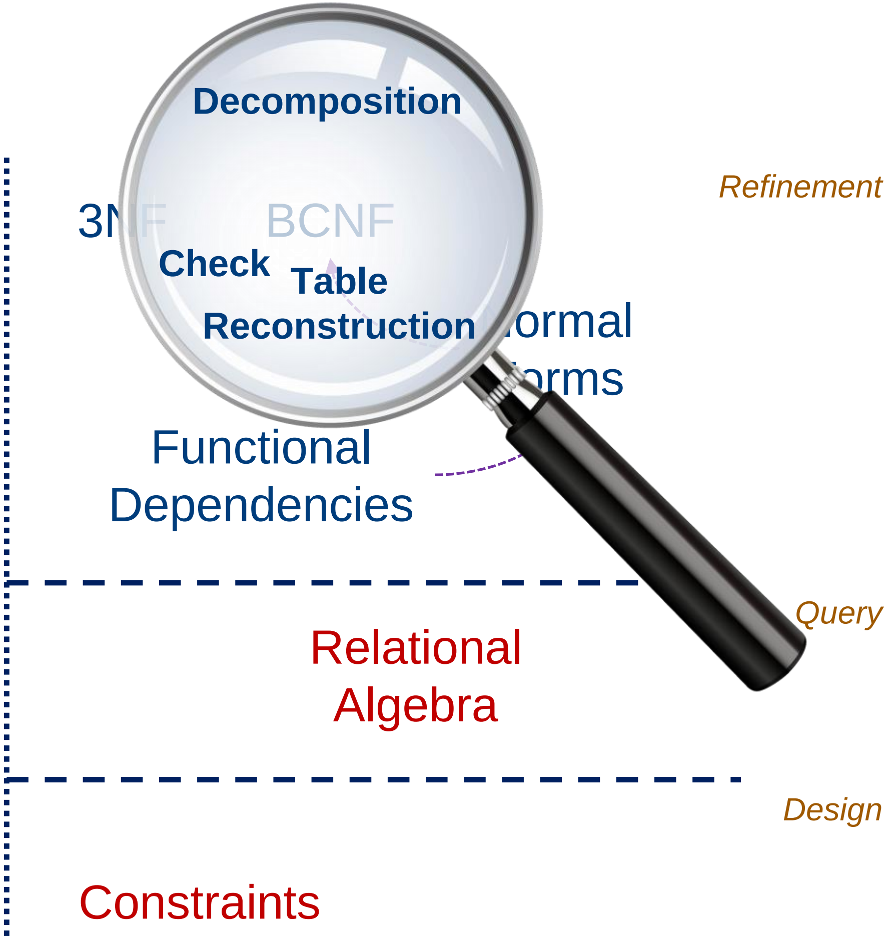
National University of Singapore | NUS Computing

CS2102 Database Systems

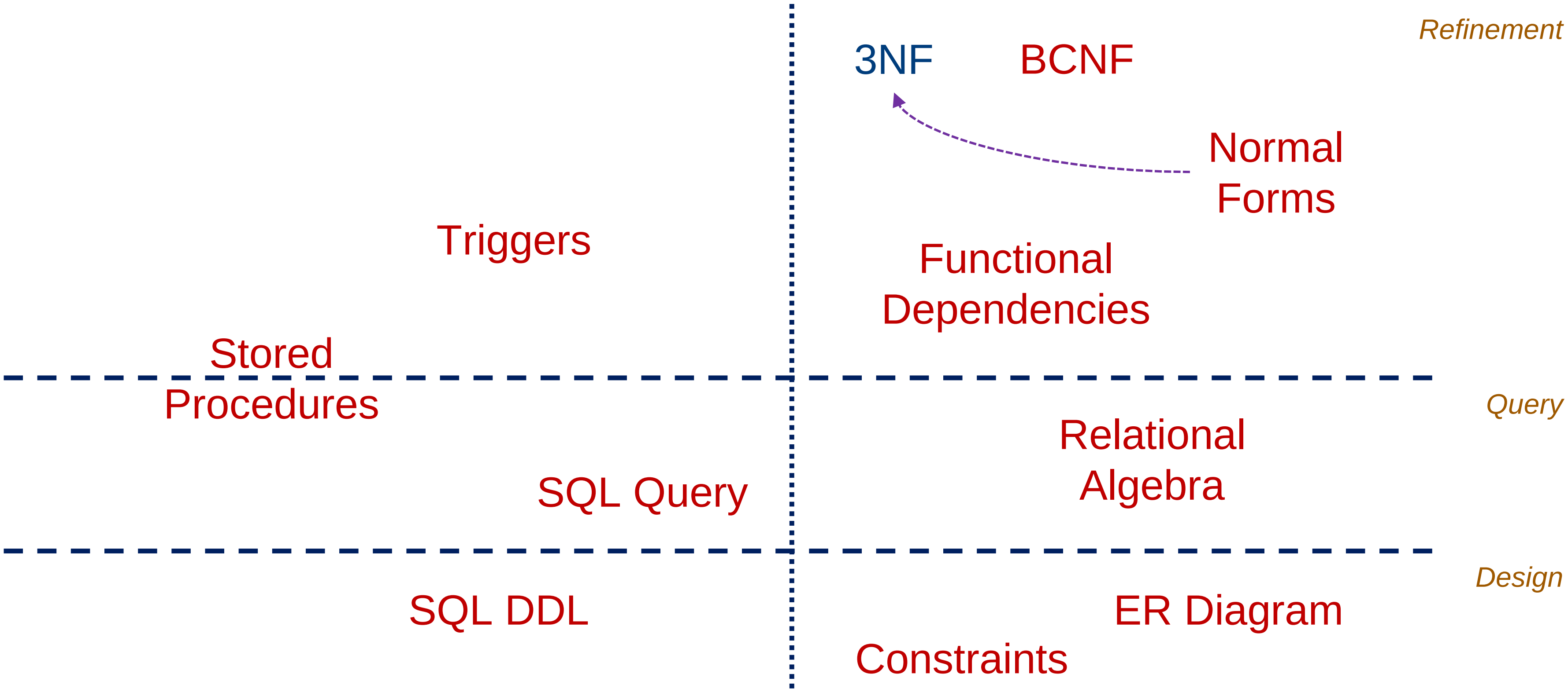
Lecture 12 – Third Normal Form

Roadmap

- **Previously**
  - **BCNF Definition:**
    - Every *non-trivial & decomposed* FD has *superkey* as LHS
  - **BCNF Check:**
    - Find violation: *"more but not all"*
  - **BCNF Decomposition:**
    - For violation $A \rightarrow A^+$
      - `R1(`A^+`)`
      - `R2(`A `∪ {R -` A^+`})`
    - *Repeat* until all tables are in BCNF
  - BCNF Properties:
    - ✓ No update or deletion anomalies
    - ✓ Small redundancies
    - ✓ Original table can be reconstructed
    - ✗ *Dependencies may not be preserved in the decomposed table*



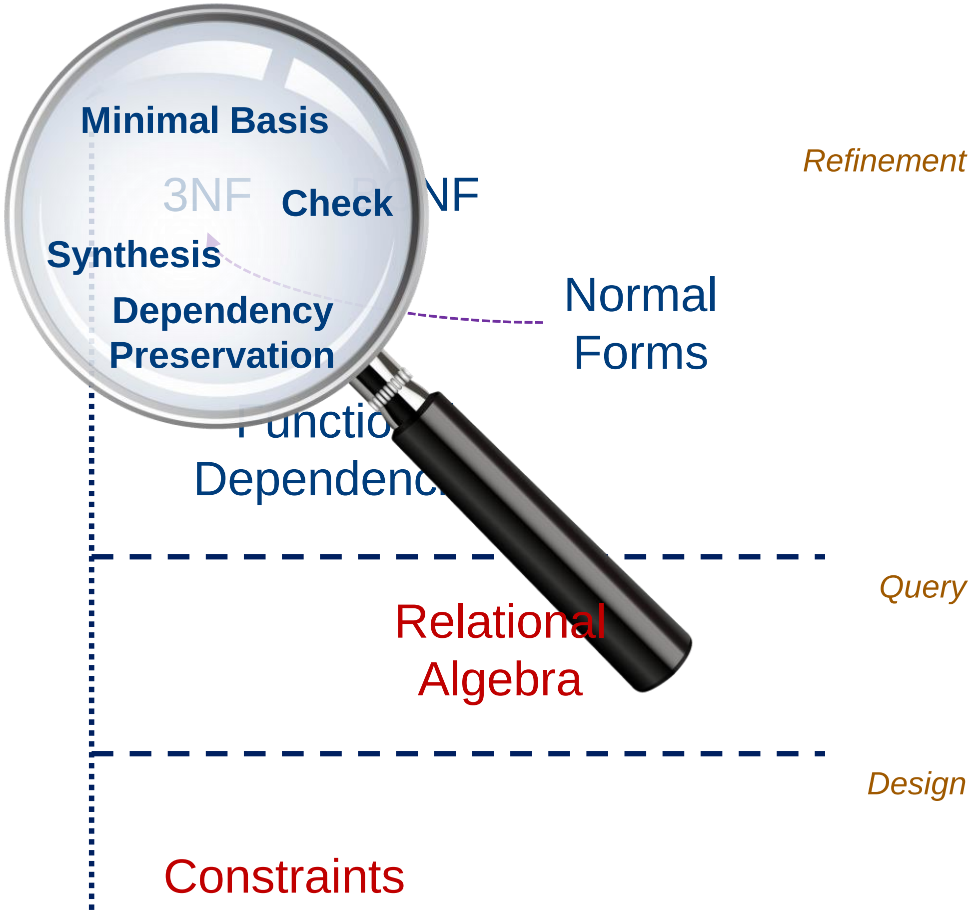
Roadmap



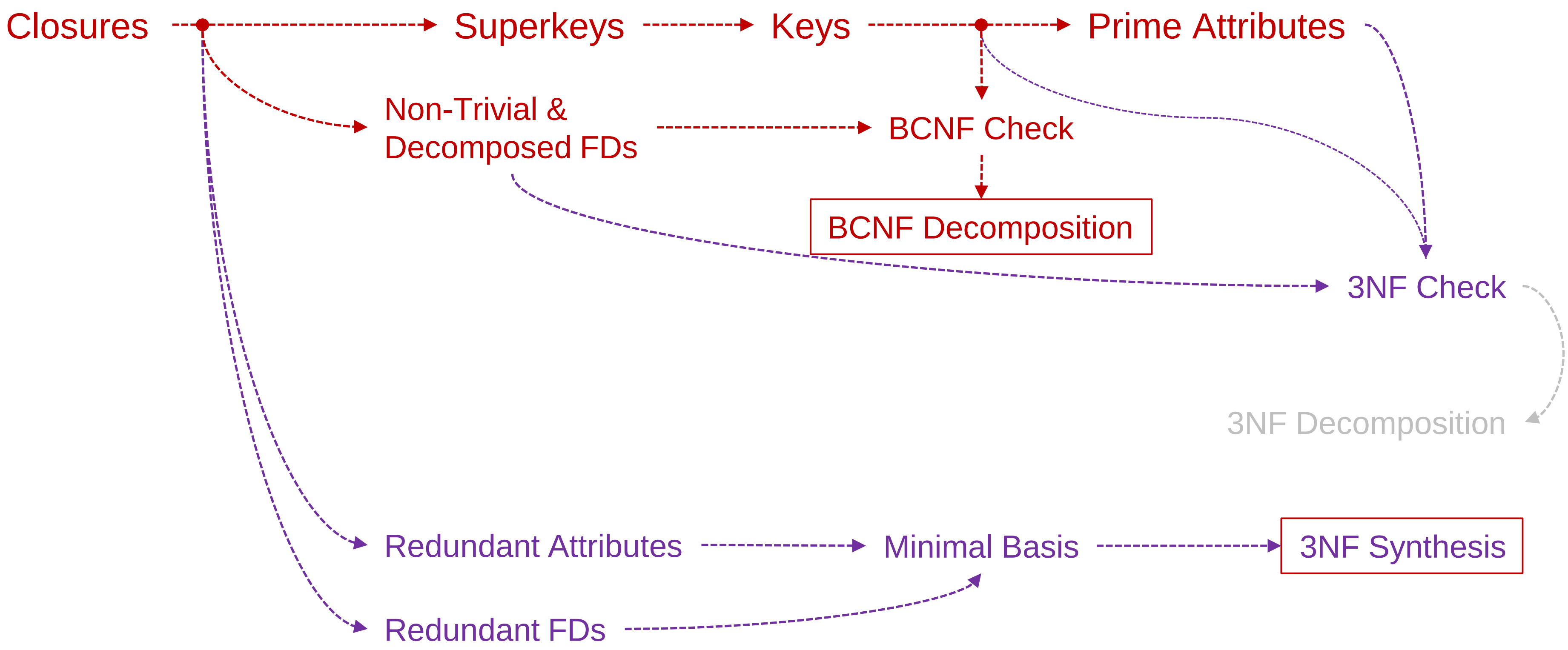
Refinement
3NF
BCNF
Normal Forms
Triggers
Functional Dependencies
Stored Procedures
Query
Relational Algebra
SQL Query
Design
SQL DDL
ER Diagram
Constraints

Roadmap

- We will do this step by step
  - Dependency Preservation
    (why we need 3NF)
  - Define 3NF
  - Check 3NF
  - Minimal Basis
  - 3NF Synthesis



Algorithm Roadmap



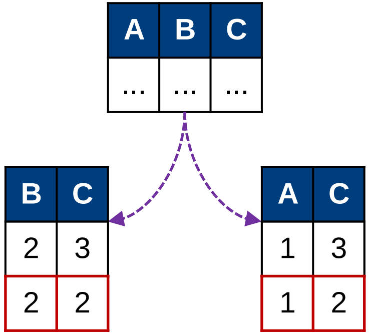
Dependency Preservation

or why we need 3NF

Dependency Preservation → Define 3NF and Check → Minimal Basis → 3NF Synthesis

Dependency Preservation

- **Motivating Example**
  - Table `R(A, B, C)` with `AB → C` and `C → B`
  - `{C} → {B, C}` violates BCNF of `R(A, B, C)`
    - `R1(B, C)` *(non-trivial & decomposed FD on R1: C → B)*
    - `R2(A, C)` *(non-trivial & decomposed FD on R2: `nothing`)*
    - *Where is `AB → C`??*
      - *Cannot even be derived from FDs on `R1` and `R2`*
      - *Totally "lost"* when a functional dependency is lost, then it is possible to insert values that violates the functional dependency

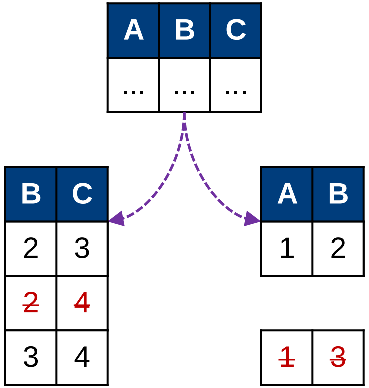


❖ This is why we say that a BCNF decomposition may not always preserve all FDs *(non dependency preserving decomposition)*
  - ❖ *Dilemma!*
    - ❖ *either the table has anomalies OR does not preserve constraints!*

Dependency Preservation

- "***What We Want***" **Example**
  - Table `R(A, B, C)` with `A → B`, `B → C` and `A → C`
  - `{B} → {B, C}` violates BCNF of `R(A, B, C)`
    - `R1(B, C)` *(non-trivial & decomposed FD on R1: B → C)*
    - `R2(A, B)` *(non-trivial & decomposed FD on R2: A → B)*

    - *Where is* `A → C`*??*
      - *Can be derived from FDs on* `R1` *and* `R2`
      - Compute $\{A\}^+$ w.r.t. `{B → C, A → B} = {A, B, C}`
        - So this FD is *preserved* even when it spans across multiple tables



❖ No valid insertion into `R1` or `R2` can violates this functional dependencies

❖ Gives rise to the notion of *FD equivalence*

Functional Dependency Equivalence

- **Definition**

essentially just using closure to ensure can derive one from the other

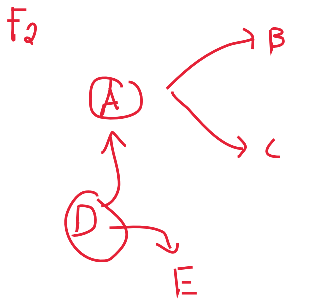
  - Let F1 and F2 be sets of FDs
  - We say that F1 is *equivalent* to F2 *(i.e., F1 ≡ F2)* if and only if
    - Every FD in F1 can be derived from F2 *(i.e., F2 ⊢ F1)*
    - Every FD in F2 can be derived from F1 *(i.e., F1 ⊢ F2)*
    - *They can look different, but they carry the same information*

- In the context of decomposition, we can let F2 be the FD from the decomposed table
  - How do we get this FD from decomposed table?
  - *Union of projection*

Functional Dependency Equivalence

best way is to solve with a hyper graph

- **Example**
  - **Example:** R(A, B, C, D, E)
    - F1 = {A → B, AB → C, D → AC, D → E}
    - F2 = {A → BC, D → AE}
    - ❖ Show F1 ≡ F2



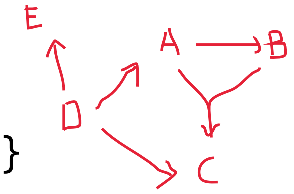
1. Prove that F1 can be derived from F2
    - A → B and D → E can be derived easily using *decomposition rule*
    - $\{AB\}^+ = \{ABC\}$ so AB → C is implied by F2
    - $\{D\}^+ = \{ABCDE\}$ so D → AC is implied by F2

∴ F1 can be derived from F2

Functional Dependency Equivalence

to prove equivalence, need to show both directions

- **Example**



  - **Example:** R(A, B, C, D, E)
    - F1 = {A → B, AB → C, D → AC, D → E}
    - F2 = {A → BC, D → AE}
    - ❖ Show F1 ≡ F2

2. Prove that F2 can be derived from F1
    - $\{A\}^+ = \{ABC\}$ so A → BC is implied by F1
    - $\{D\}^+ = \{ABCDE\}$ so D → AE is implied by F1

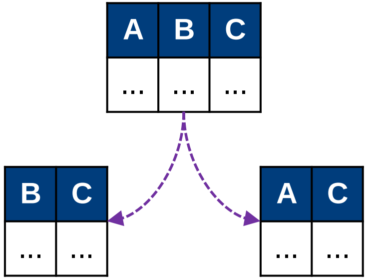
∴ F2 can be derived from F1

Dependency Preservation

- **Motivating Example**
  - Table `R(A, B, C)` with `AB → C` and `C → B`
  - `{C} → {B, C}` violates BCNF of `R(A, B, C)`
    - `R1(B, C)` *(non-trivial & decomposed FD on R1: `C → B`)*
    - `R2(A, C)` *(non-trivial & decomposed FD on R2: `nothing`)*

    - `F1 = {AB → C, C → B}`
    - `FR1 = {C → B}, FR2 = {}`
      - `F2 = {C → B}`

  ❖ We can show that `F1 ≢ F2`
  - Can we still enforce `AB → C` on the decomposed table?
  - YES! But not easy, need to use trigger!



3NF

Third Normal Form

Dependency Preservation → Define 3NF and Check → Minimal Basis → 3NF Synthesis

3NF

if not decomposed FD then RHS is a subset of prime attributes

- **Definition**
  - A table R is in 3NF if *every non-trivial and decomposed FD either:*
    - *its left hand side is a superkey* BCNF condition only
    - *the right hand side is a prime attributes* *(appears in a key)*
  - **Example:** R(A, B, C) with C → B, AC → B and AB → C
    - Key: {AB} and {AC}
    - Prime Attributes: {ABC}
    - Non-trivial and decomposed FDs on R:
      - C → B
        - B is PA
      - AB → C
        - AB is SK
      - AC → B
        - AC is SK

    ∴ R satisfies 3NF

3NF

another definition: all attribute does not transitively depend on a key

- ## Definition
  - A table R is in 3NF if *every non-trivial and decomposed FD either:*
    - *its left hand side is a superkey*
    - *the right hand side is a prime attributes* *(appears in a key)*

  - **Example:** R(A, B, C) with $A \rightarrow B$, $B \rightarrow C$, $AC \rightarrow B$ and $AB \rightarrow C$
    - Key: {A}
    - Prime Attributes: {A}
    - Non-trivial and decomposed FDs on R:
      - $A \rightarrow B$
        - A is SK
      - $B \rightarrow C$
        - B is **NOT** SK
        - C is **NOT** PA

    ∴ R violates 3NF

3NF vs BCNF

- **BCNF:** for any non-trivial and decomposed FD
  - The left hand side is a superkey

"Every attribute must depend ONLY on superkeys!"

NO EXCEPTION

- **3NF:** for any non-trivial and decomposed FD
  - The left hand side is a superkey
  - OR, the right hand side is a prime attribute

"Exceptions can be made for prime attributes"

3NF vs BCNF

- **BCNF:** for any non-trivial and decomposed FD
  - The left hand side is a superkey

- **3NF:** for any non-trivial and decomposed FD
  - The left hand side is a superkey
  - OR, the right hand side is a prime attribute

- **3NF** is more "*lenient*" than **BCNF**
  - Therefore
    - Satisfying BCNF ⇒ satisfying 3NF but not necessarily vice versa
    - Violating 3NF ⇒ violating BCNF but not necessarily vice versa

3NF

- **Checking 3NF**
  - A table R is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*
  - **By Counterexample:**
    1. Consider all non-trivial and decomposed FDs of R
    2. For each non-trivial and decomposed FDs of R, check that
       a. the left hand side is superkey
       b. the right hand side is in prime attributes
       - If not one of them, then we have a counterexample
    3. If no counterexample found, then R satisfies 3NF

More but NOT All
$\{S\} \subset \{S\}^+ \subset R$

3NF

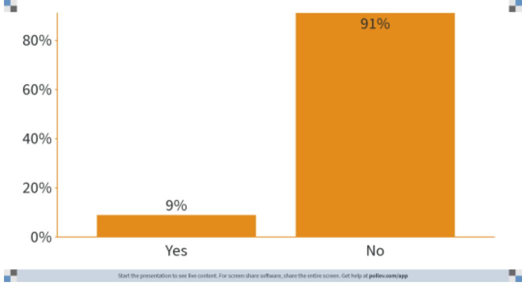
- ## Checking 3NF
  - A table `R` is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*
  - **By Counterexample:**
    1. Consider all subset of attributes of `R`
    2. Compute the closure of each subset
    3. Consider only the subset that are not superkey *(closure ≠ `R`)*
    4. Remove the “trivial” attributes
    5. We have a counterexample, if
       a. The resulting set is non-empty, **and**
       b. There is an attribute in the right hand side that is not in the left hand side and not a prime attribute

3NF

- ## Checking 3NF
  - **Example:** R(A, B, C, D) with $AB \rightarrow C$, $C \rightarrow D$ and $D \rightarrow A$
    1. Consider all subset of attributes of R
    2. Compute the closure of each subset
    3. Consider only the subset that are not superkey *(closure ≠ R)*
    4. Remove the “trivial” attributes
    5. We have a counterexample, if
        a. The resulting set is non-empty, **and**
        b. There is an attribute in the right hand side that is not in the left hand side and not a prime attribute
  - Keys = {AB}, {BC}, {BD} Prime attributes = {ABCD}

$\{A\}^+ = \{A\}$	$\{B\}^+ = \{B\}$	$\{C\}^+ = \{ACD\}$	$\{D\}^+ = \{AD\}$
$\{AB\}^+ = \{ABCD\}$	$\{AC\}^+ = \{ACD\}$	$\{AD\}^+ = \{AD\}$	
$\{BC\}^+ = \{ABCD\}$	$\{BD\}^+ = \{ABCD\}$	$\{CD\}^+ = \{ACD\}$	
$\{ABC\}^+ = \{ABCD\}$	$\{ABD\}^+ = \{ABCD\}$	$\{ACD\}^+ = \{ACD\}$	$\{BCD\}^+ = \{ABCD\}$

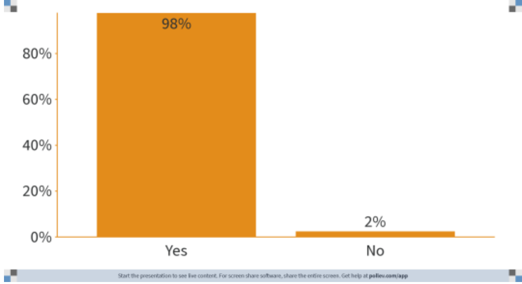
3NF



- **Checking 3NF**
  - A table `R` is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*
  - **Exercise:** `R(A, B, C, D)` with `B → C` and `B → D`
    - Keys = `{AB}` Prime attributes = `{AB}`

$\{A\}^+ = \{A\}$	$\{B\}^+ = \{BCD\}$	$\{C\}^+ = \{C\}$	$\{D\}^+ = \{D\}$
$\{AB\}^+ = \{ABCD\}$	$\{AC\}^+ = \{AC\}$	$\{AD\}^+ = \{AD\}$	
$\{BC\}^+ = \{BCD\}$	$\{BD\}^+ = \{BCD\}$	$\{CD\}^+ = \{CD\}$	
$\{ABC\}^+ = \{ABCD\}$	$\{ABD\}^+ = \{ABCD\}$	$\{ACD\}^+ = \{ACD\}$	$\{BCD\}^+ = \{BCD\}$

3NF

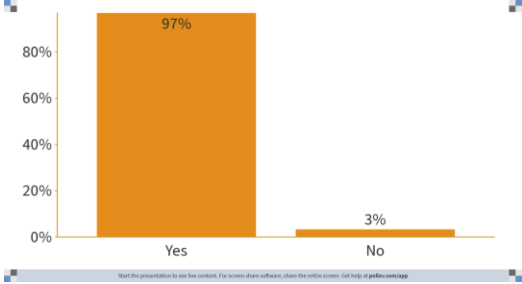


- ## Checking 3NF
  - A table R is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*

  - **Exercise:** R(A, B, C, D) with A → B, B → C, C → D and D → A
    - Keys = {A}, {B}, {C}, {D}　　　Prime attributes = {ABCD}

$\{A\}^+$ = {ABCD}	$\{B\}^+$ = {ABCD}	$\{C\}^+$ = {ABCD}	$\{D\}^+$ = {ABCD}
$\{AB\}^+$ = {ABCD}	$\{AC\}^+$ = {ABCD}	$\{AD\}^+$ = {ABCD}	
$\{BC\}^+$ = {ABCD}	$\{BD\}^+$ = {ABCD}	$\{CD\}^+$ = {ABCD}	
$\{ABC\}^+$ = {ABCD}	$\{ABD\}^+$ = {ABCD}	$\{ACD\}^+$ = {ABCD}	$\{BCD\}^+$ = {ABCD}

3NF

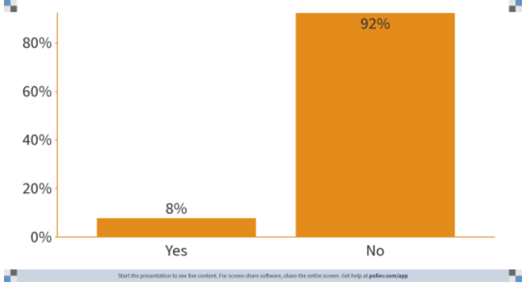


- ## Checking 3NF
  - A table R is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*

  - **Exercise:** R(A, B, C, D) with AB → D, BD → C, CD → A and AC → B
    - Keys = {AB}, {AC}, {BD}, {CD} Prime attributes = {ABCD}

$\{A\}^+$	=	{A}	$\{B\}^+$	=	{B}	$\{C\}^+$	=	{C}	$\{D\}^+$	=	{D}
$\{AB\}^+$	=	{ABCD}	$\{AC\}^+$	=	{ABCD}	$\{AD\}^+$	=	{AD}			
$\{BC\}^+$	=	{BC}	$\{BD\}^+$	=	{ABCD}	$\{CD\}^+$	=	{ABCD}			
$\{ABC\}^+$	=	{ABCD}	$\{ABD\}^+$	=	{ABCD}	$\{ACD\}^+$	=	{ABCD}	$\{BCD\}^+$	=	{ABCD}

3NF



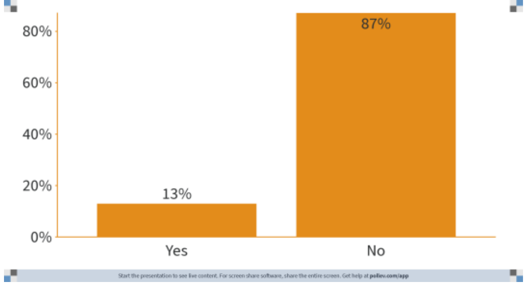
- **Checking 3NF**
  - A table R is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*
  - **Exercise:** R(A, B, C, D, E) with AB → C, C → E, E → A and E → D
    - Keys = {AB} , {BC} , {BE} Prime attributes = {ABCE}

$\{A\}^+ = \{A\}$ | $\{B\}^+ = \{B\}$ | $\{C\}^+ = \{ACDE\}$

Sometimes the violation is obvious like this E → D

Or you can always start from smallest subset of attributes and try to find violation

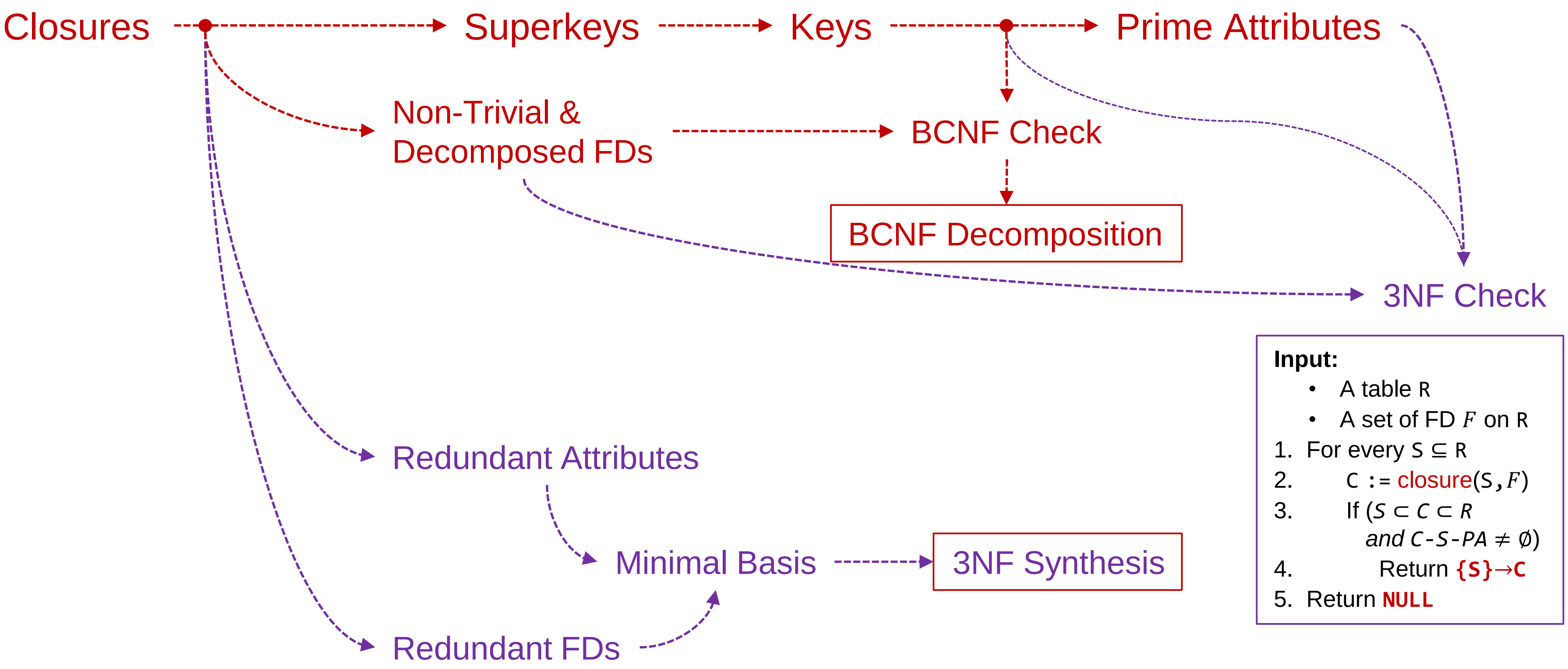
3NF



- **Checking 3NF**
  - A table `R` is in **NOT** 3NF if *every non-trivial and decomposed FD both:*
    - *its left hand side is NOT a superkey*
    - *the right hand side is NOT a prime attributes* *(does not appear in any key)*
  - **Exercise:** `R(A, B, C, D, E)` with AB → C, DE → C and B → E
    - Keys = {ABD}   Prime attributes = {ABD}

$\{A\}^+ = \{A\}$ | $\{B\}^+ = \{BE\}$

Algorithm Roadmap



Input:

- A table R
- A set of FD F on R

1. For every S ⊆ R
2.   C := closure(S, F)
3.   If ($S \subset C \subset R$ *and* $C\text{-}S\text{-}PA \neq \emptyset$)
4.   Return **{S}→C**
5. Return **NULL**

3NF Decomposition

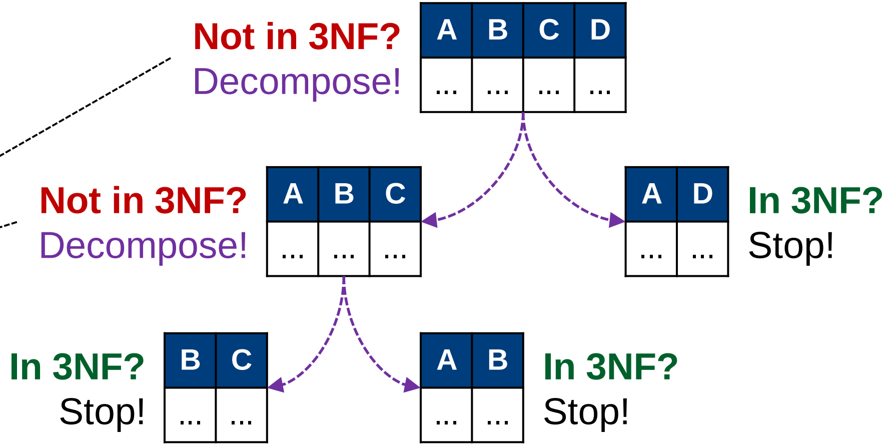
Dependency Preservation → Define 3NF and Check → Minimal Basis → 3NF Synthesis

3NF Decomposition

- **Normalization**
  - **Same Idea as BCNF Decomposition:**
    - Same potential problem of non dependency preserving decomposition
    - Is there a better idea?

Input:
- A table R
- A set of FD F on R

1. For every $S \subseteq R$
2. C := closure(S, F)
3. If ($S \subset C \subset R$ *and* $C - S - PA \neq \emptyset$)
4. Return **{S}→C**
5. Return **NULL**



Minimal Basis

on our path to 3NF synthesis

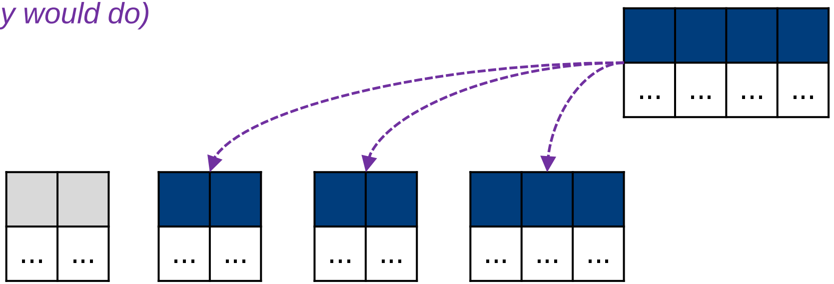
Dependency Preservation → Define 3NF and Check → Minimal Basis → 3NF Synthesis

Minimal Basis

$\{AB \rightarrow C,\ C \rightarrow DE\}$ → R1(A, B, C)
R2(C, D, E)

- **A Sneak Peek**
  - **3NF Synthesis**
    - **Input:** Table `R` and a set of FDs `F`

1. Derive minimal basis of `F`
2. From the minimal basis, combine the FDs for which the left hand sides are the same *(union rule, producing what's called as canonical cover)*
3. Create a table for each FDs remained in the minimal basis after union
4. If none of the tables contains a key of the original table R, create a table that contains a key of `R` *(any key would do)*



Minimal Basis

- **Definition**
  - Given a set F of FDs, the minimal basis of F is a *simplified* version of F
    - Also called minimal cover *(let's call this F_b)*
  - How simplified?
    - Four conditions
      1. Every FD in F_b can be derived from F and vice versa
      2. Every FD in F_b are non-trivial and decomposed FD
      3. For each FD in F_b, none of the attributes on the left hand side is redundant *(no redundant attributes)*
      4. No FD in F_b are redundant *(no redundant FD)*

❖ *Redundant means that we can remove them (attributes or FDs) without affecting the original FD (i.e., still equivalent)*

Minimal Basis

- **Definition**
  - Given a set F of FDs, the minimal basis of F is a *simplified* version of F
    - Also called minimal cover *(let's call this F_b)*
  - How simplified?
    - Four conditions

1. Every FD in F_b can be derived from F and vice versa

```
-   F  = {A → BD,  AB → C,  C → D,  BC → D}
✓  F1 = {A → B,   A → C,   C → D}
✗  F2 = {A → D,   A → C,   C → D}
✗  F3 = {A → B,   A → C,   C → D,  D → C}
```

NOTE
F cannot be derived from F2
F3 cannot be derived from F

Minimal Basis

- **Definition**
  - Given a set F of FDs, the minimal basis of F is a *simplified* version of F
    - Also called minimal cover *(let's call this F_b)*
  - How simplified?
    - Four conditions

2. Every FD in F_b are non-trivial and decomposed FD

- `F = {A → BD, AB → C, C → D, BC → D}`

✓ `F1 = {A → B, A → C, C → D}`

✗ `F2 = {A → BD, A → C, C → D}`

✓ `F3 = {A → B, A → C, C → D, BC → D}`

NOTE
A → BD is non-decomposed

Minimal Basis

- **Definition**
  - Given a set F of FDs, the minimal basis of F is a *simplified* version of F
    - Also called minimal cover *(let's call this F_b)*
  - How simplified?
    - Four conditions

3. For each FD in F_b, none of the attributes on the left hand side is redundant *(no redundant attributes)*
   - $F = \{A \rightarrow BD,\ AB \rightarrow C,\ C \rightarrow D,\ BC \rightarrow D\}$
     - Consider $AB \rightarrow C$, if we remove B from left hand side, we have $A \rightarrow C$
     - We can derive $A \rightarrow C$ from F since $\{A\}^+ = \{ABCD\}$ *(i.e., $A \rightarrow C$ is "hidden" in F)*
     - So we can add $A \rightarrow C$ without adding extraneous constraints
     - But once we add $A \rightarrow C$ then $AB \rightarrow C$ is redundant!
       - Effectively, we found that B is redundant in $AB \rightarrow C$

     ∴ $AB \rightarrow C$ should not be in minimal basis

Minimal Basis

- **Definition**
  - Given a set F of FDs, the minimal basis of F is a *simplified* version of F
    - Also called minimal cover *(let's call this F_b)*
  - How simplified?
    - Four conditions

4. No FD in F_b are redundant *(no redundant FD)*
   - F = {A → BD, AB → C, C → D, BC → D}
     - Consider BC → D
     - We can derive it from C → D
       - So we can remove it without removing any important information

     ∴ BC → D should not be in minimal basis

Minimal Basis

Conditions
1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$ $\qquad F_b = \{A \rightarrow B, \quad B \rightarrow C\}$
  - Is F_b a minimal basis for F?
    1. $A \rightarrow C$ in F can be derived from F_b
       - F_b is F by removal of $A \rightarrow C$
    2. All FDs in F_b are non-trivial and decomposed
    3. For any FD in F_b, if we remove an attribute from left hand side, then the FD cannot be derived from F *(in fact, they have no left hand side!)*
    4. If any FD in F_b is removed, then some FD in F cannot be derived

    $\therefore$ F_b is a minimal basis for F

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$ $\qquad F_b = \{A \rightarrow B, \quad AB \rightarrow C\}$
  - Is F_b a minimal basis for F?
    1. $B \rightarrow C$ in F can **NOT** be derived from F_b

    $\therefore$ F_b is **NOT** a minimal basis for F

ERRATA

There was a typo error:
Originally it was written as
$A \rightarrow C$ in F can NOT be derived from F_b
It should be $B \rightarrow C$ in F can NOT be derived from F_b

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$ $F_b = \{A \rightarrow BC,\ A \rightarrow C\}$
  - Is F_b a minimal basis for F?
    1. $B \rightarrow C$ in F can **NOT** be derived from F_b
    2. Also...
       - $A \rightarrow BC$ is **NOT** a decomposed FD

$\therefore$ F_b is **NOT** a minimal basis for F

Minimal Basis

Conditions
1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{A \rightarrow B,\ A \rightarrow C,\ C \rightarrow B\}$ $F_b = \{A \rightarrow B,\ AB \rightarrow C,\ C \rightarrow B\}$
  - Is F_b a minimal basis for F?
    1. ✓
    2. ✓
    3. B in $AB \rightarrow C$ in can be removed in Fb *(redundant attribute)*
       - $\{A\}^+ = \{ABC\}$

$\therefore F_b$ is **NOT** a minimal basis for F

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

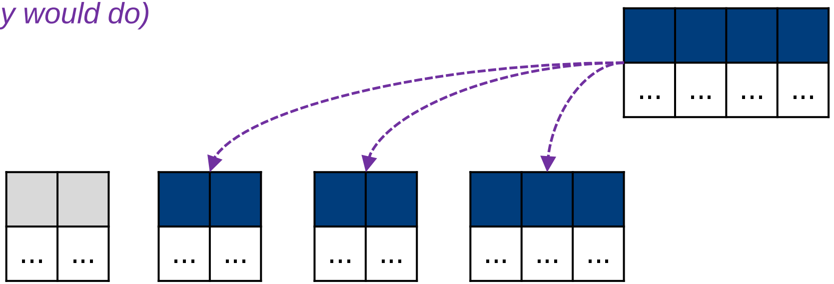
- **Example**
  - $F = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$ $F_b = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$
  - Is F_b a minimal basis for F?
    1. ✓
    2. ✓
    3. ✓
    4. $A \rightarrow C$ can be removed! ***(redundant FD)***

    since it is just transitive

    ∴ F_b is **NOT** a minimal basis for F

Minimal Basis

- **A Reminder on Why We Need Minimal Basis**
  - **3NF Synthesis**
    - **Input:** Table `R` and a set of FDs `F`
    1. Derive ***minimal basis*** of `F`
    2. From the ***minimal basis***, combine the FDs for which the left hand sides are the same *(union rule, producing what's called as canonical cover)*
    3. Create a table for each FDs remained in the ***minimal basis*** after union
    4. If none of the tables contains a key of the original table R, create a table that contains a key of `R` *(any key would do)*



Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

❖ *Clearly, at the end, we have satisfied all conditions* *(as long as we maintain $F_b \equiv F$)*

➢ But how do we perform step 3?

IDEA

Start with F = F1 ∪ {AB → C}.

- If A is redundant, then F1 ∪ {A → C} is equivalent to F.
- F can definitely be derived from F1 because {AB → C} can be derived from {A → C} using Augmentation + Decomposition.
- So we only have to check that F1 can be derived from F.
- This is done by deriving {AB → C} from F1 ∪ {A → C}
  - How? Compute $\{A\}^+$ and check if C is inside

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant Attribute

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- Consider an FD `{A}` → `B` in `F` for any `A` and `B` *(in here, `{A}` is a set of attributes)*
  1. Consider an attribute `C` in `{A}`
     - Compute $\{A\text{-}C\}^+$ using `F` *(in here, `{A-C}` means we remove `C` from `{A}`)*
     - If `B` ∈ $\{A\text{-}C\}^+$, then we can remove `C` *(because `{A-C}`→`B`)*
       - Repeat step 1 but with `{A}` → `B` changed with `{A-C}` → `B`
  2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm: Remove Redundant Attribute**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - F = {A → B, A → D, AB → C, C → D, BC → D}
    - Consider AB → C
      - $\{A\}^+$ = {A,B,C,D}
        - B can be removed from AB → C
    - Repeat with
      - F = {A → B, A → D, A → C, C → D, BC → D}

Algorithm

- For {A} → B in F
1. Consider an attribute C in {A}
   - Compute $\{A-C\}^+$ using F
   - If B in $\{A-C\}^+$, replace {A} → B with {A-C} → B then repeat
2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm: Remove Redundant Attribute**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, A → D, A → C, C → D, BC → D}`
    - Consider `BC → D`
      - `{B}⁺ = {B}`
        - `C` can **NOT** be removed from `BC → D`
      - `{C}⁺ = {CD}`
        - `B` can be removed from `BC → D`
    - Repeat with
      - `F = {A → B, A → D, A → C, C → D, C → D}`

Algorithm

- For `{A} → B` in `F`

1. Consider an attribute `C` in `{A}`
   - Compute `{A-C}⁺` using `F`
   - If `B` in `{A-C}⁺`, replace `{A} → B` with `{A-C} → B` then repeat
2. Do this for all FDs

Minimal Basis

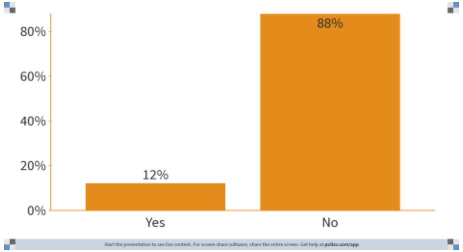
Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant Attribute

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, A → D, A → C, C → D, C → D}`
    - Nothing else to remove
- **Question:**
  - Is this always gives us a unique solution?



Algorithm

- For `{A} → B` in `F`

1. Consider an attribute `C` in `{A}`
   - Compute $\{A\text{-}C\}^+$ using `F`
   - If `B` in $\{A\text{-}C\}^+$, replace `{A} → B` with `{A-C} → B` then repeat
2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant Attribute

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, B → A, AB → C}`
    - Consider `AB → C`
      - $\{A\}^+ = \{A,B,C\}$
        - `B` can be removed from `AB → C`
    - We could have started with
      - $\{B\}^+ = \{A,B,C\}$
        - `A` can be removed from `AB → C`
    - BUT, we cannot remove both!
      - So two possible solutions

Algorithm

- For `{A} → B` in `F`

1. Consider an attribute `C` in `{A}`
   - Compute $\{A\text{-}C\}^+$ using `F`
   - If `B` in $\{A\text{-}C\}^+$, replace `{A} → B` with `{A-C} → B` then repeat
2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

❖ *Clearly, at the end, we have satisfied all conditions* *(as long as we maintain $F_b \equiv F$)*

➢ But how do we perform **step 4**?

IDEA

Start with F = F1 ∪ {A → B}.

- If {A → B} is redundant, then F1 is equivalent to F.
- F1 can definitely be derived from F because F1 contains everything that F has but F has an additional {A → B}.
- So we only have to check that F can be derived from F1.
- But the difference is only {A → B}.
- This is done by deriving {A → B} from F1
    - How? Compute $\{A\}^+$ and check if B is inside

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm: Remove Redundant FD**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

   - Consider an FD $\{A\} \rightarrow B$ in F for any A and B *(in here, $\{A\}$ is a set of attributes)*
     - **Observation:**
       - If we can remove $\{A\} \rightarrow B$, that means $F - \{\{A\} \rightarrow B\} \equiv F$
       - The only difference is the removal of $\{A\} \rightarrow B$
         - Clearly, all FD in $F - \{\{A\} \rightarrow B\}$ can be derived from F
         - So what we need to show is only that $\{A\} \rightarrow B$ can be derived from $F - \{\{A\} \rightarrow B\}$
       - Compute $\{A\}^+$ using $F - \{\{A\} \rightarrow B\}$
         - Then check if B is in $\{A\}^+$

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm: Remove Redundant FD**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

   - Consider an FD `{A} → B` in `F` for any `A` and `B` *(in here, `{A}` is a set of attributes)*
     1. Compute $\{A\}^+$ using `F - {{A} → B}`
        - If `B` $\in$ $\{A\}^+$, then we can remove `{A} → B` *(because `{A}`→`B` even in its absence)*
          - Repeat with next FD but with `{A} → B` removed
     2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant FD

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

   - **Example:**
     - F = {A → B, A → D, A → C, C → D}
       - Consider A → B
         - $\{A\}^+$ w.r.t {A → D, A → C, C → D} = {A,C,D}
           - A → B can **NOT** be remove

Algorithm

- For {A} → B in F
  1. Compute $\{A\}^+$ using F-{{A}→B}
     - If B in $\{A\}^+$, remove {A} → B then repeat with next FD
  2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant FD

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - F = {A → B, A → D, A → C, C → D}
    - Consider A → D
      - {A}$^+$ w.r.t {A → B, A → C, C → D}
        = {A,B,C,D}
        - A → D can be removed
    - Repeat with
      - F = {A → B, A → C, C → D}

Algorithm

- For {A} → B in F
  1. Compute {A}$^+$ using F-{{A}→B}
     - If B in {A}$^+$, remove {A} → B then repeat with next FD
  2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant FD

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, A → C, C → D}`
    - Consider `A → C`
      - `{A}`$^+$ w.r.t `{A → B, C → D}` `= {A,B}`
        - `A → C` can **NOT** be removed

Algorithm

- For `{A} → B` in `F`
  1. Compute `{A}`$^+$ using `F-{{A}→B}`
     - If `B` in `{A}`$^+$, remove `{A} → B` then repeat with next FD
  2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant FD

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, A → C, C → D}`
    - Consider `C → D`
      - `{C}`$^+$ w.r.t `{A → B, A → C}`
        `= {C}`
        - `C → D` can **NOT** be removed

Algorithm

- For `{A} → B` in `F`
  1. Compute `{A}`$^+$ using `F-{{A}→B}`
     - If `B` in `{A}`$^+$, remove `{A} → B`
       then repeat with next FD
  2. Do this for all FDs

Minimal Basis

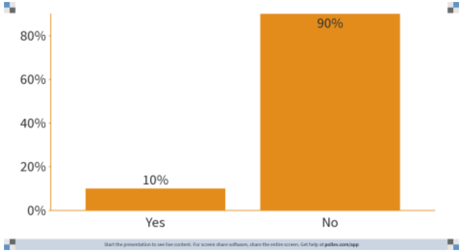
Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- ## Algorithm: Remove Redundant FD

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A → B, A → C, C → D}`
    - No other FDs to consider
- **Question:**
  - Is this always gives us a unique solution?



Algorithm

- For `{A} → B` in `F`
  1. Compute $\{A\}^+$ using `F-{{A}→B}`
     - If `B` in $\{A\}^+$, remove `{A} → B` then repeat with next FD
  2. Do this for all FDs

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm: Remove Redundant FD**

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
4. Remove redundant FDs *(maintain $F_b \equiv F$)*

- **Example:**
  - `F = {A→B, A→C, B→A, B→C, C→A, C→B}`
    - Can remove either one but not both
      - A → B
        - `{A}`$^+$ w.r.t. `{A→C, B→A, B→C, C→A, C→B}` `= {A, B, C}`
      - A → C
        - `{A}`$^+$ w.r.t. `{A→B, B→A, B→C, C→A, C→B}` `= {A, B, C}`

Algorithm

- For `{A} → B` in `F`

1. Compute `{A}`$^+$ using `F-{{A}→B}`
   - If `B` in `{A}`$^+$, remove `{A} → B` then repeat with next FD
2. Do this for all FDs

Minimal Basis

Conditions
1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Algorithm**
  2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
  3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*

▪ For `{A}` → `B` in `F`
1. Consider an attribute `C` in `{A}`
   - Compute `{A-C}`$^+$ using `F`
   - If `B` in `{A-C}`$^+$, replace `{A}` → `B` with `{A-C}` → `B` then repeat
2. Do this for all FDs

  4. Remove redundant FDs *(maintain $F_b \equiv F$)*

▪ For `{A}` → `B` in `F`
1. Compute `{A}`$^+$ using `F-{{A}→B}`
   - If `B` in `{A}`$^+$, remove `{A}` → `B` then repeat with next FD
2. Do this for all FDs

❖ *Clearly, at the end, we have satisfied all conditions* *(as long as we maintain $F_b \equiv F$)*

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{BC \rightarrow DE, \ A \rightarrow E, \ D \rightarrow A, \ E \rightarrow B\}$

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
   - $F = \{BC \rightarrow D, BC \rightarrow E, \ A \rightarrow E, \ D \rightarrow A, \ E \rightarrow B\}$
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - Two candidates
     1. $BC \rightarrow D$
     2. $BC \rightarrow E$

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - F = {BC → D, BC → E, A → E, D → A, E → B}

3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - Consider BC → D
     - {B}+ = {B}
       - So C can **NOT** be removed
     - {C}+ = {C}
       - So B can **NOT** be removed

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - `F = {BC → D, BC → E, A → E, D → A, E → B}`

3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - Consider `BC → E`
     - `{B}+ = {B}`
       - So `C` can **NOT** be removed
     - `{C}+ = {C}`
       - So `B` can **NOT** be removed

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{BC \rightarrow D,\ BC \rightarrow E,\ A \rightarrow E,\ D \rightarrow A,\ E \rightarrow B\}$

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
   - $F = \{BC \rightarrow D,\ BC \rightarrow E,\ A \rightarrow E,\ D \rightarrow A,\ E \rightarrow B\}$
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - No redundant attributes
4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Need to check everything

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{BC \rightarrow D,\ BC \rightarrow E,\ A \rightarrow E,\ D \rightarrow A,\ E \rightarrow B\}$

4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Consider $BC \rightarrow D$
     - $\{BC\}^+$ w.r.t. $\{BC \rightarrow E,\ A \rightarrow E,\ D \rightarrow A,\ E \rightarrow B\}$ $= \{B, C\}$
       - So $BC \rightarrow D$ is **NOT** redundant

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - F = {BC → D, BC → E, A → E, D → A, E → B}

4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Consider BC → E
     - $\{BC\}^+$ w.r.t. {BC → D, A → E, D → A, E → B}
       = {A, B, C, D, E}
       - So BC → E is **redundant**
   - ❖ Continue with F = {BC → D, A → E, D → A, E → B}

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{BC \rightarrow D, \quad A \rightarrow E, \quad D \rightarrow A, \quad E \rightarrow B\}$

4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Consider $A \rightarrow E$
     - $\{A\}^+$ w.r.t. $\{BC \rightarrow D, \quad D \rightarrow A, \quad E \rightarrow B\}$
       $= \{A\}$
       - So $A \rightarrow E$ is **NOT** redundant

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - $F = \{BC \rightarrow D,\ A \rightarrow E,\ D \rightarrow A,\ E \rightarrow B\}$

4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Consider $D \rightarrow A$
     - $\{D\}^+$ w.r.t. $\{BC \rightarrow D,\ A \rightarrow E,\ E \rightarrow B\}$ $= \{D\}$
       - So $D \rightarrow A$ is **NOT** redundant

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - `F = {BC → D,  A → E,  D → A,  E → B}`

4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - Consider `E → B`
     - `{E}`$^+$ w.r.t. `{BC → D,  A → E,  D → A}`
       `= {E}`
       - So `E → B` is **NOT** redundant

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Example**
  - F = {BC → D, A → E, D → A, E → B}

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
   - F = {BC → D, BC → E, A → E, D → A, E → B}
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - No redundant attributes
4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - BC → E is redundant

❖ F_b = {BC → D, A → E, D → A, E → B}

Minimal Basis

Conditions

1. $F_b \equiv F$
2. Non-trivial and decomposed
3. No redundant attributes
4. No redundant FD

- **Exercise**
  - F = {A → C, AC → D, AD → B}

2. Transform the FDs so that it is non-trivial and decomposed *(maintain $F_b \equiv F$)*
   - F = {A → C, AC → D, AD → B}
3. Remove redundant attributes on the left hand sides of each FDs *(maintain $F_b \equiv F$)*
   - Two candidates
     1. AC → D *(C is redundant)*
     2. AD → B *(D is redundant)*
   - F = {A → C, A → D, A → B}
4. Remove redundant FDs *(maintain $F_b \equiv F$)*
   - No redundant FD
   - F_b = {A → C, A → D, A → B}

3NF Synthesis

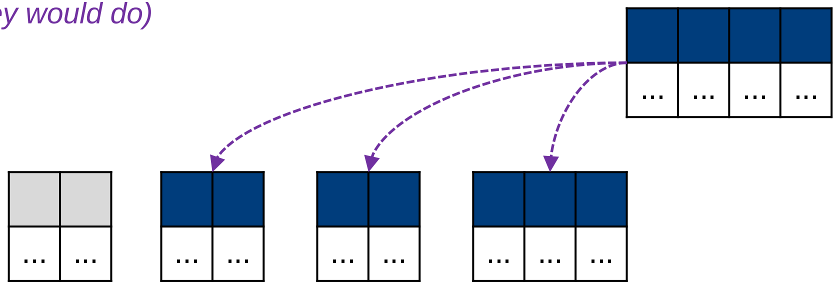
our final product

Dependency Preservation → Define 3NF and Check → Minimal Basis → 3NF Synthesis

3NF Synthesis

- **Algorithm**
  - **3NF Synthesis**
    - **Input:** Table R and a set of FDs F

1. Derive minimal basis of F
2. From the minimal basis, combine the FDs for which the left hand sides are the same *(union rule, producing what's called as canonical cover)*
3. **<u>Create a table for each FDs remained in the minimal basis after union</u>**
4. If none of the tables contains a key of the original table R, create a table that contains a key of R *(any key would do)*



3NF Synthesis

3NF Synthesis
1. Derive *minimal basis* of F
2. Produce *canonical cover*
3. Create a table for each FD
4. Add the *key* if missing

- **Algorithm**
  - **Example:** R(A, B, C, D, E) with BC → DE, A → E, D → A and E → B
    - F = {BC → DE, A → E, D → A, E → B}

    1. Derive minimal basis of F
       - F_b = {BC → D, A → E, D → A, E → B}
    2. Produce canonical cover
       - F_c = {BC → D, A → E, D → A, E → B}
    3. Create a table for each FD
       - R1(B, C, D), R2(A, E), R3(A, D) and R4(B, E)
    4. Add the key if missing
       - Keys = {AC}, {BC}, {CD}, {CE}
       - R1 contains a key

Why add Key?
Ensure *lossless join* decomposition

3NF Synthesis

3NF Synthesis
1. Derive *minimal basis* of F
2. Produce *canonical cover*
3. Create a table for each FD
4. Add the *key* if missing

- ## Lossless Join Decomposition
  - **Example:** R(A, B, C, D) with $A \rightarrow B$ and $C \rightarrow D$
    - $F = \{A \rightarrow B,\ C \rightarrow D\}$

    1. Derive minimal basis of F
       - $F_b = \{A \rightarrow B,\ C \rightarrow D\}$
    2. Produce canonical cover
       - $F_c = \{A \rightarrow B,\ C \rightarrow D\}$
    3. Create a table for each FD
       - R1(A, B) and R2(C, D)
    4. Add the key if missing
       - Keys = {AC}
       - Add R3(A, C)

Why add Key?
R1(A, B) and R2(C, D) cannot be used to reconstruct R(A, B, C, D)

3NF Synthesis

3NF Synthesis

1. Derive *minimal basis* of F
2. Produce *canonical cover*
3. Create a table for each FD
4. Add the *key* if missing

- ## Algorithm
  - **Exercise:** R(A, B, C, D, E) with $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$, $E \rightarrow C$ and $E \rightarrow D$
    - $F = \{A \rightarrow B,\ A \rightarrow C,\ B \rightarrow C,\ E \rightarrow C,\ E \rightarrow D\}$

    1. Derive minimal basis of F
       - $F_b = \{A \rightarrow B,\ B \rightarrow C,\ E \rightarrow C,\ E \rightarrow D\}$
    2. Produce canonical cover
       - $F_C = \{A \rightarrow B,\ B \rightarrow C,\ E \rightarrow CD\}$
    3. Create a table for each FD
       - R1(A, B), R2(B, C) and R3(C, D, E)
    4. Add the key if missing
       - Keys = {AE}
       - Add R4(A, E)

3NF Synthesis

3NF Synthesis
1. Derive *minimal basis* of F
2. Produce *canonical cover*
3. Create a table for each FD
4. Add the *key* if missing

- ## Algorithm
  - **Exercise:** R(A, B, C, D, E) with $A \to B$, $AB \to C$, $C \to DE$, $E \to C$ and $E \to D$
    - $F = \{A \to B,\ AB \to C,\ C \to DE,\ E \to C,\ E \to D\}$
    1. Derive minimal basis of F
       - $F_b = \{A \to B,\ A \to C,\ C \to D,\ C \to E,\ E \to C\}$
    2. Produce canonical cover
       - $F_c = \{A \to BC,\ C \to DE,\ E \to C\}$
    3. Create a table for each FD
       - R1(A, B, C), R2(C, D, E) and R3(C, E)
    4. Add the key if missing
       - Keys = {A}
       - R1 contains a key

Final Thought

closing statement

Summary

- Poorly designed tables give rise to redundancy, update anomalies and deletion anomalies
- **BCNF** eliminates these problems
  - BCNF: for any non-trivial and decomposed FD on a table `R`, its left hand side is a superkey for `R`
  - But BCNF does not always preserve all FDs *(non dependency preserving)*
  - We may need to perform a join of multiple tables to check whether an FD holds
- **3NF** is slightly weaker than BCNF *(has more redundancies, has update and deletion anomalies in some rare cases)* but preserves all FDs
  - 3NF: for any non-trivial and decomposed FD on a table `R`, either its left hand side is a superkey for `R`, *or its right hand side is a prime attribute*

BCNF or 3NF or *Lower*?

- BCNF is only *inferior* to 3NF in the sense that sometimes it does not preserve all FDs
- **Idea:**
  - Go for BCNF if we can find a BCNF decomposition that preserves all FDs
  - If such decomposition cannot be found, then
    - Still go for BCNF if preserving all FDs is not important
    - Or go for 3NF otherwise
      - If we are lucky, 3NF synthesis may actually find a BCNF decomposition!
- Should we go **lower** than 3NF?
  - Notice how there can be many tables produced by 3NF
  - What will happen to queries?
    - We may have to perform lots of joins
    - This can slow down queries by a lot since joins are expensive
  - Even 3NF may not be suitable in that case

QUESTION?