COP 4710: Database Systems
Fall 2011

Chapter 2 – Introduction to Data Modeling

Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4710/fall2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Introduction to Data Modeling

- Semantic data models attempt to capture the "meaning" of a database. Practically, they provide an approach for conceptual data modeling.
- Over the years there have been several different semantic data models that have been proposed.
- By far the most common is the *entity-relationship data model*, most often referred to as simply the *E-R data model*.
- The E-R model is often used as a form of communication between database designers and the end users during the developmental stages of a database.

Introduction to Data Modeling (cont.)

- The E-R model contains an extensive set of modeling tools, some of which we will not be concerned with as our primary objective is to give you some insight into conceptual database design and not learning all of the ins and outs of the E-R model.
- Another conceptual modeling which is becoming more common is the *Object Definition Language* (ODL) which is an object-oriented approach to database design that is emerging as a standard for object-oriented database systems.

Database Design

- The database design process can be divided into six basic steps. Semantic data models are most relevant to only the first three of these steps.

1. *Requirements Analysis:* The first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. Often this is an informal process involving discussions with user groups and studying the current environment. Examining existing applications expected to be replaced or complemented by the database system.

Database Design (cont.)

2. *Conceptual Database Design:* The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold on this data.

3. *Logical Database Design:* A DBMS must be selected to implement the database and to convert the conceptual database design into a database schema within the data model of the chosen DBMS.

Database Design (cont.)

4. *Schema Refinement*: In this step the schemas developed in step 3 above are analyzed for potential problems. It is in this step that the database is *normalized*. Normalization of a database is based upon some elegant and powerful mathematical theory. We will discuss normalization later in the term.

5. *Physical Database Design:* At this stage in the design of a database, potential workloads and access patterns are simulated to identify potential weaknesses in the conceptual database. This will often cause the creation of additional indices and/or clustering relations. In critical situations, the entire conceptual model will need restructuring.

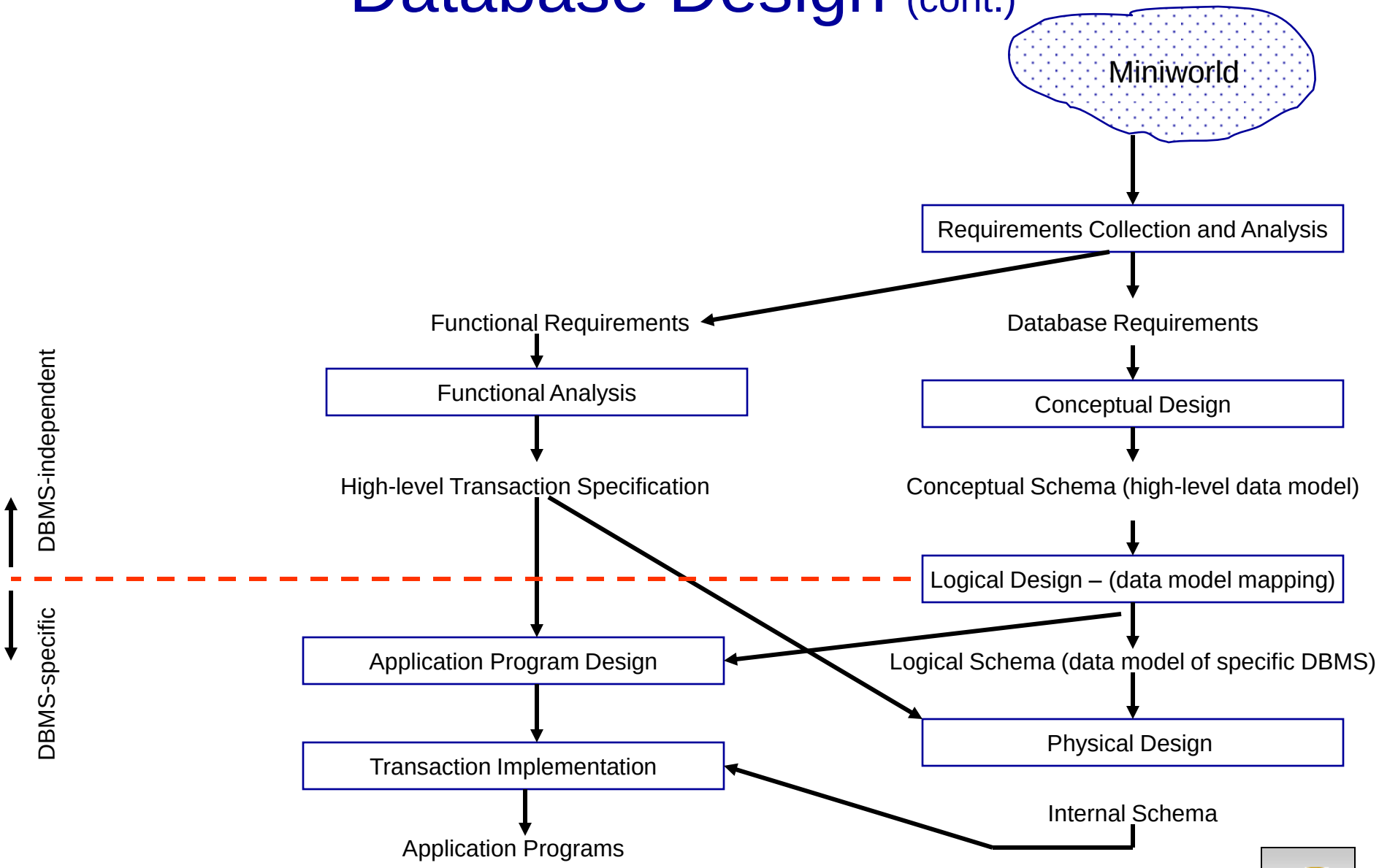
Database Design (cont.)

6. *Security Design:* Different user groups are identified and their different roles are analyzed so that access patterns to the data can be defined.

- There is often a seventh step in this process with the last step being a *tuning phase*, during which the database is made operational (although it may be through a simulation) and further refinements are made as the system is "tweaked" to provide the expected environment.

- The illustration on the following page summarizes the main phases of database design.

Database Design (cont.)



The Entity-Relationship Model

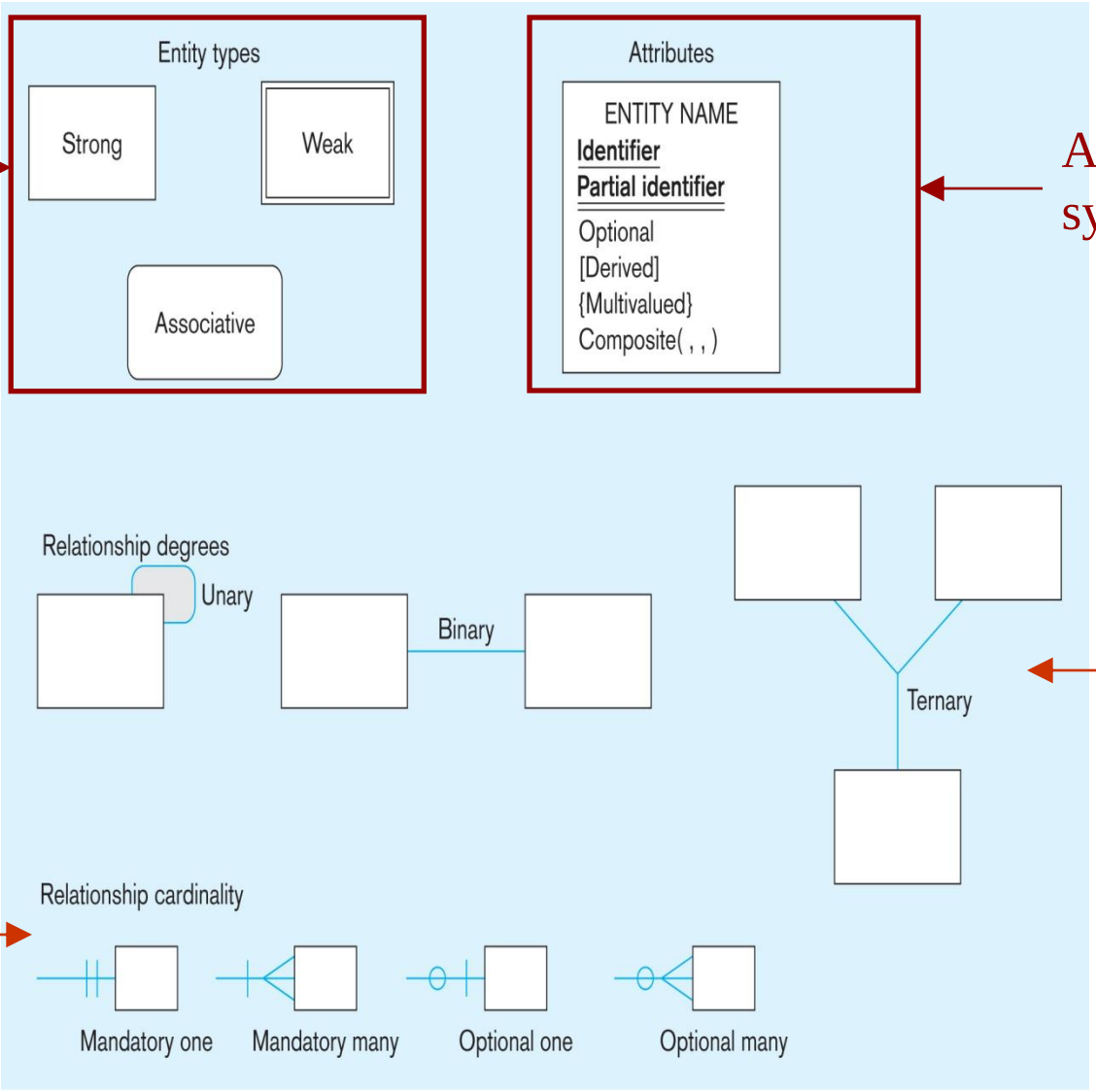
- The E-R model employs three basic notions: *entity sets, relationship sets*, and *attributes*.
- An *entity* is a "thing" or "object" in the real world that is distinguishable from all other objects. An entity may be either concrete, such as a person or a book, or it may be abstract, such as a bank loan, or a holiday, or a concept.
- An entity is represented by a set of *attributes*. Attributes are descriptive properties or characteristics possessed by an entity.
- An *entity set* is a set of entities of the same type that share the same attributes. For example, the set of all persons who are customers at a particular bank can be defined as the entity set *customers*.

The Entity-Relationship Model (cont.)

- Entity sets do not need to be disjoint. For example, we could define the entity set of all persons who work for a bank (*employee*) and the entity set of all persons who are customers of the bank (*customers*). A given person entity might be an *employee*, a *customer*, both, or neither.
- For each attribute, there is a permitted set of values, called the *domain* (sometimes called the *value set*), of that attribute. More formally, an attribute of an entity set is a function that maps from the entity set into a domain. Since an entity set may have several attributes, each entity in the set can be described by a set of <attribute, data-value> pairs, one for each attribute of the entity set.
- A database contains a collection of entity sets.

Basic ERD Notation

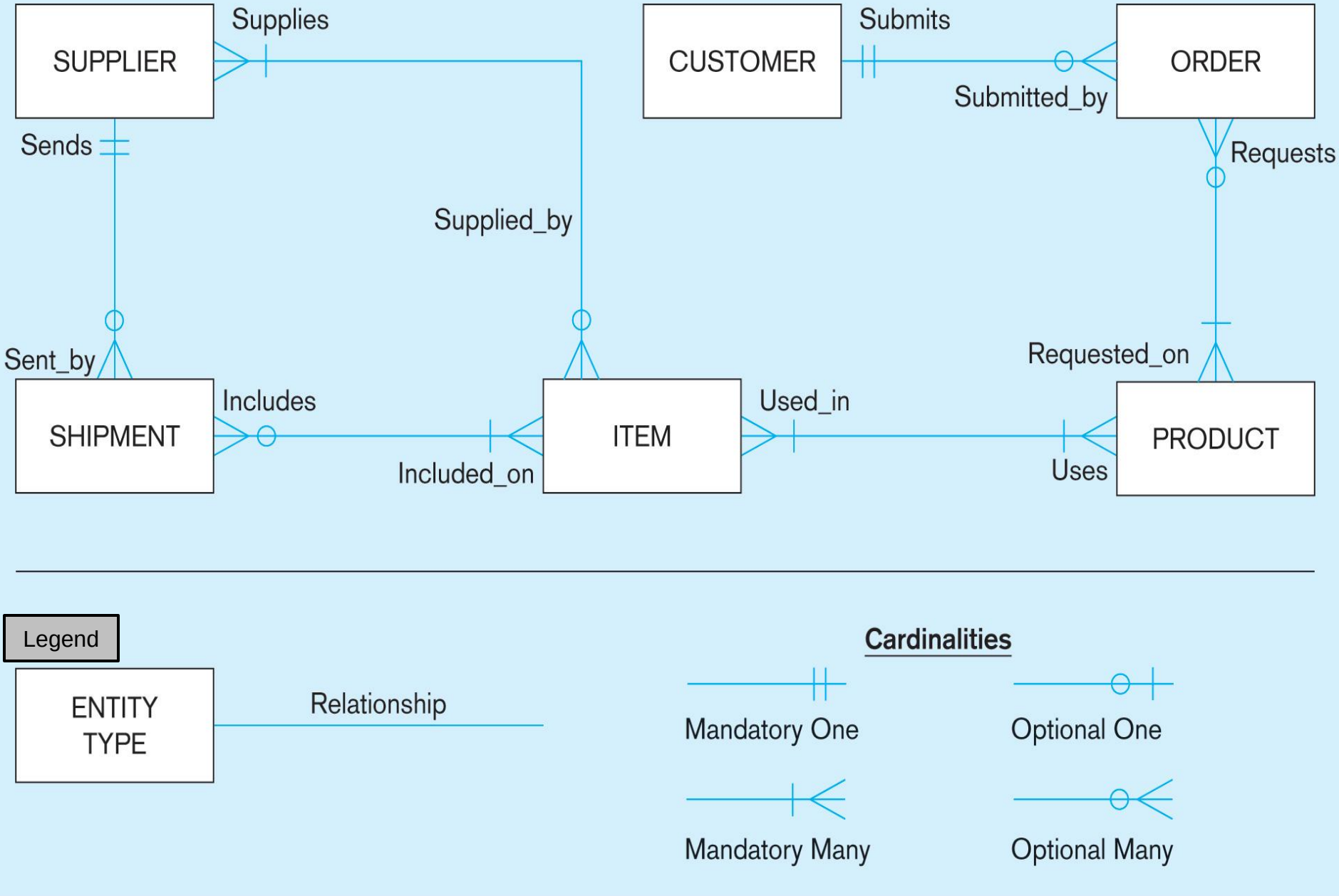
Entity symbols



Attribute symbols

Relationship degrees specify number of entity types involved

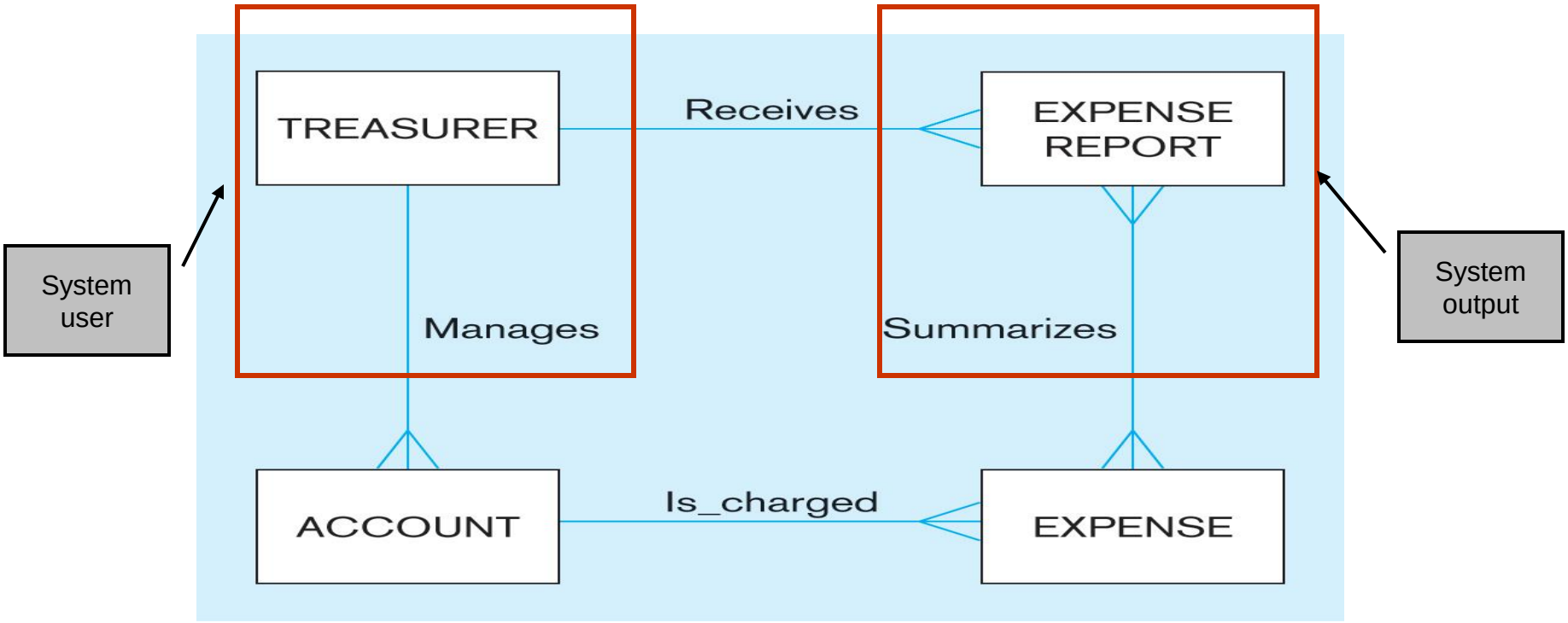
Relationship cardinalities specify how many of each entity type is allowed


SUPPLIER
Supplies
CUSTOMER
Submits
ORDER
Submitted_by
Sends
Requests
Supplied_by
Sent_by
Requested_on
SHIPMENT
Includes
ITEM
Used_in
PRODUCT
Included_on
Uses
Legend
ENTITY TYPE
Relationship
Cardinalities
Mandatory One
Optional One
Mandatory Many
Optional Many

What Should an Entity Be?

- SHOULD BE:
  - An object that will have many instances in the database
  - An object that will be composed of multiple attributes
  - An object that we are trying to model
- SHOULD NOT BE:
  - A user of the database system
  - An output of the database system (e.g. a report)

Inappropriate Entities



Only necessary entities

Attributes

- Attribute - property or characteristic of an entity type
- Classifications of attributes:
  - Required versus Optional Attributes
  - Simple versus Composite Attribute
  - Single-Valued versus Multivalued Attribute
  - Stored versus Derived Attributes
  - Identifier Attributes

Identifiers (Keys)

- Identifier (Key) - An attribute (or combination of attributes) that uniquely identifies individual instances of an entity type.
- Simple Key versus Composite Key.
- Candidate Key – an attribute that could be a key…satisfies the requirements for being a key.

Characteristics of Identifiers

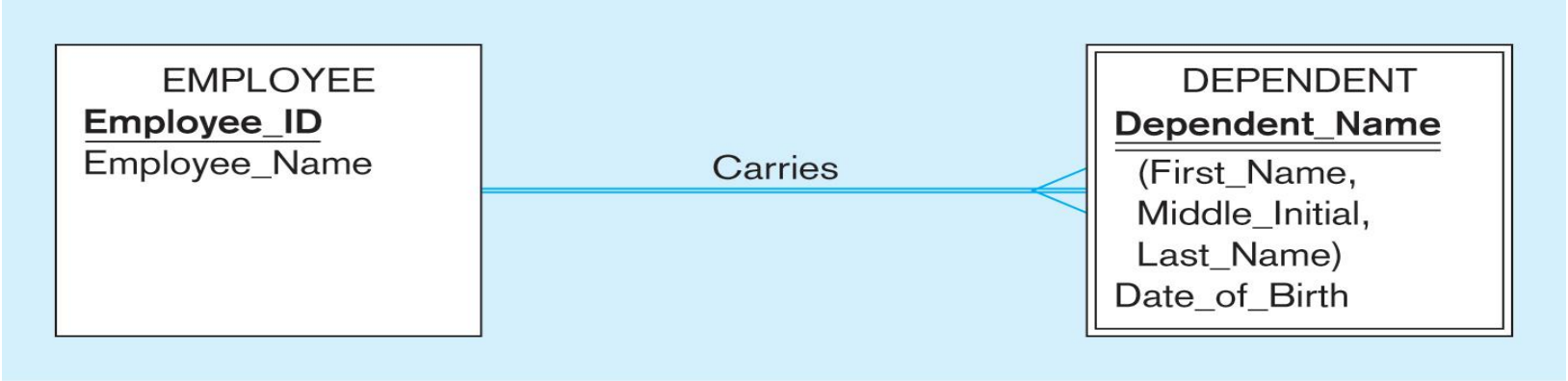
- Will not change in value.
- Will not be null.
- No intelligent identifiers (e.g. containing locations or people that might change).
- Substitute new, simple keys for long, composite keys.

Strong vs. Weak Entities, and Identifying Relationships

- Strong entities
  - exist independently of other types of entities
  - has its own unique identifier
- Weak entity
  - dependent on a strong entity…cannot exist on its own
  - does not have a unique identifier
- Identifying relationship
  - links strong entities to weak entities

Weak vs. Strong Entities

- A weak entity is an entity type whose existence depends on some other entity type.
- The entity type on which the weak entity is dependent is called the identifying owner (or simply owner).
- A weak entity does not have its own identifier.

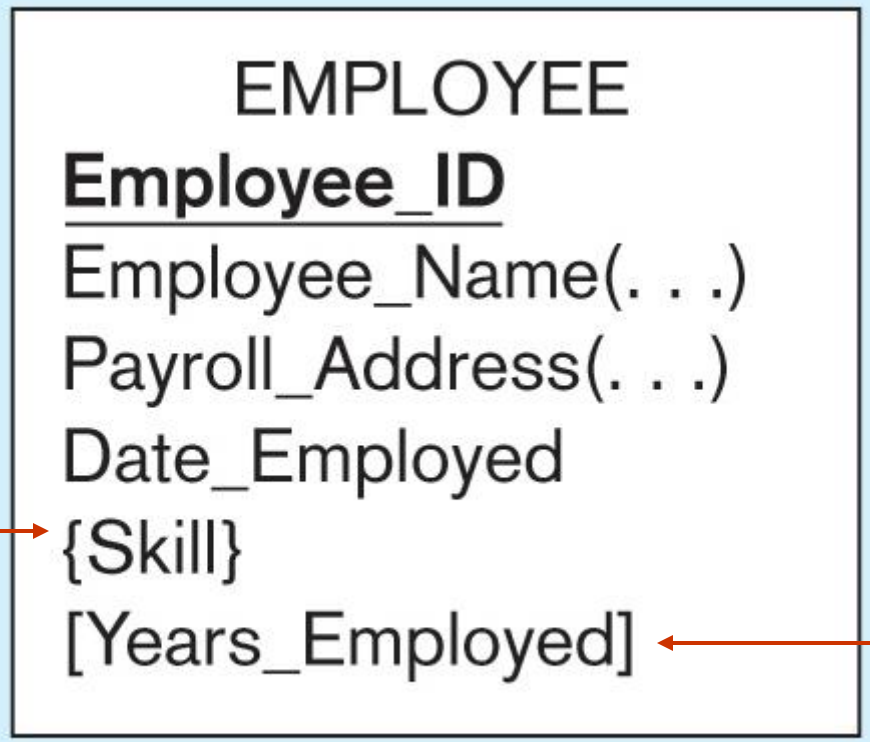


A Composite Attribute

An attribute broken into component parts

A Multi-valued Attribute And A Derived Attribute

A multi-valued attribute. Represented in curly braces.



A derived attribute. Represented in square braces.

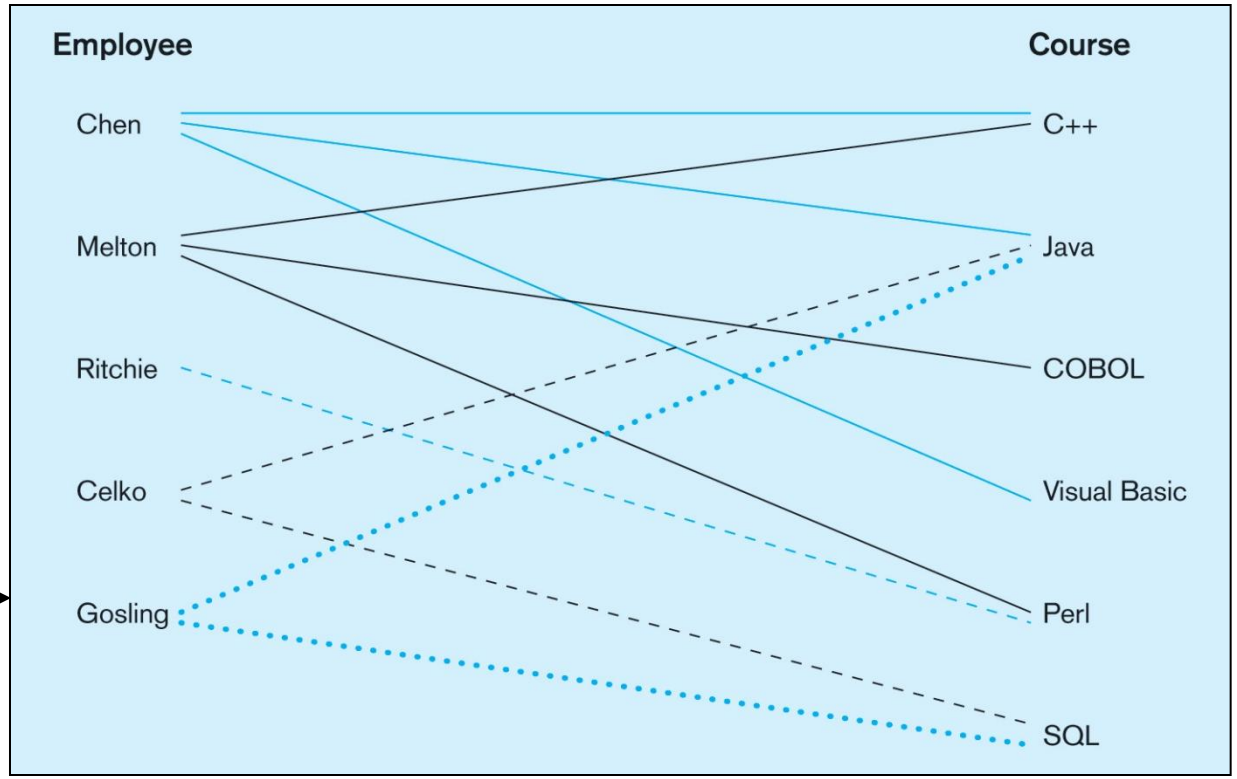
A Simple Identifier Attribute And A Composite Identifier Attribute

More on Relationships

- Relationship Types vs. Relationship Instances
  - The relationship type is as a line between entity types…the instance is between specific entity instances
- Relationships can have attributes
  - These describe features pertaining to the association between the entities in the relationship
- Two entities can have more than one type of relationship between them (multiple relationships)
- Associative Entity – combination of relationship and entity

More on Relationships

Relationship type



Relationship instance

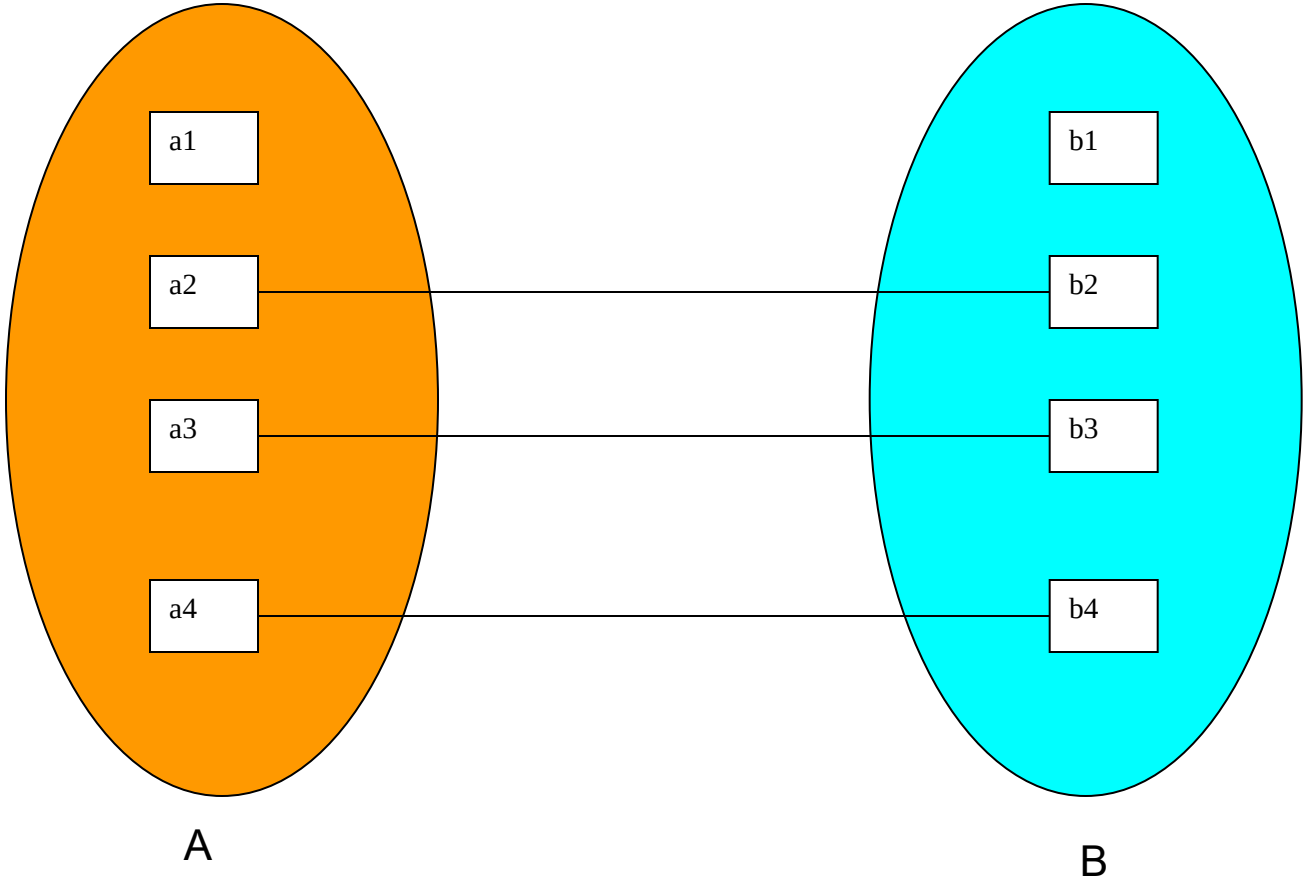
Degree of Relationships

- Degree of a relationship is the number of entity types that participate in it:
  - Unary Relationship
  - Binary Relationship
  - Ternary Relationship

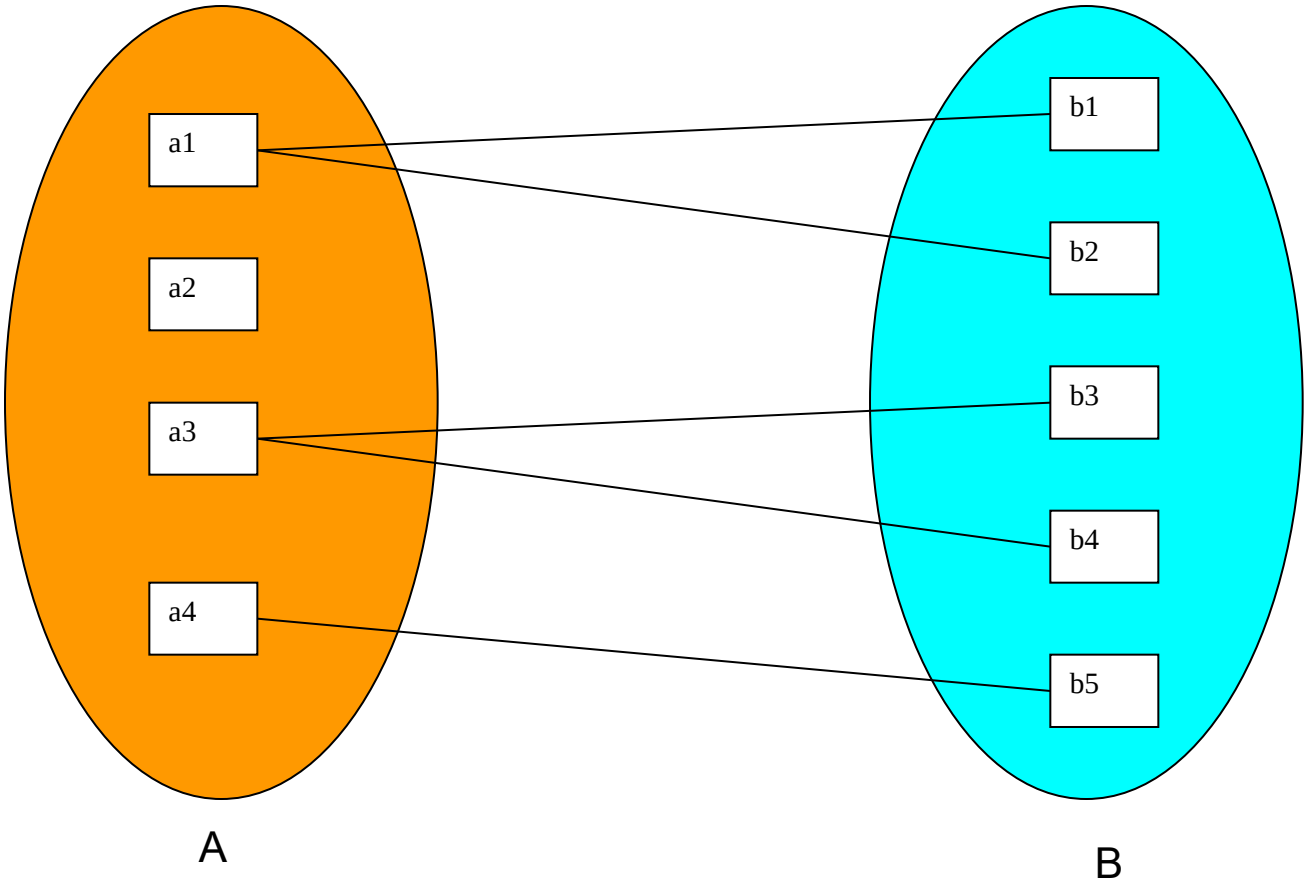
Cardinality of Relationships

- One-to-One
  - Each entity in the relationship will have exactly one related entity.
- One-to-Many
  - An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity.
- Many-to-Many
  - Entities on both sides of the relationship can have many related entities on the other side.

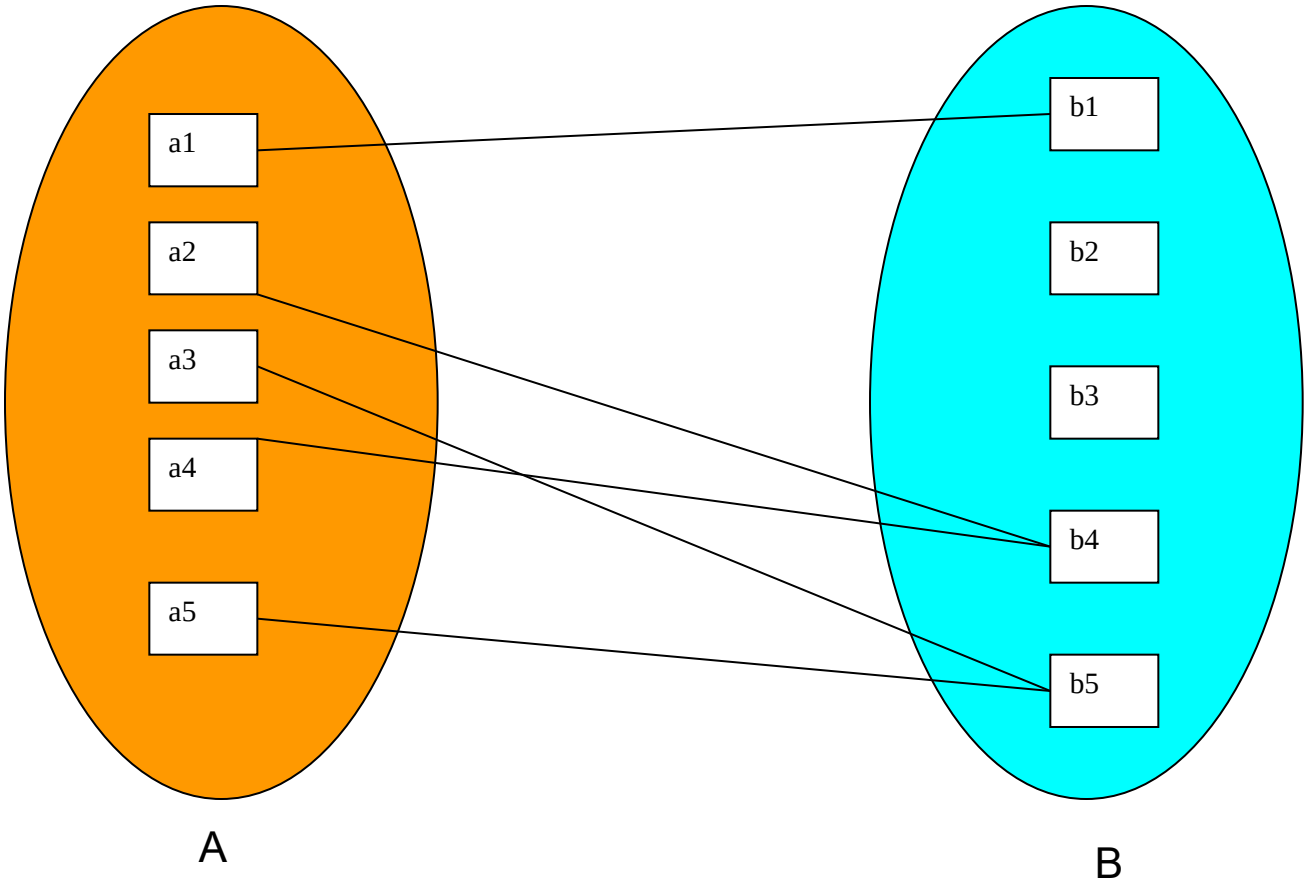
Mapping Cardinality: 1:1 from A to B



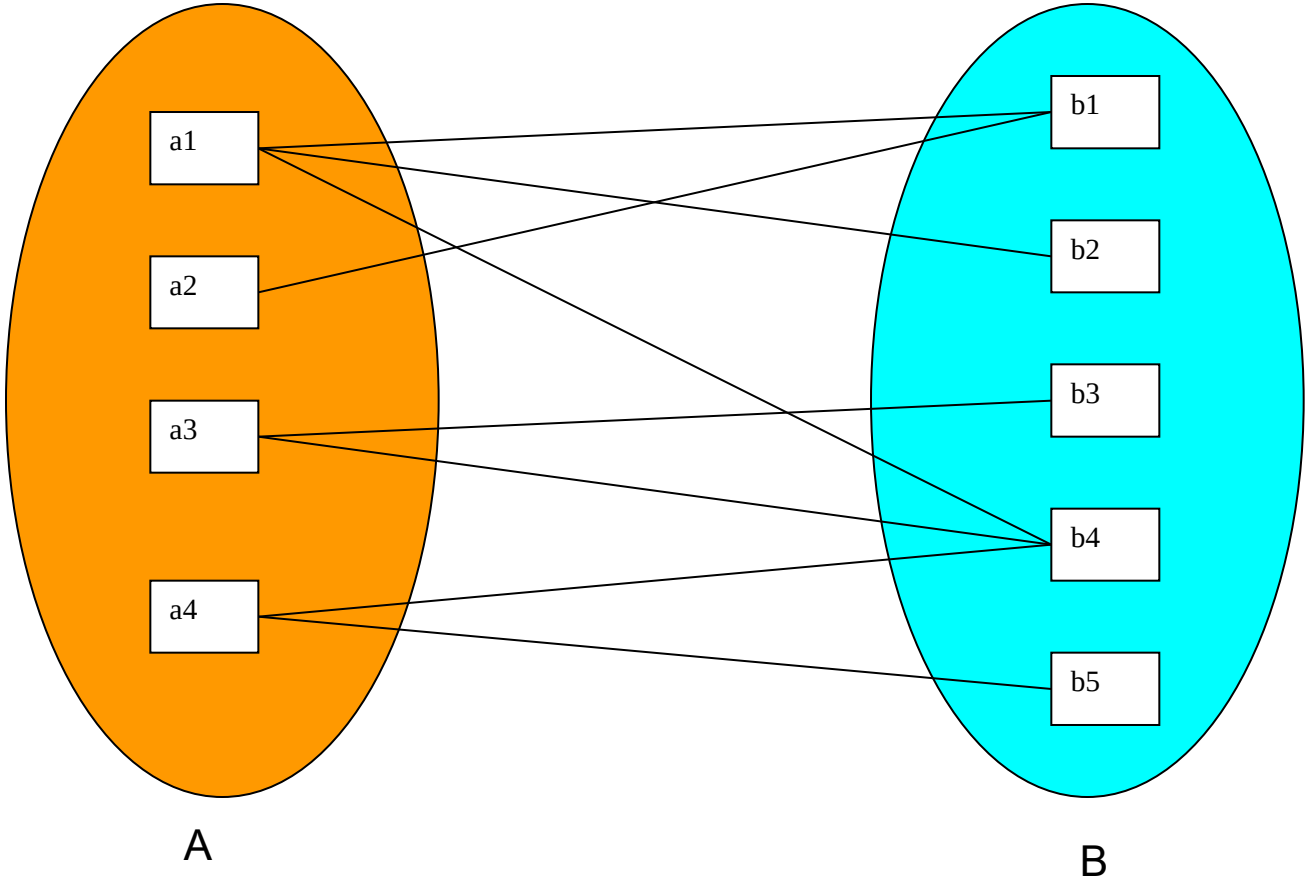
Mapping Cardinality: 1:M from A to B



Mapping Cardinality: M:1 from A to B



Mapping Cardinality: M:M from A to B

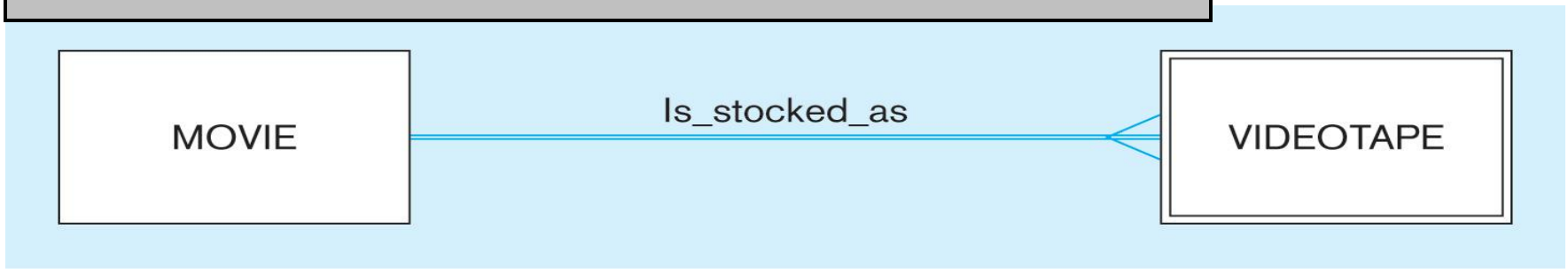


Cardinality Constraints

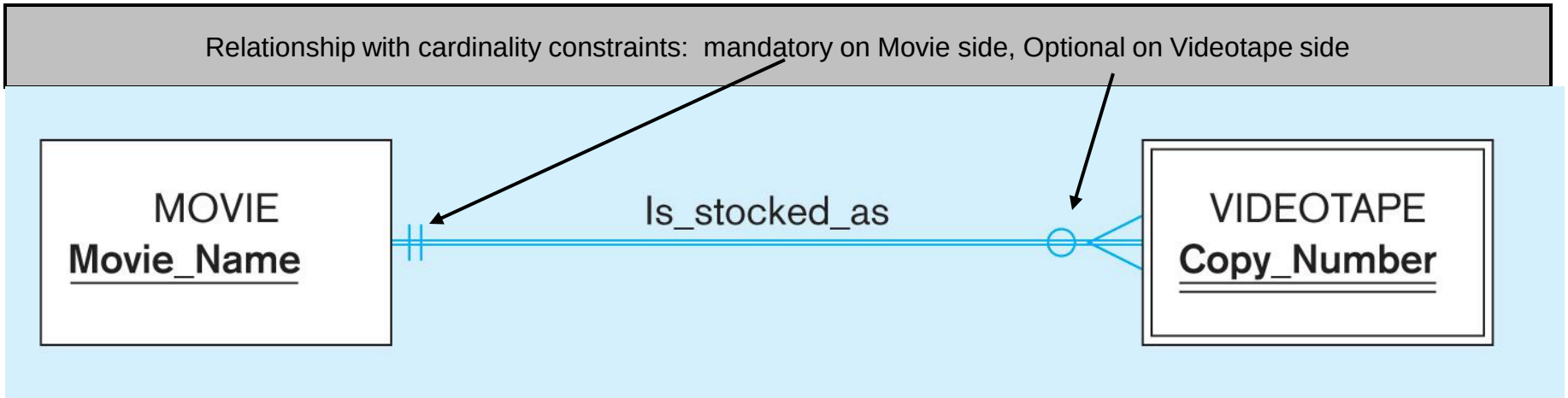
- Cardinality Constraints - the number of instances of one entity that can or must be associated with each instance of another entity.
- Minimum Cardinality
  - If zero, then optional.
  - If one or more, then mandatory.
- Maximum Cardinality
  - The maximum number possible.

Cardinality Constraints

Basic relationship: 1:M from Movie to Videotape (min =1, max = ?)

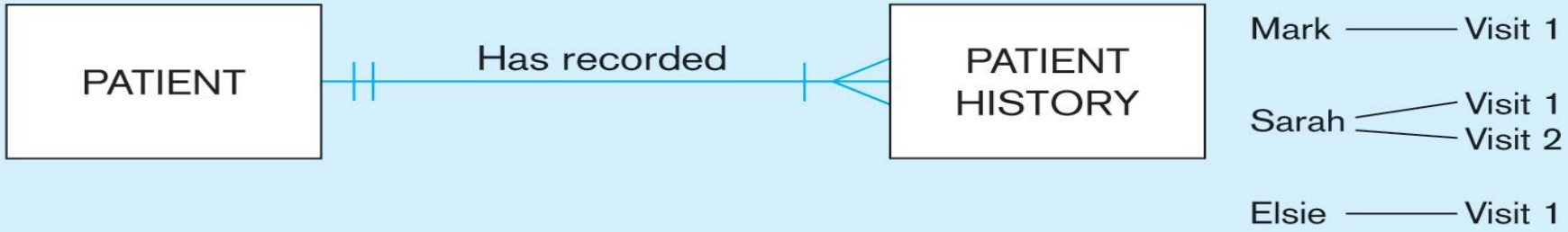


Relationship with cardinality constraints: mandatory on Movie side, Optional on Videotape side

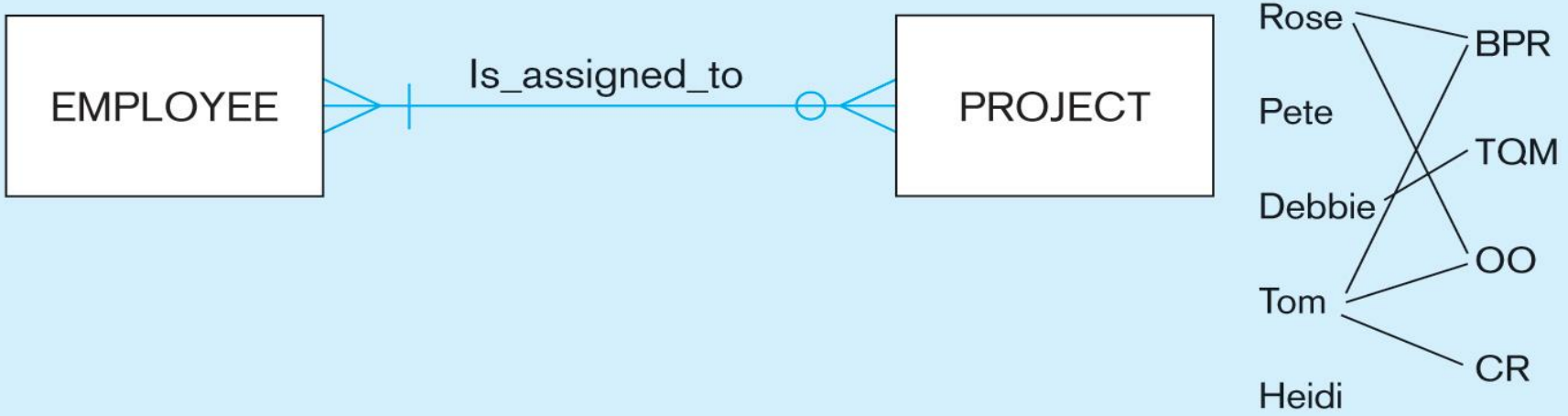


Cardinality Constraints

Mandatory cardinalities – Every patient must have at least 1 history. Every history belongs to 1 patient.

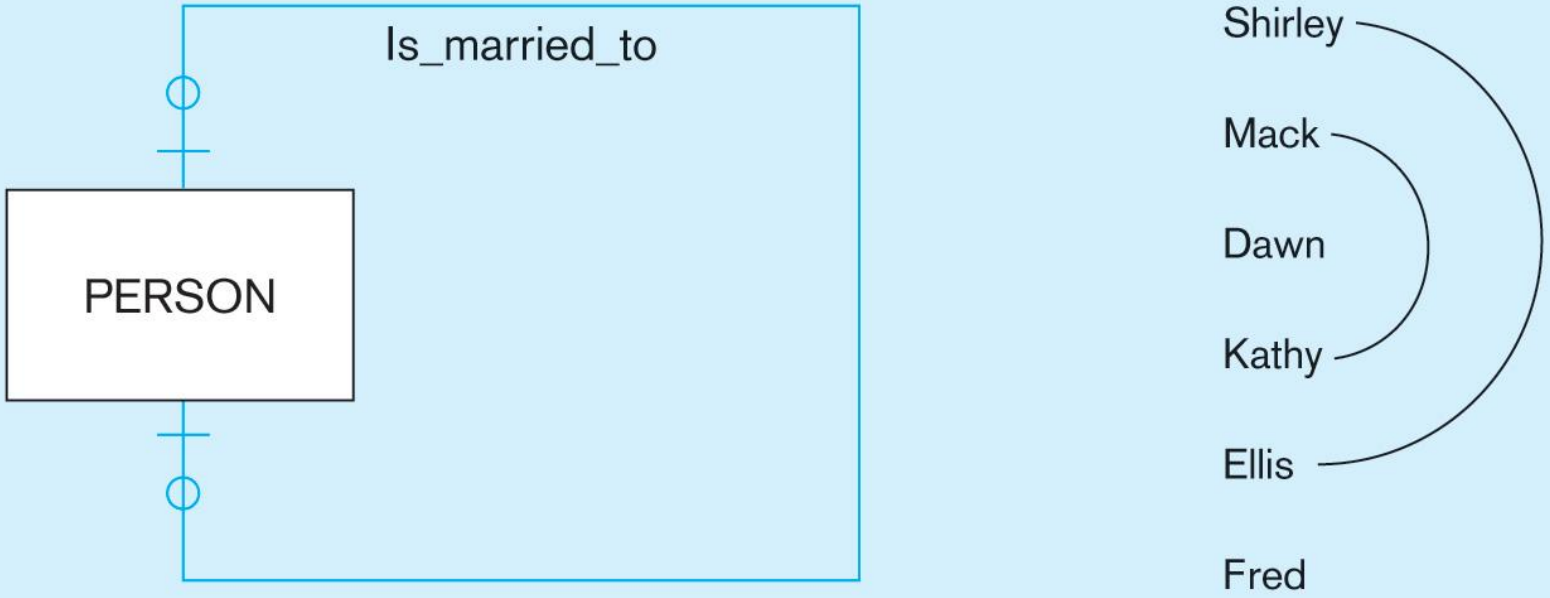


Optional cardinalities – An employee may not be assigned to a project. Every project has at least 1 employee assigned.



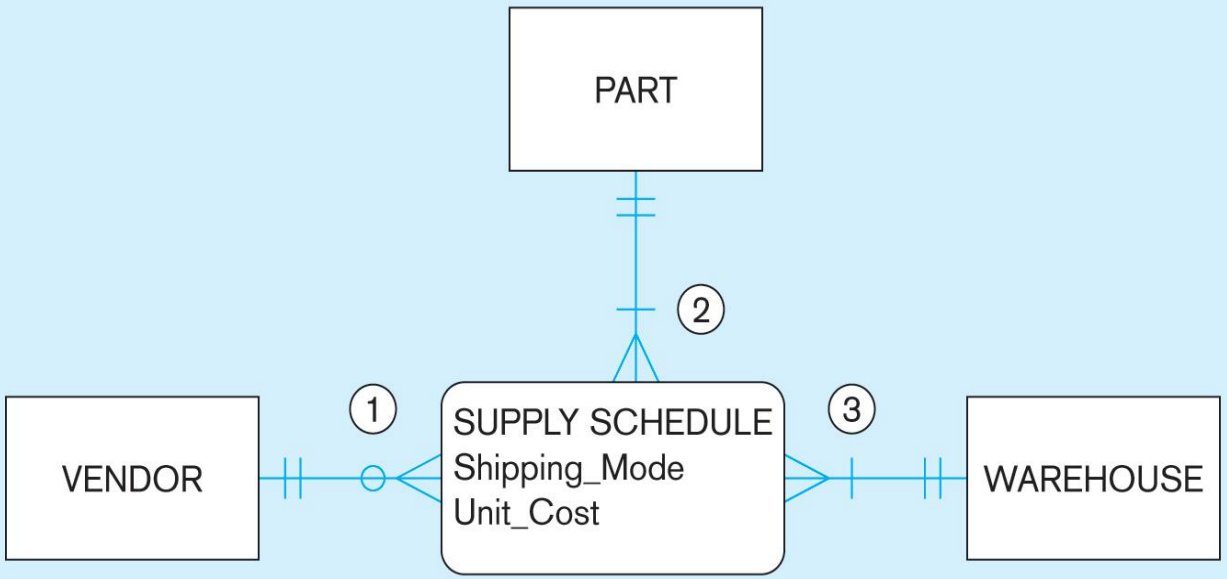
Cardinality Constraints

Optional cardinalities in a unary relationship – Not every person is married, but relationships are 1:1



Cardinality Constraints

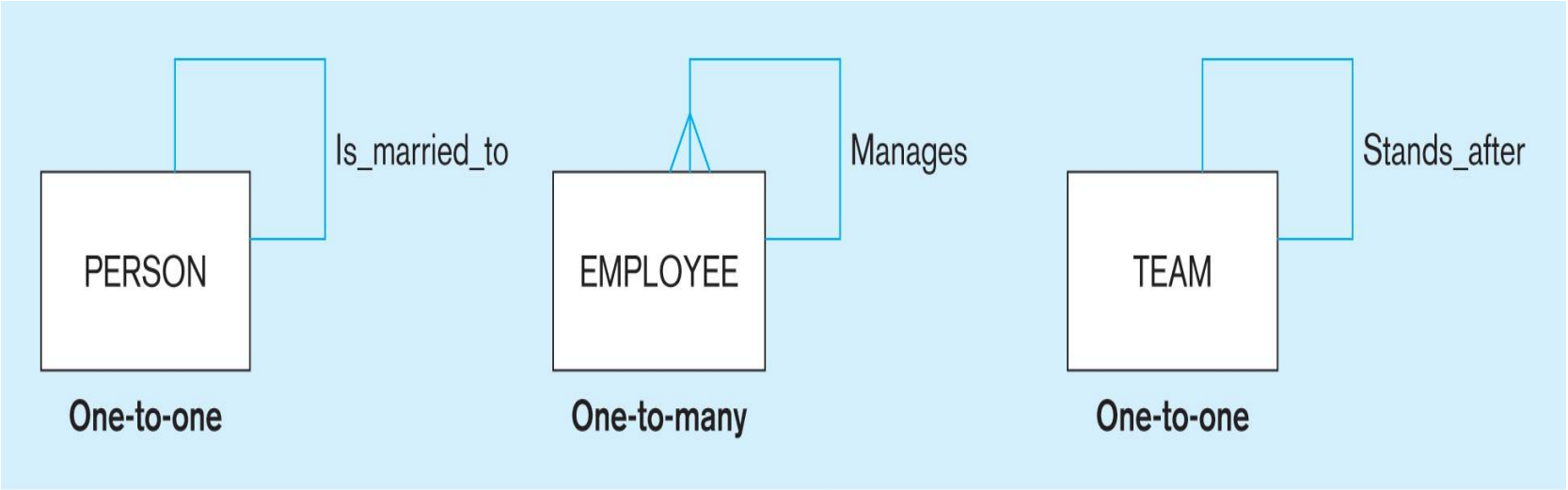
Cardinality constraints in a ternary relationship



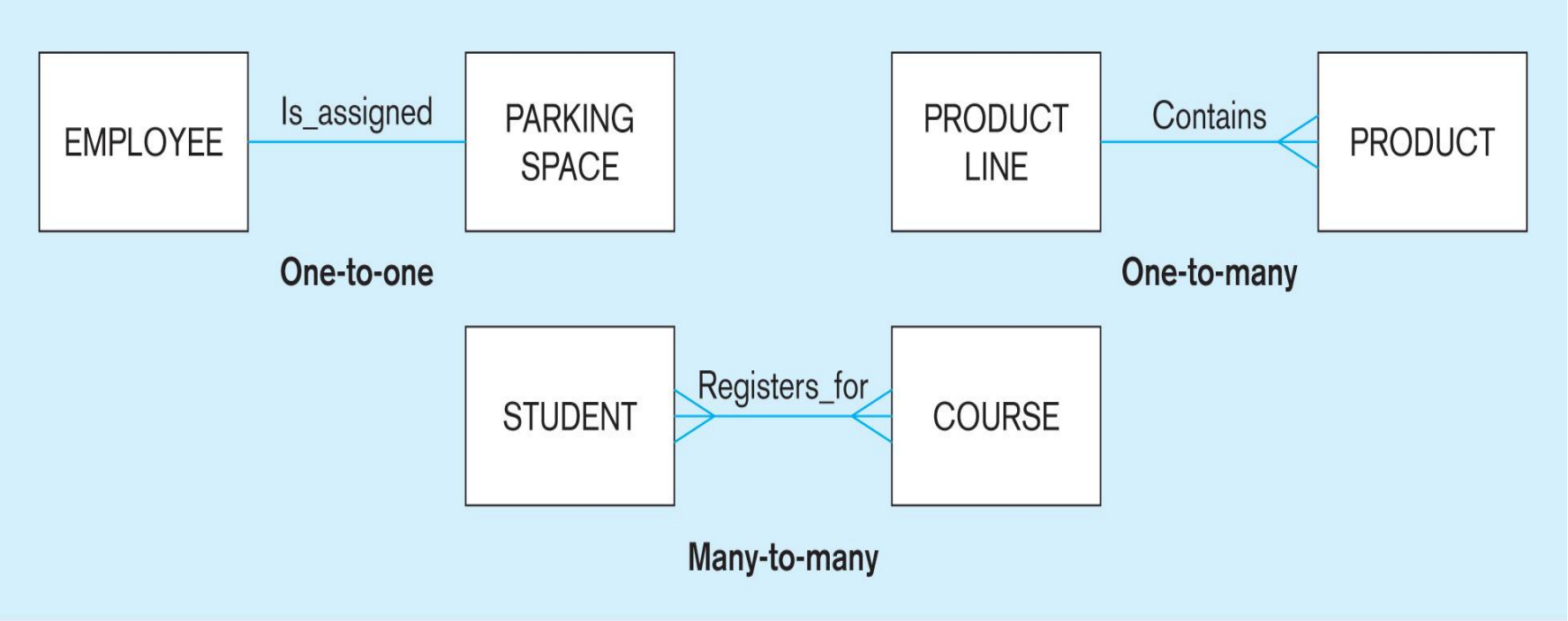
Business Rules

1. Each vendor can supply many parts to any number of warehouses, but need not supply any parts.
2. Each part can be supplied by any number of vendors to more than one warehouse, but each part must be supplied by at least one vendor to a warehouse.
3. Each warehouse can be supplied with any number of parts from more than one vendor, but each warehouse must be supplied with at least one part.

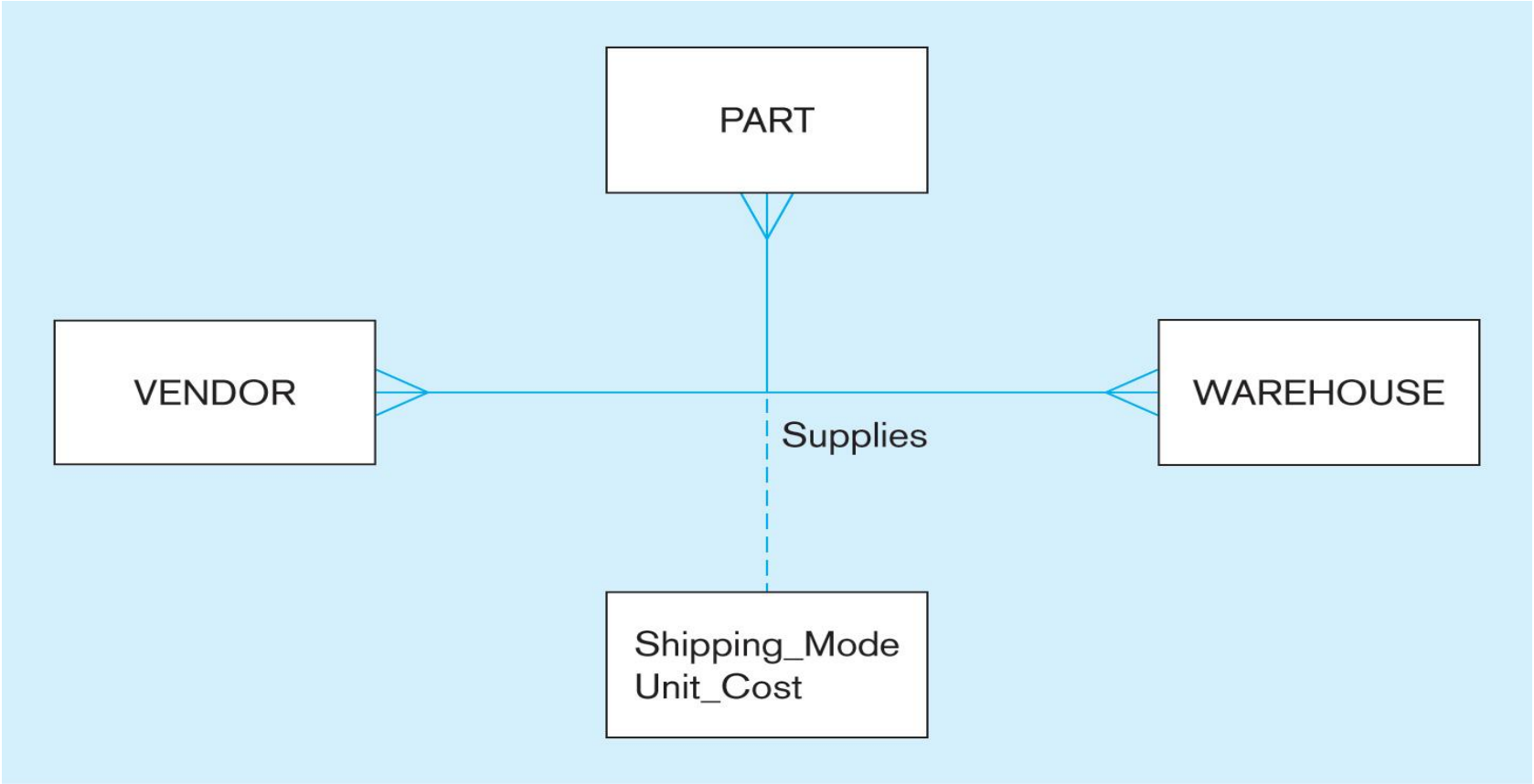
Unary Relationships



Binary Relationships



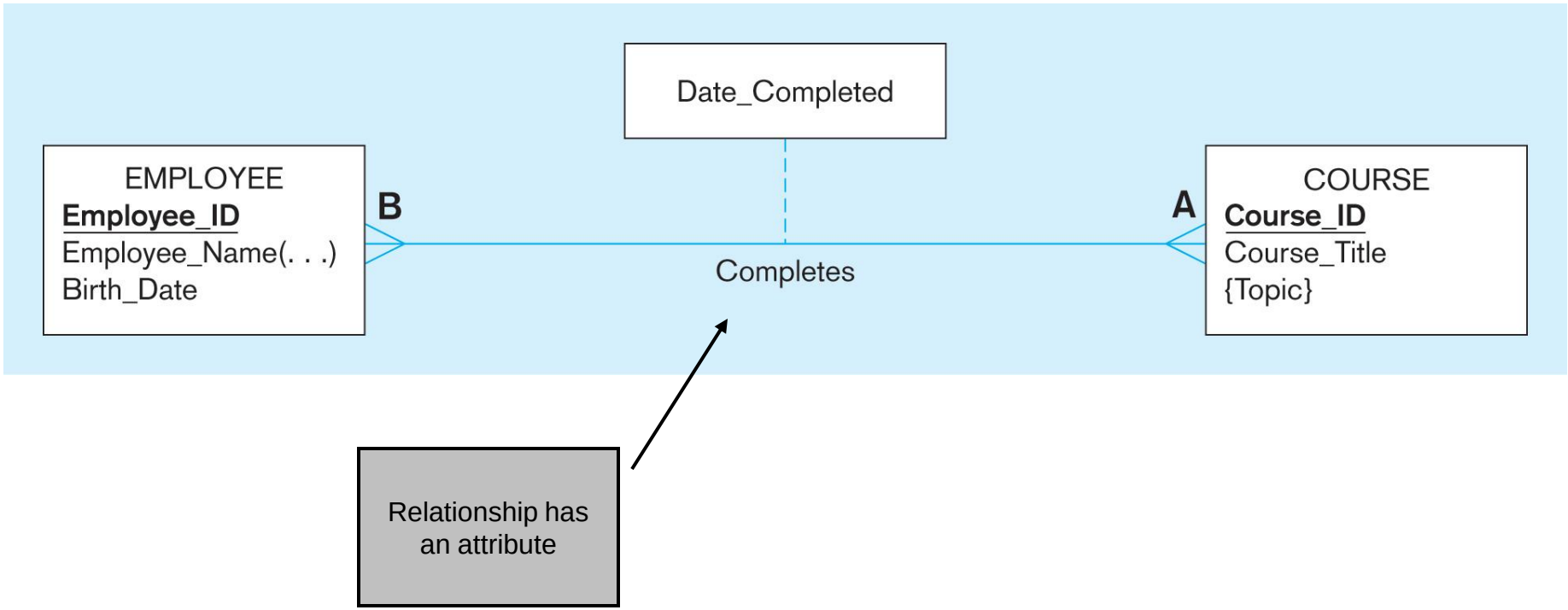
Ternary Relationships



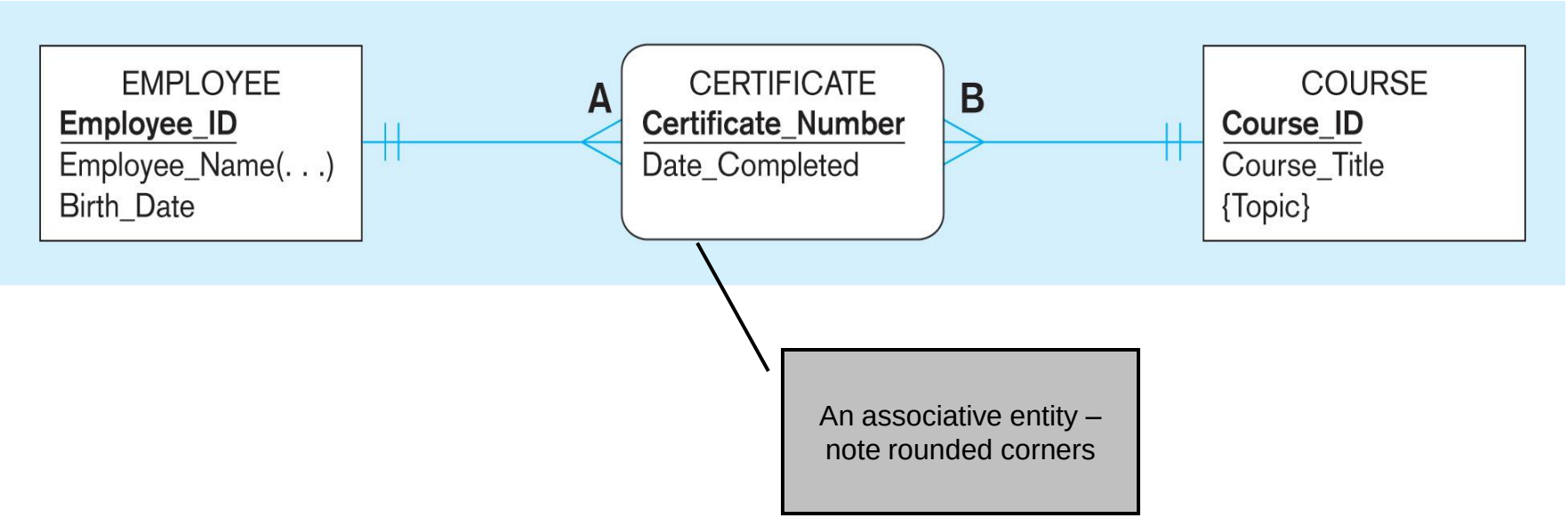
Associative Entities

- It's an entity – it has attributes; AND it's a relationship – it links entities together.
- When should a *relationship with attributes* instead be an *associative entity*?
  - All relationships for the associative entity should be many to many.
  - The associative entity could have meaning independent of the other entities.
  - The associative entity preferably has a unique identifier, and should also have other attributes.
  - The associative entity may participate in other relationships other than the entities of the associated relationship.
  - Ternary relationships should be converted to associative entities.

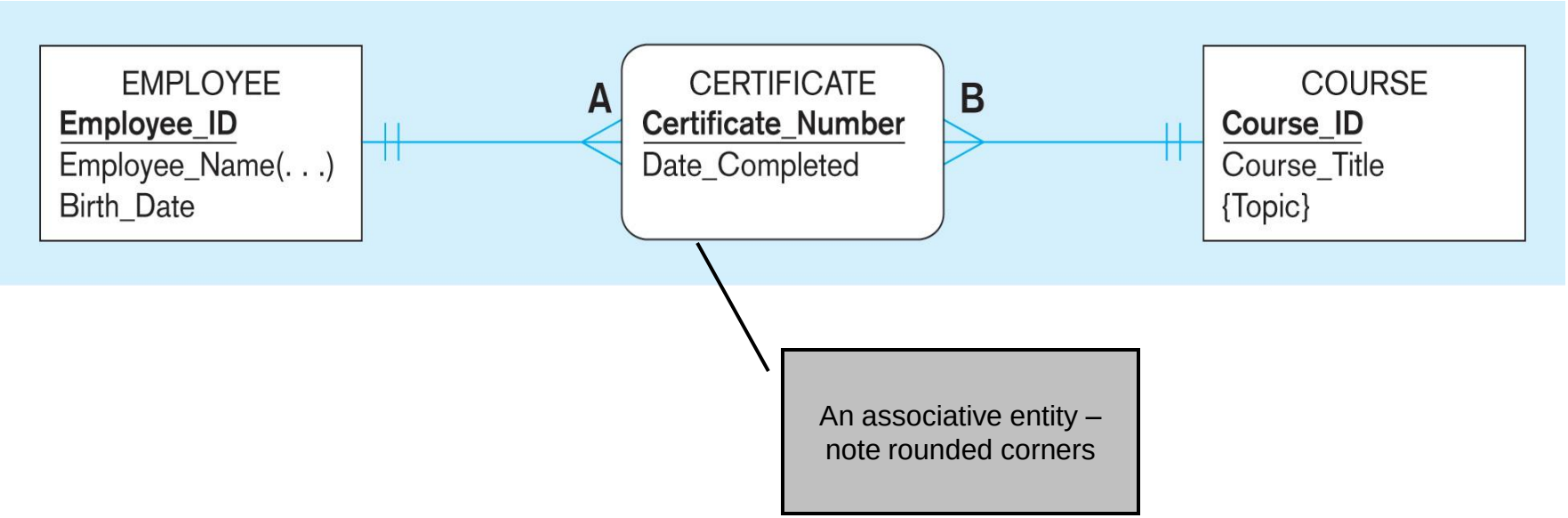
Associative Entities



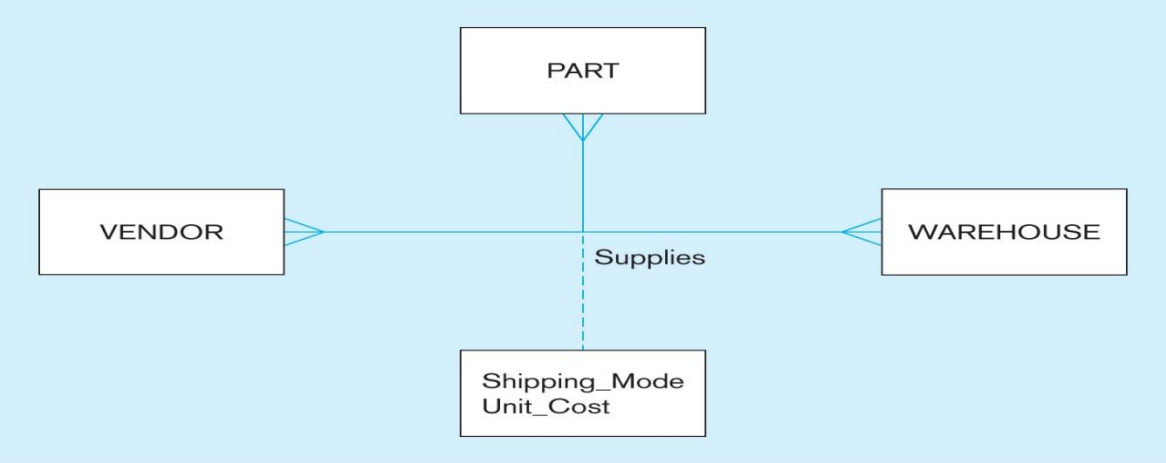
Associative Entities

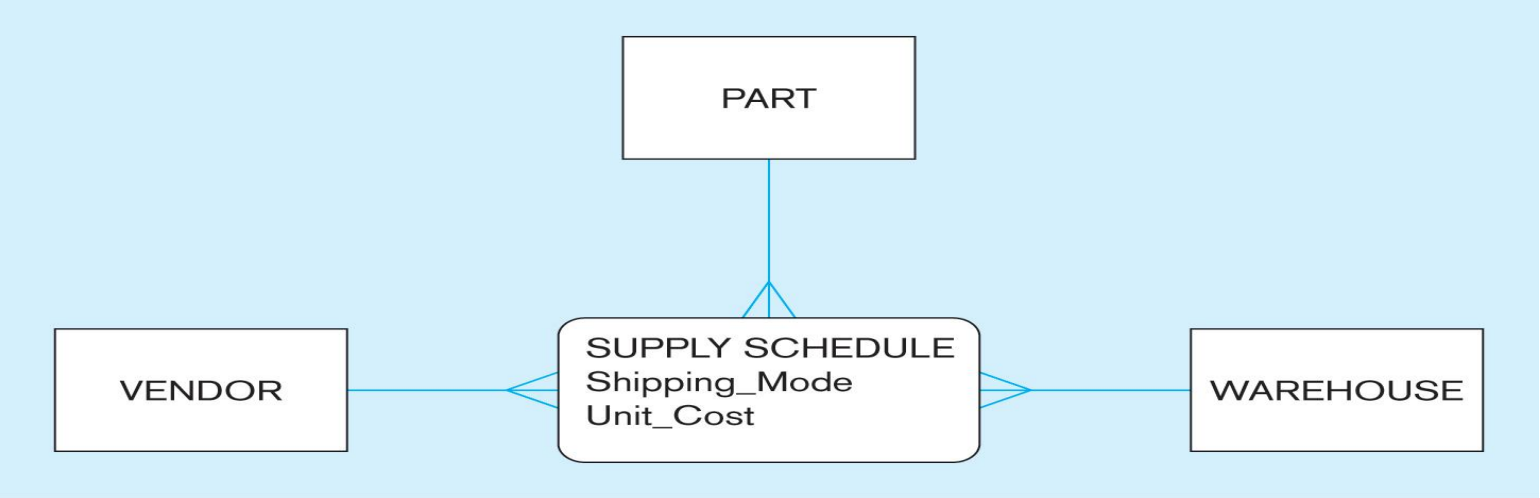


Associative Entities



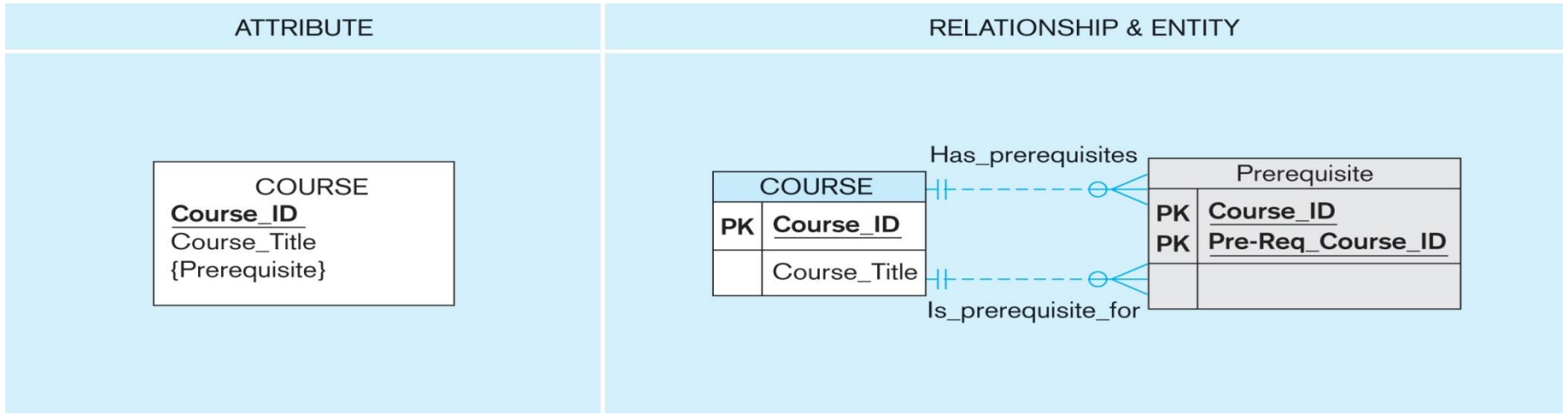
Ternary Relationship to Associative Entity





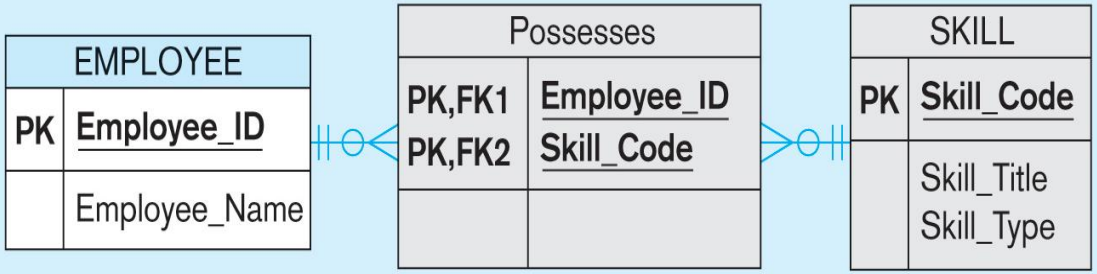
Using Relationships and Entities To Link Related Attributes

Multi-valued attribute as a relationship



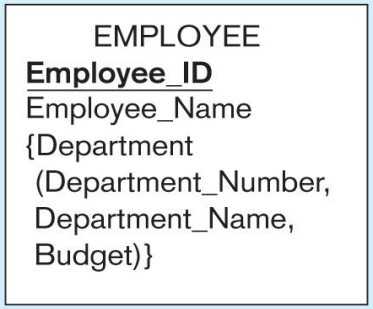
Using Relationships and Entities To Link Related Attributes

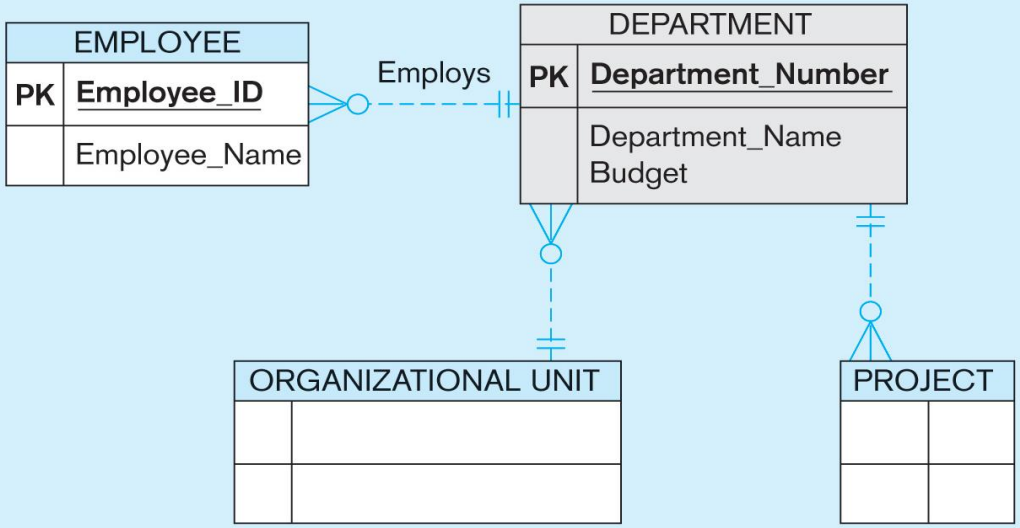
Composite, multi-valued attribute as a relationship



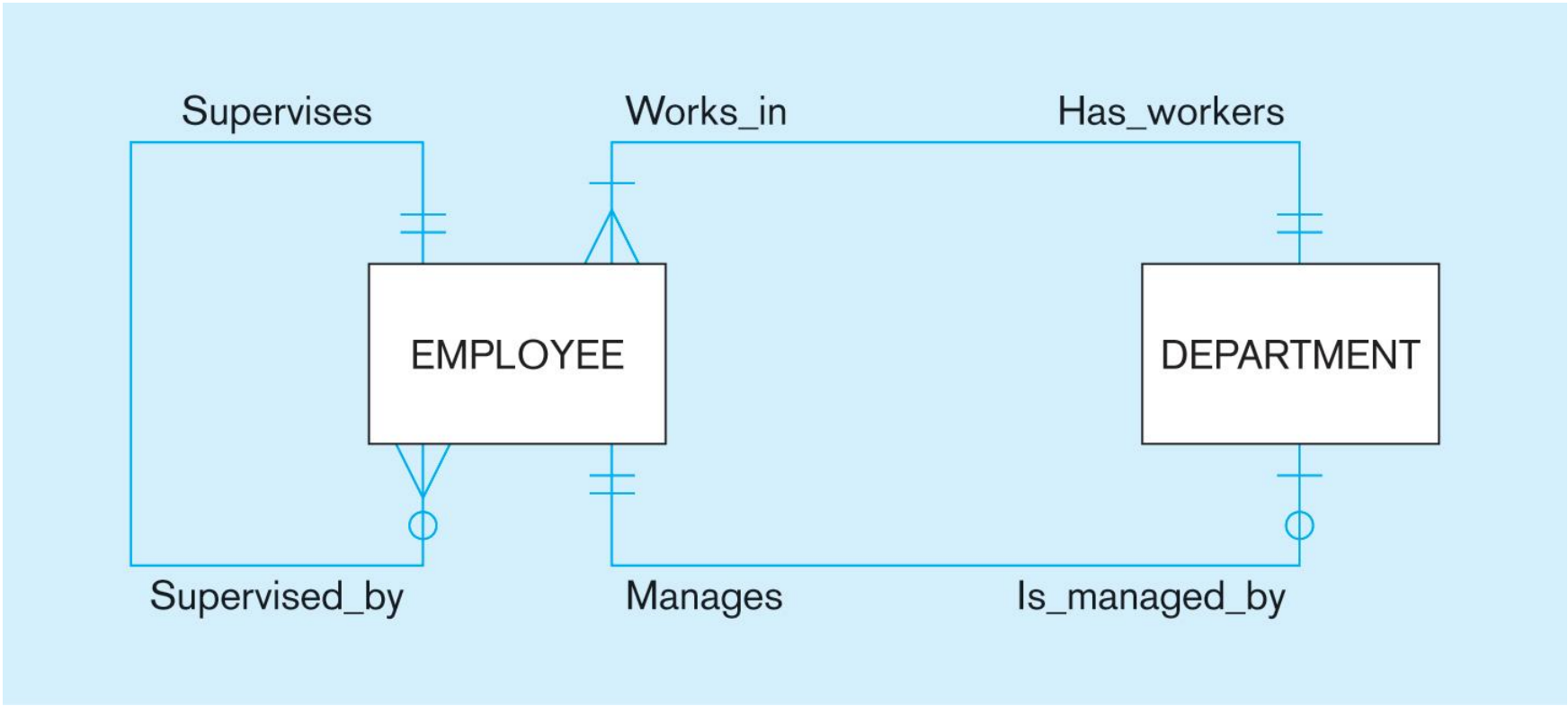
Using Relationships and Entities To Link Related Attributes

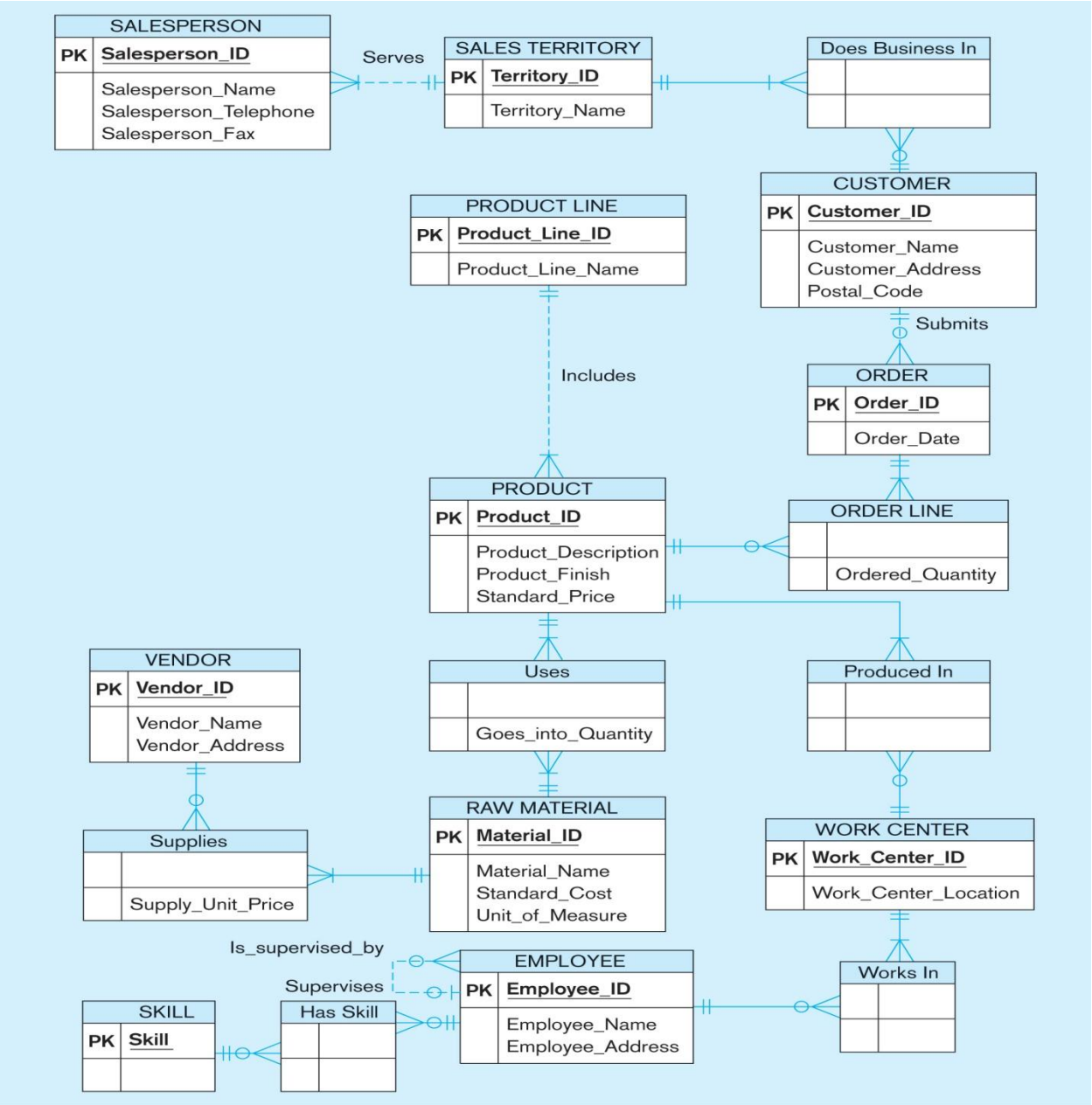
Composite attribute shared with other entities





Entities can be related to one another in more than one way



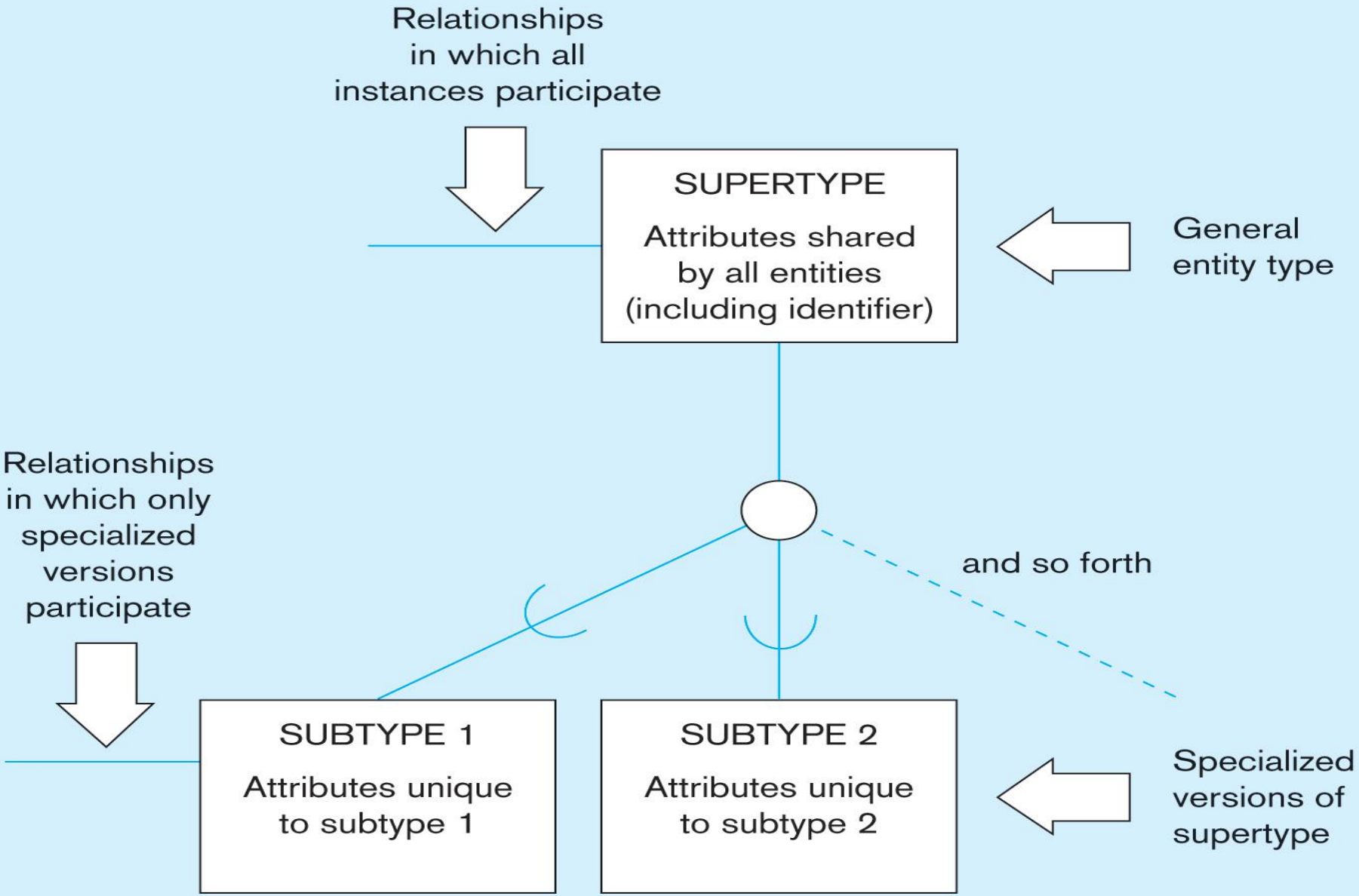


A more complex ERD

Different modeling software tools may have different notation for the same constructs

Supertypes and Subtypes

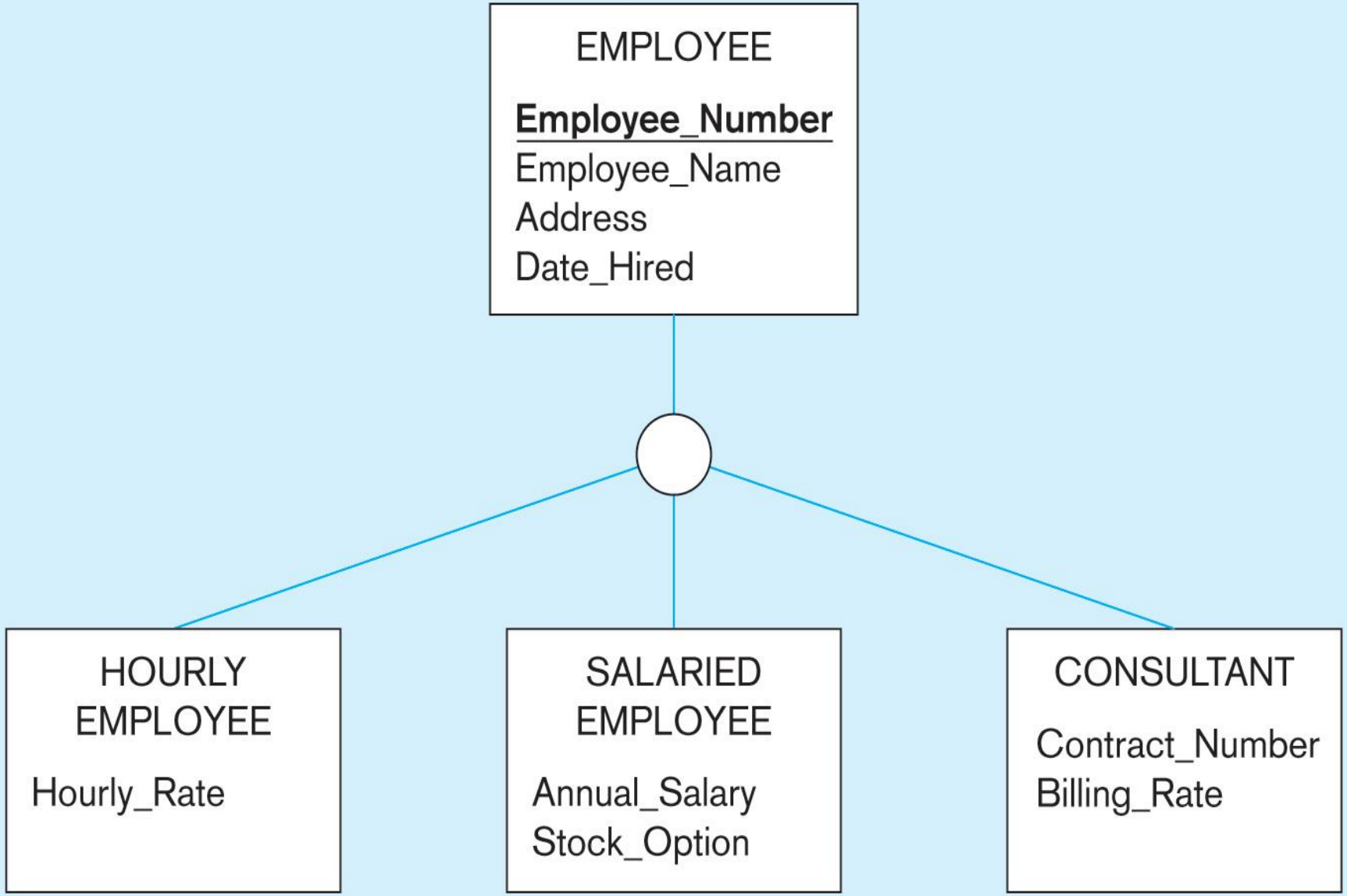
- Subtype: A subgrouping of the entities in an entity type which has attributes that are distinct from those in other subgroupings.
- Supertype: An generic entity type that has a relationship with one or more subtypes.
- Attribute Inheritance:
  - Subtype entities inherit values of all attributes of the supertype.
  - An instance of a subtype is also an instance of the supertype.


Relationships in which all instances participate
SUPERTYPE
Attributes shared by all entities (including identifier)
General entity type
Relationships in which only specialized versions participate
and so forth
SUBTYPE 1
Attributes unique to subtype 1
SUBTYPE 2
Attributes unique to subtype 2
Specialized versions of supertype

Relationships and Subtypes

- Relationships at the **supertype** level indicate that all subtypes will participate in the relationship.

- The instances of a **subtype** may participate in a relationship unique to that subtype. In this situation, the relationship is shown at the subtype level.


EMPLOYEE
Employee_Number
Employee_Name
Address
Date_Hired
HOURLY EMPLOYEE
Hourly_Rate
SALARIED EMPLOYEE
Annual_Salary
Stock_Option
CONSULTANT
Contract_Number
Billing_Rate

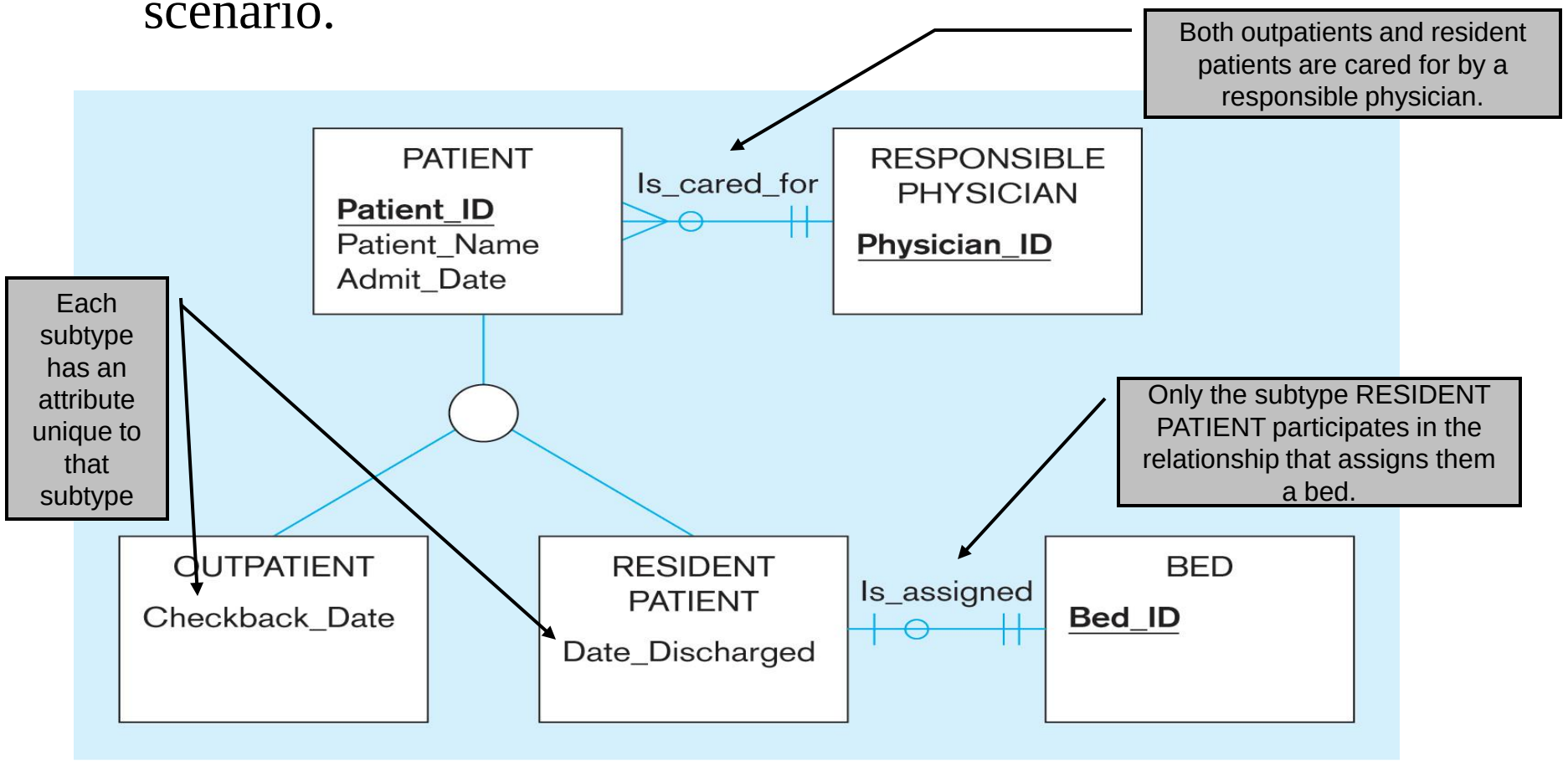
When To Use Supertype/Subtype Relationships

- Whether to use supertype/subtype relationships is a decision the data modeler must make in each situation.
- You should consider using subtypes when either (or both) of the following conditions are present:

1. There are attributes that apply to some (but not all) instances of an entity type. See the example on the previous page.
2. The instances of a subtype participate in a relationship that is unique to the subtype, i.e., other subtypes do not participate in the relationship.

When To Use Supertype/Subtype Relationships

- As an example of when to use subtypes, consider the following scenario.

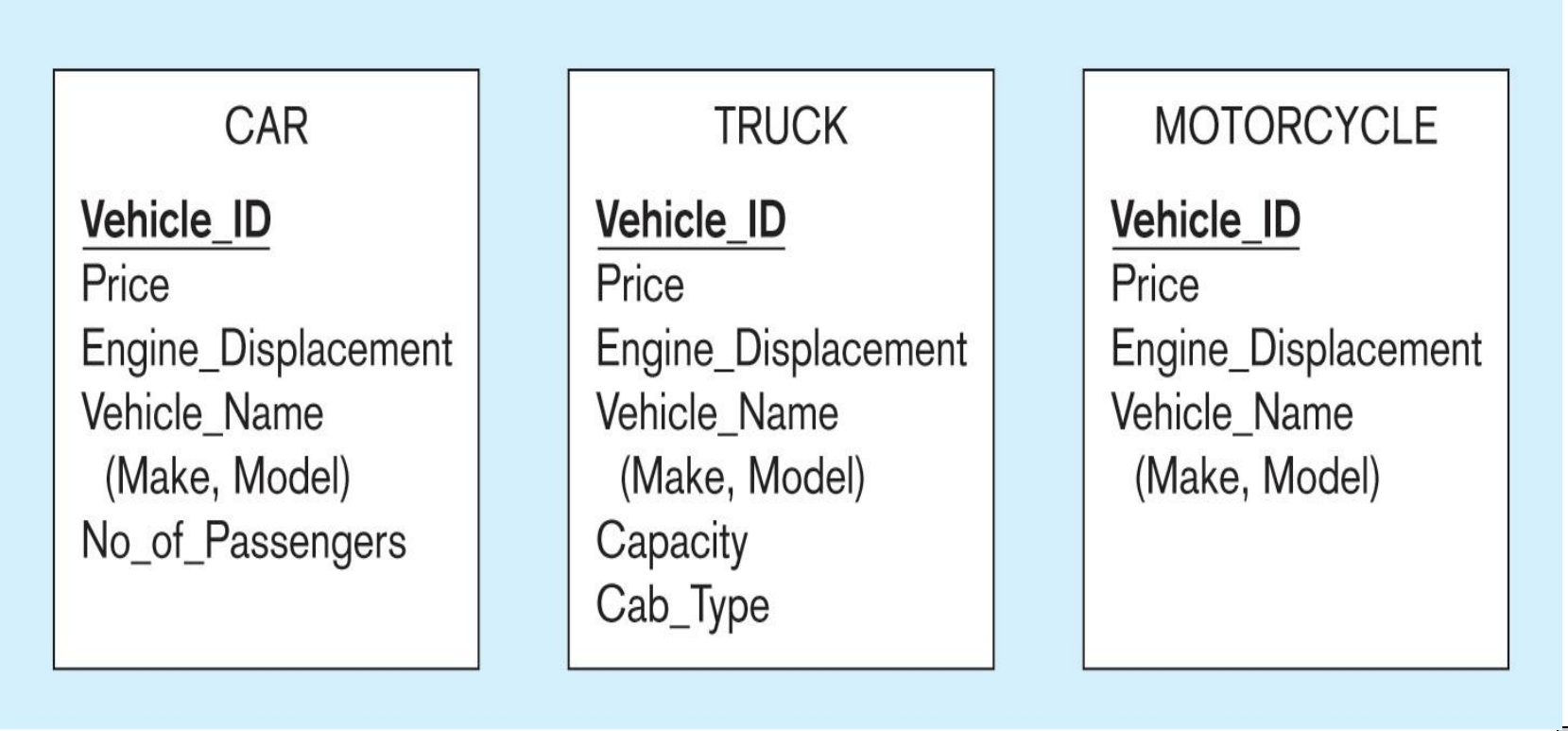


Generalization and Specialization

- **Generalization:** The process of defining a more general entity type from a set of more specialized entity types.
  - This is a BOTTOM-UP approach to design.
- **Specialization:** The process of defining one or more subtypes of the supertype, and forming supertype/subtype relationships.
  - This is a TOP-DOWN approach to design.

Generalization

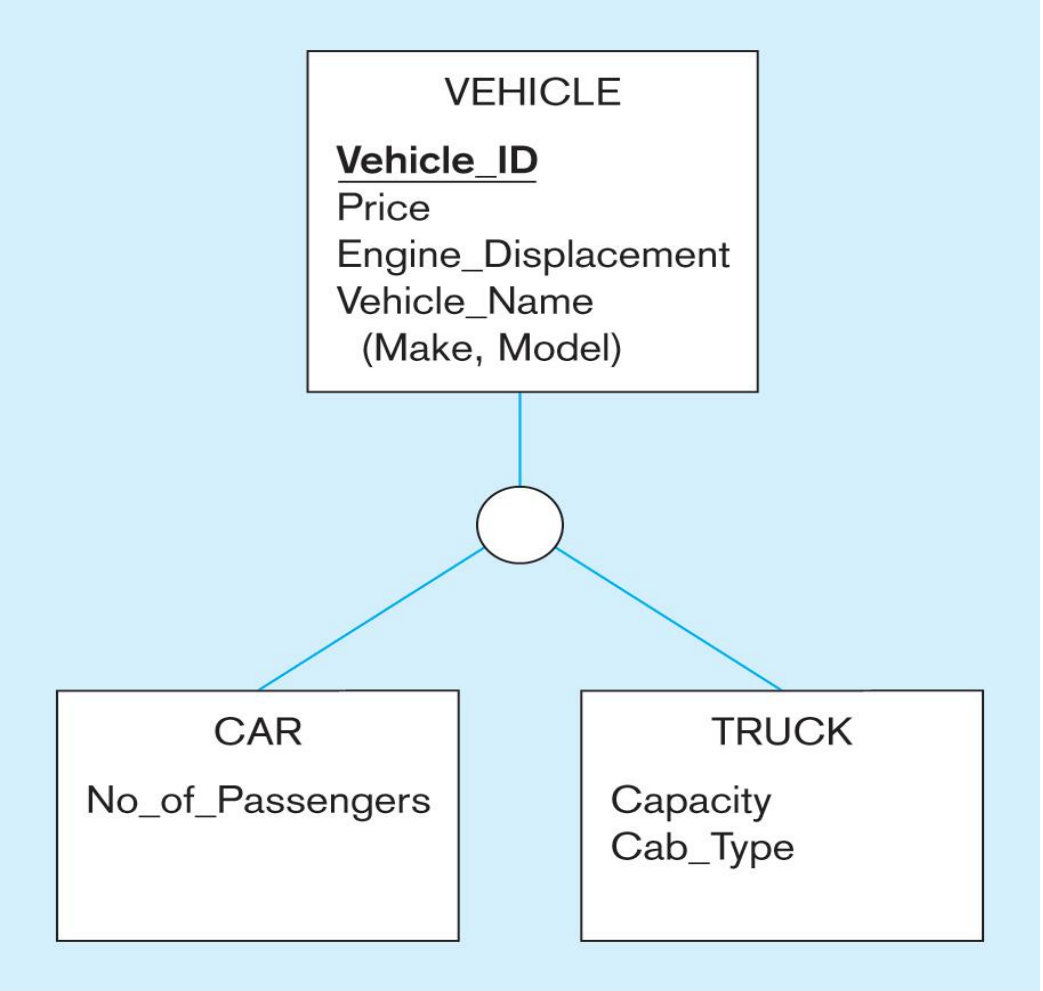
- The data modeler has identified three entity types.
- Notice the similarities and differences amongst these types.



Generalization

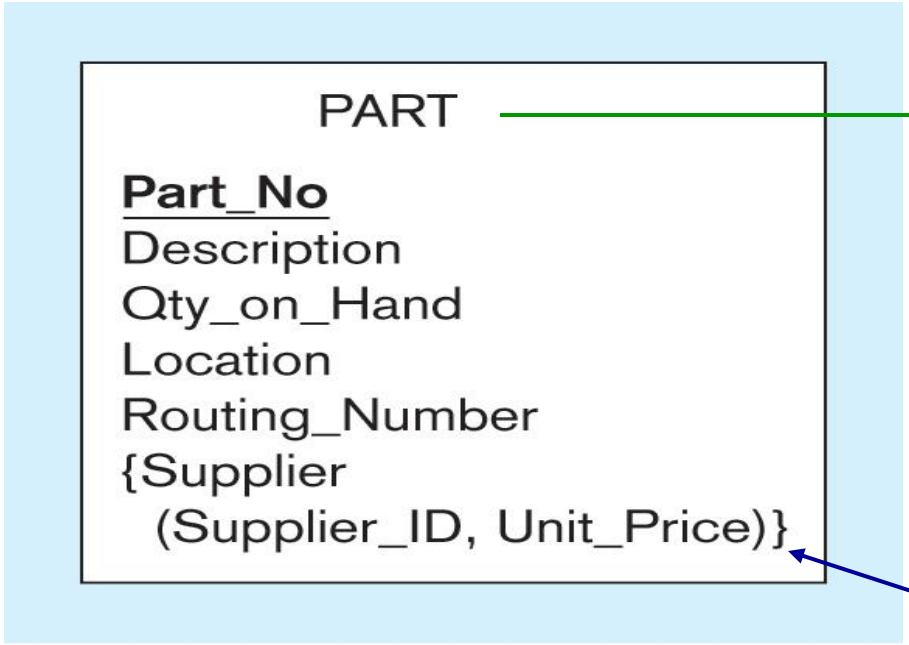
Question: What happened to the motorcycle entity type?

Answer: Since the class does not satisfy the conditions for developing a subtype. The type has no unique attributes and does not participate in any unique relationships. Therefore, motorcycles are simply vehicles without any specialization.

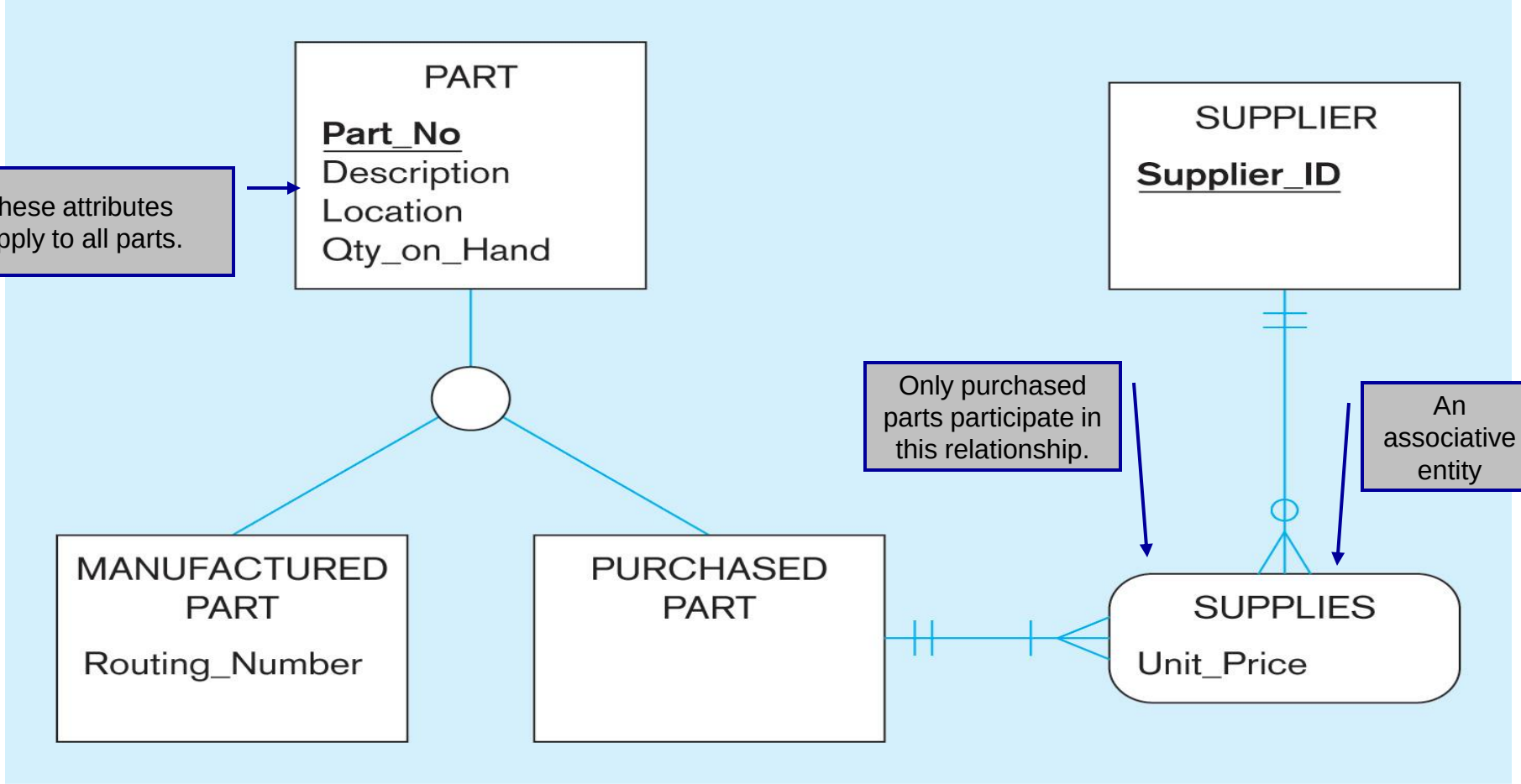


Specialization

• The data modeler has identified an entity type that contains a multi-valued attribute. Some of the attributes apply to all parts regardless of the source, while some of the attributes depend on the source.



Specialization



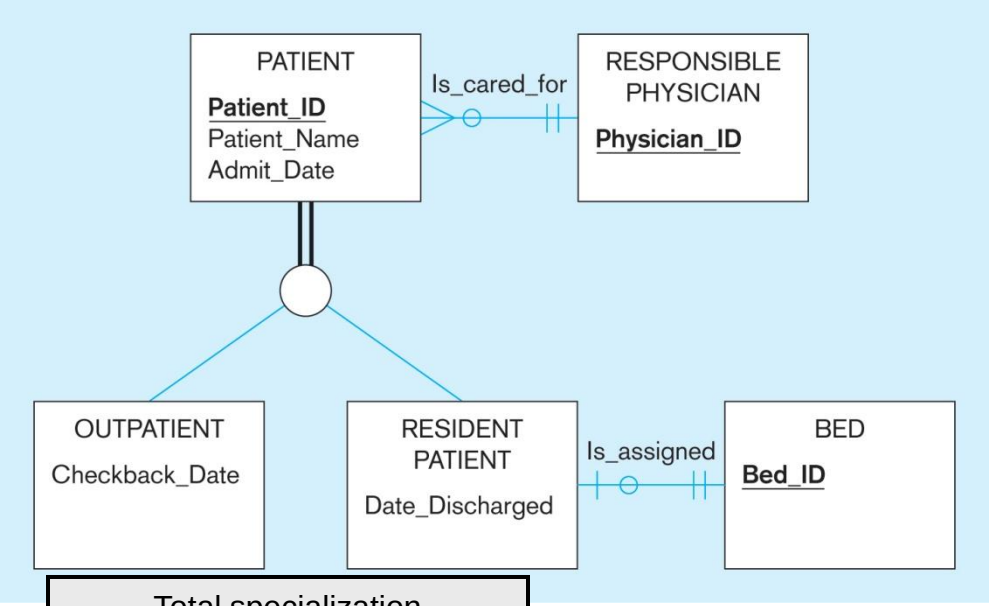
Constraints in Supertype/Subtype Relationships

Completeness Constraints

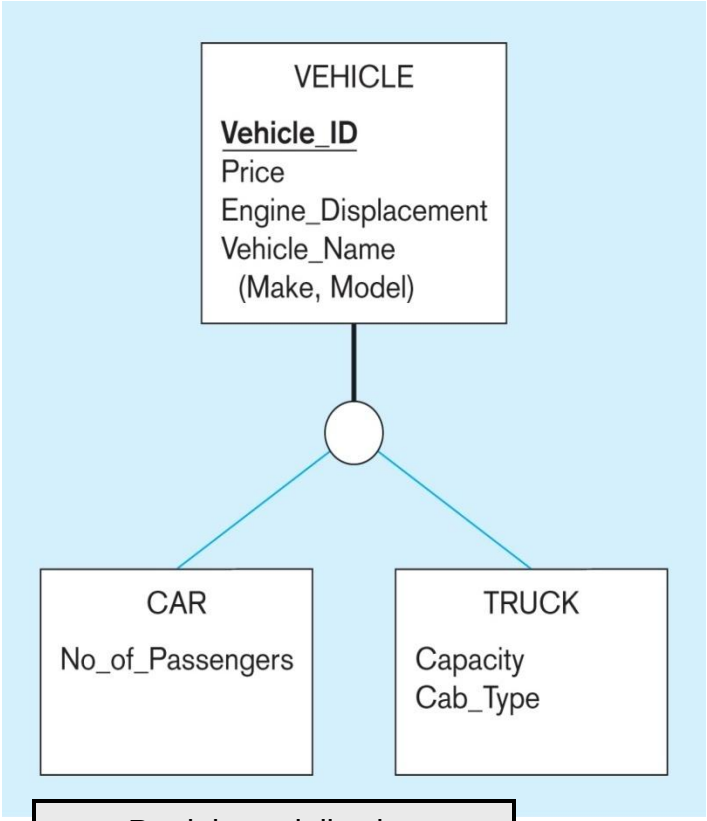
- A completeness constraint specifies whether an instance of a supertype ***must*** also be a member of at least one subtype. There are two possible cases:
  - Total Specialization Rule
    - All instances in the supertype must also be a member of at least one subtype. Represented by a double line from the supertype to the subclass split (see next page).
  - Partial Specialization Rule
    - Some instances in the supertype may not be members of any subtype. Represented by a single line (see next page).

Constraints in Supertype/Subtype Relationships

Completeness Constraints



Total specialization
A patient must either be an outpatient or a resident patient.



Partial specialization
A vehicle may be either a car or a truck or neither.

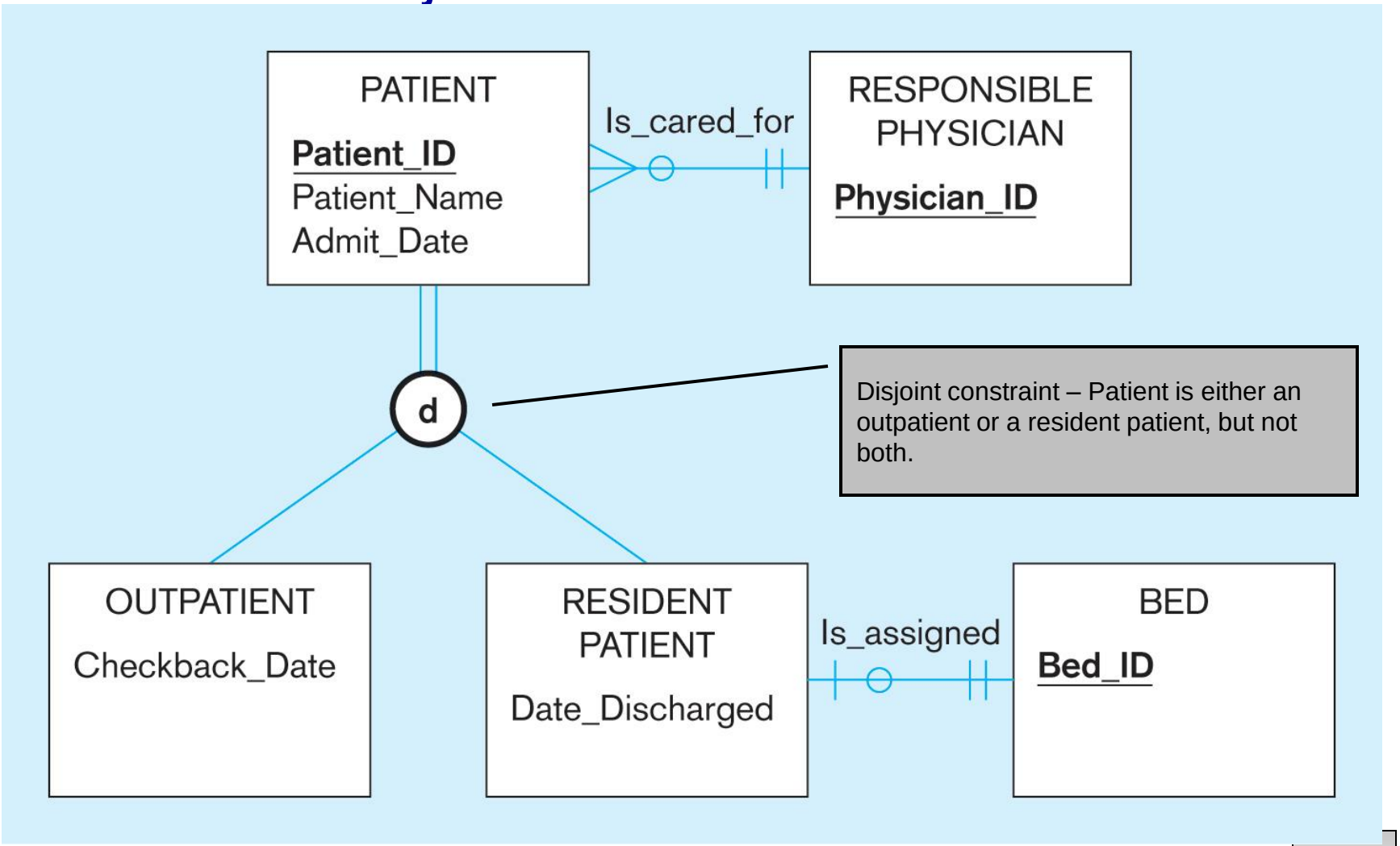
Constraints in Supertype/Subtype Relationships

Disjointness Constraints

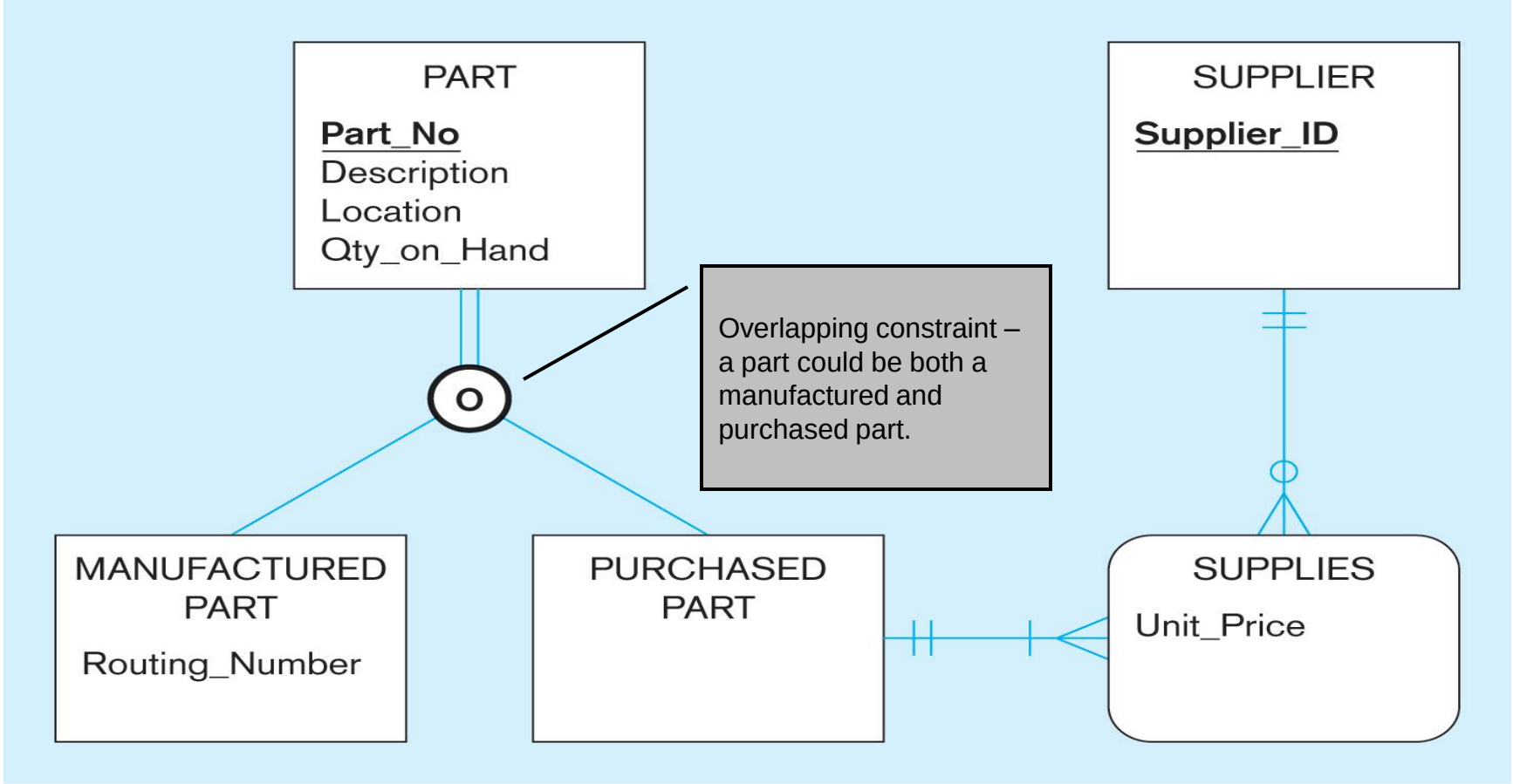
- Whether an instance of a supertype may *simultaneously* be a member of two (or more) subtypes. Again, two rules apply:
  - Disjoint Rule
    - An instance of the supertype can be only ONE of the subtypes. The letter "D" is placed in the category circle.
  - Overlap Rule
    - An instance of the supertype could be more than one of the subtypes. The letter "O" is placed in the category circle.

Constraints in Supertype/Subtype Relationships

Disjointness Constraints



Constraints in Supertype/Subtype Relationships
Disjointness Constraints



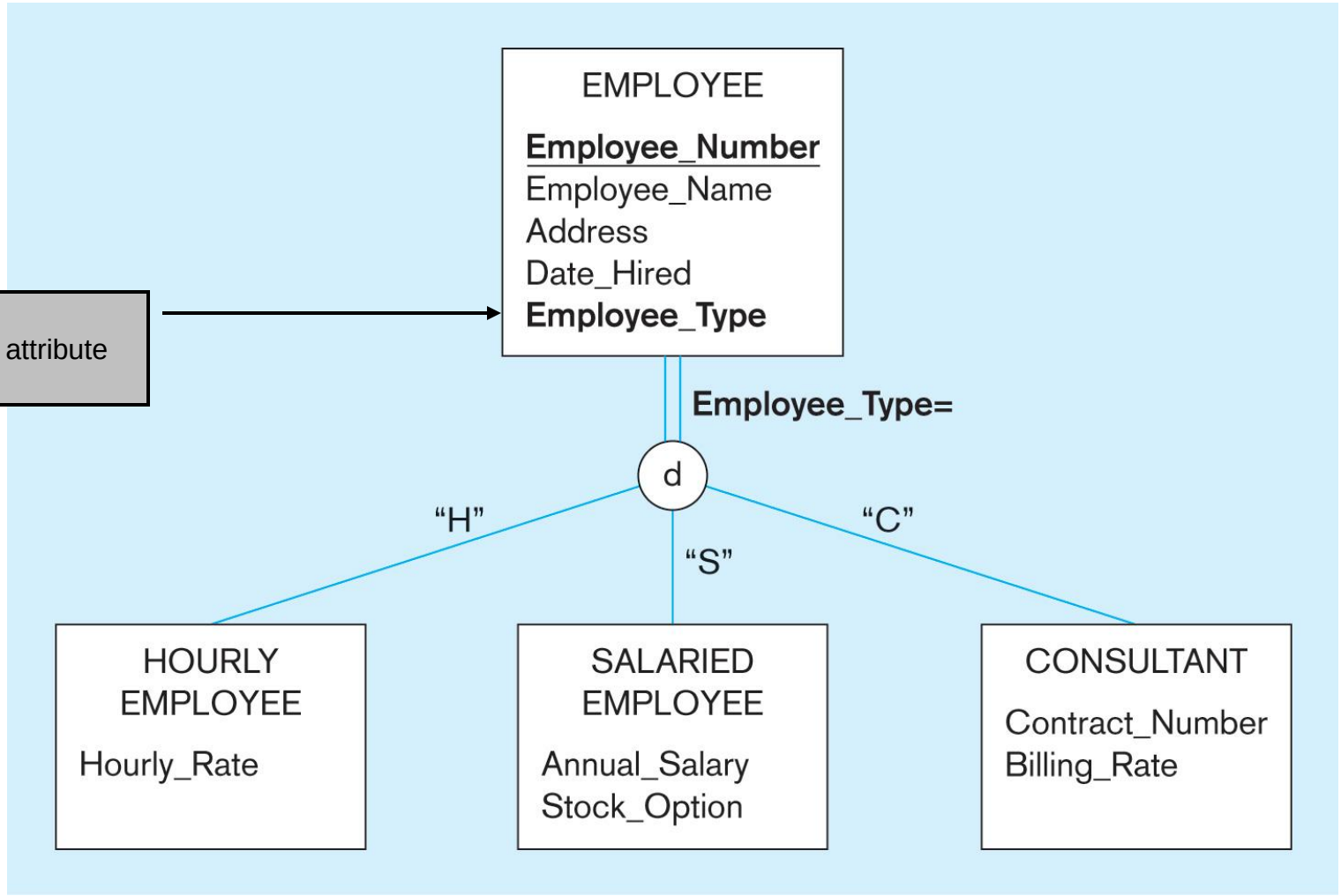
Defining Subtype Discriminators

Subtype Discriminator

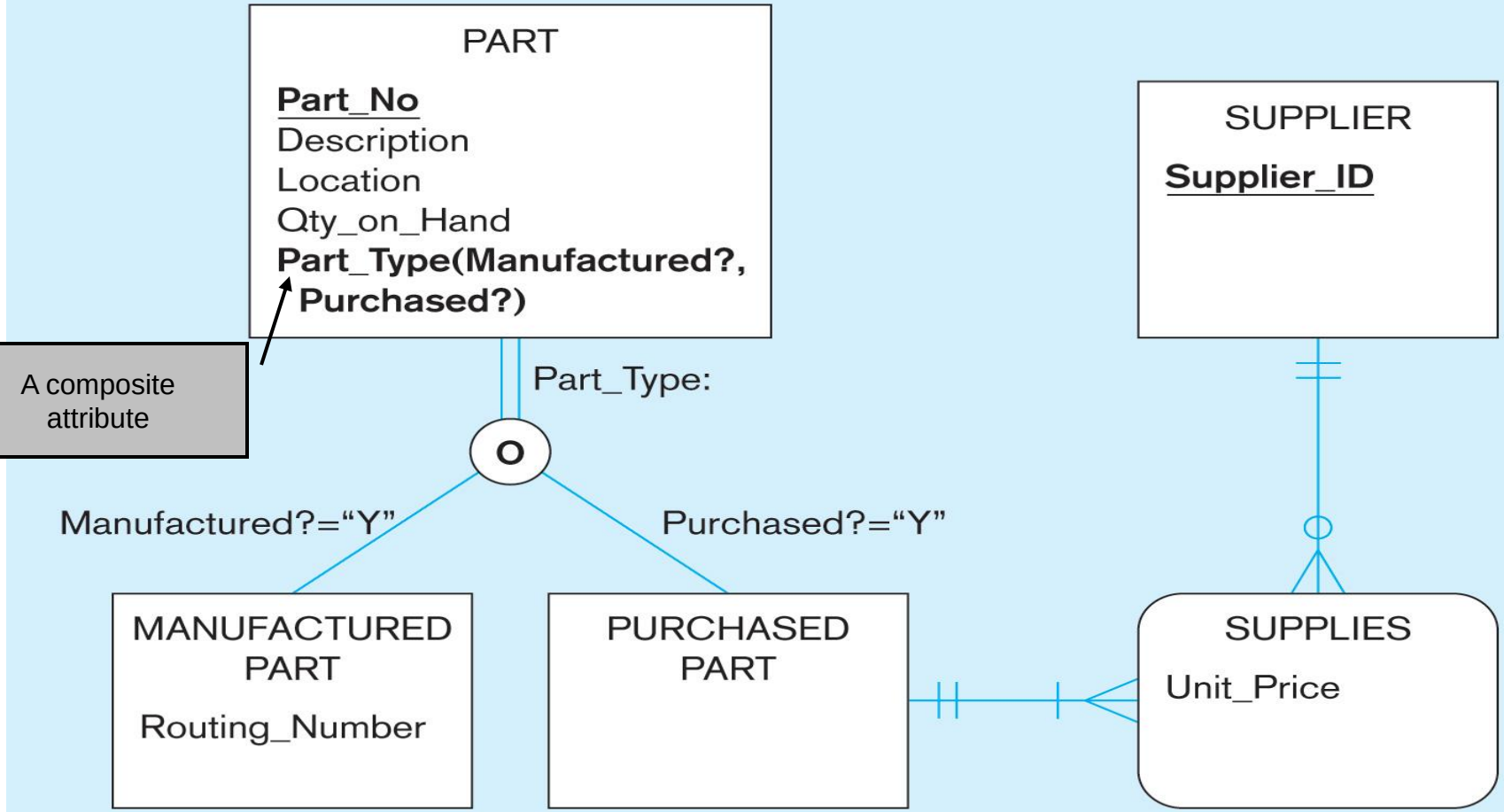
- An attribute of the supertype whose values determine the target subtype(s):
  - Disjoint – a *simple* attribute with alternative values to indicate the possible subtypes.
  - Overlapping – a *composite* attribute whose subparts pertain to different subtypes. Each subpart contains a boolean value to indicate whether or not the instance belongs to the associated subtype.

Defining Subtype Discriminators

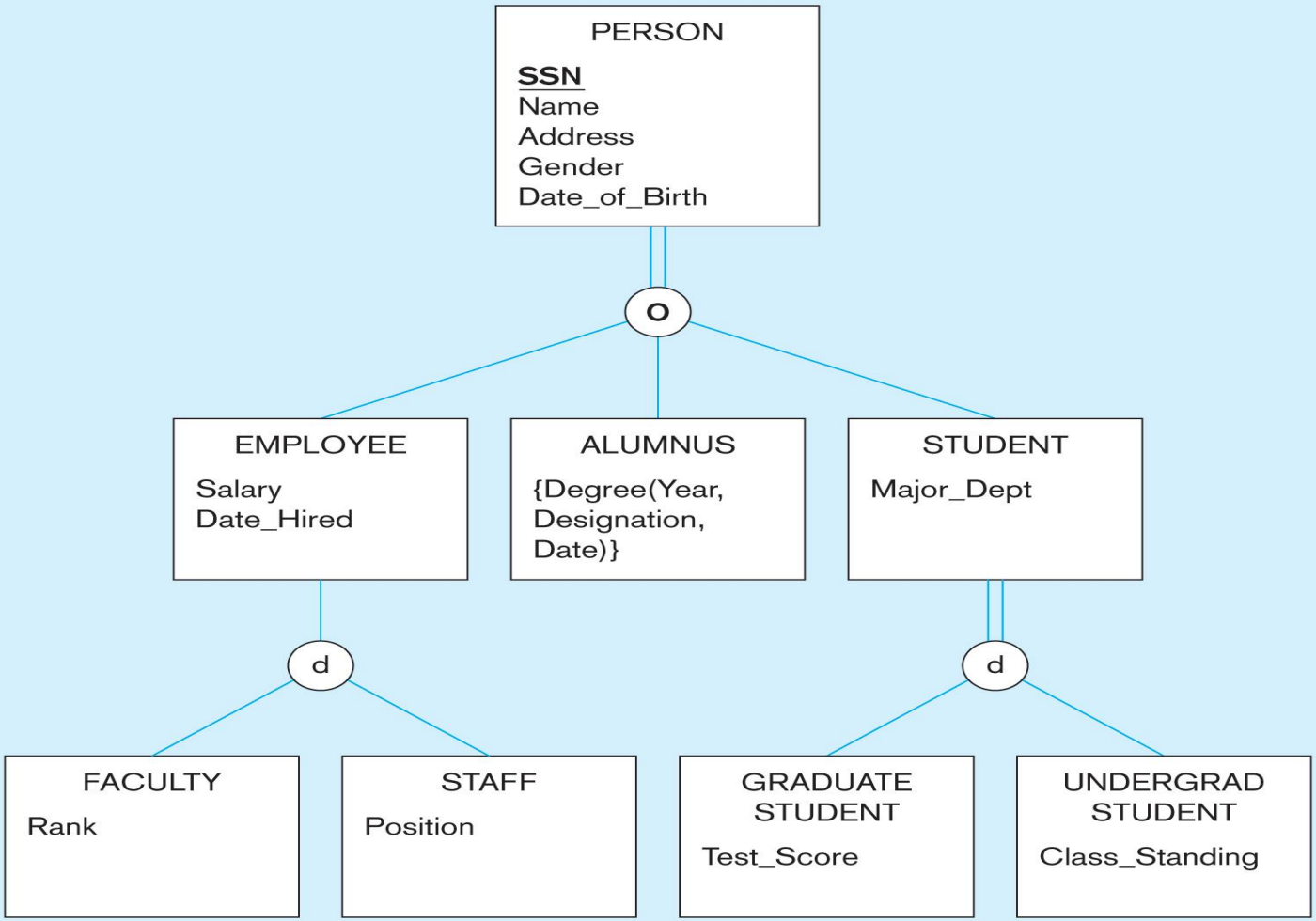
Disjoint



Defining Subtype Discriminators
Overlapping



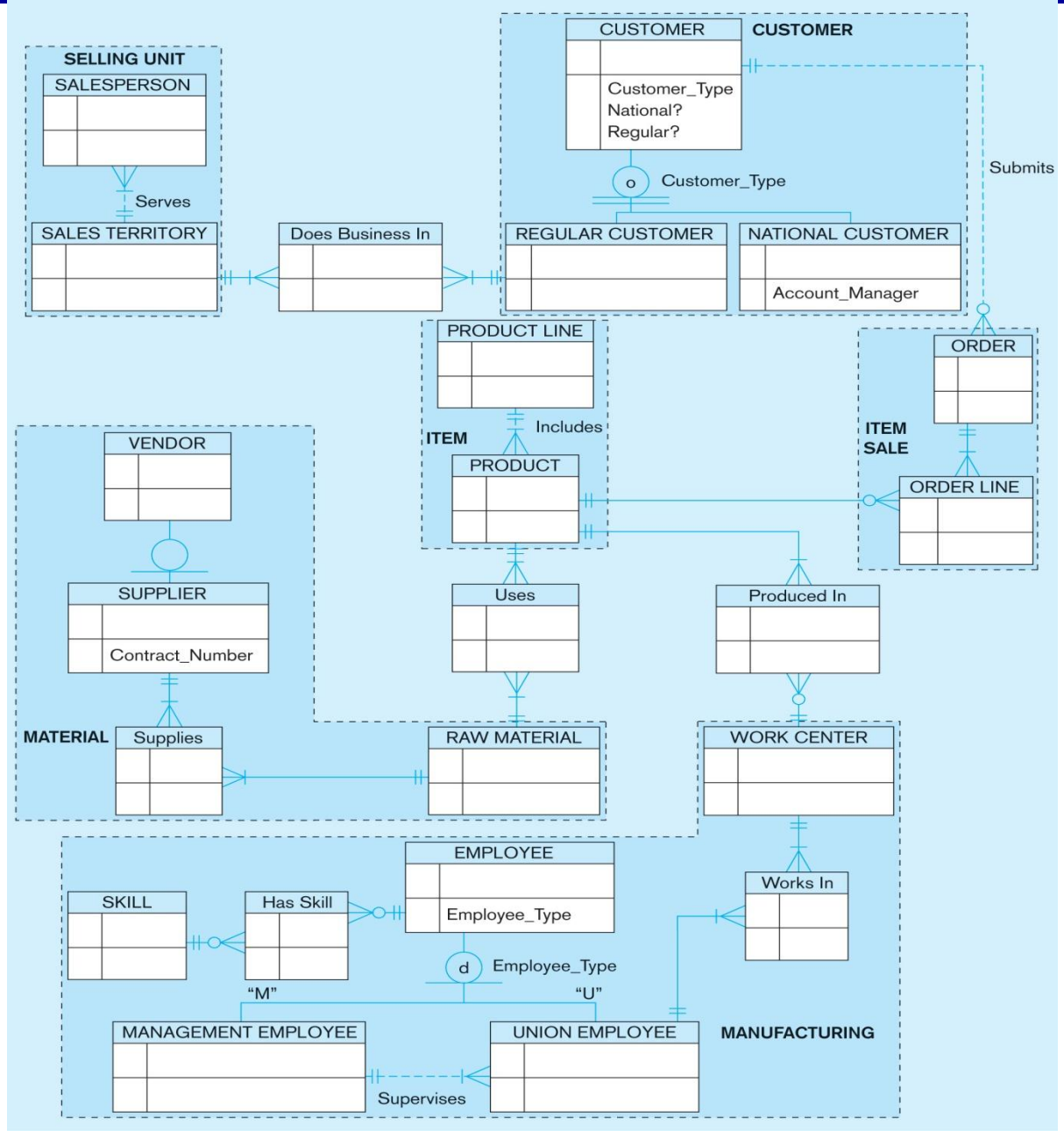
Supertype/Subtype Hierarchies



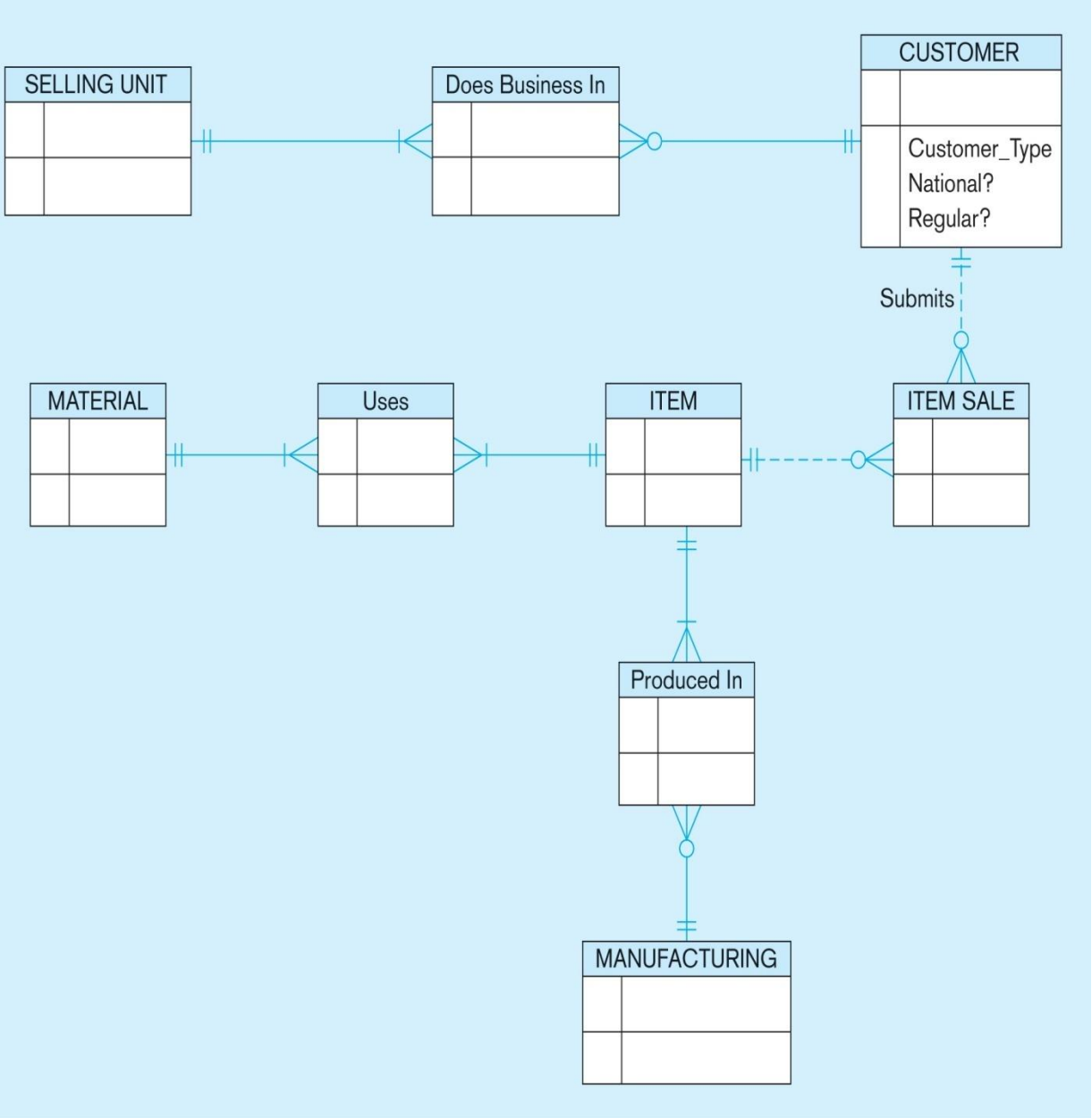
Entity Clusters

- EER diagrams are difficult to read when there are too many entities and relationships.
- Solution: group entities and relationships into entity clusters.
- Entity cluster: set of one or more entity types and associated relationships grouped into a single abstract entity type.

Entity Clusters



Entity Clusters



Attributes in the E-R Model

- *Null*: An attribute takes a null value when an entity does not have a value for it. Null values are usually special cases that can be handled in a number of different ways depending on the situation.
  - For example, it could be interpreted to mean that the attribute is "not applicable" to this entity, or it could mean that the entity has a value for this attribute but we don't know what it is. We will see later in the term how different systems handle null values and the different interpretations that may be associated with this special value.

Relationships in the E-R Model

- A *relationship* is an association among several entities.
  - For example, we can define a relationship that associates you as a student in COP 4710. This relationship might specify that you are *enrolled* in this course.

A *relationship set* is a set of relationships of the same type.
More formally, it is a mathematical relation on n ≥ 2 (possibly non distinct) entity sets.

If E_1, E_2, …, E_n are entity sets, then a relationship set *R* is a subset of:

$$\{(e_1, e_2, \cdots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \cdots, e_n \in E_n\}$$

where $(e_1, e_2, \cdots, e_n)$ is the relationship.

Relationships in the E-R Model (cont.)

- The association between entity sets is referred to as participation; that is, the entity sets E_1, E_2, …, E_n participate in relationship *R*.
- A *relationship instance* in an E-R schema represents an association between named entities in the real world enterprise which is being modeled.
- A relationship may also have attributes which are called *descriptive attributes*. For example, considering the bank scenario again, suppose that we have a relationship set *depositor* with entity sets *customer* and *account*. We might want to associate with the *depositor* relationship set a descriptive attribute called *access-date* to indicate the most recent date that a customer accessed their account.
-

Keys of an Entity Set

- We must have some mechanism for specifying how entities within a given entity set are distinguished.
- Conceptually, individual entities are distinct; from a database perspective, however, the differences among them must be expressed in terms of their attributes. Therefore, the values of the attribute values of an entity must be such that they can *uniquely identify* the entity. In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes.
- A *key* allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from one another.

Primary Keys, SuperKeys and Candidate Keys

- A *superkey* is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. Suppose that we have an entity set modeling the students in COP 4710. Suppose that we have the following schema for this entity set:

  Students(SS#, name, address, age, major, minor, gpa, spring-sch)

- Among the attributes which we have associated with each student must be a set of attributes which will uniquely distinguish each student. Suppose that we define this set of attributes to be:

  (SS#, name, major, minor)

Primary Keys, SuperKeys and Candidate Keys
(cont.)

- This set of attributes (SS#, name, major, minor) defines a superkey for the entity set Students. Notice that the set of attributes (SS#, name) also defines a superkey for this entity set, because given this second set of attributes we can still uniquely distinguish each student in the set. The concept of a superkey is not a sufficient definition of a key because the superkey, as we can see from this example, may contain extraneous attributes.

Primary Keys, SuperKeys and Candidate Keys (cont.)

- If the set *K* is a superkey of entity set *E*, then so too is any superset of *K*. We are interested only in superkeys for which no proper subset of *K* is a superkey. Such a minimal superkey is called a *candidate key*.
- For a given entity set *E* it is possible that there may be several distinct sets of attributes which are candidate keys.
- Either there is only a single such set of attributes or there are several distinct sets from which only one is selected by the database designer and this set of attributes defines the *primary key* which is typically referred to simply as the *key* of the entity set.

Primary Keys, SuperKeys and Candidate Keys
(cont.)

- A key (primary, candidate, and super) is a property of the entity set, rather than of the individual entities. Any two individual entities in the set are prohibited from having the same value on all attributes which comprise the key attributes at the same time. This constraint on the allowed values of an entity within the set is a *key constraint*.

- The database designer must use care in the selection of the set of attributes which comprise the key of an entity set to: (1) be certain that the set of attributes guarantees the uniqueness property, and (2) be certain that the set of key attributes are never, or very rarely, changed.

Relationship Sets

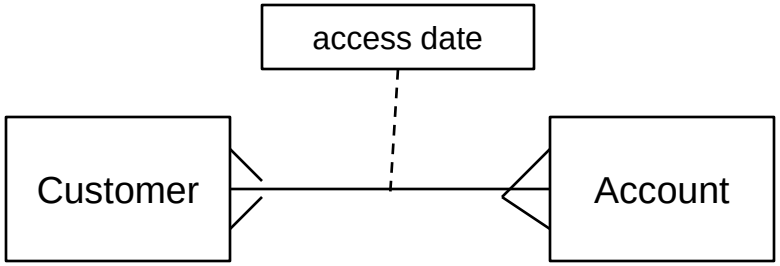
- The primary key of an entity set allows us to distinguish among the various entities in the set. There must be a similar mechanism which allows us to distinguish among the various relationships in a relationship set.
- Let R be a relationship set involving entity sets E_1, E_2, …, E_n. Let K_i denote the set of attributes which comprise the primary key of entity set E_i. For now lets assume that
  - (1) all attributes names in all primary keys are unique, it will make the notation easier to understand and it really isn't a problem if the names aren't unique anyway, and
  - (2) each entity set participates only once in the relationship.
- Then the composition of the primary key for the relationship set depends on the set of attributes associated with the relationship set R in the following ways:

Relationship Sets (cont.)

- (a) If the relationship set *R* has no attributes associated with it, then the set of attributes: $K_1 \cup K_2 \cup \ldots \cup K_n$ describes an individual relationship in set *R*.
- (b) If the relationship set *R* has attributes $a_1, a_2, \ldots, a_m$ associated with it, then the set of attributes: $K_1 \cup K_2 \cup \ldots \cup K_n \cup \{ a_1, a_2, \ldots, a_m \}$ describes an individual relationship in set *R*.
- In **both** of these cases, the set of attributes: $K_1 \cup K_2 \cup \ldots \cup K_n$ forms a superkey for the relationship set.

Effect of Cardinality Constraints on Keys

- The structure of the primary key for the relationship set depends upon the mapping cardinality of the relationship set. Consider the following case:



- This E-R diagram represents a many to many cardinality for the relationship *deposits* with an attribute of *access date* associated with the relationship set with two entities *customer* and *account* participating in the relationship. The primary key of the relationship *deposits* will consist of the union of the primary keys of *customer* and *account*.

Effect of Cardinality Constraints on Keys

- To further clarify this situation consider for a moment the schemas of these two entity sets:

  Customer (customer-id, customer-name, address, city)

  Account (account-number, balance)

- A many-to-many relationship between these two sets means that it is possible for one customer to have several accounts and similarly for a given account to be held by several customers.

- To uniquely identify a relationship between two entities in *customers* and *accounts* will require the union of the primary keys in both entity sets.

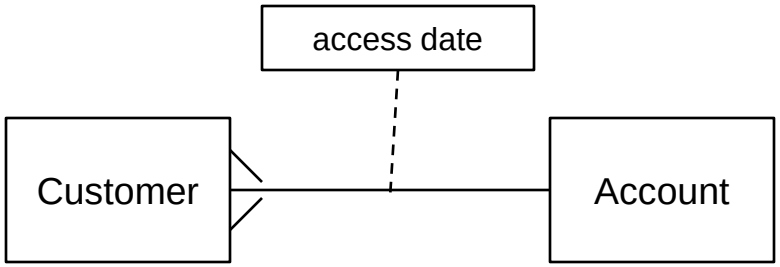
Effect of Cardinality Constraints on Keys (cont.)

- In order to "see" the last deposit made to specific account number requires that we specify by whom the deposit was made since several account holders may have made deposits to the same account.
- The schema for the *deposits* relationship is then:

Deposits (customer-id, account-number, access-date)

Effect of Cardinality Constraints on Keys (cont.)

- Now consider the case when a customer is only allowed to have one account. This means that the *deposits* relationship is many-to-one from *customer* to *account* as shown in the following diagram.



- In this case the primary key of the *deposits* relationship is simply the primary key of the *customer* entity set. To clarify this, again look at the schemas of the entity sets:

Customer (customer-id, customer-name, address, city)

Account (account-number, balance)

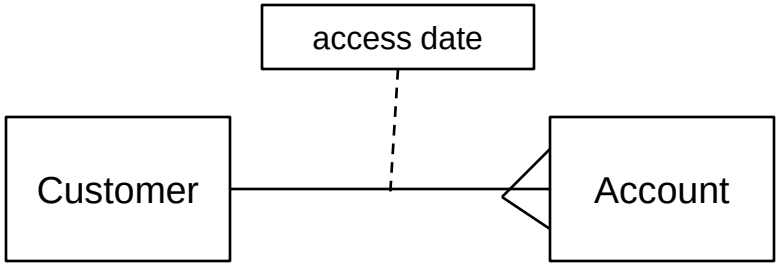
Effect of Cardinality Constraints on Keys (cont.)

- As a many-to-one relationship means that a given customer can have only a single account then the primary key of the *deposits* relationship is simply the primary key of the *customer* set since for a given customer they could only make a single most recent deposit since they only "own" one account, so specifying the account number is not necessary to identify a unique deposit by a given customer.
- The schema for the *deposits* relationship set is then:

  Deposits (customer-id, access-date)

Effect of Cardinality Constraints on Keys (cont.)

- Now consider the case when the *depositor* relationship is many-to-one from *account* to *customer*.



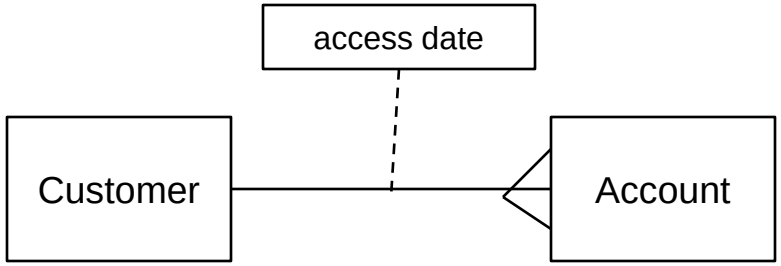
- A many-to-one relationship from account to customer means that each account is owned by at most one customer but each customer may have more than one account. In this situation the primary key of the *deposits* relationship is simply the primary key of the *account* entity set since there can be at most one most recent deposit to a given account because at most one customer could make the deposit. We do not need to uniquely identify which customer made the deposit in question because there could only be one.
- The schema for the *deposits* relationship is then:

  Deposits (<u>account-id</u>, access-date)

Placement of Relationship Attributes

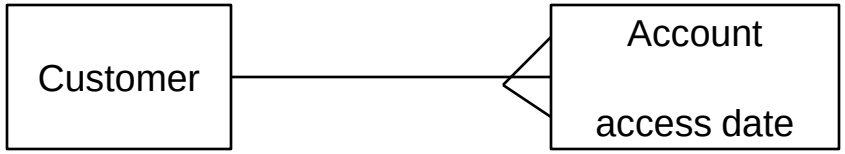
- Just as the cardinality of a relationship set affects the set of attributes which comprise the primary key of the relationship set, so too does it affect the placement of the attributes.

- The attributes of a one-to-one or one-to-many relationship set can be associated with one of the participating entity sets, rather than with the relationship set itself. For example consider the following case:



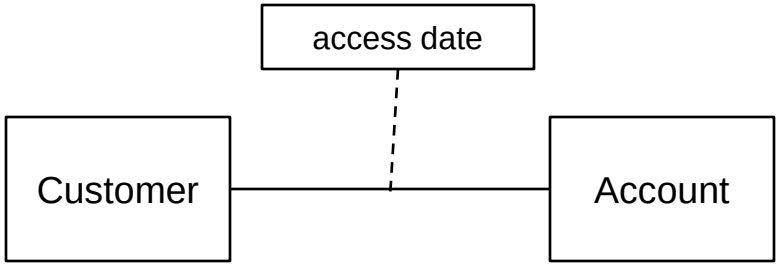
Placement of Relationship Attributes (cont.)

- The attribute *access-date* could be associated with the *account* set without loss of information. Since a given account can be owned by at most one customer it could have at most one access-date which could be stored in the *account*



Placement of Relationship Attributes (cont.)

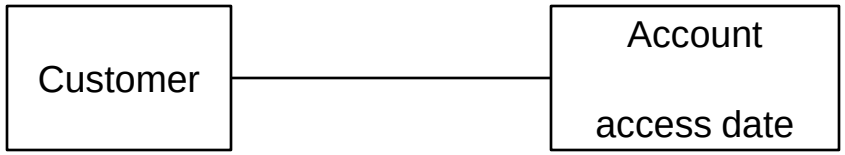
Now consider the following case:

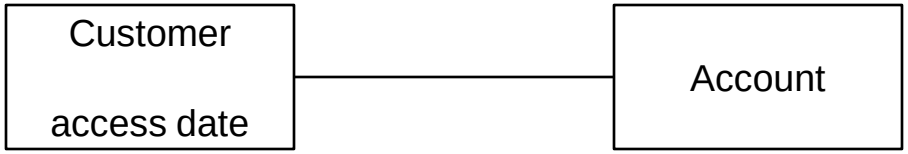


- The attribute *access-date* could be associated with either the *customer* set or the *account* set without loss of information. In this case a given account can be owned by at most one customer and a given customer can own at most one account. Therefore, if the *access-date* attribute is stored with the *customer* set then it must refer to the last access by this customer on the only account they can have. Similarly, if the *access-date* attribute is stored with the *account* set, then it must refer to the last access on this account by the only customer who owns this account.

Placement of Relationship Attributes (cont.)

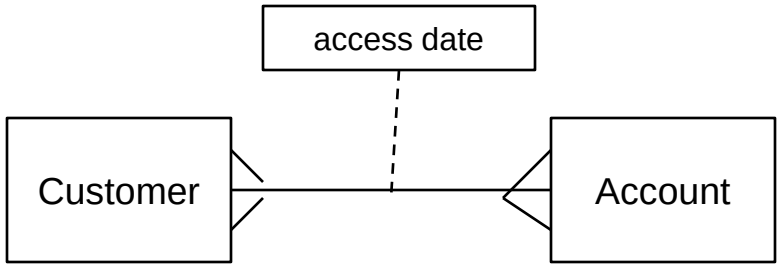
- Therefore, either diagram below would be a correct representation of this situation:





Placement of Relationship Attributes (cont.)

- When the relationship set has a cardinality constraint of many-to-many, the situation is much clearer. Consider the following situation:



- Since an account may be owned by several customers, we see that associating the *access-date* attribute with either entity set will not properly model this situation without the loss of information. If we need to model the date that a specific customer last accessed a specific account the *access-date* attribute must be an attributed of the *depositor* relationship set, rather than one of the participating entities. For example, if *access-date* were an attribute of *account* we could not determine which customer made the last access to the account. If *access-date* were an attribute of *customer* we could not determine which account the customer last accessed.

Further Design Issues

- The notions of an entity set and a relationship set are not precise.
- It is possible to define a set of entities and the relationships among them in a number of different ways. We'll look briefly at some of these different approaches to the modeling of the data.
- To some extent this is where the "art" of database design becomes tricky. Sometimes several different design scenarios may all look equally plausible and even after refinement may still be suitable, sometimes not. Only a careful design will eliminate some of the problems we've discussed earlier.

Entity Sets vs. Attributes

- Consider the entity set: *Employee(emp-name, telephone-number, age)*
- It could easily be argued that a telephone is an entity in its own right with attributes of say, *telephone-number*, *location*, *manufacturer*, *serial-num*, and so on. If we take this point of view, then:

1. The *Employee* entity set must be redefined as:

   Employee (emp-name, age)

2. Must create a new entity set:

   Telephone(telephone-number, location, manufacturer, serial-num,…)

3. A relationship set must be created to denote the association between employees and the telephones that they have.

   Emp-Phone(emp-name, telephone-number, age, location, manufacturer, serial-num)

Entity Sets vs. Attributes (cont.)

- Now we must consider what it the main difference between these two definitions of an employee?
- Treating the telephone as an attribute *telephone-number* implies that employees have precisely one telephone number each. (Note that this must be true or otherwise the telephone-number attribute would need to be a part of the key for an employee and it isn't here – not considering multiple-valued attributes).
- Treating a telephone as an entity permits employees to have several phones (including zero) associated with them. However, we could easily make the *telephone-number* attribute be a multi-valued one to allow multiple phones per employee. So clearly, this is **not** the main difference in the two representations.

Entity Sets vs. Attributes (cont.)

- The main difference then is that treating a telephone as an entity better models a situation where one might want to keep additional information about a telephone, as we have indicated with our example above.

- If we used the original approach and wished to make the telephone an attribute of an employee and we wished to maintain this additional information about their phone, then the *Employee* entity set would look like:

  Employee(emp-name, telephone-number, age, location, manufacturer,...)

- This is clearly not a good schema, for example, is the *age* attribute associated with the employee or the telephone? In this situation we are attempting to model two different entity sets inside a single entity set.

Entity Sets vs. Attributes (cont.)

- Conversely, it would not be appropriate to treat the attribute *emp-name* as an entity; it is difficult to argue that an employee name is an entity in its own right (in contrast to the telephone). Thus, it is entirely appropriate to have *emp-name* as an attribute of the *Employee* entity set.

- So, what constitutes and attribute and what constitutes an entity?

  – Unfortunately, there are no simple answers. The distinctions depend mainly upon the structure of the real-world scenario which is being modeled, and on the semantics associated with the attribute in question.

Entity Sets vs. Attributes (cont.)

- A common mistake is to use the primary key of an entity set as an attribute of another entity set, instead of using a relationship. For example, given our bank example again, it would not be appropriate to model *customer-id* as an attribute of *loan* even if each loan had only one customer associated to it. The relationship *borrower* is the correct way of representing the relationship between a loan and a customer, since it makes their connection explicit rather than implicit via an attribute.

Associative Entities (cont.)

- How do you know whether to convert a relationship into an associative entity type?
- There are four conditions that should exist:

1. All of the relationships for the participating entity types are "many" relationships.
2. The resulting associative entity type has independent meaning to end users, and preferably can be identified with a single-attribute identifier.
3. The associative entity has one or more attributes, in addition to the identifier.
4. The associative entity participates in one or more relationships independent of the entities related in the associated relationship.

Entity Sets vs. Relationship Sets

- It is not always clear whether an object is best expressed by an entity set or a relationship set.
- Consider the banking example. We have been modeling a loan as an entity. An alternative is to model a loan as a relationship between customers and say branches of the bank, with *loan-number* and *amount* as descriptive attributes. Each loan is then represented as a relationship between a customer and a branch.

Entity Sets vs. Relationship Sets (cont.)

- If every loan is owned by exactly one customer and is associated with exactly one branch, then it may be satisfactory to model the loan as a relationship.
- However, with this design we cannot represent in a convenient way the situation in which several customers jointly own a single loan.
  - To handle this type of situation, we would need to define a separate relationship for each holder of the joint loan.
  - Then we would replicate all of the values for the descriptive attributes *loan-number* and *amount* in each such relationship. Each such relationship must, of course, have the same value for the descriptive attributes.

Entity Sets vs. Relationship Sets (cont.)

- Two problems arise as a result of the replication:

  1. The data are stored in multiple locations (the very meaning of replication).

  2. Updates potentially leave the data in an inconsistent state, where the values in two different sets differ when they should be identical. We'll look at the complications that this replication causes as well as solution techniques (normalization theory) later in the course. Notice that the problem of replication is absent in our original version because *loan* is represented by an entity set in that case.

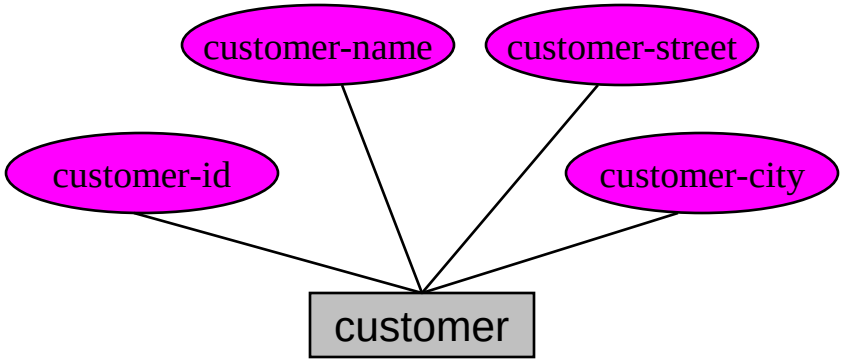
- One possible guideline in determining whether to use an entity set or a relationship set is to designate a relationship set to describe an action that occurs between entities. This approach can also be useful in deciding whether certain attributes may be more appropriately expressed as relationships.

The Unified Modeling Language (UML) (cont.)

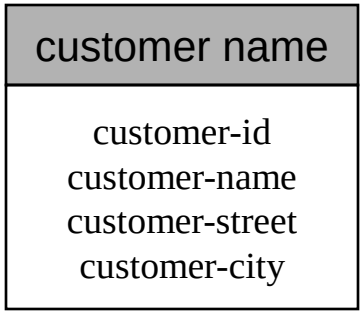
- Some of the parts of UML are:
  1. Class diagram. A class diagram is similar to an E-R diagram. We'll see the correspondence between them shortly.
  2. Use case diagram. Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money from a bank account or registering for a course).
  3. Activity diagram. Activity diagrams depict the flow of tasks between various components of the system.
  4. Implementation diagram. Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

Correspondence of Old-style ERDs & UML Class Diagrams

Entity sets and attributes



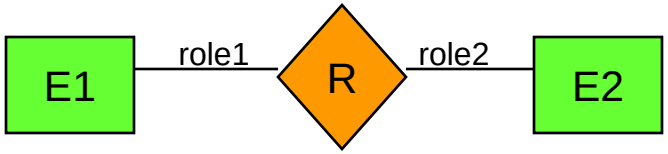
E-R Diagram

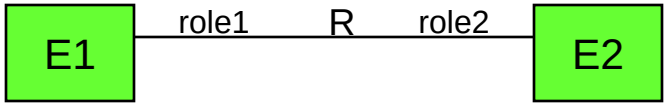


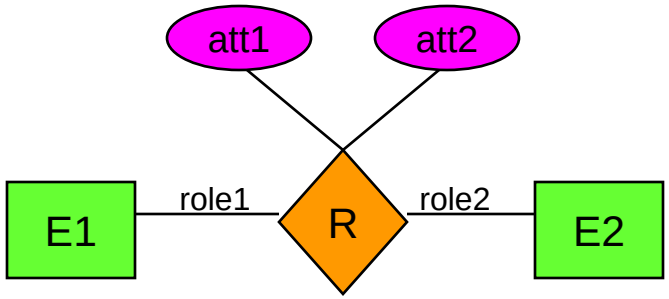
UML Class Diagram

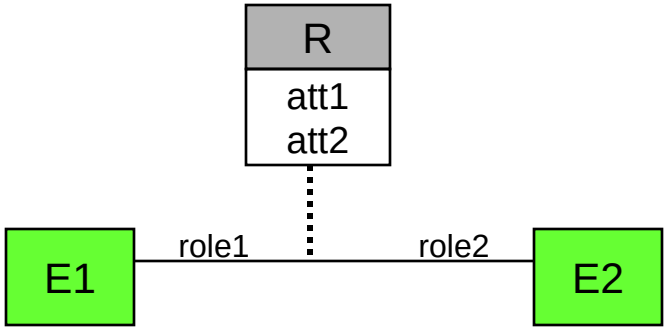
Correspondence of E-R & UML Class Diagrams (cont.)

Relationships







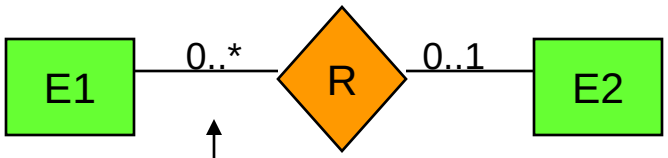


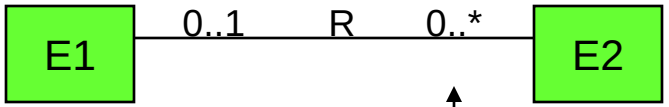
E-R Diagrams

UML Class Diagrams

Correspondence of E-R & UML Diagrams (cont.)

Cardinality Constraints





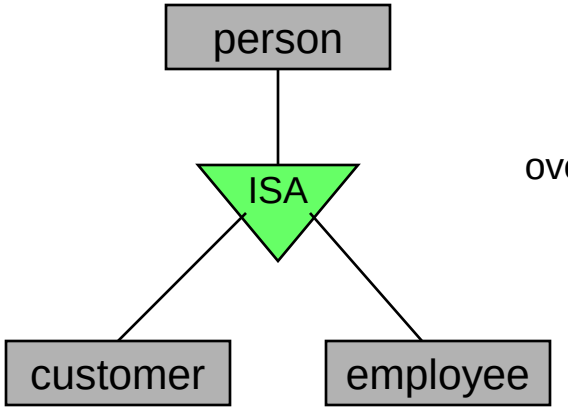
NOTE: Positioning of cardinality constraints is exactly opposite in the two models. In the UML model the constraint 0..1 on the left side means that an E2 entity can participate in at most 1 relationship, whereas each E1 entity can participate in many relationships; in other words, the relationship is many to one from E2 to E1

E-R Diagrams

UML Diagrams

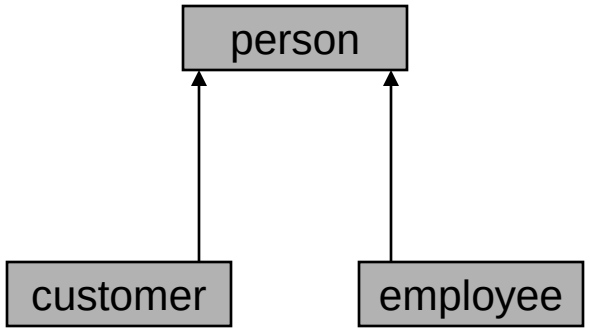
Correspondence of E-R & UML Class Diagrams (cont.)

Generalization & Specialization



E-R Diagrams

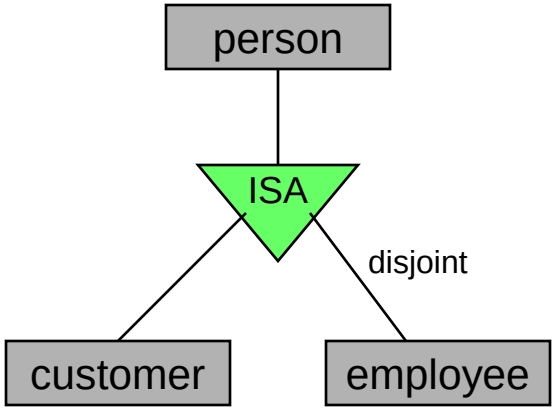
overlapping generalization



UML Class Diagrams

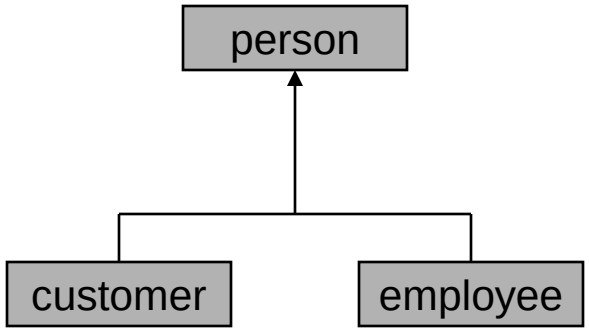
Correspondence of E-R & UML Class Diagrams (cont.)

Generalization & Specialization



E-R Diagrams

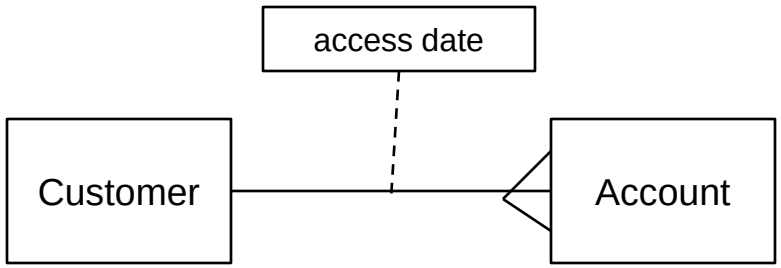
disjoint generalization



UML Class Diagrams

Referential Integrity Constraints

- Referential integrity constraints can be as simple as asserting that a given attribute have a non-null, single value. However, referential integrity constraints most commonly refer to the relationships among entity sets.
- Let's again consider our banking example and the one-to-many relationship between customer and account as shown below:



Referential Integrity Constraints (cont.)

- The one-to-many relationship depositor simply says that no account can be deposited into by more than one customer (and also that a customer can deposit into many different accounts).
- More importantly, it does **not** say that an account must be deposited into by a customer, nor does it say that a customer must make a deposit into an account. Further, it does not say that if an account is deposited into by a customer that the customer be present in the database!
- A referential integrity constraint requires that each entity "referenced" by the relationship must exist in the database.
- There are several methods which can be used to enforce referential integrity constraints:

Referential Integrity Constraints (cont.)

1. Deletion of a referenced entity is not allowed. In other words, if Kristi makes a deposit into account number 456, then subsequently we cannot delete either the information concerning either Kristi or account 456.

2. If a referenced entity is deleted, then all entries that reference the deleted entity also be deleted. In other words, if we delete the information on Kristi, then we must delete all account information for accounts that she (alone) has deposited into. Notice in the specific example we are considering, that the relationship is M:1 which means that if Kristi has deposited into an account, she will be the only customer to do so. This will not be the case for a M:M relationship however.

- Referential integrity constraints can be modeled in ERDs although the notation varies widely from tool to tool. We'll hold off on this until we see SQL later on.