H5

The University of Manchester

Designing Databases: The Enhanced Entity-Relationship Approach

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 2014-2015 Lecture 05 of 12]

Acknowledgements

- These slides are adaptations (mostly minor, but some major) of material authored and made available to instructors by **Ramez Elmasri and Shamkant B. Navathe** to accompany their textbook **Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8**

- All errors are my responsibility.

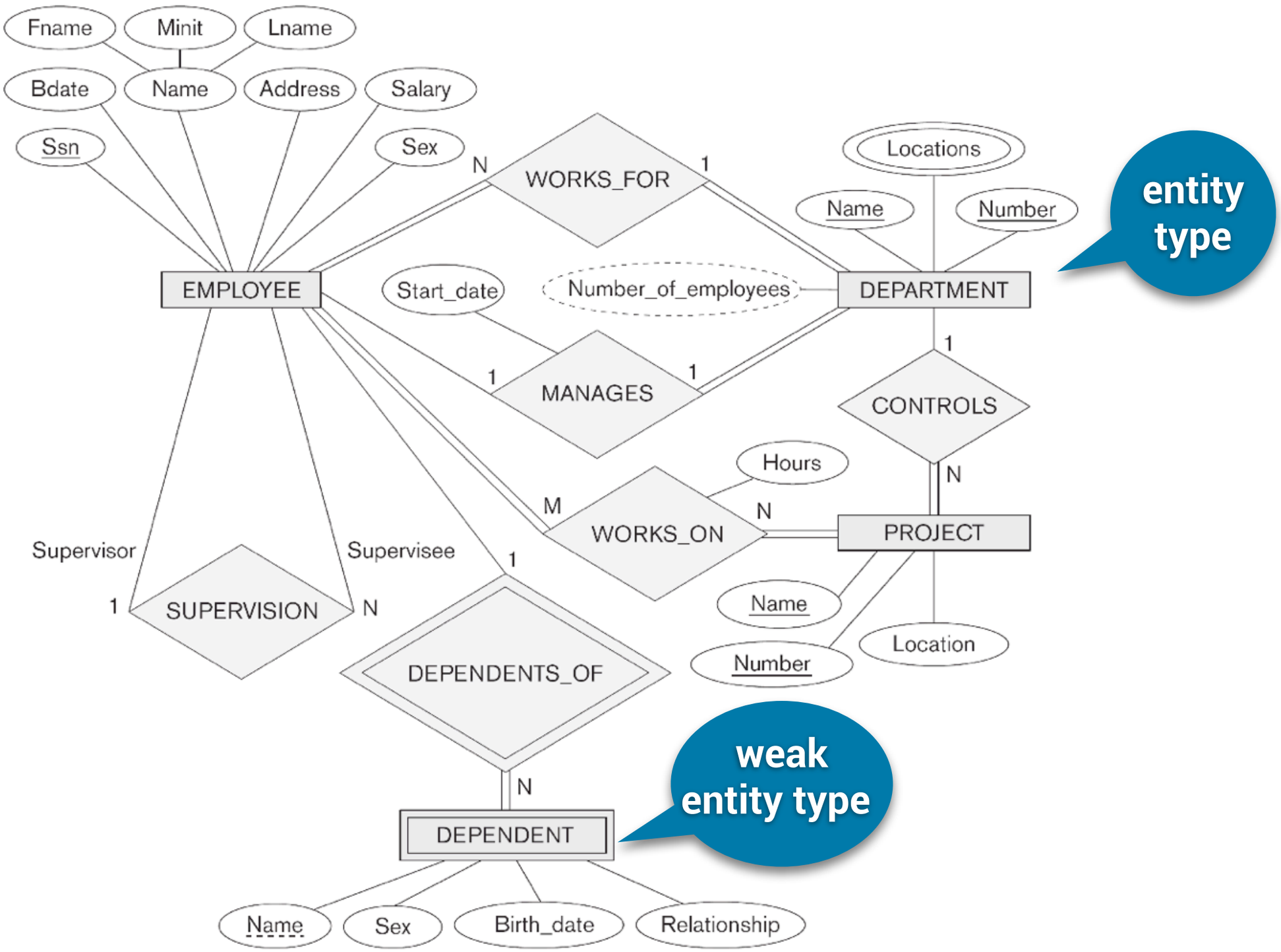
In Previous Lectures

- We learned that data is an enterprise asset and that DBMSs are crucial to manage it well.
- We learned the importance of adopting different levels of abstraction in designing and implementing databases.
- We learned how data models lead to a distinction between schemas and instances that enables a logical view of the data.
- We learned about the relational approach to logical data modelling.
- We learned about the relational algebra and SQL, both its DDL and DML capabilities and its querying constructs.
- We learned of the practical benefits of performing conceptual modelling before logical design and why the ER approach serves well this purpose.

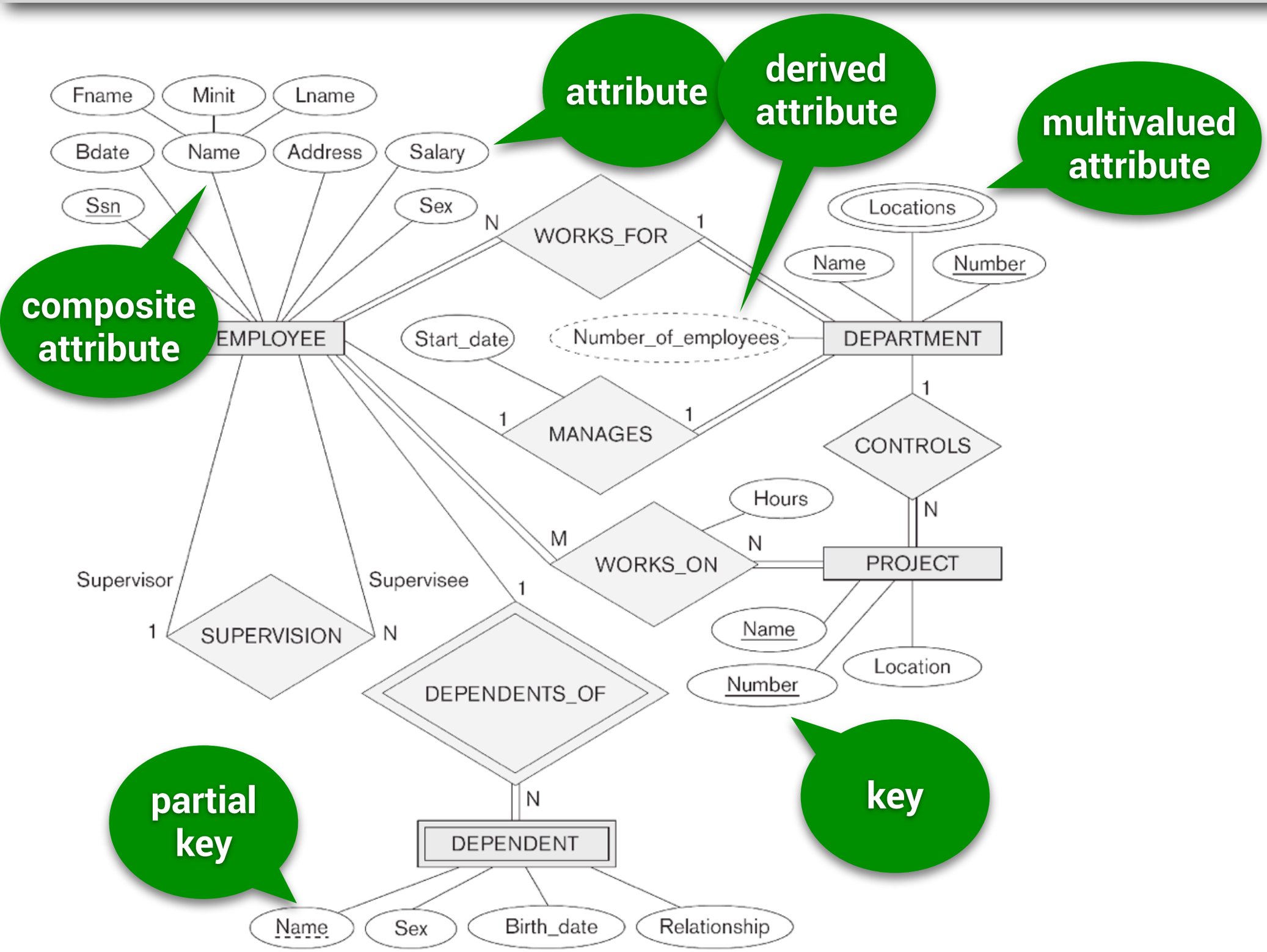
In This Lecture

- What is the enhanced entity-relationship model (EER)?
- What are specialization/ generalization (S/G) processes in EER modelling?
- What is EER-inheritance and what does it apply to?
- What constraints apply to S/G relationships?
- How do S/G hierarchies and lattices arise?
- What are union types?
- What design guidelines are useful in EER modelling?

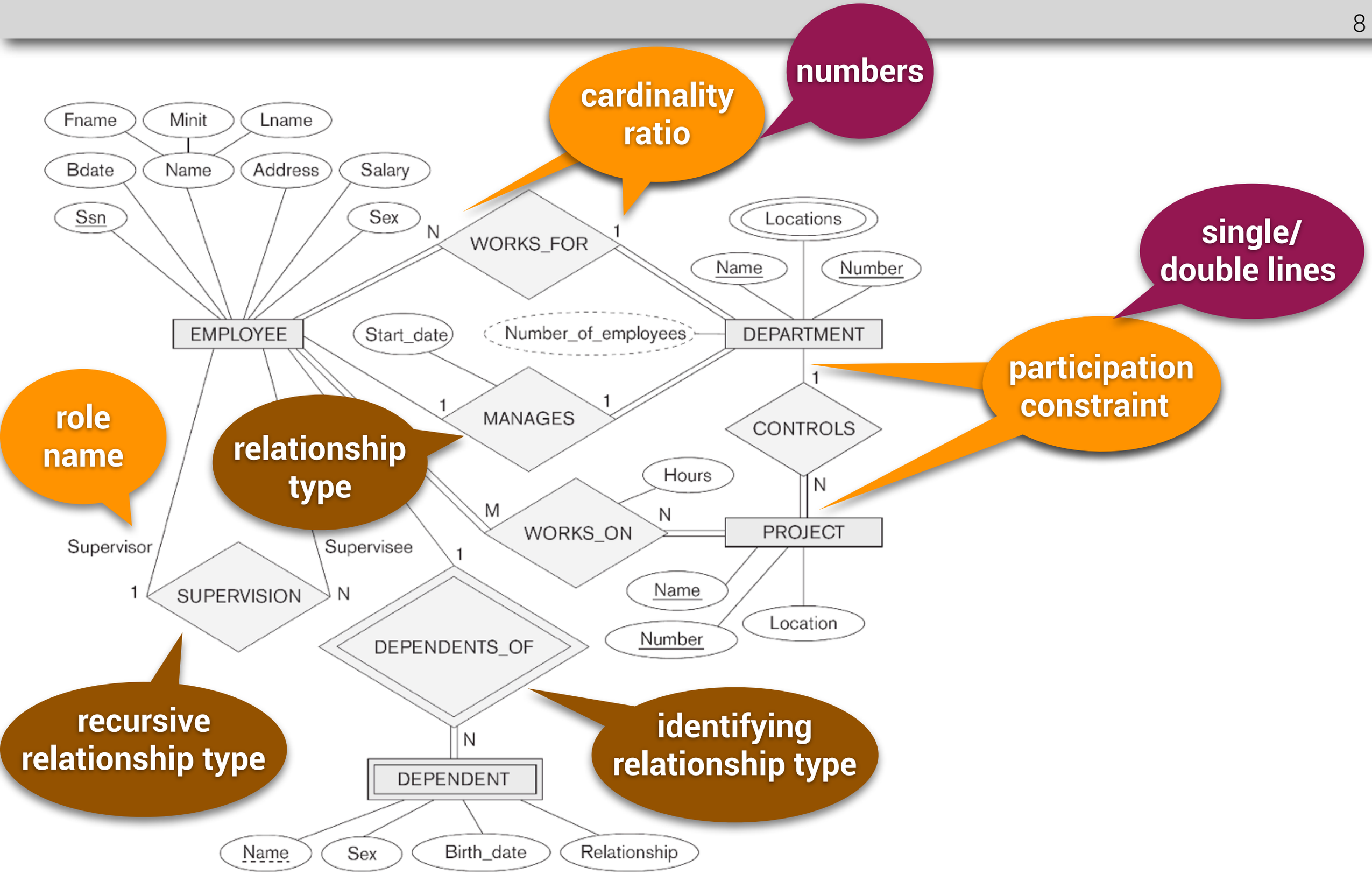
Review: ER Constructs and Notation



Review: ER Constructs and Notation



Review: ER Constructs and Notation



The Enhanced Entity-Relationship (EER) Model

- The EER model includes all the modelling constructs of the ER model and:
  - It is more expressive, and can lead to the design of database schemas that more accurately represent the application domain
  - It reflects properties and constraints of the data more precisely
  - It can capture more complex requirements than traditional transactional applications

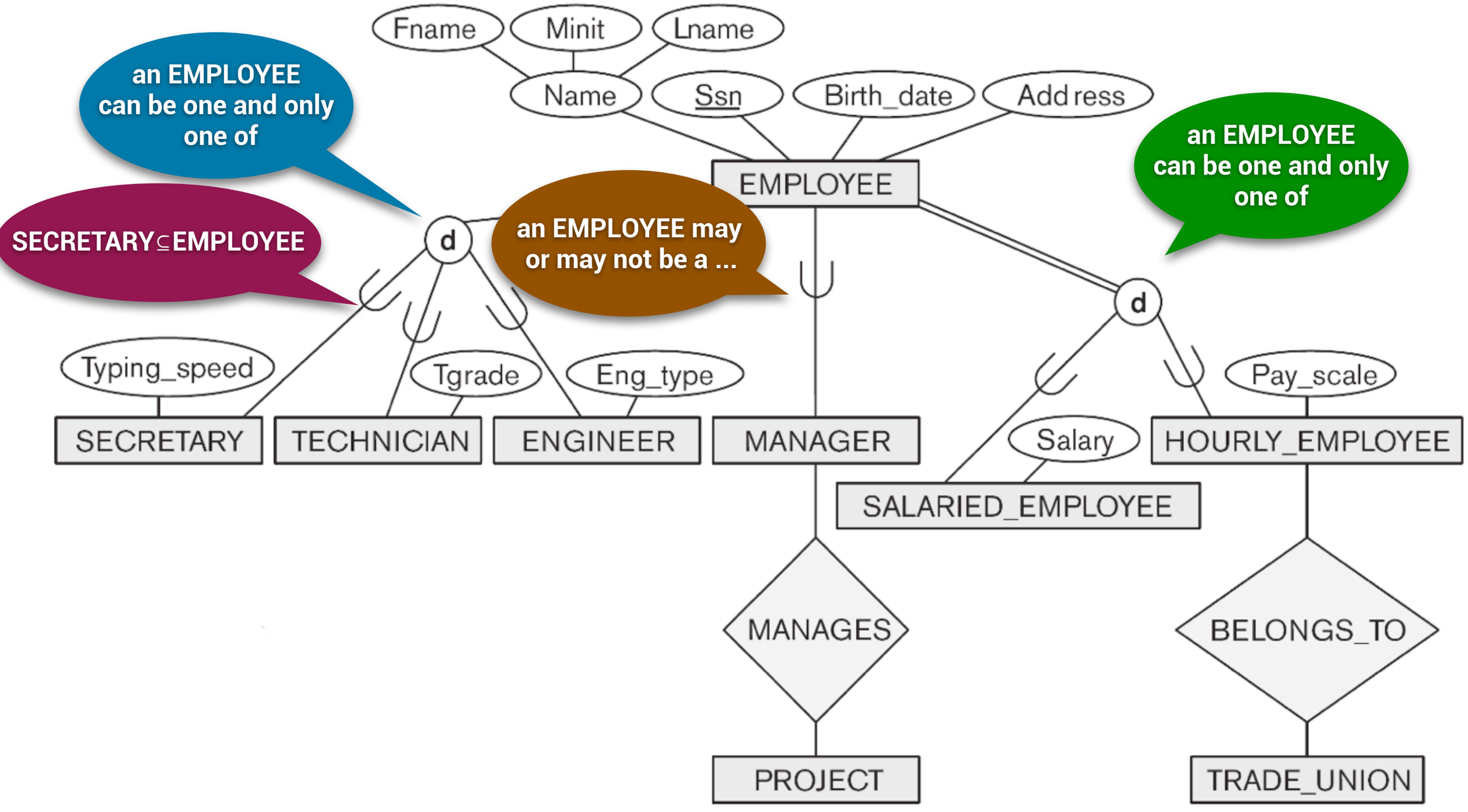
The Enhanced Entity-Relationship (EER) Model

- There are two aspects of application domains (i.e., of miniworlds) that the EER model is capable of capturing:
  - Subtypes/classes and their supertypes/classes
  - Category (or union) types
- In doing so, the EER model captures attribute and relationship inheritance from the super- to the subtype.

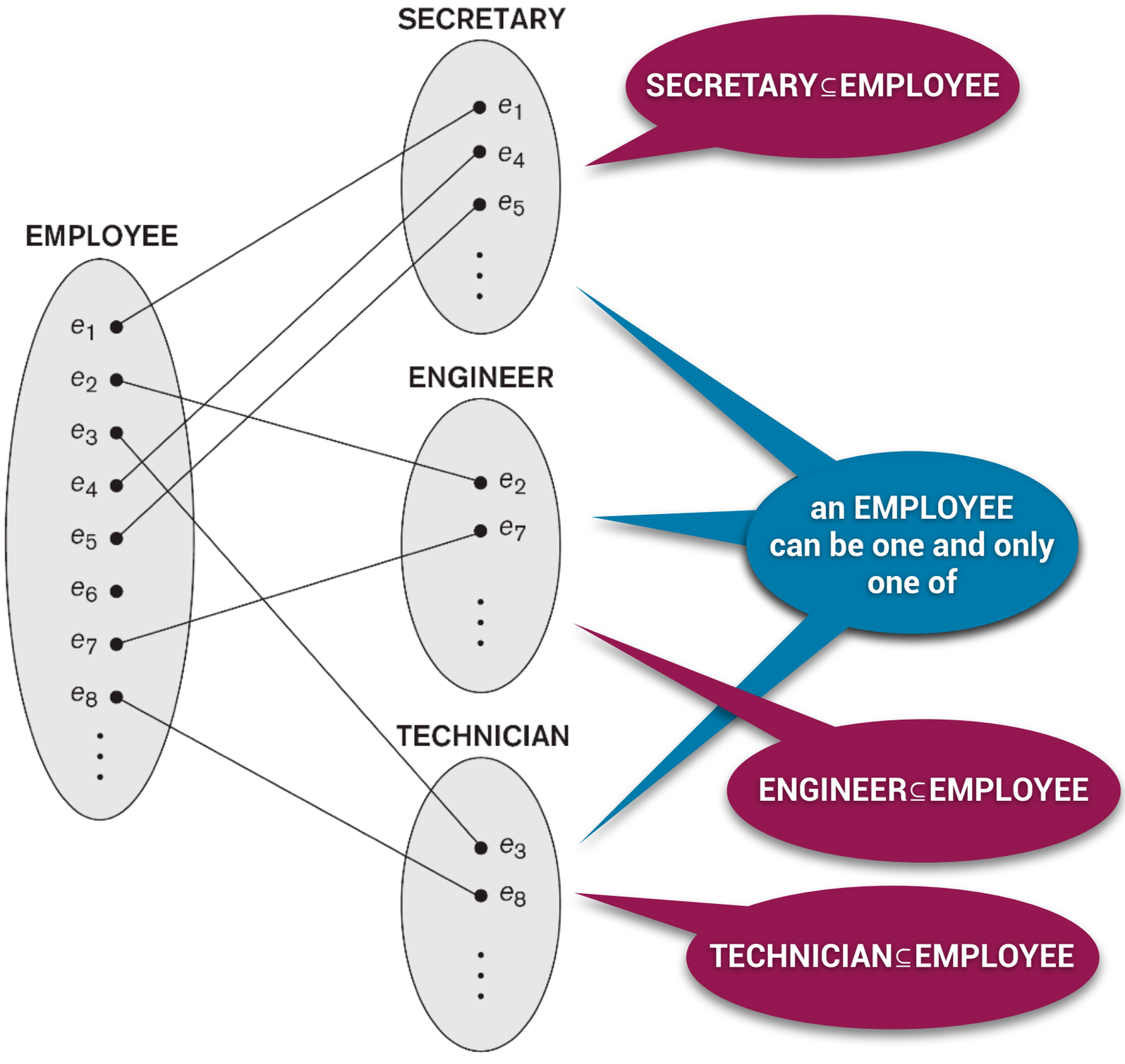
The Enhanced Entity-Relationship (EER) Model

- Compared with ER modelling, in EER modelling, one is also concerned with:
  - specializing or generalizing entity types
  - deciding whether sub-entity types are union (or category) entity types

EER: Example Specializations



EER: Example Instances of a Specialization



EER: Specialization v. Generalization

- **Specialization** is the process of identifying that some subclasses of preexisting entity types are of specific interest

- The new entity types in this case are called the **subclasses** (or sub-entity type, or subtypes)

- We specialize because we only identify a specific interest on the **subclasses later** in the design process than we identified a specific interest on the preexisting entity types they are subclasses of

EER: Specialization v. Generalization

- Generalization is the *reverse* (in the directional sense) of specialization

- **Generalization** is the process of identifying that some superclasses of preexisting entity types are of specific interest

- The new entity types in this case are called **superclasses** (or super-entity types, or supertypes)

- We generalize because we only identify a specific interest on the **superclasses later** in the design process than we identified a specific interest on the preexisting entity types they are superclasses of

EER: Subclass Characterization

- Subclasses are defined on the basis of some distinguishing characteristic of the entities in the superclass

- A subclass can, in addition, be characterized by:
  - Specific attributes
  - Specific relationship types

EER: Attribute/Relationship-Driven Identification

- There may be many or just one subclass

- The subset symbol (⊂) is used to denote the direction of the relationship

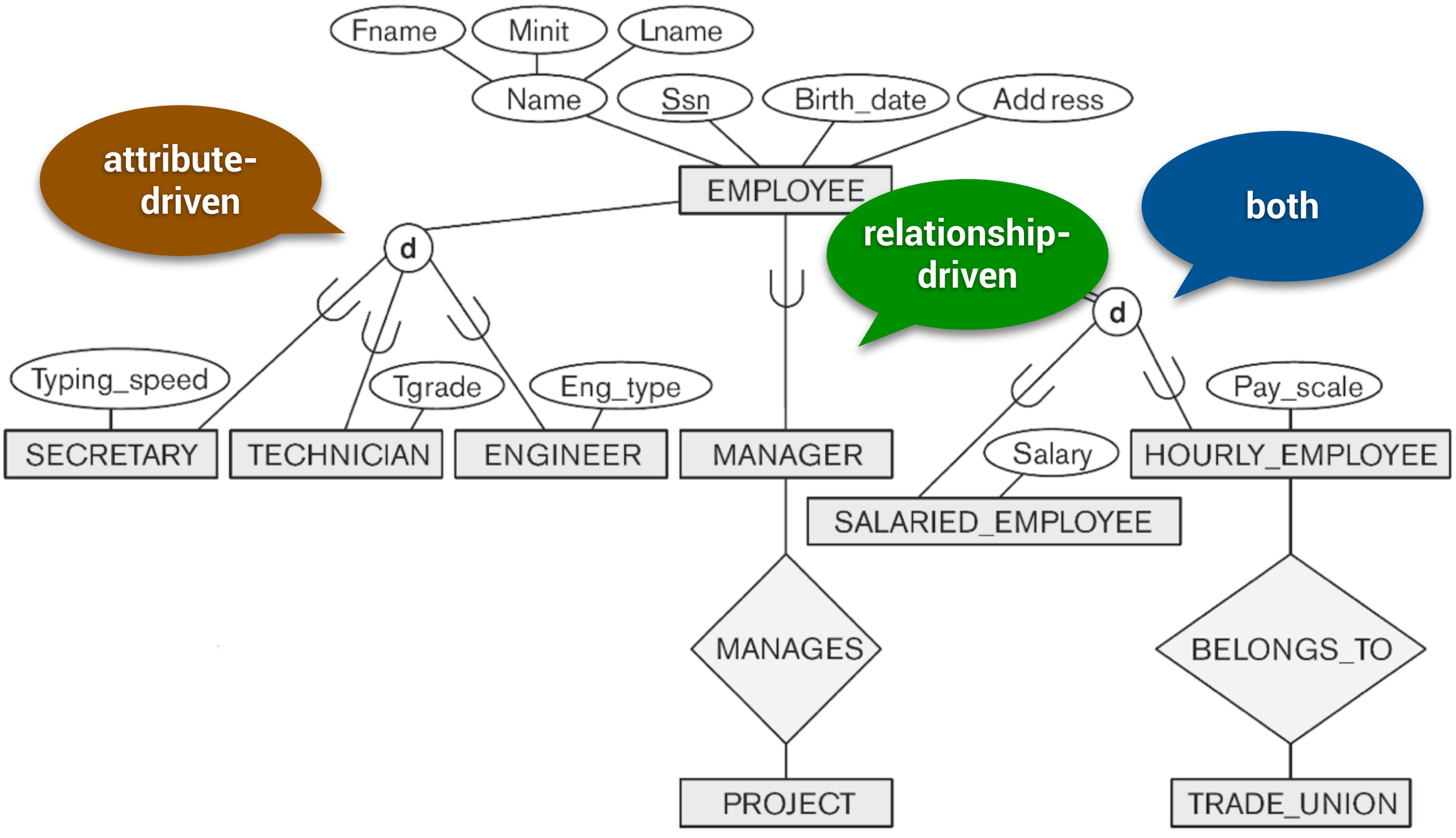
- We often read the relationship as **is-a** from the subclass to the superclass.

- For example, STUDENT is-a PERSON and EMPLOYEE is-a PERSON and so on.

EER: Attribute/Relationship-Driven Identification

- Certain attributes may apply to some but not all entities of the superclass
- Some relationship types may have the participation of members of a subclass only

EER: Attribute/Relationship-Driven Identification



EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

Assuming that both salary or residency or both are stored, then the following subtypes are **attribute-defined**:

Lower_Tax_Employee ⊂ Employee

≡

{e in Employee |
e.salary < 50K and
e.residency = 'UK'}

Upper_Tax_Employee ⊂ Employee

≡

{e in Employee |
not e in Lower_Tax_Employee}

EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

Assuming that residency is stored but EU membership is a Boolean-valued function, then the following subtypes are **predicate-defined**:

Home_Student ⊂ Student ≡
{s in Student |
s.residency = country and
country in EU}

Overseas_Student ⊂ Student ≡
{s in Student |
not s in Home_Student}

EER: Subclass Characterization

- An entity subtype may be:
  - **Attribute-defined**
  - **Predicate- (or condition-) defined**
  - **User-defined**

Assuming that prizeAwarded is a user input (as opposed to a programmed function), then the following subtype is **user-defined**:

PrizeRecipient ⊂ Student ≡
{s in Student |
prizeAwarded ←}

EER: Constraints on Subclass Relationships

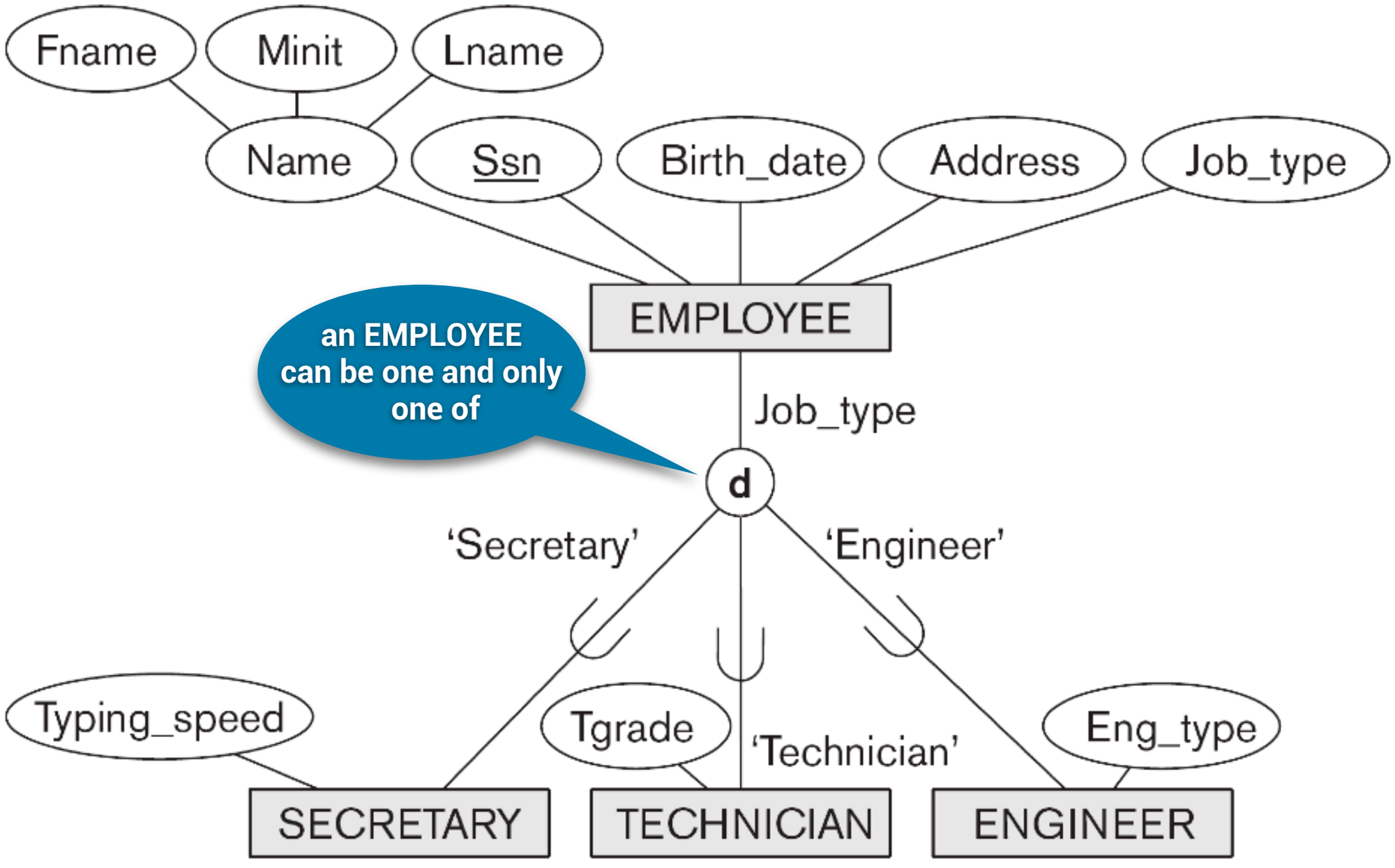
- A **disjointness** constraint specifies that the subclasses must be **disjoint**, i.e., no entity in a subclass can belong to any other subclass

- In an EER diagram, this is marked with a '**d**' inside the circle that relates the subclasses to the superclass

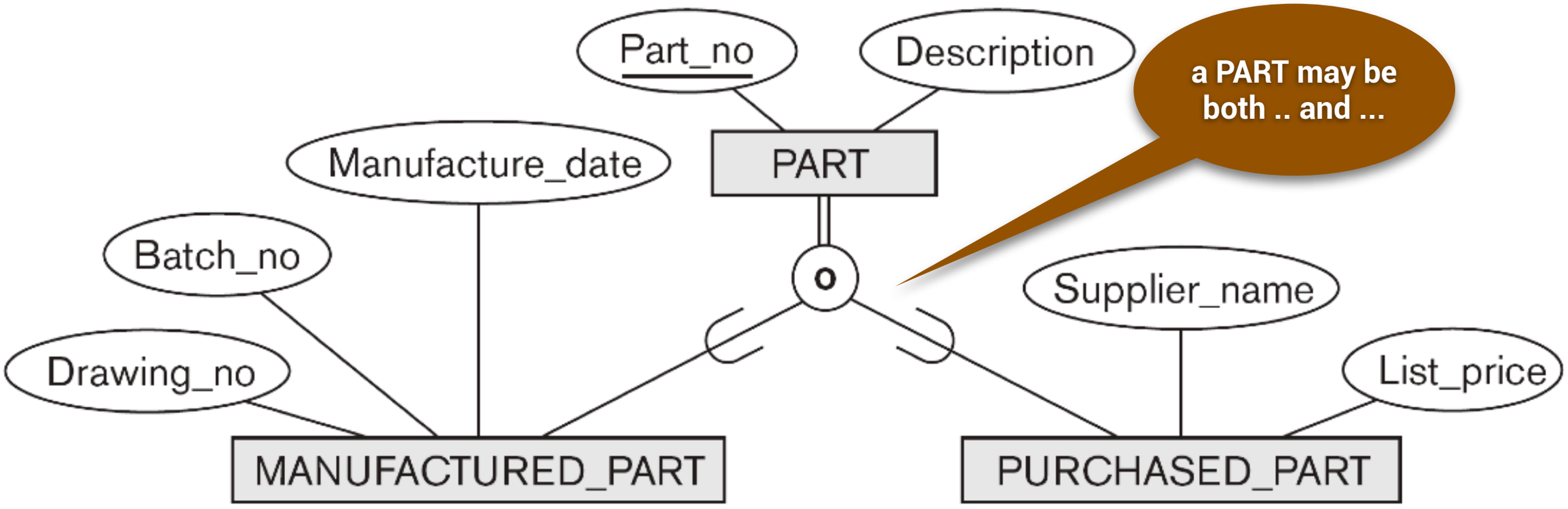
- If a disjointness constraints does not hold, i.e., if one entity can belong to more than one subclass, the subclasses are said to be **overlapping**

- In an EER diagram, this is marked with a '**o**' inside the circle that relates the subclasses to the superclass

EER: Constraints on Subclass Relationships



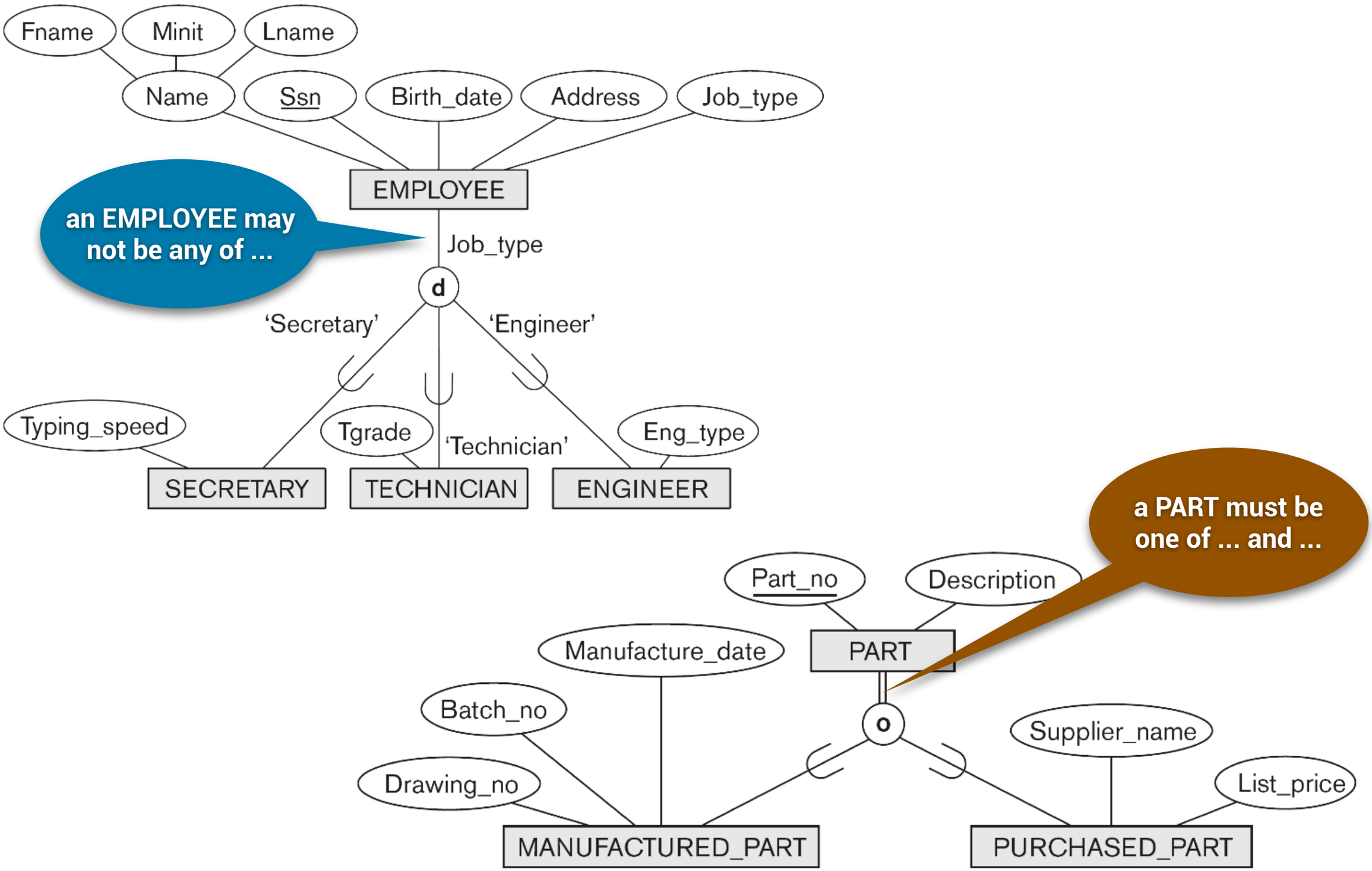
EER: Constraints on Subclass Relationships



EER: Constraints on Subclass Relationships

- A **completeness** constraint specified whether
  - **every** entity in the superclass **must** belong to at least one of the subclasses, in which case it is said to be **total** (or **covering**), marked with a double line from the superclass to the circle in EER diagrams or
  - **some** entity in the superclass **may** belong to none of the subclasses, in which case it is said to be **partial**, marked with a single line in EER diagrams
- Disjointness and completeness constraints are independent

EER: Constraints on Subclass Relationships

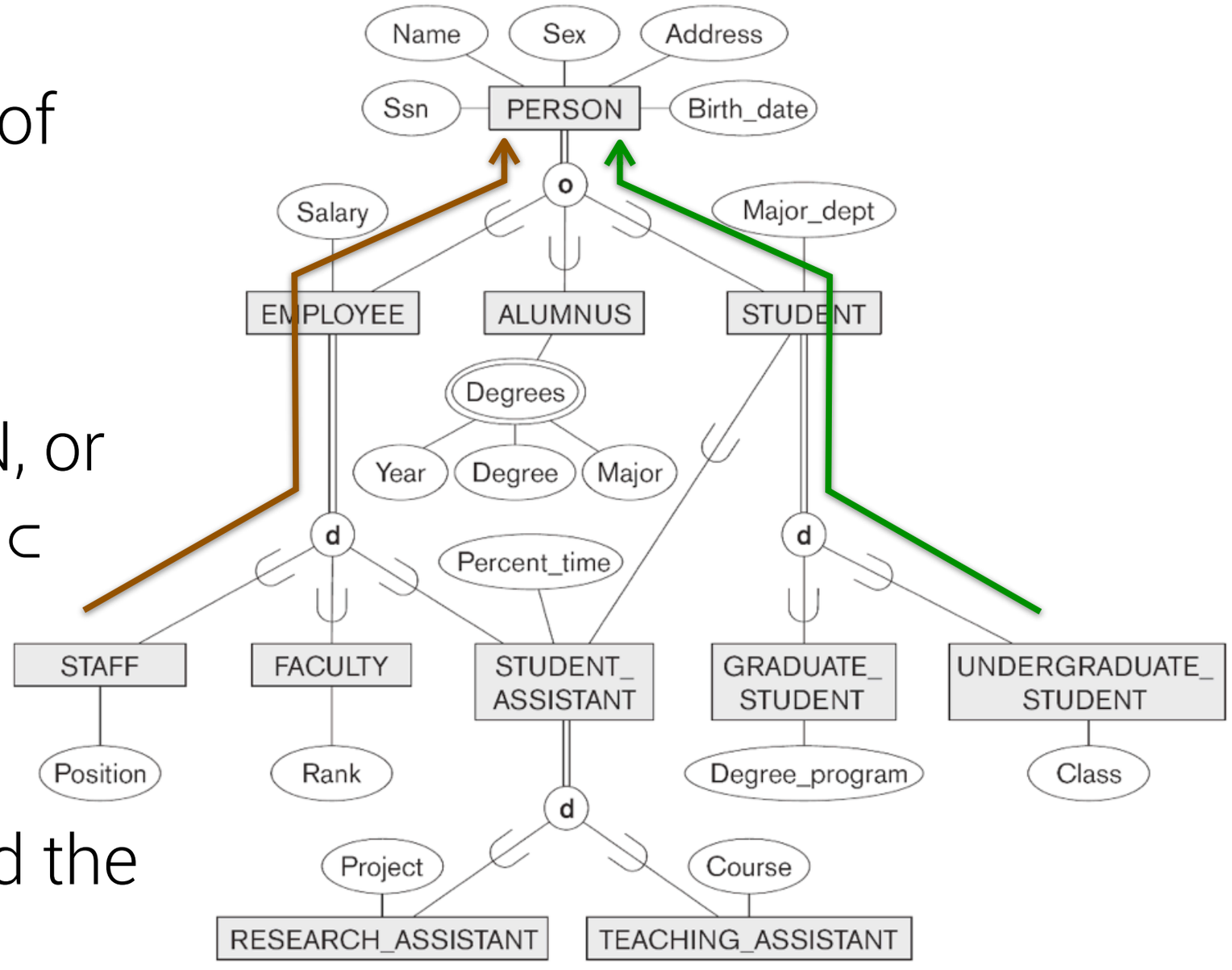


EER: Inheritance Trees and Lattices

- Specialization/ generalization relationships can be of indefinite length

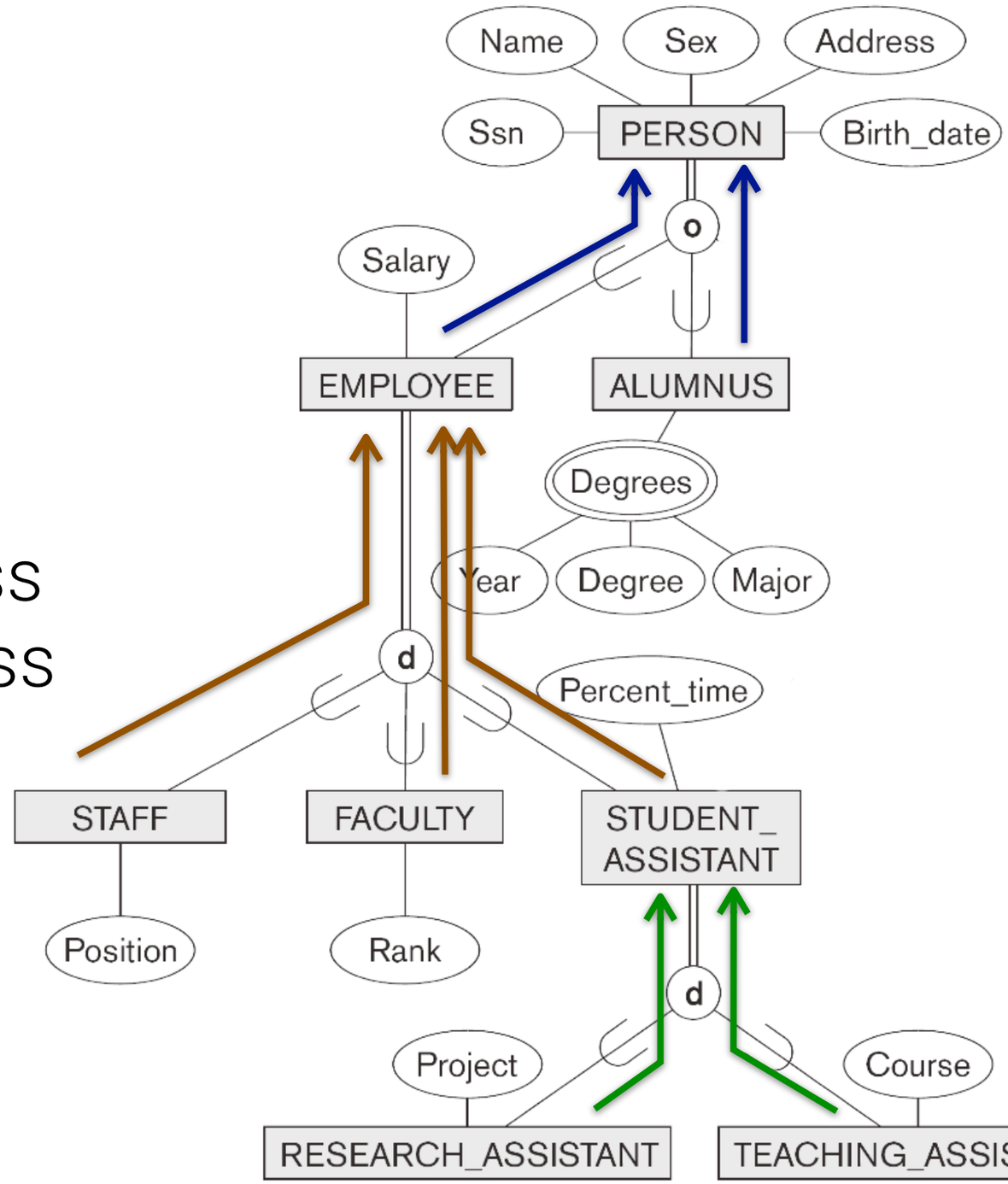
- For example, UG ⊂ STUDENT ⊂ PERSON, or STAFF ⊂ EMPLOYEE ⊂ PERSON

- These chains form inheritance paths and the paths characterize structures.



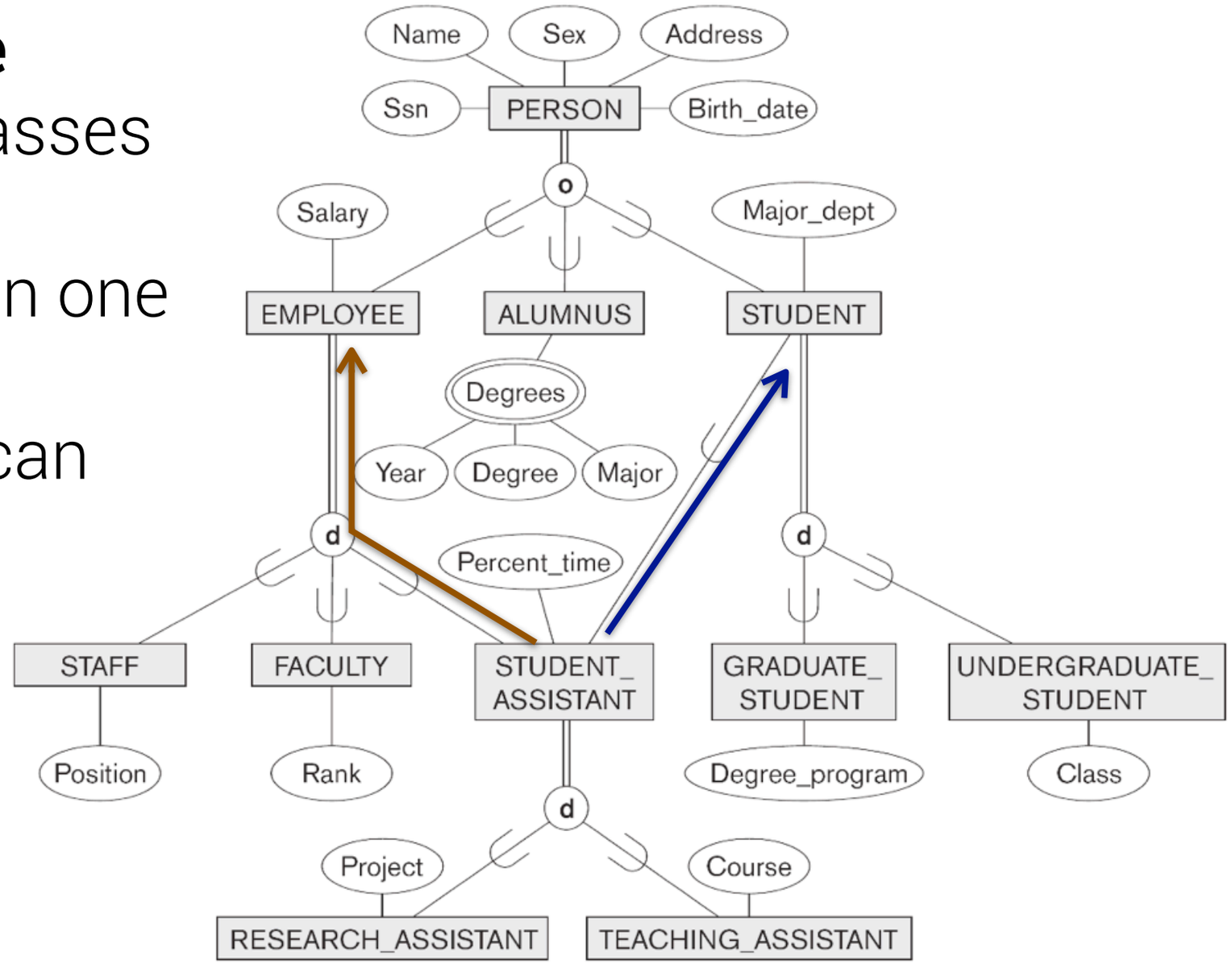
EER: Inheritance Trees and Lattices

- A specialization/ generalization **tree** (or **strict hierarchy**) arises if **every** subclass participates as a subclass in only one class/subclass relationship, i.e., has **exactly one superclass**.



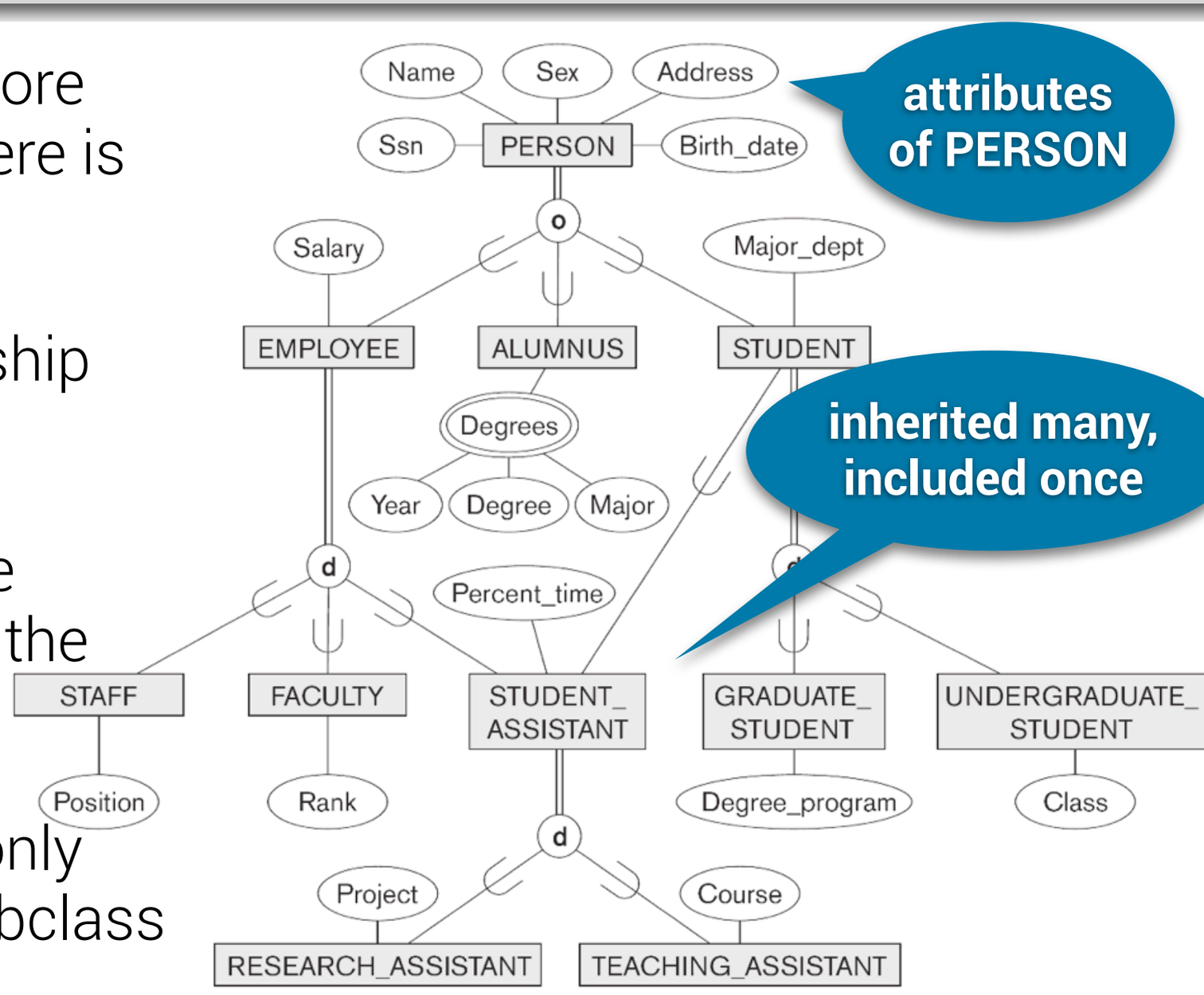
EER: Inheritance Trees and Lattices

- A specialization/ generalization **lattice** arises if **some** subclasses can participate as a subclass in more than one class/subclass relationship, i.e., if it can have **more than one superclass**.



EER: Inheritance Trees and Lattices

- When a subclass has more than one superclass, there is multiple inheritance
- An attribute (or relationship type) originating in a superclass is inherited through each path in the lattice that originates in the common subclass.
- However, it is included only once in the common subclass
- Most models and languages disallow multiple inheritance



EER: Union Types

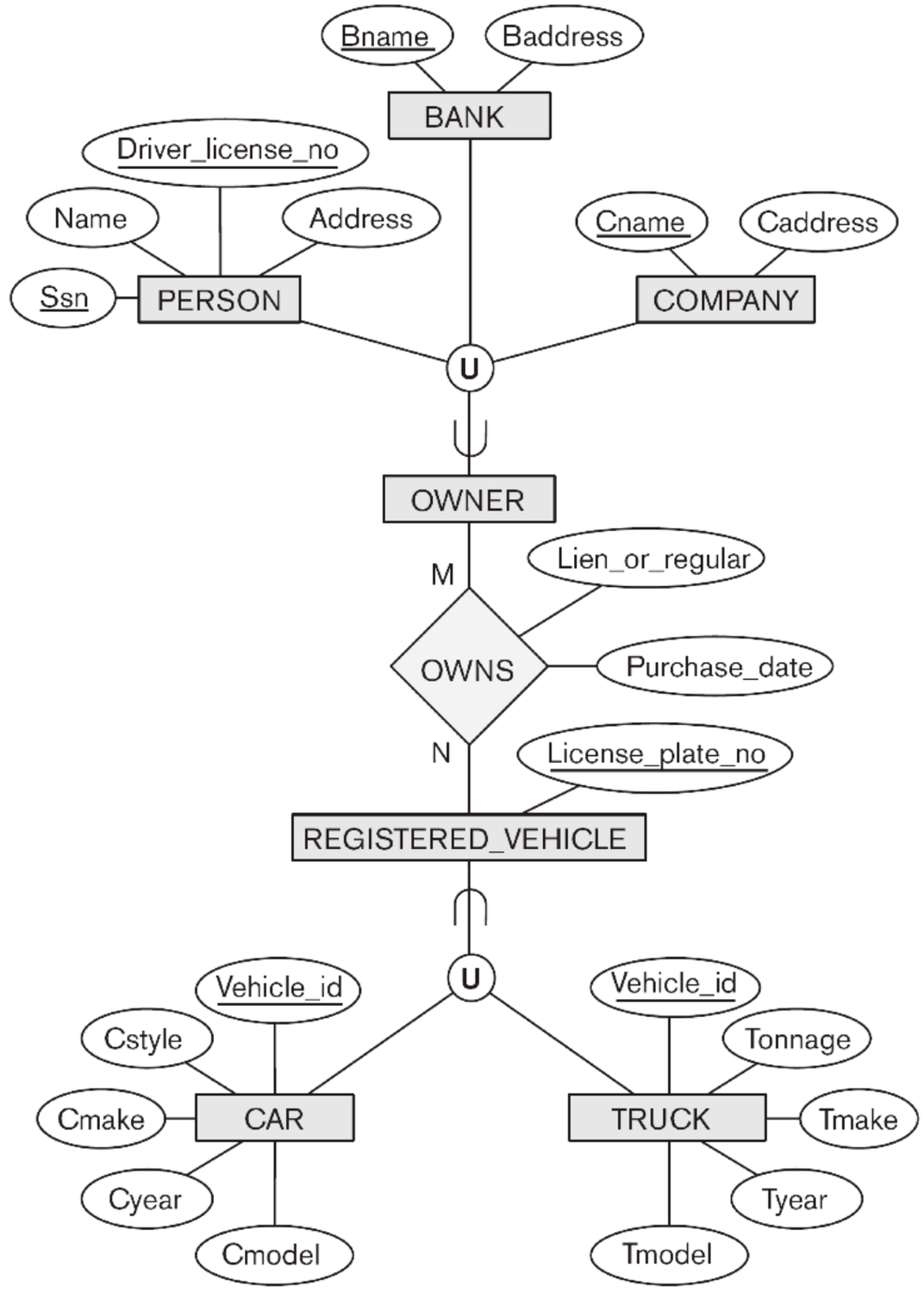
- A union (or category) type arises when several distinct, structurally dissimilar entity types contribute entities to a single subclass

- The common subclass represents a collection of objects that is a subset of the UNION of several, distinct entity types

EER: Union Types

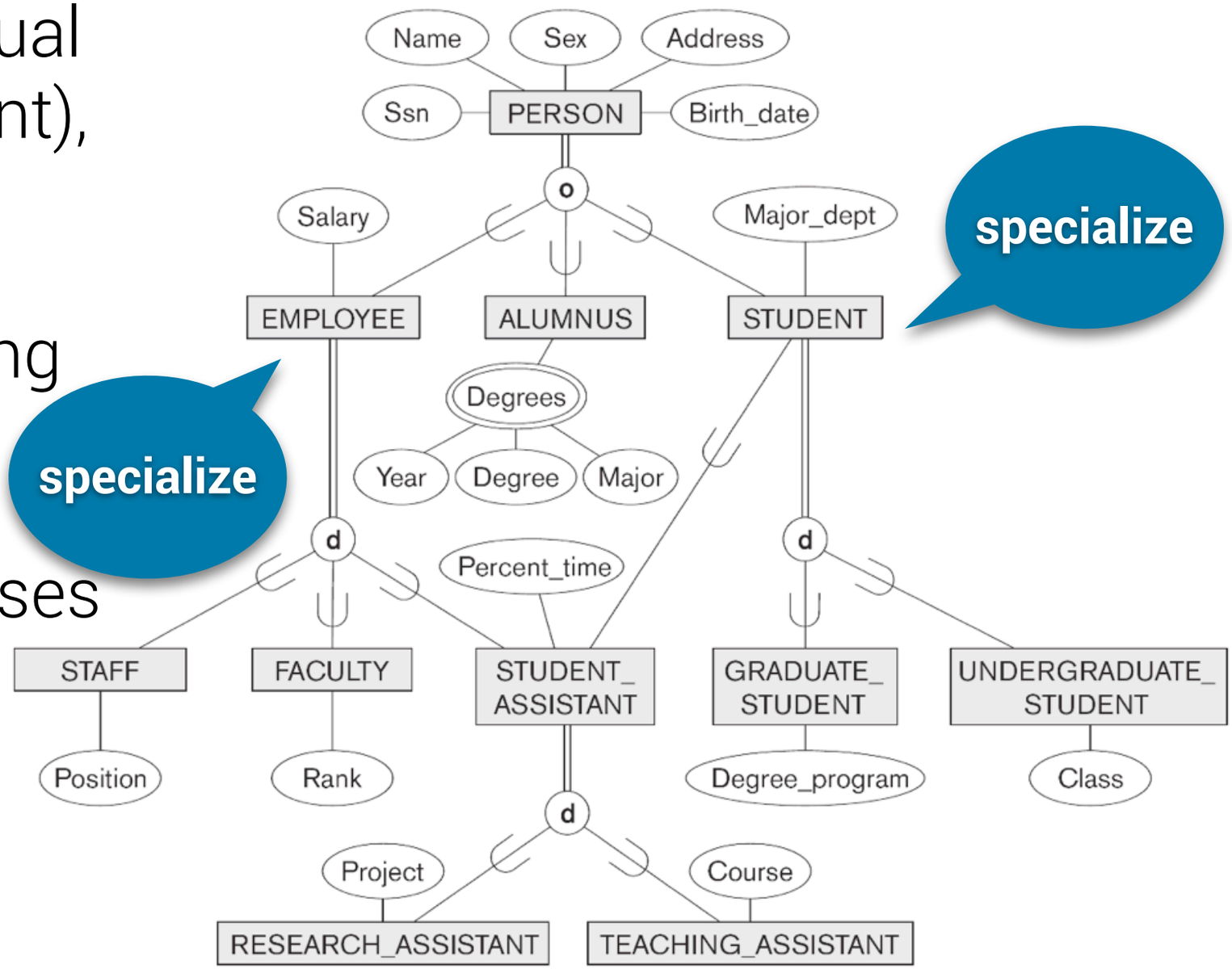
- For example, PERSON, BANK and COMPANY are quite distinct entity types (they have quite distinct attributes and/or relationships) but they can be generalized into a union type: OWNER, of REGISTERED VEHICLEs.

- REGISTERED VEHICLE itself is a UNION type of CAR, TRUCK, BOAT, etc.



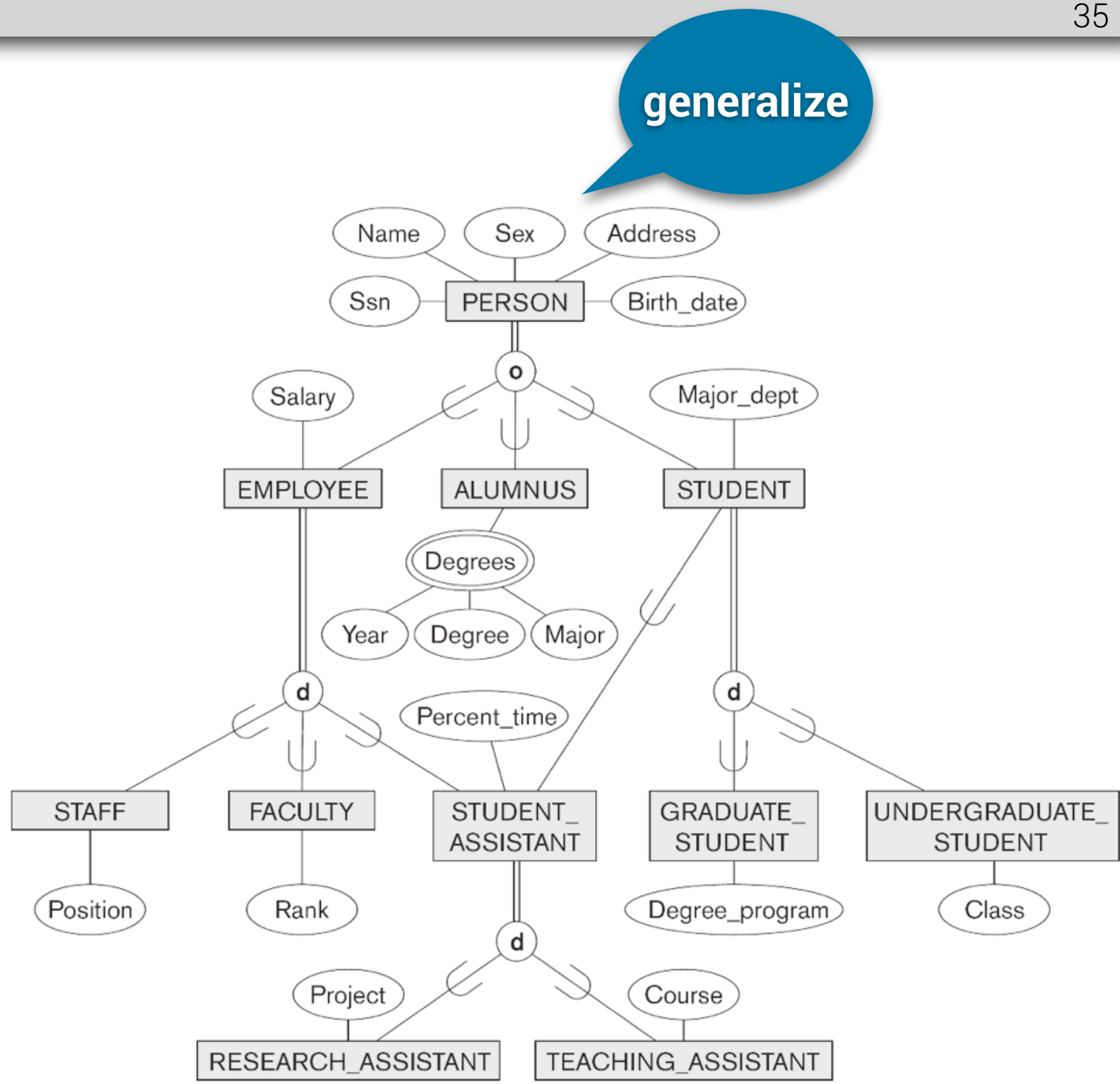
Refining ER Schemas with Specialization/Generalization

- In top-down conceptual analysis (or refinement), we:
  - start with an existing entity type
  - identify its subclasses of interests
  - then specialize



Refining ER Schemas with Specialization/Generalization

- In bottom-up conc
synthesis, we:
  - start with with a existing entity ty
  - identify their superclasses of
  - then generalize



Refining ER Schemas with Specialization/Generalization

- In practice, both are used together because often:
  - a clear concept merits specialization into subclasses of interest
  - a set of clear concepts merit generalization into a superclass of interest

Refining ER Schemas with Specialization/Generalization

- Many specializations and subclasses can be defined to make the conceptual model accurate

- If any subclass has few specific attributes and no specific relationships, it can be merged into the superclass

- In other words, the designer needs to consider whether it is worth the trouble to specialize

- We can use NULLs for those instance for which the specific attributes are not applicable

- We can use one or more specific attributes to characterize, for each entity, the subclass it belongs to

Refining ER Schemas with Specialization/

- Union types should generally be avoided as they tend to break typing discipline and make processing more complex and prone to errors

- Choice of disjoint/overlapping and total/partial constraints on specialization/generalization is driven by rules in miniworld being modeled: one needs to ask users for guidance.

- If no clear steer emerges, it is best to be permissive (i.e., allow overlapping and partiality).

Summary

The EER Approach to Database Design

- The EER approach retains all the constructs of the ER approach and, essentially, adds support for S/G and for union types.

- S/G processes in EER modelling allow the explicit definition of sub- and super-entity types of interest out of naturally identifiable ones.

- EER-inheritance denotes the consequence that a sub-entity type may have specific attributes and participate in specific relationship types but, in any case, inherits all the super-entity type's attributes and relationships.

- S/G relationships are constrained as to whether subclasses are disjoint or overlapping and whether the subclasses exhaust the superclass or not.

The EER Approach to Database Design

- S/G relationships can form chains and if such chains are linear they form hierarchies (no subclass has more than one superclass) or else they branch and form lattices (some subclasses may have more than one superclass).

- Union types are subclasses whose superclasses (called categories) are distinct, structurally dissimilar entity types.

- Useful design guidelines in EER modeling include switch between specialization and generalization depending on whether the superclass or the subclasses are more naturally occurring, avoiding excessive specialization/generalization, and avoiding poorly motivated union types.

In The Next Lecture

- We'll learn how a conceptual model can be mapped into a logical model that is capable of being implemented in a DBMS.