are like a set of rules that make sure everything goes smoothly when you're working with a database.

1. **Atomicity**: Think of atomicity as a whole package deal. Either everything you wanted to do in your transaction happens successfully, or nothing happens at all. There's no middle ground where some parts work and others don't.

2. **Consistency:** Consistency means that your database stays in a sensible state. For example, if you're transferring money between two accounts, you don't want the total amount of money to change unexpectedly. It's like making sure everything adds up correctly.

3. **Isolation:** Isolation is about keeping transactions separate from each other until they're done. It's like each transaction is in its own bubble, so they don't mess with each other's data. This helps to avoid weird things happening when multiple transactions are running at the same time.

4. **Durability:** Durability ensures that once a transaction is done and confirmed, its changes stick around even if something goes wrong, like a power outage. It's like making sure your changes are saved and won't disappear unexpectedly.

- ✓ **We'll pretend we're moving money between two bank accounts. First, we create a table for our accounts and add some initial balances.**

```
CREATE TABLE accounts (
    id INT PRIMARY KEY,
    balance DECIMAL(10, 2)
);

INSERT INTO accounts (id, balance) VALUES (1,
1000.00), (2, 2000.00);
```

- ✓ **Then, we'll do the money transfer using SQL transactions:**

```
BEGIN TRANSACTION;

UPDATE accounts
SET balance = balance - 100
WHERE id = 1;

UPDATE accounts
SET balance = balance + 100
WHERE id = 2;

COMMIT;
```

- ✓ **Now, let's show how different isolation levels work:**

1. **Read Uncommitted:** This level lets you peek at changes made by other transactions even if they haven't finished yet. It's like looking at someone's drawing while they're still working on it.
   ```
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
   ```

2. **Read Committed:** Here, you can only see changes made by other transactions after they've finished. It's like waiting for someone to finish their drawing before you look at it.
   ```
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
   ```

3. **Repeatable Read:** With this level, once you've seen some data, you'll keep seeing it the same way until your transaction is done, even if others change it. It's like taking a snapshot of the drawing and keeping it the same no matter what happens to the original.
   ```
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
   ```

4. **Serializable:** This level is the strictest. It makes sure no one can change the data you're working with until your transaction is finished. It's like putting a big "Do Not Disturb" sign on your drawing until you're done with it.
   ```
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
   ```

These isolation levels help balance keeping data consistent while allowing transactions to happen at the same time. Depending on what your application needs, you'd pick the one that works best.