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)
Announcements

• No class or office hour on Thursday
  – Classes are canceled due to Hurricane alert
  – Make up classe/office hour to be announced later

• Reminder: HW1
  – Sakai : Resources -> HW -> HW1 folder
  – Due on 09/20 (Thurs), 11:55 pm, no late days
  – Start now!
  – Submission instructions for gradescope are updated on piazza
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
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 (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>

<table>
<thead>
<tr>
<th>ssn (S)</th>
<th>name (N)</th>
<th>lot (L)</th>
<th>rating (R)</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>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>
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
  – $t_1 \in r, \ t_2 \in r, \ \Pi_X(t_1) = \Pi_X(t_2)$ implies $\Pi_Y(t_1) = \Pi_Y(t_2)$

<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: \text{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 $X \rightarrow Z$

<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>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
<td></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>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>a2</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>d2</td>
</tr>
</tbody>
</table>

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

\[
A \rightarrow BC \\
A \rightarrow B, A \rightarrow C
\]
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:

• \( \text{closure} = X \)

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

• 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)
⇒ R is in 1NF (every field has atomic values)

Only BCNF and 4NF are covered in the class
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
Decomposition

(on twitter)

- User id
- user name
- Twitter id
- Group id
- Joining Date

(to a group)

- Eliminates redundancy
- To get back to the original relation: ✗
Unnecessary decomposition

- **Fine**: join returns the original relation
- **Unnecessary**: no redundancy is removed; schema is more complicated (and *uid* is stored twice!)

```latex
\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
uid & uname & twitterid \\
\hline
142 & Bart & @BartJSimpson \\
123 & Milhouse & @MilhouseVan_ \\
857 & Lisa & @lisasimpson \\
456 & Ralph & @ralphwiggum \\
\ldots & \ldots & \ldots \\
\hline
\end{tabular}
\end{table}
```

```latex
\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
uid & uname \\
\hline
142 & Bart \\
123 & Milhouse \\
857 & Lisa \\
456 & Ralph \\
\ldots & \ldots \\
\hline
\end{tabular}
\end{table}
```

```latex
\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
uid & twitterid \\
\hline
142 & @BartJSimpson \\
123 & @MilhouseVan_ \\
857 & @lisasimpson \\
456 & @ralphwiggum \\
\ldots & \ldots \\
\hline
\end{tabular}
\end{table}
```
Bad decomposition

- Association between *gid* and *fromDate* is lost
- Join returns more rows than the original relation
Lossless join decomposition

- Decompose relation $R$ into relations $S$ and $T$
  - $\text{attrs}(R) = \text{attrs}(S) \cup \text{attrs}(T)$
  - $S = \pi_{\text{attrs}(S)}(R)$
  - $T = \pi_{\text{attrs}(T)}(R)$
- The decomposition is a lossless join decomposition if, given known constraints such as FD’s, we can guarantee that $R = S \bowtie T$

- $R \subseteq S \bowtie T$ or $R \supseteq S \bowtie T$?

- Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  - A lossy decomposition is one with $R \subset S \bowtie T$
Loss? But I got more rows!

- “Loss” refers not to the loss of tuples, but to the loss of information
  – Or, the ability to distinguish different original relations

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>fromDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>123</td>
<td>gov</td>
<td>1989-12-17</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
<td>1988-09-01</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
<td>1987-04-19</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>1991-04-25</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>1992-09-01</td>
</tr>
</tbody>
</table>

No way to tell which is the original relation

---

Duke CS, Fall 2018

CompSci 516: Database Systems
BCNF decomposition algorithm

• Find a BCNF violation
  – That is, a non-trivial FD $X \rightarrow Y$ in $R$ where $X$ is not a super key of $R$
• Decompose $R$ into $R_1$ and $R_2$, where
  – $R_1$ has attributes $X \cup Y$
  – $R_2$ has attributes $X \cup Z$, where $Z$ contains all attributes of $R$ that are in neither $X$ nor $Y$
• Repeat until all relations are in BCNF
• Also gives a lossless decomposition!
BCNF decomposition example - 1

- \text{CSJDPQV}, \text{ key C, } F = \{\text{JP} \rightarrow \text{C, SD} \rightarrow \text{P, J} \rightarrow \text{S}\}
  - To deal with \text{SD} \rightarrow \text{P}, decompose into \text{SDP, CSJDQV}.
  - To deal with \text{J} \rightarrow \text{S}, decompose \text{CSJDQV} into \text{JS} and \text{CJDQV}

- Is \text{JP} \rightarrow \text{C} a violation of BCNF?

- Note:
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions (can pick \text{J} \rightarrow \text{S} first)
BCNF decomposition example - 2

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: uid \rightarrow uname, twitterid

User (uid, uname, twitterid)

Member (uid, gid, fromDate)

uid, gid \rightarrow fromDate

uid \rightarrow uname, twitterid

twitterid \rightarrow uid
BCNF decomposition example - 3

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: twitterid → uid

UserId (twitterid, uid)

BCNF

UserJoinsGroup' (twitterid, uname, gid, fromDate)

BCNF violation: twitterid → uname

UserName (twitterid, uname)

BCNF

Member (twitterid, gid, fromDate)

BCNF

uid → uname, twitterid
twitterid → uid
uid, gid → fromDate

apply Armstrong’s axioms and rules!
Recap

• Functional dependencies: a generalization of the key concept
• Non-key functional dependencies: a source of redundancy
• BCNF decomposition: a method for removing redundancies
  – BCNF decomposition is a lossless join decomposition
• BCNF: schema in this normal form has no redundancy due to FD’s
BCNF = no redundancy?

- **User \((uid, gid, place)\)**
  - A user can belong to multiple groups
  - A user can register places she’s visited
  - Groups and places have nothing to do with other
  - FD’s?
    - None
  - BCNF?
    - Yes
  - Redundancies?
    - Tons!

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
<th>place</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>Springfield</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Morocco</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Morocco</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Multivalued dependencies

• A multivalued dependency (MVD) has the form \( X \rightarrow Y \), where \( X \) and \( Y \) are sets of attributes in a relation \( R \)

• \( X \rightarrow Y \) means that whenever two rows in \( R \) agree on all the attributes of \( X \), then we can swap their \( Y \) components and get two rows that are also in \( R \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( Y )</td>
<td>( Z )</td>
</tr>
<tr>
<td>( a )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a )</td>
<td>( b_2 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( a )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a )</td>
<td>( b_1 )</td>
<td>( c_2 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
User (uid, gid, place)

• uid → gid

• uid → place
  – Intuition: given uid, attributes gid and place are “independent”

• uid, gid → place
  – Trivial: LHS ∪ RHS = all attributes of R

• uid, gid → uid
  – Trivial: LHS ⊇ RHS
Complete MVD + FD rules

• FD reflexivity, augmentation, and transitivity
• MVD complementation:
  If \( X \rightarrow Y \), then \( X \rightarrow \text{attrs}(R) - X - Y \)
• MVD augmentation:
  If \( X \rightarrow Y \) and \( V \subseteq W \), then \( XW \rightarrow YV \)
• MVD transitivity:
  If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z - Y \)
• Replication (FD is MVD):
  If \( X \rightarrow Y \), then \( X \rightarrow Y \)
• Coalescence: \( \text{Try proving things using these!} \)
  If \( X \rightarrow Y \) and \( Z \subseteq Y \) and there is some \( W \) disjoint from \( Y \) such that \( W \rightarrow Z \), then \( X \rightarrow Z \)

Verify these yourself!
An elegant solution: “chase”

• Given a set of FD’s and MVD’s $\mathcal{D}$, does another dependency $d$ (FD or MVD) follow from $\mathcal{D}$?

• Procedure
  – Start with the premise of $d$, and treat them as “seed” tuples in a relation
  – Apply the given dependencies in $\mathcal{D}$ repeatedly
    • If we apply an FD, we infer equality of two symbols
    • If we apply an MVD, we infer more tuples
  – If we infer the conclusion of $d$, we have a proof
  – Otherwise, if nothing more can be inferred, we have a counterexample

Read this slide after looking at the examples.

TO BE CONTINUED IN LECTURE 6
Proof by chase

• In $R(A, B, C, D)$, does $A \rightarrow B$ and $B \rightarrow C$ imply that $A \rightarrow C$?

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Have:</td>
<td>$a$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need:</td>
<td>$a$</td>
<td>$b_1$</td>
<td>$c_2$</td>
<td>$d_1$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b_2$</td>
<td>$c_1$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>
Another proof by chase

• In $R(A, B, C, D)$, does $A \rightarrow B$ and $B \rightarrow C$ imply that $A \rightarrow C$?

<table>
<thead>
<tr>
<th>Have:</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
<td></td>
</tr>
</tbody>
</table>

In general, with both MVD’s and FD’s, chase can generate both new tuples and new equalities.
Counterexample by chase

- In $R(A, B, C, D)$, does $A \rightarrow BC$ and $CD \rightarrow B$ imply that $A \rightarrow B$?

Have:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>$b_2$</td>
<td>$c_2$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>$b_1$</td>
<td>$c_1$</td>
<td>$d_2$</td>
</tr>
</tbody>
</table>

Need:

$b_1 = b_2$

Counterexample!
4NF

• A relation $R$ is in **Fourth Normal Form (4NF)** if
  – For every non-trivial MVD $X \rightarrow Y$ in $R$, $X$ is a superkey
  – That is, all FD’s and MVD’s follow from “key $\rightarrow$ other attributes” (i.e., no MVD’s and no FD’s besides key functional dependencies)

• **4NF** is stronger than BCNF
  – Because every FD is also a MVD
4NF decomposition algorithm

• Find a 4NF violation
  – A non-trivial MVD $X \rightarrow Y$ in $R$ where $X$ is not a superkey

• Decompose $R$ into $R_1$ and $R_2$, where
  – $R_1$ has attributes $X \cup Y$
  – $R_2$ has attributes $X \cup Z$ (where $Z$ contains $R$ attributes not in $X$ or $Y$)

• Repeat until all relations are in 4NF

• Almost identical to BCNF decomposition algorithm
• Any decomposition on a 4NF violation is lossless
### 4NF decomposition example

**User** \((uid, gid, place)\)

4NF violation: \(uid \rightarrow gid\)

**Member** \((uid, gid)\)

4NF

---

**Visited** \((uid, place)\)

4NF

<table>
<thead>
<tr>
<th>(uid)</th>
<th>(gid)</th>
<th>(place)</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
<td>Springfield</td>
</tr>
<tr>
<td>142</td>
<td>dps</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
<td>Morocco</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
<td>Morocco</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

**Table:**

<table>
<thead>
<tr>
<th>(uid)</th>
<th>(place)</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Springfield</td>
</tr>
<tr>
<td>142</td>
<td>Australia</td>
</tr>
<tr>
<td>456</td>
<td>Springfield</td>
</tr>
<tr>
<td>456</td>
<td>Morocco</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF, 3NF, 2NF
- See book if interested (not covered in class)
Summary

• Philosophy behind BCNF, 4NF: Data should depend on the key, the whole key, and nothing but the key!
  – You could have multiple keys though
• Redundancy is not desired typically
  – not always, mainly due to performance reasons
• Functional/multivalued dependencies – capture redundancy
• Decompositions – eliminate dependencies
• Normal forms
  – Guarantees certain non-redundancy
  – BCNF, and 4NF
• Lossless join
• How to decompose into BCNF, 4NF
• Chase