Design Theory for Relational Databases

csc343, fall 2014
Diane Horton
University of Toronto

Originally based on slides by Jeff Ullman
Introduction

◆ There are always many different schemas for a given set of data.
◆ E.g., you could combine or divide tables.
◆ How do you pick a schema? Which is better? What does “better” mean?
◆ Fortunately, there are some principles to guide us.
Database Design Theory

- It allows us to improve a schema systematically.
- General idea:
  - Express constraints on the relationships between attributes
  - Use these to decompose the relations
- Ultimately, get a schema that is in a “normal form” that guarantees good properties, such as no anomalies.
- “Normal” in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called normalization.
Part I:
Functional Dependency Theory
A poorly designed table

<table>
<thead>
<tr>
<th>part</th>
<th>manufacturer</th>
<th>manAddress</th>
<th>seller</th>
<th>sellerAddress</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>5.59</td>
</tr>
<tr>
<td>8624</td>
<td>Lee Valley</td>
<td>102 Vaughn</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>23.99</td>
</tr>
<tr>
<td>9141</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>12.50</td>
</tr>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>Walmart</td>
<td>5289 St Clair W</td>
<td>4.99</td>
</tr>
</tbody>
</table>

 имени

- In any domain, there are relationships between attribute values.
- Perhaps:
  - Every part has 1 manufacturer
  - Every manufacture has 1 address
  - Every seller has 1 address
- If so, this table will have redundant data.
Principle: Avoid redundancy

Redundant data can lead to anomalies.

<table>
<thead>
<tr>
<th>part</th>
<th>manufacturer</th>
<th>manAddress</th>
<th>seller</th>
<th>sellerAddress</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>5.59</td>
</tr>
<tr>
<td>8624</td>
<td>Lee Valley</td>
<td>102 Vaughn</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>23.99</td>
</tr>
<tr>
<td>9141</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>ABC</td>
<td>1229 Bloor W</td>
<td>12.50</td>
</tr>
<tr>
<td>1983</td>
<td>Hammers ‘R Us</td>
<td>99 Pinecrest</td>
<td>Walmart</td>
<td>5289 St Clair W</td>
<td>4.99</td>
</tr>
</tbody>
</table>

- **Update anomaly**: if Hammers ‘R Us moves and we update only one tuple, the data is inconsistent.
- **Deletion anomaly**: If ABC stops selling part 8624 and Lee Valley makes only that one part, we lose track of its address.
Definition of FD

- Suppose R is a relation, and X and Y are subsets of the attributes of R.
- X -> Y asserts that:
  - If two tuples agree on all the attributes in set X, they must also agree on all the attributes in set Y.
- We say that “X -> Y holds in R”, or “X functionally determines Y.”
- An FD constrains what can go in a relation.
- [Exercise]
Why “functional dependency”? 

◆ “dependency” because the value of $Y$ depends on the value of $X$. 
◆ “functional” because there is a mathematical function that takes a value for $X$ and gives a unique value for $Y$. 
◆ (It’s not a typical function; just a lookup.)
Equivalent sets of FDs

- When we write a set of FDs, we mean that all of them hold.
- We can very often rewrite sets of FDs in equivalent ways.
- When we say $S_1$ is equivalent to $S_2$ we mean that:
  - $S_1$ holds in a relation iff $S_2$ does.
- [Exercise]
Splitting rules for FDs

- Can we split the RHS of an FD and get multiple, equivalent FDs?

- Can we split the LHS of an FD and get multiple, equivalent FDs?
Coincidence or FD?

- An FD is an assertion about *every* instance of the relation.
- You can’t know it holds just by looking at one instance.
- You must use knowledge of the domain to determine whether an FD holds.
FDs are closely related to keys

- Suppose K is a set of attributes for relation R.

- Our old definition of superkey:
  a set of attributes for which no two rows can have the same values.

- A claim about FDs:
  $K$ is a superkey for $R$ iff
  $K$ functionally determines all of $R$.

- [Exercise]
FDs are a generalization of keys

- **Superkey:**
  \[X \rightarrow R\]
  Every attribute

- **Functional dependency:**
  \[X \rightarrow Y\]

- A superkey must include *all* the attributes of the relation on the RHS.

- An FD can have just a subset of them.
Inferring FDs

- Given a set of FDs, we can often infer further FDs.
- This will come in handy when we apply FDs to the problem of database design.
- Big task: given a set of FDs, infer every other FD that must also hold.
- Simpler task: given a set of FDs, infer whether a given FD must also hold.
Examples

◆ If $A \to B$ and $B \to C$ hold, must $A \to C$ hold?

◆ If $A \to H$, $C \to F$, and $FG \to AD$ hold, must $FA \to D$ hold?
  must $CG \to FH$ hold?

◆ If $H \to GD$, $HD \to CE$, and $BD \to A$ hold, must $EH \to C$ hold?

◆ Aside: we are not generating new FDs, but testing a specific possible one.
Method 1: Prove an FD follows using first principles

- You can prove it by referring back to
  - The FDs that you know hold, and
  - The definition of functional dependency.
- But the Closure Test is easier.
Method 2: Prove an FD follows using the Closure Test

- Assume you know the values of the LHS attributes, and figure out everything else that is determined.
- If it includes the RHS attributes, then you know that LHS -> RHS
- This is called the closure test.
\textit{Y} is a set of attributes, \textit{S} is a set of FDs. \textit{Return the closure of \textit{Y} under \textit{S}.}

\textbf{Attribute\_closure(\textit{Y}, \textit{S})}: 

Initialize \textit{Y}^{+} to \textit{Y} 

Repeat until no more changes occur: 

If there is an FD \textit{LHS} \rightarrow \textit{RHS} in \textit{S} 

such that \textit{LHS} is in \textit{Y}^{+}: 

Add \textit{RHS} to \textit{Y}^{+} 

\textbf{Return } \textit{Y}^{+}
Visualizing attribute closure

If LHS is in $Y^+$ and LHS $\rightarrow$ RHS holds, we can add RHS to $Y^+$
$S$ is a set of FDs; LHS $\rightarrow$ RHS is a single FD. Return true iff LHS $\rightarrow$ RHS follows from $S$.

\[
\text{FD\_follows}(S, \text{LHS} \rightarrow \text{RHS}):
\]
\[
Y^+ = \text{Attribute\_closure}(\text{LHS}, S)
\]
\[
\text{return (RHS is in } Y^+) \]

[Exercise]
Projecting FDs

◆ Later, we will learn how to normalize a schema by decomposing relations. (This is the whole point of this theory.)
◆ We will need to be aware of what FDs hold in the new, smaller, relations.
◆ In other words, we must project our FDs onto the attributes of our new relations.
◆ [Exercise]
S is a set of FDs; L is a set of attributes.

Return the projection of S onto L:
all FDs that follow from S and involve only attributes from L.

Project(S, L):
Initialize T to {}.
For each subset X of L:
  Compute $X^+$  Close X and see what we get.
  For every attribute A in $X^+$:
    If A is in L:  $X \rightarrow A$ is only relevant if A is in L (we know X is).
    add $X \rightarrow A$ to T.
  Return T.

[Example]
A few speed-ups

- No need to add $X \rightarrow A$ if $A$ is in $X$ itself. It’s a trivial FD.
- These subsets of $X$ won’t yield anything, so no need to compute their closures:
  - the empty set
  - the set of all attributes
- Neither are big savings, but ...
A big speed-up

◆ If we find $X^+ = \text{all attributes}$, we can ignore any superset of $X$.
  ▶ It can only give use “weaker” FDs (with more on the LHS).

◆ This is a big time saver!
Projection is expensive

- Even with these speed-ups, projection is still expensive.
- Suppose $R_1$ has $n$ attributes. How many subsets of $R_1$ are there?
Minimal Bases

- We saw earlier that we can very often rewrite sets of FDs in equivalent ways.
- Example: $S_1 = \{A \rightarrow BC\}$ is equivalent to $S_2 = \{A \rightarrow B, A \rightarrow C\}$.
- Given a set of FDs $S$, we may want to find a minimal basis: A set of FDs that is equivalent, but has
  - no redundant FDs, and
  - no FDs with unnecessary attributes on the LHS.
S is a set of FDs. Return a minimal basis for S.

**Minimal_basis(S):**

Repeat until no more changes result:
   Remove FDs that are implied by the rest.
   For each FD with $2^+$ attributes on the left:
      If you can remove one attribute from the LHS and get an FD that follows from the rest:
         Do so! (It’s a stronger FD.)

[Exercise]
Some comments on computing a minimal basis

- Often there are multiple possible results, depending on the order in which you consider the possible simplifications.
- After you indentify a redundant FD, you must not use it when computing any subsequent closures (as you consider whether other FDs are redundant).
... and some that are less intuitive

- When you are computing closures to decide whether the LHS of an FD
  \[ a_1a_2...a_m \rightarrow b_1b_2...b_n \]
  can be simplified, continue to use that FD.

- When you have tried to eliminate each FD and to reduce each LHS, you must go back and try again