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

\[ A_1, A_2, \ldots, A_m \]

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_n \]

Formally:

\[ A_1, A_2, \ldots, A_m \rightarrow B_1, B_2, \ldots, B_n \]

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>

EmpID \rightarrow Name, Phone, Position
Position \rightarrow Phone
but not Phone \rightarrow Position

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 \( t, t' \in R \),
\( (t.A_1 = t'.A_1 \text{ and } \ldots t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \text{ and } \ldots t.B_n = t'.B_n) \)

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>49</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-sup.</td>
<td>59</td>
</tr>
</tbody>
</table>

Do all the FDs hold on this instance?

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)

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

Closures:

\begin{align*}
name^* &= \{name, color\} \\
\{name, category\}^* &= \{name, category, color, department, price\} \\
\text{color}^* &= \{color\}
\end{align*}
Closure Algorithm

\[ X = \{A_1, \ldots, A_n\}. \]

Repeat until \( X \) doesn't change do:

1. 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 \( X \).

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

Hence:

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

\[
\begin{align*}
\text{1. name} & \rightarrow \text{color} \\
\text{2. category} & \rightarrow \text{department} \\
\text{3. color, category} & \rightarrow \text{price}
\end{align*}
\]

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

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

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

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:

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

Find all FD's implied by:

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

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

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

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

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

Practice at Home

Find all FD's implied by:

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

Practice at Home

Find all FD's implied by:

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

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

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

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

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

• A superkey is a set of attributes $A_1, ..., A_n$ 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)

What is the key?

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

Hence \{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

$A \rightarrow B$
$B \rightarrow C$
$C \rightarrow A$

or

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

or

$B \rightarrow AC$

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

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

\[
\text{Normalize}(R) \\
\text{find } X \text{ s.t.: } X \neq X^+ \text{ and } X^+ \neq \{\text{all attributes}\} \\
\text{if } (\text{not found}) \text{ then } "R \text{ is in BCNF}"
\]

\[
\text{let } Y = X^+ - X; \quad Z = \{\text{all attributes}\} - X^+ \\
\text{decompose } R \text{ into } R_1(X \cup Y) \text{ and } R_2(X \cup Z) \\
\text{Normalize}(R_1); \text{ Normalize}(R_2);
\]

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

The only key is: (SSN, PhoneNumber)

Hence SSN \( \rightarrow \) Name, City is a "bad" dependency

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

Find X s.t.: X \#X\(^+\) and X\(^+\) \# [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 \( \rightarrow \) name, race
race \( \rightarrow \) 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)

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

R(A, B, C, D)

A → B
B → C

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

R(A, B, C, D)
A* = ABC ≠ ABCD

Example: BCNF

R(A, B, C, D)
A → B
B → C

R1(A, B, C)
R2(A, D)

Note the keys!
**Example: BCNF**

R(A, B, C, D)

A → B
B → C

R(A, B, C, D)
A* = ABC ≠ ABCD

R1(A, B, C)
B* = BC ≠ ABC

R2(A, D)

**R(A, B, C, D)**
A+ = ABC ≠ ABCD

R1(A, B, C)
B* = BC ≠ ABC

R2(A, D)

R1(B, C)
R2(A, B)

What happens if in R we first pick B*? Or AB*?

What are the keys?

---

**Decompositions in General**

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

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

S₁ = projection of R on A₁, ..., Aₙ, B₁, ..., Bₘ
S₂ = projection of R on A₁, ..., Aₙ, C₁, ..., Cₚ

---

**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>

---

**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>

---

Lossy Decomposition
Decomposition in General

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

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

The decomposition is called **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

- **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