Announcements

• Web quiz due tonight

• HW due Wednesday night
  – Watch the late days !!

• Today: Design theory (3.1-3.4)
Relational Schema Design

Conceptual Model:

Relational Model: plus FD’s

Normalization: Eliminates *anomalies*
Relational Schema Design

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

One person may have multiple phones, but lives in only one city.

Primary key is thus (SSN, PhoneNumber)

What is the problem with this schema?
Relational Schema Design

Anomalies:
- **Redundancy** = repeat data
- **Update anomalies** = what if Fred moves to “Bellevue”?
- **Deletion anomalies** = what if Joe deletes his phone number?

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>
Relation Decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:
- No more repeated data
- Easy to move Fred to “Bellevue” (how?)
- Easy to delete all Joe’s phone numbers (how?)
Relational Schema Design
(or Logical Design)

How do we do this systematically?

- Start with some relational schema
- Find out its functional dependencies (FDs)
- Use FDs to normalize the relational schema
Functional Dependencies (FDs)

**Definition**

If two tuples agree on the attributes

\[ A_1, A_2, \ldots, A_n \]

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
**Functional Dependencies (FDs)**

**Definition**  \( A_1, ..., A_m \rightarrow B_1, ..., B_n \) holds in \( R \) if:

\[
\forall t, t' \in R, \\
(t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \land ... \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>R</th>
<th>( A_1 )</th>
<th>...</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>...</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>...</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>...</td>
<td>( \text{&lt;cell&gt;} )</td>
</tr>
<tr>
<td>t'</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>...</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>( \text{&lt;cell&gt;} )</td>
<td>...</td>
<td>( \text{&lt;cell&gt;} )</td>
</tr>
</tbody>
</table>

if \( t, t' \) agree here then \( t, t' \) agree here
Example

An FD holds, or does not hold on an instance:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

EmpID → Name, Phone, Position
Position → Phone
but not Phone → Position
## Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

**Position → Phone**
Example

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

But not Phone → Position
Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

Do all the FDs hold on this instance?
Example

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
Terminology

- FD holds or does not hold on an instance

- If we can be sure that every instance of $R$ will be one in which a given FD is true, then we say that $R$ satisfies the FD

- If we say that $R$ satisfies an FD $F$, we are stating a constraint on $R$
An Interesting Observation

If all these FDs are true:

- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

Then this FD also holds:

- name, category $\rightarrow$ price

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.
Closure of a set of Attributes

Given a set of attributes \( A_1, \ldots, A_n \)

The closure, \( \{A_1, \ldots, A_n\}^+ \) = the set of attributes \( B \) s.t. \( A_1, \ldots, A_n \rightarrow B \)

Example:
1. name \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

Closures:
- \( \text{name}^+ = \{\text{name, color}\} \)
- \( \{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\} \)
- \( \text{color}^+ = \{\text{color}\} \)
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\} \]

Repeat until \( X \) doesn’t change do:

if \( B_1, \ldots, B_n \rightarrow C \) is a FD and
\( B_1, \ldots, B_n \) are all in \( X \)
then add \( C \) to \( X \).

Example:

1. name \( \rightarrow \) color
2. category \( \rightarrow \) department
3. color, category \( \rightarrow \) price

\( \{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\} \)

Hence: name, category \( \rightarrow \) color, department, price
Example

In class:

R(A,B,C,D,E,F)

A, B → C
A, D → E
B → D
A, F → B

Compute \{A, B\}^+ \quad X = \{A, B, \}

Compute \{A, F\}^+ \quad X = \{A, F, \}
Example

In class:

\[ R(A,B,C,D,E,F) \]

\[
\begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B \\
\end{array}
\]

Compute \( \{A,B\}^+ \quad X = \{A, B, C, D, E\} \)

Compute \( \{A, F\}^+ \quad X = \{A, F, \} \)
Example

In class:

\[ R(A, B, C, D, E, F) \]

\[
\begin{array}{l}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array}
\]

Compute \( \{A, B\}^+ \)  
\[ X = \{A, B, C, D, E\} \]

Compute \( \{A, F\}^+ \)  
\[ X = \{A, F, B, C, D, E\} \]
Example

In class:

\[ R(A,B,C,D,E,F) \]

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B \rightarrow C</td>
<td></td>
</tr>
<tr>
<td>A, D \rightarrow E</td>
<td></td>
</tr>
<tr>
<td>B \rightarrow D</td>
<td></td>
</tr>
<tr>
<td>A, F \rightarrow B</td>
<td></td>
</tr>
</tbody>
</table>

Compute \( \{A,B\}^+ \) \[ X = \{A, B, C, D, E\} \]

Compute \( \{A, F\}^+ \) \[ X = \{A, F, B, C, D, E\} \]

What is the key of \( R \)?
Practice at Home

Find all FD’s implied by:

- A, B → C
- A, D → B
- B → D
Practice at Home

Find all FD’s implied by:

A, B → C
A, D → B
B → D

Step 1: Compute $X^+$, for every $X$:

$A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$

$AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$,

$BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$

$ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute— why ?)

$BCD^+ = BCD$, $ABCD^+ = ABCD$
Practice at Home

Find all FD’s implied by:

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow B \\
B & \rightarrow D
\end{align*}
\]

Step 1: Compute \(X^+\), for every \(X\):

\[
\begin{align*}
A+ &= A, \quad B+ = BD, \quad C+ = C, \quad D+ = D \\
AB+ &= ABCD, \quad AC+ = AC, \quad AD+ = ABCD, \\
& \quad \quad \quad BC+ = BCD, \quad BD+ = BD, \quad CD+ = CD \\
ABC+ &= ABD+ = ACD+ = ABCD \text{ (no need to compute— why ?)} \\
BCD+ &= BCD, \quad ABCD+ = ABCD
\end{align*}
\]

Step 2: Enumerate all FD’s \(X \rightarrow Y\), s.t. \(Y \subseteq X^+\) and \(X \cap Y = \emptyset\):

\[
\begin{align*}
AB & \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B
\end{align*}
\]
Keys

• A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any other attribute $B$, we have $A_1, \ldots, A_n \rightarrow B$

• A **key** is a minimal superkey
  – A superkey and for which no subset is a superkey
Computing (Super)Keys

• For all sets \( X \), compute \( X^+ \)

• If \( X^+ = [\text{all attributes}] \), then \( X \) is a superkey

• Try only the minimal \( X \)'s to get the key
Example

Product(name, price, category, color)

name, category → price
category → color

What is the key?
Example

Product(name, price, category, color)

(name, category) + = { name, category, price, color }

Hence (name, category) is a key
Key or Keys?

Can we have more than one key?

Given $R(A,B,C)$ define FD’s s.t. there are two or more keys
Key or Keys?

Can we have more than one key?

Given \( R(A,B,C) \) define FD’s s.t. there are two or more keys:

- \( A \rightarrow B \)
- \( B \rightarrow C \)
- \( C \rightarrow A \)

- \( AB \rightarrow C \)
- \( BC \rightarrow A \)

- \( A \rightarrow BC \)
- \( B \rightarrow AC \)

what are the keys here?
Eliminating Anomalies

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

What is the key?

SSN → Name, City

Suggest a rule for decomposing the table to eliminate anomalies
Eliminating Anomalies

Main idea:

- \( X \rightarrow A \) is OK if \( X \) is a (super)key

- \( X \rightarrow A \) is not OK otherwise
  - Need to decompose the table, but how?
Boyce-Codd Normal Form

There are no “bad” FDs:

**Definition.** A relation R is in BCNF if:
Whenever \( X \rightarrow B \) is a non-trivial dependency, then X is a superkey.

Equivalently:

**Definition.** A relation R is in BCNF if:
\( \forall X \), either \( X^+ = X \) or \( X^+ = [\text{all attributes}] \).
BCNF Decomposition Algorithm

Normalize(R)
  find X s.t.: X ≠ X⁺ ≠ [all attributes]
  if (not found) then "R is in BCNF"
  let Y = X⁺ - X; Z = [all attributes] - X⁺
  decompose R into R1(X ∪ Y) and R2(X ∪ Z)
  Normalize(R1); Normalize(R2);
Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

The only key is: \{SSN, PhoneNumber\}
Hence **SSN → Name, City** is a “bad” dependency

In other words:
**SSN+ = SSN, Name, City** and is neither SSN nor All Attributes
Example BCNF Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN → Name, City

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-1234</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-6543</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-2121</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-1234</td>
</tr>
</tbody>
</table>

Let’s check anomalies:
- Redundancy ?
- Update ?
- Delete ?
Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Find X s.t.: $X \neq X^+ \neq \{\text{all attributes}\}$
Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN → name, age
age → hairColor

Find X s.t.: X ≠ X⁺ ≠ [all attributes]

Iteration 1: Person: SSN⁺ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
                Phone(SSN, phoneNumber)

CSE 414 - Spring 2015
Example BCNF Decomposition

\textbf{Person(name, SSN, age, hairColor, phoneNumber)}

\begin{align*}
\text{SSN} & \rightarrow \text{name, age} \\
\text{age} & \rightarrow \text{hairColor}
\end{align*}

\textbf{Iteration 1: Person: } \text{SSN}^+ = \text{SSN, name, age, hairColor}

\text{Decompose into: } \text{P(SSN, name, age, hairColor)}
\quad \text{Phone(SSN, phoneNumber)}

\textbf{Iteration 2: } \text{P: age}^+ = \text{age, hairColor}

\text{Decompose: } \text{People(SSN, name, age)}
\quad \text{Hair(age, hairColor)}
\quad \text{Phone(SSN, phoneNumber)}

Find X s.t.: X \neq X^+ \neq [\text{all attributes}]

What are the keys?
Example BCNF Decomposition

Person(name, SSN, age, hairColor, phoneNumber)

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Find X s.t.: $X \neq X^+$ $\neq$ [all attributes]

Iteration 1: **Person:** SSN+ = SSN, name, age, hairColor
Decompose into: $P(\text{SSN, name, age, hairColor})$
$\quad$Phone(\text{SSN, phoneNumber})

Iteration 2: **P:** age+ = age, hairColor
Decompose: People(\text{SSN, name, age})
$\quad$Hair(\text{age, hairColor})
$\quad$Phone(\text{SSN, phoneNumber})
Example: BCNF

\[ R(A,B,C,D) \]

\[ A \rightarrow B \]
\[ B \rightarrow C \]
R(A,B,C,D)

Example: BCNF

Recall: find X s.t. $X \subsetneq X^+ \subsetneq [\text{all-attrs}]$

\[
\begin{align*}
A \rightarrow B \\
B \rightarrow C
\end{align*}
\]
Example: BCNF

\[ R(A, B, C, D) \]

\[ A \rightarrow B \]
\[ B \rightarrow C \]

\[ A^+ = ABC \neq ABCD \]
Example: BCNF

R(A,B,C,D)

A → B
B → C

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)

R₂(A,D)
R(A,B,C,D)

Example: BCNF

A → B
B → C

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)
B⁺ = BC ≠ ABC

R₂(A,D)
Example: BCNF

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)
B⁺ = BC ≠ ABC

R₁₁(B,C)
R₁₂(A,B)

R₂(A,D)

What are the keys?

What happens if in R we first pick B⁺? Or AB⁺?
Decompositions in General

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

\[ S_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ S_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
## Lossless Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

### Sub-tables

#### Names and Prices

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
</tbody>
</table>

#### Names and Categories

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Lossy Decomposition

What is lossy here?

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Lossy Decomposition

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Decomposition in General

Let:

\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]
\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]

The decomposition is called \textit{lossless} if \( R = S_1 \bowtie S_2 \)

Fact: If \( A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m \) then the decomposition is lossless.

It follows that every BCNF decomposition is lossless.
Schema Refinements
= Normal Forms

• 1st Normal Form = all tables are flat
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = no bad FDs
• 3rd Normal Form = see book
  – BCNF is lossless but after joining the relation may not satisfy all original FDs (see book 3.4.4)
  – 3NF fixes that (is lossless and dependency-preserving), but some tables might not be in BCNF
    – i.e., they may have redundancy anomalies
Goal: Find ALL Functional Dependencies

- Anomalies occur when certain “bad” FDs hold
- We know some of the FDs
- Need to find all FDs
- Then look for the bad ones
Armstrong’s Rules (1/3)

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Is equivalent to

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]

Splitting rule and Combing rule

CSE 414 - Spring 2015

55
Armstrong’s Rules (2/3)

A_1, A_2, ..., A_n \rightarrow A_i

Trivial Rule

where i = 1, 2, ..., n

Why?
Armstrong’s Rules (3/3)

Transitive Rule

If \( A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \)

and \( B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p \)

then \( A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \)

Why?
Armstrong’s Rules (3/3)

Illustration

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$\ldots$</th>
<th>$A_m$</th>
<th>$B_1$</th>
<th>$\ldots$</th>
<th>$B_m$</th>
<th>$C_1$</th>
<th>$\ldots$</th>
<th>$C_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (continued)

Start from the following FDs:

1. name \(\rightarrow\) color
2. category \(\rightarrow\) department
3. color, category \(\rightarrow\) price

Infer the following FDs:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category (\rightarrow) name</td>
<td></td>
</tr>
<tr>
<td>5. name, category (\rightarrow) color</td>
<td></td>
</tr>
<tr>
<td>6. name, category (\rightarrow) category</td>
<td></td>
</tr>
<tr>
<td>7. name, category (\rightarrow) color, category</td>
<td></td>
</tr>
<tr>
<td>8. name, category (\rightarrow) price</td>
<td></td>
</tr>
</tbody>
</table>
Example (continued)

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category → name</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>5. name, category → color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category → category</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>7. name, category → color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category → price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>

THIS IS TOO HARD! Let’s see an easier way.