Lecture 6: Functional Dependencies and Normalization

CS3402 Database Systems

Definition of Functional Dependency (1/2)

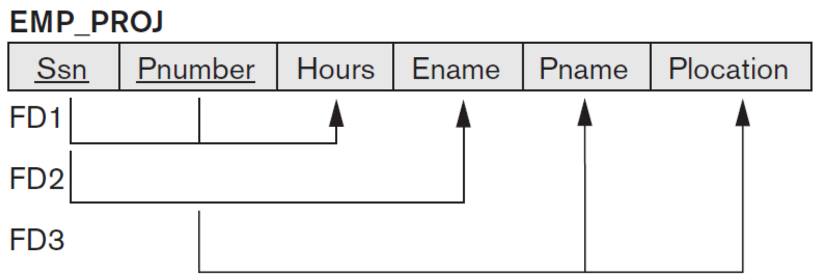
- The single most important concept in relational schema design theory is that of a functional dependency.
- A functional dependency is a constraint between two sets of attributes from the database.
- Suppose that our relational database schema has n attributes A_1, A_2, … , A_n; let us think of the whole database as being described by a single **universal** relation schema $R = \{A_1, A_2, \ldots, A_n\}$. We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies.

Definition of Functional Dependency (2/2)

- **Definition.** A **functional dependency**, denoted by $X \rightarrow Y$, 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 t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.
- 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** 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**.

Examples of Functional Dependency (1/2)

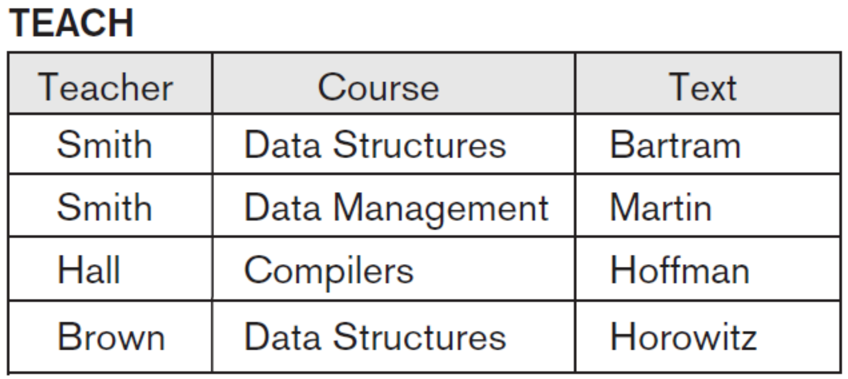
- Consider the relation schema EMP_PROJ, we know that the following functional dependencies should hold:
  a) Ssn → Ename (FD2)
  b) Pnumber → {Pname, Plocation} (FD3)
  c) {Ssn, Pnumber} → Hours (FD1)



- These functional dependencies specify that
  a) The value of an employee's Social Security number (Ssn) uniquely determines the employee name (Ename). Alternatively, we say that Ename is functionally determined by (or functionally dependent on) Ssn, or given a value of Ssn, we know the value of Ename.
  b) The value of a project's number (Pnumber) uniquely determines the project name (Pname) and location (Plocation).
  c) A combination of Ssn and Pnumber values uniquely determines the number of hours the employee currently works on the project per week (Hours).

Examples of Functional Dependency (2/2)

- For example, here shows a particular state of the TEACH relation schema.
- Although at first glance we may think that Text → Course, we cannot confirm this unless we know that it is true for all possible legal states of TEACH.
- It is, however, sufficient to demonstrate a single counterexample to disprove a functional dependency. For example, because ‘Smith’ teaches both ‘Data Structures’ and ‘Database Systems,’ we can conclude that Teacher does not functionally determine Course.



TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Inference Rules for Functional Dependencies (1/5)

- We denote by F the set of functional dependencies specified on relation schema R.
- Typically, the schema designer specifies the functional dependencies that are semantically obvious; usually, however, numerous other functional dependencies hold in all legal relation instances among sets of attributes that can be derived from and satisfy the dependencies in F.
- Those other dependencies can be inferred or deduced from the FDs in F. We call them as inferred or implied functional dependencies.
- In real life, it is impossible to specify all possible functional dependencies for a given situation. Therefore, it is useful to define a concept called closure formally that includes all possible dependencies that can be inferred from the given set F.
- 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^+.

Inference Rules for Functional Dependencies (2/5)

- We use the notation $F \models X \to Y$ to denote that the functional dependency $X \to Y$ is inferred from the set of functional dependencies F.
- Inference rules for functional dependencies:
  - IR1 (reflexive rule): If $X \supseteq Y$, then $X \to Y$.
  - IR2 (augmentation rule): $\{X \to Y\} \models XZ \to YZ$.
  - IR3 (transitive rule): $\{X \to Y, Y \to Z\} \models X \to Z$.
  - IR4 (decomposition, or projective, rule): $\{X \to YZ\} \models X \to Y$.
  - IR5 (union, or additive, rule): $\{X \to Y, X \to Z\} \models X \to YZ$.
  - IR6 (pseudotransitive rule): $\{X \to Y, WY \to Z\} \models WX \to Z$.
- IR1 states that a set of attributes always determines itself or any of its subsets, which is obvious. Because IR1 generates dependencies that are always true, such dependencies are called trivial. Formally, a functional dependency $X \to Y$ is **trivial** if $X \supseteq Y$; otherwise, it is **nontrivial**.

Inference Rules for Functional Dependencies (3/5)

- Example: Suppose we are given a schema R with attributes {A, B, C, D, E, F} and the FDs are: {A → BC, B → E, CD → EF}, show that AD → F holds.
- Proof:
  1) A → BC (given)
  2) A → C (decomposition, IR4, from 1)
  3) AD → CD (augmentation, IR2, from 2)
  4) CD → EF (given)
  5) AD → EF (transitivity, IR3, from 3 and 4)
  6) AD → F (decomposition, IR4, from 5) (proved)
- Note: A → BC means A → {B, C }

Inference Rules for Functional Dependencies (4/5)

- For a set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F; X^+ is called the **closure of X under F**.
- **Input:** A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

```
X⁺ := X;
repeat
        oldX⁺ := X⁺; // Before considering FD in an iteration
        for each functional dependency Y → Z in F do
                if X⁺ ⊇ Y then X⁺ := X⁺ ∪ Z; // After considering FD in an iteration
        until (X⁺ = oldX⁺);
```

- This algorithm starts by setting X^+ to all the attributes in X. Using inference rules IR3 and IR4, we add attributes to X^+, using each functional dependency in F. We keep going through all the dependencies in F (the repeat loop) until no more attributes are added to X^+ during a complete cycle (of the for loop) through the dependencies in F.

Inference Rules for Functional Dependencies (5/5)

- Given a schema R={A, B, C, D, E, F}, F= {$A \rightarrow BC$, $B \rightarrow E$, $E \rightarrow CF$, $CD \rightarrow EF$}, and X={A}.

	oldX$^+$ (before)	FD	X$^+$ (after)	X$^+$ = oldX$^+$?
1st Iteration	{A}	$A \rightarrow BC$	{A, B, C}	False
2nd Iteration	{A, B, C}	$B \rightarrow E$	{A, B, C, E}	False
3rd Iteration	{A, B, C, E}	$E \rightarrow CF$	{A, B, C, E, F}	False
4th Iteration	{A, B, C, E, F}	$CD \rightarrow EF$	{A, B, C, E, F}	True

- A candidate key is a set X of attributes in R such that
  - X$^+$ includes all the attributes in R.
  - There is no proper subset Y of X such that Y$^+$ includes all the attributes in R.

Equivalence of Sets of FDs

- Two sets of functional dependencies E and F are **equivalent** if $E^+ = F^+$**.** Therefore, equivalence means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions—E covers F and F covers E—hold.
- F covers E by calculating X^+ with respect to F for each FD $X \rightarrow Y$ in E, and then checking whether this X^+ includes the attributes in Y. If this is the case for every FD in E, then F covers E. We determine whether E and F are equivalent by checking that E covers F and F covers E.

Definitions of Keys

- A **superkey** of a relation schema $R = \{A_1, A_2, \dots, A_n\}$ 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 anymore.
- The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key $K = \{A_1, A_2, \dots, A_k\}$ of R, then $K - \{A_i\}$ is not a key of R for any A_i, $1 \leq i \leq k$.
- For example, {Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.

Candidate Keys and Primary 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.
- In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey.
- For example, {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key.
- 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.

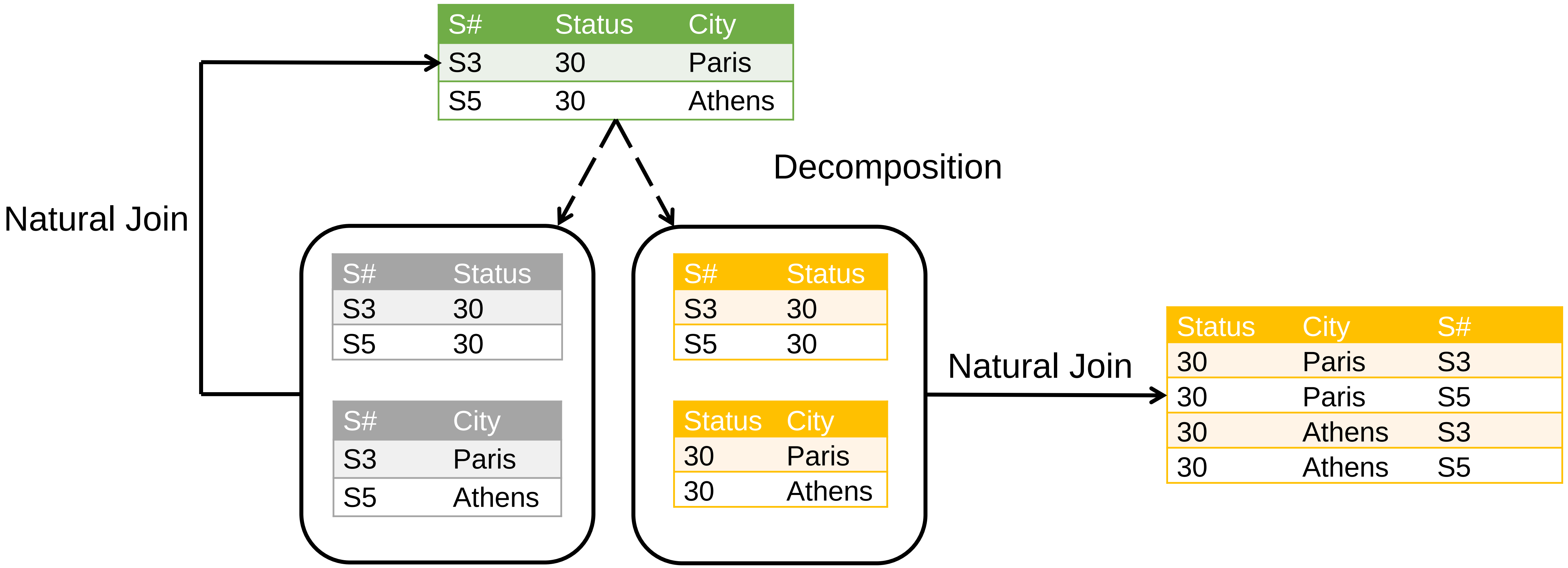
Normalization (1/3)

- 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. Normal forms, when considered in isolation from other factors, do not guarantee a good database design.
- We first introduce three normal forms: first normal form (1NF), second normal form (2NF) and third normal form (3NF) are proposed as a sequence to achieve the desirable state of 3NF relations by progressing through the intermediate states of 1NF and 2NF if needed.

Normalization (2/3)

- Normalization requires two properties
  - Non-additive or lossless join
    - Decomposition is reversible and no information is loss
    - No spurious tuples (tuples that should not exist) should be generated by doing a natural-join of any relations (extremely important)
  - Preservation of the functional dependencies
    - Ensure each functional dependency is represented in some individual relation (sometimes can be sacrificed)

Normalization (3/3)



S#	Status	City
S3	30	Paris
S5	30	Athens

S#	Status
S3	30
S5	30

S#	City
S3	Paris
S5	Athens

S#	Status
S3	30
S5	30

Status	City
30	Paris
30	Athens

Status	City	S#
30	Paris	S3
30	Paris	S5
30	Athens	S3
30	Athens	S5

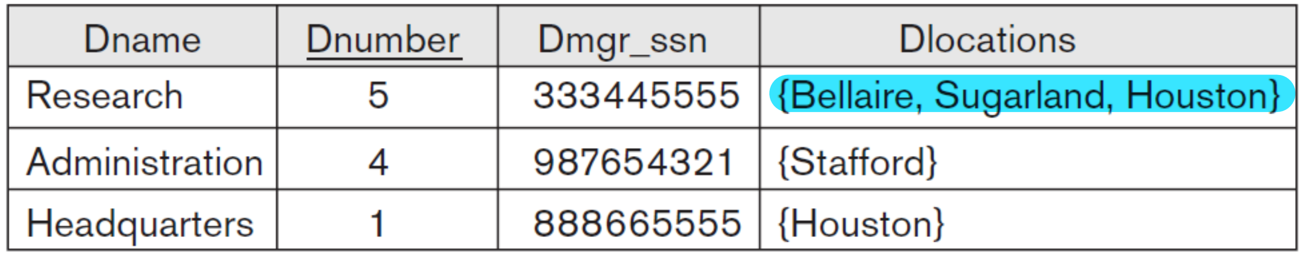
First Normal Form (1NF) (1/8)

- 1NF does not allow multivalued attributes, composite attributes, and their combinations.
- 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 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.

First Normal Form (1NF) (2/8)

- Consider the DEPARTMENT relation schema, whose primary key is Dnumber. We assume that each department can have a number of locations.
- As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple.
- The domain of Dlocations contains atomic values, but some tuples can have a set of these values.

DEPARTMENT



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

First Normal Form (1NF) (3/8)

- There are three main techniques to achieve 1NF 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}. A distinct tuple in DEPT_LOCATIONS exists for each location of a department. This decomposes the non-1NF relation into two 1NF relations.

DEPARTMENT

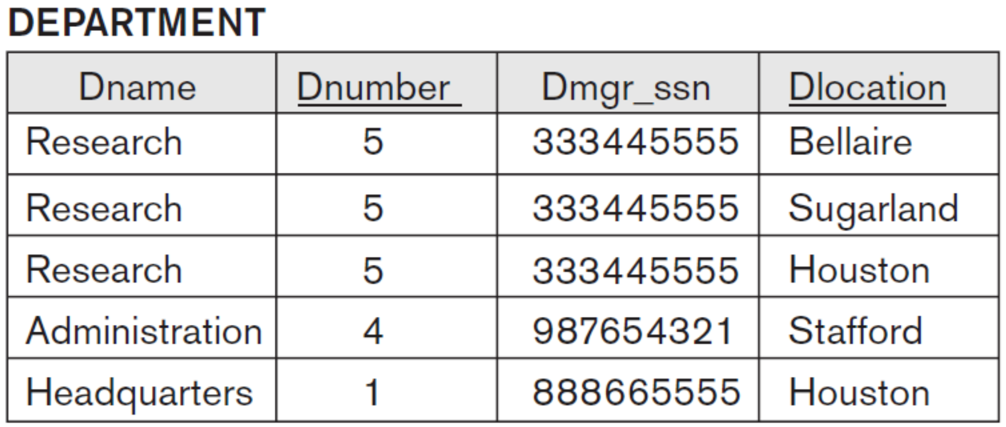
Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

First Normal Form (1NF) (4/8)

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 is rarely adopted.


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

First Normal Form (1NF) (5/8)

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 ordering 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. For all these reasons, it is best to avoid this alternative.

First Normal Form (1NF) (6/8)

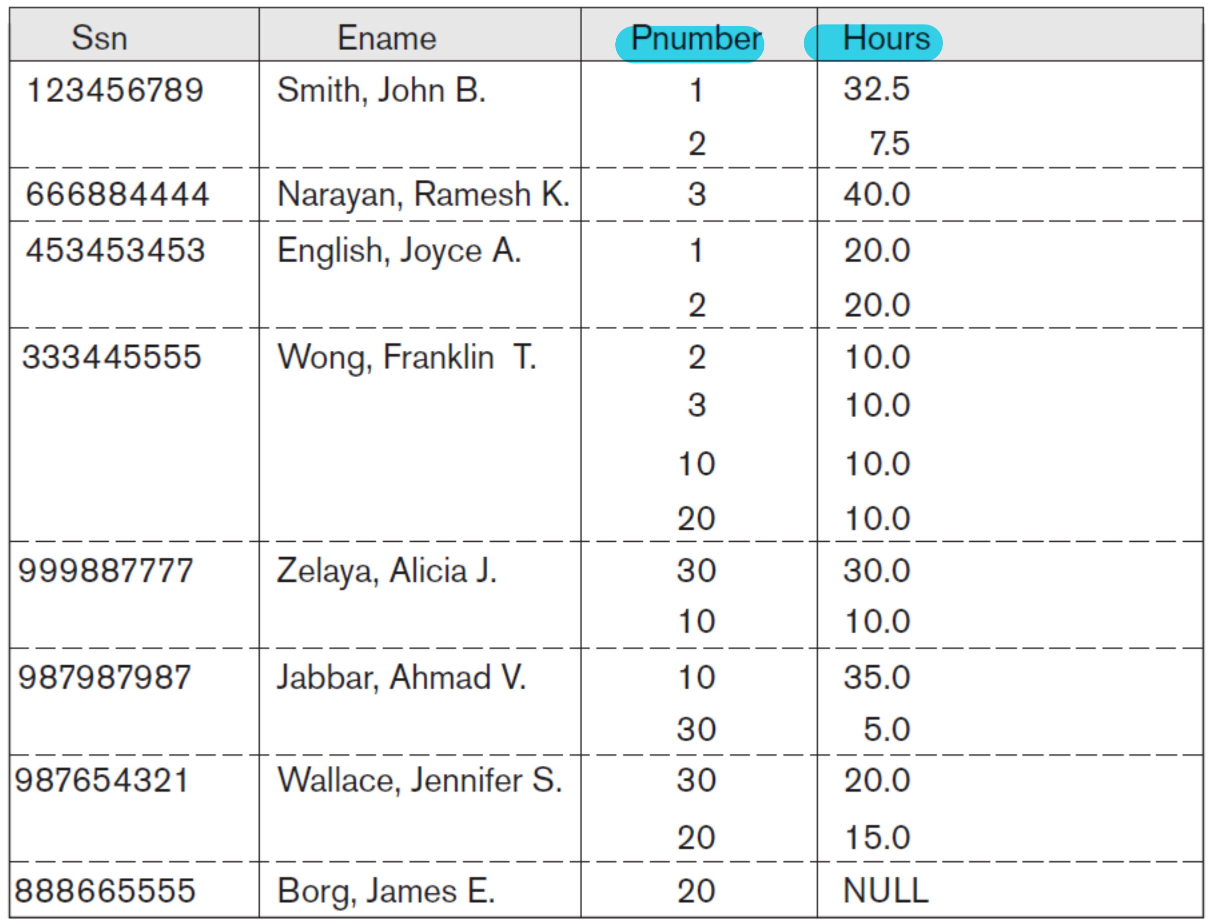
- First normal form also disallows multivalued attributes that are themselves composite. These are called **nested relations** because each tuple can have a relation within it.
- The EMP_PROJ relation could appear if nesting is allowed. Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours) within each tuple represents the employee's projects and the hours per week that employee works on each project. The schema of this EMP_PROJ relation can be represented as follows:

  EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})
- The set braces { } identify the attribute PROJS as multivalued, and we list the component attributes that form PROJS between parentheses ( ).

First Normal Form (1NF) (7/8)

- Notice that Ssn is the primary key of the EMP_PROJ relation, whereas Pnumber is the **partial key** of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber.

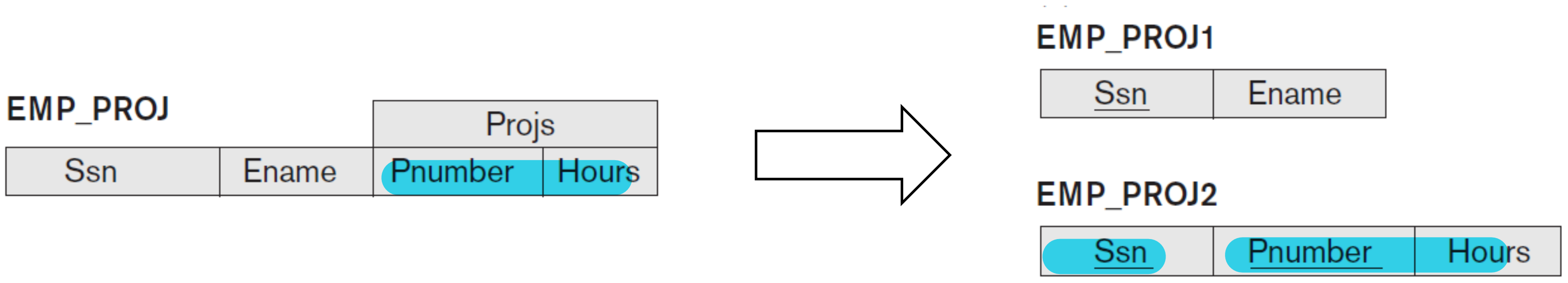
EMP_PROJ



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

First Normal Form (1NF) (8/8)

- 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.
- Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2.



EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

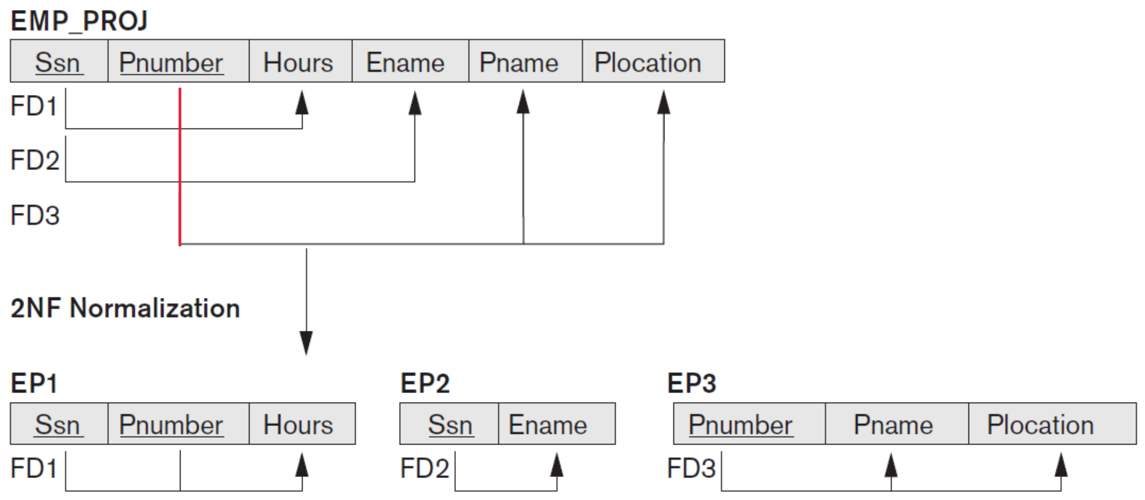
Ssn	Pnumber	Hours

Second Normal Form (2NF) (1/2)

- **Second normal form (2NF)** is based on the concept of 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.
- A relation schema R is in 2NF if every nonprime attribute A in R **is fully functionally dependent** on the primary key of R.

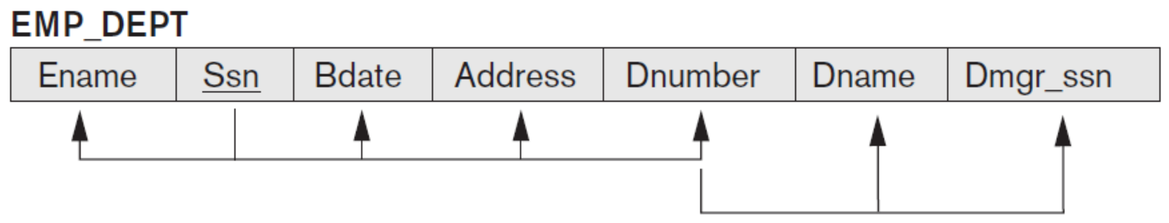
Second Normal Form (2NF) (2/2)

- The EMP_PROJ relation is in 1NF but is not in 2NF. Each of the functional dependencies FD2 and FD3 violates 2NF because nonprime attribute Ename can be functionally determined by only Ssn, and both nonprime attributes Pname and Plocation can be functionally determined by only Pnumber.
- 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.



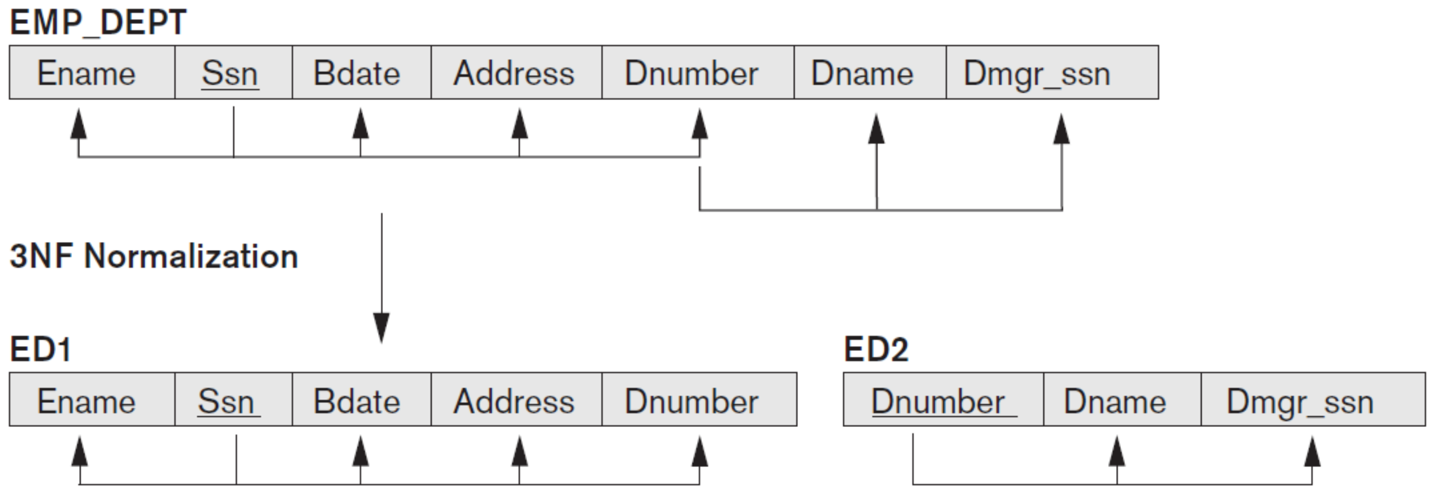
Third Normal Form (3NF) (1/2)

- **Third normal form (3NF)** is based on the concept of 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.
- A relation schema R is in **3NF** if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.
- For example, the dependency Ssn → Dmgr_ssn is transitive through Dnumber in EMP_DEPT, because both the dependencies Ssn → Dnumber and Dnumber → Dmgr_ssn hold and Dnumber is neither a key itself nor a subset of the key of EMP_DEPT.

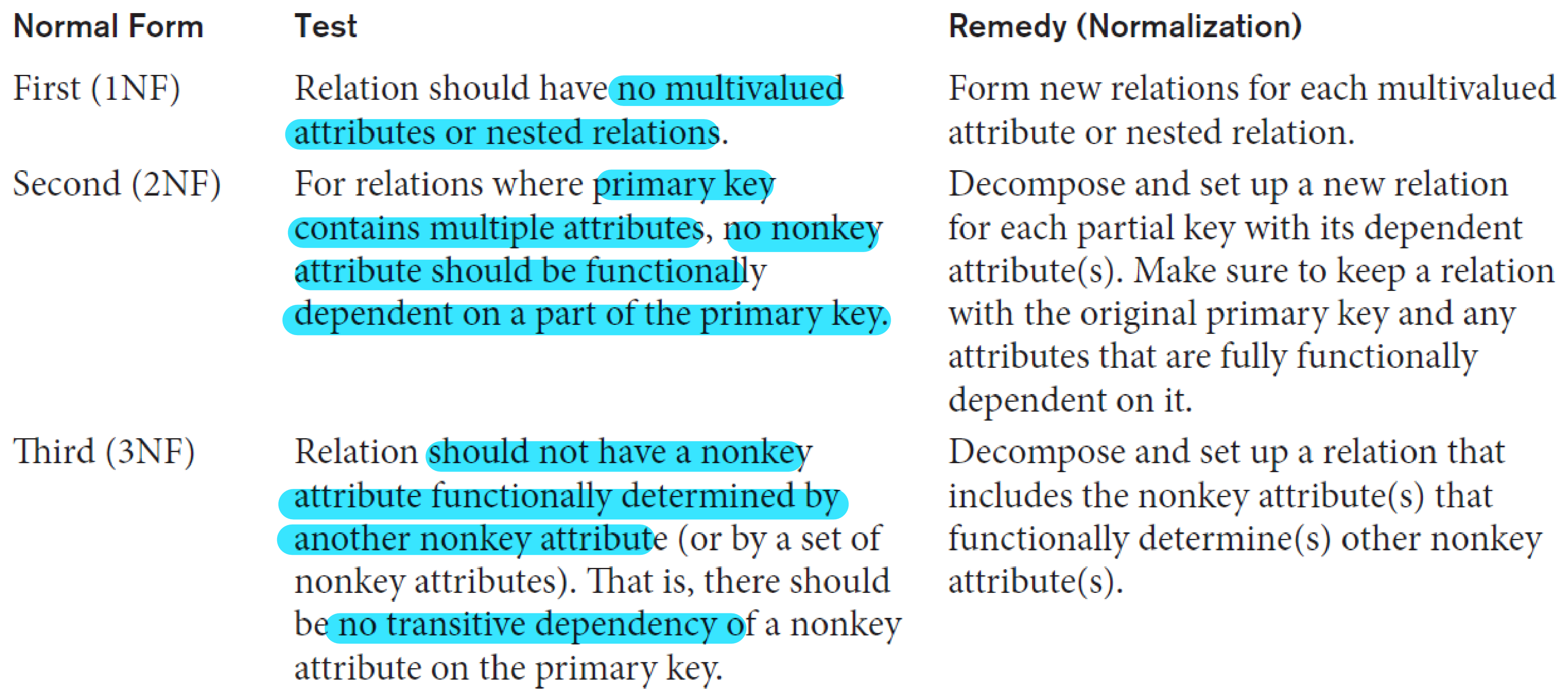


Third Normal Form (3NF) (2/2)

- We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2.
- Intuitively, we see that ED1 and ED2 represent independent facts about employees and departments, both of which are entities in their own right.



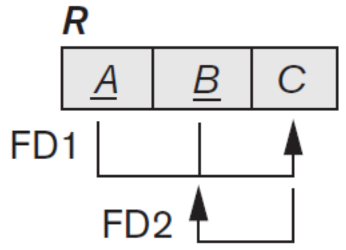
Summary of Normal Forms Based on Primary Keys and Corresponding Normalization



Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or 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 a relation with 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 includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Boyce-Codd Normal Form (1/4)

- **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.
- A relation schema R is in **BCNF** if whenever a nontrivial functional dependency X → A holds in R, then X is a superkey of R.
- For example, this schematic relation with 2 FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.

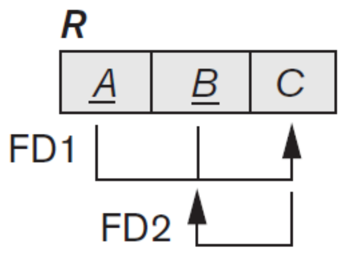


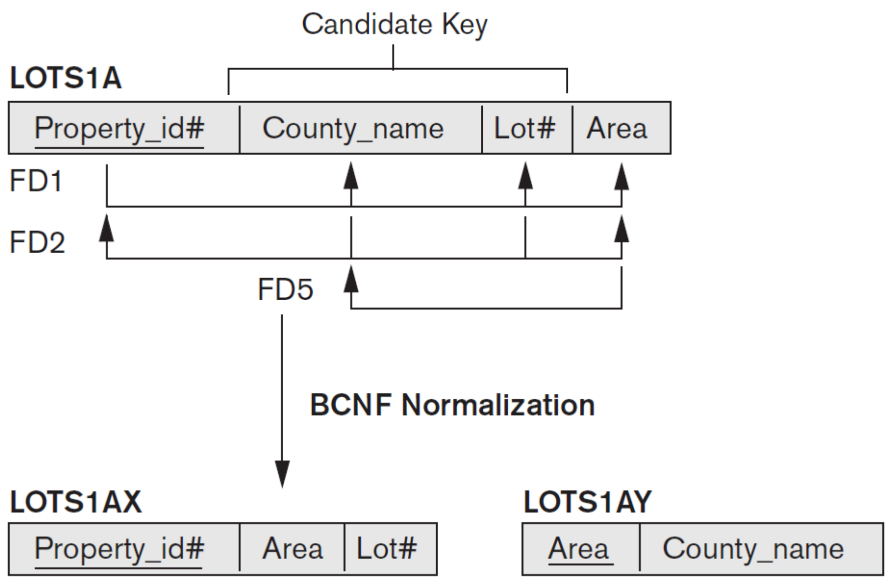
Boyce-Codd Normal Form (2/4)

- In general, a relation R not in BCNF can be decomposed so as to meet the nonadditive join property by the following procedure. It decomposes R successively into a set of relations that are in BCNF:
  - Let R be the relation not in BCNF, let $X \subseteq R$, and let $X \rightarrow A$ be the FD that causes a violation of BCNF. R may be decomposed into two relations:
    - R–A
    - XA
  - If either R–A or XA is not in BCNF, repeat the process.

Boyce-Codd Normal Form (3/4)

- For the example, R={A, B, C} and C → B violates BCNF.
  - R is decomposed into two relations (R-A)={A, C} and (XA)={C, B}, as X={C} and A={B}
- In another example, FD5 violates BCNF in LOTS1A because Area is not a superkey of LOTS1A. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.
  - R={Property_id#, Country_name, Lot#, Area}, X={Area} and A={Country_name}
  - (R-A) = {Property_id#, Lot#, Area} and (XA) = {Area, Country_name}





Boyce-Codd Normal Form (4/4)

- Let us consider the relation scheme R={A,B,C,D,E} and the FDs: {A} → {B,E}, {C} → {D}
- Candidate key: AC
- Both functional dependencies violate BCNF because the LHS is not a candidate key
- Pick {A} → {B,E}
  - We can also choose {C} → {D} – different choices lead to different decompositions
  - R={A,B,C,D,E} generates R_1={A,C,D} and R_2={A,B,E}
- We need to decompose R_1={A,C,D} because of FD {C} → {D}, so R_1 is further decomposed to R_3={A,C} and R_4={C,D}.
- Final decomposition: R_2={A,B,E}, R_3={A,C}, R_4={C,D}