Database Systems

Functional Dependencies

Feifei Li

University of Utah
Review: Database Design

- **Requirements Analysis**
  - user needs; what must database do?

- **Conceptual Design**
  - high level descr (often done w/ER model)

- **Logical Design**
  - translate ER into DBMS data model

- **Schema Refinement**
  - consistency, normalization

- **Physical Design** - indexes, disk layout

- **Security Design** - who accesses what
The Evils of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:
  - *redundant storage, insert/delete/update anomalies*
- Integrity constraints, in particular **functional dependencies**, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: **decomposition**
  - replacing ABCD with, say, AB and BCD, or ACD and ABD.
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?
Functional Dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation schema $R$ if, for every allowable instance $r$ of $R$:
  \[
  t_1 \in r, \ t_2 \in r, \ \pi_X(t_1) = \pi_X(t_2) \implies \pi_Y(t_1) = \pi_Y(t_2)
  \]
  (where $t_1$ and $t_2$ are tuples; $X$ and $Y$ are sets of attributes)

- In other words: $X \rightarrow Y$ means
  Given any two tuples in $r$, if the $X$ values are the same, then the $Y$ values must also be the same. (but not vice versa)

- Can read “$\rightarrow$” as “determines”
FD’s Continued

- An FD is a statement about all allowable relations.
  - Must be identified based on semantics of application.
  - Given some instance $r1$ of R, we can check if $r1$ violates some FD $f$, but we cannot determine if $f$ holds over R.

- Question: How related to keys?
- if “$K \rightarrow$ all attributes of R” then K is a superkey for R
  (does not require K to be minimal.)
- FDs are a generalization of keys.
FDs are extremely useful: some simple examples

- Given Enroll(sid, cid, score, semester, year)
  - All students enrolled in the same course (including the same course over different semesters and years) should be assigned a distinct score
    \[(cid, score) \rightarrow sid\]
  - Each student should be allowed to take only one course
    \[sid \rightarrow cid\]
  - A course must be always offered in the same semester
    \[cid \rightarrow semester\]
  - A student must be given different scores from different courses he/she takes.
    \[(sid, score) \rightarrow cid\]
Consider relation obtained from Hourly_Emps:

Hourly_Emps \((ssn, name, lot, rating, wage\_per\_hr, hrs\_per\_wk)\)

We sometimes denote a relation schema by listing the attributes: e.g., \(SNLRWH\)

This is really the set of attributes \(\{S,N,L,R,W,H\}\).

Sometimes, we refer to the set of all attributes of a relation by using the relation name. e.g., “Hourly_Emps” for SNLRWH

What are some FDs on Hourly_Emps?

**ssn is the key:** \(S \rightarrow SNLRWH\)

**rating determines wage\_per\_hr:** \(R \rightarrow W\)

**lot determines lot:** \(L \rightarrow L\) (“trivial” dependency)
Problems Due to $R \rightarrow W$

- **Update anomaly**: Can we modify $W$ in only the 1st tuple of SNLRWH?
- **Insertion anomaly**: What if we want to insert an employee and don’t know the hourly wage for his or her rating? (or we get it wrong?)
- **Deletion anomaly**: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Hourly_Emps
Detecting Redundancy

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Q: Why was R → W problematic, but S → W not?
Decomposing a Relation

- Redundancy can be removed by “chopping” the relation into pieces.
- FD’s are used to drive this process.

$R \rightarrow W$ is causing the problems, so decompose $SNLRWH$ into what relations?

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

$\begin{bmatrix}
R & W \\
8 & 10 \\
5 & 7 
\end{bmatrix}$

Wages

**Hourly_Emps2**
Refining an ER Diagram

- 1st diagram becomes:
  
  Workers(S,N,L,D,Si)
  Departments(D,M,B)
  
  - Lots associated with workers.

- Suppose all workers in a dept are assigned the same lot: \( D \rightarrow L \)

- Redundancy; fixed by:
  
  Workers2(S,N,D,Si) Dept_Lots(D,L)
  Departments(D,M,B)

- Can fine-tune this:
  
  Workers2(S,N,D,Si)
  Departments(D,M,B,L)

Before:

![Before Diagram](image)

After:

![After Diagram](image)
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  
  \[ \text{title} \rightarrow \text{studio, star} \] implies \[ \text{title} \rightarrow \text{studio} \] and \[ \text{title} \rightarrow \text{star} \]

  \[ \text{title} \rightarrow \text{studio} \] and \[ \text{title} \rightarrow \text{star} \] implies \[ \text{title} \rightarrow \text{studio, star} \]

  \[ \text{title} \rightarrow \text{studio, studio} \rightarrow \text{star} \] implies \[ \text{title} \rightarrow \text{star} \]

- But, \[ \text{title, star} \rightarrow \text{studio} \] does NOT necessarily imply that \[ \text{title} \rightarrow \text{studio} \] or that \[ \text{star} \rightarrow \text{studio} \]

- An FD \( f \) is **implied by** a set of FDs \( F \) if \( f \) holds whenever all FDs in \( F \) hold.

- \( F^+ = \text{closure of } F \) is the set of all FDs that are implied by \( F \). (includes “trivial dependencies”)

Rules of Inference

- **Armstrong’s Axioms** (X, Y, Z are sets of attributes):
  - *Reflexivity*: If X ⊇ Y, then X → Y
  - *Augmentation*: If X → Y, then XZ → YZ for any Z
  - *Transitivity*: If X → Y and Y → Z, then X → Z

- These are *sound* and *complete* inference rules for FDs!
  - i.e., using AA you can compute all the FDs in F+ and only these FDs.

- Some additional rules (that follow from AA):
  - *Union*: If X → Y and X → Z, then X → YZ
  - *Decomposition*: If X → YZ, then X → Y and X → Z
Sample Proof: prove by definition

- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$

- **Proof:**
  Given $X \rightarrow Y$ on a schema $R$, for any valid instance $r$ conforming to $R$, and any two records $t_1$ and $t_2$ from $r$, by definition, we know that $\pi_X(t_1) = \pi_X(t_2)$ implies $\pi_Y(t_1) = \pi_Y(t_2)$: Fact 1.

  Note that we also trivially have $\pi_Z(t_1) = \pi_Z(t_2)$ for any such two records $t_1$ and $t_2$ and any $Z$: Fact 2.

  Hence, by Facts 1 and 2, if we know $\pi_{XZ}(t_1) = \pi_{XZ}(t_2)$, we must have $\pi_{YZ}(t_1) = \pi_{YZ}(t_2)$

  This by definition, implies $XZ \rightarrow YZ$ for any $Z$. 
Sample Proof: prove by rules

• **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  
  **Proof:** By $X \rightarrow Y$ and augmentation rule, we have $XX \rightarrow XY$. Similarly, by $X \rightarrow Z$ and augmentation rule again, we have $XY \rightarrow YZ$
  
  Now by transitivity, we have $XX \rightarrow YZ$, which implies $X \rightarrow YZ$.

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

  **Proof:** By reflexivity, $YZ \rightarrow Y$ and $YZ \rightarrow Z$. Then, by transitivity and given $X \rightarrow YZ$, we have both $X \rightarrow Y$ and $X \rightarrow Z$. 


Example

Contracts\( (cid, sid, jid, did, pid, qty, value) \), and:
- C is the key: \( C \rightarrow CSJDPQV \)
- Proj purchases each part using single contract: \( JP \rightarrow C \)
- Dept purchases at most 1 part from a supplier: \( SD \rightarrow P \)

Problem: Prove that SDJ is a key for Contracts

\[ JP \rightarrow C, \quad C \rightarrow CSJDPQV \quad \text{imply} \quad JP \rightarrow CSJDPQV \]
(by transitivity) (shows that JP is a key)

\[ SD \rightarrow P \quad \text{implies} \quad SDJ \rightarrow JP \quad \text{(by augmentation)} \]

\[ SDJ \rightarrow JP, \quad JP \rightarrow CSJDPQV \quad \text{imply} \quad SDJ \rightarrow CSJDPQV \]
(by transitivity) thus SDJ is a key.

Q: can you now infer that \( SD \rightarrow CSDPQV \) (i.e., drop J on both sides)?

No! FD inference is not like arithmetic multiplication.
Attribute Closure

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)

- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs $F$. An efficient check:
  - Compute \textit{attribute closure} of $X$ (denoted $X^+$) wrt $F$. $X^+ = \text{Set of all attributes } A \text{ such that } X \rightarrow A \text{ is in } F^+$
    - $X^+ := X$
    - Repeat until no change: if there is an fd $U \rightarrow V$ in $F$ such that $U$ is in $X^+$, then add $V$ to $X^+$
  - Check if $Y$ is in $X^+$
  - Approach can also be used to find the keys of a relation.
    - If all attributes of $R$ are in the closure of $X$ then $X$ is a superkey for $R$.
    - Q: How to check if $X$ is a “candidate key”?
Attribute Closure (example)

- \( R = \{A, B, C, D, E\} \)
- \( F = \{ B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B \} \)
- Is \( B \rightarrow E \) in \( F^+ \)?
  - \( B^+ = B \)
  - \( B^+ = BCD \)
  - \( B^+ = BCDA \)
  - \( B^+ = BCDAE \) ... Yes! and \( B \) is a key for \( R \) too!
- Is \( D \) a key for \( R \)?
  - \( D^+ = D \)
  - \( D^+ = DE \)
  - \( D^+ = DEC \) ... Nope!
- Is \( AD \) a key for \( R \)?
  - \( AD^+ = AD \)
  - \( AD^+ = ABD \) and \( B \) is a key, so Yes!
- Is \( AD \) a \textit{candidate} key for \( R \)?
  - \( A^+ = A \), \( D^+ = DEC \)
  - ... \( A,D \) not keys, so Yes!
- Is \( ADE \) a \textit{candidate} key for \( R \)?
  - ... No! \( AD \) is a key, so \( ADE \) is a superkey, but not a candidate key
Schema Refinement and Normalization
Functional Dependencies (Review)

- A functional dependency $X \rightarrow Y$ holds over relation schema $R$ if, for every allowable instance $r$ of $R$:
  
  $t_1 \in r$, $t_2 \in r$, $\pi_X(t_1) = \pi_X(t_2)$
  
  implies $\pi_Y(t_1) = \pi_Y(t_2)$

  (where $t_1$ and $t_2$ are tuples; $X$ and $Y$ are sets of attributes)

- In other words: $X \rightarrow Y$ means
  
  Given any two tuples in $r$, if the $X$ values are the same, then the $Y$ values must also be the same. (but not vice versa)

- Can read “$\rightarrow$” as “determines”
Normal Forms

- Back to schema refinement...
- Q1: is any refinement needed??!
- If a relation is in a normal form (BCNF, 3NF etc.):
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given A \(\rightarrow\) B: If A is not a key, then several tuples could have the same A value, and if so, they’ll all have the same B value!
- 1\(^{st}\) Normal Form – all attributes are atomic
- 1\(^{st}\) \(\supset\) 2\(^{nd}\) (of historical interest) \(\supset\) 3\(^{rd}\) \(\supset\) Boyce-Codd \(\supset\) ...
Boyce-Codd Normal Form (BCNF)

- Reln R with FDs $F$ is in **BCNF** if, for all $X \rightarrow A$ in $F^+$
  - $A \in X$ (called a *trivial* FD), or
  - $X$ is a superkey for $R$.

- In other words: “R is in BCNF if the only non-trivial FDs over R are *key constraints*.”

- If R in BCNF, then every field of every tuple records information that **cannot be inferred** using FDs alone.
  - Say we know FD $X \rightarrow A$ holds this example relation:

<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>

- **Can you guess the value of the missing attribute?**

- **Yes, so relation is not in BCNF**
Decomposition of a Relation Schema

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.

- Suppose that relation R contains attributes A1 \ldots An. A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a *subset* of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.
Example (same as before)

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>

Hourly_Emps

- SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
- Q: Is this relation in BCNF?

No, The second FD causes a violation; $W$ values repeatedly associated with $R$ values.
Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

**S N L R H**

**8 10 5 7**

**Wages**

Hourly_Emps2

- Q: Are both of these relations are now in BCNF?
- Decompositions should be used only when needed.
  - Q: potential problems of decomposition?
Problems with Decompositions

- There are three potential problems to consider:
  1) May be **impossible** to reconstruct the original relation! (lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

**Tradeoff:** Must consider these issues vs. redundancy.
## Lossless Decomposition (example)

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>40</td>
</tr>
</tbody>
</table>

### Diagram

```
R  W
8  10
5  7
```

<table>
<thead>
<tr>
<th>S</th>
<th>N</th>
<th>L</th>
<th>R</th>
<th>W</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-22-3666</td>
<td>Attishoo</td>
<td>48</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>231-31-5368</td>
<td>Smiley</td>
<td>22</td>
<td>8</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>131-24-3650</td>
<td>Smethurst</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>434-26-3751</td>
<td>Guldu</td>
<td>35</td>
<td>5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>612-67-4134</td>
<td>Madayan</td>
<td>35</td>
<td>8</td>
<td>10</td>
<td>40</td>
</tr>
</tbody>
</table>
Lossy Decomposition (example)

A → B; C → B
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 \( r \subseteq \pi_X(r) \bowtie \pi_Y(r) \)
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- **It is essential that all decompositions used to deal with redundancy be lossless!** *(Avoids Problem #1)*
The decomposition of R into X and Y is **lossless with respect to F** if and only if the closure of F contains:

\[ X \cap Y \rightarrow X, \quad \text{or} \quad X \cap Y \rightarrow Y \]

in example: decomposing ABC into AB and BC is lossy, because intersection (i.e., “B”) is not a key of either resulting relation.

**Useful result:** If \( W \rightarrow Z \) holds over R and \( W \cap Z \) is empty, then decomposition of R into R-Z and WZ is loss-less.
Lossless Decomposition (example)

A → B; C → B

But, now we can’t check A → B without doing a join!
Dependency Preserving Decomposition

- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem #2 on our list.)*

- **Projection of set of FDs F:** If R is decomposed into X and Y the projection of F on X (denoted $F_X$) is the set of FDs $U \rightarrow V$ in $F^+$ (closure of F, not just F) such that all of the attributes $U, V$ are in X. *(same holds for Y of course)*
Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is \textit{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 $F^+$ in this definition:
  - ABC, A \rightarrow B, B \rightarrow C, C \rightarrow A, decomposed into AB and BC.
  - Is this dependency preserving? Is C \rightarrow A preserved????
    - note: $F^+$ contains $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$, so...

- FAB contains A \rightarrow B and B \rightarrow A; FBC contains B \rightarrow C and C \rightarrow B
- So, $(FAB \cup FBC)^+$ contains C \rightarrow A
**Decomposition into BCNF**

- Consider relation $R$ with FDs $F$. First, make sure all FDs in $F$ contain only single attribute on RHS (this is always doable, for example, if you have $AB \rightarrow CD$, split it into $AB \rightarrow C$ and $AB \rightarrow D$);

- Next, if $X \rightarrow Y$ (in $F$) violates BCNF, decompose $R$ into $R - Y$ and $XY$ (guaranteed to be loss-less).

- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
Decomposition into BCNF

- e.g., CSJDPQV, key C, JP → C, SD → P, J → S
- \{contractid, supplierid, projectid, deptid, partid, qty, value\}
- To deal with SD → P, decompose into SDP, CSJDQV.
- To deal with J → S, decompose CSJDQV into JS and CJDQV
- So we end up with: SDP, JS, and CJDQV

- Note: several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!
In general, there may not be a dependency preserving decomposition into BCNF.

- e.g., CSZ, CS → Z, Z → C
- Can’t decompose while preserving 1st FD; not in BCNF.

Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).

{contractid, supplierid, projectid, deptid, partid, qty, value}
- However, it is a lossless join decomposition.
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - but JPC tuples are stored only for checking the FD (Redundancy!)
Third Normal Form (3NF)

- Reln \( R \) with FDs \( F \) is in 3NF if, for all \( X \rightarrow A \) in \( F^+ \)
  - \( A \in X \) (called a trivial FD), or
  - \( X \) is a superkey of \( R \), or
  - \( A \) is part of some candidate key (not superkey!) for \( R \). (sometimes stated as “A is prime”)

- Minimality of a key is crucial in third condition above!

- If \( R \) is in BCNF, obviously in 3NF.

- If \( R \) is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good’’ decomp, or performance considerations).
  - Lossless-join, dependency-preserving decomposition of \( R \) into a collection of 3NF relations always possible.
What Does 3NF Achieve?

- If 3NF violated by \( X \rightarrow A \), one of the following holds:
  - \( X \) is a subset of some key \( K \) (“partial dependency”)
    - We store \((X, A)\) pairs redundantly.
    - e.g. Reserves SBDC (C is for credit card) with key SBD and \( S \rightarrow C \)
  - \( X \) is not a proper subset of any key. (“transitive dep.”)
    - There is a chain of FDs \( K \rightarrow X \rightarrow A \)
    - So we can’t associate an \( X \) value with a \( K \) value unless we also associate an \( A \) value with an \( X \) value (different \( K \)’s, same \( X \) implies same \( A \)!)

- But: even if \( R \) is in 3NF, these problems could arise.
  - e.g., Reserves SBDC (note: “C” is for credit card here), \( S \rightarrow C, C \rightarrow S \) is in 3NF (why?), but for each reservation of sailor \( S \), same \((S, C)\) pair is stored.

- Thus, 3NF is indeed a compromise relative to BCNF.
  - You have to deal with the partial and transitive dependency issues in your application code!
Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.

- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation $XY$.
    Problem is that $XY$ may violate 3NF! e.g., consider the addition of CJP to `preserve’ $JP \rightarrow C$.
    What if we also have $J \rightarrow C$?

- Refinement: Instead of the given set of FDs $F$, use a *minimal cover for $F$*. 
Minimal Cover for a Set of FDs

- **Minimal cover** $G$ for a set of FDs $F$:
  - Closure of $F = $ closure of $G$.
  - Right hand side of each FD in $G$ is a single attribute.
  - If we modify $G$ by deleting an FD or by deleting attributes from an FD in $G$, the closure changes.

- Intuitively, every FD in $G$ is needed, and "as small as possible" in order to get the same closure as $F$.

- e.g., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$

- M.C. implies Lossless-Join, Dependency Preserving Decomposition!!!
  - Start with M.C. of $F$, do the decomposition from last slide
3NF Decomposition and Minimal Cover

- *How to compute Minimal Cover? This course DOES NOT require you to understand this, which is quite involved.*
Summary of Schema Refinement

- BCNF: each field contains information that cannot be inferred using only FDs.
  - ensuring BCNF is a good heuristic.

- Not in BCNF? Try decomposing into BCNF relations.
  - Must consider whether all FDs are preserved!

- Lossless-join, dependency preserving decomposition into BCNF impossible? Consider 3NF.
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.

- Note: even more restrictive Normal Forms exist (we don’t cover them in this course, but some are in the book.)