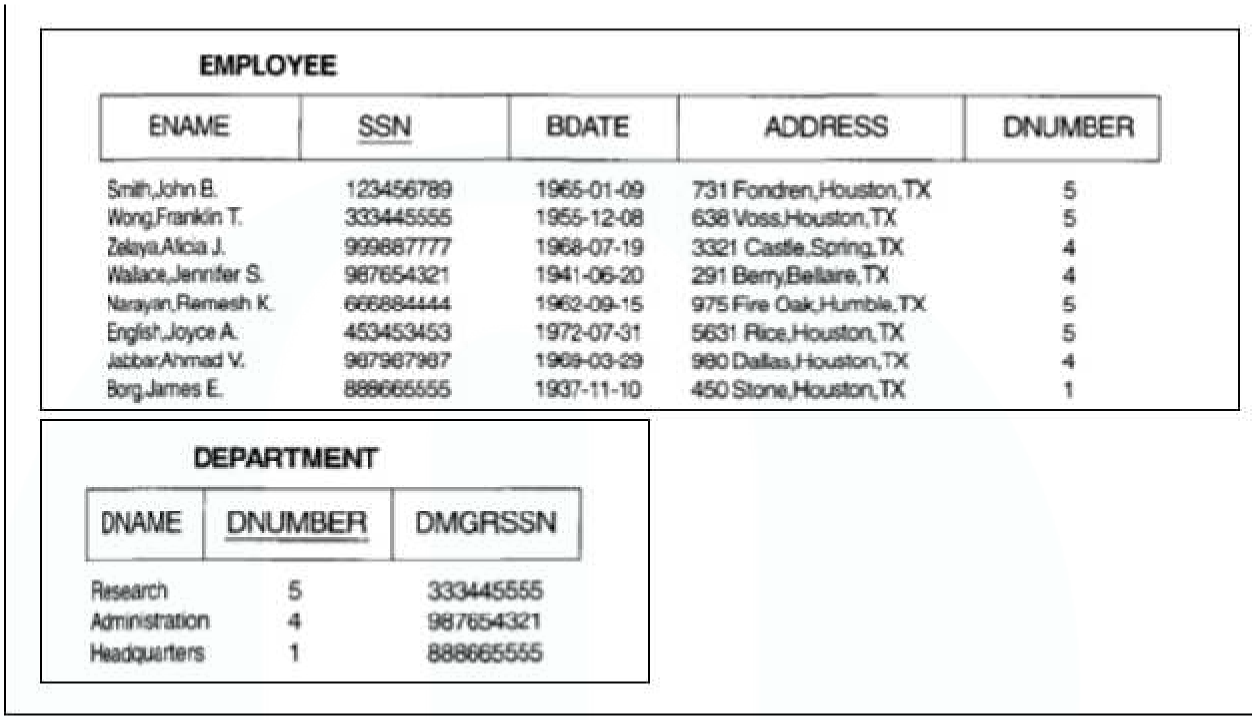
Module IV

DIFFERENT ANOMALIES IN DESIGNING A DATABASE



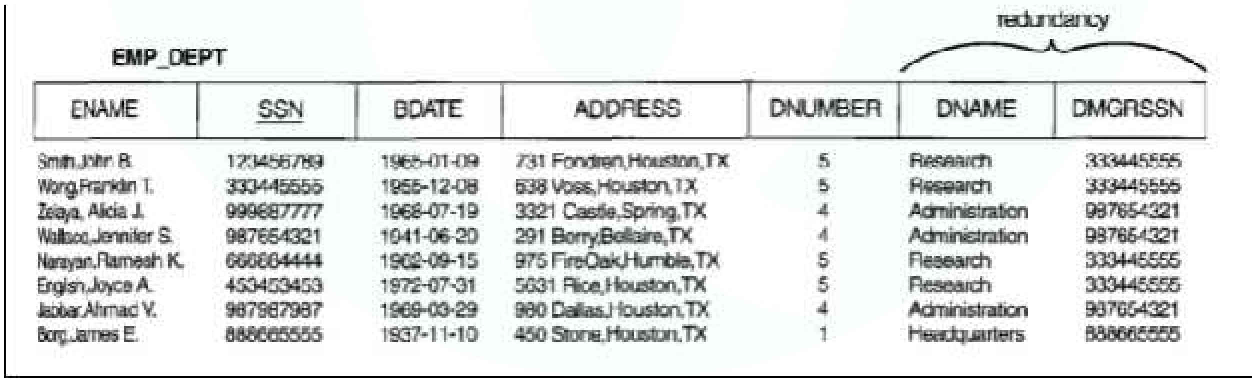
EMPLOYEE

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

DEPARTMENT

DNAME	DNUMBER	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Figure 1



redundancy

EMP_DEPT

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN
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

Figure 2

Update anomalies can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

(a)Insertion anomalies:

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

• **To insert a new employee tuple** into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or nulls (if the employee does not work for a department as yet).

For example, to insert a new tuple for an employee who works in department number 5, we must enter 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 1**, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.

• It is difficult **to insert a new department that has no employees** as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This 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 1**, 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 1 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.

FUNCTIONAL DEPENDENCY (FD)

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 tl 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) **determines** 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 .**

Example:

Consider 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).
Alternatively, we say that ENAME is **functionally determined** by (or functionally dependent on) SSN.

(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).

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

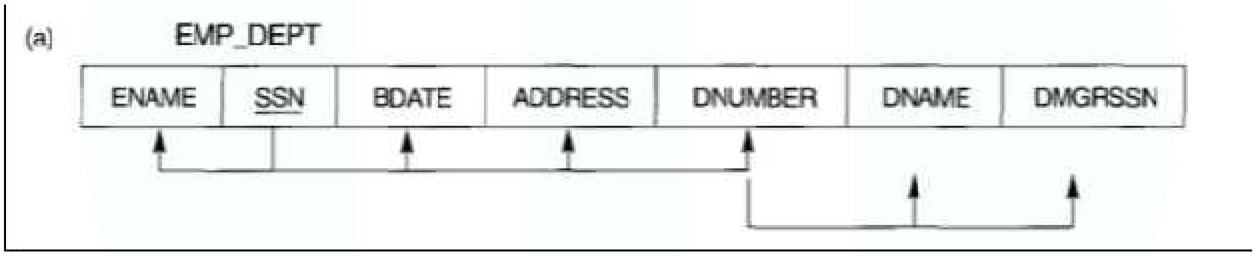
• If a constraint on R states that there cannot be more than one tuple with a given X value in any relation instance r(R)-that is, X is **a candidate key of R-**this implies thatX→ Yfor any subset of attributes Yof R

• If X→Y in R, this does not say whether or not Y → X in R.

CLOSURES

Definition. 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 Figure below:



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

DNUMBER → {DNAME, DMGRSSN}}

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

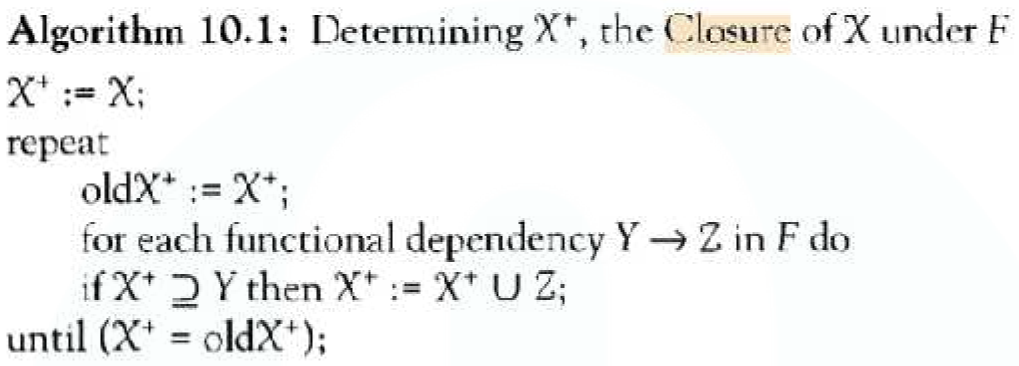
SSN → {DNAME, DMGRSSN}

SSN → SSN

DNUMBER → DNAME

An FD X → Y is inferred from a set of dependencies F specified on R if X → Y holds in every legal relation state r of R; that is, whenever r satisfies all the dependencies in F, X → Y also holds in r. The closure F^+ of F is the set of all functional dependencies that can be inferred from F.

To determine a systematic way to infer dependencies, we must **discover a set of inference rules** that 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.



Algorithm 10.1: Determining X^+, the Closure of X under F

```
$X^+ := X$;
repeat
    $oldX^+ := X^+$;
    for each functional dependency $Y \rightarrow Z$ in F do
    if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;
until ($X^+ = oldX^+$);
```

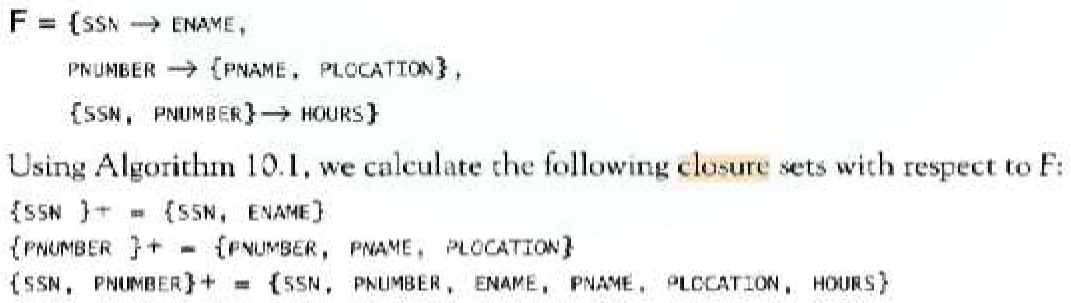
Algorithm 10.1 starts by setting X^+ to all the attributes in X.

By IRI, we know that all these attributes are functionally dependent on X.

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

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.

Example



```
F = {SSN → ENAME,
     PNUMBER → {PNAME, PLOCATION},
     {SSN, PNUMBER}→ HOURS}
```

Using Algorithm 10.1, 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}
```

ARMSTRONG'S AXIOMS

The following **six rules** IRI through IR6 are well known **inference rules for functional dependencies:**

IR1 (reflexive rule[8]): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (augmentation rule[9]): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

IR4 (decomposition, or projective, rule): $\{X \rightarrow YZ\} \models X \rightarrow Y$.

IR5 (union, or additive, rule): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.

IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

Inference rules **IR1 through IR3** are known as **Armstrong's inference rules or Armstrong's axioms.**

The reflexive rule (IR1) states that a set of attributes always determines itself or any of its subsets, which is obvious.

Because **IRl** generates dependencies that are always true, such dependencies are called **trivial.** Formally, a functional dependency X → Y is trivial if X ⊇Y; otherwise, it is nontrivial.

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.

According to **IR3**, functional dependencies are transitive.

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}

The union rule **(IR5)** 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}

<u>EQUIVALENCE OF FDS,</u>

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.

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.

<u>MINIMAL COVER (PROOFS NOT REQUIRED).</u>

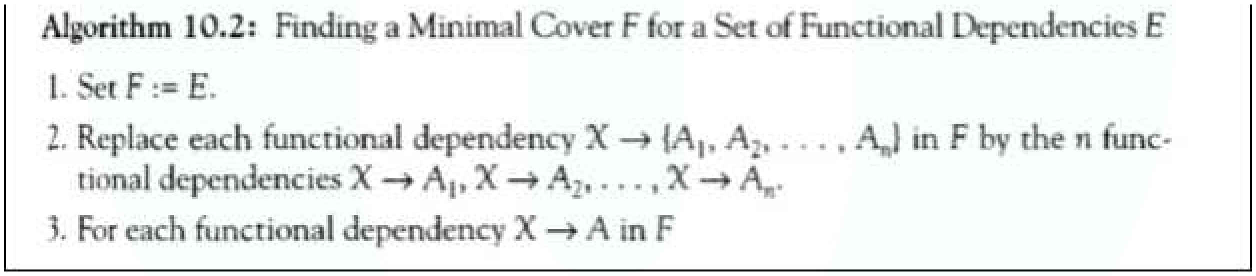
Minimal Sets of Functional Dependencies

To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions;

1.Every dependency in F has a single attribute for its right-hand side.

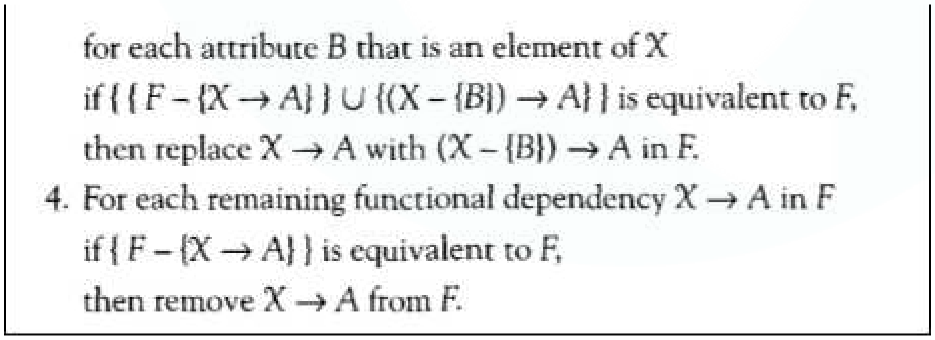
2. We cannot replace any dependency X → A in F with a dependency Y→ A, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to E

3.We cannot remove any dependency from F and still have a set of dependencies that is equivalent to E.



Algorithm 10.2: Finding a Minimal Cover F for a Set of Functional Dependencies E

```
1. Set $F := E$.
2. Replace each functional dependency $X \to \{A_1, A_2, \ldots, A_n\}$ in $F$ by the $n$ func-
   tional dependencies $X \to A_1, X \to A_2, \ldots, X \to A_n$.
3. For each functional dependency $X \to A$ in $F$
```



```
   for each attribute $B$ that is an element of $X$
   if $\{ \{F - \{X \to A\}\} \cup \{(X - \{B\}) \to A\} \}$ is equivalent to $F$,
   then replace $X \to A$ with $(X - \{B\}) \to A$ in $F$.
4. For each remaining functional dependency $X \to A$ in $F$
   if $\{F - \{X \to A\}\}$ is equivalent to $F$,
   then remove $X \to A$ from $F$.
```

NORMALIZATION USING FUNCTIONAL DEPENDENCIES,

Normalization: The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties

Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

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

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 sothat the relational database can be normalized to any desired degree

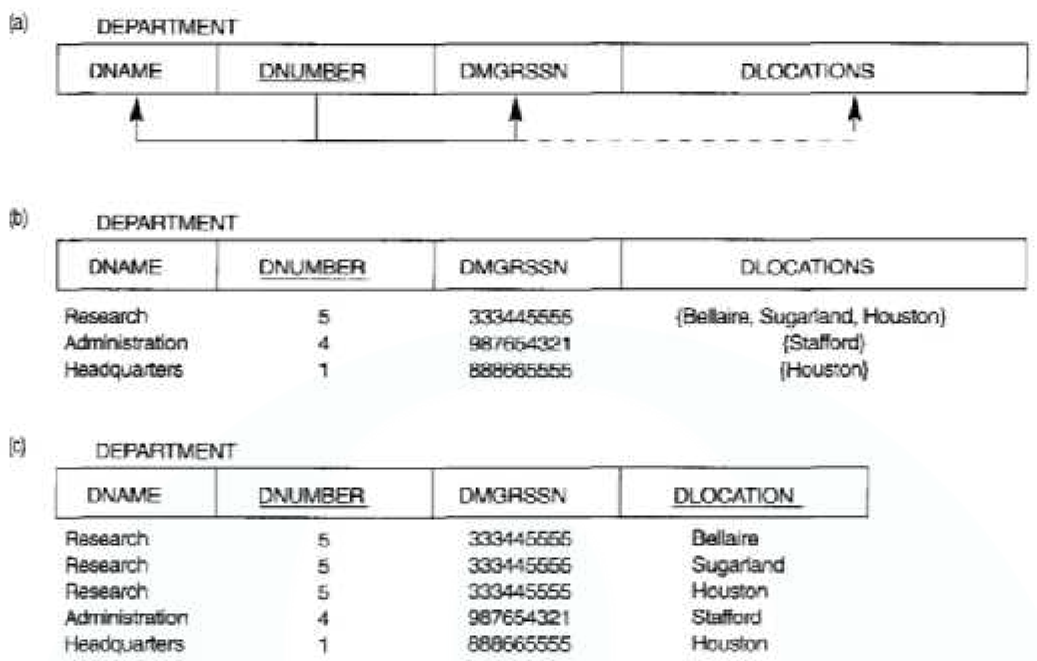
- An attribute of relation schema R is called a **prime attribute** of R if it is a **member of some candidate** key of R.
- An attribute is **called nonprime if** it is not a prime attribute-that is, if it is **not a member** of any candidate key.

1NF

Relation should have no non atomic attributes or nested relations.

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.

Consider the DEPARTMENT relation schema shown in Figure whose primary key is DNUMBER. We assume that each department can have a number of locations. The DEPARTMENT schema and an example relation state are shown in Figure 10.8. As we can see, this is **not in 1NF** because DLOCATIONS is not an atomic attribute, as illustrated by the first tuple .



(b) DEPARTMENT

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

(c) DEPARTMENT

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

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

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

1. Remove the attribute DLOCATIONS that violates 1NF **and place it in a separate relation** DEPT_LOCATIONS along with the primary key DNUMBER of DEPARTMENT. The primary key of this relation is the combination {DNUMBER, DLOCATION},as shown in Figure 10.2. A distinct tuple in DEPT_LOCATIONS exists for each location of a department. This decomposes the non-1NF relation into two 1NF relations.

DEPARTMENT

DNAME	DNUMBER	DMGRSSN
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 10.8c. In this case, the primary key

becomes the combination {DNUMBER, DLOCATION}. This solution has the **disadvantage** of introducing **redundancy** in the relation.

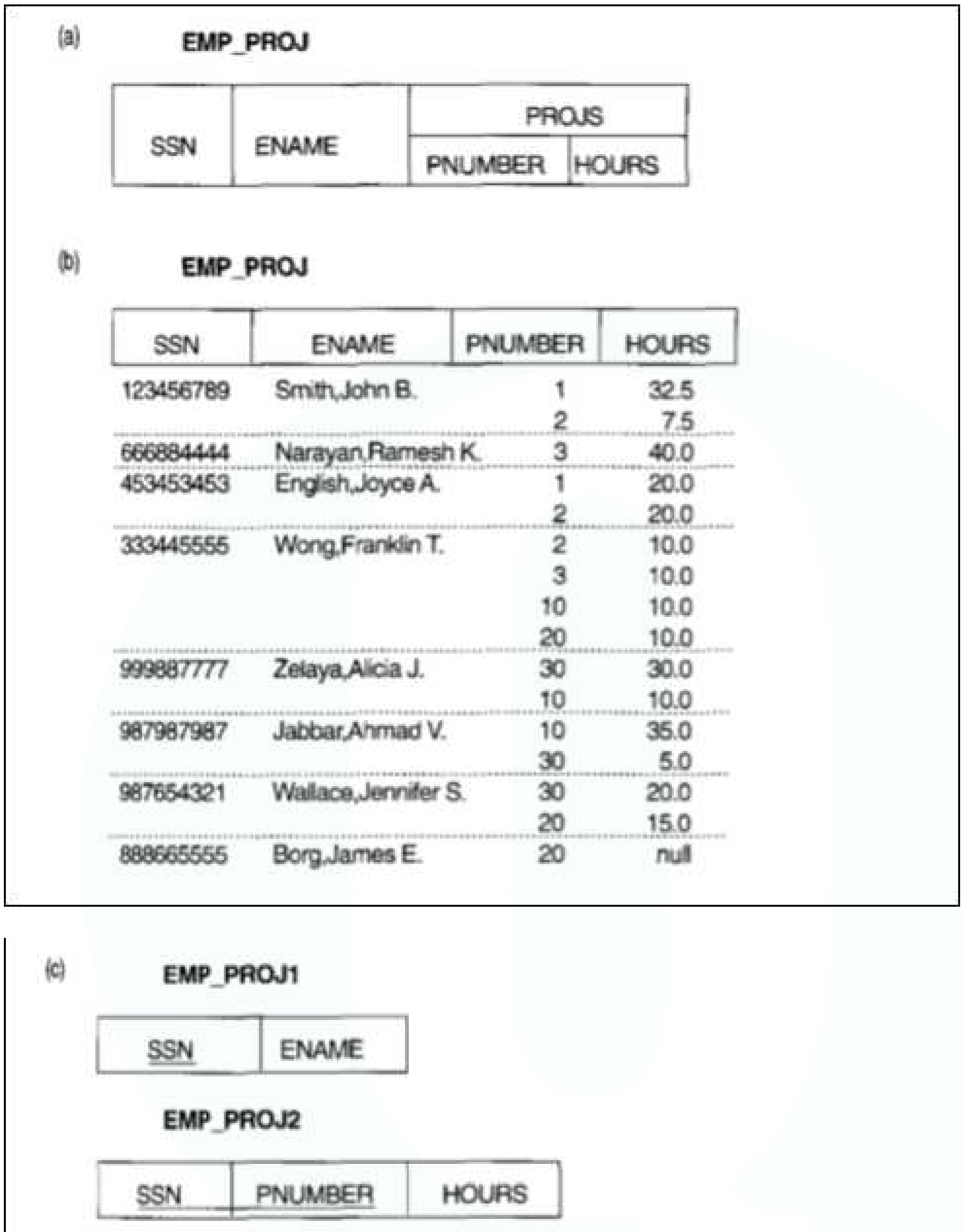
3. If a **maximum number of values is known** for the attribute-for example, if it is known that at most **three locations** can exist for a department-replace the DLOCATIONS attribute by three atomic attributes: DLOCATION l, DLOCATION 2, and DLOCATION 3. This solution has the **disadvantage** of introducing **null values** if most departments have fewer than three locations. I

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 10.9 shows how the EMP_PROJ relation could appear if nesting is allowed.

Notice that SSN is the primary key of the EMP_PROJ relation in Figures 10.9a and b, while PNUMBER is the partial key of the nested relation; that is, within each tuple, the nested relation must have unique values of PNUMBER.

To normalize this into 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_ PROJ 1 and EMP_PROJ2 shown in Figure 10.9c.



(a) EMP_PROJ

SSN	ENAME	PROJS	
		PNUMBER	HOURS

(b) EMP_PROJ

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

(c) EMP_PROJ1

SSN	ENAME

EMP_PROJ2

SSN	PNUMBER	HOURS

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

2NF

For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.

Formally, 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.

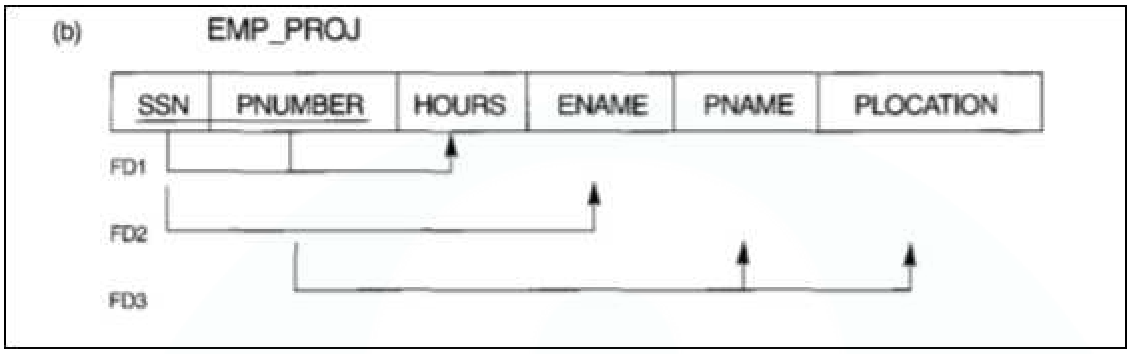


Fig 10.3b

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 l0.3b, {SSN, PNUMBER} → HOURS is a **full dependency** (neither SSN → HOURS nor PNUMBER → HOURS holds).

However, the dependency {SSN, PNUMBER} →ENAME is **partial** because SSN →ENAME holds.(FD2)

The test for 2NF involves testing for functional dependencies whose **left-hand side attributes are part of the primary key**.

If the primary key contains a **single attribute**, the test need not be applied at all.

The EMP_PROJ relation in Figure 10.3b is in INF but is not in 2NF.

The nonprime attribute ENAME violates 2NF because of **FD2(** ssn → ename ,on the left side both ssn and pnumber should be present**)**, as do the nonprime attributes PNAME and PLOCATION because of **FD3(** pnumber → pname, plocation , on the left side both ssn and pnumber should be present**)**.

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" by decomposition of FD1,FD2,FD3 of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 10.lOa, each of which is in 2NF.

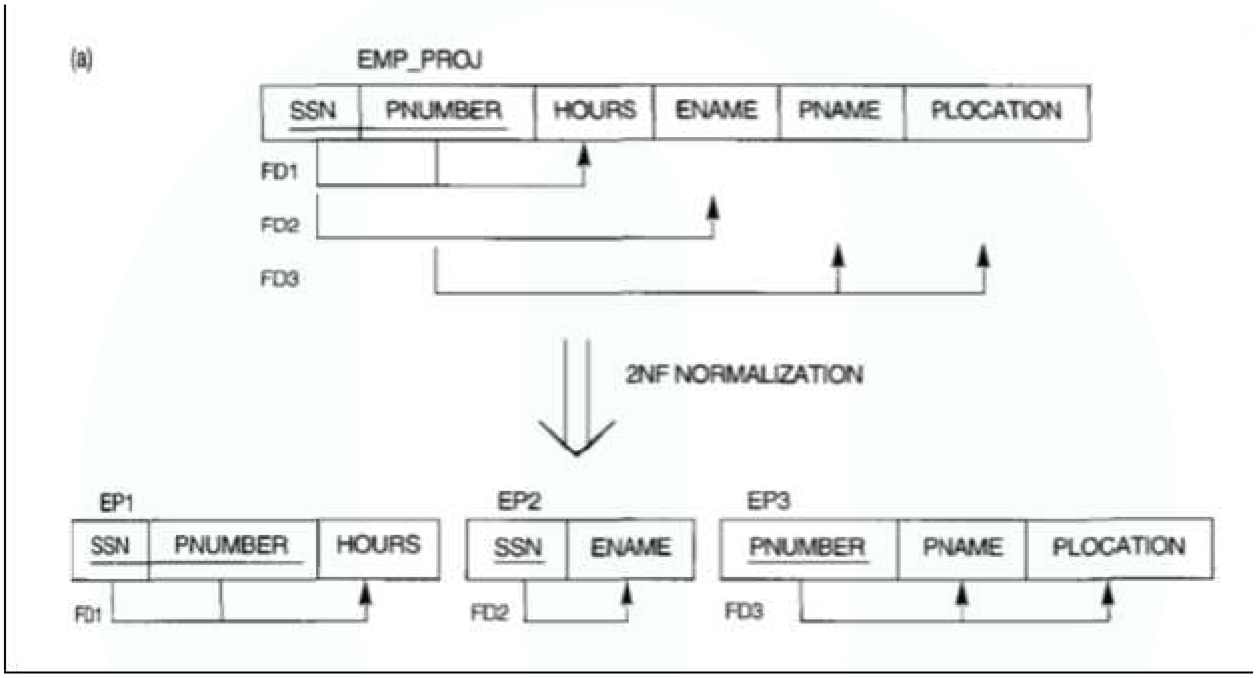


Figure: Normalizing EMP_PROJ into 2NF relations.

3NF

Relation should not have a non key attribute functionally determined by another non key attribute (or by a set of non key attributes.) That is, there should be no transitive dependency of a non key attribute on the primary key.

Formally, 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.

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.

The dependency **SSN→ DMGRSSN is transitive** through DNUMBER in EMP_DEPT of Figure below because both the dependencies SSN → **DNUMBER** and **DNUMBER** → DMGRSSN hold and DNUMBER is neither a key itself nor a subset of the key of EMP_DEPT.

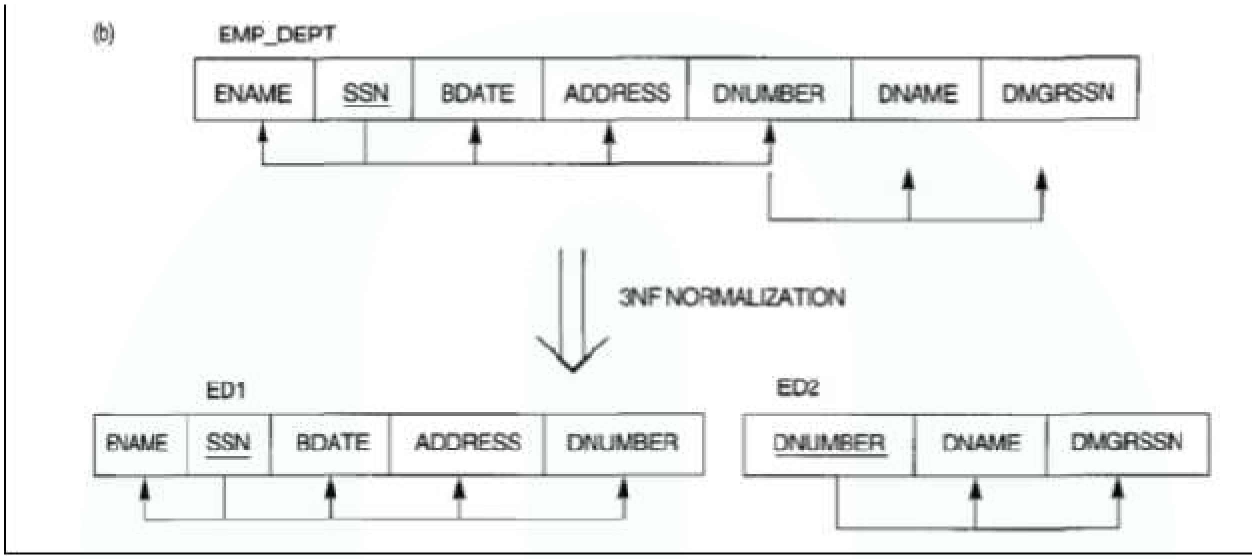


Figure : Normalizing EMP_DEPT into 3NF relations.

The relation schema EMP_DEPT in Figure above **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 Figure 10.lOb.

BCNF

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.

That is, every relation in **BCNF is also in 3NF**; however, a relation in **3NF is not necessarily in BCNF.**

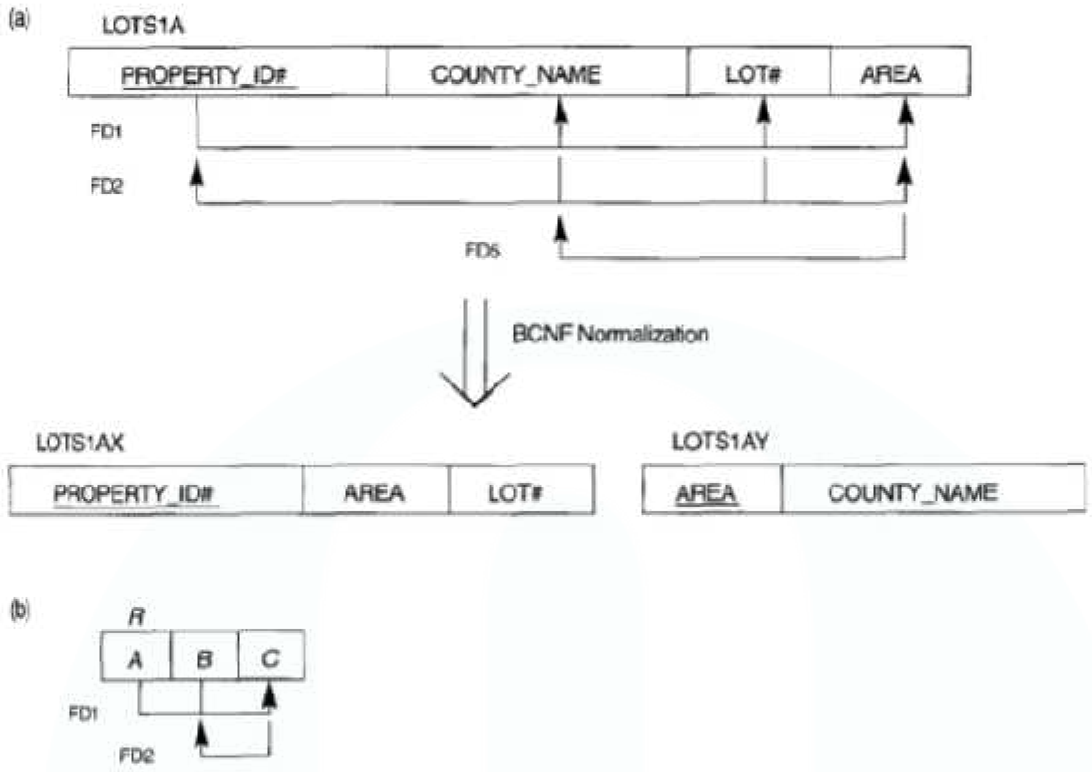


FIGURE 10.12 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

The formal definition of BCNF differs slightly from the definition of 3NF.

The **only difference** between the definitions of BCNF and 3NF is that condition (b) of 3NF, which **allows A to be prime, is absent from BCNF**.

In our example, **FD5 violates BCNF** in LOTS1A because AREA is not a superkey of LOTSlA. Note that **FD5 satisfies 3NF** in LOTS1A because **COUNTY_NAME is a prime attribute** (part of candidate key)(condition b), but this condition does not exist in the definition of BCNF.

We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY, shown in Figure 10.12a.

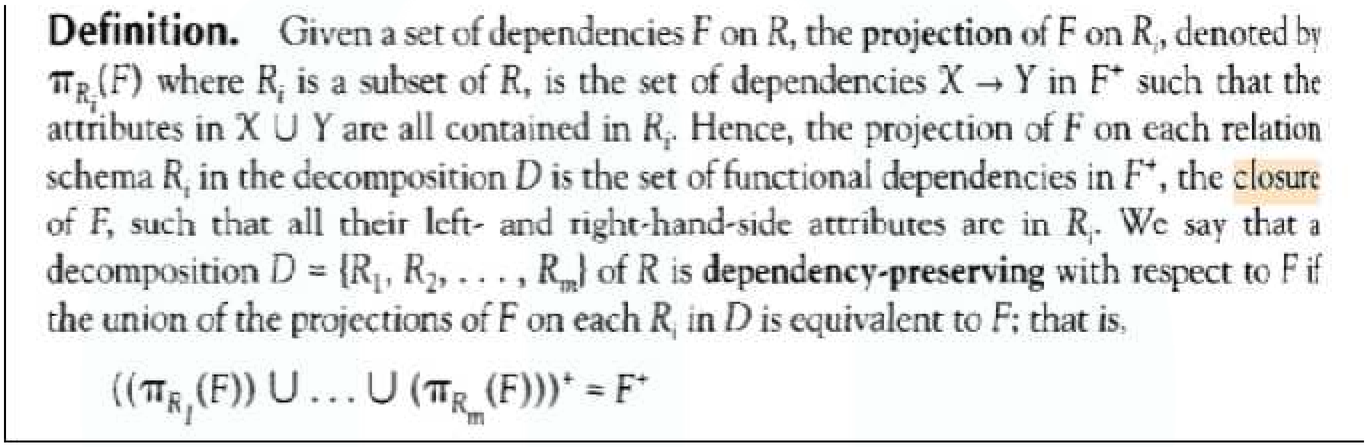
LOSSLESS AND DEPENDENCY PRESERVING DECOMPOSITIONS

Dependency Preservation Property of a Decomposition

The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition

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.

Informally, **this is the dependency preservation condition**.



Definition. Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $\pi_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies X → Y in F^+ such that the attributes in X ∪ Y are all contained in R_i. Hence, the projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left- and right-hand-side attributes are in R_i. We say that a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is **dependency-preserving** with respect to F if the union of the projections of F on each R_i in D is equivalent to F; that is,

$$((\pi_{R_1}(F)) \cup \ldots \cup (\pi_{R_m}(F)))^+ = F^+$$

- 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.

Losslesss

The lossless join or nonadditive join property, which guarantees that the spurious tuple generation problem discussed in Section 10.1.4 does not occur with respect to the relation schemas created after decomposition

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:

$$\boxed{*\,(\pi_{R_1}(r), \ldots, \pi_{R_m}(r)) = r}$$

The word loss in lossless refers to loss of information, not to loss of tuples. If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (∏) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous information.

Testing for Lossless (nonadditive) Join Property

1. Create an initial matrix S with one row i for each relation Ri in D, and one column j for each attribute Aj in R.

2. Set S(i, j):= bij for all matrix entries.

3. For each row i representing relation schema Ri

4. Repeat the following loop until a complete loop execution results in no changes to S.

5. If a row is made up entirely of "a" symbols, then the decomposition has the lossless join property; otherwise, it does not.