Chapter 9:
Mapping ER to Relational Model

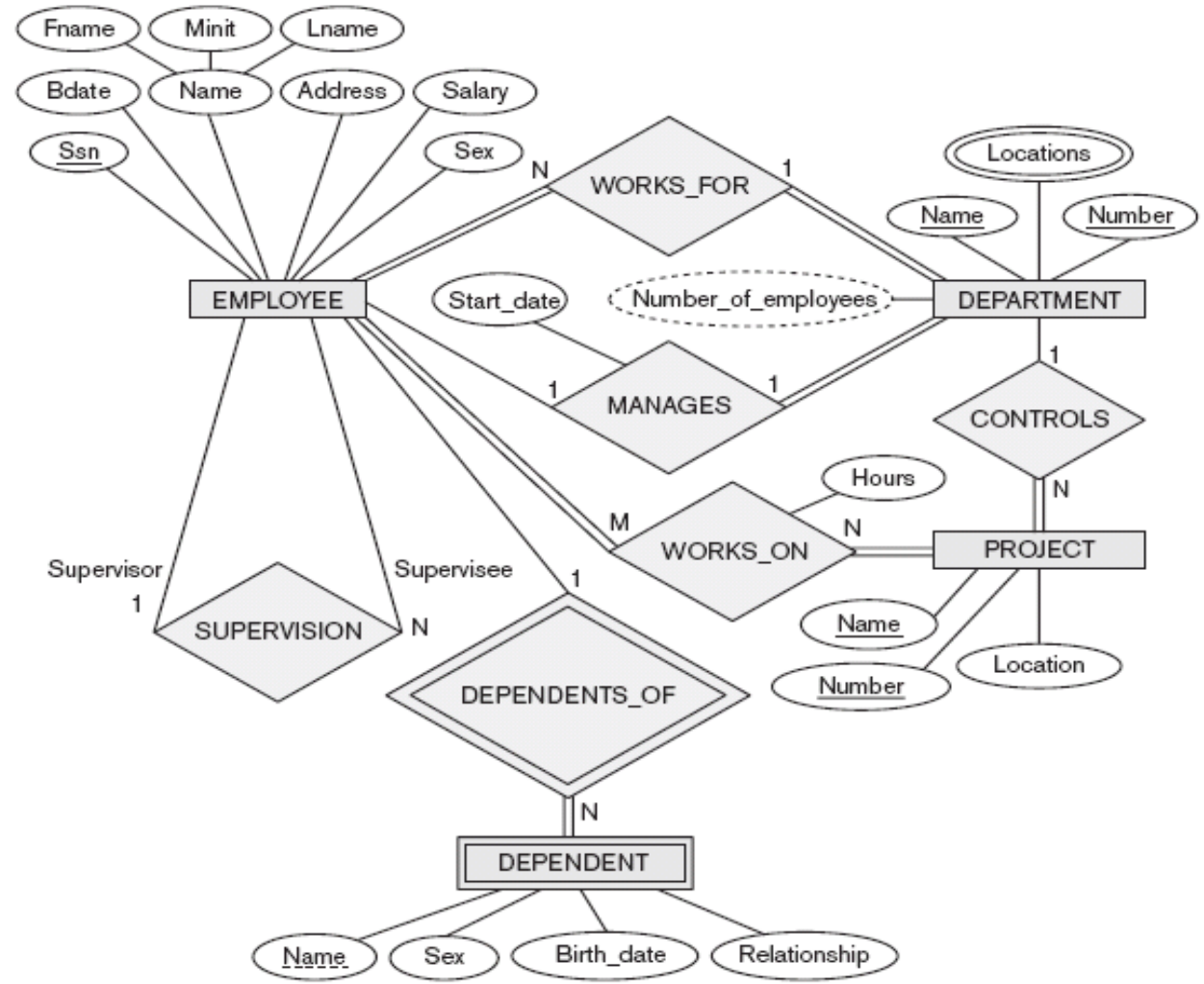
Universiteit Maastricht

The connection

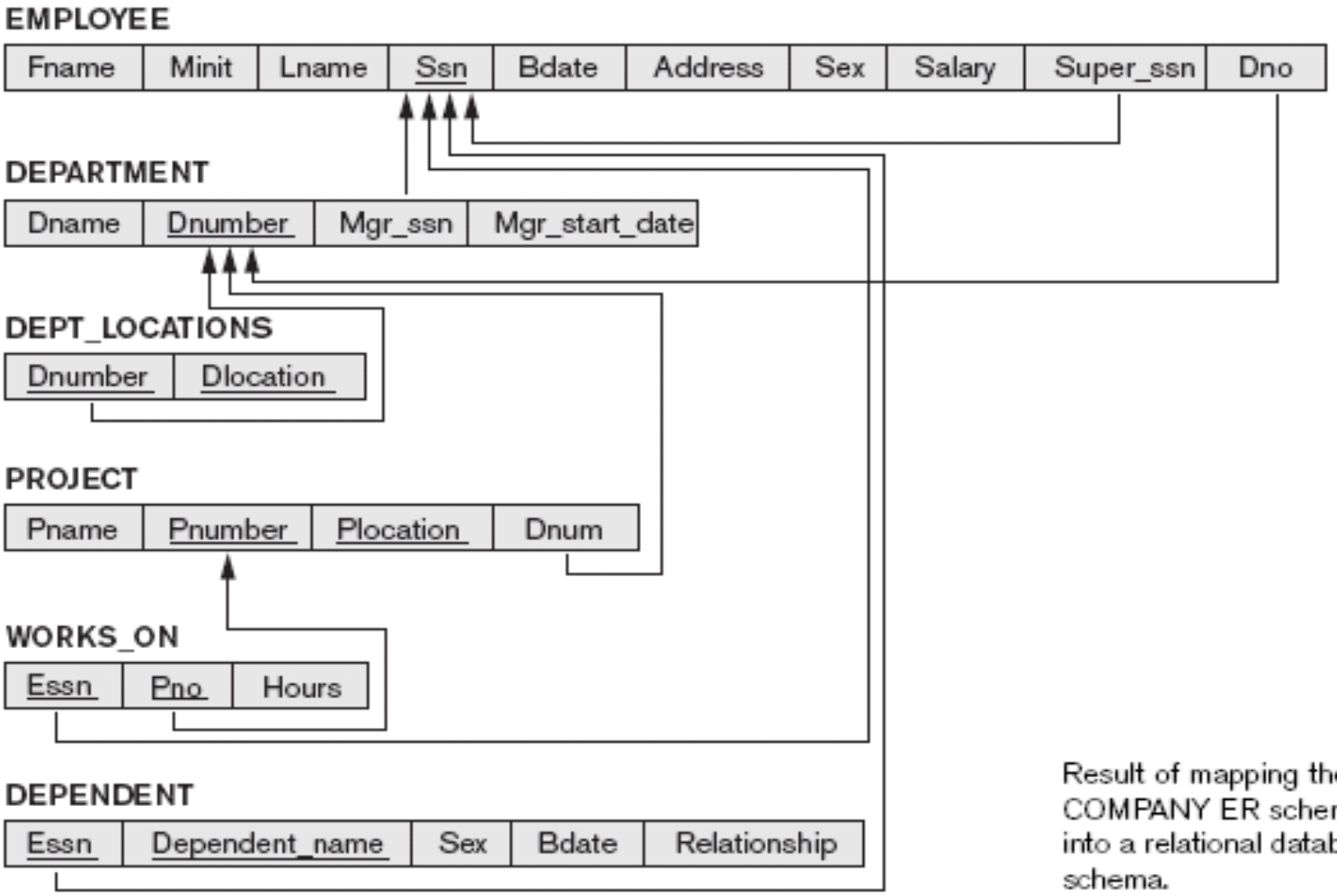
- Lecture 1: Introduction
- Lecture 2: make ER diagram from requirements, forms and texts
- Lecture 3 (today): *mapping* from ER diagrams to relational database model + Enhanced Entity Relationship Model

Start from conceptual ER diagram

The ER conceptual schema diagram for the COMPANY database.



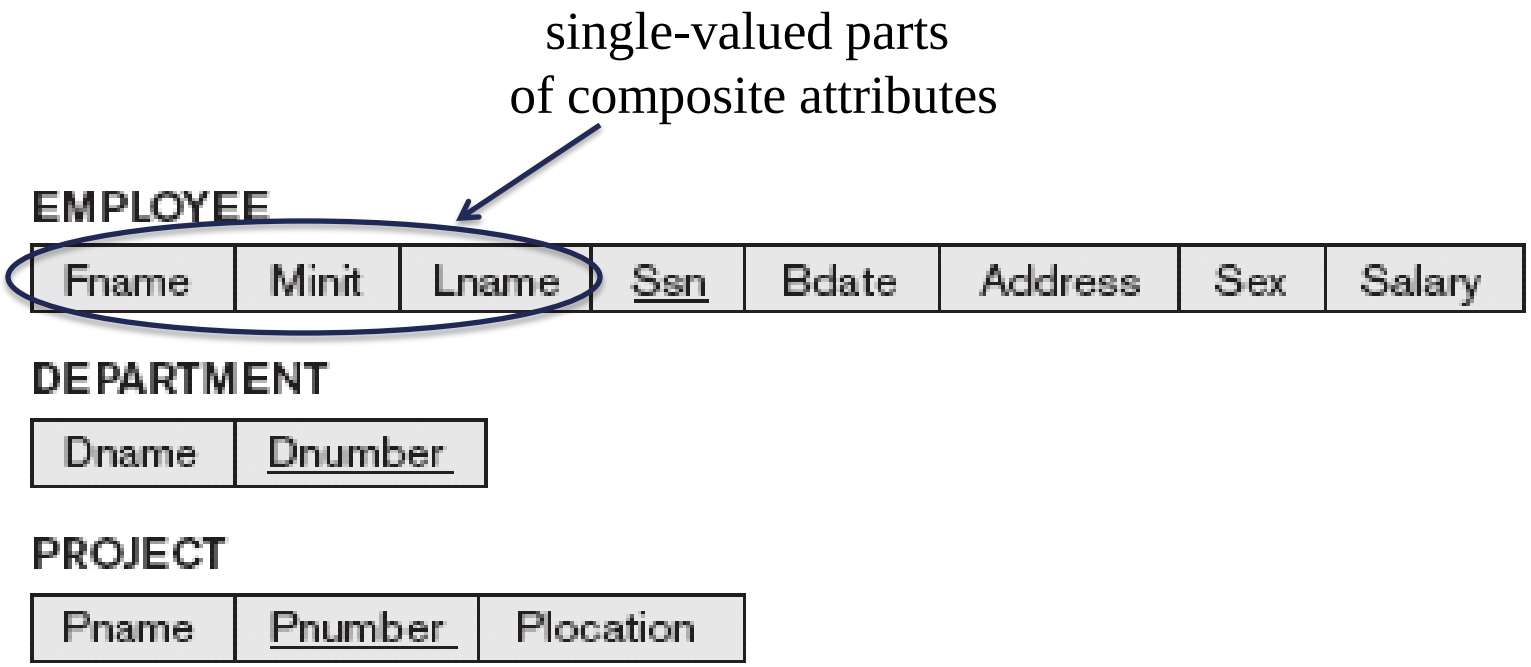
End with Relational database schema



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

Step 1: Strong entity types

- Create a relation R for each strong entity type E
- Include all single-valued attributes and all single-valued parts of composite attributes
- Appoint one of the keys in E as primary key in R


single-valued parts
of composite attributes
EMPLOYEE
Fname
Minit
Lname
Ssn
Bdate
Address
Sex
Salary
DEPARTMENT
Dname
Dnumber
PROJECT
Pname
Pnumber
Plocation

Step 2: Weak Entity types

- Make a relation R for each weak entity type W
- Add all single-valued attributes and single-valued parts of composite attributes
- Add a foreign key FK to R pointing to the primary key of the identifying entity
- The primary key of R is the foreign key FK together with a partial key of W

Step 2: the example

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 3: 1:1 relationships

- Identify relations **S** and **T** that represent the connected entity types through relationships **R**
- **1-Foreign key approach:**
  - Add to one of them (say **S**, preferably totally participating in **R**):
    - A foreign key to **T**
    - all (single-valued) attributes of the 1-1 relation

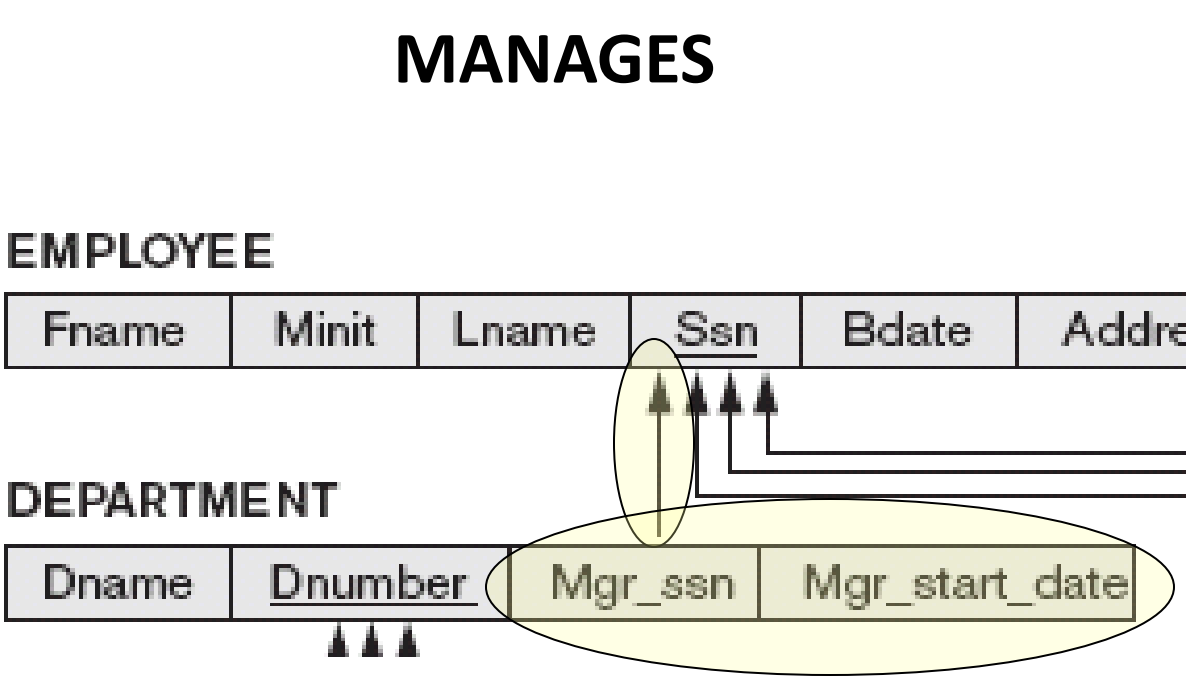
Step 3: 1:1 relationships

- **2- Merged relation approach :**
  - merge the two entity types and the relationship into a single relation. This is possible when both participations are total.
  - Ex:
    - Person + passport
    - Person + driving license

Step 3: 1:1 relationships

- **3- Cross-reference or relationship relation approach:**
  - 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.

Step 3: the example

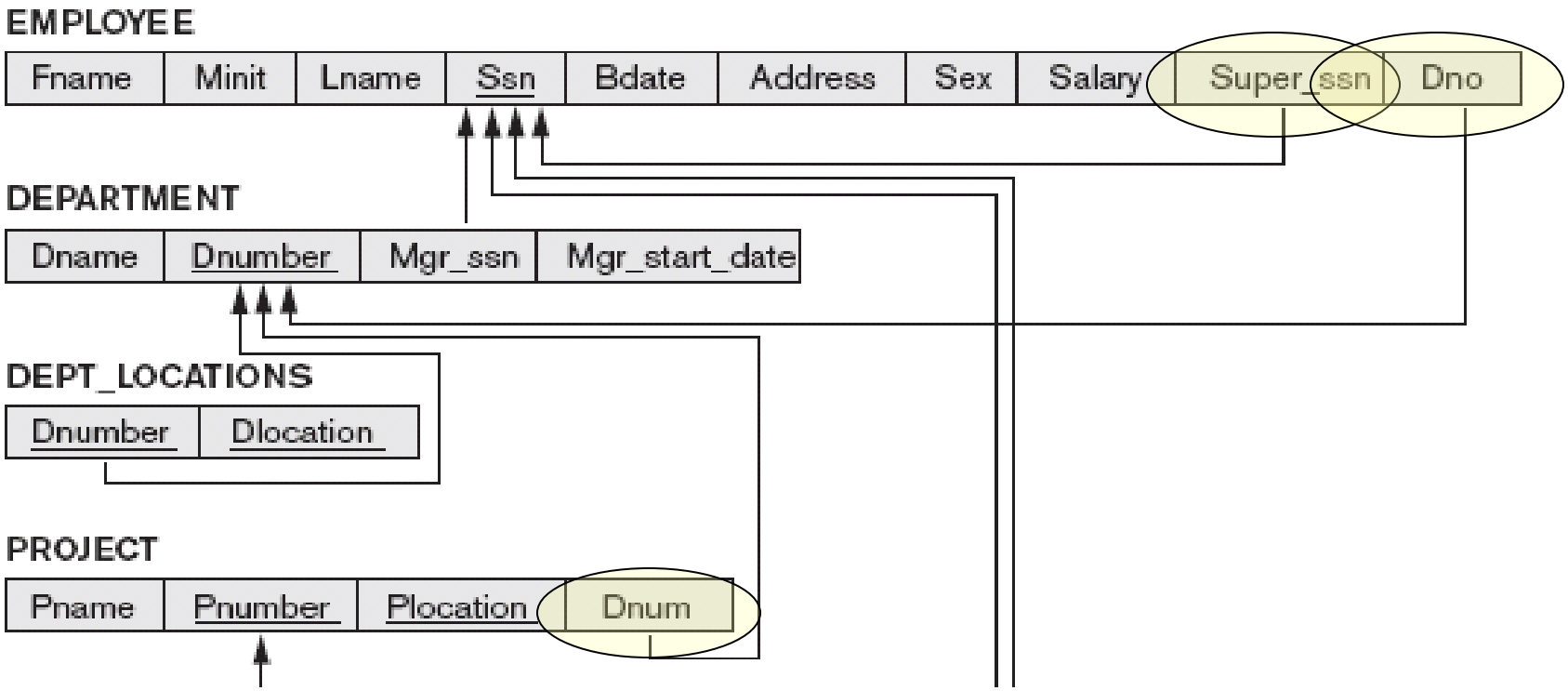


Step 4: 1:N relationships

- Identify **S** and **T** that represent the connected entity types, having **S** at the N-side
- Add a foreign key in **S**, pointing to **T**
- Add all (single-valued) attributes of the relationship **R** to **S**

Step 4: the example

WORKS_FOR, CONTROLS, SUPERVISION



Step 5: M:N relationships

- Identify **S** and **T** that represent the connected entity types
- Make a new relation **R**
- Include:
  - Foreign keys in **R** the primary keys of **S** and **T**
  - all (single-valued) attributes
  - as primary key: the combination of the two foreign keys

Step 5: the example

WORKS_ON

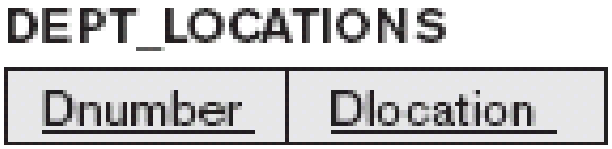
Note

- Note that step 5 **can** be used to model 1:1 and 1:N relations also.
- In that case also define a new relation **R** for the 1:1 or 1:N relation from **S** to **T**
- This is, however, never needed and only useful in rare cases (e.g., when only very few instances of **T** are related to **S**, to avoid many NULL values)

Step 6: multivalued attributes

- Make a relation **R** for each multivalued attribute **A** of entity **E** or relationship **S**.
- Locate relation **K** that represents **E** or **S**
- Add to **R** a foreign key to **K**
- The primary key of **R** is the combination of all attributes in **R**

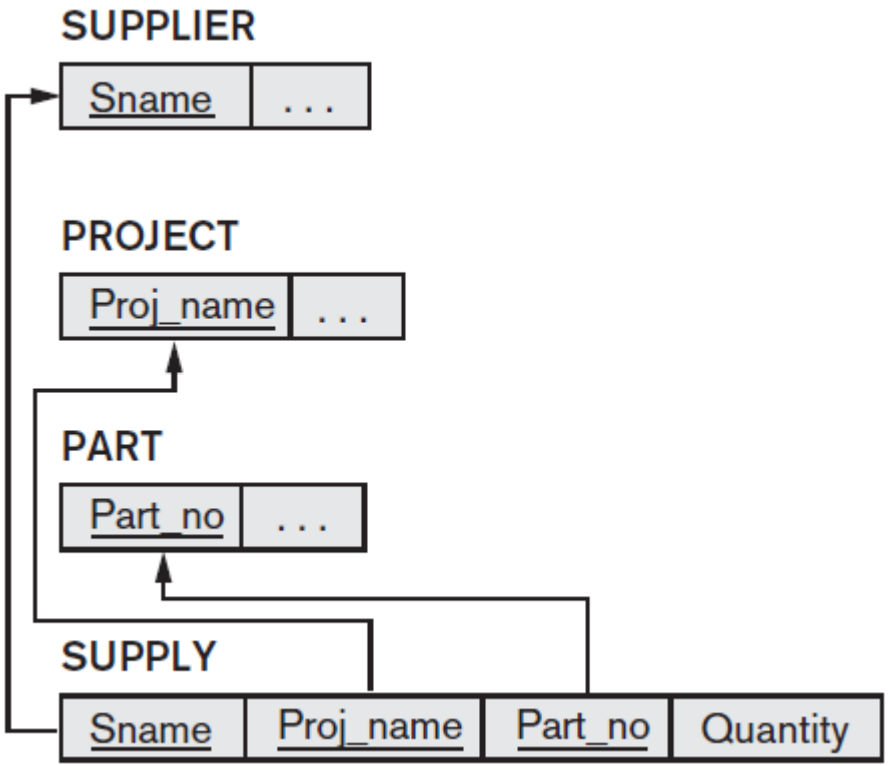
Step 6: the example



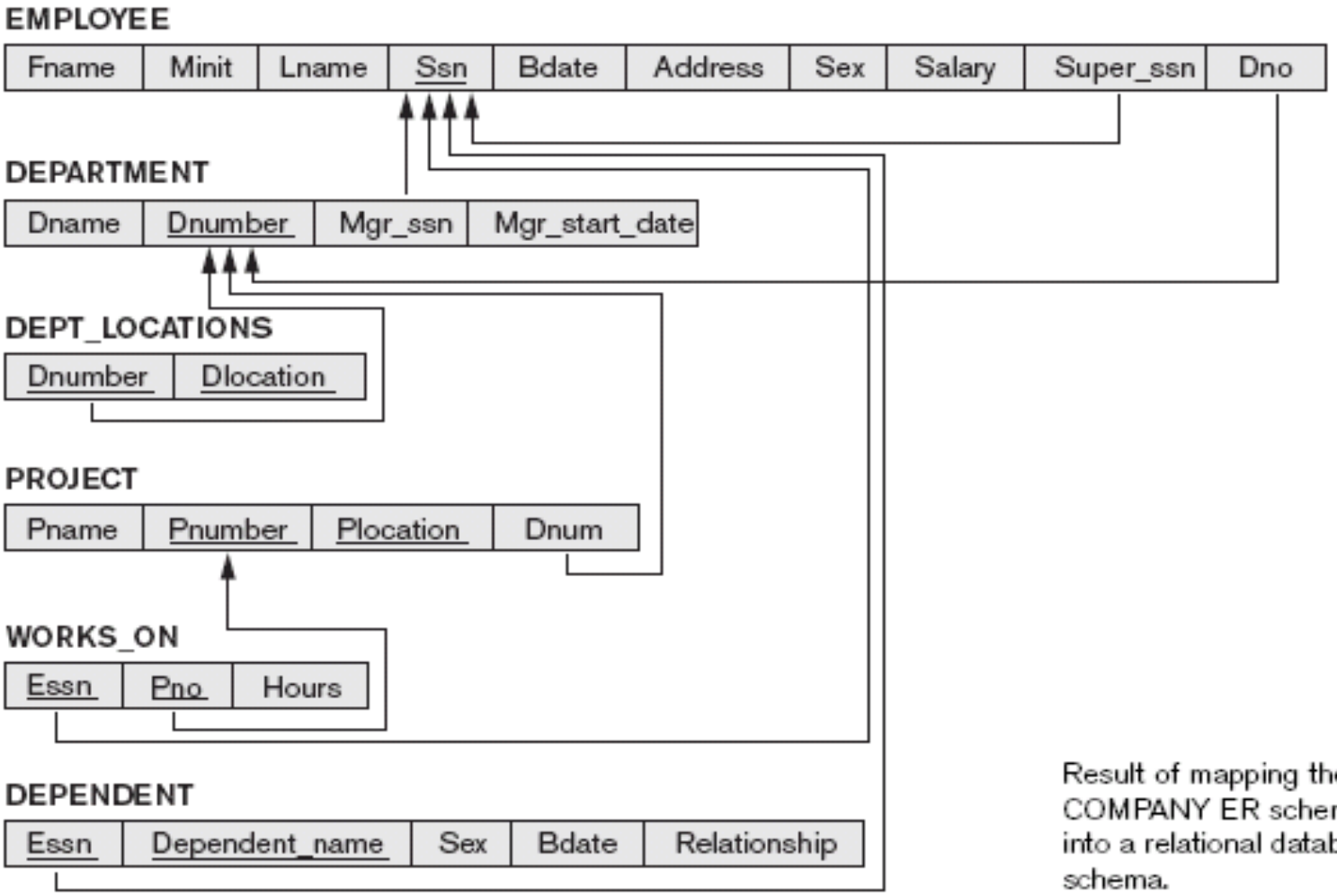
Step 7: Relationship of degree > 2

- For a relationship **R** with n>2, create a new relation **S**
- Add foreign keys in **S** the primary key of participating entity types
- Add single-valued attributes
- Primary key is the combination of foreign keys

Step 7: the example



Final relational database schema



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

Chapter 8:
Enhanced ER (EER) diagrams and schemas

Entity types

- An entity *e* can be an employee, a staff member, a PhD student and a student assistant. In which table do we store *e*?
- The traditional solutions:
  1. Store all these entity types in one table (maybe adding a “sorts” field) and include all attributes
  2. Connect PERSON and STAFF_MEMBER entities by using 1-to-1 relationships (named: “is-a”)

Problems

- But… the relationship between student and university is also valid for student assistant and university
- And… we miss restrictions to these "is-a" relations:
  - a vehicle is never a bicycle and a car
- So: we need new concepts!

Subclasses and Superclasses

- An entity type *A* is a **subclass** of *B* if all entities of type *A* are always also of type *B* (but not vice versa)
- Entity type *B* is a **superclass** of *A*
- A subclass *inherits* all attributes and relationship types of its superclass
- An entity cannot exist in the database by being a member of only the subclass
  - It must also belong to the superclass

Specialization

- *Specialization* is the process of identifying a set of subclasses of an entity type
  - Additional attributes in the subclasses are called specific (or local) attributes
  - Additional relationships on the subclasses are called specific (or local) relationships
  - There can be more than one specialization of an entity type simultaneously!
- Example:
  - specialize EMPLOYEE to {SECRETARY, ENGINEER, TECHNICIAN};
  - specialize EMPLOYEE to {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}

Example

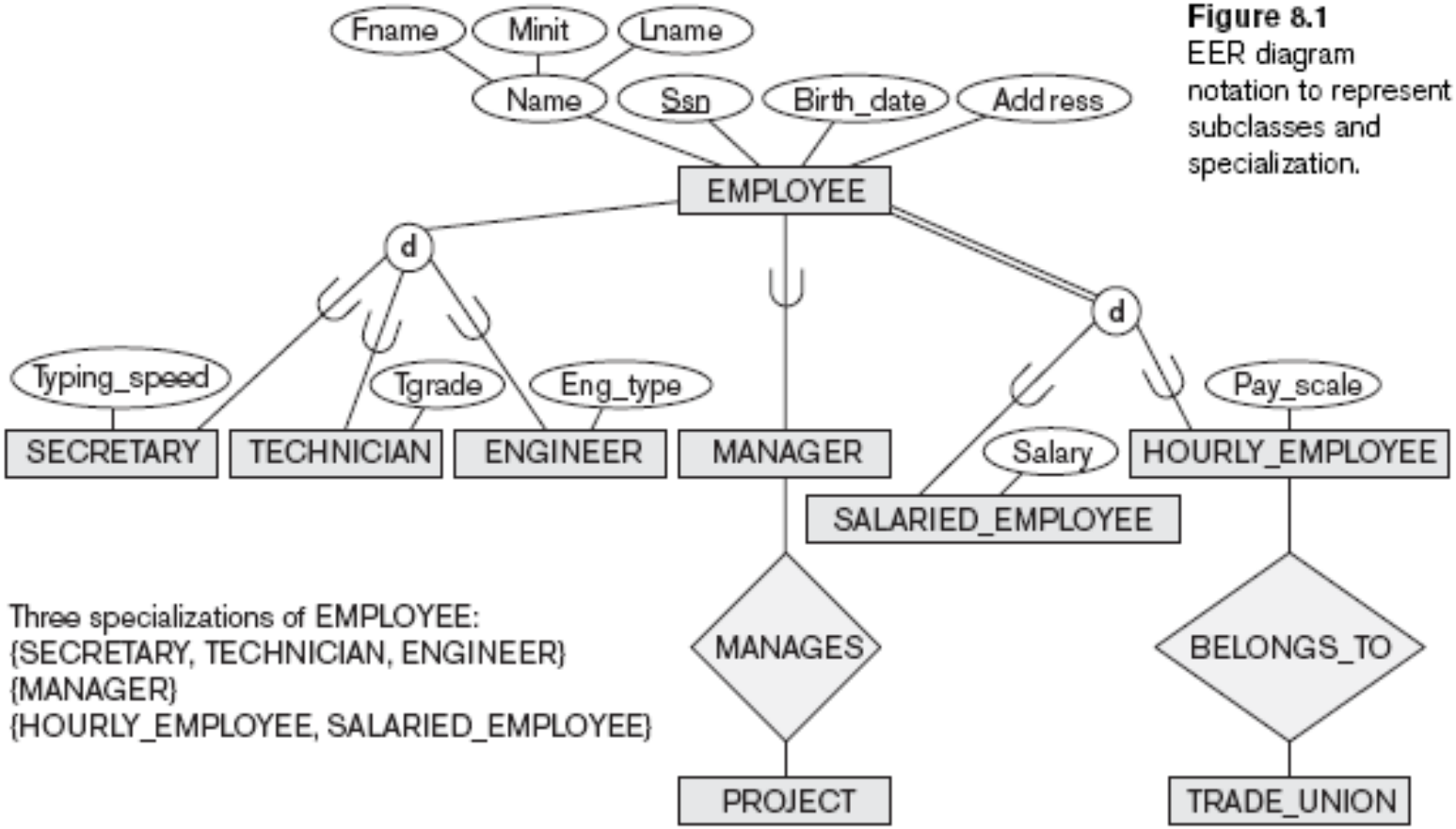
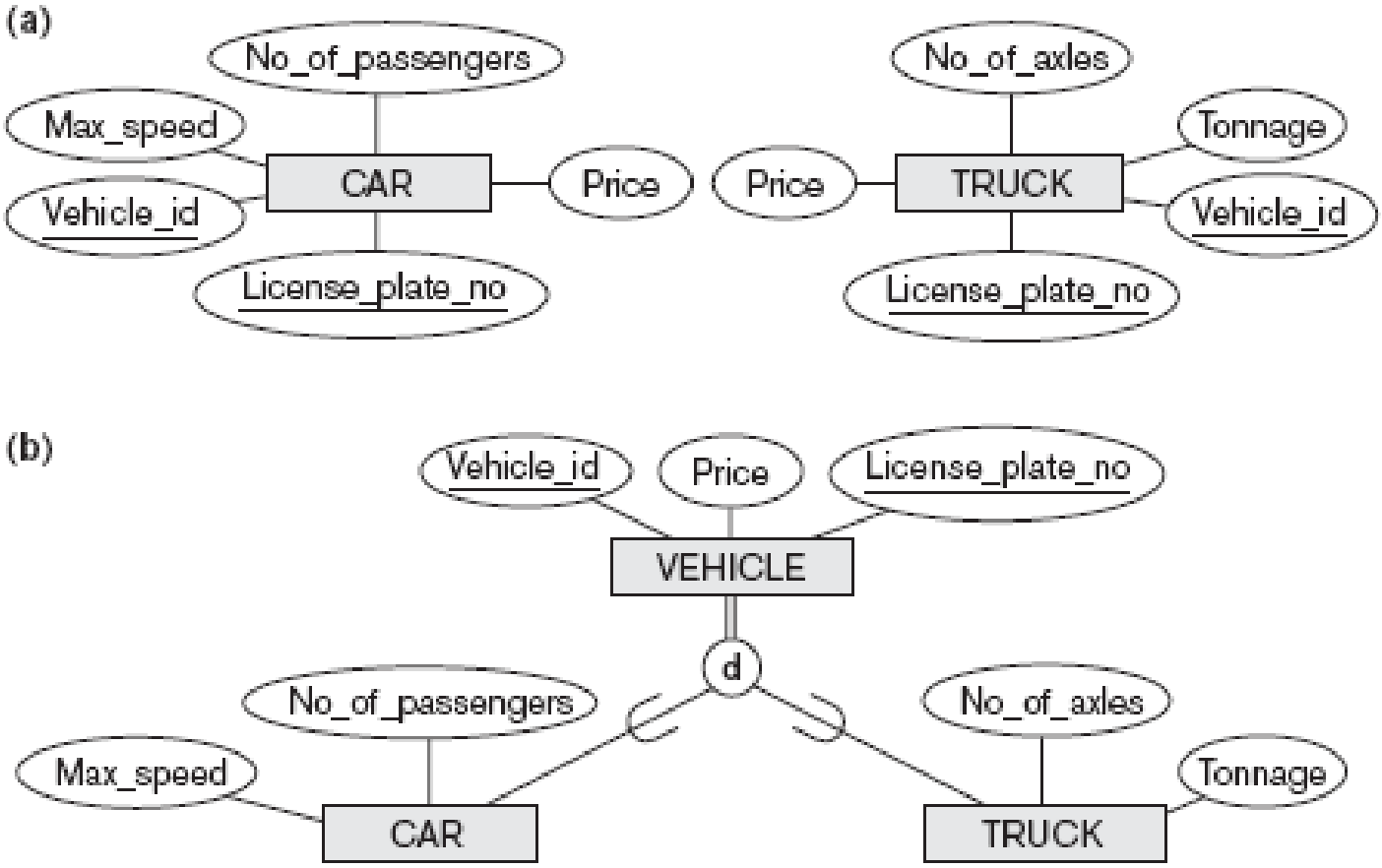


Figure 8.1
EER diagram notation to represent subclasses and specialization.

Generalization

- Generalization is the construction of an entity type that embodies the common properties (attributes and relationships) of a number of given entity types
  - The new entity type is the generalized superclass of the entity types
  - An entity type can participate in multiple generalizations
- Example:
  - generalize bicycle and car to vehicle

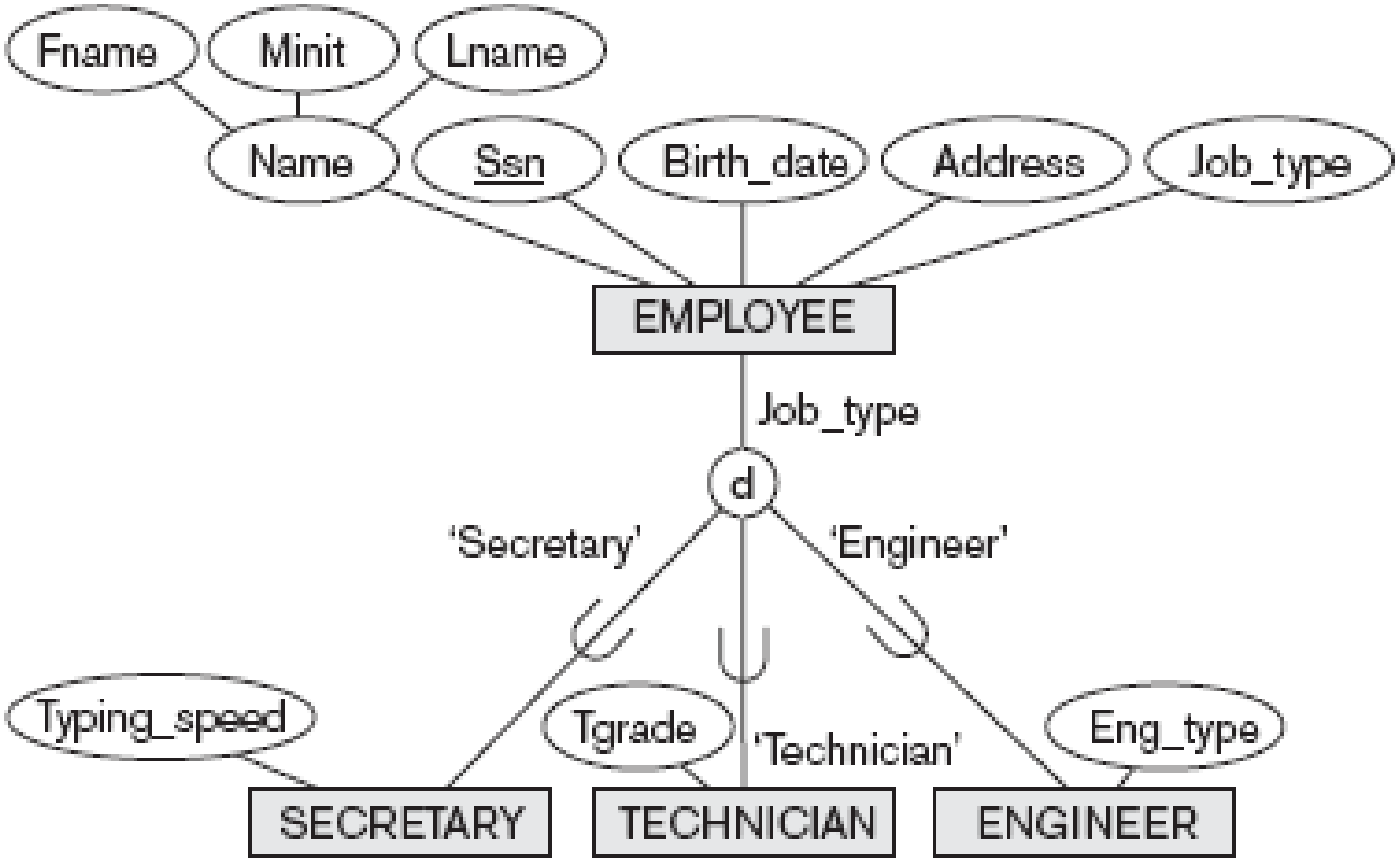
Example


(a)
No_of_passengers
Max_speed
CAR
Price
Vehicle_id
License_plate_no
No_of_axles
Tonnage
Price
TRUCK
Vehicle_id
License_plate_no
(b)
Vehicle_id
Price
License_plate_no
VEHICLE
d
No_of_passengers
Max_speed
CAR
No_of_axles
Tonnage
TRUCK

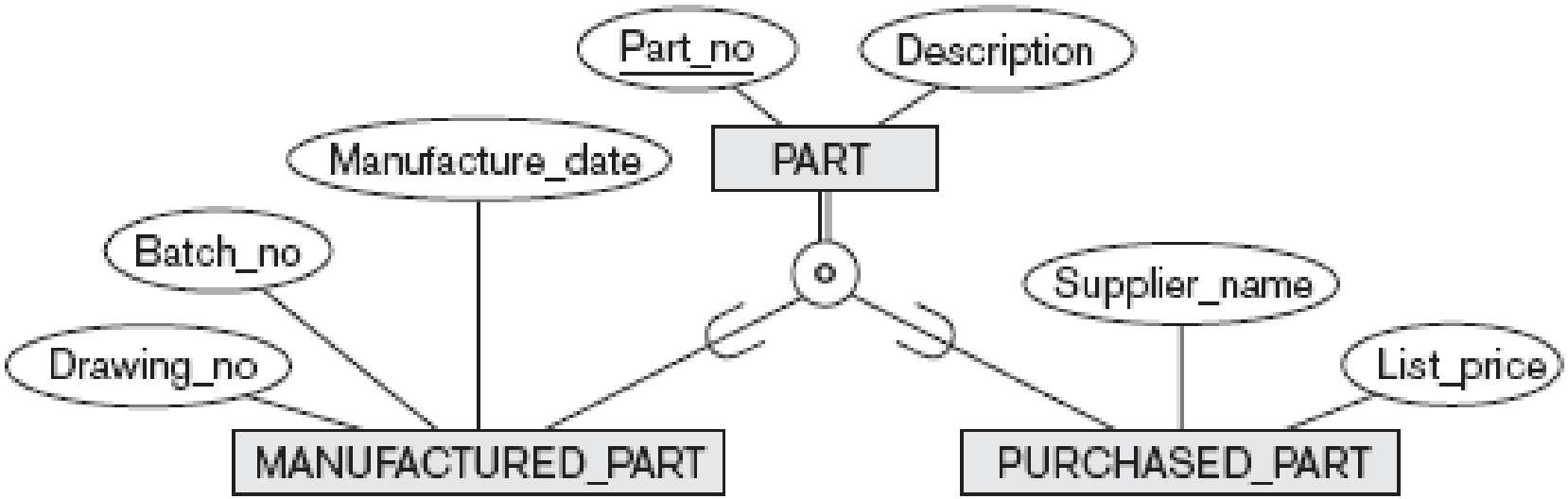
Restrictions

- *Total* or *optional* participation of the superclass:
  - **total**: double line from superclass; (every entity in the superclass must be a member of at least one subclass in the specialization.)
  - **partial**: optional participation in specialization (allows an entity not to belong to any of the subclasses.)
- *Disjoint* and o*verlapping* subclasses
  - letter **d** or letter **o** in the circle
- So we have four possible combinations
  - total/partial with d/o

Example 1: partial/disjoint



Example 2: total/overlapping

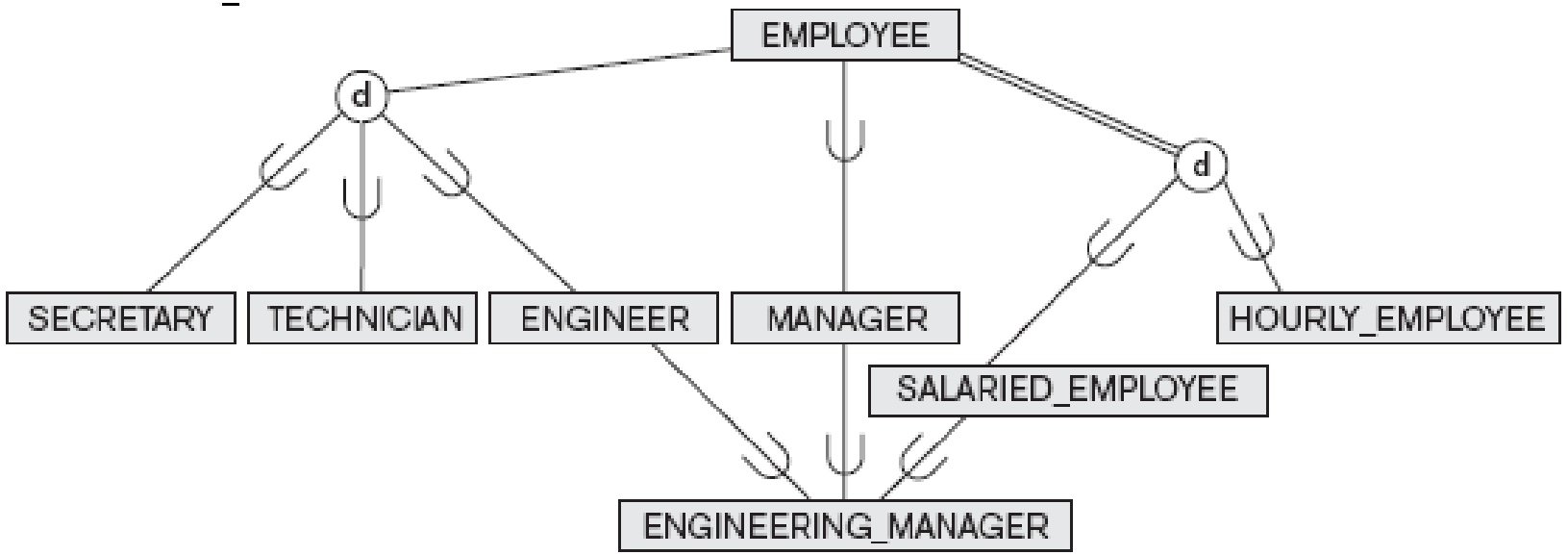


Hierarchy vs Lattice

- A **specialization hierarchy** has the constraint that every subclass participates as a *subclass* in only one *superclass/subclass* relationship
  - 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.

Example 1: specialization lattice

A specialization lattice with shared subclass ENGINEERING_MANAGER.



Example 2: specialization lattice

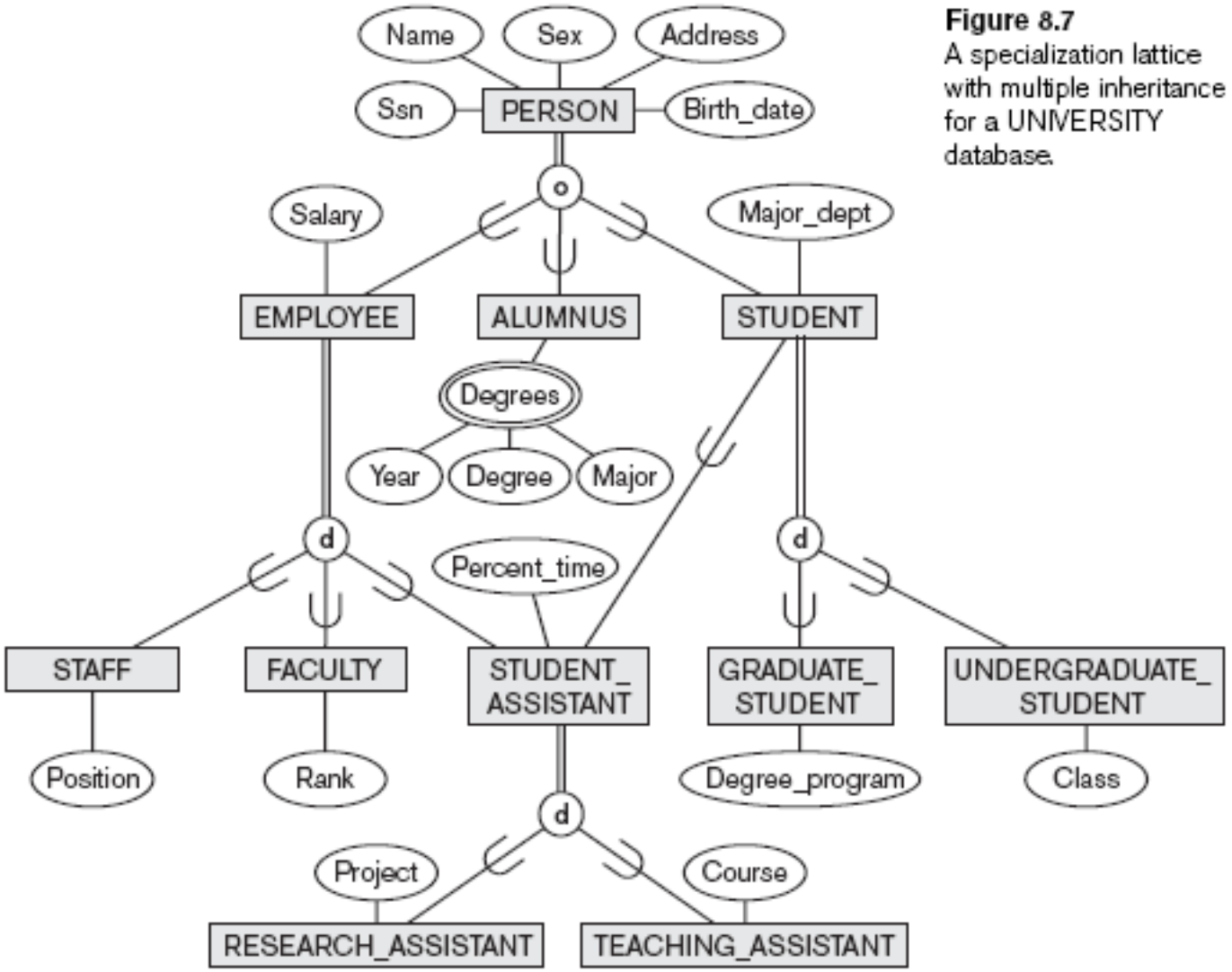


Figure 8.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

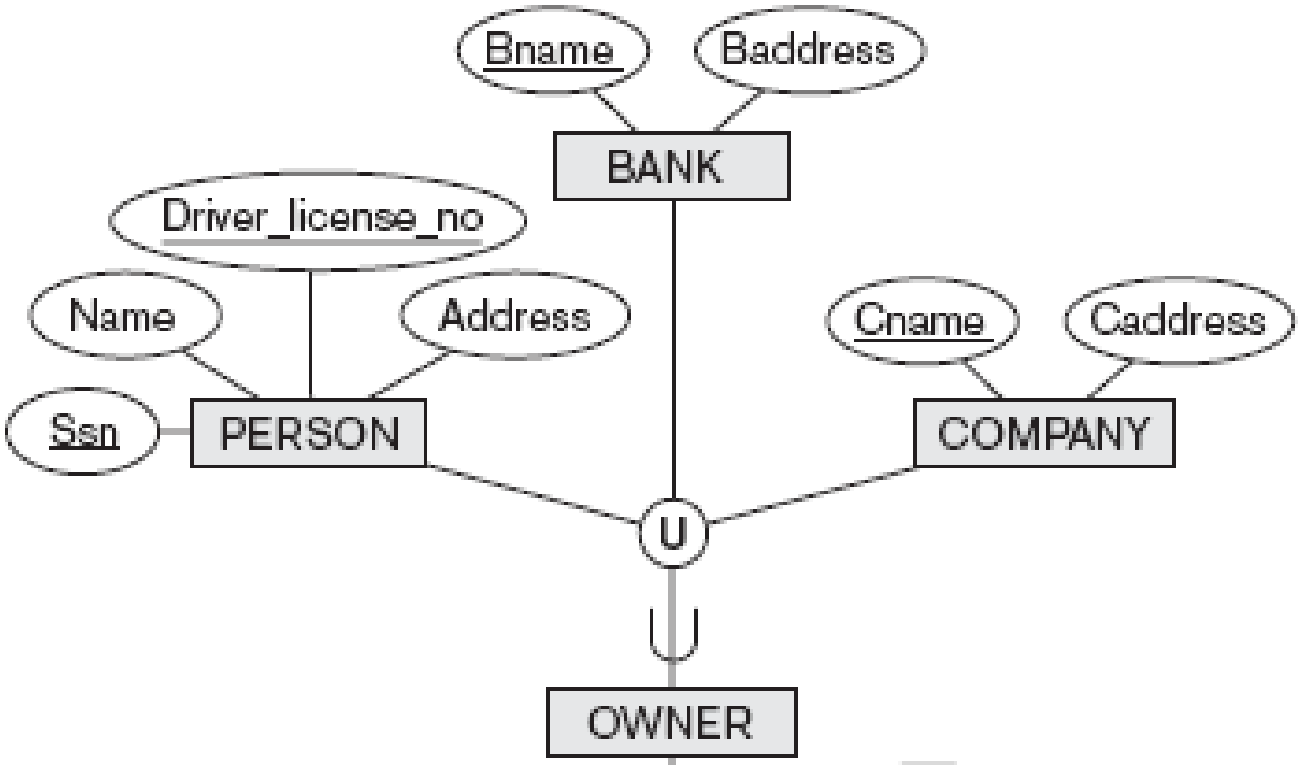
Multiple inheritance

- If a class has more than 1 superclass, it inherits **all** attributes from its superclasses (*multiple inheritance*).
- However, attributes are not duplicated!
- E.g., an ENGINEERING_MANAGER inherits all attributes (and relationships) from EMPLOYEE (once!) + from ENGINEER + from MANAGER + from SALARIED_EMPLOYEE

Categories

- Sometimes we want to create a subclass out of the *union* of some (different) entity types
  - Example: OWNER from PERSON and COMPANY
- This is called a *category* or *union-class*
- One subclass has more than one superclasses, but these are not of the same type. Can be completely different things
- An entity from the category belongs to exactly one of the superclasses, not more

Example: category



Multiple inheritance vs Categories

- ENGINEERING_MANAGER must be an ENGINEER, a MANAGER, and a SALARIED_EMPLOYEE;
  - In Union, subclass is of only one type of its superclasses
- Entries in ENGINEERING_MANAGER table must be in ENGINEER, a MANAGER, and a SALARIED_EMPLOYEE tables
  - In Union, subclass entry only exists in only one of its superclasses entries
- Entries in ENGINEERING_MANAGER are intersection of ENGINEER, a MANAGER, and a SALARIED_EMPLOYEE tables
  - In Union, subclass entries are ***subset*** of the union of its superclasses entries
- Entries in ENGINEERING_MANAGER inherent all attributes of ENGINEER, a MANAGER, and a SALARIED_EMPLOYEE tables
  - In Union, subclass inherent only attributes of its superclass

Generalization vs Categories

- Why not just to represent owner as parent super class?
  - There is not much data in common among their subclasses (i.e differences in subclasses should be minimal)
  - An OWNER should not include all BANKS and all COMPANIES and all PERSONS. It may include only some of them.
  - In generalization the participation of the **superclass** is always **total** while union (category) can be **partial** (include some of the superclasses' entries) or **total (**includes all the superclasses' entries)

Mapping EER to relation schema

- We extend our 7-step algorithm for transforming an EER diagram into a relation schema with 2 additional steps
- Step 8: mapping of generalizations / specializations (4 options: 8A-8D)
- Step 9: mapping of categories

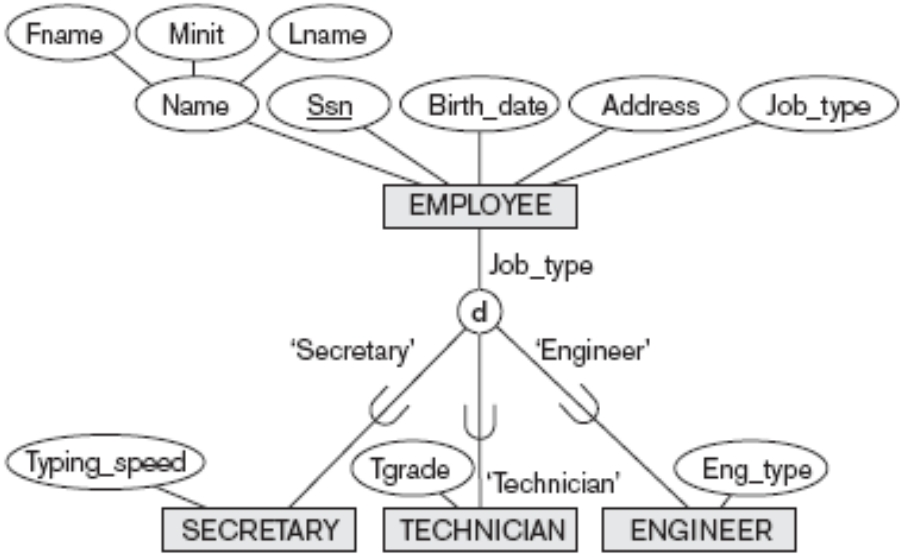
Step 8: generalization / specialization

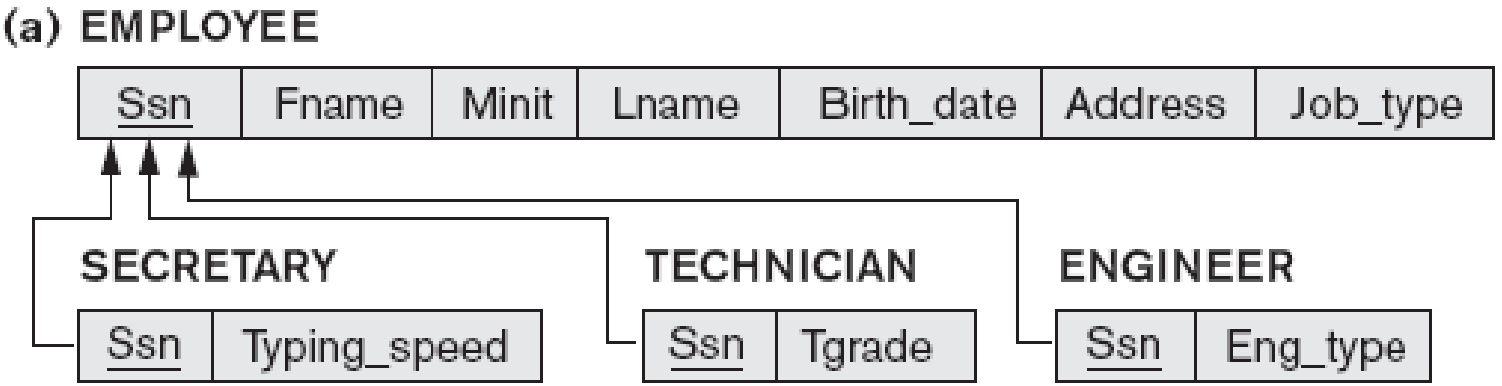
- Let's have a specialization with m subclasses $\{S_1, \ldots, S_m\}$ and superclass C, with attributes $\{k, a_1, \ldots, a_n\}$, k being the primary key,
- We may convert this EER into a relation scheme, using 1 of the 4 following options:

Step 8A: multiple relations – superclass and subclasses

- Create a relation L for C with attributes $\{k, a_1, \ldots, a_n\}$, and PK(L) = k; create a relation L_i for every subclass S_i, with attributes $\{k\}$ U {attributes of S_i}, and PK(L_i) = k
- Works for **any** specialization (total, partial, overlapping, disjoint)

Step 8A: example



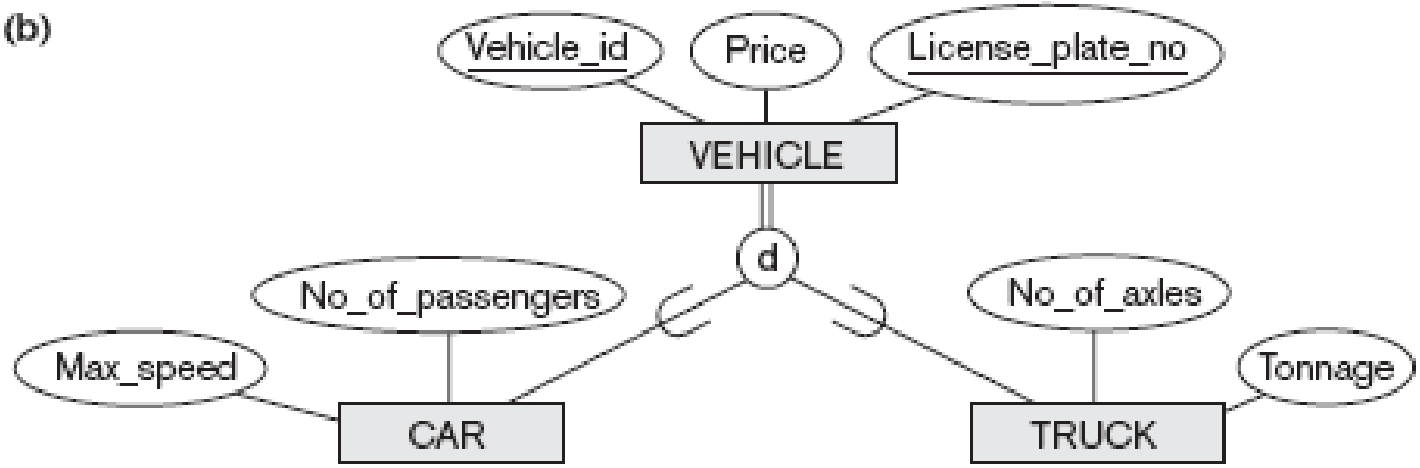


Step 8B: multiple relations – subclasses only

- Like 8A, but don't create a relation for *C*, only for the subclasses S_i, with attributes $\{k, a_1, \ldots, a_n\}$ U {attributes of S_i} and PK(L_i) = k
- Only for ***total*** specialization. Why?
- Recommended to be ***disjoint.*** Why?

Step 8B: example

(b)



(b) CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

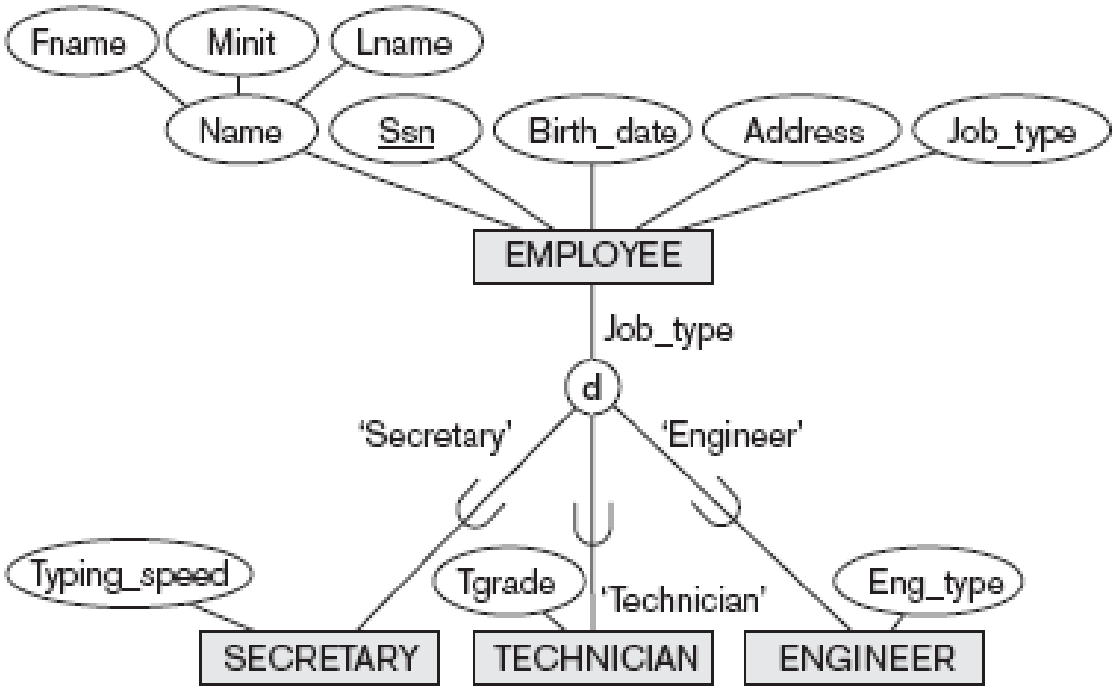
TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage

Step 8C: single relation with 1 type attribute

- Single relation L with attributes $\{k, a_1, \ldots, a_n\}$ U {attributes of S_1} U … U {attributes of S_m} U $\{t\}$, t being a *type* (or *discriminating*) attribute, and PK(L) = k
- The type attribute is only needed if it is not already contained in the superclass
- Only for ***disjoint*** specializations;
- Problem??

Step 8C: example



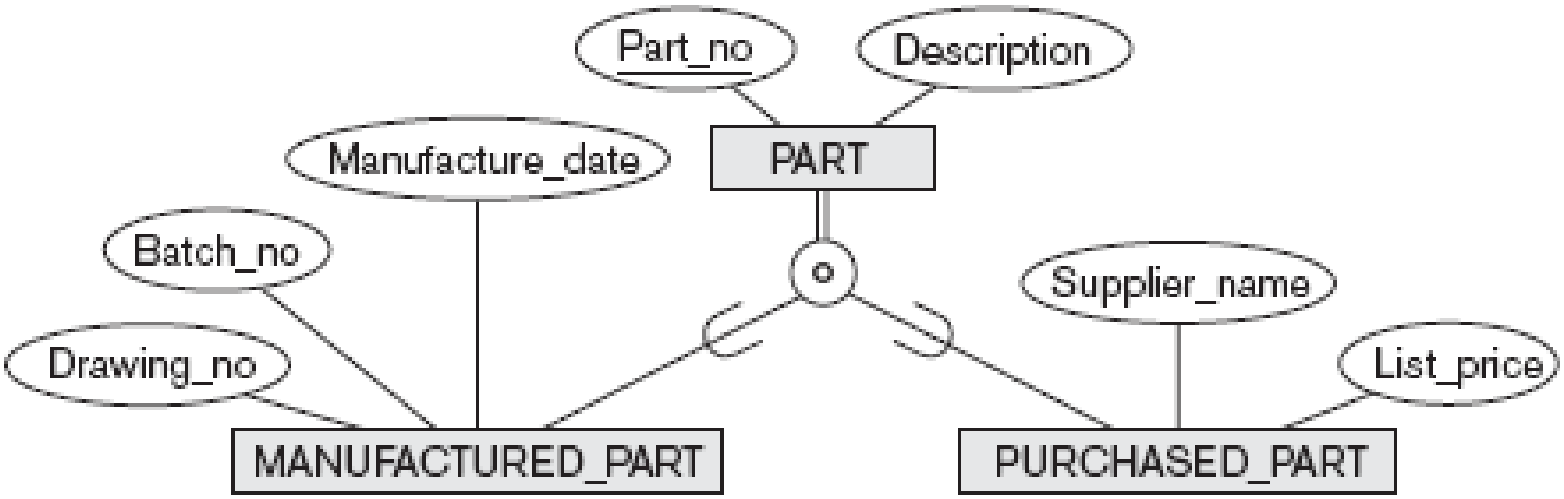
(c) **EMPLOYEE**

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type

Step 8D: single relation with multiple type attributes

- Single relation L with attributes $\{k, a_1, \ldots, a_n\}$ U {attributes of S_1} U … U {attributes of S_m} U $\{t_1, \ldots, t_m\}$, t_i being Boolean *type* (or *discriminating*) attributes, and PK(L) = k
- Useful for **overlapping** specializations (but works also for ***disjoint***);

Step 8D: example



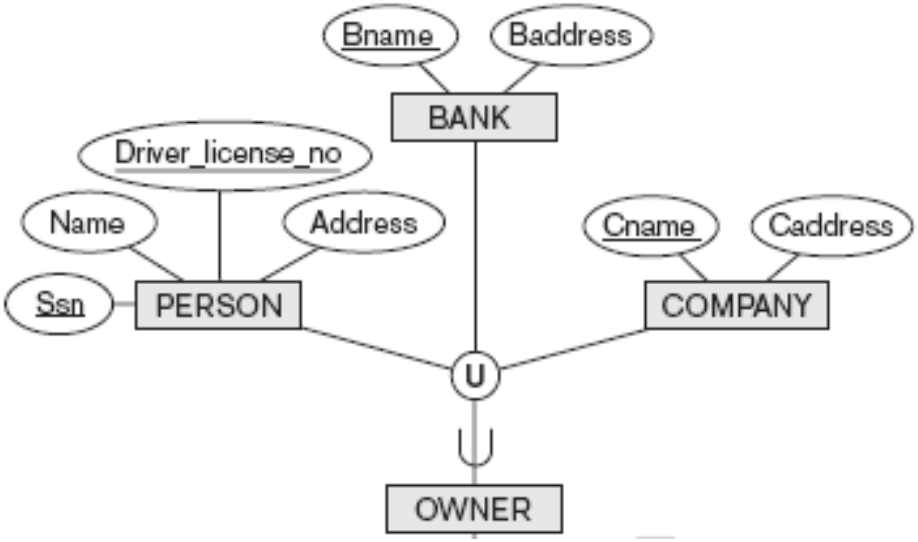
(d) PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price

Step 9: mapping of union types (categories)

- Generate a special relation with only 1 attribute, the so-called *surrogate* key, for the category.
- Add a forign key to this *surrogate* key to all superclasses relations.

Step 9: example



PERSON

Ssn	Driver_license_no	Name	Address	Owner_id

BANK

Bname	Baddress	Owner_id

COMPANY

Cname	Caddress	Owner_id

OWNER

Owner_id