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
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:

\[
A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m
\]
**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 \( t, t' \in R \),
- \((t.A_1 = t'.A_1 \text{ and } \ldots t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \text{ and } \ldots t.B_n = t'.B_n)\)

<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_n )</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>

Never have equal As but different Bs!

If \( t, t' \) agree here then \( t, t' \) agree here

CSE 414 - Fall 2017
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

Do all the FDs hold on this instance?

<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>
### Example

The table below shows a sample of items with their respective names, categories, colors, departments, and prices.

<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:

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

Then this FD also holds:

\[
\text{name, category} \rightarrow \text{price}
\]

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 $\{A_1, \ldots, A_n\}^+ = \text{the set of attributes } B$ s.t. $A_1, \ldots, A_n \rightarrow B$

Example:
1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

Closures:
- $name^+ = \{\text{name, color}\}$
- $\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$
- $\text{color}^+ = \{\text{color}\}$
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: 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 \( \{A, B\}^+ \)

\[ X = \{A, B, \} \]

Compute \( \{A, F\}^+ \)

\[ X = \{A, F, \} \]
Example

In class:

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

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

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

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

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

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:

| A, B → C |
| A, D → B |
| B → D |

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

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

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

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

\begin{algorithm}
\textbf{Normalize}(R)
\begin{align*}
\text{find } X \text{ s.t.: } X &\neq X^+ \text{ and } X^+ \neq \{\text{all attributes}\} \\
\text{if } (\text{not found}) &\text{ then } \text{“R is in BCNF”} \\
\text{let } Y &= X^+ - X; \quad Z = \{\text{all attributes}\} - X^+ \\
\text{decompose } R \text{ into } R_1(X \cup Y) \text{ and } R_2(X \cup Z) \\
\text{Normalize}(R_1); \quad \text{Normalize}(R_2);
\end{align*}
\end{algorithm}
Example

The only key is: \{SSN, PhoneNumber\}
Hence \(\text{SSN} \rightarrow \text{Name, City}\) is a “bad” dependency

In other words:
\(\text{SSN}^+ = \text{SSN, Name, City}\) and is neither \text{SSN} nor All Attributes

<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 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>

$$SSN \rightarrow Name, City$$

<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?
Example BCNF Decomposition

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

SSN → name, race
race → hairColor

Find X s.t.: X ≠X⁺ and X⁺ ≠ [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)

Find X s.t.: X ≠ X⁺ and X⁺ ≠ [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)

What are the keys?

Find X s.t.: X ≠X⁺ and X⁺ ≠ [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)

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

R(A, B, C, D)

A → B
B → C
Example: BCNF

R(A, B, C, D)

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

A → B
B → C
Example: BCNF

R(A, B, C, D)

A → B
B → C

A⁺ = ABC ≠ ABCD
Example: BCNF

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

\[ A^+ = ABC \neq ABCD \]

\[ R_1(A, B, C) \]

\[ R_2(A, D) \]

\[ A \rightarrow B \]

\[ B \rightarrow C \]
Example: BCNF

R(A, B, C, D)
A⁺ = ABC ≠ ABCD

R₁(A, B, C)
B⁺ = BC ≠ ABC

R₂(A, D)

A → B
B → C
Example: BCNF

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 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 $\rightarrow$ 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>

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</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>
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>

What is lossy here?
## 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 1: Name and Category

<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 2: Price and Category

<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(A₁, ..., Aₙ, B₁, ..., Bₘ, C₁, ..., Cₚ)

S₁(A₁, ..., Aₙ, B₁, ..., Bₘ)  S₂(A₁, ..., Aₙ, C₁, ..., Cₚ)

Let:  
S₁ = projection of R on A₁, ..., Aₙ, B₁, ..., Bₘ  
S₂ = projection of R on A₁, ..., Aₙ, C₁, ..., Cₚ  
The decomposition is called \textit{lossless} if R = S₁ \Join S₂

Fact: If A₁, ..., Aₙ → B₁, ..., Bₘ 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