PART II

Design Concepts

BP's Data Modeling Initiative

Business Vignette

British Petroleum is one of the largest energy companies in the world, engaged in fuel exploration and production in 29 countries and actively developing alternative energy sources such as solar and wind energy and biofuels. In this large, diverse corporation, management is decentralized and IT expenditure and infrastructure development has historically been project-driven. As a result, BP's Information Technology and Services (IT&S) division was unable to implement uniform IT standards and platforms throughout the company. The company had adopted well over 5,000 software applications.

The decentralized company structure strongly impacted database development. Each project created its own data models. The extent and approach to data modeling differed with each project. Project managers used a large variety of data modeling tools, including System Architecture, ERWin, ARIS, Enterprise Architecture, Visio, and even PowerPoint. Moreover, there was no central repository where models and data definitions could be stored. Once a project was finished, these models were frequently lost. So, BP suffered from inconsistent data definitions, data duplication, and quality problems.

In 2003, BP decided to change all that. The company set a goal to manage data and information "as a shared corporate asset that is easily accessible." It created an Enterprise Architecture team to identify common IT standards. By the end of 2005, the team had conducted a cross-company data modeling study and created a list of agreed upon requirements. The idea was to establish "data modeling as a service" to all business units. The function of the Enterprise Architecture team would not be to enforce standards and procedures, but to train, support, and provide resources.

Since potential users were located all over the globe, the team decided to build a data modeling portal that would house all data modeling related resources: standards and guidelines, discussion boards, registration for trainings, and a large data model repository where data models are automatically uploaded and shared. To support this effort, BP adopted a single data modeling tool, ER/Studio. Users could work in ER/Studio and the data models would automatically be published to Microsoft SharePoint. By 2009, the repository contained 235 models for over 50,000 entities.

The response from users has been very positive. A recent survey found that nearly all users agree that they are benefiting from the use of a common modeling tool, a common repository, and common standards and guidelines. In addition, the number of employees using the portal has increased. These two indicators strongly suggest that BP's "data modeling as a service" strategy is overcoming the disadvantages created by its policies of decentralized management and voluntary adoption.

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

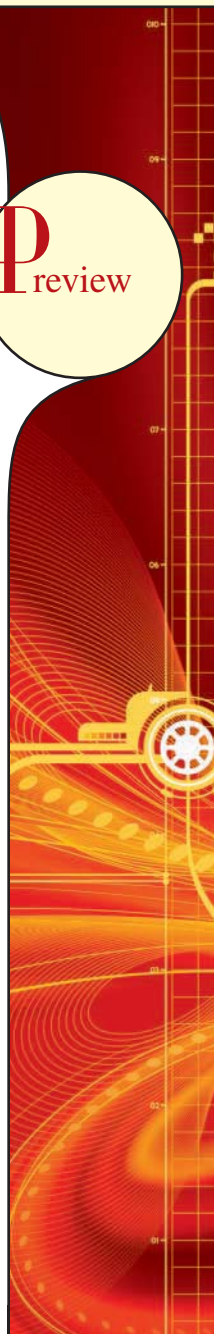
Preview

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 will 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 will also learn how the relational database's basic data components fit into a logical construct known as a table. You will 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 will 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 will be introduced to the basic concepts that shape the design of tables. Because the table is such an integral part of relational database design, you will also learn the characteristics of well-designed and poorly designed tables.

Finally, you will be introduced to some basic concepts that will become your gateway to the next few chapters. For example, you will 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 defines 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 table shown in Figure 3.1 illustrates the characteristics listed in Table 3.1.

FIGURE 3.1 STUDENT table attribute values

Table name: STUDENT **Database name: Ch03_TinyCollege**

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1975	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1981	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1969	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1958	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1986	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = **Student number**
STU_LNAME = **Student last name**
STU_FNAME = **Student first name**
STU_INIT = **Student middle initial**
STU_DOB = **Student date of birth**
STU_HRS = **Credit hours earned**
STU_CLASS = **Student classification**
STU_GPA = **Grade point average**
STU_TRANSFER = **Student transferred from another institution**
DEPT_CODE = **Department code**
STU_PHONE = **4-digit campus phone extension**
PROF_NUM = **Number of the professor who is the student's advisor**

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 Premium Website 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.) For example, row 4 in Figure 3.1 describes a student named Walter H. Oblonski. Given the table contents, the STUDENT entity set includes eight distinct entities (rows), or students.
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, in Figure 3.1, STU_HRS and STU_GPA are numeric attributes.

   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, STU_CLASS and STU_PHONE are examples of character attributes.

   c. *Date.* Date attributes contain calendar dates stored in a special format known as the Julian date format. For example, STU_DOB in Figure 3.1 is a date attribute.

   d. *Logical.* Logical data can only have true or false (yes or no) values. In Figure 3.1, the STU_TRANSFER attribute uses a logical data format.

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 seem 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 when roommates share a single land line phone number.) 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 multiattribute 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 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

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.

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

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 **Database name: Ch03_SaleCo**
Primary key: PROD_CODE
Foreign key: VEND_CODE

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 6, Normalization of Database Tables. For example, the relational schema for Figure 3.2 would be shown as:

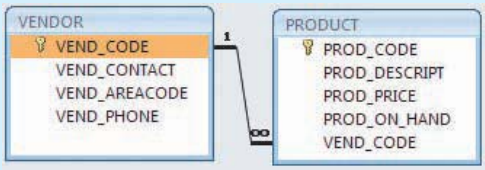
VENDOR (**VEND_CODE**, VEND_CONTACT, VEND_AREACODE, VEND_PHONE)

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,

FIGURE 3.3 The relational diagram for the Ch03_SaleCo database



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.

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.
REFERENCE 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.

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-2010	502
10011	Dunne	Leona	K	713	894-1238	T1	250.00	16-Jun-2010	501
10012	Smith	Kathy	W	615	894-2285	S2	150.00	29-Jan-2011	502
10013	Olowski	Paul	F	615	894-2180	S1	300.00	14-Oct-2010	502
10014	Orlando	Myron		615	222-1672	T1	100.00	28-Dec-2010	501
10015	O'Brian	Amy	B	713	442-3381	T2	850.00	22-Sep-2010	503
10016	Brown	James	G	615	297-1228	S1	120.00	25-Mar-2011	502
10017	Williams	George		615	290-2556	S1	250.00	17-Jul-2010	503
10018	Farriss	Anne	G	713	382-7185	T2	100.00	03-Dec-2010	501
10019	Smith	Olette	K	615	297-3809	S2	500.00	14-Mar-2011	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

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 is also 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.

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), and Chapter 8, Advanced 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 relations (tables) 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. 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 3.5.
2. 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 3.6.

FIGURE 3.5 SELECT

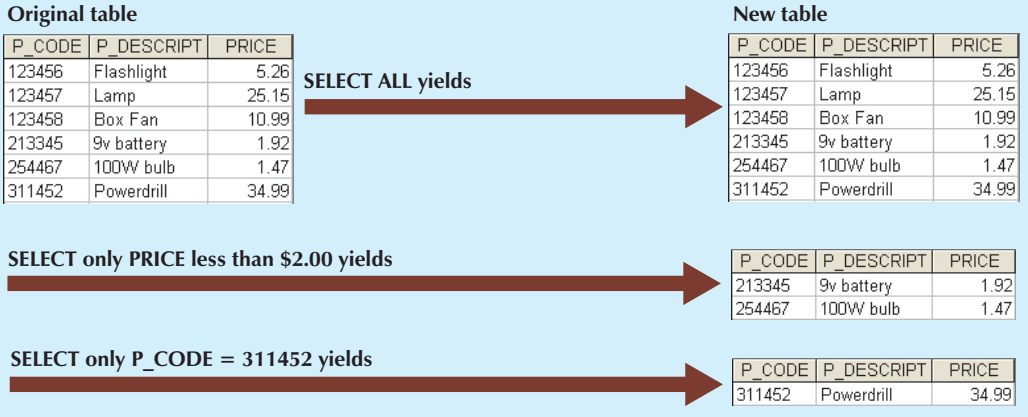
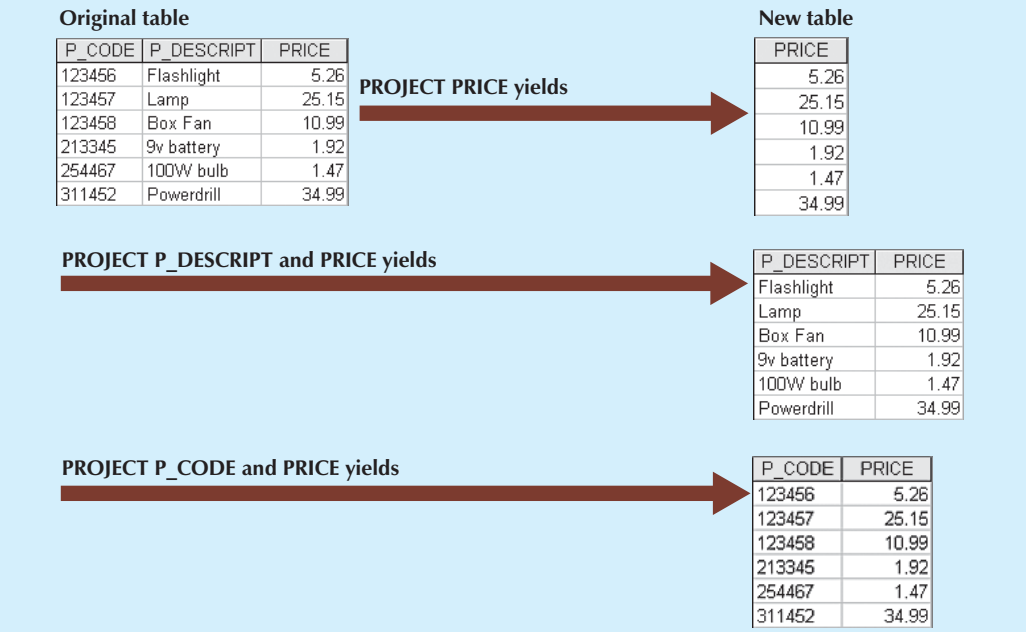


FIGURE 3.6 PROJECT



3. UNION combines all rows from two tables, excluding duplicate rows. The tables must have the same attribute characteristics (the columns and domains must be compatible) to be used in the UNION. When two or more tables share the same number of columns, and when their corresponding columns share the same (or compatible) domains, they are said to be **union-compatible**. The effect of a UNION is shown in Figure 3.7.
4. 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 3.8.

FIGURE 3.7 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

FIGURE 3.8 INTERSECT

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

INTERSECT

EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

yields

STU_FNAME	STU_LNAME
Franklin	Johnson

5. 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 3.9. 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 3.9 DIFFERENCE

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

DIFFERENCE

EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

yields

STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Martin	Lopez

6. 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 3.10.
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 3.10 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

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.

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.

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.

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

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

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

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

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**.

Each of the preceding joins is often classified as an inner join. An **inner join** is a join that only returns matched records from the tables that are being joined. In an **outer join**, the matched pairs would be retained, and any unmatched values in the other table would be left null. It is an easy mistake to think that an outer join is the opposite of an

inner join. However, it is more accurate to think of an outer join as an "inner join plus." The outer join still returns all of the matched records that the inner join returns, plus it returns the unmatched records from one of the tables. More specifically, if an outer join is produced for tables CUSTOMER and AGENT, two scenarios are possible:

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.

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	

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

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.

Generally speaking, outer joins operate like equijoins. The outer join does not drop one copy of the common attribute, and it requires the specification of the join condition. Figures 3.15 and 3.16 illustrate the result of outer joins after a relational PROJECT operation is applied to them to manually remove the duplicate column.

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 8 explores such joins in more detail.

8. The DIVIDE operation uses one single-column table (e.g., 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 (e.g., 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 matches. Figure 3.17 shows a DIVIDE.

FIGURE 3.17 DIVIDE

CODE	LOC
A	5
A	9
A	4
B	5
B	3
C	6
D	7
D	8
E	8

DIVIDE

CODE
A
B

yields

LOC
5

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.

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 27 at the end of this chapter.

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)) are 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.

3.6 RELATIONSHIPS WITHIN THE RELATIONAL DATABASE

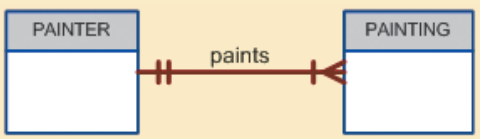
You already know that relationships are classified as one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N or M:M). This section explores those relationships further to help you apply them properly when you start developing database designs, focusing on the following points:

- 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 relationship 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 shown in Figure 3.18.

FIGURE 3.18 The 1:M relationship between PAINTER and PAINTING



Compare the data model in Figure 3.18 with its implementation in Figure 3.19.

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.

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

- 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.

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

NOTE

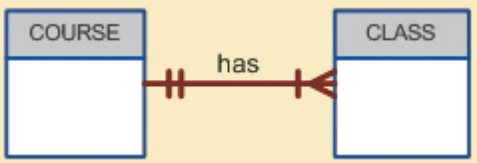
The one-to-many (1:M) relationship is easily implemented in the relational model by putting the *primary key of the 1 side in the table of the many side as a foreign key.*

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 maps the ERM (entity relationship model) for the 1:M relationship between COURSE and CLASS.

FIGURE 3.20 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
Foreign key: none

Database name: Ch03_TinyCollege

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

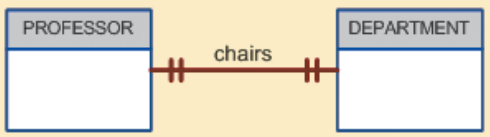
Using Figure 3.21, take a minute to review some important terminology. Note that CLASS_CODE in the CLASS table uniquely identifies each row. Therefore, CLASS_CODE has been chosen to be the primary key. However, the combination CRS_CODE and CLASS_SECTION will also uniquely identify each row in the class 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.

FIGURE 3.22 The 1:1 relationship between PROFESSOR and DEPARTMENT



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.)
- 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.
- 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.

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. In Chapter 5, Advanced Data Modeling, we will explore a concept called a generalization hierarchy, which is a powerful tool for improving our database designs under specific conditions to avoid a proliferation of nulls. One of the characteristics of generalization hierarchies is that they are implemented as 1:1 relationships.

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 3.23 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

ONLINE CONTENT

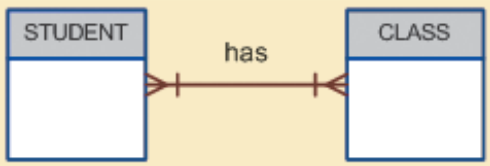
If you open the **Ch03_TinyCollege** database in the Premium Website, you'll see that the STUDENT and CLASS entities still use PROF_NUM as their foreign key. PROF_NUM and EMP_NUM are labels for the same attribute, which is an example of the use of synonyms; that is, different names for the same attribute.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

ONLINE CONTENT

If you look at the **Ch03_AviaCo** database in the Premium Website, you will see the implementation of the 1:1 PILOT to EMPLOYEE relationship. This relationship is based on a concept known as "generalization hierarchy," which you will learn about in Chapter 5.

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.

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



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

Given such a data relationship and the sample data in Table 3.7, you could wrongly assume that you could implement this M:N relationship by simply adding a foreign key in the many side of the relationship that points to the primary key of the related table, as shown in Figure 3.25.

However, the M:N relationship 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.

FIGURE 3.25 The wrong implementation of 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

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 were originally related in an 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 ENROLL table 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 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. You can increase the amount of available information even as you control the database's redundancies. Thus, Figure 3.28

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

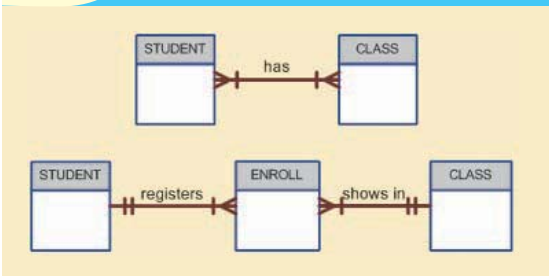
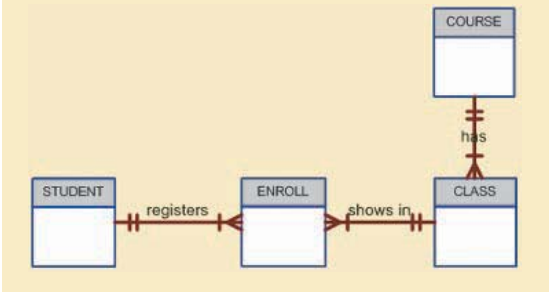


FIGURE 3.28 The expanded entity relationship model

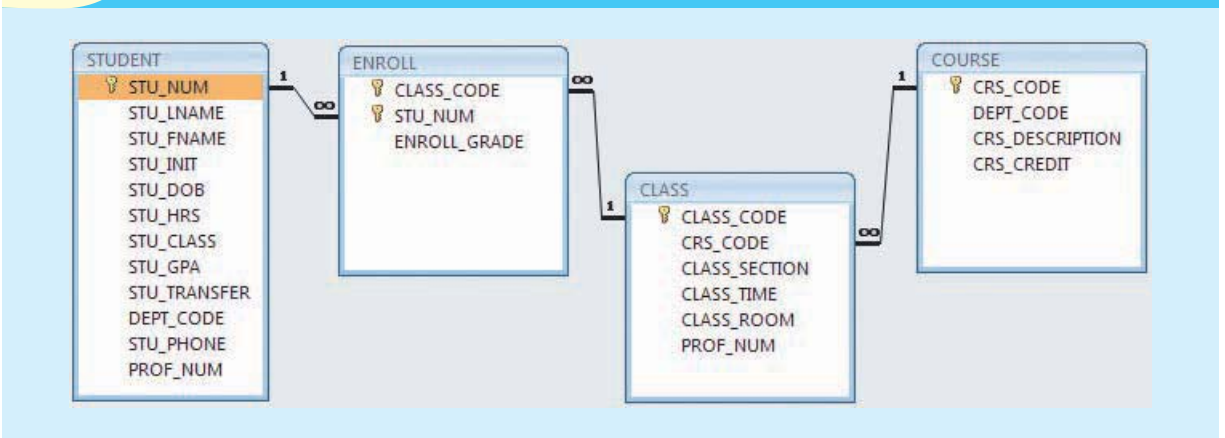


shows the expanded ERM, including the 1:M relationship between COURSE and CLASS shown in Figure 3.20. Note that the model is able to handle multiple sections of a CLASS while controlling redundancies by making sure that all of the COURSE data common to each CLASS are kept in the COURSE table.

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

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

FIGURE 3.29 The relational diagram for the Ch03_TinyCollege database



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.

FIGURE 3.30 A small invoicing system

Table name: CUSTOMER **Database name: Ch03_SaleCo**
Primary key: CUS_CODE
Foreign key: none

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-10
1002	10011	08-Mar-10
1003	10012	08-Mar-10
1004	10011	09-Mar-10

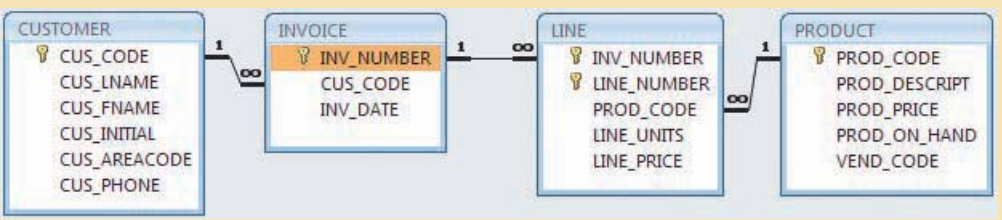
Table name: LINE
Primary key: INV_NUMBER + LINE_NUMBER
Foreign keys: INV_NUMBER, PROD_CODE

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 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, 2010, 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.

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, which 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 common practice on invoicing software that typically 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.

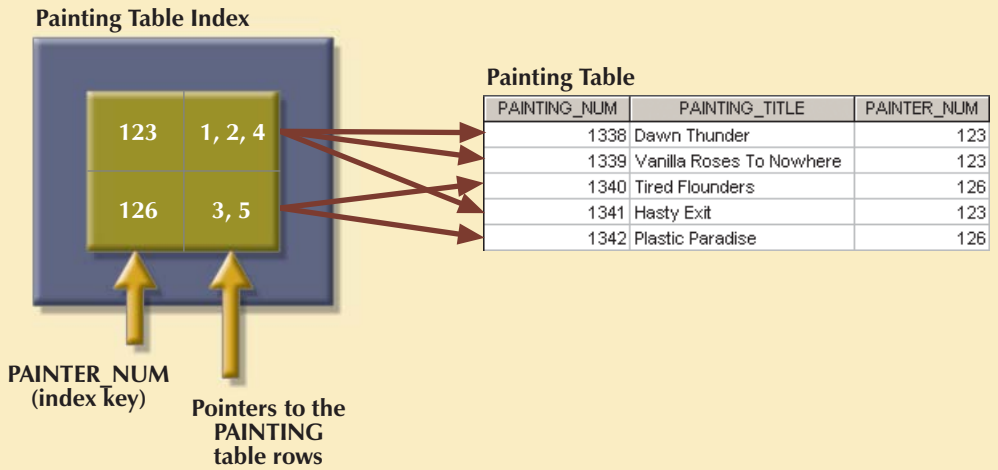
Or suppose you want to find a topic, such as "ER model," in this book. Does it make sense to read through every page until you stumble across the topic? Of course not; it is much simpler to go to the book's index, look up the phrase *ER model*, and read the page references that point you to the appropriate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

FIGURE 3.32 Components of an index



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

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 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 will 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.

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 Online 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 columns 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.

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

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, 81
attribute domain, 60
bridge entity, 81
candidate key, 64
closure, 68
composite entity, 81
composite key, 63
data dictionary, 74
determination, 62
domain, 61
entity integrity, 64
equijoin, 72
flags, 68
foreign key (FK), 65
full functional dependence, 63
functional dependence, 62
homonym, 74
index, 86
index key, 86
inner join, 72
join column(s), 71
key, 62
key attribute, 63
left outer join, 73
linking table, 82
natural join, 71
null, 64
outer join, 72
predicate logic, 59
primary key (PK), 61
referential integrity, 66
relational algebra, 68
relational schema, 65
right outer join, 73
secondary key, 66
set theory, 59
superkey, 63
synonym, 74
system catalog, 74
theta join, 72
tuple, 60
union-compatible, 69
unique index, 87

ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Premium Website 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. What are the requirements that two relations must satisfy in order to be considered union-compatible?
5. Which relational algebra operators can be applied to a pair of tables that are not union-compatible?
6. Explain why the data dictionary is sometimes called "the database designer's database."
7. 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.

ONLINE CONTENT

All of the databases used in the questions and problems are found in the Premium Website 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.

Use Figure Q3.8 to answer Questions 8–10.

FIGURE Q3.8 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

8. Using the STUDENT and PROFESSOR tables, illustrate the difference between a natural join, an equijoin, and an outer join.
9. Create the basic ERD for the database shown in Figure Q3.8.
10. Create the relational diagram for the database shown in Figure Q3.8.

Use Figure Q3.11 to answer Questions 11–13.

11. Create the table that results from applying a UNION relational operator to the tables shown in Figure Q3.11.
12. Create the table that results from applying an INTERSECT relational operator to the tables shown in Figure Q3.11.
13. Using the tables in Figure Q3.11, create the table that results from MACHINE DIFFERENCE BOOTH.
14. Suppose you have the ERM shown in Figure Q3.14. How would you convert this model into an ERM that displays only 1:M relationships? (Make sure you create the revised ERM.)

FIGURE Q3.11 The Ch03_VendingCo database tables

Database name: Ch03_VendingCo

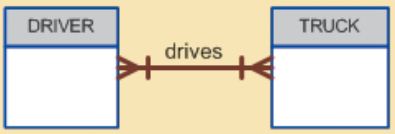
Table name: BOOTH

BOOTH_PRODUCT	BOOTH_PRICE
Chips	1.5
Cola	1.25
Energy Drink	2

Table name: MACHINE

MACHINE_PRODUCT	MACHINE_PRICE
Chips	1.25
Chocolate Bar	1
Energy Drink	2

FIGURE Q3.14 The Crow's Foot ERM for Question 14



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

15. What are homonyms and synonyms, and why should they be avoided in database design?
16. How would you implement a 1:M relationship in a database composed of two tables? Give an example.
17. Identify and describe the components of the table shown in Figure Q3.17, using correct terminology. Use your knowledge of naming conventions to identify the table's probable foreign key(s).

FIGURE Q3.17 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 shown in Figure Q3.18 to answer Questions 18–23.

18. Identify the primary keys.
19. Identify the foreign keys.
20. Create the ERM.

FIGURE Q3.18 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

21. Create the relational diagram to show the relationship between DIRECTOR and PLAY.
22. 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?
23. What would be the conceptual view of the INDEX table that is described in Question 22? Depict the contents of the conceptual INDEX table.

PROBLEMS

Use the database shown in Figure P3.1 to answer Problems 1–9.

1. 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		

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

TABLE	ENTITY INTEGRITY	EXPLANATION
EMPLOYEE		
STORE		
REGION		

3. 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	FOREIGN KEY(S)
EMPLOYEE		
STORE		
REGION		

4. Describe the type(s) of relationship(s) between STORE and REGION.
5. Create the ERD to show the relationship between STORE and REGION.
6. Create the relational diagram to show the relationship between STORE and REGION.

FIGURE P3.1 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

7. Describe the type(s) of relationship(s) between EMPLOYEE and STORE. (*Hint:* Each store employs many employees, one of whom manages the store.)
8. Create the ERD to show the relationships among EMPLOYEE, STORE, and REGION.
9. Create the relational diagram to show the relationships among EMPLOYEE, STORE, and REGION.

Use the database shown in Figure P3.10 to work Problems 10–16. 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 P3.10 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

10. 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		

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

TABLE	ENTITY INTEGRITY	EXPLANATION
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

14. 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		

15. Create the ERD to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.

16. Create the relational diagram to show the relationships among EMPLOYEE, BENEFIT, JOB, and PLAN.

Use the database shown in Figure P3.17 to answer Problems 17–23.

FIGURE P3.17 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

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)
TRUCK		
BASE		
TYPE		

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

TABLE	ENTITY INTEGRITY	EXPLANATION
TRUCK		
BASE		
TYPE		

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
TRUCK		
BASE		
TYPE		

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

TABLE	SUPERKEY	SECONDARY KEY
TRUCK		
BASE		
TYPE		

22. Create the ERD for this database.
23. Create the relational diagram for this database.

Use the database shown in Figure P3.24 to answer Problems 24–31. ROBCOR is an aircraft charter company that supplies on-demand charter flight services using a fleet of four aircraft. Aircraft are identified by a unique registration number. Therefore, the aircraft registration number is an appropriate primary key for the AIRCRAFT table.

FIGURE P3.24 **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-10	104		ATL	936.0	5.1	2.2	10011
10002	05-Feb-10	101		BNA	320.0	1.6	0.0	10016
10003	05-Feb-10	105	109	GNV	1574.0	7.8	0.0	10014
10004	06-Feb-10	106		STL	472.0	2.9	4.9	10019
10005	06-Feb-10	101		ATL	1023.0	5.7	3.5	10011
10006	06-Feb-10	109		STL	472.0	2.6	6.2	10017
10007	06-Feb-10	104	105	GNV	1574.0	7.9	0.0	10012
10008	07-Feb-10	106		TYS	644.0	4.1	0.0	10014
10009	07-Feb-10	105		GNV	1574.0	6.6	23.4	10017
10010	07-Feb-10	109		ATL	936.0	6.2	3.2	10016
10011	07-Feb-10	101	104	BNA	352.0	1.9	5.3	10012
10012	08-Feb-10	101		MOB	864.0	4.8	4.2	10010
10013	08-Feb-10	105		TYS	644.0	3.9	4.5	10011
10014	09-Feb-10	106		ATL	936.0	5.1	2.1	10017
10015	09-Feb-10	104	101	GNV	1646.0	6.7	0.0	10016
10016	09-Feb-10	109	105	MQY	312.0	1.5	0.0	10011
10017	10-Feb-10	101		STL	508.0	3.1	0.0	10014
10018	10-Feb-10	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 were 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_SEATS 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 P3.24 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-10	11-Jan-10
104	ATP	ATP/SEL/MEL/Instr	1	18-Dec-09	17-Jan-10
105	COM	COMM/SEL/MEL/Instr/CFI	2	05-Jan-10	02-Jan-10
106	COM	COMM/SEL/MEL/Instr	2	10-Dec-09	02-Feb-10
109	COM	ATP/SEL/MEL/SES/Instr/CFII	1	22-Jan-10	15-Jan-10

The pilot licenses shown in the PILOT table include the ATP = Airline Transport Pilot and COMM = 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 are 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 is 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 27. 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.

24. 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.
25. 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.)
26. Create the relational diagram.
27. Modify the ERD you created in Problem 25 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.)
28. Create the relational diagram for the design you revised in Problem 27. (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**.)

You are interested in seeing data on charters flown by either Mr. Robert Williams (employee number 105) or Ms. Elizabeth Travis (employee number 109) as pilot or copilot, but not charters flown by both of them. Complete problems 29–31 to find these data.

29. Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR_TRIP, CHAR_PILOT, and CHAR_COPILOT attributes for charters flown by either employee 104 or employee 109.
30. Create the table that would result from applying the SELECT and PROJECT relational operators to the CHARTER table to return only the CHAR_TRIP, CHAR_PILOT, and CHAR_COPILOT attributes for charters flown by both employee 104 and employee 109.
31. Create the table that would result from applying a DIFFERENCE relational operator of your result from problem 29 to your result from problem 30.

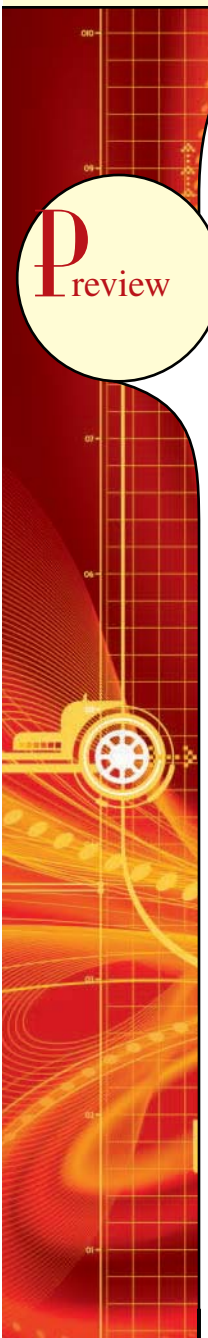
Entity Relationship (ER) Modeling

4

FOUR

In this chapter, you will learn:

- The main characteristics of entity relationship components
- How relationships between entities are defined, refined, and incorporated into the database design process
- How ERD components affect database design and implementation
- That real-world database design often requires the reconciliation of conflicting goals



Preview

This chapter expands coverage of the data-modeling aspect of database design. Data modeling is the first step in the database design journey, serving as a bridge between real-world objects and the database model that is implemented in the computer. Therefore, the importance of data-modeling details, expressed graphically through entity relationship diagrams (ERDs), cannot be overstated.

Most of the basic concepts and definitions used in the entity relationship model (ERM) were introduced in Chapter 2, Data Models. For example, the basic components of entities and relationships and their representation should now be familiar to you. This chapter goes much deeper and further, analyzing the graphic depiction of relationships among the entities and showing how those depictions help you summarize the wealth of data required to implement a successful design.

Finally, the chapter illustrates how conflicting goals can be a challenge in database design, possibly requiring you to make design compromises.

NOTE

Because this book generally focuses on the relational model, you might be tempted to conclude that the ERM is exclusively a relational tool. Actually, conceptual models such as the ERM can be used to understand and design the data requirements of an organization. Therefore, the ERM is independent of the database type. Conceptual models are used in the conceptual design of databases, while relational models are used in the logical design of databases. However, because you are now familiar with the relational model from the previous chapter, the relational model is used extensively in this chapter to explain ER constructs and the way they are used to develop database designs.

4.1 THE ENTITY RELATIONSHIP MODEL (ERM)

You should remember from Chapter 2, Data Models, and Chapter 3, The Relational Database Model, that the ERM forms the basis of an ERD. The ERD represents the conceptual database as viewed by the end user. ERDs depict the database's main components: entities, attributes, and relationships. Because an entity represents a real-world object, the words *entity* and *object* are often used interchangeably. Thus, the entities (objects) of the Tiny College database design developed in this chapter include students, classes, teachers, and classrooms. The order in which the ERD components are covered in the chapter is dictated by the way the modeling tools are used to develop ERDs that can form the basis for successful database design and implementation.

In Chapter 2, you also learned about the various notations used with ERDs—the original Chen notation and the newer Crow's Foot and UML notations. The first two notations are used at the beginning of this chapter to introduce some basic ER modeling concepts. Some conceptual database modeling concepts can be expressed only using the Chen notation. However, because the emphasis is on *design and implementation* of databases, the Crow's Foot and UML class diagram notations are used for the final Tiny College ER diagram example. Because of its implementation emphasis, the Crow's Foot notation can represent only what could be implemented. In other words:

- The Chen notation favors conceptual modeling.
- The Crow's Foot notation favors a more implementation-oriented approach.
- The UML notation can be used for both conceptual and implementation modeling.

ONLINE CONTENT

To learn how to create ER diagrams with the help of Microsoft Visio, see the Premium Website for this book:

- **Appendix A, Designing Databases with Visio Professional: A Tutorial** shows you how to create Crow's Foot ERDs.
- **Appendix H, Unified Modeling Language (UML)**, shows you how to create UML class diagrams.

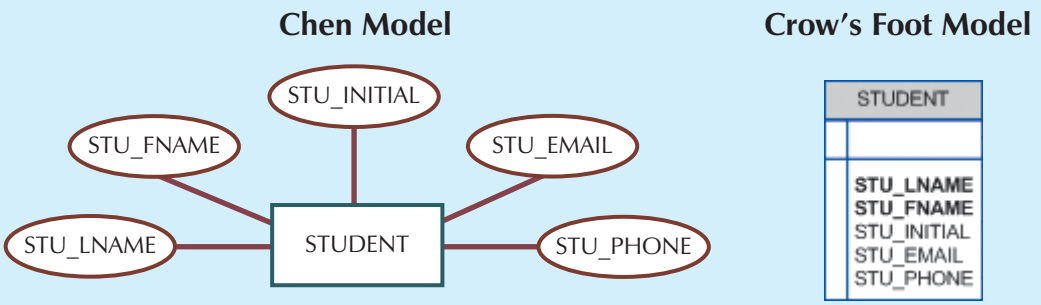
4.1.1 ENTITIES

Recall that an entity is an object of interest to the end user. In Chapter 2, you learned that at the ER modeling level, an entity actually refers to the *entity set* and not to a single entity occurrence. In other words, the word *entity* in the ERM corresponds to a table—not to a row—in the relational environment. The ERM refers to a table row as an *entity instance* or *entity occurrence*. In both the Chen and Crow's Foot notations, an entity is represented by a rectangle containing the entity's name. The entity name, a noun, is usually written in all capital letters.

4.1.2 ATTRIBUTES

Attributes are characteristics of entities. For example, the STUDENT entity includes, among many others, the attributes STU_LNAME, STU_FNAME, and STU_INITIAL. In the original Chen notation, attributes are represented by ovals and are connected to the entity rectangle with a line. Each oval contains the name of the attribute it represents. In the Crow's Foot notation, the attributes are written in the attribute box below the entity rectangle. (See Figure 4.1.) Because the Chen representation is rather space-consuming, software vendors have adopted the Crow's Foot attribute display.

FIGURE 4.1 The attributes of the STUDENT entity: Chen and Crow's Foot



Required and Optional Attributes

A **required attribute** is an attribute that must have a value; in other words, it cannot be left empty. As shown in Figure 4.1, there are two boldfaced attributes in the Crow's Foot notation. This indicates that a data entry will be required. In this example, STU_LNAME and STU_FNAME require data entries because of the assumption that all students have a last name and a first name. But students might not have a middle name, and perhaps they do not (yet) have a phone number and an e-mail address. Therefore, those attributes are not presented in boldface in the entity box. An **optional attribute** is an attribute that does not require a value; therefore, it can be left empty.

Domains

Attributes have a domain. As you learned in Chapter 3, a *domain* is the set of possible values for a given attribute. For example, the domain for the grade point average (GPA) attribute is written (0,4) because the lowest possible GPA value is 0 and the highest possible value is 4. The domain for the gender attribute consists of only two possibilities: M or F (or some other equivalent code). The domain for a company's date of hire attribute consists of all dates that fit in a range (for example, company startup date to current date).

Attributes may share a domain. For instance, a student address and a professor address share the same domain of all possible addresses. In fact, the data dictionary may let a newly declared attribute inherit the characteristics of an existing attribute if the same attribute name is used. For example, the PROFESSOR and STUDENT entities may each have an attribute named ADDRESS and could therefore share a domain.

Identifiers (Primary Keys)

The ERM uses **identifiers**, that is, one or more attributes that uniquely identify each entity instance. In the relational model, such identifiers are mapped to primary keys (PKs) in tables. Identifiers are underlined in the ERD. Key attributes are also underlined in a frequently used table structure shorthand notation using the format:

TABLE NAME (**KEY_ATTRIBUTE 1**, ATTRIBUTE 2, ATTRIBUTE 3, . . . ATTRIBUTE K)

For example, a CAR entity may be represented by:

CAR (**CAR_VIN**, MOD_CODE, CAR_YEAR, CAR_COLOR)

(Each car is identified by a unique vehicle identification number, or CAR_VIN.)

Composite Identifiers

Ideally, an entity identifier is composed of only a single attribute. For example, the table in Figure 4.2 uses a single-attribute primary key named CLASS_CODE. However, it is possible to use a **composite identifier**, that is, a primary key composed of more than one attribute. For instance, the Tiny College database administrator may decide to identify each CLASS entity instance (occurrence) by using a composite primary key composed of the combination of CRS_CODE and CLASS_SECTION instead of using CLASS_CODE. Either approach uniquely identifies each entity instance. Given the current structure of the CLASS table shown in Figure 4.2, CLASS_CODE is the primary key, and the combination of CRS_CODE and CLASS_SECTION is a proper candidate key. If the CLASS_CODE attribute is deleted from the CLASS entity, the candidate key (CRS_CODE and CLASS_SECTION) becomes an acceptable composite primary key.

FIGURE 4.2 **The CLASS table (entity) components and contents**

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	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
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

NOTE

Remember that Chapter 3 made a commonly accepted distinction between COURSE and CLASS. A CLASS constitutes a specific time and place of a COURSE offering. A class is defined by the course description and its time and place, or section. Consider a professor who teaches Database I, Section 2; Database I, Section 5; Database I, Section 8; and Spreadsheet II, Section 6. That instructor teaches two courses (Database I and Spreadsheet II), but four classes. Typically, the COURSE offerings are printed in a course catalog, while the CLASS offerings are printed in a class schedule for each semester, trimester, or quarter.

If the CLASS_CODE in Figure 4.2 is used as the primary key, the CLASS entity may be represented in shorthand form by:

CLASS (**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

On the other hand, if CLASS_CODE is deleted, and the composite primary key is the combination of CRS_CODE and CLASS_SECTION, the CLASS entity may be represented by:

CLASS (**CRS_CODE**, **CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

Note that *both* key attributes are underlined in the entity notation.

Composite and Simple Attributes

Attributes are classified as simple or composite. A **composite attribute**, not to be confused with a composite key, is an attribute that can be further subdivided to yield additional attributes. For example, the attribute ADDRESS can be subdivided into street, city, state, and zip code. Similarly, the attribute PHONE_NUMBER can be subdivided into area code and exchange number. A **simple attribute** is an attribute that cannot be subdivided. For example, age, sex, and marital status would be classified as simple attributes. To facilitate detailed queries, it is wise to change composite attributes into a series of simple attributes.

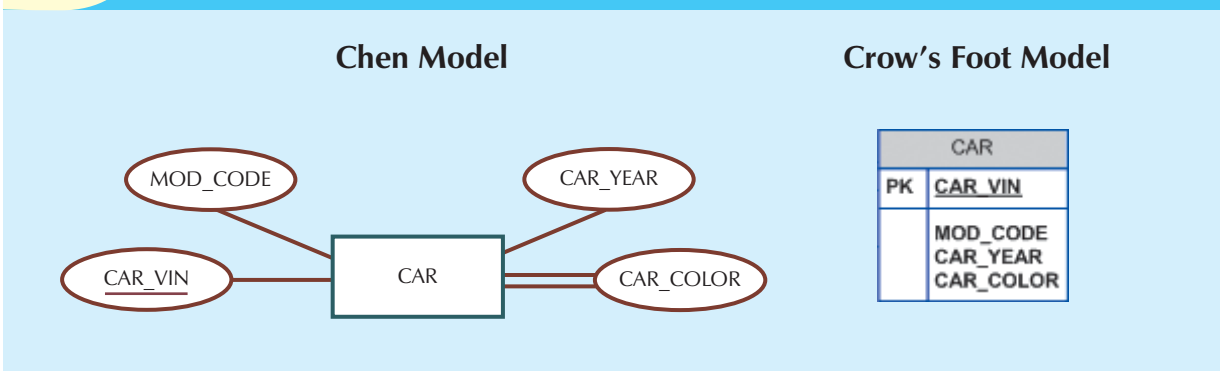
Single-Valued Attributes

A **single-valued attribute** is an attribute that can have only a single value. For example, a person can have only one Social Security number, and a manufactured part can have only one serial number. *Keep in mind that a single-valued attribute is not necessarily a simple attribute*. For instance, a part's serial number, such as SE-08-02-189935, is single-valued, but it is a composite attribute because it can be subdivided into the region in which the part was produced (SE), the plant within that region (08), the shift within the plant (02), and the part number (189935).

Multivalued Attributes

Multivalued attributes are attributes that can have many values. For instance, a person may have several college degrees, and a household may have several different phones, each with its own number. Similarly, a car's color may be subdivided into many colors (that is, colors for the roof, body, and trim). In the Chen ERM, the multivalued attributes are shown by a double line connecting the attribute to the entity. The Crow's Foot notation does not identify multivalued attributes. The ERD in Figure 4.3 contains all of the components introduced thus far. In Figure 4.3, note that CAR_VIN is the primary key, and CAR_COLOR is a multivalued attribute of the CAR entity.

FIGURE 4.3 **A multivalued attribute in an entity**



NOTE

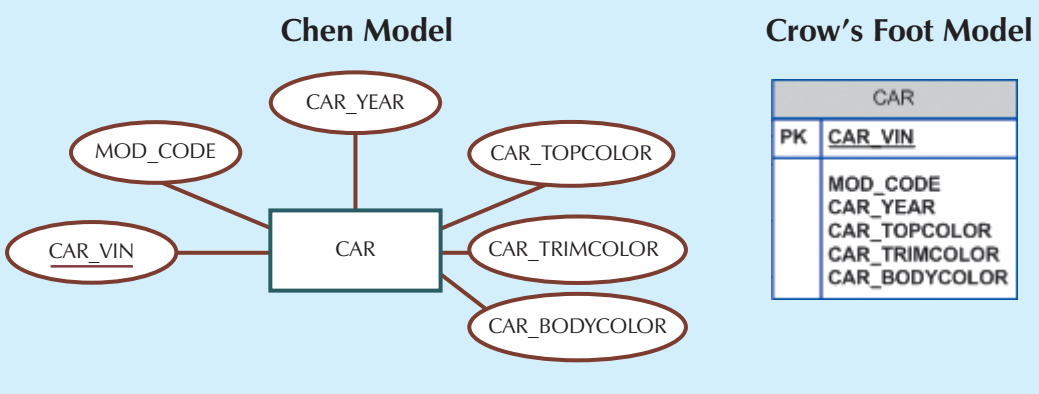
In the ERD models in Figure 4.3, the CAR entity's foreign key (FK) has been typed as MOD_CODE. This attribute was manually added to the entity. Actually, proper use of database modeling software will automatically produce the FK when the relationship is defined. In addition, the software will label the FK appropriately and write the FK's implementation details in a data dictionary. Therefore, when you use database modeling software like Visio Professional, *never type the FK attribute yourself*; let the software handle that task when the relationship between the entities is defined. (You can see how that's done in **Appendix A, Designing Databases with Visio Professional: A Tutorial**, in the Premium Website.)

Implementing Multivalued Attributes

Although the conceptual model can handle M:N relationships and multivalued attributes, *you should not implement them in the RDBMS.* Remember from Chapter 3 that in the relational table, each column/row intersection represents a single data value. So if multivalued attributes exist, the designer must decide on one of two possible courses of action:

1. Within the original entity, create several new attributes, one for each of the original multivalued attribute's components. For example, the CAR entity's attribute CAR_COLOR can be split to create the new attributes CAR_TOPCOLOR, CAR_BODYCOLOR, and CAR_TRIMCOLOR, which are then assigned to the CAR entity. (See Figure 4.4.)

FIGURE 4.4 **Splitting the multivalued attribute into new attributes**



Although this solution seems to work, its adoption can lead to major structural problems in the table. For example, if additional color components—such as a logo color—are added for some cars, the table structure must be modified to accommodate the new color section. In that case, cars that do not have such color sections generate nulls for the nonexisting components, or their color entries for those sections are entered as N/A to indicate "not applicable." (Imagine how the solution in Figure 4.4—splitting a multivalued attribute into new attributes—would cause problems if it were applied to an employee entity containing employee degrees and certifications. If some employees have 10 degrees and certifications while most have fewer or none, the number of degree/certification attributes would number 10, and most of those attribute values would be null for most of the employees.) In short, although you have seen solution 1 applied, it is not an acceptable solution.

2. Create a new entity composed of the original multivalued attribute's components. This new entity allows the designer to define color for different sections of the car. (See Table 4.1.) Then, this new CAR_COLOR entity is related to the original CAR entity in a 1:M relationship.

TABLE 4.1 Components of the Multivalued Attribute

SECTION	COLOR
Top	White
Body	Blue
Trim	Gold
Interior	Blue

Using the approach illustrated in Table 4.1, you even get a fringe benefit: you are now able to assign as many colors as necessary without having to change the table structure. Note that the ERM shown in Figure 4.5 reflects the components listed in Table 4.1. This is the preferred way to deal with multivalued attributes. Creating a new entity in a 1:M relationship with the original entity yields several benefits: it's a more flexible, expandable solution, and it is compatible with the relational model!

FIGURE 4.5 A new entity set composed of a multivalued attribute's components

Derived Attributes

Finally, an attribute may be classified as a derived attribute. A **derived attribute** is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. For example, an employee's age, EMP_AGE, may be found by computing the integer value of the difference between the current date and the EMP_DOB. If you use Microsoft Access, you would use the formula INT((DATE() – EMP_DOB)/365). In Microsoft SQL Server, you would use SELECT DATEDIFF("YEAR", EMP_DOB, GETDATE()), where DATEDIFF is a function that computes the difference between dates. The first parameter indicates the measurement, in this case, years.

If you use Oracle, you would use SYSDATE instead of DATE(). (You are assuming, of course, that the EMP_DOB was stored in the Julian date format.) Similarly, the total cost of an order can be derived by multiplying the quantity ordered by the unit price. Or the estimated average speed can be derived by dividing trip distance by the time spent en route. A derived attribute is indicated in the Chen notation by a dashed line connecting the attribute and the entity. (See Figure 4.6.) The Crow's Foot notation does not have a method for distinguishing the derived attribute from other attributes.

Derived attributes are sometimes referred to as *computed attributes*. A derived attribute computation can be as simple as adding two attribute values located on the same row, or it can be the result of aggregating the sum of values located on many table rows (from the same table or from a different table). The decision to store derived attributes in database tables depends on the processing requirements and the constraints placed on a particular application. The designer should be able to balance the design in accordance with such constraints. Table 4.2 shows the advantages and disadvantages of storing (or not storing) derived attributes in the database.

4.1.3 Relationships

Recall from Chapter 2 that a relationship is an association between entities. The entities that participate in a relationship are also known as **participants**, and each relationship is identified by a name that describes the relationship. The relationship name is an active or passive verb; for example, a STUDENT *takes* a CLASS, a PROFESSOR *teaches* a CLASS, a DEPARTMENT *employs* a PROFESSOR, a DIVISION *is managed by* an EMPLOYEE, and an AIRCRAFT *is flown by* a CREW.

FIGURE 4.6 Depiction of a derived attribute

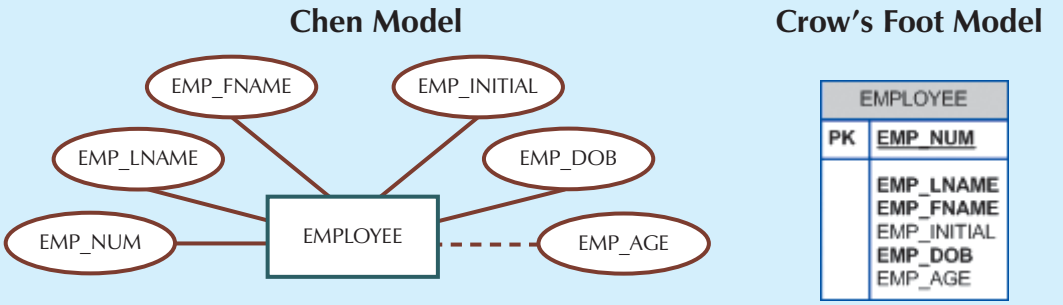


TABLE 4.2 Advantages and Disadvantages of Storing Derived Attributes

	DERIVED ATTRIBUTE	
	STORED	NOT STORED
Advantage	Saves CPU processing cycles Saves data access time Data value is readily available Can be used to keep track of historical data	Saves storage space Computation always yields current value
Disadvantage	Requires constant maintenance to ensure derived value is current, especially if any values used in the calculation change	Uses CPU processing cycles Increases data access time Adds coding complexity to queries

Relationships between entities always operate in both directions. That is, to define the relationship between the entities named CUSTOMER and INVOICE, you would specify that:

- A CUSTOMER may generate many INVOICEs.
- Each INVOICE is generated by one CUSTOMER.

Because you know both directions of the relationship between CUSTOMER and INVOICE, it is easy to see that this relationship can be classified as 1:M.

The relationship classification is difficult to establish if you know only one side of the relationship. For example, if you specify that:

A DIVISION is managed by one EMPLOYEE.

You don't know if the relationship is 1:1 or 1:M. Therefore, you should ask the question "Can an employee manage more than one division?" If the answer is yes, the relationship is 1:M, and the second part of the relationship is then written as:

An EMPLOYEE may manage many DIVISIONs.

If an employee cannot manage more than one division, the relationship is 1:1, and the second part of the relationship is then written as:

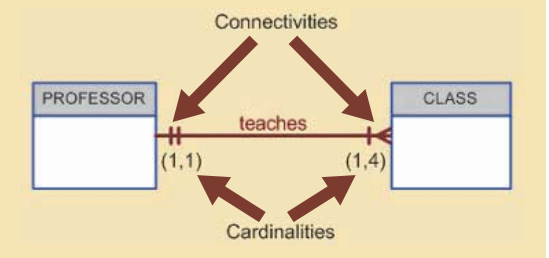
An EMPLOYEE may manage only one DIVISION.

4.1.4 Connectivity and Cardinality

You learned in Chapter 2 that entity relationships may be classified as one-to-one, one-to-many, or many-to-many. You also learned how such relationships were depicted in the Chen and Crow's Foot notations. The term **connectivity** is used to describe the relationship classification.

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (x,y). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities. Many database designers who use Crow's Foot modeling notation do not depict the specific cardinalities on the ER diagram itself because the specific limits described by the cardinalities cannot be implemented directly through the database design. Correspondingly, some Crow's Foot ER modeling tools do not print the numeric cardinality range in the diagram; instead, you can add it as text if you want to have it shown. When the specific cardinalities are not included on the diagram in Crow's Foot notation, cardinality is implied by the use of the symbols shown in Figure 4.7, which describe the connectivity and participation (discussed below). The numeric cardinality range has been added using the Visio text drawing tool.

FIGURE 4.7 Connectivity and cardinality in an ERD



Knowing the minimum and maximum number of entity occurrences is very useful at the application software level. For example, Tiny College might want to ensure that a class is not taught unless it has at least 10 students enrolled. Similarly, if the classroom can hold only 30 students, the application software should use that cardinality to limit enrollment in the class. However, keep in mind that the DBMS cannot handle the implementation of the cardinalities at the table level—that capability is provided by the application software or by triggers. You will learn how to create and execute triggers in Chapter 8, Advanced SQL.

As you examine the Crow's Foot diagram in Figure 4.7, keep in mind that the cardinalities represent the number of occurrences in the *related* entity. For example, the cardinality (1,4) written next to the CLASS entity in the "PROFESSOR teaches CLASS" relationship indicates that each professor teaches up to four classes, which means that the PROFESSOR table's primary key value occurs at least once and no more than four times as foreign key values in the CLASS table. If the cardinality had been written as (1,N), there would be no upper limit to the number of classes a professor might teach. Similarly, the cardinality (1,1) written next to the PROFESSOR entity indicates that each class is taught by one and only one professor. That is, each CLASS entity occurrence is associated with one and only one entity occurrence in PROFESSOR.

Connectivities and cardinalities are established by very concise statements known as business rules, which were introduced in Chapter 2. Such rules, derived from a precise and detailed description of an organization's data environment, also establish the ERM's entities, attributes, relationships, connectivities, cardinalities, and constraints. Because business rules define the ERM's components, making sure that all appropriate business rules are identified is a very important part of a database designer's job.

NOTE

The placement of the cardinalities in the ER diagram is a matter of convention. The Chen notation places the cardinalities on the side of the related entity. The Crow's Foot and UML diagrams place the cardinalities next to the entity to which the cardinalities apply.

ONLINE CONTENT

Because the careful definition of complete and accurate business rules is crucial to good database design, their derivation is examined in detail in **Appendix B, The University Lab: Conceptual Design.** The modeling skills you are learning in this chapter are applied in the development of a real database design in Appendix B. The initial design shown in Appendix B is then modified in **Appendix C, The University Lab: Conceptual Design Verification, Logical Design, and Implementation.** (Both appendixes are found in the Premium Website.)

4.1.5 EXISTENCE DEPENDENCE

An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with another related entity occurrence. In implementation terms, an entity is existence-dependent if it has a mandatory foreign key—that is, a foreign key attribute that cannot be null. For example, if an employee wants to claim one or more dependents for tax-withholding purposes, the relationship "EMPLOYEE claims DEPENDENT" would be appropriate. In that case, the DEPENDENT entity is clearly existence-dependent on the EMPLOYEE entity because it is impossible for the dependent to exist apart from the EMPLOYEE in the database.

If an entity can exist apart from all of its related entities (it is **existence-independent)**, then that entity is referred to as a **strong entity** or **regular entity**. For example, suppose that the XYZ Corporation uses parts to produce its products. Furthermore, suppose that some of those parts are produced in-house and other parts are bought from vendors. In that scenario, it is quite possible for a PART to exist independently from a VENDOR in the relationship "PART is supplied by VENDOR," because at least some of the parts are not supplied by a vendor. Therefore, PART is existence-independent from VENDOR.

NOTE

The relationship strength concept is not part of the original ERM. Instead, this concept applies directly to Crow's Foot diagrams. Because Crow's Foot diagrams are used extensively to design relational databases, it is important to understand relationship strength as it affects database implementation. The Chen ERD notation is oriented toward conceptual modeling and therefore does not distinguish between weak and strong relationships.

4.1.6 RELATIONSHIP STRENGTH

The concept of relationship strength is based on how the primary key of a related entity is defined. To implement a relationship, the primary key of one entity appears as a foreign key in the related entity. For example, the 1:M relationship between VENDOR and PRODUCT in Chapter 3, Figure 3.3, is implemented by using the VEND_CODE primary key in VENDOR as a foreign key in PRODUCT. There are times when the foreign key also is a primary key component in the related entity. For example, in Figure 4.5, the CAR entity primary key (CAR_VIN) appears as both a primary key component and a foreign key in the CAR_COLOR entity. In this section, you will learn how various relationship strength decisions affect primary key arrangement in database design.

Weak (Non-identifying) Relationships

A **weak relationship**, also known as a **non-identifying relationship**, exists if the PK of the related entity does not contain a PK component of the parent entity. By default, relationships are established by having the PK of the parent entity appear as an FK on the related entity. For example, suppose that the COURSE and CLASS entities are defined as:

COURSE(**CRS_CODE**, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS(**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

In this case, a weak relationship exists between COURSE and CLASS because the CLASS_CODE is the CLASS entity's PK, while the CRS_CODE in CLASS is only an FK. In this example, the CLASS PK did not inherit the PK component from the COURSE entity.

Figure 4.8 shows how the Crow's Foot notation depicts a weak relationship by placing a dashed relationship line between the entities. The tables shown below the ERD illustrate how such a relationship is implemented.

FIGURE 4.8 **A weak (non-identifying) relationship between COURSE and CLASS**

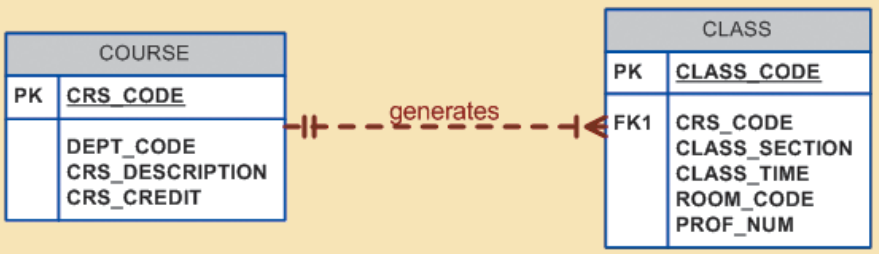


Table name: COURSE **Database name: Ch04_TinyCollege**

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
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	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
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

ONLINE CONTENT

All of the databases used to illustrate the material in this chapter are found in the Premium Website.

NOTE

If you are used to looking at relational diagrams such as the ones produced by Microsoft Access, you expect to see the relationship line *in the relational diagram* drawn from the PK to the FK. However, the relational diagram convention is not necessarily reflected in the ERD. In an ERD, the focus is on the entities and the relationships between them, rather than on the way those relationships are anchored graphically. You will discover that the placement of the relationship lines in a complex ERD that includes both horizontally and vertically placed entities is largely dictated by the designer's decision to improve the readability of the design. (Remember that the ERD is used for communication between the designer(s) and end users.)

Strong (Identifying) Relationships

A **strong relationship**, also known as an **identifying relationship**, exists when the PK of the related entity contains a PK component of the parent entity. For example, the definitions of the COURSE and CLASS entities

COURSE(**CRS_CODE**, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS(**CRS_CODE**, **CLASS_SECTION** , CLASS_TIME, ROOM_CODE, PROF_NUM)

indicate that a strong relationship exists between COURSE and CLASS, because the CLASS entity's composite PK is composed of CRS_CODE + CLASS_SECTION. (Note that the CRS_CODE in CLASS is *also* the FK to the COURSE entity.)

The Crow's Foot notation depicts the strong (identifying) relationship with a solid line between the entities, shown in Figure 4.9. Whether the relationship between COURSE and CLASS is strong or weak depends on how the CLASS entity's primary key is defined.

Keep in mind that *the order in which the tables are created and loaded is very important*. For example, in the "COURSE generates CLASS" relationship, the COURSE table must be created before the CLASS table. After all, it would not be acceptable to have the CLASS table's foreign key reference a COURSE table that did not yet exist. In fact, *you must load the data of the "1" side first in a 1:M relationship to avoid the possibility of referential integrity errors*, regardless of whether the relationships are weak or strong.

As you examine Figure 4.9 you might wonder what the O symbol next to the CLASS entity signifies. You will discover the meaning of this cardinality in Section 4.1.8, Relationship Participation.

Remember that the nature of the relationship is often determined by the database designer, who must use professional judgment to determine which relationship type and strength best suit the database transaction, efficiency, and information requirements. That point will often be emphasized in detail!

4.1.7 WEAK ENTITIES

In contrast to the strong or regular entity mentioned in Section 4.1.5, a **weak entity** is one that meets two conditions:

1. The entity is existence-dependent; that is, it cannot exist without the entity with which it has a relationship.
2. The entity has a primary key that is partially or totally derived from the parent entity in the relationship.

FIGURE 4.9 **A strong (identifying) relationship between COURSE and CLASS**

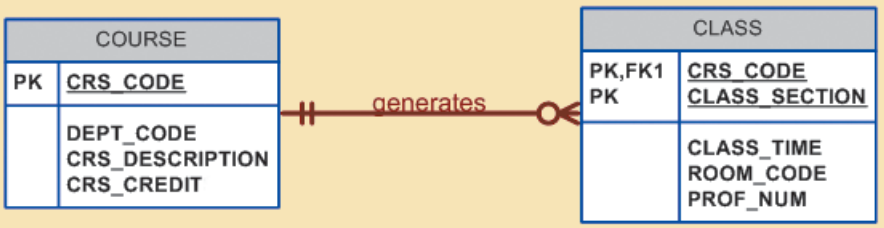


Table name: COURSE **Database name: Ch04_TinyCollege_Alt**

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
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS

CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325
QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

For example, a company insurance policy insures an employee and his/her dependents. For the purpose of describing an insurance policy, an EMPLOYEE might or might not have a DEPENDENT, but the DEPENDENT must be associated with an EMPLOYEE. Moreover, the DEPENDENT cannot exist without the EMPLOYEE; that is, a person cannot get insurance coverage as a dependent unless s(he) happens to be a dependent of an employee. DEPENDENT is the weak entity in the relationship "EMPLOYEE has DEPENDENT." This relationship is shown in Figure 4.10.

Note that the Chen notation in Figure 4.10 identifies the weak entity by using a double-walled entity rectangle. The Crow's Foot notation generated by Visio Professional uses the relationship line and the PK/FK designation to indicate whether the related entity is weak. A strong (identifying) relationship indicates that the related entity is weak. Such a relationship means that both conditions for the weak entity definition have been met—the related entity is existence-dependent, and the PK of the related entity contains a PK component of the parent entity. (Some versions of the Crow's Foot ERD depict the weak entity by drawing a short line segment in each of the four corners of the weak entity box.)

Remember that the weak entity inherits part of its primary key from its strong counterpart. For example, at least part of the DEPENDENT entity's key shown in Figure 4.10 was inherited from the EMPLOYEE entity:

EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_DOB, EMP_HIREDATE)

DEPENDENT (**EMP_NUM**, **DEP_NUM**, DEP_FNAME, DEP_DOB)

FIGURE 4.10 A weak entity in an ERD

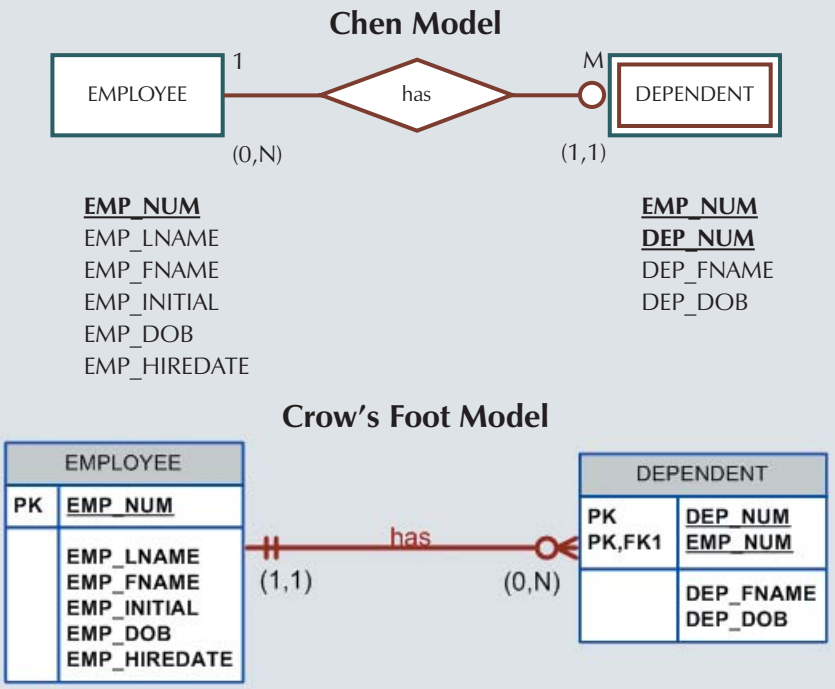


Figure 4.11 illustrates the implementation of the relationship between the weak entity (DEPENDENT) and its parent or strong counterpart (EMPLOYEE). Note that DEPENDENT's primary key is composed of two attributes, EMP_NUM and DEP_NUM, and that EMP_NUM was inherited from EMPLOYEE.

FIGURE 4.11 A weak entity in a strong relationship

Table name: EMPLOYEE **Database name: Ch04_ShortCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIREDATE
1001	Callifante	Jeanine	J	12-Mar-64	25-May-97
1002	Smithson	William	K	23-Nov-70	28-May-97
1003	Washington	Herman	H	15-Aug-68	28-May-97
1004	Chen	Lydia	B	23-Mar-74	15-Oct-98
1005	Johnson	Melanie		28-Sep-66	20-Dec-98
1006	Ortega	Jorge	G	12-Jul-79	05-Jan-02
1007	O'Donnell	Peter	D	10-Jun-71	23-Jun-02
1008	Brzenski	Barbara	A	12-Feb-70	01-Nov-03

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

Given this scenario, and with the help of this relationship, you can determine that:

Jeanine J. Callifante claims two dependents, Annelise and Jorge.

Keep in mind that the database designer usually determines whether an entity can be described as weak based on the business rules. An examination of the relationship between COURSE and CLASS in Figure 4.8 might cause you to conclude that CLASS is a weak entity to COURSE. After all, in Figure 4.8, it seems clear that a CLASS cannot exist without a COURSE; so there is existence dependence. For example, a student cannot enroll in the Accounting I class ACCT-211, Section 3 (CLASS_CODE 10014) unless there is an ACCT-211 course. However, note that the CLASS table's primary key is CLASS_CODE, which is not derived from the COURSE parent entity. That is, CLASS may be represented by:

CLASS (**CLASS_CODE**, CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM)

The second weak entity requirement has not been met; therefore, by definition, the CLASS entity in Figure 4.8 may not be classified as weak. On the other hand, if the CLASS entity's primary key had been defined as a composite key, composed of the combination CRS_CODE and CLASS_SECTION, CLASS could be represented by:

CLASS (**CRS_CODE**, **CLASS_SECTION**, CLASS_TIME, ROOM_CODE, PROF_NUM)

In that case, illustrated in Figure 4.9, the CLASS primary key is partially derived from COURSE because CRS_CODE is the COURSE table's primary key. Given this decision, CLASS is a weak entity by definition. (In Visio Professional Crow's Foot terms, the relationship between COURSE and CLASS is classified as strong, or identifying.) In any case, CLASS is always existence-dependent on COURSE, *whether or not it is defined as weak*.

4.1.8 Relationship Participation

Participation in an entity relationship is either optional or mandatory. Recall that relationships are bidirectional; that is, they operate in both directions. If COURSE is related to CLASS, then by definition, CLASS is related to COURSE. Because of the bidirectional nature of relationships, it is necessary to determine the connectivity of the relationship from COURSE to CLASS and the connectivity of the relationship from CLASS to COURSE. Similarly, the specific maximum and minimum cardinalities must be determined in each direction for the relationship. Once again, you must consider the bidirectional nature of the relationship when determining participation.

Optional participation means that one entity occurrence does not *require* a corresponding entity occurrence in a particular relationship. For example, in the "COURSE generates CLASS" relationship, you noted that at least some courses do not generate a class. In other words, an entity occurrence (row) in the COURSE table does not necessarily require the existence of a corresponding entity occurrence in the CLASS table. (Remember that each entity is implemented as a table.) Therefore, the CLASS entity is considered to be *optional* to the COURSE entity. In Crow's Foot notation, an optional relationship between entities is shown by drawing a small circle (O) on the side of the optional entity, as illustrated in Figure 4.9. The existence of an *optional entity* indicates that the minimum cardinality is 0 for the optional entity. (The term *optionality* is used to label any condition in which one or more optional relationships exist.)

NOTE

Remember that the burden of establishing the relationship is always placed on the entity that contains the foreign key. In most cases, that will be the entity on the "many" side of the relationship.

Mandatory participation means that one entity occurrence *requires* a corresponding entity occurrence in a particular relationship. If no optionality symbol is depicted with the entity, the entity is assumed to exist in a mandatory relationship with the related entity. If the mandatory participation is depicted graphically, it is typically shown as a small

hash mark across the relationship line, similar to the Crow's Foot depiction of a connectivity of 1. The existence of a mandatory relationship indicates that the minimum cardinality is at least 1 for the mandatory entity.

NOTE

You might be tempted to conclude that relationships are weak when they occur between entities in an optional relationship and that relationships are strong when they occur between entities in a mandatory relationship. However, this conclusion is not warranted. Keep in mind that relationship participation and relationship strength do not describe the same thing. You are likely to encounter a strong relationship when one entity is optional to another. For example, the relationship between EMPLOYEE and DEPENDENT is clearly a strong one, but DEPENDENT is clearly optional to EMPLOYEE. After all, you cannot *require* employees to have dependents. And it is just as possible for a weak relationship to be established when one entity is mandatory to another. *The relationship strength depends on how the PK of the related entity is formulated, while the relationship participation depends on how the business rule is written*. For example, the business rules "Each part must be supplied by a vendor" and "A part may or may not be supplied by a vendor" create different optionalities for the same entities! Failure to understand this distinction may lead to poor design decisions that cause major problems when table rows are inserted or deleted.

When you create a relationship in MS Visio, the default relationship will be mandatory on the "1" side and optional on the "many" side. Table 4.3 shows the various connectivity and participation combinations that are supported by the Crow's Foot notation. Recall that these combinations are often referred to as cardinality in Crow's Foot notation when specific cardinalities are not used.

TABLE 4.3 Crow's Foot Symbols

CROW'S FOOT SYMBOL	CARDINALITY	COMMENT
	(0,N)	Zero or many. Many side is optional.
	(1,N)	One or many. Many side is mandatory.
	(1,1)	One and only one. 1 side is mandatory.
	(0,1)	Zero or one. 1 side is optional.

Because relationship participation turns out to be a very important component of the database design process, let's examine a few more scenarios. Suppose that Tiny College employs some professors who conduct research without teaching classes. If you examine the "PROFESSOR teaches CLASS" relationship, it is quite possible for a PROFESSOR not to teach a CLASS. Therefore, CLASS is *optional* to PROFESSOR. On the other hand, a CLASS must be taught by a PROFESSOR. Therefore, PROFESSOR is *mandatory* to CLASS. Note that the ERD model in Figure 4.12 shows the cardinality next to CLASS to be (0,3), thus indicating that a professor may teach no classes at all or as many as three classes. And each CLASS table row will reference one and only one PROFESSOR row—assuming each class is taught by one and only one professor—represented by the (1,1) cardinality next to the PROFESSOR table.

Failure to understand the distinction between *mandatory* and *optional* participation in relationships might yield designs in which awkward (and unnecessary) temporary rows (entity instances) must be created just to accommodate the creation of required entities. Therefore, it is important that you clearly understand the concepts of mandatory and optional participation.

It is also important to understand that the semantics of a problem might determine the type of participation in a relationship. For example, suppose that Tiny College offers several courses; each course has several classes. Note

FIGURE 4.12 An optional CLASS entity in the relationship "PROFESSOR teaches CLASS"

again the distinction between *class* and *course* in this discussion: a CLASS constitutes a specific offering (or section) of a COURSE. (Typically, courses are listed in the university's course catalog, while classes are listed in the class schedules that students use to register for their classes.)

Analyzing the CLASS entity's contribution to the "COURSE generates CLASS" relationship, it is easy to see that a CLASS cannot exist without a COURSE. Therefore, you can conclude that the COURSE entity is *mandatory* in the relationship. But two scenarios for the CLASS entity may be written, shown in Figures 4.13 and 4.14.

FIGURE 4.13 CLASS is optional to COURSE

FIGURE 4.14 COURSE and CLASS in a mandatory relationship

The different scenarios are a function of the semantics of the problem; that is, they depend on how the relationship is defined.

1. *CLASS is optional*. It is possible for the department to create the entity COURSE first and then create the CLASS entity after making the teaching assignments. In the real world, such a scenario is very likely; there may be courses for which sections (classes) have not yet been defined. In fact, some courses are taught only once a year and do not generate classes each semester.
2. *CLASS is mandatory*. This condition is created by the constraint that is imposed by the semantics of the statement "Each COURSE generates one or more CLASSes." In ER terms, each COURSE in the "generates" relationship must have at least one CLASS. Therefore, a CLASS must be created as the COURSE is created, in order to comply with the semantics of the problem.

Keep in mind the practical aspects of the scenario presented in Figure 4.14. Given the semantics of this relationship, the system should not accept a course that is not associated with at least one class section. Is such a rigid environment

desirable from an operational point of view? For example, when a new COURSE is created, the database first updates the COURSE table, thereby inserting a COURSE entity that does not yet have a CLASS associated with it. Naturally, the apparent problem seems to be solved when CLASS entities are inserted into the corresponding CLASS table. However, because of the mandatory relationship, the system will be in temporary violation of the business rule constraint. For practical purposes, it would be desirable to classify the CLASS as optional in order to produce a more flexible design.

Finally, as you examine the scenarios presented in Figures 4.13 and 4.14, keep in mind the role of the DBMS. To maintain data integrity, the DBMS must ensure that the "many" side (CLASS) is associated with a COURSE through the foreign key rules.

4.1.9 Relationship Degree

A **relationship degree** indicates the number of entities or participants associated with a relationship. A **unary relationship** exists when an association is maintained within a single entity. A **binary relationship** exists when two entities are associated. A **ternary relationship** exists when three entities are associated. Although higher degrees exist, they are rare and are not specifically named. (For example, an association of four entities is described simply as a *four-degree relationship*.) Figure 4.15 shows these types of relationship degrees.

Unary Relationships

In the case of the unary relationship shown in Figure 4.15, an employee within the EMPLOYEE entity is the manager for one or more employees within that entity. In this case, the existence of the "manages" relationship means that EMPLOYEE requires another EMPLOYEE to be the manager—that is, EMPLOYEE has a relationship with itself. Such a relationship is known as a **recursive relationship**. The various cases of recursive relationships will be explored in Section 4.1.10.

Binary Relationships

A binary relationship exists when two entities are associated in a relationship. Binary relationships are most common. In fact, to simplify the conceptual design, whenever possible, most higher-order (ternary and higher) relationships are decomposed into appropriate equivalent binary relationships. In Figure 4.15, the relationship "a PROFESSOR teaches one or more CLASSes" represents a binary relationship.

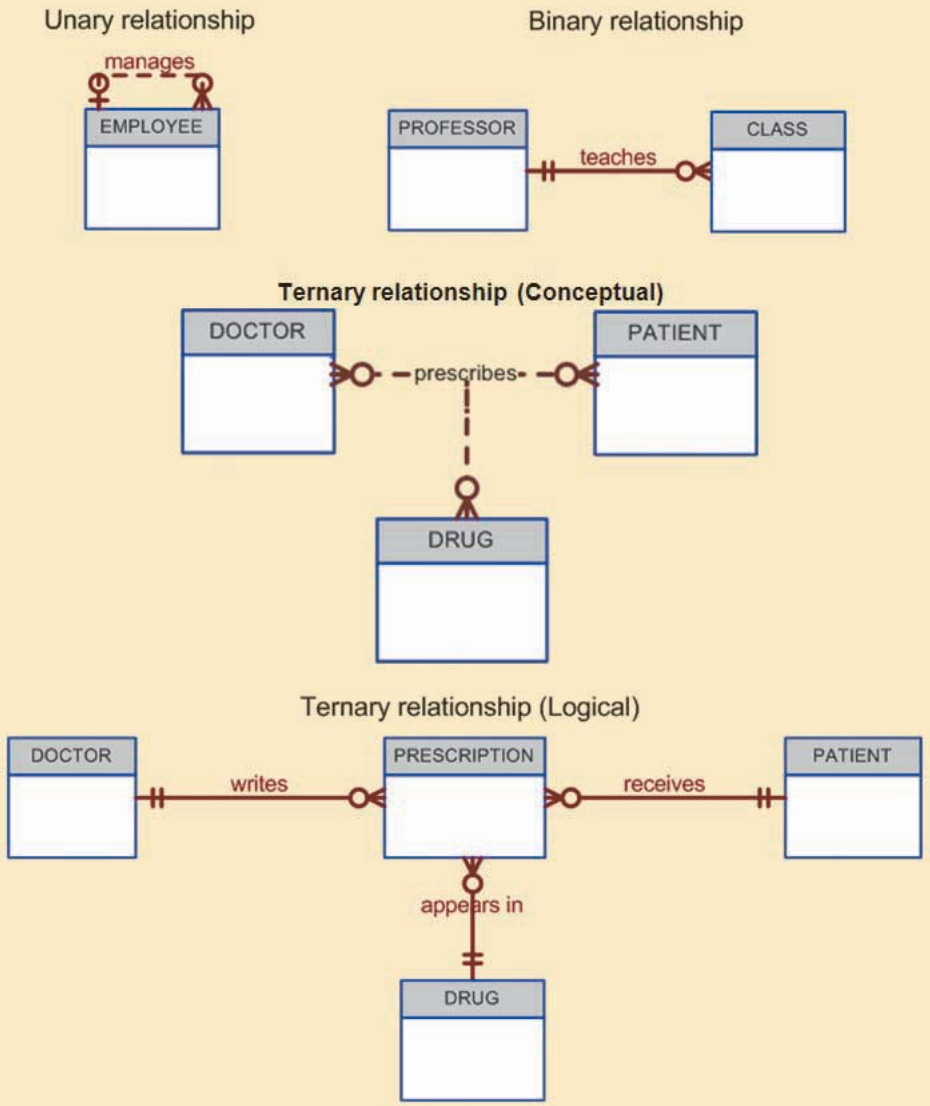
Ternary and Higher-Degree Relationships

Although most relationships are binary, the use of ternary and higher-order relationships does allow the designer some latitude regarding the semantics of a problem. A ternary relationship implies an association among three different entities. For example, note the relationships (and their consequences) in Figure 4.16, which are represented by the following business rules:

- A DOCTOR writes one or more PRESCRIPTIONs.
- A PATIENT may receive one or more PRESCRIPTIONs.
- A DRUG may appear in one or more PRESCRIPTIONs. (To simplify this example, assume that the business rule states that each prescription contains only one drug. In short, if a doctor prescribes more than one drug, a separate prescription must be written for each drug.)

As you examine the table contents in Figure 4.16, note that it is possible to track all transactions. For instance, you can tell that the first prescription was written by doctor 32445 for patient 102, using the drug DRZ.

FIGURE 4.15 Three types of relationship degree



4.1.10 Recursive Relationships

As was previously mentioned, a *recursive relationship* is one in which a relationship can exist between occurrences of the same entity set. (Naturally, such a condition is found within a unary relationship.) For example, a 1:M unary relationship can be expressed by "an EMPLOYEE may manage many EMPLOYEEs, and each EMPLOYEE is managed by one EMPLOYEE." And as long as polygamy is not legal, a 1:1 unary relationship may be expressed by "an EMPLOYEE may be married to one and only one other EMPLOYEE." Finally, the M:N unary relationship may be expressed by "a COURSE may be a prerequisite to many other COURSEs, and each COURSE may have many other COURSEs as prerequisites." Those relationships are shown in Figure 4.17.

The 1:1 relationship shown in Figure 4.17 can be implemented in the single table shown in Figure 4.18. Note that you can determine that James Ramirez is married to Louise Ramirez, who is married to James Ramirez. And Anne Jones is married to Anton Shapiro, who is married to Anne Jones.

FIGURE 4.16 The implementation of a ternary relationship

Database name: Ch04_Clinic

Table name: DRUG

DRUG_CODE	DRUG_NAME	DRUG_PRICE
AF15	Afgapan-15	25.00
AF25	Afgapan-25	35.00
DRO	Droalene Chloride	111.89
DRZ	Druzocholar Cryptolene	18.99
KO15	Koliabar Oxyhexalene	65.75
OLE	Oleander-Drizapan	123.95
TRYP	Tryptolac Heptadimetric	79.45

Table name: PATIENT

PAT_NUM	PAT_TITLE	PAT_LNAME	PAT_FNAME	PAT_INITIAL	PAT_DOB	PAT_AREACODE	PAT_PHONE
100	Mr.	Kolmycz	George	D	15-Jun-1942	615	324-5456
101	Ms.	Lewis	Rhonda	G	19-Mar-2005	615	324-4472
102	Mr.	Vandam	Rhett		14-Nov-1958	901	675-8993
103	Ms.	Jones	Anne	M	16-Oct-1974	615	898-3456
104	Mr.	Lange	John	P	08-Nov-1971	901	504-4430
105	Mr.	Williams	Robert	D	14-Mar-1975	615	890-3220
106	Mrs.	Smith	Jeanine	K	12-Feb-2003	615	324-7883
107	Mr.	Diante	Jorge	D	21-Aug-1974	615	890-4567
108	Mr.	Wiesenbach	Paul	R	14-Feb-1966	615	897-4358
109	Mr.	Smith	George	K	18-Jun-1961	901	504-3339
110	Mrs.	Genkazi	Leighla	W	19-May-1970	901	569-0093
111	Mr.	Washington	Rupert	E	03-Jan-1966	615	890-4925
112	Mr.	Johnson	Edward	E	14-May-1961	615	898-4387
113	Ms.	Smythe	Melanie	P	15-Sep-1970	615	324-9006
114	Ms.	Brandon	Marie	G	02-Nov-1932	901	882-0845
115	Mrs.	Saranda	Hermine	R	25-Jul-1972	615	324-5505
116	Mr.	Smith	George	A	08-Nov-1965	615	890-2984

Table name: DOCTOR

DOC_ID	DOC_LNAME	DOC_FNAME	DOC_INITIAL	DOC_SPECIALTY
29827	Sanchez	Julio	J	Dermatology
32445	Jorgensen	Annelise	G	Neurology
33456	Korenski	Anatoly	A	Urology
33989	LeGrande	George		Pediatrics
34409	Washington	Dennis	F	Orthopaedics
36221	McPherson	Katye	H	Dermatology
36712	Dreifag	Herman	G	Psychiatry
38995	Minh	Tran		Neurology
40004	Chin	Ming	D	Orthopaedics
40028	Feinstein	Denise	L	Gynecology

Table name: PRESCRIPTION

DOC_ID	PAT_NUM	DRUG_CODE	PRES_DOSAGE	PRES_DATE
32445	102	DRZ	2 tablets every four hours -- 50 tablets total	12-Nov-09
32445	113	OLE	1 teaspoon with each meal -- 250 ml total	14-Nov-09
34409	101	KO15	1 tablet every six hours -- 30 tablets total	14-Nov-09
36221	109	DRO	2 tablets with every meal -- 60 tablets total	14-Nov-09
38995	107	KO15	1 tablet every six hours -- 30 tablets total	14-Nov-09

FIGURE 4.17 An ER representation of recursive relationships

FIGURE 4.18 The 1:1 recursive relationship "EMPLOYEE is married to EMPLOYEE"

Database name: CH04_PartCo

Table name: EMPLOYEE_V1

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
345	Ramirez	James	347
346	Jones	Anne	349
347	Ramirez	Louise	345
348	Delaney	Robert	
349	Shapiro	Anton	346

Unary relationships are common in manufacturing industries. For example, Figure 4.19 illustrates that a rotor assembly (C-130) is composed of many parts, but each part is used to create only one rotor assembly. Figure 4.19 indicates that a rotor assembly is composed of four 2.5-cm washers, two cotter pins, one 2.5-cm steel shank, four 10.25-cm rotor blades, and two 2.5-cm hex nuts. The relationship implemented in Figure 4.19 thus enables you to track each part within each rotor assembly.

If a part can be used to assemble several different kinds of other parts and is itself composed of many parts, two tables

FIGURE 4.19 **Another unary relationship: "PART contains PART"**

Table name: PART_V1 **Database name; CH04_PartCo**

PART_CODE	PART_DESCRIPTION	PART_IN_STOCK	PART_UNITS_NEEDED	PART_OF_PART
AA21-6	2.5 cm. washer, 1.0 mm. rim	432	4	C-130
AB-121	Cotter pin, copper	1034	2	C-130
C-130	Rotor assembly	36		
E129	2.5 cm. steel shank	128	1	C-130
X10	10.25 cm. rotor blade	345	4	C-130
X34AW	2.5 cm. hex nut	879	2	C-130

are required to implement the "PART contains PART" relationship. Figure 4.20 illustrates such an environment. Parts tracking is increasingly important as managers become more aware of the legal ramifications of producing more complex output. In fact, in many industries, especially those involving aviation, full parts tracking is required by law.

FIGURE 4.20 **Implementation of the M:N recursive relationship "PART contains PART"**

Table name: COMPONENT **Database name: Ch04_PartCo**

COMP_CODE	PART_CODE	COMP_PARTS_NEEDED
C-130	AA21-6	4
C-130	AB-121	2
C-130	E129	1
C-131A2	E129	1
C-130	X10	4
C-131A2	X10	1
C-130	X34AW	2
C-131A2	X34AW	2

Table name: PART

PART_CODE	PART_DESCRIPTION	PART_IN_STOCK
AA21-6	2.5 cm. washer, 1.0 mm. rim	432
AB-121	Cotter pin, copper	1034
C-130	Rotor assembly	36
E129	2.5 cm. steel shank	128
X10	10.25 cm. rotor blade	345
X34AW	2.5 cm. hex nut	879

The M:N recursive relationship might be more familiar in a school environment. For instance, note how the M:N "COURSE requires COURSE" relationship illustrated in Figure 4.17 is implemented in Figure 4.21. In this example, MATH-243 is a prerequisite to QM-261 and QM-362, while both MATH-243 and QM-261 are prerequisites to QM-362.

Finally, the 1:M recursive relationship "EMPLOYEE manages EMPLOYEE," shown in Figure 4.17, is implemented in Figure 4.22.

One common pitfall when working with unary relationships is to confuse participation with referential integrity. In theory, participation and referential integrity are very different concepts and are normally easy to distinguish in binary relationships. In practical terms, conversely, participation and referential integrity are very similar because they are both implemented through constraints on the same set of attributes. This similarity often leads to confusion when the concepts are applied within the limited structure of a unary relationship. Consider the unary 1:1 relationship described in Figure 4.18 of a spousal relationship between employees. Participation, as described above, is bidirectional,

FIGURE 4.21 Implementation of the M:N recursive relationship "COURSE requires COURSE"

Table name: COURSE **Database name: Ch04_TinyCollege**

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
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: PREREQ

CRS_CODE	PRE_TAKE
CIS-420	CIS-220
QM-261	MATH-243
QM-362	MATH-243
QM-362	QM-261

FIGURE 4.22 Implementation of the 1:M recursive relationship "EMPLOYEE manages EMPLOYEE"

Database name: Ch04_PartCo
Table name: EMPLOYEE_V2

EMP_CODE	EMP_LNAME	EMP_MANAGER
101	vVaddell	102
102	Orincona	
103	Jones	102
104	Reballoh	102
105	Robertson	102
106	Deltona	102

meaning that it must be addressed in both directions along the relationship. Participation in Figure 4.18 addresses the questions:

- Must every employee have a spouse who is an employee?
- Must every employee be a spouse to another employee?

For the data shown in Figure 4.18, the correct answer to both of those questions is "No." It is possible to be an employee and not have another employee as a spouse. Also, it is possible to be an employee and not be the spouse of another employee.

Referential integrity deals with the correspondence of values in the foreign key with values in the related primary key. Referential integrity is not bidirectional, and therefore has only one question that it answers.

- Must every employee spouse be a valid employee?

For the data shown in Figure 4.18, the correct answer is "Yes." Another way to frame this question is to consider whether or not every value provided for the EMP_SPOUSE attribute must match some value in the EMP_NUM attribute.

In practical terms, both participation and referential integrity involve the values used as primary key/foreign key to implement the relationship. Referential integrity requires that the values in the foreign key correspond to values in the primary key. In one direction, participation considers whether or not the foreign key can contain a null. In Figure 4.18,

for example, employee Robert Delaney is not required to have a value in EMP_SPOUSE. In the other direction, participation considers whether or not every value in the primary key must appear as a value in the foreign key. In Figure 4.18, for example, employee Robert Delaney's value for EMP_NUM (348) is not required to appear as a value in EMP_SPOUSE for any other employee.

4.1.11 Associative (Composite) Entities

In the original ERM described by Chen, relationships do not contain attributes. You should recall from Chapter 3 that the relational model generally requires the use of 1:M relationships. (Also, recall that the 1:1 relationship has its place, but it should be used with caution and proper justification.) If M:N relationships are encountered, you must create a bridge between the entities that display such relationships. The associative entity is used to implement a M:N relationship between two or more entities. This associative entity (also known as a *composite* or *bridge entity*) is composed of the primary keys of each of the entities to be connected. An example of such a bridge is shown in Figure 4.23. The Crow's Foot notation does not identify the composite entity as such. Instead, the composite entity is identified by the solid relationship line between the parent and child entities, thereby indicating the presence of a strong (identifying) relationship.

FIGURE 4.23 Converting the M:N relationship into two 1:M relationships

Table name: STUDENT **Database name: Ch04_CollegeTry**

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL

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

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

Note that the composite ENROLL entity in Figure 4.23 is existence-dependent on the other two entities; the composition of the ENROLL entity is based on the primary keys of the entities that are connected by the composite entity. The composite entity may also contain additional attributes that play no role in the connective process. For example, although the entity must be composed of at least the STUDENT and CLASS primary keys, it may also include such additional attributes as grades, absences, and other data uniquely identified by the student's performance in a specific class.

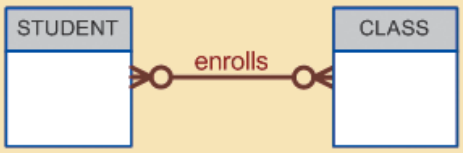
Finally, keep in mind that the ENROLL table's key (CLASS_CODE and STU_NUM) is composed entirely of the primary keys of the CLASS and STUDENT tables. Therefore, no null entries are possible in the ENROLL table's key attributes.

Implementing the small database shown in Figure 4.23 requires that you define the relationships clearly. Specifically, you must know the "1" and the "M" sides of each relationship, and you must know whether the relationships are mandatory or optional. For example, note the following points:

- A class may exist (at least at the start of registration) even though it contains no students. Therefore, if you examine Figure 4.24, an optional symbol should appear on the STUDENT side of the M:N relationship between STUDENT and CLASS.

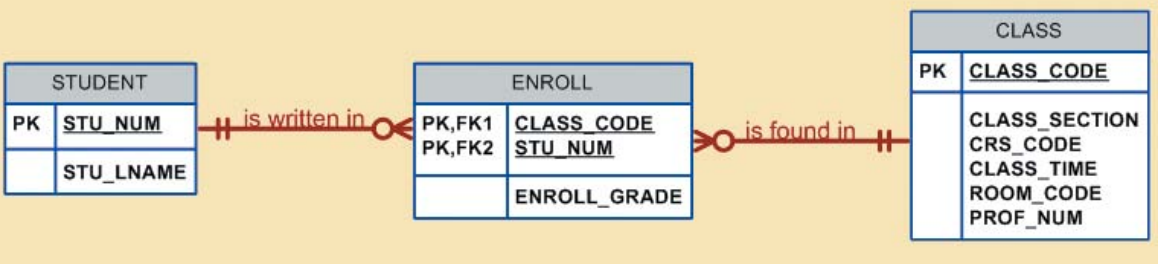
  You might argue that to be classified as a STUDENT, a person must be enrolled in at least one CLASS. Therefore, CLASS is mandatory to STUDENT from a purely conceptual point of view. However, when a student is admitted to college, that student has not (yet) signed up for any classes. Therefore, *at least initially*, CLASS is optional to STUDENT. Note that the practical considerations in the data environment help dictate the use of optionalities. If CLASS is *not* optional to STUDENT—from a database point of view—a class assignment must be made when the student is admitted. But that's *not* how the process actually works, and the database design must reflect this. In short, the optionality reflects practice.

FIGURE 4.24 The M:N relationship between STUDENT and CLASS



Because the M:N relationship between STUDENT and CLASS is decomposed into two 1:M relationships through ENROLL, the optionalities must be transferred to ENROLL. (See Figure 4.25.) In other words, it now becomes possible for a class not to occur in ENROLL if no student has signed up for that class. Because a class need not occur in ENROLL, the ENROLL entity becomes optional to CLASS. And because the ENROLL entity is created before any students have signed up for a class, the ENROLL entity is also optional to STUDENT, at least initially.

FIGURE 4.25 A composite entity in an ERD



- As students begin to sign up for their classes, they will be entered into the ENROLL entity. Naturally, if a student takes more than one class, that student will occur more than once in ENROLL. For example, note that in the ENROLL table in Figure 4.23, STU_NUM = 321452 occurs three times. On the other hand, each student occurs only once in the STUDENT entity. (Note that the STUDENT table in Figure 4.23 has only one STU_NUM = 321452 entry.) Therefore, in Figure 4.25, the relationship between STUDENT and ENROLL is shown to be 1:M, with the M on the ENROLL side.

- As you can see in Figure 4.23, a class can occur more than once in the ENROLL table. For example, CLASS_CODE = 10014 occurs twice. However, CLASS_CODE = 10014 occurs only once in the CLASS table to reflect that the relationship between CLASS and ENROLL is 1:M. Note that in Figure 4.25, the M is located on the ENROLL side, while the 1 is located on the CLASS side.

4.2 DEVELOPING AN ER DIAGRAM

The process of database design is an iterative rather than a linear or sequential process. The verb *iterate* means "to do again or repeatedly." An **iterative process** is, thus, one based on repetition of processes and procedures. Building an ERD usually involves the following activities:

- Create a detailed narrative of the organization's description of operations.
- Identify the business rules based on the description of operations.
- Identify the main entities and relationships from the business rules.
- Develop the initial ERD.
- Identify the attributes and primary keys that adequately describe the entities.
- Revise and review the ERD.

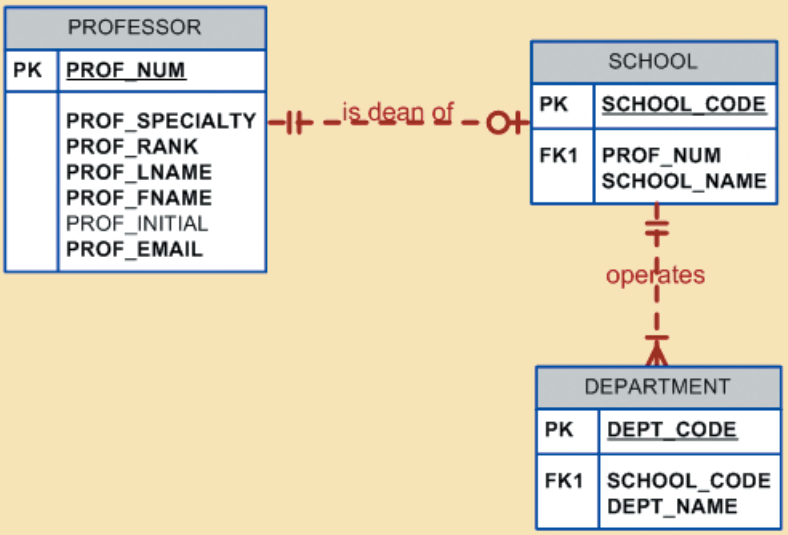
During the review process, it is likely that additional objects, attributes, and relationships will be uncovered. Therefore, the basic ERM will be modified to incorporate the newly discovered ER components. Subsequently, another round of reviews might yield additional components or clarification of the existing diagram. The process is repeated until the end users and designers agree that the ERD is a fair representation of the organization's activities and functions.

During the design process, the database designer does not depend simply on interviews to help define entities, attributes, and relationships. A surprising amount of information can be gathered by examining the business forms and reports that an organization uses in its daily operations.

To illustrate the use of the iterative process that ultimately yields a workable ERD, let's start with an initial interview with the Tiny College administrators. The interview process yields the following business rules:

1. Tiny College (TC) is divided into several schools: a school of business, a school of arts and sciences, a school of education, and a school of applied sciences. Each school is administered by a dean who is a professor. Each professor can be the dean of only one school, and a professor is not required to be the dean of any school. Therefore, a 1:1 relationship exists between PROFESSOR and SCHOOL. Note that the cardinality can be expressed by writing (1,1) next to the entity PROFESSOR and (0,1) next to the entity SCHOOL.
2. Each school comprises several departments. For example, the school of business has an accounting department, a management/marketing department, an economics/finance department, and a computer information systems department. Note again the cardinality rules: The smallest number of departments operated by a school is one, and the largest number of departments is indeterminate (N). On the other hand, each department belongs to only a single school; thus, the cardinality is expressed by (1,1). That is, the minimum number of schools that a department belongs to is one, as is the maximum number. Figure 4.26 illustrates these first two business rules.

FIGURE 4.26 The first Tiny College ERD segment



NOTE

It is again appropriate to evaluate the reason for maintaining the 1:1 relationship between PROFESSOR and SCHOOL in the PROFESSOR is dean of SCHOOL relationship. It is worth repeating that the existence of 1:1 relationships often indicates a misidentification of attributes as entities. In this case, the 1:1 relationship could easily be eliminated by storing the dean's attributes in the SCHOOL entity. This solution would also make it easier to answer the queries, "Who is the dean?" and "What are that dean's credentials?" The downside of this solution is that it requires the duplication of data that are already stored in the PROFESSOR table, thus setting the stage for anomalies. However, because each school is run by a single dean, the problem of data duplication is rather minor. The selection of one approach over another often depends on information requirements, transaction speed, and the database designer's professional judgment. In short, do not use 1:1 relationships lightly, and make sure that each 1:1 relationship within the database design is defensible.

3. Each department may offer courses. For example, the management/marketing department offers courses such as Introduction to Management, Principles of Marketing, and Production Management. The ERD segment for this condition is shown in Figure 4.27. Note that this relationship is based on the way Tiny College operates. If, for example, Tiny College had some departments that were classified as "research only," those departments would not offer courses; therefore, the COURSE entity would be optional to the DEPARTMENT entity.
4. The relationship between COURSE and CLASS was illustrated in Figure 4.9. Nevertheless, it is worth repeating that a CLASS is a section of a COURSE. That is, a department may offer several sections (classes) of the same database course. Each of those classes is taught by a professor at a given time in a given place. In short, a 1:M relationship exists between COURSE and CLASS. However, because a course may exist in Tiny College's course catalog even when it is not offered as a class in a current class schedule, CLASS is optional to COURSE. Therefore, the relationship between COURSE and CLASS looks like Figure 4.28.

FIGURE 4.27 The second Tiny College ERD segment

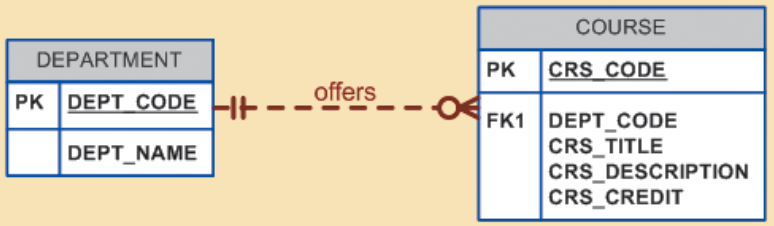
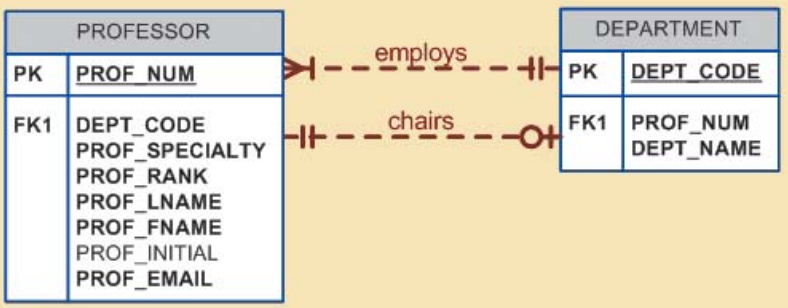


FIGURE 4.28 The third Tiny College ERD segment

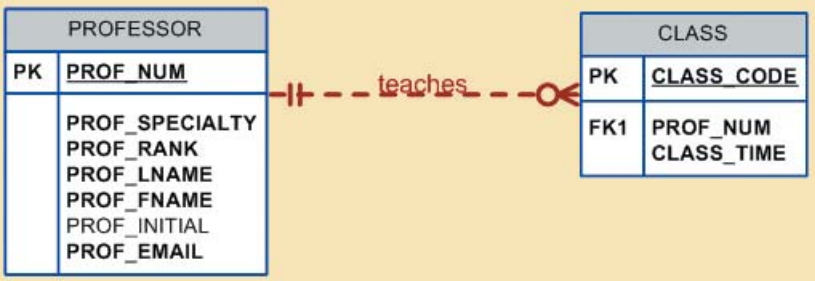
5. Each department should have one or more professors assigned to it. One and only one of those professors chairs the department, and no professor is required to accept the chair position. Therefore, DEPARTMENT is optional to PROFESSOR in the "chairs" relationship. Those relationships are summarized in the ER segment shown in Figure 4.29.

FIGURE 4.29 The fourth Tiny College ERD segment



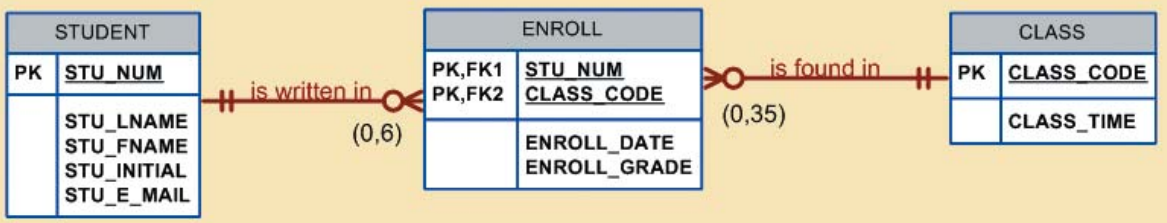
6. Each professor may teach up to four classes; each class is a section of a course. A professor may also be on a research contract and teach no classes at all. The ERD segment in Figure 4.30 depicts those conditions.
7. A student may enroll in several classes but takes each class only once during any given enrollment period. For example, during the current enrollment period, a student may decide to take five classes—Statistics, Accounting, English, Database, and History—but that student would not be enrolled in the same Statistics class five times during the enrollment period! Each student may enroll in up to six classes, and each class may have up to 35 students, thus creating an M:N relationship between STUDENT and CLASS. Because a CLASS can

FIGURE 4.30 The fifth Tiny College ERD segment



initially exist (at the start of the enrollment period) even though no students have enrolled in it, STUDENT is optional to CLASS in the M:N relationship. This M:N relationship must be divided into two 1:M relationships through the use of the ENROLL entity, shown in the ERD segment in Figure 4.31. But note that the optional symbol is shown next to ENROLL. If a class exists but has no students enrolled in it, that class doesn't occur in the ENROLL table. Note also that the ENROLL entity is weak: it is existence-dependent, and its (composite) PK is composed of the PKs of the STUDENT and CLASS entities. You can add the cardinalities (0,6) and (0,35) next to the ENROLL entity to reflect the business rule constraints, as shown in Figure 4.31. (Visio Professional does not automatically generate such cardinalities, but you can use a text box to accomplish that task.)

FIGURE 4.31 The sixth Tiny College ERD segment



8. Each department has several (or many) students whose major is offered by that department. However, each student has only a single major and is, therefore, associated with a single department. (See Figure 4.32.) However, in the Tiny College environment, it is possible—at least for a while—for a student not to declare a major field of study. Such a student would not be associated with a department; therefore, DEPARTMENT is optional to STUDENT. It is worth repeating that the relationships between entities and the entities themselves reflect the organization's operating environment. That is, the business rules define the ERD components.
9. Each student has an advisor in his or her department; each advisor counsels several students. An advisor is also a professor, but not all professors advise students. Therefore, STUDENT is optional to PROFESSOR in the "PROFESSOR advises STUDENT" relationship. (See Figure 4.33.)
10. As you can see in Figure 4.34, the CLASS entity contains a ROOM_CODE attribute. Given the naming conventions, it is clear that ROOM_CODE is an FK to another entity. Clearly, because a class is taught in a room, it is reasonable to assume that the ROOM_CODE in CLASS is the FK to an entity named ROOM. In turn, each room is located in a building. So the last Tiny College ERD is created by observing that a BUILDING

FIGURE 4.32 The seventh Tiny College ERD segment

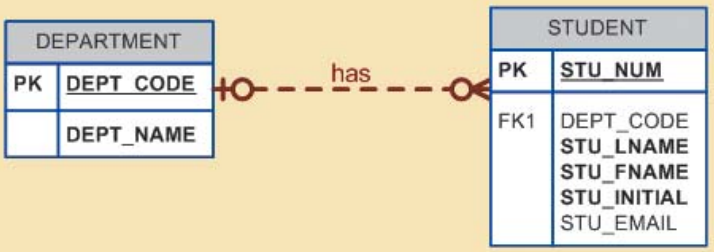
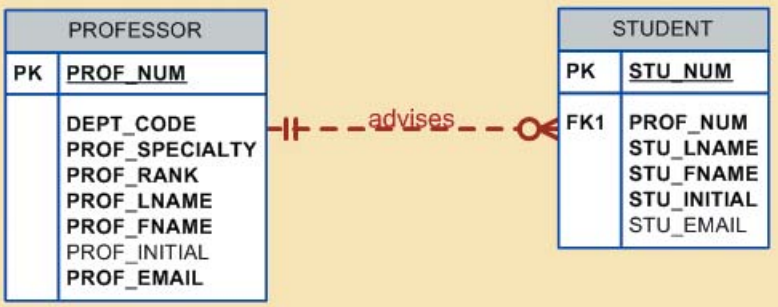
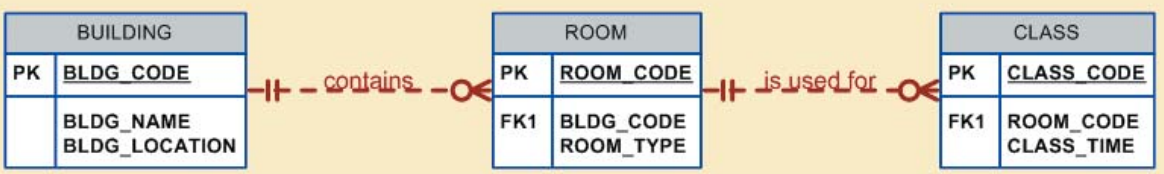


FIGURE 4.33 The eighth Tiny College ERD segment



can contain many ROOMs, but each ROOM is found in a single BUILDING. In this ERD segment, it is clear that some buildings do not contain (class) rooms. For example, a storage building might not contain any named rooms at all.

FIGURE 4.34 The ninth Tiny College ERD segment



Using the preceding summary, you can identify the following entities:

SCHOOL	COURSE
DEPARTMENT	CLASS
PROFESSOR	STUDENT
BUILDING	ROOM

ENROLL (the associative entity between STUDENT and CLASS)

Once you have discovered the relevant entities, you can define the initial set of relationships among them. Next, you describe the entity attributes. Identifying the attributes of the entities helps you to better understand the relationships among entities. Table 4.4 summarizes the ERM's components, and names the entities and their relations.

TABLE 4.4 Components of the ERM

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrolls in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS
*Note: ENROLL is the composite entity that implements the M:N relationship "*STUDENT enrolls in CLASS.*"*			

You must also define the connectivity and cardinality for the just-discovered relations based on the business rules. However, to avoid crowding the diagram, the cardinalities are not shown. Figure 4.35 shows the Crow's Foot ERD for Tiny College. Note that this is an implementation-ready model. Therefore it shows the ENROLL composite entity.

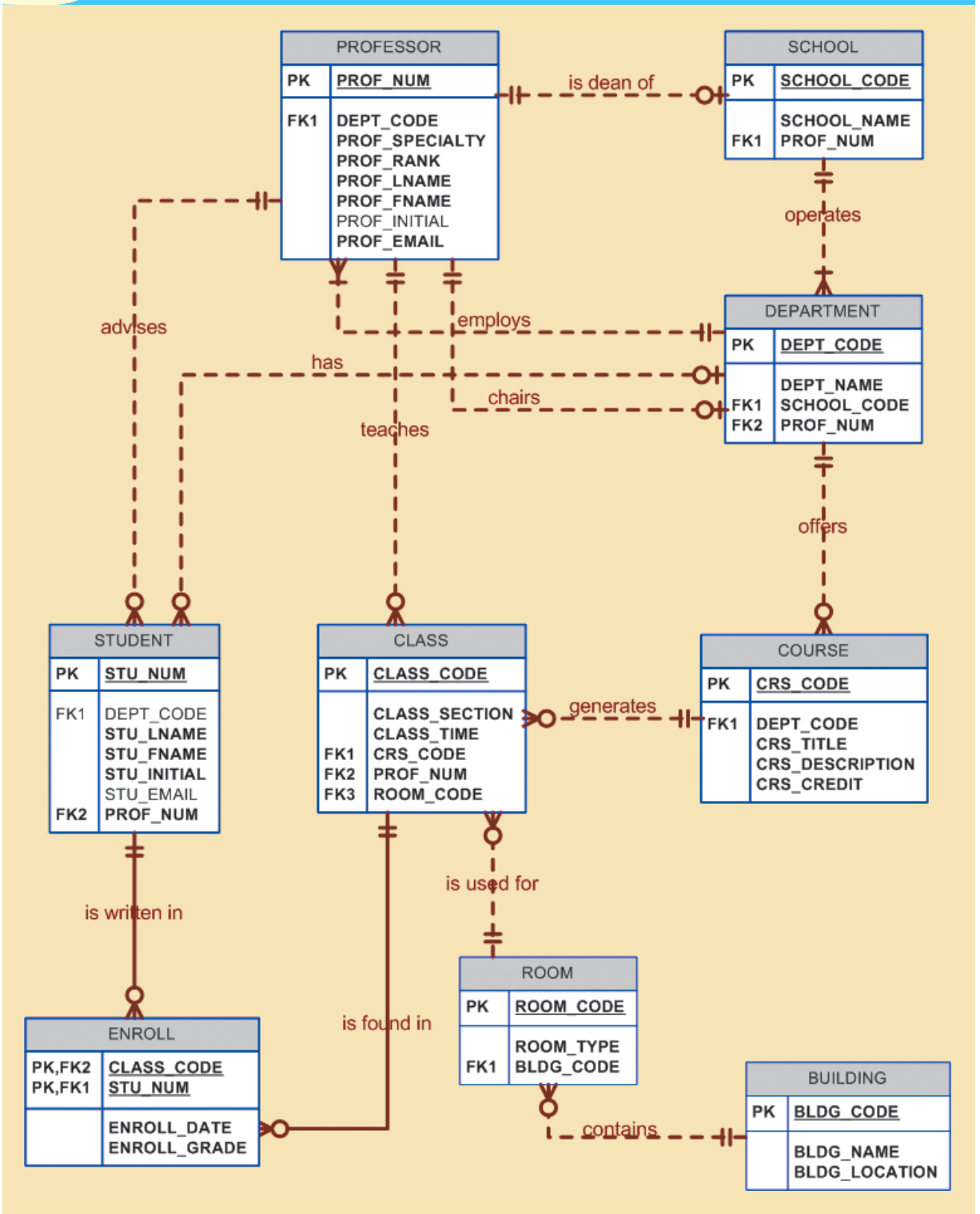
Figure 4.36 shows the conceptual UML class diagram for Tiny College. Note that this class diagram depicts the M:N relationship between STUDENT and CLASS. Figure 4.37 shows the implementation-ready UML class diagram for Tiny College (note that the ENROLL composite entity is shown in this class diagram.

4.3 DATABASE DESIGN CHALLENGES: CONFLICTING GOALS

Database designers often must make design compromises that are triggered by conflicting goals, such as adherence to design standards (design elegance), processing speed, and information requirements.

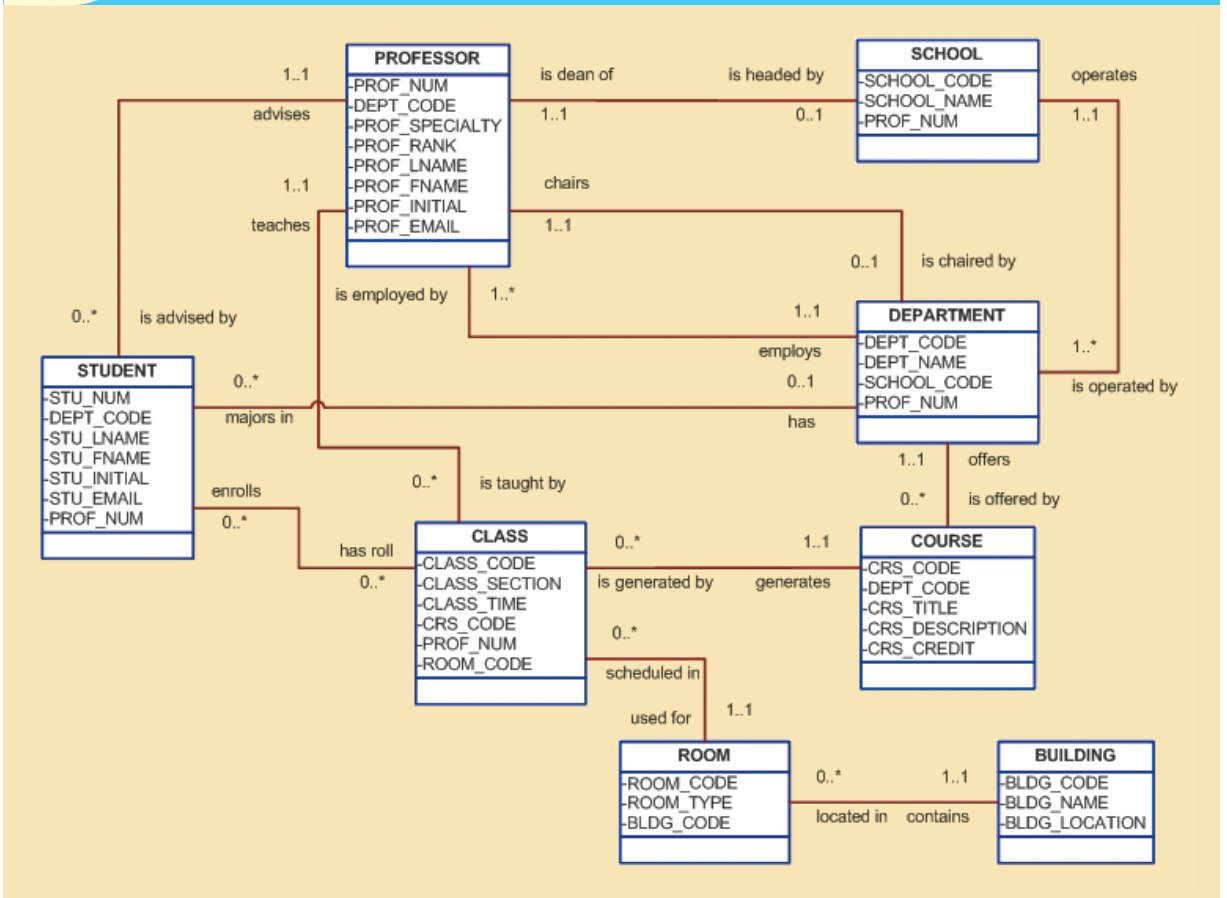
- *Design standards*. The database design must conform to design standards. Such standards have guided you in developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur. You have also learned how standards prescribe avoiding nulls to the greatest extent possible. In fact, you have learned that design standards govern the presentation of all components within the database design. In short, design standards allow you to work with well-defined components and to evaluate the interaction of those components with some precision. Without design standards, it is nearly impossible to formulate a proper design process, to evaluate an existing design, or to trace the likely logical impact of changes in design.
- *Processing speed*. In many organizations, particularly those generating large numbers of transactions, high processing speeds are often a top priority in database design. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships. For example, a "perfect" design might use a 1:1 relationship to avoid nulls, while a higher transaction-speed design might combine the two tables to avoid the use of an additional relationship, using dummy entries to avoid the nulls. If the focus is on data-retrieval speed, you might also be forced to include derived attributes in the design.
- *Information requirements.* The quest for timely information might be the focus of database design. Complex information requirements may dictate data transformations, and they may expand the number of entities and

FIGURE 4.35 The completed Tiny College ERD



attributes within the design. Therefore, the database may have to sacrifice some of its "clean" design structures and/or some of its high transaction speed to ensure maximum information generation. For example, suppose

FIGURE 4.36 The conceptual UML class diagram for Tiny College

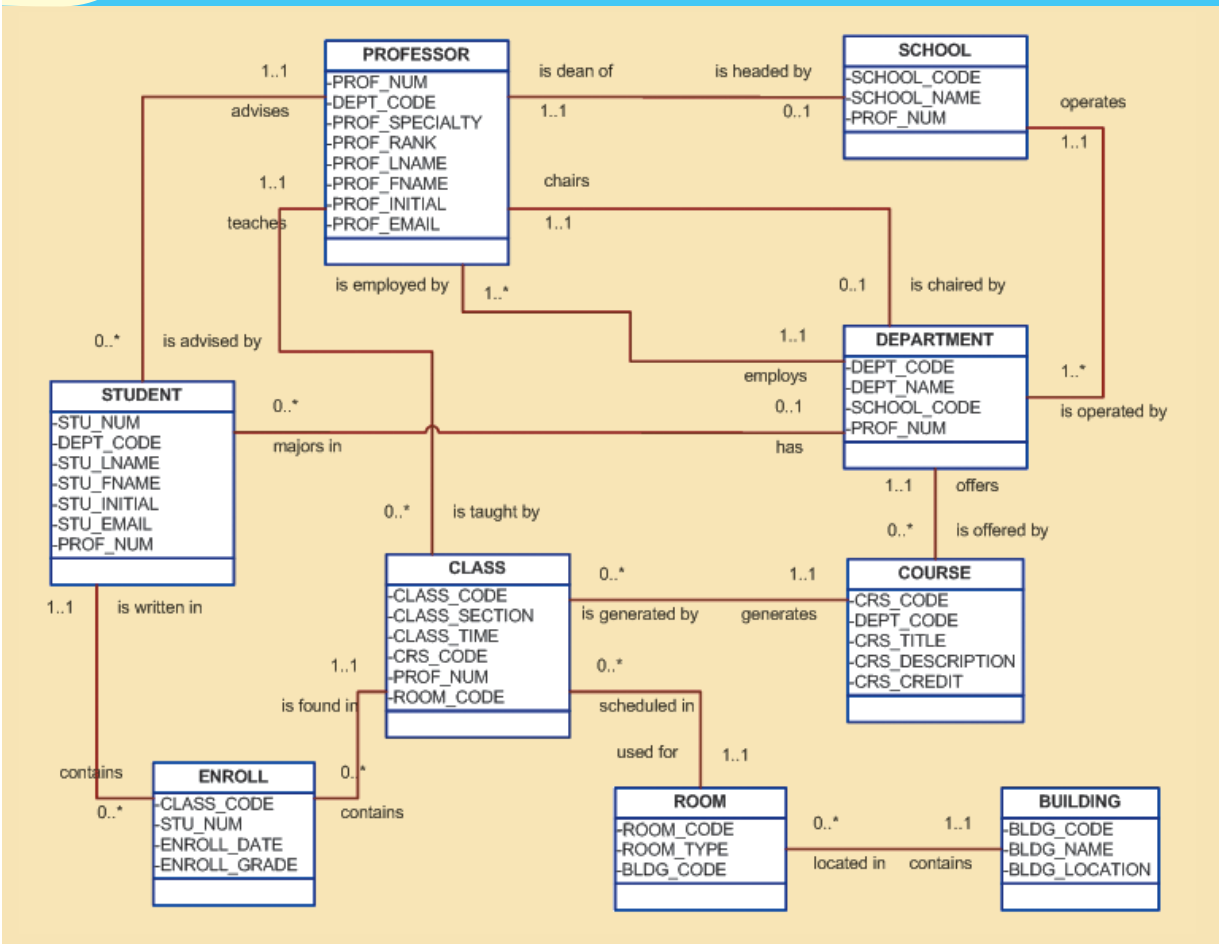


that a detailed sales report must be generated periodically. The sales report includes all invoice subtotals, taxes, and totals; even the invoice lines include subtotals. If the sales report includes hundreds of thousands (or even millions) of invoices, computing the totals, taxes, and subtotals is likely to take some time. If those computations had been made and the results had been stored as derived attributes in the INVOICE and LINE tables at the time of the transaction, the real-time transaction speed might have declined. But that loss of speed would only be noticeable if there were many simultaneous transactions. The cost of a slight loss of transaction speed at the front end and the addition of multiple derived attributes is likely to pay off when the sales reports are generated (not to mention the fact that it will be simpler to generate the queries).

A design that meets all logical requirements and design conventions is an important goal. However, if this perfect design fails to meet the customer's transaction speed and/or information requirements, the designer will not have done a proper job from the end user's point of view. Compromises are a fact of life in the real world of database design.

Even while focusing on the entities, attributes, relationships, and constraints, the designer should begin thinking about end-user requirements such as performance, security, shared access, and data integrity. The designer must consider processing requirements and verify that all update, retrieval, and deletion options are available. Finally, a design is of little value unless the end product is capable of delivering all specified query and reporting requirements.

FIGURE 4.37 The implementation-ready UML class diagram for Tiny College



You are quite likely to discover that even the best design process produces an ERD that requires further changes mandated by operational requirements. Such changes should not discourage you from using the process. ER modeling is essential in the development of a sound design that is capable of meeting the demands of adjustment and growth. Using ERDs yields perhaps the richest bonus of all: a thorough understanding of how an organization really functions.

There are occasional design and implementation problems that do not yield "clean" implementation solutions. To get a sense of the design and implementation choices a database designer faces, let's revisit the 1:1 recursive relationship "EMPLOYEE is married to EMPLOYEE" first examined in Figure 4.18. Figure 4.38 shows three different ways of implementing such a relationship.

Note that the EMPLOYEE_V1 table in Figure 4.38 is likely to yield data anomalies. For example, if Anne Jones divorces Anton Shapiro, two records must be updated—by setting the respective EMP_SPOUSE values to null—to properly reflect that change. If only one record is updated, inconsistent data occur. The problem becomes even worse if several of the divorced employees then marry each other. In addition, that implementation also produces undesirable nulls for employees who are *not* married to other employees in the company.

Another approach would be to create a new entity shown as MARRIED_V1 in a 1:M relationship with EMPLOYEE. (See Figure 4.38.) This second implementation does eliminate the nulls for employees who are not married to somebody working for the same company. (Such employees would not be entered in the MARRIED_V1 table.) However, this approach still yields possible duplicate values. For example, the marriage between employees 345 and

FIGURE 4.38 Various implementations of the 1:1 recursive relationship

Table name: EMPLOYEE_V1 — **Database name: Ch04_PartCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
345	Ramirez	James	347
346	Jones	Anne	349
347	Ramirez	Louise	345
348	Delaney	Robert	
349	Shapiro	Anton	346

First implementation

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME
345	Ramirez	James
346	Jones	Anne
347	Ramirez	Louise
348	Delaney	Robert
349	Shapiro	Anton

Table name: MARRIED_V1

EMP_NUM	EMP_SPOUSE
345	347
346	349
347	345
349	346

Second implementation

Table name: MARRIAGE

MAR_NUM	MAR_DATE
1	04-Mar-03
2	02-Feb-99

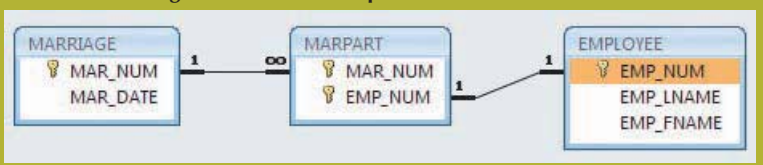
Table name: MARPART

MAR_NUM	EMP_NUM
1	345
1	347
2	346
2	349

Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME
345	Ramirez	James
346	Jones	Anne
347	Ramirez	Louise
348	Delaney	Robert
349	Shapiro	Anton

The relational diagram for the third implementation



Third implementation

347 may still appear twice, once as 345,347 and once as 347,345. (Since each of those permutations is unique the first time it appears, the creation of a unique index will not solve the problem.)

As you can see, the first two implementations yield several problems:

- Both solutions use synonyms. The EMPLOYEE_V1 table uses EMP_NUM and EMP_SPOUSE to refer to an employee. The MARRIED_V1 table uses the same synonyms.
- Both solutions are likely to produce inconsistent data. For example, it is possible to enter employee 345 as married to employee 347 and to enter employee 348 as married to employee 345.
- Both solutions allow data entries to show one employee married to several other employees. For example, it is possible to have data pairs such as 345,347 and 348,347 and 349,347, none of which will violate entity integrity requirements, because they are all unique.

A third approach would be to have two new entities, MARRIAGE and MARPART, in a 1:M relationship. MARPART contains the EMP_NUM foreign key to EMPLOYEE. (See the relational diagram in Figure 4.38.) But even this approach has issues. It requires the collection of additional data regarding the employees' marriage—the marriage

date. If the business users do not need this data, then requiring them to collect it would be inappropriate. To ensure that an employee occurs only once in any given marriage, you would have to create a unique index on the EMP_NUM attribute in the MARPART table. Another potential problem with this solution is that the database implementation will allow more than two employees to "participate" in the same marriage.

As you can see, a recursive 1:1 relationship yields many different solutions with varying degrees of effectiveness and adherence to basic design principles. Any of the above solutions would likely involve the creation of program code to help ensure the integrity and consistency of the data. In a later chapter, we will examine the creation of database triggers that can do exactly that. Your job as a database designer is to use your professional judgment to yield a solution that meets the requirements imposed by business rules, processing requirements, and basic design principles.

Finally, document, document, and document! Put all design activities in writing. Then review what you've written. Documentation not only helps you stay on track during the design process, but also enables you (or those following you) to pick up the design thread when the time comes to modify the design. Although the need for documentation should be obvious, one of the most vexing problems in database and systems analysis work is that the "put it in writing" rule is often not observed in all of the design and implementation stages. The development of organizational documentation standards is a very important aspect of ensuring data compatibility and coherence.

SUMMARY

- The ERM uses ERDs to represent the conceptual database as viewed by the end user. The ERM's main components are entities, relationships, and attributes. The ERD also includes connectivity and cardinality notations. An ERD can also show relationship strength, relationship participation (optional or mandatory), and degree of relationship (unary, binary, ternary, etc.).
- Connectivity describes the relationship classification (1:1, 1:M, or M:N). Cardinality expresses the specific number of entity occurrences associated with an occurrence of a related entity. Connectivities and cardinalities are usually based on business rules.
- In the ERM, an M:N relationship is valid at the conceptual level. However, when implementing the ERM in a relational database, the M:N relationship must be mapped to a set of 1:M relationships through a composite entity.
- ERDs may be based on many different ERMs. However, regardless of which model is selected, the modeling logic remains the same. Because no ERM can accurately portray all real-world data and action constraints, application software must be used to augment the implementation of at least some of the business rules.
- Unified Modeling Language (UML) class diagrams are used to represent the static data structures in a data model. The symbols used in the UML class and ER diagrams are very similar. The UML class diagrams can be used to depict data models at the conceptual or implementation abstraction levels.
- Database designers, no matter how well they are able to produce designs that conform to all applicable modeling conventions, are often forced to make design compromises. Those compromises are required when end users have vital transaction-speed and/or information requirements that prevent the use of "perfect" modeling logic and adherence to all modeling conventions. Therefore, database designers must use their professional judgment to determine how and to what extent the modeling conventions are subject to modification. To ensure that their professional judgments are sound, database designers must have detailed and in-depth knowledge of data-modeling conventions. It is also important to document the design process from beginning to end, which helps keep the design process on track and allows for easy modifications down the road.

KEY TERMS

binary relationship, 116
cardinality, 107
composite attribute, 103
composite identifier, 102
connectivity, 107
derived attribute, 105
existence-dependent, 108
existence-independent, 108
identifiers, 101
identifying relationship, 110
iterative process, 123
mandatory participation, 113
multivalued attributes, 103
non-identifying relationship, 109
optional attribute, 101
optional participation, 113
participants, 105
recursive relationship, 116
regular entity, 108
relationship degree, 116
required attribute, 101
simple attribute, 103
single-valued attribute, 103
strong entity, 108
strong relationship, 110
ternary relationship, 116
unary relationship, 116
weak entity, 110
weak relationship, 109

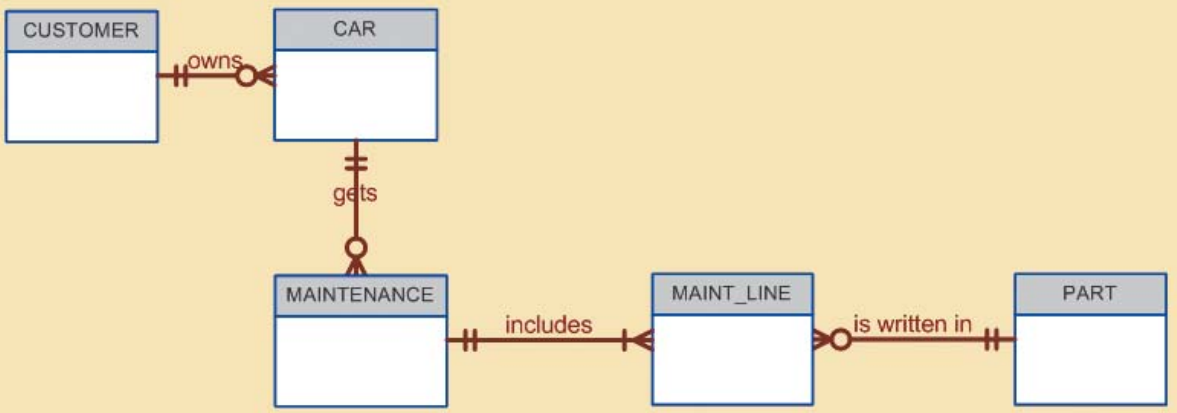
ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Premium Website for this book.

REVIEW QUESTIONS

1. What two conditions must be met before an entity can be classified as a weak entity? Give an example of a weak entity.
2. What is a strong (or identifying) relationship, and how is it depicted in a Crow's Foot ERD?
3. Given the business rule "an employee may have many degrees," discuss its effect on attributes, entities, and relationships. (*Hint:* Remember what a multivalued attribute is and how it might be implemented.)
4. What is a composite entity, and when is it used?
5. Suppose you are working within the framework of the conceptual model in Figure Q4.5.

FIGURE Q4.5 **The conceptual model for Question 5**



Given the conceptual model in Figure Q4.5:

a. Write the business rules that are reflected in it.

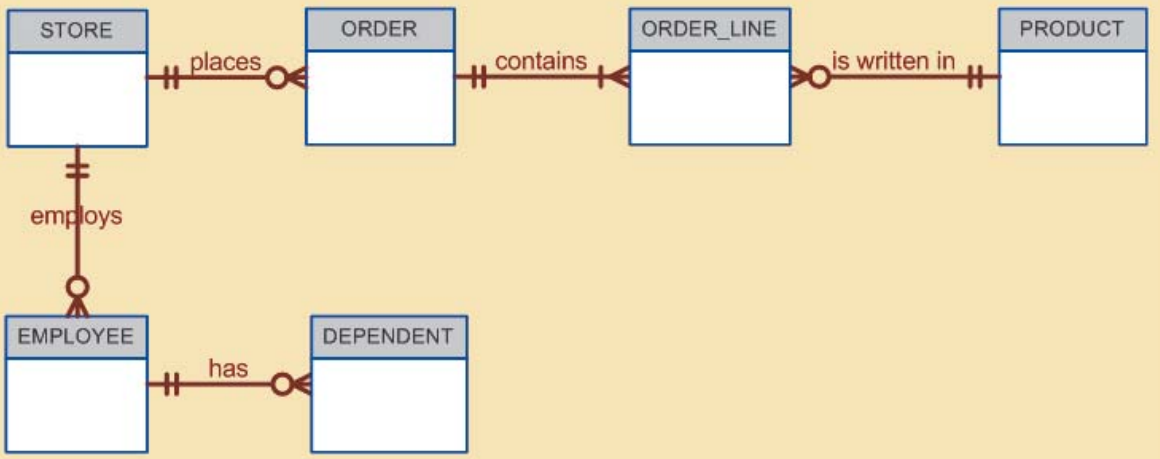
b. Identify all of the cardinalities.

6. What is a recursive relationship? Give an example.
7. How would you (graphically) identify each of the following ERM components in a Crow's Foot notation?
   a. an entity
   b. the cardinality (0,N)
   c. a weak relationship
   d. a strong relationship
8. Discuss the difference between a composite key and a composite attribute. How would each be indicated in an ERD?
9. What two courses of action are available to a designer encountering a multivalued attribute?

10. What is a derived attribute? Give an example.
11. How is a relationship between entities indicated in an ERD? Give an example, using the Crow's Foot notation.
12. Discuss two ways in which the 1:M relationship between COURSE and CLASS can be implemented. (*Hint:* Think about relationship strength.)
13. How is a composite entity represented in an ERD, and what is its function? Illustrate the Crow's Foot notation.
14. What three (often conflicting) database requirements must be addressed in database design?
15. Briefly, but precisely, explain the difference between single-valued attributes and simple attributes. Give an example of each.
16. What are multivalued attributes, and how can they be handled within the database design?

The next four questions are based on the ERD in Figure Q4.17.

FIGURE Q4.17 **The ERD for Questions 17–20**



17. Write the 10 cardinalities that are appropriate for this ERD.
18. Write the business rules reflected in this ERD.
19. What two attributes must be contained in the composite entity between STORE and PRODUCT? Use proper terminology in your answer.
20. Describe precisely the composition of the DEPENDENT weak entity's primary key. Use proper terminology in your answer.
21. The local city youth league needs a database system to help track children who sign up to play soccer. Data need to be kept on each team and the children who will be playing on each team and their parents. Also, data need to be kept on the coaches for each team.

    Draw the data model described below.

    Entities required: Team, Player, Coach, and Parent.

    Attributes required:

    Team: Team ID number, Team name, and Team colors.

    Player: Player ID number, Player first name, Player last name, and Player age.

    Coach: Coach ID number, Coach first name, Coach last name, and Coach home phone number.

    Parent: Parent ID number, Parent last name, Parent first name, Home phone number, and Home Address (Street, City, State, and Zip Code).

The following relationships must be defined:

- Team is related to Player.
- Team is related to Coach.
- Player is related to Parent.

Connectivities and participations are defined as follows:

- A Team may or may not have a Player.
- A Player must have a Team.
- A Team may have many Players.
- A Player has only one Team.
- A Team may or may not have a Coach.
- A Coach must have a Team.
- A Team may have many Coaches.
- A Coach has only one Team.
- A Player must have a Parent.
- A Parent must have a Player.
- A Player may have many Parents.
- A Parent may have many Players.

PROBLEMS

1. Use the following business rules to create a Crow's Foot ERD. Write all appropriate connectivities and cardinalities in the ERD.
   a. A department employs many employees, but each employee is employed by only one department.
   b. Some employees, known as "rovers," are not assigned to any department.
   c. A division operates many departments, but each department is operated by only one division.
   d. An employee may be assigned many projects, and a project may have many employees assigned to it.
   e. A project must have at least one employee assigned to it.
   f. One of the employees manages each department, and each department is managed by only one employee.
   g. One of the employees runs each division, and each division is run by only one employee.
2. The Jonesburgh County Basketball Conference (JCBC) is an amateur basketball association. Each city in the county has one team as its representative. Each team has a maximum of 12 players and a minimum of 9 players. Each team also has up to three coaches (offensive, defensive, and physical training coaches). During the season, each team plays two games (home and visitor) against each of the other teams. Given those conditions, do the following:
   a. Identify the connectivity of each relationship.
   b. Identify the type of dependency that exists between CITY and TEAM.
   c. Identify the cardinality between teams and players and between teams and city.
   d. Identify the dependency between coach and team and between team and player.
   e. Draw the Chen and Crow's Foot ERDs to represent the JCBC database.
   f. Draw the UML class diagram to depict the JCBC database.
3. Create an ERD based on the Crow's Foot notation, using the following requirements:
   a. An INVOICE is written by a SALESREP. Each sales representative can write many invoices, but each invoice is written by a single sales representative.
   b. The INVOICE is written for a single CUSTOMER. However, each customer can have many invoices.

c. An INVOICE can include many detail lines (LINE), each of which describes one product bought by the customer.

d. The product information is stored in a PRODUCT entity.

e. The product's vendor information is found in a VENDOR entity.

4. The Hudson Engineering Group (HEG) has contacted you to create a conceptual model whose application will meet the expected database requirements for the company's training program. The HEG administrator gives you the description (see below) of the training group's operating environment. (*Hint:* Some of the following sentences identify the volume of data rather than cardinalities. Can you tell which ones?)

   The HEG has 12 instructors and can handle up to 30 trainees per class. HEG offers five Advanced Technology courses, each of which may generate several classes. If a class has fewer than 10 trainees, it will be canceled. Therefore, it is possible for a course not to generate any classes. Each class is taught by one instructor. Each instructor may teach up to two classes or may be assigned to do research only. Each trainee may take up to two classes per year.

   Given that information, do the following:

   a. Define all of the entities and relationships. (Use Table 4.4 as your guide.)

   b. Describe the relationship between instructor and class in terms of connectivity, cardinality, and existence dependence.

5. Automata, Inc. produces specialty vehicles by contract. The company operates several departments, each of which builds a particular vehicle, such as a limousine, a truck, a van, or an RV.

   - Before a new vehicle is built, the department places an order with the purchasing department to request specific components. Automata's purchasing department is interested in creating a database to keep track of orders and to accelerate the process of delivering materials.
   - The order received by the purchasing department may contain several different items. An inventory is maintained so that the most frequently requested items are delivered almost immediately. When an order comes in, it is checked to determine whether the requested item is in inventory. If an item is not in inventory, it must be ordered from a supplier. Each item may have several suppliers.

   Given that functional description of the processes encountered at Automata's purchasing department, do the following:

   a. Identify all of the main entities.

   b. Identify all of the relations and connectivities among entities.

   c. Identify the type of existence dependence in all the relationships.

   d. Give at least two examples of the types of reports that can be obtained from the database.

6. United Helpers is a nonprofit organization that provides aid to people after natural disasters. Based on the following brief description of operations, create the appropriate fully labeled Crow's Foot ERD.

   - Individuals volunteer their time to carry out the tasks of the organization. The name, address, and telephone number for each voluteer are tracked. Each volunteer may be assigned to several tasks during the time that he or she is doing volunteer work, and some tasks require many volunteers. It is possible for a volunteer to be in the system without having been assigned a task yet. It is possible to have tasks that no one has been assigned. When a volunteer is assigned to a task, the system should track the start time and end time of that assignment.
   - For each task, there is a task code, task description, task type, and task status. For example, there may be a task with task code "101," a description of "answer the telephone," a type of "recurring," and a status of "ongoing." There could be another task with a code of "102," a description of "prepare 5000 packages of basic medical supplies," a type of "packing," and a status of "open."

- For all tasks of type "packing," there is a packing list that specifies the contents of the packages. There are many different packing lists to produce different packages, such as basic medical packages, child-care packages, food packages, etc. Each packing list has a packing list ID number, a packing list name, and a packing list description, which describes the items that ideally go into making that type of package. Every packing task is associated with only one packing list. A packing list may not be associated with any tasks, or may be associated with many tasks. Tasks that are not packing tasks are not associated with any packing list.
- Packing tasks result in the creation of packages. Each individual package of supplies that is produced by the organization is tracked. Each package is assigned an ID number. The date the package was created and the total weight of the package are recorded. A given package is associated with only one task. Some tasks (e.g., "answer the phones") will not have produced any packages, while other tasks (e.g., "prepare 5000 packages of basic medical supplies") will be associated with many packages.
- The packing list describes the *ideal* contents of each package, but it is not always possible to include the ideal number of each item. Therefore, the actual items included in each package should be tracked. A package can contain many different items, and a given item can be used in many different packages.
- For each item that the organization provides, there is an item ID number, item description, item value, and item quantity on hand stored in the system. Along with tracking the actual items that are placed in each package, the quantity of each item placed in the package must be tracked too. For example, a packing list may state that basic medical packages should include 100 bandages, 4 bottles of iodine, and 4 bottles of hydrogen peroxide. However, because of the limited supply of items, a given package may include only 10 bandages, 1 bottle of iodine, and no hydrogen peroxide. The fact that this package includes bandages and iodine needs to be recorded along with the quantity of each that is included. It is possible for the organization to have items donated that have not been included in any package yet, but every package will contain at least one item.

7. Using the Crow's Foot notation, create an ERD that can be implemented for a medical clinic, using the following business rules:
   - A patient can make many appointments with one or more doctors in the clinic, and a doctor can accept appointments with many patients. However, each appointment is made with only one doctor and one patient.
   - Emergency cases do not require an appointment. However, for appointment management purposes, an emergency is entered in the appointment book as "unscheduled."
   - If kept, an appointment yields a visit with the doctor specified in the appointment. The visit yields a diagnosis and, when appropriate, treatment.
   - With each visit, the patient's records are updated to provide a medical history.
   - Each patient visit creates a bill. Each patient visit is billed by one doctor, and each doctor can bill many patients.
   - Each bill must be paid. However, a bill may be paid in many installments, and a payment may cover more than one bill.
   - A patient may pay the bill directly, or the bill may be the basis for a claim submitted to an insurance company.
   - If the bill is paid by an insurance company, the deductible is submitted to the patient for payment.

NOTE

The following cases and additional problems from the Instructor Online Companion may be used as the basis for class projects. These problems illustrate the challenge of translating a description of operations into a set of business rules that will define the components for an ERD that can be successfully implemented. These problems can also be used as the basis for discussions about the components and contents of a proper description of operations. One of the things you must learn if you want to create databases that can be successfully implemented is to separate the generic background material from the details that directly affect database design. You must also keep in mind that many constraints cannot be incorporated into the database design; instead, such constraints are handled by the applications software.

CASES

8. The administrators of Tiny College are so pleased with your design and implementation of their student registration/tracking system that they want you to expand the design to include the database for their motor vehicle pool. A brief description of operations follows:
   - Faculty members may use the vehicles owned by Tiny College for officially sanctioned travel. For example, the vehicles may be used by faculty members to travel to off-campus learning centers, to travel to locations at which research papers are presented, to transport students to officially sanctioned locations, and to travel for public service purposes. The vehicles used for such purposes are managed by Tiny College's Travel Far But Slowly (TFBS) Center.
   - Using reservation forms, each department can reserve vehicles for its faculty, who are responsible for filling out the appropriate trip completion form at the end of a trip. The reservation form includes the expected departure date, vehicle type required, destination, and name of the authorized faculty member. The faculty member arriving to pick up a vehicle must sign a checkout form to log out the vehicle and pick up a trip completion form. (The TFBS employee who releases the vehicle for use also signs the checkout form.) The faculty member's trip completion form includes the faculty member's identification code, the vehicle's identification, the odometer readings at the start and end of the trip, maintenance complaints (if any), gallons of fuel purchased (if any), and the Tiny College credit card number used to pay for the fuel. If fuel is purchased, the credit card receipt must be stapled to the trip completion form. Upon receipt of the faculty trip completion form, the faculty member's department is billed at a mileage rate based on the vehicle type (sedan, station wagon, panel truck, minivan, or minibus) used. (*Hint:* Do *not* use more entities than are necessary. Remember the difference between attributes and entities!)
   - All vehicle maintenance is performed by TFBS. Each time a vehicle requires maintenance, a maintenance log entry is completed on a prenumbered maintenance log form. The maintenance log form includes the vehicle identification, a brief description of the type of maintenance required, the initial log entry date, the date on which the maintenance was completed, and the identification of the mechanic who released the vehicle back into service. (Only mechanics who have an inspection authorization may release the vehicle back into service.)
   - As soon as the log form has been initiated, the log form's number is transferred to a maintenance detail form; the log form's number is also forwarded to the parts department manager, who fills out a parts usage form on which the maintenance log number is recorded. The maintenance detail form contains separate lines for each maintenance item performed, for the parts used, and for identification of the mechanic who performed the maintenance item. When all maintenance items have been completed, the maintenance detail form is stapled to the maintenance log form, the maintenance log form's completion date is filled out, and the mechanic who releases the vehicle back into service signs the form. The stapled forms are then filed, to be used later as the source for various maintenance reports.

- TFBS maintains a parts inventory, including oil, oil filters, air filters, and belts of various types. The parts inventory is checked daily to monitor parts usage and to reorder parts that reach the "minimum quantity on hand" level. To track parts usage, the parts manager requires each mechanic to sign out the parts that are used to perform each vehicle's maintenance; the parts manager records the maintenance log number under which the part is used.
- Each month TFBS issues a set of reports. The reports include the mileage driven by vehicle, by department, and by faculty members within a department. In addition, various revenue reports are generated by vehicle and department. A detailed parts usage report is also filed each month. Finally, a vehicle maintenance summary is created each month.

Given that brief summary of operations, draw the appropriate (and fully labeled) ERD. Use the Chen methodology to indicate entities, relationships, connectivities, and cardinalities.

9. During peak periods, Temporary Employment Corporation (TEC) places temporary workers in companies. TEC's manager gives you the following description of the business:
   - TEC has a file of candidates who are willing to work.
   - If the candidate has worked before, that candidate has a specific job history. (Naturally, no job history exists if the candidate has never worked.) Each time the candidate works, one additional job history record is created.
   - Each candidate has earned several qualifications. Each qualification may be earned by more than one candidate. (For example, it is possible for more than one candidate to have earned a Bachelor of Business Administration degree or a Microsoft Network Certification. And clearly, a candidate may have earned both a BBA and a Microsoft Network Certification.)
   - TEC offers courses to help candidates improve their qualifications.
   - Every course develops one specific qualification; however, TEC does not offer a course for every qualification. Some qualifications have multiple courses that develop that qualification.
   - Some courses cover advanced topics that require specific qualifications as prerequisites. Some courses cover basic topics that do not require any prerequisite qualifications. A course can have several prerequisites. A qualification can be a prerequisite for more than one course.
   - Courses are taught during training sessions. A training session is the presentation of a single course. Over time, TEC will offer many training sessions for each course; however, new courses may not have any training sessions scheduled right away.
   - Candidates can pay a fee to attend a training session. A training session can accommodate several candidates, although new training sessions will not have any candidates registered at first.
   - TEC also has a list of companies that request temporaries.
   - Each time a company requests a temporary employee, TEC makes an entry in the Openings folder. That folder contains an opening number, a company name, required qualifications, a starting date, an anticipated ending date, and hourly pay.
   - Each opening requires only one specific or main qualification.
   - When a candidate matches the qualification, the job is assigned, and an entry is made in the Placement Record folder. That folder contains an opening number, a candidate number, the total hours worked, etc. In addition, an entry is made in the job history for the candidate.
   - An opening can be filled by many candidates, and a candidate can fill many openings.
   - TEC uses special codes to describe a candidate's qualifications for an opening. The list of codes is shown in Table P4.9.

TABLE P4.9

CODE	DESCRIPTION
SEC-45	Secretarial work, at least 45 words per minute
SEC-60	Secretarial work, at least 60 words per minute
CLERK	General clerking work
PRG-VB	Programmer, Visual Basic
PRG-C++	Programmer, C++
DBA-ORA	Database Administrator, Oracle
DBA-DB2	Database Administrator, IBM DB2
DBA-SQLSERV	Database Administrator, MS SQL Server
SYS-1	Systems Analyst, level 1
SYS-2	Systems Analyst, level 2
NW-NOV	Network Administrator, Novell experience
WD-CF	Web Developer, ColdFusion

TEC's management wants to keep track of the following entities:

COMPANY, OPENING, QUALIFICATION, CANDIDATE, JOB_HISTORY, PLACEMENT, COURSE, and SESSION. Given that information, do the following:

a. Draw the Crow's Foot ERDs for this enterprise.

b. Identify all necessary relationships.

c. Identify the connectivity for each relationship.

d. Identify the mandatory/optional dependencies for the relationships.

e. Resolve all M:N relationships.

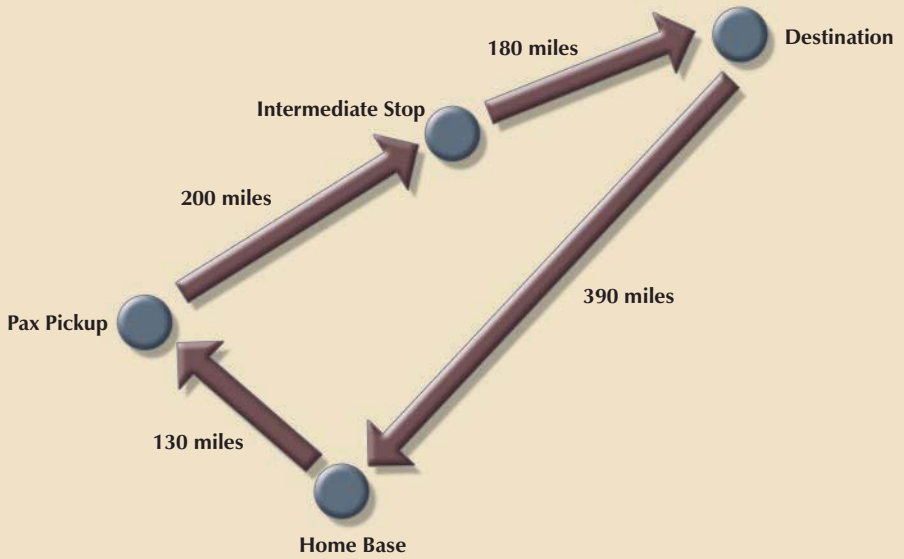
10. Use the following description of the operations of the RC_Charter2 Company to complete this exercise.

- The RC_Charter2 Company operates a fleet of aircraft under the Federal Air Regulations (FAR) Part 135 (air taxi or charter) certificate, enforced by the FAA. The aircraft are available for air taxi (charter) operations within the United States and Canada.
- Charter companies provide so-called "unscheduled" operations—that is, charter flights take place only after a customer reserves the use of an aircraft to fly at a customer-designated date and time to one or more customer-designated destinations, transporting passengers, cargo, or some combination of passengers and cargo. A customer can, of course, reserve many different charter flights (trips) during any time frame. However, for billing purposes, each charter trip is reserved by one and only one customer. Some of RC_Charter2's customers do not use the company's charter operations; instead, they purchase fuel, use maintenance services, or use other RC_Charter2 services. However, this database design will focus on the charter operations only.
- Each charter trip yields revenue for the RC_Charter2 Company. This revenue is generated by the charges a customer pays upon the completion of a flight. The charter flight charges are a function of aircraft model used, distance flown, waiting time, special customer requirements, and crew expenses. The distance flown charges are computed by multiplying the round-trip miles by the model's charge per mile. Round-trip miles are based on the actual navigational path flown. The sample route traced in Figure P4.10 illustrates the procedure. Note that the number of round-trip miles is calculated to be 130 + 200 + 180 + 390 = 900.

Depending on whether a customer has RC_Charter2 credit authorization, the customer may:

- Pay the entire charter bill upon the completion of the charter flight.
- Pay a part of the charter bill and charge the remainder to the account. The charge amount may not exceed the available credit.
- Charge the entire charter bill to the account. The charge amount may not exceed the available credit.
- Customers may pay all or part of the existing balance for previous charter trips. Such payments may be

FIGURE P4.10 Round-trip mile determination



made at any time and are not necessarily tied to a specific charter trip. The charter mileage charge includes the expense of the pilot(s) and other crew required by FAR 135. However, if customers request *additional* crew *not* required by FAR 135, those customers are charged for the crew members on an hourly basis. The hourly crew-member charge is based on each crew member's qualifications.

- The database must be able to handle crew assignments. Each charter trip requires the use of an aircraft, and a crew flies each aircraft. The smaller piston-engine-powered charter aircraft require a crew consisting of only a single pilot. Larger aircraft (i.e., aircraft having a gross takeoff weight of 12,500 pounds or more) and jet-powered aircraft require a pilot and a copilot, while some of the larger aircraft used to transport passengers may require flight attendants as part of the crew. Some of the older aircraft require the assignment of a flight engineer, and larger cargo-carrying aircraft require the assignment of a loadmaster. In short, a crew can consist of more than one person, and not all crew members are pilots.
- The charter flight's aircraft waiting charges are computed by multiplying the hours waited by the model's hourly waiting charge. Crew expenses are limited to meals, lodging, and ground transportation.

The RC_Charter2 database must be designed to generate a monthly summary of all charter trips, expenses, and revenues derived from the charter records. Such records are based on the data that each pilot in command is required to record for each charter trip: trip date(s) and time(s), destination(s), aircraft number, pilot (and other crew) data, distance flown, fuel usage, and other data pertinent to the charter flight. Such charter data are then used to generate monthly reports that detail revenue and operating cost information for customers, aircraft, and pilots. All pilots and other crew members are RC_Charter2 Company employees; that is, the company does not use contract pilots and crew.

FAR Part 135 operations are conducted under a strict set of requirements that govern the licensing and training of crew members. For example, pilots must have earned either a commercial license or an Airline Transport Pilot (ATP) license. Both licenses require appropriate ratings. Ratings are specific competency requirements. For example:

- To operate a multiengine aircraft designed for takeoffs and landings on land only, the appropriate rating is MEL, or Multiengine Landplane. When a multiengine aircraft can take off and land on water, the appropriate rating is MES, or Multiengine Seaplane.

- The instrument rating is based on a demonstrated ability to conduct all flight operations with sole reference to cockpit instrumentation. The instrument rating is required to operate an aircraft under Instrument Meteorological Conditions (IMC), and all such operations are governed under FAR-specified Instrument Flight Rules (IFR). In contrast, operations conducted under "good weather" or *visual* flight conditions are based on the FAR Visual Flight Rules (VFR).
- The type rating is required for all aircraft with a takeoff weight of more than 12,500 pounds or for aircraft that are purely jet-powered. If an aircraft uses jet engines to drive propellers, that aircraft is said to be turboprop-powered. A turboprop—that is, a turbo-propeller-powered aircraft—does not require a type rating unless it meets the 12,500-pound weight limitation.
- Although pilot licenses and ratings are not time-limited, exercising the privilege of the license and ratings under Part 135 requires both *a current medical certificate and a current Part 135 checkride*. The following distinctions are important:
- The medical certificate may be Class I or Class II. The Class I medical is more stringent than the Class II, and it must be renewed every six months. The Class II medical must be renewed yearly. If the Class I medical is not renewed during the six-month period, it automatically reverts to a Class II certificate. If the Class II medical is not renewed within the specified period, it automatically reverts to a Class III medical, which is not valid for commercial flight operations.
- A Part 135 checkride is a practical flight examination that must be successfully completed every six months. The checkride includes all flight maneuvers and procedures specified in Part 135.

Nonpilot crew members must also have the proper certificates in order to meet specific job requirements. For example, loadmasters need an appropriate certificate, as do flight attendants. In addition, crew members such as loadmasters and flight attendants, who may be required in operations that involve large aircraft (more than a 12,500-pound takeoff weight and passenger configurations over 19) are also required periodically to pass a written and practical exam. The RC_Charter2 Company is required to keep a complete record of all test types, dates, and results for each crew member, as well as pilot medical certificate examination dates.

In addition, all flight crew members are required to submit to periodic drug testing; the results must be tracked, too. (Note that nonpilot crew members are not required to take pilot-specific tests such as Part 135 checkrides. Nor are pilots required to take crew tests such as loadmaster and flight attendant practical exams.) However, many crew members have licenses and/or certifications in several areas. For example, a pilot may have an ATP and a loadmaster certificate. If that pilot is assigned to be a loadmaster on a given charter flight, the loadmaster certificate is required. Similarly, a flight attendant may have earned a commercial pilot's license. Sample data formats are shown in Table P4.10.

Pilots and other crew members must receive recurrency training appropriate to their work assignments. Recurrency training is based on an FAA-approved curriculum that is job-specific. For example, pilot recurrency training includes a review of all applicable Part 135 flight rules and regulations, weather data interpretation, company flight operations requirements, and specified flight procedures. The RC_Charter2 Company is required to keep a complete record of all recurrency training for each crew member subject to the training.

The RC_Charter2 Company is required to maintain a detailed record of all crew credentials and all training mandated by Part 135. The company must keep a complete record of each requirement and of all compliance data.

To conduct a charter flight, the company must have a properly maintained aircraft available. A pilot who meets all of the FAA's licensing and currency requirements must fly the aircraft as Pilot in Command (PIC). For those aircraft that are powered by piston engines or turboprops and have a gross takeoff weight under 12,500 pounds, single-pilot operations are permitted under Part 135 as long as a properly maintained autopilot is available. However, even if FAR Part 135 permits single-pilot operations, many customers require the presence of a copilot who is capable of conducting the flight operations under Part 135.

The RC_Charter2 operations manager anticipates the lease of turbojet-powered aircraft, and those aircraft are required to have a crew consisting of a pilot and copilot. Both pilot and copilot must meet the same Part 135 licensing, ratings, and training requirements.

TABLE P4.10

PART A TESTS		
TEST CODE	TEST DESCRIPTION	TEST FREQUENCY
1	Part 135 Flight Check	6 months
2	Medical, Class 1	6 months
3	Medical, Class 2	12 months
4	Loadmaster Practical	12 months
5	Flight Attendant Practical	12 months
6	Drug test	Random
7	Operations, written exam	6 months

PART B RESULTS			
EMPLOYEE	TEST CODE	TEST DATE	TEST RESULT
101	1	12-Nov-09	Pass-1
103	6	23-Dec-09	Pass-1
112	4	23-Dec-09	Pass-2
103	7	11-Jan-10	Pass-1
112	7	16-Jan-10	Pass-1
101	7	16-Jan-10	Pass-1
101	6	11-Feb-10	Pass-2
125	2	15-Feb-10	Pass-1

PART C LICENSES AND CERTIFICATIONS	
LICENSE OR CERTIFICATE	LICENSE OR CERTIFICATE DESCRIPTION
ATP	Airline Transport Pilot
Comm	Commercial license
Med-1	Medical certificate, Class I
Med-2	Medical certificate, Class II
Instr	Instrument rating
MEL	Multiengine Land aircraft rating
LM	Loadmaster
FA	Flight Attendant

EMPLOYEE	LICENSE OR CERTIFICATE	DATE EARNED
101	Comm	12-Nov-93
101	Instr	28-Jun-94
101	MEL	9-Aug-94
103	Comm	21-Dec-95
112	FA	23-Jun-02
103	Instr	18-Jan-96
112	LM	27-Nov-05

The company also leases larger aircraft that exceed the 12,500-pound gross takeoff weight. Those aircraft can carry the number of passengers that requires the presence of one or more flight attendants. If those aircraft carry cargo weighing over 12,500 pounds, a loadmaster must be assigned as a crew member to supervise the loading and securing of the cargo. *The database must be designed to meet the anticipated additional charter crew assignment capability.*

a. Given this incomplete description of operations, write all applicable business rules to establish entities, relationships, optionalities, connectivities, and cardinalities. (*Hint:* Use the following five business rules as examples, writing the remaining business rules in the same format.)

- A customer may request many charter trips.
- Each charter trip is requested by only one customer.

- Some customers have not (yet) requested a charter trip.
- An employee may be assigned to serve as a crew member on many charter trips.
- Each charter trip may have many employees assigned to it to serve as crew members.

b. Draw the fully labeled and implementable Crow's Foot ERD based on the business rules you wrote in Part (a) of this problem. Include all entities, relationships, optionalities, connectivities, and cardinalities.