Introduction to Database Systems
CSE 444

Lectures 6-7: Database Design

Outline

• Design theory: 3.1-3.4
  – [Old edition: 3.4-3.6]

Schema Refinements = Normal Forms

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

First Normal Form (1NF)

• A database schema is in First Normal Form if all tables are flat

<table>
<thead>
<tr>
<th>Student</th>
<th>Name</th>
<th>GPA</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relational Schema Design

Conceptual Model:

Relational Model: plus FD’s

Normalization:

Data Anomalies

When a database is poorly designed we get anomalies:

Redundancy: data is repeated

Updated anomalies: need to change in several places

Delete anomalies: may lose data when we don’t want
Relational Schema Design
Recall set attributes (persons with several phones):

<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)
The above is in 1NF, but was 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>

Anomalies have gone:
• No more repeated data
• Easy to move Fred to “Bellevue” (how ?)
• Easy to delete all Joe’s phone numbers (how ?)

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
\[ A_1, A_2, \ldots, A_n \]
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: \( A_1, ..., A_m \rightarrow B_1, ..., B_n \) holds in \( R \) if:

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

Example

EmpID \( \rightarrow \) Name, Phone, Position
Position \( \rightarrow \) Phone
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:

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

Then this FD also holds:

- name, category → price

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)

A₁, A₂, ..., Aₙ → B₁, B₂, ..., Bₘ

Is equivalent to

- A₁, A₂, ..., Aₙ → B₁
- A₁, A₂, ..., Aₙ → B₂
- ...
- A₁, A₂, ..., Aₙ → Bₘ

Splitting rule and Combing rule

Armstrong's Rules (2/3)

A₁, A₂, ..., Aₙ → Aᵢ

Trivial Rule

where i = 1, 2, ..., n

Why?

Armstrong's Rules (3/3)

Transitive Rule

If

A₁, A₂, ..., Aₙ → B₁, B₂, ..., Bₘ

and

B₁, B₂, ..., Bₘ → C₁, C₂, ..., Cₚ

then

A₁, A₂, ..., Aₙ → C₁, C₂, ..., Cₚ

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:

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

Example (continued)

Answers:

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
color, category \(\rightarrow\) price

Closures:

name\(^+\) = \{name, color\}
(name, category\(^+\) = \{name, category, color, department, price\}
color\(^+\) = \{color\}

Example:

(name, category\(^+\) = \{name, category, color, department, price\}
Hence: name, category \(\rightarrow\) color, department, price

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:

(name \(\rightarrow\) color)
category \(\rightarrow\) department
color, category \(\rightarrow\) 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 \((A, F)^+\) \(X = \{A, F,\}\)
Example

In class:

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

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

Compute \((A, B)^+\):

\[ X = \{ A, B, C, D, E \} \]

Compute \((A, F)^+\):

\[ X = \{ A, F, B, C, D, E \} \]

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:

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

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

- \( A^+ = A \)
- \( B^+ = BD \)
- \( C^+ = C \)
- \( D^+ = D \)
- \( AB^+ = ABCD, AC^+ = AC, AD^+ = ABD, AC \)
- \( BC^+ = BCD, BD^+ = BD, CD^+ = CD \)
- \( ABC^+ = ABD^+ = ACD^+ = ABCD \) (no need to compute-- why?)
- \( BCD^+ = BCD, AB^+ = ABD^+ = ABCD \)

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

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

Another Example

- Enrollment(\( student, major, course, room, time \))
  - \( student \rightarrow major \)
  - \( major, course \rightarrow room \)
  - \( course \rightarrow time \)

What else can we infer? (in class, or at home)

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
category \rightarrow color

What is the key?

Example

Product(name, price, category, color)

name, category \rightarrow price
category \rightarrow color

What is the key?

(name, category) \rightarrow \{ name, category, price, color \}
Hence (name, category) is a key

Examples of Keys

Enrollment(student, address, course, room, time)

student \rightarrow address
room, time \rightarrow course
student, course \rightarrow room, time

(find keys at home)

Eliminating Anomalies

Main idea:

\bullet X \rightarrow A is OK if X is a (super)key
\bullet 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>206-555-6543</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

What is the key?

\{ SSN, PhoneNumber \} 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

\[ \begin{align*}
AB & \rightarrow C \\
BC & \rightarrow A \\
\text{or} & \\
A & \rightarrow BC \\
B & \rightarrow AC
\end{align*} \]

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

Is there a 2-attribute relation that is not in BCNF?

In practice, we have a better algorithm (coming up)

Example

Name | SSN    | PhoneNumber | City   |
-----|--------|-------------|--------|
Fred | 123-45-6789 | 206-555-1234 | Seattle |
Fred | 123-45-6789 | 206-555-6543 | Seattle |
Joe  | 987-65-4321 | 908-555-2121 | Westfield|
Joe  | 987-65-4321 | 908-555-1234 | Westfield|

What is the key? (SSN, PhoneNumber) use SSN \( \rightarrow \) Name, City to split

Example

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: ...

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

Example BCNF 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: ...

Decompositions in General
R(A1, ..., An, B1, ..., Bm, C1, ..., Cn)
R1(A1, ..., An, B1, ..., Bm)
R2(A1, ..., An, C1, ..., Cn)
R1 = projection of R on A1, ..., An, B1, ..., Bm
R2 = projection of R on A1, ..., An, C1, ..., Cn

Theory of Decomposition
- Sometimes it is correct:

<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>
<tr>
<td>OneClick</td>
<td>29.99</td>
<td>OneClick</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gizmo</td>
</tr>
</tbody>
</table>

Lossless decomposition
Incorrect Decomposition

- Sometimes it is not:

<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's incorrect? Lossy decomposition

Decompositions in General

- *R(A₁, ..., Aₙ, B₁, ..., Bₘ, C₁, ..., Cₚ)*
- *R₁(A₁, ..., Aₙ) ⊆ R₂(A₁, ..., Aₙ, C₁, ..., Cₚ)*

If *A₁, ..., Aₙ ⊆ B₁, ..., Bₘ*
Then the decomposition is lossless
Note: don’t need *A₁, ..., Aₙ ⊆ C₁, ..., Cₚ*

BCNF decomposition is always lossless. WHY?

Optional

- The following four slides are optional
- The content will not be on any exam
- But please take a look because they motivate the need for 3NF
- It’s good to know at least why 3NF exists

General Decomposition Goals

1. Elimination of anomalies
2. Recoverability of information
   - Can we get the original relation back?
3. Preservation of dependencies
   - Want to enforce FDs without performing joins

Sometimes cannot decomposed into BCNF without losing ability to check some FDs

BCNF and Dependencies

FD’s: *Unit → Company; Company, Product → Unit*
So, there is a BCNF violation, and we decompose.

FD’s: *Unit → Company; Company, Product → Unit*
So, there is a BCNF violation, and we decompose.

In BCNF we lose the FD: *Company, Product → Unit*
3NF Motivation

A relation R is in 3rd normal form if:

Whenever there is a nontrivial dep. A₁, A₂, ..., Aₙ → B for R,
then \{A₁, A₂, ..., Aₙ\} is a super-key for R,
or B is part of a key.

Tradeoffs
BCNF = no anomalies, but may lose some FDs
3NF = keeps all FDs, but may have some anomalies