SCHEMA REFINEMENT AND NORMAL FORMS

Modified from

Raghu Ramakrishnan and Johannes Gehrke, Database Management Systems, Third Edition, 2003 &

Ramez Elmasri and Shamkant B. Navathe, Fundamentals of Database Systems, Sixth Edition, Pearson Education, 2010.

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)

Functional dependencies (FDs) are used to specify formal measures of the "goodness" of relational designs

FDs and keys are used to define **normal forms** for relations

FDs are **constraints** that are derived from the meaning and interrelationships of the data attributes

A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y

FUNCTIONAL DEPENDENCIES (FDS)

A functional dependency $X \longrightarrow Y$ holds over relation R if, for every allowable instance *r* of R:

- $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)

An FD is a statement about *all* allowable relations.

- Must be identified based on semantics of application.

K is a candidate key for R means that $K \rightarrow R$

- However, $K \rightarrow R$ does not require K to be *minimal*!

PROJECTION π คือ การเลือก colum

$$\pi_{\text{list of attributes separated by comma}}\ (RELATION)$$

Deletes unwanted attributes (not in *projection list*.)

Schema of the result contains only the fields in the projection list, with the same names.

Projection operator eliminates duplicates.

Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it.

S2

sid	sname	rating	age
28	ลิซ่า	10	23
31	เจนนี่	8	24
44	วี	5	24
58	โรเซ่	10	23

$\pi_{sname,\ rating}\ (\mathrm{s2})$

sname	rating
ลิซ่า	10
เจนนี่	8
วี	5
โรเซ่	10

$\pi_{age}\ (\mathrm{s2})$

age
23
24

FUNCTIONAL DEPENDENCIES (FDS)

A functional dependency $X \longrightarrow Y$ holds over relation R if, for every allowable instance *r* of R:

- $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)

An FD is a statement about *all* allowable relations.

- Must be identified based on semantics of application.

K is a candidate key for R means that $K \rightarrow R$

- However, $K \rightarrow R$ does not require K to be *minimal*!

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 $Y \subseteq X$, then $X \rightarrow Y$
- ***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 (CONT.)

Couple of additional rules (that follow from AA):

- ***Union:*** If X → Y and X → Z, then X → YZ
- ***Decomposition:*** If X → YZ, then X → Y and X → Z

Example:

CONTRACTS(*<u>contractid</u>,supplierid,projectid,deptid,partid,qty,value*)

- c is the key: c → csjdpqv
- 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

REASONING ABOUT FDS (CONT.)

Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # of attributes!)

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^+
- Check if Y is in X^+

Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E\}$ imply $A \rightarrow E$?

- i.e., is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

FUNCTIONAL DEPENDENCIES

a	b	c	d
X	α	1	U
X	α	1	V
X	β	5	W
Y	β	3	W
Y	β	3	V

$a\ b \rightarrow c$

" ab determines c"

two tuples with the same values for a and b
will also have the same value for c

HOW TO USE FUNCTIONAL DEPENDENCIES TO DETERMINE KEYS

- ✓ An attribute is **PRIME** if it is part of **any candidate key.**

- ✓ An attribute is **NON-PRIME** if it is **not** part of **any candidate key.**

HOW TO USE FUNCTIONAL DEPENDENCIES TO DETERMINE KEYS

Example 1:

R(a,b,c)

F = { $a \rightarrow b$, $b \rightarrow c$}

L : appears only on the **left** side of FDs
R: appears only on the **right** side of FDS
M: appears on **both left and right** sides.

Must be part of the key

L	M	R
a	b	c

May or may not be part of the key

Never be part of any key

Keys: a

Prime attribute: a

Non-prime attribute: b,c

HOW TO USE FUNCTIONAL DEPENDENCIES TO DETERMINE KEYS

Example 2:

R(a,b,c,d)

F = { $ab \rightarrow c$, $c \rightarrow b$, $c \rightarrow d$}

Must be part of the key

L	M	R
a	bc	d

May or may not be part of the key

Never be part of any key

Keys: ab or ac

Prime attribute: a,b,c

Non-prime attribute: d

HOW TO USE FUNCTIONAL DEPENDENCIES TO DETERMINE KEYS

Example 3:

R(a,b,c)

$F = \{ a \rightarrow b, b \rightarrow c, c \rightarrow a\}$

L	M	R
	abc	

Keys: a or b or c

Prime attribute: a,b,c

Non-prime attribute: -

HOW TO USE FUNCTIONAL DEPENDENCIES TO DETERMINE KEYS

Example 4:

R(a,b,c,d,e)

$F = \{ a \rightarrow d, d \rightarrow b, b \rightarrow c, e \rightarrow b\}$

L	M	R
ae	bd	c

Keys: ae

Prime attribute: a,e

Non-prime attribute: b,c,d

FUNCTIONAL DEPENDENCIES

EMPLOYEE (1NF)				
emp_no	**name**	**dept_no**	**dept_name**	**skills**
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

name, dept_no, and dept_name are functionally dependent on emp_no **(emp_no $\rightarrow$ name, dept_no, dept_name)**

Skills is not functionally dependent on emp_no since it is not unique to each emp_no.

EXAMPLE

MOVIES

title	*year*	*length*	*filmType*	*studioName*	*starName*
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez
Wayne's World	1992	95	color	Paramount	Dana Carvey
Wayne's World	1992	95	color	Paramount	Mike Myers

Can assert FDs:

title year → length

title year → filmType

title year → studioName

But not:

title year → starName

EXAMPLE: CONSTRAINTS ON ENTITY SET

Consider relation obtained from HOURLY_EMPS:

- HOURLY_EMPS (<u>*ssn*</u>, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)

<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's name. (e.g., HOURLY_EMPS for snlrwh)

Some FDs on HOURLY_EMPS:

- *ssn* is the key: s $\rightarrow$ snlrwh
- *rating* determines *hrly_wages*: r $\rightarrow$ w

EXAMPLE (CONT.)

HOURLY_EMPS relation

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

Assume no Functional Dependencies

Any instance is legal here (no constraints)

No redundancy

EXAMPLE: CONSTRAINTS ON ENTITY SET

HOURLY_EMPS relation

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

If we add some FDs on HOURLY_EMPS:

- *ssn* is the key: s $\rightarrow$ snlrwh
  - Values of ssn have to be unique in the relation
- *rating* determines *hrly_wages*: r $\rightarrow$ w
  - For two rows that have same value of r, the rows also have same value for w
  - Above relation stores r and w values “redundantly”

EXAMPLE (CONT.)

Problems due to r $\rightarrow$ w :

- *Update anomaly*: Can we change w in just the 1st tuple of snlrwh?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

HOURLY_EMPS

s	n	l	r	w	h
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Will 2 smaller tables be better?

EXAMPLE (CONT.)

Will 2 smaller tables be better? Yes

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

Solution to avoid redundancy: **Decomposition to smaller tables**

But:

what criteria should the new 'smaller' tables satisfy so that you can stop decomposition? What is a good design?

---- Normal Forms

NULL VALUES IN TUPLES

Relations should be designed such that their tuples will have as few NULL values as possible

Attributes that are NULL frequently could be placed in separate relations (with the primary key)

NULL values can address insertion (but not all) and deletion anomalies.

HOURLY_EMPS (*<u>ssn</u>*, *name*, *lot*, *rating*, *hrly_wages*, hrs_*worked*)

- Can insert an employee tuple with NULL value in *hrly_wages*
- Can't store NULL values in *ssn*

What happens if the last tuple with a given rating would be deleted?

NORMALIZATION

Normalization: The process of **decomposing** unsatisfactory "bad" relations by breaking up their attributes into smaller relations

Normal form: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

2NF, 3NF, BCNF based on keys and FDs of a relation schema

PROPERTIES OF DECOMPOSITIONS

There are two important properties of decompositions:

- **Lossless-Join decomposition** Must have
  - The **lossless-join** (also called *nonloss-* or *nonadditive*-join) property ensures that any instance of the original relation can be identified from corresponding instances in the smaller relations
- **Dependency-Preserving Decomposition**
  - The **dependency preservation** property ensures that a constraint on the original relation can be maintained by simply enforcing some constraint on each of the smaller relations.
  - Intuition: If R is decomposed into R_1, R_2, and we enforce the FDs that hold individually on R_1, and on R_2, then all FDs that were given to hold on R must also hold

A

l-name	f-name	age
Bouvier	Selma	40
Bouvier	Patty	40
Smith	Maggie	2

B

l-name	f-name	id
Bouvier	Selma	1232
Smith	Selma	4423

axb

common attributes --> intersec สองตาราง

Both the *l-name* and the *f-name* match, so select.

Only the *f-names* match, so don't select.

Only the *l-names* match, so don't select.

l-name	f-name	age	l-name	f-name	id
Bouvier	Selma	40	Bouvier	Selma	1232
Bouvier	Selma	40	Smith	Selma	4423
Bouvier	Patty	40	Bouvier	Selma	1232
Bouvier	Patty	40	Smith	Selma	4423
Smith	Maggie	2	Bouvier	Selma	1232
Smith	Maggie	2	Smith	Selma	4423

We remove duplicate attributes…

l-name	f-name	age	l-name	f-name	id
Bouvier	Selma	40	Bouvier	Selma	1232

The natural join of *A* and *B*

$a \bowtie b =$

l-name	f-name	age	id
Bouvier	Selma	40	1232

DECOMPOSITION

1. Decomposing the schema

R = (bname, bcity, assets, cname, lno, amt)

R = R1 U R2

R1 = (bname, bcity, assets, cname)

R2 = (cname, lno, amt)

2. Decomposing the instance

bname	bcity	assets	cname	lno	amt
Downtown	Bkln	9M	Jones	L-17	1000
Downtown	Bkln	9M	Johnson	L-23	2000
Mianus	Horse	1.7M	Jones	L-93	500
Downtown	Bkln	9M	Hayes	L-17	1000

R1

bname	bcity	assets	cname
Downtown	Bkln	9M	Jones
Downtown	Bkln	9M	Johnson
Mianus	Horse	1.7M	Jones
Downtown	Bkln	9M	Hayes

R2

cname	lno	amt
Jones	L-17	1000
Johnson	L-23	2000
Jones	L-93	500
Hayes	L-17	1000

GOAL #1: LOSSLESS JOINS

A bad decomposition:

bname	bcity	assets	cname	lno	amt
Downtown	Bkln	9M	Jones	L-17	1000
Downtown	Bkln	9M	Johnson	L-23	2000
Mianus	Horse	1.7M	Jones	L-93	500
Downtown	Bkln	9M	Hayes	L-17	1000

bname	bcity	assets	cname
Downtown	Bkln	9M	Jones
Downtown	Bkln	9M	Johnson
Mianus	Horse	1.7M	Jones
Downtown	Bkln	9M	Hayes

$\bowtie$

cname	lno	amt
Jones	L-17	1000
Johnson	L-23	2000
Jones	L-93	500
Hayes	L-17	1000

=

bname	bcity	assets	cname	lno	amt
Downtown	Bkln	9M	Jones	L-17	1000
Downtown	Bkln	9M	Jones	L-93	500
Downtown	Bkln	9M	Johnson	L-23	2000
Mianus	Horse	1.7M	Jones	L-17	1000
Mianus	Horse	1.7M	Jones	L-93	500
Downtown	Bkln	9M	Hayes	L-17	1000

Problem: join adds meaningless tuples "lossy join": by adding noise, have lost meaningful information

GOAL #1: LOSSLESS JOINS

Is the following decomposition lossless or lossy?

bname	assets	cname	lno
Downtown	9M	Jones	L-17
Downtown	9M	Johnson	L-23
Mianus	1.7M	Jones	L-93
Downtown	9M	Hayes	L-17

lno	bcity	amt
L-17	Bkln	1000
L-23	Bkln	2000
L-93	Horse	500

Ans: Lossless: $r = r1 \bowtie r2$, it has same 4 tuples as original

R1 and R2 share the lno which is the key to R2.

LOSSLESS-JOIN DECOMPOSITION

A decomposition of R : $R = R1 \cup R2$

Is lossless iff

$R1 \cap R2 \rightarrow R1$, or

$R1 \cap R2 \rightarrow R2$

(i.e., intersecting attributes must be a superkey for one of the resulting smaller relations)

In the previous example, lno is the common attribute and lno is the key to second relation R2

LOSSLESS-JOIN DECOMPOSITION

Suppose that we decompose the schema R = (a, b, c, d, e) into

R1 = (a, b, c)

R2 = (a, d, e)

Show that this decomposition is a lossless-join decomposition if the following set F of functional dependencies holds:

$a \rightarrow bc$

$cd \rightarrow e$

$b \rightarrow d$

$e \rightarrow a$

A decomposition {R1, R2} is a lossless-join decomposition if

$R1 \cap R2 \rightarrow R1$ **or** $R1 \cap R2 \rightarrow R2$

Since $R1 \cap R2 =$ **a** and **a** is a candidate key of R1, therefore, $R1 \cap R2 \rightarrow R1$

ANOTHER EXAMPLE OF LOSSY-JOIN DECOMPOSITION

Lossy-join decompositions result in information loss.

Example: Decomposition of

ไม่มี common attribute --> R1 intersec R2 = { }

$R(\mathbf{a}, \mathbf{b})$

a	b
α	1
α	2
β	1

r

$R_1(\mathbf{a})$

a
α
β

$\pi_a(r)$

$R_2(\mathbf{b})$

b
1
2

$\pi_b(r)$

$r1 \bowtie r2$

a	b
α	1
α	2
β	1
β	2

EXAMPLE: *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

DEPENDENCY PRESERVATION: EXAMPLE

Take R=R(city, street_no, zipcode) with FDs:

- city,street_no → zipcode
- zipcode → city

Decompose to

- R1(street_no,zipcode)
- R2(city,zipcode)

Claim: This is a lossless-join decomposition

Is it dependency preserving? No

FIRST NORMAL FORM

Disallows composite attributes, multivalued attributes, and nested relations; attributes whose values for an individual tuple are non-atomic

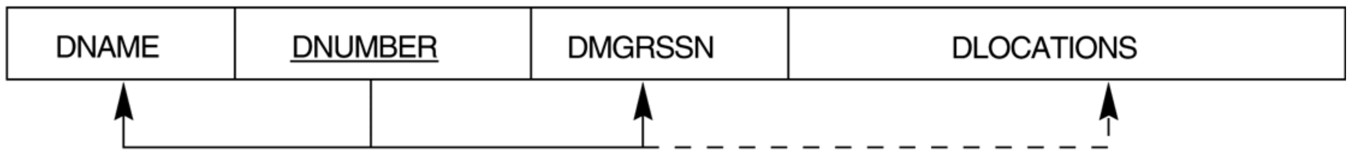
The only attributes values permitted by 1NF are single atomic (or indivisible) values

NORMALIZATION INTO 1NF

(A) A RELATION SCHEMA THAT IS NOT IN 1NF
(B) EXAMPLE STATE OF RELATION DEPARTMENT
(C) 1NF VERSION OF SAME RELATION WITH REDUNDANCY

(a)

DEPARTMENT



(b)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

Not 1NF

(c)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

OTHER POSSIBLE SOLUTIONS

Remove **dlocations** and place in a separate relation DEPT_LOCATIONS (dnumber, dlocation)

Expand the key of the relation to {dnumber, dlocation}

If a maximum number of values is known for the attribute – replace **dlocation** by **dlocation1**, **dlocation2**, and **dlocation3**

- There are disadvantages: introducing null values, and querying on the attribute becomes more difficult

1NF

EMPLOYEE (unnormalized)

emp_no	**name**	**dept_no**	**dept_name**	**skills**
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

EMPLOYEE (1NF)

emp_no	**name**	**dept_no**	**dept_name**	**skills**
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

NORMALIZATION INTO 1NF.

(A) A RELATION SCHEMA THAT IS NOT IN 1NF.

(B) EXAMPLE STATE OF RELATION EMP_PROJ.

(C) 2NF VERSION WITHOUT REDUNDANCY.

(a) **EMP_PROJ**

SSN	ENAME	PROJS	
		PNUMBER	HOURS

(b) **EMP_PROJ**

SSN	ENAME	PNUMBER	HOURS
123456789	Smith,John B.	1	32.5
		2	7.5
666884444	Narayan,Ramesh K.	3	40.0
453453453	English,Joyce A.	1	20.0
		2	20.0
333445555	Wong,Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya,Alicia J.	30	30.0
		10	10.0
987987987	Jabbar,Ahmad V.	10	35.0
		30	5.0
987654321	Wallace,Jennifer S.	30	20.0
		20	15.0
888665555	Borg,James E.	20	null

(c) **EMP_PROJ1**

<u>SSN</u>	ENAME

EMP_PROJ2

<u>SSN</u>	<u>PNUMBER</u>	HOURS

SECOND NORMAL FORM

A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on the ~~primary~~ key

Candidate

Definitions:

Prime attribute - attribute that is member of the primary key

Full functional dependency - an FD $Y \rightarrow Z$ where removal of any attribute from Y means the FD does not hold any more

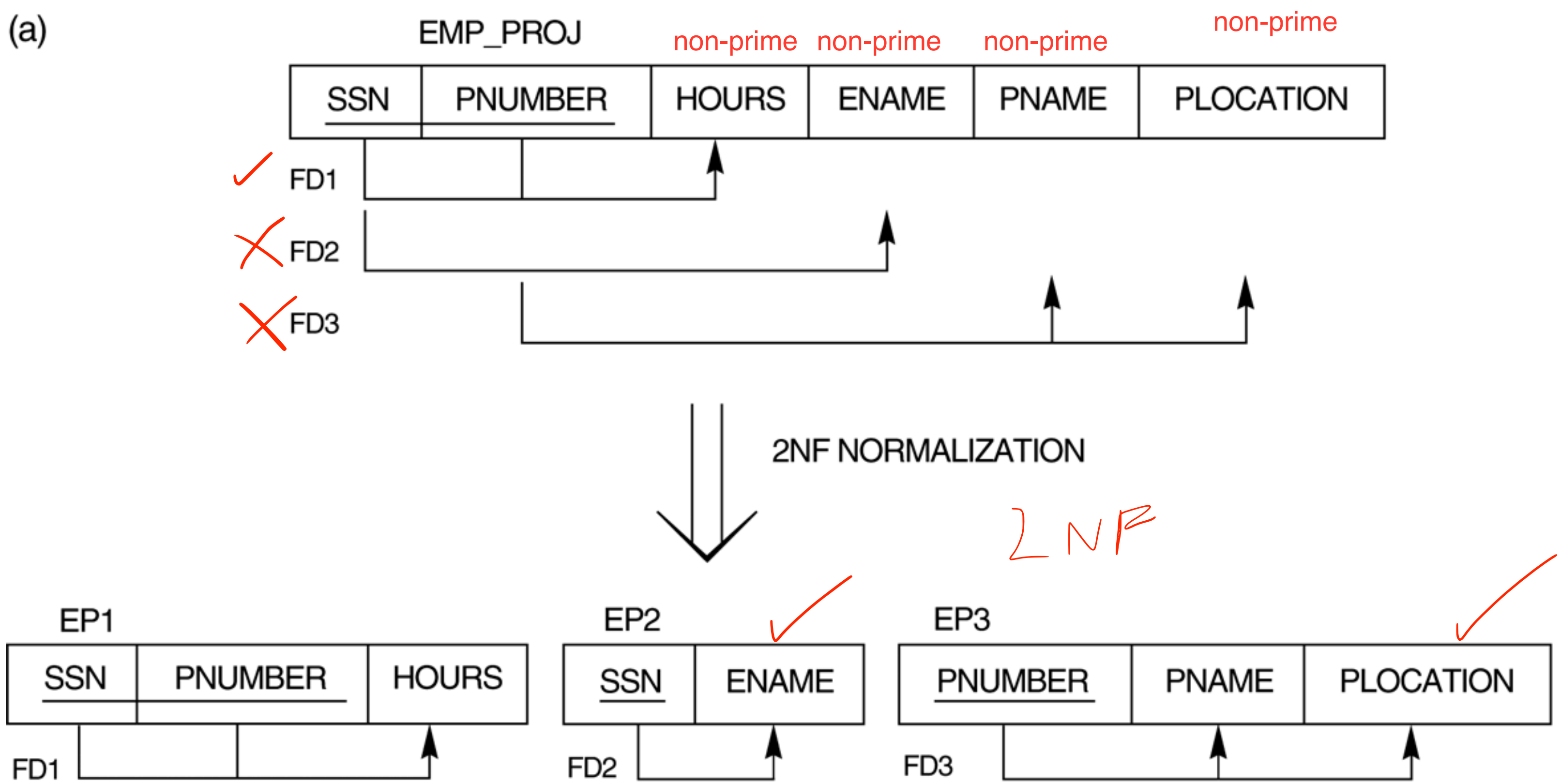
Examples:

- {ssn, pnumber} → hours is a full FD since neither ssn → hours nor pnumber → hours hold

dependency not dependency

- {ssn, pnumber} → ename is not a full FD (it is called a partial dependency) since ssn → ename also holds

NORMALIZING EMP_PROJ INTO 2NF RELATIONS



2NF

emp_no → name, dept_no, dept_name

EMPLOYEE (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

EMPLOYEE (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

SKILLS (2NF)

emp_no	skills
1	C
1	Perl
1	Java
2	Linux
2	Mac
3	DB2
3	Oracle
3	Java

THIRD NORMAL FORM – 3NF

A relation schema R is in **third normal form (3NF)** if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key

<u>Definition:</u>

Transitive functional dependency - a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$

<u>Examples:</u>

- ssn $\rightarrow$ dmgrssn is a *transitive* FD since

  ssn $\rightarrow$ dnumber and dnumber $\rightarrow$ dmgrssn hold

- ssn $\rightarrow$ ename is *non-transitive* since there is no set of attributes X where ssn $\rightarrow$ X and X $\rightarrow$ ename

THIRD NORMAL FORM – 3NF (CONT.)

NOTE:

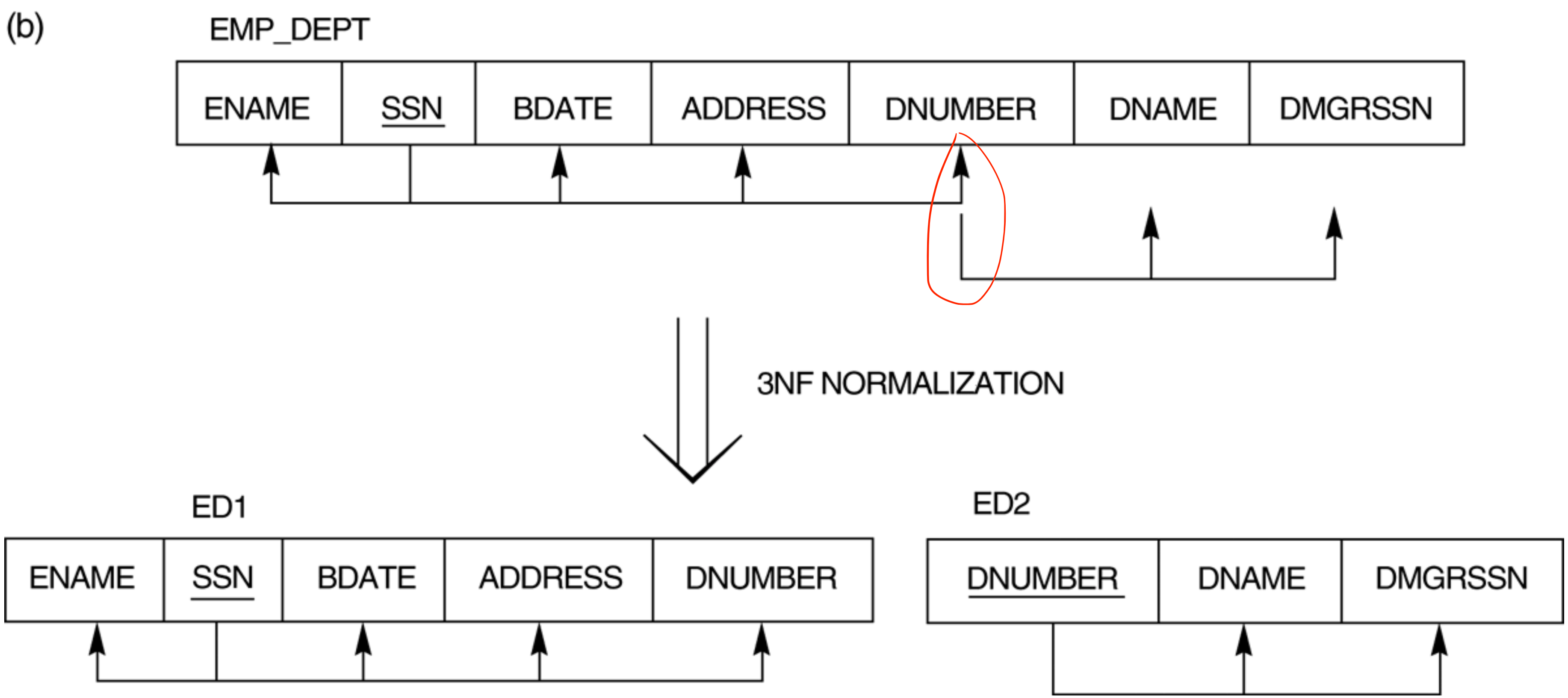
In X $\rightarrow$ Y and Y $\rightarrow$ Z, with X as the primary key, we consider this a problem only if Y is <u>not</u> a candidate key.

When Y is a candidate key, there is no problem with the transitive dependency .

E.g., Consider EMP (<u>ssn</u>, emp#, salary).

Here, ssn $\rightarrow$ emp# $\rightarrow$ salary and emp# is a candidate key.

NORMALIZING EMP_DEPT INTO 3NF RELATIONS.



THIRD NORMAL FORM – 3NF (CONT.)

The previous definition considers the primary key only

The following more general definitions take into account relations with multiple candidate keys

A relation schema R is in second normal form (2NF) if every non-prime attribute A in R is fully functionally dependent on every key of R

THIRD NORMAL FORM – 3NF (CONT.)

Definition:

Superkey of relation schema R - a set of attributes S of R that contains a key of R

A relation schema R is in **third normal form (3NF)** if whenever a FD $X \rightarrow A$ holds in R, then either:

(a) X is a superkey of R, or

(b) A is a prime attribute of R

NOTE: Boyce-Codd normal form disallows condition (b) above

1. L | M | R

ABC D | ABCDEF

A^+ ✓

BC^+ ✓

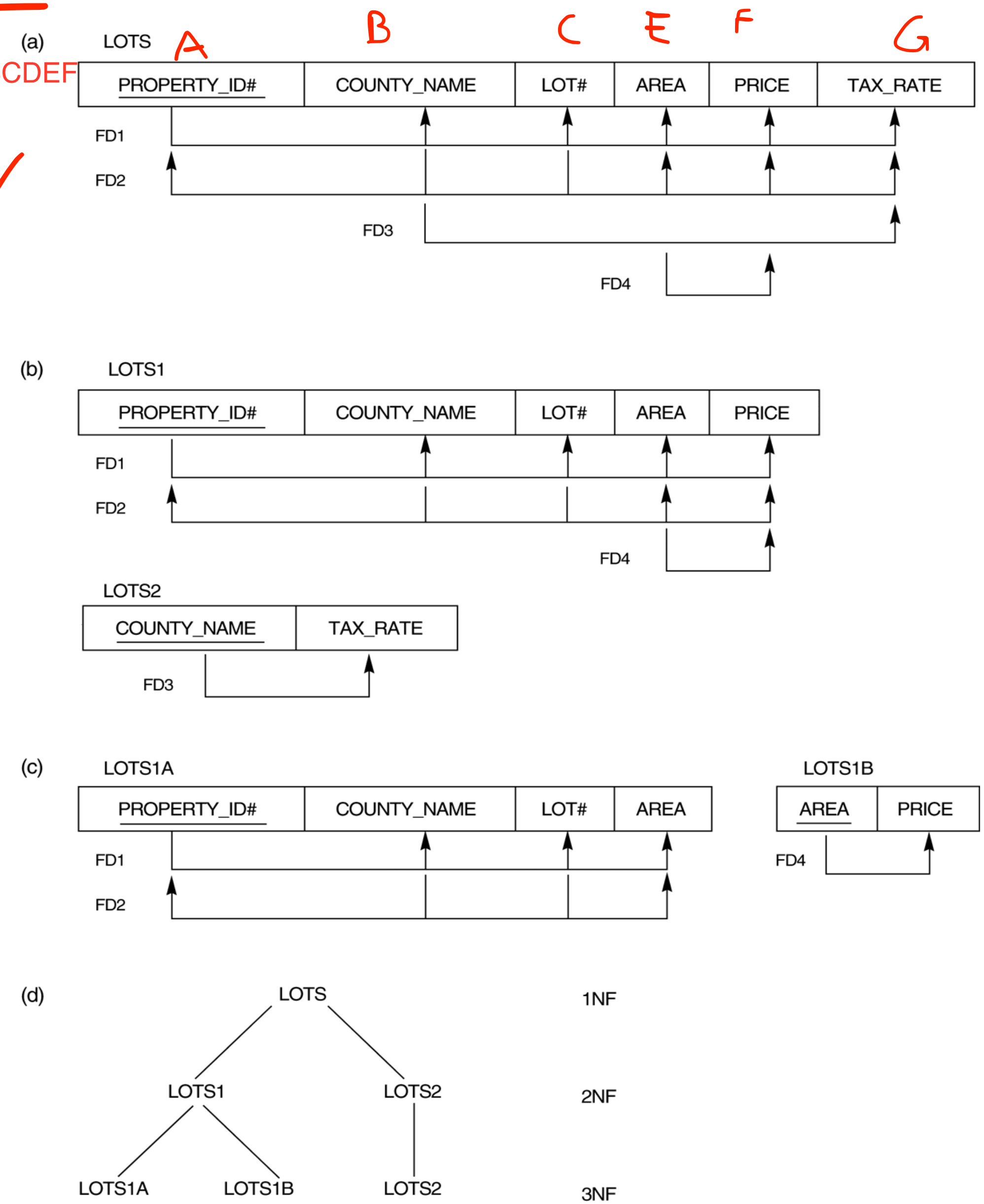
NORMALIZATION INTO 2NF AND 3NF.

(A) THE LOTS RELATION WITH ITS FUNCTIONAL DEPENDENCIES FD1 THOUGH FD4.

(B) DECOMPOSING INTO THE 2NF RELATIONS LOTS1 AND LOTS2.

(C) DECOMPOSING LOTS1 INTO THE 3NF RELATIONS LOTS1A AND LOTS1B.

(D) SUMMARY OF THE PROGRESSIVE NORMALIZATION OF LOTS.



BCNF (BOYCE-CODD NORMAL FORM)

A relation schema R is in **Boyce-Codd Normal Form** (**BCNF**) if whenever an FD $X \rightarrow A$ holds in R, then X is a superkey of R

Each normal form is strictly stronger than the previous one

- Every 2NF relation is in 1NF
- Every 3NF relation is in 2NF
- Every BCNF relation is in 3NF

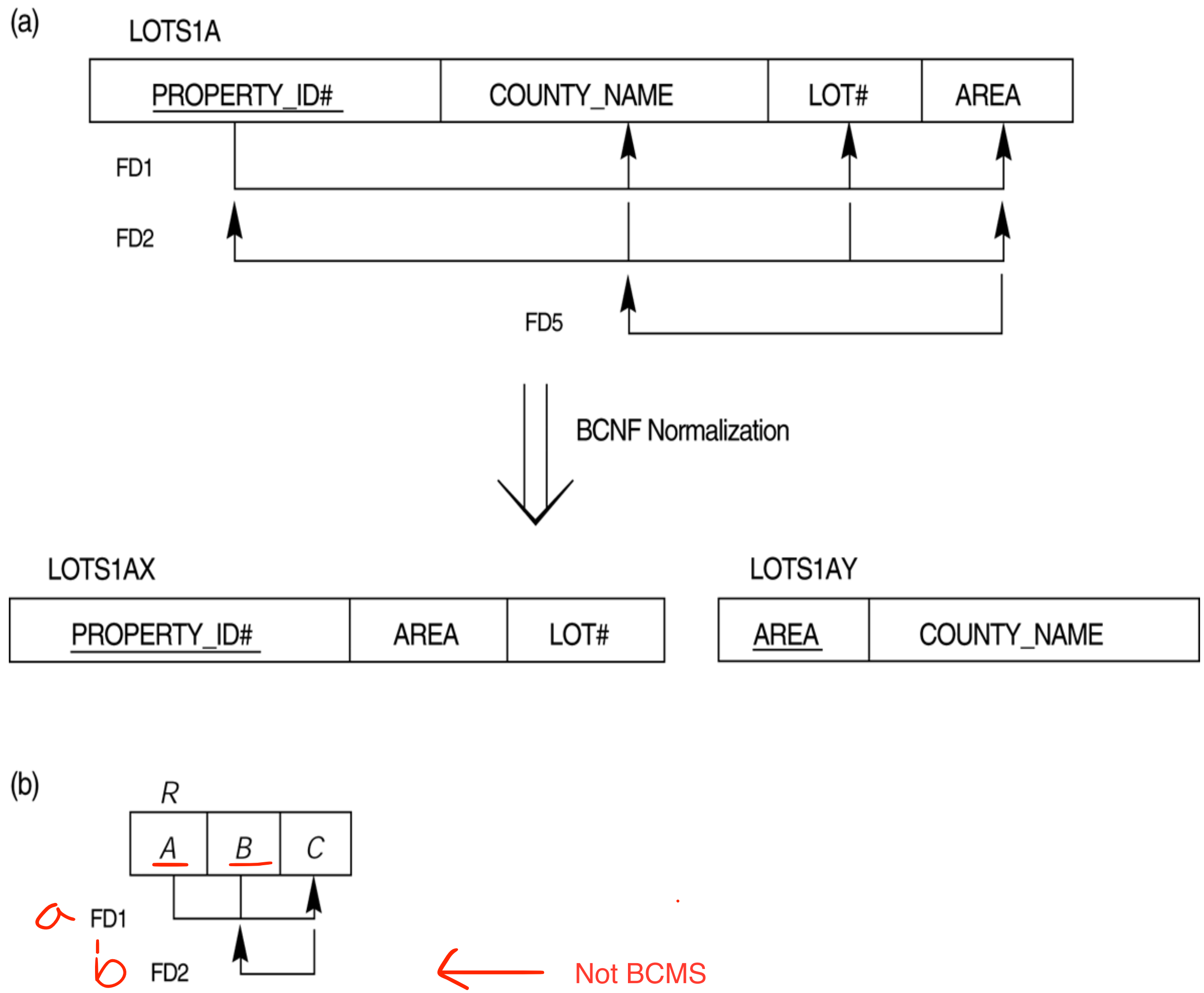
There exist relations that are in 3NF but not in BCNF

The goal is to have each relation in BCNF (or 3NF)

BOYCE-CODD NORMAL FORM.

(A) BCNF NORMALIZATION OF LOTS1A WITH THE FUNCTIONAL DEPENDENCY FD2 BEING LOST IN THE DECOMPOSITION.

(B) A SCHEMATIC RELATION WITH FDS; IT IS IN 3NF, BUT NOT IN BCNF.



A RELATION TEACH THAT IS IN 3NF BUT NOT IN BCNF

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

ACHIEVING THE BCNF BY DECOMPOSITION

Two FDs exist in the relation TEACH:

fd1: { student, course} $\rightarrow$ instructor

fd2: instructor $\rightarrow$ course

{student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in slide 47 (b). So this relation is in 3NF but not in BCNF

A relation NOT in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

ACHIEVING THE BCNF BY DECOMPOSITION (CONT.)

Three possible decompositions for relation TEACH

- {student, instructor} and {student, course}
- {course, instructor } and {course, student}
- {instructor, course } and {instructor, student}

All three decompositions will lose fd1. We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additive property after decomposition.

Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additive property).

REFINING AN ER DIAGRAM

1[st] diagram translated:
WORKERS(i,n,l,d,s)
DEPARTMENTS(d,m,b)

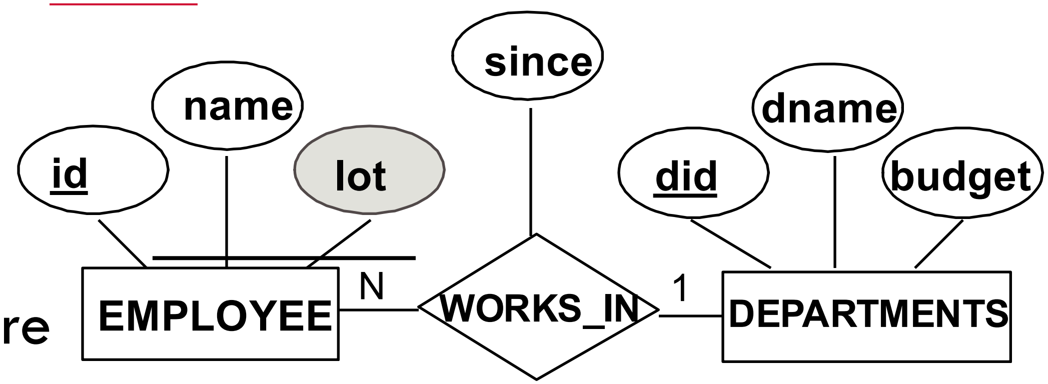
- Lots associated with workers.

Suppose all workers in a dept are assigned the same lot: d ⟶ l

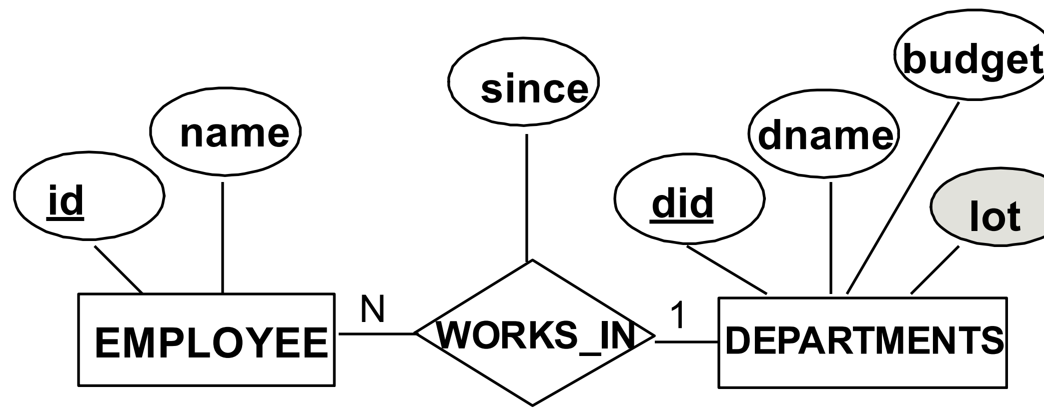
Redundancy; fixed by:
WORKERS2(i,n,d,s)
DEPT_LOTS(d,l)

Can fine-tune this:
WORKERS2(i,n,d,s)
DEPARTMENTS (d,m,b,l)

Before:



After:



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.

A NOTE ON DENORMALIZATION

Denormalization is said to be necessary to improve performance

In practice, denormalization will speed up some queries, and drag down others

Proceed with caution

NORMALIZATION IS GOOD… OR IS IT?

In some cases, we might not mind redundancy, if the data isn't directly updated:

- Reports (people like to see breakdowns by semester, department, course, etc.)
- Warehouses (archived copies of data for doing complex analysis)
- Data sharing (sometimes we may export data into object-oriented or hierarchical formats)