Module No.2

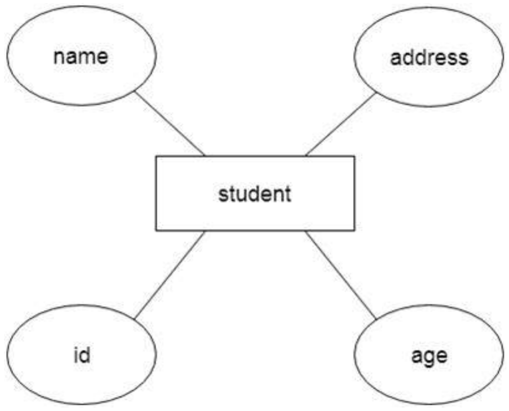
Entity-Relationship Data Model

The Entity-Relationship (ER) Model: Entity types: Weak and strong entity sets, Entity sets, Types of Attributes, Keys, Relationship constraints: Cardinality and Participation, Extended Entity-Relationship (EER) Model: Generalization, Specialization and Aggregation

The Entity-Relationship Model:

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



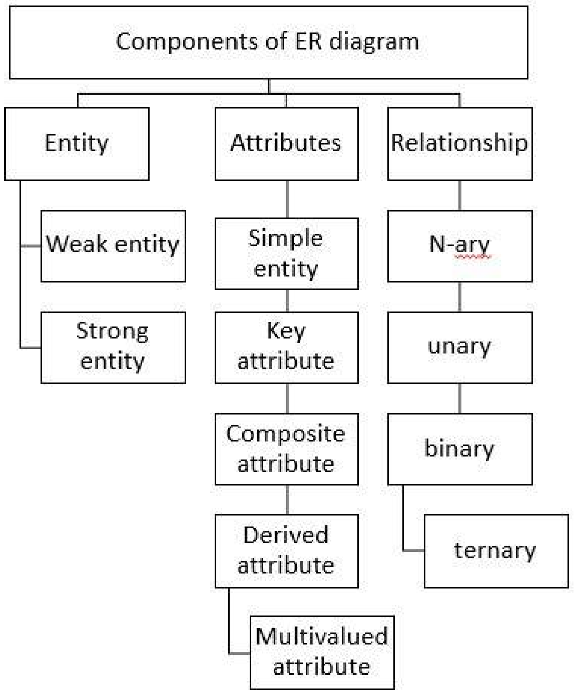
Components of ER diagram

The components of ER diagram are as follows −

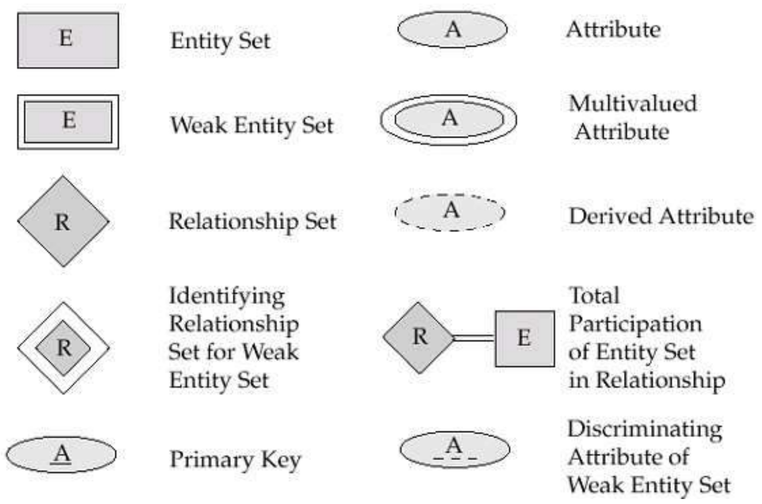
- Entity
- Attributes
- Relationship
- Weak entity
- Strong entity
- Simple attribute
- Key attribute
- Composite attribute

- Derived attribute
- Multivalued attribute

The components of the ER diagram are pictorially represented as follows −



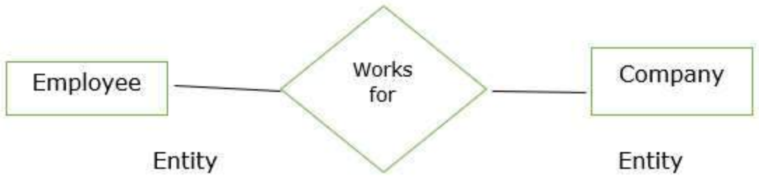
Notations used in ER model:


E
Entity Set
A
Attribute
E
Weak Entity Set
A
Multivalued Attribute
R
Relationship Set
A
Derived Attribute
R
Identifying Relationship Set for Weak Entity Set
R
E
Total Participation of Entity Set in Relationship
A
Primary Key
A
Discriminating Attribute of Weak Entity Set

Entity

It may be an object, person, place or event that stores data in a database. In a relationship diagram an entity is represented in rectangle form. For example, students, employees, managers, etc.

The entity is pictorially depicted as follows −

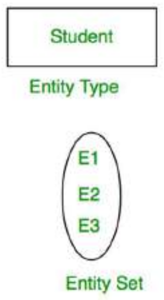


Entity set

It is a collection of entities of the same type which share similar properties. For example, a group of students in a college and students are an entity set.

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

An Entity is an object of Entity Type and a set of all entities is called as an entity set. e.g.; E1 is an entity having Entity Type Student and set of all students is called Entity Set. In ER diagram, Entity Type is represented as:



Entity is characterised into two types as follows −

- Strong entity set
- Weak entity set

Strong entity set

The entity types which consist of key attributes or if there are enough attributes for forming a primary key attribute are called a strong entity set. It is represented by a single rectangle.

For Example,

Roll no of student

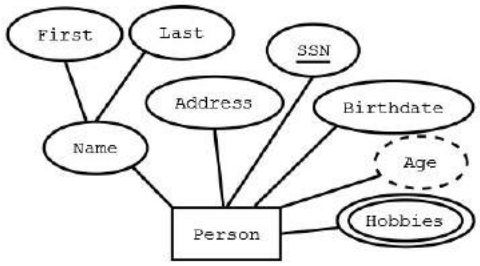
EmpID of employee

Weak entity set

An entity does not have a primary key attribute and depends on another strong entity via foreign key attribute. It is represented by a double rectangle.

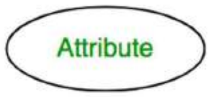
Attributes

It is the name, thing etc. These are the data characteristics of entities or data elements and data fields.



Attributes:

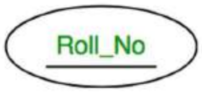
Attributes are the properties that define the entity type. For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.



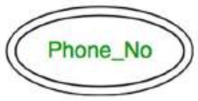
Types of attributes

The types of attributes in the Entity Relationship (ER) model are as follows –

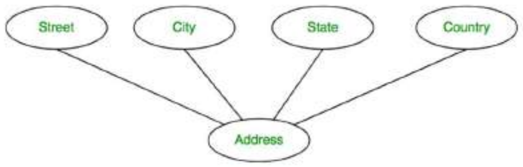
- **Key Attribute –** The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll_No will be unique for each student. In ER diagram, key attribute is represented by an oval with underlying lines.

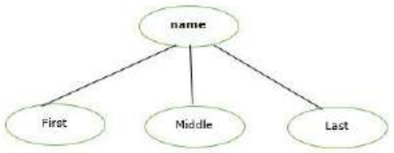


- **Single value attribute** − These attributes contain a single value. For example, age, salary etc.
- **Multivalued attribute** − They contain more than one value of a single entity. An attribute consisting more than one value for a given entity.
  - For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval. For example, phone numbers.

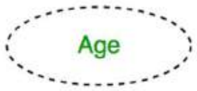


  -
- **Composite attribute** − The attributes which can be further divided. An attribute composed of many other attribute is called as composite attribute.
  - For example, Address attribute of student Entity type consists of Street, City, State, and Country. In ER diagram, composite attribute is represented by an oval comprising of ovals.
  - Name-> First name, Middle name, last name

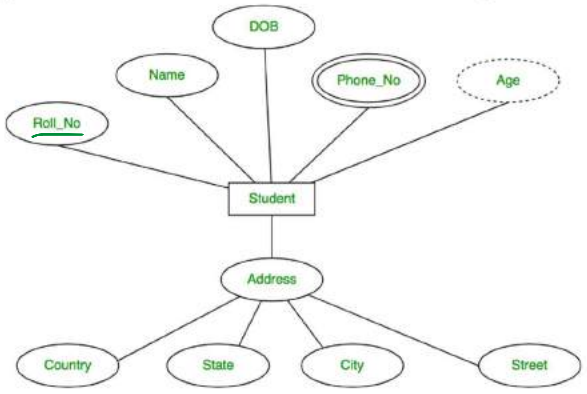




- **Derived attribute** − The attribute that can be derived from others. An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.



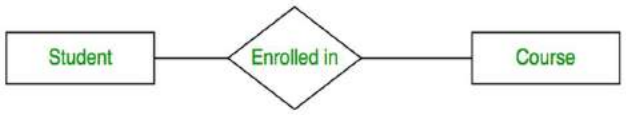
The complete entity type **Student** with its attributes can be represented as:



Relationship

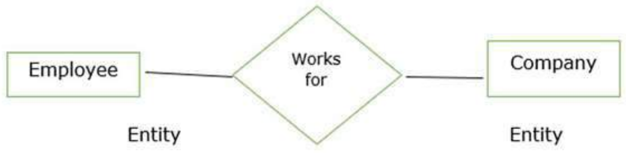
It is used to describe the relation between two or more entities. It is represented by a diamond shape. A relationship type represents the association between entity types.

For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.

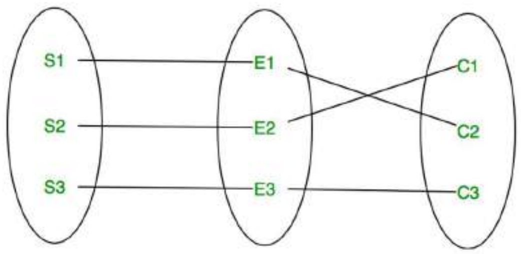


For Example, students study in college and employees work in a department.

The relationship is pictorially represented as follows −



A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 is enrolled in C1, and S3 is enrolled in C3.



Here works for is a relation between two entities.

Degree of Relationship

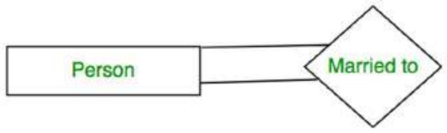
A relationship where a number of different entities set participate is called a degree of a relationship.

It is categorised into the following −

- Unary Relationship
- Binary Relationship
- Ternary Relationship
- n-ary Relationship

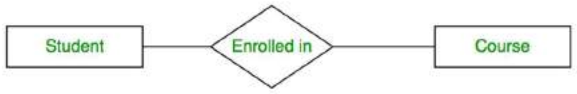
1. Unary Relationship –

When there is **only ONE entity set participating in a relation**, the relationship is called a unary relationship. For example, one person is married to only one person.



2. Binary Relationship –

When there are **TWO entities set participating in a relationship**, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



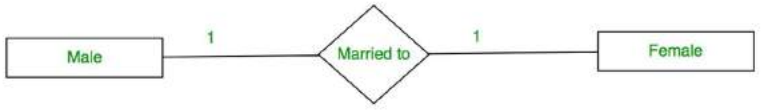
3. n-ary Relationship –

When there are n entities set participating in a relation, the relationship is called an an n-ary relationship.

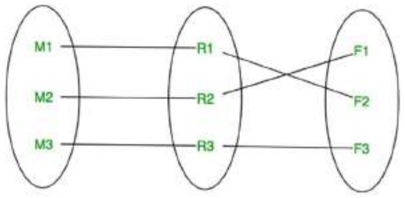
Cardinality:

The **number of times an entity of an entity set participates in a relationship** set is known as cardinality. Cardinality can be of different types:

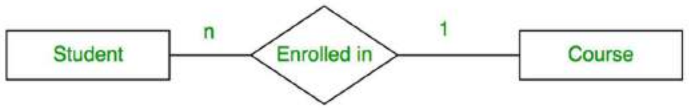
1. One-to-one – When each entity in each entity set can take part **only once in the relationship**, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.
the total number of tables that can be used in this is **2**.



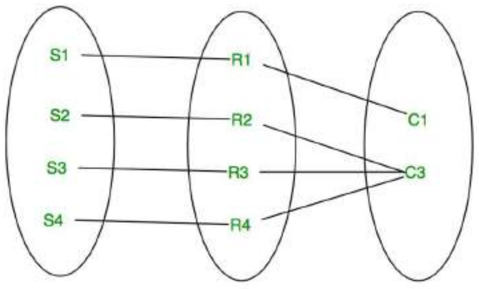
Using Sets, it can be represented as:



2. Many to one – When entities in one entity set **can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set,** cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.
The total number of tables that can be used in this is **3**.



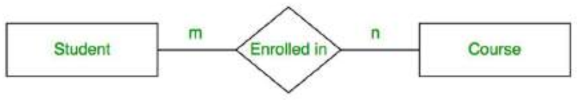
Using Sets, it can be represented as:



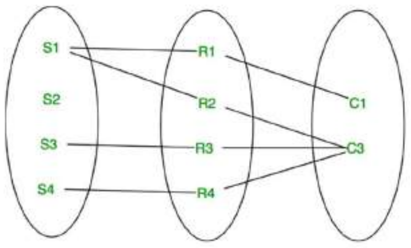
In this case, each student is taking only 1 course but 1 course has been taken by many students.

3. Many to many – When entities in all entity sets can **take part more than once in the relationship** cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

the total number of tables that can be used in this is **3**.



Using sets, it can be represented as:



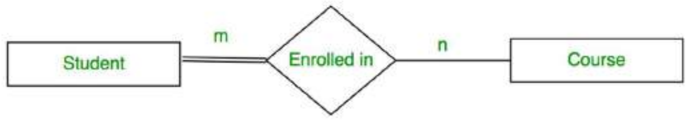
In this example, student S1 is enrolled in C1 and C3 and Course C3 is enrolled by S1, S3, and S4. So it is many-to-many relationships.

In this, there is **one-to-many** mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is **2**.

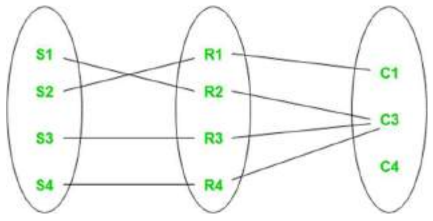
Participation Constraint:

Participation Constraint is applied to the entity participating in the relationship set.

1. Total Participation – Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.
2. Partial Participation – The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the students, the participation of the course will be partial.
The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Using set, it can be represented as,

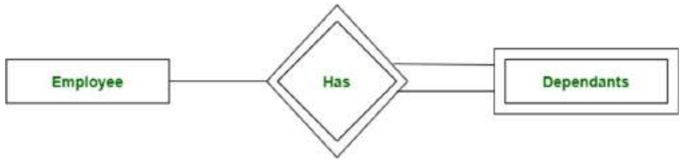


Every student in the Student Entity set is participating in a relationship but there exists a course C4 that is not taking part in the relationship.

Weak Entity Type and Identifying Relationship:

As discussed before, an entity type has a key attribute that uniquely identifies each entity in the entity set. But there exists **some entity type for which key attributes can't be defined**. These are called Weak Entity types.
For example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existed without the employee. So Dependent will be a weak entity type and Employee will be Identifying Entity type for Dependent.

A weak entity type is represented by a double rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.



Extended Entity Relationship (EER) Model:

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

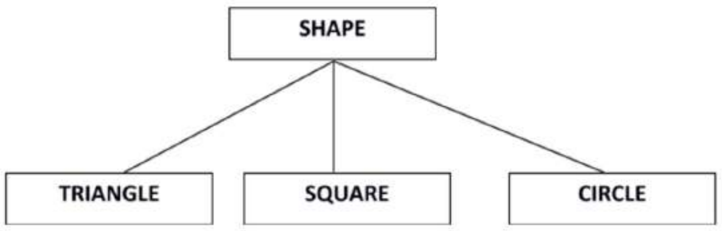
In addition to ER model concepts EE-R includes −

- Subclasses and Super classes.
- Specialization and Generalization.
- Aggregation.

Subclasses and Super class

Super class is an entity that can be divided into further subtype.

For **example** − consider Shape super class.

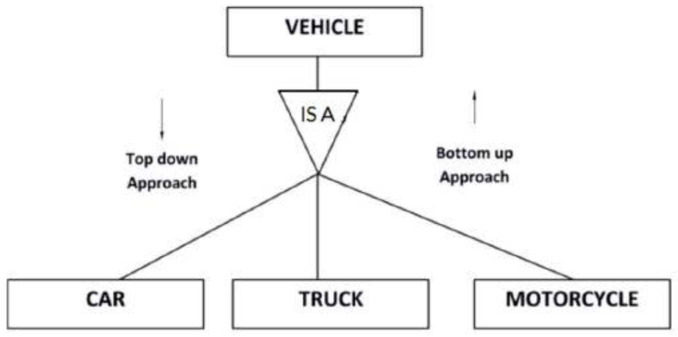


Super class shape has sub groups: Triangle, Square and Circle.

Sub classes are the group of entities with some unique attributes. Sub class inherits the properties and attributes from super class.

Specialization and Generalization

Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities.



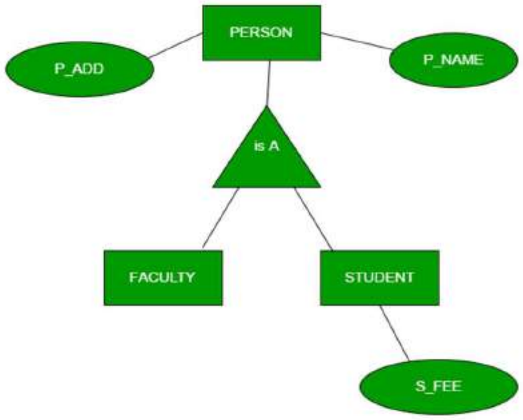
It is a Bottom up process i.e. consider we have 3 sub entities Car, Truck and Motorcycle. Now these three entities can be generalized into one super class named as Vehicle.

Specialization is a process of identifying subsets of an entity that share some different characteristic. It is a top down approach in which one entity is broken down into low level entity.

In above example Vehicle entity can be a Car, Truck or Motorcycle.

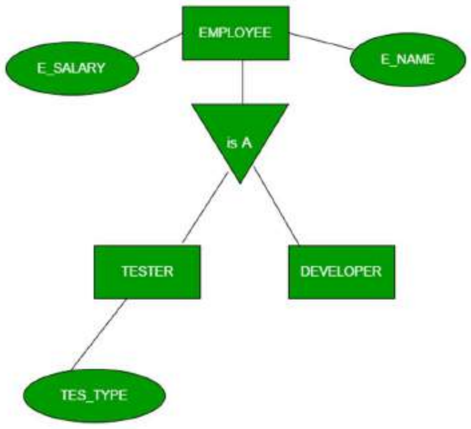
Generalization –

Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



Specialization –

In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).

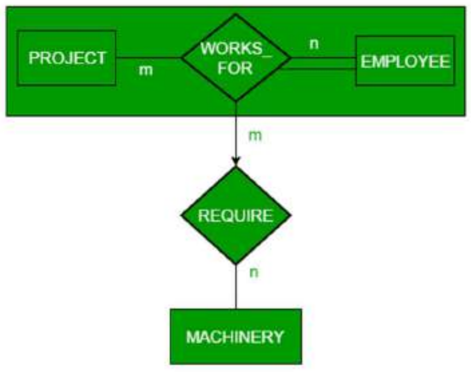


Specialization

Aggregation

An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. Aggregation is an abstraction through which we can represent relationships as higher level entity sets.

For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity andMACHINERY.



Aggregation

Representing aggregation via schema –

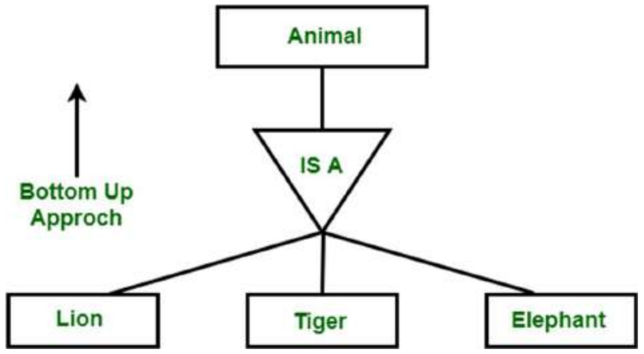
To represent aggregation, create a schema containing:

1. primary key of the aggregated relationship
2. primary key of the associated entity set
3. descriptive attribute, if exists.

Constraints on generalization and specialization

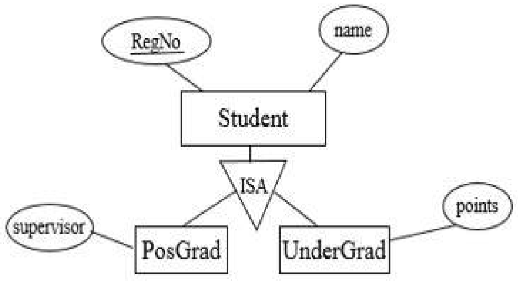
There are three types of constraints on generalization which are as follows:

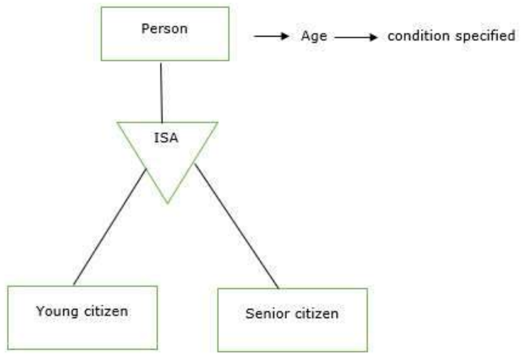
1. First one determines which entity can be a member of the low-level entity set.
2. Second relates to whether or not entities belong to more than one lower-level entity set.
3. Third specifies whether an entity in the higher level entity set must belong to at least one of the lower level entity set within generalization.



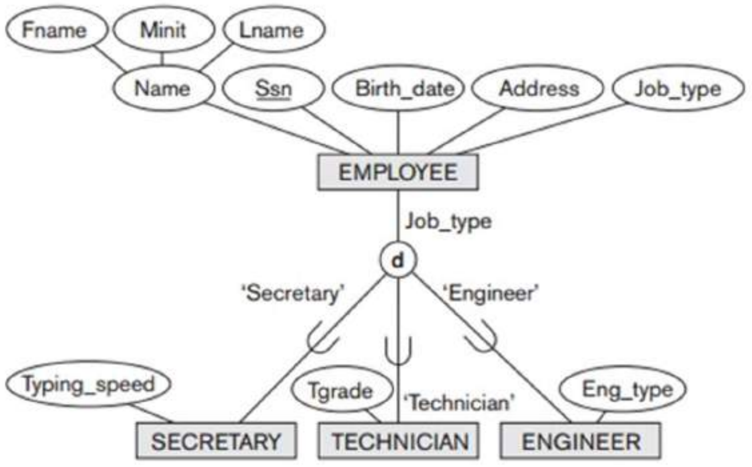
1. First one determines which entity can be a member of the low-level entity set:
Such kind of membership may be one of the following:

- **Condition-defined** –
In this lower-level entity sets, evaluation of membership is on basis whether an entity satisfies an explicit condition or not. For example, Let us assume, higher-level entity set student which has attribute student type. All the entities of student are evaluated by definition of attribute of student. Entities are accepted by the satisfaction of condition i.e. student type = "graduate" then only they are allowed to belong to lower-level entity set i.e. graduate student. By the satisfaction of condition student type = "undergraduate" then they are included in undergraduate student. In fact, all the lower-level entities are evaluated on the basis of the same attribute, thus it is also referred as attribute-defined.





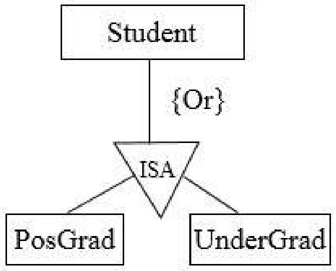
- **User-defined** –
In this lower-level entity sets are not get constrained by a condition named membership; users of database assigns entities to a given entity set. For example, Consider a situation where after 3 months of employment, the employees of the university are assigned to one of four work teams. For this purpose, we represent teams them as four lower-level entity sets of higher-level employee entity set. On the basis of an explicit defining condition, a given employee is not assigned to specific team entity. User in charge of this decision makes the team assignment on an individual basis. By adding entity to an entity set, assignment is implemented.

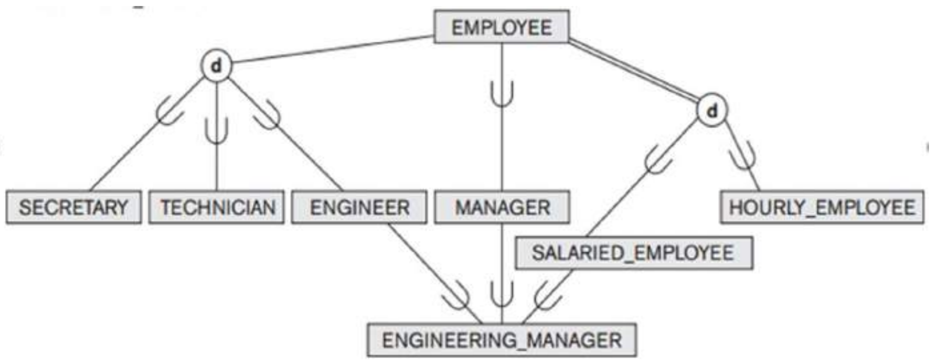


2. Second relates to whether or not entities belong to more than one lower-level entity set :

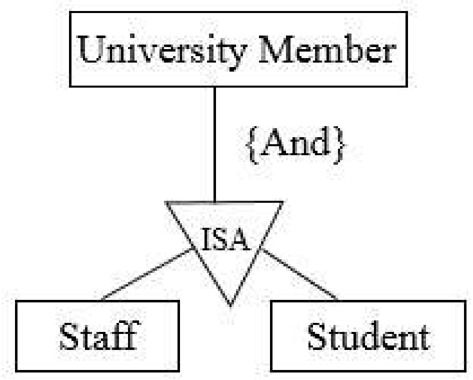
Following is one of the lower-level entity sets:

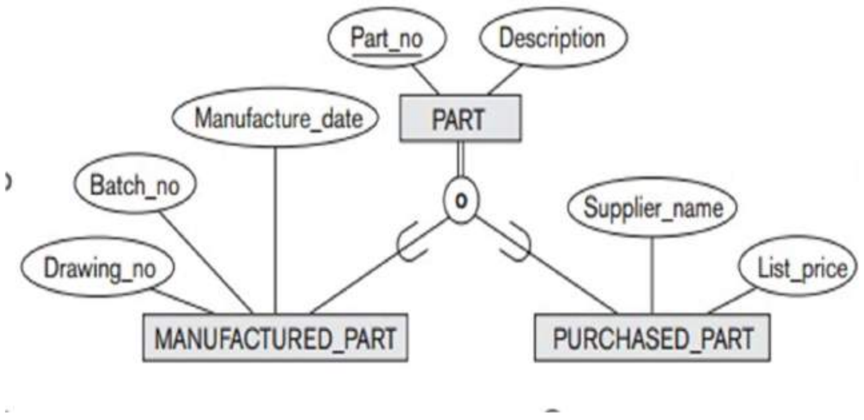
- **Disjoint** –
  The requirement of this constraint is that an entity should not belong to no more than one lower-level entity set. For example, the entity of student entity satisfy only one condition for student type attribute i.e. Either an entity can be a graduate or an undergraduate student, but cannot be both at the same time.





- **Overlapping** –
  In this category of generalizations, within a single generalization, the same entity may belong to more than one lower-level entity set. For example, in the employee work-team assume that certain employees participate in more than one work team. Thus, it offers a given employee that he may appear in more than one of the team entity sets that are lower-level entity sets of employee. Thus, generalization is overlapping.

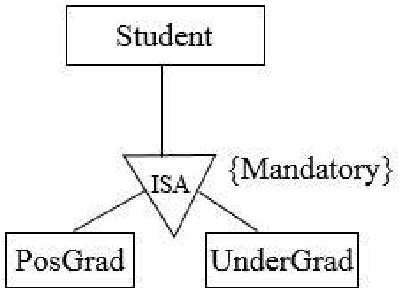




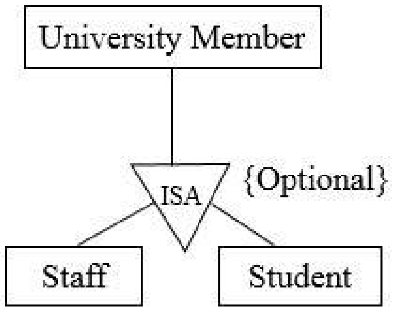
3. Third specifies whether pr not an entity in the higher level entity set must belong to at least one of the lower level entity set within generalization :

This constraint may be one of the following:

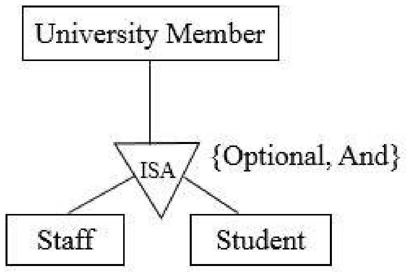
- **Total generalization or specialization –**
  According to this constraint, each higher-level entity must belong to a lower-level entity set.
  - To represent completeness in the specialization/generalization relationship, the keyword "**Mandatory**" is used.



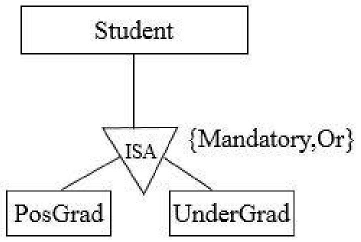
- **Partial generalization or specialization –**
  According to this constraint, some higher-level entities may not belong to any lower-level entity set.
  - The keyword **"Optional"** is used to represent a partial specialization/generalization relationship



We can show both disjoint and completeness constraints in the ER diagram. Following our examples, we can combine disjoint and completeness constraints.



Some members of a university are both students and staff. Not all members of the university are staff and students.



A student in the university must be either an undergraduate or postgraduate, but not both.

Mapping specialization/generalization to relational tables

Method 1

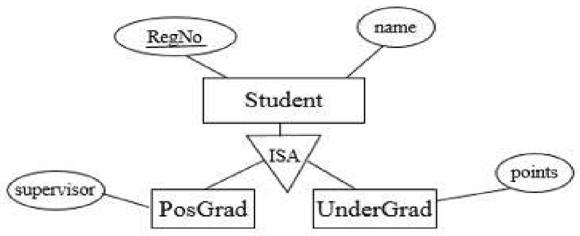
- All the entities in the relationship are mapped to individual tables.
- Student (*Regno*, name)
- PosGrad (*Regno*, supervisor)
- UnderGrad (*Regno*, points)

Method 2

- Only subclasses are mapped to tables. The attributes in the superclass are duplicated in all subclasses.
- PosGrad (*Regno*, name, supervisor)
- UnderGrad (*Regno*, name, points)
- This method is most preferred when inheritance is disjoint and complete, e.g. every student is either PosGrad or UnderGrad and nobody is both.

Method 3

- Only the superclass is mapped to a table. The attributes in the subclasses are taken to the superclass.
- Student (*Regno*, name, supervisor, points)


RegNo
name
Student
ISA
supervisor
PosGrad
UnderGrad
points