Lecture 3.2

Extended ER Modeling

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Introduction

- The original ER Model(1970) was able to handle the data modelling of the common business problems.
- In 1980s, there was a rapid increase in the development of many new database applications like **CAD(Computer Aided designing)**, **GIS(Geographical Information System)**, **Multimedia, Knowledge base for AI(Artificial Intelligence) database** and so on.
- The **basic ER-Modeling** concepts were no longer sufficient to represents the requirements of these newer and complex applications.
- The ER-Model that is supported with the additional semantic concepts is called **Extended ER Model(EER Model).**
- The EER model includes *all the modeling concepts of the ER model*.
- In addition, it includes the concepts of **subclass** and **superclass** and the related concepts of **specialization** and **generalization.**

Subclasses and Superclasses

- In some cases an Entity type has numerous additional subgrouping of its entities that are meaningful and need to be represented explicitly because of their significance to the database application.
- **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.
- 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.
- We call the relationship between a superclass and any one of its subclasses a **superclass/subclass** or **supertype/subtype** or simply **class/subclass relationship.**

Subclasses and Superclasses

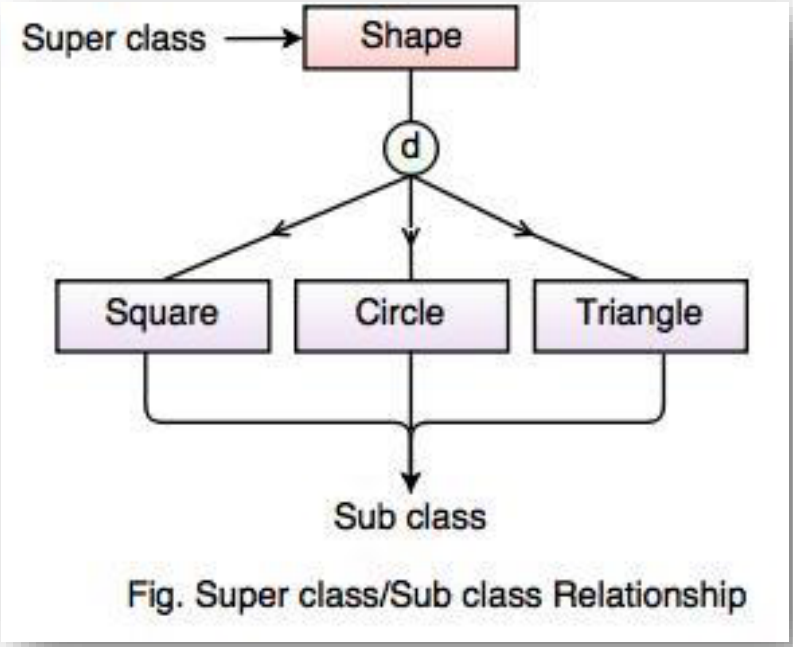
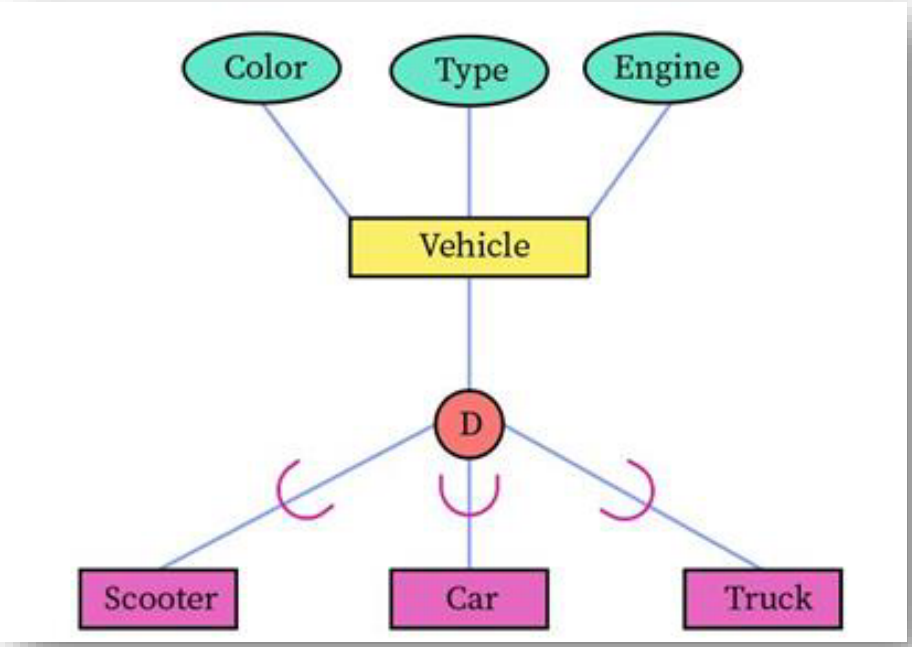
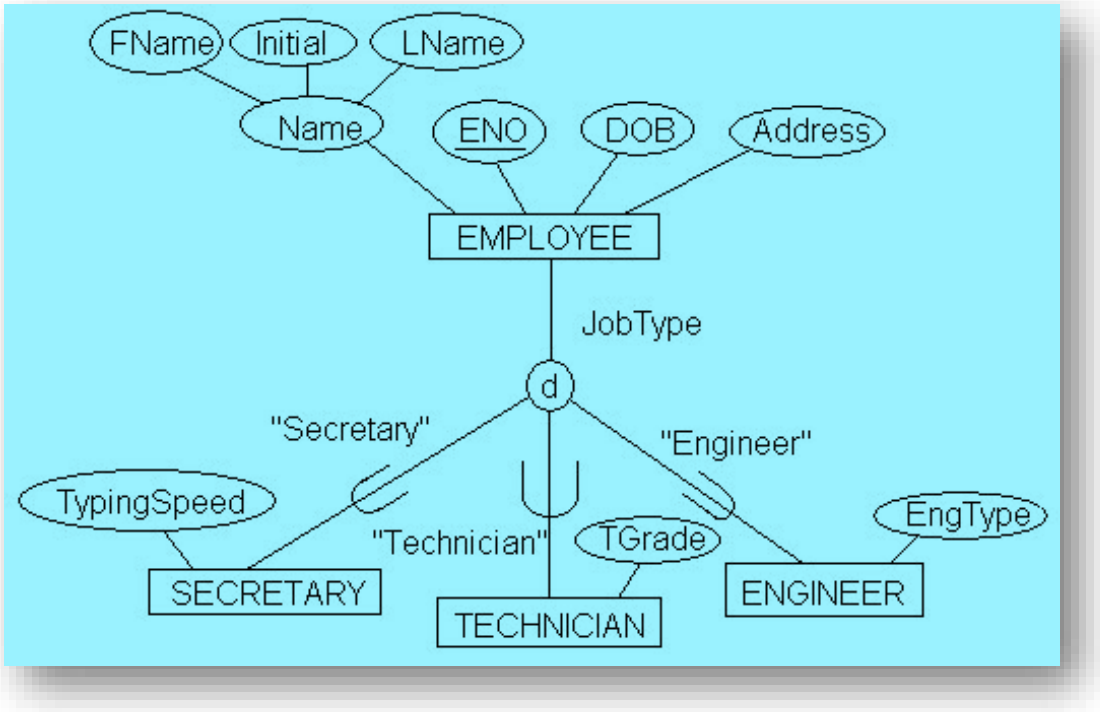


Fig. Super class/Sub class Relationship



Attribute Inheritance

- **Attribute inheritance** is the property by which **subclass entities inherits all** the **attributes** of the **superclass entity.**
- This feature makes it **unnecessary to associate** the superclass attributes with the subclass entities thus **avoid redundancy.**



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.**
- **Specialization** is a **top-down process** of **maximizing the differences** between members of an entity by identifying their **distinguishing characteristics**.
- **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.**
- **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.***

Representation of a Specialization diagrammatically in an EER

- The **subclasses** that define a **specialization** are attached by **lines** to a **circle** that represents the specialization, which is connected in turn to the **superclass.**
- The ***subset symbol*** on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.
- Attributes that apply only to entities of a particular subclass—such as TypingSpeed of SECRETARY—are attached to the rectangle representing that subclass.
- These are called **specific attributes** (or **local attributes**) of the **subclass.**

Representation of a specialization diagrammatically in an EER

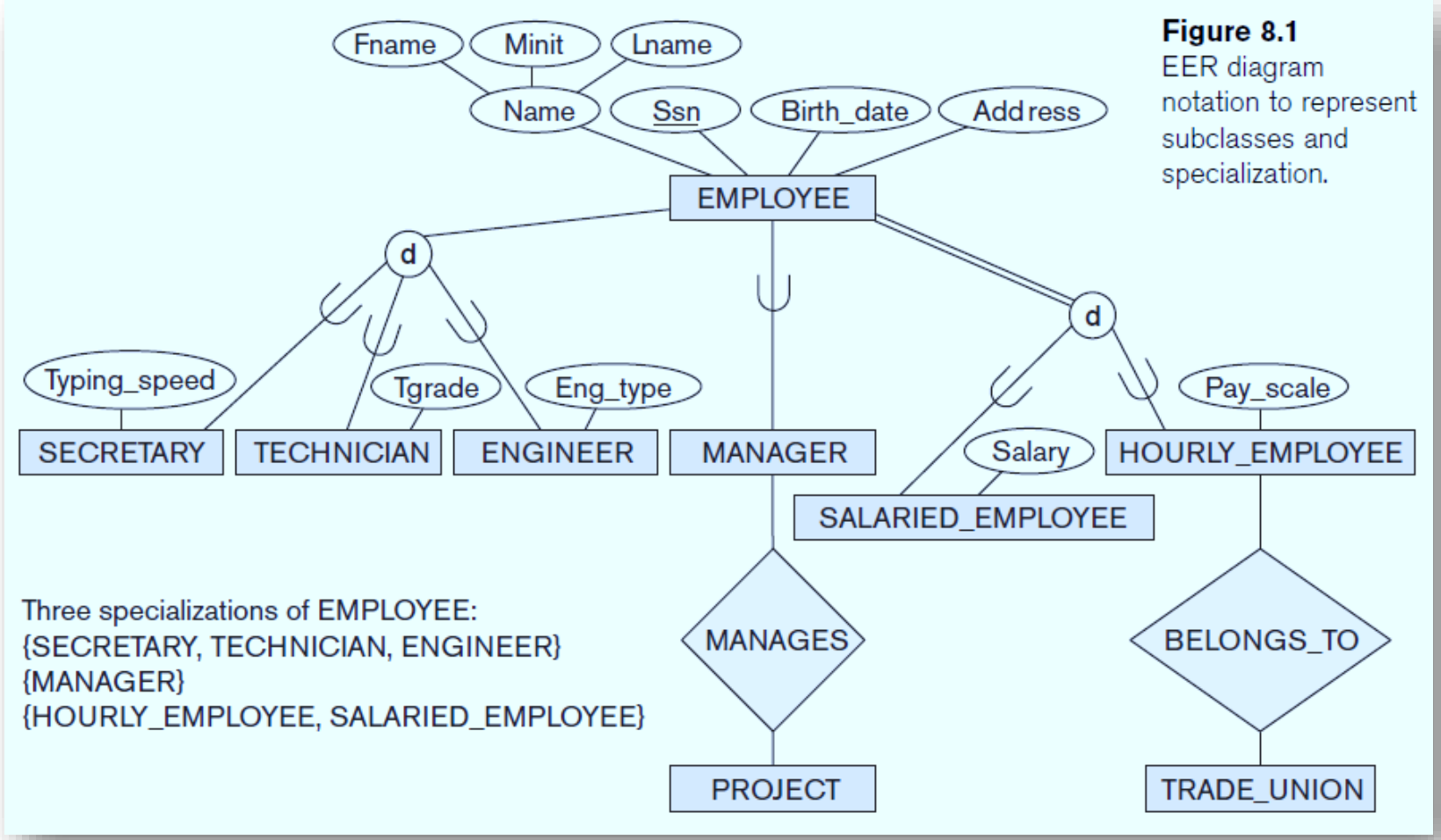
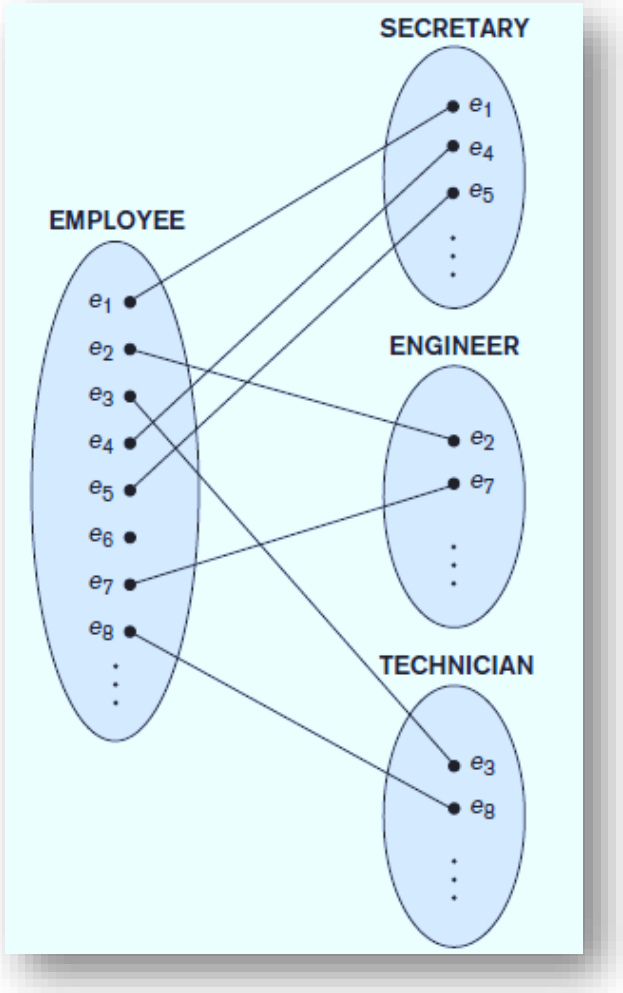


Figure 8.1
EER diagram notation to represent subclasses and specialization.

Instances of a Specialization.

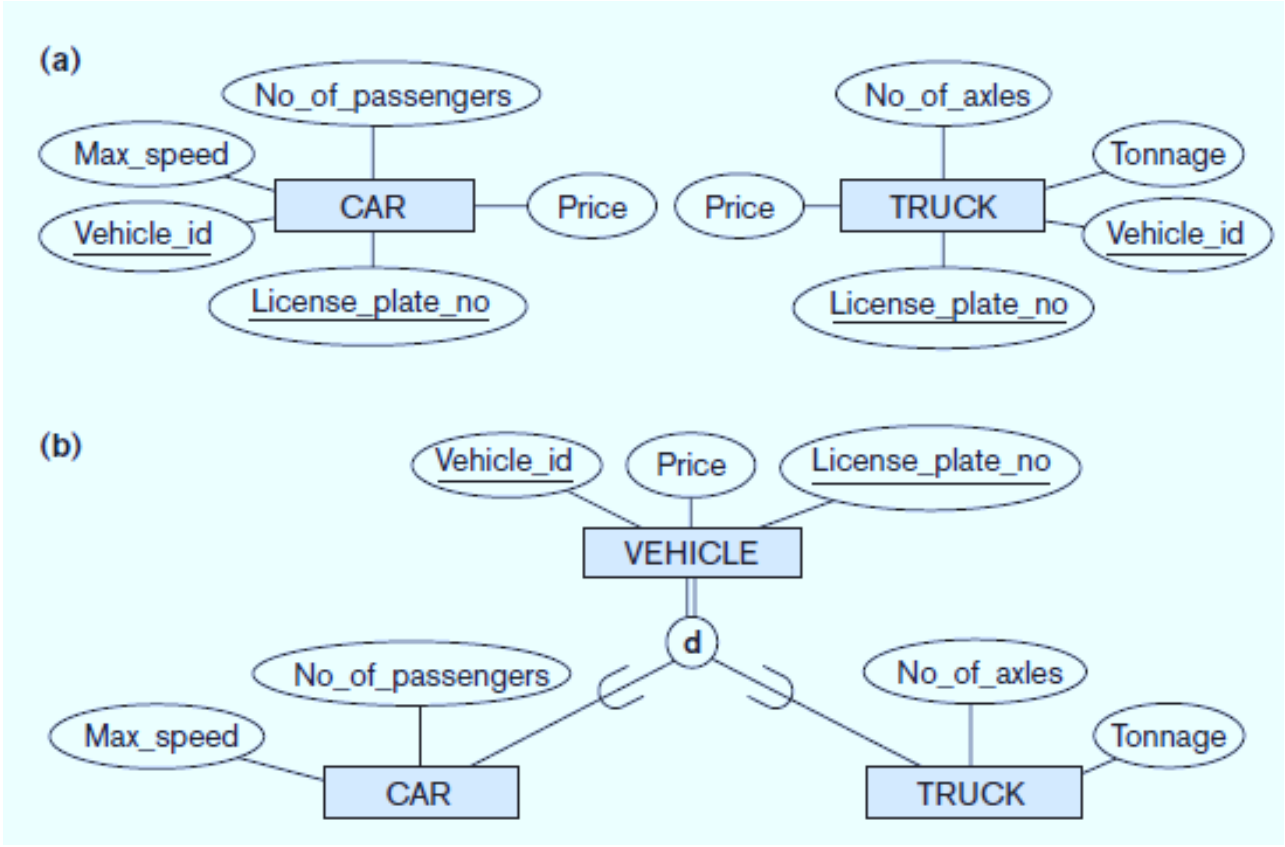


Generalization

- **Generalization** refer to the process of **defining a generalized entity type** from the given entity types.
- **Generalization** is a **bottom-up process** of **minimizing the differences** between members of an entity by identifying their **common characteristics**.
- The **Generalization** process can be viewed as being **functionally the inverse** of the **Specialization** process.
- This approach results in the **identification** of a **generalized superclass** from the **original subclass.**
- **Example**, consider the **entity types** CAR and TRUCK.
- As they have several common attributes, they can be generalized into the entity type VEHICLE.
- Both CAR and TRUCK are now subclasses of the **generalized superclass** VEHICLE.

Representation of a Generalization diagrammatically in an EER

- An arrow pointing to the generalized superclass represents a generalization,.



Constraints on Specialization and Generalization

- To model an enterprise more accurately, the database designer may choose to place certain **constraints** on a particular **Generalization/Specialization.**

  1. **Membership Constraints**
  2. **Disjoint Constraints**
  3. **Overlapping Constraints**
  4. **Completeness** (or **totalness**) **constraint**

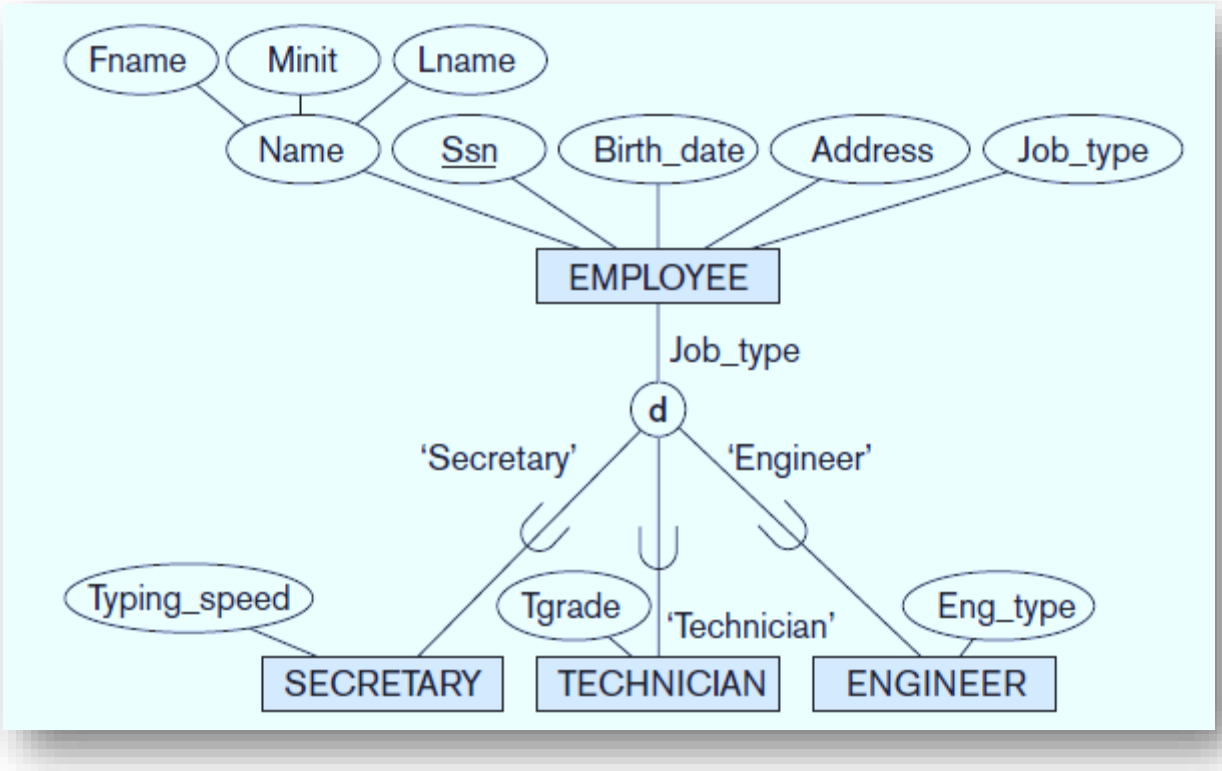
1. Membership Constraints

- This **constraint** is used for determining which entities can be members of a given lower-level entity set.
- 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.**

Constraints on Specialization and Generalization

- Such **subclasses** are called **predicate-defined** (or **condition-defined**) **subclasses**.
- **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.

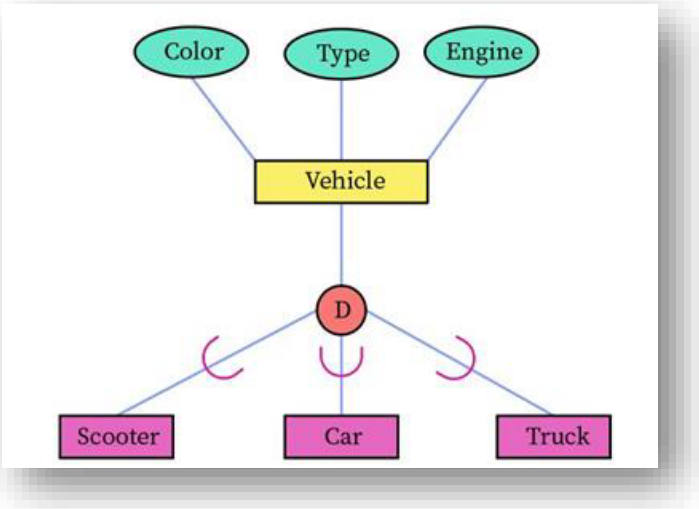
Constraints on Specialization and Generalization



Constraints on Specialization and Generalization

2. Disjoint constraint

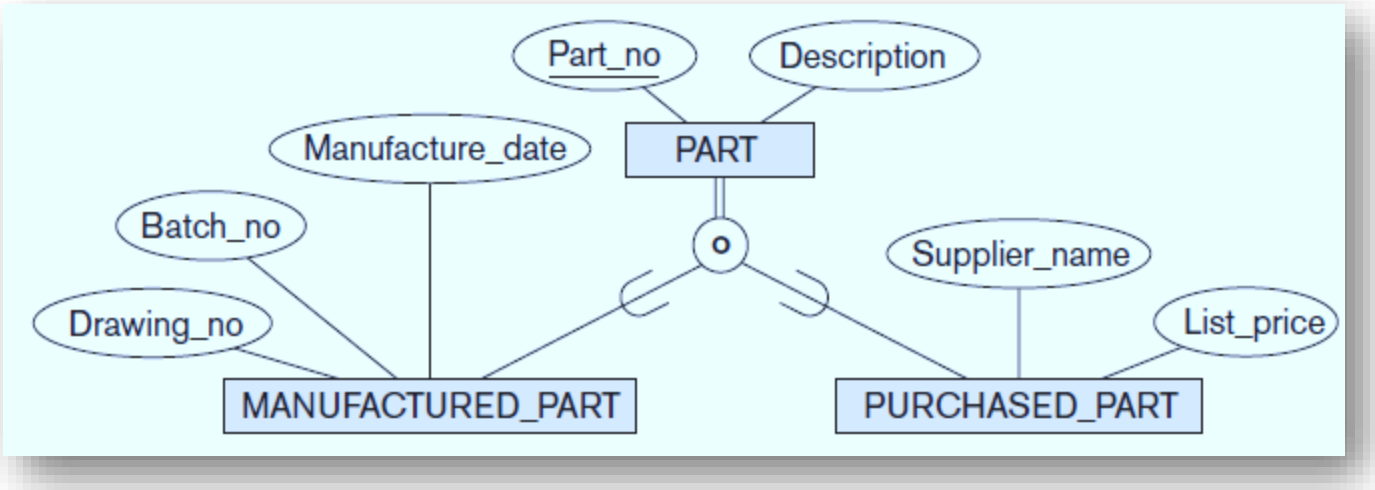
- This **constraint** specifies that the **subclasses** of the **specialization must be disjoint**.
- This means that **an entity** can be a member of **at most one** of the **subclasses** of the **specialization.**
- A **specialization** that is **attribute-defined** implies the **disjoint ness constraint** if the **attribute** used to define the membership predicate is **single valued.**
- It is displayed by placing **d** in the **circle.**



Constraints on Specialization and Generalization

3. Overlap constraint

- This **constraint** specifies that the **subclasses** of the **specialization** are **not constrained** to be **disjoint.**
- This means that an **entity** can be a **member of more then one subclasses** of the **specialization.**
- This case,is displayed by placing **O** in the **circle.**



Constraints on Specialization and Generalization

4. **Completeness** (or **totalness**) **constraint:**

   (a). Total specialization

   (b). Partial specialization

(a). Total specialization

- **Total specialization** constraint specifies that ***every*** **entity** in the **superclass** must be a **member of at least one subclass** in the **specialization.**
- For **example,** if every EMPLOYEE must be either an HOURLY_EMPLOYEE or a SALARIED_EMPLOYEE, then the specialization {HOURLY_EMPLOYEE, SALARIED_EMPLOYEE} is a **total specialization** of EMPLOYEE.
- This is shown in **EER diagrams** by using a **double line** to connect the **superclass** to the **circle.**

Constraints on Specialization and Generalization

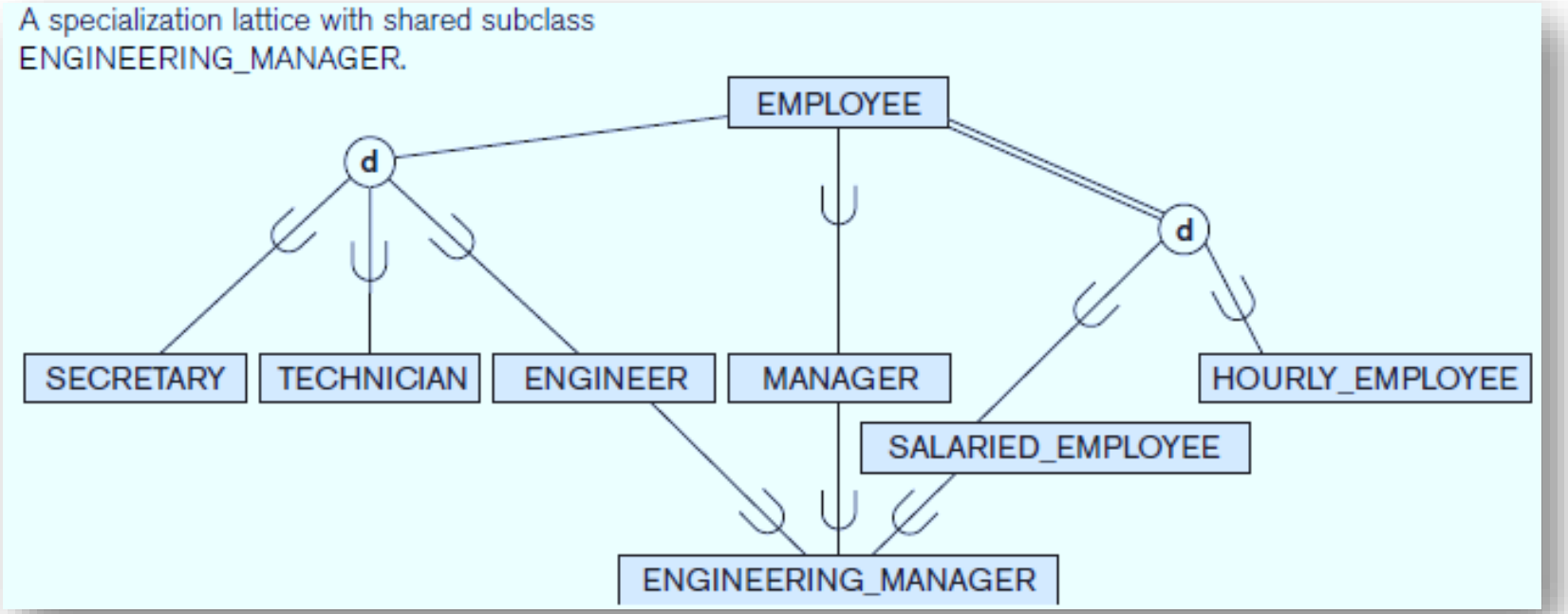
(b). Partial specialization

- **Partial specialization** constraint specifies that ***every*** **entity** in the **superclass** need not be a member of all the subclass in the **specialization.**
- A **single line** is used to display a **partial specialization**, which allows an entity not to belong to any of the subclasses.
- For example, if some EMPLOYEE entities do not belong to any of the subclasses {SECRETARY, ENGINEER, TECHNICIAN}, then that specialization is **partial.**

Specialization Hierarchies and Lattices

- A **subclass itself** may have **further subclasses** specified on it, forming a **hierarchy** or a **lattice of specializations**.
- For **example**, ENGINEER is a **subclass** of EMPLOYEE and is also a **superclass** of ENGINEERING_MANAGER.
- This represents the real-world constraint that every engineering manager is required to be an engineer.
- A **specialization hierarchy** has the **constraint** that each **subclass** has **only one parent**, which results in a **tree structure** or **strict hierarchy**.
- In contrast, for a **specialization lattice**, a **subclass** can have ***more than one*** **parent**.

Specialization Lattice



A specialization lattice with multiple inheritance

- In 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.
- **Example**, an entity in **GRADUATE_STUDENT** inherits all the attributes of that entity as a STUDENT *and* as a PERSON.
- A **subclass** with ***more than one*** **superclass** is called a **shared subclass**, such as ENGINEERING_MANAGER and STUDENT_ASSISTANT.
- This leads to the concept known as **multiple inheritance**, where the **shared subclass** directly **inherits attributes** from **multiple classes**.
- The **existence** of **at least one shared subclass** leads to a **lattice** (and hence to ***multiple inheritance*****).**
- If **no shared subclasses** existed, we would have a hierarchy rather than a lattice and only **single inheritance** would exist.

A specialization lattice with multiple inheritance

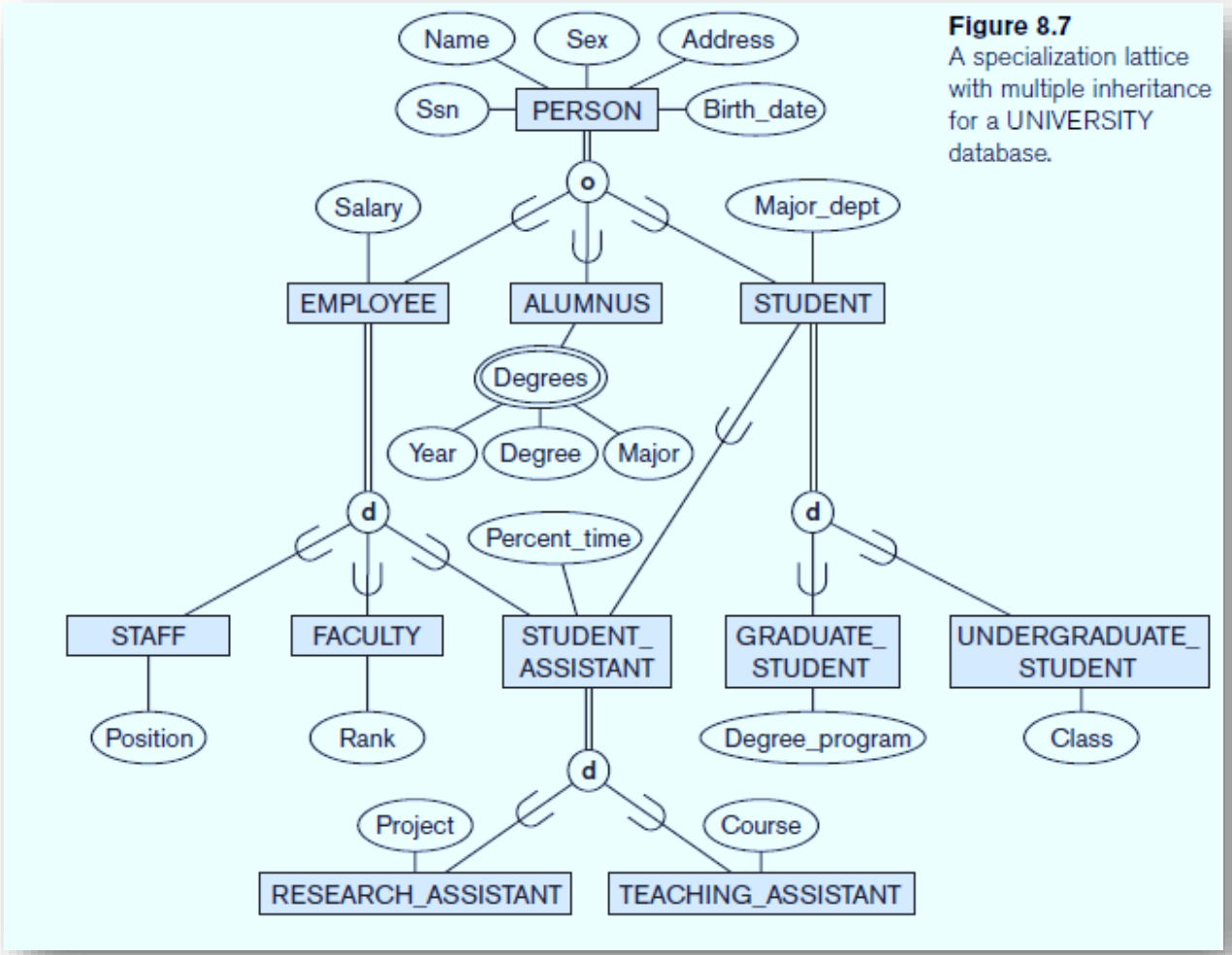
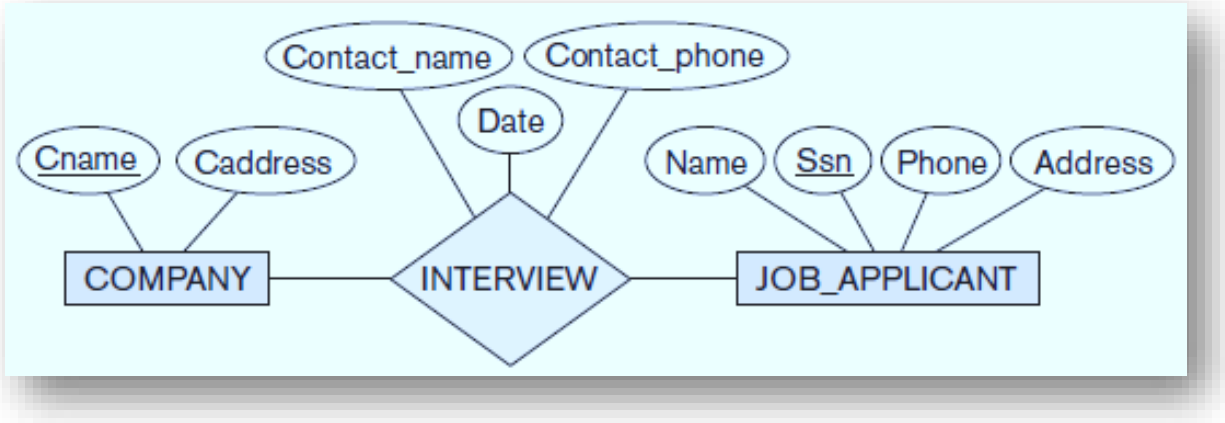


Figure 8.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Aggregation

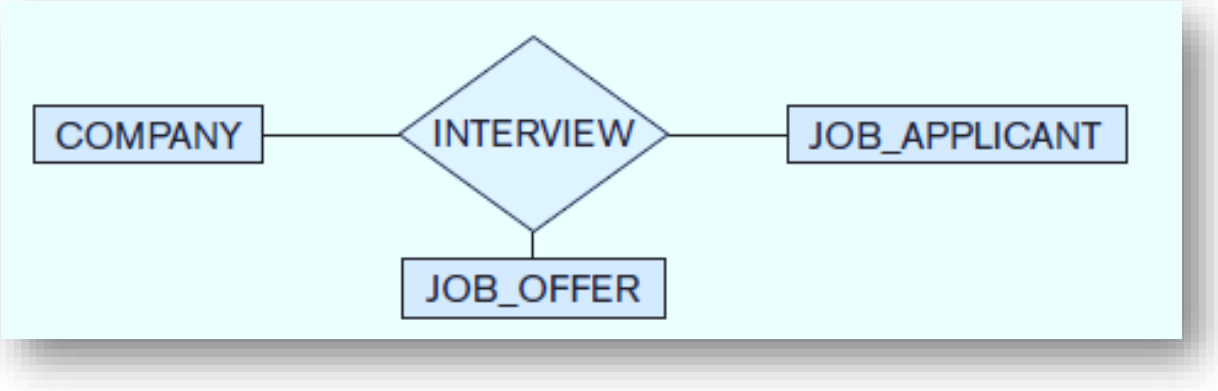
- **Aggregation** is an **abstraction concept** for building **Composite objects(Entity set, Relationship Set)** from their **component objects.**
- Consider the **ER schema** shown in Figure, which stores information about **interviews** by **job applicants** to various companies.



- The **relationship** **INTERVIEW** **attributes** Contact_name and Contact_phone represent the name and phone number of the **person** in the company who is responsible for the interview.

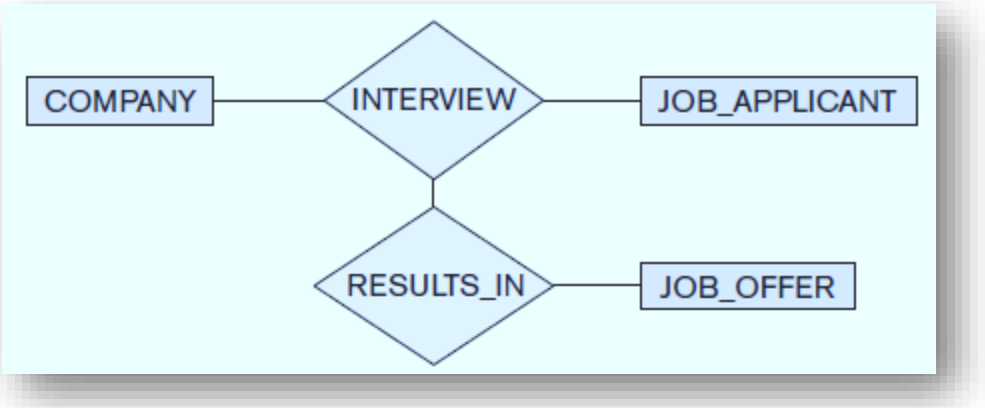
Aggregation

- Suppose that **some interviews** result in **job offers**, whereas others do not.
- We would like to treat INTERVIEW as a class to associate it with JOB_OFFER.
- The **schema** shown in Figure is ***incorrect*** because it requires each interview relationship instance to have a job offer.



Aggregation

- The schema shown in Figure is ***not allowed*** because the **ER model does not allow relationships among relationships.**



- One way to represent this situation is to create a **higher-level aggregate class** composed of COMPANY, JOB_APPLICANT, and INTERVIEW and to relate this class to JOB_OFFER, as shown in Figure.
- Thus **Aggregation** is an **abstraction** through which **relationships** are treated as **higher level entities.**

Aggregation

- Thus, for our example, we regard the **relationship set *INTERVIEW*** (relating the entity sets *COMPANY and JOB_APPLICANT*) as a **higher-level entity set.**
- Such an **entity set** is treated in the same manner as is any other entity set.
- We can then create a **binary relationship** *RESULTS_IN* between ***aggregated entity set*** and ***JOB_OFFER.***