# CIA - 3 (Component a)

## Indian Education System

**Domain Description**: The **Indian Education System** domain is modeled to represent the structure of schools and their operations, involving students, teachers, courses, and enrollments. The system helps manage student enrollments, teacher assignments to courses, and academic performance. The database includes tables for students, teachers, and courses, along with an enrollment table that records which students are enrolled in which courses, and their grades. The relationships between these entities form the core of the education management system, which can be used for both administrative and academic purposes.

### ER DIAGRAM

An Entity-Relationship (ER) Diagram is a graphical representation of the structure of a database. It illustrates the relationships between different entities in a system and defines how data is stored, organized, and associated within the database.

Key Components of an ER Diagram:

1. **Entities:**

   - Entities represent real-world objects or concepts, such as "Students," "Teachers," "Courses," etc., in the Indian Education System. These entities are typically the tables in a database.

- Each entity has attributes (columns) that provide details about it, like student_id, student_name, and age for the Students entity.

2. **Attributes:**

   - Attributes are the properties or characteristics of an entity. For example, the Students entity may have attributes like student_name, age, class, etc.

   - Primary Key (PK): A unique identifier for an entity (e.g., student_id for the Students table).

   - Foreign Key (FK): An attribute in one table that refers to the Primary Key in another table, establishing a relationship (e.g., teacher_id in the Courses table refers to the Teachers table).

3. **Relationships:**

   - Relationships represent the associations between different entities. For instance, the relationship between Students and Courses is shown via an Enrollment entity, which links students to the courses they enroll in.

   - Relationships are depicted using lines connecting entities, with specific notations to describe the type of relationship, such as one-to-one, one-to-many, or many-to-many.

4. **Cardinality:**

   - Cardinality defines how many instances of one entity relate to instances of another entity. For example, a single Teacher can teach multiple Courses (one-to-many relationship), while each Course is taught by only one Teacher (one-to-one relationship).

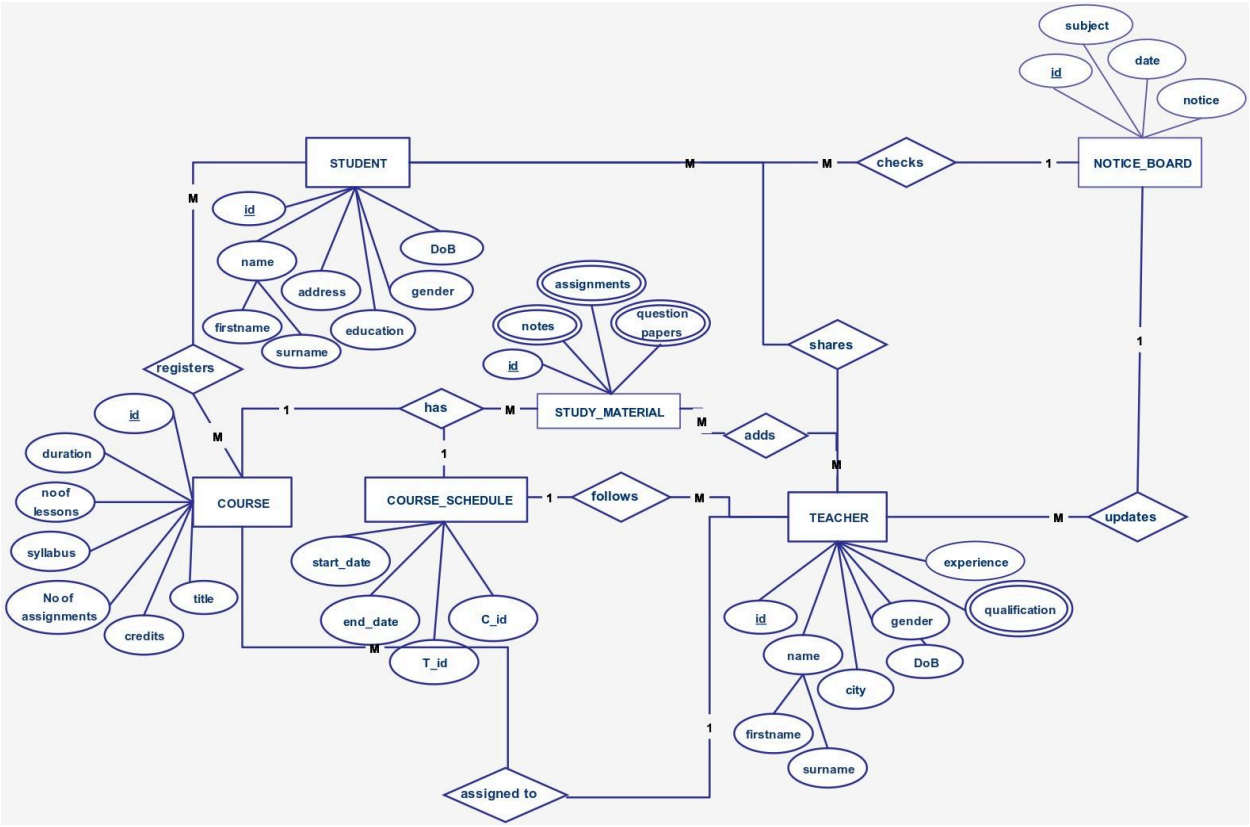
- Notations like “1”, “0..*”, or “N” indicate the type of relationship at each end.

Purpose of an ER Diagram:

- Database Design: ER diagrams help in designing the logical structure of a database. They provide a visual representation of how data is interconnected, which makes it easier to create and maintain databases.

- Data Modeling: ER diagrams are a crucial part of the data modeling process. They help define how entities relate to each other and ensure that data is normalized (organized efficiently).

- Communication Tool: ER diagrams are used by database administrators, developers, and stakeholders to understand how data will be stored and managed in a system.



**1. Data Definition Language (DDL)**

DDL commands define the structure of the database objects such as tables and schemas. Common DDL commands include CREATE, ALTER, DROP, and TRUNCATE.

- **CREATE**: Used to create tables in the database.

```
CREATE TABLE Students (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  student_name VARCHAR(100) NOT NULL,
  age INT CHECK (age >= 5 AND age <= 20),
  class INT NOT NULL
);
```

- **ALTER**: Modifies the structure of an existing table.

```
ALTER TABLE Students ADD address VARCHAR(255);
```

- **DROP**: Deletes a table from the database.

```
DROP TABLE Courses;
```

- **TRUNCATE**: Deletes all records from a table without removing the table itself.

```
TRUNCATE TABLE Teachers;
```

**2. Data Manipulation Language (DML)**

DML commands manipulate the data in the database tables. Common DML commands are INSERT, UPDATE, DELETE, and SELECT.

- **INSERT**: Adds new records to a table.

```
INSERT INTO Students (student_name, age, class)
```

```
VALUES ('Rahul Sharma', 15, 10);
```

- **UPDATE**: Modifies existing records in a table.

```
UPDATE Students SET age = 16 WHERE student_name = 'Rahul Sharma';
```

- **DELETE**: Removes records from a table.

```
DELETE FROM Students WHERE student_id = 1;
```

- **SELECT**: Retrieves data from one or more tables.

```
SELECT * FROM Students WHERE class = 10;
```

### 3. Transaction Control Language (TCL)

TCL commands are used to manage transactions within the database. Common TCL commands include COMMIT, ROLLBACK, and SAVEPOINT.

- **START TRANSACTION**: Marks the beginning of a transaction.

```
START TRANSACTION;
```

- **COMMIT**: Commits the current transaction, making all changes permanent.

```
INSERT INTO Students (student_name, age, class)
VALUES ('Priya Singh', 14, 9);
COMMIT;
```

- **ROLLBACK**: Reverts the current transaction, undoing all changes made since the last commit.

```
DELETE FROM Students WHERE student_id = 2;
ROLLBACK;
```

- **SAVEPOINT**: Sets a point within a transaction to which changes can be rolled back.

```
SAVEPOINT sp1;
```

**4. View Definition Language (VDL)**

VDL commands are used to create and manipulate views. Views are virtual tables created by querying existing tables.

- **CREATE VIEW**: Defines a new view.

```
CREATE VIEW TopStudents AS
SELECT student_name, age FROM Students WHERE age > 15;
```

- **UPDATE VIEW**: Updates data through a view, which also updates the underlying table.

```
UPDATE TopStudents SET age = 17 WHERE student_name = 'Priya Singh';
```

- **DROP VIEW**: Removes a view.

```
DROP VIEW TopStudents;
```

**5. Joins**

Joins are used to retrieve data from multiple tables based on relationships between them.

- **INNER JOIN**: Returns records that have matching values in both tables.

```
SELECT Students.student_name, Courses.course_name
FROM Students
INNER JOIN Enrollment ON Students.student_id = Enrollment.student_id
INNER JOIN Courses ON Enrollment.course_id = Courses.course_id;
```

- **LEFT JOIN**: Returns all records from the left table and matching records from the right table. If there is no match, NULL is returned from the right table.

```
SELECT Students.student_name, Courses.course_name
FROM Students
LEFT JOIN Enrollment ON Students.student_id = Enrollment.student_id;
```

- **RIGHT JOIN**: Returns all records from the right table and matching records from the left table. If there is no match, NULL is returned from the left table.

```
SELECT Courses.course_name, Students.student_name
FROM Courses
RIGHT JOIN Enrollment ON Courses.course_id = Enrollment.course_id;
```

- **FULL OUTER JOIN**: Returns all records when there is a match in either table. Non-matching rows are returned as NULL.

```
SELECT Students.student_name, Courses.course_name
FROM Students
FULL OUTER JOIN Enrollment ON Students.student_id = Enrollment.student_id;
```

**EXECUTION:**

**Create the database**:

```
CREATE DATABASE Indian_Education_System;
USE Indian_Education_System;
```

**Create the** Students **table**:

```
CREATE TABLE Students (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  student_name VARCHAR(100) NOT NULL,
  class INT NOT NULL,
  age INT CHECK (age BETWEEN 5 AND 20),
  address VARCHAR(255),
  gender CHAR(1) CHECK (gender IN ('M', 'F')),
  UNIQUE (student_name, class)
);
```

**Create the** Teachers **table**:

```
CREATE TABLE Teachers (
  teacher_id INT AUTO_INCREMENT PRIMARY KEY,
  teacher_name VARCHAR(100) NOT NULL,
  department VARCHAR(100) NOT NULL,
  contact VARCHAR(15) UNIQUE
);
```

**Create the** Courses **table**:

```
CREATE TABLE Courses (
  course_id INT AUTO_INCREMENT PRIMARY KEY,
  course_name VARCHAR(100) NOT NULL,
  credits INT CHECK (credits > 0),
  teacher_id INT,
```

```
  FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id)
);
```

**Create the** Enrollment **table**:

```
CREATE TABLE Enrollment (
  enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
  student_id INT,
  course_id INT,
  grades CHAR(2),
  FOREIGN KEY (student_id) REFERENCES Students(student_id),
  FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

**Truncate the Courses table**:

```
TRUNCATE TABLE Courses;
```

**Drop the Enrollment table**:

```
DROP TABLE Enrollment;
```

**Rename Child Table**
**Create a child table:**

```
CREATE TABLE Child_Table1 (
  child_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  student_id INT,
  FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

**Rename the child table to a more relevant name:**

```
ALTER TABLE Child_Table1 RENAME TO Attendance;
```

**Insert Data into Tables**

**Insert records into the Students and Teachers tables:**

```
INSERT INTO Students (student_name, class, age, address, gender)
VALUES ('Rahul Sharma', 10, 15, 'New Delhi', 'M'),
      ('Priya Singh', 9, 14, 'Mumbai', 'F'),
      ('Aman Verma', 8, 13, 'Chennai', 'M');

INSERT INTO Teachers (teacher_name, department, contact)
VALUES ('Dr. Anita Rao', 'Mathematics', '9876543210'),
      ('Mr. Rajesh Gupta', 'Science', '9988776655');
```

**Perform Updates**

**Update the child table's ID column using a text column**:

```
UPDATE Attendance SET student_id = (SELECT student_id FROM
Students WHERE student_name = 'Rahul Sharma') WHERE child_id =
1;
```

**Update the entity name using the primary key**:

```
UPDATE Students SET student_name = 'Rohan Sharma' WHERE
student_id = 1;
```

**Add Integrity Constraints**

**Add referential integrity to the child table**:

```
ALTER TABLE Attendance ADD CONSTRAINT fk_student
FOREIGN KEY (student_id) REFERENCES Students(student_id);
```

**Create a new table with constraints**:

```
CREATE TABLE Child_Table2 (
  child_id INT AUTO_INCREMENT PRIMARY KEY,
  teacher_id INT NOT NULL,
  FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id)
);
```

**Rename the second child table**:

```
ALTER TABLE Child_Table2 RENAME TO Teacher_Attendance;
```

**Implement SQL Commands and Clauses**

**Where Clause**:

```
SELECT * FROM Students WHERE class = 10;
```

**Group By Clause**:

```
SELECT class, COUNT(*) FROM Students GROUP BY class;
```

**Having Clause**:

```
SELECT class, COUNT(*) FROM Students GROUP BY class
HAVING COUNT(*) > 1;
```

**Order By Clause**:

```
SELECT * FROM Students ORDER BY age DESC;
```

**Distinct and Limit**:

```
SELECT DISTINCT class FROM Students LIMIT 3;
```

**Aggregate Functions**:
SELECT AVG(age) FROM Students;

**Pattern Matching with LIKE;**
SELECT * FROM Students WHERE student_name LIKE 'R%';

**Implement Joins**

**LEFT, RIGHT, INNER Joins with 2 tables**:

SELECT s.student_name, t.teacher_name FROM Students s

LEFT JOIN Teachers t ON s.class = t.teacher_id;

**FULL OUTER Join with 2 tables**:

SELECT * FROM Students s FULL OUTER JOIN Teachers t ON s.class = t.teacher_id;

**Subqueries and Set Operations**
**Subquery with Self-query**:

SELECT student_name FROM Students WHERE student_id = (SELECT MAX(student_id) FROM Students);

**Set Operations**:

SELECT student_name FROM Students WHERE class = 10

UNION

SELECT student_name FROM Students WHERE gender = 'F';

**Implement TCL Commands**
**Start a Transaction and Commit**:

START TRANSACTION;

INSERT INTO Students (student_name, class, age, address, gender)
VALUES ('Anjali', 9, 14, 'Bangalore', 'F');

COMMIT;

**Rollback**:

START TRANSACTION;

DELETE FROM Students WHERE student_id = 2;

ROLLBACK;

**Create Views**

CREATE VIEW TopStudents AS SELECT student_name, age FROM Students WHERE age > 15;

**Update and Delete from View**:

UPDATE TopStudents SET age = 16 WHERE student_name = 'Anjali';

DELETE FROM Students WHERE age < 12;