Normalization

Outline

- Decomposition
- Functional dependencies and normalization
- Normalization theory
- First Normal Form
- Second Normal Form
- Third Normal Form
- Boyce-Codd Normal Form

Decomposition

- Suppose we combine *instructor* and *department* into *inst_dept*
- Result is possible repetition of information

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Decomposition

- Suppose we had started with *inst_dept.* How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
- Denote as a **functional dependency**:

  $$dept\_name \rightarrow building,\ budget$$

- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose *inst_dept*
- Not all decompositions are good. Suppose we decompose *employee(ID, name, street, city, salary)* into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

A Lossy Decomposition

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$
  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies

Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

  $$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- Example:
  - R(a,b,c,d,e,f) F={a→bc, e→af}
  - Decomposition of R into R1(a,b,c) and R2(a,d,e,f)
  - R1 $\cap$ R2 = a
  - Since a→bc, it is a lossless join decomposition (a→abc)

Example

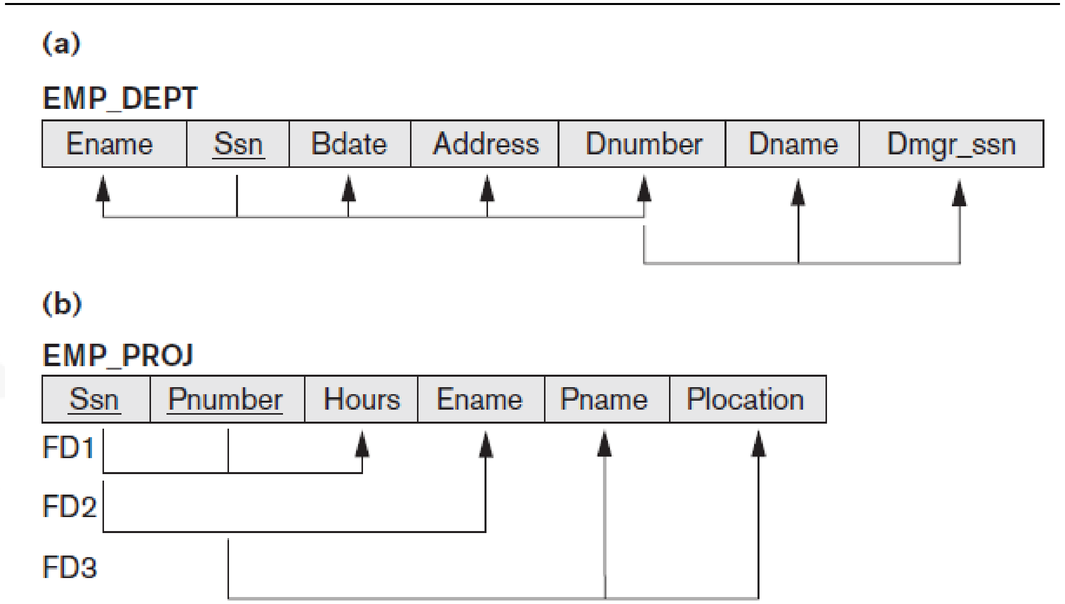
- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    ($B \rightarrow C$ is lost)

Informal Design Guidelines for Relation Schemas

- Measures of quality
  - Making sure attribute semantics are clear: do not combine attributes from multiple entity types and relationship types into a single relation
  - Reducing redundant information in tuples: significant effect on storage space
  - Reducing NULL values in tuples: waste of storage space due to NULLs
  - Disallowing possibility of generating spurious tuples: represent spurious information that is not valid

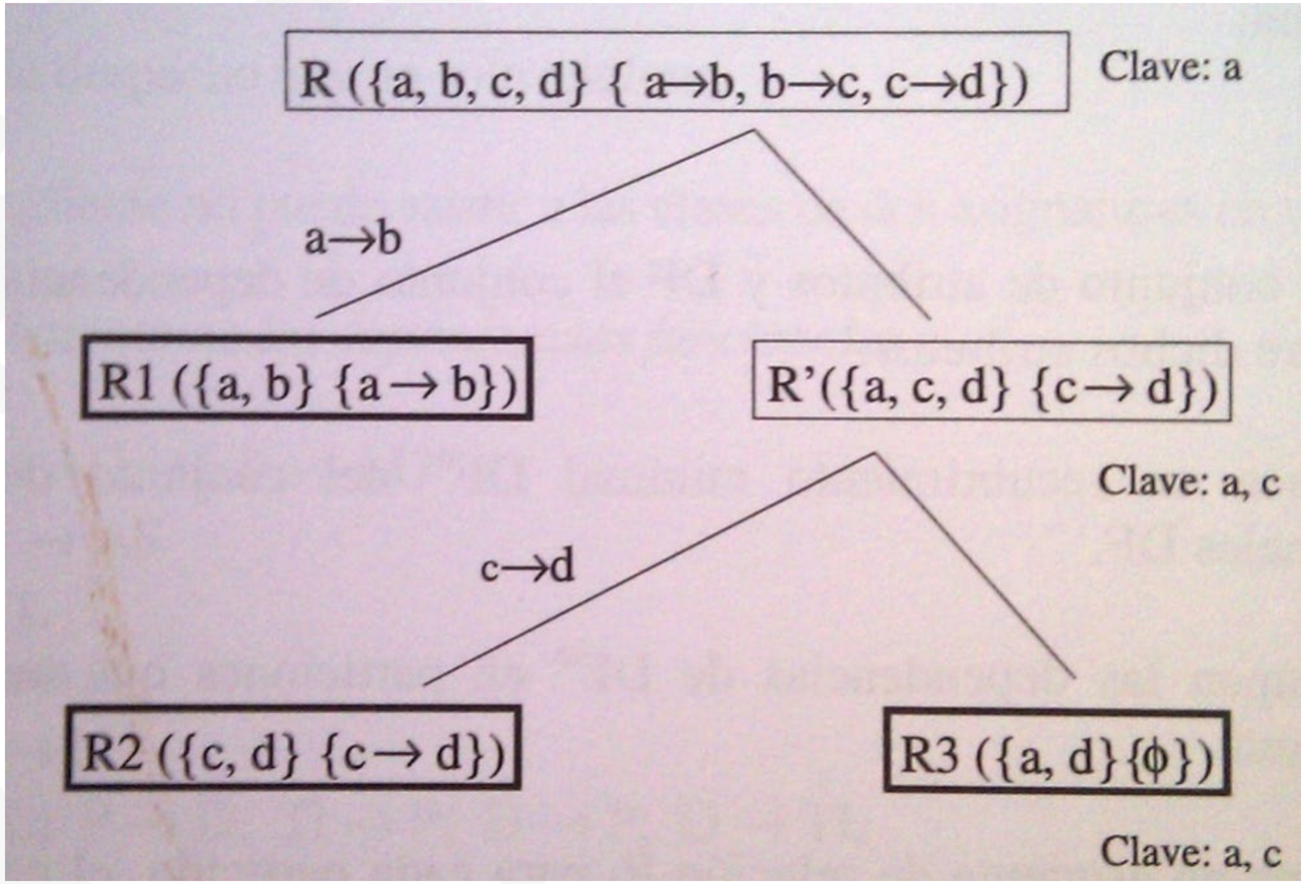


Functional Dependencies and Normalization

- Formal tool for analysis of relational schemas
- Enables us to detect and describe some of the above-mentioned problems in precise terms
- Theory of functional dependency
- Properties that the relational schemas should have:
  - **Nonadditive join property (lossless-join decomposition)**
    - Extremely critical
  - **Dependency preservation property**
    - Desirable but sometimes sacrificed for other factors

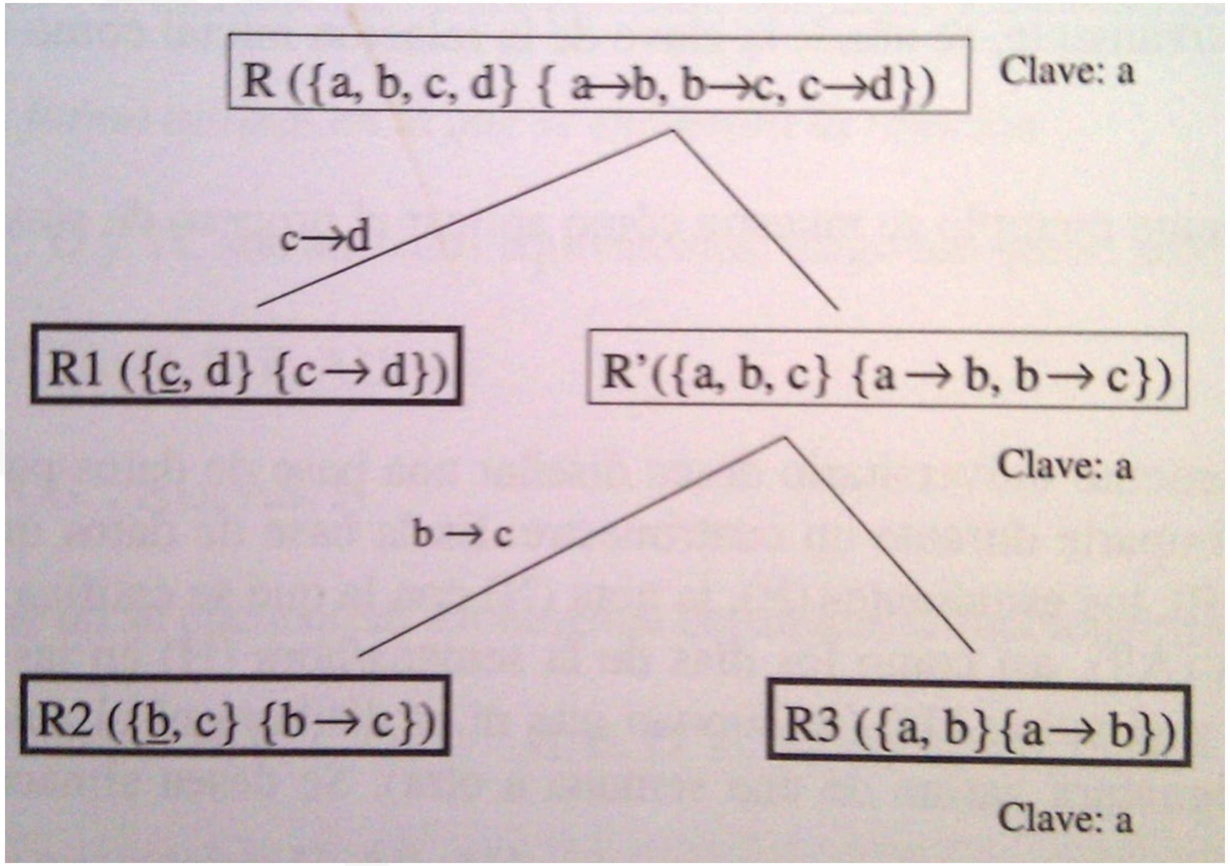
Dependency preservation property

- Incorrect example



Dependency preservation property

- Correct example



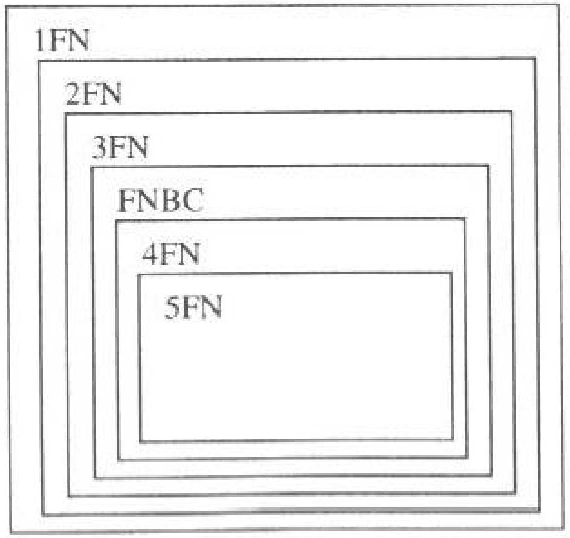
Definitions of Keys and Attributes Participating in Keys

- Definition of **superkey** and **key**
- **Candidate key**
  - If more than one key in a relation schema
    - One is **primary key**

Definition. An attribute of relation schema R is called a **prime attribute** of R if it is a member of *some candidate key* of R. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key.

Normalization Theory

- **Normalization** is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy.
- Let *R* be a relation scheme with a set *F* of functional dependencies.
- Decide whether a relation scheme *R* is in "good" form.
- In the case that a relation scheme *R* is not in "good" form, decompose it into a set of relation scheme {*R*1, *R*2, ..., *Rn*} such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving.

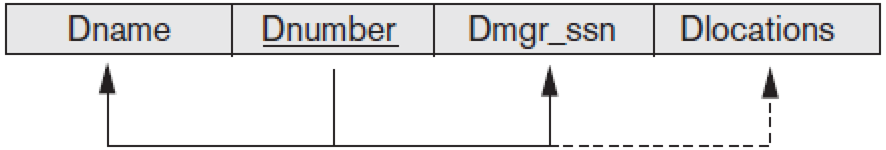


First Normal Form

- Only attribute values permitted are single **atomic (or indivisible) values**
- Techniques to achieve first normal form
  - Remove attribute and place in separate relation
  - Expand the key
  - Use several atomic attributes

(a)

DEPARTMENT



(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 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Second Normal Form

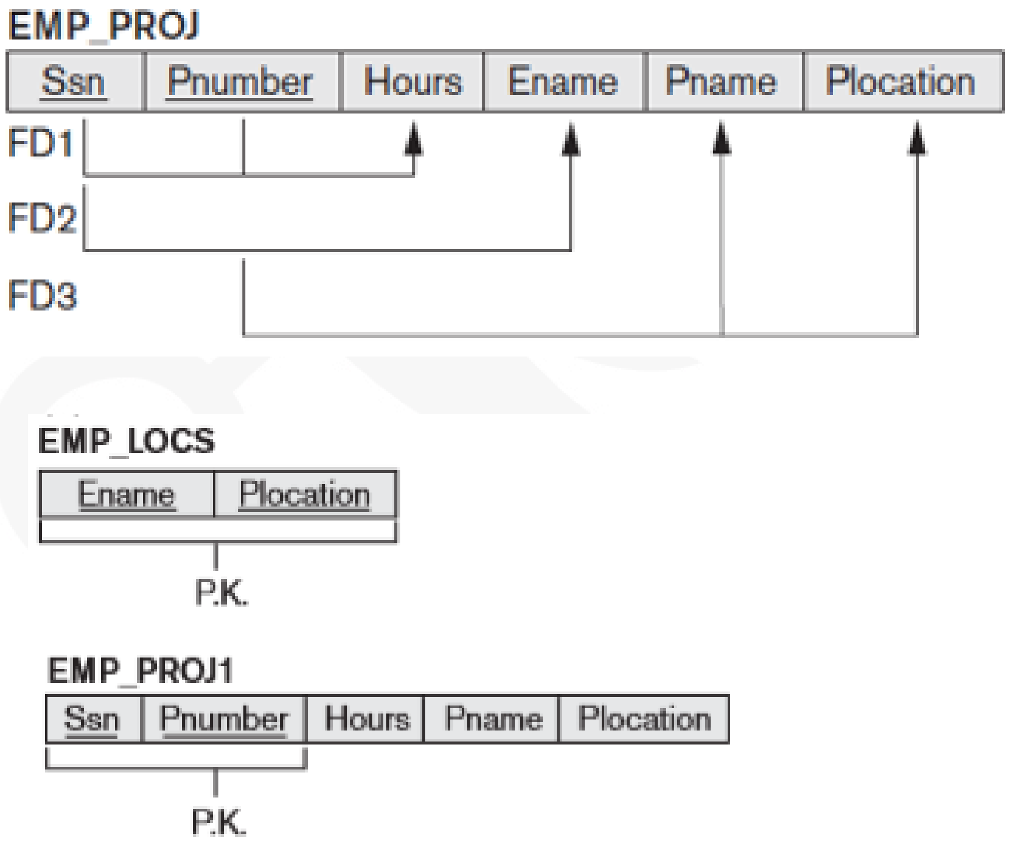
- Based on concept of **full functional dependency**
  - Versus **partial dependency**

- SSN,ProjectNumber → Hours, is full
- SNS,ProjectNumber → NameE, is partial because if ProjectNumber is removed, the functional dependency is still valid.
- Nonprime attributes are associated only with part of primary key on which they are fully functionally dependent

Definition. A relation schema R is in 2NF if every nonprime attribute A in R is *fully functionally dependent* on the primary key of R.

Second Normal Form

- EMP_PROJ is decomposed into relation schemas EMP_LOCS and EMP_PROJ1



Third Normal Form

- Based on concept of transitive dependency: X→Z y Z→Y then X→Y (being Z a nonprime attribute)

Definition. According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF *and* no nonprime attribute of R is transitively dependent on the primary key.

- SSN→DepartmentNumber
- DepartmentNumber→ManagerSSN
- Therefore SSN→ManagerSSN and DepartmentNumber is not a prime attribute

General Definitions of Second and Third Normal Forms

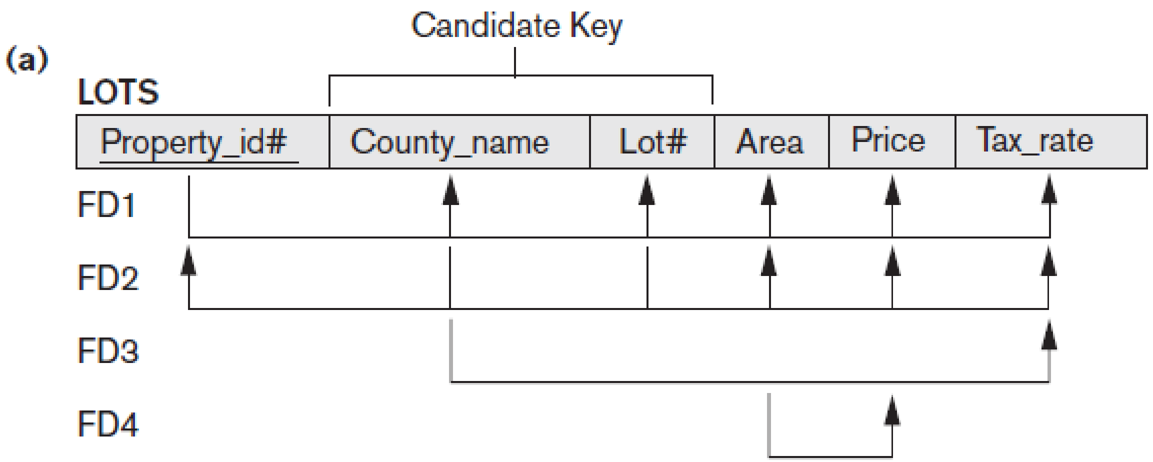
Table 15.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

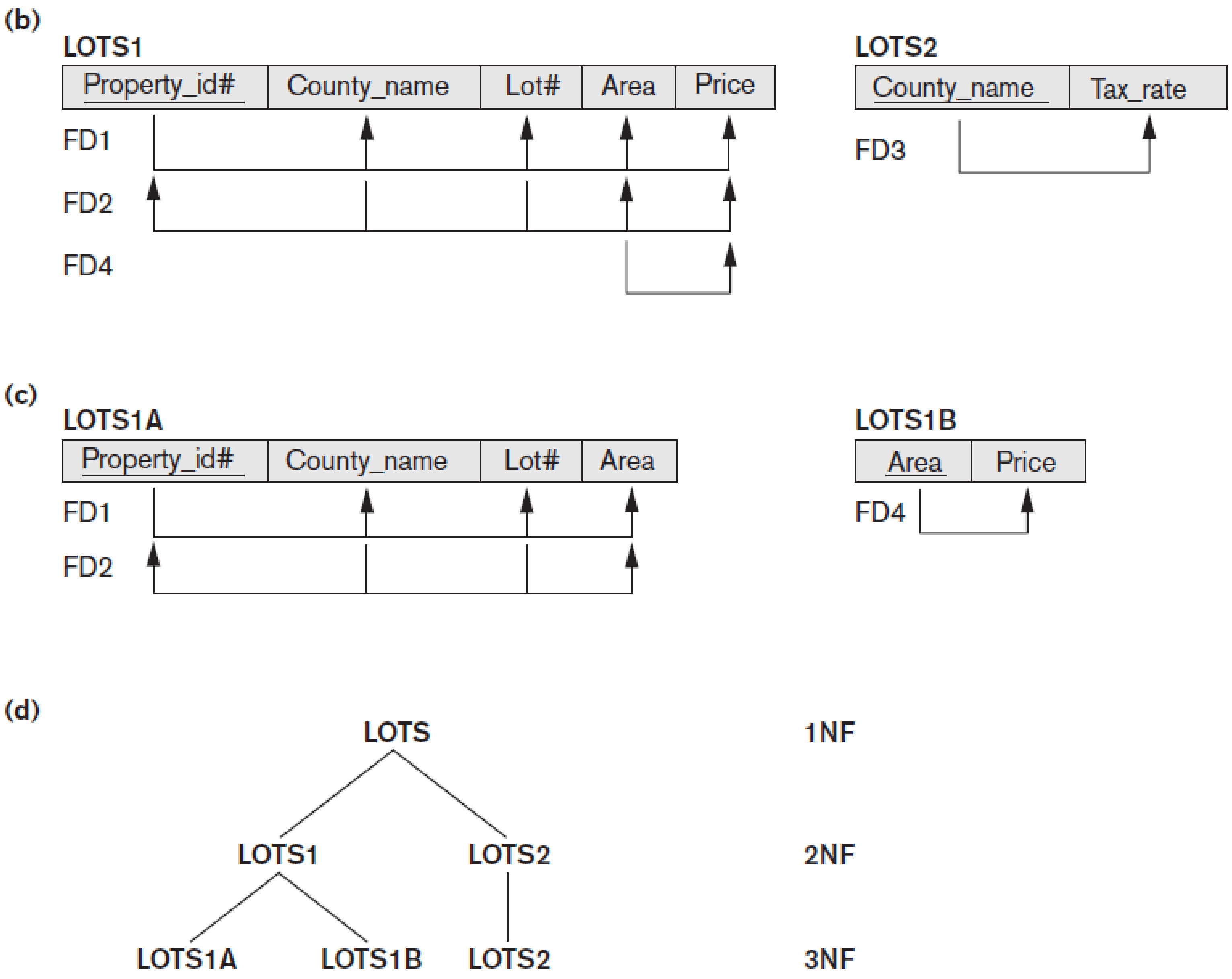
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

General Definition of Second Normal Form

Definition. A relation schema R is in **second normal form** (**2NF**) if every non-prime attribute A in R is not partially dependent on *any* key of R.[11]

Figure 15.12
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.




(b)
LOTS1
Property_id#
County_name
Lot#
Area
Price
FD1
FD2
FD4
LOTS2
County_name
Tax_rate
FD3
(c)
LOTS1A
Property_id#
County_name
Lot#
Area
FD1
FD2
LOTS1B
Area
Price
FD4
(d)
LOTS
1NF
LOTS1
LOTS2
2NF
LOTS1A
LOTS1B
LOTS2
3NF

General Definition of Third Normal Form

Definition. A relation schema R is in **third normal form** (3NF) if, whenever a *nontrivial* functional dependency $X \to A$ holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

Alternative Definition. A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

- It is fully functionally dependent on every key of R.
- It is nontransitively dependent on every key of R.

Boyce-Codd Normal Form

- Every relation in BCNF is also in 3NF
  - Relation in 3NF is not necessarily in BCNF

Definition. A relation schema R is in BCNF if whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.

- Difference:
  - Condition which allows A to be prime is absent from BCNF
- Most relation schemas that are in 3NF are also in BCNF

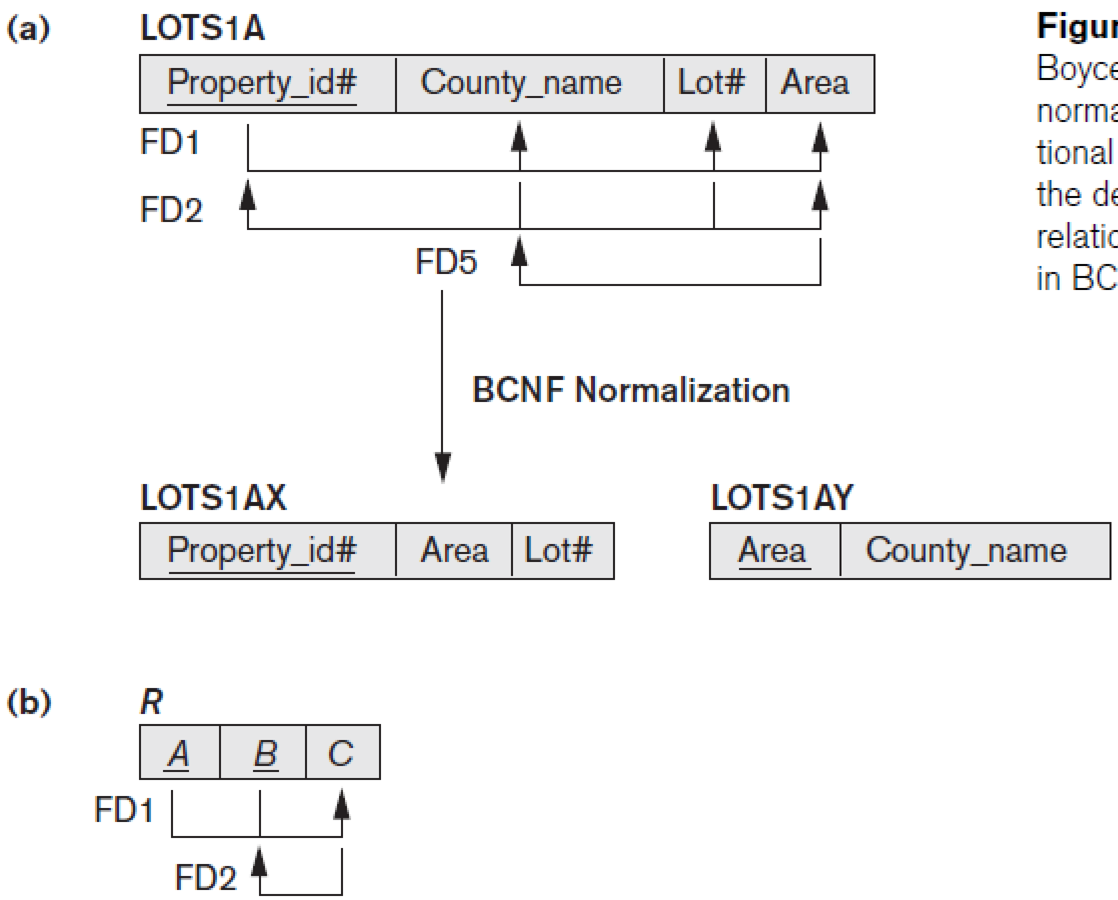


Figure 15.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)

BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?
- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $\quad L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

3NF Example

- Relation *dept_advisor*:
  - *dept_advisor* (*s_ID, i_ID, dept_name)*
    $F = \{s\_ID, dept\_name \rightarrow i\_ID,\ \ i\_ID \rightarrow dept\_name\}$
  - Two candidate keys: *s_ID, dept_name,* and *i_ID, s_ID*
  - *R* is in 3NF
    - *s_ID, dept_name* → *i_ID*
      - *s_ID, dept_name* is a superkey
    - *i_ID* → *dept_name*
      - *dept_name* is contained in a candidate key