Lossless Schema Decomposition

- A decomposition should not lose information
- A decomposition ($\mathbf{R}_1,\ldots,\mathbf{R}_n$) of a schema, **R**, is *lossless* if every valid instance, **r**, of **R** can be reconstructed from its components:

$$\mathbf{r} = \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

- where each $\mathbf{r}_i = \Pi_{\mathbf{R}i}(\mathbf{r})$

Lossy Decomposition

The following is always the case (Think why?):

$$\mathbf{r} \subseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

But the following is not always true:

$$\mathbf{r} \supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

Example: $\mathbf{r} \not\supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$

SSN	*Name*	*Address*
1111	Joe	1 Pine
2222	Alice	2 Oak
3333	Alice	3 Pine

SSN	*Name*
1111	Joe
2222	Alice
3333	Alice

Name	*Address*
Joe	1 Pine
Alice	2 Oak
Alice	3 Pine

The tuples (222, Alice, 3 Pine) and (333, Alice, 2 Oak) *are in the join, but not in the original*

Lossy Decompositions: *What is Actually Lost*?

- In the previous example, the tuples (222, Alice, 3 Pine) and (333, Alice, 2 Oak) were *gained*, not lost! Why do we say that the decomposition was lossy?
- What was lost is *information*:
  - That 222 lives at 2 Oak: *In the decomposition, 222 can live at either 2 Oak or 3 Pine*
  - That 333 lives at 3 Pine: *In the decomposition, 333 can live at either 2 Oak or 3 Pine*

Testing for Losslessness

- A (binary) decomposition of $\mathbf{R} = (R, F)$ into $\mathbf{R}_1 = (R_1, F_1)$ and $\mathbf{R}_2 = (R_2, F_2)$ is lossless *if and only if* :
  - either the FD
    - $(R_1 \cap R_2) \rightarrow R_1$ is in F^+
  - or the FD
    - $(R_1 \cap R_2) \rightarrow R_2$ is in F^+

Example

Schema *(R, F)* where

$R = \{SSN, Name, Address, Hobby\}$

$F = \{SSN \rightarrow Name, Address\}$

can be decomposed into

$R_1 = \{SSN, Name, Address\}$

$F_1 = \{SSN \rightarrow Name, Address\}$

and

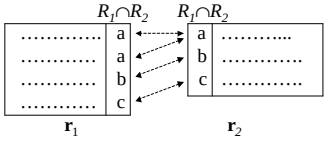
$R_2 = \{SSN, Hobby\}$

$F_2 = \{\ \}$

Since $R_1 \cap R_2 = SSN$ and $SSN \rightarrow R_1$ the decomposition is lossless

Intuition for Test for Losslessness

- Suppose $R_1 \cap R_2 \rightarrow R_2$. Then a row of $\mathbf{r}_1$ can combine with exactly one row of $\mathbf{r}_2$ in the natural join (since in $\mathbf{r}_2$ a particular set of values for the attributes in $R_1 \cap R_2$ defines a unique row)



Proof of Lossless Condition

- $\mathbf{r} \subseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$ – *this is true for any decomposition*
- $\mathbf{r} \supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2$

  If $R_1 \cap R_2 \rightarrow R_2$ then

  card $(\mathbf{r}_1 \bowtie \mathbf{r}_2)$ = *card* $(\mathbf{r}_1)$

  (because each row of r_1 joins with exactly one row of r_2)

  But *card* $(\mathbf{r}) \geq$ *card* $(\mathbf{r}_1)$ (*since* $\mathbf{r}_1$ *is a projection of* $\mathbf{r}$) and therefore *card* $(\mathbf{r}) \geq$ *card* $(\mathbf{r}_1 \bowtie \mathbf{r}_2)$

  Hence $\mathbf{r} = \mathbf{r}_1 \bowtie \mathbf{r}_2$

40

Dependency Preservation

- Consider a decomposition of $\mathbf{R} = (R, F)$ into $\mathbf{R}_1 = (R_1, F_1)$ and $\mathbf{R}_2 = (R_2, F_2)$
  - An FD $X \rightarrow Y$ of F is in F_i iff $X \cup Y \subseteq R_i$
  - An FD, $f \in F$ may be in neither F_1, nor F_2, nor even $(F_1 \cup F_2)^+$
    - Checking that f is true in $\mathbf{r}_1$ or $\mathbf{r}_2$ is (relatively) easy
    - Checking f in $\mathbf{r}_1 \bowtie \mathbf{r}_2$ is harder – requires a join
    - *Ideally*: want to check FDs locally, in $\mathbf{r}_1$ and $\mathbf{r}_2$, and have a guarantee that every $f \in F$ holds in $\mathbf{r}_1 \bowtie \mathbf{r}_2$
- The decomposition is *dependency preserving* iff the sets F and $F_1 \cup F_2$ are equivalent: $F^+ = (F_1 \cup F_2)^+$
  - Then checking all FDs in F, as $\mathbf{r}_1$ and $\mathbf{r}_2$ are updated, can be done by checking F_1 in $\mathbf{r}_1$ and F_2 in $\mathbf{r}_2$

41

Dependency Preservation

- If f is an FD in F, but f is not in $F_1 \cup F_2$, there are two possibilities:
  - $f \in (F_1 \cup F_2)^+$
    - If the constraints in F_1 and F_2 are maintained, f will be maintained automatically.
  - $f \notin (F_1 \cup F_2)^+$
    - f can be checked only by first taking the join of r_1 and r_2. This is costly.

42

Example

Schema *(R, F)* where

R = {SSN, Name, Address, Hobby}

F = {SSN → Name, Address}

can be decomposed into

R_1 *= {SSN, Name, Address}*

F_1 *= {SSN → Name, Address}*

and

R_2 *= {SSN, Hobby}*

F_2 *= { }*

Since $F = F_1 \cup F_2$ the decomposition is dependency preserving

43

Example

- Schema: (*ABC; F*) , $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$
- Decomposition:
  - (*AC*, F_1), $F_1 = \{A \rightarrow C\}$
    - Note: $A \rightarrow C \notin F$, but in F^+
  - (*BC*, F_2), $F_2 = \{B \rightarrow C, C \rightarrow B\}$
- $A \rightarrow B \notin (F_1 \cup F_2)$, **but** $A \rightarrow B \in (F_1 \cup F_2)^+$.
  - So the decompositions is still dependency preserving

44

Example

- HasAccount (*AccountNumber, ClientId, OfficeId*)
- Functional dependencies:
  - f_1: *AccountNumber → OfficeId*
  - f_2: *ClientId, OfficeId → AccountNumber*
- Decomposition:
  - AcctOffice = (*AccountNumber, OfficeId*; {*AccountNumber → OfficeId*})
  - AcctClient = (*AccountNumber, ClientId*; {})
- Decomposition is lossless: $R_1 \cap R_2 =$ {*AccountNumber*} and *AccountNumber → OfficeId*
- Decomposition is not dependency preserving: $f_2 \notin (F_1 \cup F_2)^+$

BCNF Decomposition Algorithm

***Input*: R** = (*R; F*)

Decomp := **R**
while there is **S** = (*S; F'*) ∈ *Decomp* and **S** not in BCNF **do**
  Find $X \rightarrow Y \in F'$ that violates BCNF // *X isn't a superkey in* **S**
  Replace **S** in *Decomp* with $\mathbf{S_1} = (XY; F_1)$, $\mathbf{S_2} = (S - (Y - X); F_2)$
  // F_1 = *all FDs of F' involving only attributes of XY*
  // F_2 = *all FDs of F' involving only attributes of S - (Y - X)*
end
return *Decomp*

46

Example

Given: **R** = (*R; T*) where *R = ABCDEFGH* and
T = {ABH→ C, A→ DE, BGH→ F, F→ ADH, BH→ GE}
step 1: Find a FD that violates BCNF.
Not $ABH \rightarrow C$ since $(ABH)^+$ includes all attributes
(*BH* is a key)
$A \rightarrow DE$ violates BCNF since *A* is not a superkey ($A^+ = ADE$)
step 2: Split **R** into:
$\mathbf{R_1}$ = (*ADE, {A→ DE }*)
$\mathbf{R_2}$ = (*ABCFGH; {ABH→ C, BGH→ F, F→ AH , BH→ G}*)
Note 1: $\mathbf{R_1}$ is in BCNF.
Note 2: Decomposition is *lossless* since *A* is a key of $\mathbf{R_1}$.
Note 3: FDs $F \rightarrow D$ and $BH \rightarrow E$ are not in T_1 or T_2. But both can be derived from $T_1 \cup T_2$.
(*E.g.*, $F \rightarrow A$ and $A \rightarrow D$ implies $F \rightarrow D$)
Hence, decomposition is *dependency preserving*.

47

Properties of BCNF Decomposition Algorithm

Let $X \rightarrow Y$ violate BCNF in **R** = (*R,F*) and $\mathbf{R_1} = (R_1, F_1)$, $\mathbf{R_2} = (R_2, F_2)$ is the resulting decomposition. Then:

- There are *fewer violations* of BCNF in $\mathbf{R_1}$ and $\mathbf{R_2}$ than there were in **R**
  - $X \rightarrow Y$ implies *X* is a key of $\mathbf{R_1}$
  - Hence $X \rightarrow Y \in F_1$ does not violate BCNF in $\mathbf{R_1}$ and, since $X \rightarrow Y \notin F_2$, does not violate BCNF in $\mathbf{R_2}$ either
  - Suppose f is $X' \rightarrow Y'$ and $f \in F$ doesn't violate BCNF in **R**. **I**f $f \in F_1$ or F_2 it does not violate BCNF in $\mathbf{R_1}$ or $\mathbf{R_2}$ either since *X'* is a superkey of **R** and hence also of $\mathbf{R_1}$ and $\mathbf{R_2}$.
- The decomposition is *lossless*
  - Since $F_1 \cap F_2 = X$

48

Example (con't)

Given: $\mathbf{R_2}$ = (*ABCFGH; {ABH→C, BGH→F, F→AH, BH→G}*)
step 1: Find a FD that violates BCNF.
Not $ABH \rightarrow C$ or $BGH \rightarrow F$, since *BH* is a key of $\mathbf{R_2}$
$F \rightarrow AH$ violates BCNF since *F* is not a superkey ($F^+ = AH$)
step 2: Split $\mathbf{R_2}$ into:
$\mathbf{R_{21}}$ = (*FAH, {F → AH}*)
$\mathbf{R_{22}}$ = (*BCFG; {}*)

Note 1: Both $\mathbf{R_{21}}$ and $\mathbf{R_{22}}$ are in BCNF.
Note 2: The decomposition is *lossless* (since *F* is a key of $\mathbf{R_{21}}$)
Note 3: FDs $ABH \rightarrow C$, $BGH \rightarrow F$, $BH \rightarrow G$ are not in T_{21} or T_{22}, and they cannot be derived from $T_1 \cup T_{21} \cup T_{22}$.
Hence the decomposition is *not* dependency-preserving

49

Properties of BCNF Decomposition Algorithm

- A BCNF decomposition is *not necessarily* dependency-preserving
- But *always* lossless

50