CSCI1270
Introduction to Database Systems

Normalization
Another Use for FD’s: Schema Design

Schema Design: Approach #1

1. Construct E/R diagram
2. Translate into tables
   Subjective: How do we know if any good?

Schema Design: Approach #2

1. Start with universal relation
2. Determine FD’s
3. “Decompose” UR using FD’s as guide

Schema Design: Approach #3

1. Construct E/R diagram to come up with 1st cut design
2. Use FD’s to verify or refine
Decomposition

1. Decomposing the Schema

\[ R = (\text{bname, bcity, assets, cname, lno, amt}) \]

\[ R_1 = (\text{bname, bcity, assets, cname}) \]

\[ R_2 = (\text{cname, lno, amt}) \]

Notation:

\[ R = R_1 \cup R_2 \]
2. Decomposing the Instance

\[ R = \]

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

\[ R_1 = \]

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

\[ R_2 = \]

<table>
<thead>
<tr>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Johnson</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

**BTW: Not a Good Decomposition**
Goals of Decomposition

1. **Lossless Joins**
   
   Want to be able to reconstruct big relation by joining smaller ones (Natural join) (i.e.: $R_1 \bowtie R_2 = R$)

2. **Dependency Preservation**

   Want to minimize the cost of global integrity constraints based on FD’s (i.e.: Avoid big joins in assertions)

3. **Redundancy Avoidance**

   Avoid unnecessary data dupl. (motivation for decomposition)

**Summary:**

- LJ: Information loss
- DP: Efficiency (time)
- RA: Efficiency (space), update anomalies
Another Use for FD’s: Schema Design

Example:

R =

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

R: “Universal Relation”

Tuple meaning: Jones has a loan (L-17) for $1000 taken out of the Dntn branch in Bkln which has assets of $9M

Design: pro: Fast queries (No need for joins!)
con: Redundancy, update anomalies, deletion anomalies
### Decomposition Goal #1: Lossless Joins

#### A Bad Decomposition

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Johnson</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

### Correct Decomposition

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
<td>L-23</td>
<td>3000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

CSCI11270: Introduction to Database Systems
Decomposition Goal #1: Lossless Joins

A Bad Decomposition

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
<th>lno</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
<td>L-23</td>
<td>3000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-93</td>
<td>500</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
<td>L-17</td>
<td>1000</td>
</tr>
</tbody>
</table>

Problem: × adds meaningless tuples

“Lossy join”: By adding noise, have lost meaningful information as a result of decomposition
Lossless Joins

Is the Following Decomposition Lossless or Lossy?

\( R_1 = \)

<table>
<thead>
<tr>
<th>bname</th>
<th>bcity</th>
<th>assets</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Johnson</td>
</tr>
<tr>
<td>Mianus</td>
<td>Hnck</td>
<td>1.7M</td>
<td>Jones</td>
</tr>
<tr>
<td>Dntn</td>
<td>Bkln</td>
<td>9M</td>
<td>Hayes</td>
</tr>
</tbody>
</table>

\( R_2 = \)

<table>
<thead>
<tr>
<th>bname</th>
<th>Ino</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Dntn</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Mianus</td>
<td>L-93</td>
<td>500</td>
</tr>
</tbody>
</table>

A: Lossy.

\( R_1 \bowtie R_2 \) includes:

(R\(_1 \bowtie R_2\) has 7
tuples, whereas \(R\) has 4)
Is the Following Decomposition Lossless or Lossy?

R₁ =

<table>
<thead>
<tr>
<th>bname</th>
<th>assets</th>
<th>cname</th>
<th>lno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dntn</td>
<td>9M</td>
<td>Jones</td>
<td>L-17</td>
</tr>
<tr>
<td>Dntn</td>
<td>9M</td>
<td>Johnson</td>
<td>L-23</td>
</tr>
<tr>
<td>Mianus</td>
<td>1.7M</td>
<td>Jones</td>
<td>L-93</td>
</tr>
<tr>
<td>Dntn</td>
<td>9M</td>
<td>Hayes</td>
<td>L-17</td>
</tr>
</tbody>
</table>

R₂ =

<table>
<thead>
<tr>
<th>lno</th>
<th>bcity</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-17</td>
<td>Bkln</td>
<td>1000</td>
</tr>
<tr>
<td>L-23</td>
<td>Bkln</td>
<td>2000</td>
</tr>
<tr>
<td>L-93</td>
<td>Hnck</td>
<td>500</td>
</tr>
</tbody>
</table>

A: Lossless. R₁⋈ R₂ has 4 tuples
Lossless Joins

Lossless or Lossy?

\[ R_1 = \begin{array}{ccc} 
\text{bname} & \text{bcity} & \text{assets} \\
\text{Dntn} & \text{Bkln} & \text{9M} \\
\text{Mianus} & \text{Bkln} & \text{1.7M} \\
\end{array} \]

\[ R_2 = \begin{array}{cccc} 
\text{bname} & \text{lno} & \text{amt} & \text{cname} \\
\text{Dntn} & \text{L-17} & \text{1000} & \text{Jones} \\
\text{Dntn} & \text{L-23} & \text{2000} & \text{Johnson} \\
\text{Mianus} & \text{L-93} & \text{500} & \text{Jones} \\
\text{Dntn} & \text{L-17} & \text{1000} & \text{Hayes} \\
\end{array} \]

A: Lossless. \( R_1 \bowtie R_2 \) has 4 tuples

Q: When is decomposition lossless?
Ensuring Lossless Joins

A Decomposition of $\mathbf{R}$, $\mathbf{R} = \mathbf{R}_1 \cup \mathbf{R}_2$ is **lossless** iff

\[
\begin{align*}
\mathbf{R}_1 \cap \mathbf{R}_2 & \rightarrow \mathbf{R}_1 \quad \text{or} \\
\mathbf{R}_1 \cap \mathbf{R}_2 & \rightarrow \mathbf{R}_2
\end{align*}
\]

(i.e.: Intersecting atts must form a super key for one of the resulting smaller relations)

**Intuition:** Original relation $\mathbf{R}$ has $n$ tuples

- A key $\Rightarrow |\mathbf{R}_1| = n$
- A not a key $\Rightarrow |\mathbf{R}_1| = n$

$\therefore n$ tuples in result
Decomposition Goal #2: Dependency Preservation

Goal: Efficient integrity checks of FD’s

An Example With No Dependency Preservation:

\[ R = (\text{bname}, \text{bcity}, \text{assets}, \text{cname}, \text{lno}, \text{amt}) \]
- \text{bname} \rightarrow \text{bcity} \text{ assets}
- \text{lno} \rightarrow \text{amt} \text{ bname}

Decomposition: \[ R = R_1 \cup R_2 \]

- \[ R_1 = (\text{bname}, \text{assets}, \text{cname}, \text{lno}) \]
- \[ R_2 = (\text{lno}, \text{bcity}, \text{amt}) \]

Lossless, but Not DP. Why?
Decomposition Goal #2: Dependency Preservation (cont.)

Decomposition (cont.): \( R = R_1 \cup R_2 \)

\( R_1 = (\text{bname, assets, cname, lno}) \)
\( R_2 = (\text{lno, bcity, amt}) \)

Lossless, but Not DP. Why?

A: bname \( \rightarrow \) bcity crosses 2 tables

CREATE ASSERTION bname-bcity
CHECK NOT EXISTS
(SELECT *
FROM \( R_1 \) AS \( x_1 \), \( R_2 \) AS \( y_1 \), \( R_1 \) AS \( x_2 \), \( R_2 \) AS \( y_2 \)
WHERE \( x_1\text{.lno} = y_1\text{.lno} \) AND \( x_2\text{.lno} = y_2\text{.lno} \) AND
\( x_1\text{.lno} = x_2\text{.lno} \) AND \( x_1\text{.bname} = x_2\text{.bname} \) AND
\( y_1\text{.bcity} <> y_2\text{.bcity} \)
Decomposition Goal #2: Dependency Preservation

To Ensure Best Possible Efficiency of FD Checks

Ensure that only a SINGLE table be examined for each FD

i.e.: Ensure that $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ can be checked by examining one table as in:

$$R_i = \begin{array}{cccccc}
... & A_1 & ... & A_n & B_1 & ... & B_m & ... \\
\hline
... & \hline & \hline & \hline & \hline & \hline & \hline & \\
\end{array}$$

Above: $R_i$ “covers” the FD, $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$

To Test if Decomposition $R = R_1 \cup \ldots \cup R_n$ is DP,

1. See which FD’s of $R$ are covered by $R_1, \ldots, R_n$
2. Compare closure of (1) to closure of FD’s of $R$
Decomposition Goal #2: Dependency Preservation

More Formally:

To test if \( R = R_1 \cup \ldots \cup R_n \) is dependency preserving wrt \( R \)'s FD set, \( F \):

1. Compute \( F^+ \)
2. Compute \( G \)

\[
G \leftarrow \emptyset \\
\text{For } i \leftarrow 1 \text{ to } n \text{ DO} \\
\quad \text{Add to } G \text{ those FD's in } F^+ \text{ covered by } R_i
\]
3. Compute \( G^+ \)

4. If \( F^+ = G^+ \): Decomposition is DP
   If \( F^+ \neq G^+ \): Decomposition is not DP
Decomposition Goal #2: Dependency Preservation (cont.)

More Formally (cont.):

To test if \( R = R_1 \cup \ldots \cup R_n \) is dependency preserving wrt \( R \)'s FD set, \( F \):

1. Compute \( F^+ \)
2. Compute \( G \)
3. Compute \( G^+ \)
4. Compute \( F^+ - G^+ \)

Example:

\[
F = \{ A \rightarrow B, \quad AB \rightarrow D, \quad C \rightarrow D \} \\
R_1 = (A, B, C); \quad R_2 = (C, D)
\]

Is this decomposition of \((A, B, C, D)\) DP?
Decomposition Goal #2: Dependency Preservation

Example:

\[ F = \{ A \rightarrow B, \ AB \rightarrow D, \ C \rightarrow D \} \]
\[ R_1 = (A, B, C); R_2 = (C, D) \]

Is \( R = R_1 \cup R_2 \) DP?

A: 1. \( F^+ = \{ A \rightarrow B, \ AB \rightarrow D, \ C \rightarrow D \}^+ \)
   
   Note: \((A \rightarrow D) \in F^+\)

2. \( G = \emptyset \cup \{ A \rightarrow B, \ ... \} \cup \{ C \rightarrow D, \ ... \} \)

   Note: \((A \rightarrow D) \notin G\)

3. \( G^+ = \{ ... \} \)

   Note: \((A \rightarrow D) \notin G^+\)

4. \( F^+ \neq G^+ \) because \((A \rightarrow D) \in (F^+ - G^+)\)

\[ \therefore \text{ Decomposition is not DP} \]
Decomposition Goal #2: Dependency Preservation

Example:

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

What is a DP decomposition of \( F \)?

\[ A: \quad R = R_1 \cup R_2 \quad \text{s.t.} \quad R_1 = (A, B, D) ; \ R_2 = (C, D) \]

1. \( F^+ = \{ A \rightarrow B, \ AB \rightarrow D, C \rightarrow D \}^+ \)
2. \( G^+ = \{ A \rightarrow B, AB \rightarrow D, C \rightarrow D \}^+ \)
3. \( F^+ = G^+ \)

Note: \( G^+ \) cannot introduce FD’s not in \( F^+ \)

\[ \therefore \text{Decomposition is DP} \]

Q: Does it satisfy lossless joins?

A: No
Decomposition Goals Summary

**Lossless Joins**

Motivation: Avoid information loss

Idea: No noise introduced when reconstitution universal relation via joins

Test: At each decomposition test: \( R = R_1 \cup R_2 \)

\[(R_1 \cap R_2) \rightarrow R_1 \text{ or } (R_1 \cap R_2) \rightarrow R_2\]

Ensured for: BCNF, 3NF

**Dependency Preservation**

Motivation: Efficient FD assertions

Idea: No gic’s require joins of more than 1 table with itself

Test: \( R = R_1 \cup \ldots \cup R_n \) is DP if closure of FD’s covered by each \( R_i = \text{closure of FD’s covered by } R \) = \( F^+ \)

Ensured for: 3NF
Decomposition Goal #3
Redundancy Avoidance

Redundancy:

1. Name FD of this relation?
   Ans: \( B \rightarrow C \)

2. Name the super keys of this relation
   A: All sets of atts that include \( A \)

3. When do we have redundancy?
   A: When \( \exists \) some FD, \( X \rightarrow Y \) covered by relation & \( X \) not a super key
Redundancy Avoidance

Motivation: Avoid update, deletion anomalies

Idea: Avoid update anomalies, wasted space

Test: For any $X \rightarrow Y$ covered by $R_i$,

$X$ should be a superkey of $R_i$

Ensured for: BCNF
Boyce-Codd Normal Form

What is a Normal Form?

Characterization of schema decomposition in terms of properties it satisfies

BCNF:

Guarantees no redundancy and lossless joins (Not DP!)

Defined: Relation schema \( R \), with FD set \( F \), is in BCNF if:

For all nontrivial \( X \rightarrow Y \) in \( F^+ \): \( X \rightarrow R \)

(i.e.: \( X \) is a super key)
**Example:**

\[ R = (A, B, C) \]
\[ F = \{ A \rightarrow B, B \rightarrow C \} \]

Is \( R \) in BCNF?

**A:** Consider the nontrivial dependencies in \( F^+ \):

1. \( A \rightarrow B, \quad A \rightarrow R \) (A is a key)
2. \( A \rightarrow C, \quad A \rightarrow R \) (A is a key)
3. \( B \not\rightarrow C, \quad B \not\rightarrow A \) (B is not a key)

Therefore, \( R \) not in BCNF
Example:

\[ R = R_1 \cup R_2 \quad R_1 = (A, B); \quad R_2 = (B, C) \]

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

Are \( R_1, R_2 \) in BCNF?

A: 1. Test \( R_1 \):
   - \( A \rightarrow B \) covered, \( A \rightarrow R_1 \) (all other FD’s covered trivial)

2. Test \( R_2 \):
   - \( B \rightarrow C \) covered, \( B \rightarrow R_2 \) (all other FD’s covered trivial)

\[ \therefore R_1, R_2 \text{ in BCNF} \]
BCNF

Decomposition Algorithm

ALGORITHM BCNF (R: Relation, F: FD set)
BEGIN

1. Compute F+
2. Result ← {R}
3. While some $R_i \in$ Result not in BCNF, DO
   a. Choose $(X \rightarrow Y) \in F^+$ s.t.
      $\rightarrow (X \rightarrow Y)$ covered by $R_i$
      $\rightarrow X \rightarrow R_i$
   b. Decompose $R_i$ on $(X \rightarrow Y)$
      $R_{i1} \leftarrow X \cup Y$
      $R_{i2} \leftarrow R_i - Y$
   c. Result ← Result - {$R_i$} $\cup$ {$R_{i1}, R_{i2}$}
4. Return result
END
BCNF

Decomposition Algorithm

Each Step:
Decompose $R_i$ that is not in BCNF

$R_i = \{A, B, C, D, E\}$
$(B \rightarrow CD) \in F^+, \quad B \not\rightarrow R_i$

$R_{i1} = (B, C, D)$
Note: $B \rightarrow CD$
Covered, and
$B \rightarrow R_{i1}$

$R_{i2} = (A, B, E)$

Progress!
BCNF

Decomposition Algorithm (cont.)

Example:

\[ R = (A, B, C, D) \]
\[ F = \{ A \rightarrow B, \ AB \rightarrow D, \ B \rightarrow C \} \]

Decompose \( R \) into BCNF?

1. Compute \( F^+ \):

\[ F^+ = \{ A \rightarrow B, \ AB \rightarrow D, \ B \rightarrow C, \ A \rightarrow C, \ A \rightarrow D, \ AB \rightarrow C, \ AC \rightarrow D, \ AD \rightarrow C, \ ABC \rightarrow D, \ ABD \rightarrow C \} \] + all trivial dep’s
BCNF

Decomposition Algorithm (cont.)

\[ R_i = \{A, B, C, D\} \]

B → C covered, B \→ R_i

\[ R_1 = (B, C) \]

In BCNF

1. B → C &
   B \→ R_1

\[ R_2 = (A, B, D) \]

In BCNF

2. A → B,
3. AB → D,
4. A → D
   covered &
   A \→ R_2, AB \→ R_2

\solution is R = R_1 \cup R_2
BCNF

\[ R = (A, B, C, D, E, H) \]
\[ F = \{A \rightarrow BC, E \rightarrow HA\} \]

Decompose \( R \) into BCNF:

\[ F^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow BC \]
\[ E \rightarrow H, E \rightarrow A, E \rightarrow HA \]
\[ E \rightarrow B, E \rightarrow C, E \rightarrow BC \]
\[ E \rightarrow HB, E \rightarrow HC, E \rightarrow AB \]
\[ E \rightarrow AC, \]
\[ AE \rightarrow \ldots, \]
\[ ABE \rightarrow \ldots, \]
\[ ACE \rightarrow \ldots, \]
\[ ADE \rightarrow \ldots, \]
\[ \ldots\} + \text{all trivial dep's} \]

Note: This will suffice!

Find 2 decompositions, 1 DP and 1 not DP
BCNF Decomposition

\[ R = (A, B, C, D, E, H) \]
\[ F = \{ A \rightarrow BC, E \rightarrow HA \} \]

(Note: \( F_c = F \))

Decomposition #1: \( R = R_1 \cup R_3 \cup R_4 \)

\[ R_1 = (A, B, C) \]
Decompose on \( A \rightarrow BC \)

\[ R_2 = (A, D, E, H) \]
Decompose on \( E \rightarrow HA \)

\[ R_3 = (A, E, H) \]
\[ R_4 = (D, E) \]

Q: Is this DP?

A: Yes. All \( F_c \) covered by \( R_1, R_3, R_4 \). Therefore \( F^+ \) covered
BCNF Decomposition (cont.)

\[ R = (A, B, C, D, E, H) \]
\[ F = \{ A \rightarrow BC, E \rightarrow HA \} \]
(Note: \( F_c = F \))

**Decomposition #2:** \( R = R_1 \cup R_3 \cup R_5 \cup R_6 \)

\[ R = (A, B, C, D, E, H) \]
Decompose on \( A \rightarrow B \)

- \( R_1 = (A, B) \)
- \( R_2 = (A, C, D, E, H) \)
Decompose on \( E \rightarrow HA \)

- \( R_3 = (A, E, H) \)
- \( R_4 = (C, D, E) \)
Decompose on \( E \rightarrow C \)

- \( R_5 = (C, E) \)
- \( R_6 = (E, D) \)

**Q:** Not DP. Why?

**A:** \( A \rightarrow C \) not covered by \( R_1, R_3, R_5, R_6 \).
**More BCNF (cont.)**

**Q:** Can we decompose on FD’s in $F_c$ to get a DP BCNF decomposition?

**A:** Sometimes, BCNF + DP not possible

$$R = (J, K, L)$$
$$F = \{JK \rightarrow L, L \rightarrow K\}$$

### Decomposition #1:

- Decompose on $L \rightarrow K$

  $$R_1 = (L, K) \quad R_2 = (J, L)$$

Not DP: $JK \rightarrow L$ not covered

### Decomposition #2:

- Decompose on $JK \rightarrow L$

  $$R_1 = (J, K, L) \quad R_2 = (J, L)$$

Still not in BCNF (L not a superkey)
Aside

Is This a Realistic Example?

JK $\rightarrow$ L

L $\rightarrow$ K

A: BankerName $\rightarrow$ BranchName

BranchName CustomerName $\rightarrow$ BankerName

Every banker works at one branch

A customer works with the same banker at a given branch
Testing for FDs Across Relations

• Decomposition not dependency preserving => an extra materialized view (MV) for each dependency $\alpha \rightarrow \beta$ in $F_c$ that is not preserved in the decomposition

• The MV is a projection on $\alpha \beta$ of the join of the relations in the decomposition

• DBMS maintains MV when the relations are updated.  
  \[ \Rightarrow \quad \text{No extra coding effort for programmer.} \]

- Space overhead: storing MV
- Time overhead: keeping MV up to date
Multivalued Dependencies

- There are database schemas in BCNF that do not seem to be sufficiently normalized.
- Consider a database
  \[ \text{classes}(\text{course}, \text{teacher}, \text{book}) \]
- The database lists for each course the set of teachers any one of which can be the course’s instructor, and the set of books, all of which are required for the course (no matter who teaches it).
<table>
<thead>
<tr>
<th>course</th>
<th>teacher</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Avi</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Avi</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>Ullman</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Avi</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>Shaw</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Jim</td>
<td>Shaw</td>
</tr>
<tr>
<td>operating systems</td>
<td>Jim</td>
<td></td>
</tr>
</tbody>
</table>

(course, teacher, book) is the only key, and therefore the relation is in BCNF.

Insertion anomalies – i.e., if Sara is a new teacher that can teach database, two tuples need to be inserted:

(database, Sara, DB Concepts)
(database, Sara, Ullman)
Therefore, it is better to decompose *classes* into:

<table>
<thead>
<tr>
<th>course</th>
<th>teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>Avi</td>
</tr>
<tr>
<td>database</td>
<td>Hank</td>
</tr>
<tr>
<td>database</td>
<td>Sudarshan</td>
</tr>
<tr>
<td>operating systems</td>
<td>Avi</td>
</tr>
<tr>
<td>operating systems</td>
<td>Jim</td>
</tr>
</tbody>
</table>

**teaches**

<table>
<thead>
<tr>
<th>course</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>DB Concepts</td>
</tr>
<tr>
<td>database</td>
<td>Ullman</td>
</tr>
<tr>
<td>operating systems</td>
<td>OS Concepts</td>
</tr>
<tr>
<td>operating systems</td>
<td>Shaw</td>
</tr>
</tbody>
</table>

**text**

We shall see that these two relations are in Fourth Normal Form (4NF)
Multivalued Dependencies (MVDs)

Let $R$ be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency

$$\alpha \longrightarrow \beta$$

holds on $R$ if in any legal relation $r(R)$, for all pairs of tuples $t_1$ and $t_2$ in $r$ such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples $t_3$ and $t_4$ in $r$ such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_3[\beta] = t_1[\beta]$$

$$t_3[R - \beta] = t_2[R - \beta]$$

$$t_4[\beta] = t_2[\beta]$$

$$t_4[R - \beta] = t_1[R - \beta]$$
## MVD (Cont.)

**Tabular representation of** $\alpha \rightarrow\rightarrow \beta$

<table>
<thead>
<tr>
<th></th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$R - \alpha - \beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>$a_1...a_i$</td>
<td>$a_i+1...a_j$</td>
<td>$a_j+1...a_n$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$a_1...a_i$</td>
<td>$b_i+1...b_j$</td>
<td>$b_j+1...b_n$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$a_1...a_i$</td>
<td>$a_i+1...a_j$</td>
<td>$b_j+1...b_n$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$a_1...a_i$</td>
<td>$b_i+1...b_j$</td>
<td>$a_j+1...a_n$</td>
</tr>
</tbody>
</table>
Example

• Let $R$ be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets. $Y, Z, W$

• We say that $Y ightarrowightarrow Z$ ($Y$ multidetermines $Z$) if and only if for all possible relations $r(R)$
  
  $< y_1, z_1, w_1 > \in r$ and $< y_1, z_2, w_2 > \in r$

  implies

  $< y_1, z_1, w_2 > \in r$ and $< y_1, z_2, w_1 > \in r$

• Note that since the behavior of $Z$ and $W$ are identical it follows that $Y \rightarrowightarrow Z$ if $Y \rightarrowightarrow W$
Example (Cont.)

• In our example:

\[ \text{course} \rightarrow \rightarrow \text{teacher} \]
\[ \text{course} \rightarrow \rightarrow \text{book} \]

• The above formalizes the notion that a particular value of \( Y (\text{course}) \) has associated with it a set of values of \( Z (\text{teacher}) \) and a set of values of \( W (\text{book}) \), and these two sets are in some sense independent of each other.

Note:

If \( Y \rightarrow Z \) then \( Y \rightarrow\rightarrow Z \)

Indeed we have (in above notation) \( Z_1 = Z_2 \)

The claim follows.
Use of Multivalued Dependencies

• We use multivalued dependencies in two ways:
  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
  2. To specify constraints on the set of legal relations. We shall thus concern ourselves only with relations that satisfy a given set of functional and multivalued dependencies.

• If a relation $r$ fails to satisfy a given multivalued dependency, we can construct a relation $r'$ that does satisfy the multivalued dependency by adding tuples to $r$. 
Fourth Normal Form

• A relation schema $R$ is in 4NF with respect to a set $D$ of functional and multivalued dependencies if for all multivalued dependencies in $D^+$ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:

  $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)

  $\alpha$ is a superkey for schema $R$

• If a relation is in 4NF it is in BCNF