

## 5 Database Development

### Introduction to Entity Relationship Diagrams

Gaining an initial understanding of entity relationship diagrams (ERDs) requires careful study. This section introduces the Crow's Foot notation for ERDs, a popular notation supported by many CASE tools. To get started, let's begin with the basic symbols of entity types, relationships, and attributes. Then go to explain cardinalities and their appearance in the Crow's Foot notation. Finally concluded by comparing the Crow's Foot notation to relational database diagrams.

---

**Entity Type** a collection of entities (persons, places, events, or things) of interest in an application, represented by a rectangle in an entity relationship diagram.

---

### Basic Symbols

ERDs have three basic elements: entity types, relationships, and attributes. Entity types (also known as object types) are collections of things of interest (entities) in an application. Entity types can represent physical objects such as books, people, and places, as well as events such as payments. Entities are uniquely identified to allow tracking across business processes. For example, customers have a unique identification to support order processing, shipment, and product warranty processes. In the Crow's Foot notation as well as most other notations, rectangles denote entity types. In Figure 1, the Course entity type represents the set of courses in the database.

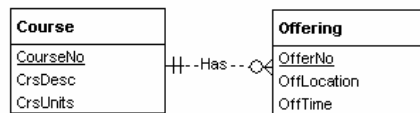


Figure 1 Entity relationship diagram illustrating basic symbols

---

**Attribute** a property of an entity type or relationship. Each attribute has a data type that defines the kind of values and permissible operations on the attribute.

---

Attributes are properties of entity types or relationships. An entity type should have a primary key as well as other descriptive attributes. Attributes are shown inside an entity type rectangle. If there are many attributes, the attributes can be suppressed and listed on a separate page. Some ERD drawing tools show attributes in a zoomed view, separate from the rest of the diagram. Underlining indicates that the attribute(s) serves as the primary key of the entity type.

---

**Relationship** a named association among entity types. A relationship represents a two-way or bi-directional association among entities. Most relationships involve two entity types.

---

Relationships are named associations among entity types. In the Crow's Foot notation, relationship names appear on the line connecting the entity types involved in the relationship. In Figure 1, the **Has** relationship shows that the *Course* and *Offering* entity types are directly related. Relationships store associations in both directions. For example, the **Has** relationship shows what offerings exist for a given course and what course corresponds to a given offering. The **Has** relationship is binary because it involves two entity types. In Section "Understanding Relationships", presents examples of more complex relationships involving only one entity type (unary relationships) and more than two entity types (M-way relationships).

In a loose sense, ERDs have a natural language correspondence. Entity types can correspond to nouns and relationships to verbs or prepositional phrases connecting nouns. In this sense, one can read an entity relationship diagram as a collection of sentences. For example, the ERD in Figure 1 can be read as "course has offerings." Note that there is an implied direction in relationships. In the other direction, one could write, "offering is given for a course." If practical, it is a good idea to use active rather than passive verbs for relationships. Therefore, **Has** is preferred as the relationship name. You should use the natural language correspondence as a guide rather than strict rule. For large ERDs, you will not always find a good natural language correspondence for all parts of the diagrams.

---

**Cardinality** a constraint on the number of entities that participate in a relationship. In an ERD, the minimum and maximum number of entities are specified for both directions of a relationship.

---

### Relationship Cardinality

Cardinalities constrain the number of objects that participate in a relationship. To depict the meaning of cardinalities, an object or instance diagram is useful. Figure 2 shows a set of courses ({Course1, Course2, Course3}), a set of offerings ({Offering1, Offering2, Offering3, Offering4}), and connections between the two sets. In Figure 2, Course1 is related to Offering1, Offering2, and Offering3, Course2 is related to Offering4, and Course3 is not related to any Offering objects. Likewise, Offering1 is related to Course1, Offering2 is related to Course2, Offering3 is related to Course1, and Offering4 is related to Course2. From this instance diagram, we might conclude that each offering is related to exactly one course. In the other direction, each course is related to 0 or more offerings.

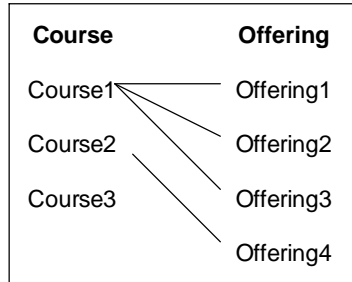


Figure 2 Instance diagram for the **Has** relationship

### Crow's Foot Representation of Cardinalities

The Crow's Foot notation uses three symbols to represent cardinalities. The Crow's Foot symbol (two angled lines and one straight line) denotes the many cardinality. In Figure 3, the Crow's Foot symbol near the *Offering* entity type means that a course can be related to many offerings. The circle means a cardinality of zero, while a single line means a cardinality of one.

To depict minimum and maximum cardinalities, the cardinality symbols are placed adjacent to each entity type in a relationship. The minimum cardinality symbol appears toward the relationship name while the maximum cardinality symbol appears toward the entity type. In Figure 3, a course is related to a minimum of zero offerings (circle in the inside position) and a maximum of many offerings (Crow's Foot in the outside position). Similarly, an offering is related to exactly one (one and only one) course as shown by the single vertical lines in both inside and outside positions.

**Existence Dependency** an entity that cannot exist unless another related entity exists. A mandatory relationship produces an existence dependency.

### Classification of Cardinalities

Cardinalities are classified by common values for minimum and maximum cardinality. Table 1 shows two classifications for minimum cardinalities. A minimum cardinality of one or more indicates a mandatory relationship. For example, participation in the **Has** relationship is mandatory for each *Offering* entity due to the minimum cardinality of one. A mandatory relationship makes the entity type existence dependent on the relationship. The *Offering* entity type depends on the **Has** relationship because an *Offering* object cannot be stored without a related *Course* object. In contrast, a minimum cardinality of 0 indicates an optional relationship. For example, the **Has** relationship is optional to the *Course* entity type because a *Course* object can be stored without being related to an *Offering* object. Figure 4 shows that the *Teaches* relationship is optional for both entity types.

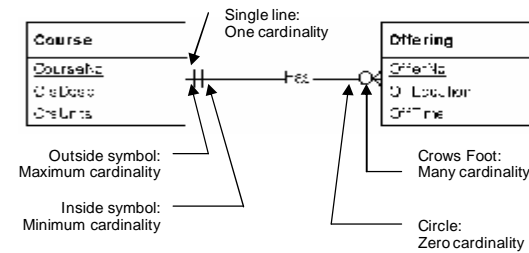


Figure 3 Entity relationship diagram with cardinalities noted

Table 1 Summary of cardinality classification

Classification	Cardinality Restrictions
Mandatory	Min. Card. $\leq 1$
Optional	Min. Card. = 0
Functional or single-valued	Max. Card. = 1
1-M	Max. Card. = 1 in one direction
	Max. Card. > 1 in other direction
M-N	Max. Card. is > 1 in both directions
1-1	Max. Card. = 1 in both directions

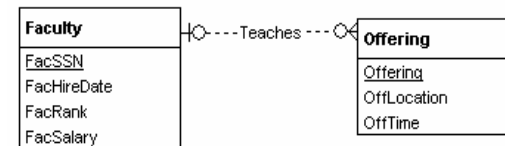


Figure 4 Optional relationship for both entity types

Table 1 also shows several classifications for maximum cardinalities. A maximum cardinality of one means the relationship is single-valued or functional. For example, the **Has** and *Teaches* relationships are functional for *Offering* because an *Offering* object can be related to a maximum of one *Course* and one *Faculty* object. The word "function" comes from mathematics where a function gives one value. A relationship that has a maximum cardinality of one in one direction and more than one (many) in the other direction is called a 1-M (read one-to-many) relationship. Both the **Has** and *Teaches* relationships are 1-M.

Similarly, a relationship that has a maximum cardinality of more than one in both directions is known as an M-N (many-to-many) relationship. In Figure 5, the

*TeamTeaches* relationship allows multiple professors to jointly teach the same offering, as shown in the instance diagram of Figure 6. M-N relationships are common in business databases. For example, M-N relationships usually represent the connection between parts and suppliers, authors and books, and skills and employees. Many suppliers can supply a part and a supplier can supply many parts.

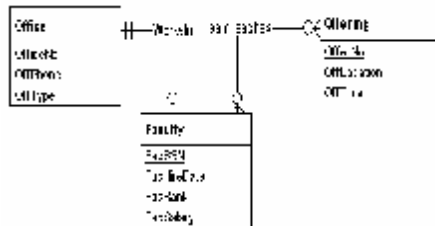


Figure 5 M-N and 1-1 relationship examples

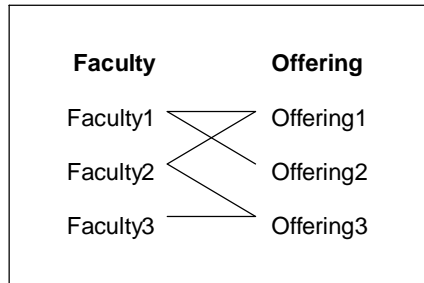


Figure 6 Instance diagram for the M-N *Teameaches* relationship

Less common are 1-1 relationships in which the maximum cardinality equals one in both directions. For example, the *WorksIn* relationship in Figure 5 allows a faculty to be assigned to one office and an office to be occupied by at most one faculty.

## Comparison to Relational Database Diagrams

To finish this section, let us compare the notation in Figure 3 with the relational database diagrams (from Microsoft Access) with which you are familiar. It is easy to become confused between the two notations. Some of the major differences are listed below. To help you visualize these differences, Figure 7 shows a relational database diagram for the *Course-Offering* example.

1. Relational database diagrams do not use names for relationships. Instead foreign keys represent relationships. The ERD notation does not use foreign keys. For example, *Offering.CourseNo* is a column in Figure 7 but not an attribute in Figure 3.
2. Relational database diagrams show only maximum cardinalities.

3. Some ERD notations (including the Crow's Foot notation) allow both entity types and relationships to have attributes. Relational database diagrams only allow tables to have columns.
4. Relational database diagrams allow a relationship between two tables. Some ERD notations (although not the Crow's Foot notation) allow M-way relationships involving more than two entity types. The next section shows how to represent M-way relationships in the Crow's Foot notation.
5. In some ERD notations (although not the Crow's Foot notation), the position of the cardinalities is reversed.

## Understanding Relationship

This section explores the entity relationship notation in more depth by examining important aspects of relationships. The first subsection describes identification dependency, a specialized kind of existence dependency. The second subsection describes three important relationship patterns: (1) relationships with attributes, (2) self-referencing relationships, and (3) associative entity types representing multi-way (M-way) relationships. The final subsection describes an important equivalence between M-N and 1-M relationships.

**Weak Entity** an entity type that borrows all or part of its primary key from another entity type. Identifying relationships indicate the entity types that supply components of the borrowed primary key.



Figure 7 Relational database diagram for the Course-Offering example

## Identification Dependency (Weak Entities and Identifying Relationships)

In an ERD, some entity types may not have their own primary key. Entity types without their own primary key must borrow part (or all) of their primary key from other entity types. Entity types that borrow part or their entire primary key are known as weak entities. The relationship(s) that provides components of the primary key is known as an identifying relationship. Thus, an identification dependency involves a weak entity and one or more identifying relationships.

**Identification dependency** occurs because some entities are closely associated with other entities. For example, a room does not have a separate identity from its building a room is physically contained in a building. You can reference a room only by providing its associated building identifier. In the ERD for buildings and rooms (Figure 8), the *Room* entity type is identification dependent on the *Building* entity type in the *Contains* relationship. A solid relationship line indicates an identifying relationship. For weak entities, the underlined attribute (if present) is part of the primary key, but not the entire primary key. Thus, the primary key of *Room* is a combination of *BldgID* and *RoomNo*. As another example, Figure 9 depicts an identification dependency involving the weak entity *State* and the identifying relationship *Holds*.

Identification dependency is a specialized kind of existence dependency. Recall that an existent-dependent entity type has a mandatory relationship (minimum cardinality of one). Weak entities are existent dependent on the identifying relationships. In addition to the existence dependency, weak entities borrow part or their entire primary key.

The next section shows several additional examples of identification dependency in the discussion of associative entity types and M-way relationships. The use of identification dependency is necessary for associative entity types.

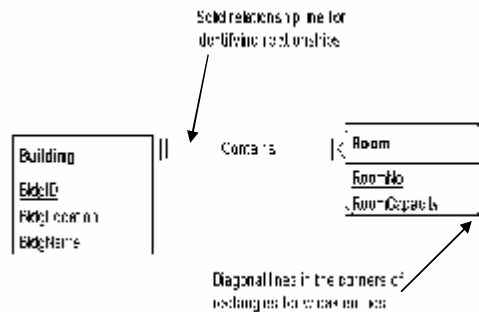


Figure 8 Identification dependency example

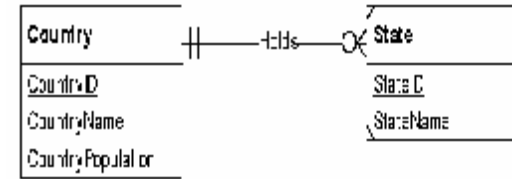


Figure 9 Another identification dependency example

## Relationship Patterns

This section discusses three patterns for relationships that you will encounter in database development efforts:

1. M-N relationships with attributes.
2. Self-referencing (unary) relationships.
3. Associative entity types representing M-way relationships.

Although these relationship patterns do not dominate ERDs, they are important when they occur. You need to study these patterns carefully to correctly apply them in database development efforts.

### M-N Relationships with Attributes

As briefly mentioned in introduction section, relationships can have attributes. This situation typically occurs with M-N relationships. In an M-N relationship attributes are associated with the combination of entity types, not just one of the entity types. If an attribute is associated with only one entity type, then it should be part of that entity type, not the relationship. Figures 10 and 11 depict M-N relationships with attributes. In Figure 10, the attribute *Enr-Grade* is associated with the combination of a student and offering, not either one alone. For example, the *EnrollsIn* relationship records the fact that the student with social security number 123-77-9993 has a grade of 3.5 in the offering with offer number 1256. In Figure 11(a), the attribute *Qty* represents the quantity of a part supplied by a given supplier. In Figure 11(b), the attribute *AuthOrder* represents the order in which the author's name appears in the title of a book. To reduce clutter on a large diagram, the attributes of relationships may not be shown.

1-M relationships also can have attributes, but 1-M relationships with attributes are much less common than M-N relationships with attributes. In Figure 12, the *Commission* attribute is associated with the *Lists* relationship, not with either the *Agent* or *Home* entity type. A home will only have a commission if an agent lists it.

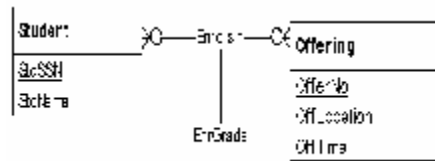


Figure 10 M-N relationship with an attribute

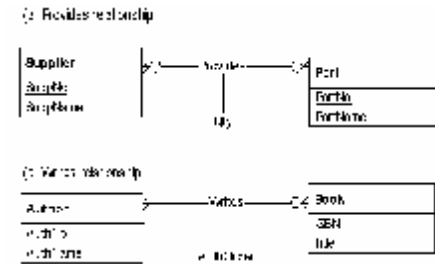


Figure 11 Additional M-N relationships with attributes

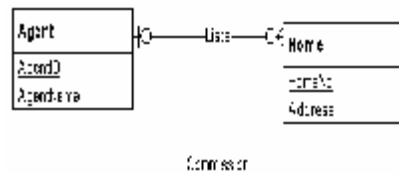


Figure 12 1-M relationship with an attribute

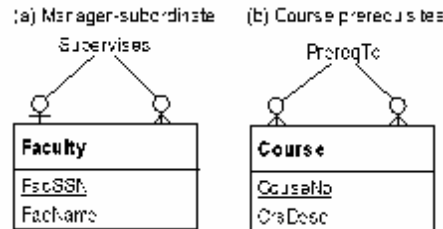


Figure 13 Examples of self-referencing (unary) relationships

## Self-Referencing (Unary) Relationships

A self-referencing (unary) relationship involves connections among members of the same set. Self-referencing relationships are sometimes called reflexive relationships because they are like a reflection in a mirror. Figure 13 displays two self-referencing relationships involving the *Faculty* and *Course* entity types. Both relationships

involve two entity types that are the same (*Faculty* for *Supervises* and *Course* for *PrereqTo*). These relationships depict important concepts in a university database. The *Supervises* relationship depicts an organizational chart, while the *PrereqTo* relationship depicts dependencies among courses that can affect a student's course planning.

For self-referencing relationships, it is important to distinguish between 1-M and M-N relationships. An instance diagram can help you understand the difference. Figure 14(a) shows an instance diagram for the *Supervises* relationship. Notice that each faculty can have at most one superior. For example, Faculty2 and Faculty3 have faculty1 as a superior. Therefore, *Supervises* is a 1-M relationship because each faculty can have at most one supervisor. In contrast, there is no such restriction in the instance diagram for the *PrereqTo* relationship Figure 14(b). For example, both *IS480* and *S460* are prerequisites to *IS461*. Therefore, *PrereqTo* is an M-N relationship because a course can be a prerequisite to many courses and a course can have many prerequisites.

Self-referencing relationships occur in a variety of business situations. Any data that can be visualized like Figure 14 can be represented as a self-referencing relationship. Typical examples include hierarchical charts of accounts, genealogical charts, part designs, and transportation routes. In these examples, self-referencing relationships are an important part of the database.

There is one other noteworthy aspect of self-referencing relationships. Sometimes a self-referencing relationship is not needed. For example, if you only want to know whether an employee is a supervisor, a self-referencing relationship is not needed. Rather, an attribute can be used to indicate whether an employee is a supervisor.

## Associative Entity Types Representing Multi-way (M-Way) Relationships

Some ERD notations support relationships involving more than two entity types, known as M-way (multi-way) relationships where the M means more than two. For example, the Chen ERD notation (with diamonds for relationships) allows relationships to connect more than two entity types, as depicted in Figure 15. The *Uses* relationship lists suppliers and parts used on projects. For example, a relationship instance involving *Supplier1*, *Part1*, and *Project1* means that *Supplier1* supplies *Part1* on *Project1*. An M-way relationship involving three entity types is called a ternary relationship.

Although you cannot directly represent M-way relationships in the Crow's Foot notation, you should understand how to indirectly represent them. You use an associative entity type and a collection of binary relationships to represent an M-way relationship. In Figure 16, three 1-M relationships link the associative entity type, *Uses*, to the *Part*, *Supplier*, and *Project* entity type. The *Uses* entity type is associative because its role is to connect other entity types. Because associative entity types provide a connecting role, they are sometimes given names using active verbs. In addition, associative entity types are always weak, as they must borrow the

entire primary key. For example, the *Uses* entity type obtains its primary key through the three identifying relationships.

As another example, Figure 17 shows the associative entity type *Provides* that connects the *Employees*, *Skill*, and *Project* entity types. An example instance of the *Provides* entity type contains *Employee1* providing *Skill1* on *Project1*.

**Self-Referencing Relationship** a relationship involving the same entity type. Self-referencing relationships represent associations among members of the same set.

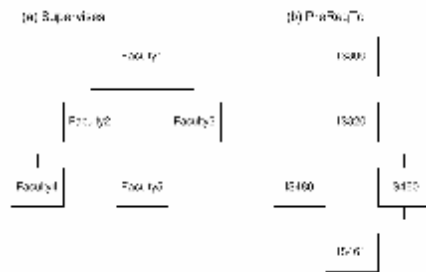


Figure 14 Instance diagrams for self-referencing relationships

**Associative Entity Type** a weak entity that replaces an M-way relationship. An associative entity type depends on two or more entity types tier its primary key.

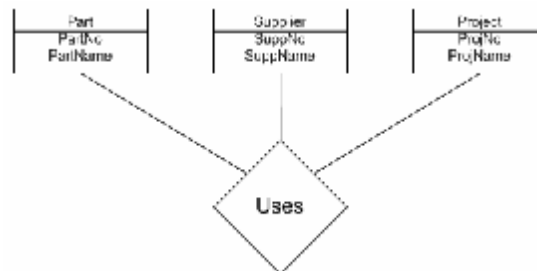


Figure 15 M-way (ternary) relationship using Chen notation

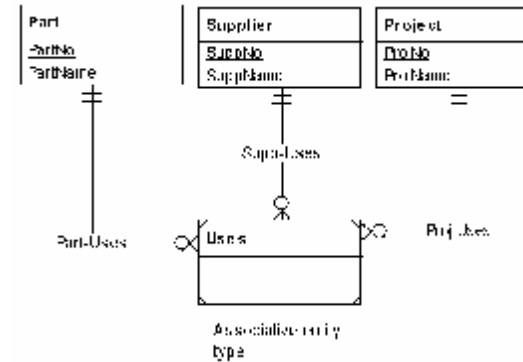


Figure 16 Associate entity type to represent a ternary relationship

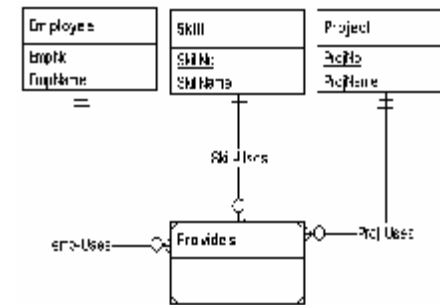


Figure 17 Associative entity type connecting Employees, Skill, and Project

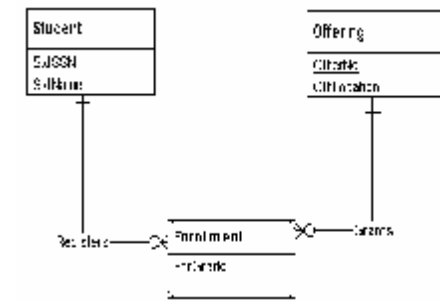


Figure 18 EnrollsIn M-N relationship (Figure 10) transformed into 1-M relationship.

The issue of when to use an associative entity type representing an M-way relationship can be difficult to understand. If a database only needs to record pairs of facts, an associative entity type is not needed. For example, if a database only needs to record supplies a part and what projects use a part, then an associative entity type should not be used. In this case, there should be binary relationships between *Supplier* and *Part* and between *Project* and *Part*. You should use an

associative entity type when the database should record combinations of three (or more) objects rather than just combinations of two objects. For example, if a database needs to record which supplier provides parts on specific projects, an associative entity type is needed.

### Equivalence between 1-M and M-N Relationships

To improve your understanding of M-N relationships, you should know an important equivalence for M-N relationships. An M-N relationship can be replaced by an associative entity type and two 1-M relationships. Figure 18 shows the *Enrollsn* (Figure 10) relationship converted to this 1-M style. In Figure 18, two identifying relationships and an associative entity type replace the *Enrollsn* relationship. The relationship name (*Enrollsn*) has been changed to a noun (*Enrollment*) to follow the convention of nouns for entity type names. The 1-M style is similar to the representation in a relational database diagram. If you feel more comfortable with the 1-M style, then use it. In terms of the ERD the M-N and 1-M styles have the same meaning.

The transformation of a binary M-N relationship into 1-M relationships is similar to representing an M-way relationship using 1-M relationships. Whenever an M-N relationship is represented as an associative relationship and two 1-M relationships, the new entity type is identification dependent on both 1-M relationships, as shown in Figure 18. Similarly, when representing M-way relationships, the associative entity type is identification dependent on all 1-M relationships, as shown in Figures 16 and 17.

There is one situation when the 1-M style is preferred to the M-N style. When an M-N relationship must be related to other entity types in another relationship, use the 1-M style. For example, assume that in addition to enrollment in a course offering, attendance in each class session should be recorded. In this situation, the 1-M style is preferred because it is necessary to link an enrollment with attendance records. Figure 19 shows the *Attendance* entity type added to the ERD of Figure 18. Note that an M-N relationship between the *Student* and *Offering* entity types would not have allowed another relationship with *Attendance*.

Figure 19 provides other examples of identification dependencies. The solid line by *Attendance* means that *Attendance* is identification dependent on *Enrollment* in the Record- relationship. The primary key of *Attendance* consists of *AttDate* along with the primary key of *Enrollment*. Similarly, *Attendance* is identification dependent on both *Student* and *Offering*. The primary key of *Enrollment* is a combination of *StdSSN* and *OfferNo*.

### Classification in the Entity Relationship Model

People classify objects to better understand their environment. For example, animals are classified into mammal, reptiles, and other categories to understand the similarities and differences among different species. In business classification is also pervasive. Classification can be applied to investments, employees, customers, loans, parts, and so on. For example, when applying for a home mortgage, an

important distinction is between fixed-rate and adjustable-rate mortgages. Within each kind of mortgage, there are many variations distinguished by features such as repayment period, prepayment penalties, and amount of loan.

This section describes ERD notation to support classification. You will learn to use generalization hierarchies, specify cardinality constraints for generalization hierarchies, and use multiple-level generalization hierarchies for complex classifications.

**Relationship Equivalence** is an M-N relationship can be replaced by an associative entity type and two identifying 1-M relationship.

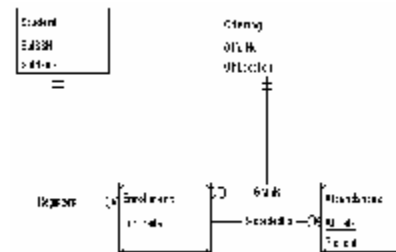


Figure 19 Attendance entity type added to the ERD of Fig. 18

### Generalization Hierarchies

Generalization hierarchies allow entity types to be related by the level of specialization. Figure 20 depicts a generalization hierarchy to classify employees as salaried versus hourly. Both salaried and hourly employees are specialized kinds of employees. The *Employee* entity type is known as the supertype (or parent). The entity types, *SalaryEmp* and *HourlyEmp* are known as the (or children): Because each subtype object is a supertype object, the relationship between a subtype and a supertype is known as ISA. For example, a salaried employee is an employee. Because the relationship name ISA is always the same, it is not shown on the diagram.

Inheritance supports sharing between a supertype and its subtypes. Because every subtype object is also a supertype object, the attributes of the supertype also apply to all subtypes. For example, every entity of *SalaryEmp* has an employee number, name, and hiring date because it is also an entity of *Employee*. Inheritance means that the attributes of a supertype are automatically part of its subtypes. That is, each subtype inherits the attributes of its supertype. For example, the attributes of the *SalaryEmp* entity type are its direct attribute (*EmpSalary*) and its inherited attributes from *Employee* (*EmpNo*, *EmpName*, *EmpHireDate*, etc.). Inherited attributes are not shown in an ERD. Whenever you have a subtype, assume that it inherits the attributes from its supertype.

Generalization Hierarchy a collection Of entity types arranged in a hierarchical structure to show similarity in attributes Each subtype or child entity represents a subset of its supertype of parent entity.

Inheritance a data modeling feature that supports sharing of attributes between a supertype and a subtype. Subtypes inherit attributes from their supertype.

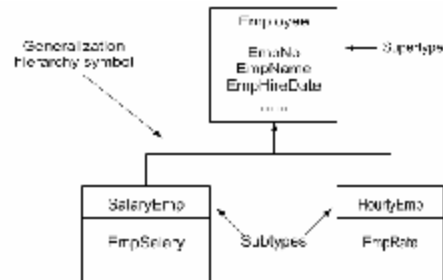


Figure 20 Generalization hierarchy for employees

### Disjointness and Completeness Constraints

Generalization hierarchies do not show cardinalities because they are always the same. Rather, disjointness and completeness constraints can be shown. Disjointness means that subtypes in a generalization hierarchy do not have any entities in common. In Figure 21, the generalization hierarchy is disjoint because a security cannot be both a stock and a bond. In contrast, the generalization hierarchy in Figure 22 is not disjoint because teaching assistants can be considered both students and faculty. Thus, the set of students overlaps with the set of faculty. Completeness means that every entity of a supertype must be an entity in one of the subtypes in the generalization hierarchy. The completeness constraint in Figure 21 means that every security must be either a stock or a bond.

Some generalization hierarchies lack both disjointness and completeness constraints. In Figure 20, the lack of a disjointness constraint means that some employees can be both salaried and hourly. The lack of a completeness constraint indicates that some employees are not paid by salary or the hour (perhaps by commission).

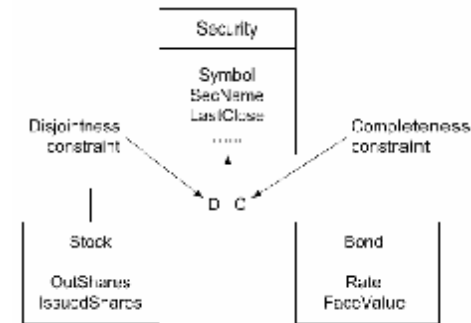


Figure 21 Generalization hierarchy for securities

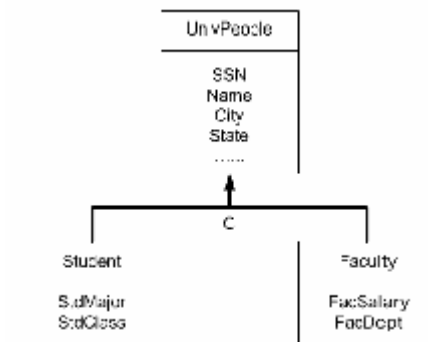


Figure 22 Generalization hierarchy for university people

### Multiple Levels of Generalization

Generalization hierarchies can be extended to more than one level. This practice can be useful in disciplines such as investments where knowledge is highly structured. In Figure 23, there are two levels of subtypes beneath securities. Inheritance extend, to all subtypes, direct and indirect. Thus, both the *Common* and *Preferred* entity types inherit the attributes of *Stock* (the immediate parent) and *Security* (the indirect parent). Note that disjointness and completeness constraints can be made for each group of subtypes.

### Review of Notation and comparison to other notations

You have seen a lot of ERD notation in the previous sections of this chapter. It is easy to become overwhelmed without a review to depict the most important points. To help you recall the notation introduced in the earlier sections, Table 2 presents a summary.



## Comprehensive ERD Example

Figure 24 demonstrates most of the ERD notation for the university database of last chapter. Some of the attributes are omitted for brevity. Note that the *Enrollment* entity type (associative) and the identifying relationships (*Registers* and *Grants*) could appear as an M-N relationship as previously shown in Figure 10.

## Diagram Variations

The ERD notation presented in this chapter is similar but not identical to what you may encounter later. There is no standard notation for ERDs. There are perhaps four to six reasonably popular ERD notations, each having its own small variations that appear in practice. The notation in this chapter comes from the Crow's Foot stencil in Visio Professional 5 with the addition of tile generalization notation. In Next chapter on Object Technology, a case presents the class diagram notation of the Unified Modeling Language, an emerging notation for data modeling. The notations that you encounter in practice will depend on factors such as the data-modeling tool (if any) used in your organization and the industry. One thing is certain: you should be prepared to adapt to the notation in use.

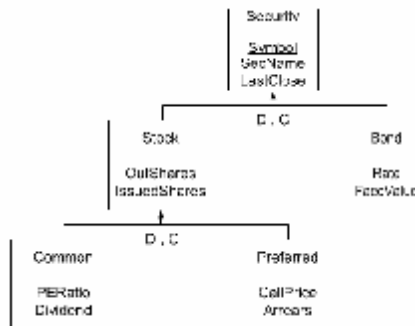


Figure 23 Multiple levels of generalization hierarchies

## Symbol Variations

Because there is no widely accepted ERD standard, different symbols can be used to represent the same concept. Relationship cardinalities are a source of wide variation. You should pay attention to the placement of the cardinality symbols. The notation in this chapter places the symbols close to the "far" entity type, while other notations place the cardinality symbols close to the "near" entity type. The notation in this chapter uses a visual representation of cardinalities with the minimum and maximum cardinalities given by three symbols. Other notations use a text representation with letters and integers instead of symbols. For example, Figure 7.25

shows a Chen ERR of Figure 7.3 with the position of cardinalities reversed, cardinalities depicted with text, and relationships denoted by the diamonds.

Other symbol variations are visual representations for certain kinds of entity types. In some notations, weak entities and M-N relationships have special representations. Weak entities are sometimes enclosed in double rectangles. Identifying relationships are sometimes enclosed in double diamonds. M-N relationships with attributes are sometimes shown as a rectangle with a diamond inside denoting the dual qualities (both relationship and entity type).

## Rule Variations

In addition to symbol variations, there are also rule variations, as shown in the following list. In each restriction, there is a remedy. For example, if only binary relationships are supported. M-way relationships must be represented as an associative entity type with 1-M relationships.

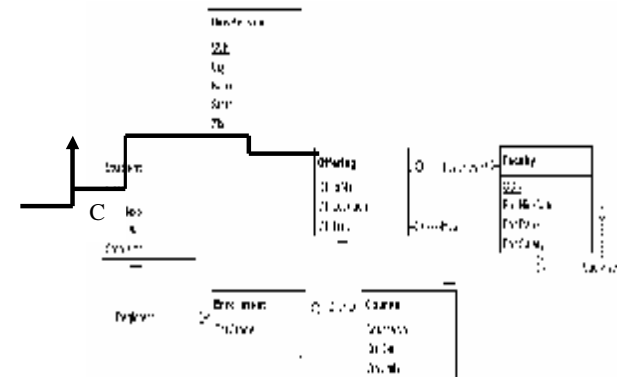


Figure 24 ERD for the university database

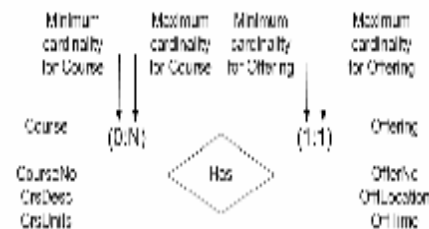


Figure 25 Chen notation for the course-offering ERD

1. Some notations do not support M-way relationships.
2. Some notations do not support M-N relationships.
3. Some notations do not support relationships with attributes.
4. Some notations do not support self-referencing (unary) relationships.
5. Some notations permit relationships to be connected to other relationships.
6. Some notations show foreign keys as attributes.
7. Some notations allow attributes to have more than one value (multi-valued attributes).

Relational database diagrams in Microsoft Access have most of these restrictions. Relational database diagrams do not support M-way relationships and M-N relationships. Despite these restrictions, relational database diagrams are no less expressive than notations without these restrictions. In certain situations, you may need additional symbols, but the same concept can be represented. For example, relational database diagrams may require more symbols to represent an M-N relationship, but an M-N relationship can still be represented.

### Converting an ERD to relational tables

Conversion from the ERD notation to relational tables is important because of industry practice. Computer-aided software engineering (CASE) tools support some kind of entity relationship notation. It is common practice to use a CASE tool as an aid in developing an ERD. Because most commercial DBMSs use the Relational Model, you must convert your ERD into relational tables to implement your database design.

This section describes the conversion process in two parts. First, the basic rules to convert entity types, relationships and attributes are described. Second, specialized rules to convert optional 1-M relationships, generalization hierarchies, and 1-1 relationships are shown.

### Basic Conversion Rules

The rules that follow convert everything on an ERD except generalization hierarchies. You should apply these rules until everything in your ERD is converted. The first two rules should be used before the other rules. As you apply these rules, you can use a check mark to indicate what parts of your ERD have been converted.

1. **Entity Type Rule:** Each entity type (except subtypes) becomes a table. The primary key of the entity type (if not weak) becomes the primary key of the table. The attributes of the entity type become columns in the table. This rule should be used first before the relationship rules.
2. **1-M Relationship Rule:** Each 1-M relationship becomes a foreign key in the table corresponding to the many entity type (the entity type near the Crow's Foot symbol). If the minimum cardinality is one, the foreign key cannot accept null values.

3. **M-N Relationship Rule:** Each M-N relationship becomes a separate table. The primary key of the table is a combined key consisting of the primary keys of the entity types participating in the M-N relationship.
4. **Identification Dependency Rule:** Each identifying relationship (denoted by a solid relationship line) adds a column to a primary key. The primary key of the table corresponding to the weak entity type consists of:
  - (i) The underlined local key (if any) in the weak entity type and
  - (ii) The primary key(s) of the entity type(s) connected by the identifying relationship(s).

To understand these rules, you can apply them to some of the ERDs given earlier in the chapter. Using Rules 1 and 2, you can convert Figure 26 into the CREATE TABLE statements shown in Figure 27. Rule 1 is applied to convert the *Course* and *Offering* entity types to tables. Then, Rule 2 is applied to convert the *Has* relationship to a foreign key (*Offering.CourseNo*). The *Offering* table contains the foreign key because the *Offering* entity type has the maximum cardinality of one.

Next, you can apply the M-N relationship rule (Rule 3) to convert the ERD in Figure 28. Following this rule leads to the *Enrolls In* table in Figure 29. The primary key of *Enrolls In* is a combination of the primary keys of the *Student* and the *Offering* entity types.

To gain practice with the identification dependency rule (Rule 4), you can use it to convert the ERD in Figure 30. The result of converting Figure 30 is identical to Figure 27 except that the *Enrolls In* table is renamed *Enrollment*. The ERD in Figure 30 requires two applications of the identification dependency rule. Each application of the identification dependency rule adds a component to the primary key of the *Enrollment* table.

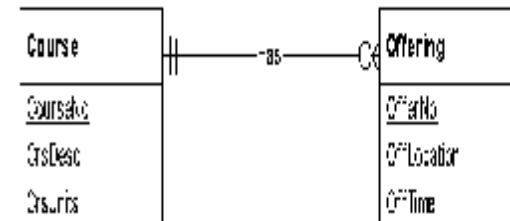


Figure 26 ERD with 1-M relationship

```
CREATE TABLE Course
    (CourseNo CHAR(6) NOT NULL,
    CrsDesc VARCHAR,
    CrsUnits SMALLINT
    CONSTRAINT PKCourse PRIMARY KEY (CourseNo) )

CREATE TABLE Offering
```

```
( OfferNo    LONG NOT NULL,
Off Location CHAR(20),
           CourseNo  CHAR(6)   NOT NULL,
OffTime     TIME,
.....

CONSTRAINT PKOffering PRIMARY KEY (OfferNo),
CONSTRAINT FKCourseNo FOREIGN KEY (CourseNo) REFERENCES Course )
```

Figure 27 Conversion of Figure 26

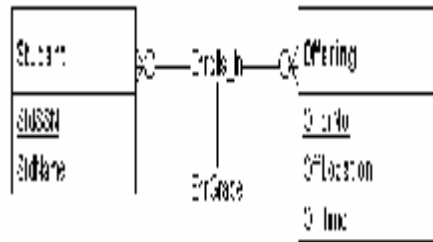


Figure 28 M-N relationship with an attribute

```
CREATE TABLE Student
(StdSSN    CHAR(11)  NOT NULL
StdName    VARCHAR,
CONSTRAINT PKStudent PRIMARY KEY (StdSSN))

CREATE TABLE Offering
(OfferNo    LONG NOT NULL
OffLocation VARCHAR,
OffTime     TIME,
.....

CONSTRAINT PKOffering PRIMARY KEY (OfferNo) )
CREATE TABLE Enrolls_In
(OfferNo    LONG NOT NULL,
StdSSN     CHAR(11)  NOT NULL,
EnrGrade    DECIMAL(2,1),
CONSTRAINT PKEnrolls_In PRIMARY KEY (OfferNo, StdSSN),
CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering,
CONSTRAINT FKStdSSN FOREIGN KEY (StdSSN) REFERENCES Student )
```

Figure 29 Conversion of Figure 28

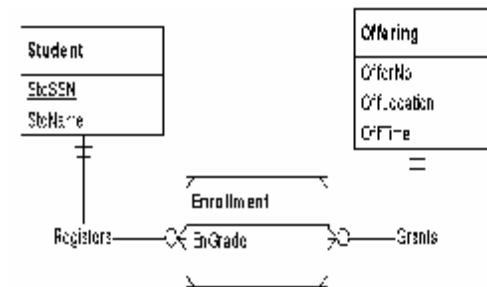


Figure 30 Enrolls\_In M-N relationship transformed into 1-M relationship

You also can apply the rules to convert self-referencing relationships. For example, you can apply the 1-M and M-N relationship rules to convert the self-referencing relationships in Figure 31. Using the 1-M relationship rule, the *Supervises* relationship converts to a foreign key in the *Faculty* table, as shown in Figure 32. Using the M-N relationship Rule, the *Prereq\_To* relationship converts to the *Prereq\_To* table with a combined primary key of the course number of the prerequisite course and the course number of the dependent course.

You also can apply conversion rules to more complex identification dependencies, as depicted in Figure 33. The first part of the conversion is identical to the conversion of Figure 30. Application of the 1-M rule makes the combination of *StdSSN* and *OfferNo* foreign keys in the *Attendance* table (Figure 34). Note that the foreign keys in *Attendance* refer to *Enrollment*, not to *Student* and *Offering*. Finally, one application of the identification dependency rule makes the combination of *StdSSN*, *OfferNo* and *AttDate* the primary key of the *Attendance* table.

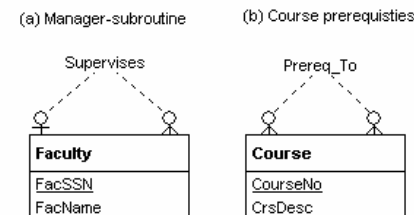


Figure 31 Example of 1-M and M-N self-referencing relationships

```
CREATE TABLE Faculty
(FacSSN    CHAR(11)  NOT NULL,
FacName    VARCHAR,
FacSupervisor CHAR(11),
CONSTRAINT PKFaculty PRIMARY KEY (FacSSN),
FOREIGN KEY (FacSupervisor) REFERENCES Faculty)

CREATE TABLE Course
(CourseNo  CHAR(6)   NOT NULL,
```

```

CrsDesc    VARCHAR,
CrsUnits   SMALLINT
CONSTRAINT PI(Course PRIMARY KEY (CourseNo))

CREATE TABLE Prereq_To
(Prereq_CNo CHAR(6) NOT NULL,
 Depend CNo CHAR(6) NOT NULL,
CONSTRAINT PKPrereq_To PRIMARY KEY (Prereq_CNo, Depend CNo),
CONSTRAINT FKPrereq_CNo FOREIGN KEY (Prereq_CNo) REFERENCES
Course,
CONSTRAINT FKDepend_CNo FOREIGN KEY (Depend_CNo) REFERENCES
Course)

```

Figure 32 Conversion of Figure 31.

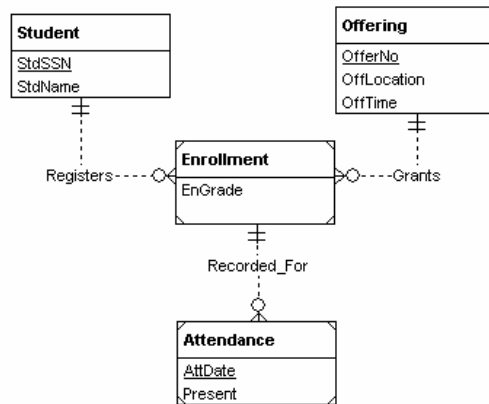


Figure 33 ERD with two weak entity types

```

CREATE TABLE Attendance
(OfferNo    LONG      NOT NULL,
 StdSSN     CHAR(11)  NOT NULL,
 AttDate    Date      NOT NULL,
 Present    BOOL,
CONSTRAINT PKAttendance PRIMARY KEY (OfferNo, StdSSN, AttDate),
CONSTRAINT FKOfferNoStdSSN FOREIGN KEY (OfferNo, StdSSN)
REFERENCES Enrollment)

```

Figure 34 Conversion of Attendance entity type in Figure 33

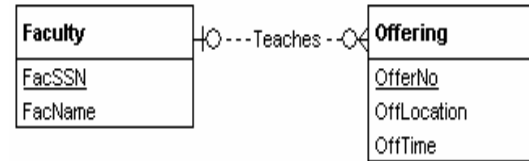


Figure 35 Optional 1-M relationship

### Converting Optional 1-M Relationships

Using the 1-M relationship rule results in null values when converting optional relationships. Recall that a relationship with a minimum cardinality of 0 is optional. For example, the *Teaches* relationship (Figure 35) is optional to *Offering* because an *Offering* object can be stored without being related to a *Faculty* object. Converting Figure 35 results in two tables (*Faculty* and *Offering*) as well as a foreign key (*FacSSN*) in the *Offering* table. The foreign key should allow null values because the minimum cardinality of the *Offering* entity type in the relationship is optional (0). However, null values can lead to complications in evaluating the results of queries.

To avoid null values when converting optional 1-M relationships, you can apply Rule 5 below. Rule 5 converts optional 1-M relationships into a table instead of a foreign key. Figure 36 shows an application of Rule 5 to the ERD in Figure 35. The *Teaches* table contains the foreign key *FacSSN*. Note that the *FacSSN* column does not permit null values. The *Offering* table no longer has a foreign key referring to the *Faculty* table.

- Optional 1-M Relationship Rule:** Each 1-M relationship with a minimum cardinality of 0 and a maximum cardinality of 1 becomes a new table. The primary key of the new table is the primary key of the entity type on the many side of the relationship. The primary key of the other entity type becomes a foreign key in the new table. The foreign key in the new table does not permit null values.

Rule 5 is controversial. Using Rule 5 in place of Rule 2 (1-M Relationship Rule) avoids null values in foreign keys. However, using Rule 5 results in more tables. Query formulation can be more difficult with additional tables. In addition, query execution can be slower due to extra joins. The choice of using Rule 5 in place of Rule 2 depends on the importance of avoiding null values versus avoiding extra tables. In many databases, avoiding extra tables may be more important than avoiding null values.

### Converting Generalization Hierarchies

The approach to convert generalization hierarchies mimics the entity relationship notation as much as possible. Rule 6 converts each entity type of a generalization hierarchy into a table. The only column appearing in a table that is different from the ERD is the inherited primary key. In Figure 37 *EmpNo* is a column in the *SalaryEmp* and *HourlyEmp* tables because it is the primary key of the parent entity type

(Employee). In addition, the *SalaryEmp* and *HourlyEmp* tables have a foreign key constraint referring to the *Employee* table. The *CASCADE delete* option is set in both foreign key constraints (see Figure 38).

6. **Generalization Hierarchy Rule:** Each entity type of a generalization hierarchy becomes a table. The *Columns* of a table are the attributes of the corresponding entity type plus the primary key of the parent entity type. For each table representing a subtype, define a foreign key constraint that references the table corresponding to the parent entity type. Use the *CASCADE* option for deletions of referenced rows.

```
CREATE TABLE Faculty
(FacSSN CHAR(11) NOT NULL,
FacName VARCHAR,
...,
CONSTRAINT PKFaculty PRIMARY KEY (FacSSN))

CREATE TABLE Offering
(OfferNo LONG NOT NULL,
Off Location VARCHAR,
OffTime TIME,
...,
CONSTRAINT PKOffering PRIMARY KEY (OfferNo))

CREATE TABLE Teaches
(OfferNo LONG NOT NULL,
FacSSN CHAR(11) NOT NULL,
CONSTRAINT PKTeaches PRIMARY KEY (OfferNo),
CONSTRAINT FKFacSSN FOREIGN KEY (FacSSN) REFERENCES Faculty)
```

Figure 36 Conversion of Figure 35.

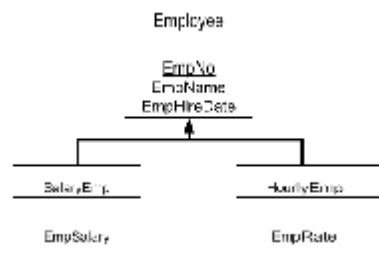


Figure 37 Generalization hierarchy for employees

Rule 6 also applies to generalization hierarchies of more than one level. To convert the generalization hierarchy of Figure 39 five tables are produced (see Figure 40). In each table, the primary key of the parent (*Security*) is included. In addition, foreign key constraints are added in each table corresponding to a subtype.

```
CREATE TABLE Employee
(EmpNo LONG NOT NULL,
EmpName VARCHAR,
EmpHireDate DATE,
CONSTRAINT PKEmployee PRIMARY KEY (EmpNo) )

CREATE TABLE SalaryEmp
(EmpNo LONG NOT NULL,
EmpSalary DECIMAL(10,2),
CONSTRAINT PKSalaryEmp PRIMARY KEY (EmpNo) ,
CONSTRAINT FKSalaryEmp FOREIGN KEY (EmpNo) REFERENCES
Employee
ON DELETE CASCADE)

CREATE TABLE HourlyEmp
(EmpNo LONG NOT NULL,
EmpRate DECIMAL(10,2),
CONSTRAINT PKHourlyEmp PRIMARY KEY (EmpNo) ,
CONSTRAINT FKHourlyEmp FOREIGN KEY (EmpNo) REFERENCES
Employee
ON DELETE CASCADE)
```

Figure 38 Conversion of generalization hierarchy in Figure 37

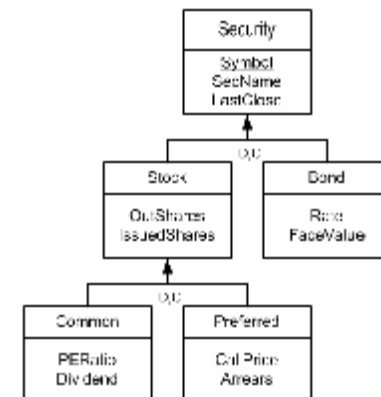


Figure 39 Multiple levels of generalization hierarchies

```
CREATE TABLE Security
(Symbol CHAR(6) NOT NULL,
SecName VARCHAR,
LastClose DECIMAL(10,2),
CONSTRAINT PKSecurity PRIMARY KEY (Symbol) )

CREATE TABLE Stock
(Symbol CHAR(6) NOT NULL,
```

```
OutShares      INTEGER,
IssuedShares   INTEGER,
CONSTRAINT PKStock PRIMARY KEY (Symbol),
CONSTRAINT FKStock FOREIGN KEY (Symbol) REFERENCES Security ON
DELETE CASCADE )
```

```
CREATE TABLE Bond
(Symbol      CHAR(6)   NOT NULL,
Rate  DECIMAL(12,4),
FaceValue  DECIMAL(10,2),
CONSTRAINT PKBond PRIMARY KEY (Symbol),
CONSTRAINT FKBond FOREIGN KEY (Symbol) REFERENCES Security ON
DELETE CASCADE )
```

```
CREATE TABLE Common
(Symbol      CHAR(6)   NOT NULL,
PE/Ratio    DECIMAL(12,4),
Dividend    DECIMAL(10,2),
CONSTRAINT PKCommon PRIMARY KEY (Symbol),
CONSTRAINT FKCommon FOREIGN KEY (Symbol) REFERENCES Stock ON
DELETE CASCADE )
```

```
CREATE TABLE Preferred
(Symbol      CHAR(6)   NOT NULL,
CallPrice   DECIMAL(12,2),
Arrears     DECIMAL(10,2),
CONSTRAINT PKPreferred PRIMARY KEY (Symbol),
CONSTRAINT FKPreferred FOREIGN KEY (Symbol) REFERENCES Stock ON
DELETE CASCADE )
```

Figure 40 Conversion of generalization hierarchy in Figure 39

Because the Relational Model does not directly support generalization hierarchies, there are several other ways to convert generalization hierarchies. The other approaches vary depending on the number of tables and the placement of inherited columns. Rule 6 may result in extra joins to gather all data about an entity, but there are no null values and only small amounts of duplicate data. For example, to collect all data about a common stock, you should join the *Common*, *Stock*, and *Security* tables. Other conversion approaches may require fewer joins but result in more redundant data and null values. The references at the end of this chapter discuss the pros and cons of several approaches to convert generalization hierarchies.

### Converting 1-1 Relationships

Outside of generalization hierarchies, 1-1 relationships are not common. They can occur when entities with separate identifiers are closely related. For example, Figure 41 shows the *Employee* and *Office* entity types connected by a 1-1 relationship. Separate entity types seem intuitive but a 1-1 relationship connects the entity types. Rule 7 converts 1-1 relationships into two foreign keys unless many null values will

result. In Figure 41, most employees will not manage offices. Thus, the conversion in Figure 42 eliminates the foreign key (*OfficeNo*) in the employee table.

7. **1-1 Relationship Rule:** The 1-1 relationship is converted into two foreign keys. If the relationship is optional with respect to one of the entity types, the corresponding foreign key may be dropped to eliminate null values.

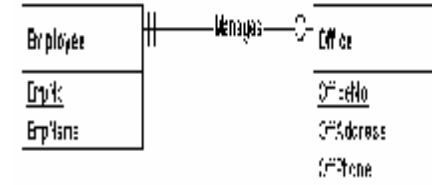


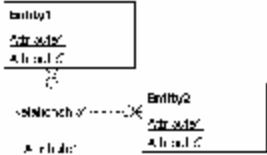


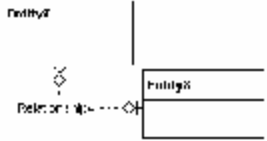
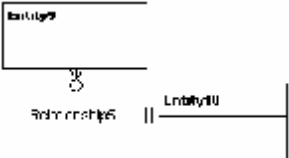
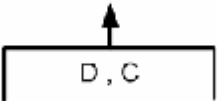
Figure 41 1-1 Relationship

```
CREATE TABLE Employee
(EmpNo      LONG NOT NULL,
EmpName     VARCHAR,
CONSTRAINT  PKEmployee PRIMARY KEY (EmpNo) )
```

```
CREATE TABLE Office
(OfficeNo   LONG NOT NULL,
OffAddress  VARCHAR,
OffPhone    CHAR(10),
EmpNo      LONG,
CONSTRAINT PKOffice PRIMARY KEY (OfficeNo) ,
CONSTRAINT FKEmpNo FOREIGN KEY (EmpNo) REFERENCES Employee
CONSTRAINT EmpNoUnique UNIQUE (EmpNo) )
```

Figure 42 Conversion of the 1-1 relationship in Figure 41

## APPENDIX

Symbol	Meaning
	<p>M-N relationship with attributes: attributes are shown if room permits; otherwise attributes are listed separately.</p> <p>Entity type with attributes (primary key underlined).</p>
	<p>Identification dependency: identifying relationship(s) (solid relationship lines) and weak entity (Diagonal lines in the corners of the rectangle). Associative entity types also are weak because they are (by definition) identification dependent.</p>
	<p>Existence dependent cardinality (min. cardinality of 1): inner symbol is a solid line.</p>
	<p>Optional cardinality (min. cardinality of 0): inner symbol is a circle.</p>
	<p>Single-valued cardinality (max. cardinality of 1): outer symbol is a solid line.</p>
	<p>Generalization hierarchy with disjointness and completeness constraints.</p>