# Analysis

## Structuring System Data Requirements

Prepared by:

Hiranya Prasad Bastakoti

# Contents

- Introduction
- Conceptual Data Modeling
- Gathering Information for Conceptual Data Modeling
- Introduction to E-R Modeling

**Figure 9-1** Systems development life cycle with analysis phase highlighted

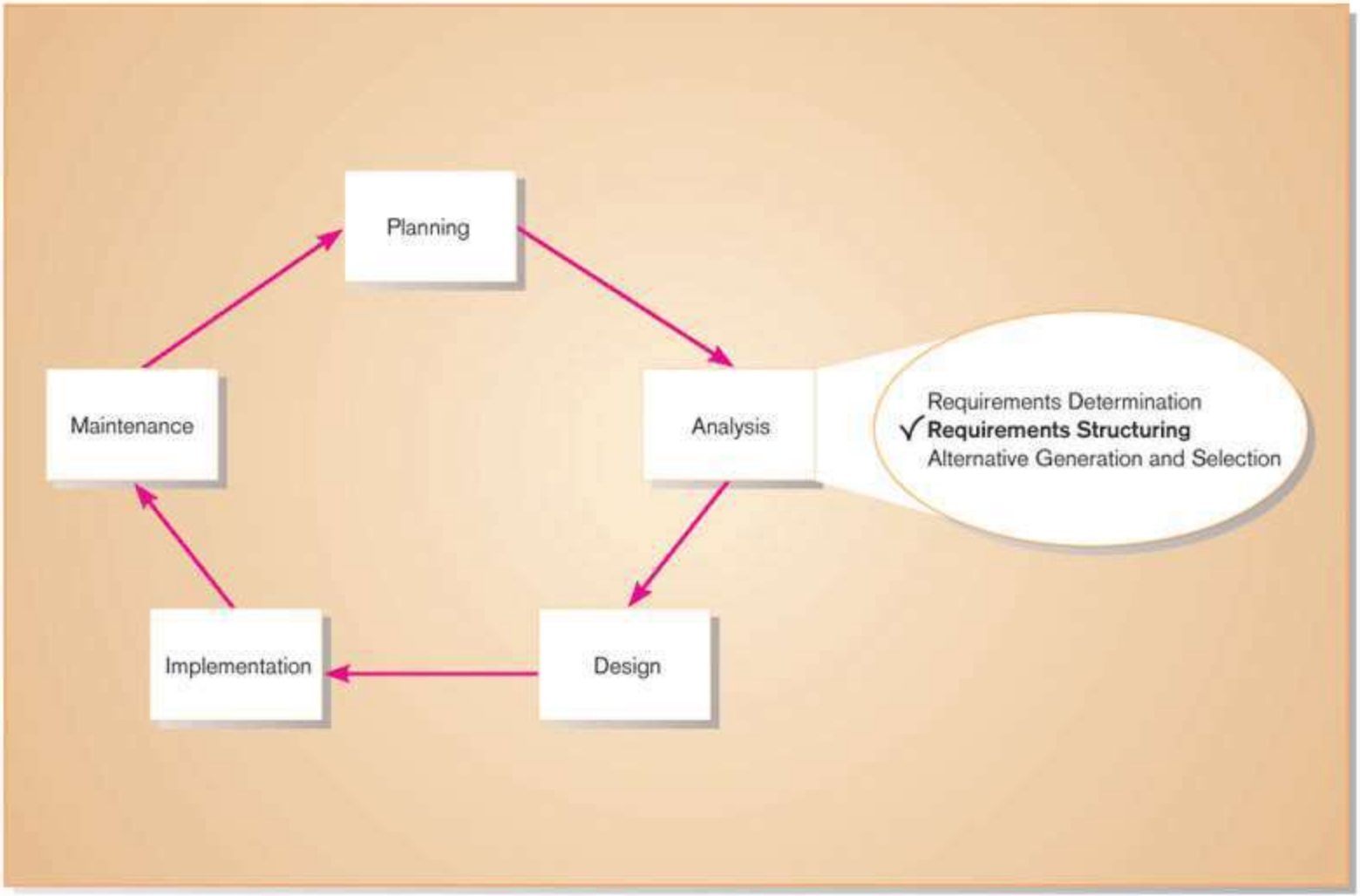
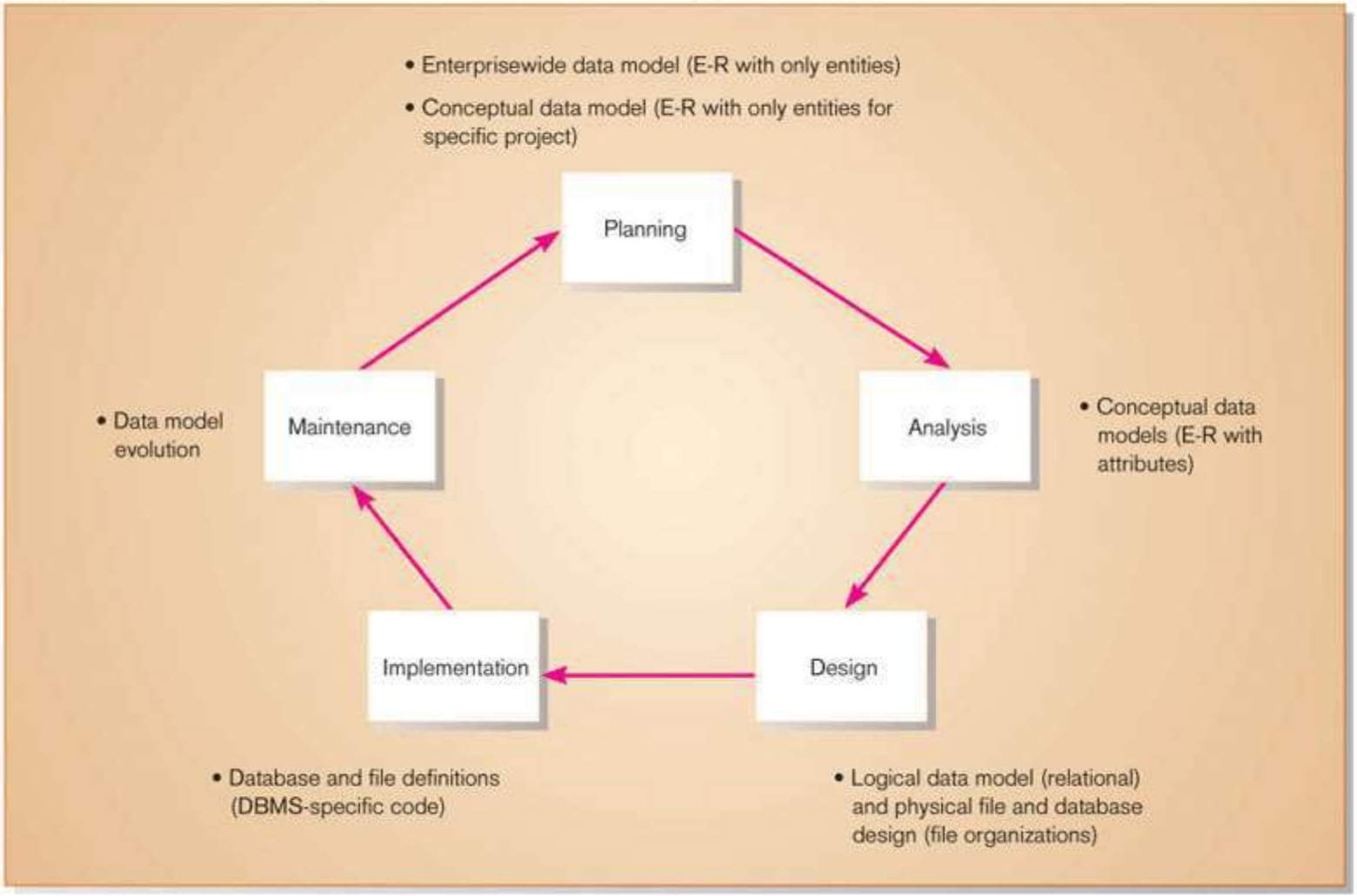


**Figure 9-2** Relationship between data modeling and the systems development life cycle



# Conceptual Data Modeling

- A detailed model that captures the overall structure of data in an organization
- Independent of any database management system (DBMS) or other implementation considerations

# Process of Conceptual Data Modeling

- Develop a data model for the current system
- Develop a new conceptual data model that includes all requirements of the new system
- In the design stage, the conceptual data model is translated into a physical design
- Project repository links all design and data modeling steps performed during SDLC

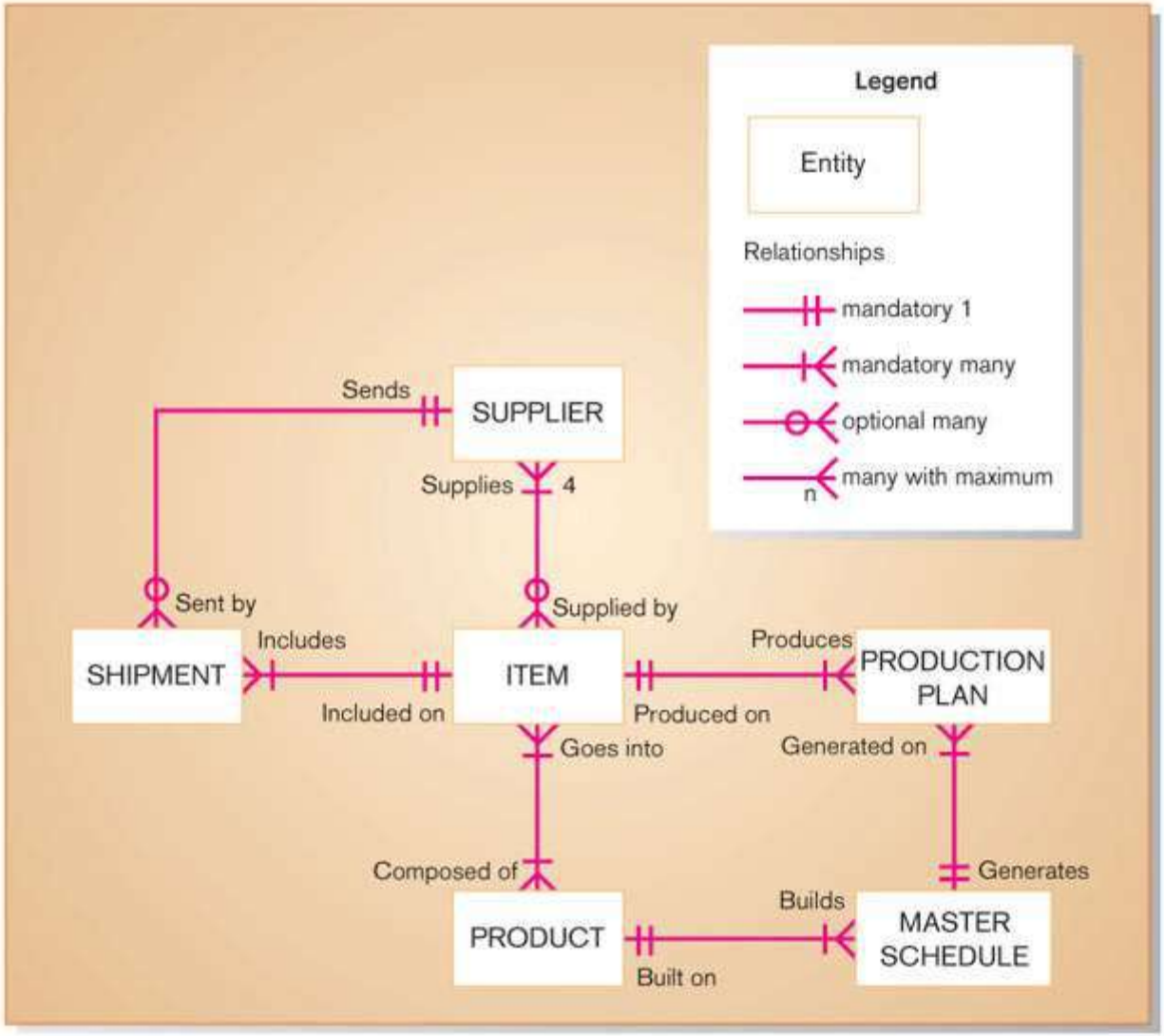
# Deliverables and Outcome

- Primary deliverable is an entity-relationship (E-R) diagram or class diagram
- As many as 4 E-R or class diagrams are produced and analyzed
  - E-R diagram that covers data needed in the project's application
  - E-R diagram for the application being replaced (not produced if the proposed system supports a completely new business function)
  - E-R diagram for the whole database from which the new application's data are extracted (show how the new application shares the contents of more widely used DBs)
  - E-R diagram for the whole database from which data for the application system being replaced is drawn

# Deliverables and Outcome (cont.)

- Second deliverable is a set of entries about data objects to be stored in repository or project dictionary.
  - Repository links data, process, and logic models of an information system.
  - Data elements included in the DFD must appear in the data model and vice versa.
  - Each data store in a process model must relate to business objects represented in the data model.

**Figure 9-3** Sample conceptual data model



- A SUPPLIER *sometimes* **supplies** ITEMs to the company.
- An ITEM is *always* **supplied** by one to four SUPPLIERs.
- A SHIPMENT is **sent by** one and only one SUPPLIER.
- A SUPPLIER **sends** zero or many SHIPMENTS.

# Gathering Information for Conceptual Data Modeling

- Two perspectives
  - Top-down
    - Data model is derived from an intimate understanding of the business.
  - Bottom-up
    - Data model is derived by reviewing specifications and business documents.

# Requirements Determination Questions for Data Modeling

- What are subjects/objects of the business?
  - ➔ Data entities and descriptions
- What unique characteristics distinguish between subjects/objects of the same type?
  - ➔ Primary keys
- What characteristics describe each subject/object?
  - ➔ Attributes and secondary keys
- How do you use the data?
  - ➔ Security controls and user access privileges

# Requirements Determination Questions for Data Modeling (cont.)

- Over what period of time are you interested in the data?
  - ➔ Cardinality and time dimensions
- Are all instances of each object the same?
  - ➔ Supertypes, subtypes, and aggregations
- What events occur that imply associations between objects?
  - ➔ Relationships and cardinalities
- Are there special circumstances that affect the way events are handled? Can the associations change over time? (an employee change department?)
  - ➔ Integrity rules, min & max cardinalities, time dimensions of data

# Introduction to Entity-Relationship (E-R) Modeling

- Entity-Relationship (E-R) Diagram
  - A detailed, logical representation of the entities, associations and data elements for an organization or business
- Notation uses three main constructs
  - Data entities
  - Relationships
  - Attributes

Association between the instances of one or more entity

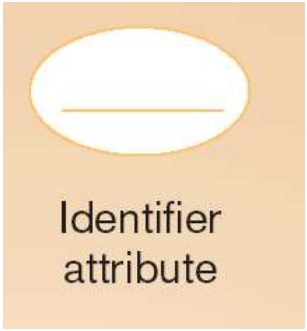

Person, place, object, event or concept about which data is to be maintained

**Entity type**: collection of entities with common characteristics

**Entity instance**: single entity

named property or characteristic of an entity

# Identifier Attributes




- Candidate key
  - Attribute (or combination of attributes) that uniquely identifies each instance of an entity type
- Identifier
  - A candidate key that has been selected as the unique identifying characteristic for an entity type

# Identifier Attributes (cont.)

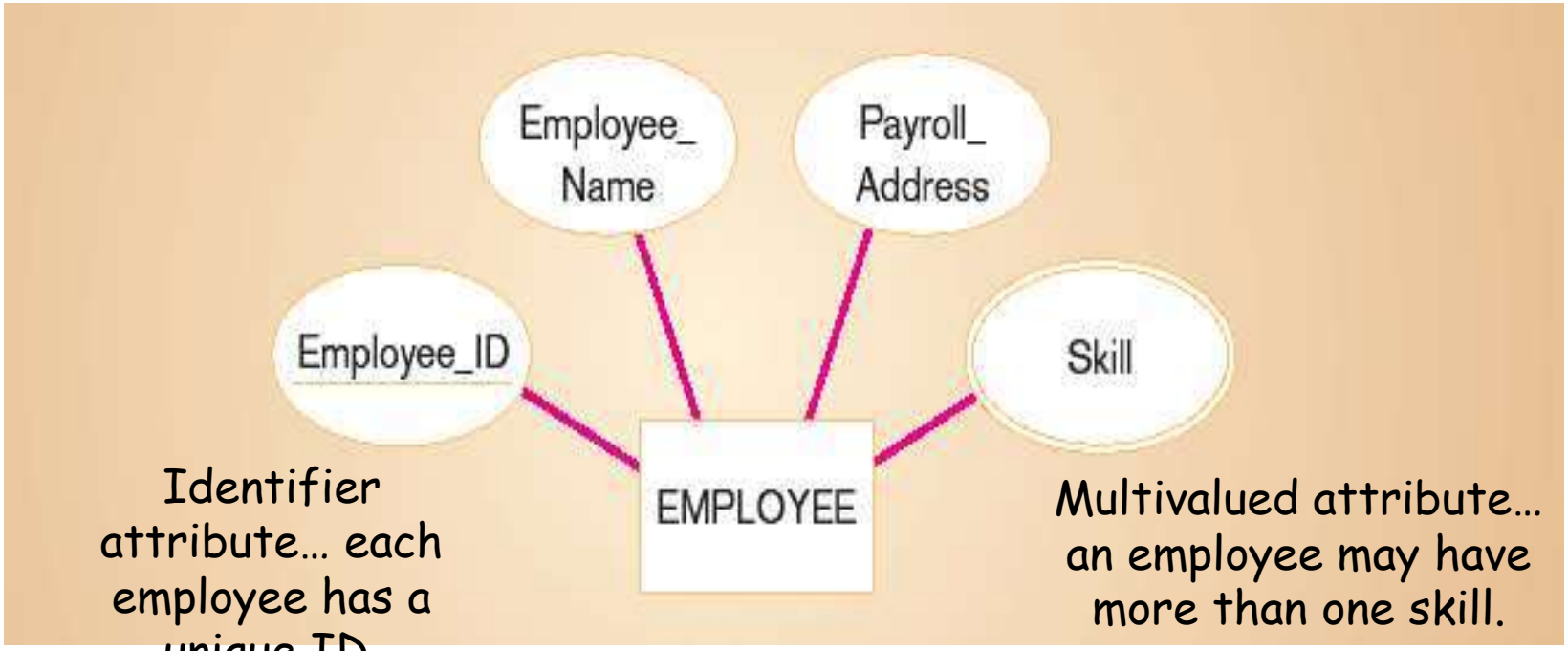

- Selection rules for an identifier
  1. Choose a candidate key that will not change its value.
  2. Choose a candidate key that will never be null.
  3. Avoid using intelligent keys.
  4. Consider substituting single value surrogate keys for large composite keys.

# Multivalued Attributes


- An attribute that may take on more than one value *for each entity instance*
- Showing multi-valued attributes:
  - double-lined ellipse in ERD
  - Separating the repeating data into another entity: called a weak (or attribute) entity

# Entity and Attribute Example

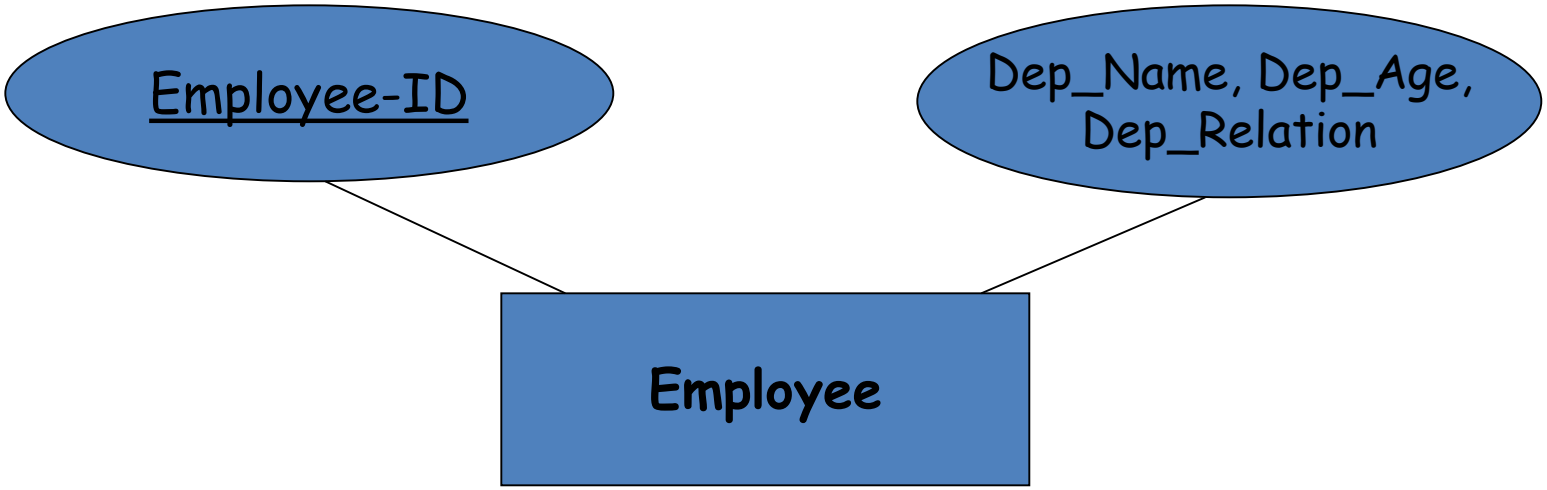
Simple attributes



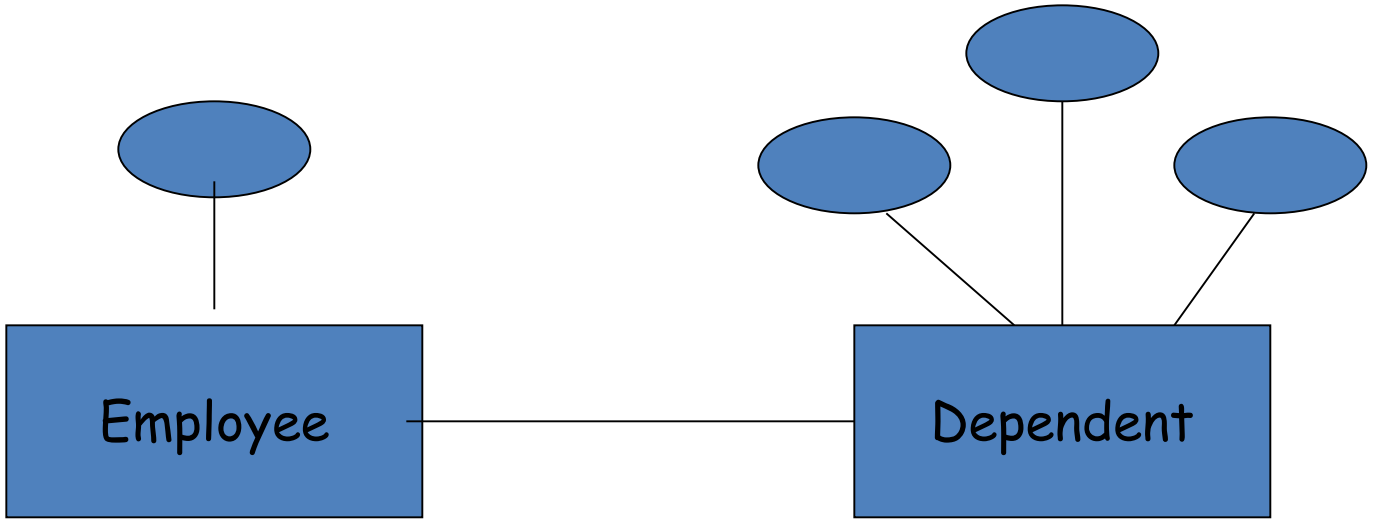

Identifier attribute… each employee has a unique ID.

Multivalued attribute… an employee may have more than one skill.

- Repeating Group



# Weak entity



# Degree of Relationship

- Degree: number of entity types that participate in a relationship
- Three cases
  - **Unary:** between two instances of one entity type
  - **Binary:** between the instances of two entity types
  - **Ternary:** among the instances of three entity types

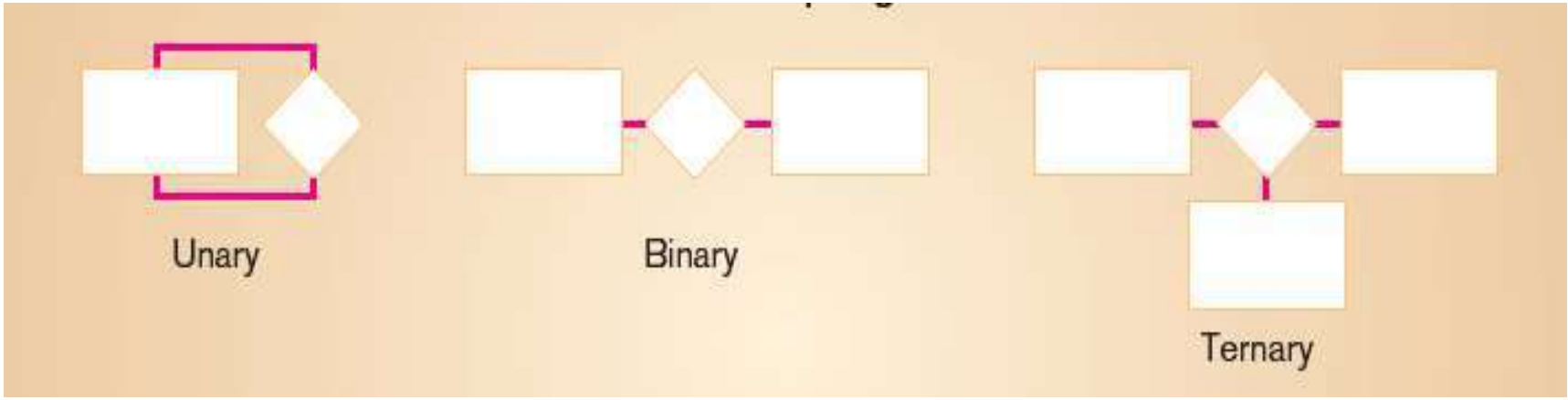
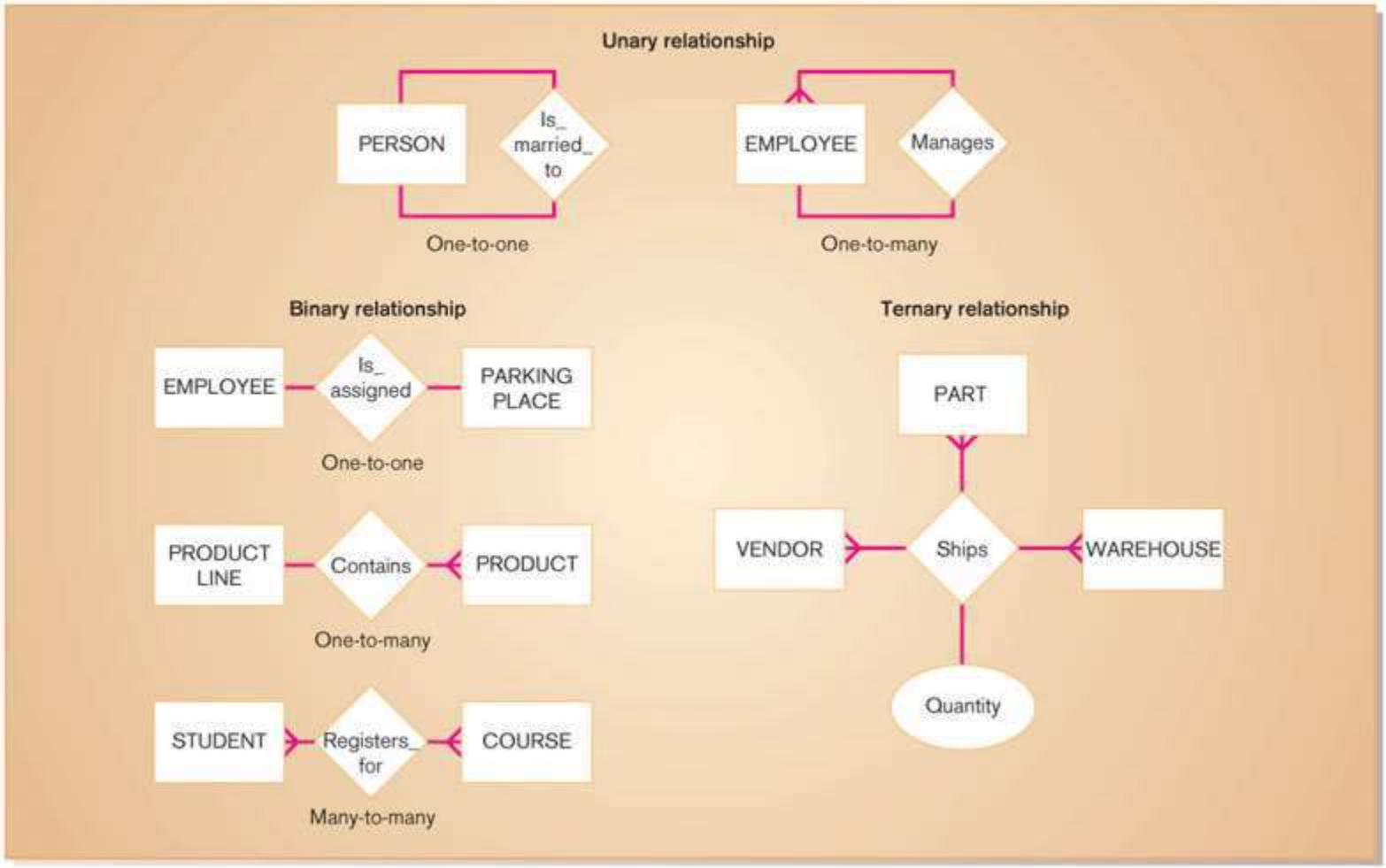


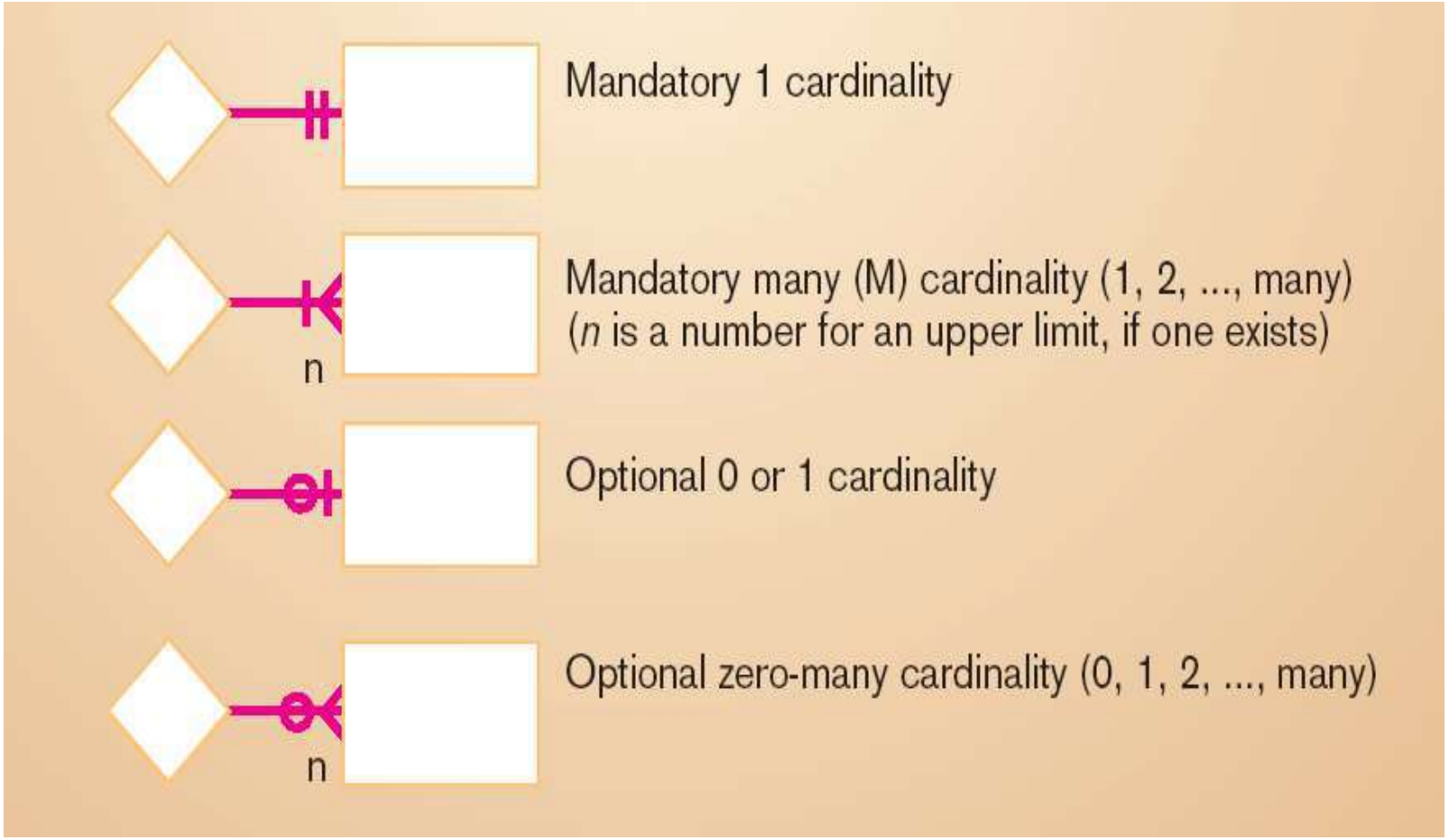
**Figure 9-6** Example relationships of different degrees



# Cardinality

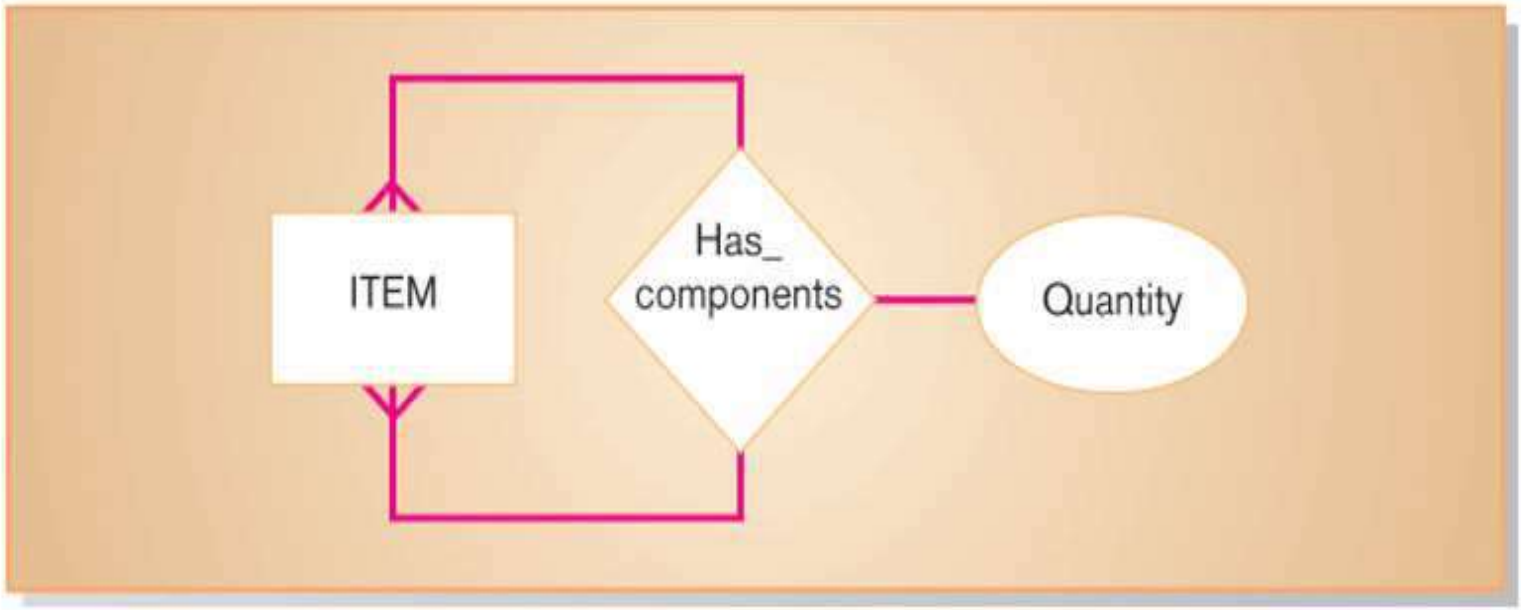
- The number of instances of entity B that can or must be associated with each instance of entity A
- Minimum Cardinality
  - The minimum number of instances of entity B that may be associated with each instance of entity A
- Maximum Cardinality
  - The maximum number of instances of entity B that may be associated with each instance of entity A
- Mandatory vs. Optional Cardinalities
  - Specifies whether an instance must exist or can be absent in the relationship

# Cardinality Symbols



# Unary Relationship Example

**Figure 9-7a** Bill-of-materials unary relationship - Many-to-many relationship



# Binary Relationship Examples

**Figure 9-8a** Examples of cardinalities in relationships - Mandatory cardinalities

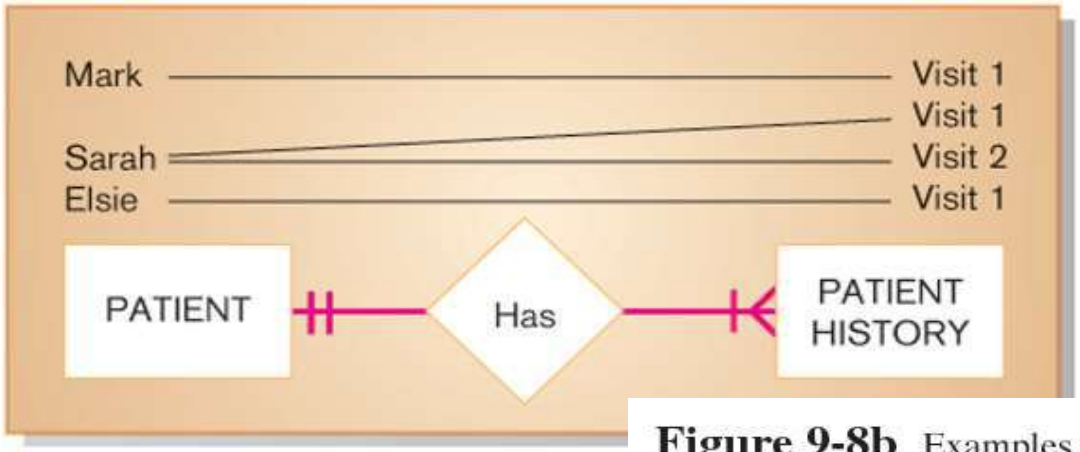
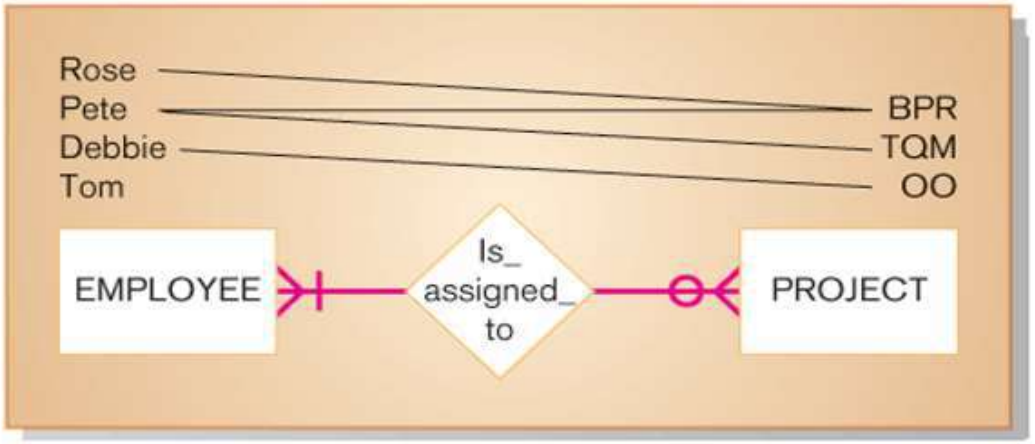


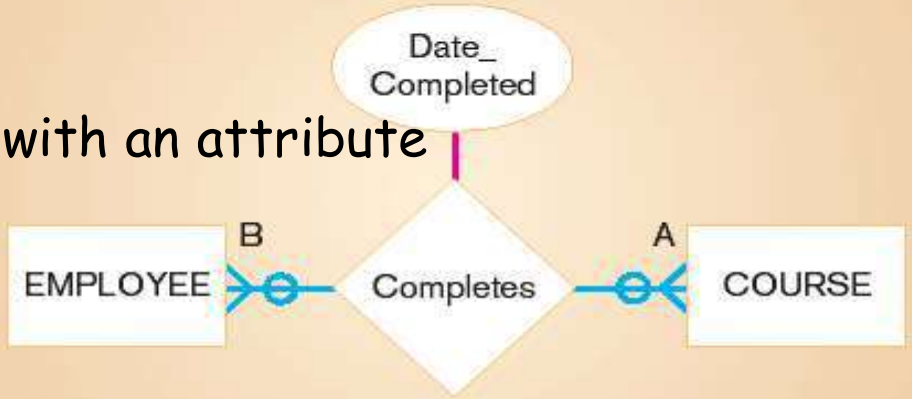

**Figure 9-8b** Examples of cardinalities in relationships - One optional, one mandatory cardinality



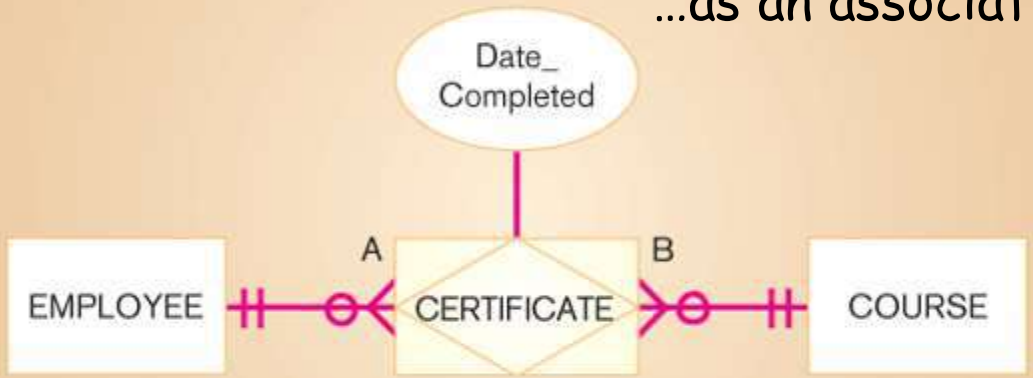
# Associative Entities

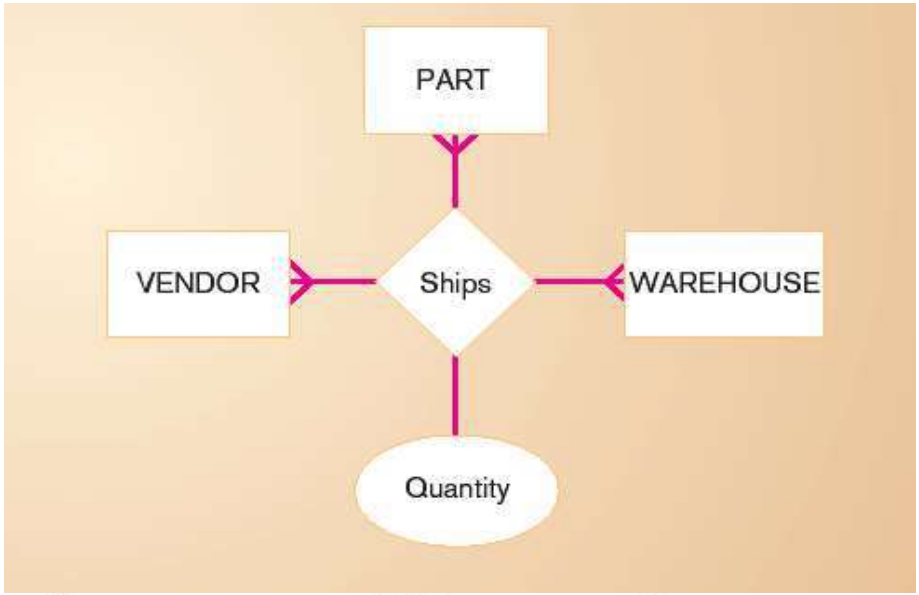

- An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances
- An associative entity is:
  - An entity
  - A relationship
- This is the preferred way of illustrating a relationship with attributes

A relationship with an attribute

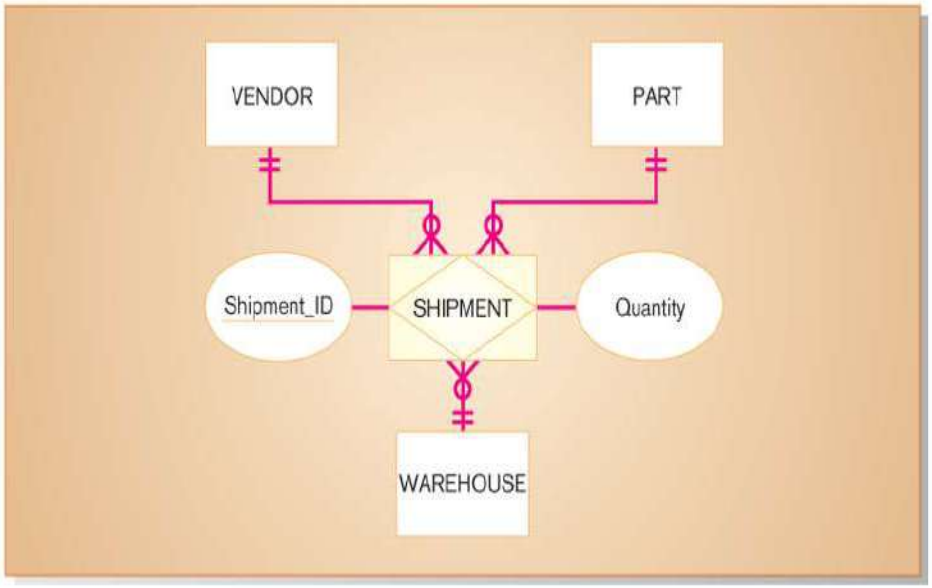



…as an associative entity

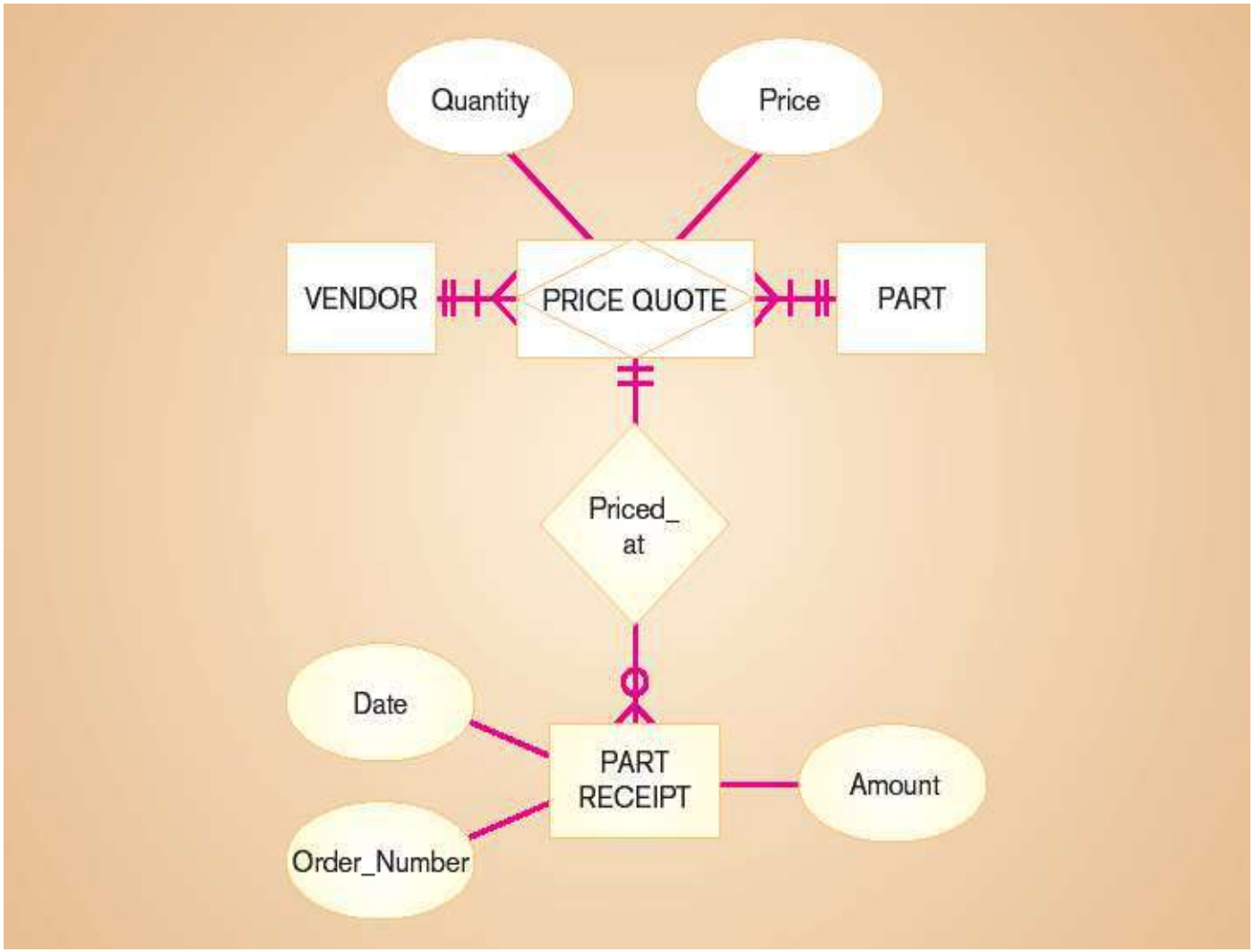





Ternary relationship




…as an associative entity




A relationship that itself is related to other entities via another relationship *must* be represented as an associative entity. (it is not a ternary relationship)

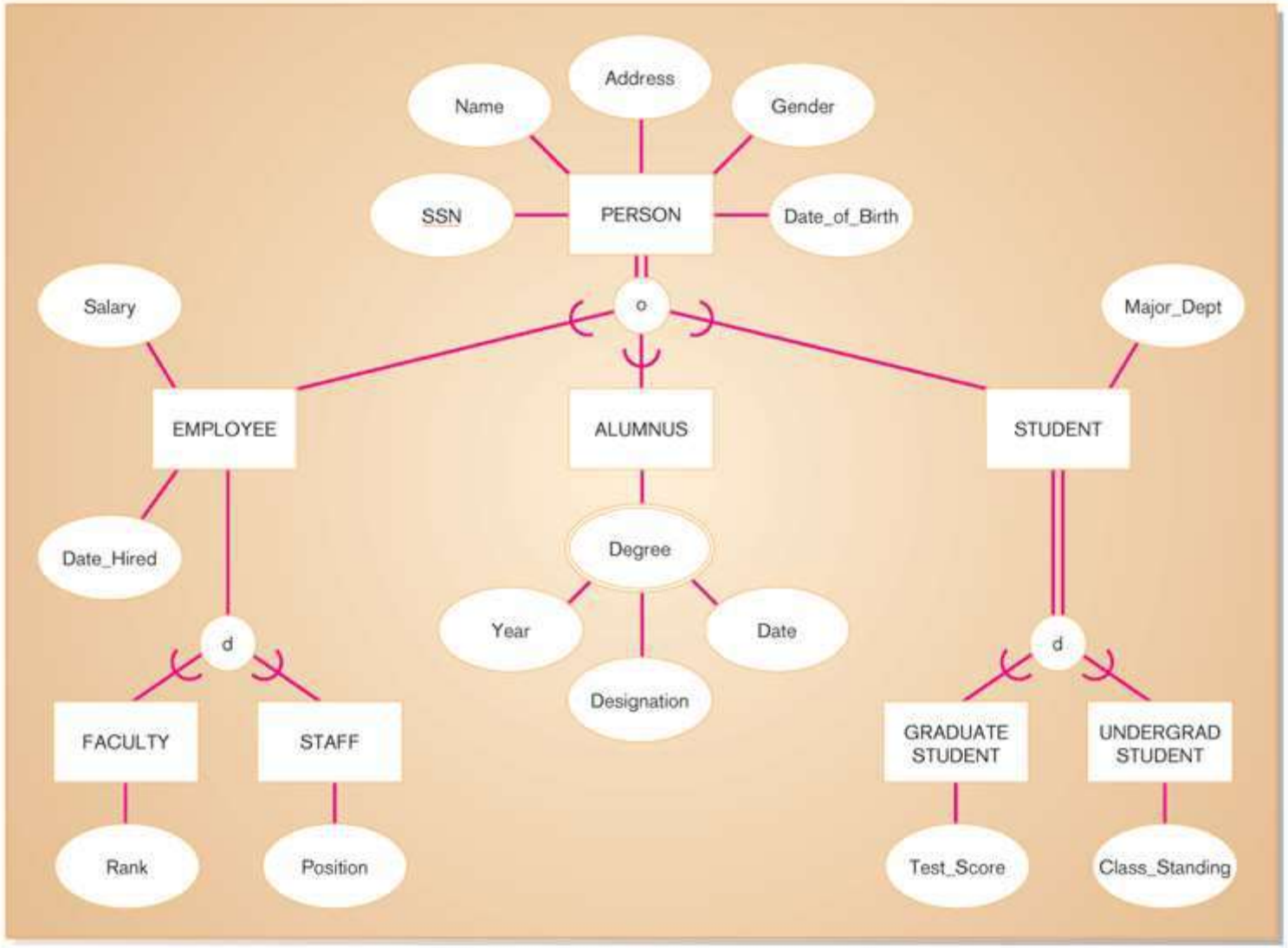
# Supertypes and Subtypes

- Subtype: a subgrouping of the entities in an entity type that shares common attributes or relationships distinct from other subtypes
- Supertype: a generic entity type that has a relationship with one or more subtype

# Rules for Supertype/Subtypes Relationships

- Total specialization: an entity instance of the supertype must be an instance of one of the subtypes
- Partial specialization: an entity instance of the supertype may or may not be an instance of one of the subtypes
- Disjoint: an entity instance of the supertype can be an instance of only one subtype
- Overlap: an entity instance of the supertype may be an instance of multiple subtypes

**Figure 9-12** Example of supertype/subtype heirarchy



# Business Rules

- Specifications that preserve the integrity of the logical data model
- Four types
  - Entity integrity: unique, non-null identifiers
  - Referential integrity constraints: rules governing relationships
  - Domains: valid values for attributes
  - Triggering operations: other business rules protect the validity of attribute values

# Domains

- The set of all data types and ranges of values that an attribute can assume
- Several advantages
  1. Verify that the values for an attribute are valid
  2. Ensure that various data manipulation operations are logical
  3. Help conserve effort in describing attribute characteristics

# Triggering Operations

- An assertion or rule that governs the validity of data manipulation operations such as insert, update and delete
- Components:
  - User rule: statement of the business rule to be enforced by the trigger
  - Event: data manipulation operation that initiates the operation
  - Entity Name: name of entity being accessed or modified
  - Condition: condition that causes the operation to be triggered
  - Action: action taken when the operation is triggered

**Figure 9-13b** Examples of business rules - Typical domain definitions

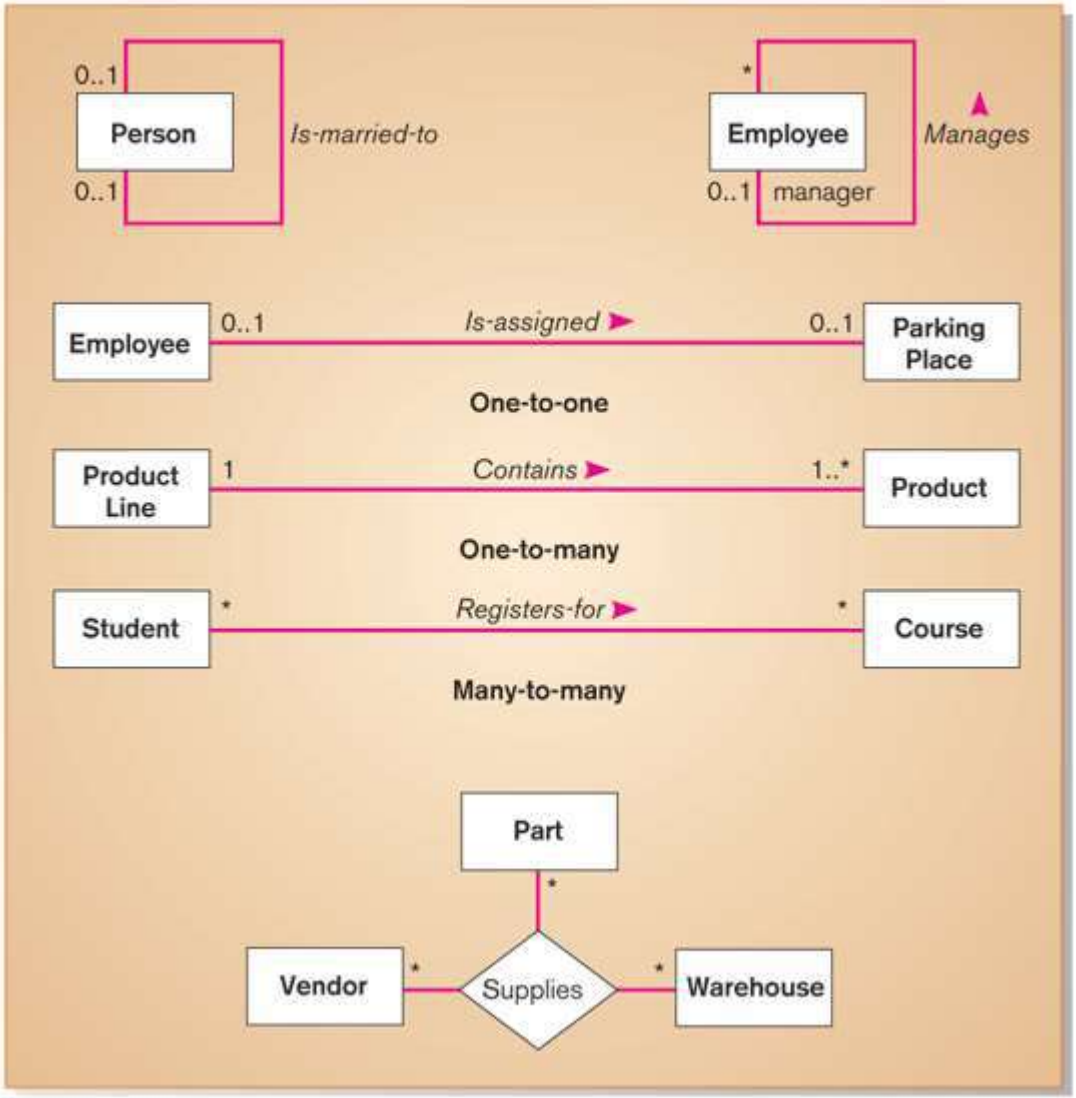
Name: Account_Number
Meaning: Customer account number in bank
Data type: Character
Format: nnn-nnnn
Uniqueness: Must be unique
Null support: Non-null

Name: Amount
Meaning: Dollar amount of transaction
Data type: Numeric
Format: 2 decimal places
Range: 0-10,000
Uniqueness: Nonunique
Null support: Non-null

**Figure 9-13c** Examples of business rules - Typical triggering operation

User rule: WITHDRAWAL Amount may not exceed ACCOUNT Balance
Event: Insert
Entity Name: WITHDRAWAL
Condition: WITHDRAWAL Amount > ACCOUNT Balance
Action: Reject the insert transaction

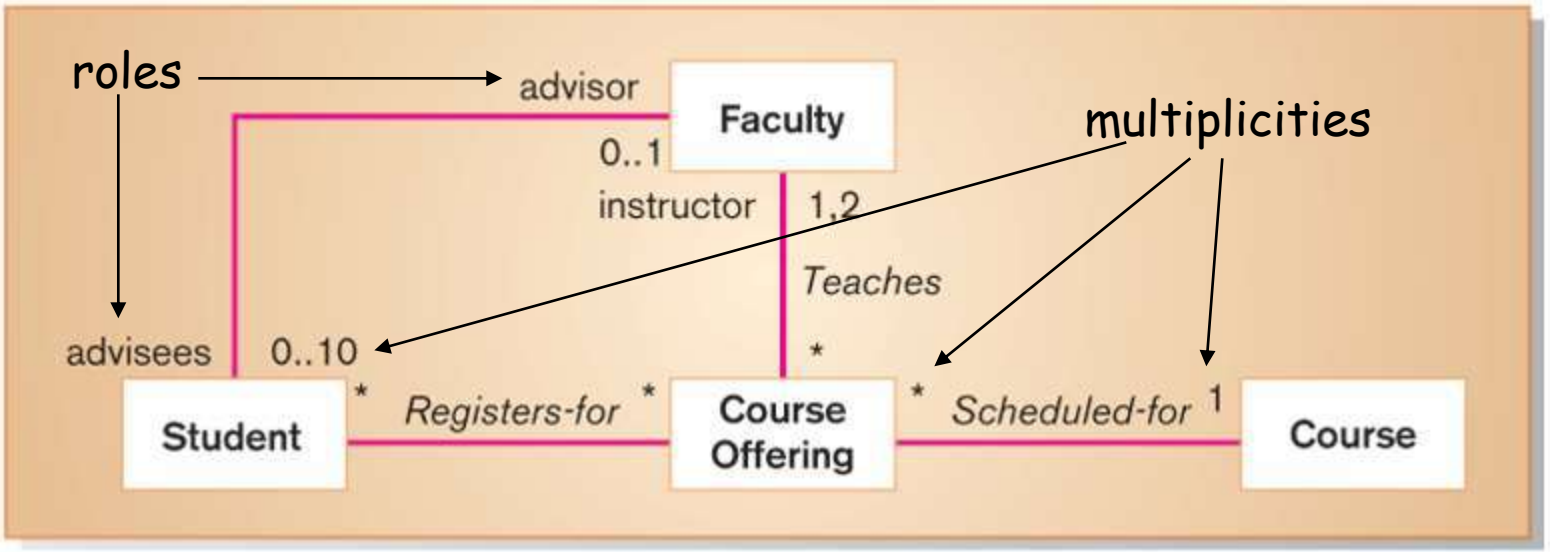
**Figure 9-16** Examples of association relationships of different degrees




UML associations are analogous to E-R relationships.

UML multiplicities are analogous to E-R cardinalities.

**Figure 9-17** Example of binary association replationships




Multiplicity notation:

0..10 means minimum of 0 and maximum of 10

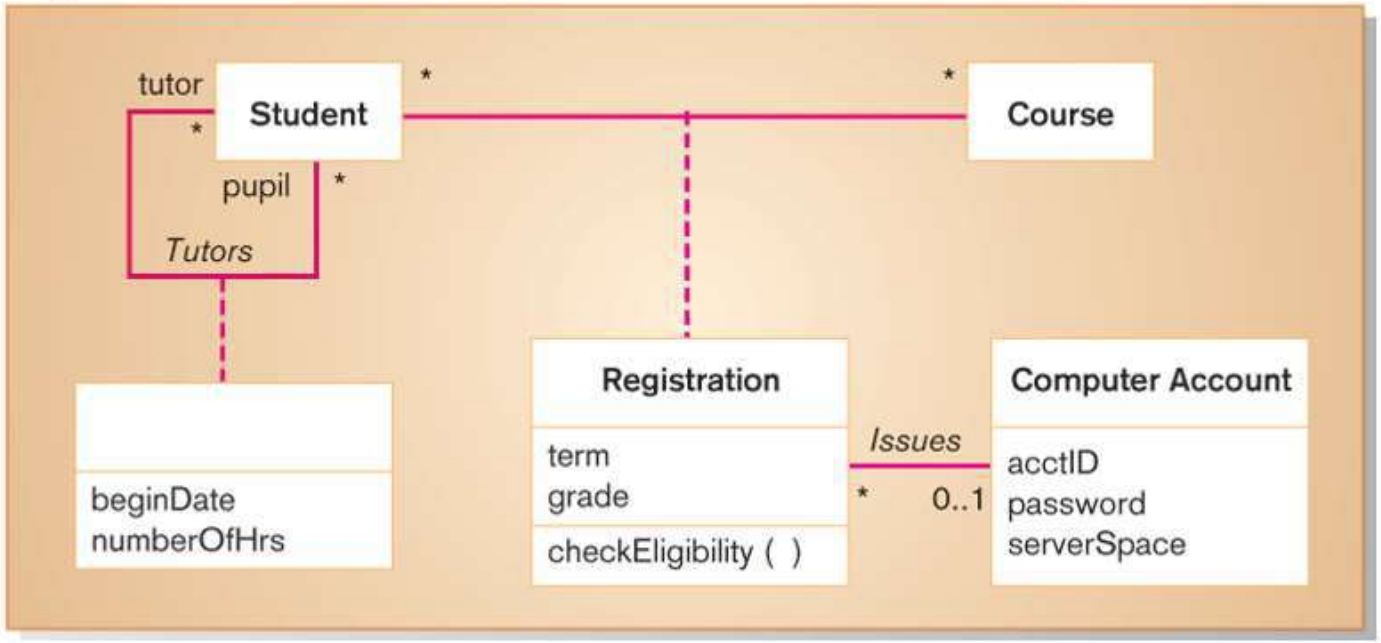
1, 2 means can be either 1 or 2

* means any number

# Association Class

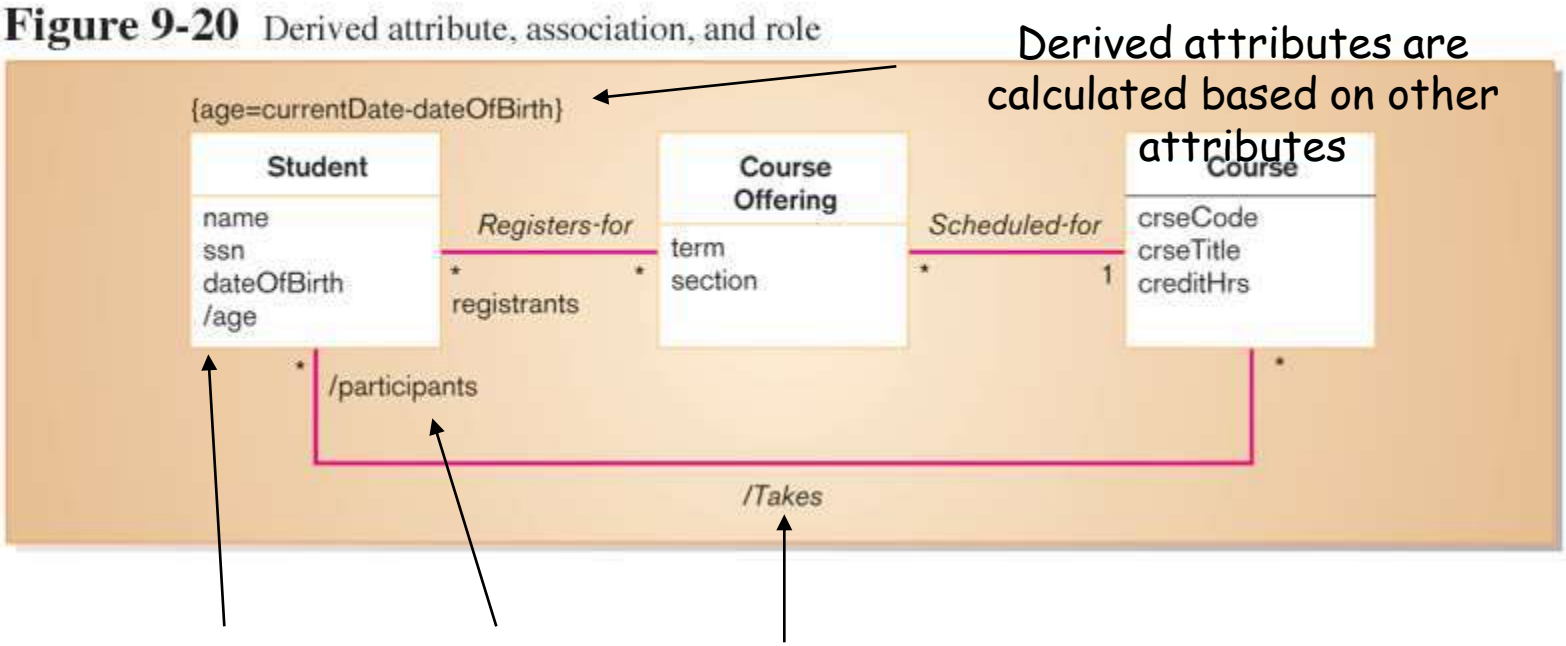
- An association with its own attributes, operations, or relationships

**Figure 9-18** Class diagram showing association classes




UML association classes are analogous to E-R associative entities.

# Derived Attributes, Associations, and Roles



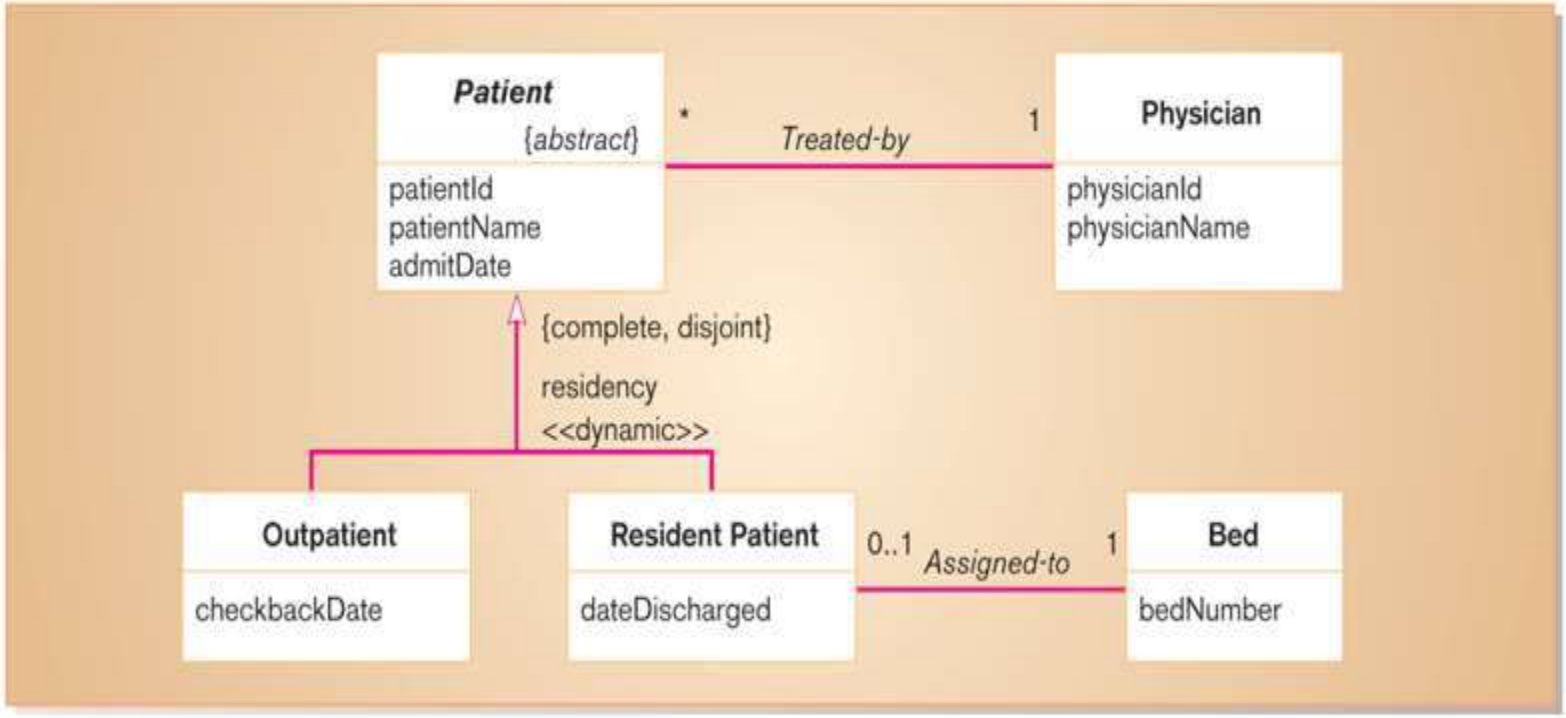

Derived attributes are calculated based on other attributes

Derived items are represented with a slash (/).

# Generalization

- Superclass-subclass relationships
- Subclass inherits attributes, operations, and associations of the superclass
- Types of superclasses
  - Abstract: cannot have any direct instances
  - Concrete: can have direct instances

**Figure 9-21** Example of generalization, inheritance, and constraints

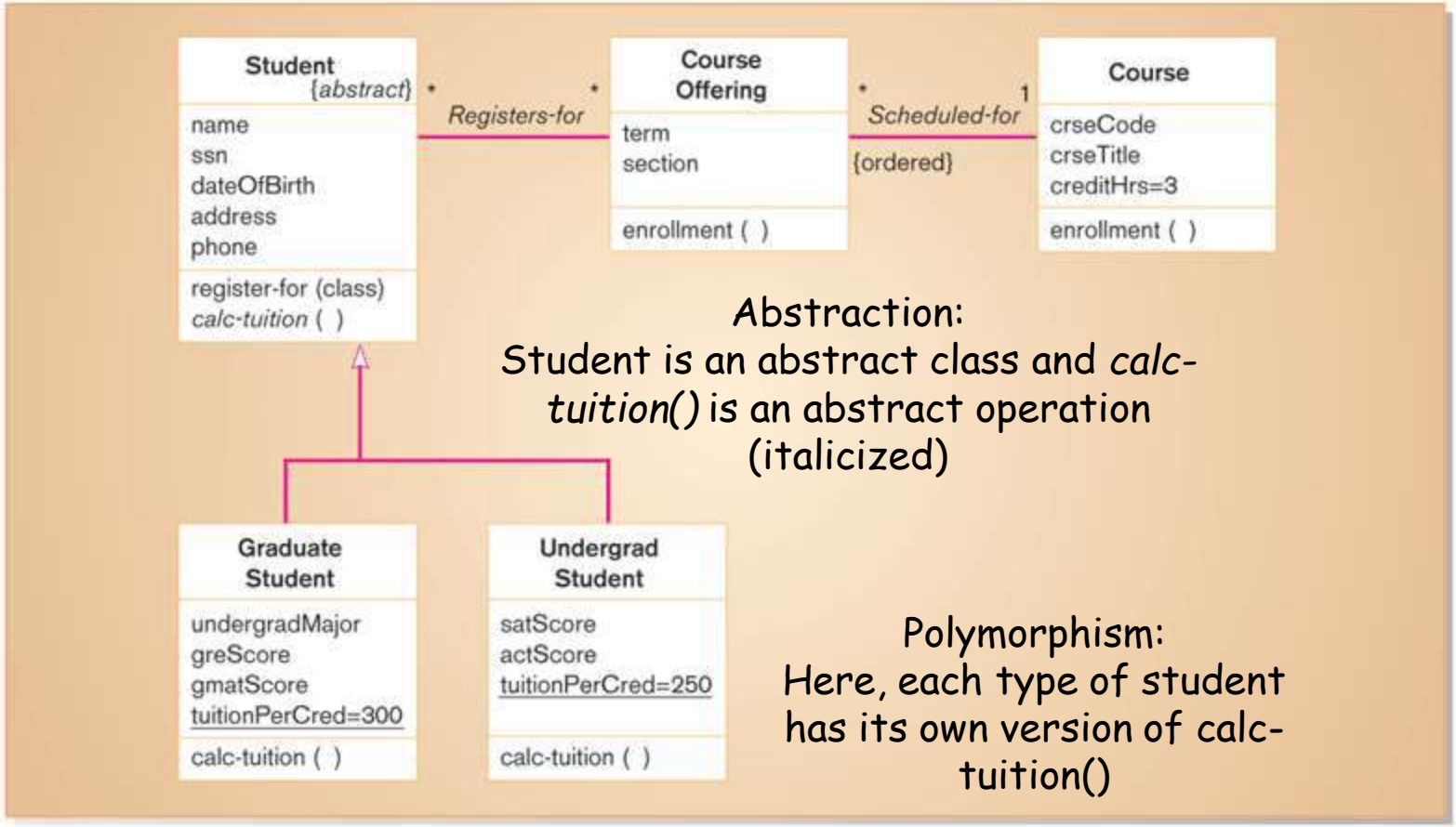



Generalization and inheritance implemented via superclass/subclasses in UML, supertypes/subtypes in E-R

# Polymorphic Operations

- The same operation may apply to two or more classes in different ways
- Abstract operations
  - defined in abstract classes
  - defined the protocol, but not the implementation of an operation
- Methods
  - the implementation of an operation

**Figure 9-22** Polymorphism, abstract operation, class-scope attribute, and ordering

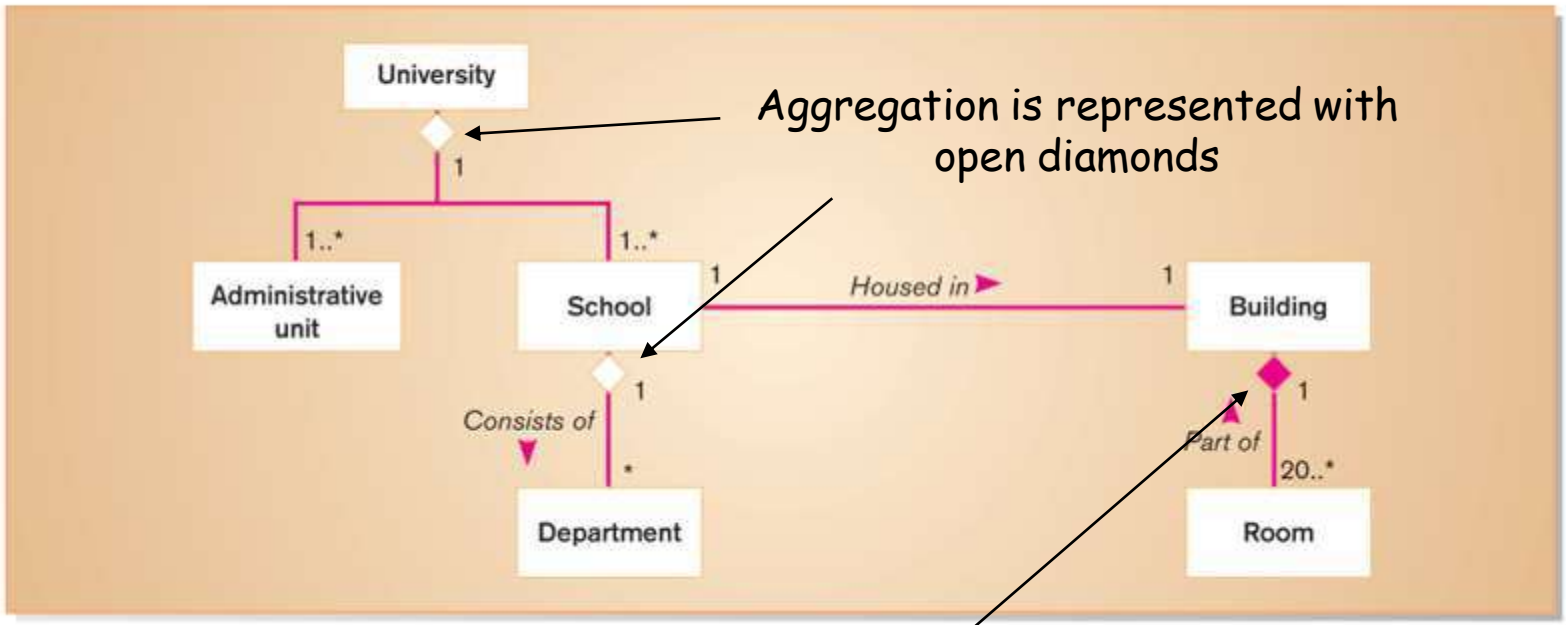



Class scope:
tuitionPerCred is a class-wide attribute

# Aggregation and Composition

- Aggregation
  - A part-of relationship between a component and an aggregate object
- Composition
  - An aggregation in which the part object belongs to only one aggregate object and lives and dies with the aggregate object

**Figure 9-23** Aggregation and composition



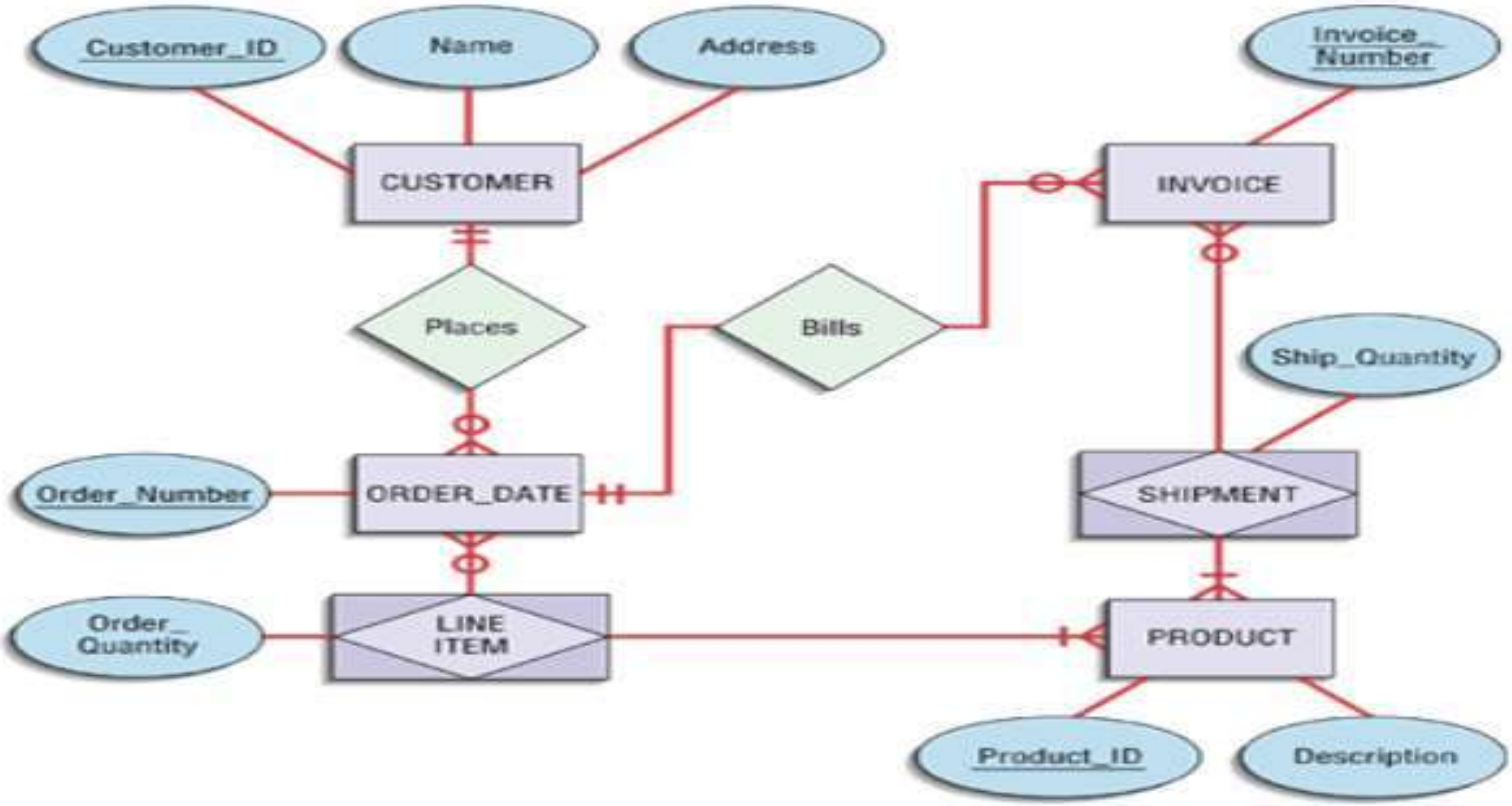

Aggregation is represented with open diamonds

Composition is represented with filled diamonds

# Transferring E-R Diagram into Relations

# Entity Sets to Tables

- Each attribute of the E. S. becomes an attribute of the table




Relations:

CUSTOMER(Customer_ID,Name,Address)
PRODUCT(Product_ID,Description)
ORDER(Order_Number,Customer_ID,Order_Date)
LINE ITEM(Order_Number,Product_ID,Order_Quantity)
INVOICE(Invoice_Number,Order_Number)
SHIPMENT(Invoice_Number,Product_ID,Ship_Quantity)

# Transforming E-R Diagrams into Relations

- It is useful to transform the conceptual data model into a set of normalized relations
- Steps
  1. Represent entities
  2. Represent relationships
  3. Normalize the relations
  4. Merge the relations

# Transforming E-R Diagrams into Relations

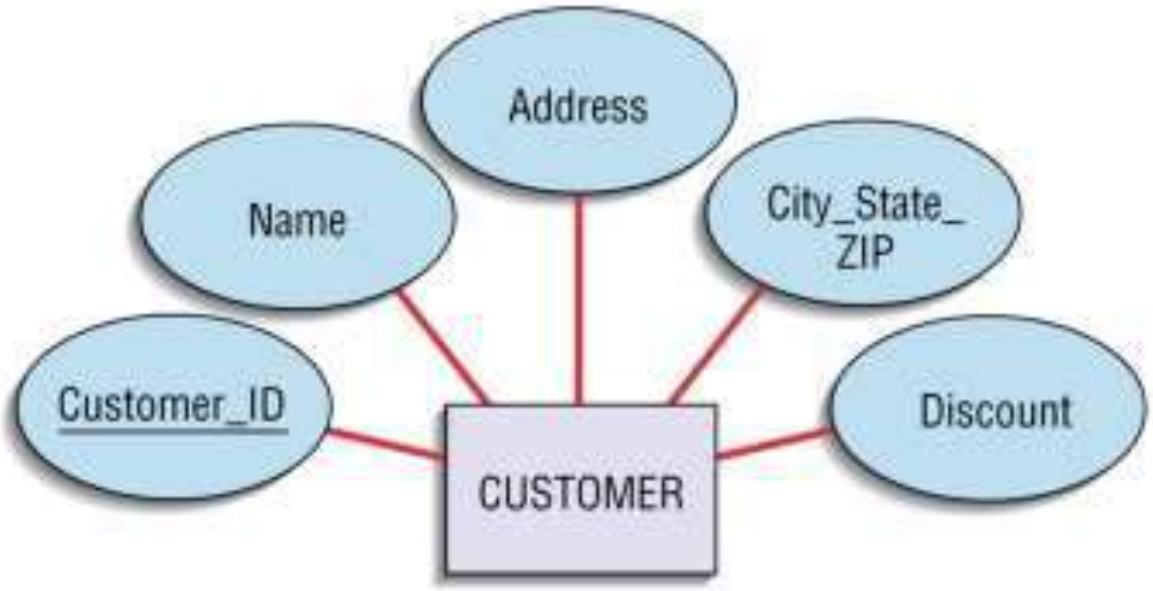
- In translating a relationship set to a relation, attributes of the relation must include:
  - The primary key for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes of the relationship set

– ving two conditions

a. The value of the key must uniquely identify every row in the relation

b. The key should be nonredundant

9.52



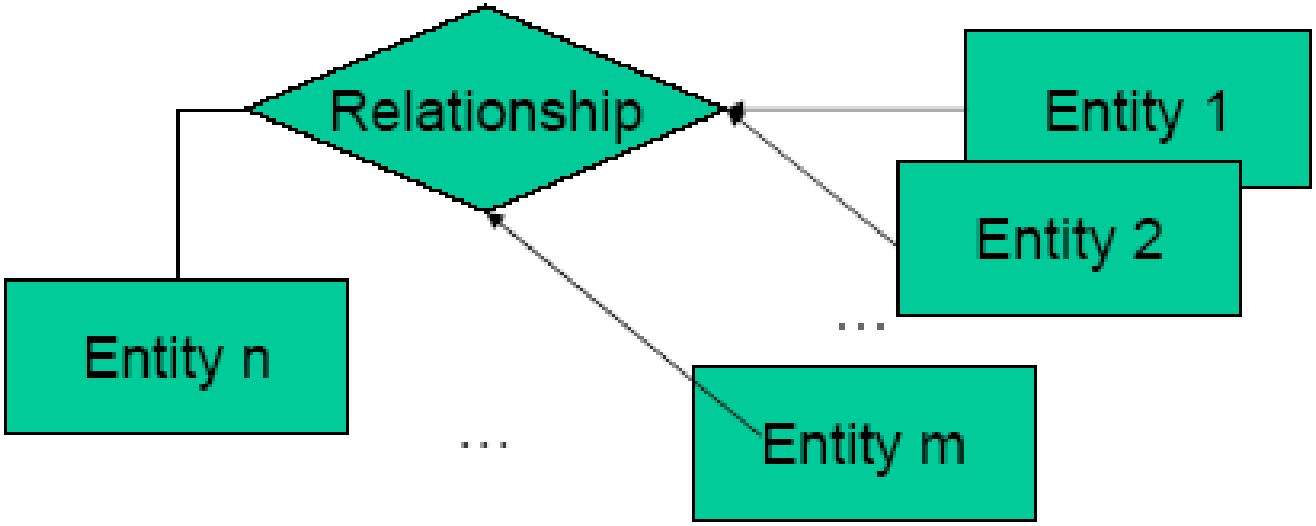

CUSTOMER

| Customer_ID | Name | Address | City_State_ZIP | Discount |
|---|---|---|---|---|
| 1273 | Contemporary Designs | 123 Oak St. | Austin, TX 28384 | 5% |
| 6390 | Casual Corner | 18 Hoosier Dr. | Bloomington, IN 45821 | 3% |

9.53

# General case

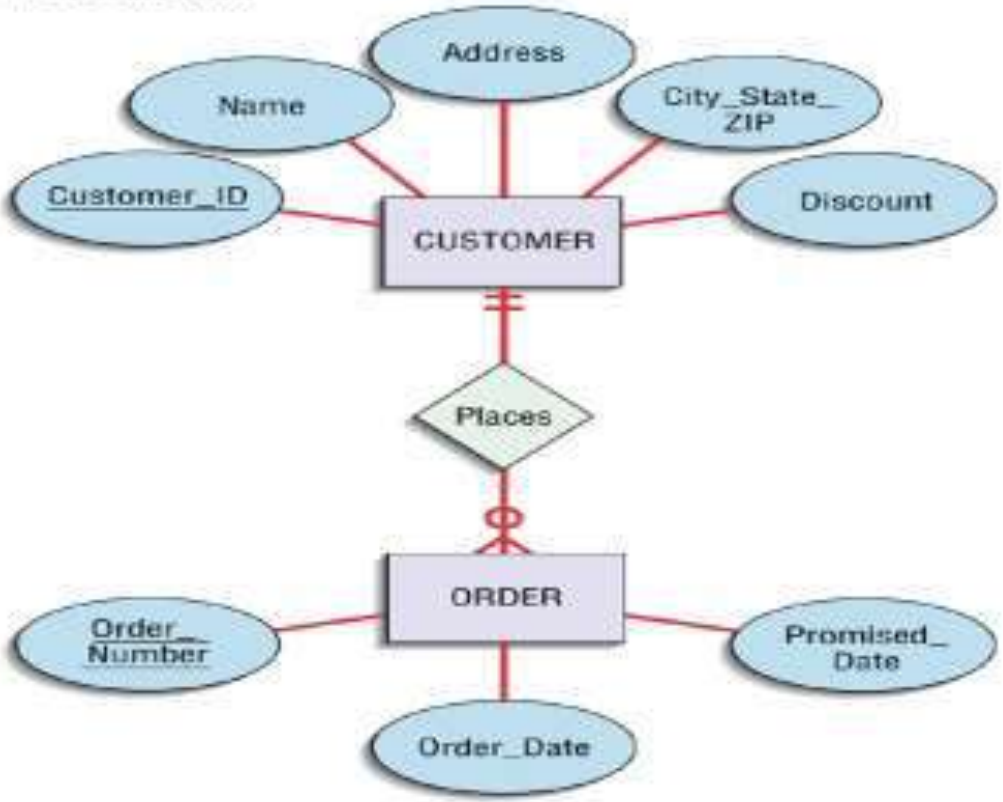
- A relationship with n entity sets and some m of them have one-to-one or one-to-many constraints (arrows in the E/R diagram)




- The key for any of m entity sets is a candidate key for the relation
- One of them should be designated as the primary key.

# Transforming E-R Diagrams into Relations

2. Represent Relationships
   - Binary 1:N Relationships
     - Add the primary key attribute (or attributes) of the entity on the one side of the relationship as a foreign key in the relation on the right side
     - The one side *migrates* to the many side




CUSTOMER

| Customer_ID | Name | Address | City_State_ZIP | Discount |
|---|---|---|---|---|
| 1273 | Contemporary Designs | 123 Oak St. | Austin, TX 28384 | 5% |
| 6390 | Casual Corner | 18 Hoosier Dr. | Bloomington, IN 45821 | 3% |

ORDER

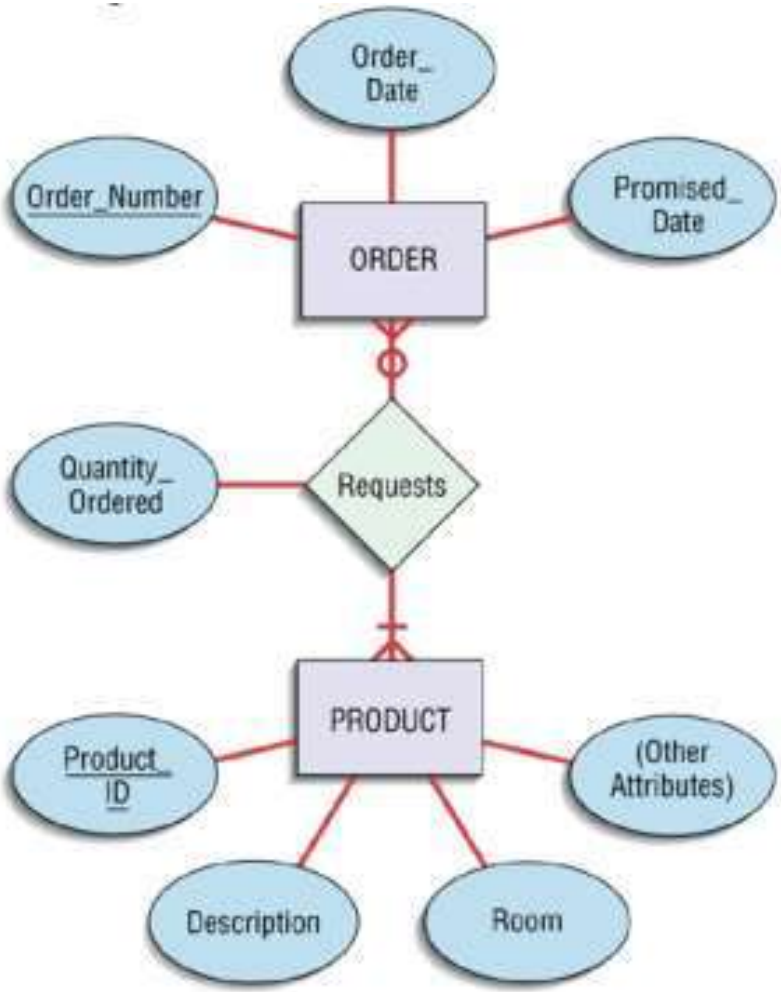
| Order_Number | Order_Date | Promised_Date | Customer_ID |
|---|---|---|---|
| 57194 | 3/15/0X | 3/28/0X | 6390 |
| 63725 | 3/17/0X | 4/01/0X | 1273 |
| 80149 | 3/14/0X | 3/24/0X | 6390 |

# Transforming E-R Diagrams into Relations

- Binary or Unary 1:1
  - Three possible options
    a. Add the primary key of A as a foreign key of B
    b. Add the primary key of B as a foreign key of A
    c. Both

# Transforming E-R Diagrams into Relations

2. Represent Relationships (continued)
   - Binary and higher M:N relationships
     - Create another relation and include primary keys of all relations as primary key of new relation




ORDER

| Order_Number | Order_Date | Promised_Date |
|---|---|---|
| 61384 | 2/17/2002 | 3/01/2002 |
| 62009 | 2/13/2002 | 2/27/2002 |
| 62807 | 2/15/2002 | 3/01/2002 |

ORDER LINE

| Order_Number | Product_ID | Quantity_ Ordered |
|---|---|---|
| 61384 | M128 | 2 |
| 61384 | A261 | 1 |

PRODUCT

| Product_ID | Description | (Other Attributes) |
|---|---|---|
| M128 | Bookcase | — |
| A261 | Wall unit | — |
| R149 | Cabinet | — |

9.59

# Transforming E-R Diagrams into Relations

- Unary 1:N Relationships
  - Relationship between instances of a single entity type
  - Utilize a recursive foreign key
    - A foreign key in a relation that references the primary key values of that same relation
- Unary M:N Relationships
  - Create a separate relation
  - Primary key of new relation is a composite of two attributes that both take their values from the same primary key

9.60

**Figure 9.13b** Two Unary Relations — Bill-of-Materials Structure (M:N)

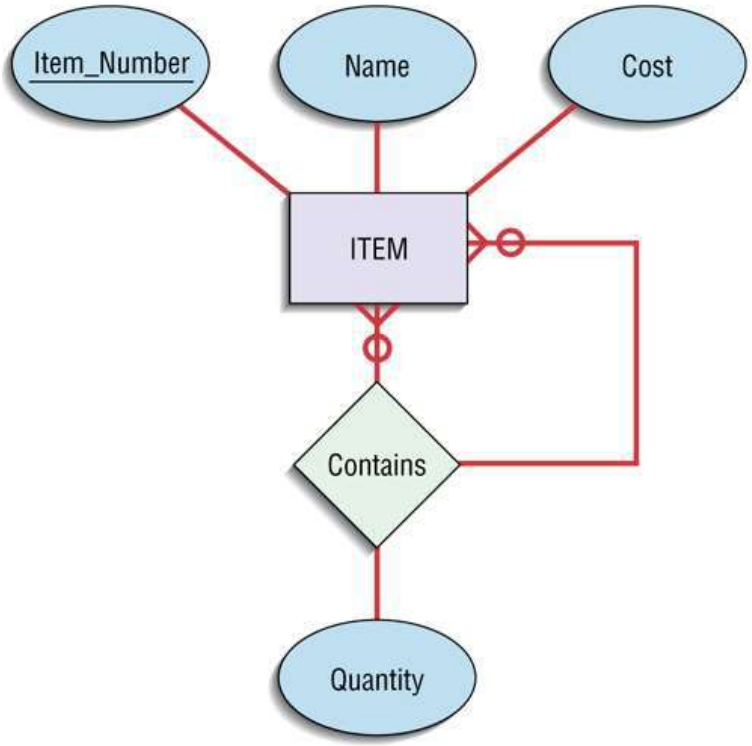
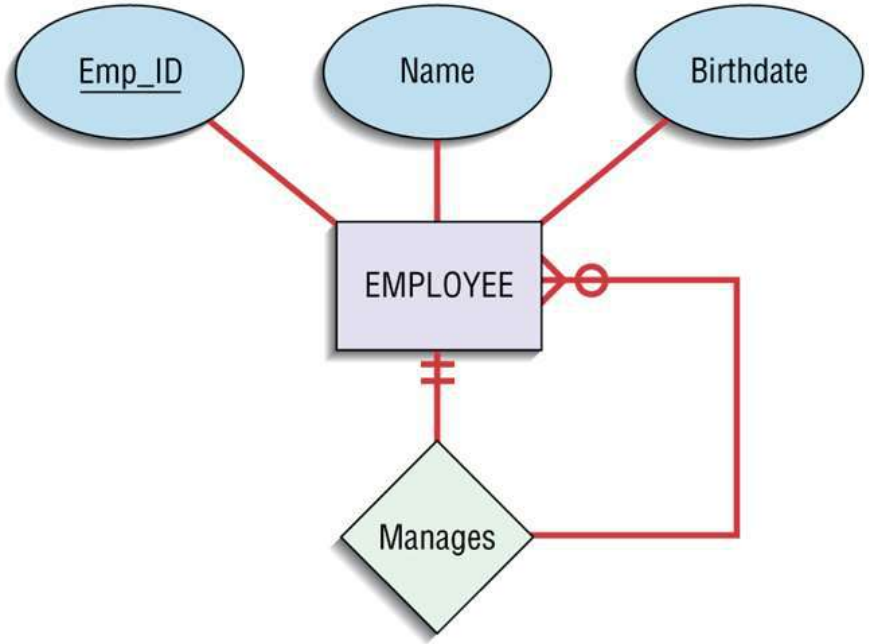


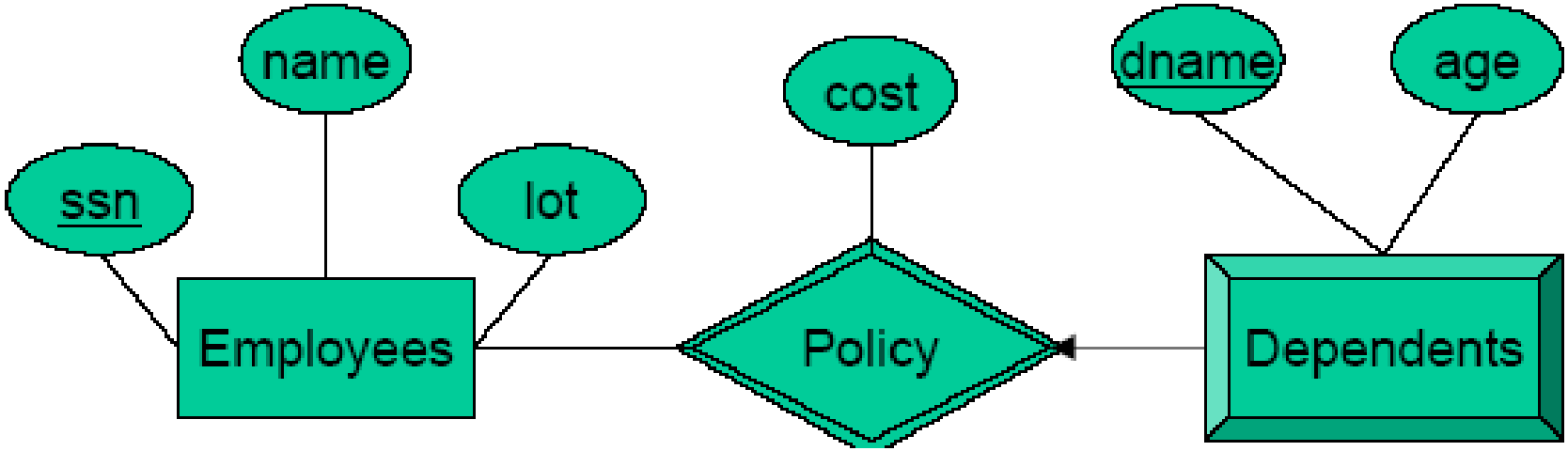

**Figure 9-13a** Two Unary Relations — EMPLOYEE with Manages Relationship (1:N)




9.61

# Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.




9.62

# Primary Key Constraints

- A set of fields is a *key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Primary key can not have null value

# Foreign Keys, Referential Integrity

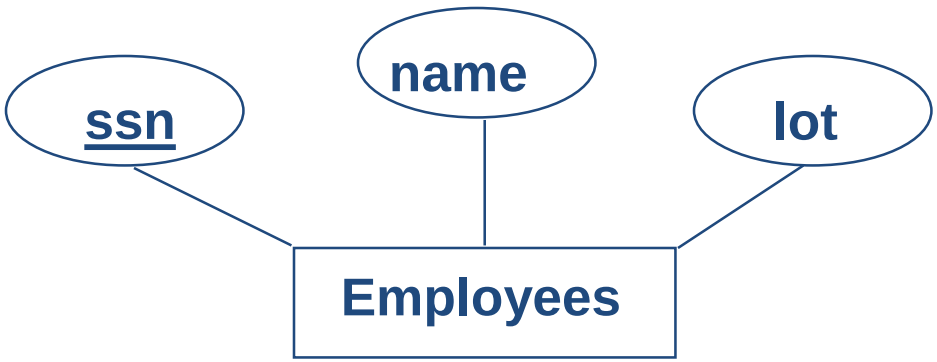
- *Foreign key* : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    - Links in HTML!

# Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)
- Similar if primary key of Students tuple is updated.

# Logical DB Design: ER to Relational

- Entity sets to tables.




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

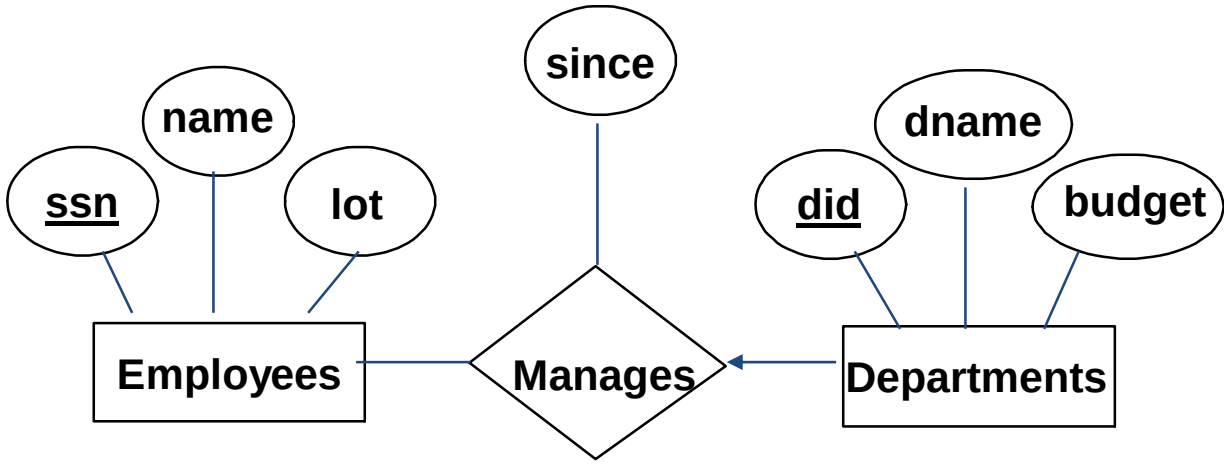
# Relationship Sets to Tables

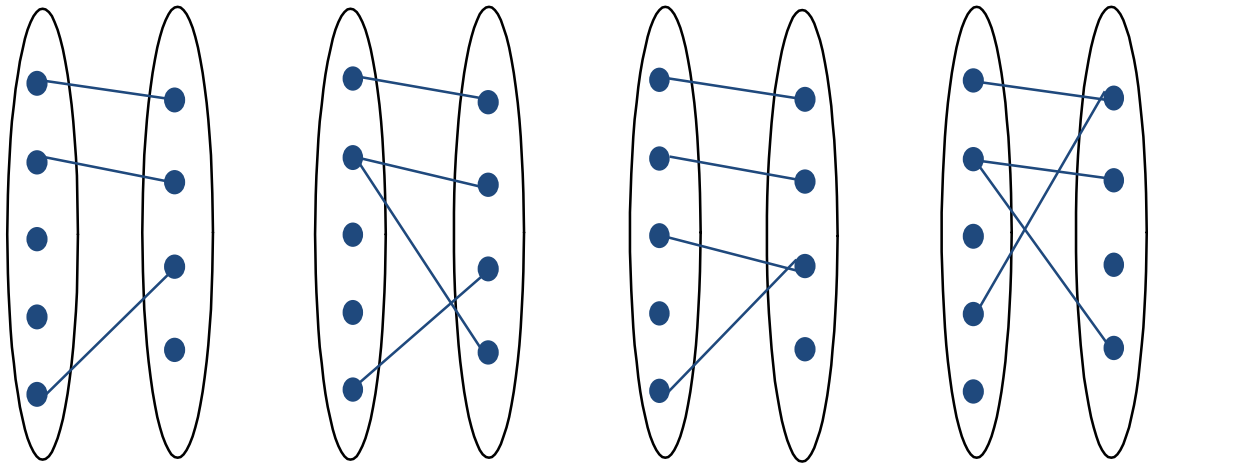
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms a *superkey* for the relation.
  - All descriptive attributes.

```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
     REFERENCES Employees,
  FOREIGN KEY (did)
     REFERENCES Departments)
```

# Review: Key Constraints

- Each dept has at most one manager, according to the *key constraint* on Manages.




*Translation to relational model?*

# Translating ER Diagrams with Key Constraints

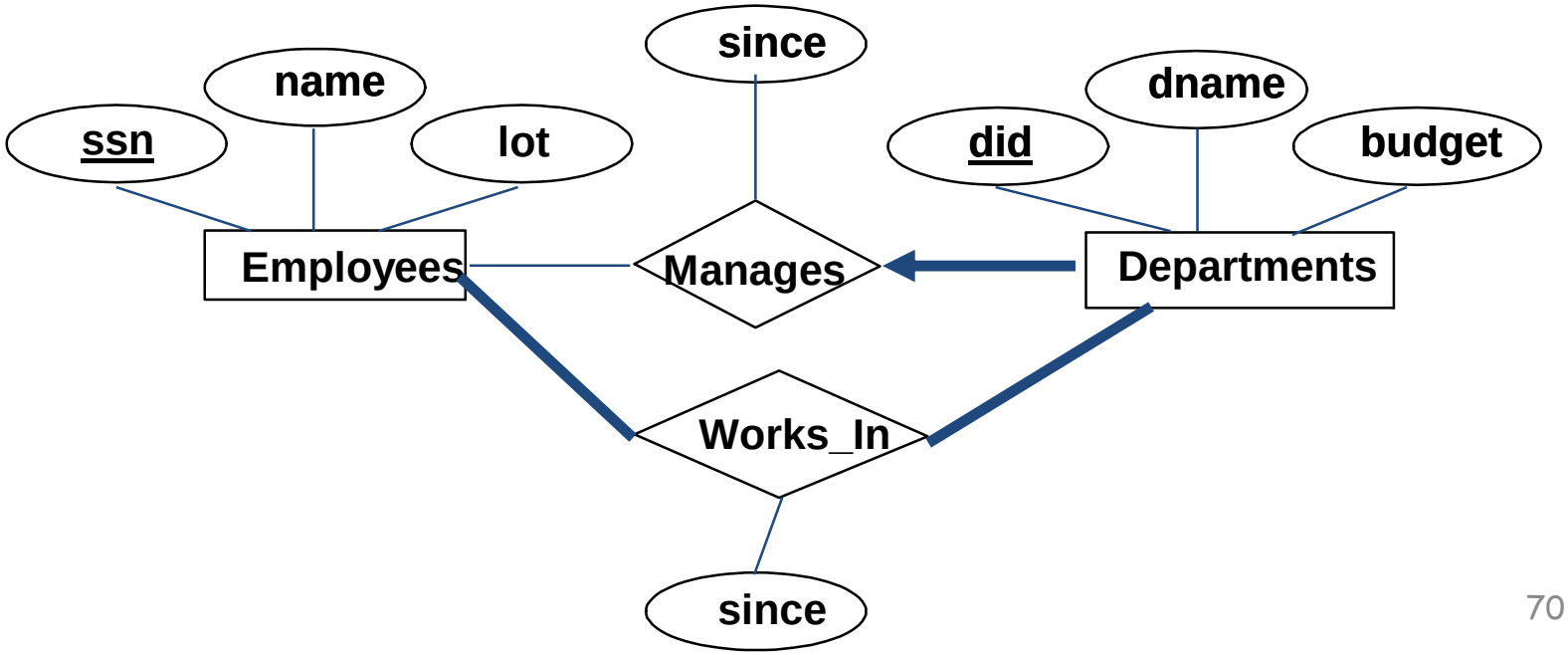
- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.
- Since each department has a unique manager, we could instead combine Manages and Departments.

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

```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname CHAR(20),
  budget REAL,
  ssn CHAR(11),
  since DATE,
  PRIMARY KEY (did),
  FOREIGN KEY (ssn) REFERENCES Employees)
```

# Review: Participation Constraints

- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



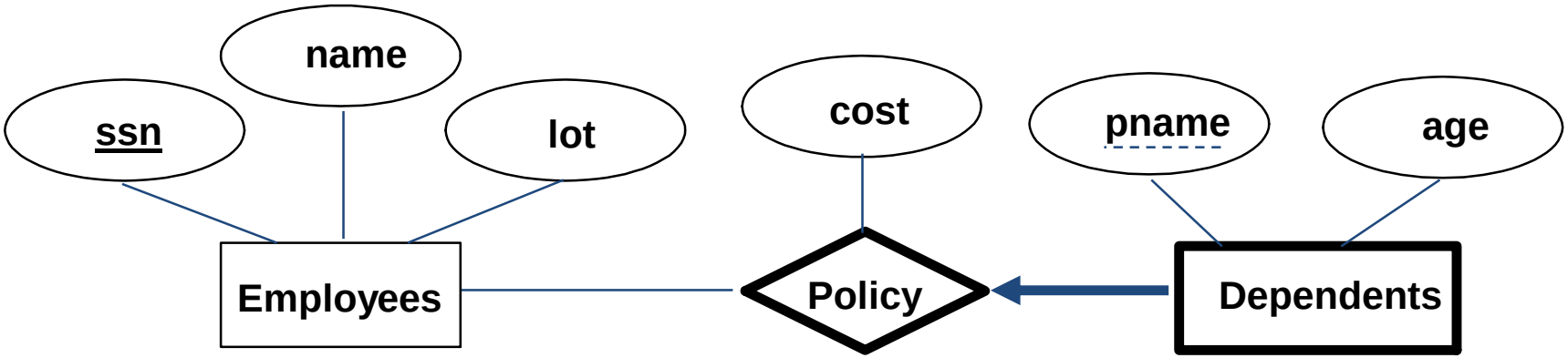
# Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE  Dept_Mgr(
   did  INTEGER,
   dname  CHAR(20),
   budget  REAL,
   ssn  CHAR(11) NOT NULL,
   since  DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY  (ssn) REFERENCES Employees,
     ON DELETE NO ACTION)
```

# Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



# Translating weak entities

- Weak entity set and identifying relationship set are translated into a single table --- it has a (1,1) cardinality constraint.

```
CREATE TABLE  Dep_Policy (
    dname  CHAR(20),
    age  INTEGER,
    cost  REAL,
    parent_ssn  CHAR(9) NOT NULL,
    PRIMARY KEY  (dname, parent_ssn),
    FOREIGN KEY  (parent_ssn) REFERENCES Employees,
       ON DELETE CASCADE)
```

- When an owner entity is deleted all owned entity should also be deleted.

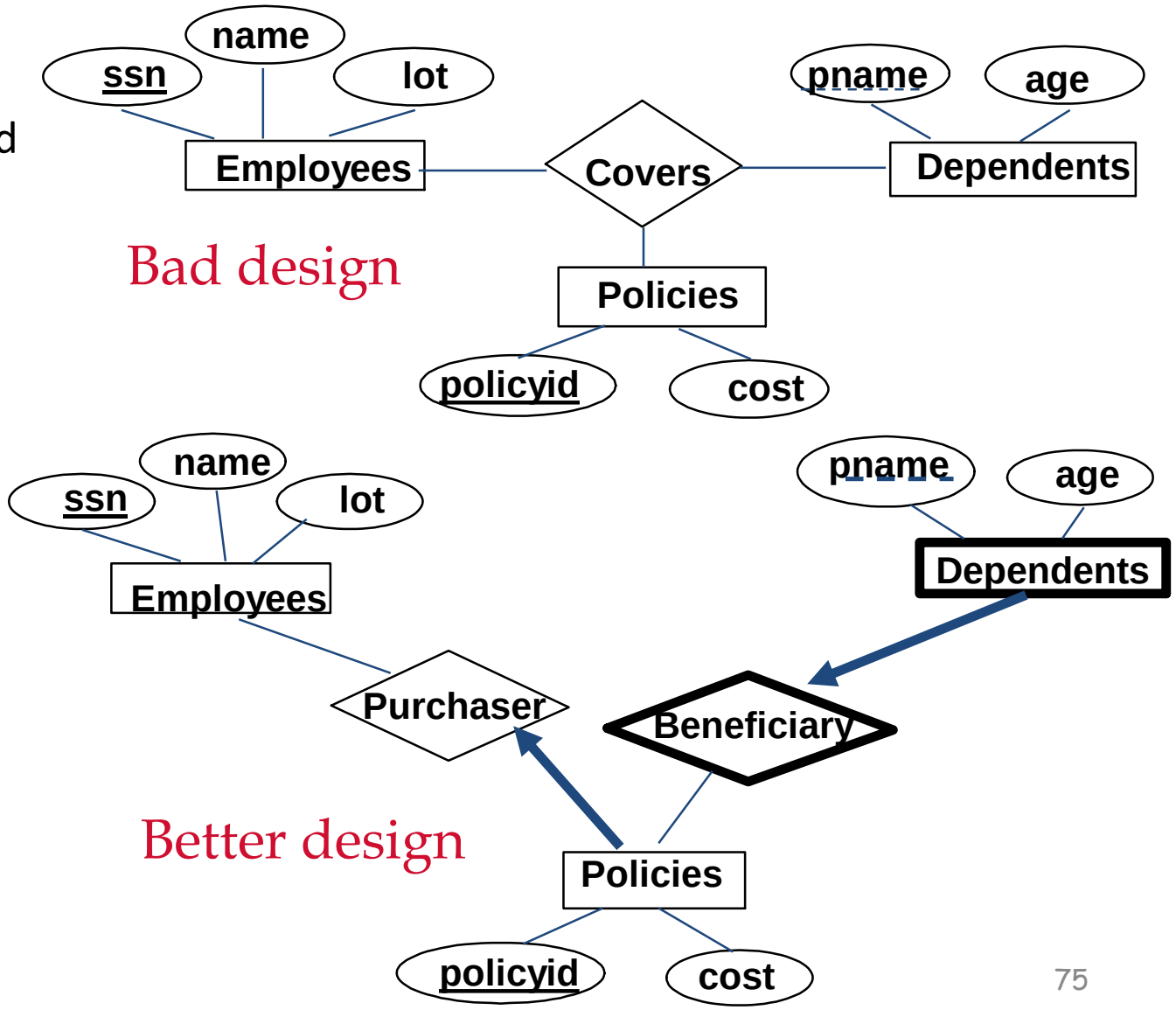
# Translating Weak Entity Sets

- 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 Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

# Review: Binary vs. Ternary Relationships

- If each policy is owned by just 1 employee:
  - Key constraint on Policies would mean policy can only cover 1 dependent!
- What are the additional constraints in the 2nd diagram?



# Binary vs. Ternary Relationships (Contd.)

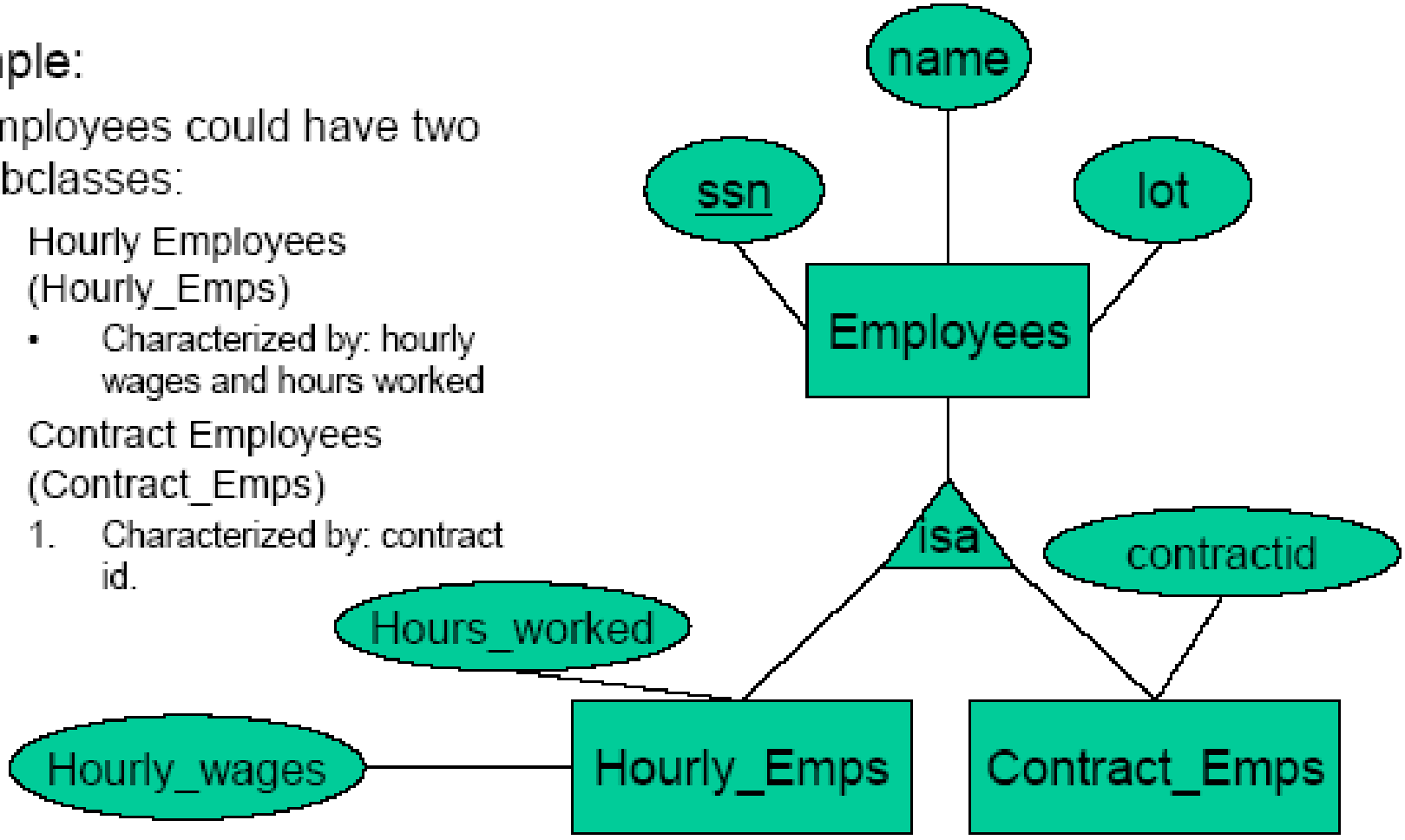
- The key constraints allow us to combine Purchaser with Policies and Beneficiary with Dependents.
- Participation constraints lead to NOT NULL constraints.
- What if Policies is a weak entity set?

```
CREATE TABLE Policies (
    policyid  INTEGER,
    cost  REAL,
    ssn  CHAR(11)  NOT NULL,
    PRIMARY KEY (policyid).
    FOREIGN KEY (ssn) REFERENCES Employees,
      ON DELETE CASCADE)

CREATE TABLE Dependents (
    pname  CHAR(20),
    age  INTEGER,
    policyid  INTEGER,
    PRIMARY KEY (pname, policyid).
    FOREIGN KEY (policyid) REFERENCES Policies,
      ON DELETE CASCADE)
```

# Translating Class Hierarchies

- Example:
  - Employees could have two subclasses:
    1. Hourly Employees (Hourly_Emps)
       - Characterized by: hourly wages and hours worked
    2. Contract Employees (Contract_Emps)
       1. Characterized by: contract id.



# Translating Class Hierarchies

- Two approaches
  - Three tables: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *ssn)*;
    - We must delete Hourly_Emps tuple if referenced Employees tuple is deleted).
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes.
- Alternative: Just Hourly_Emps and Contract_Emps.
  - *Hourly_Emps*: *ssn*, *name*, *lot*, *hourly_wages*, *hours_worked*.
  - Each employee must be in one of these two subclasses.

# An Example

```
CREATE TABLE Student (
        ID          NUMBER,
        Fname       VARCHAR2(20),
        Lname       VARCHAR2(20),
);
```

# Constraints in Create Table

- Adding constraints to a table enables the database system to enforce data integrity.
- Different types of constraints:

  * Not Null
  * Unique
  * Foreign Key
  * Default Values
  * Primary Key
  * Check Condition

# Not Null Constraint

```
CREATE TABLE Student (
        ID          NUMBER,
        Fname       VARCHAR2(20) NOT NULL,
        Lname       VARCHAR2(20) NOT NULL,
);
```

# Primary Key Constraint

```
CREATE TABLE Student (
        ID          NUMBER    PRIMARY KEY,
        Fname       VARCHAR2(20) NOT NULL,
        Lname       VARCHAR2(20) NOT NULL,
);
```

Primary Key implies: * NOT NULL * UNIQUE.
There can only be one primary key.

# Primary Key Constraint (Syntax 2)

```
CREATE TABLE Students (
        ID          NUMBER,
        Fname       VARCHAR2(20) NOT NULL,
        Lname       VARCHAR2(20) NOT NULL,
        PRIMARY KEY(ID)
);
```

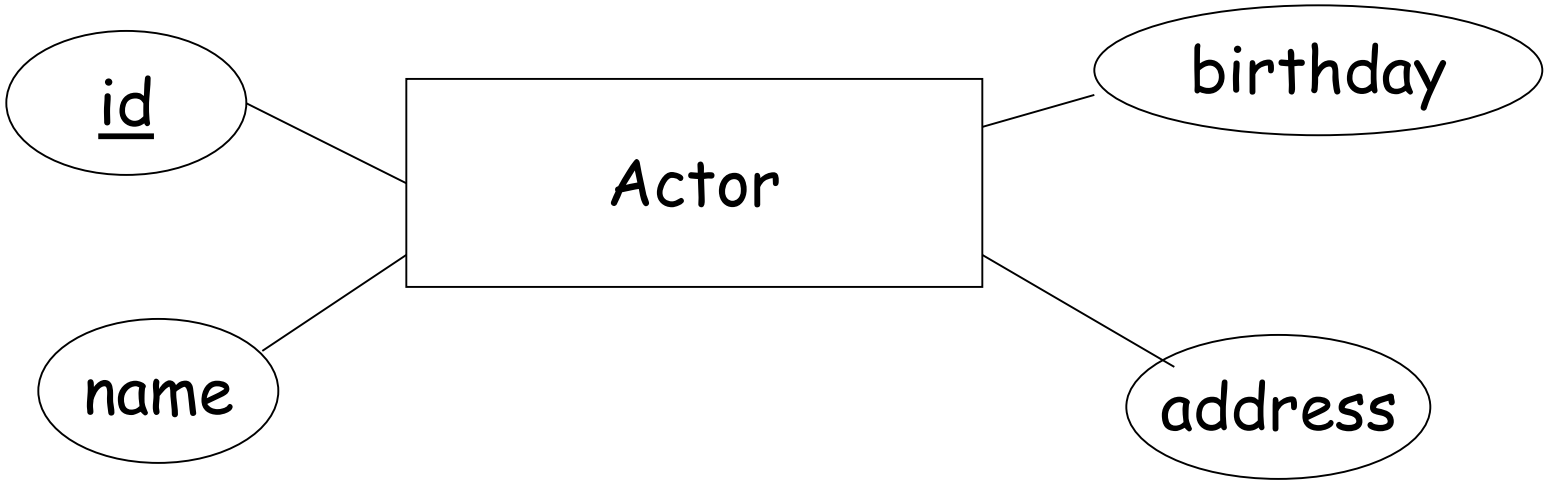
Needed when the primary key is made up of two or more fields

# Translating ER-Diagrams to Table Definitions

# Relations vs. Tables

- We show how to translate ER-Diagrams to table definitions
- Sometimes, people translate ER-Diagrams to relation definitions, which is more abstract than table definitions.
  - e.g., Student(ID, Fname, Lname);
  - table definitions contain, in addition, constraints and datatypes

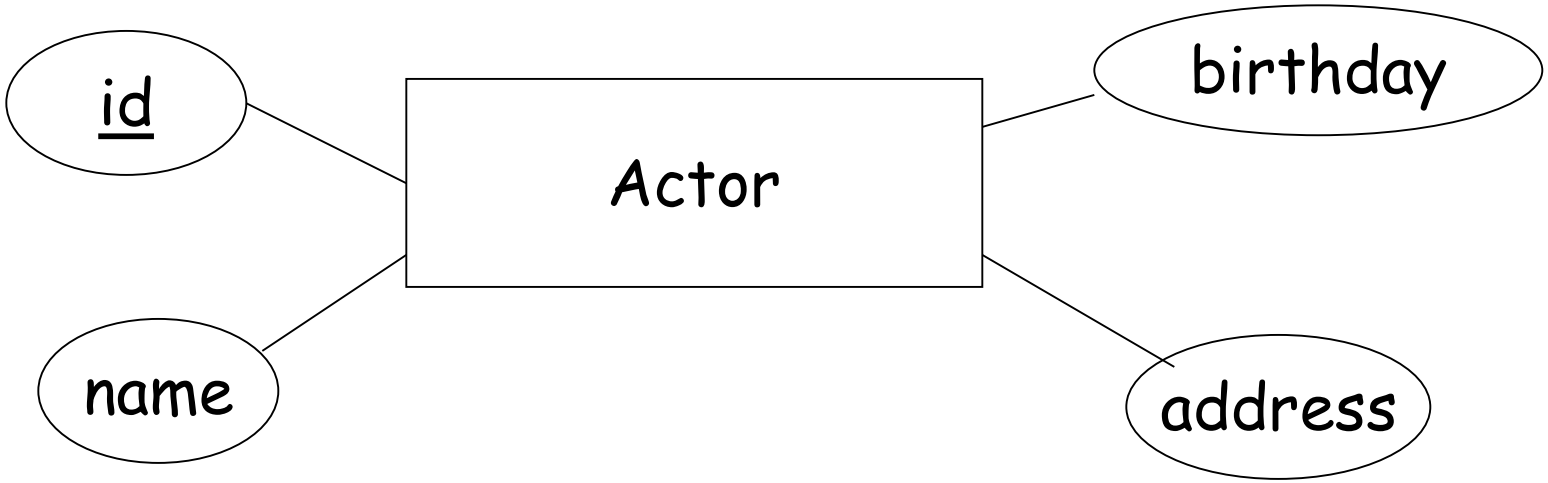
# Translating Entities




General Rule:

- Create a table with the name of the Entity.
- There is a column for each attribute
- The key in the diagram is the primary key of the table

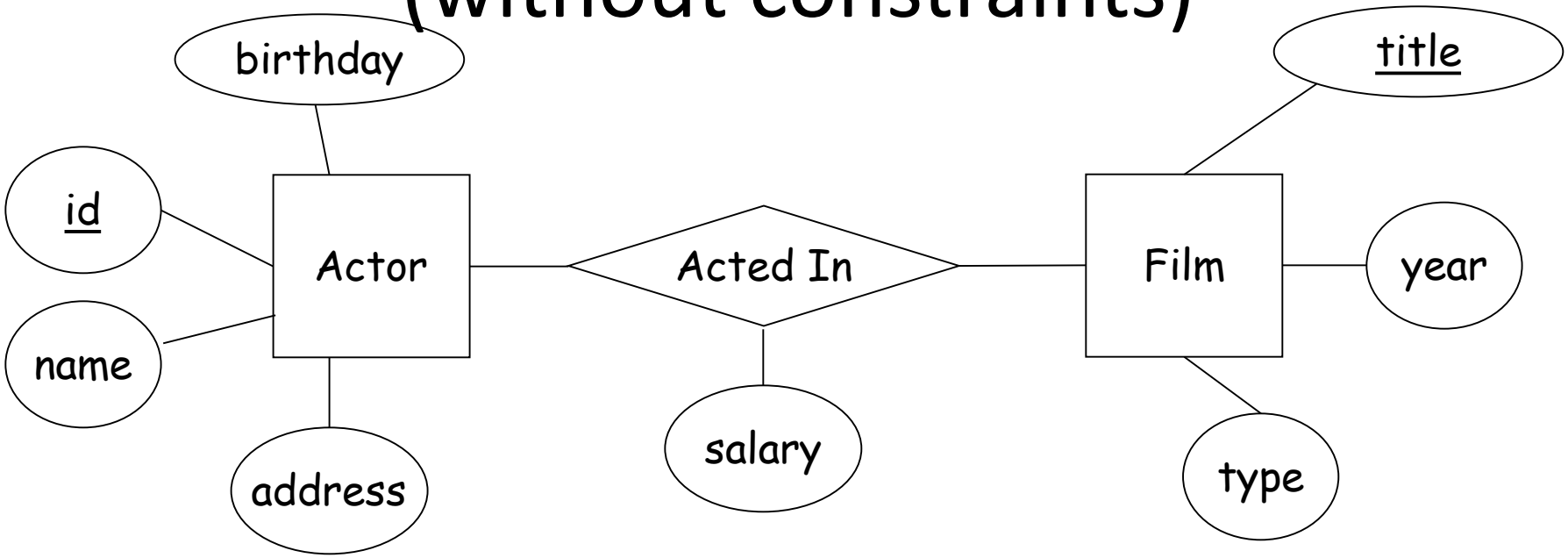
# Translating Entities




Relation: Actor (id, name, birthday, address)

```
create table Actor(id varchar2(20) primary key,
                   name varchar2(40),
                   birthday date,
                   address varchar2(100));
```

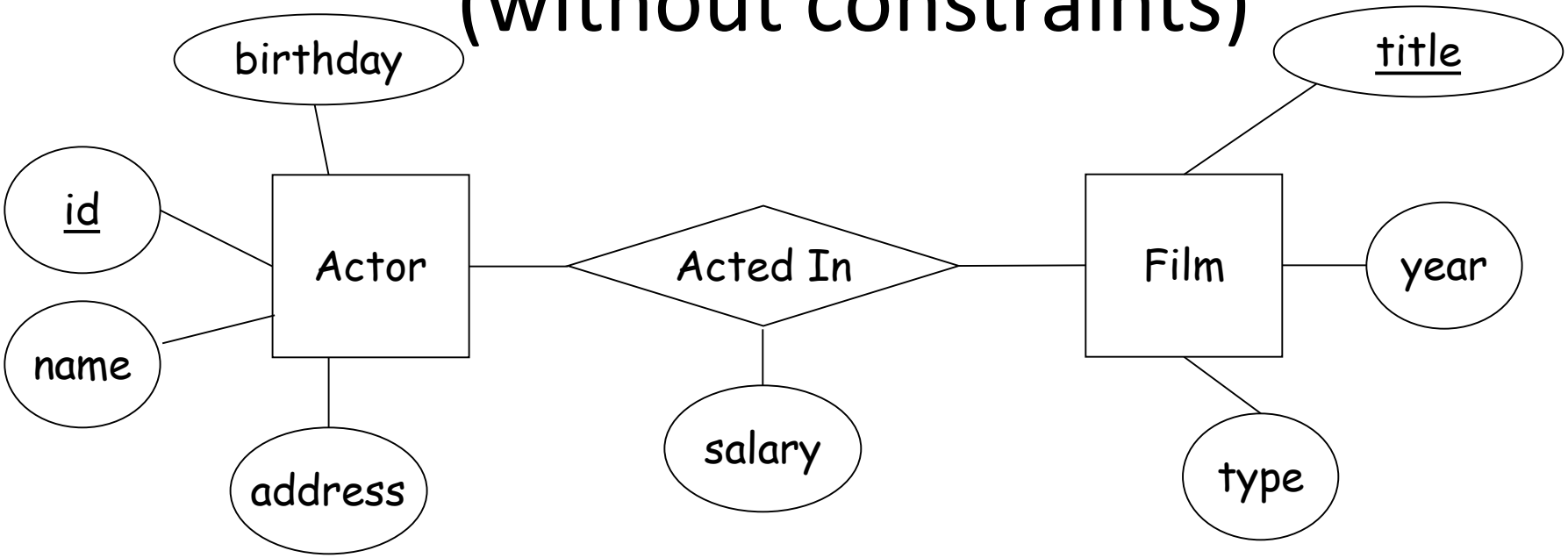
# Translating Relationships (without constraints)




General Rule:

- Create a table with the name of the relationship
- The table has columns for all of the relationship's attributes and for the keys of each entity participating in the relationship
- What is the primary key of the table?
- What foreign keys are needed?

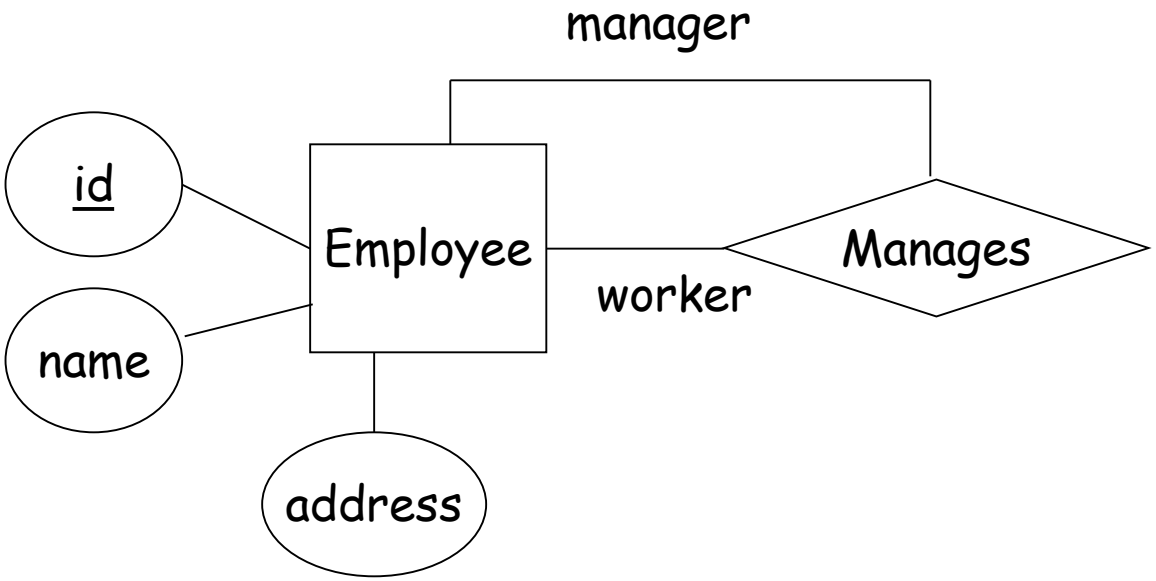
# Translating relationships (without constraints)




What would be the relation for ActedIn?

How would you define the table for ActedIn?

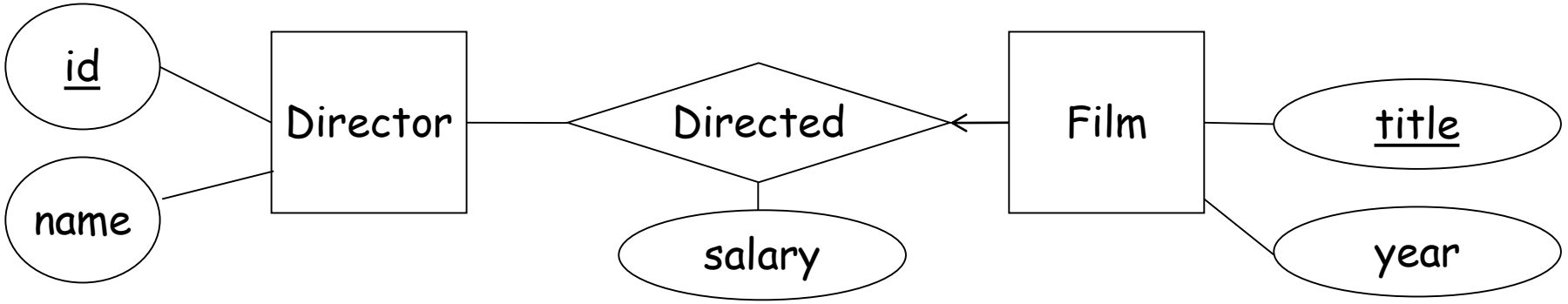
# Translating Recursive Relationships (without constraints)




Relation: Actor (worker-id, manager-id)

What would be the table definition?

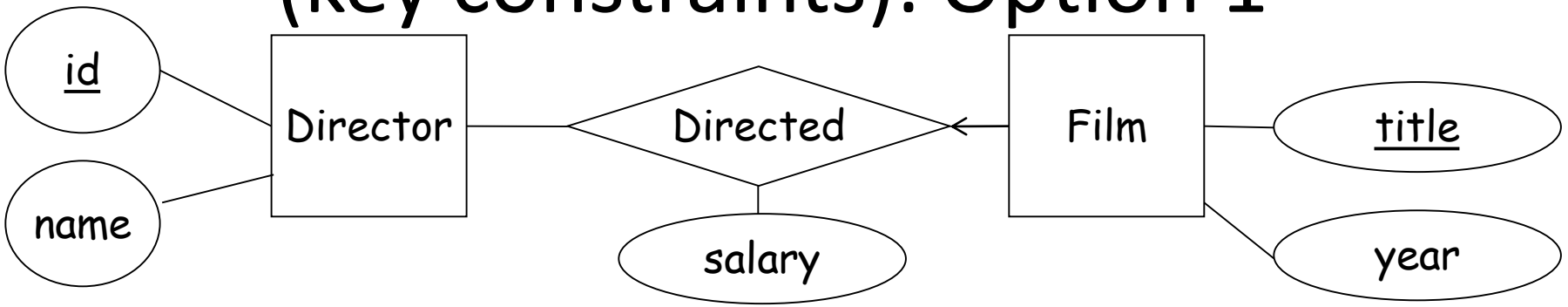
# Translating relationships (key constraints): Option 1




General Rule for Option 1:

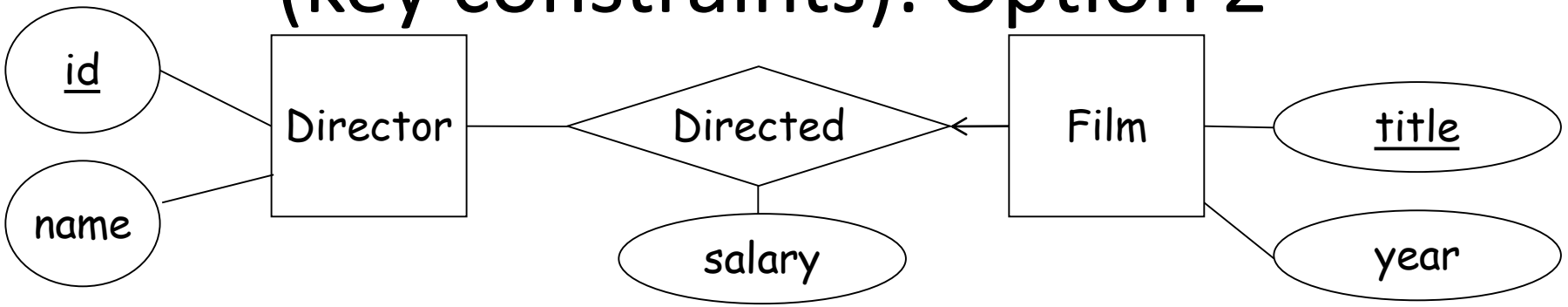
- Same as without key constraints, except that the primary key is defined differently

# Translating relationships (key constraints): Option 1




```
create table Directed(
        id varchar2(20),
        title varchar2(40),
        salary integer,
        What primary and foreign keys are missing?
)
```

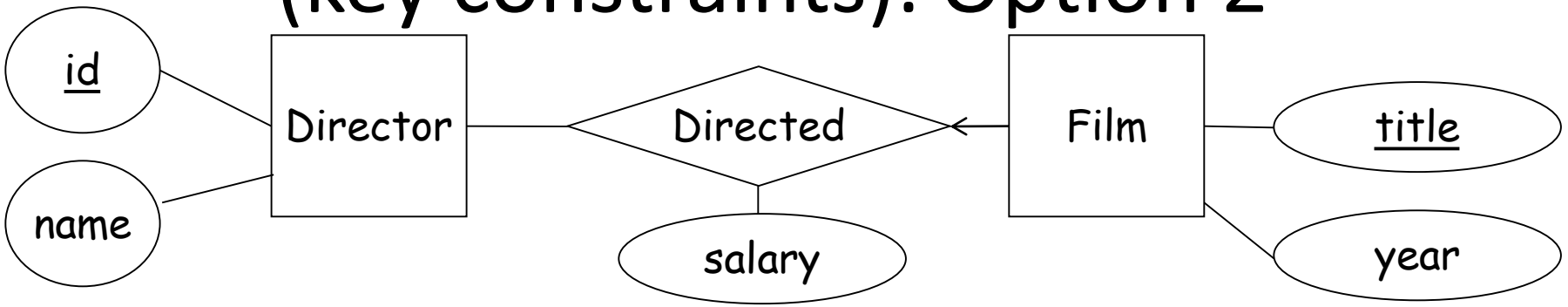
# Translating relationships (key constraints): Option 2




General Rule for Option 2:

- Do not create a table for the relationship
- Add information columns that would have been in the relationship's table to the table of the entity with the key constraint
- What is the disadvantage of this method?
- What is the advantage of this method?

# Translating relationships (key constraints): Option 2

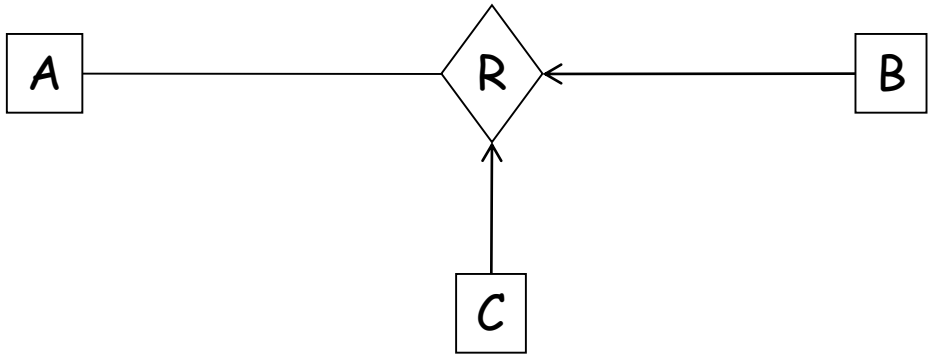



```
create table Film(
        title varchar2(40),
        year integer,
        primary key (title),


)
```

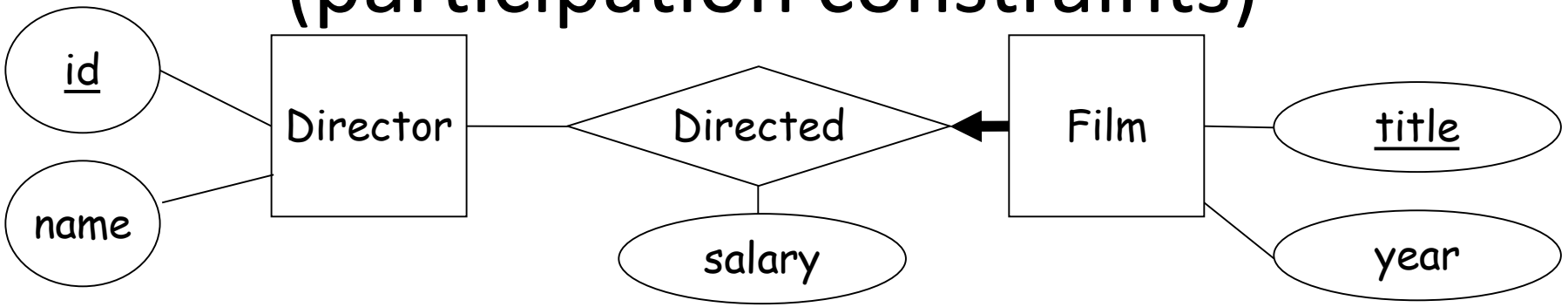
What 3 lines are missing?

# Translating relationships (key constraints)




- What are the different options for translating this diagram?

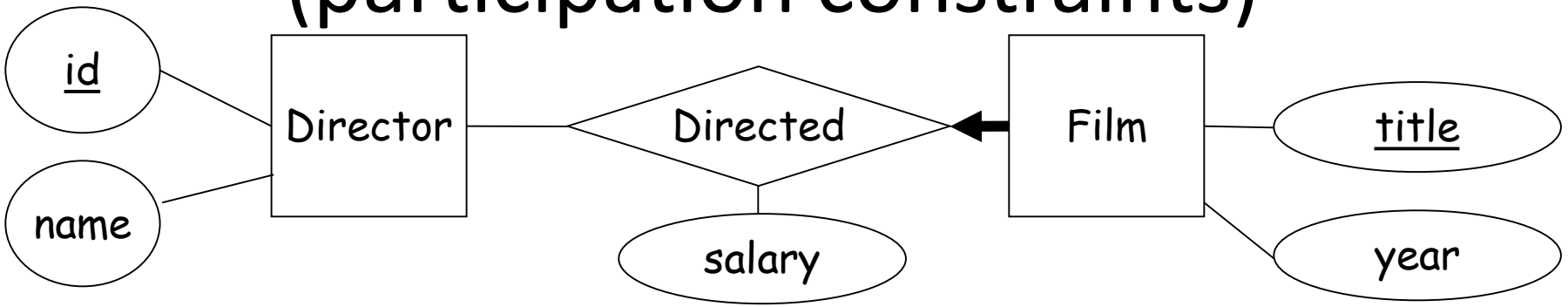
# Translating relationships (participation constraints)




General Rule:

- If has both participation and key constraint, use Option 2 from before.
- Add the not null constraint to ensure that there will always be values for the key of the other entity

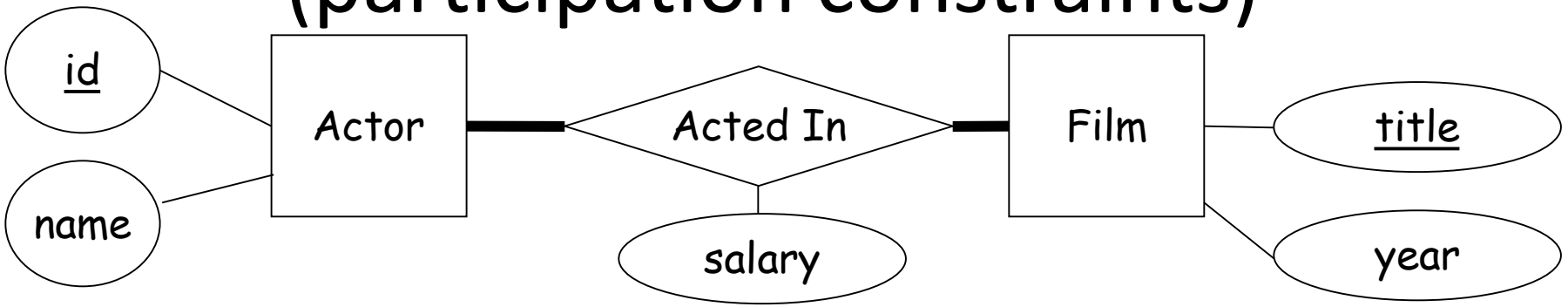
# Translating relationships (participation constraints)




```
create table Film(
        title varchar2(40),
        year integer,
        id varchar2(20),
        salary integer,
        foreign key (id) references Director,
        primary key (title))
```

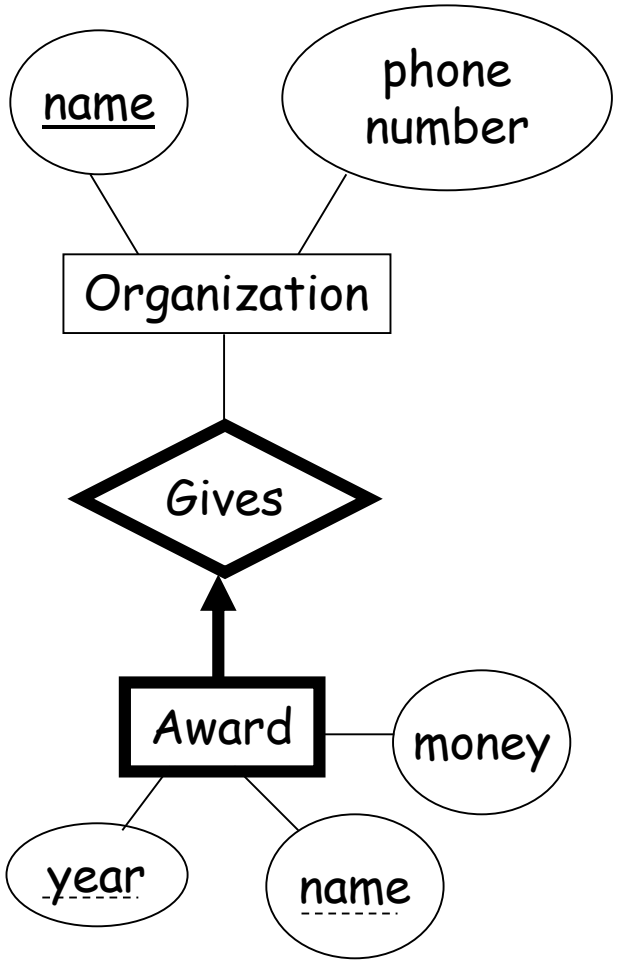
Where should we add NOT NULL?

# Translating relationships (participation constraints)



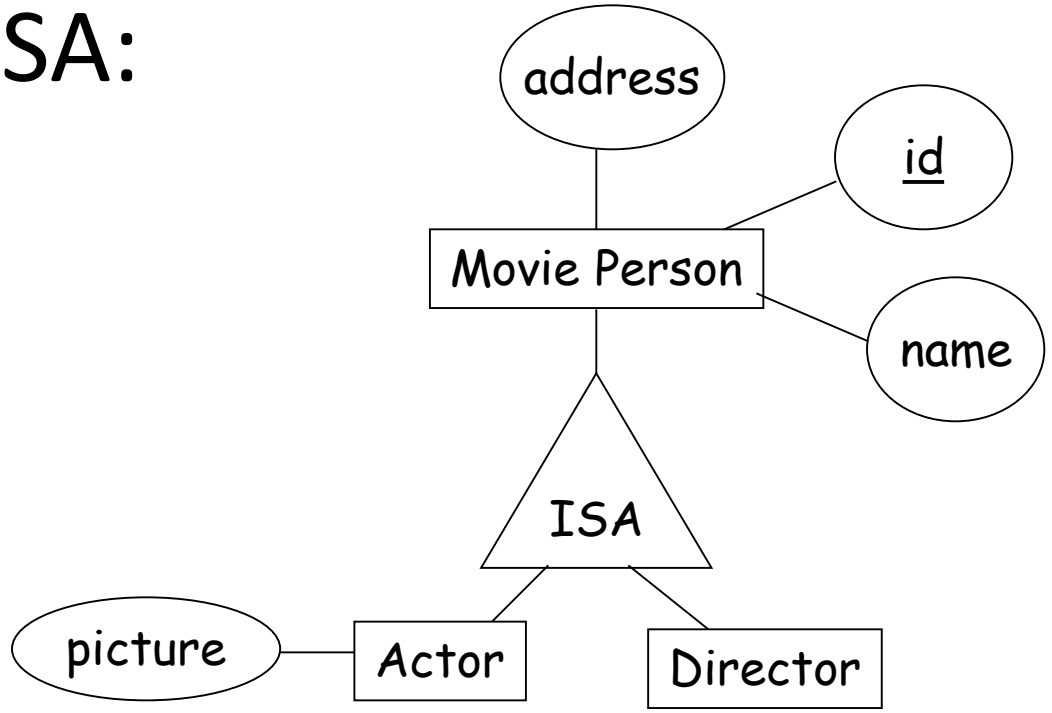

- How would we translate this?

# Translating Weak Entity Sets



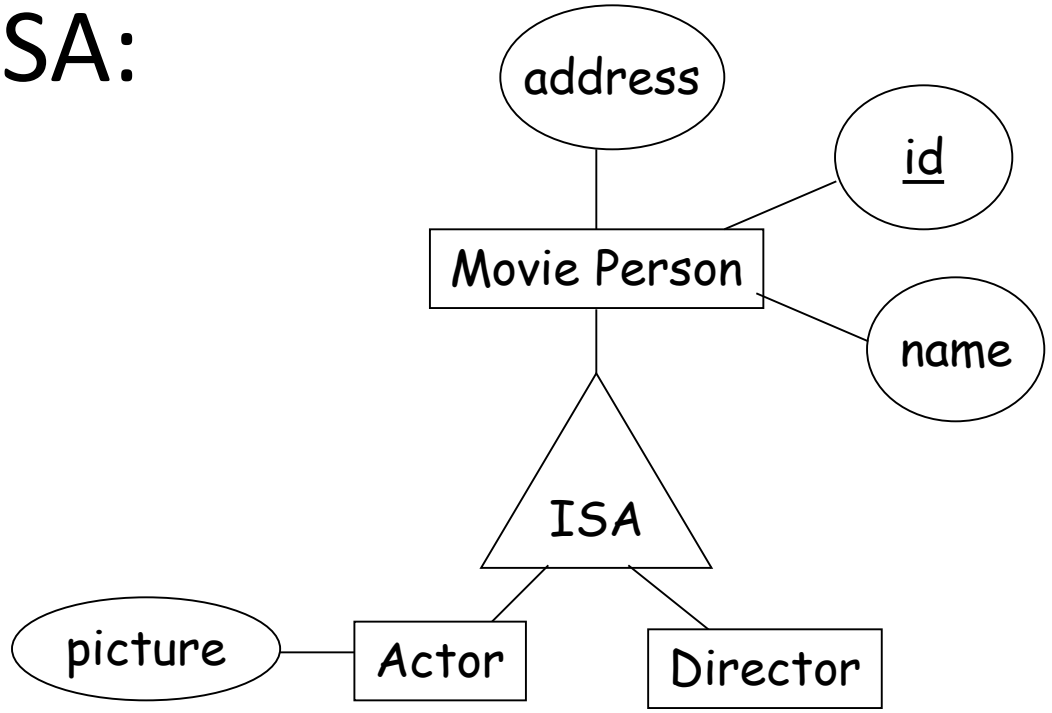

```
create table award(
    name varchar2(40),
    year integer,
    money number(6,2),
    o_name varchar2(40),
    primary key(name, year, o_name),
    foreign key (o_name) references
            Organization(name)
            on delete cascade
)
```

# Translating ISA: Option 1




```
create table MoviePerson( ... )
create table Actor(id varchar2(20),
                   picture bfile,
                   primary key(id),
                   foreign key (id) references MoviePerson))
create table Director(...)
```

# Translating ISA: Option 2



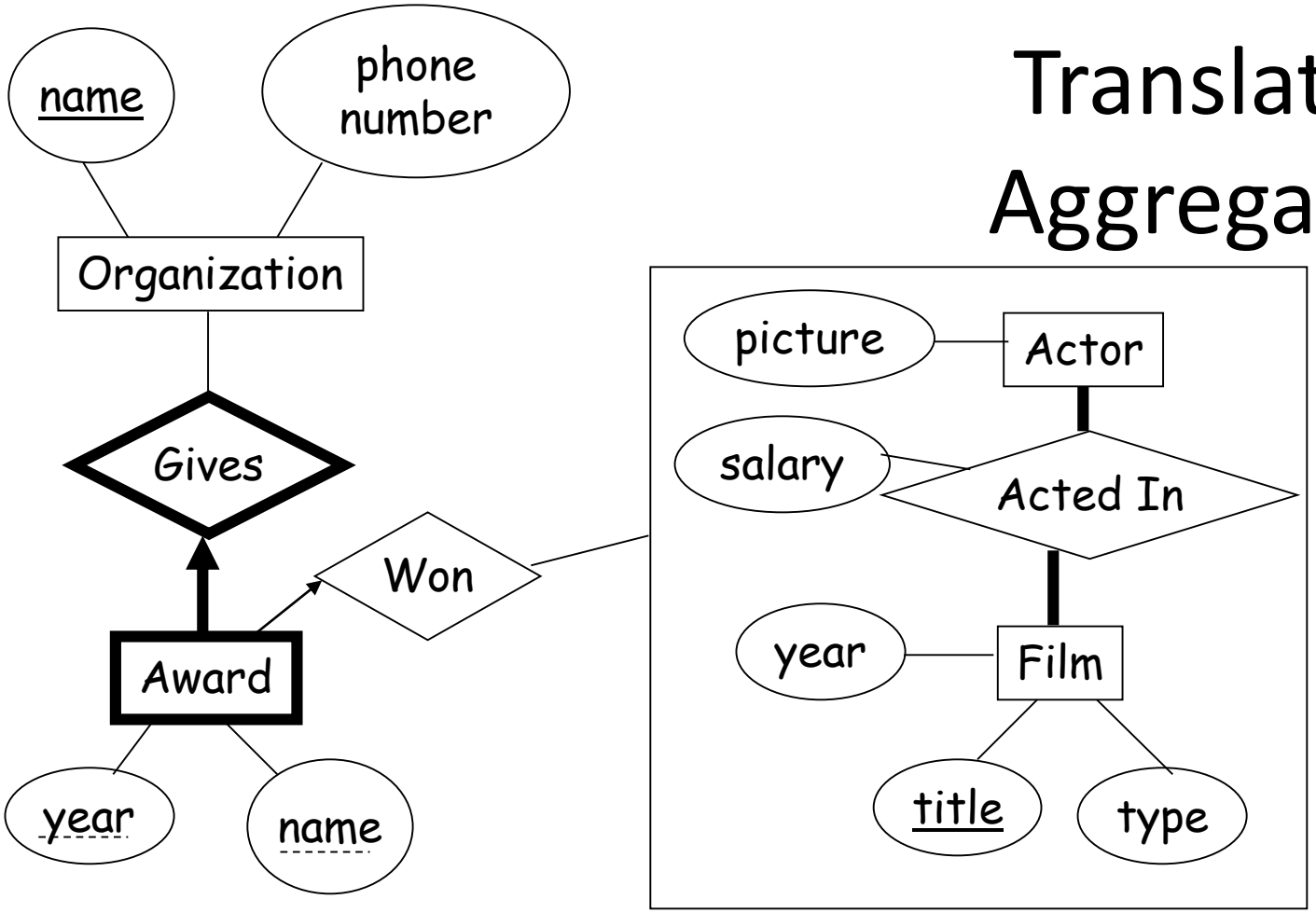

No table for MoviePerson!

```
create table Actor(id varchar2(20),
                   address varchar2(100),
                   name varchar2(20),
                   picture blob,
                  primary key(id));
create table Director(...)
```

# Which Option To Choose?

- What would you choose if:
  - Actor and Director DO NOT COVER MoviePerson?
  - Actor OVERLAPS Director?

# Translating Aggregation




- Create table for Won using:
  - key of ActedIn
  - key of Award (careful, award is a weak entity)