Experiment 1

Name: Sunil Bhagat
Roll no:20
Batch:C12

Q.1 What is Entity?

Ans: An **entity** is a real-world object, concept, or thing that can be distinctly identified in a database. Entities are represented as **tables** in relational databases, and they contain attributes that describe their properties.

Key Features of an Entity:

- **Uniquely identifiable:** Each entity should have an attribute (or a set of attributes) that differentiates it from other entities.
- **Has attributes:** Entities have characteristics that define them.
- **Exists in the database:** Entities are stored as records (rows) in a table.

Example:

Consider a university database:

- A **Student** is an entity with attributes:
  - **Student_ID** (Unique identifier)
  - **Name**
  - **Age**
  - **Course**
- A **Course** is another entity with attributes:
  - **Course_ID**
  - **Course_Name**
  - **Credits**

Entities are the building blocks of a database system because they define what data will be stored.

Q.2 What are different types of entity?

Ans: Entities in DBMS are classified into different types based on their behavior and dependency:

1. Strong Entity

- An entity that exists independently and has a primary key to uniquely identify its records.
- It does not depend on any other entity.

Example:

In a school database, the Student entity is a strong entity because it has a primary key (Student_ID) and does not depend on another entity.

Student (Student_ID, Name, Age, Course)

2. Weak Entity

- A weak entity cannot exist without a strong entity.
- It does not have a primary key but instead uses a foreign key from a related strong entity.
- It must have a relationship with a strong entity and needs a discriminator attribute to identify instances uniquely.

Example:
A Dependent entity that stores information about dependents of students:
Here, Student_ID is a foreign key linking the dependent to a specific student.

Dependent (Dependent_ID, Dependent_Name, Student_ID)

3. Super Entity and Sub Entity

- Super Entity: A higher-level entity that generalizes multiple lower-level entities.
- Sub Entity: A lower-level entity that inherits attributes from a super entity.

Example:
A Vehicle entity can be a Super Entity for Car and Bike sub-entities:

Vehicle (Vehicle_ID, Vehicle_Type)
Car (Vehicle_ID, Number_of_Doors)
Bike (Vehicle_ID, Has_Gear)

Q.3 Different types of Relationship ?

Ans: A **relationship** is an association between two or more entities in a database. It defines how the entities interact with each other.

Example of Relationships:

- **Student** enrolls in a **Course** → Relationship: **Enrolls**
- **Doctor** treats a **Patient** → Relationship: **Treats**
- **Employee** works for a **Company** → Relationship: **Works_for**

Types of Relationships in DBMS:

1. One-to-One (1:1) Relationship

- One instance of Entity A is related to only one instance of Entity B.

Example:
Each **employee** has **one** unique **company ID card**, and each ID card belongs to only **one** employee.

Employee (Emp_ID, Name, ID_Card_No)
ID_Card (ID_Card_No, Issue_Date)

2. One-to-Many (1:M) Relationship

- One instance of Entity A can be associated with multiple instances of Entity B, but each instance of B is related to only one instance of A.

Example:
One **teacher** teaches multiple **students**, but each **student** has only one teacher.

Teacher (Teacher_ID, Name)
Student (Student_ID, Name, Teacher_ID)

3. Many-to-Many (M:N) Relationship

- Multiple instances of Entity A are associated with multiple instances of Entity B.

Example:
A student can enroll in multiple courses, and a course can have multiple students.

Student (Student_ID, Name)
Course (Course_ID, Course_Name)
Enrollment (Student_ID, Course_ID)

Q.4 Explain cardinality in DBMS ?

Ans: **Cardinality** in DBMS defines the number of instances of one entity that can or must be associated with an instance of another entity. It determines the **minimum** and **maximum** number of relationships an entity can have in a database.
Cardinality plays a crucial role in database design, helping in defining **constraints** and relationships between tables.

Types of Cardinality:
1. One-to-One (1:1)

- **Definition**: In this type of relationship, **one instance** of Entity A is related to **one and only one** instance of Entity B, and vice versa.
- **Implication**:
  - This relationship is **not very common** in databases.

- Often, the two entities could be **merged into a single entity** if they share most attributes.

Example: Country and President
Each **country** has **one president**, and each **president** governs **only one country**.

Country (Country_ID, Country_Name, President_ID)
President (President_ID, Name, Age, Term)

Here, President_ID is a **foreign key** in the Country table, ensuring that each country has only one president.

2. One-to-Many (1:M)

- **Definition**: One instance of Entity A can be associated with **multiple instances** of Entity B, but each instance of Entity B is related to **only one** instance of Entity A.
- **Implication**:
  - The **primary key** of Entity A is **referenced as a foreign key** in Entity B.
  - This is one of the most common relationships in databases.

Example 1: Teacher and Students
One **teacher** can teach **many students**, but each **student** is taught by only one **teacher**.

Teacher (Teacher_ID, Name, Subject)
Student (Student_ID, Name, Teacher_ID)

Here, Teacher_ID in the Student table is a **foreign key** referring to the primary key Teacher_ID in the Teacher table.

3. Many-to-Many (M:N)

- **Definition**: One instance of Entity A can be associated with **multiple instances** of Entity B, and vice versa.
- **Implication**:
  - This relationship **requires a junction (bridge) table** to resolve the many-to-many connection.
  - The bridge table contains **foreign keys** from both entities.

Example 1: Students and Courses
A **student** can enroll in **multiple courses**, and each **course** can have **multiple students**.

Student (Student_ID, Name)
Course (Course_ID, Course_Name)
Enrollment (Student_ID, Course_ID)

Here, Enrollment is a **junction table** that stores the relationships. It contains:

- Student_ID (Foreign key referencing Student)
- Course_ID (Foreign key referencing Course)

In an **ER diagram**, cardinality is represented as:

- (1,1) → One-to-One
- (1,N) → One-to-Many
- (M,N) → Many-to-Many

Q.5 Explain Extended ER features

Ans: The **Extended Entity-Relationship (EER) Model** expands the ER model by introducing advanced features such as **Generalization, Specialization, Inheritance, and Aggregation**.

1. Generalization

- It is the process of combining **two or more lower-level entities** into a **higher-level entity**.
- The higher-level entity **inherits** the attributes of the lower-level entities.
- **Example:**
    - Car and Bike are generalized into **Vehicle**.

Vehicle (Vehicle_ID, Brand)
Car (Vehicle_ID, Number_of_Doors)
Bike (Vehicle_ID, Has_Gear)

2. Specialization

- It is the opposite of generalization.
- A higher-level entity is divided into multiple specialized lower-level entities.
- **Example:**
    - Employee is specialized into Teacher and Administrator.

Employee (Emp_ID, Name, Salary)
Teacher (Emp_ID, Subject)
Administrator (Emp_ID, Department)

3. Inheritance

- Inheritance allows sub-entities to acquire attributes and relationships of a super-entity**.**
- **Example:**
    - Teacher and Admin inherit common attributes from Employee.

4. Aggregation

- Aggregation is a higher-level abstraction that treats relationships as entities.
- It is used when a relationship itself has attributes.
- **Example:**
    - A Project is assigned to multiple Employees, and a Manager oversees multiple projects.

Project (Project_ID, Name)
Employee (Emp_ID, Name)
Manager (Manager_ID, Name)
Works_On (Project_ID, Emp_ID, Manager_ID, Hours_Worked)

- Aggregation helps in representing complex relationships efficiently.