Outline

• Chapter 2: Database design

• Chapter 19: Normal forms

Note: slides for Lecture 1 have been updated. Please reprint.
Database Design

• Requirements analysis
  – Discussions with user groups
• Conceptual database design
  – E/R model
• Logical Database design
  – Database normalization
Entity / Relationship Diagrams

- Entities:
- Attributes:
- Relationships:
Keys in E/R Diagrams

• Every entity set must have a key
What is a Relation?

- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of $A \times B$

- $A=\{1,2,3\}$, $B=\{a,b,c,d\}$, $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$ $A=$
  $R = \{(1,a), (1,c), (3,b)\}$

- makes is a subset of Product × Company:
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many

Note: “many-one” actually means “many-[zero-or-one]”
Notation in Class v.s. the Book

In class:

Product → makes → Company

In the book:

Product → makes → Company
What does this say?
Multi-way Relationships

How do we model a purchase relationship between buyers, products and stores?
Q: what does the arrow mean?

A: a given person buys a given product from at most one store
Q: what does the arrow mean?

A: a given person buys a given product from at most one store AND every store sells to every person at most one product
Q: How do we say that every person shops at at most one store?

A: cannot. This is the best approximation. (Why only approximation?)
Reification: Multi-way to Binary

date

ProductOf

Product

StoreOf

Store

BuyerOf

Person
3. Design Principles

What’s wrong?

Product \rightarrow Purchase \leftrightharpoons Person

Country \rightarrow President \rightarrow Person

Moral: be faithful!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

• Entity set $\rightarrow$ relation
• Relationship $\rightarrow$ relation
Entity Set to Relation

Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadgets</td>
<td>$19.99</td>
</tr>
</tbody>
</table>
Relationships to Relations

<table>
<thead>
<tr>
<th>ProdName</th>
<th>ProdCategory</th>
<th>CompanyName</th>
<th>StartYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadgets</td>
<td>gizmoWorks</td>
<td>1963</td>
</tr>
</tbody>
</table>

(watch out for attribute name conflicts)
No need for **Makes**. Modify **Product**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
<th>CompanyName</th>
<th>StartYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadgets</td>
<td>$19.99</td>
<td>gizmoWorks</td>
<td>1963</td>
</tr>
</tbody>
</table>
Multi-way Relationships to Relations

Product
  name
  price

Purchase
  Purchase(productName, ssn, storeName)

Person
  name
  ssn

Store
  name
  address
Modeling Subclasses

Some objects in a class may be special
- define a new class
- better: define a subclass

Products

Software products

Educational products

So --- we define subclasses in E/R
Subclasses

Product

- name
- category
- price

isa

Software Product
- platforms

isa

Educational Product
- Age Group
Understanding Subclasses

• Think in terms of records:
  – Product
    
    | field1 |
    | field2 |
  
  – SoftwareProduct
    
    | field1 |
    | field2 |
    | field3 |
  
  – EducationalProduct
    
    | field1 |
    | field2 |
    | field3 |
    | field4 |
    | field5 |
Subclasses to Relations

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>todler</td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
</tr>
</tbody>
</table>
Difference between OO and E/R inheritance

• OO: classes are disjoint (same for Java, C++)
Difference between OO and E/R inheritance

- E/R: entity sets overlap
No need for multiple inheritance in E/R

We have three entity sets, but four different kinds of objects.
Modeling UnionTypes With Subclasses

Say: each piece of furniture is owned either by a person, or by a company
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person, or by a company

Solution 1. Acceptable, imperfect (What’s wrong?)
Modeling Union Types with Subclasses

Solution 2: better, more laborious

Use THIS solution in homework 2!
Constraints in E/R Diagrams

• Key constraints

• Single value constraints

• Referential integrity constraints

• Cardinality constraints
Keys in E/R Diagrams

In E/R diagrams each entity set must have exactly one key (consisting of one or more attributes)
Single Value Constraints

v. s.

makes
Referential Integrity Constraints

Each product made by at most one company. Some products made by no company.

Each product made by exactly one company.
Cardinality Constraints

Product \lt 100 \text{ makes } \rightarrow \text{ Company}

What does this mean?
Weak Entity Sets

Weak entity set = entity where part of the key comes from another

Convert to a relational schema (in class)
What Are the Keys of R?
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>Name</th>
<th>GPA</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td>Math, DB, OS</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>DB, OS</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math, OS</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>

**Course**

<table>
<thead>
<tr>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<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

**Anomalies:**

- **Redundancy** = repeat data
- **Update anomalies** = Fred moves to “Bellevue”
- **Deletion anomalies** = Joe deletes his phone number: what is his city?
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 number (how ?)
Relational Schema Design (or Logical Design)

Main idea:

• Start with some relational schema
• Find out its *functional dependencies*
• 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
• Also used in normalizing the relations
Functional Dependencies

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:

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

<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
Examples

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? (At home…)

**name → color
category → department
color, category → price**
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

\[ A_1, A_2, \ldots, A_n \rightarrow B_1 \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_2 \]
\[ \ldots \ldots \]
\[ A_1, A_2, \ldots, A_n \rightarrow B_m \]
Armstrong’s Rules (2/3)

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

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>

Trivial Rule
Armstrong’s Rules (3/3)

Transitive Closure 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?
<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>...</th>
<th>$A_m$</th>
<th></th>
<th>$B_1$</th>
<th>...</th>
<th>$B_m$</th>
<th></th>
<th>$C_1$</th>
<th>...</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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example (continued)

Start from the following FDs:

- name \(\rightarrow\) color
- category \(\rightarrow\) department
- 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>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>

1. name → color
2. category → department
3. color, category → price

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**, $\{A_1, \ldots, A_n\}^+ = \text{the set of attributes } B$

s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

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

Closures:

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

\[ \{\text{name, category}\}^+ = \{ \text{name, category, color, department, price} \} \]

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

In class:

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

| A, B | → | C |
| A, D | → | E |
| B   | → | D |
| A, F | → | B |

Compute \(\{A, B\}^+\)  
\[X = \{A, B, \ldots\}\]

Compute \(\{A, F\}^+\)  
\[X = \{A, F, \ldots\}\]
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 \(X\):

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

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

\[
\begin{align*}
AB & \rightarrow CD, \quad AD \rightarrow BC, \quad BC \rightarrow D
\end{align*}
\]
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 key
- List only the minimal $X$’s
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) + = 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>

What the key? 
{SSN, PhoneNumber} 

Hence SSN $\rightarrow$ 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
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:

$\forall X$, 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 the key?  
$\{\text{SSN, PhoneNumber}\}$  
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>

SSN $\Rightarrow$ Name, City

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

BCNF_Decompose(R)

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

if (not found) then “R is in BCNF”

let Y = $X^+ - X$
let Z = [all attributes] - $X^+$

decompose R into R1(X $\cup$ Y) and R2(X $\cup$ Z)
continue to decompose recursively R1 and R2
Example BCNF Decomposition

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

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

**Example BCNF Decomposition**

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

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

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

**What are the keys?**
Example

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

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

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

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

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

\[ R_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \]
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>

Lossy decomposition
Decompositions in General

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

If \( A_1, ..., A_n \rightarrow B_1, ..., B_m \)  
Then the decomposition is lossless

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

BCNF decomposition is always lossless. WHY?