Introduction to Database Systems
CSE 444

Lectures 8 & 9
Database Design

April 16 & 18, 2008

Outline

• The relational data model: 3.1
• Functional dependencies: 3.4

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</td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>Math, DB</td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math, OS</td>
</tr>
</tbody>
</table>
Relational Schema Design

Conceptual Model:

Person

buys

Product

Relational Model: plus FD’s

name

Conceptual Model:

Relational Model:

name

Normalization: Eliminates anomalies

Data Anomalies

When a database is poorly designed we get anomalies:

Redundancy: data is repeated

Update anomalies: need to change in several places

Delete anomalies: may lose data when we don’t want

Anomalies:

• Redundancy = repeated 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:

Anomalies are gone:

• No more repeated data
• Easy to move Fred to “Bellevue” (how?)
• Easy to delete all Joe’s phone numbers (how?)

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
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: \( A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \) holds in \( R \) if:
\[ \forall t, t' \in R, (t.A_1=t'.A_1 \wedge \ldots \wedge t.A_m=t'.A_m \Rightarrow t.B_1=t'.B_1 \wedge \ldots \wedge t.B_n=t'.B_n) \]
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 → 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

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-sup.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?

An Interesting Observation

If all these FDs are true:

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>color</td>
<td>category</td>
<td>department</td>
<td>price</td>
</tr>
</tbody>
</table>

Then this FD also holds:

<table>
<thead>
<tr>
<th>name, category</th>
<th>price</th>
</tr>
</thead>
</table>

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)

- Splitting rule
- Combing rule

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

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 (1/3)

**Trivial Rule**

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

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

Why?

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Armstrong’s Rules (1/3)

**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?

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>
</tbody>
</table>
Example (continued)

Answers:

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

<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.

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The closure, $\{A_1, \ldots, A_n\}^+ = \{B \mid A_1, \ldots, A_n \rightarrow B\}$

Example: 

$name \rightarrow color$

category → department

color, category → 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 $X$.

Example:

$name \rightarrow color$

category → department

color, category → price

{name, category}$^+ = \{name, category, color, department, price\}$

Hence: name, category → color, department, price

Example

In class:

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

$A, B \rightarrow C$

$A, D \rightarrow E$

$B \rightarrow D$

$A, F \rightarrow B$

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

Compute $\{A, F\}^+$ $X = \{A, F, \}$
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:

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
</tbody>
</table>
```

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 ?)
- $BCD^+ = BCD$
- $ABCD^+ = ABCD$

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

- $AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$

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)

\[
\begin{align*}
\text{name, category} & \rightarrow \text{price} \\
\text{category} & \rightarrow \text{color}
\end{align*}
\]

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)

\[
\begin{align*}
\text{student} & \rightarrow \text{address} \\
\text{room, time} & \rightarrow \text{course} \\
\text{student, course} & \rightarrow \text{room, time}
\end{align*}
\]

(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 $\rightarrow$ Name, City

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

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

$AB \rightarrow C$ or $A \rightarrow BC$

BC $\rightarrow A$ or $B \rightarrow AC$

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

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?

\{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?
Example Decomposition

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

Decompose in BCNF (in class):

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

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)

Example BCNF Decomposition

What are the keys?

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

What happens if in R we first pick B^+? Or AB^+? Write the keys.
### Decompositions in General

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

- \( R_1(A_1, ..., A_n, B_1, ..., B_m) \)
- \( R_2(A_1, ..., A_n, C_1, ..., C_p) \)

\( R_1 \) = projection of \( R \) on \( A_1, ..., A_n, B_1, ..., B_m \)
\( R_2 \) = projection of \( R \) on \( A_1, ..., A_n, C_1, ..., C_p \)

### Theory of Decomposition

- Sometimes it is correct:
  - Name | Price | Category
  - Gizmo | 19.99 | Gadget
  - OneClick | 24.99 | Camera
  - Gizmo | 19.99 | Camera

- Lossless decomposition

### Incorrect Decomposition

- Sometimes it is not:
  - Name | Price | Category
  - Gizmo | 19.99 | Gadget
  - OneClick | 24.99 | Camera
  - Gizmo | 19.99 | Camera

### Decompositions in General

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

- \( R_1(A_1, ..., A_n, B_1, ..., B_m) \)
- \( R_2(A_1, ..., A_n, C_1, ..., C_p) \)

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?