**1 Informal Design Guidelines for Relational Databases:-** Semantics of the Relation Attributes,Redundant Information in Tuples and Update Anomalies, Null Values in Tuples , Spurious Tuples

**2 Functional Dependencies (FDs):** Definition of FD, Inference Rules for FDs, Equivalence of Sets of FDs, Minimal Sets of FDs

**3 Normal Forms Based on Primary Keys:- N**ormalization of Relations , Practical Use of Normal Forms , Definitions of Keys and Attributes Participating in Keys , First Normal Form, Second Normal Form, Third Normal Form

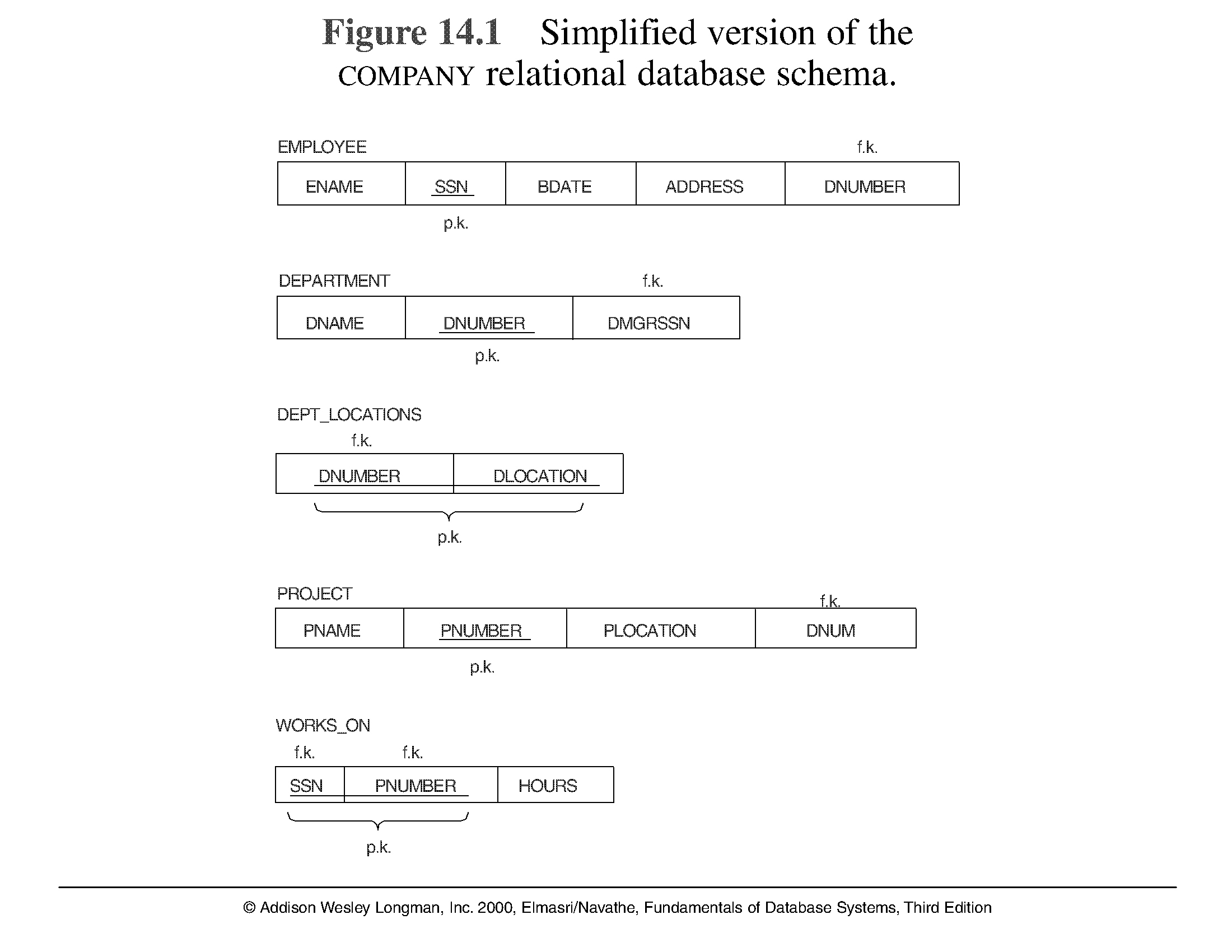
**4 General Normal Form Definitions (For Multiple Keys)**

**5 BCNF (Boyce-Codd Normal Form)**

**GUIDELINE 1:** Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).

* Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
* Only foreign keys should be used to refer to other entities
* Entity and relationship attributes should be kept apart as much as possible.

*Bottom Line:* Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.



**1.2 Redundant Information in Tuples and Update Anomalies**

* Mixing attributes of multiple entities may cause problems
* Information is stored redundantly wasting storage
  + Problems with update anomalies: Insertion anomalies, Deletion anomalies , Modification anomalies

**EXAMPLE OF AN UPDATE ANOMALY**

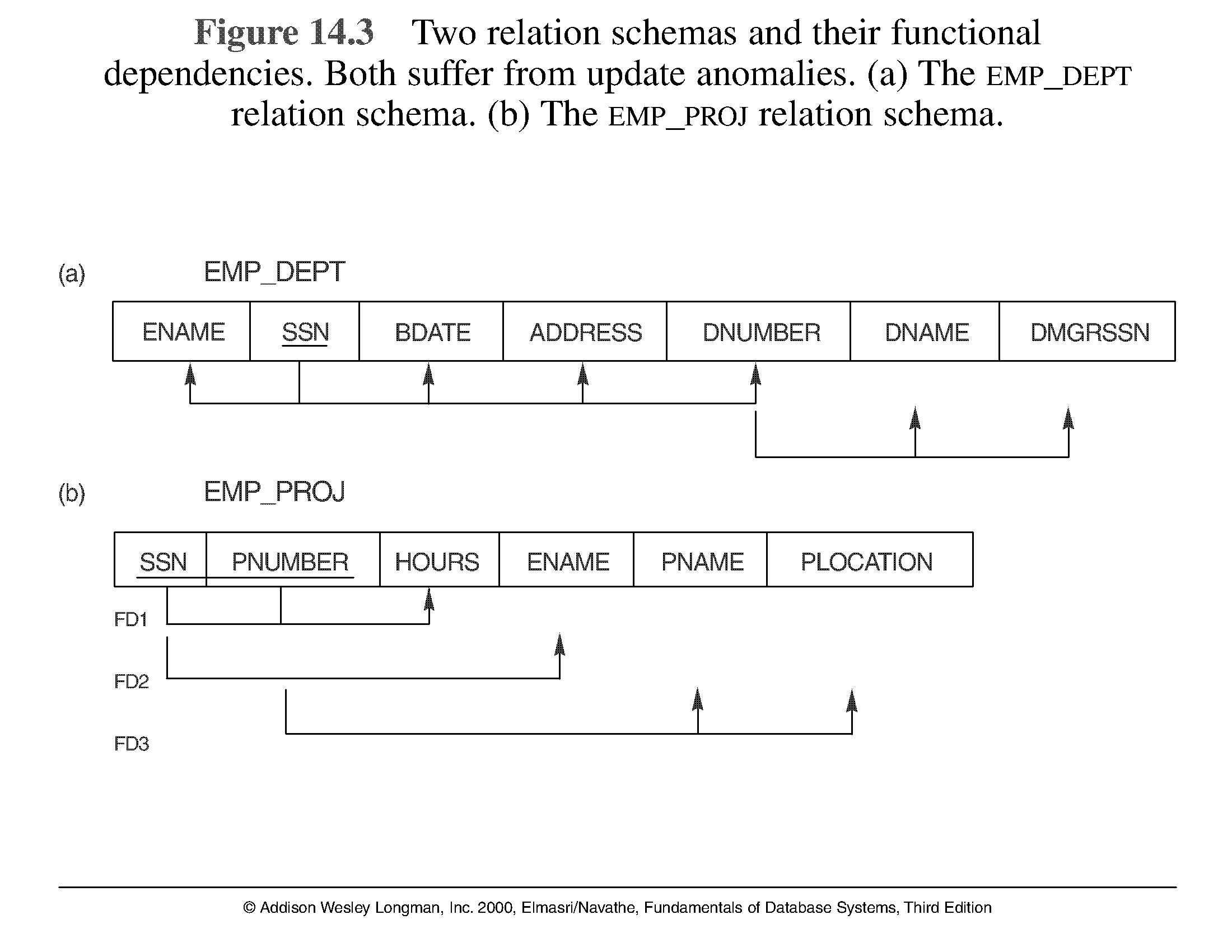
Consider the relation: EMP\_PROJ ( Emp#, Proj#, Ename, Pname, No\_hours)

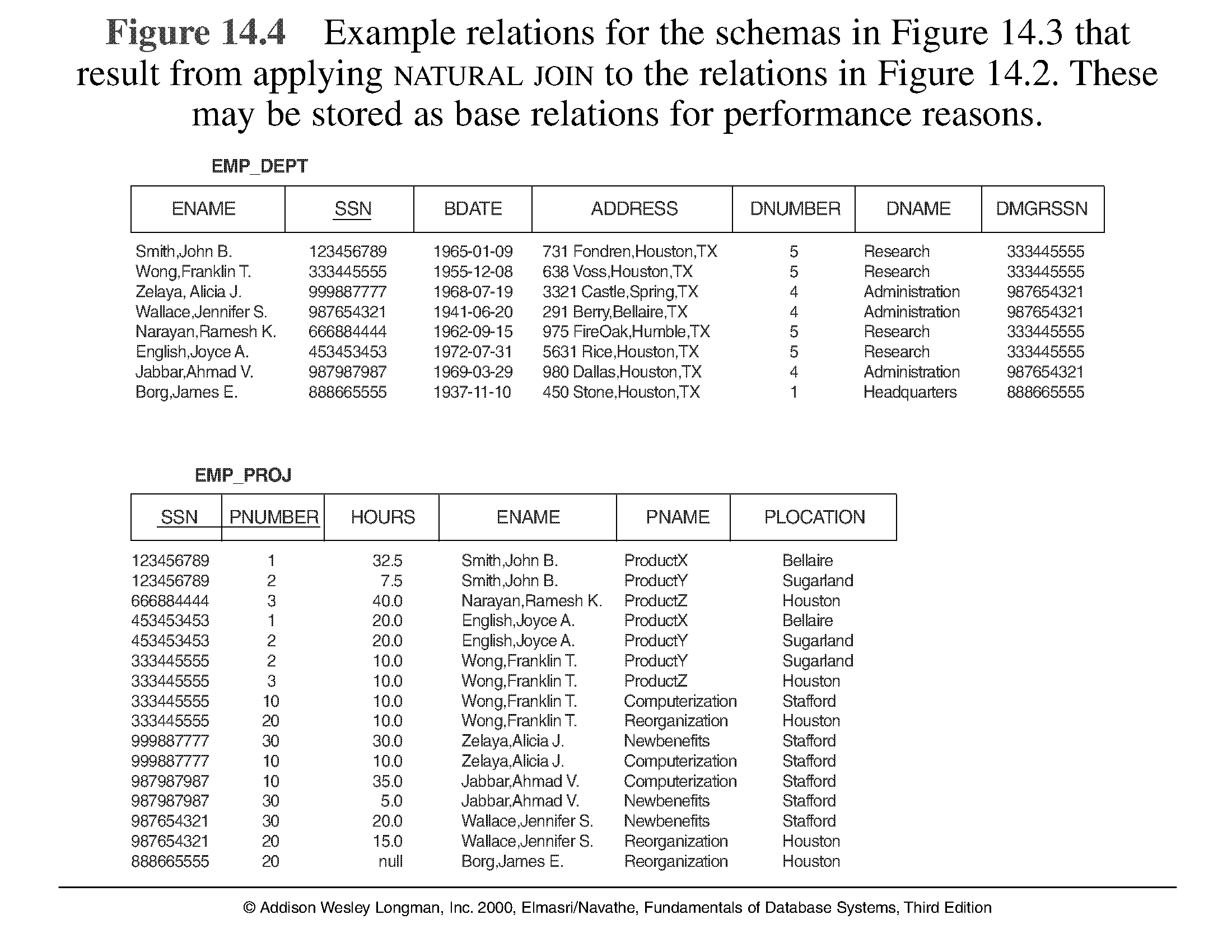
**Update Anomaly:** Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

* **Insert Anomaly:** Cannot insert a project unless an employee is assigned to .

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

* **Delete Anomaly:** When a project is deleted, it will result in deleting all the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.





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

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

* Attributes that are NULL frequently could be placed in separate relations (with the primary key)
* Reasons for nulls:
  + attribute not applicable or invalid , attribute value unknown (may exist) ,value known to exist, but unavailable

**1.4 Spurious Tuples**

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

**GUIDELINE 4:** The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join of any relations.

 There are two important properties of decompositions:

1. non-additive or losslessness of the corresponding join
2. preservation of the functional dependencies.

Note that property (a) is extremely important and *cannot* be sacrificed. Property (b) is less stringent and may be sacrificed. (See Chapter 11).

**2.1 Functional Dependencies**

* Functional dependencies (FDs) are used to specify *formal measures* of the "goodness" of relational designs
* FDs and keys are used to define **normal forms** for relations
* FDs are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes
* A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y
* X -> Y holds 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)
* Written as X -> Y; can be displayed graphically on a relation schema as in Figures. ( denoted by the arrow: ).
* FDs are derived from the real-world constraints on the attributes

**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
* 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])

**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

**Armstrong's inference rules:**

IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y

IR2. (**Augmentation**) If X -> Y, then XZ -> YZ , (Notation: XZ stands for X U Z)

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

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

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)
* **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

**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**

* 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 subset-of X) and still have a set of dependencies that is equivalent to F.
4. Every set of FDs has an equivalent minimal set, There can be several equivalent minimal sets
5. There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
6. 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)

**Normal Forms Based on Primary Keys**

The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**. The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.* Initially, Codd proposed three normal forms, which he called first, second, and third normal form. A stronger def-inition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd. All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation. Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies.

**Normalization of data** can be considered a process of analyzing the given relationschemas based on their FDs and primary keys to achieve the desirable properties of minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies. It can be considered as a “filtering” or “purifi-cation” process to make the design have successively better quality. 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. Thus, the normalization procedure provides database design-ers with the following:

* 1. A formal framework 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 carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree

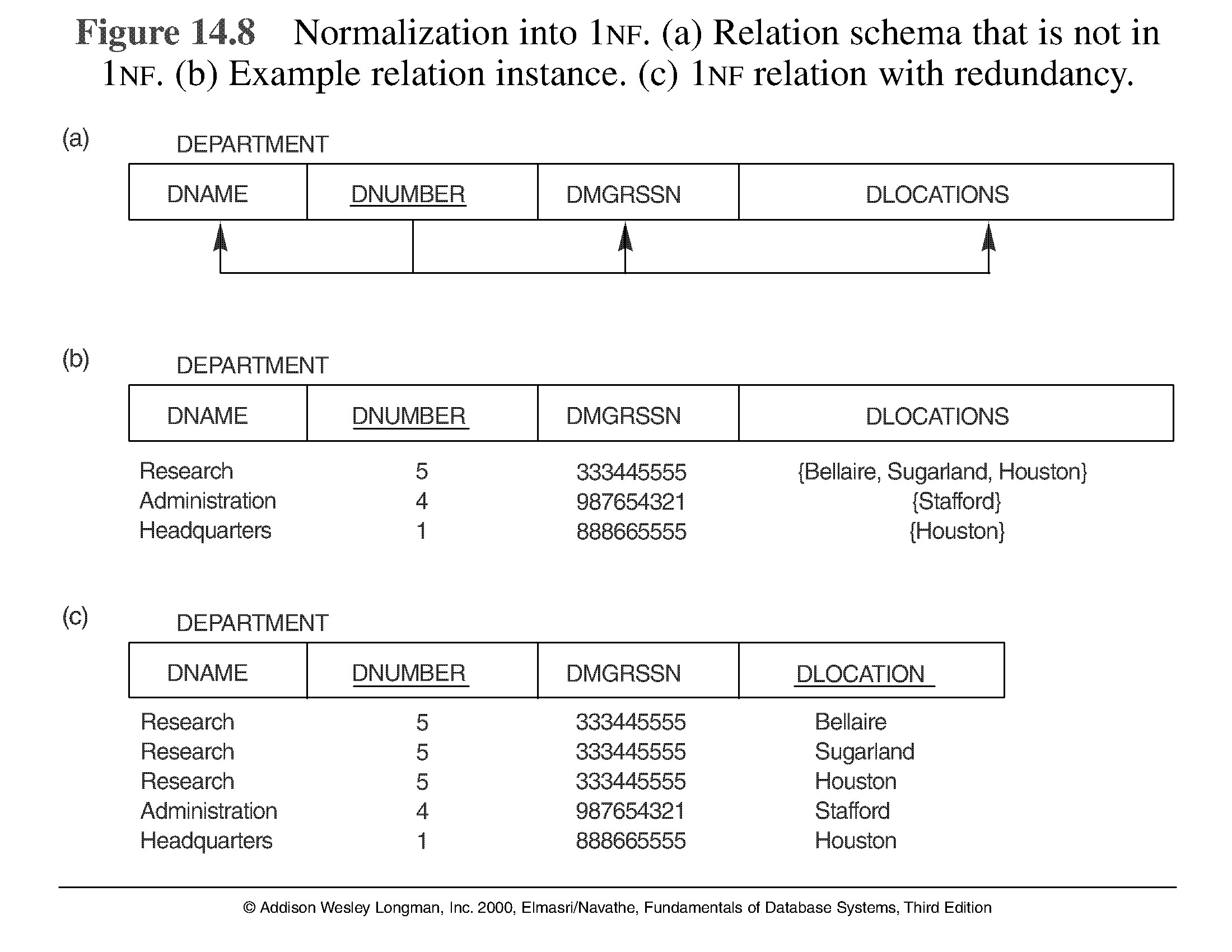
**Definition.** The **normal form** of a relation refers to the highest normal formcondition that it meets, and hence indicates the degree to which it has been nor-malized.

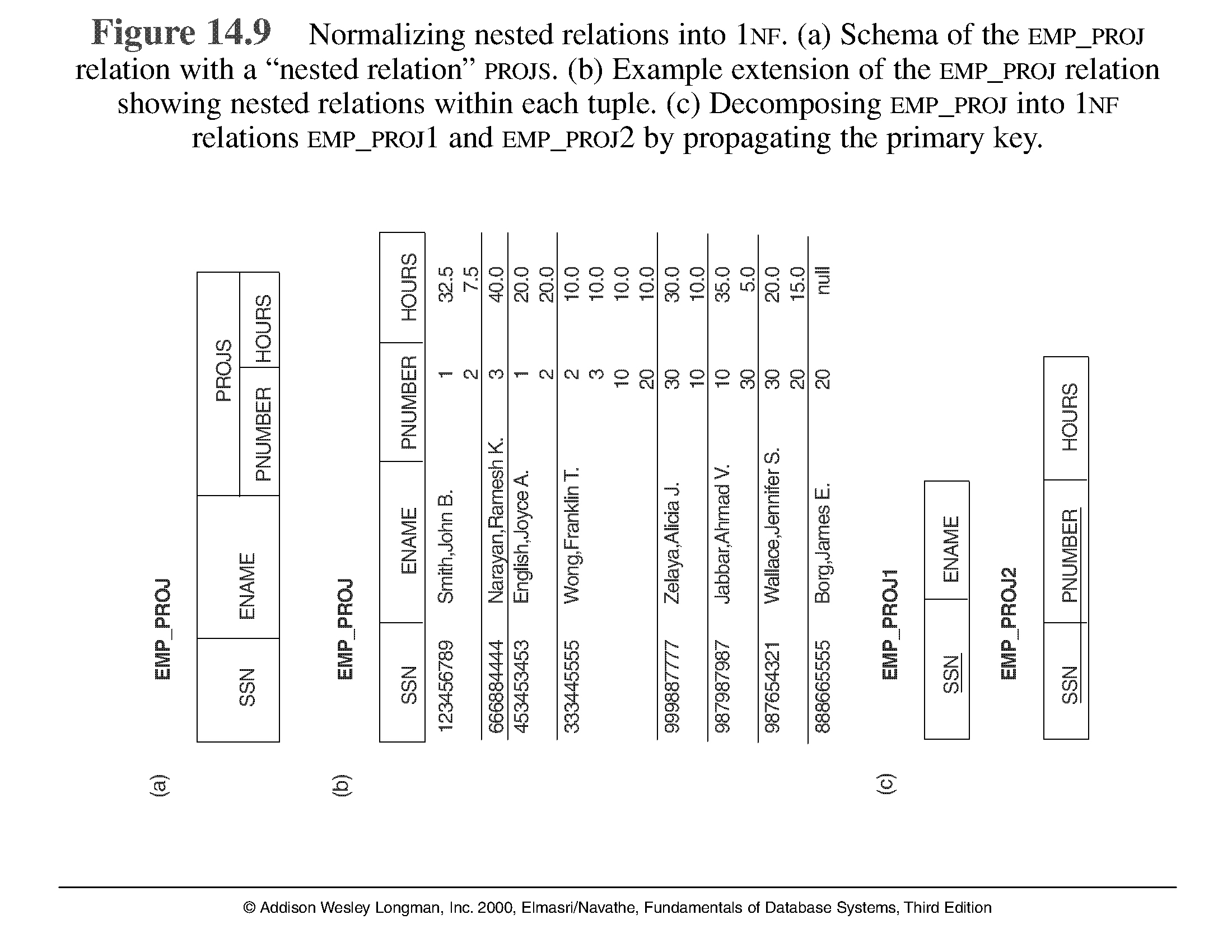
* **Normalization**: The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
* **Normal form**: Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
* 2NF, 3NF, BCNF based on keys and FDs of a relation schema
* 4NF based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs (Chapter 11)
* Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; Chapter 11)
* **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
* The practical utility of these normal forms becomes questionable when the constraints on which they are based are **hard to understand** or to **detect**
* 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
* 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.
* 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 bea member of *some candidate key*
* A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

**3.2 First Normal Form**

**First normal form (1NF**) is now considered to be part of the formal definition of arelation in the basic (flat) relational model; historically, it was defined to disallow 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. 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 dis-allows *relations within relations* or *relations as attribute values within tuples*. The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.

* Disallows composite attributes, multivalued attributes, and **nested relations**; attributes whose values *for an individual tuple* are non-atomic
* Considered to be part of the definition of relation





**3.3 Second Normal Form**

* Uses the concepts of **FD**s, **primary key**
* **Second normal form (2NF)** is based on the concept of*full functional dependency.*Afunctional dependency *X* → *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* ε *X*, (*X* – {*A*}) does *not* functionally determine *Y*. A functional dependency *X* → *Y* is a **partial dependency** if some attribute *A* ε *X* can be removed from *X* and the dependency still holds; that is, for some *A* ε *X*, (*X* – {*A*}) → *Y*. In Figure 15.3(b), {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber → Hours holds). However, the dependency {Ssn, Pnumber} → Ename is partial because Ssn → Ename holds.
* **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*.

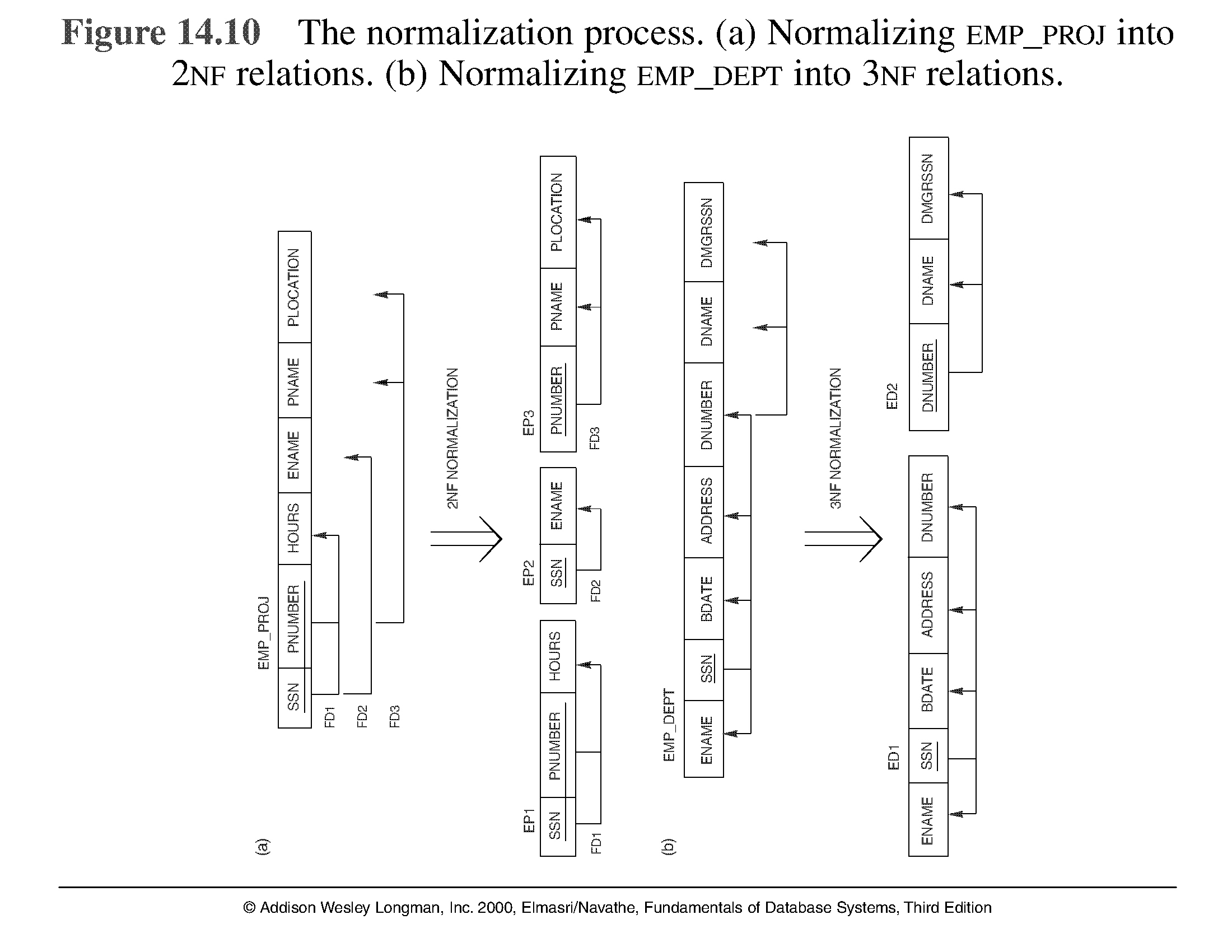
Definitions:

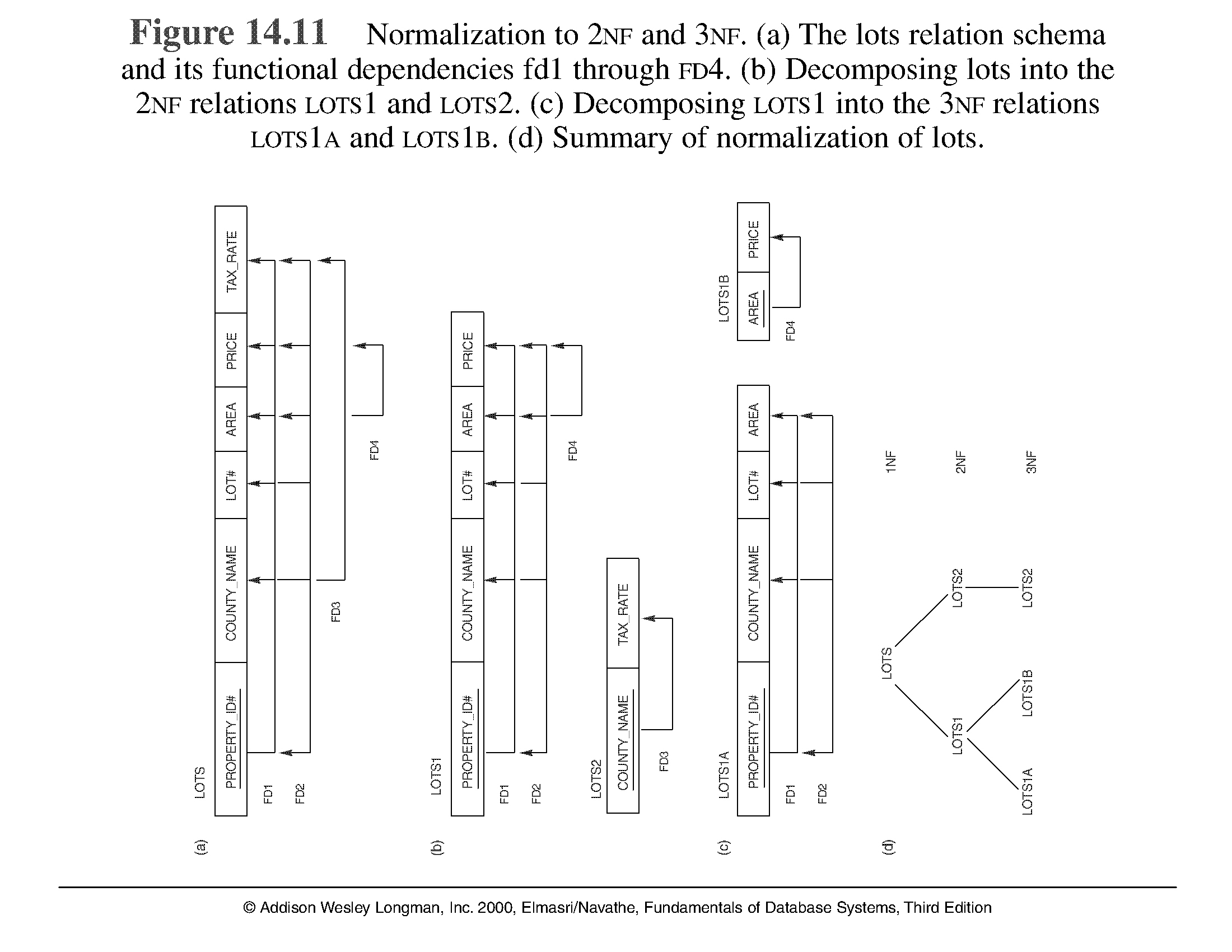
* **Prime attribute** - attribute that is member of the primary key K
* **Full functional dependency** - a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more

Examples: - {SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold

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

* 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
* R can be decomposed into 2NF relations via the process of 2NF normalization





**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**

**Third normal form (3NF)** is based on the concept of*transitive dependency*. Afunctional 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*,10 and both *X* → *Z* and *Z* → *Y* hold. The dependency Ssn → Dmgr\_ssn is transitive through Dnumber in EMP\_DEPT

* 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:**

InX -> 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.

**General Normal Form Definitions**

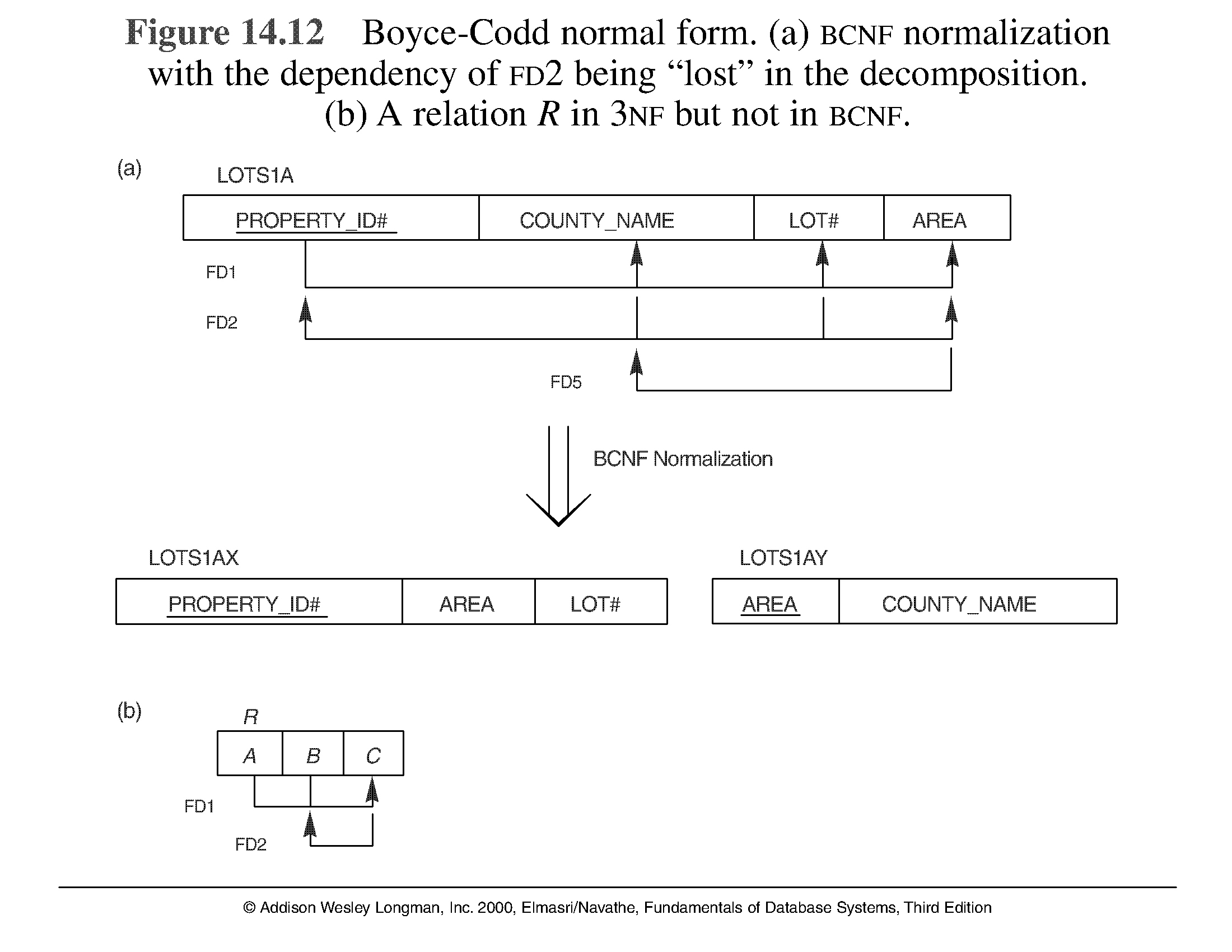
**Definition:**

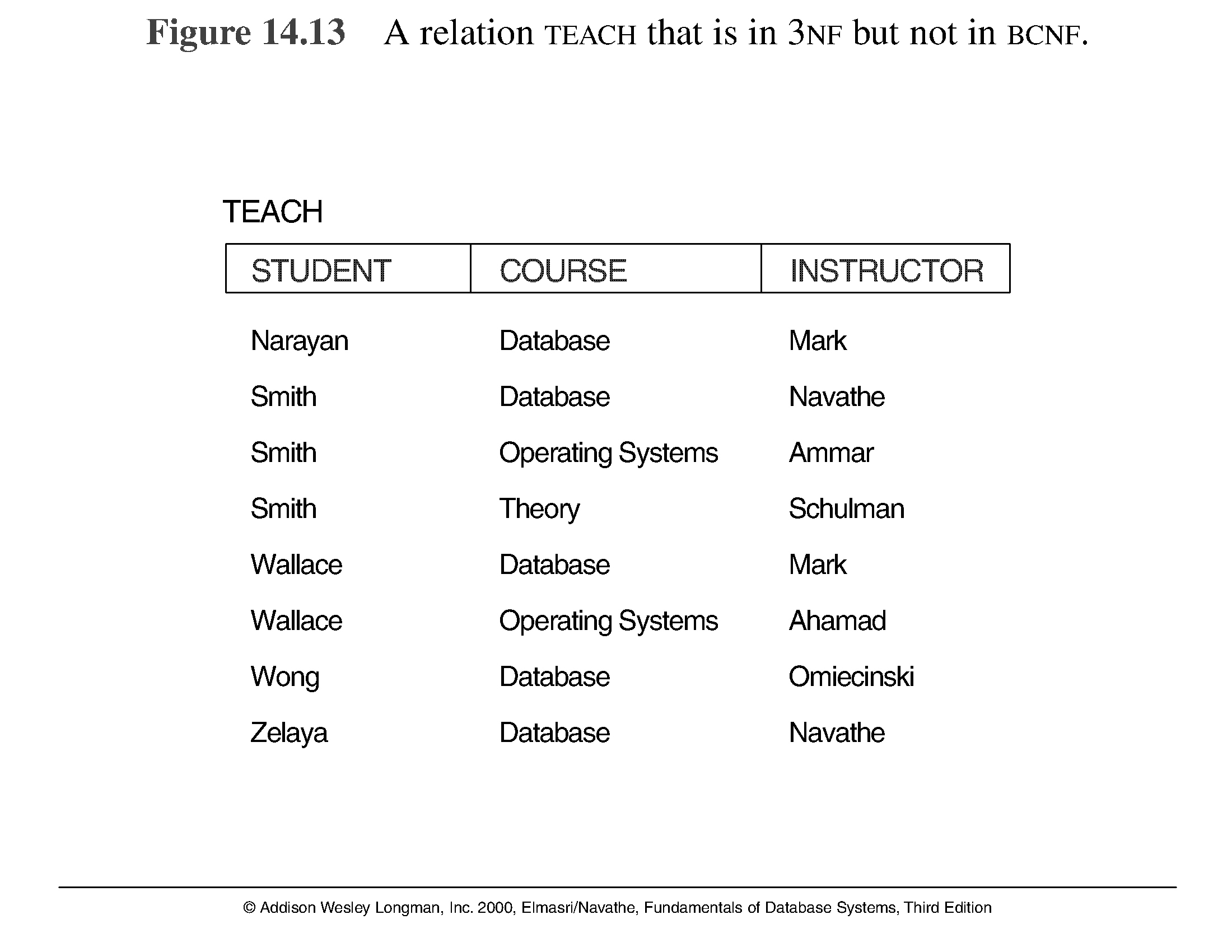
* Superkey of relation schema R - a set of attributes S of R that contains a key of R
* A relation schema R is in third normal form (3NF) if whenever a FD X -> A holds in R, then either:

(a) X is a superkey of R, or , (b) A is a prime attribute of R

NOTE: Boyce-Codd normal form disallows condition (b) above

**5 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
* Each normal form is strictly stronger than the previous one
  + Every 2NF relation is in 1NF, Every 3NF relation is in 2NF, Every BCNF relation is in 3NF
* There exist relations that are in 3NF but not in BCNF , The goal is to have each relation in BCNF (or 3NF) 



**Achieving the BCNF by Decomposition**

* Two FDs exist in the relation TEACH: fd1: { student, course} -> instructor

fd2: instructor ->course

* {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 10.12 (b). So this relation is in 3NF but not in BCNF
* A relation NOT in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations. (See Algorithm 11.3)

**Achieving the BCNF by Decomposition**

* Three possible decompositions for relation TEACH
  1. {student, instructor} and {student, course}
  2. {course, instructor } and {course, student}
  3. {instructor, course } and {instructor, student}
* All three decompositions will lose fd1. We have to settle for sacrificing the functional dependency preservation. But we cannot sacrifice the non-additivity property after decomposition.
* Out of the above three, only the 3rd decomposition will not generate spurious tuples after join.(and hence has the non-additivity property).
* A test to determine whether a binary decomposition (decomposition into two relations) is nonadditive (lossless) is discussed in section 11.1.4 under Property LJ1. Verify that the third decomposition above meets the property.