

## Unit II : Relational Model

### Part 1 : Entity-Relationship Modeling

#### **Introduction:**

**Entity-Relationship (ER) model**, is a popular high-level conceptual data model. This model is frequently used for the conceptual design of database applications.

(Another type of Object modeling can be done using **Unified Modeling Language (UML)** which are becoming increasingly popular in both database and software design. )

### Entity types, Entity Sets, Attributes and Keys

**Entity:** The basic object that the ER model represents is an **entity**. An entity is a thing in the real world with an independent existence. (Item about which we store information).

An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for example, a match, a game, a company, a job, or a university course).

In the E-R diagram, each Entity Type is represented by a **Rectangle**.

**Attribute:** Each entity has **attributes**. The properties that describe Entity are called Attributes. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.

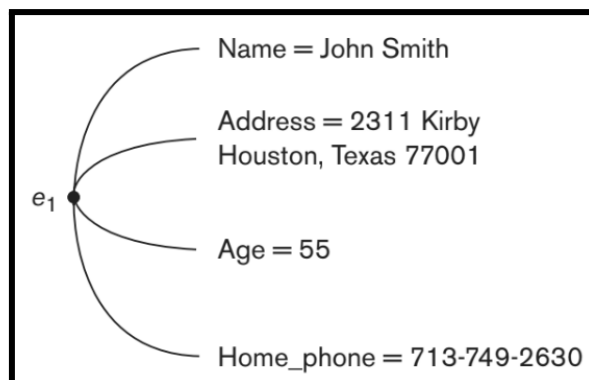


Figure shows an entity and the values of its attributes. The EMPLOYEE entity e1 has four attributes: Name, Address, Age, and Home\_phone. Their values are 'John Smith,' '2311 Kirby, Houston, Texas 77001', '55', and '713-749-2630', respectively.

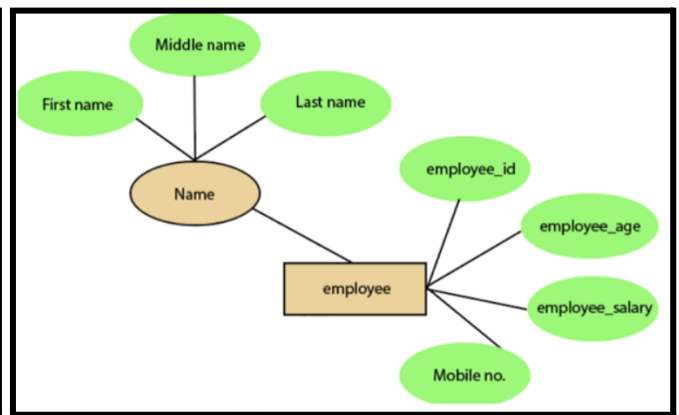
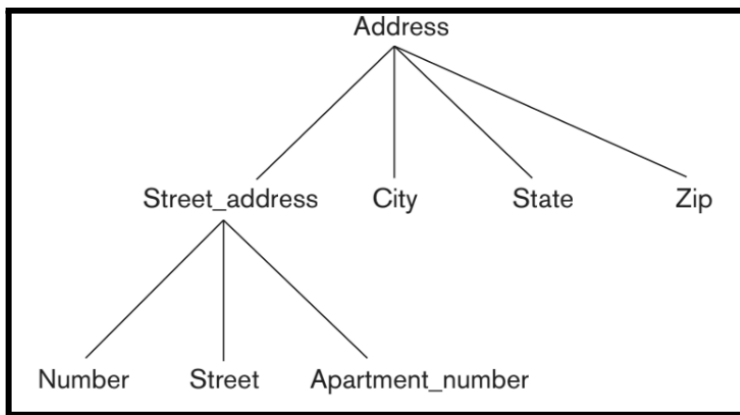
In the E-R diagram, each Attribute is represented by an Oval.

Several types of attributes in the ER model are:

- Simple Attribute versus Composite Attribute
- Single Valued Attribute versus Multivalued Attribute
- Stored Attribute versus Derived Attribute

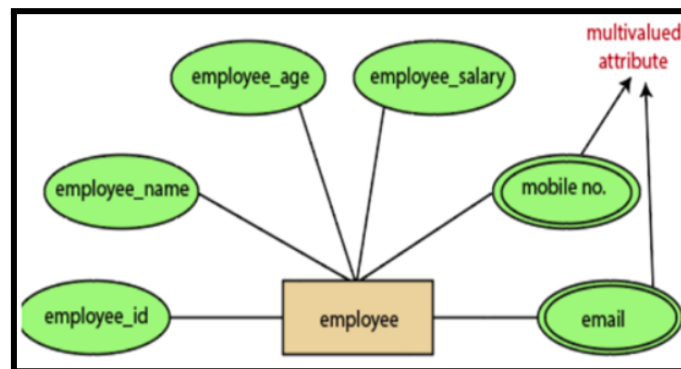
**Simple Attribute versus Composite Attribute :** **Simple attribute** consists of a single atomic value. They are not divided into subparts. For example the attributes age, sex etc are simple attributes.

**Composite attribute** can be divided into smaller subparts. For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street\_address, City, State, and Zip. Composite attributes can form a hierarchy. The Street\_address can be further subdivided into: Number, Street, and Apartment\_number. Composite attributes are represented by ellipses that are connected with an ellipse.

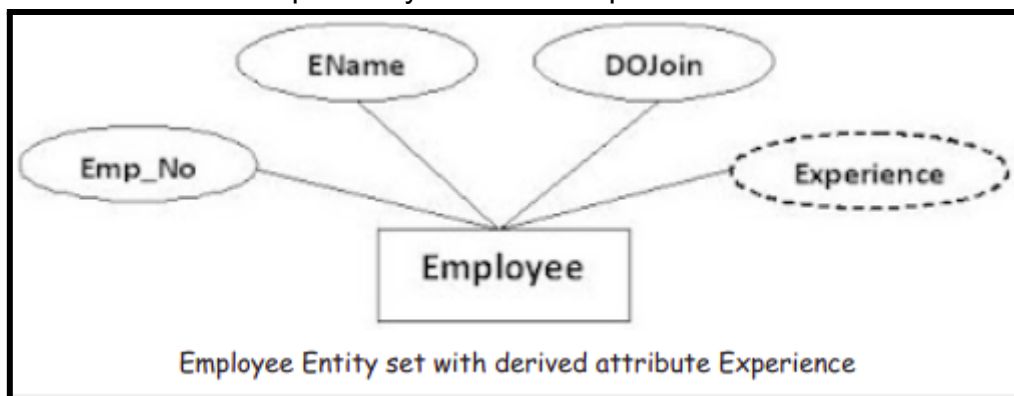


**Single-Valued versus Multivalued Attributes:** Most attributes have a single value for a particular entity; such attributes are called **single-valued**. For example, Age is a single-valued attribute of a person.

In some cases an attribute can have a set of values for the same entity. For example, a Colors attribute for a car, or a College\_degrees attribute for a person. Such attributes are called **multivalued**. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values. Multivalued attributes are depicted by double ellipse.



**Stored versus Derived Attributes:** In some cases, two (or more) attribute values are related. For example, the Age and Birth\_date attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth\_date. The Age attribute is called a **derived attribute** and Birth\_date is called a stored attribute. Derived attributes are depicted by a dashed ellipse.



**NULL Values:** In some cases, a particular entity may not have an applicable value for an attribute. For example, a College\_degrees attribute applies only to people with college degrees. For such situations, a special value called NULL is created. A person with no college degree would have NULL for College\_degrees. NULL can also be used if we do not know the value of an attribute for a particular entity.

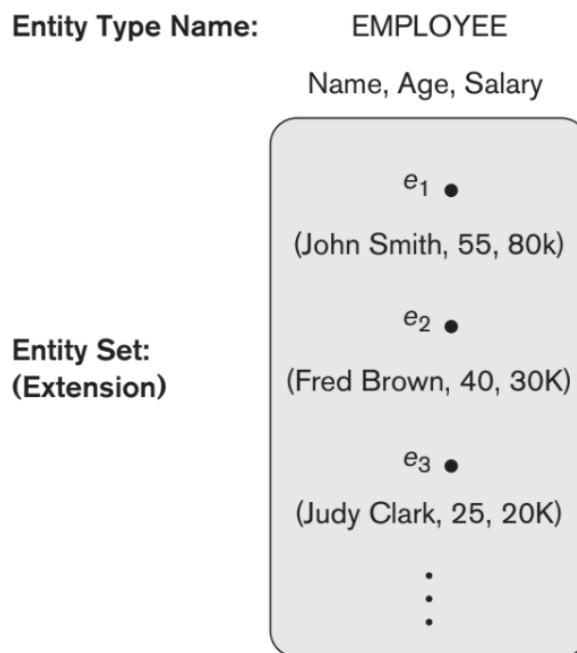
**Complex Attributes:** A complex attribute is both composite and multi valued. Composite and multivalued attributes can be nested arbitrarily. We can group components of a composite attribute between parentheses () and separate the components with commas, and display the multivalued attributes between braces { }. Such attributes are called **complex attributes**. For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address\_phone for a person can be specified as shown below:

```
{Address_phone( {Phone(Area_code,Phone_number)},Address(Street_address  
(Number,Street,Apartment_number),City,State,Zip) )}
```

### Entity Types, Entity Sets:

An entity type defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attributes. The collection of all entities of a particular entity type in the database at any point in time is called an entity set. An entity type describes the schema or intention for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the extension of the entity type.

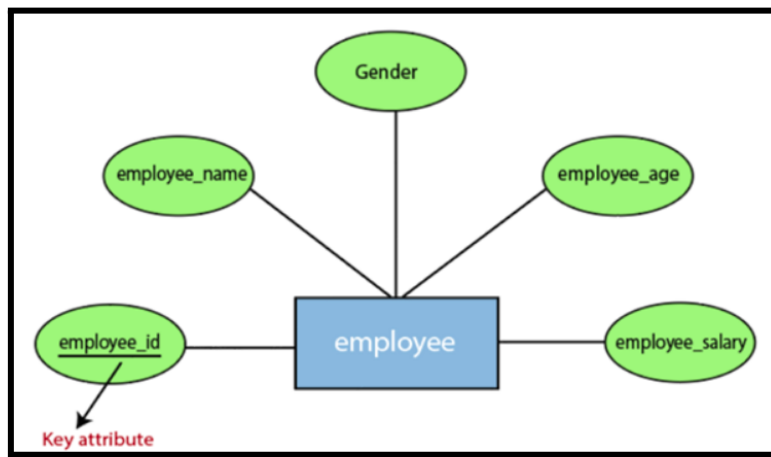
Figure shows the entity type: EMPLOYEE and a list of some of its attributes:



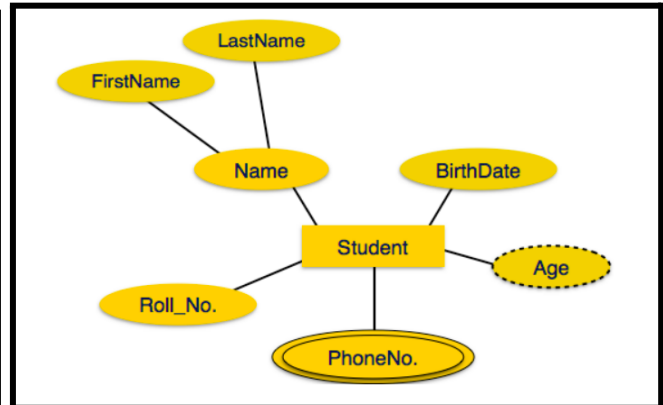
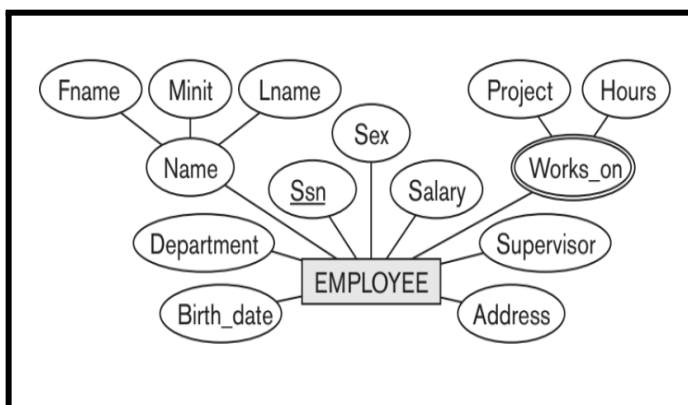
**Key:** Usually Key attribute has values that are distinct for each individual entity in the entity set. Its values can be used to identify each entity uniquely. For example, the Employee\_Id attribute is a key of the EMPLOYEE entity. No two employees are allowed to have the same Employee\_Id. Key is also called uniqueness constraint on attributes.

In E-R diagram, each key attribute has its name **underlined** inside the oval.

Sometimes several attributes together form a key called **composite key**. A composite key must be minimal.



Example 1: An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth\_date, Department, and Supervisor. Both Name and Address may be composite attributes.



Example 2: Student Entity

**Value Sets (Domains) of Attributes:-** Each simple attribute of an entity type is associated with a **value set** or **domain** of values. Domain specifies the set of values that may be assigned to that attribute for each individual entity. (Set of all possible values of an attribute is called Domain)

Eg:- If the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70. Value sets are not displayed in ER diagrams

### Relationship Types, Relationship Sets, Roles, and Structural Constraints

#### **Relationship:**

A relationship describes an association among entities.

#### **Relationship Type:**

A Relationship Type is a type of association that can exist between two entity types.

#### **Relationship set:**

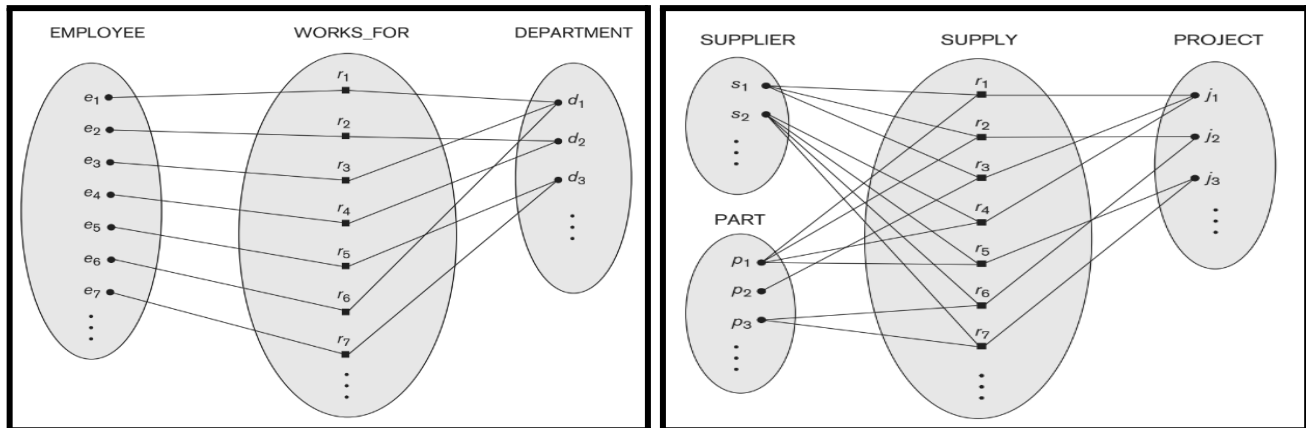
A relationship set is a set of associations of the same entity types.

A **relationship type**  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations or a relationship set among entities from these entity types. A relationship type and its corresponding relationship set are referred to by the same name,  $R$ . Mathematically, the relationship set  $R$  is a set of **relationship instances**  $r_i$ , where each  $r_i$  associates  $n$  individual entities  $(e_1, e_2, \dots, e_n)$ , and each entity  $e_j$  in  $r_i$  is a member of entity set  $E_j$ ,

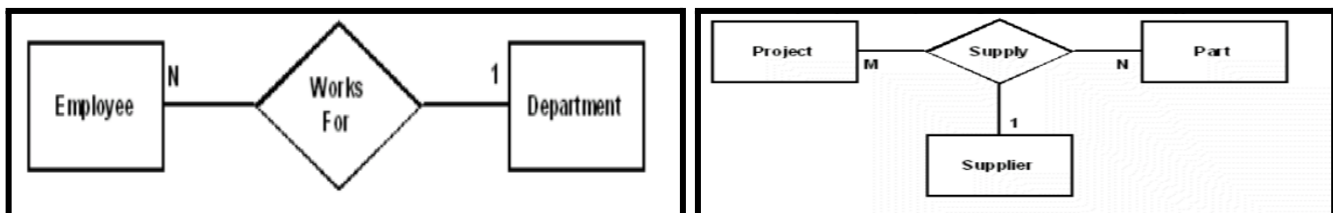
Hence, a relationship set is a mathematical relation on  $E_1, E_2, \dots, E_n$ . It can be defined as a

subset of the Cartesian product of the entity sets  $E_1 \times E_2 \times \dots \times E_n$ . Each of the entity types  $E_1, E_2, \dots, E_n$  is said to participate in the relationship type  $R$ ; similarly, each of the individual entities  $e_1, e_2, \dots, e_n$  is said to **participate** in the relationship instance  $r_i = (e_1, e_2, \dots, e_n)$ .

The **degree** of a relationship type is the number of participating entity types. Hence, the WORKS\_FOR relationship between EMPLOYEE and DEPARTMENT is of degree two. A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.



In ER Diagram, Relationships are represented by a diamond symbol connected to the related entities. The relationship name (an active or passive verb), is written inside the diamond.



### Role Names and Recursive Relationships.

Each entity type that participates in a relationship type plays a particular role in the relationship. The role name indicates the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS\_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

In some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships**.

### Constraints on Binary Relationship Types:

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. For example if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. There are two main types of binary relationship constraints:

1. Cardinality ratio
2. Participation.

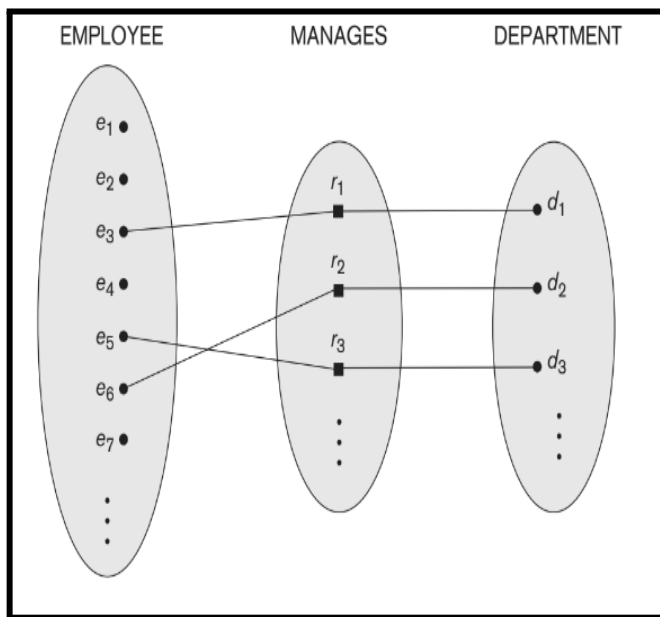
## 1. Cardinality Ratios for Binary Relationships:

The **cardinality ratio** for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

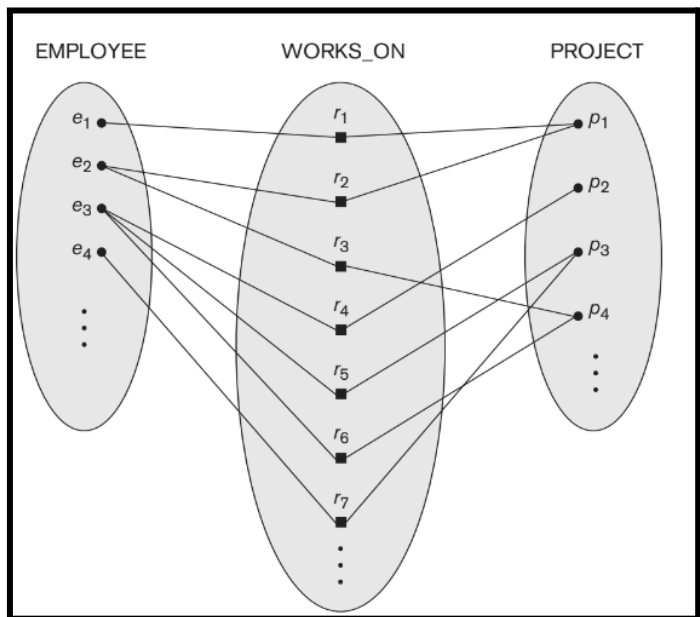
Eg:-In the WORKS\_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number of employees. But an employee can be related to (work for) only one department. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

That is,

- One to One
- One to Many
- Many to One
- Many to Many.



One to One

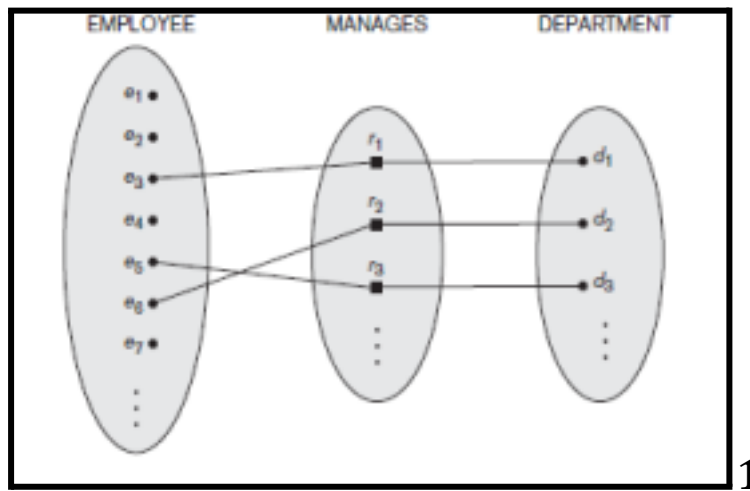


Many to Many

## 2. Participation Constraints and Existence Dependencies:

The **participation constraint** specifies whether the existence of an entity depends on a relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the **minimum cardinality constraint**.

There are two types of participation constraints—total and partial. If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS\_FOR relationship instance. Thus, the participation of EMPLOYEE in WORKS\_FOR is called **total participation**. It means that every entity in the total set of employee entities must be related to a department entity via WORKS\_FOR. Total participation is also called **existence dependency**.



1

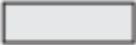







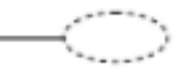


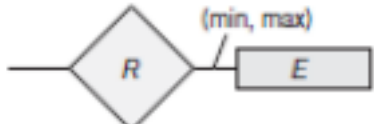
In manages relationship, we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial**. It means that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all. The cardinality ratio and participation constraints, taken together is called **structural constraints** of a relationship type.

### Weak Entity Types

Entity types that do not have key attributes of their own are called weak entity types. In contrast, regular entity types that do have a key attribute are called strong entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. This other entity type is called the identifying or owner entity type. A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.

A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.

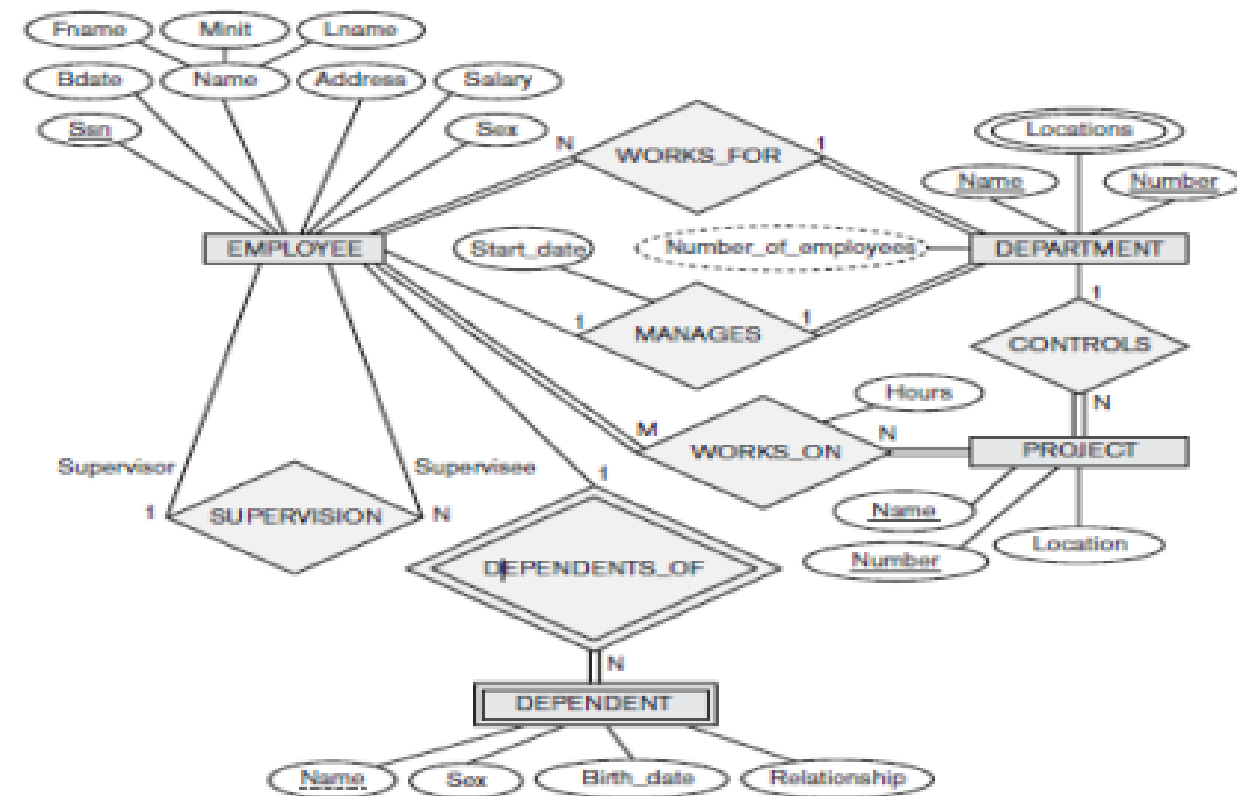
**Notation for ER Diagrams:** ER Diagrams are used to represent the ER model Design.

Symbol	Meaning	Summary of the notation for ER diagrams.
	Entity	
	Weak Entity	
	Relationship	
	Identifying Relationship	
	Attribute	
	Key Attribute	
	Multivalued Attribute	
	Composite Attribute	
	Derived Attribute	
	Total Participation of $E_2$ in $R$	
	Cardinality Ratio 1: N for $E_1:E_2$ in $R$	
	Structural Constraint (min, max) on Participation of $E$ in $R$	

Relationships can also have attributes associated to them. These attributes are called **relationship attributes**.



The following figure shows the ER diagram for the COMPANY database.



An ER schema diagram for the COMPANY database.

Entity types EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes in the COMPANY ER-diagram.

Relationship types are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Eg: WORKS\_FOR, MANAGES etc.

Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute.

Multivalued attributes are shown in double ovals. Eg: Locations attribute of DEPARTMENT.

Key attributes have their names underlined. Eg: ssn

Derived attributes are shown in dotted ovals. Eg: Number\_of\_Employees

Hours is a Relationships attribute associated with the relationship WORKS\_ON.

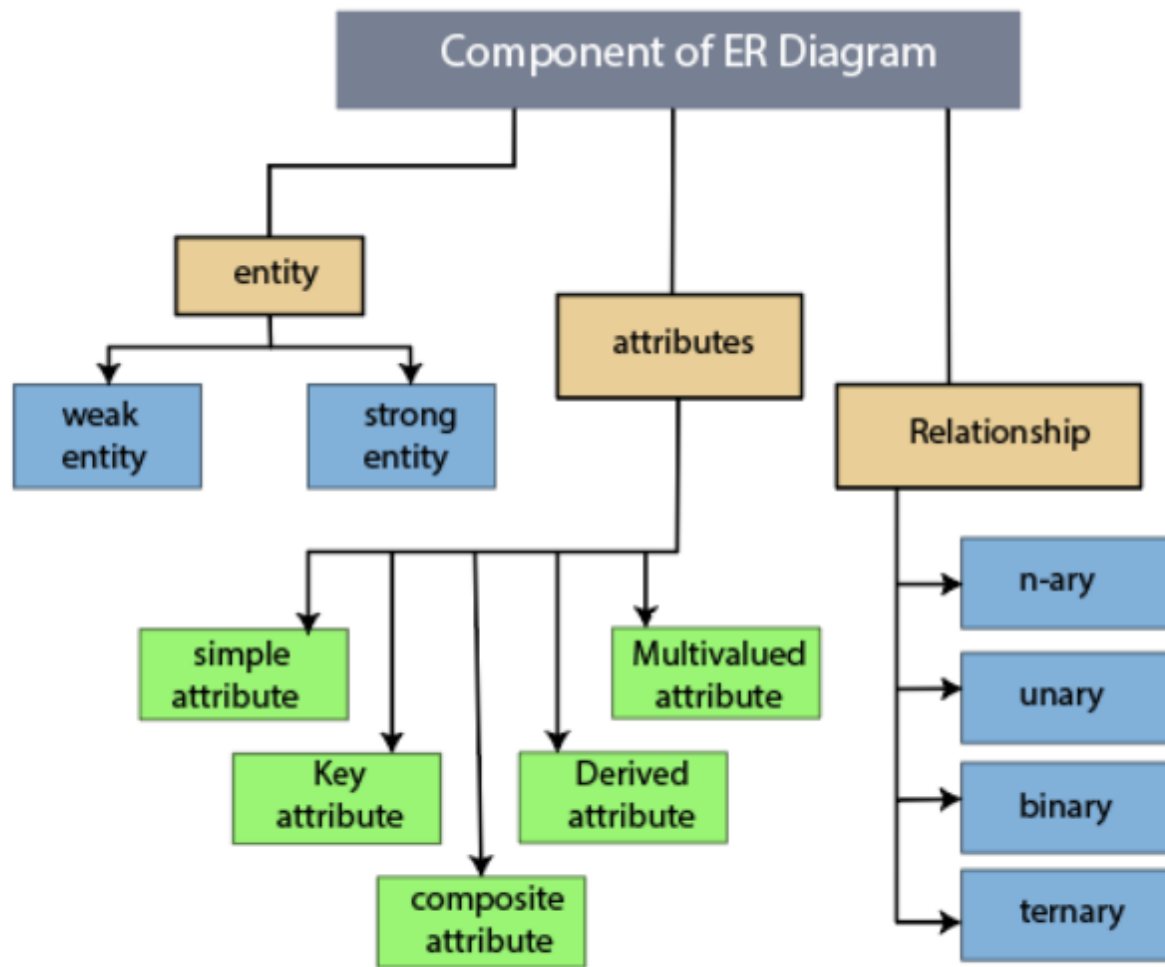
Weak entity types are placed in double rectangles and their identifying relationships are placed in double diamonds.

The partial key of the weak entity type is underlined with a dotted line.

The cardinality ratio of each binary relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas it is 1:N for DEPARTMENT: EMPLOYEE in WORKS\_FOR, and M:N for WORKS\_ON.

The participation constraint is specified by a single line for partial participation and by double lines for total participation (existence dependency).

The SUPERVISION relationship type has role names because the same EMPLOYEE entity type plays two distinct roles in that relationship. The cardinality ratio is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor. But an employee in the role of supervisor can supervise zero or more employees.



### Exercise 1:

(Part C, Mar 2021)

Construct an E-R schema Diagram for Company Database:

Details of COMPANY database is as follows. (1) The company is organised into departments. Each department has a unique name, number and a particular employee who manages the department. Keep track of the start date when that employee began managing the department. A department may have several locations. (2) A department controls a number of projects, each of which has a name, a unique number and a single location. (3) Employee has name, SSN, address, salary, gender and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily be controlled by the same department. Keep track of the current number of hours per week that an employee works on each project. Also keep track of the direct supervisor of each employee. (4) Keep track of the dependants of each employee for instance purposes. Dependant has name, gender, dob and relationship to employee.

**Assignment 1:** Construct an E-R Diagram for University Database

## Part 2 : Relational Model Concepts

### **Relations, Tuples, Attributes and Domains**

**Relation:** The relational model represents the database as a collection of relations. Each relation is a **table** of values.

Eg:- STUDENT table

**Tuple:** A row of a table is called a tuple. Each **row** in the table represents a collection of related data values. A row represents an entity or a relationship.

**Attribute:** A column name is called an attribute. All values in a column are of the same data type. The **table name** and **column names** are used to give the meaning of the values in each row.

Eg:- Name, Date\_of\_Birth, Grade, Grade\_Point\_Average



name	birth	gpa	grad
Anderson	1987-10-22	3.9	2009
Jones	1990-4-16	2.4	2012
Hernandez	1989-8-12	3.1	2011
Chen	1990-2-4	3.2	2011

**Domain:** **Domain** is a set of all possible values of a column. The data type describing the types of values that can appear in each column is represented by a **domain**.

A domain D is a set of atomic values. Atomic means that each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain.

Eg:- Names - The set of character strings that represent names of persons.

Employee\_ages - Possible ages of employees in a company; each must be an integer value between 15 and 80.

A domain is thus given a name, data type, and format. Additional information such as pounds or kilograms also be given.

**Relation schema:** A relation schema is used to describe a relation. A relation schema R, denoted by R(A1, A2, ..., An), is made up of a relation name R and a list of attributes, A1, A2, ..., An. Each attribute Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai).

Example: Relational Schema corresponding to ER Diagram of COMPANY.

Employee (ENo, FName, MName, LName, DOB, Address, Sex, Salary, DNo, SupervisNo)

Department(DNo, DName, Location1, Location2, NumOfEmp)

Project(PNo, PName, Location, DNo)

Dependent(ENo, Name, Sex, DOB, Relationship)

Manager(Eno, DNo, StartDate)

WorksOn(ENo, PNo, Hours)

**Degree (or arity):** The degree (or arity) of a relation is the total number of attributes of its relation schema.

A relation of degree seven, which stores information about university students, would contain seven attributes describing each student.

STUDENT(Name, Ssn, Home\_phone, Address, Office\_phone, Age, Gpa).

Using the data type of each attribute, the definition is sometimes written as:

STUDENT(Name: string, Ssn: string, Home\_phone: string, Address: string,  
Office\_phone: string, Age: integer, Gpa: real)

A **relation** (or relation state) of the relation schema  $R(A_1, A_2, \dots, A_n)$ , is also denoted by  $r(R)$ .

A relation is a set of n-tuples. Each n-tuple,  $t$  is an ordered list of  $n$  values  $\langle v_1, v_2, \dots, v_n \rangle$ . Each value  $v_i$ ,  $1 \leq i \leq n$ , is an element of  $\text{dom}(A_i)$  or a NULL value. NULL values represent attributes whose values are unknown or do not exist for some individual tuple.

The  $i$ th value in tuple  $t$ , which corresponds to the attribute  $A_i$ , is referred to as  $t[A_i]$  or  $t.A_i$ . The terms relation intension for the schema  $R$  and relation extension for a relation state  $r(R)$  are also commonly used.

**Definition:** A relation  $R$  is a subset of the Cartesian product of the domains of  $R$ .

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$$

A relation  $r(R)$  is a mathematical relation of degree  $n$  on the domains  $\text{dom}(A_1)$ ,  $\text{dom}(A_2)$ , ...,  $\text{dom}(A_n)$ .

**Cardinality:** Cardinality is the total number of values in a domain  $D$  and is denoted by  $|D|$ . The total number of tuples in the Cartesian product is

$$|\text{dom}(A_1)| \times |\text{dom}(A_2)| \times \dots \times |\text{dom}(A_n)|$$

A relation state reflects only the valid tuples that represent a particular state of the real world.

It is possible for several attributes to have the same domain. The attribute names indicate different roles for the domain. For example, in the EMPLOYEE relation, the same domain phone\_numbers plays the role of Home\_phone, and the role of Office\_phone of the student.

### Characteristics of Relations

**1. Ordering of Tuples in a Relation is not important:** A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them. So tuples in a relation do not have any particular order. When we display a relation as a table, the rows are displayed in a certain order. Tuple ordering is not part of a relation definition. Many tuple orders can be specified on the same relation. For example, tuples in the STUDENT relation could be ordered by values of Name, Ssn, Age, or some other attribute.

**2. Ordering of Values within a Tuple is important:** An n-tuple is an ordered list of  $n$  values. So the ordering of values in a tuple is important. Hence ordering of attributes in a relation schema is important. But the order of attributes and their values is not that important if the correspondence between attributes and values is maintained. A relation schema  $R = \{A_1, A_2, \dots, A_n\}$  is a set of attributes. A tuple can be considered as a set of ( $\langle \text{attribute} \rangle$ ,  $\langle \text{value} \rangle$ ) pairs. Each pair gives the value  $v_i$  of attribute  $A_i$  from  $\text{dom}(A_i)$ .

Eg: CREATE TABLE Student ( RNo number, FName char(10) , SName char(10))

INSERT INTO Student VALUES (1, 'Ananth', 'B')

INSERT INTO Student(FName, SName, RNo) VALUES ('Akash', 'Krishna', 2)

INSERT INTO Student VALUES (3, 'Albin', NULL);

3. Values in the Tuples should be atomic: Each value in a tuple is an atomic value. Composite and multivalued attributes are not allowed.

NULL values are allowed: There are several meanings for NULL values, such as unknown value, value exists but is not available, or attribute does not apply to this tuple (value undefined or NA)

4. Relational schema gives Interpretation (Meaning) of a Relation: The relation schema can be interpreted as a declaration or a type of assertion(അവകാശവാദം). For example, the schema of the STUDENT relation declares that a student entity has a Name, Ssn, Home\_phone, Address, Office\_phone, Age, and Gpa. Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion.

Notice that some relations may represent facts about entities and some other relations may represent facts about relationships.

### **Relational Model Constraints and Relational Database Schemas**

On modelling relational database design, we can put some restrictions like what values are allowed to be inserted, modified and deleted. These constraints are derived from the rules in the mini-world that the database represents. Constraints are a set of rules used to maintain the quality of information. Constraints are used to guard against accidental damage to the database.

Constraints on databases can be divided into three main categories:

1. **Implicit constraints**: Constraints that are present in the data model. They are called **inherent model based constraints or implicit constraints**.
2. **Explicit constraints**: Constraints that can be directly expressed in schemas of the data model, by specifying them in the DDL. They are called **schema-based constraints or explicit constraints**.
3. **Application-based constraints**: Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. They are called **application-based or semantic constraints or business rules**.

For example, the constraint that a relation cannot have duplicate tuples is an inherent constraint.

The schema-based constraints include

- (i) domain constraints
- (ii) key constraints
- (iii) constraints on NULLs
- (iv) entity integrity constraints
- (v) referential integrity constraints.

#### **(i) Domain Constraints**

A Domain constraint specifies the value of an attribute within each tuple. Attribute A must be an atomic value from the domain of A.

The data types associated with domains include

- numeric data types for Integers (such as Short Integer, Integer, and Long Integer)
- numeric data types for Real numbers (Float and Double Precision Float)
- Booleans
- Characters (fixed-length strings, and variable-length strings)
- Date
- Time
- Timestamp
- Money
- or other special data types.

## **(ii) Key Constraints :**

Keys are used to uniquely identify an entity within an entity set.

**Super Keys :** A Super Key is a set of one or more attributes that are taken collectively and can identify uniquely an entity in an entity set.

Eg: {Ssn, Name, Age}, {Ssn, Name}, {Ssn}, ... are super keys in the relation STUDENT.

Here any set of attributes that includes Ssn is a superkey

**Candidate Keys :** Candidate keys are minimal super keys.

Eg: {Ssn}

Two student tuples can not have the same value for Ssn.

**Primary Key :** It is a candidate key that is chosen by the database designer to identify entities within an entity set uniquely. Although several candidate keys may exist, one of the candidate keys is selected to be the primary key.

**Composite Key :** Ideally a primary key is composed of only a single attribute. But it is possible to have a primary key composed of more than one attribute. Composite key consists of more than one attributes.

No two tuples in a relation can be the same. There are subsets of attributes of a relation also with this property. Suppose that we denote one such subset of attributes by SK. then for any two distinct tuples t1 and t2 in a relation, we have the constraint that:

$$t1[SK] \neq t2[SK]$$

Any such set of attributes SK is called a **superkey** of the relation schema R. Every relation has at least one default superkey. That is the set of all its attributes.

**Key:** A key K of a relation schema R is

- a superkey of R and
- removing any attribute A from K leaves a set of attributes K\_ that is not a superkey of R any more.

Hence, a key satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for all the attributes in the key.
2. It is a minimal superkey.

This first property also applies to a superkey. The second property is not required by a superkey. Hence, a key is also a superkey but not vice versa.

Example: Consider the STUDENT relation. The attribute set {Ssn} is a key of STUDENT.

{Ssn, Name, Age} is a superkey. However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.

In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require all its attributes together to have the uniqueness property.

A relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. For example, the CAR relation has two candidate keys: License\_number and Engine\_serial\_number.

We can choose one of the candidate keys as the **primary key** of the relation. We can make the other candidate keys as **unique keys**, and are not underlined.

### (iii) Constraints on NULL Values:

**NOT NULL constraint:** Another constraint on attributes specifies whether NULL values are permitted

or are not permitted. For example, if every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

### (iv) Entity Integrity :

The **entity integrity constraint** states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations. Key constraints and entity integrity constraints are specified on individual relations.

### (v) Referential Integrity and Foreign Keys :

The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples in the two relations. The referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Eg: CREATE TABLE Department (DNo number PRIMARY KEY, DName char (15))  
CREATE TABLE Student ( RNo number PRIMARY KEY, FName char(10) , SName char(10),  
DNo char(15), FOREIGN KEY DNo REFERENCES Department (DNo))  
INSERT INTO Student VALUES (1, 'Ananth', 'B')  
INSERT INTO Student(FName,SName,RNo) VALUES ('Akash', 'Krishna',2)  
INSERT INTO Student VALUES (3,'Albin',NULL);

For example, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

To define referential integrity we define the concept of a **foreign key**. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R1 and R2. A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference the relation R2.
2. A value of FK in a tuple  $t_1$  of the current state  $r_1(R_1)$  either occurs as a value of PK for some tuple  $t_2$  in the current state  $r_2(R_2)$  or is NULL.

When  $t_1[FK] = t_2[PK]$ , we say that the tuple  $t_1$  references or refers to the tuple  $t_2$ .

In this definition,  $R_1$  is called the referencing relation and  $R_2$  is the referenced relation. If these two conditions hold, a referential integrity constraint from  $R_1$  to  $R_2$  is said to hold. Referential integrity constraints typically arise from the relationships among the entities represented by the relation schemas.

Data dependencies, which include functional dependencies and multivalued dependencies are also important constraints.



## **Relational Databases and Relational Database Schemas**

A relational database usually contains many relations, with tuples in relations that are related in various ways.

A relational database schema  $S$  is a set of relation schemas  $S = \{R_1, R_2, \dots, R_m\}$  and a set of integrity constraints  $IC$ .

A relational database state  $DB$  of  $S$  is a set of relation states  $DB = \{r_1, r_2, \dots, r_m\}$  such that each  $r_i$  is a state of  $R_i$  and such that the  $r_i$  relation states satisfy the integrity constraints specified in  $IC$ . Figure 3.5 shows a relational database schema that we call  $COMPANY = \{\text{EMPLOYEE}, \text{DEPARTMENT}, \text{DEPT\_LOCATIONS}, \text{PROJECT}, \text{WORKS\_ON}, \text{DEPENDENT}\}$ . The underlined attributes represent primary keys

When we refer to a relational database, we include both its schema and its current state. A database state that does not obey all the integrity constraints is called an invalid state, and a state that satisfies all the constraints in the defined set of integrity constraints  $IC$  is called a valid state.

NULL :unknown value

value exists but is not available

attribute does not apply to this tuple

value undefined