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

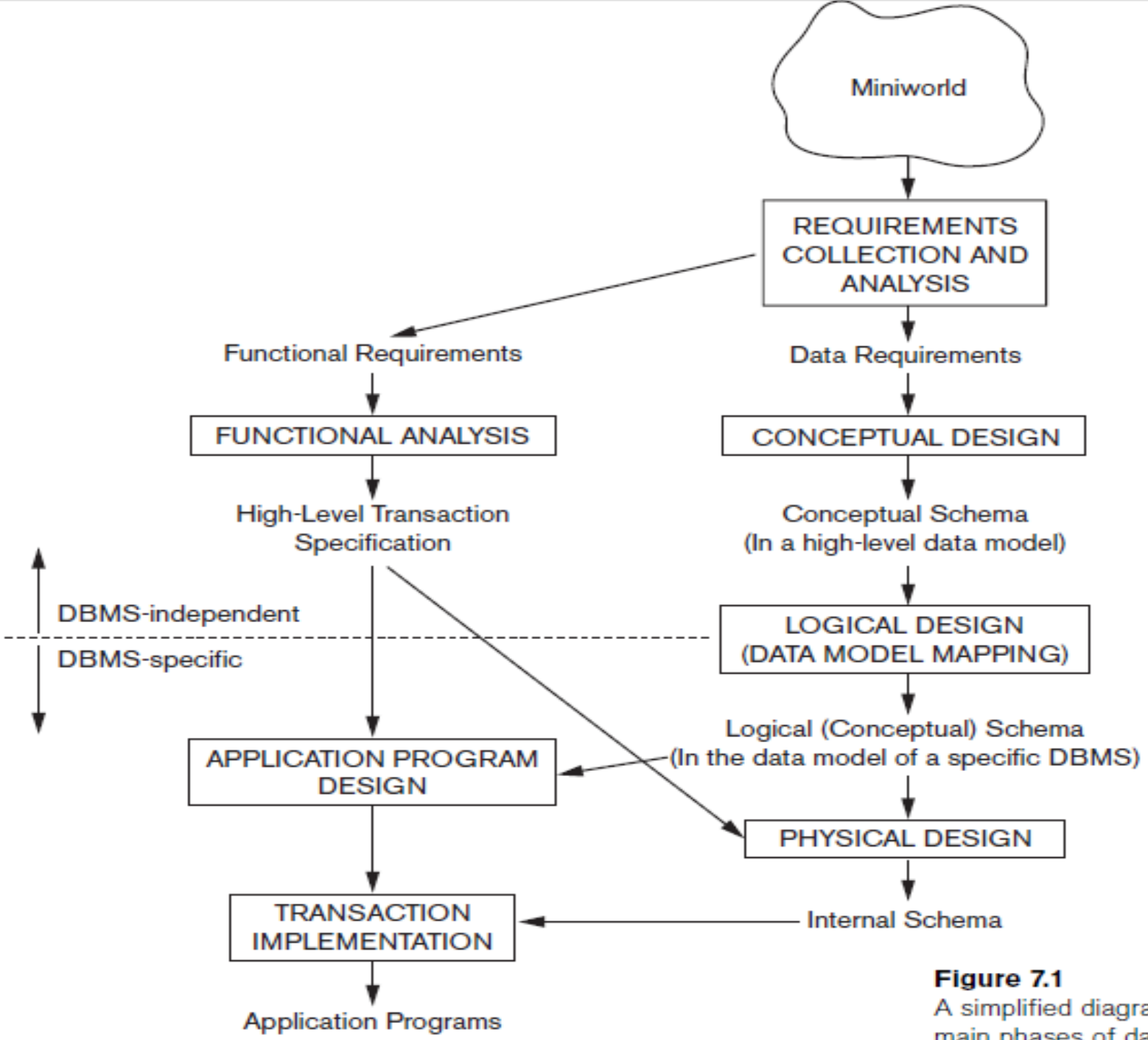


Figure 7.1
A simplified diagram to illustrate the main phases of database design.

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

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.

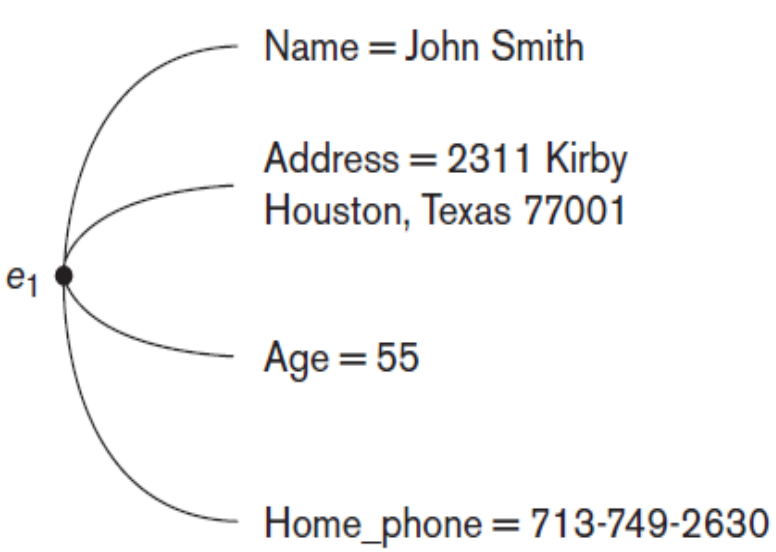
- A specific entity will have a value for each of its attributes.
- For example a specific employee entity may have,
  - ✓ Name='John Smith',
  - ✓ SSN='123456789',
  - ✓ Address ='731, Fondren, Houston, TX',
  - ✓ Sex='M',
  - ✓ BirthDate='09-JAN-55'
- Each attribute has a *value set* (or data type) 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.



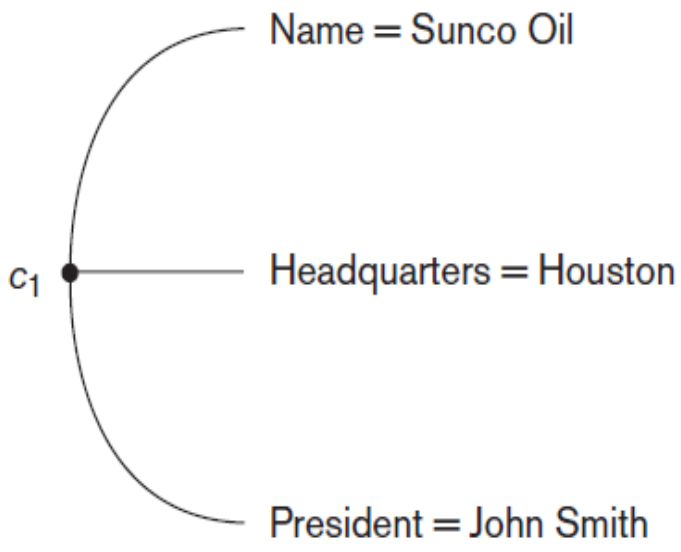


Figure 7.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

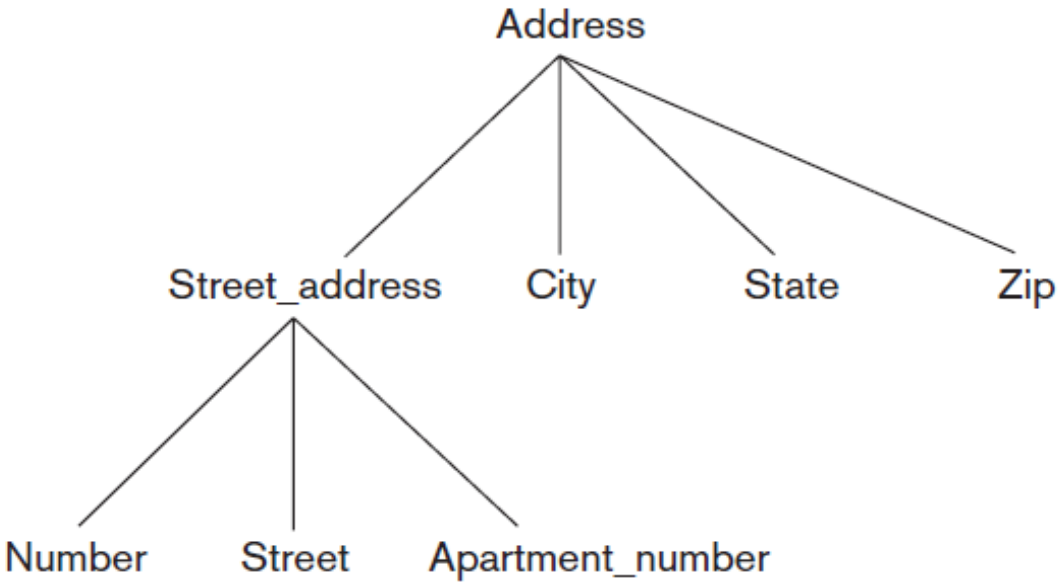


Figure 7.4
A hierarchy of composite attributes.

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

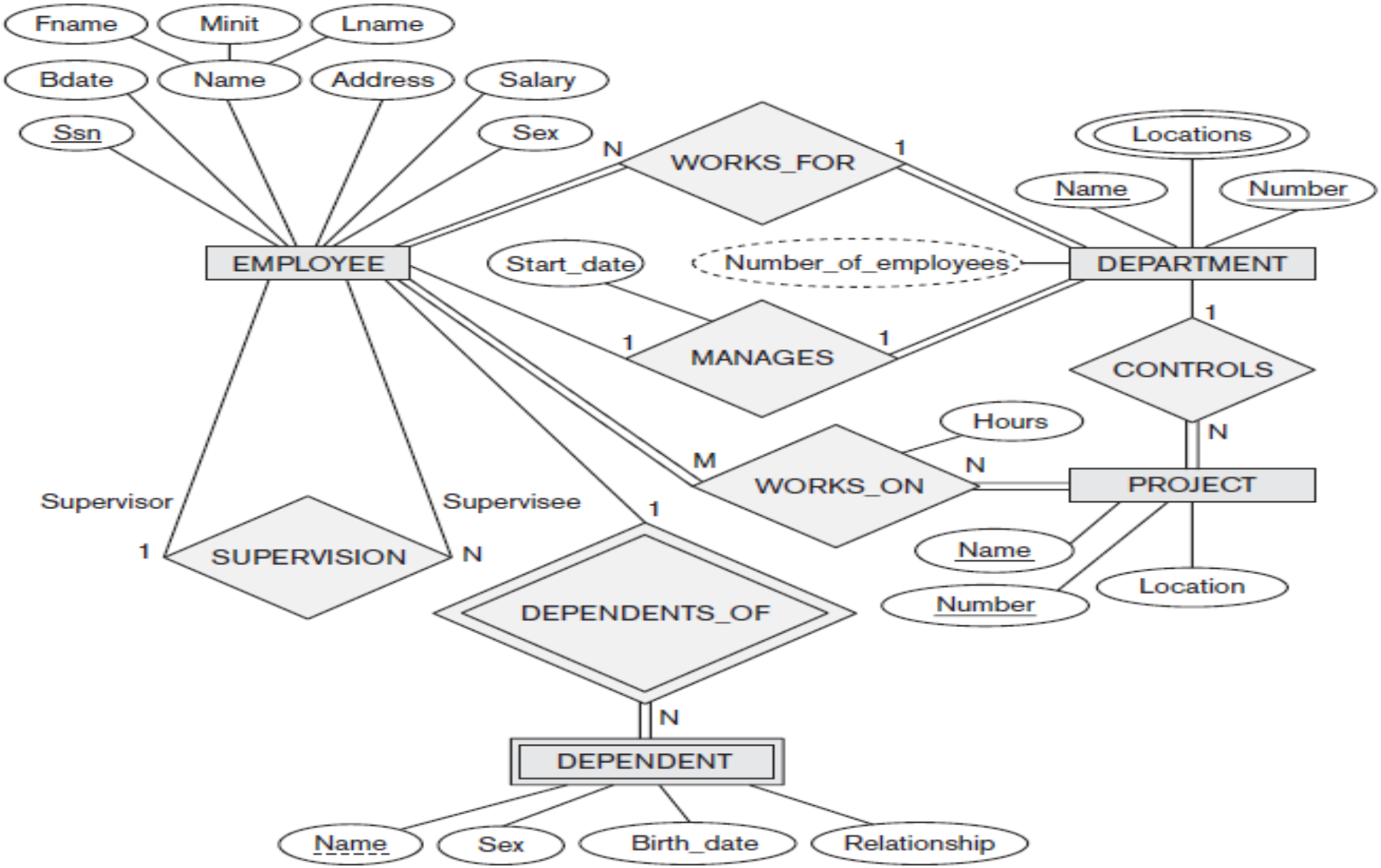


Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

Entity Types

Entity Sets

Keys

Value Sets

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.

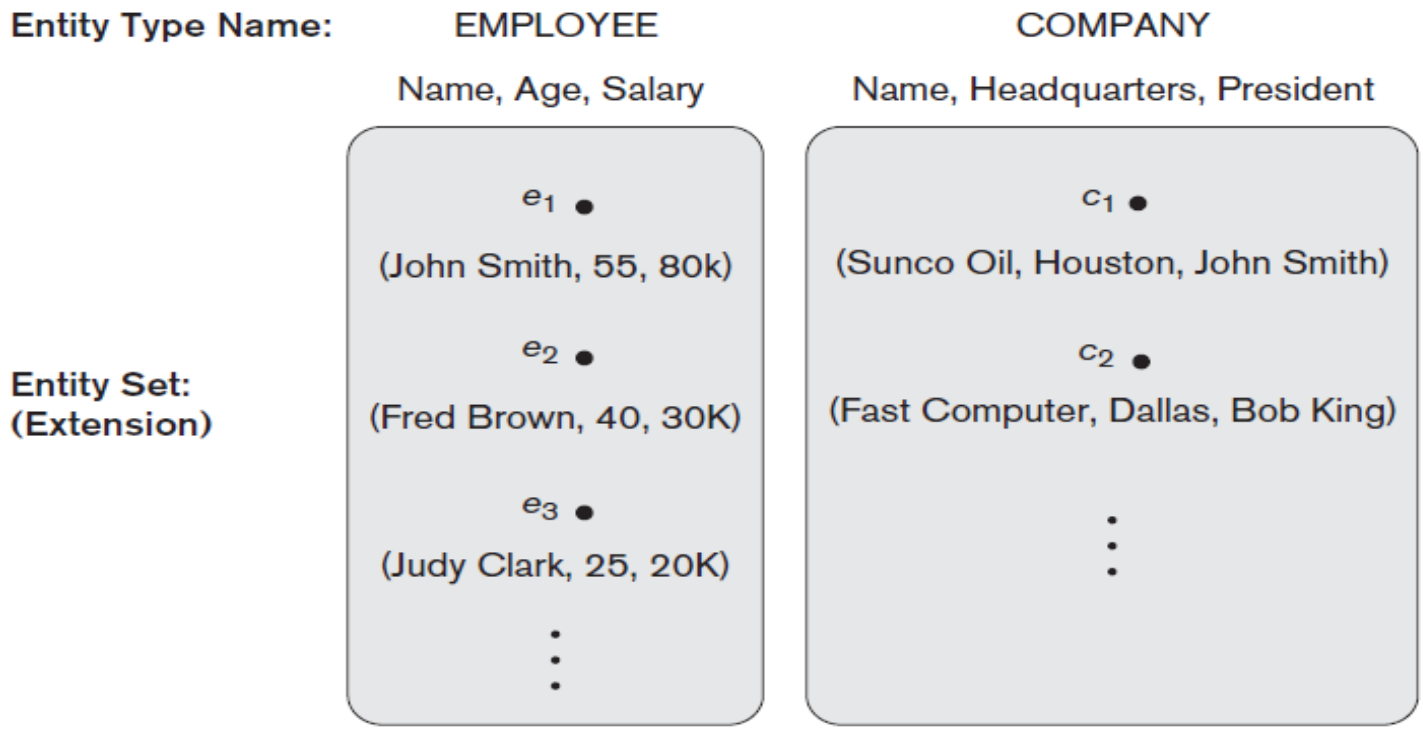


Figure 7.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- 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 groped into an entity set called **extension** of the entity type.

KEY ATTRIBUTES OF AN ENTITY TYPE

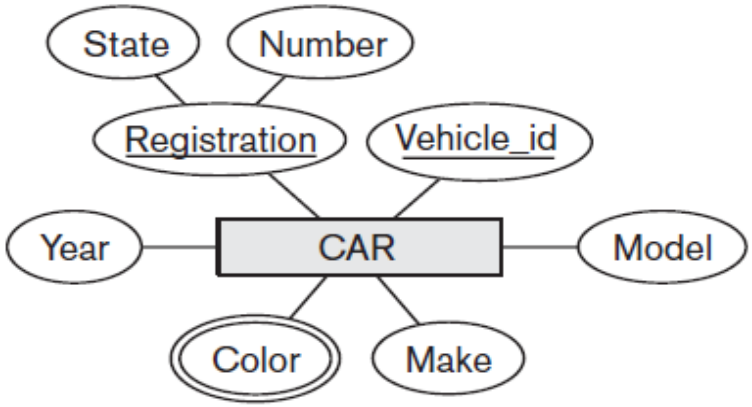
- An important constraint on the entities of a entity type is the key or uniqueness constraint on attributes.
- An entity type usually has an attribute whose values are distinct for each individual entity in the entity set, such attribute is called a **key attribute**.
- The values of key attribute can 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_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})

⋮

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).

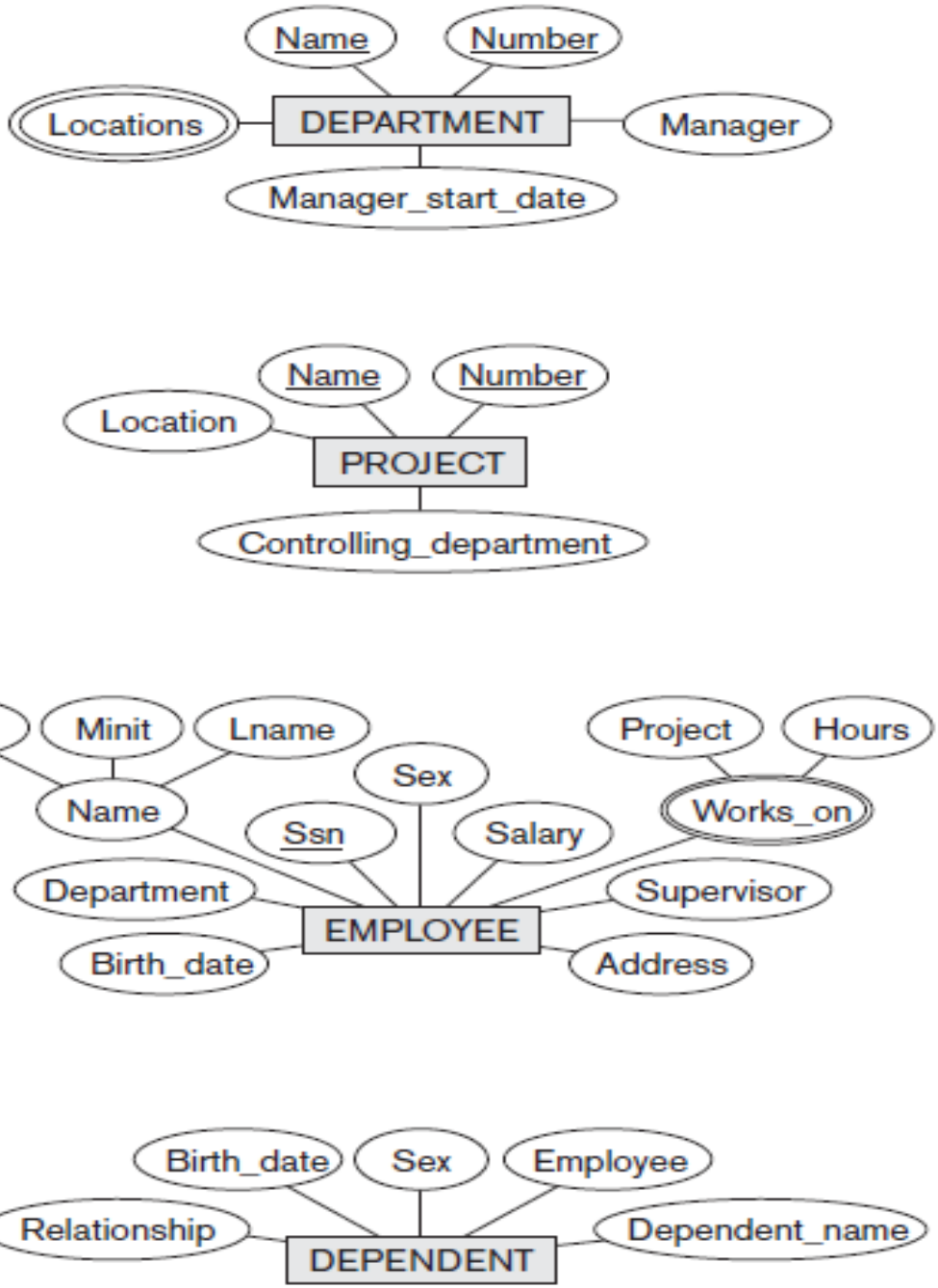


Figure 7.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Relationship Types,

Relationship Sets,

Roles, and

Structural Constraints

RELATIONSHIP TYPES, SETS, AND INSTANCES

- A **relationship type** *R* among *n* entity types *E*1, *E*2, ..., *En* 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…..e_n)
- Each entity e_j in r_i is a member of entity set E_j, $1 <= j <= n$.
- Hence, a relationship set is a mathematical relation on E_1, E_2, ...,E_n
- Alternatively, it can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \ldots \times E_n$.
- Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R.
- similarly, each of the individual entities e_1, e_2, ...,e_n is said to participate in the relationship instance $r_i = (e_1, e_2, \ldots, e_n)$.

- 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

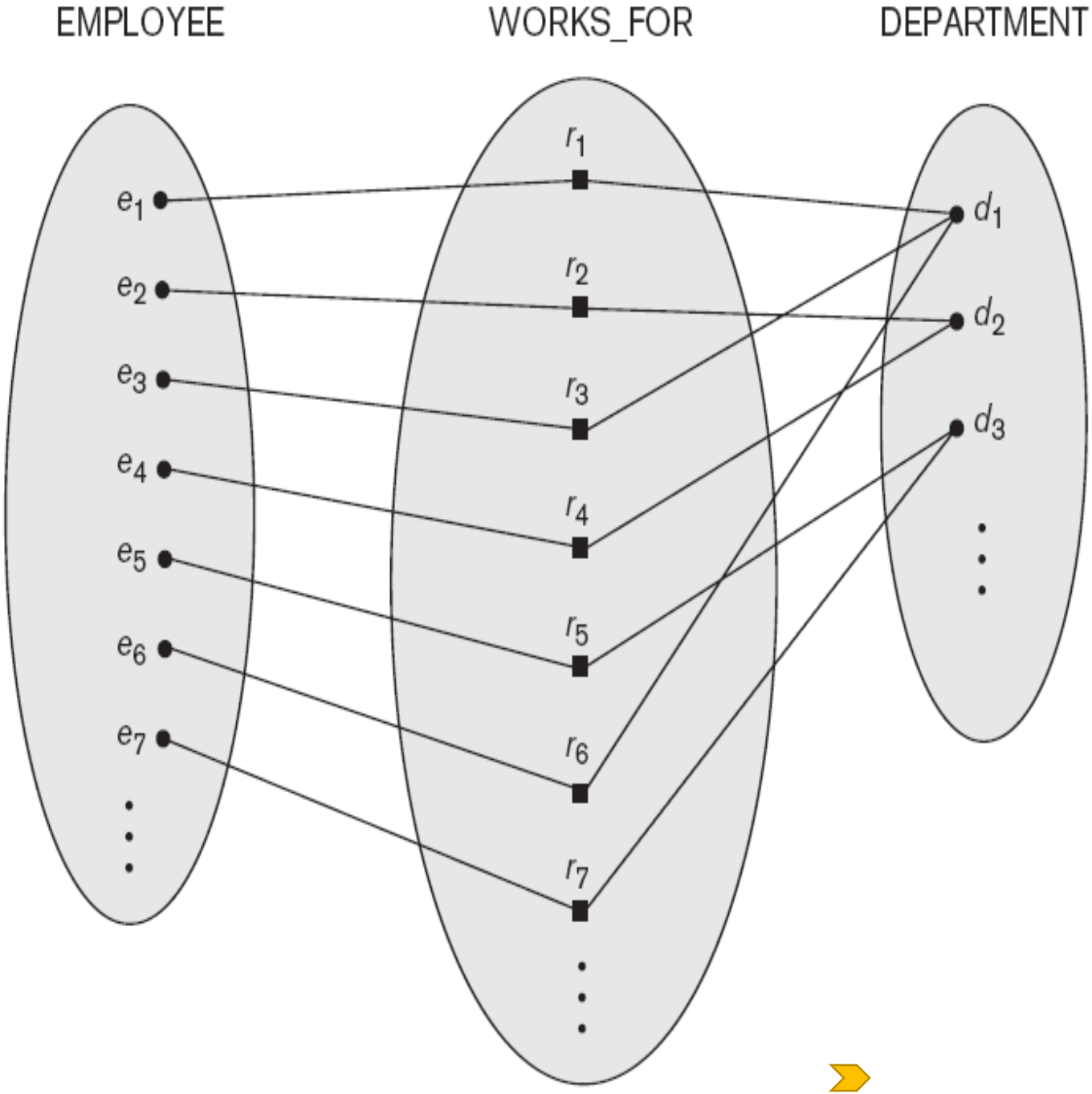


Figure 7.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship Degree

Role Names and

Recursive Relationships

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.

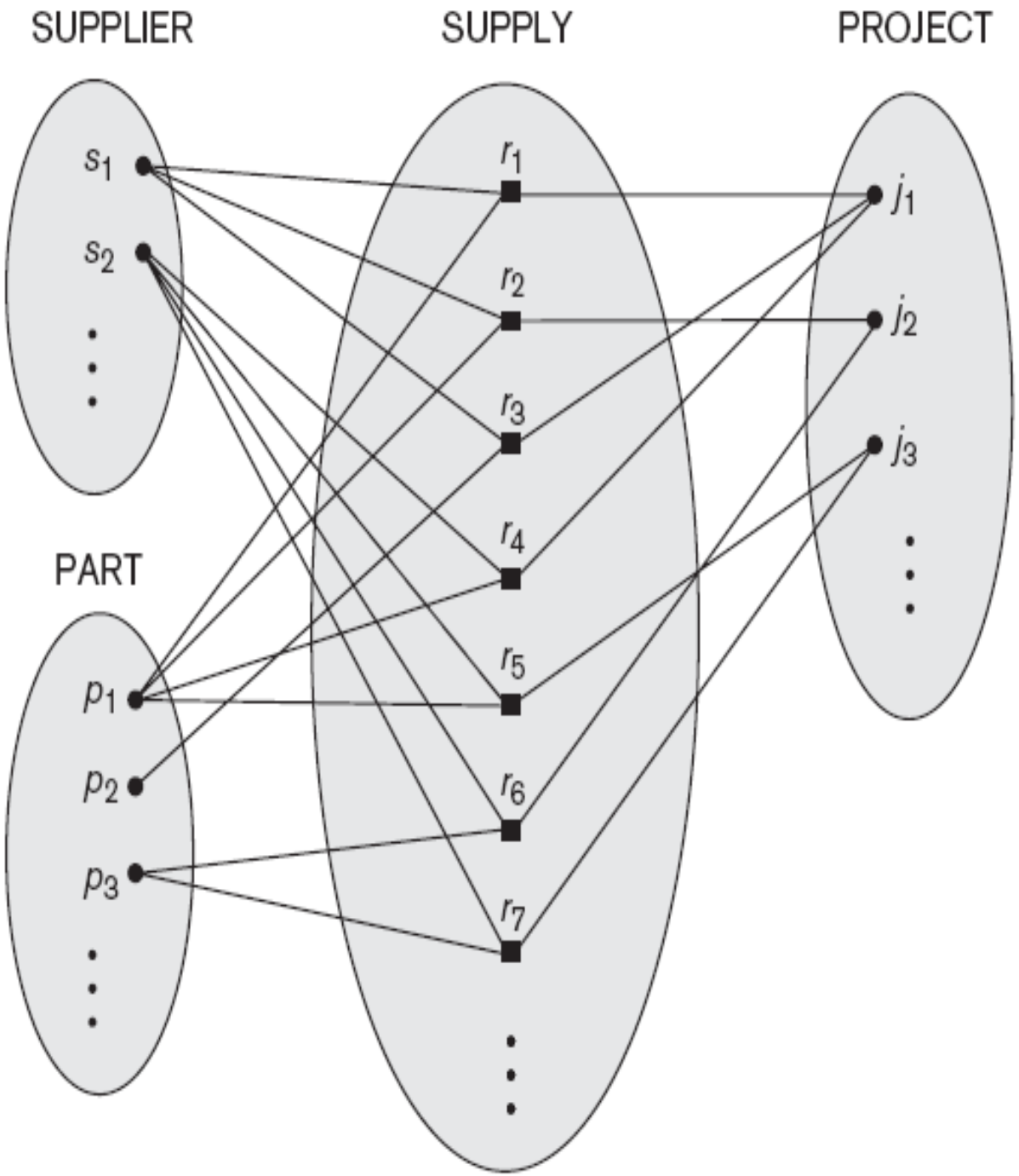


Figure 7.10
Some relationship instances in the SUPPLY ternary relationship set.

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)

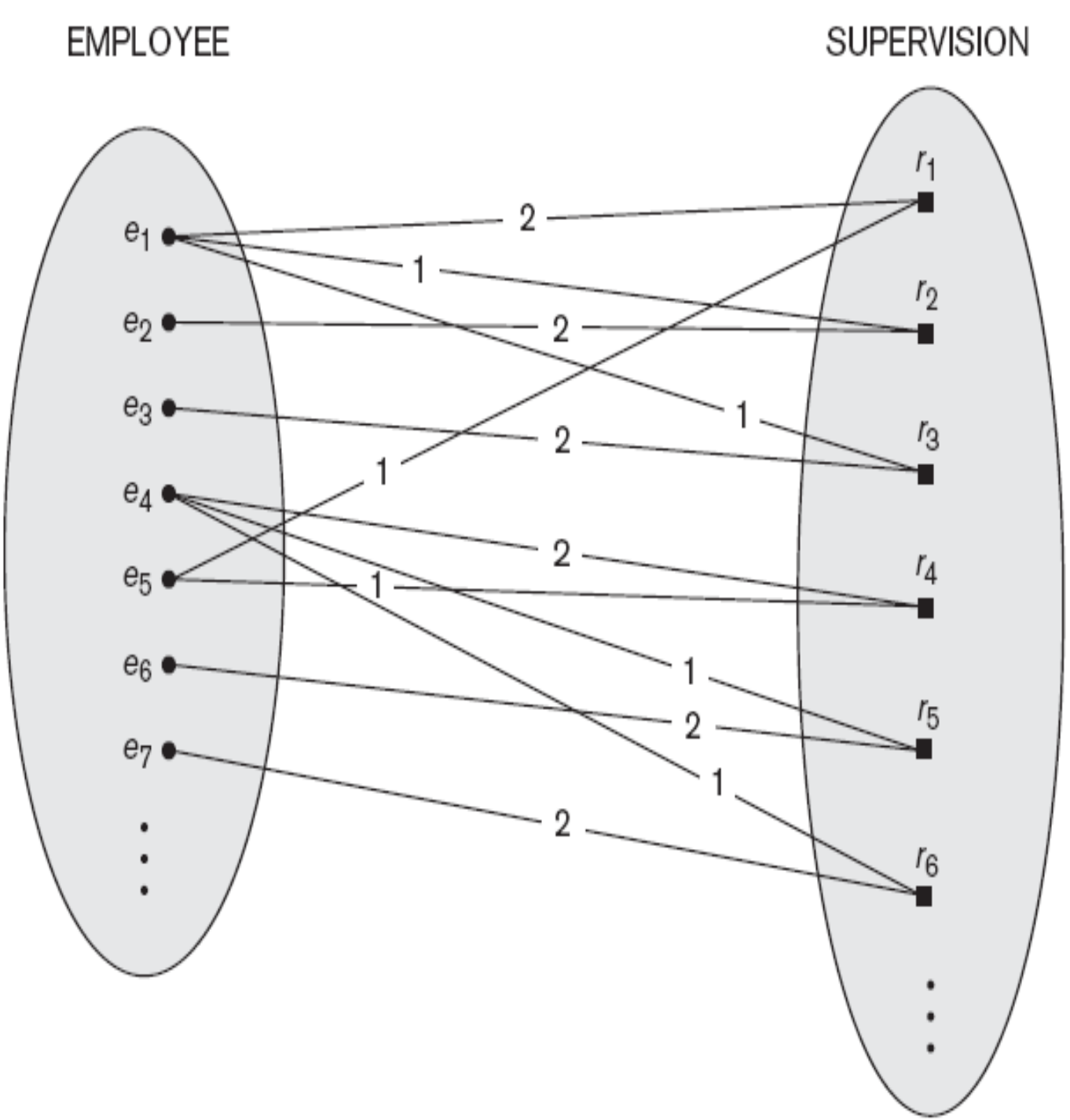


Figure 7.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

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*.

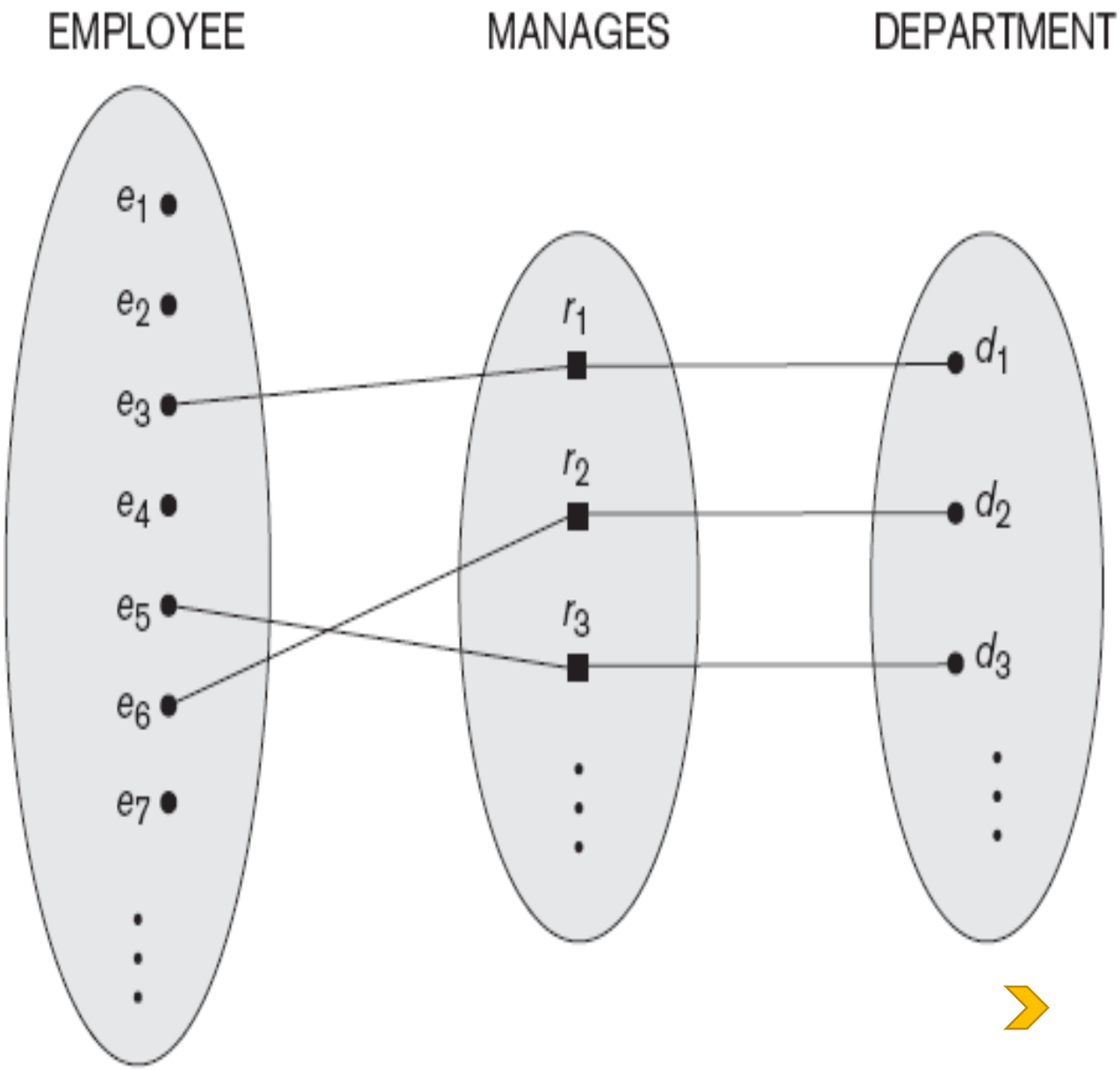
CARDINALITY RATIOS FOR BINARY RELATIONSHIPS

- 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

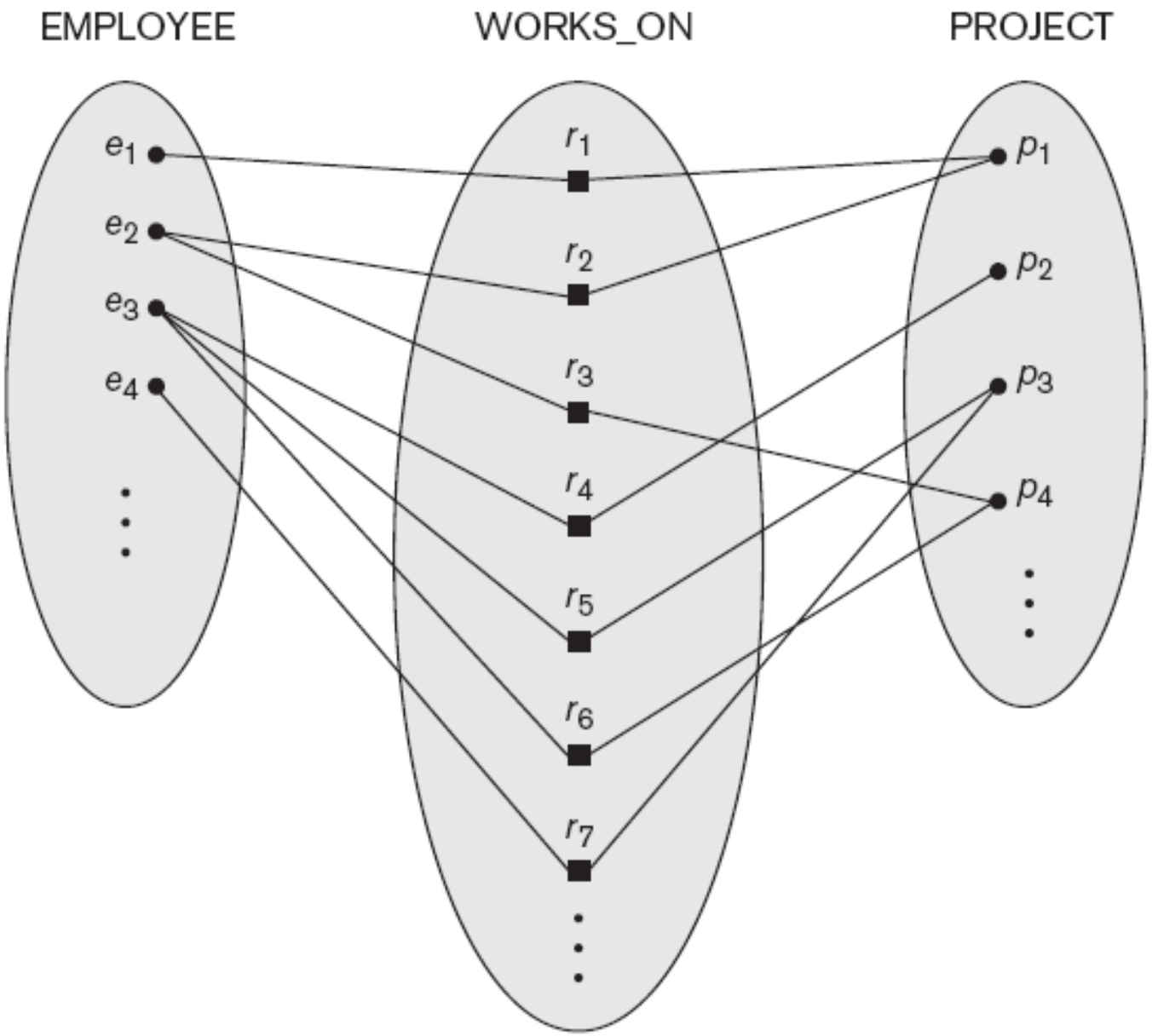


Figure 7.13
An M:N relationship, WORKS_ON.

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 birhtdate, 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

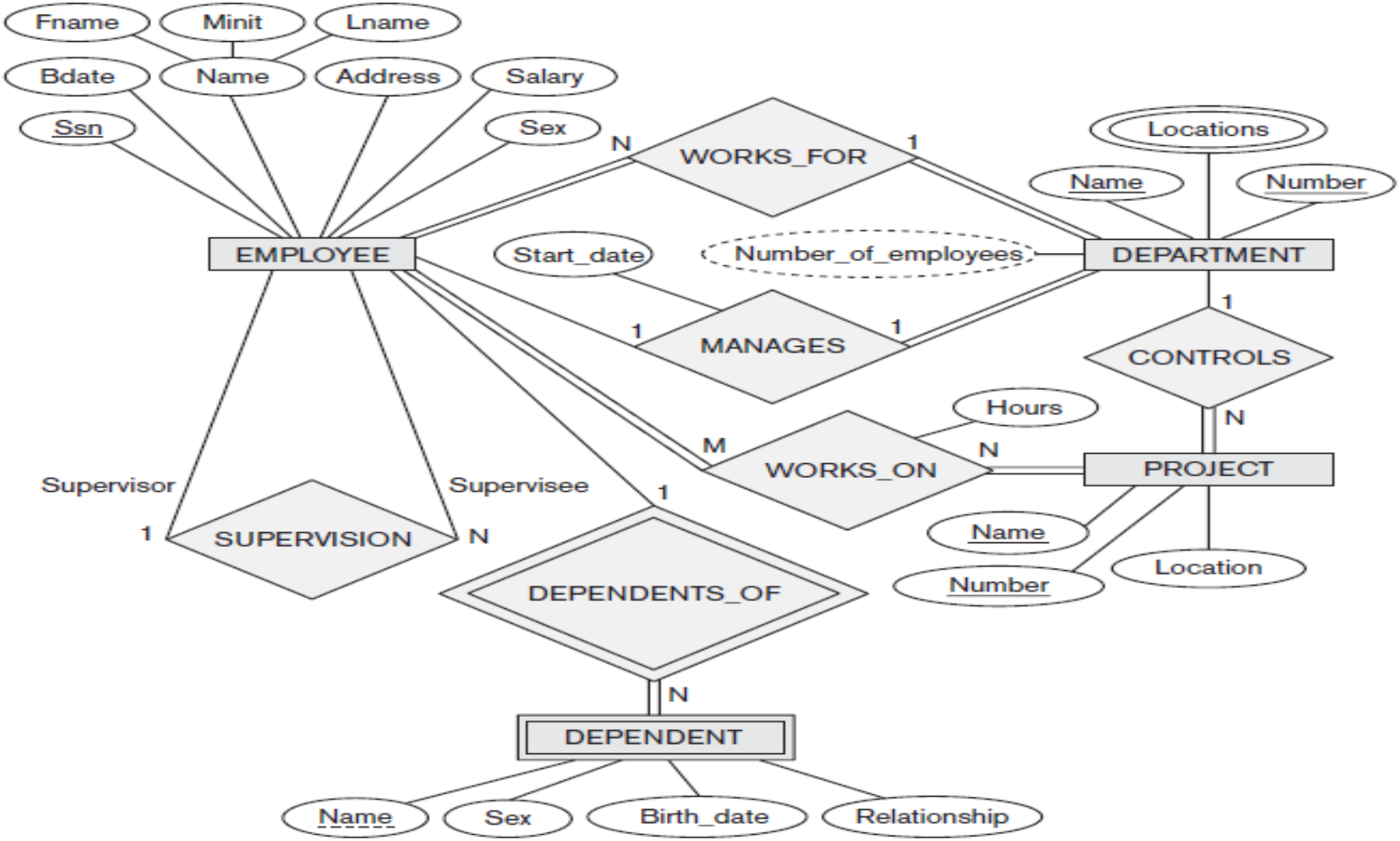


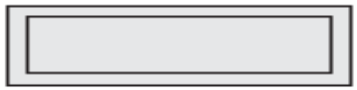
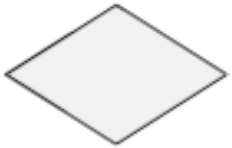
Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

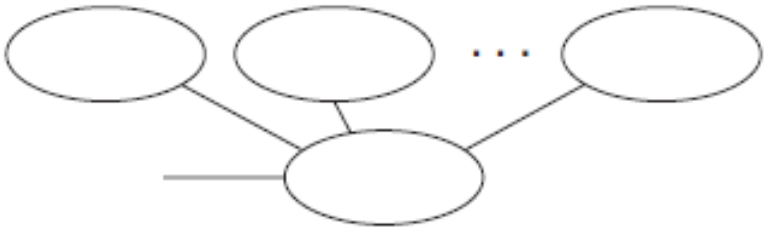
Refining the ER Design for the COMPANY Database

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.[13] 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

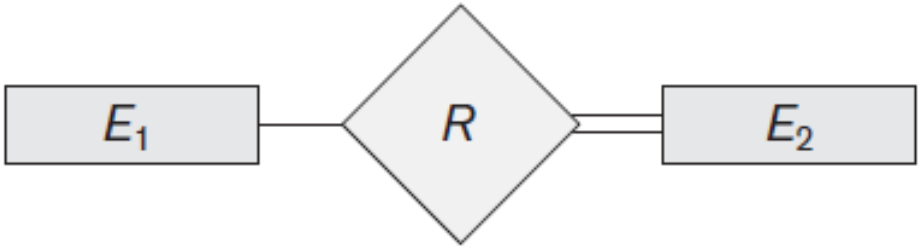
NOTATION FOR ER SCHEMAS

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

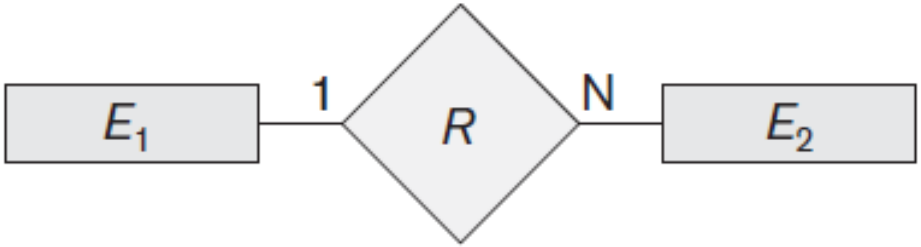


Composite Attribute

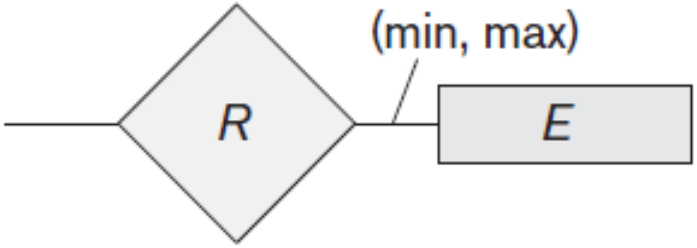
Derived Attribute



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 CONC

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

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

NAME	ROLL_NO	SEM	M
AJAY	10	3	
PRAKASH	30	5	

- In this table each row represents a fact about a particular student e
- The column names – Name, Roll_No, Sem, Major specify how to 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 indivisibl

  formal relational model is concerned.
- A common method for specifying a domain is to specify a d

  which the data values forming the domain are drawn.

- It is also helpful to specify a name for the domain, to help in inter
- 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 compa

   integer value between 20 and 60.

- A relation schema R, denoted by R(A1, A2,…, An) is made up of and a list of attributes A1, A2, … An.
- Each attribute Ai is the name of a role played by some domain schema R.
- Here D is called the domain of Ai and is denoted by dom(Ai).
- 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 relatio

- A relation of degree 7, which stores information about universit

  contain 7 attributes describing each student as,

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

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

  STUDENT (Name: String, Roll#: String, Phone#: Integer,

  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(A1,A2,..

  by r (R), is a set of N tuples r={t1,t2,..tn}.

- Each n-tuple t is an ordered list of n values, t={v1, v2, v3,..,Vn}.

- The i^{th} value of tuple t, which corresponds to the attribute Ai is ref

- The term relation intension is used for the schema R.

- Relation extension is used for relation sate 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 c

    dom(A2)…dom(An), which is the subset of the cartesian

    domains that defines R.

  - ✓ R subset (dom(A1), dom(A2)…dom(An))

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 relat

- A relation is defined as set of tuples.
- Mathematically, elements of a set have no order among them; h

  a relation do not have any particular order.
- A relation is not sensitive to the ordering of tuples.

Ordering of values within a tup

An alternative definition of a relation.

- According to the previous definition of a relation, an n-tuple is an o values, the ordering of values in a tuple and attributes in a rela important.
- However, at more abstract level, the order of attributes and their va important as long as the correspondence between attributes maintained

Values and NULL's in the Tup

- Each value in a tuple is atomic value. [composite and multivalu
  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
  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 th entity has a Name, Roll#, Mobile.. Etc attributes.
- Each tuple in the relation can then be interpreted as a **fact** or a instance of the assertion.
- For example, the first tuple in the STUDENT relation asserts th 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(A1,A2,..,An).
- ✓ 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 wh

  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 di
- ✓ An n-tuple t in a relation r(R) is denoted by t= <v1, v2, …,Vn>,

corresponding to value attribute Ai.

- ✓ Both t[Ai] and t.Ai refere to the value Vi in t for attribute Ai.
- ✓ **example:** consider the tuple t=<'Kevin Williamson','553889',8978

t[name]=<'Kevin Williamson'> and t[ssn,age]=<'553889', 40>

Relational Model Constraints & Rela
Database Schemas

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

✓ Constraints that are inherent in the data model, we call this as a

based constraint.

✓ Constraints that can be directly expressed in schemas of the data m

specifying them in DDL, we call this as **schema based constraints**.

✓ Constraints that cannot be directly expressed in the schemas of th

hence must be expressed and enforced by the application program

application based constraints.

✓ The characteristics of relations are inherent constraints.

✓ For example, Some attributes should not have duplicate valu

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 dependenci**

  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 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 t
- By definition all elements of a set are distinct; hence, all tuple 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 sam 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 constrai

$$\mathbf{t_1[SK] \neq t_2[SK]}$$

- Any such set of attributes SK is called as **Super Key** of the relatio
- A Super Key specifies the uniqueness constraints that no two dis 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 Ke**
- This is the candidate key whose values are used to identify tuples in
- We use the convention that the attributes that form the primary schema are underlined.
- If we want to have all the tuples should have values, we use NOT N

Relational Databases & Relational Da Schemas

- A relational database schema S is a set of relation schemas S= and set of Integrity Constraints IC.
- A relational database state DB of S is a set of relation states D such that each ri is a state of Ri And ri relation states sati constraints.

- For example, relational database schema of a COMPANY,

COMPANY={EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS

EMPLOYEE

Fname	M_Name	Lname	SSN	Bdate	Address	Salar

DEPARTMENT

Dname	Dno	Mgr_SSN	Mgr_Star

DEPT_LOCATIONS

Dno	Dloc

PROJECT

Pname	Pno	Ploc	Dno

WORKS_ON

ESSN	Pno	Hours

DEPENDENT

ESSN	Dept_Name	Gender	Bdate	Relat

- When we refer to relational database, we implicitly include both i

  current state.
- A database state that does not obey all the integrity constraints

  state.
- A state that satisfies all the constraints in the defined set of integ

  called a **valid state.**

Integrity, Referential Integrity & Foreig

- The entity integrity constraint states that no primary key value c
- This is because the primary key value is used to identify individ relation.
- Having NULL values for the primary key implies that we ca some tuples.
- Key constraints and entity integrity constraints are specified relations.

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

- A set of attributes FK in relation schema R1 is a foreign key of R relation R2 if it satisfies the following rule,
  1. The attribute in the FK have the same domain as the primary of R2; The attributes FK are said to reference or refer to the r
  2. A value of FK in tuple t1 of the current state r1(R1) either oc PK for some tuple t2 in the current state r2(R2) or is NULL.

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

- In this definition, R1 is called the **referencing relation** and R2 **referenced relation.**
- If these two condition hold, a **referential integrity constraint** fro said to hold.

Update Operations, Transactions, and I with Constraint Violations

- There are 3 basic update operations on relations; **insert**, **delete**
- They insert new data, delete old data and modify existing data t 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 attribu tuples.

The Insert Operation

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

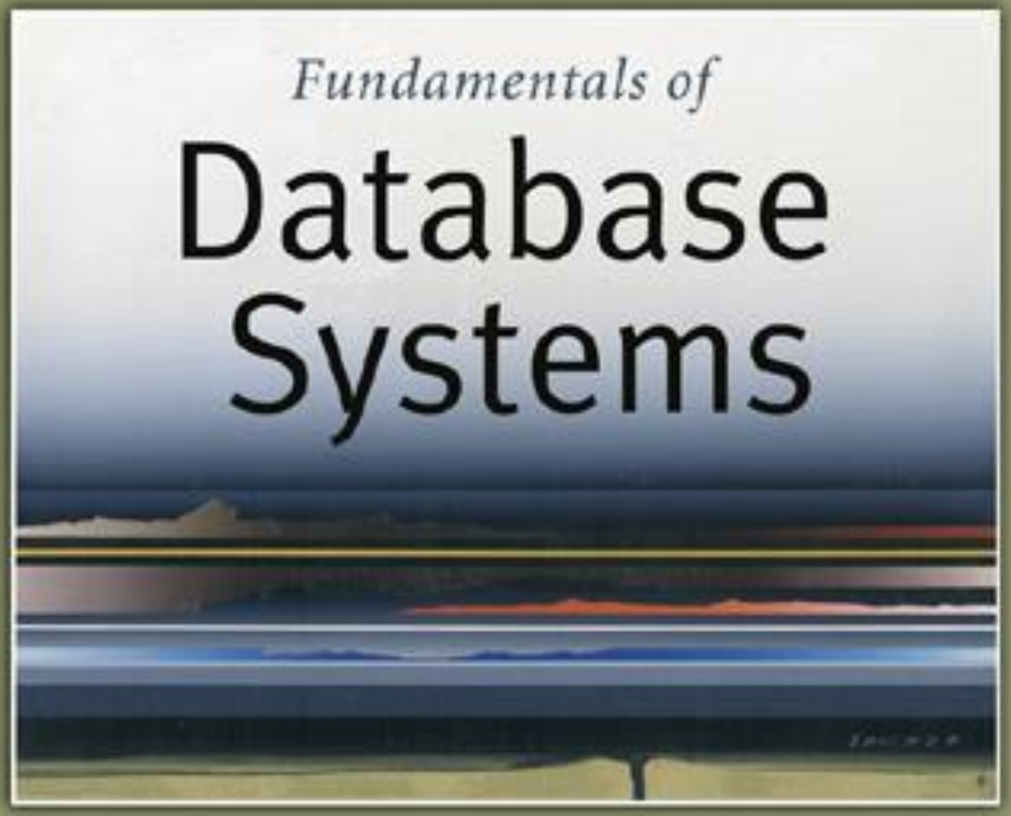
The Delete Operation

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

  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 rel tuples to be modified.
- Updating neither primary nor foreign key creates no problems



5th Edition

Elmasri / Navathe

Relational Algebra Overview

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: σ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, **–**)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
  - Additional Relational Operations
    - OUTER JOINS
    - INNER JOINS
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**.
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$$\sigma_{DNO = 4} (EMPLOYEE)$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{SALARY > 30,000} (EMPLOYEE)$$

Unary Relational Operations: SELECT

- In general, the *select* operation is denoted by

$$\sigma_{\text{<selection condition>}}(R) \text{ where}$$

  - the symbol σ (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition **true** are selected
    - appear in the result of the operation
  - tuples that make the condition **false** are filtered out
    - discarded from the result of the operation

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by π (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

Unary Relational Operations: PROJECT (cont.)

- The general form of the *project* operation is:

$$\pi_{<\text{attribute list}>}(R)$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.
- The project operation *removes any duplicate tuples*
  - This is because the result of the *project* operation must be a *set of tuples*
    - Mathematical sets *do not allow* duplicate elements.

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a single **relational algebra expression** by nesting the operations, or
  - We can apply one operation at a time and create **intermediate result relations**.
- In the latter case, we must give names to the relations that hold the intermediate results.

Single expression versus sequence of relational operations (Example)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a *single relational algebra expression* as follows:
  - $\pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
  - $DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$
  - $RESULT \leftarrow \pi_{FNAME, LNAME, SALARY}(DEP5\_EMPS)$

Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)

Unary Relational Operations: RENAME (contd.)

- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S\ (B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..Bn
  - $\rho_S(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, …..Bn

Unary Relational Operations: RENAME (contd.)

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
  - If we write:
    - RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)← $\pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
    - The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

Relational Algebra Operations from Set Theory: UNION

- UNION Operation
  - Binary operation, denoted by $\cup$
  - The result of R $\cup$ S, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be “type compatible” (or UNION compatible)
    - R and S must have same number of attributes
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

Relational Algebra Operations from Set Theory

- Type Compatibility of operands is required for the binary set operation UNION $\cup$, INTERSECTION $\cap$, and SET DIFFERENCE –
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n).
- The resulting relation for R1$\cup$R2 also for R1$\cap$R2, or R1–R2 has the same attribute names as the *first* operand relation R1

Relational Algebra Operations from Set Theory: INTERSECTION

- INTERSECTION is denoted by ∩
- The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R

Relational Algebra Operations from Set Theory: SET DIFFERENCE (cont.)

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R

Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT − INSTRUCTOR. (e) INSTRUCTOR − STUDENT.

Some properties of UNION, INTERSECT, and DIFFERENCE

- Notice that both union and intersection are *commutative* operations; that is
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can supports *associative* operations; that is
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- The minus operation is not commutative and not *associative*; that is, in general
  - $R - S \neq S - R$

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples.
  - The two operands do NOT have to be "type compatible"

Cartesian Product: *instructor X teaches*

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Binary Relational Operations: JOIN

- Joins can be simply defined as the combining or merging the related tuples from the two different relations into a single type.
- The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
- The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$R \bowtie_{\text{<join condition>}} S$$

- where R and S can be any relations that result from general *relational algebra expressions*.

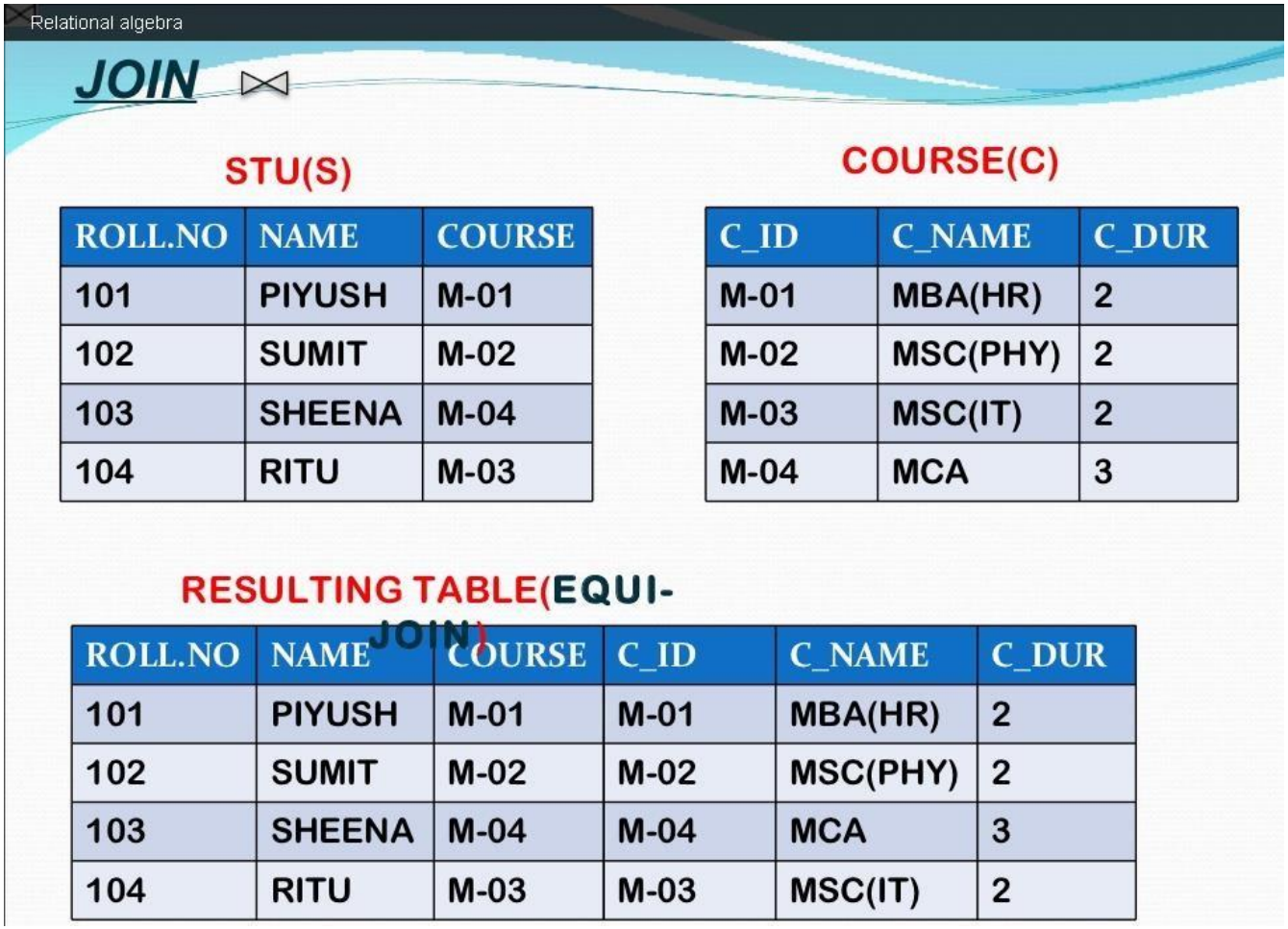
We can classify joins basically into two types

- **INNER JOINS**: These joins are the one that has the tuples that satisfy some conditions and rest are discarded. Further they are classified as
  - Theta join
  - Equi join
  - Natural join
- **OUTER JOINS**: These have all the tuples from either or both the relations. Further they are classified as
  - Left outer join
  - Right outer join
  - Full outer join

Binary Relational Operations: EQUIJOIN

- **EQUIJOIN Operation**
- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN.
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
  - The JOIN seen in the previous example was an EQUIJOIN.

EQUIJOIN



STU(S)

ROLL.NO	NAME	COURSE
101	PIYUSH	M-01
102	SUMIT	M-02
103	SHEENA	M-04
104	RITU	M-03

COURSE(C)

C_ID	C_NAME	C_DUR
M-01	MBA(HR)	2
M-02	MSC(PHY)	2
M-03	MSC(IT)	2
M-04	MCA	3

RESULTING TABLE(EQUI-JOIN)

ROLL.NO	NAME	COURSE	C_ID	C_NAME	C_DUR
101	PIYUSH	M-01	M-01	MBA(HR)	2
102	SUMIT	M-02	M-02	MSC(PHY)	2
103	SHEENA	M-04	M-04	MCA	3
104	RITU	M-03	M-03	MSC(IT)	2

Binary Relational Operations: NATURAL JOIN Operation

- **NATURAL JOIN Operation**
  - Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
    - because one of each pair of attributes with identical values is superfluous
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
  - If this is not the case, a renaming operation is applied first.

Additional Relational Operations (cont.)

- The left outer join operation keeps every tuple in the first or left relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values.
- A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R ⟖ S.
- A third operation, full outer join, denoted by ⟗ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Recap of Relational Algebra Operations

Table 6.1
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
  - These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include
  - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Aggregate Function Operation

- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, without removing duplicates