



# Schema Refinement & Normalization Theory



# What's the Problem

- Consider relation obtained (call it SNLRHW)  
Hourly\_Emps(ssn, name, lot, rating, hrly\_wages, hrs\_worked)
- What if we *know* rating determines hrly\_wages?

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 the existence of *integrity constraints*.

# 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?

# Refining an ER Diagram

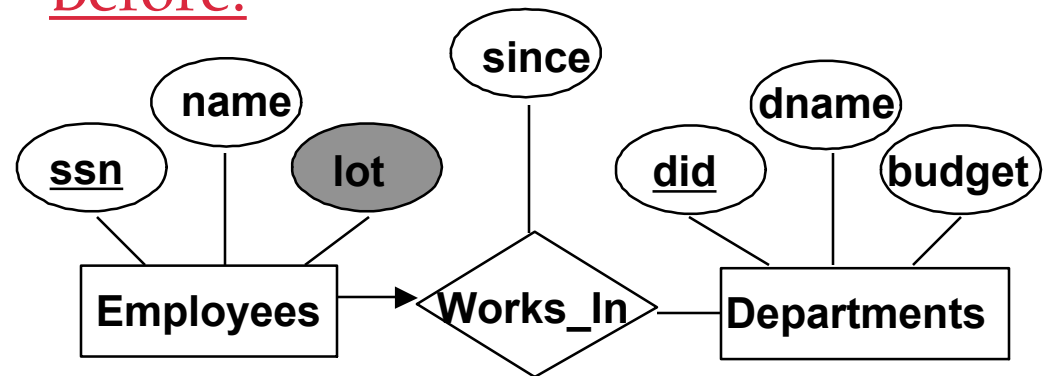
- 1st diagram translated:

Workers(S,N,L,D,S)

Departments(D,M,B)

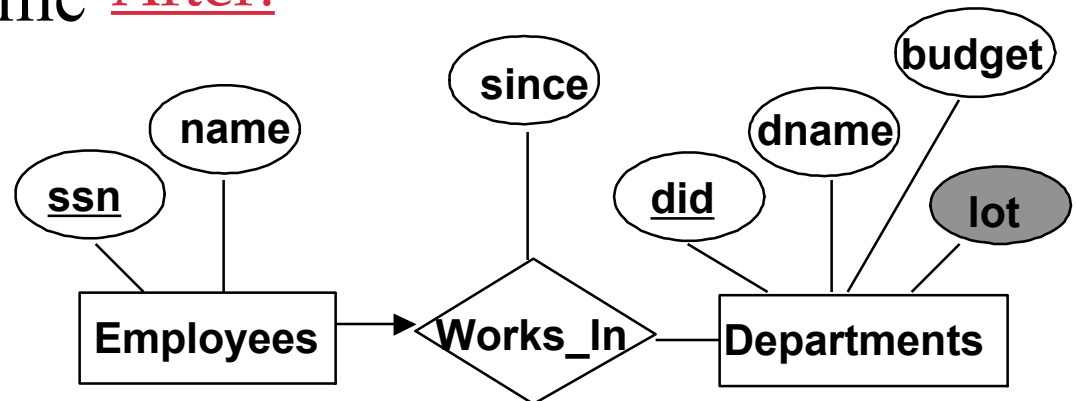
- Lots associated with workers.

Before:



- Suppose all workers in a dept are assigned the same lot:  $D \rightarrow L$

After:



- Can fine-tune this:

Workers2(S,N,D,S)

Departments(D,M,B,L)

# Functional Dependencies (FDs)

- A functional dependency (FD) has the form:  $X \rightarrow Y$ , where  $X$  and  $Y$  are two sets of attributes.
  - Examples: Rating  $\rightarrow$  hrly\_Wage, AB  $\rightarrow$  C
- The FD  $X \rightarrow Y$  *is satisfied by a relation instance  $r$  if:*
  - for each pair of tuples  $t_1$  and  $t_2$  in  $r$ :  
 $t_1.X = t_2.X$  implies  $t_1.Y = t_2.Y$
  - 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 (ssn, name, lot, rating, hrly\_wages, hrs\_worked)
- Notation: 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:  $S \rightarrow \text{SNLRWH}$
  - *rating* determines *hrly\_wages*:  $R \rightarrow W$

# One more example

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

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

49.

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

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

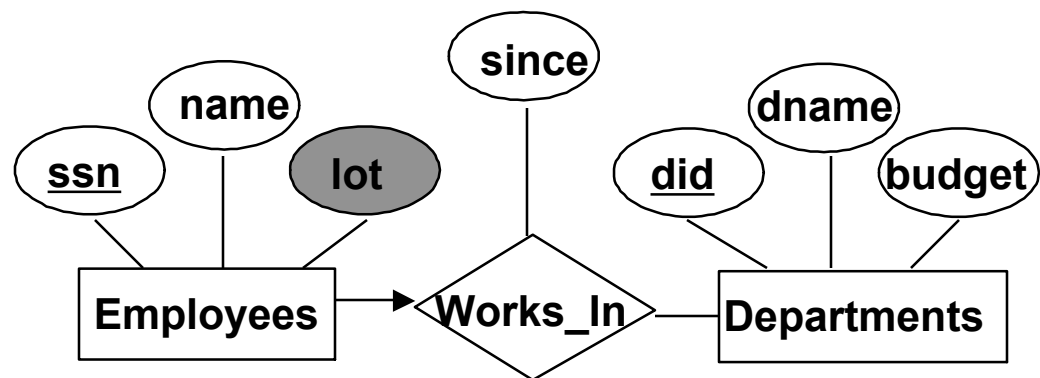
# Some other FDs

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  $F$ .

# 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  $X \rightarrow Z$

# 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.

# 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  $X \rightarrow YZ$
- $X \rightarrow YX$  (augment)
- $YX \rightarrow YZ$  (augment)
- Thus  $X \rightarrow YZ$  (transitive)

# Derive Decomposition

- Show that
  - If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
- $YZ \rightarrow Y$ ;  $YZ \rightarrow Z$  (reflexive)
- Thus  $X \rightarrow Y$ ;  $X \rightarrow Z$  (transitive)

## Another Useful Rule: Accumulation Rule

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

Proof:

- From  $Z \rightarrow W$ , augment with  $YZ$  to get  $YZ \rightarrow YZW$
- Thus  $X \rightarrow YZW$  (transitive)

# 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  $F^+$ 
  - $A \rightarrow H$ 
    - by transitivity from  $A \rightarrow B$  and  $B \rightarrow H$
  - $AG \rightarrow I$ 
    - by augmenting  $A \rightarrow C$  with  $G$ , to get  $AG \rightarrow CG$   
and then transitivity with  $CG \rightarrow I$
  - $CG \rightarrow HI$ 
    - by augmenting  $CG \rightarrow I$  to infer  $CG \rightarrow CGI$ ,  
and augmenting of  $CG \rightarrow H$  to infer  $CGI \rightarrow HI$ ,  
and then transitivity

# 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  $F$ :
    - $F = \{A \rightarrow B, B \rightarrow C, C D \rightarrow E; CD \rightarrow; CD \rightarrow D\}$
- 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.)

- Example:  $\text{Contracts}(cid, sid, jid, did, pid, qty, value)$ , and:
  - C is the key:  $C \rightarrow CSJDPQV$
  - Project (jid) purchases each part using single contract:  
 $JP \rightarrow C$
  - Dept purchases at most one part from a supplier:  $SD \rightarrow P$
- $JP \rightarrow C, C \rightarrow CSJDPQV$  imply  $JP \rightarrow CSJDPQV$
- $SD \rightarrow P$  implies  $SDJ \rightarrow JP$
- $SDJ \rightarrow JP, JP \rightarrow CSJDPQV$  imply  $SDJ \rightarrow CSJDPQV$

# Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # 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  $A$  such that  $X \rightarrow A$  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:  $\text{Result} = X^+$  (under  $F$ )
- Method:
  - Step 1:  $\text{Result} := X$ ;
  - Step 2: Take  $Y \rightarrow Z$  in  $F$ , and  $Y$  is in  $\text{Result}$ , do:  
 $\text{Result} := \text{Result} \cup Z$
  - Repeat step 2 until  $\text{Result}$  cannot be changed and then output  $\text{Result}$ .

# 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  
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 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.  $result = ABCG(A \rightarrow C \text{ and } A \rightarrow B)$
  3.  $result = ABCGH \quad (CG \rightarrow H \text{ and } CG \subseteq AGBC)$
  4.  $result = ABCGHI \quad (CG \rightarrow I \text{ and } CG \subseteq AGBCH)$
- Is  $AG$  a candidate key?
  - Is  $AG$  a super key?
    - Does  $AG \rightarrow R? == \text{Is } (AG)^+ \supseteq R$
  - Is any subset of  $AG$  a superkey?
    - Does  $A \rightarrow R? == \text{Is } (A)^+ \supseteq R$
    - Does  $G \rightarrow R? == \text{Is } (G)^+ \supseteq R$

# Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if  $\alpha$  is a superkey, we compute  $\alpha^+$ , and check if  $\alpha^+$  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  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - Is a simple and cheap test, and very useful
- Computing closure of  $F$ 
  - For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , and for each  $S \subseteq \gamma^+$ , we output a functional dependency  $\gamma \rightarrow S$ .

# 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)<sup>+</sup>

# 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