The Entity-Relationship Model

Steps in Database Design

1) Requirement Analysis
   - Identify the data that needs to be stored
     - data requirements
   - Identify the operations that need to be executed on the data
     - functional requirements

2) Conceptual Database Design
   - Use a semantic data model to develop semantic schema

3) Map semantic schema to relational schema
Requirement Analysis

- Similar to first phase of general software design
- focus: data & functional requirements
- Questions to ask:
  - What are the entities and relationships in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the integrity constraints or business rules that hold?
  - What operations do we want to execute on the entities and relationships?
- Tools: data-flow diagrams, sequence diagrams...
  - In this course: half-formal, structured specification in plain English

Requirement Analysis: Student Database

- The database contains information about persons:
  - Data:
    - each person has an identifying ID
    - each person has a PIN, name, permanent address + phone number, 0 or more emergency contacts
  - Functionality
    - all data can be viewed and changed
- A person can be a student. A student has
  - Data:
    - a permanent code, an email, (account information)
    - for past terms:
      - courses taken and grades
      - Term gpa
      - cum gpa up to this term
      - total credits up to this term
    - for current term:
      - a faculty, 0 or more programs, courses registered
Student Database (contd)

- A person can be an employee:
  - it has an employee email address, pay information etc.
- A person can be a faculty:
  - Only faculty can teach a course

Requirement Analysis: Student Database

- A course
  - ...
- Functionality:
  - students can register in course
  - ...

421B: Database Systems - ER Model
Semantic Data Model: Entity-Relationship Model (ER)

- The E/R model is a language that allows for a pictorially description of the data determined through the requirement analysis.
  - An E/R diagram or schema is a representation of the data model of the application.
  - An ER schema should be understandable by non-computer scientists.
- The main concepts of the E/R model are entities and relationships.
- An ER schema can be translated into the relational schema in a quite straightforward way.

Entity

- **Entity**: Real-world object distinguishable from other objects. An entity is described using a set of **attributes**.
  - Similar to an object in OO.
- **Entity Set**: A collection of similar entities, e.g., all employees (similarity to a "class" in OO).
  - All entities in an entity set have the same attributes (until we consider ISA hierarchies later...). An attribute describes a property of the entity set.
  - Each entity set has a **key**:
    - Minimum set of attributes whose values uniquely identify an entity in the set.
- Each attribute has a **domain**:
  - Integer, 20-character string.

Entity set = rectangle, Attribute = oval
**ISA ("is a") Hierarchies: Subclasses**

- **Subclass** = special case = fewer entities = more properties
- "A ISA B" then every A entity is also a B entity
  - Key only in B.

- **Reasons for using Subclasses:**
  - Additional descriptive attributes specific for a subclass
  - Identification of a subclass that participates in a relationship (will see later)

- **Where do the entities reside:**
  - E/R-viewpoint: An entity has a component in each entity set to which it logically belongs. Its properties are the union of these entity sets.
  - Contrast OO-viewpoint: An object belongs to exactly one class. The subclass inherits the attributes of its superclass.

---

**ISA Hierarchies (Contd.)**

- **Overlap Constraint**: Can an entity be in more than one subclass? (allowed/disallowed)
- **Covering Constraint**: Must every entity of the superclass be in one of the subclasses? (yes/no)

- **Developing Class Hierarchies**
  - **Specialization**: Identifying subsets of an entity set (the superclass) that share some distinguishing characteristic. Represents top-down design: superclass is first, then subclasses are defined and specific attributes and relationships are set.
  - **Generalization**: Several entity sets are generalized into a common entity set. Represents bottom-up design: common characteristics of a collection of entity sets are identified, a superclass entity set is built with these characteristics as attributes.

- Theoretically multiple inheritance possible, in practice not used.
**Relationship**

- **Relationship**: Association among two or more entities. E.g. Employee A works in department X.
- **Relationship Set**: Collection of similar relationships
  - An n-ary relationship set \( R \) relates n entity sets \( E_1...E_n \); each relationship in \( R \) involves entities \( e_1 \in E_1, ..., e_n \in E_n \)
  - Same entity set could participate in different relationship sets, or in different *roles* in the same set.

**ER Model (contd.)**

- Employees with different roles in a relationship
  - Role indicator: supervisor/subordinate
**Multiplicity: Many-to-many**

- **Works_In:**
  - Each entity of Employees can have relationships many entities from Departments
    - Employee A works in departments X and Y
  - Each entity of Departments can have relationships with many entities from Employees
    - Department X has several employees that work there (e.g., employees A and B)
  - Many-many relationship between departments and employees
  - Note: A relationship is uniquely defined by the primary keys of participating entities
    - Employee A cannot work twice in the department X

![ER Diagram](image)

**Multiplicity: one-to-many, many-to-one**

- **Manages:**
  - One employee can manage several departments; each department is managed by only one employee;
  - One-to-many relationship between employee and department
  - Many-to-one relationship between department and employee;
  - The condition “each department is managed by only one employee” is called a **key constraint** on Manages and depicted with arrow from Departments to Manages
**Multiplicity: one-to-one**

- **Manages-2:**
  - Each employee can only manage one department; each department is managed by only one employee;
  - One-to-one relationship between Employees and Departments
  - Key constraints in both directions

One to one

- **Participation Constraints**
  - **Works_in2:**
    - Each employee must work in at least one department
    - The *participation constraint* requires that the participation of Employee in Works_in2 is total (vs. partial) (depicted with a thick line)
Participation Constraints

- **Works_in3:**
  - Each employee works in exactly one department
  - key constraint: at most once
  - participation constraint: at least one
  - combined: exactly once

![Diagram of Employees, Works_In, and Departments]

Weak Entities

- A **weak** entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - The key in weak entity set is the union of the key of the owner entity set and a set of its own attributes (underlined with dashes)
  - Owner entity set and weak entity set must participate in a **supporting one-to-many** relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this **identifying set.**

![Diagram of Employees, Policy, and Dependents]
Ternary Relationship

3-ary relationship

- **Examples**
  - Employee A works in department X at the Montreal location
  - Employee B works in department X and the Toronto location

- **Note that given relationship set is many-many-many**
  - Employee A works in department X at the Montreal location and in department Y at the Montreal location
  - Or Employee A works in department X at the Montreal location and department Y at the Lachine location

- **Examples**
  - Employee A works in department X at the Montreal location
  - Employee B works in department X and the Toronto location

Aggregation

- **Aggregation** allows us to treat a relationship set R as an entity set so that R can participate in (other relationships).

- **Example**
  - Each department can sponsor many projects
  - Each project is sponsored by at least one dep
  - Each sponsorship is monitored by at least one employee

- **Aggregation vs. ternary relationship:**
  - Monitors is a distinct relationship with a descriptive attribute
  - Can provide additional multiplicity constraints
Conceptual Design with ER

- Design Principles:
  - Keep it simple
  - Avoid redundancies
  - Capture as many constraints as possible
- Design Choices
  - Entity or attribute?
  - Attributes and relationships
  - Identifying relationships: Binary or ternary?
  - Aggregation?

Design Principles

- Avoid redundancies

Left side:
- The address of a supplier is stored with each part it provides; if the supplier provides more than one part, then the address is stored several times.

Right side:
- By making supplier its own entity-set all supplier related information is stored only once.
### Design Principles

- **Keep it simple**

  - **Left side:**
    - The only information that exists about a supplier is its name and that is the primary key.

  - **Right side:**
    - For each part we keep the supplier name in an extra attribute. As there is no other information about suppliers there is no redundancy.

### Relationships and Attributes

- Employees: `eid`, `name`, `salary`  
  - Manages: `did`, `dname`, `budget`  
  - ISA: `mbudget`

- Managers: `mbudget`

- Departments: `did`, `dname`, `budget`
Entity vs. Attribute

- Should address be an attribute of Employees or an entity (connected to Employees by a relationship)?
- Depends on the use and the semantics of the information
  - If we an arbitrary number of addresses for each employee, address must be an entity (since attributes cannot be set-valued).
  - If address represents a lodging or place of business and hence is a real object that is independent of the people who live or work there, then it should be an entity.
  - Note, if we want to capture the structure of address (because we want to retrieve all employees in a given city), we must use several separate attributes (street, zip, city, etc.). This can happen within the Employees entity or as an extra entity.

Entity vs. Relationship
**Binary vs. Ternary vs. Aggregation**

- Situation:
  - Stars play in movies and movies are produced by studios
  - Several constraints possible:
    - Each movie is only produced by one studio or
    - A star has only a single contract for a movie (even if several studios involved) or
    - A contract for a movie is always between a star and a single studio (even if several studios involved)

- Solution I:
  - Ternary relation
  - Each arrow would lead to a nonsense restriction
  - Consequence:
    - Each movie can be produced by several studios
    - But only possible if at least one star has contract with this studio for this movie

- Solution II:
  - Two binary relations
  - A star plays now exactly once for a movie (with one contract)
  - But if several studios produce a movie we cannot express with which studio the contract is done

- Solution III:
  - Aggregation
  - Movie produced by one studio possible
  - Star can have one or more contracts for a movie; each with a different studio
Binary vs. Ternary Relationships

- **Situation I:**
  - Employee can own several policies
  - Each policy can be owned by several employees
  - Each dependent can be covered by several policies
  - A policy can cover several dependents

- **Situation II:**
  - Employee can own several policies
  - Each policy has only one owner
  - Each dependent is identified by its policy
  - A policy can cover several dependents