# CEng 302
# Database Management Systems

# The Entity-Relationship (ER) Model

**Prof. Dr. Adnan YAZICI**
Department of Computer Engineering,
Middle East Technical University
(**Fall 2021**)

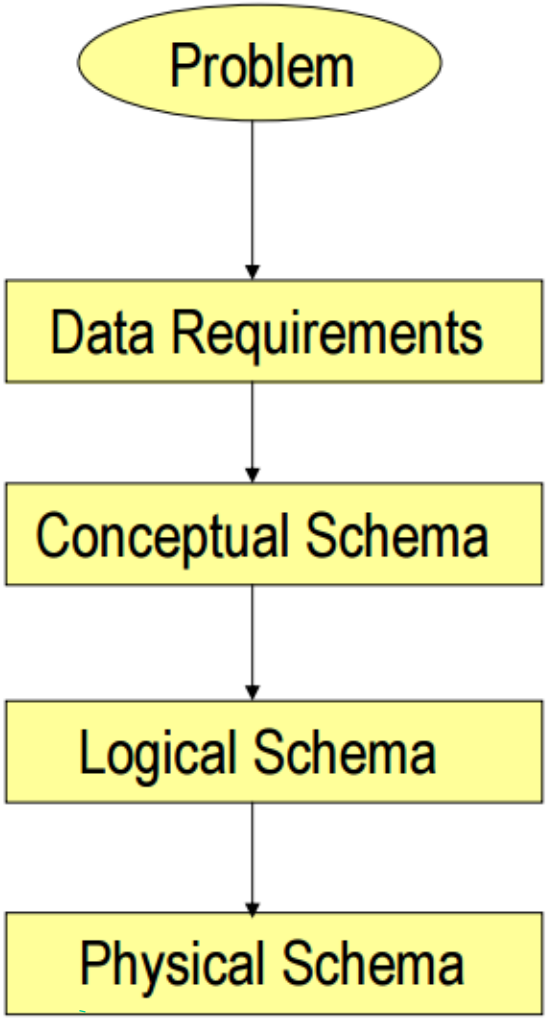
# Outline

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

# Overview of Database Design Process

- Two main activities:
  - Database design
  - Applications design
- Focus on this lecture (2 weeks) on database design
  - To design the conceptual schema for a database application
- Applications design focuses on the programs and interfaces that access the database
  - Generally considered part of software engineering

# Database Design Goes Through Stages




*Requirements Analysis*

*Data Analysis, Conceptual Design*

*Logical Database Design*

*Physical Database Design*

# Database Design Process

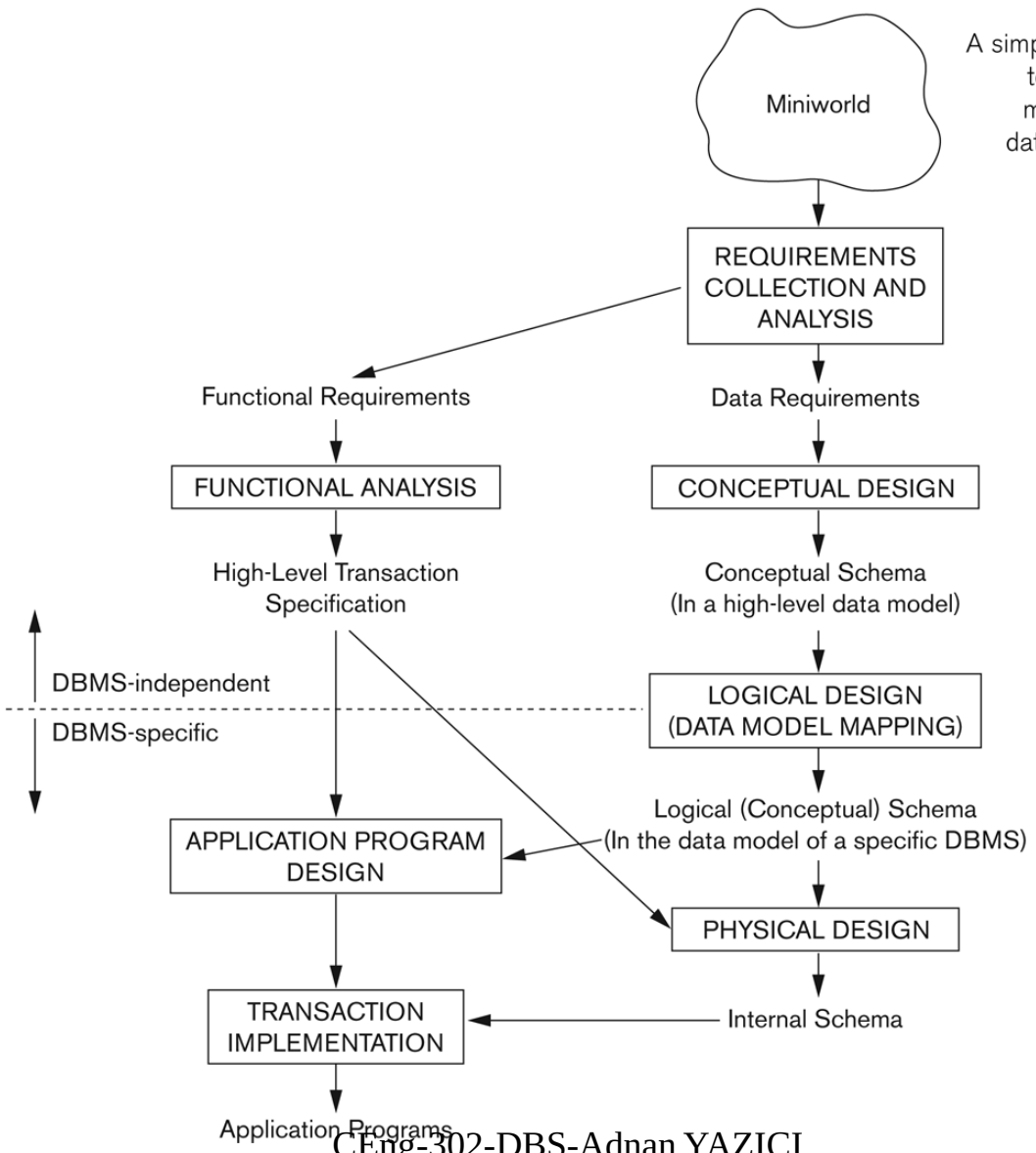



**Figure 3.1**
A simplified diagram to illustrate the main phases of database design.

# Overview of Database Design

- **Conceptual design**: (**ER Model** *is used at this stage.*)

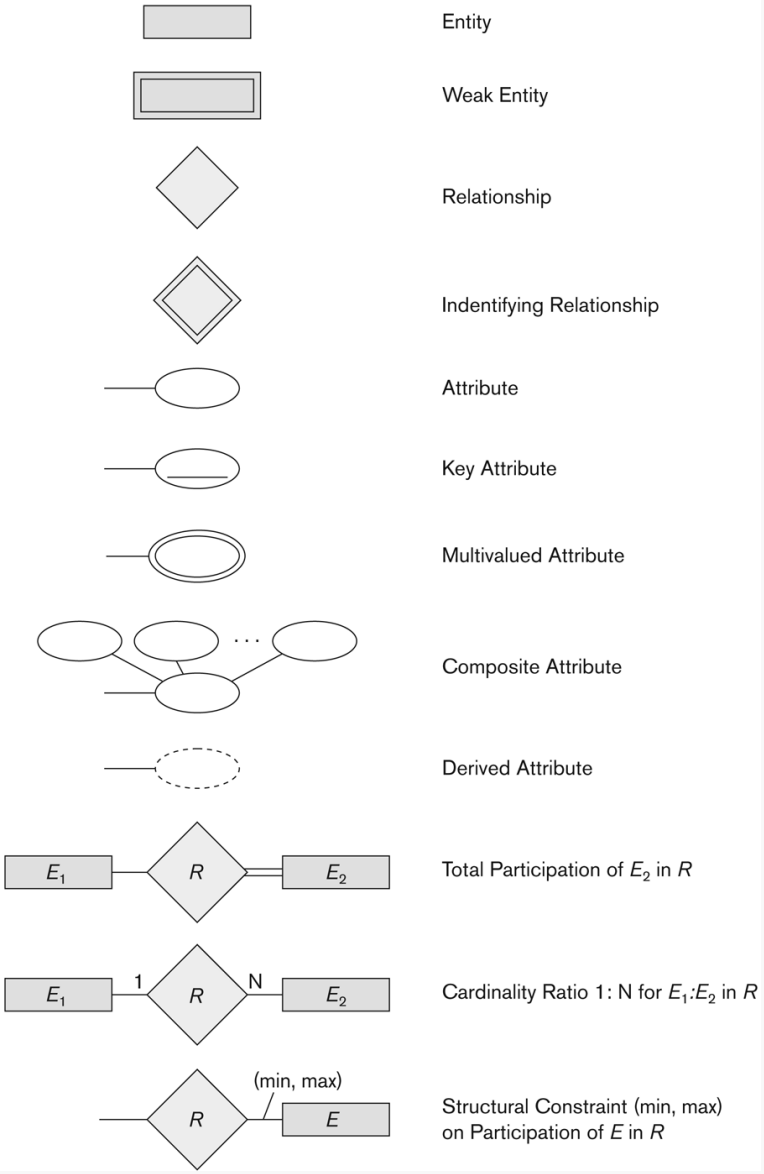
  ER model has three main concepts:
  - Entities (and their entity types and entity sets)
  - Attributes (simple, composite, multivalued)
  - Relationships (and their relationship types and relationship sets)
  - We can map an ER diagram into a relational schema.
- **Schema Refinement:** (normalization, db design)
  - Check relational schema for redundancies and anomalies and lossless join decomposition.
- **Physical Database Design and Tuning:**
  - Consider typical workloads and further refine the db design.

# Example: COMPANY Database

- We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:
  - The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
  - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.
  - We store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
    - Each employee *works for* one department but may *work on* several projects.
    - We keep track of the number of hours per week that an employee currently works on each project.
    - We also keep track of the *direct supervisor* of each employee.
  - Each employee may *have* a number of DEPENDENTs.
    - For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

# Summary of notation for ER diagrams

**Figure 3.14**
Summary of the notation for ER diagrams.



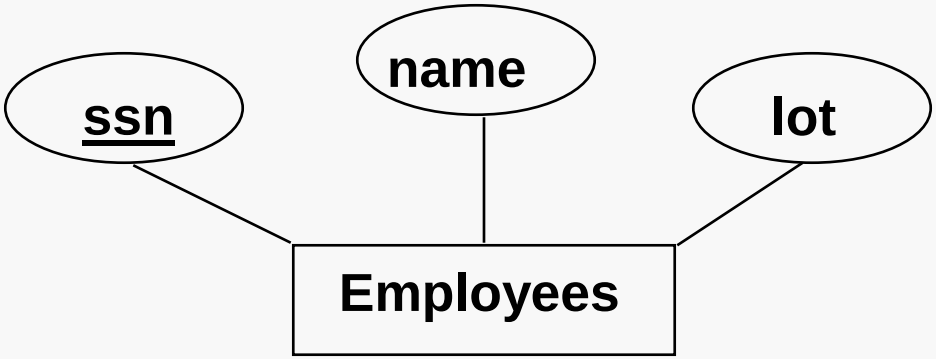
# Types of Attributes

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

# Displaying an Entity type

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

# ER Model Basics




- **Entity:** Real-world object distinguishable from other objects. An entity is described (in DB) using a set of **attributes.**
- **Entity Set:** A collection of similar entities.

  E.g., all students.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a **key**.
  - Each attribute has a **domain.**

# Entity Type CAR with two keys and a corresponding Entity Set

(a)

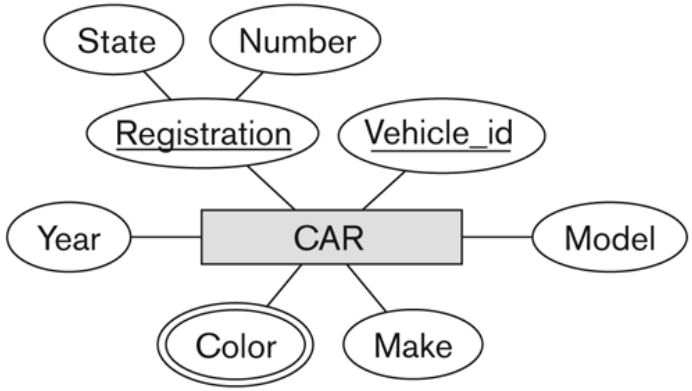



**Figure 3.7**
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(b)

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

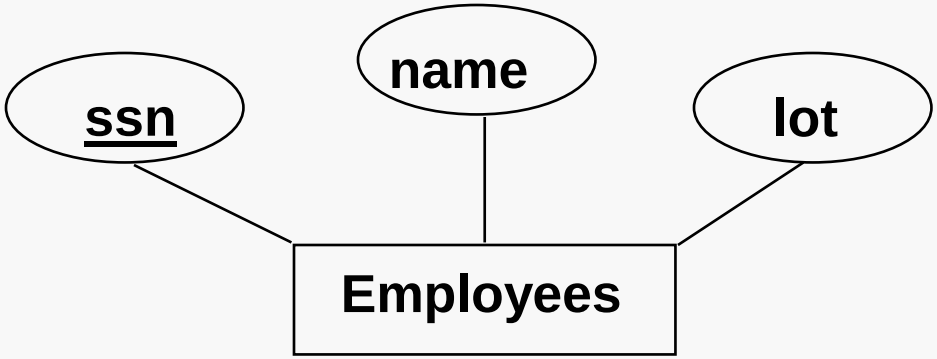
$CAR_1$
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

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

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

# Logical DB Design: ER to Relational

- Entity sets to tables:

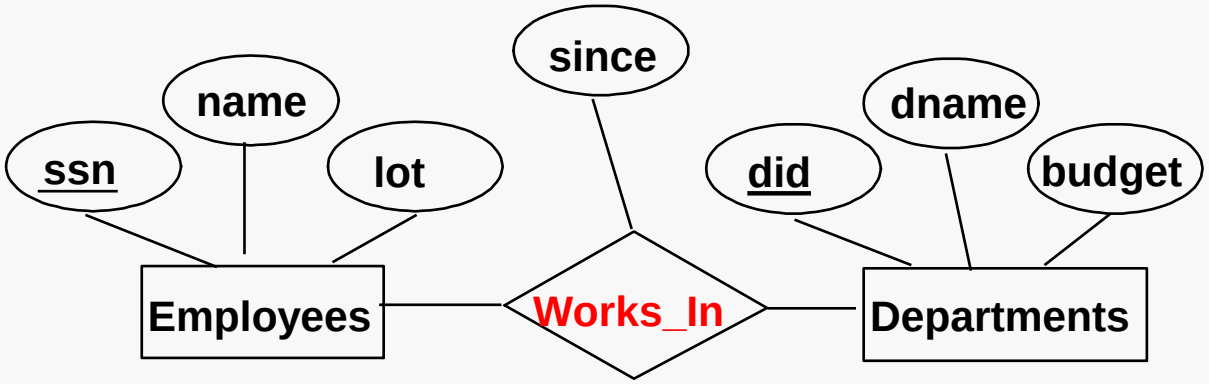


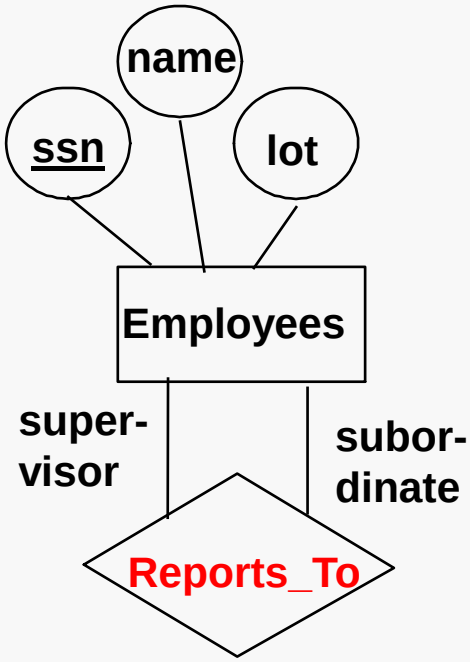

```
CREATE TABLE Employees
    (ssn CHAR(11),
    name VAR CHAR(20),
    lot  INTEGER,
    PRIMARY KEY  (ssn))
```

# Relationships and Relationship Types

- A **relationship** relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith *works on* the ProductX PROJECT, or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
  - Both MANAGES and WORKS_ON are *binary* relationships.

# ER Model Basics (Contd.)



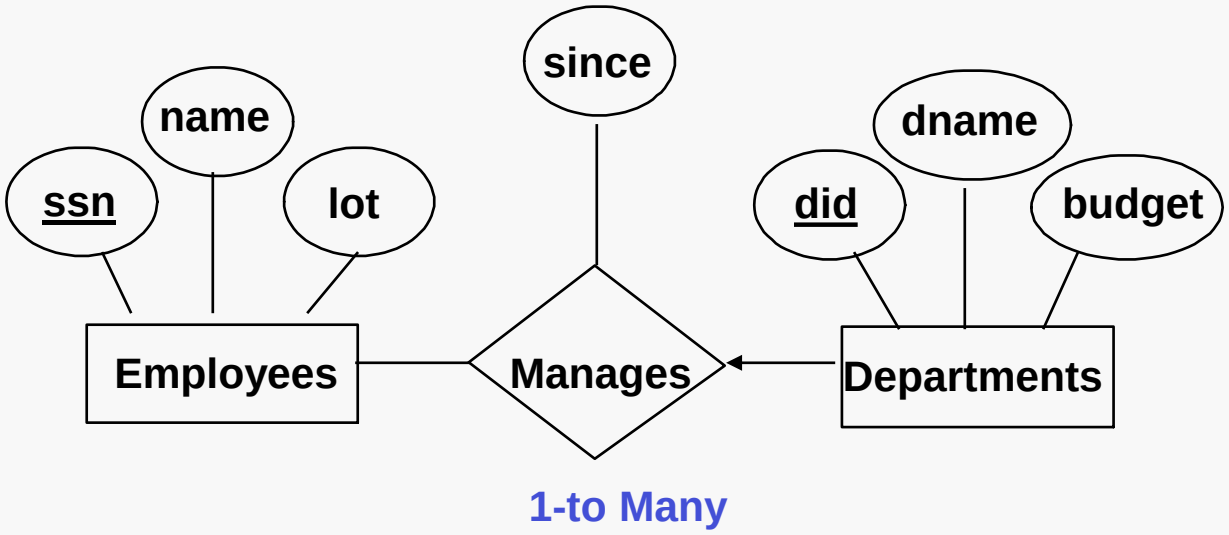

- **Relationship:** Association among two or more entities.
  - e.g., John works in Pharmacy department.
- **Relationship Set:** Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets $E_1$ ... $E_n$; each relationship in R involves entities $e_1 \in E_1$, ..., $e_n \in E_n$
    - Same entity set could participate in different relationship sets, or in different "roles" in same set.

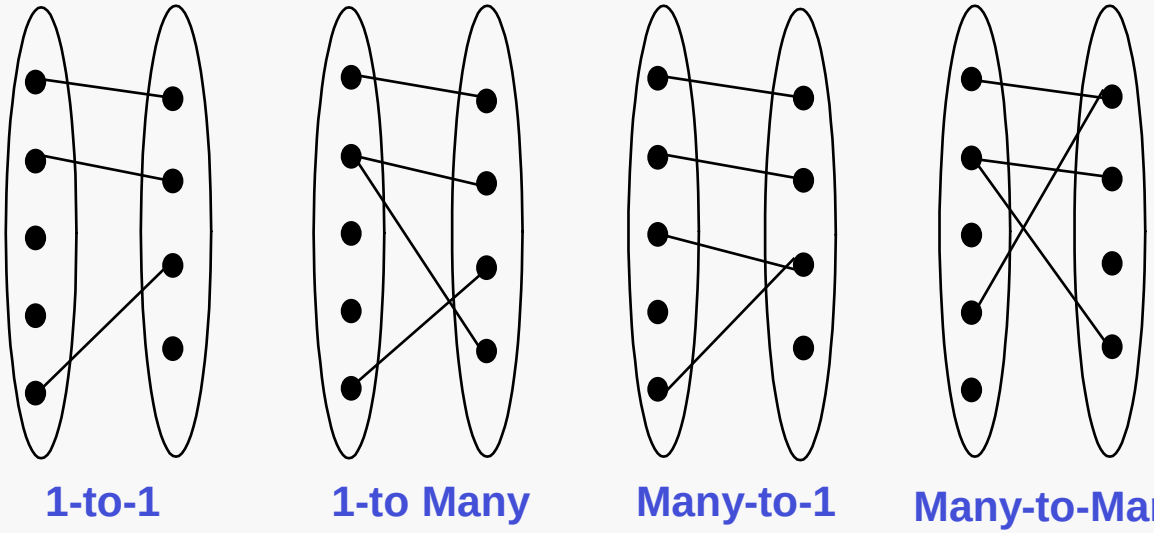
# Constraints on Relationships

- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - **Cardinality Ratio** (specifies *maximum* participation)
    - **One-to-one (1:1)**
    - **One-to-many (1:N)** or **Many-to-one (N:1)**
    - **Many-to-many (M:N)**
  - **Existence Dependency Constraint** (specifies *minimum* participation) (also called participation constraint)
    - **zero** (optional participation, not existence-dependent), **partial participation**
    - **one or more** (mandatory participation, existence-dependent), **total participation**.

# Constraints

- An employee can work in many departments; that is, a dept can have many employees.
- In contrast, each dept has at most one manager, according to the key constraint on Manages.



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

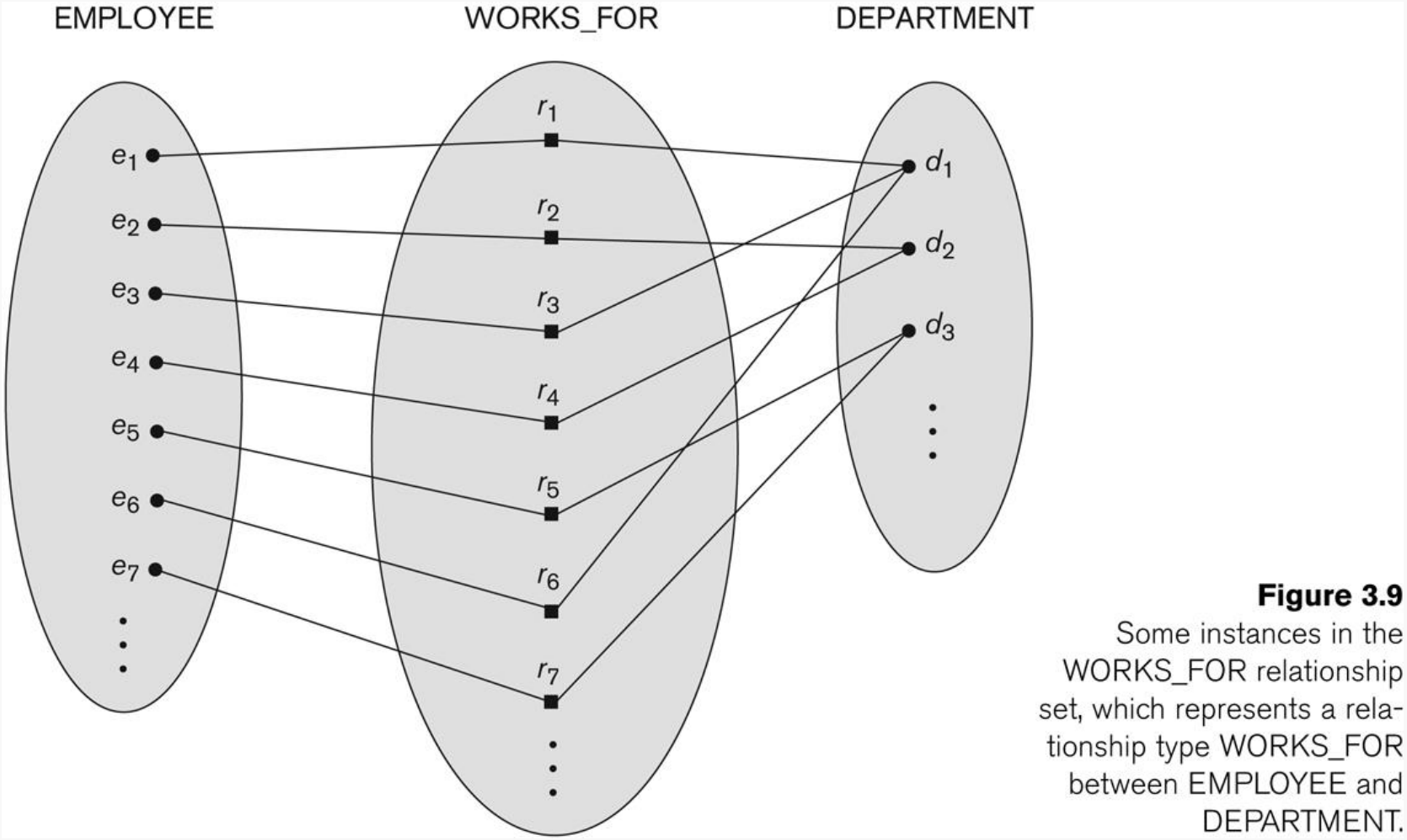



**Figure 3.9**
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

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

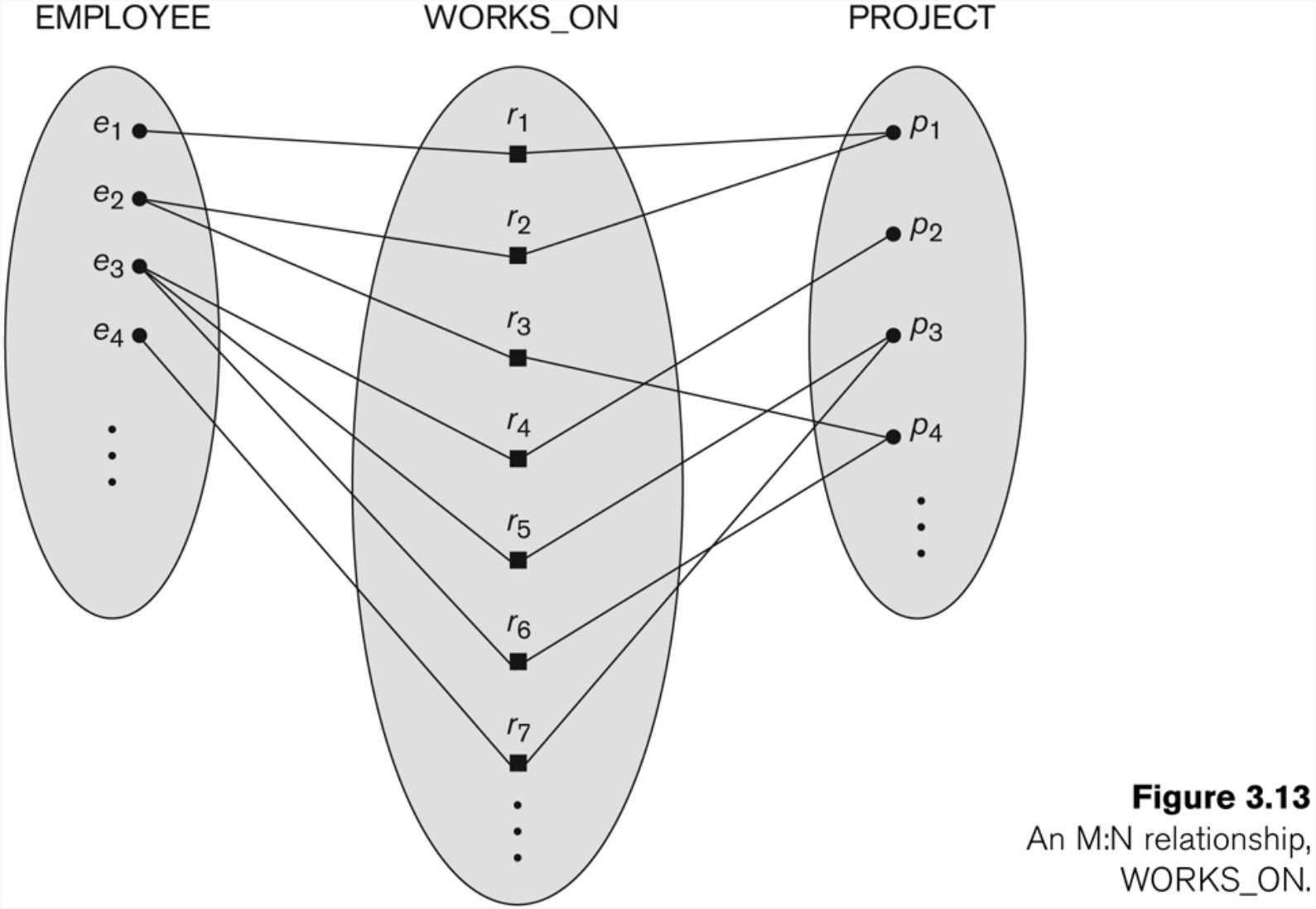



**Figure 3.13**
An M:N relationship, WORKS_ON.

# Weak Entity Types

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

# Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE Dependent(
   name CHAR(20),
   birthdate INTEGER,
   sex CHAR (1),
   relationship CHAR (10)
   ssn CHAR(11) NOT NULL,
   PRIMARY KEY (name, ssn),
   FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

# ER DIAGRAM – Relationship Types are:

**WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF**

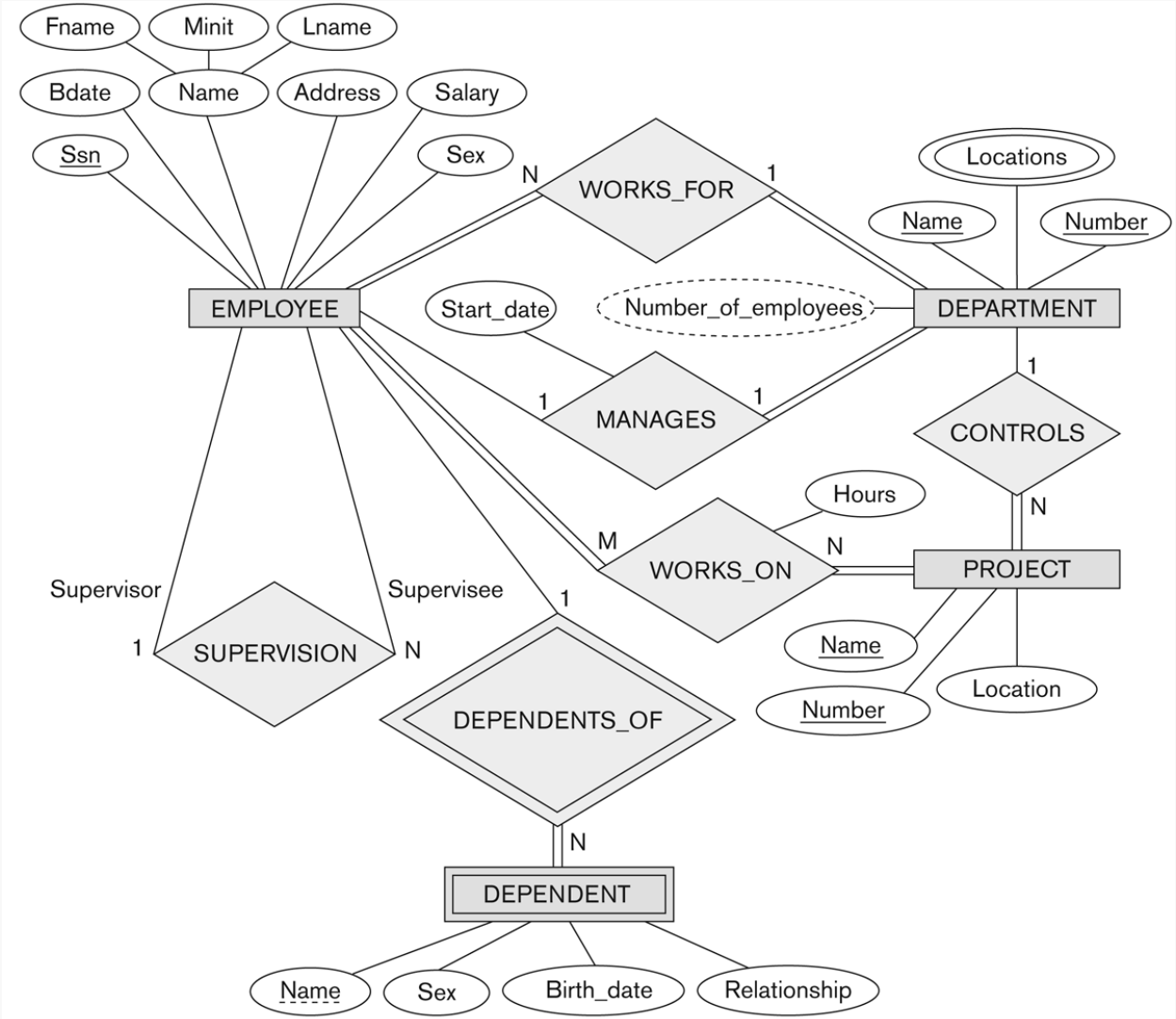
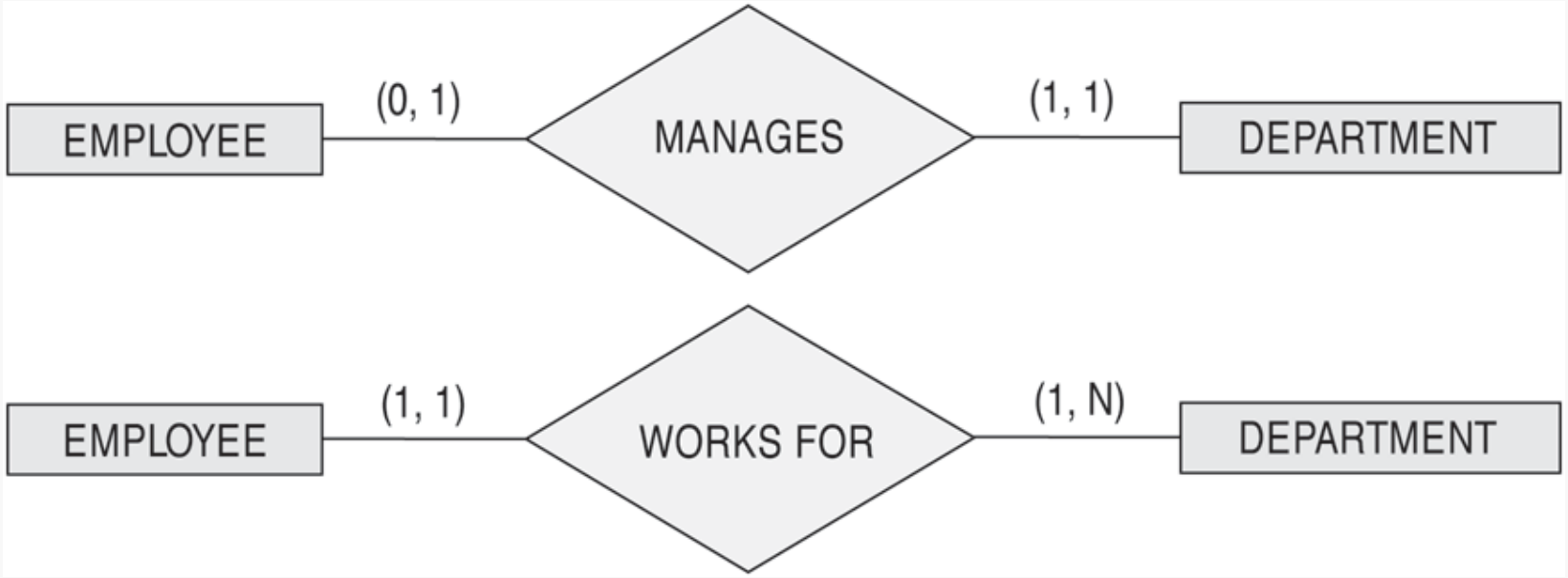



**Figure 3.2**
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

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

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department, but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

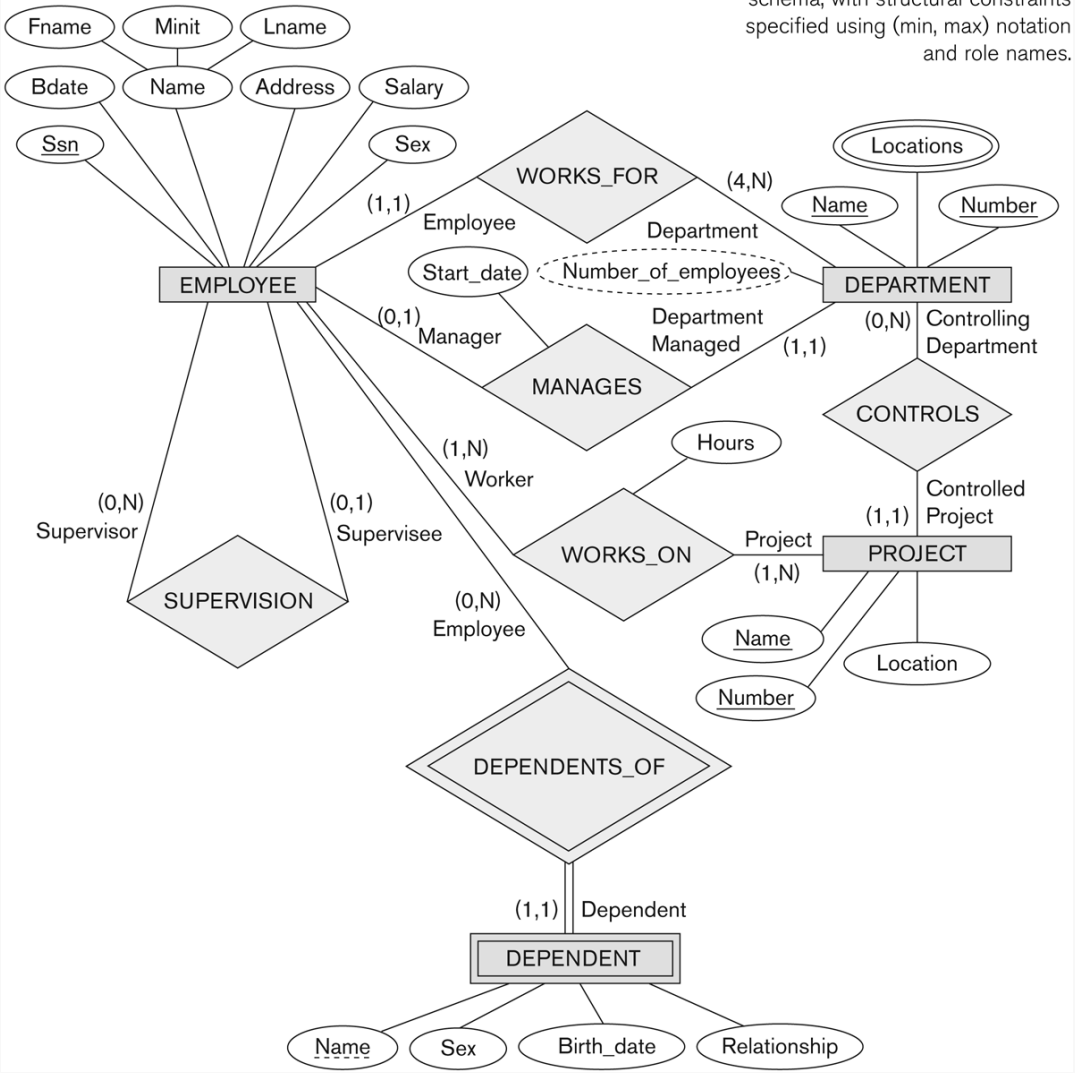
# *The (min,max) notation for relationship constraints*




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

# *COMPANY ER Schema Diagram using (min, max) notation*

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



# Relationship Sets to Tables

- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

```
CREATE TABLE Works_In(
  ssn  CHAR(11),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
      REFERENCES Employees,
  FOREIGN KEY (did)
      REFERENCES Departments)
```

# Translating ER Diagrams with Key Constraints

- Map relationship to a seperate table:
  - Note that did is the key now!
  - **Separate tables** for Manages, Employees and Departments.
- Since each department has a unique manager, we could instead **combine** Manages and Departments in one table, Dept_Mng.

**Two alternative table designs:**

1

```
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```

2

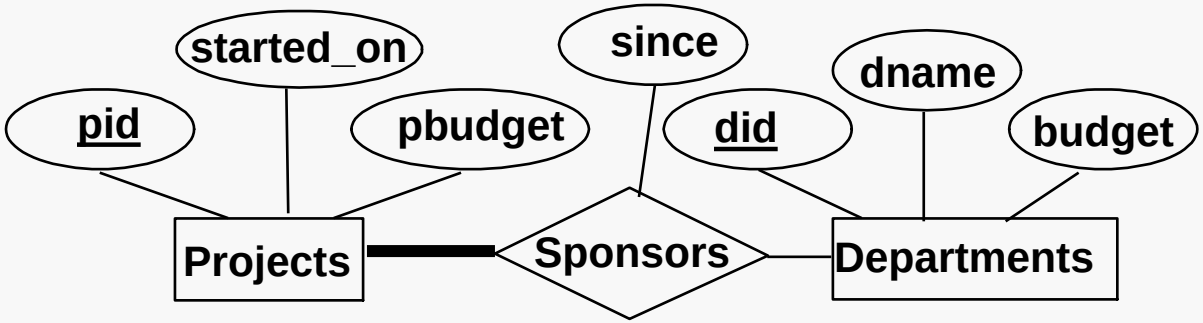
```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did, ssn),
  FOREIGN KEY (ssn) REFERENCES Employees)
        ON DELETE NO ACTION)
```

# Aggregation

- Aggregation is used when we have to model a relationship involving (entity sets and) another relation: a *relationship set*.

Suppose:

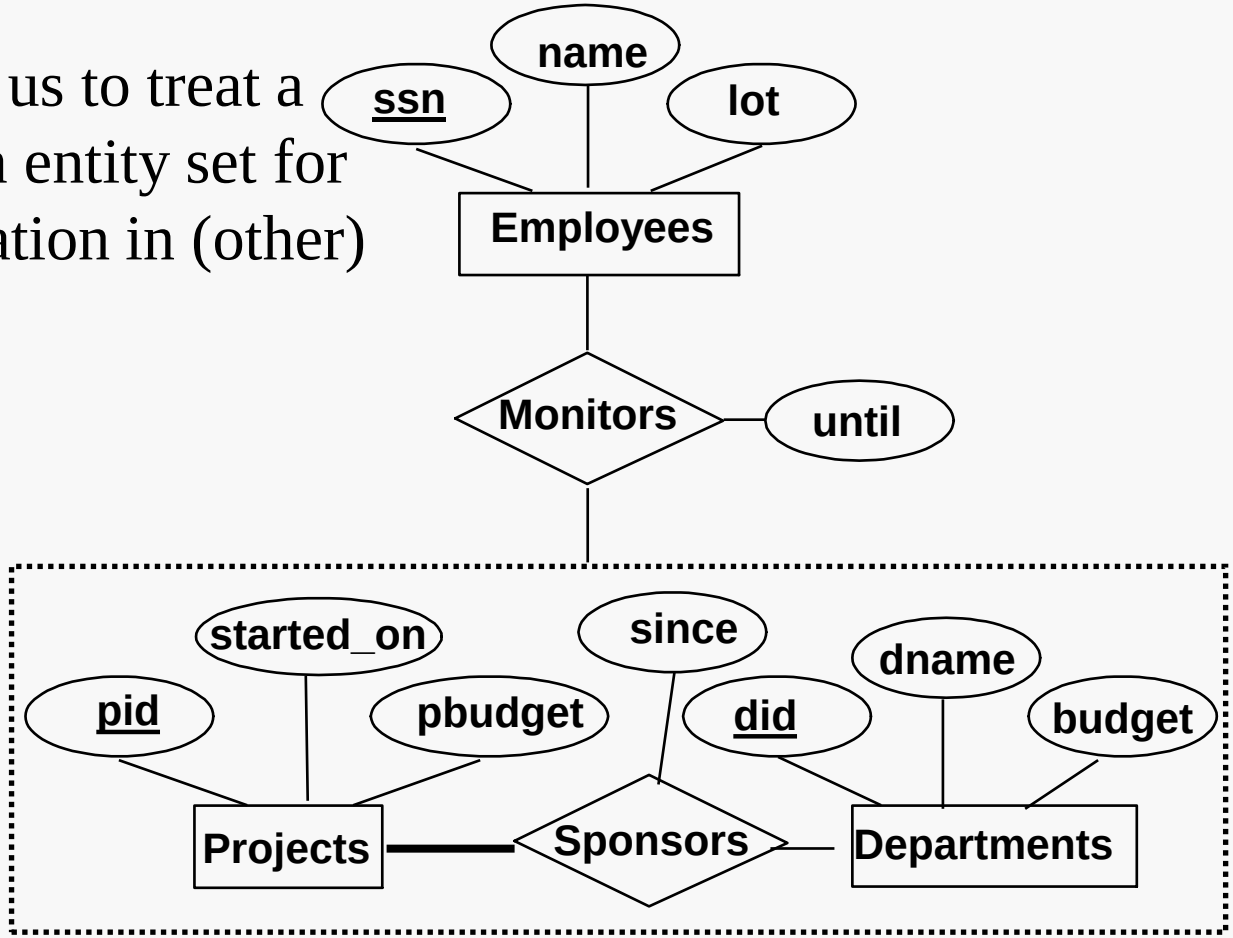
- entity set *Projects*
- each *Projects* is sponsored by at least one *Departments* and each *Departments Sponsors* many *Projects*.




n-m

# Aggregation

- **Aggregation** allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships.
- Each *Departments* that sponsors a *Projects* might assign employees to monitor sponsorship



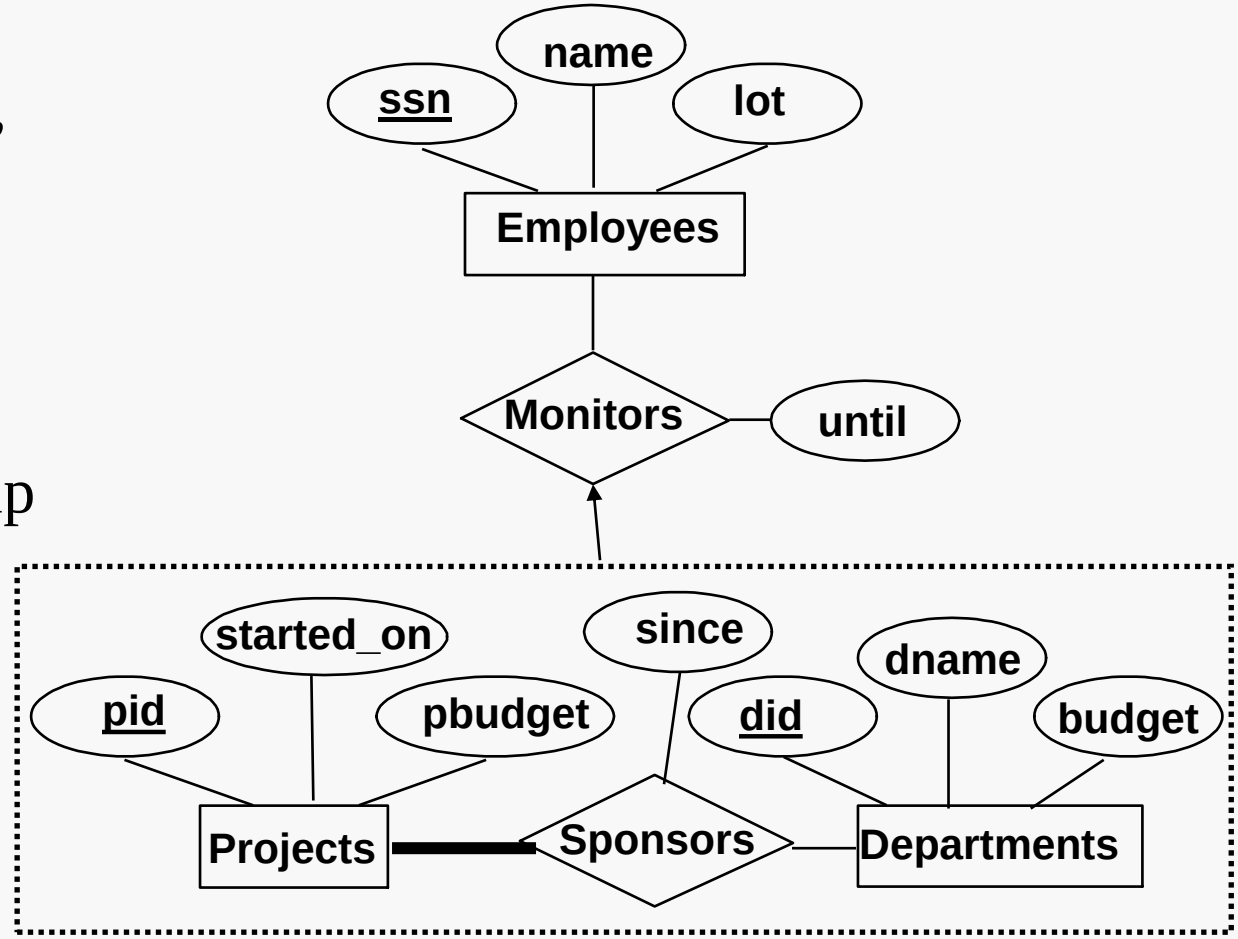

**Intuitively…**

- *Monitors* should be a relationship set that associates a *Sponsors* relation (versus a *Projects* or *Departments*) with an *Employees* entity.

# Aggregation vs Ternary relation(Contd.)

- *Monitors* is a distinct relationship, with a descriptive attribute (*until*).

- Also, we can say that each sponsorship is monitored by at most one employee.



# Entity vs. Attribute

- Should **address** be an attribute of Employees or an entity (connected to Employees by a relationship)?

- Depends upon the use we want to make of address information, and the semantics of the data:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

# 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.

# Discussion of n-ary relationships (n > 2)

- In general, 3 binary relationships can represent different information than a single ternary relationship (see the next slide).
- If needed, the binary and n-ary relationships can all be included in the schema design (see the next slide, where all relationships convey different meanings).
- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types) (see the next slide, Figure 3.17c).

# Example of a ternary relationship

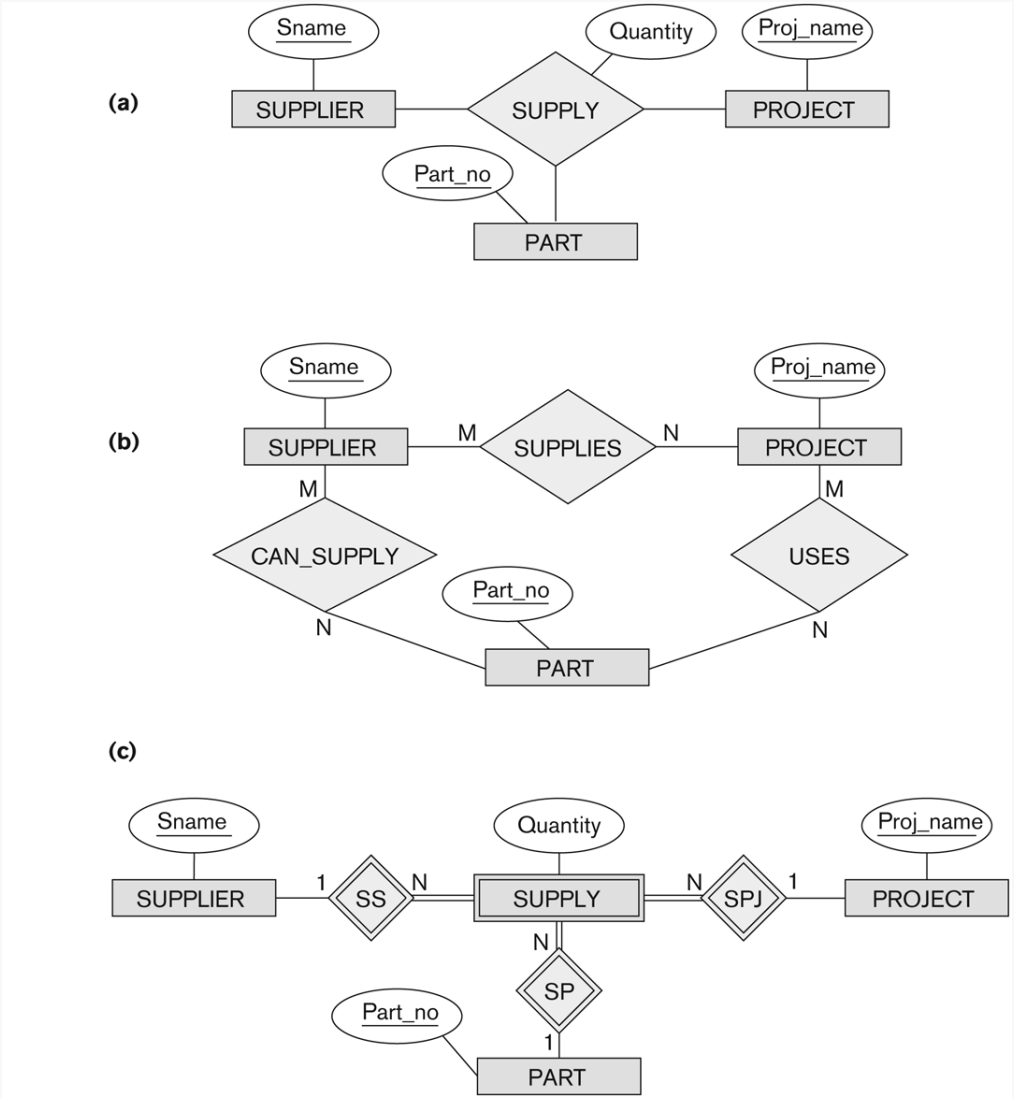



**Figure 3.17**
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

# Binary vs. Ternary Relationships (Cont.)

- An example in the other direction: a ternary relation **Supply** relates entity sets **Parts, Projects** (or alternatively **Departments**) and **Suppliers,** and has descriptive attribute ***quantity***.
- No combination of binary relationships is an adequate substitute for at least two reasons:
  - S "can-supply" P, Prj (or D) "needs" P, and Prj (or D) "deals-with" S

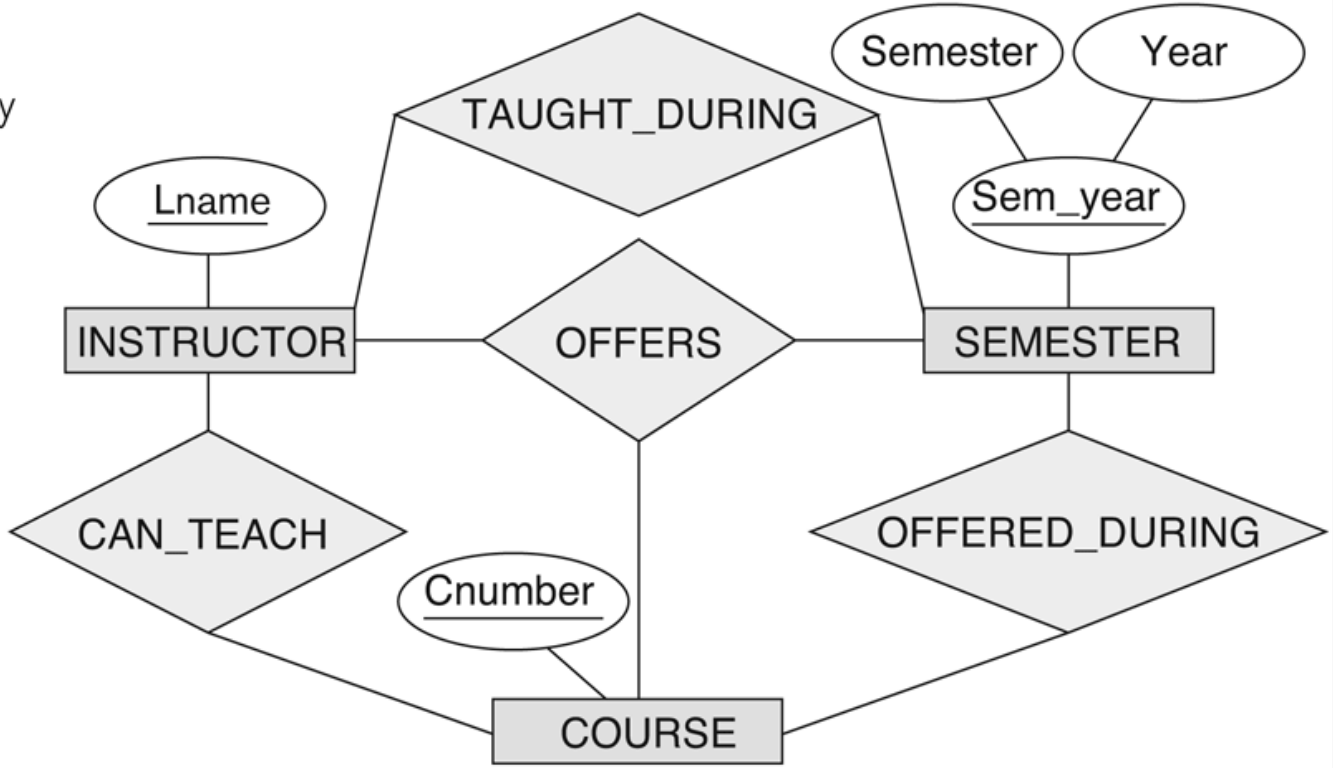
    does not imply that D has agreed to buy P from S.
  - How do we record *quantity*?

# Discussion of n-ary relationships (n > 2)

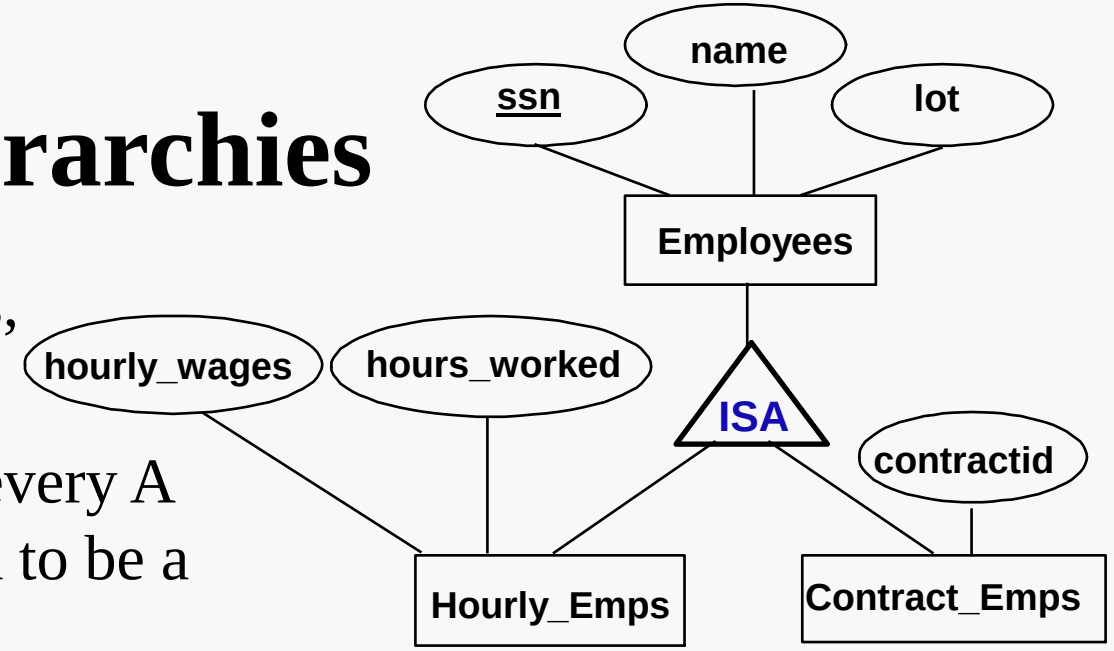
- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant.
- For example, the TAUGHT_DURING binary relationship can be derived from the ternary relationship OFFERS (based on the meaning of the relationships).

# Another example of a ternary relationship

**Figure 3.18**
Another example of ternary versus binary relationship types.



# ISA (`is a´) Hierarchies




- As in C++, or other PLs, attributes are inherited.
  - If we declare A **ISA** B, every A entity is also considered to be a B entity.

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.
- **Disjoint/overlap constraints**: Can Joe be an *Hourly_Emps* as well as a *Contract_Emps* entity? **(Allowed/disallowed)**
- **Covering constraints**: Does every *Employees* entity also have to be an *Hourly_Emps* or a *Contract_Emps* entity? **(Yes/no)**

# Conceptual Design Using the ER Model

- **Design choices:**
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships: Binary or ternary? Aggregation?
- Constraints in the ER Model:
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.