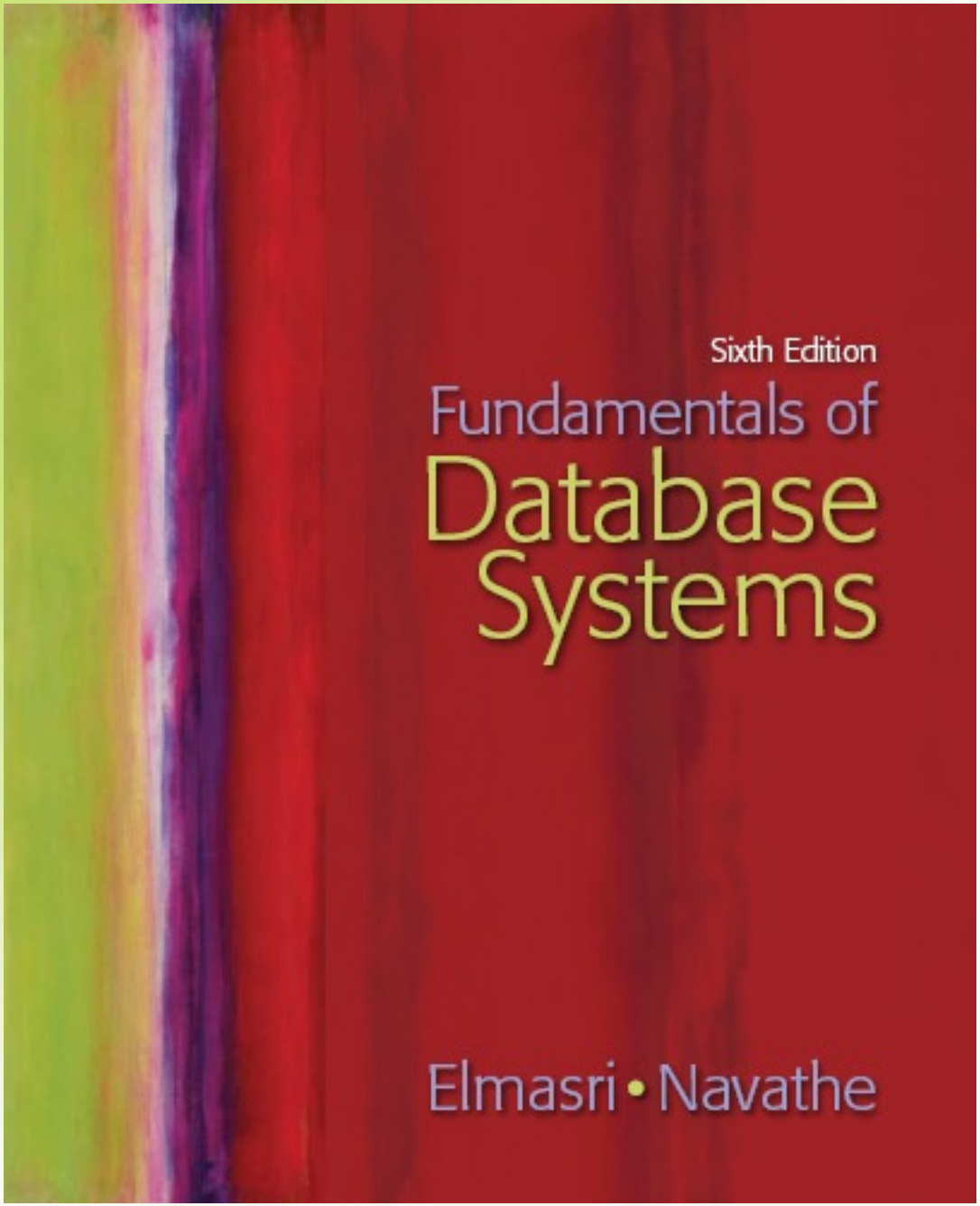
Chapter 9

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



Addison-Wesley
is an imprint of
PEARSON

Chapter 9 Outline

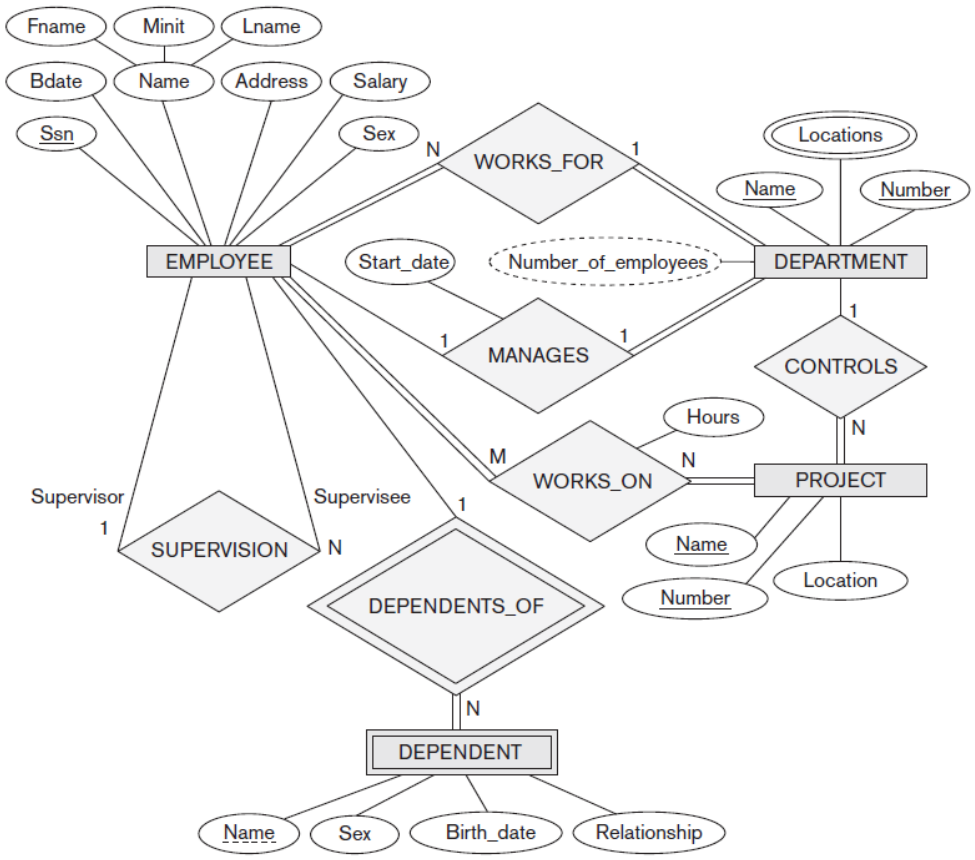
- Relational Database Design Using ER-to-Relational Mapping
- Mapping EER Model Constructs to Relations

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

- **Design a relational database schema**
  - Based on a conceptual schema design
- Seven-step algorithm to convert the basic ER model constructs into relations
- Additional steps for EER model

Relational Database Design Using ER-to-Relational Mapping

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



ER-to-Relational Mapping Algorithm

- COMPANY database example
  - Assume that the mapping will create tables with simple single-valued attributes
- Step 1: Mapping of Regular Entity Types
  - For each regular entity type, create a relation *R* that includes all the simple attributes of *E*
  - Called **entity relations**
    - Each tuple represents an entity instance
  - 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

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 2: Mapping of Weak Entity Types
  - For each weak entity type, create a relation *R* and include all simple attributes of the entity type as attributes of *R*
  - Include primary key attribute of owner as foreign key attributes of *R*
  - 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, if any

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 3: Mapping of Binary 1:1 Relationship Types
  - For each binary 1:1 relationship type
    - Identify relations that correspond to entity types participating in *R* (for example, S and T)
  - Possible approaches:
    - **Foreign key approach**
    - **Merged relationship approach**
    - **Crossreference or relationship relation approach**

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*.

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

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.

Cross-reference 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.
- The relation *R* is called a **relationship relation** (or sometimes a **lookup table**)
- 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*
- Only recommended when few relationship instances exist, in order to avoid NULL values in foreign keys

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 4: Mapping of Binary 1:*N* Relationship Types
  - For each regular binary 1:*N* relationship type
    - Identify relation S that represents participating entity type at *N*-side of relationship type
    - Include primary key of other entity type as foreign key in *S*
    - Include simple attributes of 1:*N* relationship type as attributes of *S*
    - Foreign key approach

ER-to-Relational Mapping Algorithm (cont'd.)

- Alternative approach
  - Use the **relationship relation** (cross-reference) option as in the third option for binary 1:1 relationships

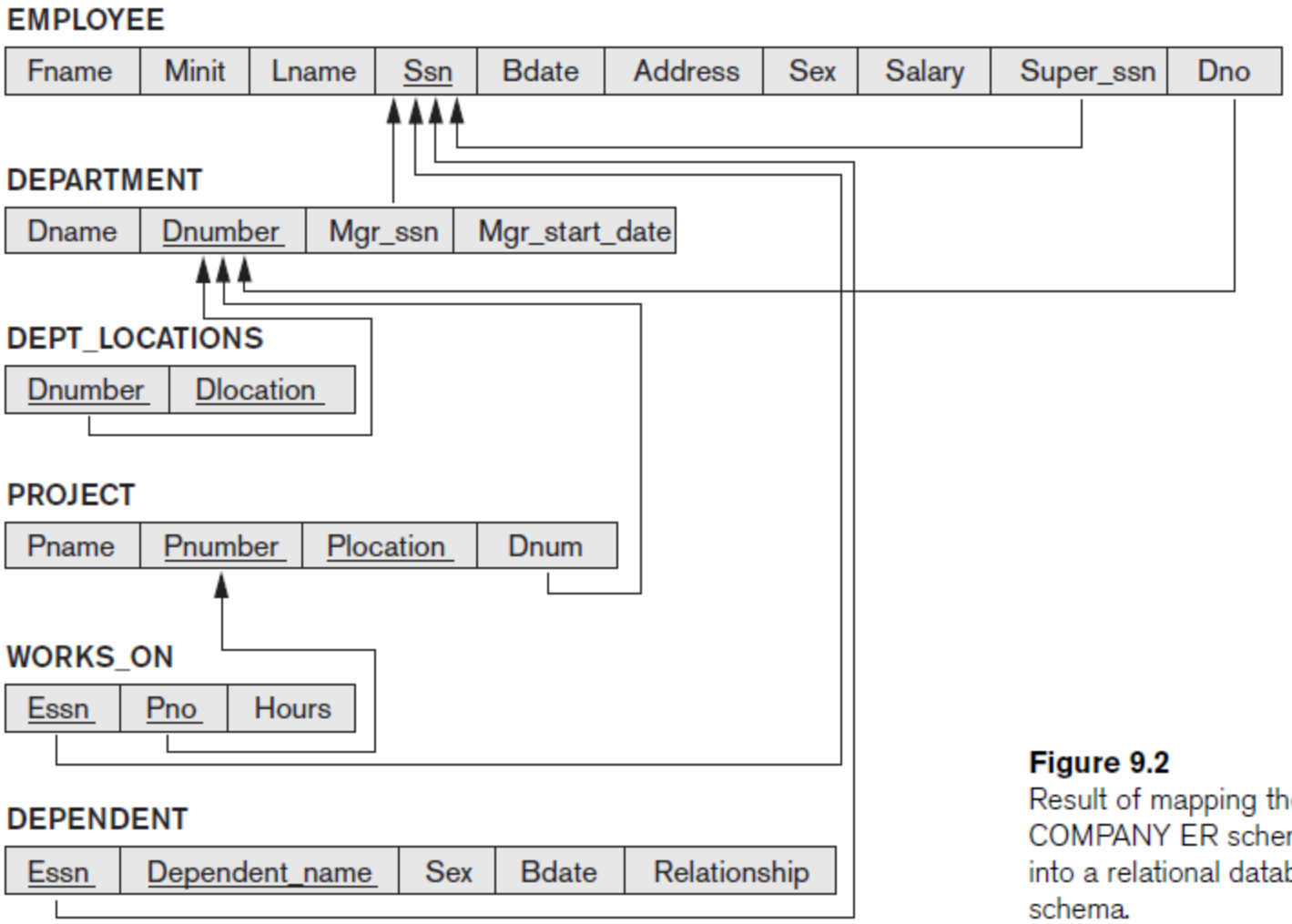


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 5: Mapping of Binary *M*:*N* Relationship Types
  - For each binary *M*:*N* relationship type
    - Create a new relation *S*
    - Include primary key of participating entity types as foreign key attributes in *S*
    - Primary key will be the combination of both foreign keys
    - Include any simple attributes of *M*:*N* relationship type
    - Relationship relation approach
    - No other approach
    - The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 6: Mapping of Multivalued Attributes
  - For each multivalued attribute A in a relationship with primary key K
    - Create a new relation
    - Primary key of *R* is the combination of *A* and *K*
    - If the multivalued attribute is composite, include its simple components
    - This step is not necessary target relation model includes array data types
    - The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key

ER-to-Relational Mapping Algorithm (cont'd.)

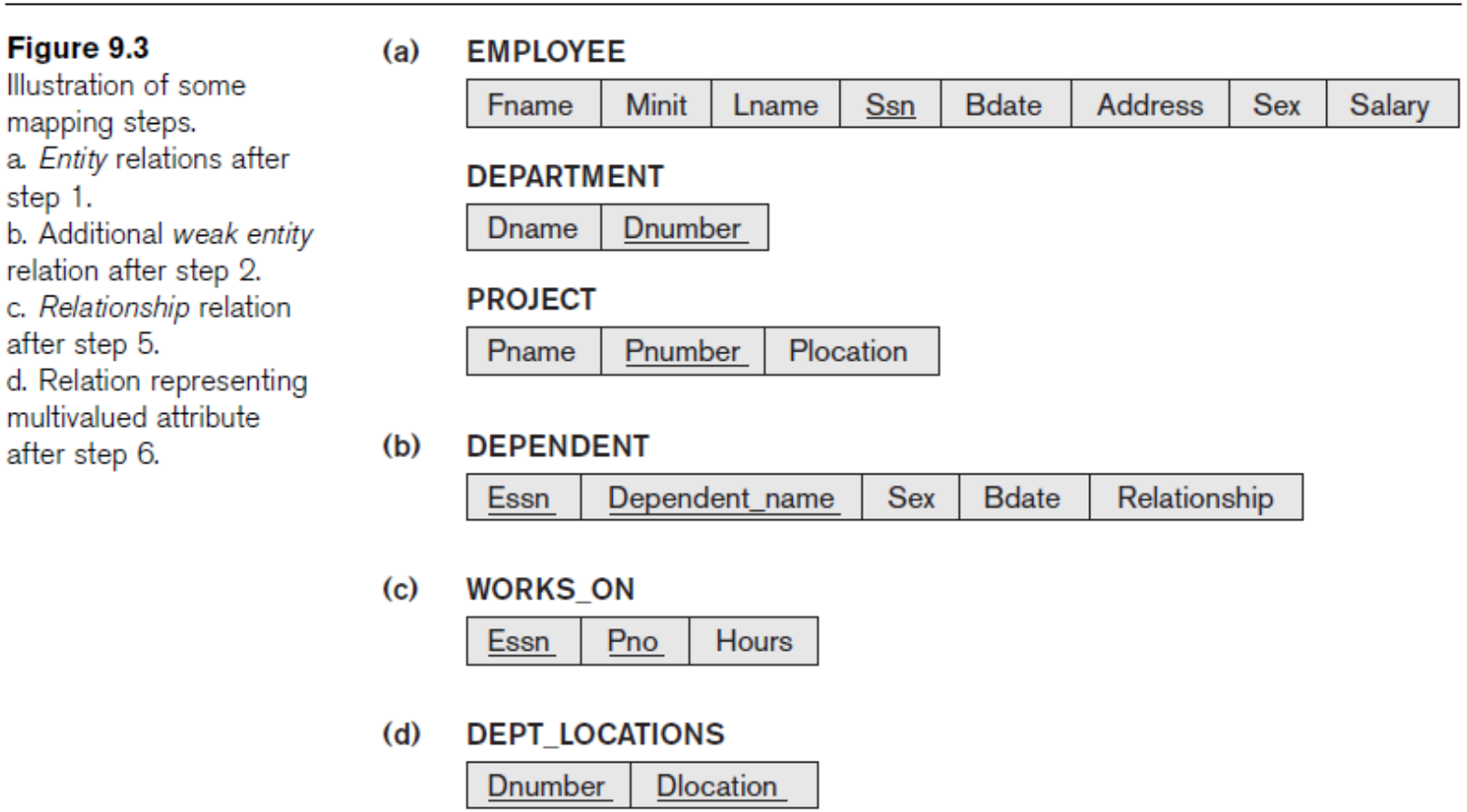
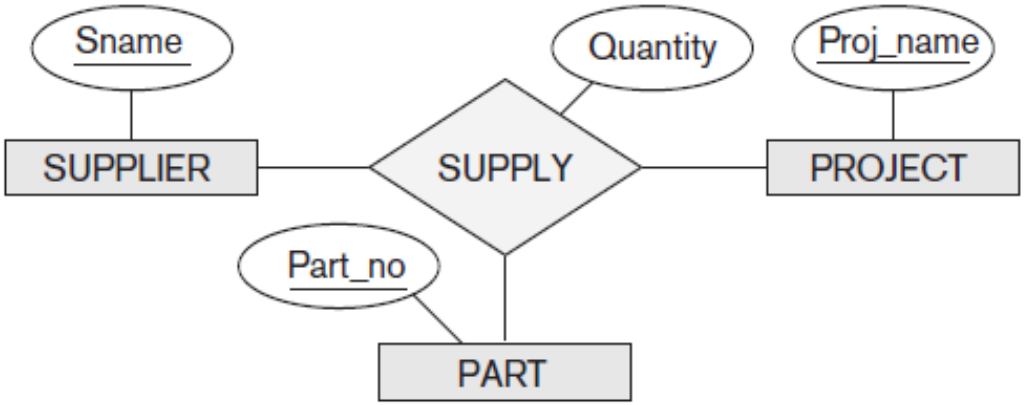


Figure 9.3
Illustration of some mapping steps.
a. *Entity* relations after step 1.
b. Additional *weak entity* relation after step 2.
c. *Relationship* relation after step 5.
d. Relation representing multivalued attribute after step 6.

ER-to-Relational Mapping Algorithm (cont'd.)

- Step 7: Mapping of *N*-ary Relationship Types
  - For each *n*-ary relationship type *R*
    - Create a new relation *S* to represent *R*
    - Include primary keys of participating entity types as foreign keys
    - Include any simple attributes as attributes
    - The primary key of *S* is usually a combination of all the foreign keys



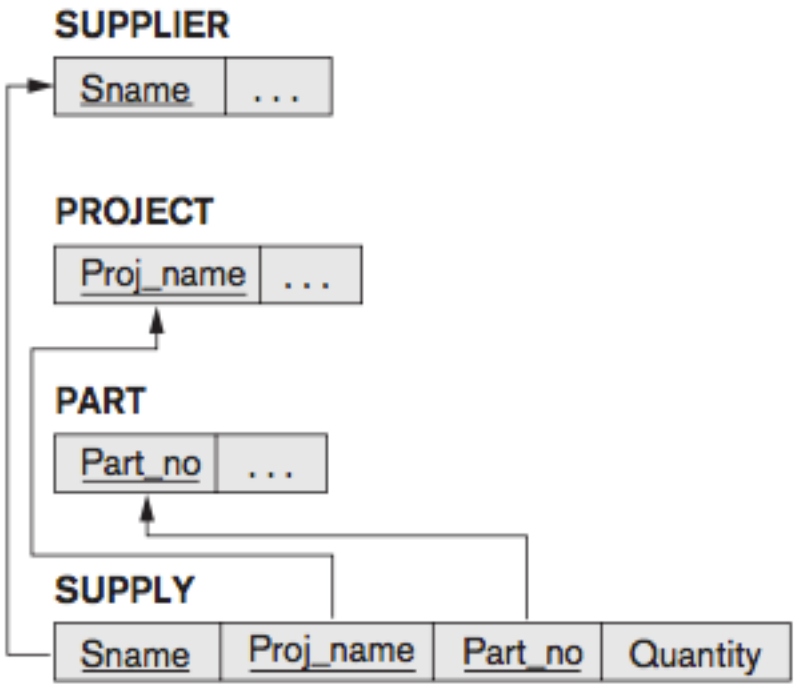


Figure 9.4
Mapping the *n*-ary relationship type SUPPLY from Figure 3.17(a).

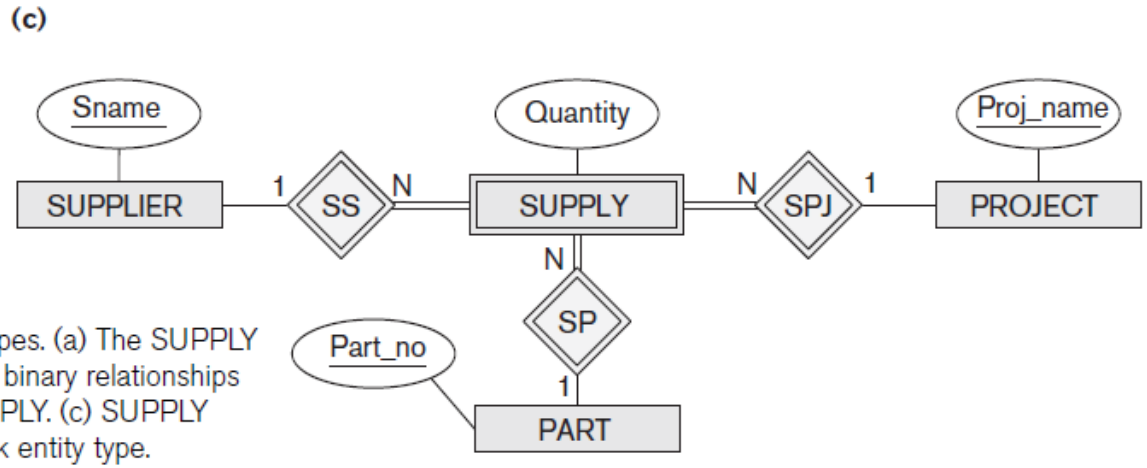
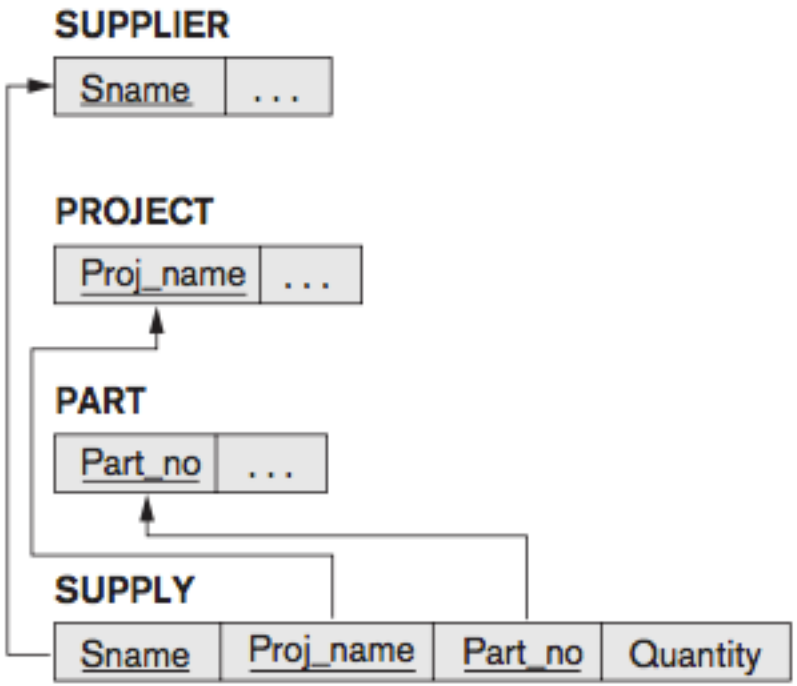


Figure 7.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Figure 9.4
Mapping the *n*-ary relationship type SUPPLY from Figure 3.17(a).



Discussion and Summary of Mapping for ER Model Constructs

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Discussion and Summary of Mapping for ER Model Constructs (cont'd.)

- In a relational schema relationship, types are not represented explicitly
  - Represented by having two attributes *A* and *B*: one a primary key and the other a foreign key

Mapping EER Model Constructs to Relations

- Extending ER-to-relational mapping algorithm

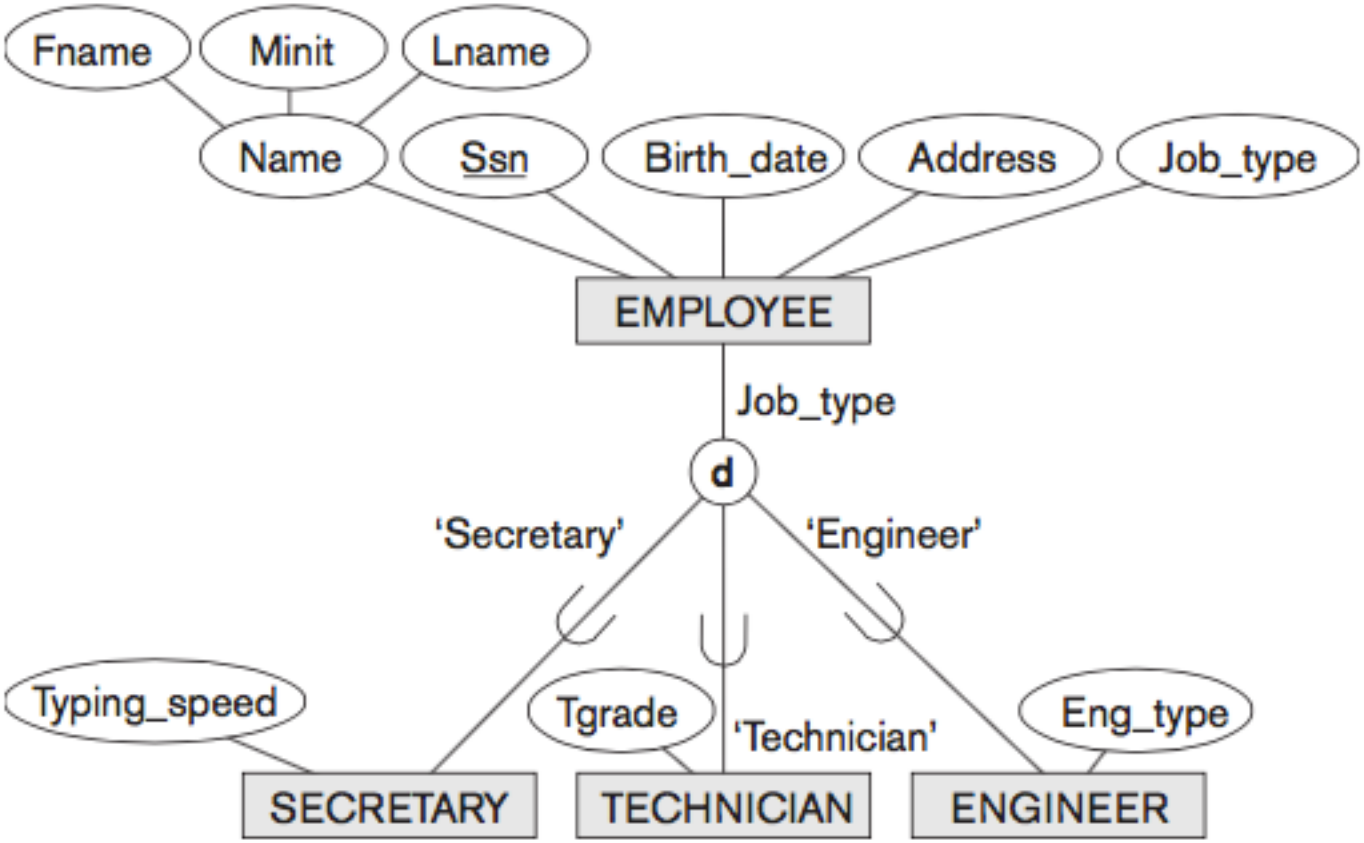
- Attrs(*R*) denote *the attributes of a relation R*, and PK(*R*) denote the *primary key of R*.

Mapping of Specialization or Generalization

- Step 8: Options for Mapping Specialization or Generalization
  - Convert each specialization with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and (generalized) superclass C, where the attributes of C are $\{k, a_1, \ldots, a_n\}$ and k is the (primary) key, into relation schemas using one of the following options:

Mapping of Specialization or Generalization

- **Option 8A: Multiple relations—superclass and subclasses**
  - For any specialization (total or partial, disjoint or overlapping)
  - Create a relation L for C with attributes Attrs(L) = {k, a_1, ... , a_n} and PK(L) = k. Create a relation L_i for each subclass S_i, $1 \leq i \leq m$, with the attributes Attrs(L_i) = {k} $\cup$ {attributes of S_i} and PK(L_i) = k.



(a) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type

SECRETARY

Ssn	Typing_speed

TECHNICIAN

Ssn	Tgrade

ENGINEER

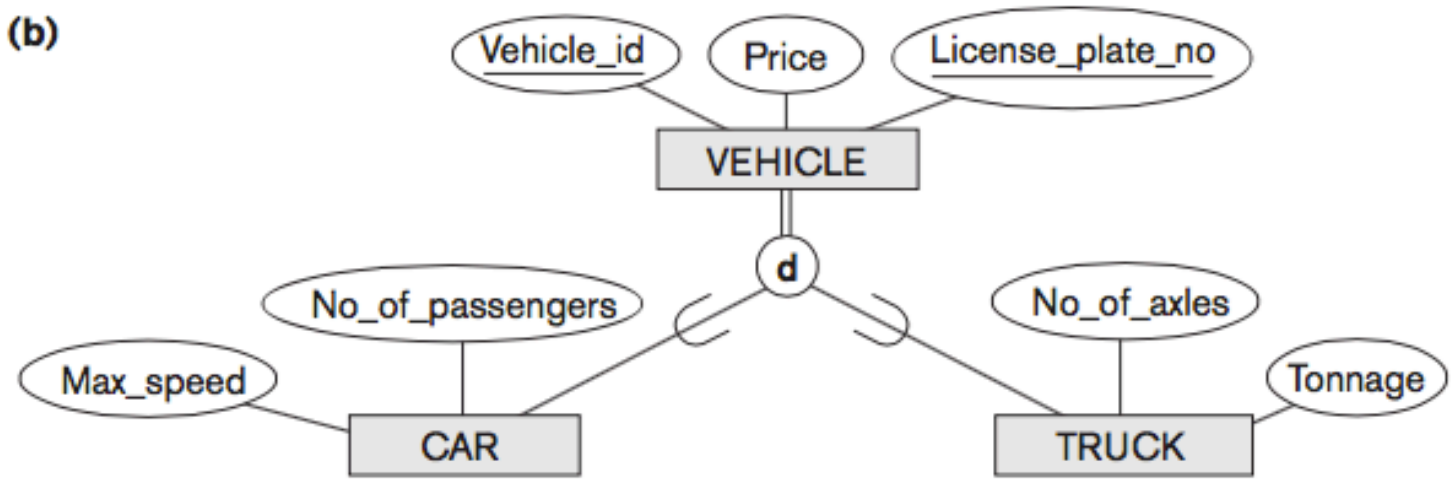
Ssn	Eng_type

Mapping of Specialization or Generalization

- Option 8A creates a relation L for the superclass C and its attributes, plus a relation L_i for each subclass S_i; each L_i includes the specific (local) attributes of S_i, plus the primary key of the superclass C, which is propagated to L_i and becomes its primary key. It also becomes a foreign key to the superclass relation
- An EQUIJOIN operation on the primary key between any L_i and L produces all the specific and inherited attributes of the entities in S_i
- Option 8A works for any constraints on the specialization: disjoint or overlapping, total or partial

Mapping of Specialization or Generalization

- **Option 8B: Multiple relations—subclass relations only**
  - Subclasses are total
  - Specialization has disjointedness constraint
  - Create a relation L_i for each subclass S_i, $1 \leq i \leq m$, with the attributes Attrs(L_i) = {attributes of S_i} $\cup$ {k, a_1, ... , a_n} and PK(L_i) = k. 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,* if the specialization is *overlapping*, the same entity may be duplicated in several relations.



(b) CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage

Mapping of Specialization or Generalization

- Option 8B works well only when *both* the disjoint and total constraints hold. If the specialization is not total, an entity that does not belong to any of the subclasses S_i is lost
- If the specialization is not disjoint, an entity belonging to more than one subclass will have its inherited attributes from the superclass *C* stored redundantly in more than one table L_i
- With option 8B, no relation holds all the entities in the superclass *C*; consequently, we must apply an OUTER UNION operation to the L_i relations to retrieve all the entities in *C*
- The result of the outer union will be similar to the relations under options 8C and 8D except that the type fields will be missing. Whenever we search for an arbitrary entity in *C*, we must search all the *m* relations L_i

Mapping of Specialization or Generalization (cont'd.)

- **Option 8C: Single relation with one type attribute**
  - Type or discriminating attribute indicates subclass of tuple
  - Subclasses are disjoint
    - Potential for generating many NULL values if many specific attributes exist in the subclasses
  - Create a single relation L with attributes Attrs(L) = {k, a_1, ..., a_n} ∪ {attributes of S_1} ∪ ... ∪ {attributes of S_m} ∪ {t} and PK(L) = k. The attribute t is called a **type** (or **discriminating**) attribute whose value indicates the subclass to which each tuple belongs, if any
  - 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

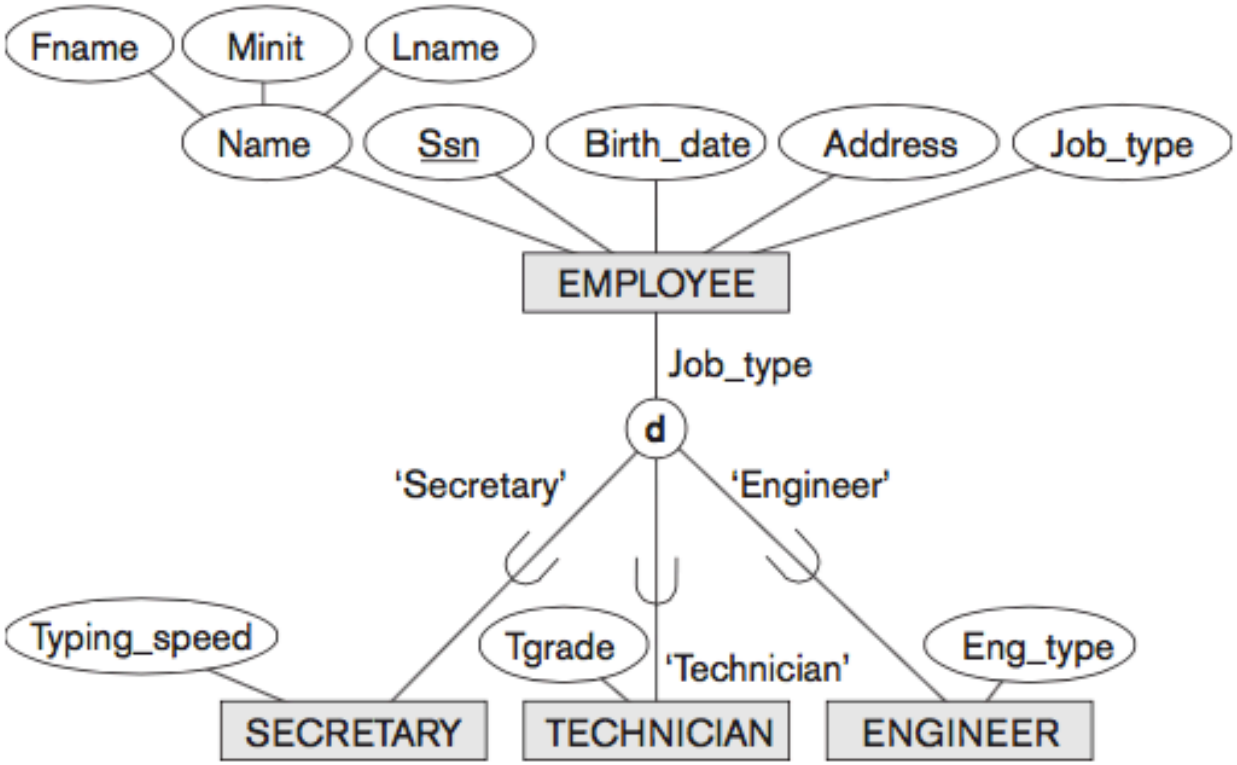


Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.

(c) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type

Mapping of Specialization or Generalization

- Option 8C is used to handle disjoint subclasses by including a single **type** (or **image** or **discriminating**) **attribute** *t* to indicate to which of the *m* subclasses each tuple belongs
  - The domain of *t* could be {1, 2, ... , *m*}. If the specialization is partial, *t* can have NULL values in tuples that do not belong to any subclass
  - If the specialization is attribute-defined, that attribute itself serves the purpose of *t* and *t* is not needed

Mapping of Specialization or Generalization (cont'd.)

- **Option 8D: Single relation with multiple type attributes**
  - Subclasses are overlapping
  - Will also work for a disjoint specialization
  - Create a single relation schema L with attributes Attrs(L) = $\{k, a_1, ..., a_n\} \cup \{\text{attributes of } S_1\} \cup ... \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, ..., t_m\}$ and PK(L) = k. Each t_i, $1 \leq i \leq m$, is a **Boolean type attribute** indicating whether or not a tuple belongs to subclass S_i
  - This option is used for a specialization whose subclasses are *overlapping* (but will also work for a disjoint specialization)

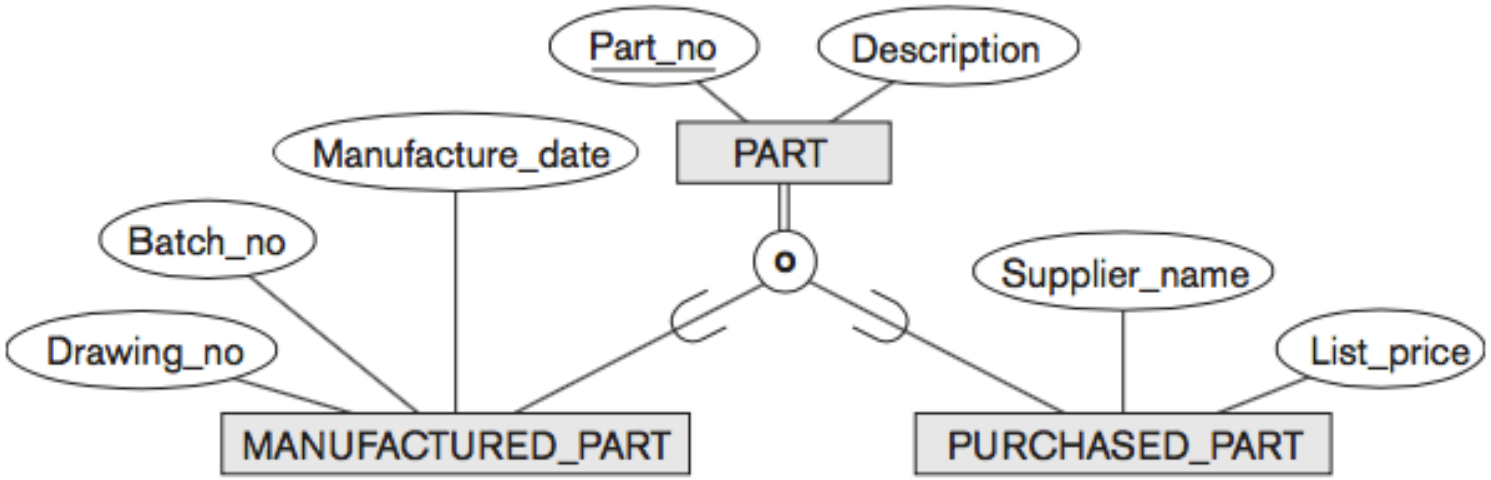


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

(d) PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price

Mapping of Specialization or Generalization

- Option 8D is designed to handle overlapping subclasses by including *m* *Boolean* **type** (or **flag**) fields, one for *each* subclass
  - It can also be used for disjoint subclasses
- Each type field t_i can have a domain {yes, no}, where a value of yes indicates that the tuple is a member of subclass S_i

Mapping of Specialization or Generalization

- Options 8A and 8B are the **multiple-relation options**, whereas options 8C and 8D are the **single-relation options**
- Options 8C and 8D create a single relation to represent the superclass *C* and all its subclasses
  - An entity that does not belong to some of the subclasses will have NULL values for the specific (local) attributes of these subclasses
  - These options are not recommended if many specific attributes are defined for the subclasses
  - If few local subclass attributes exist, however, these mappings are preferable to options 8A and 8B because they do away with the need to specify JOIN operations; therefore, they can yield a more efficient implementation for queries.

Mapping of Shared Subclasses (Multiple Inheritance)

- A shared subclass 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)
- Apply any of the options discussed in step 8 to a shared subclass (subject to the restrictions)

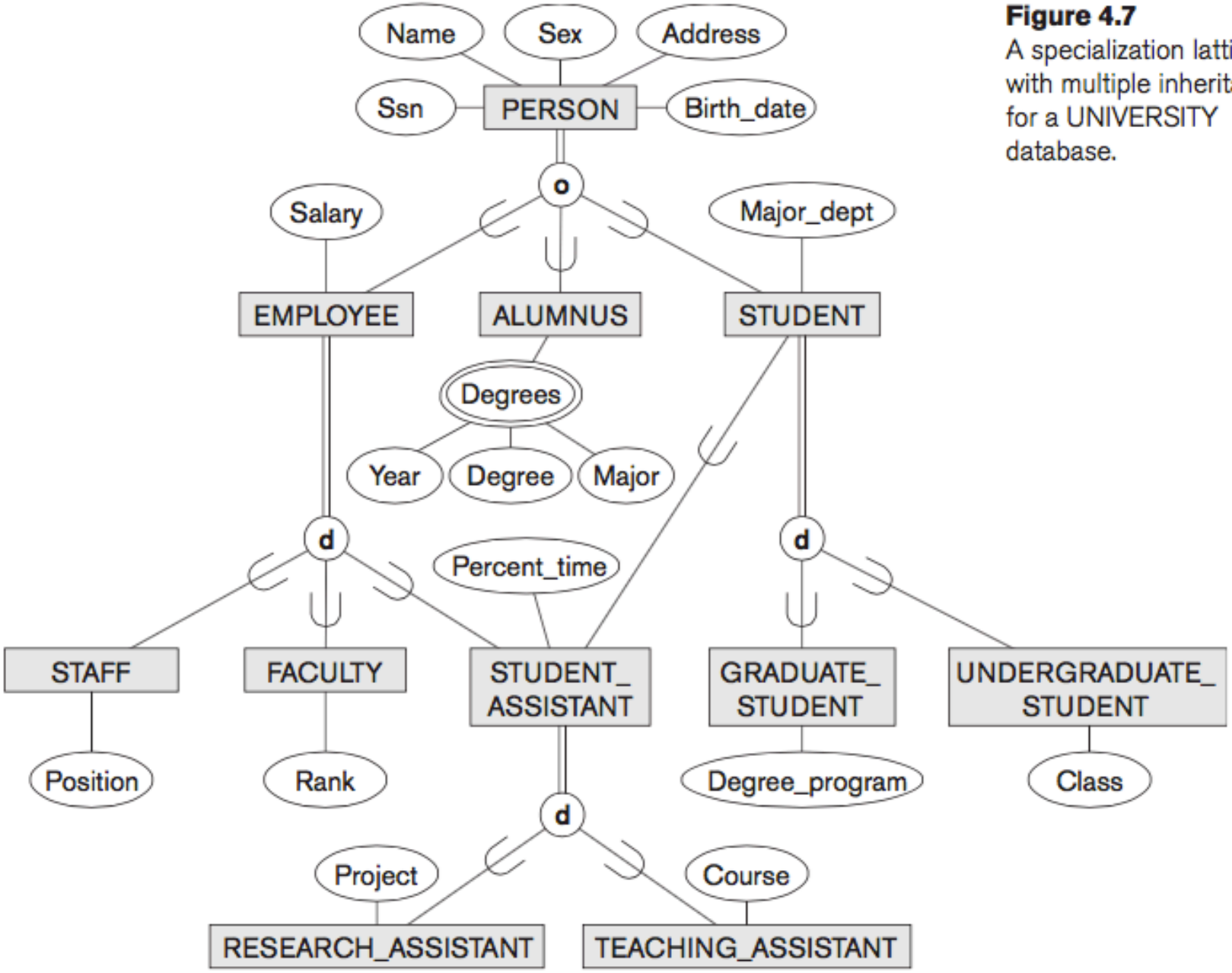


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

- Option 8C and 8D are used for the shared subclass STUDENT_ASSISTANT
- Option 8C is used in the EMPLOYEE relation (Employee_type attribute)
- Option 8D is used in the STUDENT relation (Student_assist_flag attribute)

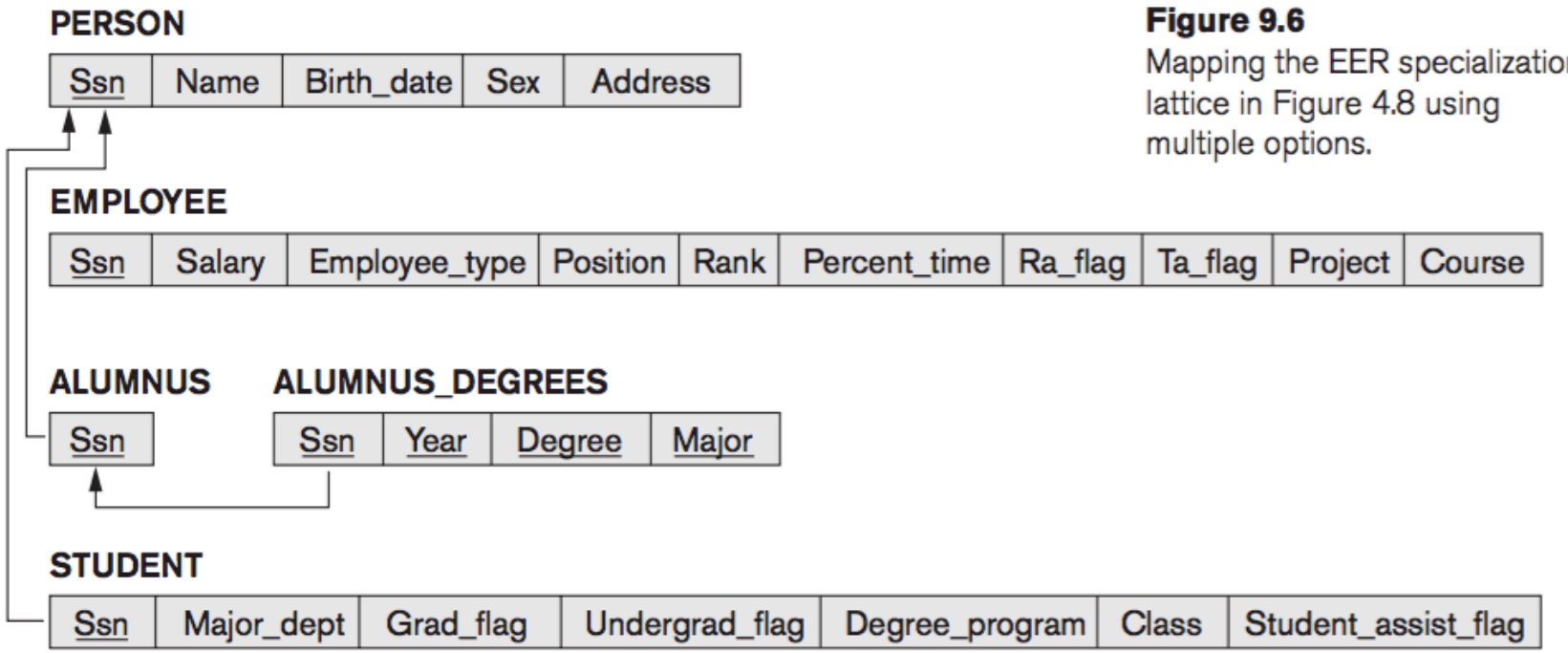


Figure 9.6
Mapping the EER specialization lattice in Figure 4.8 using multiple options.

Mapping of Categories (Union Types)

- Step 9: Mapping of Union Types (Categories)
  - Defining superclasses have different keys
  - Specify a new key attribute
    - **Surrogate key**
      - The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the relation

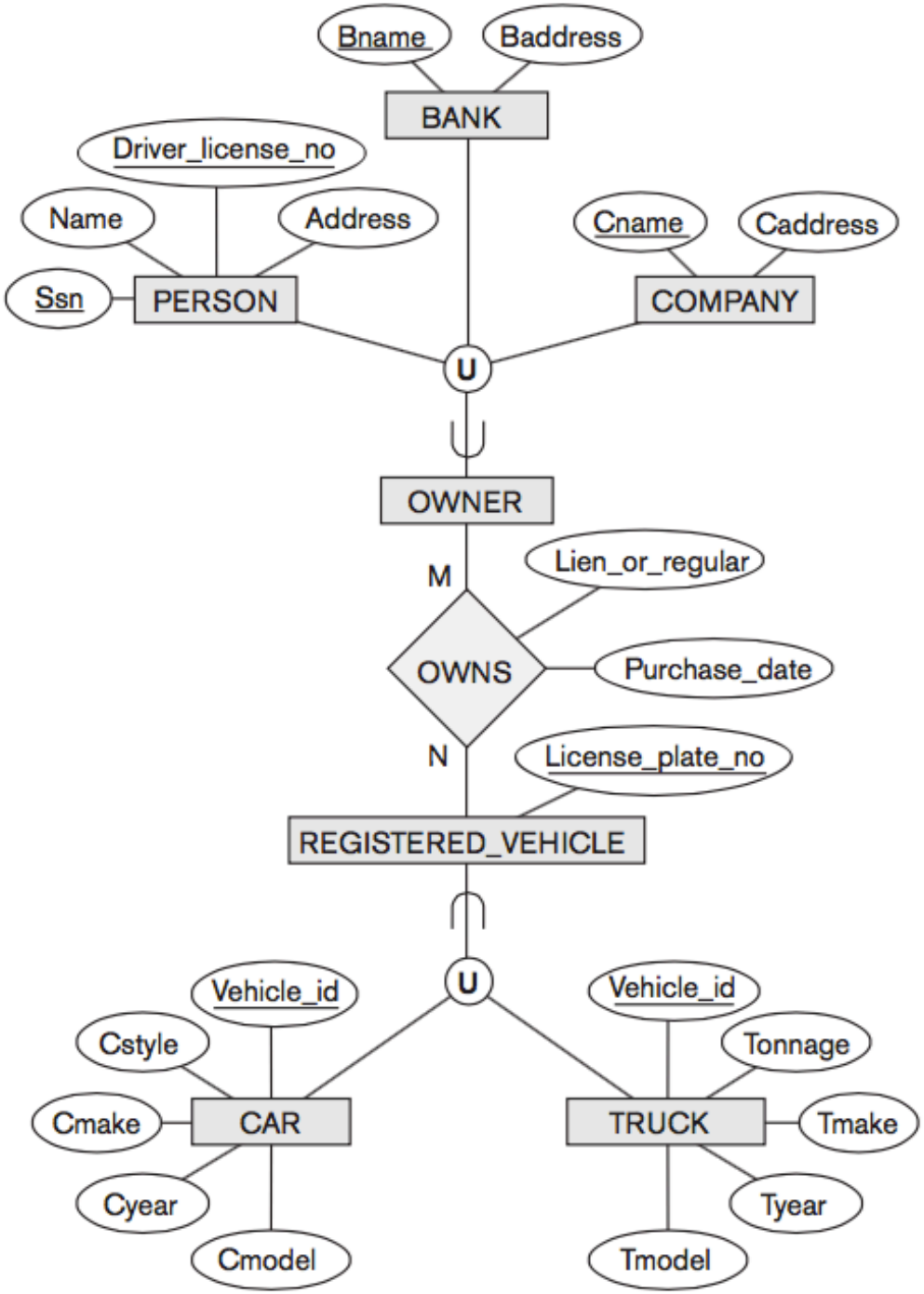
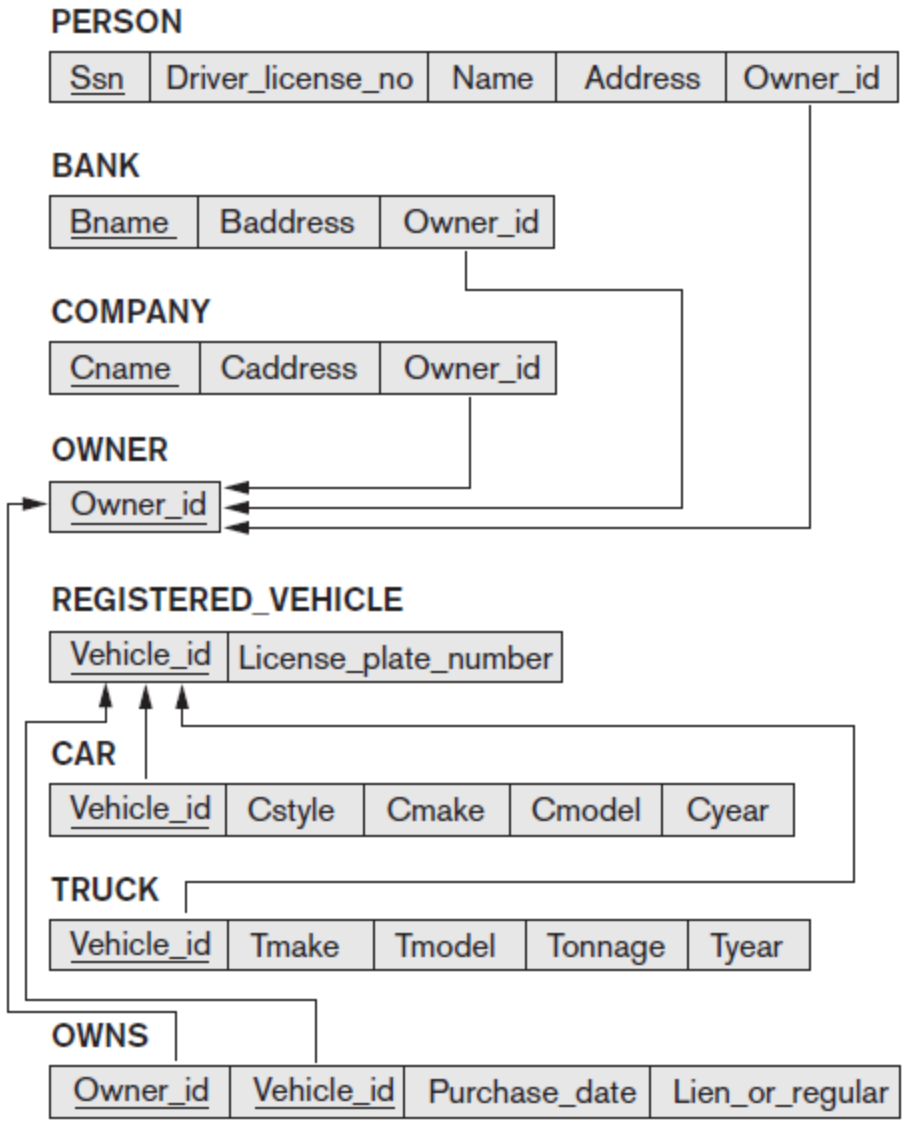


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Figure 9.7
Mapping the EER categories (union types) in Figure 8.8 to relations.



Summary

- Map conceptual schema design in the ER model to a relational database schema
  - Algorithm for ER-to-relational mapping
  - Illustrated by examples from the COMPANY database
- Include additional steps in the algorithm for mapping constructs from EER model into relational model