Schema Refinement and Normal Forms

Yanlei Diao
UMass Amherst
Consider an Example

- Consider relation obtained from **Hourly_Emps**:
  - **Hourly_Emps**(ssn, name, lot, rating, hrly_wages, hrs_worked)
  - Denote the schema by listing all its attributes: SNLRWH
Example (Contd.)

<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>

- Rating (R) determines hourly wages (W):
  - **Redundant storage**
  - **Update**: Can we change W in just the 1st tuple of rating 8?
  - **Insertion**: Insert an employee without knowing the hourly wage for his rating? Insert the hourly wage for rating 10 with no employee?
  - **Deletion**: Delete all employees with rating 5.
# Will Two Smaller Tables be Better?

<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_Emps2**

<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>

**Wages**

<table>
<thead>
<tr>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
The Evils of Redundancy

- Redundant storage causes several operation anomalies:
  - Insert/delete/update anomalies

- **Functional dependencies**, a new type of integrity constraint, can be used to identify schemas with such problems.
  - IC’s we have seen: *attribute constraints, key constraints, foreign key constraints, general constraints*
  - A new type of IC: *functional dependencies*
Functional Dependencies (FDs)

- A *functional dependency* $X \rightarrow Y$ holds over relation $R$ if:
  - $X$ and $Y$ are two sets of attributes of $R$;
  - $\forall$ allowable instance $r$ of $R$:
    
    $$t_1 \in r, t_2 \in r, \pi_X(t_1) = \pi_X(t_2) \implies \pi_Y(t_1) = \pi_Y(t_2)$$

- An FD holds for all allowable instances of a schema.

- Key constraint is a special form of FD:
  - $K$ is a candidate key for $R$ means that $K \rightarrow R$.
  - $K \rightarrow R$ does not require $K$ to be *minimal*. 

FDs in the Hourly_Emps Example

- **Hourly_Emps**\((ssn, name, office, rating, hrly\_wages, hrs\_worked)\)
  - Denoted by SNLRWH

- Some FDs on Hourly_Emps:
  - \(ssn\) is the key: \(S \rightarrow SNLRWH\)
  - \(rating\) determines \(hrly\_wages\): \(R \rightarrow W\)
Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - \( \text{ssn} \rightarrow \text{did}, \ \text{did} \rightarrow \text{building} \) implies \( \text{ssn} \rightarrow \text{building} \)

- Given a set of FDs \( F \), \textit{closure of} \( F \) (\( F^+ \)) is the set of all FDs that are implied by \( F \).
  - All FDs in \( F^+ \) hold over the relation \( R \).
Axioms and Rules

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

- A few additional rules (that follow from AA):
  - **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - **Decomposition**: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

- Computing the closure $F^+$ using the axioms/rules:
  - Compute for all FD’s.
  - Size of closure is exponential in number of attrs!
Attribute Closure

- What if we just want to check if a given FD $X \rightarrow Y$ is in $F^+$?
- Simple algorithm for attribute closure $X^+$:
  - $X^+ := \{X\}$
  - DO if there is $U \rightarrow V$ in $F$, s.t. $U \subseteq X^+$,
    then $X^+ = X^+ \cup V$
  UNTIL no change
- Check if a given FD $X \rightarrow Y$ is in $F^+$:
  - Simply check if $Y \subseteq X^+$.
- Does $F = \{A \rightarrow B, B \rightarrow C, \ C D \rightarrow E\}$ imply $A \rightarrow E$?
  - Is $A \rightarrow E$ in the closure $F^+$?
  - Equivalently, is $E$ in $A^+$?
Normal Forms

- Role of **FDs** in detecting **redundancy**: R(A,B,C)
  
  - **No FDs hold**: No redundancy here.
  
  - **Given A → B**: Two tuples have the same A value will have the same B value!

- **Normal forms**: If a reln does not have certain kinds of FDs, certain **redundancy-related problems** are known not to occur.
Boyce-Codd Normal Form (BCNF)

- Rewrite every FD in the form of $X \rightarrow A$, $X$ is a set of attributes, $A$ is a **single** attribute
  - Use the decomposition rule

- Reln R with FDs $F$ is in **BCNF** if $\forall X \rightarrow A$ in $F^+$:
  1. $A \in X$ (called a **trivial** FD), or
  2. $X$ is a **superkey** (i.e., contains a key) for R.

- In BCNF, the only non-trivial FDs are key constraints!
Can we infer the value marked by ‘?’?
- If \( X \rightarrow A \), then the relation is not in BCNF
- A reln in BCNF can’t have \( X \rightarrow A \)

Relation in BCNF:
- Every field of every tuple records information that can’t be inferred using FD’s from other fields.
- *No redundancy can be detected using FDs!*
Decomposing a Relation Scheme

- A *decomposition* of $R$ breaks $R$ into two or more relns s.t.
  - Each new reln contains a subset of the attributes of $R$.
  - Every attribute of $R$ appears in at least one new reln.

- Decompositions should be used only when:
  - $R$ has redundancy related problems (not in BCNF),
  - We can afford the joins in queries later.
Example Decomposition

- Hourly_Emps (SNLRWH)
  - FDs: \( S \rightarrow SNLRWH \) and \( R \rightarrow W \).
  - \( R \rightarrow W \) violates BCNF (3NF).
  - And it causes repeated (R,W) storage.

- To fix this, create a relation RW, remove W from the main schema. (SNLRWH) \( \rightarrow \) (SNLRH) and (RW).
Lossless Join Decompositions

- Decomposition of R into R1 and R2 is lossless-join w.r.t. a set of FDs F if \( \forall \) instance \( r \) that satisfies F:
  - \( r = \pi_{R1} (r) \bowtie \pi_{R2} (r) \)
- It is always true that \( r \subseteq \pi_{R1} (r) \bowtie \pi_{R2} (r) \).
  - A bad decomposition can cause \( r \subset \pi_{R1} (r) \bowtie \pi_{R2} (r) \).
A Simple Test for Lossless Join

- Decomposition of R into R1 and R2 is *lossless-join* wrt F iff the $F^+$ contains:
  - $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$
  - Intersection of R1 and R2 is a (super) key of one of them.

- How to apply this result?
  - If $U \rightarrow V$ holds over R and violates a BCNF definition, the decomposition into $UV$ and $R - V$ is lossless-join.
Dependency Preserving Decomposition

- **Contracts**\((Contractid, Supplierid, Projectid, Deptid, Partid, Qty, Value)\), CSJDPQV, with FDs:
  - **C** is key.
  - **JP \(\rightarrow\) C**: a project buys a given part using a single contract.
  - **SD \(\rightarrow\) P**: a department buys at most one part from a supplier.

- What are the keys? Is it in BCNF?
  - **C, JP, SDJ**: Not in BCNF (but in 3NF).
  - **SD \(\rightarrow\) P**: violates BCNF.

- Lossless-join BCNF decomposition: CSJDPQV, SDP
  - Problem: Checking **JP \(\rightarrow\) C** requires an assertion (using join)!
**Dependency Preserving Decomposition**

- The *projection* of a FD set F onto a decomposed reln R1:
  - all U → V s.t. (a) U, V are both in R1, (b) U → V is in closure F⁺.
  - Notation: F_{R1} = F⁺_{R1}

- Decomposition of R into R1, R2 is *dependency preserving* if
  \[(F_{R1} \cup F_{R2})^+ = F^+\]

- Important to consider F⁺ (not F!) in this definition:
  - ABC, A → B, B → C, C → A, decomposed into AB and BC.
  - Is this dependency preserving? Is C → A preserved?
Decomposition into BCNF

- Relation R with FDs F. If $X \rightarrow Y$ violates BCNF, decompose R into $R1 = R - Y$ and $R2 = XY$.
  - For each Ri, compute $F_{Ri}$ and check if it is in BCNF.
  - If not, pick a FD violating BCNF and keep composing Ri.

- Repeated application of this process yields a lossless join decomposition into BCNF relations.
Steps of BCNF Decomposition

- Contracts(CSJDQPQV), key C, JP → C, SD → P, J → S.

1. Keys and FDs.

C is key
JP is key
DJ is key:
  J → S, so DJ → DS → P.
We also have DJ → J. Using the union rule, we have
  DJ → JP
FDs:
  SD → P and
  J → S.
Steps of BCNF Decomposition

- Contracts(CSJDPQV), key C, JP → C, SD → P, J → S.
  1. **Keys and FDs.** Keys: C, JP, DJ. FDs: SD → P and J → S.
  2. **Normal form.** SD → P and J → S violate BCNF (Not in 3NF)
  3. **Decomposition.** To deal with SD → P, decompose into SDP, CSJDQV.
    - **SDP** is in BCNF. **Key: SD.**
    - CSJDQV is not in BCNF because:
      1. **Projection of FDs and keys.** Projection of FDs: keys C and DJ, J → S.
      2. **Normal form.** Not BCNF; J → S violates BCNF.
      3. **Decomposition.** For J → S, decompose CSJDQV into JS and CJDQV.
        - **JS** is in BCNF. **Key: J.**
        - **CJDQV** is also in BCNF. **Keys: C, DJ.**

- If several FDs violate BCNF, the order of "dealing with" them could lead to very different sets of relations!
BCNF and Dependency Preservation

- Is a *lossless-join BCNF* decomposition *dependency-preserving*?
  - CSJDPQV with JP $\rightarrow$ C, SD $\rightarrow$ P and J $\rightarrow$ S.
  - CSJDPQV $\rightarrow$ SDP, JS and CJDQV
  - What about JP $\rightarrow$ C?
    - Adding JPC as a new relation to preserve JP $\rightarrow$ C introduces *redundancy across relations and more joins*
    - If we also have J $\rightarrow$ C, JPC is not in BCNF.

- In general, there may **not** exist a *lossless join, dependency-preserving* decomposition into BCNF.
  - But there is always a *lossless join, dependency-preserving* decomposition into 3NF *(not required in this class).*