3 Data Modeling Using the Entity-Relationship Model

Conceptual modeling is a very important phase in designing a successful database application. Generally, the term **database application** refers to a particular database and the associated programs that implement the database queries and updates. For example, a BANK database application that keeps track of customer accounts would include programs that implement database updates corresponding to customers making deposits and withdrawals. These programs provide user-friendly graphical user interfaces (GUIs) utilizing forms and menus for the end users of the application—the bank tellers, in this example. Hence, part of the database application will require the design, implementation, and testing of these **application programs.** Traditionally, the design and testing of application programs has been considered to be more in the realm of the software engineering domain than in the database domain. As database design methodologies include more of the concepts for specifying operations on database objects, and as software engineering methodologies specify in more detail the structure of the databases that software programs will use and access, it is clear that these activities are strongly related. We briefly discuss some of the concepts for specifying database operations in Chapter 4, and again when we discuss database design methodology with example applications in Chapter 12 of this book.

In this chapter, we follow the traditional approach of concentrating on the database structures and constraints during database design. We present the modeling concepts of the **Entity-Relationship (ER) model,** which is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts. We describe

the basic data-structuring concepts and constraints of the ER model and discuss their use in the design of conceptual schemas for database applications. We also present the diagrammatic notation associated with the ER model, known as **ER diagrams.**

Object modeling methodologies such as **UML** (**Universal Modeling Language**) are becoming increasingly popular in software design and engineering. These methodologies go beyond database design to specify detailed design of software modules and their interactions using various types of diagrams. An important part of these methodologies—namely, *class diagrams*[1]—are similar in many ways to the ER diagrams. In class diagrams, *operations* on objects are specified, in addition to specifying the database schema structure. Operations can be used to specify the *functional requirements* during database design, as discussed in Section 3.1. We present some of the UML notation and concepts for class diagrams that are particularly relevant to database design in Section 3.8, and briefly compare these to ER notation and concepts. Additional UML notation and concepts are presented in Section 4.6 and in Chapter 12.

This chapter is organized as follows. Section 3.1 discusses the role of high-level conceptual data models in database design. We introduce the requirements for an example database application in Section 3.2 to illustrate the use of concepts from the ER model. This example database is also used in subsequent chapters. In Section 3.3 we present the concepts of entities and attributes, and we gradually introduce the diagrammatic technique for displaying an ER schema. In Section 3.4 we introduce the concepts of binary relationships and their roles and structural constraints. Section 3.5 introduces weak entity types. Section 3.6 shows how a schema design is refined to include relationships. Section 3.7 reviews the notation for ER diagrams, summarizes the issues that arise in schema design, and discusses how to choose the names for database schema constructs. Section 3.8 introduces some UML class diagram concepts, compares them to ER model concepts, and applies them to the same database example. Section 3.9 summarizes the chapter.

The material in Sections 3.8 may be left out of an introductory course if desired. On the other hand, if more thorough coverage of data modeling concepts and conceptual database design is desired, the reader should continue on to the material in Chapter 4 after concluding Chapter 3. Chapter 4 describes extensions to the ER model that lead to the Enhanced-ER (EER) model, which includes concepts such as specialization, generalization, inheritance, and union types (categories). We also introduce some additional UML concepts and notation in Chapter 4.

3.1 Using High-Level Conceptual Data Models for Database Design

Figure 3.1 shows a simplified description of the database design process. The first step shown is **requirements collection and analysis.** During this step, the database designers interview prospective database users to understand and document their **data requirements.** The result of this

1. A **class** is similar to an *entity type* in many ways.

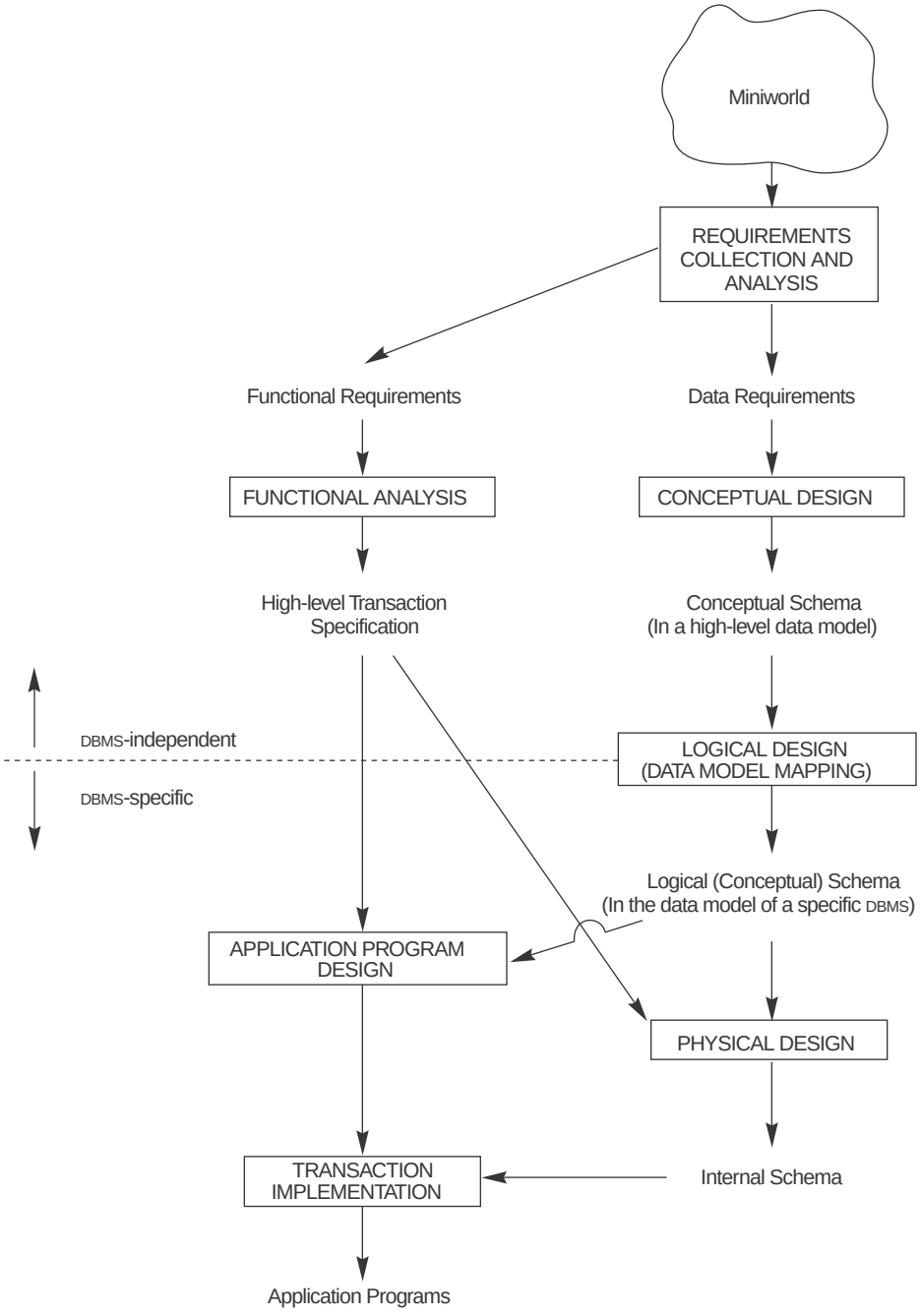

FIGURE 3.1 A simplified diagram to illustrate the main phases of database design

step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user-defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates. In software design, it is common to use *data flow diagrams*, *sequence diagrams*, *scenarios*, and other techniques for specifying functional requirements. We will not discuss any of these techniques here because they are usually described in detail in software engineering texts. We give an overview of some of these techniques in Chapter 12.

Once all the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design.** The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users. The high-level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict. This approach enables the database designers to concentrate on specifying the properties of the data, without being concerned with storage details. Consequently, it is easier for them to come up with a good conceptual database design.

During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user operations identified during functional analysis. This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping,** and its result is a database schema in the implementation data model of the DBMS.

The last step is the **physical design** phase, during which the internal storage structures, indexes, access paths, and file organizations for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications. We discuss the database design process in more detail in Chapter 12.

We present only the basic ER model concepts for conceptual schema design in this chapter. Additional modeling concepts are discussed in Chapter 4, when we introduce the EER model.

3.2 An Example Database Application

In this section we describe an example database application, called COMPANY, that serves to illustrate the basic ER model concepts and their use in schema design. We list the data requirements for the database here, and then create its conceptual schema step by step as

we introduce the modeling concepts of the ER model. The COMPANY database keeps track of a company's employees, departments, and projects. Suppose that after the requirements collection and analysis phase, the database designers provided the following description of the "miniworld"—the part of the company to be represented in the database:

1. The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
2. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
3. We store each employee's name, social security number,[2] address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

Figure 3.2 shows how the schema for this database application can be displayed by means of the graphical notation known as **ER diagrams.** We describe the step-by-step process of deriving this schema from the stated requirements—and explain the ER diagrammatic notation—as we introduce the ER model concepts in the following section.

3.3 Entity Types, Entity Sets, Attributes, and Keys

The ER model describes data as *entities*, *relationships*, and *attributes*. In Section 3.3.1 we introduce the concepts of entities and their attributes. We discuss entity types and key attributes in Section 3.3.2. Then, in Section 3.3.3, we specify the initial conceptual design of the entity types for the COMPANY database. Relationships are described in Section 3.4.

3.3.1 Entities and Attributes

Entities and Their Attributes. The basic object that the ER model represents is an **entity,** which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or

2. The social security number, or SSN, is a unique nine-digit identifier assigned to each individual in the United States to keep track of his or her employment, benefits, and taxes. Other countries may have similar identification schemes, such as personal identification card numbers.

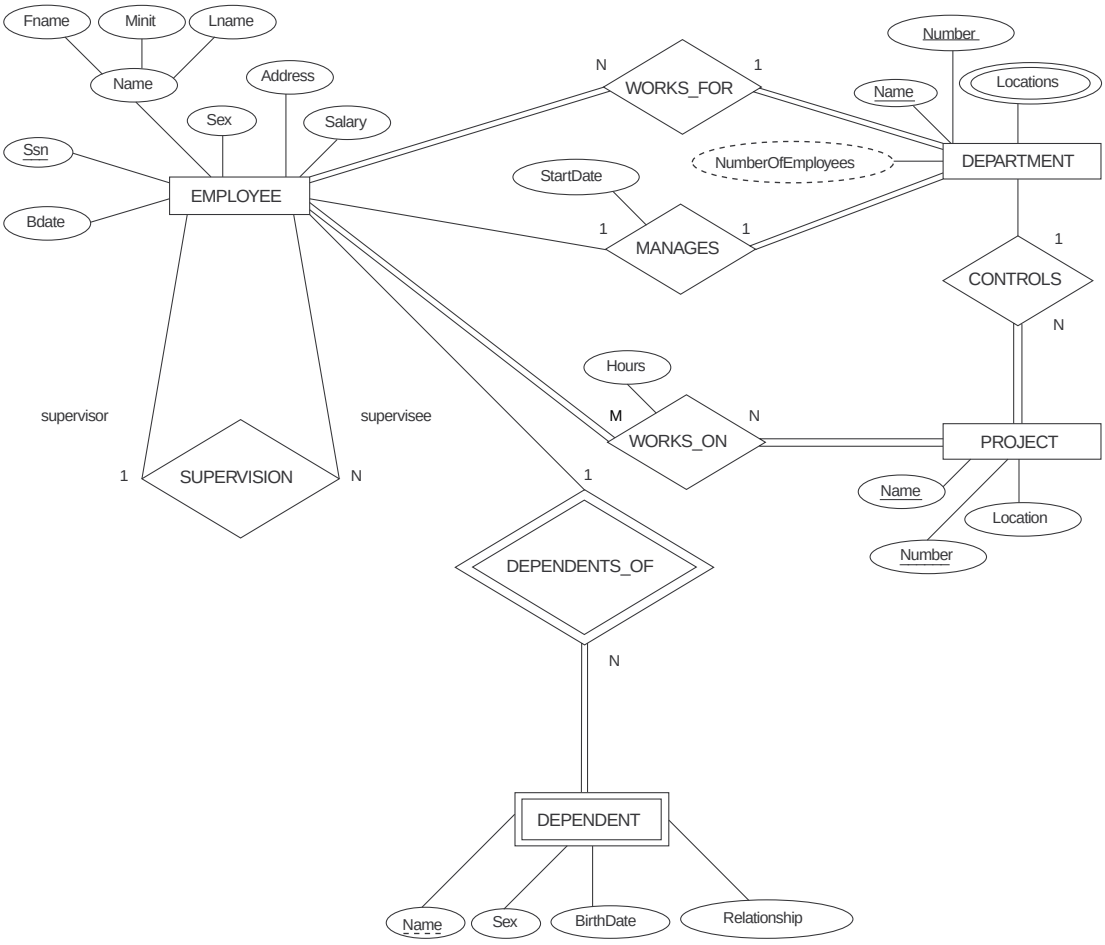

FIGURE 3.2 An ER schema diagram for the COMPANY database

employee) or it may be an object with a conceptual existence (for example, a company, a job, or a university course). Each entity has **attributes**—the particular properties that describe it. For example, an employee entity may be described by the employee's name, age, address, salary, and job. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Figure 3.3 shows two entities and the values of their attributes. The employee entity e_1 has four attributes: Name, Address, Age, and HomePhone; their values are "John Smith," "2311 Kirby, Houston, Texas 77001," "55," and "713-749-2630," respectively. The company entity c_1 has three attributes: Name, Headquarters, and President; their values are "Sunco Oil," "Houston," and "John Smith," respectively.

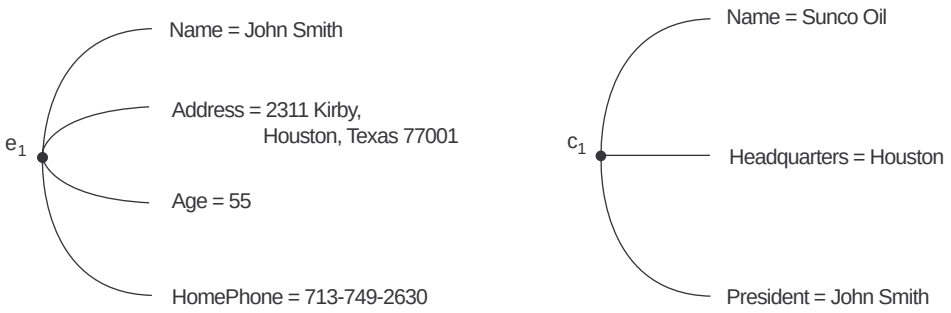

FIGURE 3.3 Two entities, employee e_1 and company c_1, and their attributes

Several types of attributes occur in the ER model: *simple* versus *composite*, *single-valued* versus *multivalued*, and *stored* versus *derived*. We first define these attribute types and illustrate their use via examples. We then introduce the concept of a *null value* for an attribute.

Composite versus Simple (Atomic) Attributes. **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the employee entity shown in Figure 3.3 can be subdivided into StreetAddress, City, State, and Zip,[3] with the values "2311 Kirby," "Houston," "Texas," and "77001." Attributes that are not divisible are called **simple** or **atomic attributes.** Composite attributes can form a hierarchy; for example, StreetAddress can be further subdivided into three simple attributes: Number, Street, and ApartmentNumber, as shown in Figure 3.4. The value of a composite attribute is the concatenation of the values of its constituent simple attributes.

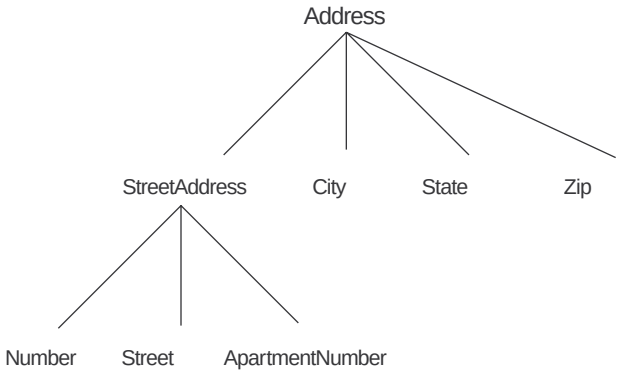

FIGURE 3.4 A hierarchy of composite attributes

3. The zip code is the name used in the United States for a 5-digit postal code.

Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes. For example, if there is no need to refer to the individual components of an address (zip code, street, and so on), then the whole address can be designated as a simple attribute.

Single-Valued versus Multivalued Attributes. Most attributes have a single value for a particular entity; such attributes are called **single-valued.** For example, Age is a single-valued attribute of a person. In some cases an attribute can have a set of values for the same entity—for example, a Colors attribute for a car, or a CollegeDegrees attribute for a person. Cars with one color have a single value, whereas two-tone cars have two values for Colors. Similarly, one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different persons can have different *numbers of values* for the CollegeDegrees attribute. Such attributes are called **multivalued.** A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may have between one and three values, if we assume that a car can have at most three colors.

Stored versus Derived Attributes. In some cases, two (or more) attribute values are related—for example, the Age and BirthDate attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's BirthDate. The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the BirthDate attribute, which is called a **stored attribute.** Some attribute values can be derived from *related entities;* for example, an attribute NumberOfEmployees of a department entity can be derived by counting the number of employees related to (working for) that department.

Null Values. In some cases a particular entity may not have an applicable value for an attribute. For example, the ApartmentNumber attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly, a CollegeDegrees attribute applies only to persons with college degrees. For such situations, a special value called **null** is created. An address of a single-family home would have null for its ApartmentNumber attribute, and a person with no college degree would have null for CollegeDegrees. Null can also be used if we do not know the value of an attribute for a particular entity—for example, if we do not know the home phone of "John Smith" in Figure 3.3. The meaning of the former type of null is *not applicable*, whereas the meaning of the latter is *unknown*. The "unknown" category of null can be further classified into two cases. The first case arises when it is known that the attribute value exists but is *missing*—for example, if the Height attribute of a person is listed as null. The second case arises when it is *not known* whether the attribute value exists—for example, if the HomePhone attribute of a person is null.

Complex Attributes. Notice that composite and multivalued attributes can be nested in an arbitrary way. We can represent arbitrary nesting by grouping components of

```
{AddressPhone( {Phone(AreaCode,PhoneNumber)},
Address(StreetAddress(Number,Street,ApartmentNumber),
                City,State,Zip) ) }
```

FIGURE 3.5 A complex attribute: AddressPhone

a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces {}. Such attributes are called **complex attributes.** For example, if a person can have more than one residence and each residence can have multiple phones, an attribute AddressPhone for a person can be specified as shown in Figure 3.5.[4]

3.3.2 Entity Types, Entity Sets, Keys, and Value Sets

Entity Types and Entity Sets. A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has *its own value(s)* for each attribute. An **entity type** defines a *collection* (or *set*) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. Figure 3.6 shows two entity types, named EMPLOYEE and COMPANY, and a list of attributes for each. A few individual entities of each type are also illustrated, along with the values of their attributes. The collection of all entities of a particular entity type in the database at any point in time is called an **entity set;** the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a *type of entity* as well as the current *set of all employee entities* in the database.

An entity type is represented in ER diagrams[5] (see Figure 3.2) as a rectangular box enclosing the entity type name. Attribute names are enclosed in ovals and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

An entity type describes the **schema** or **intension** for a *set of entities* that share the same structure. The collection of entities of a particular entity type are grouped into an entity set, which is also called the **extension** of the entity type.

Key Attributes of an Entity Type. An important constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes. An entity type usually has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute,** and its values can be used to identify each entity

4. For those familiar with XML, we should note here that complex attributes are similar to complex elements in XML (see Chapter 26).

5. We are using a notation for ER diagrams that is close to the original proposed notation (Chen 1976). Unfortunately, many other notations are in use. We illustrate some of the other notations in Appendix A and later in this chapter when we present UML class diagrams.

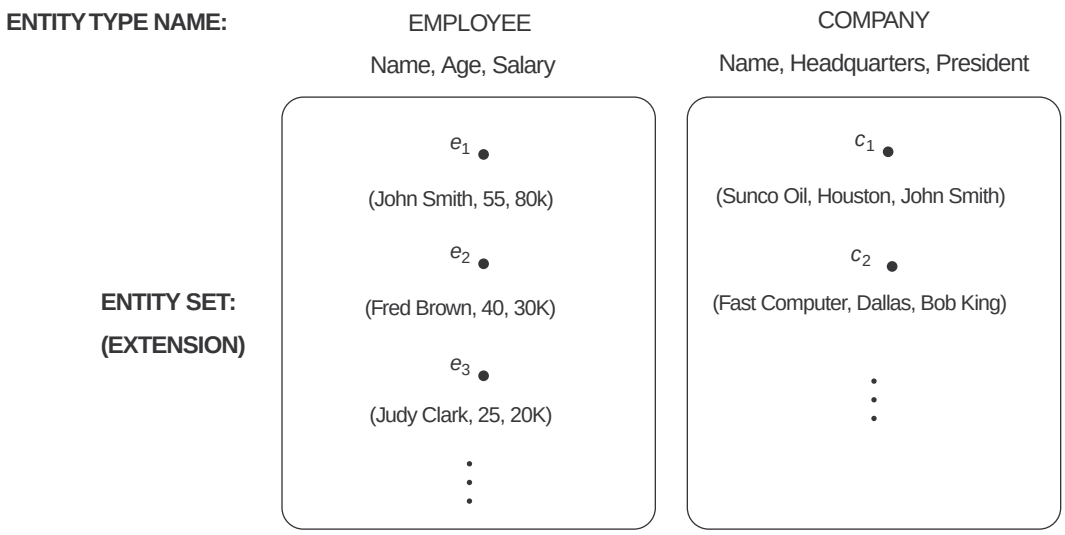

FIGURE 3.6 Two entity types, EMPLOYEE and COMPANY, and some member entities of each

uniquely. For example, the Name attribute is a key of the COMPANY entity type in Figure 3.6, because no two companies are allowed to have the same name. For the PERSON entity type, a typical key attribute is SocialSecurityNumber. Sometimes, several attributes together form a key, meaning that the *combination* of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model that we describe here is to define a *composite attribute* and designate it as a key attribute of the entity type. Notice that such a composite key must be *minimal;* that is, all component attributes must be included in the composite attribute to have the uniqueness property.[6] In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval, as illustrated in Figure 3.2.

Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for *every entity set* of the entity type. Hence, it is a constraint that prohibits any two entities from having the same value for the key attribute at the same time. It is not the property of a particular extension; rather, it is a constraint on *all extensions* of the entity type. This key constraint (and other constraints we discuss later) is derived from the constraints of the miniworld that the database represents.

Some entity types have *more than one* key attribute. For example, each of the VehicleID and Registration attributes of the entity type CAR (Figure 3.7) is a key in its own right. The Registration attribute is an example of a composite key formed from two simple component attributes, RegistrationNumber and State, neither of which is a key on its own. An entity type may also have *no key*, in which case it is called a *weak entity type* (see Section 3.5).

6. Superfluous attributes must not be included in a key; however, a **superkey** may include superfluous attributes, as explained in Chapter 5.

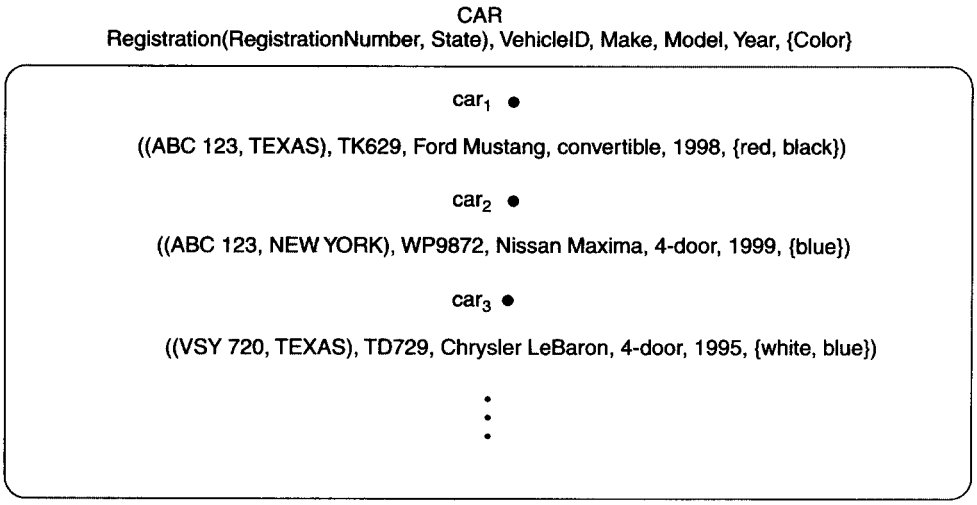

FIGURE 3.7 The CAR entity type with two key attributes, Registration and VehicleID

Value Sets (Domains) of Attributes. Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity. In Figure 3.6, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70. Similarly, we can specify the value set for the Name attribute as being the set of strings of alphabetic characters separated by blank characters, and so on. Value sets are not displayed in ER diagrams. Value sets are typically specified using the basic **data types** available in most programming languages, such as integer, string, boolean, float, enumerated type, subrange, and so on. Additional data types to represent date, time, and other concepts are also employed.

Mathematically, an attribute A of entity type E whose value set is V can be defined as a **function** from E to the power set[7] $P(V)$ of V:

$$A : E \rightarrow P(V)$$

We refer to the value of attribute A for entity e as $A(e)$. The previous definition covers both single-valued and multivalued attributes, as well as nulls. A null value is represented by the *empty set*. For single-valued attributes, $A(e)$ is restricted to being a *singleton set* for each entity e in E, whereas there is no restriction on multivalued attributes.[8] For a composite attribute A, the value set V is the Cartesian product of $P(V_1)$,

7. The **power set** $P(V)$ of a set V is the set of all subsets of V.

8. A **singleton set** is a set with only one element (value).

$P(V_2)$, . . . , $P(V_n)$, where V_1, V_2, . . . , V_n are the value sets of the simple component attributes that form A:

$$V = P(V_1) \times P(V_2) \times \ldots \times P(V_n)$$

3.3.3 Initial Conceptual Design of the COMPANY Database

We can now define the entity types for the COMPANY database, based on the requirements described in Section 3.2. After defining several entity types and their attributes here, we *refine* our design in Section 3.4 after we introduce the concept of a relationship. According to the requirements listed in Section 3.2, we can identify four entity types—one corresponding to each of the four items in the specification (see Figure 3.8):

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and ManagerStartDate. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes, because each was specified to be unique.
2. An entity type PROJECT with attributes Name, Number, Location, and ControllingDepartment. Both Name and Number are (separate) key attributes.
3. An entity type EMPLOYEE with attributes Name, SSN (for social security number), Sex, Address, Salary, BirthDate, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name—FirstName, MiddleInitial, LastName—or of Address.
4. An entity type DEPENDENT with attributes Employee, DependentName, Sex, BirthDate, and Relationship (to the employee).

DEPARTMENT
Name, Number, {Locations}, Manager, ManagerStartDate

PROJECT
Name, Number, Location, ControllingDepartment

EMPLOYEE
Name (FName, MInit, LName), SSN, Sex, Address, Salary,
BirthDate, Department, Supervisor, {WorksOn (Project, Hours)}

DEPENDENT
Employee, DependentName, Sex, BirthDate, Relationship

FIGURE 3.8 Preliminary design of entity types for the COMPANY database

So far, we have not represented the fact that an employee can work on several projects, nor have we represented the number of hours per week an employee works on each project. This characteristic is listed as part of requirement 3 in Section 3.2, and it can be represented by a multivalued composite attribute of EMPLOYEE called WorksOn with the simple components (Project, Hours). Alternatively, it can be represented as a multivalued composite attribute of PROJECT called Workers with the simple components (Employee, Hours). We choose the first alternative in Figure 3.8, which shows each of the entity types just described. The Name attribute of EMPLOYEE is shown as a composite attribute, presumably after consultation with the users.

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

In Figure 3.8 there are several *implicit relationships* among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department; the attribute ControllingDepartment of PROJECT refers to the department that controls the project; the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee); the attribute Department of EMPLOYEE refers to the department for which the employee works; and so on. In the ER model, these references should not be represented as attributes but as **relationships,** which are discussed in this section. The COMPANY database schema will be refined in Section 3.6 to represent relationships explicitly. In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

This section is organized as follows. Section 3.4.1 introduces the concepts of relationship types, relationship sets, and relationship instances. We then define the concepts of relationship degree, role names, and recursive relationships in Section 3.4.2, and discuss structural constraints on relationships—such as cardinality ratios and existence dependencies—in Section 3.4.3. Section 3.4.4 shows how relationship types can also have attributes.

3.4.1 Relationship Types, Sets, and Instances

A **relationship type** R among n entity types $E_1, E_2, \ldots, E_n$ defines a set of associations—or a **relationship set**—among entities from these entity types. As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the *same name*, R. Mathematically, the relationship set R is a set of **relationship instances** r_i, where each r_i associates n individual entities $(e_1, e_2, \ldots, e_n)$, and each entity e_j in r_i is a member of entity type E_j, $1 \leq j \leq n$. Hence, a relationship type is a mathematical relation on $E_1, E_2, \ldots, E_n$; alternatively, it can be defined as a subset of the Cartesian product $E_1 \times E_2 \times \ldots \times E_n$. Each of the entity types $E_1, E_2, \ldots, E_n$ is said to

participate in the relationship type R; similarly, each of the individual entities $e_1, e_2, \ldots, e_n$ is said to participate in the relationship instance $r_i = (e_1, e_2, \ldots, e_n)$.

Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type. Each such relationship instance r_i represents the fact that the entities participating in r_i are related in some way in the corresponding miniworld situation. For example, consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works. Each relationship instance in the relationship set WORKS_FOR associates one employee entity and one department entity. Figure 3.9 illustrates this example, where each relationship instance r_i is shown connected to the employee and department entities that participate in r_i. In the miniworld represented by Figure 3.9, employees e_1, e_3, and e_6 work for department d_1; e_2 and e_4 work for d_2; and e_5 and e_7 work for d_3.

In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types. The relationship name is displayed in the diamond-shaped box (see Figure 3.2).

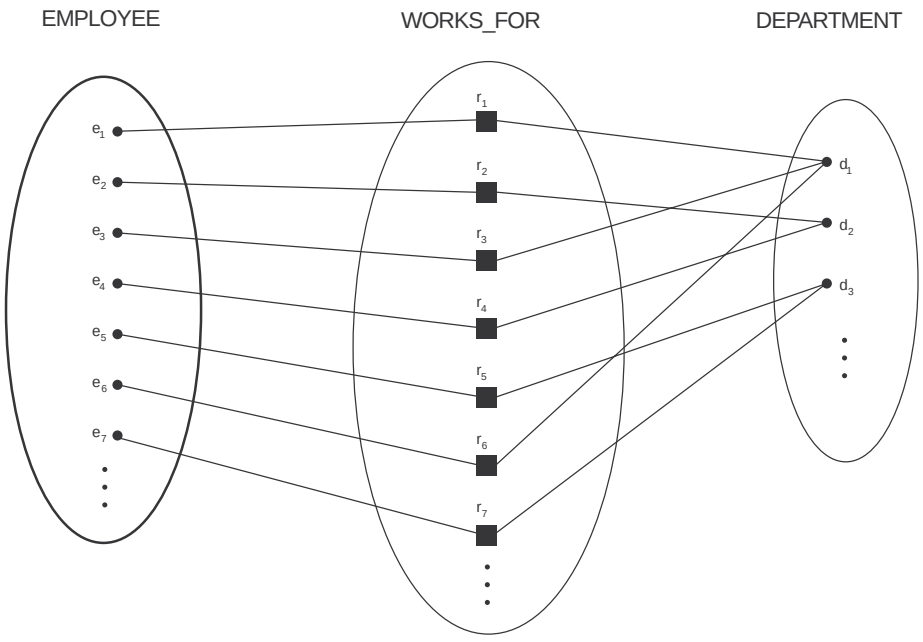

FIGURE 3.9 Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT

3.4.2 Relationship Degree, Role Names, and Recursive Relationships

Degree of a Relationship Type. The **degree** of a relationship type is the number of participating entity types. Hence, the WORKS_FOR relationship is of degree two. A relationship type of degree two is called **binary,** and one of degree three is called **ternary.** An example of a ternary relationship is SUPPLY, shown in Figure 3.10, where each relationship instance r_i associates three entities—a supplier s, a part p, and a project j—whenever s supplies part p to project j. Relationships can generally be of any degree, but the ones most common are binary relationships. Higher-degree relationships are generally more complex than binary relationships; we characterize them further in Section 4.7.

Relationships as Attributes. It is sometimes convenient to think of a relationship type in terms of attributes, as we discussed in Section 3.3.3. Consider the WORKS_FOR relationship type of Figure 3.9. One can think of an attribute called Department of the EMPLOYEE entity type whose value for each employee entity is (a reference to) the *department entity* that the employee works for. Hence, the value set for this Department attribute is the *set of all* DEPARTMENT *entities*, which is the DEPARTMENT entity set. This is what we did in Figure 3.8 when we specified the initial design of the entity type EMPLOYEE for the COMPANY database. However, when we think of a binary relationship as an attribute, we always have

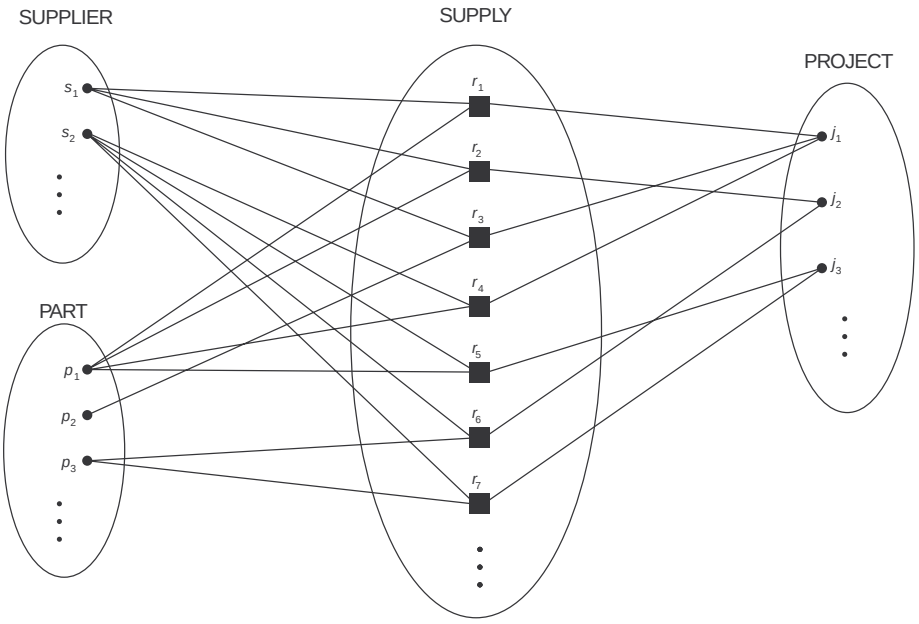

FIGURE 3.10 Some relationship instances in the SUPPLY ternary relationship set

two options. In this example, the alternative is to think of a multivalued attribute Employees of the entity type DEPARTMENT whose values for each department entity is the *set of employee entities* who work for that department. The value set of this Employees attribute is the power set of the EMPLOYEE entity set. Either of these two attributes—Department of EMPLOYEE or Employees of DEPARTMENT—can represent the WORKS_FOR relationship type. If both are represented, they are constrained to be inverses of each other.[9]

Role Names and Recursive Relationships. Each entity type that participates in a relationship type plays a particular **role** in the relationship. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of *employee* or *worker* and DEPARTMENT plays the role of *department* or *employer*.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the *same* entity type participates more than once in a relationship type in *different roles*. In such cases the role name becomes essential for distinguishing the meaning of each participation. Such relationship types are called **recursive relationships.** Figure 3.11 shows an example. The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity type. Hence, the EMPLOYEE entity type *participates twice* in SUPERVISION: once in the role of *supervisor* (or *boss*), and once in the role of *supervisee* (or *subordinate*). Each relationship instance r_i in SUPERVISION associates two employee entities e_j and e_k, one of which plays the role of supervisor and the other the role of supervisee. In Figure 3.11, the lines marked "1" represent the supervisor role, and those marked "2" represent the supervisee role; hence, e_1 supervises e_2 and e_3, e_4 supervises e_6 and e_7, and e_5 supervises e_1 and e_4.

3.4.3 Constraints on Relationship Types

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the miniworld situation that the relationships represent. For example, in Figure 3.9, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. We can distinguish two main types of relationship constraints: *cardinality ratio* and *participation*.

9. This concept of representing relationship types as attributes is used in a class of data models called **functional data models.** In object databases (see Chapter 20), relationships can be represented by reference attributes, either in one direction or in both directions as inverses. In relational databases (see Chapter 5), foreign keys are a type of reference attribute used to represent relationships.

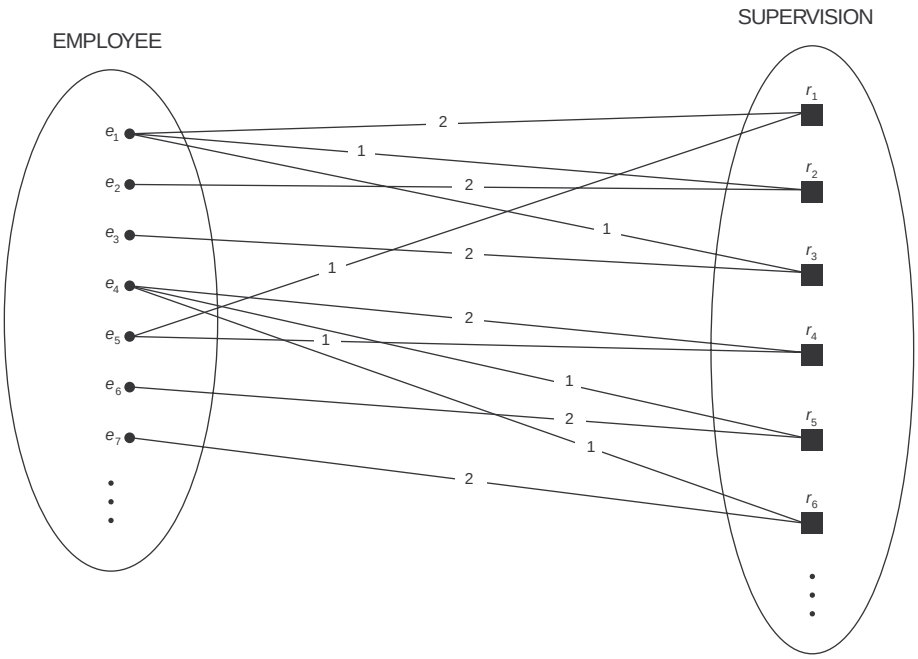

FIGURE 3.11 A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2)

Cardinality Ratios for Binary Relationships. The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees,[10] but an employee can be related to (work for) only one department. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

An example of a 1:1 binary relationship is MANAGES (Figure 3.12), which relates a department entity to the employee who manages that department. This represents the miniworld constraints that—at any point in time—an employee can manage only one department and a department has only one manager. The relationship type WORKS_ON (Figure 3.13 is of cardinality ratio M:N, because the miniworld rule is that an employee can work on several projects and a project can have several employees.

Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds as shown in Figure 3.2.

10. N stands for *any number* of related entities (zero or more).

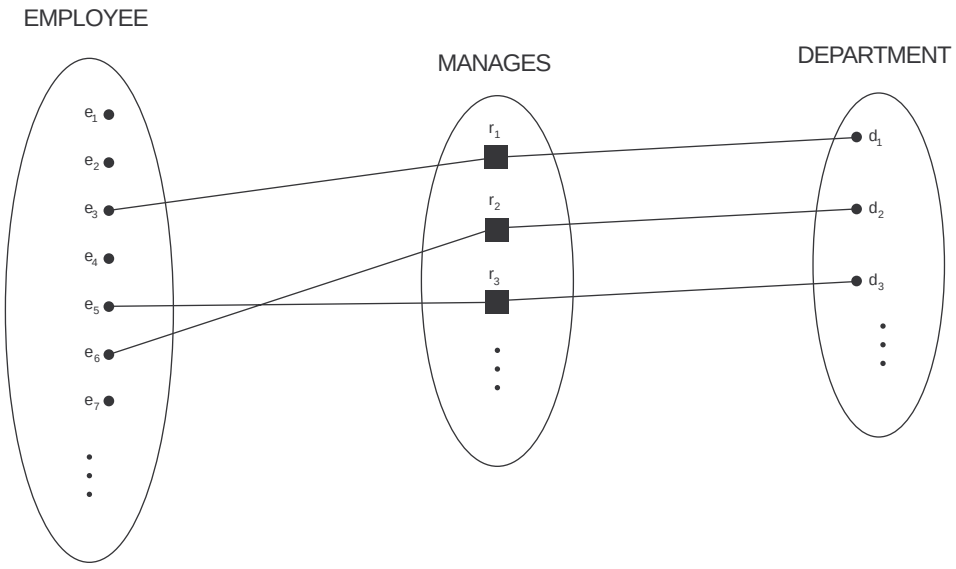

FIGURE 3.12 A 1:1 relationship, MANAGES

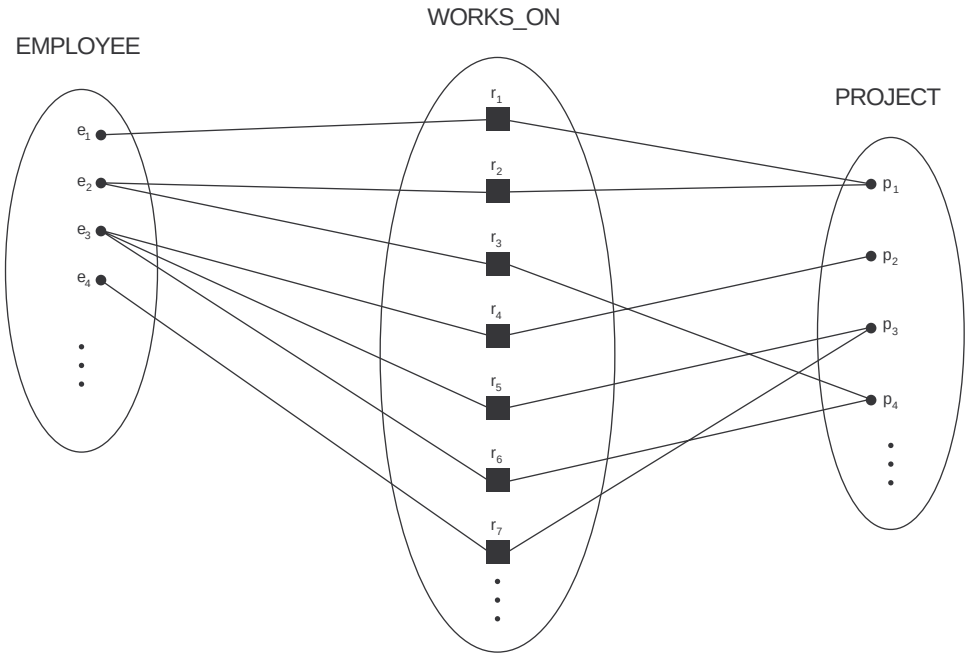

FIGURE 3.13 An M:N relationship, WORKS_ON

Participation Constraints and Existence Dependencies. The **participation constraint** specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the *minimum* number of relationship instances that each entity can participate in, and is sometimes called the **minimum cardinality constraint.** There are two types of participation constraints—total and partial—which we illustrate by example. If a company policy states that *every* employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance (Figure 3.9). Thus, the participation of EMPLOYEE in WORKS_FOR is called **total participation,** meaning that every entity in "the total set" of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency.** In Figure 3.12 we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial,** meaning that *some* or "part of the set of" employee entities are related to some department entity via MANAGES, but not necessarily all. We will refer to the cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.

In ER diagrams, total participation (or existence dependency) is displayed as a *double line* connecting the participating entity type to the relationship, whereas partial participation is represented by a *single line* (see Figure 3.2).

3.4.4 Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type of Figure 3.13. Another example is to include the date on which a manager started managing a department via an attribute StartDate for the MANAGES relationship type of Figure 3.12.

Notice that attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types. For example, the StartDate attribute for the MANAGES relationship can be an attribute of either EMPLOYEE or DEPARTMENT, although conceptually it belongs to MANAGES. This is because MANAGES is a 1:1 relationship, so every department or employee entity participates in *at most one* relationship instance. Hence, the value of the StartDate attribute can be determined separately, either by the participating department entity or by the participating employee (manager) entity.

For a 1:N relationship type, a relationship attribute can be migrated *only* to the entity type on the N-side of the relationship. For example, in Figure 3.9, if the WORKS_FOR relationship also has an attribute StartDate that indicates when an employee started working for a department, this attribute can be included as an attribute of EMPLOYEE. This is because each employee works for only one department, and hence participates in at most one relationship instance in WORKS_FOR. In both 1:1 and 1:N relationship types, the decision as to where a relationship attribute should be placed—as a relationship type attribute or as an attribute of a participating entity type—is determined subjectively by the schema designer.

For M:N relationship types, some attributes may be determined by the *combination of participating entities* in a relationship instance, not by any single entity. Such attributes

must be specified as relationship attributes. An example is the Hours attribute of the M:N relationship WORKS_ON (Figure 3.13); the number of hours an employee works on a project is determined by an employee-project combination and not separately by either entity.

3.5 Weak Entity Types

Entity types that do not have key attributes of their own are called **weak entity types.** In contrast, **regular entity types** that do have a key attribute—which include all the examples we discussed so far—are called **strong entity types.** Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the **identifying** or **owner entity type,**[11] and we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.[12] A weak entity type always has a *total participation constraint* (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity. However, not every existence dependency results in a weak entity type. For example, a DRIVER_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key (LicenseNumber) and hence is not a weak entity.

Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1:N relationship (Figure 3.2). The attributes of DEPENDENT are Name (the first name of the dependent), BirthDate, Sex, and Relationship (to the employee). Two dependents of *two distinct employees* may, by chance, have the same values for Name, BirthDate, Sex, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the *particular employee entity* to which each dependent is related. Each employee entity is said to **own** the dependent entities that are related to it.

A weak entity type normally has a **partial key,** which is the set of attributes that can uniquely identify weak entities that are *related to the same owner entity*.[13] In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key. In the worst case, a composite attribute of *all the weak entity's attributes* will be the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines (see Figure 3.2). The partial key attribute is underlined with a dashed or dotted line.

Weak entity types can sometimes be represented as complex (composite, multivalued) attributes. In the preceding example, we could specify a multivalued attribute Dependents for EMPLOYEE, which is a composite attribute with component attributes Name, BirthDate,

11. The identifying entity type is also sometimes called the **parent entity type** or the **dominant entity type.**

12. The weak entity type is also sometimes called the **child entity type** or the **subordinate entity type.**

13. The partial key is sometimes called the **discriminator.**

Sex, and Relationship. The choice of which representation to use is made by the database designer. One criterion that may be used is to choose the weak entity type representation if there are many attributes. If the weak entity participates independently in relationship types other than its identifying relationship type, then it should *not* be modeled as a complex attribute.

In general, any number of levels of weak entity types can be defined; an owner entity type may itself be a weak entity type. In addition, a weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two, as we illustrate in Section 4.7.

3.6 Refining the ER Design for the COMPANY Database

We can now refine the database design of Figure 3.8 by changing the attributes that represent relationships into relationship types. The cardinality ratio and participation constraint of each relationship type are determined from the requirements listed in Section 3.2. If some cardinality ratio or dependency cannot be determined from the requirements, the users must be questioned further to determine these structural constraints.

In our example, we specify the following relationship types:

1. MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation.[14] The attribute StartDate is assigned to this relationship type.
2. WORKS_FOR, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
3. CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.
4. SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
5. WORKS_ON, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.

14. The rules in the miniworld that determine the constraints are sometimes called the *business rules*, since they are determined by the "business" or organization that will utilize the database.

6. DEPENDENTS_OF, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

After specifying the above six relationship types, we remove from the entity types in Figure 3.8 all attributes that have been refined into relationships. These include Manager and ManagerStartDate from DEPARTMENT; ControllingDepartment from PROJECT; Department, Supervisor, and WorksOn from EMPLOYEE; and Employee from DEPENDENT. It is important to have the least possible redundancy when we design the conceptual schema of a database. If some redundancy is desired at the storage level or at the user view level, it can be introduced later, as discussed in Section 1.6.1.

3.7 ER Diagrams, Naming Conventions, and Design Issues

3.7.1 Summary of Notation for ER Diagrams

Figures 3.9 through 3.13 illustrate examples of the participation of entity types in relationship types by displaying their extensions—the individual entity instances and relationship instances in the entity sets and relationship sets. In ER diagrams the emphasis is on representing the schemas rather than the instances. This is more useful in database design because a database schema changes rarely, whereas the contents of the entity sets change frequently. In addition, the schema is usually easier to display than the extension of a database, because it is much smaller.

Figure 3.2 displays the COMPANY **ER database schema** as an **ER diagram.** We now review the full ER diagram notation. Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes. Relationship types such as WORKS_FOR, MANAGES, CONTROLS, and WORKS_ON are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE. Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT. Key attributes have their names underlined. Derived attributes are shown in dotted ovals, as illustrated by the NumberOfEmployees attribute of DEPARTMENT.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS_OF identifying relationship type. The partial key of the weak entity type is underlined with a dotted line.

In Figure 3.2 the cardinality ratio of each *binary* relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas 1:N for DEPARTMENT:EMPLOYEE in WORKS_FOR, and M:N for WORKS_ON. The

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

In Figure 3.2 we show the role names for the SUPERVISION relationship type because the EMPLOYEE entity type plays both roles in that relationship. Notice that the cardinality is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees.

Figure 3.14 summarizes the conventions for ER diagrams.

3.7.2 Proper Naming of Schema Constructs

When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use *singular names* for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type. In our ER diagrams, we will use the convention that entity type and relationship type names are in uppercase letters, attribute names are capitalized, and role names are in lowercase letters. We have already used this convention in Figure 3.2.

As a general practice, given a narrative description of the database requirements, the *nouns* appearing in the narrative tend to give rise to entity type names, and the *verbs* tend to indicate names of relationship types. Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.

Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom. We have generally followed this guideline in Figure 3.2. To explain this naming convention further, we have one exception to the convention in Figure 3.2—the DEPENDENTS_OF relationship type, which reads from bottom to top. When we describe this relationship, we can say that the DEPENDENT entities (bottom entity type) are DEPENDENTS_OF (relationship name) an EMPLOYEE (top entity type). To change this to read from top to bottom, we could rename the relationship type to HAS_DEPENDENTS, which would then read as follows: An EMPLOYEE entity (top entity type) HAS_DEPENDENTS (relationship name) of type DEPENDENT (bottom entity type). Notice that this issue arises because each binary relationship can be described starting from either of the two participating entity types, as discussed in the beginning of Section 3.4.

3.7.3 Design Choices for ER Conceptual Design

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this section, we give some brief guidelines as to which construct should be chosen in particular situations.

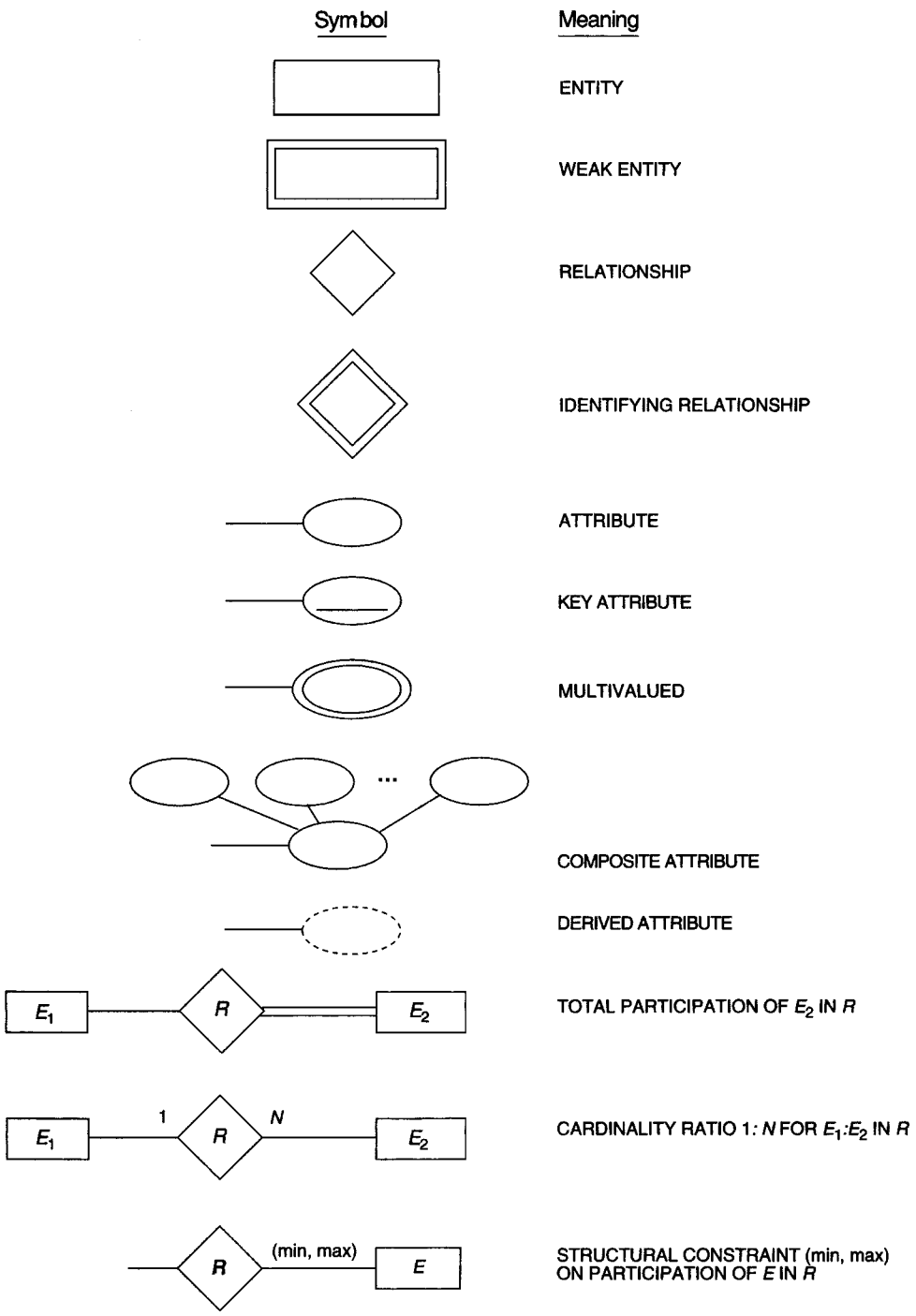

FIGURE 3.14 Summary of the notation for ER diagrams

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that a pair of such attributes that are inverses of one another are refined into a binary relationship. We discussed this type of refinement in detail in Section 3.6.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute DeptName and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships. Other attributes/relationships of DEPARTMENT may be discovered later.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute DeptName and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.
- In Chapter 4, we discuss other refinements concerning specialization/generalization and relationships of higher degree. Chapter 12 discusses additional top-down and bottom-up refinements that are common in large-scale conceptual schema design.

3.7.4 Alternative Notations for ER Diagrams

There are many alternative diagrammatic notations for displaying ER diagrams. Appendix A gives some of the more popular notations. In Section 3.8, we introduce the Universal Modeling Language (UML) notation for class diagrams, which has been proposed as a standard for conceptual object modeling.

In this section, we describe one alternative ER notation for specifying structural constraints on relationships. This notation involves associating a pair of integer numbers (min, max) with each *participation* of an entity type E in a relationship type R, where $0 \leq \text{min} \leq \text{max}$ and $\text{max} \geq 1$. The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R *at any point in time*. In this method, $\text{min} = 0$ implies partial participation, whereas $\text{min} > 0$ implies total participation.

Figure 3.15 displays the COMPANY database schema using the (min, max) notation.[15] Usually, one uses either the cardinality ratio/single-line/double-line notation *or* the (min,

15. In some notations, particularly those used in object modeling methodologies such as UML, the (min, max) is placed on the *opposite sides* to the ones we have shown. For example, for the WORKS_FOR relationship in Figure 3.15, the (1,1) would be on the DEPARTMENT side, and the (4,N) would be on the EMPLOYEE side. Here we used the original notation from Abrial (1974).

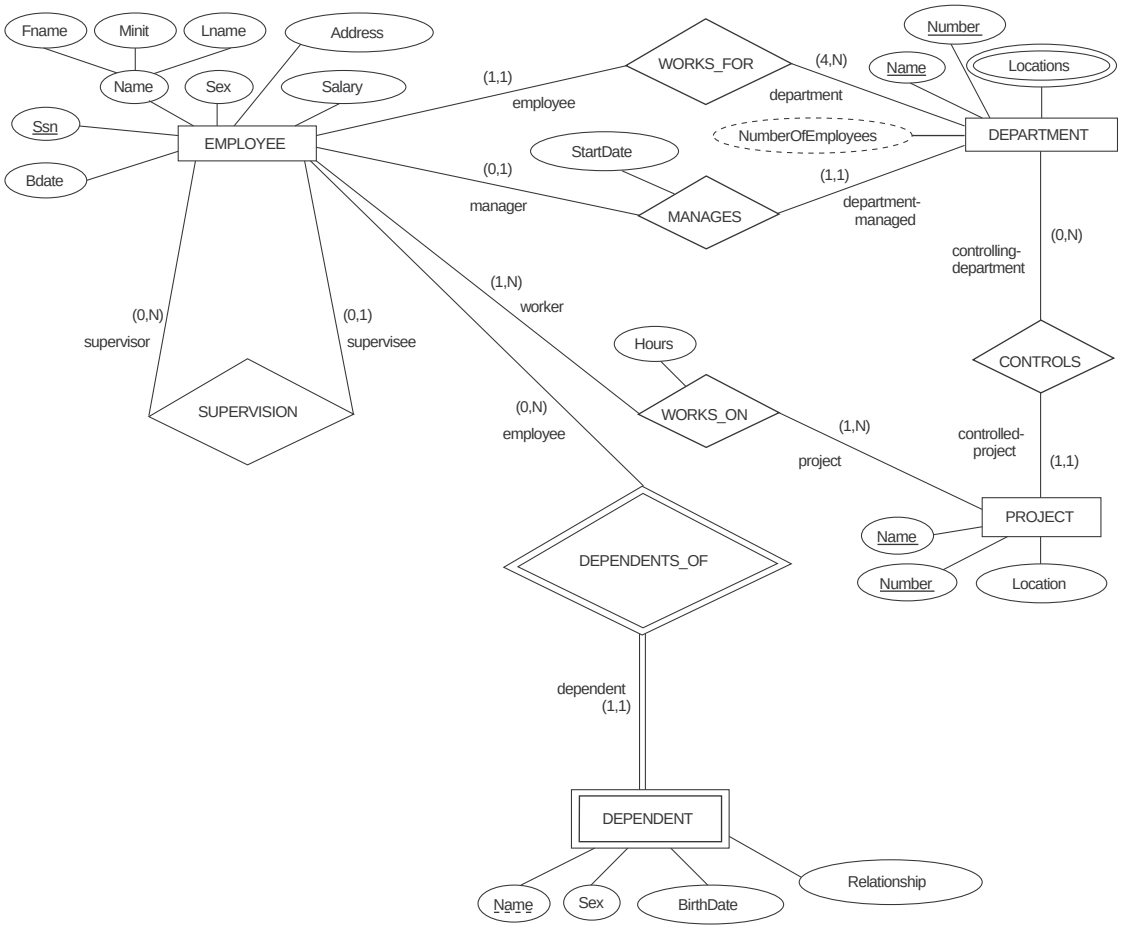

FIGURE 3.15 ER diagrams for the COMPANY schema, with structural constraints specified using (min, max) notation

max) notation. The (min, max) notation is more precise, and we can use it easily to specify structural constraints for relationship types of *any degree*. However, it is not sufficient for specifying some key constraints on higher-degree relationships, as discussed in Section 4.7.

Figure 3.15 also displays all the role names for the COMPANY database schema.

3.8 NOTATION FOR UML CLASS DIAGRAMS

The UML methodology is being used extensively in software design and has many types of diagrams for various software design purposes. We only briefly present the basics of **UML**

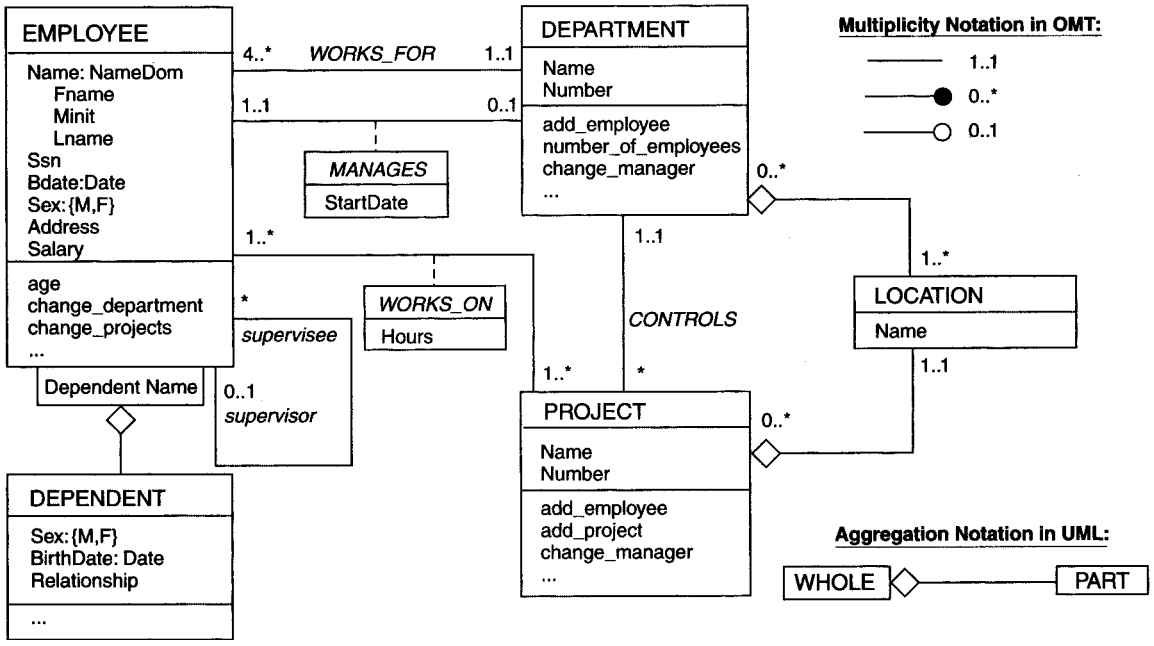

FIGURE 3.16 The COMPANY conceptual schema in UML class diagram notation

class diagrams here, and compare them with ER diagrams. In some ways, class diagrams can be considered as an alternative notation to ER diagrams. Additional UML notation and concepts are presented in Section 4.6, and in Chapter 12. Figure 3.16 shows how the COMPANY ER database schema of Figure 3.15 can be displayed using UML class diagram notation. The *entity types* in Figure 3.15 are modeled as *classes* in Figure 3.16. An *entity* in ER corresponds to an *object* in UML.

In UML class diagrams, a **class** is displayed as a box (see Figure 3.16) that includes three sections: The top section gives the **class name,** the middle section includes the **attributes** for individual objects of the class; and the last section includes **operations** that can be applied to these objects. Operations are *not* specified in ER diagrams. Consider the EMPLOYEE class in Figure 3.16. Its attributes are Name, Ssn, Bdate, Sex, Address, and Salary. The designer can optionally specify the **domain** of an attribute if desired, by placing a colon (:) followed by the domain name or description, as illustrated by the Name, Sex, and Bdate attributes of EMPLOYEE in Figure 3.16. A composite attribute is modeled as a **structured domain,** as illustrated by the Name attribute of EMPLOYEE. A multivalued attribute will generally be modeled as a separate class, as illustrated by the LOCATION class in Figure 3.16.

Relationship types are called **associations** in UML terminology, and relationship instances are called **links.** A **binary association** (binary relationship type) is represented as a line connecting the participating classes (entity types), and may optionally have a

name. A relationship attribute, called a **link attribute,** is placed in a box that is connected to the association's line by a dashed line. The (min, max) notation described in Section 3.7.4 is used to specify relationship constraints, which are called **multiplicities** in UML terminology. Multiplicities are specified in the form *min..max*, and an asterisk (*) indicates no maximum limit on participation. However, the multiplicities are placed *on the opposite ends of the relationship* when compared with the notation discussed in Section 3.7.4 (compare Figures 3.16 and 3.15). In UML, a single asterisk indicates a multiplicity of 0..*, and a single 1 indicates a multiplicity of 1..1. A recursive relationship (see Section 3.4.2) is called a **reflexive association** in UML, and the role names—like the multiplicities—are placed at the opposite ends of an association when compared with the placing of role names in Figure 3.15.

In UML, there are two types of relationships: association and aggregation. **Aggregation** is meant to represent a relationship between a whole object and its component parts, and it has a distinct diagrammatic notation. In Figure 3.16, we modeled the locations of a department and the single location of a project as aggregations. However, aggregation and association do not have different structural properties, and the choice as to which type of relationship to use is somewhat subjective. In the ER model, both are represented as relationships.

UML also distinguishes between **unidirectional** and **bidirectional** associations (or aggregations). In the unidirectional case, the line connecting the classes is displayed with an arrow to indicate that only one direction for accessing related objects is needed. If no arrow is displayed, the bidirectional case is assumed, which is the default. For example, if we always expect to access the manager of a department starting from a DEPARTMENT object, we would draw the association line representing the MANAGES association with an arrow from DEPARTMENT to EMPLOYEE. In addition, relationship instances may be specified to be **ordered.** For example, we could specify that the employee objects related to each department through the WORKS_FOR association (relationship) should be ordered by their Bdate attribute value. Association (relationship) names are *optional* in UML, and relationship attributes are displayed in a box attached with a dashed line to the line representing the association/aggregation (see StartDate and Hours in Figure 3.16).

The operations given in each class are derived from the functional requirements of the application, as we discussed in Section 3.1. It is generally sufficient to specify the operation names initially for the logical operations that are expected to be applied to individual objects of a class, as shown in Figure 3.16. As the design is refined, more details are added, such as the exact argument types (parameters) for each operation, plus a functional description of each operation. UML has *function descriptions* and *sequence diagrams* to specify some of the operation details, but these are beyond the scope of our discussion. Chapter 12 will introduce some of these diagrams.

Weak entities can be modeled using the construct called **qualified association** (or **qualified aggregation**) in UML; this can represent both the identifying relationship and the partial key, which is placed in a box attached to the owner class. This is illustrated by the DEPENDENT class and its qualified aggregation to EMPLOYEE in Figure 3.16. The partial key DependentName is called the **discriminator** in UML terminology, since its value distinguishes the objects associated with (related to) the same EMPLOYEE. Qualified associations are not restricted to modeling weak entities, and they can be used to model other situations in UML.

3.9 SUMMARY

In this chapter we presented the modeling concepts of a high-level conceptual data model, the Entity-Relationship (ER) model. We started by discussing the role that a high-level data model plays in the database design process, and then we presented an example set of database requirements for the COMPANY database, which is one of the examples that is used throughout this book. We then defined the basic ER model concepts of entities and their attributes. We discussed null values and presented the various types of attributes, which can be nested arbitrarily to produce complex attributes:

- Simple or atomic
- Composite
- Multivalued

We also briefly discussed stored versus derived attributes. We then discussed the ER model concepts at the schema or "intension" level:

- Entity types and their corresponding entity sets
- Key attributes of entity types
- Value sets (domains) of attributes
- Relationship types and their corresponding relationship sets
- Participation roles of entity types in relationship types

We presented two methods for specifying the structural constraints on relationship types. The first method distinguished two types of structural constraints:

- Cardinality ratios (1:1, 1:N, M:N for binary relationships)
- Participation constraints (total, partial)

We noted that, alternatively, another method of specifying structural constraints is to specify minimum and maximum numbers (min, max) on the participation of each entity type in a relationship type. We discussed weak entity types and the related concepts of owner entity types, identifying relationship types, and partial key attributes.

Entity-Relationship schemas can be represented diagrammatically as ER diagrams. We showed how to design an ER schema for the COMPANY database by first defining the entity types and their attributes and then refining the design to include relationship types. We displayed the ER diagram for the COMPANY database schema. Finally, we discussed some of the basic concepts of UML class diagrams and how they relate to ER model concepts.

The ER modeling concepts we have presented thus far—entity types, relationship types, attributes, keys, and structural constraints—can model traditional business data-processing database applications. However, many newer, more complex applications—such as engineering design, medical information systems, or telecommunications—require additional concepts if we want to model them with greater accuracy. We discuss these advanced modeling concepts in Chapter 4. We also describe ternary and higher-degree relationship types in more detail in Chapter 4, and discuss the circumstances under which they are distinguished from binary relationships.

Review Questions

3.1. Discuss the role of a high-level data model in the database design process.
3.2. List the various cases where use of a null value would be appropriate.
3.3. Define the following terms: *entity*, *attribute*, *attribute value*, *relationship instance*, *composite attribute*, *multivalued attribute*, *derived attribute*, *complex attribute*, *key attribute*, *value set (domain)*.
3.4. What is an entity type? What is an entity set? Explain the differences among an entity, an entity type, and an entity set.
3.5. Explain the difference between an attribute and a value set.
3.6. What is a relationship type? Explain the differences among a relationship instance, a relationship type, and a relationship set.
3.7. What is a participation role? When is it necessary to use role names in the description of relationship types?
3.8. Describe the two alternatives for specifying structural constraints on relationship types. What are the advantages and disadvantages of each?
3.9. Under what conditions can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity types?
3.10. When we think of relationships as attributes, what are the value sets of these attributes? What class of data models is based on this concept?
3.11. What is meant by a recursive relationship type? Give some examples of recursive relationship types.
3.12. When is the concept of a weak entity used in data modeling? Define the terms *owner entity type*, *weak entity type*, *identifying relationship type*, and *partial key*.
3.13. Can an identifying relationship of a weak entity type be of a degree greater than two? Give examples to illustrate your answer.
3.14. Discuss the conventions for displaying an ER schema as an ER diagram.
3.15. Discuss the naming conventions used for ER schema diagrams.

Exercises

3.16. Consider the following set of requirements for a university database that is used to keep track of students' transcripts. This is similar but not identical to the database shown in Figure 1.2:
   a. The university keeps track of each student's name, student number, social security number, current address and phone, permanent address and phone, birthdate, sex, class (freshman, sophomore, . . . , graduate), major department, minor department (if any), and degree program (B.A., B.S., . . . , Ph.D.). Some user applications need to refer to the city, state, and zip code of the student's permanent address and to the student's last name. Both social security number and student number have unique values for each student.
   b. Each department is described by a name, department code, office number, office phone, and college. Both name and code have unique values for each department.

c. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.
d. Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, . . . , up to the number of sections taught during each semester.
e. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).

Design an ER schema for this application, and draw an ER diagram for that schema. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

3.17. Composite and multivalued attributes can be nested to any number of levels. Suppose we want to design an attribute for a STUDENT entity type to keep track of previous college education. Such an attribute will have one entry for each college previously attended, and each such entry will be composed of college name, start and end dates, degree entries (degrees awarded at that college, if any), and transcript entries (courses completed at that college, if any). Each degree entry contains the degree name and the month and year the degree was awarded, and each transcript entry contains a course name, semester, year, and grade. Design an attribute to hold this information. Use the conventions of Figure 3.5.

3.18. Show an alternative design for the attribute described in Exercise 3.17 that uses only entity types (including weak entity types, if needed) and relationship types.

3.19. Consider the ER diagram of Figure 3.17, which shows a simplified schema for an airline reservations system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirements and constraints specification.

3.20. In Chapters 1 and 2, we discussed the database environment and database users. We can consider many entity types to describe such an environment, such as DBMS, stored database, DBA, and catalog/data dictionary. Try to specify all the entity types that can fully describe a database system and its environment; then specify the relationship types among them, and draw an ER diagram to describe such a general database environment.

3.21. Design an ER schema for keeping track of information about votes taken in the U.S. House of Representatives during the current two-year congressional session. The database needs to keep track of each U.S. STATE's Name (e.g., Texas, New York, California) and include the Region of the state (whose domain is {Northeast, Midwest, Southeast, Southwest, West}). Each CONGRESSPERSON in the House of Representatives is described by his or her Name, plus the District represented, the StartDate when the congressperson was first elected, and the political Party to which he or she belongs (whose domain is {Republican, Democrat, Independent, Other}). The database keeps track of each BILL (i.e., proposed law), including the BillName, the DateOfVote on the bill, whether the bill PassedOrFailed (whose domain is {Yes, No}), and the Sponsor (the congressperson(s) who sponsored—

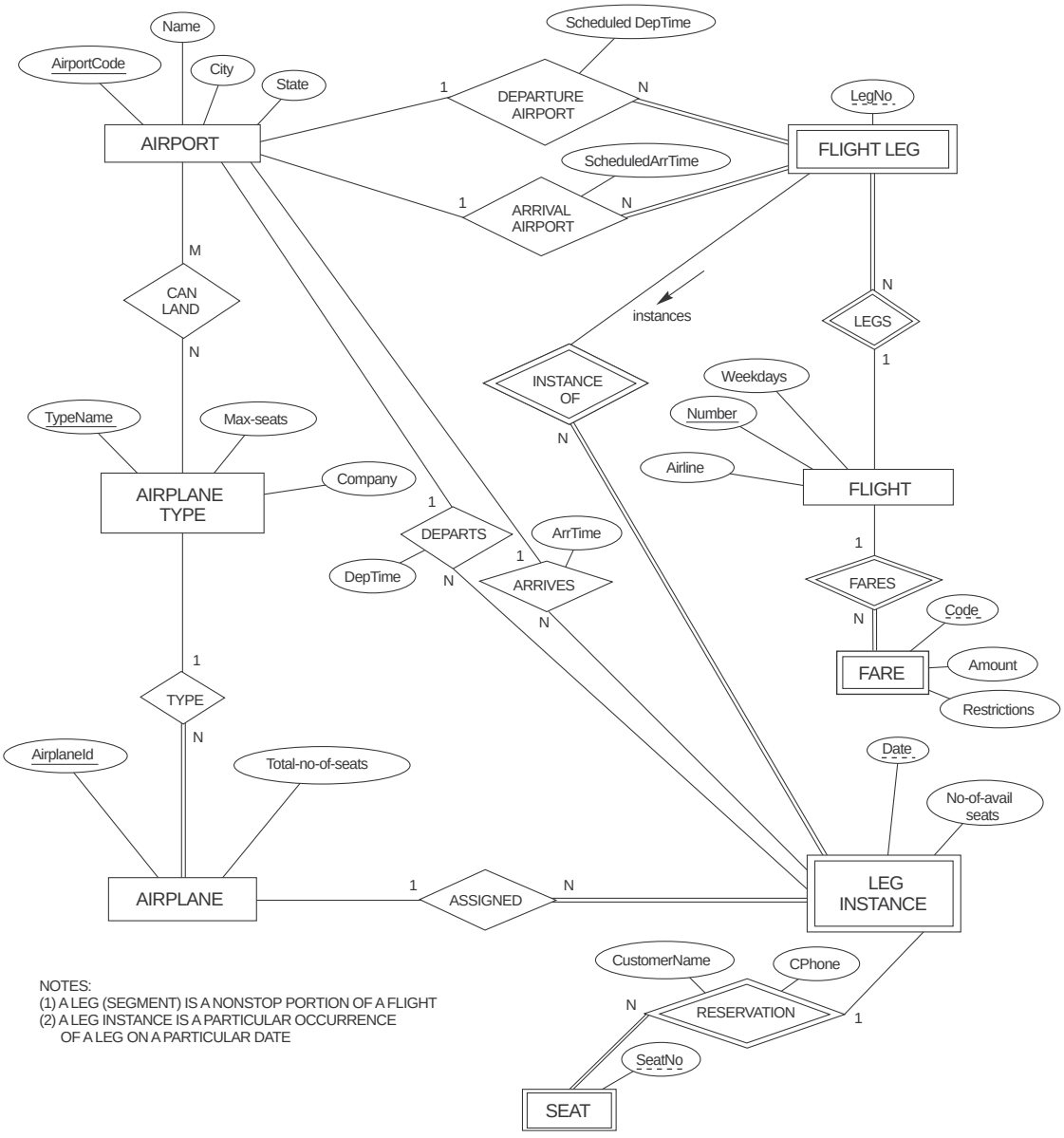

FIGURE 3.17 An ER diagram for an AIRLINE database schema

that is, proposed—the bill). The database keeps track of how each congressperson voted on each bill (domain of vote attribute is {Yes, No, Abstain, Absent}). Draw an ER schema diagram for this application. State clearly any assumptions you make.

3.22. A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Design an ER schema diagram for this application, stating any assumptions you make. Choose your favorite sport (e.g., soccer, baseball, football).

3.23. Consider the ER diagram shown in Figure 3.18 for part of a BANK database. Each bank can have multiple branches, and each branch can have multiple accounts and loans.

a. List the (nonweak) entity types in the ER diagram.
b. Is there a weak entity type? If so, give its name, partial key, and identifying relationship.
c. What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?
d. List the names of all relationship types, and specify the (min, max) constraint on each participation of an entity type in a relationship type. Justify your choices.
e. List concisely the user requirements that led to this ER schema design.
f. Suppose that every customer must have at least one account but is restricted to at most two loans at a time, and that a bank branch cannot have more than 1000 loans. How does this show up on the (min, max) constraints?

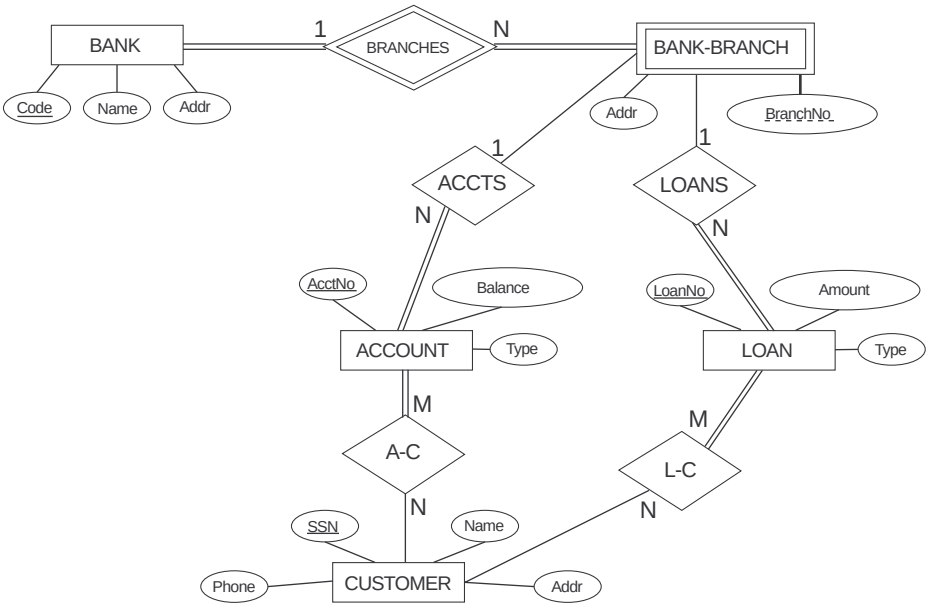

FIGURE 3.18 An ER diagram for a BANK database schema

3.24. Consider the ER diagram in Figure 3.19. Assume that an employee may work in up to two departments or may not be assigned to any department. Assume that each department must have one and may have up to three phone numbers. Supply (min, max) constraints on this diagram. *State clearly any additional assumptions you make.* Under what conditions would the relationship HAS_PHONE be redundant in this example?

3.25. Consider the ER diagram in Figure 3.20. Assume that a course may or may not use a textbook, but that a text by definition is a book that is used in some course. A course may not use more than five books. Instructors teach from two to four courses. Supply (min, max) constraints on this diagram. *State clearly any additional assumptions you make.* If we add the relationship ADOPTS between INSTRUCTOR and TEXT, what (min, max) constraints would you put on it? Why?

3.26. Consider an entity type SECTION in a UNIVERSITY database, which describes the section offerings of courses. The attributes of SECTION are SectionNumber, Semester, Year, CourseNumber, Instructor, RoomNo (where section is taught), Building (where section is taught), Weekdays (domain is the possible combinations of weekdays in which a section can be offered {MWF, MW, TT, etc.}), and Hours (domain is all possible time periods during which sections are offered {9–9:50 A.M., 10–10:50 A.M., . . . , 3:30–4:50 P.M., 5:30–6:20 P.M., etc.}). Assume that SectionNumber is

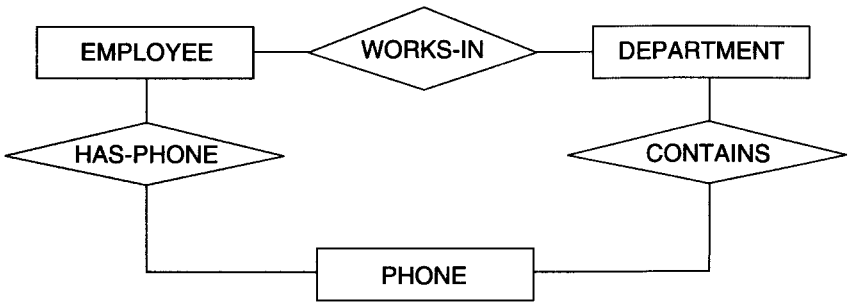

FIGURE 3.19 Part of an ER diagram for a COMPANY database

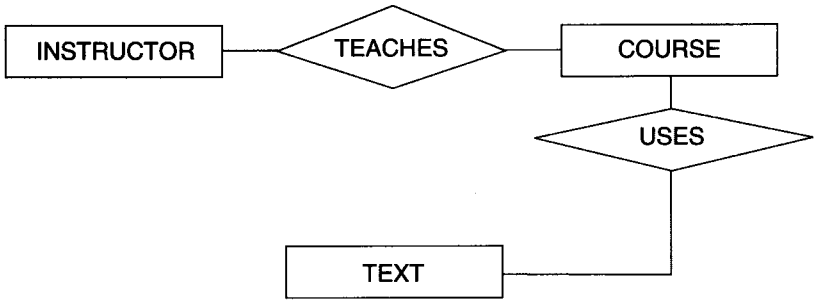

FIGURE 3.20 Part of an ER diagram for a COURSES database

unique for each course within a particular semester/year combination (that is, if a course is offered multiple times during a particular semester, its section offerings are numbered 1, 2, 3, etc.). There are several composite keys for SECTION, and some attributes are components of more than one key. Identify three composite keys, and show how they can be represented in an ER schema diagram.

Selected Bibliography

The Entity-Relationship model was introduced by Chen (1976), and related work appears in Schmidt and Swenson (1975), Wiederhold and Elmasri (1979), and Senko (1975). Since then, numerous modifications to the ER model have been suggested. We have incorporated some of these in our presentation. Structural constraints on relationships are discussed in Abrial (1974), Elmasri and Wiederhold (1980), and Lenzerini and Santucci (1983). Multivalued and composite attributes are incorporated in the ER model in Elmasri et al. (1985). Although we did not discuss languages for the entity-relationship model and its extensions, there have been several proposals for such languages. Elmasri and Wiederhold (1981) proposed the GORDAS query language for the ER model. Another ER query language was proposed by Markowitz and Raz (1983). Senko (1980) presented a query language for Senko's DIAM model. A formal set of operations called the ER algebra was presented by Parent and Spaccapietra (1985). Gogolla and Hohenstein (1991) presented another formal language for the ER model. Campbell et al. (1985) presented a set of ER operations and showed that they are relationally complete. A conference for the dissemination of research results related to the ER model has been held regularly since 1979. The conference, now known as the International Conference on Conceptual Modeling, has been held in Los Angeles (ER 1979, ER 1983, ER 1997), Washington, D.C. (ER 1981), Chicago (ER 1985), Dijon, France (ER 1986), New York City (ER 1987), Rome (ER 1988), Toronto (ER 1989), Lausanne, Switzerland (ER 1990), San Mateo, California (ER 1991), Karlsruhe, Germany (ER 1992), Arlington, Texas (ER 1993), Manchester, England (ER 1994), Brisbane, Australia (ER 1995), Cottbus, Germany (ER 1996), Singapore (ER 1998), Salt Lake City, Utah (ER 1999), Yokohama, Japan (ER 2001), and Tampere, Finland (ER 2002). The next conference is scheduled for Chicago in October 2003.