Chapter 5: Database Constraints and Relational Database Desing

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 *A*1, *A*2, ...,*An*; let us think of the whole database as being described by a single **universal** relation schema *R* = {*A*1, *A*2, ... , *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.

Normalization:

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.

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 below, whose primary key is Dnumber. We assume that each department can have *a number of* locations. As we can see, this is not in 1NF because Dlocations is not an atomic attribute.

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

<u>To Convert it into 1NF:</u>

There are two 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 below. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.

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

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 15.9(c). In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing *redundancy* in the relation.

DEPARTMENT

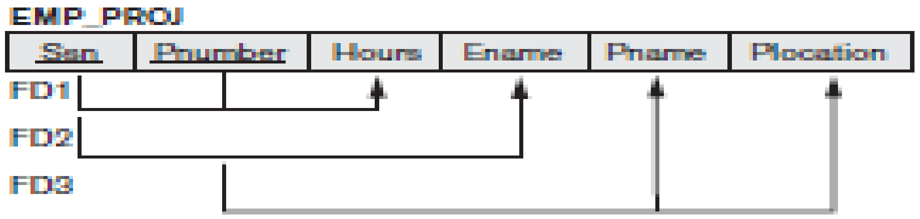
Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

<u>Second Normal Form:</u>

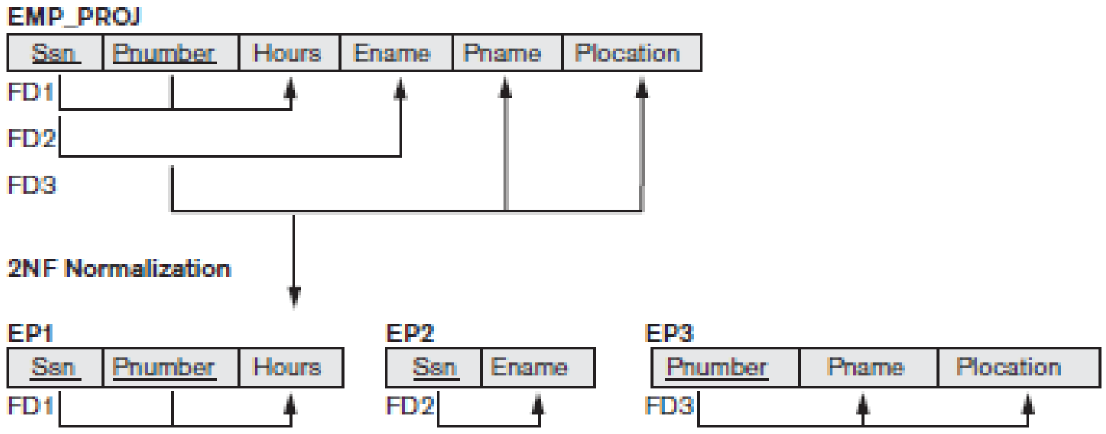
Second normal form (2NF) is based on the concept of *full functional dependency.* A functional dependency *X* → *Y* is a **full functional dependency** if removal of any attribute *A* from *X* means that the dependency does not hold any more; that is, for any attribute *A* ε *X*, (*X* – {*A*}) does *not* functionally determine *Y*. A functional dependency *X*→*Y* is a **partial dependency** if some attribute *A* ε *X* can be removed from *X* and the dependency still holds; that is, for some *A* ε *X*, (*X* – {*A*}) → *Y*. In Figure EMP_PROJ, {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 below 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 of above Figure lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure below each of which is in 2NF.

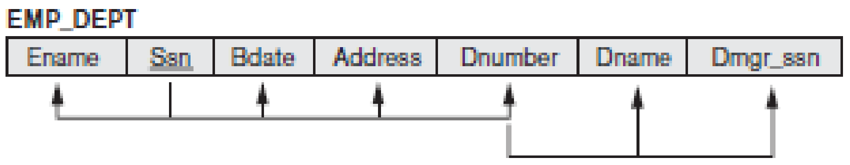


Third Normal Form:

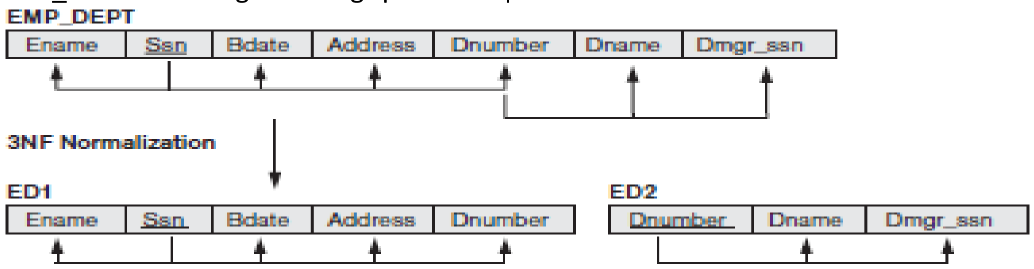
Third normal form (3NF) is based on the concept of *transitive dependency*. A functional dependency *X*→*Y* in a relation schema *R* is a **transitive dependency** if there exists a set of attributes *Z* in *R* that is neither a candidate key nor a subset of any key of *R*, and both *X*→*Z* and *Z*→*Y* hold. The dependency Ssn→Dmgr_ssn is transitive through Dnumber in EMP_DEPT in Figure below, 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 below 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 below. 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.



Summary of Normal Forms Based on Primary Key and Corresponding Normalization

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).

Boyce-Codd Normal Form (BCNF):

- A relation is in BCNF if every determinant is a candidate key
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.
- **Consider the following example:**
    - **Funds consist of one or more Investment Types.**
    - **Funds are managed by one or more Managers**
    - **Investment Types can have one more Managers**
    - **Managers only manage one type of investment.**
- **Relation: FUNDS (FundID, InvestmentType, Manager)**

Fund ID	**Investment Type**	**Manager**
99	**Common Stock**	**Smith**
33	**Common Stock**	**Green**
22	**Growth Stocks**	**Brown**
11	**Municipal Bonds**	**Smith**

- **FD1: FundID, InvestmentType → Manager**
- **FD2: FundID, Manager → InvestmentType**
- **FD3: Manager → InvestmentType**

- **In this case, the combination FundID and InvestmentType form a *candidate key* because we can use FundID,InvestmentType to uniquely identify a tuple in the relation.**
- **Similarly, the combination FundID and Manager also form a *candidate key* because we can use FundID, Manager to uniquely identify a tuple.**
- **Manager by itself is not a candidate key because we cannot use Manager alone to uniquely identify a tuple in the relation.**
- **Is this relation FUNDS(FundID, InvestmentType, Manager) in 1NF, 2NF or 3NF ?**

Given we pick FundID, InvestmentType as the *Primary Key:* 1NF for sure. 2NF because all of the non-key attributes (Manager) is dependant on all of the key. 3NF because there are no transitive dependencies.

- **However consider what happens if we delete the tuple with FundID 22. We loose the fact that Brown manages the InvestmentType "Growth Stocks."**
- **Therefore, while FUNDS relation is in 1NF, 2NF and 3NF, it is in BCNF because not all determinants (Manager in FD3) are candidate keys.**
- **The following are steps to normalize a relation into BCNF:**
    1. **List all of the determinants.**
    2. **See if each determinant can act as a key (candidate keys).**
    3. **For any determinant that is *not* a candidate key, create a new relation from the functional dependency. Retain the determinant in the original relation.**
- **For our example:**

FUNDS (FundID, InvestmentType, Manager)

1. The determinants are:

FundID, InvestmentType
FundID, Manager
Manager

2. Which determinants can act as keys ?

FundID, InvestmentType *YES*
FundID, Manager *YES*
Manager *NO*

3. Create a new relation from the functional dependency:

MANAGERS(<u>Manager</u>, InvestmentType)
FUND_MANAGERS(<u>FundID</u>, <u>Manager</u>)

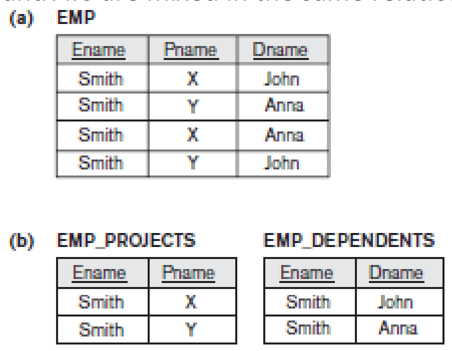
In this last step, we have retained the determinant "Manager" in the original relation MANAGERS.

- **Each of the new relations sould be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF**

Multivalued Dependency and Fourth Normal Form:

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 1(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 spec fied 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.


(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

Figure 1:Multivalued dependency and fourth normal form.(a) The Emp relation with two MVDs:Ename→→PNAME and ENAME→→DNAME. (b) Decomposing EMP into two relations in 4NF

__Formal Definition of Multivalued Dependency:__

Definition. A multivalued dependency *X*→→*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* ∪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*→→*Y* holds, we say that *X* **multidetermines** *Y*. Because of the symmetry in the definition, whenever *X* →→ *Y* holds in *R*, so does *X* →→ *Z*. Hence, *X* →→ *Y* implies *X*→→*Z*, and therefore it is sometimes written as *X*→→*Y*|*Z*.

An MVD X →→ Y in R is called a **trivial MVD** if (a) *Y* is a subset of *X*, or (b) *X* ∪Y = *R*. For example, the relation EMP_PROJECTS in Figure 1(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 1(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).

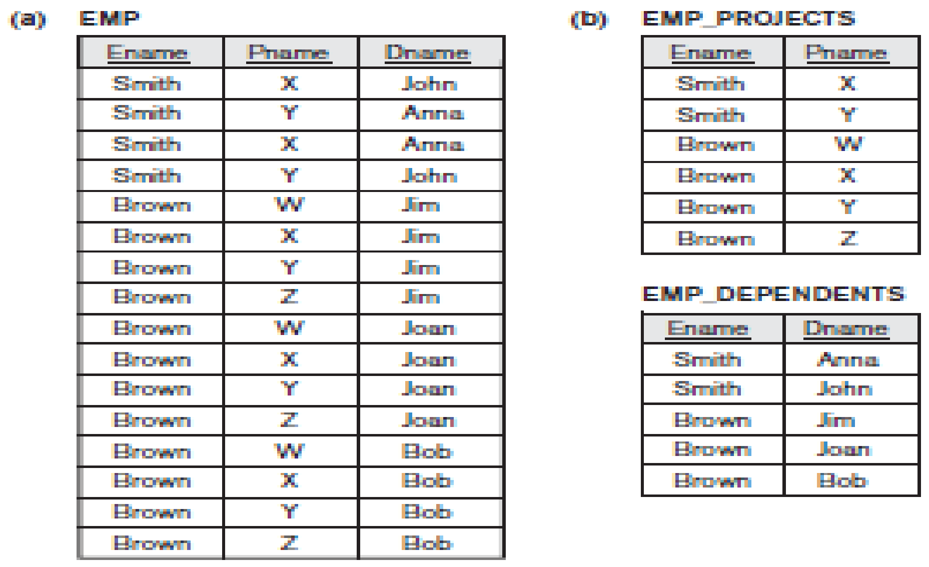
__Fourth Normal Form:__

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

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 1(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 1(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.

To illustrate the importance of 4NF, Figure 16.4(a) shows the EMP relation in Figure 1(a) with an additional employee, 'Brown', who has three dependents ('Jim', 'Joan', and 'Bob') and works on four different projects (‘W’, ‘X’, ‘Y’, and ‘Z’). There are 16 tuples in EMP in Figure 16.4(a). If we decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS, as shown in Figure 16.4(b), we need to store a total of only 11 tuples in both relations. Not only would the decomposition save on storage, but the update anomalies associated with multivalued dependencies would also be avoided. For example, if ‘Brown’ starts working on a new additional project ‘P,’we must insert *three* tuples in EMP—one for each dependent. If we forget to insert any one of those, the relation violates the MVD and becomes inconsistent in that it incorrectly implies a relationship between project and dependent. If the relation has nontrivial MVDs, then insert, delete, and update operations on single tuples may cause additional tuples to be modified besides the one in question. If the update is handled incorrectly, the meaning of the relation may change. However, after normalization into 4NF, these update anomalies disappear. For example, to add the information that ‘Brown’ will be assigned to project ‘P’, only a single tuple need be inserted in the 4NF relation EMP_PROJECTS.

Figure 16.4
Decomposing a relation state of EMP that is not in 4NF. (a) EMP relation with additional tuples. (b) Two corresponding 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.



(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John
Brown	W	Jim
Brown	X	Jim
Brown	Y	Jim
Brown	Z	Jim
Brown	W	Joan
Brown	X	Joan
Brown	Y	Joan
Brown	Z	Joan
Brown	W	Bob
Brown	X	Bob
Brown	Y	Bob
Brown	Z	Bob

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

EMP_DEPENDENTS

Ename	Dname
Smith	Anna
Smith	John
Brown	Jim
Brown	Joan
Brown	Bob