Designing Good Designed Database

- Functional Dependencies:
- Partial dependencies
- Transitive dependency
- Join dependency
- Multivalued dependency
- Update anomalies
- Normalization
- Normal Form:
  - First, second, third, fourth, fifth

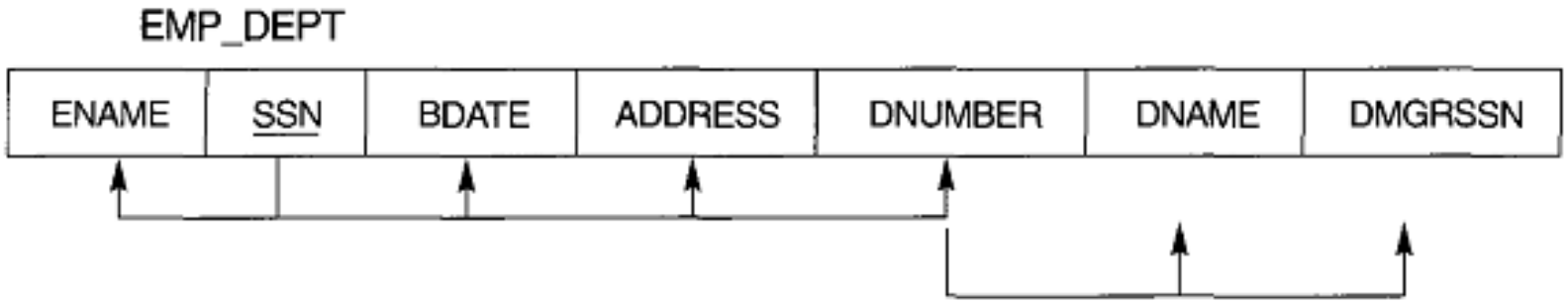
INFORMAL DESIGN GUIDE LINES FOR RELATION SCHEMAS

- Semantics of the attributes
- •Reducing the redundant values in tuples AND update anomalies
- •Reducing the null values in tuples
- •Disallowing the possibility of generating spurious tuples

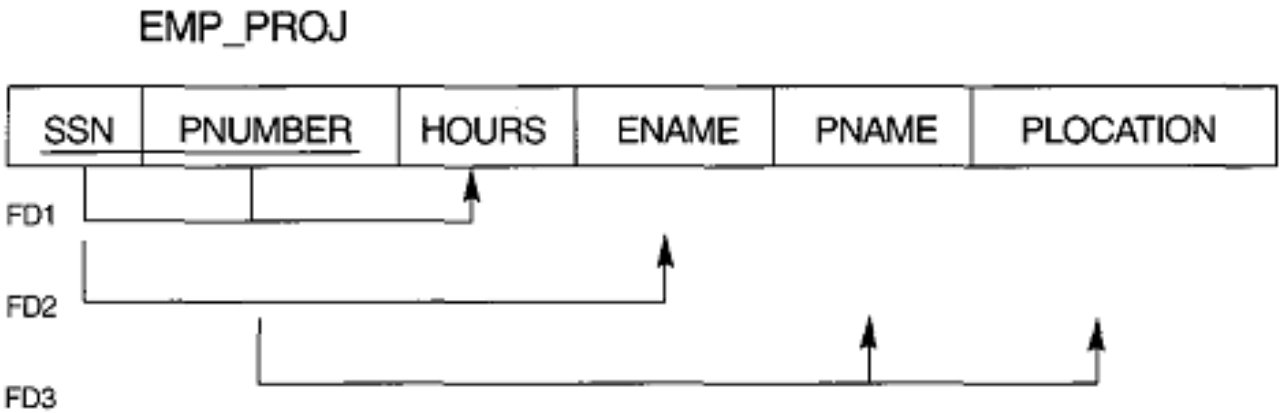
Semantics of the attributes

- GUIDELINE 1.
- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- Intuitively, if a relation schema corresponds to one entity type or one relation-ship type, it is straightforward to explain its meaning. Other wise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

(a)


EMP_DEPT
ENAME
SSN
BDATE
ADDRESS
DNUMBER
DNAME
DMGRSSN

(b)


EMP_PROJ
SSN
PNUMBER
HOURS
ENAME
PNAME
PLOCATION
FD1
FD2
FD3

Reducing the redundant values in tuples & update anomalies

- GUIDELINE2
- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly

Reducing the null values in tuples

Nulls can have multiple interpretations, such as the following:

- The attribute does not apply to this tuple.
- The attribute value for this tuple is unknown.
- The value is known but absent; that is, it has not been recorded yet.

GUIDELINE3.

- As far as possible, avoid placing attributes in a base relation whose values may frequently be null.
- If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Disallowing the possibility of generating spurious tuples

GUIDELINE4.

- Design relation schemas so that they can be joined with equality

conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

- Avoid relations that contain matching
- Attributes that are not(foreign key, primary key) combinations, because joining on such attributes may produce spurious tuples.

FUNCTIONALDEPENDENCIES

Definition.

A functional dependency, denoted by X →Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that

have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component.

We also say

- that there is a functional dependency from X to Y, or that Y is functionally dependent on X.
- The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD , and Y is called the right-hand side.
- Thus, X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X- value, they must necessarily agree on their Y-value.

Inference Rules for Functional Dependencies

The following six rules IRI through IR6 are well known inference rules for functional dependencies:

IR1 (reflexive rule[8]): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (augmentation rule[9]): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

IR4 (decomposition, or projective, rule): $\{X \rightarrow YZ\} \models X \rightarrow Y$.

IR5 (union, or additive, rule): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.

IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

Normalization

- Normalization is the process of decomposing a “bad” relation by breaking up their attributes into smaller relations.
- The process of normalization through decomposition should conform the following two properties:
  - The lossless join or non additive join property, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition
  - The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition

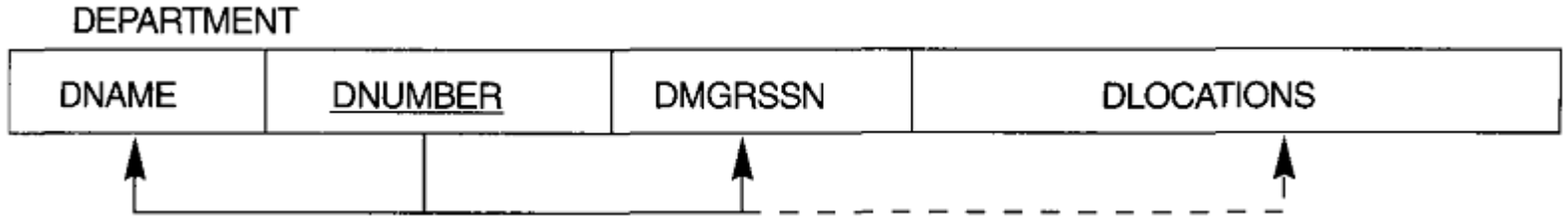
- During normalization we analyze the given relation schema based on their functional dependencies(FD) and primary key to achieve the desired properties of
  - Minimizing redundancy and
  - Minimizing insertion, deletion and update anomalies.
- Unsatisfactory relation schemas that do not meet certain conditions: the normal form test are performed so that those unsatisfactory relation are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Hence, a normal form is a condition using keys and FD's of a relation to certify whether a relation schema is in a particular normal form.

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.
- Although several higher normal forms have been defined, the practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect by the database designer and users who must discover these constraints.
- Thus database design as practiced in industry today pay particular attention to normalization only up to 3NF, BCNF or 4NF.
- The process of storing the Join of higher normal form relations which is in a lower normal form is known as **denormalization**

First Normal Form(1NF)

- First normal form (INF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model.
- Historically, it was defined to disallow multival- ued attributes, composite attributes, and their combinations.
- It states that the domain of an attribute must include only **atomic** (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- Hence, INF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.
- In other words, INF disallows "relations within relations" or "relations as attribute values within tuples." The only attribute values permitted by lNF are single atomic (or indivisible) values.



- Consider the DEPARTMENT relation schema shown in above Figure , whose primary key is DNUMBER
- We assume that each department can have a number of locations.
- We see that this relation is not in 1NF because DLOCATION is not an atomic attributes.
- To achieve 1NF for such a relation, we remove attribute DLOCATION that violates 1NF and place it in a separate relation DEPT_LOCATION along with the primary key DNUMBER of DEPERTMENT.
- The primary key is the combination of {DNUMBER,DLOCATION}

DEPARTMENT

DNAME	DNUMBER	DMGRSSN

DEPT_LOCATIONS

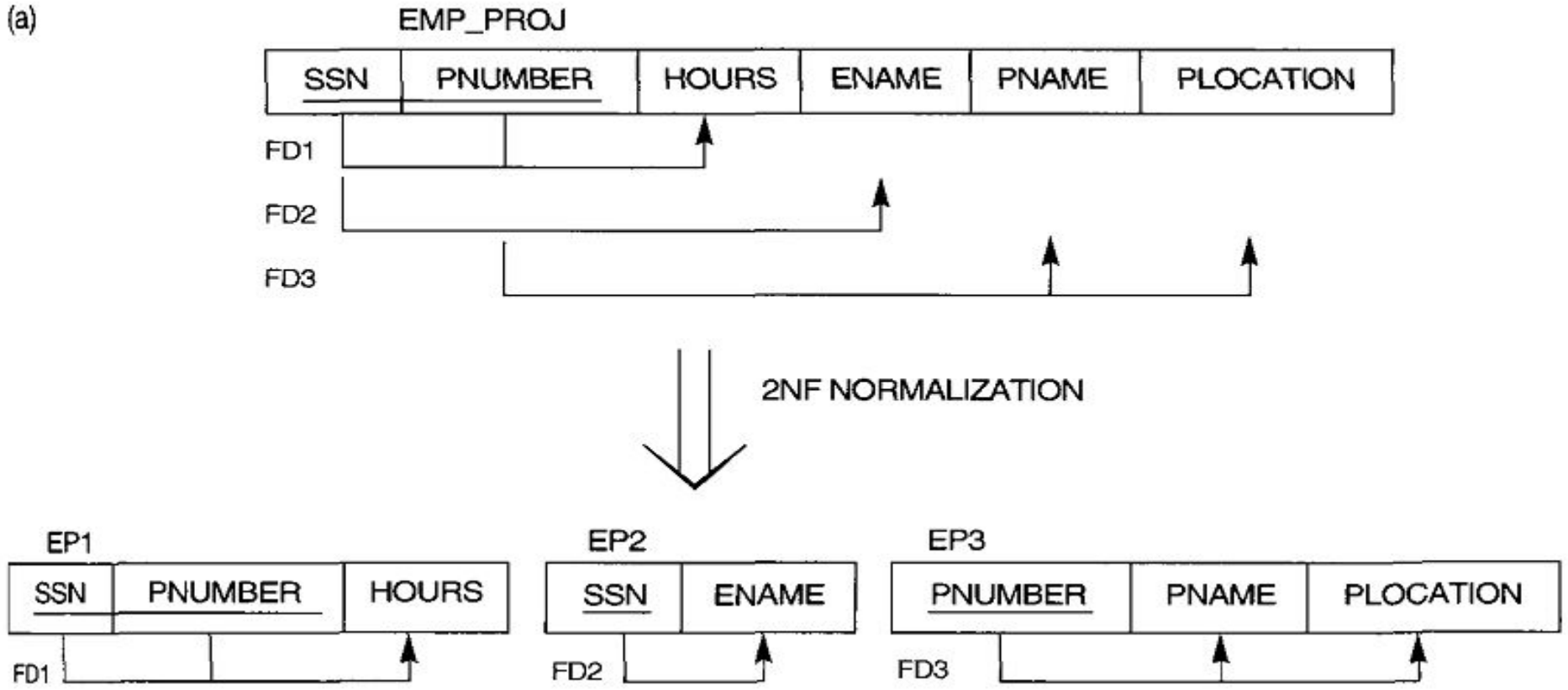
DNUMBER	DLOCATION

Second normal form (2NF)

- Second normal form (2NF) is based on the concept of full functional dependency.
- A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ϵ X, (X - {A}) does not functionally determine Y.
- A functional dependency X → Y is a partial dependency if some attribute A ϵ X can be removed from X and the dependency still holds; that is, for some A E X, (X - {A}) → Y.
- In Figure {SSN, PNUMBER} → HOURS is a full dependency (neither SSN → HOURS nor PNUMBER → HOURS holds). However, the dependency {SSN, PNUMBER} → ENAME is partial because SSN → ENAME holds.

- 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.

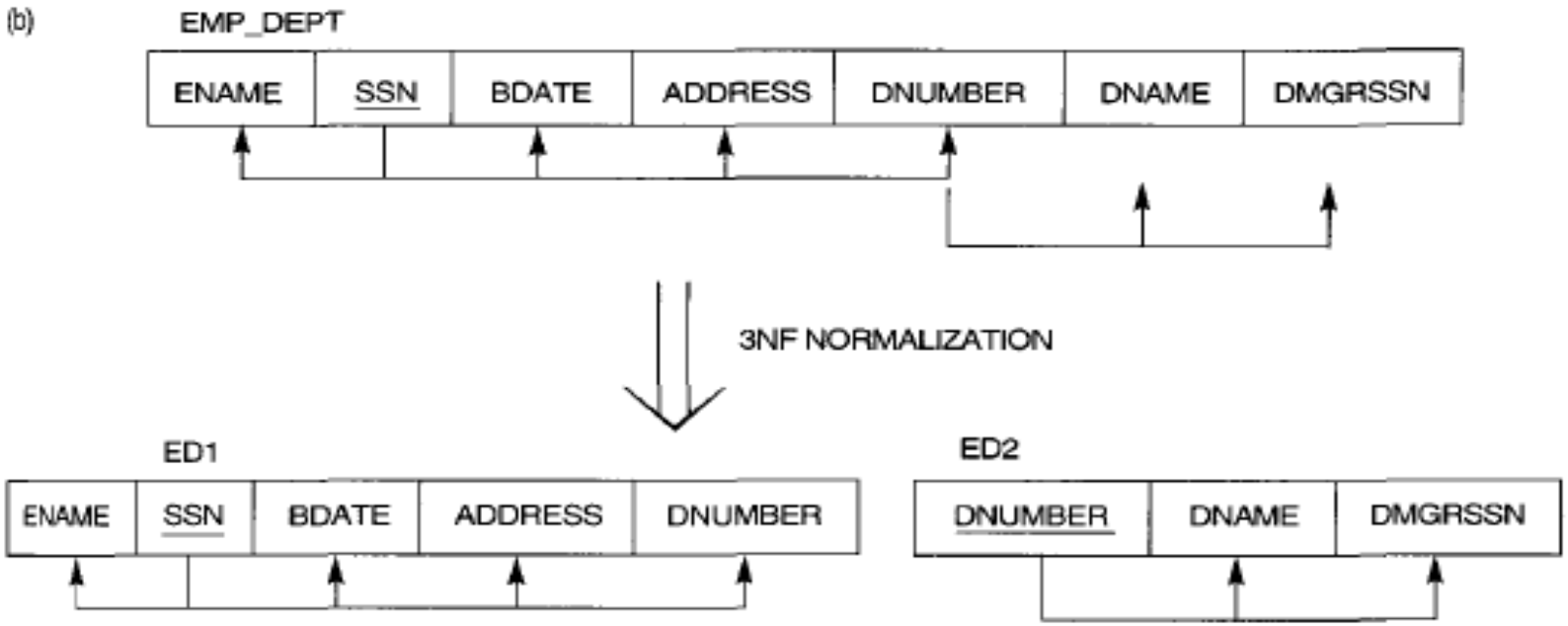


The above relation schema is not in 2NF, it can be "second normalized" or "2NFnormalized" into a number of 2NFrelations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.
Hence, the decomposition of EMP_PROJ into the three relation schemas EPl, EP2, and EP3 shown in Figure can achieve 2NF.

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

Third normal form (3NF)

- Third normal form (3NF) is based on the concept of *transitive dependency.*
- A functional dependency X →*Y* in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R, and both X → Z and Z →Y hold.



- The dependency SSN → DMGRSSN is transitive through DNUMBER in EMP_DEPT table as shown in the figure above because both the dependencies SSN → DNUMBER and DNUMBER → DMGRSSN hold *and* DNUMBER is neither a key itself nor a subset of the key of EMP_DEPT.
- **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.
- The relation schema EMP_DEPT in above Figure is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN (and also DNAME) on SSN via DNUMBER.
- We can normalize EMP_DEPT by decomposing it into the two 3NFrelation schemas EDl and ED2 shown in Figure.

General Definition of Third Normal Form

A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X →A holds in R,

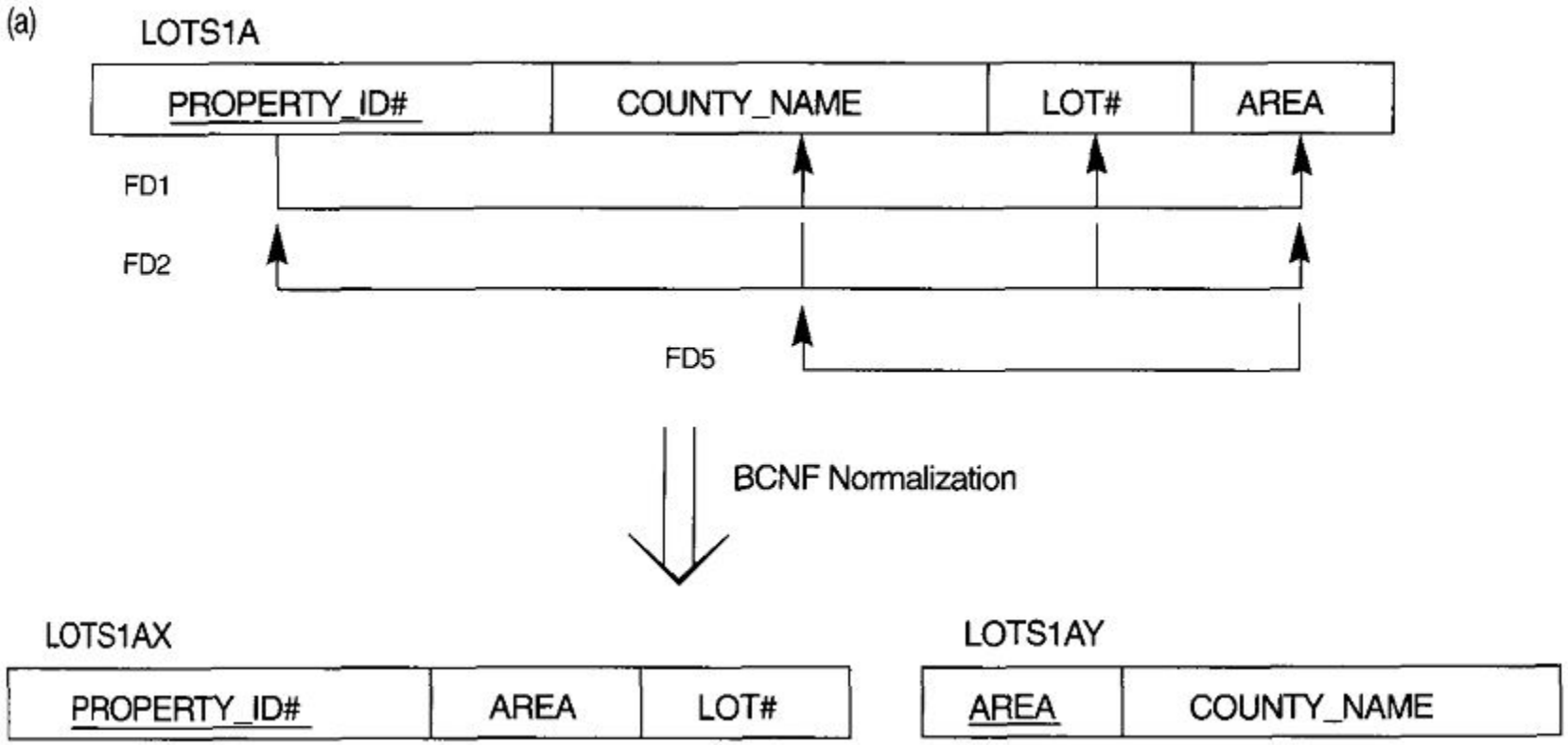
either (a) X is a super key of R,

or (b) A is a prime attribute of R.

BOYCE-CODD NORMAL FORM(BCNF)

- BCNF is simpler than 3NF but strict than 3NF. Every relation in BCNF is also in 3NF, however a relation in 3NF is not necessarily in BCNF.
- **Definition.** A relation schema R is in BCNF if whenever a nontrivial functional dependency X → A holds in R, then X is a super key of R.
- The formal definition of BCNF differs slightly from the definition of 3NF. The only difference between the definitions of BCNF and 3NF is that condition (b) of 3NF, which allows A to be prime, is absent from BCNF

Consider the following example:



Multivalued Dependencies and Fourth Normal Form

(a) The EMP relation with two MVDs: ENAME —>> PNAME and ENAME —>> DNAME.

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Definition:

- A **multivalued dependency** (**MVD**) X —>> Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R:
- If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

  $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.

  $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.

  $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.
- An MVD X —>> Y in R is called a **trivial MVD** if (a) Y is a subset of X, or (b) $X \cup Y = R$.

Join Dependencies and Fifth Normal Form

Definition:

- A **join dependency** (**JD**), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a non-additive join decomposition into R_1, R_2, ..., R_n; that is, for every such r we have

  * $(\pi_{R1}(r), \pi_{R2}(r), ..., \pi_{Rn}(r)) = r$

- A join dependency JD(R_1, R_2, ..., R_n), specified on relation schema R, is a **trivial JD** if one of the relation schemas R_i in JD(R_1, R_2, ..., R_n) is equal to R.

Definition:

- A relation schema R is in **fifth normal form** (**5NF**) (or **Project-Join Normal Form** (**PJNF**)) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (that is, implied by F), every R_i is a superkey of R.