# 2. Conceptual Modeling with Entity-Relationship Diagrams

# 1. Introduction

- Database design: A description of the data environment that is to be captured in a computer database.
- The design is an abstraction of the "real world", driven by the anticipated applications.
- Data model: A formalism for specifying database designs.
- Every database management system is based on a particular data model:
  - ✓ Hierarchical, network, relational, object-oriented
- Analogy with programming:
  - ✓ Database design ~ Programming
  - ✓ Data model ~ Programming language
  - ✓ E-R methodology ~ Flowcharting

# Introduction (cont.)

- Hence, the Entity-Relationship approach is a design methodology, not a formal data model.
  - ✓ In the same way that flow-charts are not executable programs.
- E-R diagram: A high-level graphical view of the enterprise information.
- E-R diagram can later be converted to a database schema in an actual data model.
  - ✓ In our case, the relational data model.
  - ✓ This schema is then specified in SQL.
  - ✓ Given this SQL specification, the DBMS will create a new database.
- The E-R approach has never been standardized.
  - ✓ Different notations have been proposed.
  - ✓ We adopt the notation in [KBL].

# 2. Entities and Entity Types

- Essentially, we view all information as
  - ✓ Entities, and
  - ✓ Relationships among the entities.
- Entity: An object that is distinguishable from other objects.
  - ✓ Similar to objects in object-oriented programming, except that we are not concerned with methods.
- Examples:
  - ✓ The student John Doe
  - ✓ The student Betty Brown
  - ✓ The course CS 450
  - ✓ The course CS 650
- Entities may be concrete objects (e.g., a particular person), or abstract objects (e.g., a particular course).

# Entities and Entity Types (cont.)

- Attribute: A property of an entity.
- Examples:
  - ✓ The Name or Major of a Student entity
  - ✓ The Title or Credits of a Course entity
- Each entity is described by a set of attributes.
- Entity Type: A set of entities.
  - ✓ The set must be homogeneous: Every entity in the set is described by the same attributes.
  - ✓ Hence, the attributes may be associated with the entity type.
- Examples:
  - ✓ The entity type Student contains the entities 'John Doe' and 'Betty Brown' — both are described by Name and Major.
  - ✓ The entity type Course contains the entities 'CS 450' and 'CS 650' — both are described by Title and Credits.

# Entities and Entity Types (cont.)

- Domain: The domain of an attribute is the set from which its values are drawn.
- Examples:
  - ✓ The domain of Age is the range 0–120.
  - ✓ The domain of Title is {'Database Systems', 'Java Programming', … }.
- Attributes may be either
  - ✓ Single-valued
    - Example: The attribute Date_of_Birth of the entity type Person.
  - ✓ Set-valued
    - Example: The attribute Hobbies of the entity type Person.

# Entities and Entity Types (cont.)

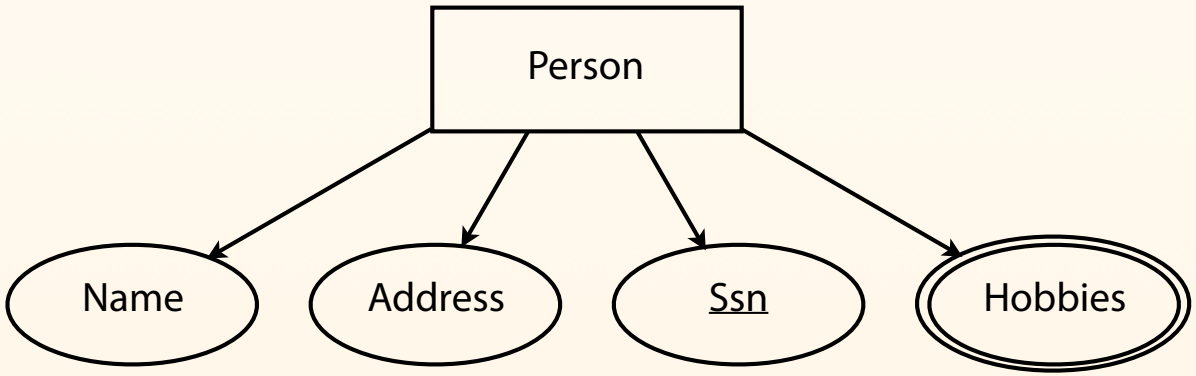
- Key: A subset *A* of the attributes of entity type *S* is a key, if
  1. Uniqueness: No two entities in *S* have the same value for all the attributes in *A*.
  2. Minimality: No proper subset of *A* has the uniqueness property.
- Examples:
  - ✓ The attribute Ssn is key of entity type Person.
  - ✓ The attributes (Address, Last_Name, First_Name) are key of entity type Person.
  - ✓ It is possible to have different keys for the same entity type!
- Keys are constraints: When the designer specifies the key attributes of an entity type, he/she states that two entities of that type may not have the same values in these attributes.
  - ✓ Consequently, specifying a key value selects at most one entity.
  - ✓ During the life time of the database, the database system would monitor all updates to prevent violation of key constraints.

# Entities and Entity Types (cont.)

- Earlier, we defined an entity as an object distinguishable from other objects.
- In other words, it should have a distinct description.
- Consequently, no two entities in an entity type should have identical values for all the attributes of the type.
- Therefore, each entity is guaranteed to have a key: the complete set of attributes.
  - ✓ Note: We differ here from [KBL].
- Schema of an entity type (what need to be specified when defining a new entity type):
  1. The name of the entity type
  2. The attributes of the entity type (with their associated domains, and an indication of whether each attribute is single-valued or set-valued)
  3. The key constraints

# Entities and Entity Types (cont.)

- E-R diagram:
  - ✓ Entity type: Rectangle
  - ✓ Attribute: Oval connected to the rectangle
    - Double oval for set-valued attributes.
    - Key attributes are underlined.
    - Notation allows specifying only one key (the primary key).

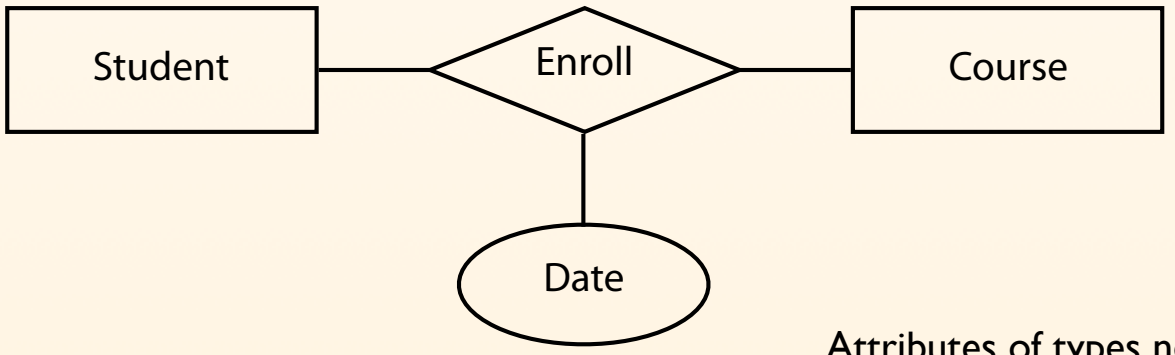


# 3. Relationships and Relationship Types

- Relationship: An association among two or more entities.
- Example: The entity 'John Doe' (of entity type Student) and the entity 'CS 450' (of entity type Course) are related, with the meaning that the student is enrolled in the course.
- Relationship Type: A homogeneous set of relationships:
  - ✓ All the relationships in the set associate entities of the same types and with have same meaning.
- Example: The relationship type Enroll is the set of relationships in which an entity in Student and an entity in Course are related, with the meaning that the student is enrolled in the course.
  - ✓ 'John Doe' is enrolled in 'CS 450'
  - ✓ 'John Doe' is enrolled in 'Math 501'
  - ✓ 'Betty Brown' is enrolled in 'CS 650'
  - ✓ 'Betty Brown' is enrolled in 'CS 450'

# Relationships, Relationship Types (cont.)

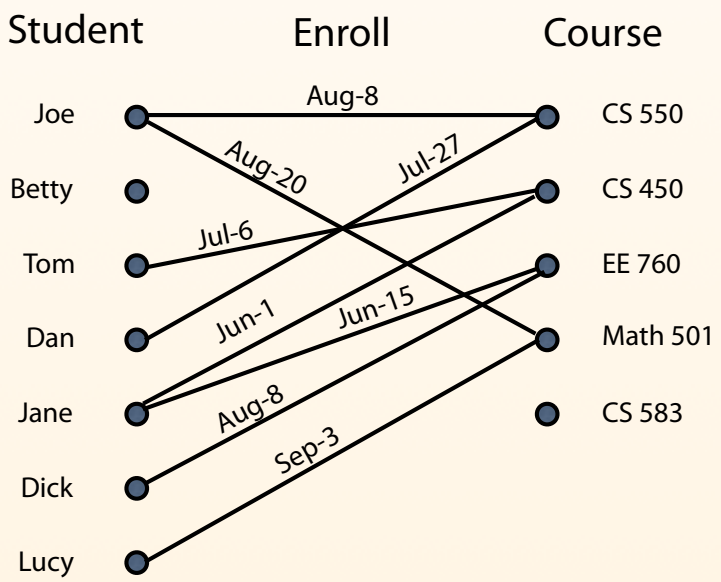
- Attribute: Like entities, relationships can have attributes.
- Since all the relationships of a relationship type are described by the same attributes, the attributes may be associated with the relationship type.
- Example: The relationship type Enroll between the entity types Student and Course may have the attribute Date (denoting the date the student enrolled in the course).
- E-R diagram:
  - ✓ Relationship type: Diamond, connected to the participating entity types




Attributes of types not shown

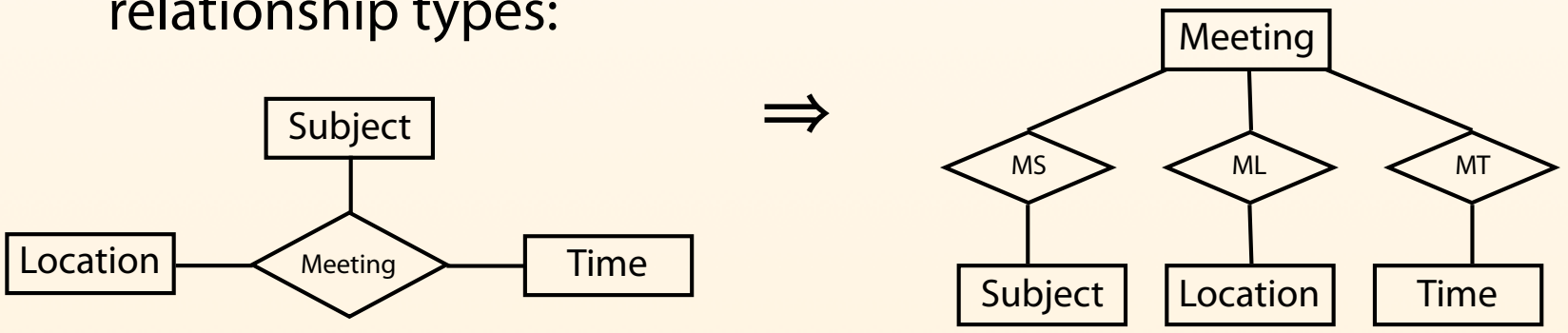
# Relationships, Relationship Types (cont.)

- Example: We draw individual entities and relationships to help understand the concepts.
  - ✓ This is not part of database design (or the E-R notation).
  - ✓ A database design does not indicate individuals, only types.
  - ✓ 7 students, 5 courses and 8 enrollments.
  - ✓ Each enrollment is associated with a date.
  - ✓ Note that Betty is not enrolled in any courses, and no students are enrolled in CS 583.



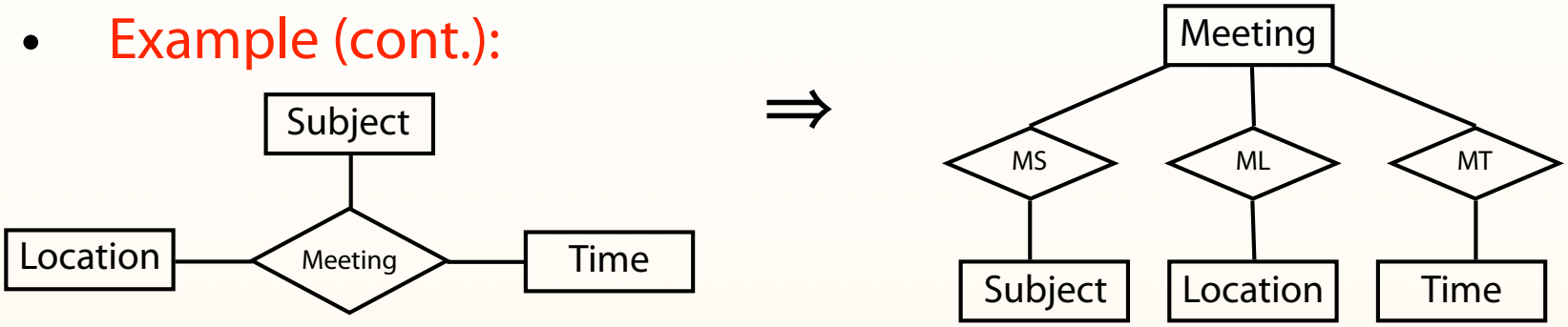
# Relationships, Relationship Types (cont.)

- The number of participating entity types in a relationship type is its degree.
- A relationship type of degree 2 is called binary.
- Example of a non-binary relationship:
  - ✓ The relationship type Meeting associates the entity types Subject, Location and Time.
- Non-binary relationship types may be restated as several binary relationship types.
- Example: The ternary relationship type Meeting can be represented with a fourth entity type Meeting and three relationship types:



# Relationships, Relationship Types (cont.)

- Example (cont.):




✓ In the first option, assume the entity types have these entities:
Subject = $\{s_1, s_2, s_3, s_4\}$
Location = $\{l_1, l_2, l_3, l_4, l_5\}$
Time = $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$

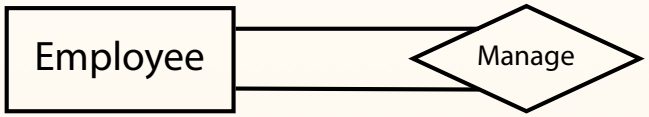
✓ In this option, Meeting relationships correspond to triplets.
For example, assume 3 meetings:
Meeting = $\{<s_1, l_3, t_2>, <s_1, l_3, t_4>, <s_2, l_2, t_4>\}$

✓ In the second option, we define a new entity type with three entities:
Meeting = $\{m_1, m_2, m_3\}$

✓ And relationships that are pairs:
MS = $\{<m_1, s_1>, <m_2, s_1>, <m_3, s_2>\}$
ML = $\{<m_1, l_3>, <m_2, l_3>, <m_2, l_2>\}$
MT = $\{<m_1, t_2>, <m_2, t_4>, <m_3, t_4>\}$

# Relationships, Relationship Types (cont.)

- A relationship type may relate an entity type with itself (a "recursive" relationship type).
- Example: The relationship type Manage between Employee and Employee.




- Compare Manage with Enroll:

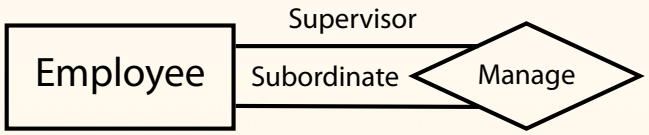

✓ There is no ambiguity as to the role of Student or Course in the relationship type Enroll.

✓ There is ambiguity in the role of Employee in the relationship type Manage:

- Who is the supervisor and who is the subordinate?

# Relationships, Relationship Types (cont.)

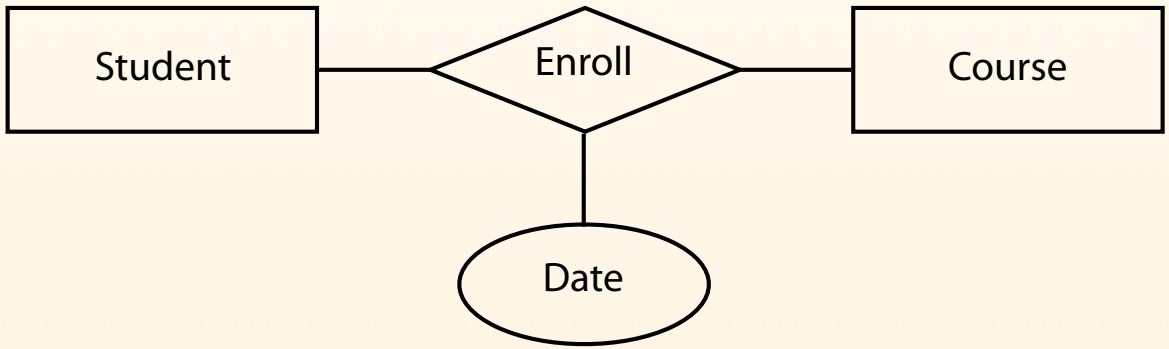
- Role: For each entity type participating in a relationship type we define a role — its function in the relationship.
- Example: The relationship type Manage has two roles: Supervisor and Subordinate.
- E-R diagram: Roles are indicated on the links between the entity types and the relationship type.




- When the relationship types associate distinct entity types (as in Enroll), there is no need to name the roles explicitly.
  - ✓ The roles could default to the names of the corresponding entity types: Enroll_Student and Enroll_Course.
  - ✓ Indeed, the concept is useful only when a relationship type involves the same entity type more than once.

# Relationships, Relationship Types (cont.)

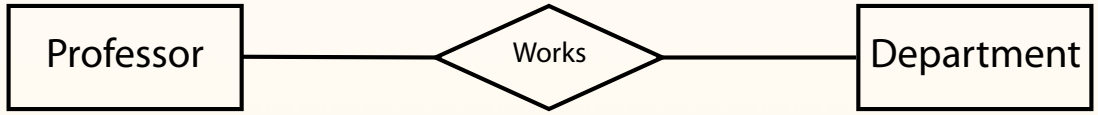
- Keys: Like entity types, relationship types need keys as well.
- The purpose is to uniquely identify the relationships in a relationship type.
- Example 1: Assuming that each student can enroll in a particular course just once, relationships in the relationship type Enroll can be identified by a combination of its two roles (the participating entity types): Student and Course.
  - ✓ Specifying a Student entity and a Course entity identifies at most one Enroll relationship.



# Relationships, Relationship Types (cont.)

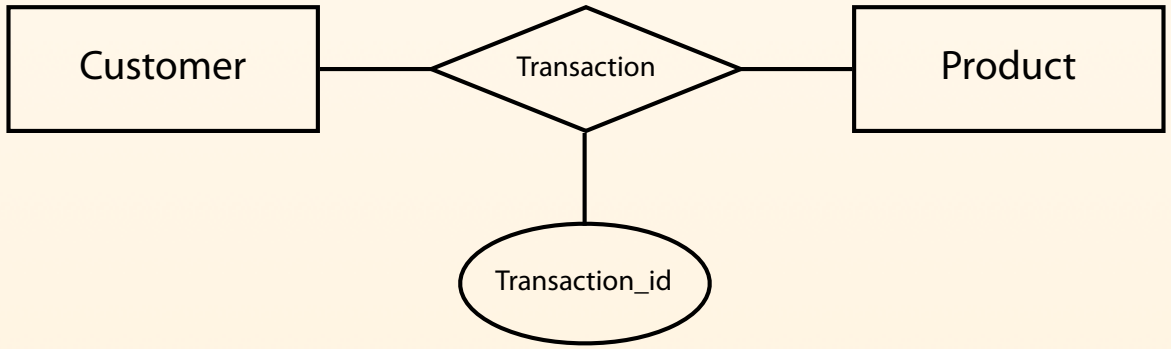
- Example 2: Assuming that each professor works in just one department, relationships in the relationship type Works may be identified by a single role: Professor.

✓ Specifying a professor identifies at most one Works relationship.




- Example 3: Consider a relationship type Transaction between the entity types Customer and Product, and assume that each transaction is assigned a unique id.

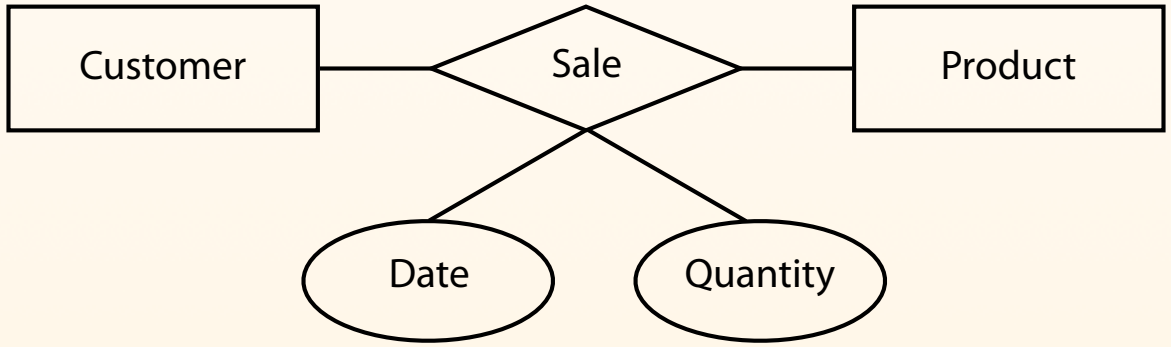
✓ Specifying a value of Transaction_id identifies at most one Transaction relationship.



# Relationships, Relationship Types (cont.)

- Example 4: Consider a relationship type Sale between the entity types Customer and Product, and assume that a customer could buy a product multiple times, but only once in a given date.

  ✓ The roles Customer and Product are insufficient to identify a Sale relationship, but a combination of the roles Customer and Product and the attribute Date identifies at most one Sale relationship.



# Relationships, Relationship Types (cont.)

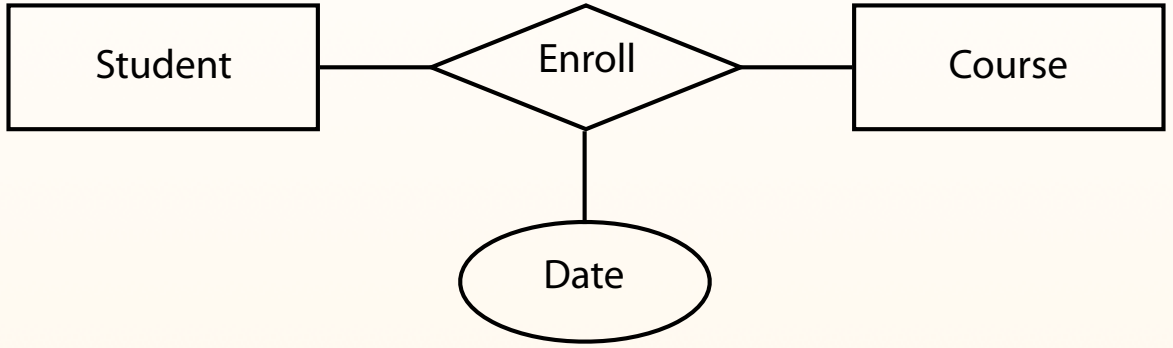
- In these examples relationship types were identified by the relationship roles only, by the relationship attributes only, and by a combination of roles and attributes.
- Key: A set of roles *R* and a set of attributes *A* (both of a relationship type *T*) is a key, if
  1. Uniqueness: No two relationships in *T* have the same value for each of the roles in *R* and the attributes in *A*.
  2. Minimality: No proper subset of $R \cup A$ has the uniqueness property.

  ✓ This definition uses roles rather than entity-types, because an entity-type could participate more than once in a relationship-type, but possibly only one of the roles will be included in the key.

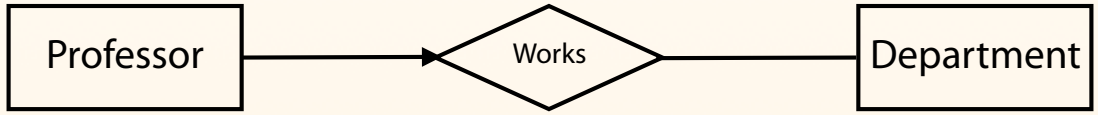
# Relationships, Relationship Types (cont.)

- E-R diagram: Keys are indicated in four different ways.

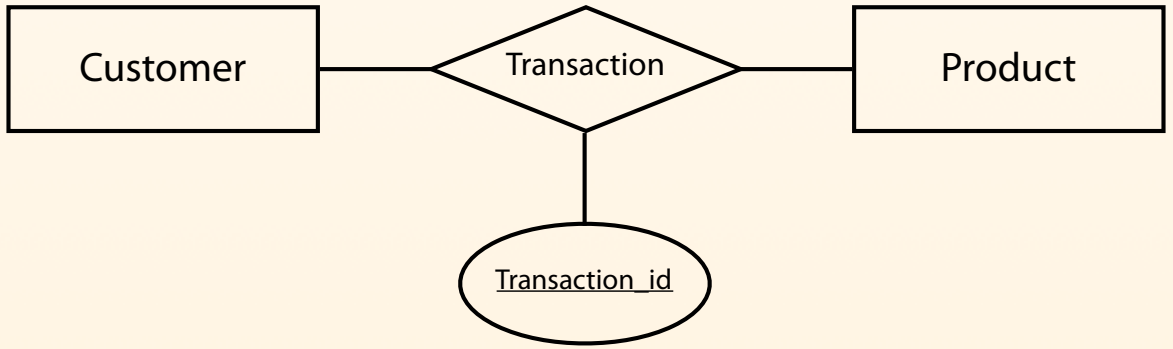
1. A key which is a combination of all the roles requires no notation:




2. A single-role key is indicated with an arrow:

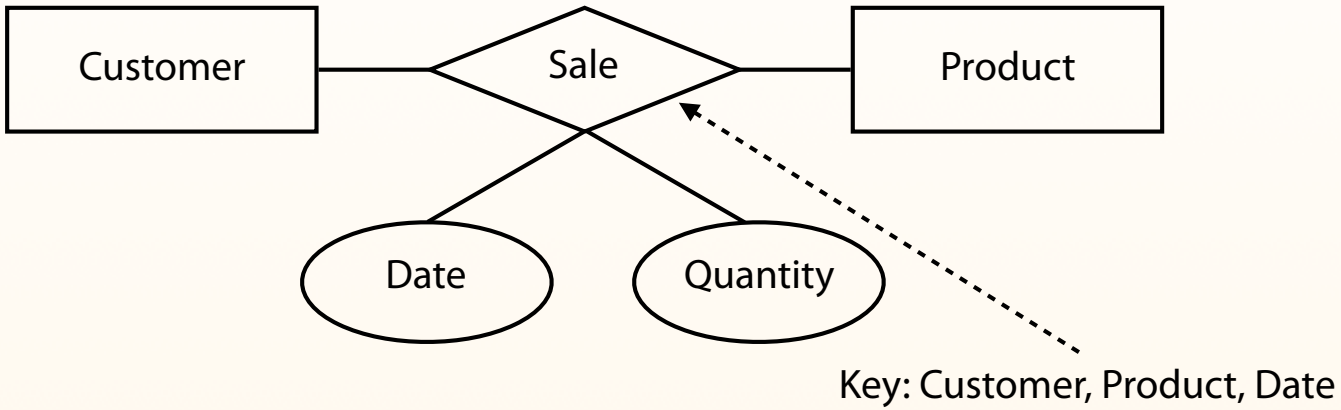



3. A key of attributes only is indicated by underlining these attributes:

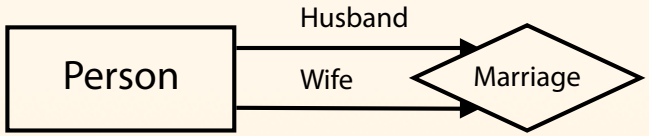


# Relationships, Relationship Types (cont.)

4. In all other cases, the key is written explicitly:



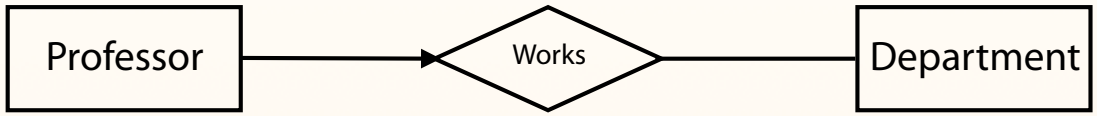

- Note that the arrows in the following diagram mean that each of the roles Husband and Wife is a key to Marriage, not their combination!




- Altogether: The indication of relationship keys in E-R diagrams is unattractive.

# Relationships, Relationship Types (cont.)

- Example: Recall that this diagram incorporates a single-role constraint that a Professor entity is associated with at most one Works relationship:

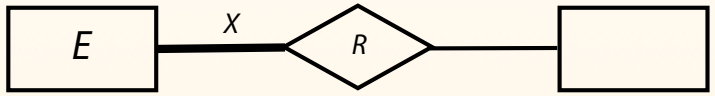



That is, a professor works in at most one department.
However, it allows situations in which a professor is not in any department.
That is, it does not reflect the requirement that each professor works in at least one department.

- Such requirements are expressed with participation constraints.

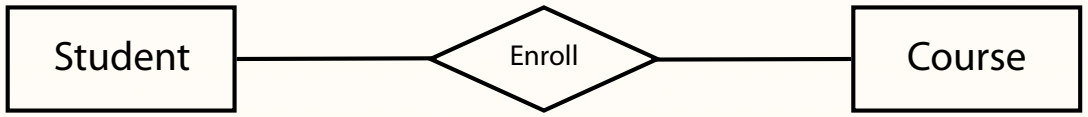
# Relationships, Relationship Types (cont.)

- Participation constraint: A participation constraint of entity type *E* in relationship type *R* in role *X* states that for every entity *e* in *E*, there is a relationship *r* in *R*, such that *e* participates in *r* in role *X*.
- E-R Diagram: A participation constraint is indicated by a thick connecting line for the appropriate role.



# Relationships, Relationship Types (cont.)

- Example: Consider again



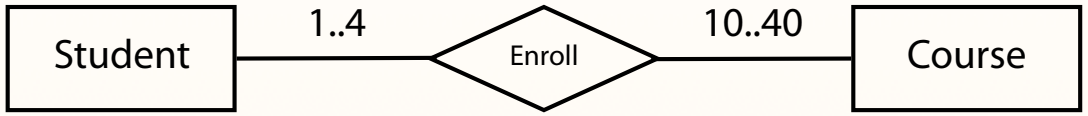

  - ✓ Assume that university rules dictate that each student must enroll in at least 1 course but in no more than 4 courses.
  - ✓ This is a new type of constraint that should be included in the design.

- Cardinality constraint: A constraint placed on a role of an entity type in a relationship type, to restrict the number of relationships in which a single entity may participate.
  - ✓ The constraint is of the form *min* .. *max,* where $0 \leq min \leq max \leq \infty$.

- E-R Diagram: The constraint is written next to the link of the particular role.

# Relationships, Relationship Types (cont.)

- Example: Two independent cardinality constraints:




✓ Each student must enroll in at least one course, but in no more than 4.

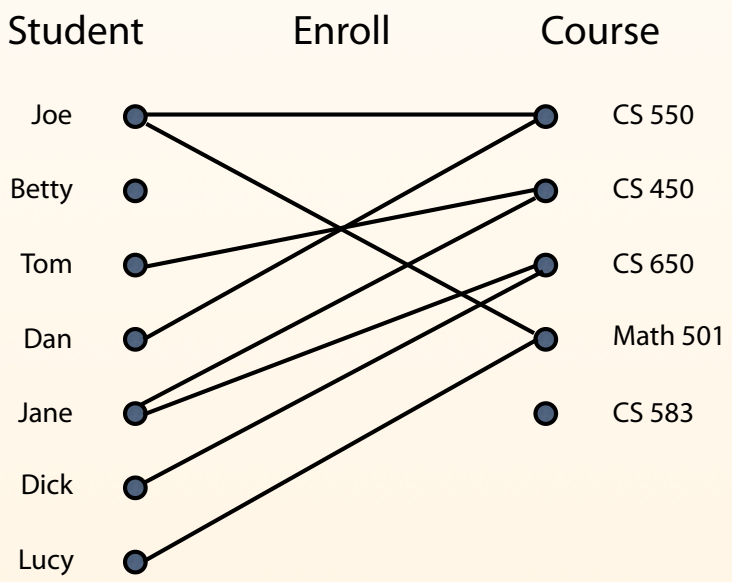
✓ Each course must enroll at least 10 students, but no more than 40.

- Special situations:

1. *N*..*N* — The minimum and maximum are identical (often abbreviated *N*).
2. 0..*N* — An entity does not have to participate in the relationship type.
3. *N*..* — There is no upper limit on the number of relationship instances in which an entity participates.
4. 0..* — There are no restrictions on the number of relationship instances in which an entity participates (cardinality is unconstrained).

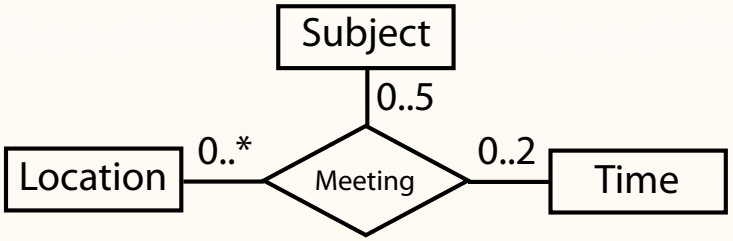
# Relationships, Relationship Types (cont.)

- The cardinality constraint on Student controls the number of links connecting each student.
- The cardinality constraint on Course controls the number of links connecting each course.



# Relationships, Relationship Types (cont.)

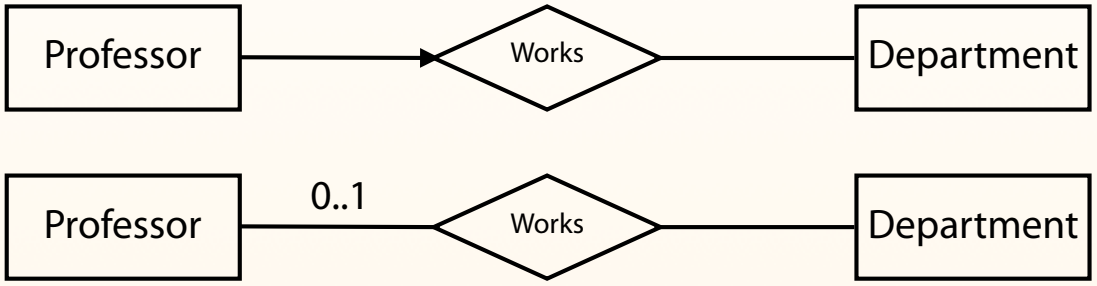
- Example: Consider the ternary relationship type Meeting, associating the entity types Subject, Location and Time:



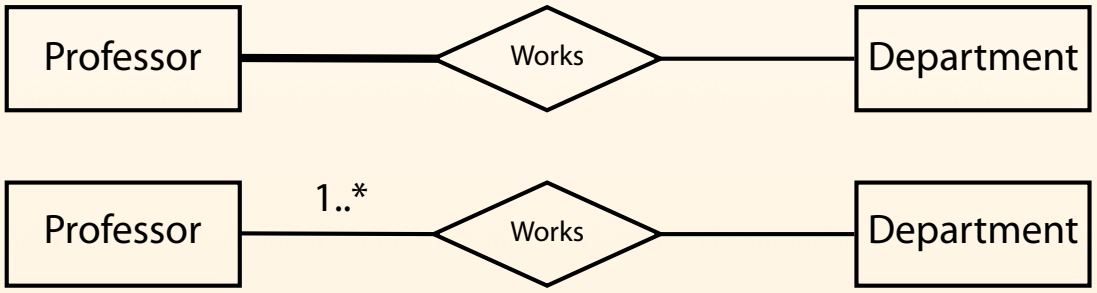

- ✓ Each relationship instance of Meeting can be viewed as a triplet of values <subject, location, time>.
- ✓ The cardinality constraint 0..5 means that each subject can appear in at most 5 triplets (no subject shall be discussed in more than 5 meetings).
- ✓ The cardinality constraint 0..2 means that each time can appear in at most 2 triplets (at most 2 meetings can be scheduled at the same time).
- ✓ The cardinality constraint 0..* means that each location can appear in any number of triplets (there are no restrictions on the use of locations)
- ✓ Note that this diagram does not prevent situations in which more than one meeting is scheduled in the same location and at the same time!

# Relationships, Relationship Types (cont.)

- Cardinality constraints can express single-role keys (indicated with arrows).
  - ✓ A single-role key constraint is similar to a 0..1 cardinality constraint:



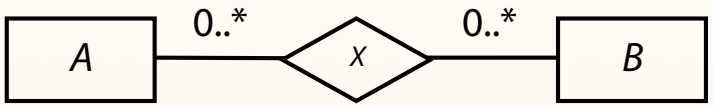

- Cardinality constraints can express participation constraints (indicated with thick lines).
  - ✓ A participation constraint is similar to a 1:* cardinality constraint:



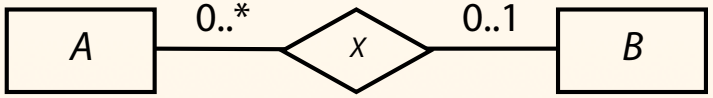
# Relationships, Relationship Types (cont.)

- Other terminology: E-R descriptions often refer to three types of cardinality constraints:

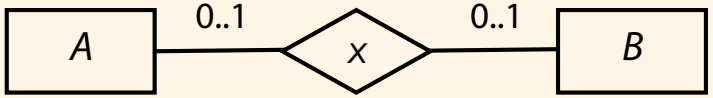
1. Many-to-many: Each entity of *A* is associated with any number of *B* entities, and each entity of *B* is associated with any number of *A* entities
   - Example: The relationship type Enroll between Student and Course entity types.
   - In our notation:




2. One-to-many: Each entity of *A* is associated with any number of *B* entities, but each entity of *B* is associated with at most one *A* entity.
   - Example: The relationship type Works between Department and Professor entity types.
   - In our notation:

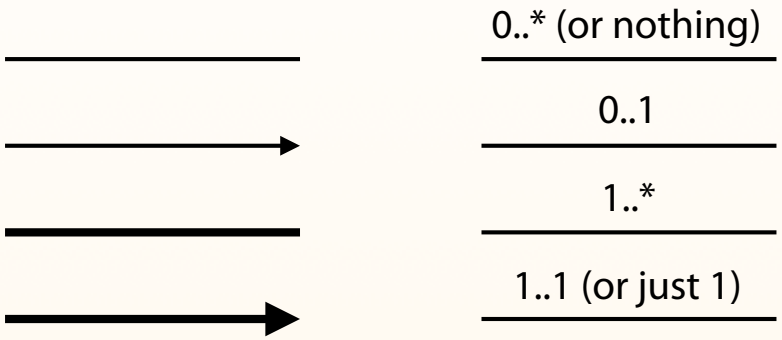



3. One-to-one: Each entity of *A* is associated with at most one *B* entity, and each entity of *B* is associated with at most one *A* entity.
   - Example: The relationship type Marriage between Person (in role Husband) and Person (in role Wife).
   - In our notation:

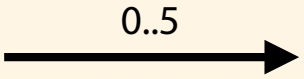


# Relationships, Relationship Types (cont.)

- Summary of notation alternatives:



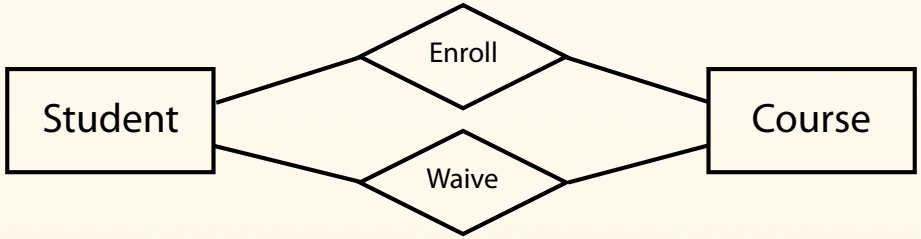

- Cardinality constraints can express single-role keys and participation constraints, as well as many-to-many, one-to-many and one-to-one relationships.
- However, cardinality constraints can also express additional constraints (where the minimum is other than 0 or 1 and the maximum is other than 1 or *).
- When mixing the notations, care must be taken to avoid contradictions, such as:



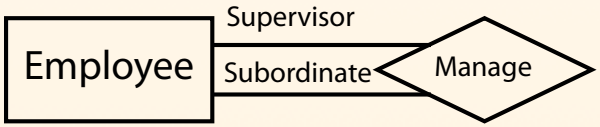
# Relationships, Relationship Types (cont.)

- Design possibilities:
  - ✓ One entity type in multiple relationship types:


  - ✓ Multiple relationship types between the same entity types:




  - ✓ A relationship type between the same entity types:



# Relationships, Relationship Types (cont.)

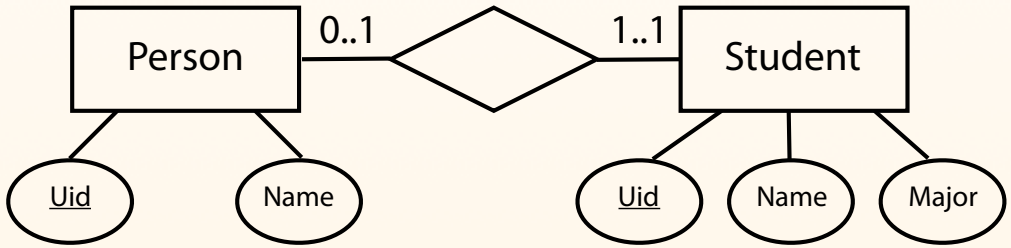
- Schema of a relationship type (what need to be specified when defining a new relationship type):
  1. The name of the relationship type
  2. The attributes of the relationship type (with their associated domains, and an indication whether attributes are single-valued or set-valued)
  3. The roles of the relationship type
  4. The key and cardinality constraints

# 4. Advanced Features

- Of the many features that have been incorporated at one point or another into the E-R approach, we discuss two:
  1. Entity type hierarchies
  2. Weak entity types

# Entity Type Hierarchies

- Example: Consider the entity types Person and Student.
  - ✓ Each entity of type Student corresponds to a unique entity of type Person.
  - ✓ Both Person and Student have attributes Uid and Name. But, in addition, Student has a Major.
- We could model the relationship with




- However, this solution does not enforce the constraint that the person corresponding to a student should have the same Uid and Name.
- We recognize that this is a different type of relationship, and introduce the concept of subtype.

# Entity Type Hierarchies (cont.)

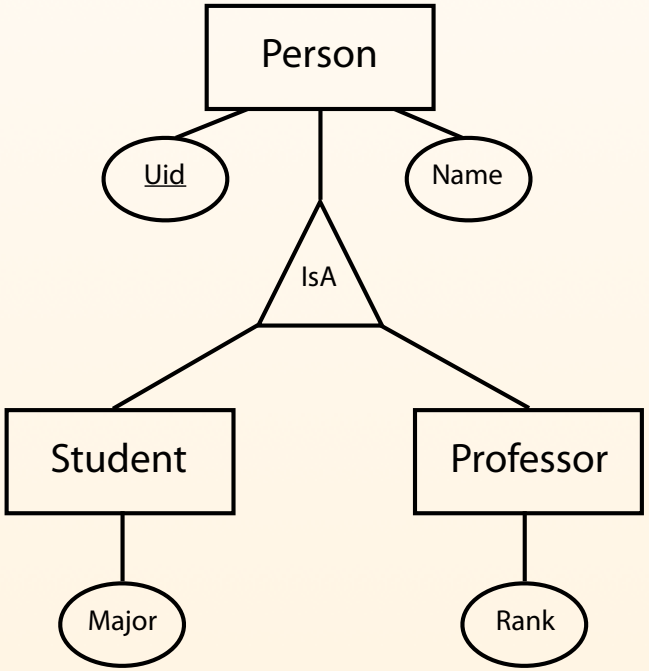
- Two entity types may be related in a special relationship type denoting that one is a subtype of the other.
  - ✓ Example: The Student entity type is a subtype of the Person entity type.
- There are three unique features of this relationship type:
  1. Every entity of the subtype is also an entity of the supertype. The same entity is an element of both entity types.
     - Example: Every student is also a person.
  2. Every attribute of the supertype is also an attribute of the subtype.
     - Example: If Age is an attribute of Person, then it is also an attribute of Student.
  3. Every entity of the supertype has the same value for each of the attributes that it has as an element of the subtype.
     - Example: The age of 'John Doe' the Person is the same as the age of 'John Doe' the Student.
- Consequence: A key of the supertype is also a key of the subtype.
  - Example: If Ssn is a key of Person then it is also a key of Student.

# Entity Type Hierarchies (cont.)

- The subtype relationship is called IsA.
  - ✓ Each student is a person.
- This relationship is also referred to as specialization-generalization:
  - ✓ The Student entity type is a specialization of the Person entity type.
  - ✓ The Person entity type is a generalization of the Student Entity type.
- The subtype is said to inherit the attributes of the supertype.
  - ✓ The subtype will usually have additional attributes (but not necessarily).

# Entity Type Hierarchies (cont.)

- E-R diagram:
  - ✓ IsA relationship type: Triangle, with the "super" entity type connected to a point, and the "sub" entity types connected to the opposite side.
  - ✓ A single triangle represents multiple IsA relationships: Student IsA Person and Professor IsA Person.
  - ✓ The inherited attributes are suppressed (although they are valid).

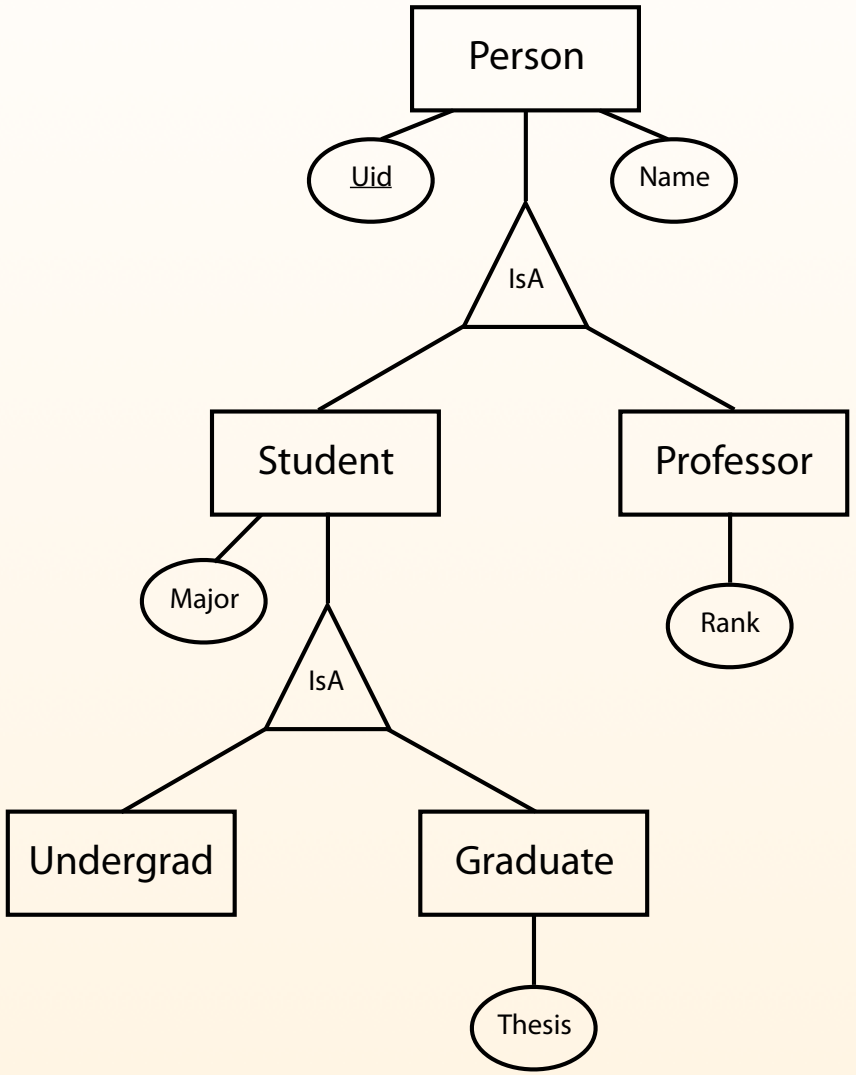


# Entity Type Hierarchies (cont.)

- A supertype may, in turn, become a subtype in another IsA relationship, thus creating a classification hierarchy.
- Transitivity: *A* IsA *B*, *B* IsA *C* ⇒ *A* IsA *C*
- Example: If Graduate_Student IsA Student, and Student IsA Person, then also Graduate_Student IsA Person.
- IsA relationships derived through transitivity are suppressed in the diagram.

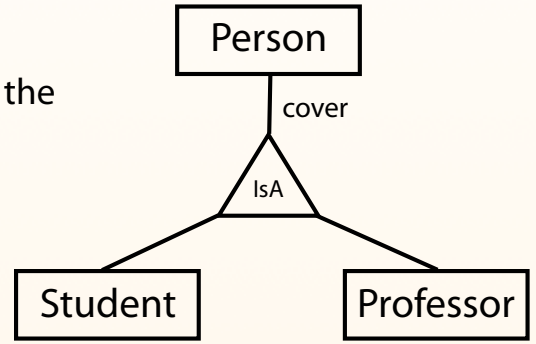
# Entity Type Hierarchies (cont.)

- Example:

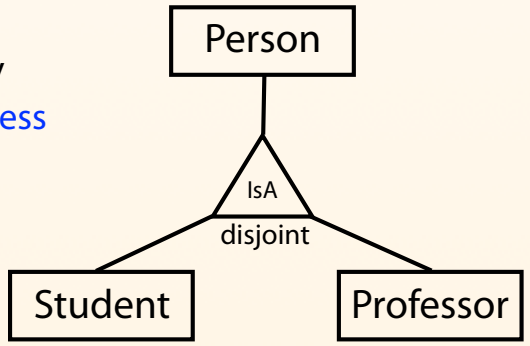


# Entity Type Hierarchies (cont.)

- IsA relationships are subject to two possible constraints.
  - ✓ Covering constraint: Is the union of the entities in the subtypes equal to the entities in the supertype, or does the supertype contain additional entities?
    - Example: If there are no other entities in Person except the entities in Student and Professor, then the covering constraint is satisfied.
    - Notation: Write "cover" above the triangle.

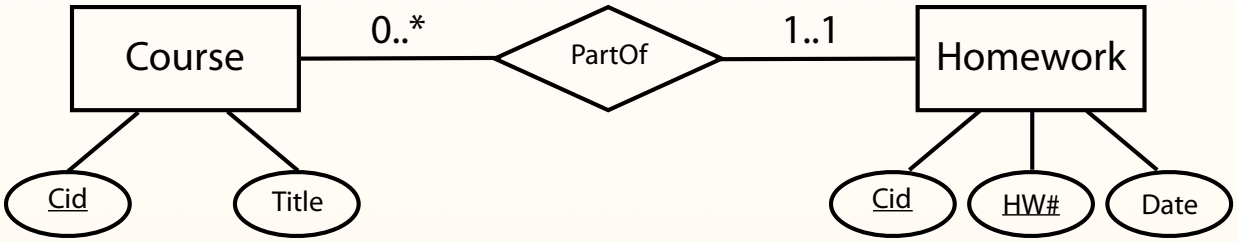



  - ✓ Disjointness constraint: Are the set of entities in the subtypes disjoint, or do they allow overlap?
    - Example: If an entity in Person cannot be both an entity in Student and an entity in Professor, then the disjointness constraint is satisfied.
    - Notation: Write "disjoint" below the triangle.



# Weak Entity Types

- Example: Consider the entity types Course and Homework:



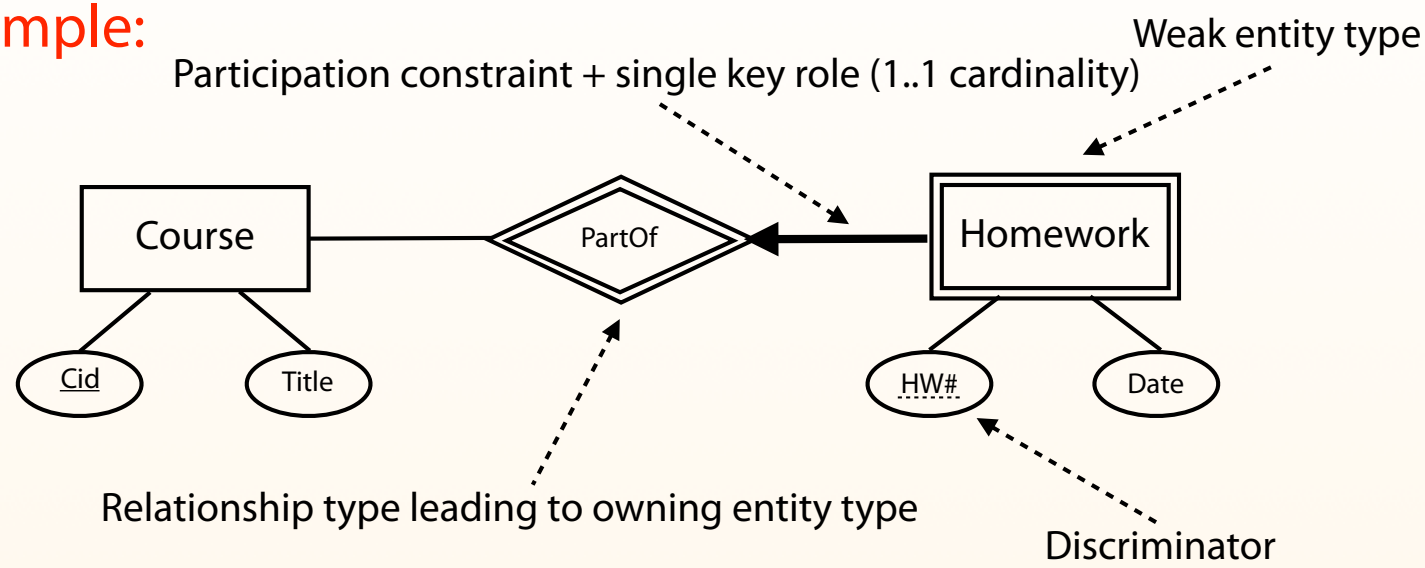

- ✓ Each Homework entity is associated with precisely one Course entity; i.e., each Course entity "exclusively owns" its Homework entities (one-to-many relationship). This is denoted by the cardinality constraint 1..1.
- ✓ A disadvantage of this design is that the Cid of each homework entity is listed twice: Once as an attribute of Homework, and again as an attribute of the related entity type Course.
- ✓ Such redundancy might lead to inconsistency:
  - A homework could end up having different Cid.
- ✓ A solution is to remove the attribute Cid from Homework.
  - But then, Homework will have no key!
  - There could be several Homework entities with same HW# and Date.
- ✓ Yet, a combination of HW# and the Cid of the associated course uniquely identifies each homework.

# Weak Entity Types (cont.)

- Such relationships are modeled with weak entity types: Entity types that are part-of other entity types.
- Requirements for weak entity type *W*:
  - ✓ *W* is related to an entity type *E* with a relationship type that has a cardinality constraint 1..1 (i.e., each *W* entity is owned exclusively by an *E* entity).
  - ✓ *W* includes attribute(s) that discriminate(s) among the *W* entities that are owned by the same *E* entity — discriminating attribute(s).
- Note: *W* could be related to additional entity types, so the relationship type leading to the owning entity type must be indicated.
- E-R diagram:
  - ✓ Weak entity type: double rectangle
  - ✓ Discriminating attribute(s): dashed underline
  - ✓ Relationship to owning entity type: double diamond

# Weak Entity Types (cont.)

- Example:




- Notes:
  - ✓ Each Homework entity is part of a course; it does not have existence of its own; if a Course entity is deleted, then the associated Homework entities must be deleted as well.
  - ✓ The relationship type need not be called PartOf.
  - ✓ The key to Homework is composed of an external attribute and an internal discriminating attribute.
  - ✓ In turn, the owning entity type might also be weak (owned by yet another entity type).

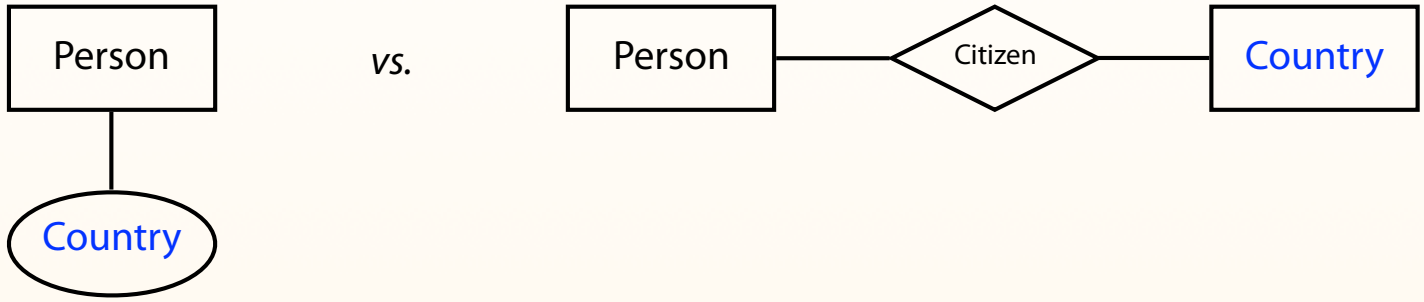
# 5. Design Guidelines

- Designs may often be specified in alternative ways.
  - ✓ Sometimes, it is simply a matter of designer preference.
  - ✓ At other times, a particular design choice is superior to others.
  - ✓ Good design requires considerable experience.

1. Economize: Keep the number of structures to a minimum.
   - ✓ Avoid unnecessary attributes, entity types or relationship types.
2. Avoid redundancy: It leads to inconsistency!
   - ✓ Redundancy may be subtle.
     - Example: Storing both Age and Date_of_Birth for entity type Person.
     - Example: If attribute Total_Students is added to Course, the value stored for a particular course might be inconsistent with the number of students that are enrolled in the course.
     - Example: If attribute Active is added to Student, the value stored for a particular student might be "yes", although the student is not enrolled.
     - In these examples an attribute is included (Age, Total_Students, Active) that could be derived from other information already stored in the database.

# Design Guidelines (cont.)

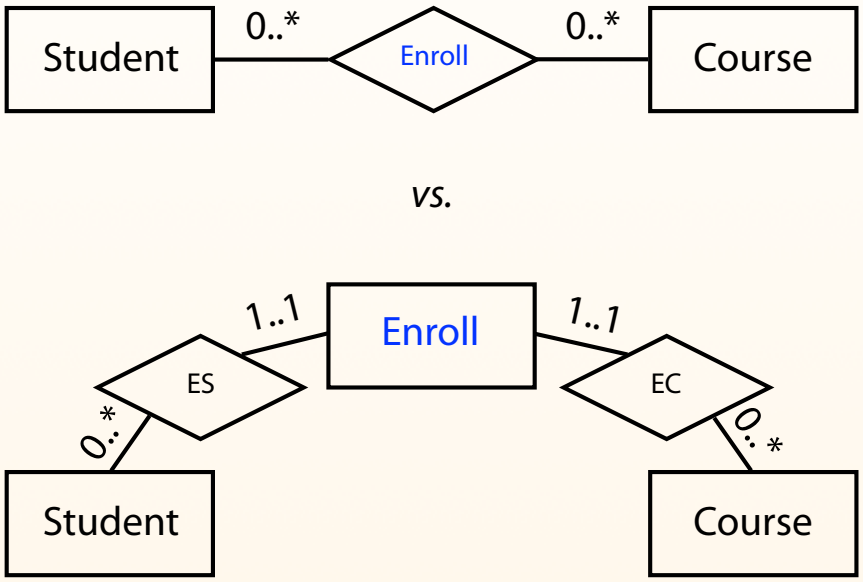
3. Entity type or attribute?




- ✓ if there is information to be stored about each country (other than its name), then it should be an entity type, and the information should be described with attributes (population, etc.).
- ✓ Similarly, if Country is related to other entity types;
  e.g., the relationship type Serve between Country and Airline.

# Design Guidelines (cont.)

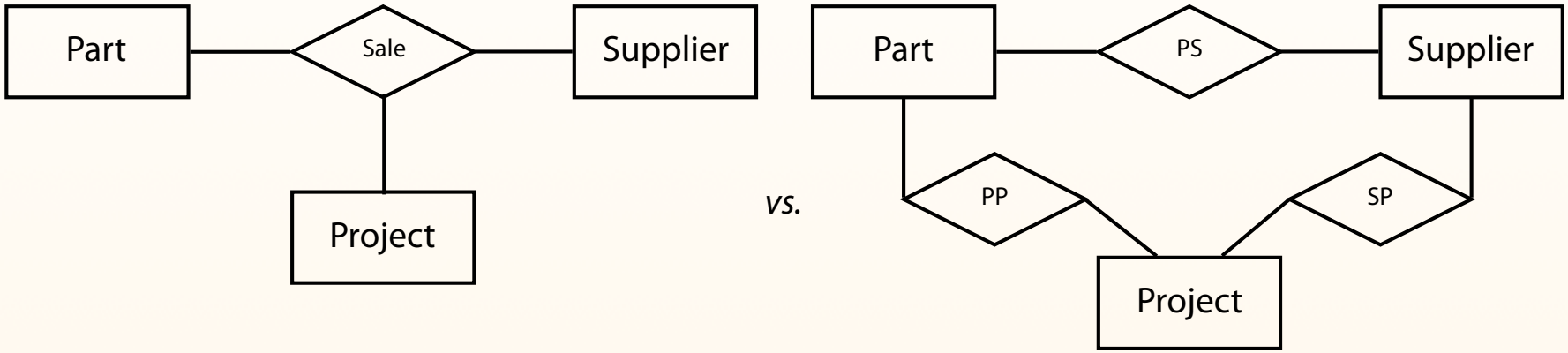
4. Entity type or relationship type?




✓ The second option appears unjustifiably complex.

# Design Guidelines (cont.)

5. Avoid navigation traps:

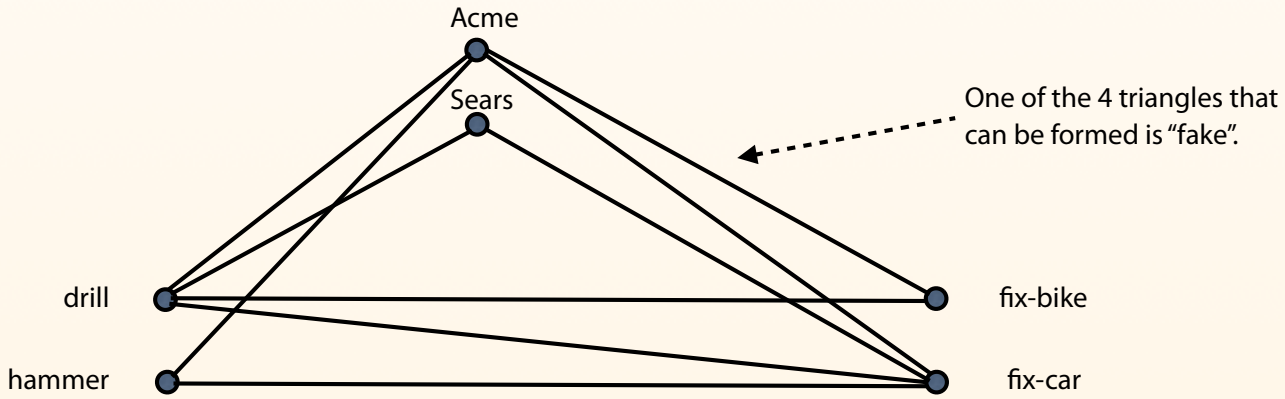



- ✓ In the first option, each Sale relationship associates three entities; <drill, Sears, fix-car>.
- ✓ In the second option, this information is represented with three pairs; <drill, Sears>, <Sears, fix-car>, <fix-car, drill>.
- ✓ It cannot be concluded from the 3 pairs that a drill was purchased from Sears for the project fix-car!
- ✓ In other words, the decomposition in the second option lost the true relationships.

# Design Guidelines (cont.)

5. Avoid navigation traps (cont.):

- ✓ Assume 3 decomposed Sale relationships:
  {<drill, Sears, fix-car>, <drill, Acme, fix-bike>, <hammer, Acme, fix-car>}
- ✓ The 9 decomposed pairs are:
  PS = {<drill, Sears>, <drill, Acme>, <hammer, Acme>}
  SP = {<Sears, fix-car>, <Acme, fix-bike>, <Acme, fix-car>}
  PP = {<fix-car, drill>, <fix-bike, drill>, <fix-car, hammer>}




- ✓ There are multiple ways to "navigate" this diagram to reconstruct the true relationship. For example:
  {<drill, Sears>, <Sears, fix-car>, <fix-car, drill>} — correct!
  {<drill, Acme>, <Acme, fix-car>, <fix-car, drill>} — incorrect!

# Design Guidelines (cont.)

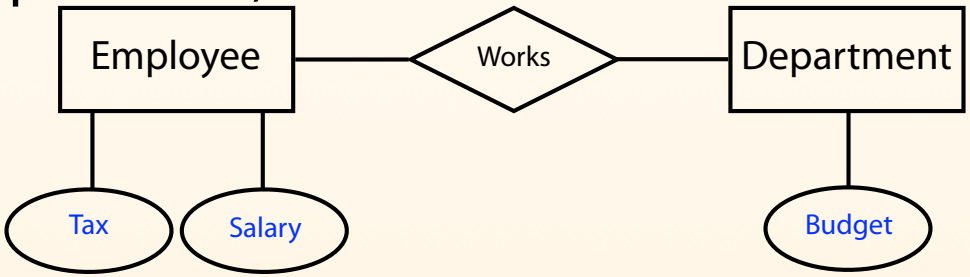
6. Use subtypes judiciously. It may look attractive to construct an elaborate type hierarchy, but it needs to be justified.

- Example: When should we generalize Student and Professor with entity type University_Person?
  - ✓ When there are relationships that associate University_Person, which otherwise would have to be repeated for both Student and Professor.
    - e.g., Assignment to parking lots.
  - ✓ When it is desirable to enforce a disjointness or a covering constraint.
- Example: When should we specialize Student into separate entity types Graduate and Undergraduate (rather than just add an attribute Level to Student)?
  - ✓ When Graduate and Undergraduate are described by different attributes.
  - ✓ When they are associated with different relationship types.
  - ✓ When it is desirable to enforce a disjointness or a covering constraint.

# Design Guidelines (cont.)

7. Use constraints extensively. However, the type of constraints that can be specified is quite limited:
   - ✓ Constraints on entity types: Key constraints.
   - ✓ Constraints on relationship types: Key constraints and participation constraints; or the more general cardinality constraints.
   - ✓ Constraints on IsA relationships: disjointness and covering.

- For example, one cannot specify that the total of the salaries of all the employees in a department is less than the budget of the department, or that tax is smaller than salary.




- Strategy for large designs: Design independent views, each describing a natural subpart, then merge the views in a single design.

# Further Reading

- Chapter 4
  - ✓ Sections: 4.1, 4.2, 4.3, 4.4, 4.9