Day 5

**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.**

Let's represent the university database scenario:

Entities:

1. Student

- Student_ ID (Primary Key)
- Name
- Email
- Phone_ Number
- Date_ of_ Birth

2. Course

- Course_ ID (Primary Key)
- Course_ Name
- Credit_ Hours
- Department

3. Instructor

- Instructor_ ID (Primary Key)
- Name
- Email
- Phone_ Number
- Office_ Location

Relationships:

1. Enrollment

- Student_ ID (Foreign Key)
- Course_ ID (Foreign Key)

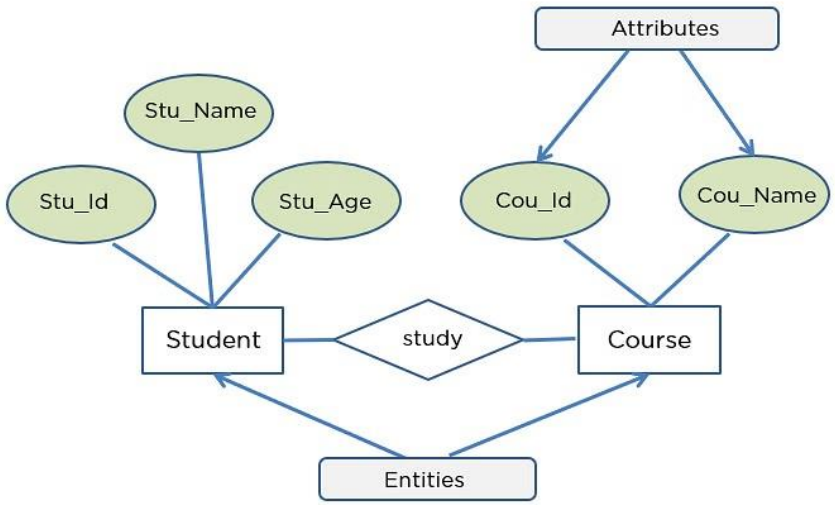
- Grade

2. Teaches

- Instructor_ ID (Foreign Key)

- Course_ ID (Foreign Key)

ER Diagram:




Cardinality:

- One Student can enroll in Many Courses (One-to-Many)

- One Course can have Many Students enrolled (One-to-Many)

- One Instructor can teach Many Courses (One-to-Many)

- Many Courses can have One Instructor (One-to-Many)

This representation maintains the same relationships and attributes but arranges them in a slightly different format for clarity.

**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**.

A) The ACID properties of a transaction ensure reliability and consistency in database operations:

Atomicity: This means that a transaction is an indivisible unit of work. It either completes successfully, committing all its changes, or fails, rolling back any changes made.

Consistency: This property ensures that the database remains in a consistent state before and after the transaction. In other words, all constraints, triggers, and relationships are maintained.

Isolation: Isolation ensures that the concurrent execution of transactions does not result in data inconsistency. Each transaction should appear to execute in isolation from others, regardless of their concurrent execution.

Durability: Once a transaction is committed, its changes are permanently saved in the database, even in the event of system failure.

---

CREATE TABLE Accounts ( Account ID INT PRIMARY KEY, Account Name VARCHAR(50), Balance DECIMAL(10, 2));
INSERT INTO Accounts (Account ID, Account Name, 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 Account ID = 1;

WAITFOR DELAY '00:00:10';

UPDATE Accounts SET Balance Balance 100 WHERE Account ID = 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 Account ID = 1.

We introduce a delay using WAITFOR DELAY to simulate concurrent access.

During this delay, another transaction can update the balance of Account ID = 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 behavior of transactions accessing the same data concurrently.