Normalization

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. There are two levels at which we can discuss the "goodness" of relation schemas:

a) Logical (or conceptual) level:

- It discusses how users interpret the relation schemas and the meaning of their attributes.
- Having good relation schemas at this level enables users to understand clearly the meaning of the data in the relations, and hence to formulate their queries correctly.
- At this level we are interested in schemas of both base relations and views (virtual relations).

b) Implementation (or storage) level:

- It discusses how the tuples in a base relation are stored and updated.
- This level applies only to schemas of base relations-which will be physically stored as files.

Database design may be performed using two approaches:

- **Bottom-up design methodology (also called design by synthesis) :**
  - This approach considers the basic relationships among individual attributes as the starting point and uses those to construct relation schemas.
  - This approach is not very popular in practice because it suffers from the problem of having to collect a large number of binary relationships among attributes as the starting point.
- **Top-down design methodology (also called design by analysis)** :
  - This approach starts with a number of groupings of attributes into relations that exist together naturally.
  - The relations are then analyzed individually and collectively, leading to further decomposition until all desirable properties are met.

4.1 INFORMAL DESIGN GUIDELINES FOR RELATION SCHEMAS:

There are four *informal measures* of quality for relation schema design:

i) Semantics of the attributes

ii) Reducing the redundant values in tuples

iii) Reducing the null values in tuples

iv) Disallowing the possibility of generating spurious tuples

4.1.1 Semantics of the Relation Attributes:

- 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.
- This meaning, or semantics, specifies how to interpret the attribute values stored in a tuple of the relation-in other words, how the attribute values in a tuple relate to one another.
- To illustrate this, consider **Figure 4.1**, a simplified version of the COMPANY relational database schema and **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 (ENAMEl. 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 DMGRSSN 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.
- 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).

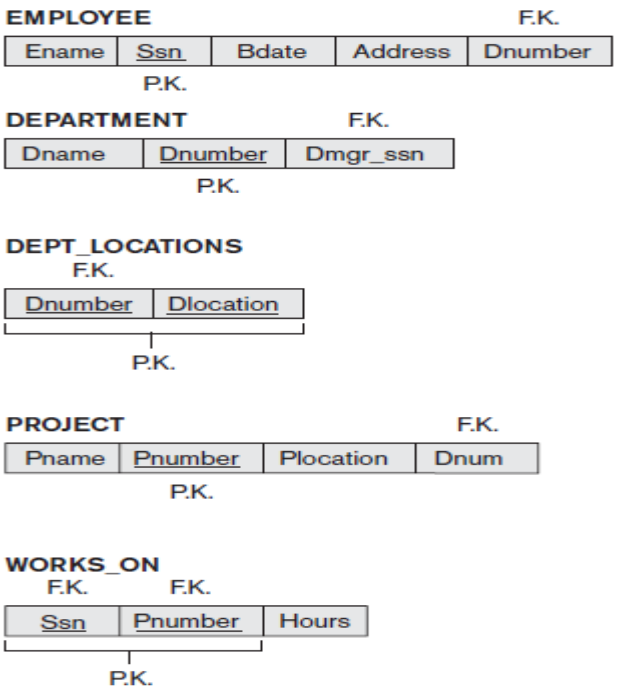


FIGURE 4.1: A simplified COMPANY relational database schema.

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	291Berry, 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

Example database state for the relational database schema of Figure 6.1.

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.

Examples of violating Guideline 1: The relation schemas in **Figures 4.3a** and **4.3b** also have clear semantics. A tuple in the **EMP_DEPT** relation schema of Figure 4.3a 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 (DMGRSSN) of the department manager. For the **EMP_PROJ**

relation of Figure 4.3b, each tuple relates an employee to a project but also includes the employee name (ENAME), project name (PNAME), and project location (PLOCATION).

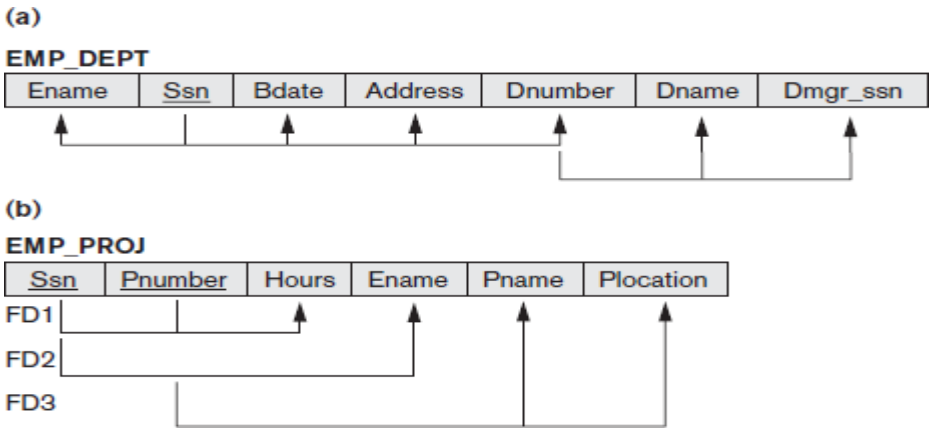


FIGURE 4.3 Two relation schemas suffering from update anomalies.

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, and DMGRSSN) are ***repeated for every employee who works for that department***.

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 6.4 Example states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 4.2.

- ***The various update anomalies in non normalized database (DB) can be classified into***
  - ***a) Insertion anomalies***
  - ***b) Deletion anomalies***
  - ***c) Modification anomalies.***

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

1st type of Insertion Anomaly:

- To insert a new employee tuple into EMP_DEPT, we must include either all 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 the attribute values of department 5 correctly so that they are *consistent* with 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.

2nd type of Insertion Anomaly:

- 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 causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity-not a department entity.
- 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.

b) **Deletion Anomalies**: The problem of deletion anomalies is related to the second insertion anomaly situation discussed earlier.

- 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.

c) **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.

Based on the preceding three anomalies, we can state the guideline that 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.

4.1.3 Null Values in Tuples:

- 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.

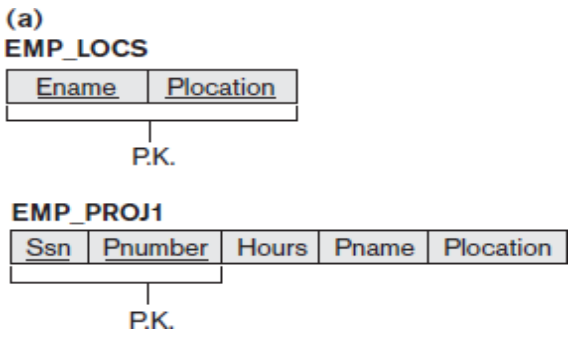
- Moreover, nulls can have multiple interpretations, such as the following:
  - **a)** The attribute *does not apply* to this tuple.
  - ***b)*** The attribute value for this tuple is *unknown.*
  - **c)** The value is *known but absent;* that is, it has not been recorded yet.

We may have another guideline to deal with NULL values as follows:

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.

4.1.4 Generation of Spurious Tuples:

- Consider the two relation schemas **EMP_LOCS** and **EMP_PROJ1** in **Figure 4.5a**, which are decomposed from **EMP_PROJ** relation of **Figure 4.3b**.
- **Figure 4.5b** shows relation states of EMP_LOCS and EMP_PROJ1 corresponding to the EMP_PROJ relation of **Figure 4.4**.



(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 of Figure 4.3b. (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_PROJI.

- 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 dotted lines in Figure 4.5b is shown.

	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 dotted lines in EMP_PROJ1 and

EMP_LOCS of Figure 6.5. Generated spurious tuples are marked by asterisks.

- Additional tuples in the **Figure 4.6** that were not in **EMP_PROJ (Figure 4.4)** are called **spurious tuples** because they represent spurious or *wrong* information that is not valid.

We can now informally state another design guideline as follows:

GUIDELINE 4 : Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

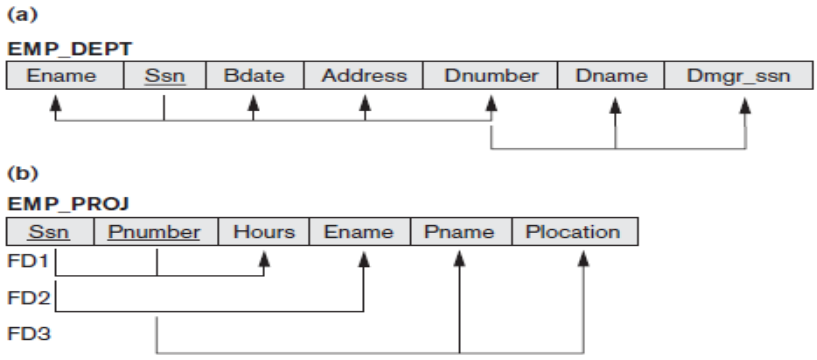
4.2 FUNCTIONAL DEPENDENCIES:

4.2.1 Definition of Functional Dependency:

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*.

  Ex: VehicleId→State (By knowing the vehicle id it is possible to determine the state that vehicle belongs to).
- 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.
- Consider the relation schema **EMP_PROJ** and **EMP_DEPT** given below. From the semantics of the attributes, 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**).

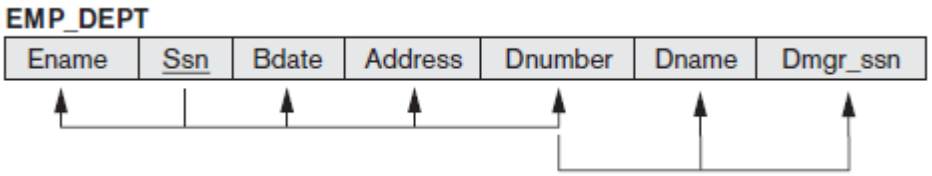
- **Figure 6.3** introduces a diagrammatic notation for displaying FDs:
  - Each FD is displayed as a horizontal line.
  - The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD.
  - The right-hand-side attributes are connected by arrows pointing toward the attributes.

4.2.2 Inference Rules for Functional Dependencies:

- We denote by **F** the ***set of functional dependencies*** that are specified on ***relation schema*** **R**.
- Usually, however, numerous other functional dependencies hold that satisfy the dependencies in F. ***Those other dependencies can be inferred or deduced from the FDs in F***.
- Therefore, formally it is useful to define a concept called ***closure that includes all possible dependencies that can be inferred from the given set F***.

Definition of closure : Formally, the set of all dependencies that include F as well as all dependencies that can be inferred from F is called the **closure of F**; it is denoted by F+.

- For example, suppose that we specify the following set F of obvious functional dependencies on the relation schema of EMP_DEPT:



F= {SSN → {ENAME, BDATE, ADDRESS, DNUMBER},
DNUMBER → {DNAME, DMGRSSN}}

Some of the additional functional dependencies that we can infer from F are the following:

SSN → {DNAME, DMGRSSN}
SSN → SSN
DNUMBER → DNAME

- A set of inference rules can be used to infer new dependencies from a given set of dependencies.
- We use the notation **F |= X → Y** to denote that the functional dependency X → *Y* is **inferred** from the set of functional dependencies **F**.

4.2.2.1 INFERENCE RULES for functional dependencies:

a) IRI (reflexive rule):

- The reflexive rule (IR1) states that ***a set of attributes always determines itself or any of its subsets***, which is obvious.
- Because IR l generates dependencies that are always true, such dependencies are called ***trivial***.
- Formally, a functional dependency X → Y is *trivial* if ***Y is a subset of X (X⊃ Y)***; otherwise, it is nontrivial.

If X ⊃ Y, then X → Y

Ex:- X={ SSN,FNAME,LNAME} , Y={ FNAME,LNAME}
Therefore {SSN, FNAME, LNAME} → {FNAME, LNAME}

b) IR2 (augmentation rule):

- The augmentation rule (IR2) says that adding the same set of attributes to both the left- and right-hand sides of a dependency results in another valid dependency.

{X → Y} |= XZ → YZ

Ex:- X= {SSN} , Y={ FNAME}
Therefore {SSN, LNAME} → {FNAME, LNAME}

- **IR3 (transitive rule):**
  - According to IR3, functional dependencies are ***transitive***.

{X → Y, Y → Z}|= X → Z

Ex:- X= {SSN}, Y={DNUMBER}, Z={DNAME}
Therefore {SSN} → {DNAME}

- **IR4 (decomposition, or projective, rule):**
  - The decomposition rule (IR4) says that we can remove attributes from the right-hand side of a dependency;
  - Applying this rule repeatedly can decompose the FD **X → {A1, A2, ... , An}** into the set of dependencies **{X → A1, X → A2 , ... , X → An}.**

{X → YZ} |= X → Y

Ex:- X= {SSN}, Y= {FNAME}, Z={LNAME}
Therefore {SSN} → {FNAME}

- **IR5 (union, or additive, rule):**
  - The union rule (IRS) allows us to do the opposite; we can combine a set of dependencies **{X A1), X→A2, ... , X→An}** into the single FD **X→{A1, A2,.....,An}**

X → Y, X → Z}|= X → YZ

Ex:- X={SSN}, Y={FNAME}, Z={LNAME}
Therefore {SSN} → {FNAME, LNAME}

- **IR6 (pseudotransitive rule):**

{X → Y, WY→ Z}|= WX → Z

Ex:- X={SSN}, W={DNAME}, Y={DNAME}, Z={MGRSSN}
Therefore {DNAME, SSN} → {MGRSSN}

4.2.2.2 PROOF OF INFERENCE RULES:

- Each of the inference rules can be proved from the definition of functional dependency, either by ***direct proof or by contradiction***.
- A proof by contradiction assumes that the rule does not hold and shows that this is not possible.

PROOF OF IR1:

Suppose that **X Y** and that two tuples **t1** and **t2** exist in some relation instance 'r' of 'R' such that t1 [X] = t2 [X]. Then t1[Y] = t2[Y] because X Y; hence, X → Y must hold in 'r'.

PROOF OF IR2 (BY CONTRADICTION):

Assume that X → Y holds in a relation instance 'r' of 'R' but that XZ → YZ does not hold. Then there must exist two tuples 't1' and 't2' in 'r' such that:
1. **t1[X] = t2[X]**
2. **t1[Y] = t2[Y**]
3. **t1[XZ] = t2 [XZ]**
4. **t1[Y] ≠' t2[YZ]**.

This is not possible because from **(1)** and **(3)** we deduce (5) **t1[Z] = t2 [Z]**, and from **(2)** and **(5)** we deduce **(6) t1 [YZ] = t2 [Y]**, contradicting **(4)**.

PROOF OF IR3:

Assume that (1) X → Yand (2) Y → Z both hold in a relation 'r'. Then for any two tuples 't1' and 't2' in 'r' such that t1[X] = t2 [X]. We must have (3) t1[Y] = t2[Y], from assumption (1); hence we must also have (4) t1 [Z] = t2[Z], from (3) and assumption (2); hence X Z must hold in 'r'.

PROOF OF IR4 (USING IRl THROUGH IR3):

1. X → YZ (given).
2. YZ →Y (using IRI and knowing that YZ Y).
3. X →Y (using IR3 on 1 and 2).

PROOF OF IR5 (USING IRl THROUGH IR3):

1. X → Y (given).
2. X → Z (given).

3. X → XY (using IR2on 1 by augmenting with X; notice that XX = X).
4. XY → YZ (using IR2on 2 by augmenting with *Y).*
5. X → YZ (using lR3 on 3 and 4).

PROOF OF IR6 (USING IRl THROUGH IR3):

1. X → *Y* (given).
2. WY → Z (given).
3. WX → WY (using IR2 on 1 by augmenting with *W).*
4. WX → Z (using IR3 on 3 and 2).

- ***The set of dependencies F+, which we called the closure of F, can be determined from F by using only inference rules IRI through IR3.***
- Inference rules IR1 through IR3 are known as ***Armstrong's inference rules***.
- A systematic way to determine these additional functional dependencies is:

  First determine each set of attributes 'X' that appears as a left-hand side of some functional dependency in F and then to determine the set of all attributes that are dependent on X.

  Definition: For each attributes 'X' that appears as a left-hand side of some functional dependency in 'F', we determine the set 'X+' of attributes that are functionally determined by 'X' based on 'F'; here 'X+' is called the **closure of X under F**. Algorithm 4.1 can be used to calculate 'X+'.

  Algorithm 4.1: Determining X+, the Closure of X under F

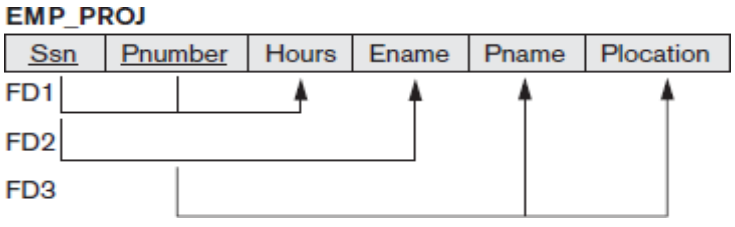
```
X+ := X;
Repeat
        oldX+ := X+;
        for each functional dependency Y → Z in F do
        if X+ Y then X+ := X+ Ụ Z;
until (X+ = oldX+),
```

  a) Algorithm 4.1 starts by setting X+ to all the attributes in X. By IR1, we know that all these attributes are functionally dependent on X.

  b) Using inference rules IR3 and IR4, we add attributes to X+, using each functional dependency in F.

  c) We keep going through all the dependencies in F (the *repeat* loop) until no more attributes are added to X+ *during a complete cycle* (of the *for* loop) through the dependencies in F.

  For example, consider the relation schema EMP_PROJ. From the semantics of the attributes, we specify the following set F of functional dependencies that should hold on EMP_PROJ;



a. SSN → ENAME
b. PNUMBER → {PNAME, PLOCATION}
c. {SSN, PNUMBER} → HOURS

Using Algorithm 4.1, we calculate the following closure sets with respect to F;

{SSN}+ = {SSN, ENAME}
{PNUMBER}+ = {PNUMBER, PNAME, PLOCATION}
{SSN, PNUMBER}+ = {SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS}

4.2.3 Equivalence of Sets of Functional Dependencies:

Definition: A set of functional dependencies F is said to **cover** another set of functional dependencies E if every FD in E is also in F+; that is, if every dependency in E can be inferred from F; alternatively, we can say that **E is covered by F**.

Definition: Two sets of functional dependencies E and F are **equivalent** if E+ = F+. Hence, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions **E covers F *and* F covers E** hold.

4.2.4 Minimal Sets of Functional Dependencies:

- A **minimal cover** of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F+ of F.
- This property is lost if any dependency from the set F is removed; F must have no redundancies in it, and the dependencies in E are in a standard form.
- To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:
  - **a)** Every dependency in F has a single attribute for its right-hand side.
  - **b)** We cannot replace any dependency X → A in F with a dependency Y → A, where *Y* is a proper subset of X, and still have a set of dependencies that is equivalent to F.
  - **c)** We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.
- A minimal cover of a set of functional dependencies E is a minimal set of dependencies F that is equivalent to E. There can be several minimal covers for a set of functional dependencies.
- We can always find *at least one* minimal cover F for any set of dependencies E using **Algorithm 4.2**.

Algorithm 4.2: Finding a Minimal Cover F for a Set of Functional Dependencies E

1. Set F := E.
2. Replace each functional dependency X → {A1, A2, ..., An} in F by the n functional dependencies X → A1, X → A2, ..., X → An.

3. For each functional dependency X → A in F
   for each attribute B that is an element of X
   if {{F - {X → A} } U {(X - {B}) → A} } is equivalent to F,
   then replace X → A with (X - {B}) → A in F.

4. For each remaining functional dependency X → A in F
   if { F - {X → A} } is equivalent to F,
   then remove X → A from F.

Example: Let the given set of FD's be E:{B→ A, D→ A,AB→ D}. Find the minimal cover of E.

Answer:

- All the above dependencies are in canonical form, so we have completed step 1 of the algorithm and can proceed to step2. In step2 we need to determine if AB→D has any redundant attributes on the left hand side; that is can it be replaced by B→D or A→D?
- Since B→A, by augmenting with B on both sides (IR2), we get BB→AB, or B→AB (i). However AB→D as given (ii).
- Hence by the transitive rule (IR3), we get from (i) and (ii), B→D. Hence AB→D may be replaced by B→D.
- Now we have a set E'= {B→ A, D→ A, B→ D}. No further reduction is possible in step 2 since all FDs have a single attribute on the left hand side.
- In step 3 we look for a redundant FD in E'. By using the transitive rule on B→ D and D→ A, we derive B→ A. hence B→ A is redundant in E' and can be eliminated.
- Hence the minimum cover of E is { B→ D, D→ A }

4.3 NORMAL FORMS BASED ON PRIMARY KEYS:

Normalization is a process of analyzing the given relation schemas based on their Functional Dependencies and primary keys to achieve the desirable properties of

(1) Minimizing redundancy and
(2) Minimizing the insertion, deletion, and update anomalies

4.3.1 Normalization of Relations:

- The normalization process, as first proposed by Codd (l972), takes a relation schema through a series of tests to "certify" whether it satisfies a certain normal form.
- Codd proposed three normal forms: 1NF, 2NF, and 3NF.
- The process proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary. It is also called as relational design by analysis.
- Thus, the normalization procedure provides database designers with the following:
  i) A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.

ii) 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.

- 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.
- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas should possess. These would include two properties:

a) The lossless join or nonadditive join property: This guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.

b) The dependency preservation property: This ensures that each functional dependency is represented in some individual relation resulting after decomposition.

- The process of storing the join of higher normal form relations as a base relation-which is in a lower normal form-is known as "**denormalization**". This is sometimes done for some performance reasons.

Practical Use of Normal Forms

- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties stated previously.
- Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

4.3.2 Definitions of Keys and Attributes Participating in Keys:

a) **Definition:** A **superkey** of a relation schema $\mathbf{R = \{A_1, A_2, \ldots, A_n\}}$ is a set of attributes $\mathbf{S \subseteq R}$ with the property that no two tuples 't1' and 't2' in any legal relation state 'r' of 'R' will have t1[S] = t2[S].

b) **Definition:** 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 $\mathbf{K = \{A_1, A_2, \ldots, A_k\}}$ of 'R', then $\mathbf{K - \{A_i\}}$ is not a key of 'R' for any $\mathbf{A_i}$ where $\mathbf{1 \leq i \leq k}$.

In the following figure, **{SSN} is a key** for EMPLOYEE, whereas **{SSN}, {SSN, ENAME}, {SSN, ENAME, BDATE},** and any set of attributes that includes SSN are all **superkeys**.

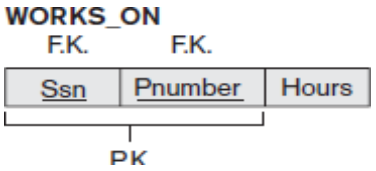
c) **Definition:** If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**. Each relation schema must have a primary key.

{SSN} is the only candidate key for EMPLOYEE, so it is also the primary key.

d) **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 the following figure, 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 (INF)** is defined to disallow ***multivalued attributes***, ***composite attributes***, and ***their combinations***.
- 1NF 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".
- Consider the **DEPARTMENT** relation schema shown in **Figure 6.7a**, whose primary key is **DNUMBER**, and suppose that we extend it by including the **DLOCATIONS** attribute as shown in **Figure 6.7a**. We assume that each department can have a number of locations. The example relation state for DEPARTMENT is shown in **Figure 6.7b**.

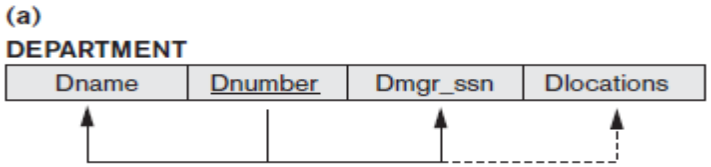


FIGURE 4.7: Normalization into 1NF. (a) A relation schema that is not in 1NF.

(b)

DEPARTMENT

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

FIGURE 4.7: Normalization into 1NF. (b) Example state of relation DEPARTMENT without normalization.

- As we can see, **state of relation DEPARTMENT** in **Figure 4.7 is not in 1NF because DLOCATIONS is not an atomic attribute.**

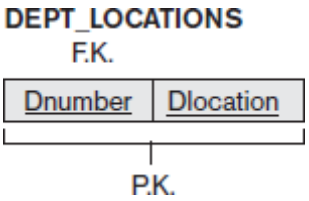
There are three main techniques to achieve first normal form for such a relation:

a) Remove the attribute DLOCATIONS that violates 1NF and place it in a **separate relation** DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT.

i) The primary key of this relation is the combination {DNUMBER, DLOCATION}, as shown in following Figure.

ii) A distinct tuple in DEPT_LOCATIONS exists for each location of a department.

iii) This decomposes the non-1NF relation into two 1NFrelations.



b) 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 the following Figure 6.8.

i) In this case, the primary key becomes the combination {DNUMBER, DLOCATION}.

ii) 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

Figure 6.8: 1NF version of DEPARTMENT relation with redundancy.

c) 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**: DLOCATIONl, DLOCATION2, and DLOCATION3.

i)

DNAME	DNUMBER	DMGRSSN	DLOCATION1	DLOCATION2	DLOCATION3

This solution has the disadvantage of introducing ***null values*** if most departments have fewer than three locations.

ii) It further introduces a spurious semantics about the ordering among the location values that is not originally intended.

- 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.
- 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 4.9a** shows how the **EMP_PROJ** relation schema 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 ( ).

(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

FIGURE 4.9 Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a "nested relation" attribute PROJS. (b) Example extension of the EMPROJ relation showing nested relations within each tuple.

To normalize this into INF, 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_PROJl and EMP_PROJ2.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

4.3.4 Second Normal Form:

- 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 € X, (X - {A}) does not functionally determine 'Y'.
- A functional dependency X→Y is a partial dependency if some attribute A € X can be removed from 'X' and the dependency still holds.
  That is, for some A € X, (X - {A}) → Y.

- In the following **Figure 6.10**, **{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.

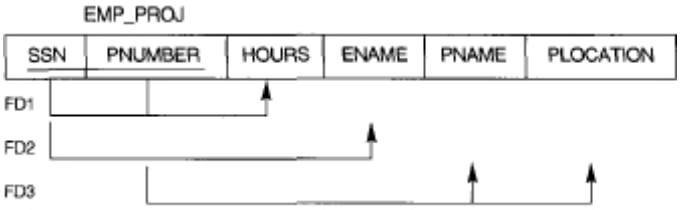
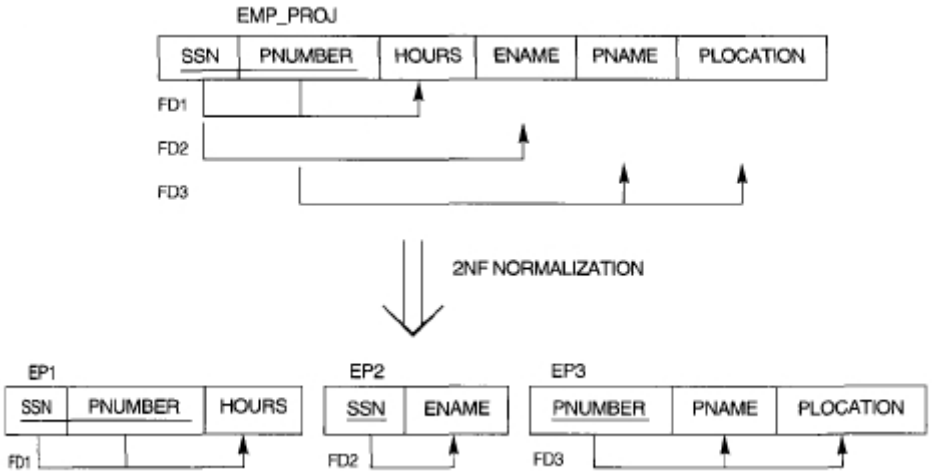


Figure 6.10: Relation EMP_PROJ is in 1NF but not in 2NF

- **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'. or 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 is a primary key composed of multiple attributes. If the primary key contains a single attribute, the test need not be applied at all.
- The EMP_PROJ relation in the above figure is in 1NF but is not in 2NF.
  - **a)** The nonprime attribute ENAME violates 2NF because of FD2. ENAME is partially dependent on {SSN,PNUMBER} and not dependent on PNUMBER.(Given ENAME can be determined only by SSN. So the other attributes are not needed for that table)
  - **b)** The nonprime attributes PNAME and PLOCATION violates 2NF because of FD3. PNAME and PLOCATION are partially dependent on {SSN,PNUMBER} and not dependent on SSN.
- The functional dependencies **FD1, FD2** and **FD3** in **Figure 4.10** hence lead to the decomposition of EMP_PROJ into the three relation schemas **EPl, EP2**, and EP3 shown below, each of which is in 2NF.



4.3.4 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 is a set of attributes 'Z' that is neither a candidate key nor a subset of any key of R, and both X→Z and Z→Y hold.
- **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. A relation schema 'R' is in **third** normal form (3NF) if, whenever a *nontrivial* functional dependency X → A holds in 'R', either
  (a) 'X' is a superkey of 'R', or
  (b) 'A' is a prime attribute of R.
- The dependency **SSN→ DMGRSSN** is transitive through **DNUMBER in EMP_DEPT** of **Figure 4.11** because:
  a) Both the dependencies **SSN→DNUMBER** and **DNUMBER→DMGRSSN** hold.
  b) **DNUMBER** is neither a key itself nor a subset of the key of **EMP_DEPT**.
  c) 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.

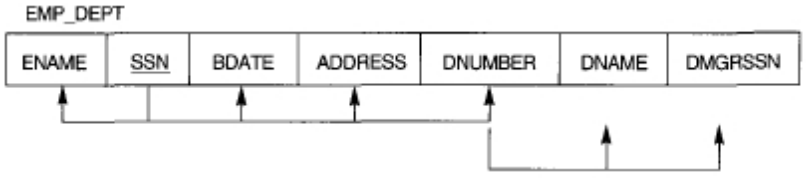


Figure 4.11 Relation EMP_DEPT is in 1NF and 2NF but not in 3NF

- The relation schema EMP_DEPT in Figure 4.11 is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN (and also DNAME) on SSN via DNUMBER.
- We can normalize EMP_DEPT by **decomposing** it into the two 3NF relation schemas **EDl** and **ED2** shown in following Figure.

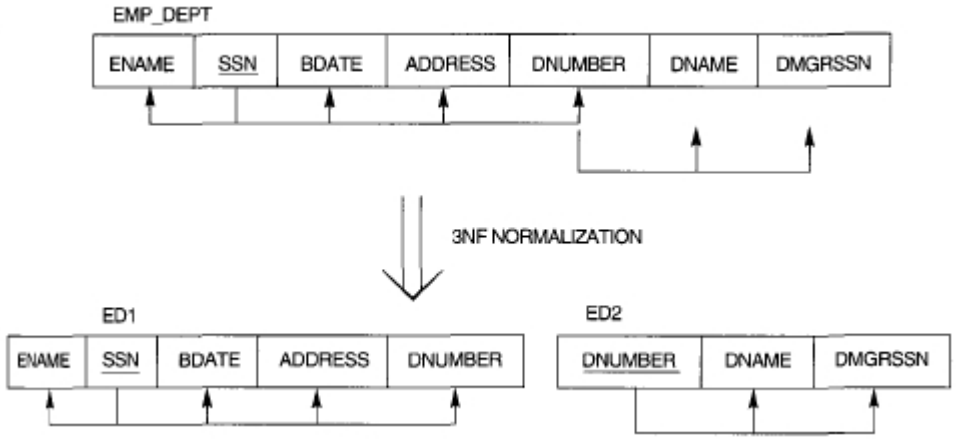


Table 6.1 informally summarizes the three normal forms based on primary keys, the tests used in each case, and the corresponding "remedy" or normalization performed to achieve the normal form.

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 6.1: SUMMARY OF NORMAL FORMS BASED ON PRIMARY KEYS AND CORRESPONDING NORMALIZATION

4.4. EXAMPLE:

- Suppose that there are two candidate keys: 1) **PROPERTY_ID#** and 2) **{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.

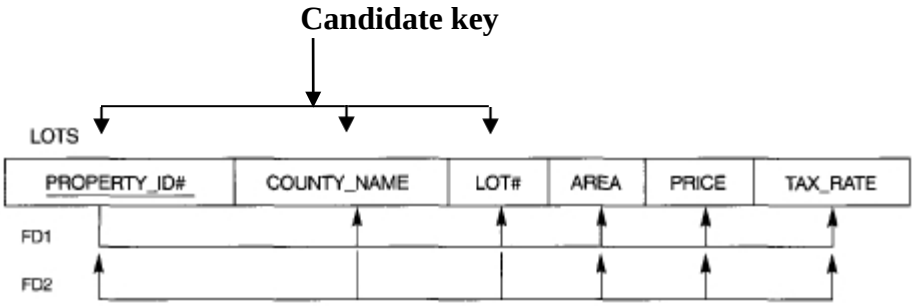


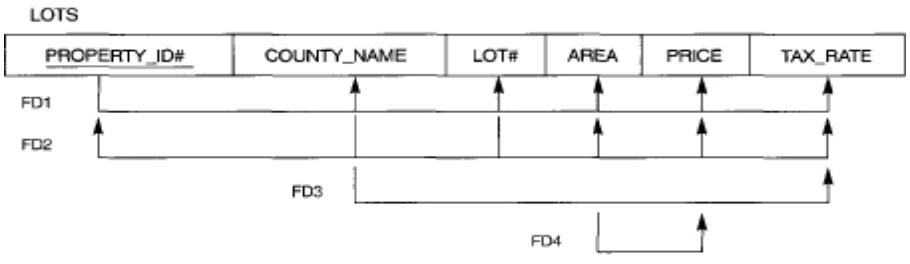
Figure 4.12: The LOTS relation with its functional dependencies FD1 and FD2

- Based on the two candidate keys **PROPERTY_ID#** and **{COUNTY_NAME, LOT#}**, the functional dependencies **FD1** and **FD2** of **Figure 4.12** hold.

- We choose **PROPERTY_ID#** as the **primary key**, so it is underlined in **Figure 4.12.**

- Suppose that the following two additional functional dependencies hold in LOTS:

FD3: COUNTY_NAME → TAX_RATE
FD4: AREA → PRICE



- The LOTS relation schema violates the general definition of 2NF as ***TAX_RATE is partially dependent on the candidate key {COUNTY_NAME, LOT#}*** because:
  - **a)** Due to FD2 **{COUNTY_NAME, LOT#} → TAX_RATE**
  - **b)** Due to FD3 {**COUNTY_NAME**} → **TAX_RATE**

- ***To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2***, shown below.

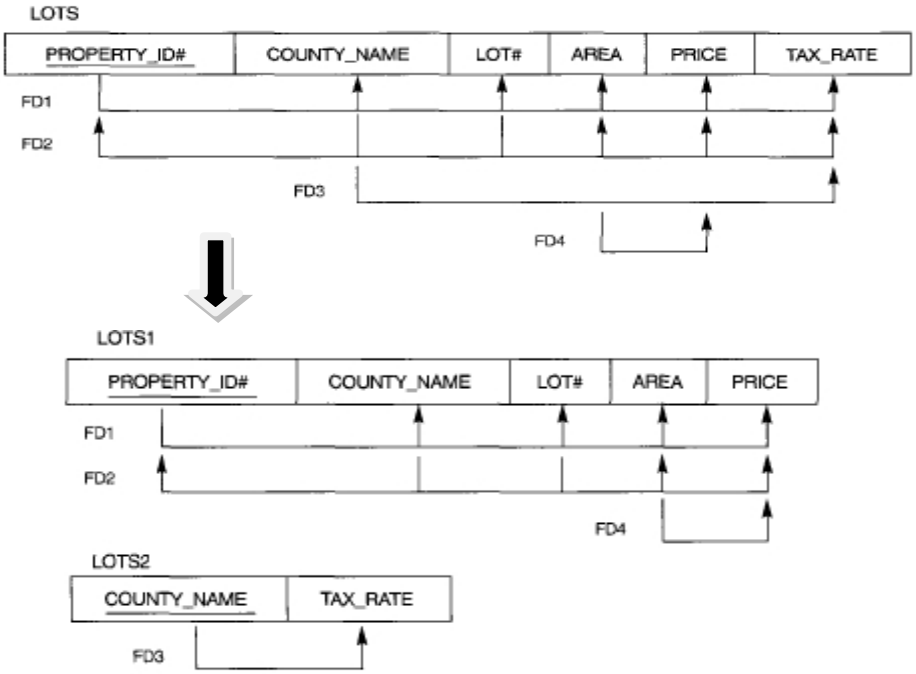


Figure 4.13: LOTS1 and LOTS2 in 2NF

➢ To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B as shown in Figure 4.14

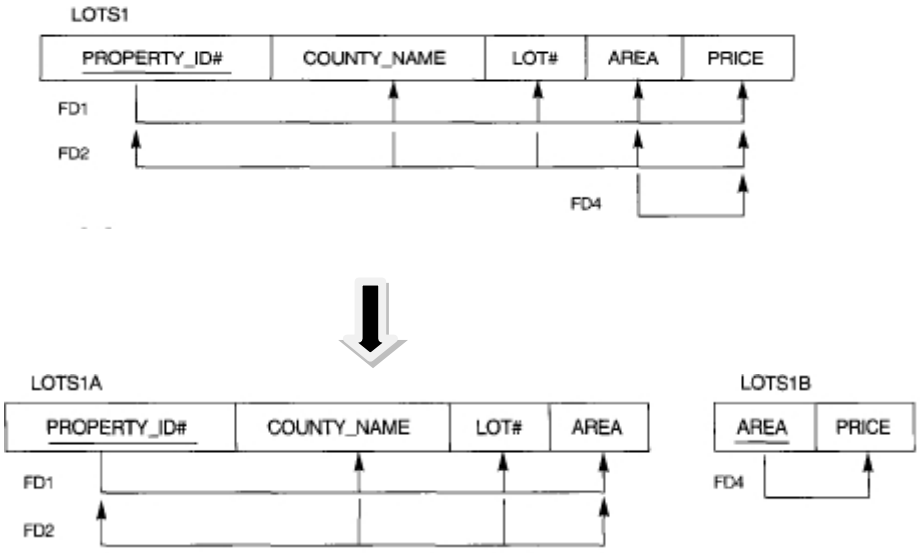


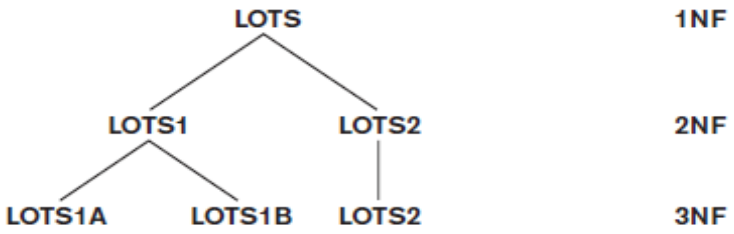
Figure 4.14: LOTS1A and LOTS1B in 3NF

a) We construct LOTS1A by removing the attribute PRICE that violates 3NF from LOTS1 and placing it with AREA (the left-hand side of FD4 that causes the transitive dependency) into another relation LOTS1B.

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

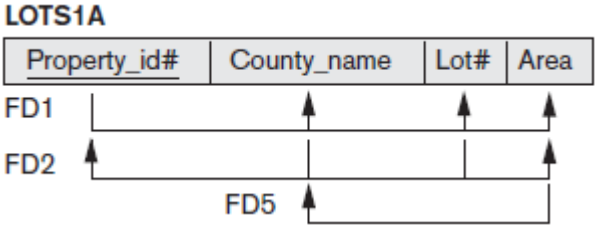
a) LOTS1 violates 3NF because PRICE is transitively dependent on each of the candidate keys of LOTS1 via the nonprime attribute AREA.

b) we find that *both* FD3 and FD4 violate 3NF. We could hence decompose LOTS into LOTSlA, LOTSlB, and LOTS2 directly.



4.5 BOYCE-CODD NORMAL FORM:

- Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
- ***Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF.***
- **Definition.** A relation schema R is in BCNF if whenever a *nontrivial* functional dependency X → A holds in R, then X is a superkey of R.
- Consider the relation schema of **LOTS1A** given below

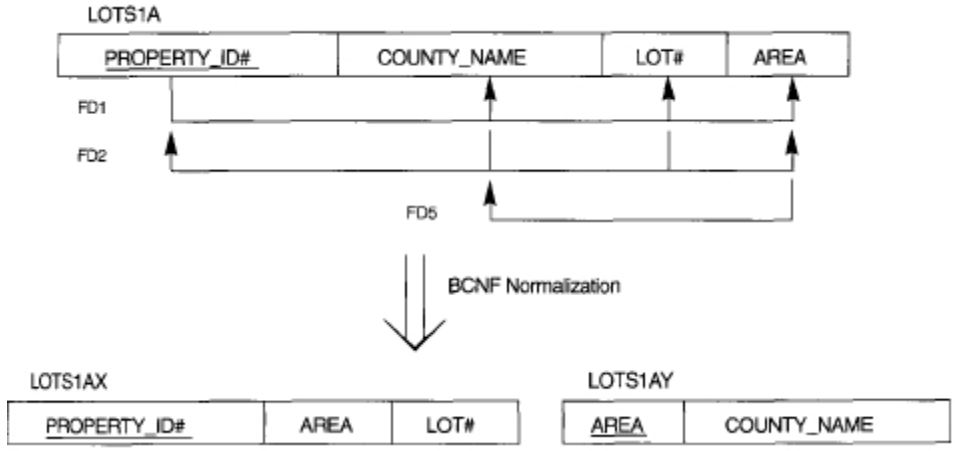


FD5: AREA → COUNTY_NAME

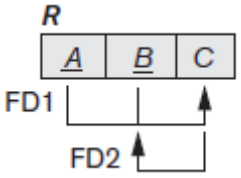
- The relation schema LOTS1A still is in 3NF because COUNTY_NAME *is* a prime attribute.

- ***The only difference between the definitions of BCNF and 3NF is that condition (b) of 3NF, which allows A to be prime, is absent from BCNF***.
- In our example, ***FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A***.
- Note that ***FD5 satisfies 3NF in LOTSIA because COUNTY_NAME is a prime attribute (condition b)***, but this condition does not exist in the definition of BCNF.

- We can decompose LOTSIA into two BCNF relations LOTS1AX and LOTS1AY as shown below.



- The relation schema R shown in following Figure illustrates the general case of a relation being in 3NF but not in BCNF.



4.6 MULTIVALUED DEPENDENCY (MVD) AND FOURTH NORMAL FORM

- 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**.

4.6.1 Formal Definition of Multivalued Dependency

Definition of Multivalued dependency: 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 t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote (R – (X ∪ Y)):

t3[X] = t4[X] = t1[X] = t2[X].
t3[Y] = t1[Y] and t4[Y] = t2[Y].
t3[Z] = t2[Z] and t4[Z] = t1[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 $X \twoheadrightarrow Y$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Y$, and therefore it is sometimes written as $X \twoheadrightarrow Y$.

Definition of 4NF: 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 →→ Y in F+ X is a superkey for R.

- Inference rules followed in 4NF are:

IR1 (reflexive rule for FDs): If $X \supseteq Y$, then $X \rightarrow Y$.
IR2 (augmentation rule for FDs): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.
IR3 (transitive rule for FDs): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
IR4 (complementation rule for MVDs): $\{X \twoheadrightarrow Y\} \models \{X \twoheadrightarrow (R - (X \cup Y))\}$.
IR5 (augmentation rule for MVDs): If $X \twoheadrightarrow Y$ and $W \supseteq Z$, then $WX \twoheadrightarrow YZ$.
IR6 (transitive rule for MVDs): $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.
IR7 (replication rule for FD to MVD): $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$.
IR8 (coalescence rule for FDs and MVDs): If $X \twoheadrightarrow Y$ and there exists W with the properties that (a) $W \cap Y$ is empty, (b) $W \rightarrow Z$, and (c) $Y \supseteq Z$, then $X \rightarrow Z$.

In the EMP relation of Figure 4.15(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).

In 4.15 (c), not every Sname determines various Part_name and not every Sname determines multiple Proj_name. so it is not MVD. Therefore it is in 4NF.

Example 1: Figure 4.15

Fourth and fifth normal forms.
(a) The EMP relation with two MVDs: Ename →→ Pname and Ename →→ Dname.
(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.

(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

Example 2:

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

4.7 JOIN DEPENDENCIES AND FIFTH NORMAL FORM (5NF)

Definition of join dependency: 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

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

Definition of 5 NF: 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*.

Figure 4.16(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.

(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.

(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

Figure 4.16

Algorithm 4.12(a). Finding a Key *K* for *R* Given a set *F* of Functional Dependencies:

Input: A relation *R* and a set of functional dependencies *F* on the attributes of *R*.

1. Set *K* := *R*.
2. For each attribute *A* in *K*

{compute (*K* – A)+ with respect to *F*;
if (K – A)+ contains all the attributes in R, then set K := K – {A} };

4.8 PROPERTIES OF RELATIONAL DECOMPOSITIONS

4.8.1 Relation Decomposition and Insufficiency of Normal Forms:

- Let universal relation schema R = {A1, A2, ..., An} that includes all the attributes of the database.
- We implicitly make the universal relation assumption, which states that every attribute name is unique.
- The set F of functional dependencies that should hold on the attributes of R is specified by the database designers and is made available to the design algorithms.
- Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas D = {R1, R2, ..., Rm} that will become the relational database schema; D is called a **decomposition of R**.
- We must make sure that each attribute in R will appear in at least one relation schema Ri in the decomposition so that no attributes are lost. This is called the **attribute preservation condition of a decomposition**.

$$\bigcup_{i=1}^{m} R_i = R$$

consider the EMP_LOCS(Ename, Plocation) relation in below, which is in 3NF and also in BCNF. In fact, any relation schema with only two attributes is automatically in BCNF. Although EMP_LOCS is in BCNF, it still gives rise to spurious tuples when joined with EMP_PROJ (Ssn, Pnumber, Hours, Pname, Plocation), which is not in BCNF.

(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

4.8.2 Dependency Preservation Property of a Decomposition

- It would be useful if each functional dependency X->Y specified in F either appeared directly in one of the relation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. This is the **dependency preservation condition.**
- We want to preserve the dependencies because each dependency in F represents a constraint on the database.
- If one of the dependencies is not represented in some individual relation Ri of the decomposition, we cannot enforce this constraint by dealing with an individual relation.
- We may have to join multiple relations so as to include all attributes involved in that dependency.
- Given a set of dependencies F on R, the projection of F on Ri, denoted by $\pi_{Ri}(F)$ where Ri is a subset of R, is the set of dependencies $X \rightarrow Y$ in F+ such that the attributes in $X \cup Y$ are all contained in Ri.
- If a decomposition is not dependency-preserving, some dependency is lost in the decomposition. To check that a lost dependency holds, we must take the JOIN of two or more relations in the decomposition to get a relation that includes all left and right-hand-side attributes of the lost dependency, and then check that the dependency holds on the result of the JOIN.

4.8.3 Nonadditive (Lossless) Join Property of a Decomposition

- The nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.
- Because this is a property of a decomposition of relation schemas, the condition of no spurious tuples should hold on every legal relation state—that is, every relation state that satisfies the functional dependencies in F.
- Hence, the lossless join property is always defined with respect to a specific set F of dependencies.

Definition: Formally, a decomposition D = {R1, R2, ..., Rm} of R has the lossless (nonadditive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all the relations in D: $*(\pi_{R1(r)}, ..., \pi_{Rm(r)}) = r$.

- The decomposition of EMP_PROJ(Ssn, Pnumber, Hours, Ename, Pname, Plocation) in into EMP_LOCS(Ename, Plocation) and EMP_PROJ1(Ssn, Pnumber, Hours, Pname, Plocation) does not have the nonadditive join property.

Algorithm 4.8. Testing for Nonadditive Join Property

Input: A universal relation R, a decomposition D = {R1, R2, ..., Rm} of R, and a set F of functional dependencies.

Note: comments follow the format: (* comment *).

1. Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute A_j in R.
2. Set S(i, j):= b_{ij} for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i, j) *).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
   {if (relation R_i includes attribute A_j) then set S(i, j):= a_j;};}; (* each a_j is a distinct symbol associated with index (j) *).
4. Repeat the following loop until a complete loop execution results in no changes to S
   {for each functional dependency X→Y in F
   {for all rows in S that have the same symbols in the columns corresponding to attributes in X
   {make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows:
   If any of the rows has an ***a*** symbol for the column, set the other rows to that same ***a*** symbol in the column.
   If no ***a*** symbol exists for the attribute in any of the rows, choose one of the ***b*** symbols that appears in one of the rows for the attribute and set the other rows to that same b symbol in the column}}}
5. If a row is made up entirely of '***a'*** symbols, then the decomposition has the nonadditive join property; otherwise, it does not.

Nonadditive join test for *n*-ary decompositions. (a) Case 1: Decomposition of EMP_PROJ into EMP_PROJ1 and EMP_LOCS fails test. (b) A decomposition of EMP_PROJ that has the lossless join property. (c) Case 2: Decomposition of EMP_PROJ into EMP, PROJECT, and WORKS_ON satisfies test.

(a) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2\}$

R_1 = EMP_LOCS = {Ename, Plocation}

R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(No changes to matrix after applying functional dependencies)

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

(c) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2, R_3\}$

R_1 = EMP = {Ssn, Ename}

R_2 = PROJ = {Pnumber, Pname, Plocation}

R_3 = WORKS_ON = {Ssn, Pnumber, Hours}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matrix S after applying the first two functional dependencies; last row is all "a" symbols so we stop)

Figure: 4.8

R1 (Emp_ssn , Esal, Ephone, Dno)
R2 (Pno, Pname, Plocation)
R3 (Emp_ssn, Pno)
This design achieves both the desirable properties of dependency preservation and nonadditive join.

4.8.4 Testing Binary Decompositions for the Nonadditive Join Property (NJB)

- There is a special case of a decomposition called a **binary decomposition**—decomposition of a relation R into two relations, Property NJB (Nonadditive Join Test for Binary Decompositions).
- A decomposition D = {R1, R2} of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R if and only if either
  The FD ((R1 ∩ R2)→(R1 – R2)) is in F+, or
  The FD ((R1 ∩ R2)→(R2 – R1)) is in F+

- Example decomposition of the TEACH(Instructor, Course, Student) relation into the two relations {Instructor, Course} and {Instructor, Student}. These are valid decompositions because they are nonadditive per the above test.

4.8.5 Successive Nonadditive Join Decompositions

If a decomposition D = {R1, R2, ..., Rm} of R has the nonadditive (lossless) join property with respect to a set of functional dependencies F on R, and if a decomposition Di = {Q1, Q2, ..., Qk} of Ri has the nonadditive join property with respect to the projection of F on Ri,
then the decomposition D2 = {R1, R2, ..., Ri−1, Q1, Q2, ..., Qk, R_{i+1}, ..., R_m} of R has the nonadditive join property with respect to F.

4.9 ABOUT NULLS, DANGLING TUPLES, AND ALTERNATIVE RELATIONAL DESIGNS

4.9.1 Problems with NULL Values and Dangling Tuples:

- We must carefully consider the problems associated with NULLs when designing a relational database schema.
- One problem occurs when some tuples have NULL values for attributes that will be used to join individual relations in the decomposition.

To illustrate this, consider the database shown in Figure 16.2(a), where two The last two employee tuples— ‘Berger’ and ‘Benitez’—represent newly hired employees who have not yet been assigned to a department).
Now suppose that we want to retrieve a list of (Ename, Dname) values for all the employees. If we apply the NATURAL JOIN operation on EMPLOYEE and DEPARTMENT (Figure 16.2(b)), the two forementioned tuples will *not* appear in the result. The OUTER JOIN operation can deal with this problem.

- In general, whenever a relational database schema is designed in which two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential NULL values in foreign keys. This can cause unexpected loss of information in queries that involve joins on that foreign key. Moreover, if NULLs occur in other attributes, such as Salary, their effect on built-in functions such as SUM and AVERAGE must be carefully evaluated.
- A related problem is that of ***dangling tuples***, which may occur if we carry a decomposition too far. Suppose that we decompose the EMPLOYEE relation in Figure 16.2(a) further into EMPLOYEE_1 and EMPLOYEE_2, shown in Figure 16.3(a) and 16.3(b).

If we apply the NATURAL JOIN operation to EMPLOYEE_1 and EMPLOYEE_2, we get the original EMPLOYEE relation. However, we may use the alternative representation, shown in Figure 16.3(c), where we *do not include a tuple* in EMPLOYEE_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE_2).

If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result; these are called **dangling tuples** in EMPLOYEE_1 because they are represented in only one of the two relations that represent employees, and hence are lost if we apply an
(INNER) JOIN operation.

Figure 16.2 (a)

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
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
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	NULL

(a) EMPLOYEE_1

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

(b) EMPLOYEE_2

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLOYEE_3

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure 16.3
The dangling tuple problem.
(a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 16.2(a) except Dnum).
(b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values).
(c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

4.10 OTHER DEPENDENCIES AND NORMAL

4.11 FORMS Other types of dependencies:

4.10.1 Inclusion Dependencies

Inclusion dependencies were defined in order to formalize two types of inter relational constraints:

- The foreign key (or referential integrity) constraint cannot be specified as a functional or multivalued dependency because it relates attributes across relations.

Definition: An inclusion dependency R.X < S.Y between two sets of attributes— X of relation schema R, and Y of relation schema S—specifies the constraint that, at any specific time when r is a relation state of R and s a relation state of S, we must have

$$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

For example, we can specify the following inclusion dependencies on the relational schema in which represent represent referential integrity constraints:

DEPARTMENT.Dmgr_ssn < EMPLOYEE.Ssn
WORKS_ON.Ssn < EMPLOYEE.Ssn
EMPLOYEE.Dnumber < DEPARTMENT.Dnumber
PROJECT.Dnum < DEPARTMENT.Dnumber
WORKS_ON.Pnumber < PROJECT.Pnumber
DEPT_LOCATIONS.Dnumber < DEPARTMENT.Dnumber

We can also use inclusion dependencies to represent class/subclass relationships. For example, we can

specify the following inclusion dependencies:

EMPLOYEE.Ssn < PERSON.Ssn
STUDENT.Ssn < PERSON.Ssn

As with other types of dependencies, there are inclusion dependency inference rules (IDIRs). The following are three examples:

IDIR1 (reflexivity): R.X < R.X.

IDIR2 (**attribute correspondence):** If R.X < S.Y, where X = {A1, A2, ..., An} and Y = {B1, B2, ..., Bn} and Ai corresponds to Bi, then R.Ai < S.Bi for $1 \leq i \leq n$.

IDIR3 (transitivity): If R.X < S.Y and S.Y < T.Z, then R.X < T.Z.

4.10.2 Template Dependencies

- The idea behind template dependencies is to specify a template—or example—that defines each constraint or dependency.
- There are two types of templates: tuple-generating templates and constraint generating templates.
- A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations.The other part of the template is the template conclusion.
- For **tuple-generating templates**, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there.
- For **constraint-generating templates**, the template conclusion is a condition that must hold on the hypothesis tuples.

Figure 16.5
Templates for some common type of dependencies.
(a) Template for functional dependency X → Y.
(b) Template for the multivalued dependency X →→ Y.
(c) Template for the inclusion dependency R.X < S.Y.

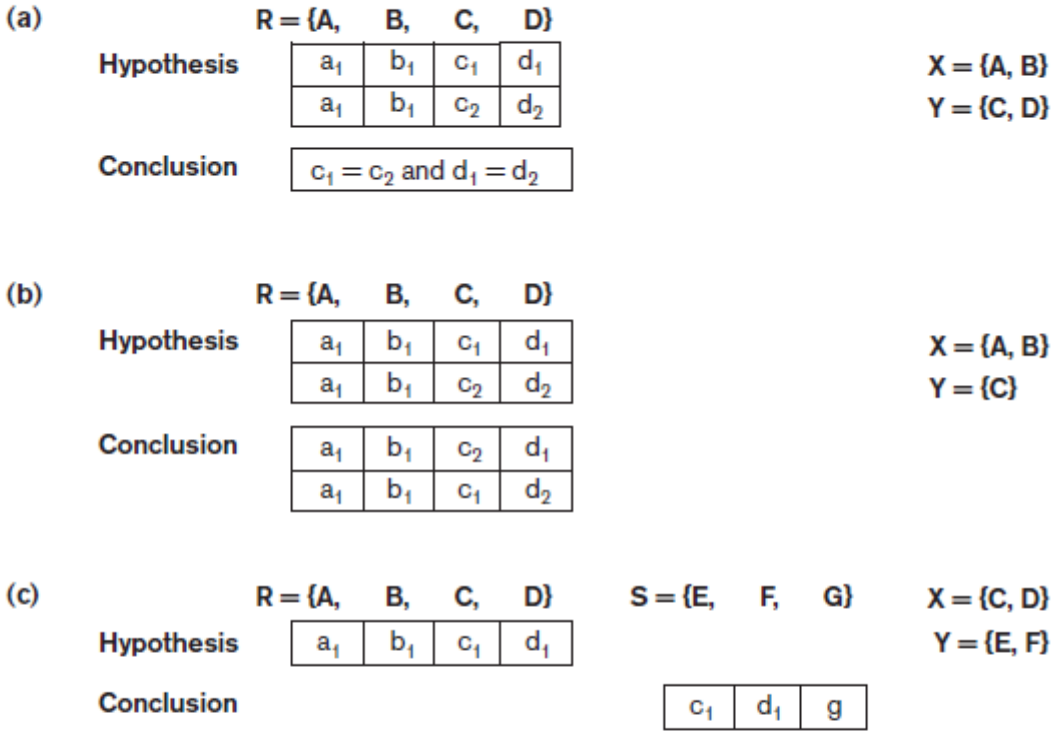
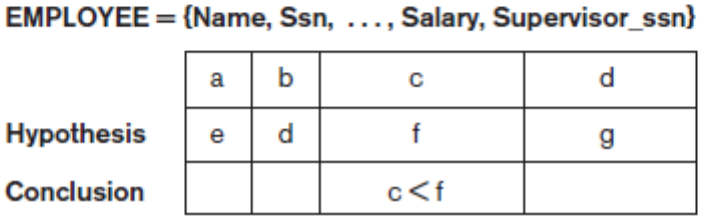


Figure 16.6
Templates for the constraint that an employee's salary must be less than the supervisor's salary.



4.10.3 Functional Dependencies Based on Arithmetic Functions and Procedures

- Sometimes some attributes in a relation may be related via some arithmetic function or a more complicated functional relationship.

For example, in the relation

ORDER_LINE (Order#, Item#, Quantity, Unit_price, Extended_price, Discounted_price)

each tuple represents an item from an order with a particular quantity, and the price per unit for that item.

In this relation, **(Quantity, Unit_price)→Extended_price** by the formula
Extended_price = Unit_price * Quantity.

Hence, there is a unique value for Extended_price for every pair (Quantity, Unit_price),
and thus it conforms to the definition of functional dependency.

- Therefore, we can say
  (Item#, Quantity, Unit_price) → Discounted_price, or
  (Item#, Quantity, Extended_price) → Discounted_price.

4.10.4 Domain-Key Normal Form

- The idea behind **domain-key normal form** (**DKNF**) is to specify (theoretically, at least) the *ultimate normal form* that takes into account all possible types of dependencies and constraints.
- A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation.
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- However, because of the difficulty of including complex constraints in a DKNF relation,

its practical utility is limited,

For example, consider a relation CAR(Make, Vin#) (where Vin# is the vehicle identification number) and another relation MANUFACTURE(Vin#, Country) (where Country is the country of manufacture).
A general constraint may be of the following form:
If the Make is either 'Toyota' or 'Lexus,' then the first character of the Vin# is a 'J' if the country of manufacture is 'Japan'; if the Make is 'Honda' or 'Acura,' the second character of the Vin# is a 'J' if the country of manufacture is 'Japan.'