Database Systems
CSE 414

Lectures 23-24: Design Theory
(Ch. 3.1, 3.3-4)
Announcements

• HW6 will be due next Monday 11pm

• HW8 will be posted next Tuesday and due on Dec. 8, 11pm
Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema

CSE 414 - Fall 2017
What makes good schemas?
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

<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?

These can cause bugs! Worry most about later two.
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:

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

$$A_1 \ldots A_n \text{ determines } B_1 \ldots B_m$$
Functional Dependencies (FDs)

**Definition**  
FD $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ holds in $R$ if:

- for every pair of tuples $t, t' \in R$,
- $t.A_1 = t'.A_1$ and ... $t.A_m = t'.A_m$ $\rightarrow$ $t.B_1 = t'.B_1$ and ... $t.B_n = t'.B_n$

Never have equal $A$s but different $B$s!
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 \(\rightarrow\) 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>

Do all the FDs hold on this instance?

name → color
category → department
color, category → price
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>49</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** (part of schema)
An Interesting Observation

If all these FDs are true:

- name → color
- category → department
- color, category → price

Then this FD also holds:

- name, category → 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. \( \text{name} \rightarrow \text{color} \)
2. \( \text{category} \rightarrow \text{department} \)
3. \( \text{color, category} \rightarrow \text{price} \)

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

In class:

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

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

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

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

In class:

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

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

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

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

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

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

Find all FD’s implied by:

- A, B \rightarrow C
- A, D \rightarrow B
- B \rightarrow D
Practice at Home

Find all FD’s implied by:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>C</td>
<td>A, D</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>D</td>
<td></td>
<td></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 $X \cap Y = \emptyset$:

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

{\text{name, category} \rightarrow \text{price}, \text{category} \rightarrow \text{color}}

What is the key?

{\text{name, category}} + = \{\text{name, category, price, color}\}

Hence \{\text{name, category}\} is a (super)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

A → B
B → C
C → A

or

AB → C
BC → A

or

A → BC
B → AC

what are the keys here?
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
Boyce-Codd Normal Form

Dr. Raymond F. Boyce
Boyce-Codd Normal Form

There are no "bad" FDs:

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

Equivalently:

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

Normalize(R)

find X s.t.: X ≠ X⁺ and 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>

SSN → Name, City

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

**Example BCNF Decomposition**

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

- SSN $\rightarrow$ name, race
- race $\rightarrow$ hairColor
Example BCNF Decomposition

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

SSN → name, race
race → hairColor

Iteration 1: Person:  SSN⁺ = SSN, name, race, hairColor
Decompose into: P(SSN, name, race, hairColor)
Phone(SSN, phoneNumber)
Example BCNF Decomposition

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

\[
\text{SSN} \rightarrow \text{name, race} \\
\text{race} \rightarrow \text{hairColor}
\]

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

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

Find \( X \) s.t.: \( X \neq X^+ \) and \( X^+ \neq [\text{all attributes}] \)
Example BCNF Decomposition

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

SSN → name, race
race → hairColor

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

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

Note the keys!

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

R(A, B, C, D)

R(A, B, C, D)

A \rightarrow B
B \rightarrow C
Example: BCNF

Recall: find X s.t.
X ⊏ X⁺ ⊏ [all-attrs]

R(A, B, C, D)

A → B
B → C
Example: BCNF

R(A, B, C, D)

A → B
B → C

A⁺ = ABC ≠ ABCD
Example: BCNF

R(A, B, C, D)

A → B
B → C

A⁺ = ABC ≠ ABCD

R₁(A, B, C)

R₂(A, D)
Example: BCNF

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

\[ \text{R}_1(A, B, C) \]
\[ B^+ = BC \neq ABC \]

\[ \text{R}_2(A, D) \]

\[ A \rightarrow B \]
\[ B \rightarrow C \]
Example: BCNF

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

$A^+ = ABC \neq ABCD$

$R_1(A, B, C)$

$B^+ = BC \neq ABC$

$R_11(B, C)$

$R_12(A, B)$

$R_2(A, D)$

What 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

name → price, but not category

<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>

- **Left Side:**
  - **Name:** Gizmo, OneClick, Gizmo
  - **Price:** 19.99, 24.99, 19.99

- **Right Side:**
  - **Name:** Gizmo, OneClick, Gizmo
  - **Category:** Gadget, Camera, Camera
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>

CSE 414 - Fall 2017
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

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

\[ S_1(A_1, ..., A_n, B_1, ..., B_m) \quad S_2(A_1, ..., A_n, C_1, ..., C_p) \]

Let:
- \( S_1 = \) projection of \( R \) on \( A_1, ..., A_n, B_1, ..., B_m \)
- \( S_2 = \) projection of \( R \) on \( A_1, ..., A_n, C_1, ..., C_p \)

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

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

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

• 1st Normal Form = all tables are flat (no list values)
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = no bad FDs
• 3rd Normal Form = see book
  – BCNF is lossless but can cause lose ability to check some FDs without a join (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