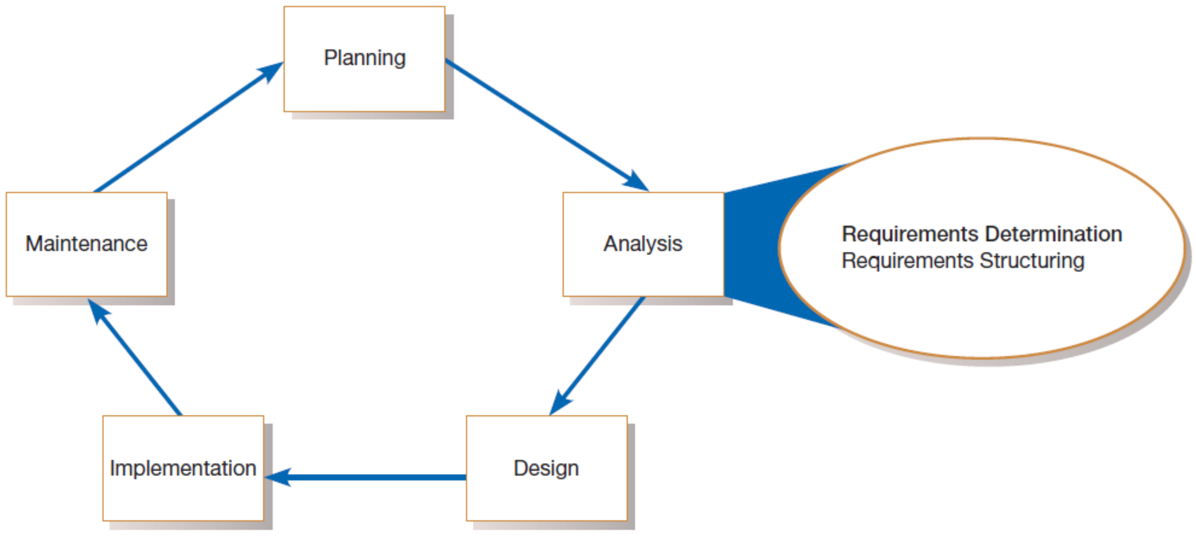
Determining System Requirements

Introduction

Systems analysis is the part of the systems development life cycle in which you determine how the current information system functions and assess what users would like to see in a new system. As shown in the figure below, analysis has two sub-phases: *Requirements determination* and *Requirements structuring*.



Requirements determination is primarily a fact-finding activity. Techniques used in requirements determination have evolved over time to become more structured and increasingly rely on computer support. The most common requirements determination methods include *interviewing*, *observing users in their work environment*, *collecting procedures and other written documents*, Joint Application Design (JAD), prototyping etc.

Performing Requirements Determination

Requirements determination and requirements structuring, the two sub-phases of analysis phase, are considered as parallel and iterative. For example, as you determine some aspects of the current and desired system(s), you begin to structure these requirements or build prototypes to show users how a system might behave.

The Process of determining requirements:

Requirements determination is about determining what the new system should do. During requirements determination, you and other analysts gather information on what the system should do from as many sources as possible: from users of the current system; from observing users; and from reports, forms, and procedures. All of the system requirements are carefully documented and prepared for structuring.

In many ways, gathering system requirements is like conducting any investigation. The characteristics for a good systems analyst during requirements determination sub-phase are:

- *Impertinence* – You should question everything.
- *Impartiality* – Your role is to find the best solution to a business problem or opportunity. You must consider issues raised by all parties and try to find the best organizational solution.
- *Relax constraints* – Assume that anything is possible and eliminate the infeasible.
- *Attention to details* – Every fact must fit with every other fact. One element out of place means that even the best system will fail at some time.

- *Reframing* – Analysis is, in part, a creative process. You must challenge yourself to look at the organization in new ways even though you have already worked with similar systems.

Deliverables and outcomes:
The primary deliverables from requirements determination are the various forms of information gathered during the determination process. In short, anything that the analysis team collects as part of determining system requirements is included in the deliverables. Examples of these deliverables from requirements determination are:

- Information collected from conversations with or observations of users: interview transcripts, notes from observation, meeting minutes
- Existing written information: business mission and strategy statements, sample business forms and reports and computer displays, procedure manuals, job descriptions, training manuals, flowcharts and documentation of existing systems, consultant reports
- Computer-based information: results from JAD sessions, CASE repository contents and reports of existing systems, and displays and reports from system prototypes

These deliverables contain the information you need for systems analysis within the scope of the system you are developing. In addition, you need to understand the following components of an organization:

- The business objectives that drive what and how work is done
- The information people need to do their jobs
- The data (definition, volume, size, etc.) handled within the organization to support the jobs
- When, how, and by whom or what the data are moved, transformed, and stored
- The sequence and other dependencies among different data-handling activities
- The rules governing how data are handled and processed
- Policies and guidelines that describe the nature of the business and the market and environment in which it operates
- Key events affecting data values and when these events occur

Traditional Methods for Determining Requirements

System analysts collect information about the information systems that are currently being used and how users would like to improve the current systems and organizational operations with new or replacement information systems.
One of the best ways to get this information is to talk to the people who are directly or indirectly involved in the different parts of the organizations affected by the possible system changes: users, managers, funders, and so on. Another way to find out about the current system is to gather copies of documentation relevant to current systems and business processes. The traditional methods of collecting system requirements are:

- Individually interview people informed about the operation and issues of the current system and future systems needs
- Interview groups of people with diverse needs to find synergies and contrasts among system requirements
- Observe workers at selected times to see how data are handled and what information people need to do their jobs
- Study business documents to discover reported issues, policies, rules, and directions as well as concrete examples of the use of data and information in the organization

1. Interviewing and Listening

Interviewing is one of the primary ways analysts gather information about an information systems project. Early in a project, an analyst may spend a large amount of time interviewing people about their work, the information they use to do it, and the types of information processing that might supplement their work. Other stakeholders are interviewed to understand organizational direction, policies, expectations managers have on the units they supervise, and other nonroutine aspects of organizational operations. During interviewing you will gather facts, opinions, and speculation and observe body language, emotions, and other signs of what people want and how they assess current systems. Some guidelines you should keep in mind when you interview are:

- Plan the Interview
  - Prepare interviewee: appointment, priming questions
  - Prepare checklist, agenda, and questions
- Listen carefully and take notes (record if permitted)
- Review notes within 48 hours of interview
- Be neutral
- Seek diverse views

Choosing Interview Questions:

You need to decide what mix and sequence of **open-ended** and **closed-ended** questions you will use. **Open-ended** questions have no pre-specified answers. These are usually used to probe for information for which you cannot anticipate all possible responses or for which you do not know the precise question to ask. The person being interviewed is encouraged to talk about whatever interests him or her within the general bounds of the question. One advantage of open-ended questions is that previously unknown information can surface. You can then continue exploring along unexpected lines of inquiry to reveal even more new information. Open-ended questions also often put the interviewees at ease because they are able to respond in their own words using their own structure; open-ended questions give interviewees more of a sense of involvement and control in the interview. A major disadvantage of open-ended questions is the length of time it can take for the questions to be answered. In addition, open-ended questions can be difficult to summarize.

Closed-ended questions provide a range of answers from which the interviewee may choose. Closed-ended questions work well when the major answers to questions are well known. Another plus is that interviews based on closed-ended questions do not necessarily require a large time commitment – more topics can be covered. You can see body language and hear voice tone, which can aid in interpreting the interviewee's responses. A major disadvantage of closed-ended questions is that useful information that does not quite fit into the defined answers may be overlooked as the respondent tries to make a choice instead of providing his or her best answer. Closed-ended questions, like objective questions on an examination, can follow several forms, including the following choices:

- True or false.
- Multiple choice (with only one response or selecting all relevant choices).
- Rating a response or idea on a scale, say from bad to good or strongly agree to strongly disagree. Each point on the scale should have a clear and consistent meaning to each person, and there is usually a neutral point in the middle of the scale.
- Ranking items in order of importance.

Interview Guidelines:

First, with either open- or closed-ended questions, do not phrase a question in a way that implies a right or wrong answer. The respondent must feel that he or she can state his or her true opinion and perspective and that his or her idea will be considered equally with those of others.

The second guideline to remember about interviews is to listen very carefully to what is being said. Take careful notes or, if possible, record the interview with permission. If you run out of time and still need to get information from the person you are talking to, ask to schedule a follow-up interview.

Third, once the interview is over, go back to your office and type up your notes within 48 hours. If you recorded the interview, use the recording to verify the material in your notes. After 48 hours, your memory of the interview will fade quickly. As you type and organize your notes, write down any additional questions that might arise from lapses in your notes or from ambiguous information. Make a list of unclear points that need clarification. Call the person you interviewed and get answers to these new questions. You may also want to send a written copy of your notes to the person you interviewed so the person can check your notes for accuracy. Finally, make sure you thank the person for his or her time.

Fourth, be careful during the interview not to set expectations about the new or replacement system unless you are sure these features will be part of the delivered system. Let respondents know that their ideas will be carefully considered, but that due to the iterative nature of the systems development process, it is premature to say now exactly what the ultimate system will or will not do.

Fifth, seek a variety of perspectives from the interviews. You want to understand all possible perspectives so that in a later approval step you will have information on which to base a recommendation or design decision that all stakeholders can accept.

2. Interviewing Groups

Individual interviews have many drawbacks like contradictions and inconsistencies between interviewees, follow-up discussions are time consuming, and new interviews may reveal new questions that require additional interviews with those interviewed earlier. Clearly, gathering information about an information system through a series of individual interviews and follow-up calls is not an efficient process.

Another option available to you is the **group interview**. In a group interview, several key people are interviewed at once. To make sure all of the important information is collected, you may conduct the interview with one or more analysts. In the case of multiple interviewers, one analyst may ask questions while another takes notes, or different analysts might concentrate on different kinds of information. The number of interviewees involved in the process may range from two to however many you believe can be comfortably accommodated.

A group interview has a few ***advantages***. One, it is a much more effective use of your time than a series of interviews with individuals. Two, interviewing several people together allows them to hear the opinions of other key people and gives them the opportunity to agree or disagree with their peers. Synergies also often occur.

The primary ***disadvantage*** of a group interview is the difficulty in scheduling it. The more people who are involved, the more difficult it will be finding a convenient time and place for everyone. Modern videoconferencing technology can minimize the geographical dispersion factors that make scheduling meetings so difficult.

A specific technique used for working with groups is called **Nominal Group Technique (NGP)**. NGT is exactly what the name indicates – the individuals working together to solve a problem are a group in name only, or nominally. Members come together as a group in a same room, but initially work separately. Each person writes ideas. Facilitator reads ideas out loud, and they are written on a blackboard or flipchart. Group openly discusses the ideas for clarification. Ideas are prioritized, combined, selected, and reduced.

3. Directly Observing Users

We can discover requirements by watching what users do or by how people behave in work situations. It is an effective requirements discovery technique for obtaining an understanding of a system. Observing current system users is a more direct way of seeing how an existing system operates. Here, the systems analyst either participates in or watches a person performing activities to learn about the system.

This technique has many advantages.

- Data gathered can be highly reliable.
- Sometimes observations can be conducted to check the validity of data obtained directly from individuals.
- The systems analyst is able to see exactly what is being done.
- It is relatively inexpensive compared to other fact-finding techniques, because other techniques usually require more employees.
- It allows the system analyst to do work measurement.

This technique also has many drawbacks as given below.

- People usually feel uncomfortable when being watched to their work.
- Some system activities may take place at odd times, causing a scheduling inconvenience for the system analyst.
- It may cause interruption.
- Some tasks may not always be performed by observation.

4. Analyzing Procedures and other Documents

All the methods of collecting information that we have been discussing up until now can be enhanced by examining system and organizational documentation to discover more details about current systems and the organization these systems support.

You should attempt to find all written documents about the organizational areas relevant to the systems under redesign. Few specific documents are organizational mission statements, business plans, organization charts, business policy manuals, job descriptions, internal and external correspondence, and reports from prior organizational studies.

One type of useful document is a written ***work procedure*** for an individual or a work group. The procedure describes how a particular job or task is performed, including data and information that are used and created in the process of performing the job.

A **second type** of document useful to systems analysts is a ***business form***. Forms are used for all types of business functions. Forms are important for understanding a system because they explicitly indicate what data flow in or out of a system and which are necessary for the system to function. A form gives us crucial information about the nature of the organization. A printed form may correspond to a computer display that the system will generate for someone to enter and maintain data or to display data. Forms are most useful to you when they contain actual organizational data because this allows you to determine the characteristics of the data that are actually used by the application.

A **third type** of useful document is a ***report*** generated by current systems. As the primary output for some types of systems, a report enables you to work backward from the information on the report to the data that must have been necessary to generate them. You would analyze such reports to determine which data need to be captured over what time period and what manipulation of these raw data would be necessary to produce each field on the report.

If the current system is computer-based, a **fourth set** of useful documents are those that describe the current information systems – how they were designed and how they work.

Contemporary Methods for Determining System Requirements

Even though we called interviews, observation, and document analysis traditional methods for determining a system's requirements, all of these methods are still very much used by analysts to collect important information. Today, however, there are additional techniques to collect information. These techniques can support effective information collection and structuring while reducing the amount of time required for analysis.

1. Joint Application Design

Joint Application Design (JAD) started in the late 1970s at IBM, and since then the practice of JAD has spread throughout many companies and industries. The main idea behind JAD is to bring together the key users, managers, and systems analysts involved in the analysis of a current system. The primary purpose of using JAD is to collect systems requirements simultaneously from the key people involved with the system. As with a group interview, having all the key people together in one place at one time allows analysts to see where there are areas of agreement and where there are conflicts. Meeting with all of these important people for over a week of intense sessions allows you the opportunity to resolve conflicts, or at least to understand why a conflict may not be simple to resolve.

JAD sessions are usually conducted at a location other than the place where the people involved normally work. The idea behind such a practice is to keep participants away from as many distractions as possible so that they can concentrate on systems analysis. A JAD may last anywhere from four hours to an entire week and may consist of several sessions. A JAD employs thousands of dollars of corporate resources, the most expensive of which is the time of the people involved. Other expenses include the costs associated with flying people to a remote site and putting them up in hotels and feeding them for several days. The typical participants in a JAD are listed below:

- ***JAD session leader*** – The **JAD session leader** organizes and runs the JAD. The JAD leader sets the agenda and sees that it is met; he or she remains neutral on issues and does not contribute ideas or opinions, but rather concentrates on keeping the group on the agenda, resolving conflicts and disagreements, and soliciting all ideas.
- ***Users*** – The key users of the system under consideration are vital participants in a JAD. They are the only ones who have a clear understanding of what it means to use the system on a daily basis.
- ***Managers*** – Managers of the work groups who use the system in question provide insight into new organizational directions, motivations for and organizational impacts of systems, and support for requirements determined during the JAD.
- ***Sponsor*** – JAD must be sponsored by someone at a relatively high level in the company. If the sponsor attends any sessions, it is usually only at the very beginning or the end.
- ***Systems analysts*** – Members of the systems analysis team attend the JAD, although their actual participation may be limited. Analysts are there to learn from users and managers, not to run or dominate the process.
- ***Scribe*** – The **scribe** takes notes during the JAD sessions usually using laptop, word processor. Notes and diagrams may be entered directly into a CASE tool.
- ***IS staff*** – Besides systems analysts, other information systems (IS) staff may attend to learn from the discussion and possibly contribute their ideas on the technical feasibility of proposed ideas or the technical limitations of current systems.

JAD sessions are usually held in special-purpose rooms where participants sit around tables. These rooms are typically equipped with whiteboards. Other audiovisual tools may be used, flip charts, and computer-generated displays. Computers may be used to create and display form or report designs, diagram existing or replacement systems, or create prototypes.

When a JAD is completed, the end result is a set of documents that detail the workings of the current system related to the study of a replacement system.

2. Prototyping

Prototyping is an iterative process involving analysts and users whereby a rudimentary version of an information system is built and rebuilt according to user feedback. Through such an iterative process, the chances are good that you will be able to better capture a system's requirements. In order to gather an initial basic set of requirements, you will still have to interview users and collect documentation.

Prototyping can replace the systems development life cycle or augment it. What we are interested in here is how prototyping can augment the requirements determination process.

Prototyping is most useful for requirements determination when

- user requirements are not clear or well understood
- few users are involved with the system
- possible designs are complex and require concrete form to fully evaluate
- communication problems have existed in the past between users and analysts
- tools are readily available to rapidly build prototypes

Prototyping also has some drawbacks as a tool for requirements determination. These include the following:

- Prototypes have a tendency to avoid creating formal documentation of system requirements, which can then make the system more difficult to develop into a fully working system
- Prototypes can become very individual to the initial user and difficult to diffuse or adapt to other potential users
- Prototypes are often built as stand-alone systems, thus ignoring issues of sharing data and interactions with other existing systems, as well as issues with scaling up applications
- Checks in the SDLC are bypassed so that some more subtle, but still important, system requirements might be forgotten

There are two types of prototyping approaches: *evolutionary prototyping* and *throwaway prototyping*.

a) **Evolutionary Prototyping:** In evolutionary prototyping, you begin by modelling parts of the target system and, if the prototyping process is successful, you evolve the rest of the system from those parts. A life-cycle model of evolutionary prototyping illustrates the iterative nature of the process and the tendency to refine the prototype until it is ready to release. One key aspect of this approach is that the prototype becomes the actual production system. Because of this, you often start with those parts of the system that are most difficult and uncertain.

b) **Throwaway Prototyping:** Unlike evolutionary prototyping, throwaway prototyping does not preserve the prototype that has been developed. With throwaway prototyping, there is never any intention to convert the prototype into a working system. Instead, the prototype is developed quickly to demonstrate some aspect of a system design that is unclear or to help users decide among different features. Once the uncertainty the prototype was created to address has been reduced, the prototype can be discarded, and the principles learned from its creation and testing can then become part of the requirements determination.

Radical Methods for Determining Requirements

Traditional and contemporary methods for determining system requirements involve automating existing business processes. In contrast, radical methods are used for determining requirements for new ways to perform current tasks. These new ways may be radically different from how things are done now, but the payoffs may be enormous: Fewer people may be needed to do the same work, relationships with customers may improve dramatically, and processes may become much more efficient and effective, all of which can result in increased profits. The overall process by which current methods are replaced with radically new methods is generally referred to as **business process reengineering (BPR)**. BPR is the search for, and implementation of, radical change in business processes to achieve breakthrough improvements in products and services.

A first step in any BPR effort relates to understanding what processes to change. To do this, you must first understand which processes represent the key business processes for the organization. **Key business processes** are the structured set of measurable activities designed to produce a specific output for a particular customer or market.

Structuring System Process Requirements

Introduction

The information gathered as part of requirements determination is represented coherently using **data flow diagrams (DFD**. Data flow diagrams enable you to model how data flow through an information system, the relationships among the data flows, and how data come to be stored at specific locations. Data flow diagrams also show the processes that change or transform data. Because data flow diagrams concentrate on the movement of data between processes, these diagrams are called *process models*. As its name indicates, a data flow diagram is a graphical tool that allows analysts to depict the flow of data in an information system.

Process Modeling

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)**. DFD is the traditional process modeling technique of structured analysis and design and one of the techniques most frequently used today for process modeling.

Modeling a System's Process for structured Analysis:

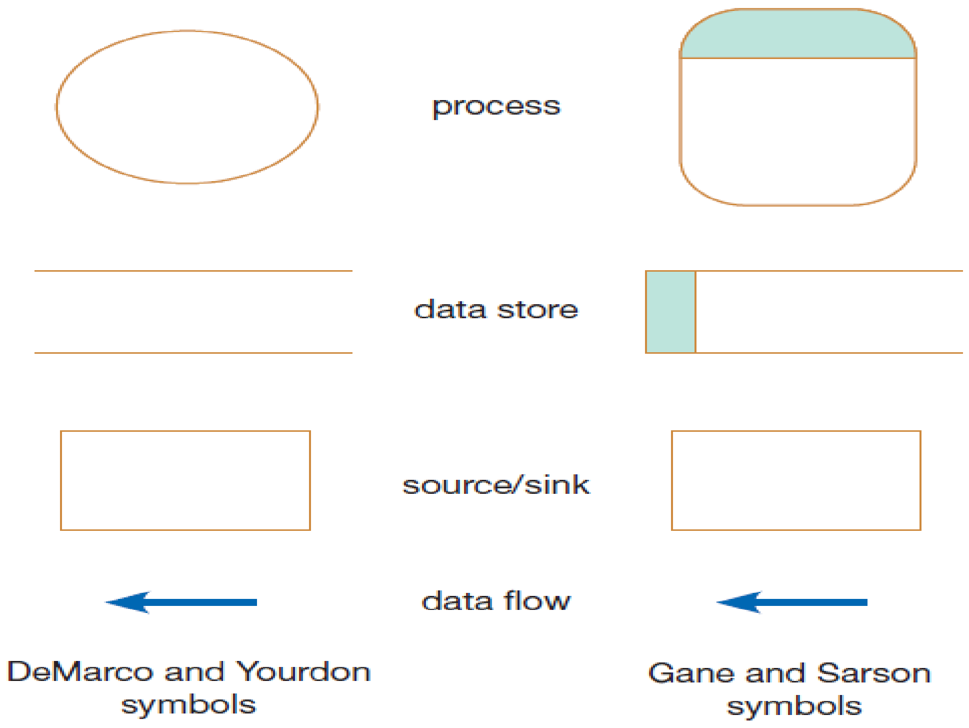
The analysis team enters the requirements structuring phase with an abundance of information gathered during the requirements determination phase. During requirements structuring, you and the other team members must organize the information into a meaningful representation of the information system that currently exists and of the requirements desired in a replacement system. For modelling processes for structured analysis, a commonly used technique is drawing data flow diagrams.

Deliverables and Outcomes:

In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated DFDs. First, a **context diagram** shows the scope of the system, indicating which elements are inside and which are outside the system. Second, DFDs of the system specify which processes move and transform data, accepting inputs and producing outputs. These diagrams are developed with sufficient detail to understand the current system and to eventually determine how to convert the current system into its replacement. This logical progression of deliverables allows you to understand the existing system. You can then abstract this system into its essential elements to show how the new system should meet the information-processing requirements identified during requirements determination.

Developing DFDs

DFDs are versatile diagramming tools. With only four symbols, you can use DFDs to represent both physical and logical information systems. 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.
The highest-level view of a system is called **context diagram**. Context diagram is an overview of an organizational system that shows the system boundaries, external entities that interact with the system and the major information flows between the entities and the system. A context DFD is used to document the scope of the project of development.

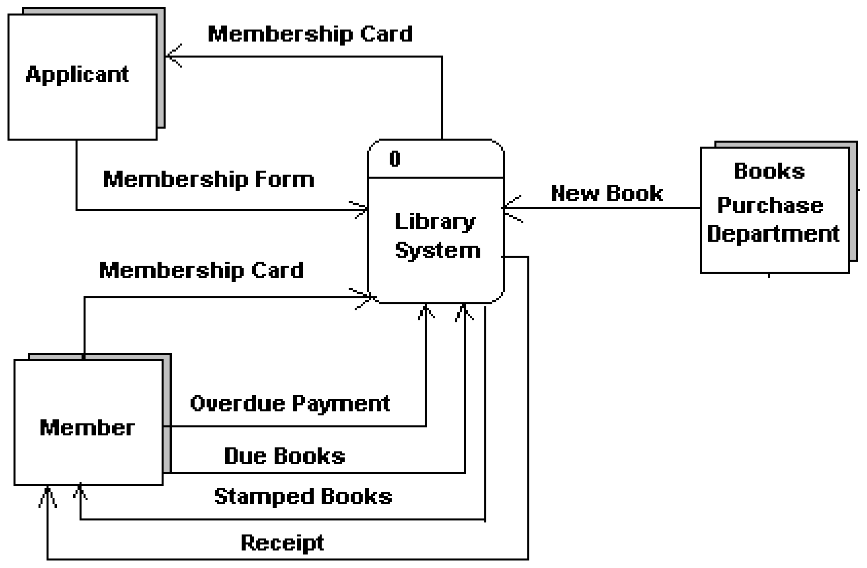


Fig: Context Diagram

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 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 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.

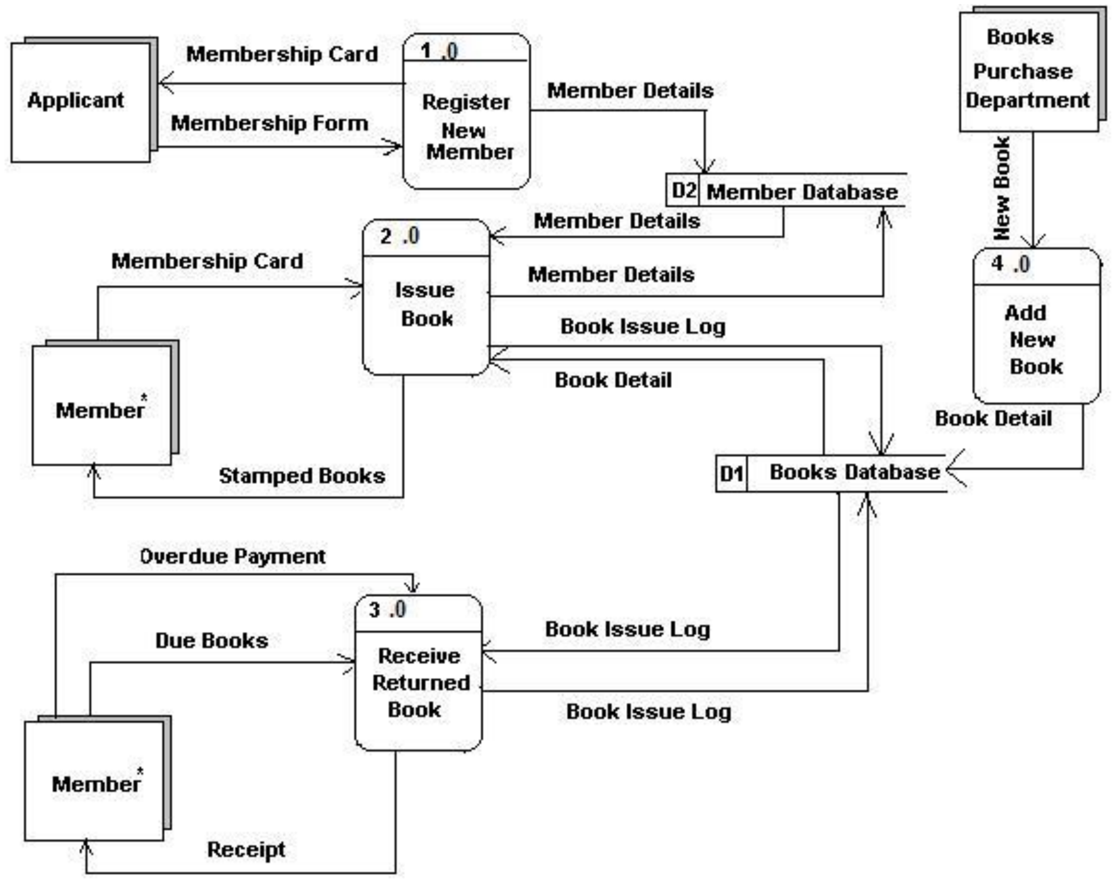


Fig: Level 0 DFD

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 sub-processes 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 sub-processes of process 1.0 to 1.1, 1.2, and so on. Similarly, sub-processes 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.

Data Flow Diagramming Rules

You must follow a set of rules when drawing DFDs. In addition to the rules given in the table below, there are two DFD guidelines that often apply.

1. *The inputs to a process are different from the outputs of that process.* Processes transform inputs to outputs.
2. *Every process on a DFD has unique names.* To keep a DFD uncluttered, however, you may repeat data stores and sources/sinks. When two arrows have the same data flow name, you must be careful that these flows are exactly the same.

Component	**Rule**
Process	• No process can have only outputs. • No process can have only inputs. • A process has a verb phrase label.
Data Store	• Data cannot move directly from one data store to another data store. Data must be moved by a process. • Data cannot move directly from an outside source to a data store. Data must be moved by a process that receives data from the source and places the data into the data store. • Data cannot move directly to an outside sink from a data store. Data must be moved by a process. • A data store has a noun phrase label.
Source/Sink	• Data cannot move directly from a source to a sink. It must be moved by a process. • A source/sink has a noun phrase label.
Data Flow	• A data flow has only one direction of flow between symbols. • A fork in a data flow means that exactly the same data goes from a common location to two or more different processes, data stores, or sources/sinks. • A join in a data flow means that exactly the same data come from any of two or more different processes, data stores, or sources/sinks to a common location. • A data flow cannot go directly back to the same process it leaves. There must be at least one other process that handles the data flow, produces some other data flow, and returns the original data flow to the beginning process. • A data flow to a data store means update. • A data flow from a data store means retrieve or use. • A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Balancing DFDs

When you decompose a DFD from one level to the next, there is a conservation principle at work. You must conserve inputs and outputs to a process at the next level of decomposition. In other words, Process 1.0, which appears in a level-0 diagram, must have the same inputs and outputs when decomposed into a level-1 diagram. This conservation of inputs and outputs is called **balancing**.

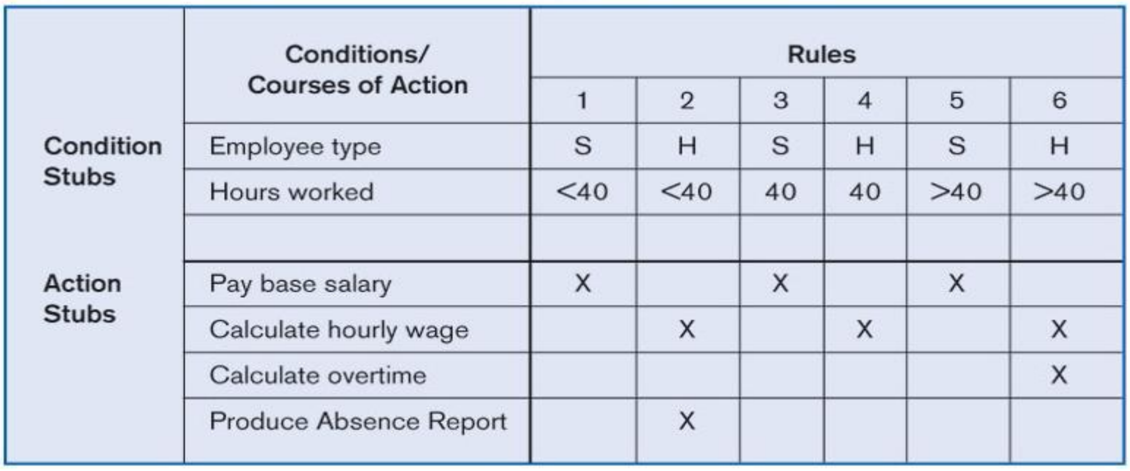
Modeling Logic with Decision Table

Many processes are governed by complex combinations of conditions that are not easily expressed. This commonly happens in business policies. In this case, we use **decision table**. A **decision table** is a diagram of process logic where the logic is reasonably complicated.

Decision table is a tabular form of presentation that specifies a set of conditions and their corresponding actions. Decision tables are very useful for specifying complex policies and decision-making rules. Three components of the decision table are:

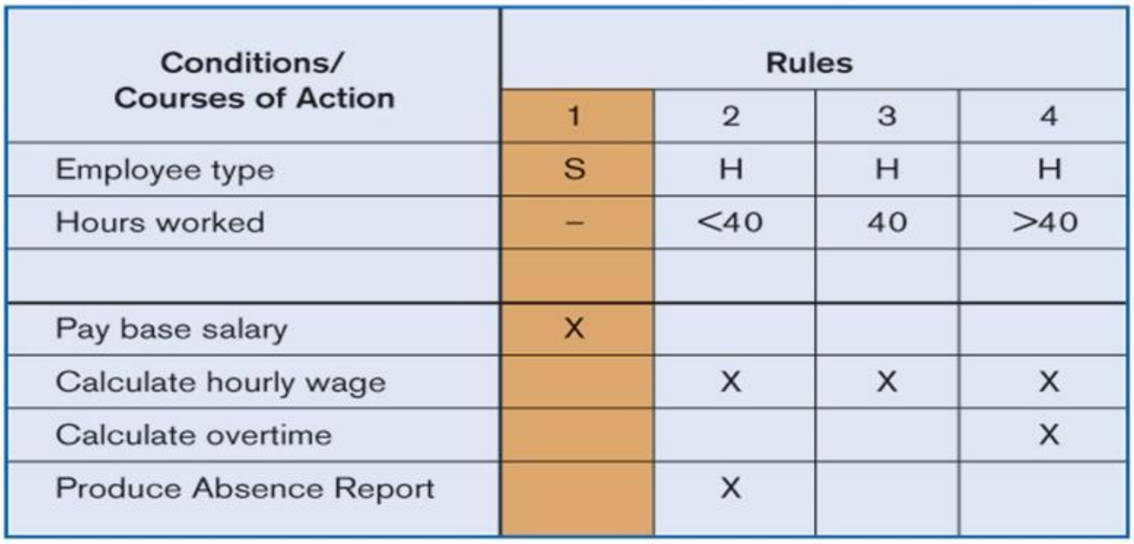
- **Condition stubs** (the upper rows) describes the conditions or factors that will affect the decision policy.
- **Action stubs** (the lower rows) describe, in the form of statements, the possible policy actions or decisions.
- **Rules** (the columns) describe which actions are to be taken under a specific combination of conditions. Each combination of conditions defines a rule that results in an action, denoted by an **X**.

For example, the figure below models the logic of a generic payroll system. The condition stubs contain tow conditions employee type (salarized and hourly) and hours worked (less than 40, exactly 40, and more than 40). The action stub contains four courses of action (pay basic salary, calculate hourly wage, calculate overtime, and produce absence report) that result from combining values of the condition stubs. And there 6 rules (each combination of conditions) that result in actions.



	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

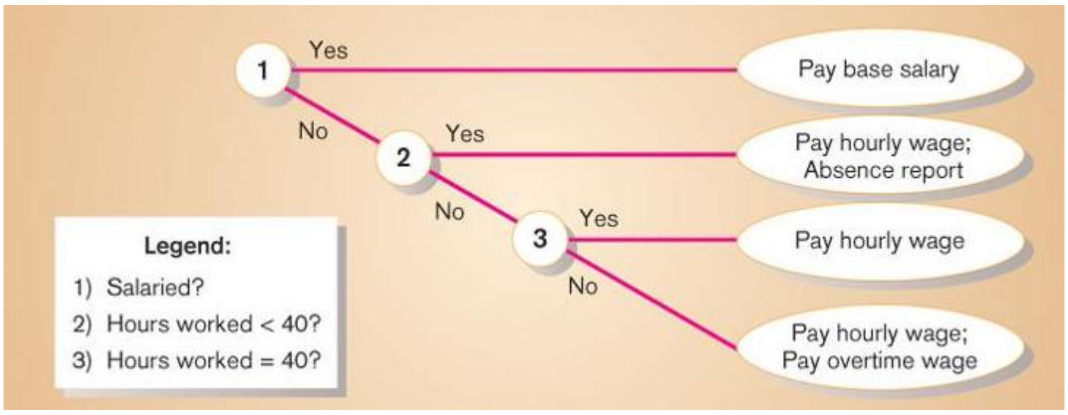
In a decision table, an **indifferent condition** is a condition whose value does not affect which actions are taken for two or more rules. In the figure above, the number of hours worked does not affect the outcome for Rules 1, 3, and 5. For these rules, hours worked is an indifferent condition. Hence, we can reduce the number of rules by condensing Rules 1, 3, and 5 into one rule as shown in the figure below.



Conditions/ Courses of Action	Rules			
	1	2	3	4
Employee type	S	H	H	H
Hours worked	–	<40	40	>40
Pay base salary	X			
Calculate hourly wage		X	X	X
Calculate overtime				X
Produce Absence Report		X		

Modeling Logic with Decision Tree

Decision tree is a graphical representation of a decision situation in which decision situation points (nodes) are connected together by arcs (one for each alternative on a decision) and terminate in ovals (the action that is the result of all of the decisions made on the path leading to that oval). The figure below shows the decision tree for the payroll system discussed before.



Modeling Logic with Pseudocode

We can also model logic of processes using pseudocode. Pseudocode is an informal high-level description of a computer program. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. It has no syntax like any of the programming language and thus can’t be compiled or interpreted by the computer. Pseudocode resembles skeleton of a computer program.
Pseudocode typically omits details that are essential for machine understanding, such as variable declarations, system-specific code and some subroutines. The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation.
The purpose of using pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description.
Pseudocode generally does not actually obey the syntax rules of any particular language; there is no systematic standard form. Some writers borrow style and syntax for control structures from some conventional programming language. Variable declarations are typically omitted. Function calls and blocks of code, such as code contained within a loop, are often replaced by a one-line natural language sentence. Depending on the writer, pseudocode may therefore vary widely in style, from a near-exact imitation of a real programming language at one extreme, to a description approaching formatted prose at the other. Below is an example of C style pseudocode.

```
void function fizzbuzz
{
    for (i = 1; i <= 100; i++)
    {
        set print_number to true;
        If i is divisible by 3
        {
            print "Fizz";
            set print_number to false;
        }
        If i is divisible by 5
        {
            print "Buzz";
            set print_number to false;
        }
        If print_number
            print i;
        print a newline;
    }
}
```

Structuring System Data Requirements

Introduction

System data requirements are structured using data models. Data modelling shows *definition, structure,* and *relationships* within the data. The most common format used for data modeling in structured development is *entity-relationship (E-R) diagramming.* Data models that use E-R diagram notations explain the characteristics and structure of data independent of how the data may be stored in computer memory.

A data model is usually developed iteratively, either from scratch or from a purchased data model for the industry or business area to be supported. Information system (IS) planners use this preliminary data model to develop an enterprise-wide data model with very broad categories of data and little detail. Next, during the definition of a project, a specific data model is built to help explain the scope of a particular systems analysis and design effort. During requirements structuring, a data model represents conceptual data requirements for a particular system. Then, after system inputs and outputs are fully described during logical design, the data model is refined before it is translated into a logical format (typically relational model) from which database definition and physical database design are done.

A data model represents certain types of business rules that govern the properties of data. Business rules are important statements of business policies that ideally will be enforced through the database and database management system ultimately used for the application you are designing.

Conceptual Data Modeling

A **conceptual data model** is a detailed model that captures the overall structure of organizational data that is independent of any database management system or other implementation considerations. A **conceptual data model** is a representation of organizational data. The purpose of a conceptual data model is to show as many rules about the meaning and interrelationships among data as are possible.

Conceptual data modeling is typically done in parallel with other requirements analysis and structuring steps during systems analysis. On larger systems development teams, a subset of the project team concentrates on data modeling while other team members focus attention on process or logic modeling.

It is essential that the process, logic, and data model descriptions of a system are consistent and complete because each describes different, but complementary, views of the same information system. For example, the names of data stores on the primitive DFD often correspond to the names of data entities in E-R diagrams, and the data elements associated with data flows on DFDs must be accounted for by attributes of entities and relationships in E-R diagrams.

The Conceptual Data Modeling Process

A new conceptual data model is built (or a standard one is purchased) that includes all of the data requirements for the new system. You discovered these requirements from the fact-finding methods employed during requirements determination.

Conceptual data modeling is one kind of data modeling and database design carried out throughout the systems development process. The planning phase of the SDLC addresses issues of system scope, general requirements, and content independent of technical implementation. E-R diagramming is suited for this phase because this diagram can be translated into technical architecture like relational database. A data model evolves from the early stages of planning through the analysis phase as it becomes more specific and is validated by more detailed analyses of system needs.

In the design phase, the final data model developed in analysis is matched with designs for systems inputs and outputs and is translated into a format from which physical data storage decisions can be made. After specific data storage architectures are selected, then files and databases are defined as the system is coded during implementation.

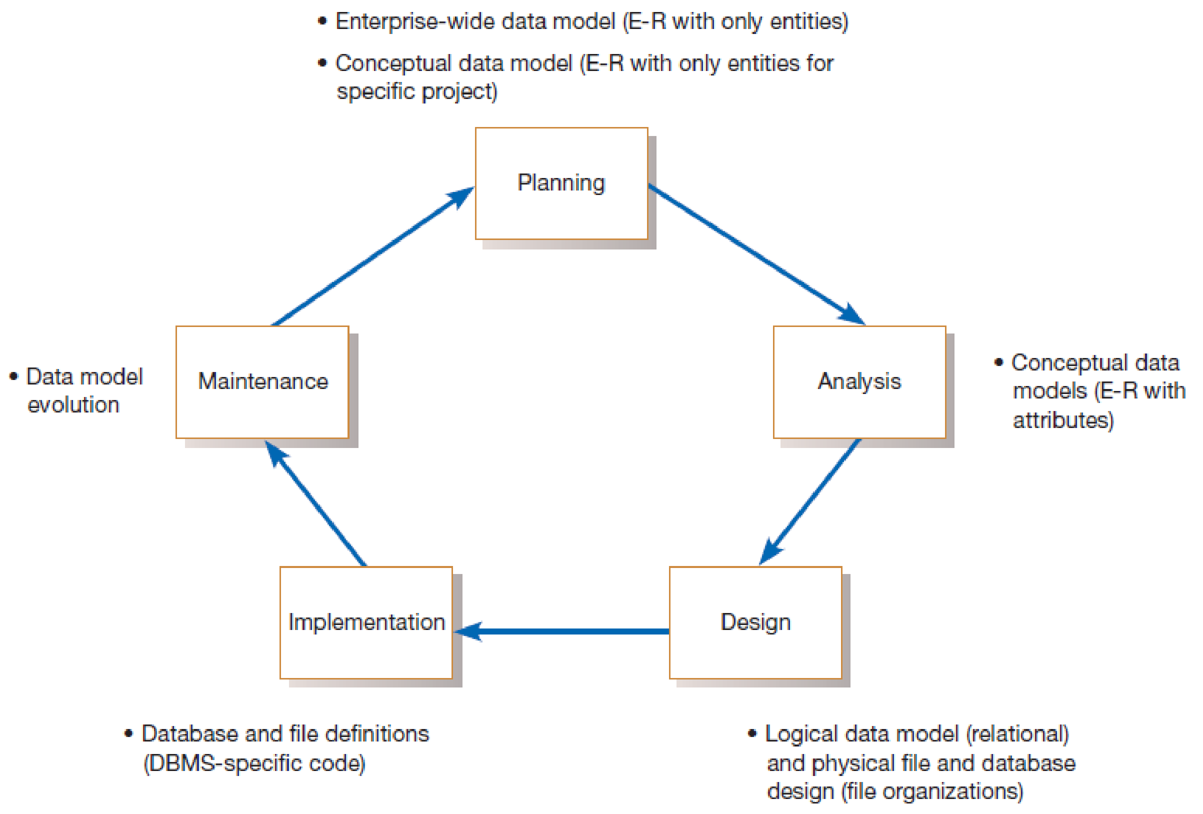

• Enterprise-wide data model (E-R with only entities)
• Conceptual data model (E-R with only entities for specific project)
Planning
Analysis
• Conceptual data models (E-R with attributes)
Design
• Logical data model (relational) and physical file and database design (file organizations)
Implementation
• Database and file definitions (DBMS-specific code)
Maintenance
• Data model evolution

Fig: Relationship between data modelling and SDLC

Deliverables and Outcomes

Most organizations today do conceptual data modeling using E-R modeling, which uses a special notation to represent as much meaning about data as possible. The primary deliverable from the conceptual data modeling step within the analysis phase is an E-R diagram. The other deliverable from conceptual data modeling is a full set of entries about data objects that will be stored in the project dictionary, repository, or data modeling software.

Gathering Information for Conceptual Data Modeling

Requirements determination methods must include questions and investigations that take a data, not only a process and logic, focus. You must ask specific questions in order to gain the perspective on data that you need to develop or tailor a purchased data model. Some key questions you should ask system users and business managers so that you can develop an accurate and complete data model tailored to the particular situation are:

- *What are the subjects/objects of the business?* What types of people, places, things, materials, events, etc. are used or interact in this business, about which data must be maintained? How many instances of each object might exist? – **data entities and their descriptions**
- *What unique characteristic (or characteristics) distinguishes each object from other objects of the same type?* Might this distinguishing feature change over time or is it permanent? Might this characteristic of an object be missing even though we know the object exists? – **primary key**

- *What characteristics describe each object?* On what basis are objects referenced, selected, qualified, sorted, and categorized? What must we know about each object in order to run the business? – **attributes and secondary keys**
- *How do you use these data?* That is, are you the source of the data for the organization, do you refer to the data, do you modify it, and do you destroy it? Who is not permitted to use these data? Who is responsible for establishing legitimate values for these data? – **security controls and understanding who really knows the meaning of data**
- *Over what period of time are you interested in these data?* Do you need historical trends, current "snapshot" values, and/or estimates or projections? If a characteristic of an object changes over time, must you know the obsolete values? – **cardinality and time dimensions of data**
- *Are all instances of each object the same?* That is, are there special kinds of each object that are described or handled differently by the organization? Are some objects summaries or combinations of more detailed objects? – **supertypes, subtypes, and aggregations**
- *What events occur that imply associations among various objects?* What natural activities or transactions of the business involve handling data about several objects of the same or a different type? – **relationships and their cardinality and degree**
- *Is each activity or event always handled the same way or are there special circumstances?* Can an event occur with only some of the associated objects, or must all objects be involved? Can the associations between objects change over time (for example, employees change departments)? Are values for data characteristics limited in any way? – **integrity rules, minimum and maximum cardinality, time dimensions of data**

You can also gather the information you need for data modeling by reviewing specific business documents – computer displays, reports, and business forms – handled within the system.

Introduction to ER Modeling

The basic E-R modeling notation uses three main constructs: **data entities**, **relationship**, and their associated **attributes**. An **entity-relationship data model (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.

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**) is a collection of entities that share common properties or characteristics. An **entity 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:

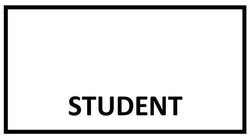
- An entity type name is a *singular noun* like CUSTOMER, STUDENT, or AUTOMOBILE.
- An entity type name should be *descriptive and specific to the organization* like PURCHASE ORDER for orders placed with suppliers to distinguish it from CUSTOMER ORDER for orders placed by customers.

- 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:

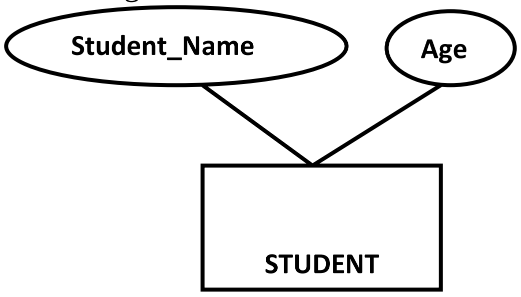
- An attribute is a *noun* like Customer_ID, Age, or Skill.
- An attribute name should be *unique*. No two attributes of the same entity type may have the same name, and it is desirable, for clarity purposes, that no two attributes across all entity types have the same name.
- To make an attribute unique 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.

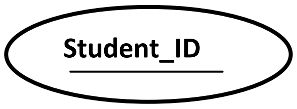


Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. A **candidate key** is an attribute or combination of attributes that uniquely identifies each instance of an entity type. For example, a candidate key for a STUDENT entity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

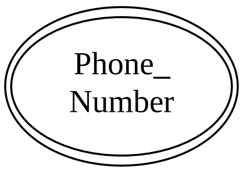
Some entity type may have more than one candidate key. In such a case, we must choose one of the candidate keys as the identifier. An **identifier** (or **primary key**) is a candidate key that has been selected to be used as the unique characteristic for an entity type. We can use the following selection 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* whose indicates classifications, locations etc.
- Consider substituting single value *surrogate keys* for large composite keys.

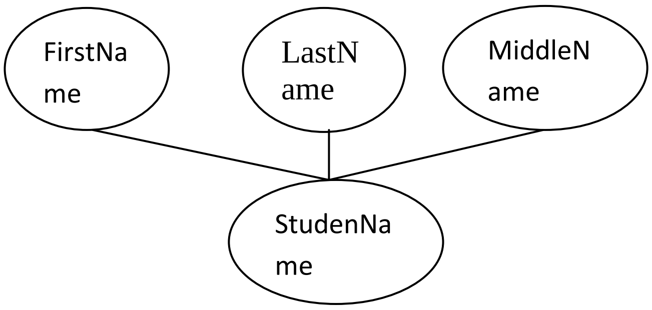
The name of the identifier is underlined on an E-R diagram. For example, the figure below shows Student_Id as an identifier for a STUDENT entity type.



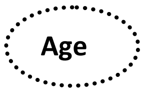
A **multivalued attribute** may take more than one value for each entity instance. For example Phone_Number attribute of STUDENT entity type. We use a double-lined ellipse to represent multivalued attribute.



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.

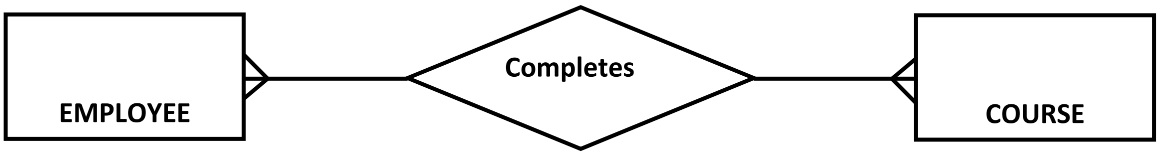


An attribute whose value can be computed from related attribute values is called **derived attribute**. For example, value of Age attribute is computed from Date_of_Birth attribute. We use dashed ellipse to denote derived attribute.



Relationships

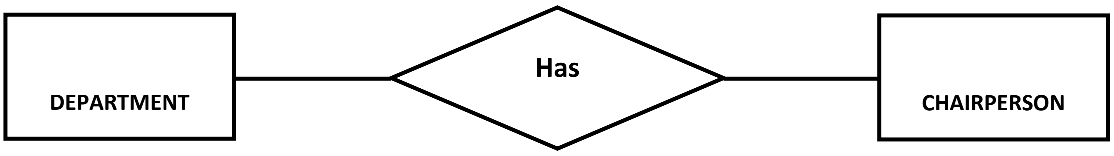
A **relationship** is an association between the instances of one or more entity types that is of interest to the organization. We use **diamond** to denote relationships. Relationships are labeled with *verb phrases*. For example, if a training department in a company is interested in tracking with 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.



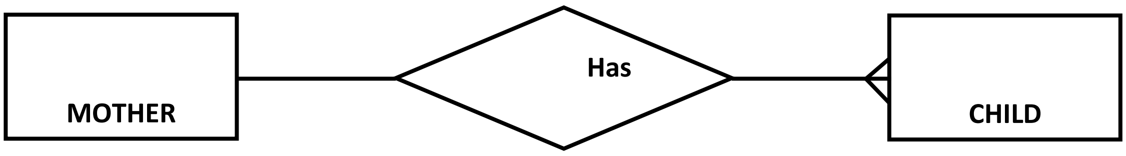
As indicated by arrows, the *cardinality* of this relationship is many-to-many since each employee may complete more than one course, and each course may be completed by more than one employee.

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*.

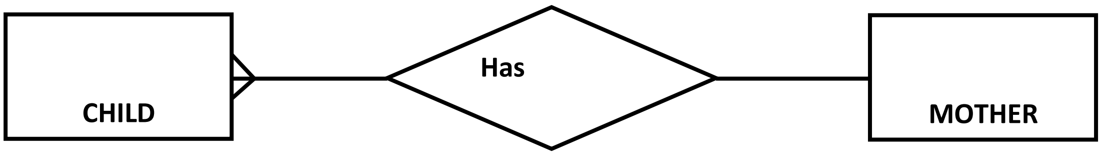
- **One-to-one:** An instance in entity type **A** is associated with at most one instance in entity type **B**, and an instance in entity type **B** is associated with at most one instance in entity type **A**. For example, cardinality between DEPARTMENT and CHAIRPERSON.



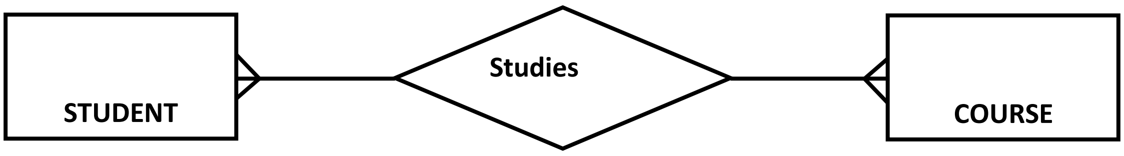
- **One-to-many:** An instance in entity type **A** is associated with any number (zero or more) of instances in entity type **B**, and an instance in entity type **B**, however, can be associated with at most one instance in entity type **A**. For example, cardinality between MOTHER and CHILD.



- **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.

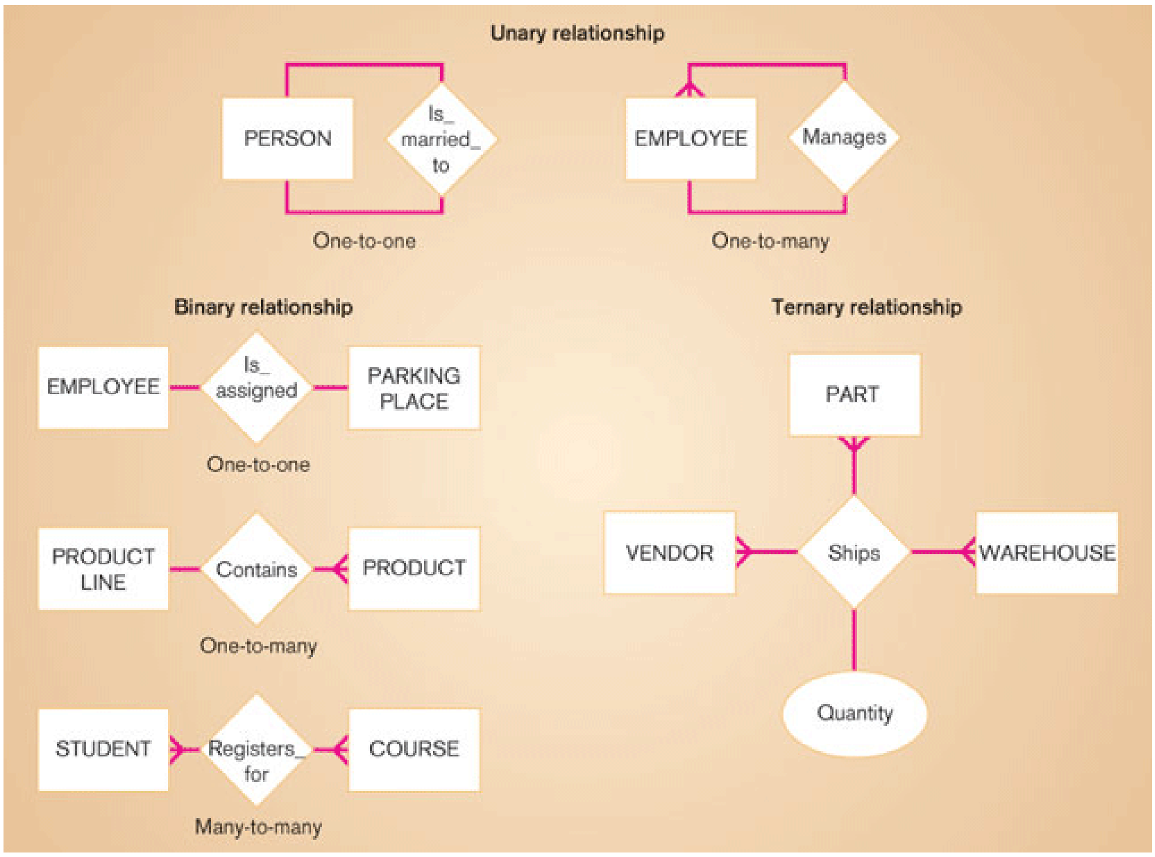
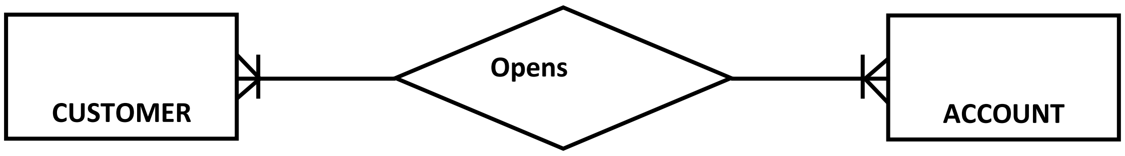


Fig: Example relationships of different degrees

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.

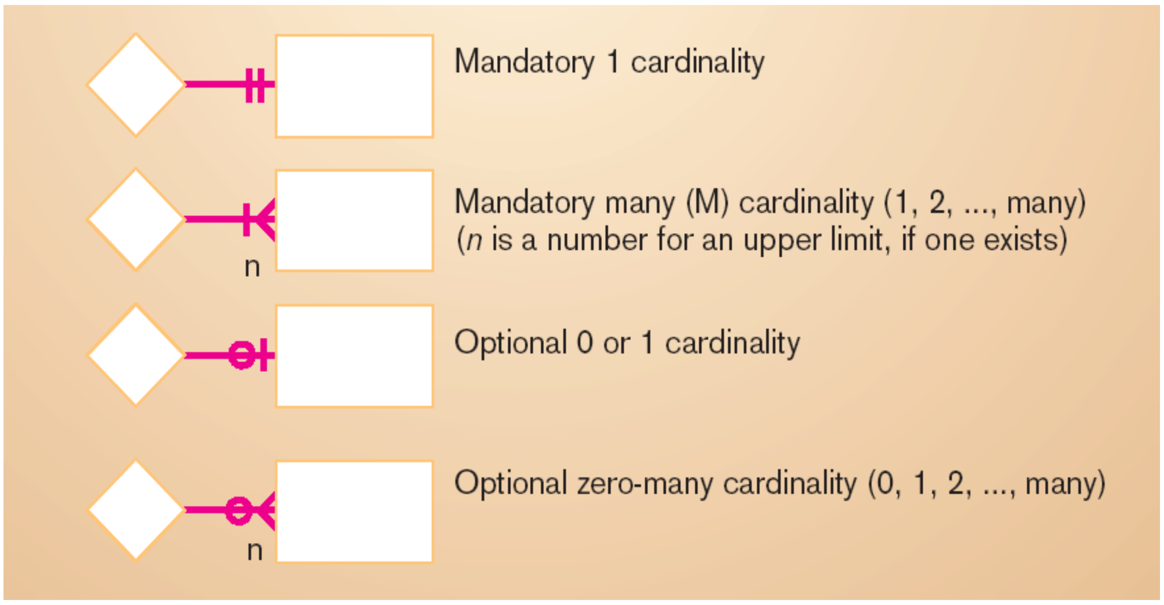

Mandatory 1 cardinality
n
Mandatory many (M) cardinality (1, 2, ..., many)
(*n* is a number for an upper limit, if one exists)
Optional 0 or 1 cardinality
n
Optional zero-many cardinality (0, 1, 2, ..., many)

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.

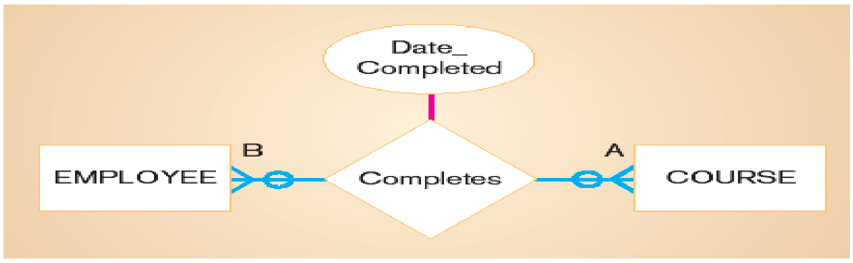
Each relationship in E-R diagram should be defined. When defining relationships, we should use the following guidelines:

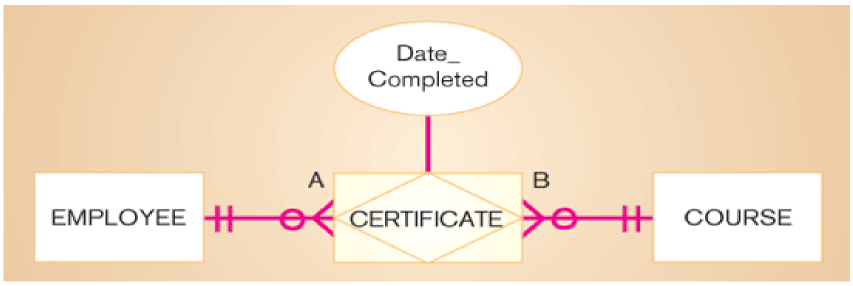
- A relationship definition should explain *what action is being taken and possibly why it is important*.
- It may be important to *give examples to clarify the action*.
- The definition should explain any *optional participation*.
- A relationship definition should *explain any restrictions on participation 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*.

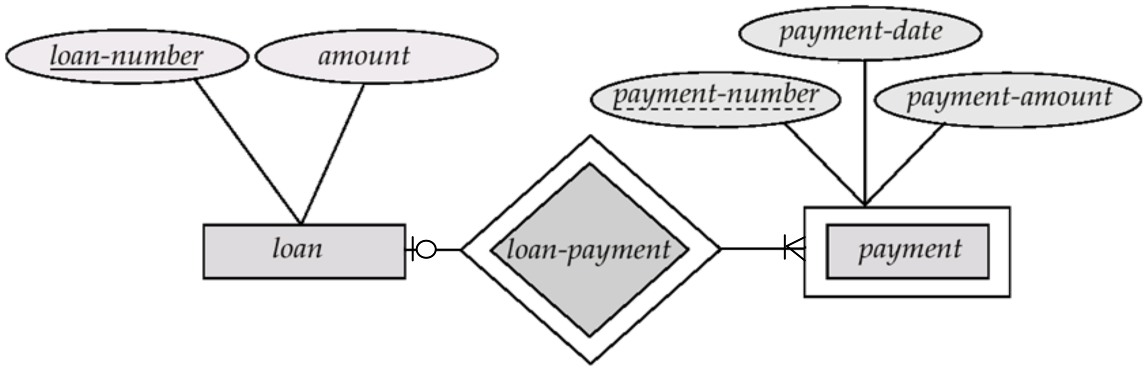
Other E-R Notations

- **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.



Prepared by: Nawaraj Paudel

Supertypes and Subtypes

A 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. These business reues are:

- **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.

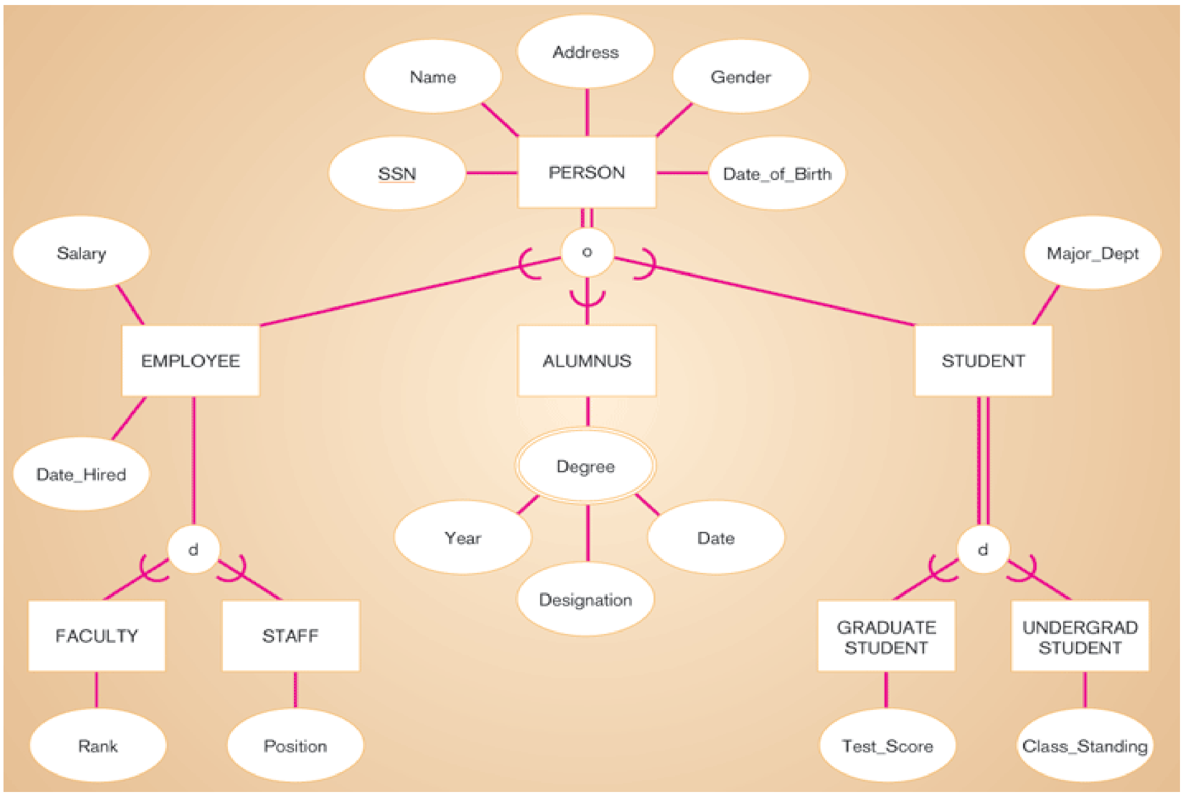


Fig: Example of supertype/subtype hierarchy

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.