ICS 421 Spring 2010

Normal Forms

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Two More Rules

- **Union**
  - If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Eg. $FLD \rightarrow A$ and $FLD \rightarrow T$, then $FLD \rightarrow AT$

- **Decomposition**
  - If $X \rightarrowYZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Eg. $FLD \rightarrow AT$, then $FLD \rightarrow A$ and $FLD \rightarrow T$

- **Trivial FDs**
  - Right side is a subset of Left side
  - Eg. $F \rightarrow F$, $FLD \rightarrow FD$
Closure

• **Implication**: An FD $f$ is *implied by* a set of FDs $F$ if $f$ holds whenever all FDs in $F$ hold.
  
  – $f=A \rightarrow C$ is implied by $F=\{ A \rightarrow B, B \rightarrow C \}$ (using Armstrong’s transitivity)

• **Closure $F^+$**: the set of all FDs implied by $F$
  
  – **Algorithm**:
    
    • start with $F^+=F$
    
    • keep adding new implied FDs to $F^+$ by applying the 5 rules (Armstrong’s Axioms + union + decomposition)
    
    • Stop when $F^+$ does not change anymore.
Example: Closure

- Given FLD is the primary key and C → Z
- Find the closure:
  - Start with \{ FLD → FLDSCZT, C→Z \}
  - Applying reflexivity, \{ FLD → F, FLD →L, FLD → D, FLD → FL, FLD → LD, FLD →DF, FLDSCZT → FLD, ... \}
  - Applying augmentation, \{ FLDS → FS, FLDS → LS, ... \}
  - Applying transitivity ...
  - Applying union ...
  - Applying decomposition ...
  - Repeat until \( F^+ \) does not change
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 *attribute closure* of $X$ (denoted $X^+$) wrt $F$:
    • Set of all attributes $A$ such that $X \rightarrow A$ is in $F^+$
    • There is a linear time algorithm to compute this.
  – 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

• Helps with the question: do we need to refine the schema?

• If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.

• Role of FDs in detecting redundancy:
  – Consider a relation R with 3 attributes, ABC.
    • No FDs hold: There is no redundancy here.
    • Given A → B: Several tuples could have the same A value, and if so, they’ll all have the same B value!
Boyce-Codd Normal Form (BCNF)

• Let $R$ denote a relation, $X$ a set of attributes from $R$, $A$ an attribute from $R$, and $F$ the set of FDs that hold over $R$.

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

• **Negation**: $R$ is not in BCNF if there exists an $X \rightarrow A$ in $F^+$, such that $A \notin X$ (non-trivial FD) AND $X$ is not a key


The only non-trivial FDs that hold are key constraints
Examples: BCNF

• Are the following in BCNF?

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>DOB</th>
<th>Address</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Sep 9 1979</td>
<td>Honolulu, HI</td>
<td>808-343-0809</td>
</tr>
</tbody>
</table>

\[ F = \{ \text{FLD} \rightarrow \text{FLDAT} \} \]

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>DOB</th>
<th>Street</th>
<th>CityState</th>
<th>Zipcode</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Sep 9 1979</td>
<td>1680 East West Rd.</td>
<td>Honolulu, HI</td>
<td>96822</td>
<td>808-343-0809</td>
</tr>
</tbody>
</table>

\[ F = \{ \text{FLD} \rightarrow \text{FLDSCZT}, \text{C} \rightarrow \text{Z} \} \]
Third Normal Form (3NF)

• Let R denote a relation, X a set of attributes from R, A an attribute from R, and F the set of FDs that hold over R.
• R is in **3NF** if for all \( X \rightarrow A \) in \( F^+ \),
  – \( A \in X \) (trivial FD) or
  – \( X \) is a superkey or
  – \( A \) is part of some key
• **Negation**: R is not in 3NF if there exists an \( X \rightarrow A \) in \( F^+ \), such that \( A \notin X \) (non-trivial FD) AND \( X \) is not a key AND \( A \) is not part of some key
• 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).
Example: 3NF

• Which of the following is in 3NF and which in BCNF?

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>DOB</th>
<th>Address</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Sep 9 1979</td>
<td>Honolulu, HI</td>
<td>808-343-0809</td>
</tr>
</tbody>
</table>

F = { FLD → FLDAT }

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>DOB</th>
<th>Street</th>
<th>CityState</th>
<th>Zipcode</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Sep 9 1979</td>
<td>1680 East West Rd.</td>
<td>Honolulu, HI</td>
<td>96822</td>
<td>808-343-0809</td>
</tr>
</tbody>
</table>

F = { FLD → FLDSCZT, C→Z }

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>OS</td>
<td>Mark</td>
</tr>
</tbody>
</table>

F = { SC → I, I→C }
Decompositions

• Reduces redundancies and anomalies, but could have the following potential problems:
  – Some queries become more expensive.
  – Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
  – Checking some dependencies may require joining the instances of the decomposed relations.

• Two desirable properties:
  – Lossless-join decomposition
  – Dependency-preserving decomposition
Lossless-join Decomposition

• 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:

\[ \pi_X(r) \Join \pi_Y(r) = r \]

• In general one direction \( \pi_X(r) \Join \pi_Y(r) \supseteq r \) is always true, but the other may not hold.

• 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 (2).)
Conditions for Lossless Join

• The decomposition of $R$ into $X$ and $Y$ is lossless-join wrt $F$ if and only if the closure of $F$ contains:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$

• In particular, the decomposition of $R$ into $UV$ and $R - V$ is lossless-join if $U \rightarrow V$ holds over $R$. 

\begin{align*}
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 2 & 8 \\
\end{array}
\end{align*}

\begin{align*}
\begin{array}{cc}
A & B \\
1 & 2 \\
4 & 5 \\
7 & 2 \\
\end{array}
\end{align*}

\begin{align*}
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 2 & 8 \\
1 & 2 & 8 \\
7 & 2 & 3 \\
\end{array}
\end{align*}

\begin{align*}
\begin{array}{cc}
B & C \\
2 & 3 \\
5 & 6 \\
2 & 8 \\
\end{array}
\end{align*}
Dependency-preserving Decomposition

F = \{ SC \rightarrow I, I \rightarrow C \} 

- **Dependency preserving decomposition (Intuitive):**
  - 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. *(Avoids Problem (3)).*

- **Projection of set of FDs F:** If R is decomposed into X, ... projection of F onto X (denoted \( F_X \)) is the set of FDs \( U \rightarrow V \) in \( F^+ \) *(closure of F)* such that \( U, V \) are in X.

Checking SC \( \rightarrow \) I requires a join!
Dependency-preserving Decomp. (Cont)

• 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 \(F\), 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??

• Dependency preserving does not imply lossless join:
  – ABC, \(A \rightarrow B\), decomposed into AB and BC.

• And vice-versa! (Example?)
Decomposition into BCNF

• Consider relation R with FDs F. How do we decompose R into a set of small relations that are in BCNF?

• Algorithm:
  – If X → Y violates BCNF, decompose R into R-Y and XY
  – Repeat until all relations are in BCNF.

• Example: CSJDPQV, key C, JP→C, SD→P, J→S
  – To deal with J→S, decompose CSJDPQV into JS and CJDPQV
  – To deal with SD→P, decompose into SDP, CSJDQV

• Order in which we deal with the violating FD can lead to different relations!
BCNF & Dependency Preservation

- BCNF decomposition is lossless join, but 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 CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (Redundancy!)
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).

• How can we ensure dependency preservation?
  – 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 $F$*. 
Minimum Cover for a Set of FDs

- **Minimal cover** $G$ for a set of FDs $F$:
  - Closure of $F = \text{closure of } G$.
  - 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 $EF \rightarrow H$
Computing the Minimal Cover

• Algorithm
  1. **Put the FDs into standard form** $X \rightarrow A$. RHS is a single attribute.
  2. **Minimize the LHS of each FD**. For each FD, check if we can delete an attribute from LHS while preserving $F^+$.
  3. **Delete redundant FDs**.

• Minimal covers are not unique. Different order of computation can give different covers.

• 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 $EF \rightarrow H$
Summary of Schema Refinement

• If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.

• If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  – Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  – Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.