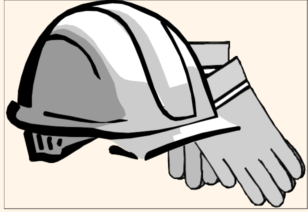
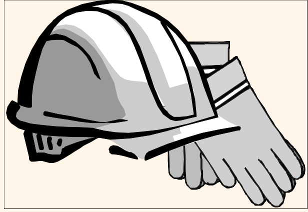


Schema Refinement and Normal Forms

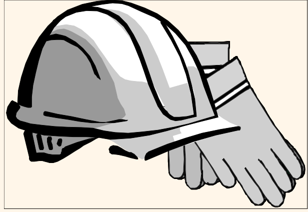
Chapter 19

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 <u>functional dependency</u> $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:

  $t1 \in r$, $t2 \in r$, $\prod_X(t1) = \prod_X(t2)$ implies $\prod_Y(t1) = \prod_Y(t2)$

Given two tuples in *r*

if the *X* values agree

then the *Y* values must also agree

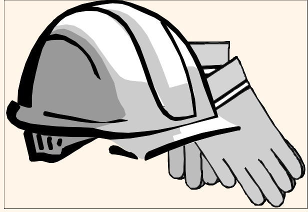
Y and *Y* are sets of attributes

- Example: SSN $\rightarrow$ StudentNum

Functional Dependencies (FDs)

- An FD is a statement about *all* allowable instances of a relation.
  - Must be identified based on semantics of application.
  - Given some allowable instance 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 $K \longrightarrow R$
  - However, $K \longrightarrow R$ does not require K to be *minimal*!

Example: Constraints on Entity Set



- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*<u>ssn</u>*, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)

    S N L R W H
- *<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: $S \longrightarrow SNLRWH$
  - *rating* determines *hrly_wages*: $R \longrightarrow W$

Example (Contd.)

Problems due to R → 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!

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

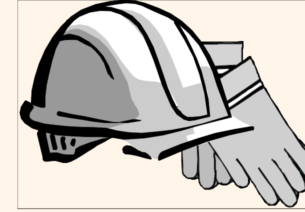
S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Wages

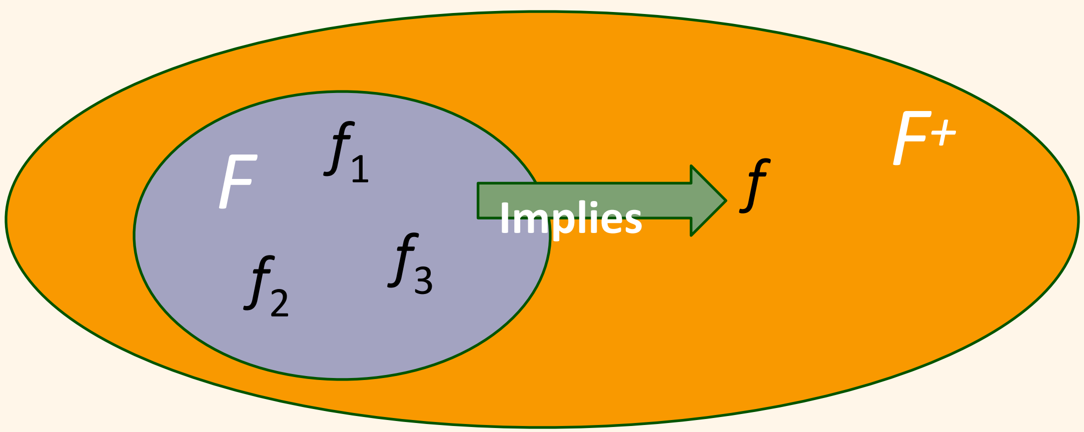
R	W
8	10
5	7

Using two smaller tables is better

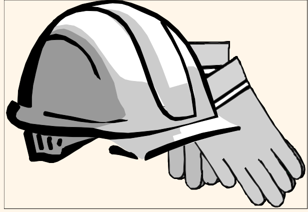
Reasoning About FDs



- 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.

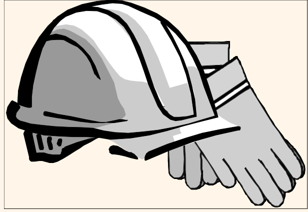


Armstrong's Axiom



- 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!
- 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 $X \rightarrow Z$

Reasoning About FDs - Example



Project ID Department ID

Supplier ID Part ID

Contract ID Quantity

Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- C is the key: C → CSJDPQV (C is a candidate key)
- Project purchases each part using single contract: JP → C
- Dept purchases at most one part from a supplier: SD → P

❖ JP → C, C → CSJDPQV imply JP → CSJDPQV

❖ SD → P implies SDJ → JP

❖ SDJ → JP, JP → CSJDPQV imply SDJ → CSJDPQV

These are also candidate keys

Closure of a FD set

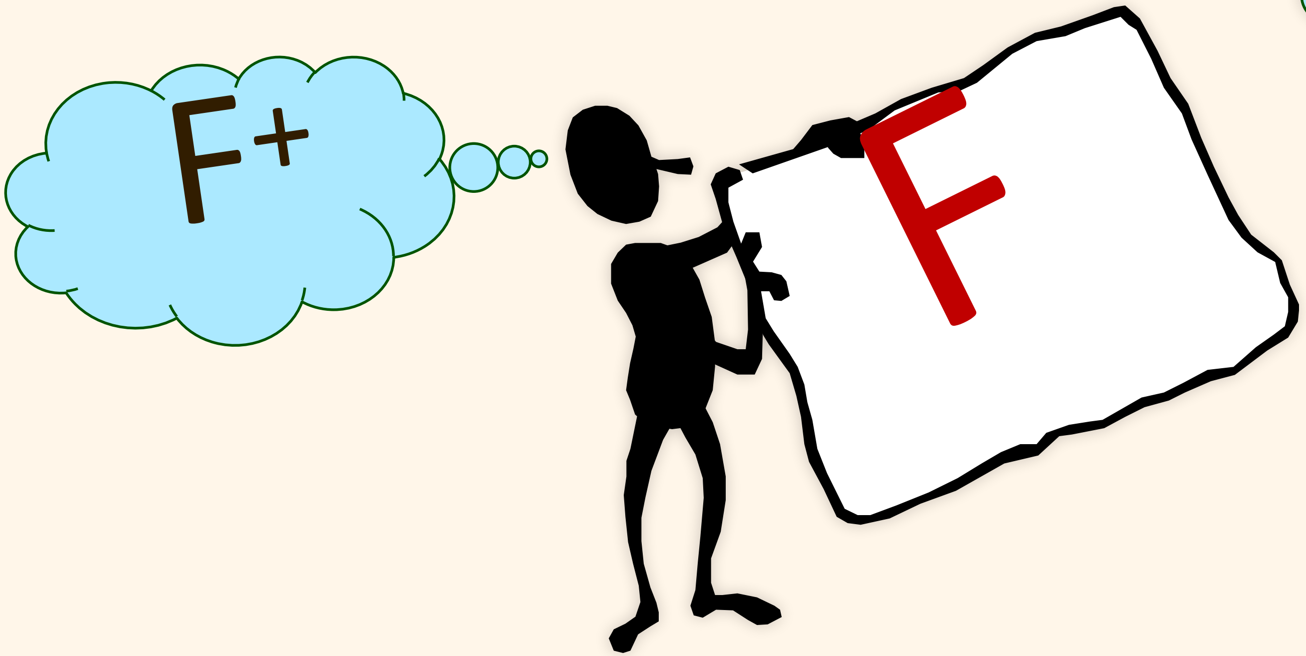
F

X→AB
…

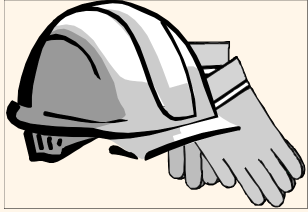
Explicit set

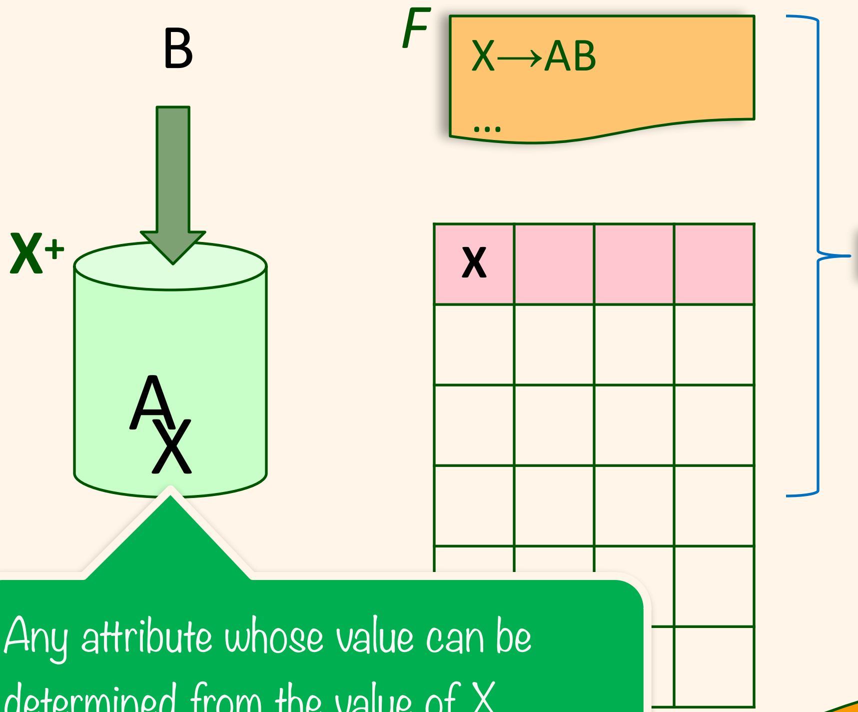
F^+ = *closure of F* is the set of all FDs that are implied by *F*

Implicit set



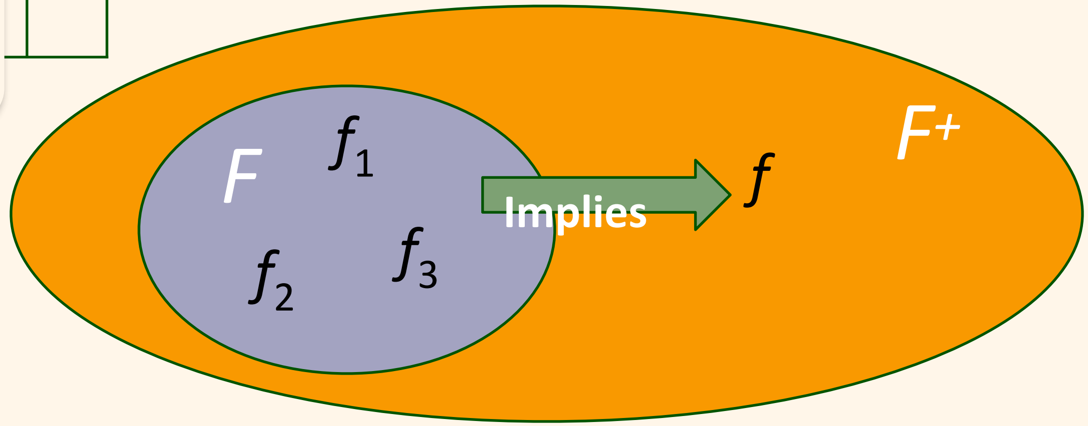
Attribute Closure (≠ closure of FD set)



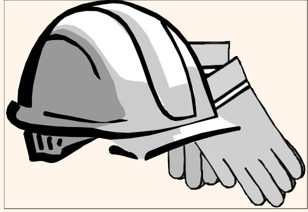


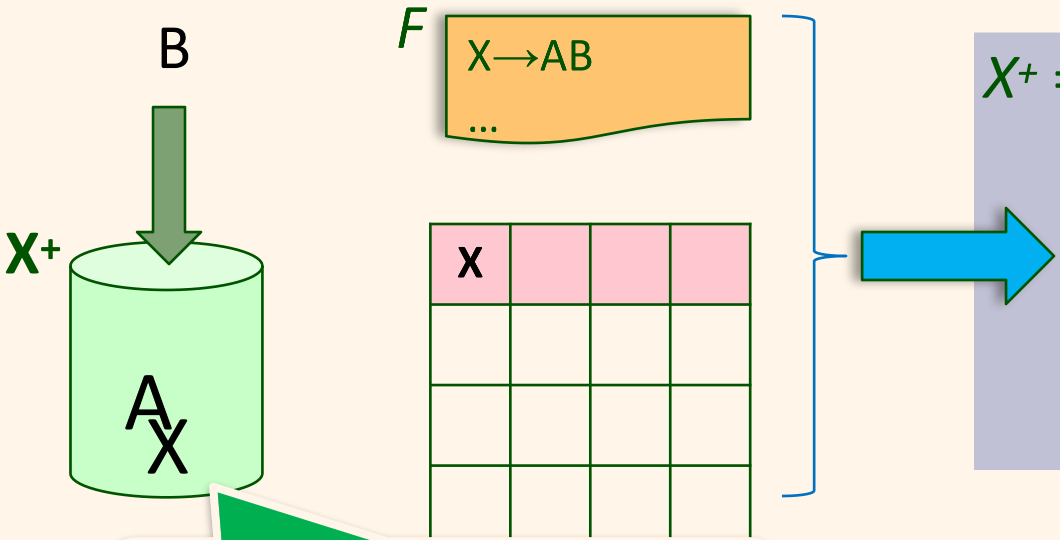
X^+ = *attribute closure* of an attribute set X wrt to a set of FD *F is the set of* all attributes A such that X ⟶ A is in F^+

Any attribute whose value can be determined from the value of X (according to F^+)



Attribute Closure (≠ closure of FD set)





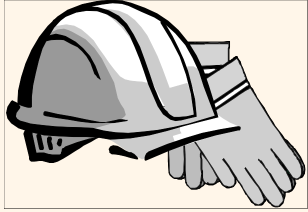
X^+ = ***attribute closure*** of an attribute set X wrt to a set of FD *F is the set of* all attributes A such that X ⟶ A is in F^+

Any attribute whose value can be determined from the value of X (according to F^+)

If left-hand side is in the closure, add the right hand side to the closure

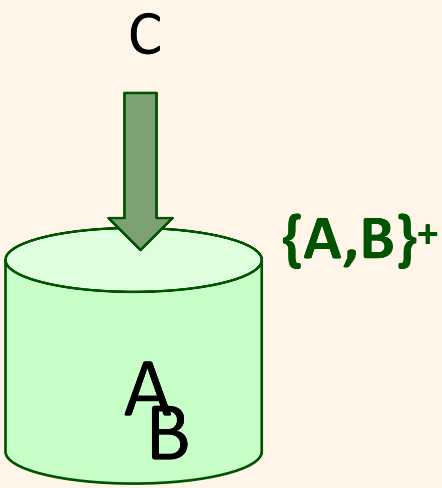
Closure = X;
Repeat until there is no change {
if there is an FD $U \longrightarrow V$ in F^+ such that $U \subseteq$ *closure*,
then set *closure = closure* $\cup$ *V* }

Attribute Closure - Example



Relation *ABCDEF* with FD's $\{AB \longrightarrow C,\ BC \longrightarrow AD,\ D \longrightarrow E,\ CF \longrightarrow B\}$

What is $\{A, B\}^+$?



Initially, $\{A, B\}^+ = \{A, B\}$

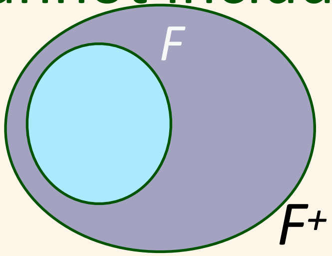
$AB \longrightarrow C \Rightarrow \{A, B\}^+ = \{A, B, C\}$

$BC \longrightarrow AD \Rightarrow \{A, B\}^+ = \{A, B, C, D\}$

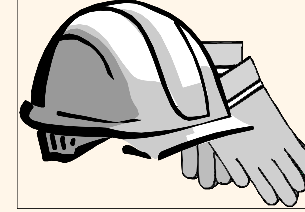
$D \longrightarrow E \Rightarrow \{A, B\}^+ = \{A, B, C, D, E\}$

$CF \longrightarrow B \Rightarrow$ *F* is on the left-hand side, we cannot include *F*

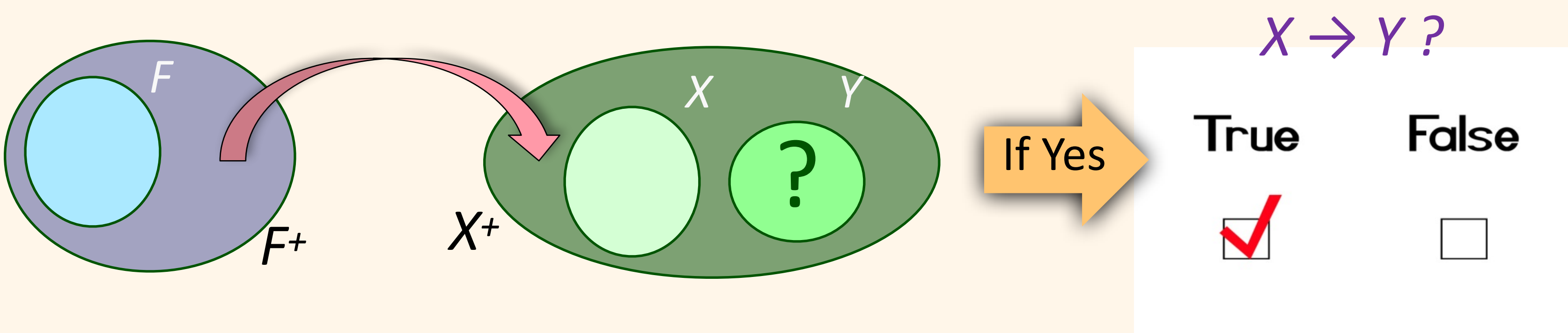
Make sure to consider the closure of the FD set



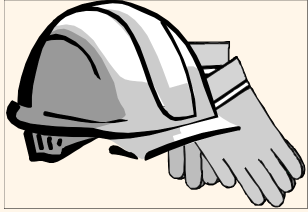
Reasoning About FDs (Contd.)



- 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:
  1. Compute X^+ wrt F
  2. Check if Y is in X^+ (i.e., Do we have $X \rightarrow Y$?)

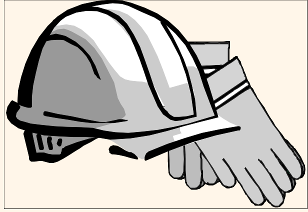


Normal Forms



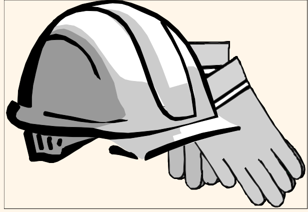
- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- Role of FDs in detecting redundancy:
  - Consider a relation *R* with 3 attributes, *ABC*.
    - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value - redundancy !
    - No FDs hold: There is no redundancy here
    - Note: $A \rightarrow B$ potentially causes problems. However, if we know that no two tuples share the same value for A, then such problems cannot occur (a normal form)
- 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.

Boyce-Codd Normal Form (BCNF)



R	A relation
F	The set of FD hold over *R*
X	A subset of the attributes of *R*
A	An attribute of *R*

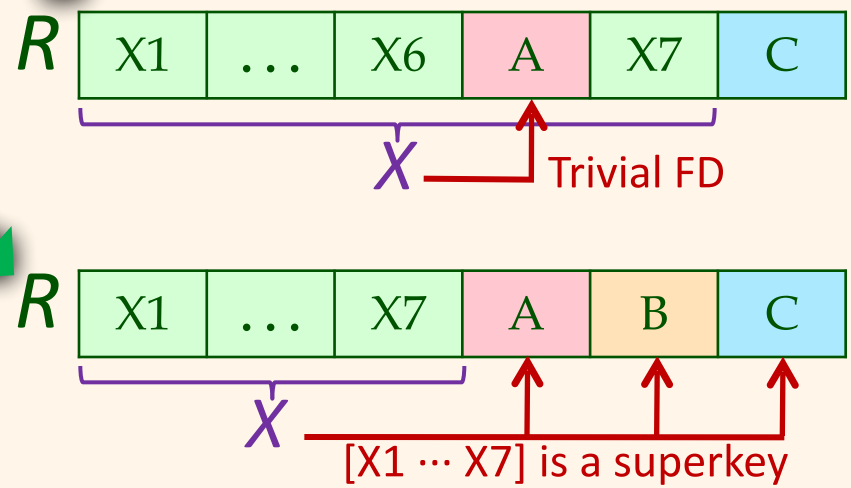
Boyce-Codd Normal Form (BCNF)



Relation *R* is in BCNF if, for all *X* → *A* in *F*,

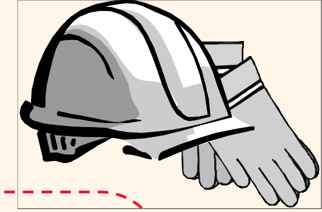
- *A* ∈ *X* (called a *trivial* FD), or
- *X* is a superkey (i.e., contains a key of *R*)

R	A relation
F	The set of FD hold over *R*
X	A subset of the attributes of *R*
A	An attribute of *R*



In other words, *R* is in BCNF if the only non-trivial FDs that hold over *R* are key constraints (i.e., *X* must be a superkey!)

BCNF is Desirable



Consider the relation:

X	Y	A
x	y1	y2
x	y2	?

$X \rightarrow A$

Should be y2

Not in BCNF

"$X \rightarrow A$" $\Rightarrow$ The 2nd tuple also has y2 in the third column

$\Rightarrow$ an example of redundancy

Such a situation cannot arise in a BCNF relation:

BCNF $\Rightarrow$ **X must be a key**

$\Rightarrow$ we must have $X \rightarrow Y$

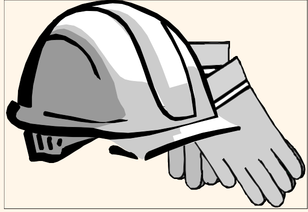
$\Rightarrow$ we must have "y1 = y2" (1)

$X \rightarrow A$ $\Rightarrow$ The two tuples have the same value for A (2)

(1) & (2) $\Rightarrow$ The two tuples are identical

$\Rightarrow$ This situation cannot happen in a relation

BCNF: Desirable Property



A relation is in BCNF

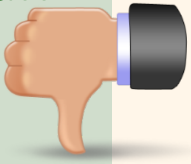
⇒ every entry records a piece of information that cannot be inferred (using only FDs) from the other entries in the relation instance

⇒ No redundant information !

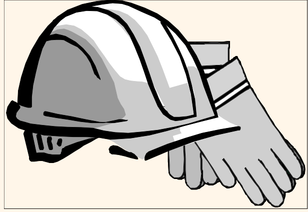
Key constraint is the only form of FDs allowed in BCNF

A relation *R*(*ABC*)

- ***B*→*C***: The value of *B* determines *C*, and the value of *C* can be inferred from another tuple with the same *B* value
  redundancy ! (not BCNF)



- ***A*→*BC***: Although the value of *A* determines the values of *B* and *C*, we cannot infer their values from other tuples because no two tuples in *R* have the same value for *A*
  ⇒ no redundancy ! (BCNF)



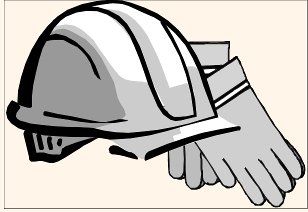
Third Normal Form (3NF)

- Let R be a relation with the set of FDs F, X be a subset of the attributes, and A be an attribute of R
- Relation R is in 3NF if, for all $X \longrightarrow A$ in F
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey (containing some key), or
  - A is part of some key for R.

Same as in BCNF

- *Minimality* of a key is crucial in third condition above!
  - If A is part of some superkey, then this condition would be true for any relation (because we can add any additional attribute to the superkey to make a bigger superkey).

3NF is a Compromise



- Let R be a relation with the set of FDs F, X be a subset of the attributes, and A be an attribute of R
- Relation R is in 3NF if, for all $X \longrightarrow A$ in F
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey (containing some key), or
  - A is part of some key for R.

Same as in BCNF

Observation:

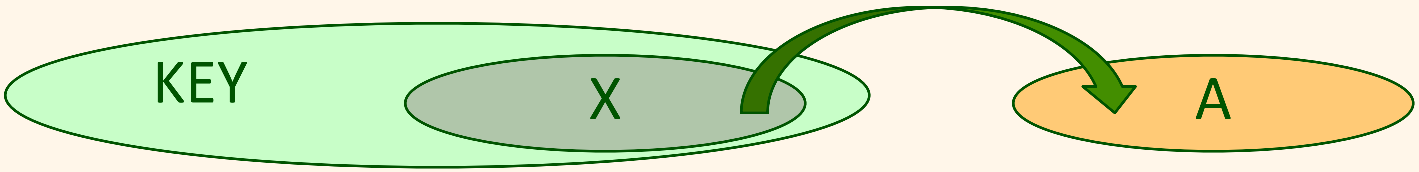
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).

Discussed later

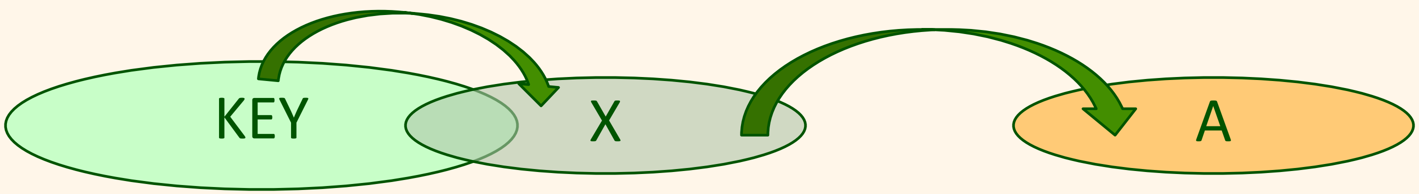
Compromise (1)

Suppose $X \rightarrow A$ causes a violation of 3NF $\Rightarrow$ There are two cases

CASE 1: *X* is a **proper subset** of some key *K* (partial dependency)

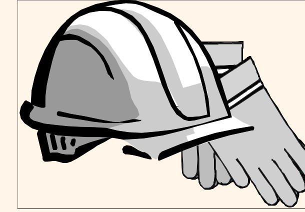


CASE 2: *X* is **not a proper subset** of any key (transitive dependency because we have a chain of dependencies KEY → X → A)



3NF

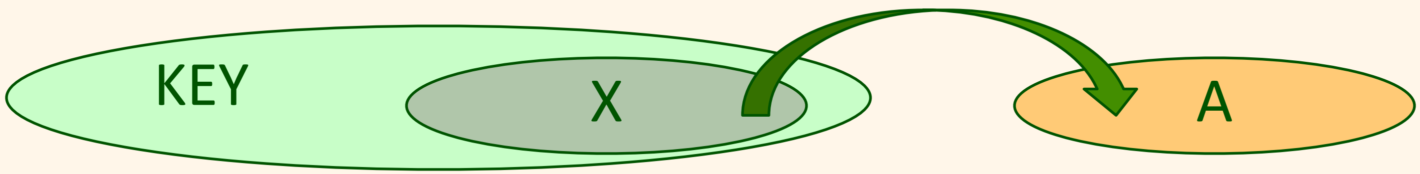
$A \in X$ (called a *trivial* FD), or
X is a superkey (containing some key), or
A is part of some key for *R*.



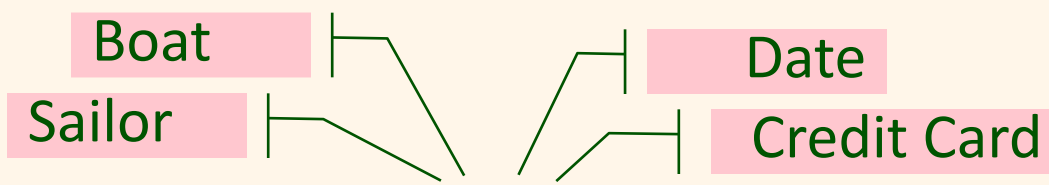
Compromise (2)

Suppose $X \rightarrow A$ causes a violation of 3NF $\Rightarrow$ There are two cases

CASE 1: X is a proper subset of some key K (partial dependency)



In this case we store (X,A) pairs redundantly



EXAMPLE: Relation ***Reserves(SBDC)*** with FD $S \longrightarrow C$

We store the credit card number for a sailor as many times as there are reservations for that sailor.

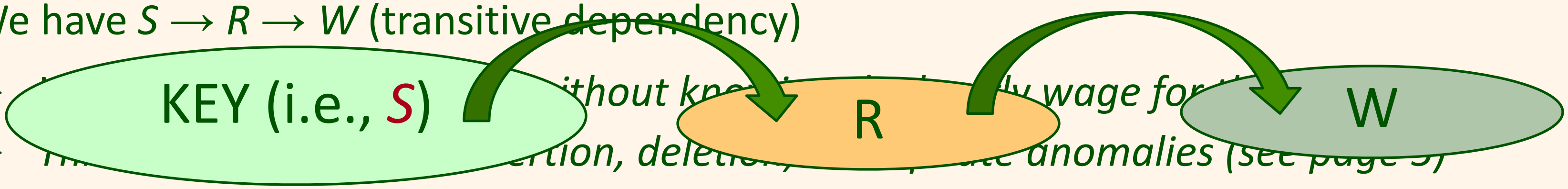
Compromise (3)

Suppose $X \rightarrow A$ causes a violation of 3NF $\Rightarrow$ There are two cases

CASE 2: X is not a proper subset of any key (transitive dependency)

EXAMPLE: ***Hourly_Emps(SNLRWH)*** with FD $R \rightarrow W$ (i.e., rating determines wage)

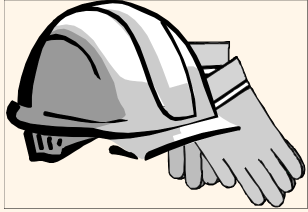
We have $S \rightarrow R \rightarrow W$ (transitive dependency)



⇒ *… ithout kn… wage for …*

⇒ *… ertion, deletion, … anomalies (see page 5)*

Redundancy in 3NF



<u>EXAMPLE</u>: Relation *Reserves*(<u>*SBD*</u>*C*) with the FD's

$S \longrightarrow C$ and $C \longrightarrow S$

S is part of the key $\Rightarrow$ "$C \longrightarrow S$" does not violate 3NF 1

$C \longrightarrow S$ (*i.e.*, **C**redit card uniquely identifies the **s**ailor)

$\Rightarrow$ $CBD \longrightarrow SBD$ (Augmentation axiom)

$\Rightarrow$ $CBD \longrightarrow SBD \longrightarrow SBDC$ (*SBD* is a key)

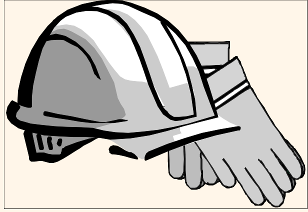
$\Rightarrow$ *CBD* is also a key of *Reserves* (Transitivity axiom)

$\Rightarrow$ "$S \longrightarrow C$" does not violate 3NF because *C* is part of the key 2

1 & 2 $\Rightarrow$ *Reserves* relation is in 3NF. Nonetheless, the same (*S*,*C*) pair is redundantly recorded for all tuples with the same *S* value.

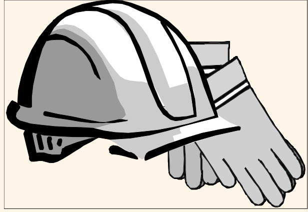
3NF is indeed a compromise relative to BCNF

Motivation for 3NF



- 3NF weakens the BCNF requirements just enough to ensure that every relation can be decomposed into a collection of 3NF relations
  - *Lossless-join & dependency-preserving decomposition of R into a collection of 3NF relations always possible.*
- The above guarantee does not exist for BCNF relations

Decomposition of a Relation Schema

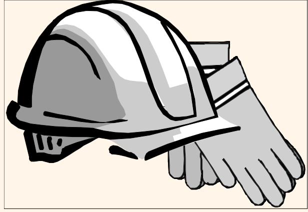


- Suppose that relation *R* contains attributes *A1 ... An.* A ***decomposition*** of *R* consists of replacing *R* by two or more relations such that:
  - Each new relation schema contains a subset of the attributes of *R* (and no attributes that do not appear in *R*), and
  - Every attribute of *R* appears as an attribute in at least one of the new relations.

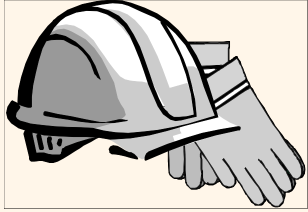
Decomposition

A	B	C	D	E

Decomposition of a Relation Schema



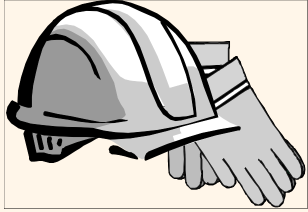
- Intuitively, decomposing *R* means we will store instances of the relation schemas produced by the decomposition, instead of instances of *R*.
- E.g., Can decompose *SNLRWH* into *SNLRH* and *RW*.



Example Decomposition

- Decompositions should be used only when needed.
  - *SNLRWH* has FDs $S \longrightarrow SNLRWH$ and $R \longrightarrow W$
  - $R \longrightarrow W$ causes violation of 3NF; W values repeatedly associated with R values.
    - Solution: Decompose SNL**RW**H into SNL**R**H 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 ?
  - We might have duplicates in RW

Problems with Decompositions



decompose SNLRWH into SNLRH and RW

- There are three potential problems to consider:
  1. Some queries become more expensive.
     - e.g., How much did sailor Joe earn? (salary = W*H)
  2. 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.

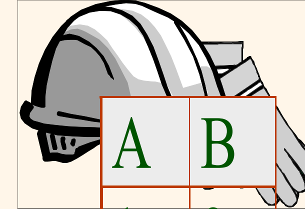
Lossless Join

  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, not in the SNLRWH example.

Dependency Preserving

- Tradeoff: Must consider these issues vs. redundancy.

Lossless Join Decompositions



- Decomposition of *R* into *X* and *Y* is <u>*lossless-join*</u> w.r.t. a set of FDs *F* if, for every instance *r* that satisfies F, we have $\pi_X(r) \bowtie \pi_Y(r) = r$
- 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.

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

A	B
1	2
4	5
7	2

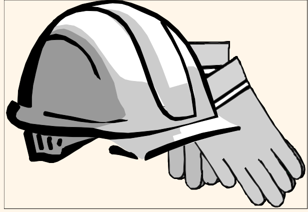
B	C
2	3
5	6
2	8

NOT LOSSLESS JOIN

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

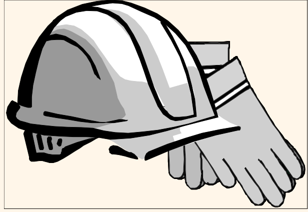
Join two tables on *B*

Lossless Join Decompositions



- 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) in page 30)*

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 ∩ Y → X, or
  - X ∩ Y → Y

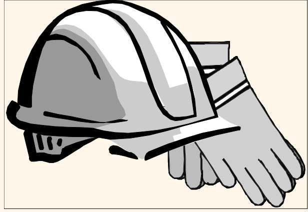
The intersection must be the primary key of X or Y

Example:
Decompose *SNLRWH* into *SNLRH* and *RW*

Foreign key

The intersection "***R***" is the primary key

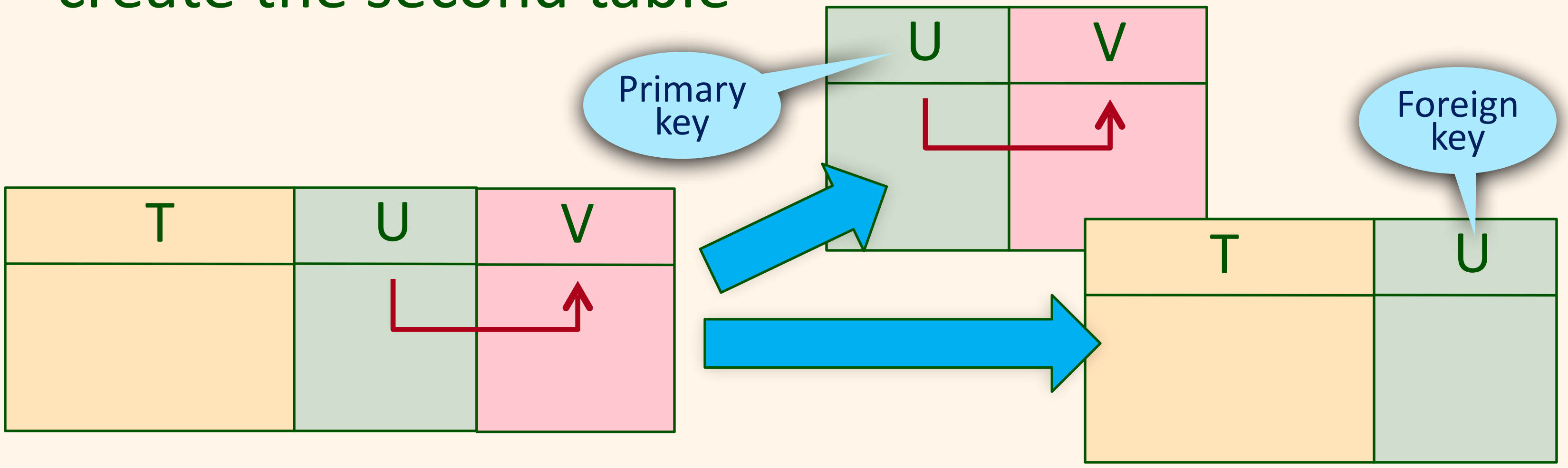
- In particular, the decomposition of *R* into *UV* and *R-V* is lossless-join if *U*→*V* holds over *R*. (Next page)



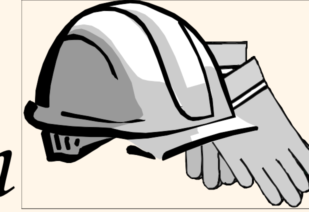
Loss-Less Join Decomposition

the decomposition of *R* into ***UV*** and ***R-V*** is lossless-join if *U*→*V* holds over *R*.

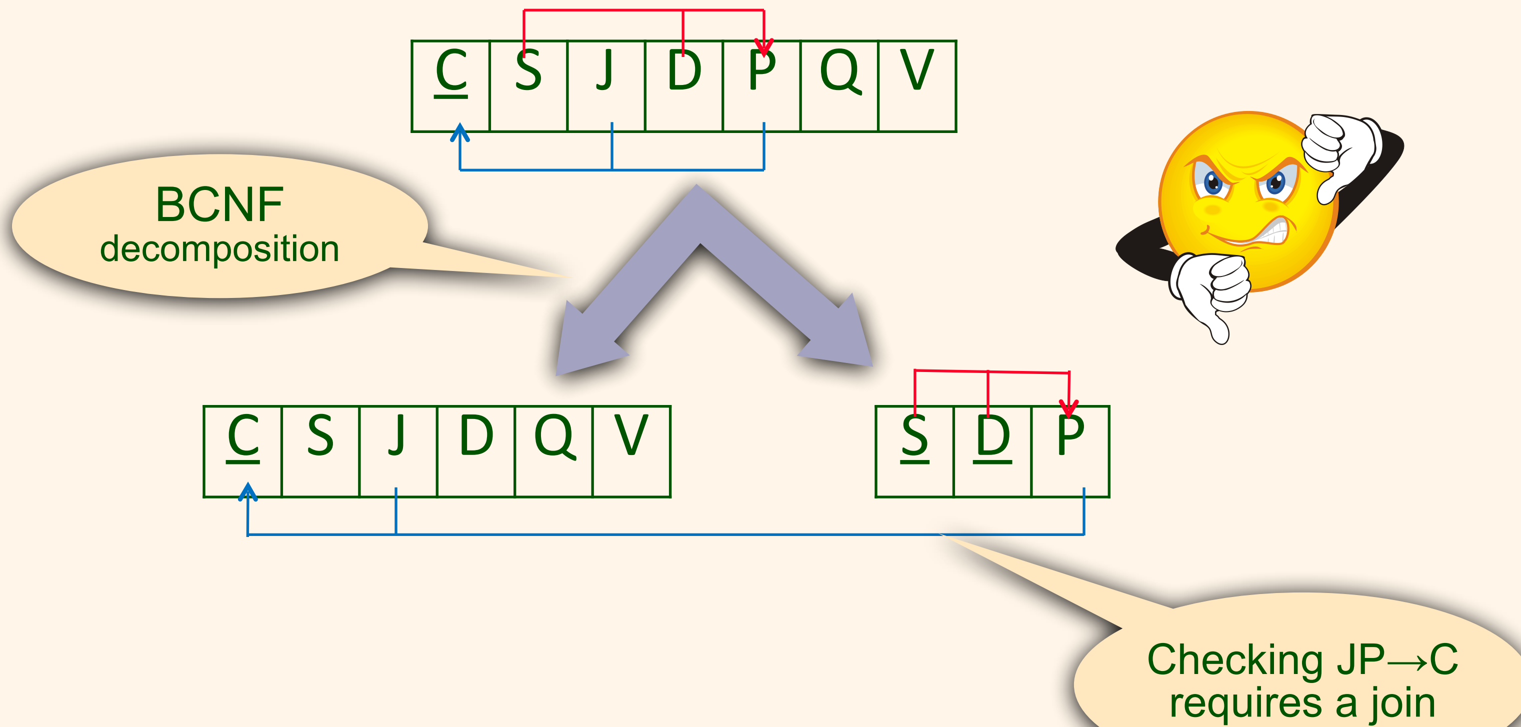
1. Given *U*→*V*, we create the first table as ***UV***
2. Remove *V* from the original table (i.e., ***R-V***) to create the second table



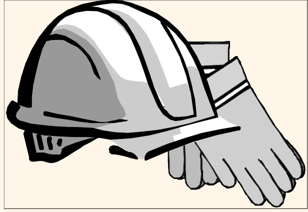
Dependency Preserving Decomposition



- Consider *<u>C</u>SJDPQV*, *C* is key, *JP* → *C* and ***SD* →*P***.
  - BCNF decomposition: *<u>C</u>SJDQV* and ***<u>SD</u>P***
  - Problem: Checking *JP* → *C* requires a join!

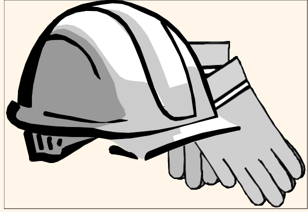


Dependency Preserving Decomposition



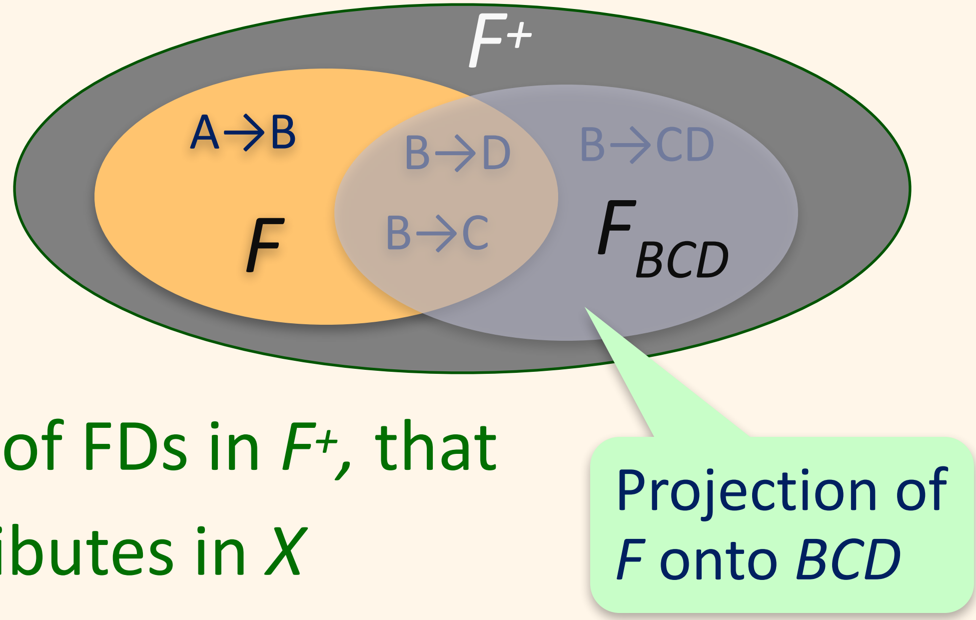
If *R* is decomposed into *X*, *Y* and *Z*, and we enforce the FDs that hold on *X*, on *Y* and on *Z*, then all FDs that were given to hold on *R* must also hold.
(Avoids Problem (3) in page 30.)

Projection of a Set of FDs

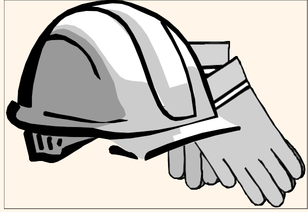


If *R* is decomposed into *X*, ... projection of *F* onto *X* (denoted F_X) is the set of FDs $U \rightarrow V$ in F^+ such that *U*, *V* are in *X*

R (*A*, *B*, *C*, *D*, *E*)
F: Set of FD's



- i.e., F_X is the set of FDs in F^+, that involve only attributes in *X*
- Important to consider F^+, not *F*, in this definition (see page 40)



Dependency Preserving Decomposition vs. Lossless Join Decomposition

- Dependency preserving does not imply lossless join:
  - *ABC*, with $F = \{A \rightarrow B\}$, decomposed into *AB* and *BC*.
  - $F_{AB} = \{A \rightarrow B\}$ and $F_{BC} = \phi$

  $\Rightarrow (F_{AB} \cup F_{BC})^+ = \{A \rightarrow B\} = F^+$

  $\Rightarrow$ The decomposition is dependency preserving.

  Nonetheless, it is not a lossless-join decomposition !

The intersection “B” is not a key of either relation

- And vice-versa!

Consider *CSJDPQV*, *C* is key, $JP \rightarrow C$ and $SD \rightarrow P$

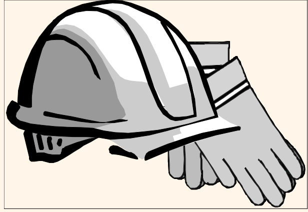
BCNF decomposition: *CSJDQV* and *SDP*

Problem: Checking $JP \rightarrow C$ requires a join!

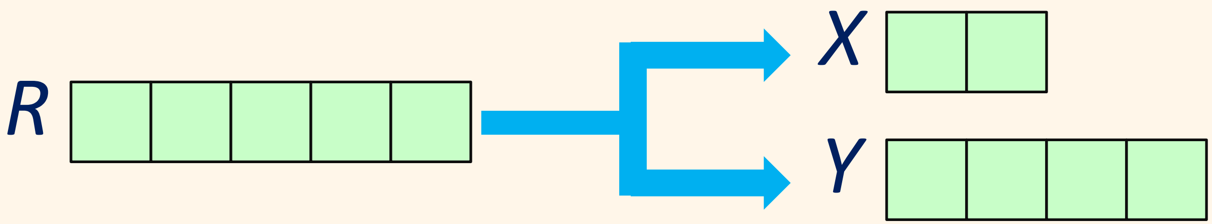
Lossless Join decomposition

Not dependency preserving !

Dependency Preserving Decompositions (Contd.)

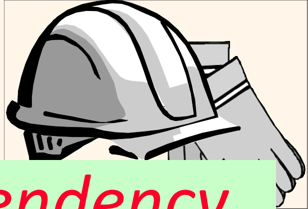


Decomposition of R into X and Y is ***dependency preserving*** if $(F_X \cup F_Y)^+ = F^+$



- We need to enforce only the dependencies in F_X and F_Y; and all FDs in F^+ are then sure to be satisfied
- To enforce F_X, we need to examine only relation X on inserts to that relation.
- To enforce F_Y, we need to examine only relation Y.

Dependency Preserving Decompositions: Example



Decomposition of R into X and Y is ***dependency preserving*** if $(F_X \cup F_Y)^+ = F^+$

ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.

$\Rightarrow F^+ = F \cup \{ A \rightarrow C, B \rightarrow A, C \rightarrow B \}$

If we consider only F (instead of F^+) $\Rightarrow$ $F_{AB} = \{A \rightarrow B\}$ and $F_{BC} = \{B \rightarrow C\}$

$\Rightarrow (F_{AB} \cup F_{BC})^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\} \neq F^+$

$\Rightarrow$ not dependency preserving ??

We need to examine F^+ (not F) when computing F_{AB} & F_{BC}

$\Rightarrow F_{AB} = \{A \rightarrow B, B \rightarrow A\}$ and $F_{BC} = \{B \rightarrow C, C \rightarrow B\}$

$\Rightarrow F_{AB} \cup F_{BC} = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B\}$

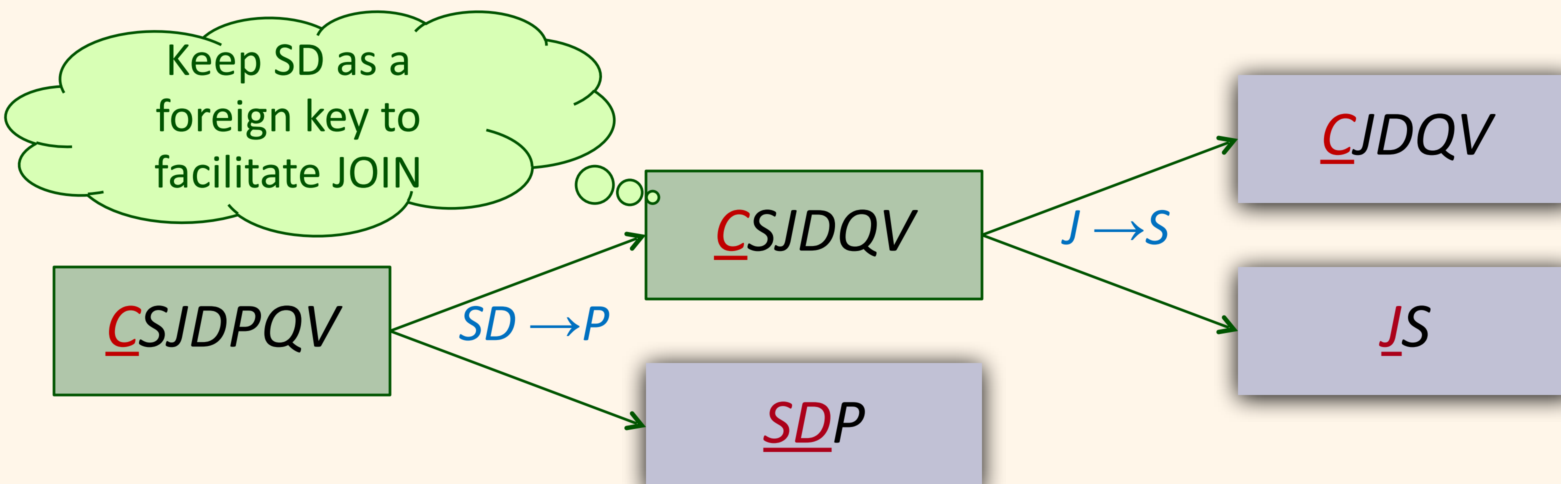
$\Rightarrow (F_{AB} \cup F_{BC})^+ = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, A \rightarrow C, C \rightarrow A\} = F^+$

$\Rightarrow$ The decomposition preserves the dependencies !!

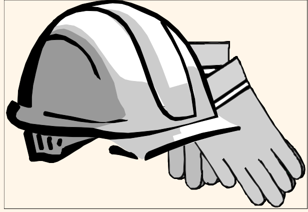
Decomposition into BCNF

- Consider relation *R* with FDs *F*. If X $\longrightarrow$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.

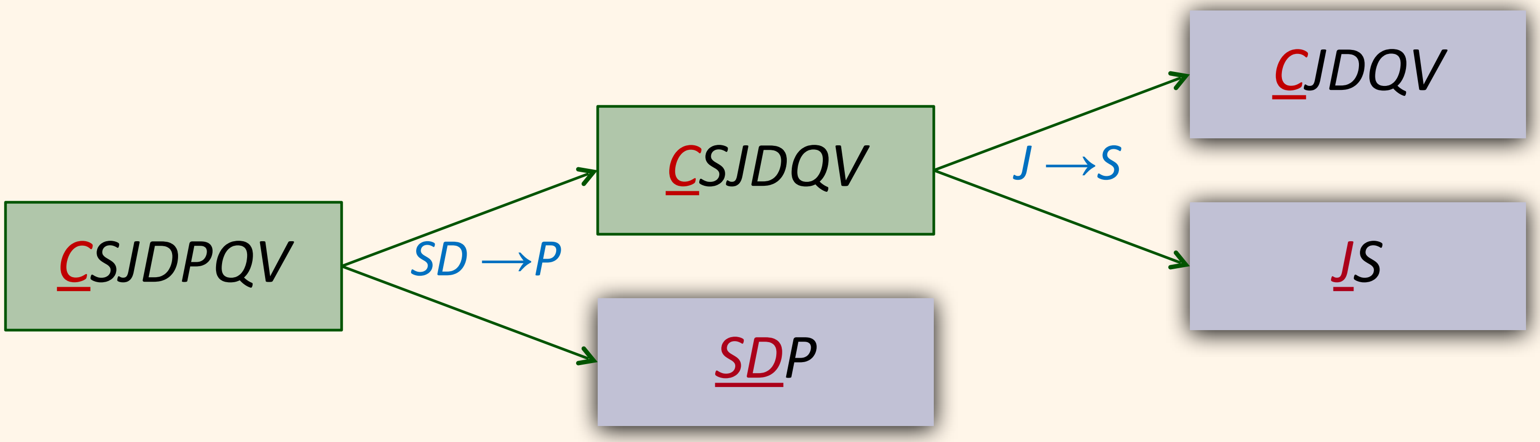
Example: *CSJDPQV*, key *C*, *JP* $\longrightarrow$*C*, *SD* $\longrightarrow$*P*, *J* $\longrightarrow$*S*



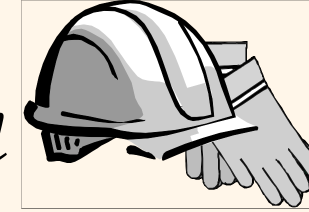
Decomposition into BCNF



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!

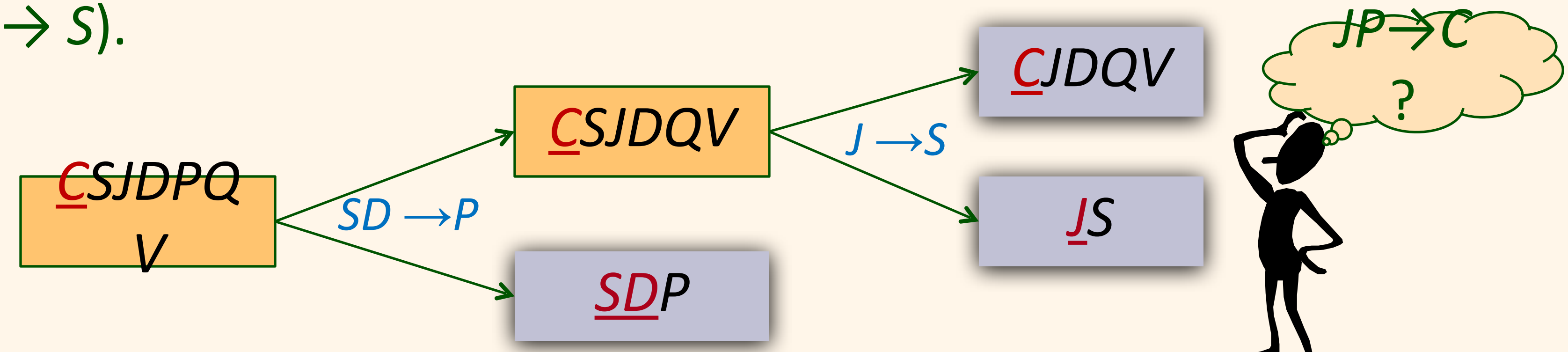


BCNF & Dependency Preservation



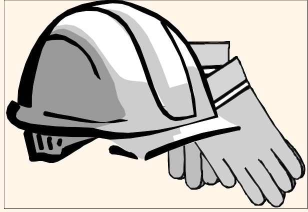
In general, there may not be a dependency preserving decomposition into BCNF

Example 2: Decomposition of *CSJDQV* into *SDP, JS,* and *CJDQV* is not dependency preserving (w.r.t. the FDs *JP* → *C*, *SD* → *P*, and *J* → *S*).



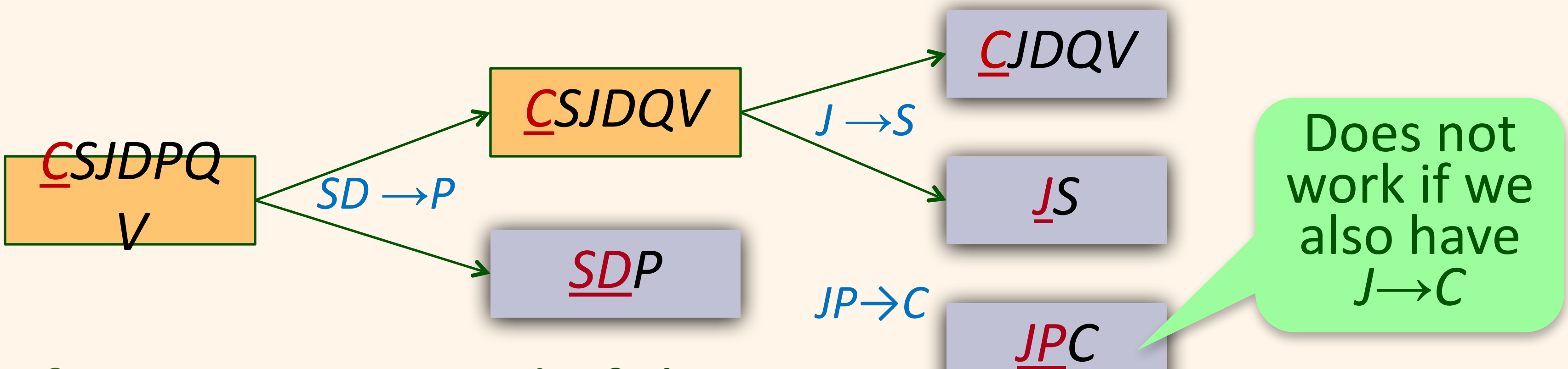
- This is a lossless join decomposition. However *JP* → *C* is not preserved
- Adding *JPC* to the collection of relations gives us a dependency preserving decomposition.
  - *JPC* tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF



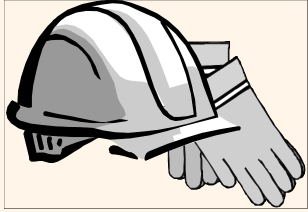
To ensure dependency preservation, one idea:

- If $X \rightarrow Y$ is not preserved, add relation *XY*.
- Problem is that *XY* may violate 3NF!
- Example: In the following decomposition, adding *JPC* to preserve $JP \rightarrow C$ does not work if we also have $J \rightarrow C$.



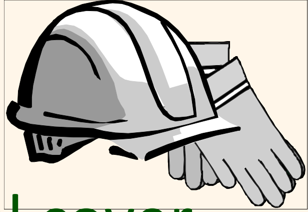
Refinement: Instead of the given set of FDs *F*, use a *minimal cover for F* (another form of *F*).

Minimal Cover for a Set of FDs



- *Minimal cover* *G* for a set of FDs *F* is a set of FDs such that
  - 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 (i.e., it is minimal).
- Intuitively, every FD in *G* is needed, and ``*as small as possible*'' in order to get the same closure as *F*.

Minimal Cover: Example



$A \rightarrow B$, $EF \rightarrow GH$, $ABCD \rightarrow E$, $ACDF \rightarrow EG$ Compute minimal cover

Initial Set = {$A \rightarrow B$, $EF \rightarrow G$, $EF \rightarrow H$, $ABCD \rightarrow E$, $ACDF \rightarrow E$, $ACDF \rightarrow G$}

- We keep $\boxed{A \rightarrow B,}$ $\boxed{EF \rightarrow G,}$ and $\boxed{EF \rightarrow H}$ because they are minimal and cannot be inferred from the other FDs in the initial set
- We can remove B from $ABCD \rightarrow E$ because

  $A \rightarrow B \Rightarrow AACD \rightarrow \boxed{BACD \rightarrow E \Rightarrow ACD \rightarrow E}$

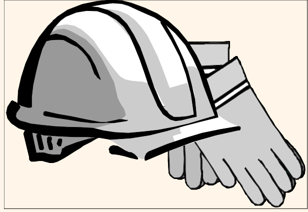
  We also have $\boxed{ACD \rightarrow E \Rightarrow ABCD \rightarrow E}$

  $\Rightarrow$ $ABCD \rightarrow E$ and $\boxed{ACD \rightarrow E}$ are equivalent
- We do not keep $ACDF \rightarrow E$ because we already keep $ACD \rightarrow E$ (i.e., F can be removed)
- We do not keep $ACDF \rightarrow G$ because we already keep $ACD \rightarrow E$

  $ACD \rightarrow E \Rightarrow ACDF \rightarrow EF \rightarrow G \Rightarrow ACDF \rightarrow G$

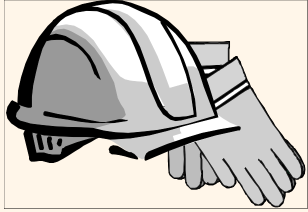
- A minimal Cover for F is {$A \rightarrow B$, $EF \rightarrow G$, $EF \rightarrow H$, $ACD \rightarrow E$}

Minimal Cover: Algorithm



- Put FDs in the standard form: Obtain a collection *G* of equivalent FDs with a single attribute on the right side (i.e., the initial set)
- Minimize the left side of each FD: For each FD in *G*, check each attribute in the left side to determine if it can be deleted while preserving equivalence to F^+
- Delete redundant FDs: Check each remaining FD in *G* to determine if it can be deleted while preserving equivalence to F^+

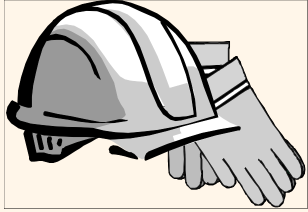
Lossless-Join Dependency-Preserving Decomposition into 3NF

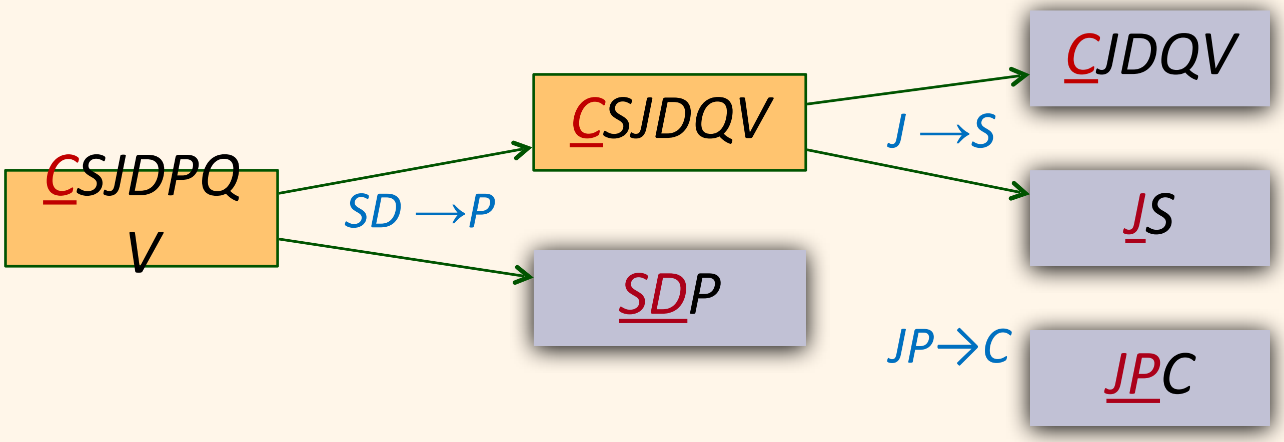


1. Compute a minimal cover *F* of the original set of FDs
2. Obtain a lossless-join decomposition: $R_1, R_2, \ldots R_n$ such that each one is in 3NF
3. Determine the projection of *F* onto each R_i, i.e., F_i
4. Identify the set *N* of FDs in F that is not preserved, i.e., not included in $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$
5. For each FD $X \rightarrow A$ in *N*, create a relation schema *XA* and add it to the result set

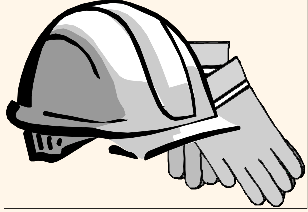
Optimization: If *N* contains $\{X \rightarrow A_1, X \rightarrow A_2, \ldots X \rightarrow A_k\}$, replace them with $X \rightarrow A_1A_2\ldots A_k$ to produce only one relation

Lossless-Join & Dependency-Preserving 3NF Example





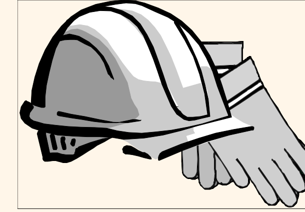
- A minimal cover: $F = \{SD \rightarrow P, J \rightarrow S, \mathbf{JP \rightarrow C}\}$
- $F_{SDP} = \{SD \rightarrow P\}$, $F_{JS} = \{J \rightarrow S\} \Rightarrow (F_{SDP} \cup F_{JS})^{+} = \{SD \rightarrow P, J \rightarrow S\}$
- $\mathbf{JP \rightarrow C}$ is not in $(F_{SDP} \cup F_{JS})^{+} \Rightarrow$ add relation *JPC*
- (SDP, CJDQV, JS, and JPC) is a lossless-join dependency preserving 3NF decomposition



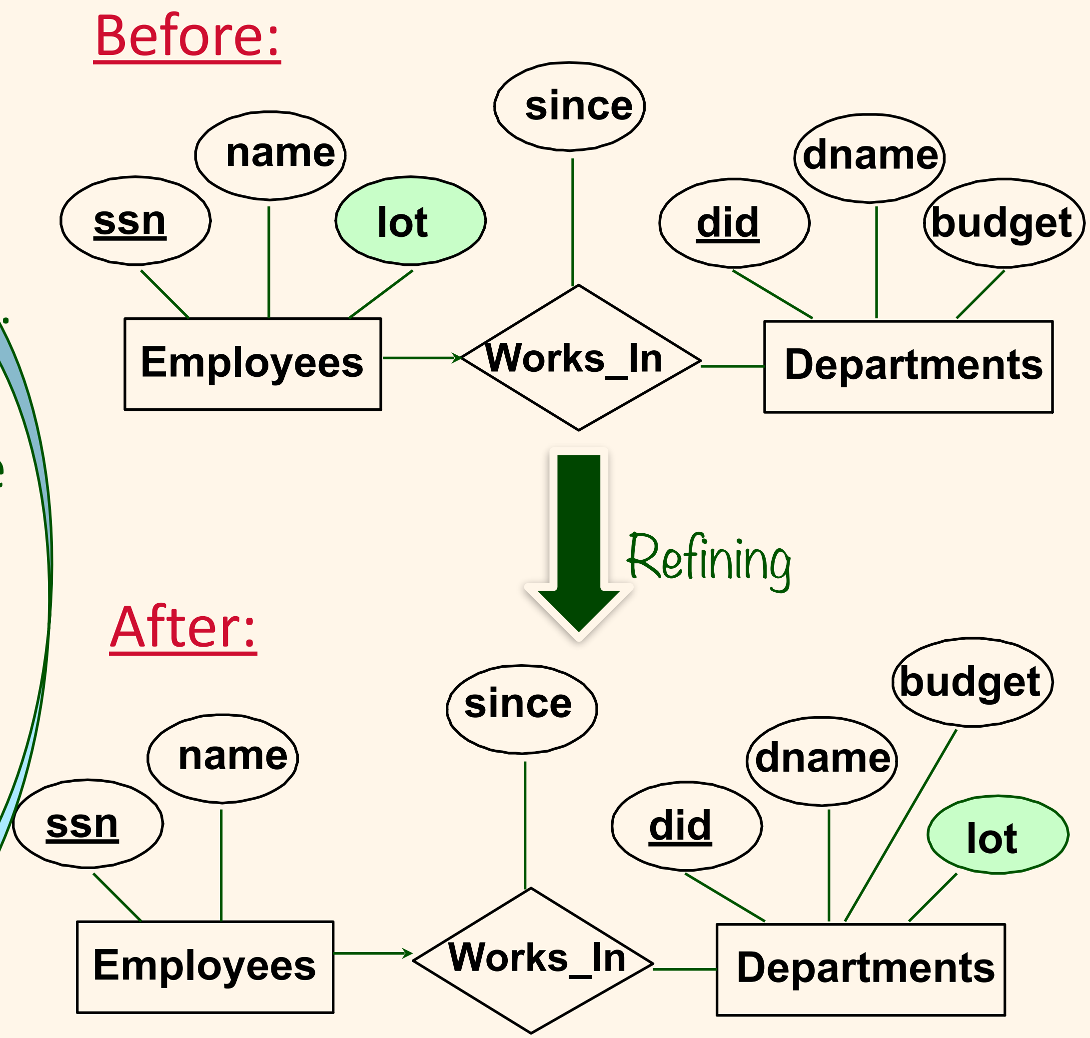
Good News !

A decomposition into 3NF relations that is lossless-join and dependency-preserving is always possible

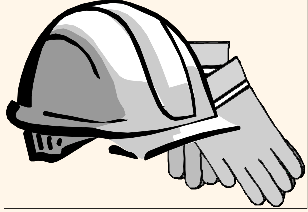
Refining an ER Diagram



- 1st diagram translated:
  Workers(S,N,L,D,Si)
  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,Si)
  Dept_Lots(D,L)
- Can fine-tune this:
  Workers2(S,N,D,Si)
  Departments(D,M,B,L)



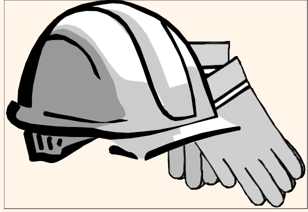
Schema Refinement



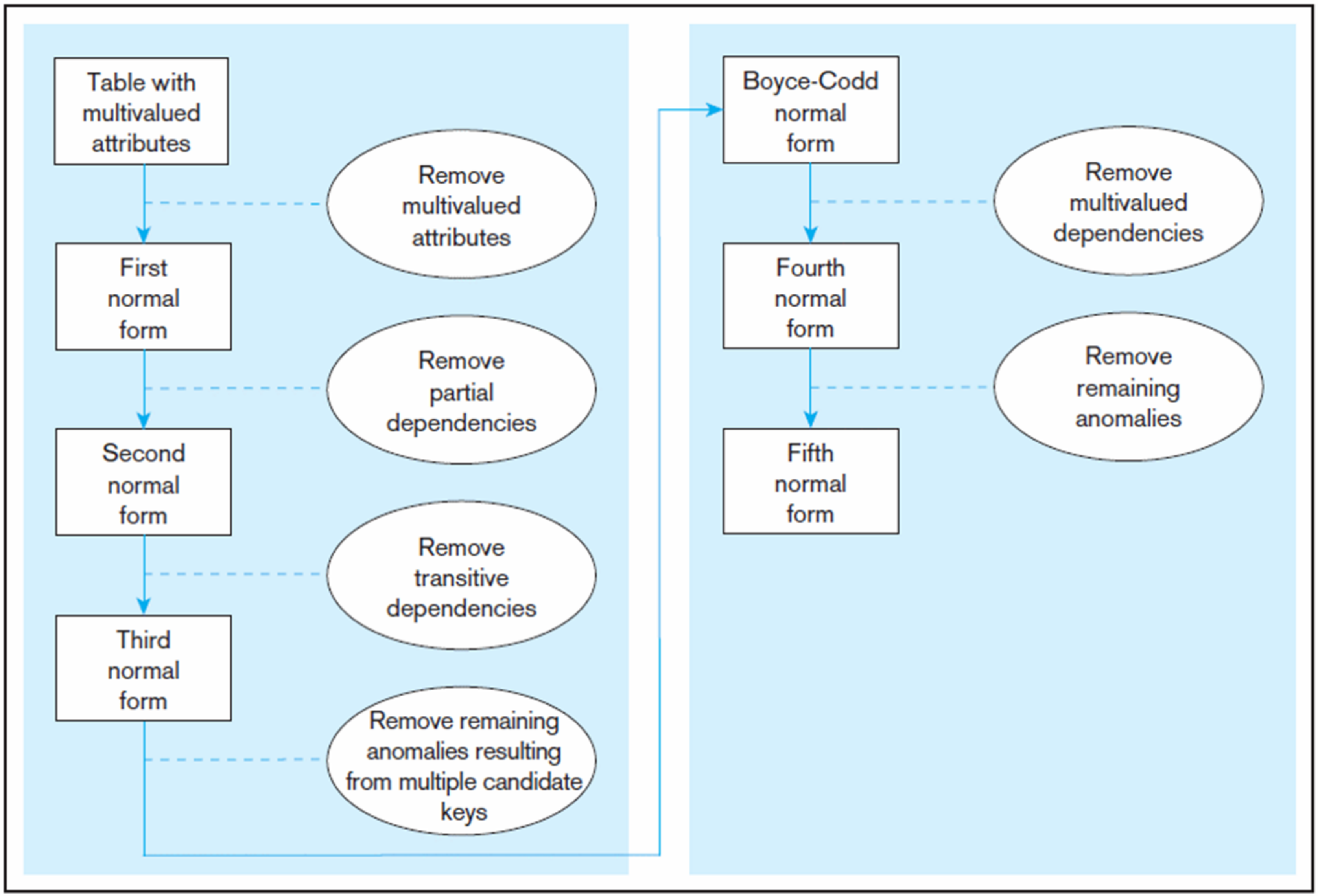
Should a good ER design not lead to a collection of relations free of redundancy problems ?

- ER design is a complex and subjective process
- Some FDs as constraints are not expressible in terms of ER diagrams
- Decomposition of relations produced through ER design might be necessary

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.



Modern Database Management11th Edition Jeffrey A. Hoffer, V. Ramesh, Heikki Topi