CompSci 516
Data Intensive Computing Systems

Lecture 10
Normalization

Instructor: Sudeepa Roy
Announcements

• Change in time of Sudeepa’s office hour (only for) next week
  – 11:45 am to 12:45 pm – Monday 10/3

• Feedback on project proposal posted on sakai

• Midterm syllabus: up to Lecture 10
  – We will start a new topic Transactions next week
Where are we now?

We learnt

- Relational Model and Query Languages
  - SQL, RA, RC
  - Postgres (DBMS)
    - HW1
- Map-reduce and spark
  - HW2
- DBMS Internals
  - Storage
  - Indexing
  - Query Evaluation
  - Operator Algorithms
  - External sort
  - Query Optimization

Next

- Database Normalization
  - (for good schema design)
- Transactions
  - Basic concepts
  - Concurrency control
  - Recovery
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.
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</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(\textit{S})</th>
<th>name(\textit{N})</th>
<th>lot(\textit{L})</th>
<th>rating(\textit{R})</th>
<th>hourly-wage(\textit{W})</th>
<th>hours-worked(\textit{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 (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>

<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, Dependency preservations (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
  - \( t_1 \in r, \ t_2 \in r, \ \Pi_X (t_1) = \Pi_X (t_2) \) 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?

\( AB \rightarrow C \)

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 \( K \rightarrow R \)
  – assume \( R = \) 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\}

• FDs on Hourly_Emps:
  – ssn is the key: S → SNLRWH
  – rating determines hourly_wages: R → 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>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>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>
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 $F^+$

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

Algorithm:

• closure = X

• Repeat until no change
  – if there is an FD U → V in F such that U ⊆ closure,
    then closure = closure ∪ V

• Check if Y is in X⁺

• Does F = {A → B, B → C, C D → E} imply A → E?
  – i.e., is A → 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
Normal Forms

- $R$ is in BCNF
  $\Rightarrow R$ is in 3NF
  $\Rightarrow R$ is in 2NF (a historical one, not covered)
  $\Rightarrow R$ 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
Observations: BCNF

R is in BCNF if the only non-trivial FDs that hold over R are key constraints

– each tuple has a key and a bunch of other attributes
– No dependency in R that can be predicted using FDs alone
– If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other

• Suppose X → A and the relation is in BCNF – what can you infer?

• The two tuples must be identical (assuming a set this relation is not possible)
  – otherwise, X is not the key
  – and X → A is a non-trivial F.D.
  – violated BCNF

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y1</td>
<td>a</td>
</tr>
<tr>
<td>x</td>
<td>y2</td>
<td>?</td>
</tr>
</tbody>
</table>
Third Normal Form (3NF)

• Relation R with FDs F is in 3NF if, for all $X \rightarrow A$ in $F^+$
  - $A \in X$ (called a trivial FD), or
  - $X$ contains a key for R, or
  - $A$ is part of some key for R.

• Minimality of a key is crucial in third condition in 3NF
  - every attribute is part of some superkey (= set of all attributes)
BCNF vs. 3NF

• If R is in BCNF, obviously in 3NF
• If R is in 3NF, some redundancy is possible
  – when X → A and A is part of a key (not allowed in BCNF)

• Example:
  – Reserves(S, B, D, C), C = credit card, S → C and C → S
  – Since SBD is a key, CBD is also a key, 3NF not violated
  – but in all tuples recording the same S value, the same (S, C) pair is redundantly recorded
  – note: relation is not in BCNF since both S and C are not superkeys
Decomposition of a Relation Schema

• Consider relation $R$ contains attributes $A_1 \ldots A_n$

• A decomposition of $R$ consists of replacing $R$ by two or more relations such that “no attribute is lost” and “no new attribute appears”, i.e.
  – Each new relation schema contains a subset of the attributes of $R$
  – Every attribute of $R$ appears as an attribute of one of the new relations
  – E.g., Can decompose $SNLRWH$ into $SNLRH$ and $RW$

• What are the potential problems with an arbitrary decomposition?
Good properties of decomposition

- Lossless join decomposition
- Dependency preserving decomposition
Lossless Join Decompositions

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance r that satisfies F: \( \pi_X(r) \bowtie \pi_Y(r) = r \)

- It is always true that \( \pi_X(r) \bowtie \pi_Y(r) \supseteq r \)

- In general, the other direction does not hold
  - If it does, the decomposition is lossless-join

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>p1</td>
<td>d1</td>
</tr>
<tr>
<td>s2</td>
<td>p2</td>
<td>d2</td>
</tr>
<tr>
<td>s3</td>
<td>p1</td>
<td>d3</td>
</tr>
</tbody>
</table>

- Decompose into SP and PD -- is the decomposition lossless?
- How about SP and SD?

For lossless decomposition of R into R1, R2
- either \( R1 \cap R2 \rightarrow R1 \)
- or \( R1 \cap R2 \rightarrow R2 \)
Dependency Preserving Decomposition

• Consider CSJDPQV, C is key, JP → C and SD → P
  – Lossless decomposition: CSJDPQV and SDP
    • SD key of (SDP)!
  – Problem: Checking JP → C requires a join

• Dependency preserving decomposition:
  – join is not needed to check a dependency
Algorithm: Decomposition into BCNF

• Input: relation R with FDs F

If \( X \rightarrow Y \) violates BCNF, decompose R into \( R - Y \) and \( XY \).
Repeat until all new relations are in BCNF w.r.t. the given F

• Gives a collection of relations that are
  – in BCNF
  – lossless join decomposition
  – and guaranteed to terminate
  – but a dependency-preserving decomposition may not exist
    (example in book)
Decomposition into BCNF (example)

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

• 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
Other kinds of dependencies and normal forms

• Multi-valued dependencies
• Join dependencies
• Inclusion dependencies
• 4NF, 5NF
• See book if interested (not covered in class)
Summary

• Redundancy is not desired typically
  – not always, mainly due to performance reasons
• Functional dependencies – capture redundancy
• Decompositions – eliminate dependencies
• Normal forms
  – Guarantees certain non-redundancy
  – BCNF and 3NF
• Lossless join and dependency-preserving joins
• How to decompose into BCNF