Using BCNF and 3NF
Boyce-Codd Normal Form

- A relation schema \( R \) is in BCNF if for all functional dependencies in \( F^+ \) of the form \( \alpha \rightarrow \beta \) at least one of the following holds:
  - \( \alpha \rightarrow \beta \) is trivial (i.e., \( \beta \subseteq \alpha \))
  - \( \alpha \) is a superkey for \( R \)

- \( \text{bor_loan} = (\text{customer_id}, \text{loan_number}, \text{amount}) \) is not in BCNF
  - \( \text{loan_number} \rightarrow \text{amount} \) holds on \( \text{bor_loan} \) but \( \text{loan_number} \) is not a superkey
Decomposing into BCNF

- For schema R and a non-trivial dependency \( \alpha \rightarrow \beta \) causes a violation of BCNF, decompose R into
  - \((\alpha \cup \beta)\): \(\alpha\) is the key
  - \((R - (\beta - \alpha))\)

- \(bor\_loan = (customer\_id, loan\_number, amount), loan\_number \rightarrow amount\)
  - (loan\_number, amount)
  - (customer\_id, loan\_number)
Third Normal Form

- A relation schema $R$ is in the third normal form (3NF) if for all $\alpha \rightarrow \beta$ in $F^+$ at least one of the following holds
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subset \alpha$)
  - $\alpha$ is a superkey for $R$
  - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$
    - Each attribute may be in a different candidate key
- If a relation is in BCNF it is in 3NF
  - In BCNF one of the first two conditions above must hold
- The third condition is a minimal relaxation of BCNF to ensure dependency preservation
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 and 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 all functional dependencies.
Using BCNF and 3NF

• How can we generate lossless decompositions into BCNF and 3NF?
• How can we test if a decomposition is dependency-preserving?
• Some critical operations
  – If $\alpha \rightarrow \beta$, whether $\alpha$ is a superkey?
  – To preserve all functional dependencies $F$, what is the minimal set of functional dependencies that we need to preserve?
Closure of Attribute Sets

• Given a set of attributes $\alpha$, the closure of $\alpha$ under $F$ (denoted by $\alpha^+$) is the set of attributes that are functionally determined by $\alpha$ under $F$
  – Application: whether $\alpha$ is a super key?

• Algorithm to compute $\alpha^+$, the closure of $\alpha$ under $F$

\[
\text{result} := \alpha; \\
\text{while (changes to result) do} \\
\quad \text{for each } \beta \rightarrow \gamma \text{ in } F \text{ do} \\
\quad \quad \text{begin} \\
\quad \quad \quad \text{if } \beta \subseteq \text{result} \text{ then } \text{result} := \text{result} \cup \gamma \\
\quad \quad \text{end}
\]
Canonical Cover

- A canonical cover for F is a set of dependencies Fc such that
  - F logically implies all dependencies in Fc
  - Fc logically implies all dependencies in F
  - No functional dependency in Fc contains an extraneous attribute, and
  - Each left side of functional dependency in Fc is unique
Canonical Cover Computation

repeat
  use the union rule to replace any dependencies in F
  \( \alpha_1 \rightarrow \beta_1 \) and \( \alpha_1 \rightarrow \beta_2 \) with \( \alpha_1 \rightarrow \beta_1 \beta_2 \)
  find a functional dependency \( \alpha \rightarrow \beta \) with an extraneous attribute either in \( \alpha \) or in \( \beta \)
  if an extraneous attribute is found, then delete it from \( \alpha \rightarrow \beta \)
  until F does not change

• The union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied
Lossless-join Decomposition

- If $R$ is decomposed into $R_1$ and $R_2$, we require that for all possible relations $r$ on schema $R$ satisfies $r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$

- A decomposition of $R$ into $R_1$ and $R_2$ is lossless join if and only if at least one of the following dependencies is in $F^+$
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
Example

• \( R = (A, B, C), F = \{ A \rightarrow B, B \rightarrow C \} \)
  – 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 \( R_1 \Join R_2 \))
Dependency Preservation

• Let $F_i$ be the set of dependencies $F^+$ that includes only attributes in $R_i$
  – A decomposition is dependency preserving, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  – If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive
Testing Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of $R$ into $R_1, R_2, \ldots, R_n$
  
  $\text{result} = \alpha$

  repeat
  
  for each $R_i$ in the decomposition
  
  $t = (\text{result} \cap R_i)^+ \cap R_i$, $\text{result} = \text{result} \cup t$

  until $\text{result}$ does not change

  - If $\text{result}$ contains all attributes in $\beta$, then the functional dependency $\alpha \rightarrow \beta$ is preserved

- We apply the test on all dependencies in $F$ to check if a decomposition is dependency preserving

- This procedure takes polynomial time
  
  - Computing $F^+$ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$ requires exponential time
Example

- \( R = (A, B, C) \)
  \( F = \{A \rightarrow B, B \rightarrow C\} \)
  \( \text{Key} = \{A\} \)

- \( R \) is not in BCNF

- Decomposition \( R_1 = (A, B), \ R_2 = (B, C) \)
  - \( R_1 \) and \( R_2 \) are in BCNF
  - Lossless-join decomposition
  - Dependency preserving
Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  - Compute $\alpha^+$ (the attribute closure of $\alpha$), and
  - Verify that it includes all attributes of R, that is, it is a superkey of R

- Simplified test: it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in $F^+$
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in $F^+$ will cause a violation of BCNF either
Testing for Decomposition to BCNF

- Using only $F$ is insufficient when testing a relation in a decomposition of $R$
- Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D \}$
  - Decompose $R$ into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
  - Neither of the dependencies in $F$ contain only attributes from $(A,C,D,E)$ so we might be misled into thinking $R_2$ satisfies BCNF
  - In fact, dependency $AC \rightarrow D$ in $F^+$ shows $R_2$ is not in BCNF
Testing Decomposition to BCNF

• To check if a relation $R_i$ in a decomposition of $R$ is in BCNF, we can test $R_i$ for BCNF with respect to the restriction of $F$ to $R_i$ (that is, all FDs in $F^+$ that contain only attributes from $R_i$)
Another Method

- Use the original set of dependencies $F$ that hold on $R$, but with the following test
  - For every set of attributes $\alpha \subseteq R_i$, check that $\alpha^+$ (the attribute closure of $\alpha$) either includes no attribute of $R_i - \alpha$, or includes all attributes of $R_i$.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in $F$, the dependency $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ can be shown to hold on $R_i$, and $R_i$ violates BCNF.
  - We use the above dependency to decompose $R_i$. 

BCNF Decomposition Algorithm

\[
\text{result} := \{ R \}; \\
\text{done} := \text{false}; \\
\text{compute } F^+; \\
\text{while (not done) do} \\
\quad \text{if (there is a schema } R_i \text{ in result that is not in BCNF)} \\
\quad \quad \text{then begin} \\
\quad \quad \quad \text{let } \alpha \rightarrow \beta \text{ be a nontrivial functional dependency} \\
\quad \quad \quad \text{that holds on } R_i \text{ such that } \alpha \rightarrow R_i \text{ is not in } F^+, \text{ and} \\
\quad \quad \quad \alpha \cap \beta = \emptyset; \quad \text{}
\quad \quad \quad \text{result} := (\text{result} - R_i) \cup (R_i - \beta) \cup (\alpha, \beta); \\
\quad \quad \text{end} \\
\quad \text{else done} := \text{true}; \\
\text{Note: each } R_i \text{ is in BCNF, and decomposition is lossless-join}
\]
Example of BCNF Decomposition

- \( R = (A, B, C) \)
  \( F = \{A \rightarrow B, B \rightarrow C\} \)
  \( \text{Key} = \{A\} \)

- \( R \) is not in BCNF \((B \rightarrow C \text{ but } B \text{ is not a superkey})\)

- Decomposition
  - \( R_1 = (B, C) \)
  - \( R_2 = (A, B) \)
Example of BCNF Decomposition

• Original relation $R$ and functional dependency $F$
  
  $R = (\text{branch\_name, branch\_city, assets, customer\_name, loan\_number, amount})$
  
  $F = \{\text{branch\_name} \rightarrow \text{assets branch\_city} \}
  \text{loan\_number} \rightarrow \text{amount branch\_name}\}
  
  Key = \{\text{loan\_number, customer\_name}\}$

• Decomposition
  
  – $R_1 = (\text{branch\_name, branch\_city, assets})$
  
  – $R_2 = (\text{branch\_name, customer\_name, loan\_number, amount})$
  
  – $R_3 = (\text{branch\_name, loan\_number, amount})$
  
  – $R_4 = (\text{customer\_name, loan\_number})$

• Final decomposition $R_1, R_3, R_4$
Testing for 3NF

- Check only functional dependencies in $F$, do not need to check all functional dependencies in $F^+$
- Follow the definition of 3NF
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$ whether $\alpha$ is a superkey
- If $\alpha$ is not a superkey, we have to verify if each attribute in $\beta$ is contained in a candidate key of $R$
  - Rather expensive, finding candidate keys
  - Testing for 3NF is NP-hard (likely cannot be done in polynomial time)
3NF Decomposition Algorithm

Let $F_c$ be a canonical cover for $F$;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in $F_c$ do
  if none of the schemas $R_j, 1 \leq j \leq i$ contains $\alpha \beta$
    then $i := i + 1; R_i := \alpha \beta$
  if none of the schemas $R_j, 1 \leq j \leq i$ contains a candidate key for $R$
    then $i := i + 1; R_i :=$ any candidate key for $R$;
return $(R_1, R_2, ..., R_i)$
3NF Decomposition

- **Banker-info-schema = \((\text{branch-name, customer-name, banker-name, office-number})\)**
  - R1: \(\text{banker-name} \rightarrow \text{branch-name office-number}\)
  - R2: \(\text{customer-name branch-name} \rightarrow \text{banker-name}\)
  - The key: \(\{\text{customer-name, branch-name}\}\)

- **Decomposition**
  - **Banker-office-schema = \((\text{banker-name, branch-name, office-number})\)**
  - **Banker-schema = \((\text{customer-name, branch-name, banker-name})\)**
3NF Decomposition

- Each relation schema $R_i$ is in 3NF
- Decomposition is functional dependency preserving and lossless-join
- The 3NF synthesis algorithm can be implemented in polynomial time
  - Testing for 3NF is NP-hard
Summary and To-Do List

• Testing for BCNF and 3NF
• Decomposition to BCNF and 3NF
• Questions in Assignment 2