UNIT-III SCHEMA REFINEMENT & NORMALIZATION

Types Of Anomalies, Concept Of Functional Dependency, Normalization, Advantages ,Types Of Normal forms(1NF, 2NF And 3NF), Boyce-Codd Normal Form(BCNF), Fourth Normal Form(4NF) .Lossless Join And Dependency Preserving Decomposition.

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.

Problems Caused by 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.

Storing the same information redundantly, that is, in more than one place within a database, can lead to several problems:

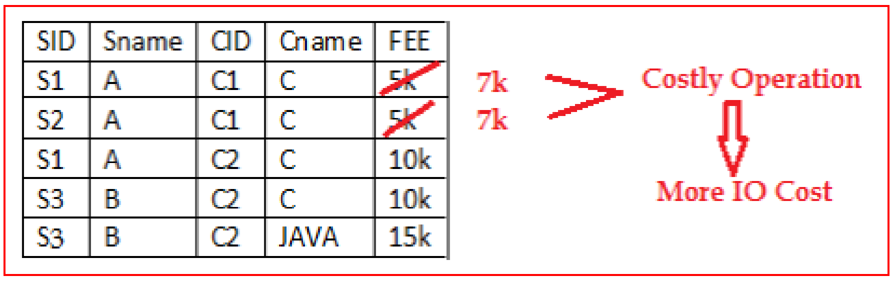
Redundant storage: Some information is stored repeatedly.

Update anomalies: If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

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

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

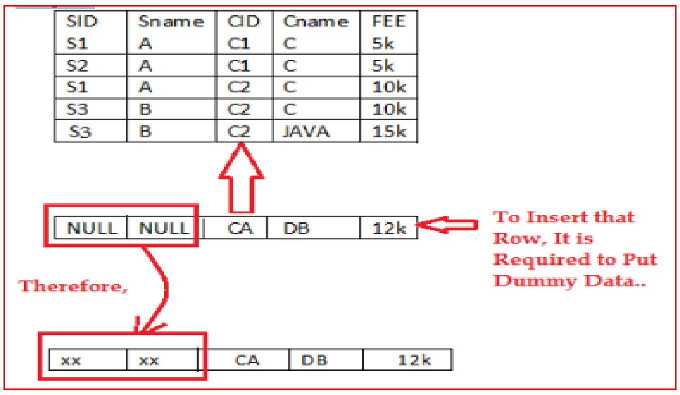


SID	Sname	CID	Cname	FEE
S1	A	C1	C	~~5k~~ 7k
S2	A	C1	C	~~5k~~ 7k
S1	A	C2	C	10k
S3	B	C2	C	10k
S3	B	C2	JAVA	15k

Costly Operation
More IO Cost

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

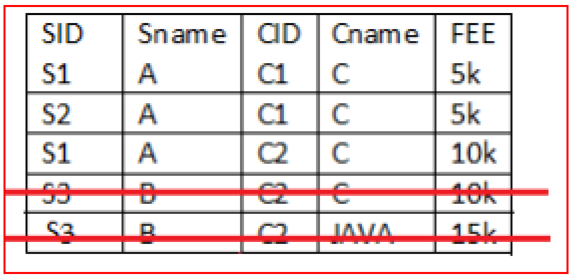
InsertionAnomalies: New course is introduced C4, But no student is there who is having C4 Subject



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

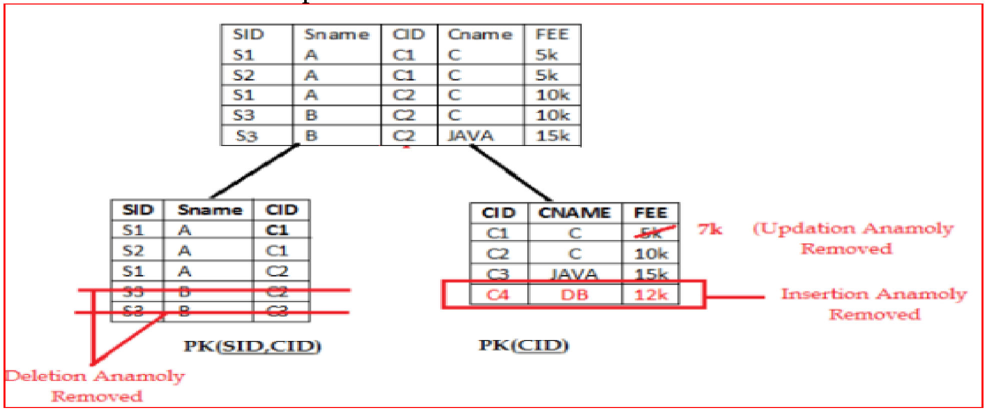
NULL	NULL	CA	DB	12k
xx	xx	CA	DB	12k

Because of insertion of some data, It is forced to insert some other dummy data.
Deletion Anomaly: Deletion of S3 student causes 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



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

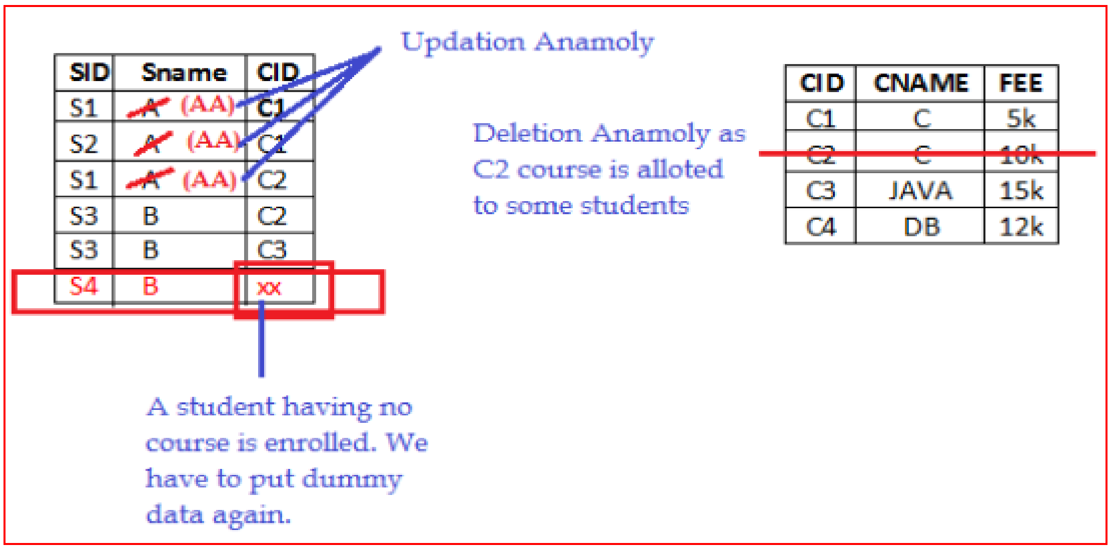
SID	Sname	CID
S1	A	C1
S2	A	C1
S1	A	C2
~~S3~~	~~B~~	~~C2~~
~~S3~~	~~B~~	~~C3~~

PK(SID,CID)

CID	CNAME	FEE
C1	C	~~5k~~ 7k
C2	C	10k
C3	JAVA	15k
C4	DB	12k

PK(CID)

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

Updation Anamoly

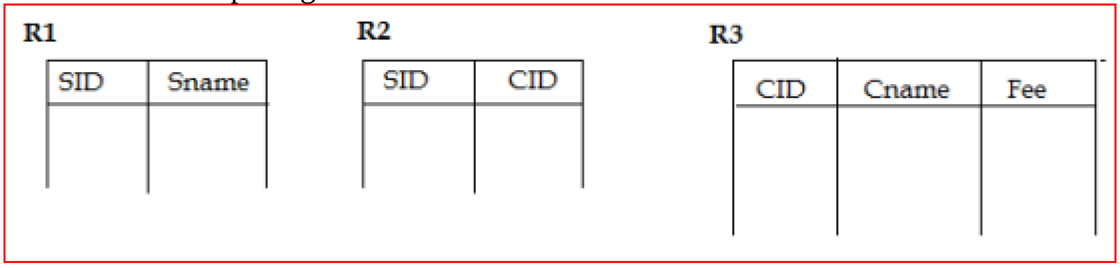
Deletion Anamoly as C2 course is alloted to some students

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

A student having no course is enrolled. We have to put dummy data again.

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

<u>Rollnum</u>	Name	Age	<u>Book#</u>	Subject
101	Anil	20	B1	NETWORKS
102	Abshik	21	B2	ALGORITHMS
103	Bhanu	20	B10	DBMS
101	Anil	20	B21	ECOM
NULL			B30	

FUNCTIONAL DEPENDENCIES:

Association among attributes is called functional dependency. It is represented by A→B.A functional dependency (FD) is a kind of IC that generalizes the concept of a key. Let R be a relation schema and let X and Y be nonempty sets of attributes in R. We say that an instance r of R satisfies the FD X → Y 1 if the following holds for every pair of tuples t1 and t2 in r: If t1.X = t2.X, then t1.Y = t2.Y.An FD X → Y essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y. 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

Example: Rno→name,Aadharnum→Ename ,Pnumber →{Pname, Plocation}, Dno→Dno, Dname, Rno,name→age

Reasoning about functional dependencies:

Armstrong's Axioms: Armstrong axioms define 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:

Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

Primary axioms: If $X \supseteq Y$, then $X \to Y$

1. Reflexive Rule (IR1): In the reflexive rule, if Y is a subset of X, then X determines Y.

Example: X = {a, b, c, d, e} Y = {a, b, c}

2. Augmentation Rule (IR2): If $X \to Y$, then $XZ \to YZ$ for any Z.

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z. Example: For R(ABCD), if $A \to B$ then $AC \to BC$

3. Transitive Rule (IR3): If $X \to Y$ and $Y \to Z$ then $X \to Z$

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

Some additional rules or secondary or derived axioms:

4. Union Rule (IR4): If $X \to Y$ and $X \to Z$, then $X \to YZ$.

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

5. Decomposition Rule (IR5): If $X \to YZ$, then $X \to Y$ and $X \to Z$.

Decomposition rule is also known as project rule. It is the reverse of union rule. This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

6. Pseudo transitive Rule (IR6): If $X \to Y$ and $YZ \to W$ then $XZ \to W$

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

Why Armstrong axioms refer to the Sound and Complete ?

By sound, we mean that given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using the primary rules of Armstrong axioms holds in every relation state r of R that satisfies the dependencies in F.

By complete, we mean that using primary rules of Armstrong axioms repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F.

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 adds elements to the result set which can be functionally determined from the elements of result set.

Algorithm: Computing the Attribute Closure of Attribute Set X

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

Method:

```
1.Closure $X^+ = X$;
2.for each FD :$Y \rightarrow Z$ in F do:
      if $Y \subseteq X^+$ then            // if Y is contained in $X^+$
        $X^+ = X^+ \cup Z$                  // 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:

- Non-Trivial functional dependency
- Trivial functional dependency
- Fully Functional Dependency
- Partial Functional Dependency
- Transitive functional dependency
- Multivalued functional dependency

1. Non-trivial Functional Dependency: If A → B has a non-trivial functional dependency if B is not a subset of A. When A intersection B is NULL, then A → B is called as complete non-trivial. Completely non-trivial functional dependency:-If X->Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency. In Non-trivial functional dependency, the dependent is strictly not a subset of the determinant.

Example: AB→C , ID → Name,

roll_no	name	age
42	smith	17
43	john	18
44	peter	18
42	raju	17

Here, **roll_no** → **name** is a non-trivial functional dependency, since the dependent name is not a subset of determinant roll_no

Similarly, {roll_no, name} → age is also a non-trivial functional dependency, since age is not a subset of {roll_no, name}

2. Trivial Functional Dependency:If A → B has trivial functional dependency if B is a subset of A. In Trivial Functional Dependency, a dependent is always a subset of the determinant i.e. If X → Y and Y is the subset of X, then it is called trivial functional dependency

The following dependencies are also trivial like: A → A, B → B

AB→AB. For example,

roll_no	name	age
42	smith	17
43	john	18
44	peter	18

Here, {roll_no, name} → name is a trivial functional dependency, since the dependent name is a subset of determinant set {roll_no, name}

Similarly, roll_no → roll_no is also an example of trivial functional dependency.

3. Fully Functional Dependency:

If X and Y are an attribute set of a relation, Y is fully functional dependent on X, if Y is functionally dependent on X but not on any proper subset of X.

Example :In the relation ABC ->D, attribute D is fully functionally dependent on ABC and not on any proper subset of ABC. That means that subsets of ABC like AB, BC, A, B, etc cannot determine D. Example :

supplier_id	item_id	price
1	1	540
2	1	545
1	2	200
2	2	201
1	1	540
2	2	201
3	1	542

From the table, we can clearly see that neither supplier_id nor item_id can uniquely determine the price but both supplier_id and item_id together can do so. So we can say that price is fully functionally dependent on { supplier_id, item_id }. This summarizes and gives our fully functional dependency .{ supplier_id , item_id } -> price

4. Partial Functional Dependency: A functional dependency X→Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.A Functional dependency X→Y is a partial dependency if, X is a part of candidate key and Y is a non-key attribute(s).

Let R(A,B,C) and AB=candidate key.

A-> C is a partial dependency (where C dependson a part of the candidate key).

AB->C is a full dependency (where c depends on the entire candidate key).

Example

R(ABCDE)

F: {AB->C, C->D, B->E }. Find partial dependency.

Solution

AB^+ = {A,B,C,D }

C^+={ C,D }

B^+= {B,E}

=>AB is the only candidate key.

=> key attributes =A,B and

=> non-key attributes = C,D,E.

Key attributes are also called prime attributes and non-key attributes are also called non-prime attributes.AB->C is not a Partial dependency // it is full dependency.

B-> E is a partial dependency, since B is a part of the candidate key and E is a non-key attribute.

Differences between Full Functional Dependency (FFD) and Partial Functional Dependency (PFD):

Full Functional Dependency	Partial Functional Dependency
A functional dependency X->Y is a fully functional dependency if Y is functionally dependent on X and Y is not functionally dependent on any proper subset of X.	A functional dependency X->Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.
In full functional dependency, the non-prime attribute is functionally dependent on the candidate key.	In partial functional dependency, the non-prime attribute is functionally dependent on part of a candidate key.
In fully functional dependency, if we remove any attribute of X, then the dependency will not exist anymore.	In partial functional dependency, if we remove any attribute of X, then the dependency will still exist.
Full Functional Dependency equates to the normalization standard of Second Normal Form.	Partial Functional Dependency does not equate to the normalization standard of Second Normal Form. Rather, 2NF eliminates the Partial Dependency.
An attribute A is fully functional dependent on another attribute B if it is functionally dependent on that attribute, and not on any part (subset) of it.	An attribute A is partially functional dependent on other attribute B if it is functionally dependent on any part (subset) of that attribute.
Functional dependency enhances the quality of the data in our database.	Partial dependency does not enhance the data quality. It must be eliminated in order to normalize in the second normal form.

4. Transitive Functional Dependency:

In transitive functional dependency, dependent is indirectly dependent on determinant.i.e. If A → B &B →C, then according to axiom of transitivity, A → C. This is a transitive 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: or Attribute Closure:

Consider the relation scheme R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R.

What is the key for R?

Answer: Finding attribute closure of all given options, we get:

{E,F}+ = {EFGIJ}

{E,F,H}+ = {EFHGIJKLMN}

{E,F,H,K,L}+ = {{EFHGIJKLMN}

{E}+ = {E}

{EFH}+ and {EFHKL}+ results in set of all attributes, but EFH is minimal. So it will be candidate key.

Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold:

{A–>B, BC–> D, E–>C, D–>A}. What are the candidate keys of R?

Normal Forms:

Normalization is the process of minimizing redundancy from a relation or set of relations. Normalization is the process of organizing the data in the database. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. Normalization divides the larger table into the smaller table and links them using relationship.

Types of Normal Forms:

- First Normal Form(1NF)
- Second Normal Form(2NF)
- Third Normal Form(3NF)
- Boyce and Codd Normal Form (BCNF)
- Fourth Normal Form(4NF)
- Fifth normal form (5NF)
- Domain-key normal form or DKNF
- Sixth normal form or (6NF)

First Normal Form(1NF):

A relation is in 1NF if it contains an atomic value. or It must hold only single-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute. It states that an attribute of a table cannot hold multiple values. A relation is in first normal form if it does not contain any composite or multi-valued attribute. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: First Normal Form.

roll_no	name	subject
101	Anil	AWS, Python
103	smith	Java
102	peter	ML, AI

But out of the 3 different students in our table, 2 have opted for more than 1 subject.And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value. Here is our updated table and it now satisfies the First Normal Form.

roll_no	Name	subject
101	Anil	AWS
101	Anil	Python
103	smith	Java
102	peter	ML
102	peter	AI

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row. using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Example:

Domain	Courses
Programming	Java, C
Web designing	PHP, HTML

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	Java
Programming	C
Web designing	PHP

Web designing	HTML

Second Normal Form :(2NF) :

A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. In the second normal form, all non-key attributes are fully functional dependent on the primary key. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table. In second normal form non-prime attributes should not depend on proper subset of key attributes.

Fully Functional Dependency (FFD): If X and Y are an attribute set of a relation, Y is fully functional dependent on X, if Y is functionally dependent on X but not on any proper subset of X.

Partial Dependency (PFD): If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example: Consider following functional dependencies in relation R (A, B , C, D)

AB → C [A and B together determine C]

BC → D [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

Example:

Student id	Student name	Project Id	Project name
101	smith	P1	x
102	john	P2	y

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
101	smith	P1
102	john	P2

Project id	Project name
P1	x

P2	y

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.

3NF is used to reduce the data duplication. It is also used to achieve the data integrity. If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form. Update anomaly is caused by a transitive dependency.

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

Let R be a relation schema, X be a subset of the attributes of R, and A be an attribute of R. R is in third normal form if for every FD X → A that holds over R, one of the following statements is true:

•A ∈ X; that is, it is a trivial FD, or

•X is a super key, or

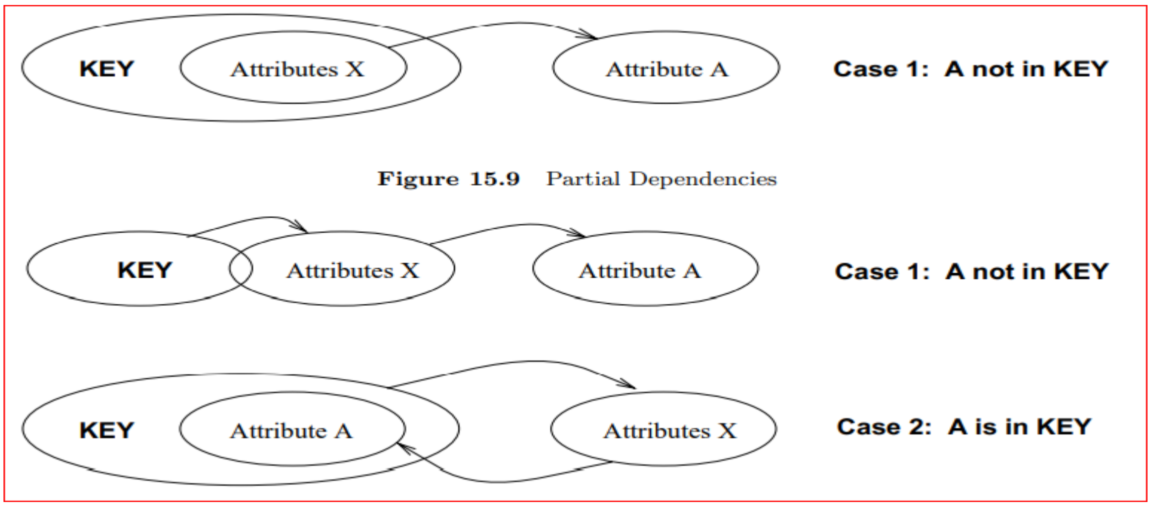
•A is part of some key for R

OR

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, i.e., each element of Y is part of some candidate key.

Transitive Functional dependency (TFD): If A→B and B→C are two FDs then A→C is called transitive dependency.



Transitive Dependencies

Suppose that a dependency X → A causes a violation of 3NF. There are two cases:

1.X is a proper subset of some key K. Such a dependency is sometimes called a partial dependency.

2.X is not a proper subset of any key. Such a dependency is sometimes called a transitive dependency because it means we have a chain of dependencies K → X → A. The problem is that we cannot associate an X value with a K value unless we also associate an A value with an X value.

Boyce-Codd Normal Form (BCNF):Let R be a relation schema, X be a subset of the attributes of R, and let A be an attribute of R. R is in Boyce-Codd normal form if for every FD X → A that holds over R, one of the following statements is true:

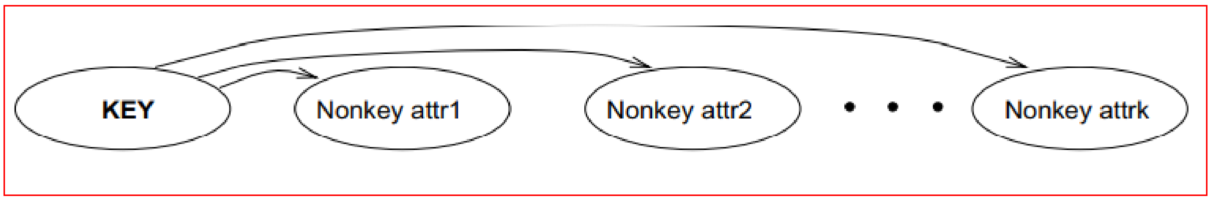
- A ∈ X; that is, it is a trivial FD, or
- X is a superkey

OR

a relation schema R in BCNF with respect to a set F of functional dependencies if ,for all functional dependencies they are in the form of x→A where X,B subset of R at least one of the following holds :

- X→A is trival FD (ie A subset of X)
- X is the super key for schema R

a relation is in BCNF every determinant is a candidate key. if non-prime attribute determine the one or more prime attribute then the relation not in BCNF.



FDs in a BCNF Relation

SID	Major	Advisor
123	physics	faculty1
123	music	faculty2
456	Bio	faculty3
789	physics	faculty4
999	physics	faculty1

There are two FDs in this Relation

- SID Major →Advisor
- Advisor →Major

Std_adv(SID,Advisor)
Adv_maj(Advisior,Major)

SID	Advisor
123	faculty1
123	faculty2
456	faculty3
789	faculty4
999	faculty1

Advidor	Major

faculty1	physics
faculty2	music
faculty3	Bio
faculty4	Physics

****

Example: convert the following relation into 1NF,2NF,3NF

so to avoid the repeating groups develop the following relation.

SID	sname	Caddr	major	CID	Ctitle	Iname	Iloc	grade
111	Anil	208 west	IS	IS11	DBMS	SB	203 east	A
111	Anil	208 west	IS	IS22	SAD	AR	204 West	B
222	Adhir	104 East	IT	IS11	DBMS	SB	203 East	C
222	Adhir	104 East	IT	IT22	COA	SB	203 East	B
222	Adhir	104 East	IT	IT33	C	HK	209 South	A

Relation name: Grade Report relation in 1NF

Second Normal Form:(2NF): in the above Grade relation there are 4 FDs

- SID-->Snsme ,Caddr,Major
- CID-->Ctitle,Iname,Iloc
- SID,CID-->Grade
- Iname-->Iloc

key: composite key is(SID,CID)

In the above grade Report relation is not in the 2NF because of there is a trival dependency exists.

CID→Ctitle,Iname,Iloc

Iname→Iloc .Here CID determines Iname and Iname determines Iloc is the transitive Dependency.
 To remove the partial dependencies decompose the relation into smaller relations. decompose the Grade Report relation into three relations they are Student Relation, Course Relation, Registration relation.

Student Relation

SID	Snsme	Caddr	Major
111	Anil	208 west	IS
222	Adhir	104 East	IT

Course Relation

CID	Ctitle	Iname	Iloc
IS11	DBMS	SB	203 east
IS22	SAD	AR	204 West
IS11	DBMS	SB	203 East
IT22	COA	SB	203 East
IT33	C	HK	209 South

CID →Ctitle

CID→Iname

CID→Iloc

Iname→Iloc is transitive dependency hence course relation not in 3NF But it is in 2NF

Registration relation

SID	CID	Grade
111	IS11	A
111	IS22	B
222	IS11	C
222	IT22	B
222	IT33	A

Multivalued Dependency (MVD) and define Fourth Normal Form:

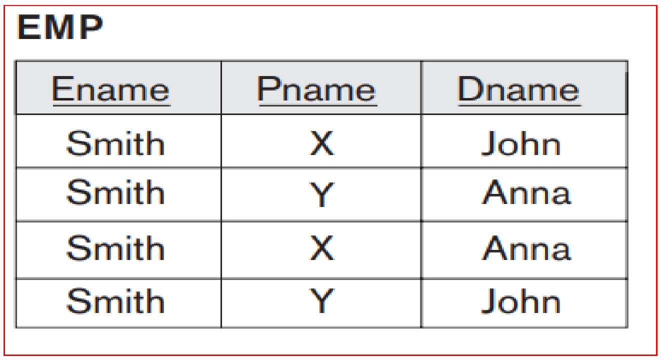
Definition: A multivalued dependency $X \twoheadrightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties, where we use Z to denote (R – (X ∪ Y))

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

Whenever $X \twoheadrightarrow Y$ holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R, so does $X \twoheadrightarrow Z$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$, and therefore it is sometimes written as $X \twoheadrightarrow Y|Z$

An MVD $X \twoheadrightarrow Y$ in R is called a trivial MVD if (a) Y is a subset of X, or (b) X ∪ Y = R.

For example, the relation EMP_PROJECTS in Figure has the trivial MVD
$\text{Ename} \twoheadrightarrow \text{Pname}$. An MVD that satisfies neither (a) nor (b) is called a nontrivial MVD.


EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Fig:The EMP relation with two MVDs: $\text{Ename} \twoheadrightarrow \text{Pname}$ and $\text{Ename} \twoheadrightarrow \text{Dname}$

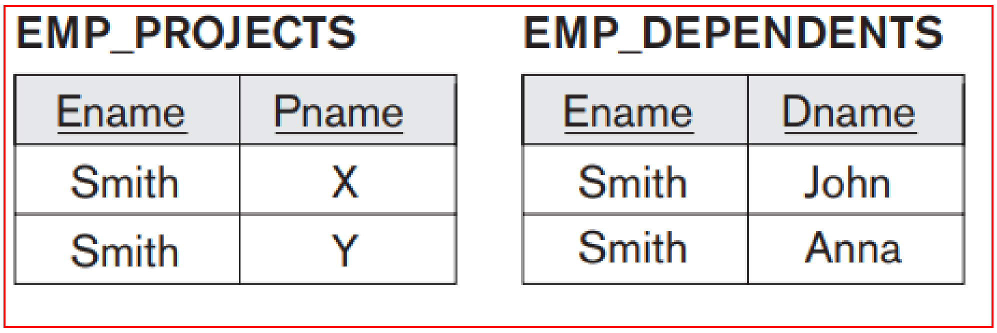
Fourth Normal Form (4NF)

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

- An all-key relation is always in BCNF since it has no FDs.

- An all-key relation such as the EMP relation in ABOVE Figure which has no FDs but has the MVD Ename →→ Pname | Dname, is not in 4NF.
- A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF.
- The decomposition removes the redundancy caused by the MVD

The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD. Consider the EMP relation in Figure EMP is not in 4NF because in the nontrivial MVDs Ename $\twoheadrightarrow$ Pname and Ename $\twoheadrightarrow$ Dname, and Ename is not a superkey of EMP.



EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

Fig:Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS, shown in above Figure. Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs Ename $\twoheadrightarrow$ Pname in EMP_PROJECTS and Ename $\twoheadrightarrow$ Dname in EMP_DEPENDENTS are trivial MVDs

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)

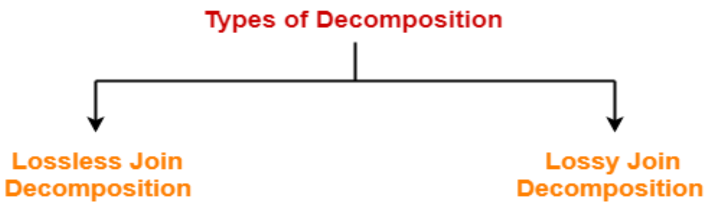
2) Both relations interaction should not give null value.

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.

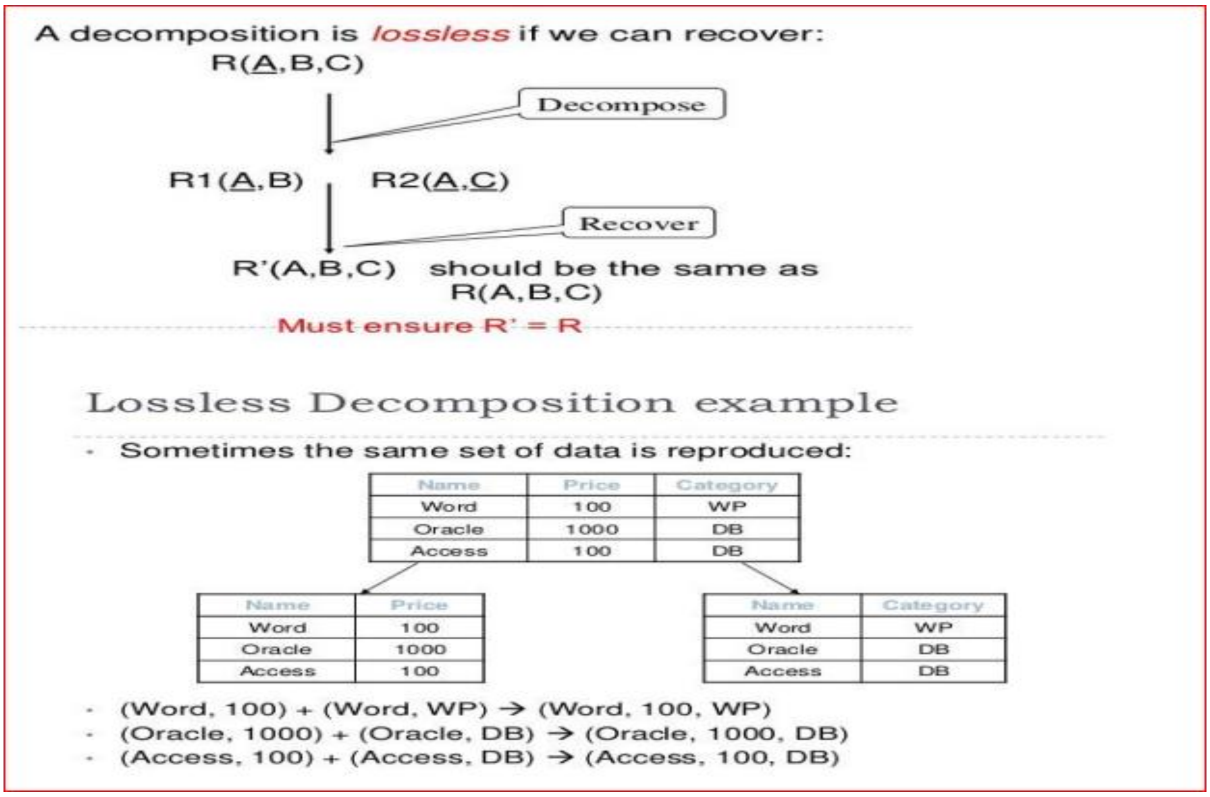
No information is lost from the original relation during decomposition.When the sub relations are joined back, the same relation is obtained that was decomposed.Every decomposition must always be lossless.

- •Consider there is a relation R which is decomposed into sub relations R1 , R2 , …. , Rn.
- •This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
- •For lossless join decomposition, we always have-

$$\boxed{R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots \bowtie R_n = R}$$

where ⋈ is a natural join operator

Lossless Join Decomposition:



A decomposition is *lossless* if we can recover:

R(A,B,C)

Decompose

R1(A,B) R2(A,C)

Recover

R'(A,B,C) should be the same as R(A,B,C)

Must ensure R' = R

Lossless Decomposition example

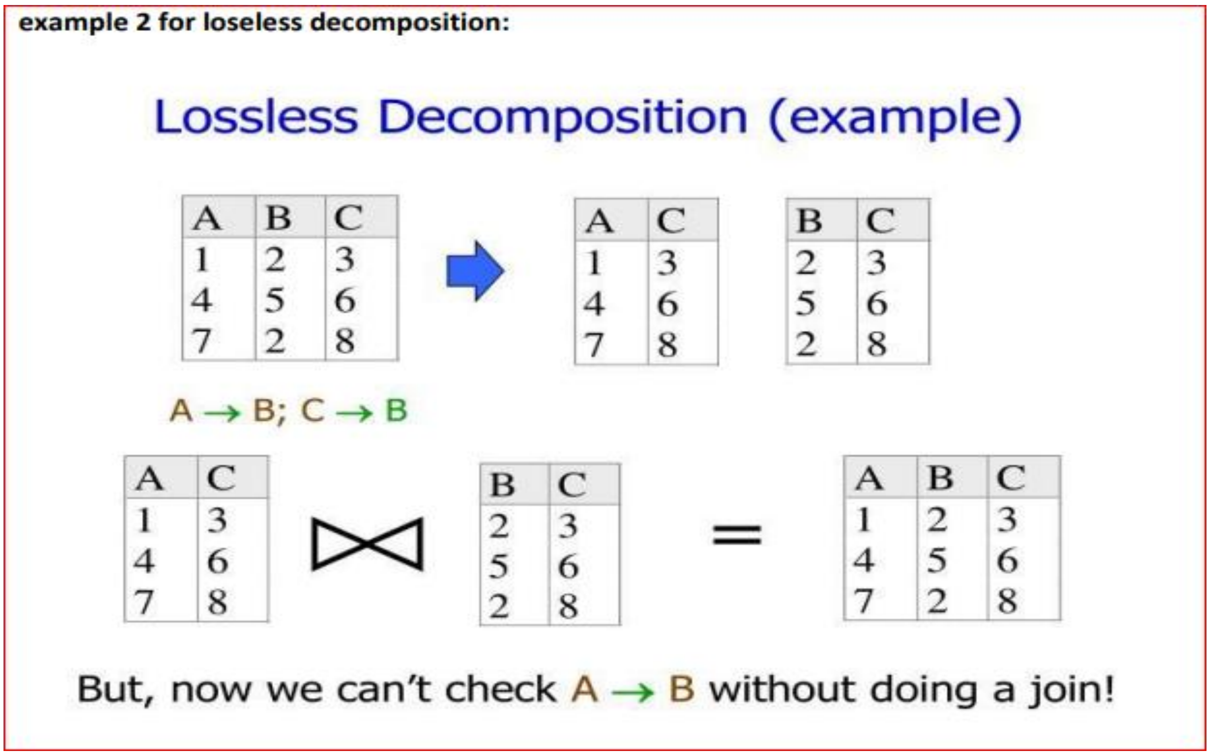
- Sometimes the same set of data is reproduced:

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

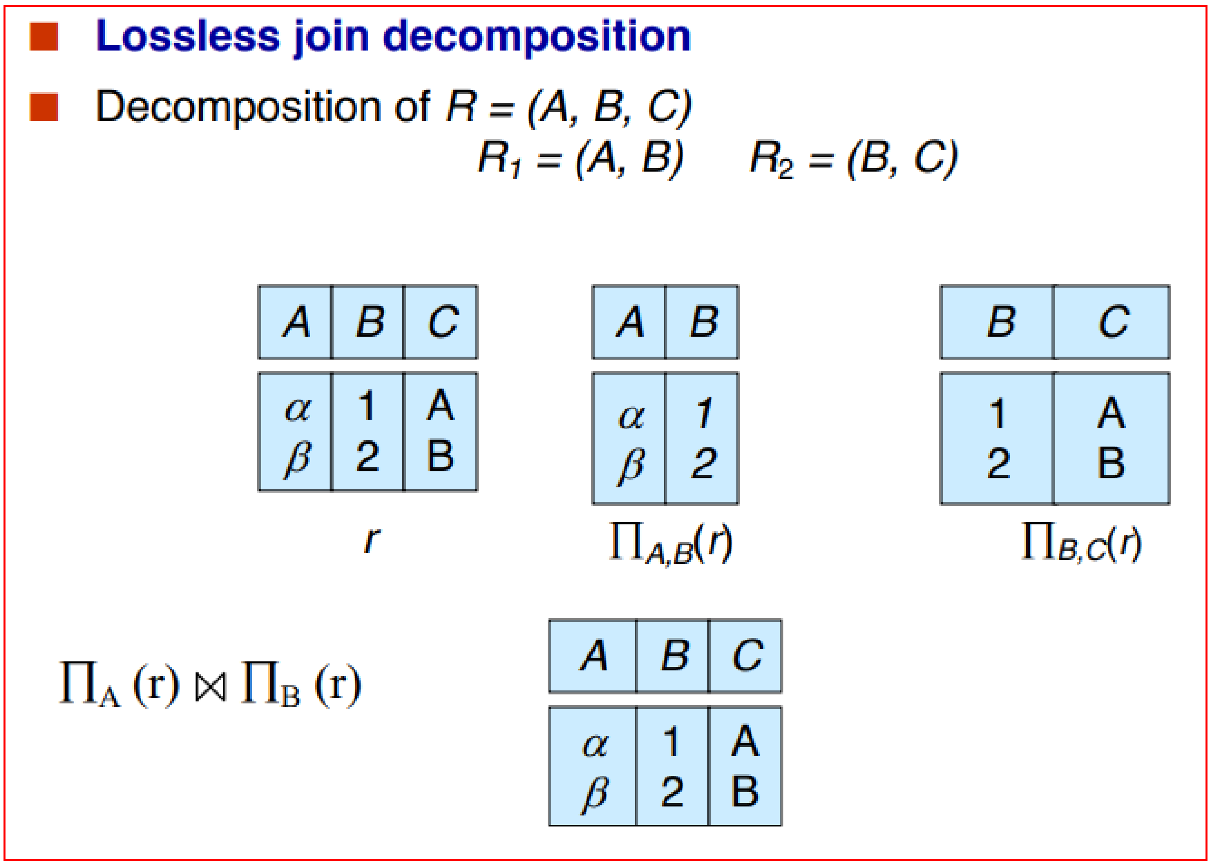
$\bowtie$

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!



- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$
  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

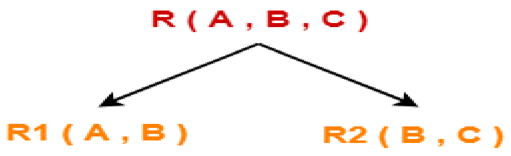
A	B	C
α	1	A
β	2	B

Example:Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations R1(A , B) and R2(B , C)



The two sub relations are-

A	B
1	2
2	5
3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossless or not.For lossless decomposition, we must have

$$R_1 \bowtie R_2 = R$$

Now, if we perform the natural join ($\bowtie$) of the sub relations R1 and R2 , we get-

A	B	C
1	2	1
2	5	3
3	3	3

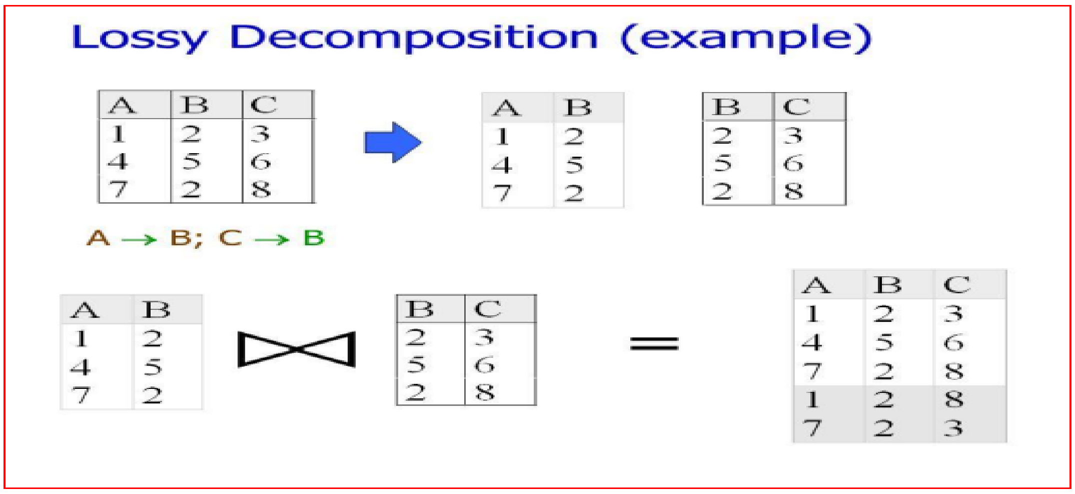
This relation is same as the original relation R.Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE: Lossless join decomposition is also known as non-additive join decomposition. This is because the resultant relation after joining the sub relations is same as the decomposed relation. No extraneous tuples appear after joining of the sub-relations.

2. Lossy Join Decomposition:

Consider there is a relation R which is decomposed into sub relations R1 , R2 , …. , Rn.This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed. The natural join of the sub relations is always found to have some extraneous tuples. For lossy join decomposition, we always have

$R1 \bowtie R2 \bowtie R3 \ldots\ldots \bowtie Rn \supset R$ where $\bowtie$ is a natural join operator.



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

$\bowtie$

B	C
2	3
5	6
2	8

=

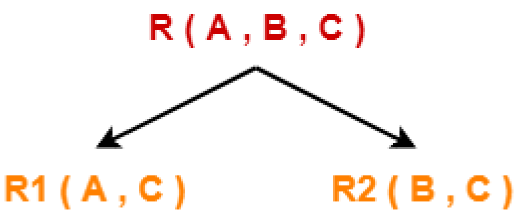
A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Example:Consider the following relation R(A , B , C)

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations as R1(A , C) and R2(B , C)



The two sub relations are

A	C
1	1

2	3
3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossy or not.For lossy decomposition, we must have **$R_1 \bowtie R_2 \supset R$** Now, if we perform the natural join (⋈) of the sub relations R_1 and R_2 we get

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not same as the original relation R and contains some extraneous tuples.Clearly,

$R_1 \bowtie R_2 \supset R$. Thus, we conclude that the above decomposition is lossy join decomposition.

NOTE:Lossy join decomposition is also known as careless decomposition. This is because extraneous tuples get introduced in the natural join of the sub-relations. Extraneous tuples make the identification of the original tuples difficult.

Dependency preservation:

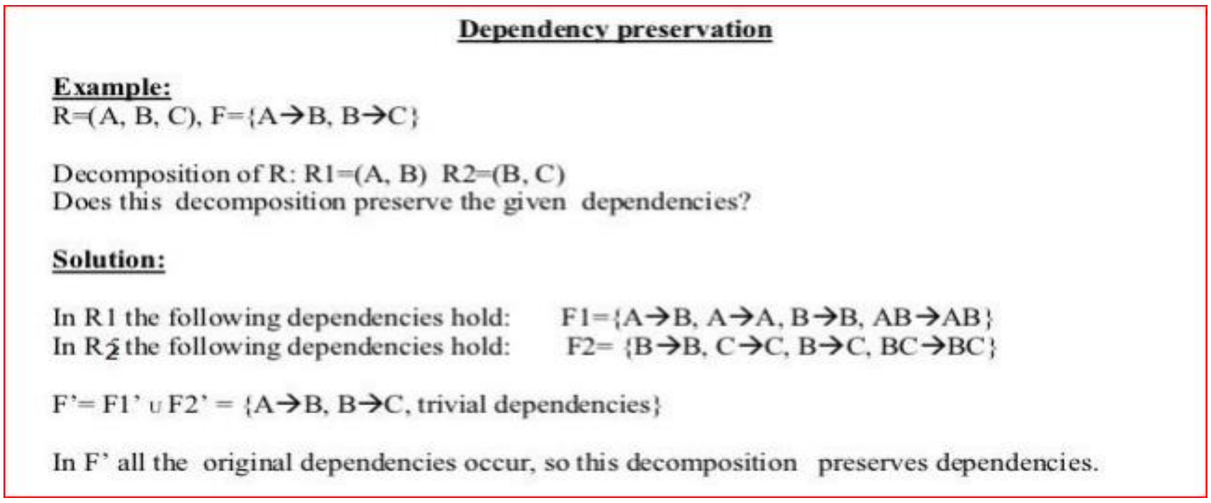
functional dependencies should be satisfied even after splitting relations and they should be satisfied by any of splitted tables. Dependency Preservation A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving w.r.t a set F of Functional dependency if (F1 ∪ F2 ∪ … ∪ Fm)+ = F+.

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.



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.