The Entity-Relationship and Enhanced Entity-Relationship Models

Objectives

- Understand the role of high level conceptual data models
- Know the concepts of the Entity-Relationship and Enhanced Entity-Relationship models
- Be able to build an Entity-Relationship or an Enhanced Entity-Relationship model for a particular application area

Data Models

- A database model is a collection of logical constructs used to represent the data structure and the data relationships found within the database.
- Database models can be grouped into two categories:
  - Conceptual Models
  - Implementation Models

Conceptual Model

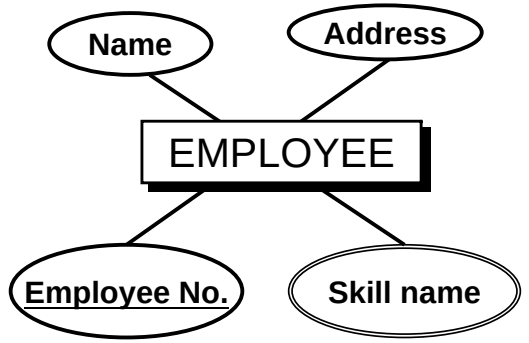
- Focuses on the logical nature of the data representation.
- Basically concerned with what is represented in the database rather than how is it represented
- Includes the Entity Relationship (E-R) model and the Enhanced Entity-Relationship (ERR) model

Implementation Model

- Places the emphasis on how the data is represented in the database or on how the data structures are implemented.
- Includes hierarchical, network, relational and the object-oriented database model.
- Hierarchical and network databases have all but disappeared and are only really ever studied by historians of database technologies

Entity Relationship Data Model

- E-R models are normally represented in an entity relationship diagram (ERD), which uses graphic representations to model the database components.



E-R Data Model Components

- The basic E-R data model is based on the following components:
  - Entity: is a person, place or thing about which data are to be collected and stored
  - Attribute: this describes a particular characteristic of the entity.
  - Relationship: describes an association between two or more entities.

Entity

- Entities are specific objects or things in the mini-world that are represented in the database
- Examples:
  - Person: EMPLOYEE, STUDENT, PATIENT
  - Place: STATE, REGION, COUNTRY
  - Object: MACHINE, BUILDING
  - Event: SALE. REGISTRATION
  - Concept: ACCOUNT, COURSE

Entity

Attribute

- Attribute is a property or characteristic used to describe an entity
- Both entity and relationships may have attributes.
- Different types of attributes: Simple or Composite; Single-valued or Multi-valued; and Stored or Derived.
- Example:
  - STUDENT: STUDENT NO.,NAME, ADDRESS
  - EMPLOYEE: EMPLOYEE NO., NAME, SKILL

Types of Attributes

- Simple
  - Each entity has a single atomic value for the attribute
    - E.g. Sex.
- Composite
  - The attribute may be composed of several components
  - Composition may form a hierarchy where some components are themselves composite.
    - E.g. Name(FirstName, MiddleName, LastName).
- Multi-valued
  - An entity may have multiple values for that attribute
    - E.g. Color of a CAR can be denoted as {Color}

Example of Entity and Attribute

- A specific entity will have a value for each of its attributes;
  - For example:
    - A specific EMPLOYEE entity may have
      - Name = ‘John Smith’
      - Address = ‘Causeway Bay’
      - Sex = ‘M’
      - BirthDate = ‘09-JAN-1980’.

Entity Types

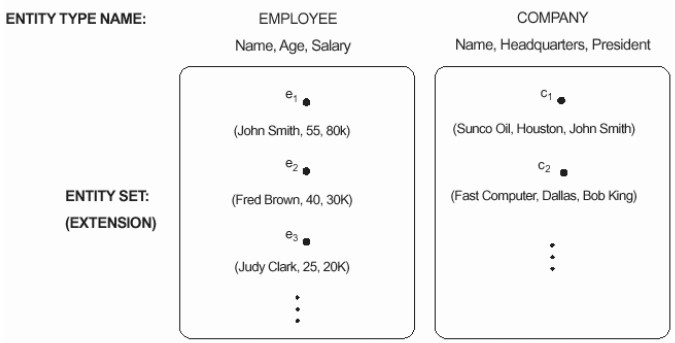
- Entities with the same basic attributes are grouped or typed into an **Entity Type**.
  - E.g. the EMPLOYEE entity type or the PROJECT entity type.

Key Attributes

- An attribute of an entity type for which each entity must have a unique value is called a **Key Attribute** of the entity type.
  - E.g. ID of EMPLOYEE.
- A key attribute may be composite
  - E.g. VehicleRegistrationNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
  - E.g. the CAR entity type may have two keys: VehicleIdentificationNumber and VehicleRegistrationNumber(Number, State).

Entity Types and Key Attribute

- Two entity types named EMPLOYEE and COMPANY,and some of the member entities in the collection of entities (or entity set) of each type.



Weak Entity Types

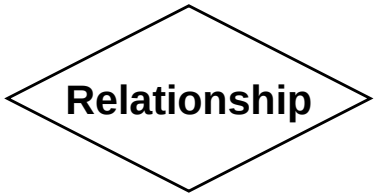
- An entity type that does not have a key attribute
- A weak entity type must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of :
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying entity type

Example

- Suppose that a DEPENDENT entity is identified by the dependent's first name and birthdate, and the specific EMPLOYEE that the dependent is related to.
- DEPENDENT is a weak entity type with EMPLOYEE as its identifying entity type via the identifying relationship type DEPENDENT_OF.

Relationship

- A **Relationship** relates two or more distinct entities with a specific meaning;
  - E.g. EMPLOYEE John Smith *works on* the ProductX PROJECT or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT



Relationship Type

- Relationships of the same type are grouped or typed into a **Relationship Type**.
  - E.g. the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.

Relationship Type

- A Relationship Type can have attributes;
  - E.g. HoursPerWeek of WORKS_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.

Degree of Relationship

- A relationship has a degree (the number of participating entity types in the relationship). The degree of a relationship type is the number of participating entity types.
  - A relationship with degree two is called a **Binary Relationship**
  - If the degree is three it is called a **Ternary Relationship**
  - If there are N entity types participating it is known as an **N-ary Relationship** (In general, an n-ary relationship is not equivalent to n binary relationships)

Structural Constraints on Relationships

- Cardinality ratio (of a binary relationship):
  - 1:1
  - 1:N or N:1
  - M:N
- Participation constraint (on each participating entity type):
  - Total (called existence dependency)
  - Partial

Three Main Types of Cardinalities

- A relationship can have one of three different cardinalities
  - One-to-One Relationship (1:1)
  - One-to-Many Relationship (1:m)
  - Many-to-Many Relationship (M:n)

One-to-One Relationship (1:1)

- It exists when exactly one of the second entity occurs for each instance of the first entity.

One-to-Many Relationship (1:M)

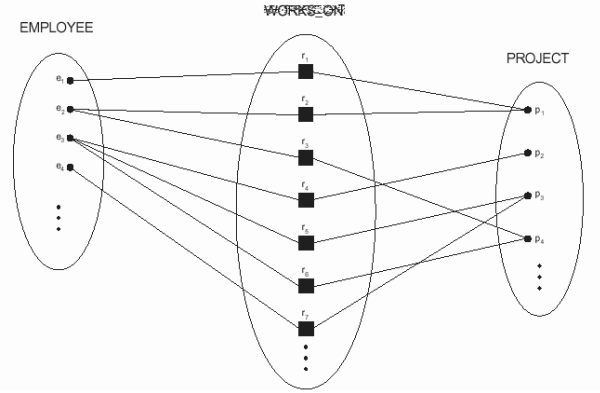
- It exists when one occurrence of the first entity can be related to many occurrences of the second entity, but each occurrence of the second entity can be associated with only one occurrence of the first entity.

Many-to-Many Relationship (M:N)

- It exists when one instance of the first entity can be related to many instances of the second entity, and one instance of the second entity can be related to many instances of the first entity.

Example

- The M:N relationship WORKS_ON between EMPLOYEE and PROJECT.

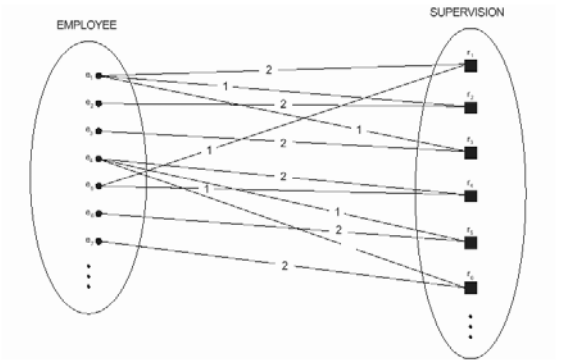


Structural Constraints and Roles

- A relationship can relate two entities of the same entity type
  - E.g. a SUPERVISION relationship type relates one EMPLOYEE (in the role of supervisee) to another EMPLOYEE (in the role of supervisor).
- This is called a **Recursive Relationship** type.

Example

- The recursive relationship SUPERVISION, where the EMPLOYEE entity type plays the two roles of supervisor (1) and supervisee (2).



Participation Constraints

- Constraints tell us whether the existence of an entity depends on it being part of a particular relationship
  - Total Participation
  - Partial Participation

Example of Total Participation

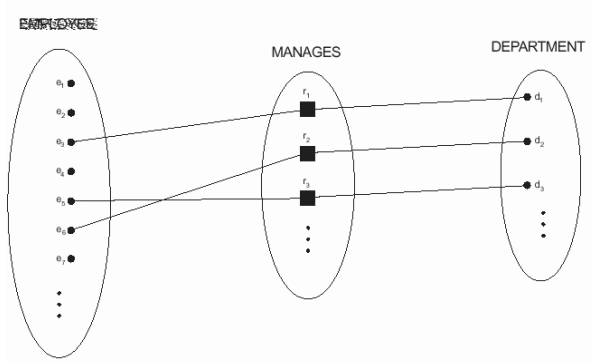
- If University policy says that every student must be enrolled on a degree course then a student entity can only exist if it participates in an ENROLLED_ON relationship with a degree course.
- This form of participation is called Total.

Example of Partial Participation

- Suppose that not every student lives in a Student Hall.
- The participation of student in the relationship LIVES_IN with the entity Student Hall would be partial, meaning that some students live in Hall while others do not.
- This form of participation is called Partial

Example

- The 1:1 relationship MANAGES, with partial participation of EMPLOYEE and total participation of DEPARTMENT.



Alternative Notation for Relationship Structural Constraints

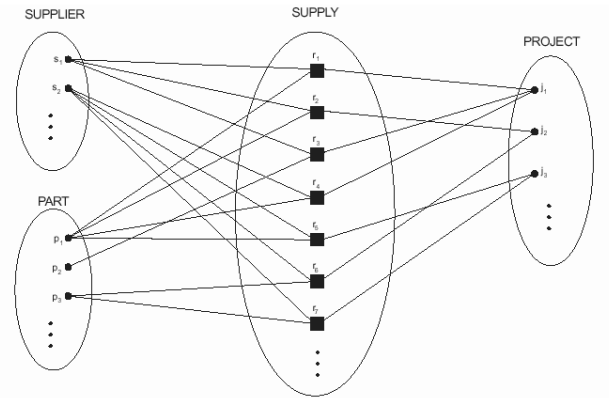
- Specified on each participation of an entity type E in a relationship type R.
- Specifies that each entity e in E participates in at least ***min*** and at most ***max*** relationship instances in R.
- Default (no constraint): min=0, max=n.
- Must have min<max, min>0, max>1.
- Derived from the mini-world constraints.

Examples

- A department has exactly one manager and an employee can manage at most one department.
  - Specify (1,1) for participation of EMPLOYEE in MANAGES
  - Specify (0,1) for participation of EMPLOYEE in MANAGES
- An employee can work for exactly one department but a department can have any number of employees.
  - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
  - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

Example of Higher Degree

- Some relationship instances of a ternary relationship SUPPLY



Enhanced Entity-Relationship Model

- The E-R model has been in use for about 30 years.
- In the immediate past a whole new type of database application areas have emerged.
- They include such things as CAD/CAM databases for engineering, databases storing images and graphics or other multimedia data, geographic information systems and databases used to index the Internet.
- All of these have more complicated structures than can be easily modeled by the E-R technique.
- So additional features have been added in and this leads to the **Enhanced Entity-Relationship** (EER) model.

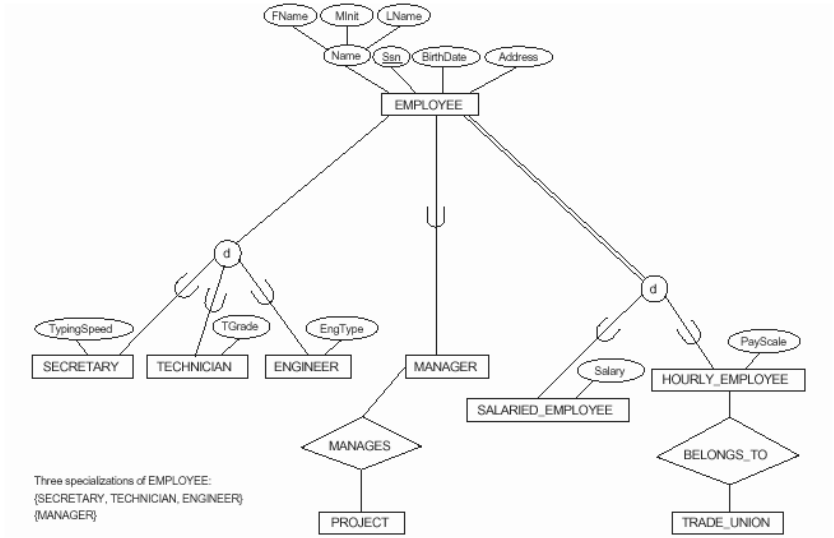
Feature of Enhanced Entity-Relationship Model

- Superclass/Subclass Relationships
- Inheritance
- Specialization and Generalization
- Category

Superclasses and subclasses

- For instance we might want to distinguish between UNDERGRADUATE, TAUGHT_POSTGRADUATE and RESEARCH_POSTGRADUATE students.
- Each of these is a subclass of the entity type STUDENT.
- We also say that STUDENT is a superclass of the three subclasses.

Notation for Representing Specialization and Subclasses



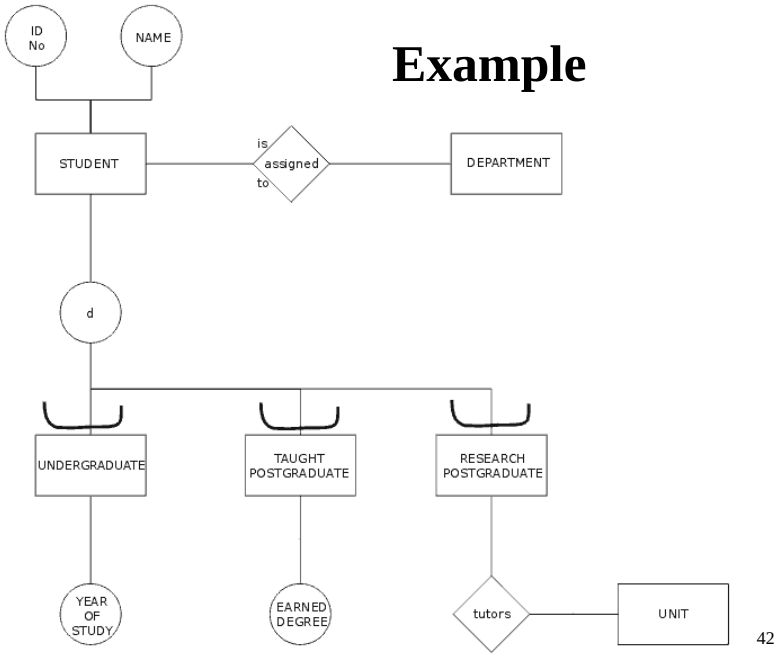
Inheritance

- If we think of an entity type as being defined by the attributes it has and the relationship types in which it participates, then a subclass entity type inherits all the attributes of its superclass as well as all the relationship types that the superclass participates in.
- It can then add its own individual attributes and participate in its own relationships.

Example

- The STUDENT superclass has a relationship with the DEPARTMENT entity and this will be inherited by each of the three subclasses.
- They will also inherit the two attributes of STUDENT.
- However two of the subclasses then add their own attributes and the other participates in its own relationship with another entity, UNIT.

Example



Specialization and Generalization

- **Specialization** is used to describe the process by means of which we define a set of subclasses of a particular entity type.
- **Generalization** is used to describe the process by which we suppress differences between several entity types, concentrate on their common features and combine them into a superclass, for which the original entity types are subclasses.

Constraints on Specializations and Generalizations

- Membership Constraint
- Disjointness Constraint
- Completeness Constraint

Membership Constraint

- Determine whether a particular entity does or does not belong to a particular subclass.
- We will decide on membership of a subclass by use of the value of an attribute of the superclass.
  - E.g. an attribute StudentType for STUDENT that takes one of the values from the set {undergraduate, taught postgraduate, research postgraduate}.
  - when StudentType = “undergraduate”, the corresponding entity will belong to the subclass UNDERGRADUATE.
- The subclass is a **Predicate-defined Subclass** and state that: StudentType = “undergraduate”

Membership Constraint

- Now in our example all the subclasses would have the membership condition defined on the same attribute of the superclass, and in this case we call this an attribute-defined specialization.
- In those case when we don’t have an explicit condition for determining membership in a subclass we say the specialization is User-defined.
- This will mean that membership of a subclass is determined by the database users as they add a new entity to the database; it will be specified individually not calculated automatically.

Disjointness Constraint

- This specifies that the subclasses of a specialization must be disjoint.
  - i.e. an entity can only belong to one subclass of the specialization.
- This is shown by putting a “d” into the specialization circle.
- If this is not the case, in other words if an entity type could belong to more than one subclass, then the subclasses are overlapping and an “o” is placed in the specialization circle to show this.

Completeness Constraint

- This is either total or partial.
  - Total implies that every entity in the superclass must belong to some subclass of the specialization.
    - E.g. if every STUDENT must be either a HOME_STUDENT or an OVERSEAS_STUDENT then the specialization of STUDENT into those two subclasses is total.
    - We show this by drawing a double line from the superclass to the specialization circle.

Completeness Constraint

- In contrast, there may be other types of student other than undergraduates and the two types of postgraduate we have previously identified (students taking just individual units or on non-degree courses for instance) so this specialization will be partial.
- There will be some students not in the specialization subclasses.

Rules to Control Insertion and Deletion of Entities

- Deleting an entity from a superclass implies that it is automatically deleted from all the subclasses to which it belongs.
- Inserting an entity in a superclass implies that the entity is mandatorily inserted in all predicate-defined (or attribute-defined) subclasses for which the entity satisfies the defining predicate.
- Inserting an entity in a superclass of a total specialization implies that the entity is mandatorily inserted in at least one of the subclasses of the specialization.