Introduction to Management  
CSE 344  

Lectures 16: Database Design

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?

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>

Name | SSN    | City   |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>

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

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

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:

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

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>

EmplID \( \rightarrow \) Name, Phone, Position
Position \( \rightarrow \) Phone
but not Phone \( \rightarrow \) Position

Example

Position \( \rightarrow \) Phone

<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>

Example

But not Phone \( \rightarrow \) 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?

An Interesting Observation

If all these FDs are true:

Then this FD also holds:

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)

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

Splitting rule and Combing rule

Armstrong’s Rules (2/3)

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

Trivial Rule

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

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?

Example (continued)

Start from the following FDs:

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

Infer the following FDs:

1. name \(\rightarrow\) category
2. name, category \(\rightarrow\) name
3. name, category \(\rightarrow\) color
4. name, category \(\rightarrow\) category
5. name, category \(\rightarrow\) color, category
6. name, category \(\rightarrow\) price
7. name, category, color, department \(\rightarrow\) price
8. name, category, color, department, price

Inferred FD | Which Rule did we apply?
--- | ---
4. name, category \(\rightarrow\) name | Trivial rule
5. name, category \(\rightarrow\) color | Transitivity on 4, 1
6. name, category \(\rightarrow\) category | Trivial rule
7. name, category \(\rightarrow\) color, category | Split/combine on 5, 6
8. name, category \(\rightarrow\) price | Transitivity on 3, 7

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

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

Example:

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

Hence:

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

In class:

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

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

Compute \( (A,B)^* \) \( X = \{ A, B, \} \)

Compute \( (A, F)^* \) \( X = \{ A, F, \} \)

Example

In class:

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

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

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

Compute \( (A, F)^* \) \( X = \{ A, F, \} \)

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:

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

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

\[
\begin{align*}
A^+ &= A \\
B^+ &= BD \\
C^+ &= C \\
D^+ &= D \\
AB^+ &= ABCD \\
AC^+ &= AC \\
AD^+ &= ABCD \\
BC^+ &= BCD \\
BD^+ &= BD \\
CD^+ &= CD \\
ABC^+ &= ACD^+ = ABCD
\end{align*}
\]

Step 2: Enumerate all FD's \( X \rightarrow Y \), s.t. \( Y \subseteq X^+ \) and \( X \setminus Y \neq \emptyset \):

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

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
  - 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 $X^+$ = 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

What is the key?

(name, category) + $\rightarrow$ { name, category, price, color }

Hence (name, category) is a key

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>206-555-5643</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} $\rightarrow$ Name, City

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

<table>
<thead>
<tr>
<th>$AB \rightarrow C$</th>
<th>$A \rightarrow BC$</th>
</tr>
</thead>
</table>

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 $A_1, \ldots, A_r \rightarrow B$ is a non-trivial dependency in $R$,
  then $(A_1, \ldots, A_r)$ 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

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>

Example Decomposition

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

FD1: SSN $\rightarrow$ name, age

FD2: age $\rightarrow$ hairColor

Decompose in BCNF (in class):

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

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

FD1: SSN → name, age
FD2: age → hairColor

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

But how to decompose?
Person(SSN, name, age)
Phone(SSN, hairColor, phoneNumber)
Or:
Person(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)
Or ....

Example BCNF Decomposition

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

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)

What are the keys?

BCNF Decomposition Algorithm

BCNF_Decompose(R)

find X s.t.: X ≠ X* ≠ [all attributes]

if (not found) then "R is in BCNF"

let Y = X* - X
let Z = [all attributes] - X*

decompose R into R1(X ∪ Y) and R2(X ∪ Z)

continue to decompose recursively R1 and R2

Schema Refinements = Normal Forms

• 1st Normal Form = all tables are flat
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = today
• 3rd Normal Form = see book