Part 6:

Functional Dependency Theory

Database System Concepts, 7th Ed.

Closure of a Set of Functional Dependencies

- We have seen that the set of **all** functional dependencies logically implied by *F* is the **closure** of *F*, and *we* denote the *closure* of *F* by F^+
- We can compute F^+ by repeatedly applying **Armstrong's Axioms:**
  - **Reflexive rule:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation rule:** if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity rule:** if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These rules are
  - **Sound** -- generate only functional dependencies that actually hold, and
  - **Complete** -- generate all functional dependencies that hold

Closure of Functional Dependencies

- Additional rules:
  - **Union rule**: If $\alpha \rightarrow \beta$ holds *and* $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition rule**: If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity rule**: If $\alpha \rightarrow \beta$ holds *and* $\gamma\beta \rightarrow \delta$ holds, then $\gamma\alpha \rightarrow \delta$ holds

Example of F^+

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Show that the following are members of F^+
  - $A \rightarrow H$
  - $AG \rightarrow I$
  - $CG \rightarrow HI$
  - $AG \rightarrow I$ (another proof)

Example of F^+

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- Show that the following are members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$ (augmentation proof)
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - Since $CG \rightarrow I$ and $CG \rightarrow H$, the union rule implies that $CG \rightarrow HI$
  - $AG \rightarrow I$ (pseudotransitivity proof)
    - Since $A \rightarrow C$ and $CG \rightarrow I$, the pseudotransitivity rule implies that $AG \rightarrow I$ holds

Procedure for Computing F^+

- The following computes the closure of a set of functional dependencies F

```
F + = F
apply the reflexivity rule  /* Generate all trivial dependencies */
repeat
    for each functional dependency f in F+
        apply the augmentation rule on f
        add the resulting functional dependencies to F +
    for each pair of functional dependencies f1 and f2 in F +
        if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F +
until F + does not change any further
```

Closure of Attribute Sets

- Given a set of attributes α, define the ***closure*** of α **under** *F* (denoted by α^+) as the set of attributes that are functionally determined by α under *F*
- Algorithm to compute α^+, the closure of α under *F*

> *result* := α;
> **repeat**
>   **for each** functional dependency $\beta \rightarrow \gamma$ **in** *F* **do**
>   **begin**
>   **if** $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$;
>   **end**
> **until** (*result* does not change)

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- $(AG)^+$
  1. *result* = *AG*
  2. *result* = *ABCG* $(A \rightarrow C$ and $A \rightarrow B)$
  3. *result* = *ABCGH* $(CG \rightarrow H$ and $CG \subseteq ABCG)$
  4. *result* = *ABCGHI* $(CG \rightarrow I$ and $CG \subseteq ABCGH)$
- Is *AG* a superkey?

  Does $AG \rightarrow R$?

  i.e., Is $(AG)^+ \supseteq R$?

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute $\alpha^{+,}$ and check if α^{+} contains all attributes of *R*
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^{+}), just check if $\beta \subseteq \alpha^{+}$
  - That is, we compute α^{+} by using attribute closure, and then check if it contains β
- Computing closure of *F*
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$

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^+

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 $A \rightarrow C$
- 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 B 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 $AB \rightarrow C$
- 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
    - $F \Rightarrow (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\} = F'$,

      replacing the functional dependency $\alpha \rightarrow \beta$ by a new functional dependency by taking out A from the left-hand side
    - i.e., it is assumed possible to replace a weaker FD by a stronger FD
  - **Remove from the right side**: Attribute A is **extraneous** in β if
    - $A \in \beta$ and
    - The set of functional dependencies

      $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\} \Rightarrow F$$
    - i.e., it is assumed possible to replace a stronger FD by a weaker FD
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" FD always implies a "weaker" one

Extraneous Attributes

Removal from Left	Removal from Right
Original:	Original:
F: $A\underline{B} \rightarrow C$, $A \rightarrow D$, $D \rightarrow C$ (weaker)	F: $AB \rightarrow \underline{C}D$, $A \rightarrow E$, $E \rightarrow C$ (stronger)
(i.e., $F \Rightarrow F'$) $\Downarrow$	(i.e., $F' \Rightarrow F$) $\Uparrow$
B removed:	C removed:
F': $A \rightarrow C$, $A \rightarrow D$, $D \rightarrow C$ (stronger)	F': $AB \rightarrow D$, $A \rightarrow E$, $E \rightarrow C$ (weaker)
If $F \Rightarrow F'$, then $F' \Rightarrow F$ is always true (stronger $\Rightarrow$ weaker)	If $F' \Rightarrow F$, then $F \Rightarrow F'$ is always true (stronger $\Rightarrow$ weaker)

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
  - Consider the set (i.e., removing *A* from the FD)
    $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  - check that α^+ contains *A* under *F'*; 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

Examples of Extraneous Attributes

- 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
  - $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ such that
  - $\alpha_1 = \alpha_2$

Canonical Cover

- To compute a canonical cover for *F*:

  $F_c = F$

  repeat

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

  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ 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 $\alpha \rightarrow \beta$ in F_c .

  until (F_c does 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.
- The canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

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
  - Note that the definition of restriction uses all dependencies in 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
- More precisely, let F_i be the set of dependencies in F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

- 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

Testing Decomposition for BCNF

To check if a relation R_i in a decomposition of R is in BCNF

- Either test R_i for BCNF with respect to the **restriction** of F^+ to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
- Or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
  - If α^+ includes none of the attributes of $R_i - \alpha$, that means α is not a (non-trivial) determinant within R_i
  - If α^+ includes all of the attributes of R_i, then α is a superkey of R_i

BCNF Decomposition Algorithm

result := {*R*};
done := false;
while (not *done)* **do**
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional dependency
      that holds on R_i such that α is not a superkey of R_i
        and $\alpha \cap \beta = \emptyset$;
      $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$
      end
  else *done* := **true;**

Note:

- If $\alpha \cap \beta \neq \emptyset$, e.g., $\alpha \cap \beta = \gamma$, then $(R_i - \beta)$ would exclude γ, losing the information on γ in $R_i - \beta$, and α is incomplete in $R_i - \beta$ which would be undesirable (see next example)

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

Example of BCNF Decomposition

- *course* is in BCNF but not *class-1*

  class-1 (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF
- For the sake of argument, suppose we write the FD

  building, *room_number*→*capacity* as

  building, *room_number*→*capacity, building* (i.e., $\alpha \cap \beta \neq \emptyset$)

  Then we would exclude *building* from the *section* table making it:

  section (*course_id*, *sec_id*, *semester*, *year*, *room_number*, *time_slot_id*)

  which is undesirable since the *building* information is lost

3NF or BCNF?

- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important
- We can use Third Normal Form (3NF)
  - Allows some redundancy
  - But functional dependencies can be checked on individual relations without computing a join
  - There is always a lossless-join, dependency-preserving decomposition into 3NF

3NF Example - Relation *dept_advisor*

- Consider R = *dept_advisor* (*s_ID, i_ID, dept_name*)
  $F = \{s\_ID, dept\_name \rightarrow i\_ID,\ i\_ID \rightarrow dept\_name\}$
- Two candidate keys: *s_ID, dept_name,* and *i_ID, s_ID*
- R is in 3NF
  - $s\_ID, dept\_name \rightarrow i\_ID$
    - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$
    - *dept_name* is contained in a candidate key
- R is not in BCNF, since *i_ID* is a determinant but not a superkey
- If we decompose it into

  (*i_ID, dept_name*) and (*s_ID, i_ID*)

  then the first FD cannot be easily checked
- In fact, any decomposition of R will not have all 3 attributes, and checking the first FD requires all 3 attributes to be present in the same relation

3NF Decomposition Algorithm

Let F_c be a canonical cover for *F;*
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
  if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$
    then begin
      $i := i + 1$;
      $R_i := \alpha\ \beta$
    end
if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for *R*
  then begin
    $i := i + 1$;
    $R_i :=$ any candidate key for *R;*
  end
repeat /* Optionally, remove redundant relations */
  if any schema R_j is contained in another schema R_k
    then /* Delete R_j */
      $R_j := R_i$;
      $i := i-1$;
until no more R_j*'s* can be deleted
return $(R_1, R_2, ..., R_i)$

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

  Where *type* indicates the type of account
- The functional dependencies for this relation schema are:
  - *customer_id, employee_id* $\rightarrow$ *branch_name, type*
  - *employee_id* $\rightarrow$ *branch_name*
  - *customer_id, branch_name* $\rightarrow$ *employee_id*
- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_c =

    customer_id, employee_id $\rightarrow$ *type*
    employee_id $\rightarrow$ *branch_name*
    customer_id, branch_name $\rightarrow$ *employee_id*
  - *customer_id, employee_id* is a candidate key

3NF Decomposition Example

customer_id, employee_id → *type*
employee_id → *branch_name*
customer_id, branch_name → *employee_id*

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added
- At end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas
- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - The decomposition is lossless
  - The dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - The decomposition is lossless
  - It may not be possible to preserve dependencies

Design Goals

- Goal for a relational database design is:
  - BCNF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF