Schema Refinement and Normal Forms

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

What is Normalization?

- Normalization is the process of efficiently organizing data in a database.
- There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

- Database normalization is the process of restructuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. It was first proposed by Edgar F. Codd as an integral part of his relational model.

Functional Dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that $K \rightarrow R$
  - However, $K \rightarrow R$ does not require K to be *minimal*!

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*ssn*, *name, lot, rating, hrly_wages*, *hrs_worked*)
- *Notation*: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - *ssn* is the key: $S \longrightarrow SNLRWH$
  - *rating* determines *hrly_wages*: $R \longrightarrow W$

Example (Contd.)

- Problems due to R$\longrightarrow$W :
  - *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
  - *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
  - *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Will 2 smaller tables be better?

Wages

R	W
8	10
5	7

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* implies *ssn* $\rightarrow$ *lot*
- An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
  - F^+ = *closure of F* is the set of all FDs that are implied by *F*.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If X $\subseteq$ Y, then Y $\rightarrow$ X
  - *Augmentation*: If X $\rightarrow$ Y, then XZ $\rightarrow$ YZ for any Z
  - *Transitivity*: If X $\rightarrow$ Y and Y $\rightarrow$ Z, then X $\rightarrow$ Z
- These are *sound* and *complete* inference rules for FDs!

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from AA):
  - *Union*: If X $\rightarrow$ Y and X $\rightarrow$ Z, then X $\rightarrow$ YZ
  - *Decomposition*: If X $\rightarrow$ YZ, then X $\rightarrow$ Y and X $\rightarrow$ Z
- Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - C is the key: C $\rightarrow$ CSJDPQV
  - Project purchases each part using single contract: JP $\rightarrow$ C
  - Dept purchases at most one part from a supplier: SD $\rightarrow$ P
- JP $\rightarrow$ C, C $\rightarrow$ CSJDPQV imply JP $\rightarrow$ CSJDPQV
- SD $\rightarrow$ P implies SDJ $\rightarrow$ JP
- SDJ $\rightarrow$ JP, JP $\rightarrow$ CSJDPQV imply SDJ $\rightarrow$ CSJDPQV

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD *X* *Y* is in$\rightarrow$the closure of a set of FDs *F*. An efficient check:
  - Compute *attribute closure* of X (denoted $\longrightarrow$) wrt *F:* X^+
    - Set of all attributes A such that X A is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in
- Does F = {A $\longrightarrow$ B, B $\longrightarrow$ C, C D $\longrightarrow$ E } imply A $\longrightarrow$ E?
  - i.e, is A $\longrightarrow$ E in the closure X^+ ? F^+ Equivalently, is E in A^+ ?

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value!

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all X $\rightarrow$ A in F^+
  - A $\in$ X (called a *trivial* FD), or
  - X contains a key for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the

    A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all X $\rightarrow$A in F^+
  - A $\in$X (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- If 3NF violated by $X \rightarrow A$, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, $S \rightarrow C$, $C \rightarrow S$ is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Decomposition of a Relation Scheme

- Suppose that relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:
    - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
    - Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S $\rightarrow$ SNLRWH and R $\rightarrow$ W
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? (salary = W*H)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- *Tradeoff*: Must consider these issues vs. redundancy.

Lossless Join Decompositions

- Decomposition of R into X and Y is ***lossless-join*** w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

More on Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$
- In particular, the decomposition of R into UV and R - V is lossless-join if $U \rightarrow V$ holds over R.

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP $\rightarrow$ C and SD $\rightarrow$ P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP $\rightarrow$ C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem (3).)*
- *Projection of set of FDs F*: If R is decomposed into X, ... projection of F onto X (denoted F_X) is the set of FDs U V in F^+ (*closure of F*) such that U, V are in X.

$\rightarrow$

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is *dependency preserving* if $(F_X \text{ union } F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?????
- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
- And vice-versa! (Example?)

Decomposition into BCNF

- Consider relation R with FDs F. If X $\rightarrow$ Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, JP$\longrightarrow$ C, SD$\longrightarrow$ P, J $\longrightarrow$S
  - To deal with SD $\longrightarrow$P, decompose into SDP, CSJDQV.
  - To deal with J$\longrightarrow$ S, decompose CSJDQV into JS and CJDQV
- In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, CS $\rightarrow$ Z, Z $\rightarrow$ C
  - Can’t decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP $\rightarrow$ C, SD $\rightarrow$ P and J $\rightarrow$ S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If $X \longrightarrow Y$ is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \longrightarrow C$. What if we also have $J \longrightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

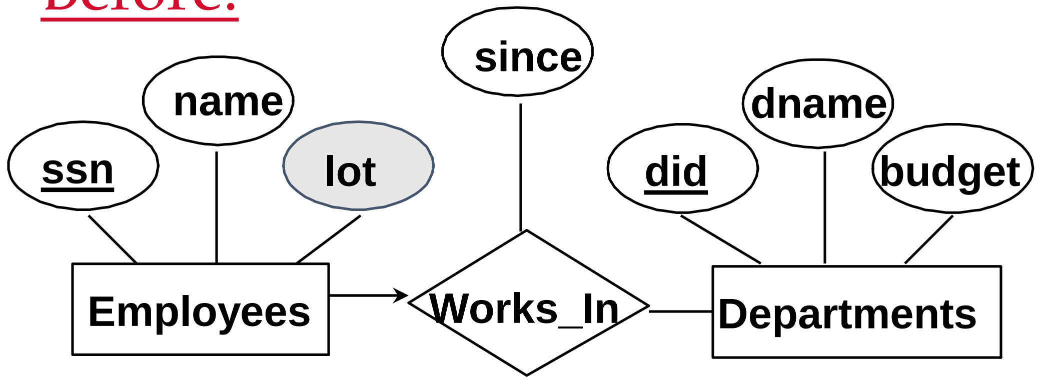
Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- e.g., A $\rightarrow$ B, ABCD $\rightarrow$ E, EF $\rightarrow$ GH, ACDF $\rightarrow$ EG has the following minimal cover:
  - A $\rightarrow$ B, ACD $\rightarrow$ E, EF $\rightarrow$ G and EF $\rightarrow$ H
- M.C. $\rightarrow$ Lossless-Join, Dep. Pres. Decomp!!! (in book)

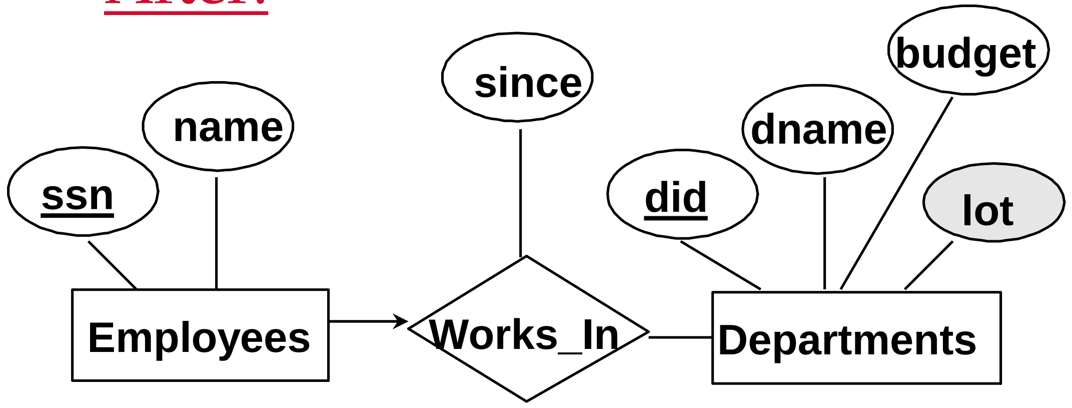
Refining an ER Diagram

- 1st diagram translated: Workers(S,N,L,D,S) Departments(D,M,B)
    - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot:
- $D \longrightarrow L$
- Redundancy; fixed by: Workers2(S,N,D,S) Dept_Lots(D,L)
- Can fine-tune this: Workers2(S,N,D,S) Departments(D,M,B,L)

Before:



After:



Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
    - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
    - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.

Additional Information on Normalization with examples

Normalization
for Relational Databases

Informal Design Guidelines for Relation Schemas

- Four informal methods are considered for relation schema design.
  - Semantics of the attributes
  - Reducing the redundant values in tuples.
  - Reducing the Null values in tuples.
  - Disallowing the possibility of generating spurious tuples.

Semantics of the relation attributes

- Semantics specify how to interpret the attribute values stored in a tuple of the relation.

- For a better relation schema design, semantics of the relation should be easier to understand.

A Company Database

EMPLOYEE

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

DEPARTMENT

DNAME	DNUMBER	DMGRSSN

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

SSN	PNUMBER	HOURS

Guideline 1

- Do not combine attributes from multiple entity types and relationship types into a single relation.

- A relation schema corresponds to one entity type or one relationship type, it is straight forward to explain its meaning.

Poor Design

EMPL_DEPT

SSN	ENAME	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

Redundant Information in Tuples

- Good schema design minimizes the storage space used by the base tables.

- Grouping attributes into relation schema has a significant effect on storage space.

Update anomalies

- These can be classified into following:

- Insertion Anomalies
- Deletion Anomalies
- Modification (Update) Anomalies

Insertion Anomalies

- It may not be possible to store certain information unless some other, unrelated information is stored also.
- e.g.

1. To insert a new employee tuple into EMP_DEPT, we must include attribute values for the department that the employee works for, or NULLS.
2. Attributes of a new department can not be inserted if it has no employee yet or NULLS have to be inserted for attributes of employee.

Poor Design

EMPL_DEPT

SSN	ENAME	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
1	Ram	12-01-88	Sfsdgfdhb	1	MCA	1
2	Shyam	12-02-88	Dsgnhfgfn	5	CS	2
3	Rita	22-01-88	Nhgngfngf	4	IS	4
4	Lata	12-01-89	dnhgngfn	1	MCA	1
5	Mina	17-01-82	Hvjkjskkl	1	MCA	1

Deletion Anomalies

- It may not be possible to delete certain information without losing some other, unrelated information is also deleted.
- e.g.

  If an employee tuple has to be deleted which happens to be last employee in a department, that department information will also be lost.

Modification Anomalies

- If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

- E.g.
- If we change any attribute of the department, all the employee tuples should also be modified otherwise inconsistency will occur.

Guideline 2

- Design the base relation schemas so that no insertion, deletion or modification anomalies are present in the relations.
- These guidelines sometimes have to be violated to improve the query performance.
- If any anomalies are present, it should be handled by programs.

- In general, Anomaly free base relations should be used.

- Views can be used to have such query where attributes from different relations can be displayed.

NULL values in Tuples

- Many NULLS can waste space at the storage level.

- Understanding about the attribute value and with specifying JOIN operation will be less.

- Accounting NULLS in aggregate operations such as COUNT or SUM is difficult.

- NULLS can have multiple interpretations:
  - The attribute does not apply to that particular tuple.
  - The attribute value for this tuple is unknown.
  - The value is known but not provided yet.
- But representation for all above is same i.e. NULL.

Guideline 3

- Avoid placing attributes in a base relation whose value may frequently be NULL.
- If NULL is unavoidable, they should be applied in exceptional cases.
- Using space efficiency and avoiding joins are two criteria to have such attributes which may have NULL values for many tuples.

- E.g.
- If only 10% employees have individual offices, it is not preferable to have attribute OFFICE_ADDRESS in employee relation because 90% employees will have NULL values for that attribute.

- A relation EMP_OFFICE can be created having attribute SSN & OFFICE_ADDRESS.

Generation of Spurious Tuples

- Additional tuples due to join operation will be generated which are wrong or have invalid information; these are called Spurious tuples.

Guideline 4

- Design relation schema so that they can be joined with equality condition on attributes that are either Primary keys or foreign keys and guarantee no spurious tuples are generated.

- Avoid relations that contain matching attributes which are not Primary keys or foreign keys combinations, because joining of such attributes may generate spurious tuples.

Design Guidelines: Summary

1. Do not combine attributes from multiple entity types relationship type into a single relation.

2. Anomalies that cause redundant work to be done during insertion into and modification of a relation, and that may cause accidental loss of information during a deletion from a relation.

Design Guidelines: Summary

3. Waste of storage space due to nulls and difficulty of performing aggregation operations and joins due to null values.

4. Generation of invalid and spurious data during joins on improperly related base relations.

Functional Dependencies

- Functional Dependency is a constraint between two sets of attributes form the relation.
- **Definition:**

  A **functional dependency,** denoted by

  X $\rightarrow$ **Y,** between two sets of attributes **X** and **Y** that are subsets of **R** specifies a **constraints** 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**] .

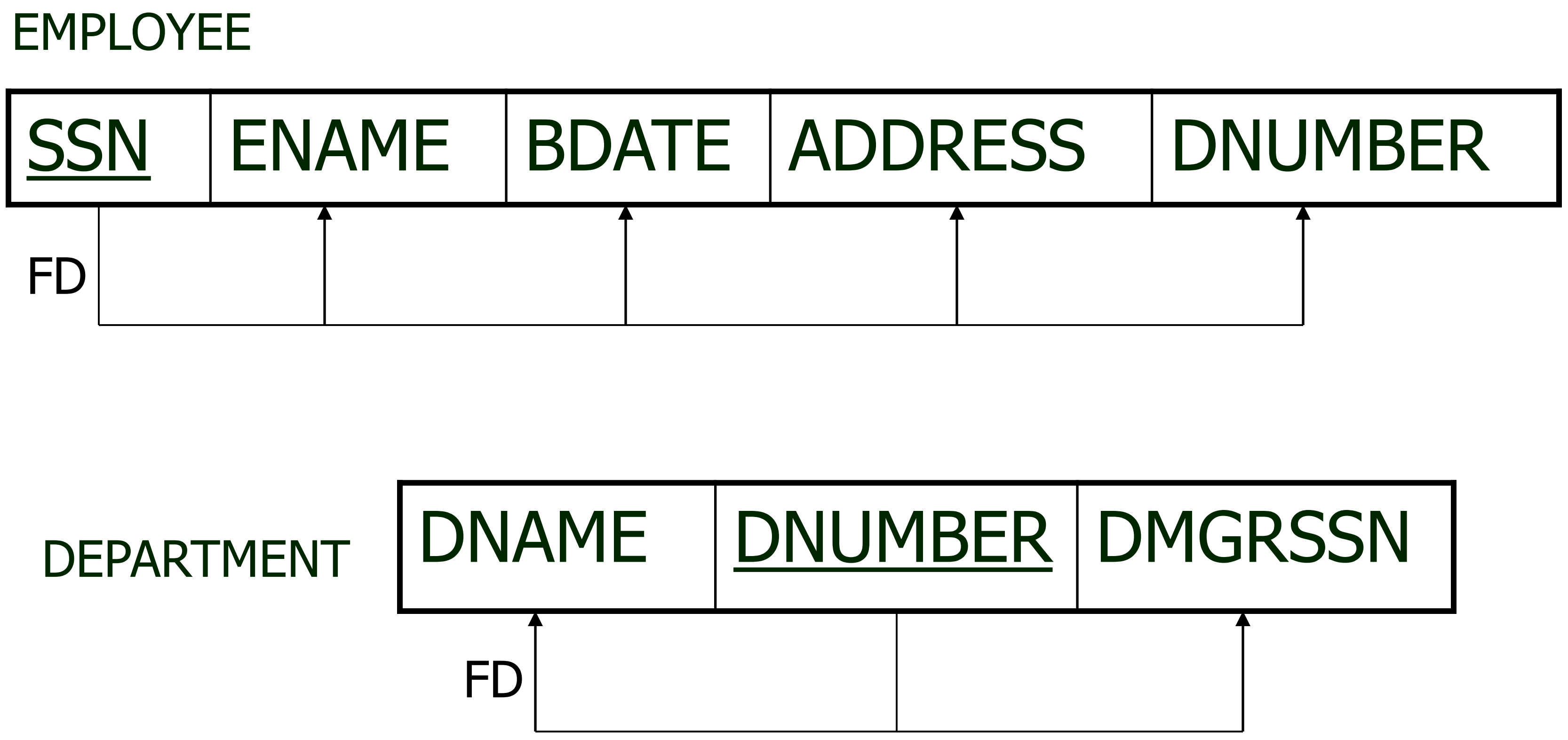
- There is a functional dependency from **X** to y, i.e. the value of **y** is determined by the value of **X,** or **X** determines the value of **Y.**

- 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.**

e.g.

- SSN $\rightarrow$ ENAME
- DNUMBER $\rightarrow$ DNAME
- {SSN, PNUM} $\rightarrow$ HOURS


EMPLOYEE
SSN
ENAME
BDATE
ADDRESS
DNUMBER
FD
DEPARTMENT
DNAME
DNUMBER
DMGRSSN
FD

- A functional dependency is the property of relation schema not of a particular state of R.

- Thus functional dependency is true for all the possible states of the relation.

NORMAL FORMS BASED ON PRIMARY KEYS

Most practical relational design projects take one of the following two approaches:

- First perform a conceptual schema design using a conceptual model such as ER or EER and then map the conceptual design into a set of relations.
- Design the relations based on external knowledge derived from an existing implementation of files or forms or reports.

- Evaluate the relations for goodness and decompose them further as needed to achieve higher normal forms using normalization theory.

Normalization of Relations

- Three normal forms were proposed by Codd (1972).
- **Normalization of data** can be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties as follows:

1. Minimizing redundancy
2. Minimizing the insertion , deletion , and update anomalies

- If Unsatisfactory relation schemas don’t meet certain conditions i.e. the **normal form test,** these are decomposed into smaller relation schemas that meet the tests and hence posses the desirable properties .

- Thus, the normalization procedure provides database designers with the following :

1. A formal frame work for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
2. A series of normal form tests that can be carries out on individual relation schemas so that relational database san be **normalized** to any desired degree

- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas (together) should possess .

- These would include two properties:
- The **lossless join** or **nonadditive 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** : ensures that each functional dependency is represented in some individual relation resulting after decomposition.

Practical Use of Normal Forms

- The process of storing the join of higher normal form relations as a base relation – which is in a lower normal form – is known as denormalization.

- Database design in industry today pays particular attention to normalization only up to 3NF, BCNF, 4NF.

- Sometimes relations may be left in a lower normalization status, such as 2NF, for performance reasons.

Definitions of keys and Attributes Participating in keys

- Superkey:
  - A superkey of a relation schema R= {A1, A2 …. An} is a set of attributes S   R with a property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- Key (minimal superkey):
  - A key K is a superkey with additional property that removal of any attribute from K will cause K not to be a superkey

- Candidate key:
  - If a relation schema has more than one key each is called a candidate key. One of the candidate key is designated as primary key, and others are called Secondary keys.

- 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 prime attribute – that is ,if it is not a member of any candidate key.

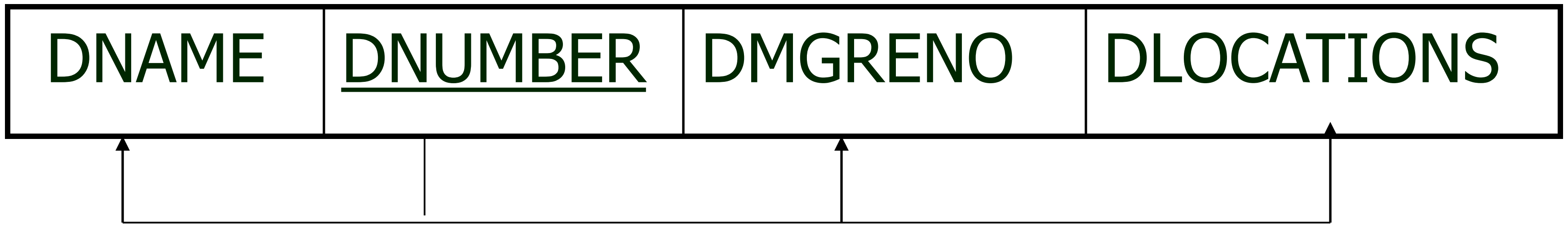
FIRST NORMAL FORM

- It states that the domain of an attribute must include only *atomic* (simple , indivisible) *values.*

- The value of any attribute in a tuple must be a single value from the domain of that attribute.

- Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.

A relation schema which is not in 1 NF



DNAME	DNUMBER	DMGRENO	DLOCATIONS
MCA	1	22222	Delhi, Pune, Bangalore
CS	2	11111	Bangalore
IS	3	33333	Hyderabad

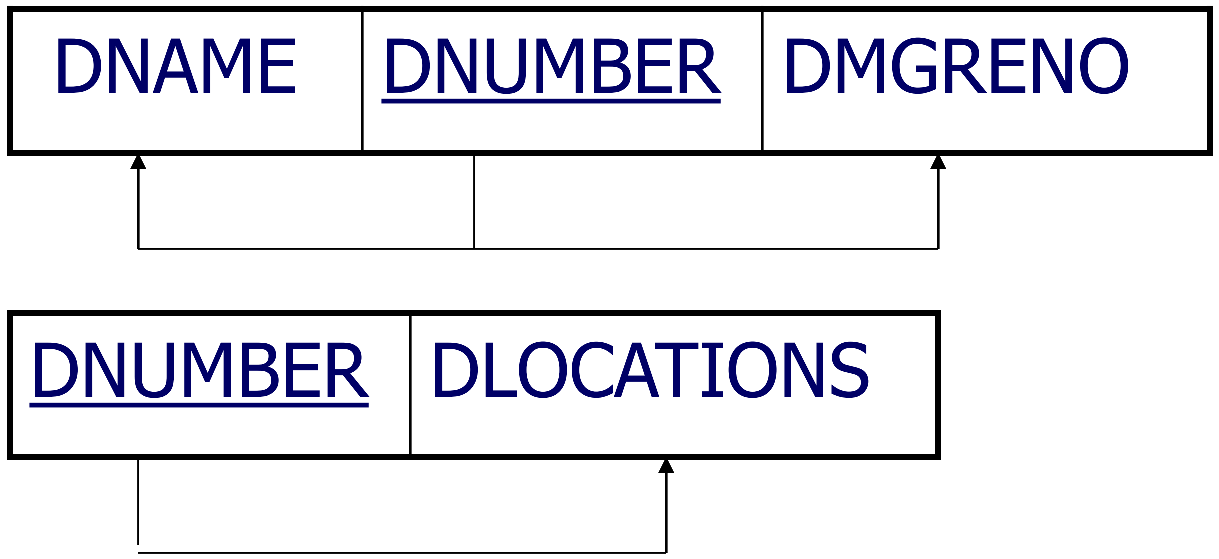
- The domain of DLOCATIONS contain sets of values and hence is nonatomic, it is not functionally dependent on DNUMBER.

- In this case DNUMBER →DLOCATIONS , because each set is considered a single member of the attribute domain.

- There are three main techniques to achieve first normal form :

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 relation **DEPT_LOCATIONS** is the combination {DNUMBER, DLOCATION}.



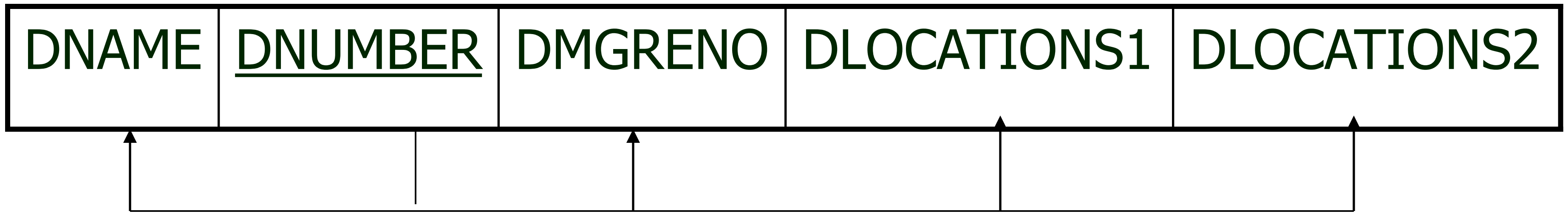
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT for each location of a DEPARTMENT.

- Then Primary Key becomes {DNUMBER, DLOCATION} and redundancy is introduced.

1NF version of same relation with redundancy

DNAME	DNUMBER	DMGRENO	DLOCATIONS
MCA	1	22222	Delhi
CS	2	11111	Bangalore
IS	3	33333	Hyderabad
MCA	1	22222	Pune
MCA	1	22222	Bangalore

- If a maximum number of values is known for the attributes:
- for example , if it is known that at most three locations can exist for a department
- replace the DLOCATIONS attribute by two atomic attributes :
  - DLOCATION1
  - DLOCATION2
- It leads to introduction of more NULL values


DNAME
DNUMBER
DMGRENO
DLOCATIONS1
DLOCATIONS2

- First is generally considered best.

- First normal form also disallows multivalued attributes that are themselves composite, which are called **nested relations** because each tuple can have a relation within it.

Normalizing nested relations into 1NF

<u>SSN</u>	ENAME	PROJS	
		PNUMBER	HOURS

Relation with nested relation

<u>SSN</u>	ENAME	<u>PNUMBER</u>	HOURS
11111	Ram	1	78
		2	87
22222	Mina	3	45
		2	54

1 NF

EMPLOEE

SSN	ENAME

Primary Key: SSN

EMP_PRJ

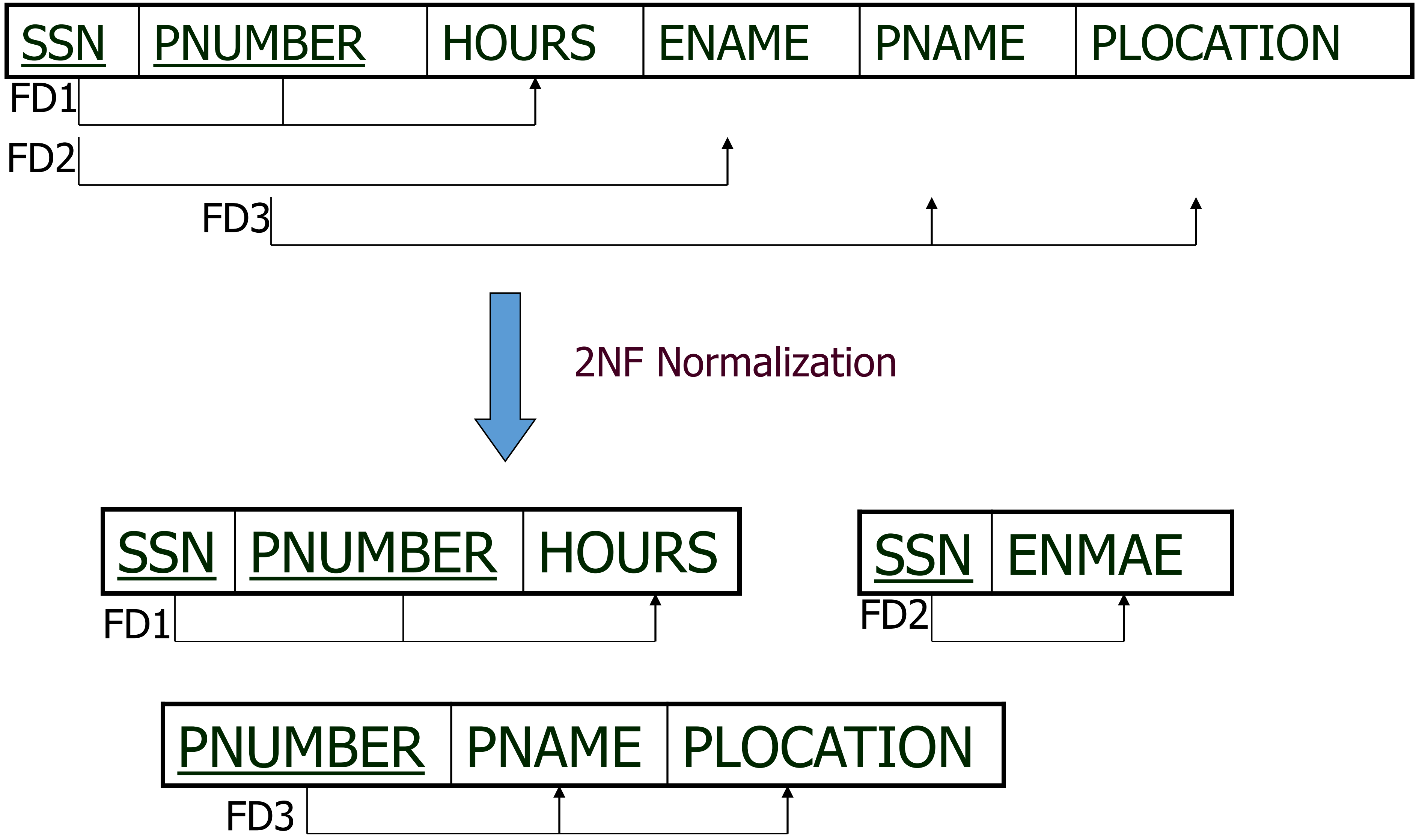
SSN	PNUMBER	HOURS

Primary Key: {SSN, PNUMBER}

Second Normal From

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

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


SSN
PNUMBER
HOURS
ENAME
PNAME
PLOCATION
FD1
FD2
FD3
2NF Normalization
SSN
PNUMBER
HOURS
FD1
SSN
ENMAE
FD2
PNUMBER
PNAME
PLOCATION
FD3

- A functional dependency X $\rightarrow$ Y is a full functional dependency, if removal of any attribute A from X means dependency does not hold any more.

- For any attribute A Є X, (X – {A}) does not functionally determine Y.

- (X – {A}) $\rightarrow$ Y is partial dependency

- The test for 2NF involves testing for functional dependencies those left side attributes are part of the Primary Key.

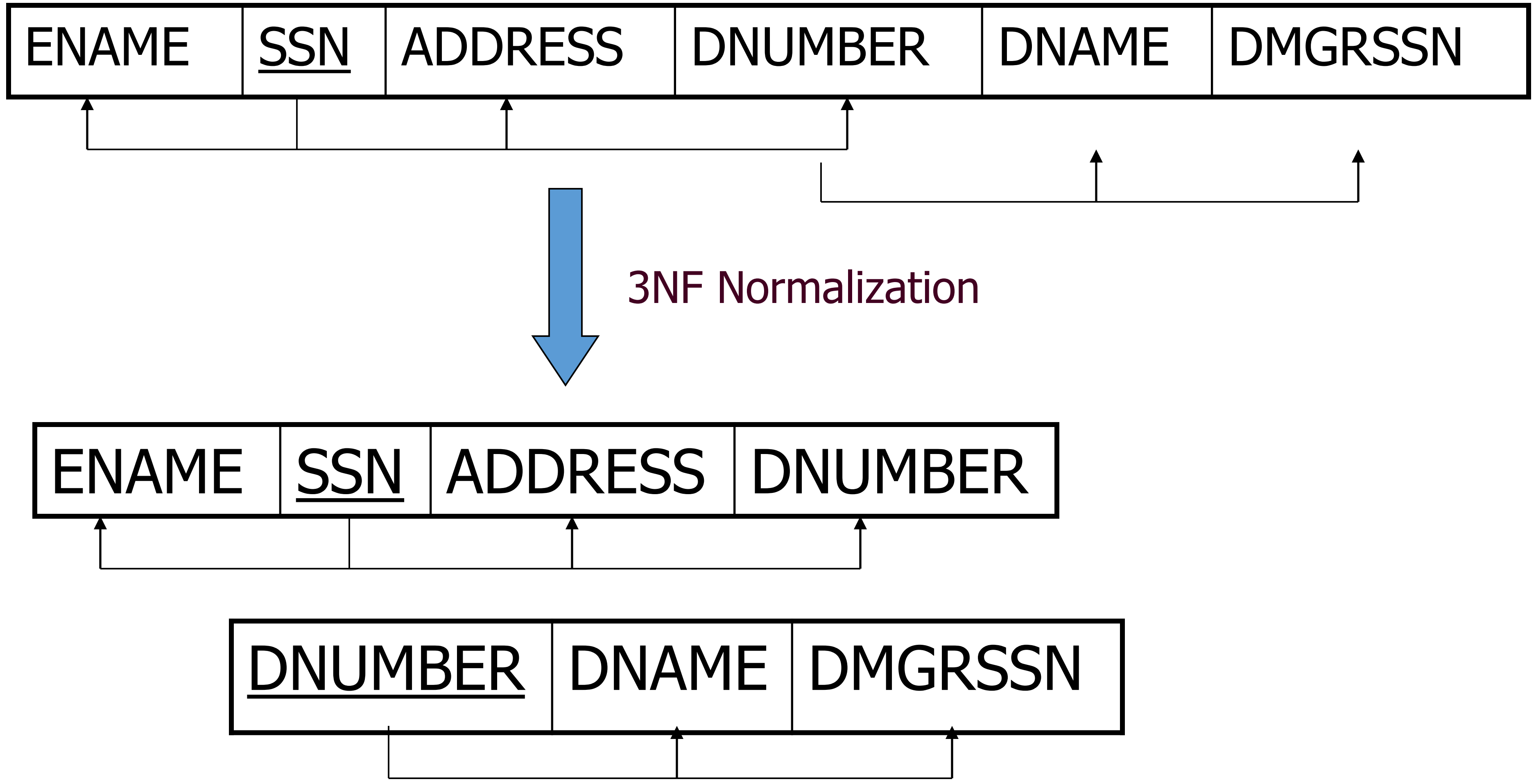
- If Primary Key contains single attribute, the test need not be applied.

Third normal form(3NF)

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

- A functional dependency $X \rightarrow Y$ in a relation schema R is **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 $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

- **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.


ENAME
SSN
ADDRESS
DNUMBER
DNAME
DMGRSSN
3NF Normalization
ENAME
SSN
ADDRESS
DNUMBER
DNUMBER
DNAME
DMGRSSN

GENERAL DEFINITIONS OF SECOND AND THIRD NORMAL FORMS

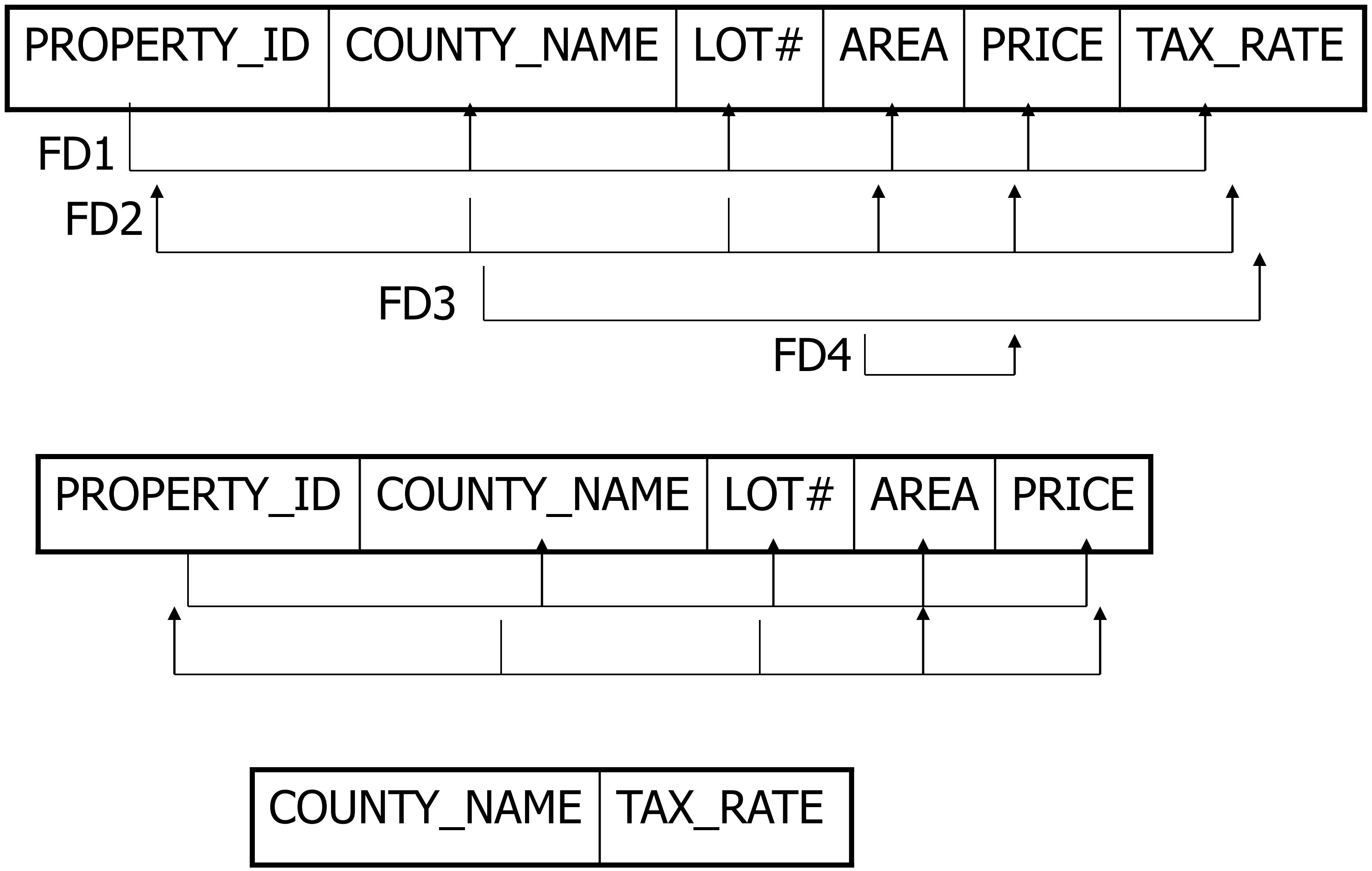
NORMAL FORM	**TEST**	**REMEDY(NORMALIZATION)**
First (1NF)	Relation should have no nonatomic attributes or nested relations	Form new relations for each nonatomic attribute of nested relation
Second (2NF)	For relations where primary key contains multiple attributes , no nonkey attribute should be functionally 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 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 include the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

General Definition of second Normal Form

- 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.

General definition of Third Normal Form

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


PROPERTY_ID
COUNTY_NAME
LOT#
AREA
PRICE
TAX_RATE
FD1
FD2
FD3
FD4
PROPERTY_ID
COUNTY_NAME
LOT#
AREA
PRICE
COUNTY_NAME
TAX_RATE

PROPERTY_ID	COUNTY_NAME	LOT#	AREA	PRICE	TAX_RATE