Database Management Systems (DBMS)

Lec 18- FDs: Inference Rules, Equivalence, and Minimal Cover (Contd.)

Ramesh K. Jallu
IIIT Raichur

Date: 03/04/21

Recap

- Functional dependency
  - Inference rules
    - Primary and secondary rules
  - Equivalence
  - Minimal cover

- *True* or *false*: Justify your answer
  i. $\{X \rightarrow A, Y \rightarrow B\} \models XY \rightarrow AB$
  ii. $XY \rightarrow A \models X \rightarrow A$ or $Y \rightarrow A$

Today's plan

- Algorithm to compute a minimal cover
- Algorithm to find a candidate key
- Dependency-preserving
- Algorithm to test NJ property

Minimal cover (Recap)

- A ***minimal cover*** of a set of FDs ***E*** is a set of FDs ***F*** that satisfies the property that every dependency in ***E*** is in the closure ***F***$^+$ of ***F***
  - In addition, this property is lost if any dependency from the set ***F*** is removed

- Ex: Let $E = \{B \rightarrow A,\ D \rightarrow A,\ AB \rightarrow D\}$
  - The minimal cover of E is $F = \{B \rightarrow D,\ D \rightarrow A\}$
  - The closure of F, $F+ = \{B \rightarrow D,\ D \rightarrow A,\ B \rightarrow A,\ AB \rightarrow D,\ \ldots\}$

1. *$B \rightarrow D$ (Given)*
2. *$D \rightarrow A$ (Given)*
3. *$AB \rightarrow AD$ (By agumenting A both sides on 1)*
4. *$AD \rightarrow AA$ (By agumenting A both sides on 2)*
5. *$AB \rightarrow A$ (by transtivity on 3 and 4)*
6. *$AB \rightarrow AB$ (By agumenting B bothsides on 5)*
7. *$AB \rightarrow B$ (By reflexive on 6), and $AB \rightarrow D$ follows from transtive on 1 and 7*

Extraneous attribute

- An attribute in an FD is considered an ***extraneous attribute*** if we can remove it without changing the closure of the set of dependencies
- i.e., given set of FDs F, and an FD $X \rightarrow A$ in F, attribute Y is extraneous in X if $Y \in X$, and F logically implies $(F - (X \rightarrow A) \cup \{(X - Y) \rightarrow A\})$
- We will use the concept of an extraneous attribute for defining the minimum cover

Conditions for a set of FDs *F* to be minimal

1. Every dependency in ***F*** has a single attribute for its right-hand side

2. We cannot replace any dependency $X \rightarrow A$ in ***F*** with a dependency $Y \rightarrow A$, where ***Y*** is a proper subset of ***X***, and still have a set of dependencies that is equivalent to ***F***

3. We cannot remove any dependency from ***F*** and still have a set of dependencies that is equivalent to ***F***

- An set of FDs obeying Condition 1 is said to be in ***standard form*** or ***canonical form*** with no redundancies

Finding a Minimal Cover *F* for a Set of FDs *E*

Algorithm: ***Input***: A set of FDs *E*; ***Output***: A minimal cover *F* of *E*

1. Set $F := E$

2. Replace each FD $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$

3. For each FD $X \rightarrow A$ in F for each attribute B that is an element of X if $\{\{F - \{X \rightarrow A\}\} \cup \{(X - \{B\}) \rightarrow A\}\}$ is equivalent to F then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F

4. For each remaining FD $X \rightarrow A$ in F if $\{F - \{X \rightarrow A\}\}$ is equivalent to F, then remove $X \rightarrow A$ from F

Example

- Let the given set of FDs be G: $\{A \rightarrow BCDE, CD \rightarrow E\}$
  1. $F := G$
  2. $F = \{A \rightarrow B, A \rightarrow \text{C}, A \rightarrow \text{D}, A \rightarrow \text{E}, CD \rightarrow E\}$
  3. Check for extraneous attributes in $CD \rightarrow E$
     - *Can we remove either C or D from* $CD \rightarrow E$*?*
  4. Check for redundant FDs
     - $A \rightarrow \text{E}$ is redundant
     - Are there any other redundant FDs?
  5. Return $\text{F} = \{A \rightarrow B, A \rightarrow \text{C}, A \rightarrow \text{D}, CD \rightarrow E\} \equiv \{A \rightarrow B\text{CD}, CD \rightarrow E\}$

Algorithm to find a candidate key

- Recall that a ***candidate key*** of a relation schema ***R*** is a subset ***X*** of the attributes of ***R*** with the following two properties

  a. ***X*** functionally determines every attribute in ***R***
    - I.e., $X^+ = R$
  b. No proper subset of ***X*** has the property (a)

- **Algorithm**: ***Input***: A relation ***R*** and a set of FDs ***F*** defined on ***R***;
  Output: A key ***K*** of ***R***

1. Set $K := R$
2. For each attribute ***A*** in ***K***
   - Compute $(K - A)^+$ with respect to ***F***
     - if $(K - A)^+$ contains all the attributes in ***R***, then set $K := K - A$

How to find all candidate keys?

- The algorithm determines only ***one key*** out of the possible candidate keys for ***R***
- The key returned is ***not guaranteed*** to contain the smallest number of attributes
  - The key returned depends on the order in which attributes are removed from ***R*** in Step 2
- We can compute all the candidate keys by enumerating all the subsets of attributes in ***R*** and checking if a subset's closure is ***R***
  - It works but inefficient

Look for necessary and useless attributes

- Focus on the ***necessary*** attributes that appear in ***every*** candidate key of ***R***
  - The attributes that never occur in the RHS of the FDs in ***F***
- Ignore ***useless*** attributes that are not part of any candidate key of ***R***
  - The attributes that occur only in the RHS of the FDs in ***F***
- Attributes of ***R*** that are neither necessary nor useless are called ***middle-ground*** attributes
- Ex: $R = (A, B, C, D, E, G)$ with set of FDs $F = \{AB \rightarrow C, C \rightarrow D, AD \rightarrow E\}$
  - A, B, G are necessary, E is useless, and C,D are middle-ground attributes

Observations

1. Necessary attributes are the attributes which are common in all candidate keys

2. The useless attributes can never be part of any candidate key

3. If K is the collection of all the necessary attributes and if $K^+ = R$, then K is the ***only*** candidate key of R. Why?

4. If $K^+ \neq R$, then we need to enumerate subsets X_i of R such that
   - ***i.*** X_i should not contain any useless attributes, and
   - ***ii.*** X_i should contain all the necessary attributes

Algorithm

Input: A relation R and a set of FDs F; ***Output***: The set of all candidate keys $K = \{K_1, K_2, \ldots, K_l\}$

1. Find the minimal cover F' of F
2. Partition attribute of R into necessary (N), useless (U), and middle-ground (M) attributes
3. If $N^+ = R$, then $K = \{N\}$ and terminate
4. List L the subsets of M in ascending order and add N to every subset
5. While $L \neq$ Empty
   - For every subset L_i in L
     - if the closure of the subset is R, i.e., $L_i^+ = R$
       - $K = K \cup L_i$
       - Delete every superset of L_i in L

Properties of normalization (Recap)

1. **The nonadditive join** or **lossless join property**
   - which guarantees that the ***spurious tuple generation problem*** does not occur with respect to the relation schemas created after decomposition

2. **The dependency preservation property**
   - which ensures that each functional dependency is represented in some individual relation resulting after decomposition

Dependency preservation propery of a decomposition

- We want to preserve the dependencies because each dependency represents a constraint on the database
- Let R be a database schema with relations $R_1, R_2, \ldots, R_m$, and let F be a set of FDs defined on R
- Each FD $X \rightarrow Y$ specified in F either appeared directly in one of R_i or could be inferred from the dependencies that appear in some R_i
  - I.e., the dependencies specified in F appear themselves in individual relations or they appear in the union of the dependencies that hold on the individual relations

Dependency preservation propery of a decomposition (Contd.)

- The ***projection*** of ***F*** on R_i is the set of FDs in F^+ such that all the LHS and RHS attributes of those dependencies are in R_i
- It is denoted by $\pi_{Ri}(F)$
- We say that a decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R is ***dependency-preserving*** with respect to F if the union of the projections of F on each R_i in D is equivalent to F
  - that is, $((\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$
- If a decomposition is not dependency-preserving, some dependency is ***lost*** in the decomposition

Ex.: Decomposition of Relations not in BCNF (Recap)

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

FD1: {Student, Course} → Instructor
FD2: Instructor → Course

1. R_1 (**Student**, **Instructor**) and R_2(**Student**, **Course**)
2. R_1 (Course, **Instructor**) and R_2(**Course**, **Student**)
3. R_1 (**Instructor**, Course) and R_2(**Instructor**, **Student**)

Nonadditive join property (Recap)

- A decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R has the ***lossless (nonadditive) join property*** with respect to the set of dependencies F on R, if for every relation state r of R that satisfies F, $\pi_{R1}(r) * \pi_{R2}(r) * \ldots * \pi_{Rm}(r) = r$

- ***NJBD testing***: A decomposition $D = \{R1, R2\}$ of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R if and only if either

  i. The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or

  ii. The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+

Algorithm for testing nonadditive join property

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R

2. Set $S(i, j)$: = b_{ij} for all matrix entries, where each b_{ij} is a distinct symbol associated with indices (i, j)

3. For each row i representing relation schema R_i {for each column j representing attribute A_j
   - if (relation R_i includes attribute A_j) then set $S(i, j)$: = a_j, *where* each a_j is a distinct symbol associated with index j

4. Repeat the following loop until a complete loop execution results in no changes to S

For each $X \rightarrow Y$ in F

For all rows in S which has the same symbols in the columns corresponding to attributes in X

make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: if any of the rows has an "a" symbol for the column, set the other rows to the same "a" symbol in the column. If no "a" symbol exists for the attribute in any of the rows, choose one of the "b" symbols that appear in one of the rows for the attribute and set the other rows to that same "b" symbol in the column

5. If a row is made up entirely of "a" symbols,
   - then the decomposition has the lossless join property;
   - otherwise it does not

R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours}

$D = \{R_1, R_2, R_3\}$

R_1 = EMP = {Ssn, Ename}

R_2 = PROJ = {Pnumber, Pname, Plocation}

R_3 = WORKS_ON = {Ssn, Pnumber, Hours}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

Another example

R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours}

$D = \{R_1, R_2\}$

R_1 = EMP_LOCS = {Ename, Plocation}

R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

Thank you!