Schema Refinement & Normalization Theory

Functional Dependencies

Week 13

What's the Problem

- Consider relation obtained (call it SNLRHW)

  Hourly_Emps(<u>*ssn,*</u> *name, lot, rating, hrly_wage*, *hrs_worked*)

- What if we ***know*** rating determines hrly_wage?

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

Redundancy

- When part of data can be derived from other parts, we say *redundancy* exists.
  - Example: the hrly_wage of Smiley can be derived from the hrly_wage of Attishoo because they have the same rating and we know rating determines hrly_wage.
- Redundancy exists because of of the existence of *integrity constraints* ***(e.g., FD: $R \rightarrow W$).***

What's the problem, again

- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

What do we do?

- Since constraints, in particular *functional dependencies*, cause problems, we need to study them, and understand when and how they cause redundancy.
- When redundancy exists, refinement is needed.
  - Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

What do we do? Decomposition

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

=

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

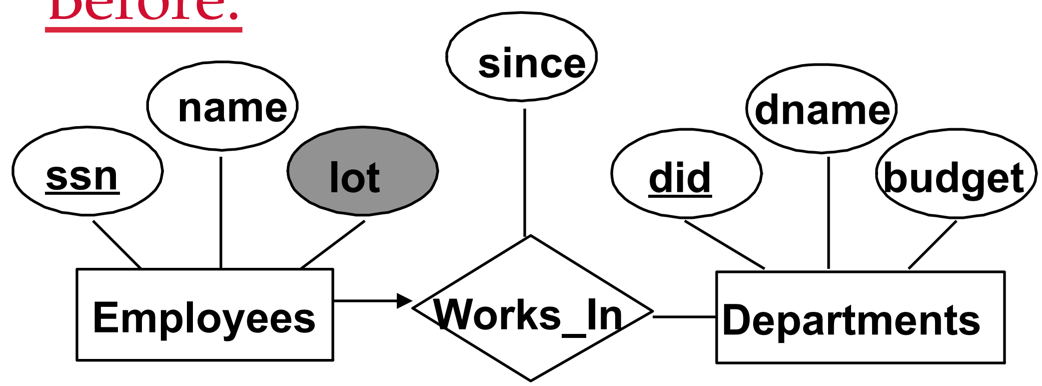
$\bowtie$

R	W
8	10
5	7

Refining an ER Diagram

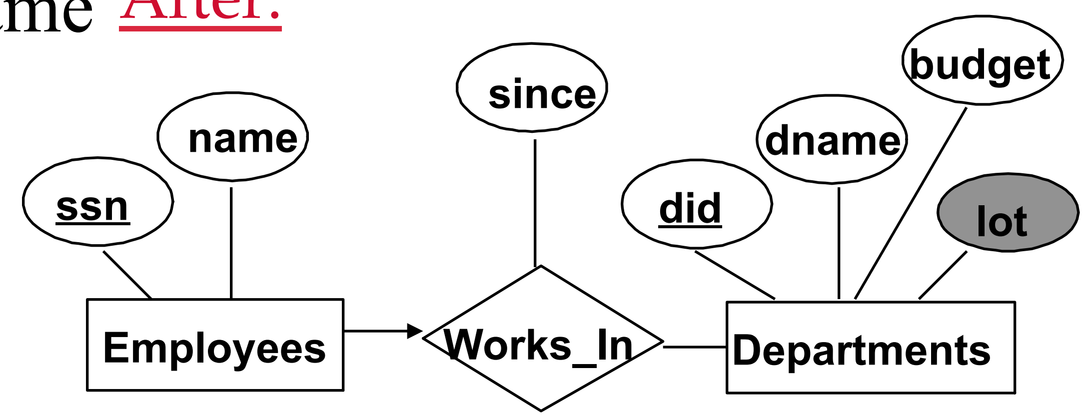
- 1st diagram translated:
  Employees(S,N,L,D,S2)
  Departments(D,M,B)
  - Lots associated with employees.

Before:



- Suppose all employees in a dept are assigned the same lot: $D \rightarrow L$
- Can fine-tune this way:
  Employees2(S,N,D,S2)
  Departments(D,M,B,L)

After:



Functional Dependencies (FDs)

- A <u>functional dependency</u> (FD) has the form: $X \rightarrow Y$, where X and Y are two *sets* of attributes.
  - Examples: rating→hrly_wage, AB →C
- The FD $X \rightarrow Y$ ***is satisfied by a relation instance r if:***
  - for each pair of tuples t1 and t2 in r:

    $t1.X = t2.X$ implies $t1.Y = t2.Y$
  - i.e., given any two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- Convention: X, Y, Z etc denote sets of attributes, and A, B, C, etc denote attributes.

Functional Dependencies (FDs)

- ***The FD holds*** over relation name R if, for every ***allowable*** instance *r* of R, *r* satisfies the FD.
- An FD, as an integrity constraint, is a statement about *all* allowable relation instances.
  - Must be identified based on semantics of application.
  - Given some instance *r1* of R, we can check if it ***violates*** some FD *f* or not
  - But we cannot tell if *f* ***holds*** over R by looking at an instance!
    - Cannot prove non-existence (of violation) out of ignorance
  - This is the same for all integrity constraints!

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*<u>ssn,</u> name, lot, rating, hrly_wage, hrs_worked*)
- *<u>Notation</u>*: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - *ssn* is the key: $S \rightarrow SNLRWH$
  - *rating* determines *hrly_wage*: $R \rightarrow W$

One more example

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FDs with A as the left side:	**Satisfied by the relation instance?**
$A \rightarrow A$	yes
$A \rightarrow B$	yes
$A \rightarrow C$	No
$A \rightarrow AB$	yes
$A \rightarrow AC$	No
$A \rightarrow BC$	No
$A \rightarrow ABC$	No

How many *possible* FDs totally on this relation instance?

Violation of FD by a relation

- The FD $X \rightarrow Y$ ***is NOT satisfied by a relation instance r if:***
  - There exists a pair of tuples t1 and t2 in r such that

    $t1.X = t2.X$ but $t1.Y \neq t2.Y$
  - i.e., we can find two tuples in *r*, such that X values agree, but Y values don't.

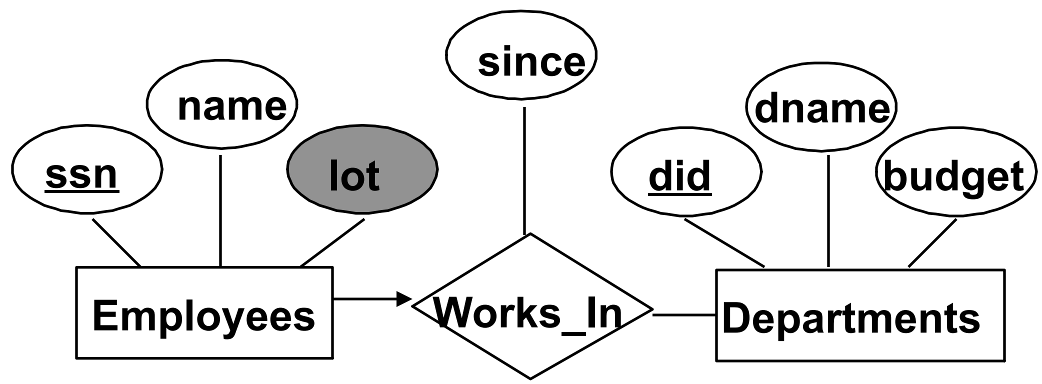
Some other FDs

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FD	**Satisfied by the relation instance?**
$C \rightarrow B$	yes
$C \rightarrow AB$	No
$B \rightarrow C$	No
$B \rightarrow B$	Yes
$AC \rightarrow B$	Yes [note!]
…	…

Relationship between FDs and Keys

- Given R(A, B, C).
  - $A \rightarrow ABC$ means that A is a key.
- In general,
  - $X \rightarrow R$ means X is a (super)key.
- How about key constraint?
  - $ssn \rightarrow did$



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did,\ did \rightarrow lot$ implies $ssn \rightarrow lot$
  - $A \rightarrow BC$ implies $A \rightarrow B$
- An FD f is *logically implied by* a set of FDs F if f holds whenever all FDs in F hold.
  - F^+ = *closure of F* is the set of all FDs that are implied by F.

Armstrong's axioms

- Armstrong's axioms are *sound* and *complete* inference rules for FDs!
  - Sound: all the derived FDs (by using the axioms) are those logically implied by the given set
  - Complete: all the logically implied (by the given set) FDs can be derived by using the axioms.

Reasoning about FDs

- How do we get all the FDs that are logically implied by a given set of FDs?
- Armstrong' s Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*:
    - If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*:
    - If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*:
    - If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

A	B	C
1	1	2
2	1	3
2	1	3
1	1	2

Example of using Armstrong's Axioms

- Couple of additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Derive the above two by using Armstrong's axioms!

Derive Union

- Show that

If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Derive Decomposition

- Show that

If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Another Useful Rule: Accumulation Rule

- If $X \rightarrow YZ$ and $Z \rightarrow W$, then $X \rightarrow YZW$

Proof:

Derivation Example

- $R = (A, B, C, G, H, I)$
  $F = \{A \rightarrow B; A \rightarrow C; CG \rightarrow H; CG \rightarrow I; B \rightarrow H \}$
- some members of F^+ (how to derive them?)
  - $A \rightarrow H$
  - $AG \rightarrow I$
  - $CG \rightarrow HI$

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$
repeat
  for each functional dependency f in F^+
    apply reflexivity and augmentation rules on f
    add the resulting functional dependencies to F^+
  for each pair of functional dependencies f_1 and f_2 in F^+
    if f_1 and f_2 can be combined using transitivity
      then add the resulting functional dependency to F^+
until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Example on Computing F+

- $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E\}$
- Step 1: For each f in F, apply reflexivity rule
  - We get: $CD \rightarrow C$; $CD \rightarrow D$
  - Add them to F:
    - $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E;\ CD \rightarrow C;\ CD \rightarrow D\}$
- Step 2: For each f in F, apply augmentation rule
  - From $A \rightarrow B$ we get: $A \rightarrow AB$; $AB \rightarrow B$; $AC \rightarrow BC$; $AD \rightarrow BD$; $ABC \rightarrow BC$; $ABD \rightarrow BD$; $ACD \rightarrow BCD$
  - From $B \rightarrow C$ we get: $AB \rightarrow AC$; $BC \rightarrow C$; $BD \rightarrow CD$; $ABC \rightarrow AC$; $ABD \rightarrow ACD$, etc etc.
- Step 3: Apply transitivity on pairs of f's
- Keep repeating… You get the idea

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:

    Result := 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^+

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

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) → (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 $A \in X$.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are <u>*key constraints*</u>.
- 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.

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

Lossless Join Decompositions

- Decomposition of R into R_1 and R_2 is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F, we have:

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = r$$

- It is always true that

$$r \subseteq \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

- In general, the other direction does not hold! If it does, the decomposition is *lossless-join*.

Example (lossy decomposition)

r

A	B	C
1	2	3
4	5	6
7	2	8

$\pi_{AB}(r)$

A	B
1	2
4	5
7	2

$\pi_{BC}(r)$

B	C
2	3
5	6
2	8

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

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Example (lossless join decomposition)

r

A	B	C
1	2	3
4	5	6
7	2	3

$\pi_{AB}(r)$

A	B
1	2
4	5
7	2

$\pi_{BC}(r)$

B	C
2	3
5	6

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

A	B	C
1	2	3
4	5	6
7	2	3

$We\ have\ (AB \cap BC) \to BC$

Lossless Join Decomposition

- The decomposition of R into R_1 and R_2 is lossless-join wrt F if and only if F^+ contains:
  - $R_1 \cap R_2 \rightarrow R_1$, or
  - $R_1 \cap R_2 \rightarrow R_2$
- In particular, the decomposition of R into (UV) and (R-V) is lossless-join if $U \rightarrow V$ holds on R
  - assume U and V do not share attributes.
  - WHY?

Decomposition

- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless!* *(Avoids Problem (2))*

Decomposition into BCNF

- Recall that for X → A in *F* over R to satisfy BCNF requirement, one of the followings must be true:
  - XA are not all in R, <u>or</u>
  - X → A is trivial, i.e. A is in X, <u>or</u>
  - X is a superkey, i.e. X → R is in F^+

- Consider relation R with FDs F. If X → A in *F* over R violates BCNF, i.e.,
  - XA are all in R, <u>and</u>
  - A is not in X, <u>and</u> → non-trivial FD
  - X → R is not in F^+ → X is not a superkey

Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow A$ in F over R violates BCNF, i.e.,
  - XA are all in R, and
  - A is not in X, and → non-trivial FD
  - $X \rightarrow R$ is not in F^+ → X is not a (super)key

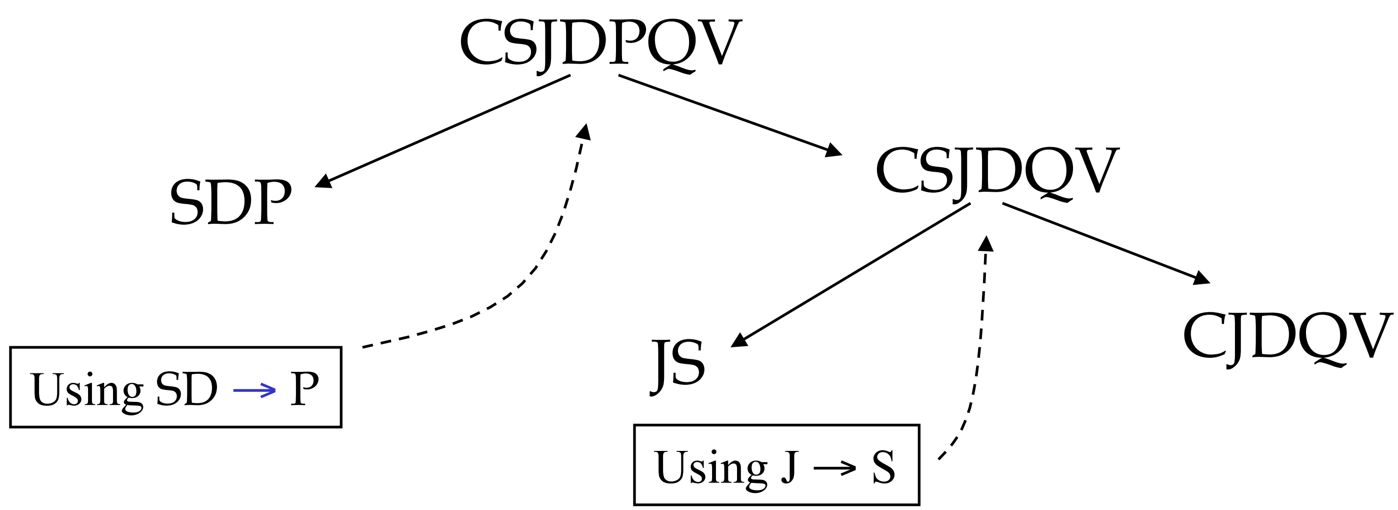
- Then: decompose R into R - A and XA.
- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.

BCNF Decomposition Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B; B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not a superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A, B)$

BCNF Decomposition Example 2

- Assume relation schema CSJDPQV:
  Contracts(*contract_id, supplier, project, dept, part, qty, value*)
  - key C, JP → C, SD → P, J → S
- To deal with SD → P, decompose into SDP, CSJDQV.
- To deal with J → S, decompose CSJDQV into JS and CJDQV
- A tree representation of the decomposition:



BCNF Decomposition

- In general, several dependencies may cause violation of BCNF. The order in which we "deal with" them could lead to very different sets of relations!

How do we know R is in BCNF?

- If R has only two attributes, then it is in BCNF
- If F only uses attributes in R, then:
  - R is in BCNF *if and only if* for each $X \rightarrow Y$ in F (***not F^+!***), X is a superkey of R, i.e., $X \rightarrow R$ is in F^+ (not F!).

Checking for BCNF Violations

- List all non-trivial FDs
- Ensure that left hand side of each FD is a superkey
- We have to first find all the keys!

Checking for BCNF Violations

- Is Courses(course_num, dept_name, course_name, classroom, enrollment, student_name, address) in BCNF?
- FDs are:
  - course_num, dept_name → course_name
  - course_num, dept_name → classroom
  - course_num, dept_name → enrollment
- What is (course_num, dept_name)$^+$?
  - {course_num, dept_name, course_name, classroom, enrollment}
- Therefore, the key is

  {course_num, dept_name, course_name, classroom, enrollment, student_name, address}
- The relation is not in BCNF