03 Entity-Attribute-Relation

1. Entity

- An **Entity** is an object , concept, or event about which data is stored or thing that can be distinctly identified
- Each entity has attributes that describe its properties
- In a relational database, entities are usually mapped to tables, with each record (row) representing an instance of the entity

Example

- An entity called `Student` might represent students in a school
- Each student is a distinct instance of the `Student` entity

Types of Entities

Entities can be categorized into different types based on their relationships and how they are used in the database model

1. **Strong (or Regular) Entity**
2. **Weak Entity**
3. **Associative Entity**

01 Strong (or Regular) Entity

- A **Strong Entity** is an entity that can exist independently of other entities
- It has a primary key that uniquely identifies each instance of the entity
- Here, the `Student` entity is a strong entity because it can exist independently and is uniquely identified by `StudentID`

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    DateOfBirth DATE
);
```

02 Weak Entity

- A **Weak Entity** is an entity that cannot exist without being related to another entity
- It does not have a primary key of its own and instead relies on a "strong" entity for identification
- A weak entity is always associated with a strong entity through a foreign key
- The `CourseEnrollment` entity might depend on the existence of a `Student` and a `Course` entity. It is a weak entity because its existence depends on both the `Student` and `Course` entities

```sql
CREATE TABLE CourseEnrollment (
    EnrollmentID INT,
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (EnrollmentID, StudentID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

03 Associative Entity

- An **Associative Entity** (also called a junction or linking entity) is used to associate two or more entities that have a many-to-many relationship
- This entity itself can be considered a weak entity but specifically exists to link other entities together
- The `StudentCourse` entity is used to link students and courses in a many-to-many relationship. Each record in this entity represents an enrollment of a student in a course

```sql
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

2. Attributes

- An **Attribute** is a property or characteristic of an entity
- Attributes provide additional details about the entity and are represented as columns in a table

Types of Attributes

- Attributes can be categorized into several types based on their nature and role in the entity

1. **Simple Attribute**
2. **Composite Attribute**
3. **Single-Valued Attribute**
4. **Multi-Valued Attribute**
5. **Derived Attribute**
6. **Key Attribute**

01 Simple Attribute

- A **Simple Attribute** is an attribute that cannot be divided further
- It represents a single piece of information
- In the `Student` entity, attributes like `StudentID` and `Name` are simple attributes

```sql
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255)
);
```

02 Composite Attribute

- A **Composite Attribute** is an attribute that can be divided into smaller sub-parts, which represent more basic attributes with independent meanings
- An `Address` attribute might be composed of `Street`, `City`, `State`, and `ZipCode`.

Composite Attribute Structure

- Address (Street, City, State, ZipCode)

```sql
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    Street VARCHAR(255),
    City VARCHAR(255),
    State VARCHAR(255),
    ZipCode VARCHAR(10)
);
```

03 Single-Valued Attribute

- A **Single-Valued Attribute** is an attribute that holds a single value for each instance of the entity
- In the `Student` entity, `DateOfBirth` is a single-valued attribute because each student has only one date of birth

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255),
    DateOfBirth DATE
);
```

04 Multi-Valued Attribute

- A **Multi-Valued Attribute** is an attribute that can hold multiple values for each instance of the entity
- A `PhoneNumbers` attribute for the `Student` entity might allow storing multiple phone numbers
- To implement this in a relational database, you might create a separate table to store the multiple values

```
CREATE TABLE StudentPhone (
    StudentID INT,
    PhoneNumber VARCHAR(15),
    PRIMARY KEY (StudentID, PhoneNumber),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

05 Derived Attribute

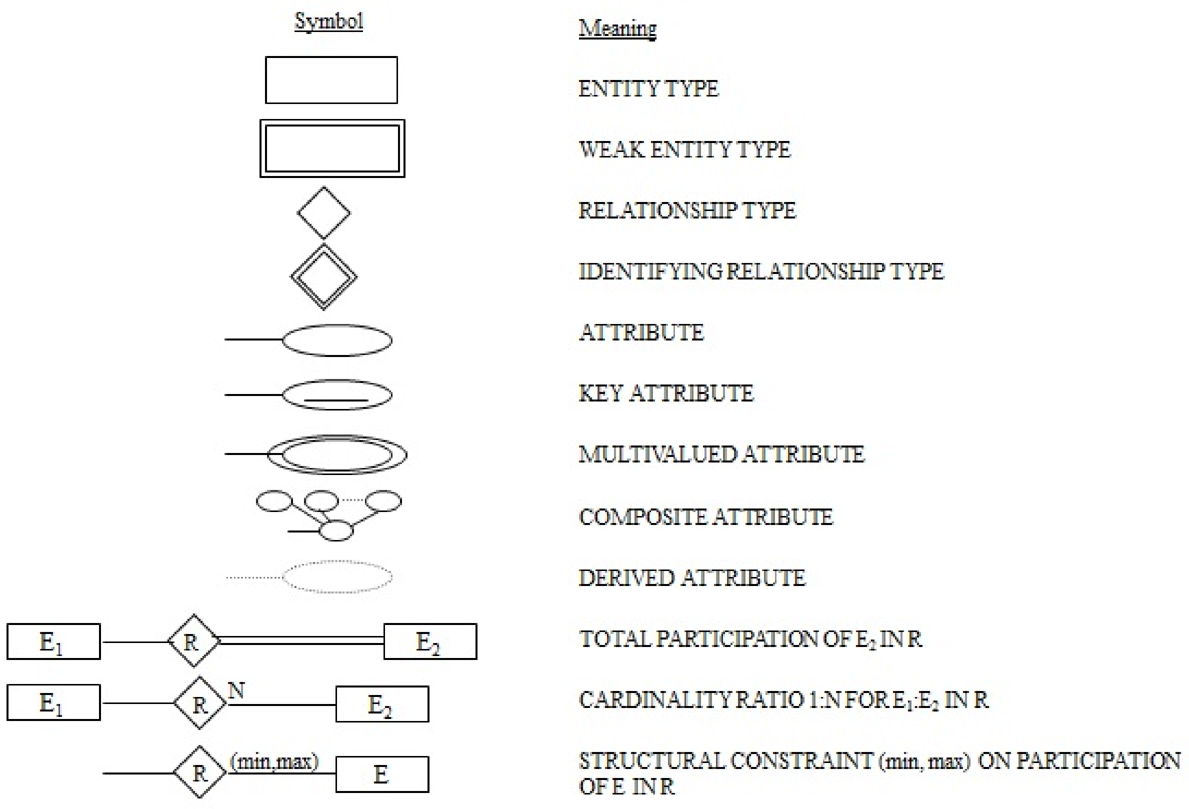
- A **Derived Attribute** is an attribute whose value is derived from other attributes
- It is not stored directly in the database but can be computed when needed
- An `Age` attribute can be derived from the `DateOfBirth` attribute

```
-- Pseudo code for a derived attribute
SELECT Name, (CURRENT_DATE - DateOfBirth) AS Age
FROM Student;
```

06 Key Attribute

- A **Key Attribute** is an attribute that is used to uniquely identify an instance of the entity
- The most common key attribute is the **Primary Key**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(255)
);
```


Symbol
Meaning
ENTITY TYPE
WEAK ENTITY TYPE
RELATIONSHIP TYPE
IDENTIFYING RELATIONSHIP TYPE
ATTRIBUTE
KEY ATTRIBUTE
MULTIVALUED ATTRIBUTE
COMPOSITE ATTRIBUTE
DERIVED ATTRIBUTE
E1
R
E2
TOTAL PARTICIPATION OF E2 IN R
N
CARDINALITY RATIO 1:N FOR E1:E2 IN R
(min,max)
E
STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

3. Relationships

- In the context of databases, **relationships** describe how different entities (usually represented as tables) are connected to one another
- Understanding these relationships is fundamental to designing an efficient and normalized relational database
- Relationships in a database model define how data in one table is associated with data in another

1. **One-to-One (1:1) Relationship**
2. **One-to-Many (1:M) Relationship**
3. **Many-to-Many (M:M) Relationship**
4. **Self-Referencing Relationships**

01 One-to-One (1:1)

- In a **One-to-One** relationship, a single record in one table is associated with a single record in another table
- Each entity in both tables can only be associated with one entity in the other table

Example

- **Users Table**: Contains information about users
- **UserProfiles Table** : Contains additional profile details for each user
- Each user in the `Users` table has exactly one corresponding profile in the `UserProfiles` table, and vice versa
- This type of relationship is often implemented by placing the foreign key in either of the two tables

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(255)
);

CREATE TABLE UserProfiles (
    ProfileID INT PRIMARY KEY,
    UserID INT,
    Address VARCHAR(255),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

02 One-to-Many (1:M)

- In a **One-to-Many** relationship, a single record in one table can be associated with multiple records in another table, but each record in the second table is associated with only one record in the first table

Example

- **Customers Table** : Contains customer details
- **Orders Table** : Contains order details
- A single customer can place multiple orders, but each order is placed by only one customer
- The `Orders` table will have a foreign key referencing the primary key in the `Customers` table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
```

```sql
    Name VARCHAR(255)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

03 Many-to-Many (M:M)

- In a **Many-to-Many** relationship, multiple records in one table can be associated with multiple records in another table
- This relationship is usually implemented using a junction table (also known as a join table or associative entity) that contains foreign keys referencing the primary keys of both related tables

Example

- **Students Table** : Contains student details
- **Courses Table** : Contains course details
- **Enrollments Table** : A junction table that records which students are enrolled in which courses
- A student can enroll in multiple courses, and a course can have multiple students
- The `Enrollments` table holds the foreign keys from both `Students` and `Courses`

```sql
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    StudentName VARCHAR(255)
);

CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(255)
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

04 Self-Referencing

- A **Self-Referencing Relationship** occurs when a table is related to itself
- This is typically used to model hierarchical data

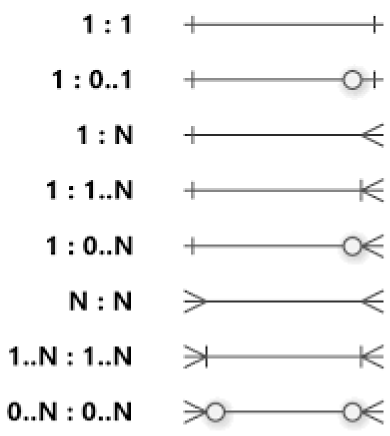
Example

- **Employees Table** : Contains employee details
- Each employee might have a manager, who is also an employee
- An employee can have a manager, and that manager is also an employee within the same `Employees` table
- The `Employees` table would have a `ManagerID` column that is a foreign key referencing the `EmployeeID` in the same table

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(255),
    ManagerID INT,
    FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)
);
```

Crow's Foot Notation

- **Crow's Foot Notation** is a popular way to represent entity-relationship diagrams (ERDs) in database design
- It visually describes how tables (entities) are related to one another in a relational database
- The notation uses symbols to indicate the type of relationship (one-to-one, one-to-many, or many-to-many) between entities


1 : 1
1 : 0..1
1 : N
1 : 1..N
1 : 0..N
N : N
1..N : 1..N
0..N : 0..N