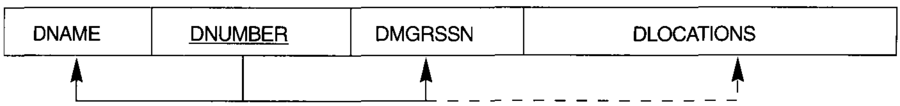
10.3.4 First Normal Form

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model;[12] historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a *single tuple*. In other words, 1NF disallows "relations within relations" or "relations as attribute values within tuples." The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values.**

Consider the DEPARTMENT relation schema shown in Figure 10.1, whose primary key is DNUMBER, and suppose that we extend it by including the DLOCATIONS attribute as shown in Figure 10.8a. We assume that each department can have *a number of* locations. The DEPARTMENT schema and an example relation state are shown in Figure 10.8. As we can see,

(a) DEPARTMENT



(b) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

FIGURE 10.8 Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of same relation with redundancy.

12. This condition is removed in the *nested relational model* and in *object-relational systems* (ORDBMSs), both of which allow *unnormalized relations* (see Chapter 22).

this is not in 1NF because DLOCATIONS is not an atomic attribute, as illustrated by the first tuple in Figure 10.8b. There are two ways we can look at the DLOCATIONS attribute:

- The domain of DLOCATIONS contains atomic values, but some tuples can have a set of these values. In this case, DLOCATIONS *is not* functionally dependent on the primary key DNUMBER.
- The domain of DLOCATIONS contains sets of values and hence is nonatomic. In this case, DNUMBER → DLOCATIONS, because each set is considered a single member of the attribute domain.[13]

In either case, the DEPARTMENT relation of Figure 10.8 is not in 1NF; in fact, it does not even qualify as a relation according to our definition of relation in Section 5.1. There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute DLOCATIONS that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT. The primary key of this relation is the combination {DNUMBER, DLOCATION}, as shown in Figure 10.2. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in Figure 10.8c. In this case, the primary key becomes the combination {DNUMBER, DLOCATION}. This solution has the disadvantage of introducing *redundancy* in the relation.
3. If a *maximum number of values* is known for the attribute—for example, if it is known that *at most three locations* can exist for a department—replace the DLOCATIONS attribute by three atomic attributes: DLOCATION1, DLOCATION2, and DLOCATION3. This solution has the disadvantage of introducing *null values* if most departments have fewer than three locations. It further introduces a spurious semantics about the ordering among the location values that is not originally intended. Querying on this attribute becomes more difficult; for example, consider how you would write the query: "List the departments that have "Bellaire" as one of their locations" in this design.

Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values. In fact, if we choose the second solution, it will be decomposed further during subsequent normalization steps into the first solution.

First normal form also disallows multivalued attributes that are themselves composite. These are called **nested relations** because each tuple can have a relation *within it*. Figure 10.9 shows how the EMP_PROJ relation could appear if nesting is allowed. Each tuple represents an employee entity, and a relation PROJS(PNUMBER, HOURS) *within each*

13. In this case we can consider the domain of DLOCATIONS to be the **power set** of the set of single locations; that is, the domain is made up of all possible subsets of the set of single locations.

(a) **EMP_PROJ**

SSN	ENAME	PROJS	
		PNUMBER	HOURS

(b) **EMP_PROJ**

SSN	ENAME	PNUMBER	HOURS
123456789	Smith,John B.	1	32.5
		2	7.5
666884444	Narayan,Ramesh K.	3	40.0
453453453	English,Joyce A.	1	20.0
		2	20.0
333445555	Wong,Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya,Alicia J.	30	30.0
		10	10.0
987987987	Jabbar,Ahmad V.	10	35.0
		30	5.0
987654321	Wallace,Jennifer S.	30	20.0
		20	15.0
888665555	Borg,James E.	20	null

(c) **EMP_PROJ1**

SSN	ENAME

EMP_PROJ2

SSN	PNUMBER	HOURS

FIGURE 10.9 Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a "nested relation" attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

tuple represents the employee's projects and the hours per week that employee works on each project. The schema of this EMP_PROJ relation can be represented as follows:

EMP_PROJ(SSN, ENAME, {PROJS(PNUMBER, HOURS)})

The set braces { } identify the attribute PROJS as multivalued, and we list the component attributes that form PROJS between parentheses ( ). Interestingly, recent trends for supporting complex objects (see Chapter 20) and XML data (see Chapter 26) using the relational model attempt to allow and formalize nested relations within relational database systems, which were disallowed early on by 1NF.

Notice that SSN is the primary key of the EMP_PROJ relation in Figures 10.9a and b, while PNUMBER is the **partial** key of the nested relation; that is, within each tuple, the nested relation must have unique values of PNUMBER. To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combine the partial key with the primary key of the original relation. Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2 shown in Figure 10.9c.

This procedure can be applied recursively to a relation with multiple-level nesting to **unnest** the relation into a set of 1NF relations. This is useful in converting an unnormalized relation schema with many levels of nesting into 1NF relations. The existence of more than one multivalued attribute in one relation must be handled carefully. As an example, consider the following non-1NF relation:

PERSON (SS#, {CAR_LIC#}, {PHONE#})

This relation represents the fact that a person has multiple cars and multiple phones. If a strategy like the second option above is followed, it results in an all-key relation:

PERSON_IN_1NF (SS#, CAR_LIC#, PHONE#)

To avoid introducing any extraneous relationship between CAR_LIC# and PHONE#, all possible combinations of values are represented for every SS#, giving rise to redundancy. This leads to the problems handled by multivalued dependencies and 4NF, which we discuss in Chapter 11. The right way to deal with the two multivalued attributes in PERSON above is to decompose it into two separate relations, using strategy 1 discussed above: P1(SS#, CAR_LIC#) and P2(SS#, PHONE#).

10.3.5 Second Normal Form

Second normal form (2NF) is based on the concept of *full functional dependency*. A functional dependency $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute $A \in X$, $(X - \{A\})$ does *not* functionally determine Y. A functional dependency $X \rightarrow Y$ is a **partial dependency** if some attribute $A \in X$ can be removed from X and the dependency still holds; that is, for some $A \in X$, $(X - \{A\}) \rightarrow Y$. In Figure 10.3b, {SSN, PNUMBER} → HOURS is a full dependency (neither SSN → HOURS nor PNUMBER → HOURS holds). However, the dependency {SSN, PNUMBER} → ENAME is partial because SSN → ENAME holds.

Definition. A relation schema R is in **2NF** if every nonprime attribute A in R is *fully functionally dependent* on the primary key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. The EMP_PROJ relation in Figure 10.3b is in 1NF but is not in 2NF. The nonprime attribute ENAME violates 2NF because of FD2, as do the nonprime attributes PNAME and PLOCATION because of FD3. The functional dependencies FD2 and FD3 make ENAME, PNAME, and PLOCATION partially dependent on the primary key {SSN, PNUMBER} of EMP_PROJ, thus violating the 2NF test.

If a relation schema is not in 2NF, it can be "second normalized" or "2NF normalized" into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. The functional dependencies FD1, FD2, and FD3 in Figure 10.3b hence lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 10.10a, each of which is in 2NF.

10.3.6 Third Normal Form

Third normal form (3NF) is based on the concept of *transitive dependency*. A functional dependency $X \rightarrow Y$ in a relation schema R is a **transitive dependency** if there is a set of

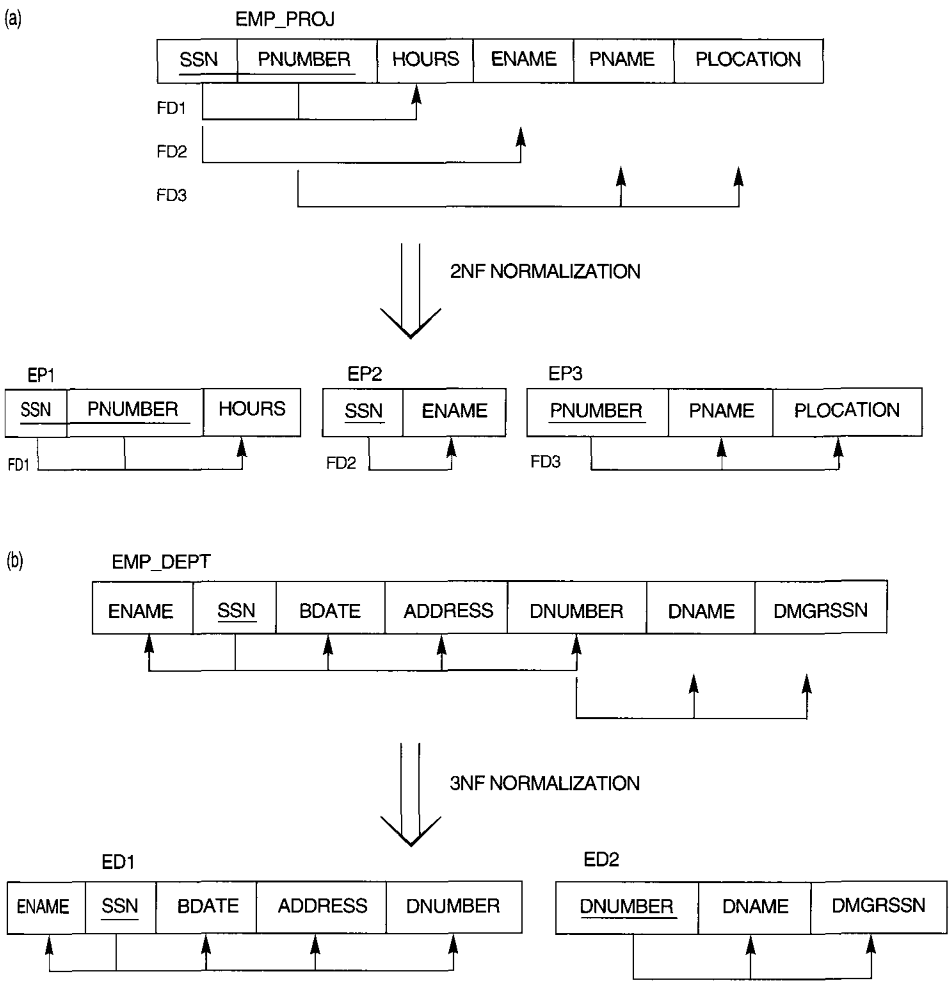


FIGURE 10.10 Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

attributes Z that is neither a candidate key nor a subset of any key of R,[14] and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. The dependency SSN → DMGRSSN is transitive through DNUMBER in EMP_DEPT of Figure 10.3a because both the dependencies SSN → DNUMBER and DNUMBER → DMGRSSN hold *and* DNUMBER is neither a key itself nor a subset of the key of EMP_DEPT. Intuitively, we can see that the dependency of DMGRSSN on DNUMBER is undesirable in EMP_DEPT since DNUMBER is not a key of EMP_DEPT.

Definition. According to Codd's original definition, a relation schema R is in **3NF** if it satisfies 2NF *and* no nonprime attribute of R is transitively dependent on the primary key.

The relation schema EMP_DEPT in Figure 10.3a is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN (and also DNAME) on SSN via DNUMBER. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Figure 10.10b. Intuitively, we see that ED1 and ED2 represent independent entity facts about employees and departments. A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples.

Intuitively, we can see that any functional dependency in which the left-hand side is part (proper subset) of the primary key, or any functional dependency in which the left-hand side is a nonkey attribute is a "problematic" FD. 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations. In terms of the normalization process, it is not necessary to remove the partial dependencies before the transitive dependencies, but historically, 3NF has been defined with the assumption that a relation is tested for 2NF first before it is tested for 3NF. Table 10.1 informally summarizes the three normal forms based on primary keys, the tests used in each case, and the corresponding "remedy" or normalization performed to achieve the normal form.

10.4 General Definitions of Second and Third Normal Forms

In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies, because these types of dependencies cause the update anomalies discussed in Section 10.1.2. The steps for normalization into 3NF relations that we have discussed so far disallow partial and transitive dependencies on the *primary key*. These definitions, however, do not take other candidate keys of a relation, if any, into account. In this section we give the more general definitions of 2NF and 3NF that take *all* candidate keys of a relation into account. Notice that this does not affect the definition of 1NF, since it is independent of keys and functional dependencies. As a general definition of **prime attribute,** an attribute that is part of *any candidate key* will be considered as prime.

14. This is the general definition of transitive dependency. Because we are concerned only with primary keys in this section, we allow transitive dependencies where X is the primary key but Z may be (a subset of) a candidate key.

TABLE 10.1 SUMMARY OF NORMAL FORMS BASED ON PRIMARY KEYS AND CORRESPONDING NORMALIZATION

NORMAL FORM	TEST	REMEDY (NORMALIZATION)
First (1NF)	Relation should have no nonatomic attributes or nested relations.	Form new relations for each nonatomic attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes.) That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Partial and full functional dependencies and transitive dependencies will now be considered *with respect to all candidate keys* of a relation.

10.4.1 General Definition of Second Normal Form

Definition. A relation schema R is in **second normal form** (2NF) if every nonprime attribute A in R is not partially dependent on *any* key of R.[15]

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are *part of* the primary key. If the primary key contains a single attribute, the test need not be applied at all. Consider the relation schema LOTS shown in Figure 10.11a, which describes parcels of land for sale in various counties of a state. Suppose that there are two candidate keys: PROPERTY_ID# and {COUNTY_NAME, LOT#}; that is, lot numbers are unique only within each county, but PROPERTY_ID numbers are unique across counties for the entire state.

Based on the two candidate keys PROPERTY_ID# and {COUNTY_NAME, LOT#}, we know that the functional dependencies FD1 and FD2 of Figure 10.11a hold. We choose PROPERTY_ID# as the primary key, so it is underlined in Figure 10.11a, but no special consideration will

15. This definition can be restated as follows: A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on *every* key of R.

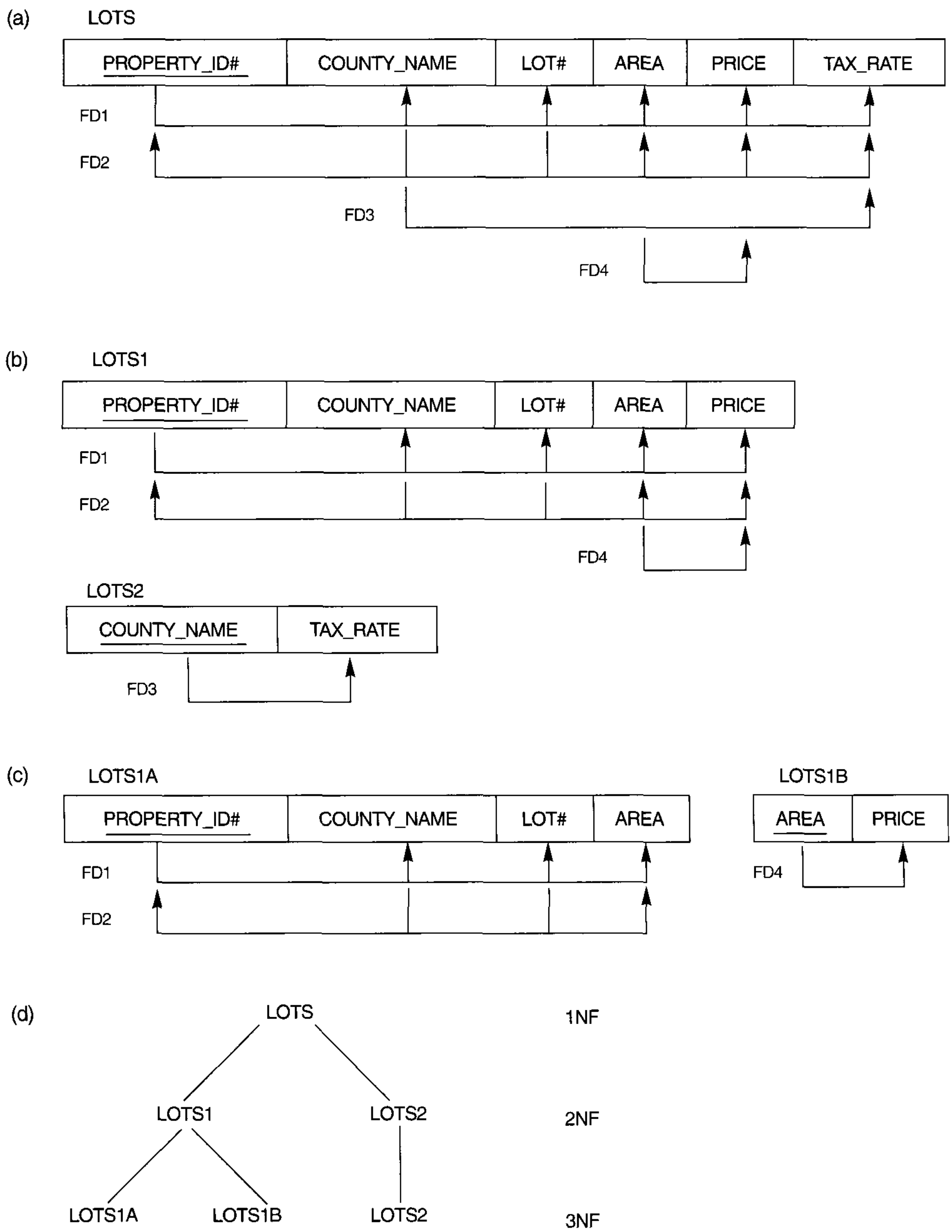


FIGURE 10.11 Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

be given to this key over the other candidate key. Suppose that the following two additional functional dependencies hold in LOTS:

FD3: COUNTY_NAME → TAX_RATE

FD4: AREA → PRICE

In words, the dependency FD3 says that the tax rate is fixed for a given county (does not vary lot by lot within the same county), while FD4 says that the price of a lot is determined by its area regardless of which county it is in. (Assume that this is the price of the lot for tax purposes.)

The LOTS relation schema violates the general definition of 2NF because TAX_RATE is partially dependent on the candidate key {COUNTY_NAME, LOT#}, due to FD3. To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2, shown in Figure 10.11b. We construct LOTS1 by removing the attribute TAX_RATE that violates 2NF from LOTS and placing it with COUNTY_NAME (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2. Both LOTS1 and LOTS2 are in 2NF. Notice that FD4 does not violate 2NF and is carried over to LOTS1.

10.4.2 General Definition of Third Normal Form

Definition. A relation schema R is in **third normal form** (3NF) if, whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

According to this definition, LOTS2 (Figure 10.11b) is in 3NF. However, FD4 in LOTS1 violates 3NF because AREA is not a superkey and PRICE is not a prime attribute in LOTS1. To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B shown in Figure 10.11c. We construct LOTS1A by removing the attribute PRICE that violates 3NF from LOTS1 and placing it with AREA (the left-hand side of FD4 that causes the transitive dependency) into another relation LOTS1B. Both LOTS1A and LOTS1B are in 3NF.

Two points are worth noting about this example and the general definition of 3NF:

- LOTS1 violates 3NF because PRICE is transitively dependent on each of the candidate keys of LOTS1 via the nonprime attribute AREA.
- This general definition can be applied *directly* to test whether a relation schema is in 3NF; it does *not* have to go through 2NF first. If we apply the above 3NF definition to LOTS with the dependencies FD1 through FD4, we find that *both* FD3 and FD4 violate 3NF. We could hence decompose LOTS into LOTS1A, LOTS1B, and LOTS2 directly. Hence the transitive and partial dependencies that violate 3NF can be removed *in any order*.

10.4.3 Interpreting the General Definition of Third Normal Form

A relation schema R violates the general definition of 3NF if a functional dependency $X \rightarrow A$ holds in R that violates *both* conditions (a) and (b) of 3NF. Violating (b) means that

A is a nonprime attribute. Violating (a) means that X is not a superset of any key of R; hence, X could be nonprime or it could be a proper subset of a key of R. If X is nonprime, we typically have a transitive dependency that violates 3NF, whereas if X is a proper subset of a key of R, we have a partial dependency that violates 3NF (and also 2NF). Hence, we can state a **general alternative definition of 3NF** as follows: A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

- It is fully functionally dependent on every key of R.
- It is nontransitively dependent on every key of R.

10.5 Boyce-Codd Normal Form

Boyce-Codd normal form (**BCNF**) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is *not necessarily* in BCNF. Intuitively, we can see the need for a stronger normal form than 3NF by going back to the LOTS relation schema of Figure 10.11a with its four functional dependencies FD1 through FD4. Suppose that we have thousands of lots in the relation but the lots are from only two counties: Dekalb and Fulton. Suppose also that lot sizes in Dekalb County are only 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0 acres, whereas lot sizes in Fulton County are restricted to 1.1, 1.2, . . . , 1.9, and 2.0 acres. In such a situation we would have the additional functional dependency FD5: AREA $\rightarrow$ COUNTY_NAME. If we add this to the other dependencies, the relation schema LOTS1A still is in 3NF because COUNTY_NAME is a prime attribute.

The area of a lot that determines the county, as specified by FD5, can be represented by 16 tuples in a separate relation R(AREA, COUNTY_NAME), since there are only 16 possible AREA values. This representation reduces the redundancy of repeating the same information in the thousands of LOTS1A tuples. BCNF is a *stronger normal form* that would disallow LOTS1A and suggest the need for decomposing it.

Definition. A relation schema R is in **BCNF** if whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.

The formal definition of BCNF differs slightly from the definition of 3NF. The only difference between the definitions of BCNF and 3NF is that condition (b) of 3NF, which allows A to be prime, is absent from BCNF. In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A. Note that FD5 satisfies 3NF in LOTS1A because COUNTY_NAME is a prime attribute (condition b), but this condition does not exist in the definition of BCNF. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY, shown in Figure 10.12a. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.

In practice, most relation schemas that are in 3NF are also in BCNF. Only if $X \rightarrow A$ holds in a relation schema R with X not being a superkey *and* A being a prime attribute will R be in 3NF but not in BCNF. The relation schema R shown in Figure 10.12b illustrates the general case of such a relation. Ideally, relational database design should strive to achieve BCNF or 3NF for every relation schema. Achieving the normalization

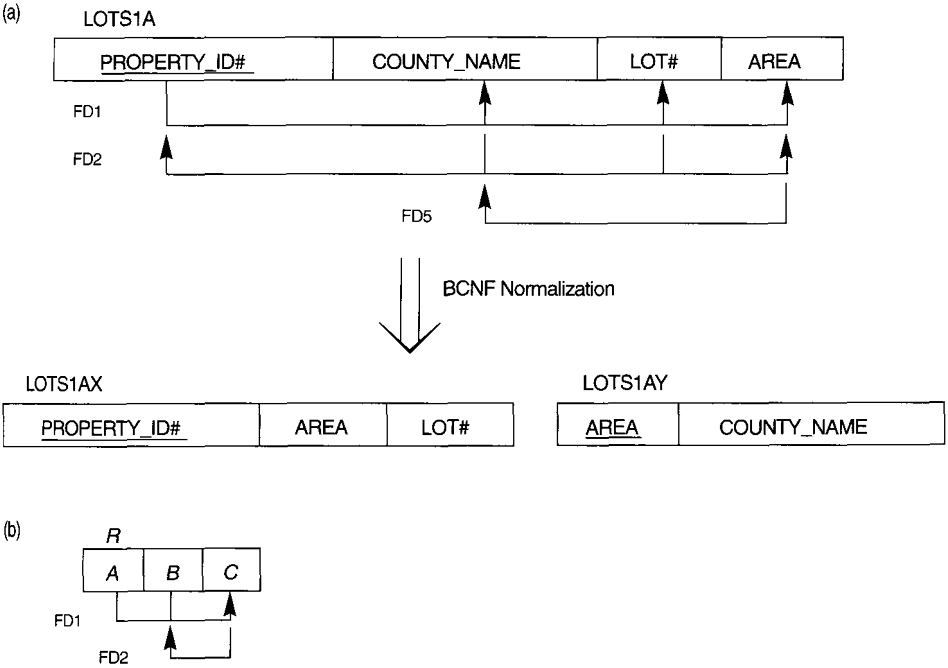


FIGURE 10.12 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

status of just 1NF or 2NF is not considered adequate, since they were developed historically as stepping stones to 3NF and BCNF.

As another example, consider Figure 10.13, which shows a relation TEACH with the following dependencies:

FD1: { STUDENT, COURSE} → INSTRUCTOR

FD2:[16] INSTRUCTOR → COURSE

Note that {STUDENT, COURSE} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 10.12b, with STUDENT as *A*, COURSE as *B*, and INSTRUCTOR as *C*. Hence this relation is in 3NF but not BCNF. Decomposition of this relation schema into two schemas is not straightforward because it may be decomposed into one of the three following possible pairs:

1. {STUDENT, INSTRUCTOR} and {STUDENT, COURSE}.
2. {COURSE, INSTRUCTOR } and {COURSE, STUDENT}.
3. {INSTRUCTOR, COURSE } and {INSTRUCTOR, STUDENT}.

16. This dependency means that "each instructor teaches one course" is a constraint for this application.

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

FIGURE 10.13 A relation TEACH that is in 3NF but not BCNF.

All three decompositions "lose" the functional dependency FD1. The *desirable decomposition* of those just shown is 3, because it will not generate spurious tuples after a join.

A test to determine whether a decomposition is nonadditive (lossless) is discussed in Section 11.1.4 under Property LJ1. In general, a relation not in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations, as is the case in this example. Algorithm 11.3 does that and could be used above to give decomposition 3 for TEACH.

10.6 SUMMARY

In this chapter we first discussed several pitfalls in relational database design using intuitive arguments. We identified informally some of the measures for indicating whether a relation schema is "good" or "bad," and provided informal guidelines for a good design. We then presented some formal concepts that allow us to do relational design in a top-down fashion by analyzing relations individually. We defined this process of design by analysis and decomposition by introducing the process of normalization.

We discussed the problems of update anomalies that occur when redundancies are present in relations. Informal measures of good relation schemas include simple and clear attribute semantics and few nulls in the extensions (states) of relations. A good decomposition should also avoid the problem of generation of spurious tuples as a result of the join operation.

We defined the concept of functional dependency and discussed some of its properties. Functional dependencies specify semantic constraints among the attributes of a relation schema. We showed how from a given set of functional dependencies, additional dependencies can be inferred using a set of inference rules. We defined the concepts of closure and cover related to functional dependencies. We then defined minimal cover of a set of dependencies, and provided an algorithm to compute a minimal cover. We also showed how to check whether two sets of functional dependencies are equivalent.

We then described the normalization process for achieving good designs by testing relations for undesirable types of "problematic" functional dependencies. We provided a treatment of successive normalization based on a predefined primary key in each relation, then relaxed this requirement and provided more general definitions of second normal form (2NF) and third normal form (3NF) that take all candidate keys of a relation into account. We presented examples to illustrate how by using the general definition of 3NF a given relation may be analyzed and decomposed to eventually yield a set of relations in 3NF.

Finally, we presented Boyce-Codd normal form (BCNF) and discussed how it is a stronger form of 3NF. We also illustrated how the decomposition of a non-BCNF relation must be done by considering the nonadditive decomposition requirement.

Chapter 11 presents synthesis as well as decomposition algorithms for relational database design based on functional dependencies. Related to decomposition, we discuss the concepts of *lossless (nonadditive) join* and *dependency preservation*, which are enforced by some of these algorithms. Other topics in Chapter 11 include multivalued dependencies, join dependencies, and fourth and fifth normal forms, which take these dependencies into account.

Review Questions

10.1. Discuss attribute semantics as an informal measure of goodness for a relation schema.

10.2. Discuss insertion, deletion, and modification anomalies. Why are they considered bad? Illustrate with examples.

10.3. Why should nulls in a relation be avoided as far as possible? Discuss the problem of spurious tuples and how we may prevent it.

10.4. State the informal guidelines for relation schema design that we discussed. Illustrate how violation of these guidelines may be harmful.

10.5. What is a functional dependency? What are the possible sources of the information that defines the functional dependencies that hold among the attributes of a relation schema?

10.6. Why can we not infer a functional dependency automatically from a particular relation state?

10.7. What role do Armstrong's inference rules—the three inference rules IR1 through IR3—play in the development of the theory of relational design?

10.8. What is meant by the completeness and soundness of Armstrong's inference rules?

10.9. What is meant by the closure of a set of functional dependencies? Illustrate with an example.

10.10. When are two sets of functional dependencies equivalent? How can we determine their equivalence?

10.11. What is a minimal set of functional dependencies? Does every set of dependencies have a minimal equivalent set? Is it always unique?

10.12. What does the term *unnormalized relation* refer to? How did the normal forms develop historically from first normal form up to Boyce-Codd normal form?

10.13. Define first, second, and third normal forms when only primary keys are considered. How do the general definitions of 2NF and 3NF, which consider all keys of a relation, differ from those that consider only primary keys?

10.14. What undesirable dependencies are avoided when a relation is in 2NF?

10.15. What undesirable dependencies are avoided when a relation is in 3NF?

10.16. Define Boyce-Codd normal form. How does it differ from 3NF? Why is it considered a stronger form of 3NF?

Exercises

10.17. Suppose that we have the following requirements for a university database that is used to keep track of students' transcripts:

a. The university keeps track of each student's name (SNAME), student number (SNUM), social security number (SSN), current address (SCADDR) and phone (SCPHONE), permanent address (SPADDR) and phone (SPPHONE), birth date (BDATE), sex (SEX), class (CLASS) (freshman, sophomore, . . . , graduate), major department (MAJORCODE), minor department (MINORCODE) (if any), and degree program (PROG) (B.A., B.S., . . . , PH.D.). Both SSSN and student number have unique values for each student.

b. Each department is described by a name (DNAME), department code (DCODE), office number (DOFFICE), office phone (DPHONE), and college (DCOLLEGE). Both name and code have unique values for each department.

c. Each course has a course name (CNAME), description (CDESC), course number (CNUM), number of semester hours (CREDIT), level (LEVEL), and offering department (CDEPT). The course number is unique for each course.

d. Each section has an instructor (INAME), semester (SEMESTER), year (YEAR), course (SECCOURSE), and section number (SECNUM). The section number distinguishes different sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, . . . , up to the total number of sections taught during each semester.

e. A grade record refers to a student (SSN), a particular section, and a grade (GRADE).

Design a relational database schema for this database application. First show all the functional dependencies that should hold among the attributes. Then design relation schemas for the database that are each in 3NF or BCNF. Specify the key attributes of each relation. Note any unspecified requirements, and make appropriate assumptions to render the specification complete.

10.18. Prove or disprove the following inference rules for functional dependencies. A proof can be made either by a proof argument or by using inference rules IR1 through IR3. A disproof should be performed by demonstrating a relation instance that satisfies the conditions and functional dependencies in the left-hand side of the inference rule but does not satisfy the dependencies in the right-hand side.

a. $\{W \rightarrow Y, X \rightarrow Z\} \models \{WX \rightarrow Y\}$

b. $\{X \rightarrow Y\}$ and $Y \supseteq Z \models \{X \rightarrow Z\}$

c. $\{X \to Y, X \to W, WY \to Z\} \models \{X \to Z\}$
d. $\{XY \to Z, Y \to W\} \models \{XW \to Z\}$
e. $\{X \to Z, Y \to Z\} \models \{X \to Y\}$
f. $\{X \to Y, XY \to Z\} \models \{X \to Z\}$
g. $\{X \to Y, Z \to W\} \models \{XZ \to YW\}$
h. $\{XY \to Z, Z \to X\} \models \{Z \to Y\}$
i. $\{X \to Y, Y \to Z\} \models \{X \to YZ\}$
j. $\{XY \to Z, Z \to W\} \models \{X \to W\}$

10.19. Consider the following two sets of functional dependencies: $F = \{A \to C, AC \to D, E \to AD, E \to H\}$ and $G = \{A \to CD, E \to AH\}$. Check whether they are equivalent.

10.20. Consider the relation schema EMP_DEPT in Figure 10.3a and the following set G of functional dependencies on EMP_DEPT: G = {SSN → {ENAME, BDATE, ADDRESS, DNUMBER}, DNUMBER → {DNAME, DMGRSSN}}. Calculate the closures $\{\text{SSN}\}^+$ and $\{\text{DNUMBER}\}^+$ with respect to G.

10.21. Is the set of functional dependencies G in Exercise 10.20 minimal? If not, try to find a minimal set of functional dependencies that is equivalent to G. Prove that your set is equivalent to G.

10.22. What update anomalies occur in the EMP_PROJ and EMP_DEPT relations of Figures 10.3 and 10.4?

10.23. In what normal form is the LOTS relation schema in Figure 10.11a with respect to the restrictive interpretations of normal form that take *only the primary key* into account? Would it be in the same normal form if the general definitions of normal form were used?

10.24. Prove that any relation schema with two attributes is in BCNF.

10.25. Why do spurious tuples occur in the result of joining the EMP_PROJ1 and EMP_ LOCS relations of Figure 10.5 (result shown in Figure 10.6)?

10.26. Consider the universal relation $R = \{A, B, C, D, E, F, G, H, I, J\}$ and the set of functional dependencies $F = \{\{A, B\} \to \{C\}, \{A\} \to \{D, E\}, \{B\} \to \{F\}, \{F\} \to \{G, H\}, \{D\} \to \{I, J\}\}$. What is the key for R? Decompose R into 2NF and then 3NF relations.

10.27. Repeat Exercise 10.26 for the following different set of functional dependencies $G = \{\{A, B\} \to \{C\}, \{B, D\} \to \{E, F\}, \{A, D\} \to \{G, H\}, \{A\} \to \{I\}, \{H\} \to \{J\}\}$.

10.28. Consider the following relation:

A	B	C	TUPLE#
10	b1	c1	#1
10	b2	c2	#2
11	b4	c1	#3
12	b3	c4	#4
13	b1	c1	#5
14	b3	c4	#6

a. Given the previous extension (state), which of the following dependencies *may hold* in the above relation? If the dependency cannot hold, explain why *by specifying the tuples that cause the violation*.

i. $A \to B$, ii. $B \to C$, iii. $C \to B$, iv. $B \to A$, v. $C \to A$

b. Does the above relation have a *potential* candidate key? If it does, what is it? If it does not, why not?

10.29. Consider a relation $R(A, B, C, D, E)$ with the following dependencies:

$AB \to C$, $CD \to E$, $DE \to B$

Is AB a candidate key of this relation? If not, is ABD? Explain your answer.

10.30. Consider the relation R, which has attributes that hold schedules of courses and sections at a university; R = {CourseNo, SecNo, OfferingDept, Credit–Hours, CourseLevel, InstructorSSN, Semester, Year, Days_Hours, RoomNo, NoOfStudents}. Suppose that the following functional dependencies hold on R:

{CourseNo} → {OfferingDept, CreditHours, CourseLevel}

{CourseNo, SecNo, Semester, Year} → {Days_Hours, RoomNo, NoOfStudents, InstructorSSN}

{RoomNo, Days_Hours, Semester, Year} → {Instructorssn, CourseNo, SecNo}

Try to determine which sets of attributes form keys of R. How would you normalize this relation?

10.31. Consider the following relations for an order-processing application database at ABC, Inc.

ORDER (<u>O#</u>, Odate, Cust#, Total_amount)

ORDER-ITEM(<u>O#</u>, <u>I#</u>, Qty_ordered, Total_price, Discount%)

Assume that each item has a different discount. The TOTAL_PRICE refers to one item, ODATE is the date on which the order was placed, and the TOTAL_AMOUNT is the amount of the order. If we apply a natural join on the relations ORDER-ITEM and ORDER in this database, what does the resulting relation schema look like? What will be its key? Show the FDs in this resulting relation. Is it in 2NF? Is it in 3NF? Why or why not? (State assumptions, if you make any.)

10.32. Consider the following relation:

CAR_SALE(Car#, Date_sold, Salesman#, Commission%, Discount_amt)

Assume that a car may be sold by multiple salesmen, and hence {CAR#, SALESMAN#} is the primary key. Additional dependencies are

Date_sold → Discount_amt

and

Salesman# → Commission%

Based on the given primary key, is this relation in 1NF, 2NF, or 3NF? Why or why not? How would you successively normalize it completely?

10.33. Consider the following relation for published books:

BOOK (Book_title, Authorname, Book_type, Listprice, Author_affil, Publisher)

Author_affil refers to the affiliation of author. Suppose the following dependencies exist:

Book_title → Publisher, Book_type

Book_type → Listprice

Authorname → Author-affil

a. What normal form is the relation in? Explain your answer.
b. Apply normalization until you cannot decompose the relations further. State the reasons behind each decomposition.

Selected Bibliography

Functional dependencies were originally introduced by Codd (1970). The original definitions of first, second, and third normal form were also defined in Codd (1972a), where a discussion on update anomalies can be found. Boyce-Codd normal form was defined in Codd (1974). The alternative definition of third normal form is given in Ullman (1988), as is the definition of BCNF that we give here. Ullman (1988), Maier (1983), and Atzeni and De Antonellis (1993) contain many of the theorems and proofs concerning functional dependencies.

Armstrong (1974) shows the soundness and completeness of the inference rules IR1 through IR3. Additional references to relational design theory are given in Chapter 11.