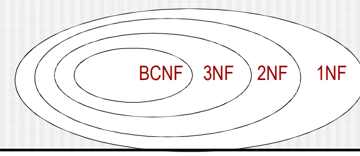
Normal Forms

- If a relation schema is in a normal forms, we know that it is in some particular shape/health in the sense that *certain kinds of problems (related to redundancy) cannot arise*
- Given a relation schema **R**, we need to be able to check if it is in certain normal form. If not, we need to be able to decompose it into smaller such normal relations. How?
- To address these issues, we need to study **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)
- These normal forms form the hierarchy:



Third Normal Form (3NF)

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

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

➔ To determine if **R** with FD **F** is in 3NF:
  - 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 **X** → **A** in **F**, at least one of the following conditions holds:
  - **A** ∈ **X**, that is, **X** → **A** is a trivial FD, or
  - **X** is a superkey
- To determine if **R** with **F** is in BCNF:

  That is, for every nontrivial FD, check if its LHS **X** is a superkey.

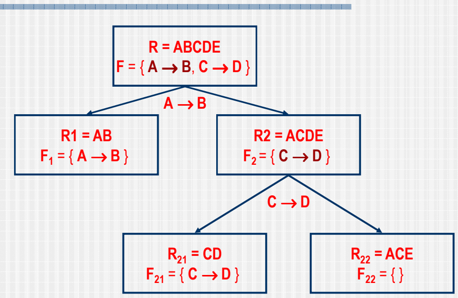
  That is, check if **X**$^+$ = **R.**

Decomposition into BCNF

- Consider <**R**, **F**>, where **R** is in 1NF.
- If **R** is not in BCNF, 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 algorithm (done recursively):

  Pick every FD **X** → **A** ∈ **F** that violates the BCNF requirement:
  1. Decompose **R** into **XA** and **R – A**
  2. If either **R–A** or **XA** is not in BCNF, decompose it further

Example (Decomposition into BCNF relations)



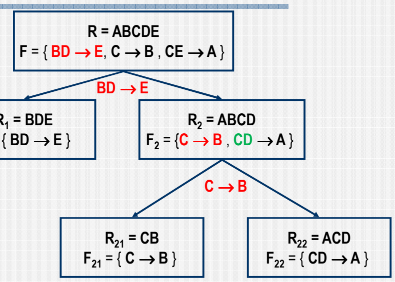
Decomposition into 3NF

- 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** = { R_1, R_2, . . . 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 **X** ⟶ **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., **X** ⟶ A_1, **X** ⟶ A_2, . . . , **X** ⟶ A_k, create **just one relation** with schema $XA_1 \ldots A_k$

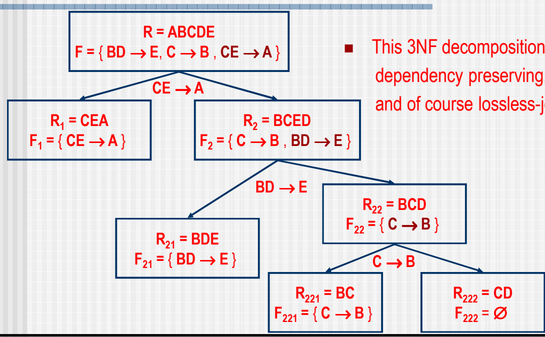
Example (3NF Decomposition)



- **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 **R**, 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

- 1^{ST} approach (binary decomposition):
  - Lossless-join √
  - May not be dependency preserving. If so, then add extra relations **XA**, for every FD **X** ⟶ **A** we lost
- **Approach 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 F^c of **F**
  - For each FD **X** ⟶ **A** in F^c, add schema **XA** to **R**
  - If the decomposition **R** is not lossless,
    add to **R** an extra relation containing any key of **R**

Example

- **R** = (**A**, **B**, **C**)
- **F** = { **A** ⟶ **B**, **C** ⟶ **B** }
- Decompose R into R_1 = (**A**, **B**) and R_2 = (**B**, **C**)
- This decomposition is not lossless
  ➔ Add R_3 = (**A**, **C**)
- The decomposition **R** = {R_1, R_2, R_3} is both lossless and dependency-preserving

An Algorithm to Check Lossless join

Suppose relation $R\{A_1, \ldots, A_k\}$ is decomposed into $R_1, \ldots, R_n$
To determine if this decomposition is lossless, we use a table,
$L[1 \ldots n][1 \ldots k]$

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 $L[i][j] \leftarrow a_j$
      else $L[i][j] \leftarrow b_{ij}$

13

Algorithm to Check Lossless (cont'd)

repeat
  for each FD $X \rightarrow Y$ in **F do:**
    if $\exists$ rows i and j such that $L[i] == L[j]$, for each attribute in **X**,
      then for $\forall$ column t corresponding to an attribute A_t in **Y do:**
        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 performing this algorithm, L contains a row of all a's. That is, if there exists a row *i* in L such that: $L[i][j] == a_j$ for every column *j* corresponding to each attribute A_j in **R**

14

Examples

- Given <**R**,**F**>, where **R** = (**A**, **B**, **C**, **D**), and **F** = { **A** → **B**, **A** → **C**, **C** → **D** } is a set of FD's on **R**
- Is the decomposition **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**) and the set G of FD's on S, where **G** = { **AB** → **CD**, **A** → **E**, **C** → **D** }
- Is decomposition of **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

15

Checking if a decomposition is Dependency-Preserving?

Inputs: Let <**R**,**F**>, where **F** = {$X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n$}.
  Suppose **R** = { $R_1, \ldots, R_k$ } is a decomposition of **R**
  and F_i is the projection of **F** on schema R_i

Method:
  preserved ← *TRUE*
  for each FD **X** → **Y** in **F** and while preserved == *TRUE*
  do compute X^+ under $F_1 \cup \ldots \cup F_k$;
    if **Y** ⊄ X^+ then {*preserved* ← *FALSE*; *exit* };
  end

16

Example

- Consider **R** = (**A**, **B**, **C**, **D**), **F** = { **A** → **B**, **B** → **C**, **C** → **D** }
- Is the decomposition **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

17