Schema Refinement and Normal Forms

UMass Amherst
Feb 14, 2007

Slides Courtesy of R. Ramakrishnan and J. Gehrke, Dan Suciu
Relational Schema Design

Conceptual Design

ER Model

Logical design

Relational Schema plus Integrity Constraints

Schema Refinement

Normalized schema
The Evils of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:
  - redundant storage
  - insert anomaly
  - delete anomaly
  - update anomaly

- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
Schema Refinement

- Main refinement technique: decomposition
  - E.g., replacing ABCD with AB and BCD, or ACD and ABD.

- 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.
Functional Dependencies

Table R(.... A₁, A₂, …, Aₙ… B₁, B₂, …, Bₘ… )

Functional Dependency:

$$A₁, A₂, …, Aₙ \rightarrow B₁, B₂, …, Bₘ$$

Meaning:

If two tuples agree on the attributes

$$A₁, A₂, …, Aₙ$$

then they must also agree on the attributes

$$B₁, B₂, …, Bₘ$$
Example

Hourly_Emps (ssn, name, lot, rating, hrly_wages, hrs_worked)

- **Notation:** We will denote this relation schema by listing the attributes: **SNLRWH**

  This is really the set of attributes \{S,N,L,R,W,H\}.

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>lot</th>
<th>rating</th>
<th>wages</th>
<th>hours</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>231-31-5386</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>

- Some FDs on Hourly_Emps:
  
  \[ S \rightarrow SNLRWH \]

  *rating determines hrly_wages:*  \[ R \rightarrow W \]
Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d2</td>
</tr>
<tr>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d1</td>
</tr>
<tr>
<td>a2</td>
<td>b1</td>
<td>c3</td>
<td>d1</td>
</tr>
</tbody>
</table>
Functional Dependencies (FDs)

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

- An FD is a statement about all allowable relations.
  - Must be identified based on semantics of application.
  - Given an allowable instance $r_1$ of $R$, we can check if $r_1$ violates some FD $f$, but we cannot tell if $f$ holds over $R$!

- A superkey is a set of attributes $K$ such that $K \rightarrow B$ for all attributes $B$.

- A key is a minimal superkey
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!

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 \textit{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):
  - \textit{Reflexivity:} If $X \subseteq Y$, then $Y \rightarrow X$
  - \textit{Augmentation:} If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any $Z$
  - \textit{Transitivity:} If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
Reasoning About FDs (Contd.)

- 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 $F^+$. 
Example (continued)

From:

1. name $\rightarrow$ color
2. category $\rightarrow$ department
3. color, category $\rightarrow$ price

To:

name, category $\rightarrow$ price

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category $\rightarrow$ name</td>
<td>Reflexivity</td>
</tr>
<tr>
<td>5. name, category $\rightarrow$ color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category $\rightarrow$ category</td>
<td>Reflexivity</td>
</tr>
<tr>
<td>7. name, category $\rightarrow$ color, category</td>
<td>Union on 5, 6</td>
</tr>
<tr>
<td>8. name, category $\rightarrow$ price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>
Reasoning About FDs (Contd.)

- Computing the closure $F^+$ can be expensive: computes for all FD’s; size of closure is exponential in # 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 $F^+$.
  - Simple algorithm: DO if there is $U \rightarrow V$ in $F$ s.t. $U \subseteq X^+$, then $X^+ = X^+ \cup UV$ UNTIL no change
  - Check if $Y$ is in $X^+$.
- Does $F = \{A \rightarrow B, B \rightarrow C, C D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e., is $A \rightarrow E$ in the closure $F^+$? Equivalently, is $E$ in $A^+$?
Computing Keys

- Compute $X^+$ for all sets $X$
- If $X^+ = \text{all attributes}$, then $X$ is a superkey
- Consider only the minimal superkeys

**Enrollment(student, address, course, room, time)**

- student $\rightarrow$ address
- room, time $\rightarrow$ course
- student, course $\rightarrow$ room, time

Please compute all keys.
Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain normal form (BCNF, 3NF etc.), it is known that certain redundancy related problems are avoided/minimized.
  - This helps us decide if decomposing the relation will help.
- 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)

- Given a relation R, and set of FD’s F on R
- R is in BCNF if:
  - For each FD $X \rightarrow A$, one of the following is true:
    - $A \in X$ (called a *trivial* FD), or
    - $X$ is a *superkey* (i.e., contains a key) for R.
- “The only non-trivial FDs that hold over R are key constraints.”

Equivalently: for any set of attributes $X$, either $X^+ = X$
  or $X^+ = \text{all attributes}
Example

- Is the following table in BCNF?
  - R(A,B,C,D)
  - FDs: B → AD
- Key is BC, so B is not a superkey
- Not in BCNF
Boyce-Codd Normal Form (BCNF)

- 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)

- R is in 3NF if:
  - For each \( X \rightarrow A \) one of the following is true:
    - \( A \in X \) (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 third condition above!

- If R is in BCNF, obviously in 3NF.

- If R is in 3NF, some redundancy is possible.
  - E.g., Reserves \{Sailor, Boat, Date, Credit_card\} with \( S \rightarrow C \), \( C \rightarrow S \) is in 3NF. But for each reservation of sailor S, same \((S, C)\) pair is stored.
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible. (not true for BCNF!)
Hierarchy of Normal Forms

- 1\textsuperscript{st} normal form 1NF: no set-valued attributes.
- 2\textsuperscript{nd} normal form 2NF: [historical interest only]
- 3\textsuperscript{rd} normal form 3NF
- Boyce-Codd normal form BCNF: 3NF, and no $X \rightarrow A$ s.t. $A$ is part of a key. No redundancy detected by FDs.
- 4\textsuperscript{th} normal form 4NF: BCNF and no multi-valued dependencies (MVD). No redundancy detected by FDs and MVD.
  - We won’t discuss in detail in this class.
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 no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of at least one new relation.

- As a result, we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
Decompositions in General

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ R_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ R_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S \(\rightarrow\) SNLRWH and R \(\rightarrow\) W.
  - R \(\rightarrow\) W causes violation of 3NF; W values repeatedly associated with R values.
  - Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW.

- If we just store the projections of SNLRWH tuples onto SNLRH and RW, are there any potential problems that we should be aware of?
Problems with Decompositions

- Three potential problems to consider:
  1. Some queries become more expensive.
     - e.g., How much did sailor Joe earn? (salary = W*H)
  2. 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.
  3. 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 X and Y is \textit{lossless-join} w.r.t. a set of FDs F if \( \forall \) instance \( r \) that satisfies F:
  - \( \pi_X(r) \bowtie \pi_Y(r) = r \)
- It is always true that \( r \subseteq \pi_X(r) \bowtie \pi_Y(r) \)
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- \textit{It is essential that all decompositions used to deal with redundancy be lossless!} (Avoids Problem (2).)
More on Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - $X \cap Y \rightarrow X,$ or
  - $X \cap Y \rightarrow Y$

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

- Dependency preserving decomposition:
  - Intuitively, if R is decomposed into X and Y, and we enforce the FDs that hold on X and Y 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 X, ..., projection of F onto X (denoted $F_X$) is the set of FDs $U \rightarrow V$ in closure $F^+$ such that $U, V$ are both in X.
Dependency Preserving Decompositions (Contd.)

- Formally, decomposition of R into X and Y is **dependency preserving** if \((F_X \text{ union } F_Y)^+ = F^+\)
- 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! (Example?)
Decomposition into BCNF

- Consider relation R with FDs F. If X \( \rightarrow \) Y violates BCNF, decompose R into XY and R - Y.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
    - e.g., CSJDPQV, key C, JP \( \rightarrow \) C, SD \( \rightarrow \) P, J \( \rightarrow \) S
    - To deal with SD \( \rightarrow \) P, decompose into SDP, CSJDQV.
    - To deal with J \( \rightarrow \) S, decompose CSJDQV into JS and CJDQV
- Several dependencies may cause violation of 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.
  - e.g., CSZ, CS → Z, Z → C
  - Not in BCNF; can’t decompose while preserving 1st FD.
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).
  - However, it is a lossless join decomposition.
  - Adding JPC to the collection of relations gives a dependency preserving decomposition. JPC tuples stored only for checking FD! (Redundancy!)
3NF Discussion

- 3NF decomposition v.s. BCNF decomposition:
  - Use same decomposition steps, for a while
  - 3NF may stop decomposing, while BCNF continues

- Tradeoffs
  - BCNF = no anomalies, but may lose some FDs
  - 3NF = keeps all FDs, but may have some anomalies
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.
Questions