

Unit III – Relational Database Design Theory

Chapter 07: Basics of Functional Dependencies and Normalizations for Relational Databases

Relational Database Design Approaches

- For most of the practical Relational Design projects, the designer uses one of the following two approaches:

1. Conceptual-to-Relational Model Mapping Approach

- Design a Conceptual Schema using a Conceptual Model (ER or EER Model) and then map the Conceptual Design into a set of relation schemas
 - Related attributes are grouped to form a relation schema by using Conceptual to Relational Model Mapping Algorithm
- Apply Normalization on relational database to improve design quality

2. External Knowledge of Existing System Based Approach

- Design a set of relations based on external knowledge derived from the existing implementation of files/Forms/Reports
 - attributes are derived from the existing manual system and grouped to form a relation schema by using the common sense/intuition of the database designer
- Apply Normalization on relational database to improve design quality

Measures of Relational Database Design Quality

- Following either of two design approaches, we need some formal measures of evaluating the relations for design quality, other than the intuition of the designer
- Relation schema may have good or bad design quality.
- Goodness of relation schemas can be defined at two levels:
 - Logical(conceptual) level : goodness at this level enables users to understand clearly the meaning of data in the relations and hence to formulate their query correctly
 - Implementation(storage) level: to understand how the tuples in a base relation are stored and updated.
- Functional Dependencies(FDs) and Normalization are formal concepts and theory used to measure the goodness and badness of the relational schemas more precisely.

Informal Design Guidelines for Relation Schemas

The following are four **informal measures of quality** for relation schema design.

1. Semantic of the attributes

2. Reducing the redundant values in tuples
3. Reducing the null values in tuples
4. Disallowing the possibility of generating spurious tuples

- Based on above informal measures, the four Informal Guidelines were proposed for Relational Database Design.

1. Semantics of the Relation Attributes

- Group the attributes that have certain real-world meaning & a proper interpretation associated with them to form a relation schema.
- Require careful conceptual design & systematic mapping into relations to obtain clear meaning
- If it is more Easier to explain the semantics of the relations, the relation schema design will be much better

GUIDELINE 1: Design a relation schema such 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 relationship types, it is straightforward to explain its meaning.

Otherwise, if the relation corresponds to a mixture of multiple entities & relationships, there will be semantic ambiguities and the relation cannot be easily explained.

- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.

e.g: **Good Design:**

EMPLOYEE(ename, ssn, bdate, address, dnumber)

DEPARTMENT(dname, dnumber, dmgrssn)

e.g: **Bad Design:**

EMP_DEPT(ename, ssn, bdate, address, dnumber, dname, dmgrssn)

Figure 14.1 Simplified version of the COMPANY relational database schema.

EMPLOYEE

ENAME	<u>SSN</u>	BDATE	ADDRESS	DNUMBER
-------	------------	-------	---------	---------

p.k.

DEPARTMENT

DNAME	<u>DNUMBER</u>	DMGRSSN
-------	----------------	---------

p.k.

DEPT_LOCATIONS

<u>DNUMBER</u>	<u>DLOCATION</u>
----------------	------------------

p.k.

PROJECT

PNAME	<u>PNUMBER</u>	PLOCATION	DNUM
-------	----------------	-----------	------

p.k.

WORKS_ON

<u>SSN</u>	<u>PNUMBER</u>	HOURS
------------	----------------	-------

p.k.

Redundant Information in Tuples & Update Anomalies

- **Mixing attributes of multiple entities** may cause redundant information problems

e.g: EMP_DEPT(ename, ssn, bdate, address, dnumber, dname, dmgrssn)

In this relation, the attribute values pertaining to a particular department(dnumber, dname, dmgrssn) are repeated for every employee who works for that department

- This redundant information in tuples further leads to cause the two main problems:
 1. Waste of storage space
 2. Problems of update anomalies:
 - Insertion anomalies
 - Deletion anomalies
 - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY (1)

Consider the relation:

EMP_DEPT(ename, ssn, bdate, address, dnumber, dname, dmgrssn)


- **Insert Anomaly:** Cannot insert a department unless an employee is assigned to. *Inversely* - Cannot insert an employee unless he/she is assigned to a department.
Only way is that insert nulls for the tuple of employee who does not work for a department yet
- **Delete Anomaly:** When a department is deleted, it will result in deleting all the employees who work on that department.
Alternately, if an employee is the sole employee in a department, deleting that employee would result in deleting the corresponding department
- **Modify Anomaly:** Changing the value of one of the attributes of a particular department (*say the Manager of department number 5*) may cause this update to be made for all employees working in that department. Otherwise, the database will become inconsistent.

2. Reducing Redundant Information in Tuples and Update Anomalies

GUIDELINE 2: Design a relation schema such that it does not suffer from the insertion, deletion and update anomalies.

If there still exist any, note it clearly and make sure that the programs that update the database will take care of it.

Null Values in Tuples

- A tuple will have NULL value for an attribute when that
 - ✓ Attribute value may exist but unknown/unavailable
 - ✓ Attribute is not applicable for a tuple
 - Relation will be have many null values if there exist many attributes that do not apply to all tuples.
 - Many null values in a relation will lead to the following problem:
 1. Wastage of storage space
 2. Problems with join operations at the logical (view) level.
 3. Problems with understanding the meaning of attributes (*not applicable/ may exist but unknown/unavailable*)
 4. Problem with what values to be consider for NULLs when performing aggregate operations such as sum(), count(), etc
-  **First two are the main criteria that determine whether to include the attributes that may have null in the relations**

3. Reducing Null Values in Tuples

GUIDELINE 3: Design a relation schema such that their tuples will have as few NULL values as possible

Attributes whose values may frequently be NULL could be placed in separate relations (with the primary key)

4. Generation of Spurious Tuples

- Bad designs for a relational database may generate erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

GUIDELINE 4: Design the relation schemas such that they can be joined with equality conditions on attributes that are either primary keys or foreign key which guarantees that no spurious tuples are generated.

Avoid relations that contain join attributes which are NOT (FK, PK) combinations, because joining on such attributes may produce spurious(invalid) tuples

Functional Dependencies (1)

- A functional dependency is a constraint between two sets of attributes of the relation that **should hold on all possible state** of the relation.
- This constraint is derived from the *meaning* and *inter-relationships* of the attribute values
- Functional dependency (FD) is used to specify *formal measures* of the "goodness" of relational designs

Functional Dependencies (2)

- Formal Definition of a functional dependency: let $R=(A_1, A_2, \dots, A_n)$ be a relation and X & Y are two sets of attributes that are subsets of R .

A functional dependency denoted by $X \rightarrow Y$ specifies a constraint on the possible tuples that can form a relation state r of R .

The constraint is that, for any two tuples t_1 & t_2 in $r(R)$ that have $t_1[X]=t_2[X]$, they must also have $t_1[Y]=t_2[Y]$.

- X *functionally determines* Y , if and only if the value of X uniquely determines a value of Y

Examples of FD constraints (1)

- Consider a relation schema

EMP_PROJ (ssn, pnumber, hours, ename, pname, plocation)

- From the semantics of the attributes, the following FDs should be hold:

➤ Ssn \rightarrow ename

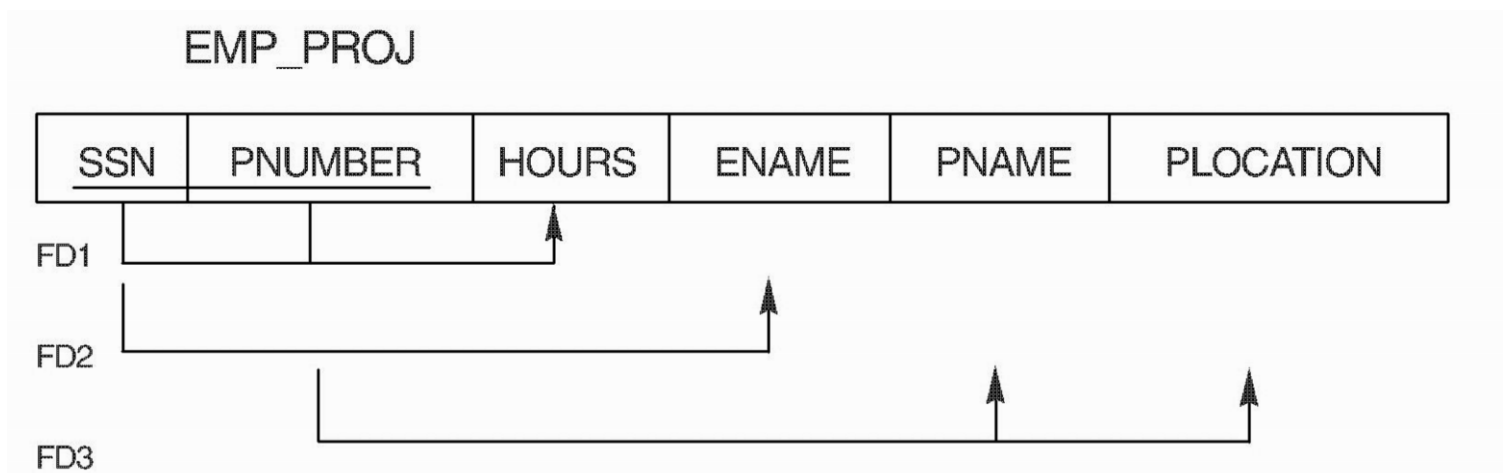
Specify that the value of employee's social security number(ssn) uniquely determines employee's name (ename)

➤ Pnumber \rightarrow {pname, plocation}

Specify that project number determines project name and location

➤ {ssn, pnumber} \rightarrow hours

Specify employee ssn and project number determines the hours per week that the employee works on the project



Example of FD constraints (2)

- ❖ A FD is a property of the semantics 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 $t1[K]=t2[K]$)

Inference Rules for FDs (1)

- ❖ Given a set of FDs F, additional FDs can be *inferred or deduced from F* that hold whenever the FDs in F hold. This is known as closure of F

For e.g: Consider a relation schema

EMP_DEPT (ename, ssn, bdate, address, dnumber, dname, dmgrssn)

- ❖ The obvious FDs on the schema are:
F = { $ssn \rightarrow \{ename, bdate, address, dnumber\}$,
 $dnumber \rightarrow \{dname, dmgrssn\}$ }

- ❖ Inferred FDs are:

$ssn \rightarrow \{dname, dmgrssn\}$

$ssn \rightarrow ssn$

$dnumber \rightarrow dname$

- ❖ An FD $X \rightarrow Y$ is inferred from a set of dependencies F specified on R , if $X \rightarrow Y$ holds in every $r(R)$.
- ❖ To determine a systematic way to infer new dependencies from a given set of FDs, a set of inference rules are used.

Armstrong's inference rules:

IR1 (**Reflexive**): $X \rightarrow X$ or If Y is *subset-of* X , then $X \rightarrow Y$

IR2 (**Augmentation**): If $X \rightarrow Y$, then $XZ \rightarrow YZ$

(Notation: XZ stands for $X \cup Z$)

IR3 (**Transitive**): If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- IR1, IR2, IR3 form a *sound* and *complete* set of inference rules Some **additional inference rules** that are useful:

IR4(**Decomposition**): If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

IR5(**Union**): If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

IR6(**Pseudo-transitivity**) : If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

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

Normalization for Relational Schema Design

1. Normalization of Relations
2. Practical Use of Normal Forms
3. Definitions of Keys and Attributes Participating in Keys
4. First Normal Form
5. Second Normal Form
6. Third Normal Form

Normalization of Relations (1)

Normalization: is the process of analyzing & decomposing a given relation into two or more relations based on their functional dependencies and primary keys to achieve the following desirable properties.

1. Minimizing redundancy
2. Minimizing update anomalies problems
3. Lossless join or nonadditive join property(ensure no spurious tuples is generated for join operations)
4. Dependency preservation property (ensure each FD is preserved in individual relations after decomposition)

Besides FDs & keys, the series of **normal form tests** are required to be carried out on individual relation schema to get normalized relations of any degree

Normal forms of Relations

- 1NF based on attribute value atomicity,
- 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)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting relational designs are of high quality and meet the desirable properties
- The database designers ***need not*** normalize to the highest possible normal form. (usually up to 3NF, BCNF or 4NF)
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are **hard to understand** or to **detect**

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 (1)

- A **superkey** of a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a set of attributes S subset-of R with the property that **no two tuples t_1 and t_2** in the valid relation state r of R will have $t_1[S] = t_2[S]$
- 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.
- 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(non-key) attribute** is not a prime attribute—that is, it is not a member of any candidate key.

First Normal Form (1NF)

- A relation is in 1NF if there is
 - no non-atomic attribute or nested relation in it.
 - no composite attributes & multivalued attributes
- For a non-1NF relation:

Decompose it to form new relations for each non-atomic attribute or nested relation

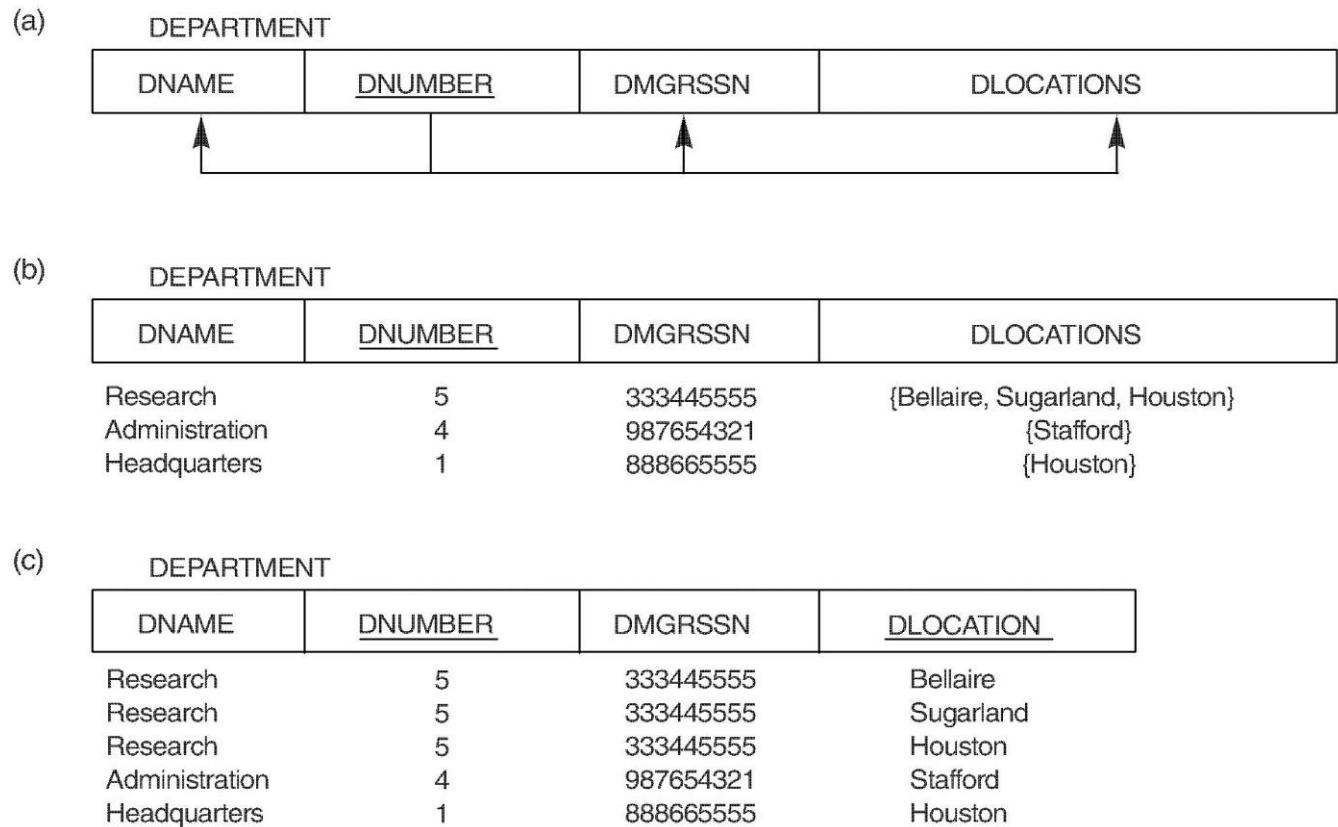
Example1:

DEPARTMENT(dnumber, dname, dmgrssn, dlocation)

- Assume that each department can have a number of locations.
- This relation is not in 1NF because dlocation is not an atomic attribute and it contains set of values.
- dnumber->dlocation because each set is considered a single member of the attribute domain

Figure 10.8 Normalization into 1NF

Figure 14.8 Normalization into 1NF. (a) Relation schema that is not in 1NF. (b) Example relation instance. (c) 1NF relation with redundancy.



First Normal Form(1NF)

DEPARTMENT(dnumber, dname, dmgrssn, dlocation)

There are **Three Techniques** to achieve first normal form for such relation:

(1) Replace the dlocation by attributes dloc1, dloc2, if a maximum number of values is known.

DEPARTMENT(dnumber, dname, dmgrssn, dloc1, dloc2, dloc3)

Drawback—introducing more null values

First Normal Form(1NF)

DEPARTMENT(dnumber, dname, dmgrssn, dlocation)

(2) Expand the key so that PK is combination of

(dnumber, dlocation)

DEPARTMENT(dnumber, dlocation, dname, dmgrssn)

Drawback—introducing redundancy data

First Normal Form(1NF)

DEPARTMENT(dnumber, dname, dmgrssn, dlocation)

(3) Remove the attribute dlocation that violates 1NF and place it in a separate relation DEPT_LOCATION along with PK dnumber of DEPARTMENT

This decomposes the non-1NF relation into two 1NF relations :

DEPARTMENT (dnumber, dname, dmgrssn)

DEPT_LOCATION(dnumber, dlocation)

Generally this technique is considered to be BEST OPTION

First Normal Form(1NF)

Example2: Consider a relation

EMP_PROJ(ssn, ename, pnumber, hours)

Is this in 1NF ?

If not why? Identify PK & partial key in the relation.

How to normalize this non-1NF relation into 1NF?

Normalization nested relations into 1NF

Figure 14.9 Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a “nested relation” PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposing EMP_PROJ into 1NF relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

(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, 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

(c)

EMP_PROJ1

<u>SSN</u>	ENAME
------------	-------

EMP_PROJ2

<u>SSN</u>	<u>PNUMBER</u>	HOURS
------------	----------------	-------

Second Normal Form (2NF)

- Uses the concepts of **FDs** & **primary key**
- **Full FD** - a FD $X \rightarrow Y$ where removal of any attribute from X means that the FD does not hold any more
- **Partial FD** - a FD $X \rightarrow Y$ where removal of some attributes from X means that the FD still hold

Examples:

EMP_PROJ(ssn, pnumber, hours, ename, pname, plocation)

- + {ssn, pnumber} \rightarrow hours is a full FD as neither ssn \rightarrow hours nor pnumber \rightarrow hours hold if remove one of the attribute from X
- + {ssn, pnumber} \rightarrow ename is *not* a full FD (called a *partial dependency*) as ssn \rightarrow ename also holds

Second Normal Form (2NF)

- A relation R is in 2NF if **every non-key attribute** A in R is **FULLY functionally dependent** on the primary key(PK) of R
- For relations where PK contains multiple attributes, **no non-key attribute** should be functionally **dependent on a part** of the PK
- For a **non-2NF** relation:
 - Decompose and form a new relation for each partial key with its dependent attribute(s)
 - Make sure to keep a relation with the original PK and any attributes that are fully functional dependent on it

Examples:

EMP_PROJ(ssn, pnumber, hours, ename, pname, plocation)

Full FDs in the relation are:

- ssn -> ename
- pnumber-> {pname, plocation}
- {ssn, pnumber} -> hours

Thus, decompose and form three 2NF relations:

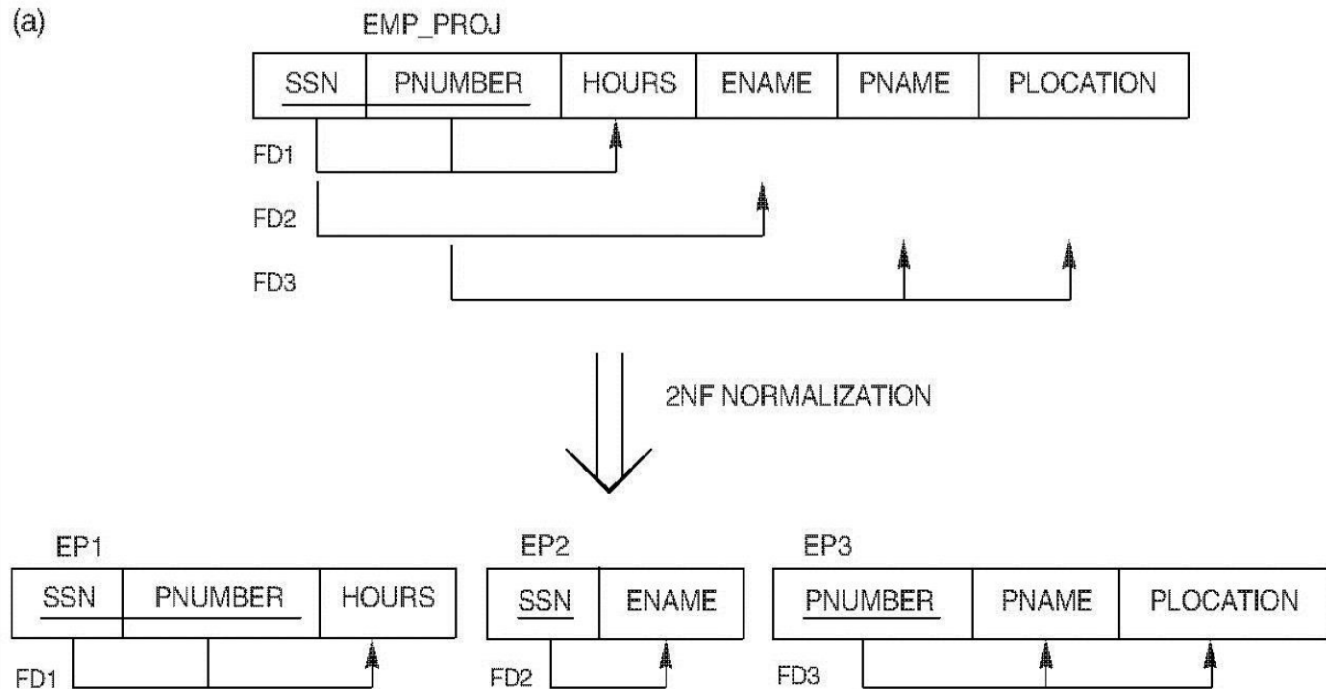
EP1(ssn, pnumber, hours)

EP2(ssn, ename)

EP3(pnumber, pname, plocation)

Normalization Process: Normalizing EMP_PROJ into 2NF Relations

(a)



Third Normal Form (3NF)

- ✚ 3NF is based on transitive FD
- ✚ Transitive functional dependency is a FD $X \rightarrow Z$ that can be derived from two FDs $X \rightarrow Y$ and $Y \rightarrow Z$

Examples:

EMP_DEPT (ename, ssn, bdate, address, dnumber, dname, dmgrssn)

- ✓ **ssn \rightarrow dmgrssn** is a *transitive* FD as
ssn \rightarrow dnumber and dnumber \rightarrow dmgrssn
hold
- ✓ **ssn \rightarrow ename** is *non-transitive* as there
no set of attributes X where ssn \rightarrow X
and X \rightarrow ename

Third Normal Form (3NF)

- ✚ A relation schema R is in **3NF** if it is in 2NF *and* **no non-key attribute A in R is transitively dependent on the primary key**
- ✚ Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes)
- ✚ For a non-3NF relation:
Decompose and form a relation that includes the non-key attribute(s) that functional determine(s) other non-key attribute(s)

Examples:

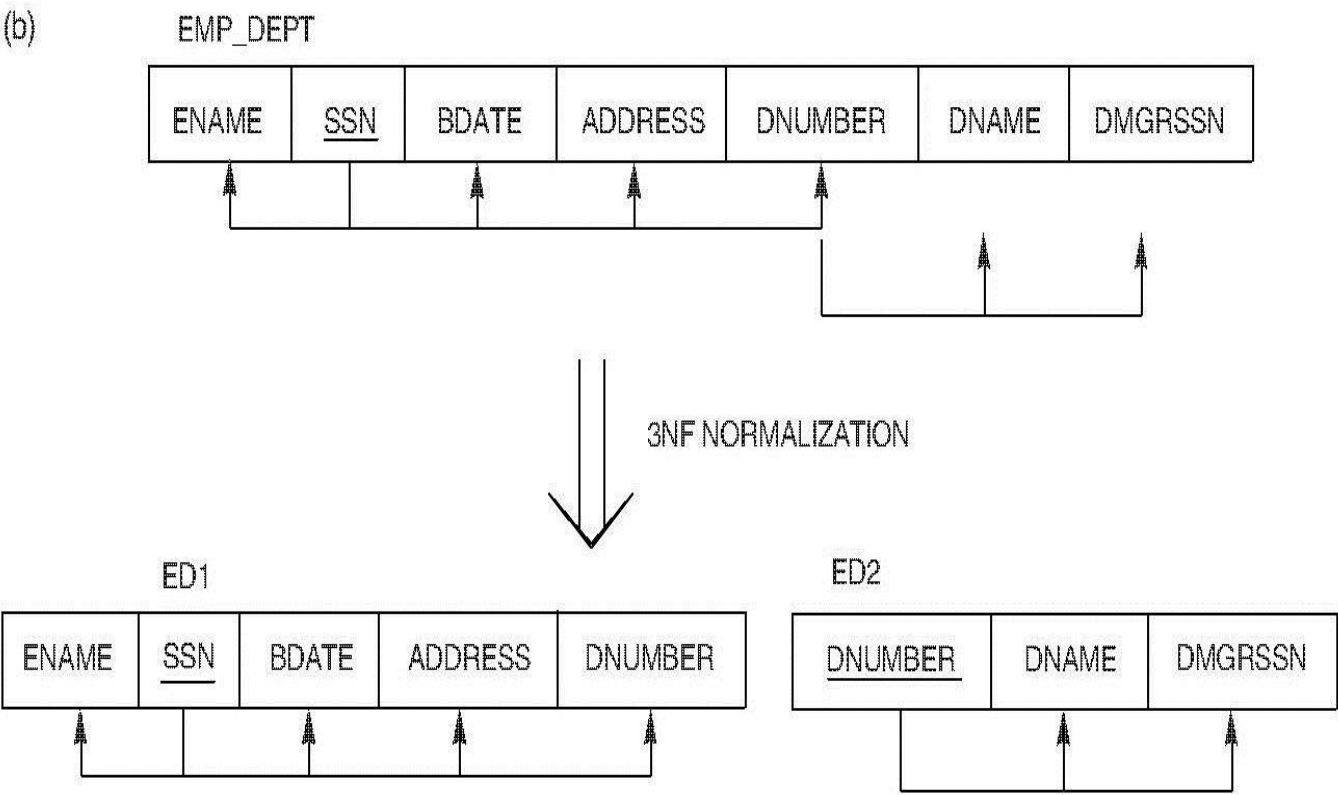
EMP_DEPT (ename, ssn, bdate, address, dnumber, dname, dmgrssn)

Normalize it into two 3NF relations:

ED1(ssn, ename, bdate, address, dnumber)

ED2(dnumber, dname, dmgrssn)

Normalization Process: Normalizing EMP_DEPT into 3NF Relations



General Normal Form Definitions

Definition:

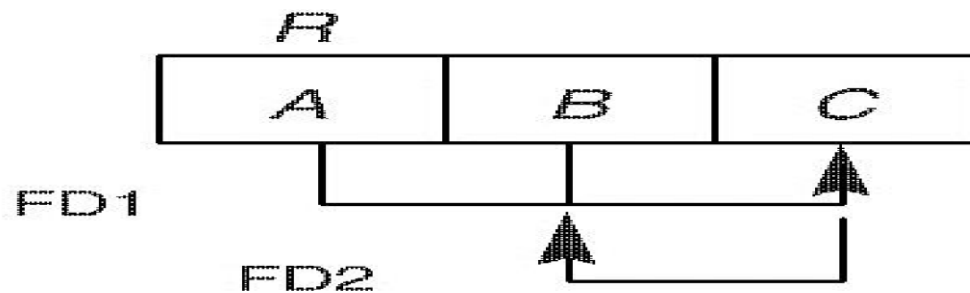
- **Superkey** of relation schema R is a set of attributes S of R that contains a key of R
- A relation schema R is in **3NF** if whenever a FD $X \rightarrow A$ holds in R, then either:
 - (a) X is a superkey of R, or
 - (b) A is a prime attribute of R

NOTE: Boyce-Codd Normal Form(BCNF) disallows condition (b)

BCNF(Boyce-Codd Normal Form)

- ✚ A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an FD $X \rightarrow A$ 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 is in 3NF

- ✚ The goal is to have each relation in BCNF (or 3NF)
- ✚ There exist relations that are in 3NF but not in BCNF
- ✚ There exist relations that are in 3NF but not in BCNF
- ✚ Example: consider a relation that FD has below



Relation R in above figure is in 3NF but not in BCNF