Schema Refinement

Database Design

Requirements Analysis

– user needs; what must the database do?

Conceptual Design

– High level descr. (Often done w/ER model)

Logical Design

– translate ER into DBMS data model

Schema Refinement

– Consistency, normalization

Physical Design

- Indexes, disk layout

Security Design

- Who accesses what?

What Do We Want to Do?

- We are given a set of tables specifying the database
- They come from some ER diagram
- We will need to examine whether the specific choice of tables is good for
  - Storing the information needed
  - Enforcing constraints
  - Avoiding anomalies, such as redundancies
- If there are issues to address, we may want to restructure the database, off-course not losing any information
- **Normalization 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.
- Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations.

Anomalies or Problems Facing without Normalization:

- Anomalies refers to the problems occurred after poorly planned and un normalized databases where all the data is stored in one table.
- Let us consider such type of schema –

SID	Sname	CID	Cname	FEE
S1	A	C1	C	5000
S2	A	C1	C	5000
S1	A	C2	C++	10000
S3	B	C2	C++	10000
S3	B	C2	JAVA	15000

- 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.

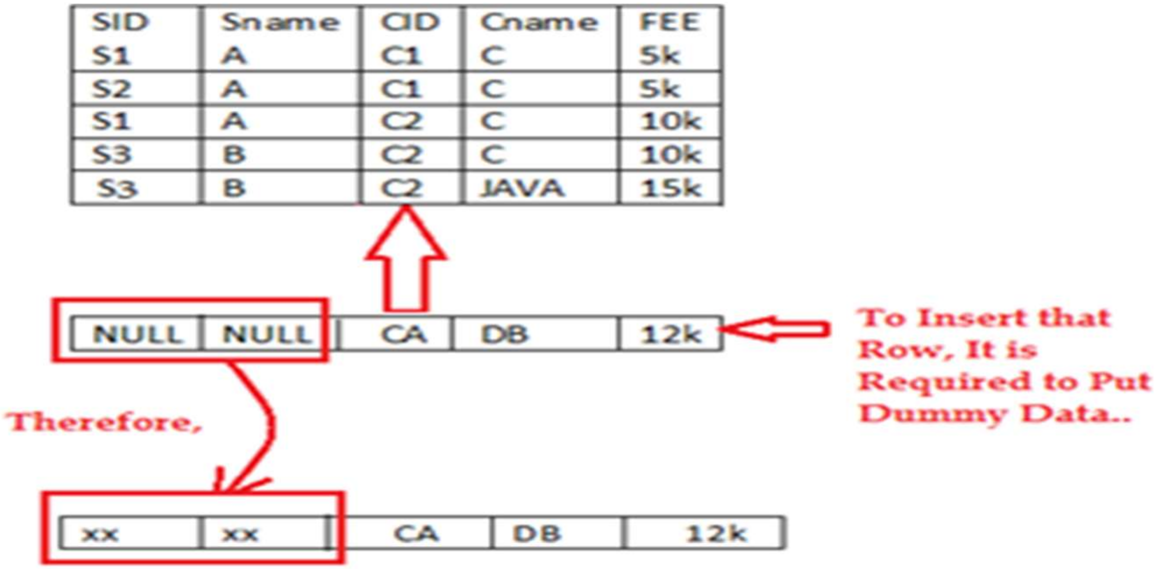
There are three types of Anomalies produced in the database because of redundancy –

- **Insertion anomalies:**
  - Insertion anomalies occur when attempting to add new data to a database, but are unable to do so because the data requires additional information.
- **Deletion anomalies:**
  - Deletion anomalies occur when removing data from a database, but accidentally removing related data as well.
- **Update anomalies:**
  - Update anomalies occur when updating data in a database, but the update affects multiple rows or columns unintentionally

Insertion Anomaly and Deletion Anomaly

These anomalies exist only due to redundancy, otherwise they do not exist.

- **Insertion Anomaly:**
  - New course is introduced C4, But no student is there who is having C4 subject.
  - Because of insertion of some data, It is forced to insert some other dummy 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

NULL	NULL	CA	DB	12k

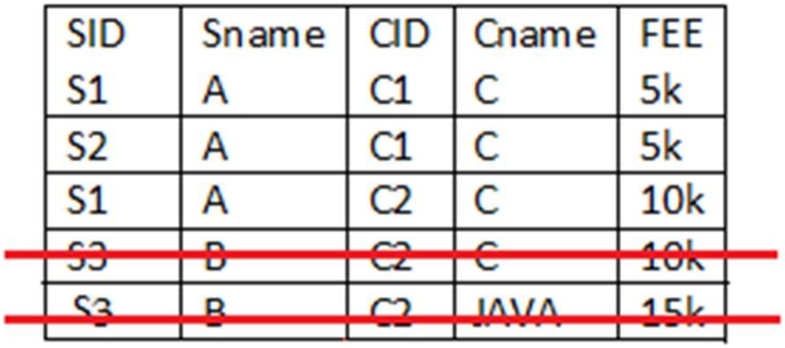
To Insert that Row, It is Required to Put Dummy Data..

Therefore,

xx	xx	CA	DB	12k

Deletion Anomaly

- Deletion of S3 student cause the deletion of course.
- Because of deletion of some data forced to delete some other useful data.

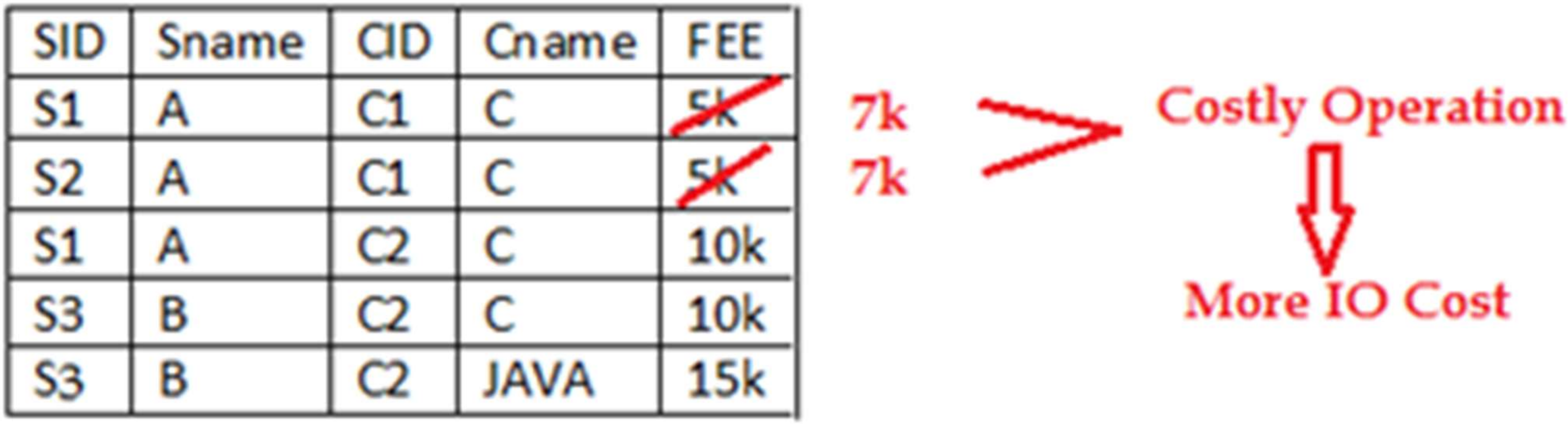


SID	Sname	CID	Cname	FEE
S1	A	C1	C	5k
S2	A	C1	C	5k
S1	A	C2	C	10k
~~S3~~	~~B~~	~~C2~~	~~C~~	~~10k~~
~~S3~~	~~B~~	~~C2~~	~~JAVA~~	~~15k~~

- Deleting student S3 will permanently delete the course B.

Problem in update / updation anomaly

– If there is updation in the fee for C is changed 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

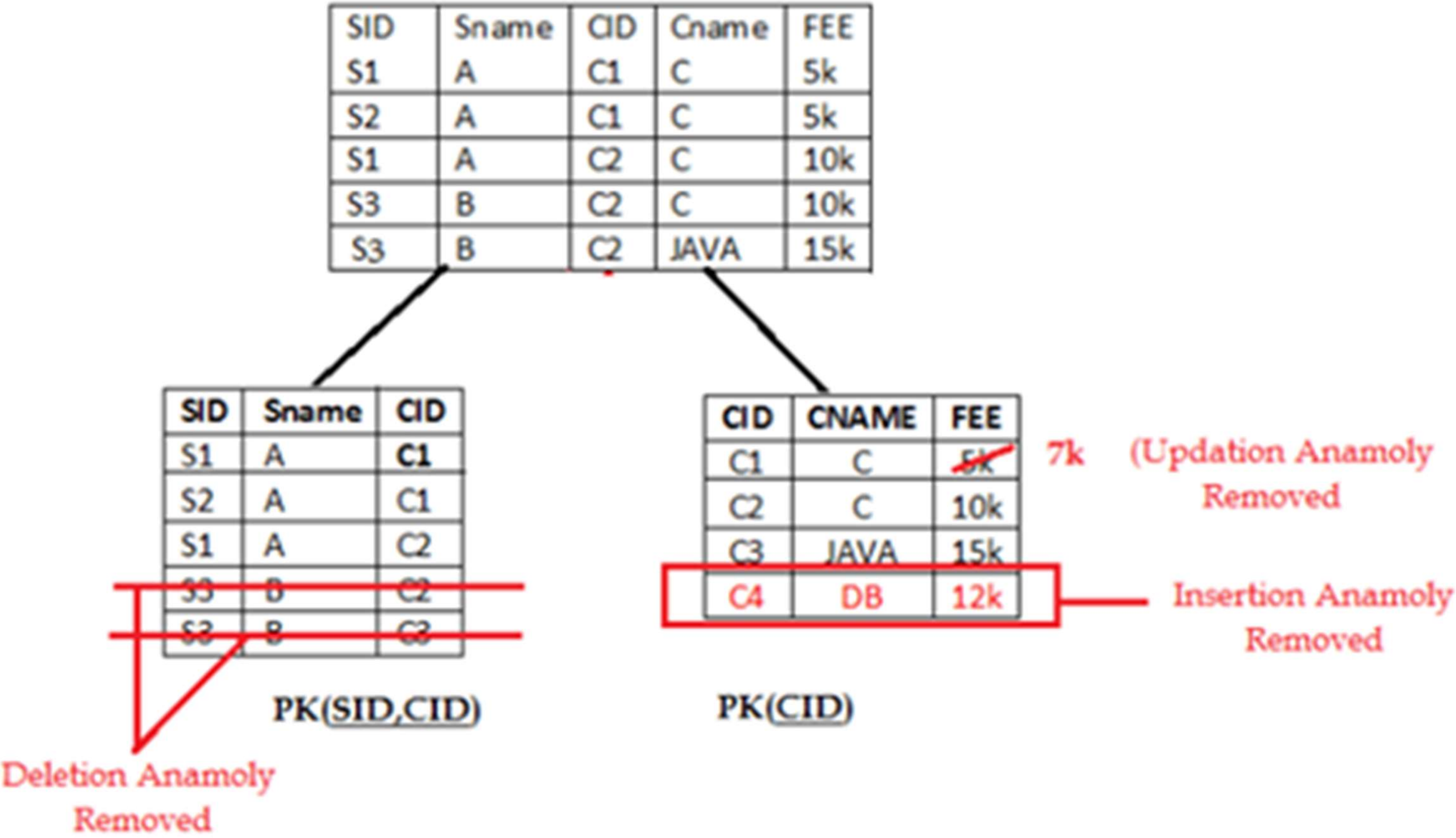
7k
7k

Costly Operation

More IO Cost

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)

Deletion Anamoly Removed

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

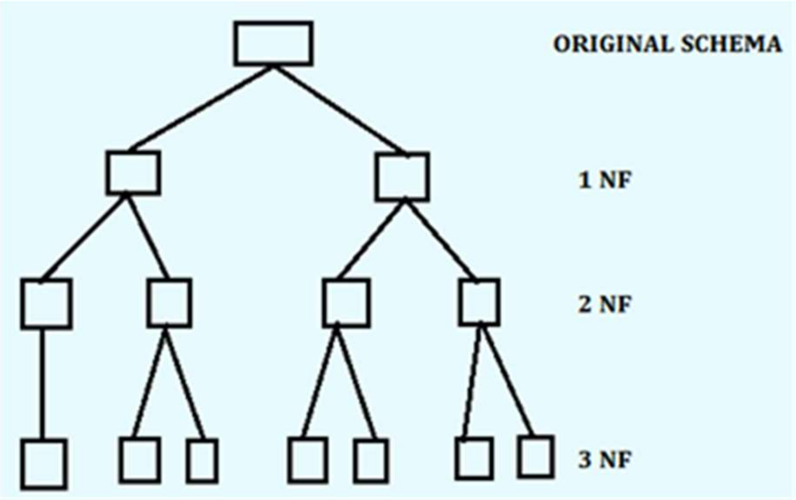
PK(CID)

(Updation Anamoly Removed

Insertion Anamoly Removed

Desirable Properties of Decomposition

- If we apply the normal forms or normalization or schema refinement technique – Decomposition to the universal table, then it may be splitted up into different fragments.



- At any stage, if we combine the fragments (de-normalization), it should give the original table in terms of columns and rows, and it will be described as the following properties:

Functional Dependency

- A **Functional dependency** is a relationship between attributes. Suppose that given the value of one attribute, we can obtain the value of another attribute.
- For example, if we know the value of customer account number, we can obtain customer address, balance etc. By this, we say that customer address and balance is functionally dependent on customer account number.
- In **general terms**, attribute Y (customer address and balance) is functionally dependent on the attribute X (customer account number), if the value of X determines the value of Y.
- Functional dependency is **represented** by an **arrow sign (→)** that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side. Attributes on the left side are called **Determinants.**

Functional Dependencies

1. If one set of attributes in a table determines another set of attributes in the table, then the second set of attributes is said to be functionally dependent on the first set of attributes.

Example 1

ISBN	Title	Price
0-321-32132-1	Balloon	$34.00
0-55-123456-9	Main Street	$22.95
0-123-45678-0	Ulysses	$34.00
1-22-233700-0	Visual Basic	$25.00

Table Schema: {ISBN, Title, Price}

Functional Dependencies: {ISBN} → {Title}
{ISBN} → {Price}

Functional Dependencies

Example 2

PubID	PubName	PubPhone
1	Big House	999-999-9999
2	Small House	123-456-7890
3	Alpha Press	111-111-1111

Table Schema: {PubID, PubName, PubPhone}

Functional Dependencies: {PubId} → {PubPhone}
{PubId} → {PubName}
{PubName, PubPhone} → {PubID}

Example 3

AuID	AuName	AuPhone
1	Sleepy	321-321-1111
2	Snoopy	232-234-1234
3	Grumpy	665-235-6532
4	Jones	123-333-3333
5	Smith	654-223-3455
6	Joyce	666-666-6666
7	Roman	444-444-4444

Table Schema: {AuID, AuName, AuPhone}
Functional Dependencies: {AuId} → {AuPhone}
{AuId} → {AuName}
{AuName, AuPhone} → {AuID}

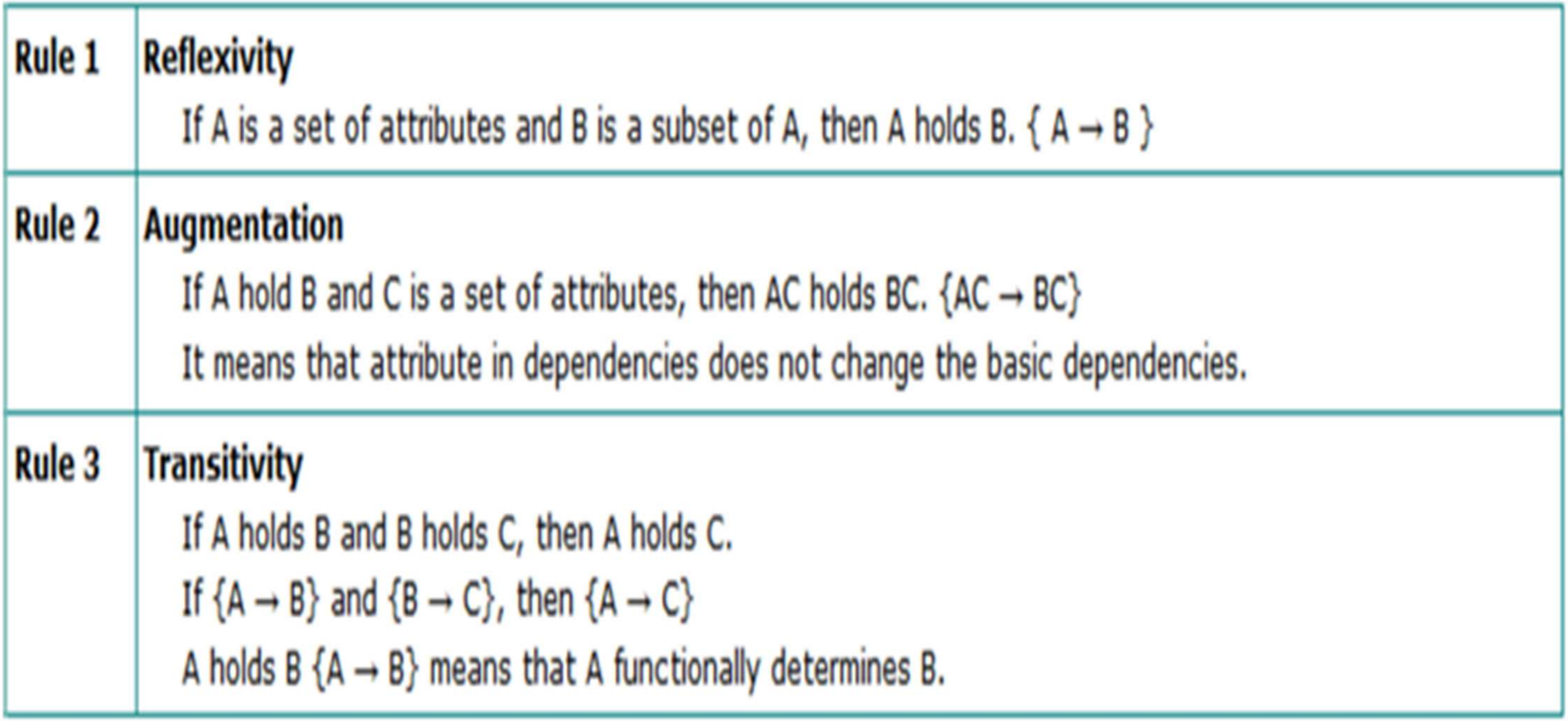
Reasoning about functional dependencies

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:

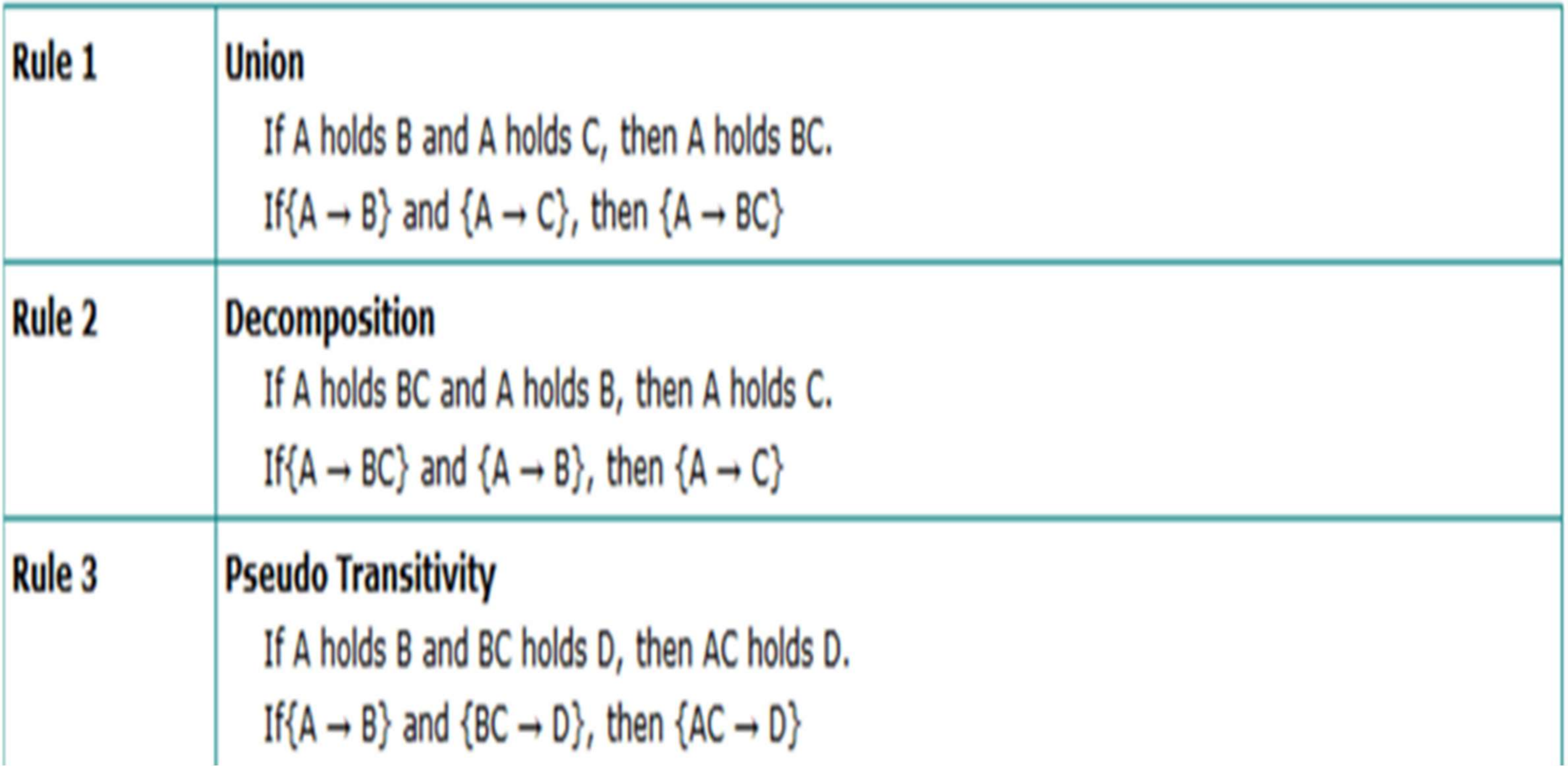
Primary axioms:



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.

Reasoning about functional dependencies..

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}

Properties of Decomposition

Following are the properties of Decomposition,

1. Lossless Decomposition (mandatory)
2. Dependency Preservation (optional)

Lossless Decomposition

- – Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed.
- – It gives a guarantee that the join will result in the same relation as it was decomposed.

Example:

- Let's take 'E' is the Relational, and E is decomposed into:
- E → E1, E2, E3, ... En, The above Relations is said to be lossless
- Iff E1⋈E2⋈E3 . . . . ⋈En =E,

⋈ → Natural Join

Lossless Join (Non –Additive) Questions

Procedure to find Lossless Decomposition:

- Let us assume R is divided into R1 & R2. The decomposition is lossless if the above decomposition must satisfy the following conditions.
- Attribute Preservation: Both relations must contain all the attributes of the original relation
  - **R1 U R2 = R**
- Intersection of both sub relations must be not null
  - **R1 ∩ R2 ≠ Ø**
- Intersection of both sub relations must have a super key of either R1 or R2 or both.
  - **R1 ∩ R2 → R1 (Super key of R1) (or)**
  - **R1 ∩ R2 → R2 (Super key of R2)**

2. Dependency Preservation

- Every dependency must be satisfied by at least one decomposed table.
- If {A → B} holds, then two sets are functional dependent. And it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

Example:

Let us Assume,

- R is an Original relation and
  - F^+ are its FD's
  - R is decomposed into
  - R1, R2, …….. Rn and F_1^+, F_2^+ ....F_n^+ are FD's respectively
  - Then $F_1^+ \cup F_2^+ \cup ....F_n^+ = F^+$ is dependency Preserving

Note: Some FDs of original relation are preserves directly and some FDs of original relation are preserved indirectly.

Question 1: R (ABCDE) and F = {AB → C, C → D, D→, E} and Decomposition R into R1(AB), R2(ACD) Whether the decomposition is Lossless or Lossy?

Solution: Lossy Decomposition (E is Missing, Not Preserving E Attribute)

Question 2: R (ABCD) and F = {A → B, B → C, C → D} and D = {AB, CD}

Solution: Lossy Decomposition (since R1 n R2 = Ø)

Question 3: R (ABCDE) and F = {A → B, C → DE, AC → E} and D = {BE, ACDEF}

Solution: Lossy Decomposition (R1 n R2 → R1 or R2)

Question 4: R (ABCD) and F = {AB → C, C → D, D → A} and D = {ABC, CD}

Solution: Lossless Decomposition

Question 5: R (ABCDE) and F = {A → B, B →C, C → D, D → E} and D = {ABC, BCD, CDE}

Solution: Lossless Decomposition

Dependency Preserving Decomposition Examples

Example:1

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

Example:2

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

Example:3

R=(A,B,C,D), F={AB→C, C→D, D→A} and

Decomposition of R: R1(A,B,C),R2(C,D)

Does this decomposition preserve the given dependencies?

Finding Attribute Closure and Candidate Keys

Candidate Key:

- An attribute or the combination of attributes is called candidate key if and only if:
  - They derive all the attributes of the relation.
  - They are the minimal subset of the super key.

Note:

- Candidate keys can be either simple or composite.
- Minimal subset is not with respect to the no of attributes however it always refer to the minimal level of subset which does not have any proper subsets that derives all the attributes of the relation.
- A relation can have more than one candidate key

Prime and non-prime attributes

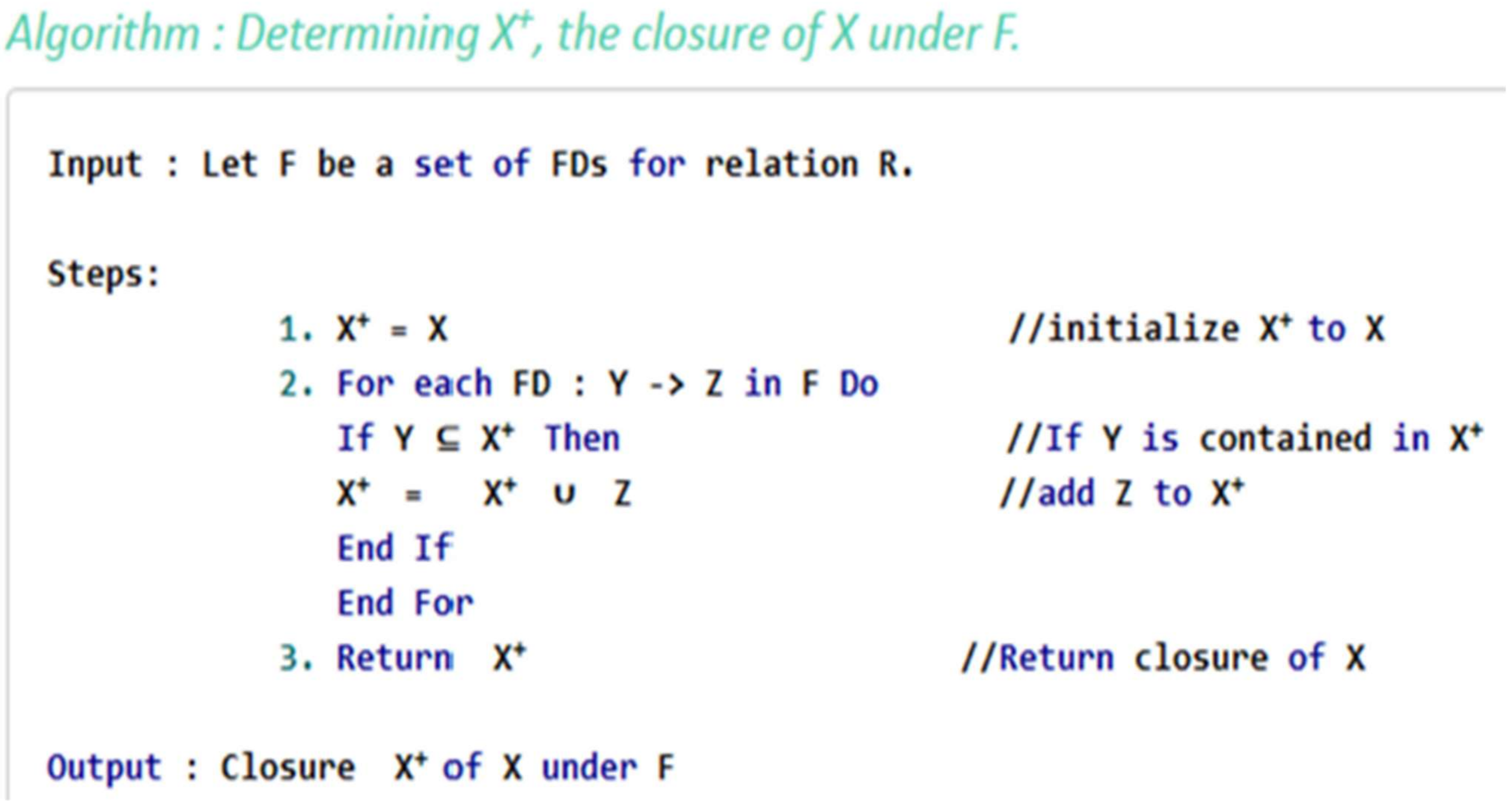
- Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes.

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-
  - Add elements of attribute set to the result set.
  - 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 $Y \subseteq X^+$ 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:

- **Trivial functional dependency**
  - If X→Y is a functional dependency where Y subset X, these type of FD's called as trivial functional dependency.
- **Non-trivial functional dependency**
  - If X→Y and Y is not subset of X then it is called non-trivial functional dependency.
- **Completely non-trivial functional dependency**
  - If X→Y and X∩Y=Φ(null) then it is called completely non-trivial functional dependency.

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

Candidate Key:

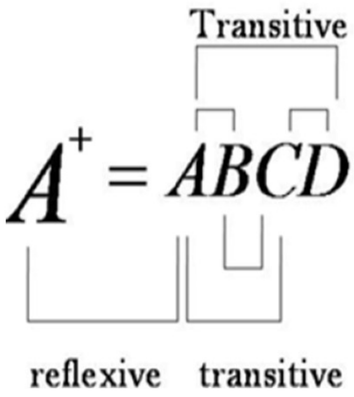
- Candidate Key is minimal set of attributes of a relation which can be used to identify a tuple uniquely.
- Consider student table: student(sno, sname,sphone,age)
  - we can take sno as candidate key. we can have more than 1 candidate key in a table.

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***.

- **Let a relation R (ABCD) and Functional Dependencies are {A → B, B → C, C → D}**

Calculate Attribute A Closure (A^+):


$$A^+ = ABCD$$

Here we drive **A → A** because of **reflexive property**

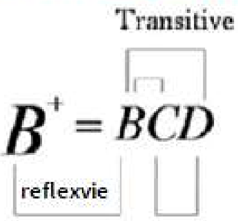
Then we can drive **A → AB** from **FD A → B**

After that we can drive **A → ABC** from **FD B → C**

Then finally we can drive **A → ABCD** from**FD C → D**

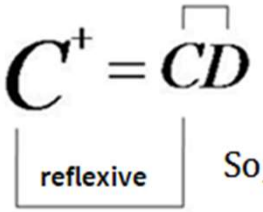
So, we can say **A → ABCD**.

Calculate Attribute B Closure (B^+):


$$B^+ = BCD$$

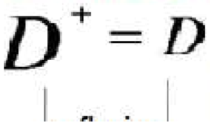
So, we can say **B → BCD**.

Calculate Attribute C Closure (C^+):


$$C^+ = CD$$

So, we can say **C → CD**.

Calculate Attribute D Closure (D^+):


$$D^+ = D$$

So, we can say **D → D**.

Closure of Functional Dependency Example

Question 1:

- Suppose we are given relation R with attributes A, B, C, D and FDs A→BC, B→CD
- Find the Closure of all the attributes

Question 2:

- Compute the closure of the following set F of functional dependencies for relation schema R = {A, B, C, D, E}.
  - A -> BC, CD -> E , B -> D ,E -> A
- List the candidate keys for R.

Question 3 :

- Given a relation R (ABCDEF) having FDs

  {AB→C, C →DE, E →F, C →B}

Identify the prime attributes and non prime attributes