Relational Normalization: Contents

- Motivation
- Functional Dependencies
- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form
- Decomposition Algorithms
- Multivalued Dependencies and Fourth Normal Form
- Join Dependencies and Fifth Normal Form
- Critique of Relational Normalization

Relational Database Design

- Choose and apply criteria for grouping attributes in relation schemas
- Necessary to characterize some groupings as better than others
- A good design can be obtained by
  - intuition, craftmanship of designer
  - design in a more expressive data model (such as E-R) and translate to relational
  - thru normalization theory
    - relational normal forms
    - important stage in the development of relational theory
    - normal forms can be defined for other data models (e.g., E-R)
## Relational Database Design: Rationale

- **Logical level**
  - clarity of semantics of data
  - convenience of query formulation
  - information content
  - applies both to base relations and views (virtual relations)

- **Physical level**
  - storage optimization
  - efficiency of access (query evaluation)
  - simplicity of updates
  - applies only to stored base relations

- The grouping of attributes in physical relation schemas has a significant effect on storage space
### Example of Relational Schema

<table>
<thead>
<tr>
<th>Employee</th>
<th>SSN</th>
<th>FName</th>
<th>MInit</th>
<th>LName</th>
<th>BDate</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>SuperSSN</th>
<th>DNo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department</th>
<th>DNumber</th>
<th>DName</th>
<th>DMgr</th>
<th>MgrStartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DNumber</td>
<td>DLocation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>PNumber</th>
<th>PName</th>
<th>PLocation</th>
<th>DNumber</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>WorksOn</th>
<th>PNumber</th>
<th>PName</th>
<th>Hours</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Dependent</th>
<th>SSN</th>
<th>DependentName</th>
<th>Sex</th>
<th>BDate</th>
<th>Relationship</th>
</tr>
</thead>
</table>

- In terms of entities and relationships
  - DeptLocations represents a multivalued attribute of Department
  - WorksOn represents a M-N relationship between Employee and Project
  - Dependent merges a relationship and a less important entity

- There are many other ways to organize the same data

- No claim that the database represents the “real world”
Logical Level: Semantics of Relation Attributes

- Meaning (semantics) is associated with attributes = interpretation of values in relation tuples
- The clearer the semantics of a relation, the better the design of the schema
- A plausible set of rules:
  - each tuple should represent one entity or one relationship instance
  - attributes of different entities and relationships should not be mixed in the same relation
  - only foreign keys should be used to refer to other entities (most probably with referential integrity)

A not so Good Design and an Improved Normalization

<table>
<thead>
<tr>
<th>EmpDept</th>
<th>EName</th>
<th>SSN</th>
<th>BDate</th>
<th>Address</th>
<th>DNumber</th>
<th>DName</th>
<th>DMgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>EName</td>
<td>SSN</td>
<td>BDate</td>
<td>Address</td>
<td>DNumber</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td>DNumber</td>
<td>DName</td>
<td>DMgr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Normalization, October 9, 2008 – 4
Clear semantics but poor design: attributes from distinct real-world entities (Employee, Department) are mixed in EmpDept.

Maybe acceptable for views, but causes problems when done with base relations.

Redundant information causes update anomalies.

A Similar Example

<table>
<thead>
<tr>
<th>EmpProj</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>PNumber</td>
<td>Hours</td>
<td>EName</td>
<td>PName</td>
<td>PLocation</td>
<td>DNumber</td>
</tr>
</tbody>
</table>

↓

<table>
<thead>
<tr>
<th>Proj</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PNumber</td>
<td>PName</td>
<td>PLocation</td>
<td>DNumber</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Emp

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>EName</td>
</tr>
</tbody>
</table>

WorksOn

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>PNumber</td>
<td>Hours</td>
</tr>
</tbody>
</table>

Insertion Anomalies

<table>
<thead>
<tr>
<th>EmpDept</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EName</td>
<td>SSN</td>
<td>BDate</td>
<td>Address</td>
<td>DNumber</td>
<td>DName</td>
<td>DMgr</td>
<td></td>
</tr>
</tbody>
</table>

- Insert a new employee into EmpDept
  - compatibility of department data to be checked with other tuples of EmpDept
  - insert nulls if no department data is input

- Insert a new department without employee into EmpDept
  - tuple with nulls for employee data (but SSN is key!)
  - remove nulls (complex insertion) when the first employee of the department is inserted
Deletion and Modification Anomalies

EmpDept

EName | SSN | BDate | Address | DNumber | DName | DMgr

- Delete from EmpDept the last employee of a department
  → lose information or complex update
- Change manager of a department
  → affects several tuples to avoid inconsistency
Spurious Tuples

- Bad design may result in erroneous results for joins

EmpProj

<table>
<thead>
<tr>
<th>SSN</th>
<th>PNumber</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>32.5</td>
<td>Smith</td>
<td>ProdX</td>
<td>Bellaire</td>
</tr>
<tr>
<td>1234</td>
<td>2</td>
<td>7.5</td>
<td>Smith</td>
<td>ProdY</td>
<td>Sugarland</td>
</tr>
<tr>
<td>6668</td>
<td>3</td>
<td>40.0</td>
<td>Narayan</td>
<td>ProdZ</td>
<td>Houston</td>
</tr>
<tr>
<td>4534</td>
<td>1</td>
<td>20.0</td>
<td>English</td>
<td>ProdX</td>
<td>Bellaire</td>
</tr>
<tr>
<td>4534</td>
<td>2</td>
<td>20.0</td>
<td>English</td>
<td>ProdY</td>
<td>Sugarland</td>
</tr>
</tbody>
</table>

\[
\downarrow
\]

EmpLocs

<table>
<thead>
<tr>
<th>EName</th>
<th>PLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Bellaire</td>
</tr>
<tr>
<td>Smith</td>
<td>Sugarland</td>
</tr>
<tr>
<td>Narayan</td>
<td>Houston</td>
</tr>
<tr>
<td>English</td>
<td>Bellaire</td>
</tr>
<tr>
<td>English</td>
<td>Sugarland</td>
</tr>
</tbody>
</table>

- \( \text{EmpProj} \neq \text{join of its projections EmpLocs and EmpProj1} \)
  - common attribute (PLocation) is not a key or a foreign key
  - joining yields more tuples than in EmpProj (“spurious tuples”)
- **Lossless-join property**: guarantees meaningful results for joins

Normalization, October 9, 2008 – 7
Spurious Tuples in \textit{EmpProj1} $\bowtie$ \textit{EmpLocs}

\begin{tabular}{|c|c|c|c|c|c|}
\hline
SSN & P\# & Hours & PName & PLocation & EName \\
\hline\hline
1234 & 1 & 32.5 & ProdX & Bellaire & Smith \\
\hline
1234 & 1 & 32.5 & ProdX & Bellaire & English \\
\hline
1234 & 2 & 7.5 & ProdY & Sugarland & Smith \\
\hline
1234 & 2 & 7.5 & ProdY & Sugarland & English \\
\hline
6668 & 3 & 40.0 & ProdZ & Houston & Narayan \\
\hline
4534 & 1 & 20.0 & ProdX & Bellaire & Smith \\
\hline
4534 & 1 & 20.0 & ProdX & Bellaire & English \\
\hline
4534 & 2 & 20.0 & ProdY & Sugarland & Smith \\
\hline
4534 & 2 & 20.0 & ProdY & Sugarland & English \\
\hline
\end{tabular}

Functional Dependencies (FDs)

- **Definition 1:**
  \[ X \rightarrow Y \] holds in relation \( R(A_1, \ldots, A_n) \), with \( X, Y \subseteq \{ A_1, \ldots, A_n \} \), if for every pair of tuples \( t_1, t_2 \) such that \( t_1[X] = t_2[X] \) then \( t_1[Y] = t_2[Y] \)

- **Definition 2:**
  \[ X \rightarrow Y \] if there cannot exists different tuples \( t_1, t_2 \) such that \( t_1[X] = t_2[X] \)

- Functional dependencies are constraints, i.e., belong to the schema
  - cannot be deduced from extension
  - can only be verified or invalidated on extension (one counterexample is enough to invalidate)
• Definition 1:
  ◦ all tuples that agree on $X$ also agree on $Y$
  ◦ values for $X$ functionally determine values for $Y$
  ◦ definition applies with $t_1 = t_2$

• Special case: if $X$ is a key or superkey then $X \rightarrow Y$ for all $Y \subseteq \{A_1, \ldots, A_n\}$

---

**FDs: Example**

### EmpDept

<table>
<thead>
<tr>
<th>EName</th>
<th>SSN</th>
<th>BDate</th>
<th>Address</th>
<th>D#</th>
<th>DName</th>
<th>DMgrSSN</th>
</tr>
</thead>
</table>

- SSN $\rightarrow$ EName
- SSN $\rightarrow$ D#
- SSN $\rightarrow$ \{EName,BDate\}
- D# $\rightarrow$ DMgrSSN

### EmpProj

<table>
<thead>
<tr>
<th>SSN</th>
<th>P#</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLocation</th>
</tr>
</thead>
</table>

- \{SSN,P\#\} $\rightarrow$ Hours
- P# $\rightarrow$ \{PName,PLocation\}
## FDs: Example, cont.

### EmpDept

<table>
<thead>
<tr>
<th>EName</th>
<th>SSN</th>
<th>BDate</th>
<th>Address</th>
<th>D#</th>
<th>DName</th>
<th>DMgrSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>1234</td>
<td>21/07/39</td>
<td>...</td>
<td>1</td>
<td>Research</td>
<td>1234</td>
</tr>
<tr>
<td>Narayan</td>
<td>6668</td>
<td>18/01/43</td>
<td>...</td>
<td>1</td>
<td>Research</td>
<td>1234</td>
</tr>
<tr>
<td>English</td>
<td>4534</td>
<td>8/05/53</td>
<td>...</td>
<td>2</td>
<td>Account</td>
<td>4534</td>
</tr>
<tr>
<td>Wong</td>
<td>9788</td>
<td>30/11/49</td>
<td>...</td>
<td>3</td>
<td>Admin</td>
<td>9788</td>
</tr>
<tr>
<td>Zelaya</td>
<td>6677</td>
<td>23/08/60</td>
<td>...</td>
<td>3</td>
<td>Admin</td>
<td>9788</td>
</tr>
</tbody>
</table>

\[ \text{DMgrSSN} \rightarrow \text{D#} \]

### EmpProj

<table>
<thead>
<tr>
<th>SSN</th>
<th>P#</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>32.5</td>
<td>Smith</td>
<td>ProdX</td>
<td>Bellaire</td>
</tr>
<tr>
<td>1234</td>
<td>2</td>
<td>7.5</td>
<td>Smith</td>
<td>ProdY</td>
<td>Sugarland</td>
</tr>
<tr>
<td>6668</td>
<td>3</td>
<td>40.0</td>
<td>Narayan</td>
<td>ProdZ</td>
<td>Houston</td>
</tr>
<tr>
<td>4534</td>
<td>1</td>
<td>20.0</td>
<td>English</td>
<td>ProdX</td>
<td>Bellaire</td>
</tr>
<tr>
<td>4534</td>
<td>2</td>
<td>20.0</td>
<td>English</td>
<td>ProdY</td>
<td>Sugarland</td>
</tr>
</tbody>
</table>

\[ \text{PLocation} \rightarrow \text{PName} \text{ does not hold} \]

## Inference of New FDs

- Only most obvious (or most important) dependencies are explicitly specified
- Many other dependencies can be deduced from them
- Closure $F^+$ of a set $F$ of dependencies:
  - set of all dependencies in $F$ + those implied by $F$
  - dependency $X \rightarrow Y$ is implied by $F$ if $X \rightarrow Y$ is valid in all relation instances for which $F$ is valid
  - $X \rightarrow Y \in F$ iff $F \models X \rightarrow Y$
Armstrong’s Inference Rules

A1 (Reflexivity) If $Y \subseteq X$, then $X \rightarrow Y$

A2 (Augmentation) If $X \rightarrow Y$, then $XZ \rightarrow YZ$ (and $XZ \rightarrow Y$)

A3 (Transitivity) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

A1, A2, A3 form a sound and complete set of inference rules

Some useful additional inference rules

A4 (Decomposition) If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

A5 (Union) If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

A6 (Pseudotransitivity) If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Inference rule: systematic way of constructing functional dependencies in $F^+$
Example: Deriving FDs

EmpDept

<table>
<thead>
<tr>
<th>EName</th>
<th>SSN</th>
<th>BDate</th>
<th>Address</th>
<th>D#</th>
<th>DName</th>
<th>DMgrSSN</th>
</tr>
</thead>
</table>

SSN → EName   D# → DName   ⇒   SSN → DName
SSN → BDate   D# → DMgrSSN  SSN → DMgrSSN
SSN → Address :      SSN → {EName,DName}
SSN → D#       :      :

17

Example: Deriving FDs, cont.

EmpProj

<table>
<thead>
<tr>
<th>SSN</th>
<th>P#</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLocation</th>
</tr>
</thead>
</table>

P# → P#   ⇒   P# → {P#,PLocation}

{SSN,P#} → Hours   {SSN,P#,Hours} → ... 7 possibilities
{SSN,P#} → EName   {SSN,P#,EName} → ... 7 possibilities
{SSN,P#} → PName   {SSN,P#,Hours,EName} → ... 3 possibilities
{SSN,P#} → PLocation :
{SSN,P#} → {Hours,EName} :

15 in all

18
Proof of the Inference Rules

- Viewed as a formal deductive system
  - a set of given FDs plays the role of axioms
  - Armstrong’s rules: sound and complete inference rules to derive new FDs from the axioms without invoking definition of what is an FD

- Structure of formal deductive system is clear and simple
- Armstrong’s rules can be proven with the definition of FD
- Derived rules (A4, A5, A6, and others) can be proven from Armstrong’s basic rules

Closure of Attribute Sets

- Closure $F(X^+)$ of a set of attributes $X$ under a set $F$ of FDs:
  - set of attributes $A$ such that $X \rightarrow A$ can be deduced from $F$ by Armstrong’s rules
  - allows to tell at a glance whether FD $X \rightarrow Y$ follows from $F$
- Algorithm for determining the closure of $X$ under $F$

\[
X^+ := X;
\]

\[
\text{repeat}
\]

\[
\text{old}X^+ := X^+;
\]

\[
\text{for each FD } Y \rightarrow Z \text{ in } F \text{ do}
\]

\[
\text{if } X^+ \supseteq Y \text{ then } X^+ := X^+ \cup Z;
\]

\[
\text{until } (X^+ = \text{old}X^+)
\]
• Algorithm starts setting $X^+$ to all attributes in $X$: by A1 all these attributes are functionally dependent on $X$

• Using A3 and A4 we add attributes to $X^+$, using each FD in $F$

• We keep going through all dependencies in $F$ (repeat loop) until no more attributes are added to $X^+$ during a complete cycle (for loop) through the dependencies in $F$

### Closure of Attribute Sets: Example

<table>
<thead>
<tr>
<th>EmpProj</th>
<th>SSN</th>
<th>P#</th>
<th>Hours</th>
<th>EName</th>
<th>PName</th>
<th>PLocation</th>
</tr>
</thead>
</table>

$\{\text{SSN}\}^+ = \{\text{SSN}, \text{EName}\}$

$\{\text{P}\#\}^+ = \{\text{P}\#, \text{PName}, \text{PLocation}\}$

$\{\text{SSN}, \text{P}\#\}^+ = \{\text{SSN}, \text{P}\#, \text{EName}, \text{PName}, \text{PLocation}, \text{Hours}\}$
Equivalence of Sets of FDs

- Two sets of FDs $F$ and $G$ are equivalent if
  - every FD in $F$ can be inferred from $G$, and
  - every FD in $G$ can be inferred from $F$
- Hence $F$ and $G$ are equivalent if $F^+ = G^+$
- $F$ covers $G$ if every FD in $G$ can be inferred from $F$ (i.e., if $G^+ \subseteq F^+$)
- $F$ and $G$ are equivalent if $F$ covers $G$ and $G$ covers $F$
- There is an algorithm for checking equivalence of sets of FDs

Minimal Sets of FDs

- A set $F$ of FDs is minimal if
  1. every FD in $F$ has a single attribute for its RHS
  2. no subset of $F$ is equivalent to $F$
  3. no dependency $X \rightarrow A$ in $F$ can be replaced by $Y \rightarrow A$ with $Y \subset X$ and yield a set of dependencies equivalent to $F$
- Minimal cover of a set $F$ of FDs: minimal set of dependencies $F_{\text{min}}$ equivalent to $F$

- Every set of FDs has a minimal cover
- There can be several minimal covers for a set of FDs
Finding Minimal Covers

Algorithm for finding a minimal cover $G$ of $F$

1. Set $G := F$

2. Replace each FD $X \rightarrow \{A_1, \ldots, A_n\}$ in $G$ by $n$ FDs $X \rightarrow A_1, \ldots, X \rightarrow A_n$

3. For each FD $X \rightarrow A$ in $G$
   for each attribute $B$ that is an element of $X$
   if $(G - \{X \rightarrow A\}) \cup \{(X - \{B\}) \rightarrow A\}$ is equivalent to $G$
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in $G$

4. For each remaining FD $X \rightarrow A$ in $G$
   if $(G - \{X \rightarrow A\})$ is equivalent to $G$
   then remove $X \rightarrow A$ from $G$

Several Minimal Covers: Example

$AB \rightarrow C \quad D \rightarrow E \quad G$
$C \rightarrow A \quad BE \rightarrow C \quad AB \rightarrow C \quad D \rightarrow E \quad CG \rightarrow B$
$BC \rightarrow D \quad CG \rightarrow BD \quad C \rightarrow A \quad D \rightarrow G \quad CG \rightarrow D$
$ACD \rightarrow B \quad CE \rightarrow AG \quad BC \rightarrow D \quad BE \rightarrow C \quad CE \rightarrow A$
$D \rightarrow E \quad CE \rightarrow G$  

● Two minimal covers

\begin{array}{c|c|c}
AB & C & D \rightarrow G \\
C & A & BE \rightarrow C \\
BC & D & CG \rightarrow D \\
CD & B & CE \rightarrow G \\
D & E & \\
\end{array}  

\begin{array}{c|c|c}
AB & C & D \rightarrow G \\
C & A & BE \rightarrow C \\
BC & D & CG \rightarrow B \\
D & E & CE \rightarrow G \\
\end{array}
First minimal cover

• $CE \rightarrow A$ implied by $C \rightarrow A$
• $CG \rightarrow B$ implied by $CG \rightarrow D$, $C \rightarrow A$, $ACD \rightarrow B$
• $ACD \rightarrow B$ replaced by $CD \rightarrow B$ since $C \rightarrow A$

Second minimal cover

• $CE \rightarrow A$ implied by $C \rightarrow A$
• $CG \rightarrow D$ implied by $CG \rightarrow B$, $BC \rightarrow D$
• $ACD \rightarrow B$ implied by $C \rightarrow A$, $D \rightarrow G$, $CG \rightarrow B$

Normalization

• Process of decomposing unsatisfactory relation schemas into smaller relations without those undesirable aspects (e.g., update anomalies)

Normalization

- Normal form = condition (integrity constraint) to certify that a relation schema is in a particular form
Normal Forms

- Several types of normal forms based on different integrity constraints
  - **Functional Dependencies**
    - 1st, 2nd, 3rd, Boyce-Codd, Improved 3rd Normal Forms
  - **Functional and Inclusion Dependencies**
    - Inclusion Normal Form
  - **Functional and Multivalued Dependencies**
    - 4th Normal Form
  - **Functional, Multivalued, and Join Dependencies**
    - 5th Normal Form

How Far to Normalize

- Relations not always normalized to the highest possible form, e.g., for performance reasons
- Higher normals forms
  - ⇒ smaller relations
  - ⇒ more joins in queries
  - ⇒ space/time or query/update tradeoff
### First Normal Form (1NF)

- Relations as defined in the relational model
- Forbids composite attributes, multivalued attributes, nested relations (relations within relations)
- Relational-model limitation to 1NF:
  - historical reasons (simplify file management)
  - on retrospect, a mistake

### Multivalued attributes

<table>
<thead>
<tr>
<th>Department</th>
<th>DName</th>
<th>DNumber</th>
<th>DMgr</th>
<th>{DLocations}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>333445555</td>
<td>{Bellaire,Sugarland,Houston}</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>987654321</td>
<td>{Stafford}</td>
<td></td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>888665555</td>
<td>{Houston}</td>
<td></td>
</tr>
</tbody>
</table>

↓ 1NF Normalization

<table>
<thead>
<tr>
<th>Department</th>
<th>DName</th>
<th>DNumber</th>
<th>DMgr</th>
<th>DLocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>Bellaire</td>
<td>333445555</td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td>5</td>
<td>Sugarland</td>
<td>333445555</td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td>5</td>
<td>Houston</td>
<td>333445555</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>Stafford</td>
<td>987654321</td>
<td></td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>Houston</td>
<td>888665555</td>
<td></td>
</tr>
</tbody>
</table>
• Redundancy: for each value of DLocation, other attributes has to be repeated
• Normalized relation is less adequate with respect to real-world intuition

### Nested Relations

<table>
<thead>
<tr>
<th>SSN</th>
<th>EName</th>
<th>PNumber</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>Smith, John B.</td>
<td>1</td>
<td>32.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>666884444</td>
<td>Narayan, Ramesh K.</td>
<td>3</td>
<td>46.0</td>
</tr>
<tr>
<td>999888777</td>
<td>Zelaya, Alicia J.</td>
<td>1</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>20.0</td>
</tr>
<tr>
<td>453453453</td>
<td>English, Joyce A.</td>
<td>30</td>
<td>30.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>10.0</td>
</tr>
</tbody>
</table>

\[ 1\text{NF Normalization (unnest operation)} \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>EName</th>
<th>PNumber</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 31

• Nested relations: value of an attribute of a relation can be a relation
• SSN: primary key of EmpProj
• PNumber: primary key of each nested Projs relation
• Unnest operation transforms the relation into 1NF
• The primary key has to be propagated into the embedded relation
Second Normal Form (2NF)

- **Prime attribute**: attribute which is a member of a key

- **Full functional dependency**: an FD $X \rightarrow Z$ where removal of any attribute from $X$ invalidates the dependency
  - $fd1$ is a full FD, neither $SSN \rightarrow Hours$ nor $PNumber \rightarrow Hours$ hold
  - $SSN,PNumber \rightarrow EName$ is not a full FD (i.e., is a **partial dependency**) since $SSN \rightarrow EName$ ($fd2$) also holds

- Definition: a relation schema $R$ is in 2NF if every nonprime attribute is fully functionally dependent on every key

- A relation where all keys are single attributes is automatically in 2NF

---

Second Normal Form : Example

<table>
<thead>
<tr>
<th>$fd1$</th>
<th>$fd2$</th>
<th>$fd3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SSN$</td>
<td>$PNumber$</td>
<td>$Hours$</td>
</tr>
<tr>
<td>$EName$</td>
<td>$PName$</td>
<td>$PLocation$</td>
</tr>
</tbody>
</table>

⇓ 2NF Normalization

- $fd1$ |
- $fd2$ |
- $fd3$ |

Emp.Proj

<table>
<thead>
<tr>
<th>$fd1$</th>
<th>$fd2$</th>
<th>$fd3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SSN$</td>
<td>$PNumber$</td>
<td>$Hours$</td>
</tr>
<tr>
<td>$EName$</td>
<td>$PName$</td>
<td>$PLocation$</td>
</tr>
</tbody>
</table>

Emp.Proj1

<table>
<thead>
<tr>
<th>$fd1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SSN$</td>
</tr>
<tr>
<td>$PNumber$</td>
</tr>
</tbody>
</table>

Emp.Proj2

<table>
<thead>
<tr>
<th>$fd2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SSN$</td>
</tr>
<tr>
<td>$EName$</td>
</tr>
</tbody>
</table>

Emp.Proj3

<table>
<thead>
<tr>
<th>$fd3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNumber$</td>
</tr>
<tr>
<td>$PName$</td>
</tr>
<tr>
<td>$PLocation$</td>
</tr>
</tbody>
</table>
Third Normal Form (3NF)

- $X \rightarrow Z$ is a **transitive functional dependency** if $\exists Y$ such that $X \rightarrow Y$ and $Y \rightarrow Z$ (and $Z$ is not a subset of a key)
  - SSN $\rightarrow$ DMgr is a transitive FD
    (SSN $\rightarrow$ DNumber and DNumber $\rightarrow$ DMgr hold)
  - SSN $\rightarrow$ EName is a non-transitive FD
    (there is no set of attributes $X$ where SSN $\rightarrow$ X and X $\rightarrow$ EName)

Third Normal Form: Definitions

3 definitions for a relation schema $R$ to be in 3NF:

1. $R$ is in 2NF and no nonprime attribute is transitively dependent on a key
2. for all $X \rightarrow A \in F^+$
   - either $X$ is a superkey,
   - or $A$ is a prime attribute
3. every nonprime attribute is
   - fully functionally dependent on every key
   - nontransitively dependent on every key

Normalization, October 9, 2008 – 35
Third Normal Form: Example

$$\downarrow 3NF\text{ Normalization}$$

Normalization: Example
Normalization: Example, cont.

<table>
<thead>
<tr>
<th>PropertyId#</th>
<th>CountyName</th>
<th>Lot#</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lots1B</th>
<th>Area</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lots 1NF

Lots 1NF and BCNF

Boyce-Codd Normal Form (BCNF)

- A relation schema $R$ is in BCNF if, for all $X \rightarrow A \in F^+$, $X$ is a superkey of $R$ (and $A \notin X$)
- BCNF is an improved 3NF: all dependencies result from keys
- A relation with 2 attributes is automatically in BCNF
- Most 3NF relations are also in BCNF
- Intuition of 3NF/BCNF: FDs concern the key, the whole key, and nothing but the key
### A relation in 3NF but not in BCNF

<table>
<thead>
<tr>
<th>Patient</th>
<th>Hospital</th>
<th>Doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Alachua</td>
<td>Atkinson</td>
</tr>
<tr>
<td>Lee</td>
<td>Shands</td>
<td>Smith</td>
</tr>
<tr>
<td>Marks</td>
<td>Alachua</td>
<td>Atkinson</td>
</tr>
<tr>
<td>Marks</td>
<td>Shands</td>
<td>Shaw</td>
</tr>
<tr>
<td>Rao</td>
<td>North Florida</td>
<td>Nefzger</td>
</tr>
</tbody>
</table>

**PatVisit**

<table>
<thead>
<tr>
<th>Patient</th>
<th>Hospital</th>
<th>Doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Atkinson</td>
<td></td>
</tr>
<tr>
<td>Lee</td>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>Marks</td>
<td>Atkinson</td>
<td></td>
</tr>
<tr>
<td>Marks</td>
<td>Shaw</td>
<td></td>
</tr>
<tr>
<td>Rao</td>
<td>Nefzger</td>
<td></td>
</tr>
</tbody>
</table>

**PatVisit has 2 keys: (Patient, Hospital) and (Patient, Doctor)**

**To reach BCNF without losing information an inter-relation constraint is needed, namely (Patient, Hospital) → Doctor**
Relational Decomposition

- Normalization decomposes relation schemas with undesirable aspects into smaller relations.
- Consider a relation schema \( R(A_1, \ldots, A_n) \) and a set of dependencies \( F \).
- Goal: produce a decomposition \( D \) of \( R \) into \( m \) relation schemas \( D = \{ R_1, \ldots, R_m \} \) where each \( R_i \) contains a subset of \( \{ A_1, \ldots, A_n \} \), and
  - every attribute \( A_i \) in \( R \) appears in at least one \( R_i \)
  - each relation \( R_i \) is at least in BCNF or in 3NF
- Extreme decomposition approach: start with a universal relation schema containing all the DB attributes and a set of FD
- \( \Rightarrow \) **Universal relation assumption** is needed: every attribute is unique, i.e., attributes with the same name in different relations have the same meaning.

Relational Decomposition, cont.

- Requiring each individual relation to be in a given normal form does not alone guarantee a good design.
- BCNF (or 3NF) measure “goodness” for individual relations based on their keys and functional dependencies.
- A set of relations must possess additional properties to ensure a good design:
  - dependency preservation
  - lossless (nonadditive) join
- In traditional approach to relational database design, dependency preservation is required because of the weak support for integrity constraints by DBMSs.
Dependency Preservation

- Consider a relation schema \( R(A_1, \ldots, A_n) \), a set of dependencies \( F \), and a decomposition \( D \) of \( R \) into \( m \) relation schemas \( D = \{ R_1, \ldots, R_m \} \).

- Dependency preservation: each FD \( X \rightarrow Y \) of \( F \) should appear explicitly or be inferrable in one relation schema \( R_i \).

- Otherwise, to preserve information, inter-relation FDs are needed (i.e., dependencies that hold on a join of several relations of the decomposition).

---

### Dependency Preservation: Example

<table>
<thead>
<tr>
<th>PropertyId#</th>
<th>CountyName</th>
<th>Lot#</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fd3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \downarrow \] BCNF Normalization with loss of a key

<table>
<thead>
<tr>
<th>PropertyId#</th>
<th>Area</th>
<th>Lot#</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lots1AY

<table>
<thead>
<tr>
<th>Area</th>
<th>CountyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd3</td>
<td></td>
</tr>
</tbody>
</table>

- An inter-relation constraint must express that \((\text{Lot#}, \text{CountyName})\) is the key of the join of Lots1AX and Lots1AY.
Dependency Preservation: Formalization

- A decomposition $D$ must preserve the dependencies: collection of all dependencies that hold on individual relations $R_i$ must be equivalent to $F$
- Formally
  - Projection $\prod_F(R_i)$ of $F$ on $R_i$: set of FDs $X \rightarrow Y$ in $F^+$ such that $(X \cup Y) \subseteq R_i$ (their left- and right-hand side attributes are in $R_i$)
  - A decomposition $D = \{R_1, \ldots, R_m\}$ is dependency-preserving if $(\prod_F(R_1) \cup \ldots \cup \prod_F(R_m))^+ = F^+$
- Dependency preservation enables checking that FDs in $F$ hold by checking them on each relation $R_i$ individually

Dependency Preserving Decomposition Algorithm

Find a dependency preserving decomposition $D = \{R_1, \ldots, R_m\}$ of a relation $R$ w.r.t. a set of FDs $F$ such that each $R_i$ is in 3NF

1. Find a minimal cover $G$ of $F$
2. For each $X$ and FD $X \rightarrow A$ in $G$
   - create relation $R_i$ in $D$ with attributes $\{X \cup A_1 \cup \ldots \cup A_k\}$ where $X \rightarrow A_1, \ldots, X \rightarrow A_k$ are the only FDs in $G$ with $X$ as left-hand side
   - $(X$ is the key of $R_i)$
3. If attributes in $G$ are not placed in any $R_j$, create another relation (all-key, without FD) in $D$ for those attributes
   - There is no similar algorithm to reach BCNF
   - There are several minimal covers in general ⇒ nondeterminism
   - Not guaranteed to be lossless (nonadditive)
Lossless (Nonadditive) Join Property

- Ensures that no spurious tuples appear when relations in the decomposition are joined.
- Decomposition $D = \{R_1, \ldots, R_m\}$ of $R$ has the lossless-join property w.r.t. a set $F$ of FDs if, for every relation instance $r(R)$ whose tuples satisfy all the FDs in $F$:
  \[ \Pi_{R_1}(r) \Join \ldots \Join \Pi_{R_m}(r) = r \]
  (This is the general form of join dependency, see later)
- Ensures that whenever a relation instance $r(R)$ satisfies $F$, no spurious tuples are generated by joining the decomposed relations $r(R_i)$.
- Necessary to generate meaningful results for queries involving joins.
- There exists an algorithm for testing whether a decomposition $D$ satisfies the lossless-join property with respect to a set $F$ of FDs.

Properties of the Nonadditive Join (1)

Decomposition $D = \{R_1, R_2\}$ of $R$ has the nonadditive-join property w.r.t. $F$ iff either $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ or $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ is in $F^+$.
Properties of the Nonadditive Join (2)

If \( D = \{R_1, \ldots, R_m\} \) of \( R \) has the nonadditive-join property w.r.t. \( F \), and \( D_i = \{Q_1, \ldots, Q_k\} \) of \( R_i \) has the nonadditive-join property w.r.t. \( \prod_{R_i}(F) \), then
\[
D = \{R_1, \ldots, R_{i-1}, Q_1, \ldots, Q_k, R_{i+1}, \ldots, R_m\}
\]
has the nonadditive join property w.r.t. \( F \)

<table>
<thead>
<tr>
<th>Emp</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>PNumber</td>
<td>Hours</td>
<td>EName</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\downarrow\]

<table>
<thead>
<tr>
<th>Emp1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>PNumber</td>
<td>Hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Emp2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>EName</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lossless Join Decomposition Algorithm

Decompose \( R \) into a lossless join decomposition \( D = \{R_1, \ldots, R_m\} \) w.r.t. \( F \) such that each \( R_i \) is in BCNF

1. Set \( D := \{R\} \)
2. While there is a relation schema \( Q \) in \( D \) that is not BCNF
   
   begin
   
   find an FD \( X \rightarrow Y \) in \( Q \) that violates BCNF;
   
   replace \( Q \) in \( D \) by two relations \( (Q - Y) \) and \( (X \cup Y) \)
   
   end

- Does not necessarily preserve the FDs
- The algorithm is very simple: normalizing up to BCNF = break relations according to FDs
**Combined Decomposition Algorithm**

Produce a lossless join and dependency-preserving decomposition into 3NF

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

2. For each $X$ in an FD $X \rightarrow A$ in $G$
   - create a relation in $D$ with attributes $\{X \cup A_1 \cup \ldots \cup A_k\}$ where
     - $X \rightarrow A_1, \ldots, X \rightarrow A_k$ are the only FDs in $G$ with $X$ as left-hand side
     - $(X$ is the key of this relation$)$

3. If none of the relations in $D$ contains a key of $R$, create one more relation schema in $D$ that contains attributes that form a key of $R$
   - Step 3 of previous algorithm is not needed because the key will include any unplaced attributes (i.e., attributes not participating in any FD)

---

**Algorithm for Finding a Key**

Finding a key $F$ for a relation schema $R$ based on a set $F$ of FDs

1. Set $K := R$

2. For each attribute $A$ in $K$
   - compute $(K - A)^+$ with respect to $F$
   - if $(K - A)^+$ contains all attributes in $R$ then set $K := K - \{A\}$;

   - Determines only one key out of the possible candidates keys for $R$
   - Key returned depends on the order in which attributes are removed
Decomposition Example

<table>
<thead>
<tr>
<th>Stock</th>
<th>Model#</th>
<th>Serial#</th>
<th>Price</th>
<th>Color</th>
<th>Name</th>
<th>Year</th>
</tr>
</thead>
</table>

 Dependencies
\{M,S\} → \{P,C\}, \{M\} → \{N\}, \{S\} → \{Y\}, \{N,Y\} → \{P\}

 Minimal Cover
\{M,S\} → \{C\}, \{M\} → \{N\}, \{S\} → \{Y\}, \{N,Y\} → \{P\}

<table>
<thead>
<tr>
<th>Stock1</th>
<th>Model#</th>
<th>Serial#</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock2</td>
<td>Model#</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Stock3</td>
<td>Serial#</td>
<td>Year</td>
<td></td>
</tr>
<tr>
<td>Stock4</td>
<td>Name</td>
<td>Year</td>
<td>Price</td>
</tr>
</tbody>
</table>

Multivalued Dependencies

- Multivalued dependencies are a consequence of requiring 1NF

<table>
<thead>
<tr>
<th>Course</th>
<th>Teacher</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics</td>
<td>{Green, Brown, Black}</td>
<td>{Mechanics, Thermodynamics}</td>
</tr>
<tr>
<td>Math</td>
<td>{White}</td>
<td>{Algebra, Geometry}</td>
</tr>
</tbody>
</table>

- Semantics: every teacher who teaches a course uses all the texts for that course (independence of Teacher and Text)

- For two or more multivalued independent attributes, every value of one of the attributes must be repeated with every value of the other attribute to keep the relation consistent
Multivalued Dependencies (cont.)

<table>
<thead>
<tr>
<th>Course</th>
<th>Teacher</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Green</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Physics</td>
<td>Brown</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Brown</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Physics</td>
<td>Black</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Black</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Math</td>
<td>White</td>
<td>Algebra</td>
</tr>
<tr>
<td>Math</td>
<td>White</td>
<td>Geometry</td>
</tr>
</tbody>
</table>

- This relation is in BCNF since it is all-key
- A set of attributes $X$ *multidetermines* a set of attributes $Y$ if the value of $X$ determines a set of values for $Y$ (*independently* of any other attributes)
- A multivalued dependency (MVD) is written as $X \rightarrow\rightarrow Y$
- In the example, Course $\rightarrow\rightarrow$ Teacher and Course $\rightarrow\rightarrow$ Text

Intuition of MVDs

- $X \rightarrow\rightarrow Y$ is sometimes paraphrased as “$X$ multidetermines $Y$” or “a given $X$-value determines a set of $Y$-values”
- But this is really not precise enough, this is dangerously close to the paraphrase of a many-to-many relationship or a multivalued attribute
- $Z = R - (X \cup Y)$ is also involved
- If $X \rightarrow\rightarrow Y$ holds, then $X \rightarrow\rightarrow Z$ also holds
- A better, more intuitive notation is $X \rightarrow\rightarrow Y \mid Z$
- $X \rightarrow\rightarrow Y \mid Z$ implies that a value of $X$ determines a set of values of $Y$ *independently from* the values of $Z
• In the example, **Course →→ Teacher | Text**: each course is associated with a set of teachers and with a set of texts, and these sets are independent of each other

---

**Multivalued Dependencies: Formal Definition**

• Consider a relation schema \( R \), \( X \) and \( Y \) are subsets of attributes in \( R \), \( Z = R - (X \cup Y) \)

• \( X →→ Y \) holds in \( R \) if, whenever tuples \( t_1 \) and \( t_2 \) exist in an instance \( r(R) \) with \( t_1[X] = t_2[X] \), then tuples \( t_3 \) and \( t_4 \) also exist in \( r(R) \) such that

\[
\begin{array}{|c|c|c|}
\hline
X & Y & Z \\
\hline
X_1 & b_1 & c_1 \\
X_2 & b_1 & c_2 \\
X_3 & b_2 & c_1 \\
X_4 & b_2 & c_2 \\
\hline
\end{array}
\]

• A MVD \( X →→ Y \) is **trivial** if either \( Y \subseteq X \) or \( (X \cup Y) = R \)

○ always holds according to the MVD definition

• FDs are special cases of MVDs: If \( X → Y \) holds, then \( X →→ Z \) also holds
Inference Rules for FDs and MVDs

\begin{align*}
I1 & \quad Y \subseteq X \Rightarrow X \rightarrow Y \\
I2 & \quad X \rightarrow Y \Rightarrow XZ \rightarrow YZ \\
I3 & \quad X \rightarrow Y \text{ and } Y \rightarrow Z \Rightarrow X \rightarrow Z \\
I4 & \quad X \rightarrow Y \Rightarrow X \rightarrow Z \text{ where } Z = R - (X \cup Y) \\
I5 & \quad X \rightarrow\rightarrow Y \text{ and } Z \subseteq W \Rightarrow WX \rightarrow\rightarrow YZ \\
I6 & \quad X \rightarrow\rightarrow Y \text{ and } Y \rightarrow\rightarrow Z \Rightarrow X \rightarrow\rightarrow (Z - Y) \\
I7 & \quad X \rightarrow Y \Rightarrow X \rightarrow\rightarrow Z \\
I8 & \quad X \rightarrow\rightarrow Y \text{ and } W \rightarrow Z \text{ (for } Z \subseteq Y, W \cap Y = \emptyset \text{ and } W \cap Z = \emptyset) \Rightarrow X \rightarrow Z
\end{align*}

A Sound and Complete Set of Inference Rules for FDs and MVDs

To compute the closure of a set $F$ of functional and multivalued dependencies ($F^+$)

\begin{align*}
I1 & \quad (\text{Reflexivity for FDs}) \text{ If } Y \subseteq X, \text{ then } X \rightarrow Y \\
I2 & \quad (\text{Augmentation for FDs}) \text{ If } X \rightarrow Y, \text{ then } XZ \rightarrow YZ \\
I3 & \quad (\text{Transitivity for FDs}) \text{ If } X \rightarrow Y \text{ and } Y \rightarrow Z, \text{ then } X \rightarrow Z \\
I4 & \quad (\text{Complementation for MVDs}) \\
& \quad \text{If } X \rightarrow Y, \text{ then } X \rightarrow Z \text{ where } Z = R - (X \cup Y) \\
I5 & \quad (\text{Augmentation for MVDs}) \\
& \quad \text{If } X \rightarrow\rightarrow Y \text{ and } Z \subseteq W, \text{ then } WX \rightarrow\rightarrow YZ \\
I6 & \quad (\text{Transitivity for MVDs}) \\
& \quad \text{If } X \rightarrow\rightarrow Y \text{ and } Y \rightarrow\rightarrow Z, \text{ then } X \rightarrow\rightarrow (Z - Y) \\
I7 & \quad (\text{Replication}) \\
& \quad \text{If } X \rightarrow Y, \text{ then } X \rightarrow\rightarrow Z \\
I8 & \quad (\text{Coalescence for MVDs}) \\
& \quad \text{If } X \rightarrow\rightarrow Y \text{ and } W \rightarrow Z, \text{ for } Z \subseteq Y, W \cap Y = \emptyset \text{ and } W \cap Z = \emptyset, \text{ then } X \rightarrow Z
\end{align*}
Motivation for 4NF

- A relational schema with non-trivial MVDs is not a good design
- Update anomalies: for a new teacher of Physics, we must insert two tuples

<table>
<thead>
<tr>
<th>Course</th>
<th>Teacher</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Green</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Physics</td>
<td>Brown</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Brown</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Physics</td>
<td>Black</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Black</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Math</td>
<td>White</td>
<td>Algebra</td>
</tr>
<tr>
<td>Math</td>
<td>White</td>
<td>Geometry</td>
</tr>
</tbody>
</table>

- This relation represents two independent 1:N relationships

<table>
<thead>
<tr>
<th>Course</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics</td>
<td>Green</td>
</tr>
<tr>
<td>Physics</td>
<td>Brown</td>
</tr>
<tr>
<td>Physics</td>
<td>Black</td>
</tr>
<tr>
<td>Math</td>
<td>White</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics</td>
<td>Thermodynamics</td>
</tr>
<tr>
<td>Math</td>
<td>Algebra</td>
</tr>
<tr>
<td>Math</td>
<td>Geometry</td>
</tr>
</tbody>
</table>
Definition of 4NF

- A relation schema $R$ is in 4NF w.r.t. a set of FDs and MVDs $F$ if, for every nontrivial multivalued dependency $X \rightarrow\rightarrow Y$ in $F^+$, $X$ is a superkey of $R$
- Since every MVD is an FD, 4NF implies BCNF
- In other words, a relation is in 4NF if it is in BCNF and if every nontrivial MVD is also an FD
- If all dependencies in $F$ are FDs, the definition of 4NF reduces to that of BCNF
- Although many relations in BCNF but not in 4NF are all-key (they have no FD), this is not necessarily so

4NF: Example

<table>
<thead>
<tr>
<th>Course</th>
<th>Teacher</th>
<th>Text</th>
</tr>
</thead>
</table>

- Dependencies
  - $\diamond$ Course $\rightarrow\rightarrow$ Teacher | Text
  - $(\text{Teacher,Text}) \rightarrow$ Course
    (a teacher does not use the same text in more than one course)
- $\{\text{Teacher,Text}\}$ is a key
- Relation is in BCNF but not in 4NF
- Breaking it to produce 4NF relations does not preserve the dependency
Decomposition in 4NF

- Given a MVD $X \rightarrow \rightarrow Y$ that holds in a schema $R$, the decomposition into $R_1 = (X \cup Y)$ and $R_2 = (R - Y)$ has the nonadditive-join property.
- The converse also holds.
- Thus, a decomposition $D = \{R_1, R_2\}$ of $R$ has the nonadditive join property with respect to $F$ if and only if either:
  - $(R_1 \cap R_2) \rightarrow \rightarrow (R_1 - R_2)$ holds in $F^+$, or
  - $(R_1 \cap R_2) \rightarrow \rightarrow (R_2 - R_1)$ holds in $F^+$
- Actually, if one of these holds, so does the other.

Lossless Join Decomposition in 4NF

Algorithm for lossless join decomposition of $R$ into 4NF relations w.r.t. a set of FDs and MVDs:

1. Set $D := \{R\}$
2. While there is a relation schema $Q$ in $D$ that is not in 4NF do
   - begin
     - choose one $Q$ in $D$ that is not in 4NF;
     - find a nontrivial MVD $X \rightarrow \rightarrow Y$ in $Q$ that violates 4NF;
     - replace $Q$ in $D$ by two relations $(Q - Y)$ and $(X \cup Y)$
   - end;
- Does not necessarily preserve FDs
  - in the example above the FD $(\text{Teacher,Text}) \rightarrow \text{Course}$ is lost
A Sufficient Condition for Testing 4NF

Given a relation schema $R(U)$, a subset $C$ of $U$ is a cut if every key of $R$ has a non-empty intersection with $C$ and a nonempty intersection with $U - C$.

<table>
<thead>
<tr>
<th>Emp</th>
<th>Proj</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>P1</td>
<td>FL</td>
</tr>
<tr>
<td>Smith</td>
<td>P2</td>
<td>CA</td>
</tr>
<tr>
<td>Smith</td>
<td>P3</td>
<td>AZ</td>
</tr>
<tr>
<td>Walton</td>
<td>P1</td>
<td>CA</td>
</tr>
<tr>
<td>Walton</td>
<td>P2</td>
<td>AZ</td>
</tr>
</tbody>
</table>

• Assume that EmpProj has the keys $\{\text{Emp},\text{Proj}\}$ (an employee works for a project in only one location) and $\{\text{Emp},\text{Loc}\}$ (an employee works in one location for only one project).

• In EmpProj there is only one cut : $\{\text{Proj},\text{Loc}\}$

• Suppose that another key is added: $\{\text{Proj},\text{Loc}\}$ (given a project and a location, only one employee is attached to them).

• Now, relation EmpProj has no cut.

**Theorem:** If a relation schema is in BCNF and has no cut, then it is in 4NF.

**Corollary:** If a relation schema is in BCNF and has a simple (non composite) key, then it is in 4NF. (A relation with a simple key has no cut.)
Embedded Multivalued Dependencies

- FDs are preserved when adding or suppressing an attribute (provided it is not involved in the FD)
- On the contrary, some MVDs are expected to hold after projection but are not explicit as MVD before the projection

<table>
<thead>
<tr>
<th>Course</th>
<th>Stud</th>
<th>Preq</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS402</td>
<td>Jones</td>
<td>CS311</td>
<td>1988</td>
</tr>
<tr>
<td>CS402</td>
<td>Smith</td>
<td>CS401</td>
<td>1989</td>
</tr>
</tbody>
</table>

FD: \(\text{Stud}, \text{Preq} \rightarrow \text{Year}\)

- Course \(\rightarrow\rightarrow\) Stud does not hold: \((\text{CS402}, \text{Jones}, \text{CS401}, 1989) \notin \text{Regist}\)

Embedded Multivalued Dependencies (cont.)

<table>
<thead>
<tr>
<th>Regist1</th>
<th>Regist2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stud</td>
<td>Preq</td>
</tr>
</tbody>
</table>

- Course \(\rightarrow\rightarrow\) Stud (and Course \(\rightarrow\rightarrow\) Preq) hold in Regist2: every student enrolled in a course is required to have taken each prerequisite for the course
- Corresponding constraint in the original relation Regist is an **embedded multivalued dependency**
- It is written Course \(\rightarrow\rightarrow\) Stud|Preq, meaning that the dependency holds in the projection \(\pi_{\text{Course}, \text{Stud}, \text{Preq}}(\text{Regist})\)
- Regist2 is not in 4NF, and it should be decomposed into relations (Course, Stud) and (Course, Preq)
Join Dependencies

- There are relations where a nonadditive-join decomposition can only be realized with more than two relation schemas

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Part</th>
<th>Proj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Bolt</td>
<td>ProjX</td>
</tr>
<tr>
<td>Smith</td>
<td>Nut</td>
<td>ProjY</td>
</tr>
<tr>
<td>Adamsky</td>
<td>Bolt</td>
<td>ProjY</td>
</tr>
<tr>
<td>Smith</td>
<td>Bolt</td>
<td>ProjY</td>
</tr>
</tbody>
</table>

- If a supplier \( s \) supplies part \( p \), a project \( j \) uses part \( p \), and the supplier \( s \) supplies at least one part to project \( j \), then supplier \( s \) also supplies part \( p \) to project \( j \)

- Relation is “all key”, involves no nontrivial FDs or MVDs \( \Rightarrow \) is in 4NF

Join Dependencies, cont.

<table>
<thead>
<tr>
<th>Sup_Part</th>
<th>Part_Proj</th>
<th>Sup_Proj</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier</td>
<td>Part</td>
<td>Part</td>
</tr>
<tr>
<td>Smith</td>
<td>Bolt</td>
<td>Bolt</td>
</tr>
<tr>
<td>Smith</td>
<td>Nut</td>
<td>Nut</td>
</tr>
<tr>
<td>Adamsky</td>
<td>Bolt</td>
<td>Bolt</td>
</tr>
</tbody>
</table>

Join over Part

Join over Supplier and Proj

Original relation Supply

- Does not hold if last tuple is removed from Supply
Join Dependencies

- The constraint in the schema is equivalent to say
  \[
  \text{if } \langle s_1, p_1, j_2 \rangle, \langle s_2, p_1, j_1 \rangle, \langle s_1, p_2, j_1 \rangle \text{ appear in Supply}
  \quad \text{then } \langle s_1, p_1, j_1 \rangle \text{ also appears in Supply}
  \]

- Supply satisfies the join dependency
  \[
  JD(\{\text{Supplier,Part}\}, \{\text{Part,Proj}\}, \{\text{Supplier,Proj}\}\}, \text{ i.e.}
  \]
  \[
  \text{Supply} = \text{Supply[Supplier,Part]} \bowtie \text{Supply[Part,Proj]} \bowtie \text{Supply[Supplier,Proj]}
  \]

- A join dependency \( JD(R_1, R_2, \ldots, R_n) \) on a relation \( R \) specifies that every instance of \( R \) has a nonadditive-join decomposition into \( R_1, R_2, \ldots, R_n \).

- A MVD is a special case of a JD where \( n = 2 \).

- A \( JD(R_1, R_2, \ldots, R_n) \) on \( R \) is a trivial JD if some \( R_i = R \).

Join Dependencies and Update Anomalies

- Join dependencies induce update anomalies

  \[
  \begin{array}{|c|c|c|} \hline
  \text{Supply1} & \text{Supply2} \\
  \hline
  \text{Supplier} & \text{Part} & \text{Proj} & \text{Supplier} & \text{Part} & \text{Proj} \\
  \hline
  \text{Smith} & \text{Bolt} & \text{ProjX} & \text{Smith} & \text{Bolt} & \text{ProjX} \\
  \text{Smith} & \text{Nut} & \text{ProjY} & \text{Smith} & \text{Nut} & \text{ProjY} \\
  \text{Walton} & \text{Bolt} & \text{ProjY} & \text{Smith} & \text{Bolt} & \text{ProjY} \\
  \hline
  \end{array}
  \]

- In \text{Supply1}, inserting \( \text{(Walton,Bolt,ProjY)} \) implies insertion of \( \text{(Smith,Bolt,ProjY)} \) (yet converse is not true).

- Deleting \( \text{(Walton,Bolt,ProjY)} \) from \text{Supply2} has no side effects.

- If \( \text{(Smith,Bolt,ProjY)} \) is deleted from \text{Supply2}, then either the first or the third tuple must be deleted.
Fifth Normal Form (5NF)

- A relation schema $R$ is in 5NF w.r.t. a set $F$ or FDs, MVDs, and JDs if for every nontrivial $JD(R_1, R_2, \ldots, R_n)$, each $R_i$ is a superkey

$\downarrow$ 5NF normalization

- 5NF is also called PJNF (project-join normal form)
- Since a MVD is a special case of a JD, every relation in 5NF is also in 4NF
- Every relation can be non losslessly decomposed into 5NF relations
- If a relation is in 3NF and all its keys are simple, then it is in 5NF
- Discovering JDs in practice for large databases is difficult

- Many obvious join dependencies are based on keys
- Further decomposition ultimately leads to irreducible relations
Critique of Relational Normalization

(1) Normal forms are easier to understand and appreciate from a richer point of view on data modeling, namely, ER or OO

(2) Practical relevance of normal forms has been overemphasized

1) Normal forms are easier to understand and appreciate from a richer point of view on data modeling, namely, ER or OO

- Particularly striking for the “higher” normal forms (4NF, 5NF)
- MVDs are not stable when relation schemas are modified (leading to embedded dependencies) ⇒ complexity of MVDs is largely a relational problem
- A “complex” normal form like 4NF is better analyzed in ER terms than just with the multi-valued dependency ⇒ there are various interpretations for an MVD
  - multi-valued attribute
  - grouping of independent facts within one relation
  - integrity constraint in a genuine relationship
- ER-based design methodologies start with entities and relationships observed in the real world, and their systematic translation into relations produces 3NF (or higher) most of the time

2) Practical relevance of normal forms has been overemphasized

- Goal of normalization: produce simple relations representing in a natural way a portion of the “real world”
- But why start with a complex description in the first place to have to normalize it afterwards?
- Reason should be traced back to pre-relational technology: space was at a premium and attributes were grouped in large physical records to save space
Another reason for favoring large relation schemas was to minimize the number of joins in access programs.

Multi-level architectures now permit different schemas at the logical and physical levels.

Normalization was a nice relatively easy piece of relational theory (hard to resist for researchers!)

Normal forms are properties of relations in isolation; if and when (inter-relation) constraints are seriously taken care of by DBMSs, normalization as a criterion for the quality of a database schema will lose some of its emphasis.

Definition of the relational model was a “revolution” against “bad” practices that were prevailing in database management before the relational model.

Revolution went too far: 1NF is too restrictive for modeling complex data, 1NF was a simple radical idea.

Decomposition algorithms suppose that all functional dependencies have been specified.

Overlooking FDs may produce undesirable designs.

Not easy to solve in practice: it is better to allow grouping attributes on less formal grounds during conceptual modeling.

Algorithms that require a minimal cover depend on which minimal cover is picked (nondeterminism).

---

**Denormalization for Efficiency**

- Normalization considers only consistency, performance also matters.
- When the normalized database entails too many costly joins, then decide, **at the physical level** to store the joins rather than the projections.
- Still, it is more appropriate to start performance tuning with a well-designed database.
- Rationale for design in stages: first analysis, then design, then implementation and performance tuning.
Relational Database Design

- A complex process
- Made simpler by starting with a more expressive schema in a suitable model (ER, OO)
- Principles of translating an ER schema into a relational schema are simple ... 
- But faithfully translating an ER schema into a relational schema is an immense task, if done without loss of information
- In practice, even with sophisticated CASE tools, some information will be lost in the translation for the relational schema to remain manageable
- Traditional relational design neglects essential dependencies (inclusion and join dependencies)
- Normalization theory concerns both ER and relational schemas

Thesis

- Relational DB design has over-emphasized importance of normalization theory