Unit 3 (Relational Database Design)

Original Content:
Ramez Elmasri and Shamkant B. Navathe

Dr. Shobha G
Professor, Department of CSE
RV College of Engineering, Bengaluru - 59

Contents

Functional Dependencies

- 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
- FD is a constraint between two sets of 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

- X Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y
- For any two tuples t1 and t2 in any relation instance r(R):

  If t1[X]=t2[X],

  then t1[Y]=t2[Y]
- X Y in R specifies a *constraint* on all relation instances r(R)
- Written as X Y; can be displayed graphically on a relation schema as in Figures. (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Example

TUPLE #	A	B	C
1	10	b1	c1
2	10	b2	c2
3	11	b4	c1
4	12	b3	c4
5	13	b1	c1
6	14	b3	c4

Does A → B?

No.

$t_1[A] = t_2[A]$, but $t_1[B] \neq t_2[B]$

Does B□C

TUPLE #	A	B	C
1	10	b1	c1
2	10	b2	c2
3	11	b4	c1
4	12	b3	c4
5	13	b1	c1
6	14	b3	c4

Examples of FD constraints

TUPLE #	A	B	C
1	10	b1	c1
2	10	b2	c2
3	11	b4	c1
4	12	b3	c4
5	13	b1	c1
6	14	b3	c4

Does $B \rightarrow C$?

Yes!

Look at tuples t_1 and t_5, and tuples t_4 and t_6

TUPLE #	A	B	C
1	10	b1	c1
2	10	b2	c2
3	11	b4	c1
4	12	b3	c4
5	13	b1	c1
6	14	b3	c4

Does $C \rightarrow B$?

No.

$t_1[C] = t_3[C]$, but $t_1[B] \neq t_3[B]$

Examples of FD constraints

- social security number determines employee name

  SSN -> ENAME
- project number determines project name and location

  PNUMBER -> {PNAME, PLOCATION}
- employee ssn and project number determines the hours per week that the employee works on the project

  {SSN, PNUMBER} -> HOURS

Examples of FD constraints

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every relation instance* r(R)
- If K is a key of R, then K functionally determines all attributes in R (since we never have two distinct tuples with t1[K]=t2[K])

FDs must hold for <u>all valid states</u> of a relation, not just current state

- So define FDs carefully!

How do we identify FDs?

- Likely, some FDs will be obvious or identified in initial design of DB

Vehicle(<u>tagno, regstate</u>, owner, make, model, year, gaseconomy, dealership, dealeraddr)

{tagno, regstate} → owner
{tagno, regstate} → {make, model, year}
{make, model, year} → gaseconomy,
dealership → dealeraddr etc…

Inferring FDs

Ex: (SSN, PNUMBER, HOURS, ENAME, PNAME, PLOCATION)

SSN → ENAME,
{SSN, PNUMBER} → HOURS, PNUMBER → PNAME, PNUMBER → PLOCATION

PNUMBER → PNAME,
so {PNUMBER, HOURS} → PNAME

PNUMBER → PNAME and PNUMBER → PLOCATION, so PNUMBER → {PNAME, PLOCATION}

Inference Rules for FDs

- Given a set of FDs F, we can *infer* additional FDs that hold whenever the FDs in F hold

Armstrong's inference rules:

IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y

IR2. (**Augmentation**) If X -> Y, then XZ -> YZ

(Notation: XZ stands for X U Z)

IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a *sound* and *complete* set of inference rules

Inference Rules for FDs

Some **additional inference rules** that are useful:

(**Decomposition**) If X **->** YZ, then X **->** Y and X **->** Z

(**Union**) If X **->** Y and X **->** Z, then X **->** YZ

(**Psuedotransitivity**) If X **->** Y and WY **->** Z, then WX **->** Z

- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Inference Rules for FDs

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F
- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

Closure of X under F (X^+)

- X^+ = set of all attributes dependent on X
- Algorithm
  1. start with $X^+ = X$
  2. for each FD $Y \rightarrow Z$ in F do

     if Y is a subset of X^+ then $X^+ = X^+ \cup Z$

3.Continue this process until no more attributes can be added to X^+

Example

Given a relation Student and a set of functional dependencies F as follows, compute the closure for all LHS.

Student(SID, dept, dept_chair)

F = { SID → {dept, dept_chair}, dept → dept_chair,
{SID, dept} → dept_chair }

$\{SID\}^{+} = \{SID, dept, dept\_chair\}$

$\{dept\}^{+} = \{dept, dept\_chair\}$

$\{SID, dept\}^{+} = \{SID, dept, dept\_chair\}$

If the closure of a LHS includes all attributes, then this LHS is a super key of the relation.

Candidate Keys

- If X^+ contains all attributes in a relation R, and if there does not exist Y in X such that $(X – Y)^+$ = all attributes in R, then X is a **candidate key** for R

- In previous example, $\{SID,dept\}^+$ includes all attributes, SID^+ also includes all attributes.
- SID is a candidate key

Exercise

R(A,B,C,D,G,H)

$F = \{ A \rightarrow B, B \rightarrow C, CD \rightarrow H, BC \rightarrow G\}$

- What is the closure of AC?

Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:

  - every FD in F can be inferred from G, *and*

  - every FD in G can be inferred from F

- Hence, F and G are equivalent if $F^+ = G^+$

Definition: F **covers** G if every FD in G can be inferred from F (i.e., if G^+ *subset-of* F^+)

- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

Example of Equivalent FD sets

F1 = { SID → {dept, dept_chair}, dept → dept_chair,
{SID, dept} → dept_chair }

F2 = { SID → {dept, dept_chair}, dept → dept_chair}

- F1 covers F2
- Only need to check if F2 can also infer F1
  - i.e. Can we generate the set of FDs in F1 using FDs defined in F2?
- Compute $\{SID,dept\}^+$ based on F2
  - Its closure includes all attributes
  - We can conclude that {SID,dept} → dept_chair

Minimal Sets of FDs

- A set of FDs is **minimal** if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for its RHS.

(2) We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

(3) We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X (Y subset-of X) and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs

Algorithm

Given a set of FDs F, find its minimal cover

- Step1: Decompose each FD to get single attribute at RHS
- Step2: For each FD, remove redundant attribute from LHS
- Step3: Remove redundant FDs

Example

Given F= {B → AB, D → A, AB → D}

- Step 1: B → AB is decomposed into B → A, B → B
  (B → B is trivial and is removed)
- Step 2: check if AB → D has redundant LHS. Can it be A → D or B → D?
  Compute AB^+, A^+, B^+ based on **F**

  AB^+ = ABD and B^+ = ABD, so A is extraneous.

- So far, we have F= {B → A, D → A, B → D}

Example (cont.)

- So far, we have F= {B → A, D → A, B → D}
- Step 3: check if there is any redundant FDs. Is B → A redundant?

Compute B^+ based on F- {B → A}

B^+ = BDA, that means we can obtain
B → A from F- {B → A} so B → A is redundant.

Similarly, check the remaining FDs.

Final answer F' = {D → A, B → D}

Exercise

Given a set of FDs F, find its **minimal cover**

- Step1: Decompose each FD to get single attribute at RHS
- Step2: For each FD, remove redundant attribute from LHS
- Step3: Remove redundant FDs

Question: what is the minimal cover of F?

R(A, B, C), F = {A → B, BC → A, AB → AC}

Minimal Set (Cover) of FDs

- There can be more than one minimal cover for a relation
- They won't necessarily have the same number of FDs

Normalization of Relations

- **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

Normalization of Relations

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

- **4NF**
  - based on keys, multi-valued dependencies : MVDs; 5NF based on keys,
  - Join dependencies : JDs (Chapter 11)

- Additional properties may be needed to ensure a good relational design (*lossless join, dependency preservation*; Chapter 11)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms is questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF, BCNF or 4NF)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

Definitions of Keys and Attributes Participating in Keys

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.

- A **Prime attribute** must be a member of *some* candidate key

- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

First Normal Form 1NF

- A relation scheme R is in first normal form (1NF) if the values in *dom* (*A*) are atomic for every attribute A in R.

- **Disallows**
  - composite attributes
  - Set-valued attributes
  - **nested relations**; a cell of an *individual tuple* is a complex relation

First Normal Form 1NF

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Normalization nested relations into 1NF

(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, AliciaJ.	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

Composite attributes

(c)

EMP_PROJ1

<u>Ssn</u>	Ename

EMP_PROJ2

<u>Ssn</u>	<u>Pnumber</u>	Hours

Figure 10.9
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Second Normal Form

- Uses the concepts of **FDs, primary key**

- **Definitions**
  - **Prime attribute:** An attribute that is member of the primary key K
  - **Left-Reduced or Full functional dependency:** a FD Y □ 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
  - {SSN, PNUMBER} □ ENAME is not a full FD (it is called a partial dependency) since SSN □ENAME also holds

Second Normal Form

- **A relation scheme R is in second normal form (2NF) with respect to a set of FDs F if it is in 1NF and every nonprime attribute is fully dependent on every key of R.**

- R can be decomposed into 2NF relations via the process of 2NF normalization

•Example

- Let R=ABCD and F = { AB → C, B → D }. Here AB is a key. C and D are non-prime. C is fully dependent on the entire key AB, however D functionally depends on just *part* of the key (B →D). This is called a *partial dependency*

Second Normal Form

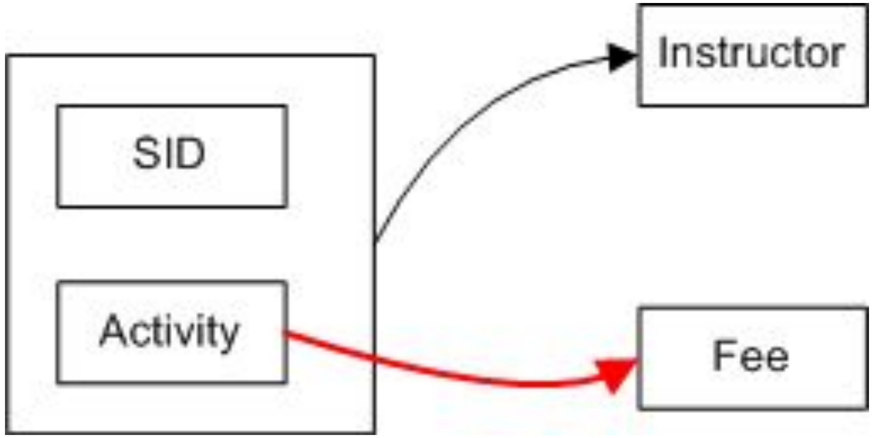
Example
We deduce from the data sample

activity →fee,
sid activity →instructor

SID	Activity	Fee	Instructor
100	Basket Ball	200	Lebron
100	Golf	65	Arnold
200	Golf	65	Jack
300	Golf	65	Lebron

Key: {*sid ,activity*}. Non-key attributes: { Fee, Instructor }

There is a partial dependency
therefore the schema is not 2NF



Normalizing into 2NF and 3NF

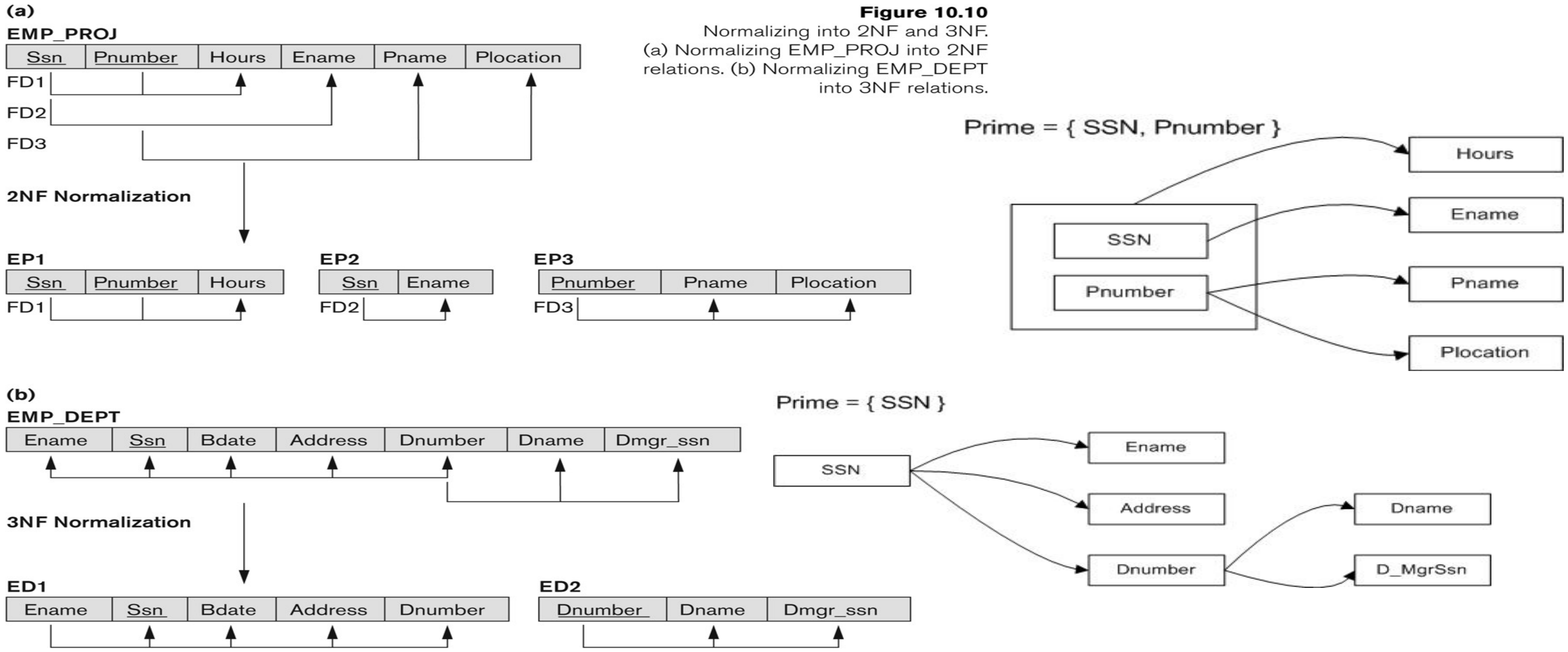


Figure 10.10
Normalizing into 2NF and 3NF.
(a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Normalization into 2NF and 3NF

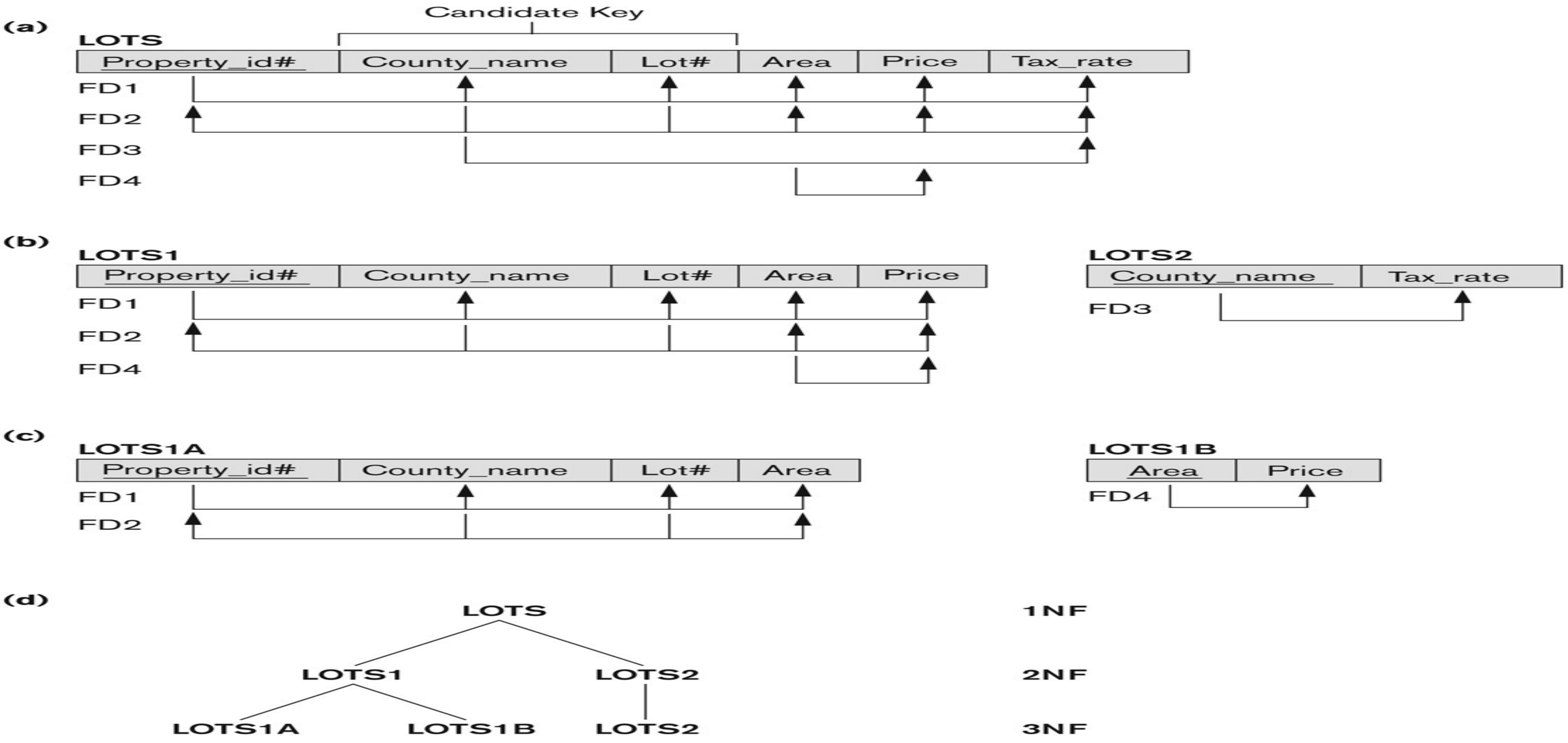


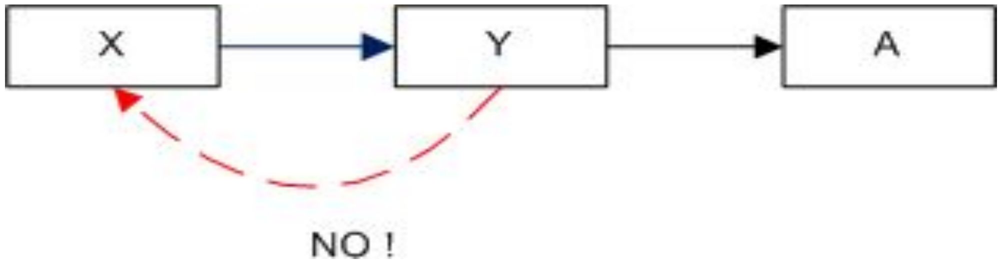
Figure 10.11
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

Third Normal Form

- **Definition**:

  Given a relation scheme R, a subset X of R, an attribute A in R, and a set of FDs F, A is **transitively dependent** upon X in R if there is a subset Y of R with:

  $X \rightarrow Y$, $Y \nrightarrow X$ and $Y \rightarrow A$ under F and $A \notin XY$.



Examples:

Schema (ABCD) and F = {$A \rightarrow B$, $B \rightarrow AC$, $C \rightarrow D$ }
D is transitively dependent on A(and B) via C, however C is not transitively dependent on A via B (B is prime).

Third Normal Form

- 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

- R can be decomposed into 3NF relations via the process of 3NF normalization

- **NOTE**:
  - In X Y and Y Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.
  - When Y is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary).
  - Here, SSN Emp# Salary and Emp# is a candidate key.

Normal Forms Defined Informally

- 1st normal form
  - All attributes depend on **the key**
- 2nd normal form
  - All attributes depend on **the whole key**
- 3rd normal form
  - All attributes depend on **nothing but the key**

General Normal Form Definitions

- The above definitions consider 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

General Normal Form Definitions

- **Example** Consider the schema
- SUPPLIER(sname, saddress, item, iname, price) and FDs

- F= { sname □saddress, item □iname, {sname, item} □price }

1. *sname item* is the primary key, all other attributes are non-prime.
2. Observe that *saddress* depends on part of the key (*sname*).
3. Likewise iname depends on part of the key (item)
4. Therefore SUPPLIER is not in 2NF

General Normal Form Definitions

- 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 A holds in R, then either:
  - (a) X is a superkey of R, or
  - (b) A is a prime attribute of R

General Normal Form Definitions

Example

SID	Building	Fee	Manager
100	Fenn	300	Mr. T
300	ABC	400	Ali
200	Holiday Inn	400	Tyson

Key: { SID }

SID→Building

Building →Fee

Building →Mgr

Fee (and *Manager*) transitively depend on *SID* via the non-prime attribute *Building*. Therefore the relation is not in 3NF.

Let's try it

name	address	beer
Sally	123 Maple	Bud
Sally	123 Maple	Miller

F= { name → address } Candidate key:

(name, beer)

Is this relation in 3NF?

No: *name* is not a super key,
and *address* is not a part of any candidate key.

Is this one in 3NF?

R(<u>student, course</u>, instructor)

F = {{student, course} → instructor,
instructor → course}

Candidate key: (student, course)

It is in 3NF.

Is 3NF Good Enough?

- Still have **data redundancy**

student,	**course,**	**instructor**
(“John Doe”,	“CS2300”,	“McGeehan”)
(“Bob Jones”,	“CS2300”,	“McGeehan”)

Caused by the FD instructor → course where instructor is not a super key.

BCNF (Boyce-Codd Normal Form)

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

Example

•Keys: { Sid Major, Sid Fname }

•Sid Major → Fname

•Sid Fname → Major

• Fname →Major

SID	MAJOR	FNAME
100	MATH	CAUCHY
100	PHYL	PLATO
200	MATH	CAUCHY
300	PHYS	NEWTON
400	PHYS	EINSTEIN

•The relation is in 3NF but not in BCNF. Observe that Fname→Major is valid, but Fname is not a superkey.

Problem: *Student 300 drops PHYS.*

We lose information that says NEWTON is a PHYS advisor.

•A solution: (SID, FNAME), (FNAME, MAJOR)

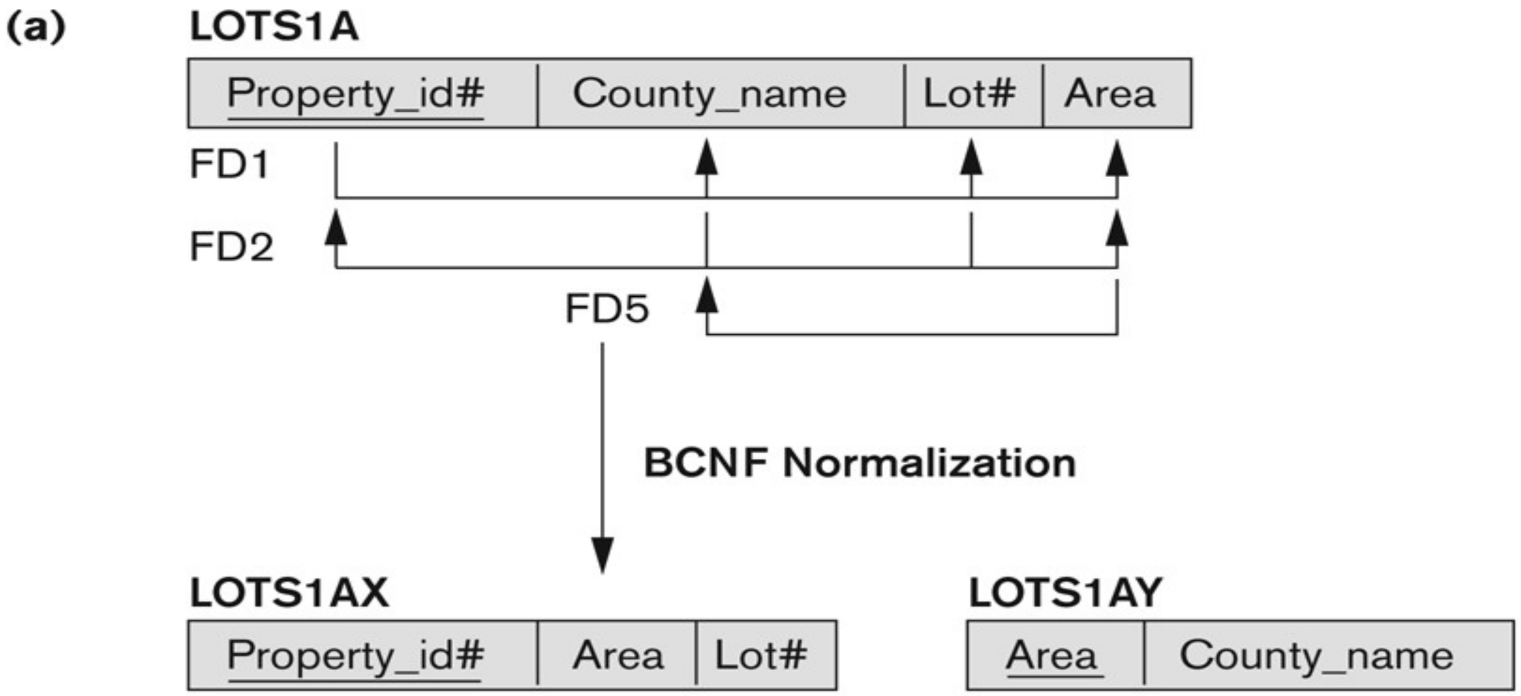
BCNF (Boyce-Codd Normal Form)

- 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



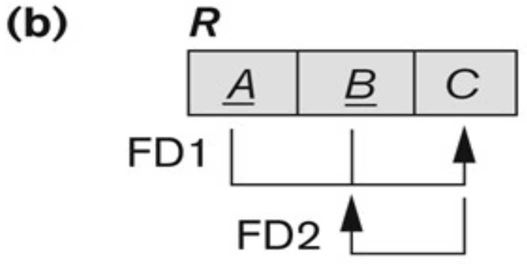


Figure 10.12
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

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
Narayan	Operating Systems	Ammar

Key: Student Course
Student Instructor

Dependencies
Stud Course □Instructor Stud Instructor□Course
Instructor□Course

Figure 10.13
A relation TEACH that is in 3NF but not BCNF.

Achieving the BCNF by Decomposition

- 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 FD { student, course} → instructor
  - We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.

- Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property – to be discussed later) .

Thank YOU