Schema Refinement & Normalization Theory

Functional Dependencies & Normalization

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # of attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F. An efficient check:
  - Compute *attribute closure* of X (denoted X^+) wrt *F:*
    - Set of all attributes Z such that $X \rightarrow Z$ is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\ D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Computing X^+

- Input F (a set of FDs), and X (a set of attributes)
- Output: Result=X^+ (under F)
- Method:
  - Step 1: Result :=X;
  - Step 2: Take $Y \rightarrow Z$ in F, **and** Y is in Result, do:

    $$\text{Result} := \text{Result} \cup Z$$
  - Repeat step 2 until Result cannot be changed and then output Result.

Example of Attribute Closure X^+

- Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Step 1: Result = A

Step 2: Consider $A \rightarrow B$, Result = AB

Consider $B \rightarrow C$, Result = ABC

Consider $CD \rightarrow E$, CD is not in ABC, so stop

Step 3: $A^+ = \{ABC\}$

E is NOT in A^+, so $A \rightarrow E$ is NOT in F^+

Example of computing X^+

$F = \{A \to B, AC \to D, AB \to C\}$?

What is X^+ for $X = A$? (i.e. what is the attribute closure for A?)

Answer: $A^+ = ABCD$

Example of Attribute Closure

$R = (A, B, C, G, H, I)$

$F = \{A \rightarrow B; A \rightarrow C; CG \rightarrow H; CG \rightarrow I; B \rightarrow H\}$

- $(AG)^+ = ?$
  - Answer: ABCGHI
- Is *AG* a candidate key?
  - This question involves two parts:
    1. Is AG a super key?
       - Does $AG \rightarrow R?$ == Is $(AG)^+ \supseteq R$
    2. Is any subset of AG a superkey?
       - Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$
       - Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if X is a superkey, we compute X^+, and check if X^+ contains all attributes of *R*.
- Testing functional dependencies
  - To check if a functional dependency $X \rightarrow Y$ holds (or, in other words, is in F^+), just check if $Y \subseteq X^+$.
  - That is, we compute X^+ by using attribute closure, and then check if it contains Y.
  - Is a simple and cheap test, and very useful
- Computing closure of F

Computing F^+

- Given $F=\{ A \rightarrow B, B \rightarrow C\}$. Compute F^+ (with attributes A, B, C).

Step 1: Construct an empty matrix, with all Possible combinations of attributes in the rows And columns

	A	B	C	AB	AC	BC	ABC
A							
B							
C							
AB							
AC							
BC							
ABC							

Step 2: Compute the attribute closures for all attribute/ combination of attributes

Attribute closure
$A^+=?$
$B^+=?$
$C^+=?$
$AB^+=?$
$AC^+=?$
$BC^+=?$
$ABC^+=?$

Step 3: Fill in the matrix using the results from Step 2

Computing F^+

- Given $F=\{A \rightarrow B, B \rightarrow C\}$. Compute F^+ (with attributes A, B, C).

We'll do an example on A^+.

Step 1: Result = A

Step 2: Consider $A \rightarrow B$, Result = $A \cup B = AB$

Consider $B \rightarrow C$, Result = $AB \cup C = ABC$

Step 3: $A^+ = \{ABC\}$

Computing F^+

- Given $F=\{ A \rightarrow B, B \rightarrow C\}$. Compute F^+ (with attributes A, B, C).

Step 1: Construct an empty matrix, with all Possible combinations of attributes in the rows And columns

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B							
C							
:							

Step 2: Compute the attribute closures for all attribute/ combination of attributes

Attribute closure
A^+=**ABC**
B^+=?
C^+=?
AB^+=?
AC^+=?
BC^+=?
ABC^+=?

Step 3: Fill in the matrix using the results from Step 2.
We have A^+=ABC. Now fill in the row for A. Consider the first column. Is A part of A^+? Yes, so check it.
Is B part of A^+? Yes, so check it… and so on.

Computing F^+

- Given $F=\{ A \rightarrow B, B \rightarrow C\}$. Compute F^+ (with attributes A, B, C).

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B		√	√			√	
C			√				
AB	√	√	√	√	√	√	√
AC	√	√	√	√	√	√	√
BC		√	√			√	
ABC	√	√	√	√	√	√	√

Attribute closure
$A^+=ABC$
$B^+=BC$
$C^+=C$
$AB^+=ABC$
$AC^+=ABC$
$BC^+=BC$
$ABC^+=ABC$

- An entry with √ means FD (the row) $\rightarrow$ (the column) is in F^+.
- An entry gets √ when (the column) is in (the row)$^+$

Computing F^+

Step 4: Derive rules.

$A \rightarrow BC$

	A	B	C	AB	AC	BC	ABC
A	√	√	√	√	√	√	√
B		√	√			√	
C			√				
AB	√	√	√	√	√	√	√
AC	√	√	√	√	√	√	√
BC		√	√			√	
ABC	√	√	√	√	√	√	√

Attribute closure
A^+=ABC
B^+=BC
C^+=C
AB^+=ABC
AC^+=ABC
BC^+=BC
ABC^+=ABC

- An entry with √ means FD (the row) $\rightarrow$ (the column) is in F^+.
- An entry gets √ when (the column) is in (the row)$^+$

Check if two sets of FDs are equivalent

- Two sets of FDs are equivalent if they logically imply the same set of FDs.
  - i.e., if $F_1^+ = F_2^+$, then they are equivalent.
- For example, $F_1=\{A \rightarrow B, A \rightarrow C\}$ is equivalent to $F_2=\{A \rightarrow BC\}$
- How to test? Two steps:
  - *Every* FD in F_1 is in F_2^+
  - *Every* FD in F_2 is in F_1^+
- These two steps can use the algorithm (many times) for X^+

Summary

- Constraints give rise to redundancy
  - Three anomalies
- FD is a “popular” type of constraint
  - Satisfaction & violation
  - Logical implication
  - Reasoning
- Armstrong’s Axioms
  - FD inference/derivation
- Computing the closure of FD’s (F^+)
- Check for existence of an FD
  - By computing the Attribute closure

Normal Forms

- The first question: Is any refinement needed?
- Normal forms:
  - If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/ minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value!

Normal Forms

- First normal form (1NF)
  - Every field must contain atomic values, i.e. no sets or lists.
  - Essentially all relations are in this normal form
- Second normal form (2NF)
  - Any relation in 2NF is also in 1NF
  - All the non-key attributes must depend upon the WHOLE of the candidate key rather than just a part of it.
    - It is only relevant when the key is composite, i.e., consists of several fields.
  - e.g. Consider a relation:
    - Inventory(part, warehouse, quantity, warehouse_address).
    - Suppose {part, warehouse} is a candidate key.
    - warehouse_address depends upon warehouse alone - 2NF violation
    - Solution: decompose

Normal Forms

- Boyce-Codd Normal Form (BCNF)
  - Any relation in BCNF is also in 2NF
- Third normal form (3NF)
  - Any relation in BCNF is also in 3NF

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if **for each non-trivial FD** $X \rightarrow A$ in F**, X is a super key for R** (i.e., $X \rightarrow R$ in F^+).
  - An FD $X \rightarrow A$ is said to be "trivial" if $A \in X$.
  - However if not all XA are in R, then we don't care.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are *key constraints*.
- If BCNF:
  - No "data" in R can be predicted using FDs alone. Why:
  - Because X is a (super)key, we can't have two different tuples that agree on the X value

X	Y	A
x	y1	a
x	y2	?

Suppose we know that this instance satisfies $X \rightarrow A$. This situation cannot arise if the relation is in BCNF.

BCNF

- Consider relation R with FDs F. If $X \rightarrow A$ in F over R violates BCNF, it means
  - XA are all in R, <u>and</u>
  - A is not in X, <u>and</u> $\rightarrow$ non-trivial FD
  - $X \rightarrow R$ is not in F^+ $\rightarrow$ X is not a superkey
- In other words, for $X \rightarrow A$ in F over R to satisfy BCNF requirement, at least one of the followings must be true:
  - XA are not all in R, <u>or</u>
  - $X \rightarrow A$ is trivial, i.e. A is in X, <u>or</u>
  - X is a superkey, i.e. $X \rightarrow R$ is in F^+

Decomposition of a Relation Schema

- When a relation schema is not in BCNF: **decompose**.
- Suppose that relation R contains attributes *A1 … An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of at least one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.

Decomposition example

Original relation (not stored in DB!)

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Decomposition (in the DB)

=

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

$\bowtie$

R	W
8	10
5	7

Problems with Decompositions

- There are three potential problems to consider:
  - ❶ Some queries become more expensive.
    - e.g., How much did sailor Attishoo earn? (earn = W*H)
  - ❷ Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - ❸ Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- *Tradeoff*: Must consider these issues vs. redundancy.

Example of problem 2

Student_ID	Name	Dcode	Cno	Grade
123-22-3666	Attishoo	INFS	501	A
231-31-5368	Guldu	CS	102	B
131-24-3650	Smethurst	INFS	614	B
434-26-3751	Guldu	INFS	614	A
434-26-3751	Guldu	INFS	612	C

$\neq$

Name	Dcode	Cno	Grade
Attishoo	INFS	501	A
Guldu	CS	102	B
Smethurst	INFS	614	B
Guldu	INFS	614	A
Guldu	INFS	612	C

$\bowtie$

Student_ID	Name
123-22-3666	Attishoo
231-31-5368	Guldu
131-24-3650	Smethurst
434-26-3751	Guldu