Module IV

Informal Design Guidelines for Relational Databases

- Semantics of the Relation Attributes
- Redundant Information in Tuples and Update Anomalies
- Null Values in Tuples
- Spurious Tuples

1.1 Semantics of the Relation Attributes

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- Bottom Line: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

Figure 10.1 A simplified COMPANY relational database schema

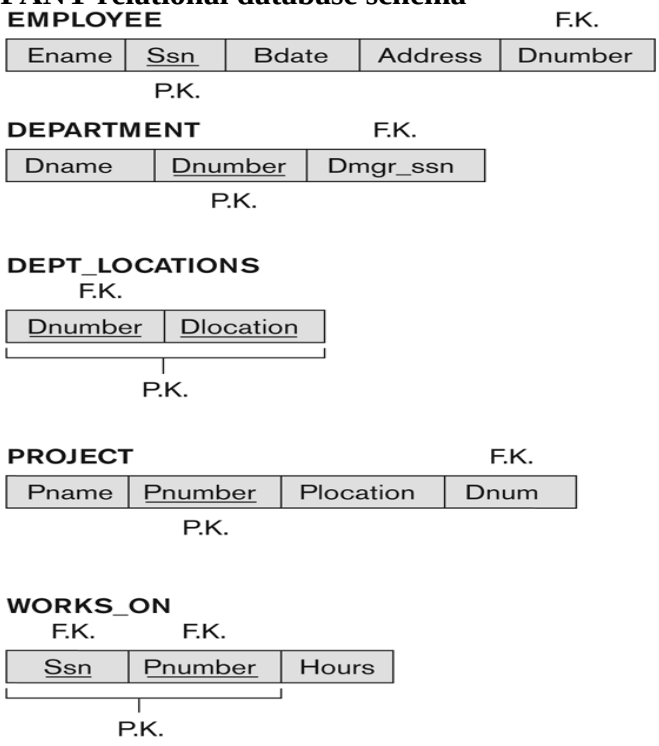


Figure 10.1
A simplified COMPANY relational database schema.

Figure 15.2
Sample database state for the relational database schema in Figure 15.1.

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

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly

- Wastes storage
- Causes problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies
  - Modification anomalies

DIFFERENT ANOMALIES IN DESIGNING A DATABASE

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

Example of an update anomaly:

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.

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 15.4
Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 15.2. These may be stored as base relations for performance reasons.

(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, 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, 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.

Guideline to Redundant Information in Tuples and Update Anomalies:

- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

GUIDELINE 3:

- Relations should be designed such that their tuples will have as few NULL values as possible
- Attributes that are NULL frequently could be placed in separate relations (with the primary key)

- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

1.4 Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

GUIDELINE 4:

- The relations should be designed to satisfy the lossless join condition.
- No spurious tuples should be generated by doing a natural-join of any relations.

- There are two important properties of decompositions:
  - Non-additive or losslessness of the corresponding join
  - Preservation of the functional dependencies.
- Note that:
  - Property (a) is extremely important and *cannot* be sacrificed.

Property (b) is less stringent and may be sacrificed

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

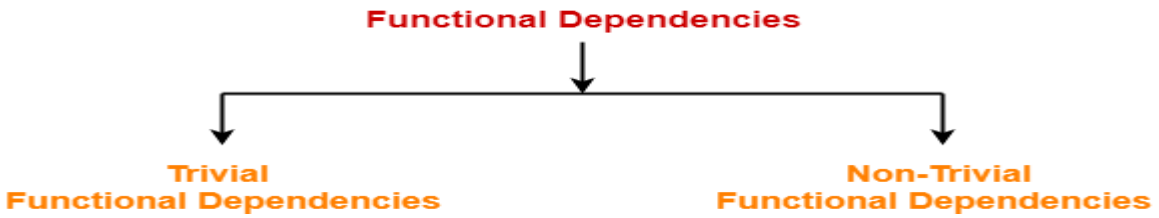
(c) A combination of SSN and PNUMBER values uniquely determines the number of hours the employee currently works on the project per week (HOURS).

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.

There are two types of functional dependencies-



1. Trivial Functional Dependencies
2. Non-trivial Functional Dependencies

1. Trivial Functional Dependencies-

- A functional dependency $X \rightarrow Y$ is said to be trivial if and only if $Y \subseteq X$.
- Thus, if RHS of a functional dependency is a subset of LHS, then it is called as a trivial functional dependency.

Examples-

The examples of trivial functional dependencies are-

- $AB \rightarrow A$
- $AB \rightarrow B$
- $AB \rightarrow AB$

2. Non-Trivial Functional Dependencies-

- A functional dependency $X \rightarrow Y$ is said to be non-trivial if and only if $Y \not\subset X$.
- Thus, if there exists at least one attribute in the RHS of a functional dependency that is not a part of LHS, then it is called as a non-trivial functional dependency.

Armstrong's Axioms

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

- IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y
- IR2. (**Augmentation**) If X -> Y, then XZ -> YZ
  - (Notation: XZ stands for X U Z)
- IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

Some additional inference rules that are useful:

- IR4. **Decomposition**: If X -> YZ, then X -> Y and X -> Z
- IR5. **Union**: If X -> Y and X -> Z, then X -> YZ
- IR6. **Psuedotransitivity**: If X -> Y and WY -> Z, then WX -> 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}

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^+.

Algorithm 16.1. Determining X^+, the Closure of X under F

Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

```
X+ := X;
repeat
    oldX+ := X+;
    for each functional dependency Y → Z in F do
        if X+ ⊇ Y then X+ := X+ ∪ Z;
until (X+ = oldX+);
```

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.

Closure of an Attribute Set-

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.
- Closure of attribute set {X} is denoted as $\{X\}^+$.

Steps to Find Closure of an Attribute Set-

Following steps are followed to find the closure of an attribute set-

Step-01:
Add the attributes contained in the attribute set for which closure is being calculated to the result set
Step-02:
Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example-

Consider a relation R (A , B , C , D , E , F , G) with the functional dependencies-

$$A \rightarrow BC$$

$$BC \rightarrow DE$$

$$D \rightarrow F$$

$$CF \rightarrow G$$

Now, let us find the closure of some attributes and attribute sets-

Closure of attribute A-

$A^+ = \{ A \}$

$= \{ A , B , C \}$ (Using $A \rightarrow BC$)

$= \{ A , B , C , D , E \}$ (Using $BC \rightarrow DE$)

$= \{ A , B , C , D , E , F \}$ (Using $D \rightarrow F$)

$= \{ A , B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)

Thus,

$$A^+ = \{ A , B , C , D , E , F , G \}$$

Closure of attribute D-

$D^+ = \{ D \}$

$= \{ D , F \}$ (Using $D \rightarrow F$)

We can not determine any other attribute using attributes D and F contained in the result set.

Thus,

$$D^+ = \{ D , F \}$$

Closure of attribute set {B, C}-

$\{ B , C \}^+ = \{ B , C \}$

$= \{ B , C , D , E \}$ (Using $BC \rightarrow DE$)

$= \{ B , C , D , E , F \}$ (Using $D \rightarrow F$)

$= \{ B , C , D , E , F , G \}$ (Using $CF \rightarrow G$)

Thus,

$$\{ B , C \}^+ = \{ B , C , D , E , F , G \}$$

Finding the Keys Using Closure-

Super Key-

- If the closure result of an attribute set contains all the attributes of the relation, then that attribute set is called as a super key of that relation.
- Thus, we can say-

"The closure of a super key is the entire relation schema."

Example-

In the above example,

- The closure of attribute A is the entire relation schema.
- Thus, attribute A is a super key for that relation.

Candidate Key-

- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Example-

In the above example,

- No subset of attribute A contains all the attributes of the relation.
- Thus, attribute A is also a candidate key for that relation.

PRACTICE PROBLEM BASED ON FINDING CLOSURE OF AN ATTRIBUTE SET-

Problem-

Consider the given functional dependencies-

$$AB \rightarrow CD$$
$$AF \rightarrow D$$
$$DE \rightarrow F$$
$$C \rightarrow G$$
$$F \rightarrow E$$
$$G \rightarrow A$$

Which of the following options is false?

(A) $\{ CF \}^+ = \{ A , C , D , E , F , G \}$

(B) $\{ BG \}^+ = \{ A , B , C , D , G \}$

(C) $\{ AF \}^+ = \{ A , C , D , E , F , G \}$

(D) $\{ AB \}^+ = \{ A , C , D , F , G \}$

Solution-

Let us check each option one by one-

Option-(A):

$\{ CF \}^+ = \{ C , F \}$

$= \{ C , F , G \}$ (Using $C \rightarrow G$)

$= \{ C , E , F , G \}$ (Using $F \rightarrow E$)

= { A , C , E , E , F } (Using G → A)

= { A , C , D , E , F , G } (Using AF → D)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(B):

$\{ BG \}^+ = \{ B , G \}$

= { A , B , G } (Using G → A)

= { A , B , C , D , G } (Using AB → CD)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option-(C):

$\{ AF \}^+ = \{ A , F \}$

= { A , D , F } (Using AF → D)

= { A , D , E , F } (Using F → E)

Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

Option-(D):

$\{ AB \}^+ = \{ A , B \}$

= { A , B , C , D } (Using AB → CD)

= { A , B , C , D , G } (Using C → G)

Since, our obtained result set is different from the given result set, so,it means it is not correctly given.

Thus,

Option (C) and Option (D) are correct.

EQUIVALENCE OF FDS,

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.

Case-01: F covers G ($F \supseteq G$)

Case-02: G covers F ($G \supseteq F$)

Case-03: Both F and G cover each other ($F = G$)

PRACTICE PROBLEM BASED ON EQUIVALENCE OF FUNCTIONAL DEPENDENCIES-

Problem-

A relation R (A , C , D , E , H) is having two functional dependencies sets F and G as shown-

Set F-

$$A \rightarrow C$$

$$AC \rightarrow D$$

$$E \rightarrow AD$$

$$E \rightarrow H$$

Set G-

$$A \rightarrow CD$$

$$E \rightarrow AH$$

Which of the following holds true?

(A) $G \supseteq F$

(B) $F \supseteq G$

(C) $F = G$

(D) All of the above

Solution-

Determining whether F covers G-

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set G

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow CD$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AH$ using set F

Step-03:

Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G.
- Thus, we conclude F covers G i.e. $F \supseteq G$.

Determining whether G covers F-

Step-01:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set F
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set F
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set F

Step-02:

- $(A)^+ = \{ A , C , D \}$ // closure of left side of $A \rightarrow C$ using set G
- $(AC)^+ = \{ A , C , D \}$ // closure of left side of $AC \rightarrow D$ using set G
- $(E)^+ = \{ A , C , D , E , H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set G

Step-03:

Comparing the results of Step-01 and Step-02, we find-

- Functional dependencies of set G can determine all the attributes which have been determined by the functional dependencies of set F.
- Thus, we conclude G covers F i.e. $G \supseteq F$.

Determining whether both F and G cover each other-

- From Step-01, we conclude F covers G.
- From Step-02, we conclude G covers F.
- Thus, we conclude both F and G cover each other i.e. $F = G$.

Thus, Option (D) is correct.

MINIMAL COVER/CANONICAL COVER

In DBMS,

- A MINIMAL cover is a simplified and reduced version of the given set of functional dependencies.
- Since it is a reduced version, it is also called as **Irreducible set**.

Characteristics-

- Canonical cover is free from all the extraneous functional dependencies.
- The closure of canonical cover is same as that of the given set of functional dependencies.
- Canonical cover is not unique and may be more than one for a given set of functional dependencies.

PRACTICE PROBLEM BASED ON FINDING CANONICAL COVER-

Problem-

The following functional dependencies hold true for the relational scheme R (W , X , Y , Z) –

$$X \rightarrow W$$

$$WZ \rightarrow XY$$

$$Y \rightarrow WXZ$$

Write the irreducible equivalent for this set of functional dependencies.

Solution-

Step-01:

Write all the functional dependencies such that each contains exactly one attribute on its right side-

$$X \rightarrow W$$

$$WZ \rightarrow X$$

$$WZ \rightarrow Y$$

$$Y \rightarrow W$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Step-02:

Check the essentiality of each functional dependency one by one.

For X → W:

- Considering X → W, $(X)^+$ = { X , W }
- Ignoring X → W, $(X)^+$ = { X }

Now,

- Clearly, the two results are different.
- Thus, we conclude that X → W is essential and can not be eliminated.

For WZ → X:

- Considering WZ → X, $(WZ)^+$ = { W , X , Y , Z }
- Ignoring WZ → X, $(WZ)^+$ = { W , X , Y , Z }

Now,

- Clearly, the two results are same.
- Thus, we conclude that WZ → X is non-essential and can be eliminated.

Eliminating WZ → X, our set of functional dependencies reduces to-

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow W$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Now, we will consider this reduced set in further checks.

For WZ → Y:

- Considering WZ → Y, $(WZ)^+$ = { W , X , Y , Z }
- Ignoring WZ → Y, $(WZ)^+$ = { W , Z }

Now,

- Clearly, the two results are different.
- Thus, we conclude that WZ → Y is essential and cannot be eliminated.

For Y → W:

- Considering Y → W, $(Y)^+$ = { W , X , Y , Z }
- Ignoring Y → W, $(Y)^+$ = { W , X , Y , Z }

Now,

- Clearly, the two results are same.
- Thus, we conclude that Y → W is non-essential and can be eliminated.

Eliminating Y → W, our set of functional dependencies reduces to-

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

For Y → X:

- Considering Y → X, $(Y)^+$ = { W , X , Y , Z }
- Ignoring Y → X, $(Y)^+$ = { Y , Z }

Now,

- Clearly, the two results are different.
- Thus, we conclude that Y → X is essential and cannot be eliminated.

For Y → Z:

- Considering Y → Z, $(Y)^+$ = { W , X , Y , Z }
- Ignoring Y → Z, $(Y)^+$ = { W , X , Y }

Now,

- Clearly, the two results are different.
- Thus, we conclude that Y → Z is essential and cannot be eliminated.

From here, our essential functional dependencies are-

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Step-03:

- Consider the functional dependencies having more than one attribute on their left side.
- Check if their left side can be reduced.

In our set,

- Only $WZ \rightarrow Y$ contains more than one attribute on its left side.
- Considering $WZ \rightarrow Y$, $(WZ)^+ = \{ W , X , Y , Z \}$

Now,

- Consider all the possible subsets of WZ.
- Check if the closure result of any subset matches to the closure result of WZ.

$$(W)^+ = \{ W \}$$

$$(Z)^+ = \{ Z \}$$

Clearly,

- None of the subsets have the same closure result same as that of the entire left side.
- Thus, we conclude that we cannot write $WZ \rightarrow Y$ as $W \rightarrow Y$ or $Z \rightarrow Y$.
- Thus, set of functional dependencies obtained in step-02 is the canonical cover.

Finally, the canonical cover is-

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

Canonical Cover

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.

First Normal Form-

A given relation is called in First Normal Form (1NF) if each cell of the table contains only an atomic value.

OR

A given relation is called in First Normal Form (1NF) if the attribute of every tuple is either single valued or a null value.

Example-

The following relation is not in 1NF-

Student_id	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management System
102	Anjali	Automata, Compiler Design

Relation is not in 1NF

However,

- This relation can be brought into 1NF.
- This can be done by rewriting the relation such that each cell of the table contains only one value.

Student_id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Automata
102	Anjali	Compiler Design

Relation is in 1NF

This relation is in First Normal Form (1NF).

NOTE-

- By default, every relation is in 1NF.
- This is because formal definition of a relation states that value of all the attributes must be atomic.

Second Normal Form-

A given relation is called in Second Normal Form (2NF) if and only if-

1. Relation already exists in 1NF.
2. No partial dependency exists in the relation.

Partial Dependency

A partial dependency is a dependency where few attributes of the candidate key determines non-prime attribute(s).

OR

A partial dependency is a dependency where a portion of the candidate key or incomplete candidate key determines non-prime attribute(s).

In other words,

A → B is called a partial dependency if and only if-

1. A is a subset of some candidate key
2. B is a non-prime attribute.

If any one condition fails, then it will not be a partial dependency.

NOTE-

- To avoid partial dependency, incomplete candidate key must not determine any non-prime attribute.
- However, incomplete candidate key can determine prime attributes.

Example-

Consider a relation- R (V , W , X , Y , Z) with functional dependencies-

$$VW \rightarrow XY$$

$$Y \rightarrow V$$

$$WX \rightarrow YZ$$

The possible candidate keys for this relation are-

VW , WX , WY

From here,

- Prime attributes = { V , W , X , Y }
- Non-prime attributes = { Z }

Now, if we observe the given dependencies-

- There is no partial dependency.
- This is because there exists no dependency where incomplete candidate key determines any non-prime attribute.

Thus, we conclude that the given relation is in 2NF.

Third Normal Form-

A given relation is called in Third Normal Form (3NF) if and only if-

1. Relation already exists in 2NF.
2. No transitive dependency exists for non-prime attributes.

Transitive Dependency

A → B is called a transitive dependency if and only if-

1. A is not a super key.
2. B is a non-prime attribute.

If any one condition fails, then it is not a transitive dependency.

NOTE-

- Transitive dependency must not exist for non-prime attributes.
- However, transitive dependency can exist for prime attributes.

OR

A relation is called in Third Normal Form (3NF) if and only if-

Any one condition holds for each non-trivial functional dependency A → B

1. A is a super key

2. B is a prime attribute

Example-

Consider a relation- R (A , B , C , D , E) with functional dependencies-

$$A \rightarrow BC$$

$$CD \rightarrow E$$

$$B \rightarrow D$$

$$E \rightarrow A$$

The possible candidate keys for this relation are-

$$A , E , CD , BC$$

From here,

- Prime attributes = { A , B , C , D , E }
- There are no non-prime attributes

Now,

- It is clear that there are no non-prime attributes in the relation.
- In other words, all the attributes of relation are prime attributes.
- Thus, all the attributes on RHS of each functional dependency are prime attributes.

Thus, we conclude that the given relation is in 3NF.

Boyce-Codd Normal Form-

A given relation is called in BCNF if and only if-

1. Relation already exists in 3NF.
2. For each non-trivial functional dependency $A \rightarrow B$, A is a super key of the relation.

Example-

Consider a relation- R (A , B , C) with the functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow C$$

$$C \rightarrow A$$

The possible candidate keys for this relation are-

$$A , B , C$$

Now, we can observe that RHS of each given functional dependency is a candidate key.

Thus, we conclude that the given relation is in BCNF.

1NF

Relation should have no non atomic attributes or nested relations.

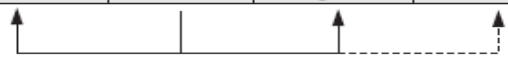
It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.

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 .

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations



(b)

DEPARTMENT

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

(c)

DEPARTMENT

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

Figure 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

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.
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

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

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

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 15.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

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.

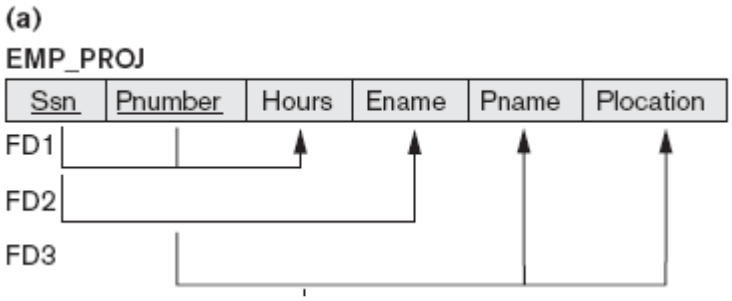


Fig 10.3b

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 below each of which is in 2NF.

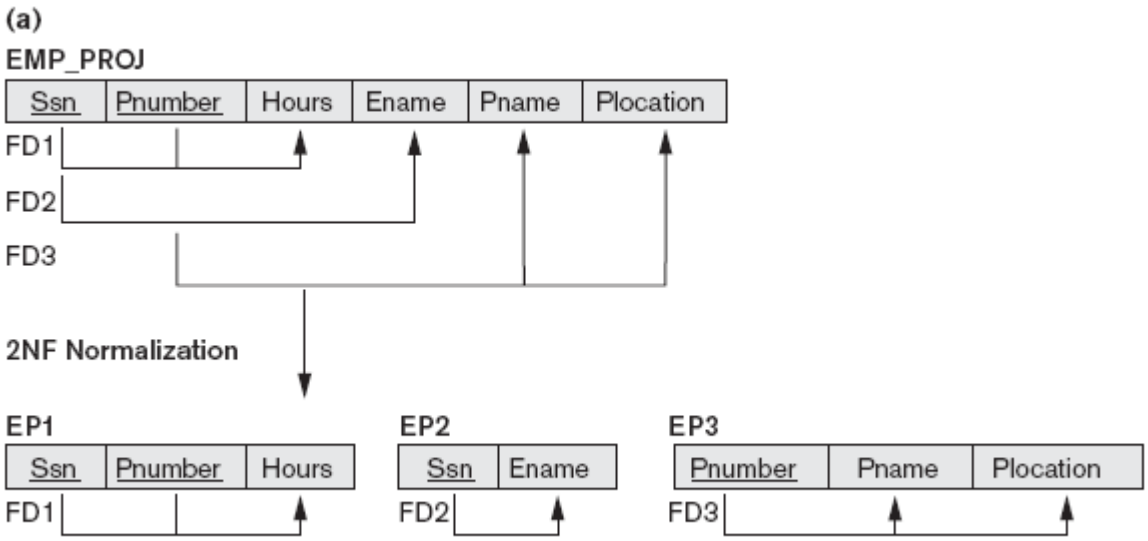


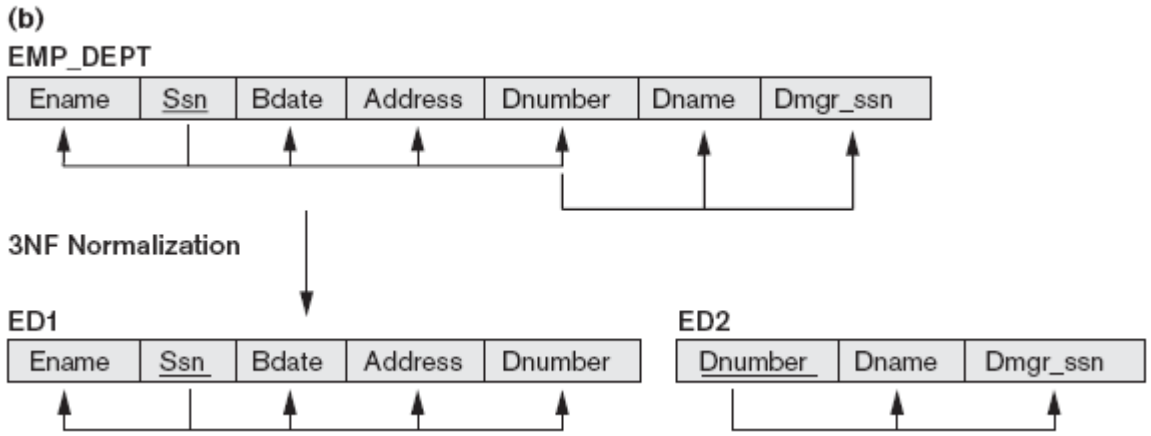
Figure: Normalizing EMP_PROJ into 2NF relations.

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.

Another Example:

Suppose that there are two candidate keys: Property_id# and {County_name, Lot#}; that is, lot numbers are unique only within each county, but Property_id# numbers are unique across counties for the entire state.

Based on the two candidate keys Property_id# and {County_name, Lot#}, the functional dependencies FD1 and FD2 in below Figure hold. We choose Property_id# as the primary key, so it is underlined in Figure but no special consideration will be given to this key over the other candidate key. Suppose that the following twoadditional functional dependencies hold in LOTS:

FD3: County_name →Tax_rate

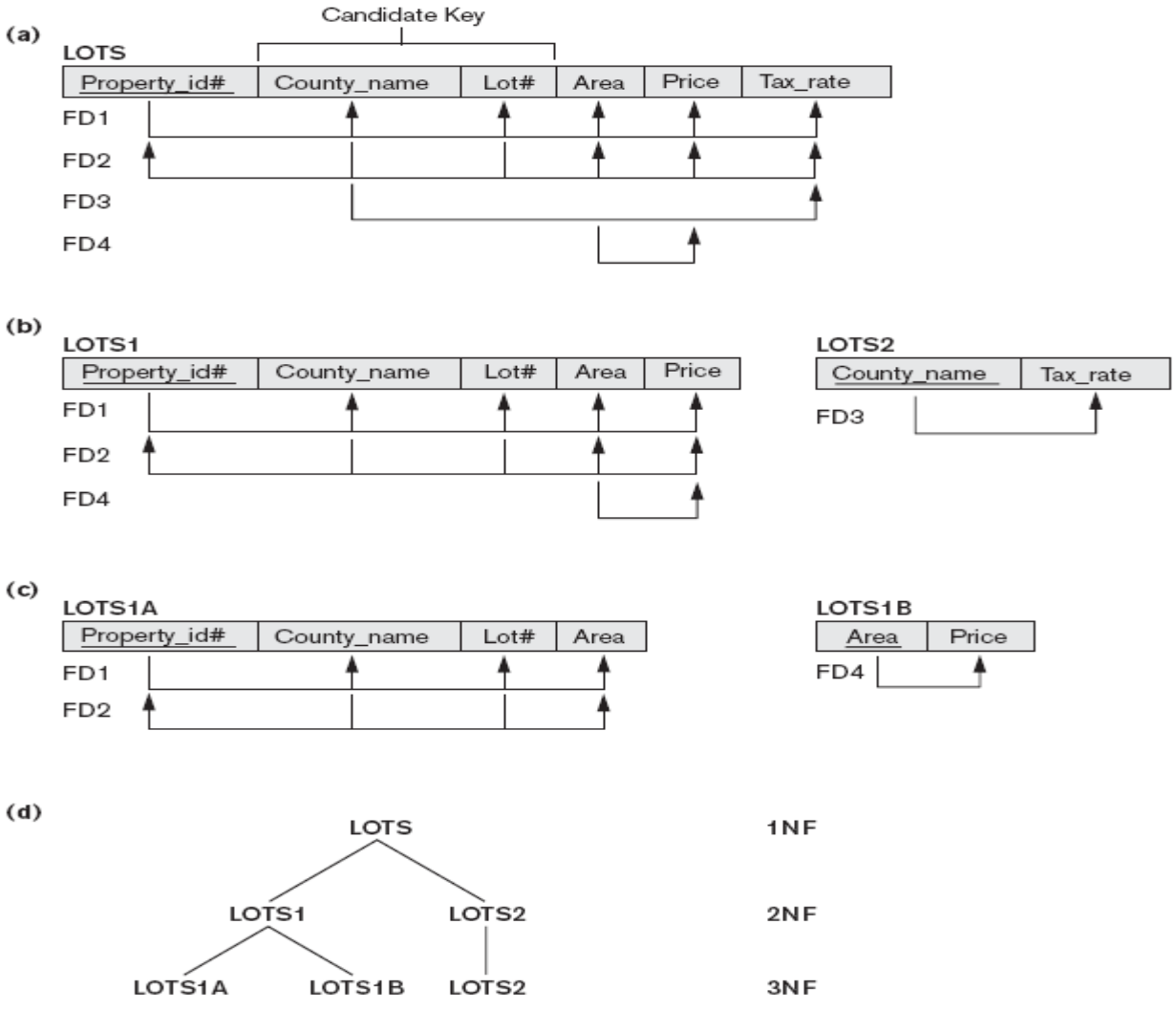
FD4: Area →Price

In words, the dependency FD3 says that the tax rate is fixed for a given county (does not vary lot by lot within the same county), while FD4 says that the price of a lot is determined by its area regardless of which county it is in. (Assume that this is the price of the lot for tax purposes.)

The LOTS relation schema violates the general definition of 2NF because Tax_rate is partially dependent on the candidate key {County_name, Lot#}, due to FD3. To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2, shown in Figure b. We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2. Both LOTS1 and LOTS2 are in 2NF. Notice that FD4 does not violate 2NF and is carried over to LOTS1.

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

- LOTS1 violates 3NF because Price is transitively dependent on each of the candidate keys of LOTS1 via the nonprime attribute Area.
- This general definition can be applied *directly* to test whether a relation schema is in 3NF; it does *not* have to go through 2NF first. If we apply the above 3NF definition to LOTS with the dependencies FD1 through FD4, we find that *both* FD3 and FD4 violate 3NF. Therefore, we could decompose LOTS into LOTS1A, LOTS1B, and LOTS2 directly. Hence, the transitive and partial dependencies that violate 3NF can be removed *in any order*.



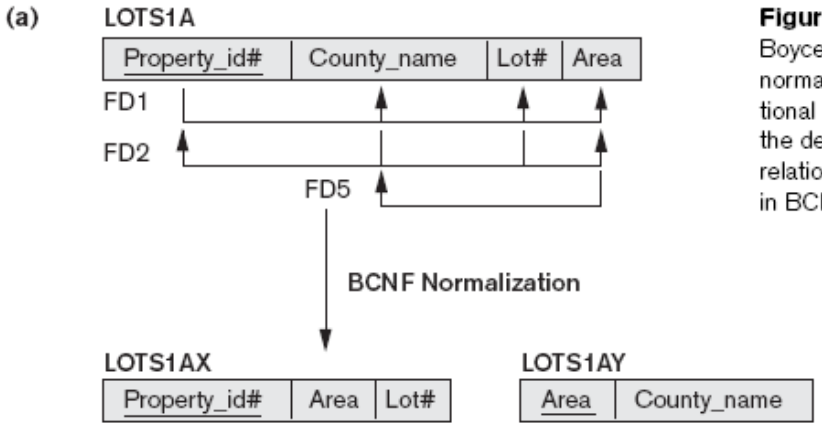
BCNF

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 below.



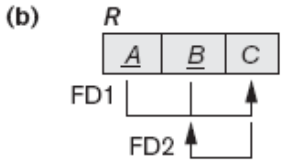


Figure 15.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

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 $\pi_{Ri}(F)$ where R_i 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 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 $D = \{R_1, R_2, ..., R_m\}$ 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, $((\pi_{R_1}(F)) \cup ... \cup (\pi_{R_m}(F)))^+ = F^+$.

- 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.

Example:

Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.
A relation R is decomposed into -

R_1 = (A, B, C) with FDs F_1 = {A -> B, A -> C}, and

R_2 = (C, D) with FDs F_2 = {C -> D}.

F' = $F_1 \cup F_2$ = {A -> B, A -> C, C -> D}

so, F' = F.

And so, $F'^+ = F^+$.

Thus, the decomposition is dependency preserving decomposition.

Or

Assume R(A, B, C, D) with FDs A→B, B→C, C→D.

Let us decompose R into R1 and R2 as follows;
R1(A, B, C)
R2(C, D)
The FDs A→B, and B→C are hold in R1.
The FD C→D holds in R2.
All the functional dependencies hold here. Hence, this decomposition is dependency preserving.

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:

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

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.

Determining Whether Decomposition Is Lossless Or Lossy-

Consider a relation R is decomposed into two sub relations R_1 and R_2.

Then,

- If all the following conditions satisfy, then the decomposition is lossless.

- If any of these conditions fail, then the decomposition is lossy.

Condition-01:

Union of both the sub relations must contain all the attributes that are present in the original relation R.

Thus, $\mathbf{R_1 \cup R_2 = R}$

Condition-02:

- Intersection of both the sub relations must not be null.
- In other words, there must be some common attribute which is present in both the sub relations.

Thus, $\mathbf{R1 \cap R2 \neq \emptyset}$

Condition-03:

- Intersection of both the sub relations must be a super key of either R_1 or R_2 or both.

Thus, $\mathbf{R_1 \cap R_2}$ **= Super key of** $\mathbf{R_1}$ **or** $\mathbf{R_2}$

PRACTICE PROBLEMS BASED ON DETERMINING WHETHER DECOMPOSITION IS LOSSLESS OR LOSSY-

Problem-01:

Consider a relation schema R (A , B , C , D) with the functional dependencies A → B and C → D. Determine whether the decomposition of R into R_1 (A , B) and R_2 (C , D) is lossless or lossy.

Solution-

To determine whether the decomposition is lossless or lossy,

- We will check all the conditions one by one.
- If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Condition-01:

According to condition-01, union of both the sub relations must contain all the attributes of relation R.

So, we have-

$$R_1 (A , B) \cup R_2 (C , D)$$

$$= R (A , B , C , D)$$

Clearly, union of the sub relations contain all the attributes of relation R.

Thus, condition-01 satisfies.

Condition-02:

According to condition-02, intersection of both the sub relations must not be null.

So, we have-

$R_1 (A , B) \cap R_2 (C , D) = \Phi$

Clearly, intersection of the sub relations is null.

So, condition-02 fails.

Thus, we conclude that the decomposition is lossy.

To determine whether the decomposition is lossless or lossy,

- We will check all the conditions one by one.
- If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Figure 16.1
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)

Lossless Join Example discussed in class

Let R = ABCDE, R1 = AD, R2 = AB, R3 = BE, R4 = CDE, and R5 = AE. Let the functional dependencies be: A -> C, B -> C, C -> D, DE -> C, CE -> A
Apply algorithm 7.2 from class handout to test if the decomposition of R into {R1,..,R5} is a lossless join decomposition.

The initial table looks as follows:

	A	B	C	D	E
R1(AD)	**a1**	b12	b13	**a4**	b15
R2(AB)	**a1**	**a2**	b23	b24	b25
R3(BE)	b31	**a2**	b33	b34	**a5**
R4(CDE)	b41	b42	**a3**	**a4**	**a5**
R5(AE)	**a1**	b52	b53	b54	**a5**

Apply FD A -> C to the initial table to modify the violating dependencies. Rows 1, 2, 5 will need to change the values for the RHS attribute C – equate b13, b23, b53 to b13 (you might very well have picked b23 or b53 to equate all three). We won't change the rows 3 & 4 yet because their symbols b31, b41 are different from a1.

A	B	C	D	E
a1	b12	~~b13~~ **b13**	a4	b15
a1	a2	~~b23~~ **b13**	b24	b25
b31	a2	b33	b34	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ **b13**	b54	a5

Apply B -> C next to equate b33 with b13

A	B	C	D	E
a1	b12	~~b13~~ b13	a4	b15
a1	**a2**	~~b23~~ **b13**	b24	b25
b31	**a2**	~~b33~~ **b13**	b34	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ b13	b54	a5

Next, use C -> D to equate a4, b24, b34, and b54

A	B	C	D	E
a1	b12	~~b13~~ **b13**	**a4**	b15
a1	a2	~~b23~~ **b13**	~~b24~~ **a4**	b25
b31	a2	~~b33~~ **b13**	~~b34~~ **a4**	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ **b13**	~~b54~~ **a4**	a5

DE -> C helps us to equate b13 (all occurrences) with a3. The following table shows the corresponding changes.

A	B	C	D	E
a1	b12	~~b13~~ ~~b13~~ **a3**	a4	b15
a1	a2	~~b23~~ ~~b13~~ **a3**	~~b24~~ a4	b25
b31	a2	~~b33~~ ~~b13~~ **a3**	~~b34~~ **a4**	**a5**
b41	b42	**a3**	**a4**	**a5**
a1	b52	~~b53~~ ~~b13~~ **a3**	~~b54~~ **a4**	**a5**

Apply CE -> A to the table above to equate b31, b41, and a1.

A	B	C	D	E
a1	b12	~~b13~~ ~~b13~~ a3	a4	b15
a1	a2	~~b23~~ ~~b13~~ a3	~~b24~~ a4	b25
~~b31~~ **a1**	**a2**	~~b33~~ ~~b13~~ **a3**	~~b34~~ **a4**	**a5**
~~b41~~ **a1**	b42	**a3**	a4	**a5**
a1	b52	~~b53~~ ~~b13~~ **a3**	~~b54~~ a4	**a5**

The middle row in the table above is all a's, and the decomposition has a lossless join.