CompSci 516
Data Intensive Computing Systems

Lecture 4
Normalization

Instructor: Sudeepa Roy
Announcement

• Homework 1
  – Part-2 will be posted right after the class/office hour today
  – Contains the questions you have to answer
  – Due on Feb 9, 11:59 pm
Today’s topic

- Database normalization
- Reading material
  - [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
• 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?

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>
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 -&gt; 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</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>hourly-wage</th>
<th>hours-worked</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>222</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>333</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>444</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>555</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 to 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.
Why is redundancy bad?

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
  – does not help redundant storage or update 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)

• Solution?
## 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

2. What are the problems with decomposition?
   - Lossless joins, Dependency preservations, Performance issues
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>
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
  - $t1 \epsilon r$, $t2 \epsilon r$, $\Pi_X (t1) = \Pi_X (t2)$ 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
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 $K \rightarrow R$
  – 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 \rightarrow\ SNLRWH
  – rating determines hourly_wages: R \rightarrow W
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 $F$
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 $XZ \rightarrowYZ$ 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></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
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 $U \rightarrow V$ in $F$ such that $U \subseteq \text{closure}$, then closure = closure $\cup$ V

• Check if Y is in $X^+$

• Does $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D \rightarrow E\}$ 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

• 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

• 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

• Sometimes other subtle integrity constraints help detect redundancy
Normal Forms

- $R$ is in BCNF
  - $\Rightarrow R$ is in 3NF
  - $\Rightarrow R$ is in 2NF (a historical one)
  - $\Rightarrow R$ 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.
– If example relation is in BCNF, the 2 tuples must be identical, since X is a key
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)
Partial and Transitive Dependencies

If 3NF violated by $X \rightarrow A$, one of the following holds:

- **$X$ is a subset of some key $K$**
  - We store $(X, A)$ pairs redundantly
  - called *partial dependency*
  - $2NF = \text{no partial dependency}$

- **$X$ is not a proper subset of any key**
  - There is a chain of FDs $K \rightarrow X \rightarrow A$, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value
  - Recall hourly_employee – cannot store the rating $R$ for an employee without knowing the hourly wage
  - called *transitive dependency*
Observations: 3NF

- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible

Example:
- X → A and X is not part of a key
- 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 some redundancy

- It is a compromise, used when BCNF not achievable
  - e.g., no "good" decomposirion, or performance considerations
- Finding all keys of a schema and detecting if a schema is in 3NF is "NP-complete"
Decomposition of a Relation Schema

• Consider relation R contains attributes A1 ... An

• A decomposition of R consists of replacing R by two or more relations such that:
  – Each new relation schema contains a subset of the attributes of R (and no attributes that do not appear in R), and
  – 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?
Problems with Decompositions

1. Some queries become more expensive
   - e.g., How much did sailor Joe earn? (salary = W*H) – now needs a join after decomposition

2. We may not be able to reconstruct the original relation from the decomposition
   - Fortunately, not in the SNLRWH example

3. Checking some original dependencies may require joining the instances of the decomposed relations
   - Fortunately, not in the SNLRWH example

• Tradeoff: Must consider these issues vs. redundancy
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 $\pi_X(r) \bowtie \pi_Y(r) \subseteq r$

• 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?
Lossless Join Decompositions

• Suppose R with FD F is decomposed into attributes R1 and R2

• The decomposition is lossless if and only if F+ contains
  – either $R_1 \cap R_2 \rightarrow R_1$
  – or $R_1 \cap R_2 \rightarrow R_2$

• Recall $SNLRWH$ and FD $R \rightarrow W$
  – Violates 3NF: R does not contain a key, W is not part of a key
  – Decompose into $SNLRH$ and $RW$
  – R is common to both and $R \rightarrow W$
  – lossless

• If $X \rightarrow Y$, and $X$, $Y$ are disjoint, then decomposing into $R-Y$ and $XY$ is lossless
Dependency Preserving Decomposition

• Consider \( CSJDPQV \), C is key, \( JP \to C \) and \( SD \to P \)
  – Lossless decomposition: \( CSJDQV \) and \( SDP \)
  – Problem: Checking \( JP \to C \) requires a join

• Dependency preserving decomposition:
  – If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold
Projection of set of FDs F:

- Suppose R is decomposed into X, Y...
- Projection of F onto X (denoted $F_X$) is the set of FDs $U \rightarrow V$ in $F^+$ such that all attributes from both $U$, $V$ are in $X$

- Note: projection from $F^+$, not only $F$
Dependency Preserving Decomposition (formal definition)

- Decomposition of R into X and Y is dependency preserving if $(F_X \cup F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure $F^+$ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in $F^+$

- Important to consider $F^+$, not only $F$, in this definition:
Dependency Preserving Decomposition (example)

- **Example**
  - ABC
  - \( F = \{ A \rightarrow B, \ B \rightarrow C, \ C \rightarrow A \} \)
  - \( F^+ = \{ A \rightarrow B, \ B \rightarrow C, \ C \rightarrow A, \ B \rightarrow A, \ C \rightarrow B, \ A \rightarrow C \} \)
  - ABC decomposed into AB and BC
  - Is this dependency preserving?
    - Yes! check yourself using the definition from the previous slide

- **Dependency preserving does not imply lossless join**
  - Check: ABC, \( A \rightarrow B \), decomposed into AB and BC
  - And *vice-versa* (see example on slide#38)
Algorithm: Decomposition into BCNF

• **Input:** relation $R$ with FDs $F$

If $X \rightarrow Y$ violates BCNF, decompose $R$ into $R \setminus Y$ 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 $SDP, CSJDVQV$.
  - To deal with $J \rightarrow S$, decompose $CSJDVQV$ 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
Other kinds of dependencies

• Multi-valued dependencies
• Join dependencies

• FDs are the most common and important
  – But these help identify redundancy that cannot be detected with FDs alone
  – Some high-level overview next
Multivalued Dependencies

<table>
<thead>
<tr>
<th>Course (C)</th>
<th>Teacher (T)</th>
<th>Book (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics101</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Green</td>
<td>Optics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Brown</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Brown</td>
<td>Optics</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Vector</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Geometry</td>
</tr>
</tbody>
</table>

- **No FDs, Key = CTB**
  - Already in BCNF
- **C is independent of B** – called Multi-valued Dependency
  - Redundancy – won’t be considered if we look at FDs only
- **Redundancy can be eliminated by decomposing CTB into CT and CB**

Duke CS, Spring 2016
## Multivalued Dependencies

<table>
<thead>
<tr>
<th>Course (C)</th>
<th>Teacher (T)</th>
<th>Book (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics101</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Green</td>
<td>Optics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Brown</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Physics101</td>
<td>Brown</td>
<td>Optics</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Mechanics</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Vector</td>
</tr>
<tr>
<td>Maths301</td>
<td>Green</td>
<td>Geometry</td>
</tr>
</tbody>
</table>

- **Multi-valued Dependency**
  - \( x \rightarrow Y \) (here \( c \rightarrow T \))
  - In every instance, each \( X \) value is associated with a set of \( Y \) values independent of the other attributes

- **Considered in 4NF**
Inclusion Dependency

• Some columns are contained in other columns
  – Usually of a second relation
  – Foreign keys are one example
  – Considered in 5NF
Summary of Schema Refinement

- Functional dependencies
- Normal forms
  - (1NF, 2NF), 3NF, BCNF, (4NF, 5NF)
- Lossless join decomposition
- Dependency preserving decomposition
- BCNF decomposition algorithm

Next topic: database internals – storage, indexing, hashing