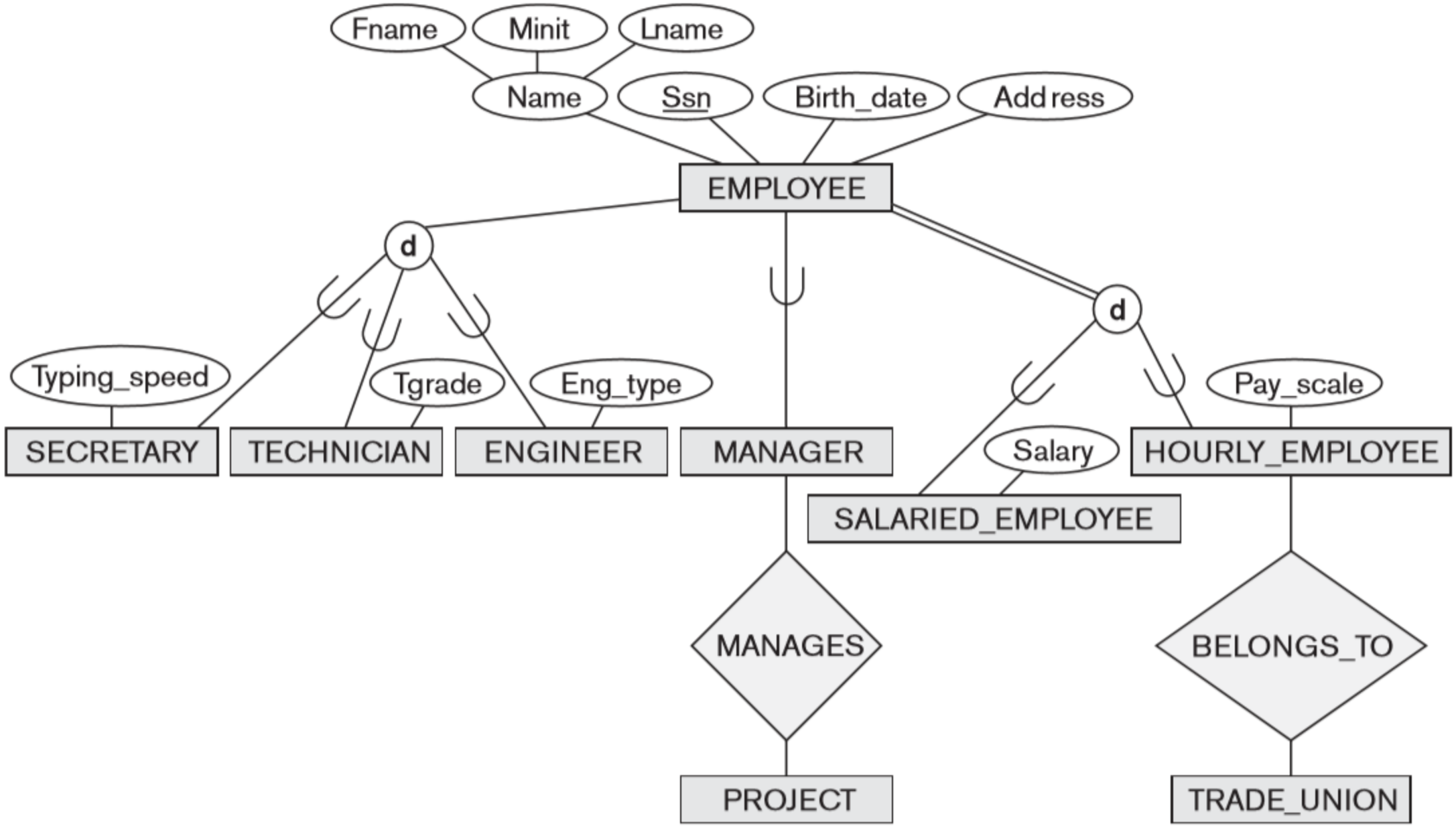
The Enhanced Entity-Relationship (EER) Model

Part 1

Subclasses, Superclasses, and Inheritance

- In many cases an entity type has numerous subgroupings or subtypes of its entities that are meaningful and need to be represented explicitly because of their significance to the database application.
- For example, the entities that are members of the EMPLOYEE entity type may be distinguished further into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOYEE, and so on.
- The set of entities in each of the latter groupings is a subset of the entities that belong to the EMPLOYEE entity set, meaning that every entity that is a member of one of these subgroupings is also an employee.
- We call each of these subgroupings a **subclass** or **subtype** of the EMPLOYEE entity type, and the EMPLOYEE entity type is called the **superclass** or **supertype** for each of these subclasses.


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
EMPLOYEE
d
d
Typing_speed
Tgrade
Eng_type
Pay_scale
SECRETARY
TECHNICIAN
ENGINEER
MANAGER
Salary
HOURLY_EMPLOYEE
SALARIED_EMPLOYEE
MANAGES
BELONGS_TO
PROJECT
TRADE_UNION

- We call the relationship between a superclass and any one of its subclasses a **superclass/subclass** or **supertype/subtype** or simply **class/subclass** relationship.
- In our previous example, `EMPLOYEE/SECRETARY` and `EMPLOYEE/TECHNICIAN` are two class/subclass relationships.
- Notice that a member entity of the subclass represents the *same real-world entity* as some member of the superclass

- An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass.
- Such an entity can be included optionally as a member of any number of subclasses.
- For example, a salaried employee who is also an engineer belongs to the two subclasses `ENGINEER` and `SALARIED_EMPLOYEE` of the `EMPLOYEE` entity type.
- However, it is not necessary that every entity in a superclass is a member of some subclass.

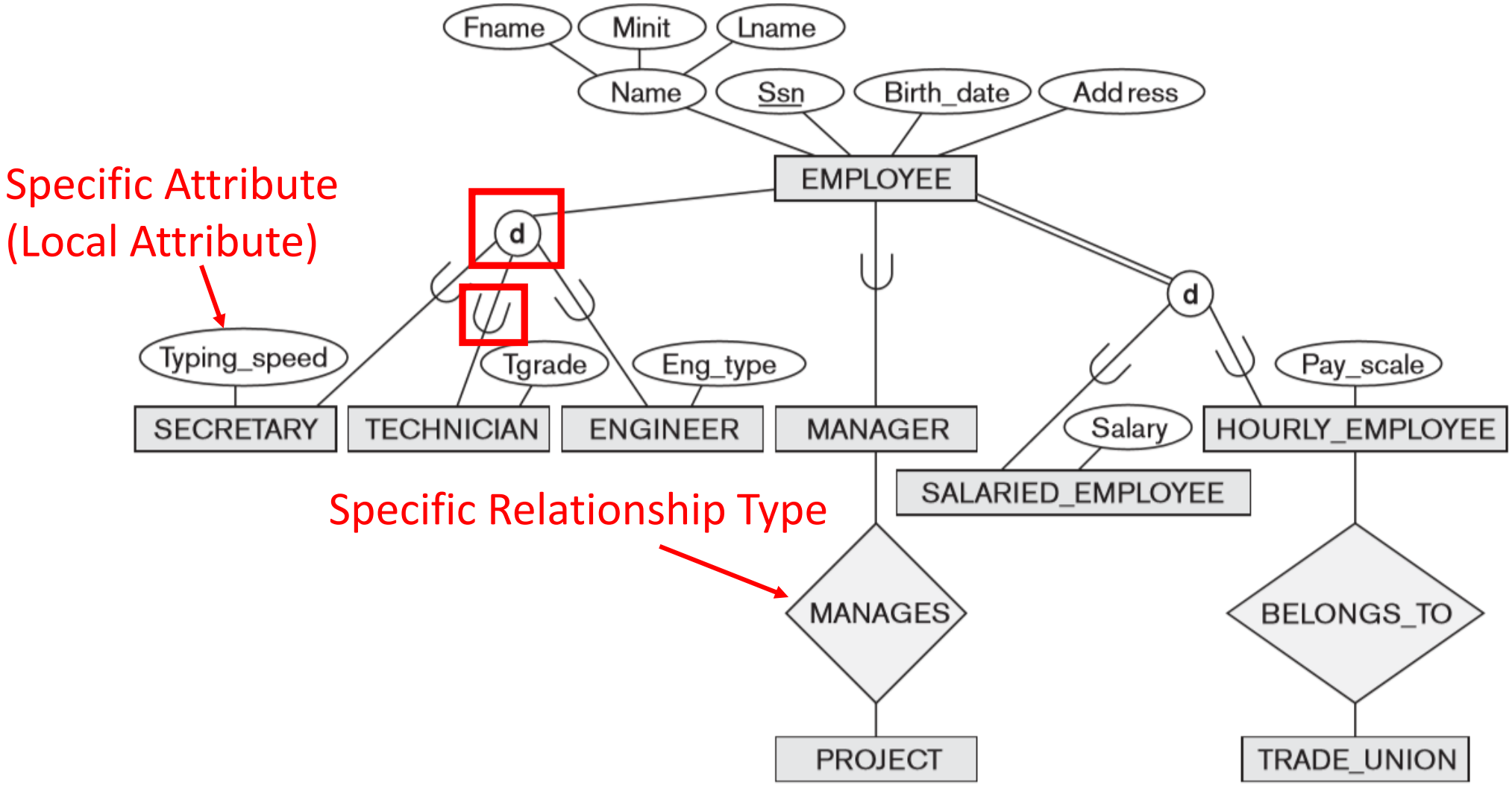
- An important concept associated with subclasses (subtypes) is that of **type inheritance**.
- Recall that the *type* of an entity is defined by the attributes it possesses and the relationship types in which it participates.
- Because an entity in the subclass represents the same real-world entity from the superclass, it should possess values for its specific attributes *as well as* values of its attributes as a member of the superclass.
- We say that an entity that is a member of a subclass **inherits** all the attributes of the entity as a member of the superclass.
- The entity also inherits all the relationships in which the superclass participates.

Specialization and Generalization

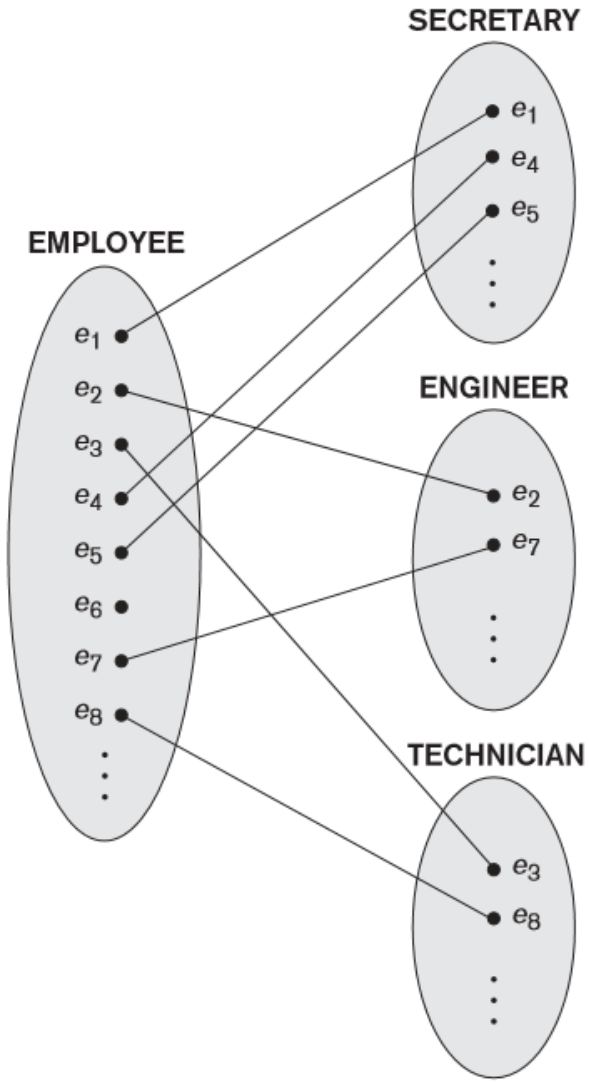
Specialization

- **Specialization** is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization.
- The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.
- For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee entity.
- We may have several specializations of the same entity type based on different distinguishing characteristics.
- For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}; this specialization distinguishes among employees based on the *method of pay*.

- The figure below shows how we represent a specialization diagrammatically in an EER diagram.



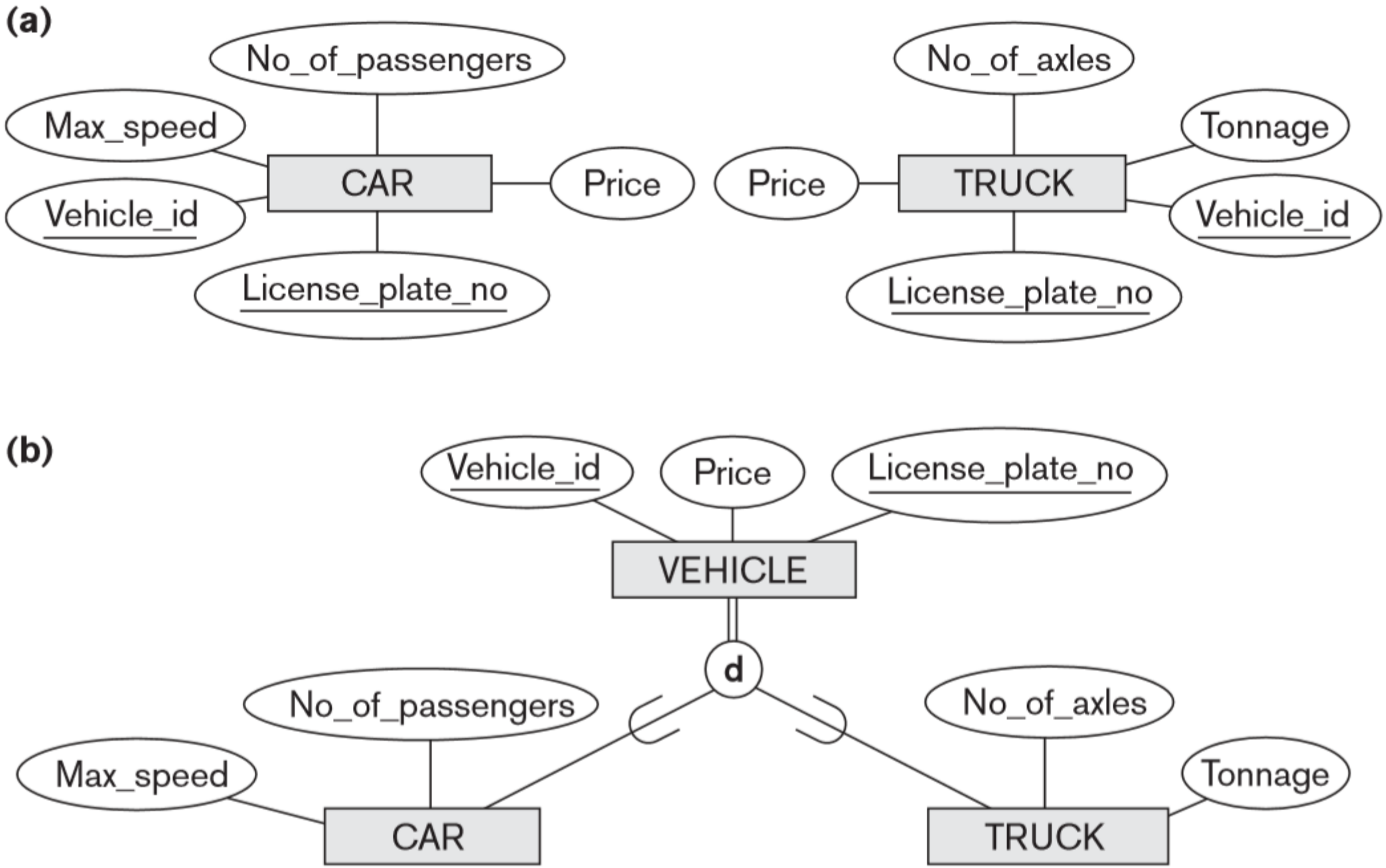
- Instances of a specialization



- There are two main reasons for including class/subclass relationships and specializations in a data model.
- The first is that certain attributes may apply to some but not all entities of the superclass.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass.

Generalization

- We can think of a *reverse process* of abstraction in which we suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass** of which the original entity types are special **subclasses**.

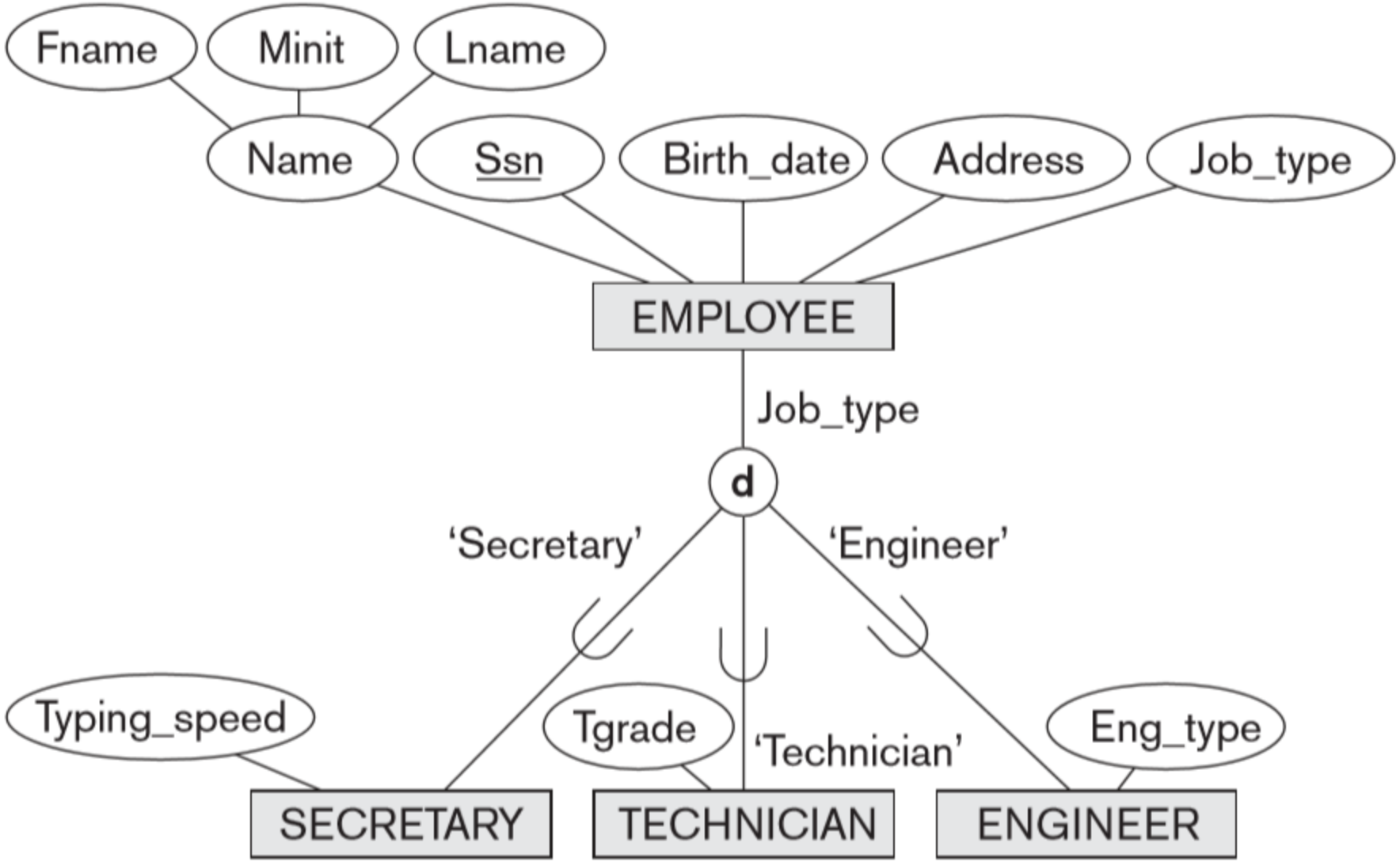

(a)
No_of_passengers
Max_speed
CAR
Price
Vehicle_id
License_plate_no
No_of_axles
Tonnage
Price
TRUCK
Vehicle_id
License_plate_no
(b)
Vehicle_id
Price
License_plate_no
VEHICLE
d
No_of_passengers
Max_speed
CAR
No_of_axles
Tonnage
TRUCK

Constraints and Characteristics of Specialization and Generalization Hierarchies

Constraints on Specialization and Generalization

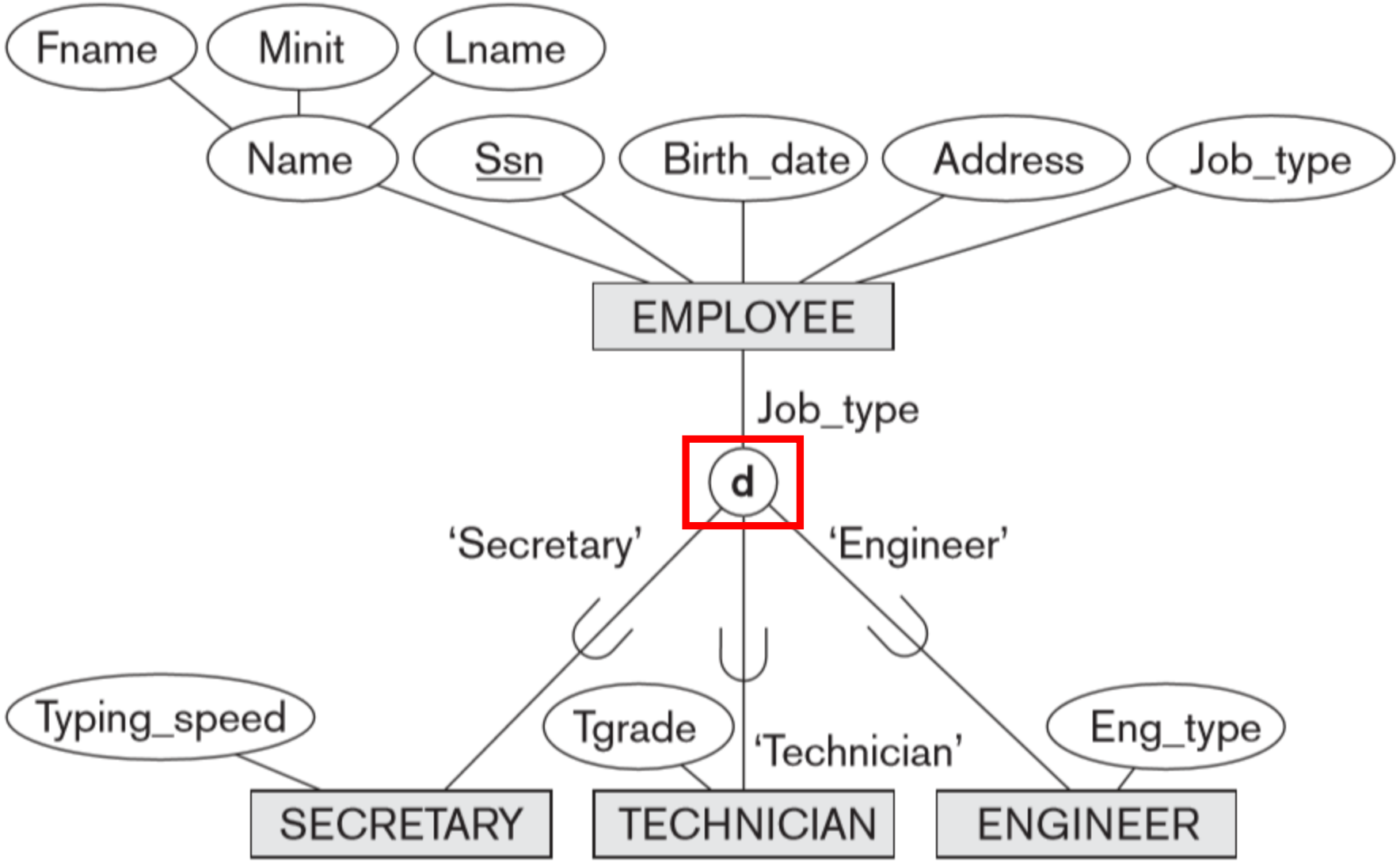
- In some specializations we can determine exactly the entities that will become members of each subclass by placing a condition on the value of some attribute of the superclass.
- Such subclasses are called **predicate-defined** (or **condition-defined**) **subclasses**.
- For example, if the `EMPLOYEE` 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.

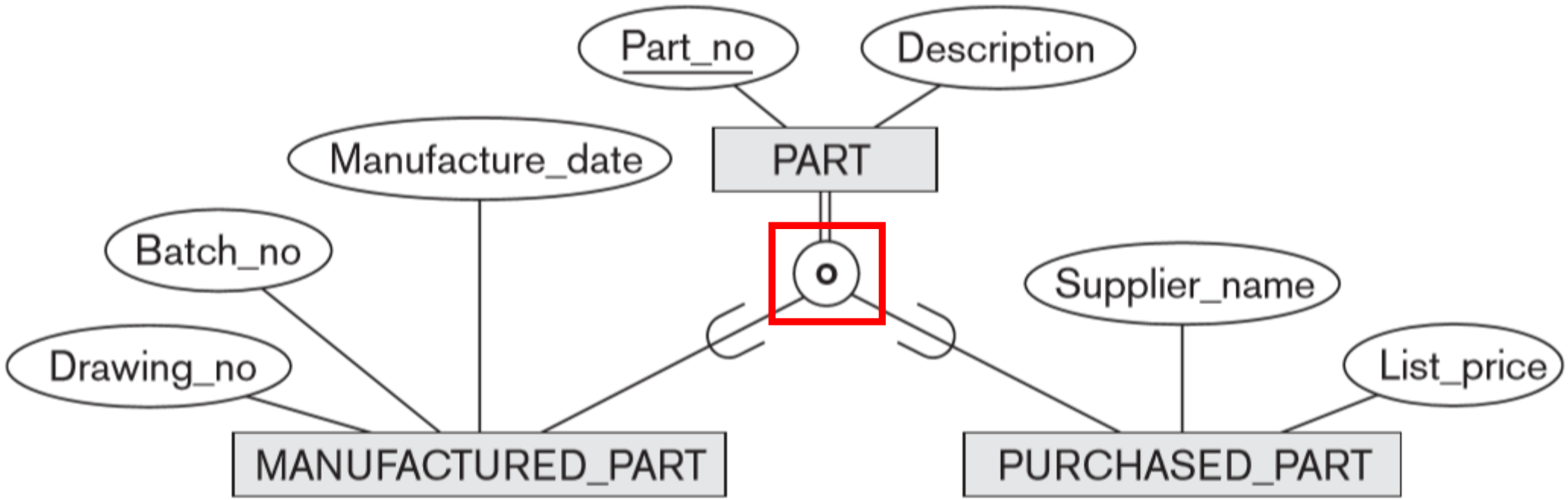
- If *all* subclasses in a specialization have their membership condition on the *same* attribute of the superclass, the specialization itself is called an **attribute-defined specialization**, and the attribute is called the **defining attribute** of the specialization.
- In this case, all the entities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


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
Job_type
EMPLOYEE
Job_type
d
'Secretary'
'Engineer'
'Technician'
Typing_speed
Tgrade
Eng_type
SECRETARY
TECHNICIAN
ENGINEER

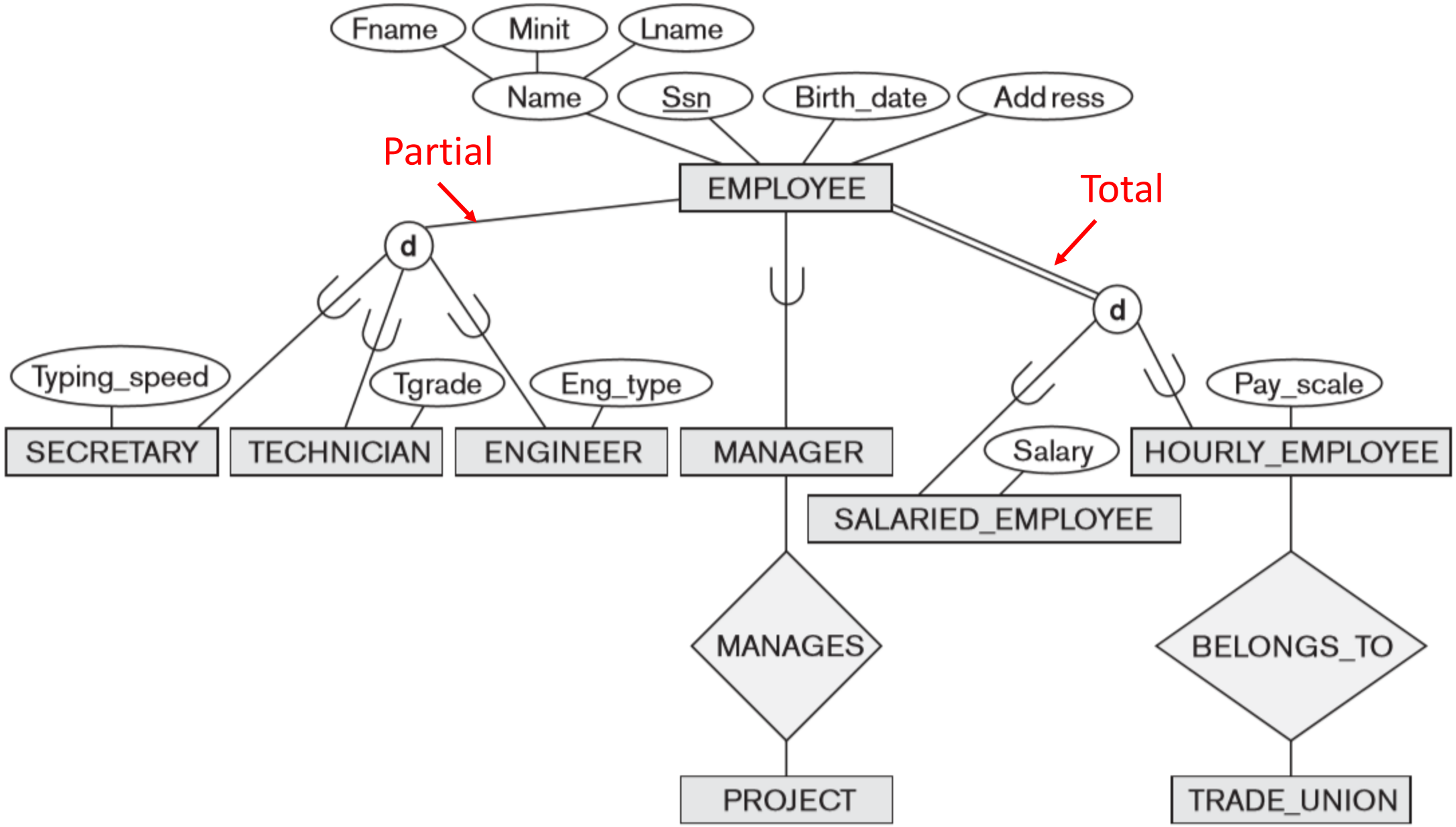
- When we do not have a condition for determining membership in a subclass, the subclass is called **user-defined**.
- Membership in such a subclass is determined by the database users when they apply the operation to add an entity to the subclass; hence, membership is *specified individually for each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

- Two other constraints may apply to a specialization.
- The first is the **disjointness** (or **disjointedness**) **constraint**, which specifies that the subclasses of the specialization must be disjoint.
- This means that an entity can be a member of *at most* one of the subclasses of the specialization.
- 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.


Fname
Minit
Lname
Name
Ssn
Birth_date
Address
Job_type
EMPLOYEE
Job_type
d
'Secretary'
'Engineer'
'Technician'
Typing_speed
Tgrade
Eng_type
SECRETARY
TECHNICIAN
ENGINEER


Part_no
Description
Manufacture_date
PART
Batch_no
o
Supplier_name
Drawing_no
List_price
MANUFACTURED_PART
PURCHASED_PART

- The second constraint on specialization is called the **completeness** (or **totalness**) **constraint**, which may be total or partial.
- A **total specialization** constraint specifies that *every* entity in the superclass must be a member of at least one subclass in the specialization.
- A **partial specialization** allows an entity not to belong to any of the subclasses.

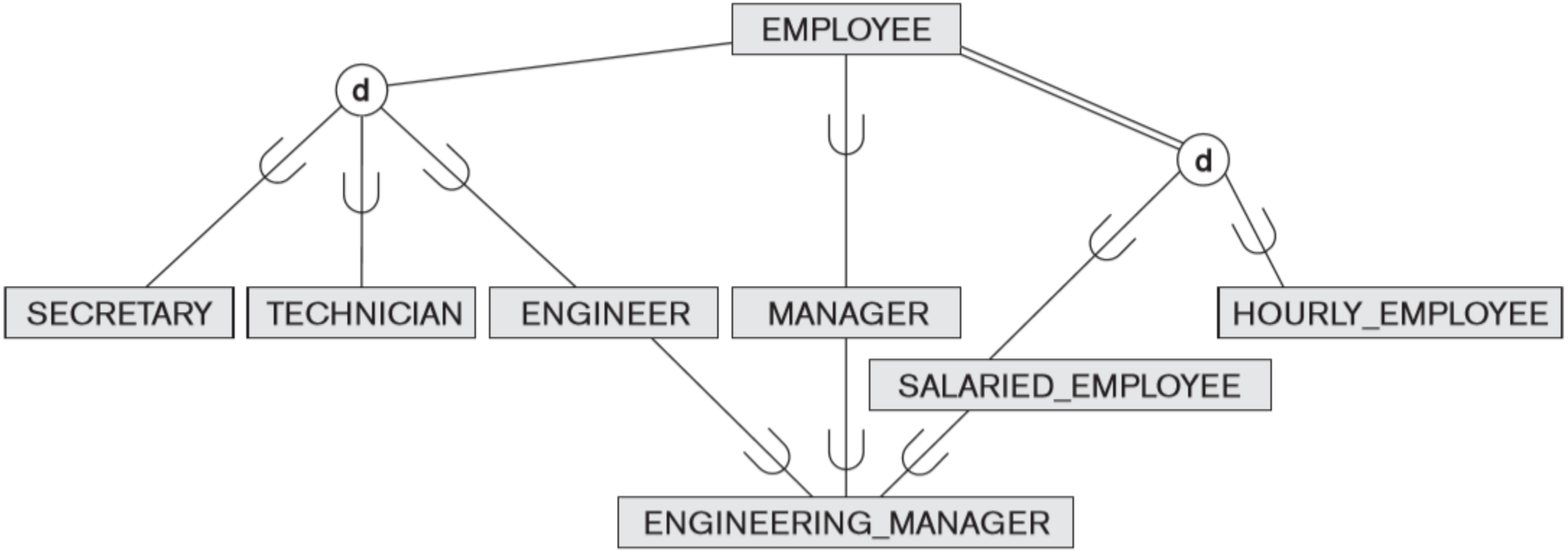

Fname
Minit
Lname
Name
Ssn
Birth_date
Address
Partial
EMPLOYEE
Total
d
d
Typing_speed
Tgrade
Eng_type
Pay_scale
SECRETARY
TECHNICIAN
ENGINEER
MANAGER
Salary
HOURLY_EMPLOYEE
SALARIED_EMPLOYEE
MANAGES
BELONGS_TO
PROJECT
TRADE_UNION

- Notice that the disjointness and completeness constraints are *independent*.
- Hence, we have the following four possible constraints on specialization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial

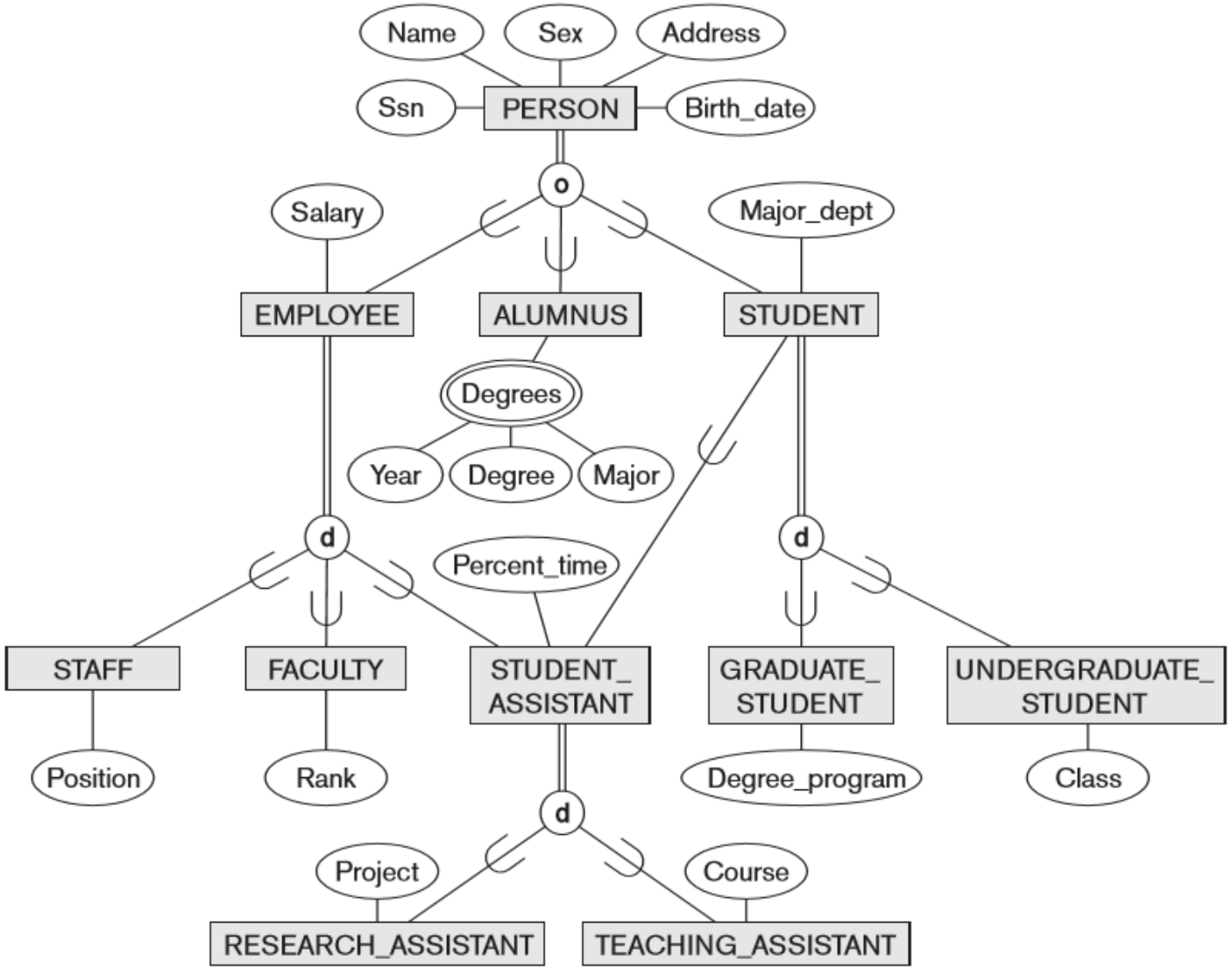
Specialization and Generalization Hierarchies and Lattices

- 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.

- A specialization lattice with shared subclass ENGINEERING_MANAGER



- A specialization lattice with multiple inheritance for a UNIVERSITY database



- In such a specialization lattice or hierarchy, a subclass inherits the attributes not only of its direct superclass, but also of all its predecessor superclasses *all the way to the root* of the hierarchy or lattice if necessary.
- For example, an entity in `GRADUATE_STUDENT` inherits all the attributes of that entity as a `STUDENT` and as a `PERSON`.
- Notice that an entity may exist in several *leaf nodes* of the hierarchy, where a **leaf node** is a class that has *no subclasses of its own*.
- For example, a member of `GRADUATE_STUDENT` may also be a member of `RESEARCH_ASSISTANT`.

- A subclass with *more than one* superclass is called a **shared subclass**.
- This leads to the concept known as **multiple inheritance**.
- An important rule related to multiple inheritance can be illustrated by the example of the shared subclass `STUDENT_ASSISTANT`, which inherits attributes from both `EMPLOYEE` and `STUDENT`.
- Here, both `EMPLOYEE` and `STUDENT` inherit *the same attributes* from `PERSON`.
- The rule states that if an attribute (or relationship) originating in the *same superclass* (`PERSON`) is inherited more than once via different paths (`EMPLOYEE` and `STUDENT`) in the lattice, then it should be included only once in the shared subclass (`STUDENT_ASSISTANT`).
- Hence, the attributes of `PERSON` are inherited only once in the `STUDENT_ASSISTANT` subclass.

- Although we have used specialization to illustrate our discussion, similar concepts *apply equally* to generalization.
- Hence, we can also speak of **generalization hierarchies** and **generalization lattices**.

Utilizing Specialization and Generalization in Refining Conceptual Schemas

- In the specialization process, we typically start with an entity type and then define subclasses of the entity type by successive specialization; that is, we repeatedly define more specific groupings of the entity type.
- This successive specialization corresponds to a **top-down conceptual refinement process** during conceptual schema design.

- It is possible to arrive at the same hierarchy or lattice from the other direction.
- In such a case, the process involves generalization rather than specialization and corresponds to a **bottom-up conceptual synthesis**.

- In structural terms, hierarchies or lattices resulting from either process may be identical; the only difference relates to the manner or order in which the schema superclasses and subclasses were created during the design process.
- In practice, it is likely that neither the generalization process nor the specialization process is followed strictly, but that a combination of the two processes is employed.
- New classes are continually incorporated into a hierarchy or lattice as they become apparent to users and designers.