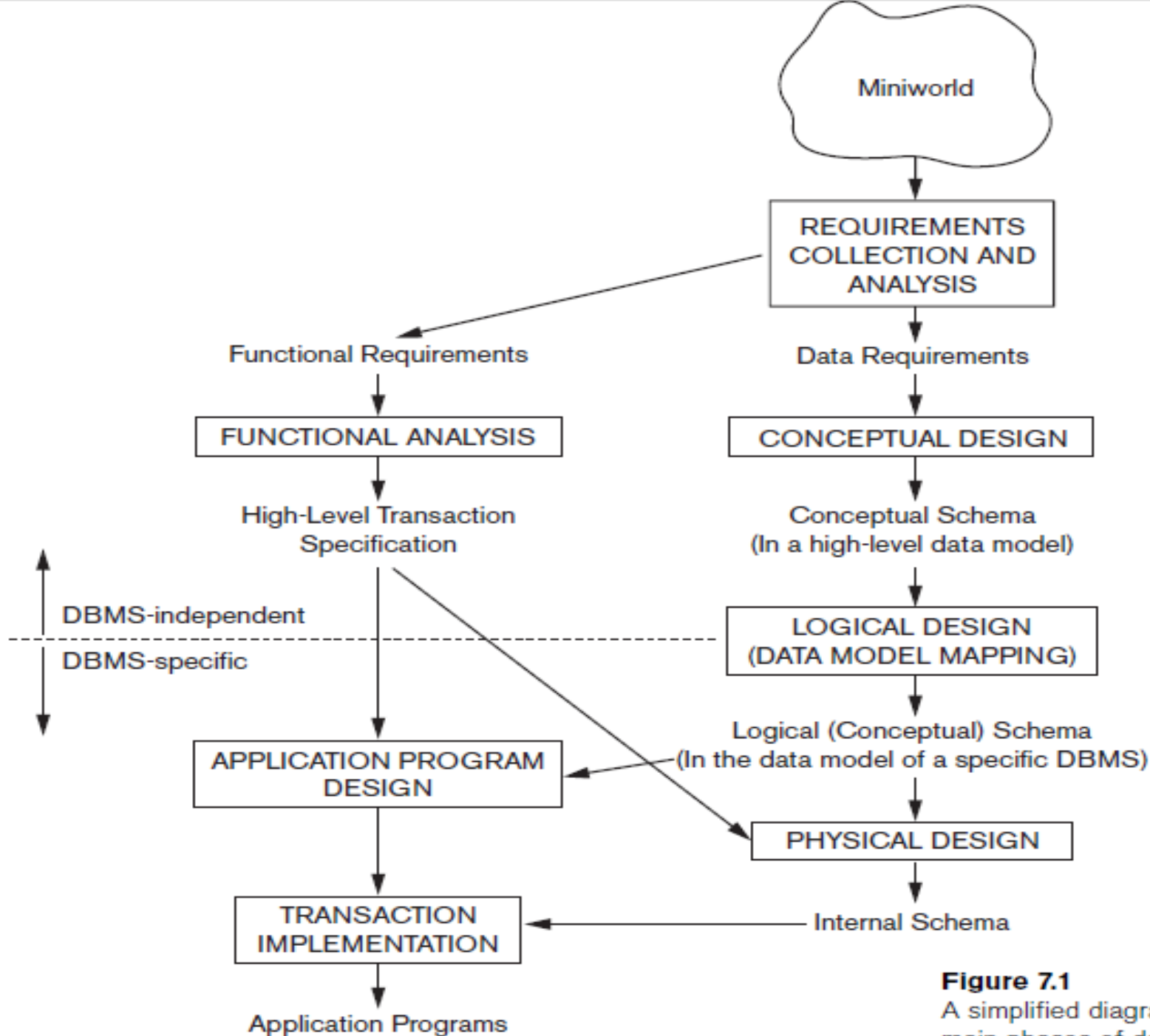


## Unit 2

# DATA MODELING USING ENTITY-RELATIONSHIP (ER) MODEL

# UNIT OUTLINE

- Using High level conceptual Data Models for Database Design
- 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



### Figure 7.1

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

# EXAMPLE COMPANY DATABASE

## Requirements of the Company

- 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.
- Each department *controls* a number of **PROJECTS**.
  - ✓ Each project has a name, number and is located at a single location.

➤ We store each **EMPLOYEE's** social security number, address, salary, Gender, and birth\_date.

✓ 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, birth\_date, and relationship to employee.

Entity Types

Entity Sets

Attributes

and

Keys

Keys

and

Attributes

# 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 Product X , PROJECT
- **Attributes** are properties used to describe an entity.
- For example an EMPLOYEE entity may have a Name, SSN, Address, 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, ...



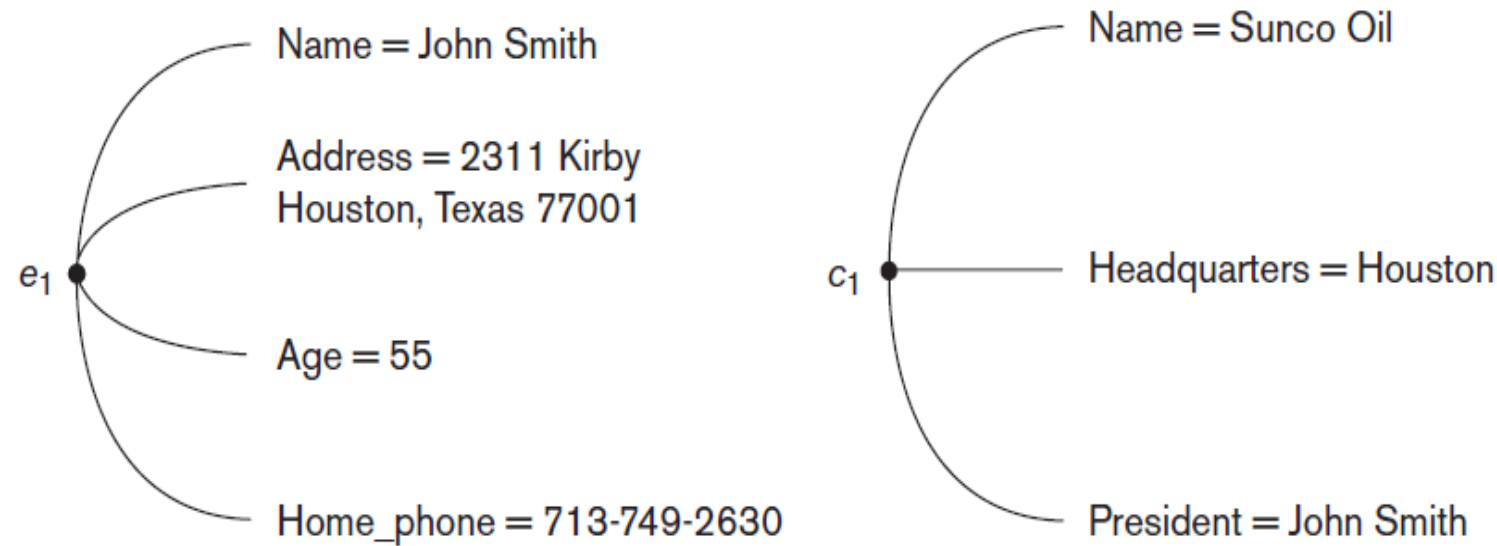
# TYPES OF ATTRIBUTES

## 1. Simple(Atomic)Attribute:

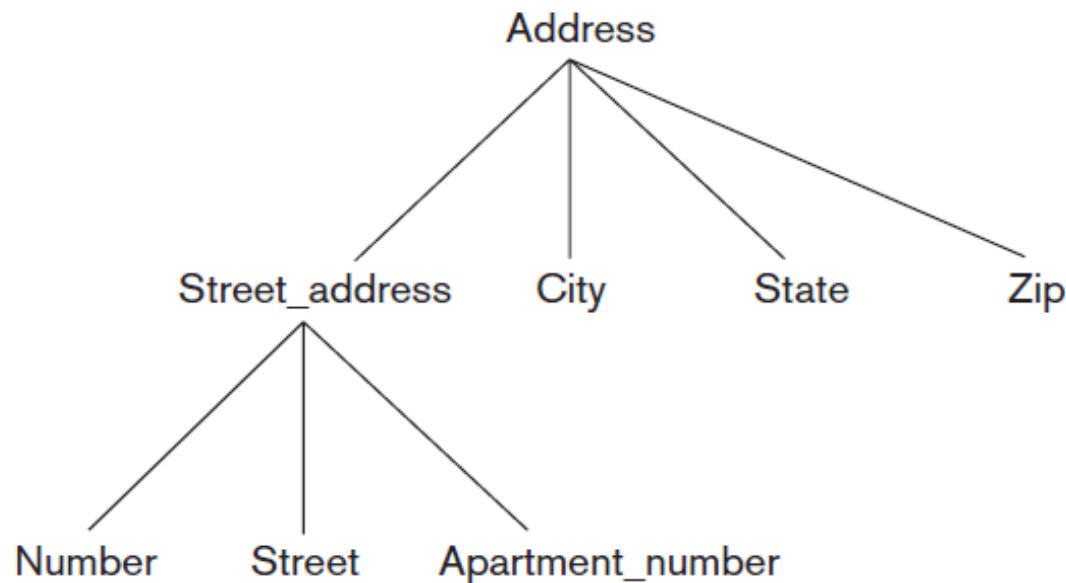
- Each entity has a single atomic value for the attribute.
- The attributes that are not divisible are called simple / atomic attributes.
- For example, Employee SSN, Student Roll#.

## 2. Composite Attribute

- The attribute may be composed of several components.
- The attributes can be divided into smaller subparts, which represent more basic attributes with independent meaning.
- For example,
  - Address (Apt#, House#, Street, City, State, ZipCode, Country)
  - Name (FirstName, MiddleName, LastName)
- Composite attributes may form hierarchy.
- The value of composite attribute is the concatenation of the values of its constituent simple attributes.



**Figure 7.3**  
Two entities,  
EMPLOYEE  $e_1$ , and  
COMPANY  $c_1$ , and  
their attributes.



**Figure 7.4**  
A hierarchy of composite  
attributes.

### 3. Single valued Attribute:

- Most of the attribute have a single value for a particular entity such attributes are called single valued attribute,
- For Example: Age is a single valued attribute of a person.

### 4. Multi-valued Attribute

- An entity may have multiple values for that attribute.
- For example, Color of a CAR - Denoted as {Color}
- PreviousDegrees of a STUDENT - Denoted as {PreviousDegrees}

## 5. Stored versus derived attribute

- In some cases the two or more attribute values are related.
- For example, Age and Birth\_date attribute of a person
- For a particular person entity, the value of Age can be determined from the current date and the value of the person's Birth\_date.
- The Age attribute is hence called derived attribute which is derived from Birth\_date attribute which is called stored attribute.

## 6. NULL values Attribute

- In some cases a particular entity may not have an applicable value for an attribute
- For Ex: Apartment# attribute applicable only for those who are staying in apartment.

## 7. Complex Attribute

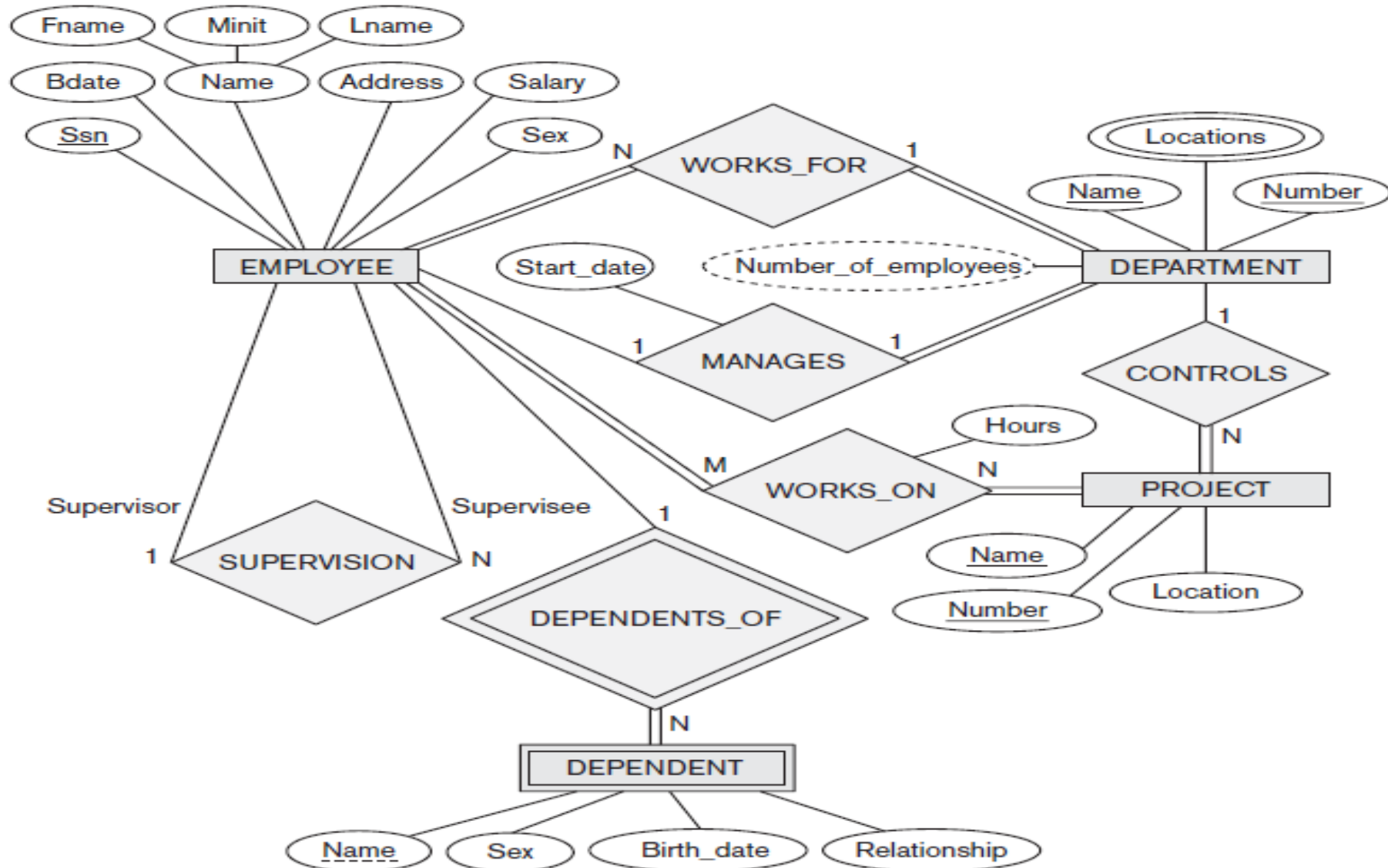
- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels.
- For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}.

```
{Address_phone( {Phone(Area_code,Phone_number)},Address(Street_address  
(Number,Street,Apartment_number),City,State,Zip) )}
```

**Figure 7.5**

A complex attribute:  
Address\_phone.

# ER DIAGRAM OF A COMPANY



**Figure 7.2**

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.



Entity Types

Entity Sets

Keys

Value Sets

Value Sets

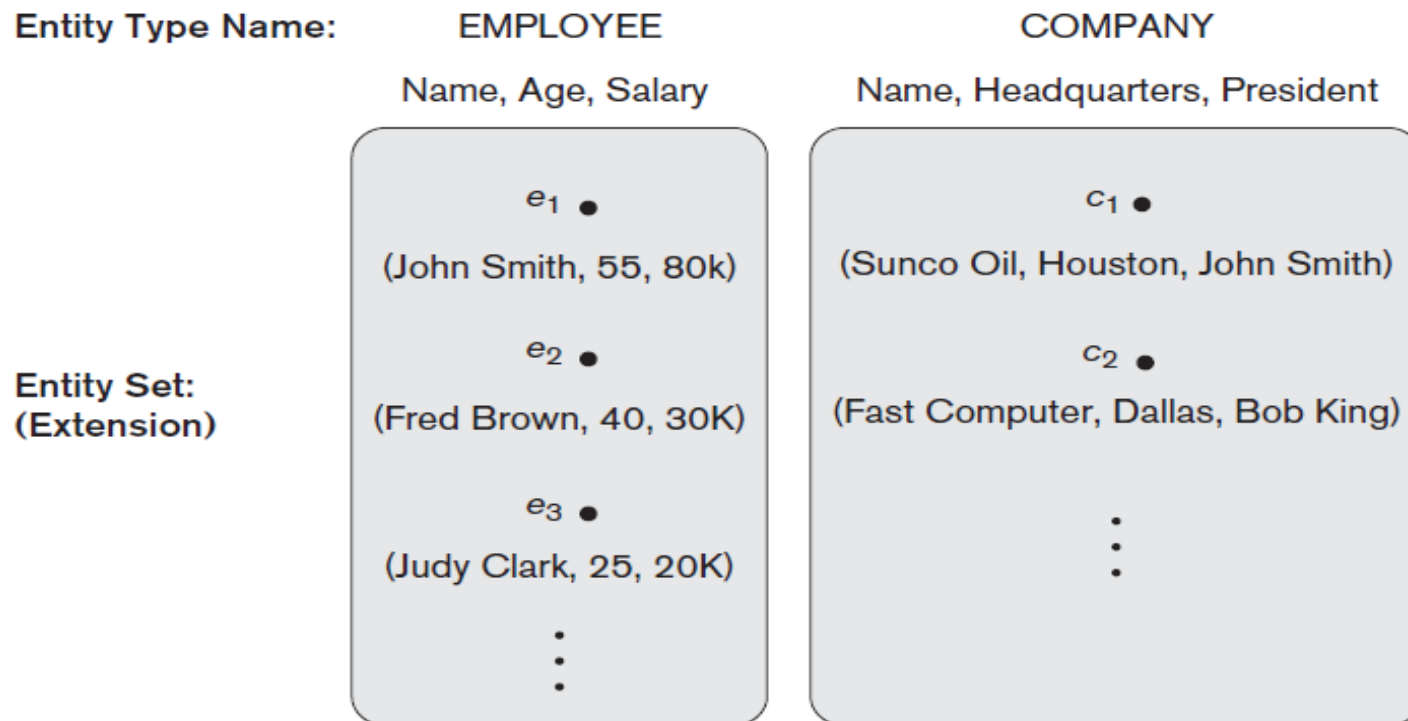
Keys

# ENTITY TYPES AND ENTITY SET

- A database usually contains group of entities that are similar.
- For a company employing hundreds of employees may want to store similar info concerning each of the employees.
- The employee entity share the same attributes but each entity has its own values for each attribute.
- An **entity type** defines a Collection of (or set) of entities that have the same attributes.

- Each entity type in the database is described by its name and attributes.
- Entity type is represented by rectangular box enclosing with the entity type name.

Figure 7.6 shows two entity types: EMPLOYEE and COMPANY, and a list of some of the attributes for each.



**Figure 7.6**  
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**
- The entity set is usually referred to using the same name as the entity type.
- For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.
- An entity type describes the **schema** or **intension** of a set of entities that share the same structure.
- The collection of entities of a particular entity type is grouped into an entity set called **extension** of the entity type.

# KEY ATTRIBUTES OF AN ENTITY TYPE

- An important constraint on the entities of a entity type is the key or uniqueness constraint on attributes.
- An entity type usually has an attribute whose values are distinct for each individual entity in the entity set, such attribute is called a **key attribute**.
- The values of key attribute can be used to identify each entity uniquely.
- For example, Name attribute is a key of the COMPANY entity type.

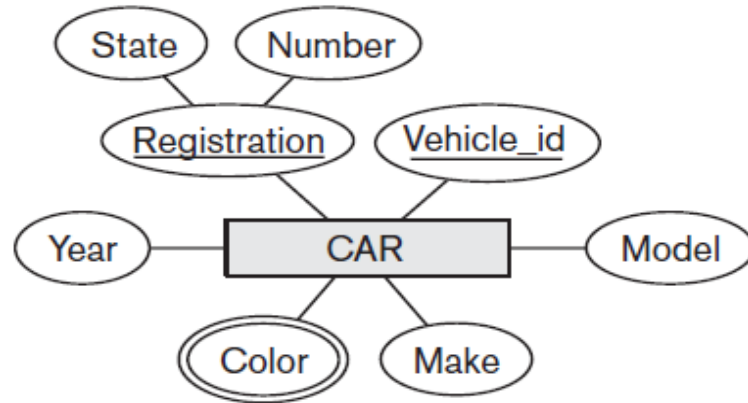
- A key attribute may be **composite**, Sometimes several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity.
- For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
- For example, the CAR entity type may have two keys:
  1. VehicleIdentificationNumber (popularly called VIN)
  2. VehicleTagNumber (Number, State), also known as license\_plate number.
- In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.

# ENTITY SET corresponding to the ENTITY TYPE CAR

**Figure 7.7**

The CAR entity type with two key attributes, Registration and Vehicle\_id. (a) ER diagram notation. (b) Entity set with three entities.

(a)



(b)

CAR  
Registration (Number, State), Vehicle\_id, Make, Model, Year, {Color}

CAR<sub>1</sub>  
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR<sub>2</sub>  
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR<sub>3</sub>  
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

# VALUE SETS (DOMAINS) OF ATTRIBUTES

- Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.
- For example, if the range of ages allowed for employees is between 16 and 70, we can specify the value set Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.
- Set of atomic values is called domain.
- Value sets are not displayed in ER diagrams.

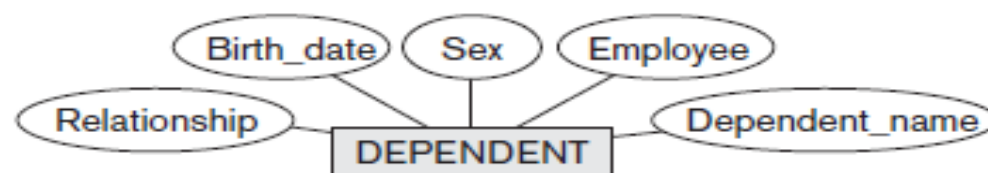
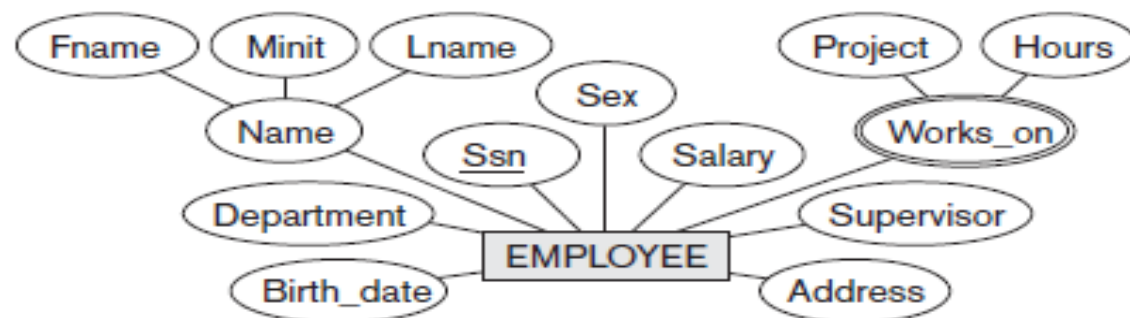
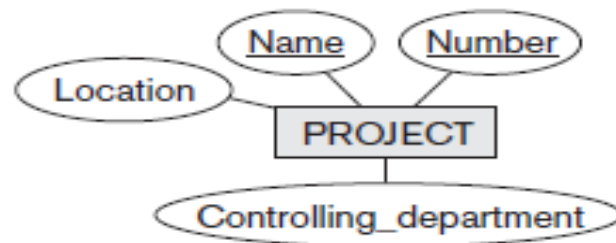
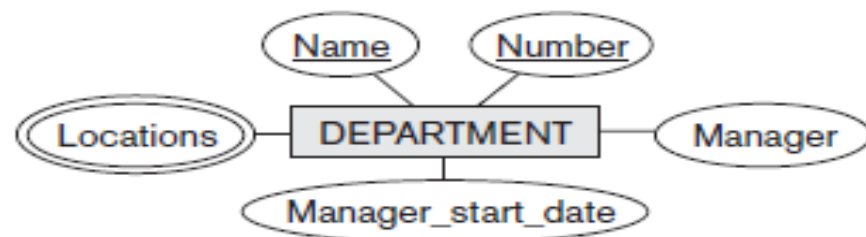


- Value sets are typically specified using the basic data types available in most programming languages.
- The data types are integer, string, Boolean, float, date, time and so on.

# INITIAL CONCEPTUAL DESIGN OF THE COMPANY DATABASE

- We can now define the entity types for the COMPANY database, based on the requirements described.
- 1. An entity type **DEPARTMENT** with attributes Name, Number, Locations, Manager, and Manager\_start\_date.
- Locations is the only multivalued attribute.
- We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.
- 2. An entity type **PROJECT** with attributes Name, Number, Location, and Controlling\_department.

- Both Name and Number are (separate) key attributes.
  - An entity type **EMPLOYEE** with attributes Name, SSN, Gender, Address, Salary, Birth\_date, Department, and Supervisor.
  - Both Name and Address may be composite attributes; however, this was not specified in the requirements.
  - We must go back to the users to see if any of them will refer to the individual components of Name—First\_name, Middle\_initial, Last\_name—or of Address.
4. An entity type **DEPENDENT** with attributes Employee, Dependent\_name, Gender, Birth\_date, and Relationship (to the employee).



**Figure 7.8**

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

Relationship Types,

Relationship Sets,

Roles, and

Structural Constraints

Structural Constraints

Roles, and

Relationship Sets,

# RELATIONSHIP TYPES, SETS, AND INSTANCES

- A **relationship type**  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations—or a **relationship set**—among entities from these entity types.
- As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the *same name*,  $R$ .

- Mathematically, the relationship set  $R$  is a set of relationship instances  $r_i$ , where each  $r_i$  associates  $n$  individual entities  $(e_1, e_2, \dots, e_n)$
- Each entity  $e_j$  in  $r_i$  is a member of entity set  $E_j$ ,  $1 \leq j \leq n$ .
- Hence, a relationship set is a mathematical relation on  $E_1, E_2, \dots, E_n$
- Alternatively, it can be defined as a subset of the Cartesian product of the entity sets  $E_1 \times E_2 \times \dots \times E_n$ .
- Each of the entity types  $E_1, E_2, \dots, E_n$  is said to participate in the relationship type  $R$ .
- similarly, each of the individual entities  $e_1, e_2, \dots, e_n$  is said to participate in the relationship instance  $r_i = (e_1, e_2, \dots, e_n)$ .

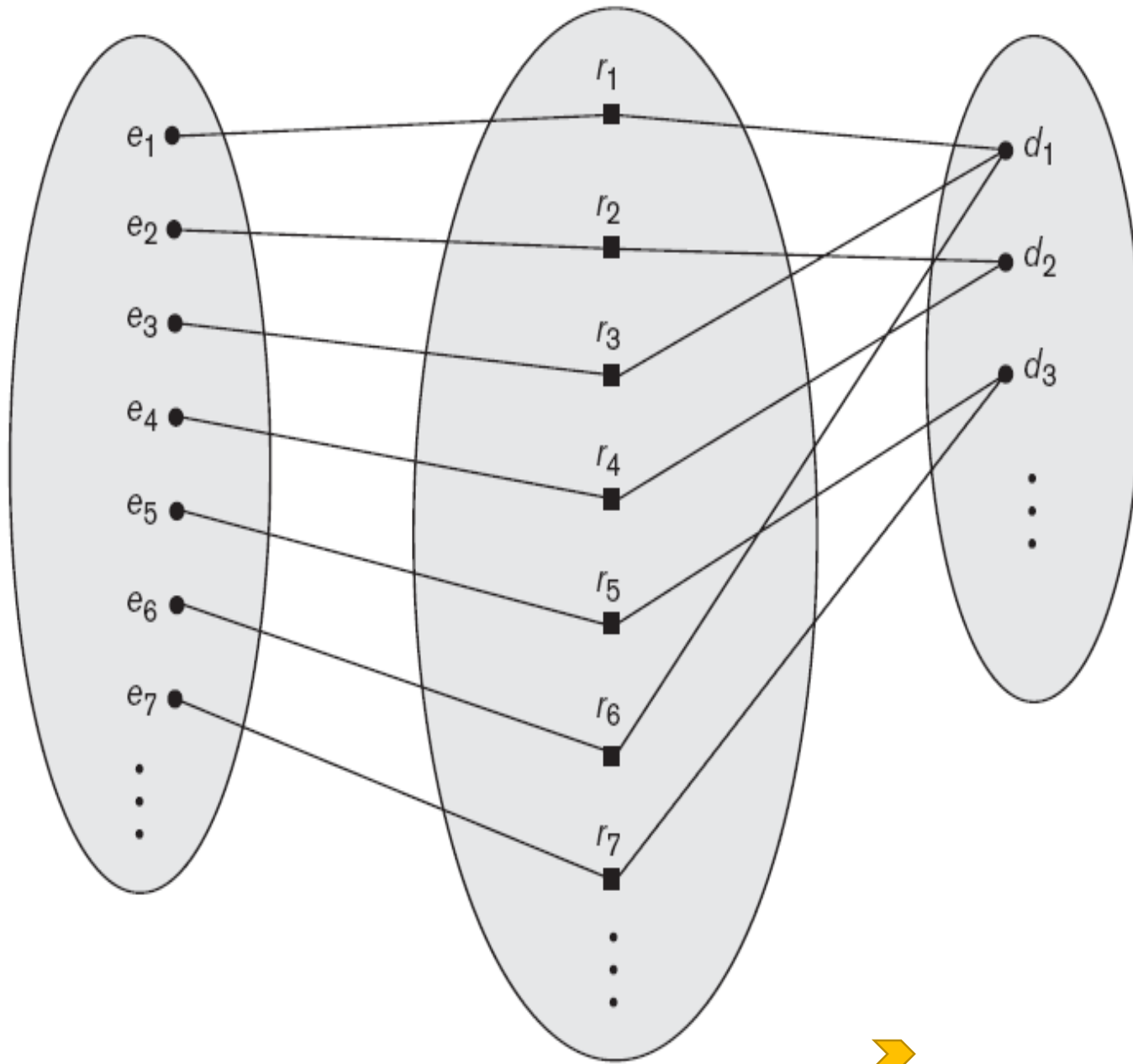
- Informally, each relationship instance  $r_i$  in  $R$  is an association of entities, where the association includes exactly one entity from each participating entity type.
- Each such relationship instance  $r_i$  represent the fact that the entities participating in  $r_i$  are related in some way in the corresponding mini-world situation.
- For example, consider a relationship type *works\_for* between two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works.
- Each relationship instance in the relationship set *works\_for* associates one EMPLOYEE entity and one DEPARTMENT entity



EMPLOYEE

WORKS\_FOR

DEPARTMENT

**Figure 7.9**

Some instances in the WORKS\_FOR relationship set, which represents a relationship type WORKS\_FOR between EMPLOYEE and DEPARTMENT.

Relationship Degree

Role Names and

Recursive Relationships

Recursive Relationships

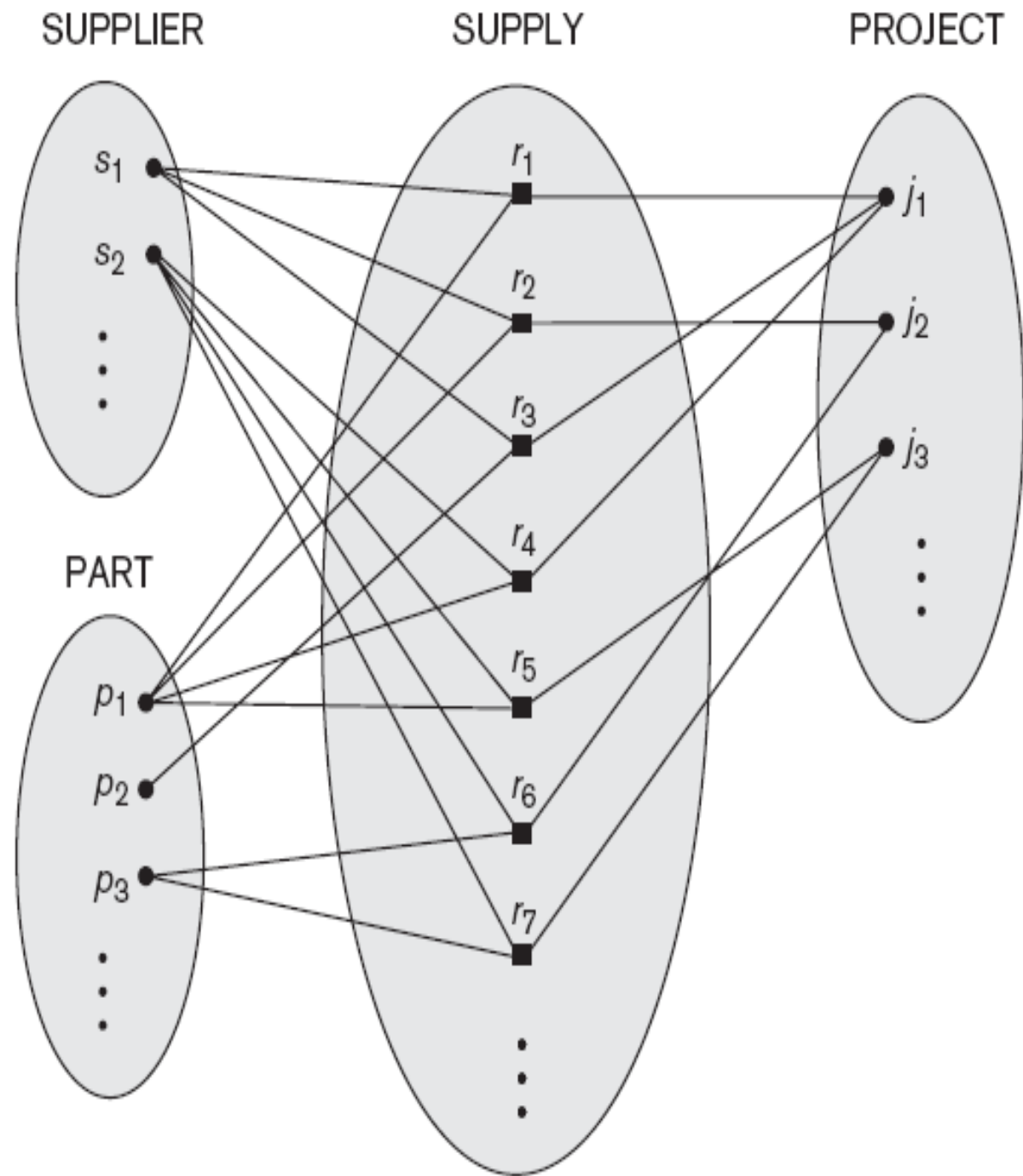
Role Names and

Relationship Degree

# DEGREE OF A RELATIONSHIP TYPE

- The **degree** of a relationship type is the number of participating entity types.
- Hence, the WORKS\_FOR relationship is of degree two.
- A relationship type of degree two is called **binary**, and one of degree three is called **ternary**

- An example of a ternary relationship is SUPPLY, where each relationship instance  $r_i$  associates three entities—a supplier  $s$ , a part  $p$ , and a project  $j$ .
- Whenever  $s$  supplies part  $p$  to project  $j$ .
- Relationships can generally be of any degree, but the ones most common are binary relationships.
- Higher degree relationships are generally more complex than binary relationships.



**Figure 7.10**

Some relationship instances in the **SUPPLY** ternary relationship set.

# RELATIONSHIPS AS ATTRIBUTES

- Consider the WORKS\_FOR relationship type in [Figure 7.9](#).
- One can think of an attribute called **Department** of the EMPLOYEE entity type,
- Where the value of **Department** for each EMPLOYEE entity is (a reference to) the DEPARTMENT entity for which that employee works.
- Hence, the value set for this **Department** attribute is the set of *all* DEPARTMENT entities, which is the DEPARTMENT entity set.

# ROLE NAMES AND RECURSIVE RELATIONSHIPS.

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- For example, in the WORKS\_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

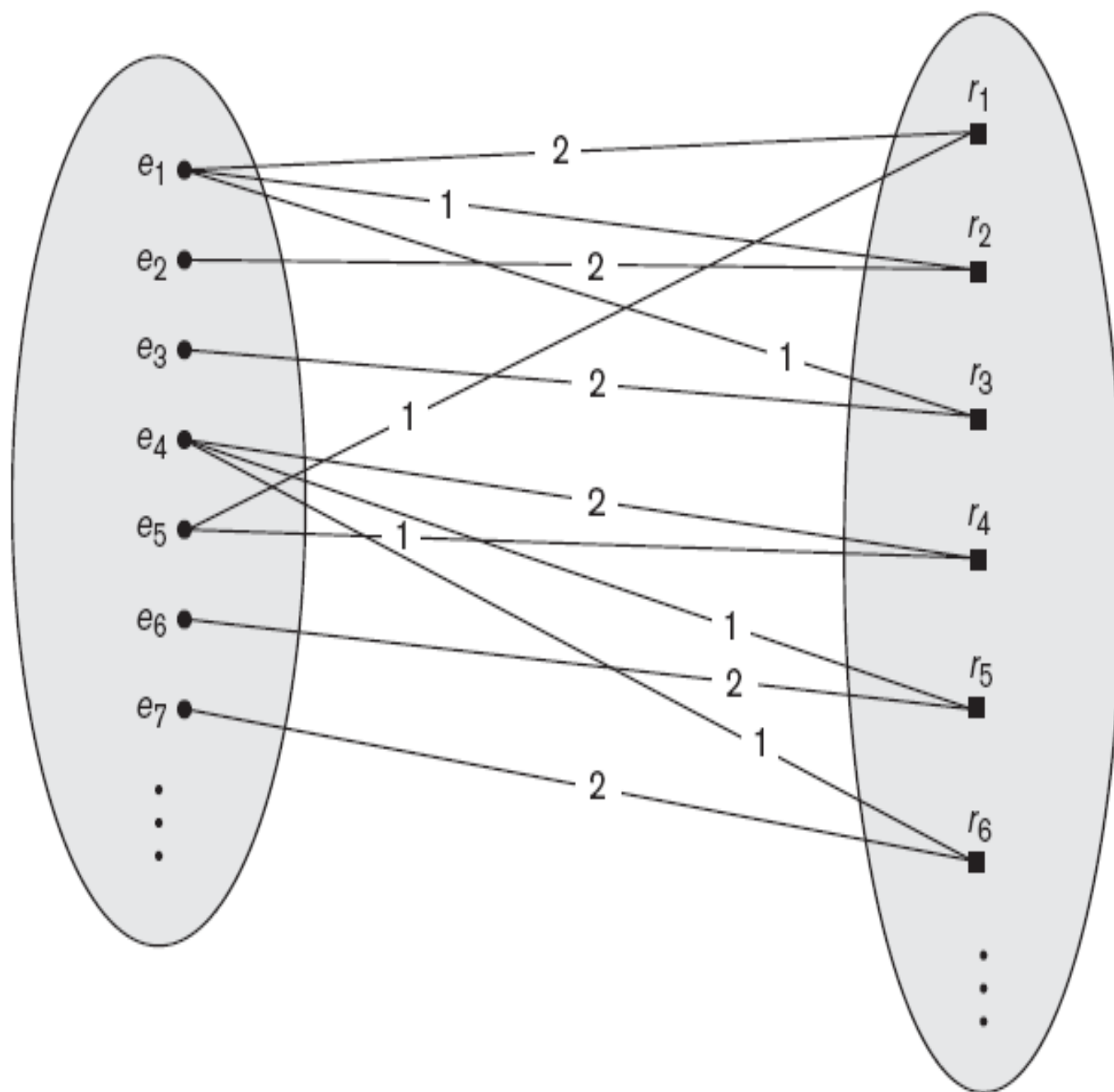
- Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name.
- However, in some cases the same entity type participates more than once in a relationship type in different roles.
- In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays.
- Such relationship types are called **recursive relationships**.
-



- For example, The SUPERVISION relationship type relates an employee to a supervisor,
- Where both employee and supervisor entities are members of the same EMPLOYEE entity set.
- Hence, the EMPLOYEE entity type participates twice in SUPERVISION:
- Once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate)

EMPLOYEE

SUPERVISION

**Figure 7.11**

A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

# CONSTRAINTS ON BINARY RELATIONSHIP TYPES

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- These constraints are determined from the mini-world situation that the relationship represents.
- For example, if a company has a rule that each employee must work for exactly one department.
- We can distinguish two main types of binary relationship constraints: *cardinality ratio* and *participation*.

# CARDINALITY RATIOS FOR BINARY RELATIONSHIPS

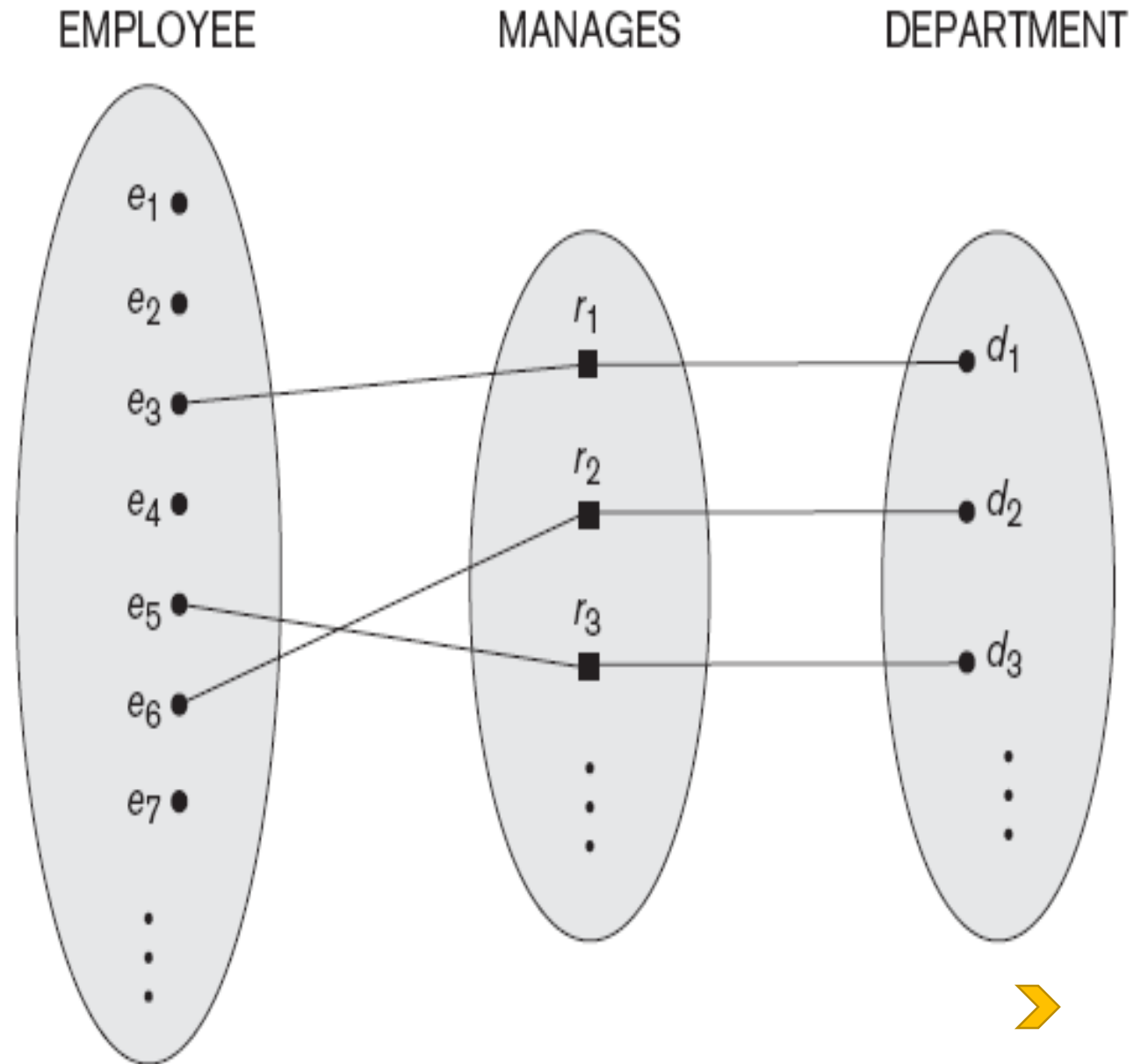
- The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in.
- Cardinality ratios for binary relationship types are
  - One-to-one (1:1)
  - One-to-many (1:N) or Many-to-one (N:1)
  - Many-to-many

- For example, in the WORKS\_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N.
- Meaning that each department can be related to (that is, employs) any number of employees.
- But an employee can be related to (work for) only one department.
- This means that for this particular relationship WORKS\_FOR, a particular department entity can be related to any number of employees (N indicates there is no maximum number).
- On the other hand, an employee can be related to a maximum of one department.

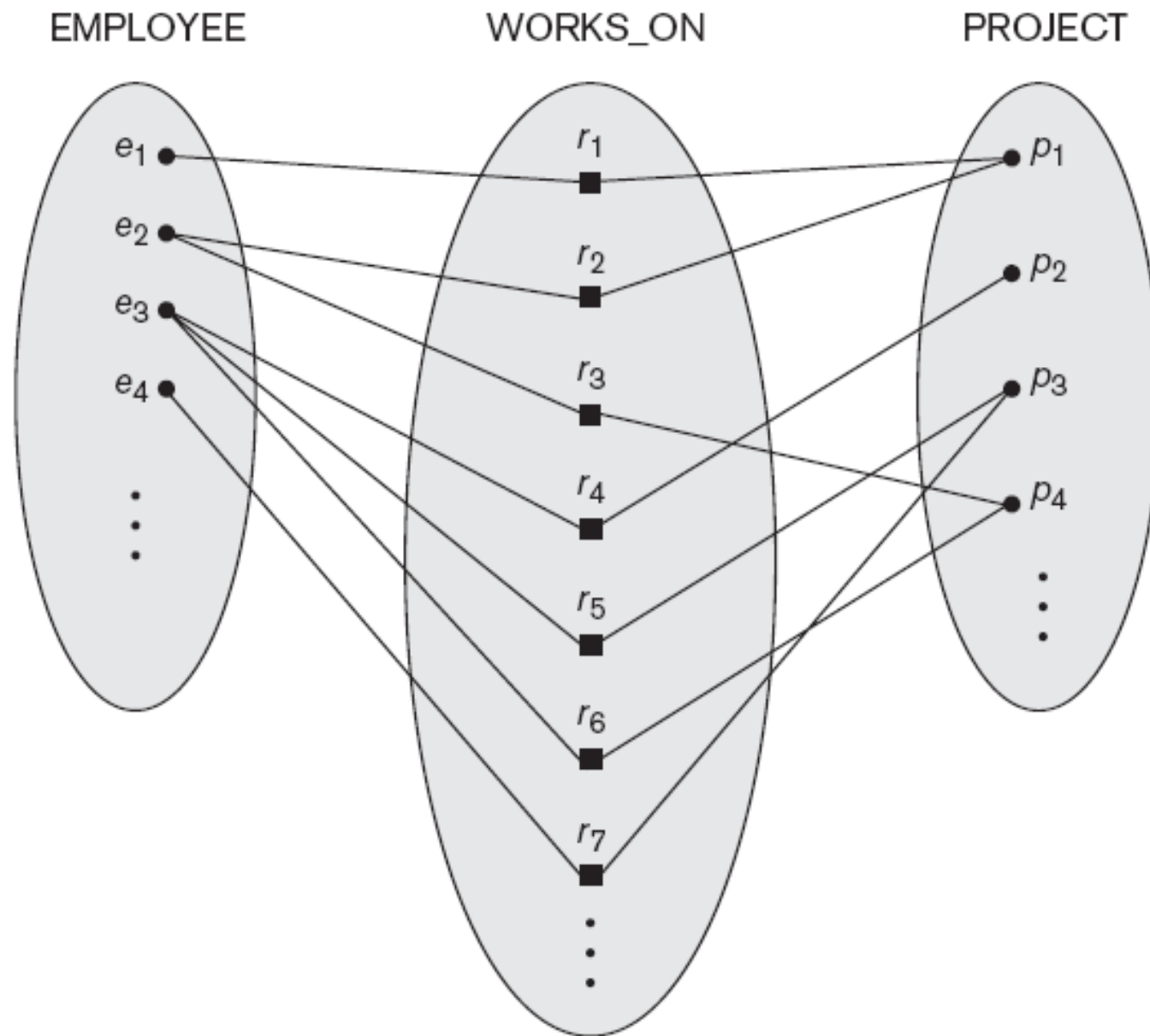
➤ An example for 1:1 binary relationship is MANAGES

**Figure 7.12**

A 1:1 relationship,  
MANAGES.



➤ An example for M:N binary relationship is WORKS\_ON



**Figure 7.13**  
An M:N relationship,  
WORKS\_ON.

# PARTICIPATION CONSTRAINTS AND EXISTENCE DEPENDENCIES.

- The **participation constraint** specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- This constraint specifies the *minimum* number of relationship instances that each entity can participate in, and is sometimes called the **minimum cardinality constraint**.
- There are two types of participation constraints—
  - 1) Total Participation
  - 2) Partial Participation



➤ We illustrate by example. If a company policy states that *every* employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS\_FOR relationship instance (Figure 7.9).

➤ Thus, the participation of EMPLOYEE in WORKS\_FOR is called **total participation**, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS\_FOR.

➤ Total participation is also called **existence dependency**.

➤ In Figure 7.12 we do not expect every employee to manage a department.

➤ So the participation of EMPLOYEE in the MANAGES relationship type is **partial**, meaning that *some or part of the set of* employee entities are related to some department entity via MANAGES, but not necessarily all.

➤ We will refer to the cardinality ratio and participation constraints, taken together, as the **structural constraints** of a relationship type.

➤ In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship.

➤ Whereas partial participation is represented by a single line (see Figure 7.2). Notice that in this notation, we can either specify no minimum (partial participation) or a minimum of one (total participation).

# ATTRIBUTES OF RELATIONSHIP TYPES

- Relationship types can also have attributes, similar to those of entity types.
- For example, to record the number of hours per week that an employee works on a particular project,
- we can include an attribute Hours for the WORKS\_ON relationship type in Figure 7.13.
- Another example is to include the date on which a manager started managing a department via an attribute Start\_date for the MANAGES relationship type in Figure 7.12.

# WEAK ENTITY TYPES

- Entity types that do not have key attributes of their own are called **weak entity types**.
- In contrast, **regular entity types** that do have a key attribute—which include all the examples discussed so far—are called **strong entity types**.
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type

➤ A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are *related to the same owner entity*.

➤ Entities are identified by the combination of:

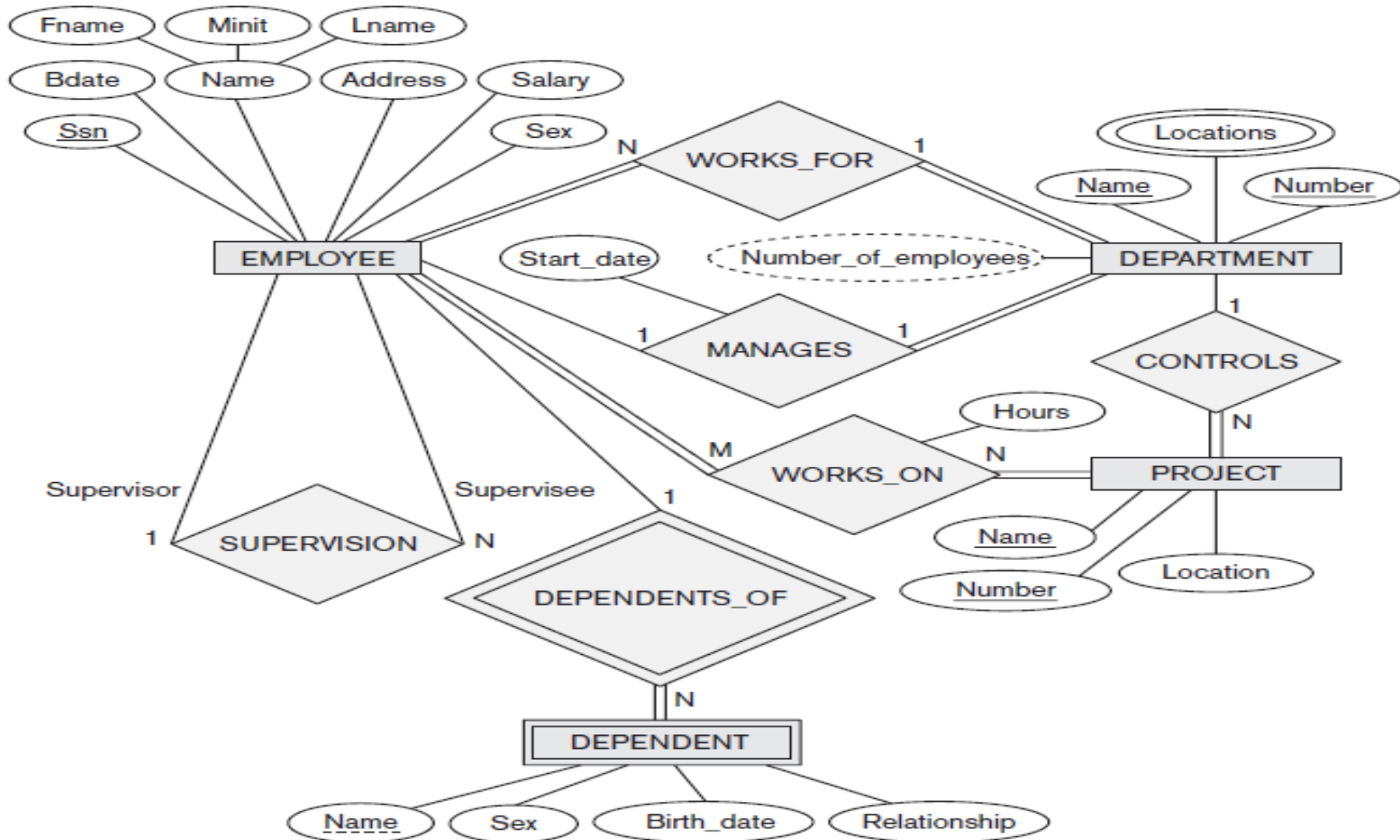
A partial key of the weak entity type and the particular entity they are related to in the identifying entity type

➤ Example: Suppose that a **DEPENDENT** entity is identified by the dependent's first name and birthdate, and the specific **EMPLOYEE** that the dependent is related to.

➤ **DEPENDENT** is a weak entity type with **EMPLOYEE** as its identifying entity type via the identifying relationship type **DEPENDENT\_OF**

# Weak Entity Type is: DEPENDENT

Identifying Relationship is: DEPENDENTS\_OF



**Figure 7.2**

An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

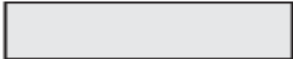
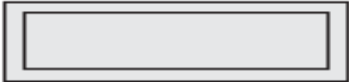





# Refining the ER Design for the COMPANY Database

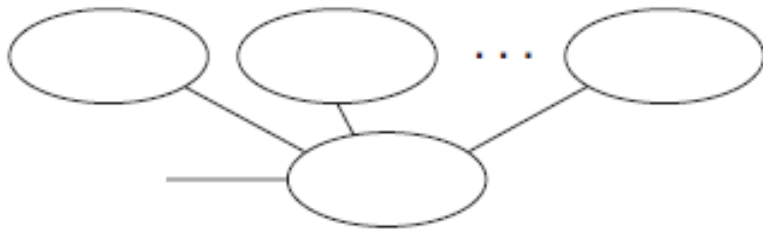
In our example, we specify the following relationship types:

- **MANAGES**, a 1:1 relationship type between **EMPLOYEE** and **DEPARTMENT**. **EMPLOYEE** participation is partial. **DEPARTMENT** participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation.<sup>13</sup> The attribute **Start\_date** is assigned to this relationship type.
- **WORKS\_FOR**, a 1:N relationship type between **DEPARTMENT** and **EMPLOYEE**. Both participations are total.
- **CONTROLS**, a 1:N relationship type between **DEPARTMENT** and **PROJECT**. The participation of **PROJECT** is total, whereas that of **DEPARTMENT** is determined to be partial, after consultation with the users indicates that some departments may control no projects.
- **SUPERVISION**, a 1:N relationship type between **EMPLOYEE** (in the supervisor role) and **EMPLOYEE** (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
- **WORKS\_ON**, determined to be an M:N relationship type with attribute **Hours**, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.
- **DEPENDENTS\_OF**, a 1:N relationship type between **EMPLOYEE** and **DEPENDENT**, which is also the identifying relationship for the weak entity



# NOTATION FOR ER SCHEMAS

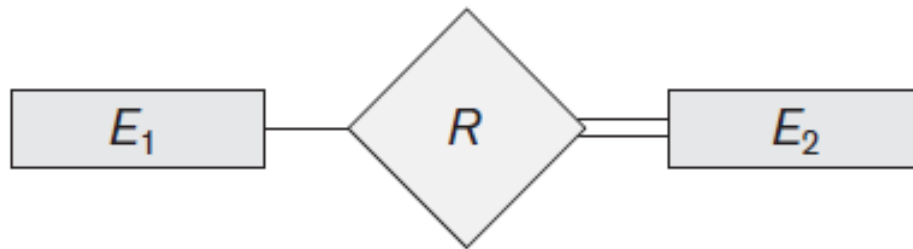
Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute



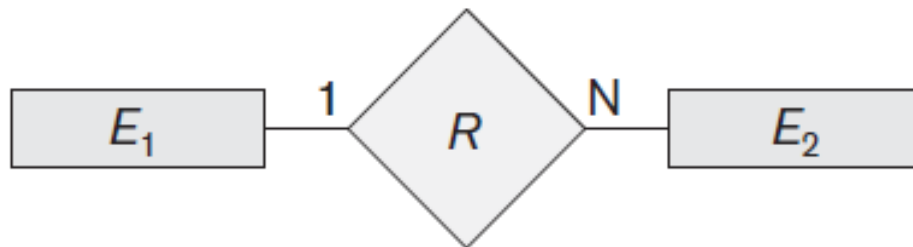
Composite Attribute



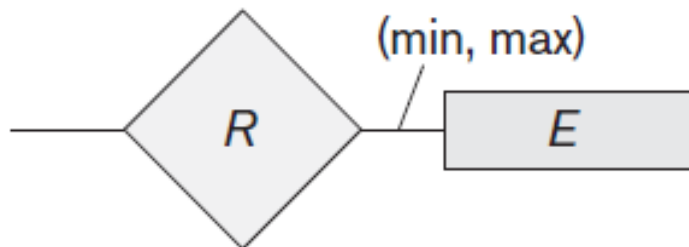
Derived Attribute



Total Participation of  $E_2$  in  $R$



Cardinality Ratio 1: N for  $E_1:E_2$  in  $R$



Structural Constraint (min, max)  
on Participation of  $E$  in  $R$

*THANK YOU*

# THE RELATIONAL MODEL

UNIT --2 – CHAPTER – 2

# RELATIONAL MODEL CONCEPTS

---

- The relational model represents the database as a collection of relations.
- Informally, each relation resembles a table of values or to some extent a flat file of records.
- The model is called a flat file because each file has a simple linear structure.
- When a relation is thought of as a table of values, each row in the table represents a collection of related data values.



- A row represents a fact that typically corresponds to a real world relationship.
- The table name and the column names are used to help to interpret the values in each row.
- For example,

NAME	ROLL_NO	SEM	MAJOR
AJAY	10	3	
PRAKASH	30	5	

- In this table each row represents a fact about a particular student enrolled in a particular semester.
- The column names – Name, Roll\_No, Sem, Major specify how to interpret the values in each row. [ All values in the columns is of same data type ]

# Domains, Attributes, Tuples & Rel

---

- A Domain  $D$  is a set of atomic values.
- By atomic we mean that each value in the domain is indivisible in the formal relational model is concerned.
- A common method for specifying a domain is to specify a domain  $D$  from which the data values forming the domain are drawn.

- It is also helpful to specify a name for the domain, to help in interpretation.
- Examples for domains,
  1. Phone\_Numbers: The set of 10 digit phone numbers is valid.
  2. Names: A set of character strings that represents a names of
  3. Employee\_Age: Possibles ages of a employee in a company.  
integer value between 20 and 60.



- A relation schema  $R$ , denoted by  $R(A_1, A_2, \dots, A_n)$  is made up of and a list of attributes  $A_1, A_2, \dots, A_n$ .
- Each attribute  $A_i$  is the name of a role played by some domain schema  $R$ .
- Here  $D$  is called the domain of  $A_i$  and is denoted by  $\text{dom}(A_i)$ .
- A relation schema is used to describe the relation.  $R$  is called relation.
- The degree of a relation is the number of attributes  $n$  of its relation.

- A relation of degree 7, which stores information about universities. Each record in the relation contains 7 attributes describing each student as,

STUDENT (Name, Roll#, Phone#, Address, MailID, Age, Major)

- Using the data type of each attribute, the above definition can be re-written as

STUDENT (Name: String, Roll#: String, Phone#: Integer, Address: String,  
MailID: String, Age: Integer, Major: String)

- Here, STUDENT is the name of the relation which has 7 attributes

- A relation ( or relation state)  $r$  of the relation schema  $R(A_1, A_2, \dots, A_n)$  by  $r(R)$ , is a set of  $N$  tuples  $r = \{t_1, t_2, \dots, t_n\}$ .
- Each  $n$ -tuple  $t$  is an ordered list of  $n$  values,  $t = \{v_1, v_2, v_3, \dots, v_n\}$ .
- The  $i^{\text{th}}$  value of tuple  $t$ , which corresponds to the attribute  $A_i$  is referred to as the  $i^{\text{th}}$  component of  $t$ .
- The term relation intension is used for the schema  $R$ .
- Relation extension is used for relation state  $r(R)$ .
- Example STUDENT relation



- The definition of a relation can be more stated more formally  
 concepts,
  - ✓ A relation  $r(R)$  is a mathematical relation of degree  $n$  on the  $d$  domains  $dom(A_1), dom(A_2), \dots, dom(A_n)$ , which is the subset of the cartesian product of the domains that defines  $R$ .
  - ✓  $R \subseteq (dom(A_1), dom(A_2), \dots, dom(A_n))$

# Characteristics of Relations

---

- Ordering of tuples in a relation
- Ordering of values within a tuple
- Values and NULLs in the tuples
- Interpretation of a relation

# 1. Ordering of tuples in a relation

---

- A relation is defined as set of tuples.
- Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order.
- A relation is not sensitive to the ordering of tuples.

# Ordering of values within a tuple

---

An alternative definition of a relation.

- According to the previous definition of a relation, an  $n$ -tuple is an ordered set of  $n$  values, the ordering of values in a tuple and attributes in a relation is important.
- However, at more abstract level, the order of attributes and their values is not important as long as the correspondence between attributes and values is maintained.



# Values and NULL's in the Tuple

---

- Each value in a tuple is atomic value. [ composite and multivalued are not allowed].
- This model is sometimes called as **flat relational model**.
- The NULL values are used to represent the values of attributes unknown or may not apply to a tuple.
- Several meaning for NULL values: Value unknown, value exists but not available, attribute doesn't apply.



# Interpretation of a Relation

---

- The relation schema can be interpreted as declaration or a type.
- For example, the schema of the STUDENT relation asserts that an entity has a Name, Roll#, Mobile.. Etc attributes.
- Each tuple in the relation can then be interpreted as a **fact** or an instance of the assertion.
- For example, the first tuple in the STUDENT relation asserts that there is a student whose name is Prakash, Roll# 10 and etc.

# Relational Model Notation

- ✓ A relation schema  $R$  of degree  $n$  is denoted by  $R(A_1, A_2, \dots, A_n)$ .
- ✓ The Uppercase letters  $Q, R, S$  denote relation names.
- ✓ The lower case letters  $q, r, s$  denote relation states.
- ✓ The letter  $t, u, v$  denote tuples.
- ✓ An attribute  $A$  can be qualified with the relation name  $R$  to which it belongs by using the dot notation  $R.A$

- ✓ For example, STUDENT.Name or STUDENT.Address
- ✓ This is because the same name may be used for two attributes in different relations
- ✓ An n-tuple  $t$  in a relation  $r(R)$  is denoted by  $t = \langle v_1, v_2, \dots, v_n \rangle$ , where  $v_i$  is the value of attribute  $A_i$  in  $t$ , corresponding to value attribute  $A_i$ .
- ✓ Both  $t[A_i]$  and  $t.A_i$  refer to the value  $V_i$  in  $t$  for attribute  $A_i$ .
- ✓ **example:** consider the tuple  $t = \langle \text{'Kevin Williamson'}, \text{'553889'}, 89789 \rangle$   
 $t[\text{name}] = \langle \text{'Kevin Williamson'} \rangle$  and  $t[\text{ssn}, \text{age}] = \langle \text{'553889'}, 40 \rangle$



# Relational Model Constraints & Relational Database Schemas

---

- Here we will discuss the various restrictions on data that can be stored in a relational database in the form of constraints.
- Constraints on databases is divided into 3 main categories,
  1. Inherent model-based constraints / Implicit Constraints
  2. Schema-based constraints / Explicit constraints
  3. Application based / Semantic constraints / Business Rules

✓ Constraints that are inherent in the data model, we call this as a **data model based constraint**.

✓ Constraints that can be directly expressed in schemas of the data model, specifying them in DDL, we call this as **schema based constraints**.

✓ Constraints that cannot be directly expressed in the schemas of the data model, hence must be expressed and enforced by the application program, we call this as **application based constraints**.

✓ The characteristics of relations are inherent constraints.

✓ For example, Some attributes should not have duplicate values.  
This is a constraint, that we call it as schema based.

✓ Some constraints that cannot be directly applied on data model [ t  
those constraints can be applied by using application program,  
application based constraints.



- Another important category of constraints is **data dependencies** which include **functional dependencies** and **multivalued dependencies**.
- These constraints are used mainly for testing the goodness of relational database.
- They are utilized in the process of **normalization**.
- The schema based constraints include,
  1. Domain Constraints
  2. Key Constraints
  3. Constraints on NULLs
  4. Entity Integrity Constraints
  5. Referential Integrity Constraints

# Domain Constraints

---

- Domain constraints specify that within each tuple, the value of  $A$  must be an atomic value from the domain  $\text{dom}(A)$ .
- The data types associated with domains include integers, characters, Booleans, fixed-length strings, variable-length strings



# Key Constraints & Constraints on NU

---

- In the formal relational model, a relation is defined as a set of tuples.
- By definition all elements of a set are distinct; hence, all tuples must also be distinct.
- Meaning that no two tuples can have the same combination of their attributes.

- Usually, there are other subset of attributes of a relation schema  $R$  that no two tuples in any relation state  $r(R)$  should have the same values for these values.
- Suppose that we denote one such subset of attributes by  $SK$ , distinct tuples  $t_1$  and  $t_2$  in a relation state  $r(R)$ , we have the constraint

$$t_1[SK] \neq t_2[SK]$$

- Any such set of attributes  $SK$  is called as **Super Key** of the relation.
- A Super Key specifies the uniqueness constraints that no two distinct tuples in any relation state  $r(R)$  can have the same value for  $SK$ .
- Every relation has at least one default super key.

- In general, a relation schema may have more than one key, in this key is called a **candidate key**.
- For example, the CAR relation has two candidate keys Engine\_Serial#.
- It is common to designate one of the candidate key as **Primary Key**.
- This is the candidate key whose values are used to identify tuples in the relation.
- We use the convention that the attributes that form the primary key in a relation schema are underlined.
- If we want to have all the tuples should have values, we use NOT NULL.



# Relational Databases & Relational Database Schemas

---

- A relational database schema  $S$  is a set of relation schemas  $S = \{R_1, R_2, \dots, R_n\}$  and set of Integrity Constraints  $IC$ .
- A relational database state  $DB$  of  $S$  is a set of relation states  $DB = \{r_1, r_2, \dots, r_n\}$  such that each  $r_i$  is a state of  $R_i$  And  $r_i$  relation states satisfy the integrity constraints.

- For example, relational database schema of a COMPANY,  
COMPANY={EMPLOYEE, DEPARTMENT, DEPT\_LOCATIONS, PROJECT, WORKS\_

#### EMPLOYEE

Fname	M_Name	Lname	SSN	Bdate	Address	Salary
-------	--------	-------	-----	-------	---------	--------

#### DEPARTMENT

Dname	Dno	Mgr_SSN	Mgr_Start
-------	-----	---------	-----------

#### DEPT\_LOCATIONS

Dno	Dloc
-----	------

#### PROJECT

Pname	Pno	Ploc	Dno
-------	-----	------	-----

#### WORKS\_ON

ESSN	Pno	Hours
------	-----	-------

#### DEPENDENT

ESSN	Dept_Name	Gender	Bdate	Relationship
------	-----------	--------	-------	--------------

- When we refer to relational database, we implicitly include both its **current state**.
- A database state that does not obey all the integrity constraints is called an **invalid state**.
- A state that satisfies all the constraints in the defined set of integrity constraints is called a **valid state**.

# Integrity, Referential Integrity & Foreign

---

- The entity integrity constraint states that no primary key value can be NULL.
- This is because the primary key value is used to identify individual tuples in a relation.
- Having NULL values for the primary key implies that we cannot uniquely identify some tuples.
- Key constraints and entity integrity constraints are specified for each relation.



- The **referential integrity** constraint is specified between two relations to maintain the consistency among the tuples in two relations.
- Informally, the **referential integrity** constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
- For example, the attribute Dno of EMPLOYEE gives the department number to which each employee works; hence, its value in every EMPLOYEE tuple must match the Dno value of some tuple in the DEPARTMENT relation.
- To define the referential integrity more formally, we have to define it more first.



- A set of attributes FK in relation schema R1 is a foreign key of R1 to relation R2 if it satisfies the following rule,

1. The attributes in the FK have the same domain as the primary key of R2; The attributes FK are said to reference or refer to the primary key of R2.
2. A value of FK in tuple t1 of the current state r1(R1) either occurs as a value of the primary key of R2 or is NULL.

In the former case, we have  $t1[FK] = t2[PK]$ , we can say that t1 references the tuple t2.

- In this definition,  $R_1$  is called the **referencing relation** and  $R_2$  **referenced relation**.
- If these two condition hold, a **referential integrity constraint** from  $R_1$  to  $R_2$  is said to hold.

# Update Operations, Transactions, and with Constraint Violations

---

- There are 3 basic update operations on relations; **insert**, **delete**, and **update**.
- They insert new data, delete old data and modify existing data to change the state of the database.
- **Insert** is used to insert a new tuple or tuples in a relation.
- **Delete** is used to delete tuples
- **Update** / **Modify** is used to change the values of some attributes of tuples.



# The Insert Operation

- The insert operation provides a list of attribute values for a new tuple inserted into a relation R.
- Insert can violate any of the four types of constraints,
  - ✓ Domain constraint can be violated if an attribute value is given that doesn't belong to the corresponding domain.
  - ✓ Key constraints can be violated if a key value in the new tuple  $t$  already exists in the relation.
  - ✓ Entity integrity can be violated if the primary key of the new tuple  $t$  is NULL.
  - ✓ Referential integrity can be violated if the value of any foreign key in  $t$  references a tuple that doesn't exist in the referenced relation.

# The Delete Operation

---

- The delete operation can violate only referential integrity.
- If tuples being deleted is referenced by the foreign keys from other tables in the database.

# The Update Operation

---

- The update / modify operation is used to change the values of attributes in a tuple / tuples of some relations
- It is necessary to specify a condition on the attributes of the relation / tuples to be modified.
- Updating neither primary nor foreign key creates no problems