EER Diagram

The **Enhanced Entity-Relationship (EER)** model is a high-level or conceptual data model incorporating extensions to the original Entity-relationship (ER) model discussed in the previous chapter. Like ER model, it is used for a logical representation of databases. It was developed to cater the need to model more precise properties and constraints that are found in complex databases, such as Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), telecommunications, and Geographic Information Systems (GIS). The EER Model includes all of the modeling concepts introduced by the ER model along with the additional concepts of subclass/superclass that further define the concepts of specialization/generalization and type inheritance, union, aggregation and composition. In this chapter we discuss all these concepts along with their representation techniques in EER diagram to model complex databases.

3.1 Subclass and Superclass

An entity type (discussed in the previous chapter) in many cases can have further sub-groupings of its entities and these subgroups, mostly being meaningful, need to be represented explicitly because of their significance to the database application. For example, the entities belonging to the Employee entity type in a software development company may be further grouped into Manager, Database Administrator, Database Designer, and Application Developer. The set of entities belonging to each of these later groupings is a subset of the entities that belong to the Employee entity set i.e. every entity that is a member of these subgroupings is also a member of Employee entity set. If such subgroups exist for an entity set E, we call each such subgroup a **subclass** of the entity type E and the entity type E as the **superclass** of these subgroups. It means that Manager is a subclass of the entity type Employee and the entity type Employee is the superclass of the subgroups Manager, Database Administrator, Database Designer, and Application Developer. Like Manager, Database Adminsitrator, Database Designer and Application Developer are also subclasses of Employee entity type. The relationship between a superclass and any one of its subclasses is called a superclass/subclass relationship, e.g., Employee/Manager has a superclass/subclass relationship. Using the concept of superclass and subclass avoids describing different types of entities with possibly different attributes within a single entity. For example, Manager may have additional attributes than the attributes of other employees like *manager start date* and *average monthly business of the manager*. It can be noted that if all the employee attributes and the attributes specific to the subclasses

of employees are described by a single Employee entity, it may result in null values for the subclass specific attributes for a lot of employees.

Thus the reason behing using the concepts of the superclass and the subclass can be summarized as follows:

- To avoid describing similar concepts more than once, thereby saving time for the designer and making the ER diagram more readable.
- To add more semantic information to the design in such a way that it becomes familiar to many people. For example, the assertions that “Manager IS-A member of Employees” communicates significant semantic content in a concise form.

Being merely a member of a subclass is not sufficient for a particular entity to be included in the database. Rather, alongwith the membership of the subclass the entity must also be a member of its superclass. However, it is not necessary for an entity of a superclass to be a member of one or more subclasses. For example, in the superclass/subclass relationship Employee/Manager all managers are also an employee but all employees need not be a manager.

3.2 Type Inheritance

In object-oriented programming inheritance is a way to form new classes using classes that have already been defined. The new classes, known as subclasses (or derived classes), inherit attributes and behavior of the pre-existing classes, which are referred to as superclasses (or ancestor classes). An entity in the subclass represents the same “real world” object as in the superclass and may possess some subclass-specific attributes in addition to those associated with the superclass i.e. we can say that a member of subclass *inherits* all the attributes of its superclass along with having its own specific attributes. For example, a member of the Manager subclass inherits all the attributes of the Employee superclass like empId, empName, empDesignation, empDateOfBirth etc. together with those specifically associated with the Manager subclass like managerStartDate. In addition to attributes, subclasses also inherit all the relationships in which the superclass participates. For example, if the entity type Employee is associated with Department entity type through WorkFor relationship, then all its subclasses will also participate in the WorkFor relationship.

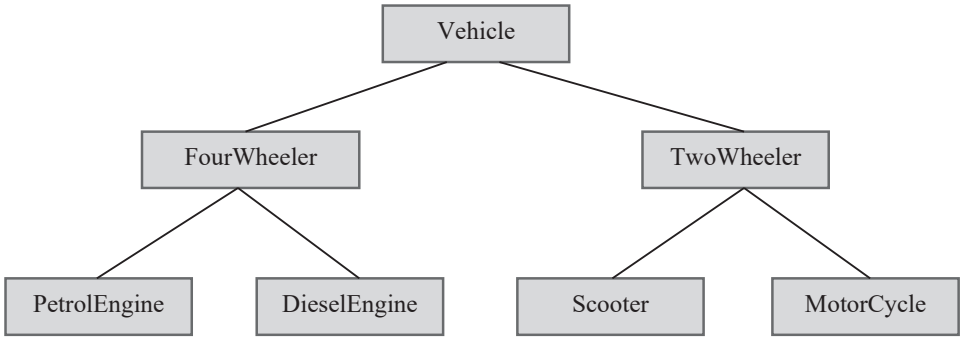

Figure 3.1 – A type hierarchy for `Vehicle` entity type

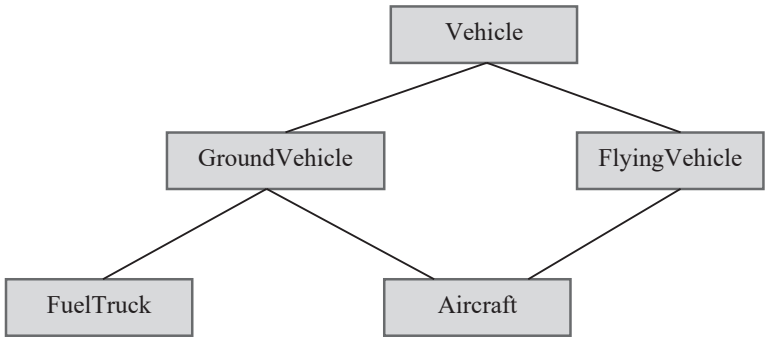

Figure 3.2 – A multiple-inheritance (lattice) in which the entity type `Aircraft` inherits attributes from `GroundVehicle` and `FlyingVehicle` entity types.

Being an entity, it is also possible for a subclass to have one or more subclasses. Similarly the subclasses of a subclass may further have one or more subclasses and so on forming a **type hierarchy**. Figure 3.1 shows a type hierarchy of `Vehicle` entity type. A type hierarchy can be represented by various names; generalization hierarchy (for example, `Employee` is a generalization of `Manager`), specialization hierarchy (`Manager` is a specialization of `Employee`), IS-A hierarchy (`Manager` IS-A `Employee`). Generalization and specialization are further discussed in the following section.

In case a subclass inherits attributes from multiple superclasses, the subclass is called a **shared subclass** and the type of inheritance demonstrated by the subclass and its super classes is referred as **multiple inheritance** and it forms a lattice. Another classification of Vehicle entity type to exemplify the concept of multiple inheritance is shown in figure 3.2.

3.3 Specialization and Generalization

The process of identifying and defining the set of subclasses of an entity type is referred to as **specialization**. The set of subclasses that forms the specialization is identified on the basis of some distinguishing characteristics of the entities in the entity type that represents the superclass. Following a top-down approach, as we identify a set of subclasses of an entity type, we associate attributes specific to each subclass (where necessary), and also identify any relationships between each subclass and other entity types or subclasses (where necessary). For example, the set of subclasses {Permanent, Temporary} represents the specialization of the superclass Employee and distinguishes the entities belonging to the Employee superclass based on the *nature of appointments* of the employees. It is possible to have multiple specializations of a single entity type based on different distinguishing characteristics. For example, the set of subclasses {Salaried, HourlyPaid} represents another specialization of the Employee entity type based on the *method of payment* for an employee. It should be however noted that within a specialization, it is possible that two subclasses overlap i.e. the same entity of the superclass may be a member of more than one subclass (overlapping subclasses) of the specialization for example an employee can be salaried but get paid on hourly basis for working on some extra projects as well.

In order to draw EER diagram, the subclasses that define a specialization are attached by lines to a circle (not used if the specialization contains only one subclass), which is connected to the superclass with single (or double) lines. Depending on the characteristics of specialization, the circle contains either 'd' or 'o' to represent *disjoint* or *overlap* respectively. The *subset* symbol on each line connecting the subclass to the circle indicates the direction of the superclass/subclass relationship. Each subclass is attached with its subclass-specific attributes. The significance of the single and double lines from the superclass to the circle and the letters d and o within the circle will be discussed in the following section. Figure 3.3 shows the EER diagram to represent specialization and subclasses for Employee entity type. The figure defines two specializations of Employee – {Permanent, Temporary} and {Salaried, HourlyPaid}.

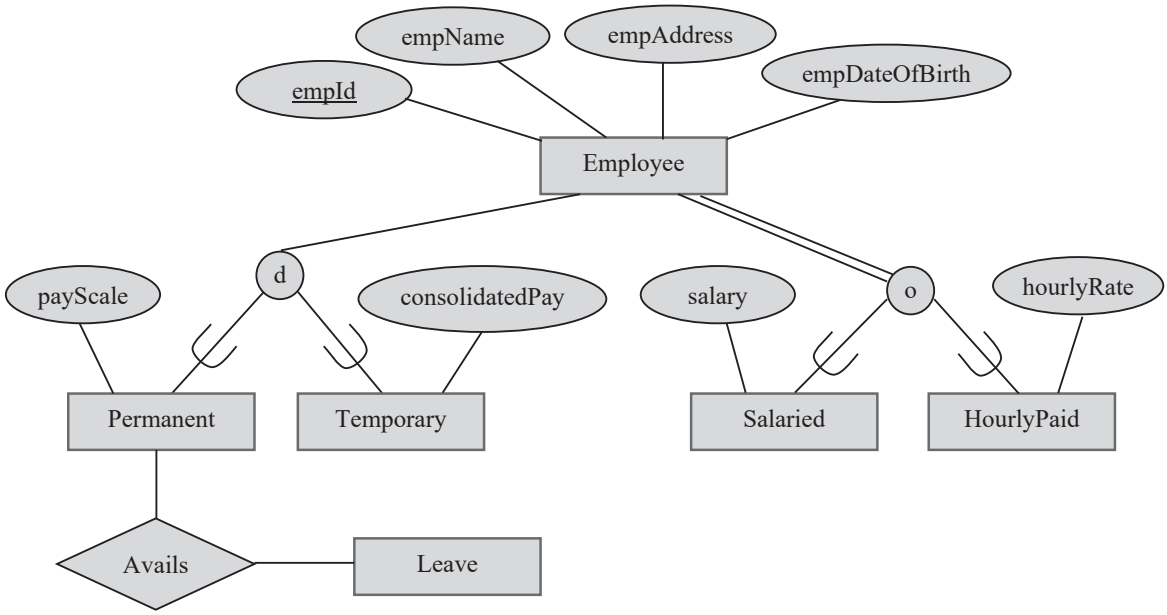

Figure 3.3 - EER diagram representing specialization of `Employee` entity type

Generalization is the inverse process of specialization. It follows a bottom-up approach to identifify a generalized superclass from the original entity types. This involves suppressing the differences among the several entity types, identifying their common features and generalizing them into a single superclass. For example, consider the entity types `Bus` and `Truck` shown in figure 3.4 (a). As their attributes depict, they can be generalized into the entity type `Vehicle` as shown in figure 3.4 (b).

As already mentioned that the generalization process can be viewed as being functionally inverse of the specialization process, figure 3.4 (b) can also be viewed as a specialization of `Vehicle` into `{Bus, Truck}` and the figure 3.3 can be viewed as a generalization of `{Permanent, Temporary}` into `Employee`. To distinguish between generalization and specialization, often a diagrammatic notation is used in some design methodologies where an arrow pointing to the generalized superclass represents generalization and the arrow pointing to the specialized subclass represents a specialization. But, mostly such a notation is not used as the decision as to which process is more appropriate in a particular situation is often subjective.

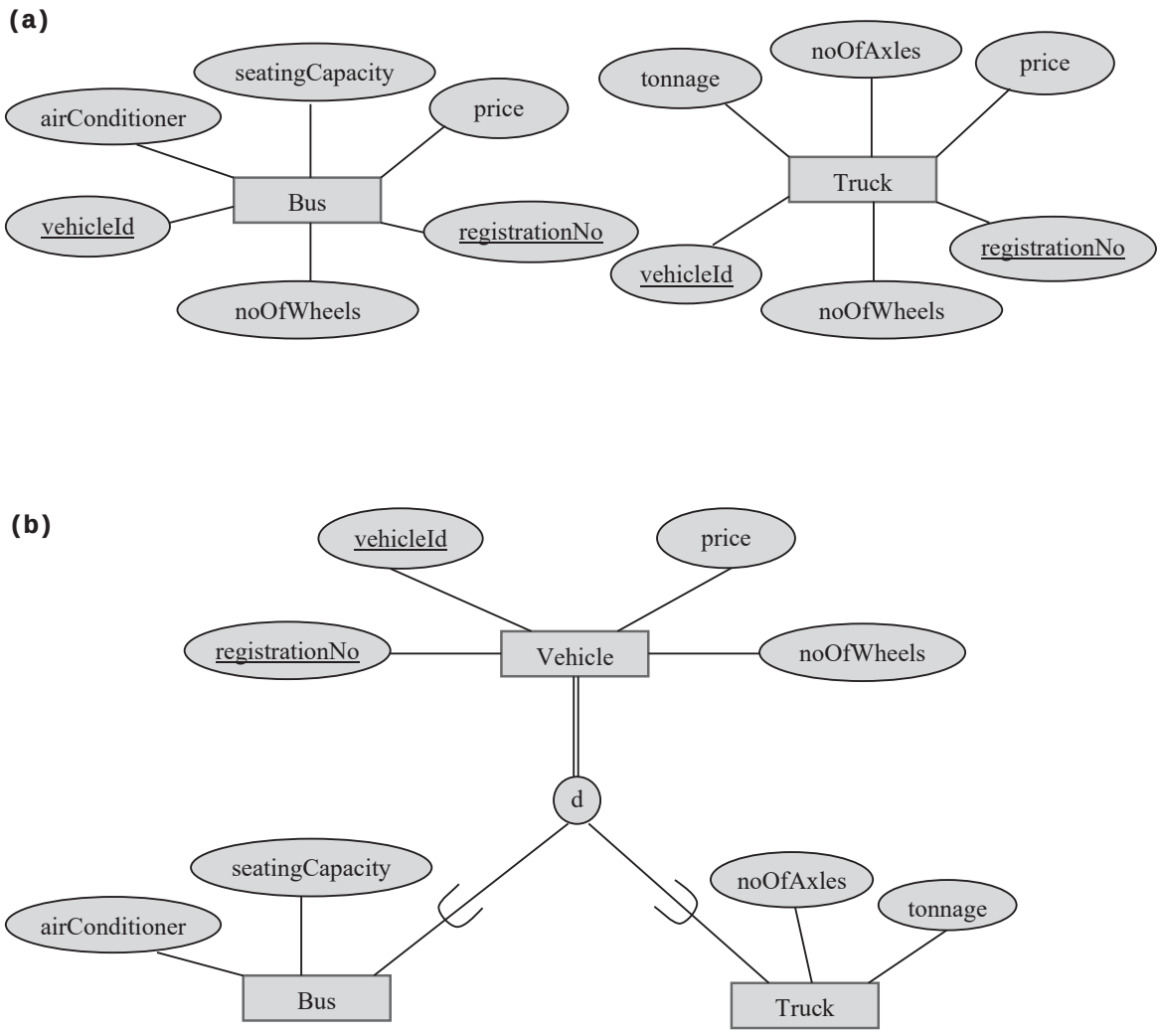

Figure 3.4 – Example of generalization (a) Two entity types `Bus` and `Truck` (b) Generalization of `Bus` and `Truck` into `Vehicle`.

3.3.1 Types of Specialization/Generalization

In this section we discuss different types of specialization/generalization. Since specialization and generalization processes are functionally inverse to each other; for brevity, our discussion refers only to specialization but it applies equally to both specialization and generalization. Table 3.1 presents a summarized view of the types of specialization that can be defined for an entity type. As discussed earlier, an entity type (superclass) may have specializations to different set of subclasses and for entities of the superclass it is not mandatory that they will belong to all subclasses of a specialization. Therefore, in some specializations predicates can be defined on the value of some

attribute of the superclass to determine exactly the superclass entities that will become members of each subclass. Such a subclass is called **condition-defined** (or **predicate-defined**) subclass. The predicate-defined subclasses are displayed in EER diagram by writing the predicate condition as a label to the line joining the subclass to the specialization circle. As shown in figure 3.5, the appointmentType attribute of Employee can be used to define predicates (appointmentType = 'Permanent') for Permanent subclass. This means that the members of the Permanent subclass must satisfy the predicate appointmentType = 'Permanent' and all entities of the Employee entity type whose attribute vale for appointmentType is 'Permanent' must belong to the Permanent subclass. Similarly, by using the predicate appointmentType = 'Temporary' the members of Temporary subclass can be determined.

Table 3.1 – Types of specialization/generalization

• Condition-defined or Predicate-defined specialization/generalization
• Attribute-defined specialization/generalization
• User-defined specialization/generalization

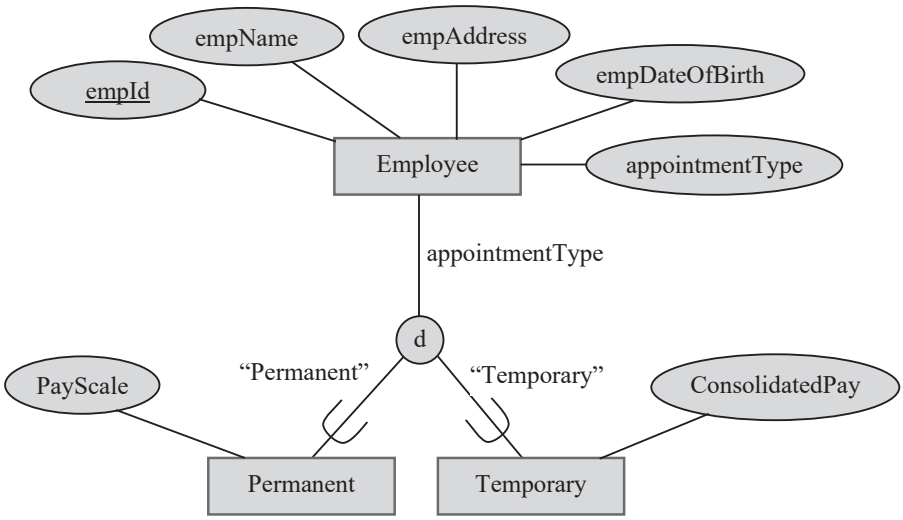

Figure 3.5 – An attribute-defined specialization on the appointmentType attribute of Employee entity type

If all subclasses in a specialization are conditioned-defined on same attribute of the superclass, the specialization is called **attribute-defined specialization** and the attribute is called the defining attribute (or discriminator in UML terminology) of the specialization. Otherwise, the specialization (subclass) is called **user-defined**

specialization (user-defined subclass). In EER diagram, as shown in figure 3.5, we write the defining attribute as label of the line joining the specialization circle to the superclass. Membership in a user-defined subclass is determined by the database users while adding entities to the subclass, not by any condition that can be evaluated automatically.

3.3.2 Constraints on Specialization and Generalization

In this section we discuss different types of constraints that can be applied to specialization and generalization. Like discussing types of specialization in the previous section, for brevity, our discussion about constraints refers only to specialization but it applies to both specialization and generalization. The constraints that apply to specialization and generalization are summarized in table 3.2 and discussed in the following paragraphs.

Table 3.2 – The constraints that apply to specialization and generalization

- Disjointness/Overlap Constraint
- Completeness Constraint
    - Total Specialization
    - Partial Specialization

Since Disjointness/Overlap and Completeness constraints are independent, the constraints on specialization can be redefined as:

- Disjoint, total
- Disjoint, partial
- Overlapping, total
- Overlapping, partial

Disjointness/Overlap Constraint

The disjointness/overlap constraint specifies that whether in a superclass/subclass relationship it is possible for a member of a superclass to be a member of one or more than one subclass. In case a superclass has more than one subclass in a specialization, the disjointness constraint specifies that the subclasses in the specialization must be disjoint, i.e., an entity occurrence can be a member of at most one of the subclasses of the specialization. For example, the subclasses `Permanent` and `Temporary` of the

superclass Employee, shown in figure 3.3, are disjoint which means that an employee can have either permanent or temporary (but not both) appointment in an organization. In order to represent disjointness constraint in the EER diagram, we place a symbol 'd' in the circle that connects a superclass and all its subclasses indicating that a member of the superclass can be a member of only one of its subclasses.

In case the subclasses of a specialization are not disjoint, their set of entities may overlap, i.e., an entity can be a member of more than one subclass of the specialization. For example, in figure 3.3 the subclasses Salaried and HourlyPaid of the superclass Employee are not disjoint. An employee can be a member of both of these subclasses since the two subclasses overlap – as an employee can be salaried but get paid on hourly basis for working on some extra projects as well. This type of constraint (overlap constraint), which is the default, is represented in the EER diagram by placing the letter 'o' in the circle connecting a superclass and its subclasses as shown in figure 3.3.

Completeness Constraint

The completeness constraint determines whether every member in the superclass must participate as a member of a subclass. It may be total or partial. If it is mandatory that every entity in the superclass must be a member of some subclass in the specialization, then the specialization is said to be **total specialization**. For example, the {Salaried, HourlyPaid} specialization of the Employee entity type in figure 3.3 is a total specialization as every member of the Employee superclass must be a member of either the salaried subclass or the hourlyPaid subclass because every employee has either of the two methods to get payment. In the EER diagram, the total participation is represented by specifying double lines when connecting a superclass with the circle that connects its subclasses as shown in figure 3.3.

Another type of completeness constraint is the **partial specialization** which specifies that it is not mandatory for the members of a superclass to belong to any of its subclasses. For example, for the members of the superclass Employee, it is not mandatory to be a member of the subclasses Permanent and Temporary as an employee can have other type of appointments like *contractual*, *against leave vacancy*, etc. In the EER diagram, partial specialization is represented by using single line to connect the superclass entity type with the circle that connects its subclasses, as shown in figure 3.3.

Since, the disjointness/overlap and completeness constraints are independent; combining these together may yield the following four types of constraints on specialization:

- Disjoint-total

- Disjoint-partial
- Overlapping-total
- Overlapping-partial

A specialization of a superclass entity type in which all subclasses are disjoint and every entity of the superclass must belongs to a subclass is called **disjoint-total specialization**. For example, in figure 3.6 the subclasses Teaching and NonTeaching are disjoint and an entity of the Employee class must belong to either Teaching subclass or NonTeaching subclass. Hence, the specialization shown in figure 3.6 is disjoint-total.

A specialization of a superclass entity type in which all subclasses are disjoint and it is not mandatory for an entity of the superclass to be a member of a subclass is called **disjoint-partial specialization**. For example, in figure 3.7 the subclasses Permanent and Temporary are disjoint and every entity of the Employee class is not compelled to be a member of these subclasses as there might be some employees having different types of appointments like *contractual*, *daily wager*, etc. Hence, the specialization shown in figure 3.7 is disjoint-partial.

A specialization of a superclass entity type in which all subclasses are not disjoint rather overlapping and every entity of the superclass must belongs to a subclass is called **overlapping-total specialization**. For example, in figure 3.8 the subclasses Salaried and HourlyPaid are overlapping as there may be some employee who is salaried but get paid on hourly basis for working on some extra projects as well. Moreover, every Employee entity will be a member of either Salaried or HourlayPaid because each of them has either of the two methods to get payment. Hence, the specialization shown in figure 3.8 is overlapping-total.

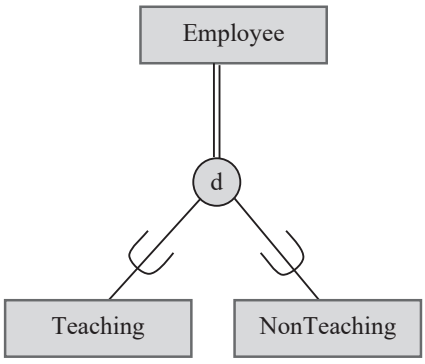

Figure 3.6 – A disjoint-total specialization of Employee entity type

A specialization of a superclass entity type in which all subclasses are not disjoint rather overlapping and it is not mandatory for an entity of the superclass to be a member of a subclass is called **overlapping-partial specialization**. For example, in figure 3.9 the specialization of `Car` entity type into the subclasses `PetrolEngineCar`, `DieselEngineCar`, and `CNGEngineCar` are overlapping – as there may be combo-engine cars, i.e. a car may have petrol as well as CNG engine. Moreover, if there is a car that has electric engine it will not be a member of either of these classes. Hence, the specialization shown in figure 3.9 is overlapping-partial.

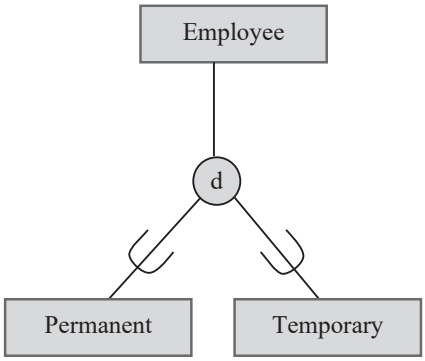

Figure 3.7 – A disjoint-partial specialization of `Employee` entity type

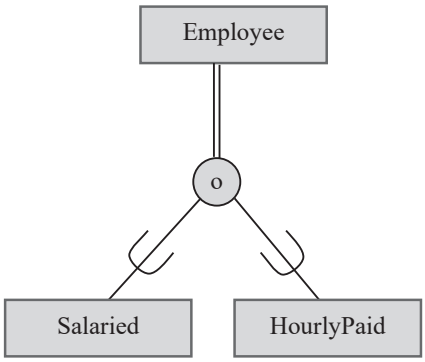

Figure 3.8 – An overlapping-total specialization of `Employee` entity type

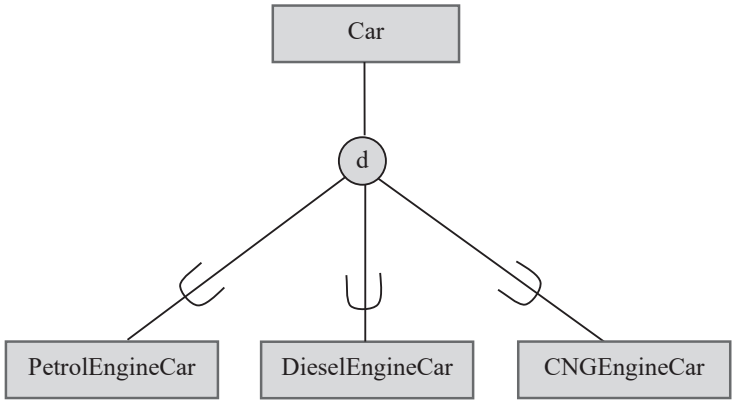

Figure 3.9 – An overlapping-partial specialization of Car entity type

3.4 UNION Type or Category

In the previous sections, we have discussed two different types of superclass/subclass relationships – (i) a type hierarchy, as shown in figure 3.1, in which a subclass has only one superclass, (ii) a lattice, as shown in figure 3.2, in which a subclass is shared among different superclass/subclass relationships. For example, in figure 3.2, the shared subclass, Aircraft, is the subclass participating in two distinct superclass/subclass relationships, where each of the two relationships has a single superclass. However, it is not uncommon that we may need to model a single superclass/subclass relationship with more than one superclass, where the superclasses represent different entity types. In such a case, the entity set for the subclass represent a collection of entity instances that is a subset of the UNION of the instances of distinct entity types (superclasses) and such a subclass is called a **union type** or **a category**. In EER diagram, all superclasses of a subclass are connected through a circle which contains a union symbol (∪) representing the *set UNION operation*. An arc with the subset symbol connects the circle to the subclass (category). It specifies that the entity set of the subclass is the subset of the UNION of the entity-instances of the superclass entity types.

For example, consider the entity types: Graduate, and NonGraduate. In a database dealing with engineering project allotment, each of the previous two entities can be an employee. Designing a model for such a database will involve creating a class (collection of entities) that includes entities of all the above two categories. That is, a category Employee which would be a subclass of the UNION of the entity types Graduate, and NonGraduate is created for this purpose. Similarly, a category, engineeringProject, is created which entity set is the subset of the UNION of the entity instances of the superclasses LongTermProject and ShortTermProject. The EER diagram modeling using categories for this assignment is shown in figure 3.10.

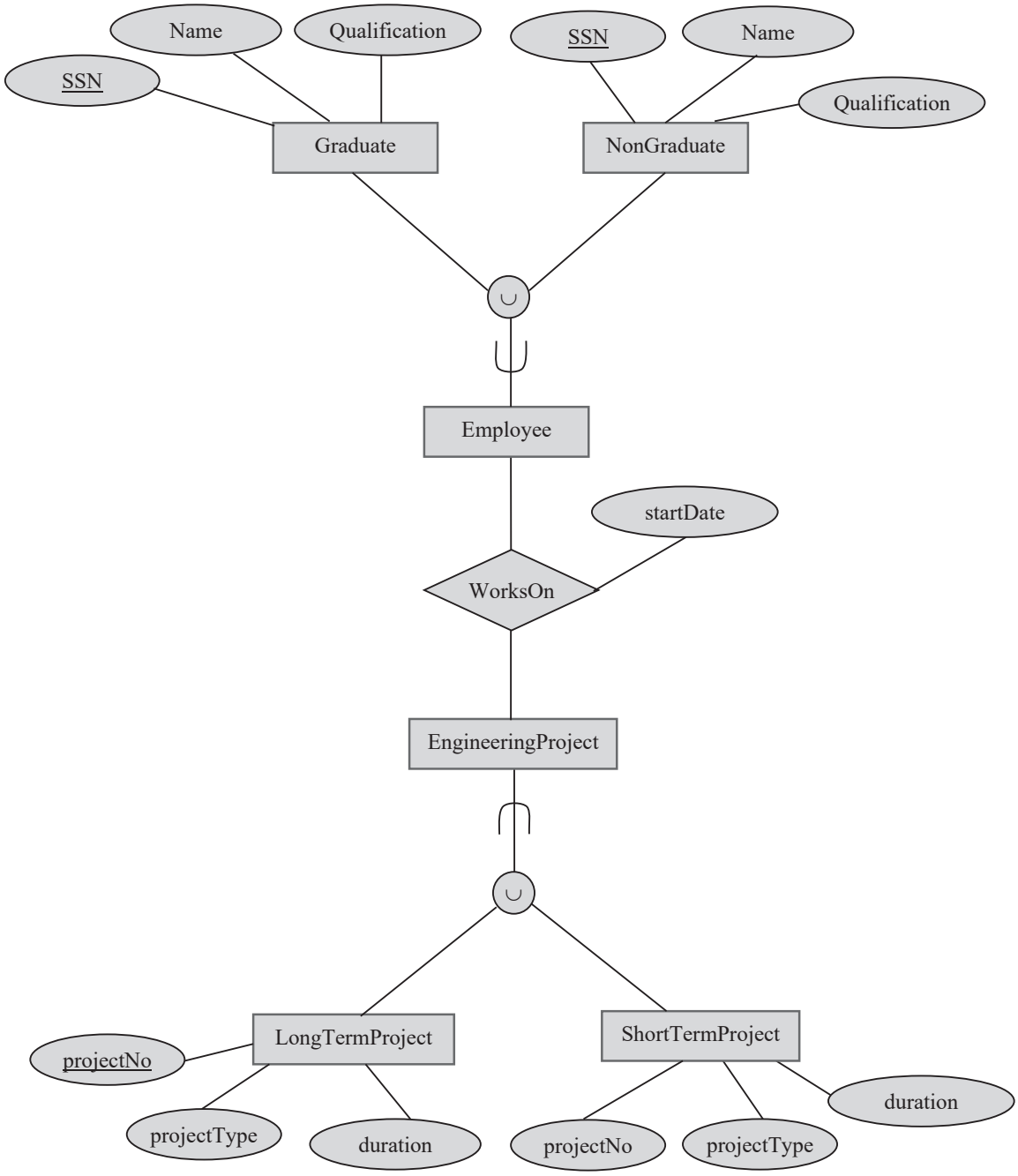

Figure 3.10 – An EER diagram modeling using categories for *engineering project* assignment

3.4.1 Difference Between Category and Lattice

It should be noted that a category is not same a lattice (multiple inheritance) in which too a subclass has two or more superclasses. For example, in figure 3.2 `Aircraft` is a

subclass of two superclasses GroundVehicle and FlyingVehicle, so an entity that is a member of Aircraft must exist in both the superclasses GroundVehicle and FlyingVehicle. This represents the constraint that the entity set of Aircraft must be a subset of the intersection of the entity sets of GroundVehicle and FlyingVehicle. On the other hand, a category has two or more superclasses that may represent distinct entity types. That is, a category is the subset of the UNION of its superclasses. Hence, it represents the constraint that an entity which is a member of a category must exist in only one of its superclasses. For example, in figure 3.10 an employee may be a graduate or non-graduate.

From attribute inheritance point of view, in category the attribute inheritance works more selectively as a subclass entity inherits only the attributes of the superclass to which it belongs and not of all its superclasses. On the other hand, in a lattice the subclass inherits all the attributes of its superclasses.

3.4.2 Types of Category

Like completeness constraint on specialization and generalization, a category can be **total** or **partial** depending on whether all the entities of its superclasses must belong to the category or not. In case all the entities of the superclasses must belong to the subclass (category), the category is said to be total otherwise partial. In a partial category, the subclass (category) is connected with the circle using single line. In figure 3.10, since it is not necessary that all graduates or non-graduates will be an employee, the Employee subclass is a partial category and connected with the circle using single line. Similary, the EngineeringProject category in figure 3.10 is also partial as for a long-term or short-term project it is not mandatory that it will an engineering project.

On the other hand, in a total category, the subclass (category) is connected with the circle using double line. For example, in figure 3.11 (a), every salaried or hourly-paid employee must be member of Employee. Hence, the category Employee is a total category and it is connected with the circle using double lines. A total category can have alternate representation using the concept of specialization/generalization. Figure 3.11 (b) presents the alterate representation of Employee category (shown in figure 3.11 (a)) using specialization/generalization. The choice of representation to use category or specialization/generalization is subjective. If the superclasses for a particular subclass represent the same type of entities and share numerous attributes specialization/generalization is preferred; otherwise use of category to model the relationships is more appropriate.

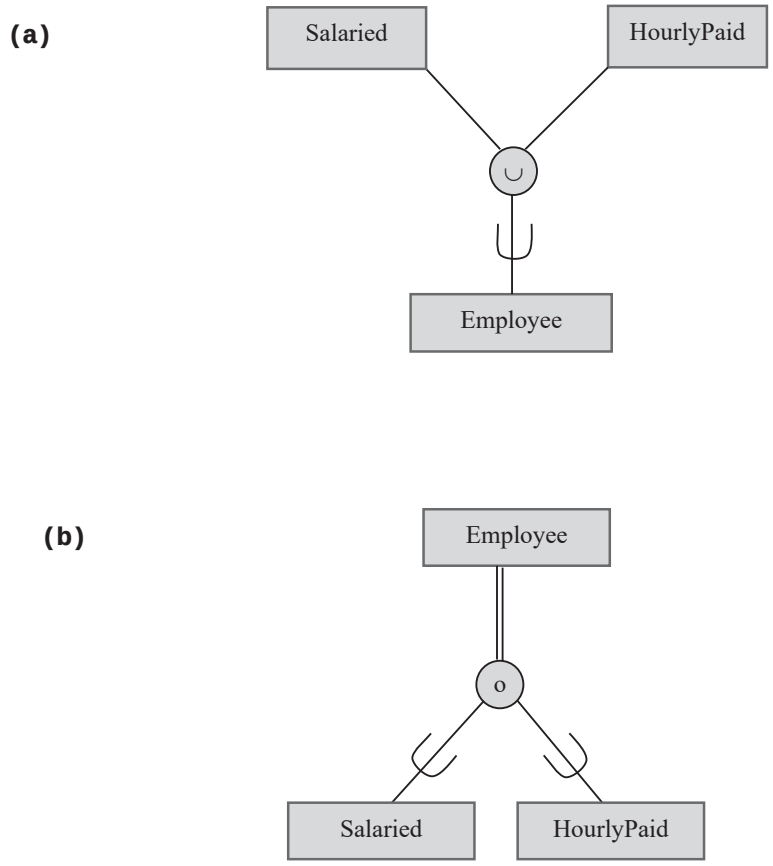

Figure 3.11 – A total category and alternate representation **(a)** A total category – `Employee`, **(b)** Alternate representation of `Employee` using specialization/generalization

3.5 Association, Aggregation and Composition

Some times we need to model association between distinct classes in which one entity represents a larger entity (the 'whole') consisting of smaller entities (the 'parts'). **Association** represents the ability to associate objects from several independent classes, i.e., to model '`IS-ASSOCIATED-WITH`' relationship. It is a type of relationship in which all objects have their own lifecycle and there is no owner. For example, consider the relationship between `Teacher` and `Student`. Multiple students can be associated with a single teacher and a single student can be associated with multiple teachers but, there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.

Both aggregation and composition are special kinds of associations and indicate something more about the relationship. **Aggregation** is a special kind of association in which all objects have their own lifecycle but there is ownership, and child object can not belongs to another parent object. In other words, in an aggregation the instances cannot

have cyclic aggregation relationships (i.e. a part can not contain its whole). Aggregation is used to represent ownership or a whole/part ('HAS-A' or 'IS-A-PART-OF') relationship. For example, consider the association between Department and Teacher which models "Department HAS Teacher" or "Teacher IS-A-PART-OF Department" relationship. This represents an aggregation in which the entity type Department is the owner and a single teacher can not belongs to multiple departments, but if we delete the department, the teacher object will not destroy. Similarly, an order consists of several products, but a product continues to exist even if the order is destroyed. Hence, the relationship between Order and Product is an aggregation.

Like aggregation, **Composition** is also used to model 'HAS-A' or 'IS-A-PART-OF' relationships but, in a composition, the child objects do not have their own lifecycle. That is, if a parent object is deleted all its child objects will also be deleted. In other words, a composition represents a strong ownership and coincidental lifetime between the 'whole' (parent) and the 'part' (child). Composition specifies that the 'whole' is responsible for the disposition of 'parts', i.e. the 'whole' must manage the creation and destruction of its 'parts'. For illustration purpose, consider the relationship between University and Department. A university can have multiple departments but, there is no independent life of departments, i.e., if the university closes, the departments will no longer exist. Moreover, a department can not be a part of more than one university. Similarly, the relationship between House and Room represents a composition. A house can contain multiple rooms but, there is no independent life of room and any room can not belongs to two different houses. If we delete the house, rooms will be automatically deleted. Similarly, a polygon is a composition of several lines. If the polygon is destroyed, so are the lines.