CMSC 424 – Database design
Lecture 11
Normalization

Mihai Pop
The Normal Forms

- **1NF**: every attribute has an atomic value (not a set value)

- **2NF**: we will not be concerned in this course

- **3NF**: if for each FD $X \rightarrow Y$ either
  - it is trivial or
  - $X$ is a superkey
  - $Y-X$ is a proper subset of a candidate key

- **BCNF**: if for each FD $X \rightarrow Y$ either
  - it is trivial or
  - $X$ is a superkey

- **4NF, ...**: we are not concerned in this course.
Goals

• Lossless decomposition

• Dependency preservation

• Recap: FD closure, attribute closure
FDs, Normal forms, etc..., why?

• Start with a schema
• Decompose relations until in a normal form
• Functional dependencies (constraints we'd like preserved) drive the decomposition
• The resulting schema is “better”

• Note that functional dependencies can either be:
  – explicit: we want to enforce these constraints irrespective of data in the relations – can be encoded in SQL
  – implicit: the data happen to satisfy them (see netflix example)

Normalization only concerned with explicit FDs
Privacy/anonymization – need to worry about implicit FDs
Boyce-Codd Normal Form

A relation schema $R$ is in BCNF with respect to a set $F$ of functional dependencies if for all functional dependencies in $F^+$ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e. $\beta \subseteq \alpha$)
- $\alpha$ is a superkey for $R$, i.e. $\alpha + = R$

Example schema not in BCNF:

bor_loan = ( customer_id, loan_number, amount )

because loan_number $\rightarrow$ amount holds on bor_loan but loan_number is not a superkey
Decomposing a Schema into BCNF

- Suppose we have a schema $R$ and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

We decompose $R$ into:

\[
\begin{align*}
\alpha \cup \beta \\
R - (\beta - \alpha)
\end{align*}
\]

- In our example,
  - $\alpha = \text{loan\_number}$
  - $\beta = \text{amount}$

  and $\text{bor\_loan}$ is replaced by
  - $\alpha \cup \beta = (\text{loan\_number}, \text{amount})$
  - $R - (\beta - \alpha) = (\text{customer\_id}, \text{loan\_number})$
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: To check if a relation schema $R$ with a given set of functional dependencies $F$ is in BCNF, it suffices to check only the dependencies in the given set $F$ for violation of BCNF, rather than checking all dependencies in $F^+$.  
  - We can show that 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 BCNF...cont

- However, using only F is incorrect when testing a relation in a decomposition of R
- E.g. Consider R (A, B, C, D), with F = { A → B, B → C}
  - Decompose R into R₁(A,B) and R₂(A,C,D)
  - Neither of the dependencies in F contain only attributes from (A,C,D) so we might be mislead into thinking R₂ satisfies BCNF.
  - In fact, dependency A → C in F⁺ shows R₂ is not in BCNF.

- Simplified test: Avoids computing F⁺
  - For every subset α of Rᵢ compute α⁺ under F
  - Then either α⁺ includes no attributes of Rᵢ-α or includes all attributes of Rᵢ
    - In R₂(A,C,D) above A⁺=ABC, A⁺-(A)=(BC) includes an attribute of Rᵢ but not all (violation)
    - Then α → (α⁺ - α)∩ Rᵢ is the violator A→BC ∩ (ACD)=C is an FD (actually in F⁺) which violates BCNF
BCNF Decomposition Algorithm

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

Note: each \( R_i \) is in BCNF, and decomposition is lossless-join.
Example of BCNF Decomposition

- \( R = (\text{branch-name, branch-city, assets, customer-name, loan-number, amount}) \)
- \( F = (\text{branch-name} \rightarrow \text{assets branch-city loan-number} \rightarrow \text{amount branch-name}) \)
  
  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
  \( R_1, R_3, R_4 \)
BCNF and Dependency Preservation

• Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation.

• If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that all functional dependencies hold, then that decomposition is dependency preserving.

• Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as third normal form.
Third Normal Form

- A relation schema \( R \) is in 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., \( \beta \in \alpha \))
  - \( \alpha \) is a superkey for \( R \)
  - Each attribute \( A \) in \( \beta \) – \( \alpha \) is contained in a candidate key for \( R \).

  (NOTE: each attribute may be in a different candidate key)

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must must hold).

- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).
3NF (Cont.)

- Example
  - \( R = (J, K, L) \)
    \( F = \{JK \rightarrow L, L \rightarrow K\} \)
  
  - Two candidate keys: \( JK \) and \( JL \)

- \( R \) is in 3NF
  
  \( JK \rightarrow L \quad JK \) is a superkey
  
  \( L \rightarrow K \quad K \) is contained in a candidate key
Redundancy in 3NF

- Example of problems due to redundancy in 3NF
  - \( R = (J, K, L) \)
  - \( F = \{JK \rightarrow L, L \rightarrow K\} \)

<table>
<thead>
<tr>
<th></th>
<th>J</th>
<th>L</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( j_1 )</td>
<td>( l_1 )</td>
<td>( k_1 )</td>
</tr>
<tr>
<td>2</td>
<td>( j_2 )</td>
<td>( l_1 )</td>
<td>( k_1 )</td>
</tr>
<tr>
<td>3</td>
<td>( j_3 )</td>
<td>( l_1 )</td>
<td>( k_1 )</td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>( l_2 )</td>
<td>( k_2 )</td>
</tr>
</tbody>
</table>

A schema in 3NF but not in BCNF has the following problems:
- redundancy of information
- need to use null values (e.g. to represent relationship \( l_2 k_2 \), when there is no corresponding \( j \) value)
Testing for 3NF

• Optimization: Need to check only FDs in $F$, need not check all FDs in $F^+$.  

• Use attribute closure to check, for each dependency $\alpha \rightarrow \beta$, if $\alpha$ 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 $R$ 
  – this test is more expensive, since it involve finding ALL candidate keys 
  – testing for 3NF has been shown to be NP-hard
Canonical Cover

• Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  – For example: \( A \rightarrow C \) is redundant in: \( \{ A \rightarrow B, \ B \rightarrow C \} \)
  – Parts of a functional dependency may be redundant
    • E.g.: on RHS: \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow CD \} \) can be simplified to
      \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D \} \)
    • E.g.: on LHS: \( \{ A \rightarrow B, \ B \rightarrow C, \ AC \rightarrow D \} \) can be simplified to
      \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D \} \)

• Intuitively, a canonical cover of \( F \) is a “minimal” set of functional dependencies equivalent to \( F \), having no redundant dependencies or redundant parts of dependencies
Extraneous Attributes

• Consider a set $F$ of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in $F$.
  – Attribute $A$ is **extraneous** in $\alpha$ if $A \in \alpha$ and $F$ logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  – Attribute $A$ is **extraneous** in $\beta$ if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies $F$.

• **Note:** implication in the opposite direction is trivial in each of the cases above, since a “stronger” functional dependency always implies a weaker one.
• Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  – $B$ is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (i.e. the result of dropping $B$ from $AB \rightarrow C$).
• Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  – $C$ is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting $C$. 

3NF Decomposition/“construction” 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 begin
      $i := i + 1$;
      $R_i := \alpha \beta$
    end
  if none of the schemas $R_j$, $1 \leq j \leq i$ contains a candidate key for $R$
    then begin
      $i := i + 1$;
      $R_i := \text{any candidate key for } R$
    end
return $(R_1, R_2, ..., R_i)$
Comparison of BCNF and 3NF

• It is always possible to decompose a relation into relations in 3NF and
  – the decomposition is lossless
  – the dependencies are preserved

• It is always possible to decompose a relation into relations in BCNF and
  – the decomposition is lossless
  – it may not be possible to preserve dependencies.
More Examples

• SUPPLY(sno,pno,jno,scity,jcity,qty)  
  – sno,pno,jno is the candidate key,  
  – sno → scity, jno= → jcity

• ED(eno,ename,byr,sal,dno,dname,floor,mgr)  
  – eno → dno → mgr

• TEACH(student,teacher,subject)  
  – student,subject → teacher  
  – teacher → subject
Normalization Using FDs

Check whether a particular relation $R$ is in “good” form: BCNF or 3NF

If not, decompose $R$ into a set of relations \( \{R_1, R_2, ..., R_n\} \) such that

- No redundancy: The relations $R_i$ preferably should be in either Boyce-Codd Normal Form or Third Normal Form.

- Lossless-join decomposition: Otherwise you have information loss.

- Dependency preservation: Let $F_i$ be the set of dependencies $F^+$ that include only attributes in $R_i$.
  - Preferably the decomposition should be dependency preserving, that is, \( (F_1 \cup F_2 \cup ... \cup F_n)^+ = F^+ \)
  - Otherwise, checking during updates for violation of functional dependencies may require expensive joins operations

- The theory is based on functional dependencies
BCNF and Over-normalization

- 3NF relation has redundancy anomalies: \text{TEACH}(\text{student}, \text{teacher}, \text{subject})
  - insertion: cannot insert a teacher until we had a student taking his subject
  - deletion: if I delete the last student of a teacher, then I loose the subject he teaches

- What is really the problem? schema \textit{overload}. We are trying to capture two meanings:
  1. subject X is (or can be) taught by teacher Y
  2. student Z takes subject W from teacher V

- it makes no sense to say we loose the subject he teaches when he does not have a student! Who does he teach to?

- normalizing it to BCNF cannot preserve dependencies. Therefore, it is better to stay with the 3NF \text{TEACH} and another relation \text{SUBJECT_TAUGHT}:

\begin{align*}
\text{TEACH}(\text{student}, \text{teacher}, \text{subject}) & \quad \text{3NF} \\
\text{SUBJECT-TAUGHT}(\text{teacher}, \text{subject}) & \quad \text{BCNF}
\end{align*}
Summary...practical issues

- **Normalization**
  - Create a good schema – low redundancy, no loss of information

- **Functional dependencies**
  - Specify constraints that must be encoded in our schema
  - Note: SQL does not allow us to specify FDs other than key constraints (PRIMARY KEY, UNIQUE)

- **Typical design process:**
  - Decompose to BCNF
  - Use materialized views to preserve any additional FDs