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  

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?  

Anomalies:  
• Redundancy = repeat data  
• Update anomalies = what if Fred moves to “Bellevue”?  
• Deletion anomalies = what if Joe deletes his phone number?
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:

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

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>

**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>

**Example**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
</tr>
</tbody>
</table>
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-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)

Closure of a set of Attributes

**Given** a set of attributes $A_1, ..., A_n$, the **closure** $\{A_1, ..., A_n\}^*$ is the set of attributes $B$ such that $A_1, ..., A_n \rightarrow B$

**Example:**

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

**Closures:**

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

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 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 \( X \).

\[ \{\text{name, category}\} + = \{\} \]

Example:

\[ \text{name, category, color, department, price} \]

Hence:

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

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

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

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 \( (A, F)^+ \)

\( X = \{A, F\} \)

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:

\[ A, B \rightarrow C \]
\[ A, D \rightarrow B \]
\[ B \rightarrow D \]

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:\n
\[ BCD^+ = BCD, \quad ABCD^+ = ABCD \]

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

\[ AB \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B \]
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)

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

\[ A \rightarrow B \] or \[ AB \rightarrow C \]

\[ B \rightarrow C \] or \[ BC \rightarrow A \]

\[ C \rightarrow A \] or \[ A \rightarrow BC \]

\[ B \rightarrow AC \]

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

\( \text{Normalize}(R) \)

1. find \( X \) s.t.: \( X \neq X^+ \) and \( X^+ \neq \text{ [all attributes]} \)
2. if (not found) then "R is in BCNF"
3. let \( Y = X^+ - X \); \( Z = \text{ [all attributes]} - X^+ \)
4. decompose \( R \) into \( R_1(X \cup Y) \) and \( R_2(X \cup Z) \)
5. \( \text{Normalize}(R_1); \text{ Normalize}(R_2); \)

---

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>

The only key is: \( (\text{SSN, PhoneNumber}) \)
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 \( \text{All Attributes} \)

---

Example BCNF Decomposition

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

1. \( \text{SSN} \rightarrow \text{Name, City} \)
2. Let's check anomalies:
   - Redundancy ?
   - Update ?
   - Delete ?

---

Example BCNF Decomposition

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

1. \( \text{SSN} \rightarrow \text{Name, race} \)
2. race \rightarrow \text{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)

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

What are the keys?

R(A, B, C, D)

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)

Example: BCNF

R(A, B, C, D)
A* = ABC ≠ ABCD
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 → B
B → C

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 are the keys?

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

Decompositions in General

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

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

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

Lossless Decomposition

name → price, but not category

Lossy Decomposition

What is lossy here?
Decomposition in General

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

= **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