Entity-Relationship (ER) Model

Data Modeling Using the Entity-Relationship Model: ER Model Concepts, Notation for ER Diagram, Mapping Constraints, Keys, Specialization, Generalization, Aggregation, Reduction of an ER Diagram to Tables, Extended ER Model.

Overview of Database Design Process

- Two main activities:
  - Database design
  - Applications design

- Focus in this chapter on conceptual database design
  - To design the conceptual schema for a database application

- Applications design focuses on the programs and interfaces that access the database
  - Generally considered part of software engineering

Overview of Database Design Process

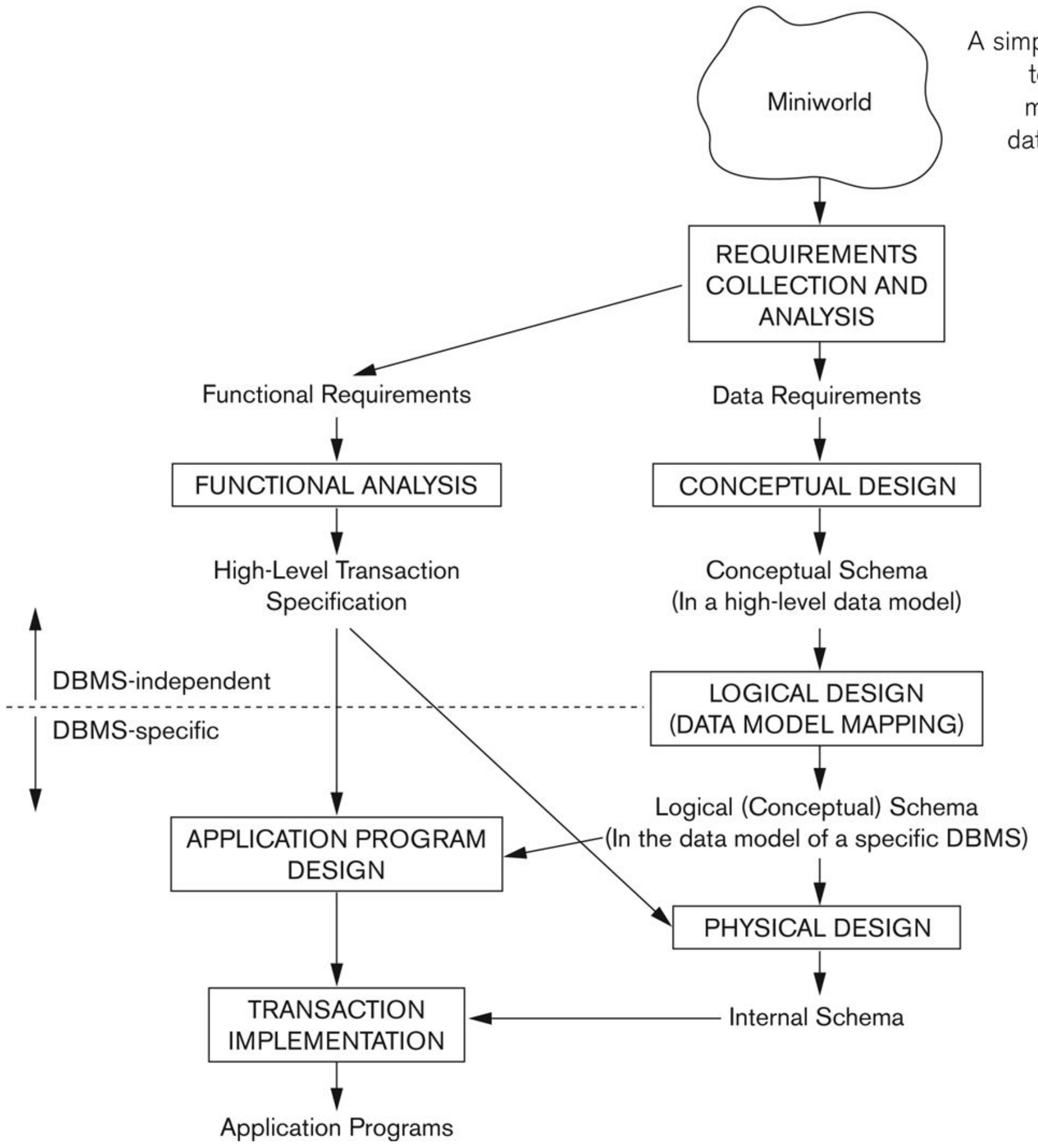


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Example COMPANY Database

- We need to create a database schema design based on the following **requirements** of the COMPANY Database:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
  - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

Example COMPANY Database

- The database will store each EMPLOYEE’s social security number, address, salary, gender, and birthdate.
  - Each employee *works for* one department but may *work on* several projects, which are not necessarily controlled by the same department.
  - The DB will keep track of the number of hours per week that an employee currently works on each project.
  - It is required to keep track of the *direct supervisor* of each employee (who is another employee).
- Each employee may *have* a number of DEPENDENTs (for insurance purposes)
  - For each dependent, the DB keeps a record of name, gender, birthdate, and relationship to the employee.

Database Design

- **Requirements collection and analysis**
  - Database designers interview database users to understand and document their data requirements.
  - Result of this step is a concisely written set of users’ requirements.
- Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**.
- Conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model.

Database Design

- The next step in database design is the actual implementation of the database, using a commercial DBMS.
- Most current commercial DBMSs use an implementation data model such as the relational or the object-relational database model.
- Conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping**.
- Its result is a database schema in the implementation data model of the DBMS.
- The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.

Database Design

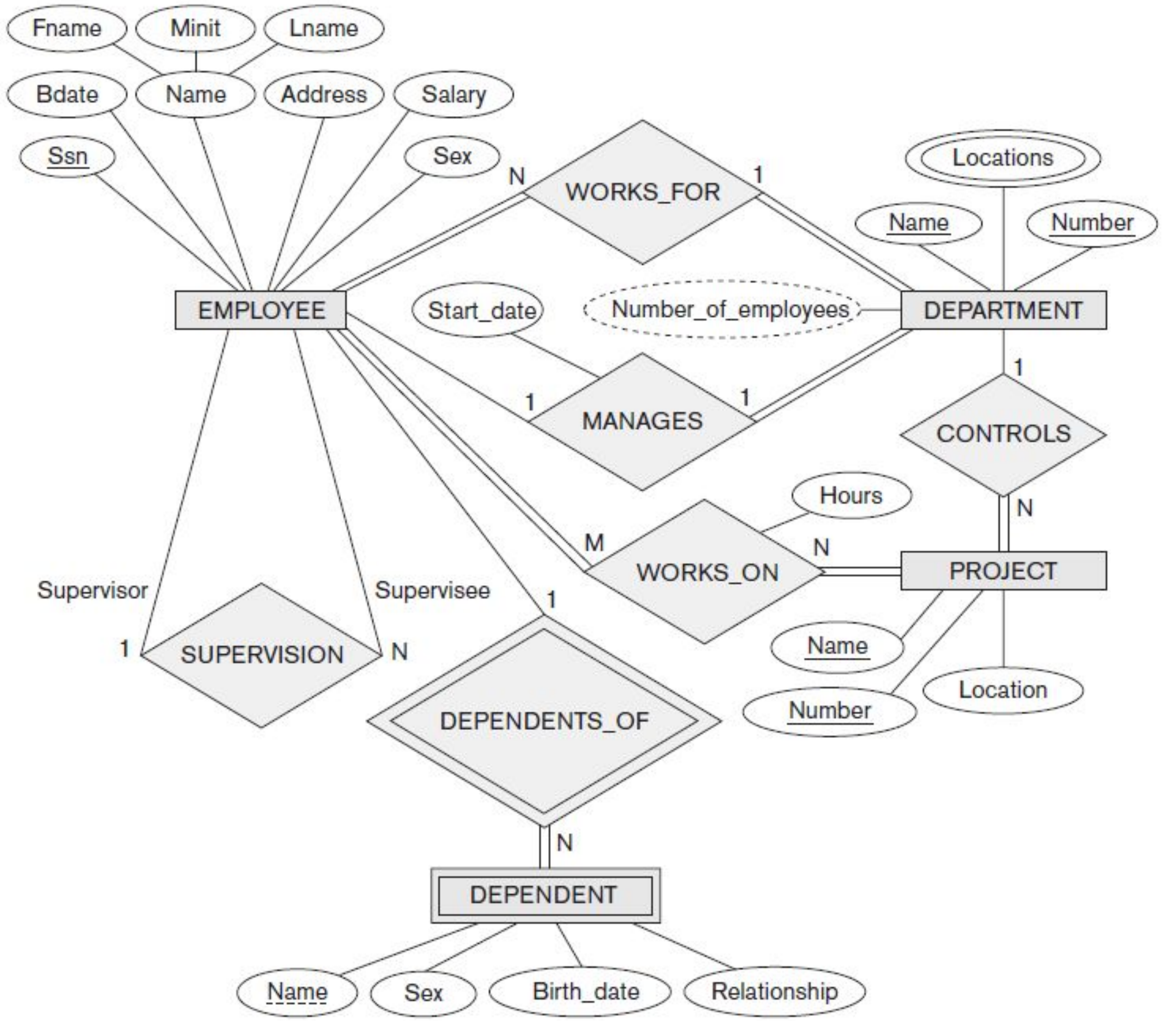


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

Methodologies for Conceptual Design

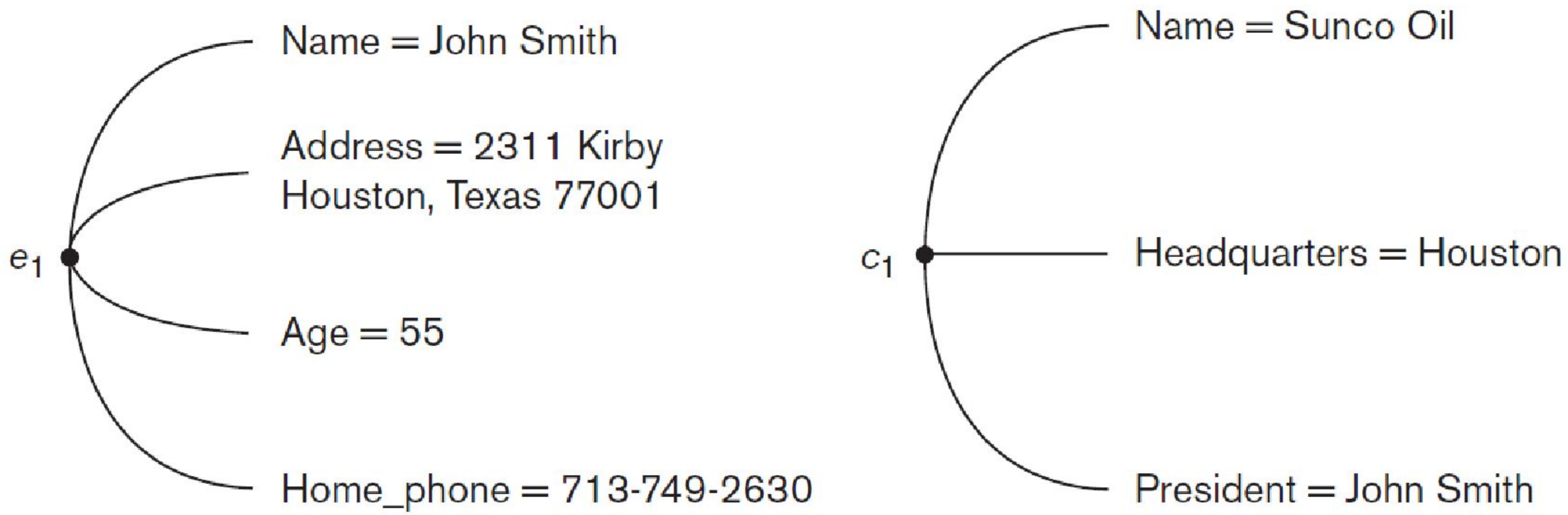
- Entity Relationship (ER) Diagrams (This Chapter)
- Enhanced Entity Relationship (EER) Diagrams (Chapter 4)
- Use of Design Tools in industry for designing and documenting large scale designs
- The UML (Unified Modeling Language) Class Diagrams are popular in industry to document conceptual database designs

ER Model Concepts

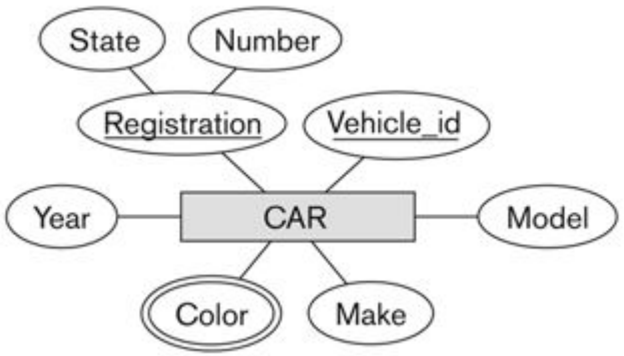
- **Entities and Attributes**
  - The basic object that the ER model represents is an **entity**, which is a thing in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).
    - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
- Each entity has **attributes**. Attributes are properties used to describe an entity.
  - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Gender, BirthDate
- A specific entity will have a value for each of its attributes.
  - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX’, Gender='M', BirthDate='09-JAN-55‘
- Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, date, enumerated type, …

Entities and Attributes

- Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes



Types of Attributes



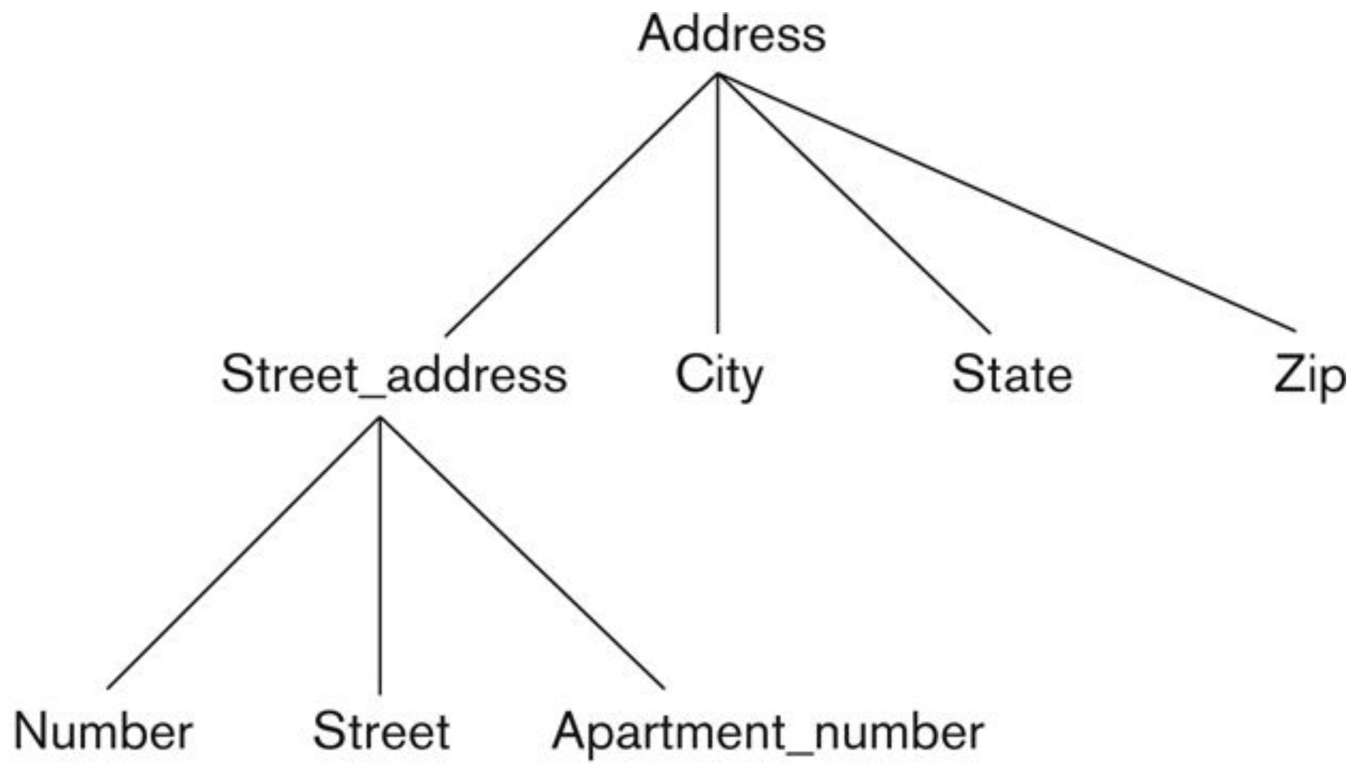
❑ Simple

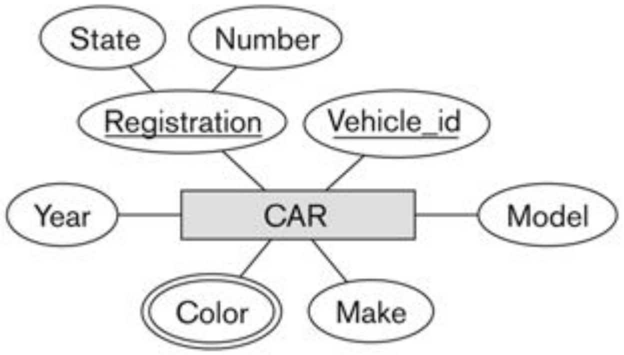
- Attributes that are not divisible are called simple or atomic attributes. Each entity has a single atomic value for the attribute. For example, SSN or Gender.

❑ Composite

- The attribute may be composed of several components. Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example:
  - Address (Street_address, City, State, Zip)
  - Name (FirstName, MiddleName, LastName)
- Composition may form a hierarchy where some components are themselves composite.

A hierarchy of composite attributes





Types of Attributes

❑ Single-Valued

- Attributes that have a single value for a particular entity. For example, Age is a single-valued attribute of a person.

❑ Multi-valued

- An attribute can have a set of values or multiple values for the same entity.
- For example, Phone_number attribute for an employee, or a Degree attribute for a person.
- Denoted as {Phone_number} or {Degrees}

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare

Types of Attributes

❑ Stored

- Attribute which already stored in database. For example, Birth_date attribute of a person.

❑ Derived

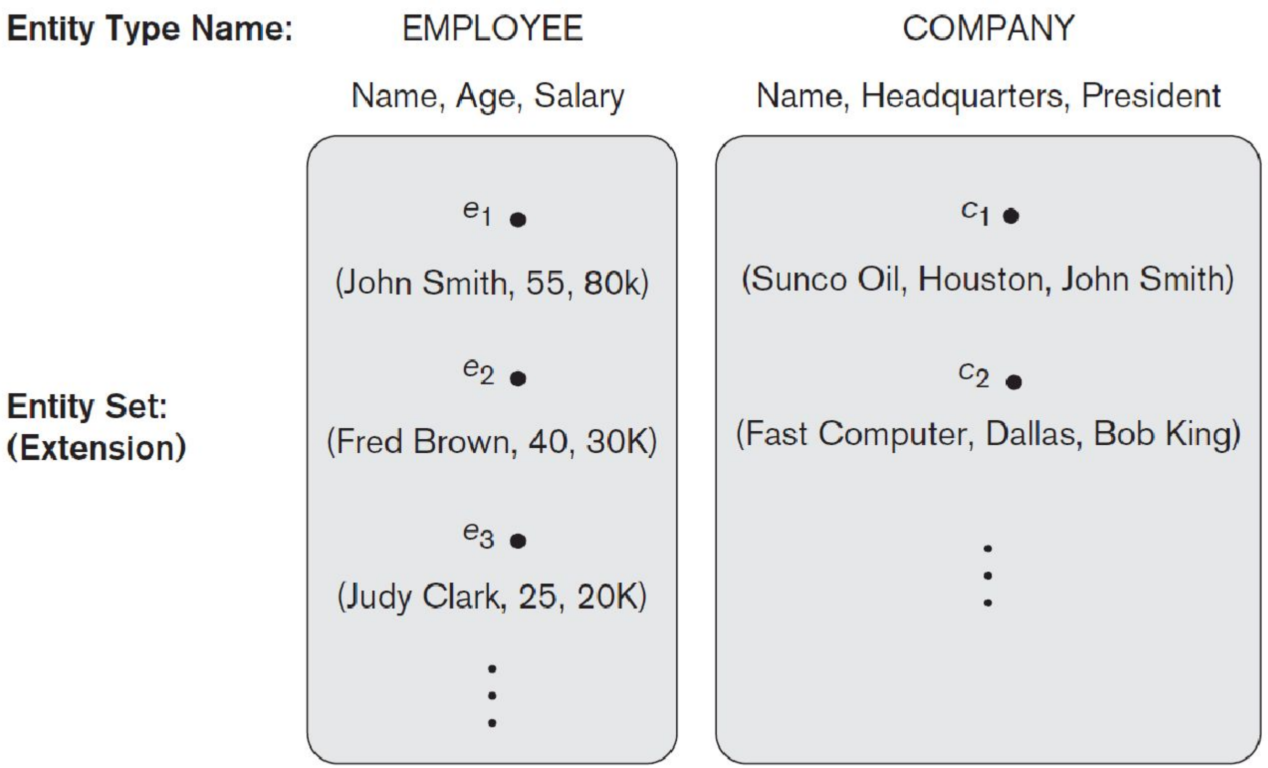
- Derivable from stored attribute. For example, Age attribute of a person. Age attribute is derivable from the Birth_date attribute, which is a stored attribute.

❑ Complex

- In general, composite and multivalued attributes can 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)}
  - Multiple PreviousDegrees values can exist
  - Each has four subcomponent attributes: College, Year, Degree, Field

Entity Types and Entity Sets

- An **entity type** defines a collection (or set) of entities that have the same attributes. Entities with the same basic attributes are grouped or typed into an entity type.
- Each entity type in the database is described by its name and attributes.

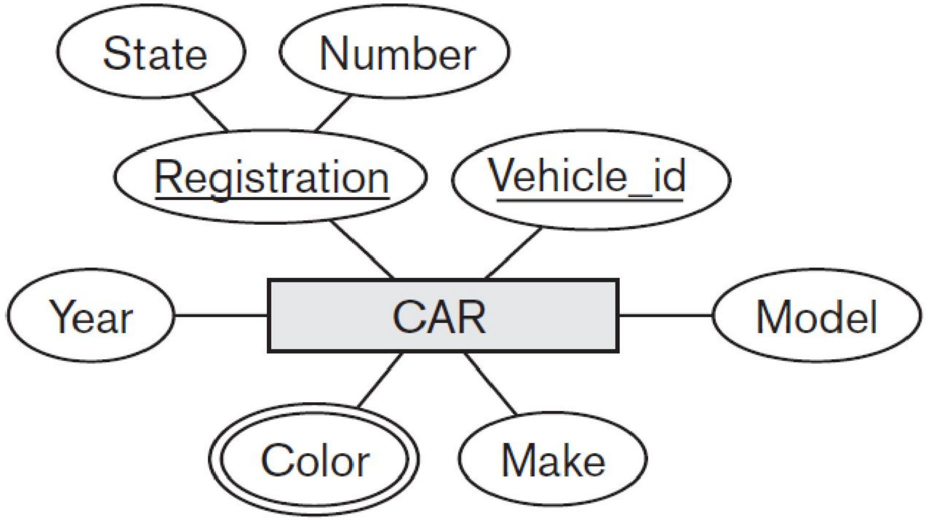


Entity Types and Entity Sets

- Each entity type will have a collection of entities stored in the database.
- The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**. Entity set is the current *state* of the entities of that type that are stored in the database.
- 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 is represented in **ER diagram** as a rectangular box enclosing the entity type name.
- Attribute names are enclosed in ovals and are attached to their entity type by straight lines.
- Composite attributes are attached to their component attributes by straight lines.
- Multivalued attributes are displayed in double ovals.
- An entity type describes the schema or intension for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the extension of the entity type.

Example: CAR Entity Type

- ER diagram notation:



- Entity set with three entities:

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})

⋮

NOTATION for ER diagrams

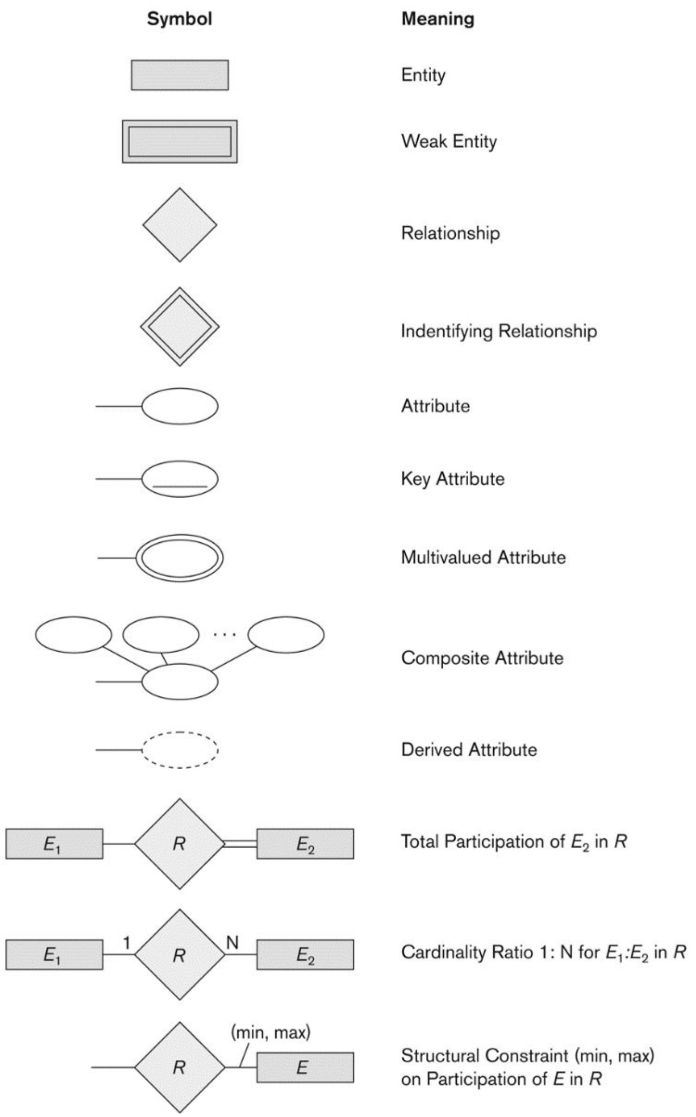


Figure 3.14
Summary of the notation for ER diagrams.

Key Attributes of an Entity Type

- An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
- For example
  - Name attribute is a key of the COMPANY entity type because no two companies are allowed to have the same name.
  - SSN (Social Security number) of EMPLOYEE
- A key attribute may be composite.
  - VehicleRegistrationNumber is a key of the CAR entity type with components (Number, State).

Key Attributes of an Entity Type

- An entity type may have more than one key. For example, each of the Vehicle_id and Registration attributes of the entity type CAR is a key.
- An entity type may also have *no key*, in which case it is called a *weak entity type*.
- In ER diagrammatic notation, each key attribute has its name underlined inside the oval.

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.
  - E.g., Lastname has a value which is a character string of upto 15 characters, say
  - Date has a value consisting of MM-DD-YYYY where each letter is an integer
- A value set specifies the set of values associated with an attribute.
- Value sets are similar to **data types** in most programming languages – e.g., integer, character, Boolean, float.
- Mathematically, an attribute A of entity set E whose value set is V is defined as a function:

$$A : E \rightarrow P(V)$$

  where P(V) indicates a power set (all possible subsets) of V.
- We refer to the value of attribute A for entity e as A(e).

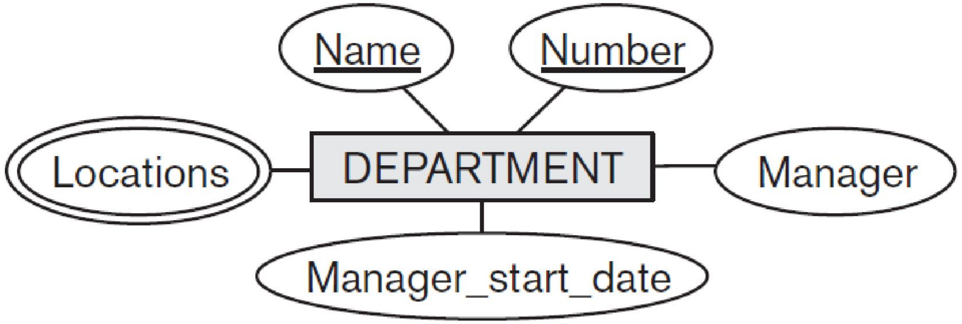
Initial Conceptual Design of Entity Types for the COMPANY Database Schema

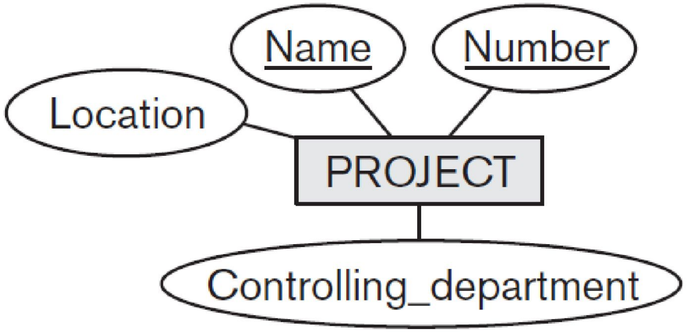
- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT

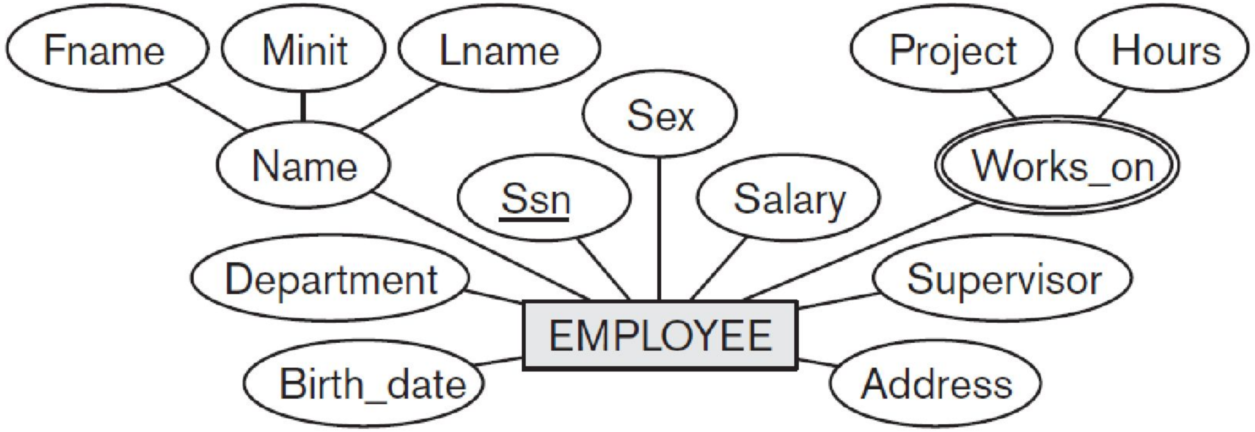
- Their initial conceptual design is shown on the following slide

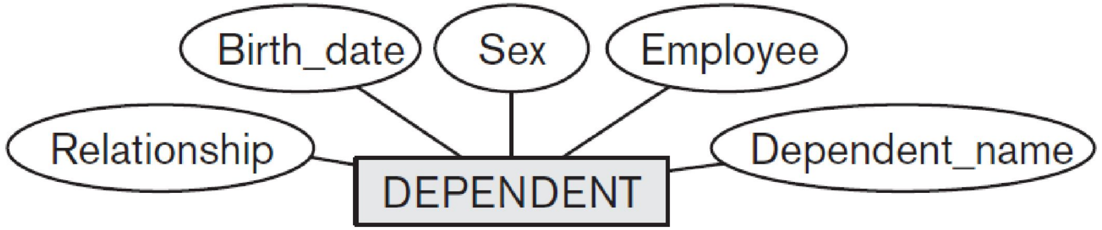
- The initial attributes shown are derived from the requirements description

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









Refining the initial design by introducing relationships

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

Relationships

- There are several *implicit relationships* among the various entity types.
- Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example,
  - Attribute Manager of DEPARTMENT refers to an employee who manages the department.
  - Attribute Controlling_department of PROJECT refers to the department that controls the project.
  - Attribute Supervisor of EMPLOYEE refers to another employee.
  - Attribute Department of EMPLOYEE refers to the department for which the employee works.
- In the ER model, these references should not be represented as attributes but as **relationships.**
- In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

Relationship Types, Sets, and Instances

- A relationship relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith works on the ProductX PROJECT, or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- A **relationship type** *R* among n entity types $E_1, E_2, \ldots, E_n$ defines a set of associations—or a **relationship set**—among entities from these entity types.

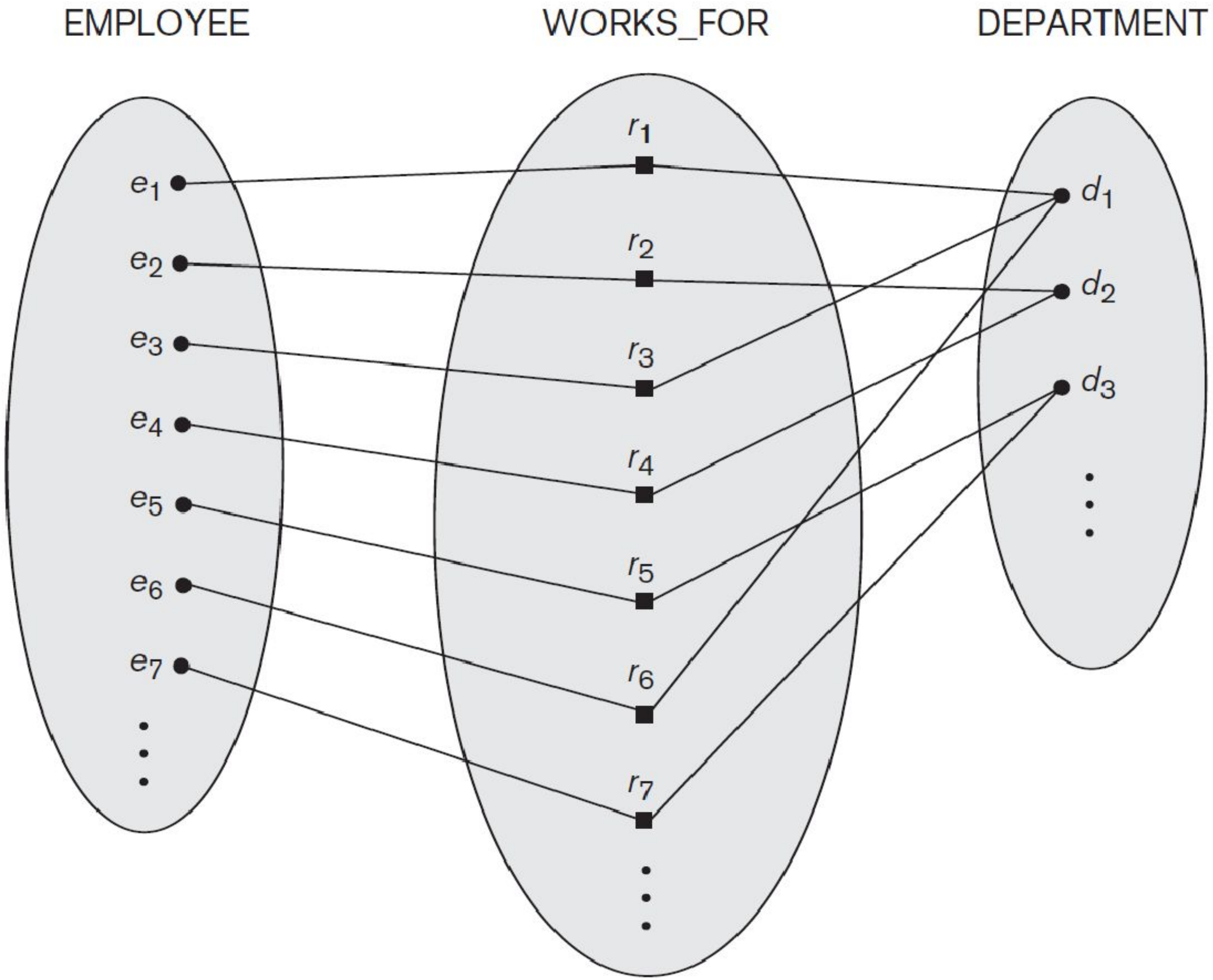
Relationship Types, Sets, and Instances

- 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), and each entity e_j in r_i is a member of entity set E_j, $1 \leq j \leq n$.
- A relationship set is a mathematical relation on $E_1, E_2, \ldots, E_n$
- 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 $E_1, E_2, \ldots, E_n$ is said to participate in the relationship type R.
- Each of the individual entities $e_1, e_2, \ldots, e_n$ is said to participate in the relationship instance $r_i = (e_1, e_2, \ldots, e_n)$

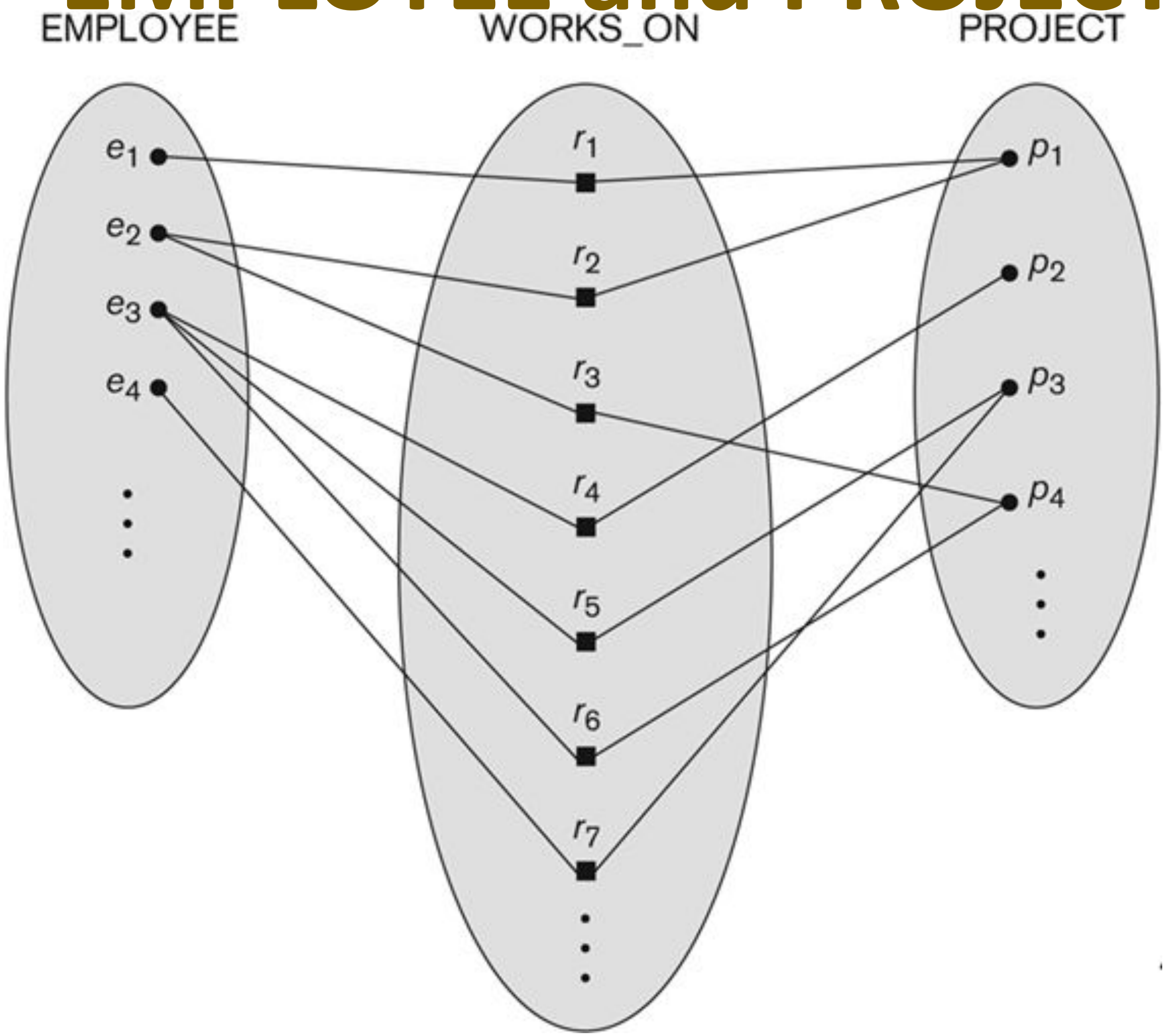
Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

- Consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set.
- Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

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



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



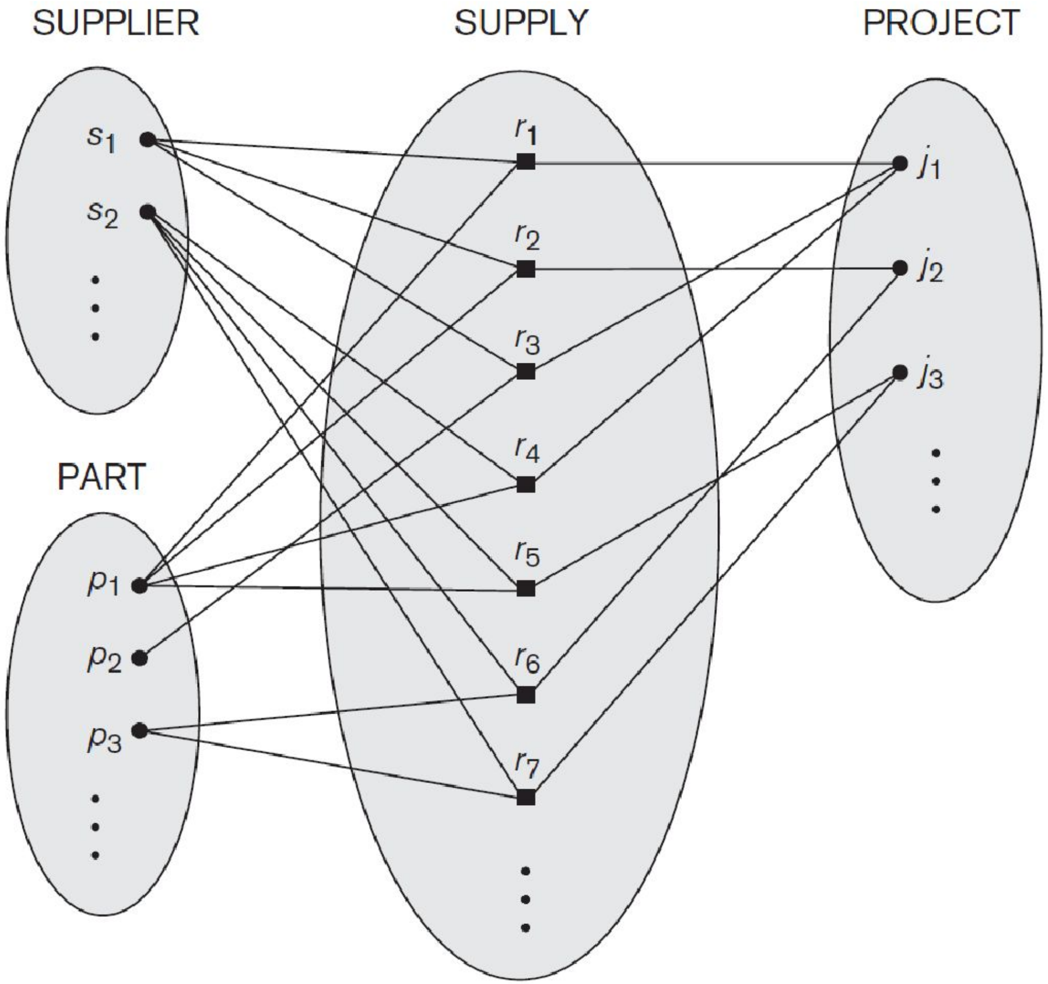
Relationship type vs. relationship set

- Relationship Type:
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints
- Relationship Set:
  - The current set of relationship instances represented in the database
  - The current state of a relationship type
- Each instance in the relationship set relates individual participating entities – one from each participating entity type.
- In ER diagrams, we represent the relationship type as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines

Degree of a Relationship Type

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

Example of ternary relationship

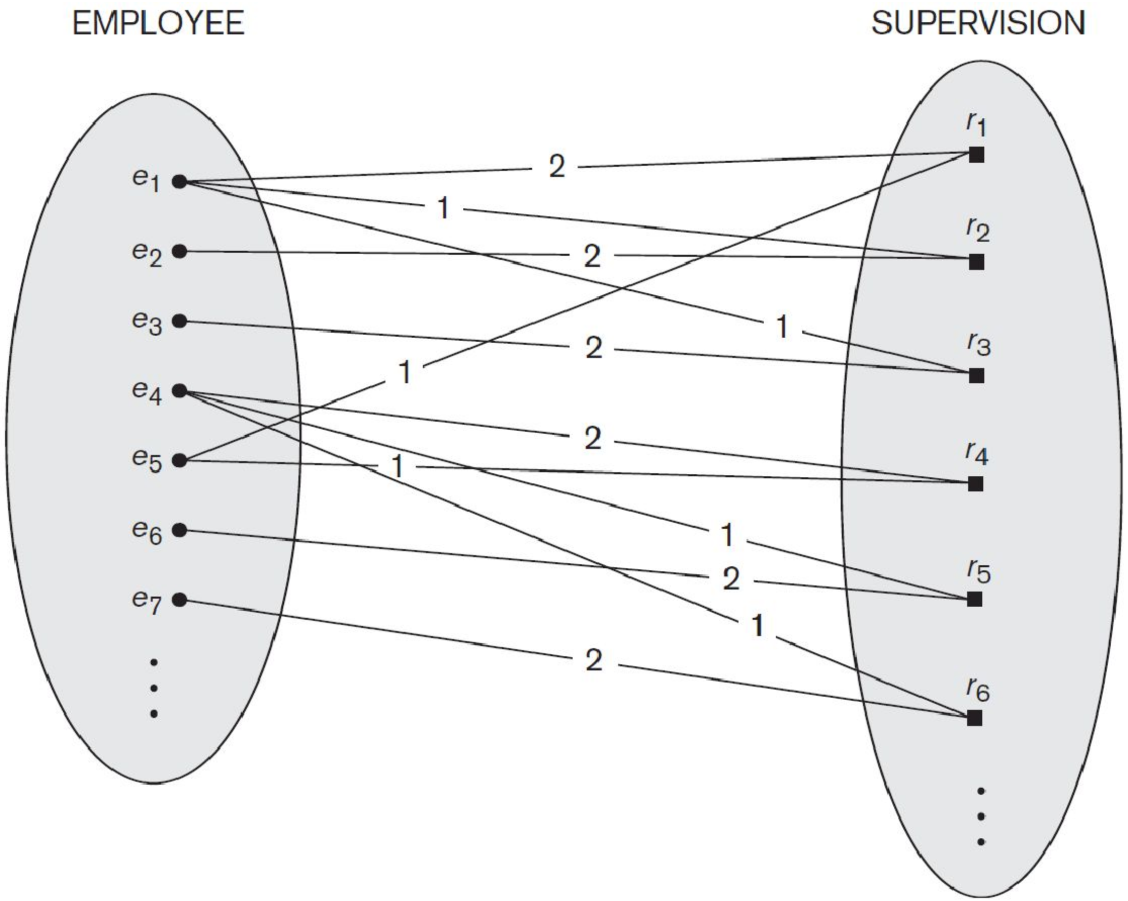


- Each relationship instance r_i associates three entities—a supplier s, a part p, and a project j—whenever s supplies part p to project j.

Recursive Relationship Type

- In some cases the *same* entity type participates more than once in a relationship type in *different roles*.
- A relationship type between the same participating entity type in **distinct roles.**
- Also called **self-referencing** relationship type.
- Example: SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set.
- EMPLOYEE participates twice in two distinct roles:
  - supervisor role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in *supervisor* role
  - One employee in *supervisee* role
- In ER diagram, need to display role names to distinguish participations.

A Recursive Relationship Supervision



- Lines marked '1' represent the supervisor role, and those marked '2' represent the supervisee role.
- e_1 supervises e_2 and e_3, e_4 supervises e_6 and e_7, and e_5 supervises e_1 and e_4
- Each relationship instance must be connected with two lines, one marked with '1' (supervisor) and the other with '2' (supervisee).

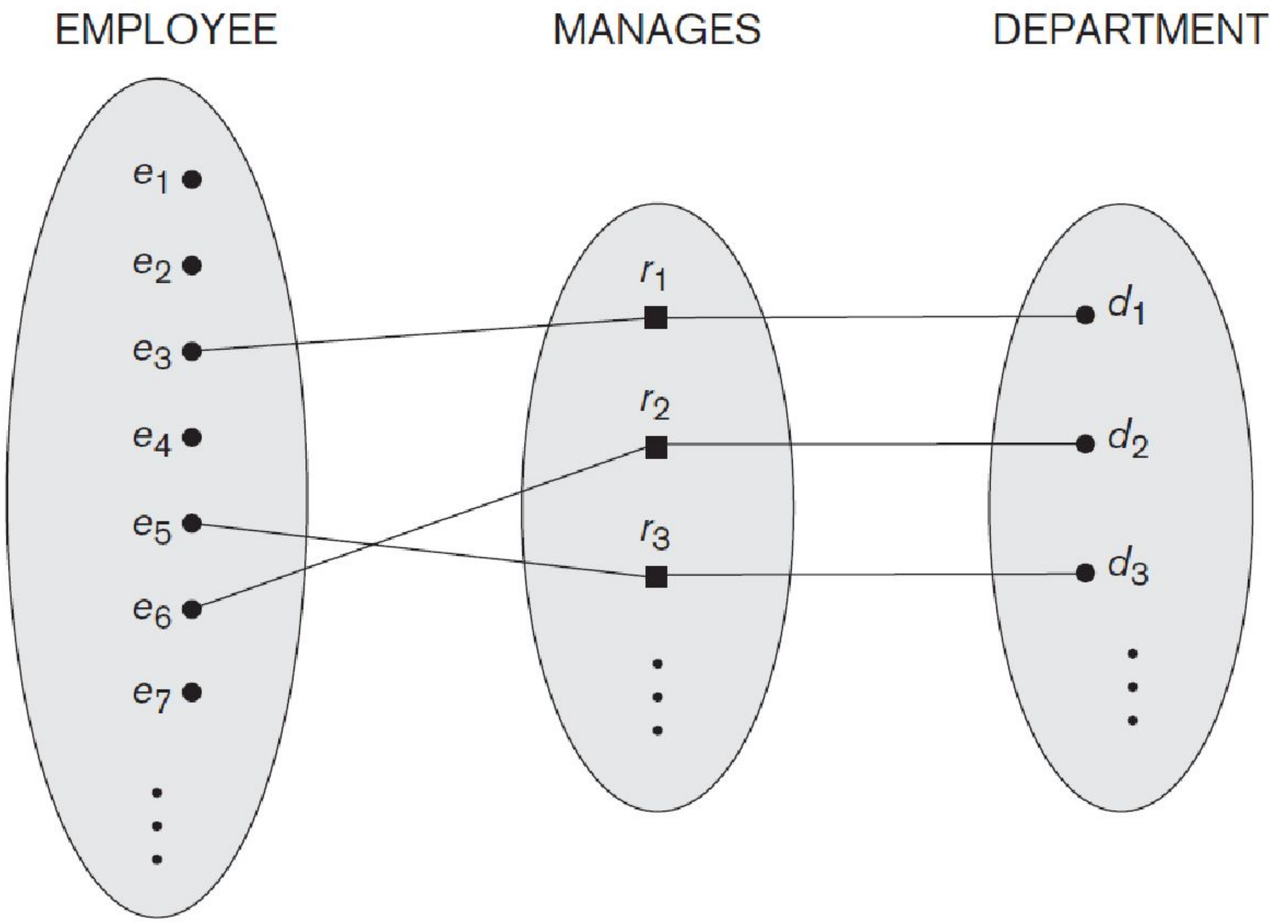
Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in.
- 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.
- 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.

Cardinality ratios

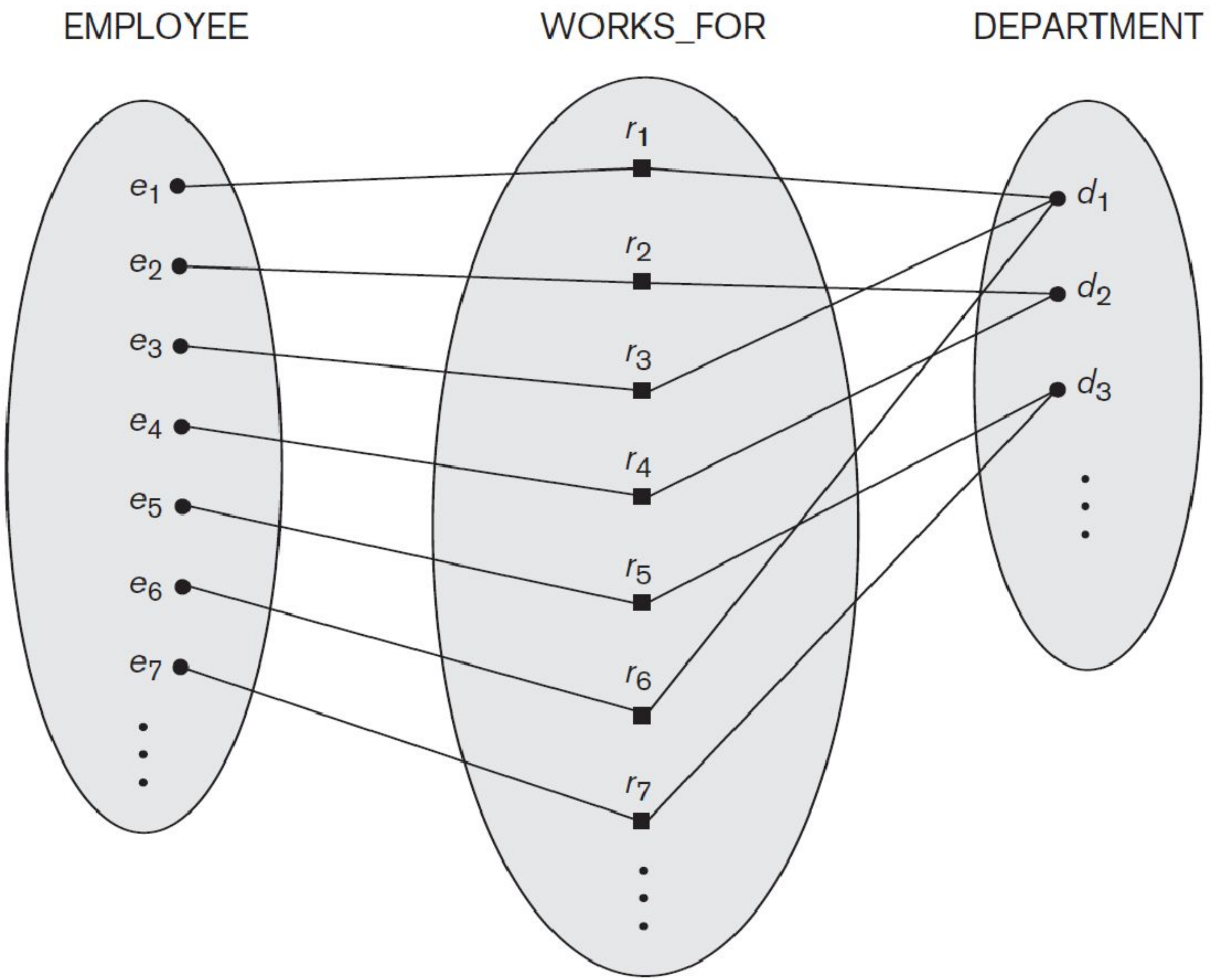
- Cardinality Ratio (specifies maximum participation)
  - One-to-one (1:1)
  - One-to-many (1:N)
  - Many-to-one (N:1)
  - Many-to-many (M:N)
- Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds.
- We can either specify no maximum (N) or a maximum of one (1) on participation.

One-to-one (1:1) Relationship

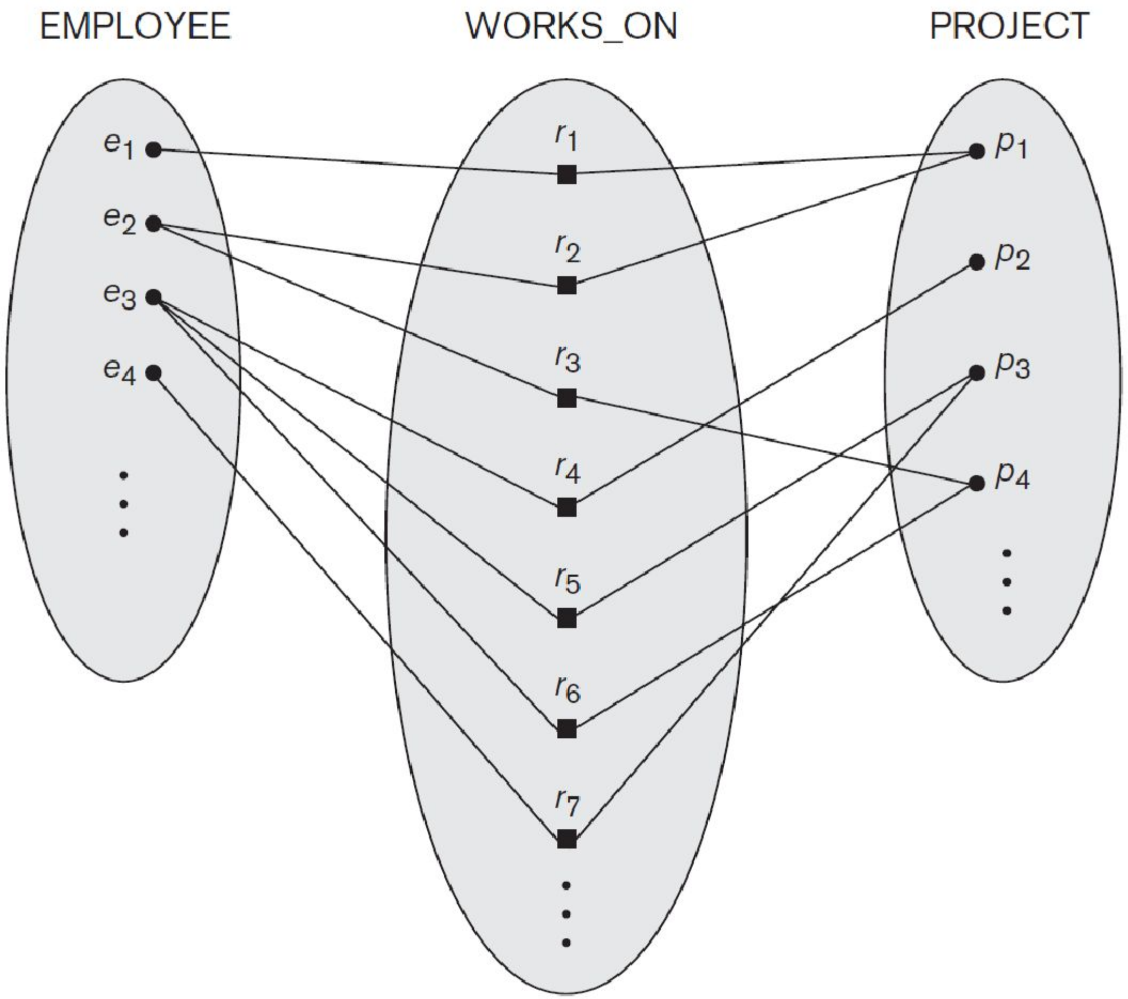


- An employee can manage one department only and a department can have one manager only.

Many-to-one (N:1) Relationship



Many-to-many (M:N) Relationship



- An employee can work on several projects and a project can have several employees.

Participation Constraints and Existence Dependencies

- The **participation constraint** specifies whether the existence of an entity depends on another entity via the relationship type.
- This constraint specifies the minimum number of relationship instances that each entity can participate in, and is called the **minimum cardinality constraint**.
- Two types of participation constraints
  - Total
  - Partial
- 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.

Participation Constraints and Existence Dependencies

- 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.**
- 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.
- Existence Dependency Constraint (specifies minimum participation) (also called participation constraint)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory participation, existence-dependent)

Participation Constraints and Existence Dependencies

- In ER diagram,
  - Total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship.
  - Partial participation is represented by a single line.

Attributes of Relationship Types

- A Relationship type can have attributes.
- 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.
    - A value of Hours depends on a particular (employee, project) combination.
  - To include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.
- Most relationship attributes are used with M:N relationships.
- Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types.
  - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship.

Weak Entity Types

- An entity type that does not have a key attribute of its own and and that is identification-dependent on another entity type.
- **Regular entity types** that do have a key attribute are called **strong entity types**.
- Entities belonging to a weak entity type are identified by being related to specific entities from **another entity type** in combination with one of their attribute values.
- We call this other entity type the **identifying** or **owner entity type** of the weak entity type.
- We call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type.

Weak Entity Types

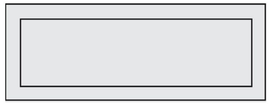
- A weak entity type always has a *total participation constraint* (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- Not every existence dependency results in a weak entity type.
- For example, a DRIVER_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key (License_number) and hence is not a weak entity.
- A weak entity type has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.
- If we assume that no two dependents of the same employee ever have the same name, the attribute Name of DEPENDENT is the partial key.
- In the worst case, a composite attribute of all the weak entity's attributes will be the partial key.

Weak Entity Types

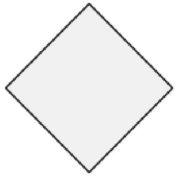
- Example:
  - A DEPENDENT entity is identified by the dependent's name, and the specific EMPLOYEE with whom the dependent is related.
  - Name of DEPENDENT is the partial key
  - DEPENDENT is a weak entity type
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF
- In ER diagram
  - Both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines.
  - The partial key attribute is underlined with a dashed or dotted line.
- An owner entity type may itself be a weak entity type.
- A weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two.

Notation for ER Diagrams

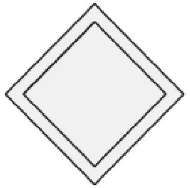
Entity



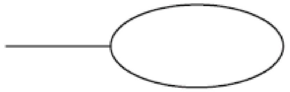
Weak Entity



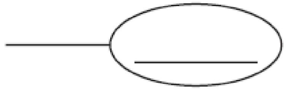
Relationship



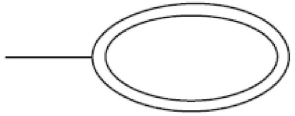
Indentifying Relationship



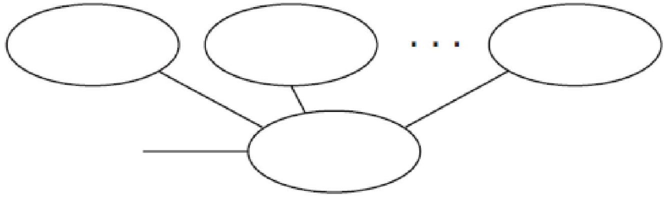
Attribute



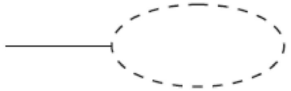
Key Attribute



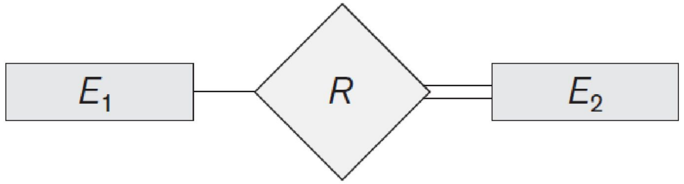
Multivalued Attribute



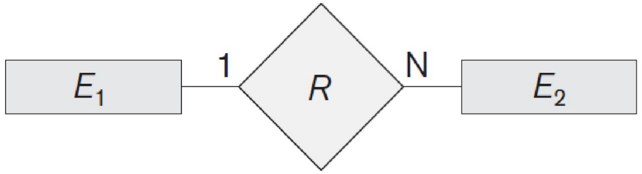
Composite Attribute



Derived Attribute



Total Participation of E_2 in R



Cardinality Ratio 1: N for E_1:E_2 in R

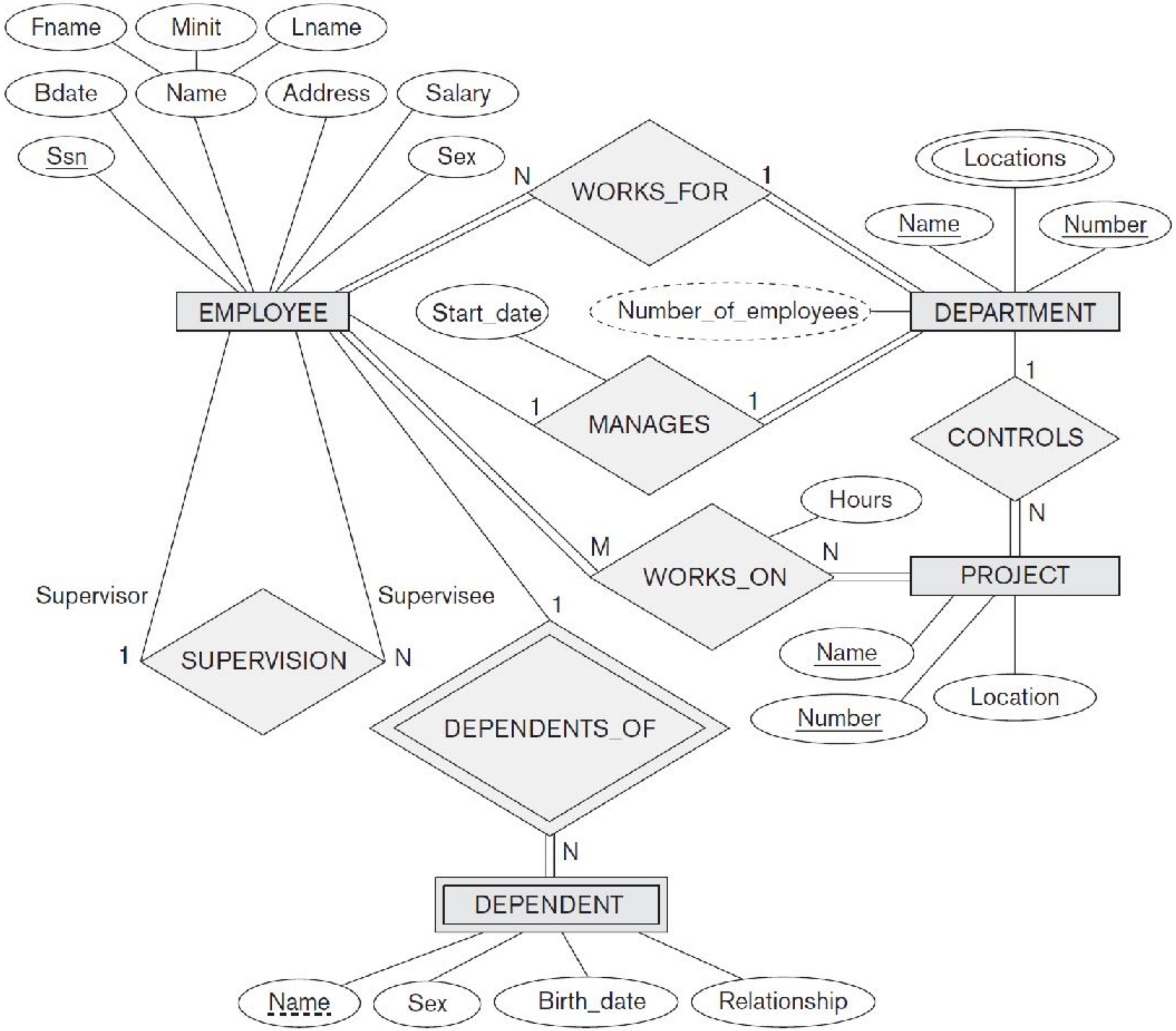
Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are binary relationships(degree 2)
- Listed below with their participating entity types:
  - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
  - MANAGES (also between EMPLOYEE, DEPARTMENT)
  - CONTROLS (between DEPARTMENT, PROJECT)
  - WORKS_ON (between EMPLOYEE, PROJECT)
  - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
  - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
  - Manager of DEPARTMENT -> MANAGES
  - Works_on of EMPLOYEE -> WORKS_ON
  - Department of EMPLOYEE -> WORKS_FOR
  - etc
- In general, more than one relationship type can exist between the same participating entity types
  - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
  - Different meanings and different relationship instances.

ER diagram for COMPANY database

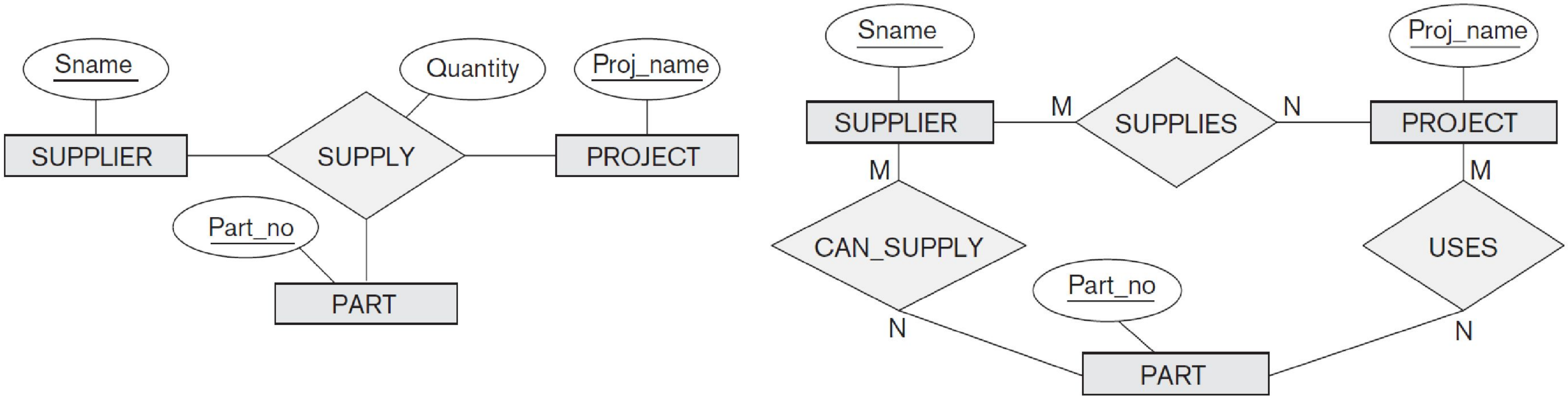


Relationships of Higher Degree

- Relationship types of degree 2 are called binary.
- Relationship types of degree 3 are called ternary and of degree n are called n-ary.
- In general, an n-ary relationship is not equivalent to n binary relationships.
- Constraints are harder to specify for higher-degree relationships (n > 2) than for binary relationships.
- In general, a relationship type R of degree n will have n edges in an ER diagram, one connecting R to each participating entity type.

Relationships of Higher Degree

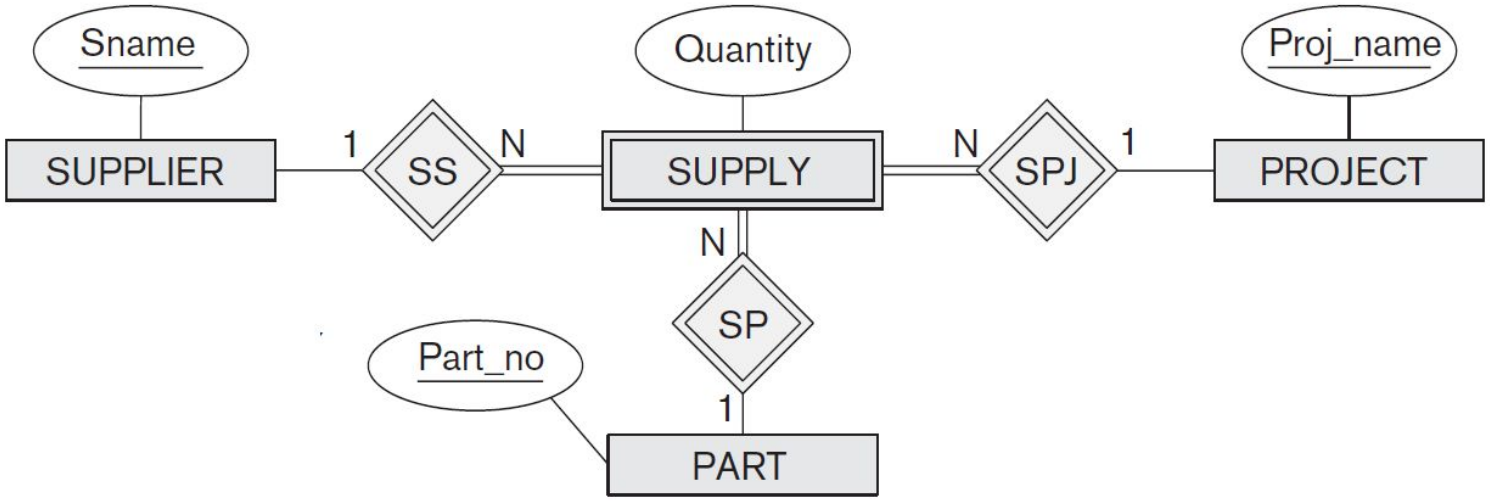
- In general, 3 binary relationships can represent different information than a single ternary relationship.



- Existence of three relationship instances (*s*, *p*), (*j*, *p*), and (*s*, *j*) in CAN_SUPPLY, USES, and SUPPLIES, respectively, does not necessarily imply that an instance (*s*, *j*, *p*) exists in the ternary relationship SUPPLY, because the *meaning is different*.

Relationships of Higher Degree

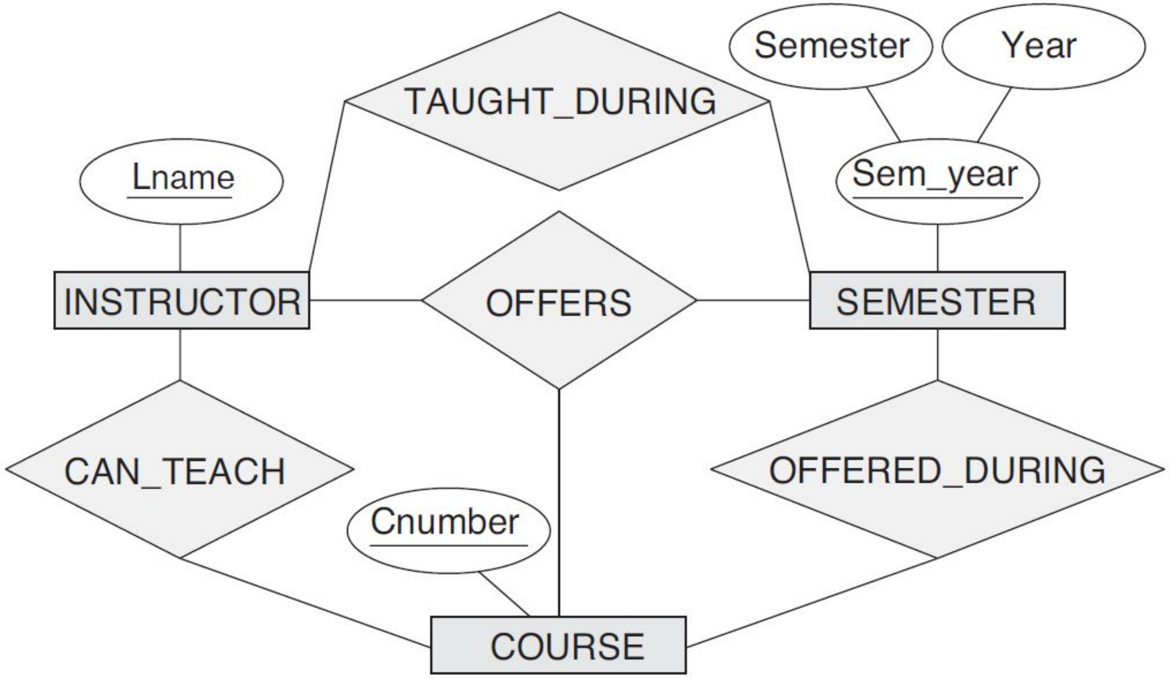
- In some cases, a ternary relationship can be represented as a weak entity.
- Then a weak entity type have multiple identifying relationships (and hence multiple owner entity types).



- SUPPLY is represented as a weak entity type, with no partial key and with three identifying relationships.

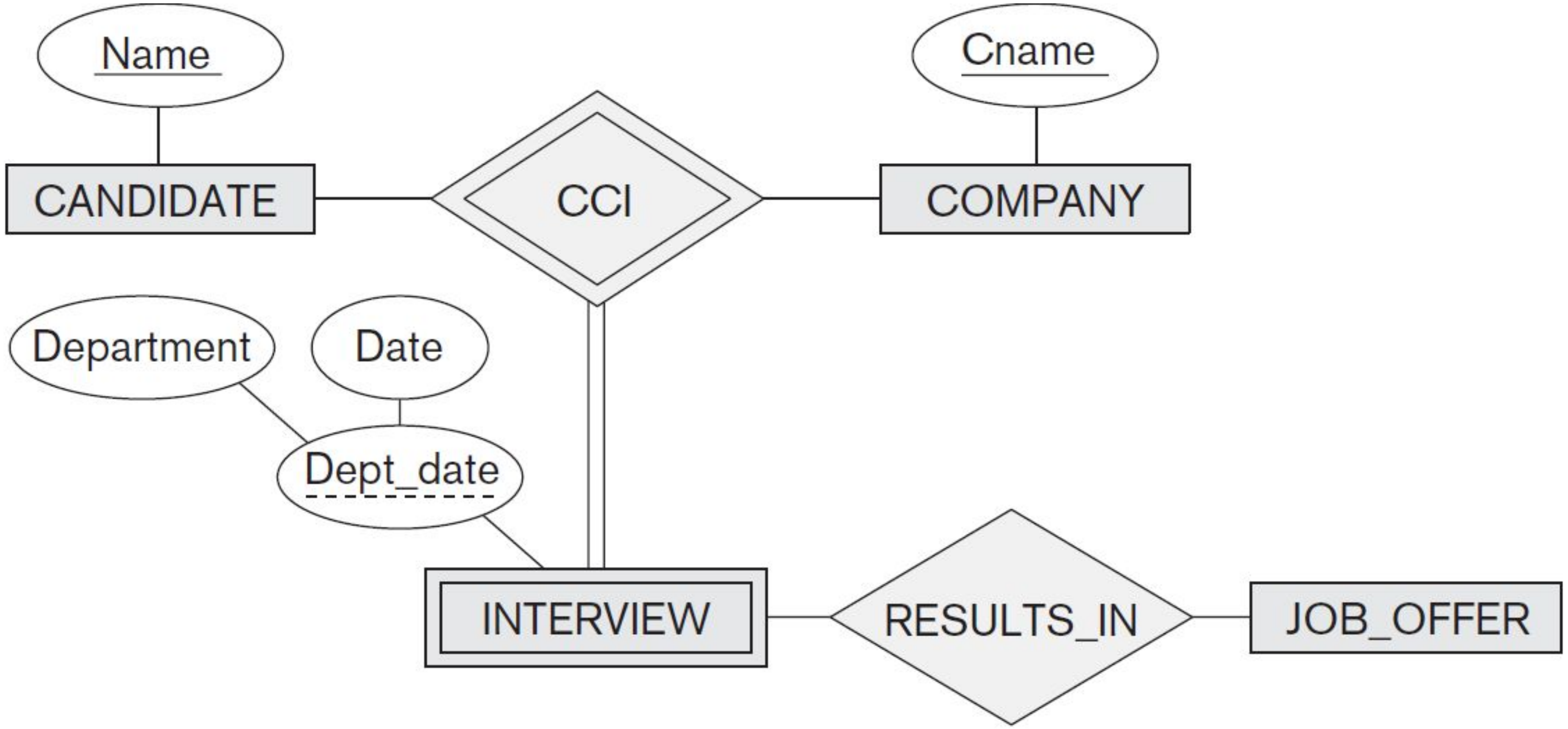
Relationships of Higher Degree

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



- TAUGHT_DURING binary relationship can be derived from the ternary relationship OFFERS.

Weak entity type with a ternary (or n-ary) identifying relationship type



- ❑ Part of a database that keeps track of candidates interviewing for jobs at various companies.
- ❑ INTERVIEW entity is uniquely identified by a candidate, a company, and the combination of the date and department of the interview.