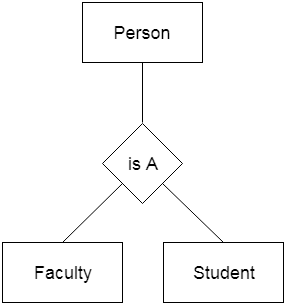
**Extended ER Model (EERD)**

As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.

**Enhanced entity-relationship diagrams** are advanced database diagrams very similar to regular ER diagrams which represent requirements and complexities of complex databases. It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

* **Generalization**
* Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher-level entity if they have some attributes in common.
* In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher-level entity.
* Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
* In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.
* A sub-class is best understood by **“IS-A analysis”**. Following statements hopefully makes some sense to your mind “Technician IS-A Employee”, “Laptop IS-A Computer”.

**For example,** Faculty and Student entities can be generalized and create a higher-level entity Person.

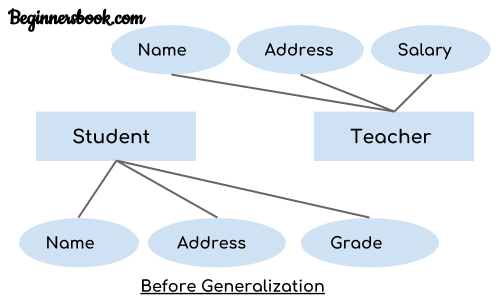


**Generalization Example:** Let’s say we have two entities Student and Teacher.

**Attributes of Entity Student are:** Name, Address & Grade

**Attributes of Entity Teacher are:** Name, Address & Salary

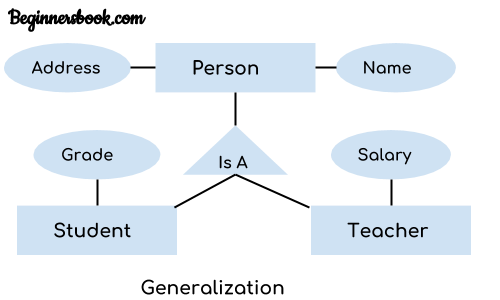
The ER diagram **before generalization** looks like this:



These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes. Let’s have a look at the ER model after generalization.

The ER diagram **after generalization**:

We have created a new generalized entity Person and this entity has the common attributes of both the entities. As you can see in the following ER diagram that after the generalization process the entities Student and Teacher only has the specialized attributes Grade and Salary respectively and their common attributes (Name & Address) are now associated with a new entity Person which is in the relationship with both the entities (Student & Teacher).



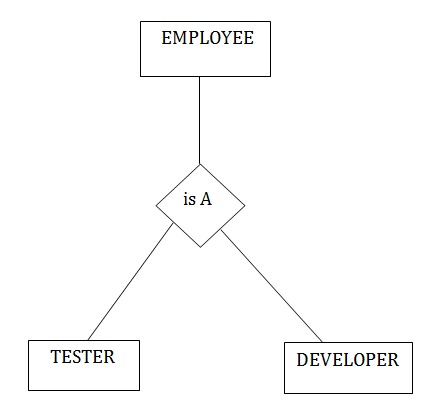
***Note:***

1. Generalization uses bottom-up approach where two or more lower-level entities combine together to form a higher-level new entity.

2. The new generalized entity can further combine together with lower-level entity to create a further higher level generalized entity.

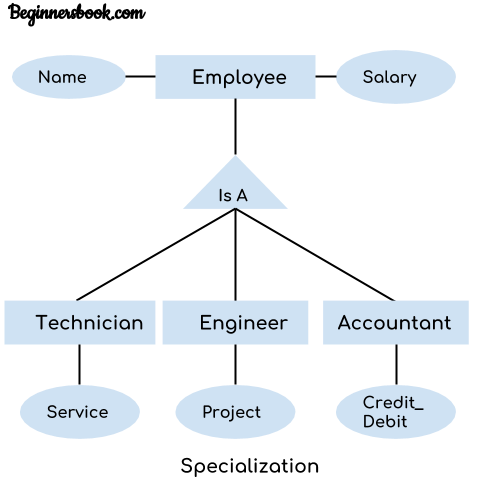
* **Specialization**
* Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower-level entities.
* Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
* Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

**For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



The idea behind Specialization is to find the subsets of entities that have few distinguish attributes. For example – Consider an entity employee which can be further classified as sub-entities Technician, Engineer & Accountant because these sub entities have some distinguish attributes.

**Specialization Example**

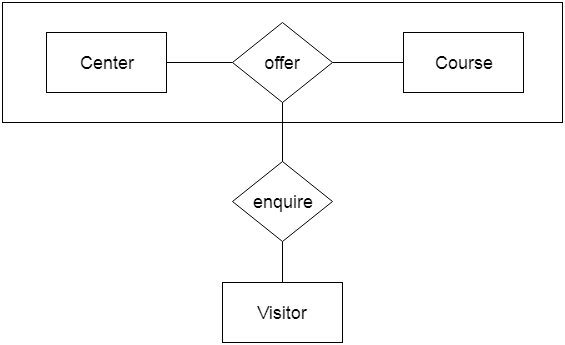


In the above diagram, we can see that we have a higher-level entity “Employee” which we have divided in sub entities “Technician”, “Engineer” & “Accountant”. All of these are just an employee of a company; however, their role is completely different and they have few different attributes. Just for the example, I have shown that Technician handles service requests, Engineer works on a project and Accountant handles the credit & debit details. All of these three employee types have few attributes common such as name & salary which we had left associated with the parent entity “Employee” as shown in the above diagram.

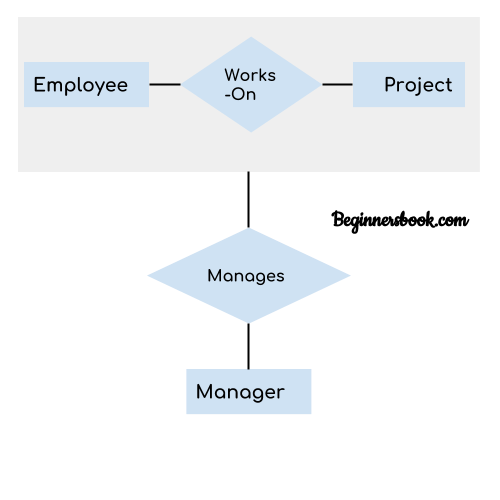
* **Aggregation**

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher-level entity.

**For example:** Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



**Aggregation Example**

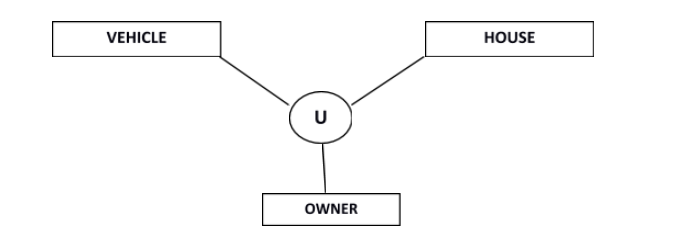


In real world, we know that a manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity “Manager” makes a “manages” relationship with either “Employee” or “Project” entity alone then it will not make any sense because he has to manage both. In these cases, the relationship of two entities acts as one entity. In our example, the relationship “Works-On” between “Employee” & “Project” acts as one entity that has a relationship “Manages” with the entity “Manager”.

* **Union or Category**

Category represents a single super class or sub class relationship with more than one super class.

Set of Library Members is UNION of Faculty, Student, and Staff. A union relationship indicates either type; for example, a library member is either Faculty or Staff or Student.



Owner is the subset of two super class: Vehicle and House.

**For example,** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.

