UNIT 4
DATABASE DESIGN THEORY AND NORMALIZATION

Informal Design Guidelines for Relation Schemas

In the formal theory of relational database design, we discuss four *informal guidelines that may be used as measures to determine the quality of relation* schema design:

- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

Informal Design Guidelines for Relation Schemas

a) Imparting Clear Semantics to Attributes in Relations

Group the attributes forming a relational schema with certain meaning associated with the attributes. The easier it is to explain the semantics of the relation, the better the relational schema design. Schemas should have well defined & unambiguous interpretations having clear semantics.

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.

Examples of Guideline 1 Violation: Although there is nothing wrong logically with the relation, it violates Guideline 1 by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

Informal Design Guidelines for Relation Schemas

b) Redundant Information in Tuples and Update Anomalies

One of the many purposes of schema design is to minimize storage space that the relation occupies. Grouping attributes into relational schema has a significant effect on storage space & we also have to deal with the problem of update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

i) Insertion Anomalies: Insertion anomalies can be differentiated into two types,
illustrated by the following examples based on the EMP_DEPT relation:

➢Considering the relation EMP_DEPT(SSN, Ename, Bdate, Addr, Dno, Dname, Dmgrssn) & the relation EMP_PROJ(SSN,Pno, Hrs, Ename,Pname), it is difficult to insert a new department that has no employees. The only way to do this is to place NULL values in the attribute for employee.

➢It is not possible to insert employee details for an employee who has not been assigned to any department. In such a case, we have to include NULL values

Informal Design Guidelines for Relation Schemas

ii) Deletion Anomalies: This anomaly occurs if we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is also lost inadvertently from the database.

iii) Modification Anomalies: In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all employees who work in that department; otherwise, the* database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

It is easy to see that these three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates that can be avoided; hence, we can state the next guideline as follows.

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any

Informal Design Guidelines for Relation Schemas

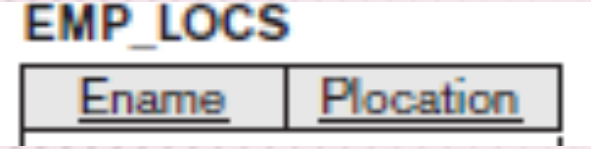
c) NULL Values in Tuples: If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes. Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied. SELECT and JOIN operations involve comparisons, if NULL values are present, the results may become unpredictable. NULLs can have multiple interpretations, such as the following:

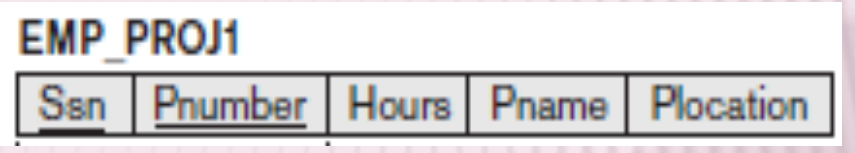
- The attribute *does not apply to this tuple. For example, Visa_status may not* apply to U.S. students.
- The attribute value for this tuple is *unknown. For example, the Date_of_birth* may be unknown for an employee.
- The value is *known but absent; that is, it has not been recorded yet. For* example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

Guideline 3. 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.

Informal Design Guidelines for Relation Schemas

d) Disallowing the possibility of generating Spurious Tuples:
Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ relation.





Joining the above two relations using natural join, is going to result in generation of some extra spurious tuples which amounts to incorrect information. This is because, in this case, Plocation is the common attribute relating EMP_LOCS & EMP_PROJ1, which is neither a primary key nor a foreign key.

Guideline 4. Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs 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.

Functional Dependencies

Definition of Functional Dependency: A functional dependency is a constraint between two sets of attributes from the database.

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 t1[X] = t2[X], they must also have t1[Y] = t2[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.
- The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

Consider the relation schema EMP_PROJ, from the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn → Ename

b. Pnumber → {Pname, Plocation}

Inference Rules for Functional Dependencies

Definition: An FD *X → Y is* ***inferred*** *from* ***or implied*** *by a set of dependencies F specified on R if X → Y holds in every legal relation state r of R; that is, whenever r satisfies all the dependencies in F, X → Y also holds in r.*

Informal **Definition.** Formally, the set of all dependencies that include *F as well as all* dependencies that can be inferred from *F is called the closure of F; it is denoted* by *F+.*

For example, suppose that we specify the following set *F of obvious functional*
dependencies on the relation schema EMP_PROJ1:
F = {Ssn → {Ename, Bdate, Address, Dnumber},
Dnumber → {Dname, Dmgr_ssn} }

Some of the additional functional dependencies that we can *infer from F are the* following:
Ssn → {Dname, Dmgr_ssn}
Dnumber → Dname

Inference Rules for Functional Dependencies

The closure *F+ of F is the set of all functional dependencies that can be inferred* from *F. To determine a systematic way to infer dependencies, we must discover a* set. of inference rules that can be used to infer new dependencies from a given set of dependencies. They were proposed first by Armstrong (1974) and hence are known as **Armstrong's axioms.**

Inference Rules for FDs:

1) Reflexive rule: If $X \supseteq Y$, *then* $X \rightarrow Y$

2) Augmentation rule: $\{X \rightarrow Y\} \models XZ \rightarrow YZ$

3) Transitive rule: $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

4) Decomposition/projective rule: $\{X \rightarrow YZ\} \models X \rightarrow Y$

5) Union, or additive, rule: $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

6) Pseudo transitive rule: $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Normalization of Relations

Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) Minimizing redundancy and (2) Minimizing the insertion, deletion, and update anomalies.

An unsatisfactory relation schema that does not meet the condition for a normal form—the **normal form test**—is decomposed into smaller relation schemas that contain a subset of the attributes and meet the test that was otherwise not met by the original relation. Thus, the normalization procedure provides database designers with the following:

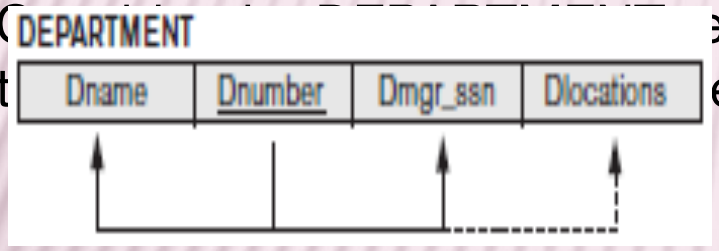
- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree

Definition. The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

First Normal Form

First normal form (1NF) Definition: 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.

In effect, 1NF disallows multi-valued attributes, composite attributes & their combinations and also nested relations. *The* only attribute values permitted by 1NF are single **atomic (or indivisible) values.**



DEPARTMENT

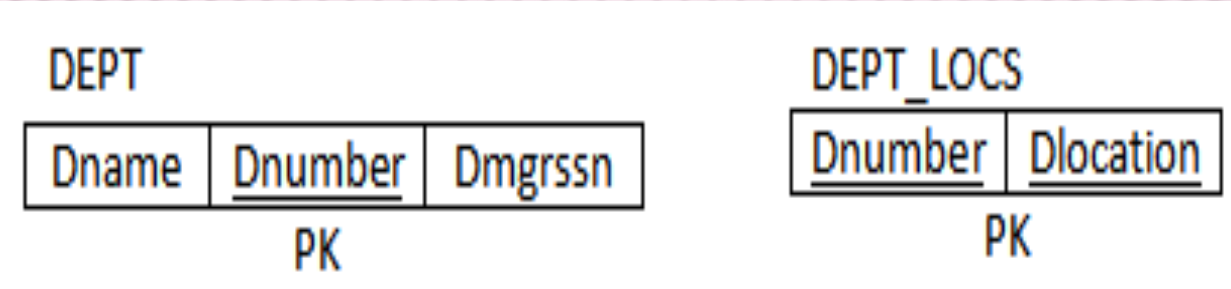
Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

As we can see, this is not in 1NF because Dlocations is not an atomic attribute,
as illustrated by the first tuple in the relation.

First Normal Form

There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this newly formed relation is the combination {Dnumber, Dlocation}, as shown in the fig below. A distinct tuple in DEPT_LOCATIONS exists for *each location of a department. This decomposes* the non-1NF relation into two 1NF relations.



First Normal Form

2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing *redundancy in* the relation and hence

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

3. If a *maximum number of values is known for the attribute—for example, if it* is known that *at most three locations can exist for a department—replace the* Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing *NULL values if* most departments have fewer than three locations.

DEPT

Dname	Dnumber	Dmgrssn	Dlocation1	Dlocation2	Dlocation3
Research	5	333445555	Bellaire	Sugarland	Houston
Administration	4	987654321	Stafford	NULL	NULL
Headquarters	1	888665555	Houston	NULL	NULL

Of the three solutions above, the first is generally considered best.

Second Normal Form

Second normal form (2NF) is based on the concept of full functional dependency.

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 anymore; that is, for* any attribute *A ε X, (X – {A}) does not functionally determine Y.*

Eg: If {Ssn, Pno} → Hrs is a full dependency, then neither {Ssn → Hrs} nor {Pno → Hrs} holds.

Partial functional dependency: 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 ε X, (X – {A}) → Y.

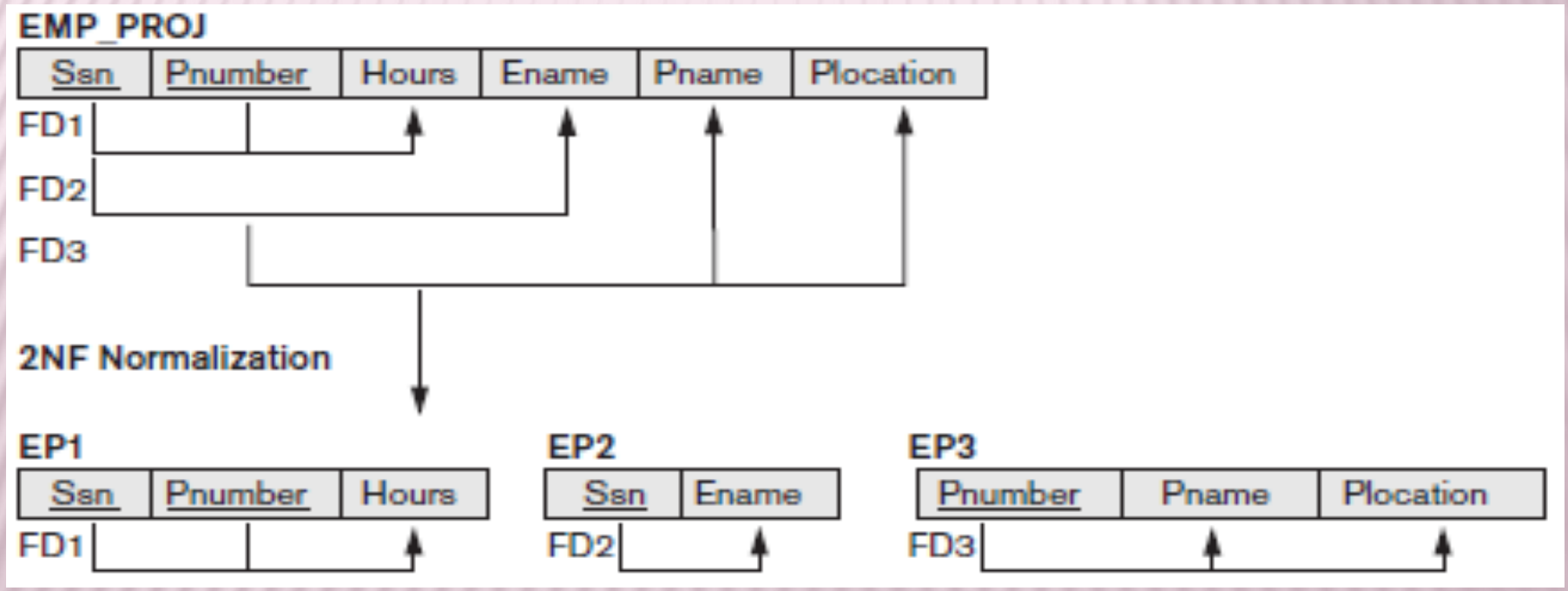
Eg: If {Ssn, Pnumber} → Ename is partial because {Ssn → Ename} holds.

Second Normal Form

2NF 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.*

OR

Definition. A relation schema *R is in second normal form (2NF) if every* nonprime attribute *A in R is not partially dependent on any key of R.*



Third Normal Form

Third normal form (3NF) is based on the concept of *transitive dependency.*

Transitive dependency: A functional dependency X → Y in a relation schema R is a **transitive dependency** if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold.

3NF Definition: 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.

Non Trivial FD: If a functional dependency X→Y holds true where Y is not a subset of X then this dependency is called non trivial FD otherwise it is called trivial FD.

For example:

An employee table with three attributes: emp_id, emp_name, emp_address.

The following functional dependencies are non-trivial:

emp_id → emp_name (emp_name is not a subset of emp_id)

emp_id → emp_address (emp_address is not a subset of emp_id)

On the other hand, the following dependencies are trivial:

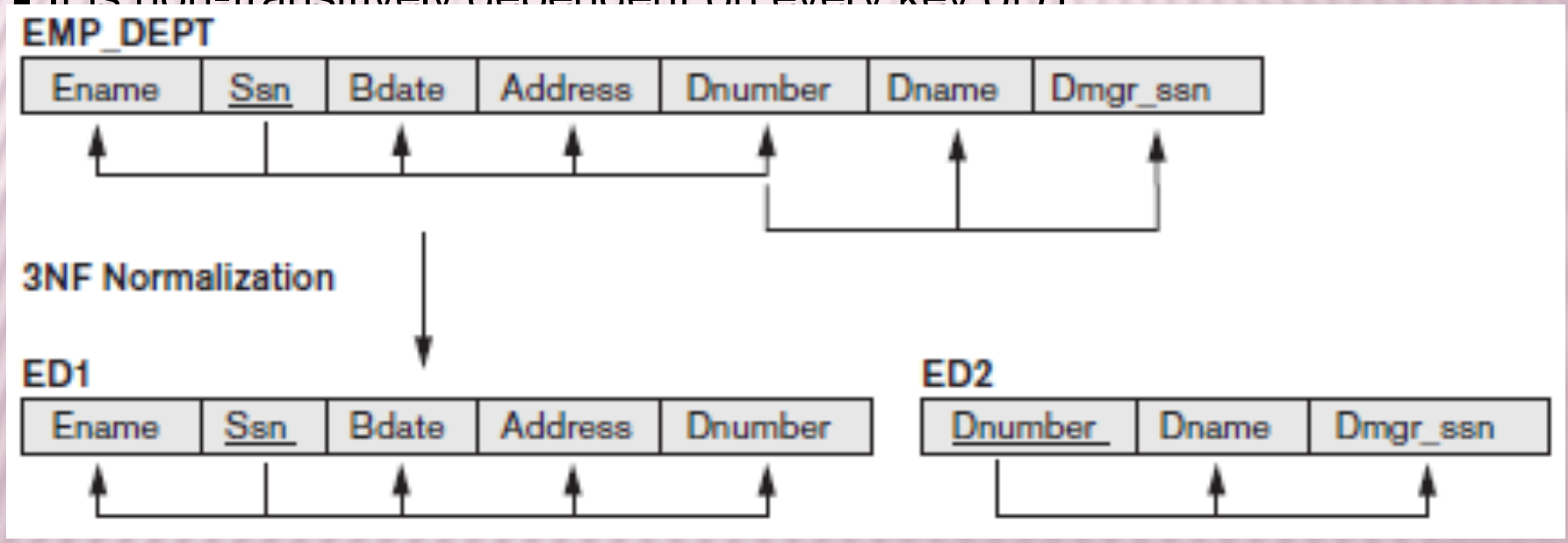
Third Normal Form

3NF Alternative 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.

OR

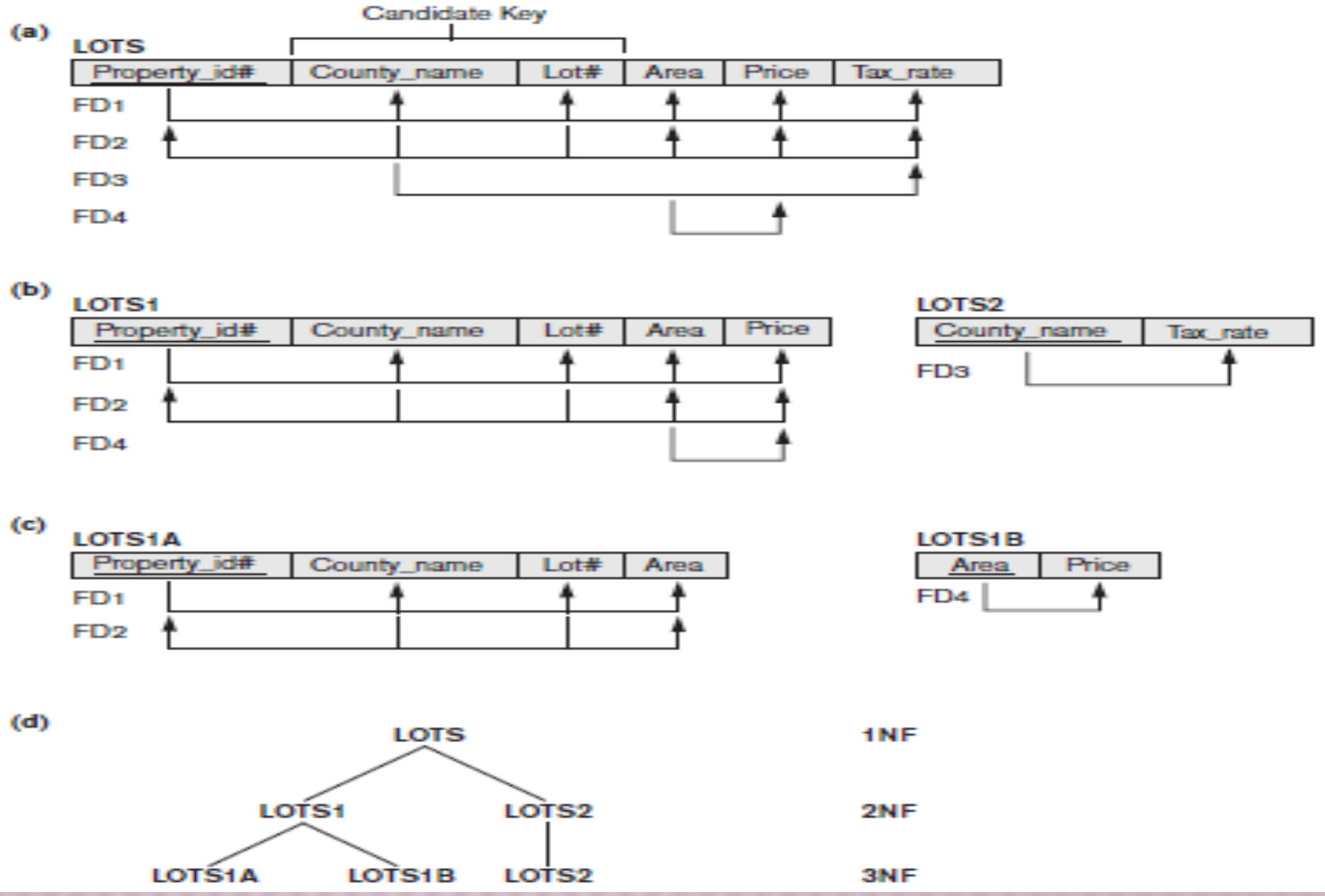
3NF Alternative Definition: A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

- It is fully functionally dependent on every key of *R.*
- It is non-transitively dependent on every key of *R*



Third Normal Form

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) Progressive normalization of LOTS into a 3NF design.

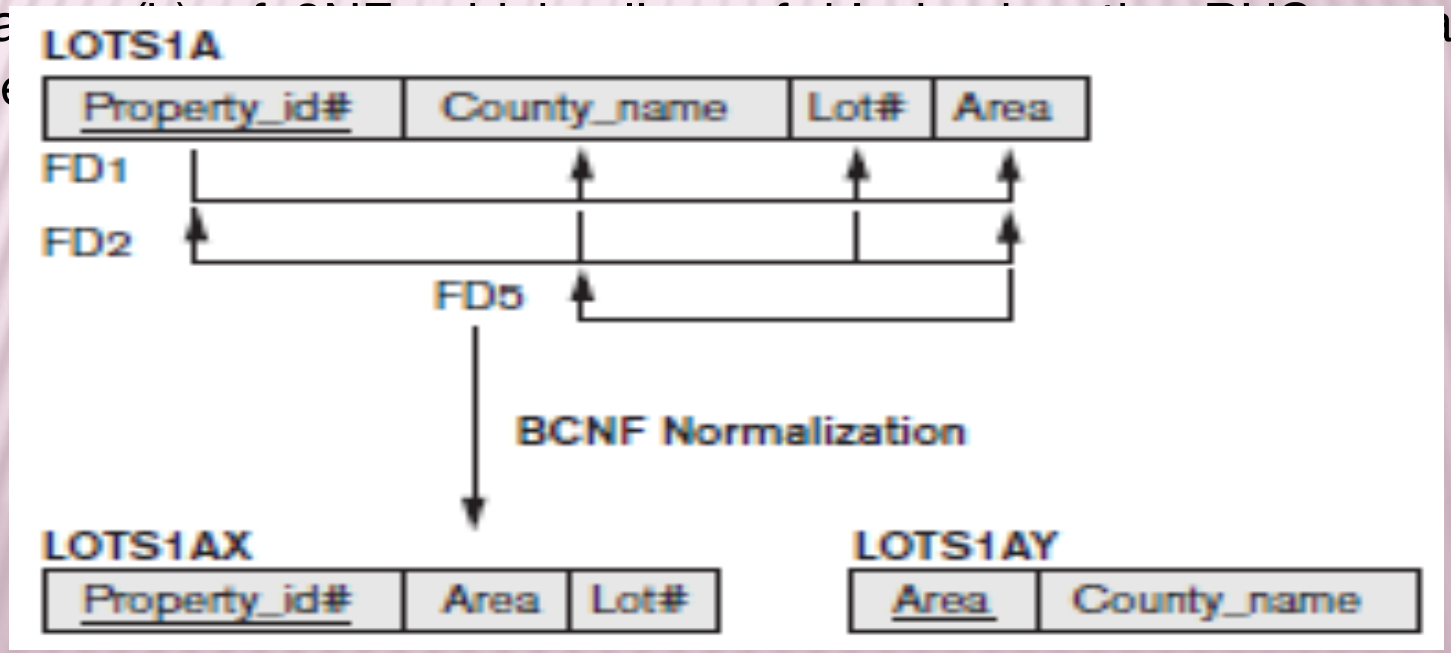


Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it
was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF,
however, a relation in 3NF is *not necessarily in BCNF.*

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 from the definition of 3NF in that cla prime attribute



Relation Decomposition & Insufficiency of Normal Forms

Consider a single **universal relation schema** $R = \{A1, A2, \ldots, An\}$ that includes *all* the attributes of the database. We implicitly make the **universal relation assumption**, which states that every attribute name is unique. The set F of functional dependencies that should hold on the attributes of R is specified by the database designers and is made available to the design algorithms. Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas $D = \{R1, R2, \ldots, Rm\}$ that will become the relational database schema; D is called a **decomposition** of R.

Attribute Preservation: We must make sure that each attribute in R will appear in at least one relation schema Ri in the decomposition so that no a s are *lost*; formally, we have

$$\bigcup_{i=1}^{m} R_i = R$$

This is called the **attribute preservation** condition of a decomposition.

Relation Decomposition & Insufficiency of Normal Forms

Dependency Preservation Property of a Decomposition: A functional dependency $X \rightarrow Y$ specified in *F(Set of FDs)* either appeared directly in one of the relation schemas R_i in the decomposition *D* or could be inferred from the dependencies that appear in some R_i. Informally, this is the *dependency preservation condition.*

Nonadditive (Lossless) Join Property of a Decomposition: Another property that a decomposition *D* should possess is the nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.

Definition of Lossless Join Property. Formally, a decomposition $D = \{R1, R2, \ldots, Rm\}$ of *R* has the **lossless (nonadditive) join property** with respect to the set of dependencies *F* on *R* if, for *every* relation state *r* of *R* that satisfies *F*, the following holds, where * is the NATURAL JOIN of all the relations in *D*

$*(\pi_{R1}(r), \ldots, \pi_{Rm}(r)) = r$.

The term *nonadditive join is used because if the property holds on a*

Multivalued Dependency and Fourth Normal Form

The Fourth Normal Form was defined based on the concept of ***multivalued dependency*** **(MVD)**. If we consider a relation having 2 or more independent attributes in it, we get into the problem of having to repeat every value of one of the attribute with every value of the other attribute to keep the relation state consistent & to maintain the independence among the attributes involved. This constraint is specified by MVD.

Formal Definition of Multivalued Dependency:

Definition. A multivalued dependency $X \rightarrow Y$ 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 $t1$ and $t2$ exist in r such that $t1[X] = t2[X]$, then two tuples $t3$ and $t4$ should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

- $t3[X] = t4[X] = t1[X] = t2[X]$
- $t3[Y] = t1[Y]$ and $t4[Y] = t2[Y]$
- $t3[Z] = t2[Z]$ and $t4[Z] = t1[Z]$

Whenever $X \twoheadrightarrow Y$ holds, we say that X **multidetermines** Y. Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R, so does $X \twoheadrightarrow Z$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$ and therefore it is sometimes written as

Multivalued Dependency and Fourth Normal Form

MVD specifies that given a particular value of x, the set of values of y determined by this value of x is completely determined by x alone & does not depend on the value of the remaining attributes z of R. Hence whenever 2 tuples exist that have distinct values of y but same values of x, these values of y must be repeatedin separate tuples with every distinct value of z that occurs with the same value of x.

Eg: Car_Sales

	MVD	MVD
Car#	Salesman#	Commission
1	1	2%
1	2	1.5%
1	3	1%
2	2	2%
3	3	2%
4	1	1%

Multivalued Dependency and Fourth Normal Form

We define F**ourth Normal Form (4NF)**, which is violated when a relation has undesirable multivalued dependencies and hence can be used to identify and decompose such relations.

4NF Definition. A relation schema *R* is in **4NF** with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every multivalued dependency $X \twoheadrightarrow Y$ in *F*, then either

- *x* is a superkey for *R*.
- y ⊆ x

Eg: Consider the EMP relation with the following attributes. It is PENDENTS.

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

Join Dependencies and Fifth Normal Form

If in the relation there are no MVD then we resort to another dependency called Join Dependency & if it is present, carry out a *multiway decomposition* into fifth normal form (5NF). It is important to note that such a dependency is a peculiar semantic constraint that is difficult to detect in practice; therefore, normalization into 5NF is rarely done in practice.

Definition. A **join dependency** (**JD**), denoted by JD(*R*1, *R*2, … , *Rn*), 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 lossless join decomposition into *R*1, *R*2, … , *Rn*. Hence, for every such *r* we have

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

Join Dependencies and Fifth Normal Form

Fifth Normal Form (5NF) Or **Project-Join Normal Form (PJNF):**
Definition. A relation schema *R* is in **Fifth Normal form (5NF)** (or **project-join normal form (PJNF)**) with respect to a set *F* of FDs, MVDs, and JDs if, for every join dependency JD(*R*1, *R*2, … , *Rn*) in *F*, every R_i is a super key of *R*.

Note: If a relation schema is in 3NF and each of its keys consists of a single attribute, it is also in 5NF.

Eg: Consider the SUPPLY relation & after decomposition we get R1, R2 &

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

R_1

Sname	Part name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part name	Proj name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX