Database Design: Functional dependencies

CS315

Feb 2 , 2015

Algorithm for Testing Lossless Joins

Input: Relation schema $R = \{A_1, A_2, \ldots, A_n\}$, a set of functional dependencies F and a decomposition scheme $\rho = (R_1, R_2, \ldots, R_k)$.
Output: A decision whether ρ is a lossless join decomposition under F.

Lossless testing algorithm

1. Construct a *table* with n columns and k rows.
   Column j corresponds to A_j and row i corresponds to relation R_i.
2. If $A_j \in R_i$, put the symbol a_j in (i, j) position.
   Otherwise, put the symbol b_{ij} in (i, j) position.
3. **repeat**
4. **for** each dependency $X \to Y$ in F **do**
5. **if** there are two (or more) rows that agree in all the columns for the attributes of X
6. equate the symbols of those rows for the attributes of Y as follow
   if one of the symbols is a_j, make the other to be a_j.
   if the symbols are b_{ij}, b_{lj}, make them both b_{ij} or b_{lj} arbitrarily.
7. **until** there is no change to the table.
8. **if** there is some row that is $a_1, \ldots, a_k$, then the join is lossless
   else it is lossy.

Example

- Consider the example schema $Suppliers(S, A, I, P)$ with functional dependencies $S \rightarrow A$ and $SI \rightarrow P$.
- Initial table created is:

	S	A	I	P
SA	a_1	a_2	b_{13}	b_{14}
SIP	a_1	b_{12}	a_3	a_4

- Consider dependency $S \rightarrow A$. Equate a_2 with b_{12}. This gives

	S	A	I	P
SA	a_1	a_2	b_{13}	b_{14}
SIP	a_1	a_2	a_3	a_4

- Second row is all a's. Decomposition is lossless join.

Example 2

- $R = \{A, B, C, D, E\}$. Functional dependencies are

$$A \to C \qquad DE \to C$$
$$B \to C \qquad CE \to A \qquad C \to D$$

  Decomposition $R_1 = AD, R_2 = AB, R_3 = BE, R_4 = CDE, R_5 = AE$.
- Initial table:

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	b_{13}	a_4	b_{15}
$R_2(AB)$	a_1	a_2	b_{23}	b_{24}	b_{25}
$R_3(BE)$	b_{31}	a_2	b_{33}	b_{34}	a_5
$R_4(CDE)$	b_{41}	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	b_{53}	b_{54}	a_5

- Apply $A \to C$. Rows 1,2 and 5 have a_1 in column A. So equate b_{13}, b_{23} and b_{53} to say b_{13}.

Example ...

$$A \rightarrow C \qquad DE \rightarrow C$$

$$B \rightarrow C \qquad CE \rightarrow A \qquad C \rightarrow D$$

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	b_{13}	a_4	b_{15}
$R_2(AB)$	a_1	a_2	b_{13}	b_{24}	b_{25}
$R_3(BE)$	b_{31}	a_2	b_{33}	b_{34}	a_5
$R_4(CDE)$	b_{41}	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	b_{13}	b_{54}	a_5

- Apply $B \rightarrow C$. Equate b_{13} and b_{33}.

Example ....

$$A \to C \qquad DE \to C$$

$$B \to C \qquad CE \to A \qquad C \to D$$

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	b_{13}	a_4	b_{15}
$R_2(AB)$	a_1	a_2	b_{13}	b_{24}	b_{25}
$R_3(BE)$	b_{31}	a_2	b_{13}	b_{34}	a_5
$R_4(CDE)$	b_{41}	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	b_{13}	b_{54}	a_5

- Apply $C \to D$. Rows 1,2,3 and 5 have b_{13} in column C. We equate the values a_4, b_{24}, b_{34} and b_{54} to a_4.

$$A \to C \qquad DE \to C$$

$$B \to C \qquad CE \to A \qquad C \to D$$

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	b_{13}	a_4	b_{15}
$R_2(AB)$	a_1	a_2	b_{13}	a_4	b_{25}
$R_3(BE)$	b_{31}	a_2	b_{13}	a_4	a_5
$R_4(CDE)$	b_{41}	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	b_{13}	a_4	a_5

- Now apply $DE \to C$. Rows 4 and 5 are equal on DE columns. Equate a_3 with b_{53}.

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	a_3	a_4	b_{15}
$R_2(AB)$	a_1	a_2	a_3	a_4	b_{25}
$R_3(BE)$	b_{31}	a_2	a_3	a_4	a_5
$R_4(CDE)$	b_{41}	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	a_3	a_4	a_5

- Now apply $CE \rightarrow A$. Rows 3, 4 and 5 are equal on CE columns. Equate a_1 with b_{31} and b_{41}.

$$A \to C \qquad DE \to C$$

$$B \to C \qquad CE \to A \qquad C \to D$$

	A	B	C	D	E
$R_1(AD)$	a_1	b_{12}	a_3	a_4	b_{15}
$R_2(AB)$	a_1	a_2	a_3	a_4	b_{25}
$R_3(BE)$	a_1	a_2	a_3	a_4	a_5
$R_4(CDE)$	a_1	b_{42}	a_3	a_4	a_5
$R_5(AE)$	a_1	b_{52}	a_3	a_4	a_5

- Row 3 is $a_1, a_2, \ldots, a_5$. Hence decomposition has a lossless join.

Proof of Correctness

- Suppose the final table produced by the algorithm does not have a row of $a_1, a_2, \ldots, a_n$.
- View the final table as a relation r on schema R. The rows are tuples, and a_i's and b_{ij}'s are symbols in the domain of A_j.
- Relation r satisfies the functional dependencies in F because, whenever a violation is found, the algorithm modifies the table accordingly (by equating symbols).

Proof ...

- Claim: $r \neq m_\rho(r)$.
- For each $r_i(R)$, there is a tuple $t_i \in r$ such that $t_i[R_i]$ is all a's.
- So the join of $\pi_{R_i}(r)$'s contains the tuple with all a's.
- But $(a_1, \ldots, a_k) \notin r$. Hence $r \neq m_\rho(r)$.
- Hence the decomposition ρ is not a lossless join.

Proof of Converse

- Conversely, suppose that the final table has a row with all a's.
- Consider the query

$$\{(a_1, \ldots, a_n) \mid (\exists b_{11}) \ldots (\exists b_{kn})(w_1 \in r \wedge \ldots \wedge w_k \in r)$$

  where w_i is the ith row of the initial table.
- View the table as shorthand for this query.
- Query defines m_ρ since $m_\rho(r)$ contains an arbitrary tuples $a_1, \ldots, a_n$ iff for each i, r contains a tuple with a's in the attributes of R_i and arbitrary values in the other attributes.

$$\{a_1, \ldots, a_n \mid (\exists b_{11}) \ldots (\exists b_{kn})(w_1 \in r \wedge \ldots \wedge w_k \in r)$$

- Since we assume that any relation r to which the query can be applied satisfies the dependencies in F, hence
- hence the query is equivalent to a set of similar formulas with some of the a's and/or b's identified.
- The modifications made by the algorithm are such that the table is always a shorthand for some formula whose value on relation r is $m_\rho(r)$ whenever r satisfies F. This can be proved by induction on the number of symbols identified.
- Since the final table contains a row of all a's, the query for the final table is of the form

$$\{a_1, \ldots, a_n \mid r(a_1, \ldots, a_n) \wedge \ldots\}$$

- This is a subset of r. But it is also $m_\rho(r)$. Hence $m_\rho(r) \subset r$.
- Hence, whenever r satisfies F, $r = m_\rho(r)$, or, that the decomposition is lossless.

Decomposition into two fragments

Lemma

If $\rho = (R_1, R_2)$ is a decomposition of R and F is the set of functional dependencies that hold on R, then, ρ is a lossless decomposition iff $(R_1 \cap R_2) \rightarrow R_2 - R_1$ or $(R_1 \cap R_2) \rightarrow R_1 - R_2$ holds in F^+.

- Consider the initial table used by the algorithm. Let $|R_1 \cap R_2| = s$, $|R_1 - R_2| = t$ and $|R_2 - R_1| = u$.

	$R_1 \cap R_2$				$R_1 - R_2$			$R_2 - R_1$		
row for R_1	a_1	a_2	$\ldots$	a_s	a_{s+1}	$\ldots$	a_{s+t}	b_{s+t+1}	$\ldots$	b_{s+t+u}
row for R_2	a_1	a_2	$\ldots$	a_s	b_{s+1}	$\ldots$	b_{s+t}	a_{s+t+1}	$\ldots$	a_{s+t+u}

	$R_1 \cap R_2$				$R_1 - R_2$			$R_2 - R_1$		
row for R_1	a_1	a_2	$\dots$	a_s	a_{s+1}	$\dots$	a_{s+t}	b_{s+t+1}	$\dots$	b_{s+t+u}
row for R_2	a_1	a_2	$\dots$	a_s	b_{s+1}	$\dots$	b_{s+t}	a_{s+t+1}	$\dots$	a_{s+t+u}

Step 1:

- Prove by induction on the number of symbols identified by algorithm that if some b_j corresponding to attribute A_j is equated with a_j, then, $A_j \in (R_1 \cap R_2)^+$.
- Base Case: Straightforward.
- Induction Case: Again, strightforward.

	$R_1 \cap R_2$				$R_1 - R_2$			$R_2 - R_1$		
row for R_1	a_1	a_2	$\dots$	a_s	a_{s+1}	$\dots$	a_{s+t}	$b_{1,s+t+1}$	$\dots$	$b_{1,s+t+u}$
row for R_2	a_1	a_2	$\dots$	a_s	$b_{2,s+1}$	$\dots$	$b_{2,s+t}$	a_{s+t+1}	$\dots$	a_{s+t+u}

Step 2:

- Suppose $(R_1 \cap R_2) \to Y$ has a proof from F using Armstrong's Axioms.
- Then, using an induction on the number of steps in this proof, show that any b_j's in the columns corresponding to Y's are changed to a_j's.

Combining Steps 1 and 2:

- Thus, the row corresponding to R_1 is all a's iff $(R_2 - R_1) \subset (R_1 \cap R_2)^+$.
- Similarly, the row corresponding to R_2 is all a's iff $(R_1 - R_2) \subset (R_1 \cap R_2)^+$.
- The lemma now follows.

Example

Example 1.

- $R = ABC$. $F = \{A \rightarrow B\}$.
- Consider $\rho = (AB, AC)$.
- Since, $AB \cap AC = A$ and $B = AB - AC$, and $A \rightarrow B$, hence,
- decomposition is lossless under F.

Example 2.

- $R = ABC$, $F = \{A \to B\}$.
- Consider $\rho = (AB, BC)$.
- So $AB \cap BC = B$ and $B^+ = B$.
- So B does not determine either A (which is $AB - BC$) or C (which is $BC - AB$).
- Hence decomposition is not lossless.

R

A	B	C
a_1	b	c_1
a_2	b	c_2

$\pi_{A,B}(R)$

A	B
a_1	b
a_2	b

$\pi_{B,C}(R)$:

B	C
b	c_1
b	c_2

$\pi_{AB}(R) \bowtie \pi_{BC}(R)$

A	B	C
a_1	b	c_1
a_1	b	c_2
a_2	b	c_1
a_2	b	c_2

Decompositions that preserve Dependencies

- Decompositions should be lossless: so that the original relation can be recovered from its projections.
- Let R be a schema and $\rho = (R_1, \ldots, R_k)$ be a decomposition and F be the set of functional dependencies.
- The *projection* of F onto a set $Z \subset R$ is the set of all dependencies $X \to Y$ in F^+ such that $XY \subset Z$. Denoted as

$$\pi_Z(F) = \{X \to Y \in F^+ \mid XY \subset Z\}$$

- ρ is said to be dependency preserving if

$$\left(\bigcup_{i=1}^{k} \pi_{R_i}(F)\right) \text{ logically implies } F$$

or

$$F \subset \left(\bigcup_{i=1}^{k} \pi_{R_i}(F)\right)^+$$

Why dependency preservation is useful?

- Dependencies in F are statements about integrity constraints on legal instances of the relation.
- If ρ has the loss join decomposition property, but is not dependency preserving, then,
- each update (insert, modify, delete) to one of the R_i's would require a join to check that the functional dependencies are satisfied.

However,

1. Lossless join decomposition property is absolutely crucial.
2. Dependency preservation is desirable. If decomposition is not dependency preserving, then it increases the runtime overhead to check dependencies by having to compute joins.

Example

- Consider schema $R(City, Street, Pincode\ )$ written as $R(C, S, P)$.
- Functional dependencies $F$are:

$$C, S \to P \qquad\qquad P \to C$$

- Consider decomposition CSP into CP and SP.
- This is lossless since, $CP \cap SP = P$ and $P \to C$, and $C = CP - SP$.
- This is not dependency preserving, since,
  1. $\pi_{CP}(F) = \{P \to C\} \cup \{\text{trivial dependencies}\}$.
  2. $\pi_{SP}(F) = \{\text{ trivial dependencies}\}$.
  3. Hence $CS \to P$ is lost.

Example

S	P
100 M.G. Road	400001
100 M.G. Road	400002

C	P
Bombay	400001
Bombay	400002

- The dependency $CS \rightarrow P$ is violated.

C	S	P
Bombay	100 M.G. Road	400001
Bombay	100 M.G. Road	400002

Testing Preservation of Dependencies

- We now see an algorithm with the following input and output.
- *Input*: A decomposition $\rho = (R_1, \ldots, R_k)$ and a set of functional dependencies F.
- *Output*: A decision whether ρ preserves F.

Method

- Let G be the union of the functional dependencies projected on the fragments that is

$$G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$$

- Test whether G *covers* F, or equivalently,
- for every $X \rightarrow Y$ in F, $Y \subset X_G^+$.
- The key is to compute X^+ without having G available explicitly.
- Trick: Repeatedly close X with respect to the projections of F on each of the R_i's.

Algorithm for testing dependency preservation

- Let R_i be one of the fragments.
- Given a subset $Z \subset R$, an R_i-operation on Z w.r.t. F is to replace Z by
$$Z := Z \cup \left((Z \cap R_i)^+ \cap R_i \right)$$
where the closure is taken with respect to F.
- An R_i-operation adjoins to Z those attributes $A \in R_i$ such that $Z \cap R_i \to A$ holds in F^+.

Computing X_G^+

Now we compute X_G^+ as follows.

1. Start with X.
2. Run through each of the R_i's and perform an R_i-operation on X.
3. If at some pass, none of the R_i-operations change X, then we terminate.
4. The resulting set is X^+.

Algorithm for computing X_G^+

1. $Z = X$
2. **while** changes to Z occur
3. **for** $i = 1$ to k **do**
4. $Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$

Algorithm for testing dependency preservation.

1. **for** each dependency $X \rightarrow Y$ in F **do**
2. compute X_G^+ using above algorithm.
3. **if** $Y \not\subset X_G^+$ **return no**
4. **return yes**

Example

- $R = (A, B, C, D)$. Dependencies

$$A \to B \qquad B \to C \qquad C \to D \qquad D \to A$$

- Decomposition

$$\{AB, BC, CD\}$$

- Is it dependency preserving?
- At first sight it seems that the dependency $D \to A$ is lost but this is not true. Compute $\{D\}^+_G$, where $G = (\pi_{AB}(F) \cup \pi_{BC}(F) \cup \pi_{CD}(F))$.

$R = (A, B, C, D)$. Dependencies

$$A \to B \qquad B \to C \qquad C \to D \qquad D \to A$$

Decomposition $\{AB, BC, CD\}$

1. Initially $Z = \{D\}$.
2. AB-operation on Z. This is $D \cup ((D \cap AB)^+ \cap AB) = D \cup \phi = D$.
3. BC-operation on Z gives D (since $Z \cap BC = \phi$).
4. CD-operation on Z gives
$$Z = D \cup (D^+ \cap CD) = D \cup (ABCD \cap CD) = CD$$
5. AB-operation on Z gives no change since $CD \cap AB = \phi$.
6. BC-operation on Z gives
$$Z = CD \cup ((CD)^+ \cap (BC)) = CD \cup (ABCD \cap BC) = BCD$$
7. CD operation gives no change.
8. AB-operation gives $Z = ABCD$.
9. Hence $D \to A$ is preserved.

Correctness Proof: Outline

- Recall $G = (\cup_{i=1}^{k} \pi_{R_i}(F))^+$.
- Each time an attribute is added to Z, we are using a dependency of G.
- So if the algorithm says "yes", it must be correct.
- Conversely, suppose $X \to Y$ is in G^+.
- Then, there is a sequence of steps of the closure algorithm (covered earlier) to take the closure of X with respect to G, we eventually include all the attributes of Y.
- Each of these steps involves the application of a dependency in G, and so is a dependency in $\pi_{R_i}(F)$ for some R_i.
- Suppose $U \to V$ be one such dependency.
- By induction on the number of steps of the closure algorithm, we can show that eventually U becomes a subset of Z and then on the next pass, the R_i-operation will add V to Z (if they are not there already)

Dependency Preserving Decomposition into 3NF

- We now give an algorithm to obtain a dependency preserving decomposition that is in 3NF.
- *Input*: Relation scheme R and set of functional dependencies F that forms a minimal cover (also called canonical cover).
- *Output*: A dependency-preserving decomposition of R such that each relation scheme is in 3NF with respect to the projection of F onto the scheme.
- For initial simplicity, assume that all *RHS* of dependencies in F are of the form $X \rightarrow A$, where, A is a single attribute.

Dependency Preserving decomposition

Algorithm:

- If there are any attributes not involved in any dependency of F, create a relation schema with these attributes.
- If one of the dependencies of F involves all the attributes of R, then output R and terminate.
- Otherwise, for each dependency of the form $X \to A$, create a fragment with schema XA.
- If there are multiple dependencies $X \to A_1, X \to A_2, \ldots, X \to A_n$ in F, then, combine these into the schema $XA_1A_2 \ldots A_n$ instead of XA_i, $i = 1, 2 \ldots, n$.

Example

Schema is $CTHRSG$, where, C = course, T = teacher, H = hour, R = room, S = student and G = grade.

$$F = \begin{aligned} C &\to T \\ HR &\to C \\ HT &\to R \\ CS &\to G \\ HS &\to R \end{aligned}$$

- F is a minimal cover.
- Dependency preserving decomposition is therefore

$$\begin{array}{ccc} CT & HRC & HTR \\ CSG & HSR & \end{array}$$

- This is also a lossless decomposition, since HS is the only key.

Correctness Proof

- Since the projected dependencies $\bigcup_{i=1}^{k} \pi_{R_i}(F)$ covers F, dependencies are preserved.
- Suppose $Y \rightarrow B$ is in the minimal cover and results in the fragment YB.
- We have to show that $R_i = YB$ is in 3NF.
- Suppose $X \rightarrow A$ is a dependency logically implied by F and holding on $R_i = YB$ such that it violates 3NF.
- Then, either X is not a superkey for YB or A is not prime.
- We know that $XA \subset YB$.

Correctness Proof

- What do we know? YB is a fragment with $Y \to B$ being in minimal cover.
- $X \to A$ is logically implied by F and $AX \subset YB$ and X is not a superkey and Y is not prime in YB.
- Two cases: Case (i) $A = B$, Case (2) $A \neq B$.
- *Case 1*: $A = B$. So $X \subset Y$ and $X \to B$. By minimality of cover $X = Y$, otherwise, $Y - X$ would be left-extraneous or redundant in the minimal cover (contradiction).
- *Case 2*: $A \neq B$. Since Y is a superkey for YB, there is a subset $Z \subset Y$ that is a key for YB.
    1. $A \in Y$ and $A \notin Z$ since A is non-prime.
    2. Then, $Z \to B$ can replace $Y \to B$, contradicting minimality of the cover.

Lossless join, dependency preserving decompositions into 3NF

- Previous algorithm did not give guarantees about the decomposition being lossless.
- Following method finds a decomposition that is lossless join and dependency preserving and is in 3NF.

Method:

1. Let ρ be the 3NF decomposition of R constructed by the previous algorithm.
2. Let X be a key for R.
3. Return $\rho \cup \{X\}$.

Proof of Correctness

- X is in 3NF. Suppose $Y \to A$ be a non-trivial dependency that holds in F and $YA \subset X$.
- Then, $X - \{A\}$ is a key for X and hence for R, contradicting that X is a key.
- Hence there cannot be any non-trivial dependencies in X.

Correctness Proof contd.

- To show that ρ has a lossless join property, apply the tabular test.
- We will show that the row corresponding to X becomes all a's.
- Proof is by induction on the order of the attributes $A_1, A_2, \ldots, A_k$ in which the attributes of $R - X$ are added to X^+ by the algorithm for computing the closure.
- By induction on i that the column corresponding to A_i for the row X is set to a_i.
- Basis: $i = 0$. Then, all columns corresponding to attributes of X are set to a's.

Correctness Proof

- Assume the result for $i-1$.
- Then, A_i is added to X^+ due to some given functional dependency $Y \to A_i$ where
$$Y \subseteq X \cup \{A_1, \ldots, A_{i-1}\}$$
- Now, YA_i is in ρ and the rows of YA_i and X agree on Y – they are all a's after the columns of X-row for $A_1, \ldots A_{i-1}$ are made a's (induction).
- Thus, in this step of the tabular algorithm, these rows are made to agree on A_i.
- Since the YA_i row has a_i, so must the X-row.