Assignment :1

Analyse 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

SOL:

Business Scenario:

Imagine a library where books are stored and lent out to library members. Each book has a unique ISBN (International Standard Book Number), title, author, and category. Library members are registered with their unique ID, name, address, and contact information. Members can borrow multiple books, and each book can be borrowed by multiple members. However, a book can only be borrowed by one member at a time, and each member can borrow multiple books at different times.

Entities:

1. Book
2. Author
3. Category
4. Member
5. Borrowing
6. Publisher
7. Staff
8. Authentication
9. Reader

Relationships:

1. Book - Author (One-to-Many): Each book can have one author, but an author can have many books.
2. Book - Category (Many-to-One): Many books can belong to one category, but each book belongs to only one category.
3. Member - Borrowing (One-to-Many): Each member can have multiple borrowings, but each borrowing is associated with only one member.

4. Book - Borrowing (Many-to-Many): Many books can be borrowed by many members, forming a many-to-many relationship through the borrowing entity.
5. Book - Publisher (Many-to-One): Many books can be published by one publisher, but each book is published by only one publisher.
6. Staff - Authentication (One-to-One): Each staff member has one authentication, and each authentication is associated with only one staff member.
7. Book - Reader (Many-to-Many): Many books can be read by many readers, forming a many-to-many relationship through the reading entity.

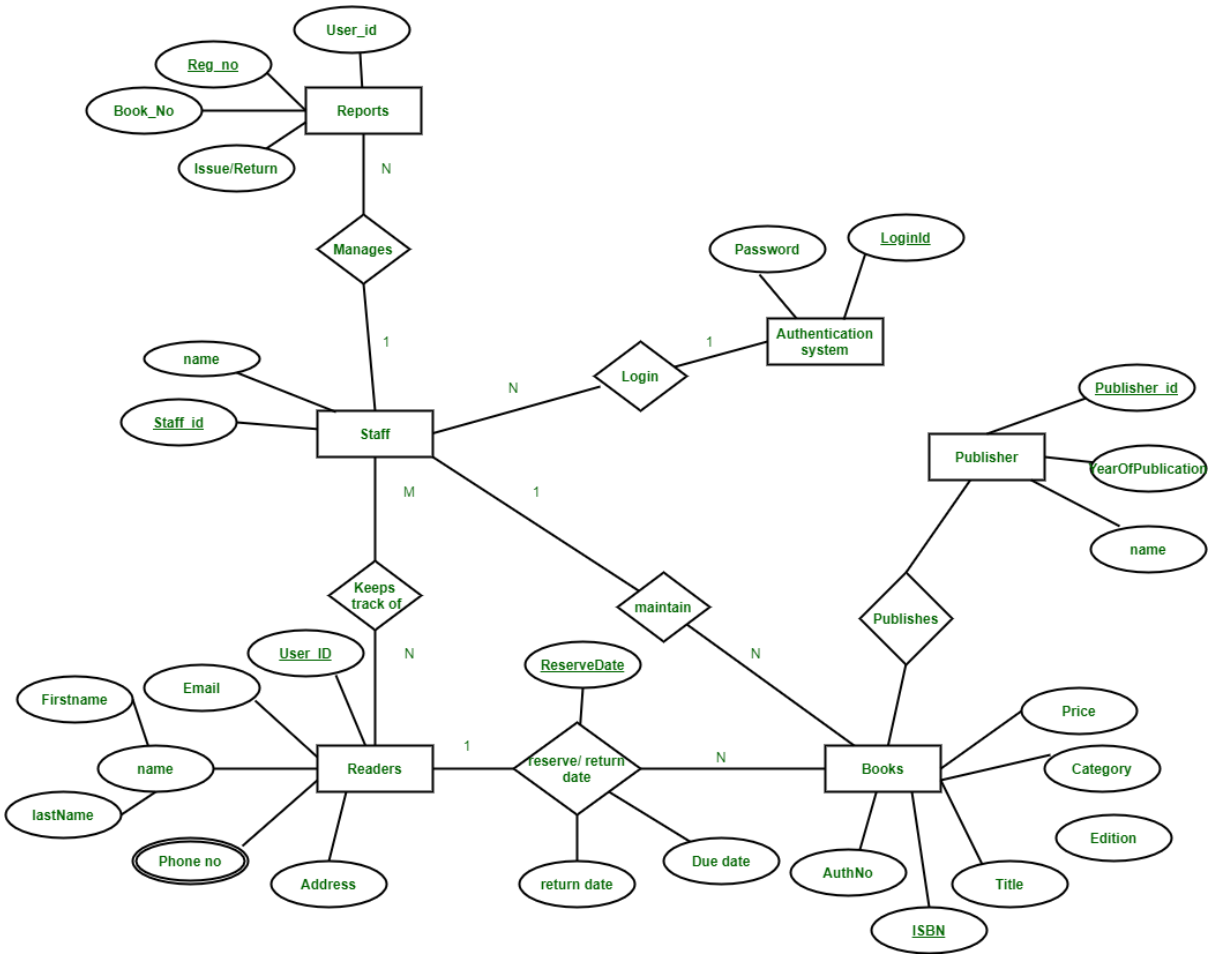
Attributes:

1. Book: ISBN (Primary Key), Title, Author_ID (Foreign Key), Category_ID (Foreign Key)
2. Author: Author_ID (Primary Key), Author_Name
3. Category: Category_ID (Primary Key), Category_Name
4. Member: Member_ID (Primary Key), Name, Address, Contact_Info
5. Borrowing: Borrowing_ID (Primary Key), Member_ID (Foreign Key), ISBN (Foreign Key), Borrow_Date, Return_Date
6. Publisher: Publisher (Primary Key), Publisher Name, Contact_Info
7. Staff: Staff_ID (Primary Key), Name, Role, Contact_Info
8. Authentication: Auth_ID (Primary Key), Username, Password, Last_Login
9. Reader: Reader_ID (Primary Key), Name, Address, Contact_Info

Cardinality:

1. Book - Author: One-to-Many (One author can have many books)
2. Book - Category: Many-to-One (Many books can belong to one category)
3. Member - Borrowing: One-to-Many (One member can have multiple borrowings)
4. Book - Borrowing: Many-to-Many (Many books can be borrowed by many members)
5. Book - Publisher (Many-to-One): Many books can be published by only one publisher, but each publisher can publish many books.

6. Staff - Authentication (One-to-One): Each staff member has one authentication, and each authentication is associated with only one staff member.
7. Book - Reader (Many-to-Many): Many books can be read by many readers, and each reader can read multiple books. Conversely, each book can be read by multiple readers.



ER Diagram of Library Management System

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.

Sol:

Explanation of the ACID properties of a transaction:

1. **Atomicity**: Atomicity ensures that a transaction is treated as a single unit of work, meaning that either all of its operations are completed successfully or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its original state.

2. **Consistency**: Consistency ensures that the database remains in a consistent state before and after the transaction. In other words, transactions should only transition the database from one consistent state to another consistent state, adhering to all constraints, rules, and integrity constraints.

3. **Isolation**: Isolation ensures that the operations within a transaction are executed independently of other transactions. Transactions should be isolated from each other to prevent interference and maintain data integrity. Isolation levels determine the degree to which transactions are isolated from each other.

4. **Durability**: Durability guarantees that once a transaction is committed, its changes are permanently saved and will not be lost, even in the event of a system failure. The changes made by committed transactions should persist in the database regardless of any system crashes or errors.

Now, let's demonstrate a transaction in SQL that includes locking and different isolation levels to show concurrency control.

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

AccountName VARCHAR(50),

Balance DECIMAL(10, 2));

INSERT INTO Accounts (AccountID, AccountName, Balance) VALUES

(1, 'Account A', 1000.00),

(2, 'Account B', 2000.00);

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT Balance FROM Accounts WHERE AccountID = 1;

WAITFOR DELAY '00:00:10';

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

COMMIT;

This SQL transaction simulates a scenario where we're updating the balance of an account (`AccountID = 1`) while another transaction may be reading the balance of the same account with different isolation levels.

- We begin the transaction.
- We set the isolation level to READ COMMITTED, which means that within the same transaction, we only see committed data from other transactions.
- We select the balance of `AccountID = 1`.
- We introduce a delay using `WAITFOR DELAY` to simulate concurrent access.
- During this delay, another transaction can update the balance of `AccountID = 1`.
- We update the balance by subtracting 100.
- Finally, we commit the transaction, making the changes permanent.

You can repeat this transaction with different isolation levels (READ COMMITTED, REPEATABLE READ, SERIALIZABLE) to observe how they affect concurrency and the behaviour of transactions accessing the same data concurrently.