

CZ2007 Introduction to Database Systems (Week 4)

Topic 3: Boyce-Codd Normal Form (2)



Dr. Ng Wee Keong

Associate Professor

This presentation is copyright property of NTU. It is intended for students of CZ2007 only.



This Lecture

- Tricky case of BCNF ←
- Properties of BCNF



Tricky Case of BCNF Decomposition

- $R(A, B, C, D, E)$
- $A \rightarrow B, BC \rightarrow D$
- Key of R: ACE
- $A \rightarrow B$ is a violation.
- Decompose R
 - $\{A\}^+ = \{A, B\}$
 - $R_1(A, B), R_2(A, C, D, E)$
 - R_1 is in BCNF
 - How about R_2 ?
- Key of R_2 : ACE
- Violations any?
- This is tricky...



Tricky Case of BCNF Decomposition

- 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 $X \rightarrow Y$, 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: $A \rightarrow B$, $BC \rightarrow D$
 - 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: $A \rightarrow B$, $BC \rightarrow D$
- 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 $\{AB\}^+$, do $\{AC\}^+ = \{ACD\}$
- This indicates that $AC \rightarrow D$ and AC does not contain key ACE of R_2
- This means that R_2 is not in BCNF, decompose R_2



Checking BCNF in a Tricky Case

- We are checking $R_2(A, C, D, E)$
- FDs that we have: $A \rightarrow B$, $BC \rightarrow D$
- We know that $AC \rightarrow D$ 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)
- What about R_3 ?
- Tricky case ($A \rightarrow B$); Need to use closure again



Checking BCNF in a Tricky Case

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



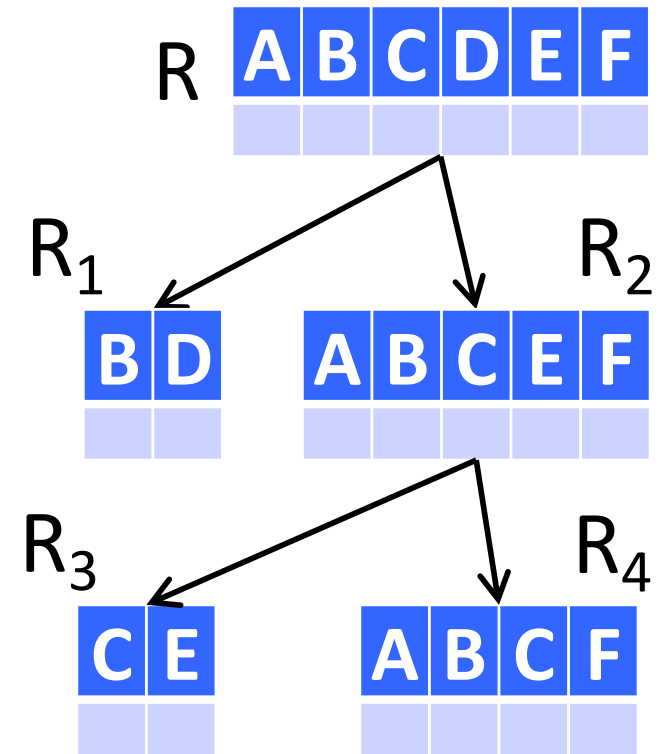
Summary

- 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)
- Previous example:
 - $R_2(A, C, D, E)$, with $A \rightarrow B$, $BC \rightarrow D$
 - $\{AC\}^+ = \{A, C, D\}$
 - $\{A, C, D\}$ does not contain all attributes in R_2 , but
 - $\{A, C, D\}$ contains more attributes than $\{AC\}$



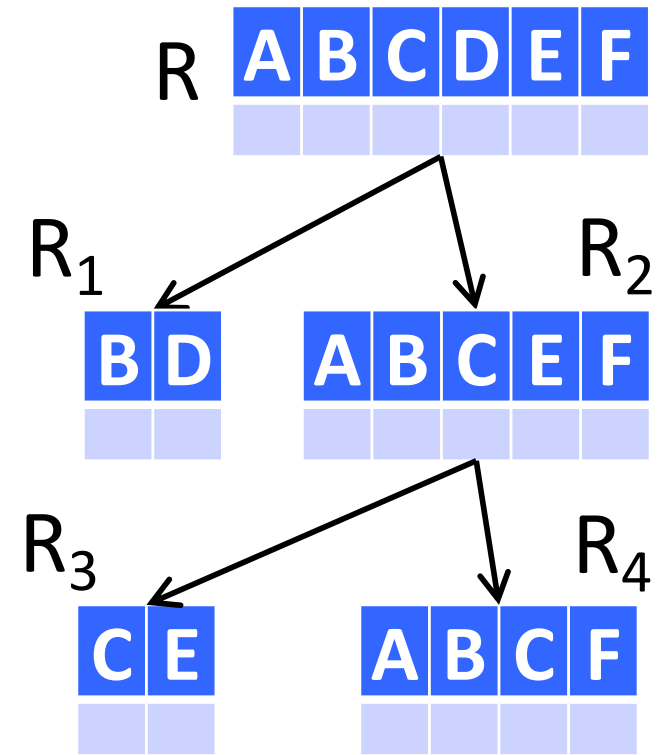
Exercise

- $R(A, B, C, D, E, F)$
- Given FDs: $B \rightarrow D$, $C \rightarrow E$, $DE \rightarrow A$
- Key: BCF
- $B \rightarrow D$ violates BCNF
- Decompose R:
 - ▣ $R_1(B, D)$, $R_2(A, B, C, E, F)$
- R_1 is in BCNF
- What about R_2 ? Key BCF in R_2
- $C \rightarrow E$ violates BCNF
- $\{C\}^+ = \{C, E\}$
- Decompose R_2
 - ▣ $R_3(C, E)$, $R_4(A, B, C, F)$



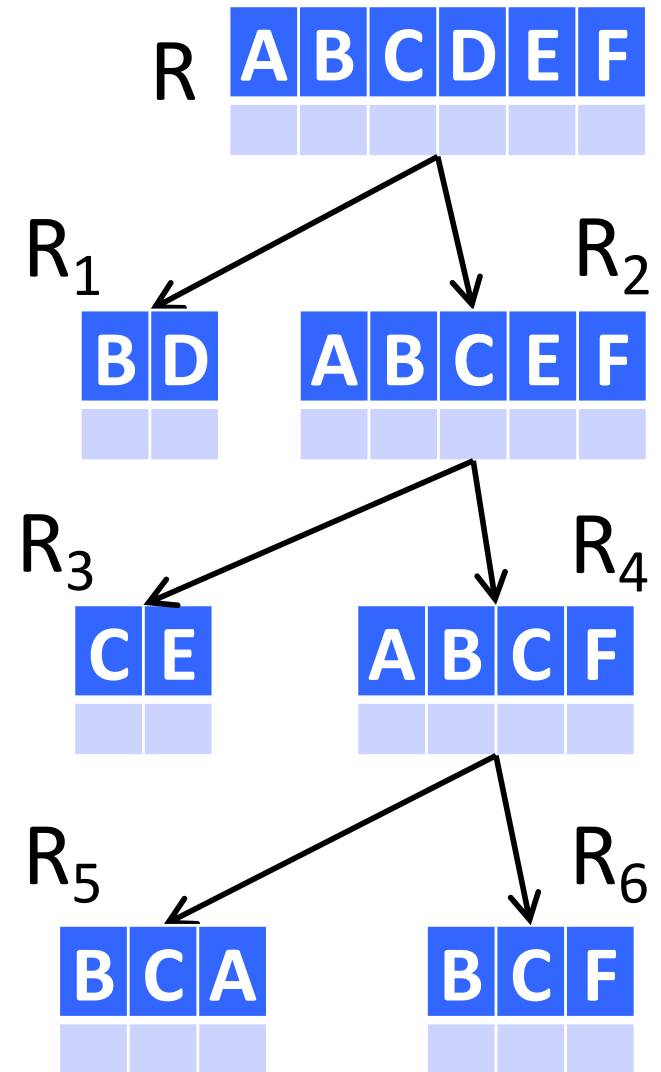
Exercise

- $R(A, B, C, D, E, F)$
- Given FDs: $B \rightarrow D$, $C \rightarrow E$, $DE \rightarrow A$
- $R_3(C, E)$ is in BCNF
- What about $R_4(A, B, C, F)$?
- Tricky case
- Check closures
 - $\{A\}^+ = \{A\}$, $\{B\}^+ = \{BD\}$, $\{C\}^+ = \{CE\}$, $\{F\}^+ = \{F\}$
 - $\{AB\}^+ = \{ABD\}$, $\{AC\}^+ = \{ACE\}$, $\{AF\}^+ = \{AF\}$, $\{BC\}^+ = \{BCDEA\}$
- So there is a non-trivial FD: $BC \rightarrow A$



Exercise

- $R(A, B, C, D, E, F)$
- Given FDs: $B \rightarrow D$, $C \rightarrow E$, $DE \rightarrow A$
- $R_3(C, E)$ is in BCNF
- What about $R_4(A, B, C, F)$?
- Key of R_4 : BCF
- There is a non-trivial FD: $BC \rightarrow A$
- It violates BCNF
- Decompose R_4
 - $R_5(B, C, A)$, $R_6(B, C, F)$
- Final decomposition:
 - $R_1(B, D)$, $R_3(C, E)$, $R_5(B, C, A)$, $R_6(B, C, F)$



This Lecture

- Tricky case of BCNF
- Properties of BCNF ←



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)



Lossless Join Property

Name	<u>NRIC</u>	<u>Phone</u>	Address
Alice	1234	67899876	Jurong East
Alice	1234	83848384	Jurong East
Bob	5678	98765432	Pasir Ris

- The table above can be perfectly reconstructed using the decomposed tables below

Name	<u>NRIC</u>	Address
Alice	1234	Jurong East
Bob	5678	Pasir Ris

<u>NRIC</u>	<u>Phone</u>
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



Why BCNF guarantees lossless join?

- 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
- BCNF Decomposition of R
 - Find a BCNF violation $X \rightarrow Y$
 - Compute $\{X\}^+$
 - 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



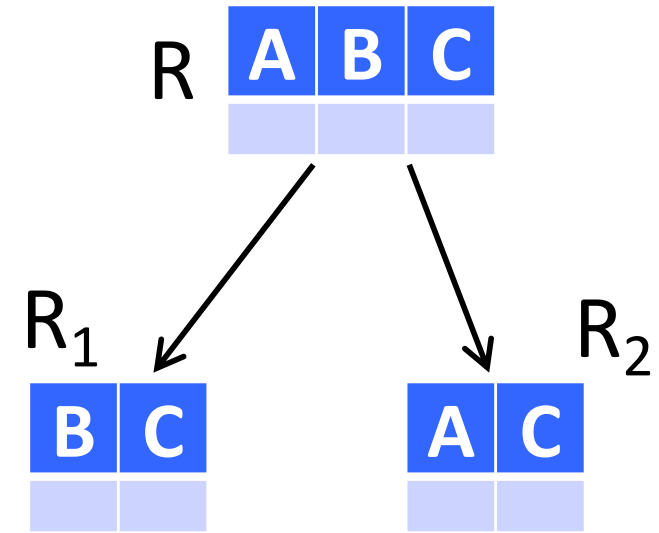
Properties of BCNF Decomposition

- 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



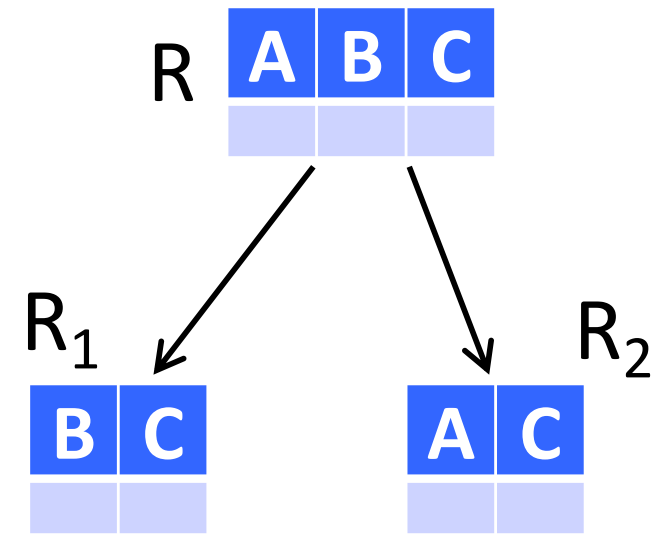
Dependency Preservation

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



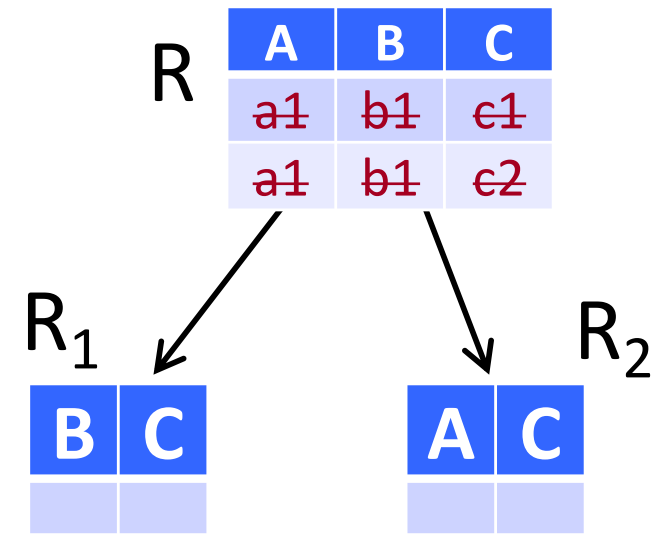
Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates
- Previous example
 - We have two tables $R_1(B, C)$, $R_2(A, C)$
 - We have $C \rightarrow B$ and $AB \rightarrow C$
 - Due to $AB \rightarrow C$, we are not suppose to have two tuples $(a1, b1, c1)$ and $(a1, b1, c2)$
 -
 -
 -



Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates
- Previous example
 - We have two tables $R_1(B, C)$, $R_2(A, C)$
 - We have $C \rightarrow B$ and $AB \rightarrow C$
 - Due to $AB \rightarrow C$, 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 , it is not easy to check whether such two tuples exist at the same time
 -
 -

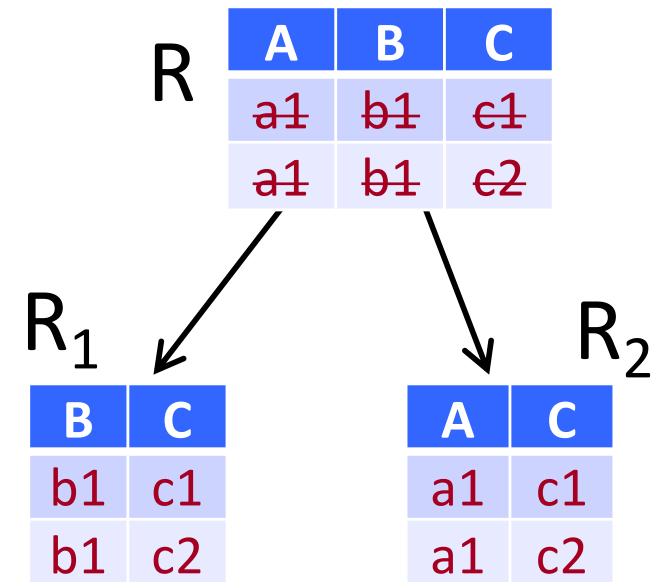


Dependency Preservation

- Why do we want to preserve FDs?
- Because we want to make it easier to avoid “inappropriate” updates

- Previous example

- We have two tables $R_1(B, C)$, $R_2(A, C)$
- We have $C \rightarrow B$ and $AB \rightarrow C$
- Due to $AB \rightarrow C$, 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



Third Normal Form (3NF)

- A relaxation of BCNF that
 - Is less strict
 - Allows decompositions that **always** preserve functional dependencies
- Will discuss in the next lecture



Next lecture:

Topic 4: Third Normal Form (1)

