Boyce-Codd Normal Form (BCNF)

→ A table R is in BCNF, if & only if LHS of EVERY non-trivial FD contains a key of R

→ Non-trivial FD: An FD not implied by axiom of reflexivity

Either FD is trivial or LHS of FD contains a key

→ Example:

Non-trivial

- $A \rightarrow B$
- $AC \rightarrow BC$

RHS not a subset of LHS

trivial

- $AC \rightarrow C$
- $AC \rightarrow A$

A and C are subsets of AC

IN BCNF	NOT IN BCNF
R(A,B) ; Given FD : $A \rightarrow B$; Key : A	R(A,B,C) ; Given FD : $A \rightarrow B$; key : AC
All FDs on R : $A \rightarrow B$, $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow AB$	All FDs on R : $A \rightarrow B$, $AB \rightarrow B$, $AC \rightarrow C$, ...
($A \rightarrow B$: LHS contains a key; $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow AB$: trivial FDs)	($A \rightarrow B$: non-trivial; but does not have key on LHS; $AB \rightarrow B$, $AC \rightarrow C$: trivial FDs)
∴ R is in BCNF	∴ R is NOT in BCNF

Example:

Name	NRIC	Phone Number	Home Address
Alice	1234	67899876	Jurong East
Alice	1234	83848884	Jurong East
Bob	5678	98765432	Pasir Ris

- NRIC → Name, Address
- key : {NRIC, Phone} ; key has 2 attributes
- Not in BCNF → FD does contain the key ∴ anomalies found

BCNF Checking

Steps	Examples	
	IN BCNF	NOT IN BCNF
→ Given: table R, set of FDs of R	R(A,B,C,D)	R(A,B,C,D)
→ Step 1: Derive the keys of R	Given : $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$	Given : $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$
→ Step 2: Derive all non-trivial FDs on R	key : A (no other keys)	key : A (no other keys)
• Too time-consuming	Check : For each given non-trivial FD, check if LHS contains a key	Check : For each given non-trivial FD, check if LHS contains a key
• Trick : only check FDs given on R instead of all FDs	$A \rightarrow B$: Non-trival, LHS contains a key	$A \rightarrow B$: Non-trival, LHS contains a key
→ Step 3: For each non-trival FD, check if its left hand side contains a key	$A \rightarrow C$: Non-trival, LHS contains a key	$B \rightarrow C$: Non-trival, LHS contains DOES NOT a key (violation!)
→ Step 4: If all FDs pass the check, then R is in BCNF; Otherwise, R not in BCNF	$A \rightarrow D$: Non-trival, LHS contains a key	
	∴ R is in BCNF	∴ R is not in BCNF

BCNF Checking : Rationale

→ R(A,B,C,D) ; Given : $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$

→ if the 3 FDs pass the check, then any other FDs derived from them will either has same LHS as them or LHS with more attributes

- Reflexivity Axiom: we ignore since this yields trivial FDs
- Augmentation Axiom: LHS of new FD will have more attributes ($A \rightarrow B$ becomes $AC \rightarrow BC$); if original LHS contains key, so will the new LHS
- Transitivity Axiom: LHS of new FD is same as original LHS ($A \rightarrow B$ and $B \rightarrow C$ yields $A \rightarrow C$)

Don't have to check EVERY FD on R, if "given" FDs pass the check, then all "hidden" ones will pass the check

BCNF Checking : Multiple-key Cases

→ R(A, B, C, D)

→ Given : A → B , B → A , B → C , B → D , A → D

→ keys : A , B

→ Check : For each given non-trivial FD, check if it LHS contains a key

- A → B , A → D : Non-trivial, LHS contains a key
- B → A , B → C , B → D : Non-trivial, LHS contains a key
- ∴ R is in BCNF

BCNF Intuition

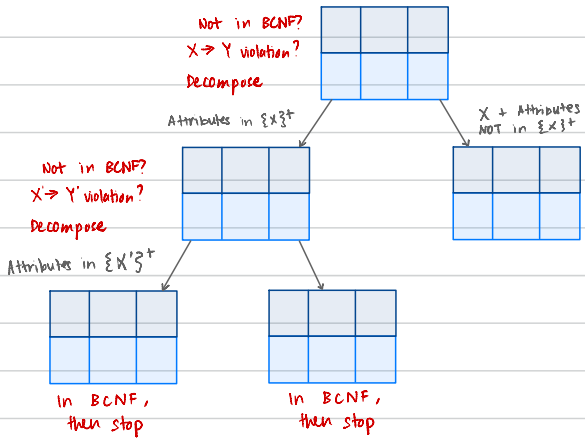
Name	NRIC	Phone Number	Home Address
Alice	1234	67899876	Jurong East
Alice	1234	83848884	Jurong East
Bob	5678	98765432	Pasir Ris

BCNF requires that there cannot be any non-trivial X → Y such that the LHS, X, does not contain a key

Intuition: X → Y indicates that the table has some redundancies.

- NRIC → Name , Address
- key : {NRIC, Phone Number}
- NRIC determines Name & Address
- Therefore, every time NRIC appears repeatedly in the table, Name & Address also appear repeatedly
- Since NRIC is not a key, same NRIC can appear multiple times in the table
- This leads to redundancies
- BCNF prevents this → make sure LHS of non-trivial FDs must contain a key

BCNF Decomposition



→ Decompose until all are in BCNF

→ BCNF table may not be unique

→ If a table has only 2 attributes, then it must be in BCNF

↳ don't need to check 2 attribute tables

BCNF Decomposition Algorithm

→ Input : A table R

→ Step 1 : Find FD X → Y on R that violates BCNF → if none, stop

→ Step 2 : Compute the closure {X}+

→ Step 3 : Break R into 2 tables R_1 and R_2

- R_1 contains all attributes in {X}+
- R_2 contains X and attributes NOT in {X}+

→ Repeat Steps 1-3 on R_1 and R_2

Example :

→ R(A, B, C, D)

→ Given : A → B , B → C , C → D

→ key of R : A

→ Step 1 : B → C is a violation

→ Step 2 : {B}+ = {BCD}

→ Step 3 : Decompose R into 2 tables R_1 & R_2

- R_1 (B, C, D), i.e., it contains all attributes in closure
- R_2 (A, B), i.e., it contains B & all attributes NOT in closure

→ Step 4 : Check R_1 & R_2, decompose if necessary

Tricky case of BCNF Decomposition

- R(A, B, C, D, E)
- A→B, BC→D
- Key of R: ACE
- A→B is a violation. ← choose 1 of the violation
- Decompose R
  - $\{A\}^+ = \{A, B\}$ ← closure
  - $R_1(A, B)$, $R_2(A, C, D, E)$
  - R_1 is in BCNF ← since 2 elements
  - How about R_2? ← need to check if FD ALL inside OR ALL NOT inside
- Key of R_2: ACE
- Violations any? ← B is not in R_2

- In general, we may have a tricky case in BCNF decomposition, if
  - We are checking whether a table T satisfies BCNF, and there is an FD X→Y, such that
    - X contains some attribute in T, but
    - Y contains some attribute NOT in T
- Example in the previous slide:
  - We are checking $R_2(A, C, D, E)$
  - FDs that we have: A→B, BC→D
  - A is in R_2, but B is not
  - This leads to a tricky case
  - In this case, we have to use closures to check whether R_2 is in BCNF

Checking BCNF in a Tricky Case

- We are checking $R_2(A, C, D, E)$
- FDs that we have: A→B, BC→D
- Check the closures:
- $\{A\}^+ = \{A, B\}$, but B is not in $R_2(A, C, D, E)$, so the closure becomes just $\{A\}^+ = \{A\}$
- Similarly, $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$, $\{E\}^+ = \{E\}$
- None of these indicates a violation of BCNF (trivial FDs)
- Skip $\{AB\}^+$ ← not in R_2, do $\{AC\}+ = \{ACD\}$
- This indicates that AC→D and AC does not contain key ACE of R_2
- This means that R_2 is not in BCNF, decompose R_2

- We are checking $R_2(A, C, D, E)$
- FDs that we have: A→B, BC→D
- We know that AC→D violates BCNF on R_2
- Decompose $R_2(A, C, D, E)$
  - $\{AC\}^+ = \{A, C, D\}$
  - $R_3(A, C, D)$, $R_4(A, C, E)$
- R_4 is in BCNF (ACE is key) ← key determines itself
- What about R_3? ↙ A in R_3, but B not in R_3
- Tricky case (A→B); Need to use closure again

- We are checking $R_3(A, C, D)$
- FDs that we have: A→B, BC→D
- $\{A\}^+ = \{A, B\}$ ➔ $\{A\}^+ = \{A\}$ on R_3
- $\{C\}^+ = \{C\}$, $\{D\}^+ = \{D\}$
- $\{AC\}^+ = \{A, B, C, D\}$ ➔ $\{AC\}^+ = \{A, C, D\}$ on R_3
- $\{AD\}^+ = \{A, B, D\}$ ➔ $\{AD\}^+ = \{A, D\}$ on R_3
- $\{CD\}^+ = \{C, D\}$
- None of the closures indicate a violation of BCNF
- Therefore, R_3 is in BCNF
- Final decomposition: $R_1(D, E)$, $R_3(A, C, D)$, $R_4(A, C, E)$

Summary

- Whenever we have a tricky case in checking BCNF, we need to resort to closures
- Using closures, a table T is NOT in BCNF:
  - If there is a closure $\{X\}^+ = \{Y\}$, such that
    - Y does not contain all attributes in T, but (not all)
    - Y contains more attributes than X (more but)
      ↖ LHS
- Previous example:
  - $R_2(A, C, D, E)$, with A→B, BC→D
  - $\{AC\}^+ = \{A, C, D\}$
  - {A, C, D} does not contain all attributes in R_2, but
  - {A, C, D} contains more attributes than {AC}

∴ R2 not in BCNF

Properties of BCNF Decomposition

→ Good properties
- No update or deletion anomalies
- Very little redundancy
- The original table can always be reconstructed from the decomposed tables (aka lossless join property)

Name	NRIC	Phone Number	Home Address
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

Example

Name	NRIC	Home Address
Alice	1234	Jurong East
Bob	5678	Pasir Ris

NRIC	Phone Number
1234	67899876
1234	83848384
5678	98765432

Why BCNF guarantees lossless join?

- Say we decompose a table R into two tables R_1 and R_2
- The decomposition guarantees lossless join, whenever the common attributes in R_1 and R_2 constitute a superkey of R_1 or R_2
- Example
  - R(A, B, C) decomposed into R_1(A, B) and R_2(B, C), with B being the key of R_2
  - R(A, B, C, D) decomposed into R_1(A, B, C) and R_2(B, C, D), with BC being the key of R_1

- The decomposition of R guarantees lossless join, whenever the common attributes in R_1 and R_2 constitute a superkey of R_1 or R_2 or both
- BCNF Decomposition of R
  - Find a BCNF violation X→Y
  - Compute $\{X\}^+$ → LHS
  - R_1 contains all attributes in $\{X\}^+$
  - R_2 contains X and all attributes NOT in $\{X\}^+$
  - X is both in R_1 and R_2
  - And X is a superkey of R_1
  - Therefore, R_1 and R_2 is a lossless decomposition of R

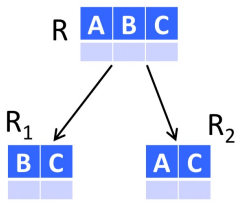
Properties of BCNF Decomposition

- Good properties
  - No update or deletion anomalies
  - Very small redundancy
  - The original table can always be reconstructed from the decomposed tables (this is called the lossless join property)
- Bad property
  - It may not preserve all functional dependencies

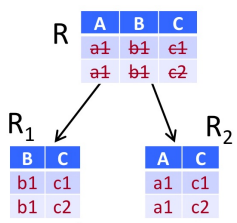
especially when a key is a multiattribute key, individual attributes in key may have some repetition

Dependency Preservation

- Given: Table R(A, B, C)
  - with AB→C, C→B
- Keys: {AB}, {AC}
- BCNF Decomposition
  - R_1(B, C)
  - R_2(A, C)
- Non-trivial FD on R_1: C→B
- Non-trivial FD on R_2: none
- The other FD AB→C is "lost"; not in R_1 nor R_2
- This why we say that a BCNF decomposition does not always preserve all FDs



- Why do we want to preserve FDs?
- Because we want to make it easier to avoid "inappropriate" updates
  - is insertion / deletion of records in table
- Previous example
  - We have two tables R_1(B, C), R_2(A, C)
  - We have C→B and AB→C
  - Due to AB→C, we are not suppose to have two tuples (a1, b1, c1) and (a1, b1, c2)
  - But as we store A and B separately in R_1 and R_2, not easy to check whether such two tuples with similar AB exist at the same time
  - If someone wants to insert (a1, b1, c2), not easy for us to check whether (a1, b1, c1) already exists
  - This is less than ideal



BCNF guarantees lossless joins, but does not guarantee dependency preservation