Module 2:
Data Modeling:
Enhanced-Entity-Relationship Model

Prof. Pradnya Bhangale
p.bhangale@somaiya.edu

Unit 2.1

Outline

- Introduction
- Benefits of Data Modeling
- Types of Models
- Phases of Database Modeling
- Entity-Relationship (ER) Model

Introduction

- Building a database system is a complex process that normally requires
  - Analyzing user requirements
  - Design process
  - Concluding with implementation
- During analysis and design, database designer needs to build model of proposed database system
- **Database modeling** is needed to adequately manage database development process
- Some examples of model building: architects, aeronautical engineers, computer architects, traffic engineers
- **A model is an abstract representation of design of complex product, process or situation**

Benefits of Data Modeling

1. **Focusing on essentials**
   - Ignore the details that tend to distract from essential features
   - Ex: modeling a library

2. **Ease of communication and understanding**
   - Helps all parties involved in building a model to reach common understanding of the subject of modeling
   - Communication involves developing documentation
   - Model can be expressed in text or graphical representation or concrete like a prototype

Benefits of Data Modeling (contd..)

3. **Product or Process improvement**
   - Communication between different stake holders lead to improvement in product or process modeling
   - Ex: process being modeled may involve enterprise customers supplying information through forms

4. **Exploring alternatives**
   - Assist in answering ‘what if’ questions
   - Sometimes it is necessary to build prototype to evaluate alternatives

Types of models

1. **Descriptive**
   - Primary purpose to **describe or understand phenomena or complex machinery**
   - Ex: meteorologists, investment companies, school teacher

2. **Prescriptive**
   - Primary purpose is to **clearly specify what a piece of machinery/ software is supposed to do**
   - Ex: Google classroom

3. **Representative**
   - Primary purpose is to **simulate behaviour of some phenomena or machinery**
   - Ex: computer games, face recognition techniques

Phases of Data Modeling/Design

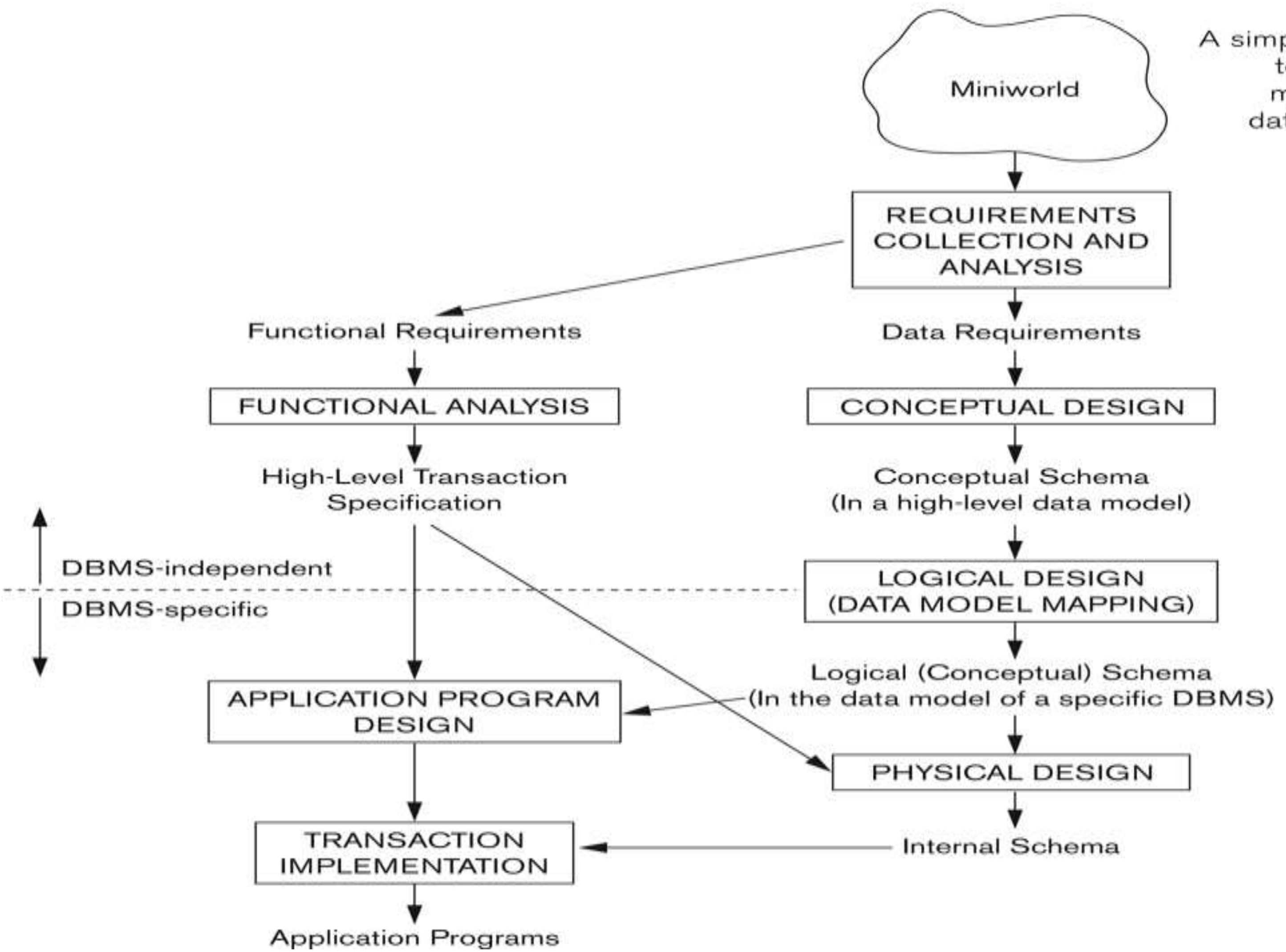


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Phases of Data Modeling(contd..)

- **Phase 1: Requirement Collection and Analysis:**
  - Database Designers interview prospective database users to understand and document their data requirements
  - Requirements should be specified in as detailed
  - In parallel, specify functional requirements of the application(user defined operations) with use of data flow, sequence diagrams etc.
- **Phase 2: Conceptual Schema:**
  - Have concise description of data requirements of users and include detailed description of entity, types, relationships and constraints represented using high level data model
  - High level data model act as reference to ensure that all users data requirements are met and that requirements do not conflict

Phases of Data Modeling(contd..)

- **Phase 3: Logical Design:**
  - Actual implementation of database using commercial DBMS(relational or Object oriented database model)
  - Conceptual schema is transformed from high-level data model to implementation data model
- **Phase 4: Physical Design:**
  - Internal storage structures, indexes, access paths and file organization for database files are specified
  - In parallel, application programs are designed and implemented as database transactions corresponding to high level transaction specification

  - **ER model concepts for conceptual schema is introduced**

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, …

Types of Attributes

- **Simple**
  - Each entity has a single atomic value for the attribute. For example, SSN or Gender.
- **Composite**
  - The attribute may be composed of several components. For example:
    - Address(Apt#, House#, Street, City, State, ZipCode, Country)
    - Name(FirstName, MiddleName, LastName).
    - Composition may form a hierarchy where some components are themselves composite.
- **Multi-valued**
  - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.
    - Denoted as {Color} or {PreviousDegrees}.

Example of a composite attribute

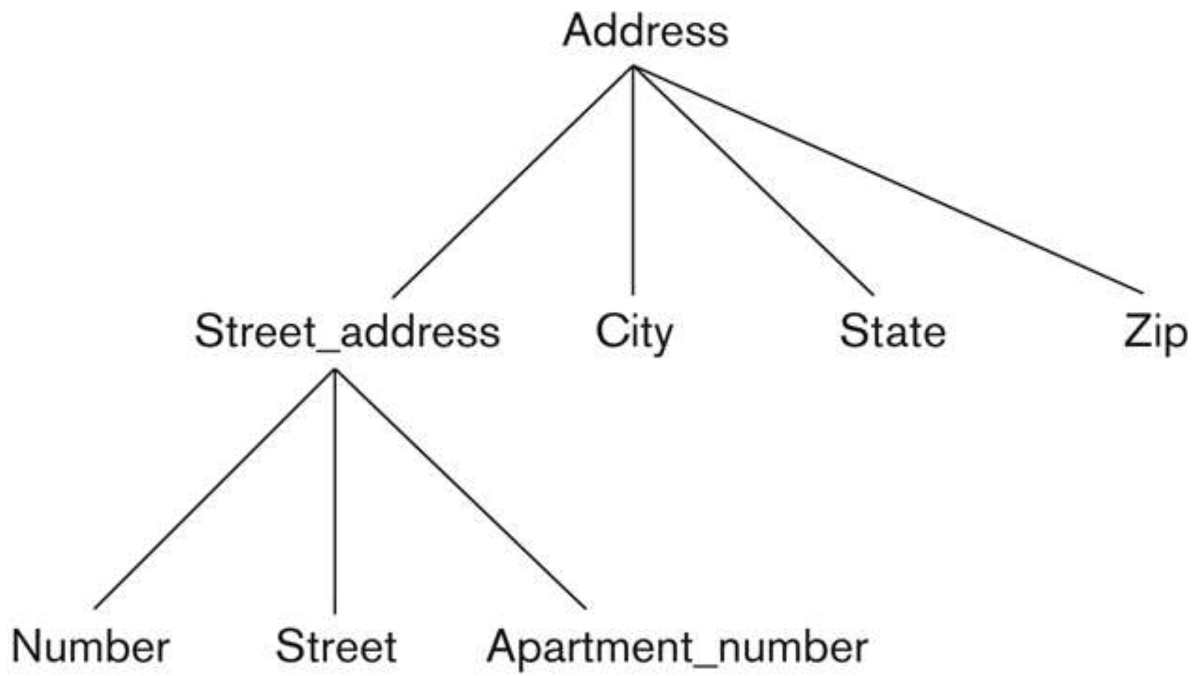


Figure 3.4
A hierarchy of composite attributes.

Types of Attributes (contd..)

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.
  - For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}
  - Multiple PreviousDegrees values can exist
  - Each has four subcomponent attributes:
    - College, Year, Degree, Field

- **Derived attributes**
  - Can be computed from other attributes
  - E.g. *age*, given date of birth

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 (contd..)

- **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), aka license plate number.
- Each key is <u>underlined</u>

Naming Conventions

- **Entities:**
  - Each entity should be a noun, singular or present tense(something about which we want to keep information)
  - First letter should be uppercase
  - Wherever necessary, entities name must be joined using underscore symbol
  - Entity names must be meaningful and must not conflict with other entity names
  - Ex: Account, Customer, Product, Employee, Department, Player etc.

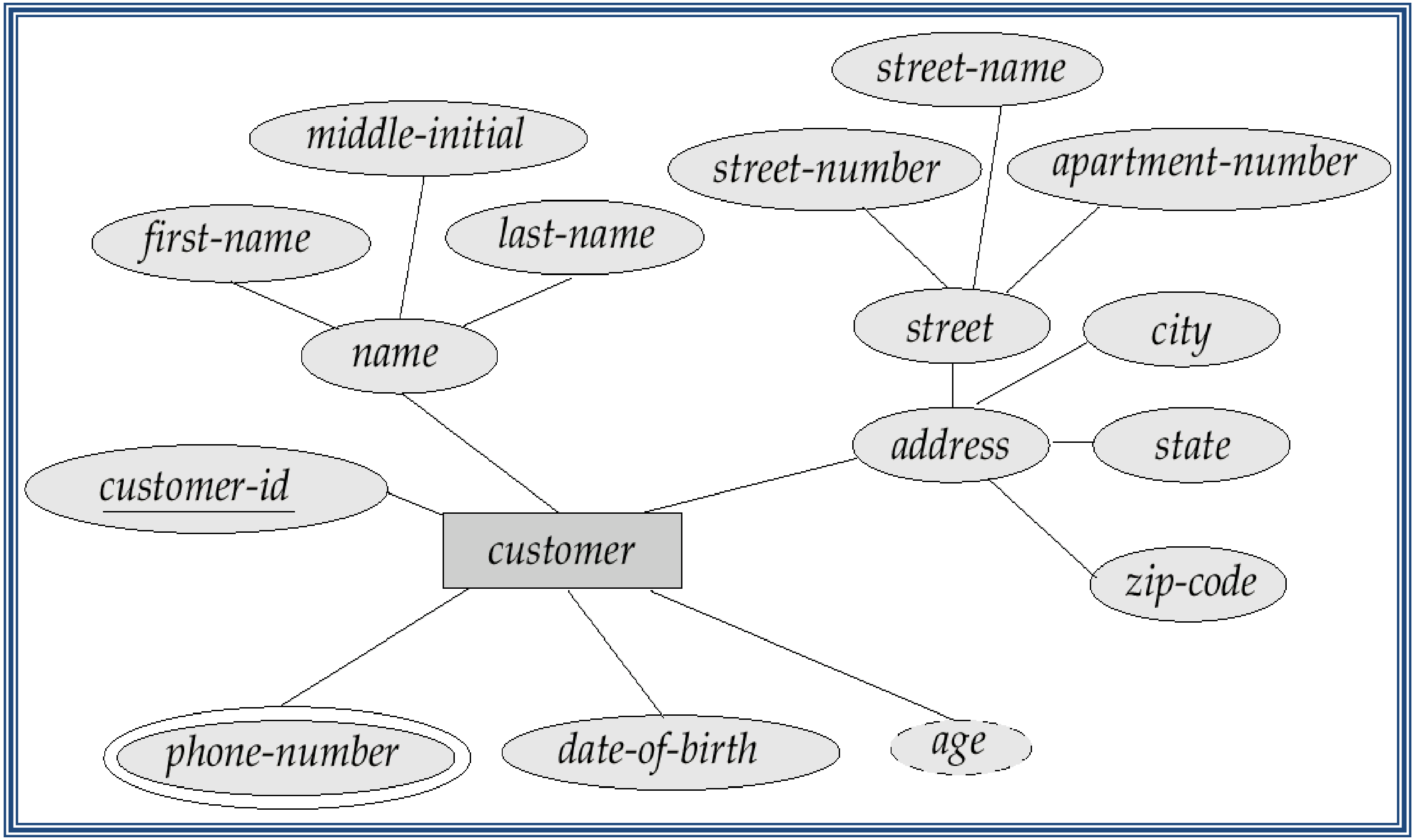
Naming Conventions(Contd…)

- **Attributes:**
  - Attribute name should be noun
  - Where necessary, attribute name can use underscore to join two words
  - Attribute name should be unique. Attributes of different entities can have same name
  - Ex: PID, EmpID, Course_No, Status, Reason, Date_of_Birth etc.

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
  - Derived attributes are represented with dotted oval

E-R Diagram With Composite, Multivalued, and Derived Attributes



Ex: Entity Type CAR with two keys and a corresponding Entity Set

(a)

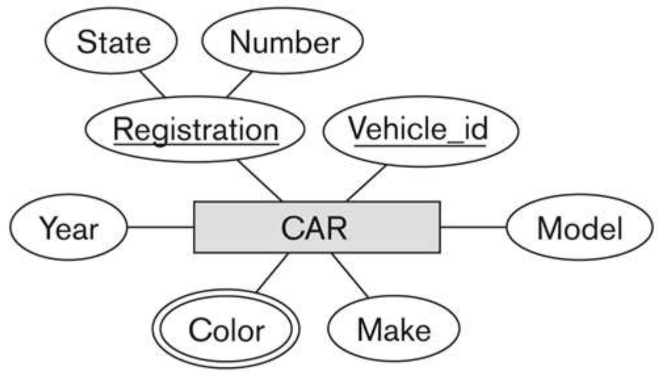


Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(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

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.

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:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

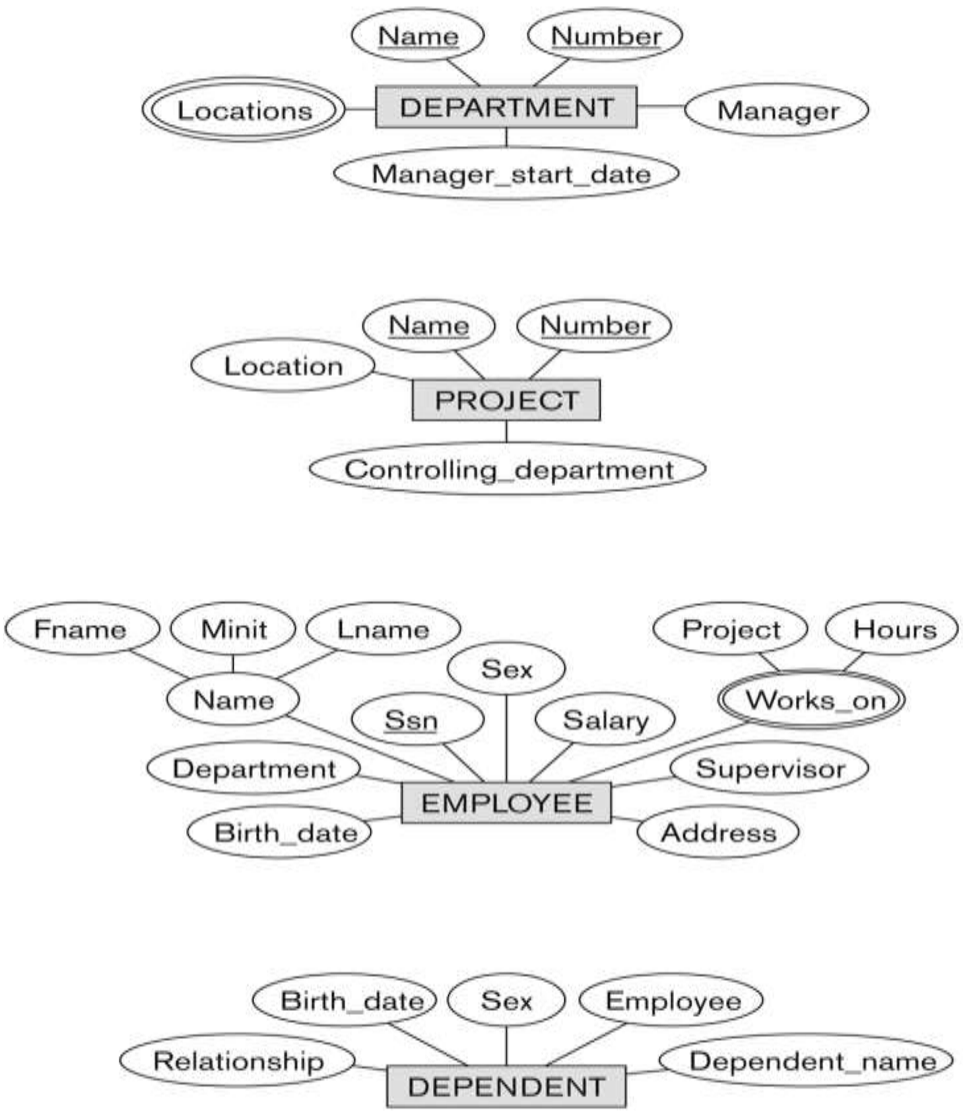


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Example:1

For your convenience, here is a summary of how the Academic Database (School Management System) works:

a. A School/University has many Departments which offer courses to students in a given academic session.

b. Each of these courses is taught by a faculty.

c. Students enroll for different courses in an academic session.

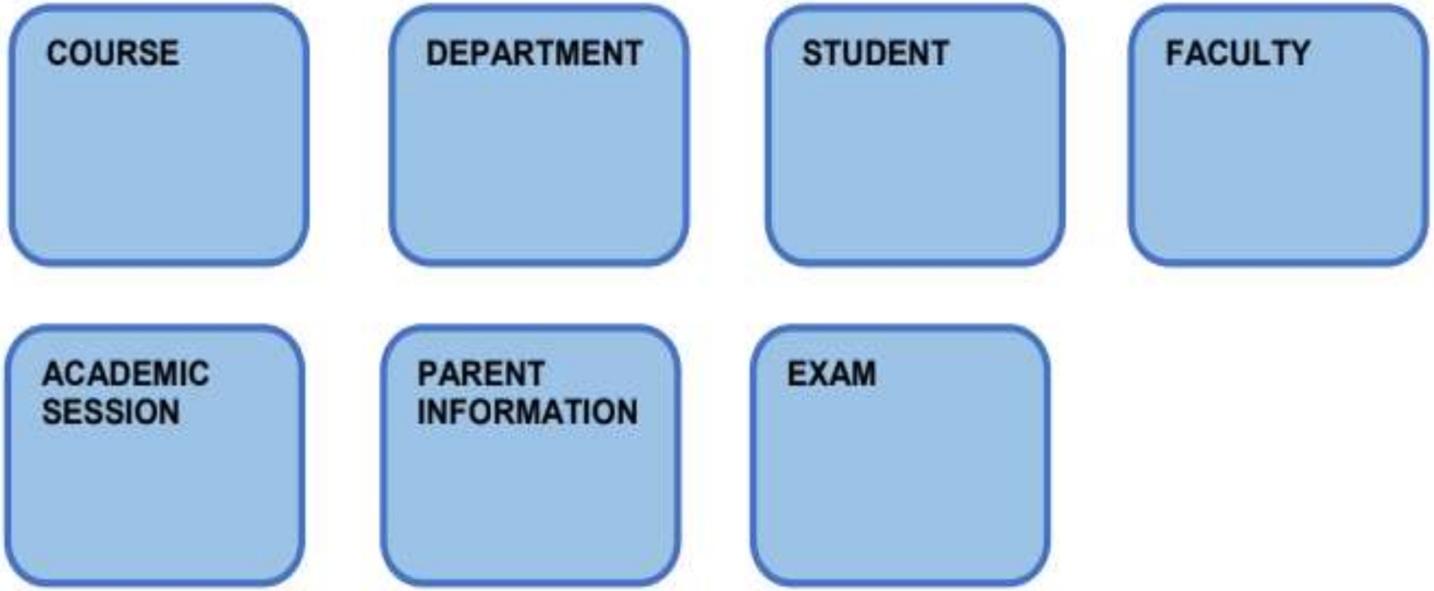
d. Besides the registration details, the parent information of the student also needs to be maintained by the University/School.

e. The Department maintains the student’s attendance details which would decide the eligibility of the student to take up the exams for that academic session.

f. For each academic session, exams are conducted and the results are shared with the student within a stipulated period of time.

g. The Department also maintains a log of the Faculty login and logout time for their reporting needs.

Example 1 Solution: Identified Entities



Example 1 Solution :Identify Attributes for Entities

COURSE
- * ID
- * Name

DEPARTMENT
- * ID
- * Name
- * Head

STUDENT
- * ID
- * First Name
- * Last Name
- * Registration Year
- * Email
- * Number of Working Days
- * Number of Days Off
- o Eligibility for Exam

FACULTY
- * ID
- * First Name
- * Last Name
- * Email
- * Login Date
- * Login Time
- o Details

ACADEMIC SESSION
- * ID
- * Name

EXAM
- * ID
- * Type
- o Start Date
- * Name
- o Description

PARENT INFORMATION
- * ID
- * Parent1 First Name
- * Parent1 Last Name
- o Parent2 First Name
- o Parent2 Last Name

Refining the initial design by introducing relationships

- The initial design is typically not complete
- Some aspects in the requirements will be represented as **relationships**
- ER model has three main concepts:
  - **Entities** (and their entity types and entity sets)
  - **Attributes** (simple, composite, multivalued, derived)
  - **Relationships** (and their relationship types and relationship sets)

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

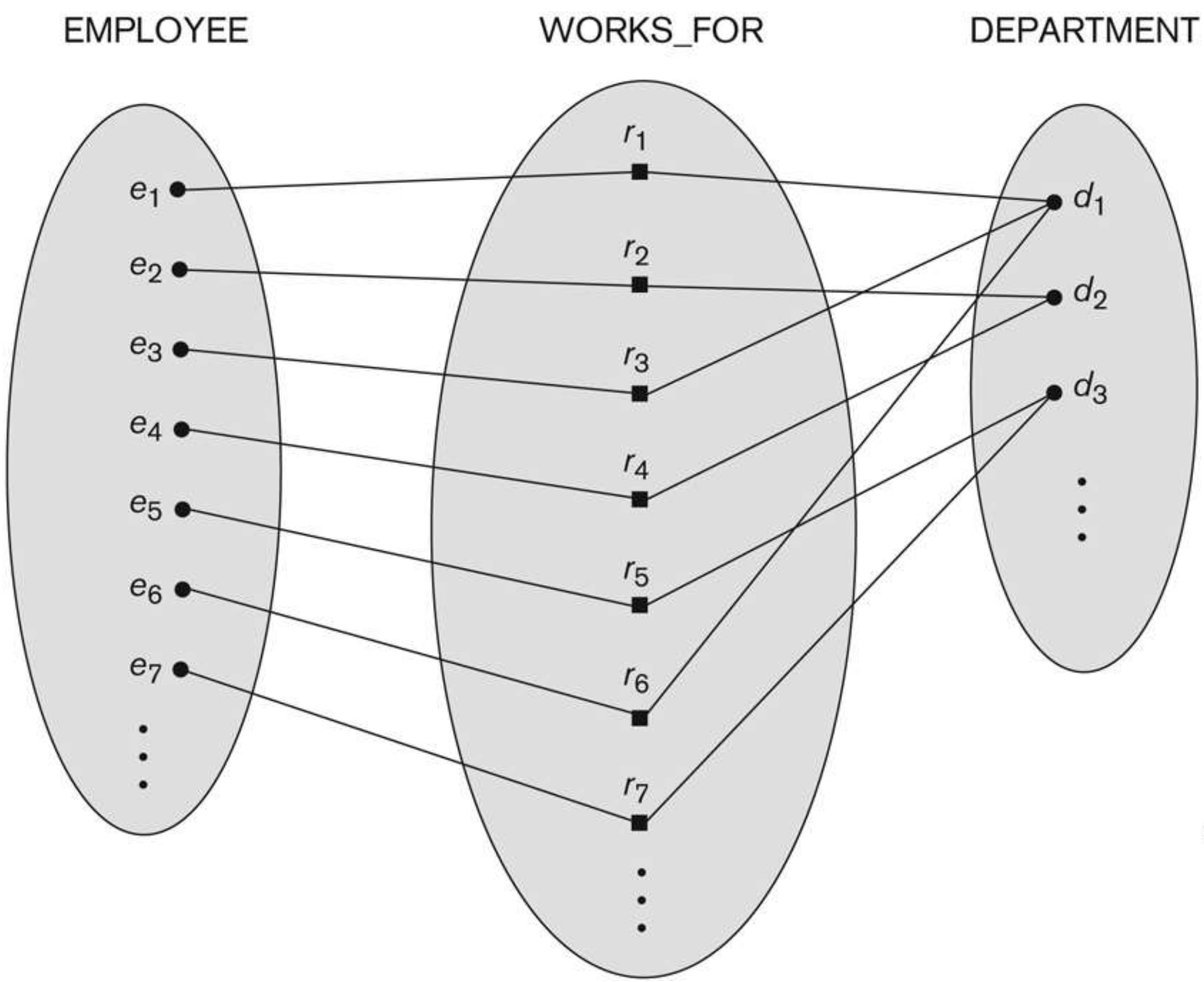


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

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

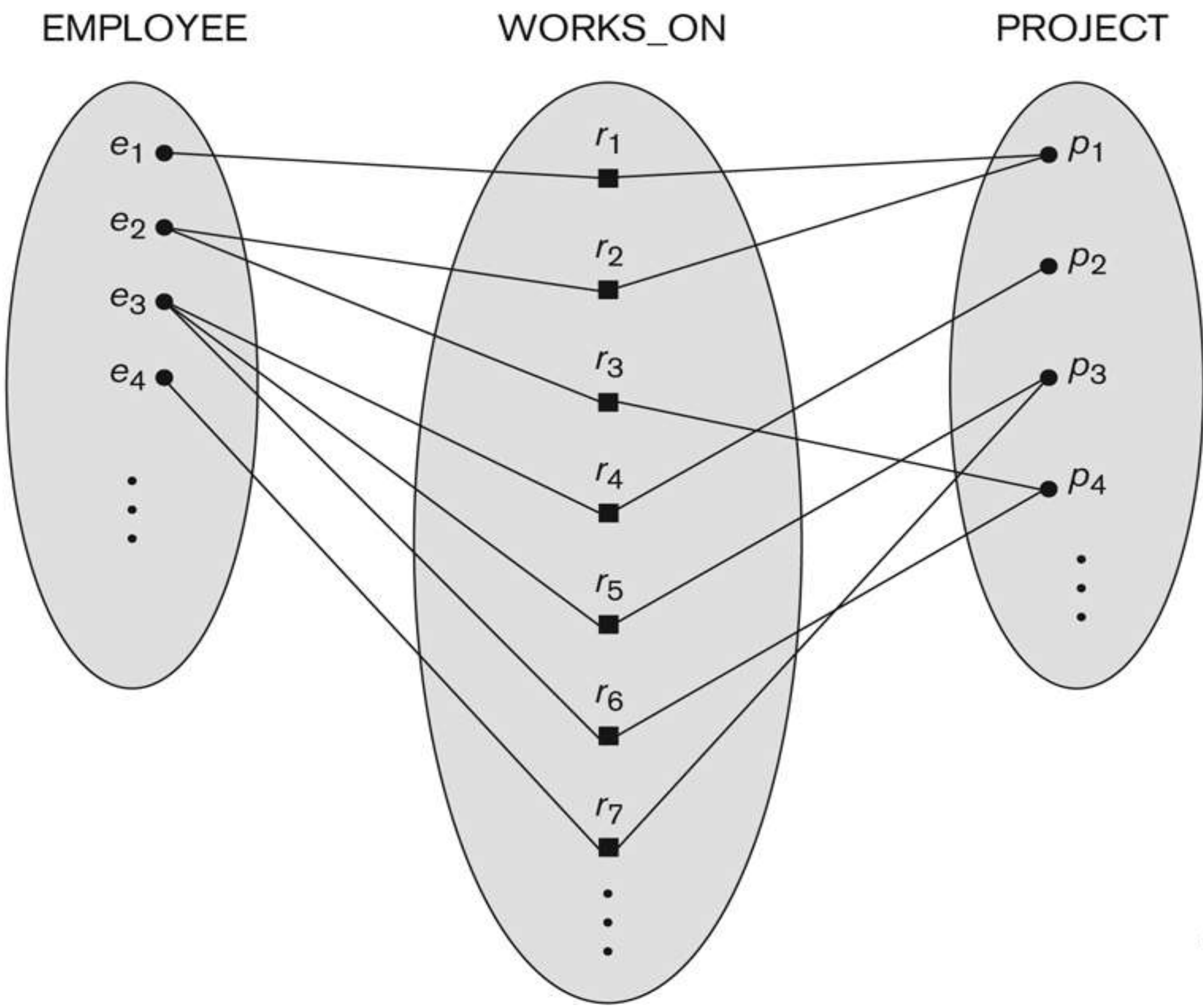


Figure 3.13
An M:N relationship, WORKS_ON.

Relationship type vs. relationship set

- **Relationship Type:**
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints

- **Relationship Set:**
  - The current set of relationship instances represented in the database
  - The current *state* of a relationship type

Relationship type vs. relationship set (contd..)

- Each instance in the set relates individual participating entities – one from each participating entity type
- In ER diagrams, we represent the *relationship type* as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines

Naming Convention(contd..)

- **Relationships:**
  - Each relationship name should be a verb that fits sentence structure
  - Where necessary, relationship name can use underscore to join two words
  - Ex: Occupies, Married_to, Represents, Batting etc.

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.

Refining the COMPANY database schema by introducing 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)

Discussion on Relationship Types

- In general, more than one relationship type can exist between the same participating entity types
  - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
  - Different meanings and different relationship instances.

ER DIAGRAM – Relationship Types are:

WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF

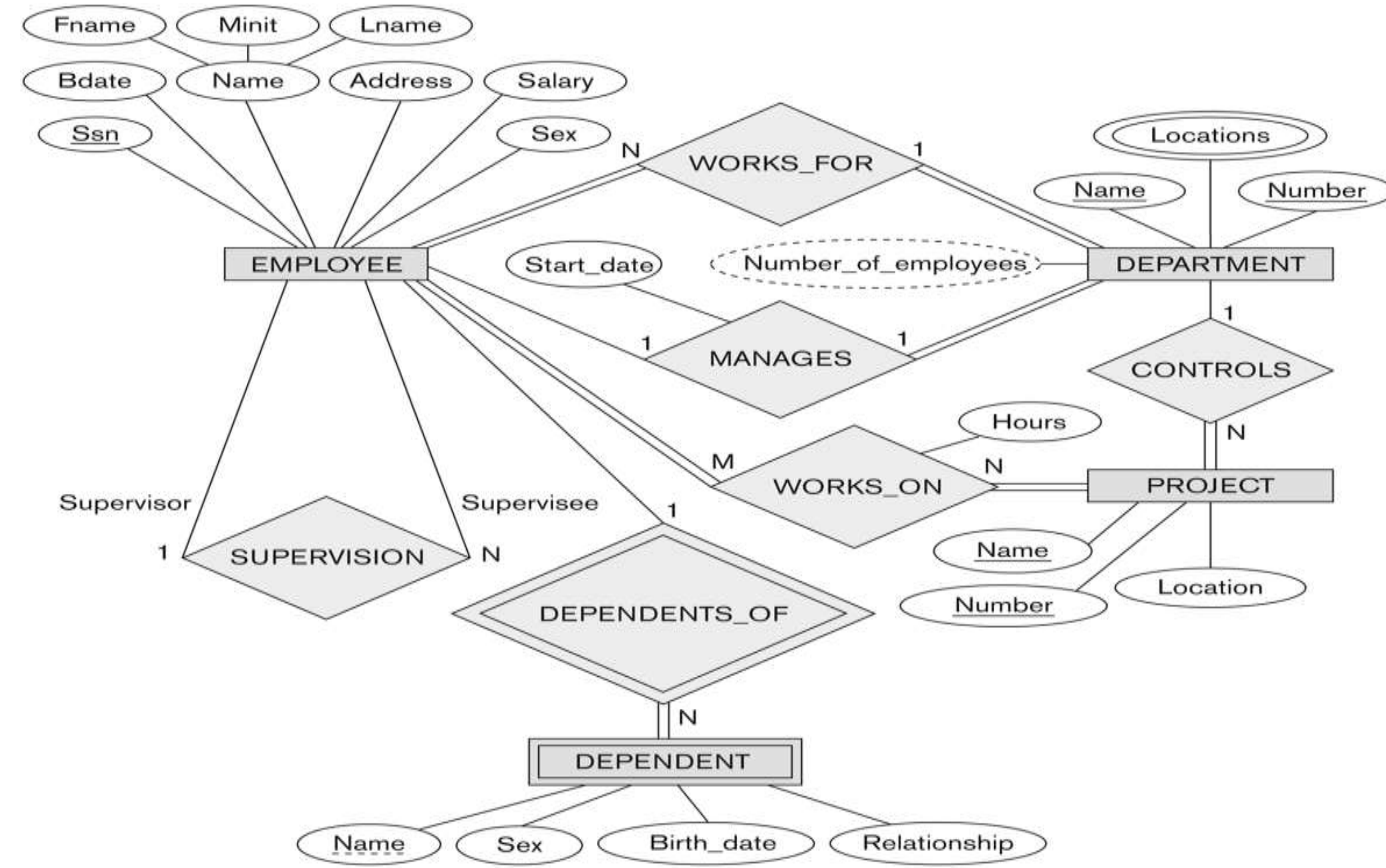


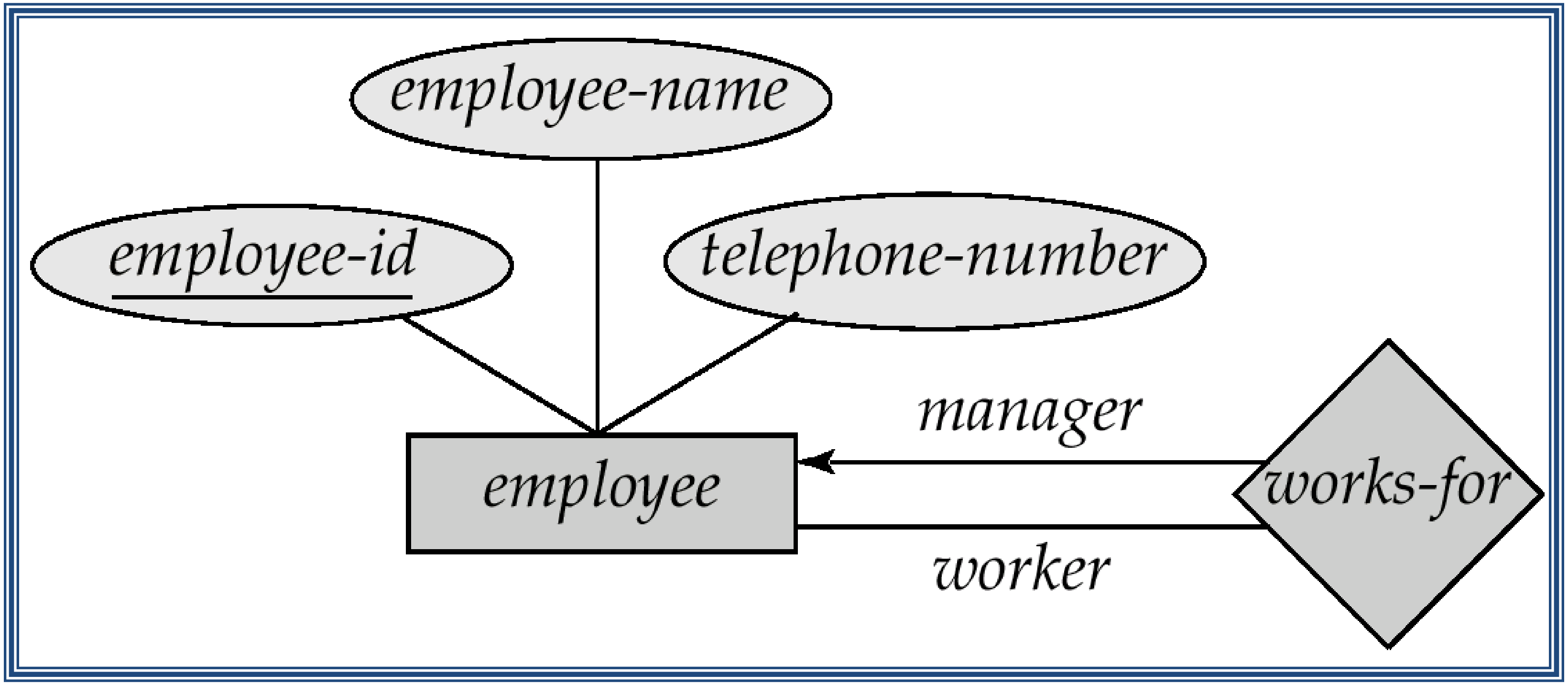
Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Role Names and Recursive Relationships

- Each entity that participates in a relationship type plays a particular role in relationship
- Role name signifies role that participating entity from entity type plays in each relationship instance
- Ex: Works_for relationship type, EMPLOYEE plays role of employee or worker and DEPARTMENT plays role of department or employer
- Role name has significant importance when same entity participates more than once in a relationship type in different roles

Such relationship types are called **recursive or self-referencing relationships**

Ex: Recursive Relationship type



Recursive Relationship Type

- An relationship type whose with the same participating entity type in **distinct roles**
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in *supervisor* role
  - One employee in *supervisee* role

Displaying a recursive relationship

- In a recursive relationship type.
  - Both participations are same entity type in different roles.
  - For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

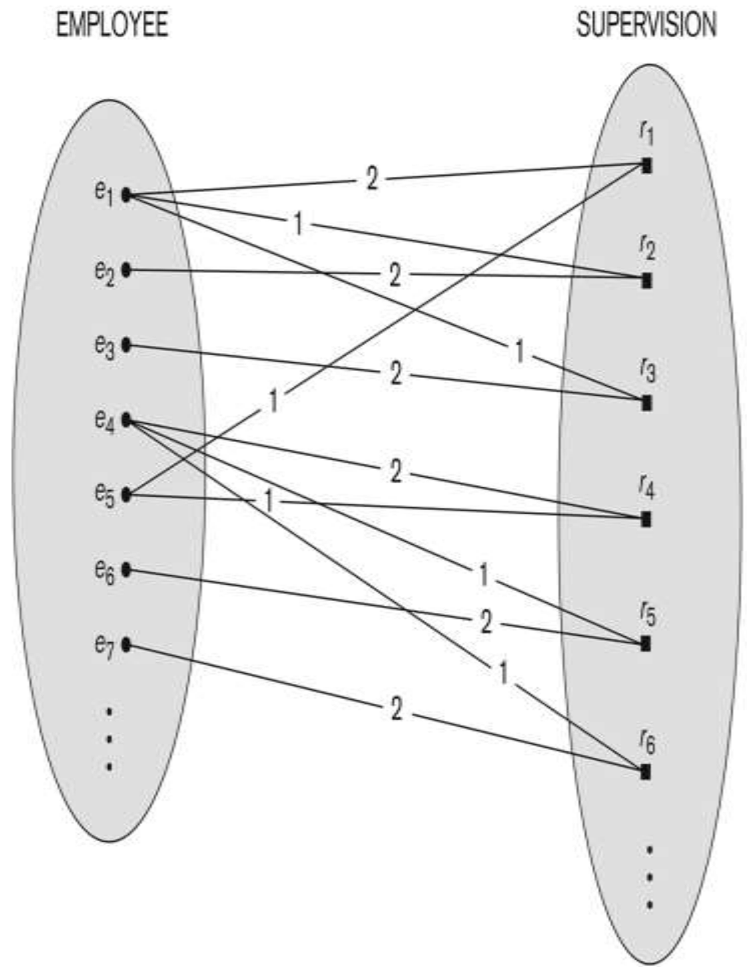


Figure 3
A recursive relati
ship SUPERVISI
between EMPLOY
in the *supervisor*
(1) and EMPLOY
in the *subordin*
role

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

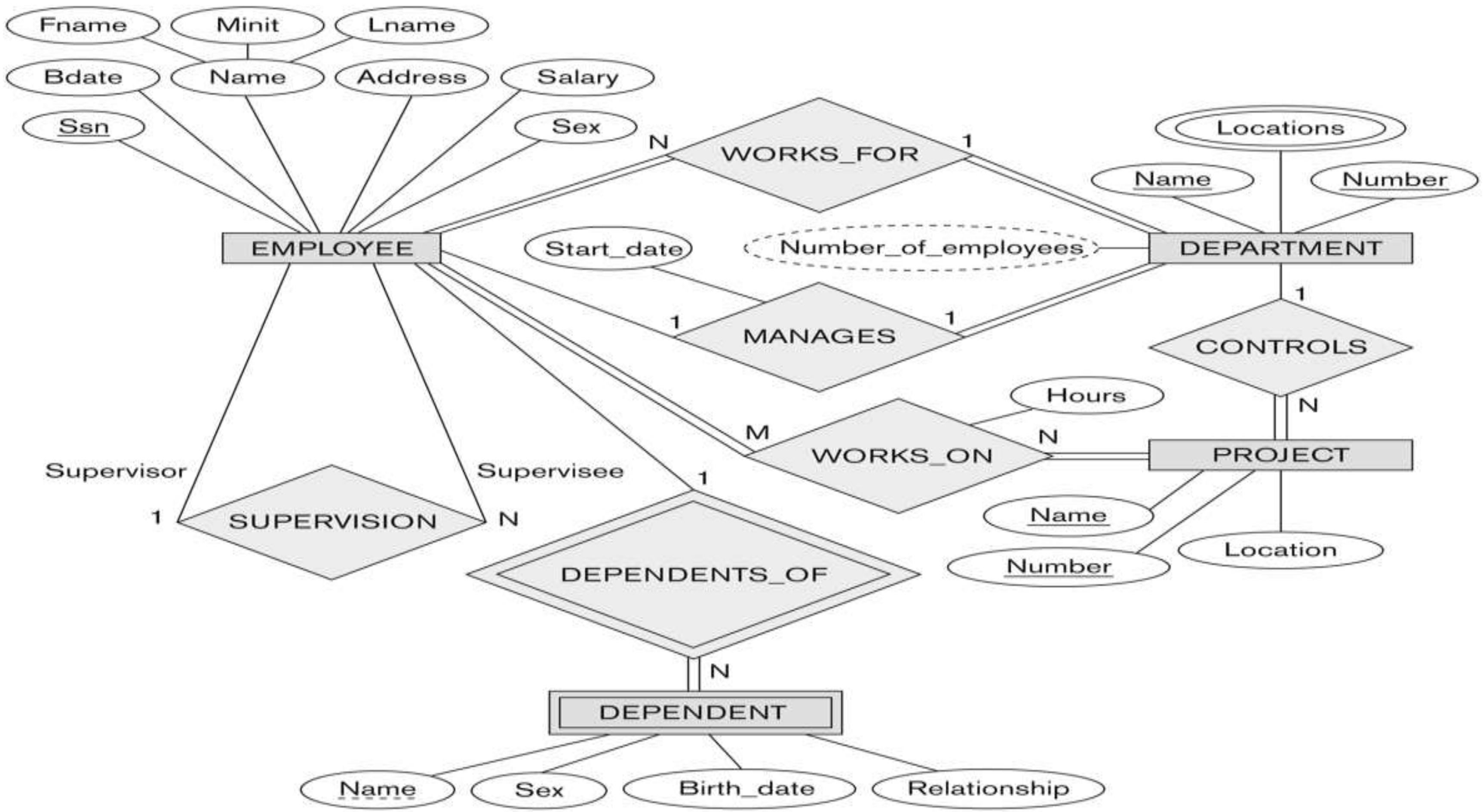


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

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(discriminator 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* represented with double rectangular box
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Example: Weak Entity Set

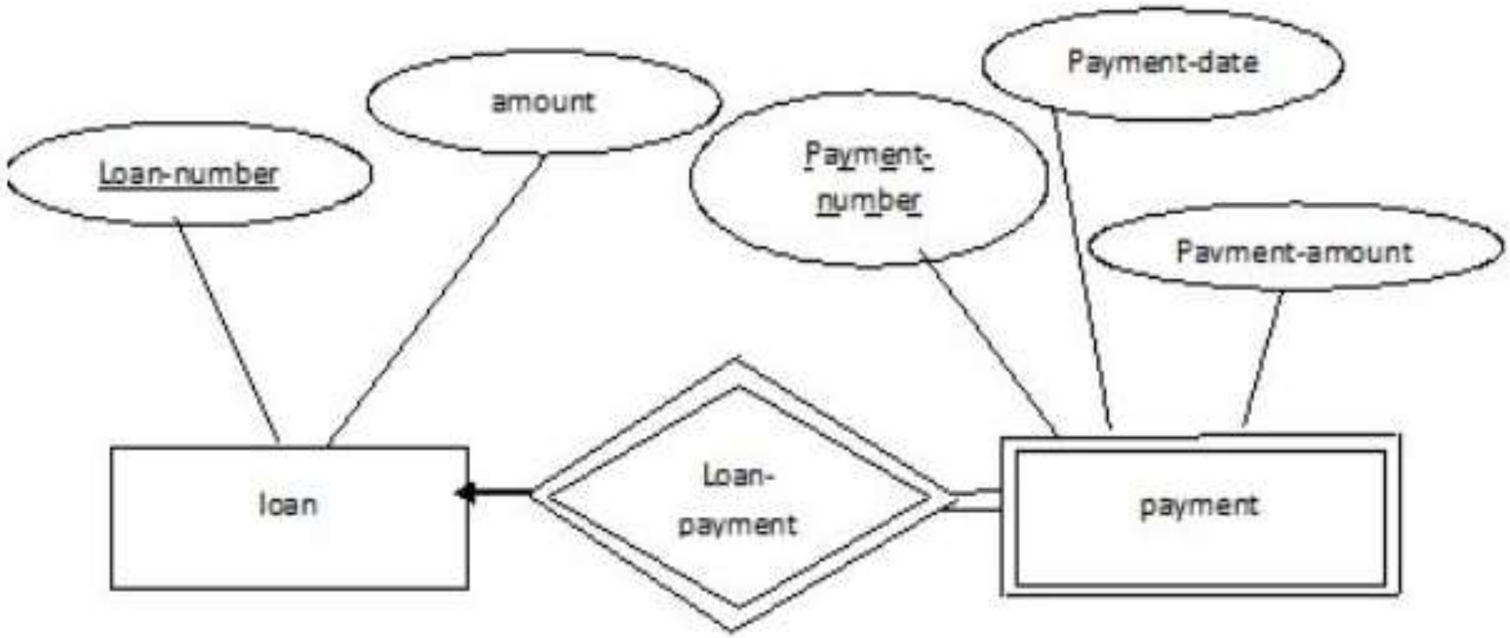
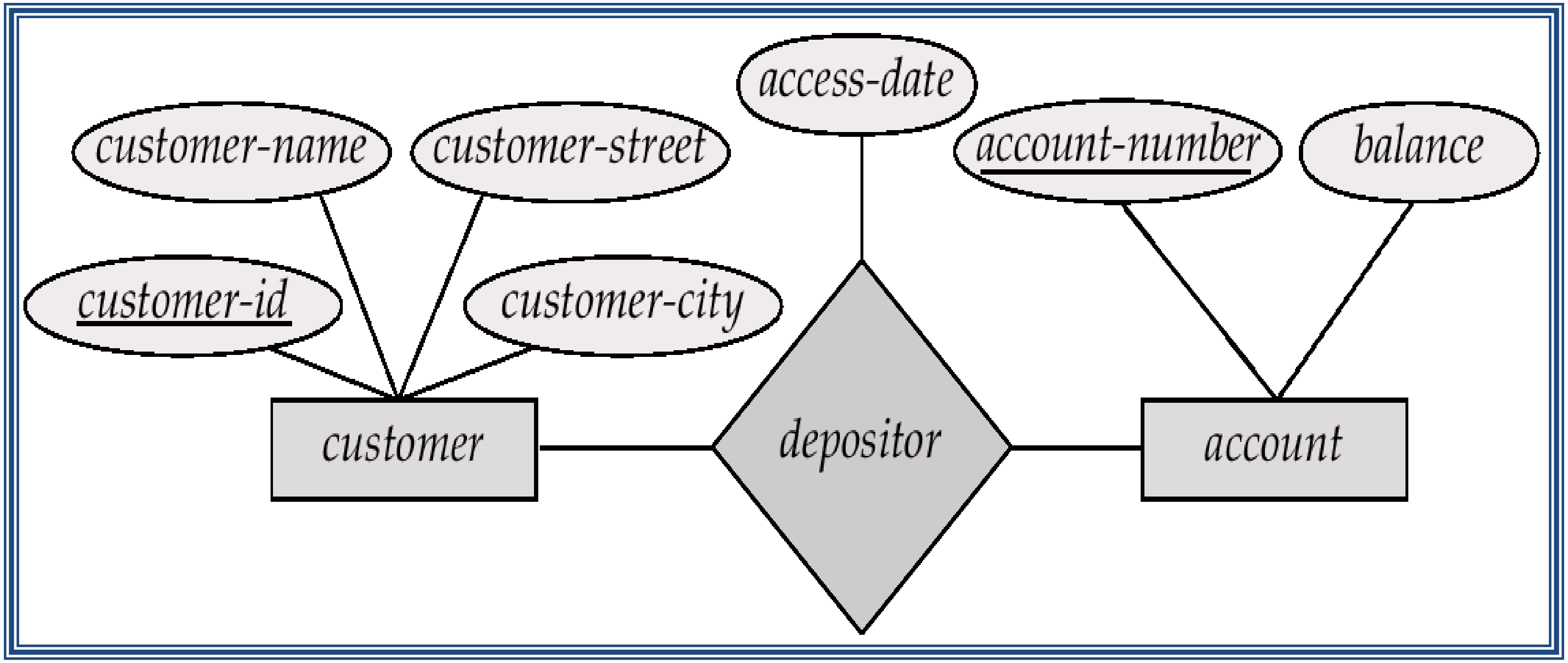


Fig1: Example of weak entity set.

Attributes of Relationship types

- A relationship type can have attributes:
  - For example, HoursPerWeek of WORKS_ON
    - 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

Relationship Sets with Attributes



Example Attribute of a Relationship Type: Hours of WORKS_ON

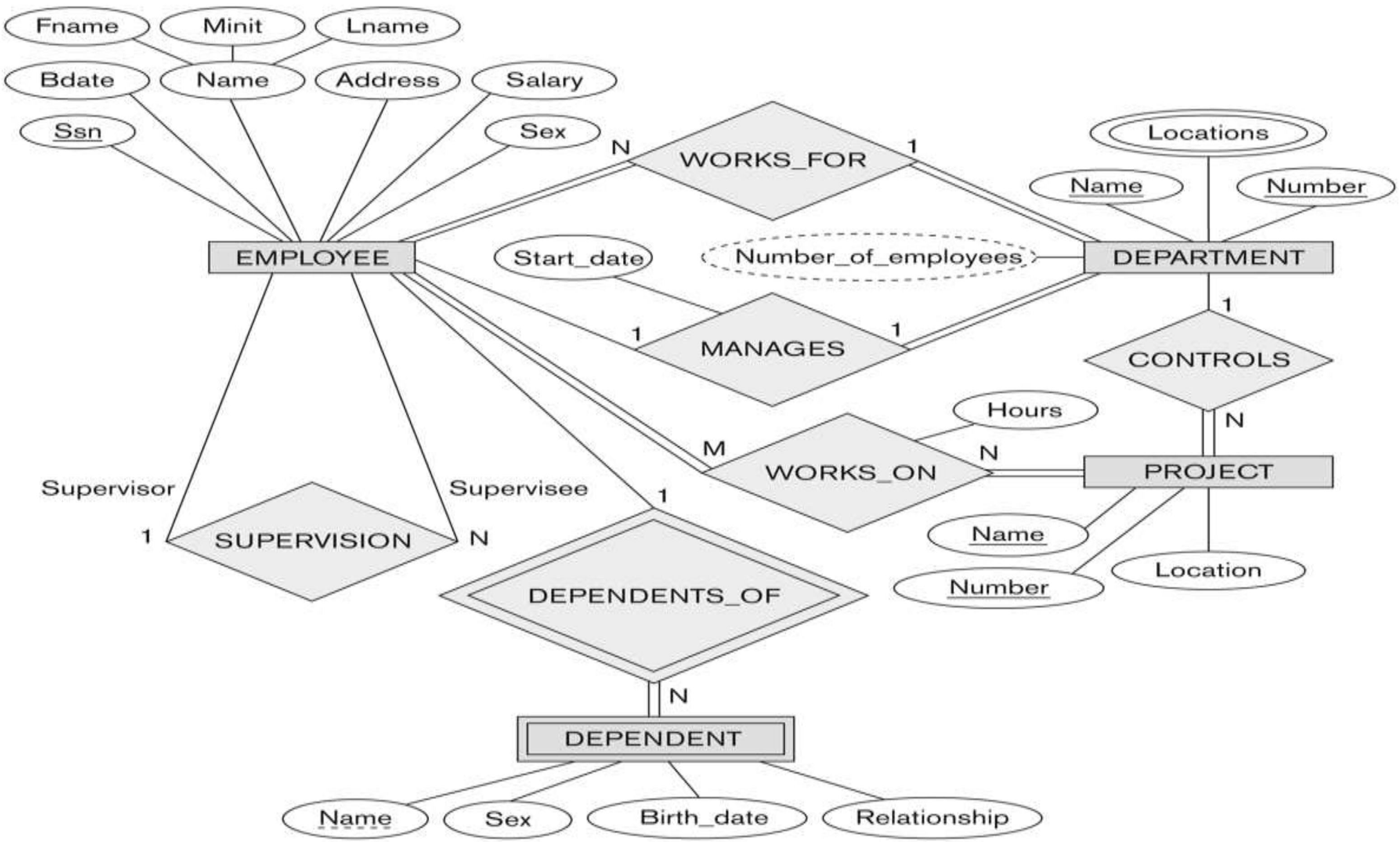


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Constraints on Relationships

- Constraints on Relationship Types
  - Can be of two types: Cardinality Ratio and Participation
  - Cardinality Ratio (specifies **maximum number of relationship instances** that an entity can participate in)
    - One-to-one (1:1)
    - One-to-many (1:N)
    - Many-to-one (N:1)
    - Many-to-many (M:N)

Many-to-one (N:1) Relationship

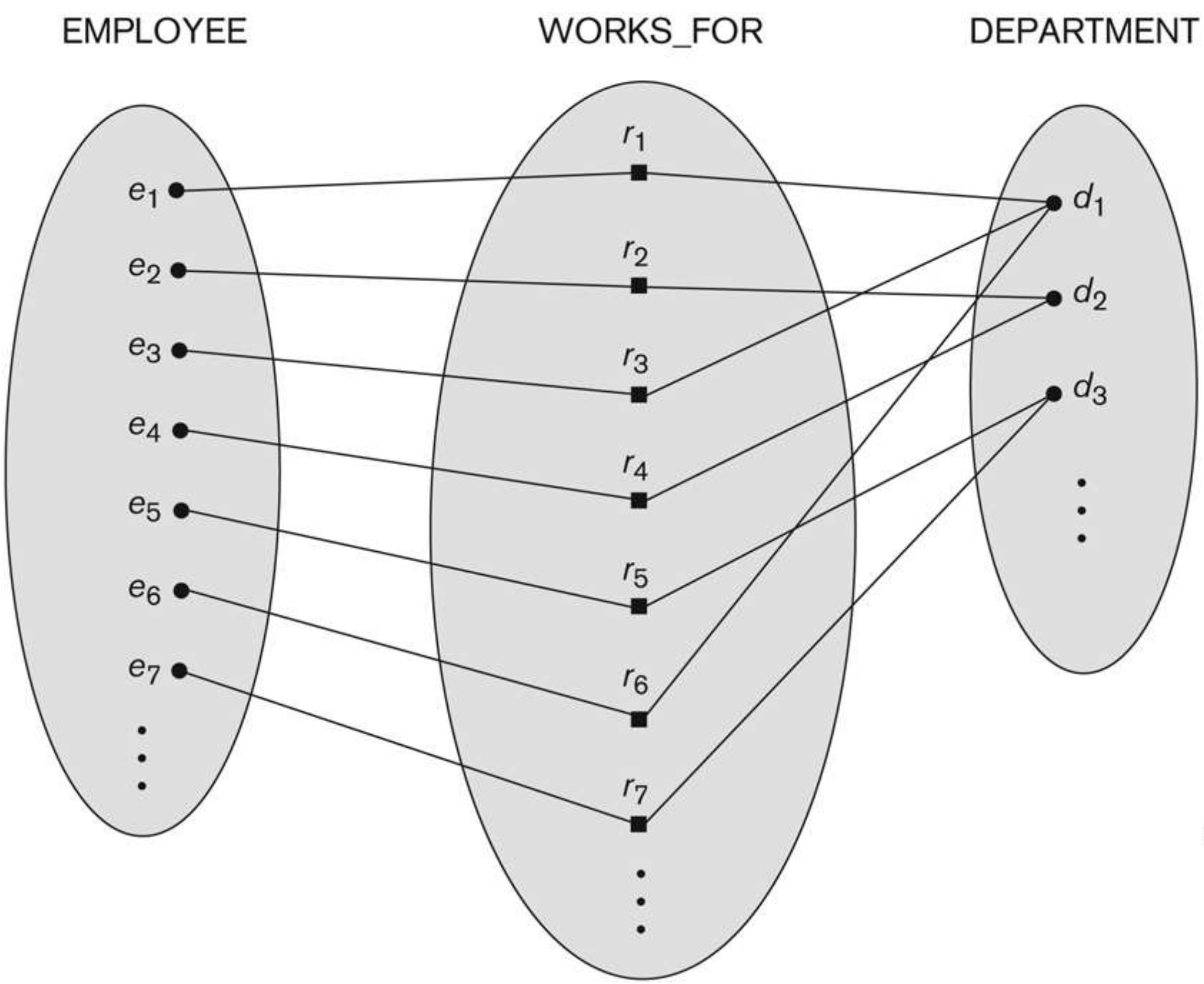


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Many-to-many (M:N) Relationship

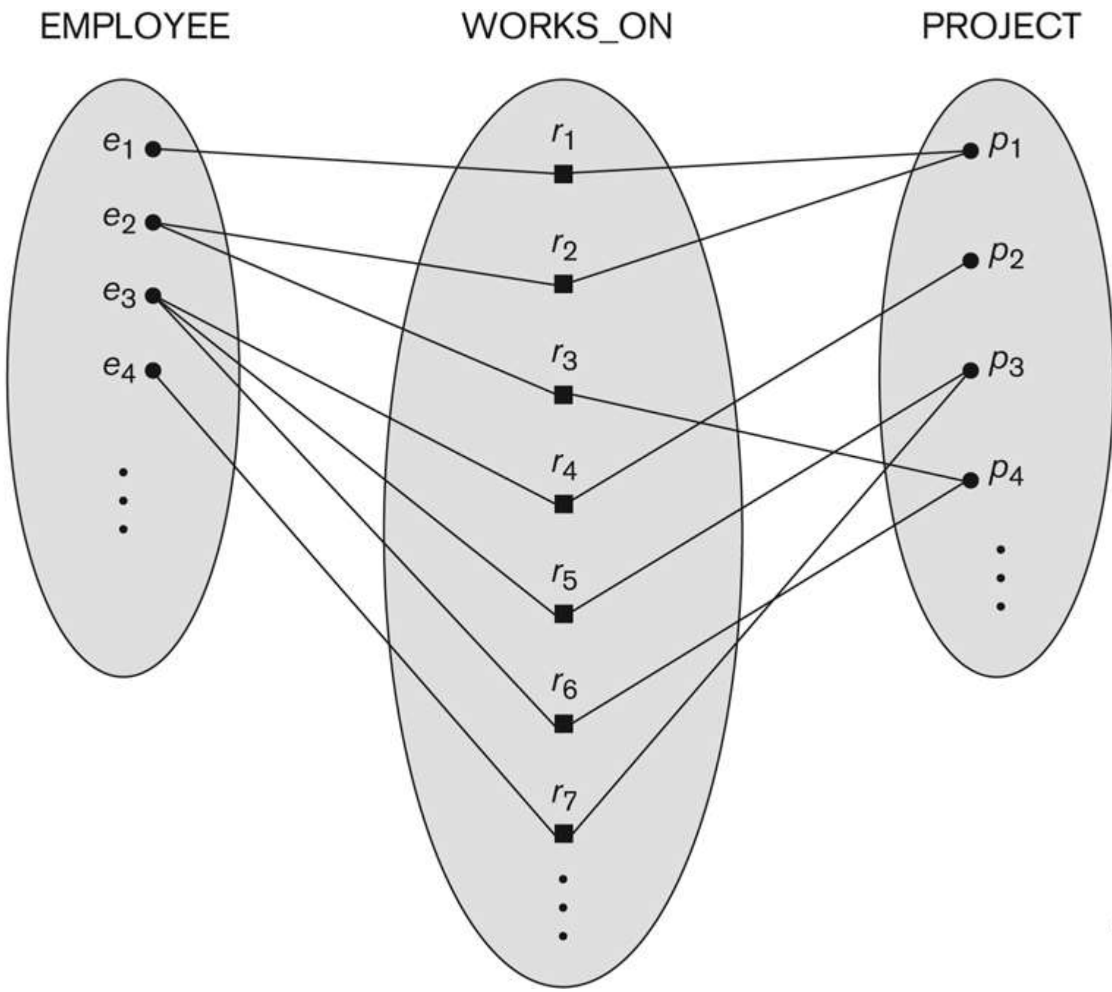


Figure 3.13
An M:N relationship, WORKS_ON.

Example: Cardinality Ratio

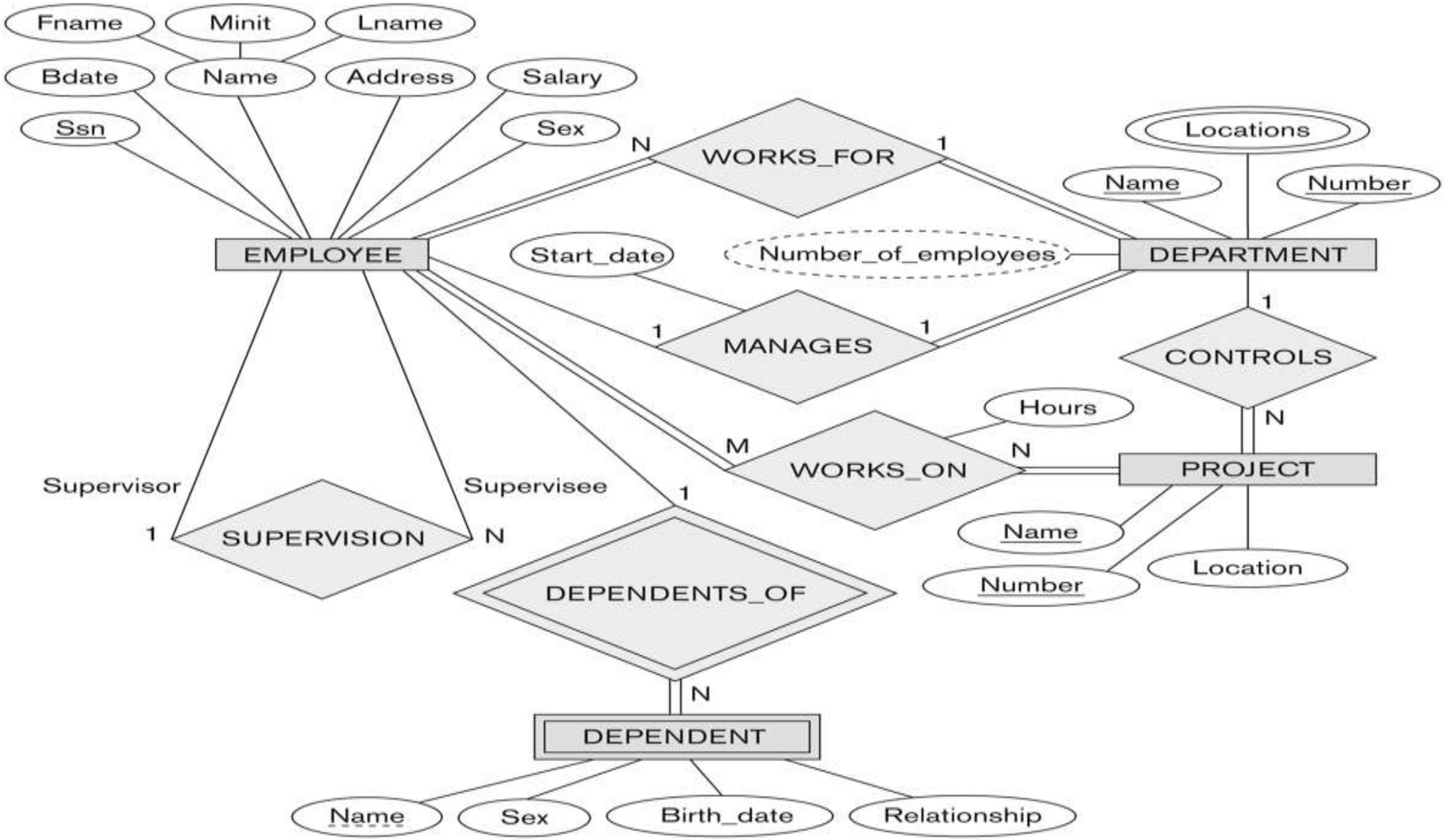


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

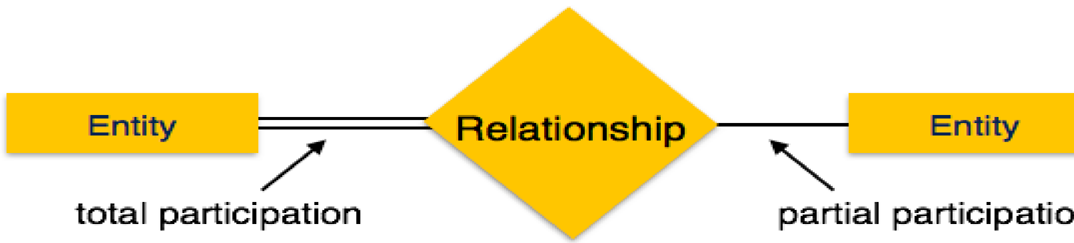
Constraints on Relationships (contd..)

- **Participation Constraints:**
- Existence of an entity depends on it being related to any other entity via a relationship type
- **Specifies minimum number of relationship instances** that each entity can participate in (minimum cardinality constraint)
- Two types of participation constraints:
  - **Total(existence dependency):** Each entity is involved in the relationship
  - Ex: ***every*** employee must work for a department
  - **Partial:** Not all entities are involved in the relationship
  - Ex: Don’t expect every employee to manage department

Notation for Constraints on Relationships

- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
  - Shown by placing appropriate numbers on the relationship edges.

- Participation constraint (on each participating entity type): total (called existence dependency) or partial.
  - Total shown by double line
  - Partial by single line.



- **NOTE: These are easy to specify for Binary Relationship Types.**

Example: Participation Constraint

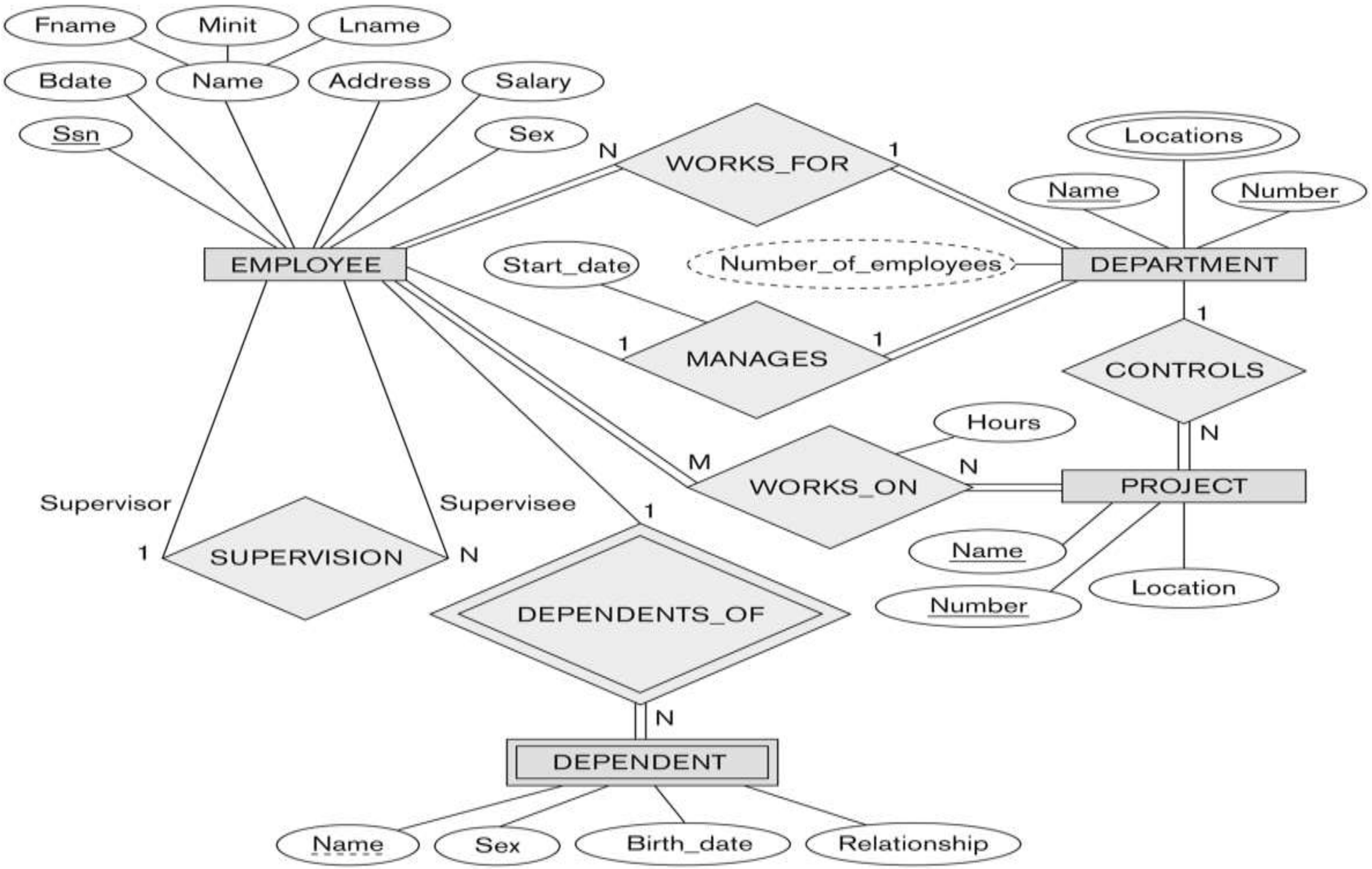
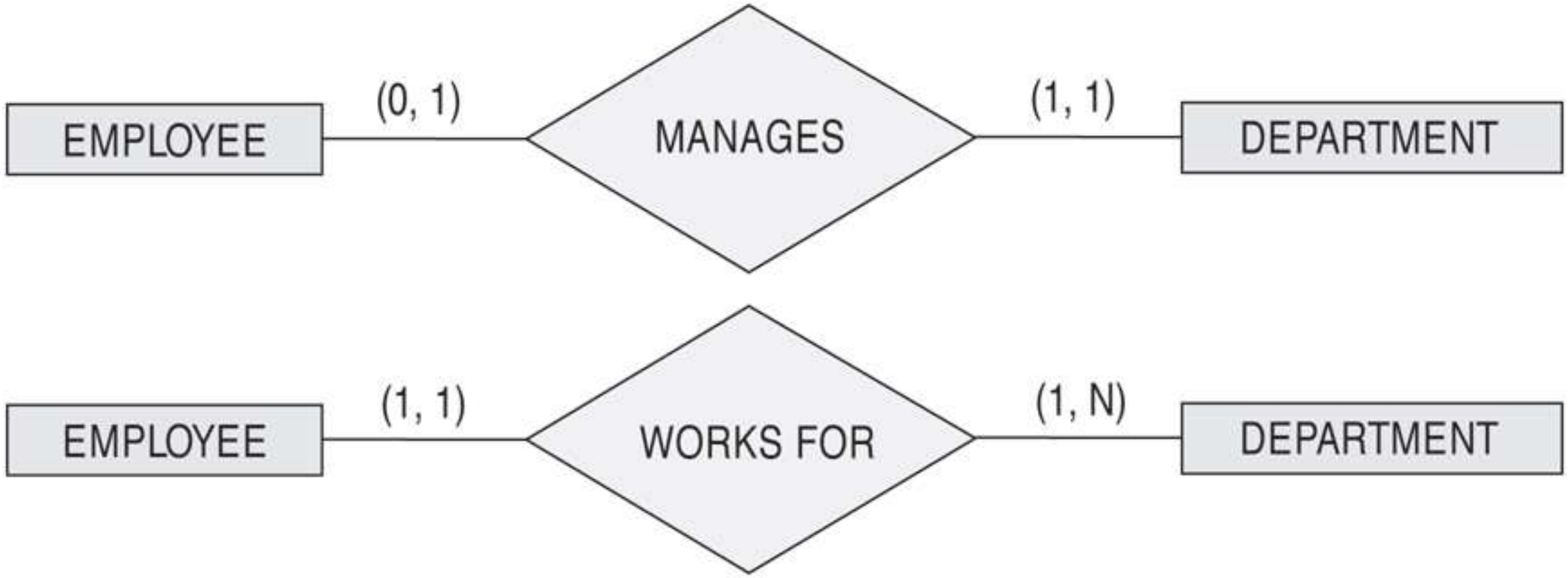


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

(min, max) notation for relationship structural constraints:

- 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
- 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



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

Example:

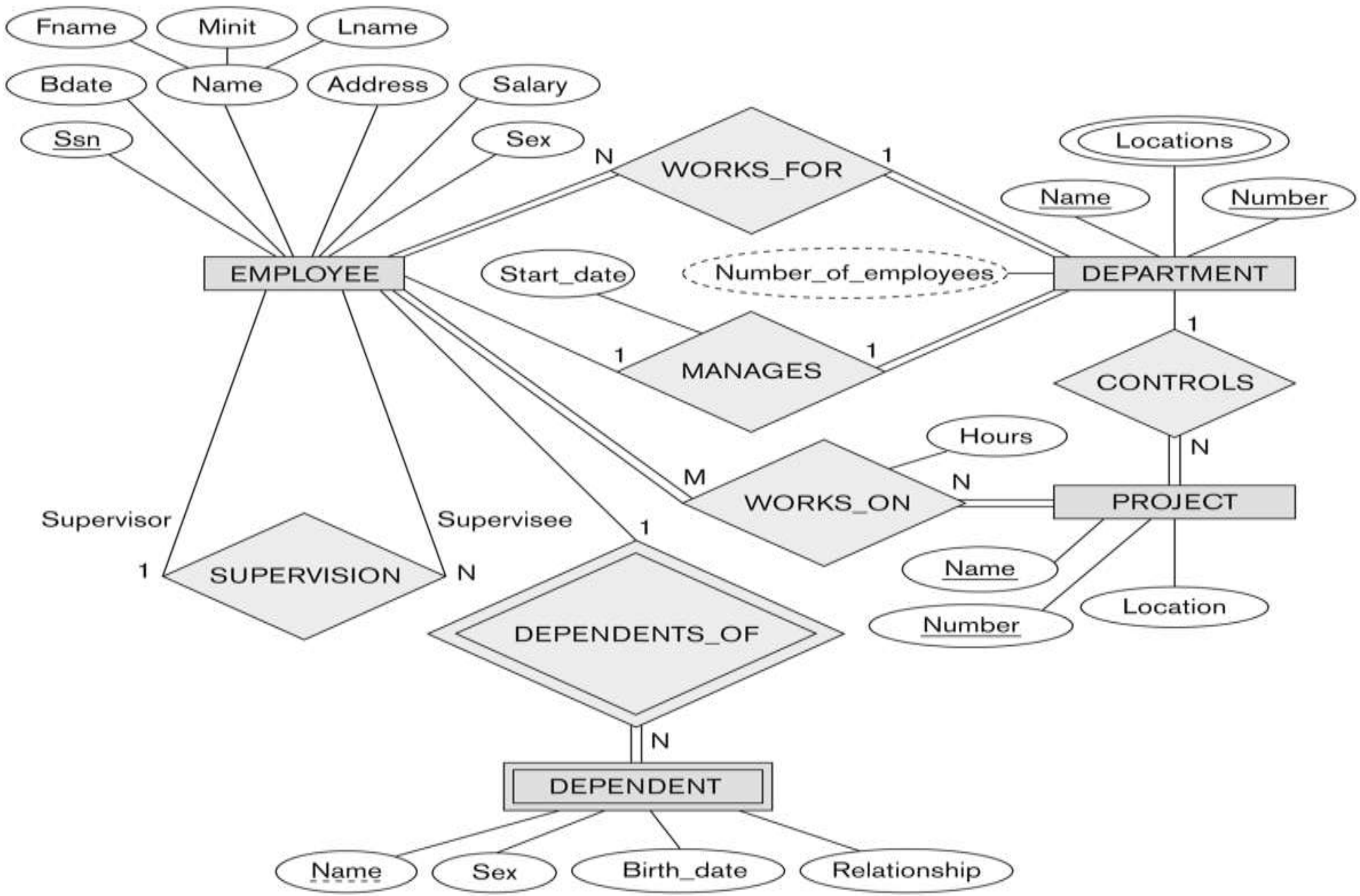


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

COMPANY ER Schema Diagram using (min, max) notation

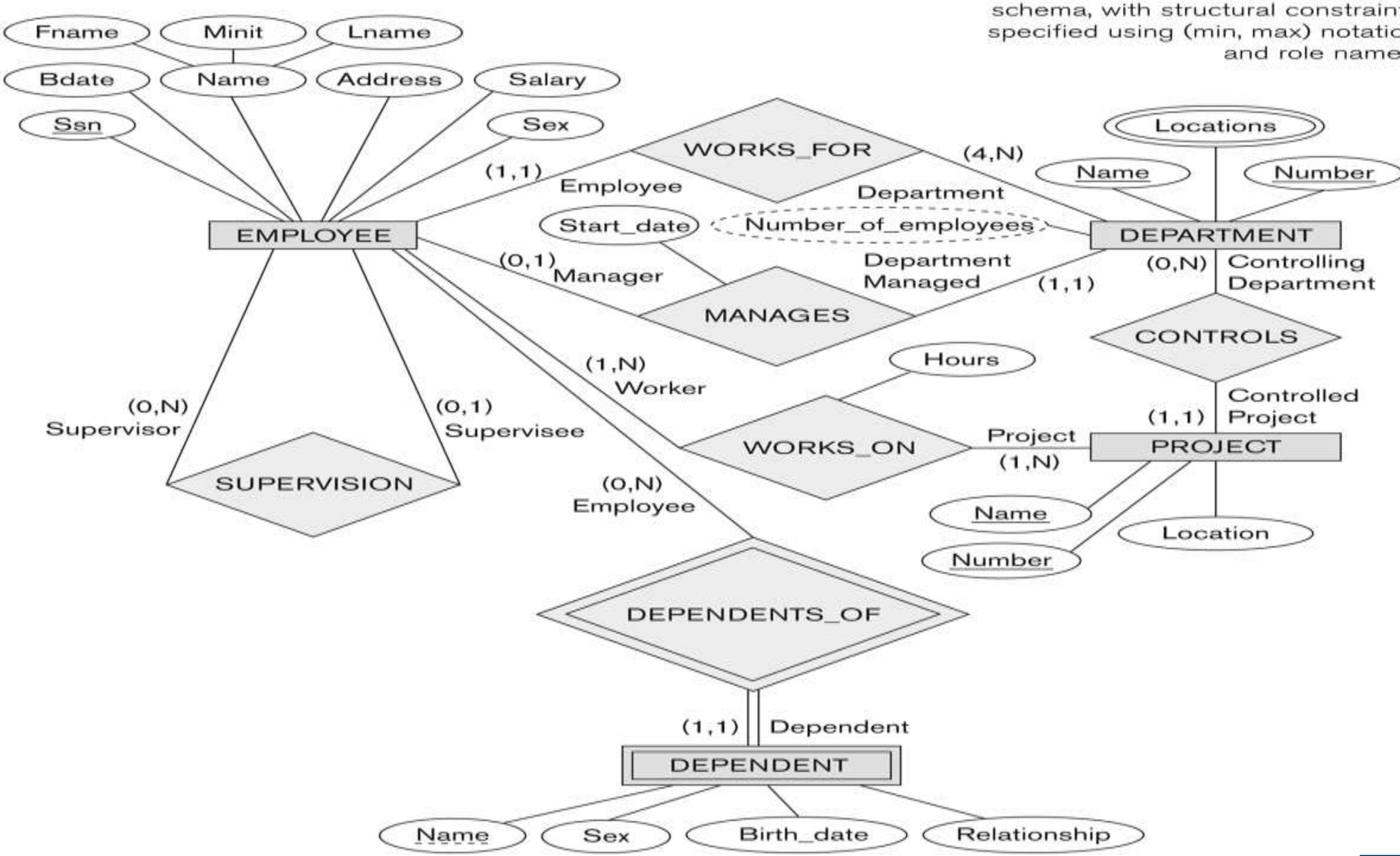


Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Summary of notation for ER diagrams

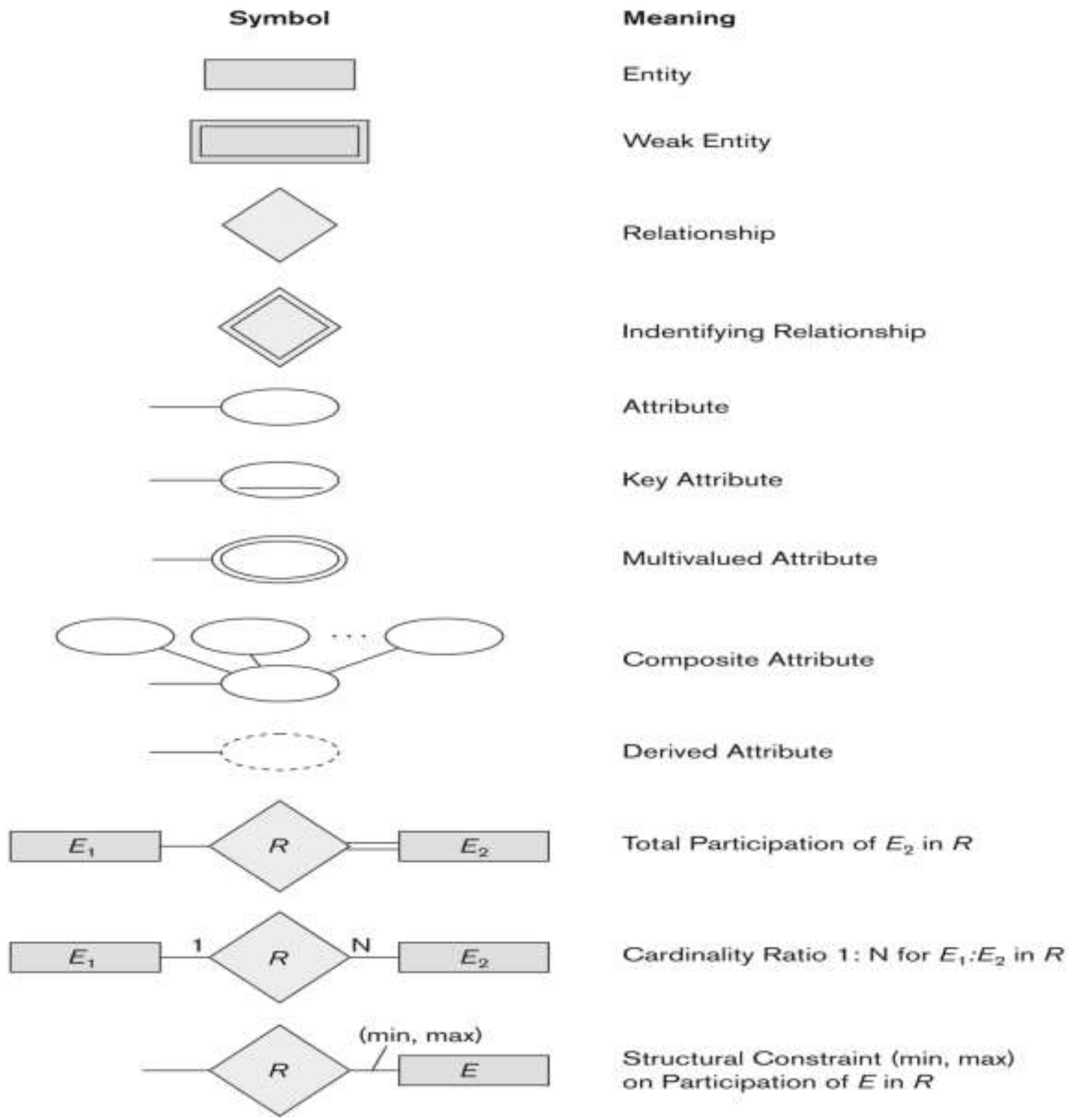


Figure 3.14
Summary of the notation for ER diagrams.

Casestudy:1

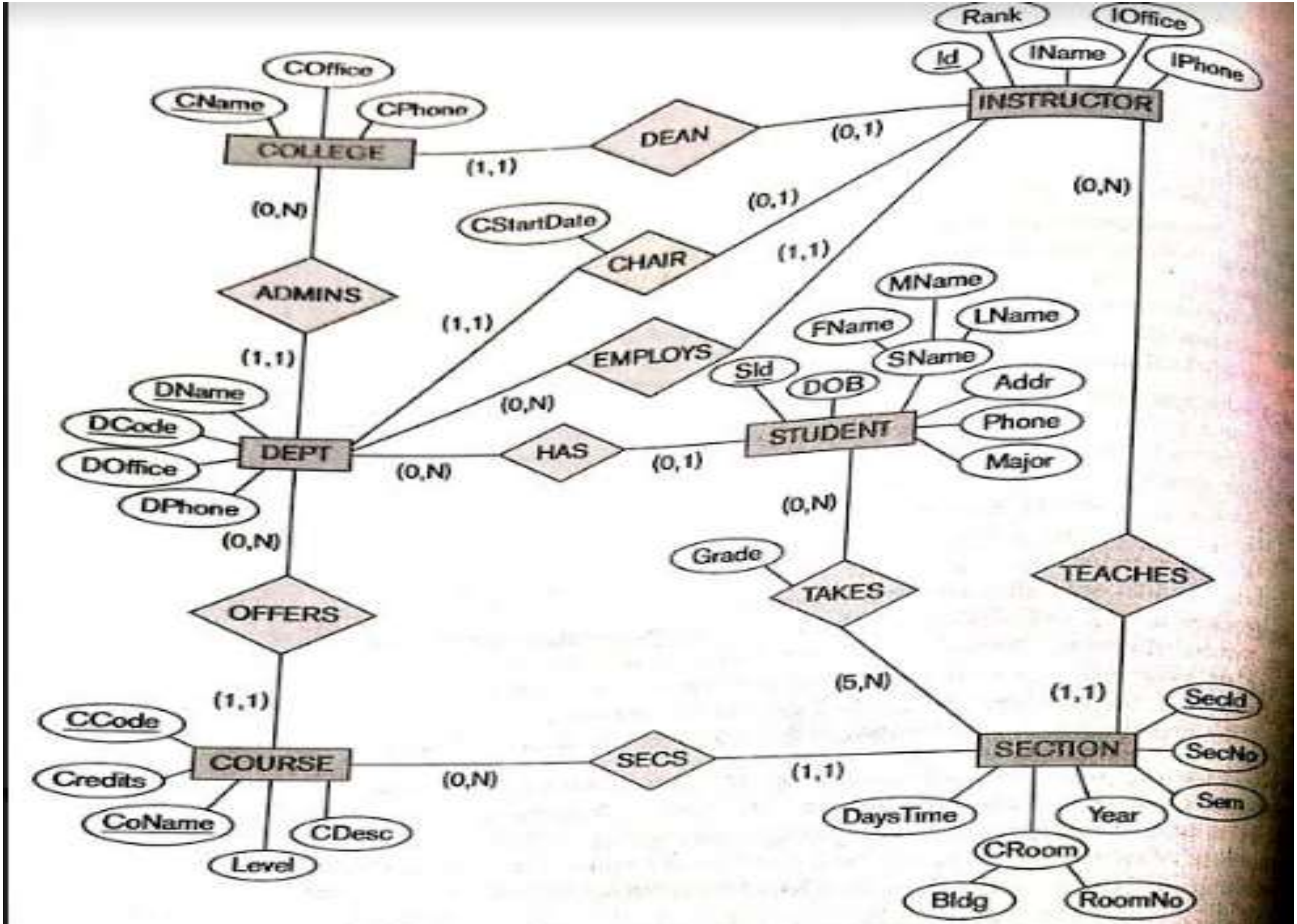
Consider following set of requirements for a UNIVERSITY database that is used to keep track of students transcripts.

- University keeps track of each students name, student number, SSN, current address and phone, permanent address and phone, birth date, gender, class, major department, minor department if any, and degree program. Some user applications need to refer to city, state and Zip code of students permanent address and to students last name. Both SSN and student number have unique values for each student
- Each department is described by a name, department code, office number, office phone and college. Both name and code have unique values for each department

Casestudy:1(contd..)

- Each course has a course name, description, course number, number of semester hours, level and offering department. The value of course number is unique for each course
- Each section has an instructor, semester, year, course and section number. The section number distinguishes sections of the same course that are taught during same semester/year, its value are 1,2,3… upto number of sections taught during each semester
- A grade report has a student, section, letter grade and numeric grade(0,1,2,3 or 4)
- **Design ER Schema for this application. Specify key attributes of each entity type, constraints on each relationship type. Note any unspecified requirements and make appropriate assumptions to make specification complete**

Solution:



Rank
IOffice
Id
IName
IPhone
INSTRUCTOR
COffice
CName
CPhone
COLLEGE
DEAN
(1,1)
(0,1)
(0,N)
(0,1)
CStartDate
CHAIR
(1,1)
ADMINS
(1,1)
MName
FName
LName
SName
EMPLOYS
(1,1)
SId
DOB
Addr
DName
DCode
DOffice
DPhone
DEPT
(0,N)
HAS
(0,N)
(0,1)
STUDENT
Phone
Major
(0,N)
(0,N)
Grade
TAKES
TEACHES
OFFERS
(1,1)
(5,N)
(1,1)
CCode
Credits
CoName
Level
CDesc
COURSE
SECS
(0,N)
(1,1)
SECTION
SecId
SecNo
Sem
Year
DaysTime
CRoom
Bldg
RoomNo

Relationships of Higher Degree

- Degree of relationship type is defined as number of participating entity types
- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are called n-ary
- In general, an n-ary relationship is not equivalent to n binary relationships
- Constraints are harder to specify for higher-degree relationships (n > 2) than for binary relationships

Discussion of n-ary relationships (n > 2)

- In general, 3 binary relationships can represent different information than a single ternary relationship (see Figure 3.17a and b on next slide)

- If needed, the binary and n-ary relationships can all be included in the schema design (see Figure 3.17a and b, where all relationships convey different meanings)

- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types) (see Figure 3.17c)

Example of a ternary relationship

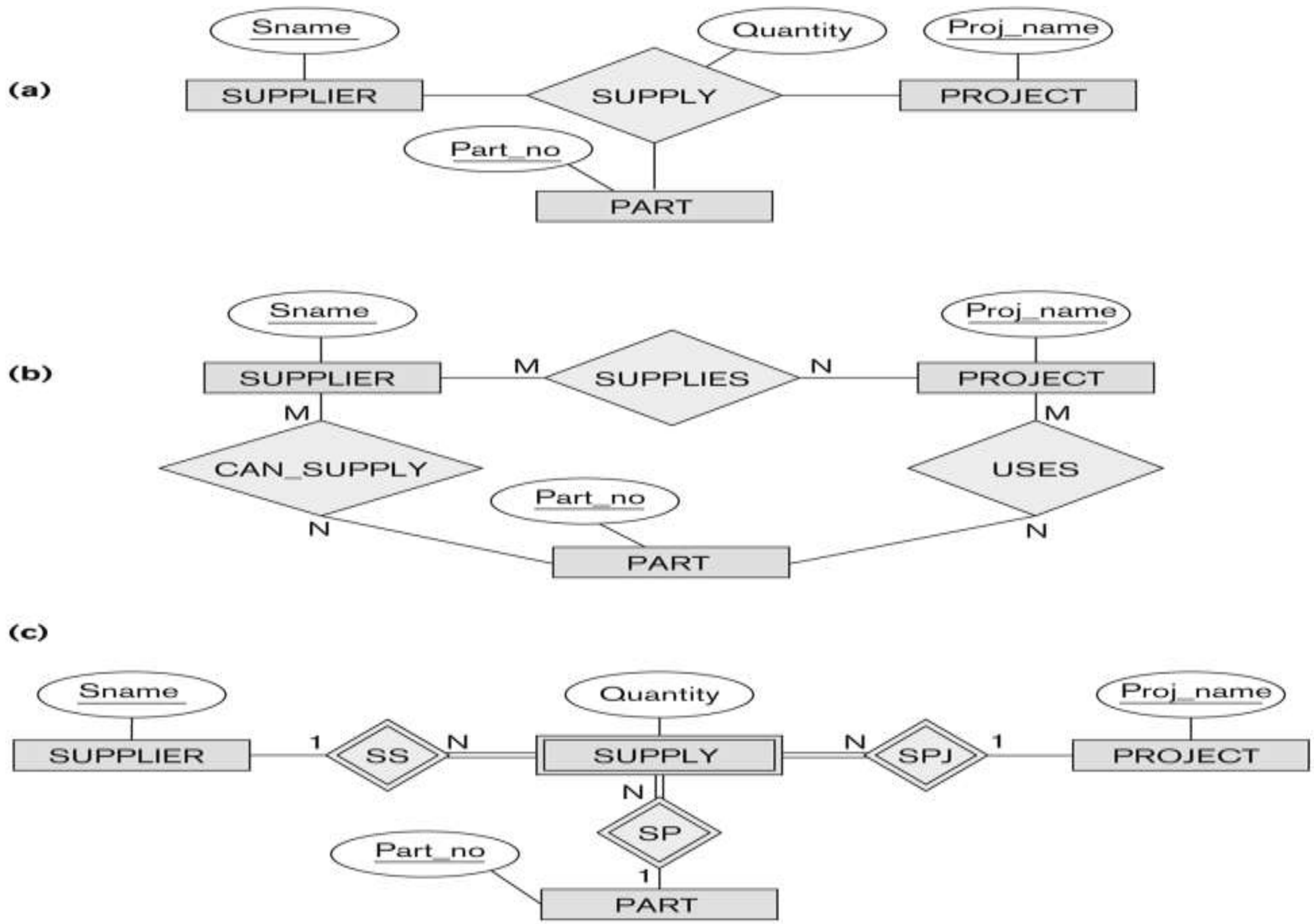


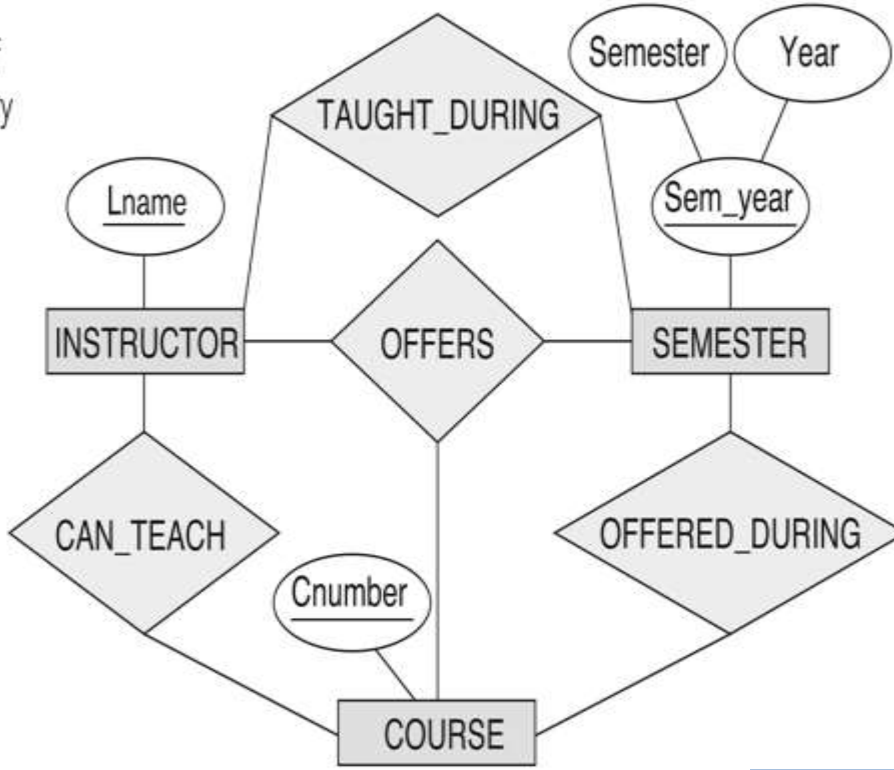
Figure 3.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Discussion of n-ary relationships (n > 2)

- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is **redundant**

- For example, the **TAUGHT_DURING, OFFERED_DURING** binary relationships can be derived from the ternary relationship **OFFERS** (based on the meaning of the relationships)

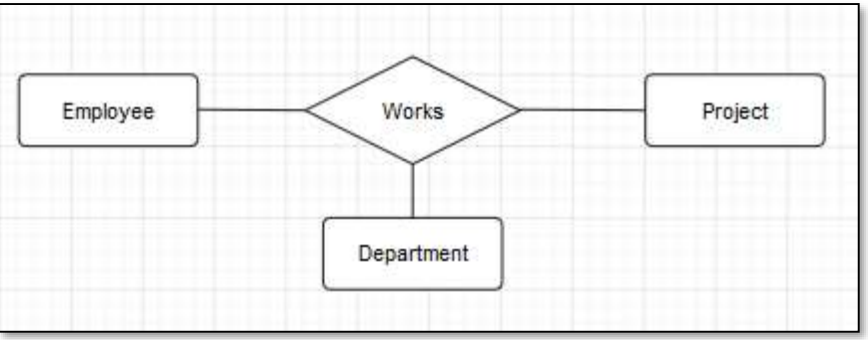
Figure 3.18
Another example of ternary versus binary relationship types.



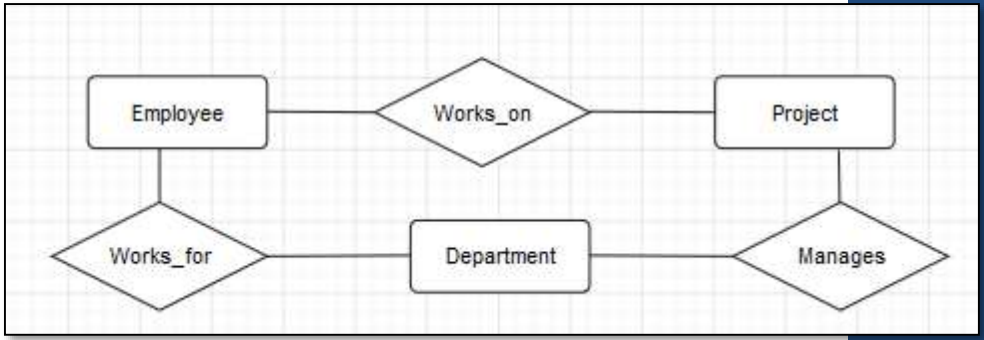
Example

- Two ER Diagrams of employees, department and projects are given below:

Model 1



Model 2



1. Which employees work on Project J1?
2. Which departments are involved in project J1?
3. Which parts are supplied to project J1?
4. How many employees of department D1 working on project J1?
5. Are there parts that S1 can supply but is not supplying currently?

- **Queries answered by model 1**
- **Queries answered by model 2**
- **Queries cannot be answered by any models?**

Unit 2.2:

Outline

- **EER** stands for **E**nhanced **ER** or **E**xtended **ER**

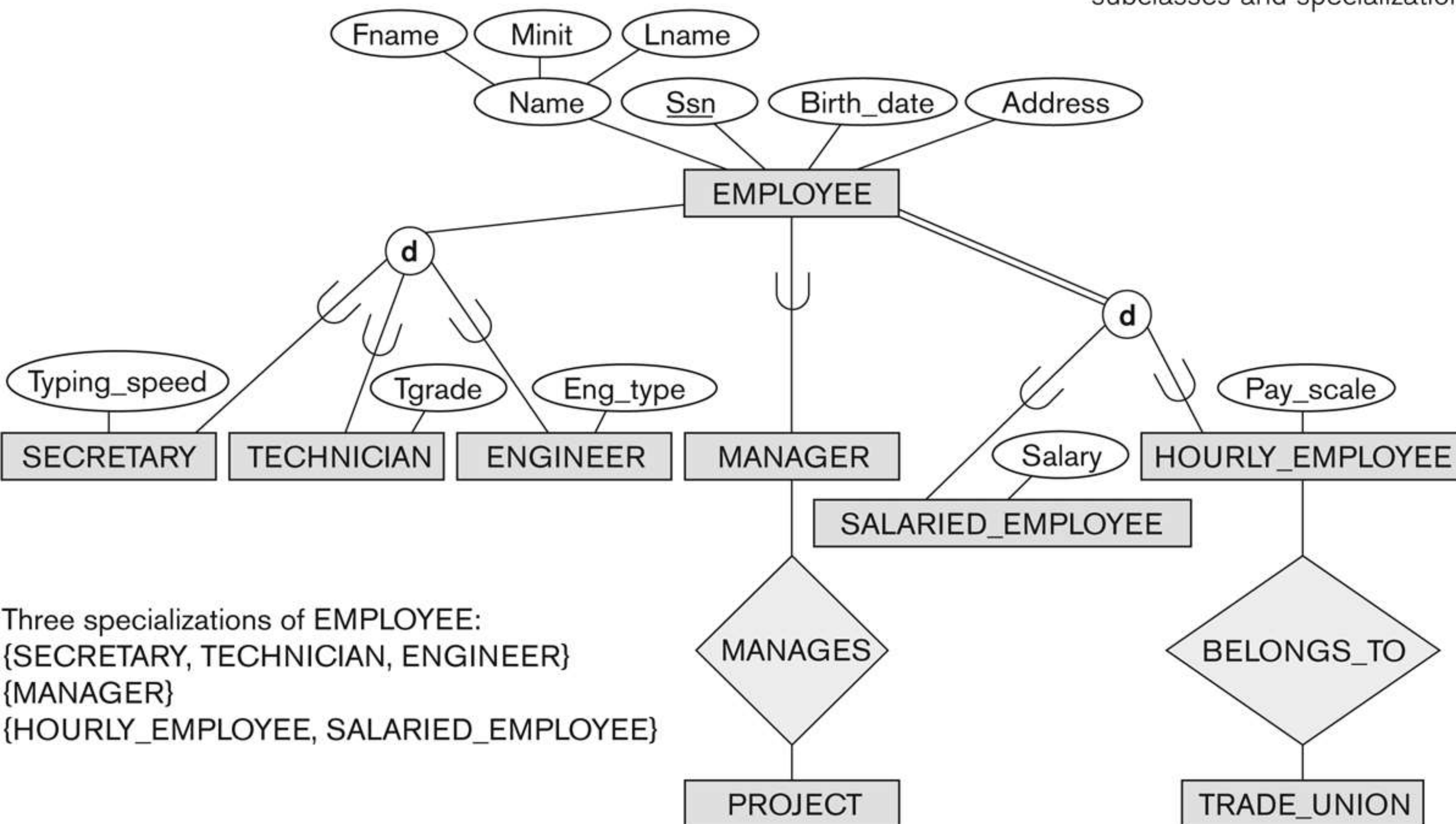
- **EER Model Concepts**
  - Includes all modeling concepts of basic ER
  - Additional concepts:
    - **subclasses/superclasses**
    - **specialization/generalization**
    - **aggregation**
    - **categories (UNION types)**
  - These are fundamental to conceptual modeling

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*

Subclasses and Superclasses(contd..)

Figure 4.1
EER diagram notation to represent subclasses and specialization.



Subclasses and Superclasses (contd..)

- 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 (contd..)

- 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 (contd..)

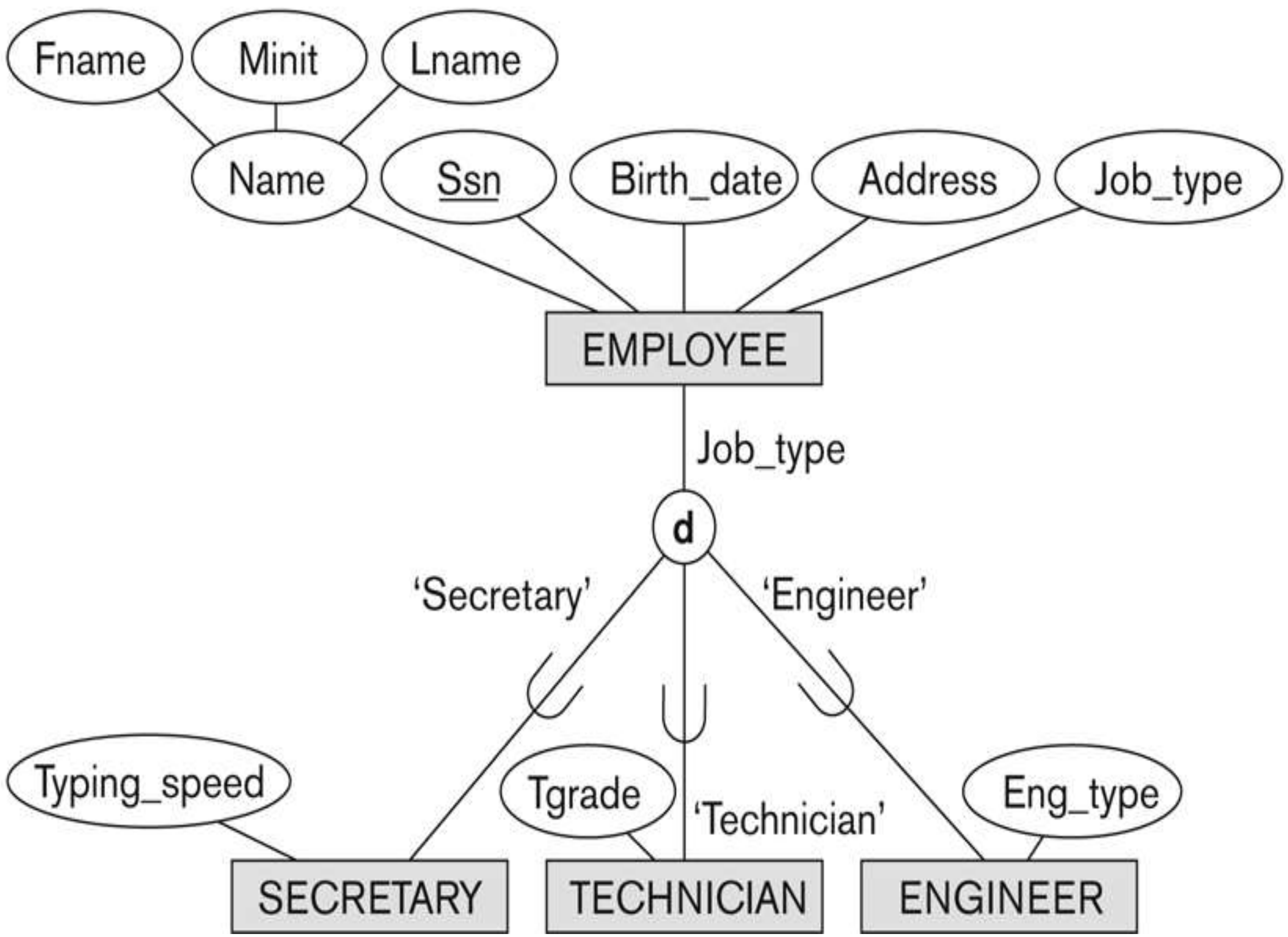
- Examples:
  - A salaried employee who is also an engineer belongs to the two subclasses:
    - ENGINEER, and
    - SALARIED_EMPLOYEE
  - A salaried employee who is also an engineering manager belongs to the three subclasses:
    - MANAGER,
    - ENGINEER, and
    - SALARIED_EMPLOYEE
- It is not necessary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

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

Attribute Inheritance in Superclass / Subclass Relationships

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



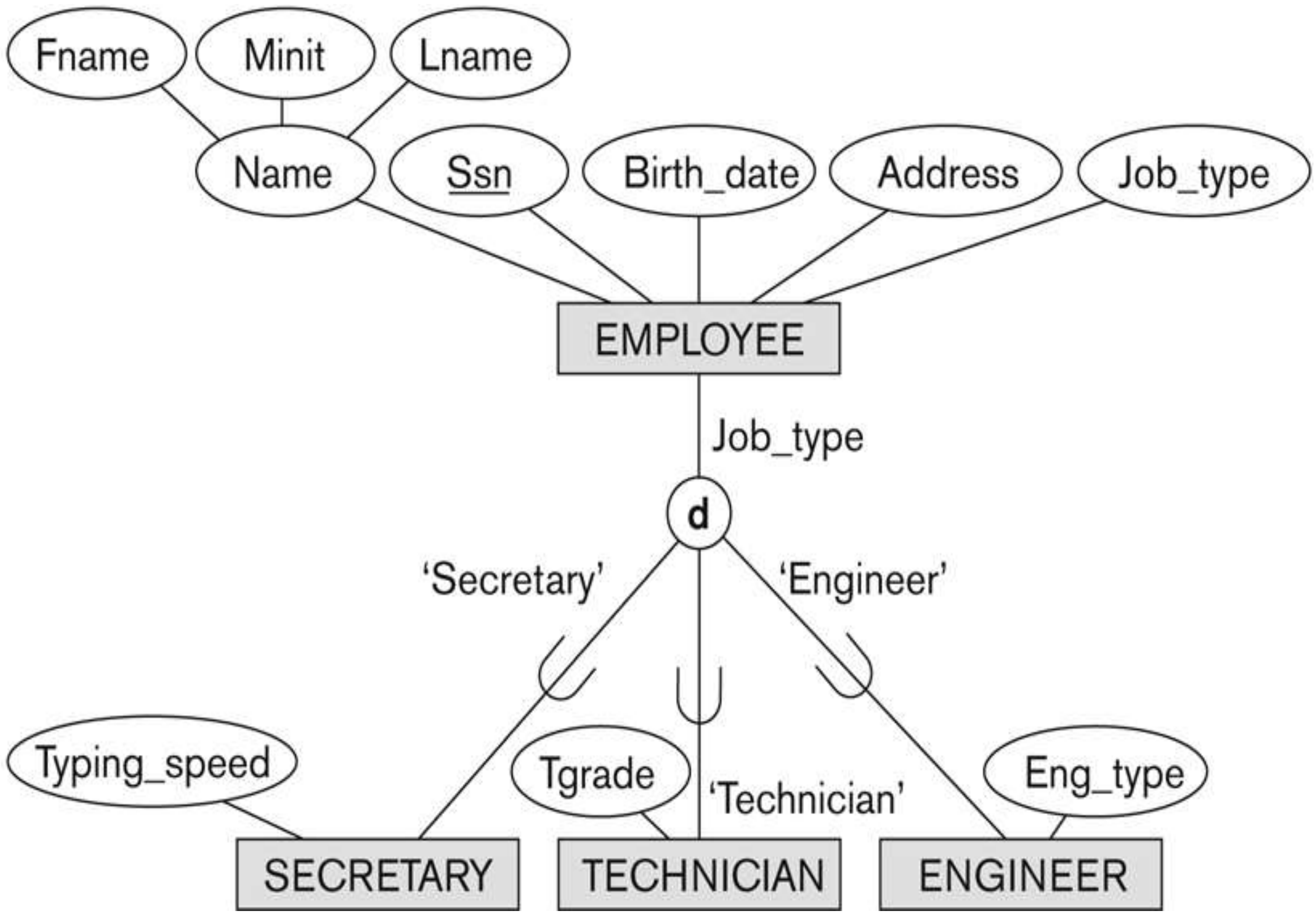
Specialization

- 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

Representing Specialization in EER Diagrams

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



Specialization (contd..)

- 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 (contd..)

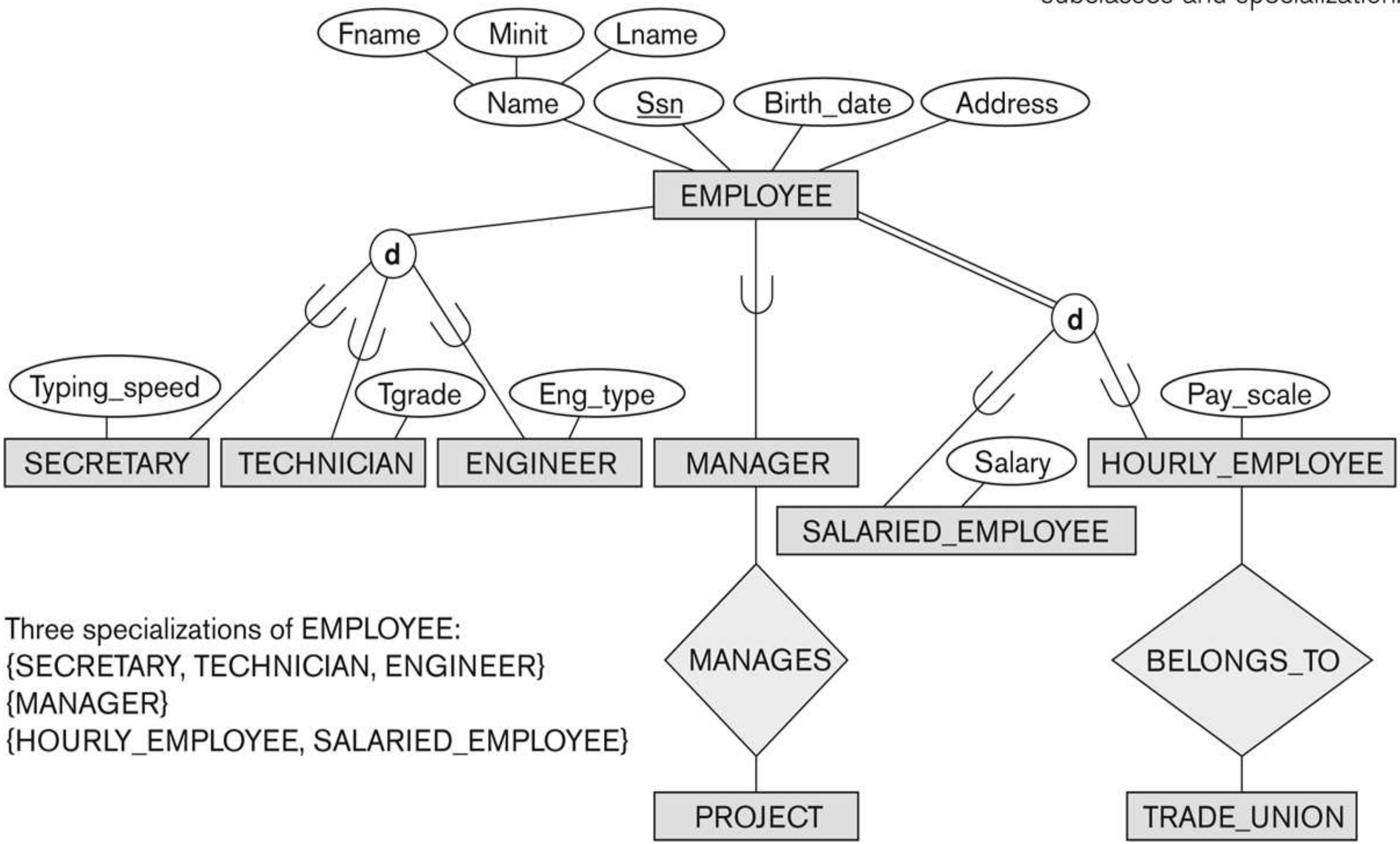


Figure 4.1
EER diagram notation to represent subclasses and specialization.

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 (contd..)

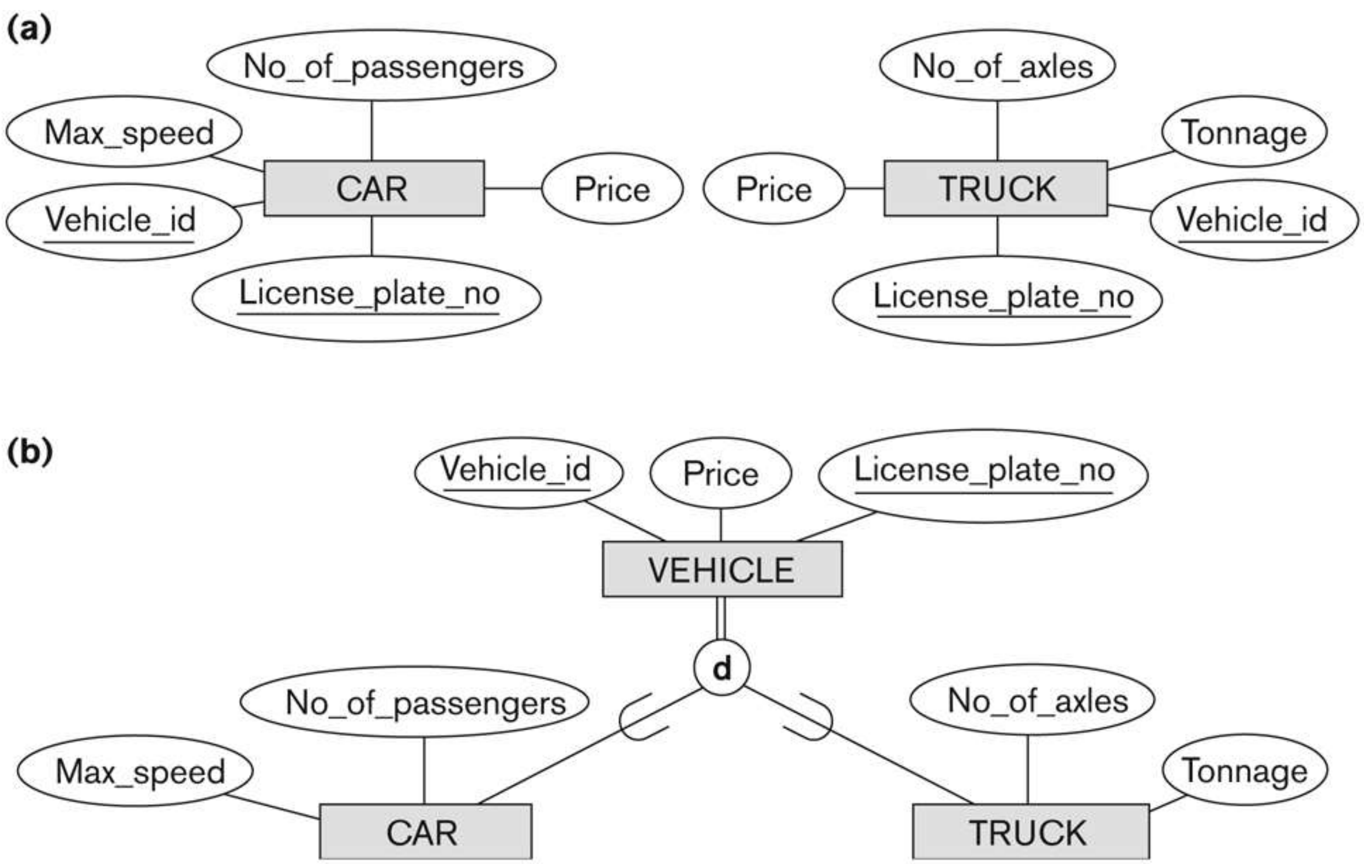


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Generalization and Specialization

- 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

Generalization and Specialization (contd..)

- 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

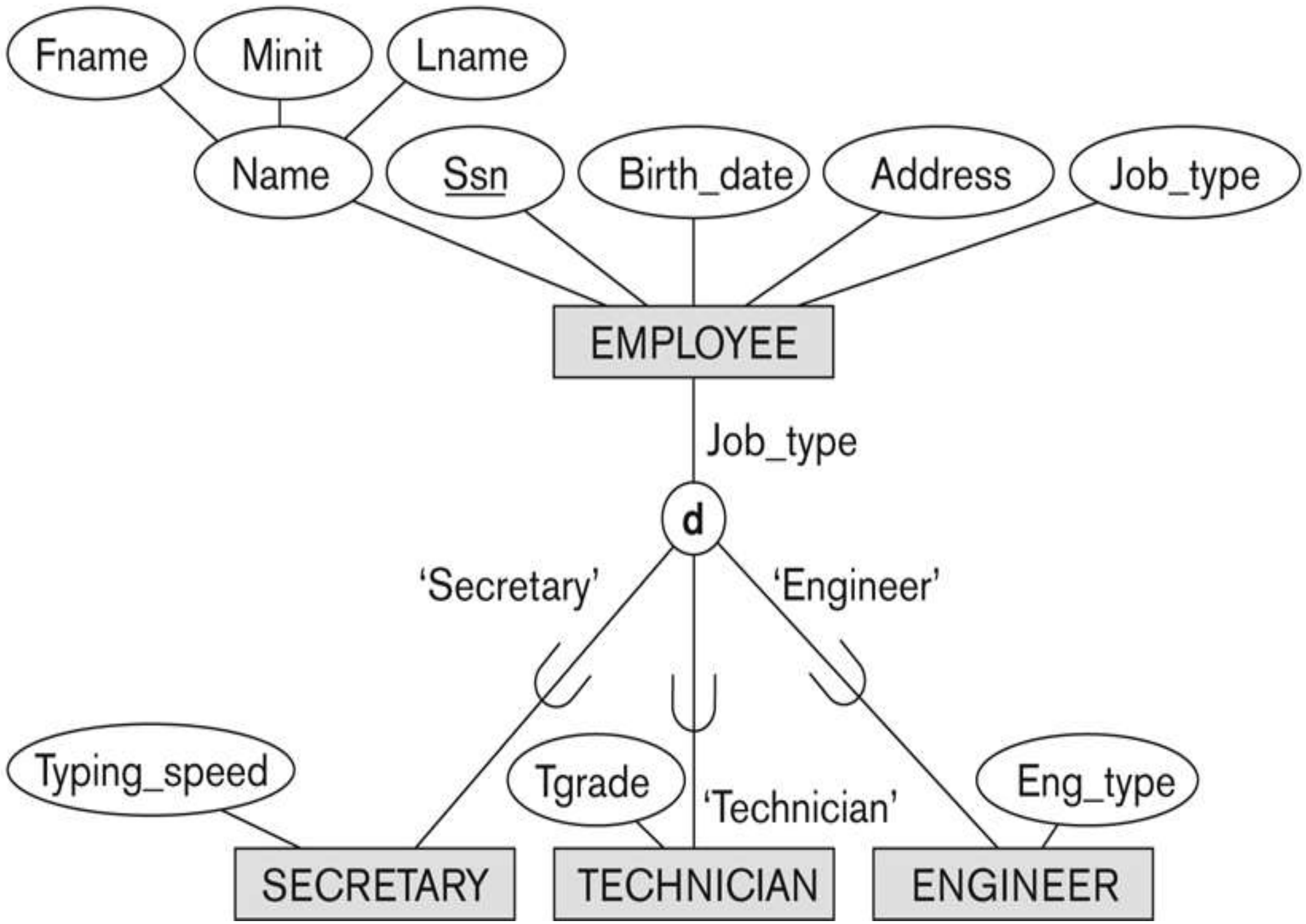
- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called **predicate-defined (or condition-defined**) subclasses
  - Condition is a constraint that determines subclass members
  - Display a predicate-defined subclass by writing the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

Constraints on Specialization and Generalization (contd..)

- If all subclasses in a specialization have membership condition **on same attribute of the superclass**, specialization is called an **attribute-defined 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 (contd..)

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

Constraints on Specialization and Generalization (contd..)

- **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 (contd..)

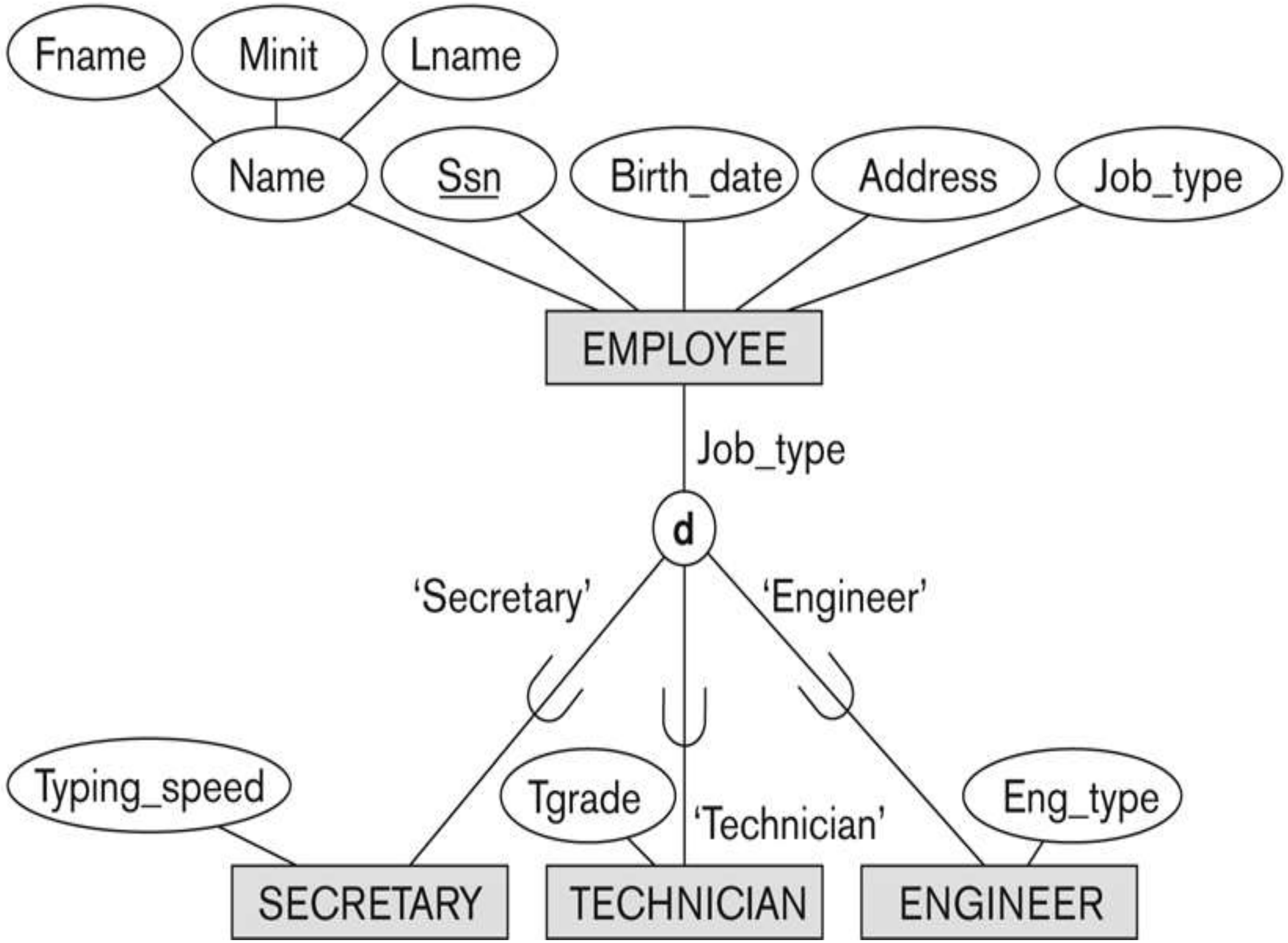
- **Completeness Constraint:**
  - *Total* specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
  - Shown in EER diagrams by a ***double line***
  - *Partial* allows an entity not to belong to any of the subclasses
  - Shown in EER diagrams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

Constraints on Specialization and Generalization (contd..)

- 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.



Example of overlapping total Specialization

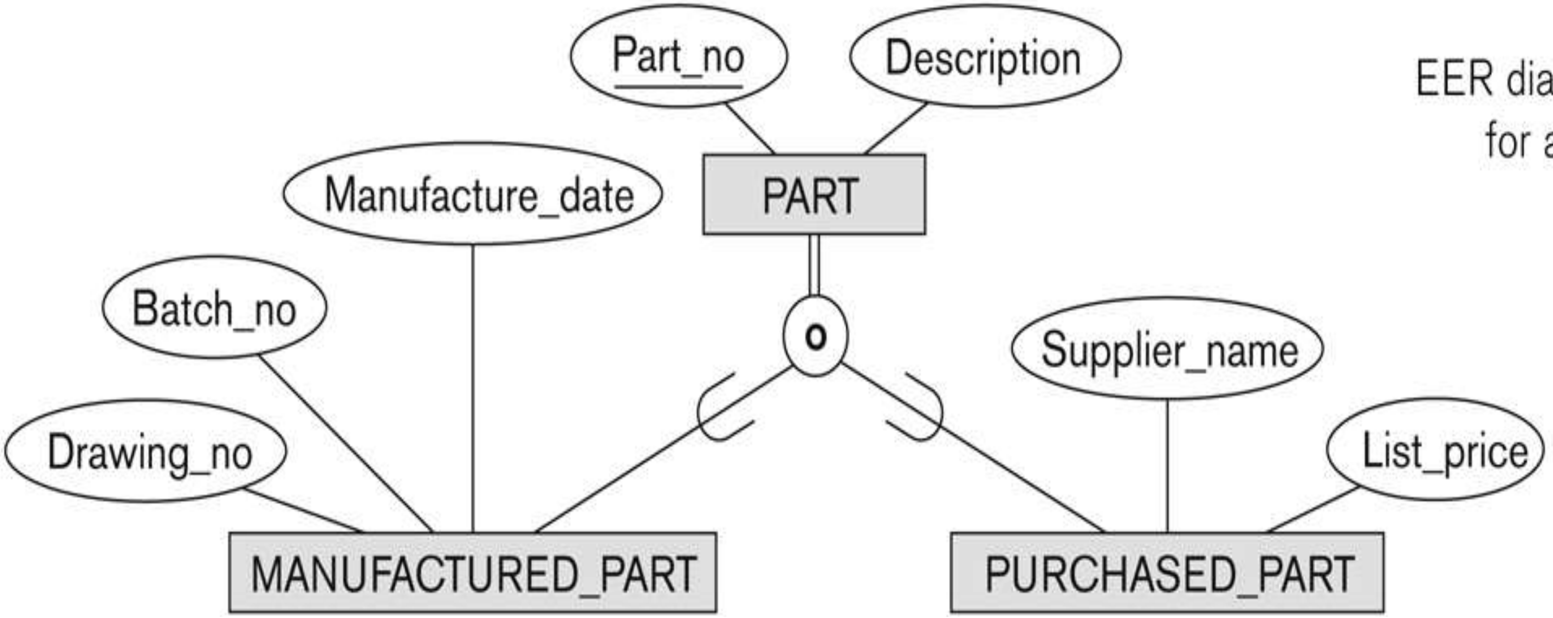


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Example:

- Explain by drawing a diagram of how could you model a driving license which may be for driving a car, bus, truck, taxi, scooter or tourist bus.
- Use a superclass/ subclass model.
- List all information that each entity must possess

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***)

Shared Subclass “Engineering_Manager”

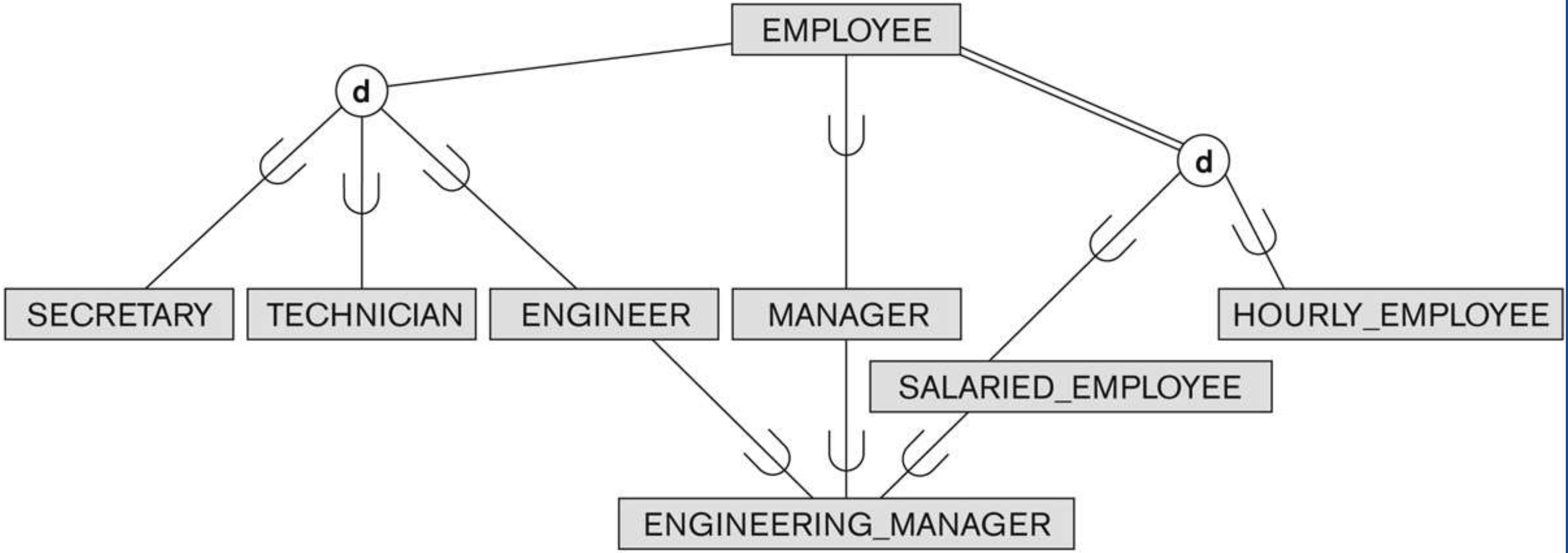


Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.

Specialization/Generalization Hierarchies, Lattices & Shared Subclasses (contd..)

- 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 (contd..)

- 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)

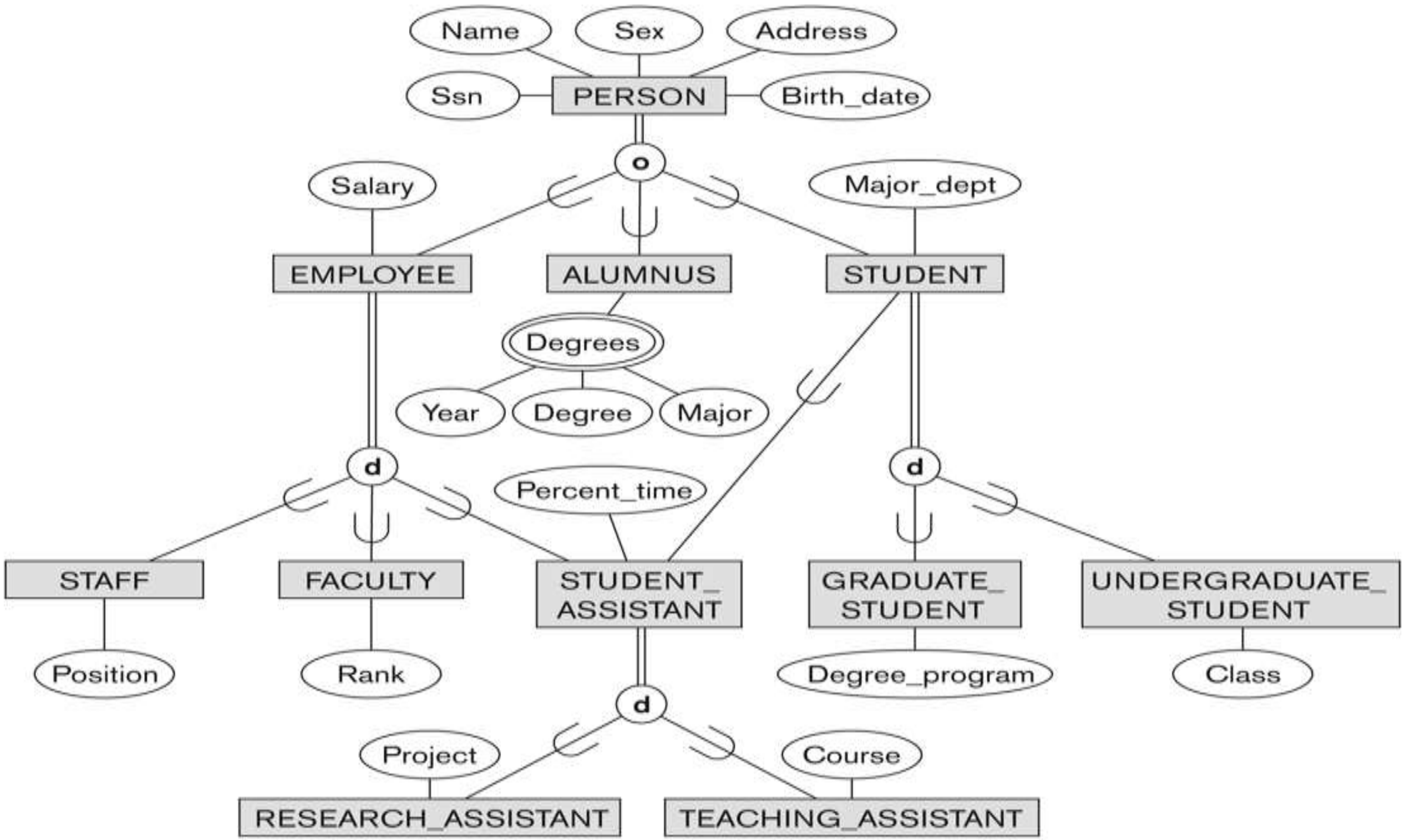


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Categories (UNION TYPES)

- 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) (contd..)

- 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

- 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 (contd..)

- 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 (contd..)

- 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

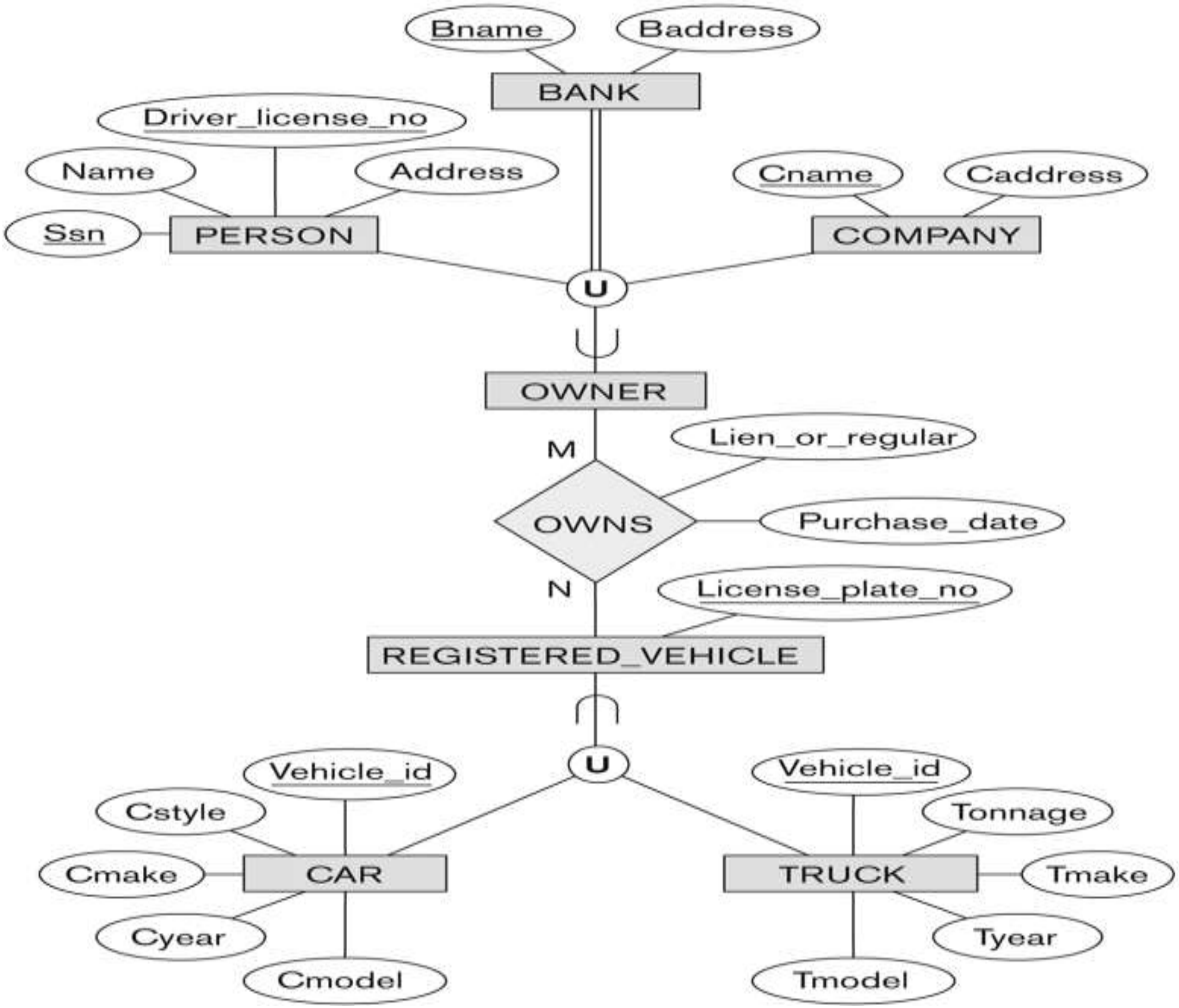


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Formal Definitions of EER Model (contd..)

- 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***

UML Example for Displaying Specialization / Generalization

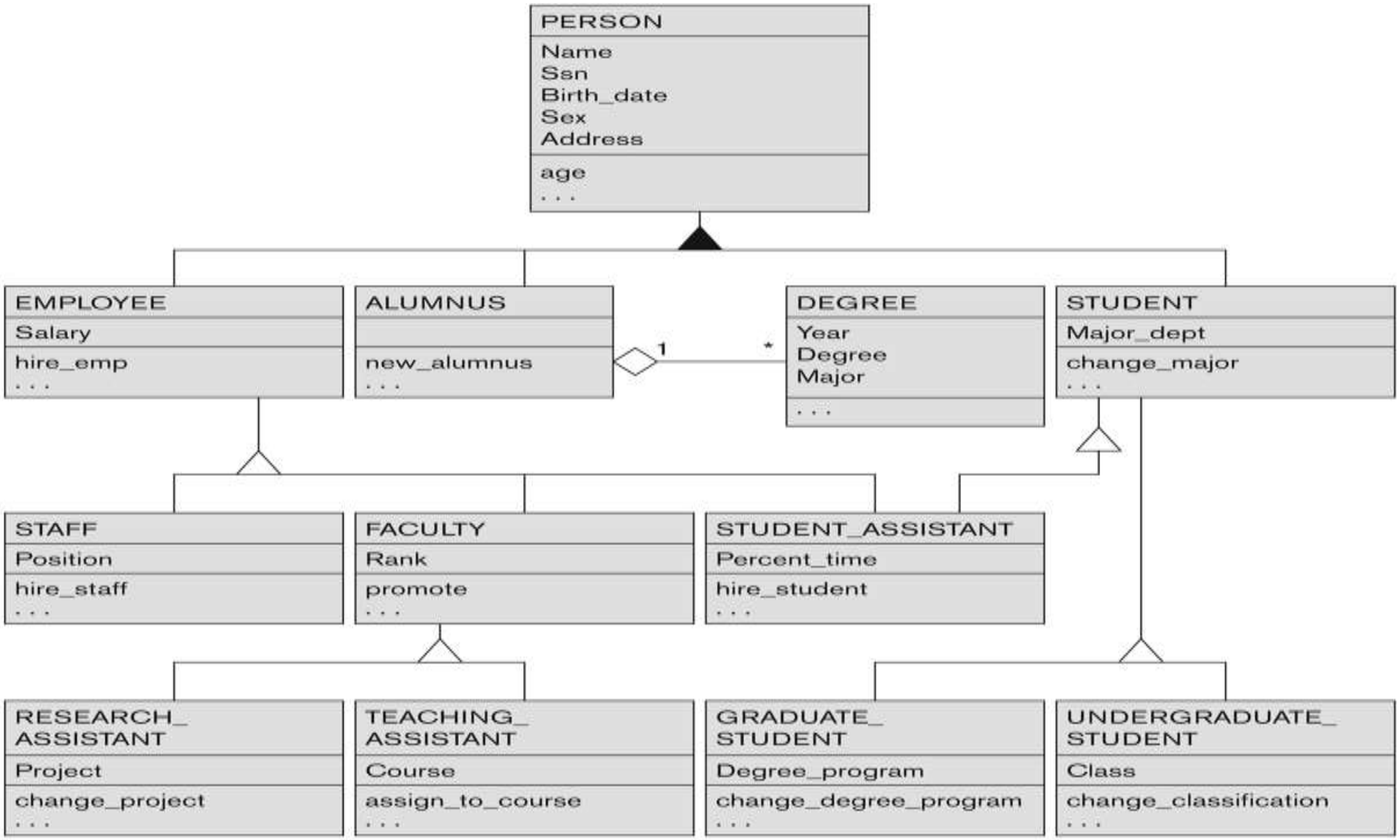


Figure 4.10
A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

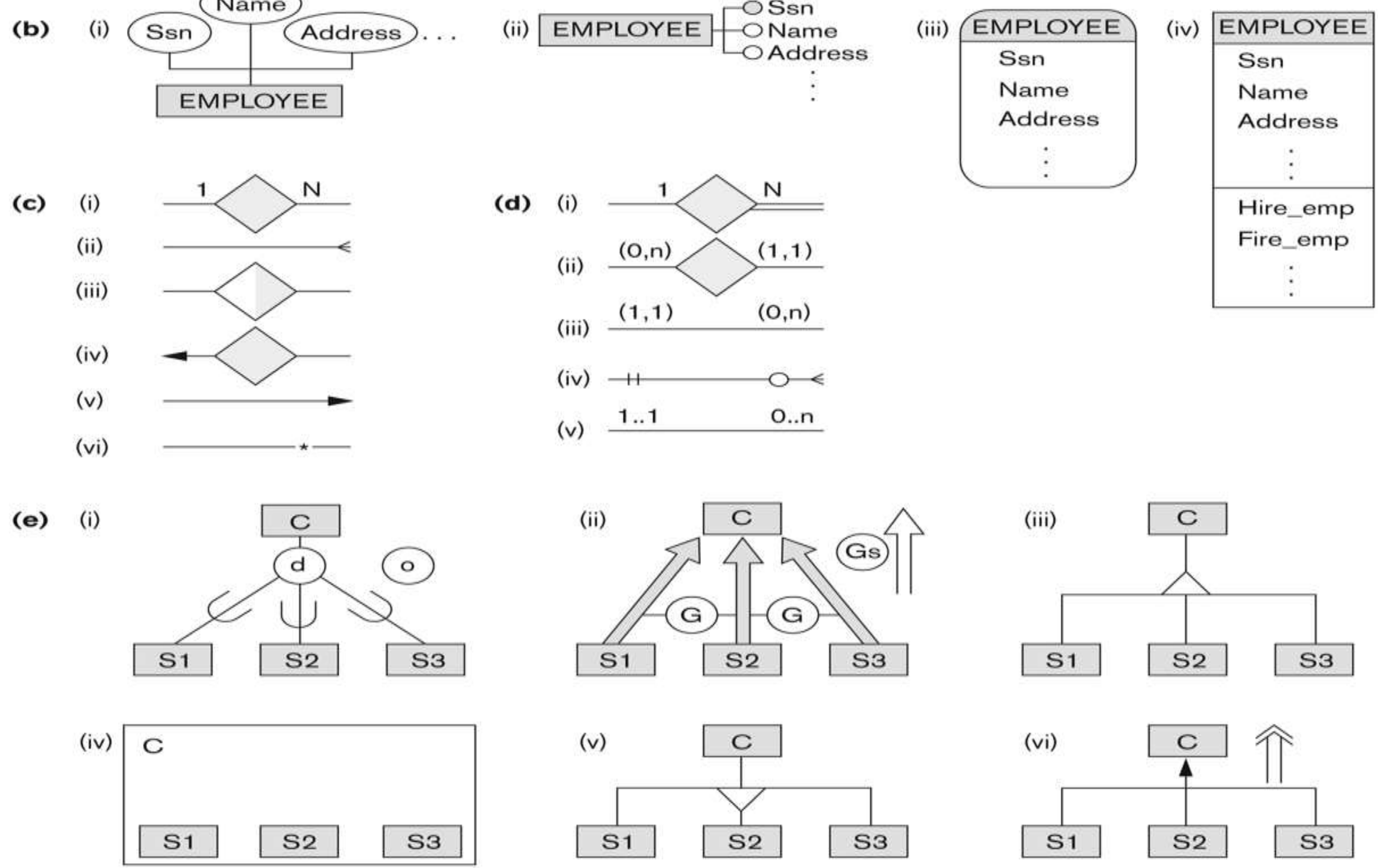


Figure A.1
Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

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)