FIGURE 3-11
The EQUIPMENT_REPAIR Relation After an Incorrect Update

	ItemNumber	EquipmentType	AcquisitionCost	RepairNumber	RepairDate	RepairCost
1	100	Drill Press	3500.00	2000	2018-05-05	375.00
2	200	Lathe	4750.00	2100	2018-05-07	255.00
3	100	Drill Press	3500.00	2200	2018-06-19	178.00
4	300	Mill	27300.00	2300	2018-06-19	1875.00
5	100	Drill Press	3500.00	2400	2018-07-05	0.00
6	100	Drill Press	5500.00	2500	2018-08-17	275.00

costs. If there were, say, 10,000 rows in the table, however, it might be very difficult to detect this error. This condition is called an **update anomaly**.

BY THE WAY Notice that the EQUIPMENT_REPAIR table in Figures 3-10 and 3-11 duplicates data. For example, the AcquisitionCost of the same item of equipment appears several times. Any table that duplicates data is susceptible to update anomalies like the one in Figure 3-11. A table that has such inconsistencies is said to have **data integrity problems.**

As we will discuss further in Chapter 4, to improve query speed, we sometimes design a table to have duplicated data. Be aware, however, that any time we design a table this way, we open the door to data integrity problems.

A Short History of Normal Forms

When Codd defined the relational model, he noticed that some tables had modification anomalies. In his second paper,[4] he defined first normal form, second normal form, and third normal form. He defined **first normal form (1NF)** as the *set of conditions for a relation*, shown in Figure 3-4. Any table meeting the conditions in Figure 3-4 is therefore a relation in 1NF.

This definition, however, brings us back to the "To Key or Not to Key" discussion. Codd's set of conditions for a relation does not require a primary key, but one is clearly implied by the condition that all rows must be unique. Thus, there are various opinions on whether or not a relation has to have a defined primary key to be in 1NF.[5]

For practical purposes, we will define 1NF as it is used in this book as a table that:

1. Meets the set of conditions for a relation, and
2. Has a defined primary key.[6]

Codd also noted that some tables (or, interchangeably in this book, relations) in 1NF had modification anomalies. He found that he could remove some of those anomalies by applying certain conditions. A relation that met those conditions, which we will discuss later in this chapter, was said to be in **second normal form (2NF)**. He also observed, however, that relations in 2NF could also have anomalies, and so he defined **third normal form (3NF)**, which is a set of conditions that removes even more anomalies and which we will also discuss later in this chapter. As time went by, other researchers found

[4] E. F. Codd and A. L. Dean, "Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description," *Access and Control*, San Diego, California, November 11-12, 1971 ACM 1971.
[5] For a review of some of the discussion, see the Wikipedia article at *http://en.wikipedia.org/wiki/First_normal_form*.
[6] Some definitions of 1NF also state that there can be "no repeating groups." This refers to the *multivalue, multicolumn problem* we discuss in Chapter 4 and also deal with in our discussion of *multivalued dependencies* later in this chapter.

still other ways that anomalies can occur, and the conditions for **Boyce-Codd Normal Form (BCNF)** were defined.

These normal forms are defined so that a relation in BCNF is in 3NF, a relation in 3NF is in 2NF, and a relation in 2NF is in 1NF. Thus, if you put a relation into BCNF, it is automatically in the lesser normal forms.

Normal forms 2NF through BCNF concern anomalies that arise from functional dependencies. Other sources of anomalies were found later. They led to the definition of **fourth normal form (4NF)** and **fifth normal form (5NF)**, both of which we will discuss later in this chapter. So it went, with researchers chipping away at modification anomalies, each one improving on the prior normal form.

In 1982, Ronald Fagin published a paper that took a different tack.[7] Instead of looking for just another normal form, Fagin asked, "What conditions need to exist for a relation to have no anomalies?" In that paper, he defined **domain/key normal form (DK/NF)** (and, no, that is not a typo—the slash appears between domain and key in the complete name, but between DK and NF in the acronym.) Fagin ended the search for normal forms by showing that a relation in DK/NF has no modification anomalies and, further, that a relation that has no modification anomalies is in DK/NF. DK/NF is discussed in more detail later in this chapter.

Normalization Categories

As shown in Figure 3-12, normalization theory can be divided into three major categories. Some anomalies arise from functional dependencies, some arise from multivalued dependencies, and some arise from data constraints and odd conditions.

2NF, 3NF, and BCNF are all concerned with anomalies that are caused by functional dependencies. A relation that is in BCNF has no modification anomalies from functional dependencies. It is also automatically in 2NF and 3NF, and, therefore, we will focus on transforming relations into BCNF. However, it is instructive to work through the progression of normal forms from 1NF to BCNF in order to understand how each normal form deals with anomalies, and we will do this later in this chapter.[8]

As shown in the second row of Figure 3-12, some anomalies arise because of another kind of dependency called a multivalued dependency. Those anomalies can be eliminated by placing each multivalued dependency in a relation of its own, a condition known as 4NF. You will see how to do that in the last section of this chapter.

The third source of anomalies is esoteric. These problems involve specific, rare, and even strange data constraints. Accordingly, we will not discuss them in this text.

FIGURE 3-12
Summary of Normalization Theory

Source of Anomaly	Normal Forms	Design Principles
Functional dependencies	1NF, 2NF, 3NF, BCNF	BCNF: Design tables so that every determinant is a candidate key.
Multivalued dependencies	4NF	4NF: Move each multivalued dependency to a table of its own.
Data constraints and oddities	5NF, DK/NF	DK/NF: Make every constraint a logical consequence of candidate keys and domains.

[7] Ronald Fagin, "A Normal Form for Relational Databases That Is Based on Domains and Keys," *ACM Transactions on Database Systems*, September 1981, pp. 387-415.
[8] See Christopher J. Date, *An Introduction to Database Systems*, 8th ed. (New York: Addison-Wesley, 2003) for a complete discussion of normal forms.

From First Normal Form to Boyce-Codd Normal Form Step by Step

As we discussed earlier in this chapter, a table is in 1NF if and only if (1) it meets the *definition of a relation* in Figure 3-4 and (2) it has a *defined primary key*. From Figure 3-4 this means that the following must hold: the cells of a table must be a single value, and neither repeating groups nor arrays are allowed as values; all entries in a column must be of the same data type; each column must have a unique name, but the order of the columns in the table is not significant; and no two rows in a table may be identical, but the order of the rows is not significant. To this, we add the requirement of having a primary key defined for the table.

Second Normal Form

When Codd discovered anomalies in 1NF tables, he defined 2NF to eliminate some of these anomalies. A relation is in 2NF if and only if *it is in 1NF* and *all non-key attributes are determined by the entire primary key*. This means that if the primary key is a composite primary key, then no non-key attribute can be determined by an attribute or set of attributes that make up only part of the key. Thus, if you have a relation **R (A, B, N, O, P)** with the composite key **(A, B)**, then none of the non-key attributes **N, O**, or **P** can be determined by just **A** or just **B**.

Note that the only way a non-key attribute can be dependent on part of the primary key is if there is a *composite primary key*. This means that relations with *single-attribute primary keys* are automatically in 2NF.

For example, consider the STUDENT_ACTIVITY relation:

STUDENT_ACTIVITY (StudentID, Activity, ActivityFee)

The STUDENT_ACTIVITY relation is in 1NF and is shown with sample data in Figure 3-13. Note that STUDENT_ACTIVITY has the composite primary key (StudentID, Activity), which allows us to determine the fee a particular student will have to pay for a particular activity. However, because fees are determined by activities, ActivityFee is also functionally dependent on just Activity itself, and we can say that ActivityFee is **partially dependent** on the key of the table. The set of functional dependencies is therefore:

(StudentID, Activity) → ActivityFee

Activity → ActivityFee

Thus, there is a non-key attribute determined by part of the composite primary key, and the STUDENT_ACTIVITY relation is *not* in 2NF. What do we do in this case? We will have to move the columns of the functional dependency based on the partial primary key attribute into a separate relation while leaving the determinant in the original relation as a foreign key. We will end up with two relations:

STUDENT_ACTIVITY (StudentID, *Activity*)

ACTIVITY_FEE (Activity, ActivityFee)

FIGURE 3-13
The 1NF STUDENT_ ACTIVITY Relation

STUDENT_ACTIVITY

	StudentID	Activity	ActivityFee
1	100	Golf	65.00
2	100	Skiing	200.00
3	200	Skiing	200.00
4	200	Swimming	50.00
5	300	Skiing	200.00
6	300	Swimming	50.00
7	400	Golf	65.00
8	400	Swimming	50.00

FIGURE 3-14
The 2NF STUDENT_ACTIVITY and ACTIVITY_FEE Relations

STUDENT_ACTIVITY

	StudentID	Activity
1	100	Golf
2	100	Skiing
3	200	Skiing
4	200	Swimming
5	300	Skiing
6	300	Swimming
7	400	Golf
8	400	Swimming

ACTIVITY_FEE

	Activity	ActivityFee
1	Golf	65.00
2	Skiing	200.00
3	Swimming	50.00

The Activity column in STUDENT_ACTIVITY becomes a foreign key. The new relations are shown in Figure 3-14. Now, are the two new relations in 2NF? Yes. STUDENT_ACTIVITY still has a composite primary key, but now has no attributes that are dependent on only a part of this composite key. ACTIVITY_FEE has a set of attributes (just one each in this case) that are dependent on the entire primary key.

Third Normal Form

However, the conditions necessary for 2NF do not eliminate all anomalies. To deal with additional anomalies, Codd defined 3NF. A relation is in 3NF if and only if *it is in 2NF* and *there are no non-key attributes determined by another non-key attribute*. The technical name for a non-key attribute determined by another non-key attribute is **transitive dependency**.[9] We can therefore restate the definition of 3NF: a relation is in 3NF if and only if *it is in 2NF* and *it has no transitive dependencies*. Thus, in order for our relation **R (A, B, N, O, P)** to be in 3NF, none of the non-key attributes **N, O**, or **P** can be determined by **N, O**, or **P**.

For example, consider the relation STUDENT_HOUSING shown in Figure 3-15. The STUDENT_HOUSING relation is in 2NF, and the table schema is:

STUDENT_HOUSING (StudentID, Building, BuildingFee)

Here we have a single-attribute primary key, StudentID, so the relation is in 2NF because there is no possibility of a non-key attribute being dependent on only part of the primary key. Furthermore, if we know the student, we can determine the building where he or she is residing, so:

StudentID → Building

FIGURE 3-15
The 2NF STUDENT_HOUSING Relation

STUDENT_HOUSING

	StudentID	Building	BuildingFee
1	100	Randoplh	3200.00
2	200	Ingersoll	3400.00
3	300	Randoplh	3200.00
4	400	Randoplh	3200.00
5	500	Pitkin	3500.00
6	600	Ingersoll	3400.00
7	700	Ingersoll	3400.00
8	800	Pitkin	3500.00

[9] In terms of functional dependencies, a transitive dependency is defined as: IF A → B and B → C, THEN A → C.

FIGURE 3-16
The 3NF STUDENT_HOUSING and BUILDING_FEE Relations

STUDENT_HOUSING

	StudentID	Building
1	100	Randoplh
2	200	Ingersoll
3	300	Randoplh
4	400	Randoplh
5	500	Pitkin
6	600	Ingersoll
7	700	Ingersoll
8	800	Pitkin

BUILDING_FEE

	Building	BuildingFee
1	Ingersoll	3400.00
2	Pitkin	3500.00
3	Randoplh	3200.00

However, the building fee is independent of which student is housed in the building, and, in fact, the same fee is charged for every room in a building. Therefore, Building determines BuildingFee:

Building → BuildingFee

Thus, a non-key attribute (BuildingFee) is functionally determined by another non-key attribute (Building), and the relation is *not* in 3NF.

To put the relation into 3NF, we will have to move the columns of the functional dependency into a separate relation while leaving the determinant in the original relation as a foreign key. We will end up with two relations:

STUDENT_HOUSING (StudentID, *Building*)
BUILDING_FEE (Building, BuildingFee)

The Building column in STUDENT_HOUSING becomes a foreign key. The two relations are now in 3NF (work through the logic yourself to make sure you understand 3NF) and are shown in Figure 3-16.

Boyce-Codd Normal Form

Some database designers normalize their relations to 3NF. Unfortunately, there are still anomalies due to functional dependences in 3NF. Together with Raymond Boyce, Codd defined BCNF to fix this situation. A relation is in BCNF if and only if *it is in 3NF* and *every determinant is a candidate key*.

For example, consider the relation STUDENT_ADVISOR shown in Figure 3-17, where a student (StudentID) can have one or more majors (Major), a major can have one or more

FIGURE 3-17
The 3NF STUDENT_ADVISOR Relation

STUDENT_ADVISOR

	StudentID	Major	AdvisorName
1	100	Math	Cauchy
2	200	Psychology	Jung
3	300	Math	Riemann
4	400	Math	Cauchy
5	500	Psychology	Perls
6	600	English	Austin
7	700	Psychology	Perls
8	700	Math	Riemann
9	800	Math	Cauchy
10	800	Psychology	Jung

faculty advisors (AdvisorName), and a faculty member advises in only one major area. Note that the figure shows two students (StudentIDs 700 and 800) with double majors (both students show Majors of Math and Psychology) and two Subjects (Math and Psychology) with two Advisors.

Because students can have several majors, StudentID does not determine Major. Moreover, because students can have several advisors, StudentID does not determine AdvisorName. Therefore, StudentID by itself cannot be a key. However, the composite key (StudentID, Major) determines AdvisorName, and the composite key (StudentID, AdvisorName) determines Major. This gives us (StudentID, Major) and (StudentID, AdvisorName) as two candidate keys. We can select either of these as the primary key for the relation. Thus, two STUDENT_ADVISOR schemas with different candidate keys are possible:

STUDENT_ADVISOR (StudentID, Major, AdvisorName)

and

STUDENT_ADVISOR (StudentID, Major, AdvisorName)

Note that STUDENT_ADVISOR is in 2NF because it has no non-key attributes in the sense that every attribute is a part of *at least one* candidate key. This is a subtle condition, based on the fact that technically the definition of 2NF states that no *non-prime attribute* can be partially dependent on a candidate key, where a **non-prime attribute** is an attribute that is not contained in *any* candidate key. Furthermore, STUDENT_ADVISOR is in 3NF because there are no transitive dependencies in the relation.

The two candidate keys for this relation are **overlapping candidate keys** because they share the attribute StudentID. When a table in 3NF has overlapping candidate keys, it can still have modification anomalies based on functional dependencies. In the STUDENT_ADVISOR relation, there will be modification anomalies because there is one other functional dependency in the relation. Because a faculty member can be an advisor for only one major area, AdvisorName determines Major. Therefore, AdvisorName is a determinant but not a candidate key.

Suppose that we have a student (StudentID = 300) majoring in psychology (Major = Psychology) with faculty advisor Perls (AdvisorName = Perls). Further, assume that this row is the only one in the table with the AdvisorName value of Perls. If we delete this row, we will lose all data about Perls. This is a deletion anomaly. Similarly, we cannot insert the data to represent the Economics advisor Keynes until a student majors in Economics. This is an insertion anomaly. Situations like this led to the development of BCNF.

What do we do with the STUDENT_ADVISOR relation? As before, we move the functional dependency creating the problem to another relation while leaving the determinant in the original relation as a foreign key. In this case, we will create the relations:

STUDENT_ADVISOR (StudentID, *AdvisorName*)
ADVISOR_MAJOR (AdvisorName, Major)

The AdvisorName column in STUDENT_ADVISOR is the foreign key, and the two final relations are shown in Figure 3-18.

Note that a relation in 3NF *may also already be* in BCNF. The only way a relation in 3NF can have problems actually requiring further normalization work to get it into BCNF is if it has *overlapping composite candidate keys*. If the relation (1) does *not* have composite candidate keys or (2) has *non-overlapping* composite candidate keys, then it is *already in BCNF* once it is in 3NF.

Eliminating Anomalies from Functional Dependencies with BCNF

Most modification anomalies occur because of problems with functional dependencies. You can eliminate these problems by progressively testing a relation for 1NF, 2NF, 3NF, and BCNF using the definitions of these normal forms given previously. We will refer to this as the "Step-by-Step" method.

FIGURE 3-18
The BCNF STUDENT_ ADVISOR and ADVISOR_ MAJOR Relations

STUDENT_ADVISOR

	StudentID	AdvisorName
1	100	Cauchy
2	200	Jung
3	300	Riemann
4	400	Cauchy
5	500	Perls
6	600	Austin
7	700	Perls
8	700	Riemann
9	800	Cauchy
10	800	Jung

ADVISOR_MAJOR

	AdvisorName	Major
1	Austin	English
2	Cauchy	Math
3	Jung	Psychology
4	Perls	Psychology
5	Riemann	Math

You can also eliminate such problems by simply designing (or redesigning) your tables so that every determinant is a candidate key. This condition, which, of course, is the definition of BCNF, will eliminate all anomalies due to functional dependencies. We will refer to this method as the "Straight-to-BCNF" or "general normalization" method.

We prefer the "Straight-to-BCNF" general normalization strategy and will use it extensively, but not exclusively, in this book. However, this is merely our preference–either method produces the same results, and you (or your professor) may prefer the "Step-by-Step" method.

The general normalization method is summarized in Figure 3-19. Identify every functional dependency in the relation, and then identify the candidate keys. If there are determinants that are not candidate keys, then the relation is not in BCNF and is subject to modification anomalies. To put the relation into BCNF, follow the procedure in step 3. To fix this procedure in your mind, we will illustrate it with five different examples. We will also compare it to the "Step-by-Step" approach.

FIGURE 3-19
Process for Putting a Relation into BCNF

Process for Putting a Relation into BCNF
1. Identify every functional dependency.
2. Identify every candidate key.
3. If there is a functional dependency that has a determinant that is not a candidate key: A. Move the columns of that functional dependency into a new relation. B. Make the determinant of that functional dependency the primary key of the new relation. C. Leave a copy of the determinant as a foreign key in the original relation. D. Create a referential integrity constraint between the original relation and the new relation.
4. Repeat step 3 until every determinant of every relation is a candidate key.

Note: In step 3, if there is more than one such functional dependency, start with the one with the most columns.

BY THE WAY Our process rule that a relation is in BCNF if and only if every determinant is a candidate key is summed up in a variation of a widely known phrase:

I swear to construct my tables so that all non-key columns are dependent on the key, the whole key and nothing but the key, so help me Codd!

This phrase actually is a very good way to remember the order of the normal forms:

I swear to construct my tables so that all non-key columns are dependent on

- *the key,* [This is **1NF**]
- *the whole key,* [This is **2NF**]
- *and nothing but the key,* [This is **3NF** and **BCNF**]

so help me Codd!

BY THE WAY The goal of the normalization process is to create relations that are in BCNF. It is sometimes stated that the goal is to create relations that are in 3NF, but after the discussion in this chapter, you should understand why BCNF is preferred to 3NF.

Note that some problems are not resolved by even BCNF, and these will require relations in 4NF. We will explain when we need to use 4NF after we discuss our examples of normalizing to BCNF.

Normalization Example 1

Consider the SKU_DATA table:

SKU_DATA (SKU, SKU_Description, Department, Buyer)

As discussed earlier, this table has three functional dependencies:

SKU → (SKU_Description, Department, Buyer)
SKU_Description → (SKU, Department, Buyer)
Buyer → Department

Normalization Example 1: The "Step-by-Step" Method

Both SKU and SKU_Description are candidate keys. Logically, SKU makes more sense as the primary key because it is a surrogate key, so our relation, which is shown in Figure 3-20, is:

SKU_DATA (SKU, SKU_Description, Department, Buyer)

Checking the relation against Figure 3-4, and noting that it has a defined primary key, we find that SKU_DATA is in 1NF.

Is the SKU_DATA relation in 2NF? A relation is in 2NF if and only if *it is in 1NF* and *all non-key attributes are determined by the entire primary key*. Because the primary key SKU is a single attribute key, all the non-key attributes are therefore dependent on the entire primary key. Thus, the SKU_DATA relation is in 2NF.

Is the SKU_DATA relation in 3NF? A relation is in 3NF if and only if *it is in 2NF* and *there are no non-key attributes determined by another non-key attribute*. Because we seem to have two non-key attributes (SKU_Description and Buyer) that determine non-key attributes, the relation is *not* in 3NF!

FIGURE 3-20
The SKU_DATA Relation

SKU_DATA

	SKU	SKU_Description	Department	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
3	100300	Std. Scuba Tank, Light Blue	Water Sports	Pete Hansen
4	100400	Std. Scuba Tank, Dark Blue	Water Sports	Pete Hansen
5	100500	Std. Scuba Tank, Light Green	Water Sports	Pete Hansen
6	100600	Std. Scuba Tank, Dark Green	Water Sports	Pete Hansen
7	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
8	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
9	201000	Half-dome Tent	Camping	Cindy Lo
10	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
11	203000	Half-dome Tent Vestibule - Wide	Camping	Cindy Lo
12	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
13	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

However, this is where things get a bit tricky. A *non-key attribute* is an attribute that is neither (1) a candidate key itself nor (2) part of a candidate key. SKU_Description, therefore, is *not* a *non-key attribute* (sorry about the double negative). The only non-key attribute is Buyer!

Therefore, we must remove only the functional dependency

Buyer → Department

We will now have two relations (using the name BUYER_2 to distinguish this relation from BUYER in the Cape Codd database as discussed earlier in this chapter):

SKU_DATA_2 (SKU, SKU_Description, *Buyer*)
BUYER_2 (Buyer, Department)

Is SKU_DATA_2 in 3NF? Yes, it is—there are no non-key attributes that determine another non-key attribute.

Is the SKU_DATA_2 relation in BCNF? A relation is in BCNF if and only if *it is in 3NF* and *every determinant is a candidate key*. The determinants in SKU_DATA_2 are SKU and SKU_Description:

SKU → (SKU_Description, Buyer)
SKU_Description → (SKU, Buyer)

Both determinants are candidate keys (they both determine all the other attributes in the relation). Thus, every determinant is a candidate key, and the relationship *is* in BCNF.

At this point, we need to check the BUYER_2 relation to determine if it is in BCNF. Work through the steps yourself for BUYER_2 to check your understanding of the "Step-by-Step" method. You will find that BUYER_2 *is* in BCNF, and therefore our normalized relations, as shown with the sample data in Figure 3-21, are:

SKU_DATA_2 (SKU, SKU_Description, *Buyer*)
BUYER_2 (Buyer, Department)

Both of these tables are now in BCNF and will have no anomalies due to functional dependencies. For the data in these tables to be consistent, however, we also need to define a referential integrity constraint (note that this is step 3D in Figure 3-19):

SKU_DATA_2.Buyer must exist in BUYER_2.Buyer

FIGURE 3-21
The Normalized SKU_DATA_2 and BUYER_2 Relations

SKU_DATA_2

	SKU	SKU_Description	Buyer
1	100100	Std. Scuba Tank, Yellow	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Pete Hansen
3	100300	Std. Scuba Tank, Light Blue	Pete Hansen
4	100400	Std. Scuba Tank, Dark Blue	Pete Hansen
5	100500	Std. Scuba Tank, Light Green	Pete Hansen
6	100600	Std. Scuba Tank, Dark Green	Pete Hansen
7	101100	Dive Mask, Small Clear	Nancy Meyers
8	101200	Dive Mask, Med Clear	Nancy Meyers
9	201000	Half-dome Tent	Cindy Lo
10	202000	Half-dome Tent Vestibule	Cindy Lo
11	203000	Half-dome Tent Vestibule - Wide	Cindy Lo
12	301000	Light Fly Climbing Harness	Jerry Martin
13	302000	Locking carabiner, Oval	Jerry Martin

BUYER_2

	Buyer	Department
1	Cindy Lo	Camping
2	Jerry Martin	Climbing
3	Nancy Meyers	Water Sports
4	Pete Hansen	Water Sports

This statement means that every value in the Buyer column of SKU_DATA_2 must also exist as a value in the Buyer column of BUYER_2.

Normalization Example 1: The "Straight-to-BCNF" Method

Now let's rework this example using the "Straight-to-BCNF" method. SKU and SKU_Description determine all of the columns in the table, so they are candidate keys. Buyer is a determinant, but it does not determine all of the other columns, and hence it is not a candidate key. Therefore, SKU_DATA has a determinant that is not a candidate key and is therefore not in BCNF. It will have modification anomalies.

To remove such anomalies, in step 3A in Figure 3-19, we move the columns of functional dependency whose determinant is not a candidate key into a new relation. In this case, we place Buyer and Department into a new relation (again using the name BUYER_2 to distinguish this relation from BUYER in the Cape Codd database as discussed earlier in this chapter):

BUYER_2 (Buyer, Department)

Next, in step 3B in Figure 3-19, we make the determinant of the functional dependency the primary key of the new relation. In this case, Buyer becomes the primary key:

BUYER_2 (Buyer, Department)

Next, following step 3C in Figure 3-19, we leave a copy of the determinant as a foreign key in the original relation. Thus, SKU_DATA becomes SKU_DATA_2:

SKU_DATA_2 (SKU, SKU_Description, *Buyer*)

The resulting relations are thus:

SKU_DATA_2 (SKU, SKU_Description, *Buyer*)
BUYER_2 (Buyer, Department)

where SKU_DATA_2.Buyer is a foreign key to the BUYER_2 relation.

Both of these relations are now in BCNF and will have no anomalies due to functional dependencies. For the data in these tables to be consistent, however, we also need to define the referential integrity constraint in step 3D in Figure 3-19:

SKU_DATA_2.Buyer must exist in BUYER_2.Buyer

This statement means that every value in the Buyer column of SKU_DATA_2 must also exist as a value in the Buyer column of BUYER_2. Sample data for the resulting tables is the same as shown in Figure 3-21.

Note that both the "Step-by-Step" method and the "Straight-to-BCNF" method produced exactly the same results. Use the method you prefer; the results will be the same. To keep this chapter reasonably short, we will use only the "Straight-to-BCNF" method for the rest of the normalization examples.

Normalization Example 2

Now consider the EQUIPMENT_REPAIR relation in Figure 3-10. The structure of the table is:

EQUIPMENT_REPAIR (ItemNumber, EquipmentType, AcquisitionCost, RepairNumber, RepairDate, RepairCost)

Examining the data in Figure 3-10, the functional dependencies are:

ItemNumber → (EquipmentType, AcquisitionCost)

RepairNumber → (ItemNumber, EquipmentType, AcquisitionCost, RepairDate, RepairCost)

Both ItemNumber and RepairNumber are determinants, but only RepairNumber is a candidate key. Accordingly, EQUIPMENT_REPAIR is not in BCNF and is subject to modification anomalies. Following the procedure in Figure 3-19, we place the columns of the problematic functional dependency into a separate table, as follows:

EQUIPMENT_ITEM (ItemNumber, EquipmentType, AcquisitionCost)

and remove all but ItemNumber from EQUIPMENT_REPAIR (and rearrange the columns so that the primary key RepairNumber is the first column in the relation) to create:

REPAIR (RepairNumber, *ItemNumber*, RepairDate, RepairCost)

We also need to create the referential integrity constraint:

REPAIR.ItemNumber must exist in EQUIPMENT_ITEM.ItemNumber

Data for these two new relations are shown in Figure 3-22.

BY THE WAY There is another, more intuitive way to think about normalization. Do you remember your eighth-grade English teacher? She said that every paragraph should have a single theme. If you write a paragraph that has two themes, you should break it up into two paragraphs, each with a single theme.

The problem with the EQUIPMENT_REPAIR relation is that it has two themes: one about repairs and a second about items. We eliminated modification anomalies by breaking that single table with two themes into two tables, each with a single theme. Sometimes, it is helpful to look at a table and ask, "How many themes does it have?" If it has more than one, then redefine the table so that it has a single theme.

Normalization Example 3

Consider now the Cape Codd database ORDER_ITEM relation shown in Figure 3-1 with the structure:

ORDER_ITEM (OrderNumber, SKU, Quantity, Price, ExtendedPrice)

FIGURE 3-22
The Normalized EQUIPMENT_ITEM and REPAIR Relations

EQUIPMENT_ITEM

	ItemNumber	EquipmentType	AcquisitionCost
1	100	Drill Press	3500.00
2	200	Lathe	4750.00
3	300	Mill	27300.00

REPAIR

	RepairNumber	ItemNumber	RepairDate	RepairCost
1	2000	100	2018-05-05	375.00
2	2100	200	2018-05-07	255.00
3	2200	100	2018-06-19	178.00
4	2300	300	2018-06-19	1875.00
5	2400	100	2018-07-05	0.00
6	2500	100	2018-08-17	275.00

with functional dependencies:

(OrderNumber, SKU) → (Quantity, Price, ExtendedPrice)
(Quantity, Price) → ExtendedPrice

This table is not in BCNF because the determinant (Quantity, Price) is not a candidate key. We can follow the same normalization practice as illustrated in examples 1 and 2, but in this case, because the second functional dependency arises from the formula

ExtendedPrice = (Quantity * Price)

we reach a silly result.

To see why, we follow the procedure in Figure 3-19 to create tables such that every determinant is a candidate key. This means that we move the columns Quantity, Price, and ExtendedPrice to tables of their own, as follows:

EXTENDED_PRICE (Quantity, Price, ExtendedPrice)
ORDER_ITEM_2 (OrderNumber, SKU, *Quantity*, *Price*)

Notice that we left both Quantity and Price in the original relation as a composite foreign key. These two tables are in BCNF, but the values in the EXTENDED_PRICE table are ridiculous. They are just the results of multiplying Quantity by Price. The simple fact is that we do not need to create a table to store these results. Instead, any time we need to know ExtendedPrice, we will just compute it. In fact, we can define this formula to the DBMS and let the DBMS compute the value of ExtendedPrice when necessary. You will see how to do this with Microsoft SQL Server 2017, Oracle's Oracle Database, and MySQL 5.7 in Chapters 10A, 10B, and 10C, respectively.

Using the formula, we can remove ExtendedPrice from the table. The resulting table is in BCNF:

ORDER_ITEM_2 (OrderNumber, SKU, Quantity, Price)

Note that Quantity and Price are no longer foreign keys. The ORDER_ITEM_2 table with sample data now appears as shown in Figure 3-23.

Normalization Example 4

Consider the following table that stores data about student activities:

STUDENT_ACTIVITY (StudentID, StudentName, Activity, ActivityFee, AmountPaid)

FIGURE 3-23
The Normalized ORDER_ITEM_2 Relation

ORDER_ITEM_2

	OrderNumber	SKU	Quantity	Price
1	1000	201000	1	300.00
2	1000	202000	1	130.00
3	2000	101100	4	50.00
4	2000	101200	2	50.00
5	3000	100200	1	300.00
6	3000	101100	2	50.00
7	3000	101200	1	50.00

where StudentID is a student identifier, StudentName is student name, Activity is the name of a club or other organized student activity, ActivityFee is the cost of joining the club or participating in the activity, and AmountPaid is the amount the student has paid toward the ActivityFee. Figure 3-24 shows sample data for this table.

StudentID is a unique student identifier, so we know that:

StudentID → StudentName

However, does the following functional dependency exist?

StudentID → Activity

It does if a student belongs to just one club or participates in just one activity, but it does not if a student belongs to more than one club or participates in more than one activity. Looking at the data, student Davis with StudentID 200 participates in both Skiing and Swimming, so StudentID does *not* determine Club. StudentID does not determine ActivityFee or AmountPaid, either.

Now consider the StudentName column. Does StudentName determine StudentID? Is, for example, the value 'Jones' always paired with the same value of StudentID? No, there are two students named 'Jones', and they have different StudentID values. StudentName does not determine any other column in this table either.

Considering the next column, Activity, we know that many students can belong to a club. Therefore, Activity does not determine StudentID or StudentName. Does Activity determine ActivityFee? Is the value 'Skiing', for example, always paired with the same value of ActivityFee? From these data, it appears so, and using just this sample data, we can conclude that Activity determines ActivityFee.

However, this data is just a sample. Logically, it is possible for students to pay different costs, perhaps because they select different levels of activity participation. If that were the case, then we would say that

(StudentID, Activity) → ActivityFee

FIGURE 3-24
Sample Data for the STUDENT_ACTIVITY Relation

STUDENT_ACTIVITY

	StudentID	StudentName	Activity	ActivityFee	AmountPaid
1	100	Jones	Golf	65.00	65.00
2	100	Jones	Skiing	200.00	0.00
3	200	Davis	Skiing	200.00	0.00
4	200	Davis	Swimming	50.00	50.00
5	300	Garrett	Skiing	200.00	100.00
6	300	Garrett	Swimming	50.00	50.00
7	400	Jones	Golf	65.00	65.00
8	400	Jones	Swimming	50.00	50.00

To find out, we need to check with the users. Here, assume that all students pay the same fee for a given activity. The last column is AmountPaid, and it does not determine anything.

So far, we have two functional dependencies:

StudentID → StudentName
Activity → ActivityFee

Are there other functional dependencies with composite determinants? No single column determines AmountPaid, so consider possible composite determinants for it. AmountPaid is dependent on both the student and the club the student has joined. Therefore, it is determined by the combination of the determinants StudentID and Activity. Thus, we can say

(StudentID, Activity) → AmountPaid

So far we have three determinants: StudentID, Activity, and (StudentID, Activity). Are any of these candidate keys? Do any of these determinants identify a unique row? From the data, it appears that (StudentID, Activity) identifies a unique row and is a candidate key. Again, in real situations, we would need to check this assumption out with the users.

STUDENT_ACTIVITY_PAYMENT is not in BCNF because columns StudentID and Activity are both determinants but neither is a candidate key. StudentID and Activity are only part of the candidate key (StudentID, Activity).

BY THE WAY Both StudentID and Activity are *part of* the candidate key (StudentID, Activity). This, however, is not good enough. A determinant must have *all* of the same columns to be the same as a candidate key.

To normalize this table, we need to construct tables so that every determinant is a candidate key. We can do this by creating a separate table for each functional dependency as we did before. The result is:

STUDENT (StudentID, StudentName)
ACTIVITY (Activity, ActivityFee)
PAYMENT (*StudentID*, *Activity*, AmountPaid)

with referential integrity constraints:

PAYMENT.StudentID must exist in STUDENT.StudentID

and

PAYMENT.Activity must exist in ACTIVITY.Activity

These tables are in BCNF and will have no anomalies from functional dependencies. The sample data for the normalized tables are shown in Figure 3-25.

Normalization Example 5

Now consider a normalization process that requires two iterations of step 3 in the procedure in Figure 3-19. To do this, we will extend the SKU_DATA relation by adding the budget code of each department. We call the revised relation SKU_DATA_3 and define it as follows:

SKU_DATA_3 (SKU, SKU_Description, Department, DeptBudgetCode, Buyer)

FIGURE 3-25
The Normalized STUDENT, ACTIVITY, and PAYMENT Relations

STUDENT

	StudentID	StudentName
1	100	Jones
2	200	Davis
3	300	Garrett
4	400	Jones

ACTIVITY

	Activity	ActivityFee
1	Golf	65.00
2	Skiing	200.00
3	Swimming	50.00

PAYMENT

	StudentID	Activity	AmountPaid
1	100	Golf	65.00
2	100	Skiing	0.00
3	200	Skiing	0.00
4	200	Swimming	50.00
5	300	Skiing	100.00
6	300	Swimming	50.00
7	400	Golf	65.00
8	400	Swimming	50.00

Sample data for this relation are shown in Figure 3-26. SKU_DATA_3 has the following functional dependencies:

SKU → (SKU_Description, Department, DeptBudgetCode, Buyer)
SKU_Description → (SKU, Department, DeptBudgetCode, Buyer)
Buyer → (Department, DeptBudgetCode)
Department → DeptBudgetCode
DeptBudgetCode → Department

Of the five determinants, both SKU and SKU_Description are candidate keys, but Buyer, Department, and DeptBudgetCode are not candidate keys. Therefore, this relation is not in BCNF.

To normalize this table, we must transform it into two or more tables that are in BCNF. In this case, there are two problematic functional dependencies. According to the note at the end of the procedure in Figure 3-19, we take the functional dependency whose determinant is not a candidate key and has the largest number of columns first. In this case, we take the columns of

Buyer → (Department, DeptBudgetCode)

and place them in a table of their own.

Next, we make the determinant the primary key of the new table, remove all columns except Buyer from SKU_DATA_3, and make Buyer a foreign key of the new version of

FIGURE 3-26
Sample Data for the SKU_DATA_3 Relation

SKU_DATA_3

	SKU	SKU_Description	Department	DeptBudgetCode	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	BC-100	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	BC-100	Pete Hansen
3	100300	Std. Scuba Tank, Light Blue	Water Sports	BC-100	Pete Hansen
4	100400	Std. Scuba Tank, Dark Blue	Water Sports	BC-100	Pete Hansen
5	100500	Std. Scuba Tank, Light Green	Water Sports	BC-100	Pete Hansen
6	100600	Std. Scuba Tank, Dark Green	Water Sports	BC-100	Pete Hansen
7	101100	Dive Mask, Small Clear	Water Sports	BC-100	Nancy Mey...
8	101200	Dive Mask, Med Clear	Water Sports	BC-100	Nancy Mey...
9	201000	Half-dome Tent	Camping	BC-200	Cindy Lo
10	202000	Half-dome Tent Vestibule	Camping	BC-200	Cindy Lo
11	203000	Half-dome Tent Vestibule - Wide	Camping	BC-200	Cindy Lo
12	301000	Light Fly Climbing Harness	Climbing	BC-300	Jerry Martin
13	302000	Locking carabiner, Oval	Climbing	BC-300	Jerry Martin

SKU_DATA_3, which we will name SKU_DATA_4. We can also now assign SKU as the primary key of SKU_DATA_4. The results are (using the name BUYER_3 to distinguish this relation from the other versions of BUYER discussed earlier in this chapter):

BUYER_3 (Buyer, Department, DeptBudgetCode)
SKU_DATA_4 (SKU, SKU_Description, *Buyer*)

We also create the referential integrity constraint:

SKU_DATA_4.Buyer must exist in BUYER_3.Buyer

The functional dependencies from SKU_DATA_4 are:

SKU → (SKU_Description, Buyer)
SKU_Description → (SKU, Buyer)

Because every determinant of SKU_DATA_4 is also a candidate key, the relationship is now in BCNF. Looking at the functional dependencies from BUYER_3, we find:

Buyer → (Department, DeptBudgetCode)
Department → DeptBudgetCode
DeptBudgetCode → Department

BUYER_3 is *not* in BCNF because neither of the determinants Department and DeptBudgetCode are candidate keys. In this case, we must move (Department, DeptBudgetCode) into a table of its own. Following the procedure in Figure 3-19 and breaking BUYER_3 into two tables (DEPARTMENT and BUYER_4) gives us a set of three tables (where *SKU* as a surrogate key is the logical for primary key fir SKU_DATA_4, and *Department* is the logical primary key for DEPARTMENT because other columns are semantically descriptors of the department):

DEPARTMENT (Department, DeptBudgetCode)
BUYER_4 (Buyer, *Department*)
SKU_DATA_4 (SKU, SKU_Description, *Buyer*)

These tables have the referential integrity constraints:

SKU_DATA_4.Buyer must exist in BUYER_4.Buyer
BUYER_4.Department must exist in DEPARTMENT.Department

The functional dependencies from all three of these tables are:

Department → DeptBudgetCode
DeptBudgetCode → Department
Buyer → Department
SKU → (SKU_Description, Buyer)
SKU_Description → (SKU, Buyer)

At last, every determinant is a candidate key, and all three of the tables are in BCNF. The resulting relations from these operations are shown in Figure 3-27.

Eliminating Anomalies from Multivalued Dependencies

All of the anomalies in the last section were due to functional dependencies, and when we normalize relations to BCNF, we eliminate these anomalies. However, anomalies can also arise from another kind of dependency: the multivalued dependency. A **multivalued dependency** occurs when a determinant is matched with a particular *set* of values.

DEPARTMENT

	Department	DeptBudgetCode
1	Camping	BC-200
2	Climbing	BC-300
3	Water Sports	BC-100

BUYER_4

	Buyer	Department
1	Cindy Lo	Camping
2	Jerry Martin	Climbing
3	Nancy Meyers	Water Sports
4	Pete Hansen	Water Sports

SKU_DATA_4

	SKU	SKU_Description	Buyer
1	100100	Std. Scuba Tank, Yellow	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Pete Hansen
3	100300	Std. Scuba Tank, Light Blue	Pete Hansen
4	100400	Std. Scuba Tank, Dark Blue	Pete Hansen
5	100500	Std. Scuba Tank, Light Green	Pete Hansen
6	100600	Std. Scuba Tank, Dark Green	Pete Hansen
7	101100	Dive Mask, Small Clear	Nancy Meyers
8	101200	Dive Mask, Med Clear	Nancy Meyers
9	201000	Half-dome Tent	Cindy Lo
10	202000	Half-dome Tent Vestibule	Cindy Lo
11	203000	Half-dome Tent Vestibule - Wide	Cindy Lo
12	301000	Light Fly Climbing Harness	Jerry Martin
13	302000	Locking carabiner, Oval	Jerry Martin

FIGURE 3-27
The Normalized DEPARTMENT, BUYER_4, and SKU_DATA_4 Relations

Examples of multivalued dependencies are:

EmployeeName →→ EmployeeDegree
EmployeeName →→ EmployeeSibling
PartKitName →→ Part

In each case, the determinant is associated with a set of values, and example data for each of these multivalued dependencies are shown in Figure 3-28. Such expressions are read as "EmployeeName multidetermines EmployeeDegree" and "EmployeeName multidetermines EmployeeSibling" and "PartKitName multidetermines Part." Note that multideterminants are shown with a double arrow rather than a single arrow.

Employee Jones, for example, has degrees AA and BA. Employee Green has degrees BS, MS, and PhD. Employee Chau has just one degree, BS. Similarly, employee Jones has siblings (brothers and sisters) Fred, Sally, and Frank. Employee Green has sibling Nikki, and employee Chau has siblings Jonathan and Eileen. Finally, PartKitName Bike Repair has parts Wrench, Screwdriver, and Tube Fix. Other kits have parts as shown in Figure 3-28.

FIGURE 3-28
Three Examples of Multivalued Dependencies

EMPLOYEE_DEGREE

	EmployeeName	EmployeeDegree
1	Chau	BS
2	Green	BS
3	Green	MS
4	Green	PhD
5	Jones	AA
6	Jones	BA

EMPLOYEE_SIBLING

	EmployeeName	EmployeeSibling
1	Chau	Eileen
2	Chau	Jonathan
3	Green	Nikki
4	Jones	Frank
5	Jones	Fred
6	Jones	Sally

PARTKIT_PART

	PartKitName	Part
1	Bike Repair	Screwdriver
2	Bike Repair	Tube Fix
3	Bike Repair	Wrench
4	First Aid	Aspirin
5	First Aid	Bandaids
6	First Aid	Elastic Band
7	First Aid	Ibuprofin
8	Toolbox	Drill
9	Toolbox	Drill bits
10	Toolbox	Hammer
11	Toolbox	Saw
12	Toolbox	Screwdriver

FIGURE 3-29
EMPLOYEE_DEGREE_SIBLING Relation with Two Multivalued Dependencies

EMPLOYEE_DEGREE_SIBLING

	EmployeeName	EmployeeDegree	EmployeeSibling
1	Chau	BS	Eileen
2	Chau	BS	Jonathan
3	Green	BS	Nikki
4	Green	MS	Nikki
5	Green	PhD	Nikki
6	Jones	AA	Frank
7	Jones	AA	Fred
8	Jones	AA	Sally
9	Jones	BA	Frank
10	Jones	BA	Fred
11	Jones	BA	Sally

Unlike functional dependencies, the determinant of a multivalued dependency can never be the primary key. In all three of the tables in Figure 3-28, the primary key consists of the composite of the two columns in each table. For example, the primary key of the EMPLOYEE_DEGREE table is the composite key (EmployeeName, EmployeeDegree).

Multivalued dependencies pose no problem as long as they exist in tables of their own. None of the tables in Figure 3-28 has modification anomalies. However, if $A \rightarrow\rightarrow B$, then any relation that contains A, B, and one or more additional columns will have modification anomalies.

For example, consider the situation if we combine the employee data in Figure 3-28 into a single EMPLOYEE_DEGREE_SIBLING table with three columns (EmployeeName, EmployeeDegree, EmployeeSibling), as shown in Figure 3-29.

Now, what actions need to be taken if employee Jones earns an MBA? We must add three rows to the table. If we do not, if we only add the row ('Jones', 'MBA', 'Fred'), it will appear as if Jones is an MBA with her brother Fred, but not with her sister Sally or her other brother Frank. However, suppose Green earns an MBA. Then we need only add one row ('Green', 'MBA', 'Nikki'). But, if Chau earns an MBA, we need to add two rows. These are insertion anomalies. There are equivalent modification and deletion anomalies as well.

In Figure 3-29, we combined two multivalued dependencies into a single table and thereby created modification anomalies. Unfortunately, we will also get anomalies if we combine a multivalued dependency with any other column, even if that other column has no multivalued dependency.

Figure 3-30 shows what happens when we combine the multivalued dependency

$$\text{PartKitName} \rightarrow\rightarrow \text{Part}$$

FIGURE 3-30
PARTKIT_PART_PRICE Relation with a Functional Dependency and a Multivalued Dependency

PARTKIT_PART_PRICE

	PartKitName	Part	PartKitPrice
1	Bike Repair	Screwdriver	14.95
2	Bike Repair	Tube Fix	14.95
3	Bike Repair	Wrench	14.95
4	First Aid	Aspirin	24.95
5	First Aid	Bandaids	24.95
6	First Aid	Elastic Band	24.95
7	First Aid	Ibuprofin	24.95
8	Toolbox	Drill	74.95
9	Toolbox	Drill bits	74.95
10	Toolbox	Hammer	74.95
11	Toolbox	Saw	74.95
12	Toolbox	Screwdriver	74.95

FIGURE 3-31
Placing the Two Multivalued Dependencies in Figure 3-2 into Separate Relations

PRODUCT_BUYER_SKU

	BuyerName	SKU_Managed
1	Cindy Lo	201000
2	Cindy Lo	202000
3	Jenny Martin	301000
4	Jenny Martin	302000
5	Nancy Meyers	101100
6	Nancy Meyers	101200
7	Pete Hansen	100100
8	Pete Hansen	100200

PRODUCT_BUYER_MAJOR

	BuyerName	CollegeMajor
1	Cindy Lo	History
2	Jenny Martin	Business Administration
3	Jenny Martin	English Literature
4	Nancy Meyers	Art
5	Nancy Meyers	Info Systems
6	Pete Hansen	Business Administration

with the functional dependency

PartKitName → PartKitPrice

For the data to be consistent, we must repeat the value of price for as many rows as each kit has parts. For this example, we must add three PARTKIT_PART_PRICE rows for the Bike Repair kit, four rows for the First Aid kit, and five rows for the Toolbox kit. The result is duplicated data that can cause data integrity problems.

Now you also know the problem with the relation in Figure 3-2. Anomalies exist in that table because it contains two multivalued dependencies:

BuyerName →→ SKU_Managed
BuyerName →→ CollegeMajor

Fortunately, it is easy to deal with multivalued dependencies: put them into a table of their own. None of the tables in Figure 3-28 has modification anomalies because each table consists of only the columns in a single, multivalued dependency. Thus, to fix the table in Figure 3-2, we must move BuyerName and SKU_Managed into one table and BuyerName and CollegeMajor into a second table:

PRODUCT_BUYER_SKU (BuyerName, SKU_Managed)
PRODUCT_BUYER_MAJOR (*BuyerName*, CollegeMajor)

The results are shown in Figure 3-31. If we want to maintain strict equivalence between these tables, we would also add the referential integrity constraint:

PRODUCT_BUYER_MAJOR.BuyerName must exist in PRODUCT_BUYER_SKU.BuyerName

This referential integrity constraint may not be necessary, depending on the requirements of the application.

Notice that when you put multivalued dependencies into a table of their own, they disappear. The result is just a table with two columns, and the primary key (and sole candidate key) is the composite of those two columns. When multivalued dependencies have been isolated in this way, the table is said to be in *fourth normal form (4NF)*.

The hardest part of multivalued dependencies is finding them. Once you know they exist in a table, just move them into a table of their own. Whenever you encounter tables with odd anomalies, especially anomalies that require you to insert, modify, or delete different numbers of rows to maintain integrity, check for multivalued dependencies.

BY THE WAY You will sometimes hear people use the term *normalize* in phrases like "that table has been normalized" or "check to see if those tables are normalized." Unfortunately, not everyone means the same thing with these words. Some people do not know about BCNF, and they will use it to mean tables in 3NF, which is a lesser form of normalization, one that allows for anomalies from functional dependencies that BCNF does not allow. Others use it to mean tables that are both BCNF and 4NF. Others may mean something else. The best choice is to use the term *normalize* to mean tables that are in both BCNF and 4NF.

Fifth Normal Form

There is a fifth normal form (5NF), also known as **Project-Join Normal Form (PJ/NF)**, which involves an anomaly where a table can be split apart but not correctly joined back together. However, the conditions under which this happens are complex, and generally if a relation is in 4NF it is in 5NF. We will not deal with 5NF in this book. For more information about 5NF, start with the works cited earlier in this chapter and the Wikipedia article at *http://en.wikipedia.org/wiki/Fifth_normal_form*.

Domain/Key Normal Form

As discussed earlier in this chapter, in 1982 Ronald Fagin published a paper that defined domain/key normal form (DK/NF). Fagin asked, "What conditions need to exist for a relation to have no anomalies?" He showed that a relation in DK/NF has no modification anomalies and, further, that a relation that has no modification anomalies is in DK/NF.

But what does this mean? Basically, DK/NF requires that all the constraints on the data values be logical implications of the definitions of domains and keys. To the level of detail in this text, and to the level of detail experienced by 99 percent of all database practitioners, this can be restated as follows: every determinant of a functional dependency must be a candidate key. This, of course, is simply our definition of BCNF, and, for practical purposes, relations in BCNF are in DK/NF as well.

Summary

Databases arise from three sources: from existing data, from new systems development, and from the redesign of existing databases. This chapter and the next are concerned with databases that arise from existing data. Even though a table is a simple concept, certain tables can lead to surprisingly difficult processing problems. This chapter uses the concept of normalization to understand and possibly solve those problems. Figure 3-3 lists terms you should be familiar with.

A relation is a special case of a table; all relations are tables, but not all tables are relations. Relations are tables that have the properties listed in Figure 3-4. Three sets of terms are used to describe relation structure: (relation, attribute, tuple); (table, column, row); and (file, field, and record). Sometimes these terms are mixed and matched. In practice, the terms *table* and *relation* are commonly used synonymously, and we will do so for the balance of this text.

In a functional dependency, the value of one attribute, or attributes, determines the value of another. In the functional dependency $A \rightarrow B$, attribute A is called the determinant. Some functional dependencies arise from equations, but many others do not. The purpose of a database is, in fact, to store instances of functional dependencies that do not arise from equations.

Determinants that have more than one attribute are called composite determinants. If A → (B, C), then A → B and A → C (decomposition rule). However, if (A, B) → C, then, in general, neither A → C nor B → C. It is true that if A → B and A → C, then A → (B, C) (union rule).

If A → B, the values of A may or may not be unique in a relation. However, every time a given value of A appears, it will be paired with the same value of B. A determinant is unique in a relation only if it determines every other attribute of the relation. You cannot always rely on determining functional dependencies from sample data. The best idea is to verify your conclusions with the users of the data.

A key is a combination of one or more columns used to identify one or more rows. A composite key is a key with two or more attributes. A determinant that determines every other attribute is called a candidate key. A relation may have more than one candidate key. One of them is selected to be used by the DBMS for finding rows and is called the primary key. A surrogate key is an artificial attribute used as a primary key. The value of a surrogate key is supplied by the DBMS and has no meaning to the user. A foreign key is a key in one table that references the primary key of a second table. A referential integrity constraint is a limitation on data values of a foreign key that ensures that every value of the foreign key has a match to a value of a primary key.

The three kinds of modification anomalies are insert, update, and delete. Codd and others defined normal forms for describing different table structures that lead to anomalies. A table that meets the conditions listed in Figure 3-4 is in 1NF. Some anomalies arise from functional dependencies. Three forms, 2NF, 3NF, and BCNF, are used to treat such anomalies.

In this text, we are only concerned with the best of these forms, BCNF. If a relation is in BCNF, then no anomalies from functional dependencies can occur. A relation is in BCNF if every determinant is a candidate key.

Relations can be normalized using either a "Step-by-Step" method or a "Straight-to-BCNF" method. Which method to use is a matter of personal preference, and both methods produce the same results.

Some anomalies arise from multivalued dependencies. A multidetermines B, or A →→ B, if A determines a set of values. If A multidetermines B, then any relation that contains A, B, and one or more other columns will have modification anomalies. Anomalies due to multivalued dependencies can be eliminated by placing the multivalued dependency in a table of its own. Such tables are in 4NF.

There is a 5NF, but generally tables in 4NF are in 5NF. DK/NF has been defined, but in practical terms, the definition of DK/NF is the same as the definition of BCNF.

Key Terms

attribute
Boyce-Codd Normal Form (BCNF)
candidate key
composite determinant
composite key
data integrity problems
database integrity
decomposition rule
deletion anomaly
determinant
domain
domain integrity constraint
domain/key normal form (DK/NF)
entity
entity integrity constraint
fifth normal form (5NF)
first normal form (1NF)
foreign key
fourth normal form (4NF)
functional dependency
functionally dependent
insertion anomaly
key
multivalued dependency
non-prime attribute
normal forms
null value
overlapping candidate key
partially dependent
primary key
Project-Join Normal Form (PJ/NF)
referential integrity constraint
relation
second normal form (2NF)
SKU (stock keeping unit)
surrogate key
third normal form (3NF)
transitive dependency
tuple
union rule
update anomaly