CHAPTER

14 Normalization

Chapter Objectives

In this chapter you will learn:

- The purpose of normalization.
- How normalization can be used when designing a relational database.
- The potential problems associated with redundant data in base relations.
- The concept of functional dependency, which describes the relationship between attributes.
- The characteristics of functional dependencies used in normalization.
- How to identify functional dependencies for a given relation.
- How functional dependencies identify the primary key for a relation.
- How to undertake the process of normalization.
- How normalization uses functional dependencies to group attributes into relations that are in a known normal form.
- How to identify the most commonly used normal forms: First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).
- The problems associated with relations that break the rules of 1NF, 2NF, or 3NF.
- How to represent attributes shown on a form as 3NF relations using normalization.

When we design a database for an enterprise, the main objective is to create an accurate representation of the data, relationships between the data, and constraints on the data that is pertinent to the enterprise. To help achieve this objective, we can use one or more database design techniques. In Chapters 12 and 13 we described a technique called ER modeling. In this chapter and the next we describe another database design technique called **normalization**.

Normalization is a database design technique that begins by examining the relationships (called functional dependencies) between attributes. Attributes describe some property of the data or of the relationships between the data that is important to the enterprise. Normalization uses a series of tests (described as normal forms) to help identify the optimal grouping for these attributes to ultimately identify a set of suitable relations that supports the data requirements of the enterprise.

Although the main purpose of this chapter is to introduce the concept of functional dependencies and describe normalization up to Third Normal Form (3NF), in Chapter 15 we take a more formal look at functional dependencies and also consider later normal forms that go beyond 3NF.

Structure of this Chapter In Section 14.1 we describe the purpose of normalization. In Section 14.2 we discuss how normalization can be used to support relational database design. In Section 14.3 we identify and illustrate the potential problems associated with data redundancy in a base relation that is not normalized. In Section 14.4 we describe the main concept associated with normalization, called functional dependency, which describes the relationship between attributes. We also describe the characteristics of the functional dependencies that are used in normalization. In Section 14.5 we present an overview of normalization and then proceed in the following sections to describe the process involving the three most commonly used normal forms, namely First Normal Form (1NF) in Section 14.6, Second Normal Form (2NF) in Section 14.7, and Third Normal Form (3NF) in Section 14.8. The 2NF and 3NF described in these sections are based on the *primary key* of a relation. In Section 14.9 we present general definitions for 2NF and 3NF based on all *candidate keys* of a relation.

Throughout this chapter we use examples taken from the *DreamHome* case study described in Section 11.4 and documented in Appendix A.

14.1 The Purpose of Normalization

Normalization A technique for producing a set of relations with desirable properties, given the data requirements of an enterprise.

The purpose of normalization is to identify a suitable set of relations that support the data requirements of an enterprise. The characteristics of a suitable set of relations include the following:

- the *minimal* number of attributes necessary to support the data requirements of the enterprise;
- attributes with a close logical relationship (described as functional dependency) are found in the same relation;
- *minimal* redundancy, with each attribute represented only once, with the important exception of attributes that form all or part of foreign keys (see Section 4.2.5), which are essential for the joining of related relations.

The benefits of using a database that has a suitable set of relations is that the database will be easier for the user to access and maintain the data, and take up

minimal storage space on the computer. The problems associated with using a relation that is not appropriately normalized is described later in Section 14.3.

14.2 How Normalization Supports Database Design

Normalization is a formal technique that can be used at any stage of database design. However, in this section we highlight two main approaches for using normalization, as illustrated in Figure 14.1. Approach 1 shows how normalization can be used as a bottom-up standalone database design technique, and Approach 2 shows how normalization can be used as a validation technique to check the structure of relations, which may have been created using a top-down approach such as ER modeling. No matter which approach is used, the goal is the same; creating a set of well-designed relations that meet the data requirements of the enterprise.

Figure 14.1 shows examples of data sources that can be used for database design. Although the users' requirements specification (see Section 10.5) is the preferred data source, it is possible to design a database based on the information taken directly from other data sources, such as forms and reports, as illustrated in this chapter and the next. Figure 14.1 also shows that the same data source can be used for both approaches; however, although this is true in principle, in practice the approach taken is likely to be determined by the size, extent, and complexity of the database being described by the data sources and by the preference and expertise of the database designer. The opportunity to use

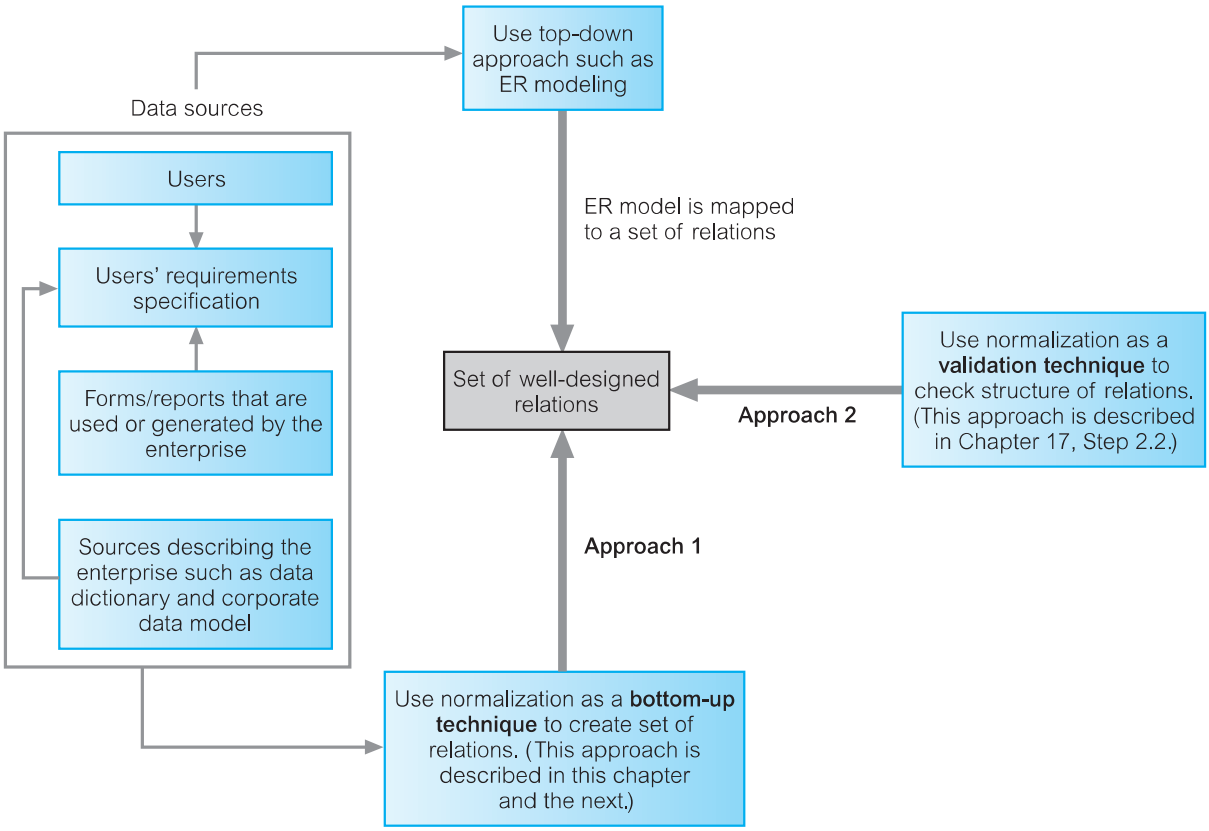

Figure 14.1 How normalization can be used to support database design.

normalization as a bottom-up standalone technique (Approach 1) is often limited by the level of detail that the database designer is reasonably expected to manage. However, this limitation is not applicable when normalization is used as a validation technique (Approach 2), as the database designer focuses on only part of the database, such as a single relation, at any one time. Therefore, no matter what the size or complexity of the database, normalization can be usefully applied.

14.3 Data Redundancy and Update Anomalies

As stated in Section 14.1, a major aim of relational database design is to group attributes into relations to minimize data redundancy. If this aim is achieved, the potential benefits for the implemented database include the following:

- updates to the data stored in the database are achieved with a minimal number of operations, thus reducing the opportunities for data inconsistencies occurring in the database;
- reduction in the file storage space required by the base relations thus minimizing costs.

Of course, relational databases also rely on the existence of a certain amount of data redundancy. This redundancy is in the form of copies of primary keys (or candidate keys) acting as foreign keys in related relations to enable the modeling of relationships between data.

In this section we illustrate the problems associated with unwanted data redundancy by comparing the Staff and Branch relations shown in Figure 14.2 with the StaffBranch relation shown in Figure 14.3. The StaffBranch relation is an alternative format of the Staff and Branch relations. The relations have the following form:

Staff	(staffNo, sName, position, salary, branchNo)
Branch	(branchNo, bAddress)
StaffBranch	(staffNo, sName, position, salary, branchNo, bAddress)

Figure 14.2
Staff and Branch relations.

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Figure 14.3 StaffBranch relation.

Note that the primary key for each relation is underlined.

In the StaffBranch relation there is redundant data; the details of a branch are repeated for every member of staff located at that branch. In contrast, the branch details appear only once for each branch in the Branch relation, and only the branch number (branchNo) is repeated in the Staff relation to represent where each member of staff is located. Relations that have redundant data may have problems called **update anomalies**, which are classified as insertion, deletion, or modification anomalies.

14.3.1 Insertion Anomalies

There are two main types of insertion anomaly, which we illustrate using the StaffBranch relation shown in Figure 14.3:

- To insert the details of new members of staff into the StaffBranch relation, we must include the details of the branch at which the staff are to be located. For example, to insert the details of new staff located at branch number B007, we must enter the correct details of branch number B007 so that the branch details are consistent with values for branch B007 in other tuples of the StaffBranch relation. The relations shown in Figure 14.2 do not suffer from this potential inconsistency, because we enter only the appropriate branch number for each staff member in the Staff relation. Instead, the details of branch number B007 are recorded in the database as a single tuple in the Branch relation.
- To insert details of a new branch that currently has no members of staff into the StaffBranch relation, it is necessary to enter nulls into the attributes for staff, such as staffNo. However, as staffNo is the primary key for the StaffBranch relation, attempting to enter nulls for staffNo violates entity integrity (see Section 4.3), and is not allowed. We therefore cannot enter a tuple for a new branch into the StaffBranch relation with a null for the staffNo. The design of the relations shown in Figure 14.2 avoids this problem, because branch details are entered in the Branch relation separately from the staff details. The details of staff ultimately located at that branch are entered at a later date into the Staff relation.

14.3.2 Deletion Anomalies

If we delete a tuple from the StaffBranch relation that represents the last member of staff located at a branch, the details about that branch are also lost from the database. For example, if we delete the tuple for staff number SA9 (Mary Howe)

from the StaffBranch relation, the details relating to branch number B007 are lost from the database. The design of the relations in Figure 14.2 avoids this problem, because branch tuples are stored separately from staff tuples and only the attribute branchNo relates the two relations. If we delete the tuple for staff number SA9 from the Staff relation, the details on branch number B007 remain unaffected in the Branch relation.

14.3.3 Modification Anomalies

If we want to change the value of one of the attributes of a particular branch in the StaffBranch relation—for example, the address for branch number B003—we must update the tuples of all staff located at that branch. If this modification is not carried out on all the appropriate tuples of the StaffBranch relation, the database will become inconsistent. In this example, branch number B003 may appear to have different addresses in different staff tuples.

The previous examples illustrate that the Staff and Branch relations of Figure 14.2 have more desirable properties than the StaffBranch relation of Figure 14.3. This demonstrates that although the StaffBranch relation is subject to update anomalies, we can avoid these anomalies by decomposing the original relation into the Staff and Branch relations. There are two important properties associated with decomposition of a larger relation into smaller relations:

- The **lossless-join** property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations.
- The **dependency preservation** property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations. In other words, we do not need to perform joins on the smaller relations to check whether a constraint on the original relation is violated.

Later in this chapter, we discuss how the process of normalization can be used to derive well-formed relations. However, we first introduce functional dependencies, which are fundamental to the process of normalization.

14.4 Functional Dependencies

An important concept associated with normalization is **functional dependency**, which describes the relationship between attributes (Maier, 1983). In this section we describe functional dependencies and then focus on the particular characteristics of functional dependencies that are useful for normalization. We then discuss how functional dependencies can be identified and used to identify the primary key for a relation.

14.4.1 Characteristics of Functional Dependencies

For the discussion on functional dependencies, assume that a relational schema has attributes (A, B, C, . . . , Z) and that the database is described by a single **universal relation** called R = (A, B, C, . . . , Z). This assumption means that every attribute in the database has a unique name.

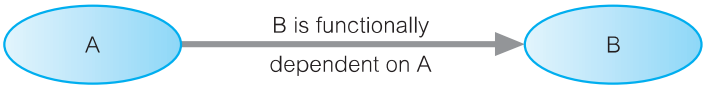

Figure 14.4 A functional dependency diagram.

Functional dependency Describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A ® B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes.)

Functional dependency is a property of the meaning or semantics of the attributes in a relation. The semantics indicate how attributes relate to one another, and specify the functional dependencies between attributes. When a functional dependency is present, the dependency is specified as a **constraint** between the attributes.

Consider a relation with attributes A and B, where attribute B is functionally dependent on attribute A. If we know the value of A and we examine the relation that holds this dependency, we find only one value of B in all the tuples that have a given value of A, at any moment in time. Thus, when two tuples have the same value of A, they also have the same value of B. However, for a given value of B, there may be several different values of A. The dependency between attributes A and B can be represented diagrammatically, as shown Figure 14.4.

An alternative way to describe the relationship between attributes A and B is to say that "A functionally determines B." Some readers may prefer this description, as it more naturally follows the direction of the functional dependency arrow between the attributes.

Determinant Refers to the attribute, or group of attributes, on the left-hand side of the arrow of a functional dependency.

When a functional dependency exists, the attribute or group of attributes on the left-hand side of the arrow is called the **determinant**. For example, in Figure 14.4, A is the determinant of B. We demonstrate the identification of a functional dependency in the following example.

EXAMPLE 14.1 An example of a functional dependency

Consider the attributes staffNo and position of the Staff relation in Figure 14.2. For a specific staffNo—for example, SL21—we can determine the position of that member of staff as Manager. In other words, staffNo functionally determines position, as shown in Figure 14.5(a). However, Figure 14.5(b) illustrates that the opposite is not true, as position does not functionally determine staffNo. A member of staff holds one position; however, there may be several members of staff with the same position.

The relationship between staffNo and position is one-to-one (1:1): for each staff number there is only one position. On the other hand, the relationship between position and staffNo is one-to-many (1:*): there are several staff numbers associated with a given position. In this example, staffNo is the determinant of this functional dependency. For the purposes of normalization, we are interested in identifying functional dependencies between attributes of a relation that have a one-to-one relationship between the attribute(s) that

makes up the determinant on the left-hand side and the attribute(s) on the right-hand side of a dependency.

When identifying functional dependencies between attributes in a relation, it is important to distinguish clearly between the values held by an attribute at a given point in time and the *set of all possible values* that an attribute may hold at different times. In other words, a functional dependency is a property of a relational schema (intension) and not a property of a particular instance of the schema (extension) (see Section 4.2.1). This point is illustrated in the following example.

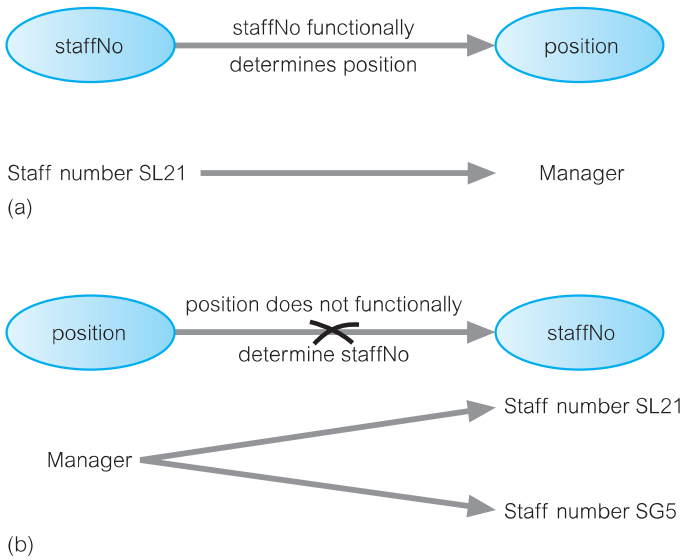

Figure 14.5 (a) staffNo functionally determines position (staffNo ® position); (b) position does *not* functionally determine staffNo (position ⊗® staffNo).

EXAMPLE 14.2 Example of a functional dependency that holds for all time

Consider the values shown in staffNo and sName attributes of the Staff relation in Figure 14.2. We see that for a specific staffNo—SL21—for example, we can determine the name of that member of staff as John White. Furthermore, it appears that for a specific sName—for example, John White—we can determine the staff number for that member of staff as SL21. Can we therefore conclude that the staffNo attribute functionally determines the sName attribute and/or that the sName attribute functionally determines the staffNo attribute? If the values shown in the Staff relation of Figure 14.2 represent the *set of all possible values* for staffNo and sName attributes, then the following functional dependencies hold:

staffNo ® sName
sName ® staffNo

However, if the values shown in the Staff relation of Figure 14.2 simply represent a *set of values* for staffNo and sName attributes at a given moment in time, then we are not so interested in such relationships between attributes. The reason is that we want to identify functional dependencies that hold for all possible values for attributes of a relation as these represent the types of integrity constraints that we need to identify. Such constraints indicate the limitations on the values that a relation can legitimately assume.

One approach to identifying the set of all possible values for attributes in a relation is to more clearly understand the purpose of each attribute in that relation. For example,

the purpose of the values held in the staffNo attribute is to uniquely identify each member of staff, whereas the purpose of the values held in the sName attribute is to hold the names of members of staff. Clearly, the statement that if we know the staff number (staffNo) of a member of staff, we can determine the name of the member of staff (sName) remains true. However, as it is possible for the sName attribute to hold duplicate values for members of staff with the same name, then we would not be able to determine the staff number (staffNo) of some members of staff in this category. The relationship between staffNo and sName is one-to-one (1:1): for each staff number there is only one name. On the other hand, the relationship between sName and staffNo is one-to-many (1:*): there can be several staff numbers associated with a given name. The functional dependency that remains true after consideration of all possible values for the staffNo and sName attributes of the Staff relation is:

staffNo ® sName

An additional characteristic of functional dependencies that is useful for normalization is that their determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the righthand side. This requirement is called **full functional dependency**.

Full functional dependency	Indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of A.

A functional dependency A ® B is a *full* functional dependency if removal of any attribute from A results in the dependency no longer existing. A functional dependency A ® B is a **partial dependency** if there is some attribute that can be removed from A and yet the dependency still holds. An example of how a full functional dependency is derived from a partial functional dependency is presented in Example 14.3.

EXAMPLE 14.3 Example of a full functional dependency

Consider the following functional dependency that exists in the Staff relation of Figure 14.2:

staffNo, sName ® branchNo

It is correct to say that each value of (staffNo, sName) is associated with a single value of branchNo. However, it is not a full functional dependency, because branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo. In other words, the functional dependency shown in the example is an example of a partial dependency. The type of functional dependency that we are interested in identifying is a full functional dependency as shown here:

staffNo ® branchNo

Additional examples of partial and full functional dependencies are discussed in Section 14.7.

In summary, the functional dependencies that we use in normalization have the following characteristics:

- There is a *one-to-one* relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency. (Note that the relationship in the opposite direction—that is, from the right-hand to the left-hand side attributes—can be a one-to-one relationship or one-to-many relationship.)
- They hold for *all* time.
- The determinant has the *minimal* number of attributes necessary to maintain the dependency with the attribute(s) on the right-hand side. In other words, there must be a full functional dependency between the attribute(s) on the left-hand and right-hand sides of the dependency.

So far we have discussed functional dependencies that we are interested in for the purposes of normalization. However, there is an additional type of functional dependency called a **transitive dependency** that we need to recognize, because its existence in a relation can potentially cause the types of update anomaly discussed in Section 14.3. In this section we simply describe these dependencies so that we can identify them when necessary.

Transitive dependency	A condition where A, B, and C are attributes of a relation such that if A ® B and B ® C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

An example of a transitive dependency is provided in Example 14.4.

EXAMPLE 14.4 Example of a transitive functional dependency

Consider the following functional dependencies within the StaffBranch relation shown in Figure 14.3:

staffNo ® sName, position, salary, branchNo, bAddress
branchNo ® bAddress

The transitive dependency branchNo ® bAddress exists on staffNo via branchNo. In other words, the staffNo attribute functionally determines the bAddress via the branchNo attribute and neither branchNo nor bAddress functionally determines staffNo. An additional example of a transitive dependency is discussed in Section 14.8.

In the following sections we demonstrate approaches to identifying a set of functional dependencies and then discuss how these dependencies can be used to identify a primary key for the example relations.

14.4.2 Identifying Functional Dependencies

Identifying all functional dependencies between a set of attributes should be quite simple if the meaning of each attribute and the relationships between the attributes are well understood. This type of information may be provided by the enterprise in

the form of discussions with users and/or appropriate documentation, such as the users' requirements specification. However, if the users are unavailable for consultation and/or the documentation is incomplete, then—depending on the database application—it may be necessary for the database designer to use their common sense and/or experience to provide the missing information. Example 14.5 illustrates how easy it is to identify functional dependencies between attributes of a relation when the purpose of each attribute and the attributes' relationships are well understood.

EXAMPLE 14.5 Identifying a set of functional dependencies for the StaffBranch relation

We begin by examining the semantics of the attributes in the StaffBranch relation shown in Figure 14.3. For the purposes of discussion, we assume that the position held and the branch determine a member of staff's salary. We identify the functional dependencies based on our understanding of the attributes in the relation as:

staffNo ® sName, position, salary, branchNo, bAddress
branchNo ® bAddress
bAddress ® branchNo
branchNo, position ® salary
bAddress, position ® salary

We identify five functional dependencies in the StaffBranch relation with staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position) as determinants. For each functional dependency, we ensure that *all* the attributes on the right-hand side are functionally dependent on the determinant on the left-hand side.

As a contrast to this example, we now consider the situation where functional dependencies are to be identified in the absence of appropriate information about the meaning of attributes and their relationships. In this case, it may be possible to identify functional dependencies if sample data is available that is a true representation of *all* possible data values that the database may hold. We demonstrate this approach in Example 14.6.

EXAMPLE 14.6 Using sample data to identify functional dependencies

Consider the data for attributes denoted A, B, C, D, and E in the Sample relation of Figure 13.6. It is important first to establish that the data values shown in this relation are representative of all possible values that can be held by attributes A, B, C, D, and E. For the purposes of this example, let us assume that this is true despite the relatively small amount of data shown in this relation. The process of identifying the functional dependencies (denoted fd1 to fd5) that exist between the attributes of the Sample relation shown in Figure 14.6 is described next.

To identify the functional dependencies that exist between attributes A, B, C, D, and E, we examine the Sample relation shown in Figure 14.6 and identify when values in one column are consistent with the presence of particular values in other columns. We begin with the first column on the left-hand side and work our way over to the right-hand side of the relation and then we look at combinations of columns; in other words, where values in two or more columns are consistent with the appearance of values in other columns.

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t

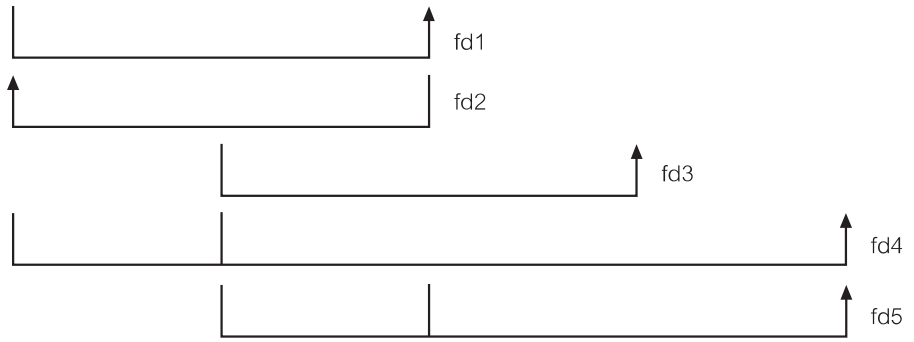

Figure 14.6 The Sample relation displaying data for attributes A, B, C, D, and E and the functional dependencies (fd1 to fd5) that exist between these attributes.

For example, when the value "a" appears in column A, the value "z" appears in column C, and when "e" appears in column A, the value "r" appears in column C. We can therefore conclude that there is a one-to-one (1:1) relationship between attributes A and C. In other words, attribute A functionally determines attribute C and this is shown as functional dependency 1 (fd1) in Figure 14.6. Furthermore, as the values in column C are consistent with the appearance of particular values in column A, we can also conclude that there is a (1:1) relationship between attributes C and A. In other words, C functionally determines A, and this is shown as fd2 in Figure 14.6. If we now consider attribute B, we can see that when "b" or "d" appears in column B, then "w" appears in column D and when "f" appears in column B, then "s" appears in column D. We can therefore conclude that there is a (1:1) relationship between attributes B and D. In other words, B functionally determines D, and this is shown as fd3 in Figure 14.6. However, attribute D does not functionally determine attribute B as a single unique value in column D, such as "w" is not associated with a single consistent value in column B. In other words, when "w" appears in column D, the values "b" *or* "d" appears in column B. Hence, there is a one-to-many relationship between attributes D and B. The final single attribute to consider is E, and we find that the values in this column are not associated with the consistent appearance of particular values in the other columns. In other words, attribute E does not functionally determine attributes A, B, C, or D.

We now consider combinations of attributes and the appearance of consistent values in other columns. We conclude that unique combination of values in columns A and B such as (a, b) is associated with a single value in column E, which in this example is "q." In other words attributes (A, B) functionally determines attribute E, and this is shown as fd4 in Figure 14.6. However, the reverse is not true, as we have already stated that attribute E, does not functionally determine any other attribute in the relation. We also conclude that attributes (B, C) functionally determine attribute E using the same reasoning described earlier, and this functional dependency is shown as fd5 in

Figure 14.6. We complete the examination of the relation shown in Figure 14.6 by considering all the remaining combinations of columns.

In summary, we describe the function dependencies between attributes A to E in the Sample relation shown in Figure 14.6 as follows:

A ® C	(fdl)
C ® A	(fd2)
B ® D	(fd3)
A, B ® E	(fd4)
B, C ® E	(fd5)

14.4.3 Identifying the Primary Key for a Relation Using Functional Dependencies

The main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation. An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation. We demonstrate the identification of a primary key for a given relation in the following two examples.

EXAMPLE 14.7 Identifying the primary key for the StaffBranch relation

In Example 14.5 we describe the identification of five functional dependencies for the StaffBranch relation shown in Figure 14.3. The determinants for these functional dependencies are staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position).

To identify the candidate key(s) for the StaffBranch relation, we must identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation. If a relation has more than one candidate key, we identify the candidate key that is to act as the primary key for the relation (see Section 4.2.5). All attributes that are not part of the primary key (non-primary-key attributes) should be functionally dependent on the key.

The only candidate key of the StaffBranch relation, and therefore the primary key, is staffNo, as *all* other attributes of the relation are functionally dependent on staffNo. Although branchNo, bAddress, (branchNo, position), and (bAddress, position) are determinants in this relation, they are not candidate keys for the relation.

EXAMPLE 14.8 Identifying the primary key for the Sample relation

In Example 14.6 we identified five functional dependencies for the Sample relation. We examine the determinant for each functional dependency to identify the candidate key(s) for the relation. A suitable determinant must functionally determine the other attributes in the relation. The determinants in the Sample relation are A, B, C, (A, B), and (B, C). However, the only determinants that determine all the other attributes of the relation are (A, B) and (B, C). In the case of (A, B), A functionally determines C, B functionally determines D, and (A, B) functionally determines E. In other words, the attributes that make up the determinant (A, B) can determine all the other attributes in the relation either separately as A or B or together as (A, B). Hence, we see that an essential characteristic for a candidate key of a relation is that the attributes of a determinant

either individually or working together must be able to functionally determine *all* the other attributes in the relation. This characteristic is also true for the determinant (B, C), but is not a characteristic of the other determinants in the Sample relation (namely A, B, or C), as in each case they can determine only one other attribute in the relation. In conclusion, there are two candidate keys for the Sample relation; namely, (A, B) and (B, C), and as each has similar characteristics (such as number of attributes), the selection of primary key for the Sample relation is arbitrary. The candidate key not selected to be the primary key is referred to as the alternate key for the Sample relation.

So far in this section we have discussed the types of functional dependency that are most useful in identifying important constraints on a relation and how these dependencies can be used to identify a primary key (or candidate keys) for a given relation. The concepts of functional dependencies and keys are central to the process of normalization. We continue the discussion on functional dependencies in the next chapter for readers interested in a more formal coverage of this topic. However, in this chapter, we continue by describing the process of normalization.

14.5 The Process of Normalization

Normalization is a formal technique for analyzing relations based on their primary key (or candidate keys) and functional dependencies (Codd, 1972b). The technique involves a series of rules that can be used to test individual relations so that a database can be normalized to any degree. When a requirement is not met, the relation violating the requirement must be decomposed into relations that individually meet the requirements of normalization.

Three normal forms were initially proposed called First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF). Subsequently, R. Boyce and E. F. Codd introduced a stronger definition of third normal form called Boyce–Codd Normal Form (BCNF) (Codd, 1974). With the exception of 1NF, all these normal forms are based on functional dependencies among the attributes of a relation (Maier, 1983). Higher normal forms that go beyond BCNF were introduced later such as Fourth Normal Form (4NF) and Fifth Normal Form (5NF) (Fagin, 1977, 1979). However, these later normal forms deal with situations that are very rare. In this chapter we describe only the first three normal forms and leave discussion of BCNF, 4NF, and 5NF to the next chapter.

Normalization is often executed as a series of steps. Each step corresponds to a specific normal form that has known properties. As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies. For the relational data model, it is important to recognize that it is only First Normal Form (1NF) that is critical in creating relations; all subsequent normal forms are optional. However, to avoid the update anomalies discussed in Section 14.3, it is generally recommended that we proceed to at least Third Normal Form (3NF). Figure 14.7 illustrates the relationship between the various normal forms. It shows that some 1NF relations are also in 2NF, and that some 2NF relations are also in 3NF, and so on.

In the following sections we describe the process of normalization in detail. Figure 14.8 provides an overview of the process and highlights the main actions

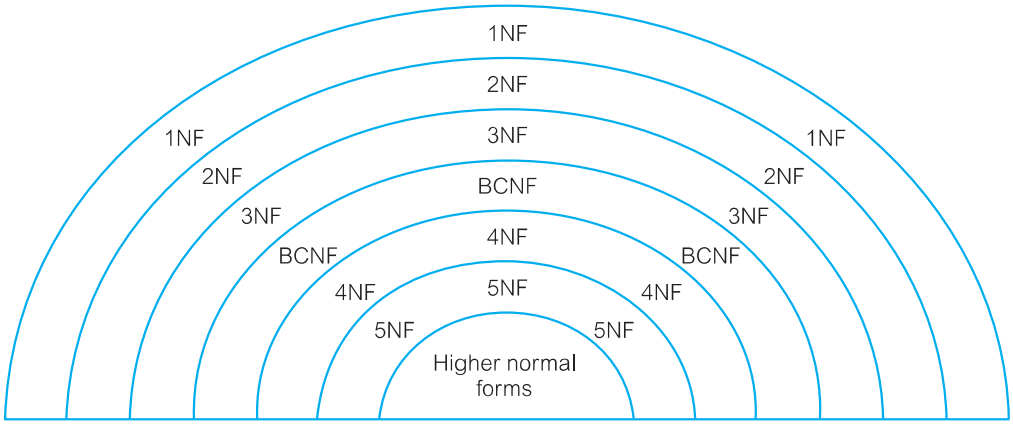

Figure 14.7 Diagrammatic illustration of the relationship between the normal forms.

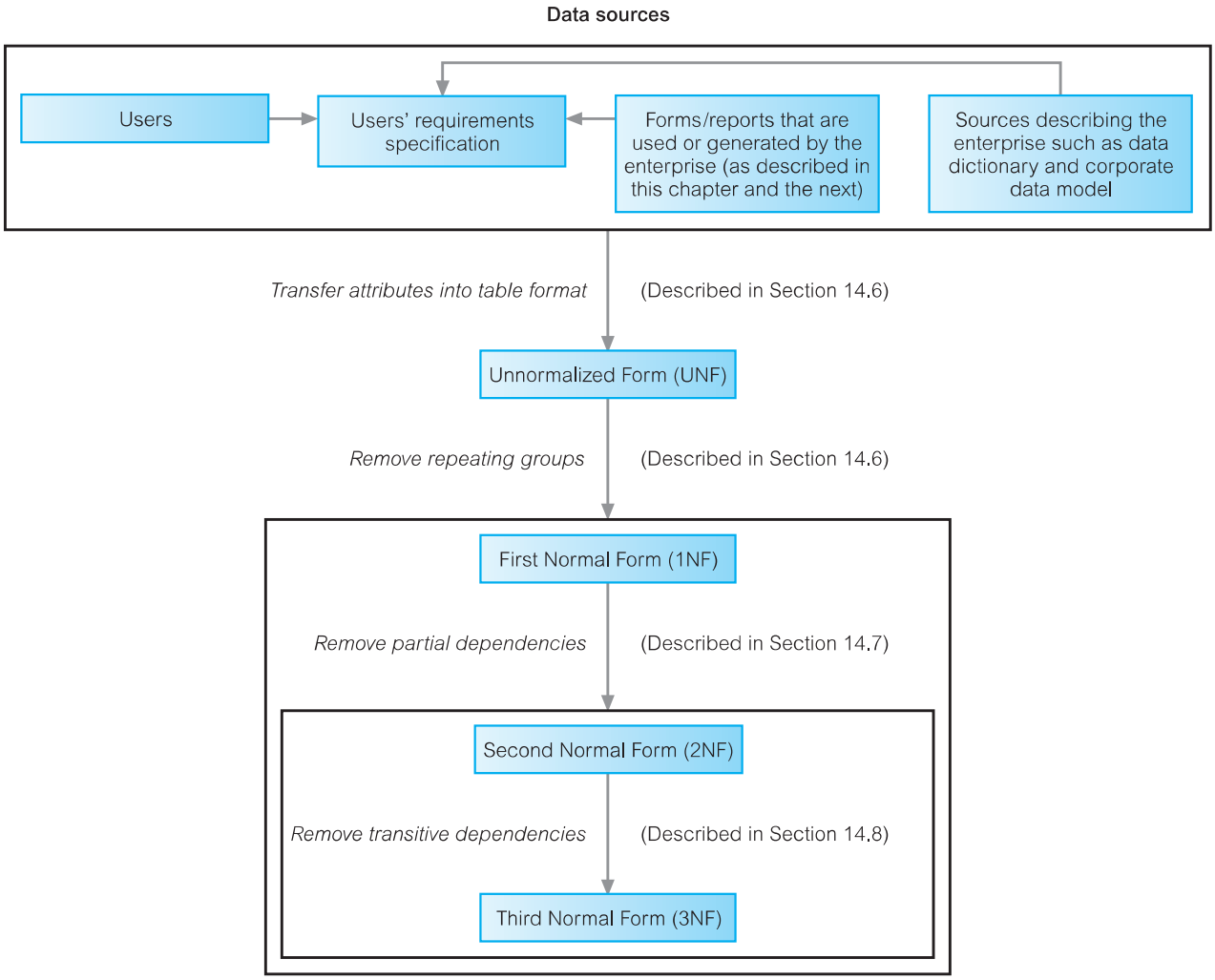

Figure 14.8 Diagrammatic illustration of the process of normalization.

taken in each step of the process. The number of the section that covers each step of the process is also shown in this figure.

In this chapter, we describe normalization as a bottom-up technique extracting information about attributes from sample forms that are first transformed into table format, which is described as being in Unnormalized Form (UNF). This table is then subjected progressively to the different requirements associated with each normal form until ultimately the attributes shown in the original sample forms are represented as a set of 3NF relations. Although the example used in this chapter proceeds from a given normal form to the one above, this is not necessarily the case with other examples. As shown in Figure 13.8, the resolution of a particular problem with, say, a 1NF relation may result in the relation being transformed to 2NF relations, or in some cases directly into 3NF relations in one step.

To simplify the description of normalization we assume that a set of functional dependencies is given for each relation in the worked examples and that each relation has a designated primary key. In other words, it is essential that the meaning of the attributes and their relationships is well understood before beginning the process of normalization. This information is fundamental to normalization and is used to test whether a relation is in a particular normal form. In Section 14.6 we begin by describing First Normal Form (1NF). In Sections 14.7 and 14.8 we describe Second Normal Form (2NF) and Third Normal Forms (3NF) based on the *primary key* of a relation and then present a more general definition of each in Section 14.9. The more general definitions of 2NF and 3NF take into account all *candidate keys* of a relation, rather than just the primary key.

14.6 First Normal Form (1NF)

Before discussing First Normal Form, we provide a definition of the state prior to First Normal Form.

Unnormalized Form (UNF) A table that contains one or more repeating groups.

First Normal Form (1NF) A relation in which the intersection of each row and column contains one and only one value.

In this chapter, we begin the process of normalization by first transferring the data from the source (for example, a standard data entry form) into table format with rows and columns. In this format, the table is in unnormalized Form and is referred to as an **unnormalized table**. To transform the unnormalized table to First Normal Form, we identify and remove repeating groups within the table. A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table. Note that in this context, the term "key" refers to the attribute(s) that uniquely identify each row within the Unnormalized table. There are two common approaches to removing repeating groups from unnormalized tables:

(1) *By entering appropriate data in the empty columns of rows containing the repeating data.* In other words, we fill in the blanks by duplicating the nonrepeating data, where required. This approach is commonly referred to as "flattening" the table.

(2) *By placing the repeating data, along with a copy of the original key attribute(s), in a separate relation.* Sometimes the unnormalized table may contain more than one repeating group, or repeating groups within repeating groups. In such cases, this approach is applied repeatedly until no repeating groups remain. A set of relations is in 1NF if it contains no repeating groups.

For both approaches, the resulting tables are now referred to as 1NF relations containing atomic (or single) values at the intersection of each row and column. Although both approaches are correct, approach 1 introduces more redundancy into the original UNF table as part of the "flattening" process, whereas approach 2 creates two or more relations with less redundancy than in the original UNF table. In other words, approach 2 moves the original UNF table further along the normalization process than approach 1. However, no matter which initial approach is taken, the original UNF table will be normalized into the same set of 3NF relations.

We demonstrate both approaches in the following worked example using the *DreamHome* case study.

EXAMPLE 14.9 First Normal Form (1NF)

A collection of (simplified) *DreamHome* leases is shown in Figure 14.9. The lease on top is for a client called John Kay who is leasing a property in Glasgow, which is owned by Tina Murphy. For this worked example, we assume that a client rents a given property only once and cannot rent more than one property at any one time.

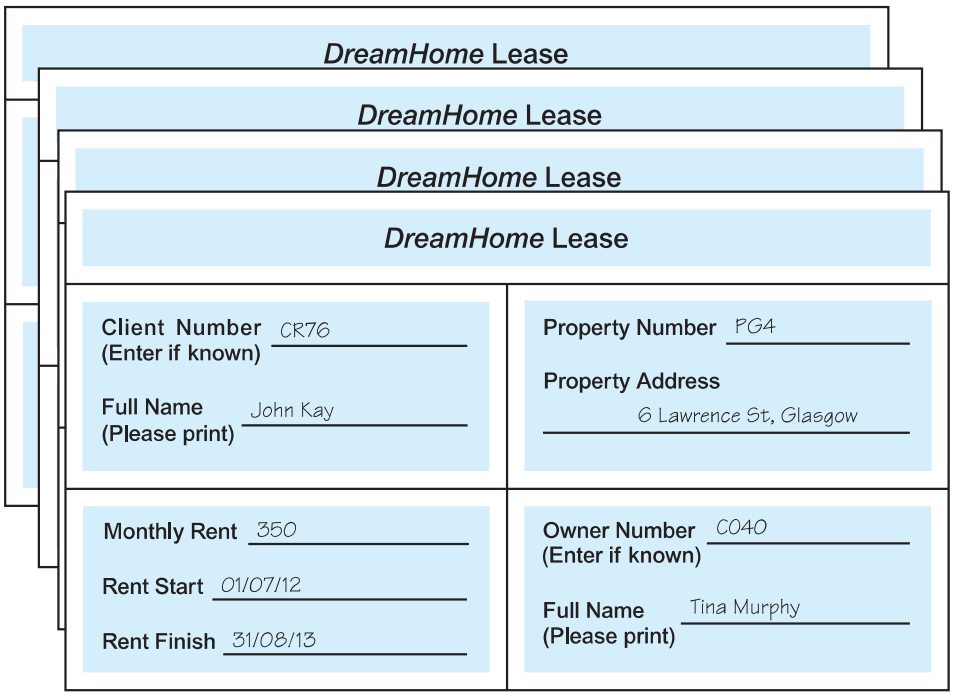

Figure 14.9 Collection of (simplified) *DreamHome* leases.

Sample data is taken from two leases for two different clients called John Kay and Aline Stewart and is transformed into table format with rows and columns, as shown in Figure 14.10. This is an example of an unnormalized table.

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.10 ClientRental unnormalized table.

We identify the key attribute for the ClientRental unnormalized table as clientNo. Next, we identify the repeating group in the unnormalized table as the property rented details, which repeats for each client. The structure of the repeating group is:

Repeating Group = (propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

As a consequence, there are multiple values at the intersection of certain rows and columns. For example, there are two values for propertyNo (PG4 and PG16) for the client named John Kay. To transform an unnormalized table into 1NF, we ensure that there is a single value at the intersection of each row and column. This is achieved by removing the repeating group.

With the first approach, we remove the repeating group (property rented details) by entering the appropriate client data into each row. The resulting first normal form ClientRental relation is shown in Figure 14.11.

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.11 First Normal Form ClientRental relation.

In Figure 14.12, we present the functional dependencies (fdl to fd6) for the ClientRental relation. We use the functional dependencies (as discussed in Section 14.4.3) to identify candidate keys for the ClientRental relation as being composite keys comprising (clientNo, propertyNo), (clientNo, rentStart), and (propertyNo, rentStart). We select (clientNo, propertyNo) as the primary key for the relation, and for clarity we place the attributes that make up the primary key together at the left-hand side of the relation. In this example, we assume that the rentFinish attribute is not appropriate as a component of a candidate key as it may contain nulls (see Section 4.3.1).

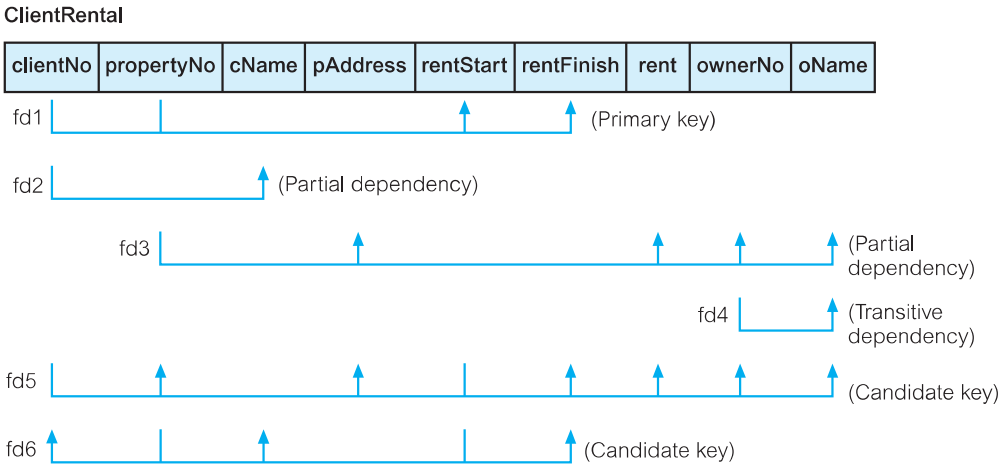

Figure 14.12 Functional dependencies of the ClientRental relation.

The ClientRental relation is defined as follows:

ClientRental (clientNo, propertyNo, cName, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

The ClientRental relation is in 1NF, as there is a single value at the intersection of each row and column. The relation contains data describing clients, property rented, and property owners, which is repeated several times. As a result, the ClientRental relation contains significant data redundancy. If implemented, the 1NF relation would be subject to the update anomalies described in Section 14.3. To remove some of these, we must transform the relation into second normal form, which we discuss shortly.

With the second approach, we remove the repeating group (property rented details) by placing the repeating data along with a copy of the original key attribute (clientNo) in a separate relation, as shown in Figure 14.13.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.13 Alternative 1NF Client and PropertyRental-Owner relations.

With the help of the functional dependencies identified in Figure 14.12 we identify a primary key for the relations. The format of the resulting 1NF relations are as follows:

Client (clientNo, cName)
PropertyRentalOwner (clientNo, propertyNo, pAddress, rentStart, rentFinish, rent, ownerNo, oName)

The Client and PropertyRentalOwner relations are both in 1NF, as there is a single value at the intersection of each row and column. The Client relation contains data describing clients and the PropertyRentalOwner relation contains data describing property rented by clients and property owners. However, as we see from Figure 14.13, this relation also contains some redundancy and as a result may suffer from similar update anomalies to those described in Section 14.3.

To demonstrate the process of normalizing relations from 1NF to 2NF, we use only the ClientRental relation shown in Figure 14.11. However, recall that both approaches are correct, and will ultimately result in the production of the same relations as we continue the process of normalization. We leave the process of completing the normalization of the Client and PropertyRentalOwner relations as an exercise for the reader, which is given at the end of this chapter.

14.7 Second Normal Form (2NF)

Second Normal Form (2NF) is based on the concept of full functional dependency, which we described in Section 14.4. Second normal form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies discussed in Section 14.3. For example, suppose we wish to change the rent of property number PG4. We have to update two tuples in the ClientRental relation in Figure 14.11. If only one tuple is updated with the new rent, this results in an inconsistency in the database.

Second Normal Form (2NF) A relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the primary key.

The normalization of 1NF relations to 2NF involves the removal of partial dependencies. If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them in a new relation along with a copy of their determinant. We demonstrate the process of converting 1NF relations to 2NF relations in the following example.

EXAMPLE 14.10 Second Normal Form (2NF)

As shown in Figure 14.12, the ClientRental relation has the following functional dependencies:

fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd2	clientNo ® cName	(Partial dependency)
fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Partial dependency)
fd4	ownerNo ® oName	(Transitive dependency)
fd5	clientNo, rentStart ® propertyNo, pAddress, rentFinish, rent, ownerNo, oName	(Candidate key)
fd6	propertyNo, rentStart ® clientNo, cName, rentFinish	(Candidate key)

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Figure 14.14 Second normal form relations derived from the ClientRental relation.

Using these functional dependencies, we continue the process of normalizing the ClientRental relation. We begin by testing whether the ClientRental relation is in 2NF by identifying the presence of any partial dependencies on the primary key. We note that the client attribute (cName) is partially dependent on the primary key, in other words, on only the clientNo attribute (represented as fd2). The property attributes (pAddress, rent, ownerNo, oName) are partially dependent on the primary key, that is, on only the propertyNo attribute (represented as fd3). The property rented attributes (rentStart and rentFinish) are fully dependent on the whole primary key; that is the clientNo and propertyNo attributes (represented as fd1).

The identification of partial dependencies within the ClientRental relation indicates that the relation is not in 2NF. To transform the ClientRental relation into 2NF requires the creation of new relations so that the non-primary-key attributes are removed along with a copy of the part of the primary key on which they are fully functionally dependent. This results in the creation of three new relations called Client, Rental, and PropertyOwner, as shown in Figure 14.14. These three relations are in second normal form, as every non-primary-key attribute is fully functionally dependent on the primary key of the relation. The relations have the following form:

Client	(clientNo, cName)
Rental	(clientNo, propertyNo, rentStart, rentFinish)
PropertyOwner	(propertyNo, pAddress, rent, ownerNo, oName)

14.8 Third Normal Form (3NF)

Although 2NF relations have less redundancy than those in 1NF, they may still suffer from update anomalies. For example, if we want to update the name of an owner, such as Tony Shaw (ownerNo CO93), we have to update two tuples in the PropertyOwner relation of Figure 14.14. If we update only one tuple and not the other, the database would be in an inconsistent state. This update anomaly is caused by a transitive dependency, which we described in Section 14.4. We need to remove such dependencies by progressing to third normal form.

Third Normal Form (3NF) A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on the primary key.

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant. We demonstrate the process of converting 2NF relations to 3NF relations in the following example.

EXAMPLE 14.11 Third Normal Form (3NF)

The functional dependencies for the Client, Rental, and PropertyOwner relations, derived in Example 14.10, are as follows:

Client

fd2	clientNo ® cName	(Primary key)

Rental

fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd5′	clientNo, rentStart ® propertyNo, rentFinish	(Candidate key)
fd6′	propertyNo, rentStart ® clientNo, rentFinish	(Candidate key)

PropertyOwner

fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Primary key)
fd4	ownerNo ® oName	(Transitive dependency)

All the non-primary-key attributes within the Client and Rental relations are functionally dependent on only their primary keys. The Client and Rental relations have no transitive dependencies and are therefore already in 3NF. Note that where a functional dependency (fd) is labeled with a prime (such as fd5′), this indicates that the dependency has altered compared with the original functional dependency shown in Figure 14.12.

All the non-primary-key attributes within the PropertyOwner relation are functionally dependent on the primary key, with the exception of oName, which is transitively dependent on ownerNo (represented as fd4). This transitive dependency was previously identified in Figure 14.12. To transform the PropertyOwner relation into 3NF, we must first remove this transitive dependency by creating two new relations called PropertyForRent and Owner, as shown in Figure 14.15. The new relations have the following form:

PropertyForRent (propertyNo, pAddress, rent, ownerNo)
Owner (ownerNo, oName)

The PropertyForRent and Owner relations are in 3NF, as there are no further transitive dependencies on the primary key.

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Figure 14.15 Third normal form relations derived from the PropertyOwner relation.

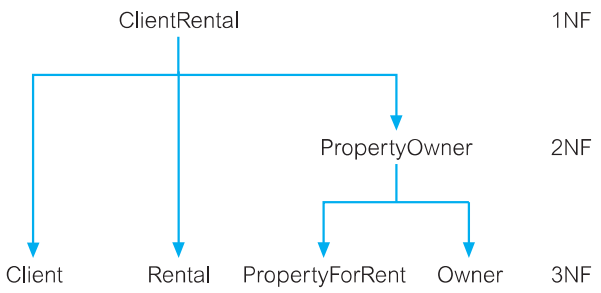

Figure 14.16 The decomposition of the ClientRental 1NF relation into 3NF relations.

The ClientRental relation shown in Figure 14.11 has been transformed by the process of normalization into four relations in 3NF. Figure 14.16 illustrates the process by which the original 1NF relation is decomposed into the 3NF relations. The resulting 3NF relations have the form:

Client	(clientNo, cName)
Rental	(clientNo, propertyNo, rentStart, rentFinish)
PropertyForRent	(propertyNo, pAddress, rent, ownerNo)
Owner	(ownerNo, oName)

The original ClientRental relation shown in Figure 14.11 can be recreated by joining the Client, Rental, PropertyForRent, and Owner relations through the primary key/foreign key mechanism. For example, the ownerNo attribute is a primary key within the Owner relation and is also present within the PropertyForRent relation as a foreign key. The ownerNo attribute acting as a primary key/foreign key allows the association of the PropertyForRent and Owner relations to identify the name of property owners.

The clientNo attribute is a primary key of the Client relation and is also present within the Rental relation as a foreign key. Note that in this case the clientNo attribute in the Rental relation acts both as a foreign key and as part of the primary key of this relation. Similarly, the propertyNo attribute is the primary key of the PropertyForRent relation and is also present within the Rental relation acting both as a foreign key and as part of the primary key for this relation.

In other words, the normalization process has decomposed the original ClientRental relation using a series of relational algebra projections (see Section 5.1). This results in a **lossless-join** (also called *nonloss-* or *nonadditive*-join) decomposition, which is reversible using the natural join operation. The Client, Rental, PropertyForRent, and Owner relations are shown in Figure 14.17.

14.9 General Definitions of 2NF and 3NF

The definitions for 2NF and 3NF given in Sections 14.7 and 14.8 disallow partial or transitive dependencies on the *primary key* of relations to avoid the update anomalies described in Section 14.3. However, these definitions do not take into account other candidate keys of a relation, if any exist. In this section, we present more general definitions for 2NF and 3NF that take into account candidate keys of a relation. Note that this requirement does not alter the definition for 1NF as this normal form is independent of keys and functional dependencies. For the general definitions, we state that a candidate-key attribute is part of any candidate key and

Figure 14.17
A summary of the 3NF relations derived from the ClientRental relation.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

that partial, full, and transitive dependencies are with respect to all candidate keys of a relation.

Second Normal Form (2NF)

A relation that is in first normal form and every non-candidate-key attribute is fully functionally dependent on *any candidate key*.

Third Normal Form (3NF)

A relation that is in first and second normal form and in which no non-candidate-key attribute is transitively dependent on *any candidate key*.

When using the general definitions of 2NF and 3NF, we must be aware of partial and transitive dependencies on all candidate keys and not just the primary key. This can make the process of normalization more complex; however, the general definitions place additional constraints on the relations and may identify hidden redundancy in relations that could be missed.

The trade-off is whether it is better to keep the process of normalization simpler by examining dependencies on primary keys only, which allows the identification of the most problematic and obvious redundancy in relations, or to use the general definitions and increase the opportunity to identify missed redundancy. In fact, it is often the case that whether we use the definitions based on primary keys or the general definitions of 2NF and 3NF, the decomposition of relations is the same. For example, if we apply the general definitions of 2NF and 3NF to Examples 14.10 and 14.11 described in Sections 14.7 and Section 14.8, the same decomposition of the larger relations into smaller relations results. The reader may wish to verify this fact.

In the following chapter we re-examine the process of identifying functional dependencies that are useful for normalization and take the process of normalization further by discussing normal forms that go beyond 3NF such as BCNF. Also in this chapter we present a second worked example taken from the *DreamHome* case study that reviews the process of normalization from UNF through to BCNF.

Chapter Summary

- **Normalization** is a technique for producing a set of relations with desirable properties, given the data requirements of an enterprise. Normalization is a formal method that can be used to identify relations based on their keys and the functional dependencies among their attributes.
- Relations with data redundancy suffer from **update anomalies**, which can be classified as insertion, deletion, and modification anomalies.
- One of the main concepts associated with normalization is **functional dependency**, which describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A ® B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes.)
- The **determinant** of a functional dependency refers to the attribute, or group of attributes, on the left-hand side of the arrow.
- The main characteristics of functional dependencies that we use for normalization have a one-to-one relationship between attribute(s) on the left-hand and right-hand sides of the dependency, hold for all time, and are fully functionally dependent.
- **Unnormalized Form** (**UNF**) is a table that contains one or more repeating groups.
- **First Normal Form** (**1NF**) is a relation in which the intersection of each row and column contains one and only one value.
- **Second Normal Form** (**2NF**) is a relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on the *primary key*. **Full functional dependency** indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A but not on any proper subset of A.
- **Third Normal Form** (**3NF**) is a relation that is in first and second normal form in which no non-primary- key attribute is transitively dependent on the *primary key*. **Transitive dependency** is a condition where A, B, and C are attributes of a relation such that if A ® B and B ® C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).
- **General definition for Second Normal Form** (**2NF**) is a relation that is in first normal form and every non-candidate-key attribute is fully functionally dependent on *any candidate key*. In this definition, a candidate-key attribute is part of any candidate key.
- **General definition for Third Normal Form** (**3NF**) is a relation that is in first and second normal form in which no non-candidate-key attribute is transitively dependent on *any candidate key*. In this definition, a candidate-key attribute is part of any candidate key.

Review Questions

14.1 Describe the purpose of normalizing data.

14.2 Discuss the alternative ways that normalization can be used to support database design.

14.3 How does normalization eradicate update anomalies from a relation?

14.4 Describe the concept of functional dependency.

14.5 What are the main characteristics of functional dependencies that are used for normalization?

14.6 Describe how a database designer typically identifies the set of functional dependencies associated with a relation.

14.7 Describe factors that would influence the choice of normalization or ER modeling when designing a database.

14.8 Why is normalization regarded as a bottom-up design approach? How does it differ from ER modeling?

14.9 Describe the two approaches to converting an UNF table to 1NF relation(s).

14.10 The second normal form (2NF) is realized by removing partial dependencies from 1NF relations. Briefly describe the term "partial dependency."

14.11 Describe the concept of transitive dependency and describe how this concept relates to 3NF. Provide an example to illustrate your answer.

14.12 Discuss how the definitions of 2NF and 3NF based on primary keys differ from the general definitions of 2NF and 3NF. Provide an example to illustrate your answer.

Exercises

14.13 Normalization is an important concept for database professionals. Whether you are the designer, database analyst or administrator, it is useful for designing, situation verification as well as performance tuning. What are the basic issues to be aware of before carrying out the normalization process?.

14.14 Examine the Patient Medication Form for the *Wellmeadows Hospital* case study (see Appendix B) shown in Figure 14.18.

(a) Identify the functional dependencies represented by the attributes shown in the form in Figure 14.18. State any assumptions that you make about the data and the attributes shown in this form.

(b) Describe and illustrate the process of normalizing the attributes shown in Figure 14.18 to produce a set of well-designed 3NF relations.

(c) Identify the primary, alternate, and foreign keys in your 3NF relations.

Wellmeadows Hospital
Patient Medication Form

Patient Number: P10034

Full Name: Robert MacDonald **Ward Number:** Ward 11

Bed Number: 84 **Ward Name:** Orthopaedic

Drug Number	Name	Description	Dosage	Method of Admin	Units per Day	Start Date	Finish Date
10223	Morphine	Pain Killer	10mg/ml	Oral	50	24/03/13	24/04/14
10334	Tetracyclene	Antibiotic	0.5mg/ml	IV	10	24/03/13	17/04/13
10223	Morphine	Pain Killer	10mg/ml	Oral	10	25/04/14	02/05/15

Figure 14.18 The *Wellmeadows Hospital* Patient Medication Form.

14.15 The table shown in Figure 14.19 lists sample dentist/patient appointment data. A patient is given an appointment at a specific time and date with a dentist located at a particular surgery. On each day of patient appointments, a dentist is allocated to a specific surgery for that day.

(a) The table shown in Figure 14.19 is susceptible to update anomalies. Provide examples of insertion, deletion, and update anomalies.

(b) Identify the functional dependencies represented by the attributes shown in the table of Figure 14.19. State any assumptions you make about the data and the attributes shown in this table.

(c) Describe and illustrate the process of normalizing the table shown in Figure 14.19 to 3NF relations. Identify the primary, alternate, and foreign keys in your 3NF relations.

staffNo	dentistName	patNo	patName	appointment date	appointment time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-13	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-13	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-13	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-13	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-13	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-13	18.00	S13

Figure 14.19 Table displaying sample dentist/patient appointment data.

14.16 An agency called *Instant Cover* supplies part-time/temporary staff to hotels within Scotland. The table shown in Figure 14.20 displays sample data, which lists the time spent by agency staff working at various hotels. The National Insurance Number (NIN) is unique for every member of staff.

(a) The table shown in Figure 14.20 is susceptible to update anomalies. Provide examples of insertion, deletion, and update anomalies.

(b) Identify the functional dependencies represented by the attributes shown in the table of Figure 14.20. State any assumptions that you make about the data and the attributes shown in this table.

(c) Describe and illustrate the process of normalizing the table shown in Figure 14.20 to 3NF. Identify primary, alternate, and foreign keys in your relations.

NIN	contractNo	hours	eName	hNo	hLoc
1135	C1024	16	Smith J	H25	East Kilbride
1057	C1024	24	Hocine D	H25	East Kilbride
1068	C1025	28	White T	H4	Glasgow
1135	C1025	15	Smith J	H4	Glasgow

Figure 14.20 Table displaying sample data for the *Instant Cover* agency.

14.17 A company called *FastCabs* provides a taxi service to clients. The table shown in Figure 14.21 displays some details of client bookings for taxis. Assume that a taxi driver is assigned to a single taxi, but a taxi can be assigned to one or more drivers.

(a) Identify the functional dependencies that exist between the columns of the table in Figure 14.21 and identify the primary key and any alternate key(s) (if present) for the table.

(b) Describe why the table in Figure 14.21 is not in 3NF.

(c) The table shown in Figure 14.21 is susceptible to update anomalies. Provide examples of how insertion, deletion, and modification anomalies could occur on this table.

JobID	JobDate Time	driverID	driver Name	taxiID	clientID	clientName	jobPickUpAddress
1	25/07/14 10.00	D1	Joe Bull	T1	C1	Anne Woo	1 Storrie Rd, Paisley
2	29/07/14 10.00	D1	Joe Bull	T1	C1	Anne Woo	1 Storrie Rd, Paisley
3	30/07/14 11.00	D2	Tom Win	T2	C1	Anne Woo	3 High Street, Paisley
4	2/08/14 13.00	D3	Jim Jones	T3	C2	Mark Tin	1A Lady Lane, Paisley
5	2/08/14 13.00	D4	Steven Win	T1	C3	John Seal	22 Red Road, Paisley
6	25/08/14 10.00	D2	Tom Win	T2	C4	Karen Bow	17 High Street, Paisley

Figure 14.21 Table displaying sample data for *FastCabs*.

14.18 Applying normalisation to 3NF on the table shown in Figure 14.21 results in the formation of the three 3NF tables shown in Figure 14.22.

(a) Identify the functional dependencies that exist between the columns of each table in Figure 14.22 and identify the primary key and any alternate and foreign key(s) (if present) for each table.

(b) Describe why storing the *FastCabs* data across three 3NF tables avoids the update anomalies described in Exercise 14.17(b).

(c) Describe how the original table shown in Figure 14.21 can be re-created through relational joins between primary key and foreign keys columns of the tables in Figure 14.22.

JobID	JobDateTime	driverID	clientID	jobPickUpAddress
1	25/07/14 10.00	D1	C1	1 Storrier Rd, Paisley
2	29/07/14 10.00	D1	C1	1 Storrier Rd, Paisley
3	30/07/14 11.00	D2	C1	3 High Street, Paisley
4	2/08/14 13.00	D3	C2	1A Lady Lane, Paisley
5	2/08/14 13.00	D4	C3	22 Red Road, Paisley
6	25/08/14 10.00	D2	C4	17 High Street, Paisley

drierID	driverName	taxiD
D1	Joe Bull	T1
D2	Tom Win	T2
D3	Jim Jones	T3
D4	Steven Win	T1

clientID	clientName
C1	Anne Woo
C2	Mark Tin
C3	John Seal
C4	Karen Bow

Figure 14.22 Tables (in 3NF) displaying sample data for *FastCabs*.

14.19 Students can lease university flats and some of the details of leases held by students for places in university flats are shown in Figure 14.23. A place number (placeNo) uniquely identifies each single room in all flats and is used when leasing a room to a student.

(a) Identify the functional dependencies that exist between the columns of the table in Figure 14.23 and identify the primary key and any alternate key(s) (if present) for the table.

(b) Describe why the table in Figure 14.23 is not in 3NF.

(c) The table shown in Figure 14.23 is susceptible to update anomalies. Provide examples of how insertion, deletion, and modification anomalies could occur on this table.