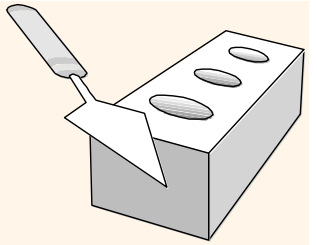
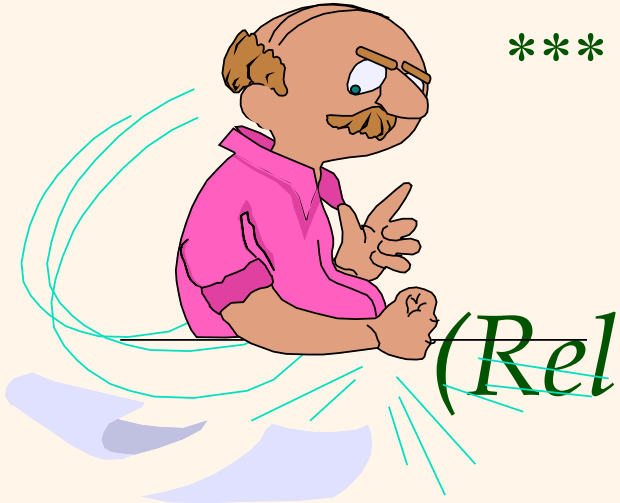


Introduction to Data Management



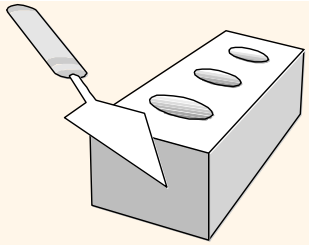
**** The "Flipped" Edition ****

Lecture #8

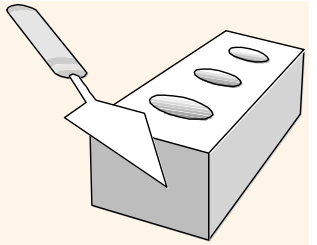
(Relational Design Theory II)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Today's Notices



- Keep one eye on the wiki page...
  - http://www.ics.uci.edu/~cs122a/
    - **Note the *Relational Design Theory* readings addition!**
- ... and the other eye on Piazza Q&A!
  - piazza.com/uci/fall2021/cs122aeecs116
- HW #2 should be well underway!
  - Its starting point: Our ***HW #1 E-R solution***
    - **Make sure that you're using the very latest version!**



Quick Roadmap Check...

Topic Coverage and Exam Schedule

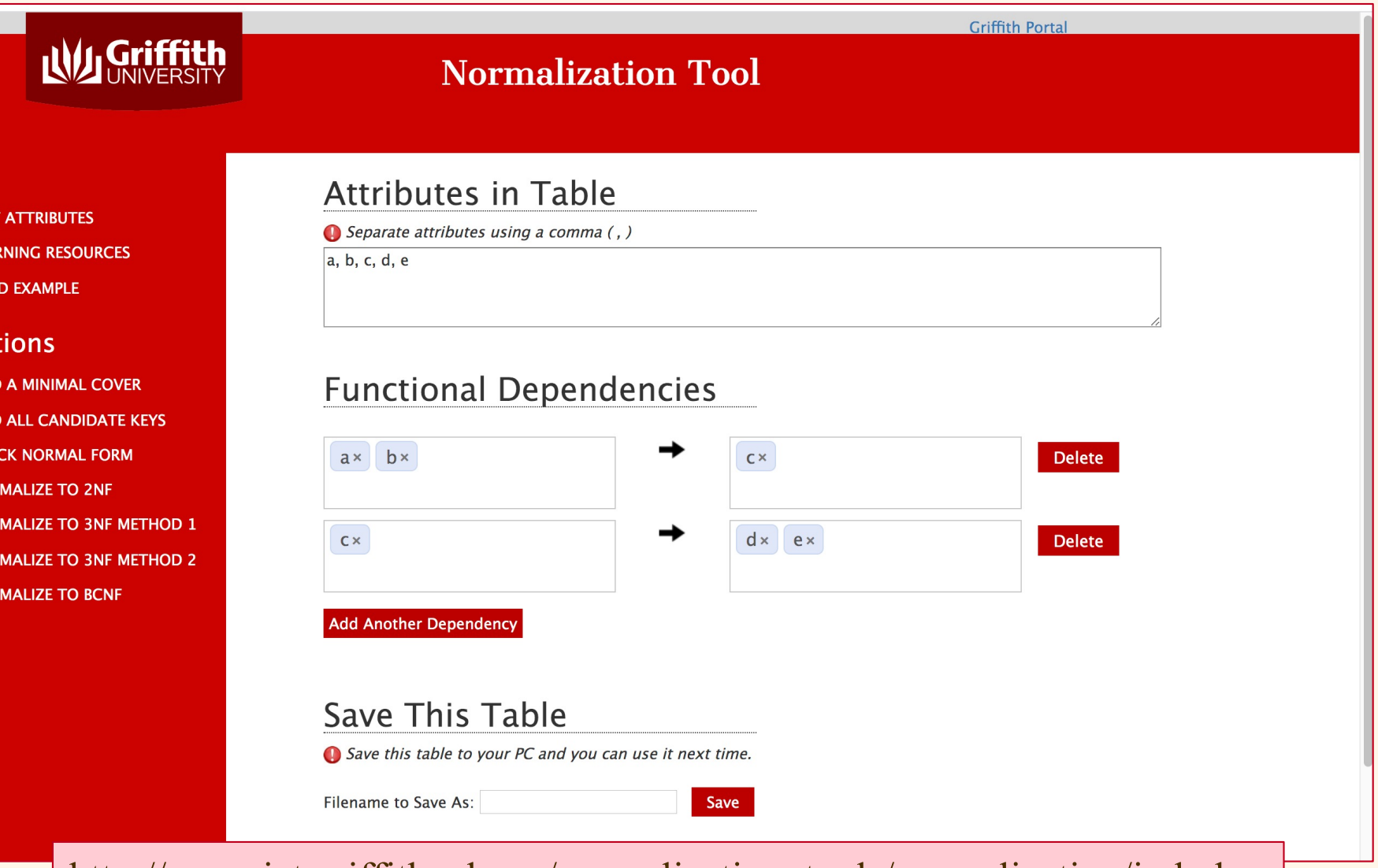
Syllabus

Topic	Reading (Required!)
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 6.1-6.5, 6.8-6.9
Relational Data Model	Ch. 2.1-2.4, 3.1-3.2
E-R to Relational Translation	Ch. 6.6-6.7
Relational Design Theory	Ch. 7.1-7.4.2
Midterm Exam 1	***Fri, Oct 22*** (during lecture time)
Relational Algebra	Ch. 2.5-2.7
Relational Calculus	Wikipedia: Tuple relational calculus
SQL Basics (SPJ and Nested Queries)	Ch. 3.3-3.5
SQL Analytics: Aggregation, Nulls, and Outer Joins	Ch. 3.6-3.9, 4.1
Advanced SQL: Constraints, Triggers, Views, and Security	Ch. 4.2, 4.4-4.5, 4.7
Midterm Exam 2	***Mon, Nov 15*** (during lecture time)
Storage	Ch. 12.1-12.4, 12.6-12.7
Indexing	Ch. 14.1-14.4, 14.5
Physical DB Design	Ch. 14.6-14.7, 15.1-15.3, 15.5.3
Semistructured Data Management (*a.k.a.* NoSQL)	Ch. 8.1, AsterixDB SQL++ Primer, Couchbase SQL++ Book
Data Science 1: Advanced SQL Analytics	Ch. 5.5, 11.3
Data Science 2: Notebooks, Dataframes, and Python/Pandas	Lecture notes and Jupyter notebook
Basics of Transactions	Ch. 4.3, Ch. 17
Endterm Exam	***Fri, Dec 3*** (during lecture time)

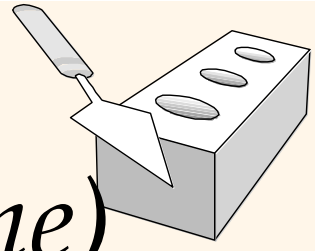
Midterm Exam 1

Time: Fri, Oct 22, Lecture Time
Place: SSLH 100

Web Tool for FDs & Normalization

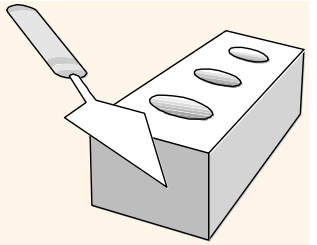


http://www.ict.griffith.edu.au/normalization_tools/normalization/ind.php



Terms & Definitions (From Last Time)

- If X is part of a (candidate) key, we will say that X is a ***prime attribute***.
- If X (an attribute set) contains a candidate key, we will say that X is a ***superkey***.
- X → Y can be pronounced as "X *determines* Y", or "Y is *functionally dependent* on X".
- Some types of dependencies (***on a key***):
  - ***Trivial***: **XY → X**
  - ***Partial***: **X**Y is a key, **X → Z** (note that Y is absent)
  - ***Transitive***: X → Y, Y → Z, Y is non-prime, **X → Z**



Second Normal Form (2NF)

- Rel'n R is in 2NF if it is in 1NF and no non-prime attribute is *partially* dependent on a candidate key of R.
- *Ex*: Supplies(sno, sname, saddr, pno, pname, pcolor)

  where: sno → sname, sno → saddr, pno → pname, pno → pcolor

  Q1: What are the candidate keys for Supplies?

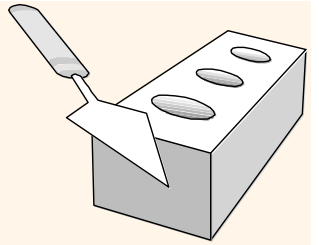
  Q2: What are the prime attributes for Supplies?

  Q3: Why is Supplies not in 2NF?

A1: (sno, pno)

A2: sno, pno

A3: *Each* of its four FDs violates 2NF!



Third Normal Form (3NF)

- Rel'n R is in 3NF if it is in 2NF and it has no *transitive* dependencies to non-prime attributes.
- *Ex*: Workers(eno, ename, esal, dno, dname, dfloor)

  where: eno→ename, eno→esal, eno→dno, dno→dname, dno→dfloor

  Q1: What are the candidate keys for Workers?

  Q2: What are the prime attributes for Workers?

  Q3: Why is Workers not in 3NF?

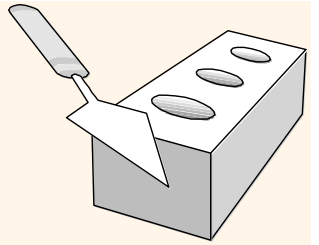
  A1: eno

  A2: eno

  A3: Two inferable FDs, eno→ dname and eno→dfloor, each violate 3NF.

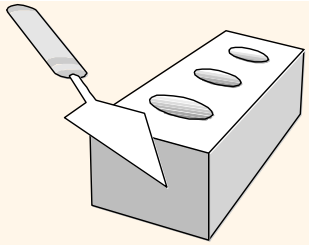
Note: *A lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is* ***always*** *possible.*

Boyce-Codd Normal Form (BCNF)



- Rel'n R with FDs *F* is in BCNF if, for all X → A in *F*+
  - A ∈ X (***trivial* FD**), or else
  - *X is a superkey* (i.e., *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*** *(key → attr)*
  - Everything depends on "the key, the whole key, and nothing but the key" (so help me Codd ☺)

Boyce-Codd Normal Form (Cont'd.)



- *Ex*: Supply2(sno, sname, pno)

  Given FDs: sno → sname, sname → sno

  Q1: What are the candidate keys for Supply2?

  Q2: What are the prime attributes for Supply2?

  Q3 Is Supply2 in 3NF?

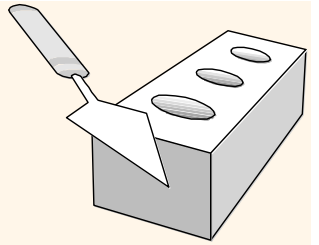
  Q4: Why is Supply2 not in BCNF?

Note: Overlapping...!

A1: (sno, pno), (sname, pno)
A2: sno, sname, pno
A3: Yes, it is in 3NF.
A4: Each of its FDs has a left-hand-side that isn't a candidate key. (Just a *part* of one.)

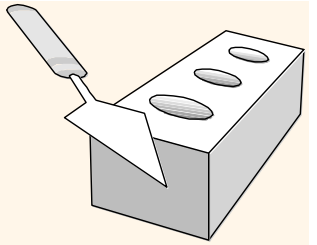
Warning: *A lossless-join, dependency-preserving decomposition of R into a collection of BCNF relations* ***is not*** *always possible.*

3NF Revisited (Alternative Def'n)

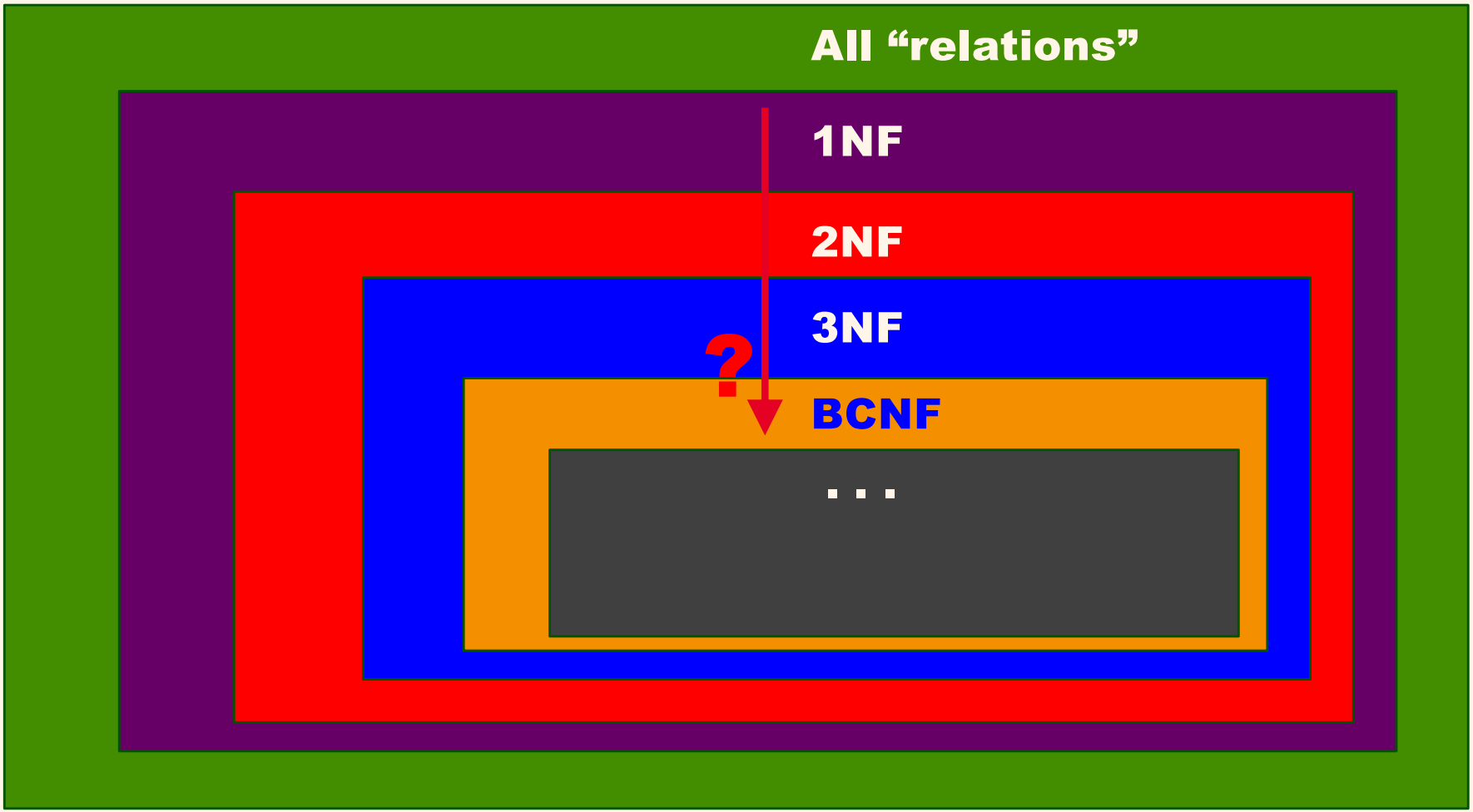


- Rel'n R with FDs *F* is in 3NF if, for all X → A in F+
  - A ∈ X (*trivial* FD), or else
  - X is a superkey (i.e., contains a key) for R, or else
  - A is part of some key for R (i.e., it's a prime attribute).
- If R is in BCNF, clearly it is also in 3NF.
- If R is in 3NF, some redundancy is possible. 3NF is a compromise to use when BCNF isn't achievable (e.g., no "good" decomp, or performance considerations).

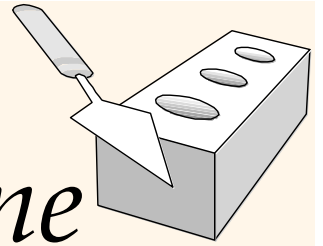
***Again**: A lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations **is always** possible.*



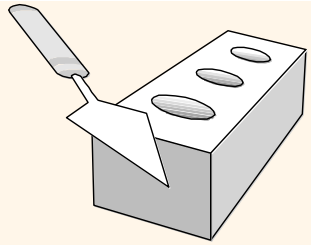
Achieving Normal Forms?



Decomposition of a Relation Scheme

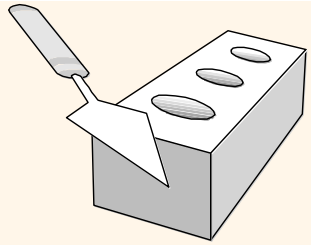


- Suppose a relation R contains attributes *A1 ... An*. A *decomposition* of R consists of ***replacing*** R by two or more relations such that:
  - Each new relation contains a subset of the attributes of R (and no attributes that did not appear in R ☺), and
  - Every attribute of R appears as an attribute of at least one of the new relations.
- Intuitively, decomposing R means we will store instances of the relations that result from our decomposition ***instead*** of storing instances of R.
- E.g., decompose SNLRWH into RW and SNLRH...



An Example Decomposition

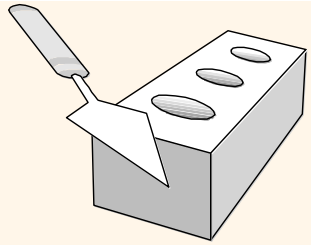
- Decompositions should be used (only) when needed.
  - Our **SNLRWH** has 2 FDs: S → SNLRWH and R → W.
  - R → W violates 3NF (so W values are repeatedly associated with R values). The fix creates a relation RW to store the associations, then removes W from the main schema:
    - I.e.: Decompose **SNLRWH** into **SNLRH** and **RW**.
- The original information to be stored consisted of SNLRWH tuples. *Q:* If we store the "projections" of those tuples onto SNL**R**H and **R**W, are there any potential issues that we should be aware of? ...→



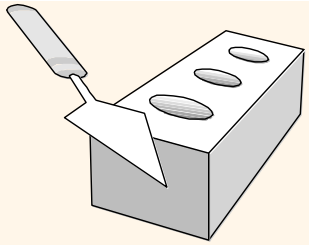
Decompositions: Possible Issues

- There are three potential "problems" to consider:
  1. Some queries will become more expensive.
     - E.g., how much did sailor Joe earn? (W*H will require a join)
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation! (If "lossy"...)
     - **Not** a problem for the SNLRWH example! (Thanks to ***R***)
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, also not a problem in the SNLRWH example.
- *Tradeoff*: Consider these issues *vs.* just tolerating the redundancy.

Lossless Join Decompositions

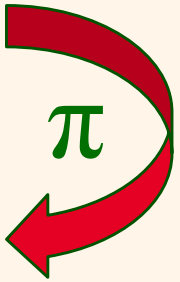


- Decomposition of **R** into **X** and **Y** is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$ (***Note**: relational algebra* ☺)
- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, then the decomposition is called ***lossless-join***.
  - Must ensure that relations X and Y ***overlap*** and the overlap contains a key for ***one*** of the two relations. *Ex:* SNL**R**H + **R**W
- The definition extends to decomposition into three or more relations as you would expect.
- *Decompositions **must** be lossless! (Avoids Problem (2).)*



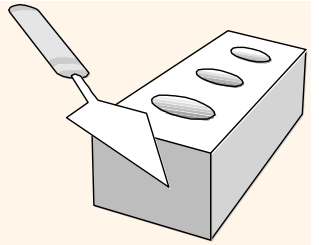
Ex: A Lossy Decomposition

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



S	N	L	W	H
123-22-3666	Attishoo	48	10	40
231-31-5368	Smiley	22	10	30
131-24-3650	Smethurst	35	7	30
434-26-3751	Guldu	35	7	32
612-67-4134	Madayan	35	10	40

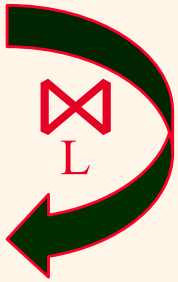
L	R
48	8
22	8
35	5
35	8



Ex: A Lossy Decomposition (cont.)

S	N	L	W	H
123-22-3666	Attishoo	48	10	40
231-31-5368	Smiley	22	10	30
131-24-3650	Smethurst	35	7	30
434-26-3751	Guldu	35	7	32
612-67-4134	Madayan	35	10	40

L	R
48	8
22	8
35	5
35	8



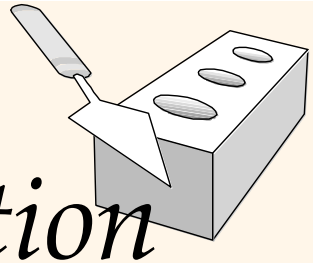
S5	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
131-24-3650	Smethurst	35	8	7	30
434-26-3751	Guldu	35	5	7	32
434-26-3751	Guldu	35	8	7	32
612-67-4134	Madayan	35	5	10	40
612-67-4134	Madayan	35	8	10	40

Whoops!

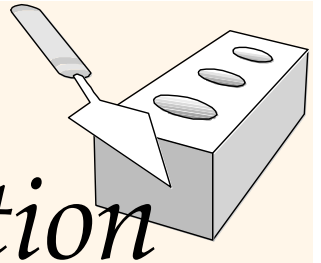
"Satisfaction guaranteed or double your data back..."

☺

Dependency Preserving Decomposition



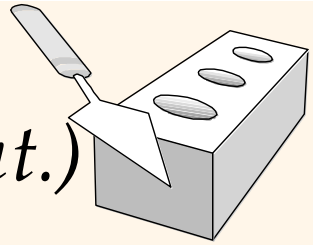
- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: Two tables, CSJDPV and SDP.
  - **Problem:** Checking JP → C now requires a join!
- Dependency preserving decomposition (intuitive):
  - I[illegible]po[illegible]Y and Z, and we enforce the FDs that hol[illegible]Y, and on Z, then 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 into **X** (denoted F_X) is the set of FDs U → V in F^+ (***closure*** *of F*) where U,V are *both* in **X**.



Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: Two tables, CSJDQV and SDP.
  - **Problem:** Checking JP → C now requires a join! (Oops)
- 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 into **X** (denoted F_X) is the set of FDs U → V in F^+ (***closure*** *of F*) where U,V are *both* in **X**.

Dependency Preserving Decomp. (Cont.)

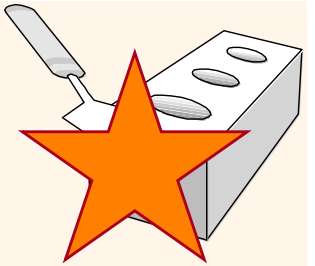


- The decomposition of R into two tables X and Y is *dependency preserving* if $(F_X \text{ union } F_Y)^+ = F^+$
  - I.e., if we consider only dependencies **in the closure** F^+ that can be checked in X **without** considering Y, and in Y **without** considering X, they ***imply*** all dependencies in F^+!
- Important that we are talking about **F^+**, not F, here:
  - *Ex:* EmpDeptMix(eid, email, ename, did, dname) with eid→email, email→eid, eid→ename, email→did, did→dname
    - Emp(eid, email, ename) - eid→email, email→eid, eid→ename
    - Dept(did, dname) - did→dname
    - Work(eid, did) - **eid→did** (instead of email→did)

Must check for both!

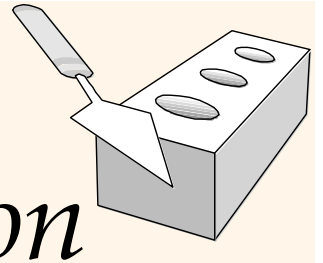
- Dependency preserving does ***not*** imply lossless join:
  - *Ex:* ABC with A→B, if decomposed into AB and C.

Decomposing a Design into BCNF

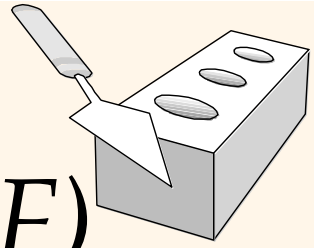


- Consider a relation R with FDs F. If **X → Y** *violates* BCNF, decompose R into R–**Y** and **XY**. *(R-Y has **X** still!)*
  - Repeated application of this idea will yield a collection of relations that are BCNF, a lossless join decomposition, and guaranteed to terminate. *(Didn't say dependency preserving...)*
- *Ex:* CSJDPQV with C→CSJDPQV, *JP*→*C*, *SD*→*P*, and *J*→*S*.
  - To deal with *SD*→*P*, decompose into SDP, CSJDQV.
  - To deal with *J*→*S*, decompose CSJDQV into JS and CJDQV.
- In general, *several* dependencies may violate BCNF. (The *order* that we process the dependencies in can lead to different decompositions, only *some* of which may be dependency preserving!)

BCNF and Dependency Preservation

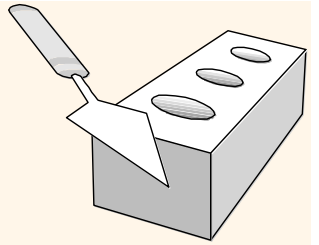


- In general, there simply may not ***be*** a dependency preserving decomposition into BCNF.
  - E.g., R(CSZ) with $CS \rightarrow Z$, $Z \rightarrow C$. (Candidate keys: CS, ZS)
  - Can't decompose and preserve the first FD; not in BCNF...
- Consider again decomposing the relation CSJDPQV into the relations SDP, JS and CJDQV:
  - *Not* dependency preserving (*w.r.t.* $\boldsymbol{JP \rightarrow C}$, $SD \rightarrow P$, $J \rightarrow S$).
  - However, it ***is*** indeed a lossless join decomposition.
  - In this case, ***adding*** relation JPC to the collection of relations would give us a dependency preserving decomposition.
    - But: JPC data would be used ***only*** for FD checking! (*Redundancy!*)



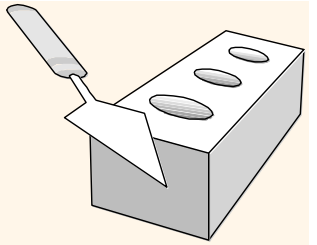
Decomposition into 3NF (vs. BCNF)

- The lossless join decomposition algorithm for BCNF can also be used to obtain a lossless join decomposition into 3NF (and might stop earlier).
- One idea to ensure dependency preservation:
  - If $X \rightarrow Y$ is not preserved in the resulting BCNF decomposition, add a relation XY.
  - Problem is that XY may violate 3NF (or even 2NF) due to other FDs, so this won't work in general.
- The **real** fix: Instead of using the ***given*** set of FDs F to guide the process, use a ***minimal cover*** for F to synthesize a decomposed schema.



Decomposition into 3NF (cont.)

- You are ***not*** going to be held responsible for being able to hand-generate a minimal cover and using it to synthesize a 3NF design...
  - There's an algorithm for that (see book for details)
  - There are tools that can do all this for you (see earlier URL)
- But – in case you ***are*** curious, I'll go ahead and "cover" that material (see what I did there? ☺) in an addendum video that you can choose to watch at your own risk!
  - It's fine to skip the extra video if you aren't that curious...
- Let's instead proceed to apply the BCNF approach from three slides back to some actual examples!
  - In many practical cases this approach will yield a decomposition that turns out to be dependency preserving (and you can check)



Ex: Second Normal Form (2NF)

- Rel'n R is in 2NF if it is in 1NF and no non-prime attribute is *partially* dependent on a candidate key of R.
- *Ex*: Supplies(sno, sname, saddr, pno, pname, pcolor)

  where: sno → sname, sno → saddr, pno → pname, pno → pcolor

  Q1: What are the candidate keys for Supplies?

  Q2: What are the prime attributes for Supplies?

  Q3: Why is Supplies **not** in 2NF?

  A1: (sno, pno)
  A2: sno, pno
  A3: *Each* of its four FDs violates 2NF!

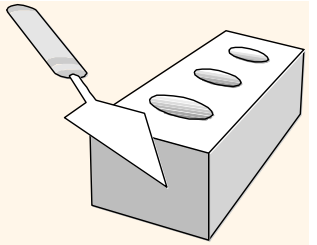
  Q4: So what's the fix?

  Supplier(sno, sname, saddr)
  Part(pno, pname, pcolor)
  Supply(sno, pno)

  Must not forget this!
  (Else "lossy join"!)

  Notice that...

  - We first factored out Supplier based on the sno → ... FDs
  - We then factored out Part based on the pno → ... FDs
  - That left us with Supply so "all was not lost" ☺



Ex: Third Normal Form (3NF)

- Rel'n R is in 3NF if it is in 2NF and it has no *transitive* dependencies to non-prime attributes.
- *Ex*: Workers(eno, ename, esal, dno, dname, dfloor)

  where: eno→ename, eno→esal, eno→dno, dno→dname, dno→dfloor

  Q1: What are the candidate keys for Workers?

  Q2: What are the prime attributes for Workers?

  Q3: Why is Workers not in 3NF?

  A1: eno
  A2: eno
  A3: Two inferable FDs, eno→ dname and eno→dfloor, each violate 3NF.

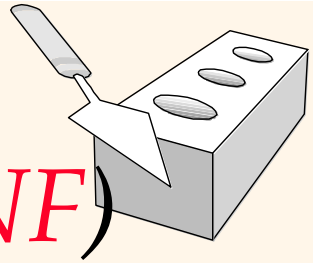
  Q4: So what's the fix?

  Emp(eno, ename, esal, dno)
  Dept(dno, dname, dfloor)

  Don't forget this!
  (Else "lossy join" !)

  Notice that...

  - We first factored out Dept based on the dno → ... FDs
  - That left us with Emp including dno, so again "all was not lost" ☺



Ex: Boyce-Codd Normal Form (*BCNF*)

- Rel'n R with FDs *F* is in BCNF if, for all X → A in *F*+
  - A ∈ X (***trivial* FD**), or else
  - *X is a superkey* (i.e., *X* **contains a key**) for R.
- *Ex*: Supply2(sno, sname, pno)

  Given FDs: sno → sname, sname → sno

  Q1: What are the candidate keys for Supply2?

  Q2: What are the prime attributes for Supply2?

  Q3 Is Supply2 in 3NF?

  Q4: Why is Supply2 not in BCNF?

  Q5: So what's the fix?

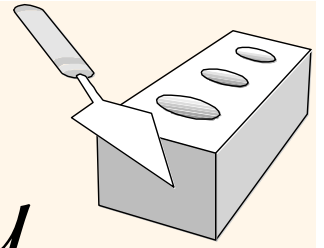
  Supplier2(sno, sname)
  Supplies2(sno, pno)

  Notice that...
  - We first factored out Supplier2 based on sno → sname
  - That left us with Supplies2 with sno so "all was not lost" ☺

Note: Overlapping...!

A1: (sno, pno), (sname, pno)
A2: sno, sname, pno
A3: Yes, it is in 3NF.
A4: Each of its FDs has a left-hand-side that isn't a candidate key. (Just a *part* of one.)

Relational Design Theory Summary

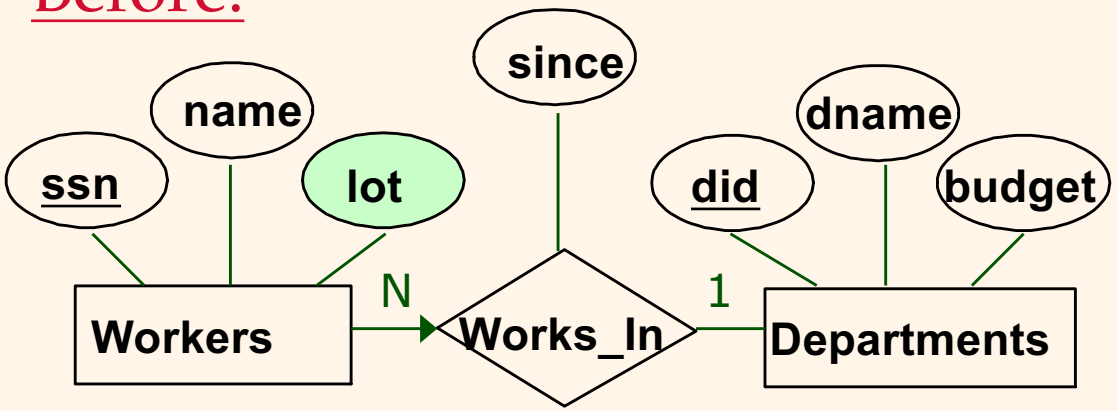


- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. (Trying to ensure that all relations are in BCNF is thus a good goal.)
- If a relation is not in BCNF, we will decompose it into a lossless-join collection of BCNF relations.
  - Are all FDs preserved? If a lossless-join, dependency-preserving decomposition into BCNF is not possible, consider 3NF instead.
  - Note: Decompositions should really be carried out while keeping *performance requirements* in mind. (More later!)

On Refining ER Based Designs

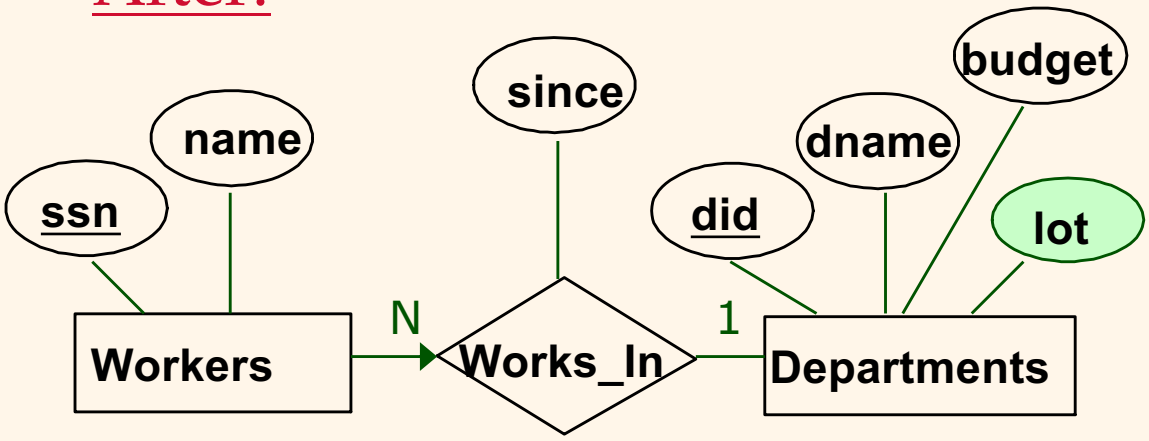
- 1st diagram translated:
  Workers(S,N,L,D,C)
  Departments(D,M,B)
  (Lots associated with workers.)
- *Suppose all workers in a dept are assigned the same lot: D → L ....*
- Redundancy! Fixed by:
  Workers2(S,N,D,C)
  WorkersLots(D,L)
  Departments(D,M,B)
- Can further fine-tune this:
  Workers2(S,N,D,C)
  Departments(D,M,B,L)

Before:



Notice: Lot **wasn't** really a "Worker attribute"!

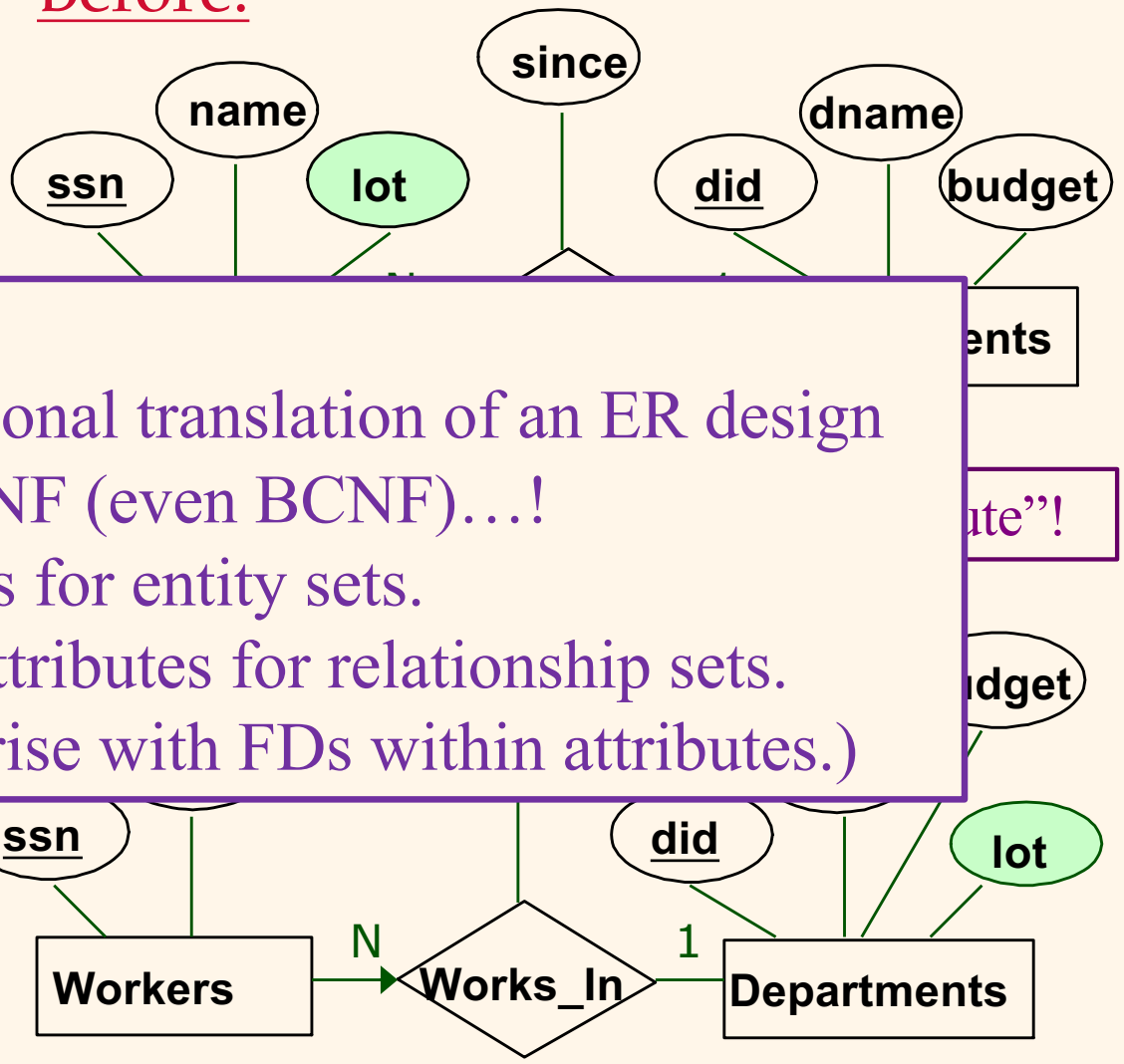
After:



On Refining ER Based Designs

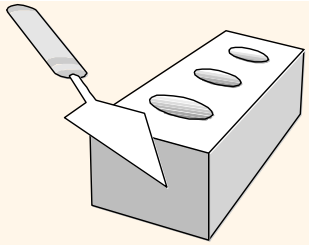
- 1st diagram translated:
  Workers(S,N,L,D,C)
  Departments(D,M,B)
  (Lots associated with workers.)
- *Suppos*
  assigne
- Redur
  Worke
  Worke
  Depar
- Can further fine-tune this:
  Workers2(S,N,D,C)
  Departments(D,M,B,L)

Before:

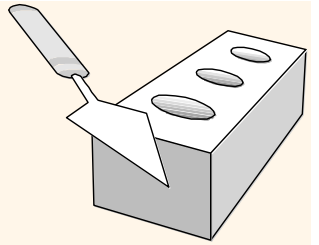


Note:
In many cases the relational translation of an ER design will take you right to 3NF (even BCNF)…!
- Entity key→attributes for entity sets.
- Relationship key→ attributes for relationship sets.

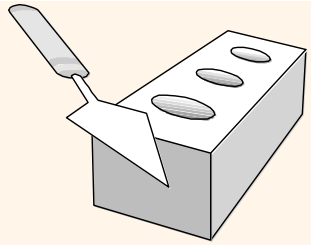
(But problems could arise with FDs within attributes.)



Questions…?

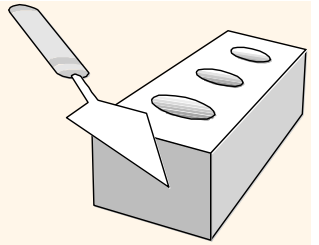


Addendum (3NF Schema Synthesis)



Minimal Cover for a Set of FDs

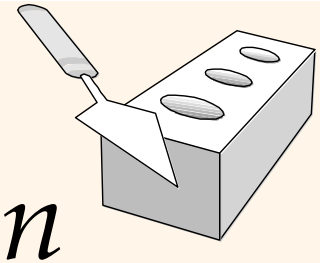
- *Minimal cover* **G** for a set of FDs F:
  - Closure of G = closure of F, i.e., $G^+ = F^+$.
  - Right hand side (RHS) of each FD in G is a ***single*** attribute.
  - If we change G by deleting any FD *or* deleting attributes from the LHS of any FD in G, the closure would change.
- Intuitively: Every FD in G is needed, with G being as "*as small as possible*" to have the same closure as F.
- *E.g.*, A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover (trust me for now):
  - A→**B**, ACD→**E**, EF→**G** and EF→**H**
- *M.C.* → lossless-join, dep. pres. 3NF decomposition!



Computing the Minimal Cover

1. Put the set of given FDs in a Standard Form.
   - This turns F into a set G of equivalent FDs with a *single attribute* on the right-hand side of each FD.
2. Minimize the left-hand side of each FD in G.
   - For each FD in G, check each LHS attribute to see if it can be *deleted* without breaking the equivalence G+ = F+.
3. Delete redundant FDs.
   - For any FDs that remain, check to see if this FD can be *deleted* without breaking the equivalence G+ = F+.

And voila – you now have a minimal cover for F…!



Obtaining that 3NF Decomposition

I. Compute F's minimal cover G (which is also sometimes denoted as F-).

II. Search for dependencies in F- that have the same attribute set on their left hand side, α:

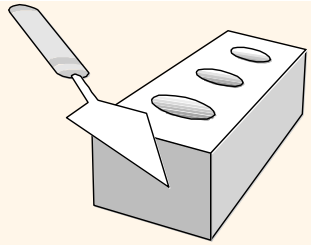
   a. α→Y1, α→Y2, ... α→Yk

   b. Construct one relation as (α,Y1, Y2, ...Yk)

   c. Repeat this process for all of the FDs' α's

   d. If none of the relations from above contains a candidate key for the original relation R, add one more relation with (just) the attributes of a candidate key for R.

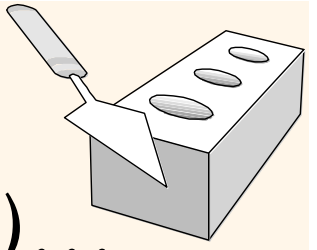
   (Q: Why…?)



Testing Your Understanding…

- Now that you now how to compute BCNF and 3NF decompositions, try it on our earlier examples!
  - *≠2NF*: Supplies(sno, sname, saddr, pno, pname, pcolor)
    with: sno→sname, sno→saddr, pno→pname, pno→pcolor
  - *≠3NF*: Workers(eno, ename, esal, dno, dname, dfloor)
    with: eno→ename, eno,ename→esal, eno→dno, dno→ dname,dfloor
  - *≠BCNF*: Supply2(sno, sname, pno)
    with: sno→sname, sname→sno

Note: I changed the *≠3NF* example's FDs to be equivalent to our earlier FDs but messier to better illustrate the nature of the minimal cover algorithm's operation.



Testing Your Understanding (cont.)…

- *≠3NF:*

  Workers(eno, ename, esal, dno, dname, dfloor)

  with: eno→ename, eno,ename→esal, eno→dno, dno→ dname,dfloor

  3NF M.C. step 1:

  eno→ename

  eno,ename→esal

  eno→dno

  dno→dname

  dno→dfloor

  3NF M.C. step 2:

  eno,ename→esal ➔ eno→esal

  3NF step II:

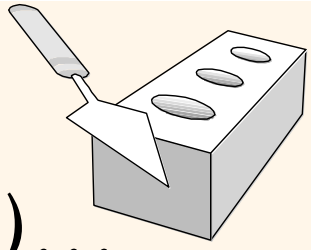
  Emp(eno, ename, esal, dno)

  Dept(dno, dname, dfloor)

Q1: What is the attribute closure of eno – and what does that mean...?

We got lucky!
(No lossy join!)

Q2: What if the Emp-Dept relationship had been M:N?



Testing Your Understanding (cont.)…

- *≠3NF:*

  Workers(eno, ename, esal, dno, dname, dfloor)

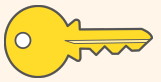
  with: eno→ename, eno,ename→esal, eno→dno, dno→ dname,dfloor

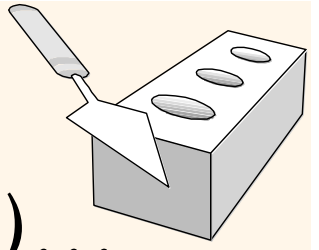
eno→ename
eno,ename→esal → eno→esal
eno→dno
dno→dname
dno→dfloor

{eno}
{eno, ename}
{eno, ename, esal}
{eno, ename, esal, dno}
{eno, ename, esal, dno, dname}
{eno, ename, esal, dno, dname, dfloor}

Q1: What is the attribute closure of eno – and what does that mean...?

→ That's everything in Workers! (***Thus***…?)





Testing Your Understanding (cont.)…

- *≠3NF (modified to M:N)*:

  Workers(eno, ename, esal, dno, dname, dfloor)

  with: eno→ename, eno,ename→esal, ~~eno→dno~~, dno→ dname,dfloor

  eno→ename
  eno,ename→esal ➔ eno→esal
  ~~eno→dno~~
  dno→dname
  dno→dfloor

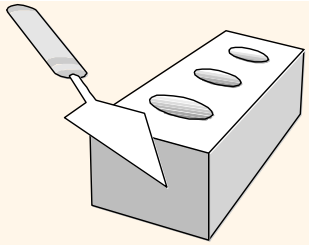
  Q2: What if the Emp-Dept relationship had been M:N?

  Without Works we'd have a **lossy join…!**

  Emp(eno, ename, esal, ~~dno~~)

  Dept(dno, dname, dfloor)

  Works(eno, dno)



Questions…?