Introduction to Database Systems
CSE 444

Lectures 6-7: Database Design
Outline

• Design theory: 3.1-3.4
  – [Old edition: 3.4-3.6]
Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = will study
- 3rd Normal Form = see book
First Normal Form (1NF)

- A database schema is in First Normal Form if all tables are flat

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Student

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Takes

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Math</td>
</tr>
<tr>
<td>Carol</td>
<td>Math</td>
</tr>
<tr>
<td>Alice</td>
<td>DB</td>
</tr>
<tr>
<td>Bob</td>
<td>DB</td>
</tr>
<tr>
<td>Alice</td>
<td>OS</td>
</tr>
<tr>
<td>Carol</td>
<td>OS</td>
</tr>
</tbody>
</table>

May need to add keys
Relational Schema Design

Conceptual Model:

Relational Model: plus FD’s

Normalization: Eliminates *anomalies*
Data Anomalies

When a database is poorly designed we get anomalies:

**Redundancy**: data is repeated

**Updated anomalies**: need to change in several places

**Delete anomalies**: may lose data when we don’t want
Relational Schema Design

Recall set attributes (persons with several phones):

<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)

The above is in 1NF, but was is the problem with this schema?
Relational Schema Design

Recall set attributes (persons with several phones):

<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:

- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Bellevue”?
- Deletion anomalies = what if Joe deletes his phone number? (what if Joe had only one phone #)
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)

Main idea:
• Start with some relational schema
• Find out its *functional dependencies*
  – They come from the application domain knowledge!
• Use them to design a better relational schema
Functional Dependencies

• A form of constraint
  – Hence, part of the schema
• Finding them is part of the database design
• Use them to normalize the relations
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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
When Does an FD Hold

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>( n_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t'</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 $\rightarrow$ Name, Phone, Position
Position $\rightarrow$ Phone
but not Phone $\rightarrow$ 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

FD’s are constraints:
• On some instances they hold
• On others they don’t

<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>

Does this instance satisfy all the FDs?
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?
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

Why ??
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

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

\[ \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]
### Armstrong’s Rules (2/3)

The Trivial Rule states:

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

where \( i = 1, 2, \ldots, n \)

**Why?**

<table>
<thead>
<tr>
<th></th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Magda Balazinska - CSE 444, Fall 2010
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></th>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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>
<td></td>
</tr>
<tr>
<td></td>
<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)

Answers:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. name → color</td>
<td></td>
</tr>
<tr>
<td>2. category → department</td>
<td></td>
</tr>
<tr>
<td>3. color, category → price</td>
<td></td>
</tr>
<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.
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:**
- name $\rightarrow$ color
- category $\rightarrow$ department
- color, category $\rightarrow$ price

**Closures:**
- $name^+ = \{name, color\}$
- $\{name, category\}^+ = \{name, category, color, department, price\}$
- $color^+ = \{color\}$
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 \( X \).

\{name, category\}^+ =

\{ name, category, color, department, price \}

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

In class:

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

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

Compute \{A,B\}⁺ \quad X = \{A, B, \}

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

In class:

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

<table>
<thead>
<tr>
<th>Relation</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>C</td>
</tr>
<tr>
<td>A, D</td>
<td>E</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>A, F</td>
<td>B</td>
</tr>
</tbody>
</table>

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

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

In class:

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

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

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

Compute \( \{A, F\}^+ \) \( X = \{A, F, B, C, D, E\} \)
Why Do We Need Closure

• With closure we can find all FD’s easily

• To check if $X \rightarrow A$
  – Compute $X^+$
  – Check if $A \in X^+$
Using Closure to Infer ALL FDs

Example:

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

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

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

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

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

- Enrollment(student, major, course, room, time)
  
  student $\rightarrow$ major
  major, course $\rightarrow$ room
  course $\rightarrow$ time

What else can we infer? [in class, or at home]
Keys

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

- A **key** is a minimal superkey
  - i.e. set of attributes which is a superkey and for which no subset is a superkey
Computing (Super)Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a superkey
- List only the minimal $X$’s to get the keys
Example

Product(name, price, category, color)

name, category $\rightarrow$ price

category $\rightarrow$ color

What is the key?
Example

Product(name, price, category, color)

(name, category) → price
category → color

What is the key?

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

Hence (name, category) is a key
Examples of Keys

Enrollment(student, address, course, room, time)

- student $\rightarrow$ address
- room, time $\rightarrow$ course
- student, course $\rightarrow$ room, time

(find keys at home)
Eliminating Anomalies

Main idea:

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

• $X \rightarrow A$ is not OK otherwise
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>

SSN \rightarrow \text{Name, City}

What is the key? 
\{SSN, PhoneNumber\} 

Hence SSN \rightarrow \text{Name, City} is a “bad” dependency
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

\begin{align*}
AB & \rightarrow C \\
BC & \rightarrow A \\
A & \rightarrow BC \\
B & \rightarrow AC
\end{align*}

what are the keys here?

Can you design FDs such that there are three keys?
Boyce-Codd Normal Form

A simple condition for removing anomalies from relations:

A relation \( R \) is in BCNF if:

If \( A_1, ..., A_n \rightarrow B \) is a non-trivial dependency in \( R \),
then \( \{A_1, ..., A_n\} \) is a superkey for \( R \)

In other words: there are no “bad” FDs

Equivalently:
for all \( X \), either \( (X^+ = X) \) or \( (X^+ = \text{all attributes}) \)
BCNF Decomposition Algorithm

repeat
  choose $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ that violates BCNF
  split $R$ into $R_1(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $R_2(A_1, \ldots, A_m, \text{[others]})$
  continue with both $R_1$ and $R_2$
until no more violations

Is there a 2-attribute relation that is not in BCNF?

In practice, we have a better algorithm (coming up)
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>

SSN $\rightarrow$ Name, City

What is the key?

\{SSN, PhoneNumber\} use SSN $\rightarrow$ Name, City to split
Example

<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>

Let’s check anomalies:
- Redundancy ?
- Update ?
- Delete ?

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>
Example Decomposition

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

FD1: SSN \rightarrow name, age
FD2: age \rightarrow hairColor

Decompose in BCNF (in class):
Example Decomposition

**Person** (name, SSN, age, hairColor, phoneNumber)
- FD1: SSN $\rightarrow$ name, age
- FD2: age $\rightarrow$ hairColor

Decompose in BCNF (in class): What is the key?
- $\{\text{SSN, phoneNumber}\}$

But how to decompose?
- **Person**(SSN, name, age)
- **Phone**(SSN, hairColor, phoneNumber)
- Or
- **Person**(SSN, name, age, hairColor)
- **Phone**(SSN, phoneNumber)
- Or ....
BCNF Decomposition Algorithm

BCNF\_Decompose(R)

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

\textbf{if} (not found) \textbf{then} “R is in BCNF”

\textbf{let} \(Y = X^+ - X\)
\textbf{let} \(Z = [\text{all attributes}] - X^+\)

decompose R into \(R_1(X \cup Y)\) and \(R_2(X \cup Z)\)
continue to decompose recursively \(R_1\) and \(R_2\)
Example BCNF Decomposition

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

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

Iteration 2: P
age+ = age, hairColor
Decompose: People(SSN, name, age)
           Hair(age, hairColor)
           Phone(SSN, phoneNumber)

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

What are the keys?
Example

\[
R(A,B,C,D) \\
A^+ = ABC \neq ABCD
\]

\[
R_1(A,B,C) \\
B^+ = BC \neq ABC
\]

\[
R_{11}(B,C) \\
R_{12}(A,B)
\]

\[
R_2(A,D)
\]

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

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

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ R_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ R_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
Theory of Decomposition

• Sometimes it is correct:

<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>

Lossless decomposition
Incorrect Decomposition

- Sometimes it is not:

<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>

What's incorrect??

Lossy decomposition
Decompositions in General

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

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

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

Then the decomposition is lossless

Note: don’t need \( A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p \)

BCNF decomposition is always lossless. WHY?
Optional

• The following four slides are optional
• The content will not be on any exam

• But please take a look because they motivate the need for 3NF

• It’s good to know at least why 3NF exists
General Decomposition Goals

1. Elimination of anomalies

2. Recoverability of information
   - Can we get the original relation back?

3. Preservation of dependencies
   - Want to enforce FDs without performing joins

Sometimes cannot decomposed into BCNF without losing ability to check some FDs
BCNF and Dependencies

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
</table>

FD’s:  \( \text{Unit} \to \text{Company}; \quad \text{Company, Product} \to \text{Unit} \)

So, there is a BCNF violation, and we decompose.
BCNF and Dependencies

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
</table>

FD’s: Unit → Company; Company, Product → Unit
So, there is a BCNF violation, and we decompose.

- Unit → Company
- No FDs

In BCNF we lose the FD: Company, Product → Unit
3NF Motivation

A relation R is in 3rd normal form if:

Whenever there is a nontrivial dep. $A_1, A_2, ..., A_n \rightarrow B$ for R, then $\{A_1, A_2, ..., A_n\}$ is a super-key for R, or B is part of a key.

Tradeoffs

BCNF = no anomalies, but may lose some FDs
3NF = keeps all FDs, but may have some anomalies