6.2.4 Normalization Forms

Normalization of a relational model is a process of analyzing the given relations based on their functional dependencies and candidate keys to minimize redundancy and insertion, deletion, and update anomalies. The normalization procedure entails various *normal form tests* which are typically sequentially evaluated. Unsatisfactory relations that do not meet the normal form tests are decomposed into smaller relations.

6.2.4.1 First Normal Form (1 NF)

The **first normal form (1 NF)** states that every attribute type of a relation must be atomic and single-valued. Hence, no composite or multi-valued attribute types are tolerated. This is the same as the domain constraint we introduced earlier.

Consider the following example:

SUPPLIER(SUPNR, NAME(FIRST NAME, LAST NAME), SUPSTATUS)

This relation is not in 1 NF as it contains a composite attribute type NAME that consists of the attribute types FIRST NAME and LAST NAME. We can bring it in 1 NF as follows:

SUPPLIER(SUPNR, FIRST NAME, LAST NAME, SUPSTATUS)

In other words, composite attribute types need to be decomposed in their parts to bring the relation in 1 NF.

Suppose we have a relation DEPARTMENT. It has a department number, a department location and a foreign key referring to the social security number of

the employee who manages the department:

DEPARTMENT(DNUMBER, DLOCATION, *DMGRSSN*)

Assume now that a department can have multiple locations and that multiple departments are possible at a given location. The relation is not in 1 NF since DLOCATION is a multi-valued attribute type. We can bring it in 1 NF by removing DLOCATION from department and putting it into a new relation DEP-LOCATION together with DNUMBER as a foreign key:

DEPARTMENT(DNUMBER, *DMGRSSN*)
DEP-LOCATION(*DNUMBER*, DLOCATION)

The primary key of this new relation is then the combination of both, since a department can have multiple locations and multiple departments can share a location. Figure 6.10 illustrates some example tuples. You can see that department number 15 has two locations: New York and San Francisco. Department number 30 also has two locations: Chicago and Boston. The two lower tables bring it in the 1 NF since every attribute type of both relations is now atomic and single-valued. To summarize, multi-valued attribute types (e.g., DLOCATION) should be removed and put in a separate relation (e.g., DEP-LOCATION) along with the primary key (e.g., DNUMBER) of the original relation (e.g., DEPARTMENT) as a foreign key. The primary key of the new relation is then the combination of the multi-valued attribute type and the primary key of the original relation (e.g., DNUMBER and DLOCATION).

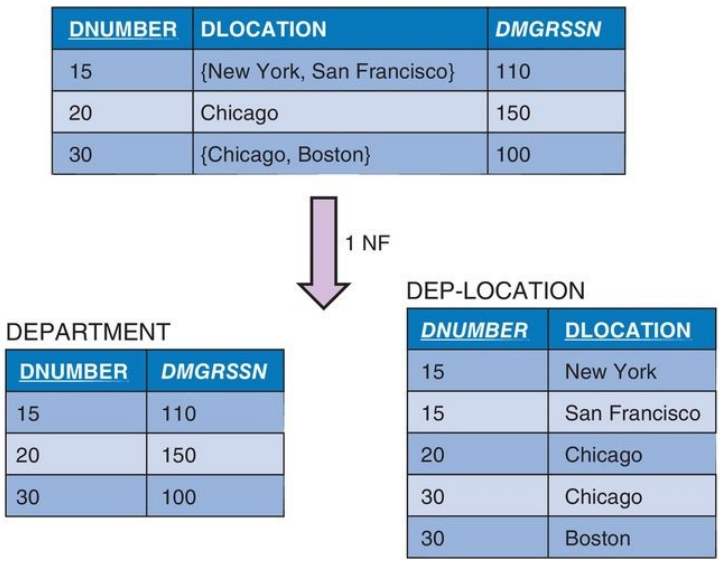

Figure 6.10 First normal form: the unnormalized relation (above) is decomposed into two relations (below) by ensuring there are no composite or multi-valued attribute types.

Let's give another example. Say we have a relation R1 with employee information such as SSN, ENAME, DNUMBER, DNAME, and PROJECT, which is a composite attribute type consisting of PNUMBER, PNAME and HOURS:

R1(SSN, ENAME, DNUMBER, DNAME, PROJECT(PNUMBER, PNAME, HOURS))

We assume an employee can work on multiple projects and multiple employees can work on the same project. Hence, we have a multi-valued composite attribute type PROJECT in our relation R1. In other words, both conditions of the first normal form are clearly violated. To bring it in first normal form, we create two relations R11 and R12 where the latter includes the project attribute types together with SSN as a foreign key:

R11(SSN, ENAME, DNUMBER, DNAME)

R12(*SSN*, PNUMBER, PNAME, HOURS)

The primary key of R12 is then the combination of SSN and PNUMBER, since an employee can work on multiple projects and multiple employees can work on a project.

6.2.4.2 Second Normal Form (2 NF)

Before we can start discussing the **second normal form (2 NF)**, we need to introduce the concepts of full and partial functional dependency. A functional dependency X → Y is a **full functional dependency** if removal of any attribute type A from X means that the dependency does not hold anymore. For example, HOURS is fully functionally dependent upon both SSN and PNUMBER:

SSN, PNUMBER → HOURS

More specifically, to know the number of hours an employee worked on a project, we need to know both the SSN of the employee and the project number. Likewise, project name is fully functionally dependent upon project number:

PNUMBER → PNAME

A functional dependency X → Y is a partial dependency if an attribute type A from X can be removed from X and the dependency still holds. As an example, PNAME is partially functionally dependent upon SSN and PNUMBER:

SSN, PNUMBER → PNAME

It only depends upon PNUMBER, not on SSN.

A relation R is in the 2 NF if it satisfies 1 NF and every non-prime attribute type A in R is fully functionally dependent on any key of R. In case the relation is not in second normal form, we must decompose it and set up a new relation for each partial key together with its dependent attribute types. Also, it is important to keep a relation with the original primary key and any attribute types that are fully functionally dependent on it. Let's illustrate this with an example.

Say we have a relation R1 with attribute types SSN, PNUMBER, PNAME, HOURS:

R1(SSN, PNUMBER, PNAME, HOURS)

It contains both project information and information about which employee worked on what project for how many hours. The assumptions are as follows: an employee can work on multiple projects; multiple employees can work on the same project; and a project has a unique name. The relation R1 is in 1 NF since there are no multi-valued or composite attribute types. However, it is not in 2 NF. The primary key of the relation R1 is a combination of SSN and PNUMBER. The attribute type PNAME is not fully functionally dependent on the primary key, it only depends on PNUMBER. HOURS, however, is fully functionally dependent upon both SSN and PNUMBER. Hence, we need to remove the attribute type PNAME and put it in a new relation R12, together with PNUMBER:

R11(SSN, *PNUMBER*, HOURS)
R12(PNUMBER, PNAME)

The relation R11 can then be called WORKS-ON(SSN, PNUMBER, HOURS) and the relation R12 can be referred to as PROJECT(PNUMBER, PNAME).

Figure 6.11 illustrates how to bring a relation into 2 NF with some example tuples. Note the redundancy in the original relation. The name “Hadoop” is repeated multiple times, which is not desirable from a storage perspective. Also, if we would like to update it (e.g., from “Hadoop” to “Big Data”), then multiple changes need to take place. This is not the case for the two normalized relations at the bottom, where the update should only be done once in the PROJECT relation.

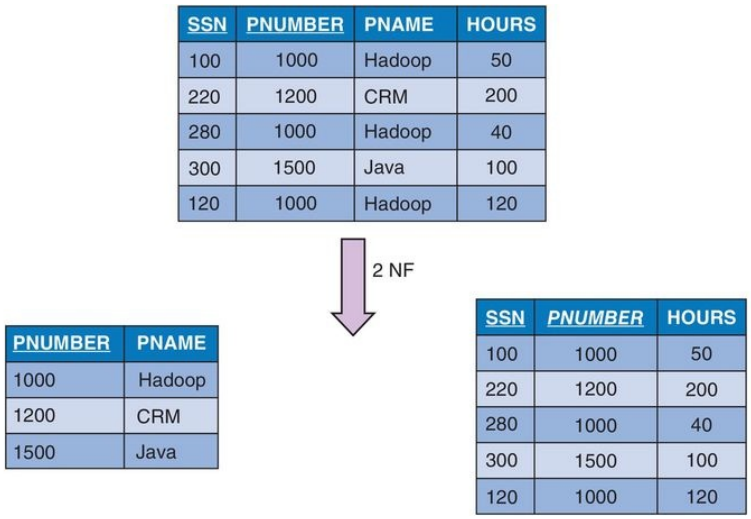

Figure 6.11 Second normal form: the unnormalized relation (above) is decomposed into two relations (below) by ensuring every non-prime attribute type is fully functionally dependent on the primary key.

6.2.4.3 Third Normal Form (3 NF)

To discuss the **third normal form (3 NF)**, we need to introduce the concept of **transitive dependency**. A functional dependency X → Y in a relation R is a transitive dependency if there is a set of attribute types Z that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold. A relation is in the 3 NF if it satisfies 2 NF and no non-prime attribute type of R is transitively dependent on the primary key. If this is not the case, we need to

decompose the relation R and set up a relation that includes the non-key attribute types that functionally determine the other non-key attribute types. Let's work out an example to illustrate this.

The relation R1 contains information about employees and departments as follows:

R1(SSN, ENAME, DNUMBER, DNAME, *DMGRSSN*)

The SSN attribute type is the primary key of the relation. The assumptions are as follows: an employee works in one department; a department can have multiple employees; and a department has one manager. Given these assumptions, we have two transitive dependencies in R. DNAME is transitively dependent on SSN via DNUMBER. In other words, DNUMBER is functionally dependent on SSN, and DNAME is functionally dependent on DNUMBER. Likewise, DMGRSSN is transitively dependent on SSN via DNUMBER. In other words, DNUMBER is functionally dependent on SSN and DMGRSSN is functionally dependent on DNUMBER. DNUMBER is not a candidate key nor a subset of a key. Hence, the relation is not in 3 NF. To bring it in 3 NF, we remove the attribute types DNAME and DMGRSSN and put them in a new relation R12 together with DNUMBER as its primary key:

R11(SSN, ENAME, *DNUMBER*)
R12(DNUMBER, DNAME, *DMGRSSN*)

The relation R11 can be called EMPLOYEE(SSN, ENAME, DNUMBER) and the relation R12 can be referred to as DEPARTMENT(DNUMBER, DNAME, DMGRSSN).

Figure 6.12 shows some example tuples for both the unnormalized and normalized relations. Note the redundancy in the unnormalized case, where the

values “marketing” for DNAME and “210” for DMGRSSN are repeated multiple times. This is not the case for the normalized relations where these values are only stored once.

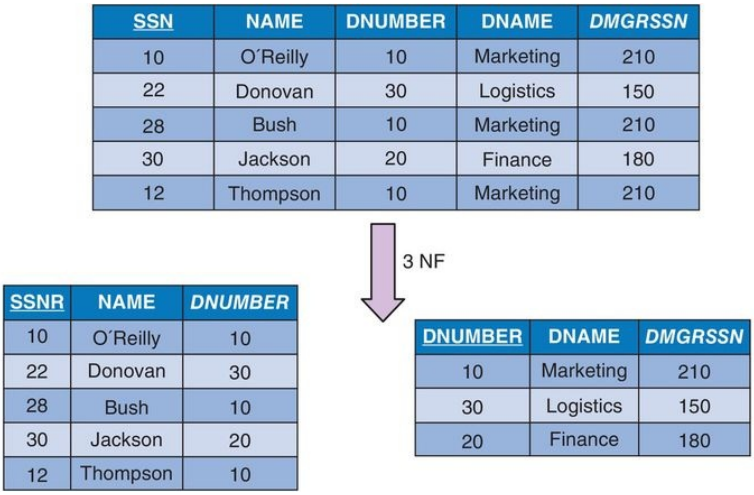

Figure 6.12 Third normal form: the unnormalized relation (above) is decomposed into two relations (below) by ensuring no non-prime attribute types are transitively dependent on the primary key.

6.2.4.4 Boyce–Codd Normal Form (BCNF)

We can now discuss the **Boyce–Codd normal form (BCNF)**, also referred to as the 3.5 normal form (3.5 NF). Let’s first introduce another concept. A functional dependency X → Y is called a **trivial functional dependency** if Y is a subset of X. An example of a trivial functional dependency is between SSN and NAME, and SSN:

SSN, NAME → SSN

A relation R is in the BCNF provided that for *each* of its non-trivial functional dependencies X → Y, X is a superkey – that is, X is either a candidate key or a superset thereof. It can be shown that the BCNF is stricter than the