UNIT - III

Equivalence of Functional Dependencies

- Two or more than two sets of functional dependencies are called equivalence if the right-hand side of one set of functional dependency can be determined using the second FD set, similarly the right-hand side of the second FD set can be determined using the first FD set.
  - **How to find relationship between two FD sets?**
    - Let FD1 and FD2 are two FD sets for a relation R.
    - If all FDs of FD1 can be derived from FDs present in FD2, we can say that FD2 ⊃ FD1.
    - If all FDs of FD2 can be derived from FDs present in FD1, we can say that FD1 ⊃ FD2.
    - If 1 and 2 both are true, FD1=FD2.

STEP 1: Suppose two FD sets P and Q are given, write FD of each set P and Q separately.

STEP 2: Take FD set P first and then find closure of each attribute of the left side of FD in P using FD set Q.

STEP 3: Now compare the closure of each attribute of P with the right-hand side of FD of P.

STEP 4: If closure of each attribute is equal to the right-hand side of FD of P, we say P is a subset of Q

STEP 5: Take FD set Q next and then find closure of each attribute of the left side of FD in Q using FD set P.

STEP 6: Now compare the closure of each attribute of Q with the right-hand side of FD of Q.

STEP 7: If closure of each attribute is equal to right-hand side of FD of Q, we say Q is a subset of P

STEP 8: IF P is a subset of Q and Q is a subset of P, we can say P = Q.

Q 1: Given a relational schema R(X, Y, Z, W, V) set of FDs P and Q such that:
P = {X → Y, XY → Z, W → XZ, W → V} and
Q = { X → YZ, W → XV } using FD sets P and Q which of the following options are correct?

A. **P is a subset of Q**
B. **Q is a subset of P**
C. **P = Q**
D. **P ≠ Q**

→Using definition of equivalence of FD set, let us determine the right-hand side of the FD set of P using FD set Q.

- Given P = {X → Y, XY → Z, W → XZ, W → V} and Q = {X → YZ, W → XV }
  - **Let's find closure of the left side of each FD of P using FD Q.**
  - **X^+ = XYZ (using X → YZ using Q FD)**
  - **XY^+ = XYZ (using X → YZ)**
  - **W^+ = WXVYZ (using W → XV and X → YZ)**
  - **W^+ = WXVYZ (using W → XV and X → YZ)**
- Now compare closure of each X, XY, W and W calculated using FD Q with the right-hand side of FD P. Closure of each X, XY, W and W has all the attributes which are on the right-hand side of each FD of P.
  - **Hence, we can say P is a subset of Q---------1**

→Using definition of equivalence of FD set, let us determine the right-hand side of the FD set of Q using FD set P.

- Given P = { X → Y, XY → Z, W → XZ, W → V} and Q = { X → YZ, W → XV }
  - **Let us find closure of the left side of each FD of Q using FD P.**
  - **X^+ = XYZ (using X → Y and XY → Z)**
  - **W^+ = WXZVY (using W → XZ, W → V, and X → Y)**
- Now compare closure of each X, W calculated using FD P with the right-hand side of FD Q. Closure of each X and W has all the attributes which are on the right-hand side of each FD of Q.
  - **Hence, we can say Q is a subset of P---------2**

  From 1 and 2 we can say that P = Q, hence option C is correct.

Question2: Given a relational schema R(A, B, C, D) set of functional dependencies P and Q such that:
P = { A → B, B → C, C → D } and Q = { A → BC, C → D } using FD sets P and Q which of the following options are correct?
a) P is a subset of Q b) Q is a subset of P c) P = Q d) P ≠ Q

<u>Canonical Cover / Minimal Cover / Irreducible set of FDs</u>

- Whenever a user updates the database, the system must check whether any of the functional dependencies are getting violated in this process. If there is a violation of dependencies in the new database state, the system must roll back. Working with a huge set of **functional dependencies** can cause unnecessary added computational time. This is where the canonical cover comes into play.

Extraneous attributes:

- ➢ An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

Need-

- Working with the set containing extraneous functional dependencies increases the computation time.
- Therefore, the given set is reduced by eliminating the useless functional dependencies.

- This reduces the computation time and working with the irreducible set becomes easier.

Steps To Find Canonical Cover-

Step-01:

- Write the given set of functional dependencies in such a way that each functional dependency contains exactly one attribute on its right side. Eg: X→YZ then X→Y & X→Z

Step-02:

- Consider each functional dependency one by one from the set obtained in Step-01.
- Determine whether it is essential or non-essential.

- To determine whether a functional dependency is essential or not, compute the closure of its left side-
  - Once by considering that the particular functional dependency is present in the set
  - Once by considering that the particular functional dependency is not present in the set
  - Then following two cases are possible-

Results Come Out to be Same-

- If results come out to be same,
  - It means that the presence or absence of that functional dependency does not create any difference.
- Thus, it is non-essential.
  - Eliminate that functional dependency from the set

NOTE-

- Eliminate the non-essential functional dependency from the set as soon as it is discovered.
- Do not consider it while checking the essentiality of other functional dependencies.

Results Come Out to be Different-

- It means that the presence or absence of that functional dependency creates a difference.
  - Thus, it is essential.
- Do not eliminate that functional dependency from the set.
  - Mark that functional dependency as essential.

Step-03:

- Consider the newly obtained set of functional dependencies after performing Step-02.
- Check if there is any functional dependency that contains more than one attribute on its left side.

Then following two cases are possible-

No-

- There exists no functional dependency containing more than one attribute on its left side.
- In this case, the set obtained in Step-02 is the canonical cover.

Yes-

- There exists at least one functional dependency containing more than one attribute on its left side.
- In this case, consider all such functional dependencies one by one.
- Check if their left side can be reduced.

Use the following steps to perform a check-

- Consider a functional dependency.
- Compute the closure of all the possible subsets of the left side of that functional dependency.
- If any of the subsets produce the same closure result as produced by the entire left side, then replace the left side with that subset.

After this step is complete, the set obtained is the canonical cover.

Problem-

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

- **X → W**
- **WZ → XY**
- **Y → WXZ**

Step-01:

- Write all the functional dependencies such that each contains exactly one attribute on its right side-
  - **X → W**
  - **WZ → X**
  - **WZ → Y**
  - **Y → W**
  - **Y → X**
  - **Y → 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 $(X)^+ = \{ X \}$

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 }

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 → W
WZ → Y
Y → W
Y → X
Y → 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 }

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

For Y → W:

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

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

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 → W
WZ → Y
Y → X
Y → Z

For Y → X:

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

Ignoring Y → X, $(Y)^+$ = { Y , Z }

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

For Y → Z:

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

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

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

From here, our essential functional dependencies are-

X → W
WZ → Y
Y → X
Y → 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 → Y contains more than one attribute on its left side.

Considering WZ → 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 can not write **WZ → Y as W → Y or Z → Y.**

Thus, set of functional dependencies obtained in step-02 is the canonical cover.

Finally, the canonical cover is-

X → W
WZ → Y
Y → X
Y → Z

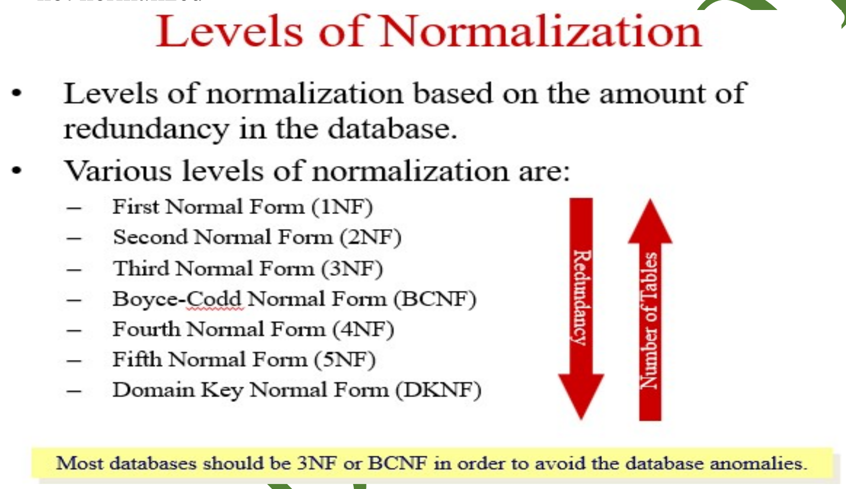
Canonical Cover

Question: Given a relational Schema R(A, B, C, D) and set of Function Dependency

FD = { B → A, AD → BC, C → ABD }. Find the canonical cover?

Normalization

- Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- Normalization is used for mainly two purposes,
  - Eliminating redundant(useless) data.
  - Ensuring data dependencies make sense i.e data is logically stored.
  - **Problems Without Normalization**
    - If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized


Redundancy
Number of Tables

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)

Most databases should be 3NF or BCNF in order to avoid the database anomalies.

When normalizing a database you should achieve four goals:

- **Arranging data into logical groups** such that each group describes a small part of the whole
- **Minimizing the amount of duplicated** data stored in a database
- Building a database in which you can **access and manipulate the data quickly and efficiently** without compromising the integrity of the data storage
- Organizing the data such that, when you modify it, you **make the changes in only one place**

First Normal Form (1NF)

- For a table to be in the First Normal Form, it should follow the following 4 rules:
  1. It should only have single(atomic) valued attributes/columns.
  2. Values stored in a column should be of the same domain
  3. All the columns in a table should have unique names.
  4. And the order in which data is stored, does not matter.

Example : Is below table in 1 NF?

Roll_no	Name	Subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

How to solve this Problem?

- It's very simple, because all we have to do is break the values into atomic values.
- Here is our updated table and it now satisfies the First Normal Form.

Roll_no	Name	Subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

- 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.

Second Normal Form (2NF):

- For a table to be in the Second Normal Form,
  - It should be in the First Normal form.
  - And, it should not have Partial Dependency.
    - (No non prime attribute should be partially dependant on candidate key)

Example:

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	36
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

Is Above table is in 2NF?

NO, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key.

- To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Third Normal Form (3NF)

- A table is said to be in the Third Normal Form when,
  1. It is in the Second Normal form.
  2. And, it doesn't have Transitive Dependency.

Example : A→B , B→C then A→C

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

- Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
- Candidate Keys: {emp_id}
- **Non-prime attributes:** all attributes except emp_id are non-prime as they are not part of any candidate keys.
- Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
- To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce-Codd Normal Form (BCNF)

- Boyce-Codd Normal Form or BCNF is an extension to the third normal form, and is also known as 3.5 Normal Form.
- For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:
  - **It should be in the Third Normal Form.**
  - **And, for any dependency A → B, A should be a super key.**
- The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a non-prime attribute, if B is a prime attribute.
- Example: Below we have a college enrolment table with columns student_id, subject and professor.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.CSharp
104	Java	P.Java

- In the table above **student_id, subject** together form the primary key.
- Hence, there is a dependency between subject and professor here, where subject depends on the professor name.
- Is Above table in 3NF?
  - ➢ This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.
  - ➢ This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.
  - ➢ And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.
  - ➢ But this table is not in **Boyce-Codd Normal Form**.

➢ **Why this table is not in BCNF?**

- In the table above, student_id, subject form primary key, which means subject column is a prime attribute.
- But, there is one more dependency, professor → subject.
- And while subject is a prime attribute, professor is a non-prime attribute, which is not allowed by BCNF.

How to satisfy BCNF?

- To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.
- Below we have the structure for both the tables.

Student Table

student_id	p_id
101	1
101	2
102	3
103	4
104	1

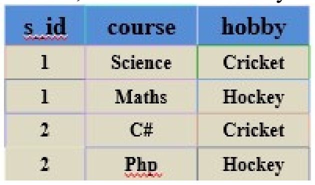
Professor Table

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
3	P.Java2	Java
4	P.Csharp	C#

Fourth Normal Form (4NF)

- For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:
  - It should be in the **Boyce-Codd Normal Form**.
  - And, the table should not have any **Multi-valued Dependency**.
- **What is Multi-valued Dependency?**
  - A table is said to have multi-valued dependency, if the following conditions are true,
    1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
    2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
    3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.
  - If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

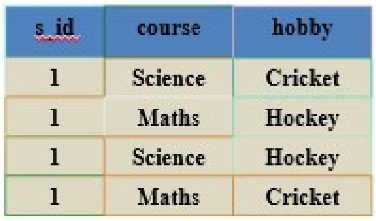
Notation: A →→ B
Example:
Below we have a college enrolment table with columns s_id, course, and hobby.
In the beside table, student with s_id 1 has opted for two courses, Science and Maths,and has two hobbies, Cricket and Hockey.



s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

what problem this can lead to…

- The two records for student with s_id 1, will give rise to two more records, as shown here, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

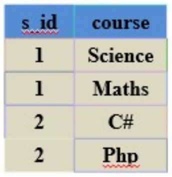


s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

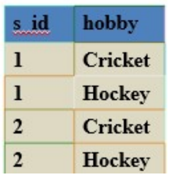
- And, in the table above, there is no relationship between the columns course and hobby. They are independent of each other.
- So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

How to satisfy 4th Normal Form?
To make the above relation satisfy the 4th normal form, we can decompose the table into 2 tables.

CourseOpted Table **And,** **Hobbies Table**



s_id	course
1	Science
1	Maths
2	C#
2	Php



s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

- **Now this relation satisfies the fourth normal form.**

- **NOTE:** A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.
- Example : R(s_id,course,hobby,address)
  - If MVD → s_id →→ course, s_id →→ hobby &
  - FD → s_id → address
  - To make R into 4NF, the above relation R is decompose into 3 tables
    - R1(s_id,course)
    - R2(s_id,hobby)
    - R3(s_id, address)

- Steps to find the highest normal form of relation:
  - Find all possible candidate keys of the relation.
  - Divide all attributes into two categories: prime attributes and non-prime attributes.
  - Check for 1st normal form then 2nd and so on. If it fails to satisfy the nth normal form condition, the highest normal form will be n-1.

Given a relation R(A, B, C, D, E) and Functional Dependency set FD = { A → B, B → E, C → D}, determine whether the given R is in 2NF? If not convert it into 2 NF.
Solution:

- **Let us calculate the closure of AC:**
  - AC+ = ACBED
- **2NF:** No non-prime attribute should be partially dependent on Candidate Key
  - Since R has 5 attributes: - A, B, C, D, E and Candidate Key is AC, Therefore, prime attribute (part of candidate key) are A and C while the non-prime attribute are B D and E
  - FD: **A → B** does not satisfy the definition of 2NF, as a non-prime attribute(B) is partially dependent on candidate key AC (i.e., key should not be broken at any cost).
  - FD: **B → E** does **not violate** the definition of 2NF, as a non-prime attribute(E) is dependent on the non-prime attribute(B), which is not related to the definition of 2NF.

- FD: **C → D** does not satisfy the definition of 2NF, as a non-prime attribute(D) is partially dependent on candidate key AC (i.e., key should not be broken at any cost)

Hence because of FD A → B and C → D, the above table R(A, B, C, D, E) is not in 2NF

Convert the table R(A, B, C, D, E) in 2NF:

- Since due to FD: A →B and C → D our table was not in 2NF, let's decompose the table
- R1(A, B, E) (from FD: A → B and B → E)
- R2(C, D) (Now in table R2 FD: C → D is Full FD, hence R2 is in 2NF)
- And create one table for candidate key AC → R3 (A, C)

Finally, the decomposed tables which are in 2NF:

- R1(A, B, E)
- R2(C, D)
- R3(A, C)

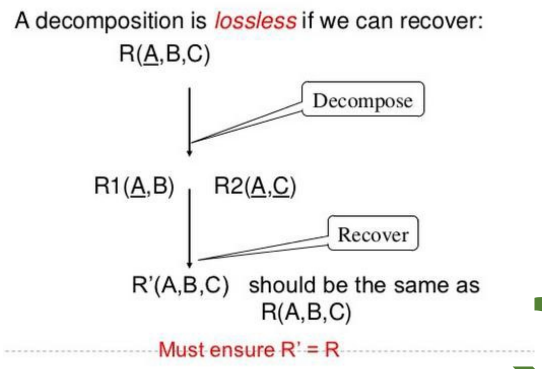
Decomposition:

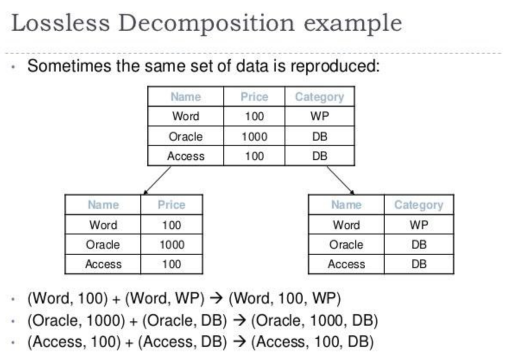
If 'R' is a relation splitted into 'R1' and 'R2' relations, the decomposition done should satisfy following-

1. Union of two smaller subsets of attributes gives all attributes of 'R'.
    a. R1(attributes)UR2(attributes)=R(attributes)
2. Both relations interaction should not give null value.
    a. R1(attributes)∩R2(attributes)!=null
3. Both relations interaction should give key attribute.
    a. 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.


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

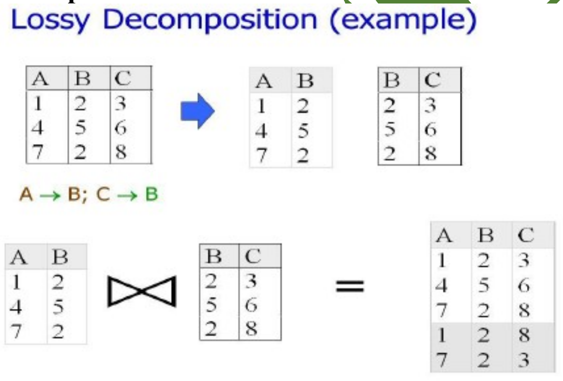
Name	Price
Word	100
Oracle	1000
Access	100

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)

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

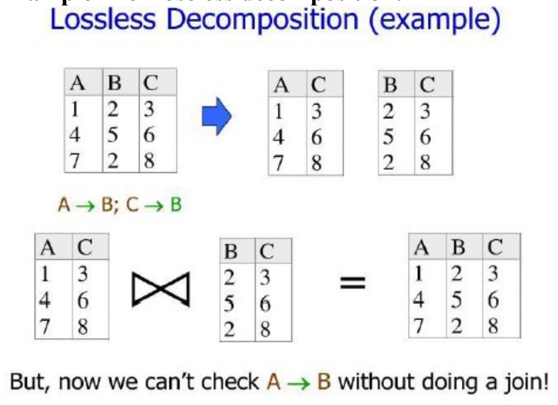
B	C
2	3
5	6
2	8

A → B; C → B

$\bowtie$ =

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

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 → B; C → B

$\bowtie$ =

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

But, now we can't check A → B without doing a join!

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 ∪ F2 ∪ … ∪ Fm)+ = F+.

Consider a relation R

R ---> F{...with some functional dependency(FD)....}

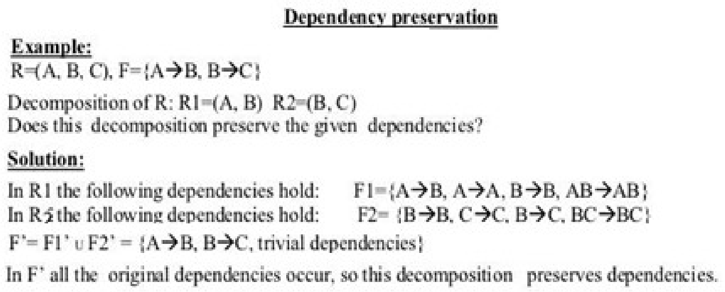
R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then there can be three cases:

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.

<u>Example for dependency preservation:</u>



<u>Dependency preservation</u>

<u>Example:</u>

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?

<u>Solution:</u>

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.