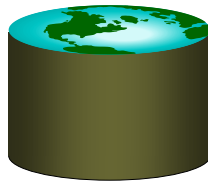


## Schema Refinement and Normalization

Nobody realizes that some people  
expend tremendous energy  
merely to be normal.

Albert Camus



## Normal Forms

- Review FDs, Armstrong's Axioms and Attr. Closures!
- Q1: is any refinement needed??!
- If a relation is in a **normal form** (BCNF, 3NF etc.):
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given  $A \rightarrow B$ : If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!
- 1<sup>st</sup> Normal Form – all attributes are atomic
  - i.e. the relational model
- 1<sup>st</sup>  $\supset$  2<sup>nd</sup> (of historical interest)  $\supset$  3<sup>rd</sup>  $\supset$  Boyce-Codd  $\supset$  ...



## Boyce-Codd Normal Form (BCNF)

- ReIn R with FDs F is in **BCNF** if, for all  $X \rightarrow A$  in  $F^+$ 
  - $A \in X$  (called a *trivial* FD), or
  - X is a superkey for R.
- In other words: "R is in BCNF if the only non-trivial FDs over R are **key constraints**."
- If R in BCNF, then every field of every tuple records information that **cannot be inferred** using FDs alone.
  - Say we know FD  $X \rightarrow A$  holds this example relation:

- Can you guess the value of the missing attribute?

X	Y	A
5	y1	7
5	y2	?

- Yes, so relation is not in BCNF



## Decomposition of a Relation Schema

- If a relation is not in a desired normal form, it can be **decomposed** into multiple relations that each are in that normal form.
- 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
  - Every attribute of R appears as an attribute of at least one of the new relations.



## Example (same as before)

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\_Emps

- SNLRWH has FDs  $S \rightarrow SNLRWH$  and  $R \rightarrow W$
- Q: Is this relation in BCNF?

No, The second FD causes a violation;  
W values repeatedly associated with R values.



## Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

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

R	W
8	10
5	7

Wages

Hourly\_Emps2

- Q: Are both of these relations are now in BCNF?
- **Decompositions should be used only when needed.**
  - Q: potential problems of decomposition?



## Problems with Decompositions

- There are three potential problems to consider:
  - 1) May be **impossible** to reconstruct the original relation! (**Lossy Decomposition**)
    - Fortunately, not in the SNLRWH example.
  - 2) Dependency checking may require joins (not **Dependency Preserving**)
    - Fortunately, not in the SNLRWH example.
  - 3) Some queries become more expensive.
    - e.g., How much does Guldu earn?

**Tradeoff:** Must consider these issues vs. redundancy.  
(Well, not usually #1)



## Lossless Decomposition (example)

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

R	W
8	10
5	7

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



## Lossy Decomposition (example)

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

  
 $A \rightarrow B; C \rightarrow B$ 
  

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



## Lossless Join Decompositions

- Decomposition of R into X and Y is **lossless** 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 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 #1**)



## More on Lossless Decomposition

- The decomposition of R into X and Y is **lossless with respect to F** if and only if the **closure** of F contains:
 
$$X \cap Y \rightarrow X, \text{ or } X \cap Y \rightarrow Y$$

i.e. the common attributes form a **superkey** for one side or the other

in example: decomposing ABC into AB and BC is lossy, because intersection (i.e., "B") is not a key of either resulting relation.

- Useful result:** If  $W \rightarrow Z$  holds over R and  $W \cap Z$  is empty, then decomposition of R into R-Z and WZ is loss-less.



## Lossless Decomposition (example)

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

  
 $A \rightarrow B; C \rightarrow B$ 
  

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

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

But, now we can't check  $A \rightarrow B$  without doing a join!



## Dependency Preserving Decomposition

- **Dependency preserving decomposition (Intuitive):**
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R must also hold. (*Avoids Problem #2 on our list.*)
    - Why do we care??
- **Projection of set of FDs F:** If R is decomposed into X and Y the projection of F on X (denoted  $F_X$ ) is the set of FDs  $U \rightarrow V$  in  $F^+$  (closure of F, not just F) such that all of the attributes U, V are in X. (same holds for Y of course)



## Dependency Preserving Decompositions (Contd.)

- **Decomposition of R into X and Y is dependency preserving if  $(F_X \cup F_Y)^+ = F^+$** 
  - 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  $F^+$  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????
    - note:  $F^+$  contains  $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$ , so...
- $F_{AB}$  contains  $A \rightarrow B$  and  $B \rightarrow A$ ;  $F_{BC}$  contains  $B \rightarrow C$  and  $C \rightarrow B$
- So,  $(F_{AB} \cup F_{BC})^+$  contains  $C \rightarrow A$



## Decomposition into BCNF

- **Consider relation R with FDs F. If  $X \rightarrow Y$  violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).**
  - 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,  $JP \rightarrow C$ ,  $SD \rightarrow P$ ,  $J \rightarrow S$ 
    - {contractid, supplierid, projectid, deptid, partid, qty, value}
    - To deal with  $SD \rightarrow P$ , decompose into SDP, CSJDQV.
    - To deal with  $J \rightarrow S$ , decompose CSJDQV into JS and CJDQV
    - So we end up with: SDP, JS, and CJDQV
- **Note: 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 CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs  $JP \rightarrow C$ ,  $SD \rightarrow P$  and  $J \rightarrow S$ ).**
  - {contractid, supplierid, projectid, deptid, partid, qty, value}
    - However, it is a lossless join decomposition.
    - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
      - but JPC tuples are stored only for checking the f.d. (Redundancy!)



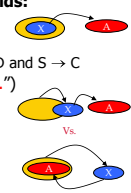
## Third Normal Form (3NF)

- **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 is a superkey of R, or
  - A is part of some *candidate* key (not superkey!) for R. (sometimes stated as “A is *prime*”)
- **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 (“*partial dependency*”)
    - We store (X, A) pairs redundantly.
    - e.g. Reserves SBDC (C is for credit card) with key SBD and  $S \rightarrow C$
  - X is not a proper subset of any key. (“*transitive dep.*”)
    - There is a chain of FDs  $K \rightarrow X \rightarrow A$
    - So we can't associate an X value with a K value unless we also associate an A value with an X value (different K's, same X implies same A!)
    - problem with initial SNLRWH example.
- **But: even if R is in 3NF, these problems could arise.**
  - e.g., Reserves SBDC (note: “C” is for credit card here),  $S \rightarrow C$ ,  $C \rightarrow S$  is in 3NF (why?)
  - Even so, for each reservation of sailor S, same (S, C) pair is stored.
- **Thus, 3NF is indeed a compromise relative to BCNF.**
  - You have to deal with the partial and transitive dependency issues in your application code!





## An Aside: Second Normal Form

- Like 3NF, but allows transitive dependencies:
  - Reln R with FDs  $F$  is in 2NF if, for all  $X \rightarrow A$  in  $F^+$ 
    - $A \in X$  (called a *trivial* FD), or
    - $X$  is a superkey of R, or
    - $X$  is not part of *any candidate* key for R. (i.e. " $X$  is *not prime*")
- There's no reason to use this in practice
  - And we won't expect you to remember it



## 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) but does not ensure dependency preservation.
- 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. implies Lossless-Join, Dep. Pres. Decomp!!!
  - (in book, p. 627)



## Summary of Schema Refinement

- **BCNF:** each field contains information that cannot be inferred using only FDs.
  - ensuring BCNF is a good heuristic.
- **Not in BCNF?** Try decomposing into BCNF relations.
  - Must consider whether all FDs are preserved!
- **Lossless-join, dependency preserving decomposition into BCNF impossible? Consider 3NF.**
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.
- **Note:** even more restrictive Normal Forms exist (we don't cover them in this course, but some are in the book.)