CompSci 516
Data Intensive Computing Systems

Lecture 5
Design Theory and Normalization

Instructor: Sudeepa Roy
Announcements

• HW1 deadline:
  – Due on 09/21 (Thurs), 11:55 pm, no late days

• Project proposal deadline:
  – Preliminary idea and team members due by 09/18 (Mon) by email to the instructor
  – Proposal due on sakai by 09/25 (Mon), 11:55 pm
Today

• Finish RC from Lecture 4
  – DRC
  – More example

• Normalization
DRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

• Find the name and age of all sailors with a rating above 7

TRC:
{P | ∃ S ∈ Sailors (S.rating > 7 ∧ P.name = S.name ∧ P.age = S.age)}

DRC:
{<N, A> | ∃ <I, N, T, A> ∈ Sailors ∧ T > 7}

• Variables are now domain variables
• We will use use TRC
  – both are equivalent
More Examples: RC

• The famous “Drinker-Beer-Bar” example!

UNDERSTAND THE DIFFERENCE IN ANSWERS FOR ALL FOUR DRINKERS

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.
Find drinkers that frequent *some* bar that serves *some* beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

a shortcut for
\[
\{ x \mid \exists F \in \text{Frequents} \exists S \in \text{Serves} \exists L \in \text{Likes} ((L.\text{drinker} = F.\text{drinker}) \land (F.\text{bar} = S.\text{bar}) \land (S.\text{beer} = L.\text{beer})) \land (x.\text{drinker} = F.\text{drinker}) \} 
\]

The difference is that in the first one, one variable = one attribute in the second one, one variable = one tuple (Tuple RC)
Both are equivalent and feel free to use the one that is convenient to you
Find drinkers that frequent **some** bar that serves **some** beer they like.

\[
Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z)
\]

Find drinkers that frequent **only** bars that serves **some** beer they like.

\[
Q(x) = ...
\]
Drinker Category 2

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]
Drinker Category 3

Find drinkers that frequent **some** bar that serves **some** beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent **only** bars that serves **some** beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

Find drinkers that frequent **some** bar that serves **only** beers they like.

\[ Q(x) = \ldots \]
Drinker Category 3

Find drinkers that frequent some bar that serves some beer they like.

\[
Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z)
\]

Find drinkers that frequent only bars that serves some beer they like.

\[
Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z))
\]

Find drinkers that frequent some bar that serves only beers they like.

\[
Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z))
\]
Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y,z) \land \text{Likes}(x,z)) \]

Find drinkers that frequent some bar that serves only beers they like.

\[ Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z)) \]

Find drinkers that frequent only bars that serves only beer they like.

\[ Q(x) = \ldots \]
Drinker Category 4

Find drinkers that frequent some bar that serves some beer they like.

\[ Q(x) = \exists y. \exists z. \text{Frequents}(x, y) \land \text{Serves}(y, z) \land \text{Likes}(x, z) \]

Find drinkers that frequent only bars that serves some beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow (\exists z. \text{Serves}(y, z) \land \text{Likes}(x, z)) \]

Find drinkers that frequent some bar that serves only beers they like.

\[ Q(x) = \exists y. \text{Frequents}(x, y) \land \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]

Find drinkers that frequent only bars that serves only beer they like.

\[ Q(x) = \forall y. \text{Frequents}(x, y) \Rightarrow \forall z. (\text{Serves}(y, z) \Rightarrow \text{Likes}(x, z)) \]
Why should we care about RC

- RC is declarative, like SQL, and unlike RA (which is operational)
- Gives foundation of database queries in first-order logic
  - you cannot express all aggregates in RC, e.g. cardinality of a relation or sum (possible in extended RA and SQL)
  - still can express conditions like “at least two tuples” (or any constant)
- RC expression may be much simpler than SQL queries
  - and easier to check for correctness than SQL
  - power to use $\forall$ and $\Rightarrow$
  - then you can systematically go to a “correct” SQL query
From RC to SQL

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]
Query: Find drinkers that like some beer so much that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\text{Serves}(z, y) \Rightarrow \text{Frequents}(x, z)) \]

\[ \equiv Q(x) = \exists y. \text{Likes}(x, y) \land \forall z. (\neg \text{Serves}(z, y) \lor \text{Frequents}(x, z)) \]

Step 1: Replace \( \forall \) with \( \exists \) using de Morgan’s Laws

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg \text{Frequents}(x, z)) \]
From RC to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

\[ Q(x) = \exists y. \text{Likes}(x, y) \land \neg \exists z. (\text{Serves}(z, y) \land \neg \text{Frequents}(x, z)) \]

Step 2: Translate into SQL

```sql
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists
    (SELECT S.bar
    FROM Serves S
    WHERE L.beer=S.beer
    AND not exists (SELECT *
                    FROM Frequents F
                    WHERE F.drinker=L.drinker
                    AND F.bar=S.bar))
```

We will see a “methodical and correct” translation through “safe queries” in Datalog.
Summary

• You learnt three query languages for the Relational DB model
  – SQL
  – RA
  – RC

• All have their own purposes

• You should be able to write a query in all three languages and convert from one to another
  – However, you have to be careful, not all “valid” expressions in one may be expressed in another
  – \{S \mid \neg (S \in \text{Sailors})\} – infinitely many tuples – an “unsafe” query
  – More when we do “Datalog”, also see Ch. 4.4 in [RG]
Where are we now?

We learnt

- Relational Model and Query Languages
  - SQL, RA, RC
  - Postgres (DBMS)
  - XML (overview)
- HW1

Next
- Database Normalization
  - (for good schema design)
Design Theory and Normalization
Reading Material

• *Database normalization*
  – [RG] Chapter 19.1 to 19.5, 19.6.1, 19.8 (overview)
  – [GUW] Chapter 3

Acknowledgement:
• The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
• Some slides have been adapted from slides by Profs. Magda Balazinska, Dan Suciu, and Jun Yang
What will we learn?

• What goes wrong if we have redundant info in a database?
• Why and how should you refine a schema?
• Functional Dependencies – a new kind of integrity constraints (IC)
• Normal Forms
• How to obtain those normal forms
The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- key = SSN
Example

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn  (S)</th>
<th>name  (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

- key = SSN
- Suppose for a given rating, there is only one hourly_wage value
- Redundancy in the table
- Why is redundancy bad?
Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

1. **Redundant storage:**
   - Some information is stored repeatedly
   - The rating value 8 corresponds to hourly_wage 10, which is stored three times
Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10 → 9</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

2. Update anomalies
   - If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
   - Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency
Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

3. **Insertion anomalies:**
   - It may not be possible to store certain information unless some other, unrelated info is stored as well
   - We cannot insert a tuple for an employee unless we know the hourly wage for the employee’s rating value
Why is redundancy bad?

The list of hourly employees in an organization

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

4. **Deletion anomalies:**
   - It may not be possible delete certain information without losing some other information as well
   - If we delete all tuples with a given rating value (Attishoo, Smiley, Madayan), we lose the association between that rating value and its hourly_wage value
Nulls may or may not help

- Does not help redundant storage or update anomalies
- May help insertion and deletion anomalies
  - can insert a tuple with null value in the hourly_wage field
  - but cannot record hourly_wage for a rating unless there is such an employee (SSN cannot be null) – same for deletion

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</th>
<th>hourly-wage (W)</th>
<th>hours-worked (H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
Summary: Redundancy

Therefore,

- Redundancy arises when the schema forces an association between attributes that is “not natural”
- We want schemas that do not permit redundancy
  - at least identify schemas that allow redundancy to make an informed decision (e.g. for performance reasons)
- Null value may or may not help

- Solution?
  - decomposition of schema
## Decomposition

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hourly-wage</th>
<th>hours-worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hours-worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-11-1111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>hourly_wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several normal forms
   - If a relation is not in one of them, may need to decompose further

2. What are the problems with decomposition?
   - Lossless joins (soon)
   - Performance issues -- decomposition may both
     • help performance (for updates, some queries accessing part of data), or
     • hurt performance (new joins may be needed for some queries)
Functional Dependencies (FDs)

- A functional dependency (FD) \(X \rightarrow Y\) holds over relation \(R\) if, for every allowable instance \(r\) of \(R\):
  - i.e., given two tuples in \(r\), if the \(X\) values agree, then the \(Y\) values must also agree
  - \(X\) and \(Y\) are sets of attributes
  - \(t1 \in r, \ t2 \in r, \ \Pi_X(t1) = \Pi_X(t2)\) implies \(\Pi_Y(t1) = \Pi_Y(t2)\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

What is an FD here?
Functional Dependencies (FDs)

- A functional dependency (FD) $X \rightarrow Y$ holds over relation $R$ if, for every allowable instance $r$ of $R$:
  - i.e., given two tuples in $r$, if the $X$ values agree, then the $Y$ values must also agree
  - $X$ and $Y$ are sets of attributes
  - $t1 \in r$, $t2 \in r$, $\Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

What is an FD here?

AB $\rightarrow$ C

Note that, AB is not a key

not a correct question though.. see next slide!
Functional Dependencies (FDs)

• An FD is a statement about all allowable relations
  – Must be identified based on semantics of application
  – Given some allowable instance \( r1 \) of \( R \), we can check if it violates some FD \( f \), but we cannot tell if \( f \) holds over \( R \)

• \( K \) is a candidate key for \( R \) means that \( K \rightarrow R \)
  – denoting \( R = \) all attributes of \( R \) too
  – However, \( S \rightarrow R \) does not require \( S \) to be minimal
  – e.g. \( S \) can be a superkey
Example

• Consider relation obtained from Hourly_Emps:
  – Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)

• Notation: We will denote a relation schema by listing the attributes: SNLRWH
  – Basically the set of attributes \{S,N,L,R,W,H\}
  – here first letter of each attribute

• FDs on Hourly_Emps:
  – ssn is the key: \( S \rightarrow SNLRWH \)
  – rating determines hourly_wages: \( R \rightarrow W \)
Armstrong’s Axioms

- X, Y, Z are sets of attributes

- **Reflexivity:** If $X \supseteq Y$, 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 $X \rightarrow Z$

### Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>

Apply these rules on $AB \rightarrow C$ and check
Armstrong’s Axioms

• X, Y, Z are sets of attributes

• Reflexivity: If $X \supseteq Y$, 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 $X \rightarrow Z$

• These are sound and complete inference rules for FDs
  – sound: then only generate FDs in $F^+$ for F
  – complete: by repeated application of these rules, all FDs in $F^+$ will be generated
Additional Rules

• Follow from Armstrong’s Axioms

• Union:  If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

• Decomposition:  If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
<td></td>
</tr>
</tbody>
</table>

A $\rightarrow$ B, A $\rightarrow$ C
A $\rightarrow$ BC
A $\rightarrow$ BC
A $\rightarrow$ B, A $\rightarrow$ C

A $\rightarrow$ B, A $\rightarrow$ C
Closure of a set of FDs

• Given some FDs, we can usually infer additional FDs:
  – SSN $\rightarrow$ DEPT, and DEPT $\rightarrow$ LOT implies SSN $\rightarrow$ LOT

• An FD $f$ is implied by a set of FDs $F$ if $f$ holds whenever all FDs in $F$ hold.

• $F^+$

  = closure of $F$ is the set of all FDs that are implied by $F$
To check if an FD belongs to a closure

• Computing the closure of a set of FDs can be expensive
  – Size of closure can be exponential in #attributes

• Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$

• No need to compute $F^+$

1. Compute attribute closure of $X$ (denoted $X^+$) wrt $F$:
   – Set of all attributes $A$ such that $X \rightarrow A$ is in $F^+$

2. Check if $Y$ is in $X^+$
Computing Attribute Closure

Algorithm:

• closure = X

• Repeat until no change
  – if there is an FD $U \rightarrow V$ in F such that $U \subseteq$ closure, then closure = closure $\cup$ V

• Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\ D \rightarrow E\}$ imply $A \rightarrow E$?
  – i.e., is $A \rightarrow E$ in the closure $F^+$? Equivalently, is $E$ in $A^+$?
Normal Forms

• Question: given a schema, how to decide whether any schema refinement is needed at all?

• If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized

• Helps us decide whether decomposing the relation is something we want to do
FDs play a role in detecting redundancy

Example

- Consider a relation R with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given A → B: Several tuples could have the same A value, and if so, they’ll all have the same B value – redundancy – decomposition may be needed if A is not a key

- Intuitive idea:
  - if there is any non-key dependency, e.g. A → B, decompose!
Normal Forms

R is in 4NF
⇒ R is in BCNF
⇒ R is in 3NF
⇒ R is in 2NF (a historical one, not covered)
⇒ R is in 1NF (every field has atomic values)

Definitions next
Boyce-Codd Normal Form (BCNF)

• Relation R with FDs $F$ is in BCNF if, for all $X \rightarrow A$ in $F$
  – $A \in X$ (called a trivial FD), or
  – $X$ contains a key for $R$
    • i.e. $X$ is a superkey

Next lecture: BCNF decomposition algorithm