

# Database Normalization

Lecture 4

EECS 339

# 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:
  - 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  $r1$  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: 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.)

Wages

R	W
8	10
5	7

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

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

- 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!

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.
  - $F^+ =$  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!

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



# Study Break: Functional Dependencies

You are designing a database for a hospital with the following requirements:

- Want to represent patients at hospitals with doctors
- Patients have names, birthdates
- Doctors have names, specialties
- Hospitals have names, addresses
- One doctor can treat multiple patients, each patient has one doctor
- Each patient in one hospital, hospitals have many patients
- Doctors work for one hospital, hospitals have many doctors

- 1) Create an ER diagram for this database.
- 2) Construct a set of relations for the hospital.
- 3) What are the functional dependencies of this database?

# Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- 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 will all have the same B value!

# First Normal Form

- If your data is stored in tables, it is in 1NF
- No recursion, tuples contain no sets or lists

# Second Normal Form

- Everything is still stored in tables
- Every attribute is fully functionally dependent on the primary key
  - Fully = need entire primary key to infer dependent values, even for composites

# 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, obviously 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.*

# What Does 3NF Achieve?

- If 3NF violated by  $X \rightarrow A$ , one of the following holds:
  - $X$  is a subset of some key  $K$ 
    - We store  $(X, A)$  pairs redundantly.
  - $X$  is not a proper subset of any key.
    - 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 reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC,  $S \rightarrow C$ ,  $C \rightarrow S$  is in 3NF, but for each reservation of sailor  $S$ , same  $(S, C)$  pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

# Boyce-Codd Normal Form (BCNF)

- Reln 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.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

# 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:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? (salary =  $W * H$ )
  - 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.
  - 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:
  - $x \text{ JOIN } y = r$
- It is always true that  $r$  is a subset of  $X \text{ JOIN } Y$ 
  - In general, the other direction does not hold! If it does, 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 deal with redundancy be lossless! (Avoids Problem (2).)*

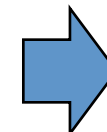
## More on Lossless Join

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

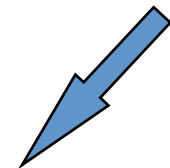
A	B	C
1	2	3
4	5	6
7	2	8



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



# Dependency Preserving Decomposition

- 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 Decompositions (Contd.)

- Decomposition of R into X and 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  $F^+$ .
- Important to consider  $F^+$ , not  $F$ , 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! (Example?)

# Decomposition into BCNF

- Consider relation R with FDs F. If  $X \rightarrow Y$  violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C,  $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $J \rightarrow S$
  - To deal with  $SD \rightarrow P$ , decompose into SDP, CSJDQV.
  - To deal with  $J \rightarrow S$ , decompose CSJDQV into JS and CJDQV
- 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!

# 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 JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*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, 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

- 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 an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and “*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. Lossless-Join, Dep. Pres. Decomp!!! (in book)

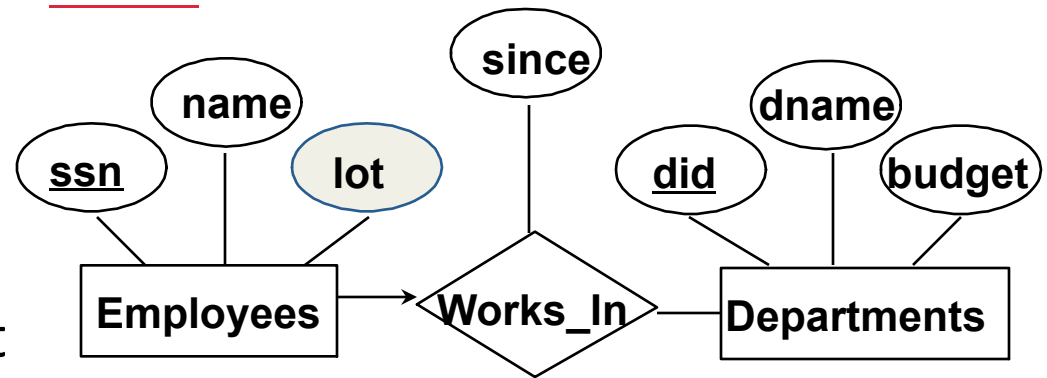
# Recap: BCNF'ing a schema

- Start with one “universal relation”
- For each FD if  $X \rightarrow Y$  violates BCNF (i.e.,  $X$  is not a superkey over  $Y$ )
  - split up  $R$  into  $R_1$  and  $R_2$ 
    - $R_1 = X \text{ Union } Y$
    - $R_2 = R - (X \cup Y)$
- Continue iterating until in BCNF

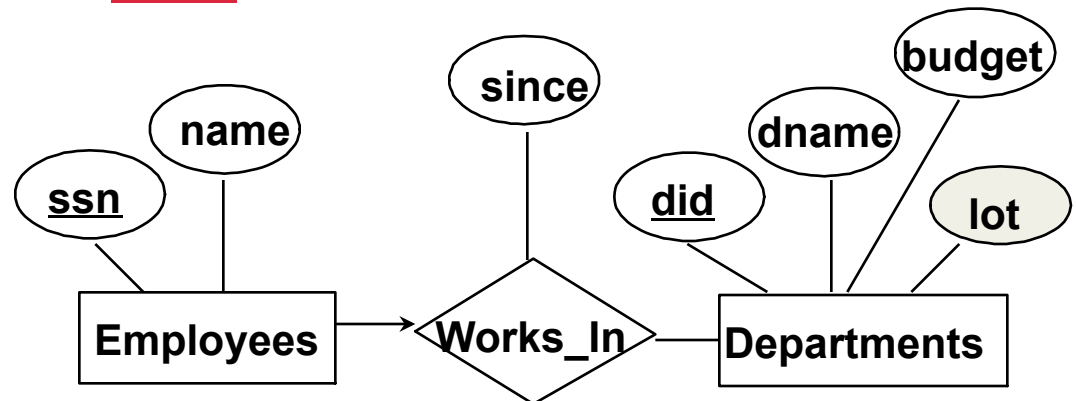
# Refining an ER Diagram

- 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:



# Study Break

- Convert your hospital schema into Boyce Codd Normal Form using its functional dependencies.

# 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.