Schema Refinement

Feb 4, 2010
Relational Schema Design

Conceptual Design

ER Model

Logical design

Relational Schema plus Integrity Constraints

Schema Refinement

Normalized schema
Outline

- ER diagrams (Ch 2)
- ER diagrams to relational tables (Ch 3.5)
- Schema Refinement, Normalization (Ch 19)

Please read in text.
Integrity Constraints

• Limitation on feasible data instances
  – Domain conditions
  – Key Constraint
  – Foreign key constraint
  – Functional dependency
• A relation instance can violate an IC
• But satisfaction of an IC can never be determined by inspecting a relation instance.
Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  
  **Redundant storage:**
  data is repeated
  
  **Update anomalies:**
  need to change in several places
  
  **Insertion anomalies:**
  may not be able to add data we want to
  
  **Deletion anomalies:**
  may lose data when we don’t want to
Schema Refinement

• Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
• Main refinement technique: decomposition (replacing ABCD with, say, AB and BCD, or ACD and ABD).

• Decomposition should be used judiciously:
  ▪ Is there reason to decompose a relation?
  ▪ What problems (if any) does the decomposition cause?
First Normal Form (1NF)

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

### Student

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
<th>Courses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>3.8</td>
<td>Math</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>3.7</td>
<td>DB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>3.9</td>
<td>Math</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
<td></td>
</tr>
</tbody>
</table>

### Takes

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Math</td>
</tr>
<tr>
<td>Carol</td>
<td>Math</td>
</tr>
<tr>
<td>Alice</td>
<td>DB</td>
</tr>
<tr>
<td>Bob</td>
<td>DB</td>
</tr>
<tr>
<td>Alice</td>
<td>OS</td>
</tr>
<tr>
<td>Carol</td>
<td>OS</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
</tr>
<tr>
<td>DB</td>
</tr>
<tr>
<td>OS</td>
</tr>
</tbody>
</table>
More Normal Forms

• Based on Functional Dependencies
  – 2nd Normal Form (obsolete)
  – 3rd Normal Form
  – Boyce Codd Normal Form (BCNF)
• Based on Multivalued Dependencies
  – 4th Normal Form
• Based on Join Dependencies
  – 5th Normal Form
Functional Dependencies

• A kind of integrity constraint
  – (hence, part of the schema)
• Finding them is part of the database design

Recall that a function returns a single unique value when evaluated on any input.
Functional Dependencies

Table R(.... A₁, A₂, ..., Aₙ... B₁, B₂, ..., Bₘ... )

Functional Dependency:

\[ A₁, A₂, ..., Aₙ \rightarrow B₁, B₂, ..., Bₘ \]

Meaning:

If two tuples agree on the attributes

\[ A₁, A₂, ..., Aₙ \]

then they must also agree on the attributes

\[ B₁, B₂, ..., Bₘ \]
### Functional Dependencies

**Definition:** \( A_1, ..., A_n \rightarrow B_1, ..., B_m \) holds in \( R \) if:

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

<table>
<thead>
<tr>
<th></th>
<th>( A_1 )</th>
<th>...</th>
<th>( A_n )</th>
<th>( B_1 )</th>
<th>...</th>
<th>( B_m )</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>

- if \( t, t' \) agree here
- then \( t, t' \) agree here
Examples

<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>E1847</td>
<td>John</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 Phone $\nRightarrow$ Position
Example

Product(name, category, color, department, price)

Consider these FDs:

- name → color
- category → department
- color, category → price
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?
### 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>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-sup.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
Anomalies

Hourly_emps( ssn, name, lot, rating, hourly_wages, hours_worked)

Suppose hourly wages is determined by rating:

rating → hourly_wages

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hourly_wages</th>
<th>hours_worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- Redundant storage: association between rating 8 and hourly wages 10 repeated 3 times.
- Update anomalies: hourly_wages updated in first tuple but not second
- Insertion anomalies: must know hourly_wage for rating value
- Deletion anomalies: delete all tuples with certain rating value, lost assoc.
Can null values fix problems?

• Not really.
• Insertion anomaly:
  – What if we know rating and hourly_wages for some rating, but there is no employee with that rating?
  – No. ssn can’t be null.
• Deletion anomaly:
  – If last employee with some rating and hourly_wages value is deleted, replace with nulls?
  – No. ssn can’t be null.
Reasoning about FD’s

If a given set of FDs hold, then it may be possible to infer that others will hold.

If all these FDs are true:

- name \(\rightarrow\) color
- category \(\rightarrow\) department
- color, category \(\rightarrow\) price

Then this FD also holds:

- name, category \(\rightarrow\) price

Why?

We say that the new FD is implied.
Closure of a Set of FDs

**Definition.** Given a set $F$ of functional dependencies, the *closure*, $F^+$, denotes all FDs implied by $F$.

How can we compute $F^+$?
Computing the Closure $F^+$

Armstrong’s Axioms
(here $X$, $Y$, $Z$ are sets of attributes):

- **Reflexivity**: If $Y \subseteq X$, then $X \rightarrow Y$
- **Augmentation**: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

**Theorem.** Armstrong axioms are *sound* and *complete* for computing $F^+$

What do *sound* and *complete* mean?
Additional convenient rule

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

Is equivalent to

\[
A_1, A_2, \ldots, A_n \rightarrow B_1 \\
A_1, A_2, \ldots, A_n \rightarrow B_2 \\
\ldots \\
A_1, A_2, \ldots, A_n \rightarrow B_m
\]
Armstrong’s Axioms

Sometimes called a trivial FD

\[ A_1, A_2, \ldots, A_n \rightarrow A_i \]

where \( i = 1, 2, \ldots, n \)

Why?
**Transitive Closure Rule**

If \( A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \) and \( B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_p \) then \( A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_p \)

<table>
<thead>
<tr>
<th></th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_n )</th>
<th>( B_1 )</th>
<th>( \ldots )</th>
<th>( B_m )</th>
<th>( C_1 )</th>
<th>( \ldots )</th>
<th>( C_p )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
<td>( )</td>
</tr>
</tbody>
</table>
Example (continued)

From:

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

To:

name, category → price

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category → name</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>5. name, category → color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category → category</td>
<td>Trivial rule</td>
</tr>
<tr>
<td>7. name, category → color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category → price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>
Decision Problem

Given F, Is $X \rightarrow Y$ in $F^+$

How to proceed?

- Apply Armstrong’s Axioms repeatedly to compute $F^+$
- Better: use the *Closure Algorithm* for a set of attributes (next)
Closure of a set of Attributes

Given Set of FDs $F$, and a set of attributes $A_1, \ldots, A_n$

The closure, \{$A_1, \ldots, A_n\}^+$, with respect to $F$, is the set of attributes $B$ s.t. $A_1, \ldots, A_n \rightarrow B$

Example: $F = \{
\text{name} \rightarrow \text{color},
\text{category} \rightarrow \text{department},
\text{color, category} \rightarrow \text{price}\}$

(Attribute) closures, with respect to $F$:

$name^+ = \{\text{name, color}\}$

$\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$

$color^+ = \{\text{color}\}$
Closure Algorithm (for Attributes)

Start with $X=\{A_1, \ldots, A_n\}$.

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

$\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$
Example

In class:

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

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

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

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

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

- A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. $A_1, \ldots, A_n \rightarrow B$ for all attributes $B$.

- A **key** is a minimal superkey.

  No subset of the attributes functionally determines all other attributes.
Computing Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a superkey
- Consider only the minimal superkeys

Note: there can be exponentially many keys!

- Example: $R(A,B,C)$, $AB \rightarrow C$, $BC \rightarrow A$
  Keys: $AB$ and $BC$
Examples of Keys

• **Product**(name, price, category, color)
  name, category $\rightarrow$ price  
category $\rightarrow$ color  

  Key:  \{name, category\}  
  Superkeys: supersets

• **Enrollment**(student, address, course, room, time)
  student $\rightarrow$ address  
  room, time $\rightarrow$ course  
  student, course $\rightarrow$ room, time  

  Keys are:  [in class]