Relational Database Management System 116U01C403

Module 2 .1 & 2.2
Jan 2024-May 2024

Data Modeling: Enhanced-Entity-Relationship Model and Relational Data Model

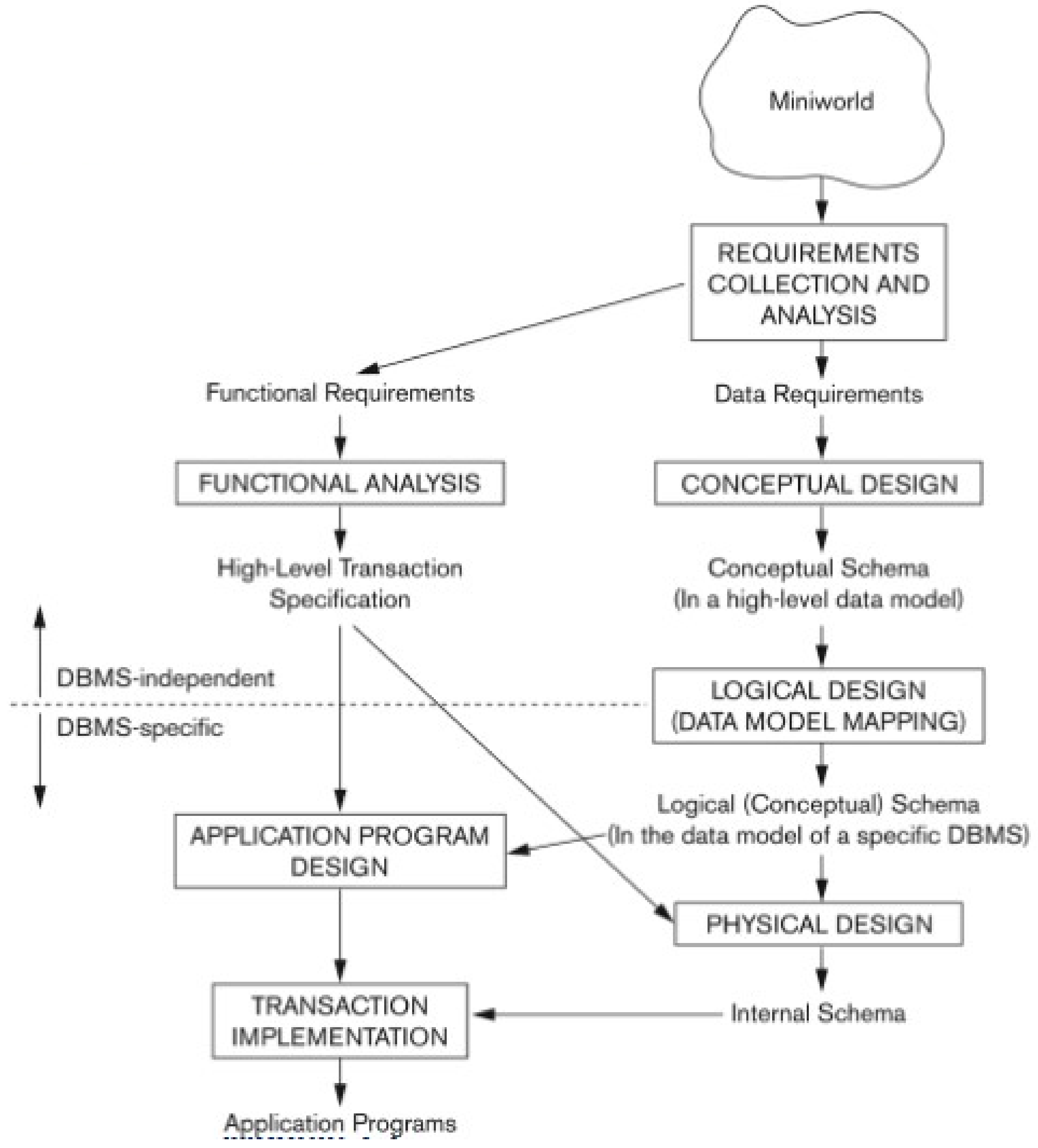
- Introduction, Benefits of Data Modeling, Types of Models, Phases of Database Modeling, The Entity-Relationship (ER) Model
- Enhanced -Entity-Relationship (EER)- Model Generalization, Specialization and Aggregation
- Relational Model: Introduction, Data Manipulation, Data Integrity, Advantages of the Relational Model
- Mapping EER Model to Relational Model

Overview of Database Design Process

- Two main activities:
- Database design:
- To design the conceptual schema for a database application

- Applications design:
- To focus on the programs and interfaces that access the database

Database design process



Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

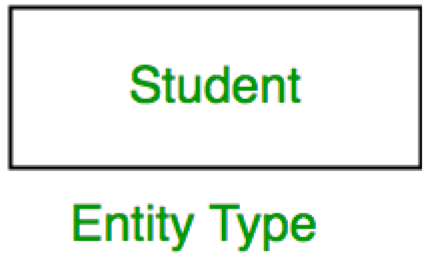
ER Model (Entity Relational Model)

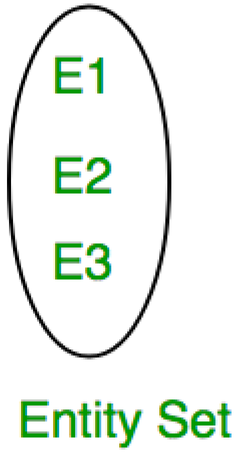
- The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related.
- The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

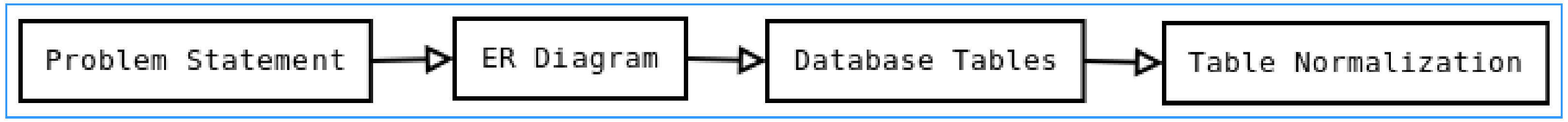
- **Why Use ER Diagrams In DBMS?**
- ER diagrams are used to represent the E-R model in a database, which makes them easy to be converted into relations (tables).
- ER diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- ER diagrams require no technical knowledge and no hardware support.
- These diagrams are very easy to understand and easy to create even for a naive user.
- It gives a standard solution for visualizing the data logically.

- **Entity**
- An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

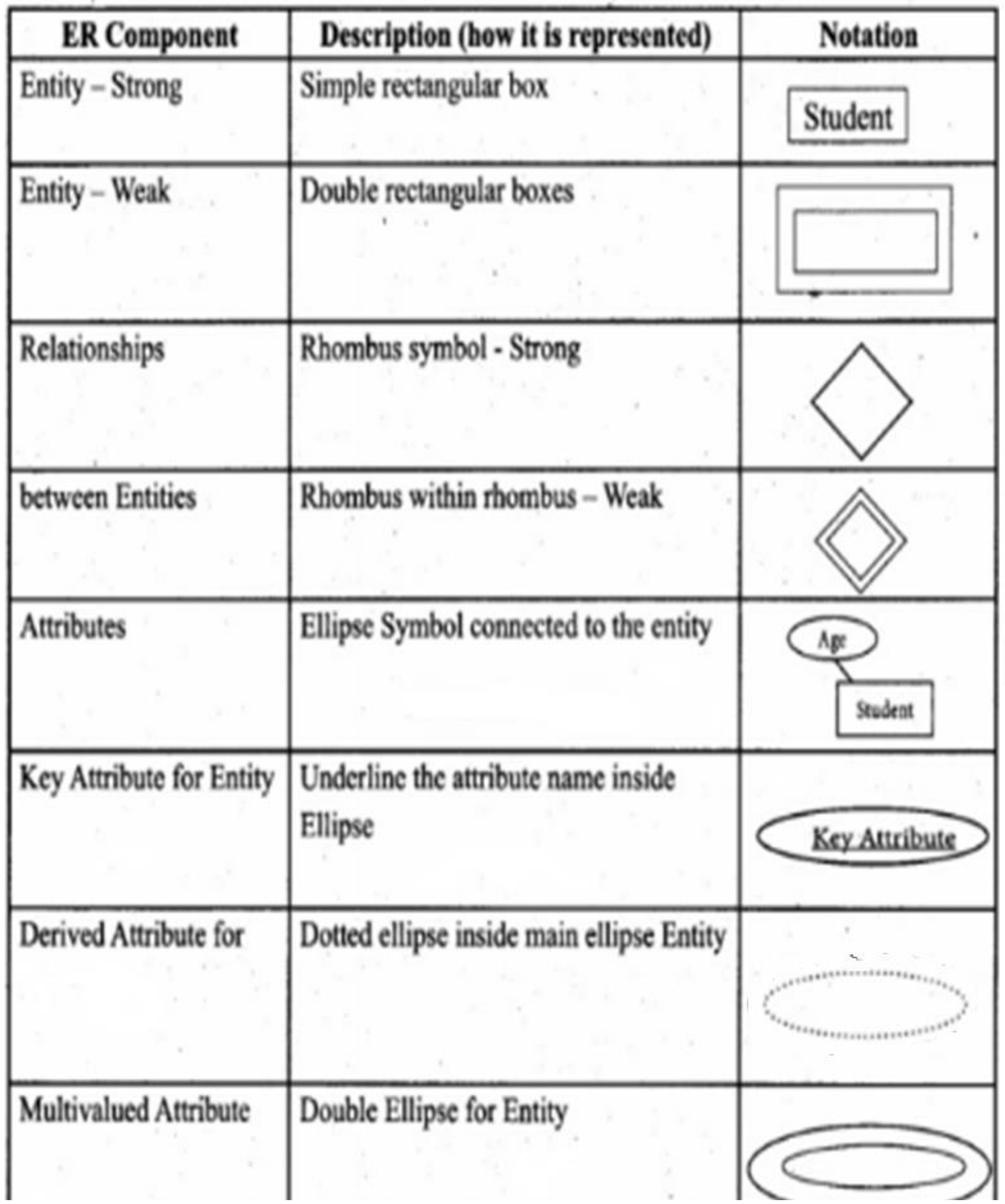
- **Entity Set:** An Entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set.





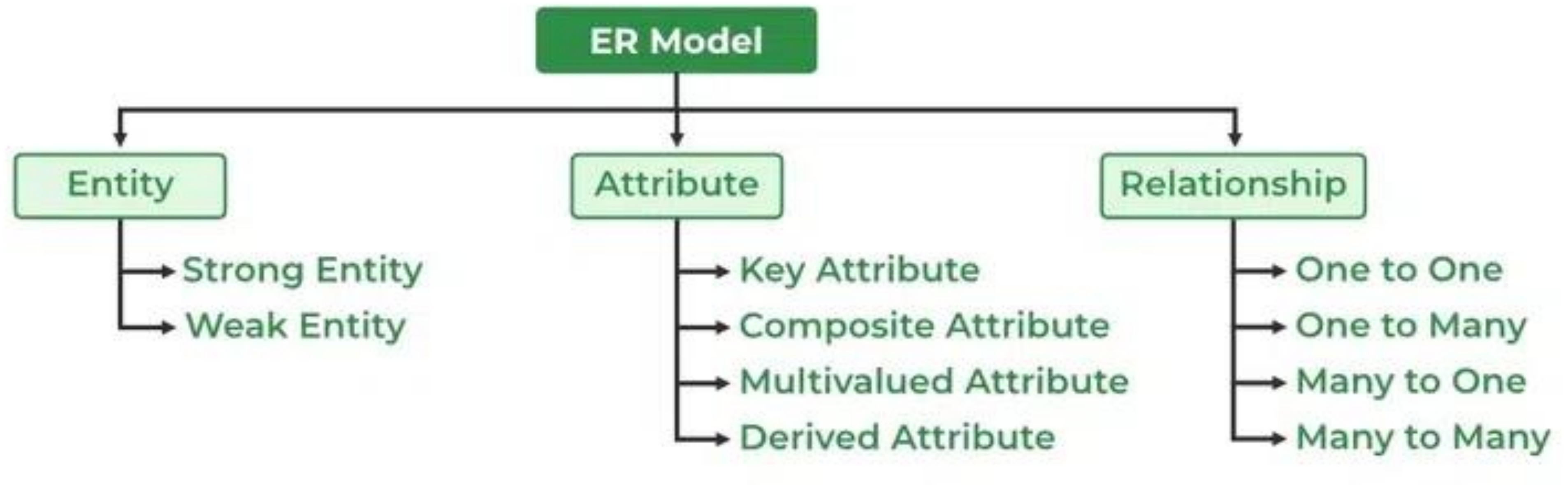


Symbols Used in ER Model

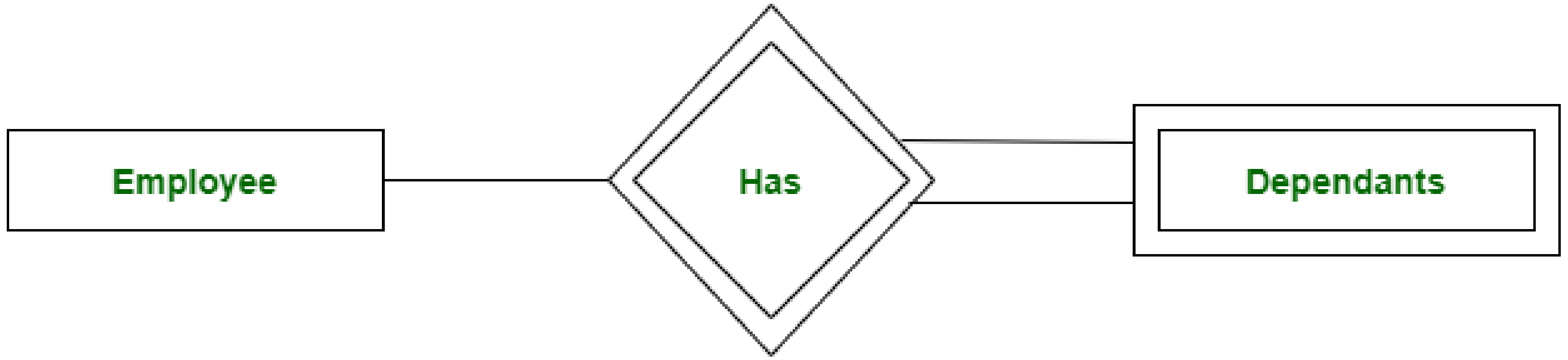


ER Component	Description (how it is represented)	Notation
Entity – Strong	Simple rectangular box	Student
Entity – Weak	Double rectangular boxes	
Relationships	Rhombus symbol - Strong	
between Entities	Rhombus within rhombus – Weak	
Attributes	Ellipse Symbol connected to the entity	Age Student
Key Attribute for Entity	Underline the attribute name inside Ellipse	Key Attribute
Derived Attribute for	Dotted ellipse inside main ellipse Entity	
Multivalued Attribute	Double Ellipse for Entity	

Components of ER Model

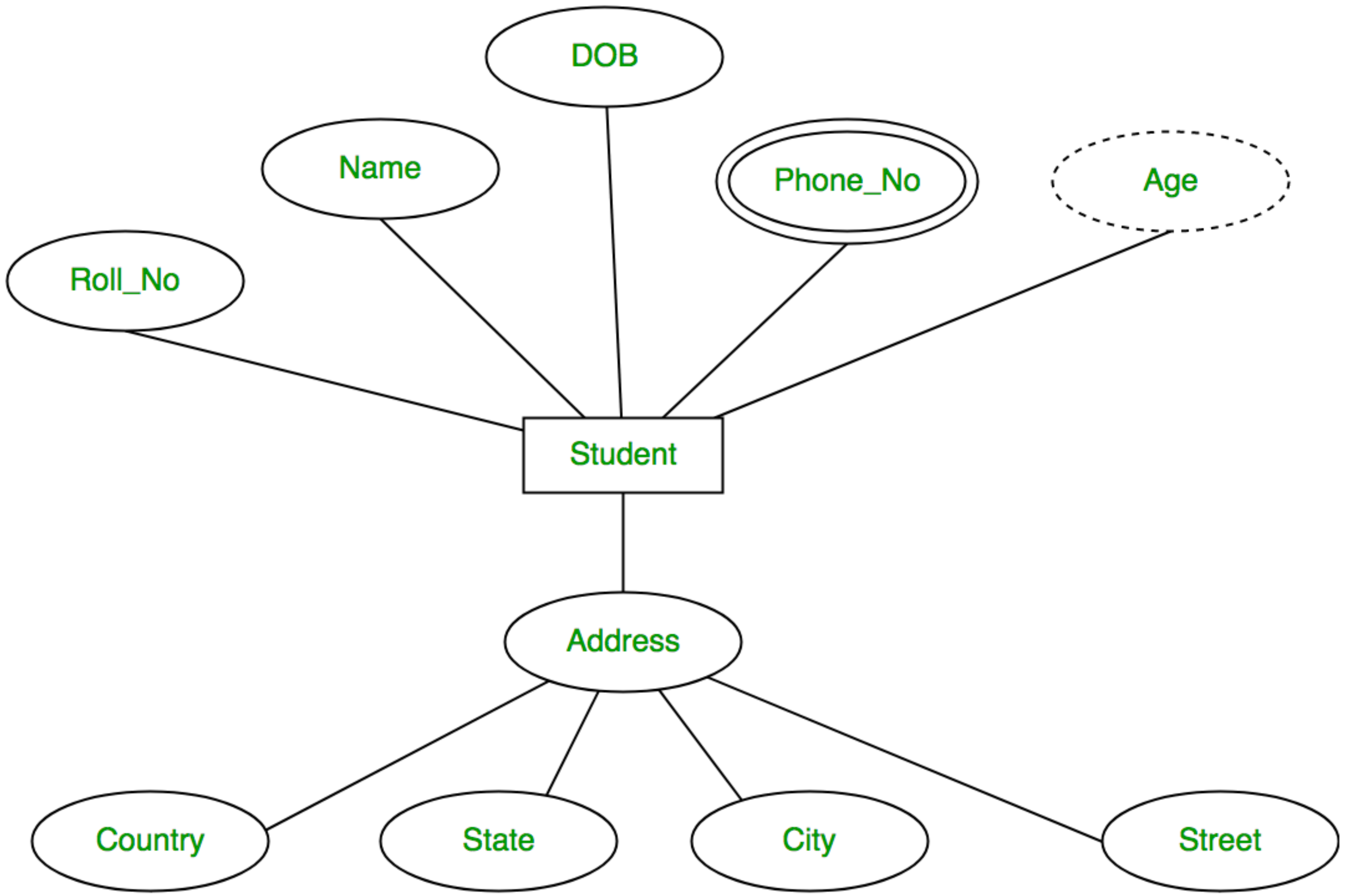


- **1. Strong Entity**
- A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These are called Strong Entity Types.
- **2. Weak Entity**
- An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can't be defined. These are called Weak Entity types.
- **For Example,** A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existed without the employee. So Dependent will be a **Weak Entity Type** and Employee will be Identifying Entity type for Dependent, which means it is **Strong Entity Type**.



Attributes

- Attributes are the properties that define the entity type. For example, Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student.
- **Key Attribute**
- The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. For example, Roll_No will be unique for each student.
- **Composite Attribute**
- An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.
- **Multivalued Attribute**
- An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student).
- **Derived Attribute**
- An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB).


DOB
Name
Phone_No
Age
Roll_No
Student
Address
Country
State
City
Street

Example COMPANY Database

- We need to create a database schema design based on the following (simplified) requirements of the COMPANY Database:
  - 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.

- We store each EMPLOYEE’s social security number, address,salary, gender, and birthdate.
  - Each employee works for one department but may work on several projects.
  - We keep track of the number of hours per week that an employee currently works on each project.
- We also keep track of the direct supervisor of each employee.
  - Each employee may have a number of DEPENDENTs.
  - For each dependent, we keep track of their name, gender, birthdate, and relationship to the employee.

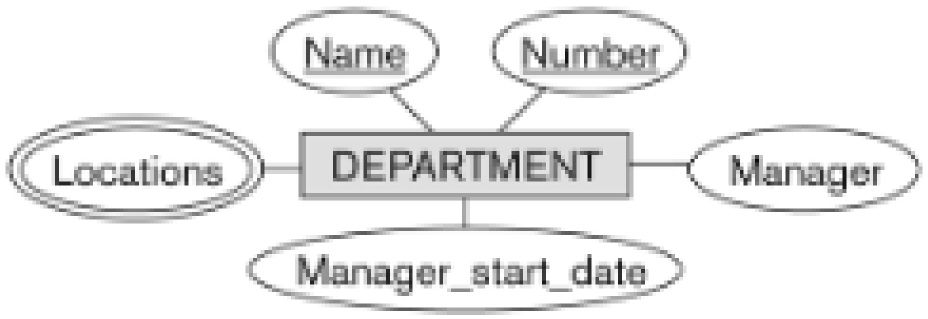
ER Model Concepts

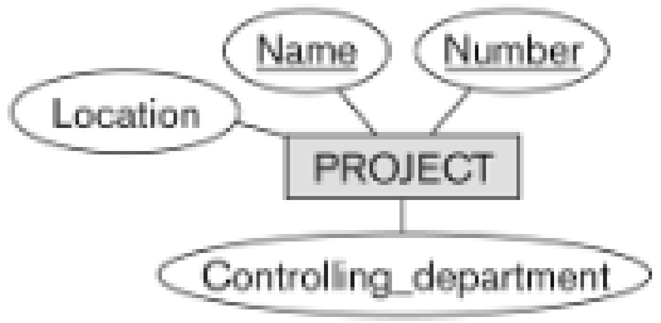
- **Entities and Attributes**
  - Entities(noun) are specific objects or things in the mini-world that are represented in the database.
    - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
  - Attributes are properties used to describe an entity.
    - For example an EMPLOYEE entity may have the attributes Name, SSN,Address, gender, BirthDate
  - A specific entity will have a value for each of its attributes.
    - For example a specific employee entity may have Name='John Smith’, SSN='123456789', Address ='731, Fondren, Houston, TX’, gender='M’, BirthDate='09-JAN-55‘
  - Each attribute has a value set (or data type) associated with it – e.g. integer, string, subrange, enumerated type, …

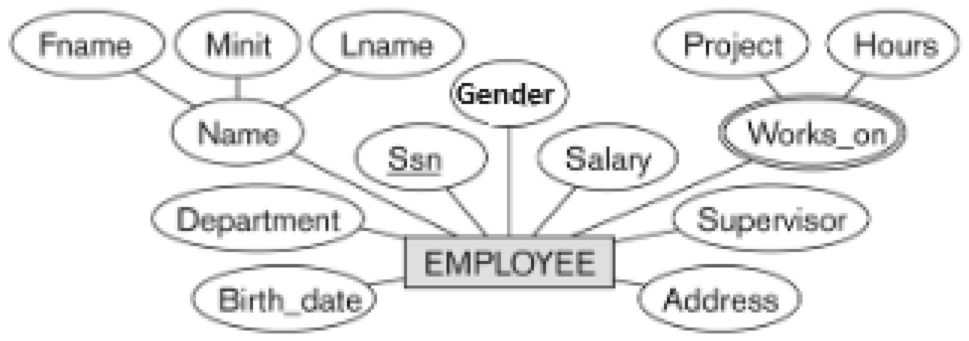
Initial Design of Entity Types for the COMPANY Database Schema

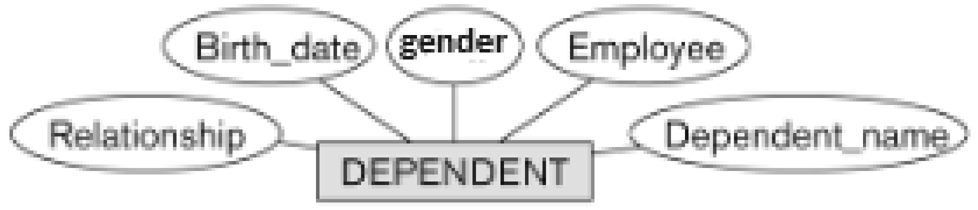
- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT
- The initial attributes shown are derived from the requirements description

Initial Design of Entity Types:









Relationships and Relationship Types

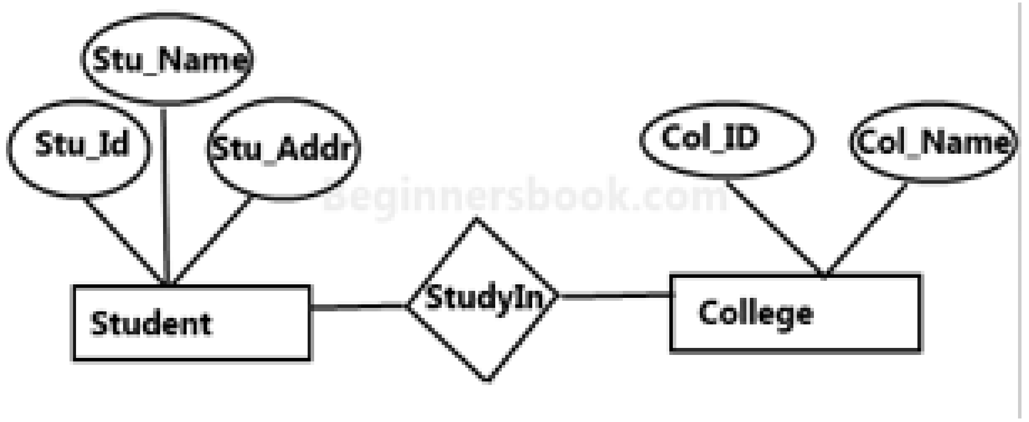
- A relationship relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith works on the ProductX PROJECT, or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a relationship type.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
  - Both MANAGES and WORKS_ON are binary relationships.

Relationship Types

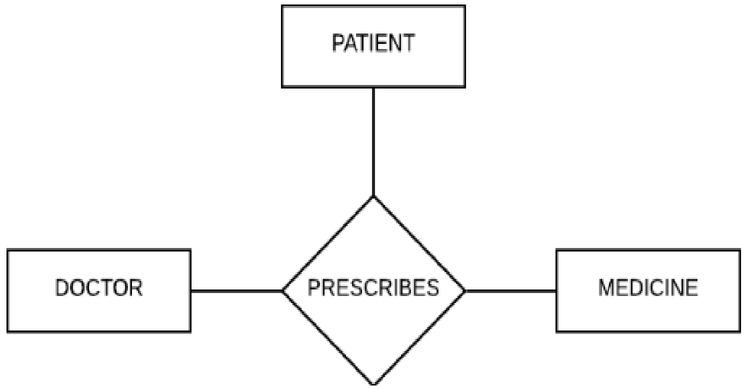
There are different types of Relationships like:

- Binary Relationship
- Ternary Relationship
- N-Ary Relationship
- Recursive Relationship

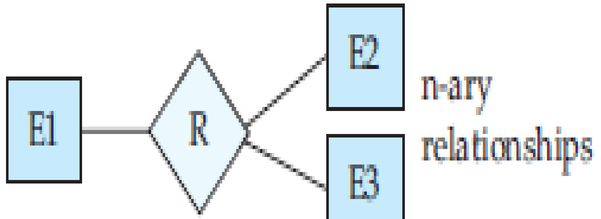
- **Binary Relationship**:
- When there is a relationship between two different entities, it is known as a binary relationship.



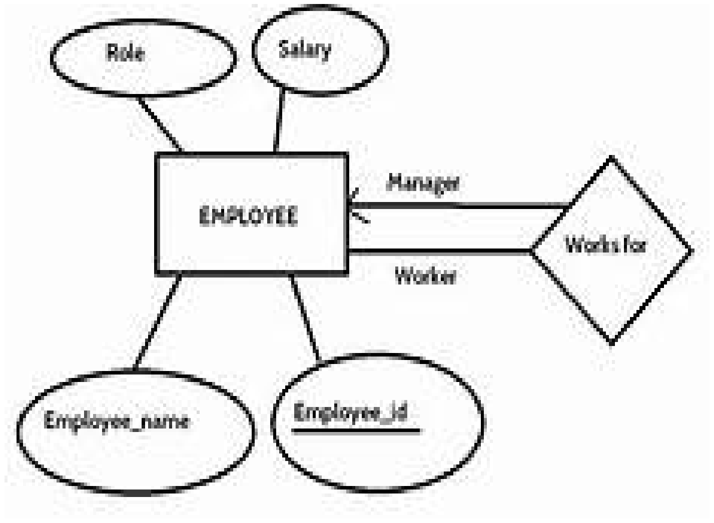
- **Ternary Relationship**:
- When there is a relationship between three different entities, it is known as a ternary relationship.



- **N-Ary Relationship**: When any relationship consists of more than two Entities then it is called N-ary relationship.

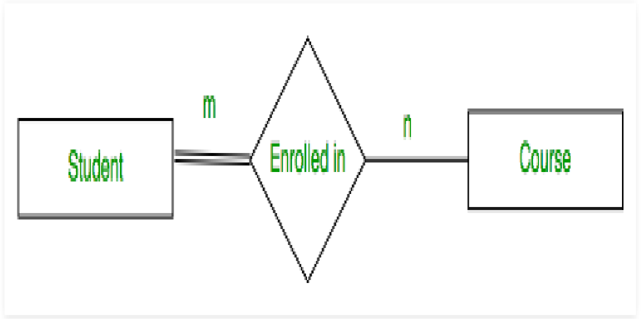


- **Recursive Relationship**
- A relationship between two entities of similar entity type is called a recursive relationship.
- Here the same entity type participates more than once in a relationship type with a different role for each instance.
- We indicate roles in E-R diagrams by labeling the lines that connect diamonds to rectangles.



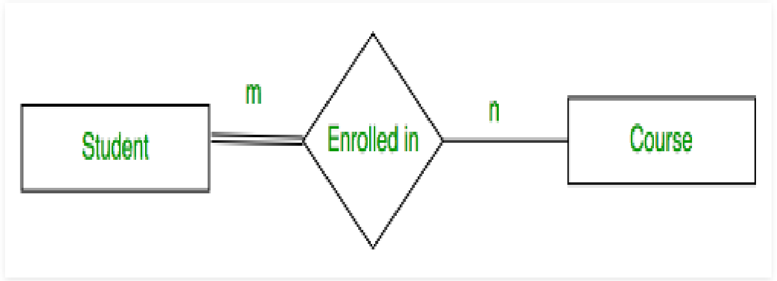
Participation Constraint

- Participation Constraint is applied on the entity participating in the relationship set.
- **Total Participation**:
- Each entity in the entity set must participate in the relationship.
- If each student must enroll in a course, the participation of student will be total.
- Total participation is shown by double line in ER diagram.



Partial Participation

- The entity in the entity set may or may NOT participate in the relationship.
- If some courses are not enrolled by any of the student, the participation of course will be partial.
- The diagram depicts the ‘Enrolled in’ relationship set with Student Entity set having total participation and Course Entity set having partial participation.

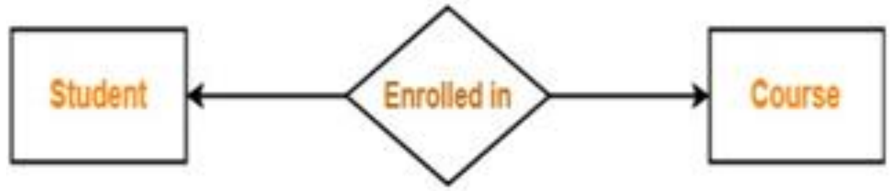


Mapping Cardinality/Mapping Constraints

- Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:
- One-To-One
- One-To-Many
- Many-To-One
- Many-To-Many

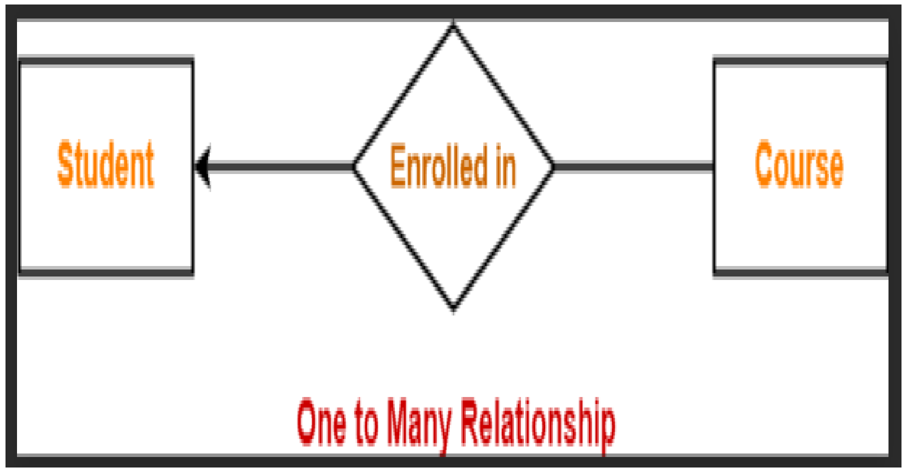
One-To-One

- An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.
- E.g. Student can enroll at most one course and course can have at most one student.
- E.g. We draw a directed line from the relationship set advisor to both entity sets instructor and student. This indicates that an instructor may advise at most one student, and a student may have at most one advisor.



One to One Relationship

Mapping Cardinality

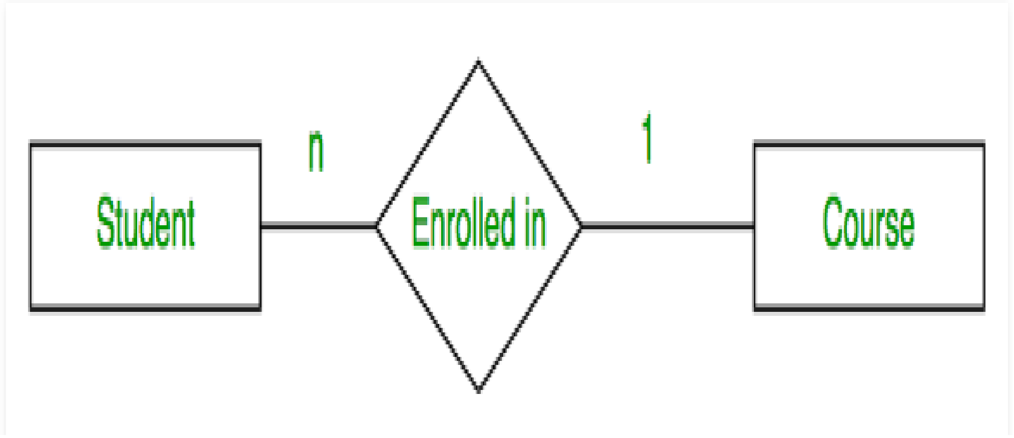


- **One-To-Many:** An entity in A is associated with any number (zero or more)of entities in B. An entity in B, however, can be associated with at most one entity in A.

- E.g. An instructor may advise many students, but a student may have at most one advisor.

- e.g. A student can enrol for many courses

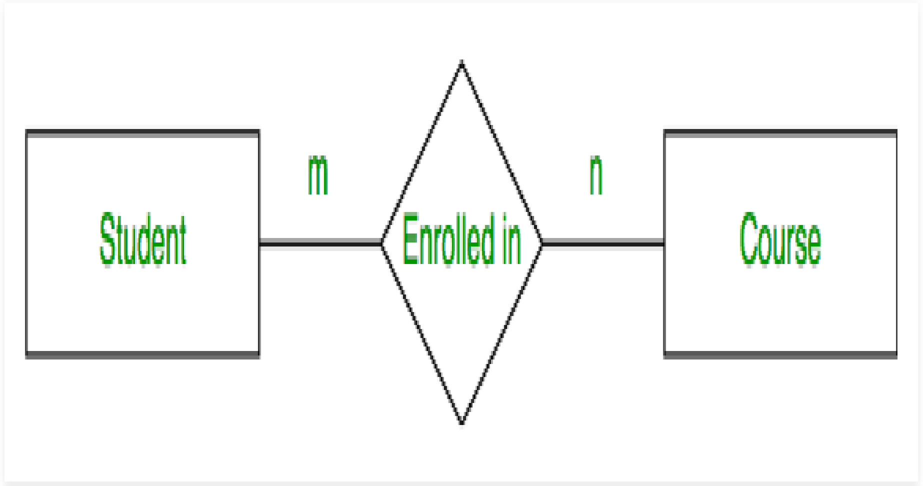
Many-To-One



- An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.

- This is another notation which shows student can take at most one course and one course can be taken by many students.

Many-To-Many



- An entity in A is associated with any number (zero or more)of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.
- An instructor may advise many students, and a student may have many advisors.
- This is another notation to show mapping cardinality. Which shows one student can take many courses and one course can be taken by many students.

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

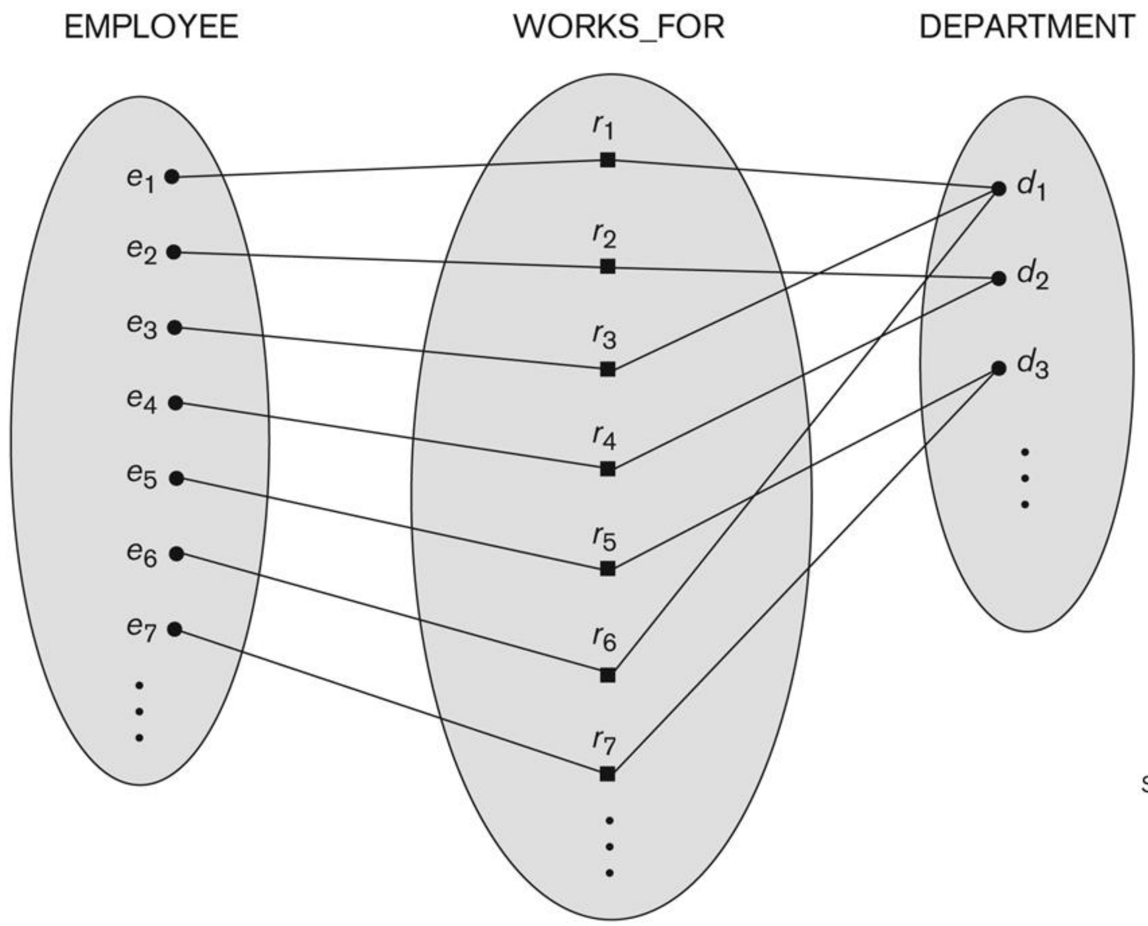


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

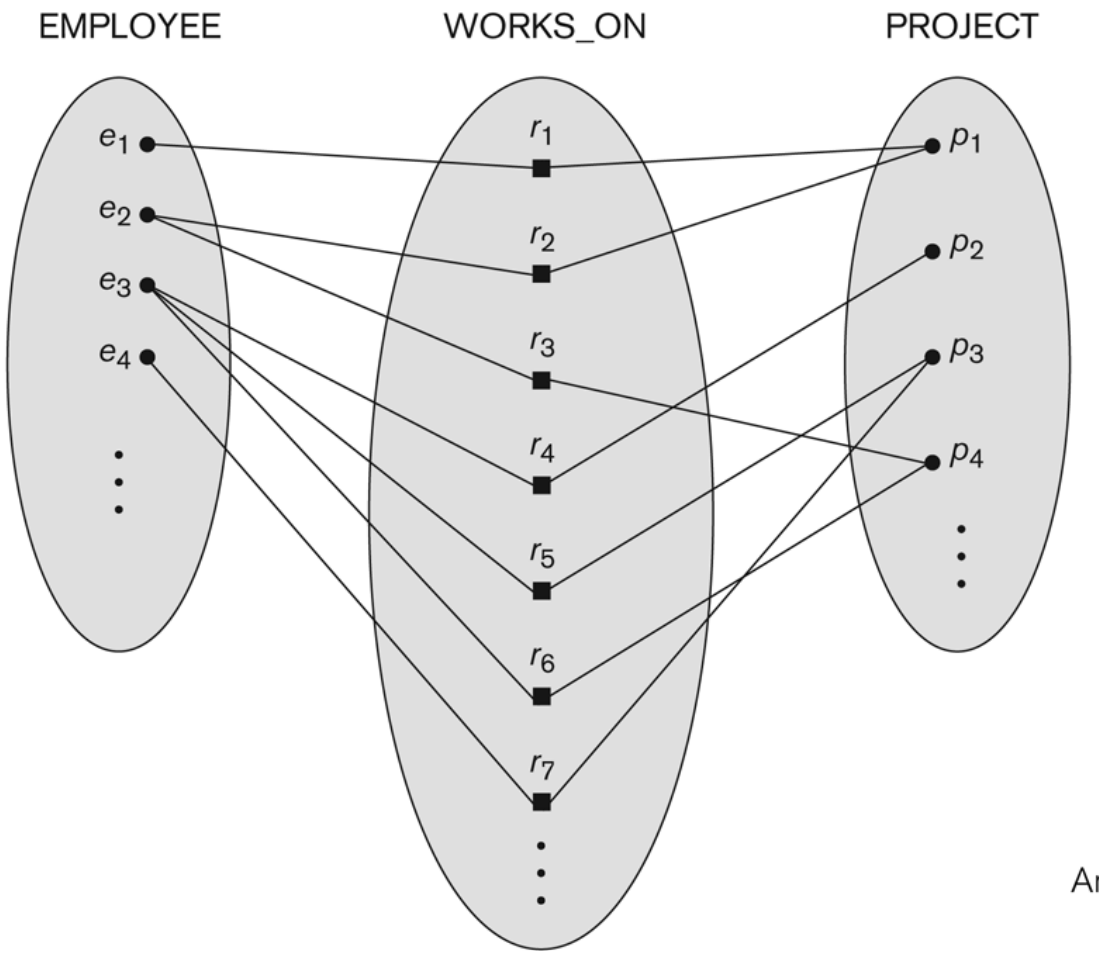


Figure 3.13
An M:N relationship, WORKS_ON.

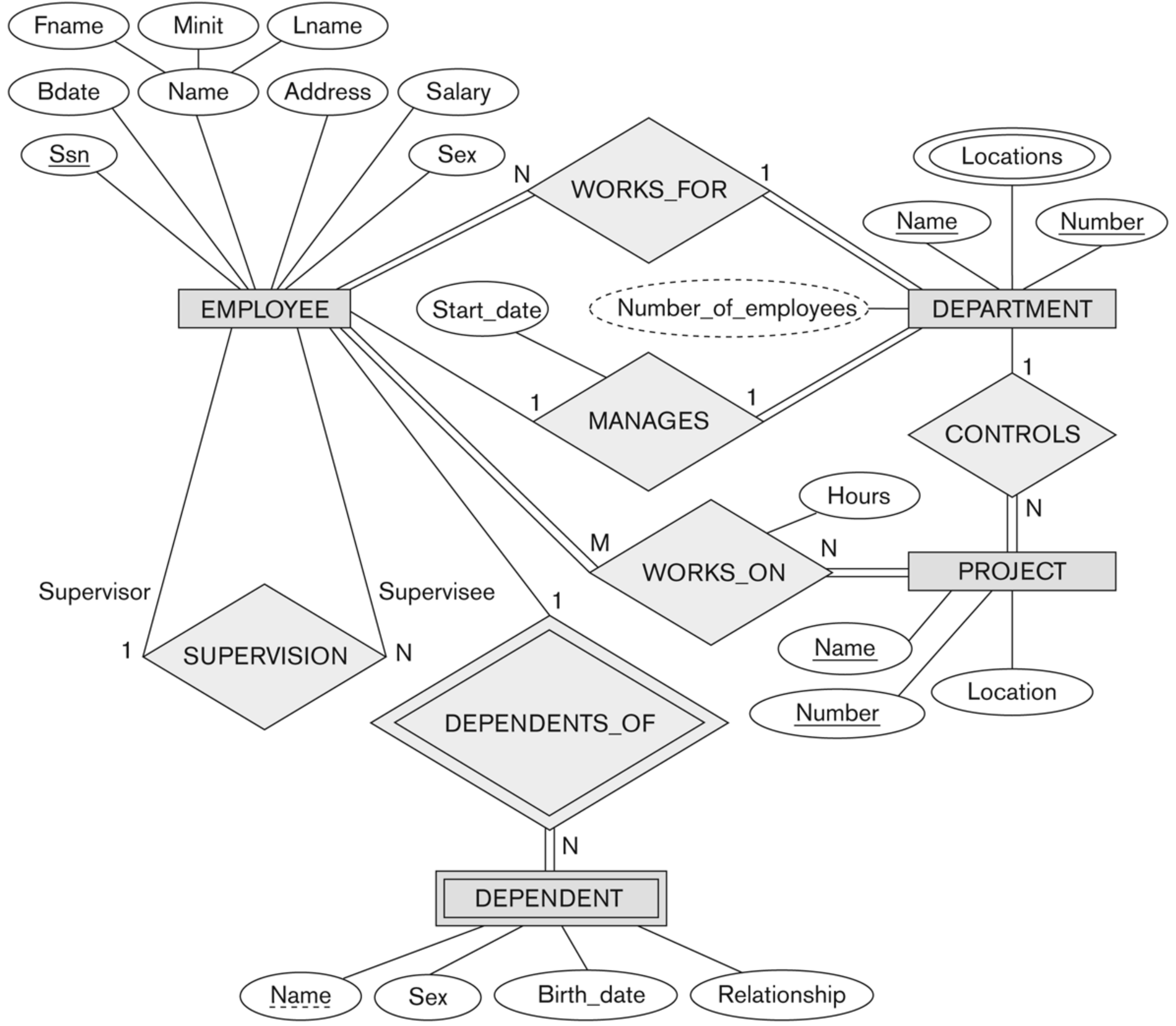
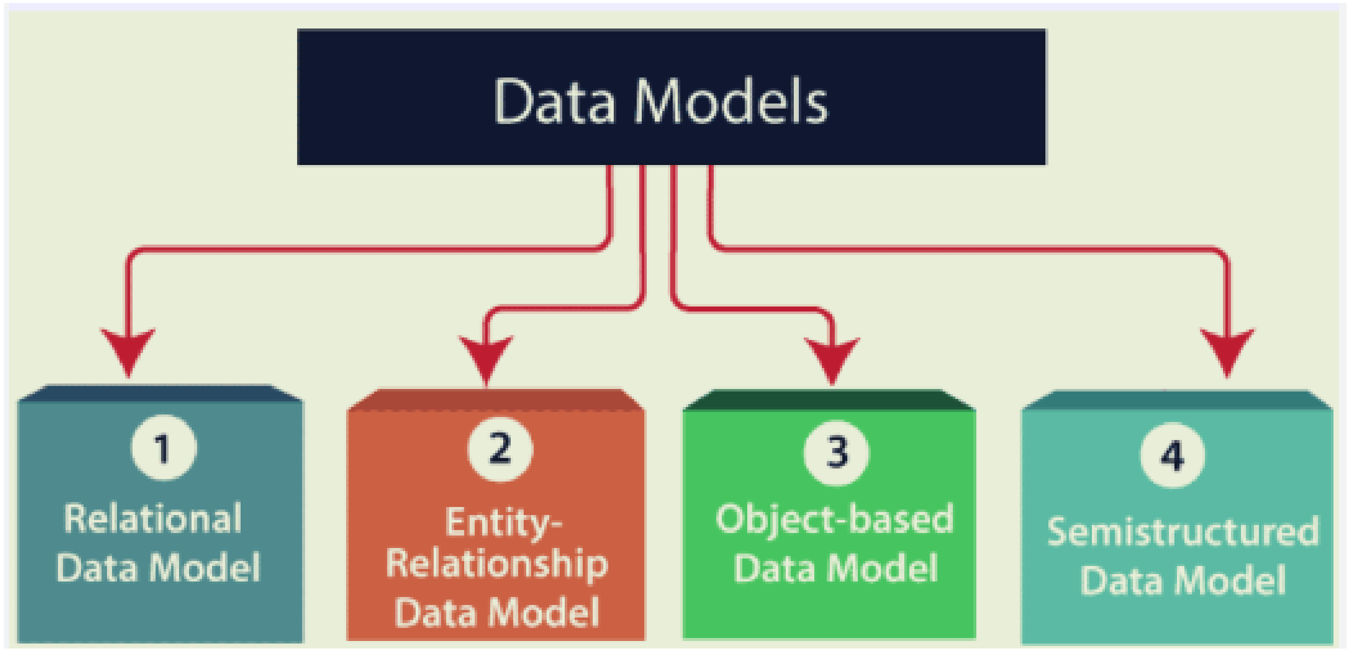


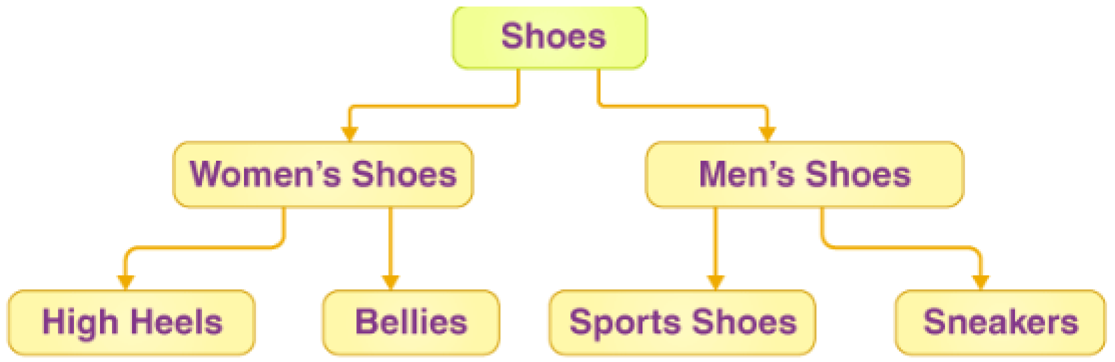
Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Types of Relational Models

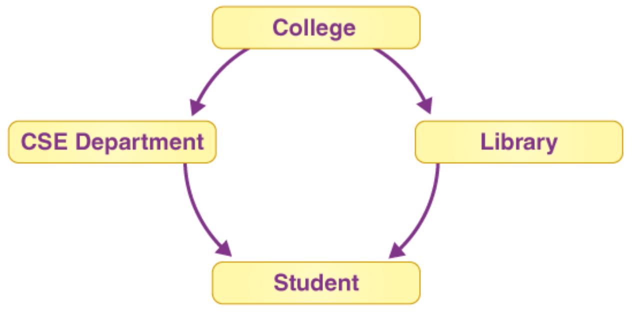


- Hierarchical Model
- Network Model
- Flat Data Model
- Associative Data Model
- Context Data Model

- Hierarchical Model
- In this, the data is organized into a tree-like structure where each record consists of one parent record and many children.

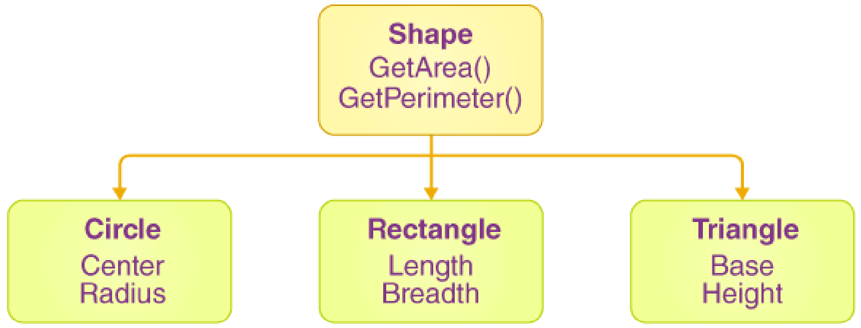


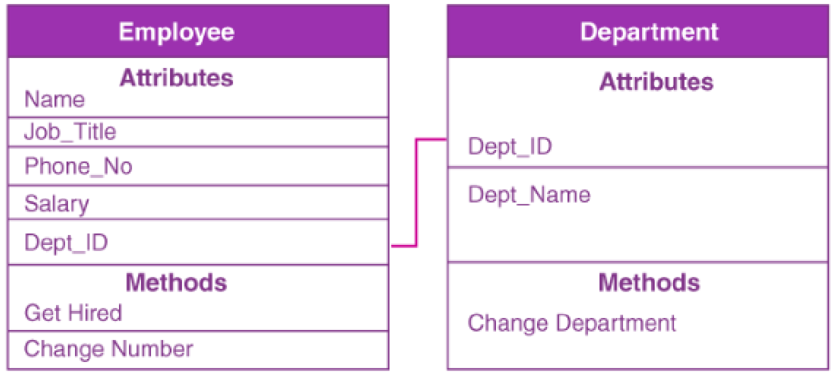
- Network Model
- Any record can have several parents in the network model. It uses a graph instead of a hierarchical tree.



Network Model

- Object-Oriented Data Model
- Both data and the data relationships are stored into a single structure that's known as an object in the object-oriented data model.





Object Oriented Model

- **Object-Relational Data Model**
- This model was developed to bridge the gap between the Object-Oriented and relational models.
- Many additional capabilities are available, such as the ability to create complicated data types based on our requirements utilizing existing data types.
- The issue with this paradigm is that it can become overly complicated and difficult to manage.

- **Flat Data Model**
- It’s a straightforward model in which the DB is depicted as a table with rows and columns.
- A flat data model is one in which all of the data is kept in the same plane.
- This data model has one drawback it cannot store a large amount of data that is the tables can not be of large size.

- **Semi-Structured Data Model**
- Semi-structured data refers to the structured data that doesn’t adhere to the tabular structure of the data models that are associated with relational DBs.

- **Associative Data Model**
- It is a model in which the data is separated into two sections.
- Everything that has its own existence is referred to as an entity, and the relationships between these entities are referred to as associations.
- Items and links are two types of data that are separated into two components.

- **Context data model**
- It is simply a data model which consists of more than one data model.
- For example, the Context data model consists of ER Model, Object-Oriented Data Model, etc. This model allows users to do more than one thing which each individual data model can do.

Enhanced Entity Relationship model

It is a diagrammatic technique for displaying the following concepts:

- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation

Subclasses and Superclasses

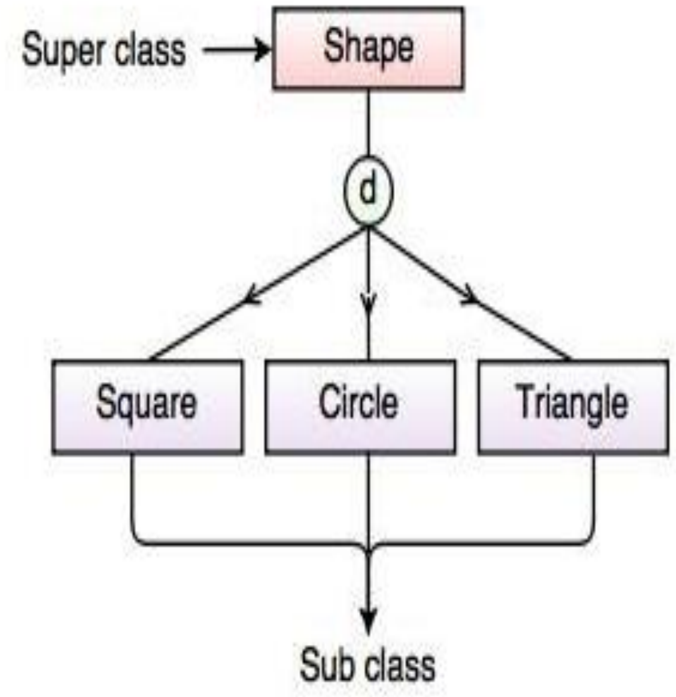
- An entity type may have additional meaningful subgroupings of its entities
    - Example: EMPLOYEE may be further grouped into:
        - SECRETARY, ENGINEER, TECHNICIAN, …
            - Based on the EMPLOYEE’s Job
        - MANAGER
            - EMPLOYEEs who are managers
        - SALARIED_EMPLOYEE,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

- Sub class and Super class relationship leads the concept of Inheritance.

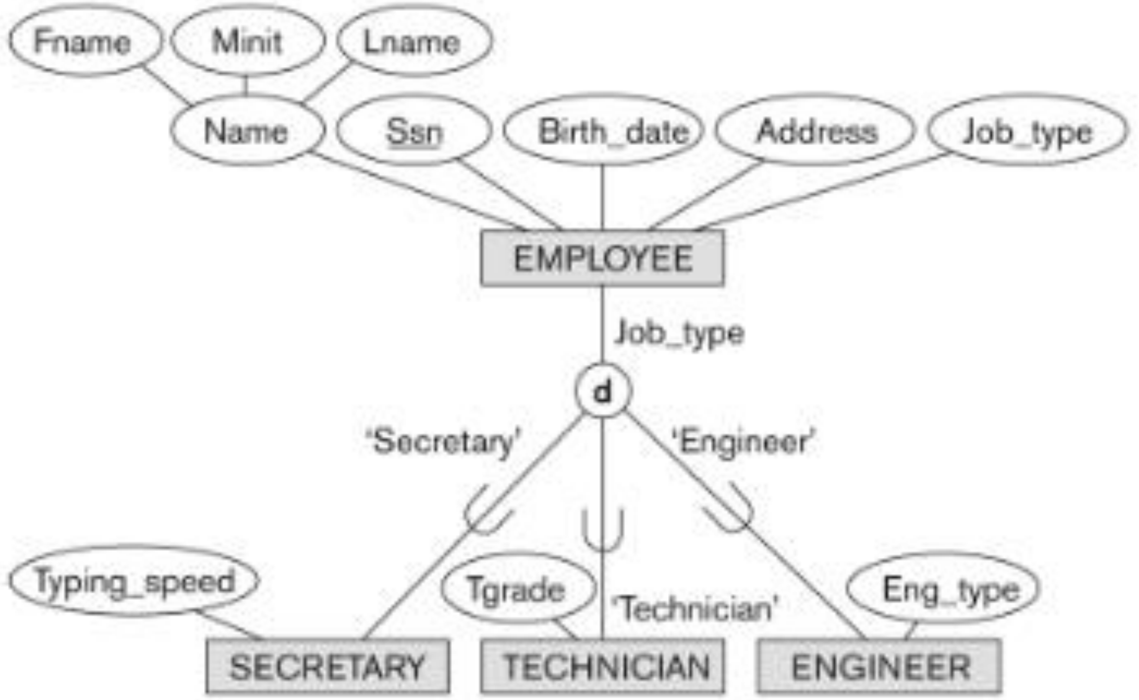
- **1. Super Class:** Super class is an entity type that has a relationship with one or more subtypes.

For example: Shape super class is having sub gr
Triangle.

- **2. Sub Class: It** is a group of entities with uni
- Sub class inherits properties and attribute
  For example: Square, Circle, Triangle are the
  class.



Representing Specialization in EER Diagrams



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 (1)

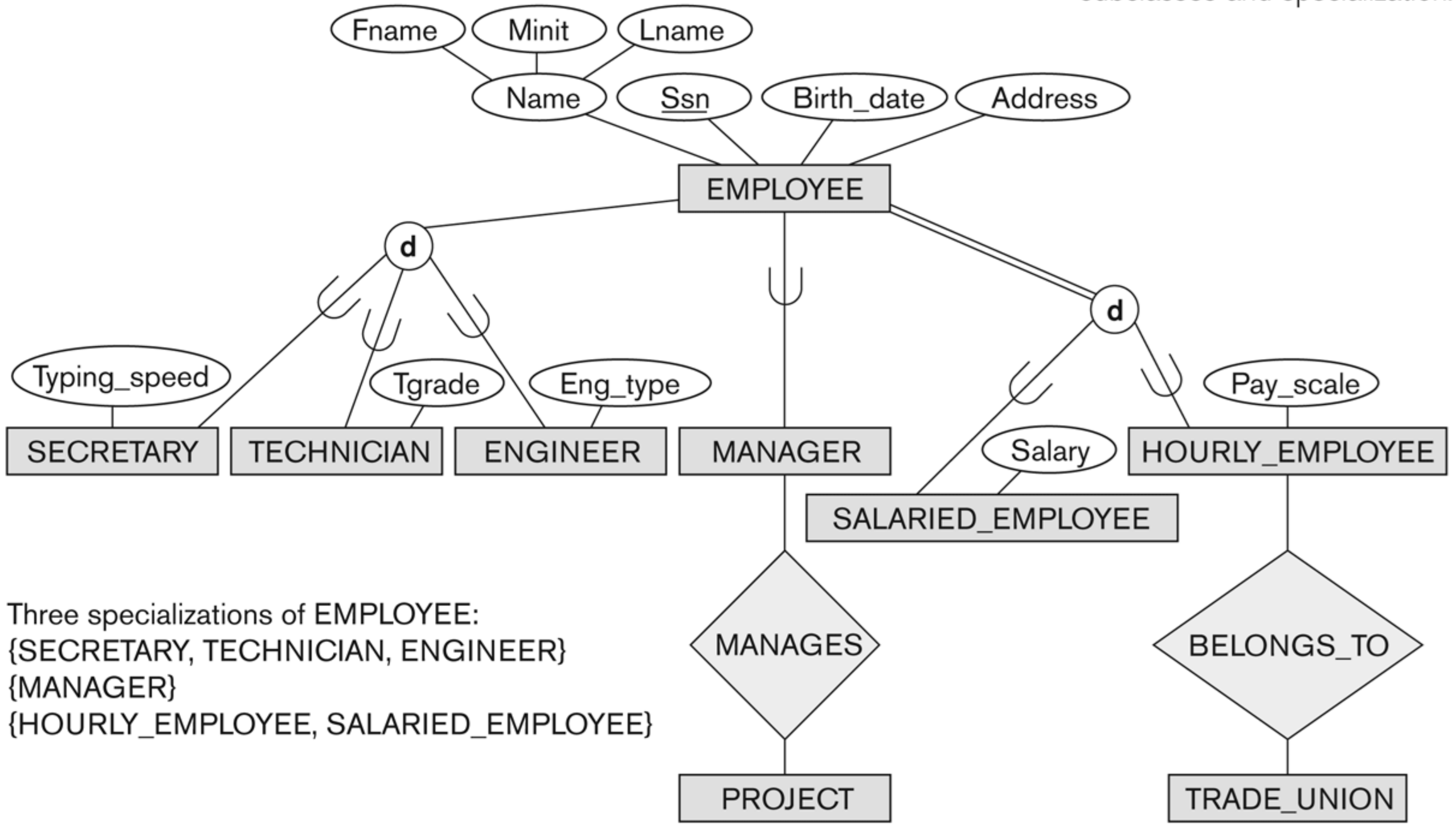
- Specialization is the process of defining a set of subclasses of a superclass
- The set of subclasses is based upon some distinguishing characteristics of the entities in the superclass
    - Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon *job type.*
        - May have several specializations of the same superclass

Specialization (2)

- 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

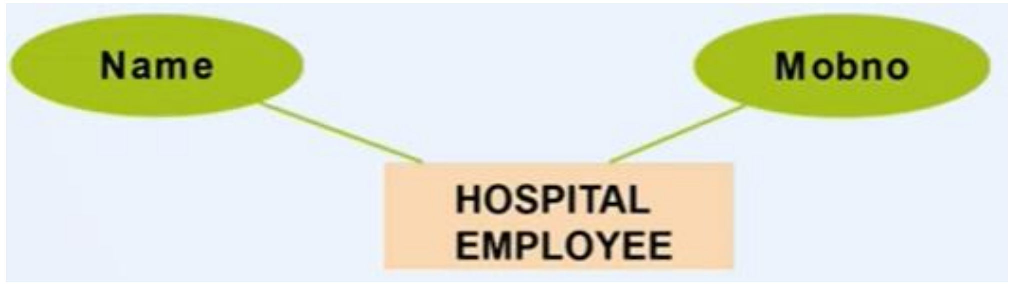
Specialization (3)

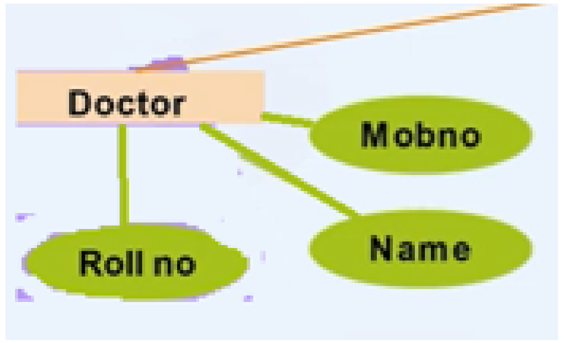
Figure 4.1
EER diagram notation to represent subclasses and specialization.

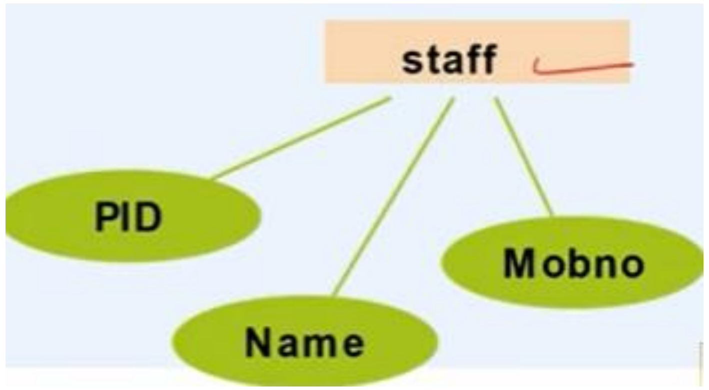


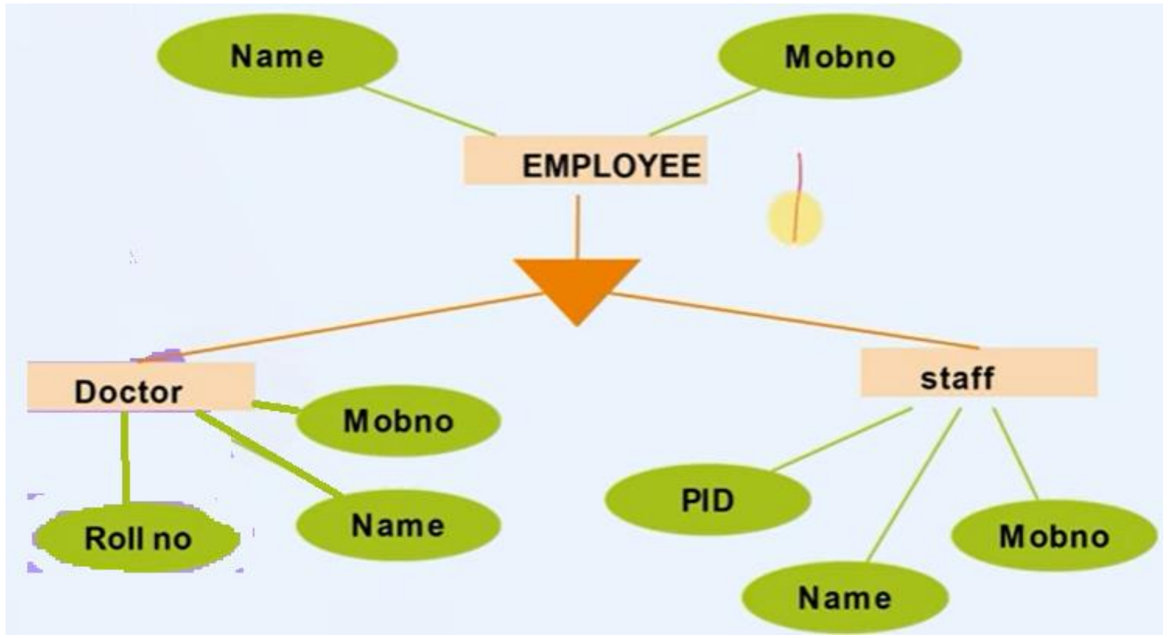
Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Example of Specialization



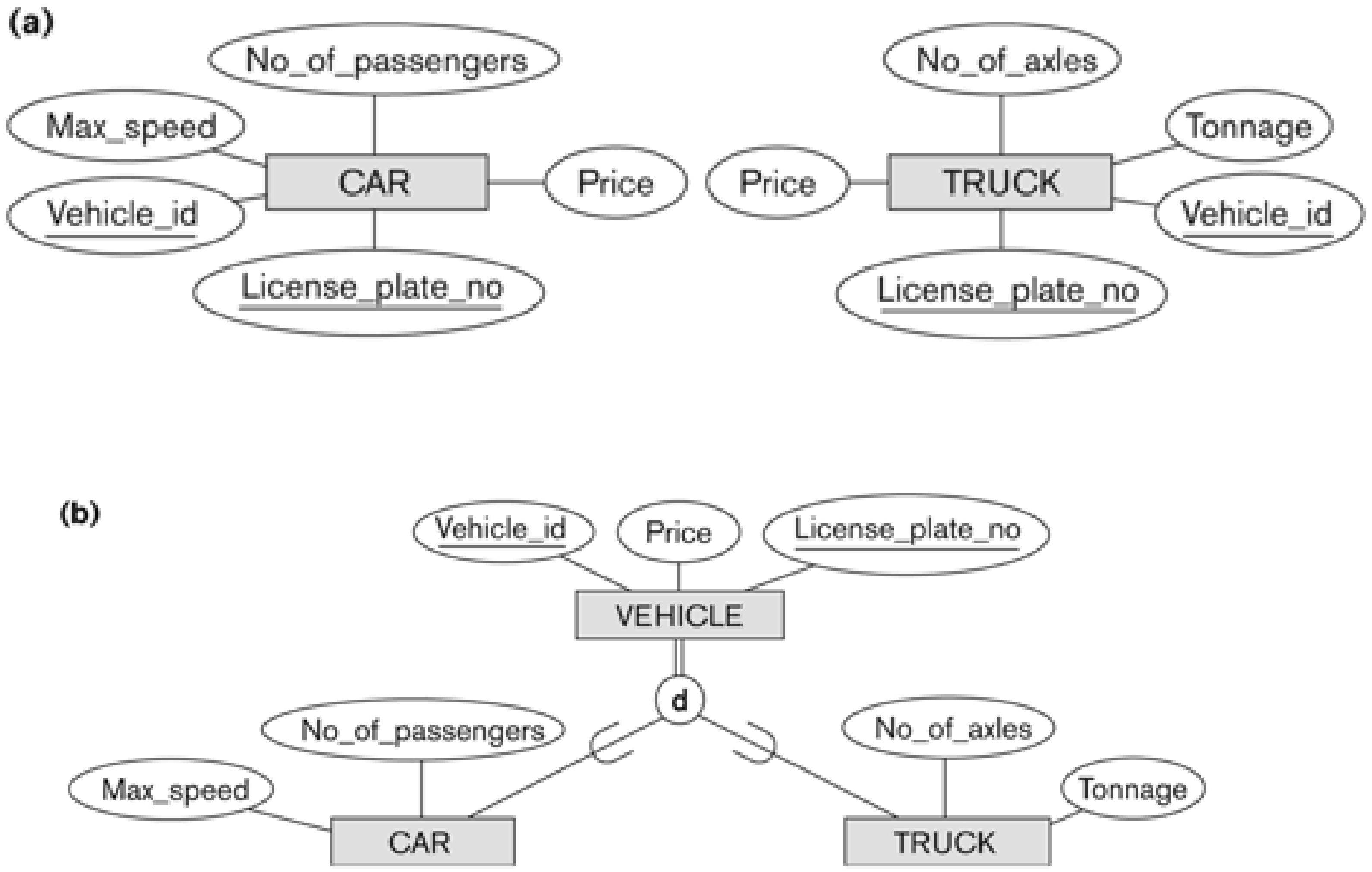






Generalization

- 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



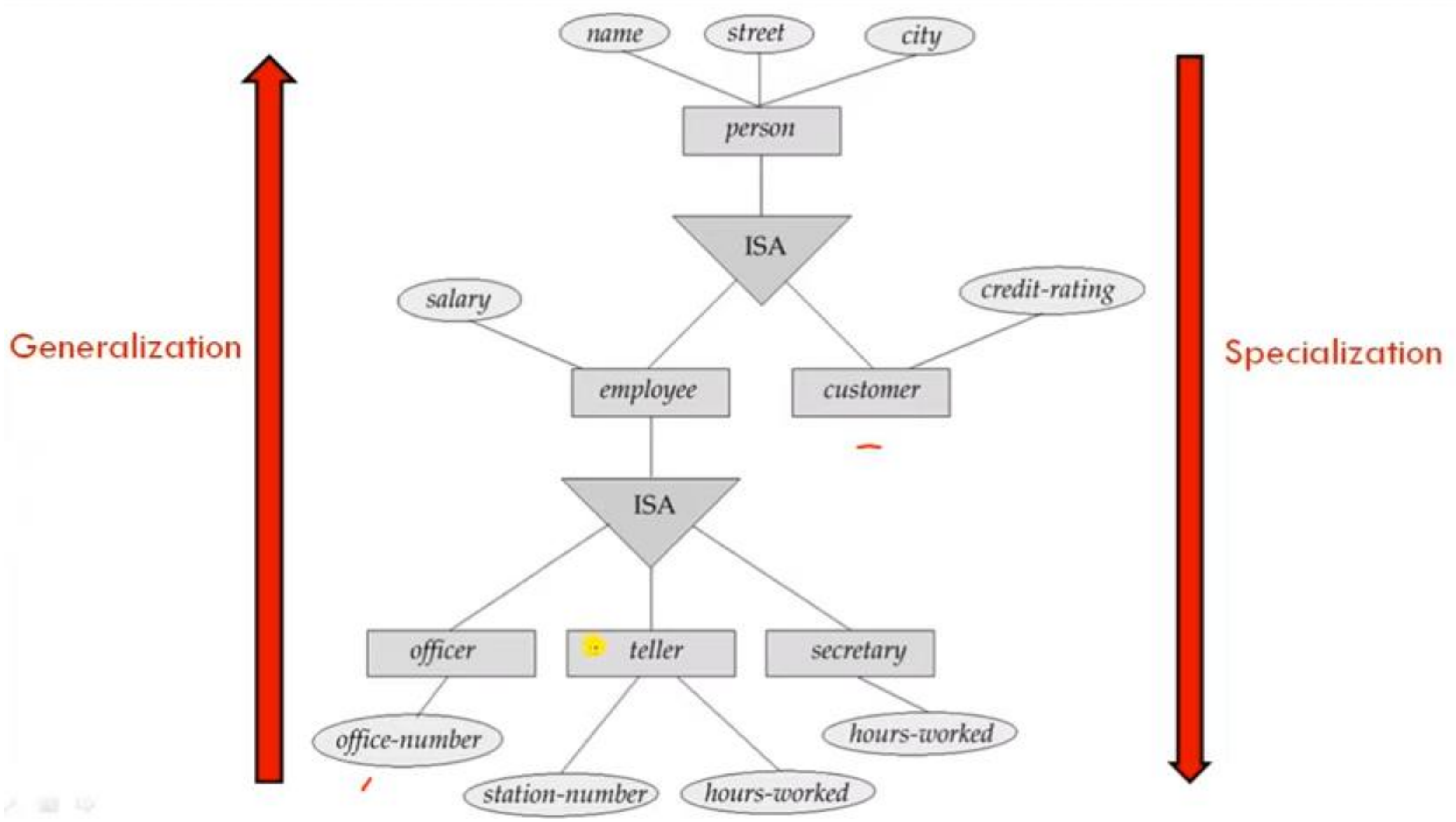
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

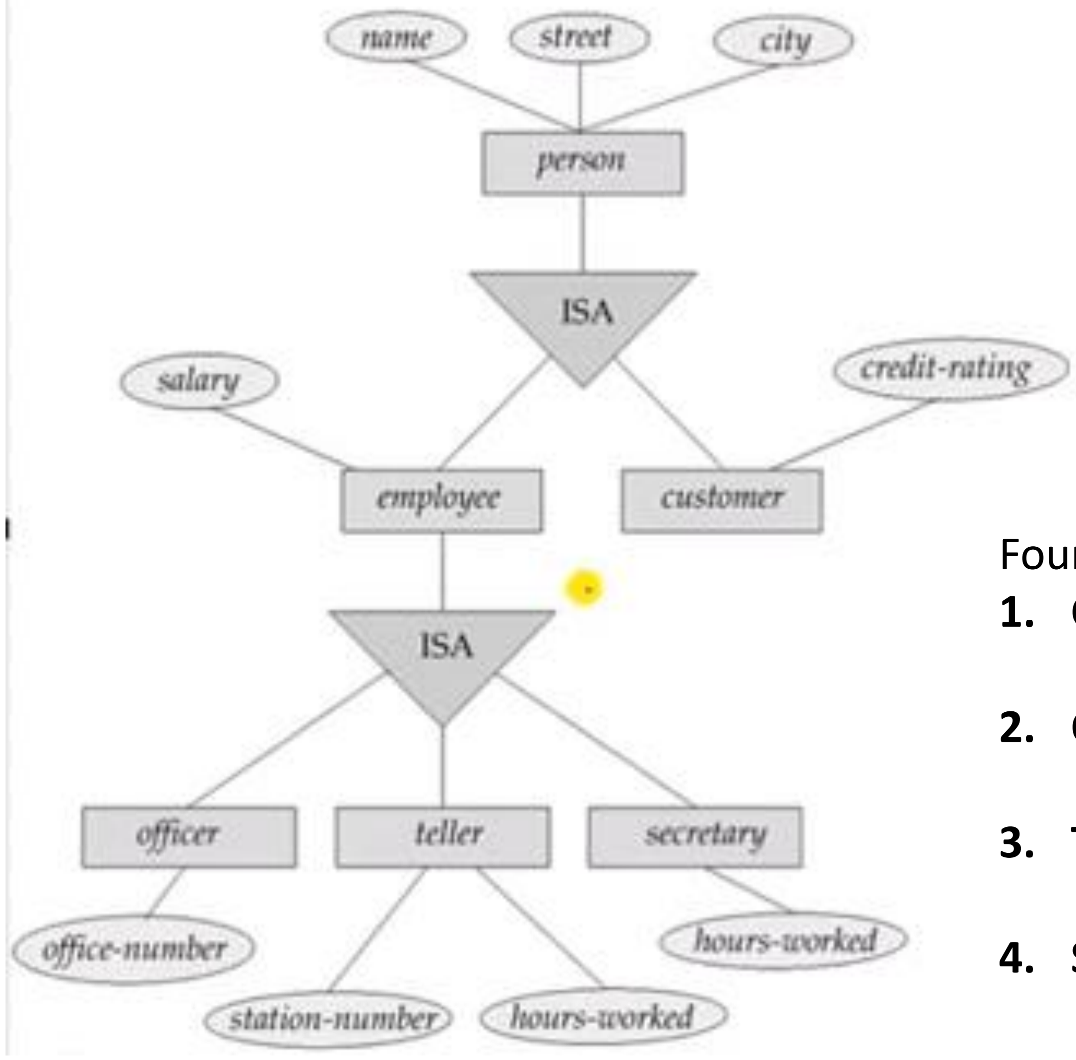
Generalization and Specialization (1)

- Diagrammatic notation are sometimes used to distinguish between generalization and specialization
  - Arrow pointing to the generalized superclass represents a generalization
  - Arrows pointing to the specialized subclasses represent a specialization
  - We *do not use* this notation because it is often subjective as to which process is more appropriate for a particular situation
  - We advocate not drawing any arrows

Generalization and Specialization (2)

- Data Modeling with Specialization and Generalization
  - A superclass or subclass represents a collection (or set or grouping) of entities
  - It also represents a particular *type of entity*
  - Shown in rectangles in EER diagrams (as are entity types)
  - We can call all entity types (and their corresponding collections) ***classes***, whether they are entity types, superclasses, or subclasses


name
street
city
person
ISA
salary
credit-rating
Generalization
Specialization
employee
customer
ISA
officer
teller
secretary
office-number
station-number
hours-worked
hours-worked



IMP

Four tables can be formed:

1. **Customer**(name,street,city,credit_rating)

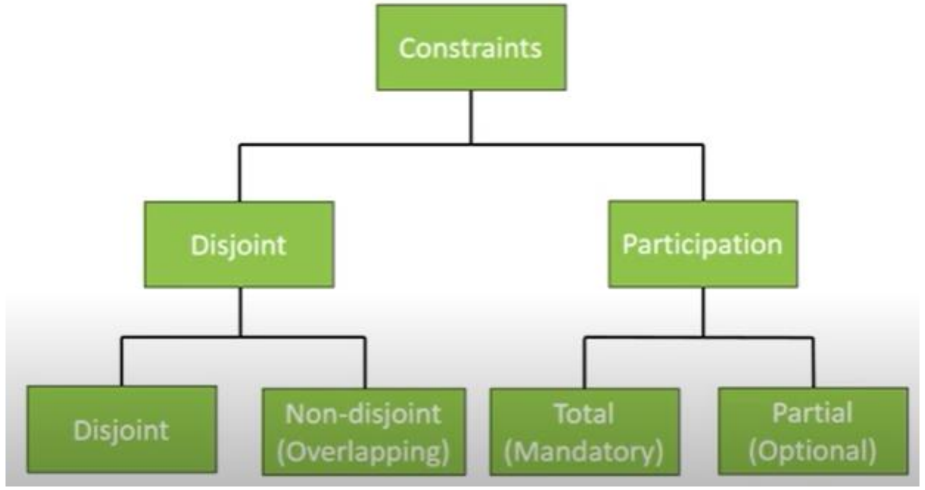
2. **Officer**(name,street,city,salary,office_number)

3. **Teller**(name,street,city,salary,station_number,hours_worked)

4. **Secretary**(name,strret,city,hours_worked)

Constraints on Specialization and Generalization

- Two basic constraints can apply to a specialization/generalization:
  - Disjoint Constraint:
  - Completeness/Participation Constraint:



Constraints on Specialization and Generalization (4)

- 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

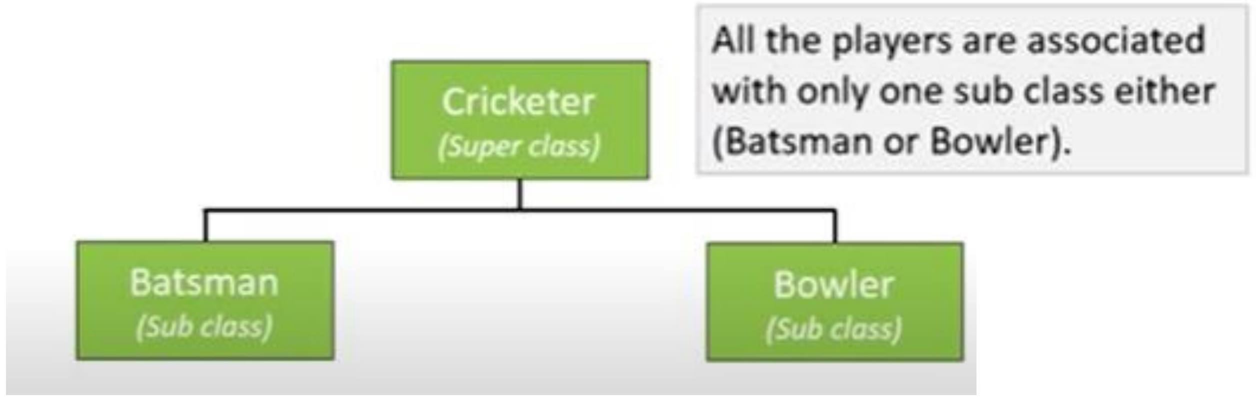
Constraints on Specialization and Generalization (5)

- 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

Constraints on Specialization and Generalization (6)

- 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.

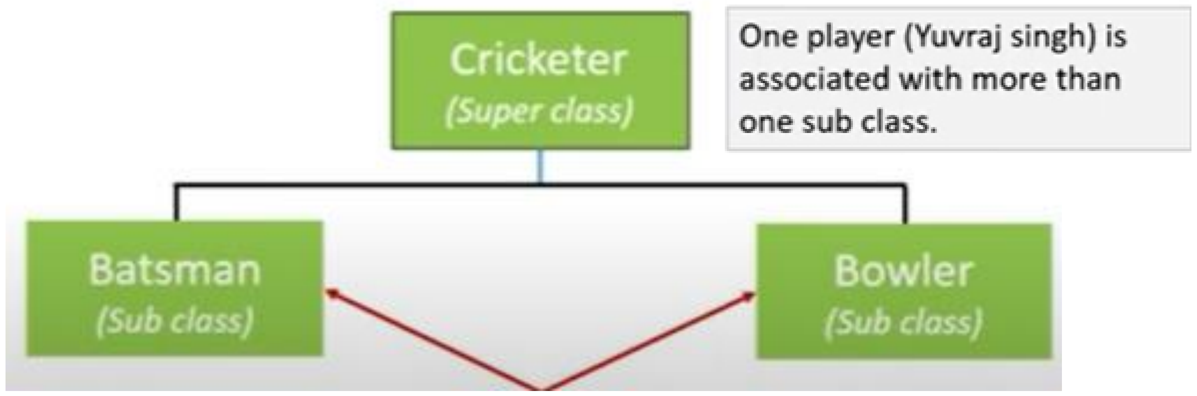
Disjoint Constraint


All the players are associated with only one sub class either (Batsman or Bowler).
Cricketer
(Super class)
Batsman
(Sub class)
Bowler
(Sub class)

It 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.
It is denoted by ‘d’in circle.

- In the diagram Employee can be either Secretary or Technician or Engineer

Overlapping Constraint

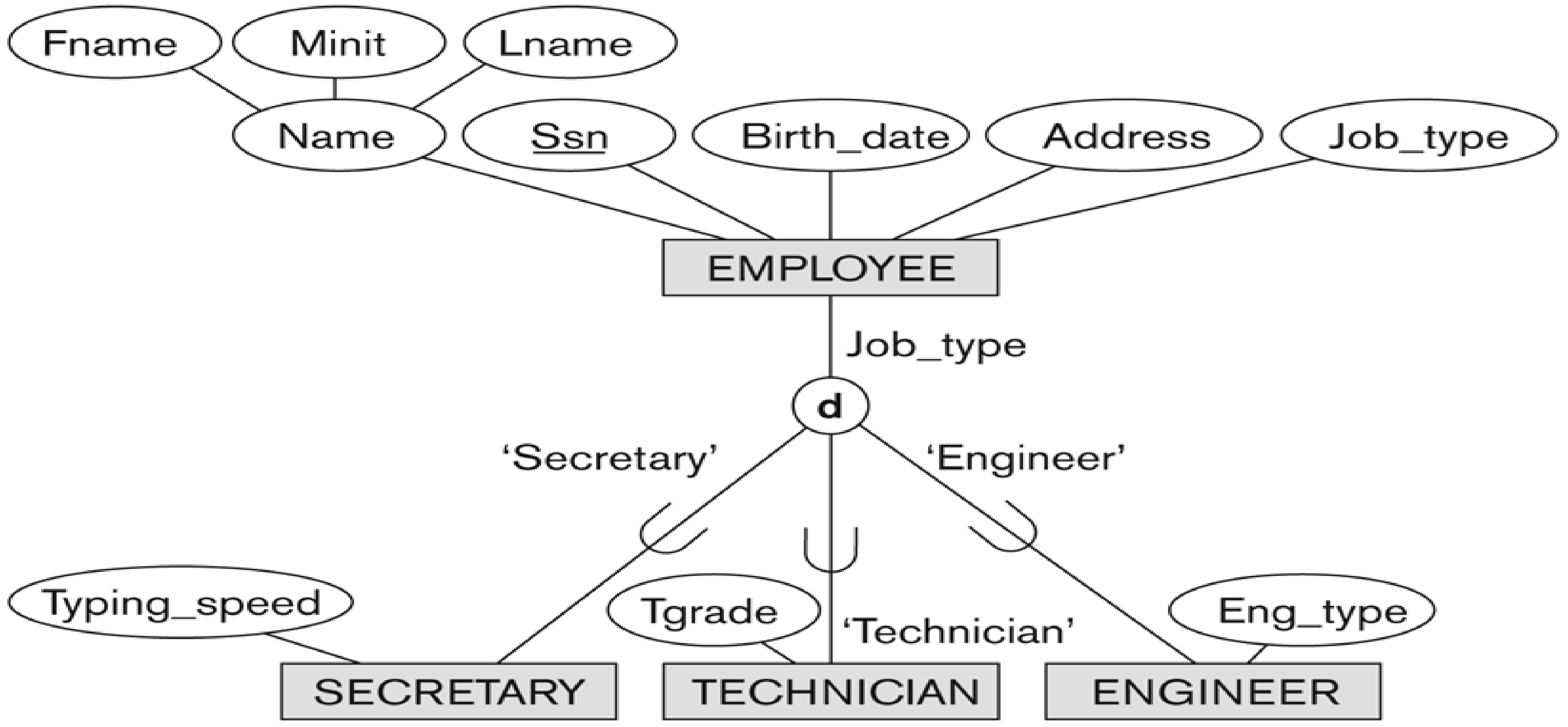

Cricketer
(Super class)
One player (Yuvraj singh) is associated with more than one sub class.
Batsman
(Sub class)
Bowler
(Sub class)

If the subclasses are not constrained to be disjoint, their sets of entities may be overlapping; that is, the same entity may be a member of more than one subclass of the specialization.

This case, which is the default, is displayed by placing an ‘o’ in the circle.

Example of disjoint partial Specialization

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



Example of overlapping total Specialization

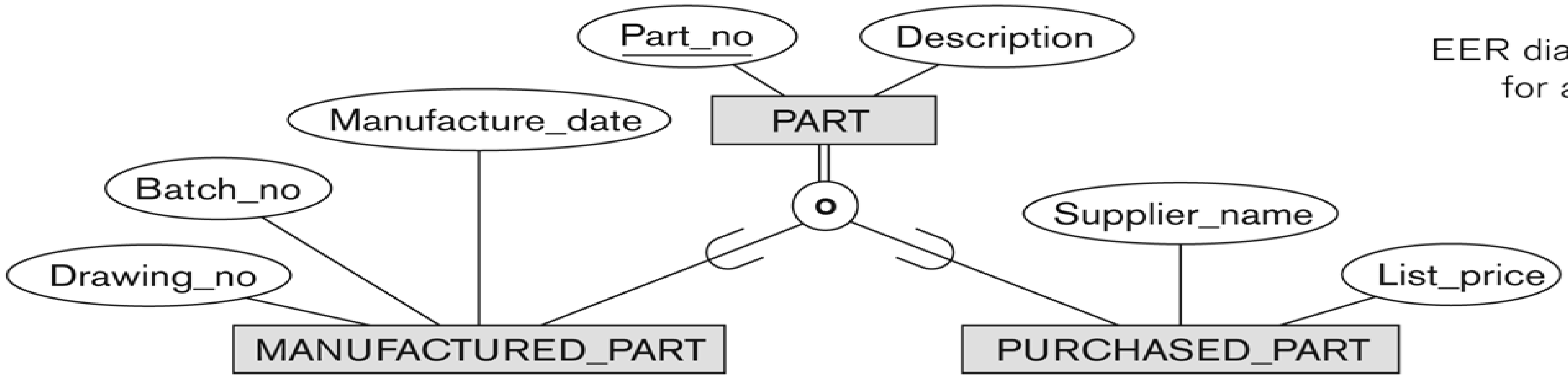


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (1)

- 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***)

Shared Subclass "Engineering_Manager"

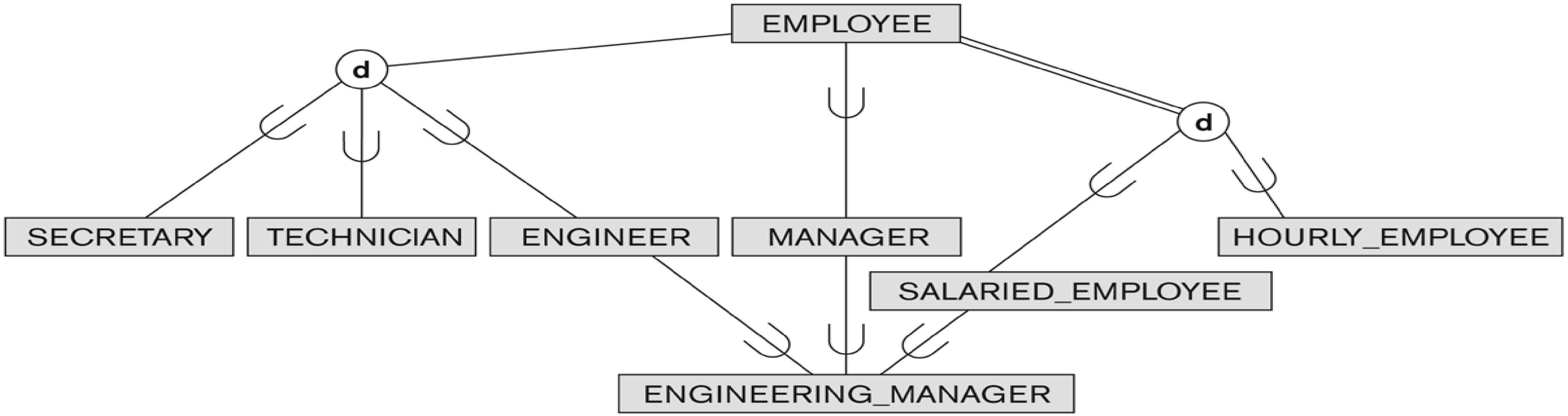


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (2)

- In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses
- 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*
- We just use *specialization* (to stand for the end result of either specialization or generalization)

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (3)

- In *specialization*, start with an entity type and then define subclasses of the entity type by successive specialization
  - called a *top down* conceptual refinement process
- In *generalization*, start with many entity types and generalize those that have common properties
  - Called a *bottom up* conceptual synthesis process
- In practice, a *combination of both processes* is usually employed

Specialization / Generalization Lattice Example
(UNIVERSITY)

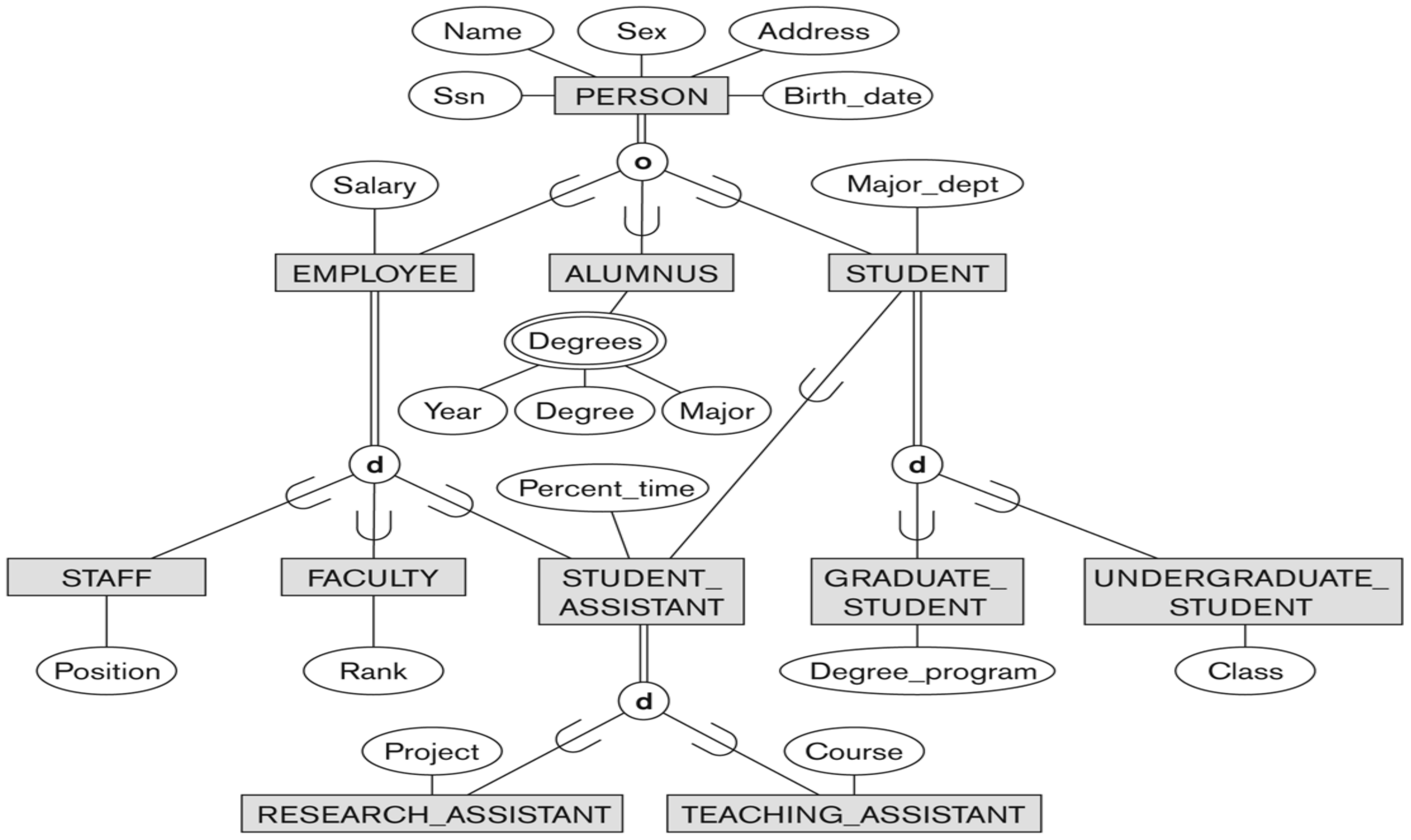


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Categories (UNION TYPES) (1)

- All of the *superclass/subclass relationships* we have seen thus far have a single superclass
- A shared subclass 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

Categories (UNION TYPES) (2)

- 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
- Difference from *shared subclass*, which is a:
  - subset of the *intersection* of its superclasses
  - shared subclass member must exist in ***all*** of its superclasses

Formal Definitions of EER Model (1)

- Class C:
  - A type of entity with a corresponding set of entities:
    - could be entity type, subclass, superclass, or category
- Note: The definition of *relationship type* in ER/EER should have 'entity type' replaced with 'class‘ to allow relationships among classes in general
- Subclass S is a class whose:
    - Type inherits all the attributes and relationship of a class C
    - Set of entities must always be a subset of the set of entities of the other class C
      - $S \subseteq C$
    - C is called the superclass of S
    - A superclass/subclass relationship exists between S and C

Formal Definitions of EER Model (2)

- Specialization Z: Z = {S1, S2,…, Sn} is a set of subclasses with same superclass G; hence, G/Si is a superclass relationship for i = 1, …., n.
  - G is called a generalization of the subclasses {S1, S2,…, Sn}
  - Z is total if we always have:
    - S1 ∪ S2 ∪ … ∪ Sn = G;
    - Otherwise, Z is partial.
  - Z is disjoint if we always have:
    - Si ∩ Sj empty-set for i ≠ j;
  - Otherwise, Z is overlapping.

Formal Definitions of EER Model (3)

- Subclass S of C is predicate defined if predicate (condition) p on attributes of C is used to specify membership in S;
  - that is, S = C[p], where C[p] is the set of entities in C that satisfy condition p
- A subclass not defined by a predicate is called user-defined
- Attribute-defined specialization: if a predicate A = ci (where A is an attribute of G and ci is a constant value from the domain of A) is used to specify membership in each subclass Si in Z
  - Note: If ci ≠ cj for i ≠ j, and A is single-valued, then the attribute-defined specialization will be disjoint.

Two categories (UNION types): OWNER, REGISTERED_VEHICLE

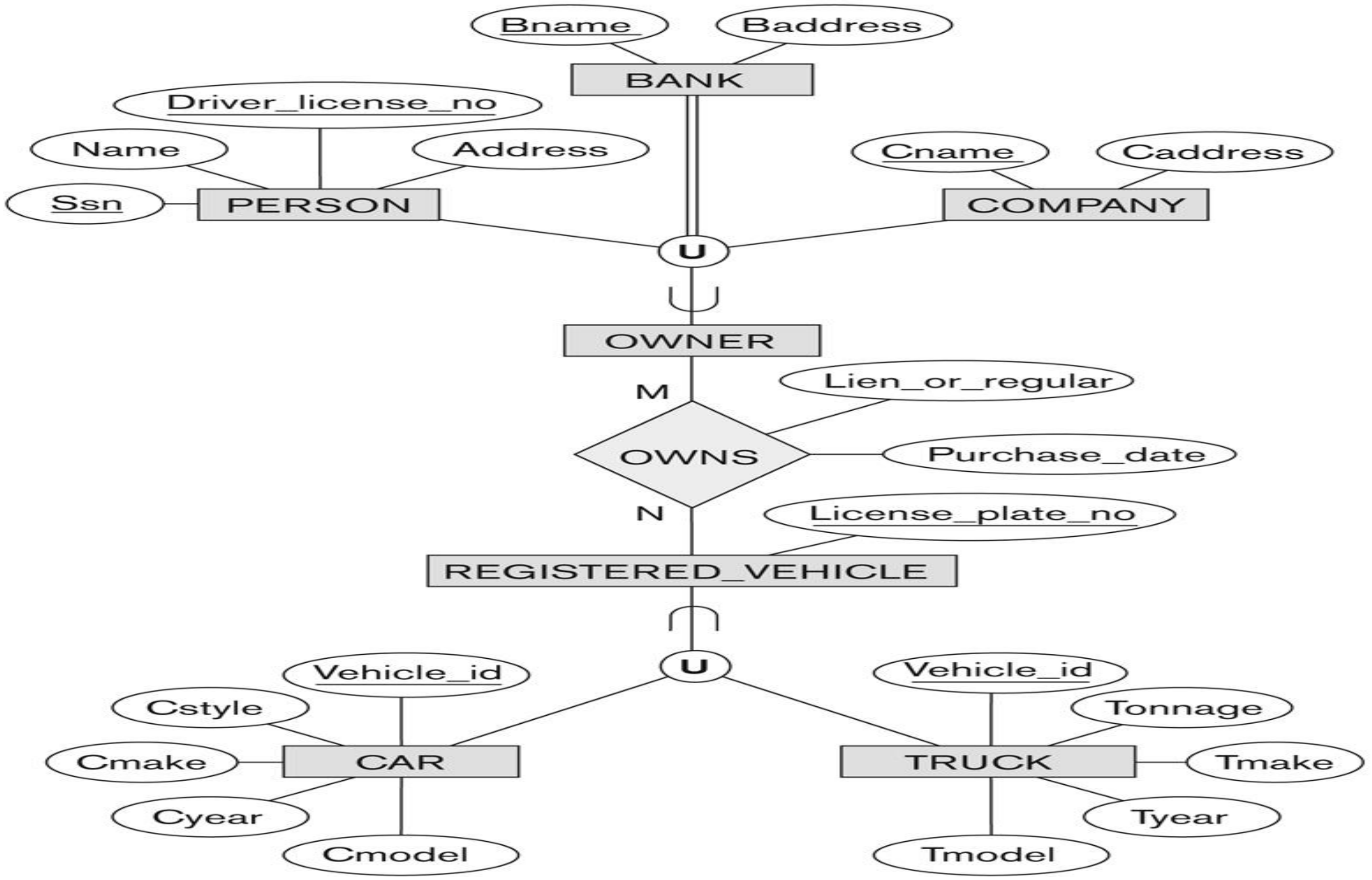


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Formal Definitions of EER Model (4)

- Category or UNION type T
  - A class that is a subset of the *union* of n defining superclasses D1, D2,…Dn, n>1:
    - $T \subseteq (D1 \cup D2 \cup \ldots \cup Dn)$
  - Can have a predicate pi on the attributes of Di to specify entities of Di that are members of T.
  - If a predicate is specified on every Di: $T = (D1[p1] \cup D2[p2] \cup \ldots \cup Dn[pn])$

Alternative diagrammatic notations

- ER/EER diagrams are a specific notation for displaying the concepts of the model diagrammatically
- DB design tools use many alternative notations for the same or similar concepts
- One popular alternative notation uses *UML class diagrams*
- see next slides for UML class diagrams and other alternative notations

UML Example for Displaying Specialization / Generalization

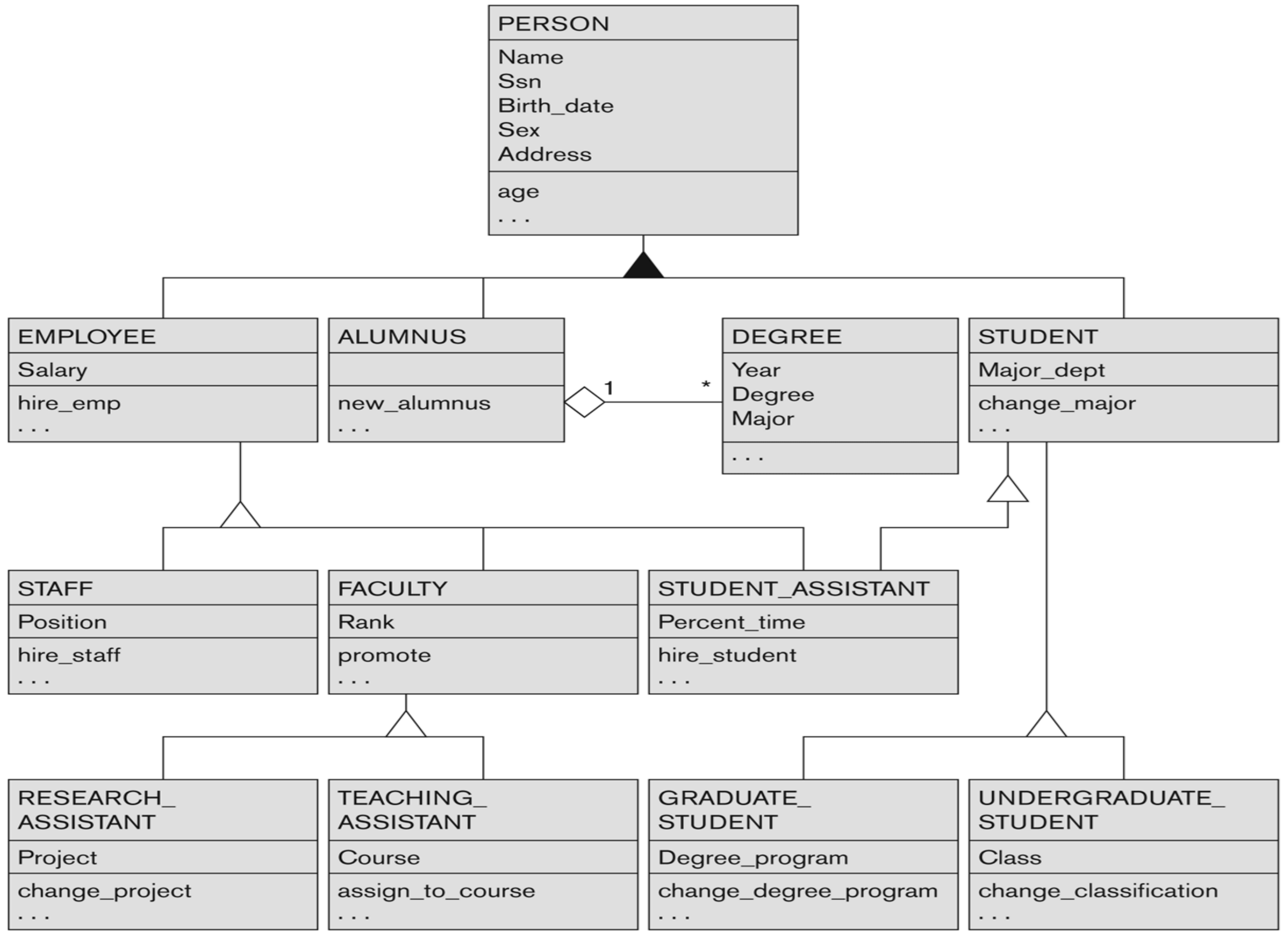


Figure 4.10
A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

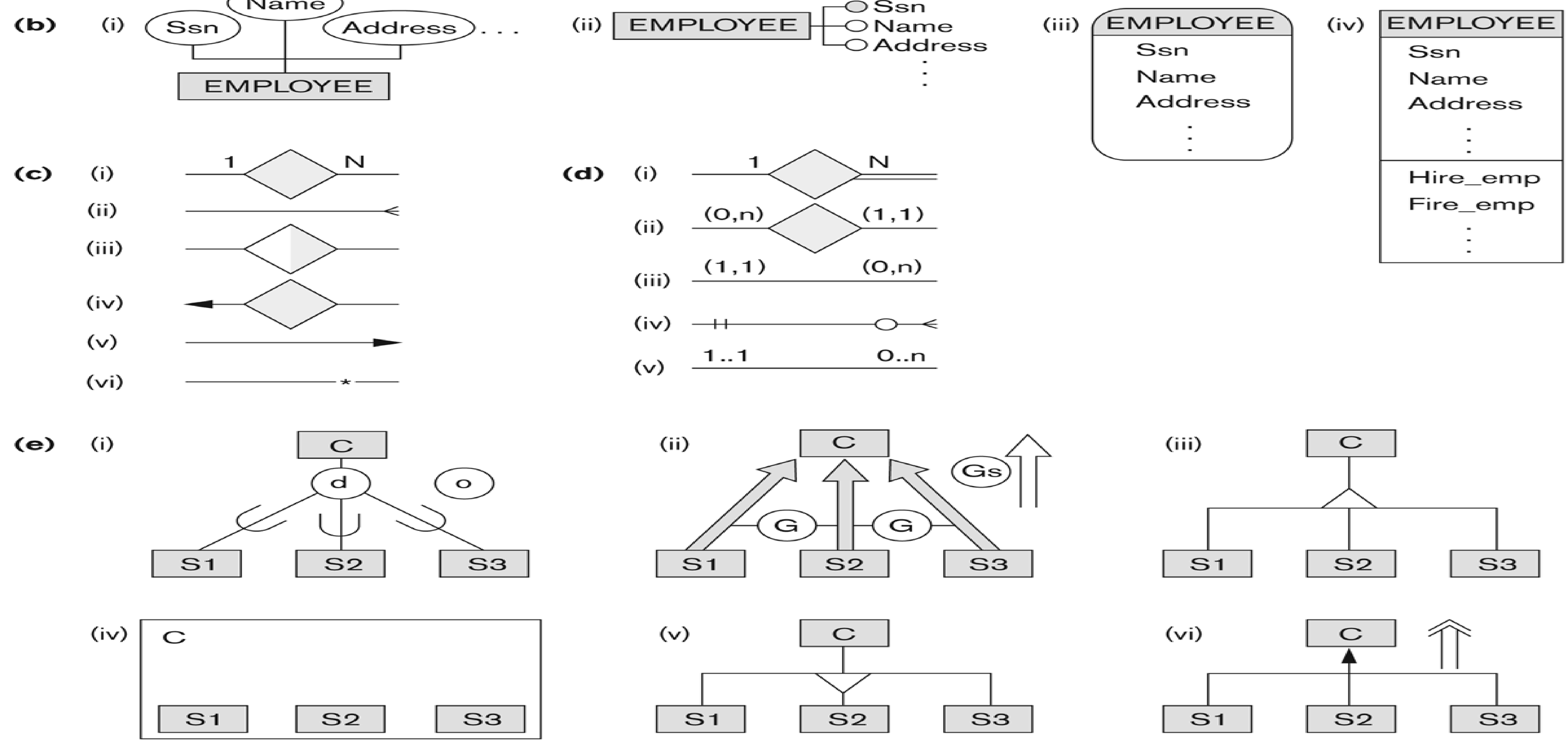


Figure A.1
Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

General Conceptual Modeling Concepts

- GENERAL DATA ABSTRACTIONS
  - CLASSIFICATION and INSTANTIATION
  - AGGREGATION and ASSOCIATION (relationships)
  - GENERALIZATION and SPECIALIZATION
  - IDENTIFICATION
- CONSTRAINTS
  - CARDINALITY (Min and Max)
  - COVERAGE (Total vs. Partial, and Exclusive (disjoint) vs. Overlapping)

Ontologies

- Use conceptual modeling and other tools to develop “a specification of a conceptualization”
  - **Specification** refers to the language and vocabulary (data model concepts) used
  - **Conceptualization** refers to the description (schema) of the concepts of a particular field of knowledge and the relationships among these concepts
- Many medical, scientific, and engineering ontologies are being developed as a means of standardizing concepts and terminology

Summary

- Introduced the EER model concepts
  - Class/subclass relationships
  - Specialization and generalization
  - Inheritance
- These augment the basic ER model concepts introduced in Chapter 3
- EER diagrams and alternative notations were presented