MODULE I
ER AND EER

Dr. Geetha Mary A
Associate Professor,
SCOPE, VIT

Sources:
Pearson Education, Inc. 2011, Elmasri/Navathe, Fundamentals of Database Systems, Sixth Edition
McGraw Hill Education , 2010, Silberschatz, Korth and Sudarshan, Database System Concepts, Sixth edition

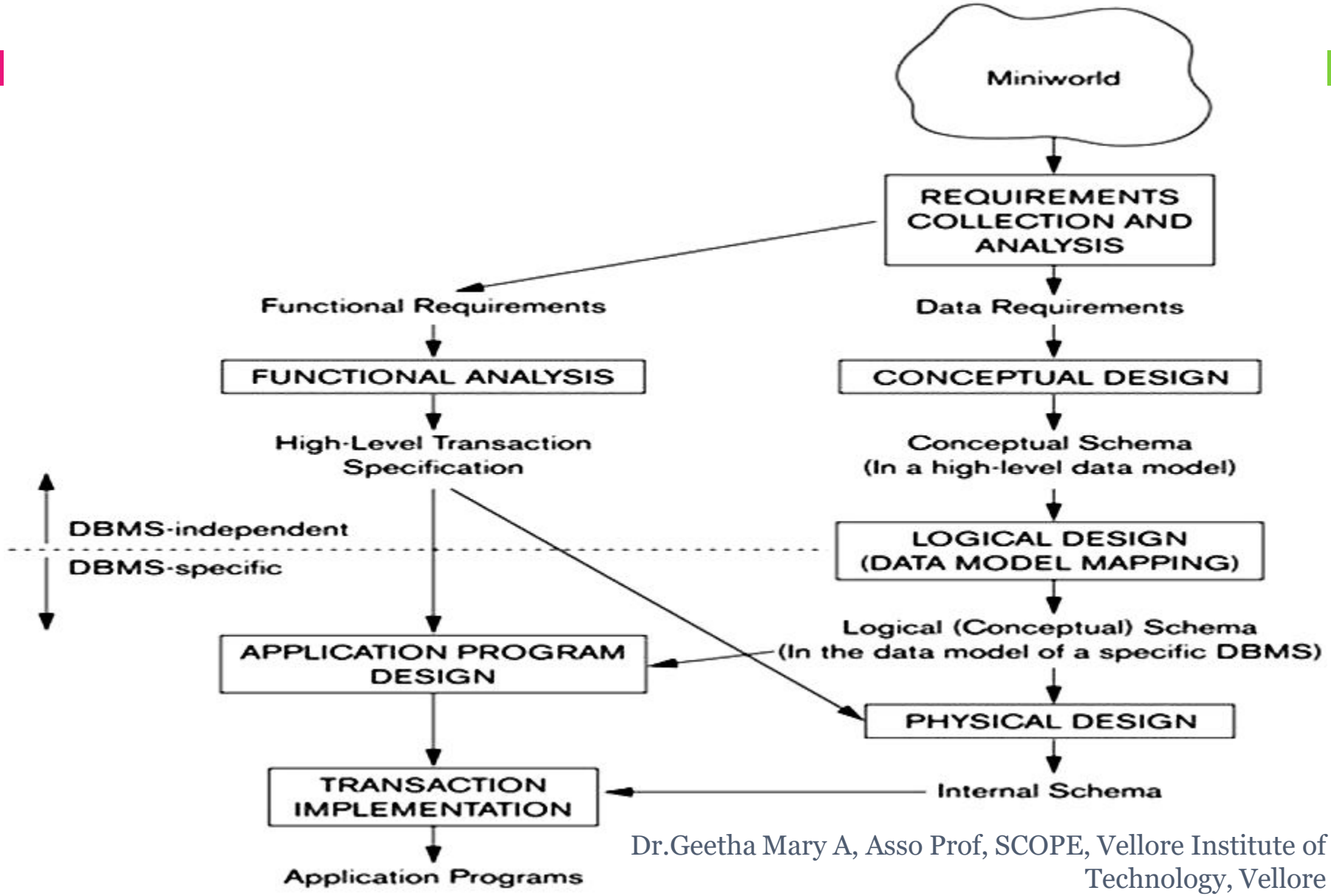
Topic Outline

- Overview of Database Design Process
- Example Database Application (COMPANY)
- ER Model Concepts
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types
- ER Diagrams - Notation
- ER Diagram for COMPANY Schema
- Alternative Notations – UML class diagrams, others

Overview of Database Design Process

- Two main activities:
  - Database design
  - Applications design
- Focus of this topic is on database design
  - To design the conceptual schema for a database application
- Applications design focuses on the programs and interfaces that access the database
  - Generally considered part of software engineering

Overview of Database Design Process



ER Model has detailed descriptions of

- What are the entities and relationships in the enterprise?
- What information about these entities and relationships should we store in the database?
- What are the integrity constraints or business rules that hold?

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.

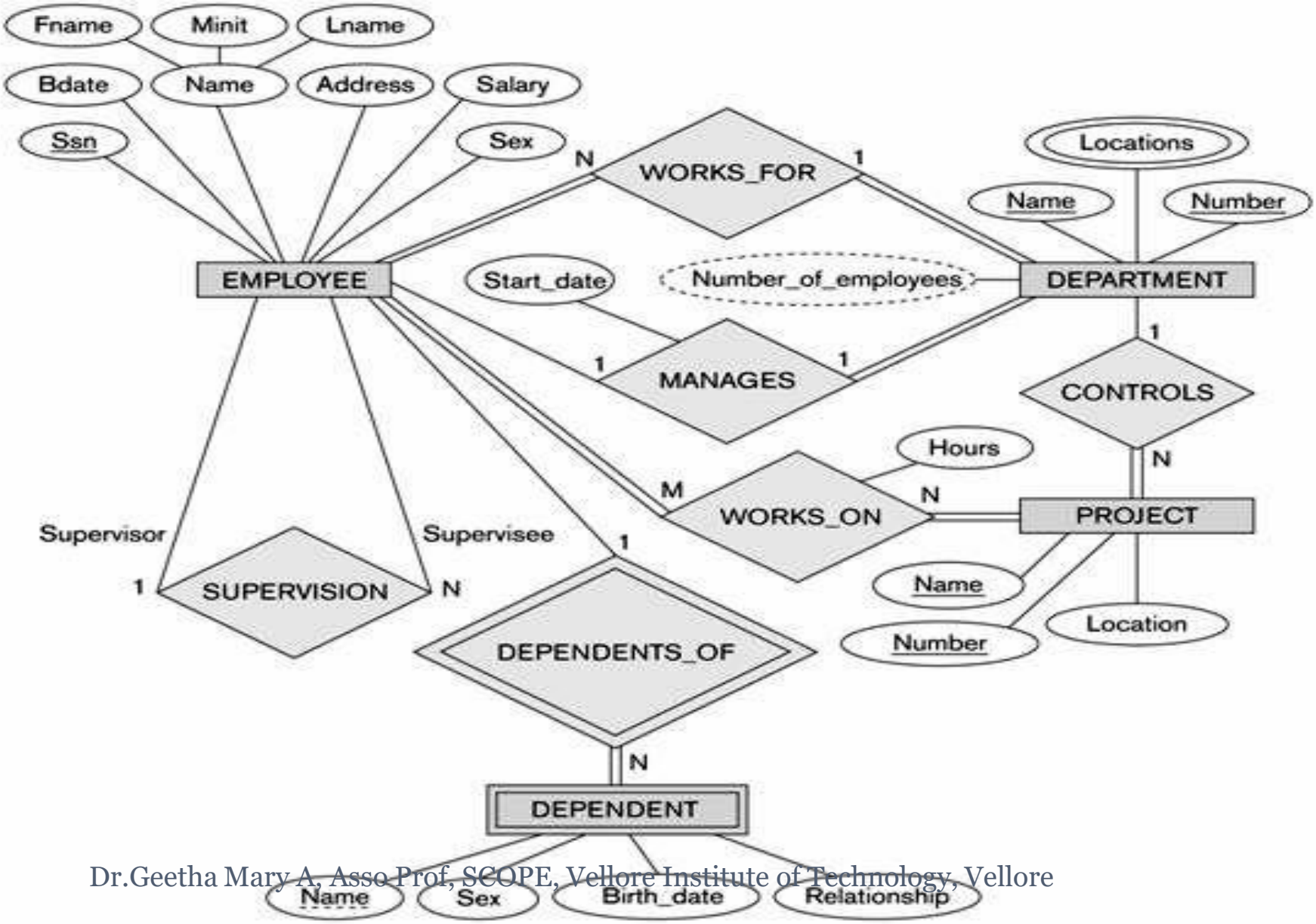
Example COMPANY Database (Contd.)

- We store each EMPLOYEE's social security number, address, salary, sex, 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, sex, birthdate, and relationship to the employee.

ER Model Concepts

- Entities and Attributes
  - **Entities** 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, Sex, 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', Sex='M', BirthDate='09-JAN-55'
  - Each attribute has a ***value set*** (or data type) associated with it – e.g. integer, string, subrange, enumerated type.

ER Schema diagram of company DB



Entity Sets

- A *database* can be modeled as:
  - a collection of entities,
  - relationship among entities.
- An *entity* is an object that exists and is distinguishable from other objects.
  - Example: specific person, company
- Entities have *attributes*
  - Example: people have *names* and *addresses*
- An *entity set* is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays

Entity Sets *customer* and *loan*

customer-id customer-name customer-street customer-city loan-number amount

321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

Attributes

- Entity is represented by a set of attributes(properties)

Eg:

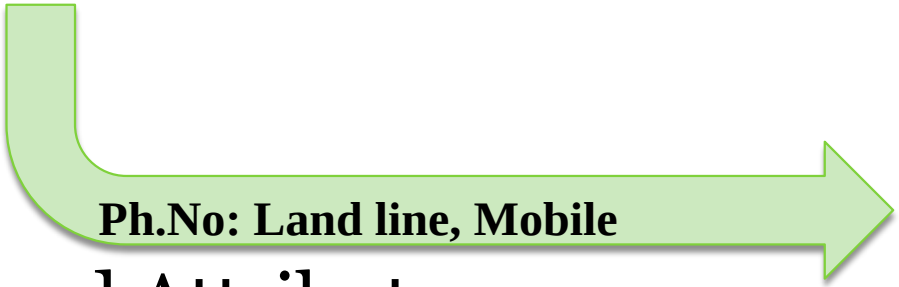
customer = (customer-id, customer-name, customer-street, customer-city)

loan = (loan-number, amount)

- *Domain* – the set of permitted values for each attribute

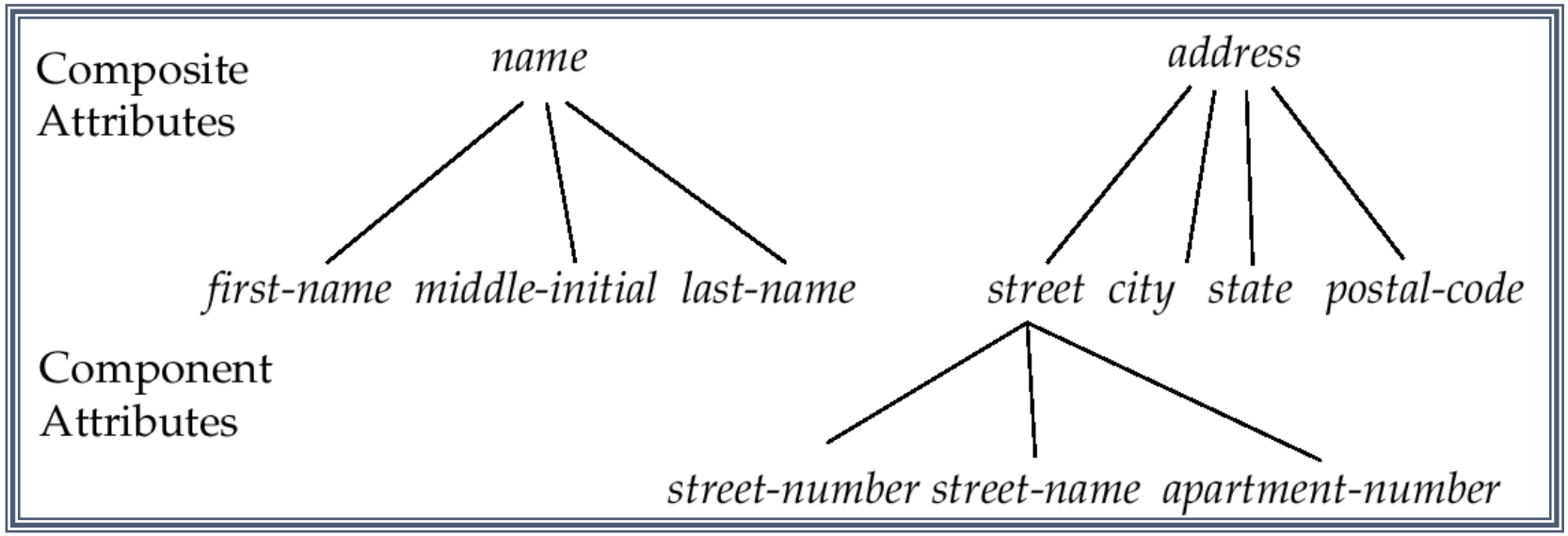
Attribute Types

- Simple and Composite
- Single valued and multi valued



- Stored versus Derived Attributes:
  - Age ➔ calculated from DOB
  - DOB ➔ Stored attribute

Composite Attributes



Grouping together related attributes

Attribute Types

- **Null Values:**
  - College Degrees of person, apartment number
- **Complex Attributes:**
  - Composite attribute and multivalued attributes are nested in an arbitrary way.
  - Composite attribute → parentheses ()
  - Multivalued attribute → braces { }
  - Eg:

    {AddressPhone({ Phone(AreaCode,PhoneNumber)},
    Address(streetAddress(number,street,apartmentNumber),
    city,state,zip)) }

Entity Types and Key Attributes

- Entities with the same basic attributes are grouped or typed into an entity type.
  - For example, the entity type EMPLOYEE and PROJECT.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
  - For example, SSN of EMPLOYEE.

Entity Types and Key Attributes

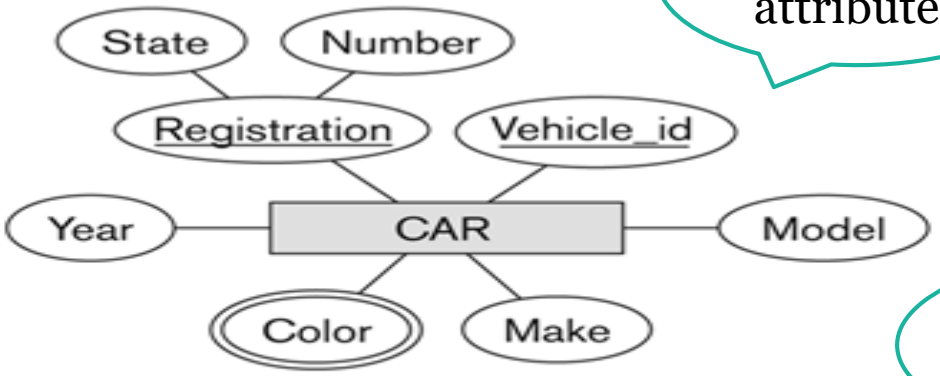
- A key attribute may be composite.
  - VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
  - The CAR entity type may have two keys:
    - VehicleIdentificationNumber (popularly called VIN)
    - VehicleTagNumber (Number, State),
- Each key is underlined

Displaying an Entity type

- In ER diagrams, an entity type is displayed in a rectangular box
- Attributes are displayed in ovals
  - Each attribute is connected to its entity type
  - Components of a composite attribute are connected to the oval representing the composite attribute
  - Each key attribute is underlined
  - Multivalued attributes displayed in double ovals
- See CAR example on next slide

Entity Type CAR with two keys and a corresponding Entity Set

(a)



Key attributes

Entity set

(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

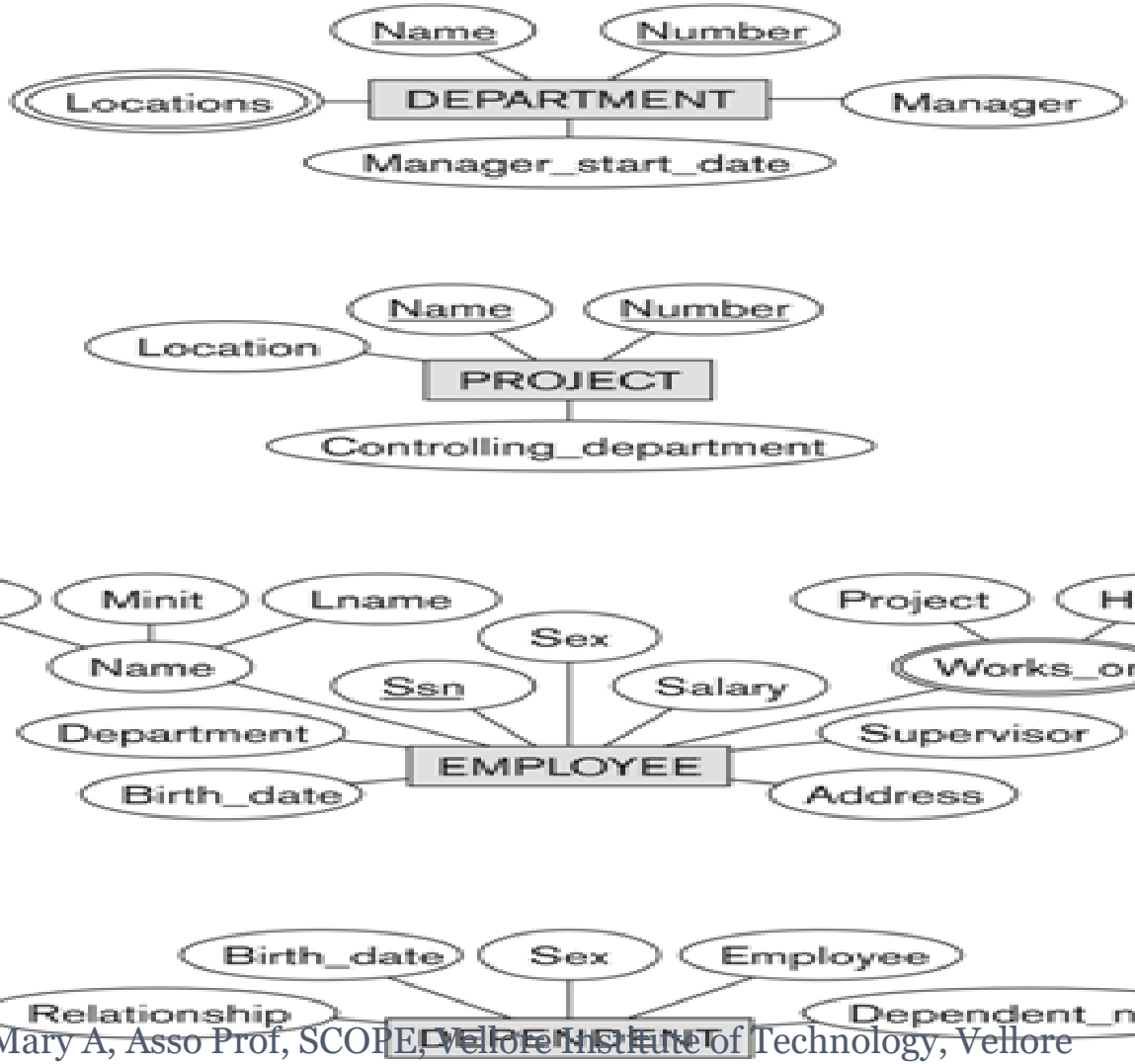
Entity Set

- Each entity type will have a collection of entities stored in the database
  - Called the **entity set**
- Previous slide shows three CAR entity instances in the entity set for CAR
- Same name (CAR) used to refer to both the entity type and the entity set
- Entity set is the current *state* of the entities of that type that are stored in the database

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
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

Initial Design of Entity Types:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT



Weak Entity Types

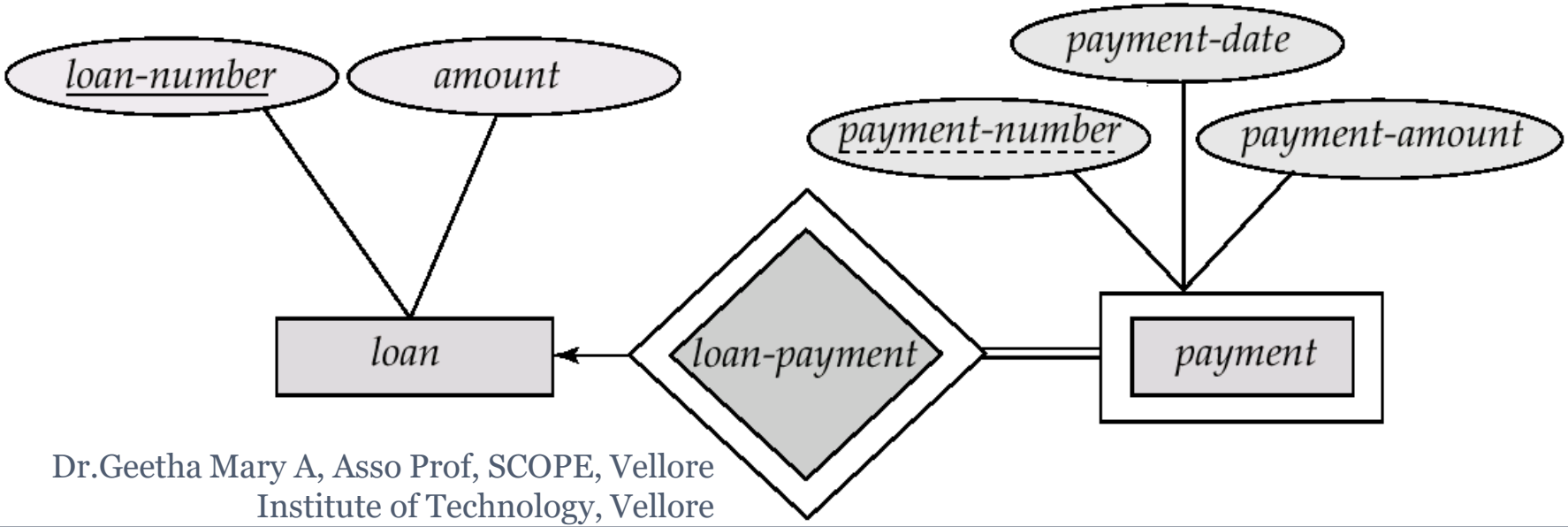
- An entity that does not have a key attribute
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying entity type
- **Example:**
  - A DEPENDENT entity is identified by the dependent's first name, *and* the specific EMPLOYEE with whom the dependent is related
  - Name of DEPENDENT is the *partial key*
  - DEPENDENT is a *weak entity type*
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Weak Entity Sets

- Is an entity set that does not have a primary key
  - Payment-number, payment date, payment amount
- Must be associated with a *identifying entity set*
  - *Loan*
  - Identifying relationship depicted using a double diamond
- The *discriminator* *is a set of attributes that allows the distinction.*
  - Payment-number
- The primary key of a weak entity set is formed by the primary key of the identifying entity set , plus the weak entity set's discriminator.
  - {loan-number,payment-number}

Weak Entity Sets (Cont.)

- weak entity set is depicted by double rectangles.
- discriminator of a weak entity set is underlined with a dashed line.
- *payment-number* – discriminator of the *payment* entity set
- Primary key for *payment* – (*loan-number*, *payment-number*)



Entity-Relationship (E-R) Modeling

- Relationship
  - An association between the instances of one or more entity types that is of interest to the organization
  - Association indicates that an event has occurred or that there is a natural link between entity types
  - Relationships are always labeled with verb phrases

Naming and Defining Relationships

- Relationship name is a verb phrase
- Avoid vague names
- Guidelines for defining relationships
  - Definition explains what action is being taken and why it is important
  - Give examples to clarify the action
  - Optional participation should be explained
  - Explain reasons for any explicit maximum cardinality

Naming and Defining Relationships

- Guidelines for defining relationships
  - Explain any restrictions on participation in the relationship
  - Explain extent of the history that is kept in the relationship
  - Explain whether an entity instance involved in a relationship instance can transfer participation to another relationship instance

Relationships

- Relationship Types and Sets
- Relationship Degree
- Entity Roles and Recursive Relationships
- Relationship Constraints
- Attributes of Relationship Types

Relationship Types and Sets

A Relationship is an association among two or more entities

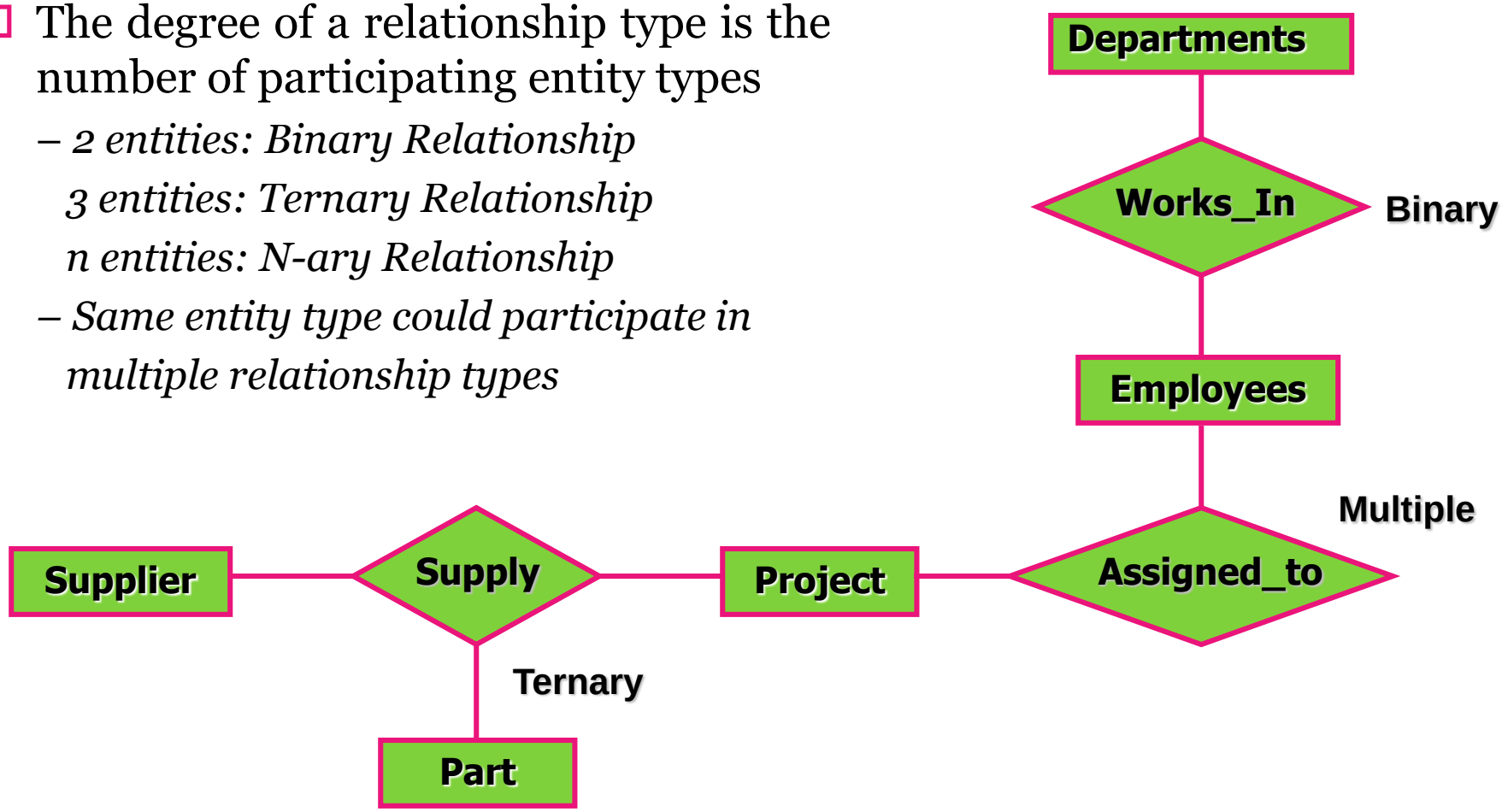
(e.g John works in Pharmacy department)

- A Relationship Type defines the relationship, and a Relationship Set represents a set of relationship instances
- A Relationship Type thus defines the structure of the Relationship Set

Relationship Type and corresponding Set are customarily referred to by the ***same*** name

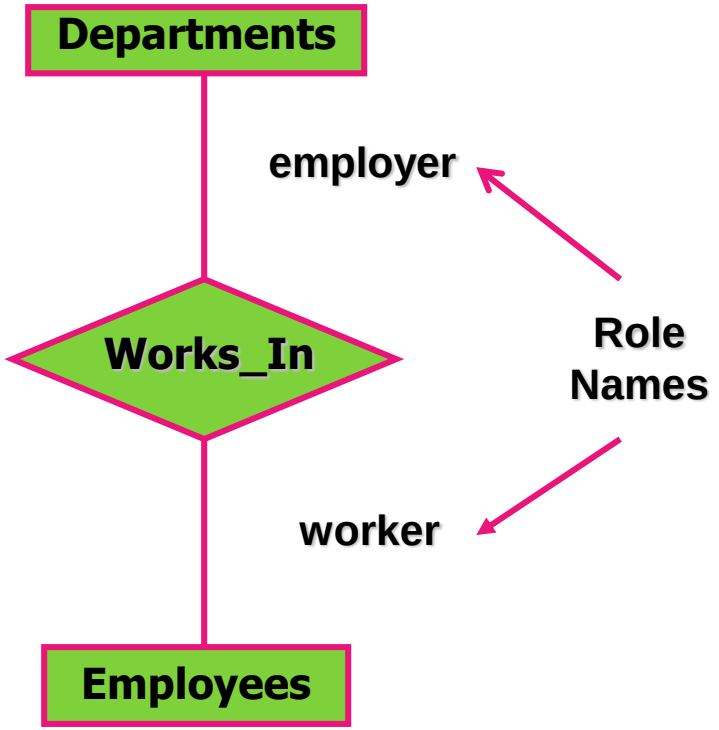
Relationship Degree

- The degree of a relationship type is the number of participating entity types
  - *2 entities: Binary Relationship*
    3 entities: Ternary Relationship
    n entities: N-ary Relationship
  - *Same entity type could participate in multiple relationship types*



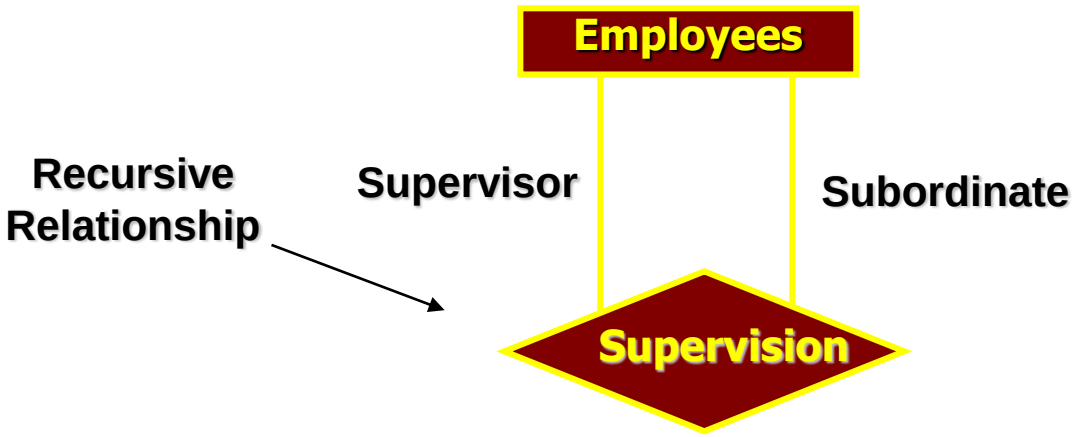
Entity Roles

- Each entity type that participates in a relationship type plays a particular role in the relationship type
- The role name signifies the role that a participating entity from the entity type plays in each relationship instance, i.e. it explains what the relationship means



Recursive Relationships

- Same entity type can participate more than once in the same relationship type under different "roles"
- Such relationships are called "**Recursive Relationships**"



Relationship Constraints

What are Relationship Constraints ?

- Constraints on the relationship type limit the possible combination of entities that may participate in the corresponding relationship set

Kinds of Constraints

What kind of constraints can be defined in the ER Model?

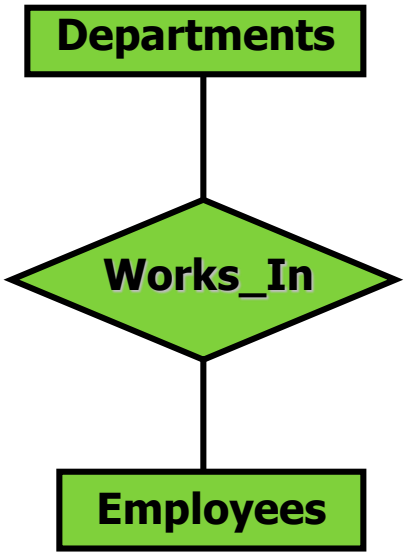
- Cardinality Constraints
- Participation Constraints

Together called "Structural Constraints"

Constraints are represented by specific notation in the ER diagram

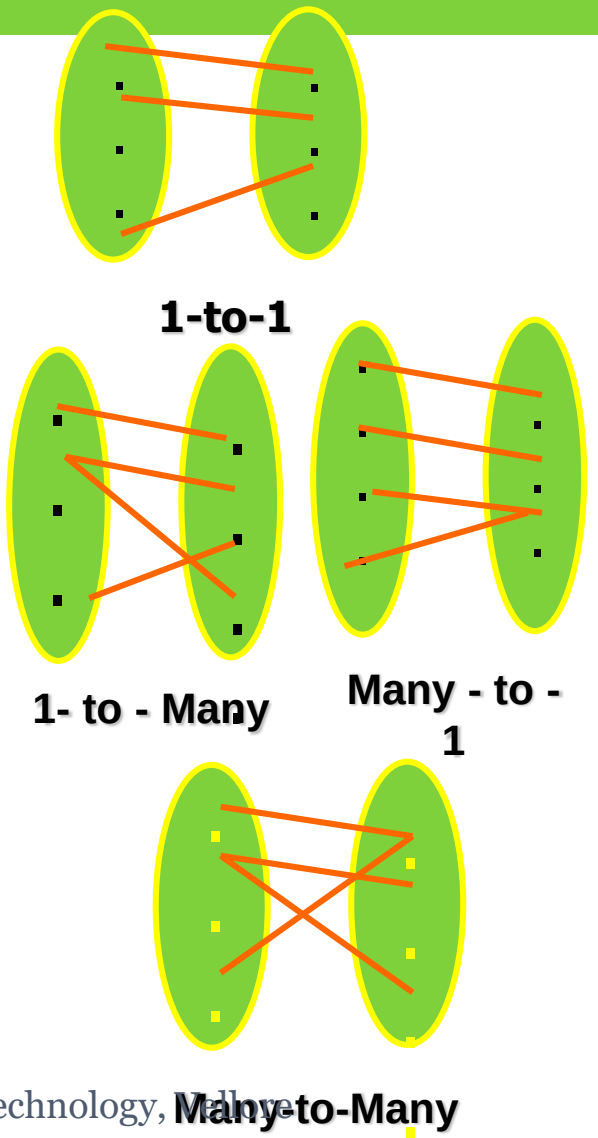
Possible Cardinality Ratios

- The "Cardinality Ratio" for a binary relationship specifies the number of relationship instances that an entity can participate in
  - Works-In is a binary relationship
  - Participating entities are DEPARTMENT : EMPLOYEE
  - *One* department can have *Many* employees - Cardinality Ratio is 1 : N



Possible Cardinality Ratios

- 1–to-1 (1 : 1)
  - – Both entities can participate in only **one** relationship instance
- 1-to-Many, Many-to-1 (1 : N, N : 1)
  - – One entity can participate in **many** relationship instances
- Many-to-Many (N: M)
  - – Both entities can participate in **many** relationship instance



Example Cardinality Constraints

How many Employees can work in a Department?

One employee can work in only one department

How many Employees can be employed by a Department?

One department can employ many employees

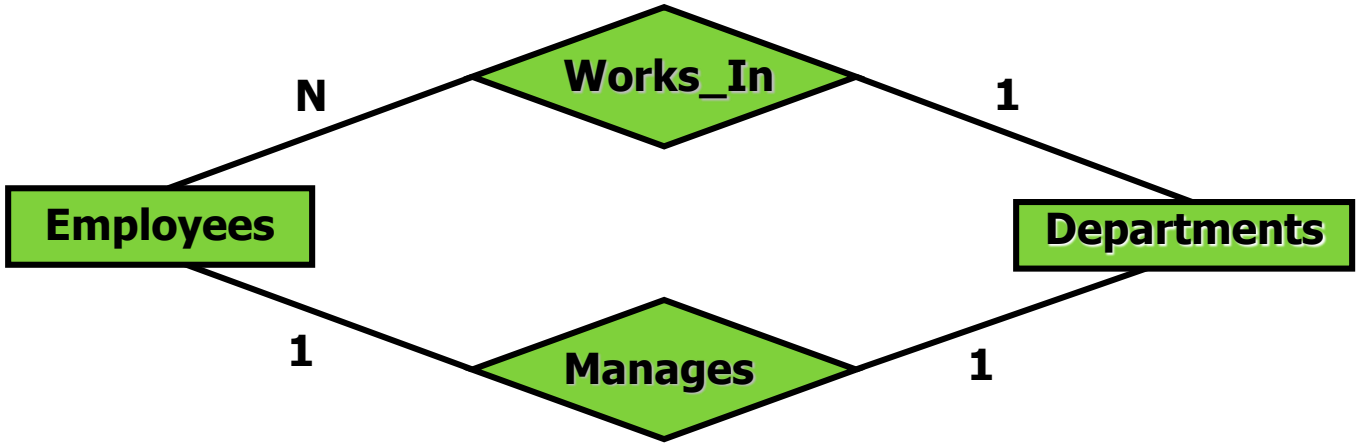
How many managers can a department have?

One department can have only one manager

How many departments can an employee manage?

One employee can have manage only one department

Representing Cardinality



One employee can work in only one department

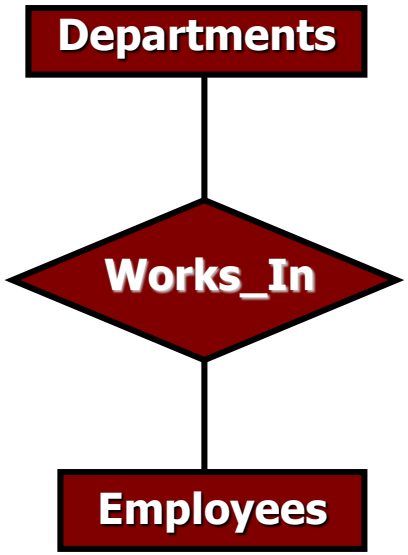
One department can employ many employees

One department can have only one manager

One employee can manage only one department

Existence Dependency

- Existence dependency indicates whether the existence of an entity depends on its relationship to another entity via the relationship type
  - Every employee must work for a department - EMPLOYEE is existentially dependent on DEPARTMENT via the Works In relationship type



Kinds of participating constraints

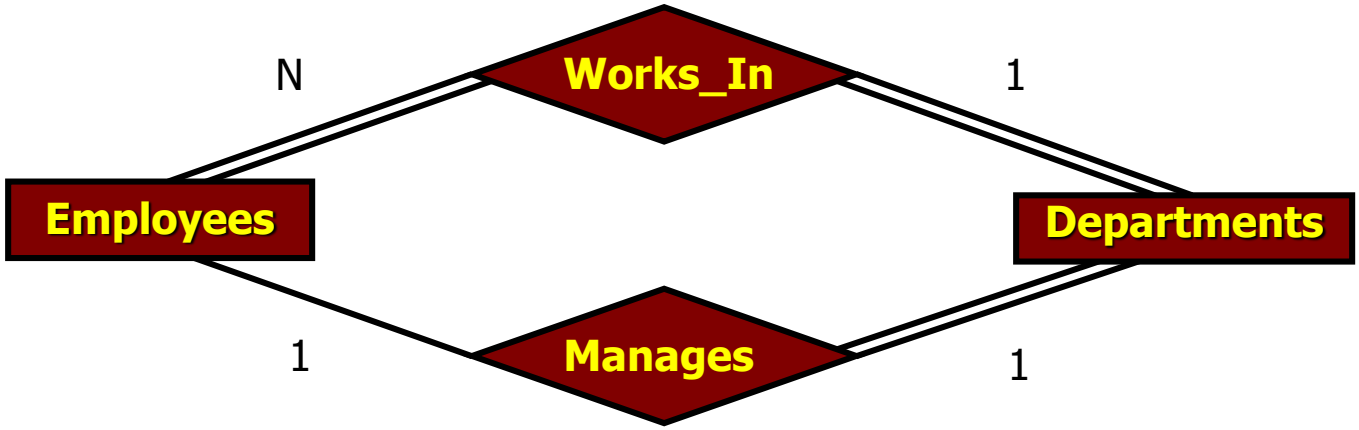
- TOTAL Participation (Existence Dependency)

  Constraint : Every employee must work for a department

- PARTIAL Participation

  Constraint : Not every employee is a manager

Representing Participation



Every employee must work for a department

Every department must have a manager

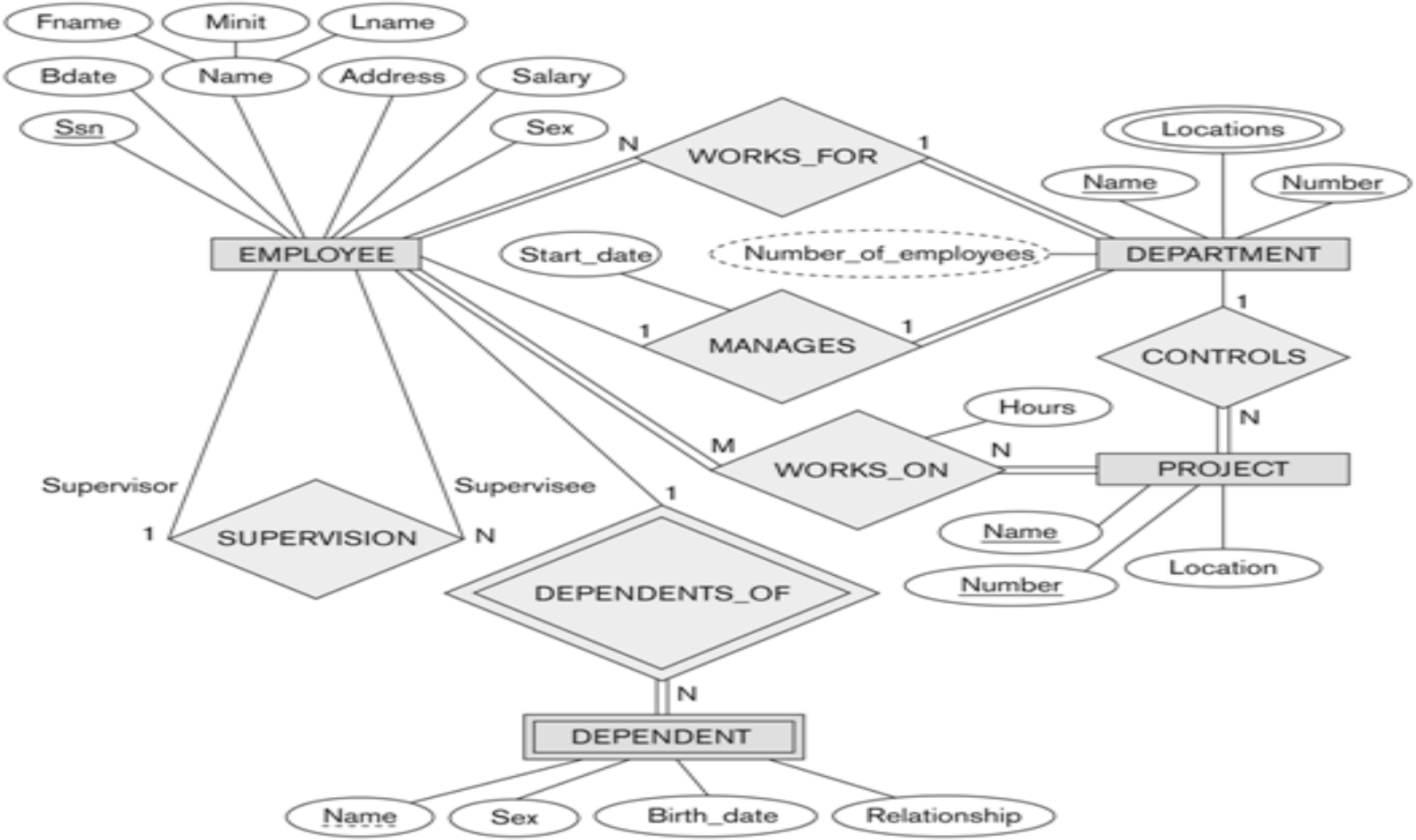
Every department must have employees

Not every employee is a manager

Attributes of Relationship types

- A relationship type can have attributes:
  - For example, HoursPerWeek of WORKS_ON
  - Its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.
    - A value of HoursPerWeek depends on a particular (employee, project) combination
  - Most relationship attributes are used with M:N relationships
    - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship

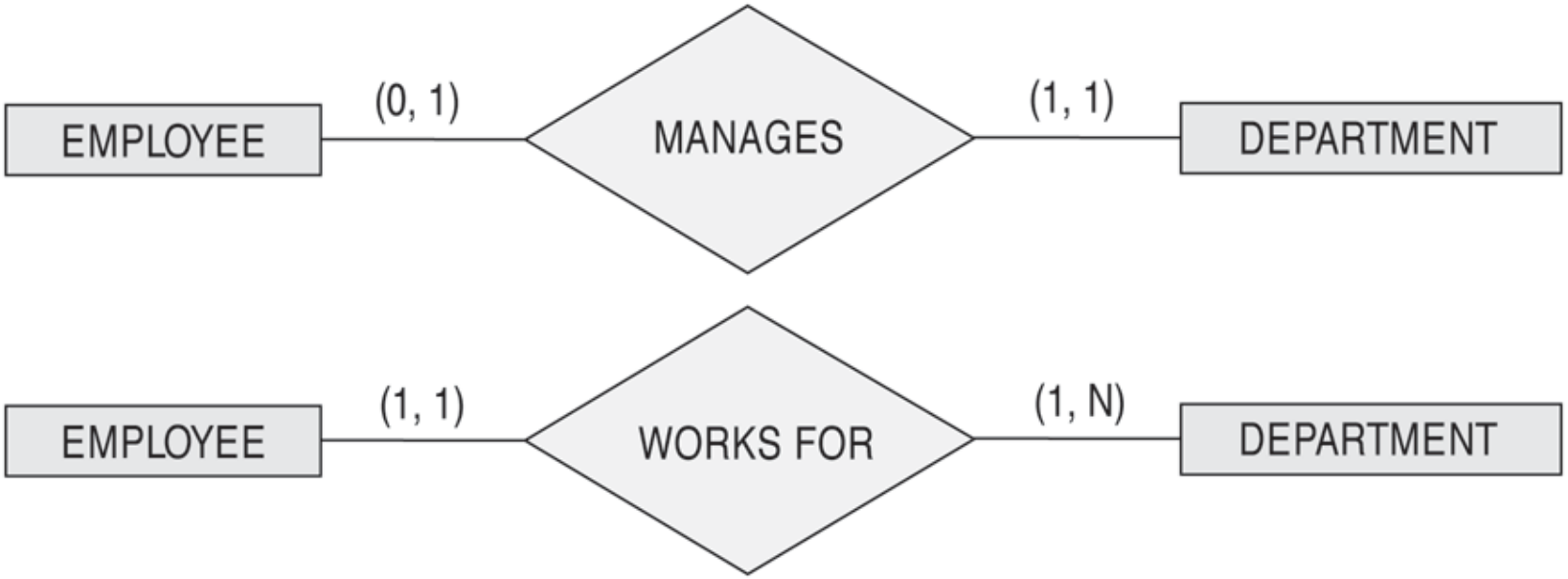
Recursive Relationship Type is: SUPERVISION (participation role names are shown)



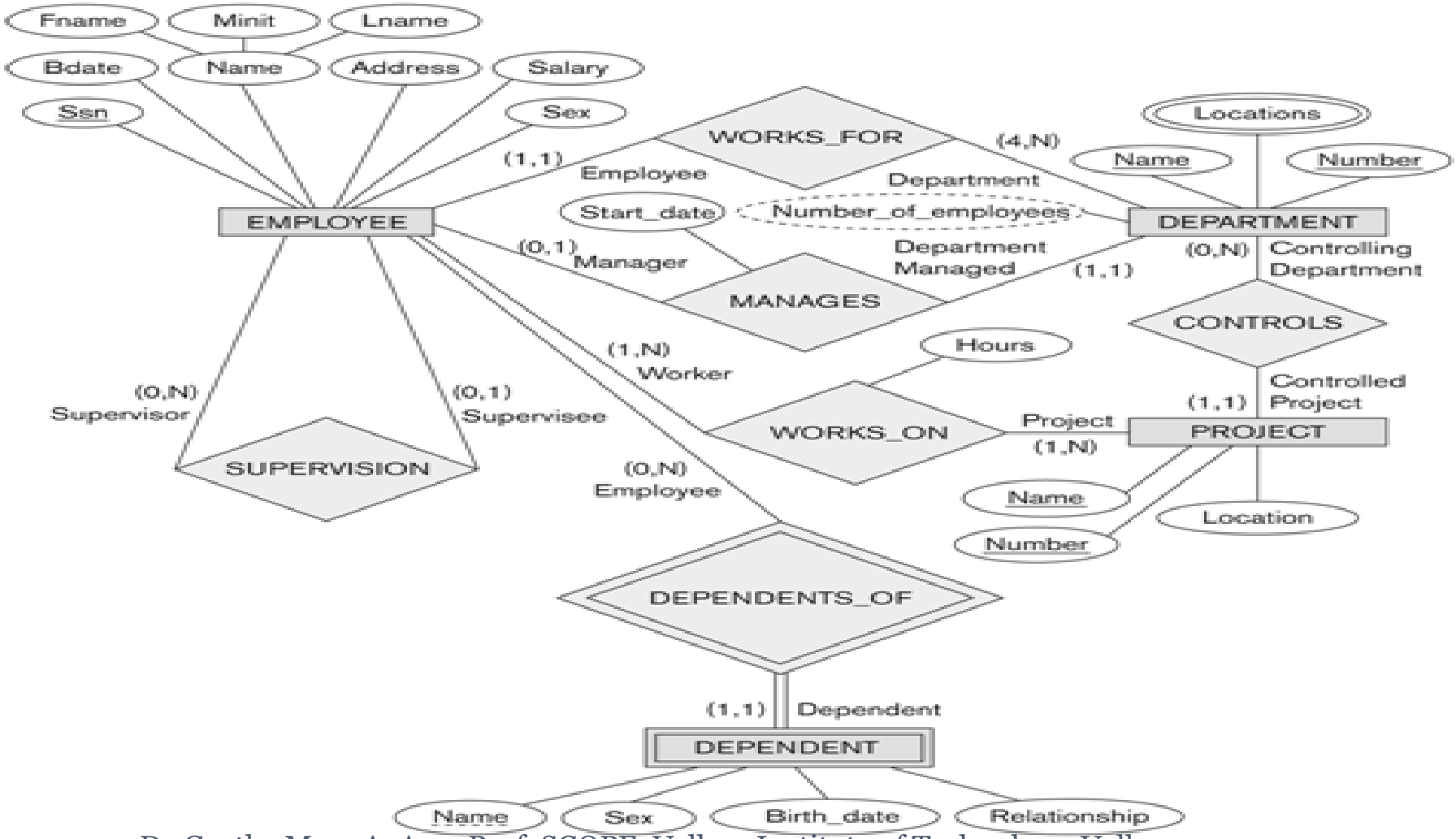
Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- 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

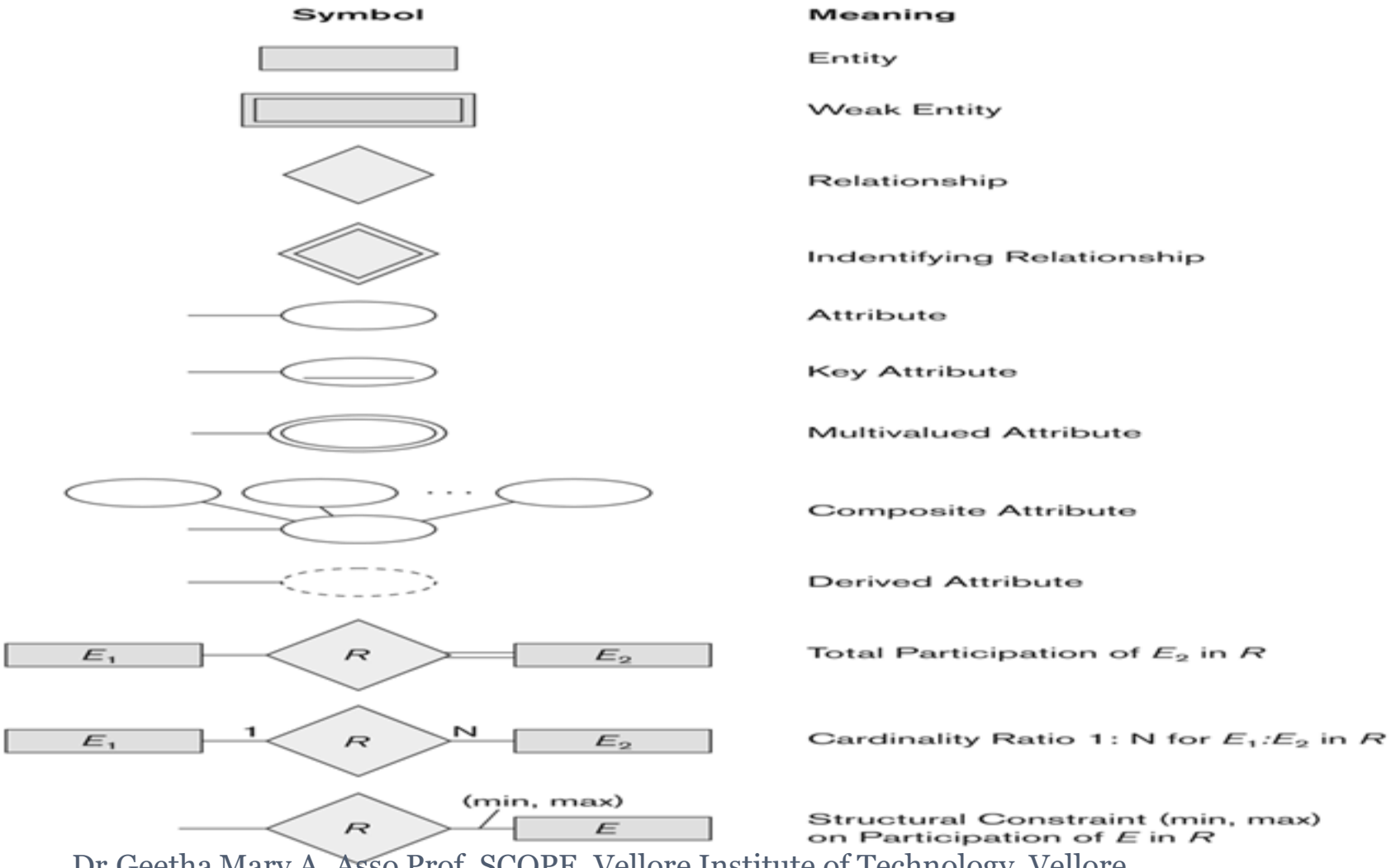
The (min,max) notation for relationship constraints



COMPANY ER Schema Diagram using (min, max) notation



Summary of notation for ER diagrams



Alternative diagrammatic notation

- ER diagrams is one popular example for displaying database schemas
- Many other notations exist in the literature and in various database design and modeling tools
- UML class diagrams is representative of another way of displaying ER concepts that is used in several commercial design tools

EER Model

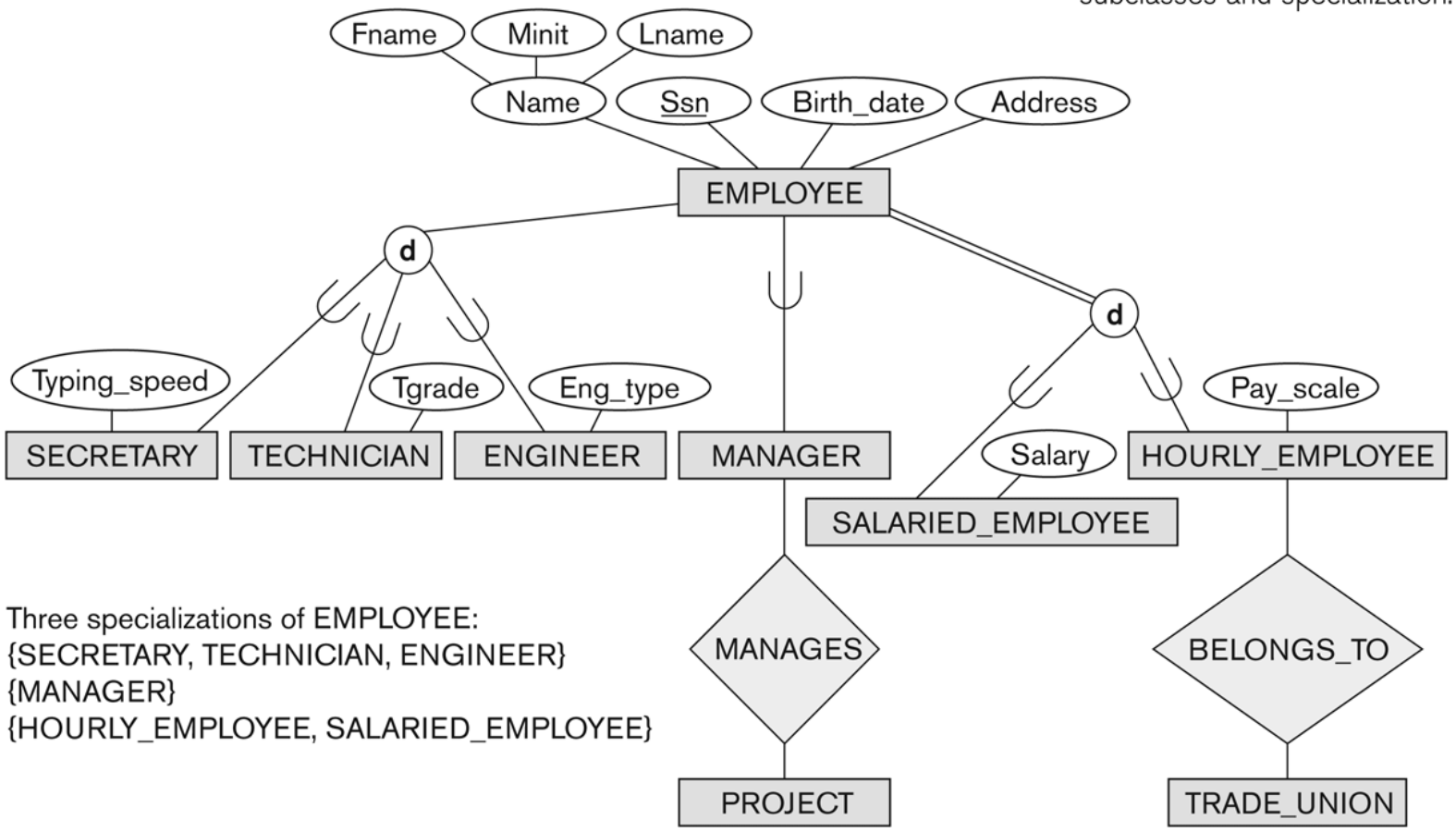
Dr.Geetha Mary A, Asso Prof, SCOPE, Vellore Institute of Technology, Vellore

Subclasses and Superclasses (1)

- 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*

Subclasses and Superclasses

Figure 4.1
EER diagram notation to represent subclasses and specialization.



Subclasses and Superclasses (2)

- 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
- These are called superclass/subclass relationships:
  - EMPLOYEE/SECRETARY
  - EMPLOYEE/TECHNICIAN
  - EMPLOYEE/MANAGER
  - …

Subclasses and Superclasses (3)

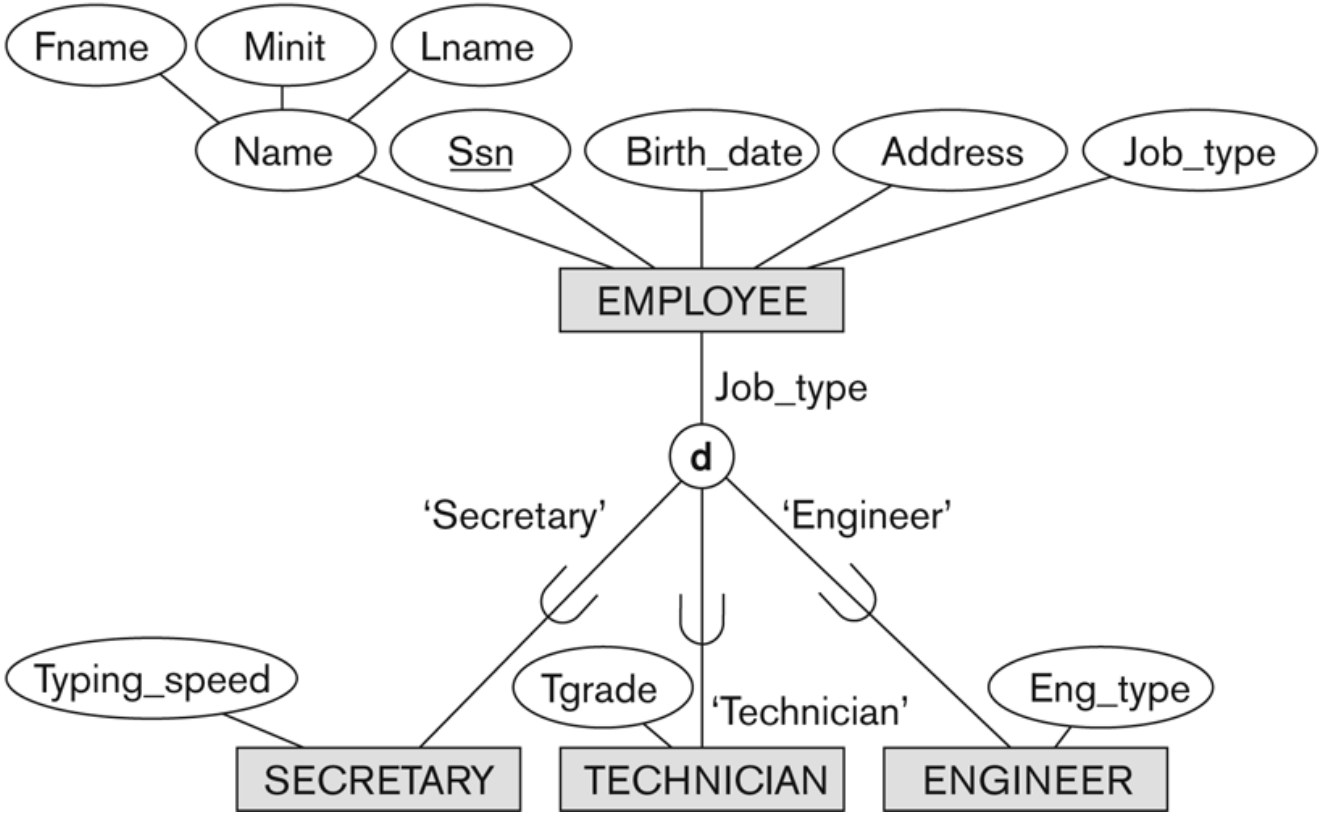
- These are also called IS-A relationships
  - SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, ….
- Note: An entity that is member of a subclass represents the same real-world entity as some member of the superclass:
  - The subclass member is the same entity in a ***distinct specific role***
  - An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
  - A member of the superclass can be optionally included as a member of any number of its subclasses

Subclasses and Superclasses (4)

- 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

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 (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

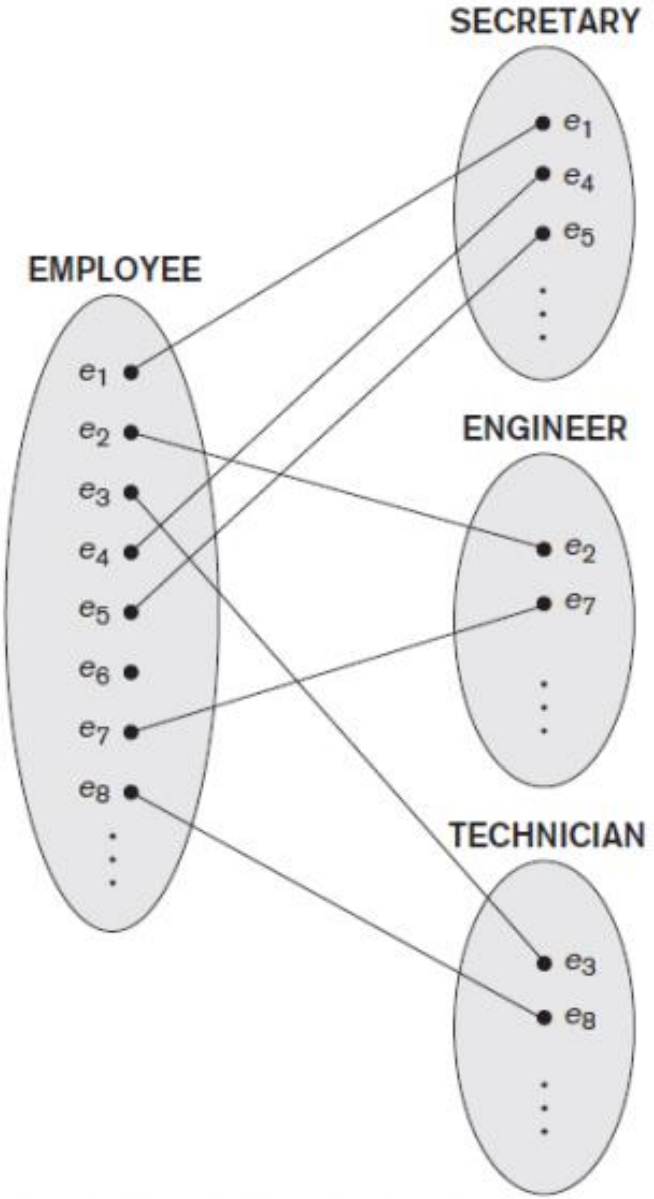
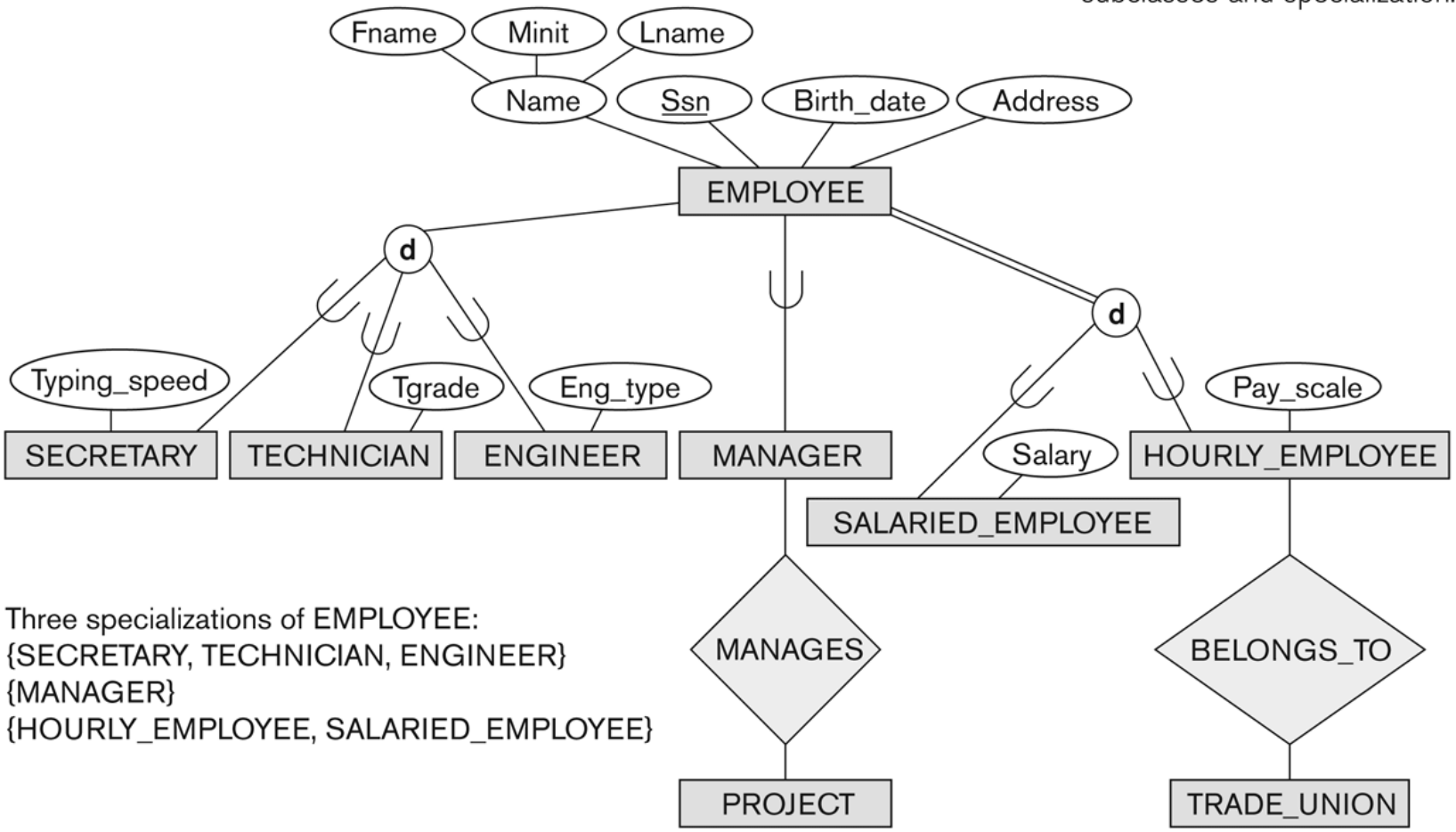


Figure 8.2
Instances of a specialization.

Specialization (3)

Figure 4.1
EER diagram notation to represent subclasses and 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 (2)

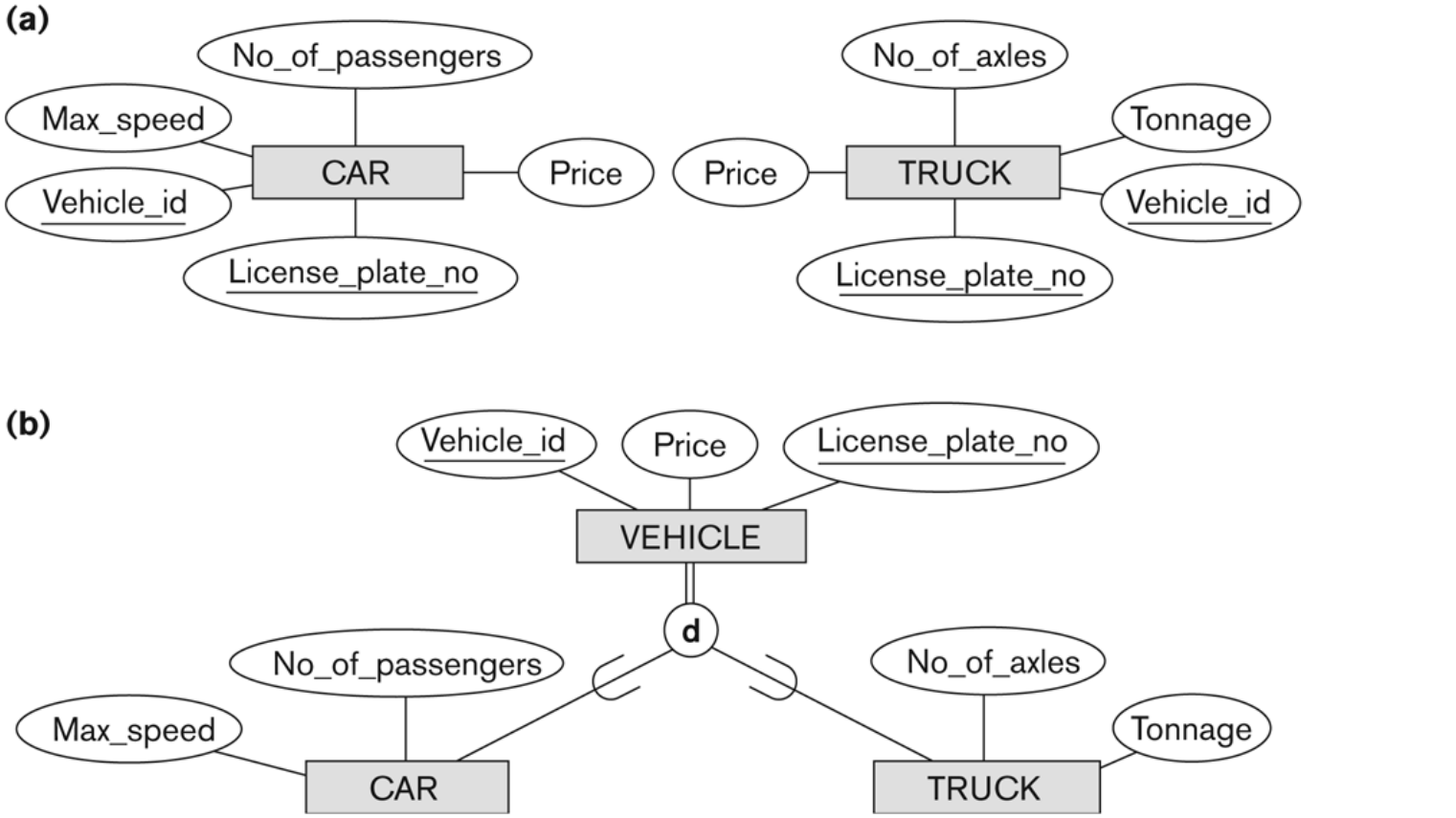


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

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

Constraints on Specialization and Generalization (1)

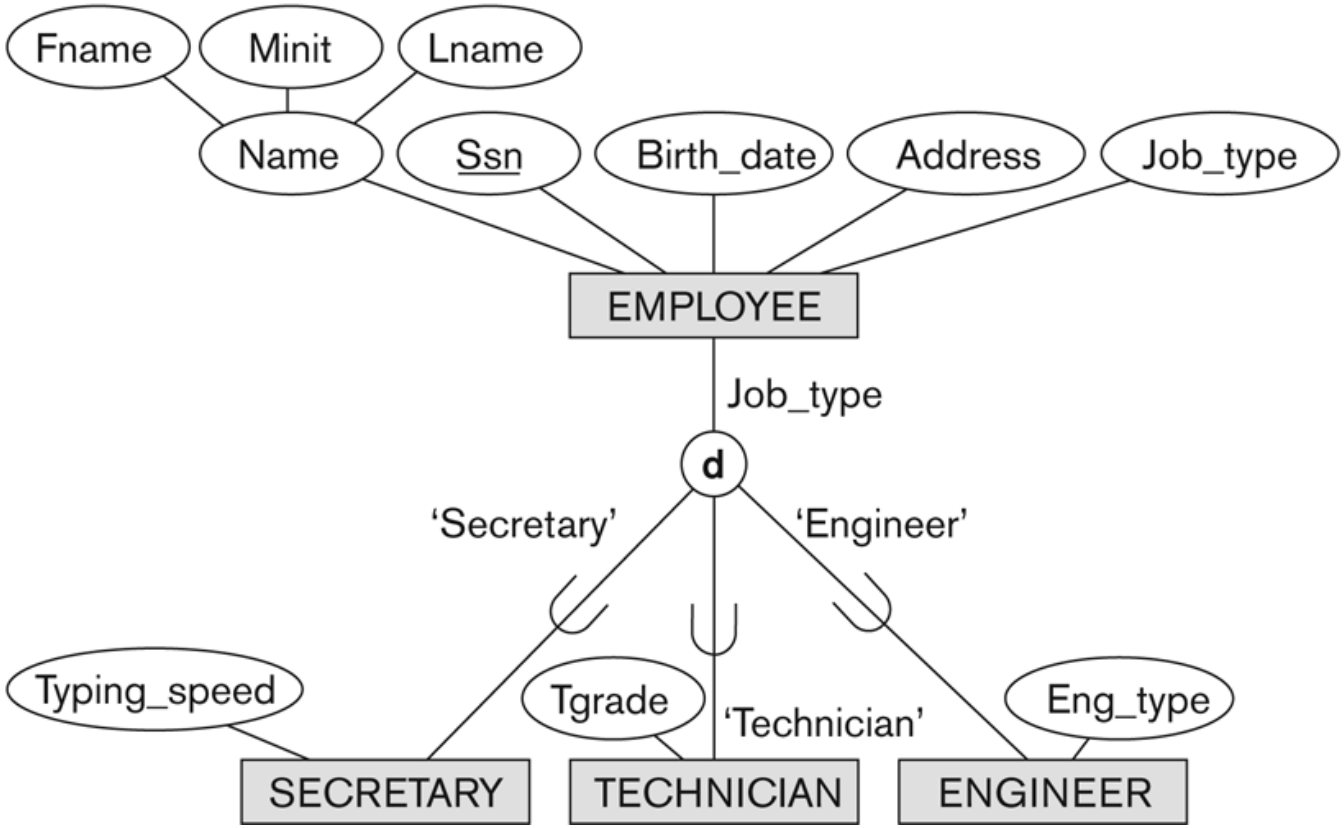
- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called predicate-defined (or condition-defined) subclasses
  - Condition is a constraint that determines subclass members
  - Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its superclass

Constraints on Specialization and Generalization (2)

- If all subclasses in a specialization have membership condition on same attribute of the superclass, specialization is called an **attribute-defined specialization**
  - Attribute is called the defining attribute of the specialization
  - Example: JobType is the defining attribute of the specialization {SECRETARY, TECHNICIAN, ENGINEER} of EMPLOYEE
- If no condition determines membership, the subclass is called **user-defined**
  - Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
  - Membership in the subclass is specified individually for each entity in the superclass by the user

Displaying an attribute-defined specialization in EER diagrams

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



Constraints on Specialization and Generalization (3)

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

Example of overlapping total Specialization

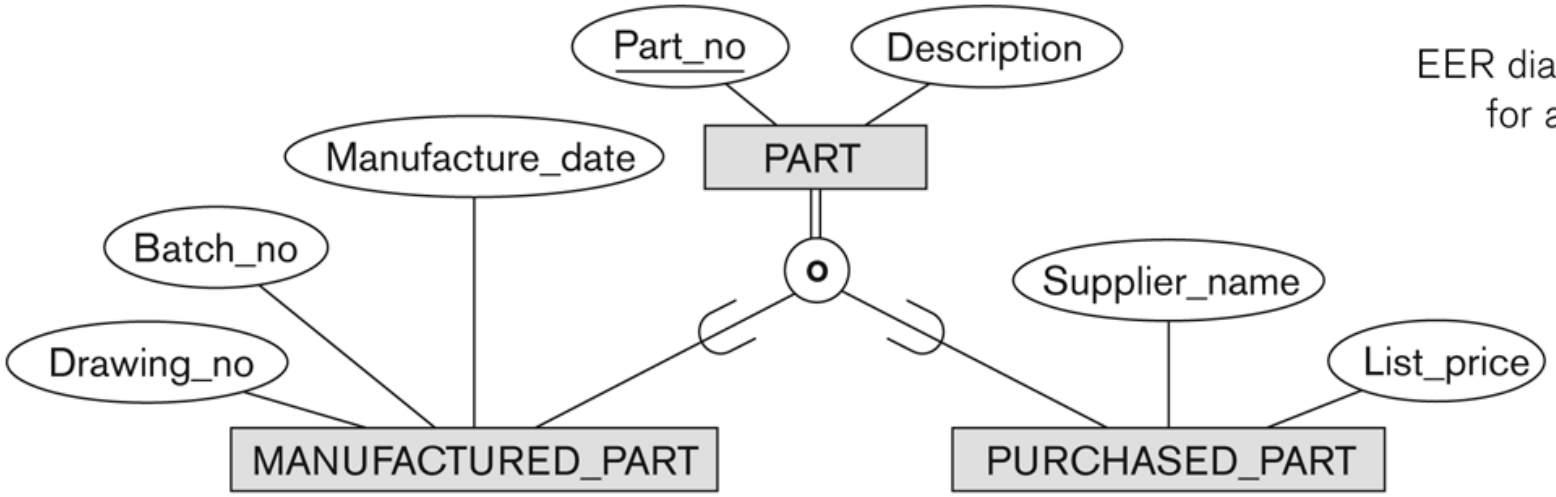


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Constraints on Specialization and Generalization (5)

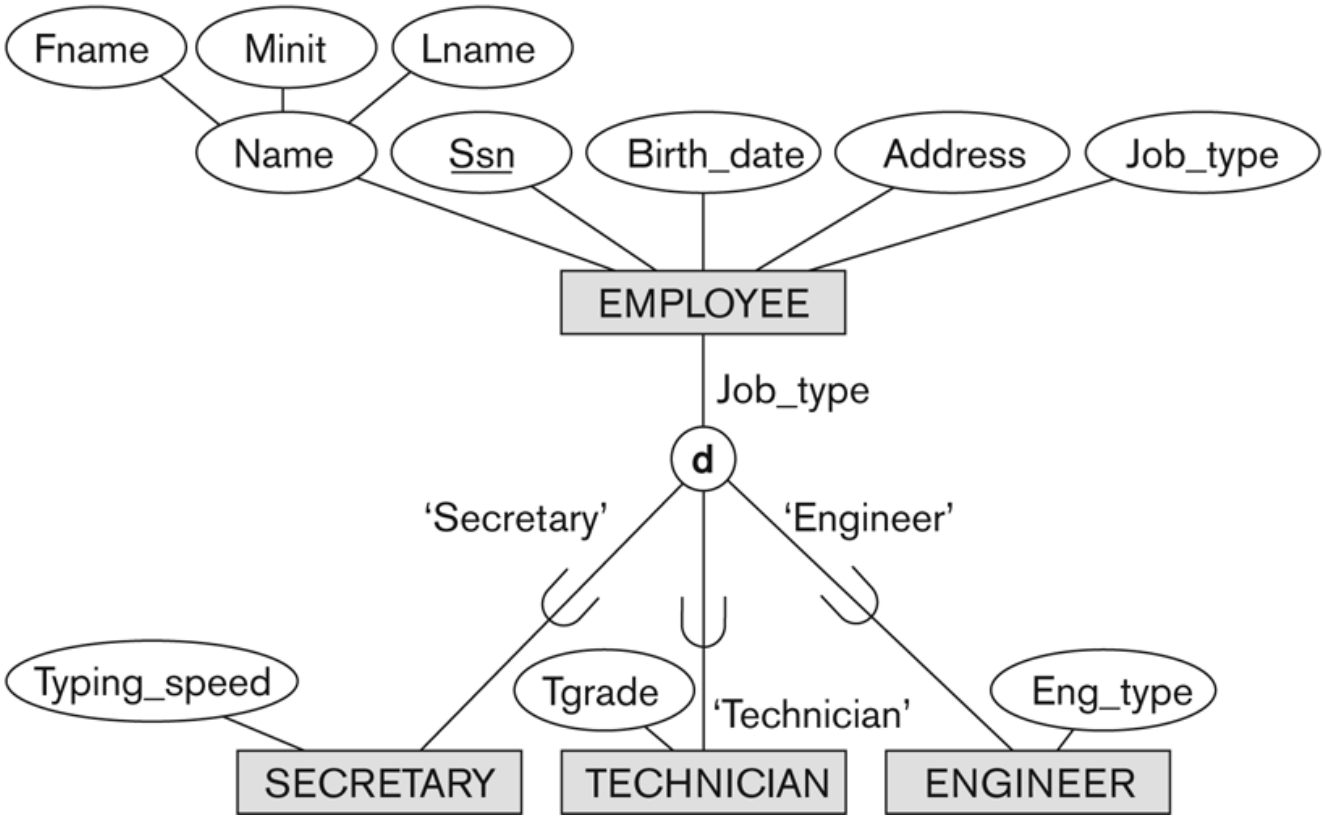
- Completeness Constraint:
  - *Total* specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
  - Shown in EER diagrams by a ***double line***
  - *Partial* allows an entity not to belong to any of the subclasses
  - Shown in EER diagrams 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.

Example of disjoint partial Specialization

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



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”

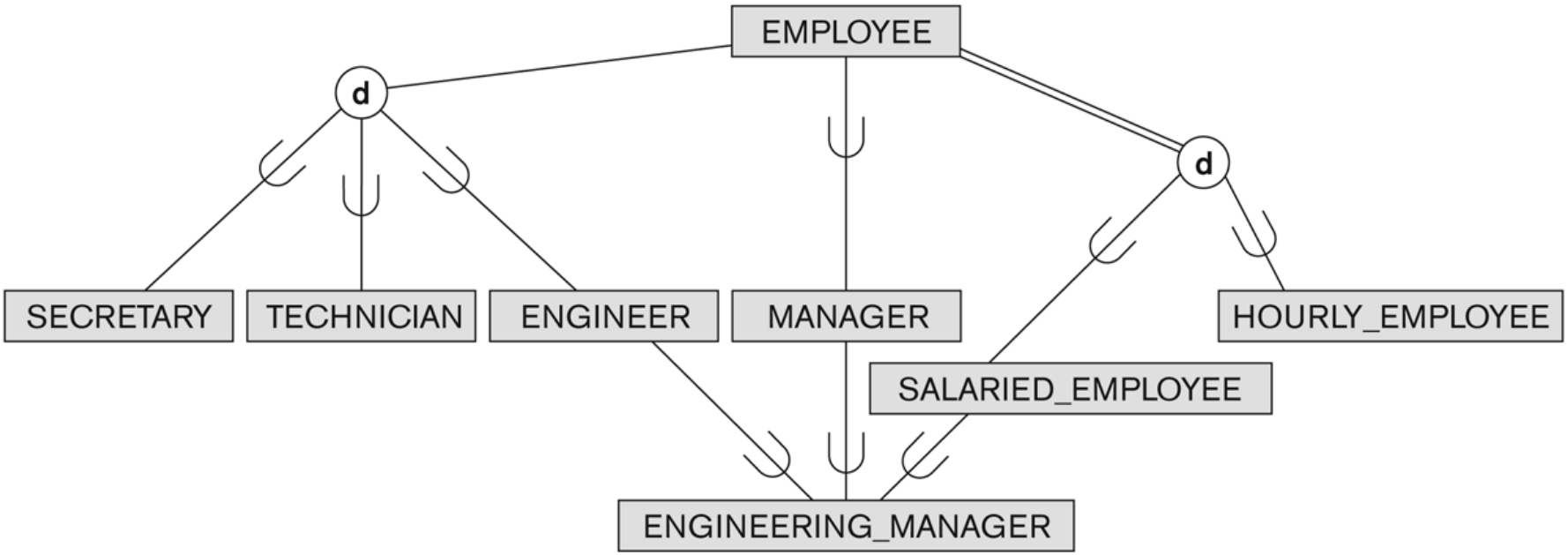


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

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)

Dr.Geetha Mary A, Asso Prof, SCOPE, Vellore Institute of Technology, Vellore

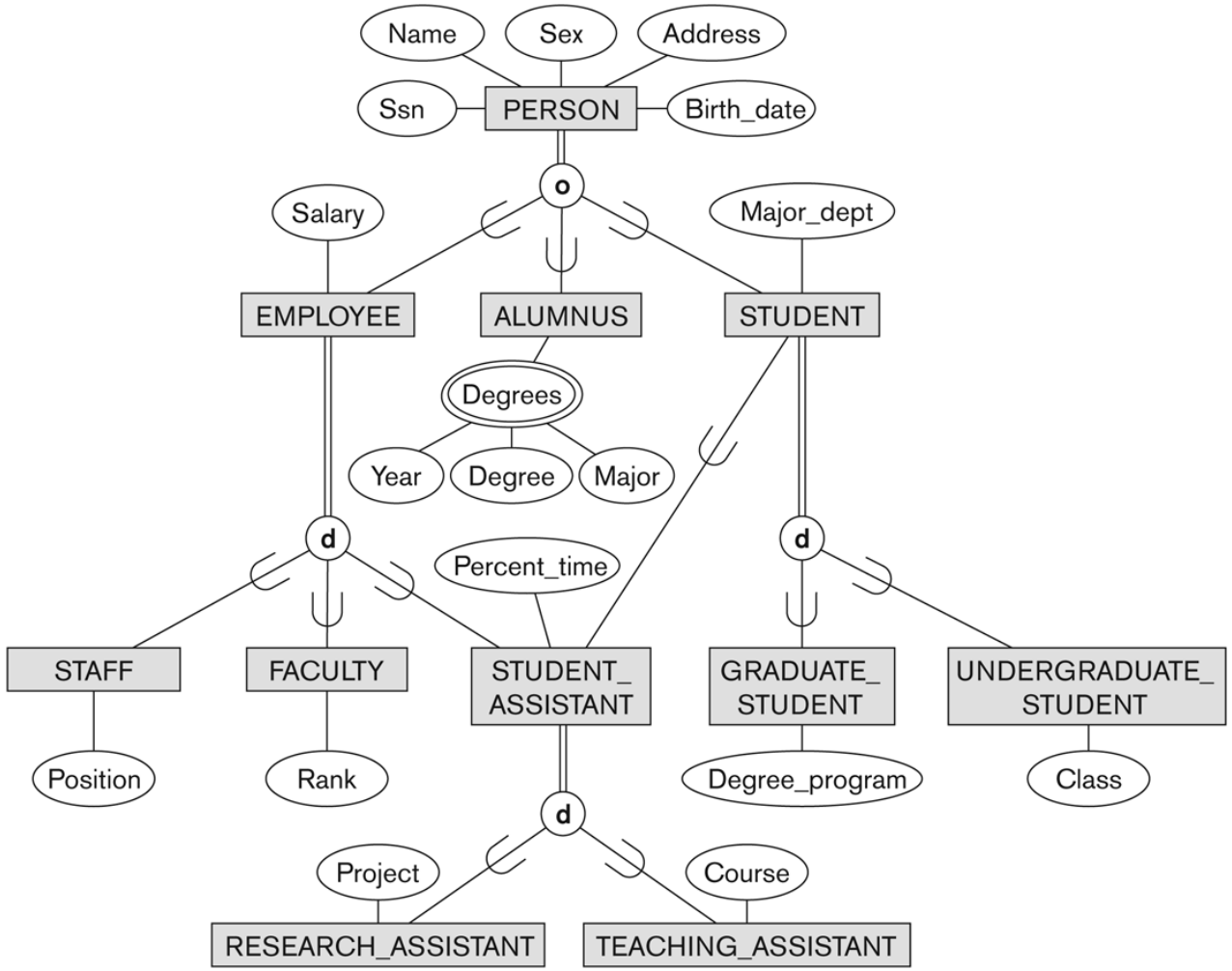


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

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

Two categories (UNION types): OWNER, REGISTERED_VEHICLE

Dr.Geetha Mary A, Asso Prof, SCOPE, Vellore Institute of Technology, Vellore

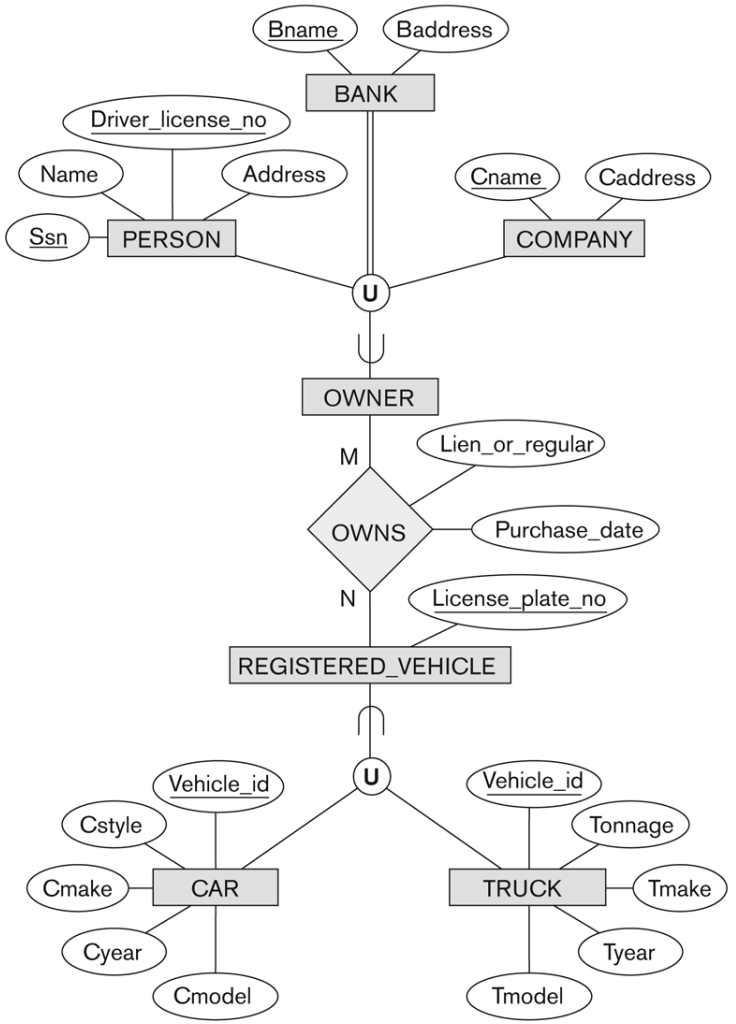


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.