CHAPTER 2

# DATABASE DESIGN FUNDAMENTALS

## LEARNING OBJECTIVES

**Objectives**

- Understand the terms *entity*, *attribute*, and *relationship*
- Understand the terms *relation* and *relational database*
- Understand functional dependence and identify when one column is functionally dependent on another
- Understand the term *primary key* and identify primary keys in tables
- Design a database to satisfy a set of requirements
- Convert an unnormalized relation to first normal form
- Convert tables from first normal form to second normal form
- Convert tables from second normal form to third normal form
- Create an entity-relationship diagram to represent the design of a database

## INTRODUCTION

In Chapter 1, you reviewed the tables and columns in the Premiere Products, Henry Books, and Alexamara Marina Group databases that you will use to complete the rest of this text. The process of determining the particular tables and columns that will comprise a database is known as **database design**. In this chapter, you will learn a method for designing a database to satisfy a set of requirements. In the process, you will learn how to identify the tables and columns in the database. You also will learn how to identify the relationships between the tables.

This chapter begins by examining some important concepts related to databases. It also presents the design method using the set of requirements that Premiere Products identified to produce the appropriate database design. The chapter then examines the process of normalization, in which you identify and fix potential problems in database designs. Finally, you will learn a way of visually representing the design of a database.

## DATABASE CONCEPTS

Before learning how to design a database, you need to be familiar with some important database concepts related to relational databases, which are the types of databases you examined in Chapter 1 and that you will use throughout the rest of this text. The terms entity, attribute, and relationship are important to understand when designing a database; the concepts of functional dependence and primary keys are critical when learning about the database design process.

### Relational Databases

A **relational database** is a collection of tables like the ones you examined for Premiere Products in Chapter 1 and that also appear in Figure 2-1. Formally, these tables are called relations, and this is how this type of database gets its name.

REP

| REP_ NUM | LAST_ NAME | FIRST_ NAME | STREET | CITY | STATE | ZIP | COMMISSION | RATE |
|---|---|---|---|---|---|---|---|---|
| 20 | Kaiser | Valerie | 624 Randall | Grove | FL | 33321 | $20,542.50 | 0.05 |
| 35 | Hull | Richard | 532 Jackson | Sheldon | FL | 33553 | $39,216.00 | 0.07 |
| 65 | Perez | Juan | 1626 Taylor | Fillmore | FL | 33336 | $23,487.00 | 0.05 |

CUSTOMER

| CUSTOMER_ NUM | CUSTOMER_ NAME | STREET | CITY | STATE | ZIP | BALANCE | CREDIT_ LIMIT | REP_ NUM |
|---|---|---|---|---|---|---|---|---|
| 148 | Al's Appliance and Sport | 2837 Greenway | Fillmore | FL | 33336 | $6,550.00 | $7,500.00 | 20 |
| 282 | Brookings Direct | 3827 Devon | Grove | FL | 33321 | $431.50 | $10,000.00 | 35 |
| 356 | Ferguson's | 382 Wildwood | Northfield | FL | 33146 | $5,785.00 | $7,500.00 | 65 |
| 408 | The Everything Shop | 1828 Raven | Crystal | FL | 33503 | $5,285.25 | $5,000.00 | 35 |
| 462 | Bargains Galore | 3829 Central | Grove | FL | 33321 | $3,412.00 | $10,000.00 | 65 |
| 524 | Kline's | 838 Ridgeland | Fillmore | FL | 33336 | $12,762.00 | $15,000.00 | 20 |
| 608 | Johnson's Department Store | 372 Oxford | Sheldon | FL | 33553 | $2,106.00 | $10,000.00 | 65 |
| 687 | Lee's Sport and Appliance | 282 Evergreen | Altonville | FL | 32543 | $2,851.00 | $5,000.00 | 35 |
| 725 | Deerfield's Four Seasons | 282 Columbia | Sheldon | FL | 33553 | $248.00 | $7,500.00 | 35 |
| 842 | All Season | 28 Lakeview | Grove | FL | 33321 | $8,221.00 | $7,500.00 | 20 |

ORDERS

| ORDER_ NUM | ORDER_ DATE | CUSTOMER_ NUM |
|---|---|---|
| 21608 | 10/20/2010 | 148 |
| 21610 | 10/20/2010 | 356 |
| 21613 | 10/21/2010 | 408 |
| 21614 | 10/21/2010 | 282 |
| 21617 | 10/23/2010 | 608 |
| 21619 | 10/23/2010 | 148 |
| 21623 | 10/23/2010 | 608 |

ORDER_LINE

| ORDER_NUM | PART_NUM | NUM_ORDERED | QUOTED_PRICE |
|---|---|---|---|
| 21608 | AT94 | 11 | $21.95 |
| 21610 | DR93 | 1 | $495.00 |
| 21610 | DW11 | 1 | $399.99 |
| 21613 | KL62 | 4 | $329.95 |
| 21614 | KT03 | 2 | $595.00 |
| 21617 | BV06 | 2 | $794.95 |
| 21617 | CD52 | 4 | $150.00 |
| 21619 | DR93 | 1 | $495.00 |
| 21623 | KV29 | 2 | $1,290.00 |

PART

| PART_NUM | DESCRIPTION | ON_HAND | CLASS | WAREHOUSE | PRICE |
|---|---|---|---|---|---|
| AT94 | Iron | 50 | HW | 3 | $24.95 |
| BV06 | Home Gym | 45 | SG | 2 | $794.95 |
| CD52 | Microwave Oven | 32 | AP | 1 | $165.00 |
| DL71 | Cordless Drill | 21 | HW | 3 | $129.95 |
| DR93 | Gas Range | 8 | AP | 2 | $495.00 |
| DW11 | Washer | 12 | AP | 3 | $399.99 |
| FD21 | Stand Mixer | 22 | HW | 3 | $159.95 |
| KL62 | Dryer | 12 | AP | 1 | $349.95 |
| KT03 | Dishwasher | 8 | AP | 3 | $595.00 |
| KV29 | Treadmill | 9 | SG | 2 | $1,390.00 |

**FIGURE 2-1** Sample data for Premiere Products

**NOTE**

The names of columns and tables in this text follow a common naming convention in which column names use uppercase letters and replace spaces between words with underscores (_). For example, Premiere Products uses the column named LAST_NAME to store last names and the column named CREDIT_LIMIT to store credit limits.

## Entities, Attributes, and Relationships

There are some terms and concepts that are very important for you to know when working in the database environment. The terms entity, attribute, and relationship are fundamental when discussing databases. An **entity** is like a noun; it is a person, place, thing, or event. The entities of interest to Premiere Products, for example, are such things as customers, orders, and sales reps. The entities that are of interest to a school include students, faculty, and classes; a real estate agency is interested in clients, houses, and agents; and a used car dealer is interested in vehicles, customers, and manufacturers.

An **attribute** is a property of an entity. The term is used here exactly as it is used in everyday English. For the entity *person*, for example, the list of attributes might include such things as eye color and height. For Premiere Products, the attributes of interest for the entity *customer* are such things as name, address, city, and so on. For the entity *faculty* at a school, the attributes would be such things as faculty number, name, office number, phone, and so on. For the entity *vehicle* at a car dealership, the attributes are such things as the vehicle identification number, model, color, year, and so on.

A **relationship** is the association between entities. There is an association between customers and sales reps, for example, at Premiere Products. A sales rep is associated with all of his or her customers, and a customer is associated with his or her sales rep. Technically, you say that a sales rep is *related* to all of his or her customers, and a customer is *related* to his or her sales rep.

The relationship between sales reps and customers is an example of a **one-to-many relationship** because one sales rep is associated with many customers, but each customer is associated with only one sales rep. (In this type of relationship, the word *many* is used in a way that is different from everyday English; it might not always mean a large number. In this context, for example, the term *many* means that a sales rep might be associated with *any* number of customers. That is, one sales rep can be associated with zero, one, or more customers.)

How does a relational database handle entities, attributes of entities, and relationships between entities? Entities and attributes are fairly simple. Each entity has its own table. In the Premiere Products database, there is one table for sales reps, one table for customers, and so on. The attributes of an entity become the columns in the table. In the table for sales reps, for example, there is a column for the sales rep number, a column for the sales rep's first name, and so on.

What about relationships? At Premiere Products, there is a one-to-many relationship between sales reps and customers (each sales rep is related to the *many* customers that he or she represents, and each customer is related to the *one* sales rep who represents the customer). How is this relationship implemented in a relational database?

Consider Figure 2-1 again. If you want to determine the name of the sales rep who represents Brookings Direct (customer number 282), you would locate the row for Brookings Direct in the CUSTOMER table and determine that the value for REP_NUM is 35. Then you would look for the row in the REP table on which the REP_NUM is 35. The *one* rep with REP_NUM 35 is Richard Hull, who represents Brookings Direct.

On the other hand, if you want to determine the names of all the customers of the rep named Valerie Kaiser, you would locate the row for Valerie Kaiser in the REP table and determine that the value in the REP_NUM column is 20. Then you would look for all the rows in the CUSTOMER table on which the REP_NUM is 20. After identifying Valerie Kaiser's rep number, you find that the *many* customers she represents are numbered 148 (Al's Appliance and Sport), 524 (Kline's), and 842 (All Season).

You implement these relationships by having common columns in two or more tables. The REP_NUM column in the REP table and the REP_NUM column in the CUSTOMER table are used to implement the relationship between sales reps and customers. Given a sales rep, you can use these columns to determine all the customers that he or she represents; given a customer, you can use these columns to find the sales rep who represents the customer.

In this context, a relation is essentially a two-dimensional table. If you consider the tables shown in Figure 2-1, however, you can see that certain restrictions are placed on relations. Each column has a unique name, and entries within each column should "match" this column name. For example, if the column name is CREDIT_LIMIT, all entries in that column must be credit limits. Also, each row should be unique—when two rows are identical, the second row does not provide any new information. For maximum flexibility, the order of the columns and rows should be immaterial. Finally, the table's design should be as simple as possible by restricting each position to a single entry and by preventing multiple entries (also called **repeating groups**) in an individual location in the table. Figure 2-2 shows a table design that includes repeating groups.

ORDERS

| ORDER_NUM | ORDER_DATE | CUSTOMER_NUM | PART_NUM | NUM_ORDERED | QUOTED_PRICE |
|---|---|---|---|---:|---:|
| 21608 | 10/20/2010 | 148 | AT94 | 11 | $21.95 |
| 21610 | 10/20/2010 | 356 | DR93<br>DW11 | 1<br>1 | $495.00<br>$399.99 |
| 21613 | 10/21/2010 | 408 | KL62 | 4 | $329.95 |
| 21614 | 10/21/2010 | 282 | KT03 | 2 | $595.00 |
| 21617 | 10/23/2010 | 608 | BV06<br>CD52 | 2<br>4 | $12.95<br>$150.00 |
| 21619 | 10/23/2010 | 148 | DR93 | 1 | $495.00 |
| 21623 | 10/23/2010 | 608 | KV29 | 2 | $325.99 |

**FIGURE 2-2** Table with repeating groups

Figure 2-3 shows a better way to represent the same information shown in Figure 2-2. In Figure 2-3, every position in the table contains a single value.

ORDERS

| ORDER_ NUM | ORDER_ DATE | CUSTOMER_ NUM | PART_ NUM | NUM_ ORDERED | QUOTED_ PRICE |
|---|---|---|---|---|---|
| 21608 | 10/20/2010 | 148 | AT94 | 11 | $21.95 |
| 21610 | 10/20/2010 | 356 | DR93 | 1 | $495.00 |
| 21610 | 10/20/2010 | 356 | DW11 | 1 | $399.99 |
| 21613 | 10/21/2010 | 408 | KL62 | 4 | $329.95 |
| 21614 | 10/21/2010 | 282 | KT03 | 2 | $595.00 |
| 21617 | 10/23/2010 | 608 | BV06 | 2 | $12.95 |
| 21617 | 10/23/2010 | 608 | CD52 | 4 | $150.00 |
| 21619 | 10/23/2010 | 148 | DR93 | 1 | $495.00 |
| 21623 | 10/23/2010 | 608 | KV29 | 2 | $325.99 |

**FIGURE 2-3** ORDERS data without repeating groups

When you remove the repeating groups from Figure 2-2, all of the rows in Figure 2-3 are single-valued. This structure is formally called a relation. A **relation** is a two-dimensional table in which the entries in the table are single-valued (each location in the table contains a single entry), each column has a distinct name, all values in the column match this name, the order of the rows and columns is immaterial, and each row contains unique values. A relational database is a collection of relations.

**NOTE**

Rows in a table (relation) are also called **records** or **tuples**. Columns in a table (relation) are also called **fields** or attributes. This text uses the terms tables, columns, and rows unless the more formal terms of relation, attributes, and tuples are necessary for clarity.

There is a commonly accepted shorthand representation to show the tables and columns in a relational database: for each table, you write the name of the table and then within parentheses list all of the columns in the table. In this representation, each table appears on its own line. Using this method, you represent the Premiere Products database as follows:

```
REP (REP_NUM, LAST_NAME, FIRST_NAME, STREET,
     CITY, STATE, ZIP, COMMISSION, RATE)
CUSTOMER (CUSTOMER_NUM, CUSTOMER_NAME, STREET,
     CITY, STATE, ZIP, BALANCE, CREDIT_LIMIT,
     REP_NUM)
ORDERS (ORDER_NUM, ORDER_DATE, CUSTOMER_NUM)
ORDER_LINE (ORDER_NUM, PART_NUM, NUM_ORDERED,
     QUOTED_PRICE)
PART (PART_NUM, DESCRIPTION, ON_HAND, CLASS,
     WAREHOUSE, PRICE)
```

Notice that some tables contain columns with duplicate names. For example, the REP_NUM column appears in both the REP table *and* the CUSTOMER table. Suppose a situation existed wherein someone (or the DBMS) might confuse the two columns. For example, if you write REP_NUM, it is not clear which REP_NUM column you want to use. You need a mechanism for indicating the REP_NUM column to which you are referring. One common approach to solving this problem is to write both the table name and the column name, separated by a period. Thus, you would reference the REP_NUM column in the CUSTOMER table as CUSTOMER.REP_NUM, and the REP_NUM column in the REP table as REP.REP_NUM. Technically, when you reference columns in this format, you say that you **qualify** the names. It is *always* acceptable to qualify column names, even when there is no potential for confusion. If confusion might arise, however, it is *essential* to qualify column names.

## FUNCTIONAL DEPENDENCE

The concept of functional dependence is crucial to understanding the rest of the material in this chapter. Functional dependence is a formal name for what is basically a simple idea. To illustrate functional dependence, suppose the REP table for Premiere Products is structured as shown in Figure 2-4. The only difference between the REP table shown in Figure 2-4 and the one shown in Figure 2-1 is the addition of an extra column named PAY_CLASS.

REP

| REP_ NUM | LAST_ NAME | FIRST_ NAME | STREET | CITY | STATE | ZIP | COMMISSION | PAY_ CLASS | RATE |
|---|---|---|---|---|---|---|---|---|---|
| 20 | Kaiser | Valerie | 624 Randall | Grove | FL | 33321 | $20,542.50 | 1 | 0.05 |
| 35 | Hull | Richard | 532 Jackson | Sheldon | FL | 33553 | $39,216.00 | 2 | 0.07 |
| 65 | Perez | Juan | 1626 Taylor | Fillmore | FL | 33336 | $23,487.00 | 1 | 0.05 |

**FIGURE 2-4** REP table with a PAY_CLASS column

Suppose one of the policies at Premiere Products is that all sales reps in any given pay class earn their commissions at the same rate. To describe this situation, you could say that a sales rep's pay class *determines* his or her commission rate. Alternatively, you could say that a sales rep's commission rate *depends on* his or her pay class. This phrasing uses the words *determines* and *depends on* in the same way that you describe functional dependency. If you wanted to be formal, you would precede either expression with the word *functionally*. For example, you might say, "A sales rep's pay class *functionally determines* his or her commission rate," and "A sales rep's commission rate *functionally depends on* his or her pay class." You can also define functional dependency by saying that when you know a sales rep's pay class, you can determine his or her commission rate.

In a relational database, column B is **functionally dependent** on another column (or a collection of columns), A, if at any point in time a value for A determines a single value for B. You can think of this as follows: when you are given a value for A, do you know that you can find a single value for B? If so, B is functionally dependent on A (often written as A → B). If B is functionally dependent on A, you also can say that A **functionally determines** B.

At Premiere Products, is the LAST_NAME column in the REP table functionally dependent on the REP_NUM column? Yes, it is. If you are given a value for REP_NUM, such as 20, there is a *single* LAST_NAME, Kaiser, associated with it. This is represented as:

```
REP_NUM → LAST_NAME
```

**Q & A**

**Question:** In the CUSTOMER table, is CUSTOMER_NAME functionally dependent on REP_NUM?
**Answer:** No. Given the REP_NUM 20, for example, you would not be able to find a single customer name, because 20 appears on more than one row in the table.

**Q & A**

**Question:** In the ORDER_LINE table, is NUM_ORDERED functionally dependent on ORDER_NUM?
**Answer:** No. An ORDER_NUM might be associated with several items in an order, so having just an ORDER_NUM does not provide enough information.

**Q & A**

**Question:** Is NUM_ORDERED functionally dependent on PART_NUM?
**Answer:** No. Again, just as with ORDER_NUM, a PART_NUM might be associated with several items in an order, so PART_NUM does not provide enough information.

**Q & A**

**Question:** On which columns in the ORDER_LINE table is NUM_ORDERED functionally dependent?
**Answer:** To determine a value for NUM_ORDERED, you need both an order number and a part number. In other words, NUM_ORDERED is functionally dependent on the combination (formally called the **concatenation**) of ORDER_NUM and PART_NUM. That is, given an order number *and* a part number, you can find a single value for NUM_ORDERED.

At this point, a question naturally arises: how do you determine functional dependencies? Can you determine them by looking at sample data, for example? The answer is no.

Consider the REP table in Figure 2-5, in which last names are unique. It is very tempting to say that LAST_NAME functionally determines STREET, CITY, STATE, and ZIP (or equivalently that STREET, CITY, STATE, and ZIP are all functionally dependent on LAST_NAME). After all, given the last name of a rep, you can find the single address.

REP

| REP_ NUM | LAST_ NAME | FIRST_ NAME | STREET | CITY | STATE | ZIP | COMMISSION | RATE |
|---|---|---|---|---|---|---|---|---|
| 20 | Kaiser | Valerie | 624 Randall | Grove | FL | 33321 | $20,542.50 | 0.05 |
| 35 | Hull | Richard | 532 Jackson | Sheldon | FL | 33553 | $39,216.00 | 0.07 |
| 65 | Perez | Juan | 1626 Taylor | Fillmore | FL | 33336 | $23,487.00 | 0.05 |

**FIGURE 2-5** REP table

What happens when rep 85, whose last name is also Kaiser, is added to the database? You then have the situation illustrated in Figure 2-6. Because there are now two reps with the last name of Kaiser, you can no longer find a single address using a rep's last name—you were misled by the original data. The only way to determine functional dependencies is to examine the user's policies. This process can involve discussions with users, an examination of user documentation, and so on. For example, if managers at Premiere Products have a policy never to hire two reps with the same last name, then LAST_NAME would indeed determine the other columns. Without such a policy, however, LAST_NAME would not determine the other columns.

REP

| REP_ NUM | LAST_ NAME | FIRST_ NAME | STREET | CITY | STATE | ZIP | COMMISSION | RATE |
|---|---|---|---|---|---|---|---|---|
| 20 | Kaiser | Valerie | 624 Randall | Grove | FL | 33321 | $20,542.50 | 0.05 |
| 35 | Hull | Richard | 532 Jackson | Sheldon | FL | 33553 | $39,216.00 | 0.07 |
| 65 | Perez | Juan | 1626 Taylor | Fillmore | FL | 33336 | $23,487.00 | 0.05 |
| 85 | Kaiser | William | 172 Bahia | Norton | FL | 39281 | $0.00 | 0.05 |

**FIGURE 2-6** REP table with two reps named Kaiser

## PRIMARY KEYS

Another important database design concept is the primary key. In the simplest terms, the **primary key** is the unique identifier for a table. For example, the REP_NUM column is the unique identifier for the REP table. Given a rep number in the table, such as 20, there will only be one row on which that rep number occurs. Thus, the rep number 20 uniquely identifies a row (in this case, the first row, and the rep named Valerie Kaiser).

In this text, the definition of primary key needs to be more precise than a unique identifier for a table. Specifically, column A (or a collection of columns) is the primary key for a table if:

Property 1. *All* columns in the table are functionally dependent on A.

Property 2. No subcollection of the columns in A (assuming A is a collection of columns and not just a single column) also has property 1.

**Q & A**

**Question:** Is the CLASS column the primary key for the PART table?
**Answer:** No, because the other columns are not functionally dependent on CLASS. Given the class HW, for example, you cannot determine a part number, description, or anything else, because there are several rows on which the class is HW.

**Q & A**

**Question:** Is the CUSTOMER_NUM column the primary key for the CUSTOMER table?
**Answer:** Yes, because Premiere Products assigns unique customer numbers. A specific customer number cannot appear on more than one row. Thus, all columns in the CUSTOMER table are functionally dependent on CUSTOMER_NUM.

**Q & A**

**Question:** Is the ORDER_NUM column the primary key for the ORDER_LINE table?
**Answer:** No, because it does not functionally determine either NUM_ORDERED or QUOTED_PRICE.

**Q & A**

**Question:** Is the combination of the ORDER_NUM and PART_NUM columns the primary key for the ORDER_LINE table?
**Answer:** Yes, because you can determine all columns by this combination of columns, and, further, neither the ORDER_NUM nor the PART_NUM alone has this property.

**Q & A**

**Question:** Is the combination of the PART_NUM and DESCRIPTION columns the primary key for the PART table?
**Answer:** No. Although it is true that you can determine all columns in the PART table by this combination, PART_NUM alone also has this property.

You can indicate a table's primary key with a shorthand representation of a database by underlining the column or collection of columns that comprise the primary key. The complete shorthand representation for the Premiere Products database is:

```
REP (REP_NUM, LAST_NAME, FIRST_NAME, STREET,
     CITY, STATE, ZIP, COMMISSION, RATE)
CUSTOMER (CUSTOMER_NUM, CUSTOMER_NAME, STREET,
     CITY, STATE, ZIP, BALANCE, CREDIT_LIMIT,
     REP_NUM)
ORDERS (ORDER_NUM, ORDER_DATE, CUSTOMER_NUM)
ORDER_LINE (ORDER_NUM, PART_NUM, NUM_ORDERED,
     QUOTED_PRICE)
PART (PART_NUM, DESCRIPTION, ON_HAND, CLASS,
     WAREHOUSE, PRICE)
```

**NOTE**

Sometimes you might identify one or more columns that you can use as a table's primary key. For example, if the Premiere Products database also included an EMPLOYEE table that contains employee numbers and Social Security numbers, either the employee number or the Social Security number could serve as the table's primary key. In this case, both columns are referred to as candidate keys. Like a primary key, a **candidate key** is a column or collection of columns on which all columns in the table are functionally dependent—the definition for primary key really defines candidate key as well. From all the candidate keys, you would choose one to be the primary key.

**NOTE**

According to the definition of a candidate key, a Social Security number is a legitimate primary key. Many databases, such as those that store data about students at a college or university or those that store data about employees at a company, store a person's Social Security number as a primary key. However, many institutions and organizations are moving away from using Social Security numbers as primary keys because of privacy issues. Instead of using Social Security numbers, many institutions and organizations use unique student numbers or employee numbers as primary keys.

**NOTE**

Some institutions prefer to assign values to use as primary keys for items such as customer numbers, part numbers, and student numbers. Others simply let the computer generate the values. In this case, the DBMS simply assigns the next available value. For example, if the database has already assigned customer numbers 1000 through 1436, it assigns the next new customer added to the database the customer number 1437.

## DATABASE DESIGN

This section presents a specific method you can follow to design a database when given a set of requirements that the database must support. The determination of the requirements is part of the process known as systems analysis. A systems analyst interviews users, examines existing and proposed documents, and examines organizational policies to determine exactly the type of data needs the database must support. This text does not cover this analysis. Rather, it focuses on how to take the set of requirements that this process produces and determine the appropriate database design.

After presenting the database design method, this section presents a sample set of requirements and illustrates the design method by designing a database to satisfy these requirements.

### Design Method

To design a database for a set of requirements, complete the following steps:

1. Read the requirements, identify the entities (objects) involved, and name the entities. For example, when the design involves departments and employees, you might use the entity names DEPARTMENT and EMPLOYEE. When the design involves customers and sales reps, you might use the entity names CUSTOMER and REP.
2. Identify the unique identifiers for the entities you identified in Step 1. For example, when one of the entities is PART, determine what information is required to uniquely identify each individual part. In other words, what information does the organization use to distinguish one part from another? For a PART entity, the unique identifier for each part might be a PART_NUM; for a CUSTOMER entity, the unique identifier might be a CUSTOMER_NUM. When no unique identifier is available from the data you know about the entity, you need to create one. For example, you might use a unique number to identify parts when no part numbers exist.
3. Identify the attributes for all the entities. These attributes become the columns in the tables. It is possible for two or more entities to contain the same attributes. At Premiere Products, for example, reps and customers both have addresses, cities, states, and zip codes. To clarify this duplication of attributes, follow the name of the attribute with the corresponding entity in parentheses. Thus, ADDRESS (CUSTOMER) is a customer address and ADDRESS (REP) is a sales rep address.
4. Identify the functional dependencies that exist among the attributes. Ask yourself the following question: if you know a unique value for an attribute, do you also

know the unique values for other attributes? For example, when you have the three attributes REP_NUM, LAST_NAME, and FIRST_NAME and you know a unique value for REP_NUM, do you also know a unique value for LAST_NAME and FIRST_NAME? If so, then LAST_NAME and FIRST_NAME are functionally dependent on REP_NUM (REP_NUM → LAST_NAME, FIRST_NAME).

5. Use the functional dependencies to identify the tables by placing each attribute with the attribute or minimum combination of attributes on which it is functionally dependent. The attribute or attributes for an entity on which all other attributes are dependent will be the primary key of the table. The remaining attributes will be the other columns in the table. Once you have determined all the columns in the table, you can give the table an appropriate name. Usually the name will be the same as the name you identified for the entity in Step 1.
6. Identify any relationships between tables. In some cases, you might be able to determine the relationships directly from the requirements. It might be clear, for example, that one rep is related to many customers and that each customer is related to exactly one rep. When it is not, look for matching columns in the tables you created. For example, if both the REP table and the CUSTOMER table contain a REP_NUM column and the values in these columns must match, you know that reps and customers are related. The fact that the REP_NUM column is the primary key in the REP table tells you that the REP table is the "one" part of the relationship and the CUSTOMER table is the "many" part of the relationship.

In the next section, you will apply this process to produce the design for the Premiere Products database using the collection of requirements that this database must support.

## Database Design Requirements

The analyst has interviewed users and examined documents at Premiere Products and has determined that the database must support the following requirements:

1. For a sales rep, store the sales rep's number, last name, first name, street address, city, state, zip code, total commission, and commission rate.
2. For a customer, store the customer's number, name, street address, city, state, zip code, balance, and credit limit. In addition, store the number, last name, and first name of the sales rep who represents this customer. The analyst has also determined that a sales rep can represent many customers, but a customer must have exactly one sales rep (in other words, a sales rep must represent a customer; a customer cannot be represented by zero or more than one sales reps).
3. For a part, store the part's number, description, units on hand, item class, the number of the warehouse in which the part is located, and the price. All units of a particular part are stored in the same warehouse.
4. For an order, store the order number, order date, the number and name of the customer that placed the order, and the number of the sales rep who represents that customer.

5. For each line item within an order, store the part number and description, the number ordered, and the quoted price. The analyst also obtained the following information concerning orders:
   a. There is only one customer per order.
   b. On a given order, there is at most one line item for a given part. For example, part DR93 cannot appear on several lines within the same order.
   c. The quoted price might differ from the actual price when the sales rep discounts a certain part on a specific order.

## Database Design Process Example

The following steps apply the design process to the requirements for Premiere Products to produce the appropriate database design:

**Step 1:** There appear to be four entities: reps, customers, parts, and orders. The names assigned to these entities are REP, CUSTOMER, PART, and ORDERS, respectively.

**Step 2:** From the collection of entities, review the data and determine the unique identifier for each entity. For the REP, CUSTOMER, PART, and ORDERS entities, the unique identifiers are the rep number, customer number, part number, and order number, respectively. These unique identifiers are named REP_NUM, CUSTOMER_NUM, PART_NUM, and ORDER_NUM, respectively.

**Step 3:** The attributes mentioned in the first requirement all refer to sales reps. The specific attributes mentioned in the requirement are the sales rep's number, name, street address, city, state, zip code, total commission, and commission rate. Assigning appropriate names to these attributes produces the following list:

```
REP_NUM
LAST_NAME
FIRST_NAME
STREET
CITY
STATE
ZIP
COMMISSION
RATE
```

The attributes mentioned in the second requirement refer to customers. The specific attributes are the customer's number, name, street address, city, state, zip code, balance, and credit limit. The requirement also mentions the number, first name, and last name of the sales rep who represents this customer. Assigning appropriate names to these attributes produces the following list:

```
CUSTOMER_NUM
CUSTOMER_NAME
STREET
CITY
STATE
ZIP
BALANCE
CREDIT_LIMIT
REP_NUM
LAST_NAME
FIRST_NAME
```

There are attributes named STREET, CITY, STATE, and ZIP for sales reps as well as attributes named STREET, CITY, STATE, and ZIP for customers. To distinguish these attributes in the final collection, follow the name of the attribute by the name of the corresponding entity. For example, the street for a sales rep is STREET (REP) and the street for a customer is STREET (CUSTOMER).

The attributes mentioned in the third requirement refer to parts. The specific attributes are the part's number, description, units on hand, item class, the number of the warehouse in which the part is located, and the price. Assigning appropriate names to these attributes produces the following list:

```
PART_NUM
DESCRIPTION
ON_HAND
CLASS
WAREHOUSE
PRICE
```

The attributes mentioned in the fourth requirement refer to orders. The specific attributes include the order number, order date, number and name of the customer that placed the order, and number of the sales rep who represents the customer. Assigning appropriate names to these attributes produces the following list:

```
ORDER_NUM
ORDER_DATE
CUSTOMER_NUM
CUSTOMER_NAME
REP_NUM
```

The specific attributes associated with the statement in the requirements concerning line items are the order number (to determine the order to which the line item corresponds), part number, description, number ordered, and quoted price. If the quoted price must be the same as the price, you could simply call it PRICE. According to requirement 5c, however, the quoted price might differ from the price, so you must add the quoted price to the list. Assigning appropriate names to these attributes produces the following list:

```
ORDER_NUM
PART_NUM
DESCRIPTION
NUM_ORDERED
QUOTED_PRICE
```

The complete list grouped by entity is as follows:

**REP**
```
REP_NUM
LAST_NAME
FIRST_NAME
STREET (REP)
CITY (REP)
STATE (REP)
ZIP (REP)
COMMISSION
RATE
```

```
CUSTOMER
CUSTOMER_NUM
CUSTOMER_NAME
STREET (CUSTOMER)
CITY (CUSTOMER)
STATE (CUSTOMER)
ZIP (CUSTOMER)
BALANCE
CREDIT_LIMIT
REP_NUM
LAST_NAME
FIRST_NAME

PART
PART_NUM
DESCRIPTION
ON_HAND
CLASS
WAREHOUSE
PRICE

ORDER
ORDER_NUM
ORDER_DATE
CUSTOMER_NUM
CUSTOMER_NAME
REP_NUM

For line items within an order
ORDER_NUM
PART_NUM
DESCRIPTION
NUM_ORDERED
QUOTED_PRICE
```

**Step 4:** The fact that the unique identifier for sales reps is the rep number gives the following functional dependencies:

```
REP_NUM → LAST_NAME, FIRST_NAME, STREET (REP), CITY (REP),
       STATE (REP), ZIP (REP), COMMISSION, RATE
```

This notation indicates that the LAST_NAME, FIRST_NAME, STREET (REP), CITY (REP), STATE (REP), ZIP (REP), COMMISSION, and RATE are all functionally dependent on REP_NUM.

The fact that the unique identifier for customers is the customer number gives the following functional dependencies:

```
CUSTOMER_NUM → CUSTOMER_NAME, STREET (CUSTOMER),
      CITY (CUSTOMER), STATE (CUSTOMER), ZIP (CUSTOMER),
      BALANCE, CREDIT_LIMIT, REP_NUM, LAST_NAME, FIRST_NAME
```

**Q & A**

**Question:** Do you really need to include the last name and first name of a sales rep in the list of attributes determined by the customer number?
**Answer:** There is no need to include them in this list, because they both can be determined from the sales rep number and are already included in the list of attributes determined by REP_NUM.

Thus, the functional dependencies for the CUSTOMER entity are as follows:

```
CUSTOMER_NUM → CUSTOMER_NAME, STREET (CUSTOMER),
     CITY (CUSTOMER), STATE (CUSTOMER), ZIP (CUSTOMER),
     BALANCE, CREDIT_LIMIT, REP_NUM
```

The fact that the unique identifier for parts is the part number gives the following functional dependencies:

```
PART_NUM → DESCRIPTION, ON_HAND, CLASS, WAREHOUSE, PRICE
```

The fact that the unique identifier for orders is the order number gives the following functional dependencies:

```
ORDER_NUM → ORDER_DATE, CUSTOMER_NUM, CUSTOMER_NAME,
     REP_NUM
```

**Q & A**

**Question:** Do you really need to include the name of a customer and the number of the customer's rep in the list of attributes determined by the order number?
**Answer:** There is no need to include the customer name and the rep number in this list, because you can determine them from the customer number and they are already included in the list of attributes determined by CUSTOMER_NUM.

The functional dependencies for the ORDERS entity are as follows:

```
ORDER_NUM → ORDER_DATE, CUSTOMER_NUM
```

The final attributes to be examined are those associated with the line items within the order: PART_NUM, DESCRIPTION, NUM_ORDERED, and QUOTED_PRICE.

**Q & A**

**Question:** Why aren't NUM_ORDERED and QUOTED_PRICE included in the list of attributes determined by the order number?
**Answer:** To uniquely identify a particular value for NUM_ORDERED or QUOTED_PRICE, ORDER_NUM alone is not sufficient. It requires the combination of ORDER_NUM and PART_NUM.

The following shorthand representation indicates that the combination of ORDER_NUM and PART_NUM functionally determines NUM_ORDERED and QUOTED_PRICE:

```
ORDER_NUM, PART_NUM → NUM_ORDERED, QUOTED_PRICE
```

**Q & A**

**Question:** Does DESCRIPTION need to be included in this list?
**Answer:** No, because DESCRIPTION can be determined by the PART_NUMBER alone, and it already appears in the list of attributes dependent on the PART_NUM.

The complete list of functional dependencies is as follows:

```
REP_NUM → LAST_NAME, FIRST_NAME, STREET (REP), CITY (REP),
     STATE (REP), ZIP(REP), COMMISSION, RATE
CUSTOMER_NUM → CUSTOMER_NAME, STREET (CUSTOMER),
     CITY (CUSTOMER), STATE (CUSTOMER), ZIP (CUSTOMER),
     BALANCE, CREDIT_LIMIT, REP_NUM
PART_NUM → DESCRIPTION, ON_HAND, CLASS, WAREHOUSE, PRICE
ORDER_NUM → ORDER_DATE, CUSTOMER_NUM
ORDER_NUM, PART_NUM → NUM_ORDERED, QUOTED_PRICE
```

**Step 5:** Using the functional dependencies, you can create tables with the attribute(s) to the left of the arrow being the primary key and the items to the right of the arrow being the other columns. For relations corresponding to those entities identified in Step 1, you can use the name you already determined. Because you did not identify any entity that had a unique identifier that was the combination of ORDER_NUM and PART_NUM, you need to assign a name to the table whose primary key consists of these two columns. Because this table represents the individual lines within an order, the name ORDER_LINE is a good choice. The final collection of tables is as follows:

```
REP (REP_NUM, LAST_NAME, FIRST_NAME, STREET,
     CITY, STATE, ZIP, COMMISSION, RATE)
CUSTOMER (CUSTOMER_NUM, CUSTOMER_NAME, STREET,
     CITY, STATE, ZIP, BALANCE, CREDIT_LIMIT,
     REP_NUM)
PART (PART_NUM, DESCRIPTION, ON_HAND, CLASS,
      WAREHOUSE, PRICE)
ORDERS (ORDER_NUM, ORDER_DATE, CUSTOMER_NUM)
ORDER_LINE (ORDER_NUM, PART_NUM, NUM_ORDERED,
      QUOTED_PRICE)
```

**Step 6:** Examining the tables and identifying common columns gives the following list of relationships between the tables:

- The CUSTOMER and REP tables are related using the REP_NUM columns. Because the REP_NUM column is the primary key for the REP table, this indicates a one-to-many relationship between REP and CUSTOMER (one rep to many customers).
- The ORDERS and CUSTOMER tables are related using the CUSTOMER_NUM columns. Because the CUSTOMER_NUM column is the primary key for the CUSTOMER table, this indicates a one-to-many relationship between CUSTOMER and ORDERS (one customer to many orders).
- The ORDER_LINE and ORDERS tables are related using the ORDER_NUM columns. Because the ORDER_NUM column is the primary key for the ORDERS table, this indicates a one-to-many relationship between ORDERS and ORDER_LINE (one order to many order lines).
- The ORDER_LINE and PART tables are related using the PART_NUM columns. Because the PART_NUM column is the primary key for the PART table, this indicates a one-to-many relationship between PART and ORDER_LINE (one part to many order lines).

## NORMALIZATION

After creating the database design, you must analyze it to make sure it is free of potential problems. To do so, you follow a process called **normalization**, in which you identify the existence of potential problems, such as data duplication and redundancy, and implement ways to correct these problems.

The goal of normalization is to convert **unnormalized relations** (tables that satisfy the definition of a relation except that they might contain repeating groups) into various types of **normal forms**. A table in a particular normal form possesses a certain desirable collection of properties. Although there are several normal forms, the most common are first normal form, second normal form, and third normal form. Normalization is a process in which a table that is in first normal form is better than a table that is not in first normal form, a table that is in second normal form is better than one that is in first normal form, and so on. The goal of this process is to allow you to take a table or collection of tables and produce a new collection of tables that represents the same information but is free of problems.

### First Normal Form

According to the definition of a relation, a relation (table) cannot contain a repeating group in which multiple entries exist on a single row. However, in the database design process, you might create a table that has all the other properties of a relation, but contains a repeating group. Removing repeating groups is the starting point when converting an unnormalized collection of data into a table that is in first normal form. A table (relation) is in **first normal form (1NF)** when it does not contain a repeating group.

For example, in the design process you might create the following ORDERS table, in which there is a repeating group consisting of PART_NUM and NUM_ORDERED. The notation for this table is as follows:

```
ORDERS (ORDER_NUM, ORDER_DATE, (PART_NUM, NUM_ORDERED) )
```

This notation describes a table named ORDERS that consists of a primary key, ORDER_NUM, and a column named ORDER_DATE. The inner parentheses indicate a repeating group that contains two columns, PART_NUM and NUM_ORDERED. This table contains one row per order with values in the PART_NUM and NUM_ORDERED columns for each order with the number ORDER_NUM and placed on ORDER_DATE. Figure 2-7 shows a single order with multiple combinations of a part number and a corresponding number of units ordered.

ORDERS

| ORDER_ NUM | ORDER_ DATE | PART_ NUM | NUM_ ORDERED |
|---|---|---|---:|
| 21608 | 10/20/2010 | AT94 | 11 |
| 21610 | 10/20/2010 | DR93 | 1 |
| | | DW11 | 1 |
| 21613 | 10/21/2010 | KL62 | 4 |
| 21614 | 10/21/2010 | KT03 | 2 |
| 21617 | 10/23/2010 | BV06 | 2 |
| | | CD52 | 4 |
| 21619 | 10/23/2010 | DR93 | 1 |
| 21623 | 10/23/2010 | KV29 | 2 |

**FIGURE 2-7** Unnormalized order data

To convert the table to first normal form, you remove the repeating group as follows:

```
ORDERS (ORDER_NUM, ORDER_DATE, PART_NUM, NUM_ORDERED)
```

Figure 2-8 shows the table in first normal form.

ORDERS

| ORDER_ NUM | ORDER_ DATE | PART_ NUM | NUM_ ORDERED |
|---|---|---|---|
| 21608 | 10/20/2010 | AT94 | 11 |
| 21610 | 10/20/2010 | DR93 | 1 |
| 21610 | 10/20/2010 | DW11 | 1 |
| 21613 | 10/21/2010 | KL62 | 4 |
| 21614 | 10/21/2010 | KT03 | 2 |
| 21617 | 10/23/2010 | BV06 | 2 |
| 21617 | 10/23/2010 | CD52 | 4 |
| 21619 | 10/23/2010 | DR93 | 1 |
| 21623 | 10/23/2010 | KV29 | 2 |

**FIGURE 2-8** Order data converted to first normal form

In Figure 2-7, the second row indicates that part DR93 and part DW11 are both included in order 21610. In Figure 2-8, this information is represented by *two* rows, the second and third. The primary key for the unnormalized ORDERS table was the ORDER_NUM column alone. The primary key for the normalized table is now the combination of the ORDER_NUM and PART_NUM columns.

When you convert an unnormalized table to a table in first normal form, the primary key of the table in first normal form is usually the primary key of the unnormalized table concatenated with the key for the repeating group, which is the column in the repeating group that distinguishes one occurrence of the repeating group from another within a given row in the table. In the ORDERS table, PART_NUM was the key to the repeating group and ORDER_NUM was the primary key for the table. When converting the unnormalized data to first normal form, the primary key becomes the concatenation of the ORDER_NUM and PART_NUM columns.

## Second Normal Form

The following ORDERS table is in first normal form, because it does not contain a repeating group:

```
ORDERS (ORDER_NUM, ORDER_DATE, PART_NUM, DESCRIPTION,
     NUM_ORDERED, QUOTED_PRICE)
```

The table contains the following functional dependencies:

```
ORDER_NUM → ORDER_DATE
PART_NUM → DESCRIPTION
ORDER_NUM, PART_NUM → NUM_ORDERED, QUOTED_PRICE
```

This notation indicates that ORDER_NUM alone determines ORDER_DATE, and PART_NUM alone determines DESCRIPTION, but it requires *both* an ORDER_NUM *and* a PART_NUM to determine either NUM_ORDERED or QUOTED_PRICE. Consider the sample of this table shown in Figure 2-9.

ORDERS

| ORDER_ NUM | ORDER_ DATE | PART_ NUM | DESCRIPTION | NUM_ ORDERED | QUOTED_ PRICE |
|---|---|---|---|---:|---:|
| 21608 | 10/20/2010 | AT94 | Iron | 11 | $21.95 |
| 21610 | 10/20/2010 | DR93 | Gas Range | 1 | $495.00 |
| 21610 | 10/20/2010 | DW11 | Washer | 1 | $399.99 |
| 21613 | 10/21/2010 | KL62 | Dryer | 4 | $329.95 |
| 21614 | 10/21/2010 | KT03 | Dishwasher | 2 | $595.00 |
| 21617 | 10/23/2010 | BV06 | Home Gym | 2 | $12.95 |
| 21617 | 10/23/2010 | CD52 | Microwave Oven | 4 | $150.00 |
| 21619 | 10/23/2010 | DR93 | Gas Range | 1 | $495.00 |
| 21623 | 10/23/2010 | KV29 | Treadmill | 2 | $325.99 |

**FIGURE 2-9** Sample ORDERS table

Although the ORDERS table is in first normal form (because it contains no repeating groups), problems exist within the table that require you to restructure it.

The description of a specific part, DR93 for example, occurs twice in the table. This duplication (formally called **redundancy**) causes several problems. It is certainly wasteful of space, but that is not nearly as serious as some of the other problems. These other problems are called **update anomalies** and they fall into four categories:

1. **Updates:** If you need to change to the description of part DR93, you must change it twice—once in each row on which part DR93 appears. Updating the part description more than once makes the update process much more cumbersome and time consuming.
2. **Inconsistent data:** There is nothing about the design that prohibits part DR93 from having two *different* descriptions in the database. In fact, if part DR93 occurs on 20 rows in the table, it is possible for this part to have 20 different descriptions in the database.
3. **Additions:** When you try to add a new part and its description to the database, you will face a real problem. Because the primary key for the ORDERS table consists of both an ORDER_NUM and a PART_NUM, you need values for both of these columns to add a new row to the table. If you add a part to the table that does not yet have any orders, what do you use for an ORDER_NUM? The only solution is to create a dummy ORDER_NUM and then replace

it with a real ORDER_NUM once an order for this part is actually received. Certainly this is not an acceptable solution.

4. **Deletions:** If you delete order 21608 from the database and it is the only order that contains part AT94, deleting the order also deletes all information about part AT94. For example, you would no longer know that part AT94 is an iron.

These problems occur because you have a column, DESCRIPTION, that is dependent on only a portion of the primary key, PART_NUM, and *not* on the complete primary key. This situation leads to the definition of second normal form. Second normal form represents an improvement over first normal form because it eliminates update anomalies in these situations. A table (relation) is in **second normal form** (**2NF**) when it is in first normal form and no **nonkey column** (that is, a column that is not part of the primary key) is dependent on only a portion of the primary key.

NOTE

When the primary key of a table contains only a single column, the table is automatically in second normal form.

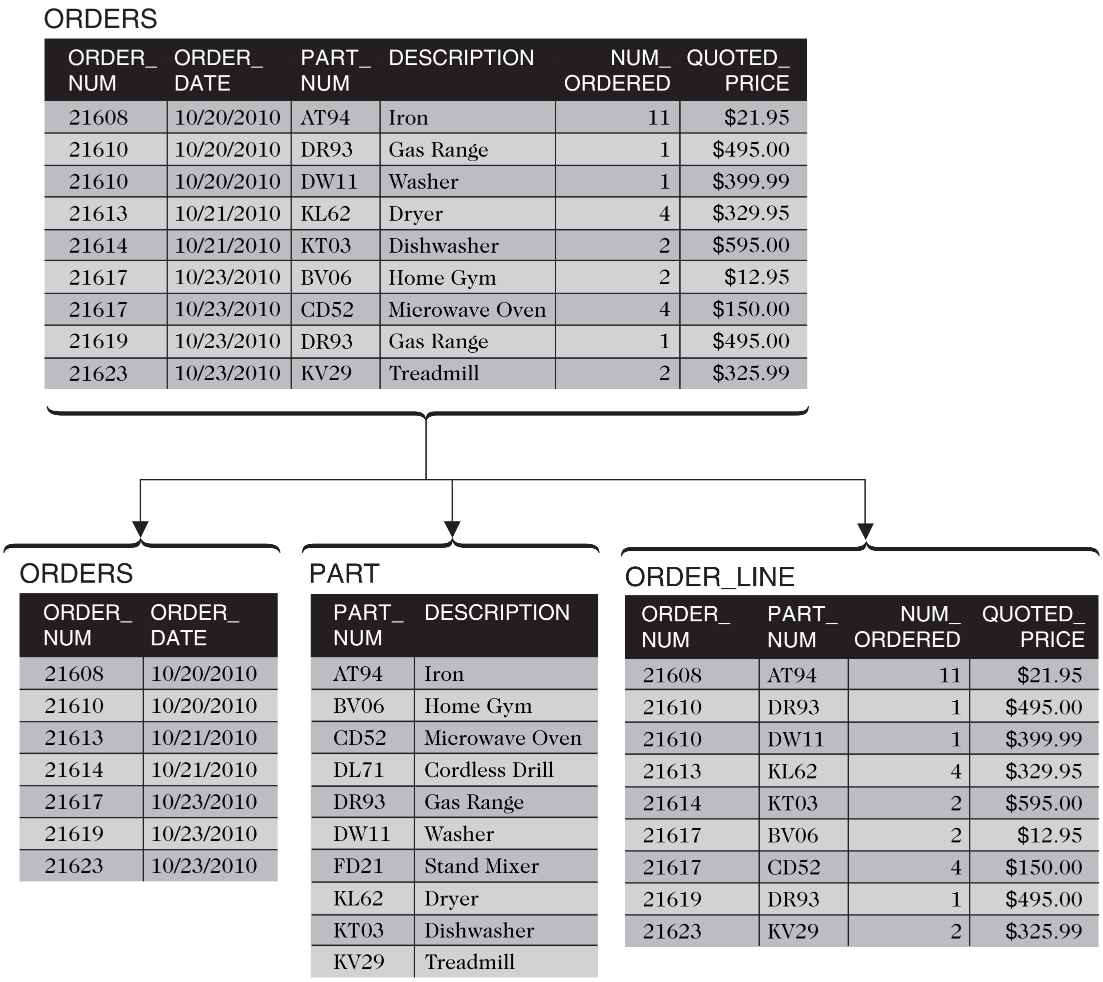
You can identify the fundamental problem with the ORDERS table: it is not in second normal form. Although it is important to identify the problem, what you really need is a method to *correct* it; you want to be able to convert tables to second normal form. First, take each subset of the set of columns that make up the primary key, and begin a new table with this subset as its primary key. For the ORDERS table, the new design is:

```
(ORDER_NUM,
(PART_NUM,
(ORDER_NUM, PART_NUM,
```

Next, place each of the other columns with the appropriate primary key; that is, place each one with the minimal collection of columns on which it depends. For the ORDERS table, add the new columns as follows:

```
(ORDER_NUM, ORDER_DATE)
(PART_NUM, DESCRIPTION)
(ORDER_NUM, PART_NUM, NUM_ORDERED, QUOTED_PRICE)
```

Each of these new tables is given a descriptive name based on the meaning and contents of the table, such as ORDERS, PART, and ORDER_LINE. Figure 2-10 shows samples of these tables.



ORDERS

| ORDER_ NUM | ORDER_ DATE | PART_ NUM | DESCRIPTION | NUM_ ORDERED | QUOTED_ PRICE |
|---|---|---|---|---|---|
| 21608 | 10/20/2010 | AT94 | Iron | 11 | $21.95 |
| 21610 | 10/20/2010 | DR93 | Gas Range | 1 | $495.00 |
| 21610 | 10/20/2010 | DW11 | Washer | 1 | $399.99 |
| 21613 | 10/21/2010 | KL62 | Dryer | 4 | $329.95 |
| 21614 | 10/21/2010 | KT03 | Dishwasher | 2 | $595.00 |
| 21617 | 10/23/2010 | BV06 | Home Gym | 2 | $12.95 |
| 21617 | 10/23/2010 | CD52 | Microwave Oven | 4 | $150.00 |
| 21619 | 10/23/2010 | DR93 | Gas Range | 1 | $495.00 |
| 21623 | 10/23/2010 | KV29 | Treadmill | 2 | $325.99 |

ORDERS

| ORDER_ NUM | ORDER_ DATE |
|---|---|
| 21608 | 10/20/2010 |
| 21610 | 10/20/2010 |
| 21613 | 10/21/2010 |
| 21614 | 10/21/2010 |
| 21617 | 10/23/2010 |
| 21619 | 10/23/2010 |
| 21623 | 10/23/2010 |

PART

| PART_ NUM | DESCRIPTION |
|---|---|
| AT94 | Iron |
| BV06 | Home Gym |
| CD52 | Microwave Oven |
| DL71 | Cordless Drill |
| DR93 | Gas Range |
| DW11 | Washer |
| FD21 | Stand Mixer |
| KL62 | Dryer |
| KT03 | Dishwasher |
| KV29 | Treadmill |

ORDER_LINE

| ORDER_ NUM | PART_ NUM | NUM_ ORDERED | QUOTED_ PRICE |
|---|---|---|---|
| 21608 | AT94 | 11 | $21.95 |
| 21610 | DR93 | 1 | $495.00 |
| 21610 | DW11 | 1 | $399.99 |
| 21613 | KL62 | 4 | $329.95 |
| 21614 | KT03 | 2 | $595.00 |
| 21617 | BV06 | 2 | $12.95 |
| 21617 | CD52 | 4 | $150.00 |
| 21619 | DR93 | 1 | $495.00 |
| 21623 | KV29 | 2 | $325.99 |

**FIGURE 2-10** ORDERS table converted to second normal form

In Figure 2-10, converting the original ORDERS table to a new ORDERS table, a PART table, and an ORDER_LINE table eliminates the update anomalies. A description appears only once for each part, so you do not have the redundancy that existed in the original table design. Changing the description of part DR93 from Gas Range to Deluxe Range, for example, is now a simple process involving a single change. Because the description for a part occurs in a single place, it is not possible to have multiple descriptions for a single part in the database at the same time.

To add a new part and its description, you create a new row in the PART table, regardless of whether that part has pending or actual orders. Also, deleting order 21608 does not delete part number AT94 from the database because it still exists in the PART table. Finally, you have not lost any information by converting the ORDERS table to second normal form. You can reconstruct the data in the original table from the data in the new tables.

## Third Normal Form

Problems can still exist with tables that are in second normal form. For example, suppose that you create the following CUSTOMER table:

```
CUSTOMER (CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, CREDIT_LIMIT,
     REP_NUM, LAST_NAME, FIRST_NAME)
```

This table has the following functional dependencies:

```
CUSTOMER_NUM → CUSTOMER_NAME, BALANCE, CREDIT_LIMIT,
     REP_NUM, LAST_NAME, FIRST_NAME
REP_NUM → LAST_NAME, FIRST_NAME
```

CUSTOMER_NUM determines all the other columns. In addition, REP_NUM determines LAST_NAME and FIRST_NAME.

When a table's primary key is a single column, the table is automatically in second normal form. (If the table were not in second normal form, some column would be dependent on only a *portion* of the primary key, which is impossible when the primary key is just one column.) Thus, the CUSTOMER table is in second normal form.

Although this table is in second normal form, Figure 2-11 shows that it still possesses update problems similar to those identified for the ORDERS table shown in Figure 2-9. In Figure 2-11, the sales rep name occurs many times in the table.

CUSTOMER

| CUSTOMER_ NUM | CUSTOMER_NAME | BALANCE | CREDIT_ LIMIT | REP_ NUM | LAST_ NAME | FIRST_ NAME |
|---|---|---|---|---|---|---|
| 148 | Al's Appliance and Sport | $6,550.00 | $7,500.00 | 20 | Kaiser | Valerie |
| 282 | Brookings Direct | $431.50 | $10,000.00 | 35 | Hull | Richard |
| 356 | Ferguson's | $5,785.00 | $7,500.00 | 65 | Perez | Juan |
| 408 | The Everything Shop | $5,285.25 | $5,000.00 | 35 | Hull | Richard |
| 462 | Bargains Galore | $3,412.00 | $10,000.00 | 65 | Perez | Juan |
| 524 | Kline's | $12,762.00 | $15,000.00 | 20 | Kaiser | Valerie |
| 608 | Johnson's Department Store | $2,106.00 | $10,000.00 | 65 | Perez | Juan |
| 687 | Lee's Sport and Appliance | $2,851.00 | $5,000.00 | 35 | Hull | Richard |
| 725 | Deerfield's Four Seasons | $248.00 | $7,500.00 | 35 | Hull | Richard |
| 842 | All Season | $8,221.00 | $7,500.00 | 20 | Kaiser | Valerie |

**FIGURE 2-11** Sample CUSTOMER table

The redundancy of including a sales rep number and name in the CUSTOMER table results in the same set of problems that existed for the ORDERS table. In addition to the problem of wasted space, you have the following update anomalies:

1. **Updates:** Changing the sales rep name requires changes to multiple rows in the table.
2. **Inconsistent data:** The design does not prohibit multiple iterations of sales rep names in the database. For example, a sales rep might represent 20 customers and his name might be entered 20 different ways in the table.
3. **Additions:** To add sales rep 87 (Emily Daniels) to the database, she must represent at least one customer. If Emily does not yet represent any customers, you either cannot record the fact that her name is Emily Daniels or you must create a fictitious customer for her to represent until she represents an actual customer. Neither of these solutions is desirable.
4. **Deletions:** If you delete all the customers of sales rep 35 from the database, you will also lose all information about sales rep 35.

These update anomalies are due to the fact that REP_NUM determines LAST_NAME and FIRST_NAME, but REP_NUM is not the primary key. As a result, the same REP_NUM and consequently the same LAST_NAME and FIRST_NAME can appear on many different rows.

You have seen that tables in second normal form represent an improvement over tables in first normal form, but to eliminate problems with tables in second normal form, you need an even better strategy for creating tables. Third normal form provides that strategy.

Before looking at third normal form, however, you need to become familiar with the special name that is given to any column that determines another column (like REP_NUM in the CUSTOMER table). Any column (or collection of columns) that determines another column is called a **determinant**. A table's primary key is a determinant. In fact, by definition, any candidate key is a determinant. (Remember that a candidate key is a column or collection of columns that could function as the primary key.) In Figure 2-11, REP_NUM is a determinant, but it is not a candidate key, and that is the problem.

A table is in **third normal form (3NF)** when it is in second normal form and the only determinants it contains are candidate keys.

**NOTE**

This text's definition of third normal form is not the original definition. This more recent definition, which is preferable to the original, is often referred to as **Boyce-Codd normal form (BCNF)** when it is important to make a distinction between this definition and the original definition. This text does not make such a distinction but will take this to be the definition of third normal form.

Now you have identified the problem with the CUSTOMER table: it is not in third normal form. There are several steps for converting tables to third normal form.

First, for each determinant that is not a candidate key, remove from the table the columns that depend on this determinant (but do not remove the determinant). Next, create a new table containing all the columns from the original table that depend on this determinant. Finally, make the determinant the primary key of this new table.

In the CUSTOMER table, for example, remove LAST_NAME and FIRST_NAME because they depend on the determinant REP_NUM, which is not a candidate key. A new table is formed, consisting of REP_NUM as the primary key, and the columns LAST_NAME and FIRST_NAME, as follows:

```
CUSTOMER (CUSTOMER_NUM, CUSTOMER_NAME, BALANCE,
     CREDIT_LIMIT, REP_NUM)
```

and

```
REP (REP_NUM, LAST_NAME, FIRST_NAME)
```

Figure 2-12 shows the original CUSTOMER table and the tables created when converting the original table to third normal form.

CUSTOMER

| CUSTOMER_ NUM | CUSTOMER_NAME | BALANCE | CREDIT_ LIMIT | REP_ NUM | LAST_ NAME | FIRST_ NAME |
|---|---|---|---|---|---|---|
| 148 | Al's Appliance and Sport | $6,550.00 | $7,500.00 | 20 | Kaiser | Valerie |
| 282 | Brookings Direct | $431.50 | $10,000.00 | 35 | Hull | Richard |
| 356 | Ferguson's | $5,785.00 | $7,500.00 | 65 | Perez | Juan |
| 408 | The Everything Shop | $5,285.25 | $5,000.00 | 35 | Hull | Richard |
| 462 | Bargains Galore | $3,412.00 | $10,000.00 | 65 | Perez | Juan |
| 524 | Kline's | $12,762.00 | $15,000.00 | 20 | Kaiser | Valerie |
| 608 | Johnson's Department Store | $2,106.00 | $10,000.00 | 65 | Perez | Juan |
| 687 | Lee's Sport and Appliance | $2,851.00 | $5,000.00 | 35 | Hull | Richard |
| 725 | Deerfield's Four Seasons | $248.00 | $7,500.00 | 35 | Hull | Richard |
| 842 | All Season | $8,221.00 | $7,500.00 | 20 | Kaiser | Valerie |

CUSTOMER

| CUSTOMER_ NUM | CUSTOMER_NAME | BALANCE | CREDIT_LIMIT | REP_NUM |
|---|---|---|---|---|
| 148 | Al's Appliance and Sport | $6,550.00 | $7,500.00 | 20 |
| 282 | Brookings Direct | $431.50 | $10,000.00 | 35 |
| 356 | Ferguson's | $5,785.00 | $7,500.00 | 65 |
| 408 | The Everything Shop | $5,285.25 | $5,000.00 | 35 |
| 462 | Bargains Galore | $3,412.00 | $10,000.00 | 65 |
| 524 | Kline's | $12,762.00 | $15,000.00 | 20 |
| 608 | Johnson's Department Store | $2,106.00 | $10,000.00 | 65 |
| 687 | Lee's Sport and Appliance | $2,851.00 | $5,000.00 | 35 |
| 725 | Deerfield's Four Seasons | $248.00 | $7,500.00 | 35 |
| 842 | All Season | $8,221.00 | $7,500.00 | 20 |

REP

| REP_NUM | LAST_NAME | FIRST_NAME |
|---|---|---|
| 20 | Kaiser | Valerie |
| 35 | Hull | Richard |
| 65 | Perez | Juan |

**FIGURE 2-12** CUSTOMER table converted to third normal form

Has this new design for the CUSTOMER table corrected all of the previously identified problems? A sales rep's name appears only once, thus avoiding redundancy and simplifying the process of changing a sales rep's name. This design prohibits a sales rep from having different names in the database. To add a new sales rep to the database, you add a row to the REP table; it is not necessary for a new rep to represent a customer. Finally, deleting all customers of a given sales rep will not remove the sales rep's record from the REP table, retaining the sales rep's name in the database. You can reconstruct all the data in the original table from the data in the new collection of tables. All previously mentioned problems have indeed been solved.

## Q & A

**Question:** Convert the following table to third normal form. In this table, STUDENT_NUM determines STUDENT_NAME, NUM_CREDITS, ADVISOR_NUM, and ADVISOR_NAME. ADVISOR_NUM determines ADVISOR_NAME. COURSE_NUM determines DESCRIPTION. The combination of a STUDENT_NUM and a COURSE_NUM determines GRADE.

```
STUDENT (STUDENT_NUM, STUDENT_NAME, NUM_CREDITS,
     ADVISOR_NUM, ADVISOR_NAME, (COURSE_NUM, DESCRIPTION,
     GRADE) )
```

**Answer:** Complete the following steps:

**Step 1.** Remove the repeating group to convert the table to first normal form, as follows:

```
STUDENT (STUDENT_NUM, STUDENT_NAME, NUM_CREDITS,
     ADVISOR_NUM, ADVISOR_NAME, COURSE_NUM, DESCRIPTION,
     GRADE)
```

The STUDENT table is now in first normal form because it has no repeating groups. It is not, however, in second normal form because STUDENT_NAME is dependent only on STUDENT_NUM, which is only a portion of the primary key.

**Step 2.** Convert the STUDENT table to second normal form. First, for each subset of the primary key, start a table with that subset as its key yielding the following:

```
(STUDENT_NUM,
(COURSE_NUM,
(STUDENT_NUM, COURSE_NUM,
```

Next, place the rest of the columns with the smallest collection of columns on which they depend, as follows:

```
(STUDENT_NUM, STUDENT_NAME, NUM_CREDITS, ADVISOR_NUM,
     ADVISOR_NAME)
(COURSE_NUM, DESCRIPTION)
(STUDENT_NUM, COURSE_NUM, GRADE)
```

Finally, assign names to each of the new tables:

```
STUDENT (STUDENT_NUM, STUDENT_NAME, NUM_CREDITS,
     ADVISOR_NUM, ADVISOR_NAME)
COURSE (COURSE_NUM, DESCRIPTION)
STUDENT_COURSE (STUDENT_NUM, COURSE_NUM, GRADE)
```

These tables are all now in second normal form, and the COURSE and STUDENT_COURSE tables are also in third normal form. The STUDENT table is not in third normal form, however, because it contains a determinant (ADVISOR_NUM) that is not a candidate key.

*continued*

**Step 3:** Convert the STUDENT table to third normal form by removing the column that depends on the determinant ADVISOR_NUM and placing it in a separate table, as follows:

```
(STUDENT_NUM, STUDENT_NAME, NUM_CREDITS, ADVISOR_NUM)
(ADVISOR_NUM, ADVISOR_NAME)
```

**Step 4:** Name the tables and put the entire collection together, as follows:

```
STUDENT (STUDENT_NUM, STUDENT_NAME, NUM_CREDITS,
     ADVISOR_NUM)
ADVISOR (ADVISOR_NUM, ADVISOR_NAME)
COURSE (COURSE_NUM, DESCRIPTION)
STUDENT_COURSE (STUDENT_NUM, COURSE_NUM, GRADE)
```

## DIAGRAMS FOR DATABASE DESIGN

For many people, an illustration of a database's structure is quite useful. A popular type of illustration used to represent the structure of a database is the **entity-relationship (E-R) diagram**. In an E-R diagram, a rectangle represents an entity (table). One-to-many relationships between entities are drawn as lines between the corresponding rectangles.

Several different styles of E-R diagrams are used to diagram a database design. In the version shown in Figure 2-13, an arrowhead indicates the "many" side of the relationship between tables. In the relationship between the REP and CUSTOMER tables, for example, the arrow points from the REP table to the CUSTOMER table, indicating that one sales rep is related to many customers. The ORDER_LINE table has two one-to-many relationships, as indicated by the line from the ORDERS table to the ORDER_LINE table and the line from the PART table to the ORDER_LINE table.

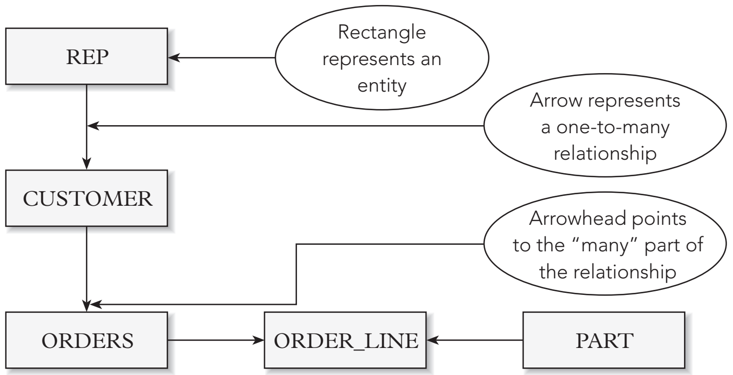

REP
Rectangle represents an entity
Arrow represents a one-to-many relationship
CUSTOMER
Arrowhead points to the "many" part of the relationship
ORDERS
ORDER_LINE
PART

**FIGURE 2-13** E-R diagram for the Premiere Products database with rectangles and arrows

**NOTE**

In this style of E-R diagram, you can put the rectangles in any position to represent the entities and relationships. The important thing is that the arrows connect the appropriate rectangles.

Another style of E-R diagram is to represent the "many" side of a relationship between tables with a crow's foot, as shown in Figure 2-14.

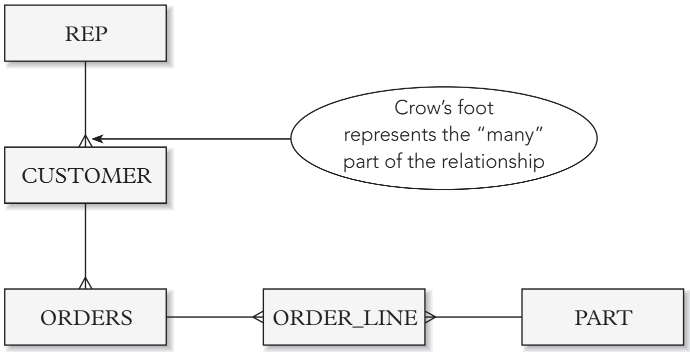


**FIGURE 2-14** E-R diagram for the Premiere Products database with a crow's foot

The E-R diagram shown in Figure 2-15 represents the original style of E-R diagrams. In this style, relationships are indicated in diamonds that describe the relationship. The relationship between the REP and CUSTOMER tables, for example, is named REPRESENTS, reflecting the fact that a sales rep represents a customer. The relationship between the CUSTOMER and ORDERS table is named PLACED, reflecting the fact that customers place orders. The relationship between the ORDERS and ORDER_LINE tables is named CONTAINS, reflecting the fact that an order contains order lines. The relationship between the PART and ORDER_LINE tables is named IS_ON, reflecting the fact that a given part is on many orders. In this style of E-R diagram, the number 1 indicates the "one" side of the relationship and the letter "n" represents the "many" side of the relationship.

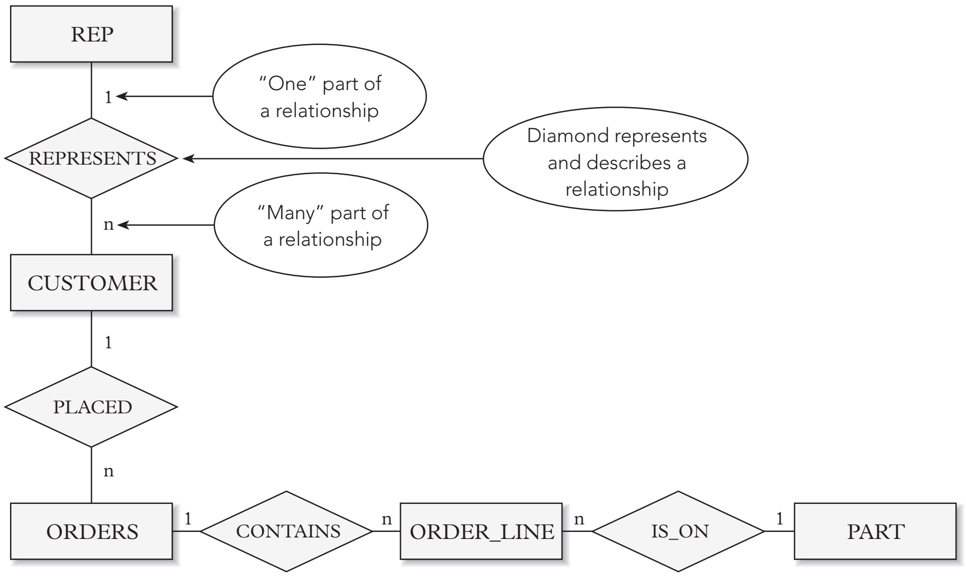

REP
1
"One" part of a relationship
REPRESENTS
Diamond represents and describes a relationship
n
"Many" part of a relationship
CUSTOMER
1
PLACED
n
ORDERS
1
CONTAINS
n
ORDER_LINE
n
IS_ON
1
PART

**FIGURE 2-15** E-R diagram for the Premiere Products database with named relationships

## Chapter Summary

- An entity is a person, place, thing, or event. An attribute is a property of an entity. A relationship is an association between entities.
- A relation is a two-dimensional table in which the entries in the table contain only single values, each column has a distinct name, all values in a column match this name, the order of the rows and columns is immaterial, and each row contains unique values. A relational database is a collection of relations.
- Column B is functionally dependent on another column, A (or possibly a collection of columns), when a value for A determines a single value for B at any one time.
- Column A (or a collection of columns) is the primary key for a relation (table), R, if *all* columns in R are functionally dependent on A and no subcollection of the columns in A (assuming A is a collection of columns and not just a single column) also has property 1.
- To design a database to satisfy a particular set of requirements, first read through the requirements and identify the entities (objects) involved. Give names to the entities and identify the unique identifiers for these entities. Next, identify the attributes for all the entities and the functional dependencies that exist among the attributes, and then use the functional dependencies to identify the tables and columns. Finally, identify any relationships between tables by looking at matching columns.
- A table (relation) is in first normal form (1NF) when it does not contain a repeating group. To convert an unnormalized table to first normal form, remove the repeating group and expand the primary key to include the original primary key along with the key to the repeating group.
- A table (relation) is in second normal form (2NF) when it is in first normal form and no nonkey column (that is, a column that is not part of the primary key) is dependent on only a portion of the primary key. To convert a table in first normal form to a collection of tables in second normal form, take each subset of the set of columns that make up the primary key, and begin a new table with this subset as its primary key. Next, place each of the other columns with the appropriate primary key; that is, place each one with the minimal collection of columns on which it depends. Finally, give each of these new tables a name that is descriptive of the meaning and contents of the table.
- A table is in third normal form (3NF) when it is in second normal form and the only determinants (columns on which at least one other column depends) it contains are candidate keys (columns that could function as the primary key). To convert a table in second normal form to a collection of tables in third normal form, first, for each determinant that is not a candidate key, remove from the table the columns that depend on this determinant (but don't remove the determinant). Next, create a new table containing all the columns from the original table that depend on this determinant. Finally, make the determinant the primary key of this new table.
- An entity-relationship (E-R) diagram is an illustration that represents the design of a database. There are several common styles of illustrating database design that use shapes to represent entities and connectors to illustrate the relationships between those entities.

## Key Terms

attribute
Boyce-Codd normal form (BCNF)
candidate key
concatenation
database design
determinant
entity
entity-relationship (E-R) diagram
field
first normal form (1NF)
functionally dependent
functionally determine
nonkey column
normal form
normalization
one-to-many relationship
primary key
qualify
record
redundancy
relation
relational database
relationship
repeating group
second normal form (2NF)
third normal form (3NF)
tuple
unnormalized relation
update anomaly

## Review Questions

1. What is an entity?
2. What is an attribute?
3. What is a relationship? What is a one-to-many relationship?
4. What is a repeating group?
5. What is a relation?
6. What is a relational database?
7. Describe the shorthand representation of the structure of a relational database. Illustrate this technique by representing the database for Henry Books as shown in Figures 1-4 through 1-7 in Chapter 1.
8. How do you qualify the name of a field, and when do you need to do this?
9. What does it mean for a column to be functionally dependent on another column?
10. What is a primary key? What is the primary key for each of the tables in the Henry Books database shown in Chapter 1?
11. A database at a college must support the following requirements:
    a. For a department, store its number and name.
    b. For an advisor, store his or her number, last name, first name, and the department number to which the advisor is assigned.
    c. For a course, store its code and description (for example, MTH110, Algebra).
    d. For a student, store his or her number, first name, and last name. For each course the student takes, store the course code, the course description, and the grade earned.

Also, store the number and name of the student's advisor. Assume that an advisor might advise any number of students but that each student has just one advisor.

Design the database for the preceding set of requirements. Use your own experience as a student to determine any functional dependencies. List the tables, columns, and relationships. In addition, represent your design with an E-R diagram.

12. Define first normal form.
13. Define second normal form. What types of problems might you encounter using tables that are not in second normal form?
14. Define third normal form. What types of problems might you encounter using tables that are not in third normal form?
15. Using the functional dependencies you determined in Question 11, convert the following table to an equivalent collection of tables that are in third normal form.

```
STUDENT (STUDENT_NUM, STUDENT_LAST_NAME, STUDENT_FIRST_NAME,
         ADVISOR_NUM, ADVISOR_LAST_NAME, ADVISOR_FIRST_NAME,
         (COURSE_CODE, DESCRIPTION, GRADE) )
```

## Exercises

### Premiere Products

Answer each of the following questions using the Premiere Products data shown in Figure 2-1. No computer work is required.

1. Indicate the changes (using the shorthand representation) that you would need to make to the original Premiere Products database design (see Figure 2-1) to support the following requirements. A customer is not necessarily represented by a single sales rep, but can be represented by several sales reps. When a customer places an order, the sales rep who gets the commission on the order must be in the collection of sales reps who represent the customer.
2. Indicate the changes (using the shorthand representation) that you would need to make to the original Premiere Products database design to support the following requirements. There is no relationship between customers and sales reps. When a customer places an order, any sales rep can process the order. On the order, you need to identify both the customer placing the order and the sales rep responsible for the order. Draw an E-R diagram for the new design.
3. Indicate the changes (using the shorthand representation) that you would need to make to the original Premiere Products database design in the event that the original Requirement 3 is changed as follows. For a part, store the part's number, description, item class, and price. In addition, for each warehouse in which the part is located, store the number of the warehouse, the description of the warehouse, and the number of units of the part stored in the warehouse. Draw an E-R diagram for the new design.

4. Using your knowledge of Premiere Products, determine the functional dependencies that exist in the following table. After determining the functional dependencies, convert this table to an equivalent collection of tables that are in third normal form.

```
PART (PART_NUM, DESCRIPTION, ON_HAND, CLASS, WAREHOUSE,
      PRICE, (ORDER_NUM, ORDER_DATE, CUSTOMER_NUM,
      CUSTOMER_NAME, NUM_ORDERED, QUOTED_PRICE) )
```

## Henry Books

Answer each of the following questions using the Henry Books data shown in Figures 1-4 through 1-7 in Chapter 1. No computer work is required.

1. Ray Henry is considering expanding the activities at his book stores to include movies. He has some ideas for how he wants to do this and he needs you to help with database design activities to address these ideas. In particular, he would like you to design a database for him. He is interested in movies and wants to store information about movies, stars, and directors in a database. He needs to be able to satisfy the following requirements:
   a. For each director, list his or her number, name, the year he or she was born, and the year of death if he or she is deceased.
   b. For each movie, list its number, title, the year the movie was made, and its type.
   c. For each movie, list its number, title, the number and name of its director, the critics' rating, the MPAA rating, the number of awards for which the movie was nominated, and the number of awards the movie won.
   d. For each movie star, list his or her number, name, birthplace, the year he or she was born, and the year of death if he or she is deceased.
   e. For each movie, list its number and title, along with the number and name of all the stars who appear in it.
   f. For each movie star, list his or her number and name, along with the number and name of all the movies in which he or she stars.

   List the tables, columns, and relationships. In addition, represent your design with an E-R diagram.
2. Determine the functional dependencies that exist in the following table, and then convert this table to an equivalent collection of tables that are in third normal form.

```
BOOK (BOOK_CODE, TITLE, TYPE, PRICE (AUTHOR_NUM,
     AUTHOR_LAST, AUTHOR_FIRST) )
```

3. Determine the functional dependencies that exist in the following table, and then convert this table to an equivalent collection of tables that are in third normal form.

```
BOOK (BOOK_CODE, TITLE, TYPE, PRICE, PUB_CODE,
     PUBLISHER_NAME, CITY)
```

## Alexamara Marina Group

Answer each of the following questions using the Alexamara Marina Group data shown in Figures 1-8 through 1-12 in Chapter 1. No computer work is required.

1. Design a database that can satisfy the following requirements:
   a. For each marina, list the number, name, address, city, state, and zip code.

b. For each boat owner, list the number, last name, first name, address, city, state, and zip code.

c. For each marina, list all the slips in the marina. For each slip, list the length of the slip, annual rental fee, name and type of the boat occupying the slip, and boat owner's number, last name, and first name.

d. For each possible service category, list the category number and description. In addition, for each service request in a category, list the marina number and slip number for the boat receiving the service, estimated hours for the service, hours already spent on the service, and next date that is scheduled for the particular service.

e. For each service request, list the marina number, slip number, category description, description of the particular service, and a description of the current status of the service.

List the tables, columns, and relationships. In addition, represent your design with an E-R diagram.

2. Determine the functional dependencies that exist in the following table, and then convert this table to an equivalent collection of tables that are in third normal form.

```
MARINA (MARINA_NUM, NAME, (SLIP_NUM, LENGTH, RENTAL_FEE,
        BOAT_NAME) )
```

3. Determine the functional dependencies that exist in the following table, and then convert this table to an equivalent collection of tables that are in third normal form.

```
MARINA_SLIP (SLIP_ID, MARINA_NUM, SLIP_NUM, LENGTH, RENTAL_FEE,
        BOAT_NAME, BOAT_TYPE, OWNER_NUM, LAST_NAME,
        FIRST_NAME)
```

APPENDIX C

# ANSWERS TO ODD-NUMBERED REVIEW QUESTIONS

This page contains answers for this chapter only.

## CHAPTER 2—DATABASE DESIGN FUNDAMENTALS

1. An entity is a person, place, thing, or event.
3. A relationship is an association between tables (entities). A one-to-many relationship between two tables is a relationship in which each row in the first table can be associated with many rows in the second table, but each row in the second table is associated with only one row in the first table.
5. A relation is a two-dimensional table in which the entries in the table are single-valued (each location in the table contains a single entry), each column has a distinct name (or attribute name), all values in a column match this name, the order of the rows and columns is immaterial, and each row contains unique values.
7. For each table, you write the name of the table and then within parentheses list all of the columns in the table. Underline the primary keys.

```
BRANCH (BRANCH_NUM, BRANCH_NAME, BRANCH_LOCATION,
       NUM_EMPLOYEES)
PUBLISHER (PUBLISHER_CODE, PUBLISHER_NAME, CITY)
AUTHOR (AUTHOR_NUM, AUTHOR_LAST, AUTHOR_FIRST)
BOOK (BOOK_CODE, TITLE, PUBLISHER_CODE, TYPE, PRICE,
       PAPERBACK)
WROTE (BOOK_CODE, AUTHOR_NUM, SEQUENCE)
INVENTORY (BOOK_CODE, BRANCH_NUM, ON_HAND)
```

9. A column (attribute), B, is functionally dependent on another column (or a collection of columns), A, if at any point in time a value for A determines a single value for B.

11. Functional dependencies:

```
DEPARTMENT_NUM → DEPARTMENT_NAME
ADVISOR_NUM → ADVISOR_LAST_NAME, ADVISOR_FIRST_NAME,
       DEPARTMENT_NUM
COURSE_CODE → DESCRIPTION
STUDENT_NUM → STUDENT_LAST_NAME, STUDENT_FIRST_NAME,
       ADVISOR_NUM
STUDENT_NUM, COURSE_CODE → GRADE
```

Relations:

```
DEPARTMENT (DEPARTMENT_NUM, DEPARTMENT_NAME)
ADVISOR (ADVISOR_NUM, ADVISOR_LAST_NAME, ADVISOR_FIRST_NAME,
       DEPARTMENT_NUM)
COURSE (COURSE_CODE, DESCRIPTION)
STUDENT (STUDENT_NUM, STUDENT_LAST_NAME, STUDENT_FIRST_NAME,
       ADVISOR_NUM)
STUDENT_COURSE (STUDENT_NUM, COURSE_CODE, GRADE)
```

Entity-relationship diagram: (*Note:* Your rectangles can be in different positions as long as they are connected by the same arrows.)

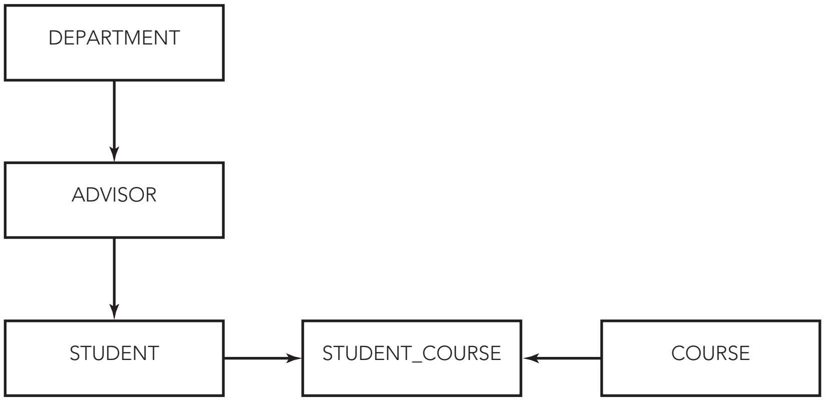


**FIGURE C-1**

13. A table (relation) is in second normal form when it is in first normal form and no nonkey column is dependent on only a portion of the primary key. When a table is not in second normal form, the table contains redundancy, which leads to a variety of update anomalies. A change in a value can require not just one change, but several. There is the possibility of inconsistent data. Adding additional data to the database might not be possible without creating artificial values for part of the key. Finally, deletions of certain items can result in inadvertently deleting crucial information from the database.

15.

```
STUDENT (STUDENT_NUM, STUDENT_LAST_NAME, STUDENT_FIRST_NAME,
       ADVISOR_NUM)
ADVISOR (ADVISOR_NUM, ADVISOR_LAST_NAME, ADVISOR_FIRST_NAME)
COURSE (COURSE_CODE, DESCRIPTION)
STUDENT_COURSE (STUDENT_NUM, COURSE_CODE, GRADE)
```