Chapter

Normalization

ıapter Objectives

ıhis 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, namely 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.

hen we design a database for an enterprise, the main objective is to create an accurate presentation of the data, relationships between the data, and constraints on the data at is pertinent to the enterprise. To help achieve this objective, we can use one or ore database design techniques. In Chapters 11 and 12 we described a technique called ntity–Relationship (ER) modeling. In this chapter and the next we describe another atabase design technique called **normalization**.

Normalization is a database design technique, which begins by examining the relationips (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 upports the data requirements of the enterprise.

While 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 14 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 13.1 we describe the purpose of normalization. In Section 13.2 we discuss how normalization can be used to support relational database design. In Section 13.3 we identify and illustrate the potential problems associated with data redundancy in a base relation that is not normalized. In Section 13.4 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 13.5 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 13.6, Second Normal Form (2NF) in Section 13.7, and Third Normal Form (3NF) in Section 13.8. The 2NF and 3NF described in these sections are based on the *primary key* of a relation. In Section 13.9 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 10.4 and documented in Appendix A.

13.1 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 3.2.5), 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 compute
ately normalized is de

How Norm
Database I

Normalization is a fo
However, in this sect
illustrated in Figure 1
up standalone databas
be used as a validatio
created using a top-d
used the goal is the sa
requirements of the en

Figure 13.1 shows
Although, the users'
source, it is possible
other data sources su

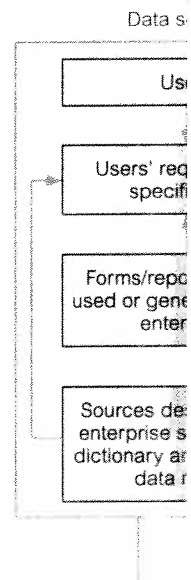


Figure 13.1 How norm

space on the computer. The problems associated with using a relation that is not appropriately normalized is described later in Section 13.3.

How Normalization Supports Database Design

13.2

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 13.1. Approach 1 shows how normalization can be used as a bottom-up standalone database design technique while 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 that of creating a set of well-designed relations that meet the data requirements of the enterprise.

Figure 13.1 shows examples of data sources that can be used for database design. Although, the users' requirements specification (see Section 9.5) 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.

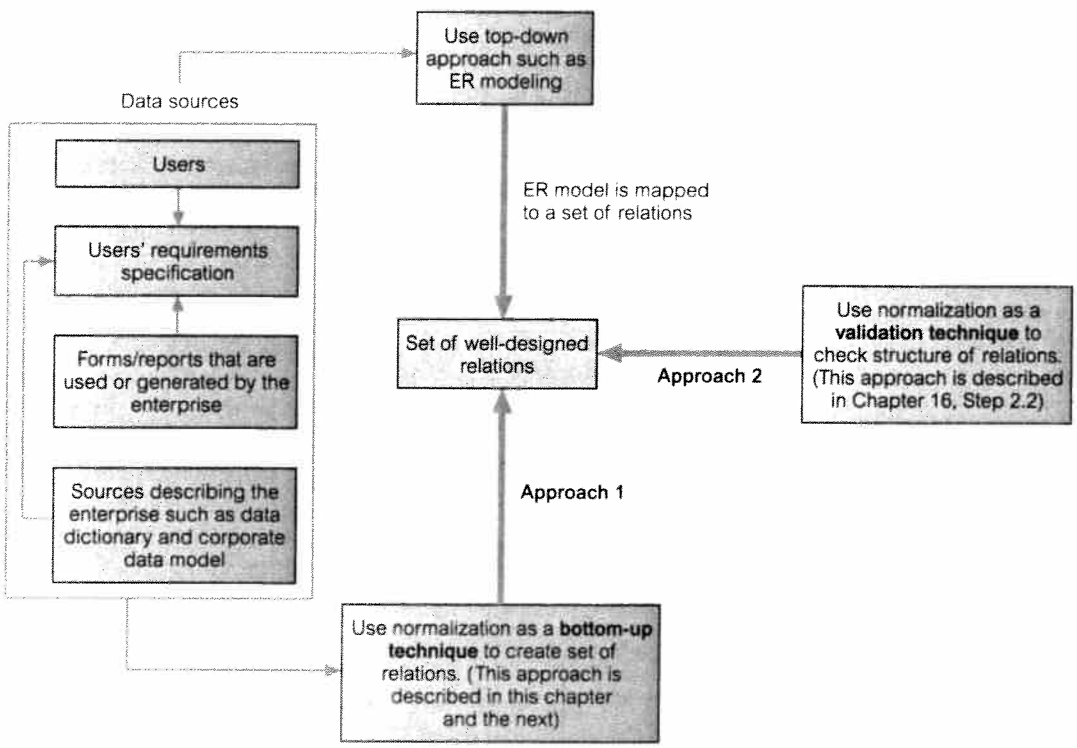


Figure 13.1 How normalization can be used to support database design.

Figure 13.1 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 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.

13.3 Data Redundancy and Update Anomalies

As stated in Section 13.1 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 13.2 with the StaffBranch relation

Figure 13.2 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 13.3 StaffBranch relation.

sh… in Figure 13.3. The StaffBranch relation is an alternative format of the Staff and … relations. The relations have the form:

…aff (staffNo, sName, position, salary, branchNo)
…anch (branchNo, bAddress)
…affBranch (staffNo, sName, position, salary, branchNo, bAddress)

N… that the primary key for each relation is underlined.

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

13.3.1 …rtion Anomalies

T…re are two main types of insertion anomaly, which we illustrate using the StaffBranch r…ation shown in Figure 13.3.

- …o insert the details of new members of staff into the StaffBranch relation, we must …clude the details of the branch at which the staff are to be located. For example, … insert the details of new staff located at branch number B007, we must enter the …orrect details of branch number B007 so that the branch details are consistent with …alues for branch B007 in other tuples of the StaffBranch relation. The relations shown …n Figure 13.2 do not suffer from this potential inconsistency because we enter only …he appropriate branch number for each staff member in the Staff relation. Instead, the …etails of branch number B007 are recorded in the database as a single tuple in the …anch relation.
- To insert details of a new branch that currently has no members of staff into the …taffBranch relation, it is necessary to enter nulls into the attributes for staff, such as …affNo. However, as staffNo is the primary key for the StaffBranch relation, attempting to …nter nulls for staffNo violates entity integrity (see Section 3.3), 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 13.2 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.

13.3.2 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 13.2 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.

13.3.3 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 above examples illustrate that the Staff and Branch relations of Figure 13.2 have more desirable properties than the StaffBranch relation of Figure 13.3. This demonstrates that while 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.

13.4 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 dependencie
dependencies that are u
dencies can be identifie

Characteristi

For the discussion on
attributes (A, B, C, ...
tion called R = (A, B, C, .
has a unique name.

Functional dependency Desc
exam
depe
with
more

Functional dependenc
relation. The semantics i
tional dependencies bet
dependency is specified
Consider a relation w
ent on attribute A. If we
dependency, we find onl
any moment in time. Th
same value of B. Howev
A. The dependency betw
shown Figure 13.4.
An alternative way to
'A functionally determin
ally follows the directio

Determinant Refe
the a

When a functional de
hand side of the arrow i
determinant of B. We d
following example.

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 use to identify the primary key for a relation.

13.4.1 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.

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 13.4.

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 13.4, A is the determinant of B. We demonstrate the identification of a functional dependency in the following example.

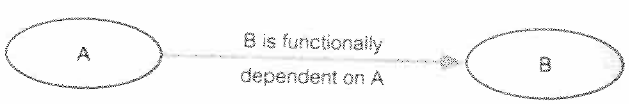


Figure 13.4 A functional dependency diagram.

Example 13.1 An example of a functional dependency

Consider the attributes staffNo and position of the Staff relation in Figure 13.2. 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 13.5(a). However, Figure 13.5(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 3.2.1). This point is illustrated in the following example.

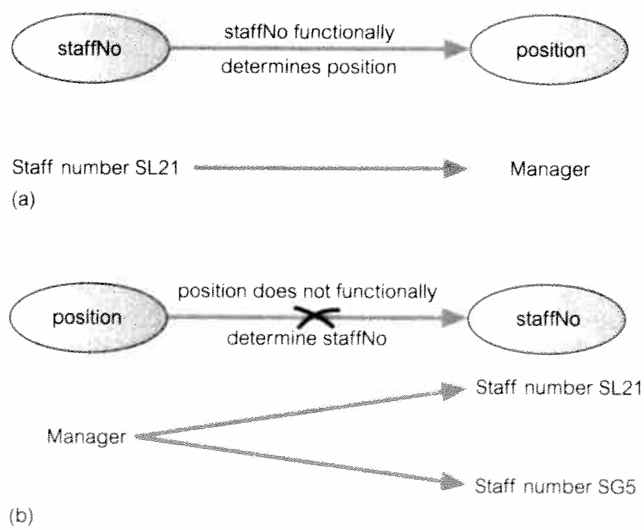


Figure 13.5 (a) staffNo functionally determines position (staffNo → position); (b) position does *not* functionally determine staffNo (position ↛ staffNo).

Example 13.2 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 13.2. We see that for a specific staffNo, for example SL21, 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 13.2 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 13.2 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 for some members of staff in this category we would not be able to determine their staff number (staffNo). 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 right hand-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 **partially 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 13.3.

Example 13.3 Example of a full functional dependency

Consider the following functional dependency that exists in the Staff relation of Figure 13.2:

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 above 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 below.

staffNo → branchNo

Additional examples of partial and full functional dependencies are discussed in Section 13.7.

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- 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- 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 13.3. 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 13.4.

Example 13.4 Example of a transitive functional dependency

Consider the following functional dependencies within the StaffBranch relation shown in Figure 13.3:

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 13.8.

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.

13.4.2 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 13.5 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 13.5 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 13.3. 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 13.6.

Example 13.6 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 fd4) that exist between the attributes of the Sample relation shown in Figure 13.6 is described below.

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

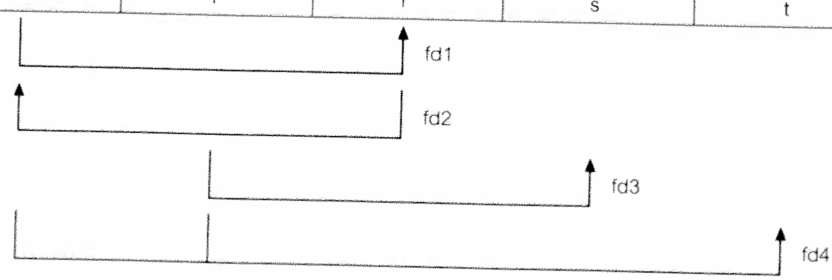


Figure 13.6 The Sample relation displaying data for attributes A, B, C, D, and E and the functional dependencies (fd1 to fd4) that exist between these attributes.

To identify the functional dependencies that exist between attributes A, B, C, D, and E, we examine the Sample relation shown in Figure 13.6 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.

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 13.6. 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 13.6. 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 13.6. 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 13.6. 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 complete the examination of the relation shown in Figure 13.6 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 13.6 as follows:

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)

13.4.3 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 13.7 Identifying the primary key for the StaffBranch relation

In Example 13.5 we describe the identification of five functional dependencies for the StaffBranch relation shown in Figure 13.3. 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 3.2.5). 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 13.8 Identifying the primary key for the Sample relation

In Example 13.6 we identified four 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, and (A, B). However, the only determinant that functionally determines all the other attributes of the relation is (A, B). In particular, 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 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. As there are no other candidate keys for the Sample relation (A, B) is identified as the primary key for this 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.

13.5 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 discussions on 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 13.3, it is generally recommended that we proceed to at least Third Normal Form (3NF). Figure 13.7 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 13.8 provides an overview of the process and highlights the main actions 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,

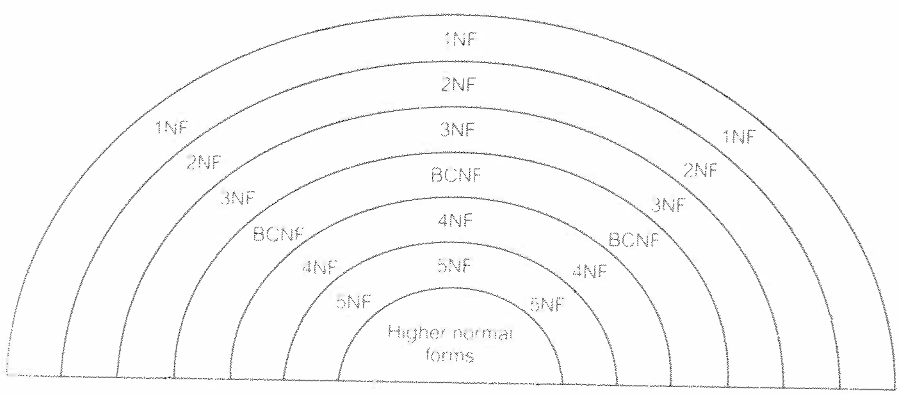


Figure 13.7 Diagrammatic illustration of the relationship between the normal forms.

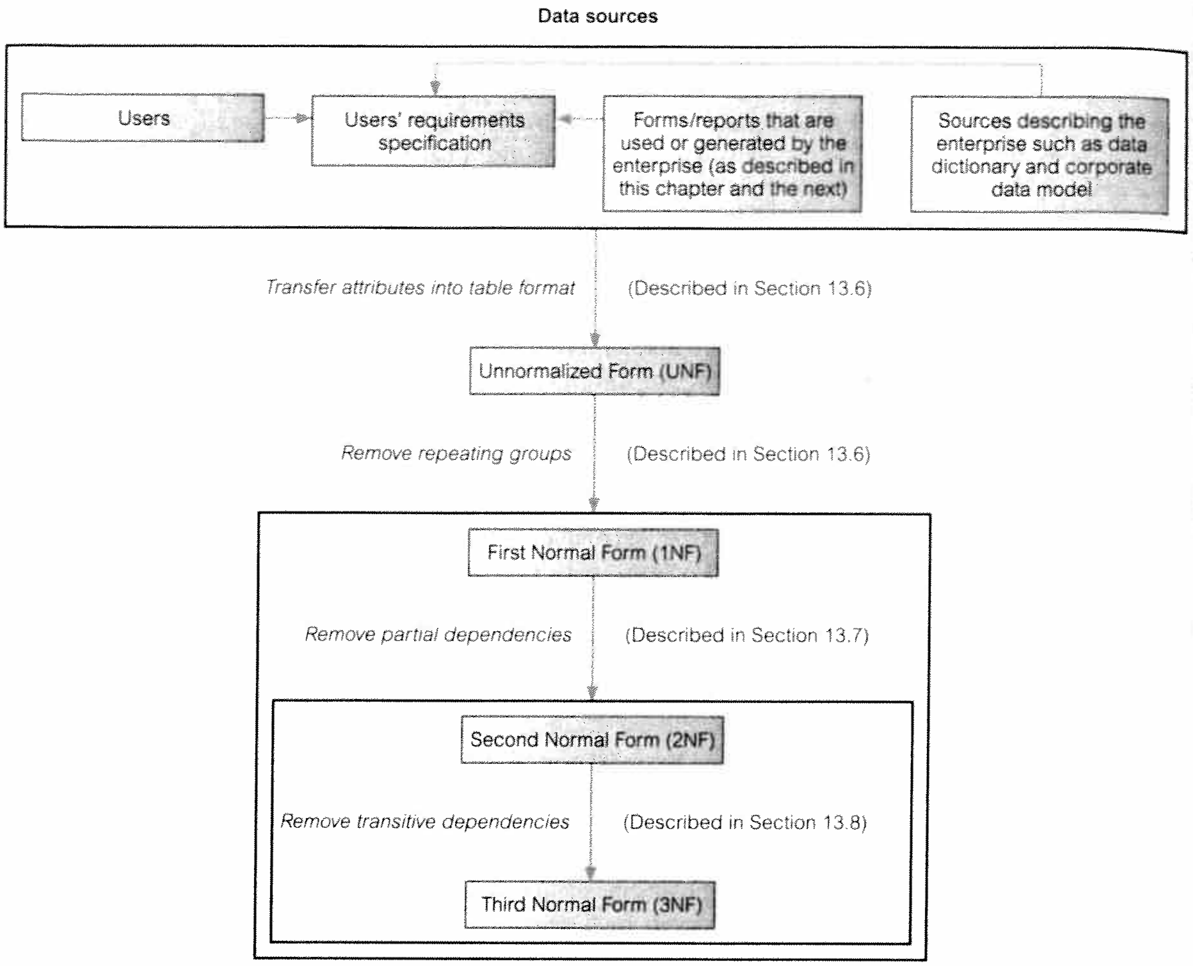


Figure 13.8 Diagrammatic illustration of the process of normalization.

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 13.6 we begin by describing First Normal Form (1NF). In Sections 13.7 and 13.8 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 13.9. The more general definitions of 2NF and 3NF take into account all *candidate keys* of a relation rather than just the primary key.

13.6 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.

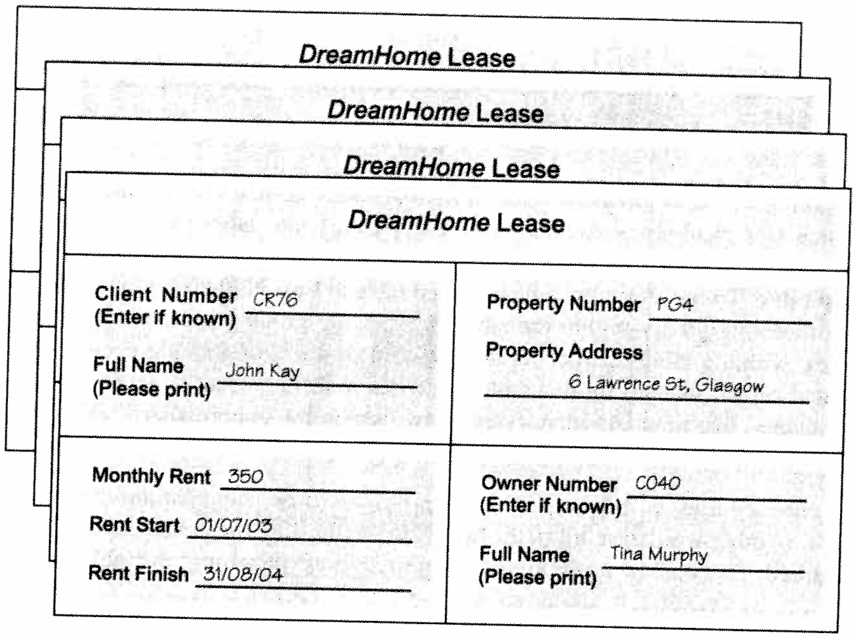
For both approaches, the resulting tables are now referred to as 1NF relations containing atomic (or single) values at the intersection of each row and column. Although both approaches are correct, approach 1 introduces more redundancy into the original UNF table as part of the 'flattening' process, 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 13.9 First Normal Form (1NF)

A collection of (simplified) *DreamHome* leases is shown in Figure 13.9. 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.

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 13.10. This is an example of an unnormalized table.


DreamHome Lease

Client Number (Enter if known) CR76	Property Number PG4
Full Name (Please print) John Kay	Property Address 6 Lawrence St, Glasgow
Monthly Rent 350	Owner Number (Enter if known) CO40
Rent Start 01/07/03	Full Name (Please print) Tina Murphy
Rent Finish 31/08/04	

Figure 13.9 Collection of (simplified) *DreamHome* leases.

Figure 13.10 ClientRental unnormalized table.

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

We identify the
identify the repeati
repeats for each cli

Repeating Gro

As a consequence
columns. For exam
named John Kay. T
single value at the i
repeating group.

With the first ap
entering the appro
ClientRental relation

In Figure 13.12,
relation. We use the
candidate keys for

ClientRental

clientNo	propertyNo
CR76	PG4
CR76	PG16
CR56	PG4
CR56	PG36
CR56	PG16

ClientRental

clientNo | propertyNo

fd1

fd2

fd3

fd5

fd6

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

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

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

With the first approach, we remove the repeating group (property rented details) by ıtering the appropriate client data into each row. The resulting first normal form ientRental relation is shown in Figure 13.11.

In Figure 13.12, we present the functional dependencies (fd1 to fd6) for the ClientRental ·lation. We use the functional dependencies (as discussed in Section 13.4.3) to identify ındidate keys for the ClientRental relation as being composite keys comprising (clientNo,

lientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Figure 13.11
First Normal Form ClientRental relation.

ClientRental

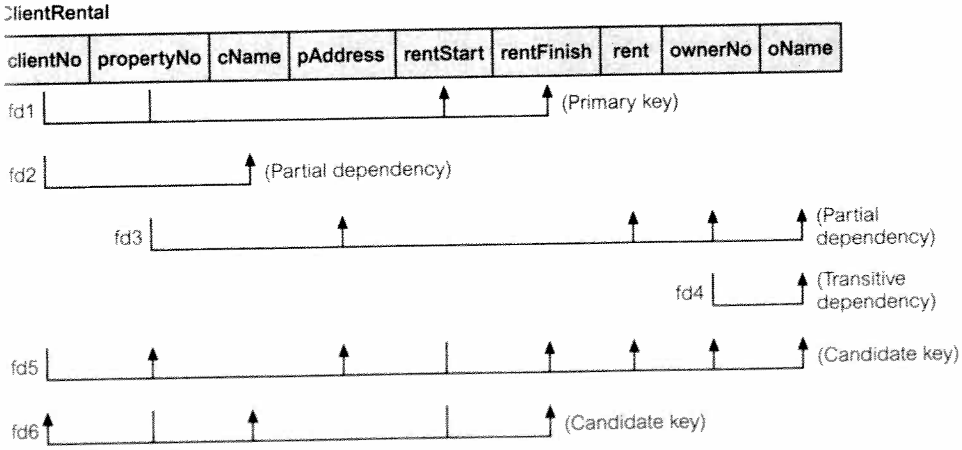


Figure 13.12
Functional dependencies of the ClientRental relation.

Figure 13.13
Alternative 1NF Client and PropertyRentalOwner relations.

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-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

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 3.3.1).

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 13.3. 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 13.13.

With the help of the functional dependencies identified in Figure 13.12 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 13.13, this relation also contains some redundancy and as a result may suffer from similar update anomalies to those described in Section 13.3.

To demonstrate the process of normalizing relations from 1NF to 2NF, we use only the ClientRental relation shown in Figure 13.11. 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.

13.7 Second Normal Form (2NF)

Second Normal Form (2NF) is based on the concept of full functional dependency, which we described in Section 13.4. 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 13.3. 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 13.11. 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 13.10 Second Normal Form (2NF)

As shown in Figure 13.12, 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)

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

Figure 13.14
Second Normal Form relations derived from the ClientRental relation.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

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

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 13.14. 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	(<u>clientNo</u>, cName)
Rental	(<u>clientNo</u>, <u>propertyNo</u>, rentStart, rentFinish)
PropertyOwner	(<u>propertyNo</u>, pAddress, rent, ownerNo, oName)

13.8 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 13.14. 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 13.4. 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 13.11 Third Normal Form (3NF)

The functional dependencies for the Client, Rental, and PropertyOwner relations, derived in Example 13.10, 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 13.12.

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 13.12. 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 13.15. The new relations have the 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.

Figure 13.15 Third Normal Form relations derived from the PropertyOwner relation.

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

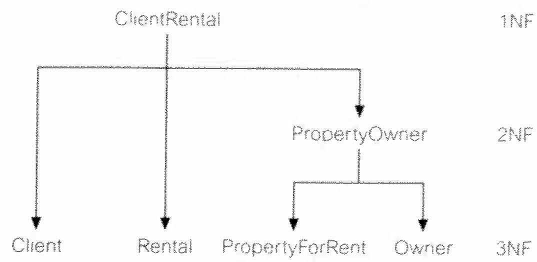


Figure 13.16 The decomposition of the ClientRental 1NF relation into 3NF relations.

The ClientRental relation shown in Figure 13.11 has been transformed by the process of normalization into four relations in 3NF. Figure 13.16 illustrates the process by which the original 1NF relation is decomposed into the 3NF relations. The resulting 3NF relations have the form:

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

The original ClientRental relation shown in Figure 13.11 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 in this case that 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 4.1). 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 13.17.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

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

13.9 General Definitions of 2NF and 3NF

The definitions for 2NF and 3NF given in Sections 13.7 and 13.8 disallow partial or transitive dependencies on the *primary key* of relations to avoid the update anomalies described in Section 13.3. 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 define that a candidate-key attribute is part of any candidate key and 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 tradeoff 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 13.10 and 13.11 described in Sections 13.7 and 13.8, 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 Boyce–Codd Normal Form (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- 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 Q

13.1 Describe
13.2 Discuss tl
can be us
13.3 Describe
may occu
data.
13.4 Describe
dependen
13.5 What are
dependen
13.6 Describe
identifies
associate
13.7 Describe
Unnorma
such a ta
Form (1N

Exercises

13.13 Continue
to 3NF re
duced fro
13.14 Examine
(a) Identi
State
(b) Descr
of we
(c) Identi
13.15 The table
ment at a
ments, a c
(a) The
deleti
(b) Ident
State
(c) Desc
Ident
13.16 An agenc
in Figure
The Nati

Review Questions

13.1 Describe the purpose of normalizing data.

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

13.3 Describe the types of update anomaly that may occur on a relation that has redundant data.

13.4 Describe the concept of functional dependency.

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

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

13.7 Describe the characteristics of a table in Unnormalized Form (UNF) and describe how such a table is converted to a First Normal Form (1NF) relation.

13.8 What is the minimal normal form that a relation must satisfy? Provide a definition for this normal form.

13.9 Describe the two approaches to converting an Unnormalized Form (UNF) table to First Normal Form (1NF) relation(s).

13.10 Describe the concept of full functional dependency and describe how this concept relates to 2NF. Provide an example to illustrate your answer.

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

13.12 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

13.13 Continue the process of normalizing the Client and PropertyRentalOwner 1NF relations shown in Figure 13.13 to 3NF relations. At the end of this process check that the resultant 3NF relations are the same as those produced from the alternative ClientRental 1NF relation shown in Figure 13.16.

13.14 Examine the Patient Medication Form for the *Wellmeadows Hospital* case study shown in Figure 13.18.

(a) Identify the functional dependencies represented by the attributes shown in the form in Figure 13.18. State any assumptions 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 13.18 to produce a set of well-designed 3NF relations.

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

13.15 The table shown in Figure 13.19 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 13.19 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 13.19. 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 13.19 to 3NF relations. Identify the primary, alternate, and foreign keys in your 3NF relations.

13.16 An agency called *Instant Cover* supplies part-time/temporary staff to hotels within Scotland. The table shown in Figure 13.20 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.

Figure 13.18
The *Wellmeadows Hospital* Patient Medication Form.

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	24/03/04	24/04/05
10334	Tetracyclene	Antibiotic	0.5mg/ml	IV	10	24/03/04	17/04/04
10223	Morphine	Pain Killer	10mg/ml	Oral	10	25/04/05	02/05/06

Figure 13.19
Table displaying sample dentist/patient appointment data.

staffNo	dentistName	patNo	patName	appointment date	time	surgeryNo
S1011	Tony Smith	P100	Gillian White	12-Sep-04	10.00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-04	12.00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-04	10.00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-04	14.00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-04	16.30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-04	18.00	S13

Figure 13.20
Table displaying sample data for the *Instant Cover* agency.

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

(a) The table shown in Figure 13.20 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 13.20. 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 13.20 to 3NF. Identify primary, alternate and foreign keys in your relations.

Chapter

Advanced Normalization

Chapter Objectives

In this chapter you will learn:

- How inference rules can identify a set of *all* functional dependencies for a relation.
- How inference rules called Armstrong's axioms can identify a *minimal* set of useful functional dependencies from the set of all functional dependencies for a relation.
- Normal forms that go beyond Third Normal Form (3NF), which includes Boyce–Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF).
- How to identify Boyce–Codd Normal Form (BCNF).
- How to represent attributes shown on a report as BCNF relations using normalization.
- The concept of multi-valued dependencies and 4NF.
- The problems associated with relations that break the rules of 4NF.
- How to create 4NF relations from a relation which breaks the rules of 4NF.
- The concept of join dependency and 5NF.
- The problems associated with relations that break the rules of 5NF.
- How to create 5NF relations from a relation which breaks the rules of 5NF.

In the previous chapter we introduced the technique of normalization and the concept of functional dependencies between attributes. We described the benefits of using normalization to support database design and demonstrated how attributes shown on sample forms are transformed into First Normal Form (1NF), Second Normal Form (2NF), and then finally Third Normal Form (3NF) relations. In this chapter, we return to consider functional dependencies and describe normal forms that go beyond 3NF such as Boyce–Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF). Relations in 3NF are normally sufficiently well structured to prevent the problems associated with data redundancy, which was described in Section 13.3. However, later normal forms were created to identify relatively rare problems with relations that, if not corrected, may result in undesirable data redundancy.

Structure of this Chapter

With the exception of 1NF, all normal forms discussed in the previous chapter and in this chapter are based on functional dependencies among the attributes of a relation. In Section 14.1 we continue the discussion on the concept of functional dependency which was introduced in the previous chapter. We present a more formal and theoretical aspect of functional dependencies by discussing inference rules for functional dependencies.

In the previous chapter we described the three most commonly used normal forms: 1NF, 2NF, and 3NF. However, R. Boyce and E.F. Codd identified a weakness with 3NF and introduced a stronger definition of 3NF called Boyce–Codd Normal Form (BCNF) (Codd, 1974), which we describe in Section 14.2. In Section 14.3 we present a worked example to demonstrate the process of normalizing attributes originally shown on a report into a set of BCNF relations.

Higher normal forms that go beyond BCNF were introduced later, such as Fourth (4NF) and Fifth (5NF) Normal Forms (Fagin, 1977, 1979). However, these later normal forms deal with situations that are very rare. We describe 4NF and 5NF in Sections 14.4 and 14.5.

To illustrate the process of normalization, examples are drawn from the *DreamHome* case study described in Section 10.4 and documented in Appendix A.

14.1 More on Functional Dependencies

One of the main concepts associated with normalization is **functional dependency**, which describes the relationship between attributes (Maier, 1983). In the previous chapter we introduced this concept. In this section we describe this concept in a more formal and theoretical way by discussing inference rules for functional dependencies.

14.1.1 Inference Rules for Functional Dependencies

In Section 13.4 we identified the characteristics of the functional dependencies that are most useful in normalization. However, even if we restrict our attention to functional dependencies with a one-to-one (1:1) relationship between attributes on the left- and right-hand sides of the dependency that hold for all time and are fully functionally dependent, then the complete set of functional dependencies for a given relation can still be very large. It is important to find an approach that can reduce that set to a manageable size. Ideally, we want to identify a set of functional dependencies (represented as X) for a relation that is smaller than the complete set of functional dependencies (represented as Y) for that relation and has the property that every functional dependency in Y is implied by the functional dependencies in X. Hence, if we enforce the integrity constraints defined by the functional dependencies in X, we automatically enforce the integrity constraints defined in the larger set of functional dependencies in Y. This requirement suggests that there must be functional dependencies that can be inferred from other functional dependencies. For example, functional dependencies $A \rightarrow B$ and $B \rightarrow C$ in a relation implies that the functional dependency $A \rightarrow C$ also holds in that relation. $A \rightarrow C$ is an example of a **transitive** functional dependency and was discussed previously in Sections 13.4 and 13.7.

How do we begin to identify useful functional dependencies on a relation? Normally, the database designer starts by specifying functional dependencies that are semantically obvious; however, there are usually numerous other functional dependencies. In fact, the task of specifying all possible functional dependencies for 'real' database projects is more often than not, impractical. However, in this section we do consider an approach that helps identify the complete set of functional dependencies for a relation and then discuss how to achieve a minimal set of functional dependencies that can represent the complete set.

The set of all functional dependencies that are implied by a given set of functional dependencies X is called the **closure** of X, written X^+. We clearly need a set of rules to help compute X^+ from X. A set of inference rules, called **Armstrong's axioms**, specifies how new functional dependencies can be inferred from given ones (Armstrong, 1974). For our discussion, let A, B, and C be subsets of the attributes of the relation R. Armstrong's axioms are as follows:

(1) **Reflexivity:** If B is a subset of A, then $A \rightarrow B$

(2) **Augmentation:** If $A \rightarrow B$, then $A,C \rightarrow B,C$

(3) **Transitivity:** If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

Note that each of these three rules can be directly proved from the definition of functional dependency. The rules are **complete** in that given a set X of functional dependencies, all functional dependencies implied by X can be derived from X using these rules. The rules are also **sound** in that no additional functional dependencies can be derived that are not implied by X. In other words, the rules can be used to derive the closure of X^+.

Several further rules can be derived from the three given above that simplify the practical task of computing X^+. In the following rules, let D be another subset of the attributes of relation R, then:

(4) **Self-determination:** $A \rightarrow A$

(5) **Decomposition:** If $A \rightarrow B,C$, then $A \rightarrow B$ and $A \rightarrow C$

(6) **Union:** If $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow B,C$

(7) **Composition:** If $A \rightarrow B$ and $C \rightarrow D$ then $A,C \rightarrow B,D$

Rule 1 Reflexivity and Rule 4 Self-determination state that a set of attributes always determines any of its subsets or itself. Because these rules generate functional dependencies that are always true, such dependencies are trivial and, as stated earlier, are generally not interesting or useful. Rule 2 Augmentation states that adding the same set of attributes to both the left- and right-hand sides of a dependency results in another valid dependency. Rule 3 Transitivity states that functional dependencies are transitive. Rule 5 Decomposition states that we can remove attributes from the right-hand side of a dependency. Applying this rule repeatedly, we can decompose $A \rightarrow B, C, D$ functional dependency into the set of dependencies $A \rightarrow B$, $A \rightarrow C$, and $A \rightarrow D$. Rule 6 Union states that we can do the opposite: we can combine a set of dependencies $A \rightarrow B$, $A \rightarrow C$, and $A \rightarrow D$ into a single functional dependency A → B, C, D. Rule 7 Composition is more general than Rule 6 and states that we can combine a set of non-overlapping dependencies to form another valid dependency.

To begin to identify the set of functional dependencies F for a relation, typically we first identify the dependencies that are determined from the semantics of the attributes of the relation. Then we apply Armstrong's axioms (Rules 1 to 3) to infer additional functional dependencies that are also true for that relation. A systematic way to determine these additional functional dependencies is to first determine each set of attributes A that appears on the left-hand side of some functional dependencies and then to determine the set of *all* attributes that are dependent on A. Thus, for each set of attributes A we can determine the set A^+ of attributes that are functionally determined by A based on F; (A^+ is called the **closure of A under F**).

14.1.2 Minimal Sets of Functional Dependencies

In this section, we introduce what is referred to as **equivalence** of sets of functional dependencies. A set of functional dependencies Y is **covered by** a set of functional dependencies X, if every functional dependency in Y is also in X^+; that is, every dependency in Y can be inferred from X. A set of functional dependencies X is minimal if it satisfies the following conditions:

- Every dependency in X has a single attribute on its right-hand side.
- We cannot replace any dependency A → B in X with dependency C → B, where C is a proper subset of A, and still have a set of dependencies that is equivalent to X.
- We cannot remove any dependency from X and still have a set of dependencies that is equivalent to X.

A minimal set of dependencies should be in a standard form with no redundancies. A minimal cover of a set of functional dependencies X is a minimal set of dependencies X_{min} that is equivalent to X. Unfortunately there can be several minimal covers for a set of functional dependencies. We demonstrate the identification of the minimal cover for the StaffBranch relation in the following example.

Example 14.1 Identifying the minimal set of functional dependencies of the StaffBranch relation

We apply the three conditions described above on the set of functional dependencies for the StaffBranch relation listed in Example 13.5 to produce the following functional dependencies:

staffNo → sName
staffNo → position
staffNo → salary
staffNo → branchNo
staffNo → bAddress

branchNo → bAddress
bAddress → branchNo
branchNo, position → salary
bAddress, position → salary

These functional dependencies satisfy the three conditions for producing a minimal set of functional dependencies for the StaffBranch relation. Condition 1 ensures that every dependency is in a standard form with a single attribute on the right-hand side. Conditions 2 and 3 ensure that there are no redundancies in the dependencies either by having redundant attributes on the left-hand side of a dependency (Condition 2) or by having a dependency that can be inferred from the remaining functional dependencies in X (Condition 3).

In the following section we return to consider normalization. We begin by discussing Boyce–Codd Normal Form (BCNF), a stronger normal form than 3NF.

14.2 Boyce–Codd Normal Form (BCNF)

In the previous chapter we demonstrated how 2NF and 3NF disallow partial and transitive dependencies on the *primary key* of a relation, respectively. Relations that have these types of dependencies may suffer from the update anomalies discussed in Section 13.3. However, the definition of 2NF and 3NF discussed in Sections 13.7 and 13.8, respectively, do not consider whether such dependencies remain on other candidate keys of a relation, if any exist. In Section 13.9 we presented general definitions for 2NF and 3NF that disallow partial and transitive dependencies on any *candidate key* of a relation, respectively. Application of the general definitions of 2NF and 3NF may identify additional redundancy caused by dependencies that violate one or more candidate keys. However, despite these additional constraints, dependencies can still exist that will cause redundancy to be present in 3NF relations. This weakness in 3NF, resulted in the presentation of a stronger normal form called Boyce–Codd Normal Form (Codd, 1974).

14.2.1 Definition of Boyce–Codd Normal Form

Boyce–Codd Normal Form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation; however, BCNF also has additional constraints compared with the general definition of 3NF given in Section 13.9.

Boyce–Codd Normal Form (BCNF) A relation is in BCNF, if and only if, every determinant is a candidate key.

To test whether a relation is in BCNF, we identify all the determinants and make sure that they are candidate keys. Recall that a determinant is an attribute, or a group of attributes, on which some other attribute is fully functionally dependent.

The difference between 3NF and BCNF is that for a functional dependency A → B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key, whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key. Therefore, Boyce–Codd Normal Form is a stronger form of 3NF, such that every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

Before considering the next example, we re-examine the Client, Rental, PropertyForRent, and Owner relations shown in Figure 13.17. The Client, PropertyForRent, and Owner relations are all in BCNF, as each relation only has a single determinant, which is the candidate key. However, recall that the Rental relation contains the three determinants (clientNo, propertyNo), (clientNo, rentStart), and (propertyNo, rentStart), originally identified in Example 13.11, as shown below:

fd1 clientNo, propertyNo → rentStart, rentFinish
fd5′ clientNo, rentStart → propertyNo, rentFinish
fd6′ propertyNo, rentStart → clientNo, rentFinish

As the three determinants of the Rental relation are also candidate keys, the Rental relation is also already in BCNF. Violation of BCNF is quite rare, since it may only happen under specific conditions. The potential to violate BCNF may occur when:

- the relation contains two (or more) composite candidate keys; or
- the candidate keys overlap, that is have at least one attribute in common.

In the following example, we present a situation where a relation violates BCNF and demonstrate the transformation of this relation to BCNF. This example demonstrates the process of converting a 1NF relation to BCNF relations.

Example 14.2 Boyce–Codd Normal Form (BCNF)

In this example, we extend the *DreamHome* case study to include a description of client interviews by members of staff. The information relating to these interviews is in the ClientInterview relation shown in Figure 14.1. The members of staff involved in interviewing clients are allocated to a specific room on the day of interview. However, a room may be allocated to several members of staff as required throughout a working day. A client is only interviewed once on a given date, but may be requested to attend further interviews at later dates.

The ClientInterview relation has three candidate keys: (clientNo, interviewDate), (staffNo, interviewDate, interviewTime), and (roomNo, interviewDate, interviewTime). Therefore the ClientInterview relation has three composite candidate keys, which overlap by sharing the

Figure 14.1 ClientInterview relation.

ClientInterview

clientNo	interviewDate	interviewTime	staffNo	roomNo
CR76	13-May-05	10.30	SG5	G101
CR56	13-May-05	12.00	SG5	G101
CR74	13-May-05	12.00	SG37	G102
CR56	1-Jul-05	10.30	SG5	G102

common attribute interviewDate. We select (clientNo, interviewDate) to act as the primary key for this relation. The ClientInterview relation has the following form:

ClientInterview (clientNo, interviewDate, interviewTime, staffNo, roomNo)

The ClientInterview relation has the following functional dependencies:

fd1	clientNo, interviewDate → interviewTime, staffNo, roomNo	(Primary key)
fd2	staffNo, interviewDate, interviewTime → clientNo	(Candidate key)
fd3	roomNo, interviewDate, interviewTime → staffNo, clientNo	(Candidate key)
fd4	staffNo, interviewDate → roomNo	

We examine the functional dependencies to determine the normal form of the ClientInterview relation. As functional dependencies fd1, fd2, and fd3 are all candidate keys for this relation, none of these dependencies will cause problems for the relation. The only functional dependency that requires discussion is (staffNo, interviewDate) → roomNo (represented as fd4). Even though (staffNo, interviewDate) is not a candidate key for the ClientInterview relation this functional dependency is allowed in 3NF because roomNo is a primary-key attribute being part of the candidate key (roomNo, interviewDate, interviewTime). As there are no partial or transitive dependencies on the primary key (clientNo, interviewDate), and functional dependency fd4 is allowed, the ClientInterview relation is in 3NF.

However, this relation is not in BCNF (a stronger normal form of 3NF) due to the presence of the (staffNo, interviewDate) determinant, which is not a candidate key for the relation. BCNF requires that all determinants in a relation must be a candidate key for the relation. As a consequence the ClientInterview relation may suffer from update anomalies. For example, to change the room number for staff number SG5 on the 13-May-05 we must update two tuples. If only one tuple is updated with the new room number, this results in an inconsistent state for the database.

To transform the ClientInterview relation to BCNF, we must remove the violating functional dependency by creating two new relations called Interview and StaffRoom, as shown in Figure 14.2. The Interview and StaffRoom relations have the following form:

Interview (clientNo, interviewDate, interviewTime, staffNo)
StaffRoom (staffNo, interviewDate, roomNo)

Interview

clientNo	interviewDate	interviewTime	staffNo
CR76	13-May-05	10.30	SG5
CR56	13-May-05	12.00	SG5
CR74	13-May-05	12.00	SG37
CR56	1-Jul-05	10.30	SG5

StaffRoom

staffNo	interviewDate	roomNo
SG5	13-May-05	G101
SG37	13-May-05	G102
SG5	1-Jul-05	G102

Figure 14.2 The Interview and StaffRoom BCNF relations.

We can decompose any relation that is not in BCNF into BCNF as illustrated. However, it may not always be desirable to transform a relation into BCNF; for example, if there is a functional dependency that is not preserved when we perform the decomposition (that is, the determinant and the attributes it determines are placed in different relations). In this situation, it is difficult to enforce the functional dependency in the relation, and an important constraint is lost. When this occurs, it may be better to stop at 3NF, which always preserves dependencies. Note in Example 14.2, in creating the two BCNF relations from the original ClientInterview relation, we have 'lost' the functional dependency, roomNo, interviewDate, interviewTime → staffNo, clientNo (represented as fd3), as the determinant for this dependency is no longer in the same relation. However, we must recognize that if the functional dependency, staffNo, interviewDate → roomNo (represented as fd4) is not removed, the ClientInterview relation will have data redundancy.

The decision as to whether it is better to stop the normalization at 3NF or progress to BCNF is dependent on the amount of redundancy resulting from the presence of fd4 and the significance of the 'loss' of fd3. For example, if it is the case that members of staff conduct only one interview per day, then the presence of fd4 in the ClientInterview relation will not cause redundancy and therefore the decomposition of this relation into two BCNF relations is not helpful or necessary. On the other hand, if members of staff conduct numerous interviews per day, then the presence of fd4 in the ClientInterview relation will cause redundancy and normalization of this relation to BCNF is recommended. However, we should also consider the significance of losing fd3; in other words, does fd3 convey important information about client interviews that must be represented in one of the resulting relations? The answer to this question will help to determine whether it is better to retain all functional dependencies or remove data redundancy.

14.3 Review of Normalization up to BCNF

The purpose of this section is to review the process of normalization described in the previous chapter and in Section 14.2. We demonstrate the process of transforming attributes displayed on a sample report from the *DreamHome* case study into a set of Boyce–Codd Normal Form relations. In this worked example we use the definitions of 2NF and 3NF that are based on the primary key of a relation. We leave the normalization of this worked example using the general definitions of 2NF and 3NF as an exercise for the reader.

Example 14.3 First normal form (1NF) to Boyce–Codd Normal Form (BCNF)

In this example we extend the *DreamHome* case study to include property inspection by members of staff. When staff are required to undertake these inspections, they are allocated a company car for use on the day of the inspections. However, a car may be allocated to several members of staff as required throughout the working day. A member of staff may inspect several properties on a given date, but a property is only inspected once on a given date. Examples of the *DreamHome* Property Inspection Report are

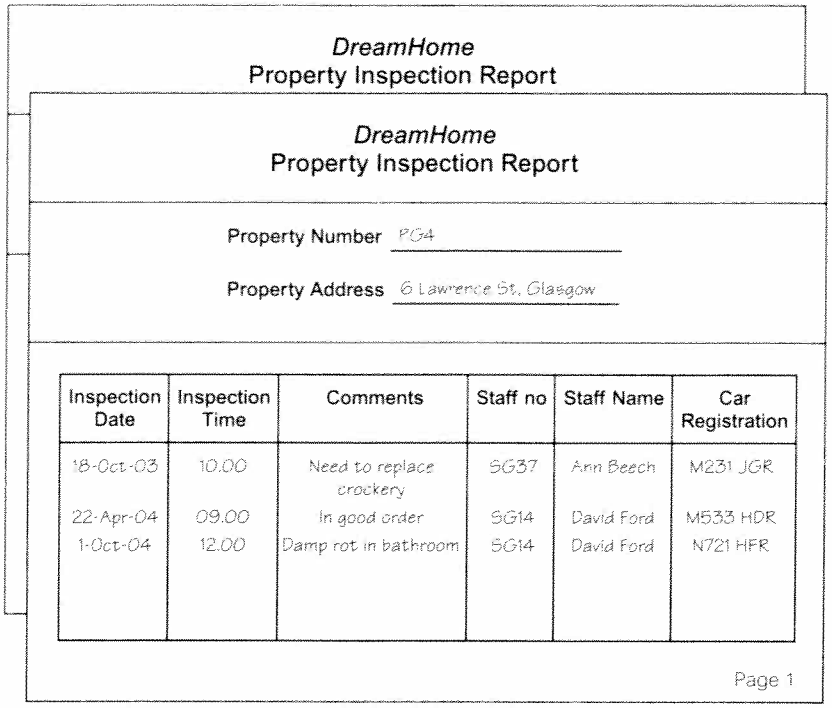


Figure 14.3 *DreamHome* Property Inspection reports.

StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-03 22-Apr-04 1-Oct-04	10.00 09.00 12.00	Need to replace crockery In good order Damp rot in bathroom	SG37 SG14 SG14	Ann Beech David Ford David Ford	M231 JGR M533 HDR N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-04 24-Oct-04	13.00 14.00	Replace living room carpet Good condition	SG14 SG37	David Ford Ann Beech	M533 HDR N721 HFR

Figure 14.4 StaffPropertyInspection unnormalized table.

presented in Figure 14.3. The report on top describes staff inspections of property PG4 in Glasgow.

First Normal Form (1NF)

We first transfer sample data held on two property inspection reports into table format with rows and columns. This is referred to as the StaffPropertyInspection unnormalized table and is shown in Figure 14.4. We identify the key attribute for this unnormalized table as propertyNo.

We identify the repeating group in the unnormalized table as the property inspection and staff details, which repeats for each property. The structure of the repeating group is:

Repeating Group = (iDate, iTime, comments, staffNo, sName, carReg)

Figure 14.5
The First Normal Form (1NF) StaffPropertyInspection relation.

StaffPropertyInspection

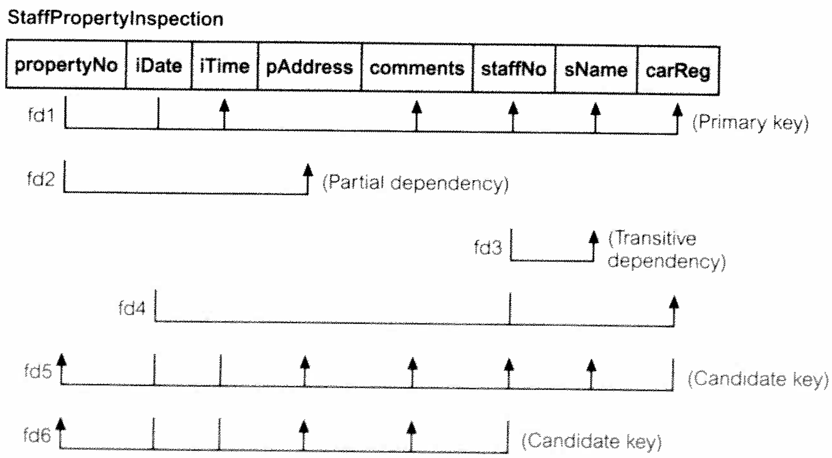
propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-03	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-04	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-04	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-04	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-04	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

As a consequence, there are multiple values at the intersection of certain rows and columns. For example, for propertyNo PG4 there are three values for iDate (18-Oct-03, 22-Apr-04, 1-Oct-04). We transform the unnormalized form to first normal form using the first approach described in Section 13.6. With this approach, we remove the repeating group (property inspection and staff details) by entering the appropriate property details (nonrepeating data) into each row. The resulting first normal form StaffPropertyInspection relation is shown in Figure 14.5.

In Figure 14.6, we present the functional dependencies (fd1 to fd6) for the StaffPropertyInspection relation. We use the functional dependencies (as discussed in Section 13.4.3) to identify candidate keys for the StaffPropertyInspection relation as being composite keys comprising (propertyNo, iDate), (staffNo, iDate, iTime), and (carReg, iDate, iTime). We select (propertyNo, iDate) as the primary key for this relation. For clarity, we place the attributes that make up the primary key together, at the left-hand side of the relation. The StaffPropertyInspection relation is defined as follows:

StaffPropertyInspection (<u>propertyNo</u>, <u>iDate</u>, iTime, pAddress, comments, staffNo, sName, carReg)

Figure 14.6
Functional dependencies of the StaffPropertyInspection relation.



The StaffPropertyInspection relation is in first normal form (1NF) as there is a single value at the intersection of each row and column. The relation contains data describing the inspection of property by members of staff, with the property and staff details repeated several times. As a result, the StaffPropertyInspection relation contains significant redundancy. If implemented, this 1NF relation would be subject to update anomalies. To remove some of these, we must transform the relation into second normal form.

Second Normal Form (2NF)

The normalization of 1NF relations to 2NF involves the removal of partial dependencies on the primary key. If a partial dependency exists, we remove the functionally dependent attributes from the relation by placing them in a new relation with a copy of their determinant.

As shown in Figure 14.6, the functional dependencies (fd1 to fd6) of the StaffPropertyInspection relation are as follows:

fd1	propertyNo, iDate → iTime, comments, staffNo, sName, carReg	(Primary key)
fd2	propertyNo → pAddress	(Partial dependency)
fd3	staffNo → sName	(Transitive dependency)
fd4	staffNo, iDate → carReg	
fd5	carReg, iDate, iTime → propertyNo, pAddress, comments, staffNo, sName	(Candidate key)
fd6	staffNo, iDate, iTime → propertyNo, pAddress, comments	(Candidate key)

Using the functional dependencies, we continue the process of normalizing the StaffPropertyInspection relation. We begin by testing whether the relation is in 2NF by identifying the presence of any partial dependencies on the primary key. We note that the property attribute (pAddress) is partially dependent on part of the primary key, namely the propertyNo (represented as fd2), whereas the remaining attributes (iTime, comments, staffNo, sName, and carReg) are fully dependent on the whole primary key (propertyNo and iDate), (represented as fd1). Note that although the determinant of the functional dependency staffNo, iDate → carReg (represented as fd4) only requires the iDate attribute of the primary key, we do not remove this dependency at this stage as the determinant also includes another non-primary-key attribute, namely staffNo. In other words, this dependency is *not* wholly dependent on part of the primary key and therefore does not violate 2NF.

The identification of the partial dependency (propertyNo → pAddress) indicates that the StaffPropertyInspection relation is not in 2NF. To transform the relation into 2NF requires the creation of new relations so that the attributes that are not fully dependent on the primary key are associated with only the appropriate part of the key.

The StaffPropertyInspection relation is transformed into second normal form by removing the partial dependency from the relation and creating two new relations called Property and PropertyInspection with the following form:

Property	(<u>propertyNo</u>, pAddress)
PropertyInspection	(<u>propertyNo</u>, <u>iDate</u>, iTime, comments, staffNo, sName, carReg)

These relations are in 2NF, as every non-primary-key attribute is functionally dependent on the primary key of the relation.

Third Normal Form (3NF)

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attributes from the relation by placing them in a new relation along with a copy of their determinant. The functional dependencies within the Property and PropertyInspection relations are as follows:

Property Relation
fd2 propertyNo → pAddress

PropertyInspection Relation
fd1 propertyNo, iDate → iTime, comments, staffNo, sName, carReg
fd3 staffNo → sName
fd4 staffNo, iDate → carReg
fd5′ carReg, iDate, iTime → propertyNo, comments, staffNo, sName
fd6′ staffNo, iDate, iTime → propertyNo, comments

As the Property relation does not have transitive dependencies on the primary key, it is therefore already in 3NF. However, although all the non-primary-key attributes within the PropertyInspection relation are functionally dependent on the primary key, sName is also transitively dependent on staffNo (represented as fd3). We also note the functional dependency staffNo, iDate → carReg (represented as fd4) has a non-primary-key attribute carReg partially dependent on a non-primary-key attribute, staffNo. We do not remove this dependency at this stage as part of the determinant for this dependency includes a primary-key attribute, namely iDate. In other words, this dependency is *not* wholly transitively dependent on non-primary-key attributes and therefore does not violate 3NF. (In other words, as described in Section 13.9, when considering all candidate keys of a relation, the staffNo, iDate → carReg dependency is allowed in 3NF because carReg is a primary-key attribute as it is part of the candidate key (carReg, iDate, iTime) of the original PropertyInspection relation.)

To transform the PropertyInspection relation into 3NF, we remove the transitive dependency (staffNo → sName) by creating two new relations called Staff and PropertyInspect with the form:

Staff (staffNo, sName)
PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

The Staff and PropertyInspect relations are in 3NF as no non-primary-key attribute is wholly functionally dependent on another non-primary-key attribute. Thus, the StaffPropertyInspection relation shown in Figure 14.5 has been transformed by the process of normalization into three relations in 3NF with the following form:

Property (propertyNo, pAddress)
Staff (staffNo, sName)
PropertyInspect (propertyNo, iDate, iTime, comments, staffNo, carReg)

Boyce–Codd Normal Form (BCNF)

We now examine the Property, Staff, and PropertyInspect relations to determine whether they are in BCNF. Recall that a relation is in BCNF if every determinant of a relation is a candidate key. Therefore, to test for BCNF, we simply identify all the determinants and make sure they are candidate keys.

The functional dependencies for the Property, Staff, and PropertyInspect relations are as follows:

Property Relation

fd2 propertyNo → pAddress

Staff Relation

fd3 staffNo → sName

PropertyInspect Relation

fd1′ propertyNo, iDate → iTime, comments, staffNo, carReg

fd4 staffNo, iDate → carReg

fd5′ carReg, iDate, iTime → propertyNo, comments, staffNo

fd6′ staffNo, iDate, iTime → propertyNo, comments

We can see that the Property and Staff relations are already in BCNF as the determinant in each of these relations is also the candidate key. The only 3NF relation that is not in BCNF is PropertyInspect because of the presence of the determinant (staffNo, iDate), which is not a candidate key (represented as fd4). As a consequence the PropertyInspect relation may suffer from update anomalies. For example, to change the car allocated to staff number SG14 on the 22-Apr-03, we must update two tuples. If only one tuple is updated with the new car registration number, this results in an inconsistent state for the database.

To transform the PropertyInspect relation into BCNF, we must remove the dependency that violates BCNF by creating two new relations called StaffCar and Inspection with the form:

StaffCar (<u>staffNo</u>, <u>iDate</u>, carReg)

Inspection (<u>propertyNo</u>, <u>iDate</u>, iTime, comments, staffNo)

The StaffCar and Inspection relations are in BCNF as the determinant in each of these relations is also a candidate key.

In summary, the decomposition of the StaffPropertyInspection relation shown in Figure 14.5 into BCNF relations is shown in Figure 14.7. In this example, the decomposition of the

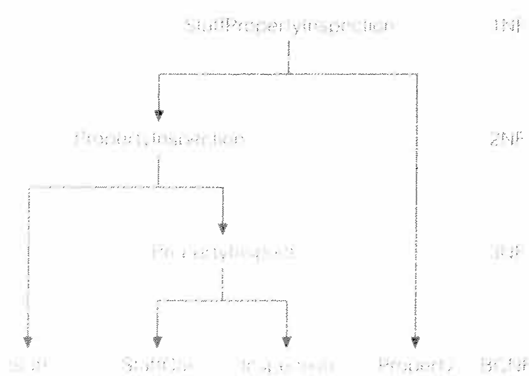


Figure 14.7 Decomposition of the StaffPropertyInspection relation into BCNF relations.

original StaffPropertyInspection relation to BCNF relations has resulted in the 'loss' of the functional dependency: carReg, iDate, iTime → propertyNo, pAddress, comments, staffNo, sName, as parts of the determinant are in different relations (represented as fd5). However, we recognize that if the functional dependency, staffNo, iDate → carReg (represented as fd4) is not removed, the PropertyInspect relation will have data redundancy.

The resulting BCNF relations have the following form:

Property (<u>propertyNo</u>, pAddress)
Staff (<u>staffNo</u>, sName)
Inspection (<u>propertyNo</u>, <u>iDate</u>, iTime, comments, staffNo)
StaffCar (<u>staffNo</u>, <u>iDate</u>, carReg)

The original StaffPropertyInspection relation shown in Figure 14.5 can be recreated from the Property, Staff, Inspection, and StaffCar relations using the primary key/foreign key mechanism. For example, the attribute staffNo is a primary key within the Staff relation and is also present within the Inspection relation as a foreign key. The foreign key allows the association of the Staff and Inspection relations to identify the name of the member of staff undertaking the property inspection.

14.4 Fourth Normal Form (4NF)

Although BCNF removes any anomalies due to functional dependencies, further research led to the identification of another type of dependency called a **Multi-Valued Dependency** (MVD), which can also cause data redundancy (Fagin, 1977). In this section, we briefly describe a multi-valued dependency and the association of this type of dependency with Fourth Normal Form (4NF).

14.4.1 Multi-Valued Dependency

The possible existence of multi-valued dependencies in a relation is due to First Normal Form, which disallows an attribute in a tuple from having a set of values. For example, if we have two multi-valued attributes in a relation, we have to repeat each value of one of the attributes with every value of the other attribute, to ensure that tuples of the relation are consistent. This type of constraint is referred to as a multi-valued dependency and results in data redundancy. Consider the BranchStaffOwner relation shown in Figure 14.8(a), which

Figure 14.8(a) The BranchStaffOwner relation.

BranchStaffOwner

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy