

The Enhanced Entity–Relationship (EER) Model

The ER modeling concepts discussed in Chapter 3 are sufficient for representing many database schemas for *traditional* database applications, which include many data-processing applications in business and industry. Since the late 1970s, however, designers of database applications have tried to design more accurate database schemas that reflect the data properties and constraints more precisely. This was particularly important for newer applications of database technology, such as databases for engineering design and manufacturing (CAD/CAM),¹ telecommunications, complex software systems, and geographic information systems (GISs), among many other applications. These types of databases have requirements that are more complex than the more traditional applications. This led to the development of additional *semantic data modeling* concepts that were incorporated into conceptual data models such as the ER model. Various semantic data models have been proposed in the literature. Many of these concepts were also developed independently in related areas of computer science, such as the **knowledge representation** area of artificial intelligence and the **object modeling** area in software engineering.

In this chapter, we describe features that have been proposed for semantic data models and show how the ER model can be enhanced to include these concepts, which leads to the **enhanced ER (EER)** model.² We start in Section 4.1 by incorporating the concepts of *class/subclass relationships* and *type inheritance* into the ER model. Then, in Section 4.2, we add the concepts of *specialization* and *generalization*. Section 4.3 discusses the various types of *constraints* on specialization/generalization, and Section 4.4 shows how the UNION construct can be modeled by including the

¹CAD/CAM stands for computer-aided design/computer-aided manufacturing.

²EER has also been used to stand for *extended* ER model.

concept of *category* in the EER model. Section 4.5 gives a sample UNIVERSITY database schema in the EER model and summarizes the EER model concepts by giving formal definitions. We will use the terms *object* and *entity* interchangeably in this chapter, because many of these concepts are commonly used in object-oriented models.

We present the UML class diagram notation for representing specialization and generalization in Section 4.6, and we briefly compare these with EER notation and concepts. This serves as an example of alternative notation, and is a continuation of Section 3.8, which presented basic UML class diagram notation that corresponds to the basic ER model. In Section 4.7, we discuss the fundamental abstractions that are used as the basis of many semantic data models. Section 4.8 summarizes the chapter.

For a detailed introduction to conceptual modeling, Chapter 4 should be considered a continuation of Chapter 3. However, if only a basic introduction to ER modeling is desired, this chapter may be omitted. Alternatively, the reader may choose to skip some or all of the later sections of this chapter (Sections 4.4 through 4.8).

4.1 Subclasses, Superclasses, and Inheritance

The EER model includes *all the modeling concepts of the ER model* that were presented in Chapter 3. In addition, it includes the concepts of **subclass** and **superclass** and the related concepts of **specialization** and **generalization** (see Sections 4.2 and 4.3). Another concept included in the EER model is that of a **category** or **union type** (see Section 4.4), which is used to represent a collection of objects (entities) that is the *union* of objects of different entity types. Associated with these concepts is the important mechanism of **attribute and relationship inheritance**. Unfortunately, no standard terminology exists for these concepts, so we use the most common terminology. Alternative terminology is given in footnotes. We also describe a diagrammatic technique for displaying these concepts when they arise in an EER schema. We call the resulting schema diagrams **enhanced ER** or **EER diagrams**.

The first enhanced ER (EER) model concept we take up is that of a **subtype** or **subclass** of an entity type. As we discussed in Chapter 3, the name of an entity type is used to represent both a *type of entity* and the *entity set* or *collection of entities of that type* that exist in the database. For example, the entity type EMPLOYEE describes the type (that is, the attributes and relationships) of each employee entity, and also refers to the current set of EMPLOYEE entities in the COMPANY database. In many cases an entity type has numerous subgroupings or subtypes of its entities that are meaningful and need to be represented explicitly because of their significance to the database application. For example, the entities that are members of the EMPLOYEE entity type may be distinguished further into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE, and so on. The set or collection of entities in each of the latter groupings is a subset of the entities that belong to the EMPLOYEE entity set, meaning that every entity that is a member of one of these subgroupings is also an employee. We call each of these subgroupings a

subclass or **subtype** of the EMPLOYEE entity type, and the EMPLOYEE entity type is called the **superclass** or **supertype** for each of these subclasses. Figure 4.1 shows how to represent these concepts diagrammatically in EER diagrams. (The circle notation in Figure 4.1 will be explained in Section 4.2.)

We call the relationship between a superclass and any one of its subclasses a **superclass/subclass** or **supertype/subtype** or simply **class/subclass relationship**.³ In our previous example, EMPLOYEE/SECRETARY and EMPLOYEE/TECHNICIAN are two class/subclass relationships. Notice that a member entity of the subclass represents the *same real-world entity* as some member of the superclass; for example, a SECRETARY entity ‘Joan Logano’ is also the EMPLOYEE ‘Joan Logano.’ Hence, the subclass member is the same as the entity in the superclass, but in a distinct *specific role*. When we implement a superclass/subclass relationship in the database system, however, we may represent a member of the subclass as a distinct database object—say, a distinct record that is related via the key attribute to its superclass entity. In Section 9.2, we discuss various options for representing superclass/subclass relationships in relational databases.

An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass. Such an entity can be included optionally

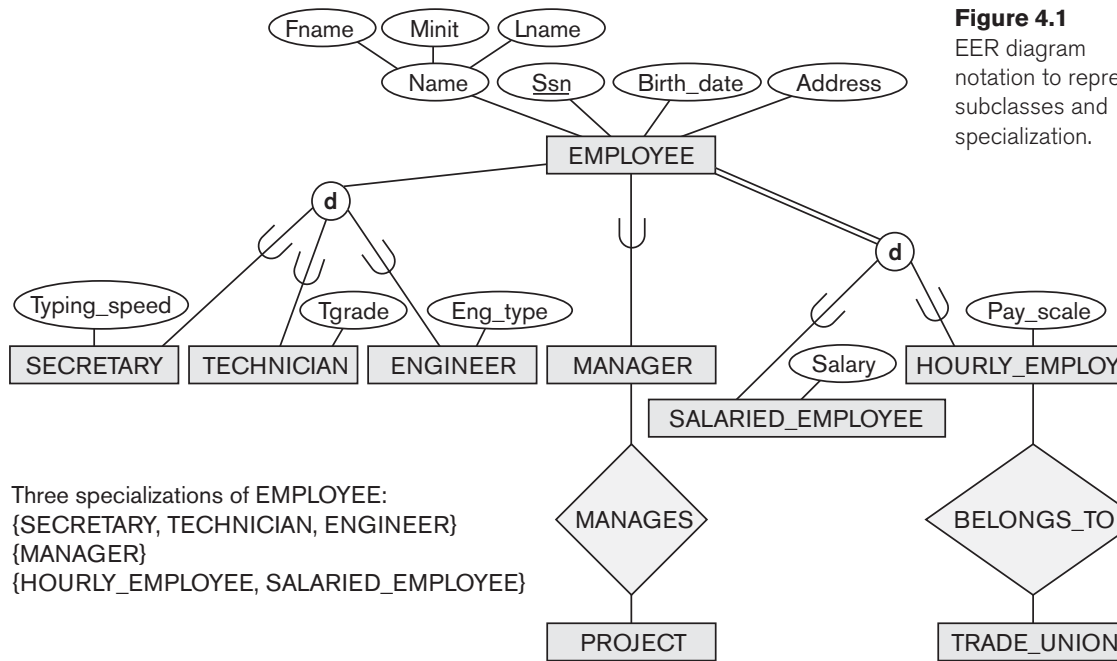


Figure 4.1
EER diagram notation to represent subclasses and specialization.

³A class/subclass relationship is often called an **IS-A** (or **IS-AN**) **relationship** because of the way we refer to the concept. We say a SECRETARY *is an* EMPLOYEE, a TECHNICIAN *is an* EMPLOYEE, and so on.

as a member of any number of subclasses. For example, a salaried employee who is also an engineer belongs to the two subclasses ENGINEER and SALARIED_EMPLOYEE of the EMPLOYEE entity type. However, it is not necessary that every entity in a superclass is a member of some subclass.

An important concept associated with subclasses (subtypes) is that of **type inheritance**. Recall that the *type* of an entity is defined by the attributes it possesses and the relationship types in which it participates. Because an entity in the subclass represents the same real-world entity from the superclass, it should possess values for its specific attributes *as well as* values of its attributes as a member of the superclass. We say that an entity that is a member of a subclass **inherits** all the attributes of the entity as a member of the superclass. The entity also inherits all the relationships in which the superclass participates. Notice that a subclass, with its own specific (or local) attributes and relationships together with all the attributes and relationships it inherits from the superclass, can be considered an *entity type* in its own right.⁴

4.2 Specialization and Generalization

4.2.1 Specialization

Specialization is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass. For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee. We may have several specializations of the same entity type based on different distinguishing characteristics. For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}; this specialization distinguishes among employees based on the *method of pay*.

Figure 4.1 shows how we represent a specialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by lines to a circle that represents the specialization, which is connected in turn to the superclass. The *subset symbol* on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.⁵ Attributes that apply only to entities of a particular subclass—such as TypingSpeed of SECRETARY—are attached to the rectangle representing that subclass. These are called **specific** (or **local**) **attributes** of the subclass. Similarly, a subclass can participate in **specific relationship types**, such as the HOURLY_EMPLOYEE subclass participating in the BELONGS_TO

⁴In some object-oriented programming languages, a common restriction is that an entity (or object) has *only one type*. This is generally too restrictive for conceptual database modeling.

⁵There are many alternative notations for specialization; we present the UML notation in Section 4.6 and other proposed notations in Appendix A.

relationship in Figure 4.1. We will explain the **d** symbol in the circles in Figure 4.1 and additional EER diagram notation shortly.

Figure 4.2 shows a few entity instances that belong to subclasses of the {SECRETARY, ENGINEER, TECHNICIAN} specialization. Again, notice that an entity that belongs to a subclass represents *the same real-world entity* as the entity connected to it in the EMPLOYEE superclass, even though the same entity is shown twice; for example, e_1 is shown in both EMPLOYEE and SECRETARY in Figure 4.2. As the figure suggests, a superclass/subclass relationship such as EMPLOYEE/SECRETARY somewhat resembles a 1:1 relationship *at the instance level* (see Figure 3.12). The main difference is that in a 1:1 relationship two *distinct entities* are related, whereas in a superclass/subclass relationship the entity in the subclass is the same real-world entity as the entity in the superclass but is playing a *specialized role*—for example, an EMPLOYEE specialized in the role of SECRETARY, or an EMPLOYEE specialized in the role of TECHNICIAN.

There are two main reasons for including class/subclass relationships and specializations. The first is that certain attributes may apply to some but not all entities of

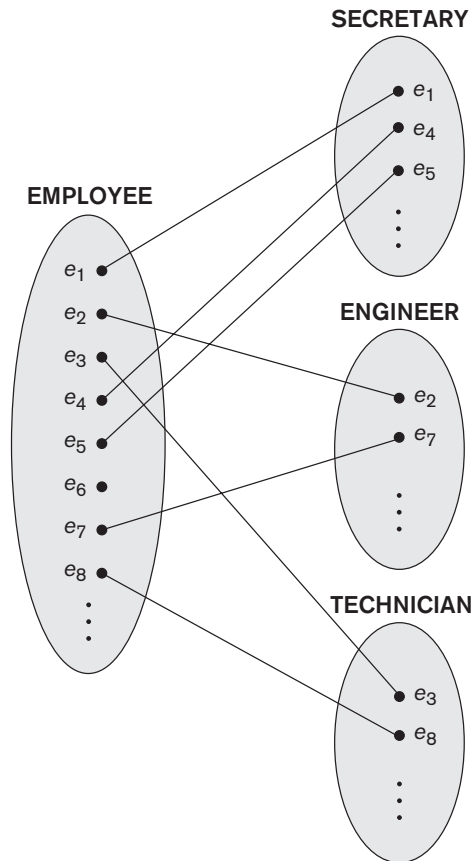


Figure 4.2

Instances of a specialization

the superclass entity type. A subclass is defined in order to group the entities to which these attributes apply. The members of the subclass may still share the majority of their attributes with the other members of the superclass. For example, in Figure 4.1 the SECRETARY subclass has the specific attribute Typing_speed, whereas the ENGINEER subclass has the specific attribute Eng_type, but SECRETARY and ENGINEER share their other inherited attributes from the EMPLOYEE entity type.

The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass. For example, if only HOURLY_EMPLOYEES can belong to a trade union, we can represent that fact by creating the subclass HOURLY_EMPLOYEE of EMPLOYEE and relating the subclass to an entity type TRADE_UNION via the BELONGS_TO relationship type, as illustrated in Figure 4.1.

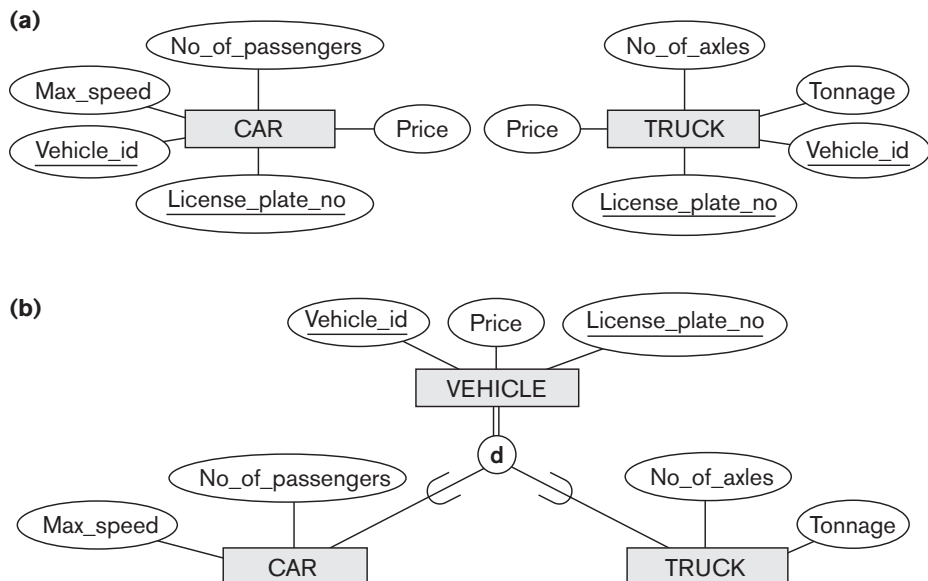
4.2.2 Generalization

We can think of a *reverse process* of abstraction in which we suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass** of which the original entity types are special **subclasses**. For example, consider the entity types CAR and TRUCK shown in Figure 4.3(a). Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure 4.3(b). Both CAR and TRUCK are now subclasses of the

Figure 4.3

Generalization. (a) Two entity types, CAR and TRUCK.

(b) Generalizing CAR and TRUCK into the superclass VEHICLE.



generalized superclass VEHICLE. We use the term **generalization** to refer to the process of defining a generalized entity type from the given entity types.

Notice that the generalization process can be viewed as being functionally the inverse of the specialization process; we can view {CAR, TRUCK} as a specialization of VEHICLE rather than viewing VEHICLE as a generalization of CAR and TRUCK. A diagrammatic notation to distinguish between generalization and specialization is used in some design methodologies. An arrow pointing to the generalized superclass represents a generalization process, whereas arrows pointing to the specialized subclasses represent a specialization process. We will *not* use this notation because the decision as to which process was followed in a particular situation is often subjective.

So far we have introduced the concepts of subclasses and superclass/subclass relationships, as well as the specialization and generalization processes. In general, a superclass or subclass represents a collection of entities of the same type and hence also describes an *entity type*; that is why superclasses and subclasses are all shown in rectangles in EER diagrams, like entity types.

4.3 Constraints and Characteristics of Specialization and Generalization Hierarchies

First, we discuss constraints that apply to a single specialization or a single generalization. For brevity, our discussion refers only to *specialization* even though it applies to *both* specialization and generalization. Then, we discuss differences between specialization/generalization *lattices* (*multiple inheritance*) and *hierarchies* (*single inheritance*), and we elaborate on the differences between the specialization and generalization processes during conceptual database schema design.

4.3.1 Constraints on Specialization and Generalization

In general, we may have several specializations defined on the same entity type (or superclass), as shown in Figure 4.1. In such a case, entities may belong to subclasses in each of the specializations. A specialization may also consist of a *single* subclass only, such as the {MANAGER} specialization in Figure 4.1; in such a case, we do not use the circle notation.

In some specializations we can determine exactly the entities that will become members of each subclass by placing a condition on the value of some attribute of the superclass. Such subclasses are called **predicate-defined** (or **condition-defined**) **subclasses**. For example, if the EMPLOYEE entity type has an attribute *Job_type*, as shown in Figure 4.4, we can specify the condition of membership in the SECRETARY subclass by the condition (*Job_type* = 'Secretary'), which we call the **defining predicate** of the subclass. This condition is a *constraint* specifying that exactly those entities of the EMPLOYEE entity type whose attribute value for *Job_type*

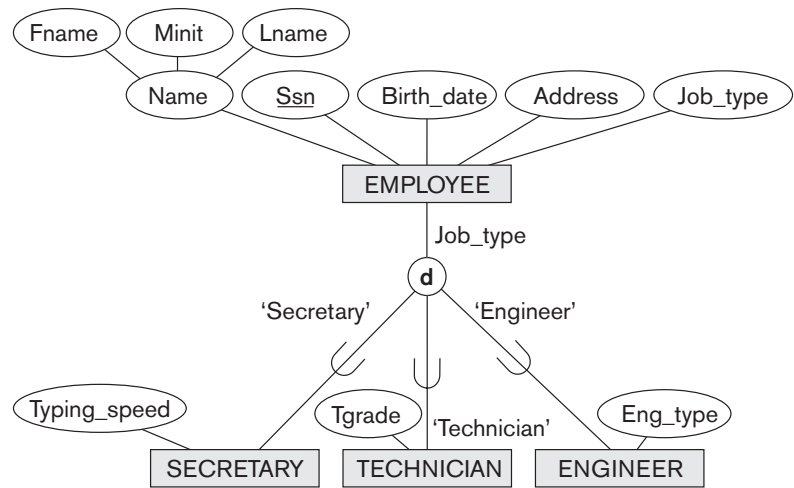


Figure 4.4
EER diagram notation
for an attribute-defined
specialization on
Job_type.

is ‘Secretary’ belong to the subclass. We display a predicate-defined subclass by writing the predicate condition next to the line that connects the subclass to the specialization circle.

If *all* subclasses in a specialization have their membership condition on the *same* attribute of the superclass, the specialization itself is called an **attribute-defined specialization**, and the attribute is called the **defining attribute** of the specialization.⁶ In this case, all the entities with the same value for the attribute belong to the same subclass. We display an attribute-defined specialization by placing the defining attribute name next to the arc from the circle to the superclass, as shown in Figure 4.4.

When we do not have a condition for determining membership in a subclass, the subclass is called **user-defined**. Membership in such a subclass is determined by the database users when they apply the operation to add an entity to the subclass; hence, membership is *specified individually for each entity by the user*, not by any condition that may be evaluated automatically.

Two other constraints may apply to a specialization. The first is the **disjointness constraint**, which specifies that the subclasses of the specialization must be disjoint sets. This means that an entity can be a member of *at most* one of the subclasses of the specialization. A specialization that is attribute-defined implies the disjointness constraint (if the attribute used to define the membership predicate is single-valued). Figure 4.4 illustrates this case, where the **d** in the circle stands for *disjoint*. The **d** notation also applies to user-defined subclasses of a specialization that must be disjoint, as illustrated by the specialization {HOURLY_EMPLOYEE, SALARIED_EMPLOYEE} in Figure 4.1. If the subclasses are not constrained to be disjoint, their sets of entities

⁶Such an attribute is called a *discriminator* or *discriminating attribute* in UML terminology.

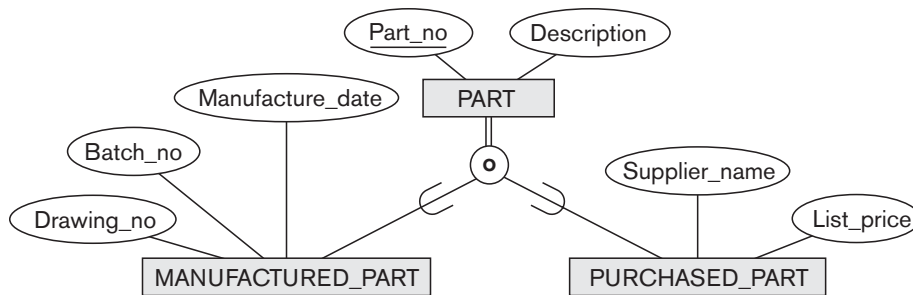


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

may be **overlapping**; that is, the same (real-world) entity may be a member of more than one subclass of the specialization. This case, which is the default, is displayed by placing an **o** in the circle, as shown in Figure 4.5.

The second constraint on specialization is called the **completeness** (or **totalness**) **constraint**, which may be total or partial. A **total specialization** constraint specifies that *every* entity in the superclass must be a member of at least one subclass in the specialization. For example, if every EMPLOYEE must be either an HOURLY_EMPLOYEE or a SALARIED_EMPLOYEE, then the specialization {HOURLY_EMPLOYEE, SALARIED_EMPLOYEE} in Figure 4.1 is a total specialization of EMPLOYEE. This is shown in EER diagrams by using a double line to connect the superclass to the circle. A single line is used to display a **partial specialization**, which allows an entity not to belong to any of the subclasses. For example, if some EMPLOYEE entities do not belong to any of the subclasses {SECRETARY, ENGINEER, TECHNICIAN} in Figures 4.1 and 4.4, then that specialization is partial.⁷

Notice that the disjointness and completeness constraints are *independent*. Hence, we have the following four possible constraints on a specialization:

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

Of course, the correct constraint is determined from the real-world meaning that applies to each specialization. In general, a superclass that was identified through the *generalization* process usually is **total**, because the superclass is *derived from* the subclasses and hence contains only the entities that are in the subclasses.

Certain insertion and deletion rules apply to specialization (and generalization) as a consequence of the constraints specified earlier. Some of these rules are as follows:

- Deleting an entity from a superclass implies that it is automatically deleted from all the subclasses to which it belongs.

⁷The notation of using single or double lines is similar to that for partial or total participation of an entity type in a relationship type, as described in Chapter 3.

- Inserting an entity in a superclass implies that the entity is mandatorily inserted in all *predicate-defined* (or *attribute-defined*) subclasses for which the entity satisfies the defining predicate.
- Inserting an entity in a superclass of a *total specialization* implies that the entity is mandatorily inserted in at least one of the subclasses of the specialization.

The reader is encouraged to make a complete list of rules for insertions and deletions for the various types of specializations.

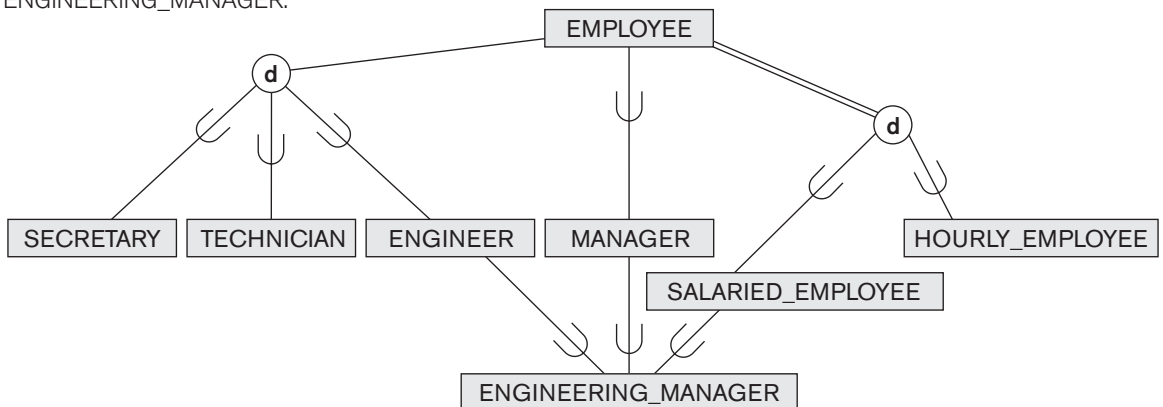
4.3.2 Specialization and Generalization Hierarchies and Lattices

A subclass itself may have further subclasses specified on it, forming a hierarchy or a lattice of specializations. For example, in Figure 4.6 ENGINEER is a subclass of EMPLOYEE and is also a superclass of ENGINEERING_MANAGER; this represents the real-world constraint that every engineering manager is required to be an engineer. A **specialization hierarchy** has the constraint that every subclass participates *as a subclass* in *only one* class/subclass relationship; that is, each subclass has only one parent, which results in a **tree structure** or **strict hierarchy**. In contrast, for a **specialization lattice**, a subclass can be a subclass in *more than one* class/subclass relationship. Hence, Figure 4.6 is a lattice.

Figure 4.7 shows another specialization lattice of more than one level. This may be part of a conceptual schema for a UNIVERSITY database. Notice that this arrangement would have been a hierarchy except for the STUDENT_ASSISTANT subclass, which is a subclass in two distinct class/subclass relationships.

Figure 4.6

A specialization lattice with shared subclass ENGINEERING_MANAGER.



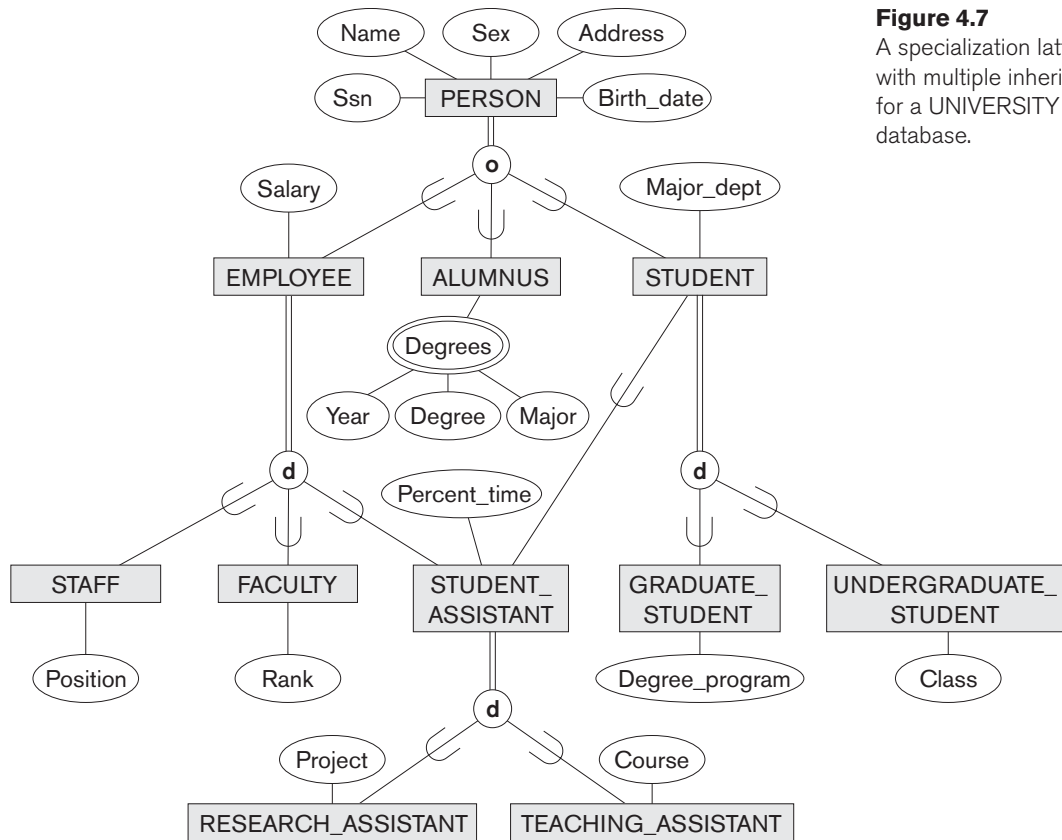


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

The requirements for the part of the UNIVERSITY database shown in Figure 4.7 are the following:

1. The database keeps track of three types of persons: employees, alumni, and students. A person can belong to one, two, or all three of these types. Each person has a name, SSN, sex, address, and birth date.
2. Every employee has a salary, and there are three types of employees: faculty, staff, and student assistants. Each employee belongs to exactly one of these types. For each alumnus, a record of the degree or degrees that he or she earned at the university is kept, including the name of the degree, the year granted, and the major department. Each student has a major department.
3. Each faculty has a rank, whereas each staff member has a staff position. Student assistants are classified further as either research assistants or teaching assistants, and the percent of time that they work is recorded in the database. Research assistants have their research project stored, whereas teaching assistants have the current course they work on.

4. Students are further classified as either graduate or undergraduate, with the specific attributes degree program (M.S., Ph.D., M.B.A., and so on) for graduate students and class (freshman, sophomore, and so on) for undergraduates.

In Figure 4.7, all person entities represented in the database are members of the PERSON entity type, which is specialized into the subclasses {EMPLOYEE, ALUMNUS, STUDENT}. This specialization is overlapping; for example, an alumnus may also be an employee and a student pursuing an advanced degree. The subclass STUDENT is the superclass for the specialization {GRADUATE_STUDENT, UNDERGRADUATE_STUDENT}, whereas EMPLOYEE is the superclass for the specialization {STUDENT_ASSISTANT, FACULTY, STAFF}. Notice that STUDENT_ASSISTANT is also a subclass of STUDENT. Finally, STUDENT_ASSISTANT is the superclass for the specialization into {RESEARCH_ASSISTANT, TEACHING_ASSISTANT}.

In such a specialization lattice or hierarchy, a subclass inherits the attributes not only of its direct superclass, but also of all its predecessor superclasses *all the way to the root* of the hierarchy or lattice if necessary. For example, an entity in GRADUATE_STUDENT inherits all the attributes of that entity as a STUDENT *and* as a PERSON. Notice that an entity may exist in several *leaf nodes* of the hierarchy, where a **leaf node** is a class that has *no subclasses of its own*. For example, a member of GRADUATE_STUDENT may also be a member of RESEARCH_ASSISTANT.

A subclass with *more than one* superclass is called a **shared subclass**, such as ENGINEERING_MANAGER in Figure 4.6. This leads to the concept known as **multiple inheritance**, where the shared subclass ENGINEERING_MANAGER directly inherits attributes and relationships from multiple superclasses. Notice that the existence of at least one shared subclass leads to a lattice (and hence to *multiple inheritance*); if no shared subclasses existed, we would have a hierarchy rather than a lattice and only **single inheritance** would exist. An important rule related to multiple inheritance can be illustrated by the example of the shared subclass STUDENT_ASSISTANT in Figure 4.7, which inherits attributes from both EMPLOYEE and STUDENT. Here, both EMPLOYEE and STUDENT inherit *the same attributes* from PERSON. The rule states that if an attribute (or relationship) originating in the *same superclass* (PERSON) is inherited more than once via different paths (EMPLOYEE and STUDENT) in the lattice, then it should be included only once in the shared subclass (STUDENT_ASSISTANT). Hence, the attributes of PERSON are inherited *only once* in the STUDENT_ASSISTANT subclass in Figure 4.7.

It is important to note here that some models and languages are limited to **single inheritance** and *do not allow* multiple inheritance (shared subclasses). It is also important to note that some models do not allow an entity to have multiple types, and hence an entity can be a member of *only one leaf class*.⁸ In such a model, it is necessary to create additional subclasses as leaf nodes to cover all

⁸In some models, the class is further restricted to be a *leaf node* in the hierarchy or lattice.

possible combinations of classes that may have some entity that belongs to all these classes simultaneously. For example, in the overlapping specialization of PERSON into {EMPLOYEE, ALUMNUS, STUDENT} (or {E, A, S} for short), it would be necessary to create seven subclasses of PERSON in order to cover all possible types of entities: E, A, S, E_A, E_S, A_S, and E_A_S. Obviously, this can lead to extra complexity.

Although we have used specialization to illustrate our discussion, similar concepts *apply equally* to generalization, as we mentioned at the beginning of this section. Hence, we can also speak of **generalization hierarchies** and **generalization lattices**.

4.3.3 Utilizing Specialization and Generalization in Refining Conceptual Schemas

Now we elaborate on the differences between the specialization and generalization processes and how they are used to refine conceptual schemas during conceptual database design. In the specialization process, the database designers typically start with an entity type and then define subclasses of the entity type by successive specialization; that is, they repeatedly define more specific groupings of the entity type. For example, when designing the specialization lattice in Figure 4.7, we may first specify an entity type PERSON for a university database. Then we discover that three types of persons will be represented in the database: university employees, alumni, and students and we create the specialization {EMPLOYEE, ALUMNUS, STUDENT}. The overlapping constraint is chosen because a person may belong to more than one of the subclasses. We specialize EMPLOYEE further into {STAFF, FACULTY, STUDENT_ASSISTANT}, and specialize STUDENT into {GRADUATE_STUDENT, UNDERGRADUATE_STUDENT}. Finally, we specialize STUDENT_ASSISTANT into {RESEARCH_ASSISTANT, TEACHING_ASSISTANT}. This process is called **top-down conceptual refinement**. So far, we have a hierarchy; then we realize that STUDENT_ASSISTANT is a shared subclass, since it is also a subclass of STUDENT, leading to the lattice.

It is possible to arrive at the same hierarchy or lattice from the other direction. In such a case, the process involves generalization rather than specialization and corresponds to a **bottom-up conceptual synthesis**. For example, the database designers may first discover entity types such as STAFF, FACULTY, ALUMNUS, GRADUATE_STUDENT, UNDERGRADUATE_STUDENT, RESEARCH_ASSISTANT, TEACHING_ASSISTANT, and so on; then they generalize {GRADUATE_STUDENT, UNDERGRADUATE_STUDENT} into STUDENT; then {RESEARCH_ASSISTANT, TEACHING_ASSISTANT} into STUDENT_ASSISTANT; then {STAFF, FACULTY, STUDENT_ASSISTANT} into EMPLOYEE; and finally {EMPLOYEE, ALUMNUS, STUDENT} into PERSON.

The final design of hierarchies or lattices resulting from either process may be identical; the only difference relates to the manner or order in which the schema superclasses and subclasses were created during the design process. In practice, it is likely that a combination of the two processes is employed. Notice that the

notion of representing data and knowledge by using superclass/subclass hierarchies and lattices is quite common in knowledge-based systems and expert systems, which combine database technology with artificial intelligence techniques. For example, frame-based knowledge representation schemes closely resemble class hierarchies. Specialization is also common in software engineering design methodologies that are based on the object-oriented paradigm.

4.4 Modeling of UNION Types Using Categories

It is sometimes necessary to represent a collection of entities from different entity types. In this case, a subclass will represent a collection of entities that is a subset of the UNION of entities from distinct entity types; we call such a *subclass* a **union type** or a **category**.⁹

For example, suppose that we have three entity types: PERSON, BANK, and COMPANY. In a database for motor vehicle registration, an owner of a vehicle can be a person, a bank (holding a lien on a vehicle), or a company. We need to create a class (collection of entities) that includes entities of all three types to play the role of *vehicle owner*. A category (union type) OWNER that is a *subclass of the UNION* of the three entity sets of COMPANY, BANK, and PERSON can be created for this purpose. We display categories in an EER diagram as shown in Figure 4.8. The superclasses COMPANY, BANK, and PERSON are connected to the circle with the \cup symbol, which stands for the *set union operation*. An arc with the subset symbol connects the circle to the (subclass) OWNER category. In Figure 4.8 we have two categories: OWNER, which is a subclass (subset) of the union of PERSON, BANK, and COMPANY; and REGISTERED_VEHICLE, which is a subclass (subset) of the union of CAR and TRUCK.

A category has two or more superclasses that may represent collections of entities from *distinct entity types*, whereas other superclass/subclass relationships always have a single superclass. To better understand the difference, we can compare a category, such as OWNER in Figure 4.8, with the ENGINEERING_MANAGER shared subclass in Figure 4.6. The latter is a subclass of *each of* the three superclasses ENGINEER, MANAGER, and SALARIED_EMPLOYEE, so an entity that is a member of ENGINEERING_MANAGER must exist in *all three collections*. This represents the constraint that an engineering manager must be an ENGINEER, a MANAGER, *and* a SALARIED_EMPLOYEE; that is, the ENGINEERING_MANAGER entity set is a subset of the *intersection* of the three entity sets. On the other hand, a category is a subset of the *union* of its superclasses. Hence, an entity that is a member of OWNER must exist in *only one* of the superclasses. This represents the constraint that an OWNER may be a COMPANY, a BANK, *or* a PERSON in Figure 4.8.

⁹Our use of the term *category* is based on the ECR (entity–category–relationship) model (Elmasri et al., 1985).

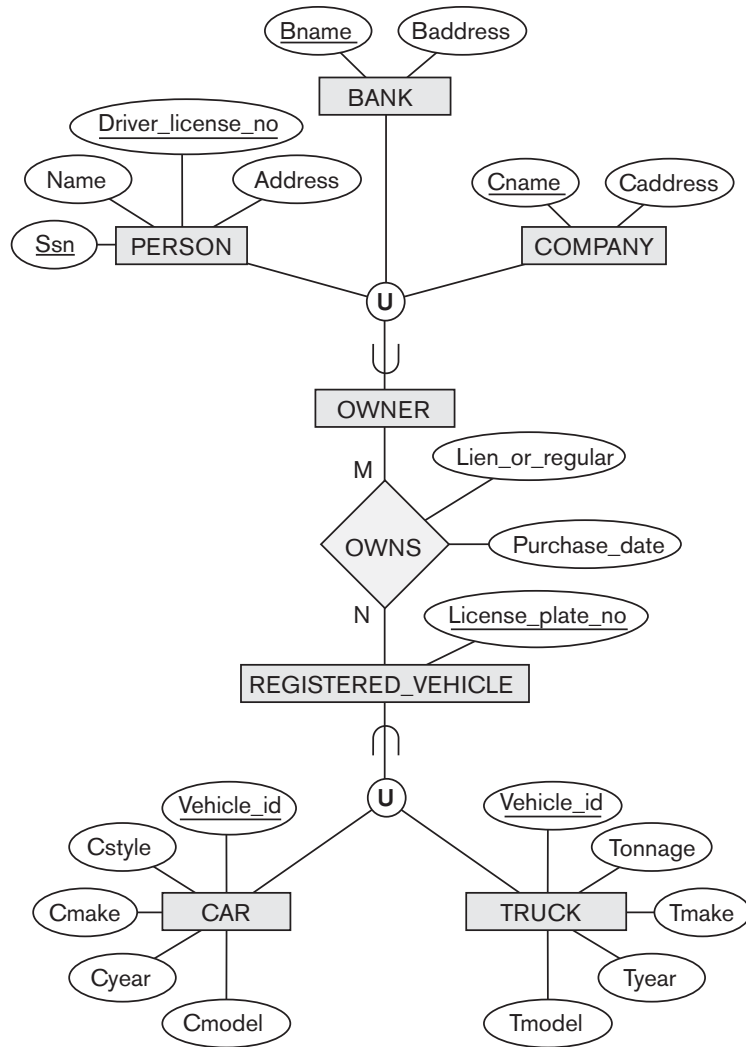


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE

Attribute inheritance works more selectively in the case of categories. For example, in Figure 4.8 each OWNER entity inherits the attributes of a COMPANY, a PERSON, or a BANK, depending on the superclass to which the entity belongs. On the other hand, a shared subclass such as ENGINEERING_MANAGER (Figure 4.6) inherits *all* the attributes of its superclasses SALARIED_EMPLOYEE, ENGINEER, and MANAGER.

It is interesting to note the difference between the category REGISTERED_VEHICLE (Figure 4.8) and the generalized superclass VEHICLE (Figure 4.3(b)). In Figure 4.3(b), every car and every truck is a VEHICLE; but in Figure 4.8, the REGISTERED_VEHICLE category includes some cars and some trucks but not necessarily

all of them (for example, some cars or trucks may not be registered). In general, a specialization or generalization such as that in Figure 4.3(b), if it were *partial*, would not preclude VEHICLE from containing other types of entities, such as motorcycles. However, a category such as REGISTERED_VEHICLE in Figure 4.8 implies that only cars and trucks, but not other types of entities, can be members of REGISTERED_VEHICLE.

A category can be **total** or **partial**. A total category holds the *union* of all entities in its superclasses, whereas a partial category can hold a *subset of the union*. A total category is represented diagrammatically by a double line connecting the category and the circle, whereas a partial category is indicated by a single line.

The superclasses of a category may have different key attributes, as demonstrated by the OWNER category in Figure 4.8, or they may have the same key attribute, as demonstrated by the REGISTERED_VEHICLE category. Notice that if a category is total (not partial), it may be represented alternatively as a total specialization (or a total generalization). In this case, the choice of which representation to use is subjective. If the two classes represent the same type of entities and share numerous attributes, including the same key attributes, specialization/generalization is preferred; otherwise, categorization (union type) is more appropriate.

It is important to note that some modeling methodologies do not have union types. In these models, a union type must be represented in a roundabout way (see Section 9.2).

4.5 A Sample UNIVERSITY EER Schema, Design Choices, and Formal Definitions

In this section, we first give an example of a database schema in the EER model to illustrate the use of the various concepts discussed here and in Chapter 3. Then, we discuss design choices for conceptual schemas, and finally we summarize the EER model concepts and define them formally in the same manner in which we formally defined the concepts of the basic ER model in Chapter 3.

4.5.1 A Different UNIVERSITY Database Example

Consider a UNIVERSITY database that has *different requirements* from the UNIVERSITY database presented in Section 3.10. This database keeps track of students and their majors, transcripts, and registration as well as of the university's course offerings. The database also keeps track of the sponsored research projects of faculty and graduate students. This schema is shown in Figure 4.9. A discussion of the requirements that led to this schema follows.

For each person, the database maintains information on the person's Name [Name], Social Security number [Ssn], address [Address], sex [Sex], and birth date [Bdate]. Two subclasses of the PERSON entity type are identified: FACULTY and STUDENT. Specific attributes of FACULTY are rank [Rank] (assistant, associate, adjunct, research,

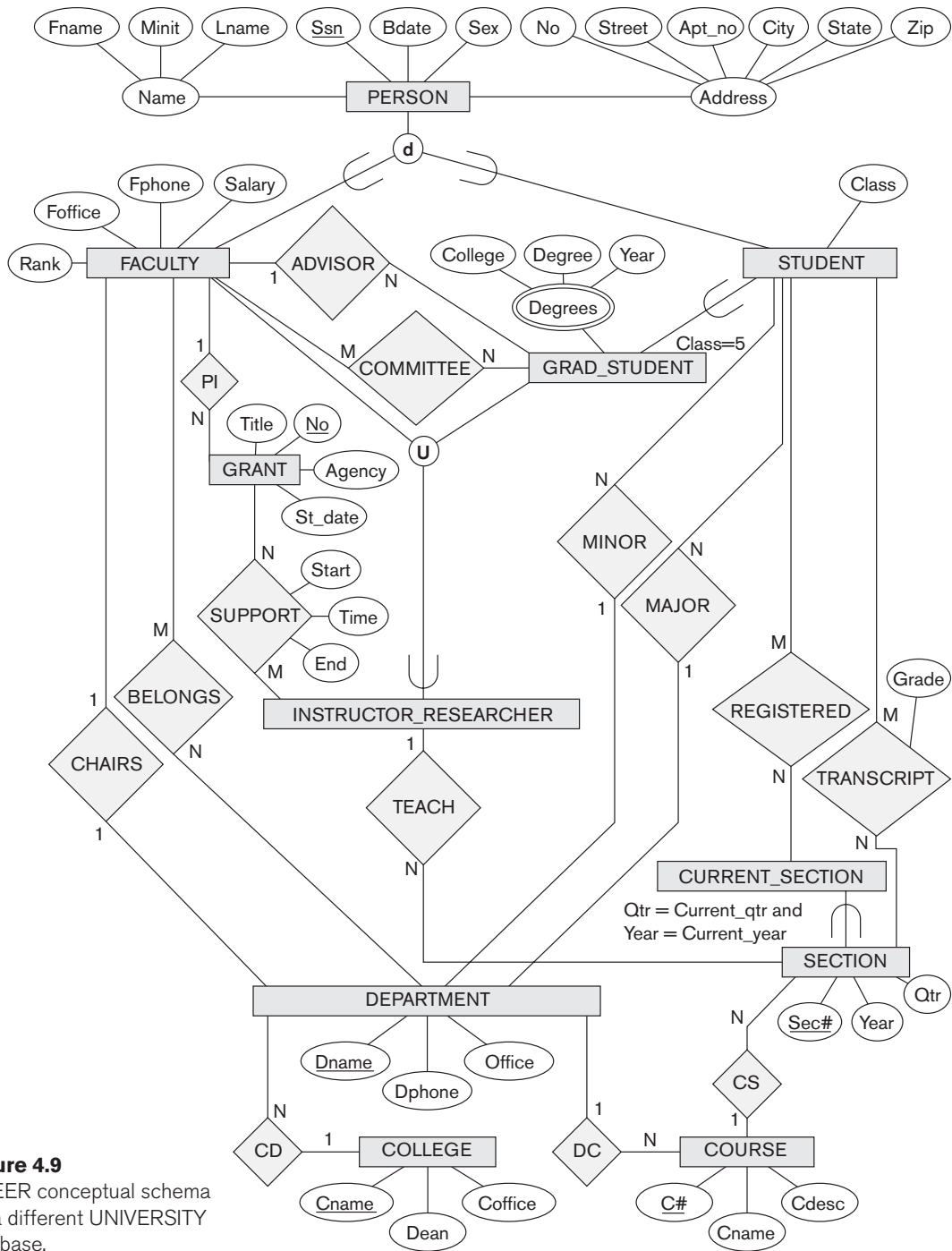


Figure 4.9
An EER conceptual schema for a different UNIVERSITY database.

visiting, and so on), office [Foffice], office phone [Fphone], and salary [Salary]. All faculty members are related to the academic department(s) with which they are affiliated [BELONGS] (a faculty member can be associated with several departments, so the relationship is M:N). A specific attribute of STUDENT is [Class] (freshman = 1, sophomore = 2, . . . , MS student = 5, PhD student = 6). Each STUDENT is also related to his or her major and minor departments (if known) [MAJOR] and [MINOR], to the course sections he or she is currently attending [REGISTERED], and to the courses completed [TRANSCRIPT]. Each TRANSCRIPT instance includes the grade the student received [Grade] in a section of a course.

GRAD_STUDENT is a subclass of STUDENT, with the defining predicate (Class = 5 OR Class = 6). For each graduate student, we keep a list of previous degrees in a composite, multivalued attribute [Degrees]. We also relate the graduate student to a faculty advisor [ADVISOR] and to a thesis committee [COMMITTEE], if one exists.

An academic department has the attributes name [Dname], telephone [Dphone], and office number [Office] and is related to the faculty member who is its chairperson [CHAIRS] and to the college to which it belongs [CD]. Each college has attributes college name [Cname], office number [Coffice], and the name of its dean [Dean].

A course has attributes course number [C#], course name [Cname], and course description [Cdesc]. Several sections of each course are offered, with each section having the attributes section number [Sec#] and the year and quarter in which the section was offered ([Year] and [Qtr]).¹⁰ Section numbers uniquely identify each section. The sections being offered during the current quarter are in a subclass CURRENT_SECTION of SECTION, with the defining predicate Qtr = Current_qtr and Year = Current_year. Each section is related to the instructor who taught or is teaching it ([TEACH]), if that instructor is in the database.

The category INSTRUCTOR_RESEARCHER is a subset of the union of FACULTY and GRAD_STUDENT and includes all faculty, as well as graduate students who are supported by teaching or research. Finally, the entity type GRANT keeps track of research grants and contracts awarded to the university. Each grant has attributes grant title [Title], grant number [No], the awarding agency [Agency], and the starting date [St_date]. A grant is related to one principal investigator [PI] and to all researchers it supports [SUPPORT]. Each instance of support has as attributes the starting date of support [Start], the ending date of the support (if known) [End], and the percentage of time being spent on the project [Time] by the researcher being supported.

4.5.2 Design Choices for Specialization/Generalization

It is not always easy to choose the most appropriate conceptual design for a database application. In Section 3.7.3, we presented some of the typical issues that confront a database designer when choosing among the concepts of entity

¹⁰We assume that the *quarter* system rather than the *semester* system is used in this university.

types, relationship types, and attributes to represent a particular miniworld situation as an ER schema. In this section, we discuss design guidelines and choices for the EER concepts of specialization/generalization and categories (union types).

As we mentioned in Section 3.7.3, conceptual database design should be considered as an iterative refinement process until the most suitable design is reached. The following guidelines can help to guide the design process for EER concepts:

- In general, many specializations and subclasses can be defined to make the conceptual model accurate. However, the drawback is that the design becomes quite cluttered. It is important to represent only those subclasses that are deemed necessary to avoid extreme cluttering of the conceptual schema.
- If a subclass has few specific (local) attributes and no specific relationships, it can be merged into the superclass. The specific attributes would hold NULL values for entities that are not members of the subclass. A *type* attribute could specify whether an entity is a member of the subclass.
- Similarly, if all the subclasses of a specialization/generalization have few specific attributes and no specific relationships, they can be merged into the superclass and replaced with one or more *type* attributes that specify the subclass or subclasses that each entity belongs to (see Section 9.2 for how this criterion applies to relational databases).
- Union types and categories should generally be avoided unless the situation definitely warrants this type of construct, which does occur in some practical situations. If possible, we try to model using specialization/generalization as discussed at the end of Section 4.4.
- The choice of disjoint/overlapping and total/partial constraints on specialization/generalization is driven by the rules in the miniworld being modeled. If the requirements do not indicate any particular constraints, the default would generally be overlapping and partial, since this does not specify any restrictions on subclass membership.

As an example of applying these guidelines, consider Figure 4.6, where no specific (local) attributes are shown. We could merge all the subclasses into the EMPLOYEE entity type and add the following attributes to EMPLOYEE:

- An attribute *Job_type* whose value set {'Secretary', 'Engineer', 'Technician'} would indicate which subclass in the first specialization each employee belongs to.
- An attribute *Pay_method* whose value set {'Salaried', 'Hourly'} would indicate which subclass in the second specialization each employee belongs to.

- An attribute `is_a_manager` whose value set {'Yes', 'No'} would indicate whether an individual employee entity is a manager or not.

4.5.3 Formal Definitions for the EER Model Concepts

We now summarize the EER model concepts and give formal definitions. A **class**¹¹ defines a type of entity and represents a set or collection of entities of that type; this includes any of the EER schema constructs that correspond to collections of entities, such as entity types, subclasses, superclasses, and categories. A **subclass** S is a class whose entities must always be a subset of the entities in another class, called the **superclass** C of the **superclass/subclass** (or **IS-A**) **relationship**. We denote such a relationship by C/S . For such a superclass/subclass relationship, we must always have

$$S \subseteq C$$

A **specialization** $Z = \{S_1, S_2, \dots, S_n\}$ is a set of subclasses that have the same superclass G ; that is, G/S_i is a superclass/subclass relationship for $i = 1, 2, \dots, n$. G is called a **generalized entity type** (or the **superclass** of the specialization, or a **generalization** of the subclasses $\{S_1, S_2, \dots, S_n\}$). Z is said to be **total** if we always (at any point in time) have

$$\bigcup_{i=1}^n S_i = G$$

Otherwise, Z is said to be **partial**. Z is said to be **disjoint** if we always have

$$S_i \cap S_j = \emptyset \text{ (empty set) for } i \neq j$$

Otherwise, Z is said to be **overlapping**.

A subclass S of C is said to be **predicate-defined** if a predicate p on the attributes of C is used to specify which entities in C are members of S ; that is, $S = C[p]$, where $C[p]$ is the set of entities in C that satisfy p . A subclass that is not defined by a predicate is called **user-defined**.

A specialization Z (or generalization G) is said to be **attribute-defined** if a predicate ($A = c_i$), where A is an attribute of G and c_i is a constant value from the domain of A , is used to specify membership in each subclass S_i in Z . Notice that if $c_i \neq c_j$ for $i \neq j$, and A is a single-valued attribute, then the specialization will be disjoint.

A **category** T is a class that is a subset of the union of n defining superclasses D_1, D_2, \dots, D_n , $n > 1$ and is formally specified as follows:

$$T \subseteq (D_1 \cup D_2 \dots \cup D_n)$$

¹¹The use of the word *class* here refers to a collection (set) of entities, which differs from its more common use in object-oriented programming languages such as C++. In C++, a class is a structured type definition along with its applicable functions (operations).

A predicate p_i on the attributes of D_i can be used to specify the members of each D_i that are members of T . If a predicate is specified on every D_i , we get

$$T = (D_1[p_1] \cup D_2[p_2] \dots \cup D_n[p_n])$$

We should now extend the definition of **relationship type** given in Chapter 3 by allowing any class—not only any entity type—to participate in a relationship. Hence, we should replace the words *entity type* with *class* in that definition. The graphical notation of EER is consistent with ER because all classes are represented by rectangles.

4.6 Example of Other Notation: Representing Specialization and Generalization in UML Class Diagrams

We now discuss the UML notation for generalization/specialization and inheritance. We already presented basic UML class diagram notation and terminology in Section 3.8. Figure 4.10 illustrates a possible UML class diagram corresponding to the EER diagram in Figure 4.7. The basic notation for specialization/generalization (see Figure 4.10) is to connect the subclasses by vertical lines to a horizontal line, which has a triangle connecting the horizontal line through another vertical line to the superclass. A blank triangle indicates a specialization/generalization with the *disjoint* constraint, and a filled triangle indicates an *overlapping* constraint. The root superclass is called the **base class**, and the subclasses (leaf nodes) are called **leaf classes**.

The preceding discussion and the example in Figure 4.10, as well as the presentation in Section 3.8, gave a brief overview of UML class diagrams and terminology. We focused on the concepts that are relevant to ER and EER database modeling rather than on those concepts that are more relevant to software engineering. In UML, there are many details that we have not discussed because they are outside the scope of this text and are mainly relevant to software engineering. For example, classes can be of various types:

- Abstract classes define attributes and operations but do not have objects corresponding to those classes. These are mainly used to specify a set of attributes and operations that can be inherited.
- Concrete classes can have objects (entities) instantiated to belong to the class.
- Template classes specify a template that can be further used to define other classes.

In database design, we are mainly concerned with specifying concrete classes whose collections of objects are permanently (or persistently) stored in the database. The bibliographic notes at the end of this chapter give some references to books that describe complete details of UML.

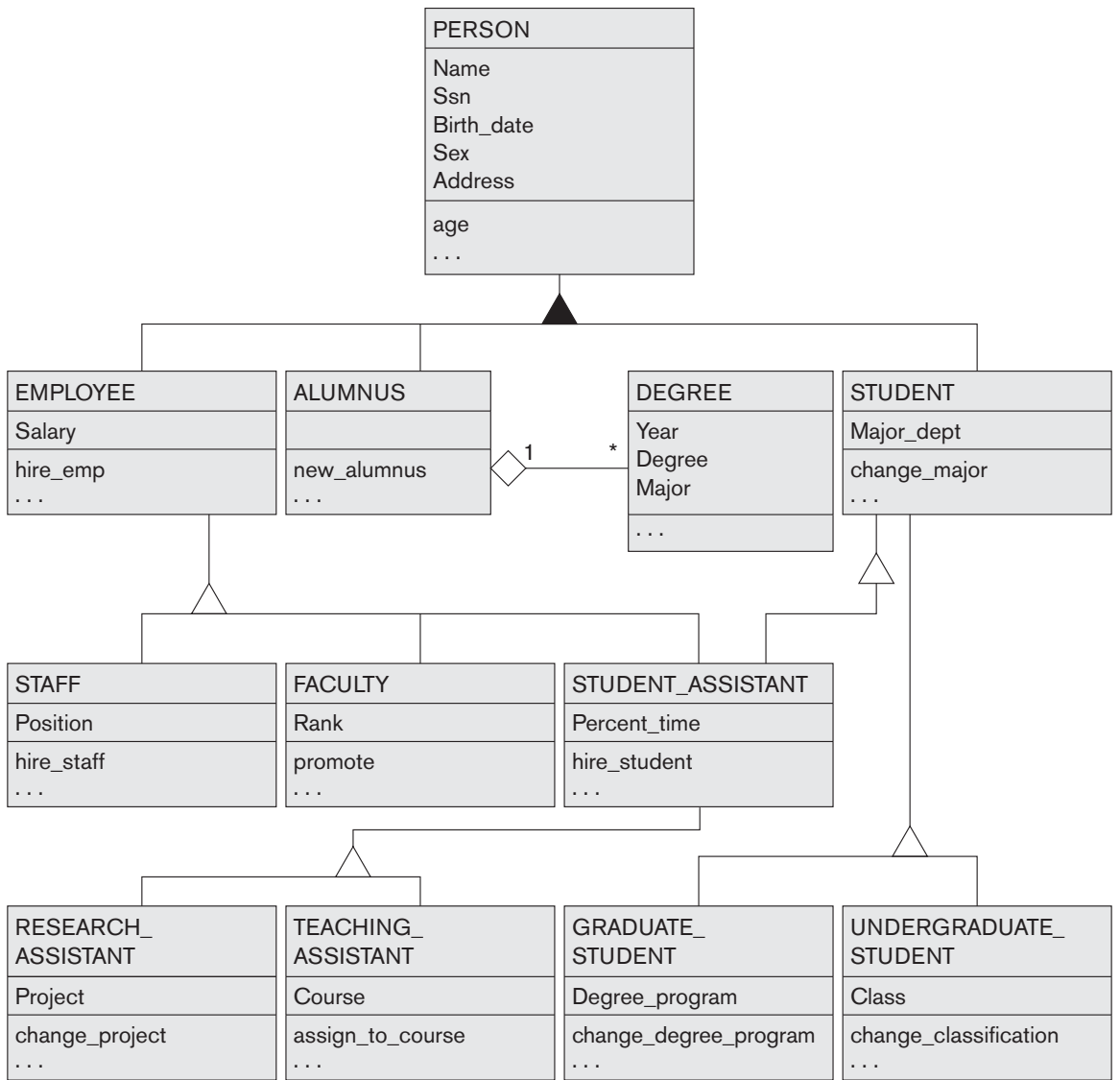


Figure 4.10

A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

4.7 Data Abstraction, Knowledge Representation, and Ontology Concepts

In this section, we discuss in general terms some of the modeling concepts that we described quite specifically in our presentation of the ER and EER models in Chapter 3 and earlier in this chapter. This terminology is not only used in conceptual

data modeling but also in artificial intelligence literature when discussing **knowledge representation (KR)**. This section discusses the similarities and differences between conceptual modeling and knowledge representation, and introduces some of the alternative terminology and a few additional concepts.

The goal of KR techniques is to develop concepts for accurately modeling some **domain of knowledge** by creating an **ontology**¹² that describes the concepts of the domain and how these concepts are interrelated. The ontology is used to store and manipulate knowledge for drawing inferences, making decisions, or answering questions. The goals of KR are similar to those of semantic data models, but there are some important similarities and differences between the two disciplines:

- Both disciplines use an abstraction process to identify common properties and important aspects of objects in the miniworld (also known as *domain of discourse* in KR) while suppressing insignificant differences and unimportant details.
- Both disciplines provide concepts, relationships, constraints, operations, and languages for defining data and representing knowledge.
- KR is generally broader in scope than semantic data models. Different forms of knowledge, such as rules (used in inference, deduction, and search), incomplete and default knowledge, and temporal and spatial knowledge, are represented in KR schemes. Database models are being expanded to include some of these concepts (see Chapter 26).
- KR schemes include **reasoning mechanisms** that deduce additional facts from the facts stored in a database. Hence, whereas most current database systems are limited to answering direct queries, knowledge-based systems using KR schemes can answer queries that involve **inferences** over the stored data. Database technology is being extended with inference mechanisms (see Section 26.5).
- Whereas most data models concentrate on the representation of database schemas, or meta-knowledge, KR schemes often mix up the schemas with the instances themselves in order to provide flexibility in representing exceptions. This often results in inefficiencies when these KR schemes are implemented, especially when compared with databases and when a large amount of structured data (facts) needs to be stored.

We now discuss four **abstraction concepts** that are used in semantic data models, such as the EER model, as well as in KR schemes: (1) classification and instantiation, (2) identification, (3) specialization and generalization, and (4) aggregation and association. The paired concepts of classification and instantiation are inverses of one another, as are generalization and specialization. The concepts of aggregation and association are also related. We discuss these abstract concepts and their relation to the concrete representations used in the EER model to clarify the data abstraction process and to improve our understanding of the related process of conceptual schema design. We close the section with a brief discussion of *ontology*, which is being used widely in recent knowledge representation research.

¹²An *ontology* is somewhat similar to a conceptual schema, but with more knowledge, rules, and exceptions.

4.7.1 Classification and Instantiation

The process of **classification** involves systematically assigning similar objects/entities to object classes/entity types. We can now describe (in DB) or reason about (in KR) the classes rather than the individual objects. Collections of objects that share the same types of attributes, relationships, and constraints are classified into classes in order to simplify the process of discovering their properties. **Instantiation** is the inverse of classification and refers to the generation and specific examination of distinct objects of a class. An object instance is related to its object class by the **IS-AN-INSTANCE-OF** or **IS-A-MEMBER-OF** relationship. Although EER diagrams do not display instances, the UML diagrams allow a form of instantiation by permitting the display of individual objects. We *did not* describe this feature in our introduction to UML class diagrams.

In general, the objects of a class should have a similar type structure. However, some objects may display properties that differ in some respects from the other objects of the class; these **exception objects** also need to be modeled, and KR schemes allow more varied exceptions than do database models. In addition, certain properties apply to the class as a whole and not to the individual objects; KR schemes allow such **class properties**. UML diagrams also allow specification of class properties.

In the EER model, entities are classified into entity types according to their basic attributes and relationships. Entities are further classified into subclasses and categories based on additional similarities and differences (exceptions) among them. Relationship instances are classified into relationship types. Hence, entity types, subclasses, categories, and relationship types are the different concepts that are used for classification in the EER model. The EER model does not provide explicitly for class properties, but it may be extended to do so. In UML, objects are classified into classes, and it is possible to display both class properties and individual objects.

Knowledge representation models allow multiple classification schemes in which one class is an *instance* of another class (called a **meta-class**). Notice that this *cannot* be represented directly in the EER model, because we have only two levels—classes and instances. The only relationship among classes in the EER model is a superclass/subclass relationship, whereas in some KR schemes an additional class/instance relationship can be represented directly in a class hierarchy. An instance may itself be another class, allowing multiple-level classification schemes.

4.7.2 Identification

Identification is the abstraction process whereby classes and objects are made uniquely identifiable by means of some **identifier**. For example, a class name uniquely identifies a whole class within a schema. An additional mechanism is necessary for telling distinct object instances apart by means of object identifiers. Moreover, it is necessary to identify multiple manifestations in the database of the same real-world

object. For example, we may have a tuple <‘Matthew Clarke’, ‘610618’, ‘376-9821’> in a PERSON relation and another tuple <‘301-54-0836’, ‘CS’, 3.8> in a STUDENT relation that happen to represent the same real-world entity. There is no way to identify the fact that these two database objects (tuples) represent the same real-world entity unless we make a provision *at design time* for appropriate cross-referencing to supply this identification. Hence, identification is needed at two levels:

- To distinguish among database objects and classes
- To identify database objects and to relate them to their real-world counterparts

In the EER model, identification of schema constructs is based on a system of unique names for the constructs in a schema. For example, every class in an EER schema—whether it is an entity type, a subclass, a category, or a relationship type—must have a distinct name. The names of attributes of a particular class must also be distinct. Rules for unambiguously identifying attribute name references in a specialization or generalization lattice or hierarchy are needed as well.

At the object level, the values of key attributes are used to distinguish among entities of a particular entity type. For weak entity types, entities are identified by a combination of their own partial key values and the entities they are related to in the owner entity type(s). Relationship instances are identified by some combination of the entities that they relate to, depending on the cardinality ratio specified.

4.7.3 Specialization and Generalization

Specialization is the process of classifying a class of objects into more specialized subclasses. **Generalization** is the inverse process of generalizing several classes into a higher-level abstract class that includes the objects in all these classes. Specialization is conceptual refinement, whereas generalization is conceptual synthesis. Subclasses are used in the EER model to represent specialization and generalization. We call the relationship between a subclass and its superclass an **IS-A-SUBCLASS-OF** relationship, or simply an **IS-A** relationship. This is the same as the IS-A relationship discussed earlier in Section 4.5.3.

4.7.4 Aggregation and Association

Aggregation is an abstraction concept for building composite objects from their component objects. There are three cases where this concept can be related to the EER model. The first case is the situation in which we aggregate attribute values of an object to form the whole object. The second case is when we represent an aggregation relationship as an ordinary relationship. The third case, which the EER model does not provide for explicitly, involves the possibility of combining objects that are related by a particular relationship instance into a *higher-level aggregate object*. This is sometimes useful when the higher-level aggregate object is itself to be related to another object. We call the relationship between the primitive objects and their aggregate object **IS-A-PART-OF**; the inverse is called **IS-A-COMPONENT-OF**. UML provides for all three types of aggregation.

The abstraction of **association** is used to associate objects from several *independent classes*. Hence, it is somewhat similar to the second use of aggregation. It is represented in the EER model by relationship types, and in UML by associations. This abstract relationship is called **IS-ASSOCIATED-WITH**.

In order to understand the different uses of aggregation better, consider the ER schema shown in Figure 4.11(a), which stores information about interviews by job applicants to various companies. The class COMPANY is an aggregation of the attributes (or component objects) Cname (company name) and Caddress (company address), whereas JOB_APPLICANT is an aggregate of Ssn, Name, Address, and Phone. The relationship attributes Contact_name and Contact_phone represent the name and phone number of the person in the company who is responsible for the interview. Suppose that some interviews result in job offers, whereas others do not. We would like to treat INTERVIEW as a class to associate it with JOB_OFFER. The schema shown in Figure 4.11(b) is *incorrect* because it requires each interview relationship instance to have a job offer. The schema shown in Figure 4.11(c) is *not allowed* because the ER model does not allow relationships among relationships.

One way to represent this situation is to create a higher-level aggregate class composed of COMPANY, JOB_APPLICANT, and INTERVIEW and to relate this class to JOB_OFFER, as shown in Figure 4.11(d). Although the EER model as described in this book does not have this facility, some semantic data models do allow it and call the resulting object a **composite** or **molecular object**. Other models treat entity types and relationship types uniformly and hence permit relationships among relationships, as illustrated in Figure 4.11(c).

To represent this situation correctly in the ER model as described here, we need to create a new weak entity type INTERVIEW, as shown in Figure 4.11(e), and relate it to JOB_OFFER. Hence, we can always represent these situations correctly in the ER model by creating additional entity types, although it may be conceptually more desirable to allow direct representation of aggregation, as in Figure 4.11(d), or to allow relationships among relationships, as in Figure 4.11(c).

The main structural distinction between aggregation and association is that when an association instance is deleted, the participating objects may continue to exist. However, if we support the notion of an aggregate object—for example, a CAR that is made up of objects ENGINE, CHASSIS, and TIRES—then deleting the aggregate CAR object amounts to deleting all its component objects.

4.7.5 Ontologies and the Semantic Web

In recent years, the amount of computerized data and information available on the Web has spiraled out of control. Many different models and formats are used. In addition to the database models that we present in this text, much information is stored in the form of **documents**, which have considerably less structure than

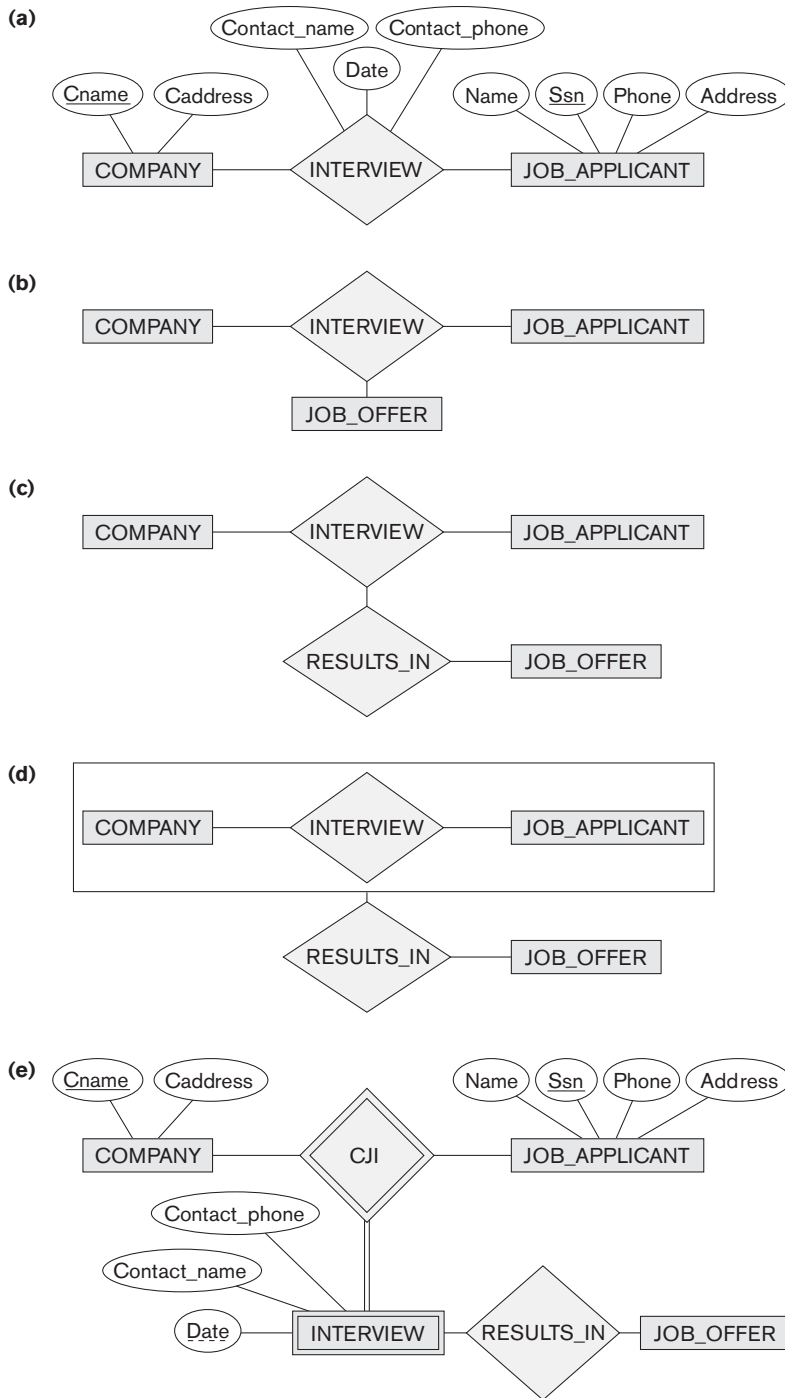


Figure 4.11

Aggregation. (a) The relationship type INTERVIEW is a ternary relationship type (incorrect). (b) Including JOB_OFFER as a participant in the relationship type INTERVIEW (not allowed in ER). (c) Having the RESULTS_IN relationship type participate in other relationships (not allowed in ER). (d) Using aggregation and a composition relationship (molecular object) (generalized relationship type, not allowed in ER but allowed by some modeling tools). (e) Correct representation of aggregation in ER.

database information does. One ongoing project that is attempting to allow information exchange among computers on the Web is called the **Semantic Web**, which attempts to create knowledge representation models that are quite general in order to allow meaningful information exchange and search among machines. The concept of *ontology* is considered to be the most promising basis for achieving the goals of the Semantic Web and is closely related to knowledge representation. In this section, we give a brief introduction to what ontology is and how it can be used as a basis to automate information understanding, search, and exchange.

The study of ontologies attempts to describe the concepts and relationships that are possible in reality through some common vocabulary; therefore, it can be considered as a way to describe the knowledge of a certain community about reality. Ontology originated in the fields of philosophy and metaphysics. One commonly used definition of **ontology** is a *specification of a conceptualization*.¹³

In this definition, a **conceptualization** is the set of concepts and relationships that are used to represent the part of reality or knowledge that is of interest to a community of users. **Specification** refers to the language and vocabulary terms that are used to specify the conceptualization. The ontology includes both *specification* and *conceptualization*. For example, the same conceptualization may be specified in two different languages, giving two separate ontologies. Based on this general definition, there is no consensus on what an ontology is exactly. Some possible ways to describe ontologies are as follows:

- A **thesaurus** (or even a **dictionary** or a **glossary** of terms) describes the relationships between words (vocabulary) that represent various concepts.
- A **taxonomy** describes how concepts of a particular area of knowledge are related using structures similar to those used in a specialization or generalization.
- A detailed **database schema** is considered by some to be an ontology that describes the concepts (entities and attributes) and relationships of a mini-world from reality.
- A **logical theory** uses concepts from mathematical logic to try to define concepts and their interrelationships.

Usually the concepts used to describe ontologies are similar to the concepts we discuss in conceptual modeling, such as entities, attributes, relationships, specializations, and so on. The main difference between an ontology and, say, a database schema, is that the schema is usually limited to describing a small subset of a mini-world from reality in order to store and manage data. An ontology is usually considered to be more general in that it attempts to describe a part of reality or a domain of interest (for example, medical terms, electronic-commerce applications, sports, and so on) as completely as possible.

¹³This definition is given in Gruber (1995).

4.8 Summary

In this chapter we discussed extensions to the ER model that improve its representational capabilities. We called the resulting model the enhanced ER or EER model. We presented the concept of a subclass and its superclass and the related mechanism of attribute/relationship inheritance. We saw how it is sometimes necessary to create additional classes of entities, either because of additional specific attributes or because of specific relationship types. We discussed two main processes for defining superclass/subclass hierarchies and lattices: specialization and generalization.

Next, we showed how to display these new constructs in an EER diagram. We also discussed the various types of constraints that may apply to specialization or generalization. The two main constraints are total/partial and disjoint/overlapping. We discussed the concept of a category or union type, which is a subset of the union of two or more classes, and we gave formal definitions of all the concepts presented.

We introduced some of the notation and terminology of UML for representing specialization and generalization. In Section 4.7, we briefly discussed the discipline of knowledge representation and how it is related to semantic data modeling. We also gave an overview and summary of the types of abstract data representation concepts: classification and instantiation, identification, specialization and generalization, and aggregation and association. We saw how EER and UML concepts are related to each of these.

Review Questions

- 4.1. What is a subclass? When is a subclass needed in data modeling?
- 4.2. Define the following terms: *superclass of a subclass*, *superclass/subclass relationship*, *IS-A relationship*, *specialization*, *generalization*, *category*, *specific (local) attributes*, and *specific relationships*.
- 4.3. Discuss the mechanism of attribute/relationship inheritance. Why is it useful?
- 4.4. Discuss user-defined and predicate-defined subclasses, and identify the differences between the two.
- 4.5. Discuss user-defined and attribute-defined specializations, and identify the differences between the two.
- 4.6. Discuss the two main types of constraints on specializations and generalizations.
- 4.7. What is the difference between a specialization hierarchy and a specialization lattice?
- 4.8. What is the difference between specialization and generalization? Why do we not display this difference in schema diagrams?

- 4.9. How does a category differ from a regular shared subclass? What is a category used for? Illustrate your answer with examples.
- 4.10. For each of the following UML terms (see Sections 3.8 and 4.6), discuss the corresponding term in the EER model, if any: *object*, *class*, *association*, *aggregation*, *generalization*, *multiplicity*, *attributes*, *discriminator*, *link*, *link attribute*, *reflexive association*, and *qualified association*.
- 4.11. Discuss the main differences between the notation for EER schema diagrams and UML class diagrams by comparing how common concepts are represented in each.
- 4.12. List the various data abstraction concepts and the corresponding modeling concepts in the EER model.
- 4.13. What aggregation feature is missing from the EER model? How can the EER model be further enhanced to support it?
- 4.14. What are the main similarities and differences between conceptual database modeling techniques and knowledge representation techniques?
- 4.15. Discuss the similarities and differences between an ontology and a database schema.

Exercises

- 4.16. Design an EER schema for a database application that you are interested in. Specify all constraints that should hold on the database. Make sure that the schema has at least five entity types, four relationship types, a weak entity type, a superclass/subclass relationship, a category, and an n -ary ($n > 2$) relationship type.
- 4.17. Consider the BANK ER schema in Figure 3.21, and suppose that it is necessary to keep track of different types of ACCOUNTS (SAVINGS_ACCTS, CHECKING_ACCTS, ...) and LOANS (CAR_LOANS, HOME_LOANS, ...). Suppose that it is also desirable to keep track of each ACCOUNT's TRANSACTIONS (deposits, withdrawals, checks, ...) and each LOAN's PAYMENTS; both of these include the amount, date, and time. Modify the BANK schema, using ER and EER concepts of specialization and generalization. State any assumptions you make about the additional requirements.
- 4.18. The following narrative describes a simplified version of the organization of Olympic facilities planned for the summer Olympics. Draw an EER diagram that shows the entity types, attributes, relationships, and specializations for this application. State any assumptions you make. The Olympic facilities are divided into sports complexes. Sports complexes are divided into *one-sport* and *multisport* types. Multisport complexes have areas of the complex designated for each sport with a location indicator (e.g., center, NE corner, and so

on). A complex has a location, chief organizing individual, total occupied area, and so on. Each complex holds a series of events (e.g., the track stadium may hold many different races). For each event there is a planned date, duration, number of participants, number of officials, and so on. A roster of all officials will be maintained together with the list of events each official will be involved in. Different equipment is needed for the events (e.g., goal posts, poles, parallel bars) as well as for maintenance. The two types of facilities (one-sport and multisport) will have different types of information. For each type, the number of facilities needed is kept, together with an approximate budget.

- 4.19. Identify all the important concepts represented in the library database case study described below. In particular, identify the abstractions of classification (entity types and relationship types), aggregation, identification, and specialization/generalization. Specify (min, max) cardinality constraints whenever possible. List details that will affect the eventual design but that have no bearing on the conceptual design. List the semantic constraints separately. Draw an EER diagram of the library database.

Case Study: The Georgia Tech Library (GTL) has approximately 16,000 members, 100,000 titles, and 250,000 volumes (an average of 2.5 copies per book). About 10% of the volumes are out on loan at any one time. The librarians ensure that the books that members want to borrow are available when the members want to borrow them. Also, the librarians must know how many copies of each book are in the library or out on loan at any given time. A catalog of books is available online that lists books by author, title, and subject area. For each title in the library, a book description is kept in the catalog; the description ranges from one sentence to several pages. The reference librarians want to be able to access this description when members request information about a book. Library staff includes chief librarian, departmental associate librarians, reference librarians, check-out staff, and library assistants.

Books can be checked out for 21 days. Members are allowed to have only five books out at a time. Members usually return books within three to four weeks. Most members know that they have one week of grace before a notice is sent to them, so they try to return books before the grace period ends. About 5% of the members have to be sent reminders to return books. Most overdue books are returned within a month of the due date. Approximately 5% of the overdue books are either kept or never returned. The most active members of the library are defined as those who borrow books at least ten times during the year. The top 1% of membership does 15% of the borrowing, and the top 10% of the membership does 40% of the borrowing. About 20% of the members are totally inactive in that they are members who never borrow.

To become a member of the library, applicants fill out a form including their SSN, campus and home mailing addresses, and phone numbers. The librari-

ans issue a numbered, machine-readable card with the member's photo on it. This card is good for four years. A month before a card expires, a notice is sent to a member for renewal. Professors at the institute are considered automatic members. When a new faculty member joins the institute, his or her information is pulled from the employee records and a library card is mailed to his or her campus address. Professors are allowed to check out books for three-month intervals and have a two-week grace period. Renewal notices to professors are sent to their campus address.

The library does not lend some books, such as reference books, rare books, and maps. The librarians must differentiate between books that can be lent and those that cannot be lent. In addition, the librarians have a list of some books they are interested in acquiring but cannot obtain, such as rare or out-of-print books and books that were lost or destroyed but have not been replaced. The librarians must have a system that keeps track of books that cannot be lent as well as books that they are interested in acquiring. Some books may have the same title; therefore, the title cannot be used as a means of identification. Every book is identified by its International Standard Book Number (ISBN), a unique international code assigned to all books. Two books with the same title can have different ISBNs if they are in different languages or have different bindings (hardcover or softcover). Editions of the same book have different ISBNs.

The proposed database system must be designed to keep track of the members, the books, the catalog, and the borrowing activity.

4.20. Design a database to keep track of information for an art museum. Assume that the following requirements were collected:

- The museum has a collection of ART_OBJECTS. Each ART_OBJECT has a unique Id_no, an Artist (if known), a Year (when it was created, if known), a Title, and a Description. The art objects are categorized in several ways, as discussed below.
- ART_OBJECTS are categorized based on their type. There are three main types—PAINTING, SCULPTURE, and STATUE—plus another type called OTHER to accommodate objects that do not fall into one of the three main types.
- A PAINTING has a Paint_type (oil, watercolor, etc.), material on which it is Drawn_on (paper, canvas, wood, etc.), and Style (modern, abstract, etc.).
- A SCULPTURE or a statue has a Material from which it was created (wood, stone, etc.), Height, Weight, and Style.
- An art object in the OTHER category has a Type (print, photo, etc.) and Style.
- ART_OBJECTs are categorized as either PERMANENT_COLLECTION (objects that are owned by the museum) and BORROWED. Information captured about objects in the PERMANENT_COLLECTION includes Date_acquired, Status (on display, on loan, or stored), and Cost. Information

captured about BORROWED objects includes the Collection from which it was borrowed, Date_borrowed, and Date_returned.

- Information describing the country or culture of Origin (Italian, Egyptian, American, Indian, and so forth) and Epoch (Renaissance, Modern, Ancient, and so forth) is captured for each ART_OBJECT.
- The museum keeps track of ARTIST information, if known: Name, DateBorn (if known), Date_died (if not living), Country_of_origin, Epoch, Main_style, and Description. The Name is assumed to be unique.
- Different EXHIBITIONS occur, each having a Name, Start_date, and End_date. EXHIBITIONS are related to all the art objects that were on display during the exhibition.
- Information is kept on other COLLECTIONS with which the museum interacts; this information includes Name (unique), Type (museum, personal, etc.), Description, Address, Phone, and current Contact_person.

Draw an EER schema diagram for this application. Discuss any assumptions you make, and then justify your EER design choices.

- 4.21. Figure 4.12 shows an example of an EER diagram for a small-private-airport database; the database is used to keep track of airplanes, their owners, airport employees, and pilots. From the requirements for this database, the following information was collected: Each AIRPLANE has a registration number [Reg#], is of a particular plane type [OF_TYPE], and is stored in a particular hangar [STORED_IN]. Each PLANE_TYPE has a model number [Model], a capacity [Capacity], and a weight [Weight]. Each HANGAR has a number [Number], a capacity [Capacity], and a location [Location]. The database also keeps track of the OWNERS of each plane [OWNS] and the EMPLOYEES who have maintained the plane [MAINTAIN]. Each relationship instance in OWNS relates an AIRPLANE to an OWNER and includes the purchase date [Pdate]. Each relationship instance in MAINTAIN relates an EMPLOYEE to a service record [SERVICE]. Each plane undergoes service many times; hence, it is related by [PLANE_SERVICE] to a number of SERVICE records. A SERVICE record includes as attributes the date of maintenance [Date], the number of hours spent on the work [Hours], and the type of work done [Work_code]. We use a weak entity type [SERVICE] to represent airplane service, because the airplane registration number is used to identify a service record. An OWNER is either a person or a corporation. Hence, we use a union type (category) [OWNER] that is a subset of the union of corporation [CORPORATION] and person [PERSON] entity types. Both pilots [PILOT] and employees [EMPLOYEE] are subclasses of PERSON. Each PILOT has specific attributes license number [Lic_num] and restrictions [Restr]; each EMPLOYEE has specific attributes salary [Salary] and shift worked [Shift]. All PERSON entities in the database have data kept on their Social Security number [Ssn], name [Name], address [Address], and telephone number [Phone]. For CORPORATION entities, the data kept includes name [Name], address [Address], and telephone number [Phone]. The database also keeps track of the types of

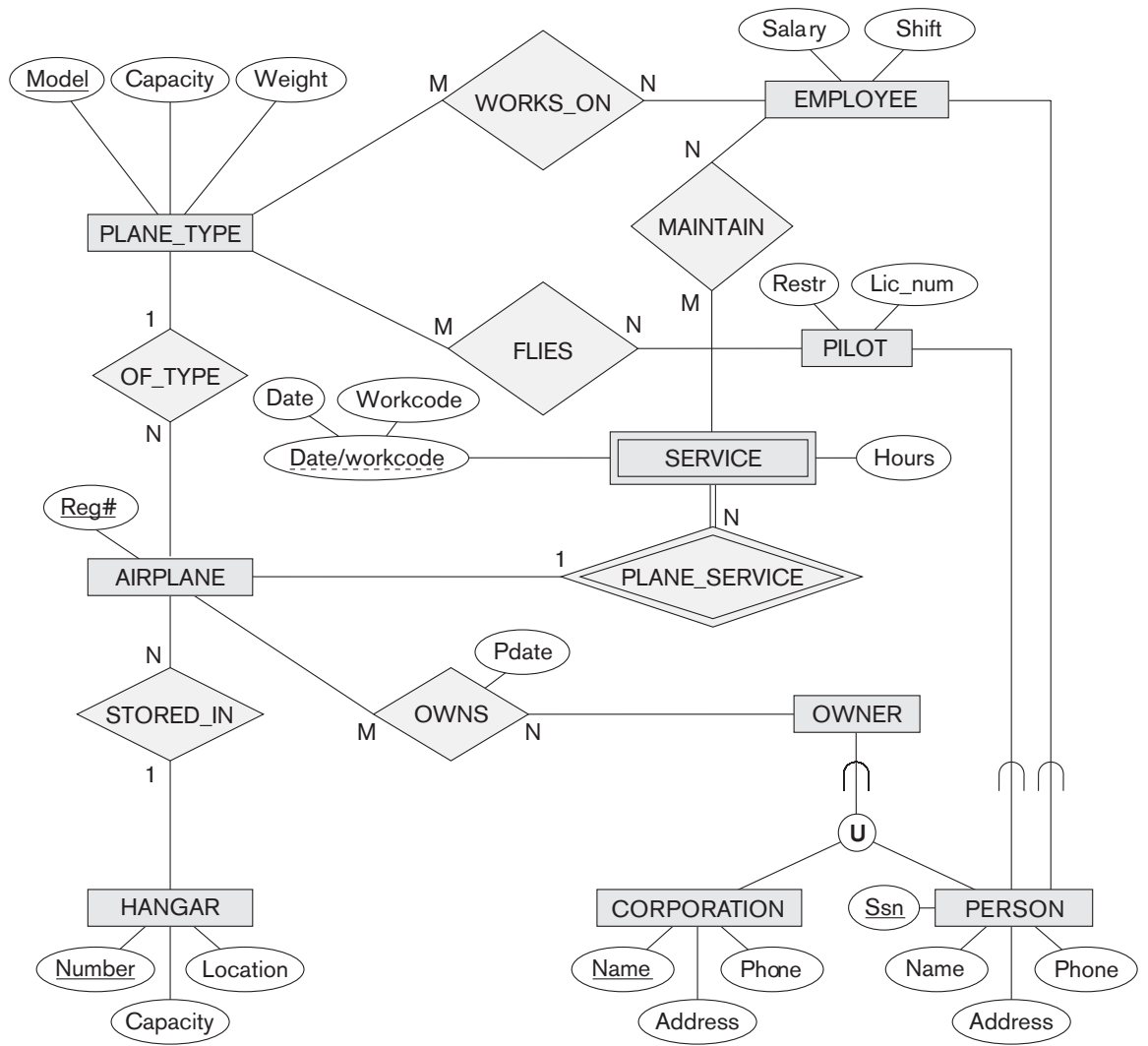


Figure 4.12
EER schema for a SMALL_AIRPORT database.

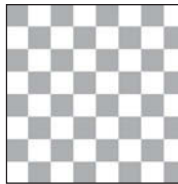
planes each pilot is authorized to fly [FLIES] and the types of planes each employee can do maintenance work on [WORKS_ON]. Show how the SMALL_AIRPORT EER schema in Figure 4.12 may be represented in UML notation. (Note: We have not discussed how to represent categories (union types) in UML, so you do not have to map the categories in this and the following question.)

- 4.22.** Show how the UNIVERSITY EER schema in Figure 4.9 may be represented in UML notation.

- 4.23. Consider the entity sets and attributes shown in the following table. Place a checkmark in one column in each row to indicate the relationship between the far left and far right columns.
- The left side has a relationship with the right side.
 - The right side is an attribute of the left side.
 - The left side is a specialization of the right side.
 - The left side is a generalization of the right side.

Entity Set	(a) Has a Relationship with	(b) Has an Attribute that is	(c) Is a Specialization of	(d) Is a Generalization of	Entity Set or Attribute
1. MOTHER					PERSON
2. DAUGHTER					MOTHER
3. STUDENT					PERSON
4. STUDENT					Student_id
5. SCHOOL					STUDENT
6. SCHOOL					CLASS_ROOM
7. ANIMAL					HORSE
8. HORSE					Breed
9. HORSE					Age
10. EMPLOYEE					SSN
11. FURNITURE					CHAIR
12. CHAIR					Weight
13. HUMAN					WOMAN
14. SOLDIER					PERSON
15. ENEMY_COMBATANT					PERSON

- 4.24. Draw a UML diagram for storing a played game of chess in a database. You may look at <http://www.chessgames.com> for an application similar to what you are designing. State clearly any assumptions you make in your UML diagram. A sample of assumptions you can make about the scope is as follows:
- The game of chess is played between two players.
 - The game is played on an 8×8 board like the one shown below:

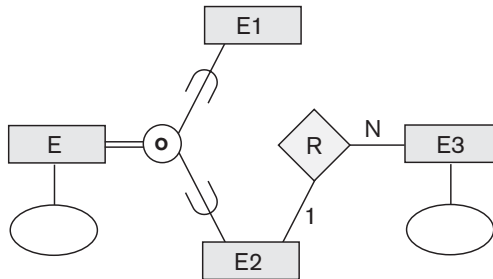


3. The players are assigned a color of black or white at the start of the game.
4. Each player starts with the following pieces (traditionally called chessmen):
 - a. king
 - b. queen
 - c. 2 rooks
 - d. 2 bishops
 - e. 2 knights
 - f. 8 pawns
5. Every piece has its own initial position.
6. Every piece has its own set of legal moves based on the state of the game. You do not need to worry about which moves are or are not legal except for the following issues:
 - a. A piece may move to an empty square or capture an opposing piece.
 - b. If a piece is captured, it is removed from the board.
 - c. If a pawn moves to the last row, it is “promoted” by converting it to another piece (queen, rook, bishop, or knight).

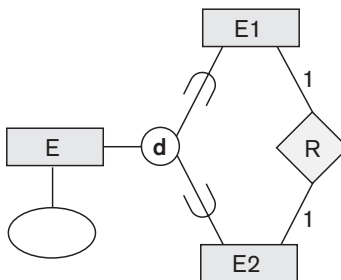
Note: Some of these functions may be spread over multiple classes.

- 4.25. Draw an EER diagram for a game of chess as described in Exercise 4. 24. Focus on persistent storage aspects of the system. For example, the system would need to retrieve all the moves of every game played in sequential order.
- 4.26. Which of the following EER diagrams is/are incorrect and why? State clearly any assumptions you make.

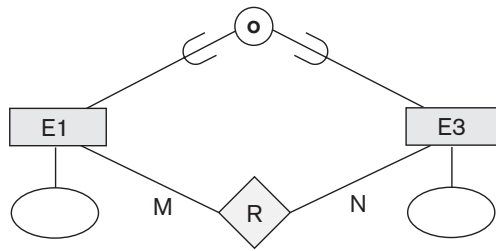
a.



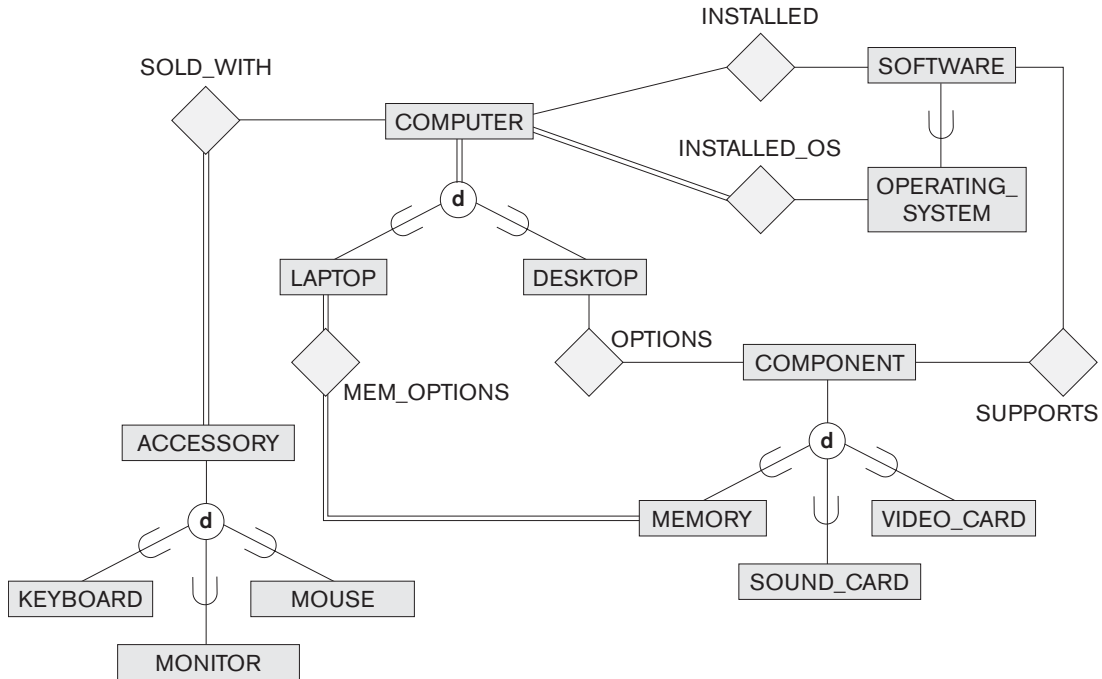
b.



c.



4.27. Consider the following EER diagram that describes the computer systems at a company. Provide your own attributes and key for each entity type. Supply max cardinality constraints justifying your choice. Write a complete narrative description of what this EER diagram represents.



Laboratory Exercises

4.28. Consider a GRADE_BOOK database in which instructors within an academic department record points earned by individual students in their classes. The data requirements are summarized as follows:

- Each student is identified by a unique identifier, first and last name, and an e-mail address.
- Each instructor teaches certain courses each term. Each course is identified by a course number, a section number, and the term in which it is taught. For

each course he or she teaches, the instructor specifies the minimum number of points required in order to earn letter grades A, B, C, D, and F. For example, 90 points for an A, 80 points for a B, 70 points for a C, and so forth.

- Students are enrolled in each course taught by the instructor.
- Each course has a number of grading components (such as midterm exam, final exam, project, and so forth). Each grading component has a maximum number of points (such as 100 or 50) and a weight (such as 20% or 10%). The weights of all the grading components of a course usually total 100.
- Finally, the instructor records the points earned by each student in each of the grading components in each of the courses. For example, student 1234 earns 84 points for the midterm exam grading component of the section 2 course CSc2310 in the fall term of 2009. The midterm exam grading component may have been defined to have a maximum of 100 points and a weight of 20% of the course grade.

Design an enhanced entity–relationship diagram for the grade book database and build the design using a data modeling tool such as ERwin or Rational Rose.

4.29. Consider an ONLINE_AUCTION database system in which members (buyers and sellers) participate in the sale of items. The data requirements for this system are summarized as follows:

- The online site has members, each of whom is identified by a unique member number and is described by an e-mail address, name, password, home address, and phone number.
- A member may be a buyer or a seller. A buyer has a shipping address recorded in the database. A seller has a bank account number and routing number recorded in the database.
- Items are placed by a seller for sale and are identified by a unique item number assigned by the system. Items are also described by an item title, a description, starting bid price, bidding increment, the start date of the auction, and the end date of the auction.
- Items are also categorized based on a fixed classification hierarchy (for example, a modem may be classified as COMPUTER → HARDWARE → MODEM).
- Buyers make bids for items they are interested in. Bid price and time of bid are recorded. The bidder at the end of the auction with the highest bid price is declared the winner, and a transaction between buyer and seller may then proceed.
- The buyer and seller may record feedback regarding their completed transactions. Feedback contains a rating of the other party participating in the transaction (1–10) and a comment.

Design an enhanced entity–relationship diagram for the ONLINE_AUCTION database and build the design using a data modeling tool such as ERwin or Rational Rose.

- 4.30. Consider a database system for a baseball organization such as the major leagues. The data requirements are summarized as follows:
- The personnel involved in the league include players, coaches, managers, and umpires. Each is identified by a unique personnel id. They are also described by their first and last names along with the date and place of birth.
 - Players are further described by other attributes such as their batting orientation (left, right, or switch) and have a lifetime batting average (BA).
 - Within the players group is a subset of players called pitchers. Pitchers have a lifetime ERA (earned run average) associated with them.
 - Teams are uniquely identified by their names. Teams are also described by the city in which they are located and the division and league in which they play (such as Central division of the American League).
 - Teams have one manager, a number of coaches, and a number of players.
 - Games are played between two teams, with one designated as the home team and the other the visiting team on a particular date. The score (runs, hits, and errors) is recorded for each team. The team with the most runs is declared the winner of the game.
 - With each finished game, a winning pitcher and a losing pitcher are recorded. In case there is a save awarded, the save pitcher is also recorded.
 - With each finished game, the number of hits (singles, doubles, triples, and home runs) obtained by each player is also recorded.

Design an enhanced entity–relationship diagram for the BASEBALL database and enter the design using a data modeling tool such as ERwin or Rational Rose.

- 4.31. Consider the EER diagram for the UNIVERSITY database shown in Figure 4.9. Enter this design using a data modeling tool such as ERwin or Rational Rose. Make a list of the differences in notation between the diagram in the text and the corresponding equivalent diagrammatic notation you end up using with the tool.
- 4.32. Consider the EER diagram for the small AIRPORT database shown in Figure 4.12. Build this design using a data modeling tool such as ERwin or Rational Rose. Be careful how you model the category OWNER in this diagram. (*Hint:* Consider using CORPORATION_IS_OWNER and PERSON_IS_OWNER as two distinct relationship types.)
- 4.33. Consider the UNIVERSITY database described in Exercise 3.16. You already developed an ER schema for this database using a data modeling tool such as

ERwin or Rational Rose in Lab Exercise 3.31. Modify this diagram by classifying COURSES as either UNDERGRAD_COURSES or GRAD_COURSES and INSTRUCTORS as either JUNIOR_PROFESSORS or SENIOR_PROFESSORS. Include appropriate attributes for these new entity types. Then establish relationships indicating that junior instructors teach undergraduate courses whereas senior instructors teach graduate courses.

Selected Bibliography

Many papers have proposed conceptual or semantic data models. We give a representative list here. One group of papers, including Abrial (1974), Senko's DIAM model (1975), the NIAM method (Verheijen and VanBekum 1982), and Bracchi et al. (1976), presents semantic models that are based on the concept of binary relationships. Another group of early papers discusses methods for extending the relational model to enhance its modeling capabilities. This includes the papers by Schmid and Swenson (1975), Navathe and Schkolnick (1978), Codd's RM/T model (1979), Furtado (1978), and the structural model of Wiederhold and Elmasri (1979).

The ER model was proposed originally by Chen (1976) and is formalized in Ng (1981). Since then, numerous extensions of its modeling capabilities have been proposed, as in Scheuermann et al. (1979), Dos Santos et al. (1979), Teorey et al. (1986), Gogolla and Hohenstein (1991), and the entity–category–relationship (ECR) model of Elmasri et al. (1985). Smith and Smith (1977) present the concepts of generalization and aggregation. The semantic data model of Hammer and McLeod (1981) introduces the concepts of class/subclass lattices, as well as other advanced modeling concepts.

A survey of semantic data modeling appears in Hull and King (1987). Eick (1991) discusses design and transformations of conceptual schemas. Analysis of constraints for n -ary relationships is given in Soutou (1998). UML is described in detail in Booch, Rumbaugh, and Jacobson (1999). Fowler and Scott (2000) and Stevens and Pooley (2000) give concise introductions to UML concepts.

Fensel (2000, 2003) discusses the Semantic Web and application of ontologies. Uschold and Gruninger (1996) and Gruber (1995) discuss ontologies. The June 2002 issue of *Communications of the ACM* is devoted to ontology concepts and applications. Fensel (2003) discusses ontologies and e-commerce.

9.2 Mapping EER Model Constructs to Relations

In this section, we discuss the mapping of EER model constructs to relations by extending the ER-to-relational mapping algorithm that was presented in Section 9.1.1.

9.2.1 Mapping of Specialization or Generalization

There are several options for mapping a number of subclasses that together form a specialization (or alternatively, that are generalized into a superclass), such as the {SECRETARY, TECHNICIAN, ENGINEER} subclasses of EMPLOYEE in Figure 4.4. The two main options are to map the whole specialization into a **single table**, or to map it into **multiple tables**. Within each option are variations that depend on the constraints on the specialization/generalization.

We can add a further step to our ER-to-relational mapping algorithm from Section 9.1.1, which has seven steps, to handle the mapping of specialization. Step 8, which follows, gives the most common options; other mappings are also possible. We discuss the conditions under which each option should be used. We use $\text{Attrs}(R)$ to denote *the attributes of a relation R*, and $\text{PK}(R)$ to denote *the primary key of R*. First we describe the mapping formally, then we illustrate it with examples.

Step 8: Options for Mapping Specialization or Generalization. Convert each specialization with m subclasses $\{S_1, S_2, \dots, S_m\}$ and (generalized) superclass C , where the attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key, into relation schemas using one of the following options:

- **Option 8A: Multiple relations—superclass and subclasses.** Create a relation L for C with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\}$ and $\text{PK}(L) = k$. Create a relation L_i for each subclass S_i , $1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{k\} \cup \{\text{attributes of } S_i\}$ and $\text{PK}(L_i) = k$. This option works for any specialization (total or partial, disjoint or overlapping).
- **Option 8B: Multiple relations—subclass relations only.** Create a relation L_i for each subclass S_i , $1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{\text{attributes of } S_i\} \cup \{k, a_1, \dots, a_n\}$ and $\text{PK}(L_i) = k$. This option only works for a specialization whose subclasses are *total* (every entity in the superclass must belong to (at least) one of the subclasses). Additionally, it is only recommended if the specialization has the *disjointness constraint* (see Section 4.3.1). If the specialization is *overlapping*, the same entity may be duplicated in several relations.
- **Option 8C: Single relation with one type attribute.** Create a single relation L with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t\}$ and $\text{PK}(L) = k$. The attribute t is called a **type** (or **discriminating**) attribute whose value indicates the subclass to which each tuple belongs, if any. This option works only for a specialization whose subclasses are *disjoint*, and has the potential for generating many NULL values if many specific (local) attributes exist in the subclasses.
- **Option 8D: Single relation with multiple type attributes.** Create a single relation schema L with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, \dots, t_m\}$ and $\text{PK}(L) = k$. Each t_i , $1 \leq i \leq m$, is a **Boolean type attribute** indicating whether or not a tuple belongs to subclass S_i . This option is used for a specialization whose subclasses are *overlapping* (but will also work for a disjoint specialization).

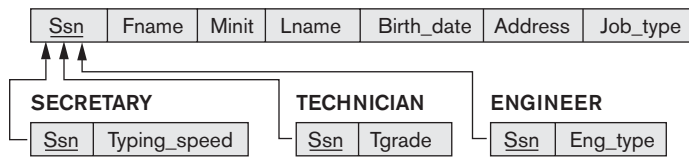
Options 8A and 8B are the **multiple-relation options**, whereas options 8C and 8D are the **single-relation options**. Option 8A creates a relation L for the superclass C and its attributes, plus a relation L_i for each subclass S_i ; each L_i includes the specific (local) attributes of S_i , plus the primary key of the superclass C , which is propagated to L_i and becomes its primary key. It also becomes a foreign key to the superclass relation. An EQUIJOIN operation on the primary key between any L_i and L produces all the specific and inherited attributes of the entities in S_i . This option is illustrated in Figure 9.5(a) for the EER schema in Figure 4.4. Option 8A works for any constraints on the specialization: disjoint or overlapping, total or partial. Notice that the constraint

$$\pi_{\langle k \rangle}(L_i) \subseteq \pi_{\langle k \rangle}(L)$$

must hold for each L_i . This specifies a foreign key from each L_i to L .

In option 8B, the EQUIJOIN operation between each subclass and the superclass is *built into* the schema and the superclass relation L is done away with, as illustrated in Figure 9.5(b) for the EER specialization in Figure 4.3(b). This option works well only when *both* the disjoint and total constraints hold. If the specialization is not total, an entity that does not belong to any of the subclasses S_i is lost. If the specialization is not disjoint, an entity belonging to more than one subclass will have its

(a) EMPLOYEE



(b) CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers
------------	------------------	-------	-----------	------------------

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage
------------	------------------	-------	-------------	---------

(c) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type
-----	-------	-------	-------	------------	---------	----------	--------------	--------	----------

(d) PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price
---------	-------------	-------	------------	------------------	----------	-------	---------------	------------

Figure 9.5

Options for mapping specialization or generalization. (a) Mapping the EER schema in Figure 4.4 using option 8A. (b) Mapping the EER schema in Figure 4.3(b) using option 8B. (c) Mapping the EER schema in Figure 4.4 using option 8C. (d) Mapping Figure 4.5 using option 8D with Boolean type fields Mflag and Pflag.

inherited attributes from the superclass C stored redundantly in more than one table L_i . With option 8B, no relation holds all the entities in the superclass C ; consequently, we must apply an OUTER UNION (or FULL OUTER JOIN) operation (see Section 6.4) to the L_i relations to retrieve all the entities in C . The result of the outer union will be similar to the relations under options 8C and 8D except that the type fields will be missing. Whenever we search for an arbitrary entity in C , we must search all the m relations L_i .

Options 8C and 8D create a single relation to represent the superclass C and all its subclasses. An entity that does not belong to some of the subclasses will have NULL values for the specific (local) attributes of these subclasses. These options are not recommended if many specific attributes are defined for the subclasses. If few local subclass attributes exist, however, these mappings are preferable to options 8A and 8B because they do away with the need to specify JOIN operations; therefore, they can yield a more efficient implementation for queries.

Option 8C is used to handle disjoint subclasses by including a single **type** (or **image** or **discriminating**) **attribute** t to indicate to which of the m subclasses each tuple belongs; hence, the domain of t could be $\{1, 2, \dots, m\}$. If the specialization is partial, t can have NULL values in tuples that do not belong to any subclass. If the specialization is attribute-defined, that attribute itself serves the purpose of t and t is not needed; this option is illustrated in Figure 9.5(c) for the EER specialization in Figure 4.4.

Option 8D is designed to handle overlapping subclasses by including m **Boolean type** (or **flag**) fields, one for *each* subclass. It can also be used for disjoint subclasses.

Each type field t_i can have a domain {yes, no}, where a value of yes indicates that the tuple is a member of subclass S_i . If we use this option for the EER specialization in Figure 4.4, we would include three type attributes—`Is_a_secretary`, `Is_a_engineer`, and `Is_a_technician`—instead of the `Job_type` attribute in Figure 9.5(c). Figure 9.5(d) shows the mapping of the specialization from Figure 4.5 using option 8D.

For a multilevel specialization (or generalization) hierarchy or lattice, we do not have to follow the same mapping option for all the specializations. Instead, we can use one mapping option for part of the hierarchy or lattice and other options for other parts. Figure 9.6 shows one possible mapping into relations for the EER lattice in Figure 4.6. Here we used option 8A for `PERSON`/{`EMPLOYEE`, `ALUMNUS`, `STUDENT`}, and option 8C for `EMPLOYEE`/{`STAFF`, `FACULTY`, `STUDENT_ASSISTANT`} by including the type attribute `Employee_type`. We then used the single-table option 8D for `STUDENT_ASSISTANT`/{`RESEARCH_ASSISTANT`, `TEACHING_ASSISTANT`} by including the type attributes `Ta_flag` and `Ra_flag` in `EMPLOYEE`. We also used option 8D for `STUDENT`/`STUDENT_ASSISTANT` by including the type attributes `Student_assist_flag` in `STUDENT`, and for `STUDENT`/{`GRADUATE_STUDENT`, `UNDERGRADUATE_STUDENT`} by including the type attributes `Grad_flag` and `Undergrad_flag` in `STUDENT`. In Figure 9.6, all attributes whose names end with *type* or *flag* are type fields.

9.2.2 Mapping of Shared Subclasses (Multiple Inheritance)

A shared subclass, such as `ENGINEERING_MANAGER` in Figure 4.6, is a subclass of several superclasses, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type) as we discussed in Section 4.4. We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm. In Figure 9.6, options 8C and 8D are used for the shared subclass `STUDENT_ASSISTANT`. Option 8C is used in the `EMPLOYEE` relation (`Employee_type` attribute) and option 8D is used in the `STUDENT` relation (`Student_assist_flag` attribute).

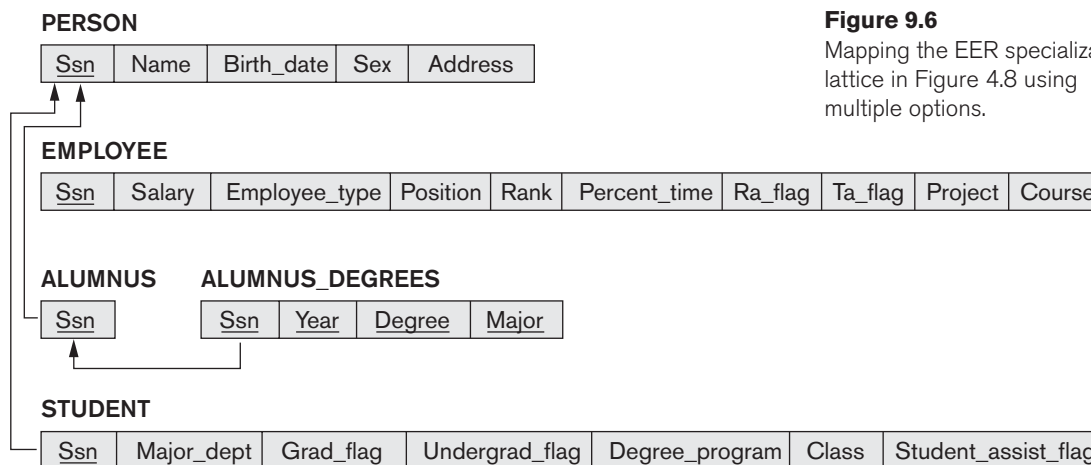


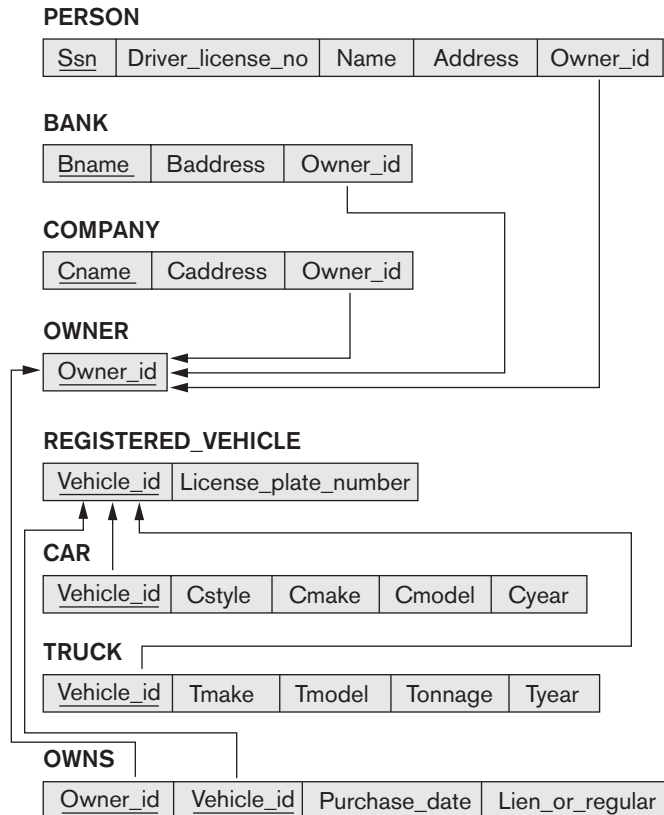
Figure 9.6
Mapping the EER specialization lattice in Figure 4.8 using multiple options.

9.2.3 Mapping of Categories (Union Types)

We add another step to the mapping procedure—step 9—to handle categories. A category (or union type) is a subclass of the *union* of two or more superclasses that can have different keys because they can be of different entity types (see Section 4.4). An example is the OWNER category shown in Figure 4.8, which is a subset of the union of three entity types PERSON, BANK, and COMPANY. The other category in that figure, REGISTERED_VEHICLE, has two superclasses that have the same key attribute.

Step 9: Mapping of Union Types (Categories). For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the union type. The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the relation. In our example in Figure 4.8, we create a relation OWNER to correspond to the OWNER category, as illustrated in Figure 9.7, and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called Owner_id. We also

Figure 9.7
Mapping the EER categories
(union types) in Figure 4.8 to
relations.



include the surrogate key attribute `Owner_id` as foreign key in each relation corresponding to a superclass of the category, to specify the correspondence in values between the surrogate key and the original key of each superclass. Notice that if a particular `PERSON` (or `BANK` or `COMPANY`) entity is not a member of `OWNER`, it would have a `NULL` value for its `Owner_id` attribute in its corresponding tuple in the `PERSON` (or `BANK` or `COMPANY`) relation, and it would not have a tuple in the `OWNER` relation. It is also recommended to add a type attribute (not shown in Figure 9.7) to the `OWNER` relation to indicate the particular entity type to which each tuple belongs (`PERSON` or `BANK` or `COMPANY`).

For a category whose superclasses have the same key, such as `VEHICLE` in Figure 4.8, there is no need for a surrogate key. The mapping of the `REGISTERED_VEHICLE` category, which illustrates this case, is also shown in Figure 9.7.