Lossless Join Decomposition (XI)

- The goal is to use F to prove that $t \in r$ holds
- Strategy of the Chase test
  - We *chase* the matrix by applying the FDs in F to equate symbols in the matrix whenever possible
  - If we manage to obtain a row that is equal to t (that is, the row only contains unsubscripted letters), we have proved that any tuple t in the join of the projections was actually a tuple of R
- Example
  - We continue our previous example and assume the set $F = \{A \rightarrow B, B \rightarrow C, CD \rightarrow A\}$ of FDs
  - The FDs in F can be applied in any order and several times

A	B	C	D
a	b_1	c_1	d
a	b_2	c	d_2
a_3	b	c	d

$\xrightarrow{A \rightarrow B}$

A	B	C	D
a	b_1	c_1	d
a	b_1	c	d_2
a_3	b	c	d

Lossless Join Decomposition (XII)

$\xrightarrow{B \to C}$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a_3	b	c	d

$\xrightarrow{CD \to A}$

A	B	C	D
a	b_1	c	d
a	b_1	c	d_2
a	b	c	d

- The last row has become equal to *t*
- We have shown that if *r* satisfies the FDs $A \to B$, $B \to C$, and $CD \to A$, then whenever we project *r* onto {*A*, *D*}, {*A*, *C*}, and {*B*, *C*, *D*} and rejoin, what we get must have been in *r*

- Example
  - Consider the relation $R(A, B, C, D)$ with $F = \{B \to AD\}$ and the decomposition {*A*, *B*}, {*B*, *C*}, and {*C*, *D*}. Apply the Chase test.

A	B	C	D
a	b	c_1	d_1
a_2	b	c	d_2
a_3	b_3	c	d

$\xrightarrow{B \to AD}$

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

There is no row that is fully unsubscripted. The decomposition is lossy.

Lossless Join Decomposition (XIII)

- Example (*continued*)
  - Another way to show this is: Treat the right table as a relation with three tuples, decompose it, and then rejoin

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a_3	b_3	c	d

$\neq$

A	B
a	b
a	b
a_3	b_3

$\bowtie$

B	C
b	c_1
b	c
b_3	c

$\bowtie$

C	D
c_1	d_1
c	d_1
c	d

=

A	B	C	D
a	b	c_1	d_1
a	b	c	d_1
a	b	c	d
a_3	b_3	c	d_1
a_3	b_3	c	d

Dependency Preservation (I)

- For performance reasons it would be useful if each FD in F either could be checked directly in one of the relation schemas R_i of the decomposition, or could be inferred from the FDs that hold on the attributes of some schema R_i
- If one of the FDs is not represented in some schema R_i of the decomposition, we cannot check and enforce this constraint on a single relation but have to join multiple relations in order to include all attributes involved in that FD
- Given a set F of FDs on a relation schema R and a decomposition $R_1, \ldots, R_n$ of R, the restriction F_{R_i} of F to R_i is defined as $F_{R_i} = \{X \to Y \in F^+ \mid X \cup Y \subseteq R_i\}$
- All FDs of F_{R_i} can be checked *locally* on R_i *alone* (without the need of joins)
- All FDs in F^+ are used, not only the FDs in F
- Example
  - Suppose we have $R(A, B, C)$, $F = \{A \to B, B \to C\}$, $R_1(A, C)$, $R_2(A, B)$
  - F_{R_1} includes $A \to C$ since $A \to C \in F^+$ but $A \to C \notin F$

Dependency Preservation (II)

- Question: Is testing only the restrictions sufficient?
- $F_r = F_{R_1} \cup F_{R_2} \cup \ldots \cup F_{R_n}$ is a set of FDs on R, i.e., $F_r \subseteq F^+$
- In general, $F_r \neq F$ holds
- Even if $F_r \neq F$ holds, it can be that $F_r^+ = F^+$ holds, i.e., $F_r \equiv F$
- If $F_r^+ = F^+$ holds, then every FD in F is logically implied by F_r
- A decomposition having the property $F_r^+ = F^+$ is a dependency-preserving decomposition
- If each FD in F can be tested on one of the relation schemas of the decomposition, the decomposition is dependency-preserving
- The following algorithm tests for dependency preservation in general
- It is expensive since it requires the computation of F^+

Dependency Preservation (III)

- Example: Let $R(A, B, C)$ and $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$. Let $R_1(A, B)$ and $R_2(B, C)$ be a decomposition of R. Check whether the decomposition is dependency-preserving.
- It is easy to see that $A \rightarrow B \in F_{R_1}$ and $B \rightarrow C \in F_{R_2}$
- The question is whether the decomposition preserves the FD $C \rightarrow A$
- Further question: Does the fact that A and C are contained together neither in R_1 nor in R_2 mean that the decomposition is not dependency-preserving?
- F^+ includes F but also other FDs such as $A \rightarrow C$, $B \rightarrow A$, and $C \rightarrow B$ (the latter three FDs are obtained by transitivity)
- This means that $B \rightarrow A \in F_{R_1}$ and $C \rightarrow B \in F_{R_2}$
- In summary, $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$, $C \rightarrow B \in F_{R_1} \cup F_{R_2}$ holds
- Consequently, $C \rightarrow A \in (F_{R_1} \cup F_{R_2})^+$ holds due to the transitivity axiom applied to $C \rightarrow B$ and $B \rightarrow A$
- Hence, the decomposition of R into $R_1(A, B)$ and $R_2(B, C)$ with $F_{R_1} = \{A \rightarrow B, B \rightarrow A\}$ and $F_{R_2} = \{B \rightarrow C, C \rightarrow B\}$ preserves the FD $C \rightarrow A$

Dependency Preservation (IV)

- Algorithm 1 for testing dependency preservation

bool IsDependencyPreserving1($\{R_1, R_2, \ldots, R_n\}, F$)
// Input: (1) A decomposition of R into the relation schemas $R_1, R_2, \ldots, R_n$
// (2) A set F of FDs on R
// Output: *true*, if the decomposition is dependency preserving under F;
// *false*, otherwise

// Step 1: Compute the closure of F
F^+ := *CalculateFDClosure*(F)

// Step 2: Compute the restrictions of F^+ to the R_i
for each i in 1..n **do**
$F_{R_i} := \varnothing$
for each $X \rightarrow Y \in F^+$ **do**
if $X \cup Y \subseteq R_i$ **then** $F_{R_i} := F_{R_i} \cup \{X \rightarrow Y\}$

Dependency Preservation (V)

- Algorithm 1 for testing dependency preservation (*continued*)

  // Step 3: Form the union of all restrictions

  $F_r := \emptyset$

  for each i in 1..n **do**

  $\quad F_r := F_r \cup F_{R_i}$

  // Step 4: Compute the closure of F_r

  $F_r^+ := \mathit{CalculateFDClosure}(F_r)$

  // Step 5: Check if the two closures are equal

  return $(F_r^+ = F^+)$

Dependency Preservation (VI)

- Algorithm 2 for testing dependency preservation without computing F^+

bool IsDependencyPreserving2($\{R_1, R_2, \ldots, R_n\}, F$)

for each $X \to Y \in F$ **do**

  Result := X

  repeat

    OldResult := *Result*

    // Compute the attribute closure of *Result* under F_r

    for each i in 1..n **do**

      // Compute the attribute closure of *Result* under F_{R_i}

      C := *CalculateAttributeClosure*(F, *Result* $\cap R_i$) $\cap R_i$

      Result := *Result* $\cup$ C

  until *OldResult* = *Result*

  if $Y \cap$ *Result* $\neq Y$ **then return** *false*   // FD $X \to Y$ is not preserved

return *true*   // All FDs in F are preserved

Dependency Preservation (VII)

- Ideas behind Algorithm 2
  - Test each FD $X \rightarrow Y$ in F to see if it is preserved in F_r (as the union of all restrictions F_{R_i})
  - For this purpose, compute the attribute closure of X under F_r, and check whether it includes Y
    - This is done without first computing F_r explicitly since this is quite expensive
    - The statement *CalculateAttributeClosure*(F, *Result* $\cap R_i$) $\cap R_i$ computes the attribute closure of *Result* under F_{R_i}
    - Reasons:
      - For any $A \subseteq R_i$, $A \rightarrow A^+ \in F^+$, and $A \rightarrow A^+ \cap R_i \in F_{R_i}$
      - Conversely, if $A \rightarrow B \in F_{R_i}$ holds, then $B \subseteq A^+ \cap R_i$
  - The decomposition is dependency-preserving if all FDs in F are preserved
  - Algorithm has a polynomial runtime complexity

Universal Relation Assumption (I)

- The transformation of an E-R diagram into a set of relation schemas already represents an anticipated decomposition of the database schema
- But we have learned that checking whether each individual relation schema of a given decomposition satisfies a desired normal form does not guarantee a good database design
- The anticipated decomposition itself could already have a problem
- A mandatory requirement is lossless join property of the decomposition
- An optional requirement is dependency preservation of the decomposition
- In order to avoid problems and a bad database design, we have to make the universal relation assumption: Each normalization algorithm starts from a *single* universal relation schema $R(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$, which includes *all* the attributes of the database schema, and the set F of FDs on R
- The universal relation assumption contributes to the correctness criteria

Universal Relation Assumption (II)

- This means for the relation schemas R_1, R_2, …, R_n obtained as the result of the transformation of an E-R diagram into relation schemas:
  - Take back the decomposition
  - If there are attributes in different relation schemas with the *same* name, make them *unique* by renaming
  - Merge the relation schemas R_1, R_2, …, R_n into the universal relation R, i.e., $R = \bigcup_{i=1}^{n} R_i$

- This does not make the E-R modeling process redundant since
  - the E-R diagram allows us to obtain an overview of the relevant data (i.e., entities, relationships, attributes) that have to be stored later in the database
  - each relation schema R_i is "represented" by the FD $K_i \rightarrow R_i$ in F if K_i is the primary key of R_i

- Additional FDs can and will lead to a different decomposition of R