Normalization

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Canonical Cover

- Suppose that we have a set of functional dependencies *F* on a relation schema
- Whenever a user performs an update on the relation, the database system must ensure that the update does not violate any functional dependencies; that is, all the functional dependencies in *F* are satisfied in the new database state
- If an update violates any functional dependencies in the set *F*, the system must roll back the update
- We can reduce the effort spent in checking for violations by testing a simplified set of functional dependencies that has the same closure as the given set
- This simplified set is termed the **canonical cover**
- To define canonical cover, we must first define **extraneous attributes**
  - An attribute of a functional dependency in *F* is **extraneous** if we can remove it without changing F^+

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - E.g.: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g. on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to
      $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
    - *In the forward:* (1) $A \rightarrow CD \Rightarrow A \rightarrow C$ and $A \rightarrow D$; (2) $A \rightarrow B$, $B \rightarrow C \Rightarrow A \rightarrow C$; $\mathbf{A^+ = ABCD}$
    - *In the reverse:* (1) $A \rightarrow B$, $B \rightarrow C \Rightarrow A \rightarrow C$; (2) $A \rightarrow C$, $A \rightarrow D \Rightarrow A \rightarrow CD$; $\mathbf{A^+ = ABCD}$

    - E.g. on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to
      $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
    - *In the forward:* (1) $A \rightarrow B$, $B \rightarrow C \Rightarrow A \rightarrow C \Rightarrow A \rightarrow AC$ and $A \rightarrow D$; (2) $A \rightarrow AC$, $AC \rightarrow D \Rightarrow A \rightarrow D$; $\mathbf{A^+ = ABCD}$
    - *In the reverse:* (1) $A \rightarrow D \Rightarrow AC \rightarrow D$; $\mathbf{AC^+ = ABCD}$

  - Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to *F*, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes

- Removing an attribute from the left side of a functional dependency could make it a stronger constraint
  - For example, if we have $AB \rightarrow C$ and remove B, we get the possibly stronger result $A \rightarrow C$
  - It may be stronger because $A \rightarrow C$ logically implies $AB \rightarrow C$, but $AB \rightarrow C$ does not, on its own, logically imply $A \rightarrow C$

- But, depending on what our set *F* of functional dependencies happens to be, we may be able to remove B from $AB \rightarrow C$ safely
  - For example, suppose that $F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow C\}$
  - Then we can show that *F* logically implies $A \rightarrow C$, making extraneous in $AB \rightarrow C$

Extraneous Attributes

- Removing an attribute from the right side of a functional dependency could make it a weaker constraint
  - For example, if we have $AB \rightarrow CD$ and remove C, we get the possibly weaker result $AB \rightarrow D$
  - It may be weaker because using just $AB \rightarrow D$, we can no longer infer $AB \rightarrow C$
- But, depending on what our set F of functional dependencies happens to be, we may be able to remove C from $AB \rightarrow CD$ safely
  - For example, suppose that $F = \{AB \rightarrow CD, A \rightarrow C\}$
- Then we can show that even after replacing $AB \rightarrow CD$ by $AB \rightarrow D$, we can still infer $AB \rightarrow C$ and thus $AB \rightarrow CD$

Extraneous Attributes

- An attribute of a functional dependency in F is **extraneous** if we can remove it without changing F^+

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F
  - **Remove from the left side**: Attribute A is **extraneous** in α if
    - $A \in \alpha$ and
    - F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$

  - **Remove from the right side**: Attribute A is **extraneous** in β if
    - $A \in \beta$ and
    - The set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F

Note: Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one

Testing if an Attribute is Extraneous

- Let R be a relation schema and let F be a set of functional dependencies that hold on R

- Consider an attribute in the functional dependency $\alpha \rightarrow \beta$

- To test if attribute $A \in \beta$ is extraneous in β
  - Compute α^+ using only the dependencies in the set F':

$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$

  - Check that α^+ contains A; if it does, A is extraneous in β

- To test if attribute $A \in \alpha$ is extraneous in α
  - Let $\gamma = \alpha - \{A\}$. Check if $\gamma \rightarrow \beta$ can be inferred from F
    - Compute γ^+ using the dependencies in F
    - If γ^+ includes all attributes in β then, A is extraneous in α

Testing if an Attribute is Extraneous

- Let $F = \{AB \rightarrow CD, A \rightarrow E, E \rightarrow C\}$

- To check if C is extraneous in $AB \rightarrow CD$, we:
  - Compute the attribute closure of AB under $F' = \{AB \rightarrow D, A \rightarrow E, E \rightarrow C\}$
  - The closure is *ABCDE,* which includes *CD*
  - This implies that *C* is extraneous

Canonical Cover

A **canonical cover** for *F* is a set of dependencies F_c such that

- *F* logically implies all dependencies in F_c , and
- F_c logically implies all dependencies in *F,* and
- No functional dependency in F_c contains an extraneous attribute, and
- Each left side of functional dependency in F_c is unique
- That is, there are no two dependencies in F_c such that
  - $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that
  - $\alpha_1 = \alpha_2$

Canonical Cover

- To compute a canonical cover for *F*:

repeat

Use the union rule to replace any dependencies in *F* of the form

$$\alpha_1 \rightarrow \beta_1 \text{ and } \alpha_1 \rightarrow \beta_2 \text{ with } \alpha_1 \rightarrow \beta_1 \beta_2$$

Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

/* Note: test for extraneous attributes done using $F_{c,}$ not F */

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until (F_c not change)

Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Example: Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - Yes: In fact, $B \rightarrow C$ is already present!
  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$

- C is extraneous in $A \rightarrow BC$

- Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
  - Yes: Using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C
    - Can use attribute closure of A in more complex cases

- The canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

Equivalence of Sets of Functional Dependencies

- Let F and G are two functional dependency sets
  - These two sets are equivalent if $F^+ = G^+$
  - Equivalence means that every functional dependency in F can be inferred from G and every functional dependency in G can be inferred from F

- Let F and G are two functional dependency sets
  - F covers G: All the functional dependency of G are logically the members of functional dependency set F $\Rightarrow G \subseteq F$
  - G covers F: All the functional dependency of F are logically the members of functional dependency set $G \Rightarrow F \subseteq G$

Condition	Cases			
F covers G	True	True	False	False
G covers F	True	False	True	False
Result	$F = G$	$G \subset F$	$F \subset G$	No comparison

Lossless Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless decomposition if at least one of the following dependencies is in F^+:

$$R_1 \cap R_2 \rightarrow R_1$$
$$R_1 \cap R_2 \rightarrow R_2$$

- The above functional dependencies are a sufficient condition for lossless join decomposition
- The dependencies are a necessary condition only if all constraints are functional dependencies

- To identify whether a decomposition is lossless or lossy, it must satisfy the following conditions:
  - $R_1 \cup R_2 = R$
  - $R_1 \cap R_2 \neq \varnothing$ and
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Lossless Decomposition

- Consider **Supplier_Parts** schema: *Supplier_Parts(S#, Sname, City, P#, Qty)*
- Having dependencies: *S#* $\rightarrow$ *Sname, S#* $\rightarrow$ *City, (S#, P#)* $\rightarrow$ *Qty*
- Decompose as: *Supplier(S#, Sname, City, Qty), Parts(P#, Qty)*
- Take *Natural Join* to reconstruct: ***Supplier*** $\bowtie$ ***Parts***
  - We get extra tuples! Join is lossy!
  - Common attribute ***Qty*** is not a superkey in **Supplier** or in **Parts**
  - Doesn't preserve *(S#, P#)* $\rightarrow$ *Qty*

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

P#	Qty
301	20
500	50
20	10
400	40
301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
5	**Nick**	**NY**	**20**	**10**
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10
2	**Steve**	**Boston**	**301**	**10**

Lossless Decomposition

- Consider **Supplier_Parts** schema: *Supplier_Parts(S#, Sname, City, P#, Qty)*
- Having dependencies: *S#* $\rightarrow$ *Sname, S#* $\rightarrow$ *City, (S#, P#)* $\rightarrow$ *Qty*
- Decompose as: *Supplier(S#, Sname, City), Parts(S#, P#, Qty)*
- Take *Natural Join* to reconstruct: ***Supplier*** $\bowtie$ ***Parts***
  - We get back the original relation! Join is lossless!
  - Common attribute ***S#*** is the superkey in **Supplier**
  - Preserve all the dependencies

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City
3	Smith	London
5	Nick	NY
2	Steve	Boston
5	Nick	NY
5	Nick	NY

S#	P#	Qty
3	301	20
5	500	50
2	20	10
5	400	40
5	301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i
- A decomposition is **dependency preserving**, if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
- If is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive
- Using the above definition, testing for dependency preservation take exponential time
- Not that if a decomposition is NOT dependency preserving then checking updates for violation of functional dependencies may require computing joins, which is expensive
- Let R be the original relational schema having set of FD F
- *Let* R_1 and R_2 having the FD set F_1 and F_2 respectively, are the decomposed subrelation of R
- The decomposition of R is said to be preserving if :
  - $F_1 \cup F_2 \equiv F$ (decomposition reserving dependencies)
  - If $F_1 \cup F_2 \subset F$ (decomposition NOT preserving dependencies) and
  - $F_1 \cup F_2 \supset F$ (this is not possible)

Dependency Preservation

- Let F be the set of dependencies on schema R and let R_1, R_2 , .., R_n be a decomposition of R
- The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include **only** attributes of R_i
- Since all functional dependencies in a restriction involve attributes of only one relation schema, it is possible to test such a dependency for satisfaction by checking only one relation
- Note that the definition of restriction uses all dependencies in F^+, not just those in F
- The set of restrictions F_1, F_2 , … , F_n is the set of functional dependencies that can be checked efficiently

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$, we apply the following test (with attribute closure done with respect to F)
  - *result* = α

    repeat

    for each R_i in the decomposition

    $t = (result \cap R_i)^+ \cap R_i$

    $result = result \cup t$

    until (*result* does not change)

  - If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved
- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example

- $R = (A, B, C, D, E, F)$
  $F = \{A \rightarrow BCD, A \rightarrow EF, BC \rightarrow AD, BC \rightarrow E, BC \rightarrow F, B \rightarrow F, D \rightarrow E\}$
  D = $\{ABCD, BF, DE\}$

- On projections:

ABCD (R1)	BF(R2)	DE(R3)
$A \rightarrow BCD$	$B \rightarrow F$	$D \rightarrow E$
$BC \rightarrow AD$		

- Need to check for: $\cancel{A \rightarrow BCD}$, $\mathbf{A \rightarrow EF}$, $\cancel{BC \rightarrow AD}$, $\mathbf{BC \rightarrow E}$, $\mathbf{BC \rightarrow F}$, $\cancel{B \rightarrow F}$, $\cancel{D \rightarrow E}$
- $(BC)^+ /F_1 = ABCD$, $(ABCD)^+ /F_2 = ABCDF$, $(ABCDF)^+ /F_3 = ABCDEF$, Preserves $\mathbf{BC \rightarrow E}$, $\mathbf{BC \rightarrow F}$
- $(A)^+ /F_1 = ABCD$, $(ABCD)^+ /F_2 = ABCDF$, $(ABCDF)^+ /F_3 = ABCDEF$, Preserves $\mathbf{A \rightarrow EF}$

Example

- $R = (A, B, C, D, E, F)$; $F = \{A \to BCD, A \to EF, BC \to AD, BC \to E, BC \to F, B \to F, D \to E\}$
- On projections:

ABCD (R1)	BF(R2)	DE(R3)
$A \to B, A \to C, A \to D$	$B \to F$	$D \to E$
$BC \to A, BC \to D$		

- Infer reverse FDs:
  - $B^+ /F = BF$: $A \to B$ can not be inferred
  - $C^+ /F = C$: $C \to A$ can not be inferred
  - $D^+ /F = DE$: $D \to A$ and $D \to BC$ can not be inferred
  - $A^+ /F = ABCDEF$: $A \to BC$ can be inferred, but it is equal to $A \to B$ and $A \to C$
  - $F^+ /F = F$: $F \to B$ can not be inferred
  - $E^+ /F = E$: $E \to B$ can not be inferred
- Need to check for: $\cancel{A \to BCD}$, $\mathbf{A \to EF}$, $\cancel{BC \to AD}$, $\mathbf{BC \to E}$, $\mathbf{BC \to F}$, $\cancel{B \to F}$, $\cancel{D \to E}$
- $(BC)^+ /F = ABCDEF$, Preserves $\mathbf{BC \to E}$, $\mathbf{BC \to F}$
- $(A)^+ /F = ABCDEF$, Preserves $\mathbf{A \to EF}$

Next Lecture

Normalization

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in