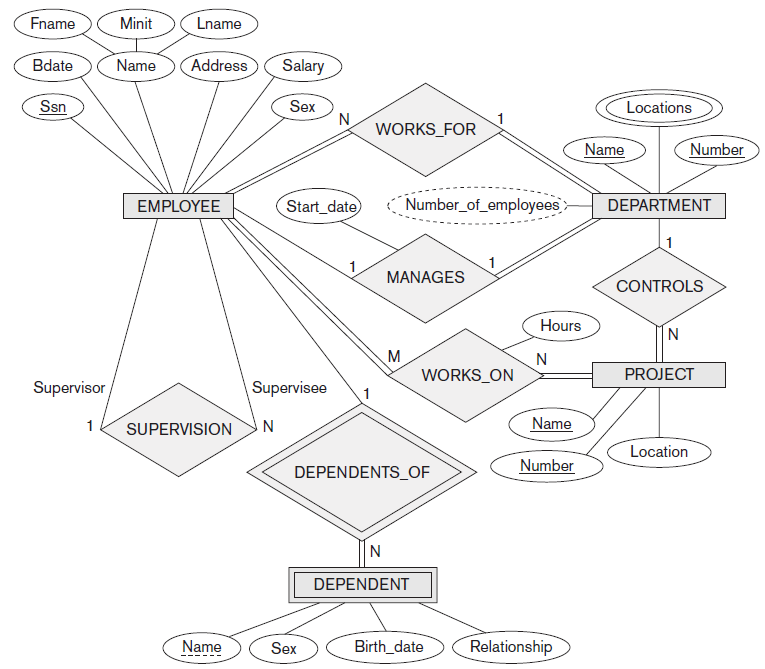
A database represents some aspect of the real world, sometimes called the **miniworld** or the **universe of discourse** (UoD).

requirements specification and analysis 🡪 conceptual design (er-model) 🡪 logical design (data model implemented in a commercial DBMS) 🡪 physical design

Advantages of using a DBMS approach: controlling redundancy 🡪 single logical update, no storage space wasted, data consistency 🡪 data normalization.

Sometimes controlled redundancy to improve the performance of the queries 🡪 denormalization.

**Data Modeling Using the Entity–Relationship (ER) Model**



**Composite attribute**: Address = 2311 Kirby Houston, Texas 77001

**Simple (Atomic) Attributes**: attributes that are not divisible

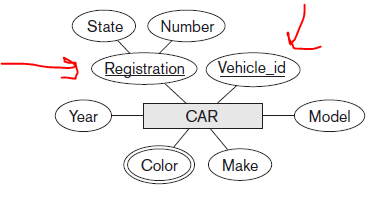
**Multivalued Attributes**: a color attribute of a two-tone car .

The Age attribute is called a **derived attribute** and is said to be derivable from the Birth\_date attribute, which is called a **stored attribute**.

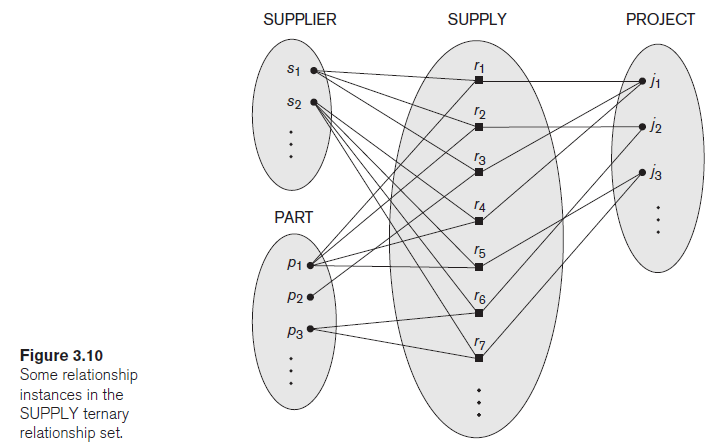
**Complex Attributes**. composite and multivalued attributes can be nested arbitrarily.

Sometimes several attributes together form a key. The proper way to represent this in the ER model is to define a composite attribute and designate it as a key attribute of the entity type.

In our diagrammatic notation, if two attributes are underlined separately, then each is a key on its own. Unlike the relational model, there is no concept of primary key in the ER model that we present here; the primary key will be chosen during mapping to a relational schema.



The **degree** of a relationship type is the number of participating entity types.

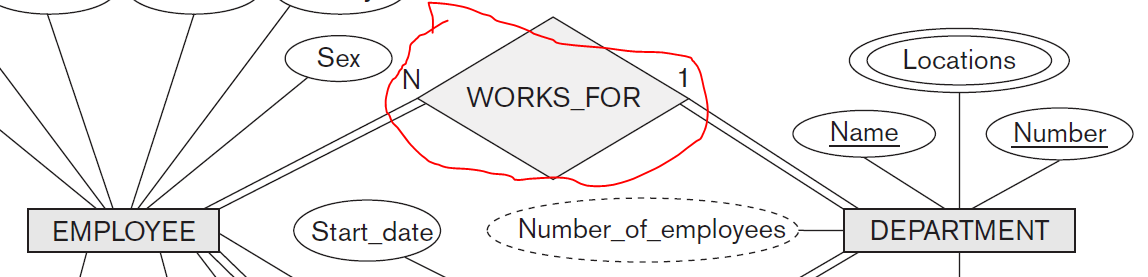


**Role Names and Recursive Relationships**. Each entity type that participates in a relationship type plays a particular role in the relationship. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means. Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships** or **self-referencing relationships.**

**Cardinality Ratios for Binary Relationships:** WORKS\_FOR 🡪 DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

Come faccio a definire la cardinalità di WORKS\_FOR: DEPARTMENT:EMPLOYEE?

1. Considero 1 Department
2. Come è relazionato con Employee? Per 1 Deparment ci sono N Employee
3. Ricavo la cardinalità di Employee 🡪 N
4. Considero 1 Employee
5. Come è relazionato don DEPARTMENT? 1 Employee può lavorare solo per 1 Department
6. Ricavolo la cardinalità di department 🡪 1
7. Cardinalità della relazione WORKS\_FOR 🡪 DEPARTMENT:EMPLOYEE? 1: N



**Participation Constraints (minimum cardinality constraint):** There are two types of participation constraints: total and partial**. Total participation**, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS\_FOR. Total participation is also called **existence dependency**.

Come faccio a definire i vincoli di partecipazione?

1. Esiste 1 DEPARTMENT non relazionato ad alcun EMPLOYEE?
2. Se si, partecipazione parziale
3. Se no, partecipazione totale

cardinality ratio + participation constraints = **structural constraints** of a relationship type.

In ER diagrams, total participation is displayed as a *double line* connecting the participating entity type to the relationship, whereas partial participation is represented by a *single line*. Notice that in this notation, we can either specify no minimum (partial participation) or a minimum of one (total participation). An alternative notation allows the designer to specify a specific *minimum number* on participation in the relationship, such as 4 or 5.

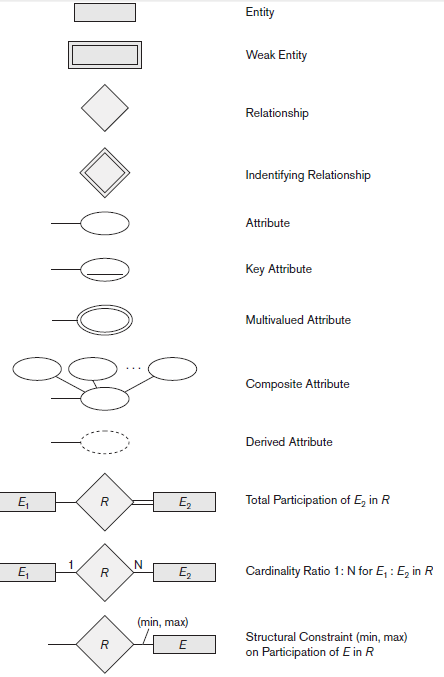
**Relationship types can also have attributes**. The Start\_date attribute for the MANAGES relationship can be an attribute of either EMPLOYEE (manager) or DEPARTMENT, although conceptually it belongs to MANAGES. This is because MANAGES is a 1:1 relationship. For a 1:N relationship type, a relationship attribute can be migrated *only* to the entity type on the N-side of the relationship (EMPLOYEE). For M:N (many-to-many) relationship types Such attributes *must be specified as relationship attributes*. An example is the Hours attribute of the M:N relationship WORKS\_ON.

Entity types that do not have key attributes of their own are called **weak entity types**. Weak entity types are related to the **identifying** or **owner entity type** through the **identifying relationship**. A weak entity type always has a *total participation constraint* with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are *related to the same owner entity*. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key. In the worst case, a composite attribute of *all the weak entity’s attributes* will be the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.

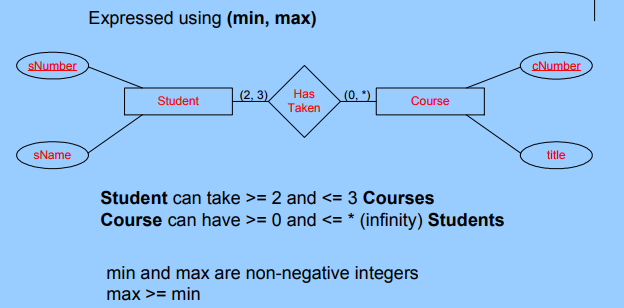
Weak entity types can sometimes be represented as complex (composite, multivalued) attributes. In the preceding example, we could specify a multivalued attribute Dependents for EMPLOYEE, which is a multivalued composite attribute with the component attributes Name, Birth\_date, Sex, and Relationship. The choice of which representation to use is made by the database designer.

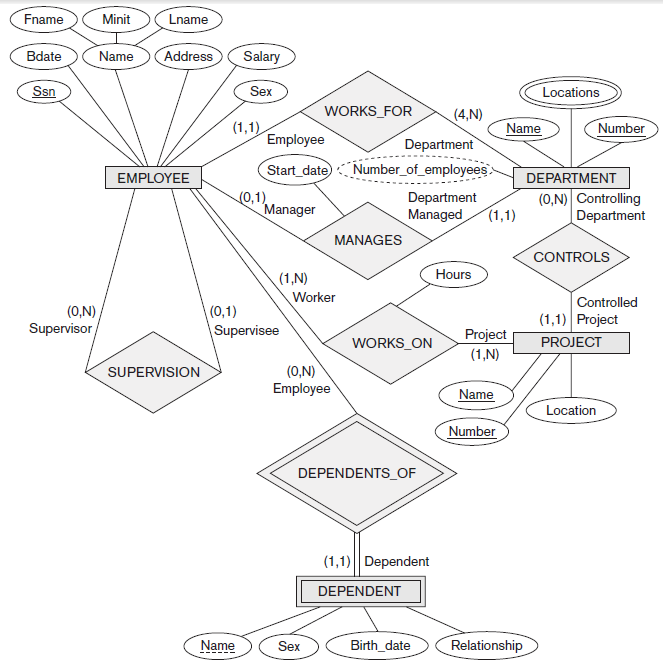
Binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom.



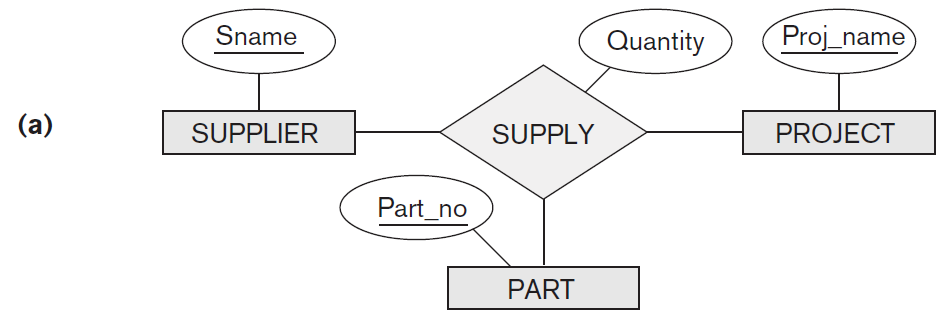
**Alternative Notations for ER Diagrams**

The (min, max) notation is more precise, and we can use it to specify some structural constraints for relationship types of *higher degree*. However, it is not sufficient for specifying some key constraints on higherdegree relationships.

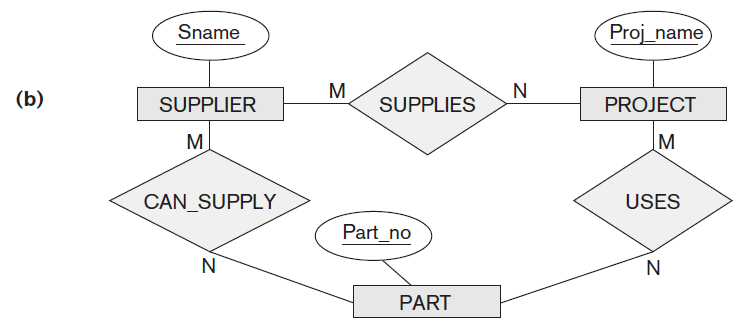




**Relationship Types of Degree Higher than Two**

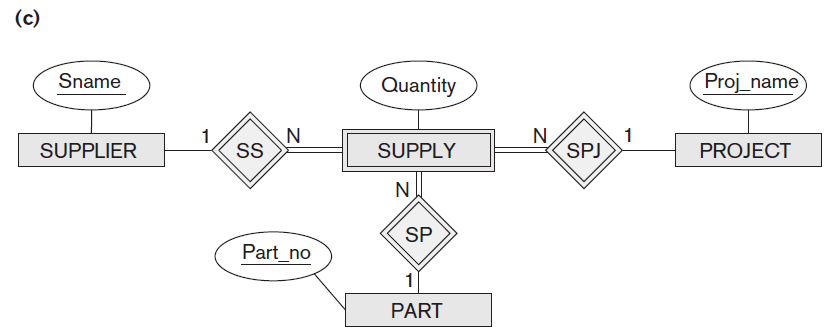


In general, a ternary relationship type represents different information than do three binary relationship types.



The existence of three relationship instances (*s*, *p*), (*j*, *p*), and (*s*, *j*) in CAN\_SUPPLY, USES, and SUPPLIES, respectively, does not necessarily imply that an instance (*s*, *j*, *p*) exists in the ternary relationship SUPPLY, because the *meaning is different*. It is often tricky to decide whether a particular relationship should be represented as a relationship type of degree *n* or should be broken down into several relationship types of smaller degrees. The designer must base this decision on the semantics or meaning of the particular situation being represented. The typical solution is to include the ternary relationship plus one or more of the binary relationships, if they represent different meanings and if all are needed by the application.

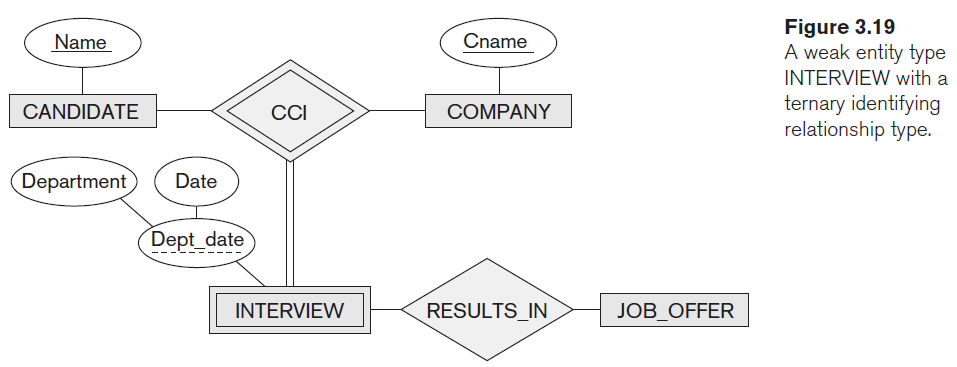
Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship such as SUPPLY must be represented as a weak entity type, with no partial key and with three identifying relationships. The three participating entity types SUPPLIER, PART, and PROJECT are together the owner entity types (see Figure 3.17(c)). Hence, an entity in the weak entity type SUPPLY in Figure 3.17(c) is identified by the combination of its three owner entities from SUPPLIER, PART, and PROJECT.



It is also possible to represent the ternary relationship as a regular entity type by introducing an artificial or surrogate key. In this example, a key attribute Supply\_id could be used for the supply entity type, converting it into a regular entity type.

Three binary N:1 relationships relate SUPPLY to each of the three participating entity types.

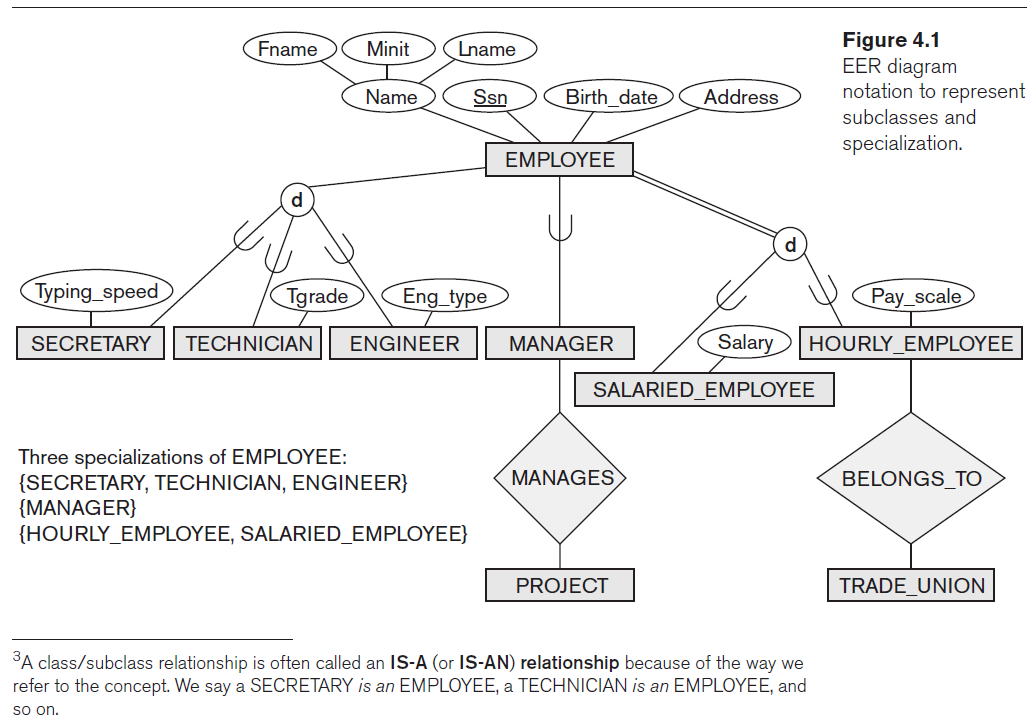
Notice that it is possible to have a weak entity type with a ternary (or *n*-ary) identifying relationship type. In this case, the weak entity type can have *several* owner entity types:



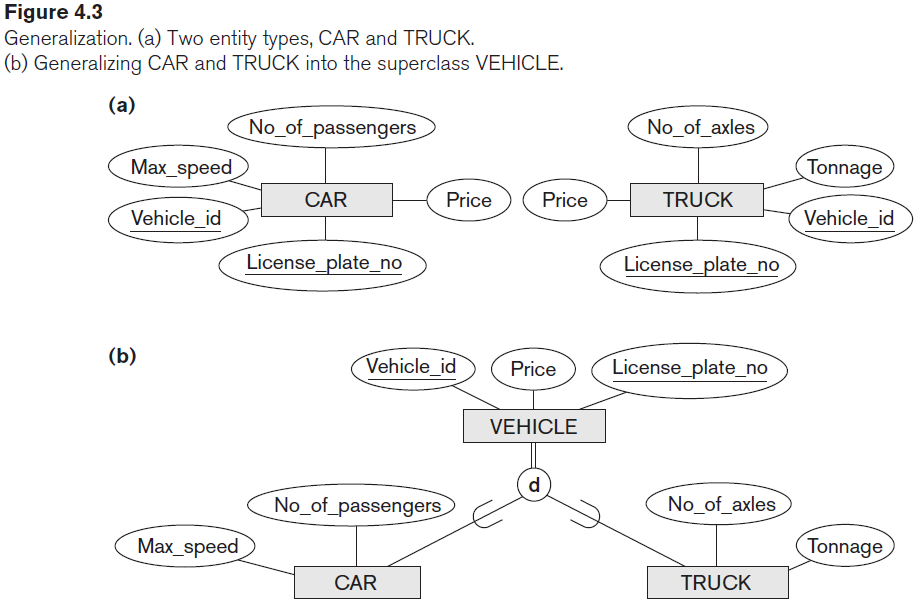
Descrivere vinconli strutturali in realzioni di oridine superiore a 2 non l’ho capito come si faccia. Provo a capirlo a fine corso.

**The Enhanced Entity–Relationship (EER) Model**

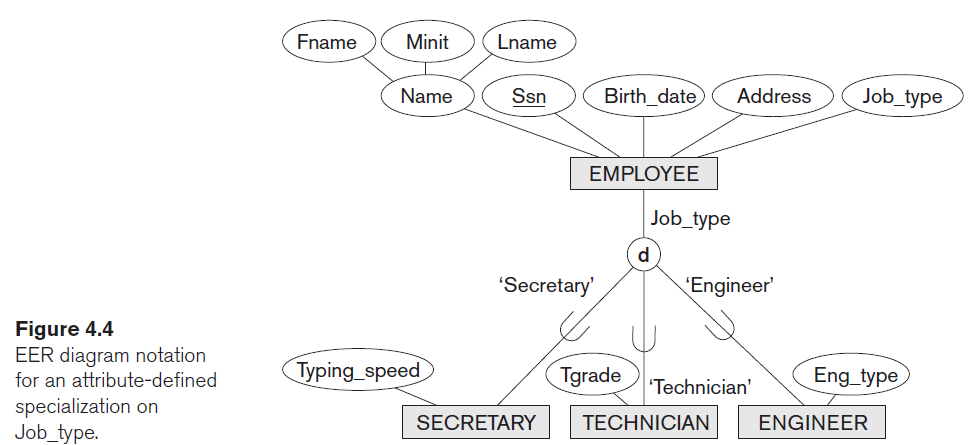
**Specialization** is the process of defining a set of subclasses of an entity type; this entity type is called the **superclass** of the specialization. For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE.



**Generalization**



**Constraints on Specialization and Generalization**



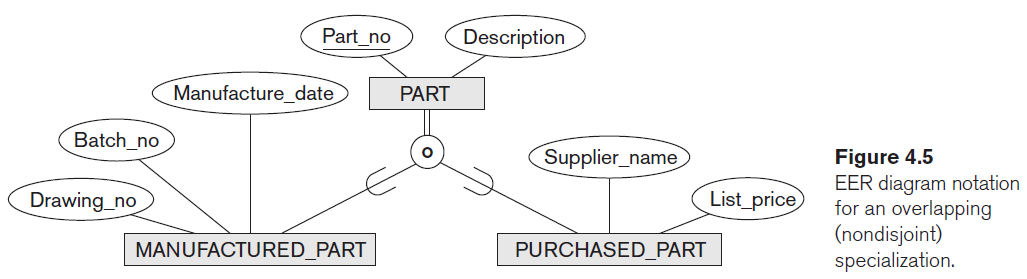
**Predicate-defined (or condition-defined) subclasses**. If the EMPLOYEE entity type has an attribute Job\_type, we can specify the condition of membership in the SECRETARY subclass by the condition (Job\_type = ‘Secretary’), which we call the defining predicate of the subclass. This condition is a constraint specifying that exactly those entities of the EMPLOYEE entity type whose attribute value for Job\_type is ‘Secretary’ belong to the subclass. We display a predicate-defined subclass by writing the predicate condition next to the line that connects the subclass to the specialization circle.

If all subclasses in a specialization have their membership condition on the same attribute of the superclass, the specialization itself is called an **attribute-defined specialization**, and the attribute is called the defining attribute of the specialization. We display an attribute-defined specialization by placing the defining attribute name next to the arc from the circle to the superclass.

When we do not have a condition for determining membership in a subclass, the subclass is called **user-defined**. Membership in such a subclass is determined by the database users when they apply the operation to add an entity to the subclass; hence, membership is specified individually for each entity by the user, not by any condition that may be evaluated automatically.

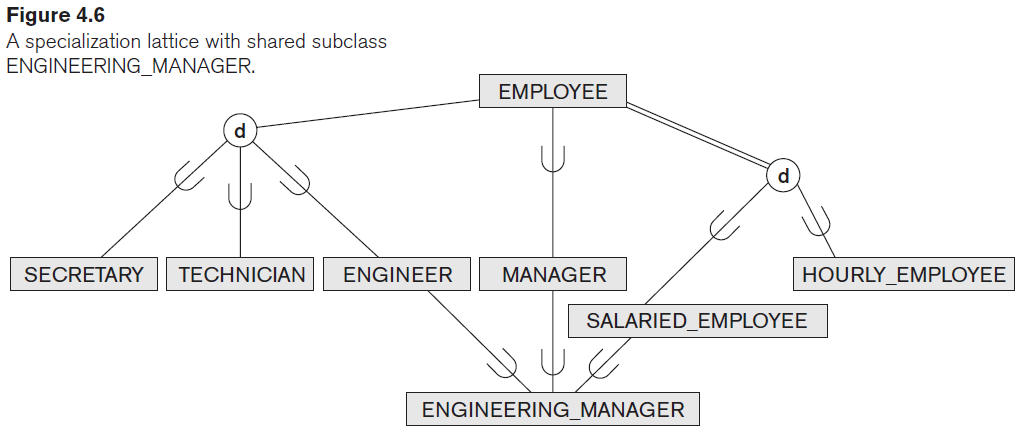
**Disjointness constraint**: an entity can be a member of at most one of the subclasses of the specialization. A specialization that is attribute-defined implies the disjointness constraint. Thee d in the circle stands for disjoint

If the subclasses are not constrained to be disjoint, their sets of entities may be **overlapping**; that is, the same (real-world) entity may be a member of more than one subclass of the specialization. This case, which is the default, is displayed by placing an o in the circle



**completeness (or totalness) constraint**: A total specialization constraint specifies that every entity in the superclass must be a member of at least one subclass in the specialization. This is shown in EER diagrams by using a double line to connect the superclass to the circle. A single line is used to display a partial specialization.

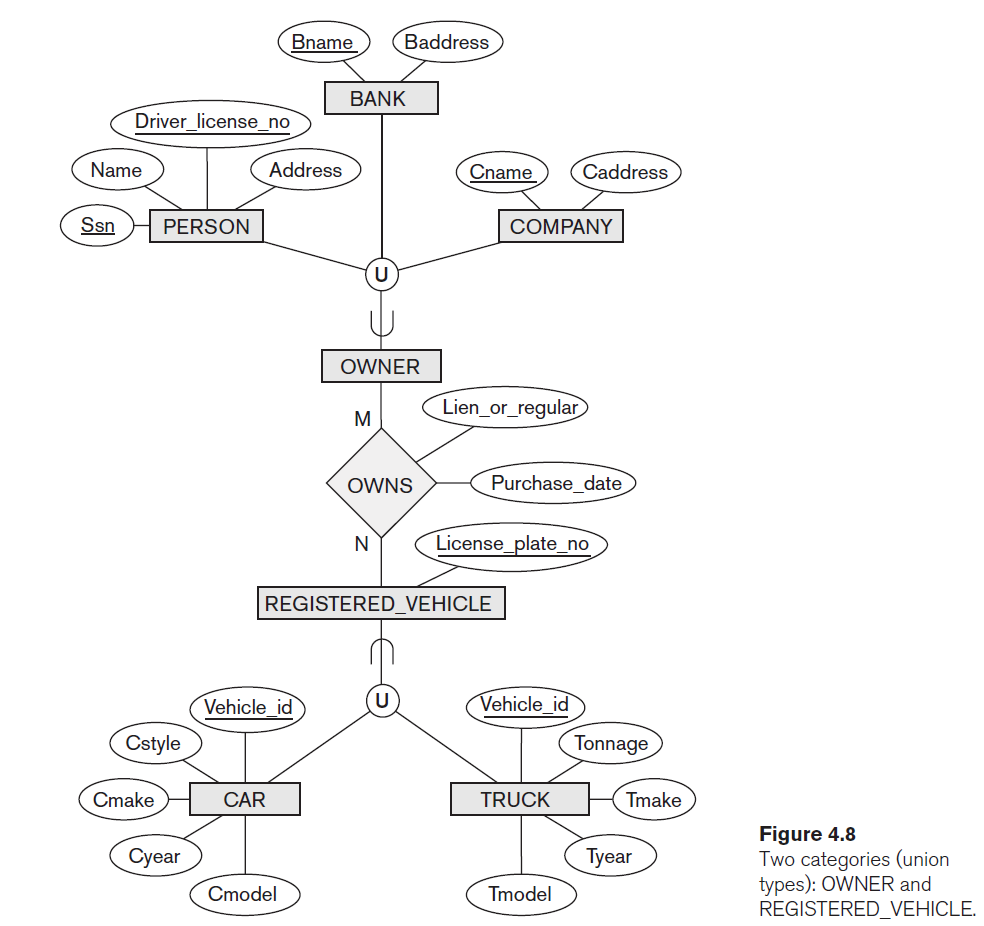
A **specialization hierarchy** has the constraint that every subclass participates as a subclass in only one class/subclass relationship; that is, each subclass has only one parent, which results in a tree structure or strict hierarchy. In contrast, for a **specialization lattice**, a subclass can be a subclass in more than one class/subclass relationship. Hence, Figure 4.6 is a lattice.



A subclass with more than one superclass is called **a shared subclass** 🡪 **multiple inheritance**

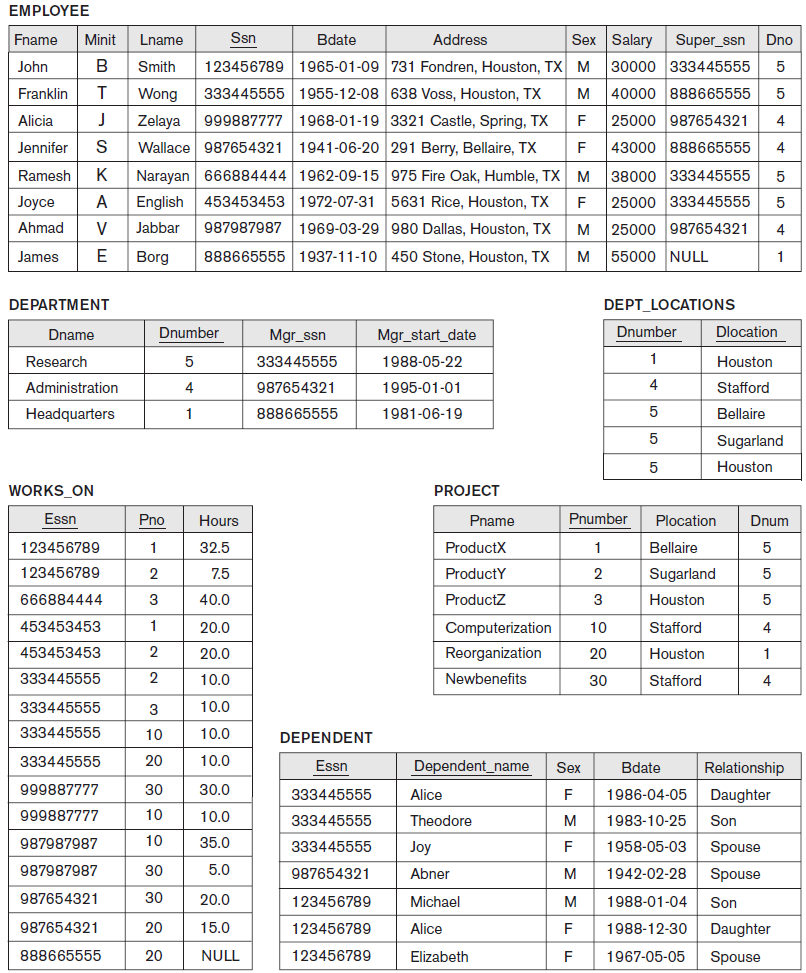
**Modeling of UNION Types Using Categories**

**union type** or a **category**.



A category can be **total** or **partial**. A total category holds the union of all entities in its superclasses, whereas a partial category can hold a subset of the union. A total category is represented diagrammatically by a double line connecting the category and the circle, whereas a partial category is indicated by a single line.

**The Relational Data Model and Relational Database Constraints**



In relational model terminology, a row is called a **tuple**, a column header is called an **attribute**, and the table is called a **relation**. The data type describing the types of values that can appear in each column is represented by a **domain** of possible values. The **degree** (or **arity**) of a relation is the number of attributes n of its relation schema.

Each value in a tuple is an **atomic** value; composite and multivalued attributes are not allowed. This model is sometimes called the **flat relational model** (**first normal form**).

**NULL values**: value unknown, value exists but is not available, or attribute does not apply to this tuple (also known as value undefined).

An assumption called the **closed world assumption** states that the only true facts in the universe are those present within the extension (state) of the relation(s). Any other combination of values makes the predicate false.

**Domain constraints** specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A).

A **superkey** SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK. Every relation has at least one default superkey— the set of all its attributes.

A **key** is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold.

A relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. It is common to designate one of the candidate keys as the **primary key** of the relation.

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.

**Primary Key Constraints:** A table typically has a column or combination of columns that contain values that uniquely identify each row in the table.

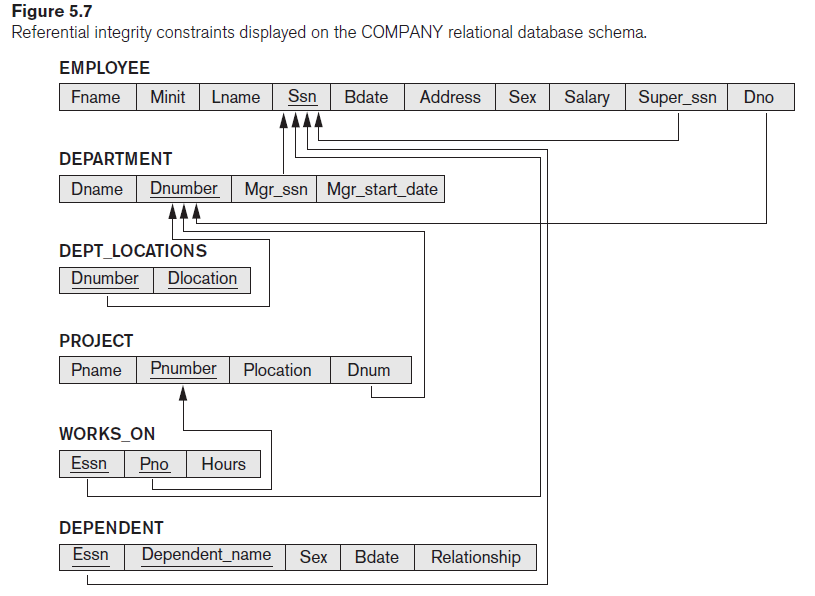
Another constraint on attributes specifies whether NULL values are or are not permitted.

The **entity integrity constraint** states that no primary key value can be NULL.

The **referential integrity constraint** states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. A set of attributes FK in relation schema R1 is a **foreign key** of R1 that references relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.

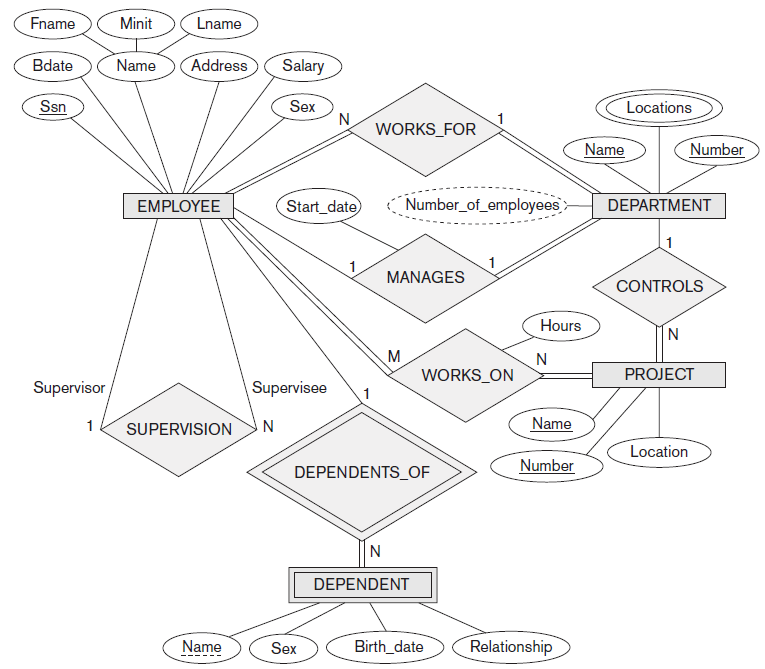
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is NULL.

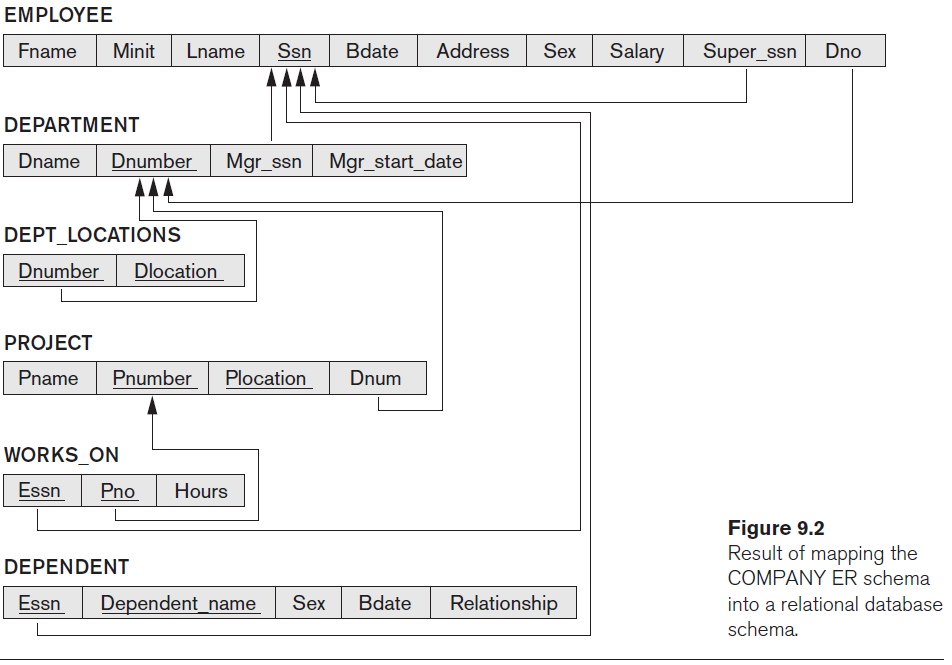


The Delete operation can violate only referential integrity. Several options: 1) **restrict**: reject the deletion. 2) **cascade**, is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted. 3) **set null** or **set default.**

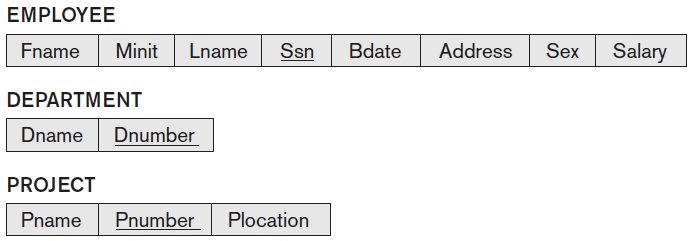
**Relational Database Design by ER- and EER-to-Relational Mapping**

**9.1.1 ER-to-Relational Mapping Algorithm**

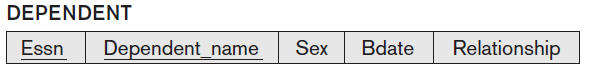




**Step 1: Mapping of Regular Entity Types**. For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R. If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify additional (unique) keys of relation R. Knowledge about keys is also kept for indexing purposes and other types of analyses.



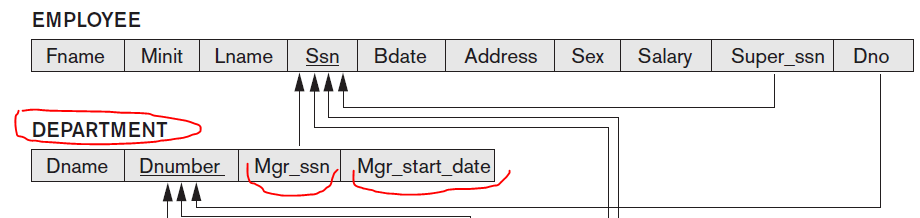
**Step 2: Mapping of Weak Entity Types**. For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first.



It is common to choose the propagate (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity.

**Step 3: Mapping of Binary 1:1 Relationship Types**. For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches. The first approach is the most useful and should be followed unless special conditions exist, as we discuss below.

1. Foreign key approach: Choose one of the relations—S, say—and include as a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.



Note that it is possible to include the primary key of S as a foreign key in T instead. In our example, this amounts to having a foreign key attribute, say Department\_managed in the EMPLOYEE relation, but it will have a NULL value for employee tuples who do not manage a department. This would be a bad choice, because if only 2% of employees manage a department, then 98% of the foreign keys would be NULL in this case. Another possibility is to have foreign keys in both relations S and T redundantly, but this creates redundancy and incurs a penalty for consistency maintenance.

2. Merged relation approach: An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when both participations are total, as this would indicate that the two tables will have the exact same number of tuples at all times.

3. Cross-reference or relationship relation approach: The third option is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. As we will see, this approach is required for binary M:N relationships. The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T. The relation R will include the primary key attributes of S and T as foreign keys to S and T. The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R. The drawback is having an extra relation, and requiring extra join operations when combining related tuples from the tables.

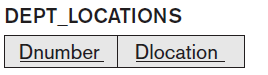
**Step 4: Mapping of Binary 1:N Relationship Types**. There are two possible approaches: (1) the foreign key approach and (2) the cross-reference or relationship relation approach. The first approach is generally preferred as it reduces the number of tables.

1. The foreign key approach: For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S.

2. The relationship relation approach: An alternative approach is to use the relationship relation (cross-reference) option as in the third option for binary 1:1 relationships. We create a separate relation R whose attributes are the primary keys of S and T, which will also be foreign keys to S and T. The primary key of R is the same as the primary key of S. This option can be used if few tuples in S participate in the relationship to avoid excessive NULL values in the foreign key.

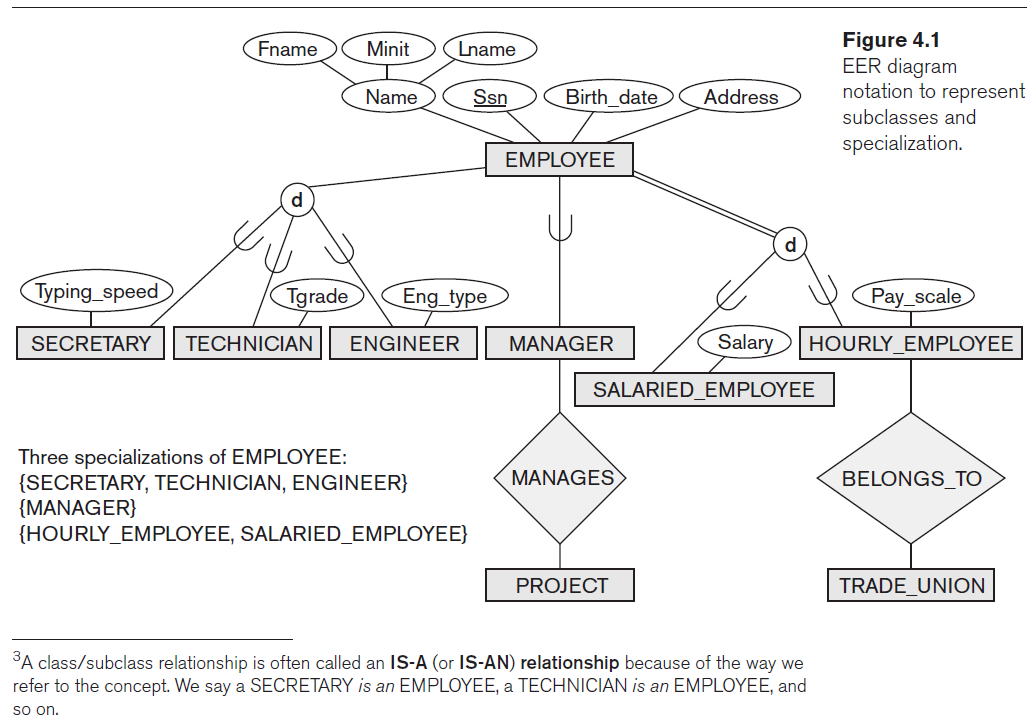
**Step 5: Mapping of Binary M:N Relationship Types**. In the traditional relational model with no multivalued attributes, the only option for M:N relationships is the relationship relation (cross-reference) option. For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S. we must create a separate relationship relation S.

**Step 6: Mapping of Multivalued Attributes**. For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.



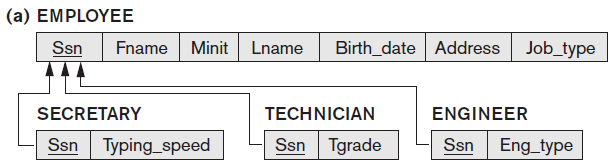
**Step 7: Mapping of N-ary Relationship Types**. We use the relationship relation option. For each n-ary relationship type R, where n > 2, create a new relationship relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types E participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation E′ corresponding to E (see the discussion in Section 3.9.2 concerning constraints on n-ary relationships). Consider the ternary relationship type SUPPLY in Figure 3.17, which relates a SUPPLIER s, PART p, and PROJECT j whenever s is currently supplying p to j; this can be mapped to the relation SUPPLY shown in Figure 9.4, whose primary key is the combination of the three foreign keys {Sname, Part\_no, Proj\_name}.

**9.2 Mapping EER Model Constructs to Relations**

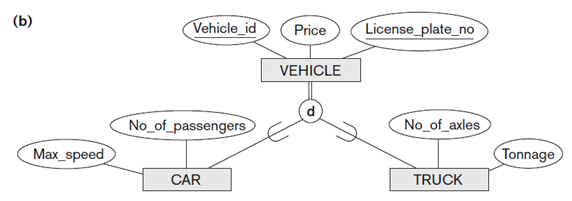


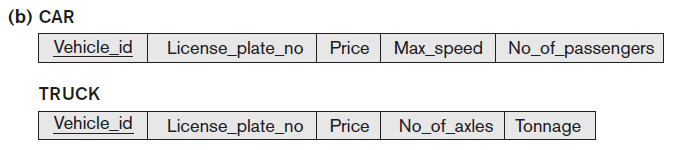
**Step 8: Options for Mapping Specialization or Generalization**

Option 8A: Multiple relations—superclass and subclasses. This option works for any specialization (total or partial, disjoint or overlapping).



Option 8B: Multiple relations—subclass relations only. This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses). Additionally, it is only recommended if the specialization has the disjointedness constraint (see Section 4.3.1). If the specialization is overlapping, the same entity may be duplicated in several relations.

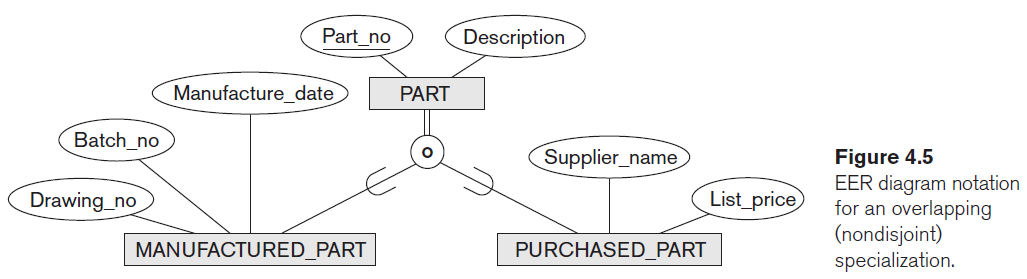


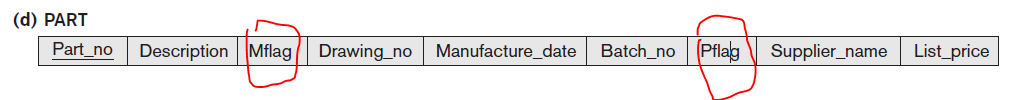


Option 8C: Single relation with one type attribute. This option works only for a specialization whose subclasses are disjoint, and has the potential for generating many NULL values if many specific (local) attributes exist in the subclasses.

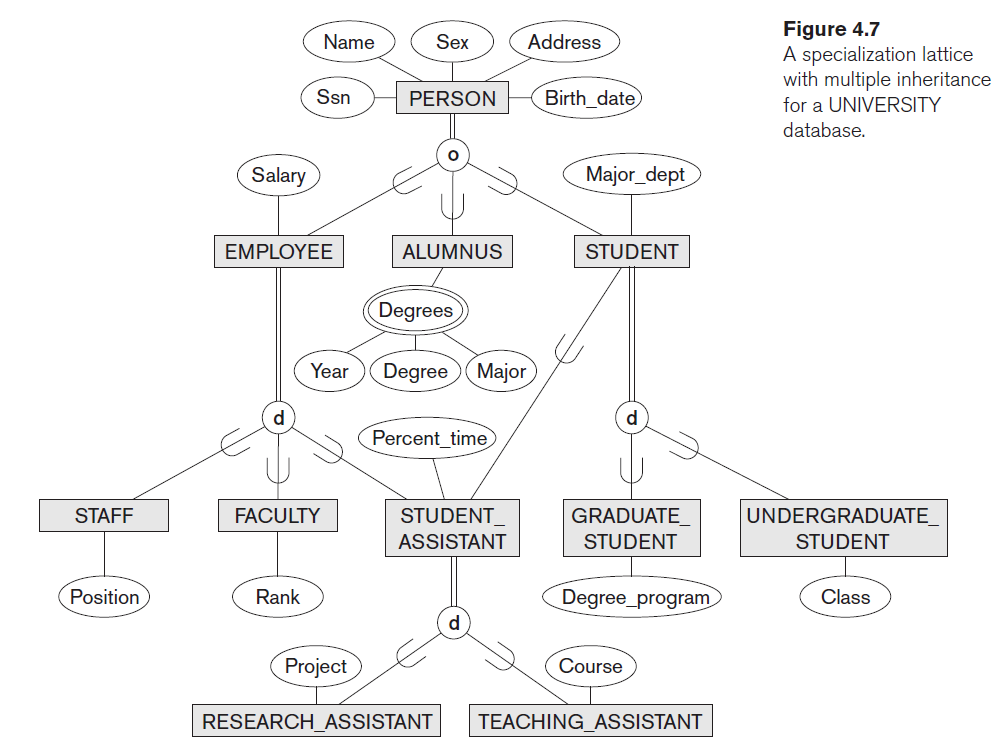


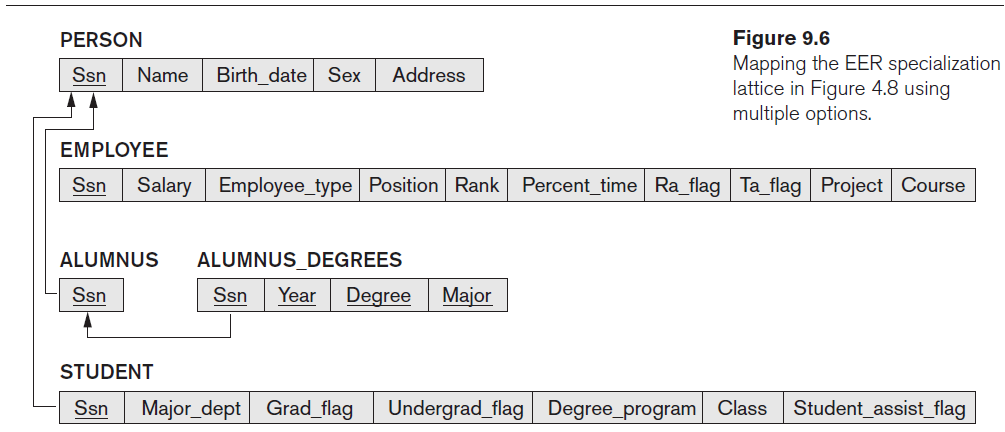
Option 8D: Single relation with multiple type attributes. Each ti, 1 ≤ i ≤ m, is a Boolean type attribute indicating whether or not a tuple belongs to subclass Si. This option is used for a specialization whose subclasses are overlapping (but will also work for a disjoint specialization).





For a multilevel specialization (or generalization) hierarchy or lattice, we do not have to follow the same mapping option for all the specializations. Instead, we can use one mapping option for part of the hierarchy or lattice and other options for other parts. Figure 9.6 shows one possible mapping into relations for the EER lattice in Figure 4.6. Here we used option 8A for PERSON/{EMPLOYEE, ALUMNUS, STUDENT}, and option 8C for EMPLOYEE/{STAFF, FACULTY, STUDENT\_ASSISTANT} by including the type attribute Employee\_type. We then used the single-table option 8D for STUDENT\_ASSISTANT/{RESEARCH\_ASSISTANT, TEACHING\_ASSISTANT} by including the type attributes Ta\_flag and Ra\_flag in EMPLOYEE. We also used option 8D for STUDENT/STUDENT\_ASSISTANT by including the type attributes Student\_assist\_flag in STUDENT, and for STUDENT/{GRADUATE\_STUDENT, UNDERGRADUATE\_STUDENT} by including the type attributes Grad\_flag and Undergrad\_flag in STUDENT. In Figure 9.6, all attributes whose names end with type or flag are type fields.

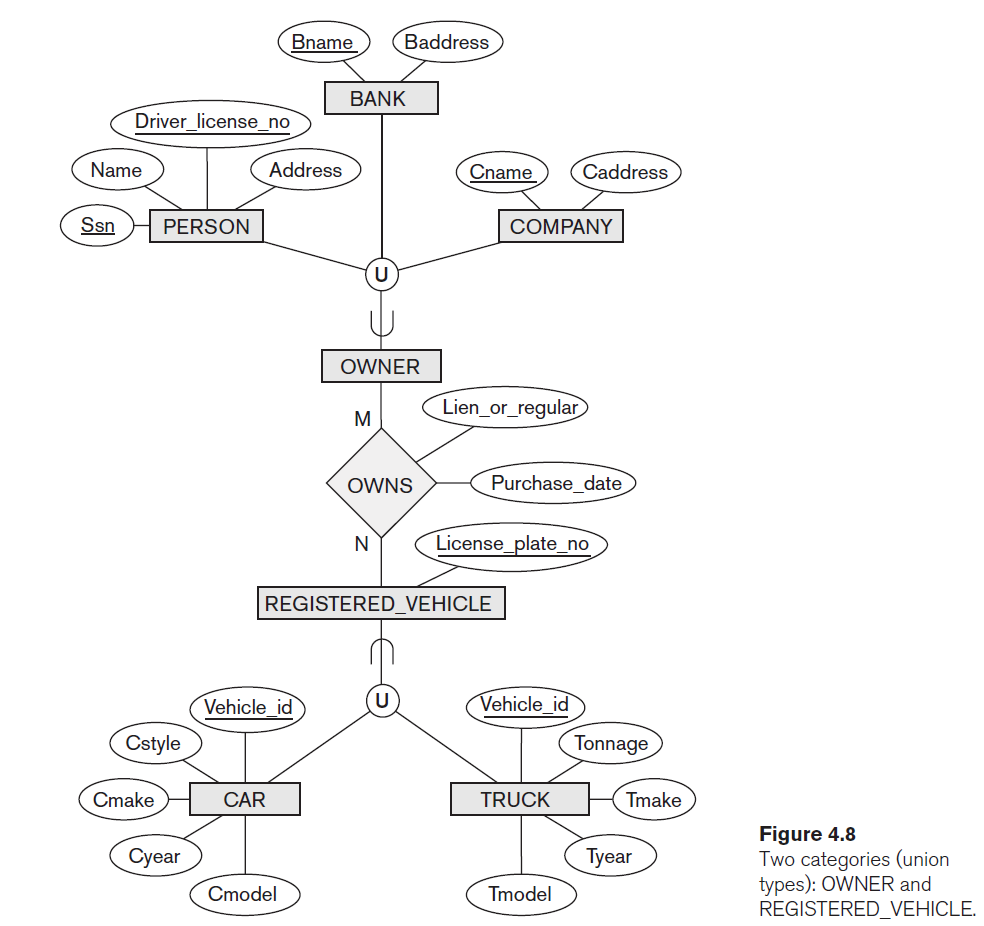


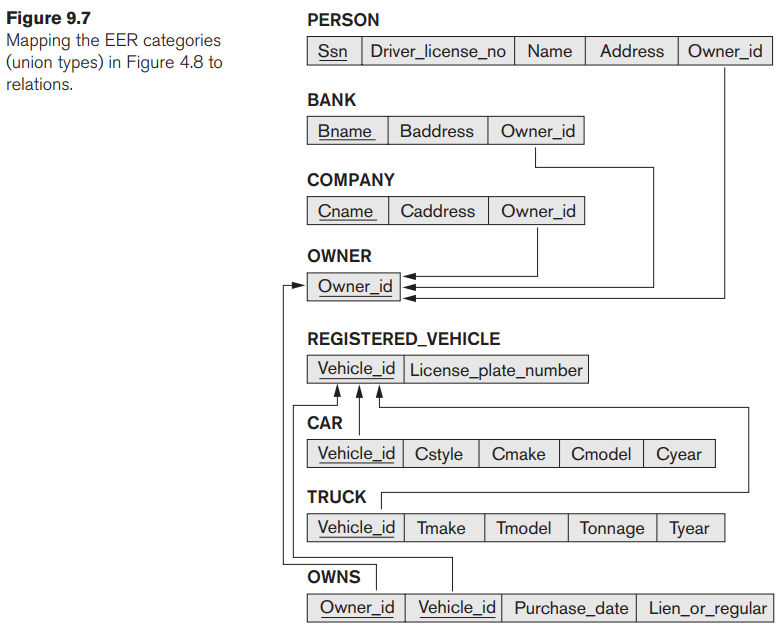


**9.2.2 Mapping of Shared Subclasses (Multiple Inheritance)**

A shared subclass, such as ENGINEERING\_MANAGER in Figure 4.6, is a subclass of several superclasses, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type) as we discussed in Section 4.4. We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm. In Figure 9.6, options 8C and 8D are used for the shared subclass STUDENT\_ASSISTANT. Option 8C is used in the EMPLOYEE relation (Employee\_type attribute) and option 8D is used in the STUDENT relation (Student\_assist\_flag attribute).

**Step 9: Mapping of Union Types (Categories)**. it is customary to specify a new key attribute, called a **surrogate key**





It is also recommended to add a type attribute (not shown in Figure 9.7) to the OWNER relation to indicate the particular entity type to which each tuple belongs (PERSON or BANK or COMPANY). For a category whose superclasses have the same key, such as VEHICLE in Figure 4.8, there is no need for a surrogate key. The mapping of the REGISTERED\_VEHICLE category, which illustrates this case, is also shown in Figure 9.7

**Basics of Functional**

**Dependencies and Normalization**

**for Relational Databases**

So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or enhanced-ER (EER) data model. However, we still need some formal way of analyzing why one grouping of attributes into a relation schema may be better than another. While discussing database design in Chapters 3, 4, and 9, we did not develop any measure of appropriateness or goodness to measure the quality of the design, other than the intuition of the designer. Relational database design ultimately produces a set of relations. The implicit goals of the design activity are information preservation and minimum redundancy.

**14.1 Informal Design Guidelines**

**for Relation Schemas**

Before discussing the formal theory of relational database design, we discuss four informal guidelines that may be used as measures to determine the quality of relation schema design:

■ Making sure that the semantics of the attributes is clear in the schema

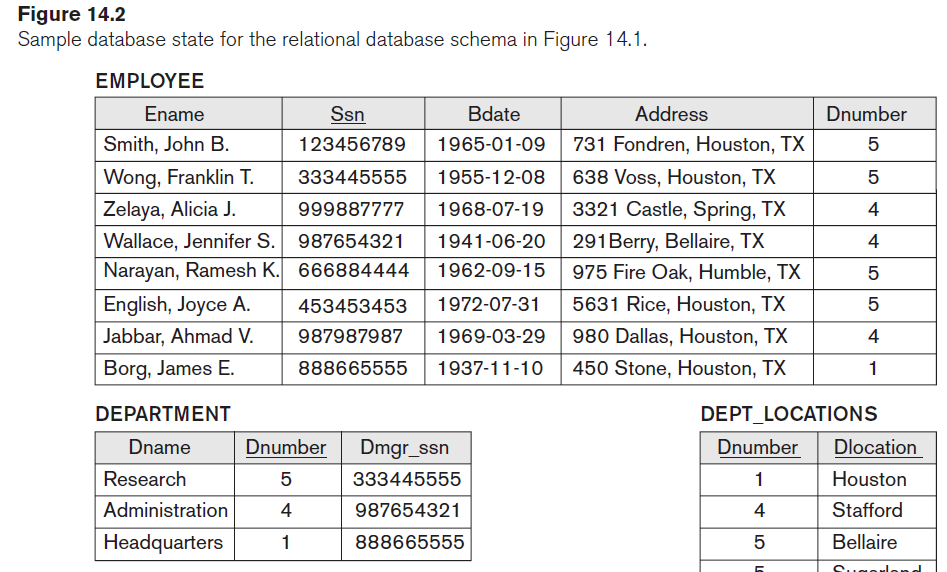
■ Reducing the redundant information in tuples (14.1.2)

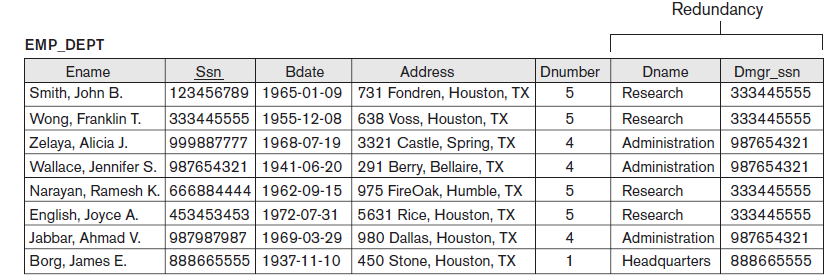
■ Reducing the NULL values in tuples

■ Disallowing the possibility of generating spurious tuples

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

One goal of schema design is to minimize the storage space used by the base relations. Grouping attributes into relation schemas has a significant effect on storage space. For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in Figure 14.2 with that for an EMP\_DEPT base relation in Figure 14.4, which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT.





In EMP\_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr\_ssn) are

repeated for every employee who works for that department. In contrast, each department’s information appears only once in the DEPARTMENT relation in Figure 14.2. Only the department number (Dnumber) is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key.

Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

**Insertion Anomalies**. Insertion anomalies can be differentiated into two types:

■ To insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP\_DEPT. In the design of Figure 14.2, we do not have to worry about this

consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation.

■ It is difficult to insert a new department that has no employees as yet in the EMP\_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP\_DEPT because its primary key Ssn cannot be null.

**Deletion Anomalies**. The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. If we delete from EMP\_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost inadvertently from the database. This problem does not occur in the database of Figure 14.2 because DEPARTMENT tuples are stored separately.

**Modification Anomalies**. In EMP\_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

Sul testo ci sono esempi interessanti che non sto riportando qui.

**4.2 Functional Dependencies**

A **functional dependency**, denoted by X 🡪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 t1 and t2 in r that have t1[X] 🡪t2[X], they must also have t1[Y] 🡪t2[Y].

A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R. Therefore, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R.

**Normalization of Relations**

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 discussed in Section 14.1.4 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, as we discuss in Section 15.2.2. We defer the discussion of the formal concepts and techniques that guarantee the above two properties to Chapter 15.

**14.3.2 Practical Use of Normal Forms**

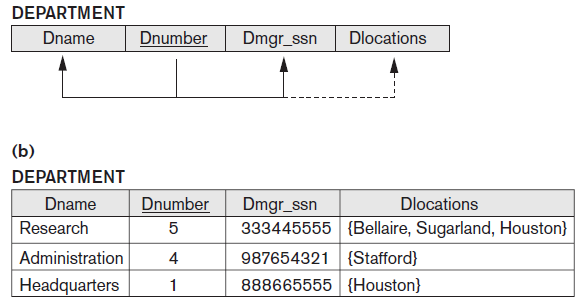
Although several higher normal forms have been defined, such as the 4NF and 5NF that we discuss in Sections 14.6 and 14.7, the practical utility of these normal forms becomes questionable. The reason is that the constraints on which they are based are rare and hard for the database designers and users to understand or to detect. Designers and users must either already know them or discover them as a part of the business. 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, such as those discussed at the end of Section 14.1.2. Doing so incurs the corresponding penalties of dealing with the anomalies. **Denormalization** is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

**Normal Forms Based on Primary Keys**

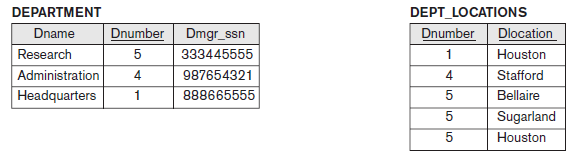
**First Normal Form**

The only attribute values permitted by 1NF are single atomic (or indivisible) values.

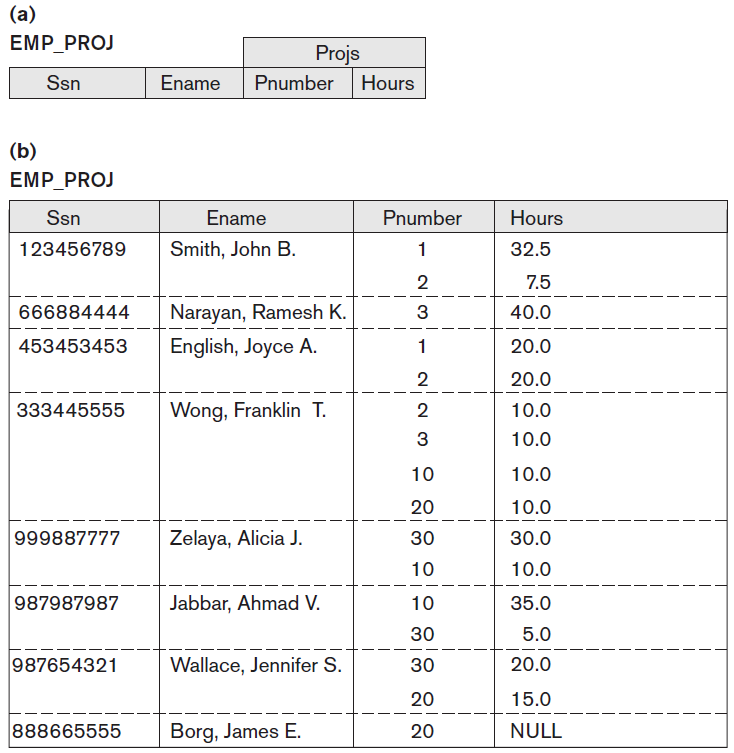
Supponiamo di violare la 1NF aggiungendo un attributo multivalore alla relazione DEPARTMENT:



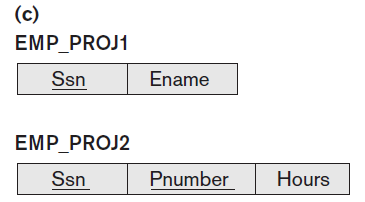
To achieve first normal form for such a relation:



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. Figure 14.10 shows how the EMP\_PROJ relation could appear if nesting is allowed.



Notice that Ssn is the primary key of the EMP\_PROJ relation in Figures 14.10(a) and (b), whereas Pnumber is the partial key of the nested relation; To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it;



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#})

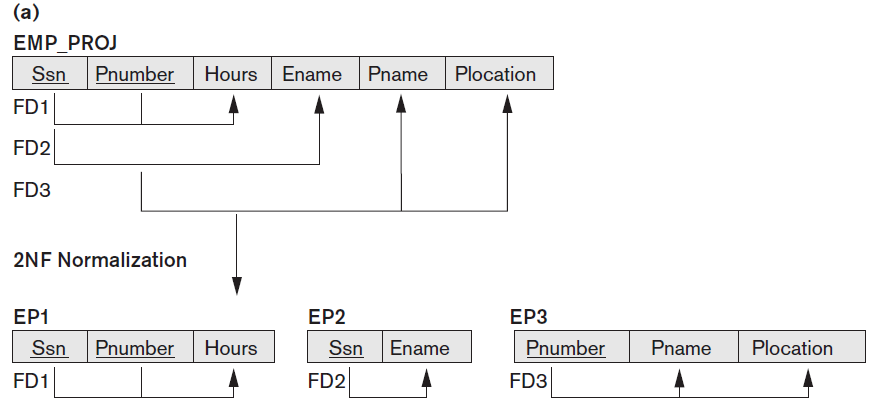
The right way to deal with the two multivalued attributes in PERSON shown previously is to decompose it into two separate relations, using strategy 1 discussed above:

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 -> 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 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 14.3(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.

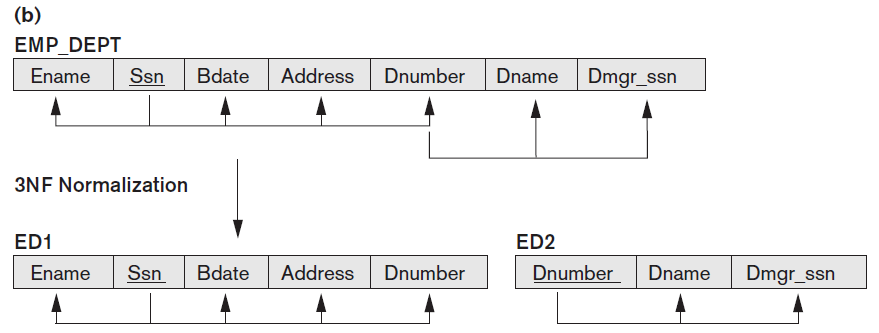


**Third Normal Form**

Teorema: Ogni relazione può essere portata in 3NF

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. The dependency Ssn -> Dmgr\_ssn is transitive through Dnumber in EMP\_DEPT in Figure 14.3(a), 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. Intuitively, we can see that the dependency of Dmgr\_ssn on Dnumber is undesirable in EMP\_DEPT since Dnumber is not a key of EMP\_DEPT.

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.

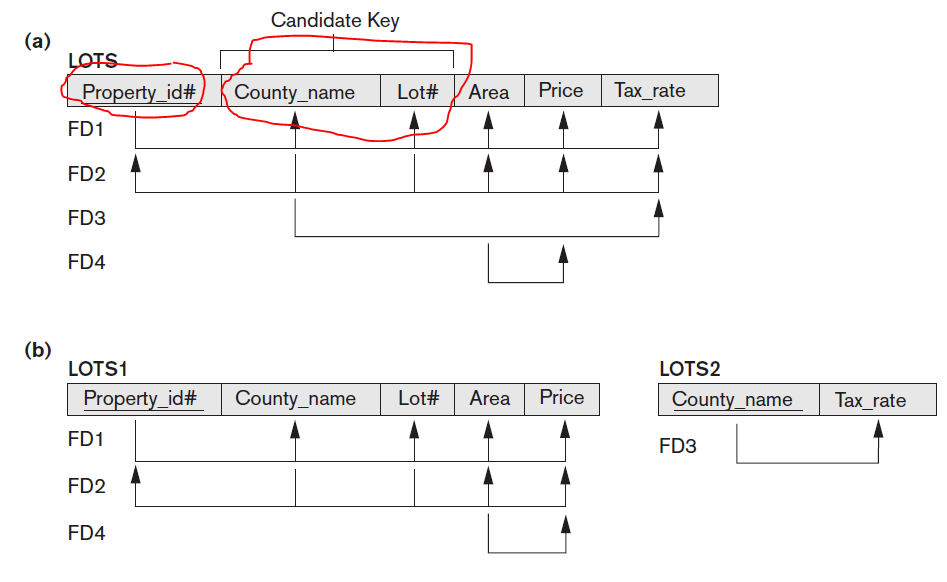


**General Definitions of Second and Third Normal Forms**

**Prime attribute**, an attribute that is part of any candidate key will be considered as prime. Partial and full functional dependencies and transitive dependencies will now be considered with respect to all candidate keys of a relation.

**General Definition of Second Normal Form**

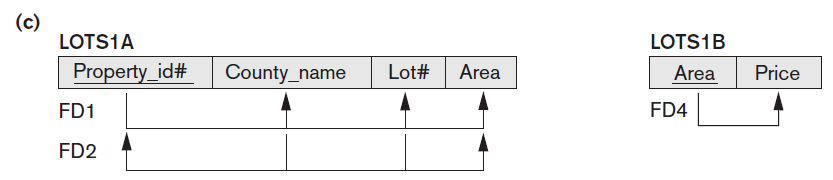
Definition. A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.



**14.4.2 General Definition of Third Normal Form**

A ->B is **trivial functional dependency** if B is a subset of A.

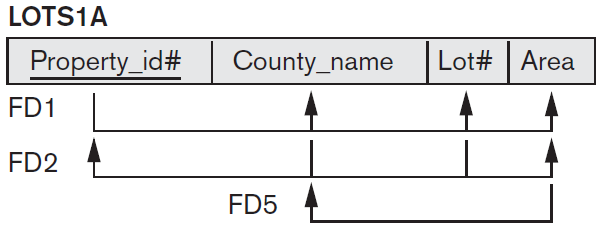
**Definition**. A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X 🡪A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.



**FORMA NORMALE DI BOYCE E CODD**

Ogni relazione in BCNF è anche in 3NF, mentre una relazione in 3NF non è necessariamente in BCNF.

Dato questo schema di relazione:



FD1: la parte sinistra della DF è superchiave (in particolare è la chiave primaria)

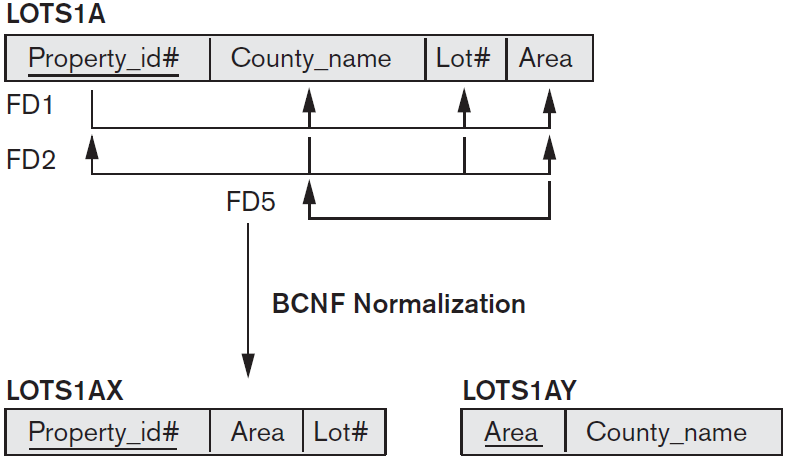
FD2: la parte sinistra (County\_name, Lot#) è superchiave (è una chiave candidata)

FD5: la parte destra è un attributo primo.

Quindi questo schema di relazione è in 3NF.

**Definizione**. Uno di schema di relazione R è in **BCNF** se, ogni volta che sussiste in R una dipendenza funzionale non-banale X 🡪 A, X è una superchiave di R.

Nel nostro esempio Area non è una superchiave di R, quindi quello schema di relazione non è in BCNF.



Si noti che in questa decomposizione perdiamo la dipendenza funzionale FD2 perchè dopo la decomposizione i suoi attributi non coesistono più nella stessa relazione.