Schema Refinement & Normalization Theory

Functional Dependencies

Week 14
Decomposition into BCNF

• Recall that for $X \rightarrow A$ in $F$ over $R$ to satisfy BCNF requirement, one of the followings must be true:
  – $XA$ are not all in $R$, or
  – $X \rightarrow A$ is trivial, i.e. $A$ is in $X$, or
  – $X$ is a superkey, i.e. $X \rightarrow R$ is in $F^+$

• Consider relation $R$ with FDs $F$. If $X \rightarrow A$ in $F$ over $R$ violates BCNF, i.e.,
  – $XA$ are all in $R$, and
  – $A$ is not in $X$, and
  – $X \rightarrow R$ is not in $F^+$
    $→$ non-trivial FD
    $→$ $X$ is not a superkey
Decomposition into BCNF

• Consider relation \( R \) with FDs \( F \). If \( X \rightarrow A \) in \( F \) over \( R \) violates BCNF, i.e.,
  – \( XA \) are all in \( R \), and
  – \( A \) is not in \( X \), and
  – \( X \rightarrow R \) is not in \( F^+ \) → non-trivial FD → \( X \) is not a (super)key

• Then: decompose \( R \) into \( R - A \) and \( XA \).
• Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
BCNF Decomposition Example

• \( R = (A, B, C) \)
  \[ F = \{A \rightarrow B; B \rightarrow C\} \]
  Key = \{A\}

• \( R \) is not in BCNF (\( B \rightarrow C \) but \( B \) is not a superkey)

• Decomposition
  – \( R_1 = (B, C) \)
  – \( R_2 = (A, B) \)
BCNF Decomposition Example 2

• Assume relation schema CSJDPQV:  
  *Contracts*(contract_id, supplier, project, dept, part, qty, value)*
  
  - key C, JP → C, SD → P, J → S
• To deal with SD → P, decompose into SDP, CSJDQV.
• To deal with J → S, decompose CSJDQV into JS and CJDQV
• A tree representation of the decomposition:

```
      CSJDPQV
     /    \      
    SDP    CSJDQV
  /        /      
 JS  CJDQV  
```

Using SD → P
Using J → S
BCNF Decomposition

- In general, several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!
How do we know R is in BCNF?

• If R has only two attributes, then it is in BCNF

• If F only uses attributes in R, then:
  – R is in BCNF if and only if for each $X \rightarrow Y$ in $F$ (not $F^+$!), $X$ is a superkey of R, i.e., $X \rightarrow R$ is in $F^+$ (not $F$!).
Checking for BCNF Violations

- List all non-trivial FDs
- Ensure that left hand side of each FD is a superkey
- We have to first find all the keys!
Checking for BCNF Violations

- Is Courses(course_num, dept_name, course_name, classroom, enrollment, student_name, address) in BCNF?
- FDs are:
  - course_num, dept_name → course_name
  - course_num, dept_name → classroom
  - course_num, dept_name → enrollment
- What is (course_num, dept_name)+?
  - {course_num, dept_name, course_name, classroom, enrollment}
- Therefore, the key is
  {course_num, dept_name, course_name, classroom, enrollment, student_name, address}
- The relation is not in BCNF
BCNF and Dependency Preservation

• In general, there may not be a dependency preserving decomposition into BCNF.
• Example: schema CSZ (city, street_name, zip_code) with FDs: CS → Z, Z → C
  
  (city, street_name) → zip_code
  
  zip_code → city

• Can’t decompose while *preserving* CS → Z, but CSZ is not in BCNF.
Example Regarding Dependency Preservation

- \( R = (A, B, C) \)
  \[ F = \{ A \rightarrow B, B \rightarrow C \} \]
  - Can be decomposed in two different ways
- \( R_1 = (A, B), \ R_2 = (B, C) \)
  - Lossless-join decomposition:
    \[ R_1 \cap R_2 = \{ B \} \text{ and } B \rightarrow BC \]
  - Dependency preserving
- \( R_1 = (A, B), \ R_2 = (A, C) \)
  - Lossless-join decomposition:
    \[ R_1 \cap R_2 = \{ A \} \text{ and } A \rightarrow AB \]
  - Not dependency preserving
    (cannot check \( B \rightarrow C \) without computing \( R_1 \bowtie R_2 \))
Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CSJDPQV and SDP
  - Problem: Checking JP → C requires a join!

- **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)*)
What FD on a decomposition?

• **Projection of set of FDs F**: If R is decomposed into X, ... the 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 $X$. 
Dependency Preserving Decompositions (Contd.)

- 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!
Another example

• Assume CSJDPQV is decomposed into
  SDP, JS, CJDQV
  It 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!
Summary of BCNF

• 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.
  – It is always possible to decompose a relation into a set of relations that are in BCNF such that:
    • the decomposition is lossless
    • it may not be possible to preserve dependencies.
Next: Third Normal Form

• There are some situations where
  – BCNF is not dependency preserving, and
  – efficient checking for FD violation on updates is important

• Solution: define a weaker normal form, called Third Normal Form (3NF)
  – Allows some redundancy (with resultant problems; we will see examples later)
  – But functional dependencies can be checked on individual relations without computing a join.
  – There is always a lossless-join, dependency-preserving decomposition into 3NF.
Third Normal Form (3NF)

• 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” decomposition, or performance considerations).
  – Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.
3NF

- Relation R with FDs F is in 3NF if, for each FD $X \rightarrow A$ ($X \in R$ and $A \in R$) in F, one of the following statements is true:
  - $A \in X$ (trivial FD), or
  - $X$ is a superkey, or
  - $A$ is part of some key for R

Not just superkey! (why not?)

If one of these two is satisfied for ALL FDs, then R is in BCNF
What Does 3NF Achieve?

• If 3NF is violated by $X \rightarrow A$, one of the following holds:
  - $X$ is a subset of some key $K$ (partial redundancy)
    • We store $(X, A)$ pairs redundantly.
  - $X$ 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.

• **But:** even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC (sid, bid, date, credit_card). Keys are SBD, CBD. FD = \{$S \rightarrow C$, $C \rightarrow S$\}. $R$ is in 3NF, but for each reservation of sailor $S$, same $(S, C)$ pair is stored.

• Thus, 3NF is indeed a compromise relative to BCNF.
Decomposition into 3NF

• Obviously, the algorithm for lossless join decom into BCNF can be used to obtain a lossless join decom into 3NF (typically, can stop earlier).

• To ensure dependency preservation, one idea:
  – If $X \rightarrow Y$ is not preserved, add relation $XY$.
  – Problem is that $XY$ may violate 3NF!

• Refinement: Instead of the given set of FDs $F$, use a \textit{minimal cover for $F$}. 
Minimal 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 $F$. 
Obtaining Minimal Cover

• Step 1: Put the FDs in a standard form (i.e. right-hand side should contain only single attribute)
• Step 2: Minimize the left side of each FD
• Step 3: Delete redundant FDs
• Find minimal cover for $F = \{ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow AD, E \rightarrow L, BH \rightarrow E\}$
• Step 1: Make RHS of each FD into a single attribute:

\[ F = \{ ABH \rightarrow C, ABH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E \} \]
• \( F = \{ABH \rightarrow C, ABH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\} \)

• **Step 2: Eliminate redundant attributes from LHS, e.g. Can an attribute be deleted from \( ABH \rightarrow C \)?**
  - Compute \((AB)^+, (BH)^+, (AH)^+\) and see if any of them contains \( C \). (Why?)

  - \((AB)^+ = ABD, (BH)^+ = ABCDEHKL, (AH)^+ = ADH\). Since \( C \in (BH)^+ \), \( BH \rightarrow C \) is entailed by \( F \). So \( A \) is redundant in \( ABH \rightarrow C \). Similarly, \( A \) is also redundant in \( ABH \rightarrow K \). Check further to see if \( B \) or \( H \) is redundant as well.

  - Similarly, for \( BGH \rightarrow L \), \( G \) is redundant since \( L \in (BH)^+ \).

  - \( F = \{BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, BH \rightarrow L, L \rightarrow A, L \rightarrow D, E \rightarrow L, BH \rightarrow E\} \)
• $F = \{\text{BH} \rightarrow \text{C}, \text{BH} \rightarrow \text{K}, \text{A} \rightarrow \text{D}, \text{C} \rightarrow \text{E}, \text{BH} \rightarrow \text{L}, \text{L} \rightarrow \text{A}, \text{L} \rightarrow \text{D}, \text{E} \rightarrow \text{L}, \text{BH} \rightarrow \text{E}\}$

• **Step 3: Delete redundant FDs from F.**

  – If $F - \{f\}$ infers $f$, then $f$ is redundant, i.e. if $f$ is $X \rightarrow A$, then check if $X^+$ using $F - f$ still contains $A$. If it does, then it means $X \rightarrow A$ can be inferred by other FDs.
  – E.g. For $\text{BH} \rightarrow \text{L}$, $(\text{BH})^+$ (not using $\text{BH} \rightarrow \text{L}$) = ACDEKL, which contains $L$. This means $\text{BH} \rightarrow \text{L}$ can be inferred by other FDs, so it’s a redundant FD.
  – In fact, $\text{BH} \rightarrow \text{L}$ can be inferred by $\text{BH} \rightarrow \text{E}$, $\text{E} \rightarrow \text{L}$.
  – Check other FDs using the same algorithm.

• **Note:** The order of Step 2 and Step 3 should not be exchanged.
What to do with Minimal Cover?

• After obtaining the minimal cover, for each FD $X \rightarrow A$ in the minimal cover that is not preserved, create a table consisting of $XA$ (so we can check dependency in this new table, i.e. dependency is preserved).

• Why is this new table guaranteed to be in 3NF (whereas if we created the new table from $F$, it might not?)
  – Since $X \rightarrow A$ is in the minimal cover, $Y \rightarrow A$ does not hold for any $Y$ that is a strict subset of $X$.
    • So $X$ is a key for $XA$ (satisfies condition #2)
    • If any other dependencies hold over $XA$, the right side can involve only attributes in $X$ because $A$ is a single attribute (satisfies condition #3).
Comparison of BCNF and 3NF

• It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  – the decomposition is lossless
  – the dependencies are preserved

• It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  – the decomposition is lossless
  – it may not be possible to preserve dependencies.
Normalization Review

- Identify all FD's in $F^+$
- Identify candidate keys
- Identify (strongest, or specific) normal forms
  - BCNF, 3NF
- Schema decomposition
  - When to decompose
  - How to check if a decomposition is lossless-join and/or dependency preserving
    - Use projection of $F^+$ to check for dependency preservation
  - Decompose into:
    - Lossless-join
    - Dependency preserving
      - Use minimal cover
Normalization Theory - Practice Questions
**Example**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FDs with A as the left side:</th>
<th>Satisfied by the relation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow A$</td>
<td>Yes (trivial FD)</td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
<td>Yes</td>
</tr>
<tr>
<td>$A \rightarrow C$</td>
<td>No: tuples 1&amp;2</td>
</tr>
<tr>
<td>$AB \rightarrow A$</td>
<td>Yes (trivial FD)</td>
</tr>
<tr>
<td>$AC \rightarrow B$</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example

Let $F = \{ A \rightarrow BC, B \rightarrow C \}$. Is $C \rightarrow AB$ in $F^+$?

Answer: No. Either of the following 2 reasons is ok:

Reason 1) $C^+ = C$, and does not include $AB$.

Reason 2) We can find a relation instance such that it satisfies $F$ but does not satisfy $C \rightarrow AB$.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
List all the **non-trivial** FDs in $F^+$

- Given $F=\{A \rightarrow B, B \rightarrow C\}$. Compute $F^+$ (with attributes A, B, C).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>AB</th>
<th>AC</th>
<th>BC</th>
<th>ABC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>B</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attribute closure

- $A^+=ABC$
- $B^+=BC$
- $C^+=C$
- $AB^+=ABC$
- $AC^+=ABC$
- $BC^+=BC$
- $ABC^+=ABC$
Example

• Given $F=\{ A \rightarrow B, B \rightarrow C \}$. Find a relation that satisfies $F$:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

• Given $F=\{ A \rightarrow B, B \rightarrow C \}$. Find a relation that satisfies $F$ but does not satisfy $B \rightarrow A$. Well, the above example suffices.

• Can you find an instance that satisfies $F$ but not $A \rightarrow C$? No. Because $A \rightarrow C$ is in $F^+$.
Examples

R(A, B, C, D, E),
F = \{A \rightarrow B, C \rightarrow D\}

Candidate key: ACE. How do we know?

Intuitively,
- A is not determined by any other attributes (like E),
  and A has to be in a candidate key (because a candidate key has to determine all the attributes).
- Now if A is in a candidate key, B cannot be in the same candidate key, since we can drop B from the candidate without losing the property of being a “key”.
- So B cannot be in a candidate key
- Same reasoning apply to others attributes.
Example

R(A, B, C, D, E),
F = \{A \rightarrow B, C \rightarrow D\} [Same as previous]

Which normal form?

Not in BCNF. This is the case where all attributes in the FDs appear in R. We consider A, and C to see if either is a superkey of not. Obviously, neither A nor C is a superkey, and hence R is not in BCNF. More precisely, we have A \rightarrow B is in F^{+} and non-trivial, but A is not a superkey of R.
Example

R(A, B, C, D, E)
F = \{A \rightarrow B, C \rightarrow D\} [Same as previous]

Which normal form?

We already know that it’s not in BCNF.
Not in 3NF either. We have A \rightarrow B is in F^+ and non-trivial, but A is not a superkey of R. Furthermore, B is not in any candidate key (since the only candidate key is ACE).
Example

- $R(A,B,F)$, $F = \{AC \rightarrow E, B \rightarrow F\}$.
- Candidate key? AB
- BCNF? No, because of $B \rightarrow F$ (B is not a superkey).
- 3NF? No, because of $B \rightarrow F$ (F is not part of a candidate key).
Example

- $R(\text{D, C, H, G})$, $F = \{A \rightarrow I, I \rightarrow A\}$
- Candidate key? DCHG
- BCNF? Yes
- 3NF? Yes
Example

- \( R(A, B, C, D, E, G, H) \)
  \[ F = \{AB \rightarrow C, AC \rightarrow B, B \rightarrow D, BC \rightarrow A, E \rightarrow G\} \]

- Candidate keys?
  - H has to be in all candidate keys
  - E has to be in all candidate keys
  - G cannot be in any candidate key (since E is in all candidate keys already).
  - Since \( AB \rightarrow C, AC \rightarrow B \) and \( BC \rightarrow A \), we know no candidate key can have ABC together.
  - AEH, BEH, CEH are not superkeys.
  - Try ABEH, ACEH, BCEH. They are all superkeys. And we know they are all candidate keys (since above properties)
  - These are the only candidate keys: (1) each candidate key either contains A, or B, or C since no attributes other than A, B, C determine A, B, C, and (2) if a candidate key contains A, then it must contain either B, or C, and so on.
Example

- Same as previous
- Not in BCNF, not in 3NF
- Decomposition:

R(A, B, C, D, E, G, H)
F={AB → C, AC → B, B → D, BC → A, E → G}
Example

- $R(A, B, C, D, E, G, H)$
  
  $F = \{AB \rightarrow C, AC \rightarrow B, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$

- Decomposition: BD, ABC, EG, ABEH

- Why good decomposition?
  - They are all in BCNF
  - Lossless-join decomposition
    - How do you know this if you don’t know how R was decomposed?
    - All dependencies are preserved.
Example

• R(A, B, D, E) decomposed into R1(A, B, D), R2(A, B, E)
• F={AB → DE}
• It is a dependency preserving decomposition!
  – AB → D can be checked in R1
  – AB → E can be checked in R2
  – {AB → DE} is equivalent to {AB → D, AB → E}