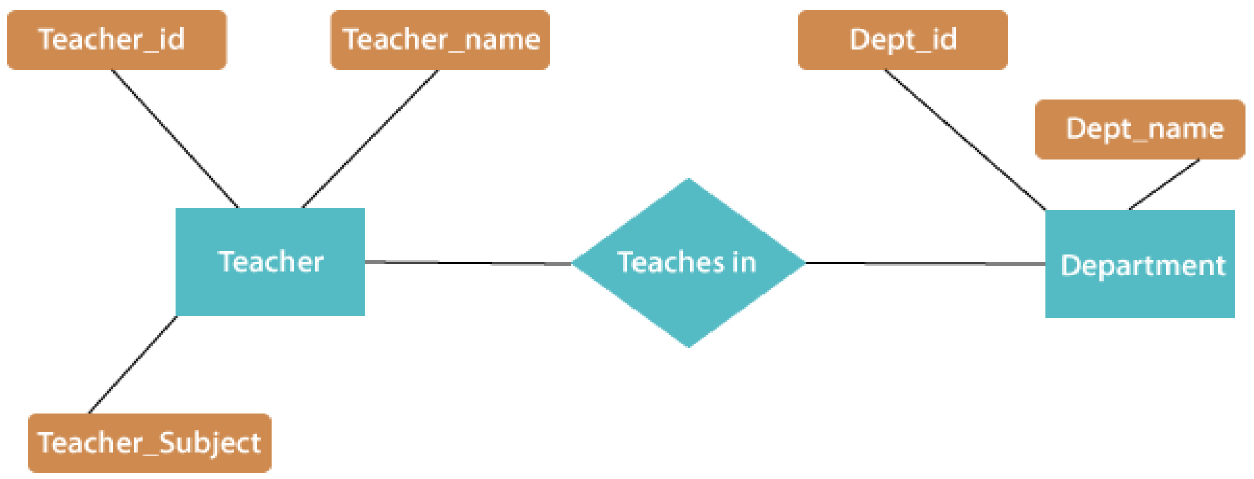
University of Bisha
Faculty Of Computing and Information Technology

Part (4)

Entity- Relationship (E-R) Model

Entity-Relationship Model



Outlines of Part 4 $\Rightarrow$ use Chapter 3 & 4

- Enhanced Entity- Relationship (EER) Model

Introduction

- **Data Modeling Using the Entity-Relationship (ER) Model**

- **Entity-Relationship (ER) model**
  - Popular high-level conceptual data model
- **ER diagrams (ERD)**
  - Diagrammatic notation associated with the ER model

ER Model Concepts

Database designers interview prospective database users at any an organization to understand and document data requirements

When we analyze the information needs of an organization, we attempt to identify ***entities, attributes, and relationships.***

- **Entity:** is a distinct object (a person, place, thing, concept, or event) in the organization that is to be represented in the database.
- **Attribute** is a property that describes some aspect of the object that we wish to record.
- **Relationship** is an association between entities.

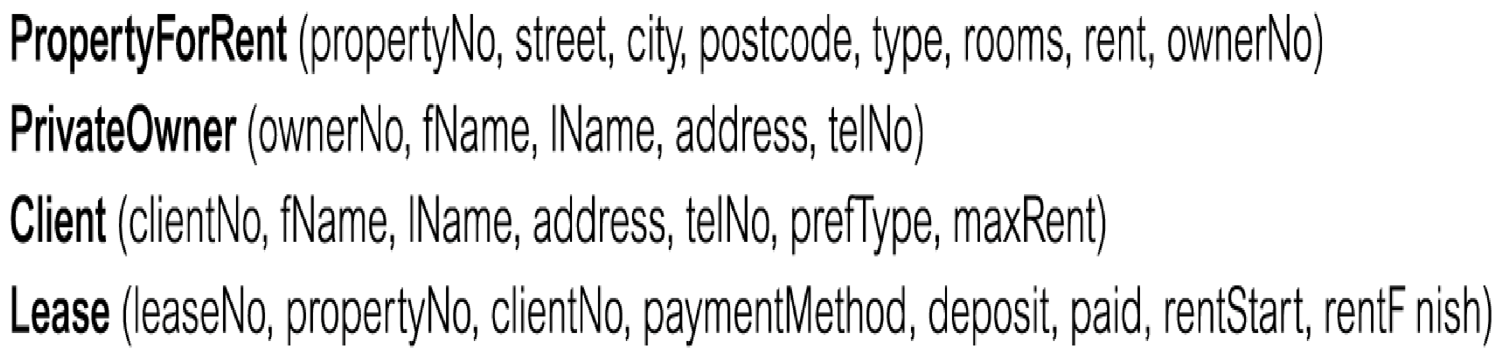
❑ After the requirements have been gathered, they are transformed into an Entity Relationship (E-R) Data Model.

❑ Entity Relationship Modeling (ER) is used by database designers to define the architecture of database systems.

ER Model Concepts – An Example

For **example**, in the *DreamHome* case study, we may be interested in modeling:

- the “real-world” **entities** ***Staff, PropertyforRent, PrivateOwner****, and* ***Client****;*
- **attributes** describing properties of each entity (for example, each Staff entry has a ***name***, ***position***, and ***salary***);
- **relationships** between these entities (for example, Staff ***Manages*** PropertyforRent**).**



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)

PrivateOwner (ownerNo, fName, lName, address, telNo)

Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentF nish)

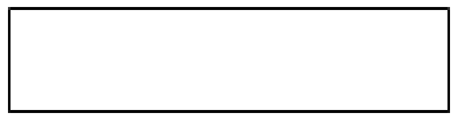
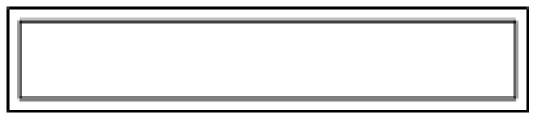
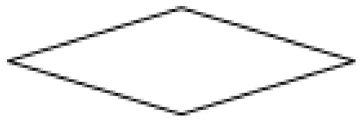
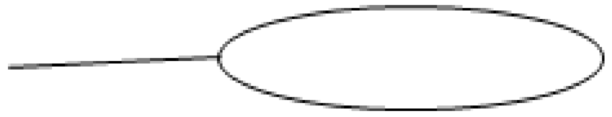
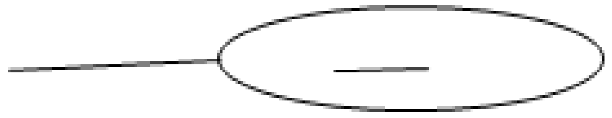
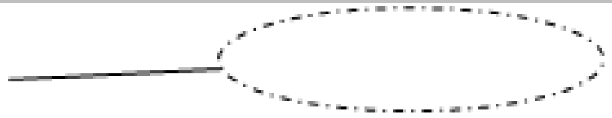
Types of Entities

- **Weak**
- **Strong**

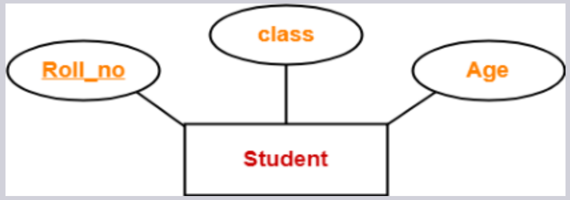
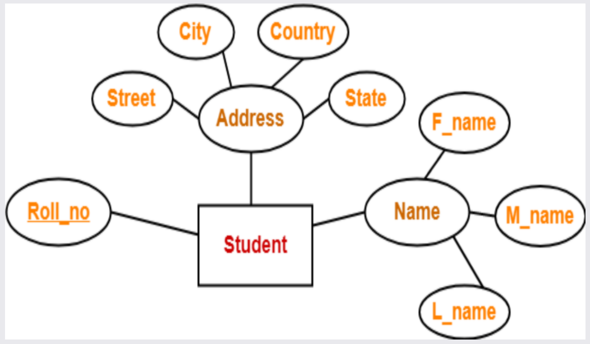
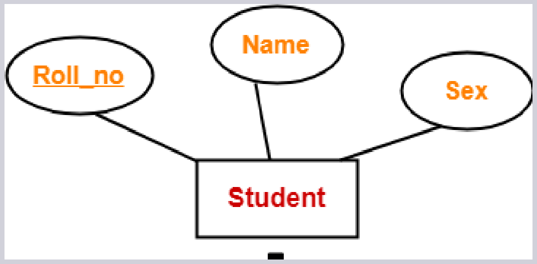
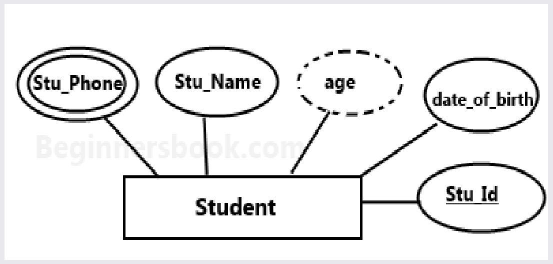
- A **weak entity** is an entity that cannot exist in the database without the existence on another entity
- Any entity that is not a weak entity is called a **strong entity**.
- **Example:** Suppose that a DEPENDENT entity is identified by the dependent's first name and birhtdate, and the specific EMPLOYEE that the dependent is related to.

DEPENDENT is a weak entity type with EMPLOYEE

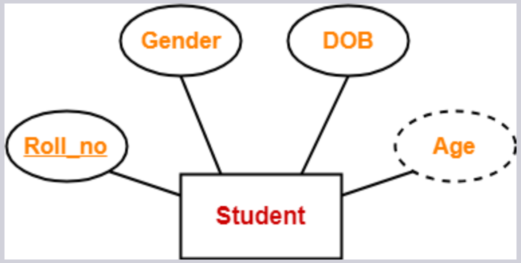
Summary of ER-Diagram Notation for ER Schemas

ENTITY TYPE	
WEAK ENTITY TYPE	
RELATIONSHIP TYPE	
ATTRIBUTE	
KEY ATTRIBUTE	
DERIVED ATTRIBUTE	

Types of Attributes

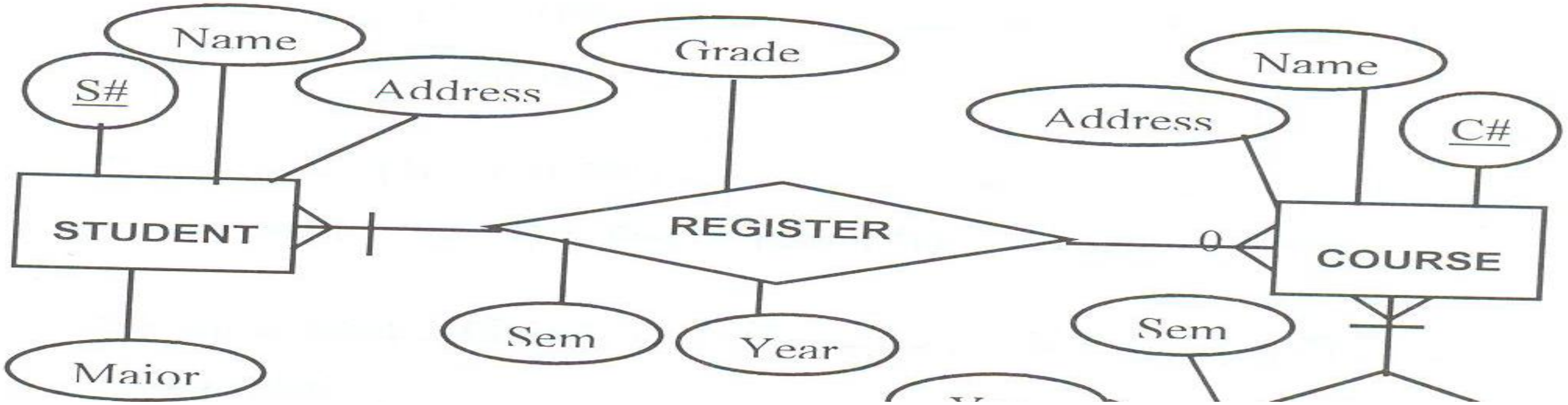
Type	Description	Examples	ER-Diagram Notation
Simple	atomic values, which cannot be divided further	student's age or phone number is an atomic value of 10 digits	
Composite	The attribute may be composed of several components	▪Address(Apt#, House#, Street, City, State, ZipCode, Country). ▪Name(FirstName, MiddleName, LastName).	 City Country Street Address State F_name Roll_no Name M_name Student L_name
Single - valued	Each entity has a single atomic value for the attribute	SSN, Sex or student's Roll number	
Multi-valued	An entity may have multiple values for that attribute	student's phone numbers if student has more than one phone number	 Stu_Phone Stu_Name age date_of_birth Student Stu_Id

Types of Attributes…………Cont

Type	Description	Examples	ER-Diagram Notation
Derived attribute	the attribute that do not exist in the physical database, but their values are derived from other attributes present in the database	Age can be derived from data_of_birth.	

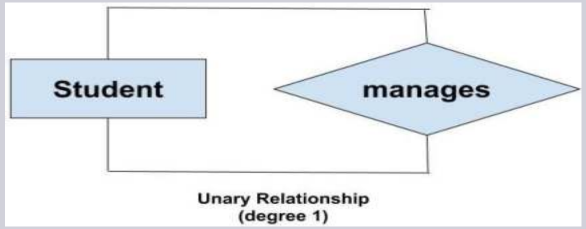
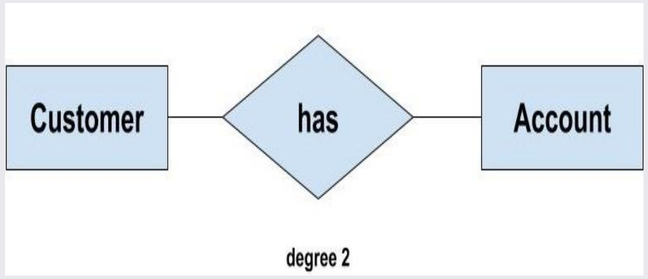
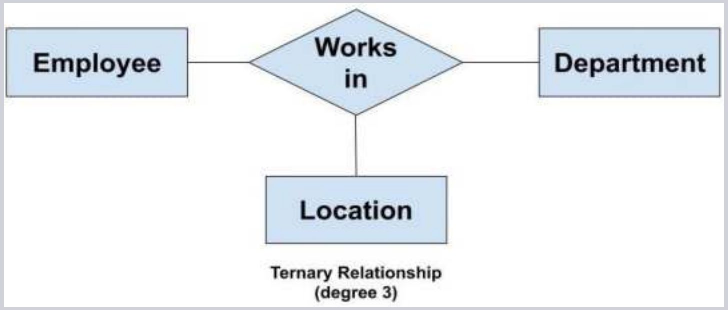
❖**Relationship can also contain attributes.**

For example the relationship **Register** between the **Student** and the **Course** could contain the attributes **Year** and **Sem** and **Grade**, which identify the Year and Semester of the registration and the grade of the course after the course is taken.



Degree of a Relationship

- **Degree of a relationship type**
  - Number of participating entity types

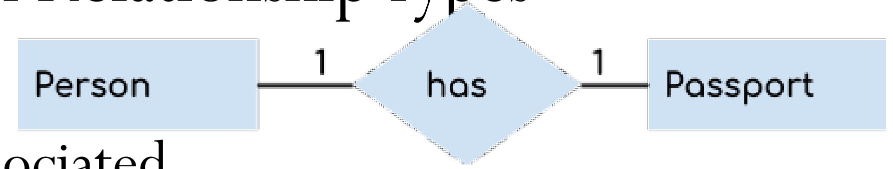
Type	Example
Unary Relationship type of degree one	
Binary Relationship type of degree two	
Ternary Relationship type of degree three	 Employee Works in Department Location Ternary Relationship (degree 3)

Constraints on Relationships

- We can distinguish two main types of binary relationship
- constraints: *cardinality ratio* and *participation.*

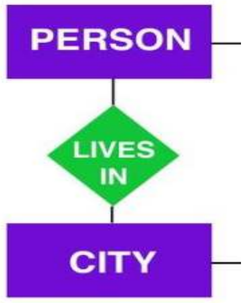
❖ ***Cardinality ratio Constraints*** on Relationship Types

- **One-to-one (1:1)**



When only one instance of an entity is associated with the relationship

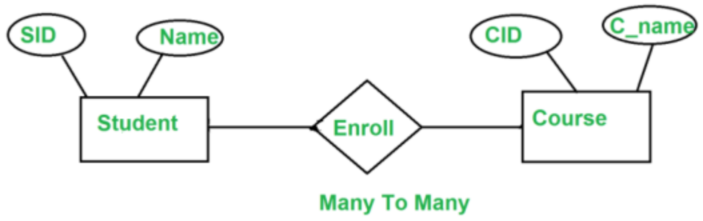
- **One-to-many (1:N) or Many-to-one (N:1)**



When more than one instance of an entity is associated with a relationship

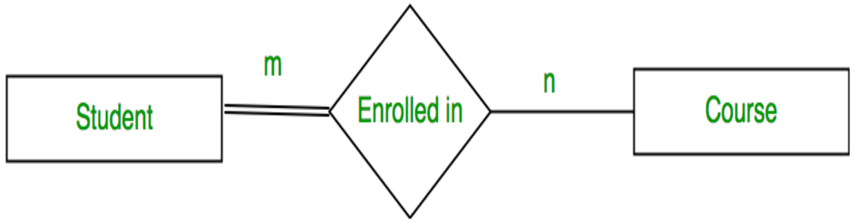
- **Many-to-many (M:N)**

more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship



Constraints on Relationships

- ***Participation constraint*** (on each participating entity type):
- **total** (called existence dependency) or **partial** **(eg**. Student & Medical file).
  - Total shown by double line (═══), partial by single line (───).



- ***(min, max) notation:***
- In this section, we describe one alternative ER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double-line notation for participation constraints.

- This notation involves associating a pair of integer numbers (min, max) with each *participation* of an entity type *E in a relationship type R, where 0 ≤ min ≤ max and max ≥ 1.*

Constraints on Relationships - (min, max) notation

The numbers mean that for each entity *e in E, e must participate in* ***at least*** *min and* ***at most*** *max relationship instances in R at any point in time.*

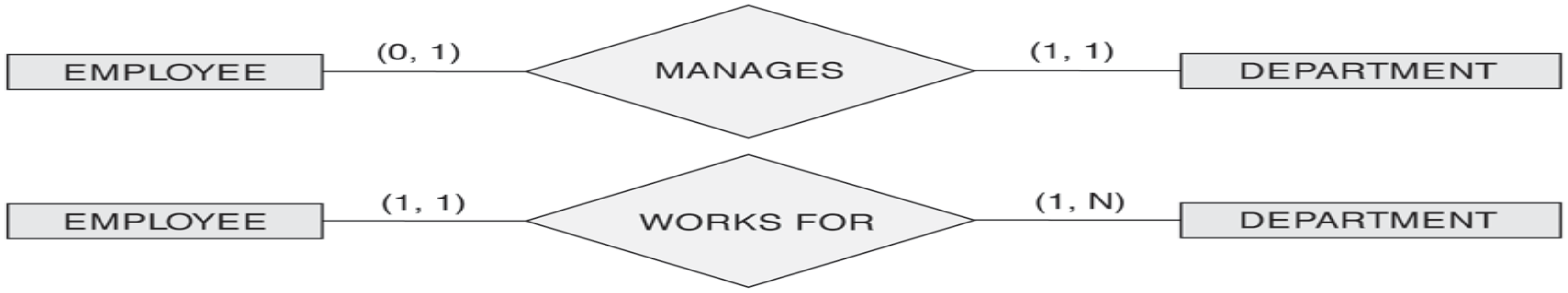
In this method, ***min = 0*** implies ***partial*** participation, whereas ***min > 0*** implies ***total*** participation.

Figure 3.15 displays the COMPANY database schema using the (min, max) notation.

Usually, one uses either the cardinality ratio/single-line/double-line notation *or the (min, max) notation.*

Constraints on Relationships …….Cont

- **Examples:**
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR



Read the min,max numbers next to the entity type and looking **away from** the entity type

ERD Example (1) - College database

- Design an ERD for a college database. The database store:
  - Students' data: academic number, name, address, specialty, courses taken by the student, the semester and the academic year in which the courses were registered and the result of each course.
  - Teachers' data: number, name, mobile, department, the courses they teach, the semester and the academic year in which each course was taught.
  - Courses' data: number, name, credit hours

The design steps:

1. Identify the entities.
2. Identify the relationships.
3. Draw the ERD.

ERD Example (1).......Cont

- **First: Identify the entities:**

  Student entity attributes: academic number (S#), student name (name), student address (address), student major (major).

  Faculty entity attributes: teacher number (F#), teacher name (name), teacher phone (phone), teacher department (department).

  Course entity attributes: course number (C#), course name (name), Course hours (hours).

- **Second: Identify the relationships.**

1. **The relationship "Register":**

   It connect "Student entity" with "Course entity"

   Its attributes: Year, Semester, grade

   Type: Many-to-Many

1. **The relationship "Teach":**

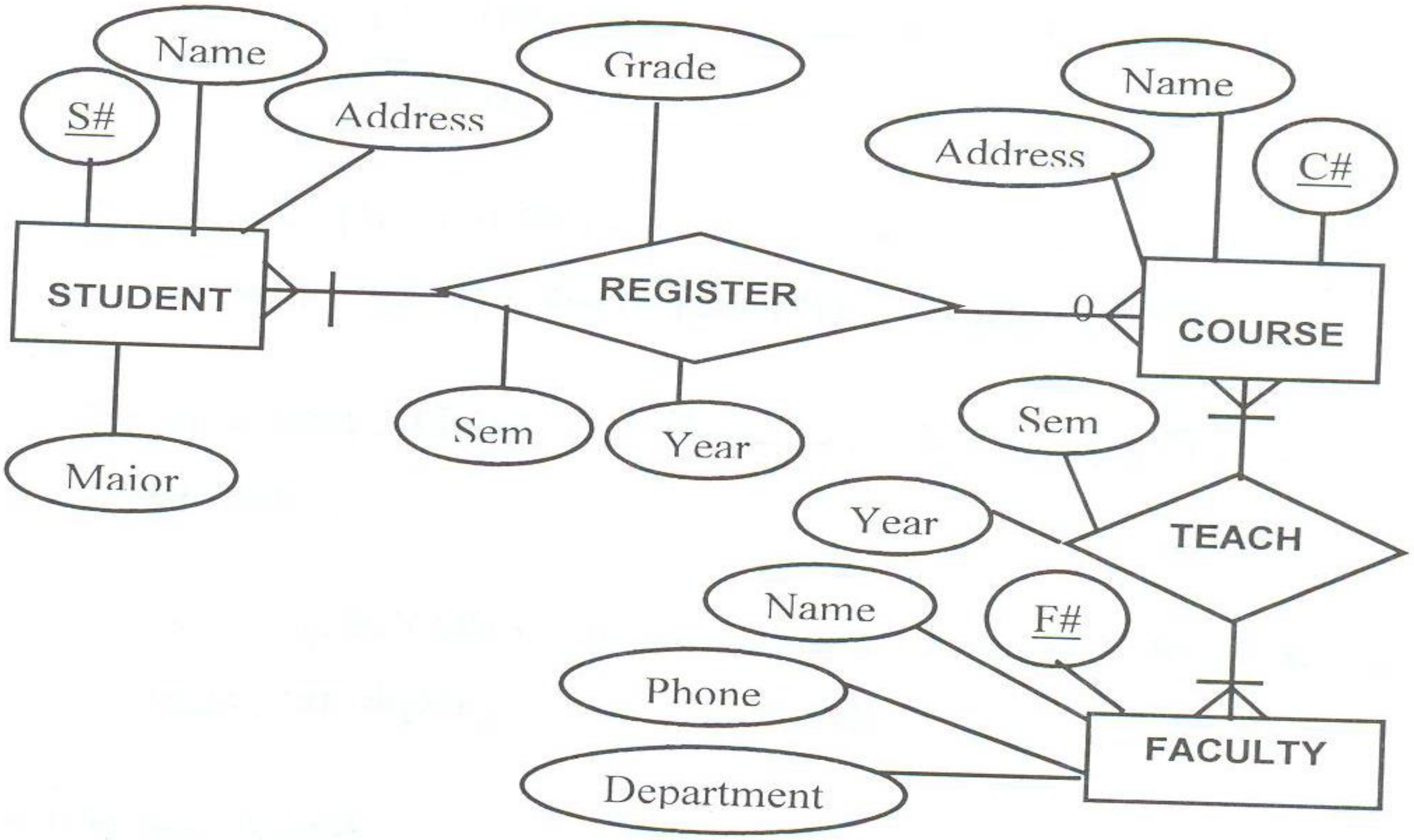
   It connect "Faculty entity" with "Course entity"

   Its attributes: Year, Semester

   Type: Many-to-Many

ERD Example(1)…….Cont

- Third: Draw the ERD.



ERD Example (2)……COMPANY

- **Requirements** of the COMPANY ERD:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department.
  - We keep track of the start date of the department manager. A department may have several locations.
  - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

ERD Example (2)……COMPANY…Cont

- The database will store each employee’s name, Social Security number,address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).

- The database will keep track of the dependents of each employee for insurance purposes, including each dependent’s first name, sex, birth date, and relationship to the employee.

ERD Example (2)……COMPANY….Cont.

- Based on the requirements, we can identify four initial ***entity*** types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

ERD Example (2)……COMPANY….Cont.

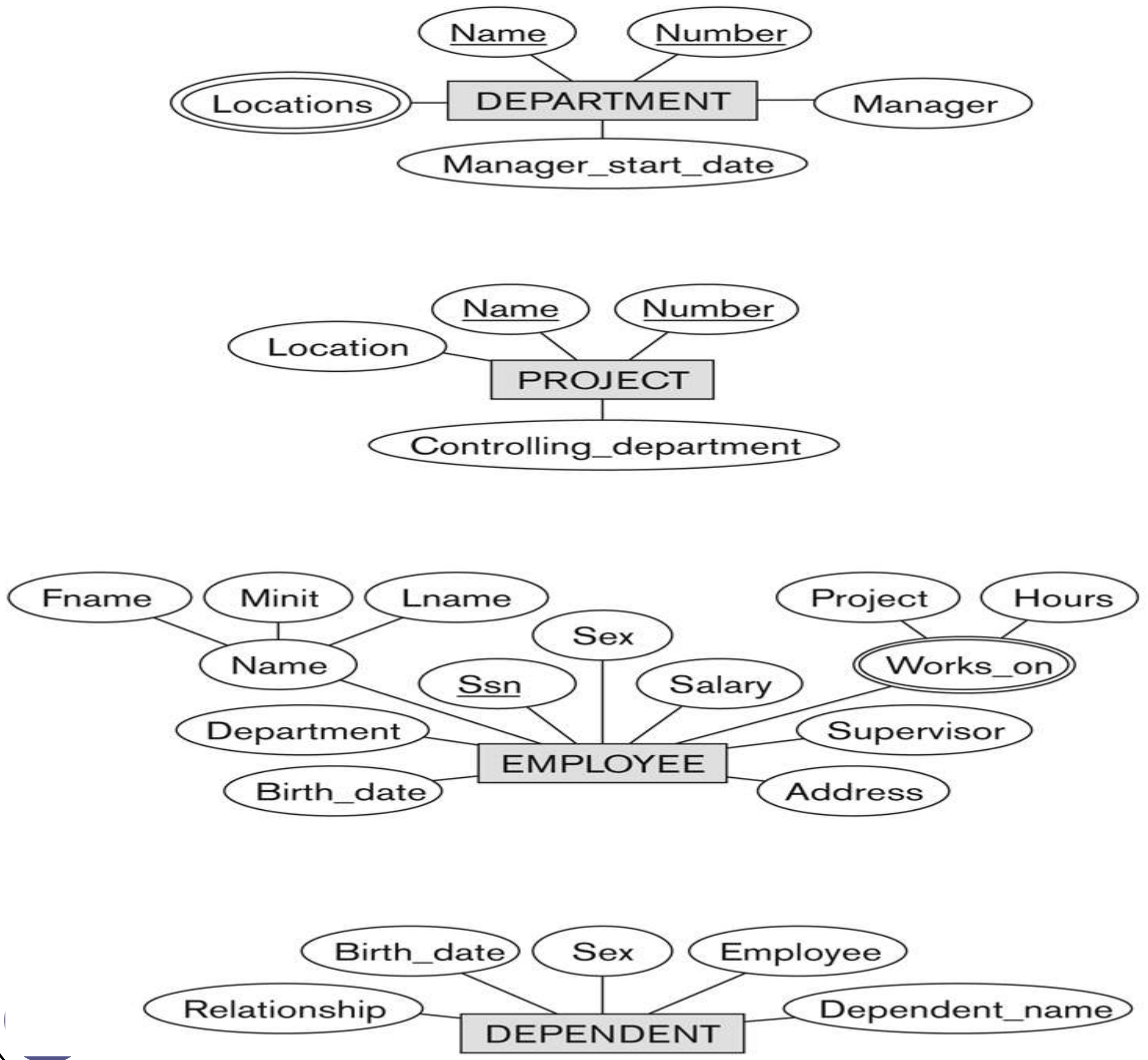


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

ERD Example (3)……COMPANY….Cont.

Relationships:

By examining the requirements, six relationship types are identified All are *binary* relationships(degree 2)Listed below with their participating entity types:

- WORKS_FOR (between EMPLOYEE, DEPARTMENT)
- MANAGES (also between EMPLOYEE, DEPARTMENT)
- CONTROLS (between DEPARTMENT, PROJECT)
- WORKS_ON (between EMPLOYEE, PROJECT)
- SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
- DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER Diagram For a Company Database……….COMPANY……Cont

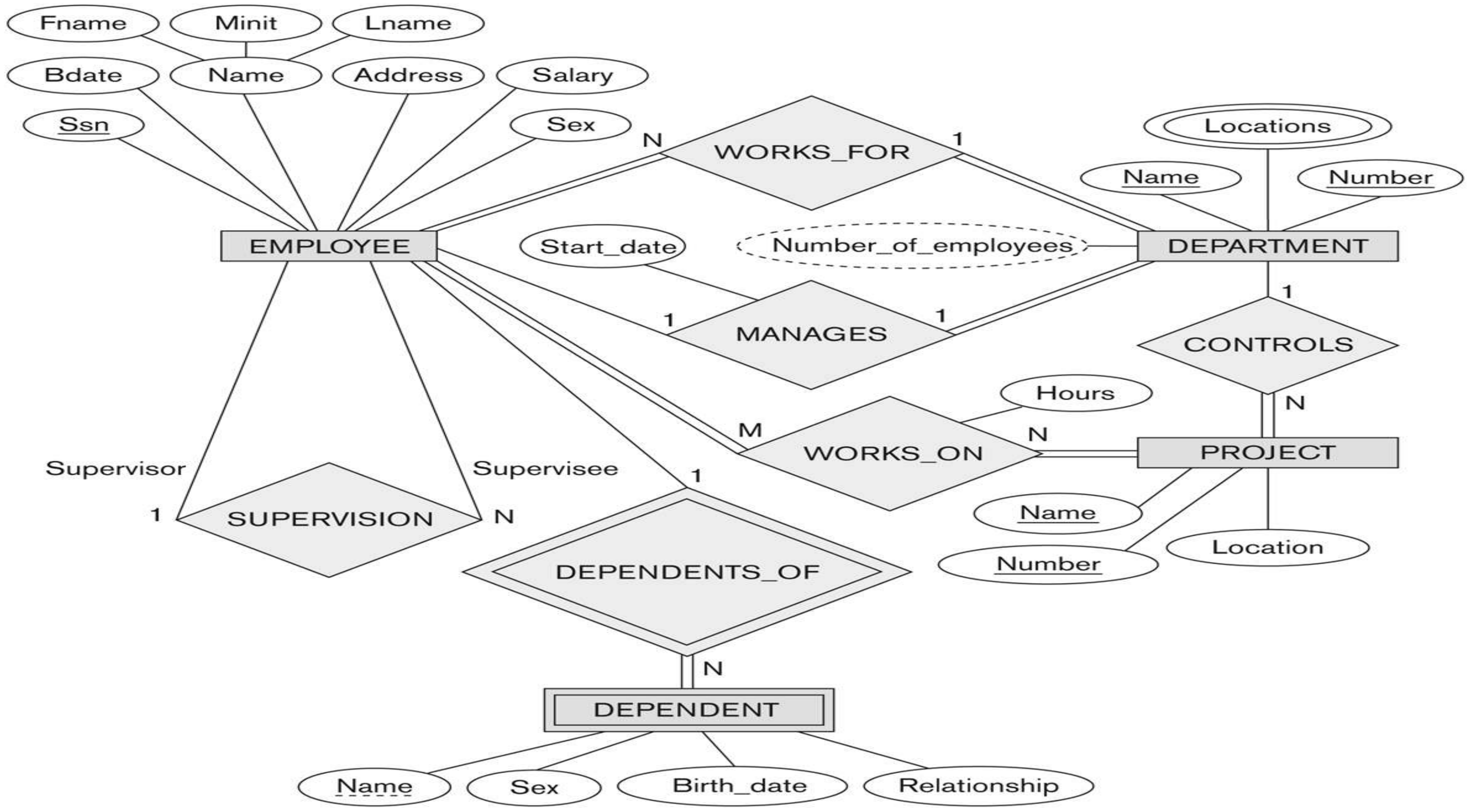


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

ER Diagram For a Company Database..........COMPANY......Cont

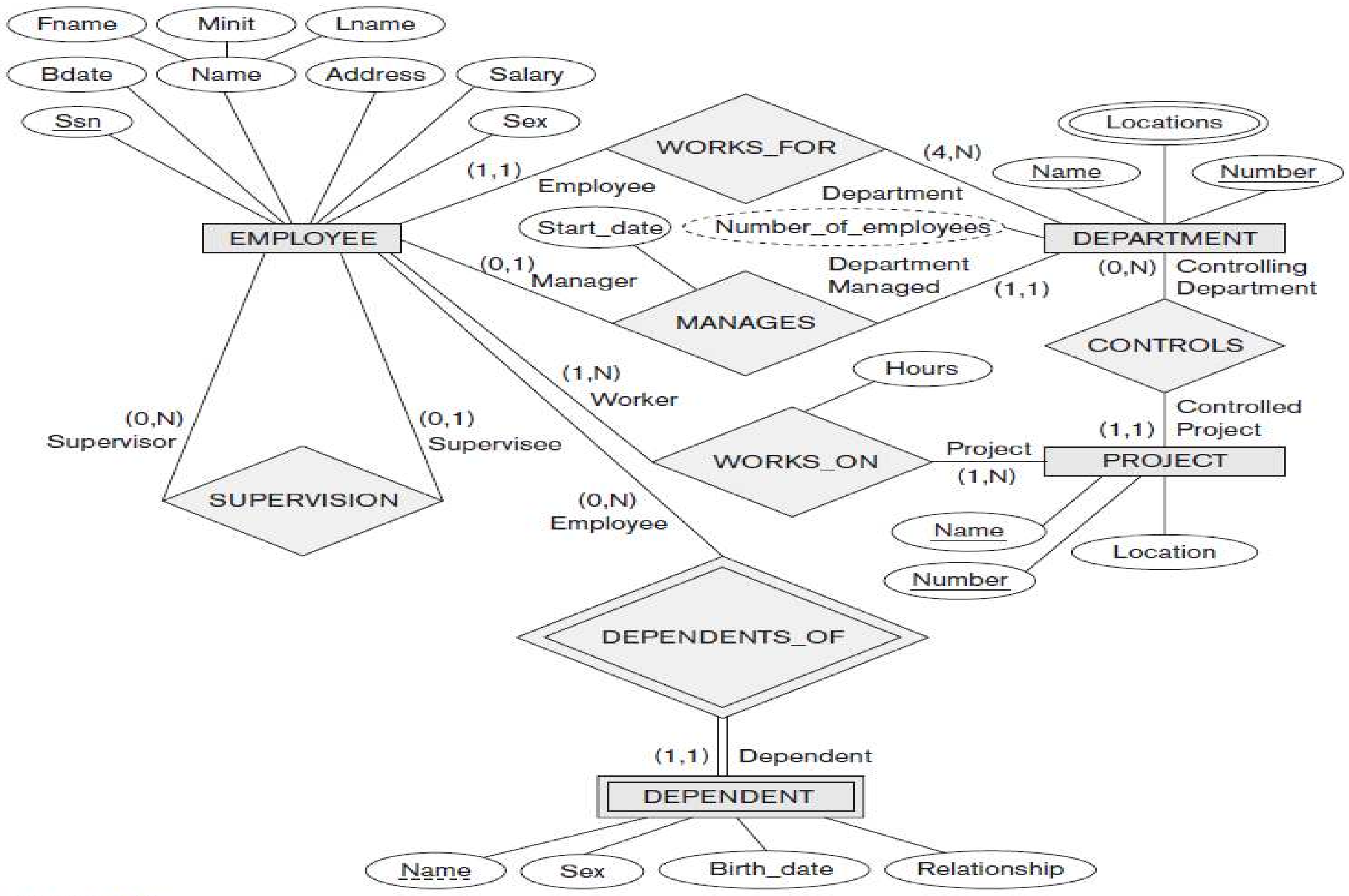


Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Mapping ERD model to Relational Database design

- Two tasks to design a database from ERD model:

1. Mapping entity to relation.
2. Mapping relationship to relation.

In general, we convert entities to relations and ER relationships to relations. For 1:1 and 1:N relationships, foreign keys can be used instead of separate relations

- **First Mapping entity to relation:**

✓ The relations' attributes include the entity attributes.

✓ The primary key of the relation is the primary key of the entity.

Database design by ERD model….Cont.

- **Second Mapping relationship to relation:**

  The relations' attributes include: Attributes of the relationship + Primary keys of the two connected entities.

  The primary key of the relation depend on the relationship type.

1. If the relationship type is One-to-One: Can be either from any of the connected entities.

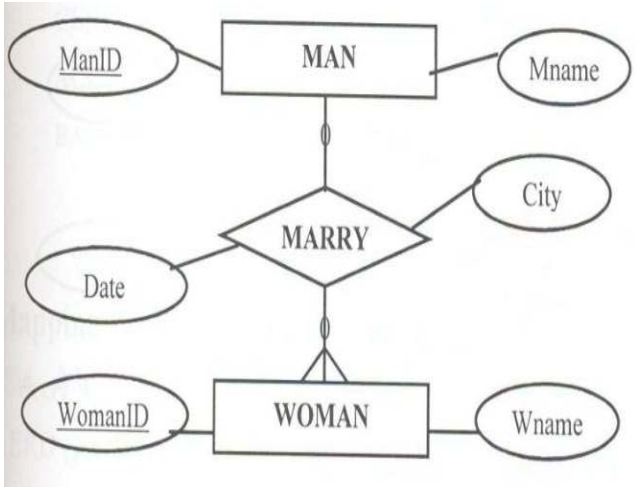
Has (Student_No, File_No)

Has (Student_No, File_No)

Database design by ERD model….Cont.

2. If the relationship type is One-to-Many: The primary key is the primary key of the entity on the side (**Many**)

Mary (<u>WomanID</u>, ManID, Date, City)



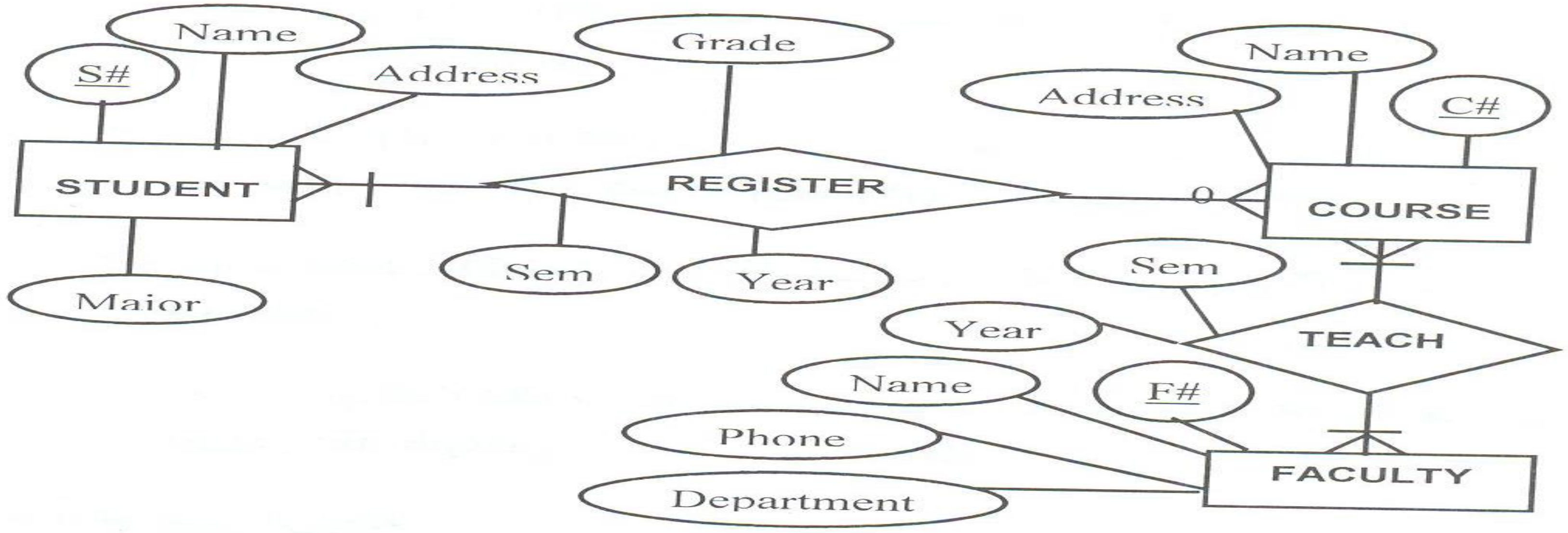
3. If the relationship type is Many-to-Many: The primary key is composite of the two primary keys of the two connected entities.

Register (<u>Student#, Course#</u>, Year, Sem)

Database design by ERD model….Cont.
College Database

- **College Database:**

1. Student (<u>S#</u>, name, address, major)
2. Course (<u>C#</u>, name, hours)
3. Faculty (<u>F#</u>, name, department, phone)
4. Register (<u>S#, C#</u>, Yaer, Sem, Grade)
5. Teach (<u>F#, C#</u>, Year, Sem)



The Enhanced Entity-Relationship (EER) Model

- EER stands for ***Enhanced*** ER or ***Extended*** ER
- **EER Model Concepts**
  - Includes all modeling concepts of basic ER
  - Additional concepts:
    - **subclasses/superclasses**
    - **specialization/generalization**
    - **categories (UNION types)**
  - These are fundamental to ***conceptual*** modeling
- The additional EER concepts are used to model applications more completely and more accurately
  - EER includes some object-oriented concepts, such as inheritance

(EER) Model….Cont.
Subclasses, Superclasses, and Inheritance

- An entity type may have additional meaningful subgroupings of its entities
  - Example: EMPLOYEE may be further grouped into:
    - SECRETARY, ENGINEER, TECHNICIAN, …
      - Based on the EMPLOYEE's Job
    - MANAGER
      - EMPLOYEEs who are managers
    - SALARIED_EMPLOYEE, HOURLY_EMPLOYEE
      - Based on the EMPLOYEE's method of pay
- EER diagrams extend ER diagrams to represent these additional subgroupings, called ***subclasses*** or ***subtypes***

(EER) Model….Cont.
Subclasses, Superclasses, and Inheritance

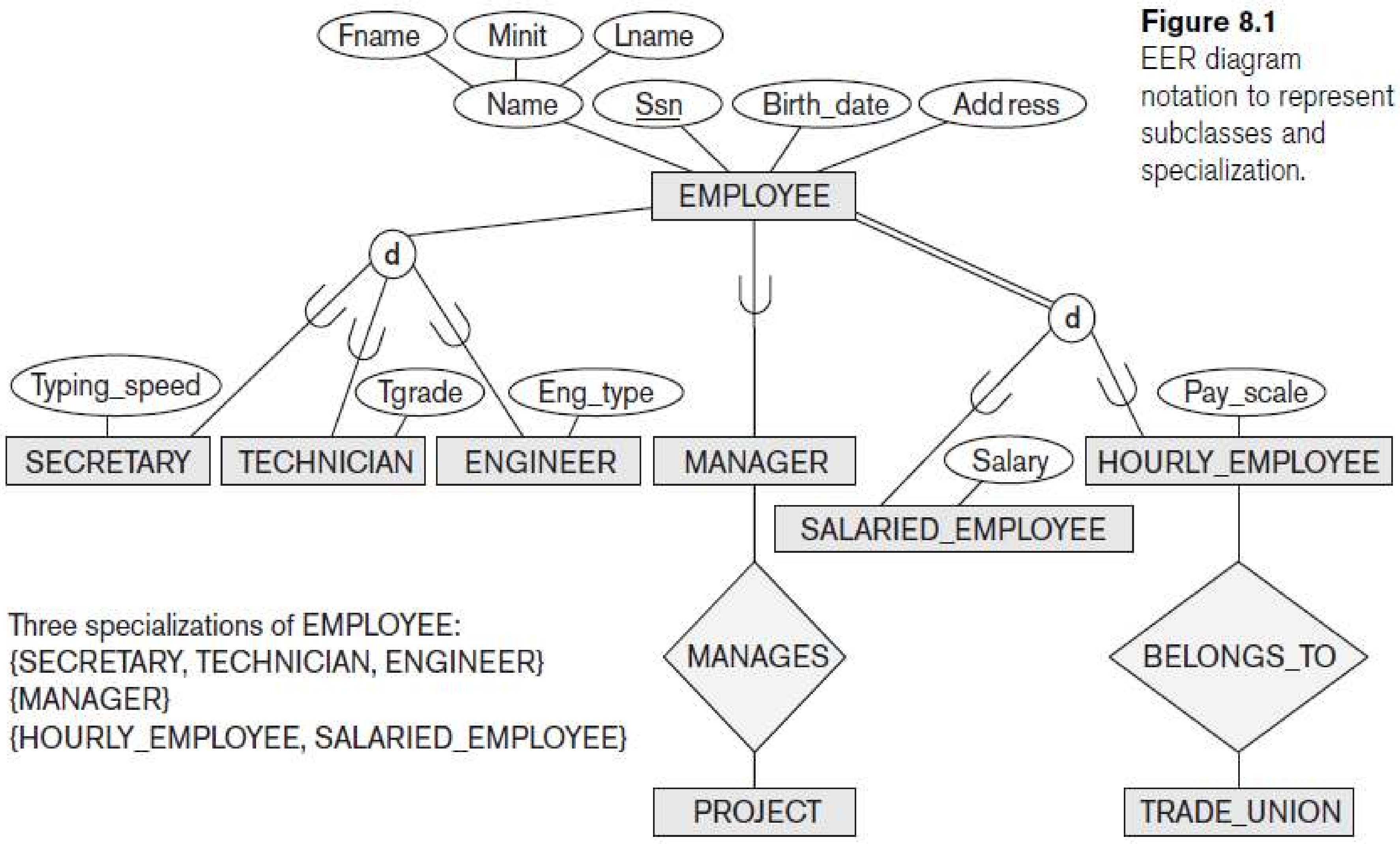


Figure 8.1
EER diagram notation to represent subclasses and specialization.

(EER) Model….Cont.
Subclasses, Superclasses, and Inheritance

- Each of these subgroupings is a subset of EMPLOYEE entities
- Each is called a ***subclass*** of EMPLOYEE
- EMPLOYEE is the ***superclass*** for each of these subclasses
  - The subclass member is the same entity in a **distinct specific role** .
  - An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass .
- These are called superclass/subclass relationships:
  - EMPLOYEE/SECRETARY
  - EMPLOYEE/TECHNICIAN
  - EMPLOYEE/MANAGER
- These are also called **IS-A** relationships
  - **SECRETARY IS-A EMPLOYEE**, TECHNICIAN IS-A EMPLOYEE,…

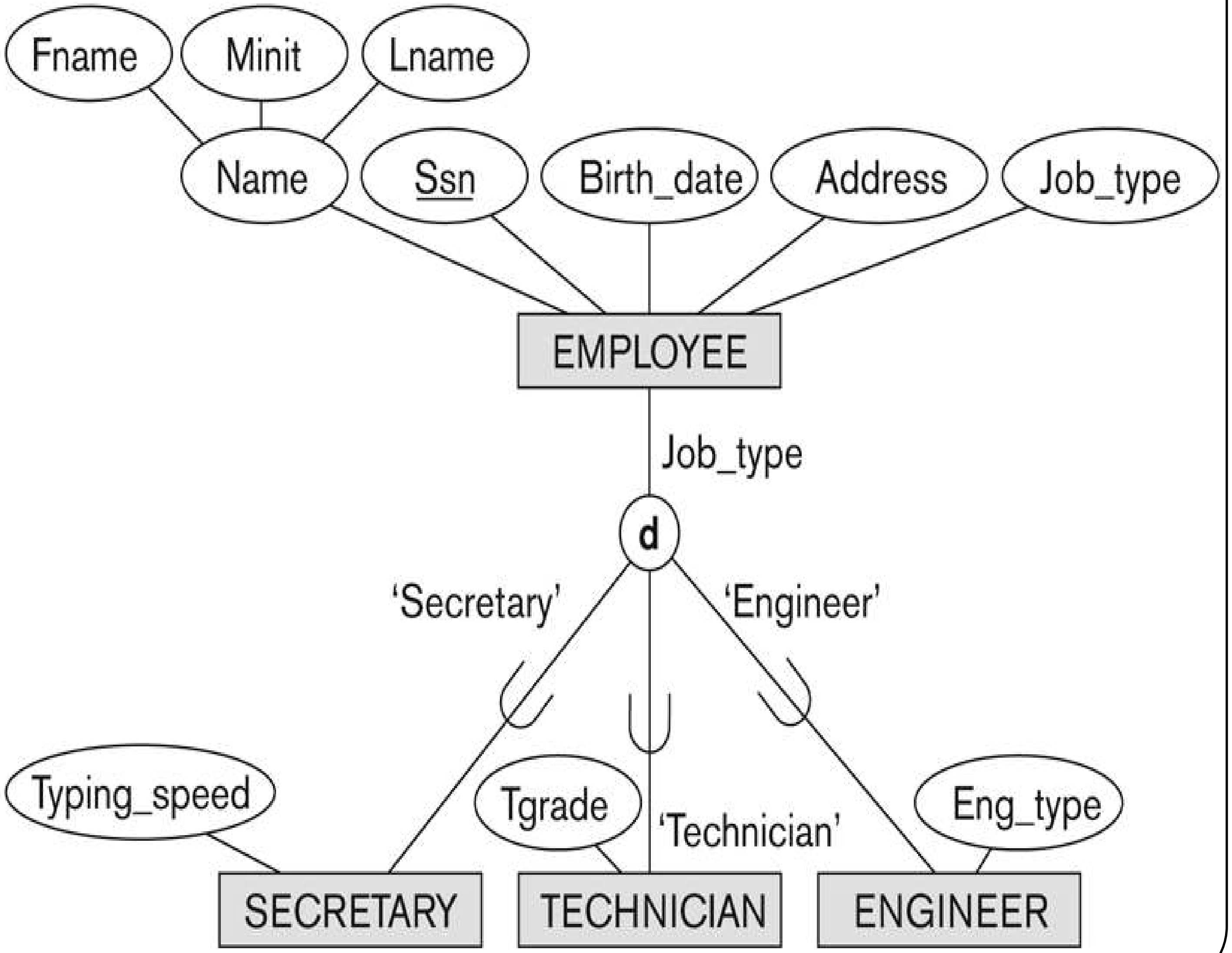
(EER) Model….Cont.
Subclasses, Superclasses, and Inheritance

- A member of the superclass can be optionally included as a member of any number of its subclasses

- Examples:
  - A **salaried** employee who is also an **engineer** belongs to the two subclasses:
    - ENGINEER, and
    - SALARIED_EMPLOYEE
  - A **salaried** employee who is also an **engineering manager** belongs to the three subclasses:
    - MANAGER,
    - ENGINEER, and
    - SALARIED_EMPLOYEE
- It is not necessary that every entity in a superclass be a member of some subclass

(EER) Model….Cont.
Representing Specialization in EER Diagrams

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



Attribute Inheritance in Superclass / Subclass Relationships

- An entity that is member of a subclass ***inherits***
  - All ***attributes*** of the entity as a member of the superclass
  - All ***relationships*** of the entity as a member of the superclass

- **Example:**
  - In the previous slide, SECRETARY (as well as TECHNICIAN and ENGINEER) inherit the attributes Name, SSN, …, from EMPLOYEE
  - Every SECRETARY entity will have values for the inherited attributes

Specialization

- Specialization is the process of defining a set of subclasses of a superclass
- The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass

- **Example:** {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type*.
  - May have several specializations of the same superclass

Specialization ....Cont.

- Example: Another specialization of EMPLOYEE based on ***method of pay*** is {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}.

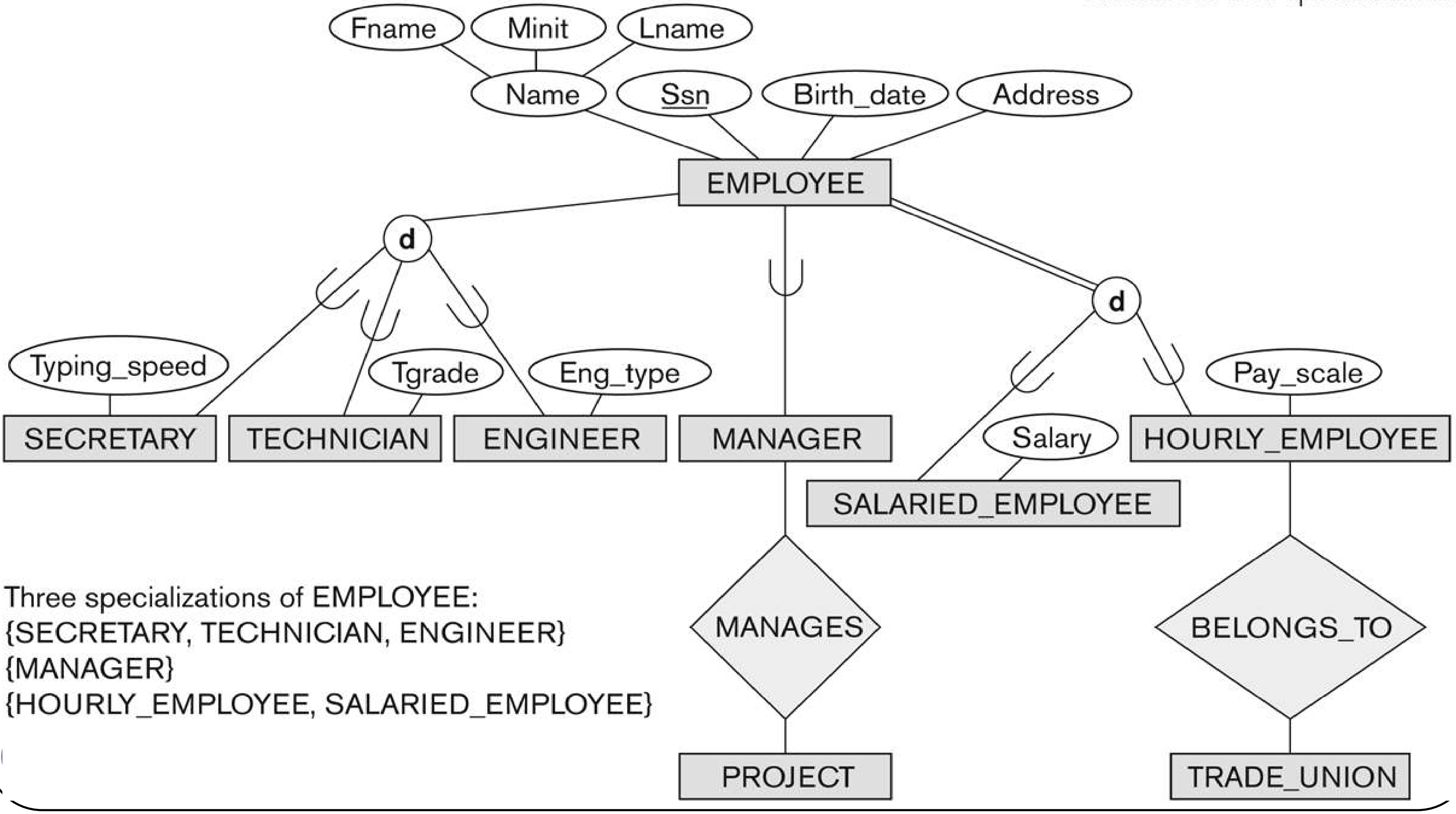
  - Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams

  - Attributes of a subclass are called ***specific*** or ***local*** attributes.
    - For example, the attribute TypingSpeed of SECRETARY

  - The subclass can also participate in specific relationship types.
    - For example, a relationship BELONGS_TO of HOURLY_EMPLOYEE.

Specialization ………..Cont.

Figure 4.1
EER diagram notation to represent subclasses and specialization.



Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Generalization

- Generalization is the reverse of the specialization process
- Several classes with common features are generalized into a superclass;
  - original classes become its subclasses

- **Example:** CAR, TRUCK generalized into ***VEHICLE***;
  - both CAR, TRUCK become subclasses of the superclass VEHICLE.
  - We can view {CAR, TRUCK} as a specialization of ***VEHICLE***
  - Alternatively, we can view ***VEHICLE*** as a generalization of CAR and TRUCK

Generalization………..Cont.

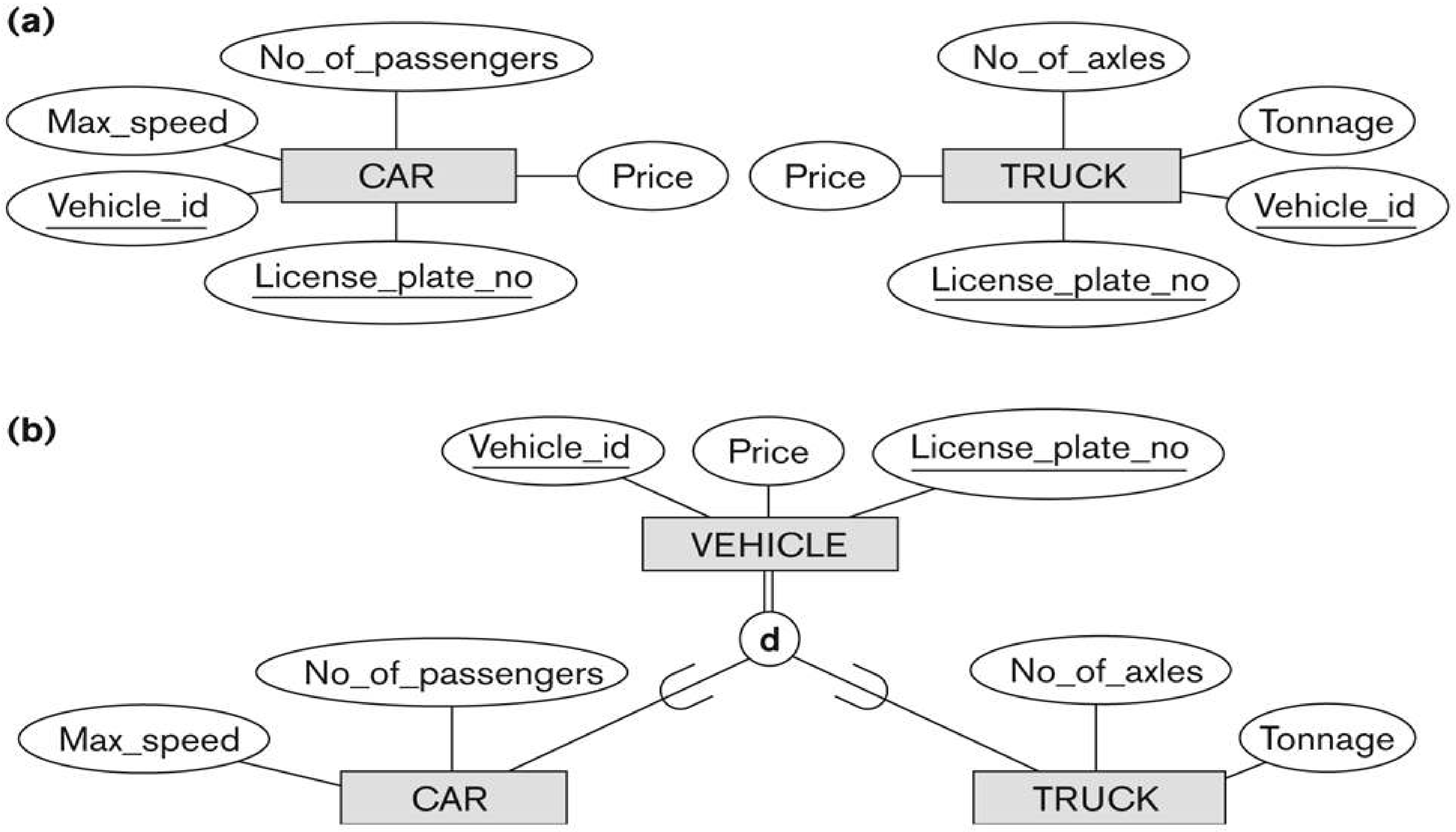


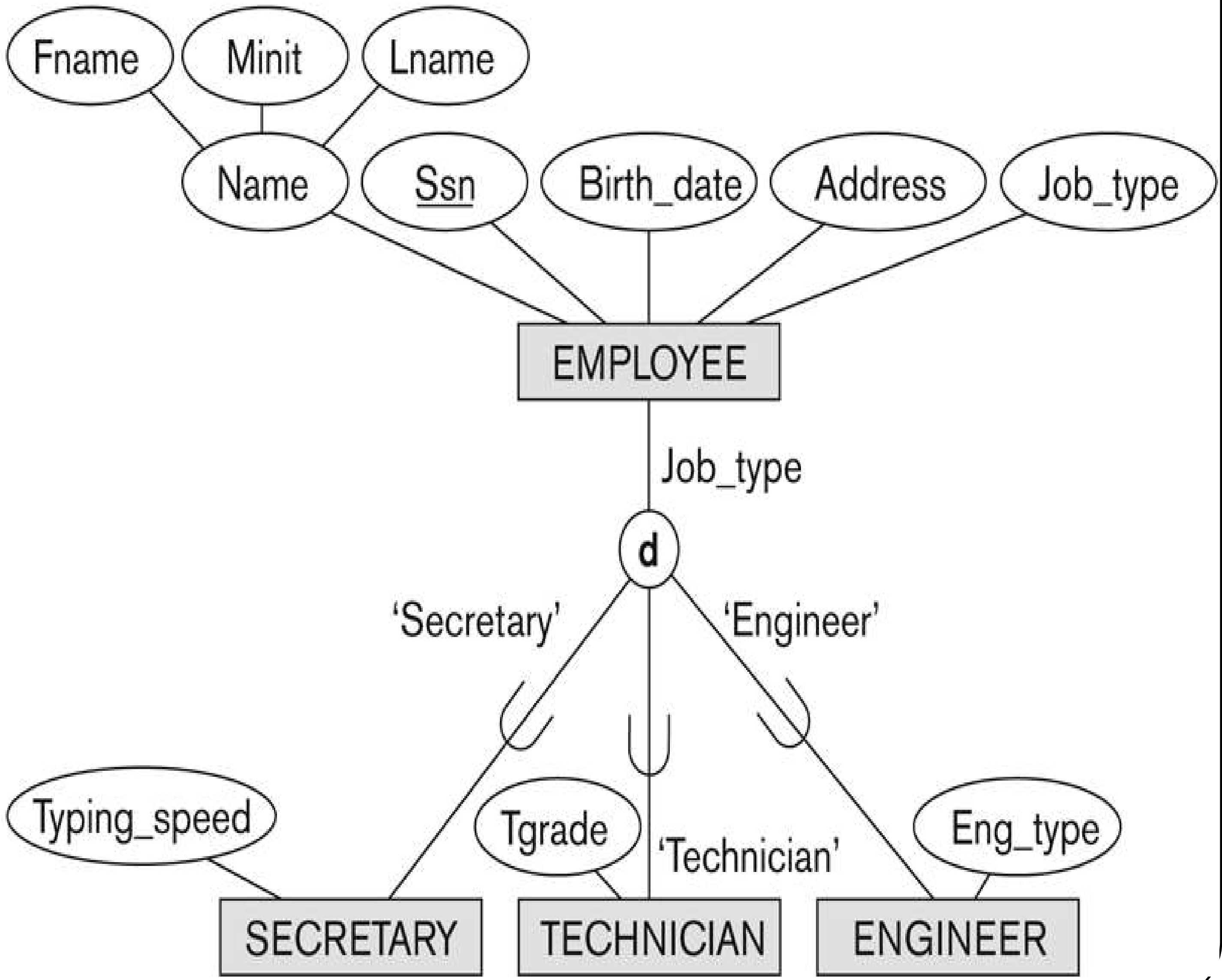
Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Constraints on Specialization and Generalization

- Two basic constraints can apply to a specialization/generalization:
  - **Disjointness** Constraint:
  - **Completeness** Constraint:
- ***Disjointness Constraint:***
  - Specifies that the subclasses of the specialization must be ***Disjoint***:
    - an entity can be a member of at most one of the subclasses of the specialization
  - Specified by ***d*** in EER diagram
  - If not disjoint, specialization is ***Overlapping*****:**
    - that is the same entity may be a member of more than one subclass of the specialization
  - Specified by ***o*** in EER diagram

Example of disjoint partial Specialization

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



Constraints on Specialization and Generalization ....Cont.

- ***Completeness Constraint:***
  - ***Total*** specifies that every entity in the superclass ***must*** be a member of some subclass in the specialization/generalization
  - Shown in EER diagrams by a ***double line***
  - ***Partial*** allows an entity ***not*** to belong to any of the subclasses
  - Shown in EER diagrams by a single line
- Hence, we have four types of specialization/generalization:
  - Disjoint, total
  - Disjoint, partial
  - Overlapping, total
  - Overlapping, partial
- **Note:** Generalization usually is total because the superclass is derived from the subclasses.

Example of overlapping total Specialization

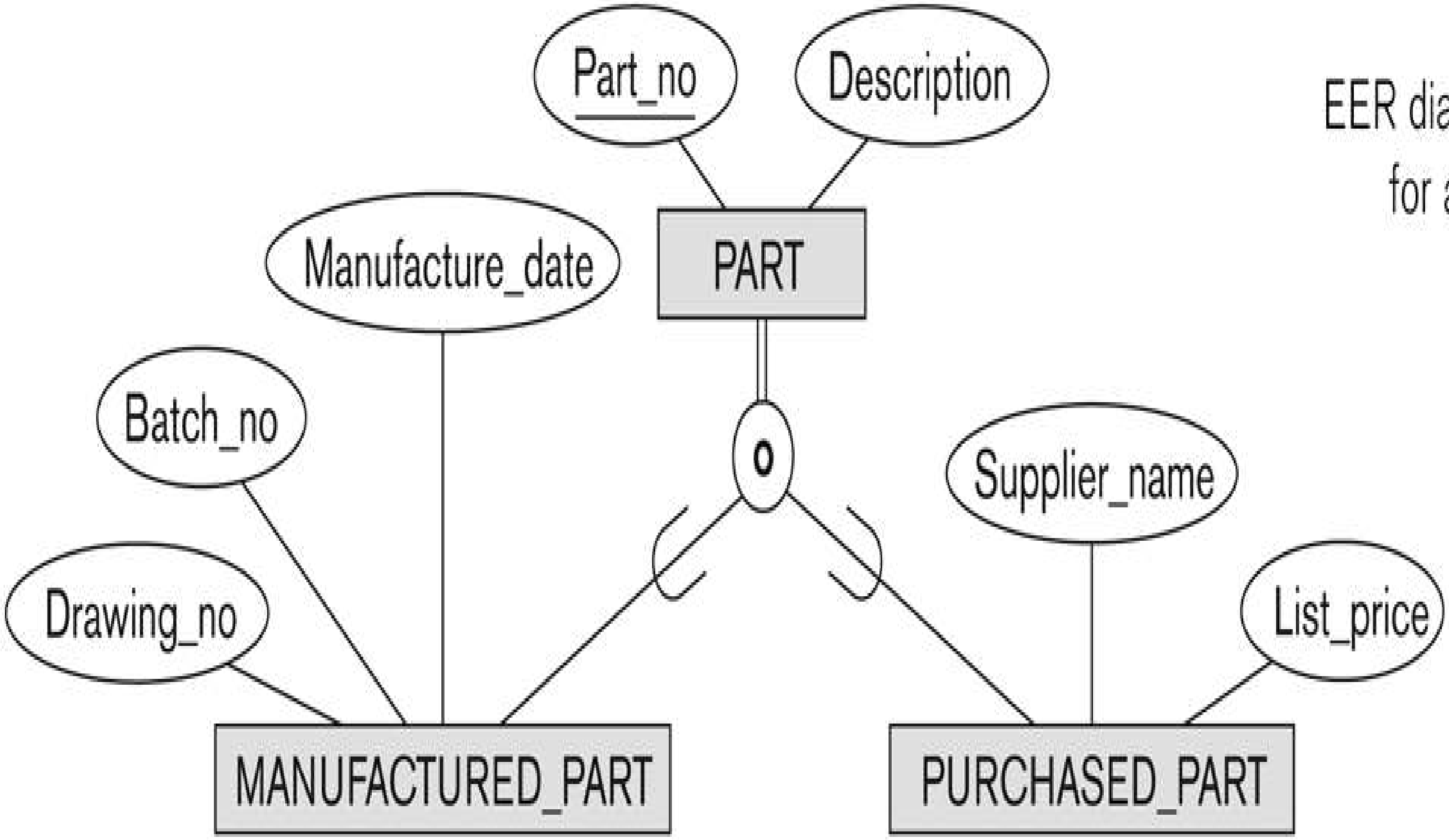


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses

- A subclass may itself have further subclasses specified on it
  - forms a ***hierarchy*** or a ***lattice***
- ***Hierarchy*** has a constraint that every subclass has only one superclass (called ***single inheritance***); this is basically a ***tree structure***
- In a ***lattice***, a subclass can be subclass of more than one superclass (called ***multiple inheritance***).
- In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses

Specialization / Generalization Lattice Example (UNIVERSITY)

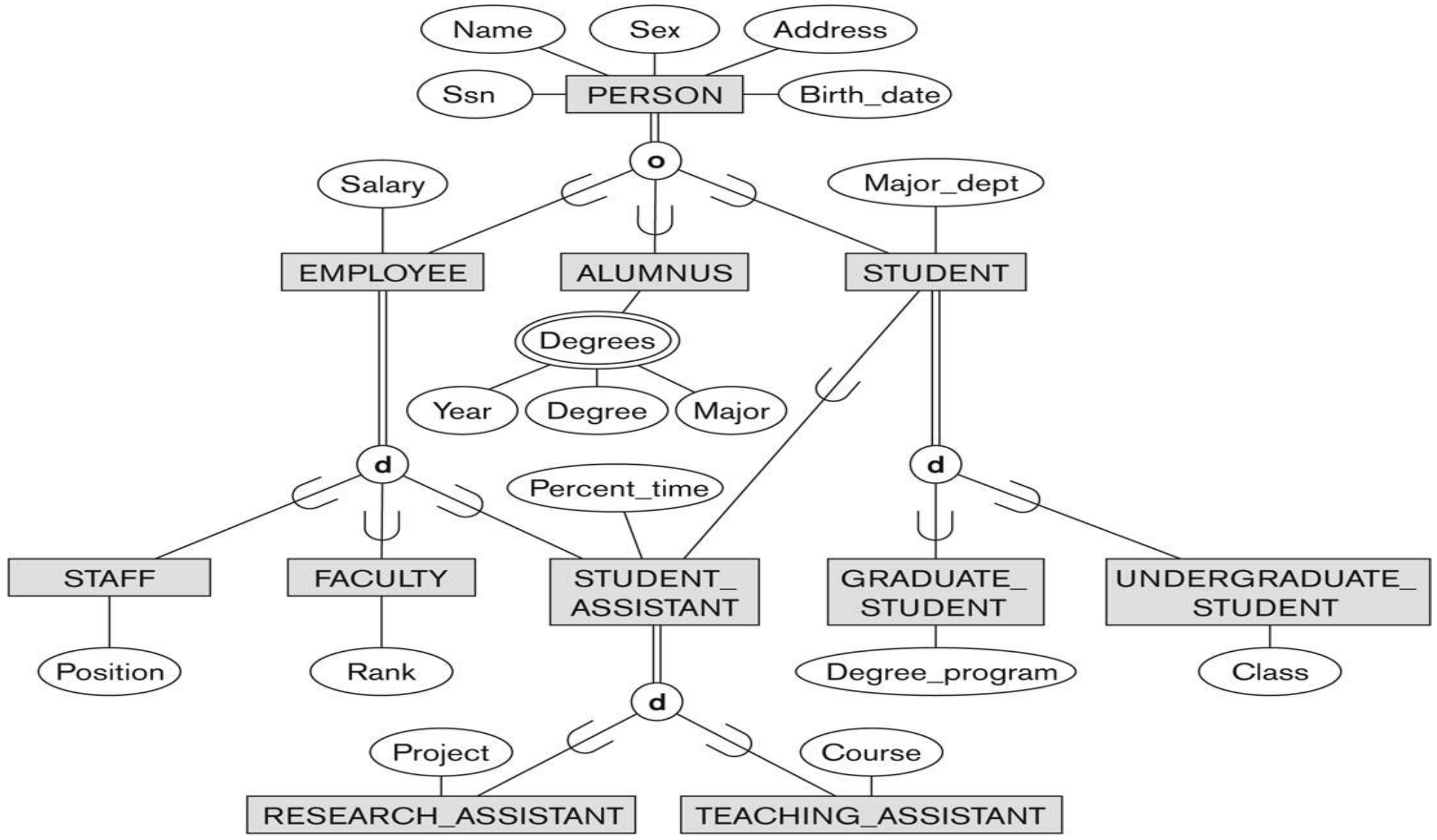


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses

- A subclass with more than one superclass is called a ***shared*** subclass (multiple inheritance).
- Can have:
  - specialization hierarchies or lattices, or
  - generalization hierarchies or lattices, depending on how they were derived
  - Shared Subclass "Engineering_Manager"

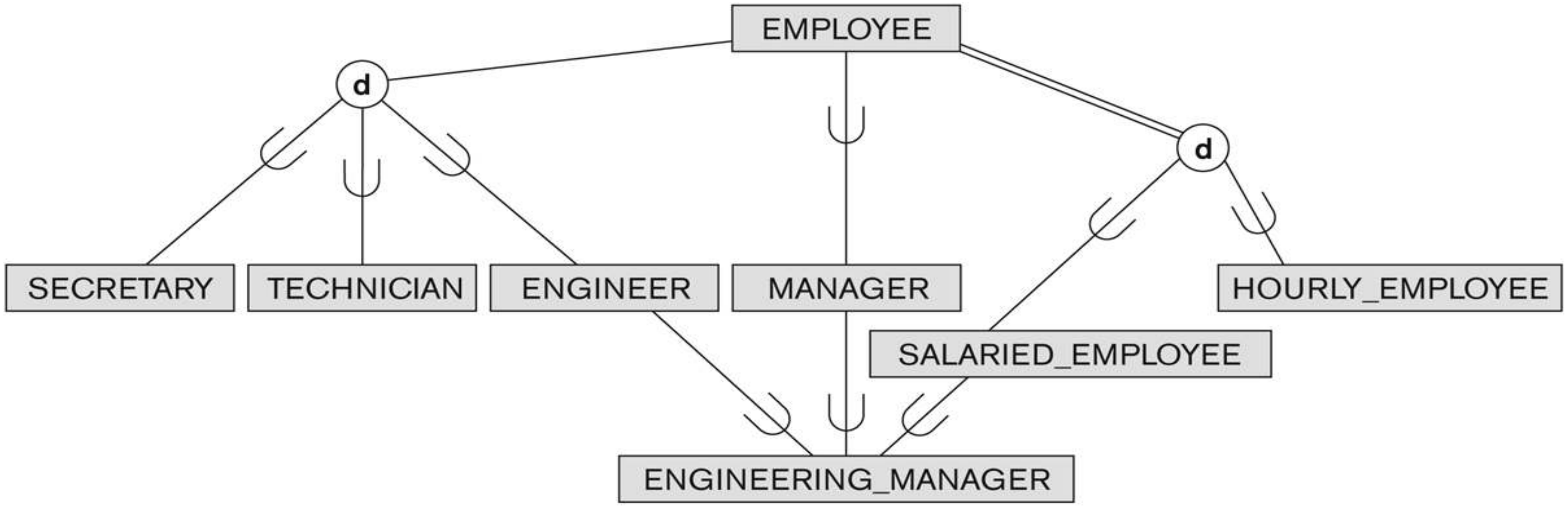


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

Categories (UNION TYPES)

- All of the *superclass/subclass relationships* we have seen thus far have a single superclass
- A ***shared*** subclass ($\cap$) is a subclass in:
  - *more than one* distinct superclass/subclass relationships
  - each relationships has a single superclass
  - shared subclass leads to multiple inheritance
- In some cases, we need to model a *single superclass/subclass relationship* with *more than one* superclass
- Superclasses can represent different entity types
- Such a subclass is called a category or ***UNION TYPE*** **($\cup$)**

Categories (UNION TYPES) …………Cont.

- **Example:** In a database for vehicle registration, a vehicle owner can be a PERSON, a BANK (holding a lien on a vehicle) or a COMPANY.
  - A *category* (UNION type) called OWNER is created to represent a subset of the *union* of the three superclasses COMPANY, BANK, and PERSON
  - A category member must exist in ***at least one*** of its superclasses ($\cup$)
- Difference from *shared subclass*, which is a:
  - subset of the ***intersection*** ***($\cap$)***of its superclasses
  - shared subclass member must exist in ***all*** of its superclasses

Two categories (UNION types): OWNER, REGISTERED_VEHICLE

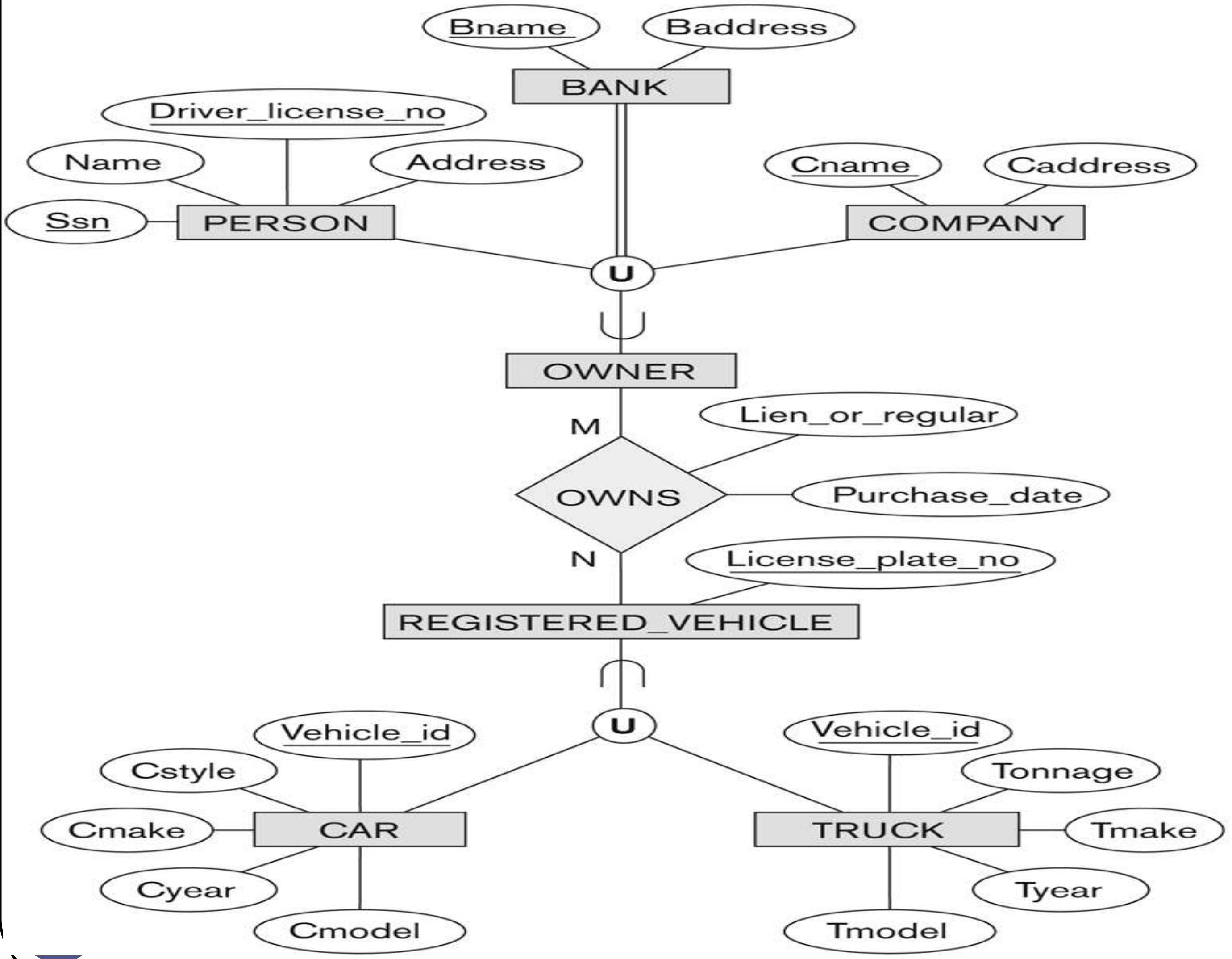


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Thank you