Lecture-14

Functional Dependencies-Finding Minimal Cover

CS211 - Introduction to Database

Good practices in Relational Database Design

- We should find a “good” collection of relation schemas. A bad design may lead to:
    - Repetition of Information.
    - Inability to represent certain information.

- Design Goals:
    - Avoid redundant data
    - Avoid insert, update, and delete anomalies.

Redundancy and Anomalies

Student Relation

Sid is the Primary Key

Sid	Name	Credits	Dept	Building	Room_no	HOD	………..
1	John	5	CS	B1	101	-	
2	Adam	8	CS	B1	101	-	
3	Jiya	9	DS	B2	201	-	
4	Salim	9	DS	B2	201	-	
5	Xi	7	Civil	B1	110	-	
6	Chen	6	EC	B2	115	-	
7	Rahul	8	Civil	B1	120	-	
8	Allan	9	CS	B1	101	-	
NULL	NULL	NULL	ME	B2	120	-	

Redundant Storage

Update Anomaly

Deletion Anomaly

Insertion Anomaly

Decomposition of Relation Schema

- The process of breaking up of a relation into smaller sub-relations is called Decomposition.

- Decomposition **converts a relation into specific normal form** which reduces redundancy, anomalies, and inconsistency in the relation.

- Database normalization is the process of organizing the data into tables in such a way as to remove anomalies and redundancy.

- Decomposition should preserve the following three properties:
    1. Lossless decomposition
    2. Dependency Preservation
    3. Remove redundant functional dependency

Properties of Decomposition

1. **Lossless decomposition:**
    - 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.

2. **Dependency preservation:**
    - None of the functional dependencies that holds on the original relation are lost.
    - The sub-relations still hold or satisfy all the functional dependencies of the original relation.

3. **Remove redundant functional dependency:**
    - All the direct and indirect redundant functional dependencies must be removed.

Lossless decomposition

Difference Between Lossless and Lossy Join Decomposition

Lossless	**Lossy**
The decompositions R1, R2, R2…Rn for a relation schema R are said to be Lossless if there **natural join** results the **original relation** R.	The decompositions R1, R2, R2…Rn for a relation schema R are said to be lossy if there **natural join** results into addition of **extraneous tuples** with the original relation R.

Difference Between Lossless and Lossy Join Decomposition

Lossless	**Lossy**
The decompositions R1, R2, R2…Rn for a relation schema R are said to be Lossless if there **natural join** results the **original relation** R.	The decompositions R1, R2, R2…Rn for a relation schema R are said to be lossy if there **natural join** results into addition of **extraneous tuples** with the original relation R.
Formally, Let R be a relation and R1, R2, R3 … Rn be it's decomposition, the decomposition is lossless if – **R = R1 ⋈ R2 ⋈ R3 …. ⋈ Rn**	Formally, Let R be a relation and R1, R2, R3 … Rn be it's decomposition, the decomposition is lossy if – **R ⊂ R1 ⋈ R2 ⋈ R3 …. ⋈ Rn**

Difference Between Lossless and Lossy Join Decomposition

Lossless	**Lossy**
The decompositions R1, R2, R2…Rn for a relation schema R are said to be Lossless if there **natural join** results the **original relation** R.	The decompositions R1, R2, R2…Rn for a relation schema R are said to be lossy if there **natural join** results into addition of **extraneous tuples** with the original relation R.
Formally, Let R be a relation and R1, R2, R3 … Rn be it's decomposition, the decomposition is lossless if – **$R = R1 \bowtie R2 \bowtie R3 \ldots\ldots \bowtie Rn$**	Formally, Let R be a relation and R1, R2, R3 … Rn be it's decomposition, the decomposition is lossy if – **$R \subset R1 \bowtie R2 \bowtie R3 \ldots\ldots \bowtie Rn$**
The **common attribute** of the sub relations is a **super-key** of any one of the relation.	The **common attribute** of the sub relation is **not a super-key** of any of the sub relation.

Lossy Join Decomposition

- Let there be a relational schema **R(A, B, C)**.
- **R1(A, C)** and **R2(B, C)** be it's decompositions.

R

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

R1

A	C
1	1
2	3
3	3

$\bowtie$

R2

B	C
2	1
5	3
3	3

=

R

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

$R \subseteq R_j$

Lossless Join Decomposition

- Let there be a relational schema **R(A, B, C)**.
- **R1(A, B)** and **R2(B, C)** be it's decompositions.

R

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

R1

A	B
1	2
2	5
3	3

$\bowtie$

R2

B	C
2	1
5	3
3	3

=

R_j

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

$R = R_j$

Properties of a Lossless Join Decomposition

R

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

1. $attr(R1) \cup attr(R2) = attr(R)$
2. $attr(R1) \cap attr(R2) \neq \phi$
3. $attr(R1) \cap attr(R2) \rightarrow attr(R1)$

OR

4. $attr(R1) \cap attr(R2) \rightarrow attr(R2)$

R1

A	B
1	2
2	5
3	3

$\bowtie$

R2

B	C
2	1
5	3
3	3

=

R_j

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

$R = R_j$

Dependency Preservation

Armstrong’s Axioms (A set of inference rules used to infer all the functional dependencies on a relational database)

- A1 Reflexivity rule: $X \rightarrow Y$ if $Y \subseteq X$
- A2 Augmentation rule: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- A3 Transitivity rule: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Armstrong’s Lemmas (Intermediate theorems: *It is possible to use Armstrong’s axioms to prove that these rules are sound*)

- Union rule: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Pseudo Transitivity: if $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$
- Decomposition rule: if $X \rightarrow Y$ and $Z \subseteq Y$, then $X \rightarrow Z$

Regular FD & Closure of FD

- Let F be the set of FDs we have collected.
- A functional dependency $X \rightarrow Y$ is regular **if $\boldsymbol{Y}$ contains only a single attribute**.
- The **closure of $\boldsymbol{F}$**, denoted as $\boldsymbol{F}^+$, is the **set of all regular FDs that can be derived from $\boldsymbol{F}$.**

Inferring functional dependencies

- Given FDs: $X_1 \rightarrow a_1, X_2 \rightarrow a_2, \ldots$

- Does some FD $Y \rightarrow B$ (not given in the above FDs) also hold ?

Example:

Consider the dependencies $\boldsymbol{A} \rightarrow \boldsymbol{B}$ and $\boldsymbol{B} \rightarrow \boldsymbol{C}$

Intuitively, $\boldsymbol{A} \rightarrow \boldsymbol{C}(\boldsymbol{inferred})$ also holds (**A3: transitivity rule**)

Computing F^+ using Armstrong's axioms

- Given 4 attributes A, B, C, D, and F = $\{A \to B, B \to C\}$.

  Compute F^+ :

  - |LHS|=1: $A \to A, A \to B, A \to C, B \to B, B \to C, C \to C, D \to D$
  - |LHS|=2: $AB \to A, AB \to B, AB \to C$ as $(AB \to B$ and $B \to C), AC \to A, AC \to B, AC \to C, AD \to A, AD \to B, AD \to C, AD \to D, BC \to B, BC \to C, BD \to B, BD \to C, BD \to D, CD \to C, CD \to D$
  - |LHS|=3: $ABC \to A, ABC \to B, ABC \to C, ABD \to A, ABD \to B, ABD \to C, ABD \to D, BCD \to B, BCD \to C, BCD \to D$
  - |LHS|=4: $ABCD \to A, ABCD \to B, ABCD \to C, ABCD \to D$

Computing F^+ using Attribute closure α^+

- Given 4 attributes A, B, C, D, and $F = \{A \rightarrow B, B \rightarrow C\}$.

  Compute F^+ :

❑|LHS|=1: $A^+ = ABC, B^+ = BC, C^+ = C, D^+ = D$

❑|LHS|=2: $AB^+ = ABC, AC^+ = ABC, AD^+ = ABCD, BC^+ = BC, BD^+ = BCD, CD^+ = CD$

❑|LHS|=3: $ABC^+ = ABC, ABD^+ = ABCD, BCD^+ = BCD$

$ABD \rightarrow A, ABD \rightarrow B, ABD \rightarrow C \ as \ (ABD \rightarrow B \ and \ B \rightarrow C), ABD \rightarrow D$

❑|LHS|=4: $ABCD^+ = ABCD$

Computing F^+ using Attribute closure α^+

- Given 3 attributes A, B, C and F = $\{A \rightarrow B, B \rightarrow C\}$.

  Compute F^+ :

❑ $|\text{LHS}|=1$: $A^+ = ABC, B^+ = BC, C^+ = C, D^+ = D$

❑ $|\text{LHS}|=2$: $AB^+ = ABC, AC^+ = ABC, AD^+ = ABCD, BC^+ = BC, BD^+ = BCD,$
$CD^+ = CD$

❑ $|\text{LHS}|=3$: $ABC^+ = ABC, ABD^+ = ABCD, BCD^+ = BCD$

$ABD \rightarrow A, ABD \rightarrow B, ABD \rightarrow C \text{ as } (ABD \rightarrow B \text{ and } B \rightarrow C), ABD \rightarrow D$

Generate $\boldsymbol{F}^+$

algorithm (F)

/* $\boldsymbol{F}$ is the set of FDs */

1. $\boldsymbol{F}^+$ = ∅
2. for each possible attribute set $\boldsymbol{\alpha}$
3. compute the closure of $\boldsymbol{\alpha}$ wrt. $\boldsymbol{F}$, i.e. $\boldsymbol{\alpha}^+$
4. for each attribute Z ∈ $\boldsymbol{\alpha}^+$
5.   add the FD: $\boldsymbol{\alpha} \rightarrow Z$ to $\boldsymbol{F}^+$
6. return $\boldsymbol{F}^+$

Example:

if $\boldsymbol{A}^+$ ***= ABC***

then

F $^+$ ***= {A->A, A->B, A->C}***

Closure test

$F: \{AB \rightarrow C,\ A \rightarrow D,\ D \rightarrow E,\ AC \rightarrow B\}$

1. Is $\boldsymbol{AB} \rightarrow \boldsymbol{E}$ in F^+ ?
   - $AB^+ = ABCDE$ So, $AB \rightarrow E$

2. Is $\boldsymbol{D} \rightarrow \boldsymbol{C}$ in F^+ ?
   - $D^+ = DE$ So, $D \rightarrow E$ not true

Example for finding F^+

R = (A, B, C, G, H, I)

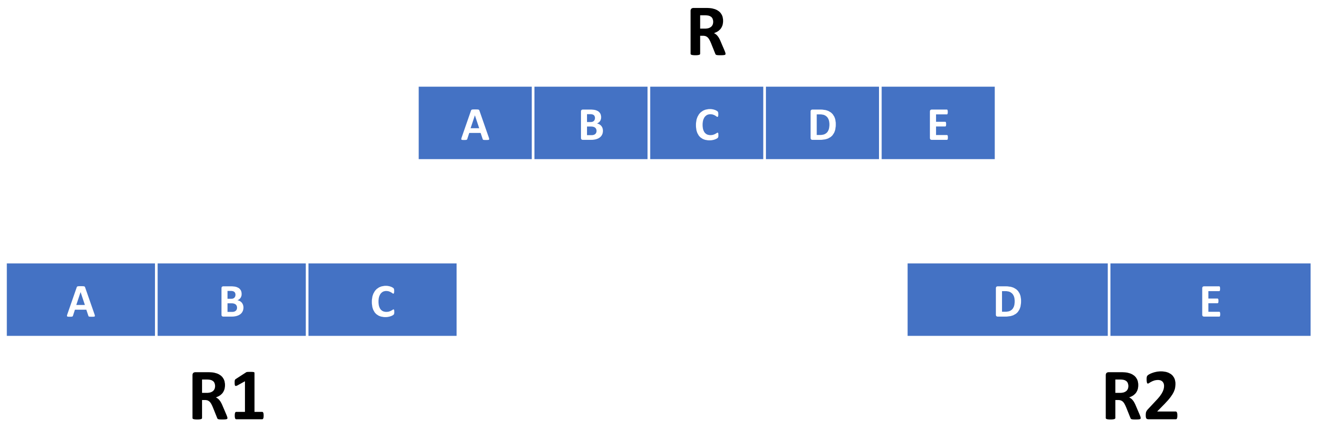
$F = \{ A \rightarrow B$
$A \rightarrow C$
$CG \rightarrow H$
$CG \rightarrow I$
$B \rightarrow H \}$

- some members of F^+
  - $A \rightarrow H$
    - by **transitivity** from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by **augmenting** $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by **union rule**, $CG \rightarrow H$ *and* $CG \rightarrow I$ *then* $CG \rightarrow HI$

Testing for Dependency preservation

- Let F be a set of functional dependencies on a schema R, and let $R_1, R_2, \ldots, R_n$ be a decomposition of R.

- The **restriction** of $\boldsymbol{F}$ to $\boldsymbol{R_i}$ is the set $\boldsymbol{F_i}$ of all functional dependencies in $\boldsymbol{F^+}$ that include *only* attributes of $\boldsymbol{R_i}$.



Restriction of ***F*** to ***R1 is***

{ A->A, A -> B, A -> C, B->B, C->C }

Restriction of ***F*** to ***R2 is***

{ D->D, D->E, E->E }

F = { A -> B , A -> C , D-> E}

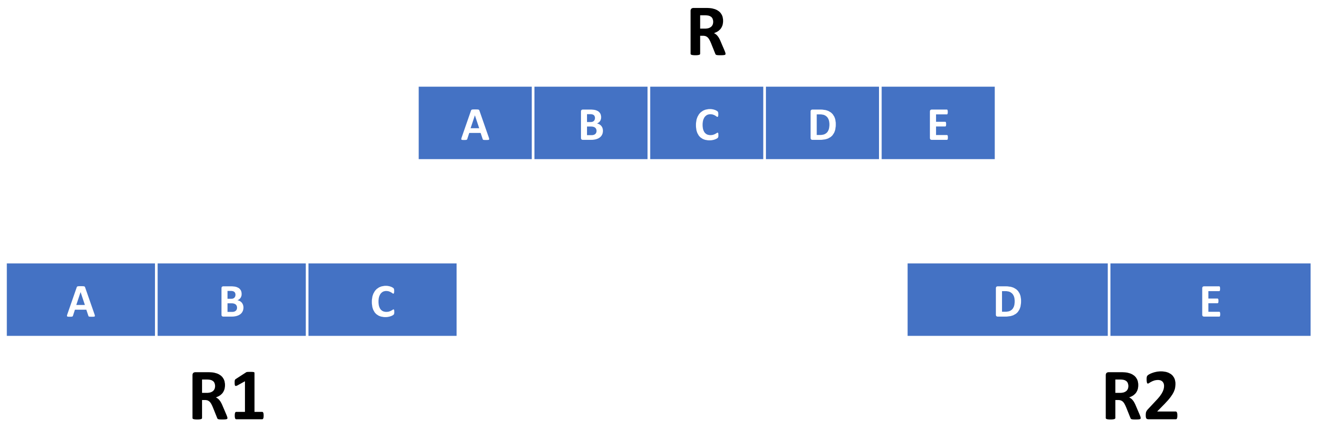
F^+ = {A->A, A->B, A->C, A->D, B->B, C->C, C->D, D->E, …..}

Dependency is preserved

Testing for Dependency preservation

- Let F be a set of functional dependencies on a schema R, and let $R_1, R_2, \ldots, R_n$ be a decomposition of R.

- The **restriction** of $\boldsymbol{F}$ to $\boldsymbol{R_i}$ is the set $\boldsymbol{F_i}$ of all functional dependencies in $\boldsymbol{F}^+$ that include *only* attributes of $\boldsymbol{R_i}$.



Restriction of ***F*** to ***R1 is***

{ A->A, A -> B, A -> C, B->B, C->C }

Restriction of ***F*** to ***R2 is***

{ D->D, D->E, E->E }

Dependency is not preserved

F = { A -> B , A -> C , C -> E, D -> E}

F^+ = {A->B, A->C, A->E, C->E, D->E, …..}

Testing for Dependency preservation

compute F^+;
for each schema R_i in D do
    begin
        $F_i :=$ the restriction of F^+ to R_i;
    end
$F' := \emptyset$
for each restriction F_i do
    begin
        $F' = F' \cup F_i$
    end
compute F'^+;
if ($F'^+ = F^+$) then return (true)
    else return (false);

R(A,B,C,D) and **F = { A -> B, A -> C, D -> E}**

Relation R is decomposed into following sub-relations with FDs defined on them:

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

R_2 = (D, E) with FDs **F_2 = { D->D, D->E, E->E }**

Dependency is preserved

Testing for Dependency preservation

compute F^+;
for each schema R_i in D do
  begin
    $F_i := $ the restriction of F^+ to R_i;
  end
$F' := \emptyset$
for each restriction F_i do
  begin
    $F' = F' \cup F_i$
  end
compute F'^+;
if $(F'^+ = F^+)$ then return (true)
  else return (false);

R(A,B,C,D) and **F = { A -> B, A -> C, C->E, D -> E}**

Relation R is decomposed into following sub-relations with FDs defined on them:

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

R_2 = (D, E) with FDs **F_2 = { D->D, D->E, E->E }**

Dependency is not preserved

Dependency preservation test

- Let a relation R(A,B,C,D) and F = { A -> B , A -> C , C -> D}.

- Relation R is decomposed into following sub-relations with FDs defined on them:
    1. R_1 = (A, B) with FDs F_1 = {A -> B}
    2. R_2 = (C, D) with FDs F_2 = {C -> D}

Let F' = $F_1 \cup F_2$

= {A -> B, C -> D}

so, F' $\neq$ F

so, $F'^{+} \neq F^{+}$

Whenever $F'^{+} \neq F^{+}$, the original FDs are not preserved

Dependency preservation test

- Let a relation R(A,B,C,D) and F = { A -> B , A -> C , C -> D}.

- Relation R is decomposed into following sub-relations with FDs defined on them:
  1. R_1 = (A, B, C) with FDs F_1 = {A -> B, A -> C}
  2. R_2 = (C, D) with FDs F_2 = {C -> D}

Let F' = $F_1 \cup F_2$

= {A -> B, A -> C, C -> D}

so, F' = F

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

Whenever $F'^+ = F^+$, all the original FDs are preserved

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- **Testing for superkey:**
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^{+} contains all attributes of ***R***.

- **Testing functional dependencies:**
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in ***F***$^{+}$), just check if $\beta \subseteq \alpha^{+}$.
  - That is, we compute α^{+} by using attribute closure, and then check if it contains β.

- **Computing closure of F:**
  - For each possible attribute set $\boldsymbol{\alpha}$*,* we find the closure $\boldsymbol{\alpha}^{+}$, and for each $\boldsymbol{S} \subseteq \boldsymbol{\alpha}^{+}$, we output a functional dependency $\boldsymbol{\alpha} \rightarrow \boldsymbol{S}$*.*

Canonical Cover
(To remove redundant FDs)

Canonical Cover F_c

- Suppose that we have a set of functional dependencies ***F*** on a relation schema.
- Whenever a user **performs an update on the relation**, the database system must ensure that the update **does not violate any functional dependencies**.
- If an update violates any functional dependencies in the set *F*, the system must **roll back** the update.
- We can reduce the effort spent in checking for violations by testing a simplified set of functional dependencies that has the same closure as the given set ***F*****.**
- This simplified set is termed the **Canonical cover.**

Extraneous Attribute

- To define canonical cover we must first define **Extraneous attributes.**
  - Assume a set of functional dependencies ***F***, and the closure of set of functional dependencies F^+.
  - Also, assume that **we remove an attribute** from any of the FDs under ***F*** and find the closure of new set of functional dependencies as $F1^+$.

If $F1^+ = F^+$ then the attribute which has been removed is called as **Extraneous Attribute**

Example:

- In F={AB→C, A→C}, **B is extraneous** in *LHS* AB→C.
  - When *A* can determine *C* alone, what is the use of extra attribute of *B* in *AB→C* ???
- In F={A→BC, B→C}, **C is extraneous** in *RHS* A→BC.
  - When *A* can determine *C* from the transitive rule, what is the use of extra attribute of *C* in A→BC ???

Finding Extraneous attributes

Let *R* be a relation schema and let *F* be a set of functional dependencies that hold on *R* . Consider an attribute in the functional dependency $\alpha \rightarrow \beta$.

1. **To test if attribute E $\in \alpha$ is extraneous in α**
   - Let $\gamma = \alpha - \{E\}$. Check if $\gamma \rightarrow \beta$ can be inferred from *F.* To do so,
     a) Compute γ^+ using the dependencies in *F*
     b) If γ^+ includes all attributes in β then , *E* is extraneous in α

***R* = (*A, B, C*) *F* = {*A* → *BC, B* → *C, A* → *B, AB* → *C*}**

A is extraneous in *AB* → *C*

C is extraneous in *A* → *BC*

New *F* = {*A* → *BC, B* → *C, A* → *B*}

Finding Extraneous attributes

2. **To test if attribute $E \in \beta$ is extraneous in β**
   - Consider the set:

     $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - E)\}$$

   - Compute α^+ under F'. If α^+ contains *E,* then *E* is extraneous in β

$R = (A, B, C)$ $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

A is extraneous in $AB \rightarrow C$

C is extraneous in $A \rightarrow BC$

New $F = \{B \rightarrow C, A \rightarrow B\}$

Definition of Canonical Cover F_c

- A **canonical cover** for F is a set of dependencies F_c such that:
  - F logically implies all dependencies in F_c , and
  - F_c logically implies all dependencies in F, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is *unique*. That is, there are no two dependencies in F_c :
    - $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that $\alpha_1 = \alpha_2$
    - In such case, combine the dependencies into $\alpha_1 \rightarrow \beta_1 \beta_2$

Algorithm for computing Canonical Cover F_c

F_C = F

Repeat

1. Use the union rule to replace any dependencies in F_c of the form

   $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

2. Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

   /* Note: test for extraneous attributes done using F_c, not F */

3. If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until (F_c not change)

/* Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied */

Example for computing Canonical Cover F_c

$R = (A, B, C)$ and $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

- **F_c = F**
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - **New F_c** is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - $B \rightarrow C$ is already present!
  - **New F_c** is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
  - **New F_c** is now $\{A \rightarrow B, B \rightarrow C\}$
- The canonical cover **F_c** is: $A \rightarrow B$
  $B \rightarrow C$

Note: F logically implies all dependencies in F_c .

Note: F_c logically implies all dependencies in F.

1. A->B
2. B->C
3. A->B ➔ AB->BB ➔AB->B and B->C =➔ AB->C
4. B->C ➔BB->BC➔B->BC and A->B➔A->BC

Minimal cover / Irreducible set of FD

Minimal Cover

- **Cover**
  - F covers another set of functional dependencies G, if every functional dependency in G can be inferred from F.
  - More formally, $\boldsymbol{F}$ **covers** $\boldsymbol{G}$ if $\boldsymbol{G}^+ \subseteq \boldsymbol{F}^+$.

- Given a set of FDs $\boldsymbol{F}$, its **minimal cover** $\boldsymbol{F}'$ is the **smallest** set of functional dependencies that covers F.

Properties of a minimal cover *F'*

- All FD in ***F'*** are regular FD (A functional dependency $X \rightarrow Y$ is **regular** if Y contains only a single attribute).

- If any FD is removed from F', F' is no longer a minimal cover.

- If, for any FD in F' we remove one or more attributes from the LHS of *F*, the result is no longer a minimal cover.

A canonical cover is "allowed" to have more than one attribute on the right hand side. A minimal cover cannot. As an example, the canonical cover may be "A -> BC" where the minimal cover would be "A -> B, A -> C". That is the only difference.

Quick manual method for finding Minimal cover F_C if all FDs contain only single attributes in both LHS & RHS

Procedure :

Given : $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, C \rightarrow A, B \rightarrow C \}$

1. For the first FD $A \rightarrow B$ find A^+ by **hiding** the FD $A \rightarrow B$. We get $A^+ = AC$ which does not include B in it. **Hence, *A* → *B* is not redundant**.

2. For the second FD $B \rightarrow A$ find B^+ by hiding the FD $B \rightarrow A$. We get $B^+ = ABC$ which includes A in it. **Hence, *B* → *A* is redundant. We have to remove it immediately**.

   - Now with the removal of $B \rightarrow A$, our F_C becomes:

     $F_c = \{ A \rightarrow B, A \rightarrow C, C \rightarrow A, B \rightarrow C \}$

Procedure :

$F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, C \rightarrow A, B \rightarrow C \}$, $F_C = \{ A \rightarrow B, A \rightarrow C, C \rightarrow A, B \rightarrow C \}$

3. For the third FD $A \rightarrow C$, find A^+ by hiding $A \rightarrow C$. We get $A^+ = ABC$ which includes C. **Hence $A \rightarrow C$ is redundant and remove it immediately from F**.
   - After removal of $A \rightarrow C$, our F_C becomes:

     $F_C = \{ A \rightarrow B, C \rightarrow A, B \rightarrow C \}$

4. For the forth FD $C \rightarrow A$, find C^+ by hiding $C \rightarrow A$. We get $C^+ = C$ and this does not include A in the result. **Hence, $C \rightarrow A$ is not redundant**.

5. For the last FD $B \rightarrow C$, find find B^+ by hiding $B \rightarrow C$. We get $B^+ = B$. **Hence, $B \rightarrow C$ is not redundant**.

Our final set of functional dependencies are minimal after the removal of FDs $\boldsymbol{B \rightarrow A}$ and $\boldsymbol{A \rightarrow C}$. Hence, the minimal cover of F is **$F_c = \{ A \rightarrow B, C \rightarrow A, B \rightarrow C \}$**

Practice-1

- Relation $R(A, B, C)$
- F={$A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C, C \rightarrow A$}
- The minimal cover of F is **{A→B, B→C, C→A}**

Practice-2

- Relation $R(A, B, C, D)$
- F={$A \rightarrow AC, B \rightarrow ABC, D \rightarrow ABC$}
- The minimal cover of F is **{A→C, B→A, D→B}**