Schema Refinement and Normal Forms

Yanlei Diao
UMass Amherst
April 10 & 15, 2007
Case Study: The Internet Shop

- **DBDudes Inc.**: a well-known database consulting firm
- **Barns and Nobble (B&N)**: a large bookstore specializing in books on horse racing
- **B&N** decides to go online, asks DBDudes to help with the database design and implementation
# Redundant Storage

## Orders

<table>
<thead>
<tr>
<th>ordernum</th>
<th>isbn</th>
<th>cid</th>
<th>cardnum</th>
<th>qty</th>
<th>order_date</th>
<th>ship_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0-07-11</td>
<td>123</td>
<td>40241160</td>
<td>2</td>
<td>Jan 3, 2006</td>
<td>Jan 6, 2006</td>
</tr>
<tr>
<td>120</td>
<td>1-12-23</td>
<td>123</td>
<td>40241160</td>
<td>1</td>
<td>Jan 3, 2006</td>
<td>Jan 11, 2006</td>
</tr>
<tr>
<td>120</td>
<td>0-07-24</td>
<td>123</td>
<td>40241160</td>
<td>3</td>
<td>Jan 3, 2006</td>
<td>Jan 26, 2006</td>
</tr>
</tbody>
</table>

## Orderlists

<table>
<thead>
<tr>
<th>ordernum</th>
<th>isbn</th>
<th>qty</th>
<th>ship_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>0-07-11</td>
<td>2</td>
<td>Jan 6, 2006</td>
</tr>
<tr>
<td>120</td>
<td>1-12-23</td>
<td>1</td>
<td>Jan 11, 2006</td>
</tr>
<tr>
<td>120</td>
<td>0-07-24</td>
<td>3</td>
<td>Jan 26, 2006</td>
</tr>
</tbody>
</table>

---

Redundant Storage!
The Evils of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:
  - Redundant storage
  - Operation (insert, delete, update) anomalies

- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
  - ICs that we have learned: *domain constraints*, *primary key*, *candidate key*, *foreign key*
  - A new type of IC: *functional dependencies*
Schema Refinement

- Main refinement technique: decomposing a relation into multiple smaller ones

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation? Theory on normal forms.
  - What problems (if any) does the decomposition cause? Properties of decomposition include lossless-join and dependency-preserving.
  - Decomposition can cause performance problems. E.g. a previous selection now requires a join!
Functional Dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation $R$ if $\forall$ allowable instance $r$ of $R$:
  - $t1 \in r, t2 \in r, \pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$, $X$ and $Y$ are sets of attributes.

- An FD is a statement about \textit{all} allowable relations.
  - Must be identified based on semantics of application.
  - Given an allowable instance $r1$ of $R$, we can check if $r1$ violates some FD $f$, but we cannot tell if $f$ holds over $R$!

- $K$ is a candidate key for $R$ means that $K \rightarrow R$.
  - However, $K \rightarrow R$ does not require $K$ to be \textit{minimal}!
Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)

- **Notation**: denote this relation schema by listing all its attributes: SNLRWH

- Some FDs on Hourly_Emps:
  - ssn is the key: S → SNLRWH
  - rating determines hrly_wages: R → W
**Example (Contd.)**

- Problems due to $R \rightarrow W$:
  - **Redundant storage**
  - **Update anomaly**: Can we change $W$ in just the 1st tuple of SNLRWH?
  - **Insertion anomaly**: What if we want to insert an employee and don’t know the hourly wage for his rating?
  - **Deletion anomaly**: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

<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>

Will 2 smaller tables be better?
Reasoning About FDs

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

- An FD \( f \) is **implied by** a set of FDs \( F \), if \( f \) holds for every reln instance that satisfies all FDs in \( F \).
  - \( F^+ = \text{Closure of } F \) is the set of all FDs that are implied by \( F \).

- 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 \( X \rightarrow Z \)
Reasoning About FDs (Contd.)

- Couple of 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 $X \rightarrow Z$

- These are *sound* and *complete* inference rules for FDs!
  - Soundness: when applied to a set $F$ of FDs, the axioms generate only FDs in $F^+$.
  - Completeness: repeated application of these axioms will generate all FDs in $F^+$.
Reasoning About FDs (Contd.)

- Computing the closure \( F^+ \) can be expensive:
  - Compute for all FD’s.
  - Size of closure is exponential in number of attrs!

- Typically, we just want to check if a given FD \( X \rightarrow Y \) is in \( F^+ \). An efficient check:
  - Compute attribute closure of \( X \) (denoted \( X^+ \)) w.r.t. \( F \), i.e., the largest attribute set \( A \) such that \( X \rightarrow A \) is in \( F^+ \).
  - Check if \( Y \subseteq X^+ \).
**Attribute Closure**

- Simple algorithm for *attribute closure* $X^+$:
  - 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 $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D \rightarrow E\}$ imply $A \rightarrow E$?
  - That is, is $A \rightarrow E$ in the closure $F^+$?
  - Equivalently, is $E$ in $A^+$?
Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- **Normal forms**: If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain redundancy related problems are avoided/minimized.
- **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)

- Rewrite every FD in the form of $X \rightarrow A$ ($X$ is a set of attributes, $A$ is a single attribute) using the decomposition rule.

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

- R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
- Can we infer the value marked by ‘?’?
  - Is the relation in BCNF?
  - If a reln is in BCNF, every field of every tuple records a piece of information that can’t be inferred (using only FD’s) from values in other fields.

- **BCNF ensures that no redundancy can be detected using FDs!**
Third Normal Form (3NF)

- Reln R with FDs F is in 3NF if \( \forall X \rightarrow A \) in \( F^+ \):
  - \( A \in X \) (called a *trivial* FD), or
  - \( X \) is a *superkey* for R, or
  - \( A \) is part of some *key* for R. (*Minimality* of a key is crucial in the third condition!)

- If R is in BCNF, obviously in 3NF.
Third Normal Form (contd.)

- If R is in 3NF, *some redundancy is possible!*
  - Reserves\{Sailor, Boat, Date, Credit_card\} with S \(\rightarrow\) C, C \(\rightarrow\) S
  - It is in 3NF, because keys are SBD and CBD.
  - But for each reservation of sailor S, same (S, C) is stored.

- Why 3NF?
  - *Lossless-join, dependency-preserving* decomposition of R into *3NF relations* is always possible.
  - This is not true for BCNF!
Decomposition of a Relation Scheme

- A decomposition of $R$ replaces $R$ by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of $R$, and
  - Every attribute of $R$ appears as an attribute of at least one new relation.

- Store instances of the relation schemas produced by the decomposition, instead of instances of $R$. 
Example Decomposition

- Decompositions should be used only when needed.
  - Hourly_Emps (SNLRWH) has FDs $S \rightarrow \text{SNLRWH}$ and $R \rightarrow W$.
  - $R \rightarrow W$ causes violation of 3NF; $W$ values repeatedly associated with $R$ values.
  - A way to fix this is to create a relation $RW$ to store these associations, and to remove $W$ from the main schema:
    - i.e., decompose $\text{SNLRWH}$ into $\text{SNLRH}$ and $RW$.
- Any potential problems with storing $\text{SNLRH}$ and $RW$ instead of $\text{SNLRWH}$?
Problems with Decompositions

- Three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? (salary = W*H)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.

- Tradeoff: Must consider these issues vs. redundancy.
Lossless Join Decompositions

- Decomposition of R into R1 and R2 is lossless-join w.r.t. a set of FDs F if ∀ instance r that satisfies F:
  - $\pi_{R1}(r) \bowtie \pi_{R2}(r) = r$

- It is always true that $r \subseteq \pi_{R1}(r) \bowtie \pi_{R2}(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.

- It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)
More on Lossless Join

- Decomposition of $R$ into $R_1$ and $R_2$ is lossless-join wrt $F$ iff the closure of $F$ contains:
  - $R_1 \cap R_2 \rightarrow R_1$, or
  - $R_1 \cap R_2 \rightarrow R_2$
  - i.e. intersection of $R_1$, $R_2$ is a (super) key of one of them.

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

- Consider Contracts(Contractid, Supplierid, Projectid, Deptid, Partid, Qty, Value), denoted by CSJDPQV.

- Functional dependencies:
  - C is key.
  - JP $\rightarrow$ C: a project purchases a given part using a single contract.
  - SD $\rightarrow$ P: a department purchases at most one part from a supplier.

- Lossless-join BCNF decomposition: CSJDQV, SDP
  - Problem: Checking JP $\rightarrow$ C requires a join!
Dependency Preserving Decomposition

- Dependency preserving decomposition:
  - If R is decomposed into R1 and R2 and we enforce the FDs that hold on R1 and R2 respectively, 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 R1, ..., projection of F onto R1 (denoted $F_{R1}$) is the set of FDs $U \rightarrow V$ such that (i) $U$, $V$ are both in R1 and (ii) $U \rightarrow V$ is in closure $F^+$.
  - $F_{R1} \equiv F^+_{R1}$
Dependency Preserving Decompositions (Contd.)

- Formally, decomposition of R into R1 and R2 is dependency preserving if \((F_{R1} \cup F_{R2})^+ = F^+\)

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

- Dependency preserving does not imply lossless join:
  - ABC, \(A \to B\), decomposed into AB and BC.
  - And vice-versa! (Example?)
Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow Y$ violates BCNF, decompose R into $R_1 = R - Y$ and $R_2 = XY$.
  - For each $R_i$, compute $F_{R_i}$ and check if it is in BCNF.
  - If not, pick a FD violating BCNF and keep composing $R_i$.
  - Repeated application of this idea gives us a \textit{lossless join} decomposition into \textit{BCNF} relations, and is guaranteed to terminate.
Decomposition into BCNF

- Contracts(CSJDPQV), key C, JP → C, SD → P, J → S.
  2. Normal form. Not BCNF, SD → P and J → S violate BCNF.
  3. Decomposition. To deal with SD → P, decompose into SDP, CSJDQV.
    - SDP is in BCNF. But CSJDQV is not because:
      1. Projection of FDs and keys. Projection of FDs: keys C and SDJ, J → S.
      3. Decomposition. For J → S, decompose CSJDQV into JS and CJDQV.
        - JS is in BCNF. So is CJDQV.
- If several FDs violate BCNF, the order in which we ``deal with’’ them could lead to very different sets of relations!
BCNF and Dependency Preservation

- In general, there may not be a dependency-preserving decomposition into BCNF.
  - Decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP $\rightarrow$ C, SD $\rightarrow$ P and J $\rightarrow$ S).
  - However, it is a lossless join decomposition.
  - Adding JPC as a new relation gives a dependency preserving decomposition. But JPC tuples stored only for checking FD — Redundancy across relations!
  - If we also have J $\rightarrow$ C, JPC is not in BCNF.
Decomposition into 3NF

- The algorithm for lossless join decomposition into BCNF can be used to obtain a lossless join decomposition into 3NF (typically, can stop earlier).

- Idea to ensure dependency preservation: If $X \rightarrow Y$ is not preserved, add relation $XY$.
  - Problem is that $XY$ may violate 3NF!
  - Suppose $AB \rightarrow C$ is lost in decomposition. Add $ABC$ to `preserve’ $AB \rightarrow C$. What if we also have $A \rightarrow B$?

- Refinement: Instead of the given set of FDs $F$, use a minimal cover for $F$ (minimal FD set $G$ s.t. $G^+ = F^+$).
Decomposition into 3NF

- Step 1: Given F of FDs, compute its minimal cover G (not required in this class).
- Step 2: Use G to create a lossless-join decomposition of R into R1, …, Rn.
- Step 3: Identify the dependencies in F+ that are not preserved. For each such FD X→A, add a new relation XA.
- This algorithm produces a lossless-join, dependency-preserving decomposition into 3NF.
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.