Basics of Functional Dependencies and Normalization

Part 3

Normal Forms Based on Primary Keys

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.
- Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.

- **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.

- Normal forms, when considered *in isolation* from other factors, do not guarantee a good database design.
- It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF.
- Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess.
- These would include two properties:
  - The **nonadditive join** or **lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
  - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

- The nonadditive join property is extremely critical and **must be achieved at any cost**, whereas the dependency preservation property, although desirable, is sometimes sacrificed.

Practical Use of Normal Forms

- Although several higher normal forms have been defined, such as the 4NF and 5NF, the practical utility of these normal forms becomes questionable when the constraints on which they are based are rare, and hard to understand or to detect by the database designers and users who must discover these constraints.
- Thus, database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.

- Another point worth noting is that the database designers *need not* normalize to the highest possible normal form.
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons
- Doing so incurs the corresponding penalties of dealing with the anomalies.

- **Definition. Denormalization** is 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

- **Definition.** A **superkey** of a relation schema $R = \{A_1, A_2, \ldots, A_n\}$ is a set of attributes $S \subseteq R$ 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.

- 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.

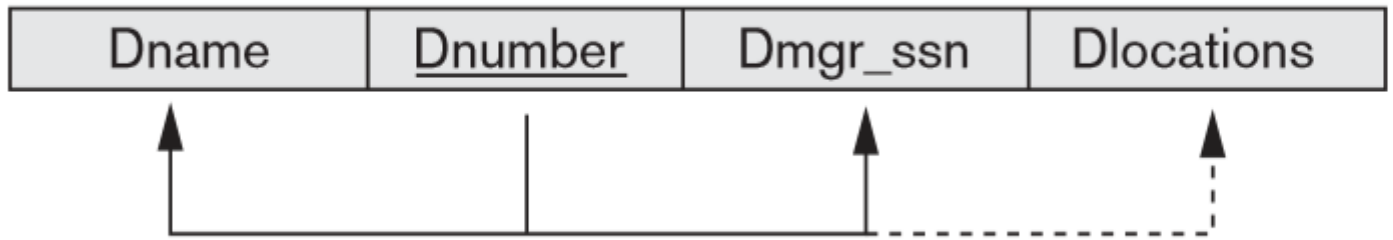
- **Definition.** An attribute of relation schema R is called a **prime attribute** of R if it is a member of *some candidate key* of R. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key.

First Normal Form

- **First normal form** (**1NF**) is now considered to be part of the formal definition of a relation in the basic (flat) relational model.
- It states that the domain of an attribute must include only *atomic* (simple, indivisible) *values* and that the value of any attribute in a tuple must be a *single value* from the domain of that attribute.
- Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a *single tuple*.
- In other words,1NF disallows *relations within relations* or *relations as attribute values within tuples*.
- The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.

(a)

DEPARTMENT



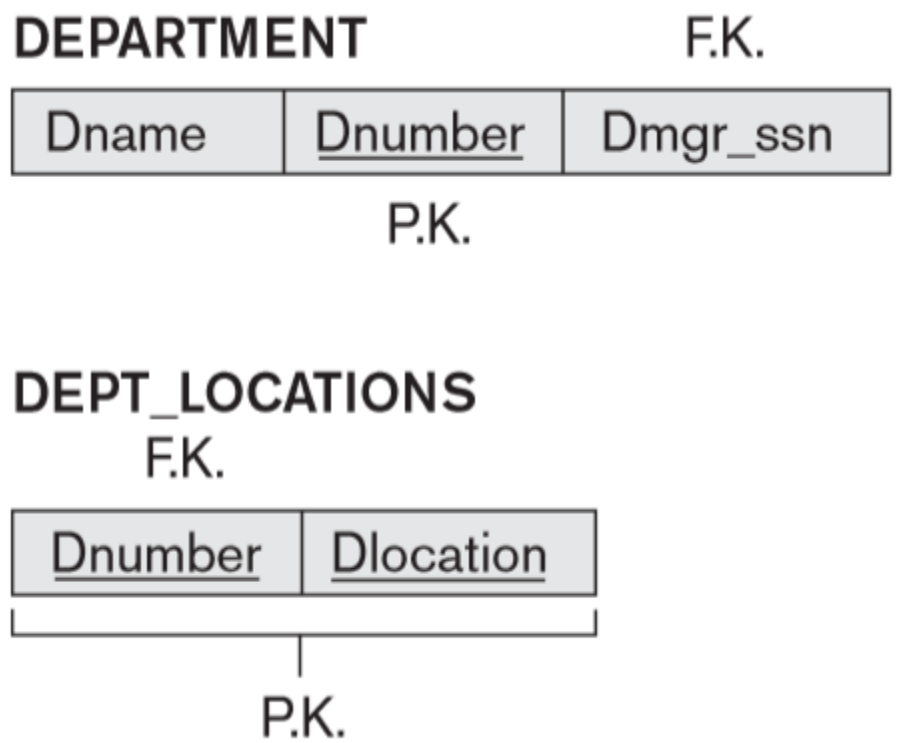
(b)

DEPARTMENT

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

- 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 relation is the combination {`Dnumber`, `Dlocation`}. A distinct tuple in `DEPT_LOCATIONS` exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.


DEPARTMENT
F.K.
Dname
Dnumber
Dmgr_ssn
P.K.
DEPT_LOCATIONS
F.K.
Dnumber
Dlocation
P.K.

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.

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. It further introduces spurious semantics about the ordering among the location values that is not originally intended. Querying on this attribute becomes more difficult; for example, consider how you would write the query: *List the departments that have 'Bellaire' as one of their locations* in this design.

- Of the three solutions above, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values.
- In fact, if we choose the second solution, it will be decomposed further during subsequent normalization steps into the first solution.

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

(a)

EMP_PROJ

		Projs	
Ssn	Ename	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

- Notice that `Ssn` is the primary key of the `EMP_PROJ`, while `Pnumber` is the **partial** key of the nested relation; that is, within each tuple, the nested relation must have unique values of `Pnumber`.
- To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key into it*; the primary key of the new relation will combine the partial key with the primary key of the original relation.

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

- This procedure can be applied recursively to a relation with multiple-level nesting to **unnest** the relation into a set of 1NF relations.
- This is useful in converting an unnormalized relation schema with many levels of nesting into 1NF relations.

- The existence of more than one multivalued attribute in one relation must be handled carefully.
- As an example, consider the following non-1NF relation:

  `PERSON(Ss#,{Car_lic#},{Phone#})`
- This relation represents the fact that a person has multiple cars and multiple phones.

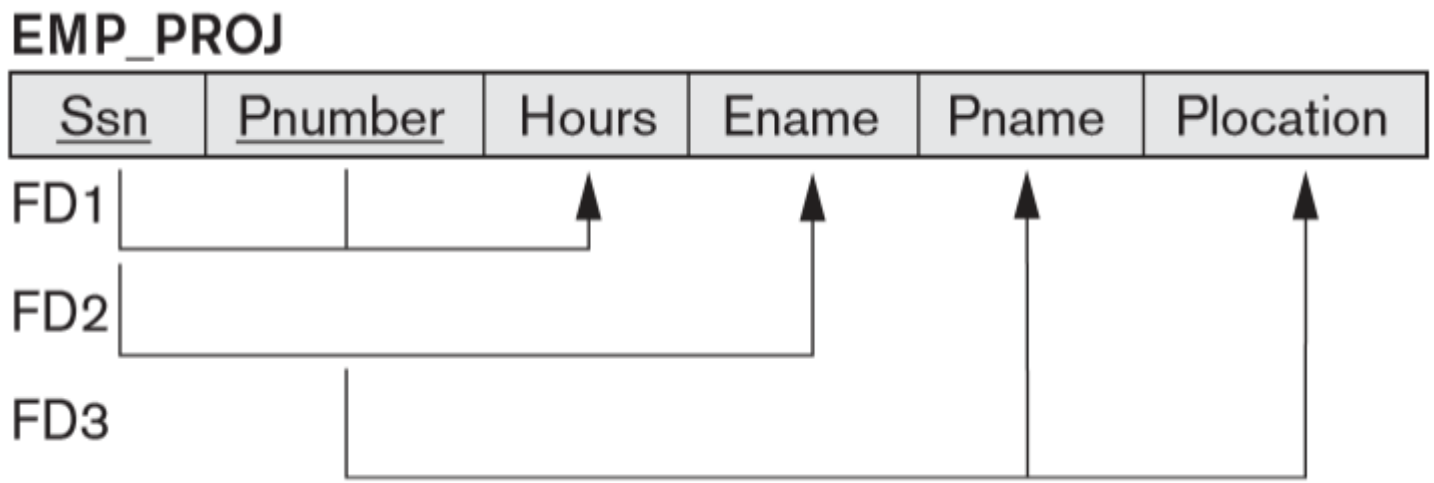
- If strategy 2 above is followed, it results in an all-key relation:
  `PERSON_IN_1NF (Ss#, Car_lic#, Phone#)`
- To avoid introducing any extraneous relationship between `Car_lic#` and `Phone#`, all possible combinations of values are represented for every `Ss#`, giving rise to redundancy.
- This leads to the problems handled by multivalued dependencies and 4NF.

- The right way to deal with the two multivalued attributes in PERSON is to decompose it into two separate relations, using strategy 1 discussed above: P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

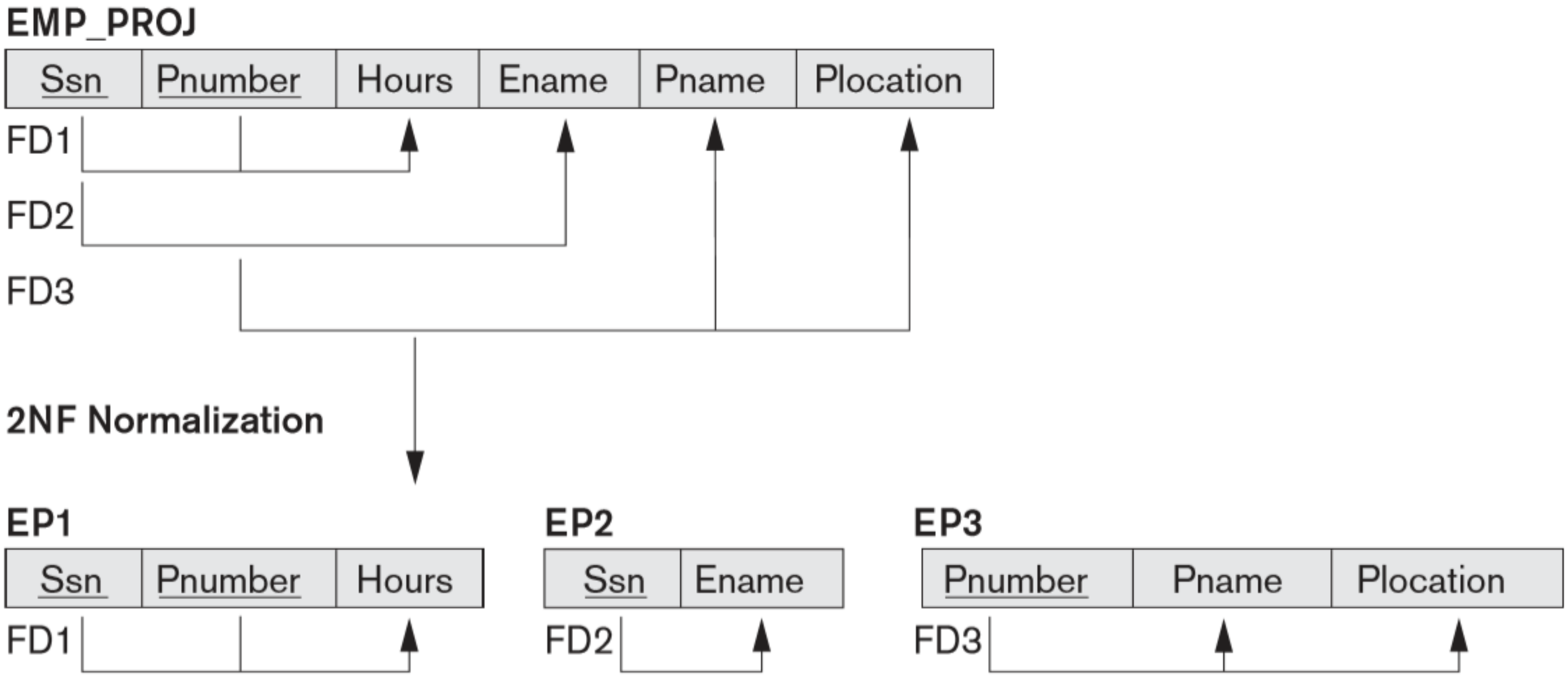
Second Normal Form

- **Second normal form** (**2NF**) is based on the concept of full functional dependency.
- A functional dependency $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y.
- A functional dependency $X \rightarrow Y$ is a **partial dependency** if some attribute $A \in X$ can be removed from X and the dependency still holds; that is, for some $A \in X$, $(X - \{A\}) \rightarrow Y$.

- **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.


EMP_PROJ
Ssn
Pnumber
Hours
Ename
Pname
Plocation
FD1
FD2
FD3

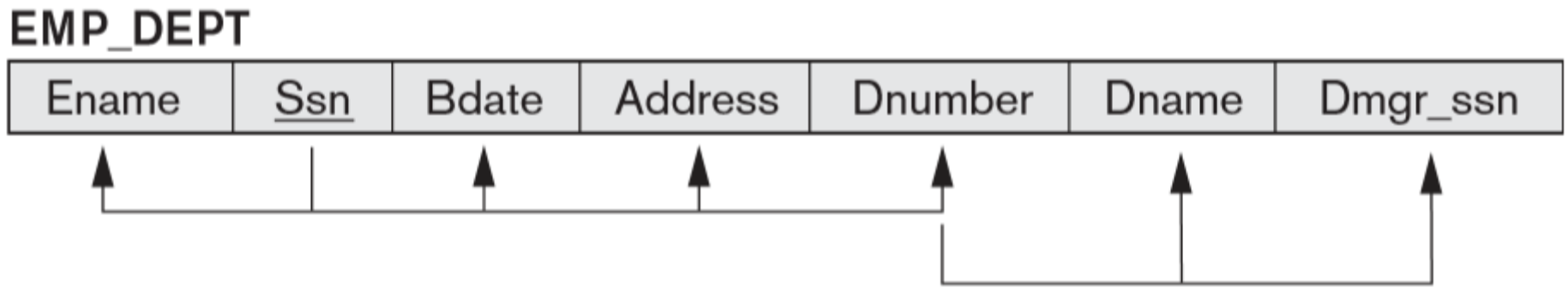
- If a relation schema is not in 2NF, it can be *second normalized* or *2NF normalized* into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.

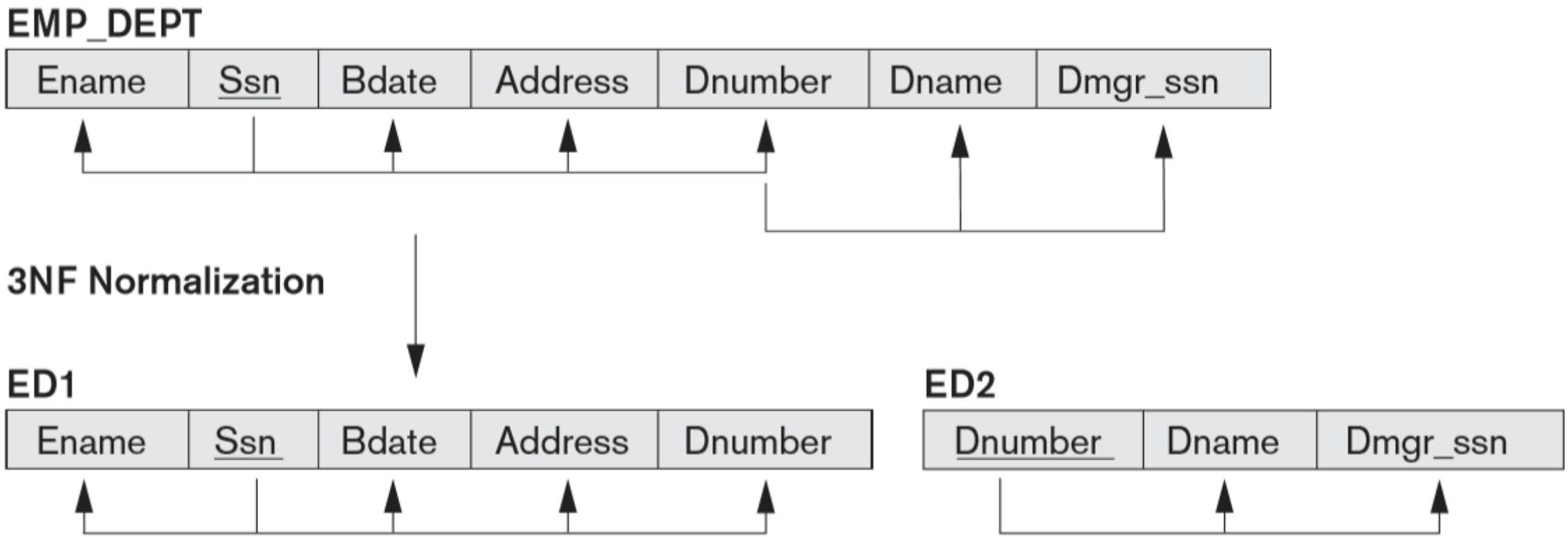

EMP_PROJ
Ssn
Pnumber
Hours
Ename
Pname
Plocation
FD1
FD2
FD3
2NF Normalization
EP1
Ssn
Pnumber
Hours
FD1
EP2
Ssn
Ename
FD2
EP3
Pnumber
Pname
Plocation
FD3

Third Normal Form

- **Third normal form** (**3NF**) is based on the concept of *transitive dependency*.
- A functional dependency $X \rightarrow Y$ 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 $X \rightarrow Z$ 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.


EMP_DEPT
Ename
Ssn
Bdate
Address
Dnumber
Dname
Dmgr_ssn


EMP_DEPT
Ename
Ssn
Bdate
Address
Dnumber
Dname
Dmgr_ssn
3NF Normalization
ED1
Ename
Ssn
Bdate
Address
Dnumber
ED2
Dnumber
Dname
Dmgr_ssn