Functional Dependencies and Normalization for Relational Databases

Handout 5

Outline(contd.)

- 3 Normal Forms Based on Primary Keys
  - 3.1 Normalization of Relations
  - 3.2 Practical Use of Normal Forms
  - 3.3 Definitions of Keys and Attributes Participating in Keys
  - 3.4 First Normal Form
  - 3.5 Second Normal Form
  - 3.6 Third Normal Form
- 4 General Normal Form Definitions (For <u>Multiple</u> Keys)
- 5 BCNF (Boyce-Codd Normal Form)

Design Guidelines for Relational Databases

- What is relational database design?

  The grouping of attributes to form "good" relation schemas
- A ***Goal*** of relation schema design is to minimize the ***storage space*** used
- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

Design Guidelines for Relational Databases

- We first discuss informal guidelines for good relational design
- Then we discuss formal concepts of functional dependencies and normal forms

  - 1NF (First Normal Form)

  - 2NF (Second Normal Form)

  - 3NF (Third Normal Form)

  - BCNF (Boyce-Codd Normal Form)

Design Guidelines for Relational Databases

1 Informal Design Guidelines for Relational Databases
- 1.1 Semantics of the Relation Attributes
- 1.2 Redundant Information in Tuples and Update Anomalies
- 1.3 Null Values in Tuples
- 1.4 Spurious Tuples

1.1 Semantics of the Relation Attributes

GUIDELINE 1: Design a relational schema so that it is easy to explain its meaning

- The semantics (meaning) of attributes should have real world meaning.
- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be mixed that are used to refer to other entities

1.1 Semantics of the Relation Attributes

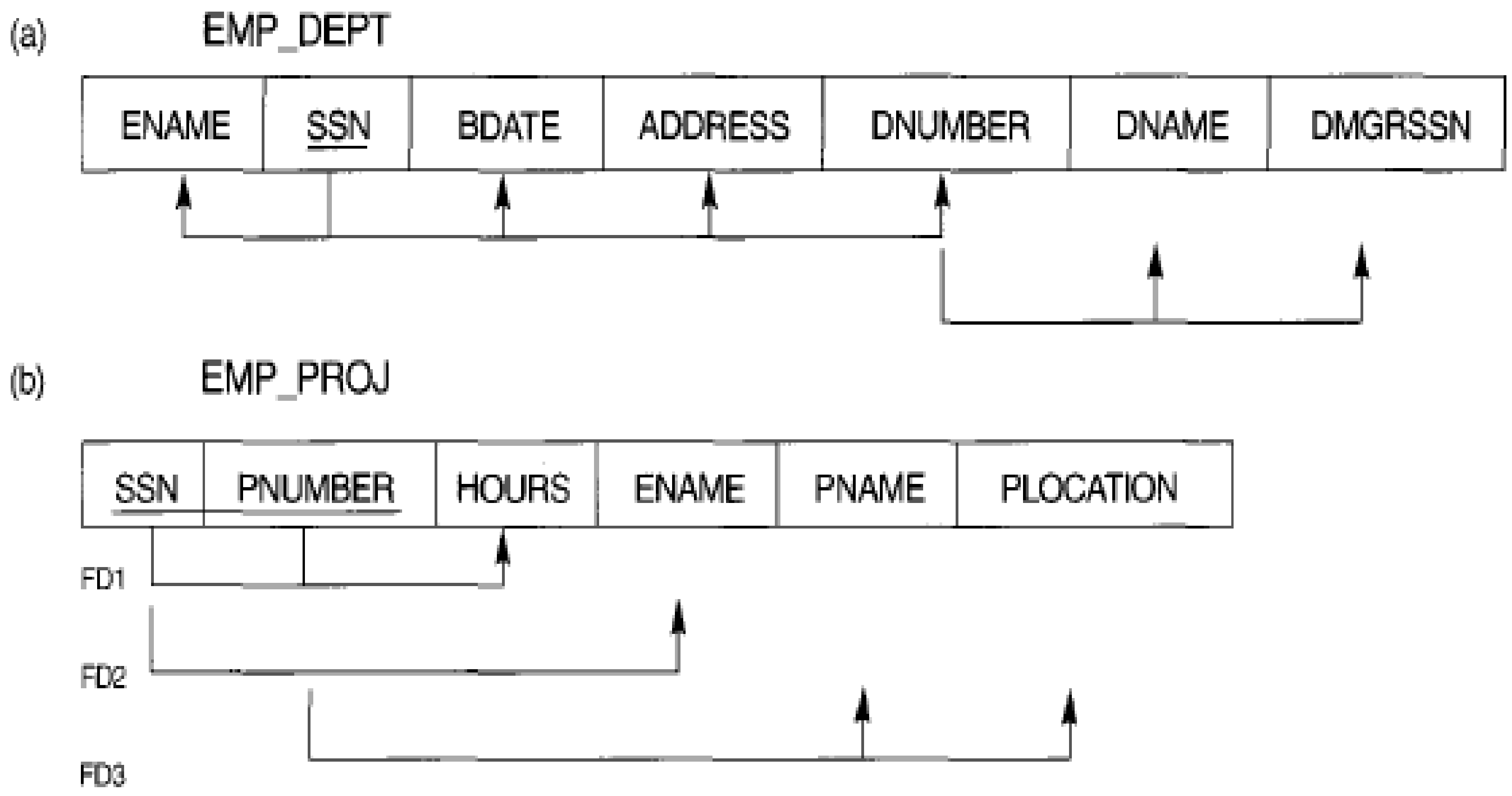


Figure 1

1.1 Semantics of the Relation Attributes

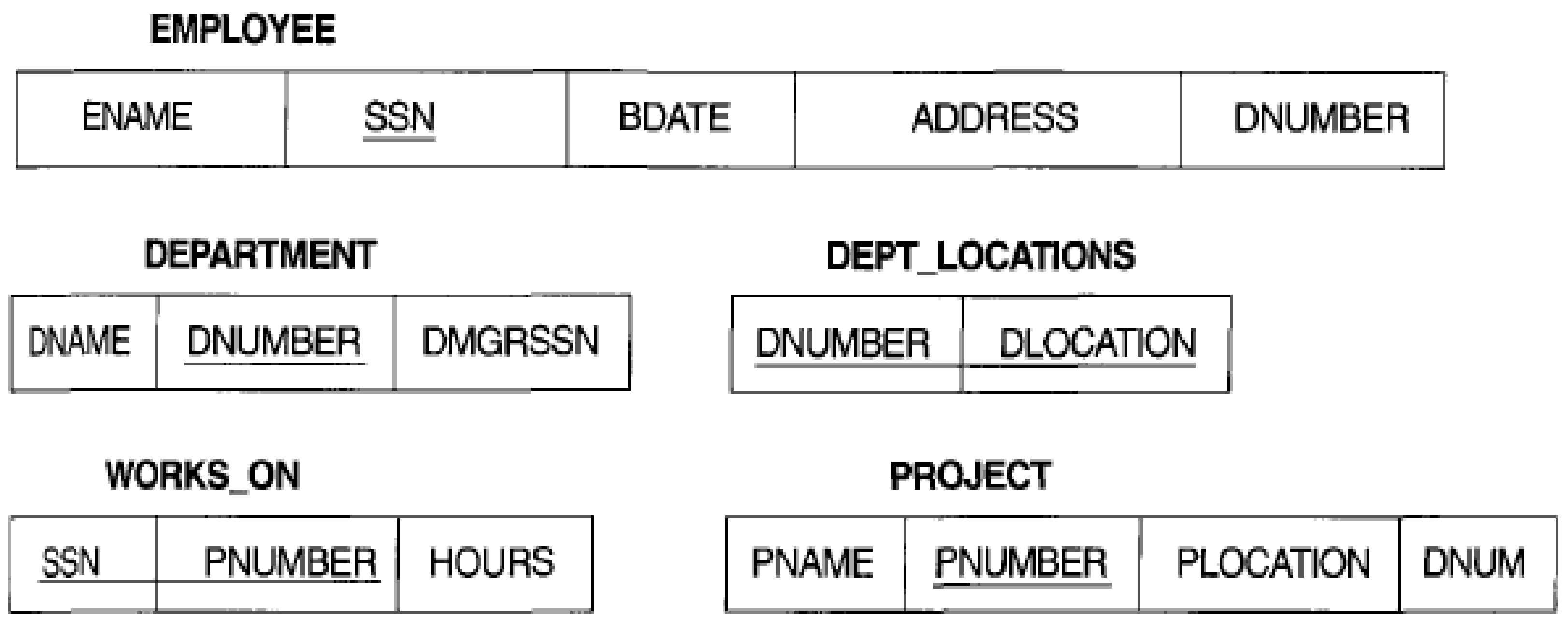


Figure 2

1.2 Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems

  1. Information is stored redundantly wasting

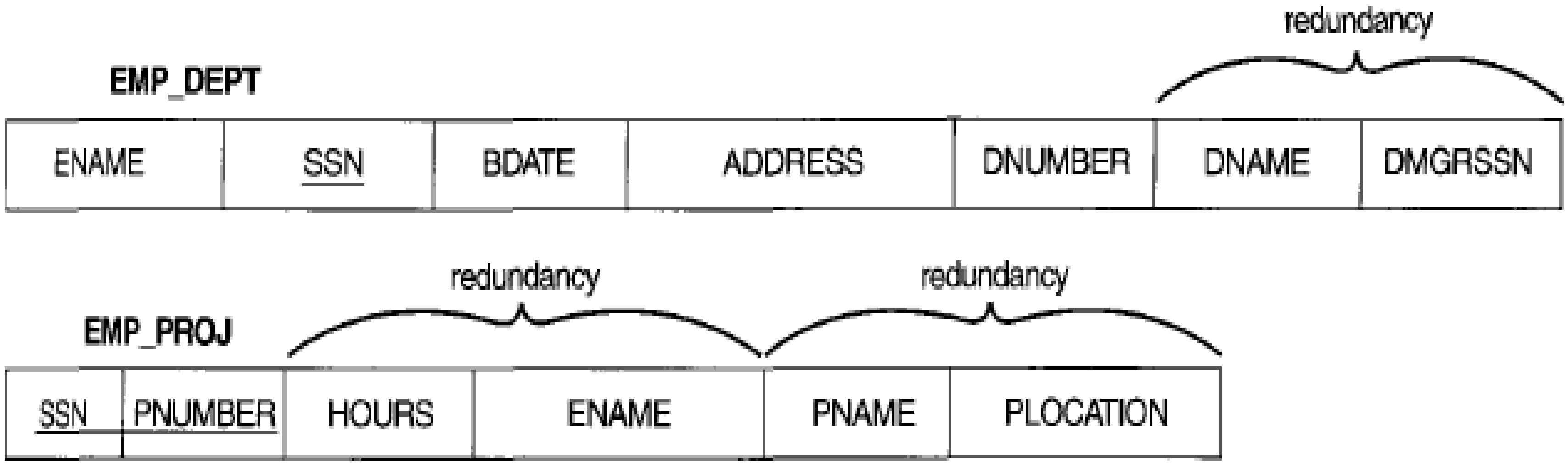


Figure 1

1.2 Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems

  2. Problems with update anomalies
  - Modification anomalies
  - Insertion anomalies
  - Deletion anomalies

Example of Modification Anomaly

Consider the relation:

EMP_PROJ (<u>Emp#, Proj#,</u> Ename, Pname, No_hours)

- **Modification Anomaly:** Changing the name of project number P1 from “Billing” to “Accounting” may cause this update to be made for all employees working on project P1.
- If fail to do so create an inconsistency of data (wrong).

Deletion Anomaly

- Loosing information about an entity due to deletion of other entity.

If we delete from EMP_PROJ an employee tuple that happens to represent the last employee working for a particular project, the information concerning that project is lost from the database.

This problem does not occur in the database of Figure 2 because PROJECT tuples are stored separately.

Insertion Anomaly

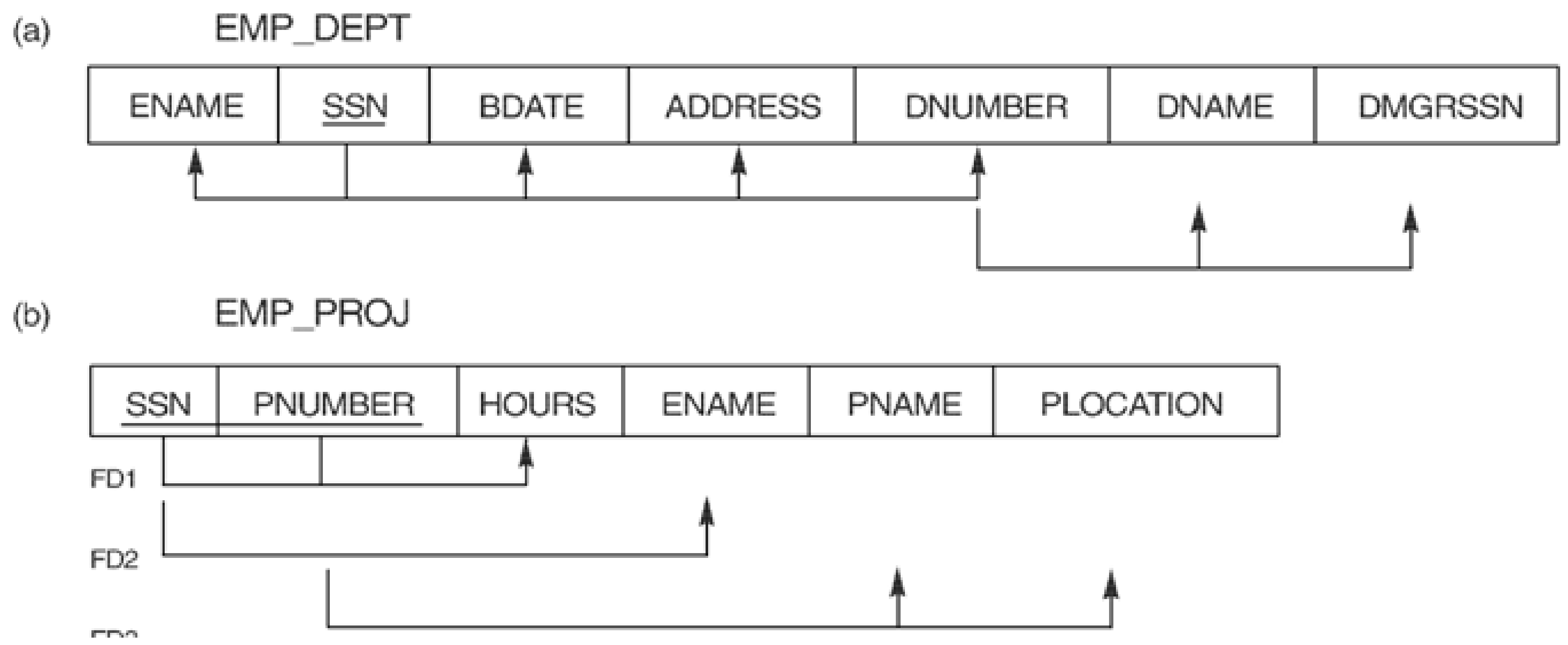
- Cannot insert an entity to the database unless another entity is inserted.

Example: Cannot insert a project unless an employee is assigned to the project.

Inversely - Cannot insert an employee unless an he/she is assigned to a project.

Exercise: Figure out the update anomalies of the following relational schemas

Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.



Guideline to Redundant Information in Tuples and Update Anomalies

GUIDELINE 2: Design a schema that does not suffer from the insertion, deletion and modification anomalies. If there are any present, then note them so that the programs (applications) can be made to take them into account.

1.3 Null Values in Tuples

GUIDELINE 3: Relations should be designed such that their tuples will have as few NULL values as possible

- Reasons for nulls:
  - attribute not applicable or invalid
  - attribute value unknown (may exist)
  - value known to exist, but unavailable

1.3 Null Values in Tuples

Problems with NULL

1. Waste of space
2. Cannot apply numeric functions to that attribute (Sum(), Average(),Min())
3. It may have multiple interpretation

- Attributes that are NULL frequently could be placed in separate relations (with the primary key)

If only 10% of employees have individual offices, no reason to have attribute OFFICE_NUMBER in the EMPLOYEE relation; rather, a relation EMP_OFFICES (ESSN, OFFICE_NUMBER) can be created to include tuples for only the employees with individual offices.

1.4 Spurious Tuples

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

Figure 3

Suppose EMP_LOCS and EMP_PROJ1 is used instead of the single EMP_PROJ

EMP_LOCS

ENAME	PLOCATION

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION

Figure 4

1.4 Spurious Tuples

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION
1	P1	22	A	A	L
2	P2	11	B	B	M
1	P2	13	A	B	M
2	P3	22	B	C	N

EMP_LOCS

ENAME	PLOCATION
A	L
B	M
A	M
B	N

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION
1	P1	22	A	L
2	P2	11	B	M
1	P2	13	B	M
2	P3	22	C	N

1.4 Spurious Tuples

EMP_PROJ

SSN	**PNUMBER**	**HOURS**	**ENAME**	**PNAME**	**PLOCATION**
1	P1	22	A	A	L
2	P2	11	B	B	M
2	**P2**	**11**	**A**	**B**	**M**
1	P2	11	A	B	M
1	**P2**	**13**	**A**	**B**	**L**
2	P3	22	B	C	N

Spurious (fake) tuples

1.4 Spurious Tuples

- A spurious tuple is, basically, a record in a database that gets created when two tables are joined badly
- Spurious tuples are created when two tables are joined on attributes that are neither primary keys nor foreign keys.

GUIDELINE 4: Design schemas so they can be Joined with primary key, foreign key pairs with no spurious tuples

2. 1 Functional Dependencies

- Functional dependencies (FDs) are used to specify *formal measures* of the "goodness" of relational designs.
- Functional dependencies are constraints between two set of attributes from the database.
- Cannot be inferred from data Defined explicitly by DB designer

Functional Dependencies

- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y.
- Written as X -> Y; can be displayed graphically on a relation schema by an arrow
- X -> Y if whenever two tuples have the same value for X, they *must have* the same value for Y
- For any two tuples t1 and t2 in any relation instance r(R): *If* t1[X]=t2[X], *then* t1[Y]=t2[Y]
- X -> Y in R specifies a *constraint* on all relation instances r(R)

Examples of FD constraints

- social security number determines employee name

  SSN -> ENAME
- project number determines project name and location

  PNUMBER -> {PNAME, PLOCATION}
- employee ssn and project number determines the hours per week that the employee works on the project

  {SSN, PNUMBER} -> HOURS

Examples of FD constraints

- An FD is a **property** of the **attributes** in the schema R
- FD is depend on the meaning (semantics) of attributes
- 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])

Examples of FD constraints

Teacher	**Course**	**Book**
A	Data Structures	Bartam
A	Programming	Allman
B	Compilers	Hoffman
C	Data Structures	Seller
D	C++	Seller

Teacher $\nrightarrow$ Course

Book $\nrightarrow$ Course

Exercise

Student No	Name	Town	DoB	Gender
A1	Aruna	Colombo	98/10/15	Male
A2	Sisira	Galle	97/11/15	Male
A3	Sitha	Matara	96/04/15	Female
A4	Maduri	Panadura	98/11/15	Female
A5	Aruna	Galle	97/11/15	Male
A6	Tikiri	Colombo	96/02/15	Female
A7	Aruni	Galle	95/01/05	Female
A8	Sidath	Panadura	97/10/15	Male

Identify some functional dependencies?

2.2 Inference Rules for FDs

- Given a set of FDs F, we can *infer* additional FDs that hold whenever the FDs in F hold

  Abbreviation: FD{X,Y,Z} -> {U,V} written as XYZ->UV

Armstrong's inference rules:

IR1. (**Reflexive**) If Y $\subseteq$ X then X -> Y

IR2. (**Augmentation**) If X -> Y, then XZ -> YZ

IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a *sound* and *complete* set of inference rules

Inference Rules for FDs

Some **additional inference rules** that are useful:

(**Decomposition**) If X -> YZ, then X -> Y and X -> Z

(**Union**) If X -> Y and X -> Z, then X -> YZ

(**Psuedotransitivity**) If X -> Y and WY -> Z, then WX -> Z

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

Inference Rules for FDs

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F

- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X

- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

2.3 Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - every FD in F can be inferred from G, *and*
  - every FD in G can be inferred from F
- Hence, F and G are equivalent if $F^+ = G^+$

Definition: F **covers** G if every FD in G can be inferred from F (i.e., if G^+ *subset-of* F^+)

- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

2.4 Minimal Sets of FDs (1)

- A set of FDs is **minimal** if it satisfies the following conditions:

(1) Every dependency in F has a single attribute for its RHS.

(2) We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.

(3) We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X (Y <u>subset-of</u> X) and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set (e.g., see algorithms 11.2 and 11.4)

Additional FDs

A systematically we can determine the additional FDs.

1. Determine each set of attributes X that appears as a left hand side of some FDs in F.

2. Use inference rules to determine the set of all attributes (X^+) that are depend on X.

Additional FDs

```
$X^+ = X$
Do {
    old $X^+ = X^+$
    for(each FD $Y \rightarrow Z$ in F)
        if ($Y \subseteq X^+$)
            $X^+ = X^+ \cup Z$
} while (old $X^+ == X^+$)
```

Example

F = {SSN→ENAME,PNUMBER→{PNAME,PLOCATION}, {SSN,PNUMBER}→HOURS}

{ SSN }$^+$ = { SSN, ENAME }
{ PNUMBER }$^+$ = { PNUMBER, PNAME, PLOCATION }
{ SSN, PNUMBER }$^+$ = { SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS }

Additional dependencies that can be derived.
{<u>SSN,PNUMBER</u>}→{SSN,ENAME,PNUMBER,PNAME, PLOCATION}

Exercise

Consider the relation schema R={A,B,C,D,E,F,G,H,I,J} and the set of functional dependencies F={{A,B}→{C}, {A}→{D,E}, {B}→{F}, {F}→{G,H}, {D}→{I,J}}

1. Find the key of R
2. Decompose R into 2 NF then 3 NF relations.

Solution

$\{A,B\}^+ = \{A,B,C,D,E,F,G,H,I,J\}$

$\{A\}^+ = \{A,D,E,I,J\}$

$\{B\}^+ = \{B,F,G,H\}$

$\{F\}^+ = \{F,G,H\}$

$\{D\}^+ = \{D,I,J\}$

Key = {A,B}

Prime attributes are A and B

Non prime attributes are C,D,E,F,G,H,I,J

$\{A\} \rightarrow \{D,E\}$ and $\{B\} \rightarrow \{F\}$ are partial dependencies.

1. Find the key of R
2. Decompose R into 2 NF then 3 NF relations.

Exercise

Consider the relation schema R={A,B,C,D,E} and the set of functional dependencies

F={{A}→{C}, {C}→{A}, {B}→{D}, {D}→{B}, {A,B}→{E}, {A,D}→{E}, {C,D}→{E}}

Find the keys of R

3 Normal Forms Based on Primary Keys

3.1 Normalization of Relations

- **Normalization** of data is the process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relations schemas that meet the desirable properties

- One objective of the normalization process is to ensure that the update anomalies do not occur.

- **Normal form**: Condition that use ***keys*** and ***FDs*** of a relation to certify whether a relation schema is in a particular normal form

3.2 Practical Use of Normal Forms

- The database designers ***need not*** normalize to the highest possible normal form. (usually up to 3NF, BCNF or 4NF)

- **Denormalization:** the process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

3.3 First Normal Form

- Disallows composite attributes, multi-valued attributes, and **nested relations**; (attributes whose values *for an individual tuple* are non-atomic).

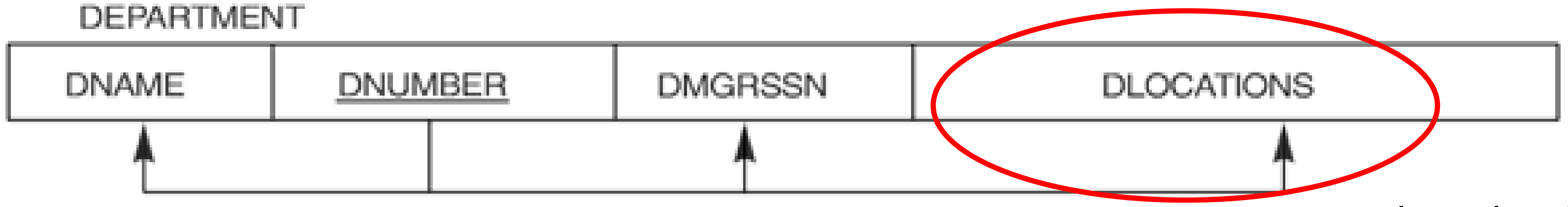
- This means domains of attributes must include only atomic, simple, indivisible values and the value of any attribute in a tuple must be a single value from the domain of that attribute.

Normalization into 1NF

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

(a)

DEPARTMENT



Multi-valued Attribute

(b)

DEPARTMENT

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

Set of values

(c)

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

Normalization into 1NF

Method 1:

Remove the attribute that violates 1NF and place it in a new relation with the primary key.

1 NF

Department(DNAME, <u>DNUMBER</u>,DMGRSSN)

Dept_Location(<u>DNUMBER</u>,<u>DLOCATION</u>)

- Create a new relation for the nested relation.

Normalization into 1NF

Method 2:

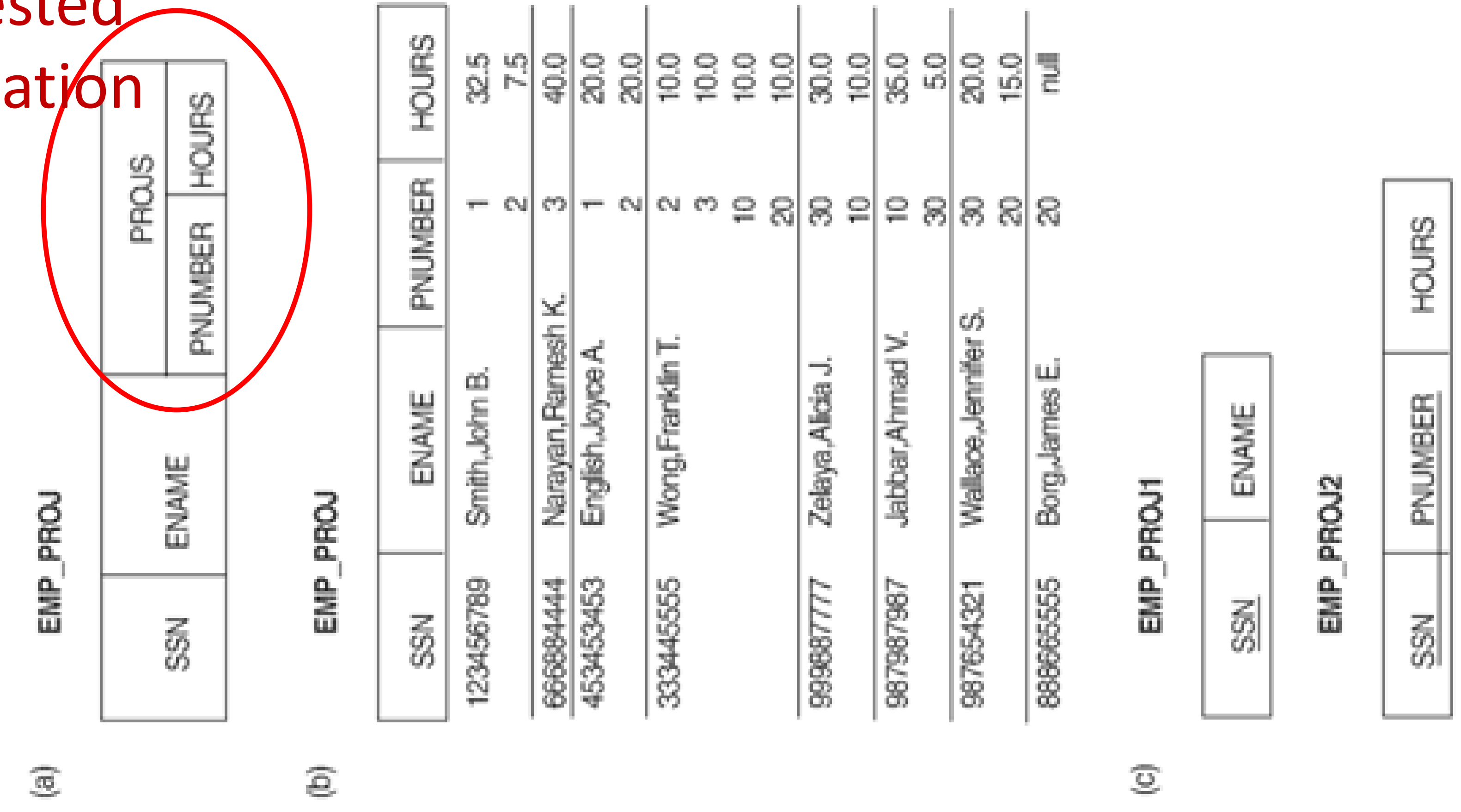
Introduce a new tuple for each multi-value in the original relation.

See Figure C

Normalization nested relations into

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.

Nested relation



(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

SSN	ENAME

EMP_PROJ2

SSN	PNUMBER	HOURS

Normalization nested relations into 1NF

- Create a new relation for the nested relation and propagate the primary key to it.

- Primary key of new relation= Primary key of original relation + partial key.

3.4 Second Normal Form

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

Definitions:

- **Full functional dependency** - A FD Y -> Z is a full functional dependency, if removal of any attribute A from Y means that the functional dependency does not hold any more.

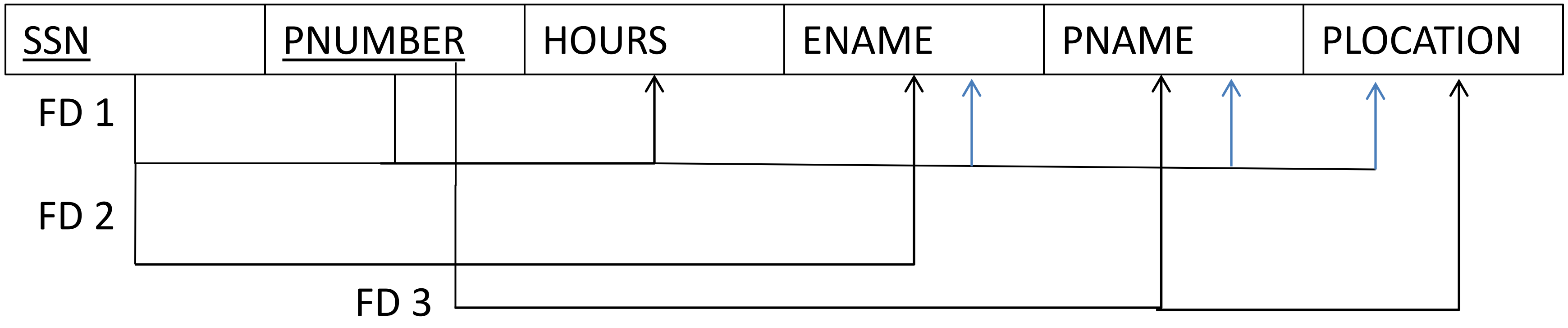
  i.e. for any $A \in Y$, $(Y-\{A\}) \not\rightarrow Z$

Second Normal Form

- **Partial functional dependency** - A FD Y -> Z is a partial dependency, if some attribute can be removed and the dependency still holds.

  i.e. for any A ε Y, (Y-{A}) → Z

EMP_PROJ



Second Normal Form

Examples:

{SSN, PNUMBER} -> HOURS is a full FD, since neither SSN -> HOURS nor PNUMBER -> HOURS does not hold if PNUMBER or SSN is removed.

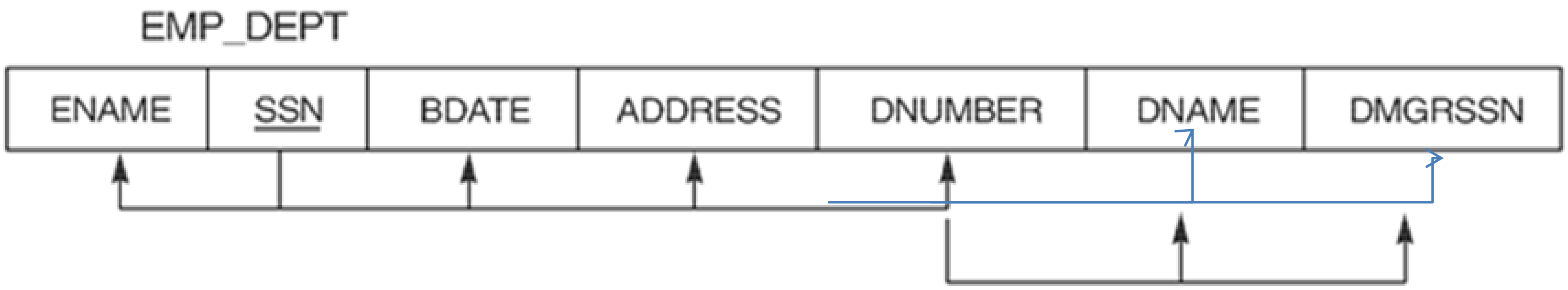
{SSN, PNUMBER} -> ENAME is *not* a full FD (it is called a *partial dependency*) since SSN -> ENAME also holds if PNUMBER is removed.

Definitions of Keys and Attributes Participating in Keys

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

- 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



Super Key = {SSN}, {SSN,ENAME},{SSN,ENAME,BDATE}……….

Key = {SSN}

Definitions of Keys and Attributes Participating in Keys

- 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*.

Prime attribute & Nonprime attribute

- An attribute of a relation schema R is called a **prime attribute** of R if it is a member of some *candidate key* of R.
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

Prime attribute : PNUMBER, SSN

Non prime attribute : PNAME, ENAME,HOURS, PLOCATION

Second Normal Form

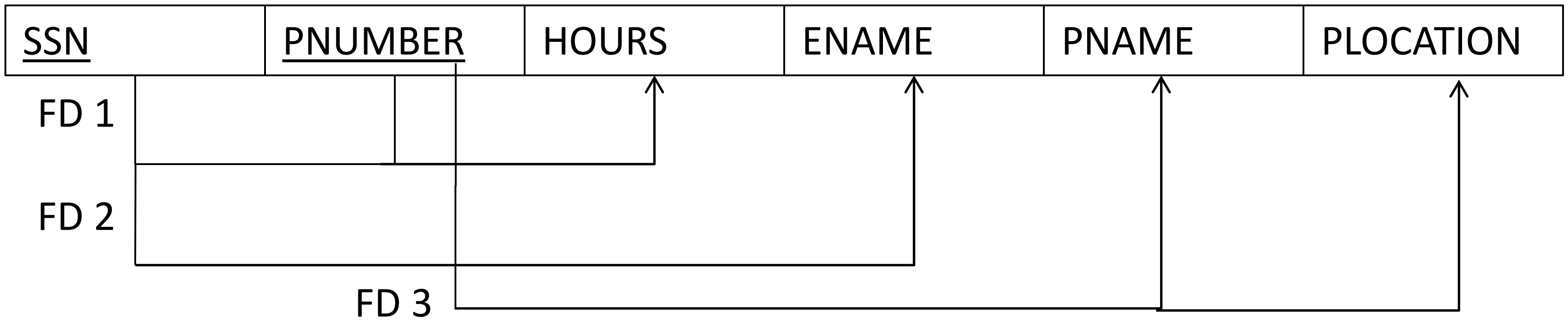
- A relation schema R is in **second normal form (2NF)** if every non-prime attribute A in R is fully functionally dependent on the primary key OR if every non-prime attribute A in R is not partially dependent on any key of R.

Second Normal Form

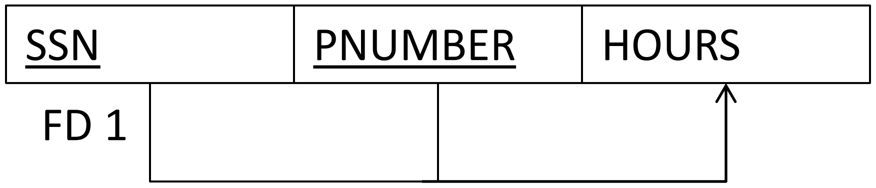
- If a relation schema R is not in **second normal form (2NF)**, it can be converted to number of 2NF relations in which non- prime attributes are associated only with part of any key on which they are fully functionally dependent.

Normalizing into 2NF

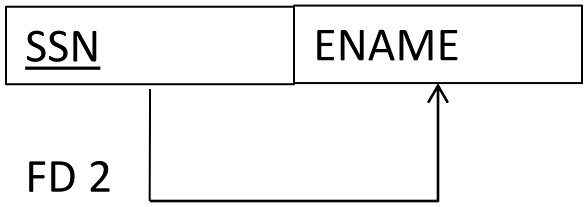
EMP_PROJ



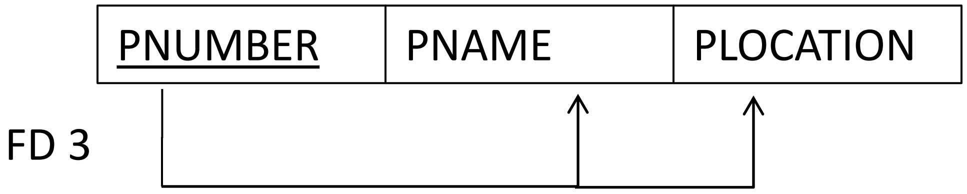
EMP_PROJ 1



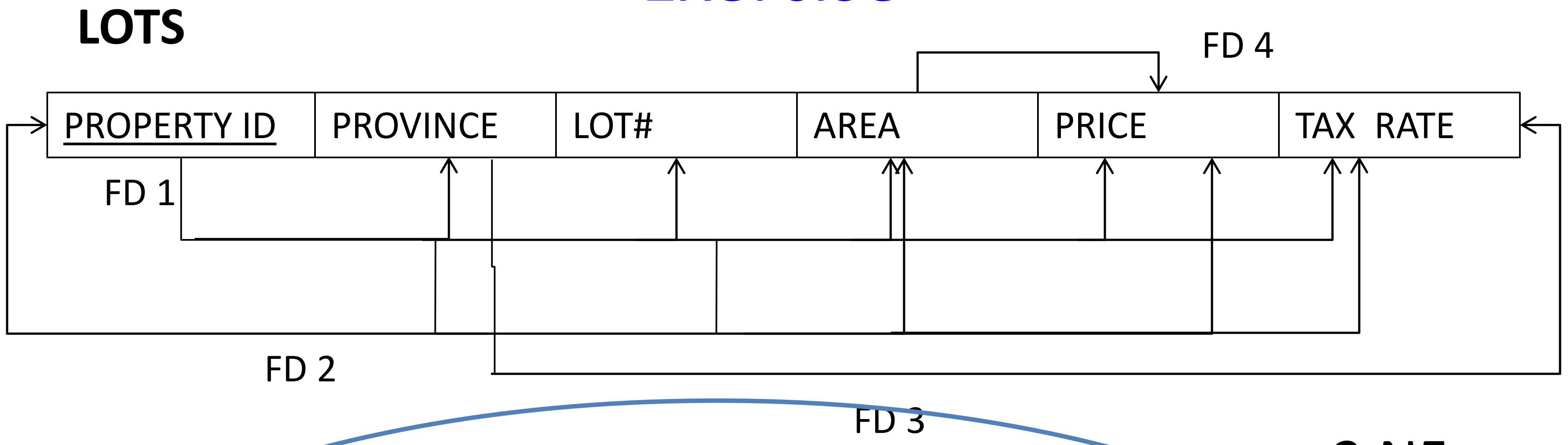
EMPLOYEE



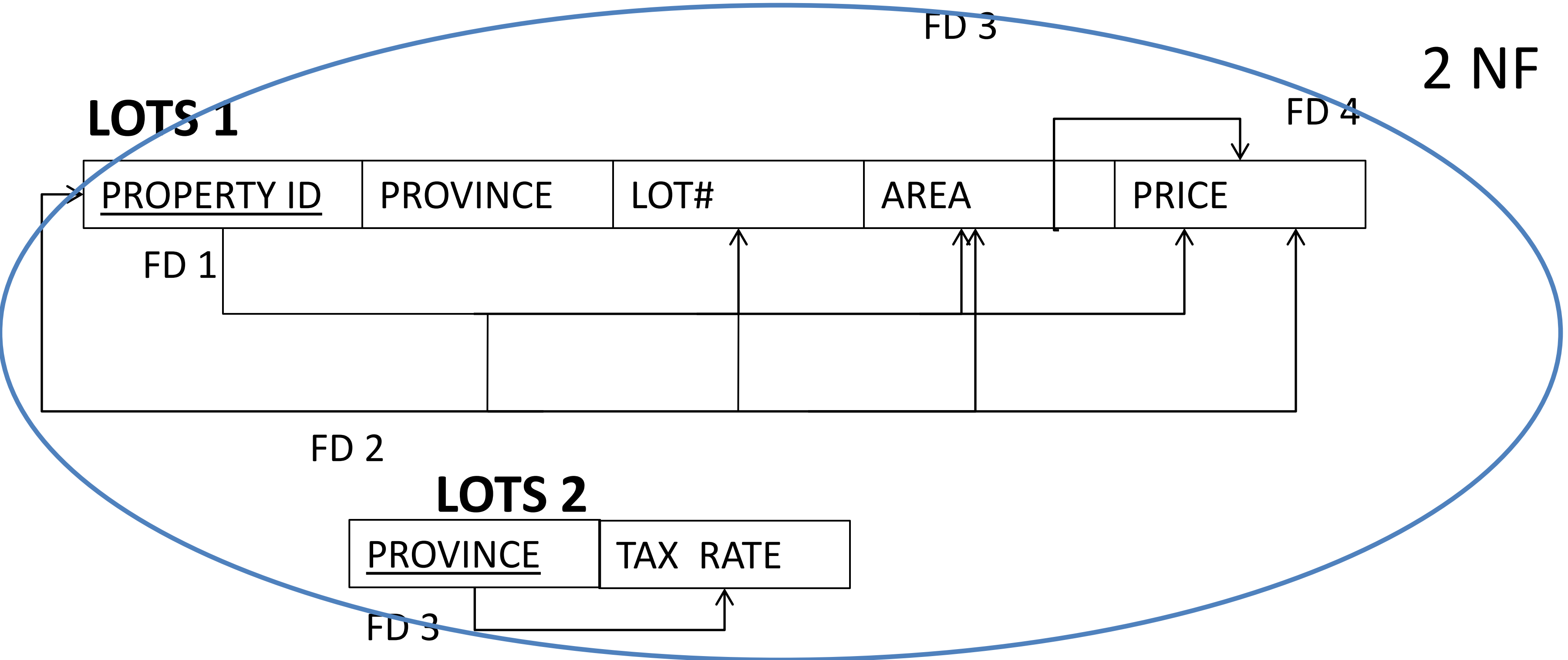
PROJECT



Exercise



2 NF



3.4 Third Normal Form

Definition:

- **Transitive functional dependency** - a FD X -> Z that can be derived from two FDs X -> Y and Y -> 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

- 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 not a candidate key. When Y is a candidate key, there is no problem with the transitive dependency .

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

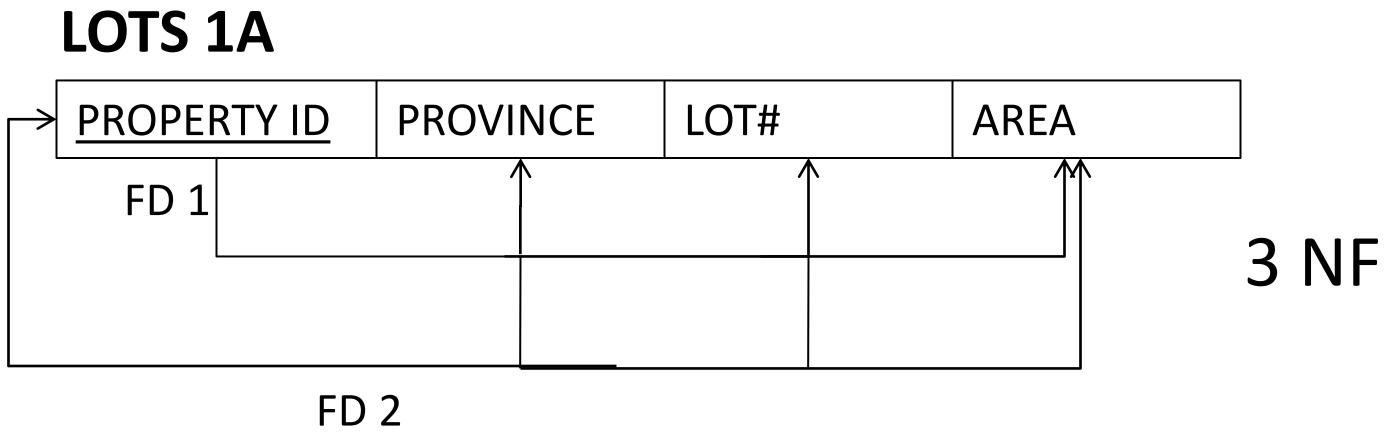
Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

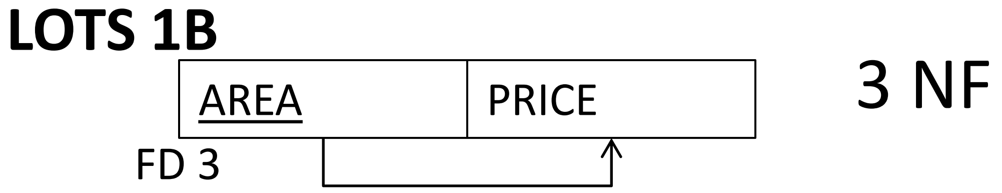
Third Normal Form

A relation schema R is in **third normal form (3NF)** if whenever a FD X -> A holds in R, then either:

(a) X is a superkey of R, or

(b) A is a prime attribute of R



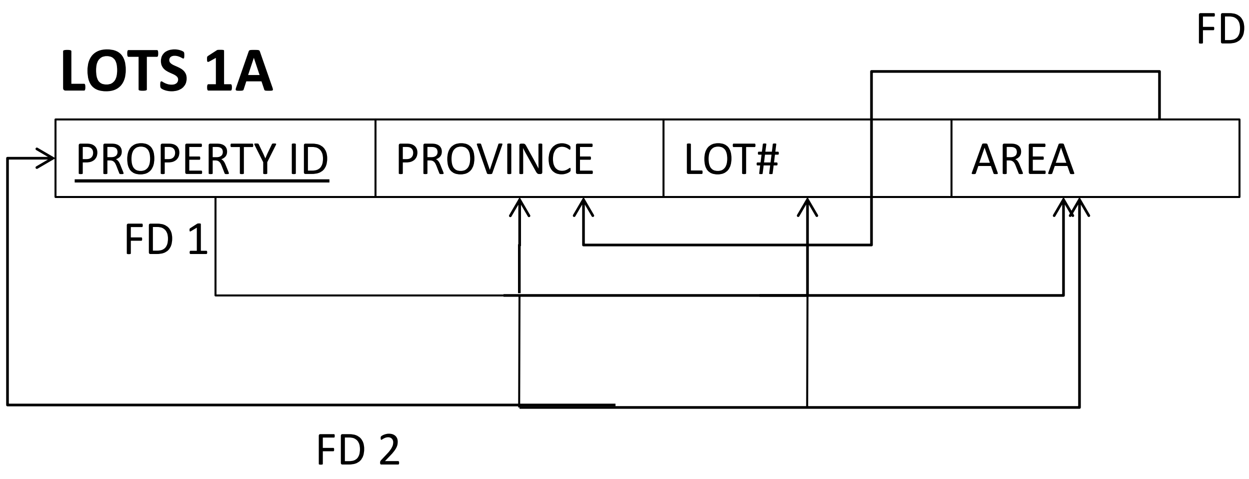


BCNF (Boyce-Codd Normal Form)

- 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
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF (reverse is not true)

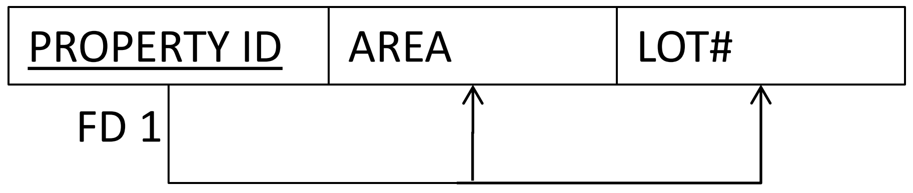
BCNF (Boyce-Codd Normal Form)

LOTS 1A

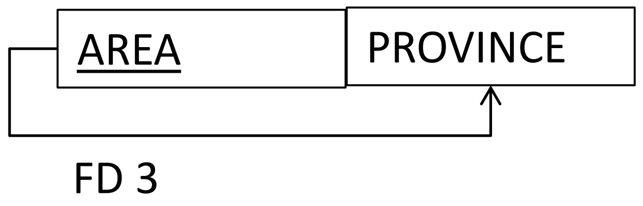


Not in BCNF

LOTS 1A_1



LOTS 1A_B


AREA
PROVINCE
FD 3