UNIT-3
SCHEMA REFINEMENT AND NORMALISATION

Unit 3 contents at a glance:

1. Introduction to schema refinement,
2. functional dependencies,
3. reasoning about FDs.
4. Normal forms: 1NF, 2NF, 3NF, BCNF,
5. properties of decompositions,
6. normalization,
7. schema refinement in database design(Refer Text Book),
8. other kinds of dependencies: 4NF, 5NF, DKNF
9. Case Studies(Refer text book)

.

1. Schema Refinement:

The Schema Refinement refers to refine the schema by using some technique. The best technique of schema refinement is **decomposition.**

Normalisation or Schema Refinement is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies: Anomalies refers to the problems occurred after poorly planned and normalised databases where all the data is stored in one table which is sometimes called a flat file database.

Anomalies or problems facing without normalization(problems due to redundancy) :

Anomalies refers to the problems occurred after poorly planned and unnormalised databases where all the data is stored in one table which is sometimes called a flat file database. Let us consider such type of schema –

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k
Primary Key(SID,CID)				

Here all the data is stored in a single table which causes redundancy of data or say anomalies as SID and Sname are repeated once for same CID . Let us discuss anomalies one by one.

Due to redundancy of data we may get the following problems, those are-

1.insertion anomalies : It may not be possible to store some information unless some other information is stored as well.

2.redundant storage: some information is stored repeatedly

3.update anomalies: If one copy of redundant data is updated, then inconsistency is created unless all redundant copies of data are updated.

4.deletion anomalies: It may not be possible to delete some information without losing some other information as well.

Problem in updation / updation anomaly – If there is updation in the fee from 5000 to 7000, then we have to update FEE column in all the rows, else data will become inconsistent.

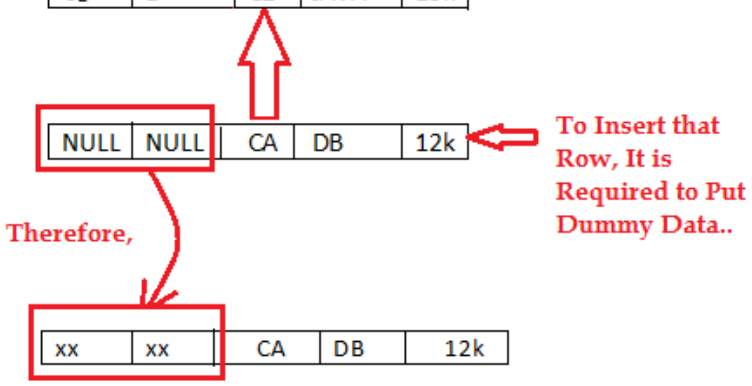
SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~
S2	A	C1	C	~~5k~~
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

7k
7k
> Costly Operation
⇩
More IO Cost

Insertion Anomaly and Deletion Anomaly- These anomalies exist only due to redundancy, otherwise they do not exist.

InsertionAnomalies: New course is introduced C4, But no student is there who is having C4 subject.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k



Because of insertion of some data, It is forced to insert some other dummy data.

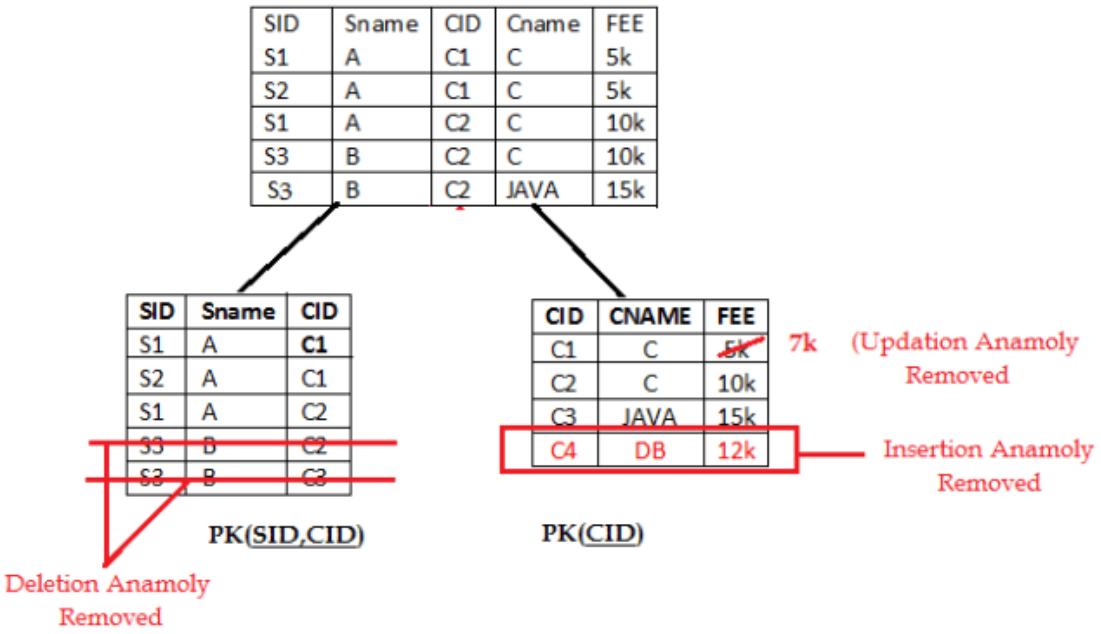
Deletion Anomaly :

Deletion of S3 student cause the deletion of course.

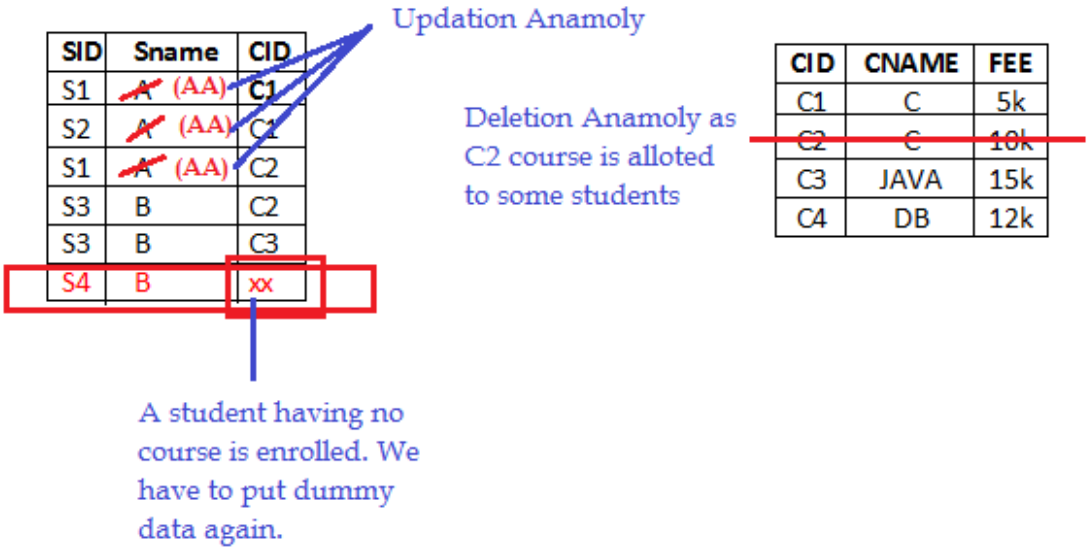
Because of deletion of some data forced to delete some other useful data.

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

Solutions To Anomalies : Decomposition of Tables – Schema Refinement



There are some Anomalies in this again –



SID	Sname	CID
S1	~~A~~ (AA)	C1
S2	~~A~~ (AA)	C1
S1	~~A~~ (AA)	C2
S3	B	C2
S3	B	C3
S4	B	xx

CID	CNAME	FEE
C1	C	5k
~~C2~~	~~C~~	~~10k~~
C3	JAVA	15k
C4	DB	12k

What is the Solution ??
Solution : decomposing into relations as shown below

R1

SID	Sname

R2

SID	CID

R3

CID	Cname	Fee

→**TO AVOID REDUNDANCY** and problems due to redundancy, we use refinement technique called **DECOMPOSITION.**

Decomposition:- Process of decomposing a larger relation into smaller relations.

→Each of smaller relations contain subset of attributes of original relation.

Functional dependencies:

→Functional dependency is a relationship that exist when one attribute uniquely determines another attribute.

→Functional dependency is a form of integrity constraint that can identify schema with redundant storage problems and to suggest refinement.

→A functional dependency A→B in a relation holds true if two tuples having the same value of attribute A also have the same value of attribute B

IF t1.X=t2.X then t1.Y=t2.Y where t1,t2 are tuples and X,Y are attributes.

Reasoning about functional dependencies:

Armstrong Axioms :

Armstrong axioms defines the set of rules for reasoning about functional dependencies and also to infer all the functional dependencies on a relational database.

Various axioms rules or inference rules:

Primary axioms:

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

secondary or derived axioms:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Attribute closure: Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.

NOTE:

To find attribute closure of an attribute set-

1)add elements of attribute set to the result set.

2)recursively add elements to the result set which can be functionally determined from the elements of result set.

Algorithm : Determining X^+, the closure of X under F.

```
Input : Let F be a set of FDs for relation R.

Steps:
            1. X+ = X                              //initialize X+ to X
            2. For each FD : Y -> Z in F Do
               If Y ⊆ X+ Then                      //If Y is contained in X+
               X+ =  X+ ∪ Z                        //add Z to X+
               End If
               End For
            3. Return  X+                        //Return closure of X

Output : Closure  X+ of X under F
```

Types of functional dependencies:

1**)Trivial functional dependency:**-If X→Y is a functional dependency where Y subset X, these type of FD's called as trivial functional dependency.

2)Non-trivial functional dependency:-If X→Y and Y is not subset of X then it is called non-trivial functional dependency.

3)Completely non-trivial functional dependency:-If X→Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency.

Prime and non-prime attributes

Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

Candidate Key:
Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
Consider student table: student(sno, sname,sphone,age)
we can take **sno** as candidate key. we can have more than 1 candidate key in a table.
types of candidate keys:
1. simple(having only one attribute)
2. composite(having multiple attributes as candidate key)

Super Key:
Super Key is set of attributes of a relation which can be used to identify a tuple uniquely.

- Adding zero or more attributes to candidate key generates super key.
- A candidate key is a super key but vice versa is not true.

Consider student table: student(sno, sname,sphone,age)
we can take sno, (sno, sname) as super key

Finding candidate keys problems:

Example 1: Find candidate keys for the relation R(ABCD) having following FD's

AB→CD, C→A, D→A

Solution:

AB^+ = {ABCD} A and B are prime attributes

C→A replace A by c

BC^+ = {ABCD} A and C are prime attributes ($A^+ = A^+ = \{AC\}$)

D→B replace B by D

AD^+ = {ABCD} A and D are prime attributes ($D^+ = \{BD\}$)

CD^+ = {ABCD} (replacing A by C in AD)

AB, BC, CD, AD are candidate keys.

Example 2: Find candidate keys for R(ABCDE) having following FD's

A→BC,CD→E,B→D, E→A

Solution:

A^+ = {ABCDE} A is candidate key and prime attribute

E→A so replace A by E

E^+ = {ABCDE} E is candidate key and prime attribute

CD→E replace E by CD

CD^+ = {ABCDE} (C^+ = C and D^+ = D) no proper subset of CD is superkey. so CD is candidate key

B→D

BC^+ = {ABCDE} (B^+ = BD) BC is candidate key

A, E, CD, BC are candidate keys

Question 1 : Given a relation R(ABCDEF) having FDs {AB→C, C→D, D→E , F→B, E→F} Identify the prime attributes and non prime attributes .

Solution :

$(AB)^+$: {ABCDEF} ⇒ Super Key
$(A)^+$: {A} ⇒ Not Super Key
$(B)^+$: {B} ⇒ Not Super Key
Prime Attributes : {A,B}
(AB) → Candidate Key
↓ (as F → B)
$(AF)^+$: {AFBCDE}
$(A)^+$: {A} ⇒ Not Super key
$(F)^+$: {FB} ⇒ Not Super Key
(AF) → Candidate Key
↓
$(AE)^+$: {AEFBCD}
$(A)^+$: {A} ⇒ Not Super key
$(E)^+$: {EFB} ⇒ Not Super key
(AE) → Candidate Key
↓
$(AD)^+$: {ADEFBC}
$(A)^+$: {A} ⇒ Not Super key
$(D)^+$: {DEFB} ⇒ Not Super key
(AD) → Candidate Key
↓
$(AC)^+$: {ACDEFB}
$(A)^+$: {A} ⇒ Not Super Key
$(C)^+$: {DCEFB} ⇒ Not Super Key
⇒ Candidate Keys {AB, AF, AE, AD, AC}
⇒ Prime Attributes {A,B,C,D,E,F}
⇒ Non Prime Attributes {}

Question 2: Given a relation R(ABCDEF) having FDs {AB → C, C → DE , E → F, C → B} Identify the prime attributes and non prime attributes.

Solution :

$(AB)^+$: {A B C D E F}
$(A)^+$: {A}
$(B)^+$: {B}
(AB) ⇒ (AC), $(AC)^+$: {ABCDEF}
$(C)^+$: {DECBF}
⇒ Candidate Keys {AB, AC}
⇒ Prime Attributes {A,B,C}
⇒ Non Prime Attributes {D,E,F}

Normalization:

Normalization is a process of designing a consistent database with minimum redundancy which support data integrity by grating or decomposing given relation into smaller relations preserving constraints on the relation.

→Normalisation removes data redundancy and it will helps in designing a good data base which involves a set of normal forms as follows -

1)First normal form(1NF)

2)Second normal form(2NF)

3)Third normal form(3NF)

4)Boyce coded normal form(BCNF)

5)Forth normal form(4NF)

6)Fifth normal form(5NF)

7)Sixth normal form(6NF)

8)Domain key normal form.

Normal Forms

1st Normal Form	No repeating data groups
2nd Normal Form	No partial key dependency
3rd Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4th Normal Form	No multi-valued dependency
5th Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

Property	3NF	BCNF	4NF
Eliminates redundancy due to FD's	Most	Yes	Yes
Eliminates redundancy due to MVD's	No	No	Yes
Preserves FD's	Yes	Maybe	Maybe
Preserves MVD's	Maybe	Maybe	Maybe

Properties of normal forms and their decompositions

1)First normal form: A relation is said to be in first normal form if it contains all atomic values or single values.

Example:

Domain	**Courses**
Programming	C , java
Web designing	HTML , PHP

The above table consist of multiple values in single columns which can be reduced into atomic values by using first normal form as follows-

Domain	**Courses**
Programming	C
Programming	Java
Web designing	HTML
Web designing	PHP

2)Second normal form: A relation is said to be in second normal form if it is in first normal form without any partial dependencies.

→In second normal form non-prime attributes should not depend on proper subset of key attributes.

Example:

Student id	**Student name**	**Project Id**	**Project name**

Here (student id, project id) are key attributes and (student name, project name) are non-prime attributes. It is decomposed as-

Student id	Student name	Project id

Project id	Project name

3)Third normal form: A relation is said to be in third normal form , if it is already in second normal form and no transitive dependencies exists.

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Student id	Student name	City	country	ZIP

It is decomposed as:

Student id	Student name	ZIP

ZIP	city	country

4)Boyce normal form: It is an extension of third normal form where in a functional dependency X→ A , X must be a super key.

A relation is in BCNF if in every non-trivial functional dependency X –> Y, X is a super key.

5)fourth normal form: A relation is said to be in fourth normal form if it is in third normal form and no multi value dependencies should exist between attributes.

Note: In some cases multi value dependencies may exist not more than one time in a given relation.

6)fifth normal form: fifth normal form is related to join dependencies.

→A relation R is said to be in fifth normal form if for every join dependency JD join {R_1 , R_2 ,…….R_N } that holds over relation R one of the following statements must be true-

1)R_i =R for some i

2)the join dependency is implied by the set of those functional dependency over relation R in which the left side is key attribute for R.

NOTE: if the relation schema is a third normal form and each of its keys consist of single attribute, we can say that it can also be in fifth normal form.

→A join dependency JD join {R_1, R_2, ……R_N} is said to hold for a relation R if R_1,R_2…..R_N this decomposition is a loss less join decomposition of R.

→When a relation is in forth normal form and decompose further to eliminate redundancy and anomalies due to insert or update or delete operation, there should not be any loss of data or should not create a new record when the decompose tables are rejoin.

7)Domain key normal form: A domain key normal form keeps a constraint that every constraint on the relation is a logical sequence of definition of keys and domains.

8)Sixth normal form: A relation is said to be in sixth normal form such that the relation R should not contain any non-trivial join dependencies.

→ Also sixth normal form considers temporal dimensions(time) to the relational model.

Key Points related to normal forms –

1. BCNF is free from redundancy.
2. If a relation is in BCNF, then 3NF is also also satisfied.
3. If all attributes of relation are prime attribute, then the relation is always in 3NF.
4. A relation in a Relational Database is always and at least in 1NF form.
5. Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.
6. If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).
7. Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.
8. There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it’s generally not required to go beyond BCNF.

problems on normal forms:

Problem 1:

Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

ABC --> D

CD --> AE

Solution:

Important Points for solving above type of question.

1) It is always a good idea to start checking from BCNF, then 3 NF and so on.

2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC –> D is in BCNF (Note that ABC is a super key), so no need to check this dependency for lower normal forms.

Candidate keys in given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD which is a proper subset of a candidate key and it determine E, which is non prime attribute. So, given relation is also not in 2 NF. S**o, the highest normal form is 1 NF.**

problem 2:

Find the highest normal form of a relation R(A,B,C,D,E) with

FD set as

{BC->D,
AC->BE,
B->E}

Step 1: As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2: Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

Step 3: The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

Decomposition: It is the process of splitting original table into smaller relations such that attribute sets of two relations will be the subset of attribute set of original table.

Rules of decomposition:

If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1)Union of two smaller subsets of attributes gives all attributes of 'R'.

R1(attributes)UR2(attributes)=R(attributes)

2)Both relations interaction should not give null value.

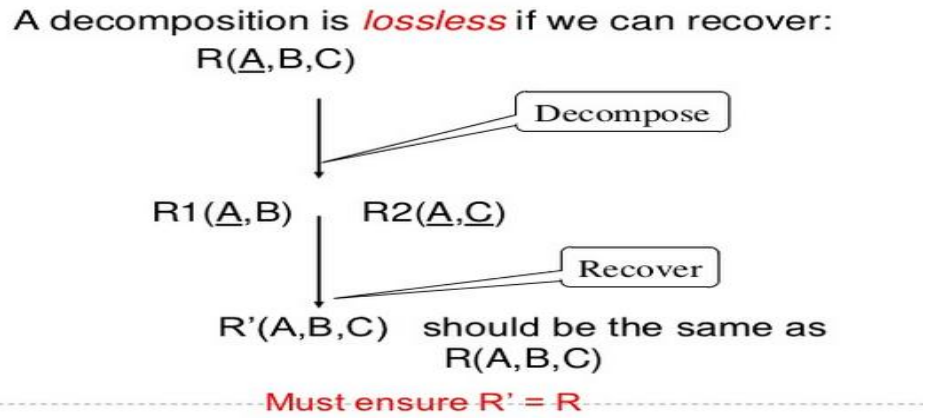
R1(attributes)∩R2(attributes)!=null

3) Both relations interaction should give key attribute.

R1(attribute)∩R2(attribute)=R(key attribute)

Properties of decomposition:

Lossless decomposition: while joining two smaller tables no data should be lost and should satisfy all the rules of decomposition. No additional data should be generated on natural join of decomposed tables.



Lossless Decomposition example

- Sometimes the same set of data is reproduced:

Name	Price	Category
Word	100	WP
Oracle	1000	DB
Access	100	DB

Name	Price
Word	100
Oracle	1000
Access	100

Name	Category
Word	WP
Oracle	DB
Access	DB

- (Word, 100) + (Word, WP) → (Word, 100, WP)
- (Oracle, 1000) + (Oracle, DB) → (Oracle, 1000, DB)
- (Access, 100) + (Access, DB) → (Access, 100, DB)

example 2 for loseless decomposition:

Lossless Decomposition (example)

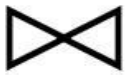
A	B	C
1	2	3
4	5	6
7	2	8

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	C
1	3
4	6
7	8



B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Lossy join decomposition: if information is lost after joining and if do not satisfy any one of the above rules of decomposition.

example 1:

Lossy Decomposition (example)

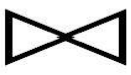
A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

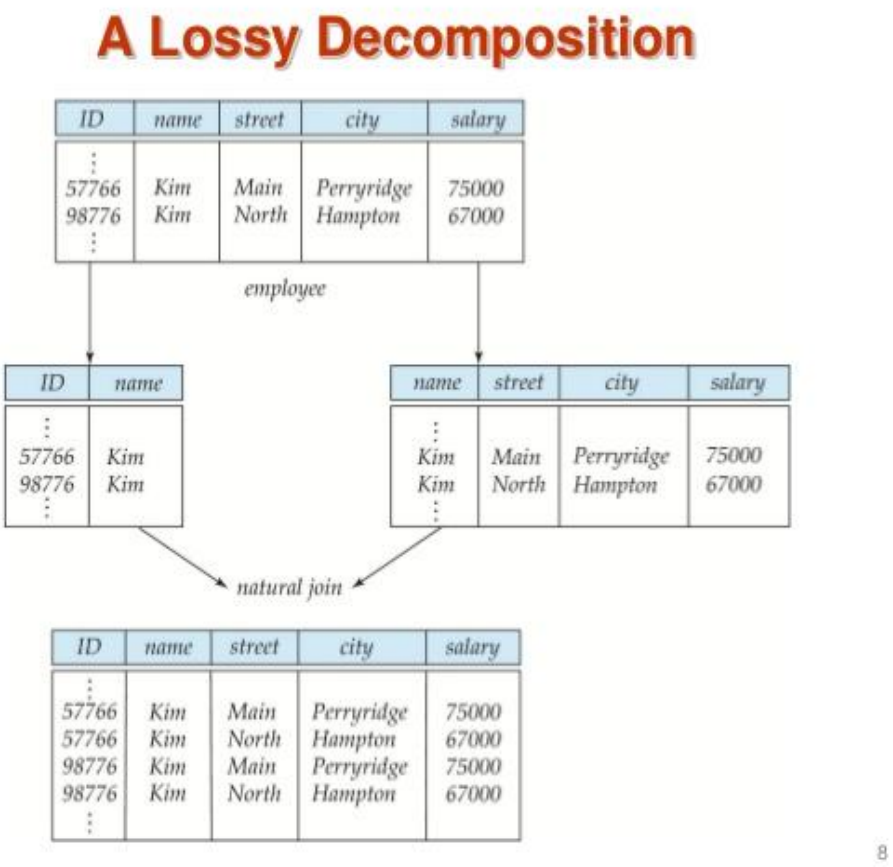
A	B
1	2
4	5
7	2



B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

example 2:



In above examples, on joining decomposed tables, extra tuples are generated.

so it is lossy join decomposition.

Dependency preservation: functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables.

Dependency Preservation

A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving wrt a set F of Functional dependency if

(F1 U F2 U … U Fm)+ = F+.
Consider a relation R
R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.
f1 U f2 is a subset of F -----> Not Dependency preserving.
f1 U f2 is a super set of F -----> This case is not possible.

example for dependency preservation:

Dependency preservation

Example:
R=(A, B, C), F={A→B, B→C}

Decomposition of R: R1=(A, B) R2=(B, C)
Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold: F1={A→B, A→A, B→B, AB→AB}
In R2 the following dependencies hold: F2= {B→B, C→C, B→C, BC→BC}

F'= F1' ∪ F2' = {A→B, B→C, trivial dependencies}

In F' all the original dependencies occur, so this decomposition preserves dependencies.

lack of redundancy: It is also known as repetition of information. The proper decomposition should not suffer from any data redundancy.