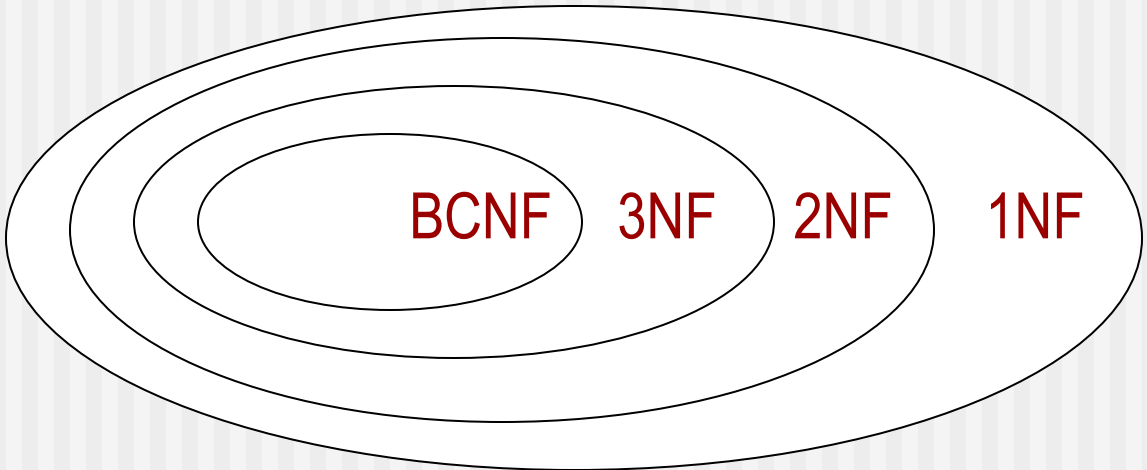
Normal Forms for Relational Data

- If a relation schema is in some normal form, it means it is in some level of quality in the sense that *certain kinds of problems and anomalies (related to redundancy) will not arise*
- Given a relation schema **R** with FDs **F**, how to determine in what normal form it is? And if it is not in a particular normal form, how to convert it into a few smaller relations all of which are in that normal form.
- To address these issues, we study definitions and algorithms for different **normal forms**.

Normal Forms

- The normal forms as defined and captured by FD's:
  - First normal form (1NF)
  - Second normal form (2NF)
  - √ Third normal form (3NF)
  - √ Boyce-Codd normal form (BCNF)
- The relationships among these normal forms:



Third Normal Form (3NF)

Let **R** be a relation schema with a set of FD's **F**.

- We say **R** w.r.t. **F** is in 3NF (**third normal form**), if for every FD $\mathbf{X} \rightarrow \mathbf{A}$ in **F**, at least one of the following conditions holds:
  - $\mathbf{X} \rightarrow \mathbf{A}$ is a trivial, i.e., $\mathbf{A} \in \mathbf{X}$, or
  - **X** is a superkey, or
  - **X** is not a key but **A** is part of some key of **R**

➔ Therefore, to determine if **R** is in 3NF w.r.t. F, we need to:

- Check if the LHS of each nontrivial FD in **F** is a superkey
- If not, check if its RHS is part of any key of **R**

Boyce-Codd Normal Form

Given: A relation schema **R** with a set of FD's **F** on **R.**

- We say **R** w.r.t. **F** is in **Boyce-Codd normal form**, if for every FD $\mathbf{X} \rightarrow \mathbf{Y}$ in **F**, at least one of the following 2 conditions holds
  - $\mathbf{Y} \subseteq \mathbf{X}$, that is, $\mathbf{X} \rightarrow \mathbf{Y}$ is a trivial FD or
  - **X** is a superkey
- To determine if **R** is in BCNF w.r.t. **F**,

  For every FD $\mathbf{X} \rightarrow \mathbf{Y}$, check if its LHS **X** is a superkey.

  That is, compute $\mathbf{X}^+$.

  If $\mathbf{X}^+ = \mathbf{R}$, then no problem; the FD $\mathbf{X} \rightarrow \mathbf{Y}$ passes**.**

  If $\mathbf{X}^+ \neq \mathbf{R}$, then **R** is not in BCNF.

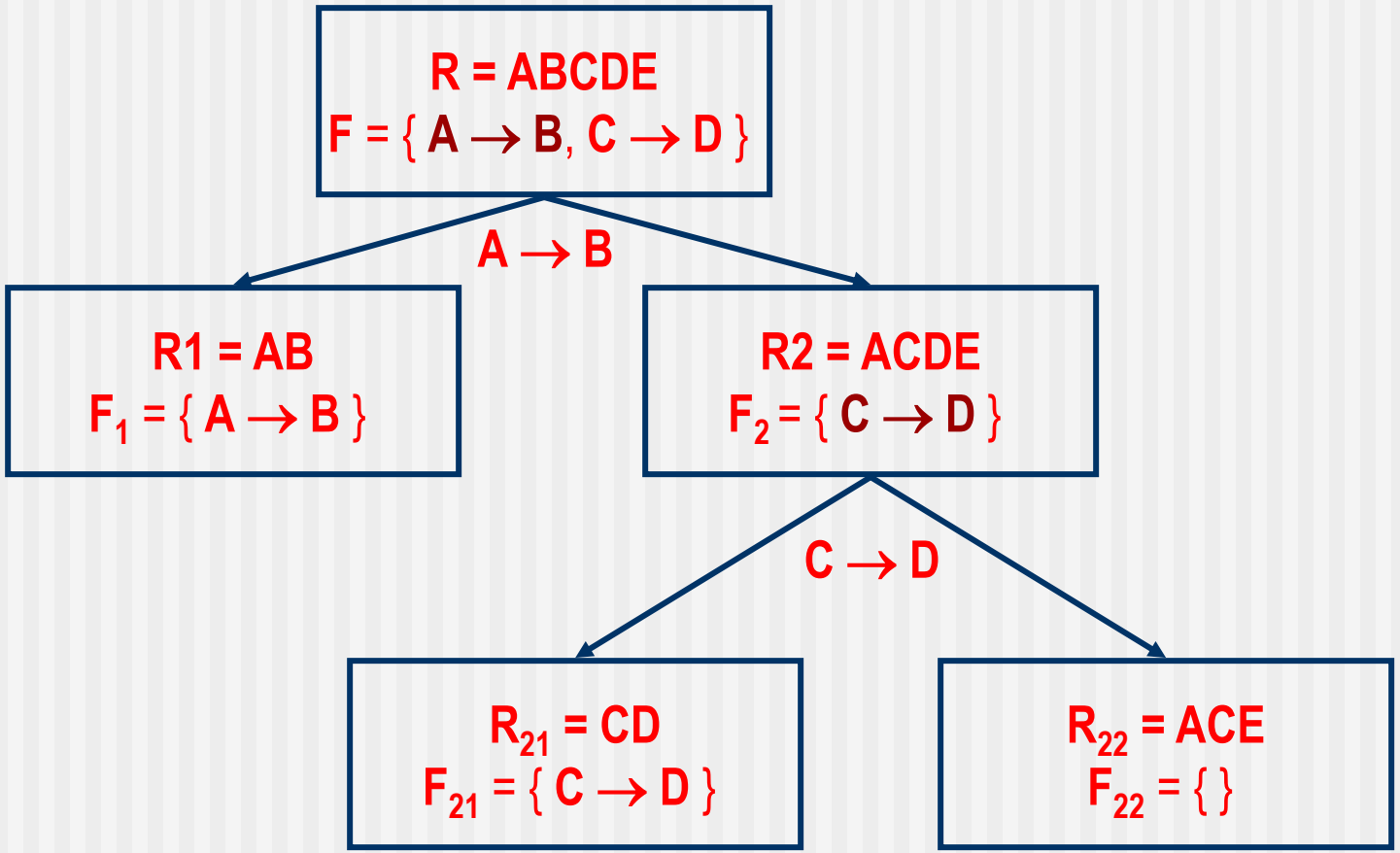
Decomposition into BCNF relations

- Suppose relation R is in 1NF. Consider <**R**, **F**>
- If **R** is not in BCNF w.r.t. **F**, we can always obtain a *lossless-join decomposition* of **R** into a collection of BCNF relations
- However, this decomposition may not always be dependency preserving.
- The basic step of a BCNF decomposition alg. (done **recursively**):

  Pick an FD $\mathbf{X} \rightarrow \mathbf{A} \in \mathbf{F}$ that violates the BCNF requirement:

  1. Decompose **R** into two relations: R1=$\mathbf{X}^+$ and R2=$(\mathbf{R} - \mathbf{X}^+) \cup \mathbf{X}$
  2. Project F onto R1 and R2. Let's call them as F1 and F2.
  3. If <R1,F1> or <R2,F2> is not in BCNF, decompose further.

Example (Decomposition into BCNF relations)



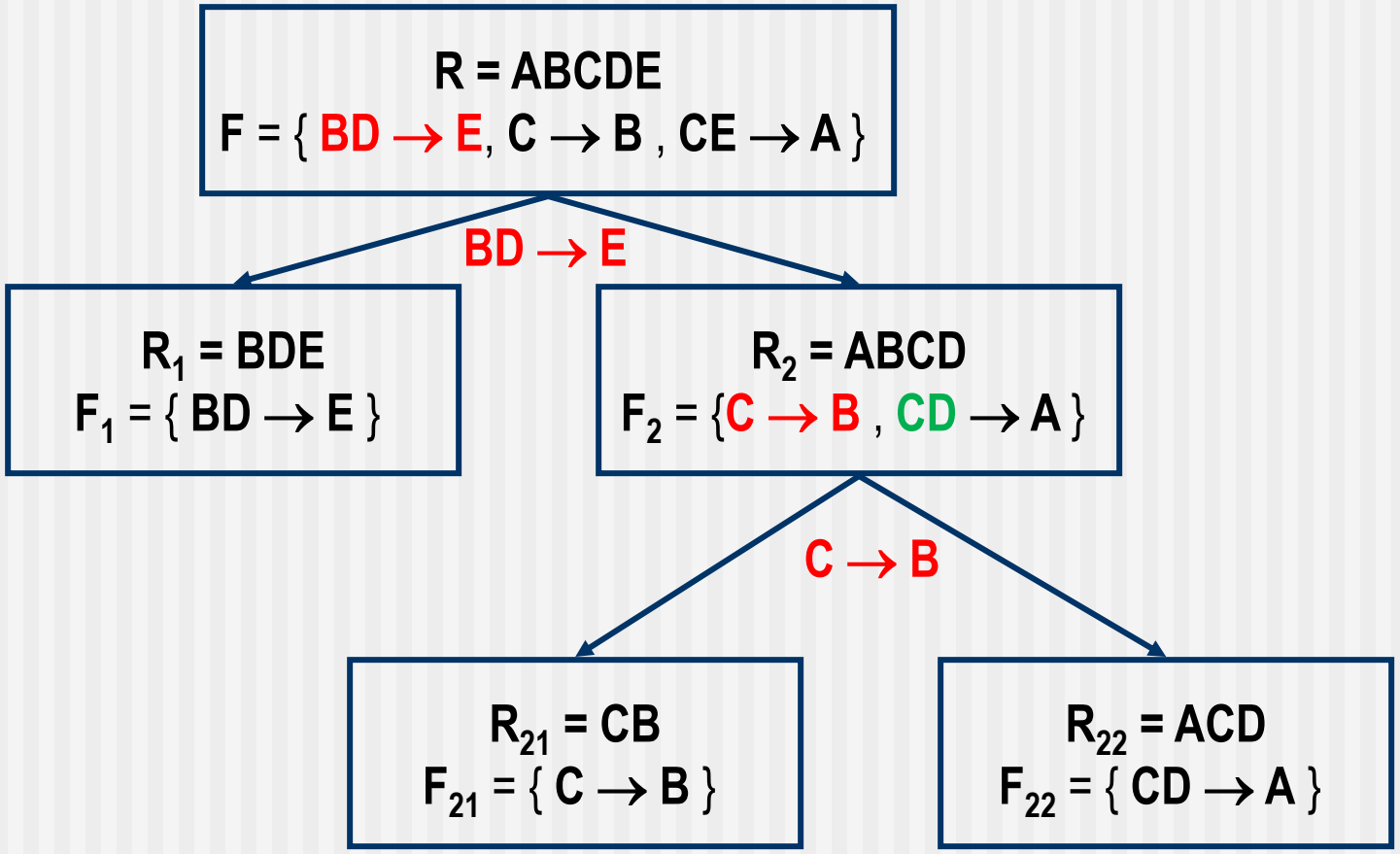
Decomposition into 3NF

- We can always obtain a lossless-join, dependency-preserving decomposition of a relation into 3NF relations. How?
- We discuss 2 solution approaches for 3NF decomposition.
- **Approach 1:** using the *binary decomposition* method.

Let **R** = { $\mathbf{R_1}$, $\mathbf{R_2}$, . . . $\mathbf{R_n}$} be the result. Recall that this is always lossless-join, but may not preserve all the FD's ➔ need to fix this!

- Identify the set **N** of FD's in **F** which we lost in the decomposition proc.
- For each FD $\mathbf{X \rightarrow A}$ in **N**, create a relation schema **XA** and add it to **R**
- A refinement step to avoid creating MANY relations: if there are several FD's with the same LHS, e.g., $\mathbf{X \rightarrow A_1}$, $\mathbf{X \rightarrow A_2}$, . . . , $\mathbf{X \rightarrow A_k}$, create **just one relation** with schema $\mathbf{XA_1 \ldots A_k}$

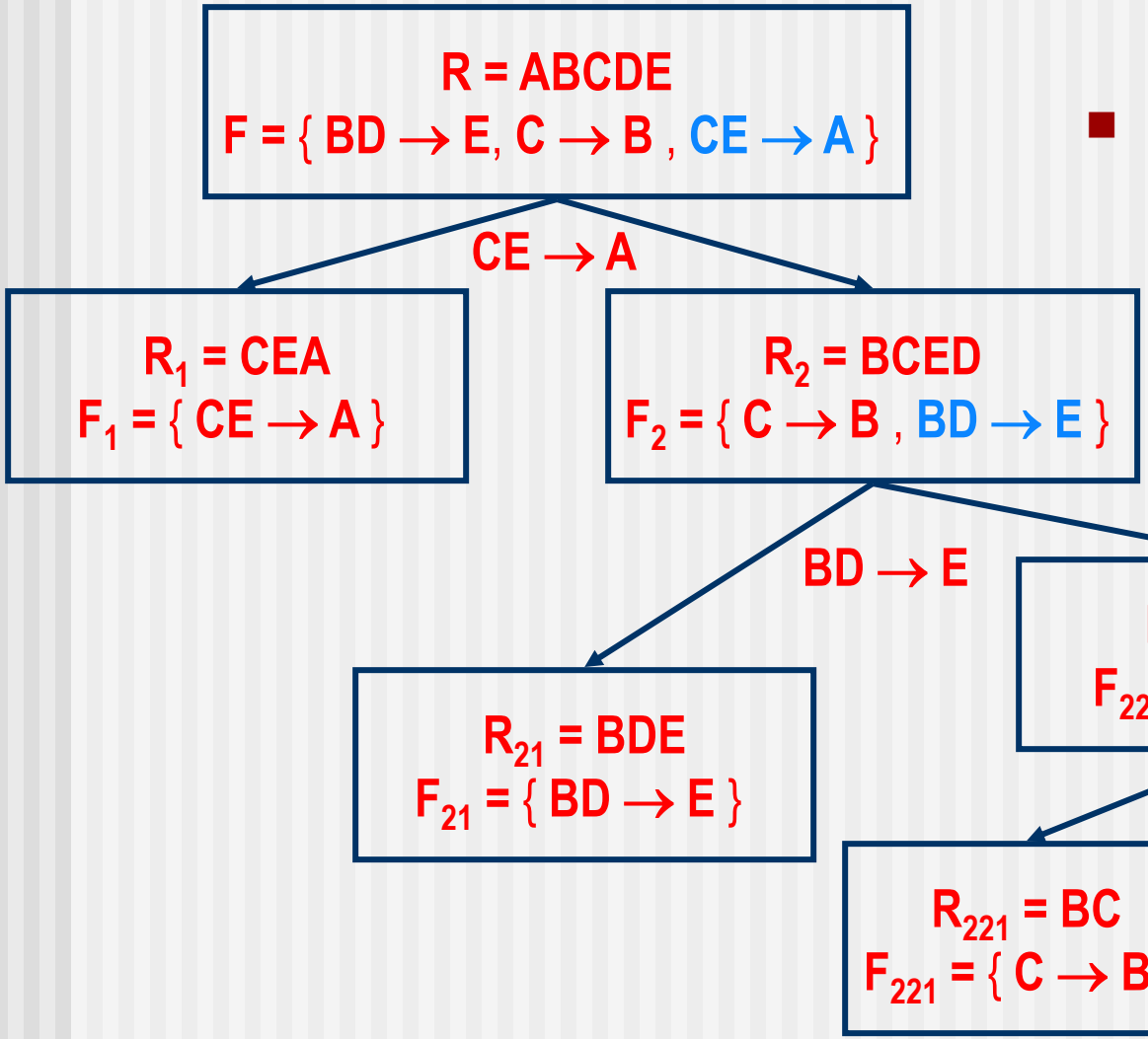
Example (3NF Decomposition)



- **CE → A** is not preserved, since **A ∉ {CE}⁺** w.r.t. $F_1 \cup F_{21} \cup F_{22}$

➔ To fix this, We add to **R₁** a new relation **R_3 = CEA** with **F_3 = {CE → A }**

Example (using a *different order*)



- This 3NF decomposition is dependency preserving, and of course lossless-join

Decomposition into 3NF

- **Method 1:** (binary decomposition):
  - Lossless-join √
  - May not be dependency preserving. If so, then add extra relations **XA**, for every FD **X** → **A** we lost
- **Method 2:** the *synthesis* approcah
  - Dependency preservation √
  - However, may not be lossless-join. If so, we must add to **R**, one extra relation that includes whose attributes form a key of **R**

    What would be the FDs on this newly added relation?

Decomposition into 3NF (Using the synthesis approach)

Consider <**R**, **F**>

- The synthesis approach:
  - Get a minimal cover $\mathbf{F^c}$ of **F**
  - For each FD $\mathbf{X \rightarrow A}$ in $\mathbf{F^c}$, add schema **XA** to **R**
  - If none of the relation schemas in **R** from previous step is a superkey for **R**, add a relation whose schema is a key for **R**

Example

- **R** = (**A**, **B**, **C**) with **F** = { **A** $\rightarrow$ **B**, **C** $\rightarrow$ **B** } is not in 3NF, so we decompose **R**. This yields:
- **R** = {R_1, R_2}, where R_1 = (**A**, **B**) with FDs F1={A $\rightarrow$ B} and R_2 = (**B**, **C**) with F2 = { C$\rightarrow$B}
- AC is the key for **R**, and since neither AB nor BC is a *superky* for R, we add R_3 = (**A**,**C**) to **R**.
- The decomposition **R** = {R_1, R_2, R_3} is both lossless and dependency-preserving. (The FD on AC is Ø).

The Chase test to check lossless join

Suppose relation **R**{A_1 , . . . , A_k} is decomposed into R_1,. . . , R_n
To determine if this decomposition is lossless, we use table
L[*1 … n*] [*1 . . . k*], and initialize it as follows.

Initializing the table:

for each relation R_i **do**
    for each attribute A_j **do**
        if A_j is an attribute in R_i
            then L [*i*][*j*] ← a_j
            else L [*i*][*j*] ← b_{ij}

Chase test (to check lossless join)

repeat

for each FD **X** $\rightarrow$ **Y** in **F do:**

for each pair of rows i and j in L that agree on X (that is, L [*i*] = L [*j*] for every attribute in X), modify every column t in L corresponding to attribute A_t in **Y** as follows:

if L [*i*][*t*] = a_t

then L [*j*][*t*] $\leftarrow$ a_t

else if L [*j*][*t*] = a_t

then L [*i*][*t*] $\leftarrow$ a_t

else L [*j*][*t*] $\leftarrow$ L [*i*][*t*]

until *no change*

The decomposition is lossless if, after this algorithm terminates, table L contains a row of all a's, that is, if there is a row *i* such that

L [*i*][*j*] = a_j for every column *j* corresponding to each attribute A_j in **R**

Examples

- Given $\prec \mathbf{R}, \mathbf{F} \succ$, where **R** = (**A**, **B**, **C**, **D**), and
  F = { $\mathbf{A} \rightarrow \mathbf{B}$, $\mathbf{A} \rightarrow \mathbf{C}$, $\mathbf{C} \rightarrow \mathbf{D}$ } is a set of FD's on **R**
- Is the decomposition $\underline{\mathbf{R}} = \{R_1, R_2\}$ lossless, where
  R_1 = (**A**, **B**, **C**) and R_2 = (**C** , **D**)?
  - To be discussed in class
- Now consider **S** = (**A**, **B**, **C**, **D**, **E**) with the FD's:
  G = { $\mathbf{AB} \rightarrow \mathbf{CD}$, $\mathbf{A} \rightarrow \mathbf{E}$, $\mathbf{C} \rightarrow \mathbf{D}$ }
- Is decomposition of $\underline{\mathbf{S}} = \{S_1, S_2, S_3\}$ lossless, where
  S_1 = (**A**, **B**, **C**), S_2 = (**B**, **C**, **D**), and S_3 = (**C**, **D**, **E**)?
  - To be discussed in class

Checking if a decomposition is Dependency-Preserving?

Inputs: Let $\prec \mathbf{R},\mathbf{F} \succ$, where $\mathbf{F} = \{\mathbf{X}_1 \rightarrow \mathbf{Y}_1, \ldots, \mathbf{X}_n \rightarrow \mathbf{Y}_n\}$.

Suppose $\underline{\mathbf{R}} = \{ \mathbf{R}_1, \ldots, \mathbf{R}_k \}$ is a decomposition of $\mathbf{R}$

and $\mathbf{F}_i$ is the projection of $\mathbf{F}$ on schema $\mathbf{R}_i$

Method:

preserved $\leftarrow$ *TRUE*

for each FD $\mathbf{X} \rightarrow \mathbf{Y}$ in $\mathbf{F}$ and while preserved = *TRUE*

do compute $\mathbf{X}^+$ under $\mathbf{F}_1 \cup \ldots \cup \mathbf{F}_k$;

if $\mathbf{Y} \not\subseteq \mathbf{X}^+$ then {*preserved* $\leftarrow$ *FALSE; exit* };

end

Example

- Consider **R** = (**A**, **B**, **C**, **D**), **F** = { **A** → **B**, **B** → **C**, **C** → **D** }
- Is the decomposition $\underline{R}$ = {R_1, R_2} dependency-preserving, where

  R_1 = (**A**, **B**), F_1 = { **A** → **B** }, R_2 = (**A**, **C** , **D**), AND F_2 = { **C** → **D**, **A** → **D**, **A** → **C** }?
  - Check if **A** → **B** is preserved
    - Compute A^+ under { **A** → **B** } ∪ { **C** → **D**, **A** → **D**, **A** → **C** }
      - A^+ = { **A**, **B**, **C**, **D**}
      - Check if **B** ∈ A^+
      - **Yes**
    - **A** → **B** is preserved
  - Check if **B** → **C** is preserved
    - Compute B^+ under { **A** → **B** } ∪ { **C** → **D**, **A** → **D**, **A** → **C** }
      - B^+ = { **B** }
    - Check if **C** ∈ B^+
      - **No**
    - **B** → **C** is not preserved

➔The decomposition is not dependency-preserving