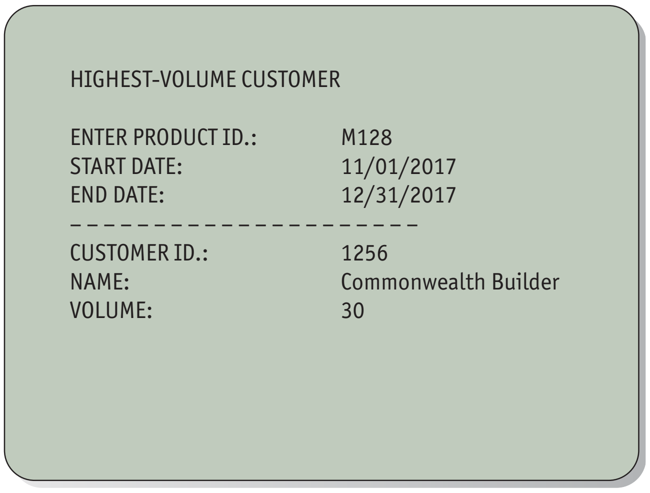
FIGURE 9-3
Simple example of logical data modeling
(a) Highest-volume customer query screen

(a)



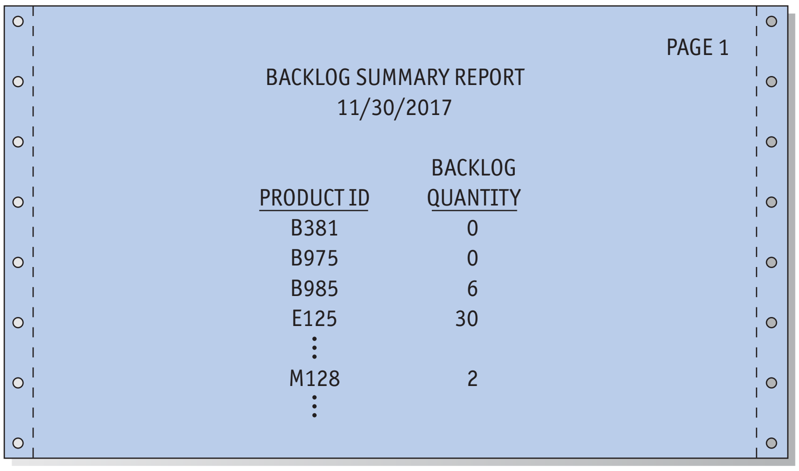
```
HIGHEST-VOLUME CUSTOMER

ENTER PRODUCT ID.:        M128
START DATE:               11/01/2017
END DATE:                 12/31/2017
- - - - - - - - - - - - - - - - - - - - -
CUSTOMER ID.:             1256
NAME:                     Commonwealth Builder
VOLUME:                   30
```

This inquiry screen shows the customer with the largest volume of total sales for a specified product during an indicated time period.

Relations:
CUSTOMER(Customer_ID,Name)
ORDER(Order_Number,Customer_ID,Order_Date)
PRODUCT(Product_ID)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)

(b) Backlog summary report

(b)



```
                                             PAGE 1
              BACKLOG SUMMARY REPORT
                    11/30/2017

                                BACKLOG
              PRODUCT ID       QUANTITY
                 B381              0
                 B975              0
                 B985              6
                 E125             30
                   ⋮
                 M128              2
                   ⋮
```

This report shows the unit volume of each product that has been ordered less that amount shipped through the specified date.

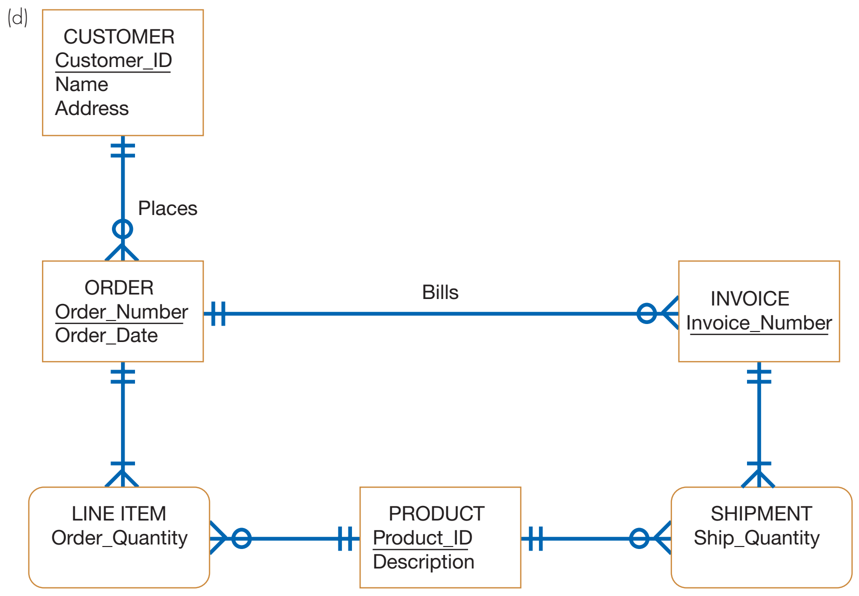
Relations:
PRODUCT(Product_ID)
LINE ITEM(Product_ID,Order_Number,Order_Quantity)
ORDER(Order_Number,Order_Date)
SHIPMENT(Product_ID,Invoice_Number,Ship_Quantity)
INVOICE(Invoice_Number,Invoice_Date,Order_Number)

(c) Integrated set of relations

(c) CUSTOMER(Customer_ID,Name)
PRODUCT(Product_ID)
INVOICE(Invoice_Number,Invoice_Date,Order_Number)
ORDER(Order_Number,Customer_ID,Order_Date)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)
SHIPMENT(Product_ID,Invoice_Number,Ship_Quantity)

FIGURE 9-3 (continued)
(d) Conceptual data model and transformed relations

(d)



Relations:
CUSTOMER(Customer_ID,Name,Address)
PRODUCT(Product_ID,Description)
ORDER(Order_Number,Customer_ID,Order_Date)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)
INVOICE(Invoice_Number,Order_Number)
SHIPMENT(Invoice_Number,Product_ID,Ship_Quantity)

(e) Final set of normalized relations

(e) CUSTOMER(Customer_ID,Name,Address)
PRODUCT(Product_ID,Description)
ORDER(Order_Number,Customer_ID,Order_Date)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)
INVOICE(Invoice_Number,Order_Number,Invoice_Date)
SHIPMENT(Invoice_Number,Product_ID,Ship_Quantity)

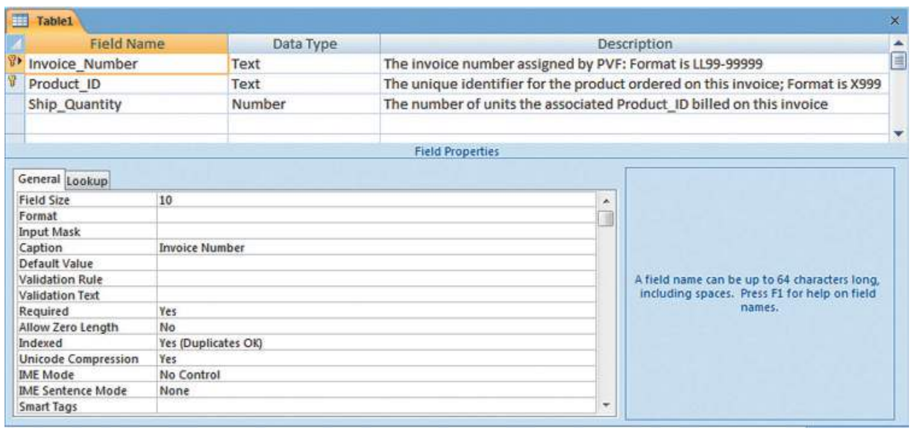


Table1

Field Name	Data Type	Description
Invoice_Number	Text	The invoice number assigned by PVF: Format is LL99-99999
Product_ID	Text	The unique identifier for the product ordered on this invoice; Format is X999
Ship_Quantity	Number	The number of units the associated Product_ID billed on this invoice

Field Properties

General Lookup

Field Size	10
Format	
Input Mask	
Caption	Invoice Number
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	No
Indexed	Yes (Duplicates OK)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

FIGURE 9-4
Definition of shipment table in Microsoft Access
(*Source:* Microsoft Corporation.)

- The Invoice Number field is required because it is part of the primary key for the SHIPMENT table (the value that makes every row of the SHIPMENT table unique is a combination of Invoice Number and Product ID).
- An index is defined for the Invoice Number field, but because there may be several rows in the SHIPMENT table for the same invoice (different products on the same invoice), duplicate index values are allowed (so Invoice Number is what we will call a secondary key).

EMPLOYEE1

Emp_ID	Name	Dept	Salary
100	Margaret Simpson	Marketing	75,000
140	Allen Beeton	Accounting	95,000
110	Chris Lucero	Info Systems	90,000
190	Lorenzo Davis	Finance	90,000
150	Susan Martin	Marketing	62,000

FIGURE 9-5
EMPLOYEE1 relation with sample data

Many other physical database design decisions were made for the SHIPMENT table, but they are not apparent on the display in Figure 9-4. Further, this table is only one table in the Pine Valley Furniture Company Order Entry database, and other tables and structures for this database are not illustrated in this figure.

The Relational Database Model

Many different database models are in use and are the bases for database technologies. Although hierarchical and network models have been popular in the past, these are not used very often today for new information systems. Object-oriented database models are emerging but are still not common. The vast majority of information systems today use the relational database model. The **relational database model** (Codd, 1970; Date, 2012; Elmasri and Navathe, 2015; Umanath and Scamell, 2014) represents data in the form of related tables, or relations. A **relation** is a named, two-dimensional table of data. Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows. Each column in a relation corresponds to an attribute of that relation. Each row of a relation corresponds to a record that contains data values for an entity.

Relational database model
Data represented as a set of related tables or relations.

Relation
A named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.

Figure 9-5 shows an example of a relation named EMPLOYEE1. This relation contains the following attributes describing employees: Emp_ID, Name, Dept, and Salary. This table has five sample rows, corresponding to five employees.

You can express the structure of a relation with a shorthand notation in which the name of the relation is followed (in parentheses) by the names of the attributes in the relation. The identifier attribute (called the *primary key of the relation*) is underlined. For example, you would express EMPLOYEE1 as follows:

EMPLOYEE1(Emp_ID,Name,Dept,Salary)

Not all tables are relations. Relations have several properties that distinguish them from nonrelational tables:

1. Entries in cells are simple. An entry at the intersection of each row and column has a single value.
2. Entries in a given column are from the same set of values.
3. Each row is unique. Uniqueness is guaranteed because the relation has a non-empty primary key value.
4. The sequence of columns can be interchanged without changing the meaning or use of the relation.
5. The rows may be interchanged or stored in any sequences.

Well-Structured Relations

What constitutes a **well-structured relation** (also known as a table)? Intuitively, a well-structured relation contains a minimum amount of redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.

Well-structured relation
A relation that contains a minimum amount of redundancy and that allows users to insert, modify, and delete the rows without error or inconsistencies; also known as a table.

EMPLOYEE2

Emp_ID	Name	Dept	Salary	Course	Date_Completed
100	Margaret Simpson	Marketing	42,000	SPSS	6/19/2017
100	Margaret Simpson	Marketing	42,000	Surveys	10/7/2017
140	Alan Beeton	Accounting	39,000	Tax Acc	12/8/2017
110	Chris Lucero	Info Systems	41,500	SPSS	1/22/2017
110	Chris Lucero	Info Systems	41,500	C++	4/22/2017
190	Lorenzo Davis	Finance	38,000	Investments	5/7/2017
150	Susan Martin	Marketing	38,500	SPSS	6/19/2017
150	Susan Martin	Marketing	38,500	TQM	8/12/2017

FIGURE 9-6
Relation with redundancy

EMP COURSE

Emp_ID	Course	Date_ Completed
100	SPSS	6/19/2017
100	Surveys	10/7/2017
140	Tax Acc	12/8/2017
110	SPSS	1/22/2017
110	C++	4/22/2017
190	Investments	5/7/2017
150	SPSS	6/19/2017
150	TQM	8/12/2017

FIGURE 9-7
EMP COURSE relation

EMPLOYEE1 (Figure 9-5) is such a relation. Each row of the table contains data describing one employee, and any modification to an employee's data (such as a change in salary) is confined to one row of the table.

In contrast, EMPLOYEE2 (Figure 9-6) contains data about employees and the courses they have completed. Each row in this table is unique for the combination of Emp_ID and Course, which becomes the primary key for the table. This is not a well-structured relation, however. If you examine the sample data in the table, you notice a considerable amount of redundancy. For example, the Emp_ID, Name, Dept, and Salary values appear in two separate rows for employees 100, 110, and 150. Consequently, if the salary for employee 100 changes, we must record this fact in two rows (or more, for some employees).

The problem with this relation is that it contains data about two entities: EMPLOYEE and COURSE. You will learn to use principles of normalization to divide EMPLOYEE2 into two relations. One of the resulting relations is EMPLOYEE1 (Figure 9-5). The other we will call EMP COURSE, which appears with sample data in Figure 9-7. The primary key of this relation is the combination of Emp_ID and Course (we emphasize this by underlining the column names for these attributes).

NORMALIZATION

Normalization
The process of converting complex data structures into simple, stable data structures.

We have presented an intuitive discussion of well-structured relations; however, we need rules and a process for designing them. **Normalization** is a process for converting complex data structures into simple, stable data structures (Date, 2012).

For example, we used the principles of normalization to convert the EMPLOYEE2 table with its redundancy to EMPLOYEE1 (Figure 9-5) and EMP COURSE (Figure 9-7).

Rules of Normalization

Normalization is based on well-accepted principles and rules. There are many normalization rules, more than can be covered in this text (see Hoffer et al. [2011], for a more complete coverage). Besides the five properties of relations outlined previously, there are two other frequently used rules:

1. *Second normal form (2NF).* Each nonprimary key attribute is identified by the whole key (what we call full functional dependency). For example, in Figure 9-7, both Emp_ID and Course identify a value of Date_Completed because the same Emp_ID can be associated with more than one Date_Completed and the same for Course.
2. *Third normal form (3NF).* Nonprimary key attributes do not depend on each other (what we call no transitive dependencies). For example, in Figure 9-5, Name, Dept, and Salary cannot be guaranteed to be unique for one another.

The result of normalization is that every nonprimary key attribute depends upon the whole primary key and nothing but the primary key. We discuss second and third normal form in more detail next.

Functional Dependence and Primary Keys

Normalization is based on the analysis of functional dependence. A **functional dependency** is a particular relationship between two attributes. In a given relation, attribute B is functionally dependent on attribute A if, for every valid value of A, that value of A uniquely determines the value of B (Date, 2012; Hoffer et al., 2016). The functional dependence of B on A is represented by an arrow, as follows: A → B (e.g., Emp_ID → Name in the relation of Figure 9-5). Functional dependence does not imply mathematical dependence—that the value of one attribute may be computed from the value of another attribute; rather, functional dependence of B on A means that there can be only one value of B for each value of A. Thus, a given Emp_ID value can have only one Name value associated with it; the value of Name, however, cannot be derived from the value of Emp_ID. Other examples of functional dependencies from Figure 9-3b are in ORDER, Order_Number, Order_Date, and in INVOICE, Invoice_Number, Invoice_Date, and Order_Number.

Functional dependency
A constraint between two attributes in which the value of one attribute is determined by the value of another attribute.

An attribute may be functionally dependent on two (or more) attributes rather than on a single attribute. For example, consider the relation EMP COURSE (Emp_ID,Course,Date_Completed) shown in Figure 9-7. We represent the functional dependency in this relation as follows:

Emp_ID,Course → Date_Completed (this is sometimes shown as Emp_ID + Course → Date_Completed). In this case, Date_Completed cannot be determined by either Emp_ID or Course alone because Date_Completed is a characteristic of an employee taking a course.

You should be aware that the instances (or sample data) in a relation do not prove that a functional dependency exists. Only knowledge of the problem domain, obtained from a thorough requirements analysis, is a reliable method for identifying a functional dependency. However, you can use sample data to demonstrate that a functional dependency does not exist between two or more attributes. For example, consider the sample data in the relation EXAMPLE(A,B,C,D), shown in Figure 9-8. The sample data in this relation prove that attribute B is not functionally dependent on attribute A because A does not uniquely determine B (two rows with the same value of A have different values of B).

EXAMPLE

A	B	C	D
X	U	X	Y
Y	X	Z	X
Z	Y	Y	Y
Y	Z	W	Z

FIGURE 9-8
EXAMPLE relation

Second Normal Form

Second normal form (2NF)
A relation is in second normal form if every nonprimary key attribute is functionally dependent on the whole primary key.

A relation is in **second normal form (2NF)** if every nonprimary key attribute is functionally dependent on the whole primary key. Thus, no nonprimary key attribute is functionally dependent on part, but not all, of the primary key. Second normal form is satisfied if any one of the following conditions apply:

1. The primary key consists of only one attribute (such as the attribute Emp_ID in relation EMPLOYEE1).
2. No nonprimary key attributes exist in the relation.
3. Every nonprimary key attribute is functionally dependent on the full set of primary key attributes.

EMPLOYEE2 (Figure 9-6) is an example of a relation that is not in second normal form. The shorthand notation for this relation is

EMPLOYEE2(<u>Emp_ID</u>,Name,Dept,Salary,Course,Date_Completed)

The functional dependencies in this relation are the following:

Emp_ID → Name,Dept,Salary
Emp_ID,Course → Date_Completed

The primary key for this relation is the composite key Emp_ID,Course. Therefore, the nonprimary key attributes Name, Dept, and Salary are functionally dependent on only Emp_ID but not on Course. EMPLOYEE2 has redundancy, which results in problems when the table is updated.

To convert a relation to second normal form, you decompose the relation into new relations using the attributes, called *determinants*, that determine other attributes; the determinants are the primary keys of these relations. EMPLOYEE2 is decomposed into the following two relations:

1. EMPLOYEE(<u>Emp_ID</u>,Name,Dept,Salary): This relation satisfies the first second normal form condition (sample data shown in Figure 9-5).
2. EMP COURSE(<u>Emp_ID</u>,Course,Date_Completed): This relation satisfies second normal form condition three (sample data appear in Figure 9-7).

Third Normal Form

Third normal form (3NF)
A relation is in second normal form and has no functional (transitive) dependencies between two (or more) nonprimary key attributes.

A relation is in **third normal form (3NF)** if it is in second normal form and there are no functional dependencies between two (or more) nonprimary key attributes (a functional dependency between nonprimary key attributes is also called a *transitive dependency*). For example, consider the relation SALES (Customer_ID, Customer_Name,Salesperson,Region) (sample data shown in Figure 9-9a).

The following functional dependencies exist in the SALES relation:

1. Customer_ID → Customer_Name,Salesperson,Region (Customer_ID is the primary key.)
2. Salesperson → Region (Each salesperson is assigned to a unique region.)

Notice that SALES is in second normal form because the primary key consists of a single attribute (Customer_ID). However, Region is functionally dependent on Salesperson, and Salesperson is functionally dependent on Customer_ID. As a result, there are data maintenance problems in SALES.

1. A new salesperson (Robinson) assigned to the North region cannot be entered until a customer has been assigned to that salesperson (because a value for Customer_ID must be provided to insert a row in the table).

FIGURE 9-9
Removing transitive dependencies
(a) Relation with transitive dependency

SALES

Customer_ID	Customer_Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

(b) Relation in 3NF

SALES1

Customer_ID	Customer_Name	Salesperson
8023	Anderson	Smith
9167	Bancroft	Hicks
7924	Hobbs	Smith
6837	Tucker	Hernandez
8596	Eckersley	Hicks
7018	Arnold	Faulb

SPERSON

Salesperson	Region
Smith	South
Hicks	West
Hernandez	East
Faulb	North

2. If customer number 6837 is deleted from the table, we lose the information that salesperson Hernandez is assigned to the East region.
3. If salesperson Smith is reassigned to the East region, several rows must be changed to reflect that fact (two rows are shown in Figure 9-9a).

These problems can be avoided by decomposing SALES into the two relations, based on the two determinants, shown in Figure 9-9b. These relations are the following:

SALES1 (Customer_ID,Customer_Name,Salesperson)
SPERSON (Salesperson,Region)

Note that Salesperson is the primary key in SPERSON. Salesperson is also a foreign key in SALES1. A **foreign key** is an attribute that appears as a nonprimary key attribute in one relation (such as SALES1) and as a primary key attribute (or part of a primary key) in another relation. You designate a foreign key by using a dashed underline.

A foreign key must satisfy **referential integrity**, which specifies that the value of an attribute in one relation depends on the value of the same attribute in another relation. Thus, in Figure 9-9b, the value of Salesperson in each row of table SALES1 is limited only to the current values of Salesperson in the SPERSON table. If some sales do not have to have a salesperson, then it is possible for the value of Salesperson to be null (i.e., have no value). Referential integrity is one of the most important principles of the relational model.

Foreign key
An attribute that appears as a nonprimary key attribute in one relation and as a primary key attribute (or part of a primary key) in another relation.

Referential integrity
A rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null (i.e., have no value).

TRANSFORMING E-R DIAGRAMS INTO RELATIONS

Normalization produces a set of well-structured relations that contains all of the data mentioned in system inputs and outputs developed in human interface design. Because these specific information requirements may not represent all future information needs, the E-R diagram you developed in conceptual data modeling is another source of insight into possible data requirements for a new application system. To compare the conceptual data model and the normalized relations developed so far, your E-R diagram must be transformed into relational notation, normalized, and then merged with the existing normalized relations.

Transforming an E-R diagram into normalized relations and then merging all the relations into one final, consolidated set of relations can be accomplished in four steps. These steps are summarized briefly here, and then steps 1, 2, and 4 are discussed in detail in the remainder of this chapter.

1. *Represent entities.* Each entity type in the E-R diagram becomes a relation. The identifier of the entity type becomes the primary key of the relation, and other attributes of the entity type become nonprimary key attributes of the relation.
2. *Represent relationships.* Each relationship in an E-R diagram must be represented in the relational database design. How we represent a relationship depends on its nature. For example, in some cases we represent a relationship by making the primary key of one relation a foreign key of another relation. In other cases, we create a separate relation to represent a relationship.
3. *Normalize the relations.* The relations created in steps 1 and 2 may have unnecessary redundancy. So we need to normalize these relations to make them well structured.
4. *Merge the relations.* So far in database design we have created various relations from both a bottom-up normalization of user views and from transforming one or more E-R diagrams into sets of relations. Across these different sets of relations, there may be redundant relations (two or more relations that describe the same entity type) that must be merged and renormalized to remove the redundancy.

Represent Entities

Each regular entity type in an E-R diagram is transformed into a relation. The identifier of the entity type becomes the primary key of the corresponding relation. Each nonkey attribute of the entity type becomes a nonkey attribute of the relation. You should check to make sure that the primary key satisfies the following two properties:

1. The value of the key must uniquely identify every row in the relation.
2. The key should be nonredundant; that is, no attribute in the key can be deleted without destroying its unique identification.

Some entities may have keys that include the primary keys of other entities. For example, an EMPLOYEE DEPENDENT may have a Name for each dependent, but to form the primary key for this entity, you must include the Employee_ID attribute from the associated EMPLOYEE entity. Such an entity whose primary key depends upon the primary key of another entity is called a *weak entity.*

Representation of an entity as a relation is straightforward. Figure 9-10a shows the CUSTOMER entity type for PVF. The corresponding CUSTOMER relation is represented as follows:

CUSTOMER(Customer_ID,Name,Address,City_State_ZIP,Discount)

In this notation, the entity type label is translated into a relation name. The identifier of the entity type is listed first and underlined. All nonkey attributes are listed after the primary key. This relation is shown as a table with sample data in Figure 9-10b.

Represent Relationships

The procedure for representing relationships depends on both the degree of the relationship—unary, binary, ternary—and the cardinalities of the relationship.