



# *Schema Refinement and Normal Forms*

## Chapter 19

### Quiz #2 Next Thursday





# *The Evils of Redundancy*

- ❖ *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- ❖ Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- ❖ 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?



# Functional Dependencies (FDs)

- ❖ A functional dependency  $X \rightarrow Y$  holds over relation R if, for every allowable instance  $r$  of R:
  - for all  $t_1 \in r, t_2 \in r, \pi_X(t_1) = \pi_X(t_2)$  implies  $\pi_Y(t_1) = \pi_Y(t_2)$
  - i.e., given two tuples in  $r$ , if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- ❖ An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance  $r_1$  of R, we can check if it violates some FD  $f$ , but we cannot tell if  $f$  holds over R!
- ❖ K is a candidate key for R means that  $K \rightarrow R$ 
  - However,  $K \rightarrow R$  does not require K to be *minimal*!



## *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 as a single letter: **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*:  $R \rightarrow W$



## Example (Contd.)

- ❖ Problems due to  $R \rightarrow W$  :
  - 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!

Hourly\_Emps

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

Hourly\_Emps2

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

Wages

R	W
8	10
5	7

Will 2 smaller tables be better?



# Reasoning About FDs

- ❖ Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did, did \rightarrow lot$  implies  $ssn \rightarrow lot$
- ❖ An FD  $f$  is implied by a set of FDs  $F$  if  $f$  holds whenever all FDs in  $F$  hold.
  - $F^+ = \text{closure of } F$  is the set of all FDs that are implied by  $F$ .
- ❖ Armstrong's Axioms ( $X, Y, Z$  are sets of attributes):
  - **Reflexivity:** If  $X \subseteq Y$ , then  $Y \rightarrow X$
  - **Augmentation:** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$
  - **Transitivity:** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- ❖ These are *sound* and *complete* inference rules for FDs!
  - *sound*: they will generate only FDs in  $F^+$
  - *complete*: repeated applications will generate all FDs in  $F^+$



## Reasoning About FDs (Contd.)

- ❖ 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  $X \rightarrow Z$
- ❖ Example: **Contracts**(cid,sid,jid,did,pid,qty,value), and:
  - C is the key:  $C \rightarrow CSJDPQV$
  - Projects purchase each part using single contract:  $JP \rightarrow C$
  - Dept purchase 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  $SDJ \rightarrow CSJDPQV$

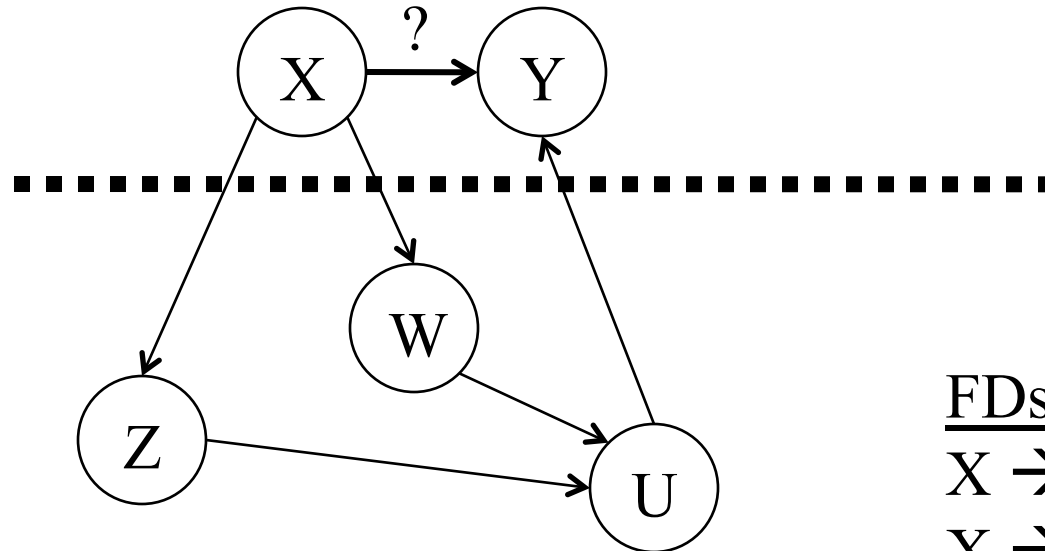


## *Reasoning About FDs (Contd.)*

- ❖ 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  $X^+$



# Example Check



FDs:

$X \rightarrow W$

$X \rightarrow Z$

$WZ \rightarrow U$

$U \rightarrow Y$

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



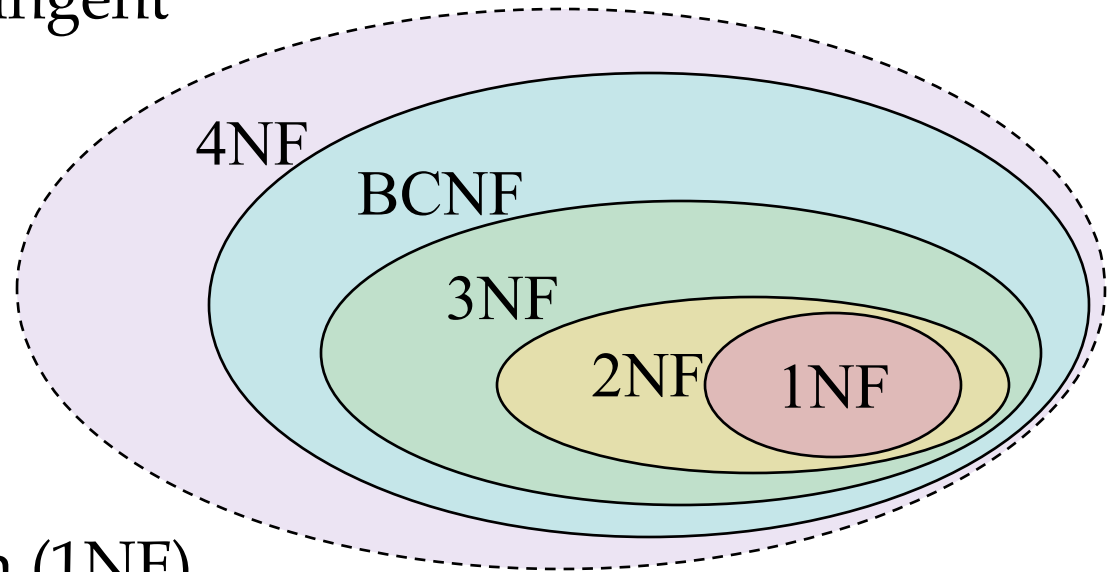
# Normal Forms

- ❖ To eliminate redundancy and potential update anomalies, one can identify generic templates called “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 Form Hierarchy

- ❖ An increasingly stringent hierarchy of “Normal Forms”
- ❖ Each outer form trivially satisfies the requirements of inner forms
- ❖ The 1<sup>st</sup> normal form (1NF) is part of the definition of the relational model. Relations must be sets (unique) and all attributes atomic (not multiple fields or variable length records).
- ❖ The 2<sup>nd</sup> normal form (2NF) requires schemas not have any FD,  $X \rightarrow Y$ , where  $X$  as a strict subset of the schema's key.





# *Boyce-Codd Normal Form (BCNF)*

- ❖ Relation  $R$  with FDs  $F$  is in **BCNF** if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - $X$  contains a key for  $R$ .
- ❖ In other words,  $R$  is in BCNF if the only non-trivial FDs that hold over  $R$  are key constraints.
- ❖ BCNF considers all domain keys, not just the primary one
- ❖ BCNF schemas do not contain redundant information that arise from FDs



# *BCNF Examples*

- ❖ In BCNF

Person(First, Last, Address, Phone)

Functional Dependencies:  $FL \rightarrow A$ ,  $FL \rightarrow P$

- ❖ Not in BCNF

Person(First, Last, Address, Phone, Hobby)

Functional Dependencies:  $FL \rightarrow A$ ,  $FL \rightarrow P$



## Third Normal Form (3NF)

- ❖ Reln  $R$  with FDs  $F$  is in **3NF** if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - $X$  contains a key for  $R$ , or
  - $A$  is part of some key for  $R$ .
- ❖ *Minimality* of a key is crucial in third condition above!
- ❖ If  $R$  is in BCNF, it is trivially in 3NF.
- ❖ If  $R$  is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no “good” decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of  $R$  into a collection of 3NF relations always possible.*



## *3NF Examples*

- ❖ Phonebook where friends have multiple addresses
- ❖ In 3NF, not in BCNF

Person(First, Last, Addr, Phone)

Functional Dependencies:

$FLA \rightarrow P, P \rightarrow A$

- ❖ Not in 3NF or BCNF

Person(First, Last, Addr, Phone, Mobile)

Functional Dependencies:

$FLA \rightarrow P, P \rightarrow A, FL \rightarrow M$



# *What Does 3NF Achieve?*

- ❖ If 3NF is violated by  $X \rightarrow A$ , one of the following holds:
  - $X$  is a proper subset of some key  $K$  (partial dependency)
    - We store  $(X, A)$  pairs redundantly.
  - $X$  is not a proper subset of any key (transitive dependency).
    - There is a chain of FDs  $K \rightarrow X \rightarrow A$ , which means that we cannot associate an  $X$  value with a  $K$  value unless we also associate an  $A$  value with an  $X$  value.
- ❖ **But**, even if relation is in 3NF, problems can arise.



# *Lingering 3NF Redundancies*

- ❖ Revisiting an old Schema

Reserves(Sailor, Boat, Date, CreditCardNo)

FDs:  $C \rightarrow S$

- ❖ In 3NF, but database likely stores many redundant copies of the (C, S) tuple
- ❖ Thus, 3NF is indeed a compromise relative to BCNF.



# *Decomposition of a Relation Scheme*

- ❖ Suppose that relation  $R$  contains attributes  $A_1 \dots A_n$ . 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 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  $R$ .
- ❖ E.g., Can decompose **SNLRWH** into **SNLRH** and **RW**.



## *Example Decomposition*

- ❖ Decompositions should be used only when needed.
  - SNLRWH has FDs  $S \rightarrow \text{SNLRWH}$  and  $R \rightarrow W$
  - Second FD causes violation of 3NF;  $W$  values repeatedly associated with  $R$  values. Easiest way to fix this is to create a relation  $RW$  to store these associations, and to remove  $W$  from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and  $RW$
- ❖ The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and  $RW$ , are there any potential problems that we should be aware of?



# *Problems with Decompositions*

- ❖ There are three potential problems to consider:

Problem 1) Some queries become more expensive.

- e.g., How much did Joe earn? (salary =  $W \times H$ )

Problem 2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding original relation!

- Fortunately, not in the SNLRWH example.

Problem 3) 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.



# *Lossless Join Decompositions*

- ❖ Decomposition of  $R$  into  $X$  and  $Y$  is lossless-join w.r.t. a set of FDs  $F$  if, for every instance  $r$  that satisfies  $F$ :
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- ❖ It is always true that  $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$ 
  - In general, the other direction does not hold!  
If equal, the decomposition is lossless-join.
- ❖ Definition extended to decomposition into 3 or more relations in a straightforward way.
- ❖ *It is essential that all decompositions used to eliminate redundancy be lossless! (Avoids Problem 2)*



## More on Lossless Join

- ❖ The decomposition of R into X and Y is **lossless-join** wrt F if and only if the closure of F contains:

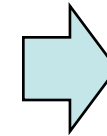
- $X \cap Y \rightarrow X$ , or
- $X \cap Y \rightarrow Y$

(in other words the attributes common to X and Y must contain a key for either X or Y)

- ❖ In particular, the decomposition of R into UV and R - V is lossless-join if  $U \rightarrow V$  holds over R.

$ABC \Rightarrow AB, BC$

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

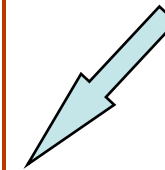


A	B
1	2
4	5
7	2



B	C
2	3
5	6
2	8

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



Not lossless



## More on Lossless Join

- ❖ The decomposition of R into X and Y is **lossless-join** wrt F if and only if the closure of F contains:

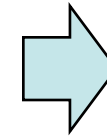
- $X \cap Y \rightarrow X$ , or
- $X \cap Y \rightarrow Y$

(in other words the attributes common to X and Y must contain a key for either X or Y)

- ❖ In particular, the decomposition of R into UV and R - V is lossless-join if  $U \rightarrow V$  holds over R.

$ABC \Rightarrow AB, AC$

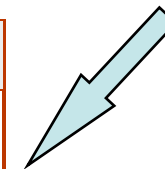
A	B	C
1	2	3
4	5	6
7	2	8



A	B
1	2
4	5
7	2



A	C
1	3
4	6
7	8



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

Lossless



# Dependency Preserving Decomposition

Contracts(Cid, Sid, Jid, Did, Pid, Qty, Value)

- ❖ Consider CSJDPQV, C is key,  $JP \rightarrow C$  and  $SD \rightarrow P$ .
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking  $JP \rightarrow C$  requires a join!
- ❖ **Dependency preserving decomposition** (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. (*Avoids Problem 3*)
- ❖ *Projection of set of FDs F*: If R is decomposed into X, ... projection of F onto X (denoted  $F_X$ ) is the set of FDs  $U \rightarrow V$  in  $F^+$  (closure of F) such that U, V are in X.



# Dependency Preserving Decomposition

- ❖ Decomposition of R into X Y is dependency preserving if  $(F_X \text{ union } F_Y)^+ = F^+$ 
  - i.e., if we consider only dependencies in the closure  $F^+$  that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in  $F^+$ .
- ❖ MUST consider  $F^+$ , (**not just F**), in this definition:
  - ABC,  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow A$ , decomposed into AB and BC.
  - Is this dependency preserving? Is  $C \rightarrow A$  preserved?????
- ❖ Dependency preserving *does not imply* lossless join:
  - ABC,  $A \rightarrow B$ , decomposed into AB and BC.
- ❖ And vice-versa!



# *Decomposition into BCNF*

- ❖ Consider relation  $R$  with FDs  $F$ .  
If  $X \rightarrow Y$  violates BCNF,  
decompose  $R$  into  $R - Y$  and  $XY$ .
  - Repeated applications of this rule gives relations in BCNF; lossless join decomposition, and is guaranteed to terminate.
- ❖ Example:  $\underline{CSJDPQV}$ ,  $SD \rightarrow P$ ,  $J \rightarrow S$  (new),  
(ignoring  $JP \rightarrow C$  for now)
  - To deal with  $SD \rightarrow P$ , decompose into  $\underline{SDP}$ ,  $\underline{CSJDQV}$ .
  - To deal with  $J \rightarrow S$ , decompose  $\underline{CSJDQV}$  into  $\underline{JS}$  and  $\underline{CJDQV}$
- ❖ The order in which we “deal with” violations could lead to a different set of relations!



# *BCNF and Dependency Preservation*

- ❖ In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g.,  $CSZ, CS \rightarrow Z, Z \rightarrow C$
  - Can't decompose while preserving 1st FD; not in BCNF.
- ❖ Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs:  $JP \rightarrow C$ ,  $SD \rightarrow P$  and  $J \rightarrow S$ ).
  - However, it is a lossless join decomposition.
  - In this case, adding  $C\underline{JP}$  to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Adds Redundancy!*)



## *Decomposition into 3NF*

- ❖ Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, it can stop earlier).
- ❖ To ensure dependency preservation, one idea:
  - If  $X \rightarrow Y$  is not preserved, add relation  $XY$ .
  - Problem is that  $XY$  may violate 3NF! e.g., consider the addition of CJP to “preserve”  $JP \rightarrow C$ . What if we also have  $J \rightarrow C$ ?
- ❖ **Refinement:** Instead of the given set of FDs  $F$ , use a *minimal cover for  $F$* .



# *Minimal Cover for a Set of FDs*

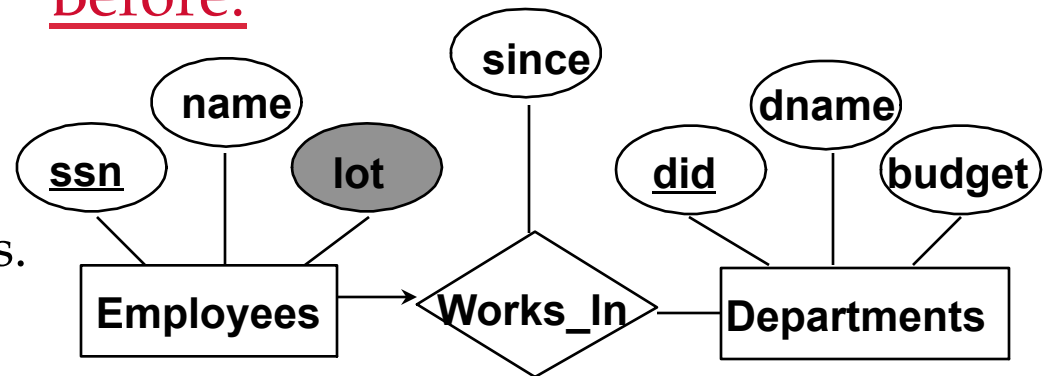
- ❖ Properties of a Minimal cover,  $G$ , for a set of FDs  $F$ :
  - Closure of  $F$  = closure of  $G$ .
  - Right hand side of each FD in  $G$  is a single attribute.
  - If we modify  $G$  by deleting a FD or by deleting attributes from an FD in  $G$ , the closure changes.
- ❖ Intuitively, every FD in  $G$  is needed, and is “*as small as possible*” in order to get the same closure as  $F$ .
- ❖ e.g.,  $A \rightarrow B$ ,  $ABCD \rightarrow E$ ,  $EF \rightarrow GH$ ,  $ACDF \rightarrow EG$  has the following minimal cover:
  - $A \rightarrow B$ ,  $ACD \rightarrow E$ ,  $EF \rightarrow G$  and  $EF \rightarrow H$
- ❖ M.C.  $\rightarrow$  Lossless-Join, Dep. Pres. Decomp!!! (in book)



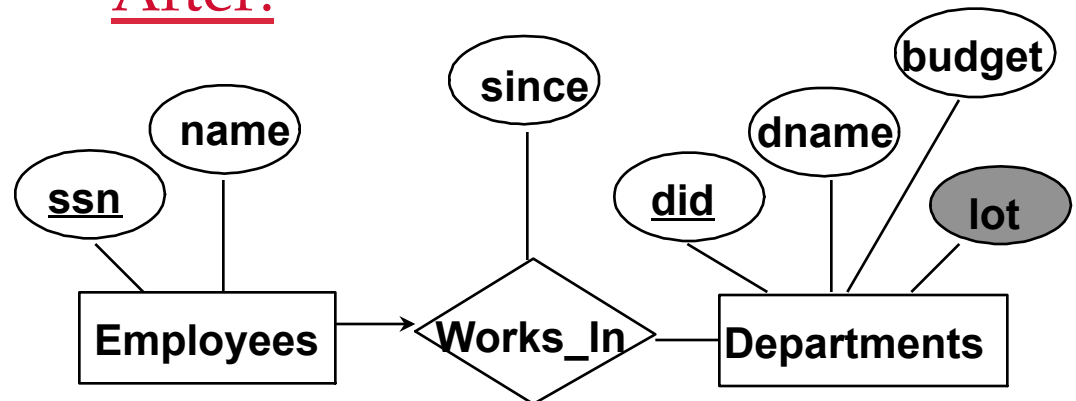
# Refining an ER Diagram

- ❖ 1st diagram translated:  
*Workers(S,N,L,D,S)*  
*Departments(D,M,B)*
  - Lots associated with workers.
- ❖ Suppose all workers in a dept are assigned the same lot:  $D \rightarrow L$
- ❖ Redundancy; fixed by:  
*Workers2(S,N,D,S)*  
*Dept\_Lots(D,L)*
- ❖ Can fine-tune this:  
*Workers2(S,N,D,S)*  
*Departments(D,M,B,L)*

Before:



After:





# *Summary of Schema Refinement*

- ❖ If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- ❖ If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.