Normal Forms

Part 2

Content

- Normalization of Relations
- Practical Use of Normal Forms
- Definitions of Keys and Attributes Participating in Keys
- First Normal Form
- Second Normal Form
- Third Normal Form

Normalization of Relations (1)

- **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
  - 4NF based on keys, multi-valued dependencies;

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 becomes 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)

Definitions of Keys and Attributes Participating in Keys (1)

- A **superkey** of a relation schema $R = \{A_1, A_2, ..., A_n\}$ is a set of attributes S <u>*subset-of*</u> R with the property that no two tuples t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$

minimal super

- A **key** K is a superkey with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

Definitions of Keys and Attributes Participating in Keys (2)

- 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

الصور ما لها type موجود (يعني ما لها ميثود/ تتعامل معها)
الخرائط

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

it handles String

المختصر:
First Normal Form فيه أثناء كثير من تجسيد صور/ خرائط/ الأشكال هندسية مثلاً

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS

Relation schema that is not in 1NF

DEPARTMENT

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

Example relation instance

صعب تحويل الشكل
عشان التكرار

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

1NF with redundancy

EMP_PROJ

SSN	ENAME	PROJS	
		PNUMBER	HOURS

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

EMP_PROJ relation with nested relation PROJS

Second Normal Form (1)

- Uses the concepts of **FD**s, **primary key**

Definitions:

- **Prime attribute** - attribute that is member of the primary key K

A Fully funtional depends on y

- **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

لأن SSN تكفي عشان تعطي Ename

Second Normal Form (2)

$X \rightarrow Y$

Key لازم يكون   non prime

الزبدة بالشرح الآتي:

لو طلع منه مفهوم و prime لازم يكون X

- 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

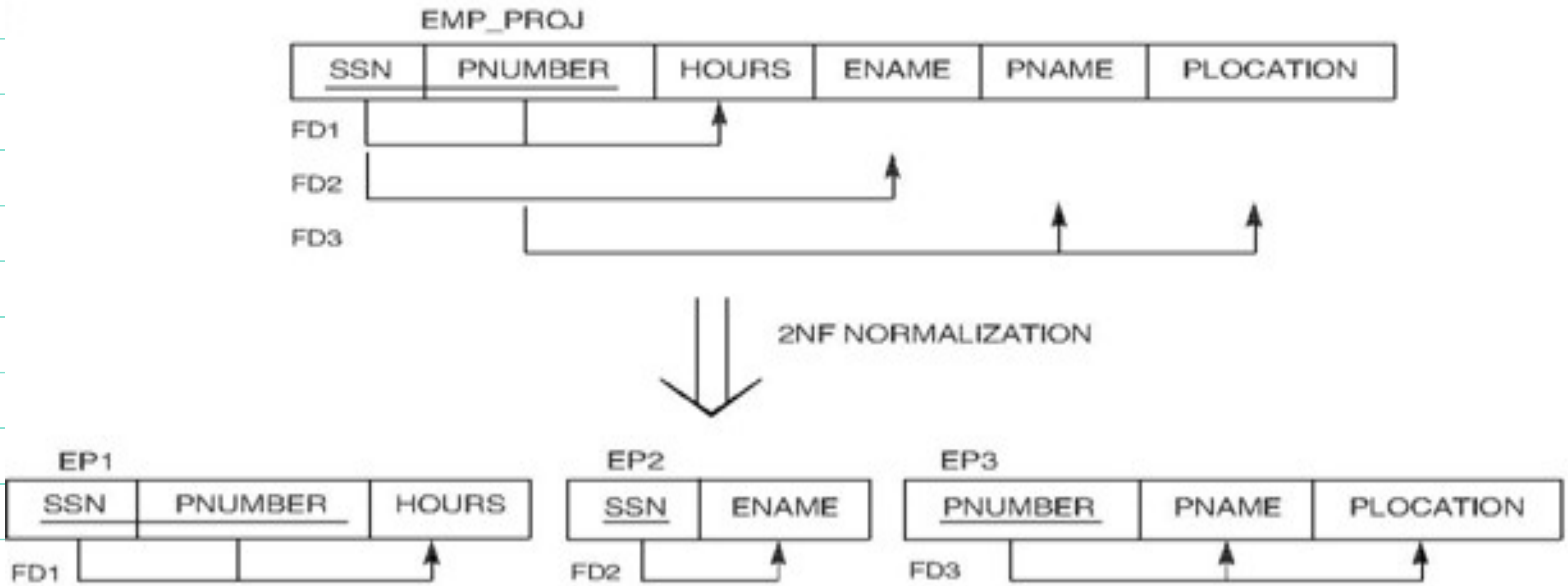
~~prime → non prime~~ : معناه جزء من المفتاح (prime) مب ممكن يعطي أي شيء آخر

non prime → non prime

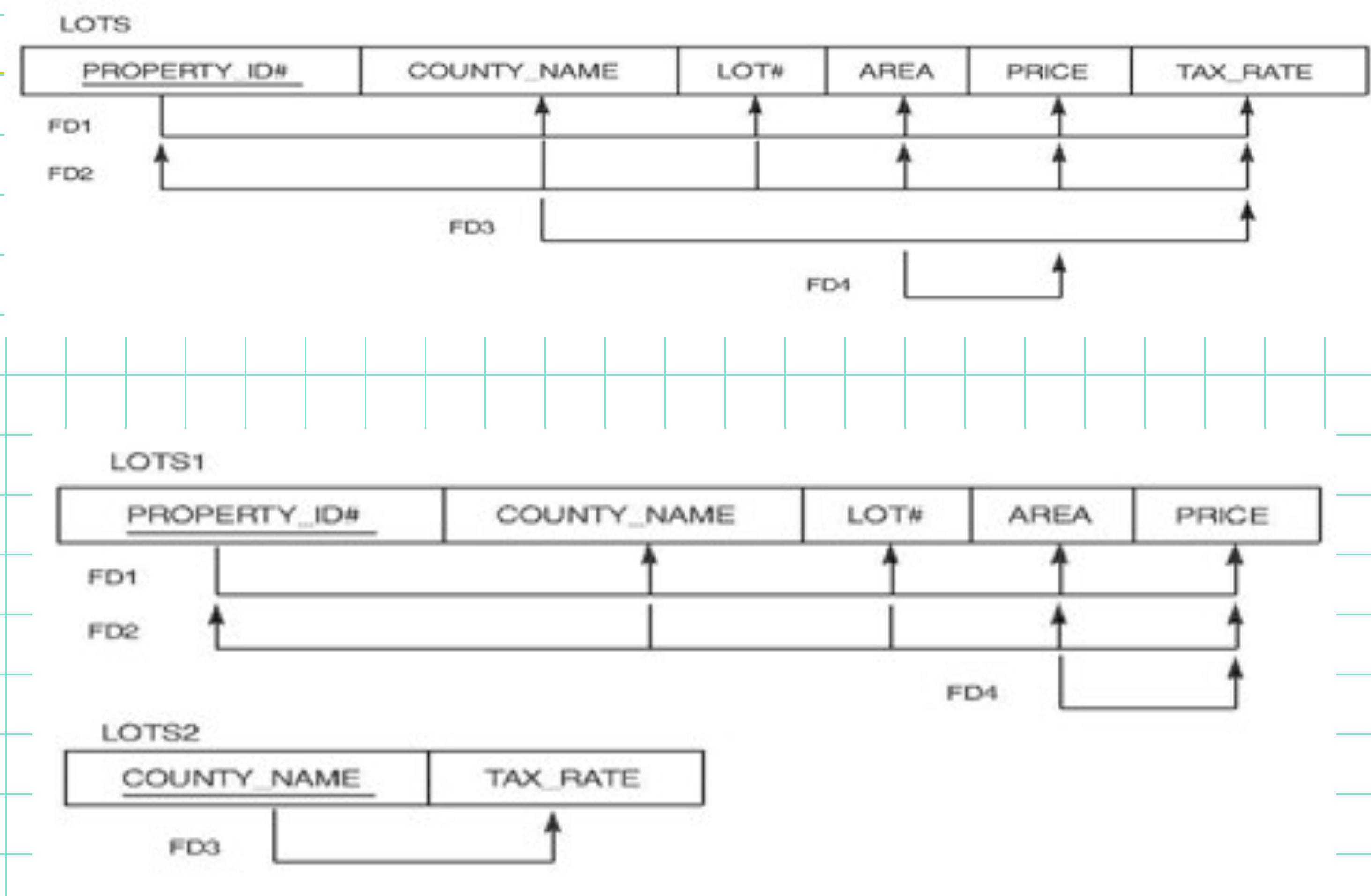
X doesn't fully determine Y

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

Normalizing EMP_PROJ in the 2NF



Normalizing LOTS in the 2NF



Third Normal Form (1)

Definition:

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

  $X \rightarrow Y$ and $Y \rightarrow Z$

  Examples:

  - SSN ➔ DMGRSSN is a *transitive* FD since

  SSN ➔ DNUMBER and DNUMBER ➔ DMGRSSN hold

  - SSN ➔ ENAME is *non-transitive* since there is no set of attributes X where SSN ➔ X and X ➔ ENAME

Third Normal Form (2)

- 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

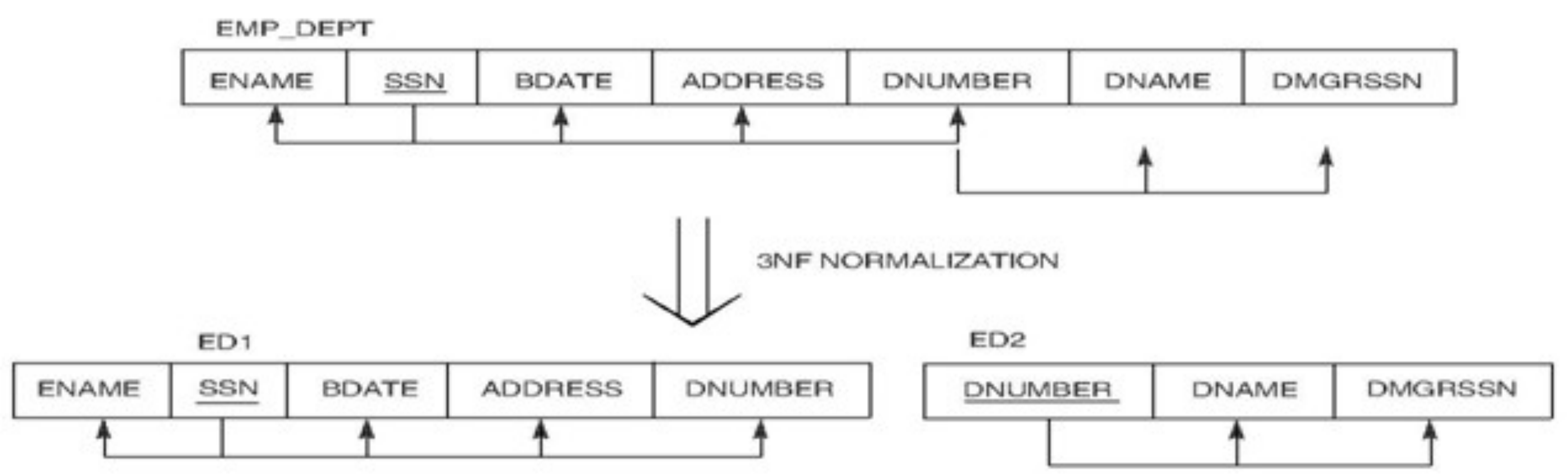
NOTE:

In X ➔ Y and Y ➔ Z, with X as the primary key, we consider this a problem only if Y is <u>not</u> a candidate key. When Y is a candidate key, there is no problem with the transitive dependency .

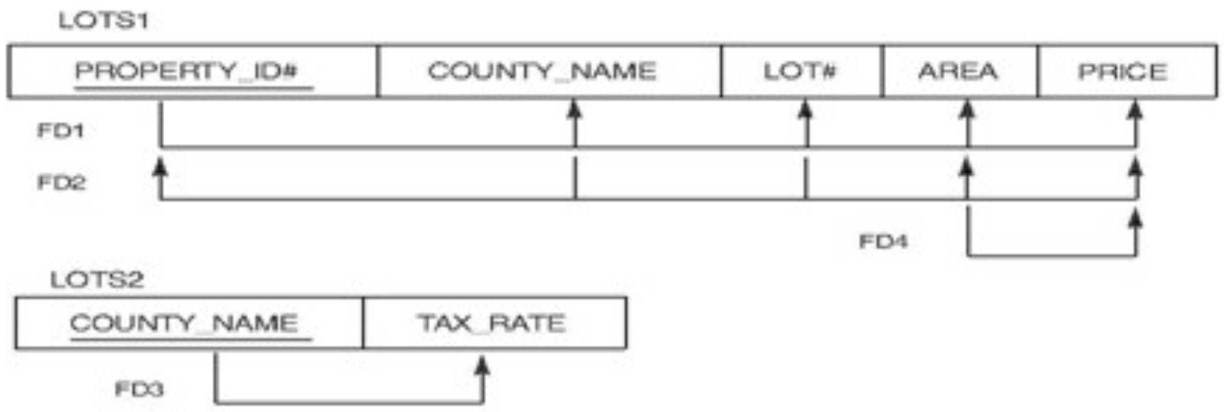
E.g., Consider EMP (SSN, Emp#, Salary).

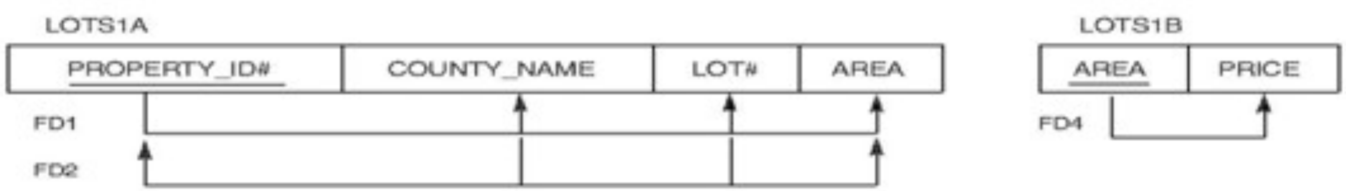
Here, SSN ➔ Emp# ➔ Salary and Emp# is a candidate key.

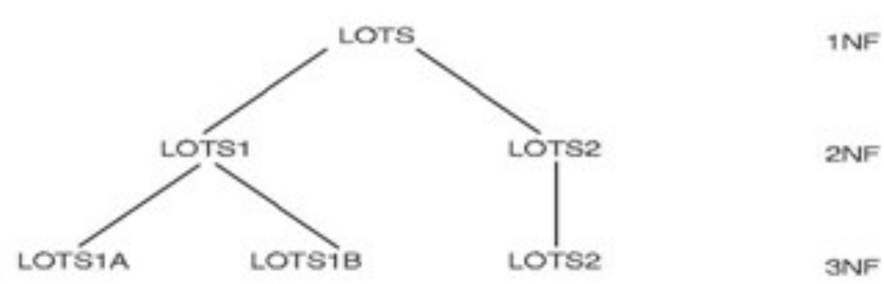
Normalizing EMP_DEPT in the 3NF



Normalizing LOTS1 in the 3NF







General Normal Form Definitions (For Multiple Keys) (1)

- 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 (2)

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 $\boxed{X \rightarrow A}$ 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

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
- 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)

Lossless Decomposition

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$

- Lossless-join decomposition.
  For all possible relations *r* on schema *R*

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

- A decomposition of R into R_1 and R_2 is lossless join if and only if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

Objective of the normalization process

- Lossless-join decomposition: Otherwise decomposition would result in information loss.
- No redundancy: The relations R_i preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
- Dependency preservation: Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - Preferably the decomposition should be dependency preserving, that is, $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - Otherwise, checking updates for violation of functional dependencies may require computing joins, which is expensive.

Example of BCNF Decomposition

- *R* = (*branch-name, branch-city, assets,*
  customer-name, loan-number, amount)
  F = {*branch-name* → *assets branch-city*
  loan-number → *amount branch-name*}
  Key = {*loan-number, customer-name*}
- Decomposition
  - *R*1 = (*branch-name, branch-city, assets*)
  - *R*2 = (*branch-name, customer-name, loan-number, amount*)
  - *R*3 = (*branch-name, loan-number, amount*)
  - *R*4 = (*customer-name, loan-number*)
- Final decomposition

  *R*1, *R*3, *R*4

BCNF and Dependency Preservation

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

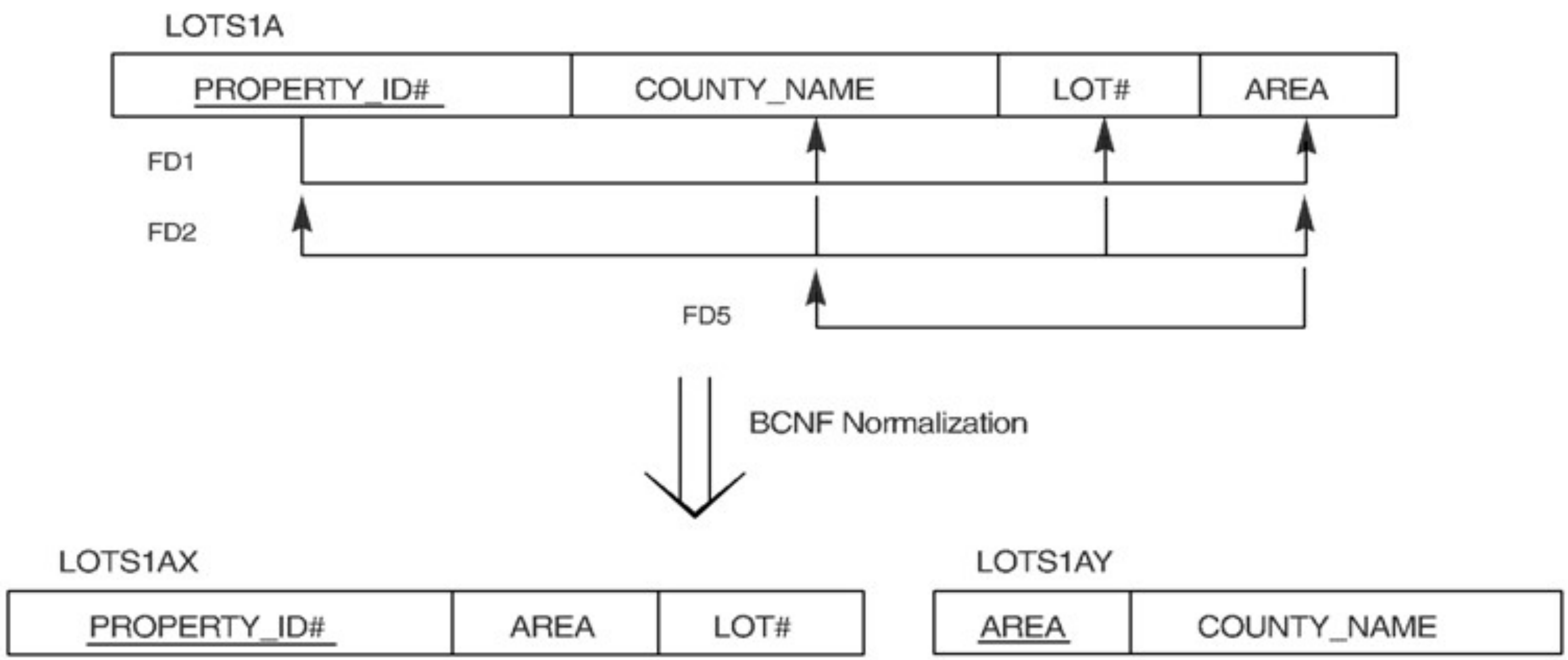
$R = (J, K, L)$
$F = \{JK \rightarrow L, \; L \rightarrow K\}$
Two candidate keys JK and JL

- R is not in BCNF
- Any decomposition of R will fail to preserve

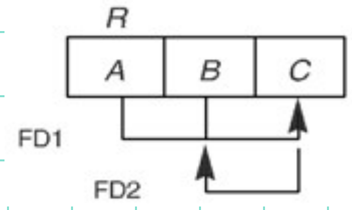
$$JK \rightarrow L$$

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.



A BCNF normalization of FD2 lost in the decomposition



A relation 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

A relation in 3NF but not in BCNF

Achieving the BCNF by Decomposition (1)

- Two FDs exist in the relation TEACH:

  fd1: { student, course} ➔ instructor

  fd2: instructor ➔ course
- {student, course} is a candidate key for this relation
- this relation is in 3NF <u>but not in</u> 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 (2)

- Three possible decompositions for relation TEACH
  1. {<u>student, instructor</u>} and {<u>student, course</u>}
  2. {course, <u>instructor</u> } and {<u>course, student</u>}
  3. {<u>instructor</u>, course } and {<u>instructor, student</u>}
- All three decompositions will lose fd1. We have to settle for sacrificing the functional dependency preservation. But we <u>cannot</u> 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).
- A test to determine whether a <u>binary decomposition</u> (decomposition into two relations) is nonadditive (lossless)
- Verify that the third decomposition above meets the property.