



**BITS Pilani**  
Pilani Campus

# Database Systems (CSF212) Lecture – 19-20-21



**BITS Pilani**  
Pilani Campus



# Relational Database Design

# Closure of a Set of Functional Dependencies

- Given a set  $F$  set of functional dependencies, there are certain other functional dependencies that are logically implied by  $F$ .
  - For example: If  $A \rightarrow B$  and  $B \rightarrow C$ , then we can infer that  $A \rightarrow C$
- *closure* of  $F$  : The set of all FDs logically implied by  $F$ . It is denoted as  $F^+$

# Closure of a Set of Functional Dependencies

- $F^+$  is computed using **Inference rules** defined for FDs:
  - **IR1:** if  $\beta \subseteq \alpha$ , then  $\alpha \rightarrow \beta$  (**reflexivity**)
  - **IR2:** if  $\alpha \rightarrow \beta$ , then  $\gamma \alpha \rightarrow \gamma \beta$  (**augmentation**)
  - **IR3:** if  $\alpha \rightarrow \beta$ , and  $\beta \rightarrow \gamma$ , then  $\alpha \rightarrow \gamma$  (**transitivity**)
- These rules are
  - **sound** (generate only functional dependencies that actually hold) and
  - **complete** (generate all functional dependencies that hold).

# Closure of a Set of Functional Dependencies

- Few other IR rules that can be used for computing  $F^+$  We can further simplify manual computation of  $F^+$  by using some additional **inference rules**:
    - If  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds, then  $\alpha \rightarrow \beta \gamma$  holds (**union**)
    - If  $\alpha \rightarrow \beta \gamma$  holds, then  $\alpha \rightarrow \beta$  holds and  $\alpha \rightarrow \gamma$  holds (**decomposition**)
    - If  $\alpha \rightarrow \beta$  holds and  $\gamma \beta \rightarrow \delta$  holds, then  $\alpha \gamma \rightarrow \delta$  holds (**pseudo-transitivity**)
- The above rules can be inferred from IR1, IR2 and IR3

# F+ Example

- $R = (A, B, C, G, H, I)$   
 $F = \{ A \rightarrow B, \quad A \rightarrow C, \quad CG \rightarrow H, CG \rightarrow I$   
 $\quad B \rightarrow H \}$
- $F^+ = \{ A \rightarrow B, \quad A \rightarrow C, \quad CG \rightarrow H, CG \rightarrow I$   
 $\quad B \rightarrow H, \textcolor{red}{A \rightarrow H}, \textcolor{red}{CG \rightarrow HI}, \textcolor{red}{AG \rightarrow I} \}$
- $\textcolor{red}{A \rightarrow H}$  by transitivity from  $A \rightarrow B$  and  $B \rightarrow H$
- $\textcolor{red}{AG \rightarrow I}$  by augmenting  $A \rightarrow C$  with  $G$ , to get  $AG \rightarrow CG$  and then transitivity with  $CG \rightarrow I$
- $\textcolor{red}{CG \rightarrow HI}$  by pseudo-transitivity ( $A \rightarrow C, CG \rightarrow I$ )

# Procedure for Computing $F^+$

- To compute the closure of a set of functional dependencies  $F$ :

**Start:**  $F^+ = F$

**repeat**

**for each** functional dependency  $f$  in  $F^+$

    apply reflexivity and augmentation rules on  $f$   
    *and* add the resulting functional

dependencies to  $F^+$

**for each** pair of functional dependencies  $f_1$  and  $f_2$   
in  $F^+$

**if**  $f_1$  and  $f_2$  can be combined using transitivity

**then** add the resulting functional dependency  
to  $F^+$

**until**  $F^+$  does not change any further

**NOTE:** We shall see an alternative procedure for this task later

# F<sup>+</sup> Example

R(A,B,C,D,E)

F = {A → C, A → D, AB → CD, DE → B}

Few FD's in F<sup>+</sup> are:

F = {A → C, A → D, AB → CD, DE → B,

AB → C (augmentation), AC → C (reflexivity) A →  
CD (union), AE → B (augmentation and transitivity)}



# $F^+$ Example

$$R = (A, B, C)$$

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

Few FD's in  $F^+$  are:

$$F^+ = \{AB \rightarrow C, A \rightarrow B, A \rightarrow C, A \rightarrow BC, AC \rightarrow B\}$$

# Closure of Attribute Sets

- Given a set of attributes  $a$ , define the *closure* of  $a$  under  $F$  (denoted by  $a^+$ ) as the set of attributes that are functionally determined by  $a$  under  $F$
- Algorithm to compute  $a^+$ , the closure of  $a$  under  $F$

*result* :=  $a$ ;

**while** (changes to *result*) **do**

**for each**  $\beta \rightarrow \gamma$  **in**  $F$  **do**

**begin**

**if**  $\beta \subseteq \textit{result}$  **then** *result* := *result*  $\cup \gamma$

**end**

# Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, \quad A \rightarrow C, \quad CG \rightarrow H, \quad CG \rightarrow I, \quad B \rightarrow H\}$
- $(AG)^+$ 
  1.  $result = AG$
  2.  $result = ABCG$  ( $A \rightarrow C$  and  $A \rightarrow B$ )
  3.  $result = ABCGHI$  ( $CG \rightarrow H$ ,  $CG \rightarrow I$ , and  $CG \subseteq ABCG$ )
- Is  $AG$  a candidate key?
  1. Is  $AG$  a super key?
    1. Does  $AG \rightarrow R$ ?  $==$  Is  $(AG)^+ \supseteq R$
  2. Is any subset of  $AG$  a superkey?
    1. Does  $A \rightarrow R$ ?  $==$  Is  $(A)^+ \supseteq R$
    2. Does  $G \rightarrow R$ ?  $==$  Is  $(G)^+ \supseteq R$

# Uses of Attribute Set Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if  $\alpha$  is a superkey, we compute  $\alpha^+$ , and check if  $\alpha^+$  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  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - Is a simple and cheap test, and very useful
- Computing closure of  $F$ 
  - For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , and for each  $S \subseteq \gamma^+$ , we output a functional dependency  $\gamma \rightarrow S$ .

# Example of Attribute Set Closure

$$R = (A, B, C)$$

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

- $(A)^+ = \{ABC\}$

- $(B)^+ = \{B\}$

- $(C)^+ = \{C\}$

- $(AB)^+ = \{ABC\}$

- $(BC)^+ = \{BC\}$

- $(CA)^+ = \{ABC\}$

- $(ABC)^+ = \{ABC\}$

- $F^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow BC, A \rightarrow AC, A \rightarrow ABC, AB \dots\}$

# Equivalence of Sets of FDs

- Two sets of FDs  $F$  and  $G$  are **equivalent** if  $F^+ = G^+$ , i.e.,
  - Every FD in  $F$  can be inferred from  $G$ , *and*
  - Every FD in  $G$  can be inferred from  $F$
- Definition:  $F$  **covers**  $G$  if every FD in  $G$  can be **inferred** from  $F$  (i.e., if  $G \subseteq F^+$ )
- $F$  and  $G$  are equivalent if  $F$  covers  $G$  and  $G$  covers  $F$

# Canonical Cover

- Sets of functional dependencies may have **redundant dependencies** that can be inferred from the others
  - For e.g.,  $A \rightarrow C$  is redundant in:  $\{A \rightarrow B, B \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS:  $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
    - E.g.: on LHS:  $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$  can be simplified to  $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
- **Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies**

# Extraneous Attributes

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
  - Attribute  $A$  is **extraneous** in  $\alpha$  if  $A \in \alpha$  and  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ .  
**Example:** Given  $F = \{A \rightarrow C, AB \rightarrow C\}$ .  $B$  is extraneous in  $AB \rightarrow C$  because  $\{A \rightarrow C, AB \rightarrow C\}$  logically implies  $A \rightarrow C$  (i.e. the result of dropping  $B$  from  $AB \rightarrow C$ ).

- Attribute  $A$  is **extraneous** in  $\beta$  if  $A \in \beta$  and  $F$  logically implies  $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ .  
**Example:** Given  $F = \{A \rightarrow C, AB \rightarrow CD\}$ .  $C$  is extraneous in  $AB \rightarrow CD$  since  $AB \rightarrow C$  can be inferred even after deleting  $C$



# Testing if an Attribute is Extraneous

- Consider a set  $F$  of functional dependencies and the functional dependency  $\alpha \rightarrow \beta$  in  $F$ .
- To test if attribute  $A \in \alpha$  is extraneous in  $\alpha$ 
  1. compute  $(\{\alpha\} - A)^+$  using the dependencies in  $F$
  2. check that  $(\{\alpha\} - A)^+$  contains  $\beta$ ; if it does,  $A$  is extraneous in  $\alpha$
- To test if attribute  $A \in \beta$  is extraneous in  $\beta$ 
  1. compute  $\alpha^+$  using only the dependencies in  $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ ,
  2. check that  $\alpha^+$  contains  $A$ ; if it does,  $A$  is extraneous in  $\beta$

# Canonical Cover

- A *canonical cover* for a set of functional dependencies  $F$  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.

# Computing a Canonical Cover

- To compute a canonical cover for  $F$ :  
**repeat**  
    Use the union rule to replace any dependencies in  $F$ :  $\alpha_1 \rightarrow \beta_1$  and  $\alpha_1 \rightarrow \beta_2$  with  $\alpha_1 \rightarrow \beta_1 \beta_2$   
    Find a functional dependency  $\alpha \rightarrow \beta$  with an extraneous attribute either in  $\alpha$  or in  $\beta$   
    If an extraneous attribute is found, delete it from  $\alpha \rightarrow \beta$   
**until**  $F$  does not change
- **Note:** Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

# Computing a Canonical Cover

- $R = (A, B, C)$        $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- Combine  $A \rightarrow BC$  and  $A \rightarrow B$  into  $A \rightarrow BC$ 
  - $F$  is now  $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- $A$  is extraneous in  $AB \rightarrow C$ 
  - Check if the result of deleting  $A$  from  $AB \rightarrow C$  is implied by the other dependencies
    - Yes: in fact,  $B \rightarrow C$  is already present!
  - Set is now  $\{A \rightarrow BC, B \rightarrow C\}$
- $C$  is extraneous in  $A \rightarrow BC$ 
  - Check if  $A \rightarrow C$  is logically implied by  $A \rightarrow B$  and the other dependencies
    - Yes: using transitivity on  $A \rightarrow B$  and  $B \rightarrow C$ .
      - Can use attribute closure of  $A$  in more complex cases
- The canonical cover is:  $A \rightarrow B, B \rightarrow C$

# Computing a Canonical Cover

- $R = (A, B, C, D, E, F)$
- $F = \{AB \rightarrow CDEF, C \rightarrow D, E \rightarrow F, C \rightarrow E\}$
- Combine  $C \rightarrow D$  and  $C \rightarrow E$  into  $C \rightarrow DE$ 
  - $F$  is now  $\{AB \rightarrow CDEF, C \rightarrow DE, E \rightarrow F\}$
- Combine  $C \rightarrow DE$  and  $E \rightarrow F$  into  $C \rightarrow DEF$ 
  - $F$  is now  $\{AB \rightarrow CDEF, C \rightarrow DEF\}$
- The canonical cover is:  $\{AB \rightarrow CDEF, C \rightarrow DEF\}$



# Lossless-join and Dependency Preserving Decomposition

- A decomposition of  $R$  into  $R_1$  and  $R_2$  is lossless join if and only if at least one of the following dependencies is in  $F^+$ :
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- Let  $F_i$  be the set of dependencies  $F^+$  that include only attributes in  $R_i$ . A decomposition is dependency preserving, if only
  - $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$
- If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

# Testing for Dependency Preservation

- To check if a dependency  $\alpha \rightarrow \beta$  is preserved in a decomposition of  $R$  into  $R_1, R_2, \dots, R_n$  we apply the following test (with attribute closure done with respect to  $F$ )
  - **result** =  $\alpha$   
**while** (changes to *result*) do  
    **for each**  $R_i$  in the decomposition  
         $t = (\text{result} \cap R_i)^+ \cap R_i$  //to remove unwanted attributes  
         $\text{result} = \text{result} \cup t$
  - If *result* contains all attributes in  $\beta$ , then the functional dependency
  - $\alpha \rightarrow \beta$  is preserved.
- We apply the test on all dependencies in  $F$  to check if a decomposition is dependency preserving
- This procedure takes **polynomial time**, instead of the **exponential time** required to compute  $F^+$  and  $(F_1 \cup F_2 \cup \dots \cup F_n)^+$

# Example

- $R = (A, B, C)$   
 $F = \{A \rightarrow B, B \rightarrow C\}$ 
  - Can be decomposed in two different ways
- $R_1 = (A, B), R_2 = (B, C)$ 
  - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{B\}$  and  $B \rightarrow BC$   **Dependency preserving**
- $R_1 = (A, B), R_2 = (A, C)$ 
  - Lossless-join decomposition:  
 $R_1 \cap R_2 = \{A\}$  and  $A \rightarrow AB$   **Not dependency preserving**
  - Not dependency preserving  
(cannot check  $B \rightarrow C$  without computing  $R_1 \bowtie R_2$ )



# Generalized Definition 3NF

- **3NF:** A relation schema  $R$  is in 3NF with respect to a set  $F$  of functional dependencies if for all functional dependencies in  $F^+$  of the form  $\alpha \twoheadrightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:
  - $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \in \alpha$ )
  - $\alpha$  is a superkey for  $R$
  - Each attribute  $A$  in  $\beta$  is a prime attribute

(**NOTE:** each attribute may be in a different candidate key)

# Testing for 3NF

- Need to check for all FDs in  $F$ , need not check all FDs in  $F^+$ .
- Use attribute closure to check for each dependency  $\alpha \rightarrow \beta$ , if  $\alpha$  is a superkey.
- If  $\alpha$  is not a superkey, we have to verify if each attribute in  $\beta$  is contained in a candidate key of  $R$ 
  - This test is rather more expensive, since it involve finding candidate keys
  - While testing for 3NF has been shown to be **NP-hard**, decomposition into third normal form can be done in polynomial time

# 3NF Decomposition Algorithm

Let  $F_c$  be a canonical cover for  $F$ ;

$i := 0$ ;

**for each** functional dependency  $\alpha \rightarrow \beta$  in  $F_c$  **do**

**if** none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains  $\alpha \rightarrow \beta$

**then begin**

$i := i + 1$ ;

$R_i := \alpha \rightarrow \beta$

**end**

**if** none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains a candidate key for  $R$

**then begin**

$i := i + 1$ ;

$R_i :=$  any candidate key for  $R$ ;

**end**

**if any schema  $R_j$  is contained in another schema  $R_k$**

**then**

        /\* Delete  $R_j$  \*/

$R_j := R_i$  ;

$i := i - 1$ ;

**until no more  $R_j$ s can be deleted**

**return**  $(R_1, R_2, \dots, R_i)$

# 3NF Decomposition Algorithm

- Above algorithm ensures:
  - each relation schema  $R_i$  is in 3NF
  - decomposition is dependency preserving and lossless-join

# 3NF Decomposition Example

- ***cust\_bank*** = (*c\_id*, *agent\_id*, *branch\_id*, *type*)
- The functional dependencies for this relation schema are:
  1. *c\_id*, *agent\_id* → *branch\_id*, *type*
  2. *agent\_id* → *branch\_id*
  3. *c\_id*, *branch\_id* → *agent\_id*
- We first compute a canonical cover
  - *branch\_name* is extraneous in the r.h.s. of the 1<sup>st</sup> dependency
  - No other attribute is extraneous, so we get  $F_c$ 
    1. *c\_id*, *agent\_id* → *type*
    2. *agent\_id* → *branch\_id*
    3. *c\_id*, *branch\_id* → *agent\_id*

# 3NF Decomposition Example

- The **for** loop generates following 3NF schema:

**R1**(*c\_id, agent\_id, type* )

**R2**(*agent\_id, branch\_id*)

**R3**(*c\_id, branch\_id, agent\_id*)

- Observe that (*c\_id, agent\_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added
- Detect and delete redundant schemas, such as (*agent\_id, branch\_id*), which are subsets of other schemas
- The resultant simplified 3NF schema is:  
**R1**(*c\_id, agent\_id, type* )  
**R3**(*c\_id, branch\_id, agent\_id*)

# Redundancy in 3NF Decomposition

- **banker**(c\_id, agent\_no, br\_code)

agent\_no  $\rightarrow$  br\_code

br\_code, c\_id  $\rightarrow$  agent\_no

candidate key = { {c\_id, agent\_no}, {c\_id, brcode } }

agent_no	br_code	c_id
A1	B1	C1
A1	B1	C2
A2	B1	C3
A3	B2	C1
A3	B2	C2
A4	B1	C5
A4	B2	C5

# Boyce-Codd Normal Form

A relation schema  $R$  is in BCNF with respect to a set  $F$  of functional dependencies if for all functional dependencies in  $F^+$  of the form  $\alpha \twoheadrightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following holds:

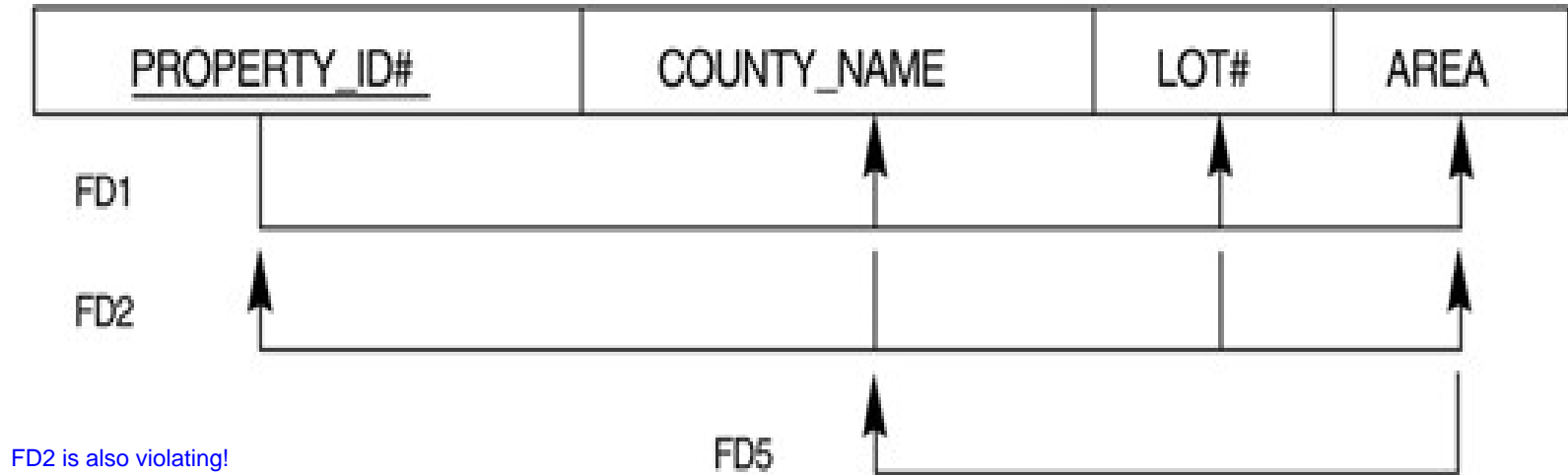
- $\alpha \twoheadrightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
- $\alpha$  is a superkey for  $R$

Let  $R$  be a schema that is not in BCNF. Then there is at least one **nontrivial functional dependency**  $\alpha \twoheadrightarrow \beta$  such that  $\alpha$  is not a superkey for  $R$ . Replace  $R$  with two schemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$



# Boyce-Codd Normal Form



Above relation is not in BCNF due to the following FD:

**AREA → COUNTY\_NAME**

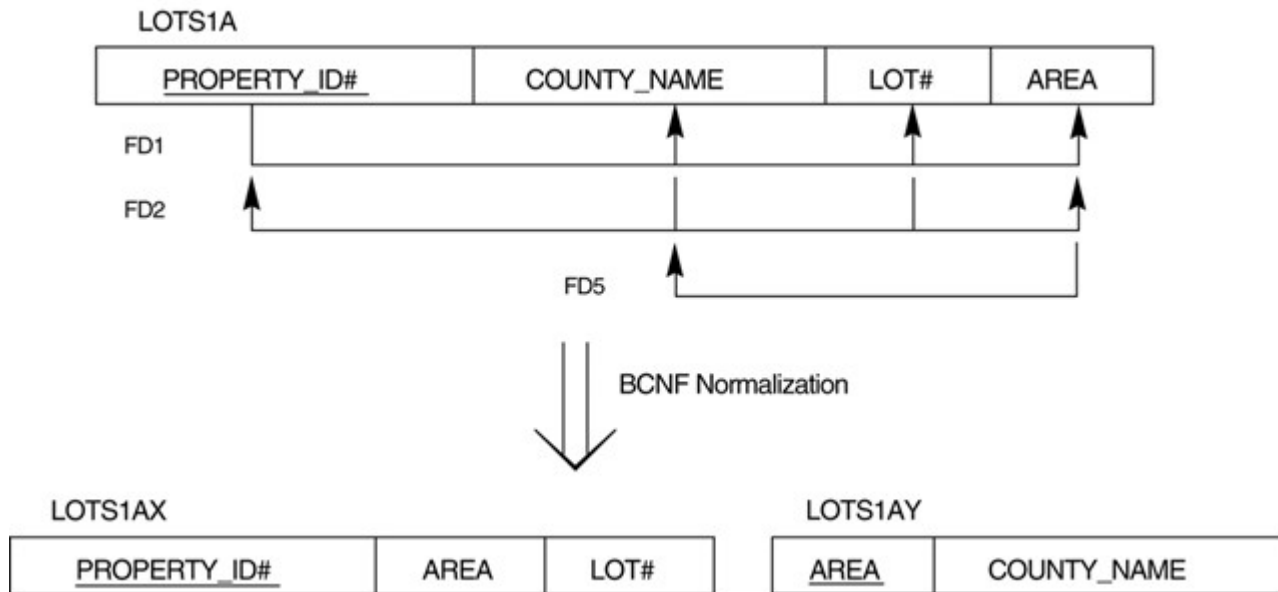
# Boyce-Codd Normal Form

Functional dependency that violates BCNF condition:

**AREA → COUNTY\_NAME**

$\alpha = \text{AREA}, \beta = \text{COUNTY\_NAME}$

- $(\alpha \cup \beta) = \{\text{AREA, COUNTY\_NAME}\}$
- $(R - (\beta - \alpha)) = \{\text{PROPERTY\_ID, AREA, LOT}\}$



# Example of BCNF Decomposition

–  $R = (A, B, C, D, E, F)$

*AB is the superkey.*

– FD's are:

$AB \rightarrow CDEF$

$C \rightarrow D$

$E \rightarrow F$

$C \rightarrow E$

**$C \rightarrow D$**

(C, D)

**(A, B, C, E, F)**

**$E \rightarrow F$**

(E, F)

**(A, B, C, E)**

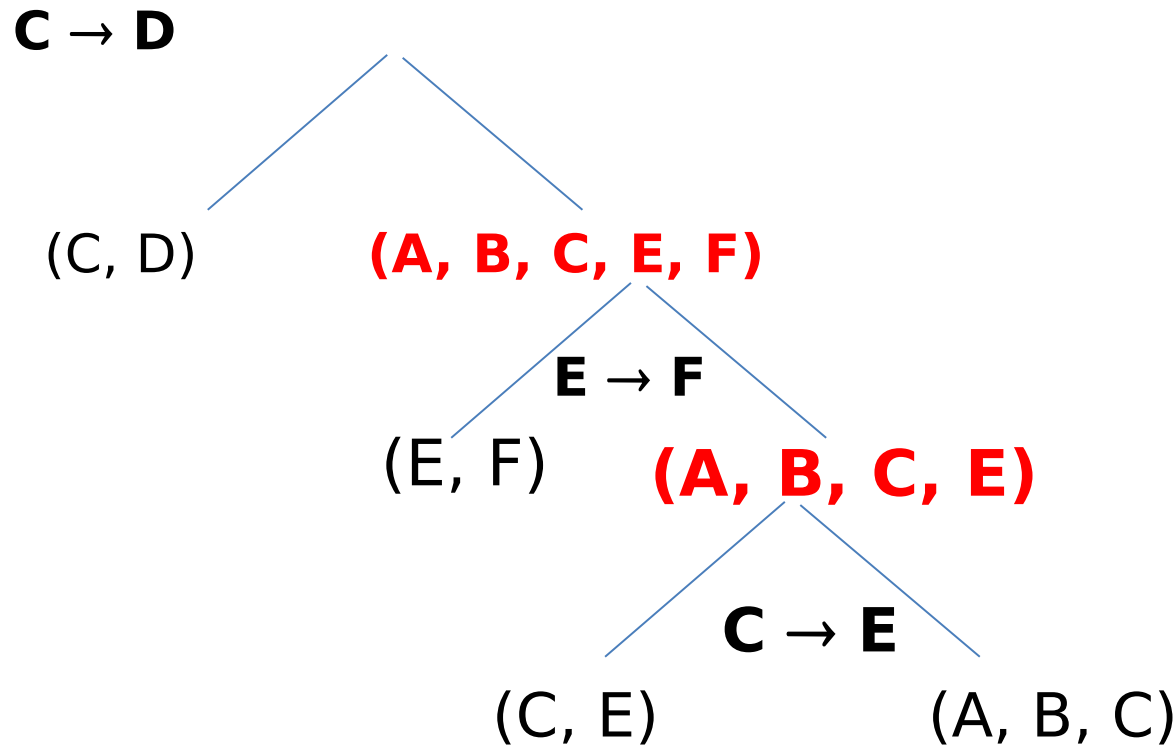
**$C \rightarrow E$**

(C, E)

(A, B, C)

This is a lossless and  
dependency preserving  
BCNF decomposition of the  
relation R

# Example of BCNF Decomposition



To check whether all the FD's in  $F$  can be determined from the set of decomposed relation obtained, **find the canonical cover  $F_c$**  of  $F$ :

$F_c = \{AB \rightarrow C \text{ (DEF are the extraneous attributes in this FD), } C \rightarrow DEF\}$

# BCNF Decomposition EXAMPLE

- **banker**(c\_id, agent\_no, br\_code)

F1: agent\_no  $\rightarrow$  br\_code

F2: br\_code, c\_id  $\rightarrow$  agent\_no

candidate key =  $\{\{c\_id, \text{agent\_no}\}, \{c\_id, \text{br\_code}\}\}$

agent_no	br_code	agent_no	c_id
A1	B1	A1	C1
A2	B1	A1	C2
A3	B2	A2	C3
		A3	C1
		A3	C2

# BCNF and Dependency Preservation

- Functional dependencies, are costly to check in practice unless they pertain to only one relation
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*

# BCNF Decomposition EXAMPLE

- **banker**(c\_id, agent\_no, br\_code)

F1: agent\_no  $\rightarrow$  br\_code

F2: br\_code, c\_id  $\rightarrow$  agent\_no

agent_no	br_code	agent_no	c_id
A1	B1	A1	C1
A2	B1	A1	C2
A3	B2	A2	C3
		A3	C1
		A3	C2

Since F2 is not preserved in any of the decomposed relation, this is not a dependency preserving BCNF decomposition of the *banker* relation

# Testing for BCNF

- To check if a non-trivial dependency  $\alpha \twoheadrightarrow \beta$  in  $F^+$  causes a violation of BCNF
  1. compute  $\alpha^+$  (the attribute closure of  $\alpha$ ), and
  2. verify that it includes all attributes of  $R$ , that is, it is a superkey of  $R$ .
- **Simplified test:** To check if a relation schema  $R$  is in BCNF, it suffices to check only the dependencies in the given set  $F$  for violation of BCNF, rather than checking all dependencies in  $F^+$ .
  - If none of the dependencies in  $F$  causes a violation of BCNF, then none of the dependencies in  $F^+$  will cause a violation of BCNF either.



# Testing for BCNF

- However, using only  $F$  is incorrect when testing a relation in a decomposition of  $R$ 
  - Consider  $R = (A, B, C, D, E)$ , with
$$F = \{ A \rightarrow B, BC \rightarrow D \}$$
    - Decompose  $R$  into  $R_1 = (A, B)$  and  $R_2 = (A, C, D, E)$
    - Neither of the dependencies in  $F$  contain only attributes from  $(A, C, D, E)$  so we might be misled into thinking  $R_2$  satisfies BCNF.
    - In fact, dependency  $AC \rightarrow D$  in  $F^+$  shows  $R_2$  is not in BCNF

# Testing Decompositions for BCNF

- To check if a relation  $R_i$  in a decomposition of  $R$  is in BCNF:
  - Either test  $R_i$  for BCNF with respect to the **restriction** of  $F$  to  $R_i$  (that is, all FDs in  $F^+$  that contain only attributes from  $R_i$ )
  - Or use the original set of dependencies  $F$  that hold on  $R$ , but with the following test:
    - for every set of attributes  $\alpha \subseteq R_i$ , check that  $\alpha^+$  (the attribute closure of  $\alpha$  under  $F$ ) either includes no attribute of  $R_i - \alpha$ , or includes all other attributes of  $R_i$ . (either derives no attribute or is a superkey in  $R_i$ )
    - If the condition is violated by some  $\alpha \twoheadrightarrow \beta$  in  $F$ , the dependency  $\alpha \twoheadrightarrow (\alpha^+ - \alpha) \cap R_i$  can be shown to hold on  $R_i$ , and  $R_i$  violates BCNF
- We use above dependency to decompose  $R_i$

# Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.