Chapter 2

Modeling Tools for Systems Analyst

Entity Relationship Diagram (E-R Diagram)

During *analysis* phase, a systems analyst uses **entity relationship data model (E-R model)** as a **conceptual data model**. A **conceptual data model** is a detailed model that captures the overall structure of organizational data while being independent of any database management system or other implementation consideration. And an **E-R model** is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships or associations among those entities, and the attributes or properties of both the entities and their relationships. An E-R model is normally expressed as an **entity relationship diagram (E-R diagram)**, which is a graphical representation of an E-R model. It has three basic concepts: **entities**, **attributes**, and **relationships**.

Entities

An entity is a person, place, object, event, or concept in the user environment about which the organization wishes to capture and store data. An **entity type** (sometimes called an **entity class** or **entity set**) is a collection of entities that share common properties or characteristics. An **entity instance** (or **instance**) is a single occurrence of an entity type.

Each entity type in E-R diagram is given a name. When naming entity types, we should use the following guidelines:

- An entity type name is a *singular noun* like CUSTOMER, STUDENT, or AUTOMOBILE.
- An entity type name should be *descriptive and specific to the organization* like PURCHASE ORDER for orders placed with suppliers to distinguish it from CUSTOMER ORDER for orders placed by customers.
- An entity type name should be *concise* like REGISTRATION for the event of a student registering for a class rather than STUDENT REGISTRATION FOR CLASS.
- *Event entity types* should be named for the *result of the event*, not the activity or process of the event like the event of a project manager assigning an employee to work on a project results in an ASSIGNMENT.

Each entity type in E-R diagram should be defined. When defining entity types, we should use the following guidelines:

- An entity type definition should include a statement of *what the unique characteristic(s) is (are) for each instance*.
- An entity type definition should make clear *what entity instances are included and not included* in the entity type.
- An entity type definition often includes a description of *when an instance of the entity type is created or deleted*.
- For some entity types the definition must specify w*hen an instance might change into an instance of another entity type*. For example, a bid for a construction company

becomes a contract once it is accepted.

- For some entity types the definition must specify *what history is to be kept about entity instances*.

An entity type in E-R diagram is drawn using **rectangle**. This shape represents all instances of the named entity. We place entity type name inside the rectangle. For example, the figure below shows STUDENT entity type.

Attributes

Each entity type has a set of attributes associated with it. An attribute is a property or characteristic of an entity that is of interest to the organization. For example, STUDENT entity type may have Student_ID, Student_Name, Home_Address, Phone_Number, and Major as its attributes. Similarly, EMPLOYEE entity type may have Employee_ID, Employee_name, and Skill, Address as its attributes.

Each attribute in E-R diagram is given a name. When naming attributes, we should use the following guidelines:

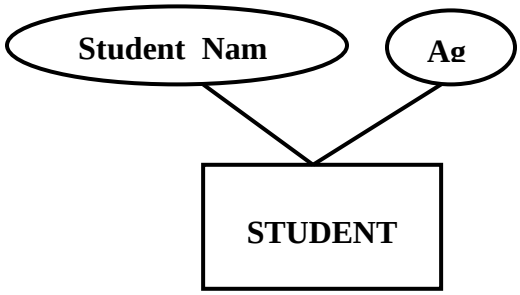
- An attribute is a *noun* like Customer_ID, Age, or Skill.
- An attribute name should be *unique*. No two attributes of the same entity type may have the same name, and it is desirable, for clarity purposes, that no two attributes across all entity types have the same name.
- To make an attribute unique and clear, *each attribute name should follow a standard form*. For example, Student_GPA as opposed to GPA_of_Students.
- Similar attributes of different entity types should use the similar but distinguishing names. For example, Student_Residence_City_Name for STUDENT entity type and Faculty_Residence_City_Name for FACULTY entity type.

Each attribute in E-R diagram should be defined. When defining attributes, we should use the following guidelines:

- An attribute definition states *what the attribute is and possibly why it is important*.
- An attribute definition should make it clear *what is included and what is not included* in the attributes value. For example, Employee_Monthly_Salary_Amount is the amount paid each month exclusive of any benefits, bonuses, or special payments.
- An attribute definition may contain any *aliases* or alternative names.
- An attribute definition may state *the source of values for the attribute to make the meaning clearer*.
- An attribute definition should indicate *if a value for the attribute is required or optional* (to maintain data integrity).
- An attribute definition may indicate *if a value for the attribute may change* (to maintain data integrity).
- An attribute definition may also indicate any *relationships that an attribute has with other attributes*. For example, Age is determined from for Date_of_Birth.

An attribute in E-R diagram is drawn using an **ellipse**. We place attribute name inside the ellipse with a line connecting it to the associated entity type. For example, the figure

below shows Student_Name and Age attributes of STUDENT entity type.



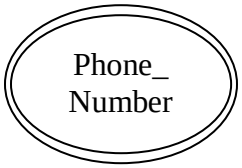
Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. A **candidate key** is an attribute or combination of attributes that uniquely identifies each instance of an entity type. For example, a candidate key for a STUDENT entity type might be Stident_ID.

Some entity type may have more than one candidate key. In such a case, we must choose one of the candidate keys as the identifier. An **identifier** (or **primary key**) is a candidate key that has been selected to be used as the unique characteristic for an entity type. We can use the following selection rules to select identifiers:

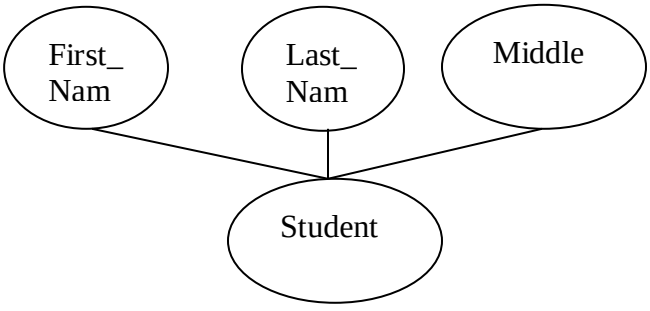
- Choose a candidate key that will not change its value over the life of each instance of the entity type.
- Choose a candidate key that will never be null.
- Avoid using *intelligent keys*.
- Consider substituting single value *surrogate keys* for large composite keys.

The name of the identifier is underlined on an E-R diagram. For example, the figure below shows Student_Id as an identifier for a STUDENT entity type.

A **multivalued attribute** may take more than one value for each entity instance. For example Phone_Number attribute of STUDENT entity type. We use a double-lined ellipse to represent multivalued attribute.



An attribute that has meaningful component parts is called **composite attribute**. For example, Student_Name attribute of STUDENT entity type has First_Name, Middle_Name, and Last_Nmae as its component parts.



An attribute whose value can be computed from related attribute values is called **derived attribute**. For example, value of Age attribute is computed from Date_of_Birth attribute. We use dashed ellipse to denote derived attribute.

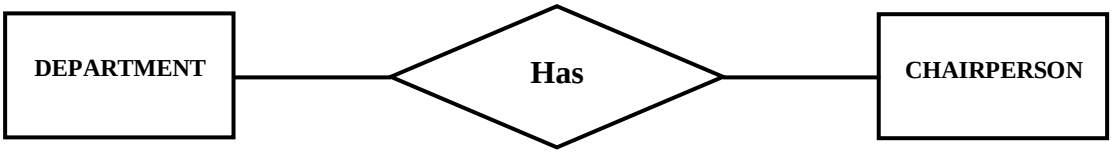
Relationships

A **relationship** is an association between the instances of one or more entity types that is of interest to the organization. We use **diamond** to denote relationships. Relationships are labeled with *verb phrases*. For example, if a training department in a company is interested in tracking with training courses each of its employees has completed, this leads to a relationship (called Completes) between the EMPLOYEE and COURSE entity types as shown in the figure below.

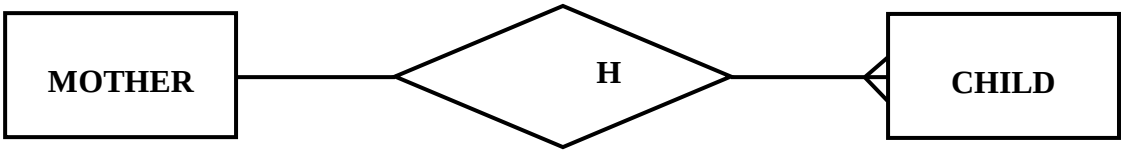
As indicated by arrows, the *cardinality* of this relationship is many-to-many since each employee may complete more than one course, and each course may be completed by more than one employee.

The **cardinality** of a relationship is the number of instances of one entity type that can (or must) be associated with each instance of another entity type. The cardinality of a relationship can be in one of the following four forms: *one-to-one*, *one-to-many*, *many-to-one*, and *many-to-many*.

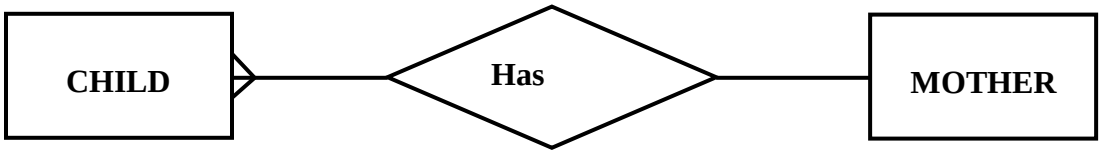
- **One-to-one:** An instance in entity type **A** is associated with at most one instance in entity type **B**, and an instance in entity type **B** is associated with at most one instance in entity type **A**. For example, cardinality between DEPARTMENT and CHAIRPERSON.



- **One-to-many:** An instance in entity type **A** is associated with any number (zero or more) of instances in entity type **B**, and an instance in entity type **B**, however, can be associated with at most one instance in entity type **A**. For example, cardinality between MOTHER and CHILD.



- **Many-to-one:** An instance in entity type **A** is associated with at most one instance in entity type **B**, and an instance in entity type **B**, however, can be associated with any number (zero or more) of instances in entity type **A**. For example, cardinality between CHILD and MOTHER.



- **Many-to-many:** An instance in entity type **A** is associated with any number (zero or more) of instances in entity type **B**, and an instance in entity type **B** is associated with any number (zero or more) of instances in entity type **A**. For example, cardinality between STUDENT and COURSE.

The **degree** of a relationship is the number of entity types that participate in the relationship. The three most common relationships in E-R models are *unary* (degree one), *binary* (degree two), and *ternary* (degree three). Higher degree relationships are also possible, but they are rarely encountered.

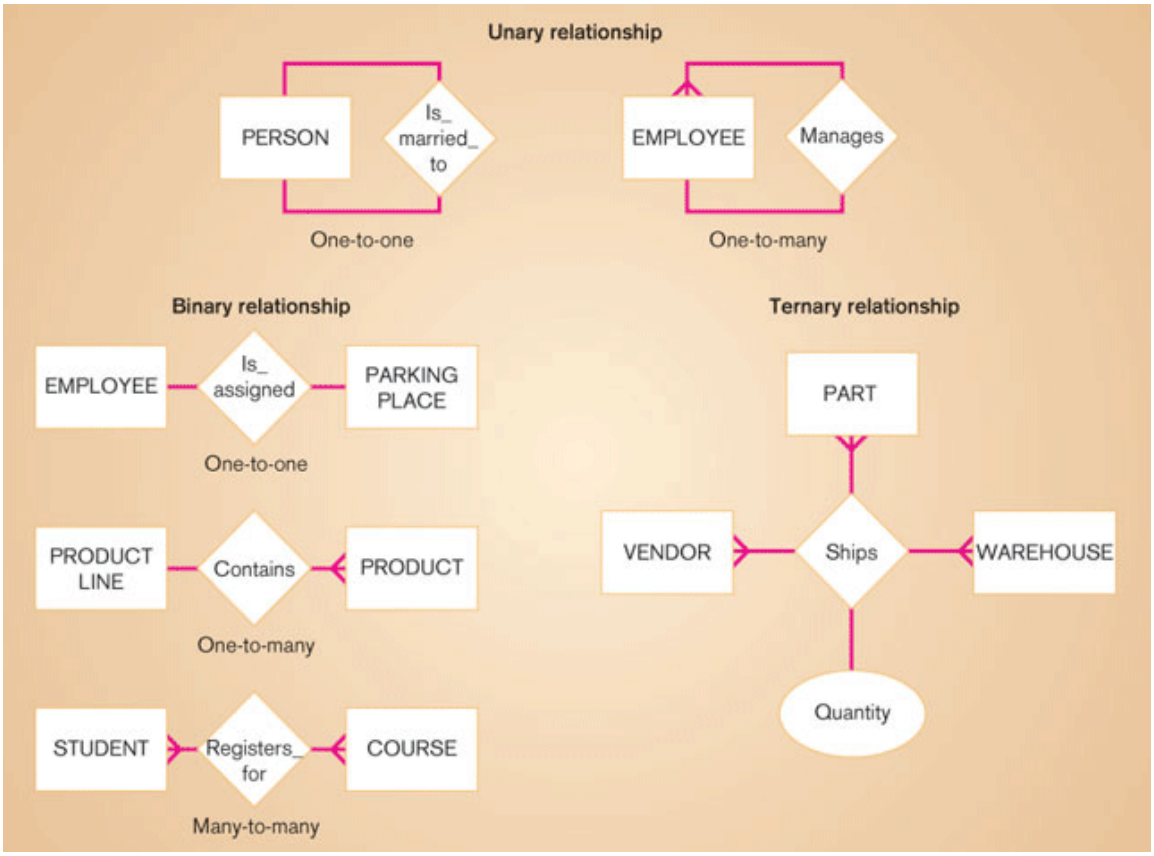


Fig: Example relationships of different degrees

A **unary relationship** is a relationship between the instances of one entity type. It is also called a **recursive relationship**. A **binary relationship** is a relationship between instances of two entity types. This is the most common type of relationship encountered in data modeling. A **ternary relationship** is a simultaneous relationship among the instances of three entity types.

We can also use the concept *minimum* and *maximum* cardinality when we draw E-R diagrams. The **minimum cardinality** of a relationship is the minimum number of instances of an entity type that may be associated with each instance of another entity type. The **maximum cardinality** of a relationship is the maximum number of instances of an entity type that may be associated with each instance of another entity type. For example, suppose a portion of banking database as shown in the figure below. Here, the minimum number of accounts for a customer is one and the maximum number of accounts is many (unspecified number greater than one). Similarly, the minimum number of customers associated with an account is one and the maximum number of customers is many.

When the minimum cardinality of a relationship is zero, then we say that the entity type is an **optional participation** or **partial participation** in the relationship. When the minimum cardinality of a relationship is one, then we say that the entity type is a **mandatory participation** or **total participation** in the relationship. In the figure above, the entity type ACCOUNT is a mandatory participation in the relationship. Similarly, the entity type CUSTOMER is a mandatory participation.

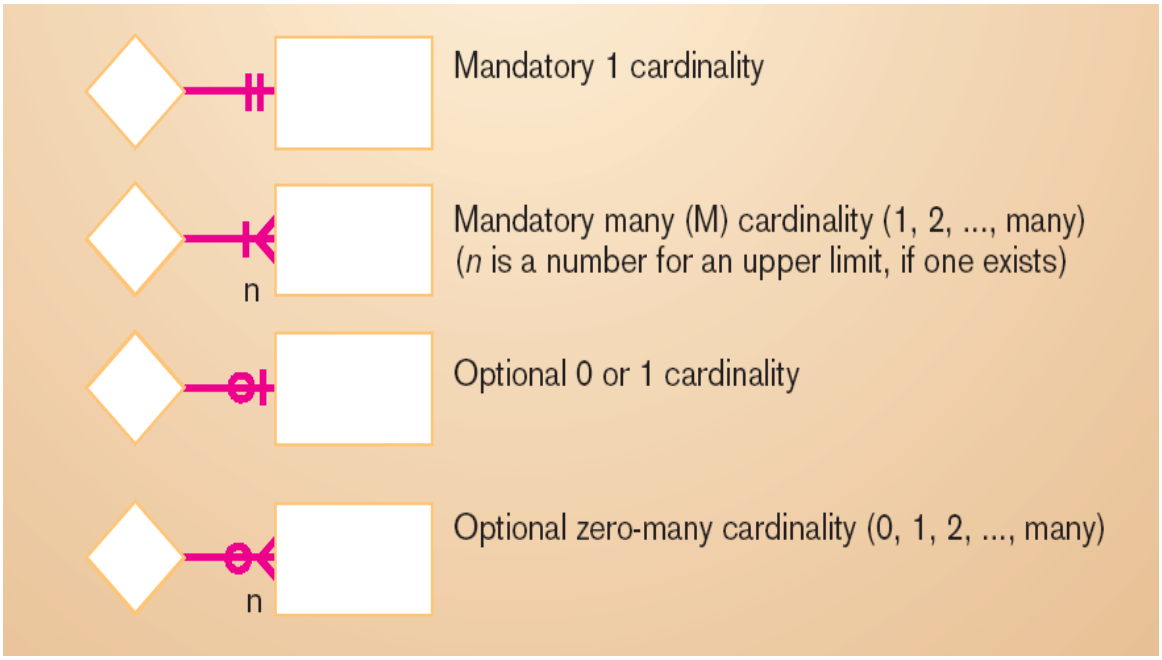


Fig: Cardinality symbols

Each relationship in E-R diagram is given a name. When naming relationships, we should use the following guidelines:

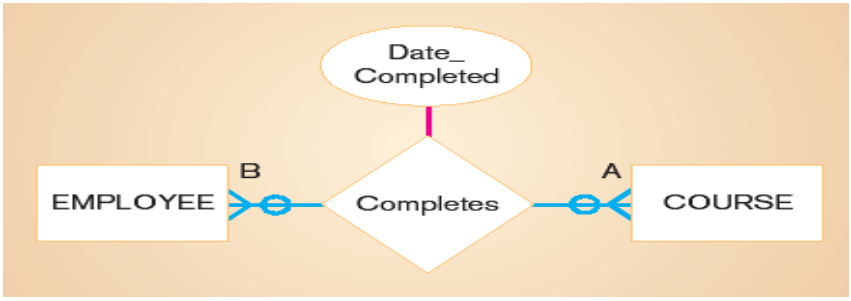
- A relationship name is *verb phrase* like Assigned_to, Supplies, or Teaches. This name represents action taken, not the result of the action, usually in the present tense.
- A relationship name should *avoid vague names*, such as Has or Is_related_to.

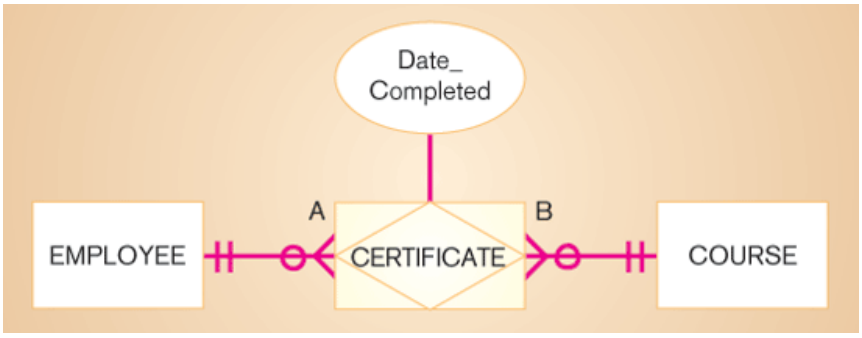
Each relationship in E-R diagram should be defined. When defining relationships, we should use the following guidelines:

- A relationship definition should explain *what action is being taken and possibly why it is important*.
- It may be important to *give examples to clarify the action*.
- The definition should explain any *optional participation*.
- A relationship definition should *explain any restrictions on participation in the relationship*.
- A relationship definition should *explain the extent of history that is kept in the relationship*.
- A relationship definition should *explain the reason for any explicit maximum cardinality* other than many.
- A relationship definition should *explain whether an entity instance* involved in a relationship instance *can transfer participation to another relationship instance*.

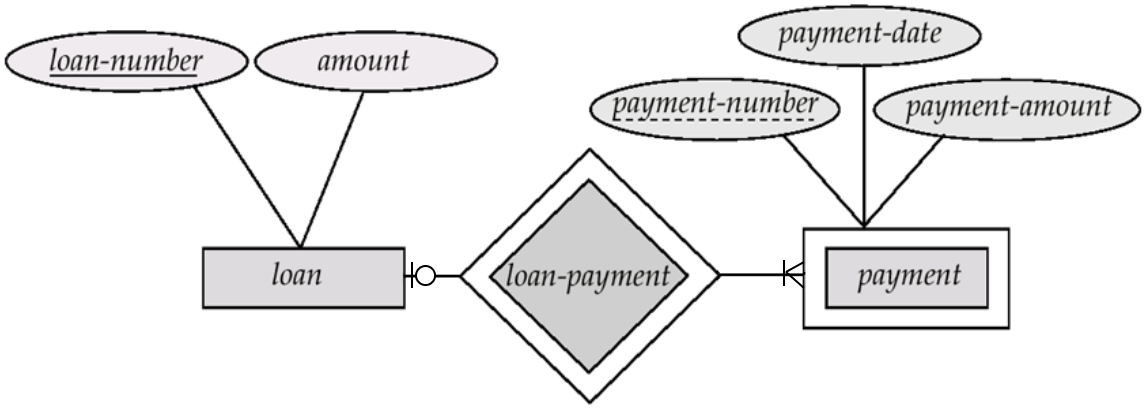
Other E-R Notations

- **Associative Entity:** An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances is called **associative entity**. Associative entity is also called a **gerund**. For example, the first figure below shows a relationship with an attribute and the second figure shows an associative entity.





- **Weak Entity Type:** An entity type may not have sufficient attributes to form a primary key. Such an entity type is termed as a weak entity type. An entity type that has a primary key is termed as a strong entity type. For a weak entity type to be meaningful, it must be associated with another entity type, called the identifying or owner entity type, using one of the key attribute of owner entity type. The weak entity type is said to be existence dependent on the identifying entity type. The relationship associating the weak entity type with the identifying entity type is called the identifying relationship. The identifying relationship is many-to-one form the weak entity type to the identifying entity type, and the participation of the weak entity type in the relationship is total. Although a weak entity type does not have a primary key, we use discriminator (or partial key) as a set of attributes that allows the distinction to be made among all the entities in the weak entity type.



Supertypes and Subtypes

- **Subtype:** Subtype is a subgrouping of the entities in an entity type that is meaningful to the organization and that shares common attributes or relationships distinct from other subgroupings. For example, an entity type STUDENT to GRADUATE STUDENT and UNDERGRADUATE SUTDENT.
- **Supertype:** Supertype is a generic entity type that has a relationship with one or more subtypes. For example PATIENT with OUTPATIENT and RESIDENTPATIENT.

There are several important business rules for supertype/subtype relationships.

- **Total specialization rule:** Specifies that each entity instance of the supertype must be a member of some subtype in the relationship.
- **Partial specialization rule:** Specifies that an entity instance of the supertype does not have to belong to any subtype. May or may not be an instance of one of the subtypes.
- **Disjoint rule:** Specifies that if an entity instance of the supertype is a member of one subtype, it cannot simultaneously be a member of any other subtype.
- **Overlap rule:** Specifies that an entity instance can simultaneously be a member of two (or more) subtypes.

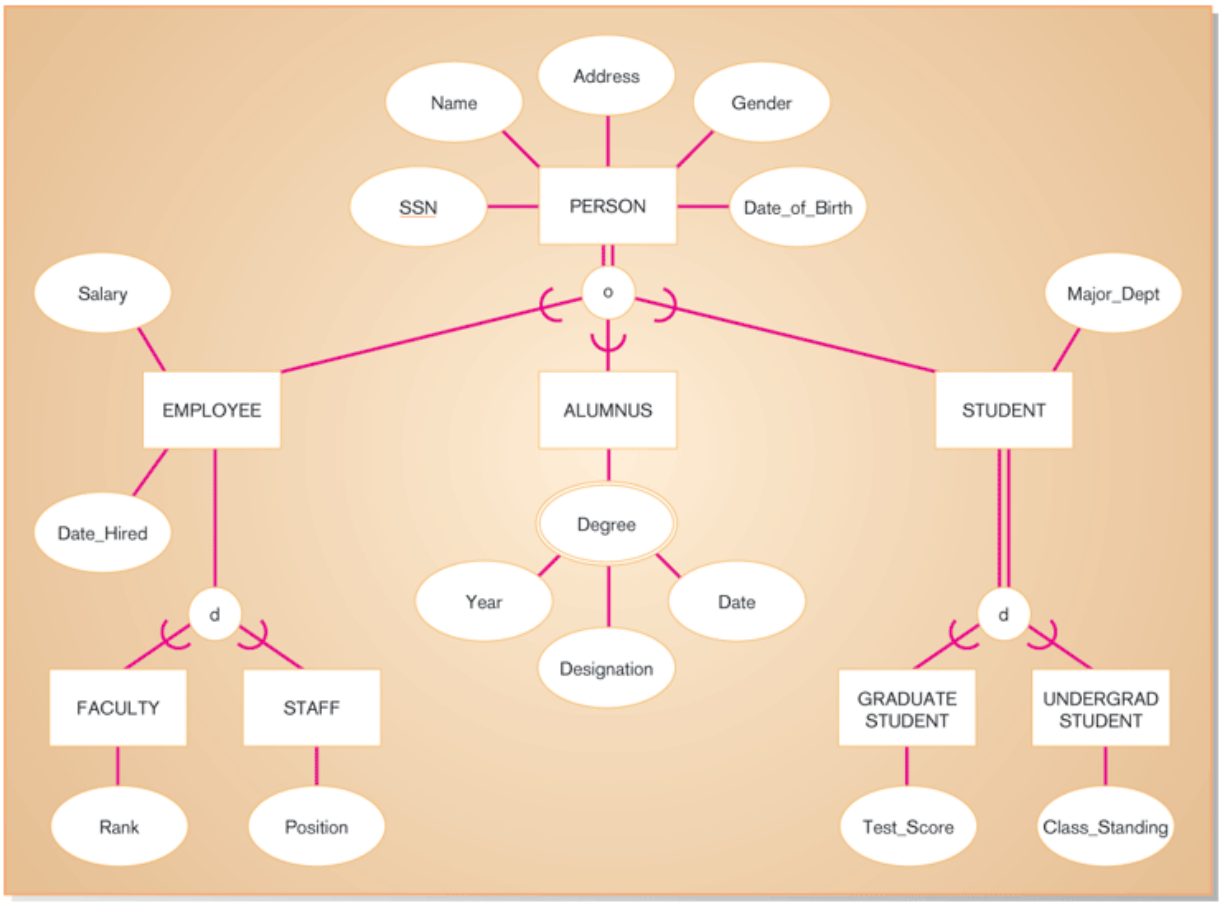


Fig: Example of supertype/subtype hierarchy

In the figure above, PERSON is supertype and EMPLOYEE, ALUMNUS, and STUDENT are subtypes. Similarly, EMPLOYEE is supertype for FACULTY and STAFF subtypes and STUDENT is supertype for GRADUATE STUDENT and UNDERGRADUATE STUDENT subtypes.

Supertype is connected with a line to a circle, which in turn is connected by a line to the subtypes. The U-shaped symbol indicates that the subtype is a subset of the supertype. Total specialization is shown by a double line from the supertype to the circle and partial specialization is shown by a single line. Disjoint versus overlap is shown by a "d" or an "o" in the circle.

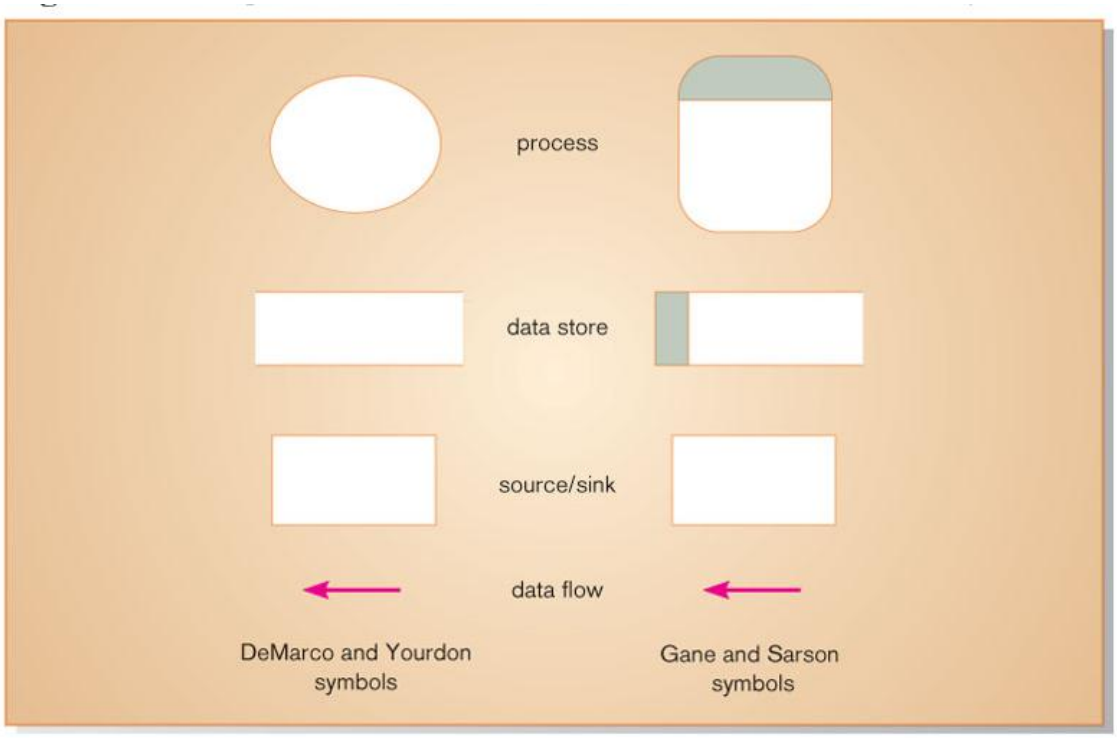
Some Exercises

1. Construct an ER diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
2. Construct an ER diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examinations conducted.
3. Construct an ER diagram of the library system in your college.
4. Construct an ER diagram to maintain data about students, instructors, semester, and courses in a college.
5. Construct an ERD to record the marks that students get in different exams of different course offerings.

Data Flow Diagram (DFD)

Process modeling involves graphically representing the functions or processes that capture, manipulate, store, and distribute data between a system and its environment and between components within a system. A common form of a process model is a **data flow diagram (DFD)**.

A **data flow diagram (DFD)** is a tool that depicts the flow of data through a system and the work or processing performed by that system. It is also called **bubble chart**, **transformation graph**, or **process model**. There are two different sets of data flow diagram symbols, but each set consists of four symbols that represent the same things: **data flows**, **data stores**, **processes**, and **sources/sinks** (or **external entities**). The figure below shows two different sets of symbols developed by DeMacro and Yourdon and Gane and Sarson. The set of symbols we will use here was deviced by Gane and Sarson.



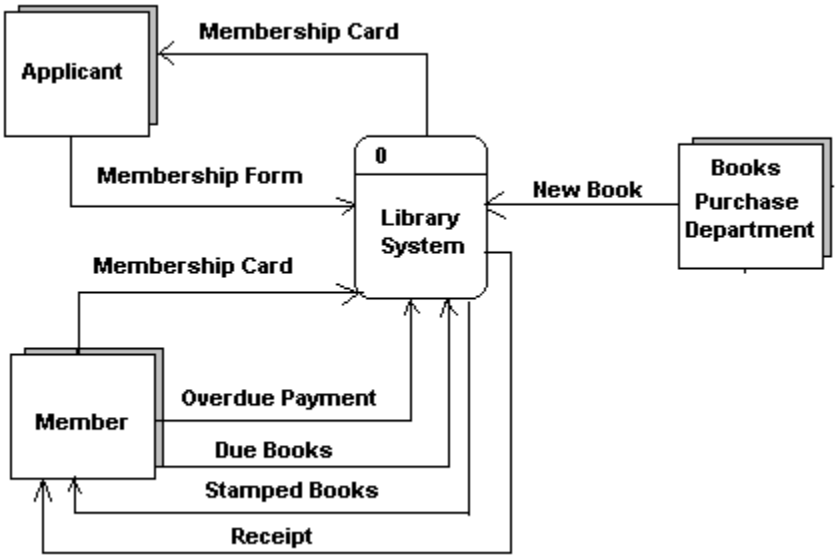
According to *Gane and Sarson*, **rounded rectangles** represent *process* or *work* to be done, **squares** represent *external agents*, **open-ended boxes** represent *data store* (sometimes called *files* or *databases*), and **arrows** represent *data flows* or *inputs* and *outputs* to and from the processes.

Process is the work or actions performed on data so that they are transformed, stored or distributed. *Data store* is the data at rest (inside the system) that may take the form of many different physical representations. External entity (source/sink) is the origin and/or destination of data. Data flow represents data in motion, moving from one place in a system to another.

Developing DFDs

The highest-level view of a system is called **context diagram** or **context DFD**. A context DFD is used to document the scope of the project of development. It is also called **environmental model**.

A project's scope defines what aspect of the business, an information system or application is supposed to support and how the system being modeled must interact with other systems and the business as a whole. Because the scope of any project is always subject to change, the context diagram is also subject to constant change. The context DFD contains one and only one process and it has no data storage. Sometimes this process is identified by the number "0". The figure below shows the context diagram of library system.



The next step is to draw a DFD with major processes that are represented by the single process in the context diagram. These major processes represent the major functions of the system. This diagram also includes data stores and called **level-0 diagram** (or **level-0 DFD**). A level-0 diagram is a DFD that represents a system's major processes, data flows, and data stores at a high level of detail. In this diagram, sources/sinks should be same as in the context diagram. Each process has a number that ends in .0. The figure below shows level-0 DFD.

Here, we started with a high-level context diagram. Upon thinking more about the system, we saw that the larger system consisted of four processes. This process is called **functional decomposition**. Functional decomposition is an iterative process of breaking the description or perspective of a system down into finer and finer detail. When we repeat the decomposition until we reached the point where no subprocesses can logically be broken down any further, we get the lowest level of DFD called **primitive DFD**.

When we decompose level-0 DFD, we get level-1 DFD. In level-1 DFD, we label subprocesses of process 1.0 to 1.1, 1.2, and so on. Similarly, subprocesses of process 2.0 to 2.1, 2.2, and so on. In general, a **level-n diagram** is a DFD that is generated from **n** nested decompositions from a level-0 diagram.