ER Model: Entities

- **Entity - Relation Model (ER Model)** is high-level model of some microworld
- Represents data in way familiar to people
- Basic elements of ER Model are *entities* and *relation(ship)s*
- **Entities**
  - Represent unique concepts
  - Entities are defined in terms of their attribute values
    - *Attribute*: Characteristic of entity
  - Can represent an entity using following syntax:
    - *entity_name*(att_value_1, att_value_2, ..., att_value_n)
    - Technically, entities do not have names, so an alternative representation is
      \[
      \{(att_{name1}, att_{value1}), ..., (att_{name_n}, att_{value_n})\}
      \]
- **Entity set**: Set of entities of same type
- **Entity type**: Schema for ES
  - Entity type represented by a name and a set of attribute names
  - E.g., *type_name*(att_name_1, att_name_2, ..., att_name_n)
- **In comparison to programming languages**
  - Entity type is data type declaration
  - Entity set is set of legal values of that data type
  - Entity is a single value of the set
- **Entity types are intensions (abstract), entity sets are extensions (concrete)**
ER Model: Attributes

- Attribute is function mapping entity set into a domain (value set)
- Domain is set of legal values for the attribute
- Types of attributes:

  1. Atomic
     - Have no structure
     - Simple
       * Represent single values
       * Formally: Let $E$ be an entity set and $D$ be the domain of attribute $A$
         - Then $A : E \rightarrow D$
     - Multivalued
       * Represent multiple (non-structured) values
       * Represent using braces:
         $\{\text{attribute\_name}\}$
       * Formally: Let $E$ be an entity set and $D$ be the domain of attribute $A$
         - Then $A : E \rightarrow P(D)$, where $P(D)$ is the power set of $D$

  2. Composite
     - Have hierarchical structure
     - Represent using nesting:
       $\text{attribute\_name}(\text{sub\_name}_1, \text{sub\_name}_2, \ldots, \text{sub\_name}_n)$
     - Formally: Let $E$ be an entity set and $D_1, D_2, \ldots, D_n$ be the domains of
       the atomic (leaf) components of attribute $A$
       * Then $A : E \rightarrow P(D_1) \times P(D_2) \times \ldots \times P(D_1)$

  3. Derived
     - Not physically represented
     - Can be determined from other attributes
     - Attributes not derived called stored attributes
ER Model: Keys

- Entities are unique
- Can only be distinguished by their attribute values
- Super key: Set of attributes that uniquely identify an entity in a given entity set
- Candidate key: Super key such that no subset is a super key
- Composite key: Key consisting of several attributes
- Strong ES: One with a superkey
- Weak ES: One without a superkey
- Partial Key/Discriminator: Set of attributes that uniquely id a weak entity wrt a strong entity
ER Model: Relation(ships)

- **Relation**: Association among two or more entities
- **Relation set**: Set of relations of same type
  - Formally: If $RS$ is a relation set defined on entity sets $ES_1, ES_2, ..., ES_n$, then
    $$RS \subseteq ES_1 \times ES_2 \times ... \times ES_n = \times_{i=1}^n ES_i$$
  - Alternative notation: \(RS = \{(e_1, e_2, ..., e_n)|e_1 \in ES_1, e_2 \in ES_2, ..., e_n \in ES_n\}\)
  - Each n-tuple \((e_1, e_2, ..., e_n)\) is a relation
- **Relation type**: Schema for a relation set
  - Degree/Arity of a relation type: Number of entity type’s that it relates
  - Role: Function an entity type plays in a relation type
  - Recursive relation type: One that relates an entity type to itself
  - Relation type may have attributes
ER Model: Constraints

- Structural constraints impose restrictions on relation types
- Types of constraints:
  1. Cardinality ratio (mapping constraint)
     - Specify the number of entities that can be associated with another in a relation set
     - Types:
       (a) 1:1
         * Each entity in set $A$ can be associated with only 1 entity from set $B$, and *vice versa*

![Diagram of 1:1 relationship]

(b) 1:many
  * Each entity in set $A$ can be associated with several entities from set $B$;
  * Each entity in set $B$ can be associated with only 1 entity from set $A$
(c) Many:many

* Each entity in set $A$ can be associated with several entities from set $B$, and *vice versa*

2. Participation

- Specify whether entities in a participating entity set must be related to entities in another participating set

- Types:
  (a) Total (existence dependency)

  * Entity set $A$ has total participation in relation $R$ if every entity in $A$ must be associated with an entity in set $B$ also participating in $R$.
  * If entity $b \in B$ were deleted, would have to delete associated entity $a \in A$
  * Total participation also known as *existence dependency* for this reason

  * Weak entity sets always have total participation wrt a strong entity set
    - The weak set is referred to as the dependent/subordinate/child set
    - The strong set is referred to as the owner/dominant/parent set
    - The relation associating the weak and strong entity sets is referred to as the *identifying relation*
ER Model: Constraints (3)

(b) Partial

- Entity set $A$ has partial participation in relation $R$ if entities in $A$ do not need to be associated with an entity in set $B$ also participating in $R$
- Such entities can exist independently of entities in $B$
- In the diagram, $A$ has partial participation, $B$ has total participation
• Generalization: High-level entity type that represents superset of two or more entity type’s
  – The generalization is a superclass
  – The original entity types are subclasses of the generalization
  – Generalization represents the attributes the subclasses have in common
  – The subclasses inherit the attributes of the generalization
  – Subclasses may have additional attributes

• Specialization: Low-level entity type that represents subset of entity type
  – Needed when
    * Subclasses have atts specific to each
    * When subclasses participate individually in relations

• Types of subclasses:
  – Predicate-defined subclass: Entity type for which membership depends on specific value of an attribute in the superclass
    * Condition called defining predicate
  – Attribute-defined specialization: One in which all subclasses are predicate-defined, based on the same attribute
  – User-defined subclass: Independent of any attribute
ER Model: Specialization and Generalization (2)

- Types of subclass constraints:
  1. Disjointness
     - Specifies whether entities may participate in more than one subclass of a given superclass
     - If the subclasses represent a partition (are mutually exclusive), they are disjoint
     - Attribute-defined subclasses are implicitly disjoint
     - If an entity may belong to several subclasses, the subclasses overlap
  2. Completeness
     (a) Total participation
        - Every member of superclass is a member of some subclass
        - Generalization usually results in total participation
     (b) Partial participation
        - Not every member of superclass is a member of some subclass

- Union type/Category
  - Subclass with multiple superclasses, whose members belong to only one of the superclasses
  - Category members are a subset of the union of the superclass members
  - Entities in a category inherit only those attributes from their respective superclasses
  - Have constraints similar to generalization/specialization
    1. Total participation
       * Every member of superclass is a member of some category
       * Comparable to generalization/specialization in opposite direction
    2. Partial participation
       * Not every member of superclass is a member of some category
ER Model: Creating ER Diagrams

1. Id entity type
   • Usually expressed in terms of nouns
   • Represent concepts that have independent existence

2. Id relation types
   • Usually expressed in terms of verbs
   • Most often have arity of 2
   • Many relations can usually be inferred from specs;
     - Only represent those explicitly stated or required for transactions
   • Usually name represents 1:n direction
   • For 1:1 and many:many, chose direction with simpler name (active verb v passive verb)

3. Id attributes
   • Usually done concurrently with above steps
   • Deciding on simple v composite:
     - If components need to accessed individually, make composite
   • Deciding on derived:
     - Often not explicitly represented
     - If dependent on another attribute that may be modified, represent explicitly

4. Id candidate keys
   • ALL candidate keys should be indicated
     - DB implementers make decision as to which will be used
   • If a CK is composite, must be represented as a composite attribute to distinguish it from other CKs
     - Introduces a bit of inconsistency in representation
5. Specialize/generalize

- This is optional
- Dependent on model developed to this point
- Main consideration is whether a subclass needs to participate in relations independently of others

6. Draw diagram
ER Model: Creating ER Diagrams - Decisions

- Entity type or att?

  - Does concept have properties of its own that need to be modeled?
  - Does concept participate in relations on its own?
  - If yes for either, use entity type

- Entity type or relation type?

  - Does concept participate in relations on its own?
  - If yes, use entity type
• Binary v ternary relation type?

- Ternary types can usually be represented as binary types
- A) does not contain as much info as original
- B) is better approach
ER Model: Creating ER Diagrams - Problems

• Fan traps

  – Situation that arises when have 2 entity types related to a third intermediate entity type

  \[
  \begin{array}{c}
  E_1 \\ M & R_1 & 1 \\ E_2 & 1 & R_2 \\ & N & E_3 \\
  \end{array}
  \]

  – Want to be able to associate entities in \( E_1 \) with unique entities in \( E_3 \), but cannot

\[
\begin{aligned}
& e_{1a} & r_1 & e_{2a} & r_2 & e_{3a} \\
& e_{1b} & & e_{2a} & & e_{3b}
\end{aligned}
\]

  – Results from having M:1:N cardinality ratios

  – To correct, reorganize as follows:

\[
\begin{array}{c}
E_2 & 1 & R_1 & M \\ & E_1 & 1 & R_2 \\ & & N & E_3 \\
\end{array}
\]

\[
\begin{aligned}
& e_{2a} & r_1 & e_{1a} & r_2 & e_{3a} \\
& e_{1b} & & e_{2a} & & e_{3b}
\end{aligned}
\]
Chasm traps

Situation that arises when have 2 entity types related to a third intermediate entity type

Want to be able to associate entities in \( E_1 \) with unique entities in \( E_3 \), but no path exists between \( E_1 \) and \( E_3 \)

Results from having partial and total participation of one entity type in relations

To correct, add an additional relation as follows:
ER Model: UML

- **Universal Markup Language** is a graphical language for representing object-oriented concepts
- Can be used instead of ER model
- Correspondence between ER model and UML
  1. Entity type corresponds to UML class
  2. Entity corresponds to UML object
  3. Attributes are the same in each model
  4. UML methods represent operations associated with a class
  5. Relation type corresponds to UML association
  6. Relation instance corresponds to UML link
  7. Cardinality ratio corresponds to UML multiplicity
  8. Recursive relation corresponds to UML reflexive association
  9. Weak entity type corresponds to UML qualified association
  10. Partial corresponds to UML discriminator
  11. UML aggregation represents *has-a* relationship

- UML representation:

```plaintext
Class Name
Attributes
Methods

C1   C2
Binary Association

A
Association Attributes

C1   C2
1:1 Association

C1   C2
0:1 Association

C1   C2
0:∞ Association

Unidirectional Association

C1   C2
Qualified Association with discriminator d

C1   C2
Aggregation

S
Disjoint Leaf Classes

S
Overlapping Leaf Classes
```
1. Classes
2. Attributes:
   - Simple attributes:
     * Name
     * Name: domain_name
   - Composite attributes:
     * Name: domain_name
     component attributes
     * Component attributes are names of attributes that compose the structure of domain_name
     * Domain_name is called a structured domain
3. Associations:
   - May be named
   - Attributes called link attributes
4. Multiplicities
   - May explicitly label cardinalities on links, or use special link types
   - May use asterisk for infinity
5. Association direction
   - Unidirectional associations limit direction of access of objects
6. Aggregation
7. Specialization/generalization