Session 4: EER: Extended (or Enhanced) ER Model
(CH-2 and 3)
CSCI-585 , Cyrus Shahabi

- Example ER for a real-world problem
- *Generalization* is the result of computing the union of two or more entity sets (or *subclasses*) to produce a higher-level entity set (or *superclass*). It represents the containment relationship that exists between the higher-level entity set and one or more lower level entity sets.
- *Specialization* constructs the lower level entity sets that are a subset of higher-level entity set. Specialization is the reverse of generalization (for the remainder of this session we focus on specialization without loss of generality). Example:

(Attribute inheritance)

- There might exist many specialization of the same entity set based on different distinguishing characteristics. Hence, an entity can be a member of a number of subclasses. Example:

- An entity cannot merely exist by being a member of a subclass but no superclass. However, it is not essential that every entity in a superclass be a member of some subclass.
- Why specialization:
  1. Define a set of subclasses of an entity set.
  2. Associate additional specific attributes with each subclass.
  3. Establish additional specific relationship sets between each subclass and other entity sets.

Different types of specialization

- **Predicate-defined** (or condition-defined): Determine subclass membership by examining the value of a specific attribute (termed, **defining attribute**).

- **User-defined**: The user specifies subclass membership individually for each entity.

- **Disjoint**: An entity can be a member of *at most one* of the subclasses.
- **Overlap**: When the subclasses are not disjoint.

- **Total:** Every entity in the superclass must be a member of some subclass.
- **Partial:** An entity might belong to no subclass.

EER-to-Relational Mapping

- Option 1: One table for superclass + two tables for subclasses (one for each) consisting of their corresponding attributes plus the primary key of the superclass.

- Option 2: Same as option 1, but without creating a table for the superclass:

(Only if specialization is both *disjoint* and *total*)

- Option 3: A singles table including all the attributes of the superclass and all subclasses, plus an extra ***type attribute t*** to indicate the subclass to which each tuple belongs:

(Only if specialization is *disjoint*; null value for t if *partial*; t can be the defining attribute for the predicate-defined specialization)

- Option 4: Same as option 3 except there are *m* Boolean *type attributes*, one for each subclass:

(This option can support *overlap* specialization)