who attended a college fair but has not applied is not a student, assuming the college fair is not a noncredit training program).

Business rules and policies are not universal; for example, different universities may have different policies for student advising and may include different types of people as students. Also, the rules and policies of an organization may change (usually slowly) over time; a university may decide that a student does not have to be assigned a faculty adviser until the student chooses a major.

Your job as a database analyst is to:

- Identify and understand those rules *that govern data*.
- Represent those rules so that they can be unambiguously understood by information systems developers and users.
- Implement those rules in database technology.

Data modeling is an important tool in this process. Because the purpose of data modeling is to document business rules about data, we introduce the discussion of data modeling and the E-R notation with an overview of business rules. Data models cannot represent all business rules (and do not need to, because not all business rules govern data); data models along with associated documentation and other types of information system models (e.g., models that document the processing of data) represent all business rules that must be enforced through information systems.

Overview of Business Rules

Business rule

A statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business.

A **business rule** is "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business … rules prevent, cause, or suggest things to happen" (GUIDE Business Rules Project, 1997). For example, the following two statements are common expressions of business rules that affect data processing and storage:

- "A student may register for a section of a course only if he or she has successfully completed the prerequisites for that course."
- "A preferred customer qualifies for a 10 percent discount, unless he has an overdue account balance."

Most organizations (and their employees) today are guided by thousands of combinations of such rules. In the aggregate, these rules influence behavior and determine how the organization responds to its environment (Gottesdiener, 1997; von Halle, 1997). Capturing and documenting business rules is an important, complex task. Thoroughly capturing and structuring business rules, then enforcing them through database technologies, helps ensure that information systems work right and that users of the information understand what they enter and see.

THE BUSINESS RULES PARADIGM The concept of business rules has been used in information systems for some time. There are many software products that help organizations manage their business rules (e.g., IBM WebSphere ILOG JRules). In the database world, it has been more common to use the related term *integrity constraint* when referring to such rules. The intent of this term is somewhat more limited in scope, usually referring to maintaining valid data values and relationships in the database.

A business rules approach is based on the following premises:

- Business rules are a core concept in an enterprise because they are an expression of business policy and guide individual and aggregate behavior. Well-structured business rules can be stated in natural language for end users and in a data model for systems developers.
- Business rules can be expressed in terms that are familiar to end users. Thus, users can define and then maintain their own rules.
- Business rules are highly maintainable. They are stored in a central repository, and each rule is expressed only once, then shared throughout the organization. Each rule is discovered and documented only once, to be applied in all systems development projects.

- Enforcement of business rules can be automated through the use of software that can interpret the rules and enforce them using the integrity mechanisms of the database management system (Moriarty, 2000).

Although much progress has been made, the industry has not realized all of these objectives to date (Owen, 2004). Possibly the premise with greatest potential benefit is "Business rules are highly maintainable." The ability to specify and maintain the requirements for information systems as a set of rules has considerable power when coupled with an ability to generate automatically information systems from a repository of rules. Automatic generation and maintenance of systems will not only simplify the systems development process but also will improve the quality of systems.

Scope of Business Rules

In this chapter and the next, we are concerned with business rules that impact only an organization's databases. Most organizations have a host of rules and/or policies that fall outside this definition. For example, the rule "Friday is business casual dress day" may be an important policy statement, but it has no immediate impact on databases. In contrast, the rule "A student may register for a section of a course only if he or she has successfully completed the prerequisites for that course" is within our scope because it constrains the transactions that may be processed against the database. In particular, it causes any transaction that attempts to register a student who does not have the necessary prerequisites to be rejected. Some business rules cannot be represented in common data modeling notation; those rules that cannot be represented in a variation of an E-R diagram are stated in natural language, and some can be represented in the relational data model, which we describe in Chapter 4.

GOOD BUSINESS RULES Whether stated in natural language, a structured data model, or other information systems documentation, a business rule will have certain characteristics if it is to be consistent with the premises outlined previously. These characteristics are summarized in Table 2-1. These characteristics will have a better chance of being satisfied if a business rule is defined, approved, and owned by business, not technical, people. Businesspeople become stewards of the business rules. You, as the database analyst, facilitate the surfacing of the rules and the transformation of ill-stated rules into ones that satisfy the desired characteristics.

TABLE 2-1 Characteristics of a Good Business Rule

Characteristic	Explanation
Declarative	A business rule is a statement of policy, not how policy is enforced or conducted; the rule does not describe a process or implementation but rather describes what a process validates.
Precise	With the related organization, the rule must have only one interpretation among all interested people, and its meaning must be clear.
Atomic	A business rule marks one statement, not several; no part of the rule can stand on its own as a rule (i.e., the rule is indivisible, yet sufficient).
Consistent	A business rule must be internally consistent (i.e., not containing conflicting statements) and must be consistent with (and not contradict) other rules.
Expressible	A business rule must be able to be stated in natural language, but it will be stated in a structured natural language so that there is no misinterpretation.
Distinct	Business rules are not redundant, but a business rule may refer to other rules (especially to definitions).
Business-oriented	A business rule is stated in terms businesspeople can understand, and because it is a statement of business policy, only businesspeople can modify or invalidate a rule; thus, a business rule is owned by the business.

Source: Based on Gottesdiener (1999) and Plotkin (1999).

GATHERING BUSINESS RULES Business rules appear (possibly implicitly) in descriptions of business functions, events, policies, units, stakeholders, and other objects. You can find these descriptions in interview notes from individual and group information systems requirements collection sessions, organizational documents (e.g., personnel manuals, policies, contracts, marketing brochures, and technical instructions), and other sources. Rules are identified by asking questions about the who, what, when, where, why, and how of the organization. Usually, a data analyst has to be persistent in clarifying initial statements of rules because initial statements may be vague or imprecise (what some people have called "business ramblings"). Thus, precise rules are formulated from an iterative inquiry process. You should be prepared to ask such questions as "Is this always true?" "Are there special circumstances when an alternative occurs?" "Are there distinct kinds of that person?" "Is there only one of those or are there many?" and "Is there a need to keep a history of those, or is the current data all that is useful?" Such questions can be useful for surfacing rules for each type of data modeling construct we introduce in this chapter and the next.

Data Names and Definitions

Fundamental to understanding and modeling data are naming and defining data objects. Data objects must be named and defined before they can be used unambiguously in a model of organizational data. In the E-R notation you will learn in this chapter, you have to give entities, relationships, and attributes clear and distinct names and definitions.

DATA NAMES We will provide specific guidelines for naming entities, relationships, and attributes as we develop the entity-relationship data model, but there are some general guidelines about naming any data object. Data names should (Salin, 1990):

- ***Relate to business, not technical (hardware or software), characteristics;*** so, Customer is a good name, but File10, Bit7, and Payroll Report Sort Key are not good names.
- ***Be meaningful,*** almost to the point of being self-documenting (i.e., the definition will refine and explain the name without having to state the essence of the object's meaning); you should avoid using generic words such as *has*, *is*, *person*, or *it*.
- ***Be unique*** from the name used for every other distinct data object; words should be included in a data name if they distinguish the data object from other similar data objects (e.g., Home Address versus Campus Address).
- ***Be readable,*** so that the name is structured as the concept would most naturally be said (e.g., Grade Point Average is a good name, whereas Average Grade Relative To A, although possibly accurate, is an awkward name).
- ***Be composed of words taken from an approved list;*** each organization often chooses a vocabulary from which significant words in data names must be chosen (e.g., maximum is preferred, never upper limit, ceiling, or highest); alternative, or alias names, also can be used as can approved abbreviations (e.g., CUST for CUSTOMER), and you may be encouraged to use the abbreviations so that data names are short enough to meet maximum length limits of database technology.
- ***Be repeatable,*** meaning that different people or the same person at different times should develop exactly or almost the same name; this often means that there is a standard hierarchy or pattern for names (e.g., the birth date of a student would be Student Birth Date and the birth date of an employee would be Employee Birth Date).
- ***Follow a standard syntax,*** meaning that the parts of the name should follow a standard arrangement adopted by the organization.

Salin (1990) suggests that you develop data names by:

1. Preparing a definition of the data. (We talk about definitions next.)
2. Removing insignificant or illegal words (words not on the approved list for names); note that the presence of AND and OR in the definition may imply that

two or more data objects are combined, and you may want to separate the objects and assign different names.
3. Arranging the words in a meaningful, repeatable way.
4. Assigning a standard abbreviation for each word.
5. Determining whether the name already exists and, if so, adding other qualifiers that make the name unique.

You will see examples of good data names as we develop a data modeling notation in this chapter.

DATA DEFINITIONS A definition (sometimes called a *structural assertion*) is considered a type of business rule (GUIDE Business Rules Project, 1997). A definition is an explanation of a term or a fact. A **term** is a word or phrase that has a specific meaning for the business. Examples of terms are *course*, *section*, *rental car*, *flight*, *reservation*, and *passenger*. Terms are often the keywords used to form data names. Terms must be defined carefully and concisely. However, there is no need to define common terms such as *day*, *month*, *person*, or *television*, because these terms are understood without ambiguity by most persons.

Term
A word or phrase that has a specific meaning for the business.

A **fact** is an association between two or more terms. A fact is documented as a simple declarative statement that relates terms. Examples of facts that are definitions are the following (the defined terms are underlined):

Fact
An association between two or more terms.

- "A course is a module of instruction in a particular subject area." This definition associates two terms: *module of instruction* and *subject area*. We assume that these are common terms that do not need to be further defined.
- "A customer may request a model of car from a rental branch on a particular date." This fact, which is a definition of *model rental request*, associates the four underlined terms (GUIDE Business Rules Project, 1997). Three of these terms are business-specific terms that would need to be defined individually (date is a common term).

A fact statement places no constraints on instances of the fact. For example, it is inappropriate in the second fact statement to add that a customer may not request two different car models on the same date. Such constraints are separate business rules.

GOOD DATA DEFINITIONS We will illustrate good definitions for entities, relationships, and attributes as we develop the E-R notation in this and the next chapters. There are, however, some general guidelines to follow (Aranow, 1989):

- Definitions (and all other types of business rules) are gathered from the same sources as all requirements for information systems. Thus, systems and data analysts should be looking for data objects and their definitions as these sources of information systems requirements are studied.
- Definitions will usually be accompanied by diagrams, such as E-R diagrams. The definition does not need to repeat what is shown on the diagram but rather supplement the diagram.
- Definitions will be stated in the singular and explain what the data element is, not what it is not. A definition will use commonly understood terms and abbreviations and stand alone in its meaning and not embed other definitions within it. It should be concise and concentrate on the essential meaning of the data, but it may also state such characteristics of a data object as:
  - Subtleties.
  - Special or exceptional conditions.
  - Examples.
  - Where, when, and how the data are created or calculated in the organization.
  - Whether the data are static or change over time.
  - Whether the data are singular or plural in their atomic form.
  - Who determines the value for the data.
  - Who owns the data (i.e., who controls the definition and usage).

- Whether the data are optional or whether empty (what we will call null) values are allowed.
- Whether the data can be broken down into more atomic parts or are often combined with other data into some more composite or aggregate form.

If not included in a data definition, these characteristics need to be documented elsewhere, where other metadata are stored.

- A data object should not be added to a data model, such as an E-R diagram, until after it has been carefully defined (and named) and there is agreement on this definition. But expect the definition of the data to change once you place the object on the diagram because the process of developing a data model tests your understanding of the meaning of data. (In other words, *modeling data is an iterative process*.)

There is an unattributed phrase in data modeling that highlights the importance of good data definitions: "The person who controls the meaning of data controls the data." It might seem that obtaining concurrence in an organization on the definitions to be used for the various terms and facts should be relatively easy. However, this is usually far from the case. In fact, it is likely to be one of the most difficult challenges you will face in data modeling or, for that matter, in any other endeavor. It is not unusual for an organization to have multiple definitions (perhaps a dozen or more) for common terms such as *customer* or *order*.

To illustrate the problems inherent in developing definitions, consider a data object of Student found in a typical university. A sample definition for Student is "a person who has been admitted to the school and who has registered for at least one course during the past year." This definition is certain to be challenged because it is probably too narrow. A person who is a student typically proceeds through several stages in relationship with the school, such as the following:

1. Prospect—some formal contact, indicating an interest in the school.
2. Applicant—applies for admission.
3. Admitted applicant—admitted to the school and perhaps to a degree program.
4. Matriculated student—registers for at least one course.
5. Continuing student—registers for courses on an ongoing basis (no substantial gaps).
6. Former student—fails to register for courses during some stipulated period (now may reapply).
7. Graduate—satisfactorily completes some degree program (now may apply for another program).

Imagine the difficulty of obtaining consensus on a single definition in this situation! It would seem you might consider three alternatives:

1. ***Use multiple definitions to cover the various situations.*** This is likely to be highly confusing if there is only one entity type, so this approach is not recommended (multiple definitions are not good definitions). It might be possible to create multiple entity types, one for each student situation. However, because there is likely considerable similarity across the entity types, the fine distinctions between the entity types may be confusing, and the data model will show many constructs.
2. ***Use a very general definition that will cover most situations.*** This approach may necessitate adding additional data about students to record a given student's actual status. For example, data for a student's status, with values of prospect, applicant, and so forth, might be sufficient. On the other hand, if the same student could hold multiple statuses (e.g., prospect for one degree and matriculated for another degree), this might not work.
3. ***Consider using multiple, related data objects for Student.*** For example, we could create a general entity type for Student and then other specific entity types for kinds of students with unique characteristics. We describe the conditions that suggest this approach in Chapter 3.

MODELING ENTITIES AND ATTRIBUTES

The basic constructs of the E-R model are entities, relationships, and attributes. As shown in Figure 2-2, the model allows numerous variations for each of these constructs. The richness of the E-R model allows designers to model real-world situations accurately and expressively, which helps account for the popularity of the model.

Entities

An **entity** is a person, a place, an object, an event, or a concept in the user environment about which the organization wishes to maintain data. Thus, an entity has a singular noun name. Some examples of each of these *kinds* of entities follow:

Entity
A person, a place, an object, an event, or a concept in the user environment about which the organization wishes to maintain data.

Person:	EMPLOYEE, STUDENT, PATIENT
Place:	STORE, WAREHOUSE, STATE
Object:	MACHINE, BUILDING, AUTOMOBILE
Event:	SALE, REGISTRATION, RENEWAL
Concept:	ACCOUNT, COURSE, WORK CENTER

ENTITY TYPE VERSUS ENTITY INSTANCE There is an important distinction between entity types and entity instances. An **entity type** is a collection of entities that share common properties or characteristics. Each entity type in an E-R model is given a name. Because the name represents a collection (or set) of items, it is always singular. We use capital letters for names of entity type(s). In an E-R diagram, the entity name is placed inside the box representing the entity type (see Figure 2-1).

Entity type
A collection of entities that share common properties or characteristics.

An **entity instance** is a single occurrence of an entity type. Figure 2-3 illustrates the distinction between an entity type and two of its instances. An entity type is described just once (using metadata) in a database, whereas many instances of that entity type may be represented by data stored in the database. For example, there is one EMPLOYEE entity type in most organizations, but there may be hundreds (or even thousands) of instances of this entity type stored in the database. We often use the single term *entity* rather than *entity instance* when the meaning is clear from the context of our discussion.

Entity instance
A single occurrence of an entity type.

ENTITY TYPE VERSUS SYSTEM INPUT, OUTPUT, OR USER A common mistake people make when they are learning to draw E-R diagrams, especially if they are already familiar with data process modeling (such as data flow diagramming), is to confuse data entities with other elements of an overall information systems model. A simple rule to avoid such confusion is that *a true data entity will have many possible instances, each with a distinguishing characteristic, as well as one or more other descriptive pieces of data*.

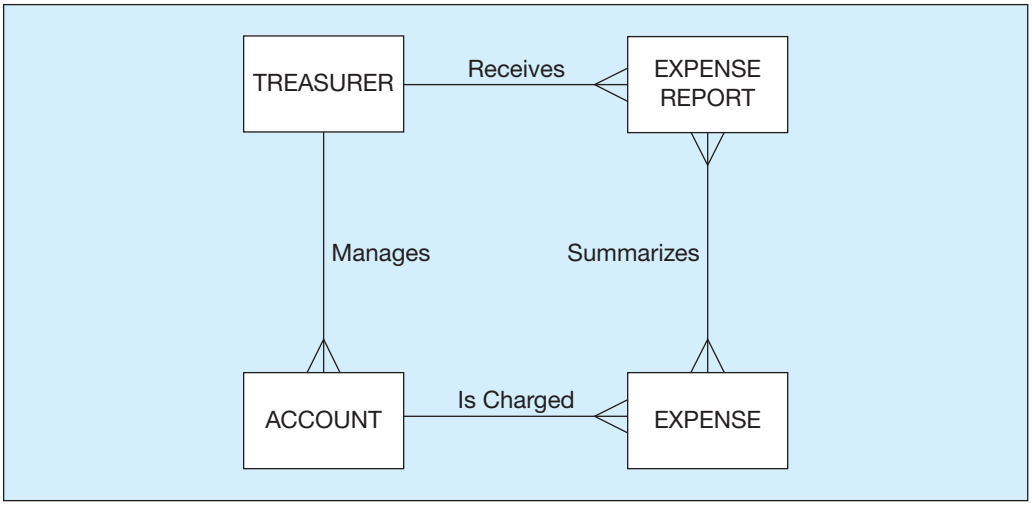
Entity type: EMPLOYEE

Attributes	Attribute Data Type	Example Instance	Example Instance
Employee Number	CHAR (10)	64217836	53410197
Name	CHAR (25)	Michelle Brady	David Johnson
Address	CHAR (30)	100 Pacific Avenue	450 Redwood Drive
City	CHAR (20)	San Francisco	Redwood City
State	CHAR (2)	CA	CA
Zip Code	CHAR (9)	98173	97142
Date Hired	DATE	03-21-1992	08-16-1994
Birth Date	DATE	06-19-1968	09-04-1975

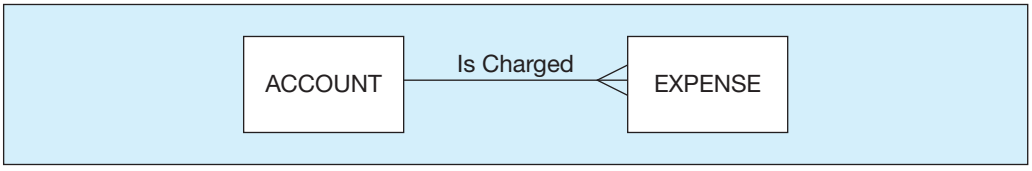
FIGURE 2-3 Entity type EMPLOYEE with two instances

FIGURE 2-4 Example of inappropriate entities

(a) System user (Treasurer) and output (Expense Report) shown as entities



(b) E-R diagram with only the necessary entities



Consider Figure 2-4a, which might be drawn to represent a database needed for a college sorority's expense system. (For simplicity in this and some other figures, we show only one name for a relationship.) In this situation, the sorority treasurer manages accounts, receives expense reports, and records expense transactions against each account. However, do we need to keep track of data about the Treasurer (the TREASURER entity type) and her supervision of accounts (the Manages relationship) and receipt of reports (the Receives relationship)? The Treasurer is the person entering data about accounts and expenses and receiving expense reports. That is, she is a user of the database. Because there is only one Treasurer, TREASURER data do not need to be kept. Further, is the EXPENSE REPORT entity necessary? Because an expense report is computed from expense transactions and account balances, it is the result of extracting data from the database and received by the Treasurer. Even though there will be multiple instances of expense reports given to the Treasurer over time, data needed to compute the report contents each time are already represented by the ACCOUNT and EXPENSE entity types.

Another key to understanding why the ERD in Figure 2-4a might be in error is the nature of the *relationship names*, Receives and Summarizes. These relationship names refer to business activities that transfer or translate data, not to simply the association of one kind of data with another kind of data. The simple E-R diagram in Figure 2-4b shows entities and a relationship that would be sufficient to handle the sorority expense system as described here. See Problem and Exercise 2-43 for a variation on this situation.

STRONG VERSUS WEAK ENTITY TYPES Most of the basic entity types to identify in an organization are classified as strong entity types. A **strong entity type** is one that exists independently of other entity types. (Some data modeling software, in fact, use the term *independent entity*.) Examples include STUDENT, EMPLOYEE, AUTOMOBILE, and COURSE. Instances of a strong entity type always have a unique characteristic (called an *identifier*)—that is, an attribute or a combination of attributes that uniquely distinguish each occurrence of that entity.

Strong entity type
An entity that exists independently of other entity types.

In contrast, a **weak entity type** is an entity type whose existence depends on some other entity type. (Some data modeling software, in fact, use the term *dependent entity*, and some data modeling tools make no distinction between strong and weak entities.) A weak entity type has no business meaning in an E-R diagram without the entity on which it depends. The entity type on which the weak entity type depends is called the **identifying owner** (or simply *owner* for short). A weak entity type does not typically

Weak entity type
An entity type whose existence depends on some other entity type.

Identifying owner
The entity type on which the weak entity type depends.

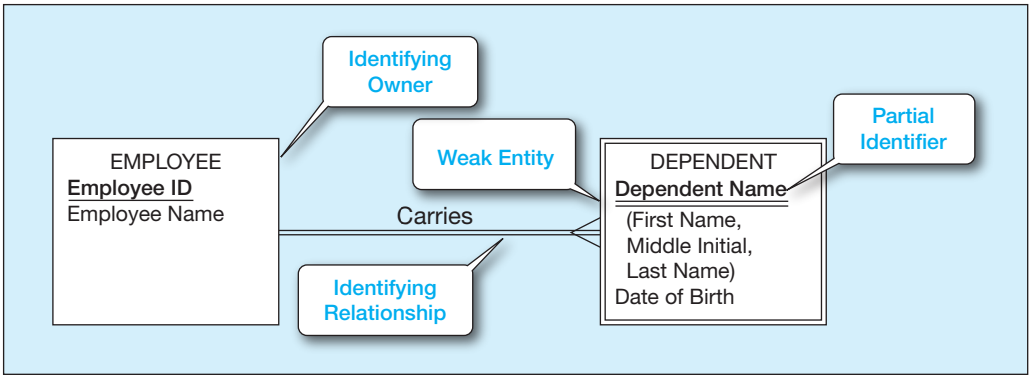


FIGURE 2-5 Example of a weak entity and its identifying relationship

have its own identifier. Generally, on an E-R diagram, a weak entity type has an attribute that serves as a *partial* identifier. During a later design stage (described in Chapter 4), a full identifier will be formed for the weak entity by combining the partial identifier with the identifier of its owner or by representing the weak entity as a strong entity with a surrogate, nonintelligent identifier attribute and the partial identifier as a regular attribute of this entity.

An example of a weak entity type with an identifying relationship is shown in Figure 2-5. EMPLOYEE is a strong entity type with identifier Employee ID (we note the identifier attribute by underlining it). DEPENDENT is a weak entity type, as indicated by the double-lined rectangle. The relationship between a weak entity type and its owner is called an **identifying relationship**. In Figure 2-5, Carries is the identifying relationship (indicated by the double line). The attribute Dependent Name serves as a *partial* identifier of DEPENDENT. (Dependent Name is a composite attribute that can be broken into component parts, as we describe later.) We use a double underline to indicate a partial identifier. During a later design stage, Dependent Name will be combined with Employee ID (the identifier of the owner) to form a full identifier for DEPENDENT. Some additional examples of strong and weak entity pairs are BOOK–BOOK COPY, PRODUCT–SERIAL PRODUCT, and COURSE–COURSE OFFERING.

Identifying relationship
The relationship between a weak entity type and its owner.

NAMING AND DEFINING ENTITY TYPES In addition to the general guidelines for naming and defining data objects, there are a few special guidelines for *naming* entity types, which follow:

- An entity type name is a *singular noun* (such as CUSTOMER, STUDENT, or AUTOMOBILE); an entity is a person, a place, an object, an event, or a concept, and the name is for the entity type, which represents a set of entity instances (i.e., STUDENT represents students Hank Finley, Jean Krebs, and so forth). It is common to also specify the plural form (possibly in a CASE tool repository accompanying the E-R diagram) because sometimes the E-R diagram is read best by using plurals. For example, in Figure 2-1, we would say that a SUPPLIER may supply ITEMs. Because plurals are not always formed by adding an *s* to the singular noun, it is best to document the exact plural form.
- An entity type name should be *specific to the organization*. Thus, one organization may use the entity type name CUSTOMER, and another organization may use the entity type name CLIENT (this is one task, e.g., done to customize a purchased data model). The name should be descriptive for everyone in the organization and distinct from all other entity type names within that organization. For example, a PURCHASE ORDER for orders placed with suppliers is distinct from a CUSTOMER ORDER for orders placed with a company by its customers. Both of these entity types cannot be named ORDER.
- An entity type name should be *concise*, using as few words as possible. For example, in a university database, an entity type REGISTRATION for the event of a student registering for a class is probably a sufficient name for this entity type; STUDENT REGISTRATION FOR CLASS, although precise, is probably too wordy because the reader will understand REGISTRATION from its use with other entity types.

- An *abbreviation*, or a *short name*, should be specified for each entity type name, and the abbreviation may be sufficient to use in the E-R diagram; abbreviations must follow all of the same rules as do the full entity names.
- *Event entity types* should be named for the result of the event, not the activity or process of the event. For example, the event of a project manager assigning an employee to work on a project results in an ASSIGNMENT, and the event of a student contacting his or her faculty adviser seeking some information is a CONTACT.
- The *name* used for the same entity type *should be the same* on all E-R diagrams on which the entity type appears. Thus, as well as being specific to the organization, the name used for an entity type should be a standard, adopted by the organization for all references to the same kind of data. However, some entity types will have aliases, or alternative names, which are synonyms used in different parts of the organization. For example, the entity type ITEM may have aliases of MATERIAL (for production) and DRAWING (for engineering). Aliases are specified in documentation about the database, such as the repository of a CASE tool.

There are also some specific guidelines for *defining* entity types, which follow:

- *An entity type definition usually starts with* "An X is …." This is the most direct and clear way to state the meaning of an entity type.
- An entity type definition should *include a statement of what the unique characteristic is for each instance of the entity type*. In many cases, stating the identifier for an entity type helps convey the meaning of the entity. An example for Figure 2-4b is "An expense is a payment for the purchase of some good or service. An expense is identified by a journal entry number."
- An entity type definition should make it clear what *entity instances are included and not included* in the entity type; often, it is necessary to list the kinds of entities that are excluded. For example, "A customer is a person or organization that has placed an order for a product from us or one that we have contacted to advertise or promote our products. A customer does not include persons or organizations that buy our products only through our customers, distributors, or agents."
- An entity type definition often includes a description of *when an instance of the entity type is created and deleted*. For example, in the previous bullet point, a customer instance is implicitly created when the person or organization places its first order; because this definition does not specify otherwise, implicitly a customer instance is never deleted, or it is deleted based on general rules that are specified about the purging of data from the database. A statement about when to delete an entity instance is sometimes referred to as the retention of the entity type. A possible deletion statement for a customer entity type definition might be "A customer ceases to be a customer if it has not placed an order for more than three years."
- For some entity types, the definition must specify *when an instance might change into an instance of another entity type*. For example, consider the situation of a construction company for which bids accepted by potential customers become contracts. In this case, a bid might be defined by "A bid is a legal offer by our organization to do work for a customer. A bid is created when an officer of our company signs the bid document; a bid becomes an instance of contract when we receive a copy of the bid signed by an officer of the customer." This definition is also a good example to note how one definition can use other entity type names (in this case, the definition of bid uses the entity type name CUSTOMER).
- For some entity types, the definition must specify *what history is to be kept about instances of the entity type*. For example, the characteristics of an ITEM in Figure 2-1 may change over time, and we may need to keep a complete history of the individual values and when they were in effect. As you will see in some examples later, such statements about keeping history may have ramifications about how we represent the entity type on an E-R diagram and eventually how we store data for the entity instances.

Attributes

Each entity type has a set of attributes associated with it. An **attribute** is a property or characteristic of an entity type that is of interest to the organization. (Later, you will see that some types of relationships may also have attributes.) Thus, an attribute has a noun name. Following are some typical entity types and their associated attributes:

Attribute
A property or characteristic of an entity or relationship type that is of interest to the organization.

STUDENT	Student ID, Student Name, Home Address, Phone Number, Major
AUTOMOBILE	Vehicle ID, Color, Weight, Horsepower
EMPLOYEE	Employee ID, Employee Name, Payroll Address, Skill

In naming attributes, we use an initial capital letter followed by lowercase letters. If an attribute name consists of more than one word, we use a space between the words and we start each word with a capital letter, for example, Employee Name or Student Home Address. In E-R diagrams, we represent an attribute by placing its name in the entity it describes. Attributes may also be associated with relationships, as described later. Note that an attribute is associated with exactly one entity or relationship.

Notice in Figure 2-5 that all of the attributes of DEPENDENT are characteristics only of an employee's dependent, not characteristics of an employee. In traditional E-R notation, an entity type (not just weak entities but any entity) does not include attributes of entities to which it is related (what might be called foreign attributes). For example, DEPENDENT does not include any attribute that indicates to which employee this dependent is associated. This nonredundant feature of the E-R data model is consistent with the shared data property of databases. Because of relationships, which we discuss shortly, someone accessing data from a database will be able to associate attributes from related entities (e.g., show on a display screen a Dependent Name and the associated Employee Name).

REQUIRED VERSUS OPTIONAL ATTRIBUTES Each entity (or instance of an entity type) potentially has a value associated with each of the attributes of that entity type. An attribute that must be present for each entity instance is called a **required attribute**, whereas an attribute that may not have a value is called an **optional attribute**. For example, Figure 2-6 shows two STUDENT entities (instances) with their respective attribute values. The only optional attribute for STUDENT is Major. (Some students, specifically Melissa Kraft in this example, have not chosen a major yet; MIS would, of course, be a great career choice!) However, every student must, by the rules of the organization, have values for all the other attributes; *that is, we cannot store any data about a student in a STUDENT entity instance unless there are values for all the required attributes*. In various E-R diagramming notations, a symbol might appear in front of each attribute to indicate whether it is required (e.g., *) or optional (e.g., o), or required attributes will be in **boldface**, whereas optional attributes will be in normal font (the format we use in this text); in many cases, required or optional is indicated within supplemental documentation.

Required attribute
An attribute that must have a value for every entity (or relationship) instance with which it is associated.

Optional attribute
An attribute that may not have a value for every entity (or relationship) instance with which it is associated.

Entity type: STUDENT				
Attributes	**Attribute Data Type**	**Required or Optional**	**Example Instance**	**Example Instance**
Student ID	CHAR (10)	Required	28-618411	26-844576
Student Name	CHAR (40)	Required	Michael Grant	Melissa Kraft
Home Address	CHAR (30)	Required	314 Baker St.	1422 Heft Ave
Home City	CHAR (20)	Required	Centerville	Miami
Home State	CHAR (2)	Required	OH	FL
Home Zip Code	CHAR (9)	Required	45459	33321
Major	CHAR (3)	Optional	MIS	

FIGURE 2-6 Entity type STUDENT with required and optional attributes

In Chapter 3, when you study entity supertypes and subtypes, you will see how sometimes optional attributes imply that there are different types of entities. (For example, we may want to consider students who have not declared a major as a subtype of the STUDENT entity type.) An attribute without a value is said to be null. Thus, each entity has an identifying attribute, which we discuss in a subsequent section, plus one or more other attributes. If you try to create an entity that has only an identifier, that entity is likely not legitimate. Such a data structure may simply hold a list of legal values for some attribute, which is better kept outside the database.

SIMPLE VERSUS COMPOSITE ATTRIBUTES Some attributes can be broken down into meaningful component parts (detailed attributes). A common example is Name, which you saw in Figure 2-5; another is Address, which can usually be broken down into the following component attributes: Street Address, City, State, and Postal Code. A **composite attribute** is an attribute, such as Address, that has meaningful component parts, which are more detailed attributes. Figure 2-7 shows the notation that we use for composite attributes applied to this example. Most drawing tools do not have a notation for composite attributes, so you simply list all the component parts.

Composite attribute
An attribute that has meaningful component parts (attributes).

Composite attributes provide considerable flexibility to users, who can either refer to the composite attribute as a single unit or else refer to individual components of that attribute. Thus, for example, a user can either refer to Address or refer to one of its components, such as Street Address. The decision about whether to subdivide an attribute into its component parts depends on whether users will need to refer to those individual components, and hence, they have organizational meaning. Of course, you must always attempt to anticipate possible future usage patterns for the database.

Simple (or atomic) attribute
An attribute that cannot be broken down into smaller components that are meaningful to the organization.

A **simple (or atomic) attribute** is an attribute that cannot be broken down into smaller components that are meaningful for the organization. For example, all the attributes associated with AUTOMOBILE are simple: Vehicle ID, Color, Weight, and Horsepower.

SINGLE-VALUED VERSUS MULTIVALUED ATTRIBUTES Figure 2-6 shows two entity instances with their respective attribute values. For each entity instance, each of the attributes in the figure has one value. It frequently happens that there is an attribute that may have more than one value for a given instance. For example, the EMPLOYEE entity type in Figure 2-8 has an attribute named Skill, whose values record the skill

FIGURE 2-7 A composite attribute

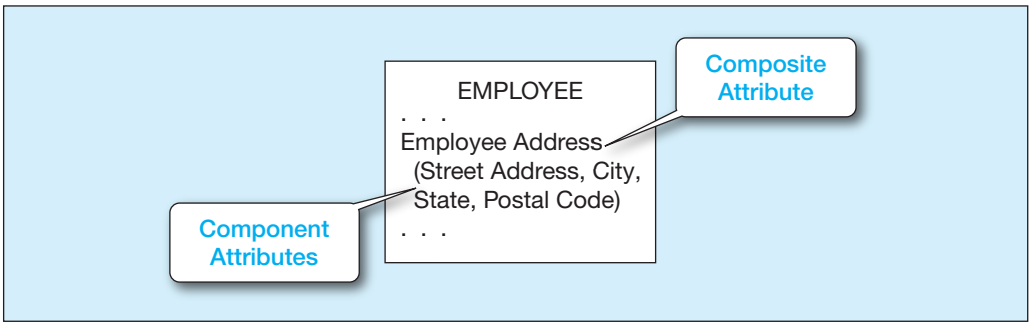
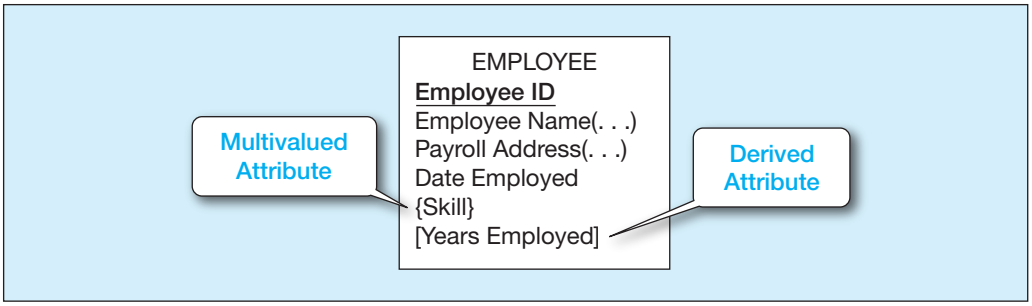


FIGURE 2-8 Entity with multivalued attribute (Skill) and derived attribute (Years Employed)



(or skills) for that employee. Of course, some employees may have more than one skill, such as PHP Programmer and C++ Programmer. A **multivalued attribute** is an attribute that may take on more than one value for a given entity (or relationship) instance. In this text, we indicate a multivalued attribute with curly brackets around the attribute name, as shown for the Skill attribute in the EMPLOYEE example in Figure 2-8. In Microsoft Visio, once an attribute is placed in an entity, you can edit that attribute (column), select the Collection tab, and choose one of the options. (Typically, MultiSet will be your choice, but one of the other options may be more appropriate for a given situation.) Other E-R diagramming tools may use an asterisk (*) after the attribute name, or you may have to use supplemental documentation to specify a multivalued attribute.

Multivalued attribute

An attribute that may take on more than one value for a given entity (or relationship) instance.

Multivalued and composite are different concepts, although beginner data modelers often confuse these terms. Skill, a multivalued attribute, may occur multiple times for each employee; Employee Name and Payroll Address are both likely composite attributes, each of which occurs once for each employee but which have component, more atomic attributes that are not shown in Figure 2-8 for simplicity. It is possible to have a multivalued composite attribute. For example, a Customer Address composite attribute may have a variable number of values for a given customer (home, office, seasonal, etc.). See Problem and Exercise 2-38 to review the concepts of composite and multivalued attributes.

STORED VERSUS DERIVED ATTRIBUTES Some attribute values that are of interest to users can be calculated or derived from other related attribute values that are stored in the database. When users want to use such derived values in calculations and displays, it is likely helpful for them to show these derived attributes in an E-R diagram. For example, suppose for an organization, the EMPLOYEE entity type has a Date Employed attribute. If users need to know how many years a person has been employed, that value can be calculated using Date Employed and today's date. A **derived attribute** is an attribute whose values can be calculated from related attribute values (plus possibly data not in the database, such as today's date, the current time, or a security code provided by a system user). We indicate a derived attribute in an E-R diagram by using square brackets around the attribute name, as shown in Figure 2-8 for the Years Employed attribute. Some E-R diagramming tools use a notation of a forward slash (/) in front of the attribute name to indicate that it is derived. (This notation is borrowed from UML for a virtual attribute.) The method for calculating the derived attribute may be stored with the description of the associated entity (or relationship), and this method may be either coded into the physical database definition or programmed into application programs based on the one method description associated with the E-R diagram and database documentation.

Derived attribute

An attribute whose values can be calculated from related attribute values.

In some situations, the value of an attribute can be derived from attributes in related entities. For example, consider an invoice created for each customer at Pine Valley Furniture Company. Order Total would be an attribute of the INVOICE entity, which indicates the total dollar amount that is billed to the customer. The value of Order Total can be computed by summing the Extended Price values (unit price times quantity sold) for the various line items that are billed on the invoice. Formulas for computing values such as this are one type of business rule.

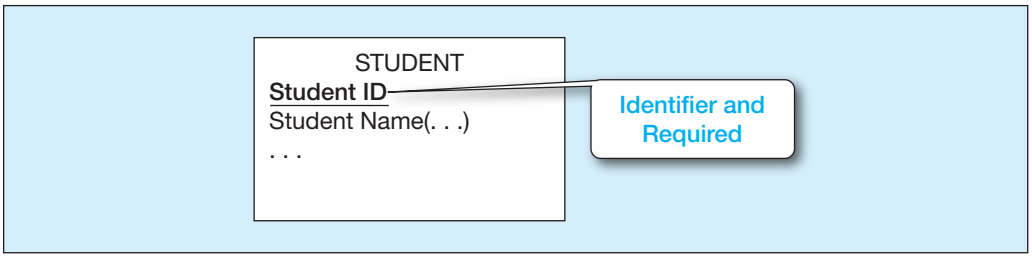
IDENTIFIER ATTRIBUTE An **identifier** is an attribute (or combination of attributes) whose value distinguishes individual instances of an entity type. That is, no two instances of the entity type may have the same value for the identifier attribute. The identifier for the STUDENT entity type introduced earlier is Student ID, whereas the identifier for AUTOMOBILE is Vehicle ID. Notice that an attribute such as Student Name is not a candidate identifier because many students may potentially have the same name and students, like all people, can change their names. To be a candidate identifier, each entity instance must have a single value for the attribute, and the attribute must be associated with the entity. We underline identifier names on the E-R diagram, as shown in the STUDENT entity type example in Figure 2-9a. To be an identifier, the attribute is also required (so the distinguishing value must exist), so an identifier

Identifier

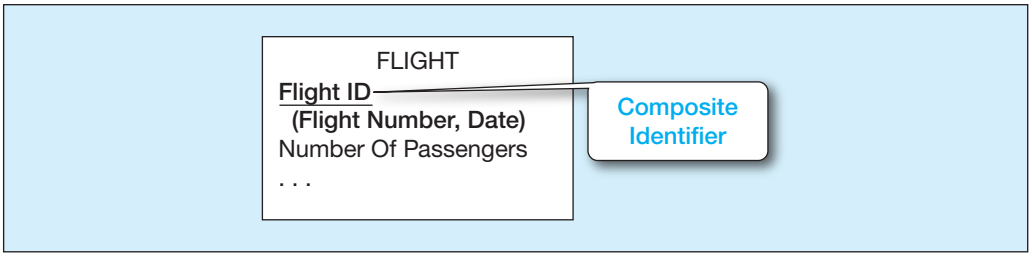
An attribute (or combination of attributes) whose value distinguishes instances of an entity type.

FIGURE 2-9 Simple and composite identifier attributes

(a) Simple identifier attribute



(b) Composite identifier attribute



is also in bold. Some E-R drawing software will place a symbol, called a stereotype, in front of the identifier (e.g., <<ID>> or <<PK>>).

Composite identifier
An identifier that consists of a composite attribute.

For some entity types, there is no single (or atomic) attribute that can serve as the identifier (i.e., that will ensure uniqueness). However, two (or more) attributes used in combination may serve as the identifier. A **composite identifier** is an identifier that consists of a composite attribute. Figure 2-9b shows the entity FLIGHT with the composite identifier Flight ID. Flight ID in turn has component attributes Flight Number and Date. This combination is required to identify uniquely individual occurrences of FLIGHT.

We use the convention that the composite attribute (Flight ID) is underlined to indicate it is the identifier, whereas the component attributes are not underlined. Some data modelers think of a composite identifier as "breaking a tie" created by a simple identifier. Even with Flight ID, a data modeler would ask a question, such as "Can two flights with the same number occur on the same date?" If so, yet another attribute is needed to form the composite identifier and to break the tie.

Some entities may have more than one candidate identifier. If there is more than one candidate identifier, the designer must choose one of them as the identifier. Bruce (1992) suggests the following criteria for selecting identifiers:

1. Choose an identifier that will not change its value over the life of each instance of the entity type. For example, the combination of Employee Name and Payroll Address (even if unique) would be a poor choice as an identifier for EMPLOYEE because the values of Employee Name and Payroll Address could easily change during an employee's term of employment.
2. Choose an identifier such that for each instance of the entity, the attribute is guaranteed to have valid values and not be null (or unknown). If the identifier is a composite attribute, such as Flight ID in Figure 2-9b, make sure that all parts of the identifier will have valid values.
3. Avoid the use of so-called intelligent identifiers (or keys), whose structure indicates classifications, locations, and so on. For example, the first two digits of an identifier value may indicate the warehouse location. Such codes are often changed as conditions change, which renders the identifier values invalid.
4. Consider substituting single-attribute surrogate identifiers for large composite identifiers. For example, an attribute called Game Number could be used for the entity type GAME instead of the combination of Home Team and Visiting Team.

NAMING AND DEFINING ATTRIBUTES In addition to the general guidelines for naming data objects, there are a few special guidelines for naming attributes, which follow:

- An attribute name is a *singular noun or noun phrase* (such as Customer ID, Age, Product Minimum Price, or Major). Attributes, which materialize as data values,

are concepts or physical characteristics of entities. Concepts and physical characteristics are described by nouns.

- An attribute name should be *unique*. No two attributes of the same entity type may have the same name, and it is desirable, for clarity purposes, that no two attributes across all entity types have the same name.
- To make an attribute name unique and for clarity purposes, *each attribute name should follow a standard format*. For example, your university may establish Student GPA, as opposed to GPA of Student, as an example of the standard format for attribute naming. The format to be used will be established by each organization. A common format is [Entity type name { [Qualifier] }] Class, where [. . .] is an optional clause, and { . . . } indicates that the clause may repeat. *Entity type name* is the name of the entity with which the attribute is associated. The entity type name may be used to make the attribute name explicit. It is almost always used for the identifier attribute (e.g., Customer ID) of each entity type. *Class* is a phrase from a list of phrases defined by the organization that are the permissible characteristics or properties of entities (or abbreviations of these characteristics). For example, permissible values (and associated approved abbreviations) for Class might be Name (Nm), Identifier (ID), Date (Dt), or Amount (Amt). Class is, obviously, required. *Qualifier* is a phrase from a list of phrases defined by the organization that are used to place constraints on classes. One or more qualifiers may be needed to make each attribute of an entity type unique. For example, a qualifier might be Maximum (Max), Hourly (Hrly), or State (St). A qualifier may not be necessary: Employee Age and Student Major are both fully explicit attribute names. Sometimes a qualifier is necessary. For example, Employee Birth Date and Employee Hire Date are two attributes of Employee that require one qualifier. More than one qualifier may be necessary. For example, Employee Residence City Name (or Emp Res Cty Nm) is the name of an employee's city of residence, and Employee Tax City Name (or Emp Tax Cty Nm) is the name of the city in which an employee pays city taxes.
- *Similar attributes* of different entity types *should use the same qualifiers and classes*, as long as those are the names used in the organization. For example, the city of residence for faculty and students should be, respectively, Faculty Residence City Name and Student Residence City Name. Using similar names makes it easier for users to understand that values for these attributes come from the same possible set of values, what we will call *domains*. Users may want to take advantage of common domains in queries (e.g., find students who live in the same city as their adviser), and it will be easier for users to recognize that such a matching may be possible if the same qualifier and class phrases are used.

There are also some specific guidelines for defining attributes, which follow:

- An attribute definition states *what the attribute is and possibly why it is important*. The definition will often parallel the attribute's name; for example, Student Residence City Name could be defined as "The name of the city in which a student maintains his or her permanent residence."
- An attribute definition should make it clear *what is included and not included* in the attribute's value; for example, "Employee Monthly Salary Amount is the amount of money paid each month in the currency of the country of residence of the employee, exclusive of any benefits, bonuses, reimbursements, or special payments."
- Any *aliases*, or alternative names, for the attribute can be specified in the definition or may be included elsewhere in documentation about the attribute, possibly stored in the repository of a CASE tool used to maintain data definitions.
- It may also be desirable to state in the definition *the source of values for the attribute*. Stating the source may make the meaning of the data clearer. For example, "Customer Standard Industrial Code is an indication of the type of business for the customer. Values for this code come from a standard set of values provided by the Federal Trade Commission and are found on a CD we purchase named SIC provided annually by the FTC."

- An attribute definition (or other specification in a CASE tool repository) also should indicate *if a value for the attribute is required or optional*. This business rule about an attribute is important for maintaining data integrity. The identifier attribute of an entity type is, by definition, required. If an attribute value is required, then to create an instance of the entity type, a value of this attribute must be provided. Required means that an entity instance must always have a value for this attribute, not just when an instance is created. Optional means that a value may not exist for an instance of an entity instance to be stored. Optional can be further qualified by stating whether once a value is entered, a value must always exist. For example, "Employee Department ID is the identifier of the department to which the employee is assigned. An employee may not be assigned to a department when hired (so this attribute is initially optional), but once an employee is assigned to a department, the employee must always be assigned to some department."
- An attribute definition (or other specification in a CASE tool repository) may also indicate *whether a value for the attribute may change* once a value is provided and before the entity instance is deleted. This business rule also controls data integrity. Nonintelligent identifiers may not change values over time. To assign a new nonintelligent identifier to an entity instance, that instance must first be deleted and then re-created.
- For a multivalued attribute, the attribute definition should indicate *the maximum and minimum number of occurrences of an attribute value for an entity instance*. For example, "Employee Skill Name is the name of a skill an employee possesses. Each employee must possess at least one skill, and an employee can choose to list at most 10 skills." The reason for a multivalued attribute may be that a history of the attribute needs to be kept. For example, "Employee Yearly Absent Days Number is the number of days in a calendar year the employee has been absent from work. An employee is considered absent if he or she works less than 50 percent of the scheduled hours in the day. A value for this attribute should be kept for each year in which the employee works for our company."
- An attribute definition may also indicate *any relationships that attribute has with other attributes*. For example, "Employee Vacation Days Number is the number of days of paid vacation for the employee. If the employee has a value of 'Exempt' for Employee Type, then the maximum value for Employee Vacation Days Number is determined by a formula involving the number of years of service for the employee."

MODELING RELATIONSHIPS

Relationships are the glue that holds together the various components of an E-R model. Intuitively, a *relationship* is an association representing an interaction among the instances of one or more entity types that is of interest to the organization. Thus, a relationship has a verb phrase name. Relationships and their characteristics (degree and cardinality) represent business rules, and usually relationships represent the most complex business rules shown in an ERD. In other words, this is where data modeling gets really interesting and fun, as well as crucial for controlling the integrity of a database. Relationships are essential for almost every meaningful use of a database; for example, relationships allow iTunes to find the music you've purchased, your cell phone company to find all the text messages in one of your SMS threads, or the campus nurse to see how different students have reacted to different treatments to the latest influenza on campus. So, fun and essential—modeling relationships will be a rewarding skill for you.

To understand relationships more clearly, we must distinguish between relationship types and relationship instances. To illustrate, consider the entity types EMPLOYEE and COURSE, where COURSE represents training courses that may be taken by employees. To track courses that have been completed by particular employees, you would define a relationship called Completes between the two entity types (see Figure 2-10a). This is a many-to-many relationship because each employee may complete any number of courses (zero, one, or many courses), whereas a given course may be completed by any number of employees (nobody, one employee, or many employees). For example, in Figure 2-10b, the employee Melton has completed three courses (C++, COBOL, and

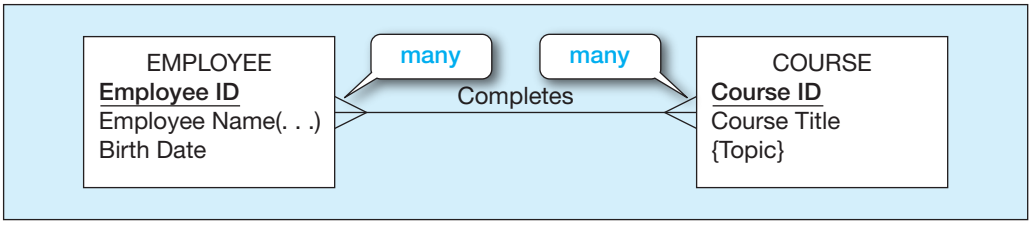
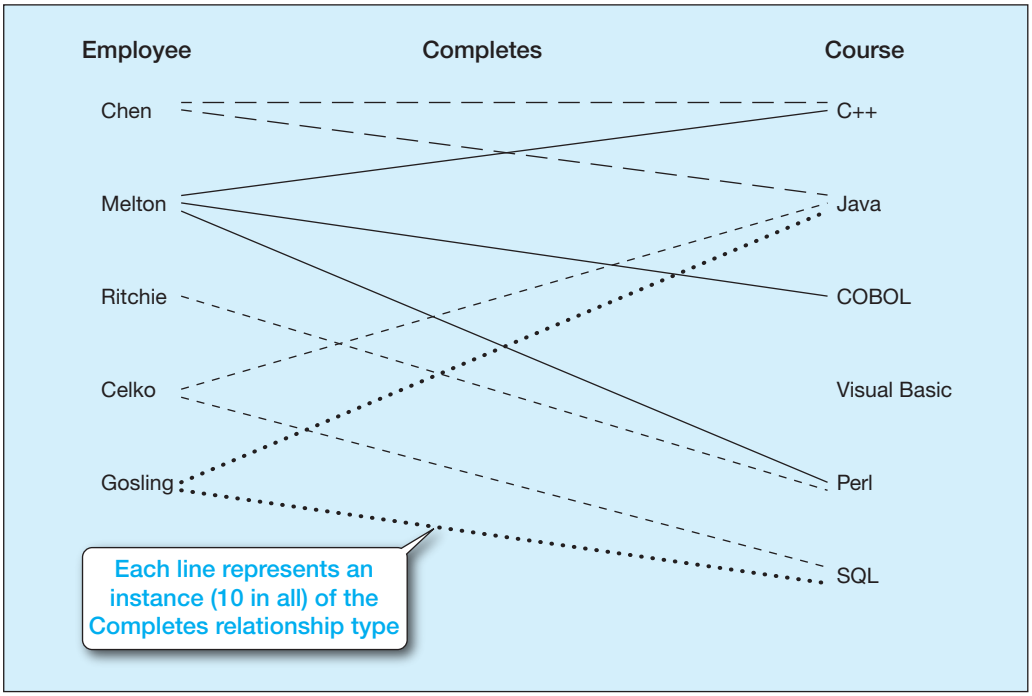


FIGURE 2-10 Relationship type and instances

(a) Relationship type (Completes)



(b) Relationship instances

Perl). The SQL course has been completed by two employees (Celko and Gosling), and the Visual Basic course has not been completed by anyone.

In this example, there are two entity types (EMPLOYEE and COURSE) that participate in the relationship named Completes. In general, any number of entity types (from one to many) may participate in a relationship.

We frequently use in this and subsequent chapters the convention of a single verb phrase label to represent a relationship. Because relationships often occur due to an organizational event, entity instances are related because an action was taken; thus, a verb phrase is appropriate for the label. This verb phrase should be in the present tense and descriptive. There are, however, many ways to represent a relationship. Some data modelers prefer the format with two relationship names, one to name the relationship in each direction. One or two verb phrases have the same structural meaning, so you may use either format as long as the meaning of the relationship in each direction is clear.

Basic Concepts and Definitions in Relationships

A **relationship type** is a meaningful association between (or among) entity types. The phrase *meaningful association* implies that the relationship allows us to answer questions that could not be answered given only the entity types. A relationship type is denoted by a line labeled with the name of the relationship, as in the example shown in Figure 2-10a, or with two names, as in Figure 2-1. We suggest you use a short, descriptive verb phrase that is meaningful to the user in naming the relationship. (We say more about naming and defining relationships later in this section.)

Relationship type
A meaningful association between (or among) entity types.

A **relationship instance** is an association between (or among) entity instances, where each relationship instance associates exactly one entity instance from each participating entity type (Elmasri and Navathe, 1994). For example, in Figure 2-10b, each of the 10 lines in the figure represents a relationship instance between one employee and

Relationship instance
An association between (or among) entity instances where each relationship instance associates exactly one entity instance from each participating entity type.

TABLE 2-2 Instances Showing Date Completed

Employee Name	Course Title	Date Completed
Chen	C++	06/2017
Chen	Java	09/2017
Melton	C++	06/2017
Melton	COBOL	02/2018
Melton	SQL	03/2017
Ritchie	Perl	11/2017
Celko	Java	03/2017
Celko	SQL	03/2018
Gosling	Java	09/2017
Gosling	Perl	06/2017

one course, indicating that the employee has completed that course. For example, the line between Employee Ritchie and Course Perl is one relationship instance.

ATTRIBUTES ON RELATIONSHIPS It is probably obvious to you that entities have attributes, but attributes may be associated with a many-to-many (or one-to-one) relationship, too. For example, suppose the organization wishes to record the date (month and year) when an employee completes each course. This attribute is named Date Completed. For some sample data, see Table 2-2.

Where should the attribute Date Completed be placed on the E-R diagram? Referring to Figure 2-10a, you will notice that Date Completed has not been associated with either the EMPLOYEE or the COURSE entity. That is because Date Completed is a property of the relationship Completes rather than a property of either entity. In other words, for each instance of the relationship Completes, there is a value for Date Completed. One such instance, for example, shows that the employee named Melton completed the course titled C++ in 06/2017.

A revised version of the ERD for this example is shown in Figure 2-11a. In this diagram, the attribute Date Completed is in a rectangle connected to the Completes relationship line. Other attributes might be added to this relationship if appropriate, such as Course Grade, Instructor, and Room Location. We will explain the A and B annotations below.

It is interesting to note that an attribute cannot be associated with a one-to-many relationship, such as Carries in Figure 2-5. For example, consider Dependent Date, similar to Date Completed above, for when the DEPENDENT begins to be carried by the EMPLOYEE. Because each DEPENDENT is associated with only one EMPLOYEE, such a date is unambiguously a characteristic of the DEPENDENT (i.e., for a given DEPENDENT, Dependent Date cannot vary by EMPLOYEE). So, if you ever have the urge to associate an attribute with a one-to-many relationship, "step away from the relationship!"

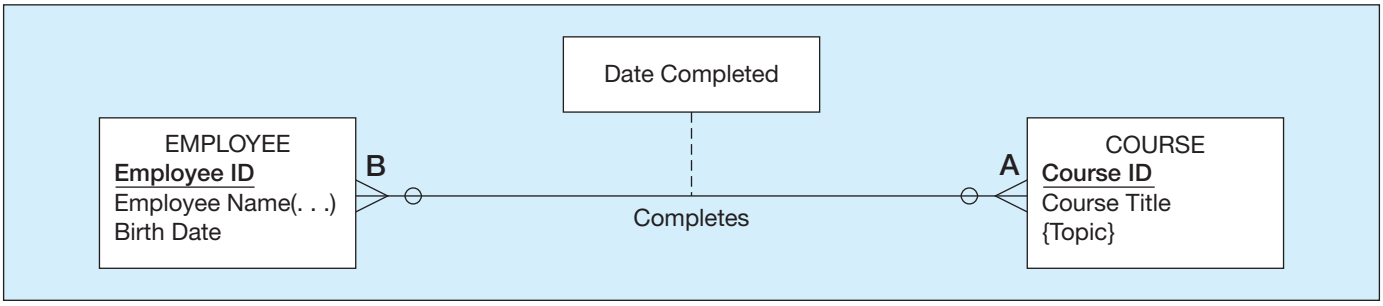
ASSOCIATIVE ENTITIES The presence of one or more attributes on a relationship suggests to the designer that the relationship should perhaps instead be represented as an entity type. To emphasize this point, most E-R drawing tools require that such attributes be placed in an entity type. An **associative entity** is an entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances. The associative entity CERTIFICATE is represented with the rectangle with rounded corners, as shown in Figure 2-11b. Most E-R drawing tools do not have a special symbol for an associative entity. Associative entities are sometimes referred to as gerunds because the relationship name (a verb) is usually converted to an entity name that is a noun. Note in Figure 2-11b that there are no relationship names on the lines between an associative entity and a strong entity. This is because the associative entity represents the relationship. Figure 2-11c shows how associative entities are drawn using Microsoft Visio, which is representative of how you

Associative entity

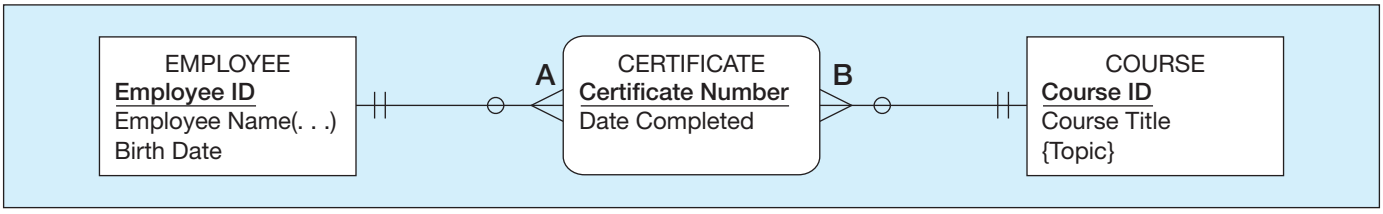
An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances.

FIGURE 2-11 An associative entity

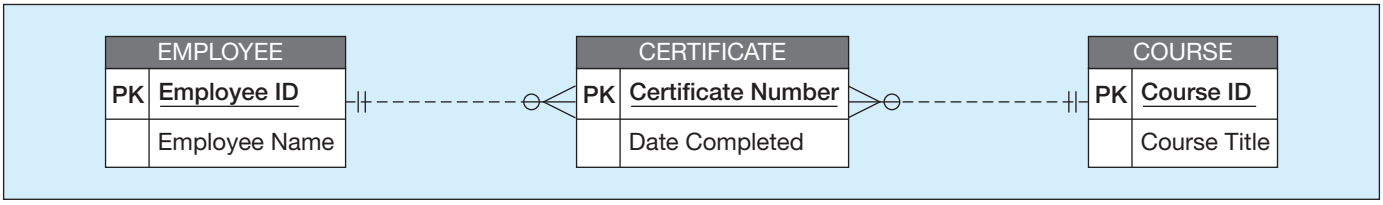
(a) Attribute on a relationship



(b) An associative entity (CERTIFICATE)



(c) An associative entity using Microsoft VISIO



would draw an associative entity with most E-R diagramming tools. In Visio, the relationship lines are dashed because CERTIFICATE does not include the identifiers of the related entities in its identifier. (Certificate Number is sufficient.)

How do you know whether to convert a relationship to an associative entity type? Following are four conditions that should exist:

1. All the relationships for the participating entity types are "many" relationships.
2. The resulting associative entity type has independent meaning to end users and, preferably, can be identified with a single-attribute identifier.
3. The associative entity has one or more attributes in addition to the identifier.
4. The associative entity participates in one or more relationships independent of the entities related in the associated relationship.

Figure 2-11b shows the relationship Completes converted to an associative entity type. In this case, the training department for the company has decided to award a certificate to each employee who completes a course. Thus, the entity is named CERTIFICATE, which certainly has independent meaning to end users. Also, each certificate has a number (Certificate Number) that serves as the identifier.

The attribute Date Completed is also included. Note also in Figure 2-11b and the Visio version of Figure 2-11c that both EMPLOYEE and COURSE are mandatory participants in the two relationships with CERTIFICATE. This is exactly what occurs when you have to represent a many-to-many relationship (Completes in Figure 2-11a) as two one-to-many relationships (the ones associated with CERTIFICATE in Figures 2-11b and 2-11c).

Notice that converting a relationship to an associative entity has caused the relationship notation to move. That is, the "many" cardinality now terminates at the associative entity rather than at each participating entity type. In Figure 2-11, this shows that an employee, who may complete one or more courses (notation A in Figure 2-11a), may be awarded more than one certificate (notation A in Figure 2-11b) and that a course, which may have one or more employees complete it (notation B in Figure 2-11a), may have

many certificates awarded (notation B in Figure 2-11b). See Problem and Exercise 2-42 for an interesting variation on Figure 2-11a, which emphasizes the rules for when to convert a many-to-many relationship, such as Completes, into an associative entity.

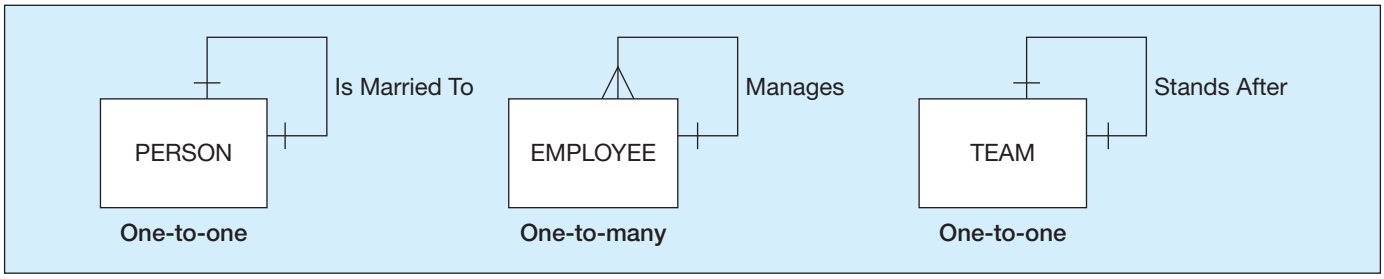
Degree of a Relationship

Degree
The number of entity types that participate in a relationship.

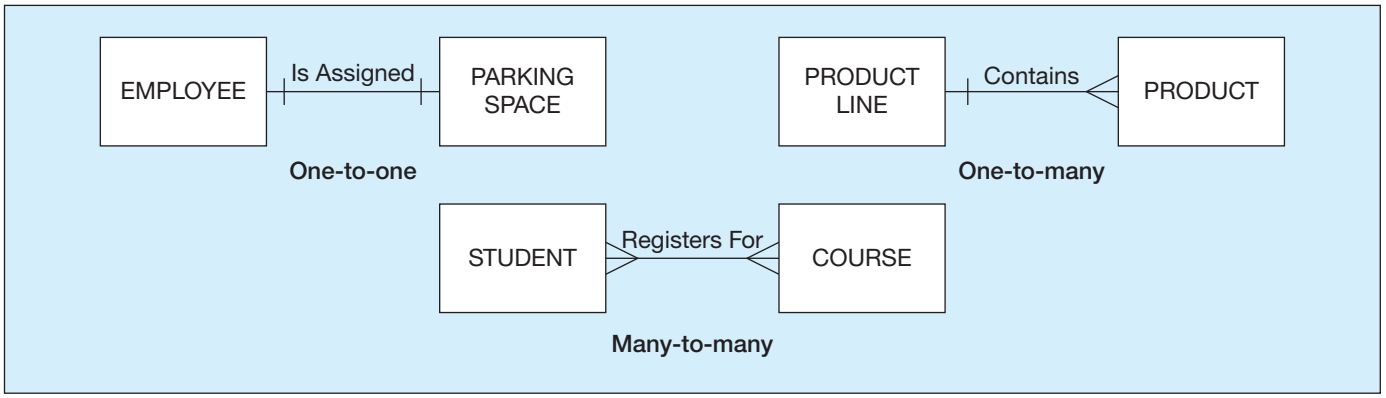
The **degree** of a relationship is the number of entity types that participate in that relationship. Thus, the relationship Completes in Figure 2-11 is of degree 2 because there are two entity types: EMPLOYEE and COURSE. The three most common relationship degrees in E-R models are unary (degree 1), binary (degree 2), and ternary (degree 3). Higher-degree relationships are possible, but they are rarely encountered in practice, so we restrict our discussion to these three cases. Examples of unary, binary, and ternary relationships appear in Figure 2-12. (Attributes are not shown in some figures for simplicity.)

FIGURE 2-12 Examples of relationships of different degrees

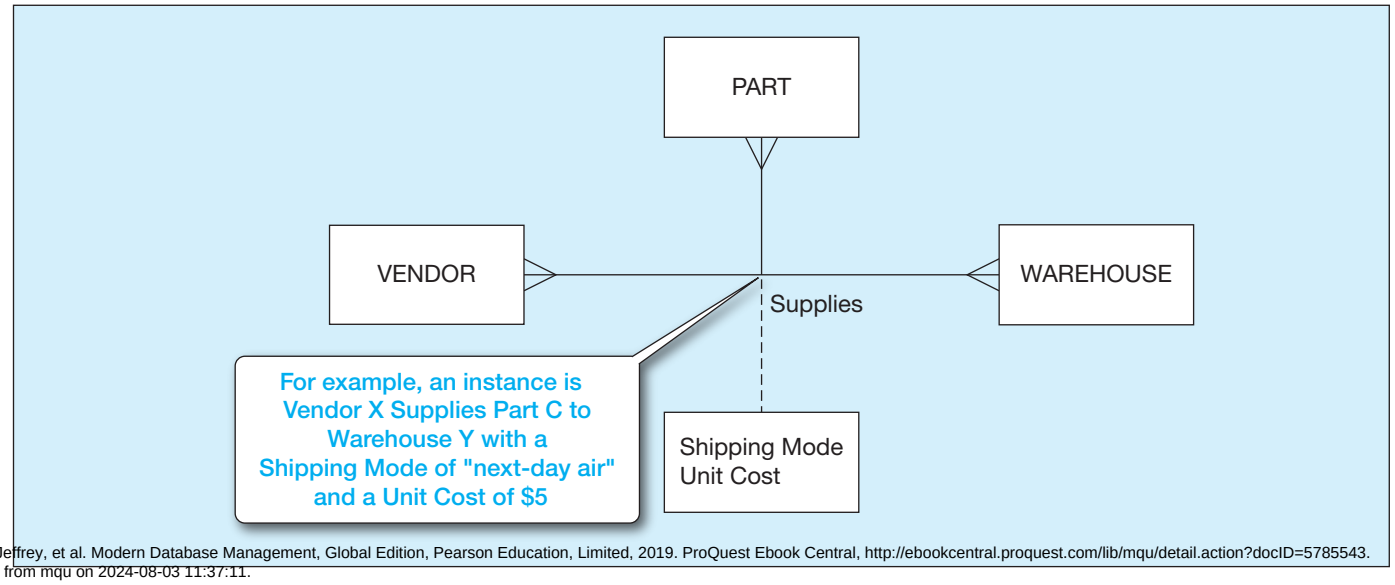
(a) Unary relationships



(b) Binary relationships



(c) Ternary relationship



As you look at Figure 2-12, understand that any particular data model represents a specific situation, not a generalization. For example, consider the Manages relationship in Figure 2-12a. In some organizations, it may be possible for one employee to be managed by many other employees (e.g., in a matrix organization). It is important when you develop an E-R model that you understand the business rules of the particular organization you are modeling.

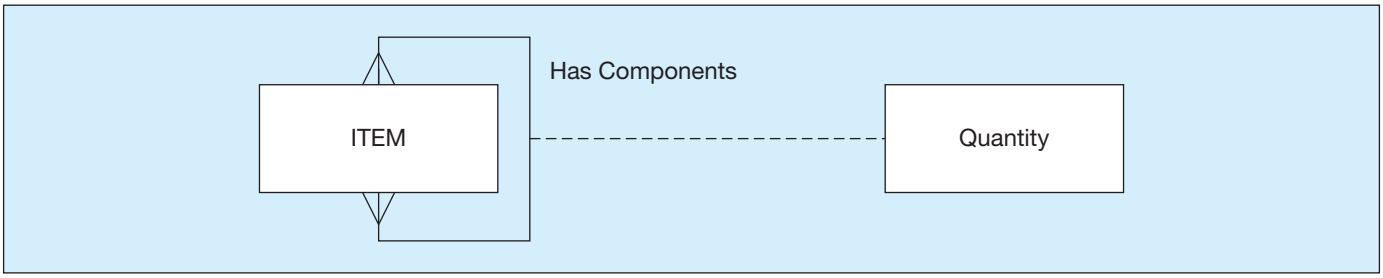
UNARY RELATIONSHIP A **unary relationship** is a relationship between the instances of a *single* entity type. (Unary relationships are also called *recursive relationships*.) Three examples are shown in Figure 2-12a. In the first example, Is Married To is shown as a one-to-one relationship between instances of the PERSON entity type. Because this is a one-to-one relationship, this notation indicates that only the current marriage, if one exists, needs to be kept about a person. What would change if we needed to retain the history of marriages for each person? See Review Question 2-20 and Problem and Exercise 2-34 for other business rules and their effect on the Is Married To relationship representation. In the second example, Manages is shown as a one-to-many relationship between instances of the EMPLOYEE entity type. Using this relationship, you could identify, for example, the employees who report to a particular manager. The third example is one case of using a unary relationship to represent a sequence, cycle, or priority list. In this example, sports teams are related by their standing in their league (the Stands After relationship). (Note: In these examples, we ignore whether these are mandatory- or optional-cardinality relationships or whether the same entity instance can repeat in the same relationship instance; we will introduce mandatory and optional cardinality in a later section of this chapter.)

Unary relationship
A relationship between instances of a single entity type.

Figure 2-13 shows an example of another unary relationship, called a *bill-of-materials structure*. Many manufactured products are made of assemblies, which in turn are composed of subassemblies and parts and so on. As shown in Figure 2-13a,

FIGURE 2-13 **Representing a bill-of-materials structure**

(a) Many-to-many relationship



(b) Two ITEM bill-of-materials structure instances

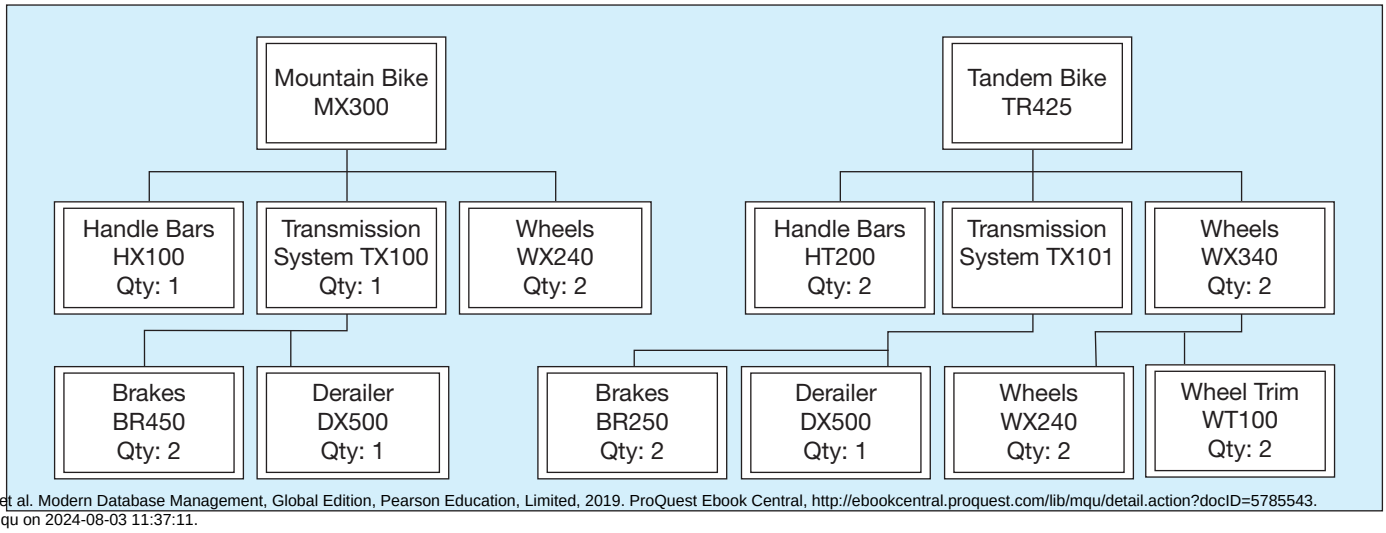
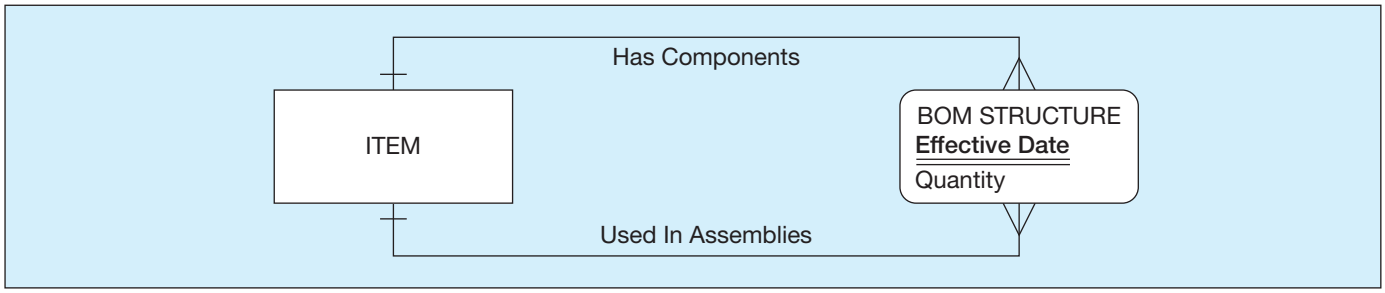


FIGURE 2-13 (*continued*)

(c) Associative entity



we can represent this structure as a many-to-many unary relationship. In this figure, the entity type ITEM is used to represent all types of components, and we use Has Components for the name of the relationship type that associates lower-level items with higher-level items.

Two occurrences of this bill-of-materials structure are shown in Figure 2-13b. Each of these diagrams shows the immediate components of each item as well as the quantities of that component. For example, item TX100 consists of item BR450 (quantity 2) and item DX500 (quantity 1). You can easily verify that the associations are in fact many-to-many. Several of the items have more than one component type (e.g., item MX300 has three immediate component types: HX100, TX100, and WX240). Also, some of the components are used in several higher-level assemblies. For example, item WX240 is used in both item MX300 and item WX340, even at different levels of the bill-of-materials. The many-to-many relationship guarantees that, for example, the same subassembly structure of WX240 (not shown) is used each time item WX240 goes into making some other item.

The presence of the attribute Quantity on the relationship suggests that the analyst consider converting the relationship Has Components to an associative entity. Figure 2-13c shows the entity type BOM STRUCTURE, which forms an association between instances of the ITEM entity type. A second (partial identifier) attribute (named Effective Date) has been added to BOM STRUCTURE to record the date when the specified quantity of this component was first used in the related assembly. Effective dates are often needed when a history of values is required. In practice, the identifiers of the component and assembly items along with Effective Date often become nonidentifier attributes, and a surrogate, nonintelligent identifier is used. Other data model structures can be used for unary relationships involving such hierarchies; we show some of these other structures in Chapter 9.

Binary relationship
A relationship between the instances of two entity types.

BINARY RELATIONSHIP A **binary relationship** is a relationship between the instances of two entity types and is the most common type of relationship encountered in data modeling. Figure 2-12b shows three examples. The first (one-to-one) indicates that an employee is assigned one parking place and that each parking place is assigned to one employee (at a given time). The second (one-to-many) indicates that a product line may contain several products and that each product belongs to only one product line. The third (many-to-many) shows that a student may register for more than one course and that each course may have many student registrants.

Ternary relationship
A simultaneous relationship among the instances of three entity types.

TERNARY RELATIONSHIP A **ternary relationship** is a *simultaneous* relationship among the instances of three entity types. A typical business situation that leads to a ternary relationship is shown in Figure 2-12c. In this example, vendors can supply various parts to warehouses. The relationship Supplies is used to record the specific parts that are supplied by a given vendor to a particular warehouse. Thus, there are three entity types involved: VENDOR, PART, and WAREHOUSE. There are two attributes on the relationship Supplies: Shipping Mode and Unit Cost. For example, one instance of Supplies might record the fact that vendor X can ship part C to warehouse Y, that the shipping mode is next-day air, and that the cost is $5 per unit.

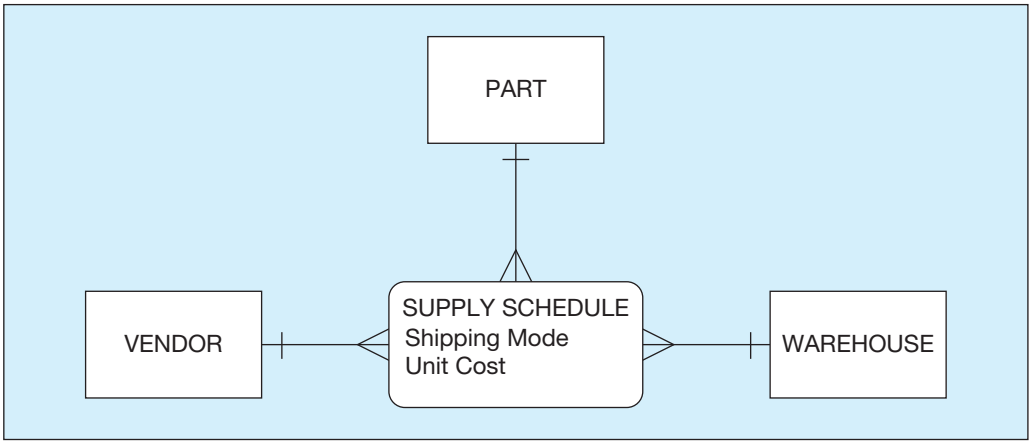


FIGURE 2-14 Ternary relationship as an associative entity

Don't be confused: A ternary relationship is not the same as three binary relationships. For example, Unit Cost is an attribute of the Supplies relationship in Figure 2-12c. Unit Cost cannot be properly associated with any one of the three possible binary relationships among the three entity types, such as that between PART and WAREHOUSE. Thus, for example, if we were told that vendor X can ship part C for a unit cost of $8, those data would be incomplete because they would not indicate to which warehouse the parts would be shipped.

As usual, the presence of an attribute on the relationship Supplies in Figure 2-12c suggests converting the relationship to an associative entity type. Figure 2-14 shows an alternative (and preferable) representation of the ternary relationship shown in Figure 2-12c. In Figure 2-14, the (associative) entity type SUPPLY SCHEDULE is used to replace the Supplies relationship from Figure 2-12c. Clearly, the entity type SUPPLY SCHEDULE is of independent interest to users. However, notice that an identifier has not yet been assigned to SUPPLY SCHEDULE. This is acceptable. If no identifier is assigned to an associative entity during E-R modeling, an identifier (or key) will be assigned during logical modeling (discussed in Chapter 4). This will be a composite identifier whose components will consist of the identifier for each of the participating entity types (in this example, PART, VENDOR, and WAREHOUSE) or a surrogate, non-intelligent identifier. Can you think of other attributes that might be associated with SUPPLY SCHEDULE?

As noted earlier, we do not label the lines from SUPPLY SCHEDULE to the three entities. This is because these lines do not represent binary relationships. To keep the same meaning as the ternary relationship of Figure 2-12c, we cannot break the Supplies relationship into three binary relationships, as we have already mentioned.

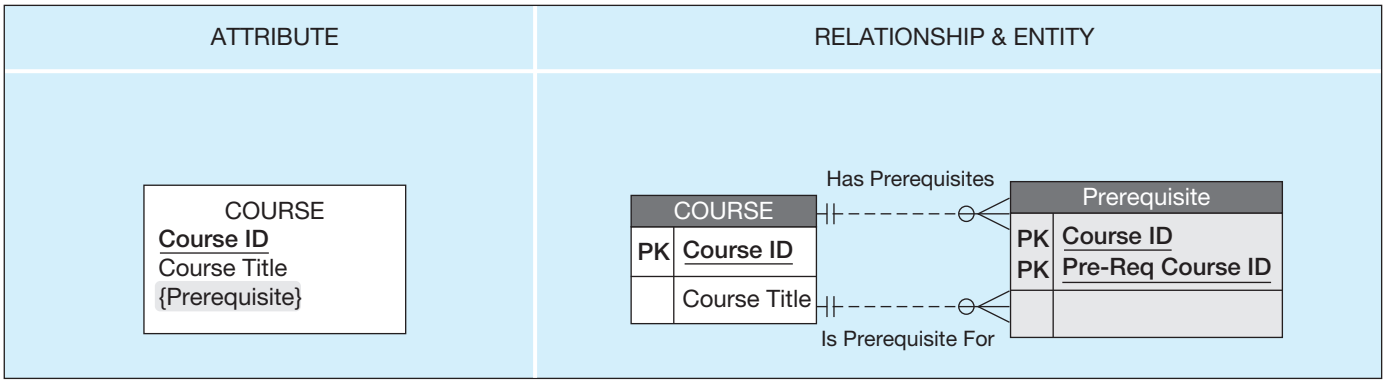
So, here is a guideline to follow: Convert all ternary (or higher) relationships to associative entities, as in this example. Song et al. (1995) show that participation constraints (described in a following section on cardinality constraints) cannot be accurately represented for a ternary relationship, given the notation with attributes on the relationship line. However, by converting to an associative entity, the constraints can be accurately represented. Also, many E-R diagram drawing tools, including most CASE tools, cannot represent ternary relationships. So, although not semantically accurate, you must use these tools to represent the ternary or higher-order relationship with an associative entity and three binary relationships, which have a mandatory association with each of the three related entity types.

Attributes or Entity?

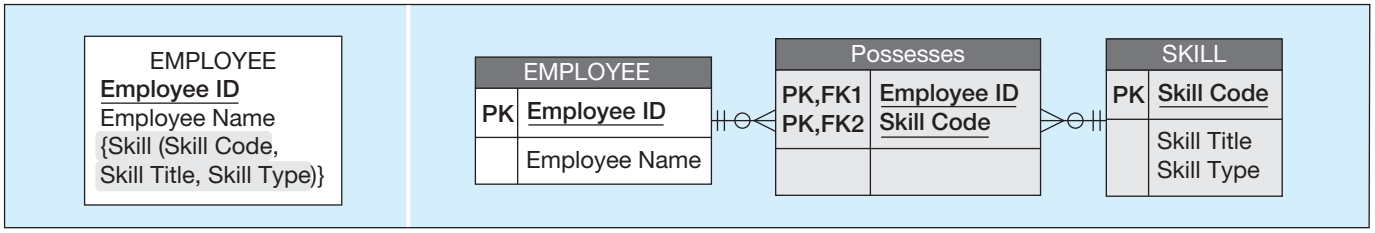
Sometimes you will wonder if you should represent data as an attribute or an entity; this is a common dilemma. Figure 2-15 includes three examples of situations when an attribute could be represented via an entity type. We use this text's E-R notation in the left column and the notation from Microsoft Visio in the right column; it is

FIGURE 2-15 **Using relationships and entities to link related attributes**

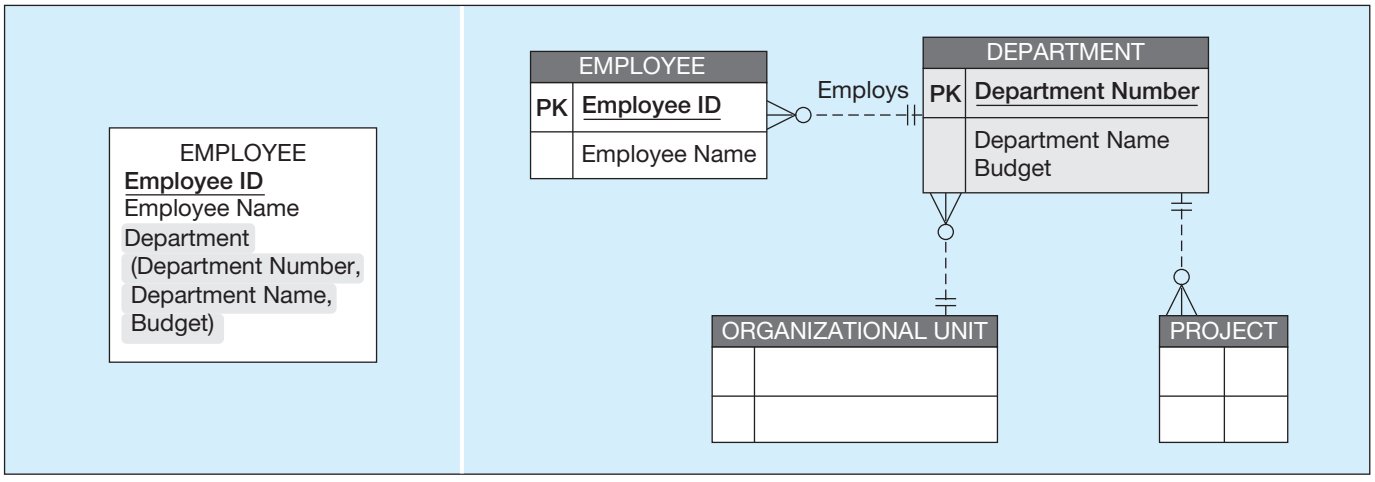
(a) Multivalued attribute versus relationships via bill-of-materials structure



(b) Composite, multivalued attribute versus relationship



(c) Composite attribute of data shared with other entity types



important that you learn how to read ERDs in several notations because you will encounter various styles in different publications and organizations. In Figure 2-15a, the potentially multiple prerequisites of a course (shown as a multivalued attribute in the Attribute cell) are also courses (and a course may be a prerequisite for many other courses). Thus, prerequisite could be viewed as a bill-of-materials structure (shown in the Relationship & Entity cell) between courses, not a multivalued attribute of COURSE. Representing prerequisites via a bill-of-materials structure also means that finding the prerequisites of a course and finding the courses for which a course is prerequisite both deal with relationships between entity types. When a prerequisite is a multivalued attribute of COURSE, finding the courses for which a course is a prerequisite means looking for a specific value for a prerequisite across all COURSE instances. As was shown in Figure 2-13a, such a situation could also be modeled as a

unary relationship among instances of the COURSE entity type. In Visio, this specific situation requires creating the equivalent of an associative entity (see the Relationship & Entity cell in Figure 2-15a; Visio does not use the rectangle with rounded corners symbol). By creating the associative entity, it is now easy to add characteristics to the relationship, such as a minimum grade required. Also note that Visio shows the identifier (in this case composite) with a PK stereotype symbol and boldface on the composite attribute names, signifying these are required attributes.

In Figure 2-15b, employees potentially have multiple skills (shown in the Attribute cell), but skill could be viewed instead as an entity type (shown in the Relationship & Entity cell as the equivalent of an associative entity) about which the organization wants to maintain data (the unique code to identify each skill, a descriptive title, and the type of skill, e.g., technical or managerial). An employee has skills, which are not viewed as attributes but rather as instances of a related entity type. In the cases of Figures 2-15a and 2-15b, representing the data as a multivalued attribute rather than via a relationship with another entity type may, in the view of some people, simplify the diagram. On the other hand, the right-hand drawings in these figures are closer to the way the database would be represented in a standard relational database management system, the most popular type of DBMS in use today. Although we are not concerned with implementation during conceptual data modeling, there is some logic for keeping the conceptual and logical data models similar. Further, as you will see in the next example, there are times when an attribute, whether simple, composite, or multivalued, should be in a separate entity.

So, when *should* an attribute be linked to an entity type via a relationship? The answer is when the attribute is the identifier or some other characteristic of an entity type in the data model and multiple entity instances need to share these same attributes. Figure 2-15c represents an example of this rule. In this example, EMPLOYEE has a composite attribute of Department. Because Department is a concept of the business and multiple employees will share the same department data, department data could be represented (nonredundantly) in a DEPARTMENT entity type, with attributes for the data about departments that all other related entity instances need to know. With this approach, not only can different employees share the storage of the same department data, but projects (which are assigned to a department) and organizational units (which are composed of departments) also can share the storage of this same department data.

Cardinality Constraints

There is one more important data modeling notation for representing common and important business rules. Suppose there are two entity types, A and B, that are connected by a relationship. A **cardinality constraint** specifies the number of instances of entity B that can (or must) be associated with each instance of entity A. For example, consider a video store that rents DVDs of movies. Because the store may stock more than one DVD for each movie, this is intuitively a one-to-many relationship, as shown in Figure 2-16a (technically, movie is a strong entity and DVD is a weak entity; thus, we represent these entities this way with a partial identifier for DVD). Yet it is also true that the store may not have any DVDs of a given movie in stock at a particular time (e.g., all copies may be checked out). We need a more precise notation to indicate the range of cardinalities for a relationship. This notation was introduced in Figure 2-2, which you may want to review at this time.

Cardinality constraint

A rule that specifies the number of instances of one entity that can (or must) be associated with each instance of another entity.

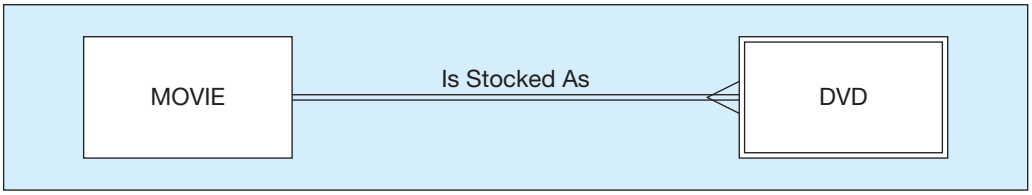
MINIMUM CARDINALITY The **minimum cardinality** of a relationship is the minimum number of instances of entity B that may be associated with each instance of entity A. In our DVD example, the minimum number of DVDs for a movie is zero. When the minimum number of participants is zero, we say that entity type B is an optional participant in the relationship. In this example, DVD (a weak entity type) is an optional participant in the Is Stocked As relationship. This fact is indicated by the symbol zero through the line near the DVD entity in Figure 2-16b.

Minimum cardinality

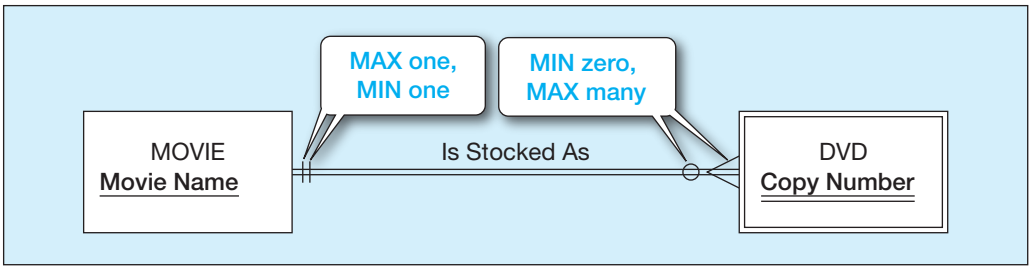
The minimum number of instances of one entity that may be associated with each instance of another entity.

FIGURE 2-16 Introducing cardinality constraints

(a) Basic relationship



(b) Relationship with cardinality constraints



Maximum cardinality

The maximum number of instances of one entity that may be associated with each instance of another entity.

MAXIMUM CARDINALITY The **maximum cardinality** of a relationship is the maximum number of instances of entity B that may be associated with each instance of entity A. In the video example, the maximum cardinality for the DVD entity type is "many"—that is, an unspecified number greater than one. This is indicated by the "crow's foot" symbol on the line next to the DVD entity symbol in Figure 2-16b. (You might find interesting the explanation of the origin of the crow's foot notation found in the Wikipedia entry about the E-R model; this entry also shows the wide variety of notation used to represent cardinality; see **http://en.wikipedia.org/wiki/Entity-relationship_model.**)

A relationship is, of course, bidirectional, so there is also cardinality notation next to the MOVIE entity. Notice that the minimum and maximum are both one (see Figure 2-16b). This is called a *mandatory one* cardinality. In other words, each DVD of a movie must be a copy of exactly one movie. In general, participation in a relationship may be optional or mandatory for the entities involved. If the minimum cardinality is zero, participation is optional; if the minimum cardinality is one, participation is mandatory.

In Figure 2-16b, some attributes have been added to each of the entity types. Notice that DVD is represented as a weak entity. This is because a DVD cannot exist unless the owner movie also exists. The identifier of MOVIE is Movie Name. DVD does not have a unique identifier. However, Copy Number is a *partial* identifier, which together with Movie Name would uniquely identify an instance of DVD.

Some Examples of Relationships and Their Cardinalities

Examples of three relationships that show all possible combinations of minimum and maximum cardinalities appear in Figure 2-17. Each example states the business rule for each cardinality constraint and shows the associated E-R notation. Each example also shows some relationship instances to clarify the nature of the relationship. You should study each of these examples carefully. Following are the business rules for each of the examples in Figure 2-17:

1. ***PATIENT Has Recorded PATIENT HISTORY (Figure 2-17a)*** Each patient has one or more patient histories. (A PATIENT cannot exist unless there is an initial instance of PATIENT HISTORY.) Each instance of PATIENT HISTORY "belongs to" exactly one PATIENT.
2. ***EMPLOYEE Is Assigned To PROJECT (Figure 2-17b)*** Each PROJECT has at least one EMPLOYEE assigned to it. (Some projects have more than one.) Each EMPLOYEE may or (optionally) may not be assigned to any existing PROJECT (e.g., employee Pete) or may be assigned to one or more PROJECTs.
3. ***PERSON Is Married To PERSON (Figure 2-17c)*** This is an optional zero or one cardinality in both directions because a person may or may not be married at a given point in time.

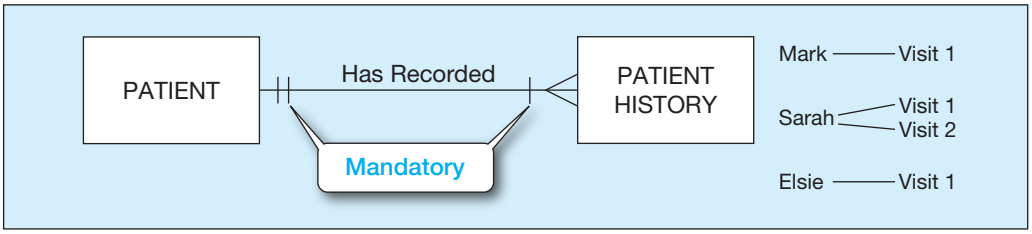
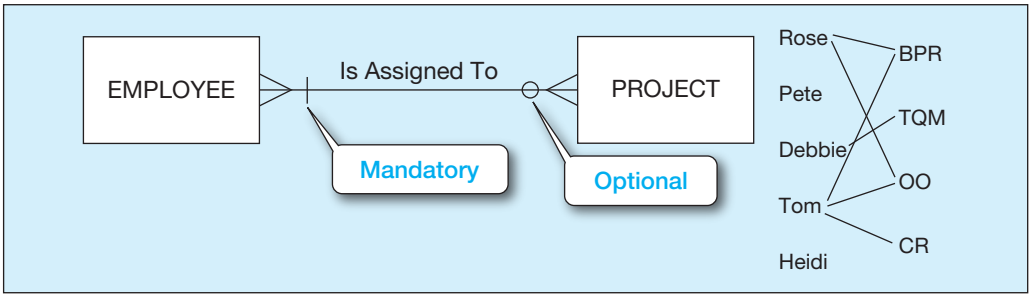
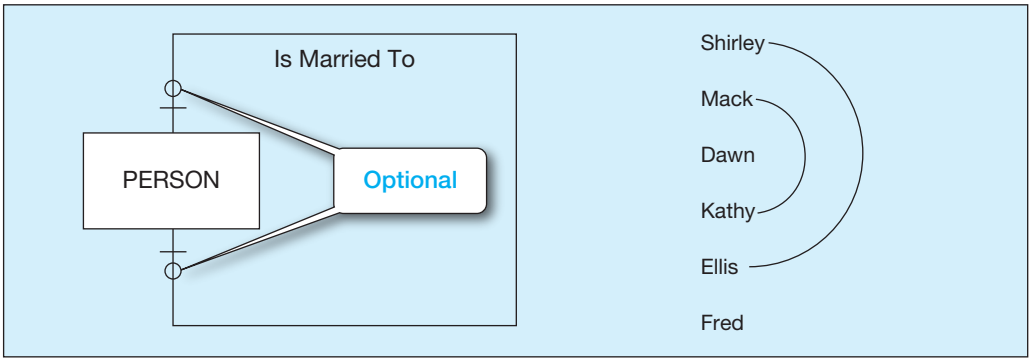


FIGURE 2-17 Examples of cardinality constraints

(a) Mandatory cardinalities



(b) One optional, one mandatory cardinality



(c) Optional cardinalities

It is possible for the maximum cardinality to be a fixed number, not an arbitrary "many" value. For example, suppose corporate policy states that an employee may work on at most five projects at the same time. We could show this business rule by placing a 5 above or below the crow's foot next to the PROJECT entity in Figure 2-17b.

A TERNARY RELATIONSHIP We showed the ternary relationship with the associative entity type SUPPLY SCHEDULE in Figure 2-14. Now let's add cardinality constraints to this diagram, based on the business rules for this situation. The E-R diagram, with the relevant business rules, is shown in Figure 2-18. Notice that PART and WAREHOUSE must relate to some SUPPLY SCHEDULE instance, and a VENDOR optionally may

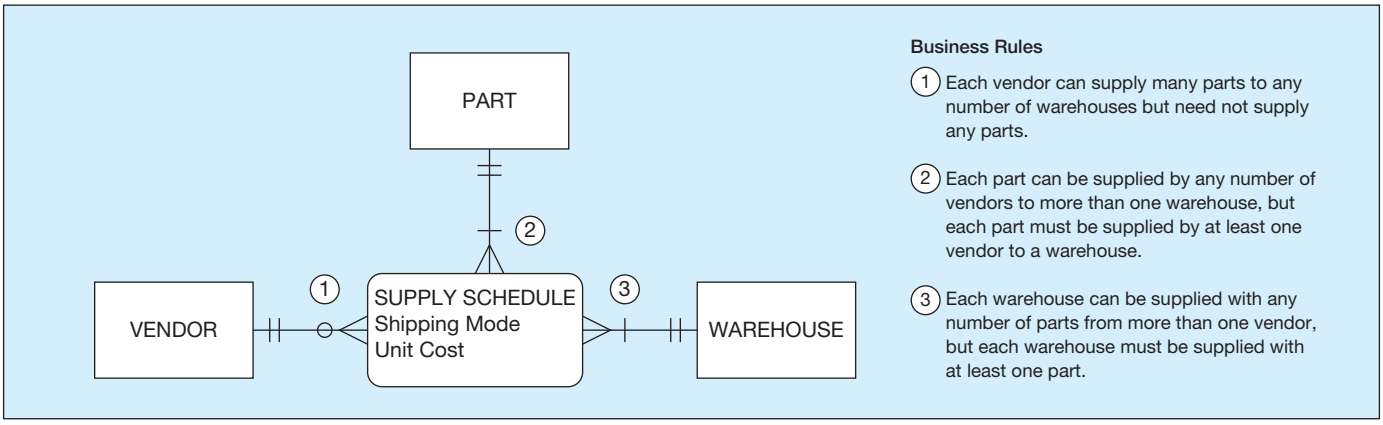


FIGURE 2-18 Cardinality constraints in a ternary relationship

not participate. The cardinality at each of the participating entities is a mandatory one because each SUPPLY SCHEDULE instance must be related to exactly one instance of each of these participating entity types. (Remember, SUPPLY SCHEDULE is an associative entity.)

As noted earlier, a ternary relationship is not equivalent to three binary relationships. Unfortunately, you are not able to draw ternary relationships with many CASE tools; instead, you are forced to represent ternary relationships as three binaries (i.e., an associative entity with three binary relationships). If you are forced to draw three binary relationships, then do not draw the binary relationships with names and be sure that the cardinality next to the three strong entities is a mandatory one.

Modeling Time-Dependent Data

Database contents vary over time. With renewed interest today in traceability and reconstruction of a historical picture of the organization for various regulatory requirements, such as HIPAA and Sarbanes-Oxley, and for business intelligence and other analytical purposes, the need to include a time series of data has become essential. For example, in a database that contains product information, the unit price for each product may be changed as material and labor costs and market conditions change. If only the current price is required, Price can be modeled as a single-valued attribute. However, for accounting, billing, financial reporting, and other purposes, we are likely to need to preserve a history of the prices and the time period during which each was in effect. As Figure 2-19 shows, we can conceptualize this requirement as a series of prices and the effective date for each price. This results in the (composite) multivalued attribute named Price History, with components Price and Effective Date. An important characteristic of such a composite, multivalued attribute is that the component attributes go together. Thus, in Figure 2-19, each Price is paired with the corresponding Effective Date.

Time stamp

A time value that is associated with a data value, often indicating when some event occurred that affected the data value.

In Figure 2-19, each value of the attribute Price is time stamped with its effective date. A **time stamp** is simply a time value, such as date and time, that is associated with a data value. A time stamp may be associated with any data value that changes over time when we need to maintain a history of those data values. Time stamps may be recorded to indicate the time the value was entered (transaction time), the time the value becomes valid or stops being valid, or the time when critical actions were performed, such as updates, corrections, or audits. This situation is similar to the employee skill diagrams in Figure 2-15b; thus, an alternative, not shown in Figure 2-19, is to make Price History a separate entity type, as was done with Skill using Microsoft Visio.

The use of simple time stamping (as in the preceding example) is often adequate for modeling time-dependent data. However, time can introduce subtler complexities to data modeling. For example, consider again Figure 2-17c. This figure is drawn for a given point in time, not to show history. If, on the other hand, we needed to record the full history of marriages for individuals, the Is Married To relationship would be an optional many-to-many relationship. Further, we might want to know the beginning and ending date (optional) of each marriage; these dates would be, similar to the bill-of-materials structure in Figure 2-13c, attributes of the relationship or associative entity.

Financial and other compliance regulations, such as Sarbanes-Oxley and Basel II, require that a database maintain history rather than just current status of critical data. In addition, some data modelers will argue that a data model should always be able to

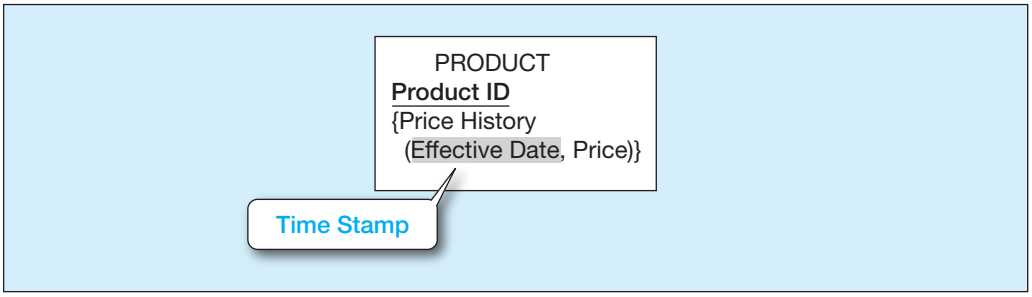


FIGURE 2-19 Simple example of time stamping

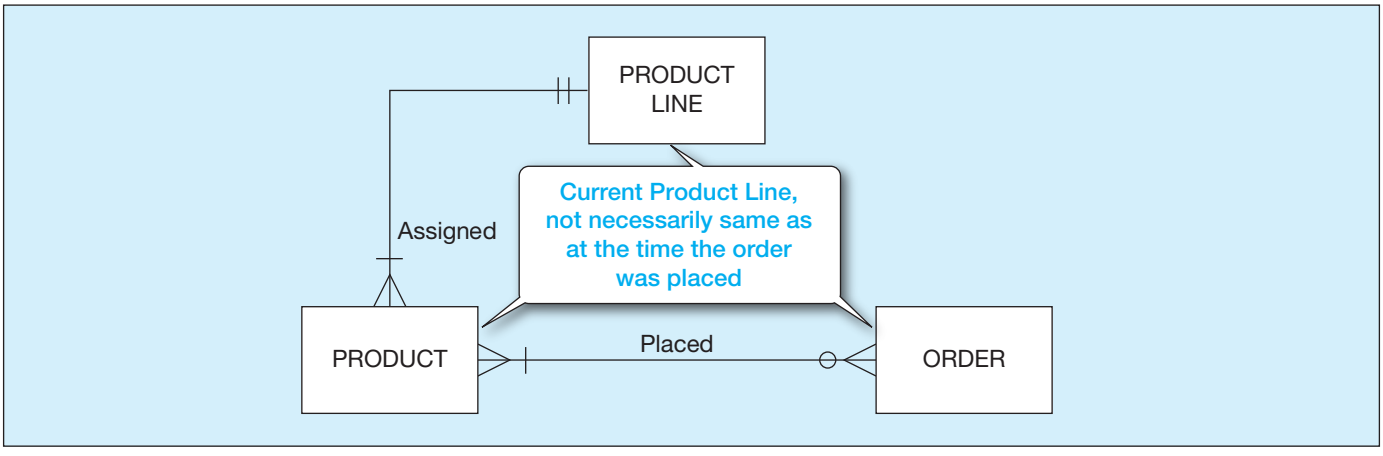
represent history, even if today's users say they need only current values. These factors suggest that all relationships should be modeled as many-to-many (which is often done in a purchased data model). Thus, for most databases, this will necessitate forming an associative entity along every relationship. There are two obvious negatives to this approach. First, many additional (associative) entities are created, thus cluttering ERDs. Second, a many-to-many (*M:N*) relationship is less restrictive than a one-to-many (1:*M*). So, if initially you want to enforce only one associated entity instance for some entity (i.e., the "one" side of the relationships), this cannot be enforced by the data model with an *M:N* relationship. It would seem likely that some relationships would never be *M:N*; for example, would a 1:*M* relationship between customer and order ever become *M:N* (but, of course, maybe someday our organization would sell items that would allow and often have joint purchasing, like vehicles or houses)? The conclusion is that if history or a time series of values might ever be desired or required by regulation, you should consider using an *M:N* relationship.

An even more subtle situation of the effect of time on data modeling is illustrated in Figure 2-20a, which represents a portion of an ERD for Pine Valley Furniture Company. Each product is assigned (i.e., current assignment) to a product line (or related group of products). Customer orders are processed throughout the year, and monthly summaries are reported by product line and by product within product line.

Suppose that in the middle of the year, due to a reorganization of the sales function, some products are reassigned to different product lines. The model shown in Figure 2-20a is not designed to track the reassignment of a product to a new product line. Thus, all sales reports will show cumulative sales for a product based on its current product line rather than the one at the time of the sale. For example, a product may have total year-to-date sales of $50,000 and be associated with product line B, yet $40,000 of

FIGURE 2-20 **Example of time in Pine Valley Furniture product database**

(a) E-R diagram not recognizing product reassignment



(b) E-R diagram recognizing product reassignment

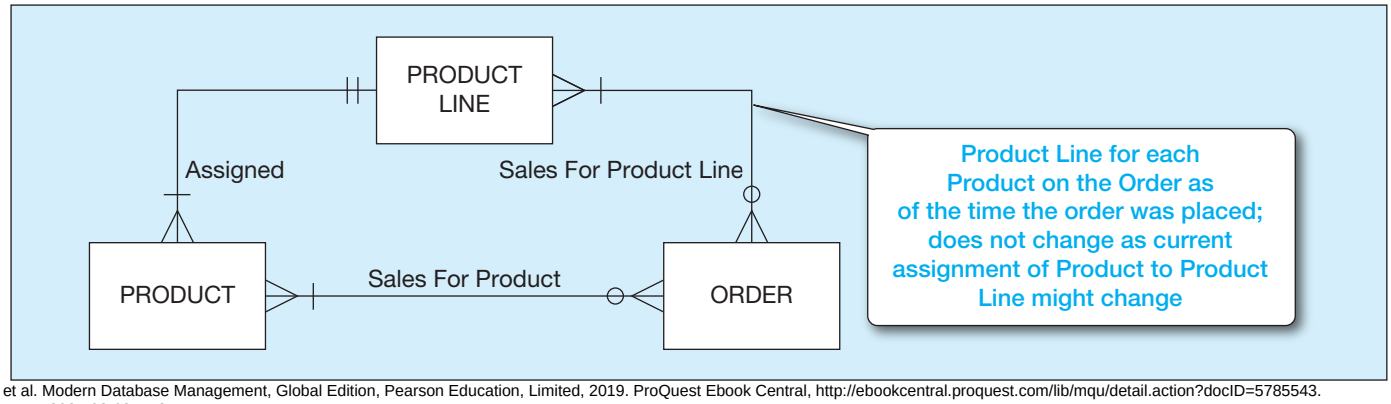
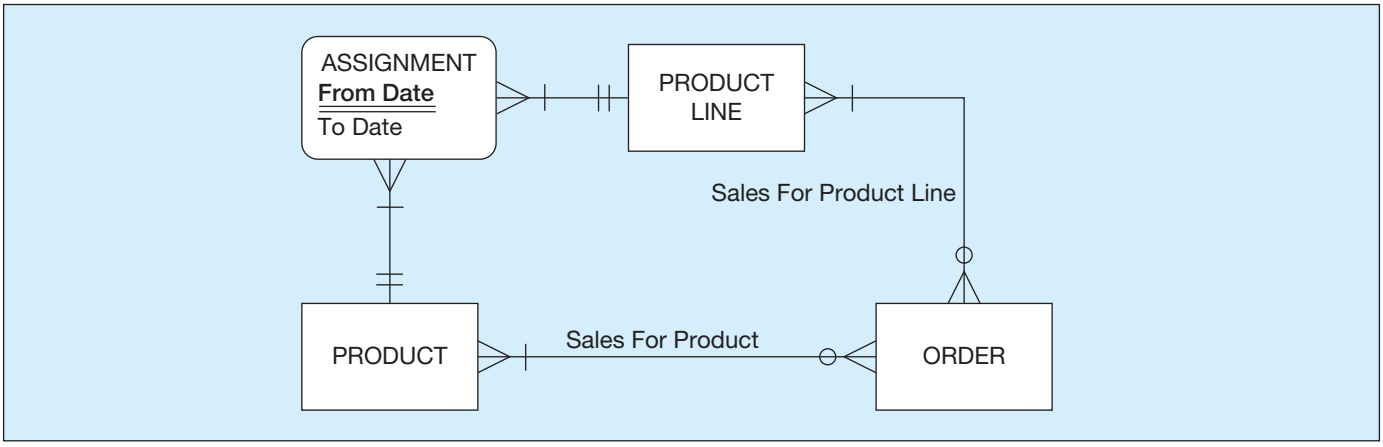


FIGURE 2-20 (*continued*)

(c) E-R diagram with associative entity for product assignment to product line over time



those sales may have occurred while the product was assigned to product line A. This fact will be lost using the model in Figure 2-20a. The simple design change shown in Figure 2-20b will correctly recognize product reassignments. A new relationship, called Sales For Product Line, has been added between ORDER and PRODUCT LINE. As customer orders are processed, they are credited to both the correct product (via Sales For Product) and the correct product line (via Sales For Product Line) as of the time of the sale. The approach of Figure 2-20b is similar to what is done in a data warehouse to retain historical records of the precise situation at any point in time. (We will return to dealing with the time dimension in Chapter 9.)

Another aspect of modeling time is recognizing that although the requirements of the organization today may be to record only the current situation, the design of the database may need to change if the organization ever decides to keep history. In Figure 2-20b, we know the current product line for a product and the product line for the product each time it is ordered. But what if the product were ever reassigned to a product line during a period of zero sales for the product? Based on this data model in Figure 2-20b, we would not know of these other product line assignments. A common solution to this need for greater flexibility in the data model is to consider whether a one-to-many relationship, such as Assigned, should become a many-to-many relationship. Further, to allow for attributes on this new relationship, this relationship should actually be an associative entity. Figure 2-20c shows this alternative data model with the ASSIGNMENT associative entity for the Assigned relationship. The advantage of the alternative is that we now will not miss recording any product line assignment, and we can record information about the assignment (such as the from and to effective dates of the assignment); the disadvantage is that the data model no longer has the restriction that a product may be assigned to only one product line at a time.

We have discussed the problem of time-dependent data with managers in several organizations who are considered leaders in the use of data modeling and database management. Before the recent wave of financial reporting disclosure regulations, these discussions revealed that data models for operational databases were generally inadequate for handing time-dependent data and that organizations often ignored this problem and hoped that the resulting inaccuracies balanced out. However, with these new regulations, you need to be alert to the complexities posed by time-dependent data as you develop data models in your organization. For a thorough explanation of time as a dimension of data modeling, see a series of articles by T. Johnson and R. Weis beginning in May 2007 in *DM Review* (now *Information Management*; see References at the end of this chapter).

Modeling Multiple Relationships Between Entity Types

There may be more than one relationship between the same entity types in a given organization. Two examples are shown in Figure 2-21. Figure 2-21a shows two relationships

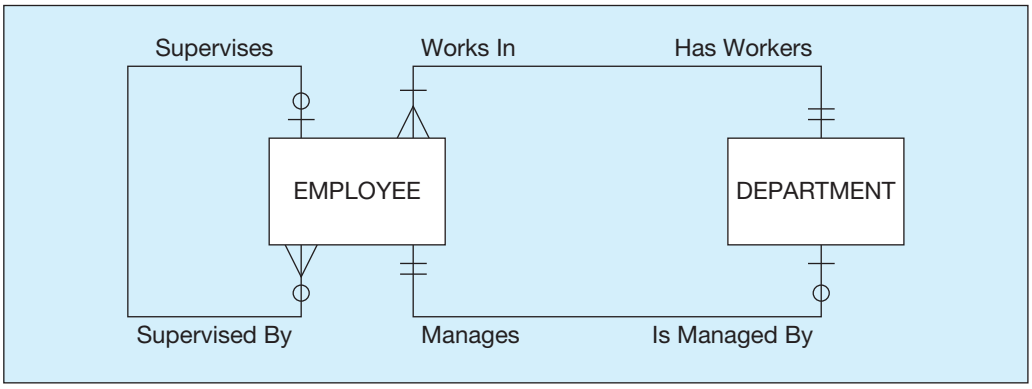
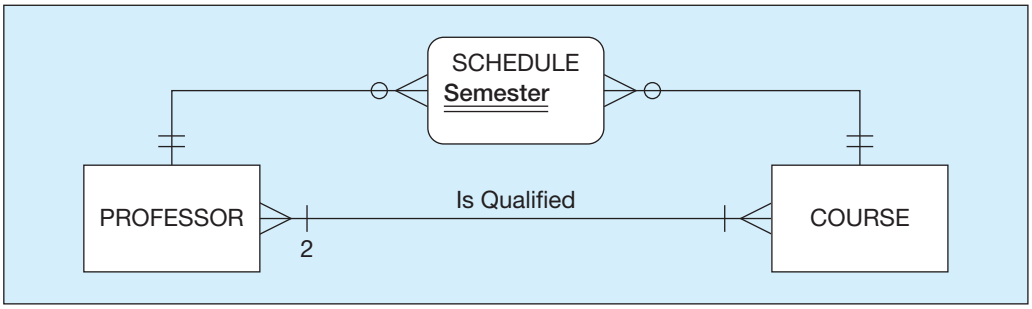


FIGURE 2-21 Examples of multiple relationships

(a) Employees and departments



(b) Professors and courses (fixed lower limit constraint)

between the entity types EMPLOYEE and DEPARTMENT. In this figure, we use the notation with names for the relationship in each direction; this notation makes explicit what the cardinality is for each direction of the relationship (which becomes important for clarifying the meaning of the unary relationship on EMPLOYEE). One relationship associates employees with the department in which they work. This relationship is one-to-many in the Has Workers direction and is mandatory in both directions. That is, a department must have at least one employee who works there (perhaps the department manager), and each employee must be assigned to exactly one department. (Note: These are specific business rules we assume for this illustration. It is crucial when you develop an E-R diagram for a particular situation that you understand the business rules that apply for that setting. For example, if EMPLOYEE were to include retirees, then each employee may not be currently assigned to exactly one department; further, the E-R model in Figure 2-21a assumes that the organization needs to remember in which DEPARTMENT each EMPLOYEE currently works rather than remembering the history of department assignments. Again, the structure of the data model reflects the information the organization needs to remember.)

The second relationship between EMPLOYEE and DEPARTMENT associates each department with the employee who manages that department. The relationship from DEPARTMENT to EMPLOYEE (called Is Managed By in that direction) is a mandatory one, indicating that a department must have exactly one manager. From EMPLOYEE to DEPARTMENT, the relationship (Manages) is optional because a given employee either is or is not a department manager.

Figure 2-21a also shows the unary relationship that associates each employee with his or her supervisor and vice versa. This relationship records the business rule that each employee may have exactly one supervisor (Supervised By). Conversely, each employee may supervise any number of employees or may not be a supervisor.

The example in Figure 2-21b shows two relationships between the entity types PROFESSOR and COURSE. The relationship Is Qualified associates professors with the courses they are qualified to teach. A given course must have at a minimum two qualified instructors (an example of how to use a fixed value for a minimum or maximum cardinality). This might happen, for example, so that a course is never the "property" of one instructor. Conversely, each instructor must be qualified to teach at least one course (a reasonable expectation).