# UNIT – 4

## Database Design Theory and Normalization

## 4.1 Informal design guidelines for relation schemas

Before discussing the formal theory of relational database design, we discuss four informal guidelines that may be used as measures to determine the quality of relation schema design:

- Making sure that the semantics of the attributes is clear in the schema.
- Reducing the redundant information in tuples.
- Reducing the NULL values in tuples.
- Disallowing the possibility of generating spurious tuples.

### 4.1.1 Imparting Clear Semantics to Attributes in Relations

Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them. The **semantics** of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.

In general, the easier it is to explain the semantics of the relation, the better the relation schema design will be. To illustrate this, consider Figure 4.1, a simplified version of the COMPANY relational database schema.

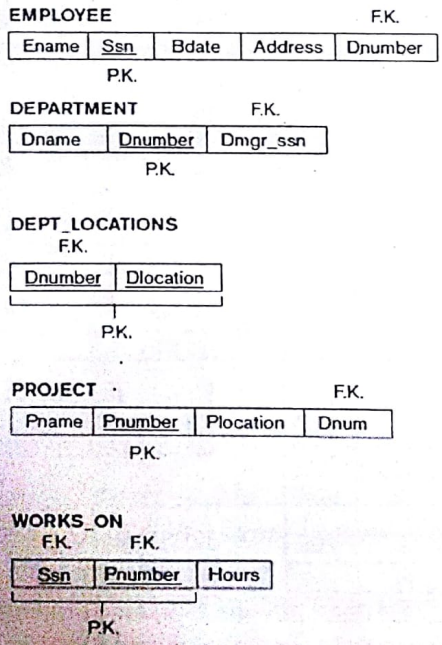



**Figure 4.1:** A simplified COMPANY relational database schema.

Figure 4.2, which presents an example of populated relation states of this schema. The meaning of the EMPLOYEE relation schema is quite simple: Each tuple represents an employee, with values for the employee's name (Ename), Social Security number (Ssn), birth date (Bdate), and address (Address), and the number of the department that the employee works for (Dnumber). The Dnumber attribute is a foreign key that represents an implicit relationship between EMPLOYEE and DEPARTMENT.

The semantics of the DEPARTMENT and PROJECT schemas are also straightforward: Each DEPARTMENT tuple represents a department entity, and each PROJECT tuple represents a project entity. The attribute Dmgr_ssn of DEPARTMENT relates a department to the employee who is its manager, while Dnum of PROJECT relates a project to its controlling department; both are foreign key attributes. The ease with which the meaning of a relation's attributes can be explained is an informal measure of how well the relation is designed.

EMPLOYEE

| Ename | Ssn | Bdate | Address | Dnumber |
|---|---|---|---|---|
| Smith, John B. | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | 5 |
| Wong, Franklin T. | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | 5 |
| Zelaya, Alicia J. | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX | 4 |
| Wallace, Jennifer S. | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | 4 |
| Narayan, Ramesh K. | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | 5 |
| English, Joyce A. | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | 5 |
| Jabbar, Ahmad V. | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | 4 |
| Borg, James E. | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | 1 |

DEPARTMENT

| Dname | Dnumber | Dmgr_ssn |
|---|---|---|
| Research | 5 | 333445555 |
| Administration | 4 | 987654321 |
| Headquarters | 1 | 888665555 |

DEPT_LOCATIONS

| Dnumber | Dlocation |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |

WORKS_ON

| Ssn | Pnumber | Hours |
|---|---|---|
| 123456789 | 1 | 32.5 |
| 123456789 | 2 | 7.5 |
| 666884444 | 3 | 40.0 |
| 453453453 | 1 | 20.0 |
| 453453453 | 2 | 20.0 |
| 333445555 | 2 | 10.0 |
| 333445555 | 3 | 10.0 |
| 333445555 | 10 | 10.0 |
| 333445555 | 20 | 10.0 |
| 999887777 | 30 | 30.0 |
| 999887777 | 10 | 10.0 |
| 987987987 | 10 | 35.0 |
| 987987987 | 30 | 5.0 |
| 987654321 | 30 | 20.0 |
| 987654321 | 20 | 15.0 |
| 888665555 | 20 | Null |

PROJECT

| Pname | Pnumber | Plocation | Dnum |
|---|---|---|---|
| ProductX | 1 | Bellaire | 5 |
| ProductY | 2 | Sugarland | 5 |
| ProductZ | 3 | Houston | 5 |
| Computerization | 10 | Stafford | 4 |
| Reorganization | 20 | Houston | 1 |
| Newbenefits | 30 | Stafford | 4 |

**Figure 4.2 Sample database state for the relational database schema in Figure 4.1.**

Each tuple in DEPT_LOCATIONS gives a department number (Dnumber) and one of the locations of the department (Dlocation). Each tuple in WORKS_ON gives an employee Social Security number (Ssn), the project number of one of the projects that the employee works on (Pnumber), and the number of hours per week that the employee works on that project (Hours). However, both schemas have a well-defined and unambiguous interpretation. The schema DEPT_LOCATIONS represents a multivalued attribute of DEPARTMENT,whereas WORKS_ON represents an M:N relationship between EMPLOYEE and PROJECT. Hence, all the relation schemas in Figure 4.1 may be considered as easy to explain and therefore good from the standpoint of having clear semantics.We can thus formulate the following informal design guideline.

**Guideline 1:** Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

**Examples of Violating Guideline 1.** The relation schemas in Figures 4.3(a) and 4.3(b) also have clear semantics. A tuple in the EMP_DEPT relation schema in Figure 4.3(a) represents a single employee but includes additional information—namely, the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager. For the EMP_PROJ relation in Figure 4.3(b), each tuple relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location (Plocation). Although there is nothing wrong logically with these two relations, they violate Guideline 1 by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments, and EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship. Hence, they fare poorly against the above measure of design quality.

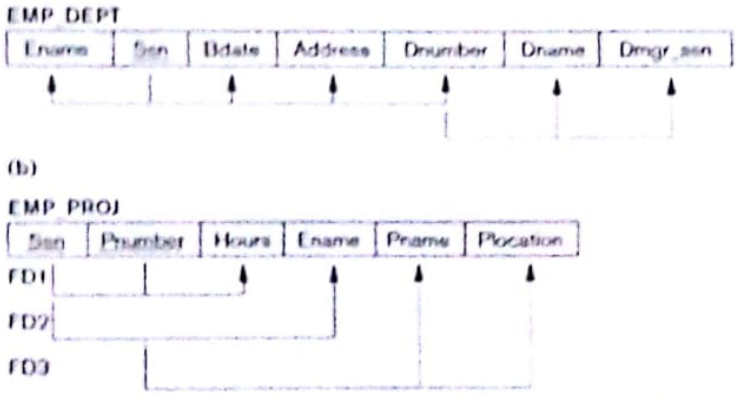



**Figure 4.3** Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

### 4.1.2 Redundant Information in Tuples and Update Anomalies

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has a significant effect on storage space. For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in Figure 4.2 with that for an EMP_DEPT base relation in Figure 4.4, which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT. In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department. In contrast, each department's information appears only once in the DEPARTMENT relation in Figure 4.2. Only the department number (Dnumber) is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key.

Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Redundancy

**EMP_DEPT**

| Ename | Ssn | Bdate | Address | Dnumber | Dname | Dmgr_ssn |
|---|---|---|---|---|---|---|
| Smith, John B. | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | 5 | Research | 333445555 |
| Wong, Franklin T. | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | 5 | Research | 333445555 |
| Zelaya, Alicia J. | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX | 4 | Administration | 987654321 |
| Wallace, Jennifer S. | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | 4 | Administration | 987654321 |
| Narayan, Ramesh K. | 666884444 | 1962-09-15 | 975 FireOak, Humble, TX | 5 | Research | 333445555 |
| English, Joyce A. | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | 5 | Research | 333445555 |
| Jabbar, Ahmad V. | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | 4 | Administration | 987654321 |
| Borg, James E. | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | 1 | Headquarters | 888665555 |

Redundancy Redundancy

**EMP_PROJ**

| Ssn | Pnumber | Hours | Ename | Pname | Plocation |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | Smith, John B. | ProductX | Bellaire |
| 123456789 | 2 | 7.5 | Smith, John B. | ProductY | Sugarland |
| 666884444 | 3 | 40.0 | Narayan, Ramesh K. | ProductZ | Houston |
| 453453453 | 1 | 20.0 | English, Joyce A. | ProductX | Bellaire |
| 453453453 | 2 | 20.0 | English, Joyce A. | ProductY | Sugarland |
| 333445555 | 2 | 10.0 | Wong, Franklin T. | ProductY | Sugarland |
| 333445555 | 3 | 10.0 | Wong, Franklin T. | ProductZ | Houston |
| 333445555 | 10 | 10.0 | Wong, Franklin T. | Computerization | Stafford |
| 333445555 | 20 | 10.0 | Wong, Franklin T. | Reorganization | Houston |
| 999887777 | 30 | 30.0 | Zelaya, Alicia J. | Newbenefits | Stafford |
| 999887777 | 10 | 10.0 | Zelaya, Alicia J. | Computerization | Stafford |
| 987987987 | 10 | 35.0 | Jabbar, Ahmad V. | Computerization | Stafford |
| 987987987 | 30 | 5.0 | Jabbar, Ahmad V. | Newbenefits | Stafford |
| 987654321 | 30 | 20.0 | Wallace, Jennifer S. | Newbenefits | Stafford |
| 987654321 | 20 | 15.0 | Wallace, Jennifer S. | Reorganization | Houston |
| 888665555 | 20 | Null | Borg, James E. | Reorganization | Houston |

**Figure 4.4** Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 15.2. These may be stored as base relations for performance reasons.

**Insertion Anomalies.** Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT. In the design of Figure 4.2, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.
- It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because Ssn is its primary key.Moreover, when the first employee is assigned to that department, we do not need this tuple with NULL values any more. This problem does not occur in the design of Figure 4.2 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

**Deletion Anomalies.** The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database. This problem does not occur in the database of Figure 4.2 because DEPARTMENT tuples are stored separately.

**Modification Anomalies.** In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong. It is easy to see that these three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates that can be avoided; hence, we can state the next guideline as follows.

**Guideline 2:** Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly. The second guideline is consistent with and, in a way, a restatement of the first guideline. We can also see the need for a more formal approach to evaluating whether a design meets these guidelines. It is important to note that these guidelines may sometimes have to be violated in order to improve the performance of certain queries. If EMP_DEPT is used as a stored relation in addition to the base relations of EMPLOYEE and DEPARTMENT, the anomalies in EMP_DEPT

must be noted and accounted for. This way, whenever the base relation is updated, we do not end up with inconsistencies. In general, it is advisable to use anomaly-free base relations.

### 4.1.3 NULL Values in Tuples

In some schema designs we may group many attributes together into a "fat" relation. If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level. Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable. Moreover, NULLs can have multiple interpretations, such as the following:

- The attribute does not apply to this tuple. For example, Visa_status may not apply to U.S. students.
- The attribute value for this tuple is unknown. For example, the Date_of_birth may be unknown for an employee.
- The value is known but absent; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet. Having the same representation for all NULLs compromises the different meanings they may have. Therefore, we may state another guideline.

**Guideline 3:** As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation. Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns (with the appropriate key columns). For example, if only 15 percent of employees have individual offices, there is little justification for including an attribute Office_number in the EMPLOYEE relation; rather, a relation EMP_OFFICES(Essn, Office_number) can be created to include tuples for only the employees with individual offices.

### 4.1.4 Generation of Spurious Tuples

Consider the two relation schemas EMP_LOCS and EMP_PROJ1 in Figure 4.5(a), which can be used instead of the single EMP_PROJ relation in Figure 4.3(b). A tuple in EMP_LOCS means that the employee whose name is Ename works on some project whose location is Plocation. A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation. Figure 4.5(b) shows relation states of EMP_LOCS and EMP_PROJ1 corresponding to the EMP_PROJ relation in Figure 4.4, which are obtained by applying the appropriate PROJECT (π) operations to EMP_PROJ.

(a)

EMP_LOCS

| Ename | Plocation |
|---|---|

PK

EMP_PROJ1

| Ssn | Pnumber | Hours | Pname | Plocation |
|---|---|---|---|---|

PK

(b)

EMP_LOCS

| Ename | Plocation |
|---|---|
| Smith, John B. | Bellaire |
| Smith, John B. | Sugarland |
| Narayan, Ramesh K. | Houston |
| English, Joyce A. | Bellaire |
| English, Joyce A. | Sugarland |
| Wong, Franklin T. | Sugarland |
| Wong, Franklin T. | Houston |
| Wong, Franklin T. | Stafford |
| Zelaya, Alicia J. | Stafford |
| Jabbar, Ahmad V. | Stafford |
| Wallace, Jennifer S. | Stafford |
| Wallace, Jennifer S. | Houston |
| Borg, James E. | Houston |

EMP_PROJ1

| Ssn | Pnumber | Hours | Pname | Plocation |
|---|---|---|---|---|
| 123456789 | 1 | 32.5 | ProductX | Bellaire |
| 123456789 | 2 | 7.5 | ProductY | Sugarland |
| 666884444 | 3 | 40.0 | ProductZ | Houston |
| 453453453 | 1 | 20.0 | ProductX | Bellaire |
| 453453453 | 2 | 20.0 | ProductY | Sugarland |
| 333445555 | 2 | 10.0 | ProductY | Sugarland |
| 333445555 | 3 | 10.0 | ProductZ | Houston |
| 333445555 | 10 | 10.0 | Computerization | Stafford |
| 333445555 | 20 | 10.0 | Reorganization | Houston |
| 999887777 | 30 | 30.0 | Newbenefits | Stafford |
| 999887777 | 10 | 10.0 | Computerization | Stafford |
| 987987987 | 10 | 35.0 | Computerization | Stafford |
| 987987987 | 30 | 5.0 | Newbenefits | Stafford |
| 987654321 | 30 | 20.0 | Newbenefits | Stafford |
| 987654321 | 20 | 15.0 | Reorganization | Houston |
| 888665555 | 20 | NULL | Reorganization | Houston |

**Figure 4.5:** Particularly poor design for the EMP_PROJ relation in Figure 4.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 4.4 onto the relations EMP_LOCS and EMP_PROJ1.

Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS. If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ. In Figure 4.6, the result of applying the join to only the tuples above the dashed lines in Figure 4.5(b) is shown (to reduce the size of the resulting relation). Additional tuples that were not in EMP_PROJ are called **spurious tuples** because they represent spurious information that is not valid. The spurious tuples are marked by asterisks (*) in Figure 4.6.

| Ssn | Pnumber | Hours | Pname | Plocation | Ename |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | ProductX | Bellaire | Smith, John B. |
| * 123456789 | 1 | 32.5 | ProductX | Bellaire | English, Joyce A. |
| 123456789 | 2 | 7.5 | ProductY | Sugarland | Smith, John B. |
| * 123456789 | 2 | 7.5 | ProductY | Sugarland | English, Joyce A. |
| * 123456789 | 2 | 7.5 | ProductY | Sugarland | Wong, Franklin T. |
| 666884444 | 3 | 40.0 | ProductZ | Houston | Narayan, Ramesh K. |
| * 666884444 | 3 | 40.0 | ProductZ | Houston | Wong, Franklin T. |
| * 453453453 | 1 | 20.0 | ProductX | Bellaire | Smith, John B. |
| 453453453 | 1 | 20.0 | ProductX | Bellaire | English, Joyce A. |
| * 453453453 | 2 | 20.0 | ProductY | Sugarland | Smith, John B. |
| 453453453 | 2 | 20.0 | ProductY | Sugarland | English, Joyce A. |
| * 453453453 | 2 | 20.0 | ProductY | Sugarland | Wong, Franklin T. |
| * 333445555 | 2 | 10.0 | ProductY | Sugarland | Smith, John B. |
| * 333445555 | 2 | 10.0 | ProductY | Sugarland | English, Joyce A. |
| 333445555 | 2 | 10.0 | ProductY | Sugarland | Wong, Franklin T. |
| * 333445555 | 3 | 10.0 | ProductZ | Houston | Narayan, Ramesh K. |
| 333445555 | 3 | 10.0 | ProductZ | Houston | Wong, Franklin T. |
| 333445555 | 10 | 10.0 | Computerization | Stafford | Wong, Franklin T. |
| * 333445555 | 20 | 10.0 | Reorganization | Houston | Narayan, Ramesh K. |
| 333445555 | 20 | 10.0 | Reorganization | Houston | Wong, Franklin T. |

**Figure 4.6:** Result of applying NATURAL JOIN to the tuples above the dashed lines in EMP_PROJ1 and EMP_LOCS of Figure 4.5. Generated spurious tuples are marked by asterisks.

**Guideline 4:** Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

## 4.2 Functional Dependencies

The single most important concept in relational schema design theory is that of a functional dependency.

### 4.2.1 Definition of Functional Dependency

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has n attributes A1, A2, ..., An; let us think of the whole database as being described by a single **universal** relation schema R = {A1, A2, ... , An}.We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies.

**Definition.** A **functional dependency,** denoted by X →Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r

of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or **functionally**) determine the values of the Y component. We also say that there is a functional dependency from X to Y, or that Y is **functionally dependent** on X. The abbreviation for functional dependency is **FD** or **f.d.** The set of attributes X is called the **left-hand side** of the FD, and Y is called the **right-hand side.**

Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value.

- If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R)—that is, X is a **candidate key** of R—this implies that X→Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X). If X is a candidate key of R, then X→R.
- If X→Y in R, this does not say whether or not Y→X in R.

A functional dependency is a property of the **semantics** or **meaning of the attributes**. The database designers will use their understanding of the semantics of the attributes of R—that is, how they relate to one another—to specify the functional dependencies that should hold on all relation states (extensions) r of R. Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint. Relation extensions r(R) that satisfy the functional dependency constraints are called **legal relation states** (or **legal extensions**) of R. Hence, the main use of functional dependencies is to describe further a relation schema R by specifying constraints on its attributes that must hold at all times.

Certain FDs can be specified without referring to a specific relation, but as a property of those attributes given their commonly understood meaning. For example, {State, Driver_license_number} → Ssn should hold for any adult in the United States and hence should hold whenever these attributes appear in a relation. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes. For example, the FD Zip_code → Area_code used to exist as a relationship between postal codes and telephone number codes in the United States, but with the proliferation of telephone area codes it is no longer true.

Consider the relation schema EMP_PROJ in Figure 4.3(b); from the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn→ Ename
b. Pnumber → {Pname, Plocation}
c. {Ssn, Pnumber} → Hours

These functional dependencies specify that (a) the value of an employee's Social Security number (Ssn) uniquely determines the employee name (Ename), (b) the value of a project's number (Pnumber) uniquely determines the project name (Pname) and location (Plocation), and (c) a combination of Ssn and Pnumber values uniquely determines the number of hours the employee currently works on the project per week (Hours). Alternatively,we say that Ename is functionally determined by (or functionally dependent on) Ssn, or given a value of Ssn, we know the value of Ename, and so on.

A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R. Therefore, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R. For example, Figure 4.7 shows a particular state of the TEACH relation schema. Although at first glance we may think that Text → Course, we cannot confirm this unless we know that it is true for all possible legal states of TEACH. It is, however, sufficient to demonstrate a single counterexample to disprove a functional dependency. For example, because 'Smith' teaches both 'Data Structures' and 'Data Management,' we can conclude that Teacher does not functionally determine Course.

Given a populated relation, one cannot determine which FDs hold and which do not unless the meaning of and the relationships among the attributes are known. All one can say is that a certain FD may exist if it holds in that particular extension. One cannot guarantee its existence until the meaning of the corresponding attributes is clearly understood. One can, however, emphatically state that a certain FD does not hold if there are tuples that show the violation of such an FD.

**TEACH**

| Teacher | Course | Text |
|---|---|---|
| Smith | Data Structures | Bartram |
| Smith | Data Management | Martin |
| Hall | Compilers | Hoffman |
| Brown | Data Structures | Horowitz |

**Figure 4.7:** A relation state of TEACH with a possible functional dependency TEXT →COURSE. However, TEACHER →COURSE is ruled out.

Figure 4.3 introduces a **diagrammatic notation** for displaying FDs: Each FD is displayed as a horizontal line. The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD, while the right-hand-side attributes are connected by the lines with arrows pointing toward the attributes. We denote by F the set of functional dependencies that are specified on relation schema R. Typically, the schema designer specifies the functional dependencies that are semantically obvious; usually, however, numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F. Those other dependencies can be inferred or deduced from the FDs in F.

## 4.3 Normal Forms Based on Primary Keys

Having introduced functional dependencies, we are now ready to use them to specify some aspects of the semantics of relation schemas. We assume that a set of functional dependencies is given for each relation, and that each relation has a designated primary key; this information combined with the tests (conditions) for normal forms drives the *normalization process* for relational schema design. Most practical relational design projects take one of the following two approaches:

- Perform a conceptual schema design using a conceptual model such as ER or EER and map the conceptual design into a set of relations.
- Design the relations based on external knowledge derived from an existing implementation of files or forms or reports

Following either of these approaches, it is then useful to evaluate the relations for goodness and decompose them further as needed to achieve higher normal forms, using the normalization theory.

### 4.3.1 Normalization of Relations

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis*. Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively.

**Normalization of data** can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies. It can be considered as a "filtering" or "purification" process to make the design have successively better quality. Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties. Thus, the normalization procedure provides database designers with the following:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree.

**Definition.** The **normal form** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Normal forms, when considered *in isolation* from other factors, do not guarantee a good database design. It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF. Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

### 4.3.2 Definitions of Keys and Attributes Participating in Keys

**Definition.** A **superkey** of a relation schema $R = \{A_1, A_2, \ldots, A_n\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples $t_1$ and $t_2$ in any legal relation state $r$ of $R$ will have $t_1[S] = t_2[S]$. A **key** $K$ is a superkey with the additional property that removal of any attribute from $K$ will cause $K$ not to be a superkey any more.

The difference between a key and a superkey is that a key has to be *minimal*; that is, if we have a key $K = \{A_1, A_2, \ldots, A_k\}$ of $R$, then $K - \{A_i\}$ is not a key of $R$ for any $A_i$, $1 \leq i \leq k$. In Figure 15.1, {Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.

If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey. In Figure 4.1, {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key.

**Definition.** An attribute of relation schema $R$ is called a **prime attribute** of $R$ if it is a member of *some candidate key* of $R$. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key. In Figure 4.1, both Ssn and Pnumber are prime attributes of WORKS_ON, whereas other attributes of WORKS_ON are nonprime.

### 4.3.3 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; 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

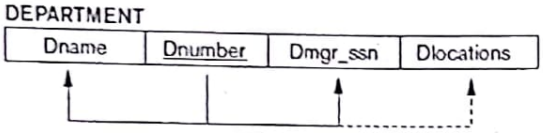
attribute in a tuple must be a *single value* from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a *single tuple*. In other words, 1NF disallows *relations within relations* or *relations as attribute values within tuples*. The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**. Consider the DEPARTMENT relation schema shown in Figure 4.1, whose primary key is Dnumber, and suppose that we extend it by including the Dlocations attribute as shown in Figure 4.8(a).

We assume that each department can have *a number of* locations. The DEPARTMENT schema and a sample relation state are shown in Figure 4.8. As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure 4.8(b). 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.

(a)

DEPARTMENT




(b)

DEPARTMENT

| Dname | Dnumber | Dmgr_ssn | Dlocations |
|---|---|---|---|
| Research | 5 | 333445555 | (Bellaire, Sugarland, Houston) |
| Administration | 4 | 987654321 | (Stafford) |
| Headquarters | 1 | 888665555 | (Houston) |

(c)

DEPARTMENT

| Dname | Dnumber | Dmgr_ssn | Dlocation |
|---|---|---|---|
| Research | 5 | 333445555 | Bellaire |
| Research | 5 | 333445555 | Sugarland |
| Research | 5 | 333445555 | Houston |
| Administration | 4 | 987654321 | Stafford |
| Headquarters | 1 | 888665555 | Houston |

**Figure 4.8** Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

In either case, the DEPARTMENT relation in Figure 4.8 is not in 1NF. 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 4.2. 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 4.8(c). 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 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 4.9 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 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 ( ).

Notice that Ssn is the primary key of the EMP_PROJ relation in Figures 4.9(a) 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, as shown in Figure 4.9 (c).

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.

(a)

EMP_PROJ

| | | Projs | |
|---|---|---|---|
| Ssn | Ename | 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 4.9** Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample 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.

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 strategy 2 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. The right way to deal with the two multivalued attributes in PERSON shown previously is to decompose it into two separate relations, using strategy 1 discussed above: P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

### 4.3.4 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 15.3(b), {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 4.3(b) 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. Therefore, the functional dependencies FD1, FD2, and FD3 in Figure 4.3(b) lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 4.10 (a), each of which is in 2NF.

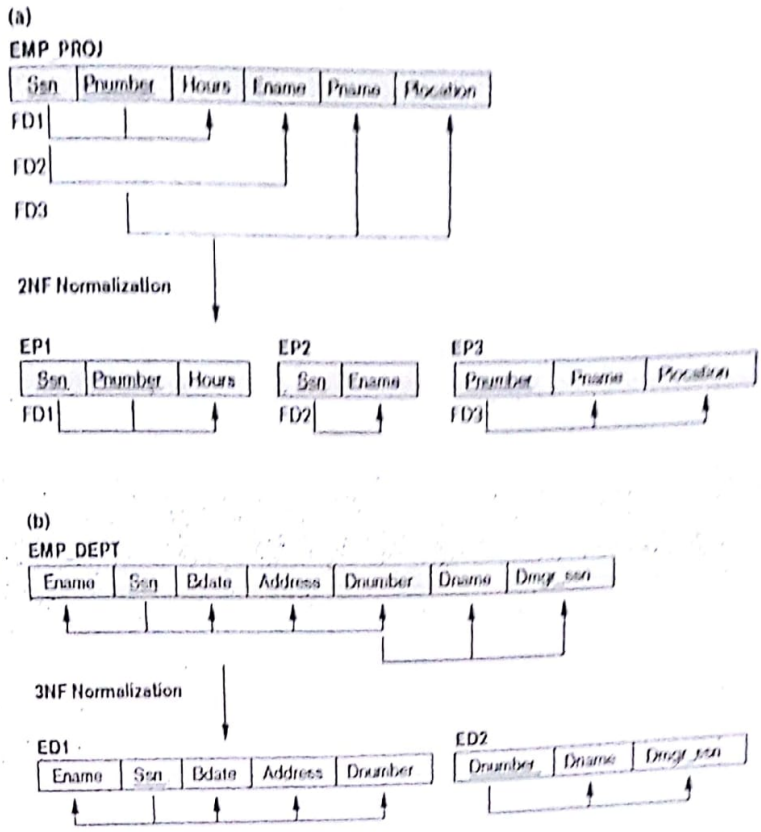



**Figure 4.10** Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

### 4.3.5 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 exists a set of attributes $Z$ in $R$ that is neither a candidate key nor a subset of any key of $R$,10 and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. The dependency Ssn→Dmgr_ssn is transitive through Dnumber in EMP_DEPT in Figure 4.3(a), because both the dependencies Ssn → Dnumber and Dnumber →Dmgr_ssn 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 Dmgr_ssn 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 4.3(a) is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (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 4.10(b). 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 (a 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 4.1 informally summarizes the three normal forms based on primary keys, the tests used in each case, and the corresponding *remedy* or normalizationperformed to achieve the normal form.

| Normal Form | Test | Remedy (Normalization) |
|---|---|---|
| First (1NF) | Relation should have no multivalued attributes or nested relations. | Form new relations for each multivalued 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). |

**Table 4.1** Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

## 4.4 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. The steps for normalization into 3NF relations that we have discussed so far disallow partial and transitive dependencies on the *primary key*.

The normalization procedure described so far is useful for analysis in practical situations for a given database where primary keys have already been defined. 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. Partial and full functional dependencies and transitive dependencies will now be considered *with respect to all candidate keys* of a relation.

### 4.4.1 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.

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 4.11(a), 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#}, the functional dependencies FD1 and FD2 in Figure 4.11(a) hold.We choose Property_id# as the primary key, so it is underlined in Figure 4.11(a), but no special consideration will 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 4.11(b). 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.

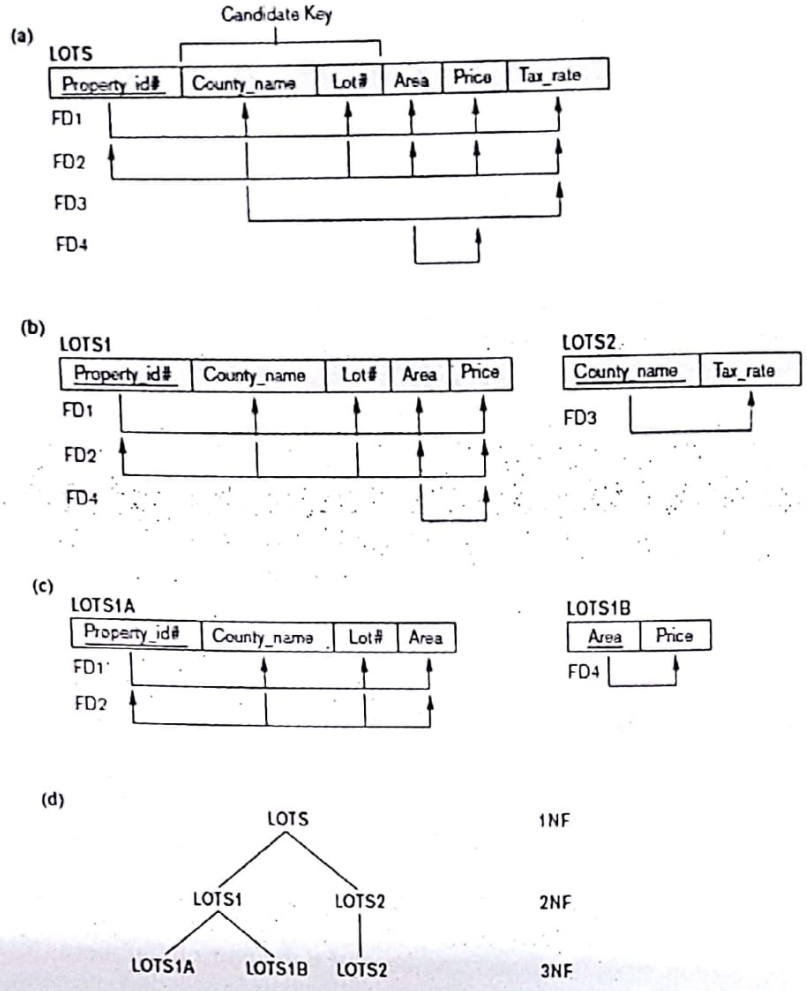



**Figure 4.11** Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

### 4.4.2 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 4.11(b)) 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 4.11(c).We construct LOTS1A by removing the attribute Price that violates 3NF

from LOTS1 and placing it with Area (the lefthand 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. Therefore, we could decompose LOTS into LOTS1A, LOTS1B, and LOTS2 directly. Hence, the transitive and partial dependencies that violate 3NF can be removed *in any order*.

### 4.4.3 Interpreting the General Definition of Third Normal Form

A relation schema $R$ violates the general definition of 3NF if functional dependencies $X \rightarrow A$ holds in $R$ that does not meet either condition—meaning that it violates *both* conditions (a) and (b) of 3NF. This can occur due to two types of problematic functional dependencies:

- A nonprime attribute determines another nonprime attribute. Here we typically have a transitive dependency that violates 3NF.
- A proper subset of a key of $R$ functionally determines a nonprime attribute. Here we have a partial dependency that violates 3NF (and also 2NF).

Therefore, we can state a **general alternative definition of 3NF** as follows:

**Alternative Definition.** 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 $R$.

## 4.5 Other Normal forms

### 4.5.1 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 in Figure 4.11(a) 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→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 (see Figure 4.12). 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*.

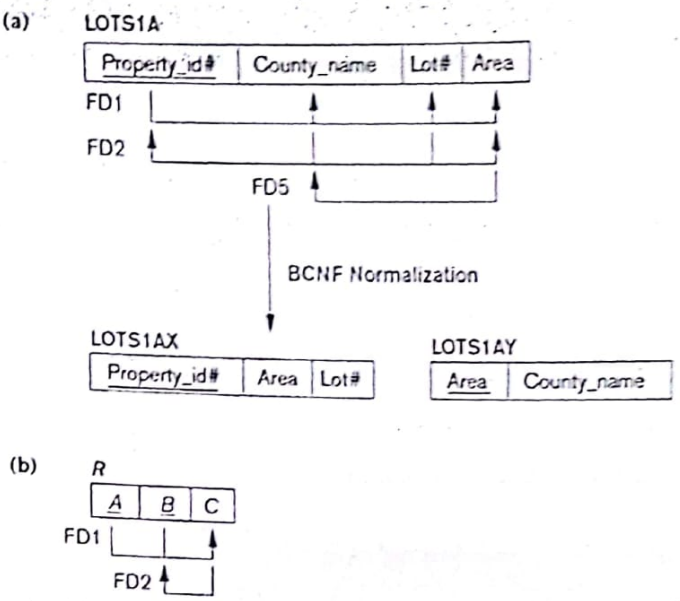



**Figure 4.12** Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows *A* to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF. 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 4.12(a). 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 4.12 (b) 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 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 4.13, which shows a relation TEACH with the following dependencies:

FD1: {Student, Course} →Instructor
FD2: Instructor →Course

Note that {Student, Course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 4.12 (b), 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}.

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 (or lossless) is discussed later under Property NJB. In general, a relation not in BCNF should be decomposed so as to meet this property.

We make sure that we meet this property, because nonadditive decomposition is a must during normalization. We may have to possibly forgo the preservation of all functional dependencies in the decomposed relations, as is the case in this example.

Below algorithm does that and could be used above to give decomposition 3 for TEACH, which yields two relations in BCNF as:

(Instructor, Course) and (Instructor, Student)

Note that if we designate (Student, Instructor) as a primary key of the relation TEACH, the FD Instructor →Course causes a partial (non-full-functional) dependency of Course on a part of this key. This FD may be removed as a part of second normalization yielding exactly the same two relations in the result. This is an example of a case where we may reach the same ultimate BCNF design via alternate paths of normalization.

**Algorithm:** Relational Decomposition into BCNF with Nonadditive Join Property
**Input:** A universal relation $R$ and a set of functional dependencies $F$ on the attributes of $R$.
1. Set $D := \{R\}$;
2. While there is a relation schema $Q$ in $D$ that is not in BCNF do
{
choose a relation schema $Q$ in $D$ that is not in BCNF;
find a functional dependency $X \rightarrow Y$ in $Q$ that violates BCNF;
replace $Q$ in $D$ by two relation schemas $(Q - Y)$ and $(X \cup Y)$;
};

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 |
| Narayan | Operating Systems | Ammar |

**Figure 4.13:** A relation TEACH that is in 3NF but not BCNF.

### 4.5.2 Multivalued Dependency and Fourth Normal Form

In this section, we discuss the concept of *multivalued dependency* (MVD) and define *fourth normal form*, which is based on this dependency. Multivalued dependencies are a consequence of first normal form (1NF), which disallows an attribute in a tuple to have a *set of values*, and the accompanying process of converting an unnormalized relation into 1NF. If we have two or more multivalued *independent* attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

For example, consider the relation EMP shown in Figure 4.14(a). A tuple in this EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname. An employee may work on several projects and may have several dependents, and the employee's projects and dependents are independent of one another. To keep the relation state consistent, and to avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project. This constraint is specified as a multivalued dependency on the EMP relation, which we

define in this section. Informally, whenever two *independent* 1:*N* relationships *A*:*B* and *A*:*C* are mixed in the same relation, *R*(*A*, *B*, *C*), an MVD may arise.

**Figure 4.14**

Fourth and fifth normal forms.
(a) The EMP relation with two MVDs: Ename $\twoheadrightarrow$ Pname and Ename $\twoheadrightarrow$ Dname.
(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.
(c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(*R*$_1$, *R*$_2$, *R*$_3$).
(d) Decomposing the relation SUPPLY into the 5NF relations *R*$_1$, *R*$_2$, *R*$_3$.

(a) EMP

| Ename | Pname | Dname |
|---|---|---|
| Smith | X | John |
| Smith | Y | Anna |
| Smith | X | Anna |
| Smith | Y | John |

(b) EMP_PROJECTS

| Ename | Pname |
|---|---|
| Smith | X |
| Smith | Y |

EMP_DEPENDENTS

| Ename | Dname |
|---|---|
| Smith | John |
| Smith | Anna |

(c) SUPPLY

| Sname | Part_name | Proj_name |
|---|---|---|
| Smith | Bolt | ProjX |
| Smith | Nut | ProjY |
| Adamsky | Bolt | ProjY |
| Walton | Nut | ProjZ |
| Adamsky | Nail | ProjX |
| Adamsky | Bolt | ProjX |
| Smith | Bolt | ProjY |

(d) $R_1$

| Sname | Part_name |
|---|---|
| Smith | Bolt |
| Smith | Nut |
| Adamsky | Bolt |
| Walton | Nut |
| Adamsky | Nail |

$R_2$

| Sname | Proj_name |
|---|---|
| Smith | ProjX |
| Smith | ProjY |
| Adamsky | ProjY |
| Walton | ProjZ |
| Adamsky | ProjX |

$R_3$

| Part_name | Proj_name |
|---|---|
| Bolt | ProjX |
| Nut | ProjY |
| Bolt | ProjY |
| Nut | ProjZ |
| Nail | ProjX |

**Formal Definition of Multivalued Dependency**

**Definition.** A multivalued dependency $X \twoheadrightarrow Y$ specified on relation schema *R*, where *X* and *Y* are both subsets of *R*, specifies the following constraint on any relation state *r* of *R*: If two tuples *t*1 and *t*2 exist in *r* such that *t*1[X] = *t*2[X], then two tuples *t*3 and *t*4 should also exist in *r* with the following properties,where we use *Z* to denote $(R - (X \cup Y))$:

- *t*3[*X*] = *t*4[*X*] = *t*1[*X*] = *t*2[*X*].
- *t*3[*Y*] = *t*1[*Y*] and *t*4[*Y*] = *t*2[*Y*].
- *t*3[*Z*] = *t*2[*Z*] and *t*4[*Z*] = *t*1[*Z*].

Whenever $X \twoheadrightarrow Y$ holds, we say that *X* **multidetermines** *Y*. Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in *R*, so does $X \twoheadrightarrow Z$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$, and therefore it is sometimes written as $X \twoheadrightarrow Y|Z$.

An MVD X →→ Y in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$. For example, the relation EMP_PROJECTS in Figure 4.14(b) has the trivial MVD Ename →→ Pname. An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**.A trivial MVD will hold in *any* relation state *r* of *R*; it is called trivial because it does not specify any significant or meaningful constraint on *R*.

If we have a *nontrivial MVD* in a relation, we may have to repeat values redundantly in the tuples. In the EMP relation of Figure 4.14(a), the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname). This redundancy is clearly undesirable. However, the EMP schema is in BCNF because *no* functional dependencies hold in EMP. Therefore, we need to define a fourth normal form that is stronger than BCNF and disallows relation schemas such as EMP. Notice that relations containing nontrivial MVDs tend to be **all-key relations**—that is, their key is all their attributes taken together. Furthermore, it is rare that such all-key relations with a combinatorial occurrence of repeated values would be designed in practice. However, recognition of MVDs as a potential problematic dependency is essential in relational design.

We now present the definition of **fourth normal form (4NF)**, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations.

**Definition.** A relation schema *R* is in 4NF with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in $F$ '*X* is a superkey for *R*.

We can state the following points:

- An all-key relation is always in BCNF since it has no FDs.
- An all-key relation such as the EMP relation in Figure 4.14(a), which has no FDs but has the MVD Ename →→ Pname | Dname, is not in 4NF.
- A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF.
- The decomposition removes the redundancy caused by the MVD.

The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD. Consider the EMP relation in Figure 4.14(a).

EMP is not in 4NF because in the nontrivial MVDs Ename →→ Pname and Ename →→ Dname, and Ename is not a superkey of EMP. We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS, shown in Figure 4.14(b). Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs Ename →→ Pname in EMP_PROJECTS and Ename →→ Dname in EMP_DEPENDENTS are trivial MVDs. No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDENTS. No FDs hold in these relation schemas either.

### 4.5.3 Join Dependencies and Fifth Normal Form

**Definition.** A **join dependency (JD)**, denoted by JD(*R*1, *R*2, ..., *Rn*), specified on relation schema *R*, specifies a constraint on the states *r* of *R*. The constraint states that every legal state *r* of *R* should have a nonadditive join decomposition into *R*1, *R*2, ..., *Rn*. Hence, for every such *r* we have

$$* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$$

Notice that an MVD is a special case of a JD where $n = 2$. That is, a JD denoted as JD (*R*1, *R*2) implies an MVD $(R1 \cap R2) \twoheadrightarrow (R1 - R2)$ (or, by symmetry, $(R1 \cap R2) \twoheadrightarrow (R2 - R1)$). A join dependency JD(*R*1, *R*2, ..., *Rn*), specified on relation schema *R*, is a **trivial** JD if one of the relation schemas *Ri* in JD(*R*1, *R*2, ..., *Rn*) is equal to *R*. Such a dependency is called trivial because it has the nonadditive join property for any relation state *r* of *R* and thus does not specify any constraint on *R*.We can now define fifth normal form, which is also called *project-join normal form*.

**Definition.** A relation schema *R* is in **fifth normal form (5NF)** (or **project-join normal form (PJNF)**) with respect to a set *F* of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(*R*1, *R*2, ..., *Rn*) in *F*+ (that is, implied by *F*), every *Ri* is a superkey of *R*.

For an example of a JD, consider once again the SUPPLY all-key relation in Figure 4.14(c). Suppose that the following additional constraint always holds: Whenever a supplier *s* supplies part *p*, *and* a project *j* uses part *p*, *and* the supplier *s* supplies *atleast one* part to project *j*, *then* supplier *s* will also be supplying part *p* to project *j*.

This constraint can be restated in other ways and specifies a join dependency JD(*R*1, *R*2, *R*3) among the three projections *R*1(Sname, Part_name), *R*2(Sname, Proj_name), and *R*3(Part_name, Proj_name) of SUPPLY. If this constraint holds, the tuples below the dashed line in Figure 4.14(c) must exist in any legal state of the SUPPLY relation that also contains the tuples above the dashed line. Figure 4.14(d) shows how the SUPPLY relation *with the join dependency* is decomposed into three relations *R*1, *R*2, and *R*3 that are each in 5NF. Notice that applying a natural join to *any two* of these relations *produces spurious tuples*, but applying a natural join to *all three together* does not. The reader should verify this on the sample relation in Figure 4.14(c) and its projections in Figure 4.14(d). This is because only the JD exists, but no MVDs are specified. Notice, too, that the JD(*R*1, *R*2, *R*3) is specified on *all* legal relation states, not just on the one shown in Figure 4.14(c).

Discovering JDs in practical databases with hundreds of attributes is next to impossible. It can be done only with a great degree of intuition about the data on the part of the designer. Therefore, the current practice of database design pays scant attention to them.