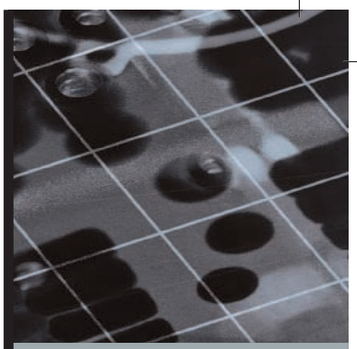


PART

II

Design Concepts

Database Modeling Supporting Communities

Business Vignette

Companies, governments, and organizations around the world turn to entity relationship diagrams and database modeling tools to help develop their databases. The advantages of using tools like Sybase PowerDesigner, Microsoft Visio Professional, ERwin Data Modeler, or Embarcadero ER/Studio significantly outweigh their expense. They improve database documentation. They facilitate staff communication, helping to ensure that the database will meet the needs of its users. They reduce development time. All these advantages translate into significant cost-savings. Yet sometimes this value goes well beyond anything that can be expressed in a dollar amount.

Rebuilding Together is a national nonprofit organization dedicated to preserving and revitalizing houses and communities for the elderly, disabled, and families with children. The national headquarters currently works with 255 affiliates serving over 1,897 communities. Based on the "barn-raising" tradition, local volunteers assemble on Rebuilding Day to help their neighbors. Over 267,000 volunteers have repaired or reconstructed approximately 9,000 houses and nonprofit facilities.

As the local affiliate in Des Moines, Iowa, founded in 1994, has grown rapidly, the organization has sought to document and improve their house selection and volunteer coordination processes. Several sources, including past participants, make referrals for potential housing projects. Each year, Rebuilding Together needs to evaluate the qualifications of each candidate, preview the site, select or reject the project, and finally implement the selected projects. Using modeling software ER/Studio, the staff built a database to keep track of these stages of the project and manage the volunteers that will work on each project.

By using the logical view of the data modeling software, the staff was able to understand the entities, their attributes, and the relationships that they were modeling prior to building the physical model. They also generated a short report and model diagram to educate all personnel involved in the project. The end result was that the company was able to develop an application process that is both more complex and user-friendly. As the organization continues to grow and the spirit of "barn-raising" spreads, the staff will be able to modify the design to accommodate its growing needs.

THREE

3 THE RELATIONAL DATABASE MODEL

In this chapter, you will learn:

- That the relational database model offers a logical view of data
- About the relational model's basic component: relations
- That relations are logical constructs composed of rows (tuples) and columns (attributes)
- That relations are implemented as tables in a relational DBMS
- About relational database operators, the data dictionary, and the system catalog
- How data redundancy is handled in the relational database model
- Why indexing is important

Preview

In Chapter 2, Data Models, you learned that the relational data model's structural and data independence allow you to examine the model's logical structure without considering the physical aspects of data storage and retrieval. You also learned that entity relationship diagrams (ERDs) may be used to depict entities and their relationships graphically. In this chapter, you learn some important details about the relational model's logical structure and more about how the ERD can be used to design a relational database.

You learn how the relational database's basic data components fit into a logical construct known as a table. You discover that one important reason for the relational database model's simplicity is that its tables can be treated as logical rather than physical units. You also learn how the independent tables within the database can be related to one another.

After learning about tables, their components, and their relationships, you are introduced to the basic concepts that shape the design of tables. Because the table is such an integral part of relational database design, you also learn the characteristics of well-designed and poorly designed tables.

Finally, you are introduced to some basic concepts that will become your gateway to the next few chapters. For example, you examine different kinds of relationships and the way those relationships might be handled in the relational database environment.

NOTE

The relational model, introduced by E. F. Codd in 1970, is based on predicate logic and set theory. **Predicate logic**, used extensively in mathematics, provides a framework in which an assertion (statement of fact) can be verified as either true or false. For example, suppose that a student with a student ID of 12345678 is named Melissa Sanduski. This assertion can easily be demonstrated to be true or false. **Set theory** is a mathematical science that deals with sets, or groups of things, and is used as the basis for data manipulation in the relational model. For example, assume that set A contains three numbers: 16, 24, and 77. This set is represented as A(16, 24, 77). Furthermore, set B contains four numbers: 44, 77, 90, and 11, and so is represented as B(44, 77, 90, 11). Given this information, you can conclude that the intersection of A and B yields a result set with a single number, 77. This result can be expressed as $A \cap B = 77$. In other words, A and B share a common value, 77.

Based on these concepts, the relational model has three well-defined components:

1. A logical data structure represented by relations (Sections 3.1, 3.2, and 3.5).
2. A set of integrity rules to enforce that the data are and remain consistent over time (Sections 3.3, 3.6, 3.7, and 3.8).
3. A set of operations that define how data are manipulated (Section 3.4).

3.1 A LOGICAL VIEW OF DATA

In Chapter 1, Database Systems, you learned that a database stores and manages both data and metadata. You also learned that the DBMS manages and controls access to the data and the database structure. Such an arrangement—placing the DBMS between the application and the database—eliminates most of the file system's inherent limitations. The result of such flexibility, however, is a far more complex physical structure. In fact, the database structures required by both the hierarchical and network database models often become complicated enough to diminish efficient database design. The relational data model changed all of that by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details. To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gearshifts. In short, the relational model enables you to view data *logically* rather than *physically*.

The practical significance of taking the logical view is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a table does resemble a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models. Logical simplicity tends to yield simple and effective database design methodologies.

Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins with an exploration of the details of table structure and contents.

3.1.1 TABLES AND THEIR CHARACTERISTICS

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end-users find it much easier to think of a relation as a table. A table is perceived as a two-dimensional structure composed of rows and columns. A table is also called a *relation* because the relational model's creator, E. F. Codd, used the term *relation* as a synonym for table. You can think of a table as a *persistent* representation of a logical relation, that is, a relation whose contents can be permanently saved for future use. As far as the table's user is concerned, *a table contains a group of related entity occurrences*, that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms *entity set* and *table* are often used interchangeably.

NOTE

The word *relation*, also known as a *dataset* in Microsoft Access, is based on the mathematical set theory from which Codd derived his model. Because the relational model uses attribute values to establish relationships among tables, many database users incorrectly assume that the term *relation* refers to such relationships. Many then incorrectly conclude that only the relational model permits the use of relationships.

You will discover that the table view of data makes it easy to spot and define entity relationships, thereby greatly simplifying the task of database design. The characteristics of a relational table are summarized in Table 3.1.

TABLE 3.1 Characteristics of a Relational Table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each row/column intersection represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or a combination of attributes that uniquely identifies each row.

The tables shown in Figure 3.1 illustrate the characteristics listed in Table 3.1.

NOTE

Relational database terminology is very precise. Unfortunately, file system terminology sometimes creeps into the database environment. Thus, rows are sometimes referred to as *records* and columns are sometimes labeled as *fields*. Occasionally, tables are labeled *files*. Technically speaking, this substitution of terms is not always appropriate; the database table is a logical rather than a physical concept, and the terms *file, record,* and *field* describe physical concepts. Nevertheless, as long as you recognize that the table is actually a logical rather than a physical construct, you may (at the conceptual level) think of table rows as records and of table columns as fields. In fact, many database software vendors still use this familiar file system terminology.

ONLINE CONTENT

All of the databases used to illustrate the material in this chapter are found in the Student Online Companion for this book. The database names used in the folder match the database names used in the figures. For example, the source of the tables shown in Figure 3.1 is the **Ch03_TinyCollege** database.

Using the STUDENT table shown in Figure 3.1, you can draw the following conclusions corresponding to the points in Table 3.1:

1. The STUDENT table is perceived to be a two-dimensional structure composed of eight rows (tuples) and twelve columns (attributes).
2. Each row in the STUDENT table describes a single entity occurrence within the entity set. (The entity set is represented by the STUDENT table.) Note that the row (entity or record) defined by STU_NUM = 321452 defines the characteristics (attributes or fields) of a student named William C. Bowser. For example, row 4 in Figure 3.1 describes a student named Walter H. Oblonski. Similarly, row 3 describes a student named Juliette Brewer. Given the table contents, the STUDENT entity set includes eight distinct entities (rows), or students.

FIGURE 3.1 **STUDENT table attribute values**

Database name: Ch03_TinyCollege

Table name: STUDENT

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS
321452	Bowser	William	C	12-Feb-1975	42	So
324257	Smithson	Anne	K	15-Nov-1981	81	Jr
324258	Brewer	Juliette		23-Aug-1969	36	So
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr
324273	Smith	John	D	30-Dec-1958	102	Sr
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr
324299	Smith	John	B	30-Nov-1986	15	Fr

STUDENT table, continued

STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
2.84	No	BIOL	2134	205
3.27	Yes	CIS	2256	222
2.26	Yes	ACCT	2256	228
3.09	No	CIS	2114	222
2.11	Yes	ENGL	2231	199
3.15	No	ACCT	2267	228
3.87	No	EDU	2267	311
2.92	No	ACCT	2315	230

STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_DOB = Student date of birth
STU_GPA = Grade point average
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

3. Each column represents an attribute, and each column has a distinct name.
4. All of the values in a column match the attribute's characteristics. For example, the grade point average (STU_GPA) column contains only STU_GPA entries for each of the table rows. Data must be classified according to their format and function. Although various DBMSs can support different data types, most support at least the following:

   a. *Numeric.* Numeric data are data on which you can perform meaningful arithmetic procedures. For example, STU_HRS and STU_GPA in Figure 3.1 are numeric attributes. On the other hand, STU_PHONE is not a numeric attribute because adding or subtracting phone numbers does not yield an arithmetically meaningful result.

   b. *Character.* Character data, also known as text data or string data, can contain any character or symbol not intended for mathematical manipulation. In Figure 3.1, for example, STU_LNAME, STU_FNAME, STU_INIT, STU_CLASS, and STU_PHONE are character attributes.

   c. *Date.* Date attributes contain calendar dates stored in a special format known as the Julian date format. Although the physical storage of the Julian date is immaterial to the user and designer, the Julian date format allows you to perform a special kind of arithmetic known as Julian date arithmetic. Using Julian date arithmetic, you can determine the number of days that have elapsed between two dates, such as 12-May-1999 and 20-Mar-2008, by simply subtracting 12-May-1999 from 20-Mar-2008. In Figure 3.1, STU_DOB can properly be classified as a date attribute. Most relational database software packages support Julian date formats. While the database's internal date format is likely to be Julian, many different *presentation* formats are available. For example, in Figure 3.1, you could show Mr. Bowser's date of birth (STU_DOB) as 2/12/75. Most relational DBMSs allow you to define your own date presentation format. For instance, Access and Oracle users might specify the "dd-mmm-yyyy" date format to show the first STU_DOB value in Figure 3.1 as 12-Feb-1975. (As you can tell by examining the STU_DOB values in Figure 3.1, the "dd-mmm-yyyy" format was selected to present the output.)

d. *Logical*. Logical data can have only a true or false (yes or no) condition. For example, is a student a junior college transfer? In Figure 3.1, the STU_TRANSFER attribute uses a logical data format. Most, but not all, relational database software packages support the logical data format. (Microsoft Access uses the label "Yes/No data type" to indicate a logical data type.)

5. The column's range of permissible values is known as its **domain**. Because the STU_GPA values are limited to the range 0–4, inclusive, the domain is [0,4].
6. The order of rows and columns is immaterial to the user.
7. Each table must have a primary key. In general terms, the **primary key** (**PK**) is an attribute (or a combination of attributes) that uniquely identifies any given row. In this case, STU_NUM (the student number) is the primary key. Using the data presented in Figure 3.1, observe that a student's last name (STU_LNAME) would not be a good primary key because it is possible to find several students whose last name is Smith. Even the combination of the last name and first name (STU_FNAME) would not be an appropriate primary key because, as Figure 3.1 shows, it is quite possible to find more than one student named John Smith.

3.2 KEYS

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. Therefore, a proper understanding of the concept and use of keys in the relational model is very important. A **key** consists of one or more attributes that determine other attributes. For example, an invoice number identifies all of the invoice attributes, such as the invoice date and the customer name.

One type of key, the primary key, has already been introduced. Given the structure of the STUDENT table shown in Figure 3.1, defining and describing the primary key seems simple enough. However, because the primary key plays such an important role in the relational environment, you will examine the primary key's properties more carefully. In this section, you also will become acquainted with superkeys, candidate keys, and secondary keys.

The key's role is based on a concept known as **determination**. In the context of a database table, the statement "A determines B" indicates that if you know the value of attribute A, you can look up (determine) the value of attribute B. For example, knowing the STU_NUM in the STUDENT table (see Figure 3.1) means that you are able to look up (determine) that student's last name, grade point average, phone number, and so on. The shorthand notation for "A determines B" is A → B. If A determines B, C, and D, you write A → B, C, D. Therefore, using the attributes of the STUDENT table in Figure 3.1, you can represent the statement "STU_NUM determines STU_LNAME" by writing:

STU_NUM → STU_LNAME

In fact, the STU_NUM value in the STUDENT table determines all of the student's attribute values. For example, you can write:

STU_NUM → STU_LNAME, STU_FNAME, STU_INIT

and

STU_NUM → STU_LNAME, STU_FNAME, STU_INIT, STU_DOB, STU_TRANSFER

In contrast, STU_NUM is not determined by STU_LNAME because it is quite possible for several students to have the last name Smith.

The principle of determination is very important because it is used in the definition of a central relational database concept known as functional dependence. The term **functional dependence** can be defined most easily this way: the attribute B is functionally dependent on A if A determines B. More precisely:

The attribute B is functionally dependent on the attribute A if each value in column A determines *one and only one* value in column B.

Using the contents of the STUDENT table in Figure 3.1, it is appropriate to say that STU_PHONE is functionally dependent on STU_NUM. For example, the STU_NUM value 321452 determines the STU_PHONE value 2134. On the other hand, STU_NUM is not functionally dependent on STU_PHONE because the STU_PHONE value 2267 is associated with two STU_NUM values: 324274 and 324291. (This could happen in a dormitory situation, where students share a phone.) Similarly, the STU_NUM value 324273 determines the STU_LNAME value Smith. But the STU_NUM value is not functionally dependent on STU_LNAME because more than one student may have the last name Smith.

The functional dependence definition can be generalized to cover the case in which the determining attribute values occur more than once in a table. Functional dependence can then be defined this way:[1]

Attribute A determines attribute B (that is, B is functionally dependent on A) if all of the rows in the table that agree in value for attribute A also agree in value for attribute B.

Be careful when defining the dependency's direction. For example, Gigantic State University determines its student classification based on hours completed; these are shown in Table 3.2.

TABLE 3.2 Student Classification

HOURS COMPLETED	CLASSIFICATION
Less than 30	Fr
30–59	So
60–89	Jr
90 or more	Sr

Therefore, you can write:

STU_HRS → STU_CLASS

But the specific number of hours is not dependent on the classification. It is quite possible to find a junior with 62 completed hours or one with 84 completed hours. In other words, the classification (STU_CLASS) does not determine one and only one value for completed hours (STU_HRS).

Keep in mind that it might take more than a single attribute to define functional dependence; that is, a key may be composed of more than one attribute. Such a multi-attribute key is known as a **composite key**.

Any attribute that is part of a key is known as a **key attribute**. For instance, in the STUDENT table, the student's last name would not be sufficient to serve as a key. On the other hand, the combination of last name, first name, initial, and home phone is very likely to produce unique matches for the remaining attributes. For example, you can write:

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS

or

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS, STU_GPA

or

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS, STU_GPA, STU_DOB

[1] *SQL:2003 ANSI standard specification*. ISO/IEC 9075-2:2003 - SQL/Foundation.

Given the possible existence of a composite key, the notion of functional dependence can be further refined by specifying **full functional dependence**:

If the attribute (B) is functionally dependent on a composite key (A) but not on any subset of that composite key, the attribute (B) is fully functionally dependent on (A).

Within the broad key classification, several specialized keys can be defined. For example, a **superkey** is any key that uniquely identifies each row. In short, the superkey functionally determines all of a row's attributes. In the STUDENT table, the superkey could be any of the following:

STU_NUM

STU_NUM, STU_LNAME

STU_NUM, STU_LNAME, STU_INIT

In fact, STU_NUM, with or without additional attributes, can be a superkey even when the additional attributes are redundant.

A **candidate key** can be described as a superkey without unnecessary attributes, that is, a minimal superkey. Using this distinction, note that the composite key

STU_NUM, STU_LNAME

is a superkey, but it is not a candidate key because STU_NUM by itself is a candidate key! The combination

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE

might also be a candidate key, as long as you discount the possibility that two students share the same last name, first name, initial, and phone number.

If the student's Social Security number had been included as one of the attributes in the STUDENT table in Figure 3.1—perhaps named STU_SSN—both it and STU_NUM would have been candidate keys because either one would uniquely identify each student. In that case, the selection of STU_NUM as the primary key would be driven by the designer's choice or by end-user requirements. In short, the primary key is the candidate key chosen to be the unique row identifier. Note, incidentally, that a primary key is a superkey as well as a candidate key.

Within a table, each primary key value must be unique to ensure that each row is uniquely identified by the primary key. In that case, the table is said to exhibit **entity integrity**. To maintain entity integrity,a **null** (that is, no data entry at all) is not permitted in the primary key.

NOTE

A null is no value at all. It does *not* mean a zero or a space. A null is created when you press the Enter key or the Tab key to move to the next entry without making a prior entry of any kind. Pressing the Spacebar creates a blank (or a space).

Nulls can *never* be part of a primary key, and they should be avoided—to the greatest extent possible—in other attributes, too. There are rare cases in which nulls cannot be reasonably avoided when you are working with nonkey attributes. For example, one of an EMPLOYEE table's attributes is likely to be the EMP_INITIAL. However, some employees do not have a middle initial. Therefore, some of the EMP_INITIAL values may be null. You will also discover later in this section that there may be situations in which a null exists because of the nature of the relationship between two entities. In any case, even if nulls cannot always be avoided, they must be used sparingly. In fact, the existence of nulls in a table is often an indication of poor database design.

Nulls, if used improperly, can create problems because they have many different meanings. For example, a null can represent:

- An unknown attribute value.
- A known, but missing, attribute value.
- A "not applicable" condition.

Depending on the sophistication of the application development software, nulls can create problems when functions such as COUNT, AVERAGE, and SUM are used. In addition, nulls can create logical problems when relational tables are linked.

Controlled redundancy makes the relational database work. Tables within the database share common attributes that enable the tables to be linked together. For example, note that the PRODUCT and VENDOR tables in Figure 3.2 share a common attribute named VEND_CODE. And note that the PRODUCT table's VEND_CODE value 232 occurs more than once, as does the VEND_CODE value 235. Because the PRODUCT table is related to the VENDOR table through these VEND_CODE values, the multiple occurrence of the values is *required* to make the 1:M relationship between VENDOR and PRODUCT work. Each VEND_CODE value in the VENDOR table is unique—the VENDOR is the "1" side in the VENDOR-PRODUCT relationship. But any given VEND_CODE value from the VENDOR table may occur more than once in the PRODUCT table, thus providing evidence that PRODUCT is the "M" side of the VENDOR-PRODUCT relationship. In database terms, the multiple occurrences of the VEND_CODE values in the PRODUCT table are not redundant because they are *required* to make the relationship work. You should recall from Chapter 2 that data redundancy exists only when there is *unnecessary* duplication of attribute values.

FIGURE 3.2 **An example of a simple relational database**

Table name: PRODUCT
Primary key: PROD_CODE
Foreign key: VEND_CODE

Database name: Ch03_SaleCo

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

link

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

As you examine Figure 3.2, note that the VEND_CODE value in one table can be used to point to the corresponding value in the other table. For example, the VEND_CODE value 235 in the PRODUCT table points to vendor Henry Ortozo in the VENDOR table. Consequently, you discover that the product "Houselite chain saw, 16-in. bar" is delivered by Henry Ortozo and that he can be contacted by calling 615-899-3425. The same connection can be made for the product "Steel tape, 12-ft. length" in the PRODUCT table.

Remember the naming convention—the prefix PROD was used in Figure 3.2 to indicate that the attributes "belong" to the PRODUCT table. Therefore, the prefix VEND in the PRODUCT table's VEND_CODE indicates that

VEND_CODE points to some other table in the database. In this case, the VEND prefix is used to point to the VENDOR table in the database.

A relational database can also be represented by a relational schema. A **relational schema** is a textual representation of the database tables where each table is listed by its name followed by the list of its attributes in parentheses. The primary key attribute(s) is (are) underlined. You will see such schemas in Chapter 5, Normalization of Database Tables. For example, the relational schema for Figure 3.2 would be shown as:

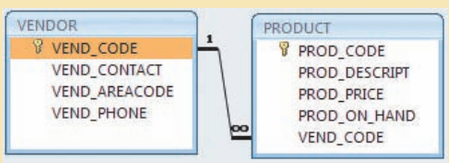
VENDOR (**VEND_CODE**, VEND_CONTACT, VEND_AREACODE, VEND_PHONE)

PRODUCT (**PROD_CODE**, PROD_DESCRIPT, PROD_PRICE, PROD_ON_HAND, VEND_CODE)

The link between the PRODUCT and VENDOR tables in Figure 3.2 can also be represented by the relational diagram shown in Figure 3.3. In this case, the link is indicated by the line that connects the VENDOR and PRODUCT tables.

Note that the link in Figure 3.3 is the equivalent of the relationship line in an ERD. This link is created when two tables share an attribute with common values. More specifically, the primary key of one table (VENDOR) appears as the *foreign key* in a related table (PRODUCT). A **foreign key** (**FK**) is an attribute whose values match the primary key values in the related table. For example, in Figure 3.2, the VEND_CODE is the primary key in the VENDOR table, and it occurs as a foreign key in the PRODUCT table. Because the VENDOR table is not linked to a third table, the VENDOR table shown in Figure 3.2 does not contain a foreign key.

FIGURE 3.3 The relational diagram for the Ch03_SaleCo database



If the foreign key contains either matching values or nulls, the table that makes use of that foreign key is said to exhibit *referential integrity*. In other words, **referential integrity** means that if the foreign key contains a value, that value refers to an existing valid tuple (row) in another relation. Note that referential integrity is maintained between the PRODUCT and VENDOR tables shown in Figure 3.2.

Finally, a **secondary key** is defined as a key that is used strictly for data retrieval purposes. Suppose customer data are stored in a CUSTOMER table in which the customer number is the primary key. Do you suppose that most customers will remember their numbers? Data retrieval for a customer can be facilitated when the customer's last name and phone number are used. In that case, the primary key is the customer number; the secondary key is the combination of the customer's last name and phone number. Keep in mind that a secondary key does not necessarily yield a unique outcome. For example, a customer's last name and home telephone number could easily yield several matches where one family lives together and shares a phone line. A less efficient secondary key would be the combination of the last name and zip code; this could yield dozens of matches, which could then be combed for a specific match.

A secondary key's effectiveness in narrowing down a search depends on how restrictive that secondary key is. For instance, although the secondary key CUS_CITY is legitimate from a database point of view, the attribute values "New York" or "Sydney" are not likely to produce a usable return unless you want to examine millions of possible matches. (Of course, CUS_CITY is a better secondary key than CUS_COUNTRY.)

Table 3.3 summarizes the various relational database table keys.

TABLE 3.3 Relational Database Keys

KEY TYPE	DEFINITION
Superkey	An attribute (or combination of attributes) that uniquely identifies each row in a table.
Candidate key	A minimal (irreducible) superkey. A superkey that does not contain a subset of attributes that is itself a superkey.
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.
Secondary key	An attribute (or combination of attributes) used strictly for data retrieval purposes.
Foreign key	An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null.

3.3 INTEGRITY RULES

Relational database integrity rules are very important to good database design. Many (but by no means all) RDBMSs enforce integrity rules automatically. However, it is much safer to make sure that your application design conforms to the entity and referential integrity rules mentioned in this chapter. Those rules are summarized in Table 3.4.

TABLE 3.4 Integrity Rules

ENTITY INTEGRITY	DESCRIPTION
Requirement	All primary key entries are unique, and no part of a primary key may be null.
Purpose	Each row will have a unique identity, and foreign key values can properly reference primary key values.
Example	No invoice can have a duplicate number, nor can it be null. In short, all invoices are uniquely identified by their invoice number.
REFERENTIAL INTEGRITY	**DESCRIPTION**
Requirement	A foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related. (Every non-null foreign key value *must* reference an *existing* primary key value.)
Purpose	It is possible for an attribute NOT to have a corresponding value, but it will be impossible to have an invalid entry. The enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.
Example	A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

The integrity rules summarized in Table 3.4 are illustrated in Figure 3.4.

Note the following features of Figure 3.4.

1. *Entity integrity*. The CUSTOMER table's primary key is CUS_CODE. The CUSTOMER primary key column has no null entries, and all entries are unique. Similarly, the AGENT table's primary key is AGENT_CODE, and this primary key column also is free of null entries.
2. *Referential integrity*. The CUSTOMER table contains a foreign key, AGENT_CODE, which links entries in the CUSTOMER table to the AGENT table. The CUS_CODE row that is identified by the (primary key) number 10013 contains a null entry in its AGENT_CODE foreign key because Mr. Paul F. Olowski does not yet have a sales representative assigned to him. The remaining AGENT_CODE entries in the CUSTOMER table all match the AGENT_CODE entries in the AGENT table.

FIGURE 3.4 An illustration of integrity rules

Table name: CUSTOMER **Database name: Ch03_InsureCo**
Primary key: CUS_CODE
Foreign key: AGENT_CODE

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_INSURE_TYPE	CUS_INSURE_AMT	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	615	844-2573	T1	100.00	05-Apr-2008	502
10011	Dunne	Leona	K	713	894-1238	T1	250.00	16-Jun-2008	501
10012	Smith	Kathy	W	615	894-2285	S2	150.00	29-Jan-2009	502
10013	Olowski	Paul	F	615	894-2180	S1	300.00	14-Oct-2008	502
10014	Orlando	Myron		615	222-1672	T1	100.00	28-Dec-2008	501
10015	O'Brian	Amy	B	713	442-3381	T2	850.00	22-Sep-2008	503
10016	Brown	James	G	615	297-1228	S1	120.00	25-Mar-2009	502
10017	Williams	George		615	290-2556	S1	250.00	17-Jul-2008	503
10018	Farriss	Anne	G	713	382-7185	T2	100.00	03-Dec-2008	501
10019	Smith	Olette	K	615	297-3809	S2	500.00	14-Mar-2009	503

Table name: AGENT
Primary key: AGENT_CODE
Foreign key: none

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS
501	713	228-1249	Alby	132735.75
502	615	882-1244	Hahn	138967.35
503	615	123-5589	Okon	127093.45

To avoid nulls, some designers use special codes, known as **flags**, to indicate the absence of some value. Using Figure 3.4 as an example, the code -99 could be used as the AGENT_CODE entry of the fourth row of the CUSTOMER table to indicate that customer Paul Olowski does not yet have an agent assigned to him. If such a flag is used, the AGENT table must contain a dummy row with an AGENT_CODE value of -99. Thus, the AGENT table's first record might contain the values shown in Table 3.5.

TABLE 3.5 A Dummy Variable Value Used as a Flag

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SALES
-99	000	000-0000	None	$0.00

Chapter 4, Entity Relationship (ER) Modeling, discusses several ways in which nulls may be handled.

Other integrity rules that can be enforced in the relational model are the *NOT NULL* and *UNIQUE* constraints. The NOT NULL constraint can be placed on a column to ensure that every row in the table has a value for that column. The UNIQUE constraint is a restriction placed on a column to ensure that no duplicate values exist for that column.

3.4 RELATIONAL SET OPERATORS

The data in relational tables are of limited value unless the data can be manipulated to generate useful information. This section describes the basic data manipulation capabilities of the relational model. **Relational algebra** defines the theoretical way of manipulating table contents using the eight relational operators: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. In Chapter 7, Introduction to Structured Query Language (SQL), you will learn how SQL commands can be used to accomplish relational algebra operations.

NOTE

The degree of relational completeness can be defined by the extent to which relational algebra is supported. To be considered minimally relational, the DBMS must support the key relational operators SELECT, PROJECT, and JOIN. Very few DBMSs are capable of supporting all eight relational operators.

The relational operators have the property of **closure**; that is, the use of relational algebra operators on existing tables (relations) produces new relations. There is no need to examine the mathematical definitions, properties, and characteristics of those relational algebra operators. However, their *use* can easily be illustrated as follows:

1. UNION combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics (the columns and domains must be identical) to be used in the UNION. When two or more tables share the same number of columns, when the columns have the same names, and when they share the same (or compatible) domains, they are said to be **union-compatible**. The effect of a UNION is shown in Figure 3.5.

FIGURE 3.5 UNION

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

UNION

P_CODE	P_DESCRIPT	PRICE
345678	Microwave	160.00
345679	Dishwasher	500.00

yields

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99
345678	Microwave	160
345679	Dishwasher	500

2. INTERSECT yields only the rows that appear in both tables. As was true in the case of UNION, the tables must be union-compatible to yield valid results. For example, you cannot use INTERSECT if one of the attributes is numeric and one is character-based. The effect of an INTERSECT is shown in Figure 3.6.

FIGURE 3.6 INTERSECT

F_NAME
George
Jane
Elaine
Wilfred
Jorge

INTERSECT

F_NAME
Jane
William
Jorge
Dennis

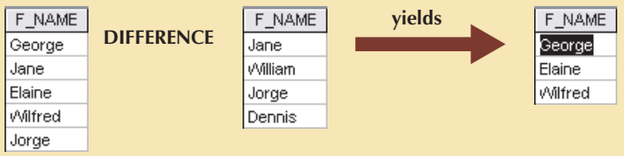
yields

F_NAME
Jane
Jorge

3. DIFFERENCE yields all rows in one table that are not found in the other table; that is, it subtracts one table from the other. As was true in the case of UNION, the tables must be union-compatible to yield valid results.

The effect of a DIFFERENCE is shown in Figure 3.7. However, note that subtracting the first table from the second table is not the same as subtracting the second table from the first table.

FIGURE 3.7 DIFFERENCE



4. PRODUCT yields all possible pairs of rows from two tables—also known as the Cartesian product. Therefore, if one table has six rows and the other table has three rows, the PRODUCT yields a list composed of 6 × 3 = 18 rows. The effect of a PRODUCT is shown in Figure 3.8.

FIGURE 3.8 PRODUCT

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PRODUCT

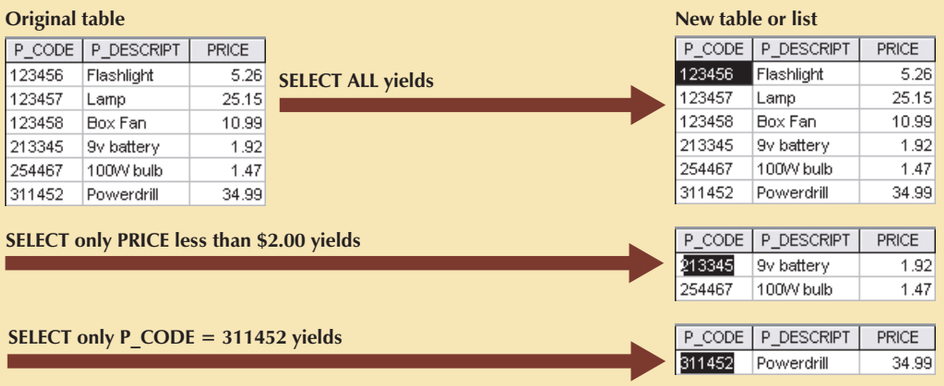
STORE	AISLE	SHELF
23	W	5
24	K	9
25	Z	6

yields

P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.26	23	W	5
123456	Flashlight	5.26	24	K	9
123456	Flashlight	5.26	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

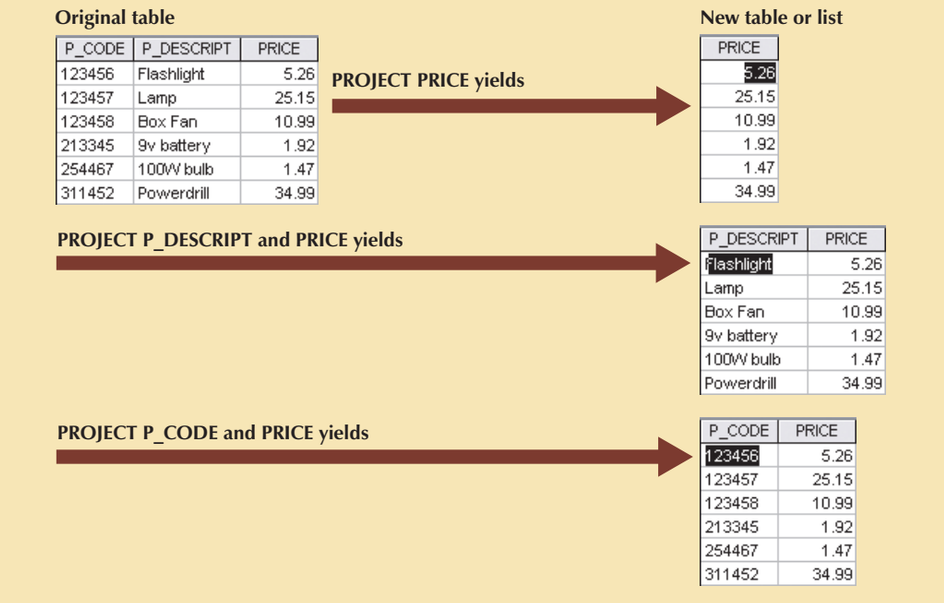
5. SELECT, also known as RESTRICT, yields values for all rows found in a table that satisfy a given condition. SELECT can be used to list all of the row values, or it can yield only those row values that match a specified criterion. In other words, SELECT yields a horizontal subset of a table. The effect of a SELECT is shown in Figure 3.9.

FIGURE 3.9 SELECT



6. PROJECT yields all values for selected attributes. In other words, PROJECT yields a vertical subset of a table. The effect of a PROJECT is shown in Figure 3.10.

FIGURE 3.10 PROJECT



7. JOIN allows information to be combined from two or more tables. JOIN is the real power behind the relational database, allowing the use of independent tables linked by common attributes. The CUSTOMER and AGENT tables shown in Figure 3.11 will be used to illustrate several types of joins.

FIGURE 3.11 **Two tables that will be used in join illustrations**

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

A **natural join** links tables by selecting only the rows with common values in their common attribute(s). A natural join is the result of a three-stage process:

a. First, a PRODUCT of the tables is created, yielding the results shown in Figure 3.12.

FIGURE 3.12 **Natural join, Step 1: PRODUCT**

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231	125	6152439887
1132445	Walker	32145	231	167	6153426778
1132445	Walker	32145	231	231	6152431124
1132445	Walker	32145	231	333	9041234445
1217782	Adares	32145	125	125	6152439887
1217782	Adares	32145	125	167	6153426778
1217782	Adares	32145	125	231	6152431124
1217782	Adares	32145	125	333	9041234445
1312243	Rakowski	34129	167	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1312243	Rakowski	34129	167	231	6152431124
1312243	Rakowski	34129	167	333	9041234445
1321242	Rodriguez	37134	125	125	6152439887
1321242	Rodriguez	37134	125	167	6153426778
1321242	Rodriguez	37134	125	231	6152431124
1321242	Rodriguez	37134	125	333	9041234445
1542311	Smithson	37134	421	125	6152439887
1542311	Smithson	37134	421	167	6153426778
1542311	Smithson	37134	421	231	6152431124
1542311	Smithson	37134	421	333	9041234445
1657399	Vanloo	32145	231	125	6152439887
1657399	Vanloo	32145	231	167	6153426778
1657399	Vanloo	32145	231	231	6152431124
1657399	Vanloo	32145	231	333	9041234445

b. Second, a SELECT is performed on the output of Step a to yield only the rows for which the AGENT_CODE values are equal. The common columns are referred to as the **join columns**. Step b yields the results shown in Figure 3.13.

FIGURE 3.13 Natural join, Step 2: SELECT

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124

c. A PROJECT is performed on the results of Step b to yield a single copy of each attribute, thereby eliminating duplicate columns. Step c yields the output shown in Figure 3.14.

FIGURE 3.14 Natural join, Step 3: PROJECT

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

The final outcome of a natural join yields a table that does not include unmatched pairs and provides only the copies of the matches.

Note a few crucial features of the natural join operation:

- If no match is made between the table rows, the new table does not include the unmatched row. In that case, neither AGENT_CODE 421 nor the customer whose last name is Smithson is included. Smithson's AGENT_CODE 421 does not match any entry in the AGENT table.
- The column on which the join was made—that is, AGENT_CODE—occurs only once in the new table.
- If the same AGENT_CODE were to occur several times in the AGENT table, a customer would be listed for each match. For example, if the AGENT_CODE 167 were to occur three times in the AGENT table, the customer named Rakowski, who is associated with AGENT_CODE 167, would occur three times in the resulting table. (A good AGENT table cannot, of course, yield such a result because it would contain unique primary key values.)

Another form of join, known as **equijoin**, links tables on the basis of an equality condition that compares specified columns of each table. The outcome of the equijoin does not eliminate duplicate columns, and the condition or criterion used to join the tables must be explicitly defined. The equijoin takes its name from the equality comparison operator (=) used in the condition. If any other comparison operator is used, the join is called a **theta join**.

In an **outer join**, the matched pairs would be retained and any unmatched values in the other table would be left null. More specifically, if an outer join is produced for tables CUSTOMER and AGENT, two scenarios are possible:

FIGURE 3.15 Left outer join

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124
1542311	Smithson	37134	421	

A **left outer join** yields all of the rows in the CUSTOMER table, including those that do not have a matching value in the AGENT table. An example of such a join is shown in Figure 3.15.

A **right outer join** yields all of the rows in the AGENT table, including those that do not have matching values in the CUSTOMER table. An example of such a join is shown in Figure 3.16.

FIGURE 3.16 Right outer join

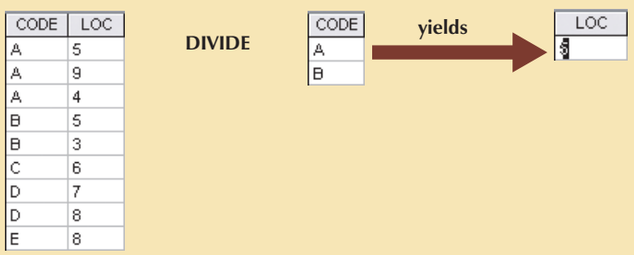
CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124
			333	9041234445

Outer joins are especially useful when you are trying to determine what value(s) in related tables cause(s) referential integrity problems. Such problems are created when foreign key values do not match the primary key values in the related table(s). In fact, if you are asked to convert large spreadsheets or other nondatabase data into relational database tables, you will discover that the outer joins save you vast amounts of time and uncounted headaches when you encounter referential integrity errors after the conversions.

You may wonder why the outer joins are labeled *left* and *right*. The labels refer to the order in which the tables are listed in the SQL command. Chapter 7 explores such joins.

8. The DIVIDE operation uses one single-column table (i.e. column "a") as the divisor and one 2-column table (i.e. columns "a" and "b") as the dividend. The tables must have a common column (i.e. column "a".) The output of the DIVIDE operation is a single column with the values of column "a" from the dividend table rows where the value of the common column (i.e. column "a") in both tables match. Figure 3.17 shows a DIVIDE.

FIGURE 3.17 DIVIDE



Using the example shown in Figure 3.17, note that:

a. Table 1 is "divided" by Table 2 to produce Table 3. Tables 1 and 2 both contain the column CODE but do not share LOC.

b. To be included in the resulting Table 3, a value in the unshared column (LOC) must be associated (in the dividing Table 2) with every value in Table 1.

c. The only value associated with both A and B is 5.

3.5 THE DATA DICTIONARY AND THE SYSTEM CATALOG

The **data dictionary** provides a detailed description of all tables found within the user/designer-created database. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata—data about data. Using the small database presented in Figure 3.4, you might picture its data dictionary as shown in Table 3.6.

TABLE 3.6 A Sample Data Dictionary

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE	REQUIRED	PK OR FK	FK REFERENCED TABLE
CUSTOMER	CUS_CODE	Customer account code	CHAR(5)	99999	10000–99999	Y	PK	
	CUS_LNAME	Customer last name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_FNAME	Customer first name	VARCHAR(20)	Xxxxxxxx		Y		
	CUS_INITIAL	Customer initial	CHAR(1)	X				
	CUS_RENEW_DATE	Customer insurance renewal date	DATE	dd-mmm-yyyy				
	AGENT_CODE	Agent code	CHAR(3)	999			FK	AGENT_CODE
AGENT	AGENT_CODE	Agent code	CHAR(3)	999		Y	PK	
	AGENT_AREACODE	Agent area code	CHAR(3)	999		Y		
	AGENT_PHONE	Agent telephone number	CHAR(8)	999-9999		Y		
	AGENT_LNAME	Agent last name	VARCHAR(20)	Xxxxxxxx		Y		
	AGENT_YTD_SLS	Agent year-to-date sales	NUMBER(9,2)	9,999,999.99		Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed character length data (1–255 characters)
VARCHAR = Variable character length data (1–2,000 characters)
NUMBER = Numeric data (NUMBER(9,2)) is used to specify numbers with two decimal places and up to nine digits, including the decimal places. Some RDBMSs permit the use of a MONEY or CURRENCY data type.

Note: Telephone area codes are always composed of digits 0–9. Because area codes are not used arithmetically, they are most efficiently stored as character data. Also, the area codes are always composed of three digits. Therefore, the area code data type is defined as CHAR(3). On the other hand, names do not conform to some standard length. Therefore, the customer first names are defined as VARCHAR(20), thus indicating that up to 20 characters may be used to store the names. Character data are shown as left-justified.

NOTE

The data dictionary in Table 3.6 is an example of the *human* view of the entities, attributes, and relationships. The purpose of this data dictionary is to ensure that all members of database design and implementation teams use the same table and attribute names and characteristics. The DBMS's internally stored data dictionary contains additional information about relationship types, entity and referential integrity checks and enforcement, and index types and components. This additional information is generated during the database implementation stage.

The data dictionary is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures.

Like the data dictionary, the **system catalog** contains metadata. The system catalog can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators, authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms *system catalog* and *data dictionary* are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer's data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table.

In effect, the system catalog automatically produces database documentation. As new tables are added to the database, that documentation also allows the RDBMS to check for and eliminate homonyms and synonyms. In general terms, **homonyms** are similar-sounding words with different meanings, such as *boar* and *bore*, or identically spelled words with different meanings, such as *fair* (meaning "just") and *fair* (meaning "festival"). In a database context, the word *homonym* indicates the use of the same attribute name to label different attributes. For example, you might use C_NAME to label a customer name attribute in a CUSTOMER table and also use C_NAME to label a consultant name attribute in a CONSULTANT table. To lessen confusion, you should avoid database homonyms; the data dictionary is very useful in this regard.

In a database context, a **synonym** is the opposite of a homonym and indicates the use of different names to describe the same attribute. For example, *car* and *auto* refer to the same object. Synonyms must be avoided. You will discover why using synonyms is a bad idea when you work through Problem 33 at the end of this chapter.

3.6 RELATIONSHIPS WITHIN THE RELATIONAL DATABASE

You already know that relationships are classified as one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N or M:M). This section explores those relationships further to help you apply them properly when you start developing database designs, focusing on the following points:

- The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relational database design.
- The 1:1 relationship should be rare in any relational database design.
- M:N relationships cannot be implemented as such in the relational model. Later in this section, you will see how any M:N relationships can be changed into two 1:M relationships.

3.6.1 THE 1:M RELATIONSHIP

The 1:M relationship is the relational database norm. To see how such a relationship is modeled and implemented, consider the PAINTER paints PAINTING example that was used in Chapter 2. Compare the data model in Figure 3.18 with its implementation in Figure 3.19.

FIGURE 3.18 The 1:M relationship between PAINTER and PAINTING

As you examine the PAINTER and PAINTING table contents in Figure 3.19, note the following features:

- Each painting is painted by one and only one painter, but each painter could have painted many paintings. Note that painter 123 (Georgette P. Ross) has three paintings stored in the PAINTING table.
- There is only one row in the PAINTER table for any given row in the PAINTING table, but there may be many rows in the PAINTING table for any given row in the PAINTER table.

FIGURE 3.19 The implemented 1:M relationship between PAINTER and PAINTING

Table name: PAINTER
Primary key: PAINTER_NUM
Foreign key: none

Database name: Ch03_Museum

PAINTER_NUM	PAINTER_LNAME	PAINTER_FNAME	PAINTER_INITIAL
123	Ross	Georgette	P
126	Itero	Julio	G

Table name: PAINTING
Primary key: PAINTING_NUM
Foreign key: PAINTER_NUM

PAINTING_NUM	PAINTING_TITLE	PAINTER_NUM
1338	Dawn Thunder	123
1339	Vanilla Roses To Nowhere	123
1340	Tired Flounders	126
1341	Hasty Exit	123
1342	Plastic Paradise	126

NOTE

The one-to-many (1:M) relationship is easily implemented in the relational model by putting the *primary key of the "1" side in the table of the "many" side as a foreign key.*

The 1:M relationship is found in any database environment. Students in a typical college or university will discover that each COURSE can generate many CLASSes but that each CLASS refers to only one COURSE. For example, an Accounting II course might yield two classes: one offered on Monday, Wednesday, and Friday (MWF) from 10:00 a.m. to 10:50 a.m. and one offered on Thursday (Th) from 6:00 p.m. to 8:40 p.m. Therefore, the 1:M relationship between COURSE and CLASS might be described this way:

- Each COURSE can have many CLASSes, but each CLASS references only one COURSE.
- There will be only one row in the COURSE table for any given row in the CLASS table, but there can be many rows in the CLASS table for any given row in the COURSE table.

FIGURE 3.20 The 1:M relationship between COURSE and CLASS

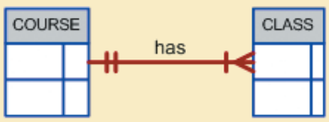


Figure 3.20 maps the ERM for the 1:M relationship between COURSE and CLASS.

The 1:M relationship between COURSE and CLASS is further illustrated in Figure 3.21.

FIGURE 3.21 The implemented 1:M relationship between COURSE and CLASS

Table name: COURSE
Primary key: CRS_CODE **Database name: Ch03_TinyCollege**
Foreign key: none

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

Using Figure 3.21, take a minute to review some important terminology. Note that CLASS_CODE in the CLASS table uniquely identifies each row. Therefore, CLASS_CODE has been chosen to be the primary key. However, the combination CRS_CODE and CLASS_SECTION will also uniquely identify each row in the class table. In other words, the *composite key* composed of CRS_CODE and CLASS_SECTION is a *candidate key*. Any candidate key must have the not null and unique constraints enforced. (You will see how this is done when you learn SQL in Chapter 7.)

For example, note in Figure 3.19 that the PAINTER table's primary key, PAINTER_NUM, is included in the PAINTING table as a foreign key. Similarly, in Figure 3.21, the COURSE table's primary key, CRS_CODE, is included in the CLASS table as a foreign key.

3.6.2 THE 1:1 RELATIONSHIP

As the 1:1 label implies, in this relationship, one entity can be related to only one other entity, and vice versa. For example, one department chair—a professor—can chair only one department and one department can have only one department chair. The entities PROFESSOR and DEPARTMENT thus exhibit a 1:1 relationship. (You might argue that not all professors chair a department and professors cannot be *required* to chair a department. That is, the relationship between the two entities is optional. However, at this stage of the discussion, you should focus your attention on the basic 1:1 relationship. Optional relationships will be addressed in Chapter 4.) The basic 1:1 relationship is modeled in Figure 3.22, and its implementation is shown in Figure 3.23.

As you examine the tables in Figure 3.23, note that there are several important features:

- Each professor is a Tiny College employee. Therefore, the professor identification is through the EMP_NUM. (However, note that not all employees are professors—there's another optional relationship.)

FIGURE 3.22 The 1:1 relationship between PROFESSOR and DEPARTMENT

- The 1:1 PROFESSOR chairs DEPARTMENT relationship is implemented by having the EMP_NUM foreign key in the DEPARTMENT table. Note that the 1:1 relationship is treated as a special case of the 1:M relationship in which the "many" side is restricted to a single occurrence. In this case, DEPARTMENT contains the EMP_NUM as a foreign key to indicate that it is the *department* that has a chair.

FIGURE 3.23 The implemented 1:1 relationship between PROFESSOR and DEPARTMENT

Table name: PROFESSOR
Primary key: EMP_NUM
Foreign key: DEPT_CODE

Database name: Ch03_TinyCollege

EMP_NUM	DEPT_CODE	PROF_OFFICE	PROF_EXTENSION	PROF_HIGH_DEGREE
103	HIST	DRE 156	6783	Ph.D.
104	ENG	DRE 102	5561	MA
105	ACCT	KLR 229D	8665	Ph.D.
106	MKT/MGT	KLR 126	3899	Ph.D.
110	BIOL	AAK 160	3412	Ph.D.
114	ACCT	KLR 211	4436	Ph.D.
155	MATH	AAK 201	4440	Ph.D.
160	ENG	DRE 102	2248	Ph.D.
162	CIS	KLR 203E	2359	Ph.D.
191	MKT/MGT	KLR 409B	4016	DBA
195	PSYCH	AAK 297	3550	Ph.D.
209	CIS	KLR 333	3421	Ph.D.
228	CIS	KLR 300	3000	Ph.D.
297	MATH	AAK 194	1145	Ph.D.
299	ECON/FIN	KLR 284	2851	Ph.D.
301	ACCT	KLR 244	4683	Ph.D.
335	ENG	DRE 208	2000	Ph.D.
342	SOC	BBG 208	5514	Ph.D.
387	BIOL	AAK 230	8665	Ph.D.
401	HIST	DRE 156	6783	MA
425	ECON/FIN	KLR 284	2851	MBA
435	ART	BBG 185	2278	Ph.D.

The 1:M DEPARTMENT employs PROFESSOR relationship is implemented through the placement of the DEPT_CODE foreign key in the PROFESSOR table.

The 1:1 PROFESSOR chairs DEPARTMENT relationship is implemented through the placement of the EMP_NUM foreign key in the DEPARTMENT table.

Table name: DEPARTMENT
Primary key: DEPT_CODE
Foreign key: EMP_NUM

DEPT_CODE	DEPT_NAME	SCHOOL_CODE	EMP_NUM	DEPT_ADDRESS	DEPT_EXTENSION
ACCT	Accounting	BUS	114	KLR 211, Box 52	3119
ART	Fine Arts	A&SCI	435	BBG 185, Box 128	2278
BIOL	Biology	A&SCI	387	AAK 230, Box 415	4117
CIS	Computer Info. Systems	BUS	209	KLR 333, Box 56	3245
ECON/FIN	Economics/Finance	BUS	299	KLR 284, Box 63	3126
ENG	English	A&SCI	160	DRE 102, Box 223	1004
HIST	History	A&SCI	103	DRE 156, Box 284	1867
MATH	Mathematics	A&SCI	297	AAK 194, Box 422	4234
MKT/MGT	Marketing/Management	BUS	106	KLR 126, Box 55	3342
PSYCH	Psychology	A&SCI	195	AAK 297, Box 438	4110
SOC	Sociology	A&SCI	342	BBG 208, Box 132	2008

- Also note that the PROFESSOR table contains the DEPT_CODE foreign key to implement the 1:M DEPARTMENT employs PROFESSOR relationship. This is a good example of how two entities can participate in two (or even more) relationships simultaneously.

ONLINE CONTENT

If you open the **Ch03_TinyCollege** database in the Student Online Companion, you'll see that the STUDENT and CLASS entities still use PROF_NUM as their foreign key. PROF_NUM and EMP_NUM are labels for the same attribute, which is an example of the use of synonyms—different names for the same attribute. These synonyms will be eliminated in future chapters as the Tiny College database continues to be improved.

The preceding "PROFESSOR chairs DEPARTMENT" example illustrates a proper 1:1 relationship. *In fact, the use of a 1:1 relationship ensures that two entity sets are not placed in the same table when they should not be.* However, the existence of a 1:1 relationship sometimes means that the entity components were not defined properly. It could indicate that the two entities actually belong in the same table!

As rare as 1:1 relationships should be, certain conditions absolutely *require* their use. For example, suppose you manage the database for a company that employs pilots, accountants, mechanics, clerks, salespeople, service personnel, and more. Pilots have many attributes that the other employees don't have, such as licenses, medical certificates, flight experience records, dates of flight proficiency checks, and proof of required periodic medical checks. If you put all of the pilot-specific attributes in the EMPLOYEE table, you will have several nulls in that table for all employees who are not pilots. To avoid the proliferation of nulls, it is better to split the pilot attributes into a separate table (PILOT) that is linked to the EMPLOYEE table in a 1:1 relationship. Because pilots have many attributes that are shared by all employees—such as name, date of birth, and date of first employment—those attributes would be stored in the EMPLOYEE table.

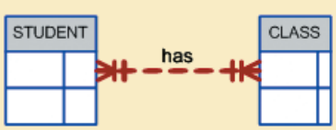
ONLINE CONTENT

If you look at the **Ch03_AviaCo** database in the Student Online Companion, you will see the implementation of the 1:1 PILOT to EMPLOYEE relationship. This type of relationship will be examined in detail in Chapter 6, Advanced Data Modeling.

3.6.3 THE M:N RELATIONSHIP

A many-to-many (M:N) relationship is not supported directly in the relational environment. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities.

FIGURE 3.24 The ERM's M:N relationship between STUDENT and CLASS



To explore the many-to-many (M:N) relationship, consider a rather typical college environment in which each STUDENT can take many CLASSes, and each CLASS can contain many STUDENTs. The ER model in Figure 3.24 shows this M:N relationship.

Note the features of the ERM in Figure 3.24.

- Each CLASS can have many STUDENTs, and each STUDENT can take many CLASSes.
- There can be many rows in the CLASS table for any given row in the STUDENT table, and there can be many rows in the STUDENT table for any given row in the CLASS table.

To examine the M:N relationship more closely, imagine a small college with two students, each of whom takes three classes. Table 3.7 shows the enrollment data for the two students.

TABLE 3.7 Sample Student Enrollment Data

STUDENT'S LAST NAME	SELECTED CLASSES
Bowser	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021
Smithson	Accounting 1, ACCT-211, code 10014 Intro to Microcomputing, CIS-220, code 10018 Intro to Statistics, QM-261, code 10021

FIGURE 3.25 The M:N relationship between STUDENT and CLASS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry

STU_NUM	STU_LNAME	CLASS_CODE
321452	Bowser	10014
321452	Bowser	10018
321452	Bowser	10021
324257	Smithson	10014
324257	Smithson	10018
324257	Smithson	10021

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: STU_NUM

CLASS_CODE	STU_NUM	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	321452	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10014	324257	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	321452	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10018	324257	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	321452	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10021	324257	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

Although the M:N relationship is logically reflected in Figure 3.24, it should *not* be implemented as shown in Figure 3.25 for two good reasons:

- The tables create many redundancies. For example, note that the STU_NUM values occur many times in the STUDENT table. In a real-world situation, additional student attributes such as address, classification, major, and home phone would also be contained in the STUDENT table, and each of those attribute values would be repeated in each of the records shown here. Similarly, the CLASS table contains many duplications: each student taking the class generates a CLASS record. The problem would be even worse if the CLASS table included such attributes as credit hours and course description. Those redundancies lead to the anomalies discussed in Chapter 1.
- Given the structure and contents of the two tables, the relational operations become very complex and are likely to lead to system efficiency errors and output errors.

Fortunately, the problems inherent in the many-to-many (M:N) relationship can easily be avoided by creating a **composite entity** (also referred to as a **bridge entity** or an **associative entity)**. Because such a table is used to link the tables that originally were related in a M:N relationship, the composite entity structure includes—as foreign keys—*at least* the primary keys of the tables that are to be linked. The database designer has two main options when defining a composite table's primary key: use the combination of those foreign keys or create a new primary key.

Remember that each entity in the ERM is represented by a table. Therefore, you can create the composite ENROLL table shown in Figure 3.26 to link the tables CLASS and STUDENT. In this example, the ENROLL table's primary key is the combination of its foreign keys CLASS_CODE and STU_NUM. But the designer could have decided to create a single-attribute new primary key such as ENROLL_LINE, using a different line value to identify each ENROLL table row uniquely. (Microsoft Access users might use the *Autonumber* data type to generate such line values automatically.)

FIGURE 3.26 **Converting the M:N relationship into two 1:M relationships**

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry2

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL
Primary key: CLASS_CODE + STU_NUM
Foreign key: CLASS_CODE, STU_NUM

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

Because the ENROLL table in Figure 3.26 links two tables, STUDENT and CLASS, it is also called a **linking table**. In other words, a linking table is the implementation of a composite entity.

NOTE

In addition to the linking attributes, the composite ENROLL table can also contain such relevant attributes as the grade earned in the course. In fact, a composite table can contain any number of attributes that the designer wants to track. Keep in mind that the composite entity, *although it is implemented as an actual table*, is *conceptually* a logical entity that was created as a means to an end: to eliminate the potential for multiple redundancies in the original M:N relationship.

The linking table (ENROLL) shown in Figure 3.26 yields the required M:N to 1:M conversion. Observe that the composite entity represented by the ENROLL table must contain at least the primary keys of the CLASS and

STUDENT tables (CLASS_CODE and STU_NUM, respectively) for which it serves as a connector. Also note that the STUDENT and CLASS tables now contain only one row per entity. The linking ENROLL table contains multiple occurrences of the foreign key values, but those controlled redundancies are incapable of producing anomalies as long as referential integrity is enforced. Additional attributes may be assigned as needed. In this case, ENROLL_GRADE is selected to satisfy a reporting requirement. Also note that the ENROLL table's primary key consists of the two attributes CLASS_CODE and STU_NUM because both the class code and the student number are needed to define a particular student's grade. Naturally, the conversion is reflected in the ERM, too. The revised relationship is shown in Figure 3.27.

As you examine Figure 3.27, note that the composite entity named ENROLL represents the linking table between STUDENT and CLASS.

The 1:M relationship between COURSE and CLASS was first illustrated in Figure 3.20 and Figure 3.21. With the help of this relationship, you can increase the amount of available information even as you control the database's redundancies. Thus, Figure 3.27 can be expanded to include the 1:M relationship between COURSE and CLASS shown in Figure 3.28. Note that the model is able to handle multiple sections of a CLASS while controlling redundancies by making sure that all of the COURSE data common to each CLASS are kept in the COURSE table.

The relational diagram that corresponds to the ERD in Figure 3.28 is shown in Figure 3.29.

FIGURE 3.27 Changing the M:N relationship to two 1:M relationships

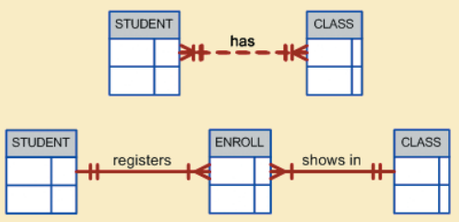


FIGURE 3.28 The expanded entity relationship model

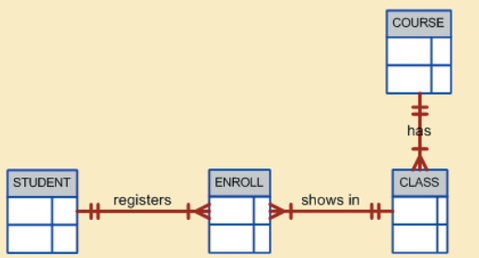
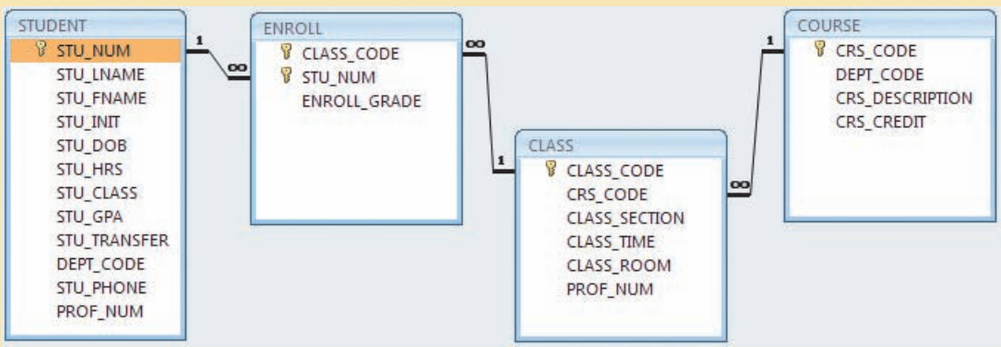


FIGURE 3.29 The relational diagram for the Ch03_TinyCollege database



The ERD will be examined in greater detail in Chapter 4 to show you how it is used to design more complex databases. The ERD will also be used as the basis for the development and implementation of a realistic database design in Appendixes B and C (see the Student Online Companion Web site) for a university computer lab.

3.7 DATA REDUNDANCY REVISITED

In Chapter 1 you learned that data redundancy leads to data anomalies. Those anomalies can destroy the effectiveness of the database. You also learned that the relational database makes it possible to control data redundancies by using common attributes that are shared by tables, called foreign keys.

The proper use of foreign keys is crucial to controlling data redundancy. Although the use of foreign keys does not totally eliminate data redundancies because the foreign key values can be repeated many times, the proper use of foreign keys *minimizes* data redundancies, thus minimizing the chance that destructive data anomalies will develop.

NOTE

The real test of redundancy is *not* how many copies of a given attribute are stored, *but whether the elimination of an attribute will eliminate information*. Therefore, if you delete an attribute and the original information can still be generated through relational algebra, the inclusion of that attribute would be redundant. Given that view of redundancy, proper foreign keys are clearly not redundant in spite of their multiple occurrences in a table. However, even when you use this less restrictive view of redundancy, keep in mind that *controlled* redundancies are often designed as part of the system to ensure transaction speed and/or information requirements. Exclusive reliance on relational algebra to produce required information may lead to elegant designs that fail the test of practicality.

You will learn in Chapter 4 that database designers must reconcile three often contradictory requirements: design elegance, processing speed, and information requirements. And you will learn in Chapter 13, Business Intelligence and Data Warehouses, that proper data warehousing design requires carefully defined and controlled data redundancies to function properly. Regardless of how you describe data redundancies, the potential for damage is limited by proper implementation and careful control.

As important as data redundancy control is, there are times when the level of data redundancy must actually be increased to make the database serve crucial information purposes. You will learn about such redundancies in Chapter 13. There are also times when data redundancies *seem* to exist to preserve the historical accuracy of the data. For example, consider a small invoicing system. The system includes the CUSTOMER, who may buy one or more PRODUCTs, thus generating an INVOICE. Because a customer may buy more than one product at a time, an invoice may contain several invoice LINEs, each providing details about the purchased product. The PRODUCT table should contain the product price to provide a consistent pricing input for each product that appears on the invoice. The tables that are part of such a system are shown in Figure 3.30. The system's relational diagram is shown in Figure 3.31.

As you examine the tables in the invoicing system in Figure 3.30 and the relationships depicted in Figure 3.31, note that you can keep track of typical sales information. For example, by tracing the relationships among the four tables, you discover that customer 10014 (Myron Orlando) bought two items on March 8, 2006 that were written to invoice number 1001: one Houselite chain saw with a 16-inch bar and three rat-tail files. (*Note*: Trace the CUS_CODE number 10014 in the CUSTOMER table to the matching CUS_CODE value in the INVOICE table. Next, take the INV_NUMBER 1001 and trace it to the first two rows in the LINE table. Finally, match the two PROD_CODE values in LINE with the PROD_CODE values in PRODUCT.) Application software will be used to write the correct bill by multiplying each invoice line item's LINE_UNITS by its LINE_PRICE, adding the results, applying appropriate taxes, etc. Later, other application software might use the same technique to write sales reports that track and compare sales by week, month, or year.

FIGURE 3.30 A small invoicing system

Table name: CUSTOMER
Primary key: CUS_CODE
Foreign key: none

Database name: Ch03_SaleCo

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
10010	Ramas	Alfred	A	615	844-2573
10011	Dunne	Leona	K	713	894-1238
10012	Smith	Kathy	W	615	894-2285
10013	Olowski	Paul	F	615	894-2180
10014	Orlando	Myron		615	222-1672
10015	O'Brian	Amy	B	713	442-3381
10016	Brown	James	G	615	297-1228
10017	Williams	George		615	290-2556
10018	Farriss	Anne	G	713	382-7185
10019	Smith	Olette	K	615	297-3809

Table name: INVOICE
Primary key: INV_NUMBER
Foreign key: CUS_CODE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	08-Mar-08
1002	10011	08-Mar-08
1003	10012	08-Mar-08
1004	10011	09-Mar-08

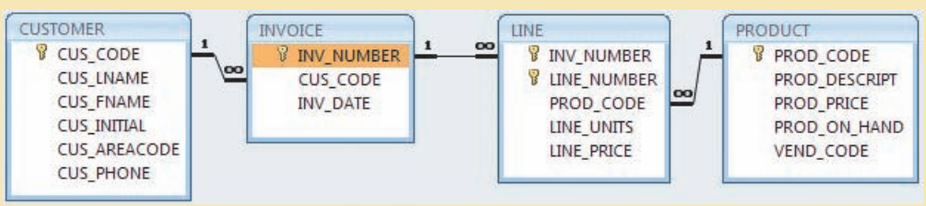
Table name: LINE
Primary key: INV_NUMBER + LINE_NUMBER
Foreign keys: INV_NUMBER, PROD_CODE

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE
1001	1	123-21UUY	1	189.99
1001	2	SRE-657UG	3	2.99
1002	1	QER-34256	2	18.63
1003	1	ZZX/3245Q	1	6.79
1003	2	SRE-657UG	1	2.99
1003	3	001278-AB	1	12.95
1004	1	001278-AB	1	12.95
1004	2	SRE-657UG	2	2.99

Table name: PRODUCT
Primary key: PROD_CODE
Foreign key: none

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

FIGURE 3.31 The relational diagram for the invoicing system



As you examine the sales transactions in Figure 3.30, you might reasonably suppose that the product price billed to the customer is derived from the PRODUCT table because that's where the product data are stored. *But why does that same product price occur again in the LINE table? Isn't that a data redundancy?* It certainly *appears* to be. But this time, the apparent redundancy is crucial to the system's success. Copying the product price from the PRODUCT table

to the LINE table maintains the *historical accuracy of the transactions*. Suppose, for instance, that you fail to write the LINE_PRICE in the LINE table and that you use the PROD_PRICE from the PRODUCT table to calculate the sales revenue. Now suppose that the PRODUCT table's PROD_PRICE changes, as prices frequently do. This price change will be properly reflected in all subsequent sales revenue calculations. However, the calculations of past sales revenues will also reflect the new product price that was not in effect when the transaction took place! As a result, the revenue calculations for all past transactions will be incorrect, thus eliminating the possibility of making proper sales comparisons over time. On the other hand, if the price data are copied from the PRODUCT table and stored with the transaction in the LINE table, that price will always accurately reflect the transaction that took place *at that time*. You will discover that such planned "redundancies" are common in good database design.

Finally, you might wonder why the LINE_NUMBER attribute was used in the LINE table in Figure 3.30. Wouldn't the combination of INV_NUMBER and PROD_CODE be a sufficient composite primary key—and, therefore, isn't the LINE_NUMBER redundant? Yes, the LINE_NUMBER is redundant, but this redundancy is quite commonly created by invoicing software that generates such line numbers automatically. In this case, the redundancy is not necessary. But given its automatic generation, the redundancy is not a source of anomalies. The inclusion of LINE_NUMBER also adds another benefit: the order of the retrieved invoicing data will always match the order in which the data were entered. If product codes are used as part of the primary key, indexing will arrange those product codes as soon as the invoice is completed and the data are stored. You can imagine the potential confusion when a customer calls and says, "The second item on my invoice has an incorrect price" and you are looking at an invoice whose lines show a different order from those on the customer's copy!

3.8 INDEXES

Suppose you want to locate a particular book in a library. Does it make sense to look through every book in the library until you find the one you want? Of course not; you use the library's catalog, which is indexed by title, topic, and author. The index (in either a manual or a computer system) points you to the book's location, thereby making retrieval of the book a quick and simple matter. An **index** is an orderly arrangement used to logically access rows in a table.

Or suppose you want to find a topic, such as "ER model," in this book. Does it make sense to read through every page until you stumble across the topic? Of course not; it is much simpler to go to the book's index, look up the phrase *ER model*, and read the page references that point you to the appropriate page(s). In each case, an index is used to locate a needed item quickly.

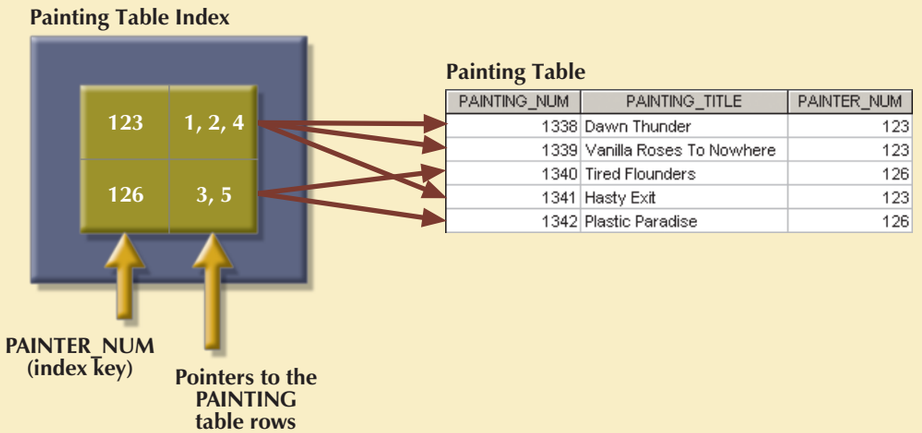
Indexes in the relational database environment work like the indexes described in the preceding paragraphs. From a conceptual point of view, an index is composed of an index key and a set of pointers. The **index key** is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.

For example, suppose you want to look up all of the paintings created by a given painter in the Ch03_Museum database in Figure 3.19. Without an index, you must read each row in the PAINTING table and see if the PAINTER_NUM matches the requested painter. However, if you index the PAINTER table and use the index key PAINTER_NUM, you merely need to look up the appropriate PAINTER_NUM in the index and find the matching pointers. Conceptually speaking, the index would resemble the presentation depicted in Figure 3.32.

As you examine Figure 3.32 and compare it to the Ch03_Museum database tables shown in Figure 3.19, note that the first PAINTER_NUM index key value (123) is found in records 1, 2, and 4 of the PAINTING table in Figure 3.19. The second PAINTER_NUM index key value (126) is found in records 3 and 5 of the PAINTING table in Figure 3.19.

DBMSs use indexes for many different purposes. You just learned that an index can be used to retrieve data more efficiently. But indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes. For example, creating an index on a customer's last name will allow you to retrieve the customer data alphabetically by the

FIGURE 3.32 Components of an index



customer's last name. Also, an index key can be composed of one or more attributes. For example, in Figure 3.30, you can create an index on VEND_CODE and PROD_CODE to retrieve all rows in the PRODUCT table ordered by vendor, and within vendor, ordered by product.

Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared. For example, in Figure 3.30, when you declare CUS_CODE to be the primary key of the CUSTOMER table, the DBMS automatically creates a unique index on that attribute. A **unique index**, as its name implies, is an index in which the index key can have only one pointer value (row) associated with it. (The index in Figure 3.32 is not a unique index because the PAINTER_NUM has multiple pointer values associated with it. For example, painter number 123 points to three rows—1, 2, and 4—in the PAINTING table.)

A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (composite index). Creating an index is easy. You learn in Chapter 7 that a simple SQL command produces any required index.

3.9 CODD'S RELATIONAL DATABASE RULES

In 1985, Dr. E. F. Codd published a list of 12 rules to define a relational database system.[2] The reason Dr. Codd published the list was his concern that many vendors were marketing products as "relational" even though those products did not meet minimum relational standards. Dr. Codd's list, shown in Table 3.8, serves as a frame of reference for what a truly relational database should be. Bear in mind that even the dominant database vendors do not fully support all 12 rules.

[2] Codd, E., "Is Your DBMS Really Relational?" and "Does Your DBMS Run by the Rules?" *Computerworld*, October 14 and October 21, 1985.

TABLE 3.8 **Dr. Codd's 12 Relational Database Rules**

RULE	RULE NAME	DESCRIPTION
1	**Information**	All information in a relational database must be logically represented as column values in rows within tables.
2	**Guaranteed Access**	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	**Systematic Treatment of Nulls**	Nulls must be represented and treated in a systematic way, independent of data type.
4	**Dynamic On-Line Catalog Based on the Relational Model**	The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
5	**Comprehensive Data Sublanguage**	The relational database may support many languages. However, it must support one well defined, declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
6	**View Updating**	Any view that is theoretically updatable must be updatable through the system.
7	**High-Level Insert, Update and Delete**	The database must support set-level inserts, updates, and deletes.
8	**Physical Data Independence**	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	**Logical Data Independence**	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of column or inserting columns).
10	**Integrity Independence**	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
11	**Distribution Independence**	The end users and application programs are unaware and unaffected by the data location (distributed vs. local databases).
12	**Nonsubversion**	If the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
	Rule Zero	All preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational facilities exclusively to manage the database.

SUMMARY

- Tables are the basic building blocks of a relational database. A grouping of related entities, known as an entity set, is stored in a table. Conceptually speaking, the relational table is composed of intersecting rows (tuples) and columns. Each row represents a single entity, and each column represents the characteristics (attributes) of the entities.
- Keys are central to the use of relational tables. Keys define functional dependencies; that is, other attributes are dependent on the key and can, therefore, be found if the key value is known. A key can be classified as a superkey, a candidate key, a primary key, a secondary key, or a foreign key.
- Each table row must have a primary key. The primary key is an attribute or a combination of attributes that uniquely identifies all remaining attributes found in any given row. Because a primary key must be unique, no null values are allowed if entity integrity is to be maintained.
- Although the tables are independent, they can be linked by common attributes. Thus, the primary key of one table can appear as the foreign key in another table to which it is linked. Referential integrity dictates that the foreign key must contain values that match the primary key in the related table or must contain nulls.
- The relational model supports relational algebra functions: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. A relational database performs much of the data manipulation work behind the scenes. For example, when you create a database, the RDBMS automatically produces a structure to house a data dictionary for your database. Each time you create a new table within the database, the RDBMS updates the data dictionary, thereby providing the database documentation.
- Once you know the relational database basics, you can concentrate on design. Good design begins by identifying appropriate entities and their attributes and then the relationships among the entities. Those relationships (1:1, 1:M, and M:N) can be represented using ERDs. The use of ERDs allows you to create and evaluate simple logical design. The 1:M relationship is most easily incorporated in a good design; you just have to make sure that the primary key of the "1" is included in the table of the "many."

KEY TERMS

associative entity, 86
attribute domain, 32
bridge entity, 86
candidate key, 68
closure, 73
composite entity, 86
composite key, 67
data dictionary, 78
determination, 66
domain, 66
entity integrity, 68
equijoin, 77
flags, 72
foreign key (FK), 70
full functional dependence, 68
functional dependence, 67
homonyms, 80
index, 90
index key, 90
join column(s), 76
key, 66
key attribute, 67
left outer join, 77
linking table, 86
natural join, 76
null, 68
outer join, 77
predicate logic, 63
primary key (PK), 66
referential integrity, 70
relational algebra, 72
relational schema, 70
right outer join, 77
secondary key, 70
set theory, 63
superkey, 68
synonym, 80
system catalog, 80
theta join, 77
tuple, 38
union-compatible, 73
unique index, 91

ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Student Online Companion for this book.

REVIEW QUESTIONS

1. What is the difference between a database and a table?
2. What does it mean to say that a database displays both entity integrity and referential integrity?
3. Why are entity integrity and referential integrity important in a database?
4. A database user manually notes that "The file contains two hundred records, each record containing nine fields." Use appropriate relational database terminology to "translate" that statement.
5. Use the small database shown in Figure Q3.5 to illustrate the difference between a natural join, an equijoin, and an outer join.

FIGURE Q3.5 The Ch03_CollegeQue database tables

Database name: Ch03_CollegeQue

Table name: STUDENT

STU_CODE	PROF_CODE
100278	
128569	2
512272	4
531235	2
531268	
553427	1

Table name: PROFESSOR

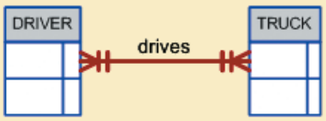
		PROF_CODE	DEPT_CODE
▶	+	1	2
	+	2	6
	+	3	6
	+	4	4

6. Create the basic ERD for the database shown in Figure Q3.5.
7. Create the relational diagram for the database shown in Figure Q3.5.
8. Suppose you have the ERM shown in Figure Q3.8. How would you convert this model into an ERM that displays only 1:M relationships? (Make sure you create the revised ERM.)
9. What are homonyms and synonyms, and why should they be avoided in database design?
10. How would you implement a l:M relationship in a database composed of two tables? Give an example.
11. Identify and describe the components of the table shown in Figure Q3.11, using correct terminology. Use your knowledge of naming conventions to identify the table's probable foreign key(s).

ONLINE CONTENT

All of the databases used in the questions and problems are found in the Student Online Companion for this book. The database names used in the folder match the database names used in the figures. For example, the source of the tables shown in Figure Q3.5 is the **Ch03_CollegeQue** database.

FIGURE Q3.8 The Crow's Foot ERM for Question 8



During some time interval, a DRIVER can drive many TRUCKs and any TRUCK can be driven by many DRIVERs.

FIGURE Q3.11 The Ch03_NoComp database EMPLOYEE table

Table name: EMPLOYEE **Database name: Ch03_NoComp**

EMP_NUM	EMP_LNAME	EMP_INITIAL	EMP_FNAME	DEPT_CODE	JOB_CODE
11234	Friedman	K	Robert	MKTG	12
11238	Olanski	D	Delbert	MKTG	12
11241	Fontein		Juliette	INFS	5
11242	Cruazona	J	Maria	ENG	9
11245	Smithson	B	Bernard	INFS	6
11248	Washington	G	Oleta	ENGR	8
11256	McBride		Randall	ENGR	8
11257	Kachinn	D	Melanie	MKTG	14
11258	Smith	W	William	MKTG	14
11260	Ratula	A	Katrina	INFS	5

Use the database composed of the two tables shown in Figure Q3.12 to answer Questions 12-17.

FIGURE Q3.12 The Ch03_Theater database tables

Database name: Ch03_Theater

Table name: DIRECTOR

DIR_NUM	DIR_LNAME	DIR_DOB
100	Broadway	12-Jan-65
101	Hollywoody	18-Nov-53
102	Goofy	21-Jun-62

Table name: PLAY

PLAY_CODE	PLAY_NAME	DIR_NUM
1001	Cat On a Cold, Bare Roof	102
1002	Hold the Mayo, Pass the Bread	101
1003	I Never Promised You Coffee	102
1004	Silly Putty Goes To Washington	100
1005	See No Sound, Hear No Sight	101
1006	Starstruck in Biloxi	102
1007	Stranger In Parrot Ice	101

12. Identify the primary keys.
13. Identify the foreign keys.
14. Create the ERM.
15. Create the relational diagram to show the relationship between DIRECTOR and PLAY.
16. Suppose you wanted quick lookup capability to get a listing of all plays directed by a given director. Which table would be the basis for the INDEX table, and what would be the index key?
17. What would be the conceptual view of the INDEX table that is described in Question 16? Depict the contents of the conceptual INDEX table.

PROBLEMS

Use the database shown in Figure P3.1 to work Problems 1–7. Note that the database is composed of four tables that reflect these relationships:

- An EMPLOYEE has only one JOB_CODE, but a JOB_CODE can be held by many EMPLOYEEs.
- An EMPLOYEE can participate in many PLANs, and any PLAN can be assigned to many EMPLOYEEs.

Note also that the M:N relationship has been broken down into two 1:M relationships for which the BENEFIT table serves as the composite or bridge entity.

FIGURE P3.1 The Ch03_BeneCo database tables

Database name: Ch03_BeneCo

Table name: EMPLOYEE

EMP_CODE	EMP_LNAME	JOB_CODE
14	Rudell	2
15	McDade	1
16	Ruellardo	1
17	Smith	3
20	Smith	2

Table name: BENEFIT

EMP_CODE	PLAN_CODE
15	2
15	3
16	1
17	1
17	3
17	4
20	3

Table name: JOB

JOB_CODE	JOB_DESCRIPTION
1	Clerical
2	Technical
3	Managerial

Table name: PLAN

PLAN_CODE	PLAN_DESCRIPTION
1	Term life
2	Stock purchase
3	Long-term disability
4	Dental

1. For each table in the database, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.

TABLE	PRIMARY KEY	FOREIGN KEY(S)
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

2. Create the ERD to show the relationship between EMPLOYEE and JOB.
3. Create the relational diagram to show the relationship between EMPLOYEE and JOB.
4. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.

TABLE	ENTITY INTEGRITY	EXPLANATION
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

5. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.

TABLE	REFERENTIAL INTEGRITY	EXPLANATION
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

6. Create the ERD to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.
7. Create the relational diagram to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.

Use the database shown in Figure P3.8 to answer Problems 8–16.

FIGURE P3.8 **The Ch03_StoreCo database tables**

Table name: EMPLOYEE **Database name: Ch03_StoreCo**

EMP_CODE	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	STORE_CODE
1	Mr.	Williamson	John	W	21-May-64	3
2	Ms.	Ratula	Nancy		09-Feb-69	2
3	Ms.	Greenboro	Lottie	R	02-Oct-61	4
4	Mrs.	Rumpersfro	Jennie	S	01-Jun-71	5
5	Mr.	Smith	Robert	L	23-Nov-59	3
6	Mr.	Renselaer	Cary	A	25-Dec-65	1
7	Mr.	Ogallo	Roberto	S	31-Jul-62	3
8	Ms.	Johnsson	Elizabeth	I	10-Sep-68	1
9	Mr.	Eindsmar	Jack	W	19-Apr-55	2
10	Mrs.	Jones	Rose	R	06-Mar-66	4
11	Mr.	Broderick	Tom		21-Oct-72	3
12	Mr.	Washington	Alan	Y	08-Sep-74	2
13	Mr.	Smith	Peter	N	25-Aug-64	3
14	Ms.	Smith	Sherry	H	25-May-66	4
15	Mr.	Olenko	Howard	U	24-May-64	5
16	Mr.	Archialo	Barry	V	03-Sep-60	5
17	Ms.	Grimaldo	Jeanine	K	12-Nov-70	4
18	Mr.	Rosenberg	Andrew	D	24-Jan-71	4
19	Mr.	Rosten	Peter	F	03-Oct-68	4
20	Mr.	Mckee	Robert	S	06-Mar-70	1
21	Ms.	Baumann	Jennifer	A	11-Dec-74	3

Table name: STORE

STORE_CODE	STORE_NAME	STORE_YTD_SALES	REGION_CODE	EMP_CODE
1	Access Junction	1003455.76	2	8
2	Database Corner	1421987.39	2	12
3	Tuple Charge	986783.22	1	7
4	Attribute Alley	944568.56	2	3
5	Primary Key Point	2930098.45	1	15

Table name: REGION

REGION_CODE	REGION_DESCRIPT
1	East
2	West

8. For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.

TABLE	PRIMARY KEY	FOREIGN KEY(S)
EMPLOYEE		
STORE		
REGION		

9. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.

TABLE	ENTITY INTEGRITY	EXPLANATION
EMPLOYEE		
STORE		
REGION		

10. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.

TABLE	REFERENTIAL INTEGRITY	EXPLANATION
EMPLOYEE		
STORE		
REGION		

11. Describe the type(s) of relationship(s) between STORE and REGION.
12. Create the ERD to show the relationship between STORE and REGION.
13. Create the relational diagram to show the relationship between STORE and REGION.
14. Describe the type(s) of relationship(s) between EMPLOYEE and STORE. (*Hint*: Each store employs many employees, one of whom manages the store.)
15. Create the ERD to show the relationships among EMPLOYEE, STORE, and REGION.
16. Create the relational diagram to show the relationships among EMPLOYEE, STORE, and REGION.

Use the database shown in Figure P3.17 to answer Problems 17–22.

FIGURE P3.17 The Ch03_CheapCo database tables

Table name: PRODUCT **Database name: Ch03_CheapCo**
Primary key: PROD_CODE
Foreign key: VEND_CODE

PROD_CODE	PROD_DESCRIPTION	PROD_STOCK_DATE	PROD_ON_HAND	PROD_PRICE	VEND_CODE
12-WW/P2	7.25-in. power saw blade	07-Apr-08	12	1.19	123
1QQ23-55	2.5-in. wood screw, 100	19-Mar-08	123	4.49	123
231-78-W	PVC pipe, 3.5-in., 8 ft.	07-Dec-07	45	8.87	121
33564/U	Rat-tail file, 0.125-in., fine	08-Mar-08	18	1.19	123
AR/3/TYR	Cordless drill, 0.25-in.	29-Nov-07	8	45.99	121
DT-34-WW	Phillips screwdriver pack	20-Dec-07	11	23.29	123
EE3-67/W	Sledge hammer, 12 lb.	25-Feb-08	9	17.99	121
ER-56/DF	Houselite chain saw, 16-in.	28-Dec-07	7	235.49	125
FRE-TRY9	Jigsaw, 12-in blade	12-Aug-07	67	1.45	125
SE-67-89	Jigsaw, 8-in. blade	11-Oct-07	34	1.35	125
ZW-QR/AV	Hardware cloth, 0.25-in.	23-Apr-08	14	12.99	123
ZX-WR/FR	Claw hammer	01-Mar-08	15	8.95	121

Table name: VENDOR
Primary key: VEND_CODE
Foreign key: none

VEND_CODE	VEND_NAME	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
120	BargainSnapper, Inc.	Melanie T. Travis	615	899-1234
121	Cut'nGlow Co.	Henry J. Olero	615	342-9896
122	Rip & Rattle Supply Co.	Anne R. Morrins	901	225-1127
123	Tools 'R Us	Juliette G. McHenry	615	546-7894
124	Trowel & Dowel, Inc.	George F. Frederick	901	453-4567
125	Bow & Wow Tools	Bill S. Sedwick	904	324-9988

17. For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.

TABLE	PRIMARY KEY	FOREIGN KEY(S)
PRODUCT		
VENDOR		

18. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.

TABLE	ENTITY INTEGRITY	EXPLANATION
PRODUCT		
VENDOR		

19. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.

TABLE	REFERENTIAL INTEGRITY	EXPLANATION
PRODUCT		
VENDOR		

20. Create the ERD for this database.

21. Create the relational diagram for this database.
22. Create the data dictionary for this database.

Use the database shown in Figure P3.23 to answer Problems 23–29.

FIGURE P3.23 The Ch03_TransCo database tables

Table name: TRUCK **Database name: Ch03_TransCo**
Primary key: TRUCK_NUM
Foreign key: BASE_CODE, TYPE_CODE

TRUCK_NUM	BASE_CODE	TYPE_CODE	TRUCK_MILES	TRUCK_BUY_DATE	TRUCK_SERIAL_NUM
1001	501	1	32123.5	23-Sep-07	AA-322-12212-W11
1002	502	1	76984.3	05-Feb-06	AC-342-22134-Q23
1003	501	2	12346.6	11-Nov-06	AC-445-78656-Z99
1004		1	2894.3	06-Jan-07	WQ-112-23144-T34
1005	503	2	45673.1	01-Mar-06	FR-998-32245-W12
1006	501	2	193245.7	15-Jul-03	AD-456-00845-R45
1007	502	3	32012.3	17-Oct-04	AA-341-96573-Z84
1008	502	3	44213.6	07-Aug-05	DR-559-22189-D33
1009	503	2	10932.9	12-Feb-08	DE-887-98456-E94

Table name: BASE
Primary key: BASE_CODE
Foreign key: none

BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER
501	Murfreesboro	TN	615	123-4567	Andrea D. Gallager
502	Lexington	KY	568	234-5678	George H. Delarosa
503	Cape Girardeau	MO	456	345-6789	Maria J. Talindo
504	Dalton	GA	901	456-7890	Peter F. McAvee

Table name: TYPE
Primary key: TYPE_CODE
Foreign key: none

TYPE_CODE	TYPE_DESCRIPTION
1	Single box, double-axle
2	Single box, single-axle
3	Tandem trailer, single-axle

23. For each table, identify the primary key and the foreign key(s). If a table does not have a foreign key, write *None* in the space provided.

TABLE	PRIMARY KEY	FOREIGN KEY(S)
TRUCK		
BASE		
TYPE		

24. Do the tables exhibit entity integrity? Answer yes or no, and then explain your answer.

TABLE	ENTITY INTEGRITY	EXPLANATION
TRUCK		
BASE		
TYPE		

25. Do the tables exhibit referential integrity? Answer yes or no, and then explain your answer. Write *NA* (Not Applicable) if the table does not have a foreign key.

TABLE	REFERENTIAL INTEGRITY	EXPLANATION
TRUCK		
BASE		
TYPE		

26. Identify the TRUCK table's candidate key(s).
27. For each table, identify a superkey and a secondary key.

TABLE	SUPERKEY	SECONDARY KEY
TRUCK		
BASE		
TYPE		

28. Create the ERD for this database.
29. Create the relational diagram for this database.

Use the database shown in Figure P3.30 to answer Problems 30–34. ROBCOR is an aircraft charter company that supplies on-demand charter flight services using a fleet of four aircraft. Aircrafts are identified by a unique registration number. Therefore, the aircraft registration number is an appropriate primary key for the AIRCRAFT table.

FIGURE P3.30 **The Ch03_AviaCo database tables**

Table name: CHARTER **Database name: Ch03_AviaCo**

CHAR_TRIP	CHAR_DATE	CHAR_PILOT	CHAR_COPILOT	CHAR_DESTINATION	CHAR_DISTANCE	CHAR_HOURS_FLOWN	CHAR_HOURS_WAIT	CUS_CODE
10001	05-Feb-08	104		ATL	936.0	5.1	2.2	10011
10002	05-Feb-08	101		BNA	320.0	1.6	0.0	10016
10003	05-Feb-08	105	109	GNV	1574.0	7.8	0.0	10014
10004	06-Feb-08	106		STL	472.0	2.9	4.9	10019
10005	06-Feb-08	101		ATL	1023.0	5.7	3.5	10011
10006	06-Feb-08	109		STL	472.0	2.6	5.2	10017
10007	06-Feb-08	104	105	GNV	1574.0	7.9	0.0	10012
10008	07-Feb-08	106		TYS	644.0	4.1	0.0	10014
10009	07-Feb-08	105		GNV	1574.0	6.6	23.4	10017
10010	07-Feb-08	109		ATL	998.0	6.2	3.2	10016
10011	07-Feb-08	101	104	BNA	352.0	1.9	5.3	10012
10012	08-Feb-08	101		MOB	884.0	4.8	4.2	10010
10013	08-Feb-08	105		TYS	644.0	3.9	4.5	10011
10014	09-Feb-08	106		ATL	936.0	6.1	2.1	10017
10015	09-Feb-08	104	101	GNV	1645.0	6.7	0.0	10016
10016	09-Feb-08	109	105	MQY	312.0	1.5	0.0	10011
10017	10-Feb-08	101		STL	508.0	3.1	0.0	10014
10018	10-Feb-08	105	104	TYS	644.0	3.8	4.5	10017

The destinations are indicated by standard three-letter airport codes. For example,
STL = St. Louis, MO ATL = Atlanta, GA BNA = Nashville, TN

Table name: AIRCRAFT

AC_NUMBER	MOD_CODE	AC_TTAF	AC_TTEL	AC_TTER
1484P	PA23-250	1833.1	1833.1	101.8
2289L	C-90A	4243.8	768.9	1123.4
2778V	PA31-350	7992.9	1513.1	789.5
4278Y	PA31-350	2147.3	622.1	243.2

AC-TTAF = Aircraft total time, airframe (hours)
AC-TTEL = Total time, left engine (hours)
AC_TTER = Total time, right engine (hours)

In a fully developed system, such attribute values would be updated by application software when the CHARTER table entries are posted.

Table name: MODEL

MOD_CODE	MOD_MANUFACTURER	MOD_NAME	MOD_SEATS	MOD_CHG_MILE
C-90A	Beechcraft	KingAir	8	2.67
PA23-250	Piper	Aztec	6	1.93
PA31-350	Piper	Navajo Chieftain	10	2.35

Customers are charged per round-trip mile, using the MOD_CHG_MILE rate. The MOD_SEAT gives the total number of seats in the airplane, including the pilot and copilot seats. Therefore, a PA31-350 trip that is flown by a pilot and a copilot has six passenger seats available.

FIGURE P3.30 **The Ch03_AviaCo database tables (continued)**

Table name: PILOT **Database name: Ch03_AviaCo**

EMP_NUM	PIL_LICENSE	PIL_RATINGS	PIL_MED_TYPE	PIL_MED_DATE	PIL_PT135_DATE
101	ATP	ATP/SEL/MEL/Instr/CFII	1	20-Jan-08	11-Jan-08
104	ATP	ATP/SEL/MEL/Instr	1	18-Dec-07	17-Jan-08
105	COM	COMM/SEL/MEL/Instr/CFI	2	05-Jan-08	02-Jan-08
106	COM	COMM/SEL/MEL/Instr	2	10-Dec-07	02-Feb-08
109	COM	ATP/SEL/MEL/SES/Instr/CFII	1	22-Jan-08	15-Jan-08

The pilot licenses shown in the PILOT table include the ATP = Airline Transport Pilot and COM = Commercial Pilot. Businesses that operate on-demand air services are governed by Part 135 of the Federal Air Regulations (FARs) that are enforced by the Federal Aviation Administration (FAA). Such businesses are known as "Part 135 operators." Part 125 operations require that pilots successfully complete flight proficiency checks every six months. The "Part 135" flight proficiency check data is recorded in PIL_PT135_DATE. To fly commercially, pilots must have at least a commercial license and a second-class medical certificate (PIL_MED_TYPE = 2).

The PIL_RATINGS include

SEL = Single Engine, Land
SES = Single Engine, Sea
CFI = Certified Flight Instructor
MEL = Multiengine, Land
Instr. = Instrument
CFII = Certified Flight Instructor, Instrument

Table name: EMPLOYEE

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-88
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86
102	Mr.	Vandam	Rhett		14-Nov-58	18-May-93
103	Ms.	Jones	Anne	M	11-May-74	26-Jul-99
104	Mr.	Lange	John	P	12-Jul-71	20-Aug-90
105	Mr.	Williams	Robert	D	14-Mar-75	19-Jun-03
106	Mrs.	Duzak	Jeanine	K	12-Feb-68	13-Mar-89
107	Mr.	Diante	Jorge	D	01-May-75	02-Jul-97
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	03-Jun-93
109	Ms.	Travis	Elizabeth	K	18-Jun-61	14-Feb-06
110	Mrs.	Genkazi	Leighla	W	19-May-70	29-Jun-90

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	896.54
10013	Olowski	Paul	F	615	894-2180	1285.19
10014	Orlando	Myron		615	222-1672	673.21
10015	O'Brian	Amy	B	713	442-3381	1014.56
10016	Brown	James	G	615	297-1228	0.00
10017	Williams	George		615	290-2556	0.00
10018	Farriss	Anne	G	713	382-7185	0.00
10019	Smith	Olette	K	615	297-3809	453.98

The nulls in the CHARTER table's CHAR_COPILOT column indicate that a copilot is not required for some charter trips or for some aircraft. Federal Aviation Administration (FAA) rules require a copilot on jet aircraft and on aircraft having a gross take-off weight over 12,500 pounds. None of the aircraft in the AIRCRAFT table are governed by this requirement; however, some customers may require the presence of a copilot for insurance reasons. All charter trips are recorded in the CHARTER table.

NOTE

Earlier in the chapter, it was stated that it is best to avoid homonyms and synonyms. In this problem, both the pilot and the copilot are pilots in the PILOT table, but EMP_NUM cannot be used for both in the CHARTER table. Therefore, the synonyms CHAR_PILOT and CHAR_COPILOT were used in the CHARTER table.

Although the solution works in this case, it is very restrictive and it generates nulls when a copilot is not required. Worse, such nulls proliferate as crew requirements change. For example, if the AviaCo charter company grows and starts using larger aircraft, crew requirements may increase to include flight engineers and load masters. The CHARTER table would then have to be modified to include the additional crew assignments; such attributes as CHAR_FLT_ENGINEER and CHAR_LOADMASTER would have to be added to the CHARTER table. Given this change, each time a smaller aircraft flew a charter trip without the number of crew members required in larger aircraft, the missing crew members would yield additional nulls in the CHARTER table.

You will have a chance to correct those design shortcomings in Problem 33. The problem illustrates two important points:

1. Don't use synonyms. If your design requires the use of synonyms, revise the design!
2. To the greatest possible extent, design the database to accommodate growth without requiring structural changes in the database tables. Plan ahead and try to anticipate the effects of change on the database.

30. For each table, where possible, identify:
    a. The primary key.
    b. A superkey.
    c. A candidate key.
    d. The foreign key(s).
    e. A secondary key.
31. Create the ERD. (*Hint*: Look at the table contents. You will discover that an AIRCRAFT can fly many CHARTER trips but that each CHARTER trip is flown by one AIRCRAFT, that a MODEL references many AIRCRAFT but that each AIRCRAFT references a single MODEL, etc.)
32. Create the relational diagram.
33. Modify the ERD you created in Problem 31 to eliminate the problems created by the use of synonyms. (*Hint*: Modify the CHARTER table structure by eliminating the CHAR_PILOT and CHAR_COPILOT attributes; then create a composite table named CREW to link the CHARTER and EMPLOYEE tables. Some crew members, such as flight attendants, may not be pilots. That's why the EMPLOYEE table enters into this relationship.)
34. Create the relational diagram for the design you revised in Problem 33. (After you have had a chance to revise the design, your instructor will show you the results of the design change, using a copy of the revised database named **Ch03_AviaCo_2**.)