5 FIVE

NORMALIZATION OF DATABASE TABLES

In this chapter, you will learn:

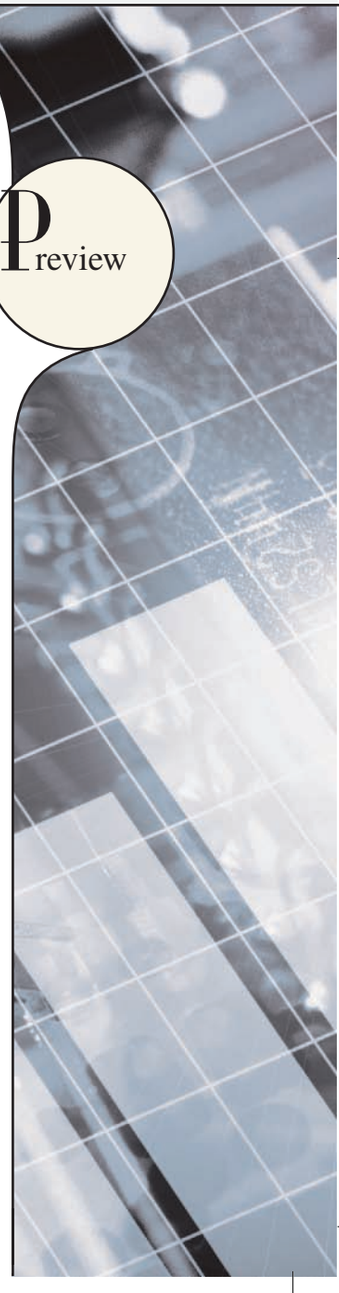
- What normalization is and what role it plays in the database design process
- About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
- How normal forms can be transformed from lower normal forms to higher normal forms
- That normalization and ER modeling are used concurrently to produce a good database design
- That some situations require denormalization to generate information efficiently

Preview

Good database design must be matched to good table structures. In this chapter, you learn to evaluate and design good table structures to control data redundancies, thereby avoiding data anomalies. The process that yields such desirable results is known as normalization.

In order to recognize and appreciate the characteristics of a good table structure, it is useful to examine a poor one. Therefore, the chapter begins by examining the characteristics of a poor table structure and the problems it creates. You then learn how to correct a poor table structure. This methodology will yield important dividends: you will know how to design a good table structure and how to repair an existing poor one.

You will discover not only that data anomalies can be eliminated through normalization, but also that a properly normalized set of table structures is actually less complicated to use than an unnormalized set. In addition, you will learn that the normalized set of table structures more faithfully reflects an organization's real operations.



5.1 DATABASE TABLES AND NORMALIZATION

Having good relational database software is not enough to avoid the data redundancy discussed in Chapter 1, Database Systems. If the database tables are treated as though they are files in a file system, the RDBMS never has a chance to demonstrate its superior data-handling capabilities.

The table is the basic building block in the database design process. Consequently, the table's structure is of great interest. Ideally, the database design process explored in Chapter 4, Entity Relationship (ER) Modeling, yields good table structures. Yet it is possible to create poor table structures even in a good database design. So how do you recognize a poor table structure, and how do you produce a good table? The answer to both questions involves normalization. **Normalization** is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. The normalization process involves assigning attributes to tables based on the concept of determination you learned about in Chapter 3, The Relational Database Model.

Normalization works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF. For most purposes in business database design, 3NF is as high as you need to go in the normalization process. However, you will discover in Section 5.3 that properly designed 3NF structures also meet the requirements of fourth normal form (4NF).

Although normalization is a very important database design ingredient, you should not assume that the highest level of normalization is always the most desirable. Generally, the higher the normal form, the more relational join operations required to produce a specified output and the more resources required by the database system to respond to end-user queries. A successful design must also consider end-user demand for fast performance. Therefore, you will occasionally be expected to *denormalize* some portions of a database design in order to meet performance requirements. **Denormalization** produces a lower normal form; that is, a 3NF will be converted to a 2NF through denormalization. However, *the price you pay for increased performance through denormalization is greater data redundancy*.

NOTE

Although the word *table* is used throughout this chapter, formally, normalization is concerned with *relations*. In Chapter 3 you learned that the terms *table* and *relation* are frequently used interchangeably. In fact, you can say that a table is the "implementation view" of a logical relation that meets some specific conditions (see Table 3.1). However, being more rigorous, the mathematical relation does not allow duplicate tuples, whereas duplicate tuples could exist in tables (see Section 5.5).

5.2 THE NEED FOR NORMALIZATION

To get a better idea of the normalization process, consider the simplified database activities of a construction company that manages several building projects. Each project has its own project number, name, employees assigned to it, and so on. Each employee has an employee number, name, and job classification, such as engineer or computer technician.

The company charges its clients by billing the hours spent on each contract. The hourly billing rate is dependent on the employee's position. For example, one hour of computer technician time is billed at a different rate than one hour of engineer time. Periodically, a report is generated that contains the information displayed in Table 5.1.

The total charge in Table 5.1 is a derived attribute and, at this point, is not stored in the table.

TABLE 5.1 **A Sample Report Layout**

PROJECT NUMBER	PROJECT NAME	EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CLASS	CHARGE/ HOUR	HOURS BILLED	TOTAL CHARGE
15	Evergreen	103	June E. Arbough	Elec. Engineer	$ 85.50	23.8	$ 2,034.90
		101	John G. News	Database Designer	$105.00	19.4	$ 2,037.00
		105	Alice K. Johnson *	Database Designer	$105.00	35.7	$ 3,748.50
		106	William Smithfield	Programmer	$ 35.75	12.6	$ 450.45
		102	David H. Senior	Systems Analyst	$ 96.75	23.8	$ 2,302.65
				Subtotal			**$10,573.50**
18	Amber Wave	114	Annelise Jones	Applications Designer	$ 48.10	25.6	$ 1,183.26
		118	James J. Frommer	General Support	$ 18.36	45.3	$ 831.71
		104	Anne K. Ramoras *	Systems Analyst	$ 96.75	32.4	$ 3,134.70
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	45.0	$ 2,067.75
				Subtotal			**$ 7,265.52**
22	Rolling Tide	105	Alice K. Johnson	Database Designer	$105.00	65.7	$ 6,998.50
		104	Anne K. Ramoras	Systems Analyst	$ 96.75	48.4	$ 4,682.70
		113	Delbert K. Joenbrood	Applications Designer	$ 48.10	23.6	$ 1,135.16
		111	Geoff B. Wabash	Clerical Support	$ 26.87	22.0	$ 591.14
		106	William Smithfield	Programmer	$ 35.75	12.8	$ 457.60
				Subtotal			**$13,765.10**
25	Starflight	107	Maria D. Alonzo	Programmer	$ 35.75	25.6	$ 915.20
		115	Travis B. Bawangi	Systems Analyst	$ 96.75	45.8	$ 4,431.15
		101	John G. News *	Database Designer	$105.00	56.3	$ 5,911.50
		114	Annelise Jones	Applications Designer	$ 48.10	33.1	$ 1,592.11
		108	Ralph B. Washington	Systems Analyst	$ 96.75	23.6	$ 2,283.30
		118	James J. Frommer	General Support	$ 18.36	30.5	$ 559.98
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	41.4	$ 1,902.33
				Subtotal			**$17,595.57**
				Total			**$49,199.69**

Note: * indicates project leader

The easiest short-term way to generate the required report might seem to be a table whose contents correspond to the reporting requirements. See Figure 5.1.

ONLINE CONTENT

The databases used to illustrate the material in this chapter are found in the Student Online Companion.

FIGURE 5.1 **Tabular representation of the report format**

Table name: RPT_FORMAT **Database name: Ch05_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Note that the data in Figure 5.1 reflects the assignment of employees to projects. Apparently, an employee can be assigned to more than one project. For example, Darlene Smithson (EMP_NUM = 112) has been assigned to two projects: Amber Wave and Starflight. Given the structure of the data set, each project includes only a single occurrence of any one employee. Therefore, knowing the PROJ_NUM and EMP_NUM value will let you find the job classification and its hourly charge. In addition, you will know the total number of hours each employee worked on each project. (The total charge—a derived attribute whose value can be computed by multiplying the hours billed and the charge per hour—has not been included in Figure 5.1. No structural harm is done if this derived attribute is included.)

Unfortunately, the structure of the data set in Figure 5.1 does not conform to the requirements discussed in Chapter 3, nor does it handle data very well. Consider the following deficiencies:

1. The project number (PROJ_NUM) is apparently intended to be a primary key or at least a part of a PK, but it contains nulls. (Given the preceding discussion, you know that PROJ_NUM + EMP_NUM will define each row.)
2. The table entries invite data inconsistencies. For example, the JOB_CLASS value "Elect. Engineer" might be entered as "Elect.Eng." in some cases, "El. Eng." in others, and "EE" in still others.
3. The table displays data redundancies. Those data redundancies yield the following anomalies:

   a. *Update anomalies*. Modifying the JOB_CLASS for employee number 105 requires (potentially) many alterations, one for each EMP_NUM = 105.

b. *Insertion anomalies*. Just to complete a row definition, a new employee must be assigned to a project. If the employee is not yet assigned, a phantom project must be created to complete the employee data entry.

c. *Deletion anomalies*. Suppose that only one employee is associated with a given project. If that employee leaves the company and the employee data are deleted, the project information will also be deleted. To prevent the loss of the project information, a fictitious employee must be created just to save the project information.

In spite of those structural deficiencies, the table structure *appears* to work; the report is generated with ease. Unfortunately, the report might yield varying results depending on what data anomaly has occurred. For example, if you want to print a report to show the total "hours worked" value by the job classification "Database Designer," that report will not include data for "DB Design" and "Database Design" data entries. Such reporting anomalies cause a multitude of problems for managers—and cannot be fixed through applications programming.

Even if very careful data entry auditing can eliminate most of the reporting problems (at a high cost), it is easy to demonstrate that even a simple data entry becomes inefficient. Given the existence of update anomalies, suppose Darlene M. Smithson is assigned to work on the Evergreen project. The data entry clerk must update the PROJECT file with the entry:

15 Evergreen 112 Darlene M. Smithson DSS Analyst $45.95 0.0

to match the attributes PROJ_NUM, PROJ_NAME, EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS. (When Ms. Smithson has just been assigned to the project, she has not yet worked, so the total number of hours worked is 0.0.)

NOTE

Remember that the naming convention makes it easy to see what each attribute stands for and what its likely origin is. For example, PROJ_NAME uses the prefix PROJ to indicate that the attribute is associated with the PROJECT table, while the NAME component is self-documenting, too. However, keep in mind that name length is also an issue, especially in the prefix designation. For that reason, the prefix CHG was used rather than CHARGE. (Given the database's context, it is not likely that that prefix will be misunderstood.)

Each time another employee is assigned to a project, some data entries (such as PROJ_NAME, EMP_NAME, and CHG_HOUR) are unnecessarily repeated. Imagine the data entry chore when 200 or 300 table entries must be made! Note that the entry of the employee number should be sufficient to identify Darlene M. Smithson, her job description, and her hourly charge. Because there is only one person identified by the number 112, that person's characteristics (name, job classification, and so on) should not have to be typed in each time the main file is updated. Unfortunately, the structure displayed in Figure 5.1 does not make allowances for that possibility.

The data redundancy evident in Figure 5.1 leads to wasted disk space. What's more, data redundancy produces data anomalies. For example, suppose the data entry clerk had entered the data as:

15 Evergeen 112 Darla Smithson DCS Analyst $45.95 0.0

At first glance, the data entry appears to be correct. But is Evergeen the same project as Evergreen? And is DCS Analyst supposed to be DSS Analyst? Is Darla Smithson the same person as Darlene M. Smithson? Such confusion is a data integrity problem that was caused because the data entry failed to conform to the rule that all copies of redundant data must be identical.

The possibility of introducing data integrity problems caused by data redundancy must be considered when a database is designed. The relational database environment is especially well suited to help the designer overcome those problems.

5.3 THE NORMALIZATION PROCESS

In this section, you learn how to use normalization to produce a set of normalized tables to store the data that will be used to generate the required information. The objective of normalization is to ensure that each table conforms to the concept of well-formed relations, that is, tables that have the following characteristics:

- Each table represents a single subject. For example, a course table will contain only data that directly pertains to courses. Similarly, a student table will contain only student data.
- No data item will be *unnecessarily* stored in more than one table (in short, tables have minimum controlled redundancy). The reason for this requirement is to ensure that the data are updated in only one place.
- All nonprime attributes in a table are dependent on the primary key—the entire primary key and nothing but the primary key. The reason for this requirement is to ensure that the data are uniquely identifiable by a primary key value.
- Each table is void of insertion, update, or deletion anomalies. This is to ensure the integrity and consistency of the data.

To accomplish the objective, the normalization process takes you through the steps that lead to successively higher normal forms. The most common normal forms and their basic characteristic are listed in Table 5.2. You will learn the details of these normal forms in the indicated sections.

TABLE 5.2 Normal Forms

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	5.3.1
Second normal form (2NF)	1NF and no partial dependencies	5.3.2
Third normal form (3NF)	2NF and no transitive dependencies	5.3.3
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	5.6.1
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	5.6.2

From the data modeler's point of view, the objective of normalization is to ensure that all tables are at least in third normal form (3NF). Even higher-level normal forms exist. However, normal forms such as the fifth normal form (5NF) and domain-key normal form (DKNF) are not likely to be encountered in a business environment and are mainly of theoretical interest. More often than not, such higher normal forms usually increase joins (slowing performance) without adding any value in the elimination of data redundancy. Some very specialized applications, such as statistical research, might require normalization beyond the 4NF, but those applications fall outside the scope of most business operations. Because this book focuses on practical applications of database techniques, the higher-level normal forms are not covered.

Functional Dependency

Before outlining the normalization process, it's a good idea to review the concepts of determination and functional dependency that were covered in detail in Chapter 3. Table 5.3 summarizes the main concepts.

TABLE 5.3 Functional Dependency Concepts

CONCEPT	DEFINITION
Functional dependency	The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B*. Example: PROJ_NUM → PROJ_NAME (read as "PROJ_NUM functionally determines PROJ_NAME) In this case, the attribute PROJ_NUM is known as the "determinant" attribute and the attribute PROJ_NAME is known as the "dependent" attribute.
Functional dependency (generalized definition)	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*.
Fully functional dependency (composite key)	If 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*.

It is crucial to understand these concepts because they are used to derive the set of functional dependencies for a given relation. The normalization process works one relation at a time, identifying the dependencies on that relation and normalizing the relation. As you will see in the following sections, normalization starts by identifying the dependencies of a given relation and progressively breaking up the relation (table) into a set of new relations (tables) based on the identified dependencies.

5.3.1 Conversion to First Normal Form

Because the relational model views data as part of a table or a collection of tables in which all key values must be identified, the data depicted in Figure 5.1 might not be stored as shown. Note that Figure 5.1 contains what is known as repeating groups. A **repeating group** derives its name from the fact that a group of multiple entries of the same type can exist for any *single* key attribute occurrence. In Figure 5.1, note that each single project number (PROJ_NUM) occurrence can reference a group of related data entries. For example, the Evergreen project (PROJ_NUM = 15) shows five entries at this point—and those entries are related because they each share the PROJ_NUM = 15 characteristic. Each time a new record is entered for the Evergreen project, the number of entries in the group grows by one.

A relational table must not contain repeating groups. The existence of repeating groups provides evidence that the RPT_FORMAT table in Figure 5.1 fails to meet even the lowest normal form requirements, thus reflecting data redundancies.

Normalizing the table structure will reduce the data redundancies. If repeating groups do exist, they must be eliminated by making sure that each row defines a single entity. In addition, the dependencies must be identified to diagnose the normal form. Identification of the normal form will let you know where you are in the normalization process. The normalization process starts with a simple three-step procedure.

Step 1: Eliminate the Repeating Groups

Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value. That change converts the table in Figure 5.1 to 1NF in Figure 5.2.

FIGURE 5.2 A table in first normal form

Table name: DATA_ORG_1NF **Database name: Ch05_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Step 2: Identify the Primary Key

The layout in Figure 5.2 represents more than a mere cosmetic change. Even a casual observer will note that PROJ_NUM is not an adequate primary key because the project number does not uniquely identify all of the remaining entity (row) attributes. For example, the PROJ_NUM value 15 can identify any one of five employees. To maintain a proper primary key that will *uniquely* identify any attribute value, the new key must be composed of a *combination* of PROJ_NUM and EMP_NUM. For example, using the data shown in Figure 5.2, if you know that PROJ_NUM = 15 and EMP_NUM = 103, the entries for the attributes PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS must be Evergreen, June E. Arbough, Elect. Engineer, $84.50, and 23.8, respectively.

Step 3: Identify All Dependencies

The identification of the PK in Step 2 means that you have already identified the following dependency:

PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS

That is, the PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS values are all dependent on—that is, they are determined by—the combination of PROJ_NUM and EMP_NUM. There are additional dependencies. For example, the project number identifies (determines) the project name. In other words, the project name is dependent on the project number. You can write that dependency as:

PROJ_NUM → PROJ_NAME

Also, if you know an employee number, you also know that employee's name, that employee's job classification, and that employee's charge per hour. Therefore, you can identify the dependency shown next:

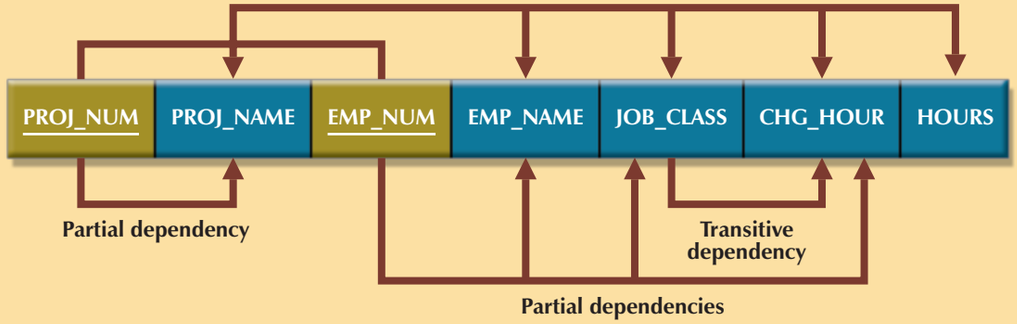
EMP_NUM → EMP_NAME, JOB_CLASS, CHG_HOUR

However, given the previous dependency components, you can see that knowing the job classification means knowing the charge per hour for that job classification. In other words, you can identify one last dependency:

JOB_CLASS → CHG_HOUR

The dependencies you have just examined can also be depicted with the help of the diagram shown in Figure 5.3. Because such a diagram depicts all dependencies found within a given table structure, it is known as a **dependency diagram**. Dependency diagrams are very helpful in getting a bird's-eye view of all of the relationships among a table's attributes, and their use makes it less likely that you will overlook an important dependency.

FIGURE 5.3 First normal form (1NF) dependency diagram



1NF (PROJ_NUM, EMP_NUM, PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOURS, HOURS)

PARTIAL DEPENDENCIES:
(PROJ_NUM → PROJ_NAME)
(EMP_NUM → EMP_NAME, JOB_CLASS, CHG_HOUR)

TRANSITIVE DEPENDENCY:
(JOB CLASS → CHG_HOUR)

As you examine Figure 5.3, note the following dependency diagram features:

1. The primary key attributes are bold, underlined, and shaded in a different color.
2. The arrows above the attributes indicate all desirable dependencies, that is, dependencies that are based on the primary key. In this case, note that the entity's attributes are dependent on the *combination* of PROJ_NUM and EMP_NUM.
3. The arrows below the dependency diagram indicate less desirable dependencies. Two types of such dependencies exist:

   a. *Partial dependencies*. You need to know only the PROJ_NUM to determine the PROJ_NAME; that is, the PROJ_NAME is dependent on only part of the primary key. And you need to know only the EMP_NUM to find the EMP_NAME, the JOB_CLASS, and the CHG_HOUR. A dependency based on only a part of a composite primary key is called a **partial dependency**.

   b. *Transitive dependencies*. Note that CHG_HOUR is dependent on JOB_CLASS. Because neither CHG_HOUR nor JOB_CLASS is a prime attribute—that is, neither attribute is at least part of a key—the condition is known as a transitive dependency. In other words, a **transitive dependency** is a dependency of one nonprime attribute on another nonprime attribute. The problem with transitive dependencies is that they still yield data anomalies.

Note that Figure 5.3 includes the relational schema for the table in 1NF and a textual notation for each identified dependency.

NOTE

The term **first normal form** (**1NF**) describes the tabular format in which:

- All of the key attributes are defined.
- There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
- All attributes are dependent on the primary key.

All relational tables satisfy the 1NF requirements. The problem with the 1NF table structure shown in Figure 5.3 is that it contains partial dependencies—that is, dependencies based on only a part of the primary key.

While partial dependencies are sometimes used for performance reasons, they should be used with caution. (If the information requirements seem to dictate the use of partial dependencies, it is time to evaluate the need for a data warehouse design, discussed in Chapter 13, Business Intelligence and Data Warehouses.) Such caution is warranted because a table that contains partial dependencies is still subject to data redundancies, and therefore, to various anomalies. The data redundancies occur because every row entry requires duplication of data. For example, if Alice K. Johnson submits her work log, then the user would have to make multiple entries during the course of a day. For each entry, the EMP_NAME, JOB_CLASS, and CHG_HOUR must be entered each time even though the attribute values are identical for each row entered. Such duplication of effort is very inefficient. What's more, the duplication of effort helps create data anomalies; nothing prevents the user from typing slightly different versions of the employee name, the position, or the hourly pay. For instance, the employee name for EMP_NUM = 102 might be entered as Dave Senior or D. Senior. The project name also might be entered correctly as Evergreen or misspelled as Evergeen. Such data anomalies violate the relational database's integrity and consistency rules.

5.3.2 Conversion to Second Normal Form

Converting to 2NF is done only when the 1NF has a composite primary key. If the 1NF has a single attribute primary key, then the table is automatically in 2NF. The 1NF-to-2NF conversion is simple. Starting with the 1NF format displayed in Figure 5.3, you do the following:

Step 1: Write Each Key Component on a Separate Line

Write each key component on a separate line; then write the original (composite) key on the last line. For example:

PROJ_NUM

EMP_NUM

PROJ_NUM EMP_NUM

Each component will become the key in a new table. In other words, the original table is now divided into three tables (PROJECT, EMPLOYEE, and ASSIGNMENT).

Step 2: Assign Corresponding Dependent Attributes

Use Figure 5.3 to determine those attributes that are dependent on other attributes. The dependencies for the original key components are found by examining the arrows below the dependency diagram shown in Figure 5.3. In other words, the three new tables (PROJECT, EMPLOYEE, and ASSIGNMENT) are described by the following relational schemas:

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

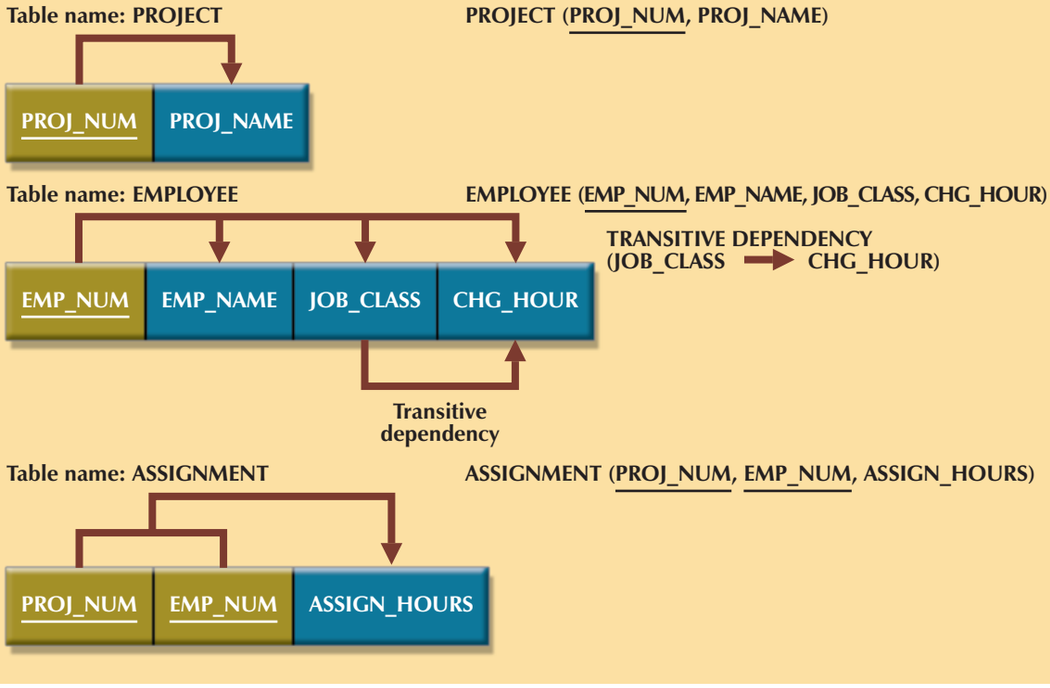
Because the number of hours spent on each project by each employee is dependent on both PROJ_NUM and EMP_NUM in the ASSIGNMENT table, you place those hours in the ASSIGNMENT table as ASSIGN_HOURS.

NOTE

The ASSIGNMENT table contains a composite primary key composed of the attributes PROJ_NUM and EMP_NUM. Any attribute that is at least part of a key is known as a **prime attribute** or a **key attribute**. Therefore, both PROJ_NUM and EMP_NUM are prime (or key) attributes. Conversely, a **nonprime attribute**, or a **nonkey attribute**, is not part of any key.

The results of Steps 1 and 2 are displayed in Figure 5.4. At this point, most of the anomalies discussed earlier have been eliminated. For example, if you now want to add, change, or delete a PROJECT record, you need to go only to the PROJECT table and make the change to only one row.

FIGURE 5.4 **Second normal form (2NF) conversion results**



Because a partial dependency can exist only when a table's primary key is composed of several attributes, a table whose primary key consists of only a single attribute is automatically in 2NF once it is in 1NF.

Figure 5.4 still shows a transitive dependency, which can generate anomalies. For example, if the charge per hour changes for a job classification held by many employees, that change must be made for *each* of those employees. If you forget to update some of the employee records that are affected by the charge per hour change, different employees with the same job description will generate different hourly charges.

NOTE

A table is in **second normal form** (**2NF**) when:

- It is in 1NF.

and

- It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.

Note that it is still possible for a table in 2NF to exhibit transitive dependency; that is, one or more attributes may be functionally dependent on nonkey attributes.

5.3.3 Conversion to Third Normal Form

The data anomalies created by the database organization shown in Figure 5.4 are easily eliminated by completing the following three steps:

Step 1: Identify Each New Determinant

For every transitive dependency, write its determinant as a PK for a new table. A **determinant** is any attribute whose value determines other values within a row. If you have three different transitive dependencies, you will have three different determinants. Figure 5.4 shows only one table that contains a transitive dependency. Therefore, write the determinant for this transitive dependency as:

JOB_CLASS

Step 2: Identify the Dependent Attributes

Identify the attributes that are dependent on each determinant identified in Step 1 and identify the dependency. In this case, you write:

JOB_CLASS → CHG_HOUR

Name the table to reflect its contents and function. In this case, JOB seems appropriate.

Step 3: Remove the Dependent Attributes from Transitive Dependencies

Eliminate all dependent attributes in the transitive relationship(s) from each of the tables that have such a transitive relationship. In this example, eliminate CHG_HOUR from the EMPLOYEE table shown in Figure 5.4 to leave the EMPLOYEE table dependency definition as:

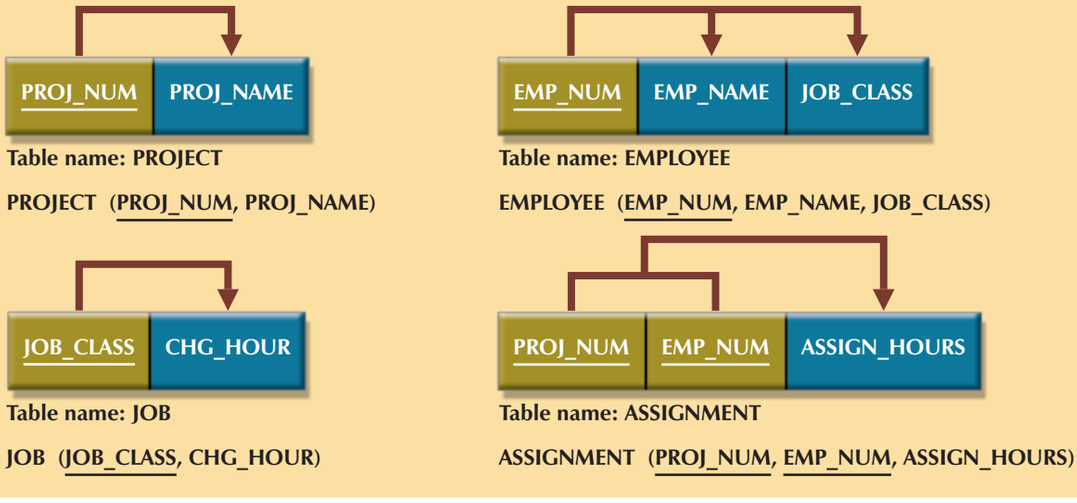
EMP_NUM → EMP_NAME, JOB_CLASS

Note that the JOB_CLASS remains in the EMPLOYEE table to serve as the FK.

Draw a new dependency diagram to show all of the tables you have defined in Steps 1–3. Check the new tables as well as the tables you modified in Step 3 to make sure that each table has a determinant and that no table contains inappropriate dependencies.

When you have completed Steps 1–3, you will see the results in Figure 5.5. (The usual procedure is to complete Steps 1–3 by simply drawing the revisions as you make them.)

FIGURE 5.5 Third normal form (3NF) conversion results



In other words, after the 3NF conversion has been completed, your database contains four tables:

PROJECT (**PROJ_NUM**, PROJ_NAME)

EMPLOYEE (**EMP_NUM**, EMP_NAME, JOB_CLASS)

JOB (**JOB_CLASS**, CHG_HOUR)

ASSIGNMENT (**PROJ_NUM**, **EMP_NUM**, ASSIGN_HOURS)

Note that this conversion has eliminated the original EMPLOYEE table's transitive dependency; the tables are now said to be in third normal form (3NF).

NOTE

A table is in **third normal form** (**3NF**) when:

- It is in 2NF.

and

- It contains no transitive dependencies.

5.4 IMPROVING THE DESIGN

The table structures are cleaned up to eliminate the troublesome partial and transitive dependencies. You can now focus on improving the database's ability to provide information and on enhancing its operational characteristics. In the next few paragraphs, you will learn about the various types of issues you need to address to produce a good normalized set of tables. Please note that for space issues, each section presents just one example—the designer must apply the principle to all remaining tables in the design. Remember that normalization cannot, by itself, be relied on to make good designs. Instead, normalization is valuable because its use helps eliminate data redundancies.

Evaluate PK Assignments

Each time a new employee is entered into the EMPLOYEE table, a JOB_CLASS value must be entered. Unfortunately, it is too easy to make data-entry errors that lead to referential integrity violations. For example, entering DB Designer instead of Database Designer for the JOB_CLASS attribute in the EMPLOYEE table will trigger such a violation. Therefore, it would be better to add a JOB_CODE attribute to create a unique identifier. The addition of a JOB_CODE attribute produces the dependency:

JOB_CODE → JOB_CLASS, CHG_HOUR

If you assume that the JOB_CODE is a proper primary key, this new attribute does produce the transitive dependency:

JOB_CLASS → CHG_HOUR

A transitive dependency exists because a nonkey attribute—the JOB_CLASS—determines the value of another nonkey attribute—the CHG_HOUR. However, that transitive dependency is an easy price to pay; the presence of JOB_CODE greatly decreases the likelihood of referential integrity violations. Note that the new JOB table now has two candidate keys—JOB_CODE and JOB_CLASS. In this case, JOB_CODE is the chosen primary key as well as a surrogate key. A **surrogate key** is an artificial PK introduced by the designer with the purpose of simplifying the assignment of primary keys to tables. Surrogate keys are usually numeric, they are often automatically generated by the DBMS, they are free of semantic content (they have no special meaning), and they are usually hidden from the end users. You will learn more about PK characteristics and assignment in Chapter 6, Advanced Data Modeling.

Evaluate Naming Conventions

It is best to adhere to the naming conventions outlined in Chapter 2, Data Models. Therefore, CHG_HOUR will be changed to JOB_CHG_HOUR to indicate its association with the JOB table. In addition, the attribute name JOB_CLASS does not quite describe entries such as Systems Analyst, Database Designer, and so on; the label JOB_DESCRIPTION fits the entries better. Also, you might have noticed that HOURS was changed to ASSIGN_HOURS in the conversion from 1NF to 2NF. That change lets you associate the hours worked with the ASSIGNMENT table.

Refine Attribute Atomicity

It generally is good practice to pay attention to the *atomicity* requirement. An **atomic attribute** is one that cannot be further subdivided. Such an attribute is said to display **atomicity**. Clearly, the use of the EMP_NAME in the EMPLOYEE table is not atomic because EMP_NAME can be decomposed into a last name, a first name, and an initial. By improving the degree of atomicity, you also gain querying flexibility. For example, if you use EMP_LNAME, EMP_FNAME, and EMP_INITIAL, you can easily generate phone lists by sorting last names, first names, and initials. Such a task would be very difficult if the name components were within a single attribute. In general, designers prefer to use simple, single-valued attributes as indicated by the business rules and processing requirements.

Identify New Attributes

If the EMPLOYEE table were used in a real-world environment, several other attributes would have to be added. For example, year-to-date gross salary payments, Social Security payments, and Medicare payments would be desirable. Adding an employee hire date attribute (EMP_HIREDATE) could be used to track an employee's job longevity and serve as a basis for awarding bonuses to long-term employees and for other morale-enhancing measures. The same principle must be applied to all other tables in your design.

Identify New Relationships

The system's ability to supply detailed information about each project's manager is ensured by using the EMP_NUM as a foreign key in PROJECT. That action ensures that you can access the details of each PROJECT's manager data without producing unnecessary and undesirable data duplication. The designer must take care to place the right attributes in the right tables by using normalization principles.

Refine Primary Keys as Required for Data Granularity

Granularity refers to the level of detail represented by the values stored in a table's row. Data stored at their lowest level of granularity are said to be *atomic data, as explained earlier*. In Figure 5.5, the ASSIGNMENT table in 3NF uses the ASSIGN_HOURS attribute to represent the hours worked by a given employee on a given project. However, are those values recorded at their lowest level of granularity? In other words, do the ASSIGN_HOURS represent the *hourly* total, *daily* total, *weekly* total, *monthly* total, or *yearly* total? Clearly, ASSIGN_HOURS requires more careful definition. In this case, the relevant question would be as follows: For what time frame—hour, day, week, month, and so on—do you want to record the ASSIGN_HOURS data?

For example, assume that the combination of EMP_NUM and PROJ_NUM is an acceptable (composite) primary key in the ASSIGNMENT table. That primary key is useful in representing only the total number of hours an employee worked on a project since its start. Using a surrogate primary key such as ASSIGN_NUM provides lower granularity and yields greater flexibility. For example, assume that the EMP_NUM and PROJ_NUM combination is used as the primary key, and then an employee makes two "hours worked" entries in the ASSIGNMENT table. That action violates the entity integrity requirement. Even if you add the ASSIGN_DATE as part of a composite PK, an entity integrity violation is still generated if any employee makes two or more entries for the same project on the same day. (The employee might have worked on the project a few hours in the morning and then worked on it again later in the day.) The same data entry yields no problems when ASSIGN_NUM is used as the primary key.

NOTE

In an ideal (database design) world, the level of desired granularity is determined at the conceptual design or at the requirements gathering phase. However, as you have already seen in this chapter, many database designs involve the refinement of existing data requirements, thus triggering design modifications. In a real-world environment, changing granularity requirements might dictate changes in primary key selection, and those changes might ultimately require the use of surrogate keys.

Maintain Historical Accuracy

Writing the job charge per hour into the ASSIGNMENT table is crucial to maintaining the historical accuracy of the data in the ASSIGNMENT table. It would be appropriate to name this attribute ASSIGN_CHG_HOUR. Although this attribute would appear to have the same value as JOB_CHG_HOUR, that is true *only* if the JOB_CHG_HOUR value remains forever the same. However, it is reasonable to assume that the job charge per hour will change over time. But suppose that the charges to each project were figured (and billed) by multiplying the hours worked on the project, found in the ASSIGNMENT table, by the charge per hour, found in the JOB table. Those charges would always show the current charge per hour stored in the JOB table, rather than the charge per hour that was in effect at the time of the assignment.

Evaluate Using Derived Attributes

Finally, you can use a derived attribute in the ASSIGNMENT table to store the actual charge made to a project. That derived attribute, to be named ASSIGN_CHARGE, is the result of multiplying the ASSIGN_HOURS by the ASSIGN_CHG_HOUR. From a strictly database point of view, such derived attribute values can be calculated when they are needed to write reports or invoices. However, storing the derived attribute in the table makes it easy to write the application software to produce the desired results. Also, if many transactions must be reported and/or summarized, the availability of the derived attribute will save reporting time. (If the calculation is done at the time of data entry, it will be completed when the end user presses the Enter key, thus speeding up the process.)

The enhancements described in the preceding sections are illustrated in the tables and dependency diagrams shown in Figure 5.6.

FIGURE 5.6 The completed database

Database name: Ch05_ConstructCo

Table name: PROJECT

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Table name: ASSIGNMENT

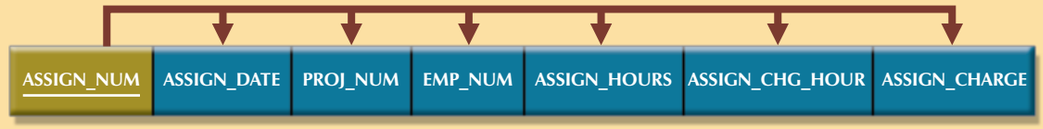


Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-08	15	103	2.6	84.50	219.70
1002	04-Mar-08	18	118	1.4	18.36	25.70
1003	05-Mar-08	15	101	3.6	105.00	378.00
1004	05-Mar-08	22	113	2.5	48.10	120.25
1005	05-Mar-08	15	103	1.9	84.50	160.55
1006	05-Mar-08	25	115	4.2	96.75	406.35
1007	05-Mar-08	22	105	5.2	105.00	546.00
1008	05-Mar-08	25	101	1.7	105.00	178.50
1009	05-Mar-08	15	105	2.0	105.00	210.00
1010	06-Mar-08	15	102	3.8	96.75	367.65
1011	06-Mar-08	22	104	2.6	96.75	251.55
1012	06-Mar-08	15	101	2.3	105.00	241.50
1013	06-Mar-08	25	114	1.8	48.10	86.58
1014	06-Mar-08	22	111	4.0	26.87	107.48
1015	06-Mar-08	25	114	3.4	48.10	163.54
1016	06-Mar-08	18	112	1.2	45.95	55.14
1017	06-Mar-08	18	118	2.0	18.36	36.72
1018	06-Mar-08	18	104	2.6	96.75	251.55
1019	06-Mar-08	15	103	3.0	84.50	253.50
1020	07-Mar-08	22	105	2.7	105.00	283.50
1021	08-Mar-08	25	108	4.2	96.75	406.35
1022	07-Mar-08	25	114	5.8	48.10	278.98
1023	07-Mar-08	22	106	2.4	35.75	85.80

FIGURE 5.6 The completed database (continued)

Table name: EMPLOYEE

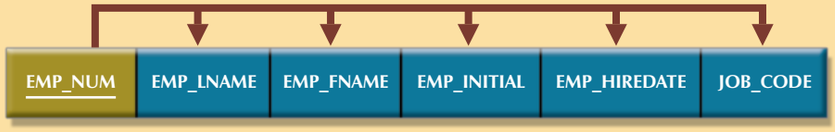


Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Figure 5.6 is a vast improvement over the original database design. If the application software is designed properly, the most active table (ASSIGNMENT) requires the entry of only the PROJ_NUM, EMP_NUM, and ASSIGN_HOURS values. The values for the attributes ASSIGN_NUM and ASSIGN_DATE can be generated by the application. For example, the ASSIGN_NUM can be created by using a counter, and the ASSIGN_DATE can be the system date read by the application and automatically entered into the ASSIGNMENT table. In addition, the application software can automatically insert the correct ASSIGN_CHG_HOUR value by writing the appropriate JOB table's JOB_CHG_HOUR value into the ASSIGNMENT table. (The JOB and ASSIGNMENT tables are related through the JOB_CODE attribute.) If the JOB table's JOB_CHG_HOUR value changes, the next insertion of that value into the ASSIGNMENT table will reflect the change automatically. The table structure thus minimizes the need for human intervention. In fact, if the system requires the employees to enter their own work hours, they can scan their EMP_NUM into the ASSIGNMENT table by using a magnetic card reader that enters their identity. Thus, the ASSIGNMENT table's structure can set the stage for maintaining some desired level of security.

5.5 SURROGATE KEY CONSIDERATIONS

Although this design meets the vital entity and referential integrity requirements, the designer still must address some concerns. For example, a composite primary key might become too cumbersome to use as the number of attributes grows. (It becomes difficult to create a suitable foreign key when the related table uses a composite primary key. In addition, a composite primary key makes it more difficult to write search routines.) Or a primary key attribute might simply have too much descriptive content to be usable—which is why the JOB_CODE attribute was added to the JOB

table to serve as that table's primary key. When, for whatever reason, the primary key is considered to be unsuitable, designers use surrogate keys.

At the implementation level, a surrogate key is a system-defined attribute generally created and managed via the DBMS. Usually, a system-defined surrogate key is numeric, and its value is automatically incremented for each new row. For example, Microsoft Access uses an AutoNumber data type, Microsoft SQL Server uses an identity column, and Oracle uses a sequence object.

Recall from Section 5.4 that the JOB_CODE attribute was designated to be the JOB table's primary key. However, remember that the JOB_CODE does not prevent duplicate entries from being made, as shown in the JOB table in Table 5.4.

TABLE 5.4 Duplicate Entries in the Job Table

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
511	Programmer	$35.75
512	Programmer	$35.75

Clearly, the data entries in Table 5.4 are inappropriate because they duplicate existing records—yet there has been no violation of either entity integrity or referential integrity. This "multiple duplicate records" problem was created when the JOB_CODE attribute was added as the PK. (When the JOB_DESCRIPTION was initially designated to be the PK, the DBMS would ensure unique values for all job description entries when it was asked to enforce entity integrity. But that option created the problems that caused use of the JOB_CODE attribute in the first place!) In any case, if JOB_CODE is to be the surrogate PK, you still must ensure the existence of unique values in the JOB_DESCRIPTION *through the use of a unique index*.

Note that all of the remaining tables (PROJECT, ASSIGNMENT, and EMPLOYEE) are subject to the same limitations. For example, if you use the EMP_NUM attribute in the EMPLOYEE table as the PK, you can make multiple entries for the same employee. To avoid that problem, you might create a unique index for EMP_LNAME, EMP_FNAME, and EMP_INITIAL. But how would you then deal with two employees named Joe B. Smith? In that case, you might use another (preferably externally defined) attribute to serve as the basis for a unique index.

It is worth repeating that database design often involves trade-offs and the exercise of professional judgment. In a real-world environment, you must strike a balance between design integrity and flexibility. For example, you might design the ASSIGNMENT table to use a unique index on PROJ_NUM, EMP_NUM, and ASSIGN_DATE if you want to limit an employee to only one ASSIGN_HOURS entry per date. That limitation would ensure that employees couldn't enter the same hours multiple times for any given date. Unfortunately, that limitation is likely to be undesirable from a managerial point of view. After all, if an employee works several different times on a project during any given day, it must be possible to make multiple entries for that same employee and the same project during that day. In that case, the best solution might be to add a new externally defined attribute—such as a stub, voucher, or ticket number—to ensure uniqueness. In any case, frequent data audits would be appropriate.

5.6 HIGHER-LEVEL NORMAL FORMS

Tables in 3NF will perform suitably in business transactional databases. However, there are occasions when higher normal forms are useful. In this section, you learn about a special case of 3NF, known as Boyce-Codd normal form (BCNF), and about fourth normal form (4NF).

5.6.1 THE BOYCE-CODD NORMAL FORM (BCNF)

A table is in **Boyce-Codd normal form** (**BCNF**) when every determinant in the table is a candidate key. (Recall from Chapter 3 that a candidate key has the same characteristics as a primary key, but for some reason, it was not chosen to be the primary key.) Clearly, when a table contains only one candidate key, the 3NF and the BCNF are equivalent. Putting that proposition another way, BCNF can be violated only when the table contains more than one candidate key.

NOTE

A table is in BCNF when every determinant in the table is a candidate key.

Most designers consider the BCNF to be a special case of the 3NF. In fact, if the techniques shown here are used, most tables conform to the BCNF requirements once the 3NF is reached. So how can a table be in 3NF and not be in BCNF? To answer that question, you must keep in mind that a transitive dependency exists when one nonprime attribute is dependent on another nonprime attribute.

In other words, a table is in 3NF when it is in 2NF and there are no transitive dependencies. But what about a case in which a nonkey attribute is the determinant of a key attribute? That condition does not violate 3NF, yet it fails to meet the BCNF requirements because BCNF requires that every determinant in the table be a candidate key.

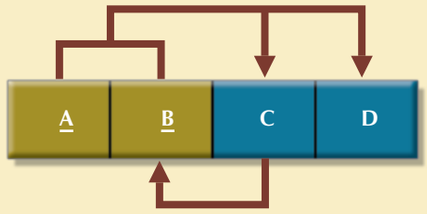
The situation just described (a 3NF table that fails to meet BCNF requirements) is shown in Figure 5.7.

Note these functional dependencies in Figure 5.7:

$A + B \rightarrow C, D$

$C \rightarrow B$

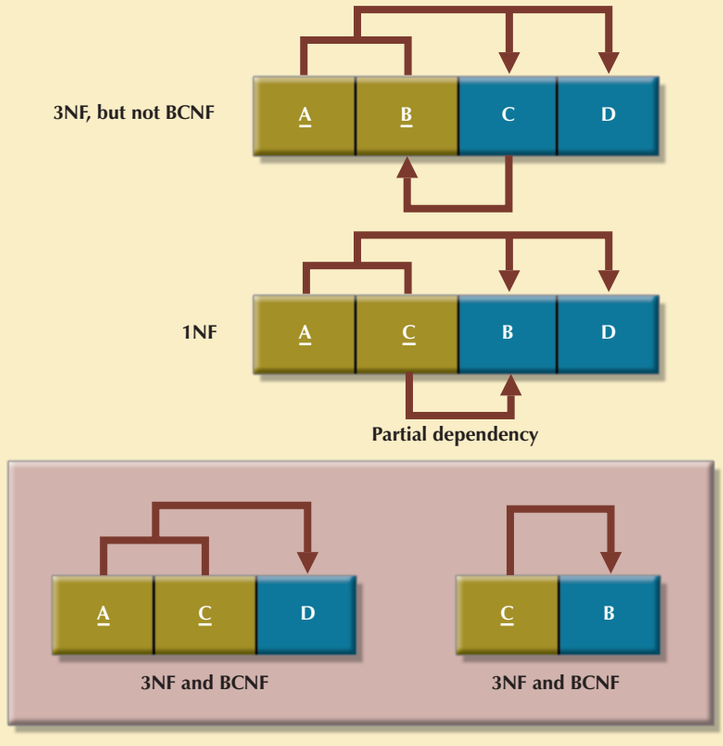
FIGURE 5.7 A table that is in 3NF but not in BCNF



The table structure shown in Figure 5.7 has no partial dependencies, nor does it contain transitive dependencies. (The condition $C \rightarrow B$ indicates that *a nonkey attribute determines part of the primary key*—and *that* dependency is *not* transitive!) Thus, the table structure in Figure 5.7 meets the 3NF requirements. Yet the condition $C \rightarrow B$ causes the table to fail to meet the BCNF requirements.

To convert the table structure in Figure 5.7 into table structures that are in 3NF and in BCNF, first change the primary key to A + C. That is an appropriate action because the dependency $C \rightarrow B$ means that C is, in effect, a superset of B. At this point, the table is in 1NF because it contains a partial dependency $C \rightarrow B$. Next, follow the standard decomposition procedures to produce the results shown in Figure 5.8.

FIGURE 5.8 Decomposition to BCNF



To see how this procedure can be applied to an actual problem, examine the sample data in Table 5.5.

TABLE 5.5 Sample Data for a BCNF Conversion

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

Table 5.5 reflects the following conditions:

- Each CLASS_CODE identifies a class uniquely. This condition illustrates the case in which a course might generate many classes. For example, a course labeled INFS 420 might be taught in two classes (sections), each identified by a unique code to facilitate registration. Thus, the CLASS_CODE 32456 might identify INFS 420, class section 1, while the CLASS_CODE 32457 might identify INFS 420, class section 2. Or the CLASS_CODE 28458 might identify QM 362, class section 5.

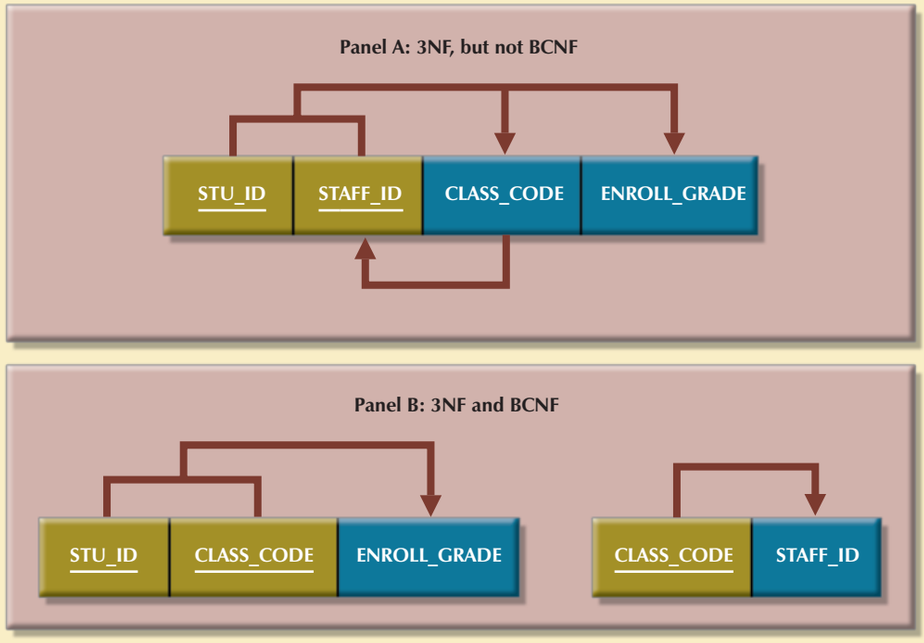
- A student can take many classes. Note, for example, that student 125 has taken both 21334 and 32456, earning the grades A and C, respectively.
- A staff member can teach many classes, but each class is taught by only one staff member. Note that staff member 20 teaches the classes identified as 32456 and 28458.

The structure shown in Table 5.5 is reflected in Panel A of Figure 5.9:

STU_ID + STAFF_ID → CLASS_CODE, ENROLL_GRADE

CLASS_CODE → STAFF_ID

FIGURE 5.9 **Another BNCF decomposition**



Panel A of Figure 5.9 shows a structure that is clearly in 3NF, but the table represented by this structure has a major problem, because it is trying to describe two things: staff assignments to classes and student enrollment information. Such a dual-purpose table structure will cause anomalies. For example, if a different staff member is assigned to teach class 32456, two rows will require updates, thus producing an update anomaly. And if student 135 drops class 28458, information about who taught that class is lost, thus producing a deletion anomaly. The solution to the problem is to decompose the table structure, following the procedure outlined earlier. Note that the decomposition of Panel B shown in Figure 5.9 yields two table structures that conform to both 3NF and BCNF requirements.

Remember that a table is in BCNF when every determinant in that table is a candidate key. Therefore, when a table contains only one candidate key, 3NF and BCNF are equivalent.

5.6.2 Fourth Normal Form (4NF)

You might encounter poorly designed databases, or you might be asked to convert spreadsheets into a database format in which multiple multivalued attributes exist. For example, consider the possibility that an employee can have multiple assignments and can also be involved in multiple service organizations. Suppose employee 10123 does volunteer work for the Red Cross and United Way. In addition, the same employee might be assigned to work on three projects: 1, 3, and 4. Figure 5.10 illustrates how that set of facts can be recorded in very different ways.

FIGURE 5.10 Tables with multivalued dependencies

Database name: Ch05_Service

Table name: VOLUNTEER_V1

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	UW	3
10123		4

Table name: VOLUNTEER_V2

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	
10123	UW	
10123		1
10123		3
10223		4

Table name: VOLUNTEER_V3

EMP_NUM	ORG_CODE	ASSIGN_NUM
10123	RC	1
10123	RC	3
10123	UW	4

There is a problem with the tables in Figure 5.10. The attributes ORG_CODE and ASSIGN_NUM each may have many different values. That is, the tables contain two sets of independent multivalued dependencies. (One employee can have many service entries and many assignment entries.) The presence of multiple sets of independent multivalued dependencies means that if versions 1 and 2 are implemented, the tables are likely to contain quite a few null values; in fact, the tables do not even have a viable candidate key. (The EMP_NUM values are not unique, so they cannot be PKs. No combination of the attributes in table versions 1 and 2 can be used to create a PK because some of them contain nulls.) Such a condition is not desirable, especially when there are thousands of employees, many of whom may have multiple job assignments and many service activities. Version 3 at least has a PK, but it is composed of all of the attributes in the table. In fact, version 3 meets 3NF requirements, yet it contains many redundancies that are clearly undesirable.

The solution is to eliminate the problems caused by independent multivalued dependencies. You do this by creating the ASSIGNMENT and SERVICE_V1 tables depicted in Figure 5.11. Note that in Figure 5.11, neither the ASSIGNMENT nor the SERVICE_V1 table contains independent multivalued dependencies. Those tables are said to be in 4NF.

If you follow the proper design procedures illustrated in this book, you shouldn't encounter the previously described problem. Specifically, the discussion of 4NF is largely academic if you make sure that your tables conform to the following two rules:

1. All attributes must be dependent on the primary key, but they must be independent of each other.
2. No row may contain two or more multivalued facts about an entity.

FIGURE 5.11 A set of tables in 4NF

Table name: EMPLOYEE

Database name: Ch05_Service

EMP_NUM	EMP_LNAME
10121	Rogers
10122	O'Leery
10123	Panera
10124	Johnson

Table name: PROJECT

PROJ_CODE	PROJ_NAME	PROJ_BUDGET
1	BeThere	1023245.00
2	BlueMoon	20198608.00
3	GreenThumb	3234456.00
4	GoFast	5674000.00
5	GoSlow	1002500.00

Table name: ORGANIZATION

ORG_CODE	ORG_NAME
RC	Red Cross
UW	United Way
WF	Wildlife Fund

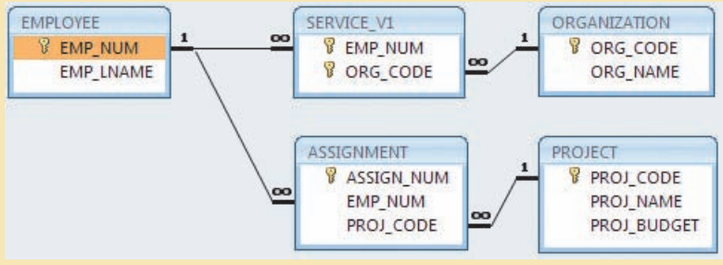
Table name: ASSIGNMENT

ASSIGN_NUM	EMP_NUM	PROJ_CODE
1	10123	1
2	10121	2
3	10123	3
4	10123	4
5	10121	1
6	10124	2
7	10124	3
8	10124	5

Table name: SERVICE_V1

EMP_NUM	ORG_CODE
10123	RC
10123	UW
10123	WF

The relational diagram



NOTE

A table is in **fourth normal form** (**4NF**) when it is in 3NF and has no multiple sets of multivalued dependencies.

5.7 NORMALIZATION AND DATABASE DESIGN

The tables shown in Figure 5.6 illustrate how normalization procedures can be used to produce good tables from poor ones. You will likely have ample opportunity to put this skill into practice when you begin to work with real-world databases. *Normalization should be part of the design process*. Therefore, make sure that proposed entities meet the required normal form *before* the table structures are created. Keep in mind that if you follow the design procedures discussed in Chapter 3 and Chapter 4 the likelihood of data anomalies will be small. But even the best database designers are known to make occasional mistakes that come to light during normalization checks. However, many of the real-world databases you encounter will have been improperly designed or burdened with anomalies if they were

improperly modified during the course of time. And that means you might be asked to redesign and modify existing databases that are, in effect, anomaly traps. Therefore, you should be aware of good design principles and procedures as well as normalization procedures.

First, an ERD is created through an iterative process. You begin by identifying relevant entities, their attributes, and their relationships. Then you use the results to identify additional entities and attributes. The ERD provides the big picture, or macro view, of an organization's data requirements and operations.

Second, normalization focuses on the characteristics of specific entities; that is, normalization represents a micro view of the entities within the ERD. And as you learned in the previous sections of this chapter, the normalization process might yield additional entities and attributes to be incorporated into the ERD. Therefore, it is difficult to separate the normalization process from the ER modeling process; the two techniques are used in an iterative and incremental process.

To illustrate the proper role of normalization in the design process, let's reexamine the operations of the contracting company whose tables were normalized in the preceding sections. Those operations can be summarized by using the following business rules:

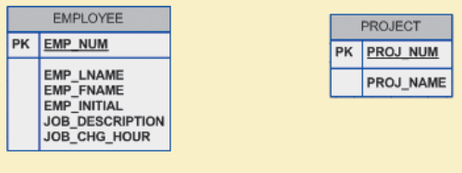
- The company manages many projects.
- Each project requires the services of many employees.
- An employee may be assigned to several different projects.
- Some employees are not assigned to a project and perform duties not specifically related to a project. Some employees are part of a labor pool, to be shared by all project teams. For example, the company's executive secretary would not be assigned to any one particular project.
- Each employee has a single primary job classification. That job classification determines the hourly billing rate.
- Many employees can have the same job classification. For example, the company employs more than one electrical engineer.

Given that simple description of the company's operations, two entities and their attributes are initially defined:

- PROJECT (**PROJ_NUM**, PROJ_NAME)
- EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_DESCRIPTION, JOB_CHG_HOUR)

Those two entities constitute the initial ERD shown in Figure 5.12.

FIGURE 5.12 **Initial contracting company ERD**



After creating the initial ERD shown in Figure 5.12, the normal forms are defined:

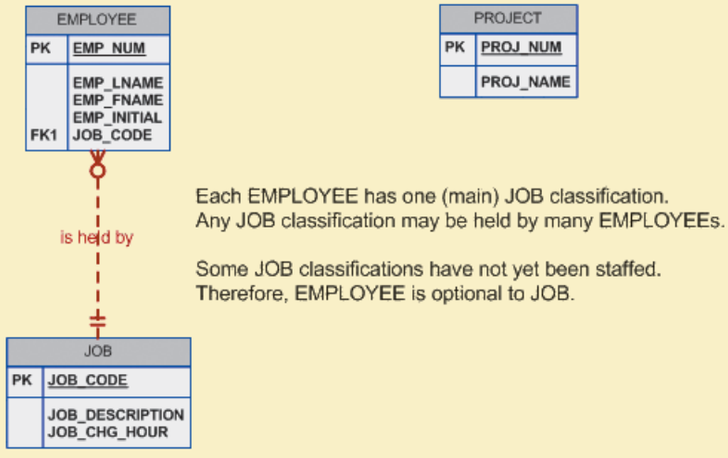
- PROJECT is in 3NF and needs no modification at this point.
- EMPLOYEE requires additional scrutiny. The JOB_DESCRIPTION attribute defines job classifications such as Systems Analyst, Database Designer, and Programmer. In turn, those classifications determine the billing rate, JOB_CHG_HOUR. Therefore, EMPLOYEE contains a transitive dependency.

The removal of EMPLOYEE's transitive dependency yields three entities:

- PROJECT (**PROJ_NUM**, PROJ_NAME)
- EMPLOYEE (**EMP_NUM**, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_CODE)
- JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

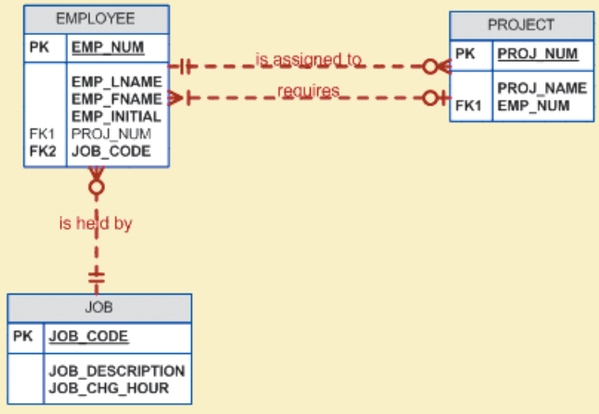
Because the normalization process yields an additional entity (JOB), the initial ERD is modified as shown in Figure 5.13.

FIGURE 5.13 **Modified contracting company ERD**



To represent the M:N relationship between EMPLOYEE and PROJECT, you might think that two 1:M relationships could be used—an employee can be assigned to many projects, and each project can have many employees assigned to it. See Figure 5.14. Unfortunately, that representation yields a design that cannot be correctly implemented.

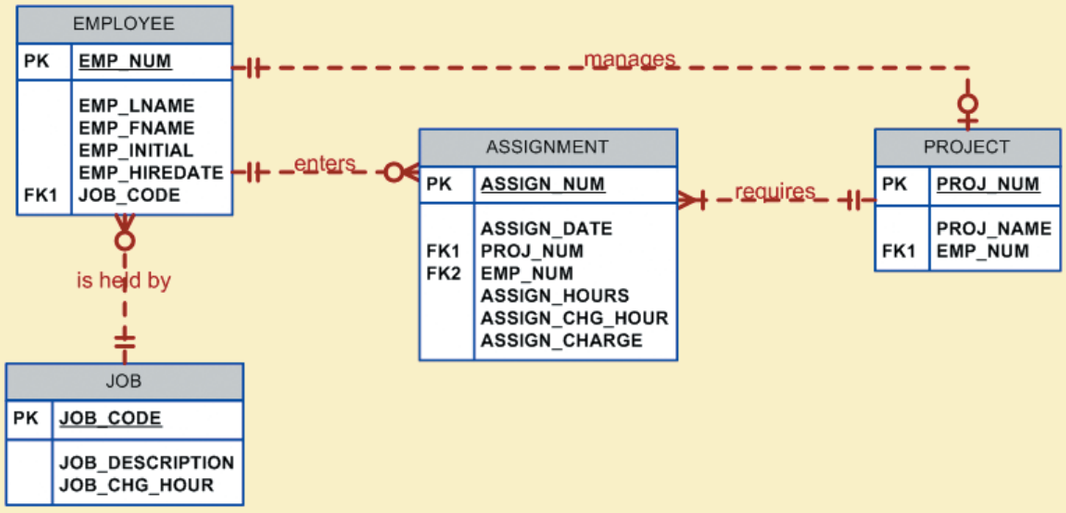
FIGURE 5.14 **Incorrect M:N relationship representation**



Because the M:N relationship between EMPLOYEE and PROJECT cannot be implemented, the ERD in Figure 5.14 must be modified to include the ASSIGNMENT entity to track the assignment of employees to projects, thus yielding the ERD shown in Figure 5.15. The ASSIGNMENT entity in Figure 5.15 uses the primary keys from the entities PROJECT and EMPLOYEE to serve as its foreign keys. However, note that in this implementation, the ASSIGNMENT entity's surrogate primary key is ASSIGN_NUM, to avoid the use of a composite primary key. Therefore, the "*enters*"

relationship between EMPLOYEE and ASSIGNMENT and the "*requires*" relationship between PROJECT and ASSIGNMENT are shown as weak or nonidentifying.

FIGURE 5.15 **Final contracting company ERD**



Note that in Figure 5.15, the ASSIGN_HOURS attribute is assigned to the composite entity named ASSIGNMENT. Because you will likely need detailed information about each project's manager, the creation of a "manages" relationship is useful. The "manages" relationship is implemented through the foreign key in PROJECT. Finally, some additional attributes may be created to improve the system's ability to generate additional information. For example, you may want to include the date on which the employee was hired (EMP_HIREDATE) to keep track of worker longevity. Based on this last modification, the model should include four entities and their attributes:

PROJECT (**PROJ_NUM**, PROJ_NAME, EMP_NUM)

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

JOB (**JOB_CODE**, JOB_DESCRIPTION, JOB_CHG_HOUR)

ASSIGNMENT (**ASSIGN_NUM**, ASSIGN_DATE, PROJ_NUM, EMP_NUM, ASSIGN_HOURS, ASSIGN_CHG_HOUR, ASSIGN_CHARGE)

The design process is now on the right track. The ERD represents the operations accurately, and the entities now reflect their conformance to 3NF. The combination of normalization and ER modeling yields a useful ERD, whose entities may now be translated into appropriate table structures. In Figure 5.15, note that PROJECT is optional to EMPLOYEE in the "manages" relationship. This optionality exists because not all employees manage projects. The final database contents are shown in Figure 5.16.

FIGURE 5.16 The implemented database

Table name: EMPLOYEE **Database name: Ch05_ConstructCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Table name: PROJECT

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-08	15	103	2.6	84.50	219.70
1002	04-Mar-08	18	118	1.4	18.36	25.70
1003	05-Mar-08	15	101	3.6	105.00	378.00
1004	05-Mar-08	22	113	2.5	48.10	120.25
1005	05-Mar-08	15	103	1.9	84.50	160.55
1006	05-Mar-08	25	115	4.2	96.75	406.35
1007	05-Mar-08	22	105	5.2	105.00	546.00
1008	05-Mar-08	25	101	1.7	105.00	178.50
1009	05-Mar-08	15	105	2.0	105.00	210.00
1010	06-Mar-08	15	102	3.8	96.75	367.65
1011	06-Mar-08	22	104	2.6	96.75	251.55
1012	06-Mar-08	15	101	2.3	105.00	241.50
1013	06-Mar-08	25	114	1.8	48.10	86.58
1014	06-Mar-08	22	111	4.0	26.87	107.48
1015	06-Mar-08	25	114	3.4	48.10	163.54
1016	06-Mar-08	18	112	1.2	45.95	55.14
1017	06-Mar-08	18	118	2.0	18.36	36.72
1018	06-Mar-08	18	104	2.6	96.75	251.55
1019	06-Mar-08	15	103	3.0	84.50	253.50
1020	07-Mar-08	22	105	2.7	105.00	283.50
1021	08-Mar-08	25	108	4.2	96.75	406.35
1022	07-Mar-08	25	114	5.8	48.10	278.98
1023	07-Mar-08	22	106	2.4	35.75	85.80

5.8 DENORMALIZATION

It's important to remember that the optimal relational database implementation requires that all tables be at least in third normal form (3NF). A good relational DBMS excels at managing normalized relations; that is, relations void of any unnecessary redundancies that might cause data anomalies. Although the creation of normalized relations is an important database design goal, it is only one of many such goals. Good database design also considers processing (or reporting) requirements and processing speed. The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Therefore, in order to generate information, data must be put together from various tables. Joining a large number of tables takes additional

input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems are able to handle joins very efficiently. However, rare and occasional circumstances may allow some degree of denormalization so processing speed can be increased.

Keep in mind that the advantage of higher processing speed must be carefully weighed against the disadvantage of data anomalies. On the other hand, some anomalies are of only theoretical interest. For example, should people in a real-world database environment worry that a ZIP_CODE determines CITY in a CUSTOMER table whose primary key is the customer number? Is it really practical to produce a separate table for

ZIP (**ZIP_CODE**, CITY)

to eliminate a transitive dependency from the CUSTOMER table? (Perhaps your answer to that question changes if you are in the business of producing mailing lists.) As explained earlier, the problem with denormalized relations and redundant data is that the data integrity could be compromised due to the possibility of data anomalies (insert, update, and deletion anomalies.) The advice is simple: use common sense during the normalization process.

Furthermore, the database design process could, in some cases, introduce some small degree of redundant data in the model (as seen in the previous example). This, in effect, creates "denormalized" relations. Table 5.6 shows some common examples of data redundancy that are generally found in database implementations.

TABLE 5.6 Common Denormalization Examples

CASE	EXAMPLE	RATIONALE AND CONTROLS
Redundant data	Storing ZIP and CITY attributes in the CUSTOMER table when ZIP determines CITY. (See Table 1.3.)	• Avoid extra join operations • Program can validate city (drop-down box) based on the zip code.
Derived data	Storing STU_HRS and STU_CLASS (student classification) when STU_HRS determines STU_CLASS. (See Figure 3.29.)	• Avoid extra join operations • Program can validate classification (lookup) based on the student hours
Pre-aggregated data (also derived data)	Storing the student grade point average (STU_GPA) aggregate value in the STUDENT table when this can be calculated from the ENROLL and COURSE tables. (See Figure 3.29.)	• Avoid extra join operations • Program computes the GPA every time a grade is entered or updated. • STU_GPA can be updated only via administrative routine.
Information requirements	Using a temporary denormalized table to hold report data. This is required when creating a tabular report in which the columns represent data that is stored in the table as rows. (See Figure 5.17 and Figure 5.18.)	• Impossible to generate the data required by the report using plain SQL. • No need to maintain table. Temporary table is deleted once report is done. • Processing speed is not an issue.

A more comprehensive example of the need for denormalization due to reporting requirements is the case of a faculty evaluation report in which each row list the scores obtained during the last four semesters taught. See Figure 5.17.

FIGURE 5.17 The faculty evaluation report

Faculty Evaluation Report

Instructor	Department	I		II		III		IV		Last Two Sem. Avg.
		Semester	Mean	Semester	Mean	Semester	Mean	Semester	Mean	
Alton	INFS	2005S	2.91	2004F	2.84	2004S	2.55	2003F	2.51	2.875
Ames	INFS	2005S	3.24	2004F	3.26	2004S	3.31	2003F	3.19	3.250
Crandon	INFS	2005S	3.93	2004F	3.95	2004S	3.91	2003F	3.88	3.940
Dumas	MGMT	2004F	3.66	2004S	3.69	2003F	3.56	2003S	3.72	3.675
Landon	BMOM	2005S	3.57	2004F	3.64	2004S	3.39	2003F	3.57	3.605
Lohar	ECON	1999F	3.53	1998F	3.53					3.530
Rolman	INFS	1996S	3.50							3.500

Although this report seems simple enough, the problem arises from the fact that the data are stored in a normalized table in which each row represents a different score for a given faculty in a given semester. See Figure 5.18.

FIGURE 5.18 The EVALDATA and FACHIST tables

Table name: EVALDATA

ID	INSTRUCTOR	DEPARTMENT	MEAN	SEMESTER
3730	Alton	INFS	2.2	2002F
3764	Alton	INFS	2.69	2002S
3975	Alton	INFS	2.51	2003F
4172	Alton	INFS	2.13	2003S
4450	Alton	INFS	2.84	2004F
4323	Alton	INFS	2.55	2004S
4571	Alton	INFS	2.91	2005S
2091	Ames	INFS	2.84	1996F
2215	Ames	INFS	2.76	1996S
2332	Ames	INFS	2.79	1997F
2450	Ames	INFS	2.74	1997S
2573	Ames	INFS	2.6	1998F
2699	Ames	INFS	2.92	1998S
2823	Ames	INFS	2.09	1999F
2948	Ames	INFS	2.98	1999S
3071	Ames	INFS	3.35	2000F
3205	Ames	INFS	2.92	2000S
3337	Ames	INFS	3.24	2001F
3473	Ames	INFS	2.79	2001S
3668	Ames	INFS	3.28	2002F
3765	Ames	INFS	3.24	2002S
3976	Ames	INFS	3.19	2003F
4155	Ames	INFS	2.59	2003S
4414	Ames	INFS	3.26	2004F
4269	Ames	INFS	3.31	2004S
4572	Ames	INFS	3.24	2005S
3613	Crandon	INFS	3.79	2002F
3977	Crandon	INFS	3.88	2003F
4040	Crandon	INFS	3.79	2003S
4346	Crandon	INFS	3.95	2004F
4199	Crandon	INFS	3.91	2004S
4573	Crandon	INFS	3.93	2005S

Table name: FACHIST **Database name: Ch05_EVAL**

ID	INSTRUCTOR	DEPARTMENT	LAST1SEM	LAST1MEAN	LAST2SEM	LAST2MEAN	LAST3SEM	LAST3MEAN	LAST4SEM	LAST4MEAN
59602	Alton	INFS	2005S	2.91	2004F	2.84	2004S	2.55	2003F	2.51
59603	Ames	INFS	2005S	3.24	2004F	3.26	2004S	3.31	2003F	3.19
59605	Crandon	INFS	2005S	3.93	2004F	3.95	2004S	3.91	2003F	3.88
59607	Dumas	MGMT	2004F	3.66	2004S	3.69	2003F	3.56	2003S	3.72
59608	Landon	BMOM	2005S	3.57	2004F	3.64	2004S	3.39	2003F	3.57
59610	Lohar	ECON	1999F	3.53	1998F	3.53				
59611	Rolman	INFS	1996S	3.5						

Denormalized

Repeating Group

Normalized

The difficulty of transposing multirow data to multicolumnar data is compounded by the fact that the last four semesters taught are not necessarily the same for all faculty members (some might have taken sabbaticals, some might have had research appointments, some might be new faculty with only two semesters on the job, etc.) To generate this report, the two tables you see in Figure 5.18 were used. The EVALDATA table is the master data table containing the evaluation scores for each faculty member for each semester taught; this table is normalized. The FACHIST table contains the last four data points—that is, evaluation score and semester—for each faculty member. The FACHIST table is a temporary denormalized table created from the EVALDATA table via a series of queries. (The FACHIST table is the basis for the report shown in Figure 5.17.)

As seen in the faculty evaluation report, the conflicts between design efficiency, information requirements, and performance are often resolved through compromises that may include denormalization. In this case and assuming there is enough storage space, the designer's choices could be narrowed down to:

- Store the data in a permanent denormalized table. This is not the recommended solution, because the denormalized table is subject to data anomalies (insert, update, and delete.) This solution is viable only if performance is an issue.
- Create a temporary denormalized table from the permanent normalized table(s). Because the denormalized table exists only as long as it takes to generate the report, it disappears after the report is produced. Therefore, there are no data anomaly problems. This solution is practical only if performance is not an issue and there are no other viable processing options.

As shown, normalization purity is often difficult to sustain in the modern database environment. You will learn in Chapter 13, Business Intelligence and Data Warehouses, that lower normalization forms occur (and are even required) in specialized databases known as data warehouses. Such specialized databases reflect the ever-growing demand for greater scope and depth in the data on which decision support systems increasingly rely. You will discover that the data warehouse routinely uses 2NF structures in its complex, multilevel, multisource data environment. In short, although normalization is very important, especially in the so-called production database environment, 2NF is no longer disregarded as it once was.

Although 2NF tables cannot always be avoided, the problem of working with tables that contain partial and/or transitive dependencies in a production database environment should not be minimized. Aside from the possibility of troublesome data anomalies being created, unnormalized tables in a production database tend to suffer from these defects:

- Data updates are less efficient because programs that read and update tables must deal with larger tables.
- Indexing is more cumbersome. It simply is not practical to build all of the indexes required for the many attributes that might be located in a single unnormalized table.
- Unnormalized tables yield no simple strategies for creating virtual tables known as *views*. You will learn how to create and use views in Chapter 7, Introduction to Structured Query Language (SQL).

Remember that good design cannot be created in the application programs that use a database. Also keep in mind that unnormalized database tables often lead to various data redundancy disasters in production databases such as the ones examined thus far. In other words, use denormalization cautiously and make sure that you can explain why the unnormalized tables are a better choice in certain situations than their normalized counterparts.

SUMMARY

- Normalization is a technique used to design tables in which data redundancies are minimized. The first three normal forms (1NF, 2NF, and 3NF) are most commonly encountered. From a structural point of view, higher normal forms are better than lower normal forms because higher normal forms yield relatively fewer data redundancies in the database. Almost all business designs use 3NF as the ideal normal form. A special, more restricted 3NF known as Boyce-Codd normal form, or BCNF, is also used.
- A table is in 1NF when all key attributes are defined and when all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies. (A partial dependency is one in which an attribute is functionally dependent on only a part of a multiattribute primary key. A transitive dependency is one in which one attribute is functionally dependent on another nonkey attribute.) A table with a single-attribute primary key cannot exhibit partial dependencies.
- A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute. A table in 2NF may still contain transitive dependencies.
- A table is in 3NF when it is in 2NF and contains no transitive dependencies. Given that definition of 3NF, the Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys. When a table has only a single candidate key, a 3NF table is automatically in BCNF.
- A table that is not in 3NF may be split into new tables until all of the tables meet the 3NF requirements. The process is illustrated in Figures 5.19 to 5.21.

FIGURE 5.19 The initial 1NF structure

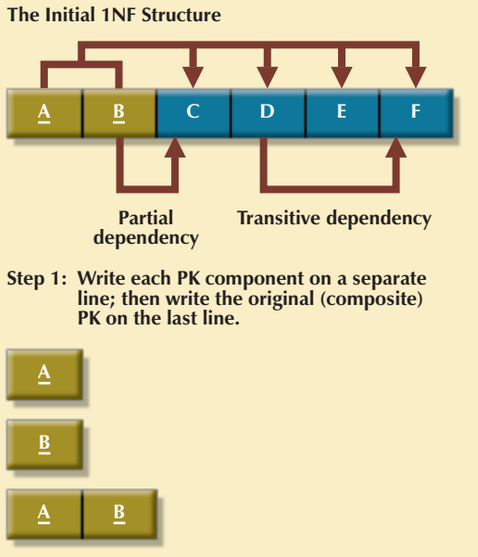


FIGURE 5.20 Identifying possible PK attributes

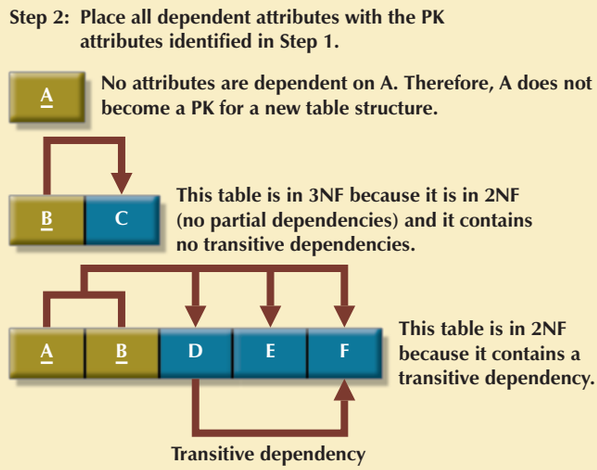
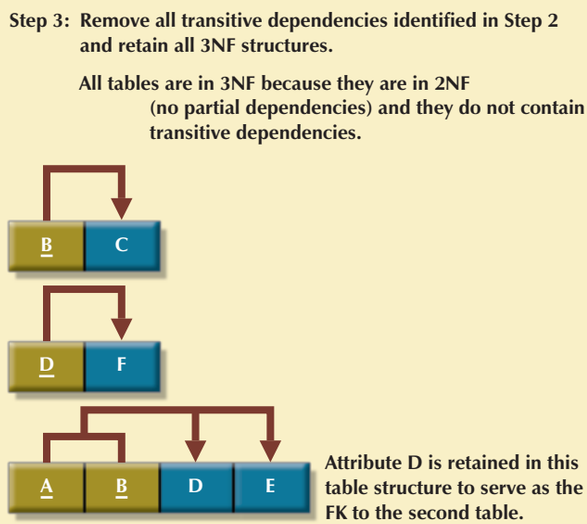


FIGURE 5.21 Table structures based on the selected PKs



- Normalization is an important part—but only a part—of the design process. As entities and attributes are defined during the ER modeling process, subject each entity (set) to normalization checks and form new entity (sets) as required. Incorporate the normalized entities into the ERD and continue the iterative ER process until all entities and their attributes are defined and all equivalent tables are in 3NF.
- A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant data. Therefore, it might be necessary to convert a 3NF table to the fourth normal form (4NF) by splitting the table to

remove the multivalued dependencies. Thus, a table is in 4NF when it is in 3NF and contains no multivalued dependencies.

- The larger the number of tables, the more additional I/O operations and processing logic required to join them. Therefore, tables are sometimes denormalized to yield less I/O in order to increase processing speed. Unfortunately, with larger tables, you pay for the increased processing speed by making the data updates less efficient, by making indexing more cumbersome, and by introducing data redundancies that are likely to yield data anomalies. In the design of production databases, use denormalization sparingly and cautiously.

KEY TERMS

atomic attribute, 165
atomicity, 165
Boyce-Codd normal form (BCNF), 170
denormalization, 153
dependency diagram, 160
determinant, 163
first normal form (1NF), 161
fourth normal form (4NF), 174
granularity, 166
key attribute, 162
nonkey attribute, 162
nonprime attribute, 162
normalization, 153
partial dependency, 160
prime attribute, 162
repeating group, 158
second normal form (2NF), 163
surrogate key, 165
third normal form (3NF), 164
transitive dependency, 160

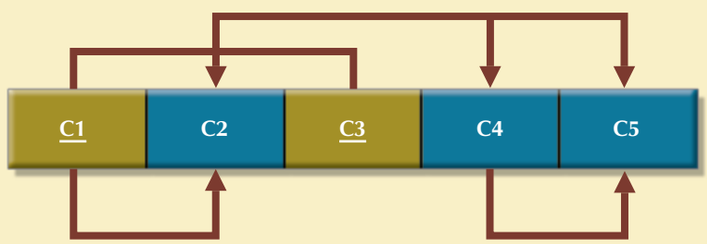
ONLINE CONTENT

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

REVIEW QUESTIONS

1. What is normalization?
2. When is a table in 1NF?
3. When is a table in 2NF?
4. When is a table in 3NF?
5. When is a table in BCNF?
6. Given the dependency diagram shown in Figure Q5.6, answer Items 6a–6c.

FIGURE Q5.6 Dependency diagram for Question 6



a. Identify and discuss each of the indicated dependencies.
b. Create a database whose tables are at least in 2NF, showing the dependency diagrams for each table.
c. Create a database whose tables are at least in 3NF, showing the dependency diagrams for each table.

7. What is a partial dependency? With what normal form is it associated?
8. What three data anomalies are likely to be the result of data redundancy? How can such anomalies be eliminated?
9. Define and discuss the concept of transitive dependency.
10. What is a surrogate key, and when should you use one?
11. Why is a table whose primary key consists of a single attribute automatically in 2NF when it is in 1NF?
12. How would you describe a condition in which one attribute is dependent on another attribute, when neither attribute is part of the primary key?
13. Suppose that someone tells you that an attribute that is part of a composite primary key is also a candidate key. How would you respond to that statement?
14. A table is in _____ normal form when it is in _____ and there are no transitive dependencies.

PROBLEMS

1. Using the INVOICE table structure shown below, write the relational schema, draw its dependency diagram, and identify all dependencies, including all partial and transitive dependencies. You can assume that the table does not contain repeating groups and that an invoice number references more than one product. (*Hint*: This table uses a composite primary key.)

TABLE P5.1

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
INV_NUM	211347	211347	211347	211348	211349
PROD_NUM	AA-E3422QW	QD-300932X	RU-995748G	AA-E3422QW	GH-778345P
SALE_DATE	15-Jan-2008	15-Jan-2008	15-Jan-2008	15-Jan-2008	16-Jan-2008
PROD_LABEL	Rotary sander	0.25-in. drill bit	Band saw	Rotary sander	Power drill
VEND_CODE	211	211	309	211	157
VEND_NAME	NeverFail, Inc.	NeverFail, Inc.	BeGood, Inc.	NeverFail, Inc.	ToughGo, Inc.
QUANT_SOLD	1	8	1	2	1
PROD_PRICE	$49.95	$3.45	$39.99	$49.95	$87.75

2. Using the answer to Problem 1, remove all partial dependencies, write the relational schema, and draw the new dependency diagrams. Identify the normal forms for each table structure you created.

NOTE

You can assume that any given product is supplied by a single vendor, but a vendor can supply many products. Therefore, it is proper to conclude that the following dependency exists:

PROD_NUM → PROD_DESCRIPTION, PROD_PRICE, VEND_CODE, VEND_NAME

(*Hint*: Your actions should produce three dependency diagrams.)

3. Using the answer to Problem 2, remove all transitive dependencies, write the relational schema, and draw the new dependency diagrams. Also identify the normal forms for each table structure you created.
4. Using the results of Problem 3, draw the Crow's Foot ERD.

5. Using the STUDENT table structure shown in Table P5.5, write the relational schema and draw its dependency diagram. Identify all dependencies, including all transitive dependencies.

TABLE P5.5

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
STU_NUM	211343	200128	199876	199876	223456
STU_LNAME	Stephanos	Smith	Jones	Ortiz	McKulski
STU_MAJOR	Accounting	Accounting	Marketing	Marketing	Statistics
DEPT_CODE	ACCT	ACCT	MKTG	MKTG	MATH
DEPT_NAME	Accounting	Accounting	Marketing	Marketing	Mathematics
DEPT_PHONE	4356	4356	4378	4378	3420
COLLEGE_NAME	Business Admin	Business Admin	Business Admin	Business Admin	Arts & Sciences
ADVISOR_LNAME	Grastrand	Grastrand	Gentry	Tillery	Chen
ADVISOR_OFFICE	T201	T201	T228	T356	J331
ADVISOR_BLDG	Torre Building	Torre Building	Torre Building	Torre Building	Jones Building
ADVISOR_PHONE	2115	2115	2123	2159	3209
STU_GPA	3.87	2.78	2.31	3.45	3.58
STU_HOURS	75	45	117	113	87
STU_CLASS	Junior	Sophomore	Senior	Senior	Junior

6. Using the answer to Problem 5, write the relational schema and draw the dependency diagram to meet the 3NF requirements to the greatest practical extent possible. If you believe that practical considerations dictate using a 2NF structure, explain why your decision to retain 2NF is appropriate. If necessary, add or modify attributes to create appropriate determinants and to adhere to the naming conventions.

NOTE

Although the completed student hours (STU_HOURS) do determine the student classification (STU_CLASS), this dependency is not as obvious as you might initially assume it to be. For example, a student is considered a junior if that student has completed between 61 and 90 credit hours. Therefore, a student who is classified as a junior may have completed 66, 72, or 87 hours or any other number of hours within the specified range of 61–90 hours. In short, any hour value within a specified range will define the classification.

7. Using the results of Problem 6, draw the Crow's Foot ERD.

NOTE

This ERD constitutes a small segment of a university's full-blown design. For example, this segment might be combined with the Tiny College presentation in Chapter 4.

8. To keep track of office furniture, computers, printers, and so on, the FOUNDIT company uses the table structure shown in Table P5.8.

TABLE P5.8

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
ITEM_ID	231134-678	342245-225	254668-449
ITEM_LABEL	HP DeskJet 895Cse	HP Toner	DT Scanner
ROOM_NUMBER	325	325	123
BLDG_CODE	NTC	NTC	CSF
BLDG_NAME	Nottooclear	Nottoclear	Canseefar
BLDG_MANAGER	I. B. Rightonit	I. B. Rightonit	May B. Next

Given that information, write the relational schema and draw the dependency diagram. Make sure that you label the transitive and/or partial dependencies.

9. Using the answer to Problem 8, write the relational schema and create a set of dependency diagrams that meet 3NF requirements. Rename attributes to meet the naming conventions and create new entities and attributes as necessary.
10. Using the results of Problem 9, draw the Crow's Foot ERD.

NOTE

Problems 11–14 may be combined to serve as a case or a miniproject.

11. The table structure shown in Table P5.11 contains many unsatisfactory components and characteristics. For example, there are several multivalued attributes, naming conventions are violated, and some attributes are not atomic.

TABLE P5.11

EMP_NUM	1003	1018	1019	1023
EMP_LNAME	Willaker	Smith	McGuire	McGuire
EMP_EDUCATION	BBA, MBA	BBA		BS, MS, Ph.D.
JOB_CLASS	SLS	SLS	JNT	DBA
EMP_DEPENDENTS	Gerald (spouse), Mary (daughter), John (son)		JoAnne (spouse)	George (spouse) Jill (daughter)
DEPT_CODE	MKTG	MKTG	SVC	INFS
DEPT_NAME	Marketing	Marketing	General Service	Info. Systems
DEPT_MANAGER	Jill H. Martin	Jill H. Martin	Hank B. Jones	Carlos G. Ortez
EMP_TITLE	Sales Agent	Sales Agent	Janitor	DB Admin
EMP_DOB	23-Dec-1968	28-Mar-1979	18-May-1982	20-Jul-1959
EMP_HIRE_DATE	14-Oct-1997	15-Jan-2006	21-Apr-2003	15-Jul-1999
EMP_TRAINING	L1, L2	L1	L1	L1, L3, L8, L15
EMP_BASE_SALARY	$38,255.00	$30,500.00	$19,750.00	$127,900.00
EMP_COMMISSION_RATE	0.015	0.010		

Given the structure shown in Table P5.11, write the relational schema and draw its dependency diagram. Label all transitive and/or partial dependencies.

12. Using the answer to Problem 11, draw the dependency diagrams that are in 3NF. (*Hint*: You might have to create a few new attributes. Also make sure that the new dependency diagrams contain attributes that meet proper

design criteria; that is, make sure that there are no multivalued attributes, that the naming conventions are met, and so on.)

13. Using the results of Problem 12, draw the relational diagram.
14. Using the results of Problem 13, draw the Crow's Foot ERD.

NOTE

Problems 15-17 may be combined to serve as a case or a miniproject.

15. Suppose you are given the following business rules to form the basis for a database design. The database must enable the manager of a company dinner club to mail invitations to the club's members, to plan the meals, to keep track of who attends the dinners, and so on.
    - Each dinner serves many members, and each member may attend many dinners.
    - A member receives many invitations, and each invitation is mailed to many members.
    - A dinner is based on a single entree, but an entree may be used as the basis for many dinners. For example, a dinner may be composed of a fish entree, rice, and corn. Or the dinner may be composed of a fish entree, a baked potato, and string beans.
    - A member may attend many dinners, and each dinner may be attended by many members.

    Because the manager is not a database expert, the first attempt at creating the database uses the structure shown in Table P5.15.

TABLE P5.15

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
MEMBER_NUM	214	235	214
MEMBER_NAME	Alice B. VanderVoort	Gerald M. Gallega	Alice B. VanderVoort
MEMBER_ADDRESS	325 Meadow Park	123 Rose Court	325 Meadow Park
MEMBER_CITY	Murkywater	Highlight	Murkywater
MEMBER_ZIPCODE	12345	12349	12345
INVITE_NUM	8	9	10
INVITE_DATE	23-Feb-2008	12-Mar-2008	23-Feb-2008
ACCEPT_DATE	27-Feb-2008	15-Mar-2008	27-Feb-2008
DINNER_DATE	15-Mar-2008	17-Mar-2008	15-Mar-2008
DINNER_ATTENDED	Yes	Yes	No
DINNER_CODE	DI5	DI5	DI2
DINNER_DESCRIPTION	Glowing Sea Delight	Glowing Sea Delight	Ranch Superb
ENTREE_CODE	EN3	EN3	EN5
ENTREE_DESCRIPTION	Stuffed crab	Stuffed crab	Marinated steak
DESSERT_CODE	DE8	DE5	DE2
DESSERT_DESCRIPTION	Chocolate mousse with raspberry sauce	Cherries jubilee	Apple pie with honey crust

Given the table structure illustrated in Table P5.15, write the relational schema and draw its dependency diagram. Label all transitive and/or partial dependencies. (*Hint*: This structure uses a composite primary key.)

16. Break up the dependency diagram you drew in Problem 15 to produce dependency diagrams that are in 3NF and write the relational schema. (*Hint*: You might have to create a few new attributes. Also make sure that the new dependency diagrams contain attributes that meet proper design criteria; that is, make sure that there are no multivalued attributes, that the naming conventions are met, and so on.)

17. Using the results of Problem 16, draw the Crow's Foot ERD.

NOTE

Problems 18–20 may be combined to serve as a case or a miniproject.

18. The manager of a consulting firm has asked you to evaluate a database that contains the table structure shown in Table P5.18.

TABLE P5.18

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
CLIENT_NUM	298	289	289
CLIENT_NAME	Marianne R. Brown	James D. Smith	James D. Smith
CLIENT_REGION	Midwest	Southeast	Southeast
CONTRACT_DATE	10-Feb-2008	15-Feb-2008	12-Mar-2008
CONTRACT_NUMBER	5841	5842	5843
CONTRACT_AMOUNT	$2,985,000.00	$670,300.00	$1,250,000.00
CONSULT_CLASS_1	Database Administration	Internet Services	Database Design
CONSULT_CLASS_2	Web Applications		Database Administration
CONSULT_CLASS_3			Network Installation
CONSULT_CLASS_4			
CONSULTANT_NUM_1	29	34	25
CONSULTANT_NAME_1	Rachel G. Carson	Gerald K. Ricardo	Angela M. Jamison
CONSULTANT_REGION_1	Midwest	Southeast	Southeast
CONSULTANT_NUM_2	56	38	34
CONSULTANT_NAME_2	Karl M. Spenser	Anne T. Dimarco	Gerald K. Ricardo
CONSULTANT_REGION_2	Midwest	Southeast	Southeast
CONSULTANT_NUM_3	22	45	
CONSULTANT_NAME_3	Julian H. Donatello	Geraldo J. Rivera	
CONSULTANT_REGION_3	Midwest	Southeast	
CONSULTANT_NUM_4		18	
CONSULTANT_NAME_4		Donald Chen	
CONSULTANT_REGION_4		West	

Table P5.18 was created to enable the manager to match clients with consultants. The objective is to match a client within a given region with a consultant in that region and to make sure that the client's need for specific consulting services is properly matched to the consultant's expertise. For example, if the client needs help with database design and is located in the Southeast, the objective is to make a match with a consultant who is located in the Southeast and whose expertise is in database design. (Although the consulting company manager tries to match consultant and client locations to minimize travel expense, it is not always possible to do so.) The following basic business rules are maintained:

- Each client is located in one region.
- A region can contain many clients.
- Each consultant can work on many contracts.
- Each contract might require the services of many consultants.
- A client can sign more than one contract, but each contract is signed by only one client.
- Each contract might cover multiple consulting classifications. (For example, a contract may list consulting services in database design and networking.)
- Each consultant is located in one region.

- A region can contain many consultants.
- Each consultant has one or more areas of expertise (class). For example, a consultant might be classified as an expert in both database design and networking.
- Each area of expertise (class) can have many consultants in it. For example, the consulting company might employ many consultants who are networking experts.

Given that brief description of the requirements and the business rules, write the relational schema and draw the dependency diagram for the preceding (and very poor) table structure. Label all transitive and/or partial dependencies.

19. Break up the dependency diagram you drew in Problem 18 to produce dependency diagrams that are in 3NF and write the relational schema. (*Hint*: You might have to create a few new attributes. Also make sure that the new dependency diagrams contain attributes that meet proper design criteria; that is, make sure that there are no multivalued attributes, that the naming conventions are met, and so on.)
20. Using the results of Problem 19, draw the Crow's Foot ERD.
21. Given the sample records in the CHARTER table shown in Table P5.21, write the relational schema and draw the dependency diagram for the table structure. Make sure that you label all dependencies. CHAR_PAX indicates the number of passengers carried. The CHAR_MILES entry is based on round-trip miles, including pickup points. (*Hint*: Look at the data values to determine the nature of the relationships. For example, note that employee Melton has flown two charter trips as pilot and one trip as copilot.)

TABLE P5.21

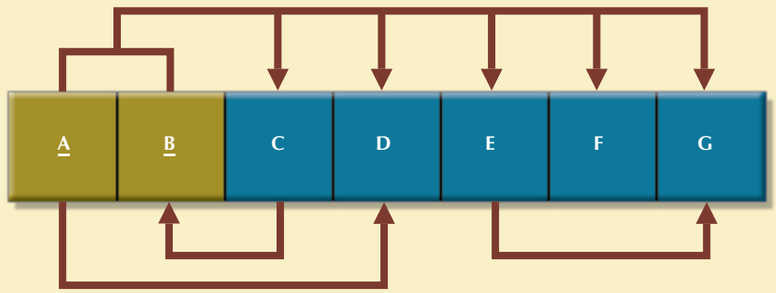
ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
CHAR_TRIP	10232	10233	10234	10235
CHAR_DATE	15-Jan-2008	15-Jan-2008	16-Jan-2008	17-Jan-2008
CHAR_CITY	STL	MIA	TYS	ATL
CHAR_MILES	580	1,290	524	768
CUST_NUM	784	231	544	784
CUST_LNAME	Brown	Hanson	Bryana	Brown
CHAR_PAX	5	12	2	5
CHAR_CARGO	235 lbs.	18,940 lbs.	348 lbs.	155 lbs.
PILOT	Melton	Chen	Henderson	Melton
COPILOT		Henderson	Melton	
FLT_ENGINEER		O'Shaski		
LOAD_MASTER		Benkasi		
AC_NUMBER	1234Q	3456Y	1234Q	2256W
MODEL_CODE	PA31-350	CV-580	PA31-350	PA31-350
MODEL_SEATS	10	38	10	10
MODEL_CHG_MILE	$2.79	$23.36	$2.79	$2.79

22. Decompose the dependency diagram you drew to solve Problem 21 to create table structures that are in 3NF and write the relational schema. Make sure that you label all dependencies.
23. Draw the Crow's Foot ERD to reflect the properly decomposed dependency diagrams you created in Problem 22. Make sure that the ERD yields a database that can track all of the data shown in Problem 21. Show all entities, relationships, connectivities, optionalities, and cardinalities.

NOTE

Use the dependency diagram shown in Figure P5.24 to work Problems 24–26.

FIGURE P5.24 Initial dependency diagram for Problems 24—26



24. Break up the dependency diagram shown in Figure P5.24 to create two new dependency diagrams, one in 3NF and one in 2NF.
25. Modify the dependency diagrams you created in Problem 24 to produce a set of dependency diagrams that are in 3NF. To keep the entire collection of attributes together, copy the 3NF dependency diagram from Problem 24; then show the new dependency diagrams that are also in 3NF. (*Hint*: One of your dependency diagrams will be in 3NF but not in BCNF.)
26. Modify the dependency diagrams you created in Problem 25 to produce a collection of dependency diagrams that are in 3NF and BCNF. To ensure that all attributes are accounted for, copy the 3NF dependency diagrams from Problem 25; then show the new 3NF and BCNF dependency diagrams.

27. Suppose you have been given the table structure and data shown in Table P5.27, which was imported from an Excel spreadsheet. The data reflect that a professor can have multiple advisees, can serve on multiple committees, and can edit more than one journal.

TABLE P5.27

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
EMP_NUM	123	104	118	
PROF_RANK	Professor	Asst. Professor	Assoc. Professor	Assoc. Professor
EMP_NAME	Ghee	Rankin	Ortega	Smith
DEPT_CODE	CIS	CHEM	CIS	ENG
DEPT_NAME	Computer Info. Systems	Chemistry	Computer Info. Systems	English
PROF_OFFICE	KDD-567	BLF-119	KDD-562	PRT-345
ADVISEE	1215, 2312, 3233, 2218, 2098	3102, 2782, 3311, 2008, 2876, 2222, 3745, 1783, 2378	2134, 2789, 3456, 2002, 2046, 2018, 2764	2873, 2765, 2238, 2901, 2308
COMMITTEE_CODE	PROMO, TRAF, APPL, DEV	DEV	SPR, TRAF	PROMO, SPR, DEV
JOURNAL_CODE	JMIS, QED, JMGT		JCIS, JMGT	

Given the information in Table P5.27:

a. Draw the dependency diagram.

b. Identify the multivalued dependencies.

c. Create the dependency diagrams to yield a set of table structures in 3NF.

d. Eliminate the multivalued dependencies by converting the affected table structures to 4NF.

e. Draw the Crow's Foot ERD to reflect the dependency diagrams you drew in Part c. (*Note:* You might have to create additional attributes to define the proper PKs and FKs. Make sure that all of your attributes conform to the naming conventions.)