Decompositions

Definition
The decomposition of a schema $R=A_1\ldots A_n$ is its replacement by a collection $D_R = \{R_1, R_2, \ldots, R_m\}$ of subsets of $R$ such that $R = R_1 \cup R_2 \cup \ldots \cup R_m$

Note: schemas $R_i$'s do not have to be disjoint!

Example 1
Assume the schema $R=ABCD$. The following are possible decompositions of $R$.

- $D_1 = \{AB, CD\}$
- $D_2 = \{AB, ACD\}$
- $D_3 = \{A, BCD\}$
- $D_4 = \{AB, BC, CD, AD\}$
Dependency Preservation Property of a Decomposition

**Observations**

- It would be useful if each functional dependency $X \rightarrow Y$ specified in $F$ either appeared directly in one of the relation schemas $R_i$ in the decomposition $\rho$ or could be inferred from the dependencies that appear in some $R_j$.

- We want to preserve the dependencies because each rule in $F$ represents a constraint on the database.

- If one of the dependencies is not represented in some individual relation $R_i$ of the decomposition, we cannot enforce this constraint by dealing with an individual relation. We may have to join multiple relations so as to include all attributes involved in that dependency.

- It is not necessary that the exact dependencies specified in $F$ appear themselves in individual relations of the decomposition $D$. It is sufficient that the union of the dependencies that hold on the individual relations in $D$ be equivalent to $F$.

**Definition**

Given a set of dependencies $F$ on $R$, the projection of $F$ on $R_i$, denoted by $\Pi_{R_i}(F)$ where $R_i$ is a subset of $R$, is the set of dependencies $X \rightarrow Y$ in $F^+$ such that the attributes in $X \cup Y$ are all contained in $R_i$.

Hence, the projection of $F$ on each relation schema $R_i$ in the decomposition $D$ is the set of functional dependencies in $F^+$ such that all their left- and right-hand-side attributes are in $R_i$.

We say that a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of $R$ is *dependency-preserving* with respect to $F$ if the union of the projections of $F$ on each $R_i$ in $D$ is equivalent to $F$; that is,

$$\left( \Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \ldots \cup \Pi_{R_m}(F) \right)^+ = F^+$$
Dependency Preservation Property of a Decomposition

Claim
It is always possible to find a dependency-preserving decomposition $D$ with respect to $F$ such that each relation is in 3NF (to be discussed later).

Example 2a
Assume $R = ABCD$, and $F = \{A \rightarrow B, C \rightarrow D\}$. The decomposition $D_1 = \{AB, CD\}$ is clearly $F$-preserving. Observe the first rule is kept in $R_1$, while the second is preserved in $R_2$. Also notice that schemas $AB$ and $CD$ are in 3NF format.

\[
\begin{array}{c|c|c|c|c}
A & B & C & D \\
\hline
F = \{A \rightarrow B, C \rightarrow D\}
\end{array}
\]

\[
\begin{array}{c|c|c}
A & B & C & D \\
\hline
F_1 = \{A \rightarrow B\} & F_2 = \{C \rightarrow D\}
\end{array}
\]

Dependency Preservation Property of a Decomposition

Example 2b
Assume $R = ABCD$, and $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$. Evaluate the decomposition $D = \{ABC, CD\}$.

$D$ is clearly $F$-preserving. Observe that

\[
F_1 = \prod_{ABC}(F) = \{A \rightarrow B, B \rightarrow C\}
\]

\[
F_2 = \prod_{CD}(F) = \{C \rightarrow D\}
\]

and $F = F_1 \cup F_2$
Dependency Preservation Property of a Decomposition

**Algorithm:** Testing Preservation of Functional Dependencies

**Input:** A set $F$ of FDs on schema $R$, a partition $D = (R_1 \ldots R_m)$ of $R$, and a dependency $X \rightarrow Y$ with $R \supseteq XY$.

**Output:** $true$ whenever the dependency $X \rightarrow Y$ is retained by the decomposition $D$, i.e. $\cup \{ \Pi_{R_1} F \} \Rightarrow X \rightarrow Y$, and $false$ otherwise.

**Method:**

```
begin
  Z = X;
  while changes to Z occur do
    for i = 1 to m do
      Z = Z \cup \left( (Z \cap R_i)^+ \cap R_i \right);
    if (Z \supseteq Y) then return (true);
    else return (false);
end;
```

Example 2c. *source Ullman, J. "Database Systems", Computer Sc. Press*

Consider schema $R=ABCD$, $D = \{ R_1=AB, R_2=BC, R_3=CD \}$ subjected to $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$.

**Question:** Is $D \rightarrow A$ lost in the projections?
**Answer:** Even though there is no segment containing attributes $DA$ the rule $D \rightarrow A$ is *not* lost ($Z_3=ABCD \supseteq A$).

**Why?** Let’s use the $F$-Preserving Test Algorithm. If $D$ preserves $F$ it must occur that $\Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \Pi_{R_3}(F) \Rightarrow D \rightarrow A$. Observe that the other three rules in $F$ are naturally retained in $R_1$, $R_2$, and $R_3$.

Let’s begin with $Z = D$ (this the LHS of $D \rightarrow A$):

1. Use $R_1=AB$. Here $Z = D \cup [(D \cap AB)^+ \cap AB] = D$
2. Use $R_2=BC$. Here $Z = D \cup [(D \cap BC)^+ \cap BC] = D$
3. Use $R_3=CD$. Here $Z = D \cup [(D \cap CD)^+ \cap CD] = DC$ (a change!)
4. Use $R_2=BC$. Here $Z = D \cup [(DC \cap BC)^+ \cap BC] = DCB$ (a change!)
5. Use $R_1=AB$. Here $Z = DCB \cup [(DCB \cap AB)^+ \cap AB] = DCBA \supseteq A$. Stop! Rule is kept in $D$
**Dependency Preservation Property of a Decomposition**

**Example 2c (cont.)* Ullman, J. "Database Systems", Computer Sc. Press**

Consider schema R=ABCD, D = \{R_1= AB, R_2= BC, R_3= CD \} subjected to F = \{ A→B, B→C, C→D, D→A \}.

**Observation:**
The rule D→A is preserved in the decomposition (R_1, R_2, R_3)

Although not obvious it is clear that the following FDs are in F⁺
F⁺ ⊇ \{ A→B, B→C, C→D, D→A, B →A, C →D, D →C \}

Therefore
F1 = \{ A→B, B →A \} on R1=(AB)
F2 = \{ B→C, C →B \} on R2=(BC)
F3 = \{ C→D, D →C \} on R3=(CD)

Finally
(F1 ∪ F2 ∪ F3)* derives the FD D→A and consequently it is not lost.

---

**Nonadditive (Lossless) Join Property of a Decomposition**

**Observation**

Another property that a decomposition D should possess is the *nonadditive* join property, which ensures that no *spurious* (phantom) tuples are generated when a NATURAL JOIN operation is applied to the relations in the decomposition.

**Definition.**

Formally, a decomposition D = \{R_1, R_2, \ldots, R_m\} of R *has the lossless* (nonadditive) join property with respect to the set of dependencies F on R, if for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all the relations in D:

\[ * (\pi_{R_1}(r), \ldots, \pi_{R_m}(r)) = r. \]

The word *loss* in *lossless* refers to *loss of information*, not to loss of tuples. If a decomposition does not have the lossless join property, we may get additional spurious tuples.

The *nonadditive* join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations.
Nonadditive Lossless-Join Decomposition

Example 3
Schema \(< R=ABC \), \( F = \{ A \rightarrow B \} \) and partition \( D_2 = \{ AB, BC \} \)

Consider the decompositions

\( D_1 = \{ AB, AC \} \)  and  \( D_2 = \{ AB, BC \} \)

Observe that \( D_1 \) is a 'good' decomposition (lossless) while \( D_2 \) is not.

Phantom /Spurious rows created when using decomposition \( D_2 \)

Nonadditive Lossless-Join Decomposition

Example 3 cont.
Schema \(< R=ABC \), \( F = \{ A \rightarrow B \} \) and partition \( D_1 = \{ AB, AC \} \)

Consider the decompositions

\( D_1 = \{ AB, AC \} \)  and  \( D_2 = \{ AB, BC \} \)

Observe that \( D_1 \) is a 'good' decomposition (lossless) while \( D_2 \) is not.

Notice that \( r' = \Pi_{AB}(r) \ast \Pi_{AC}(r) \)
Nonadditive Lossless-Join Decomposition

Example 3 cont.

Schemas < R₁ = ABC, F₁ = { A→BC, C→B } > and
< R₂ = BCD, F₂ = { C→B, B→C, D→B } >

Observe that r₁ ≠ r₁'.

Testing Lossless-Join (or Non-Additive) Decomposition

Definition (good only on binary partition)

If D = { R₁, R₂ } is a decomposition of R and F is a set of FDs on R, then D has a lossless-join with respect to F if

F ⇒ ( R₁ ∩ R₂ ) → ( R₁ - R₂ ) or F ⇒ ( R₁ ∩ R₂ ) → ( R₂ - R₁ )

Example 4

Consider the previous problem where R = ABC and F = { A→B }.

Let's assess the partition D₁ = {AB, AC}. Here R₁ = AB and R₂ = AC

Therefore

R₁ ∩ R₂ = A
R₁ - R₂ = B
R₂ - R₁ = C

The question F ⇒ ( R₁ ∩ R₂ ) → ( R₁ - R₂ ) is equivalent to F ⇒ A → B and we know this is true because F contains exactly this dependency.

We must conclude the decomposition D₁ is lossless with respect to F.
**Testing Lossless-Join (or Non-Additive) Decomposition**

**Example 4 (continued)**
Consider the previous problem where \( R = ABC \) and \( F = \{ A \rightarrow B \} \). Let’s now evaluate the partition \( D_2 = \{ AB, BC \} \).

Here \( R_1 = AB \) and \( R_2 = BC \)

therefore:
- \( R_1 \cap R_2 = B \)
- \( R_1 - R_2 = A \)
- \( R_2 - R_1 = C \)

The question \( F \supseteq ( R_1 \cap R_2 ) \rightarrow ( R_1 - R_2 ) \) is equivalent to \( F \supseteq B \rightarrow A \) (or \( F \supseteq B \rightarrow C \)). Both dependencies are NOT derivable from \( F \) (they are not in \( F^+ \)).

We conclude the decomposition \( D_2 \) is NOT lossless with respect to \( F \) (we will call it a *lossy* decomposition).

**Example 5**
Consider the schema \( R = ABCD \) and \( F = \{ A \rightarrow B, C \rightarrow D \} \). Let’s now evaluate the following binary partitions:

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>( R_1 \cap R_2 )</th>
<th>( R_1 - R_2 )</th>
<th>( R_2 - R_1 )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 = { AB, CD } )</td>
<td>( \emptyset )</td>
<td>AB</td>
<td>CD</td>
<td>No</td>
</tr>
<tr>
<td>( \rho_2 = { AC, BCD } )</td>
<td>C</td>
<td>A</td>
<td>BD</td>
<td>No (neither ( C \rightarrow A ) nor ( C \rightarrow BD ) in ( F^+ ))</td>
</tr>
<tr>
<td>( \rho_3 = { ABC, CD } )</td>
<td>C</td>
<td>AB</td>
<td>D</td>
<td>Yes ( C \rightarrow D ) in ( F^+ )</td>
</tr>
<tr>
<td>( \rho_2 = { BD, ACD } )</td>
<td>D</td>
<td>B</td>
<td>AC</td>
<td>No</td>
</tr>
</tbody>
</table>

We conclude only the decomposition \( \rho_3 \) is lossless with respect to \( F \).
Successive (Non-Additive) Lossless-Join Decompositions

Claim. If a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of $R$ has the nonadditive (lossless) join property with respect to a set of functional dependencies $F$ on $R$, and if a decomposition $D_i = \{Q_1, Q_2, \ldots, Q_k\}$ of $R_i$ has the nonadditive join property with respect to the projection of $F$ on $R_i$, then the decomposition $D_i = \{R_1, R_2, \ldots, R_{i-1}, Q_1, Q_2, \ldots, Q_k, R_{i+1}, \ldots, R_m\}$ of $R$ has the nonadditive join property with respect to $F$.

Algorithm 11.1. Testing for Nonadditive Join Property

Input: A universal relation $R$, a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of $R$, and a set $F$ of FDs

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (* comment *)

1. Create an initial matrix $S$ with one row $i$ for each relation $R_i$ in $D$, and one column $j$ for each attribute $A_j$ in $R$.
2. Set $S(i, j):= b_{ij}$ for all matrix entries.
   (* each $b_{ij}$ is a distinct symbol associated with indices $(i, j)$ *)
3. For each row $i$ representing relation schema $R_i$
   {for each column $j$ representing attribute $A_j$
    {if (relation $R_i$ includes attribute $A_j$) then set $S(i, j):= a_j$;};}
   (* each $a_j$ is a distinct symbol associated with index $(j)$ *)
4. Repeat the following loop until a complete loop execution results in no changes to $S$
   {for each functional dependency $X \rightarrow Y$ in $F$
    {for all rows in $S$ that have the same symbols in the columns corresponding to attributes in $X$
     {make the symbols in each column that correspond to an attribute in $Y$ be the same in all these rows as follows:
      {if any of the rows has an $a$ symbol for the column, set the other rows to that same $a$ symbol in the column.
      {if no $a$ symbol exists for the attribute in any of the rows, choose one of the $b$ symbols that appears in one of the rows for the attribute and set the other rows to that same $b$ symbol in the column
      };};};
    };}
5. If a row is made up entirely of $a$ symbols, then the decomposition has the non-additive join property; otherwise, it does not.
Algorithm 11.1. Testing for Nonadditive Join Property

Example 6
Consider the schema R=ABCD, subjected to FDs F= \{ A \rightarrow B, B \rightarrow C \}, and the Non-binary partition \( D_1 = \{ ACD, AB, BC \} \).

**Question** Is \( D_1 \) a Lossless decomposition?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACD</td>
<td>( b_{11} )</td>
<td>( b_{12} )</td>
<td>( b_{13} )</td>
</tr>
<tr>
<td>AB</td>
<td>( b_{21} )</td>
<td>( b_{22} )</td>
<td>( b_{23} )</td>
</tr>
<tr>
<td>BC</td>
<td>( b_{31} )</td>
<td>( b_{32} )</td>
<td>( b_{33} )</td>
</tr>
</tbody>
</table>

**Step 1.** Create initial table, entering \( b_j \) values in each cell.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACD</td>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
</tr>
<tr>
<td>AB</td>
<td>( a_3 )</td>
<td>( a_4 )</td>
<td>( b_{31} )</td>
</tr>
<tr>
<td>BC</td>
<td>( a_2 )</td>
<td>( a_4 )</td>
<td>( b_{33} )</td>
</tr>
</tbody>
</table>

**Step 2.** For each attribute mentioned in each partition \( R_i \) introduce \( a_j \) on column \( j \).

**Step 3.** Apply FDs to unify \( a_j, b_j \) symbols.
Using \( B \rightarrow C \) we discover \( a_2 \rightarrow \{ a_3, b_{31} \} \) therefore \( b_{31} \) becomes \( a_3 \).
Using \( A \rightarrow B \) we discover \( a_1 \rightarrow \{ a_2, b_{32} \} \) hence \( b_{32} \) can be replaced by \( a_2 \).
First row now has \( a_j \) values in each cell. Stop!

**Conclusion:** Decomposition \( D_1 = \{ ACD, AB, BC \} \) on \( <R, F> \) has a lossless-join.

---

Algorithm 11.1. Testing for Nonadditive Join Property

Example 7
Consider the schema R=ABCD, subjected to FDs F= \{ A \rightarrow B, B \rightarrow C \}, and the Non-binary partition \( D_2 = \{ AB, BC, CD \} \).

**Question** Is \( D_2 \) a Lossless decomposition?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>( b_{11} )</td>
<td>( b_{12} )</td>
<td>( b_{13} )</td>
</tr>
<tr>
<td>BC</td>
<td>( b_{21} )</td>
<td>( b_{22} )</td>
<td>( b_{23} )</td>
</tr>
<tr>
<td>CD</td>
<td>( b_{31} )</td>
<td>( b_{32} )</td>
<td>( b_{33} )</td>
</tr>
</tbody>
</table>

**Step 1.** Create initial table, entering \( b_j \) values in each cell.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
</tr>
<tr>
<td>BC</td>
<td>( a_2 )</td>
<td>( b_{32} )</td>
<td>( b_{33} )</td>
</tr>
<tr>
<td>CD</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( b_{33} )</td>
</tr>
</tbody>
</table>

**Step 2.** For each attribute mentioned in each partition \( R_i \) introduce \( a_j \) on column \( j \).

**Step 3.** Apply FDs to unify \( a_j, b_j \) symbols.
Using \( B \rightarrow C \) we discover \( a_2 \rightarrow \{ a_1, b_{32} \} \) therefore \( b_{32} \) becomes \( a_1 \).
Using \( A \rightarrow B \) is of no help in discovering new equivalent classes. There are no more FDs to apply and no row shows all \( a \)-values. STOP

**Conclusion:** Decomposition \( D_2 = \{ AB, BC, CD \} \) on \( <R, F> \) is NOT lossless.
Algorithm 11.1. Testing for Nonadditive Join Property

Example 8
Consider the schema $R=ABCD$, subjected to FDs $F= \{ A \rightarrow B, B \rightarrow C \}$, and the Non-binary partition $D_2 = \{ ABC, AD \}$.

Question Is $D_2$ a Lossless decomposition? (Binary test indicates: Yes!)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{11}$</td>
<td>$b_{12}$</td>
<td>$b_{13}$</td>
<td>$b_{14}$</td>
</tr>
<tr>
<td>$b_{21}$</td>
<td>$b_{22}$</td>
<td>$b_{23}$</td>
<td>$b_{24}$</td>
</tr>
</tbody>
</table>

Step 1. Create initial table, entering $b_{ij}$ values in each cell.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
<td>$b_{14}$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$a_5$</td>
<td>$a_6$</td>
<td>$a_7$</td>
</tr>
</tbody>
</table>

Step 2. For each attribute mentioned in each partition $R_i$ introduce $a_i$ on column $j$.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$a_2$</td>
<td>$a_3$</td>
<td>$b_{14}$</td>
</tr>
<tr>
<td>$a_4$</td>
<td>$a_5$</td>
<td>$a_6$</td>
<td>$a_7$</td>
</tr>
</tbody>
</table>

Step 3. Apply FDs to unify $a_i$, $b_{ij}$ symbols.
Using $A \rightarrow B$ we discover $a_1 \rightarrow \{ a_2, b_{22} \}$
therefore $b_{22}$ becomes $a_3$.

Conclusion: Decomposition $D_2 = \{ ABC, AD \}$ on $\langle R, F \rangle$ is lossless.

Algorithm 11.1. Testing for Nonadditive Join Property

Example 8b – Your turn…
Consider the schema $R=ABCD$, subjected to FDs $F= \{ A \rightarrow B, C \rightarrow D \}$, and the Non-binary partitions $D_4 = \{ AB, AC, AD \}$ and $D_5 = \{ AB, AC, CD \}$.

Question. Are partitions $D_4$ and $D_5$ Lossless decompositions?
Algorithm 11.1. Testing for Nonadditive Join Property

Example 9
Consider the schema R=ABCDE, subjected to FDs F= \{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}, and the Non-binary partition D_4 = \{ AD, AB, BE, CDE, AE\}.

Question Is D_4 a Lossless decomposition?

---

Consider the following table:

<table>
<thead>
<tr>
<th>Column</th>
<th>AD</th>
<th>AB</th>
<th>BE</th>
<th>CDE</th>
<th>AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_i</td>
<td>b_{i1}</td>
<td>b_{i2}</td>
<td>b_{i3}</td>
<td>b_{i4}</td>
<td>b_{i5}</td>
</tr>
<tr>
<td>a_i</td>
<td>a_{i1}</td>
<td>a_{i2}</td>
<td>a_{i3}</td>
<td>a_{i4}</td>
<td>a_{i5}</td>
</tr>
</tbody>
</table>

Step 1. Create initial table. Entering b_i in each cell.

Step 2. Entering a_i for each Column j mentioned in partition R_i.

---

Algorithm 11.1. Testing for Nonadditive Join Property

Example 9 continuation
F= \{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}

Step 3. Apply rules to unify a_i, b_i symbols
Using:
A \rightarrow C we discover a_1 \rightarrow \{b_{i1}, b_{i2}, b_{i3}\}
therefore b_{i1} \rightarrow b_{i3} \rightarrow b_{i2}

B \rightarrow C we discover a_2 \rightarrow \{b_{i4}, b_{i5}\}
therefore b_{i4} \rightarrow b_{i5}

Step 4. (continued)
C \rightarrow D leads to b_{i3} \rightarrow \{a_{i1}, a_{i2}, a_{i3}, a_{i4}\}
therefore a_{i4} = b_{i2} = b_{i4} = b_{i5}
Use a_{i4} instead.

Step 5. (continued)
DE \rightarrow C leads to a_{i1} \rightarrow \{b_{i4}, b_{i5}\}
therefore b_{i4} = a_{i1} We use a_{i1} instead.
Algorithm 11.1. Testing for Nonadditive Join Property

Example 9 continuation

\[ F = \{ A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A \} \]

<table>
<thead>
<tr>
<th>AD</th>
<th>A1</th>
<th>B12</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>B25</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>B25</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>B21</td>
<td>A3</td>
<td>A4</td>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDE</td>
<td>b21</td>
<td>b22</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
<td></td>
</tr>
<tr>
<td>AE</td>
<td>a1</td>
<td>b32</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
<td></td>
</tr>
</tbody>
</table>

**Step 6.** (continuation)

CE → A leads to \( b_{31} \rightarrow \{ b_{31}, a_1 \} \), therefore \( b_{31} = a_1 \). We use \( a_1 \) instead.

STOP

At this point one row (BE partition) consists of just \( a_1 \) values. Consequently the decomposition \( D_3 \) has a lossless-join.

Algorithm 11.1. Testing for Nonadditive (Lossless) Join Property

Example 10

(a) \( R \) = [San, Enames, Prnumber, Prevname, Plocation, Hours]
\( D = \{ R_1, R_2 \} \)

\( R_1 = EMP, LOC = \{ \text{San}, \text{Enames} \} \)
\( R_2 = \text{PRG} = \{ \text{San}, \text{Enames} \} \)

\( F = \{ \text{San}, \text{Enames}, \text{Prnumber}, \text{Prevname}, \text{Plocation}, \text{Hours} \} \)

<table>
<thead>
<tr>
<th>San</th>
<th>Enames</th>
<th>Prnumber</th>
<th>Prevname</th>
<th>Plocation</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>b11</td>
<td>b12</td>
<td>b13</td>
<td>b14</td>
<td>b15</td>
</tr>
<tr>
<td>R2</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
</tr>
</tbody>
</table>

No changes to matrix after applying functional dependencies

(b) EMP

<table>
<thead>
<tr>
<th>San</th>
<th>Enames</th>
<th>Prnumber</th>
<th>Prevname</th>
<th>Plocation</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
</tr>
<tr>
<td>R2</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td>a4</td>
<td>a5</td>
</tr>
</tbody>
</table>

WORKS_ON

<table>
<thead>
<tr>
<th>San</th>
<th>Prnumber</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>b11</td>
<td>b12</td>
</tr>
<tr>
<td>R2</td>
<td>b11</td>
<td>b12</td>
</tr>
</tbody>
</table>

Original matrix B at start of algorithm

Matrix B after applying the first two functional dependencies; last row is all “X” symbols as we should.
Prime, Non-Prime and Key Attributes

Key Attributes
A set X of attributes in the schema R is a key for R under the dependencies F, if X→R and no proper subset Y of X (X ⊆ Y) has the same property.

Prime Attributes
An attribute A in a relation schema R is prime when it is part of any candidate key of the relation.

If A is not included in any candidate key of R, A is called non-prime.

Example 11
Consider the relation Schema R=ABCD and its set F of functional dependencies F={A→BC, C→D, D→B}.

Brute-force discovery of all the keys in R under F is a simple but exponential problem.

Steps
1. Find all rules derived from F having only ONE attribute on the left-hand side
   - A⁺ = ABCD = R (therefore A is a key)
   - B⁺ = B (not a key)
   - C⁺ = CDB (not a key)
   - D⁺ = DB (not a key)

2. Find all rules derived from F having only TWO attribute on the left-hand side
   - AB⁺ = ABCD (a superkey)
   - AC⁺ = ABCD (a superkey)
   - AD⁺ = ABCD (a superkey)
   - BC⁺ = BCD (not a key)
   - BD⁺ = BD (not a key)
   - CD⁺ = CDB (not a key)
Prime, Non-Prime and Key Attributes

Example 11 continuation

3. Find all rules derived from F having only THREE attribute on the left-hand side
   - $ABC^+ = ABCD$ (a superkey)
   - $ABD^+ = ABCD$ (a superkey)
   - $BCD^+ = ABCD$ (a superkey)

4. Find all rules derived from F having only FOUR attribute on the left-hand side
   - $ABCD^+ = ABCD$ (a superkey)

Summary
   - Key: A
   - Prime: A
   - Non-Prime: BCD

Example 12 (your turn…)

Consider the relation schema $R = A B C D E G H I J K L$ and the set of dependencies $F$

   - $AB \rightarrow CDEGH$
   - $BD \rightarrow AC$
   - $DG \rightarrow L$
   - $EG \rightarrow L$
   - $IJ \rightarrow K$
   - $JK \rightarrow I$
   - $H \rightarrow IJK$

Find all the keys and identify the non-prime attributes

Candidate keys =
Non-prime =

PROBLEM
Write an program to find all the candidate keys of a database schema $(R,F)$. 
Algorithms for Relational Database Design

We will discuss the following three algorithms:

11.2.2 Non-Additive Decomposition into 3NF Schemas
Non-Additive Decomposition into BCNF Schemas

11.2.1 Dependency Preserving Decompositions into 3NF Schemas
   a) Decomposition Method
   b) Synthesis Method

11.2.3 Dependency-Preserving and Nonadditive (Lossless) Decompositions into 3NF Schemas

11.2.2 Non-Additive Decomposition into 3NF Schemas

Algorithm 11.2.A. 3NF Decomposition Method

Input: Relation Schema R and its set F of functional dependencies
Output: A decomposition D= (R₁, R₂, ..., Rₘ) of R, such that each Rᵢ
is in 3NF and the decomposition is lossless.

Method
1. Find a key K and a transitive dependency, such that
   A is a non prime attribute (i.e. A is not part of a key)
   K→Y→A
   Not (Y→K)
   A ∉ KY

2. Make R₁= (Y, A) and R₂= (R-A)

3. Repeat process on R₁ and R₂ until they become 3NF
11.2.2 Nonadditive Join Decomposition into 3NF Schemas

Example 13.A.
Consider the TEACH relation below.

<table>
<thead>
<tr>
<th>Teacher</th>
<th>Course</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Data Structures</td>
<td>Bartram</td>
</tr>
<tr>
<td>Smith</td>
<td>Data Management</td>
<td>Martin</td>
</tr>
<tr>
<td>Hall</td>
<td>Compilers</td>
<td>Hoffman</td>
</tr>
<tr>
<td>Brown</td>
<td>Data Structures</td>
<td>Horowitz</td>
</tr>
</tbody>
</table>

Observe that Key = { Text, Teacher }
The dependency Teacher → Course is such that its L.H.S is not a key.
Therefore we use this rule to partition the schema TEACH into TEACH1 and TEACH2
where:

The attributes in TEACH1 are those in
{Teacher, Course, Text} – {Course} and

TEACH2 contains {Teacher} ∪ {Course}

Finally the 3NF schema is:

TEACH1(Text, Teacher)
TEACH2(Teacher, Course)

A. 3NF Decomposition Method – Example 11
Consider the relation schema

R= Flight#, From, To, Departs, Arrives, Duration, PlaneType,
FirstClass, Coach, TotalSeats, #Meals

Non-Trivial Functional dependencies:

- PlaneType → FirstClass Coach TotalSeats
- Departs Duration → #Meals
- Arrives Duration → #Meals
- FirstClass Coach → TotalSeats
- FirstClass TotalSeats → Coach
- Coach TotalSeats → FirstClass

Candidate keys = { Flight#, (From To Departs), (From To Arrives) }
Non-Prime Attributes = { Duration, PlaneType, FirstClass, Coach,
TotalSeats, #Meals }
11.2.2 Non-Additive Decomposition into 3NF Schemas

A. 3NF Decomposition Method – Example 11 (continued)

1. We find an inappropriate transitive dependency

   \( \text{Flight#} \rightarrow \text{Departs Duration} \rightarrow \#\text{Meals} \)

We decompose into two fragments \( R_1 \) and \( R_2 \)

\[
R_1 = \text{Flight# From To Departs Arrives Duration PlaneType Firstclass Coach TotalSeats}
\]

With candidate keys:

\[
K_1 = \{ \text{Flight#}, (\text{From To Departs}), (\text{From To Arrives}) \}
\]

Observe that \( R_1 \) is not in 3NF. Note the transitive dependency

\( \text{Flight#} \rightarrow \text{PlaneType} \rightarrow \text{FirstClass Coach TotalSeats} \)

\[
R_2 = \text{Departs Duration #Meals}
\]

With designated key \( K_2 = (\text{Departs Duration}) \)

Schema \( R_2 \) is already in 3NF

2. We decompose the offending \( R_1 \) into two partitions \( R_{11} \) and \( R_{12} \)

\[
R_{11} = \text{Flight# From To Departs Arrives Duration PlaneType}
\]

With keys \( K_{11} = \{ \text{Flight#}, (\text{From To Departs}), (\text{From To Arrives}) \} \)

Which already is in 3NF.

\[
R_{12} = \text{PlaneType FirstClass Coach TotalSeats}
\]

With key \( K_{12} = \{ \text{PlaneType} \} \)

3. We decompose \( R_{12} \) which has the following transitive dependency

\( \text{PlaneType} \rightarrow \text{FirstClass Coach} \rightarrow \text{TotalSeats} \)

\[
R_{121} = \text{PlaneType FirstClass Coach in 3NF format}
\]

With key \( K_{121} = \text{PlaneType} \)

\[
R_{122} = \text{FirstClass Coach TotalSeats in 3NF format}
\]

With Key \( K_{122} = \text{FirstClass Coach} \)

Stop, database is already in 3NF format.
11.2.2 Non-Additive Decomposition into 3NF Schemas

A. 3NF Decomposition Method – Example 11 (continuation)

Consider the relation scheme $R = \{ABCDEG\}$ subject to FDs $F = \{A\rightarrow B, \ BC \rightarrow D, \ D \rightarrow EG\}$

**STEP 1**
$R = \{ABCDEG\}$ Key = AC Violation: AC → D, D → EG
Break $R$ into: $R_1 = \{DEG\}$ and $R_2 = \{ABCD\}$

**STEP 2**
$R_1 = \{DEG\}$ Key = D. No violations - already in 3NF
However $R_2$ is not in 3NF

**STEP 3**
$R_2 = \{ABCD\}$ Key = A Violation: AC→BC and BC→D
Break $R_2$ into: $R_{21} = \{BCD\}$ and $R_{22} = \{ABC\}$

**STEP 4**
$R_{21} = \{BCD\}$ Key = BC No violations - already in 3NF
However $R_{22}$ is not in 3NF

**STEP 5**
$R_{22} = \{ABC\}$ Key = AC There is a (partial dependency) violation: A → B
Break $R_{22}$ into $R_{221} = \{AB\}$ and $R_{222} = \{AC\}$ both in 3NF

**SOLUTION:** $D = \{DEG, BCD, AB, AC\}$
11.2.2 Non-Additive Decomposition into 3NF Schemas

A. Shortcomings of 3NF Decomposition
1. Time consuming - testing if an attribute is prime is an NP operation.
2. It may produce too many tables (more than we need for 3NF).

Example 13
Assume the relation scheme R=(ABCDE) obeys the following set of FDs
\( F = \{ \text{AB} \rightarrow \text{CDE}, \text{AC} \rightarrow \text{BDE}, \text{B} \rightarrow \text{C}, \text{C} \rightarrow \text{B}, \text{C} \rightarrow \text{D}, \text{B} \rightarrow \text{E} \} \)

STEP 1
\( R = \{ \text{ABCDE} \} \)  Key= \{ \text{AB, AC} \} Violation: \( \text{AB} \rightarrow \text{C}, \text{C} \rightarrow \text{D} \)
Break R into: \( R_1 = \{ \text{CD} \} \) and \( R_2 = \{ \text{ABCE} \} \)

STEP 2
\( R_1 = \{ \text{CD} \} \) Key= \text{C} No violations - already in 3NF, but \( R_2 \) is not

STEP 3
\( R_2 = \{ \text{ABCE} \} \) Key= \{ \text{AB, AC} \} Violation: \( \text{AB} \rightarrow \text{B}, \text{B} \rightarrow \text{E} \)
Break \( R_2 \) into \( R_{21} = \{ \text{BE} \} \) and \( R_{22} = \{ \text{ABC} \} \)

STEP 4
\( R_{21} = \{ \text{BE} \} \) Key= \{ \text{B} \} No violations - in 3NF
\( R_{22} = \{ \text{ABC} \} \) Key= \{ \text{AC, AB} \} No violations - already in 3NF

Solution \( D_1 = \{ \text{CD, BE, ABC} \} \)

Observation: \( D_2 = \{ \text{ABC, BDE} \} \) is 3NF and includes less fragments than \( D_1 \)

A serious problem with the 3NF-Decomposition methods is that dependencies in \( F \) may not be enforced on the decomposition.

Example 14
Consider the relation scheme \( R = \{ \text{ABCDE} \} \) subject to the following functional dependencies \( F = \{ \text{A} \rightarrow \text{BCDE}, \text{CD} \rightarrow \text{E}, \text{CE} \rightarrow \text{B} \} \)

STEP 1
\( R = \{ \text{ABCDE} \} \) Key= \{ \text{A} \} Violation: \( \text{A} \rightarrow \text{CD}, \text{CD} \rightarrow \text{E} \)
Break R into the following 3NF tables

\[ D = \left\{ \begin{array}{l}
R_1 = \{ \text{ABCD} \} \text{ where } \text{A} \rightarrow \text{BCD}, \\
R_2 = \{ \text{CDE} \} \text{ where } \text{CD} \rightarrow \text{E}
\end{array} \right\} \]

The decomposition \( D \) is lossless but the rule \( \text{CE} \rightarrow \text{B} \) is not retained in the two fragments \( R_1 \) and \( R_2 \)
Notice that \( \text{CE} \rightarrow \text{B} \not\in \{ (\prod_{R_1} (F) \cup \prod_{R_2} (F))^+ \} \)
Non-Additive Decomposition into BCNF Schemas

Algorithm 11.2.A. BCNF Decomposition Method

Input: Relation Schema R and its set F of functional dependencies
Output: A decomposition D = (R₁, R₂, ..., Rₘ) of R, such that each Rᵢ is in BCNF and the decomposition is lossless.

Method
1. Set D = {R}
2. While there is a relation schema Q in D that is not in BCNF;
   a. choose a relation schema Q in D that is not in BCNF;
   b. find a functional dependency X→Y in Q such that X is not a superkey in Q;
   c. replace Q in D by two relations schemas (Q – Y) and (X ∪ Y)

Claim: We can use decomposition to find a lossless BCNF database scheme for an initial relation scheme that is not in BCNF.

Example
Let R= { ABCDE } and let F = { A →BC, BC →A, BCD →E, E →C}
Convert R into a BCNF scheme (if needed!).

Key = {AD, BCD}. Observe BC → A is such that BC is not a superkey. Therefore R is not in BCNF and should be decomposed:
R₁ = (ABC) {A →BC, BC →A} in BCNF (superkey = {A, BC})
R₂ = (BCDE) {BCD →E, E →C} not in BCNF (E →C and E is not a superkey)
We continue decomposing R₂ into
R₂₁= (BDE) no FDs. R₂₁ is in BCNF
R₂₂= (EC) {E →C} E is a superkey R₂₂ is in BCNF.

Final solution:
Scheme D= { ABC, BDE, EC } is in BCNF (and it is lossless, prove it!)
Is the rule BCD →E preserved in D?
11.2.1 Dependency Preserving Decompositions into 3NF Schemas

Algorithm 11.2.B. 3NF Synthesis with Dependency Preservation

Input: A universal relation $R$ and a set of functional dependencies $F$ on $R$.

1. Find a minimal cover $G$ for $F$

2. For each left-hand-side $X$ of a functional dependency that appears in $G$, create a relation schema in $D$ with attributes \{ $X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}$ \}, where $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k$ are the only dependencies in $G$ with $X$ as the left-hand-side ($X$ is the key of this relation);

3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

Claim 3. Every relation schema created by Algorithm 11.2.B is in 3NF.

Observation

- The 3NF-Synthesis algorithm creates a dependency-preserving decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of a universal relation $R$ based on a set of functional dependencies $F$, such that each $R_i$ in $D$ is in 3NF.
- It guarantees only the dependency-preserving property; it does not guarantee the nonadditive join property.
11.2.1 Dependency Preserving Decompositions into 3NF Schemas

B. 3NF Synthesis with Dependency Preservation

Example 14. Consider the following universal relation:

\[ U(\text{Emp-ssn, Pno, Esal, Ephone, Dno, Pname, Plocation}) \]

\textit{Emp-ssn, Esal, Ephone} refer to the Social Security Number, salary and phone number of the employee. \textit{Pno, Pname, and Plocation} refer to the number, name, and location of the project. \textit{Dno} is department number.

The following dependencies are present:

- FD1: \textit{Emp-ssn} → \textit{Esal, Ephone, Dno}
- FD2: \textit{Pno} → \textit{Pname, Plocation}
- FD3: \textit{Emp-ssn, Pno} → \textit{Esal, Ephone, Dno, Pname, Plocation}

The Key is \{\textit{Emp-ssn, Pno}\}. A minimum Cover G is given by:

- \textit{Emp-ssn} → \textit{Esal, Ephone, Dno}
- \textit{Pno} → \textit{Pname, Plocation}

Example 14 continuation

From rule \textit{Emp-ssn} → \textit{Esal, Ephone, Dno} we create the partition

\[ R_1 (\text{Emp-ssn, Esal, Ephone, Dno}) \]

From rule \textit{Pno} → \textit{Pname, Plocation} we obtain

\[ R_2 (\text{Pno, Pname, Plocation}) \]

Comments

Observe the original key \{\textit{Emp-Ssn, Pno}\} is broken by the partition of \textit{R}.

The partition \(D=\{R_1, R_2\}\) \textit{enforces the rules} originally in \(F\); however the decomposition is \textit{not lossless} [it is \textit{not true} that \(R_1 \cap R_2 \rightarrow (R_1-R_2) \) or \((R_2-R_1)\)].
11.2.1 Dependency Preserving Decompositions into 3NF Schemas

B. 3NF Synthesis with Dependency Preservation

Example 15.
Consider r(ABCDEF) subject to the following FDs
F = { A → BCE, B → DE, ABD → CF, AC → DE }
Find a 3NF database for r under F.

Solution: Find a canonical cover for (r, F).
1. After Left Reduction:
   - Rule ABD → CF becomes A → CF
   - Rule AC → DE becomes A → DE
   - F = { A → BCE, B → DE, A → CF, A → DE }

2. After Right Reduction
   - Rule A → BCE becomes A → B
   - Rule A → DE becomes A → E
   - F = { A → B, B → DE, A → CF, A → E }

3. After Removing Redundant Rule(s):
   - Rule A → E is redundant (Trans. on A → B, B → E)
   - F = { A → B, B → DE, A → CF, A → E }

Final Partition D = ( ABCF, BDE ) (clearly in 3NF format)

11.2.3 Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

The following method—which includes a minor modification to Algorithm 11.2—produces a decomposition D of R that does the following:

- Preserves dependencies
- Has the nonadditive join property

Algorithm 11.4. Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property

Input: A universal relation R and a set of functional dependencies F on the attributes of R.
1. Find a minimal cover G for F.
2. For each left-hand-side X of a functional dependency that appears in G create a relation schema in D with attributes { X \{ A_1 \} \{ A_2 \} \ldots \{ A_k \} }, where X → A_1, X → A_2, \ldots, X → A_k are the only dependencies in G with X as left-hand-side (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R.
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S.
Algorithm 11.4.(a) Finding a Key K for R Given a set F of Functional Dependencies

**Input:**
A universal relation R and a set of functional dependencies F on attributes of R.

**Method:**
1. Set K := R.
2. For each attribute A in K
   {compute (K - A)⁺ with respect to F;
   if (K - A)⁺ contains all the attributes in R, then set K := K - {A} }

Example 17. Algorithm 11.4.
Let us review example 14.

U(Emp-ssn, Pno, Esal, Ephone, Dno, Pname, Plocation)

The Key is {Emp-ssn, Pno}. A minimum Cover G is given by:

Emp-ssn → Esal, Ephone, Dno
Pno → Pname, Plocation

Instead of only two partitions (each representing a rule in G) we will add one more fragment to include the key (Emp ssn, Pno), the resulting design contains:

R₁ (Emp-ssn, Esal, Ephone, Dno )
R₂ (Pno, Pname, Plocation )
R₃ (Emp ssn, Pno)

This design achieves both the desirable properties of dependency preservation and nonadditive join.
Example 18. Algorithm 11.4.
Consider the relation LOTS1 and its functional dependencies F:
\[ F = \{ \text{P} \rightarrow \text{LCA}, \text{LC} \rightarrow \text{AP}, \text{A} \rightarrow \text{C} \} \]

A minimum cover G is
\[ \{\text{P} \rightarrow \text{LC}, \text{LC} \rightarrow \text{AP}, \text{A} \rightarrow \text{C} \} \].

In step 2 of Algorithm 11.4 we produce design X (before removing redundant relations) as

Design X: \( R_1 (\text{P, L, C}), R_2 (\text{L, C, A, P}), \) and \( R_3 (\text{A, C}) \).

In step 4 of the algorithm, we find that \( R_3 \) and \( R_1 \) are subsumed by \( R_2 \). Hence both of those relations are redundant. Thus the 3NF schema that achieves both of the desirable properties is (after removing redundant relations)

Design X: \( R_2 (\text{L, C, A, P}) \).

11.2.3 Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

11.2. Summary of Relational Database Design Algorithms

Table 11.1
Summary of the Algorithms Discussed in Sections 11.1 and 11.2

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Input</th>
<th>Output</th>
<th>Properties/Purpose</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>A decomposition ( D ) of ( R ) and a set ( F ) of functional dependencies</td>
<td>Boolean result: yes or no for nonadditive join property</td>
<td>Testing for nonadditive join decomposition</td>
<td>See a simpler test in Section 11.1.4 for binary decompositions</td>
</tr>
<tr>
<td>11.2</td>
<td>Set of functional dependencies ( F )</td>
<td>A set of relations in 3NF</td>
<td>Dependency preservation</td>
<td>No guarantee of satisfying lossless join property</td>
</tr>
<tr>
<td>11.3</td>
<td>Set of functional dependencies ( F )</td>
<td>A set of relations in BCNF</td>
<td>Nonadditive join decomposition</td>
<td>No guarantee of dependency preservation</td>
</tr>
<tr>
<td>11.4</td>
<td>Set of functional dependencies ( F )</td>
<td>A set of relations in 3NF</td>
<td>Nonadditive join and dependency-preserving decomposition</td>
<td>May not achieve BCNF, but achieves all desirable properties and 3NF</td>
</tr>
<tr>
<td>11.4a</td>
<td>Relation schema ( R ) with a set of functional dependencies ( F )</td>
<td>Key K of ( R ) (that is a subset of ( R ))</td>
<td>To find a key K</td>
<td>The entire relation ( R ) is always a default superkey</td>
</tr>
</tbody>
</table>
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

**Introduction.**

Functional Dependencies are not capable of representing all type of associations between attributes.

In general a FD $X \rightarrow Y$ represents a many:1 relationship.

Sometimes we want to emphasize a many:many association or multiple 1:many relationships in the same table.

In this case multivalued dependencies might be more appropriated.

Example 19.

- Multiple independent 1:many relationships in the same table.
- Observe that ProjectName and DependentName are independent.
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

**Definition**

A multivalued dependency $X \rightarrow Y$ specified on relation schema $R$, where $X$ and $Y$ are both subsets of $R$, specifies the following constraint on any relation state $r$ of $R$:

If two tuples $t_1$ and $t_2$ exist in $r$ such that $t_1[X] = t_2[X]$, then two tuples $t_3$ and $t_4$ should also exist in $r$ with the following properties:

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$  

[we use $Z$ to denote $(R - (X \cup Y))$.]

Whenever $X \rightarrow Y$ holds, we say that $X$ multi-determines $Y$.

Because of the symmetry in the definition, whenever $X \rightarrow Y$ holds in $R$, so does $X \rightarrow Z$. Hence, $X \rightarrow Y$ implies $X \rightarrow Z$, and therefore it is sometimes written as $X \rightarrow Y | Z$.

**Example 20.a**

Consider the tuples $t_1$, $t_2$, $t_3$, and $t_4$ shown below. Observe that $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

Therefore $X \rightarrow Y$ and $X \rightarrow Z$, i.e. $Ename \rightarrow Pname$, $Ename \rightarrow DependatName$

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>Smith</td>
<td>Proj-X</td>
<td>Johnny</td>
</tr>
<tr>
<td>$t_2$</td>
<td>Smith</td>
<td>Proj-Y</td>
<td>Anna</td>
</tr>
<tr>
<td>$t_3$</td>
<td>Smith</td>
<td>Proj-X</td>
<td>Anna</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Smith</td>
<td>Proj-Y</td>
<td>Johnny</td>
</tr>
</tbody>
</table>

$t_3[X] = t_4[X] = t_1[X] = t_2[X] = "Smith"

t_3[Y] = t_1[Y] = ‘Proj-X’

t_3[Y] = t_2[Y] = ‘Proj-Y’

t_3[Z] = t_2[Z] = ‘Anna’

t_3[Z] = t_1[Z] = ‘Johnny’
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

**Observation**
A multivalued dependency (or MVD) $X \rightarrow Y$ specifies that given a particular value of $X$, the set of values of $Y$ determined by this value of $X$ is completely determined by $X$ alone and does not depend on the values of the remaining attributes $Z$ of $R$.

**Definition**
An MVD $X \rightarrow Y$ in $R$ is called a trivial MVD if
(a) $Y$ is a subset of $X$, or
(b) $X \cup Y = R$.

**Observation**
If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples (which is unwanted!).

---

**Example 20.b**
The Employee-Family-Project table shown before in Example 20.a holds the MVDs:

<table>
<thead>
<tr>
<th>Ename</th>
<th>Pname</th>
<th>DependentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Proj-X</td>
<td>Johnny</td>
</tr>
<tr>
<td>Smith</td>
<td>Proj-Y</td>
<td>Anna</td>
</tr>
<tr>
<td>Smith</td>
<td>Proj-X</td>
<td>Anna</td>
</tr>
<tr>
<td>Smith</td>
<td>Proj-Y</td>
<td>Johnny</td>
</tr>
</tbody>
</table>

Equivalent representations of the original relation:

<table>
<thead>
<tr>
<th>Ename</th>
<th>Pname</th>
<th>DependentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Proj-X</td>
<td></td>
</tr>
<tr>
<td>Smith</td>
<td>Proj-Y</td>
<td></td>
</tr>
</tbody>
</table>

Equivalently:

<table>
<thead>
<tr>
<th>Ename</th>
<th>Pname</th>
<th>DependentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Proj-X, Proj-Y</td>
<td>Johnny, Anna</td>
</tr>
</tbody>
</table>

Copyright © 2007 Pearson Education, Inc. Publishing as Pearson Addison-Wesley
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Inference rules for Multivalued Dependencies
The following inference rules IR1 through IR8 form a sound and complete set for inferring functional and multivalued dependencies from a given set of dependencies.

Assume that all attributes are included in a universal relation schema R = \{A_1, A_2, \ldots, A_m\} and that X, Y, Z, and W are subsets of R.

IR1 (reflexive rule for FDs): If X \supseteq Y, then X \rightarrow Y.
IR2 (augmentation rule for FDs): \{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ.
IR3 (transitive rule for FDs): \{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z.
IR4 (complementation rule for MVDs): \{X \downarrow Y\} \Rightarrow \{X \downarrow (R - (X \cup Y))\}.
IR5 (augmentation rule for MVDs): If X \rightarrow Y and W \supseteq Z, then WX \rightarrow YZ.
IR6 (transitive rule for MVDs): \{X \downarrow Y, Y \rightarrow Z\} \Rightarrow X \downarrow (Z \cdot Y).
IR7 (replication rule for FD to MVD): \{X \rightarrow Y\} \Rightarrow X \downarrow Y.
IR8 (coalescence rule for FDs and MVDs): If X \downarrow Y and there exists W with the properties that (a) W \cap Y is empty, (b) W \rightarrow Z, and (c) Y \subseteq Z, then X \rightarrow Z.
IR9 (Additivity) If X \downarrow Y and X \downarrow Z then X \downarrow YZ.
IR10(Projectivity) If X \downarrow Y and X \downarrow Z then X \downarrow Y \cap Z, X \downarrow Y - Z, and X \downarrow Z - Y.

Note: IR9 and IR10 are derived from IR1 - IR8. See D. Maier Chp 7 pp 129.

11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

How to know if a given m.v.d. holds in a relation r?

Is A \rightarrow BC valid in r?

Method
1. Decompose r into r1(ABC) and r2(ADE)
2. Join r1 and r2 (call it r12)
3. Compare r with r12 if both are equal the dependency holds.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d3</td>
<td>e3</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d4</td>
<td>e4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d3</td>
<td>e4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d4</td>
<td>e3</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>d5</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>d5</td>
<td>e6</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>d6</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d5</td>
<td>e6</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d6</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d6</td>
<td>e6</td>
</tr>
</tbody>
</table>

1. In this example A \rightarrow BC is valid.
2. Your turn try BC \rightarrow CD
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

How to know if a given m.v.d. holds in a relation r?

Testing validity of BC → CD

**Method**
1. Decompose r into r1(BCD) and r2(ABDE)
2. Join r1 and r2 (call it r12)
3. Compare r with r12 if both are equal the dependency holds.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td>e1</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d3</td>
<td>e3</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d4</td>
<td>e4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d3</td>
<td>e4</td>
</tr>
<tr>
<td>a2</td>
<td>b3</td>
<td>c3</td>
<td>d4</td>
<td>e3</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>d5</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b4</td>
<td>c4</td>
<td>d5</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d5</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d5</td>
<td>e6</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d6</td>
<td>e5</td>
</tr>
<tr>
<td>a3</td>
<td>b5</td>
<td>c5</td>
<td>d6</td>
<td>e6</td>
</tr>
</tbody>
</table>

In this example BC → BD is valid.

11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Inference rules for Multivalued Dependencies

**MVD Transitivity**

Rule IR6 (transitive rule for MVDs): \{X → Y, Y → Z\} ⇒ X → (Z → Y)

**Example**

A → BC
BC → CD
A → D (i.e. CD → BC)

Observe that
a1 → \{b1c1\} → \{d1\}
a2 → \{b3c3\} → \{d3,d4\}
a3 → \{b4c4, b5c5\} → \{d5, d6\}
Inference rules for Multivalued Dependencies

**MVD Coalescense**

**IR8** (coalescence rule for FDs and MVDs):
If \( X \rightarrow Y \) and there exists \( W \) with the properties that
(a) \( W \cap Y \) is empty,
(b) \( W \rightarrow Z \), and
(c) \( Y \supseteq Z \),
then \( X \rightarrow Z \).

**Example**

- \( S \rightarrow \text{Day} \ \text{Hour} \ \text{Channel} \)
- \( S \rightarrow \text{Sponsor} \)
- \( S \rightarrow \text{Actor} \)
- \( \text{TVstation} \rightarrow \text{Channel} \)
- \( S \rightarrow \text{Channel} \)

<table>
<thead>
<tr>
<th>Show</th>
<th>Day</th>
<th>Hour</th>
<th>Channel</th>
<th>TVstation</th>
<th>Sponsor</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Mo</td>
<td>9p</td>
<td>1</td>
<td>ABC</td>
<td>Nike</td>
<td>Angeline</td>
</tr>
<tr>
<td>s1</td>
<td>Mo</td>
<td>9p</td>
<td>1</td>
<td>ABC</td>
<td>Nike</td>
<td>Sandra</td>
</tr>
<tr>
<td>s1</td>
<td>Mo</td>
<td>9p</td>
<td>1</td>
<td>ABC</td>
<td>Pepsi</td>
<td>Angeline</td>
</tr>
<tr>
<td>s1</td>
<td>Mo</td>
<td>9p</td>
<td>1</td>
<td>ABC</td>
<td>Pepsi</td>
<td>Sandra</td>
</tr>
<tr>
<td>s2</td>
<td>We</td>
<td>8p</td>
<td>3</td>
<td>CBS</td>
<td>ATT</td>
<td>Fox</td>
</tr>
<tr>
<td>s2</td>
<td>We</td>
<td>8p</td>
<td>3</td>
<td>CBS</td>
<td>IBM</td>
<td>Scully</td>
</tr>
<tr>
<td>s3</td>
<td>Tu</td>
<td>7p</td>
<td>1</td>
<td>ABC</td>
<td>Toyota</td>
<td>Columbus</td>
</tr>
<tr>
<td>s3</td>
<td>Fr</td>
<td>7p</td>
<td>1</td>
<td>ABC</td>
<td>Toyota</td>
<td>Columbus</td>
</tr>
</tbody>
</table>

**Example 21A. Inference rules for Multivalued Dependencies**

Consider the dependencies \( F = \{ A \rightarrow BC, DE \rightarrow C \} \) over the schema \( R = \{ ABCDE \} \).

**Question**

Does \( AD \rightarrow BE \) ?

**Proof.** Using the Inference Axioms IR1 to IR8 we have
1. \( A \rightarrow BC \) given
2. \( A \rightarrow DE \) by complementation respect to \( R \)
3. \( DE \rightarrow C \) given
4. \( A \rightarrow C \) transitivity on 2 and 3
5. \( AD \rightarrow C \) augmentation of 4 by \( D \)
6. \( AD \rightarrow BE \) complementation of 5 respect to \( R \)
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Example 21B. Inference rules for Multivalued Dependencies

Consider the dependencies $F = \{ A \rightarrow EI, C \rightarrow AB \}$ on the schema $R = \{ ABCDEI \}$.

Question
Does $AC \rightarrow BI$?

Proof.
Using the Inference Axioms IR1 to IR8 we have
1. $C \rightarrow AB$ given
2. $C \rightarrow DEI$ complementation of 1 respect to $R$
3. $CEI \rightarrow DEI$ augmentation of 2 by EI
4. $A \rightarrow EI$ given
5. $AC \rightarrow CEI$ augmentation of 4 by C
6. $AC \rightarrow D$ transitivity on 5 and 3
7. $AC \rightarrow BEI$ complementation of 5 respect to $R$

Example 21C. Inference rules for Multivalued Dependencies

Consider the dependencies $F = \{ A \rightarrow EI, C \rightarrow AB \}$ on the schema $R = \{ ABCDEI \}$.

Question
Does $AC \rightarrow B$?

Proof.
Using the Inference Axioms IR1 to IR8 we have
1. $C \rightarrow AB$ given
2. $C \rightarrow DEI$ complementation of 1 respect to $R$
3. $AC \rightarrow ADEI$ augmentation of 3 by A
4. $AC \rightarrow B$ complementation of 3 respect to $R$
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

**Minimal Disjoint Set Basis**

**Definition:**
Given a collection of sets $S = \{S_1, S_2, \ldots, S_p\}$, where the universe $U = S_1 \cup S_2 \cup \ldots \cup S_p$, the *minimal disjoint set basis* of $S$ (mdsb($S$)) is the partition $T= \{T_1, T_2, \ldots, T_q\}$ of $U$ such that:

1. Every $S_i$ is a union of some of the $T_j$'s
2. No partition of $U$ with fewer cells has the first property

**Example**
Assume $S = \{ABC, BCD, AD\}$ then $\text{mdsb}(S) = \{A, BC, D\}$

Observe that:

- $ABC = A + BC$
- $BCD = BC + D$
- $AD = A + D$

---

11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

**Algorithm. Computing Dependency Basis**

**Input:**
A set of multivalued dependencies $M$ over a set of attributes $R$, and a set $X \subseteq U$

**Output:** The dependency basis for $X$ with respect to $M$.

**Method:**
1. Let $T$ be the set of sets $Z \subseteq X$ such that for some $W \rightarrow Y$ in $M$, we have $W \subseteq X$, and $Z$ is either $(Y - X)$ or $(R - X - Y)$

2. Until $T$ consists of a disjoint collection of sets, find a pair of sets $Z_1$ and $Z_2$ in $T$ that are not disjoint and replace them by sets $(Z_1 - Z_2)$, $(Z_2 - Z_1)$ and $(Z_1 \cap Z_2)$. Let $S$ be the final collection of sets.

3. Until no more changes can be made to $S$, look for dependencies $V \rightarrow W$ in $M$ and a set $Y$ in $S$ such that $Y$ intersects $W$ but not $V$. Replace $Y$ by the sets $(Y \cap W)$ and $(Y - W)$ in $S$.

4. The final collection of sets in $S$ is the dependency basis for $X$. 
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Algorithm. Computing Dependency Basis

Intuition
Assume we are computing DEP(AC) for schema R under M(Functional and Multivalued). STEP1 begins with the application of reflexivity AC → AC and complementation AC → R-AC. Therefore the initial set T_{AC} includes {A, R-A}.

Exploring rules in M of the form AC → X1, A → X2 and C → X3 brings to T_{AC} other attributes directly implied by AC. Observe that by augmentation if A → X2 then AC → X2, similarly if C → X3 by augmentation AC → X3. Therefore fragments X1, X2, X3 (and their complements) must also be added to T_{AC}.

If no more elements can be included to T_{AB} a fine fragmentation of the existing T_{AB} components should follow to produce its msdb(T_{AC}) (called S_{AC} in the algorithm).

In the final step, we try one more refinement of the elements Y in S_{AC} = msdb(T_{AC}). The Coalescence axiom (IR8) states that if there is a dependency V → W in M and a set Y in S_{AC} such that Y intersects W but not V then AC → W. Therefore the set Y could be replaced by the finer sets (Y ∩ W) and (Y - W) in S_{AC}.

The final collection of sets in S_{AC} is the dependency basis DEP(AC).

Claim
To test whether a MVD X → Y holds in F, it suffices to determine DEP(X) and see whether (Y - X) is the union of some sets in DEP(X).

Example 22A
Let F= {A → BC, DE → C} be a set of MVDs over ABCDE.

Compute DEP(A)
To compute it we begin with T_{A} = {A, BCDE} (A → A, A → BCDE).

Step 1 allows us to use A → BC (given) and introduce BC in T_{A}.

Observe that A → DE (by use of complementation), therefore DE is also added to T_{A} = {A, BCDE, BC, DE}.

Refining this set produces S_{A} = {A, BC, DE}.

Using Step3 of the method we find rule DE → C and element BC intersects the RHS; however BC does not share common attributes with DE (the LHS). Consequently we replace BC with (BC ∩ C) and (BC - C).

Hence DEP(A)={A, B, C, DE}
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Example 22A continuation
Let \( F = \{ A \rightarrow BC, DE \rightarrow C \} \) be a set of MVDs over ABCDE.

Hence \( \text{DEP}(A) = \{ A, B, C, DE \} \)

**Significance**
Attribute \( A \) multi-determines any union of elements in \( \text{DEP}(A) \), for example

\[
\begin{align*}
A \rightarrow A & \quad A \rightarrow BC \\
A \rightarrow B & \quad A \rightarrow DE \\
A \rightarrow C & \quad A \rightarrow CDE \\
A \rightarrow BCDE & \\
A \rightarrow BDE & 
\end{align*}
\]

Similarly we may say it is NOT true that \( A \rightarrow D \) [notice D is not an element of \( \text{DEP}(A) \)]

In a similar way \( \text{DEP}(AD) = \{ A, B, C, D, E \} \)

\[
\begin{align*}
\text{DEP}(BC) = \{ B, C, ADE \}
\end{align*}
\]

**Why?**
Let us compute the dependency basis of AD. We begin with a set \( T_{AD} \) consisting of \( \{ AD, BCE \} \). Then we look for MVDs whose LHS is in AD, such as

- \( A \rightarrow B \), \( A \rightarrow C \), \( A \rightarrow A \) (see first part of this problem)

Therefore \( T_{AD} = \{ AD, BCE, A, B, C, BCDE, CDE, BDE, DE, BC \} \)

Refining of this set produces \( S_{AD} = \{ A, B, C, D, E \} = \text{DEP}(AD) \)

Similarly, computing \( \text{DEP}( BC ) \) begins with \( T_{BC} = \{ BC, ADE \} = S_{BC} \)

Nothing else can be added to \( T_{BC} \). Now consider \( Y = BC \) and the rule \( DE \rightarrow C \).

The RHS of this rule and \( Y \) intersect but its LHS has nothing in common with BC. Step3 of the DEP _BASIS algorithm tells us to replace BC with (BC\(\cap\)C) and (BC - C). That is, C and B. Therefore \( \text{DEP}( BC ) = \{ B, C, ADE \} \).
Example 22B
Let F = { A→EI, C→AB } be a set of MVDs over ABCDEI.
Find DEP(AC).  See problems 21B and 21C

DEP(AC) = { A, B, C, D, EI }

Why?
Let us compute the dependency basis of AC.
We begin with a set T_{AC} consisting of {AC, BDEI}. Then we look for MVDs
whose LHS is in AD, such as A→EI, A→BCD, C→AB, C→DEI.
Therefore T_{AC} = { AC, BDEI, EI, BCD, AB, DEI }
Refining of this set produces S_{AC} = {A, B, C, D, EI }.

In problem 21B we were asked to show that AC→BEI. Observe that B and EI
are elements in DEP(AC) therefore AC→BEI, similarly AC→B (see problem
21C).

11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Example 22C. (from J. Ullman – Database Systems)
Consider the following relation
R= (Course, Teacher, Hour, Room, Student, Grade)

By observation we find the following MVDs

- C→HR
- C→SG
- C→T
- HT→R
- HR→C
- CS→G
- HS→R

Notice that DEP(C) = {T, SG, HR}.

Therefore  C→T, C→HR, C→THR, C→TSG, C→HRSG, C→THRSG,
Also observe that the MVD C→R is invalid (Room and Hour are depend on each other)
Other example is DEP(HR) = {HR, TSG} from here we conclude it is not true that HR→SG
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Definition. Nonadditive Join Decomposition into 4NF Relations

Whenever we decompose a relation schema \( R \) into \( R_1 = (X \cup Y) \) and \( R_2 = (R - Y) \) based on an MVD \( X \rightarrow Y \) that holds in \( R \), the decomposition has the nonadditive join property.

Property NJB. The relation schemas \( R_1 \) and \( R_2 \) form a non-additive join decomposition of \( R \) with respect to a set \( F \) of functional and multivalued dependencies if and only if

\[
R_1 \cap R_2 \supseteq (R_1 - R_2) \quad \text{or} \\
R_1 \cap R_2 \supseteq (R_2 - R_1)
\]

Example:

The relation Book(Book#, Author, Price) with multivalued dependencies \( F = \{ \text{Book#} \rightarrow \text{Author}, \text{Book#} \rightarrow \text{Price} \} \) has a lossless decomposition in

\[ R_1 = (\text{Book#}, \text{Author}) \]
\[ R_2 = (\text{Book#}, \text{Price}) \]

11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Definition Fourth Normal Form (4NF)

Fourth normal form (4NF) schema is violated when a relation has undesirable non-trivial multivalued dependencies (it should be decomposed)

Definition.

A relation schema \( R \) is in 4NF with respect to a set of dependencies \( F \) (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency \( X \rightarrow Y \) in \( F^* \), \( X \) is a superkey for \( R \).

Observation.

In the previous definition the determination of superkey values is solely based on functional dependencies in \( F \).
11.3 Multivalued Dependencies and Fourth Normal Form (4NF)

Algorithm 11.5. Relational Decomposition into 4NF Relations with Nonadditive Join Property

Input:
A universal relation R and a set of functional and multivalued dependencies F.

Method
1. Set D:= { R }
2. While there is a relation schema Q in D that is not in 4NF, do
   { choose a relation schema Q in D that is not in 4NF;
      find a nontrivial MVD X \rightarrow^* Y in Q that violates 4NF;
      replace Q in D by two relation schemas (Q - Y) and (X \cup Y);
   }

Example 23.
Consider the previous example where R= \{ CTHRSG \} and the dependencies are F = \{ C \rightarrow HR, C \rightarrow SG, C \rightarrow T \}.

1. Observe that MVDs in F are not trivial and C is NOT a superkey in R.
2. Therefore we could use the rule C \rightarrow HR to decompose the table into fragments: R_1=CHR, R_2=CSGT. The mvd C \rightarrow HR in R_1 becomes trivial.
3. Again, C is not a superkey in R2 and the MVD C \rightarrow SG is not trivial
4. Therefore we could partition R2 into R_{21}=CSG, R_{22}=CT where R_{21} holds a trivial MVD, and R_{22} has only one rule C \rightarrow T in which C is a superkey.
5. The final 4NF Lossless database schema is D = \{ CHR, CSG, CT \}
### 11.4 Join Dependencies and Fifth Normal Form (5NF)

**Definition. Join Dependencies**

A join dependency (JD), denoted by JD(R₁, R₂, . . . , Rₙ), on relation schema R, specifies a constraint on the states r of R. The constraint requests that every legal state r of R should have a non-additive join decomposition into R₁, R₂, . . . , Rₙ.

That is, for every such r we have

\[
* \left( \pi_{R_1}(r), \pi_{R_2}(r), \ldots, \pi_{R_n}(r) \right) = r
\]

Notice that an MVD is a special case of a JD where n = 2. That is, a JD denoted as JD(R₁, R₂) implies an MVD (R₁ ∩ R₂) → (R₁ – R₂) (or (R₂ – R₁)).

**Definition**

A join dependency JD(R₁, R₂, . . . , Rₙ), specified on relation schema R, is a trivial JD if one of the relation schemas Rᵢ in JD(R₁, R₂, . . . , Rₙ) is equal to R.

### 11.4 Join Dependencies and Fifth Normal Form (5NF)

**Definition. Fifth Normal Form (5NF) or Projection-Join NF (PJNF)**

A relation schema R is in fifth normal form (5NF) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(R₁, R₂, . . . , Rₙ), in F⁺ every Rᵢ is a superkey of R.

**Observation**

Fifth normal form deals with cases where information can be reconstructed from smaller pieces of information that can be maintained with less redundancy. The big difficulty of reaching 5NF is that all possible decompositions of R must support the lossless join property.

*Note: Discovering JDs in practical databases with hundreds of attributes is next to impossible. Therefore, the current practice of database design pays little attention to them.*
11.4 Join Dependencies and Fifth Normal Form (5NF)

Example 24. Fifth Normal Form (5NF) or Projection-Join NF (PJNF)

5NF demands that all possible projections of \( r(R) \) on a given decomposition \( D_R \), losslessly regenerate the original table. Consider a decomposition of the example schema \( R= \) (Agent, Company, Product) into the following two sub-schemas: \( R_1(\text{Agent, Company}) \) and \( R_2(\text{Company, Product}) \).

Using the given data is easy to proof that \( \pi_{R_1}(r) \ast \pi_{R_2}(r) \neq r \) (phantom tuples such as \( <\text{Brown, GM, truck}> \) will be produced). Hence \( R \) is not in 5NF.

However \( r(R) \) has a 5NF representation using the three trivial schemas: \( R_1, R_2, R_3 \) given below.

<table>
<thead>
<tr>
<th>AGENT</th>
<th>COMPANY</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Ford</td>
<td>car</td>
</tr>
<tr>
<td>Smith</td>
<td>Ford</td>
<td>truck</td>
</tr>
<tr>
<td>Smith</td>
<td>GM</td>
<td>car</td>
</tr>
<tr>
<td>Smith</td>
<td>GM</td>
<td>truck</td>
</tr>
<tr>
<td>Jones</td>
<td>Ford</td>
<td>car</td>
</tr>
<tr>
<td>Jones</td>
<td>Ford</td>
<td>truck</td>
</tr>
<tr>
<td>Brown</td>
<td>Ford</td>
<td>car</td>
</tr>
<tr>
<td>Brown</td>
<td>GM</td>
<td>car</td>
</tr>
<tr>
<td>Brown</td>
<td>Toyota</td>
<td>car</td>
</tr>
<tr>
<td>Brown</td>
<td>Toyota</td>
<td>bus</td>
</tr>
</tbody>
</table>

11.5 Inclusion Dependencies

Observation

Inclusion dependencies were defined in order to formalize two types of inter-relational constraints that cannot be represented with FD/MVD/JDs:

- **Foreign key** (or referential integrity) relates attributes across relations.
- **Inheritance** represent a class/subclass relationship.

Definition.

An inclusion dependency \( R.X < S.Y \) between two sets of attributes-\( X \) of relation schema \( R \), and \( Y \) of relation schema \( S \)-specifies the constraint that, at any specific time when \( r \) is a relation state of \( R \) and \( s \) a relation state of \( S \), we must have

\[
\pi_X(s(S)) \subseteq \pi_Y(r(R))
\]
11.5 Inclusion Dependencies

**Example 25**
we can specify the following (referential Integrity type) *inclusion dependencies* on the relational schema in Figure 10.1

![Diagram of relations](image)

**Example 26**
we can specify the following (inheritance type) *inclusion dependencies* on the relational schema in Figure 7.1

![Diagram of relations](image)
11. 5 Inclusion Dependencies

**Definition**
As with other types of dependencies, there are inclusion dependency inference rules (IDIRs). The following are three examples:

**IDIR1** (reflexivity): \( R.X < R.X \).

**IDIR2** (attribute correspondence): If \( R.X < S.Y \), where \( X = \{A_1, A_2, \ldots, A_n\} \) and \( Y = \{B_1, B_2, \ldots, B_n\} \) and \( A_i \) Corresponds to \( B_i \) then \( R.A_i < S.B_i \) for \( 1 \leq i \leq n \).

**IDIR3** (transitivity): If \( R.X < S.Y \) and \( S.Y < T.Z \), then \( R.X < T.Z \).

The preceding inference rules were shown to be sound and complete for inclusion dependencies. So far, no normal forms have been developed based on inclusion dependencies.

11. 6 Template Dependencies

**Rationale**
Template dependencies provide a technique for representing constraints in relations that typically have no easy and formal definitions.

The idea behind template dependencies is to specify a template-or example-that defines each constraint or dependency.

There are two types of templates: *tuple-generating* templates and *constraint generating* templates.

- A template consists of a number of **hypothesis tuples** that are meant to show an example of the tuples that may appear in one or more relations.

- The other part of the template is the **template-conclusion**, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there.

- For constraint-generating templates, the template conclusion is a condition that must hold on the hypothesis tuples.
11.6 Template Dependencies

**Figure 11.6**
Templates for some common type of dependencies.
(a) Template for functional dependency $X \rightarrow Y$.
(b) Template for the multivalued dependency $X \rightarrow Y$.
(c) Template for the inclusion dependency $RX < S.Y$.

(a) $R = \{A, B, C, D\}$

Hypothesis

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$c_1$</th>
<th>$d_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

Conclusion

$c_1 = c_2$ and $d_1 = d_2$

(b) $R = \{A, B, C, D\}$

Hypothesis

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$c_1$</th>
<th>$d_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

Conclusion

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$c_2$</th>
<th>$d_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

(c) $R = \{A, B, C, D\}$

Hypothesis

<table>
<thead>
<tr>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$c_1$</th>
<th>$d_1$</th>
</tr>
</thead>
</table>

Conclusion

$c_1 \leq d_1$

---

11.6 Template Dependencies

**Figure 11.7**
Templates for the constraint that an employee’s salary must be less than the supervisor’s salary.

$EMPLOYEE = \{\text{Name, Ssn, . . . , Salary, Supervisor_ssn}\}$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e$</td>
<td>$d$</td>
<td>$f$</td>
<td>$g$</td>
</tr>
</tbody>
</table>

Hypothesis

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
</table>

Conclusion

$c < f$
11.6 Template Dependencies

**Functional Dependencies Based on Arithmetic Functions and Procedures**

Sometimes the attributes in a relation may be related via some arithmetic function or a more complicated functional relationship. As long as a unique value of \( Y \) is associated with every \( X \), we can still consider that the FD \( X \rightarrow Y \) exists. For example, in the relation

\[ \text{ORDER-LINE (Order#, Item#, Quantity, Unit-price, Extended-price, Discounted-price)} \]

In this relation,

\[ (\text{Quantity, Unit-price} ) \rightarrow \text{Extended-price} \]

by the formula

\[ \text{Extended-price} = \text{Unit-price} \times \text{Quantity}. \]

Although the above kinds of FDs are technically present in most relations, they are not given particular attention during normalization.

**References**


**Questions ?**