Unit 3 Analysis

Analysis is the first systems development life cycle (SDLC) phase where we begin to understand, in depth, the need for system changes. Systems analysis involves a substantial amount of effort and cost, and is therefore undertaken only after management has decided that the system development project under consideration has merit and should be pursued through this phase. The analysis team should not take the analysis process for granted or attempt to speed through it. Most observers would agree that many of the errors in developed systems are directly traceable to inadequate efforts in the analysis and design phases of the life cycle. Because analysis is a large and involved process, we divided it into **two main** activities to make the overall process easier to understand:

- **Requirements determination**. This is primarily a fact finding activity.
- **Requirements structuring**. This activity creates a thorough and clear description of current business operations and new information processing services.

Determining System Requirements

Collection of information is at the core of systems analysis. **Information requirement determination** (IRD) is frequently and convincingly presented as the most critical phase of information system (IS) development, and many IS failures have been attributed to incomplete and inaccurate information requirements. System analysis must collect the information about the current and how users would like to improve their performance with new information system. Accurately understanding the user's requirement will help the system developing team deliver a proper system to the end users in limited time and limited budget.

Information on what the new system should do is gathered from as many sources as possible. Users of current system, from observing users, from the reports, forms and procedures. All this is documented and made ready for structuring. Main characteristics of a good systems analysis are listed below:

- **impertinence** (ask questions about everything that exists and also about what may exist in the future),
- **impartiality** (fine the best solution to a business problem or opportunity - consider issues raised by all parties),

- **relax constraints** (eliminate unfeasibility, assume that anything is possible, traditions may not always be reasonable),
- **attention to details** (everything must fit together so that the system works properly),
- **Reframing** (every system is different and needs a new creative approach).

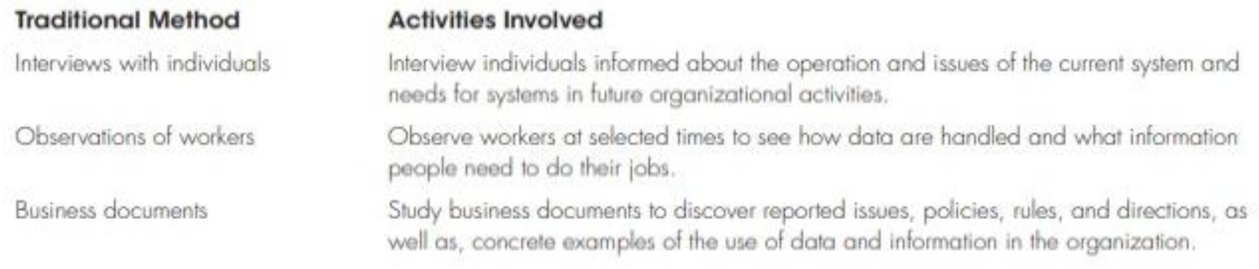
The primary **deliverables** from requirements determination are various forms of information gathered during the determination process: transcripts of interviews, notes from observations and analysis of documents, analyzed responses from questionnaires, sets of forms, reports, job descriptions, or computer-generated output such as system prototypes. We could group all this information in three groups:

- Information collected from conversations with or observations of users (interview transcripts, notes form observations etc.)
- Existing written information (business mission or strategy, forms, reports, training manuals, flow charts and documentation of existing system etc.)
- Computer-based information (result from JRP sessions, CASE repository contents)

Traditional Methods for determining requirements

Collection of information is at the core of systems analysis. At the outset, you must collect information about the information systems that are currently in use. You need to find out 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 those directly or indirectly involved in the different parts of the organization affected by the possible system changes. Another way is to gather copies of documentation relevant to current systems and business processes. In this, you learn about traditional ways to get information directly from those who have the information you need: **interviews and direct observation**. You learn about collecting documentation on the current system and organizational operation in the form of written procedures, forms, reports, and other hard copy. These traditional methods of collecting system requirements are listed in Table below:

Table: Traditional Methods of Collecting System Requirements

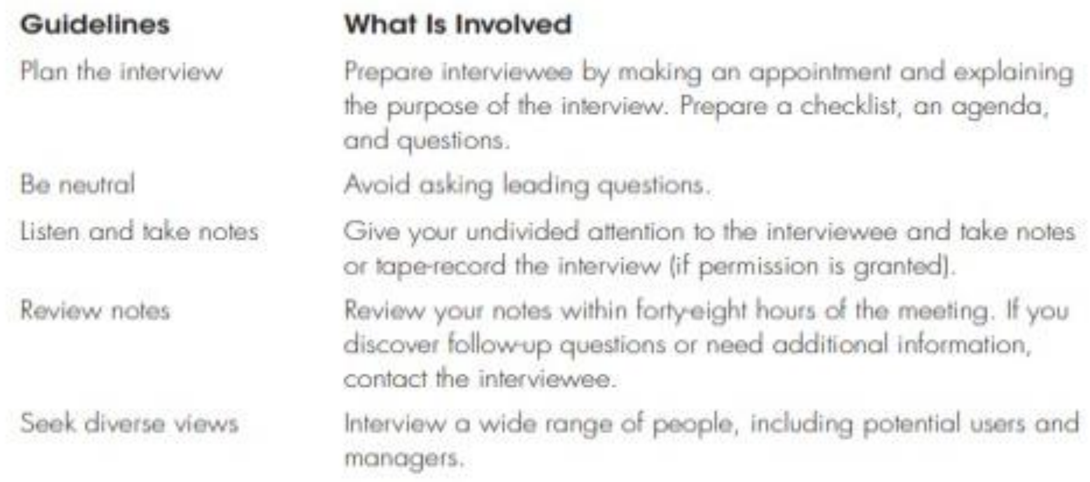


Traditional Method	Activities Involved
Interviews with individuals	Interview individuals informed about the operation and issues of the current system and needs for systems in future organizational activities.
Observations of workers	Observe workers at selected times to see how data are handled and what information people need to do their jobs.
Business documents	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.

Interviewing and Listening

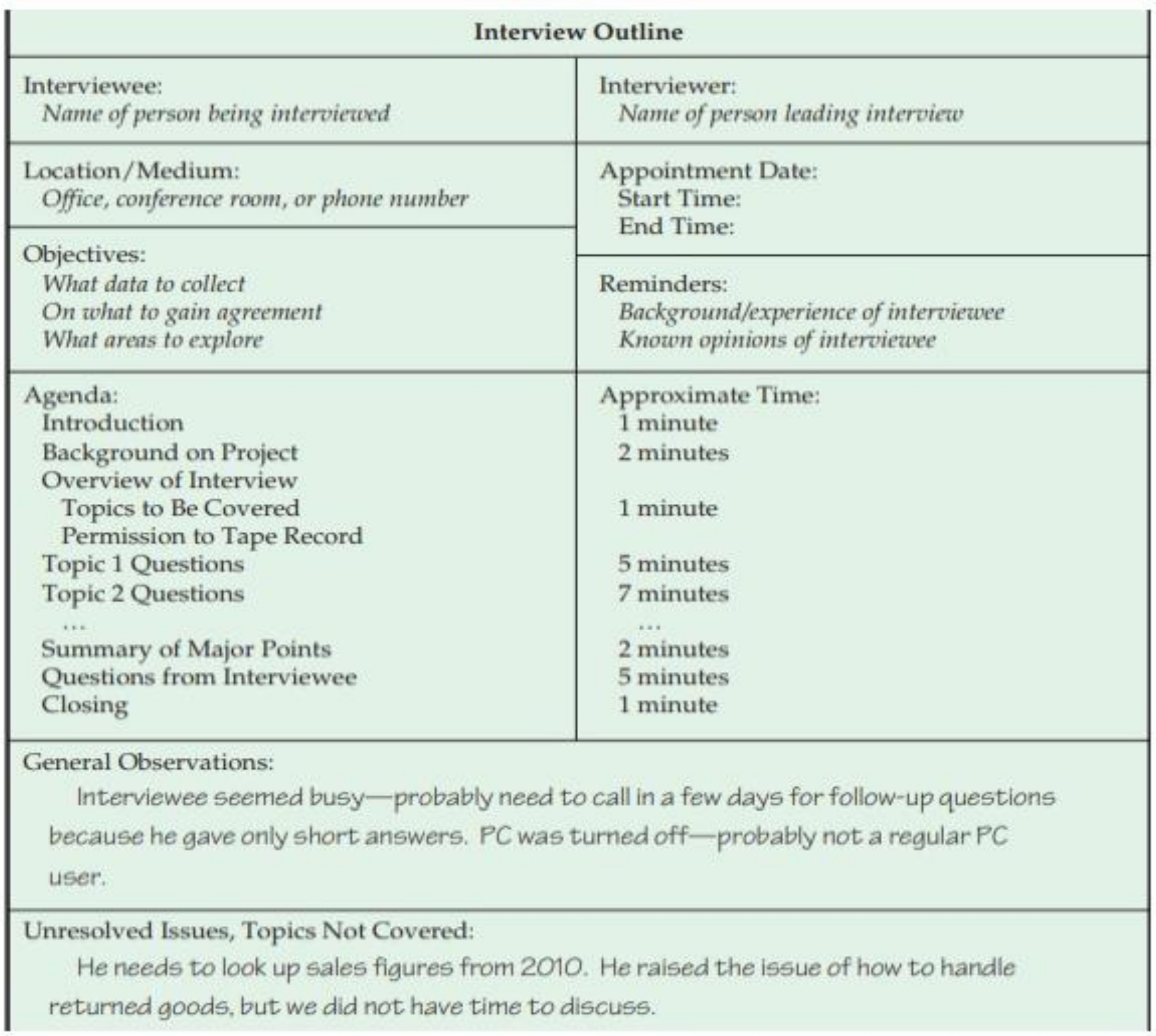
Interviewing is one of the primary ways analysts gather information about an information systems project. Early in a project, an analyst may spend a large amount of time interviewing people about their work, the information they use to do it, and the types of information processing that might supplement their work. Others are interviewed to understand organizational direction, policies, and expectations that managers have of the units they supervise. During interviewing, you gather facts, opinions, and speculation and observe body language, emotions, and other signs of what people want and how they assess current systems. Interviewing someone effectively can be done in many ways, and no one method is necessarily better than another. Some guidelines to keep in mind when you interview are summarized in Table below and are discussed next. First, prepare thoroughly before the interview. Set up an appointment at a time and for duration that is convenient for the interviewee.

Table: Guidelines for Effective Interviewing



Guidelines	What Is Involved
Plan the interview	Prepare interviewee by making an appointment and explaining the purpose of the interview. Prepare a checklist, an agenda, and questions.
Be neutral	Avoid asking leading questions.
Listen and take notes	Give your undivided attention to the interviewee and take notes or tape-record the interview (if permission is granted).
Review notes	Review your notes within forty-eight hours of the meeting. If you discover follow-up questions or need additional information, contact the interviewee.
Seek diverse views	Interview a wide range of people, including potential users and managers.

The general nature of the interview should be explained to the interviewee in advance. You may ask the interviewee to think about specific questions or issues, or to review certain documentation to prepare for the interview. Spend some time thinking about what you need to find out, and write down your questions. Do not assume that you can anticipate all possible questions. You want the interview to be natural and, to some degree, you want to direct the interview spontaneously as you discover what expertise the interviewee brings to the session. Prepare an interview guide or checklist so that you know in which sequence to ask your questions and how much time to spend in each area of the interview. The checklist might include some probing questions to ask as follow-up if you receive certain anticipated responses. You can, to some extent, integrate your interview guide with the notes you take during the interview, as depicted in a sample guide in Figure below:



Interview Outline	
Interviewee: *Name of person being interviewed*	Interviewer: *Name of person leading interview*
Location/Medium: *Office, conference room, or phone number*	Appointment Date: Start Time: End Time:
Objectives: *What data to collect* *On what to gain agreement* *What areas to explore*	Reminders: *Background/experience of interviewee* *Known opinions of interviewee*
Agenda:	Approximate Time:
Introduction	1 minute
Background on Project	2 minutes
Overview of Interview	
Topics to Be Covered	1 minute
Permission to Tape Record	
Topic 1 Questions	5 minutes
Topic 2 Questions	7 minutes
…	…
Summary of Major Points	2 minutes
Questions from Interviewee	5 minutes
Closing	1 minute

General Observations:
Interviewee seemed busy—probably need to call in a few days for follow-up questions because he gave only short answers. PC was turned off—probably not a regular PC user.

Unresolved Issues, Topics Not Covered:
He needs to look up sales figures from 2010. He raised the issue of how to handle returned goods, but we did not have time to discuss.

Figure: **A typical interview guide**

This same guide can serve as an outline for a summary of what you discover during an interview. The first page of the sample interview guide contains a general outline of the interview. Besides basic information on who is being interviewed and when, list major objectives for the interview. These objectives typically cover the most important data you need to collect, a list of issues on which you need to seek agreement (e.g., content for certain system reports), and which areas you need to explore. Also, include reminder notes to yourself on key information about the interviewee (e.g., job history, known positions taken on issues, and role with current system).

This information helps you to be personal, shows that you consider the interviewee important, and may assist in interpreting some answers. Also included is an agenda with approximate time limits for different sections of the interview. You may not follow the time limits precisely, but the schedule helps you cover all areas during the time the interviewee is available. Space is also allotted for general observations that do not fit under specific questions and for notes taken during the interview about topics skipped or issues raised that could not be resolved. On subsequent pages, list specific questions. The sample form in above figure includes space for taking notes on these questions. Because the interviewee may provide information you were not expecting, you may not follow the guide in sequence. You can, however, check off questions you have asked and write reminders to yourself to return to or skip other questions as the interview takes place.

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. Respondents must feel free to state their true opinions and perspectives and trust that their ideas will be considered. Avoid questions such as “Should the system continue to provide the ability to override the default value, even though most users now do not like the feature?” because such wording predefines a socially acceptable answer.
- Second, listen carefully to what is being said. Take careful notes or, if possible, record the interview on a tape recorder (be sure to ask permission first!). The answers may contain extremely important information for the project. Also, this may be your only chance to get information from this particular person. If you run out of time and still need more 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 key in your notes within forty-eight hours with a word processing program such as Microsoft Word. For numerical data, you can use a spreadsheet program such as Microsoft Excel. If you recorded the interview, use the recording to verify your notes. After forty-eight 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 ambiguous information. Separate facts from your opinions and interpretations. Make a list of unclear points that need clarification. Call the person you interviewed and get

answers to these new questions. Use the phone call as an opportunity to verify the accuracy of your notes. You may also want to send a written copy of your notes to the person you interviewed to check your notes for accuracy. Finally, make sure to thank the person for his or her time. You may need to talk to your respondent again. If the interviewee will be a user of your system or is involved in some other way in the system's success, you want to leave a good impression.

- 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 the interviewee know that there are many steps to the project. Many people will have to be interviewed. Choices will have to be made from among many technically possible alternatives. Let respondents know that their ideas will be carefully considered. Because of the repetitive nature of the systems development process, however, 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. Talk to several different people: potential users of the system, users of other systems that might be affected by this new system, managers and superiors, information systems staff, and others. Encourage people to think about current problems and opportunities and what new information services might better serve the organization. You want to understand all possible perspectives so that later you will have information on which to base a recommendation or design decision that everyone can accept.

Directly Observing Users

Interviewing involves getting people to recall and convey information they have about organizational processes and the information systems that support them. People, however, are not always reliable, even when they try to be and say what they think is the truth. As odd as it may sound, people often do not have a completely accurate appreciation of what they do or how they do it, especially when infrequent events, issues from the past, or issues for which people have considerable passion are involved. Because people cannot always be trusted to interpret and report their own actions reliably, you can supplement what people tell you by watching what they do in work situations. For example, one possible view of how a hypothetical manager does her job is that a manager carefully plans her activities, works long and consistently on solving problems, and controls the pace of her work. A manager might tell you that is how she spends her day. Several studies have shown, however, that a manager's day is actually punctuated by many, many interruptions.

The intent behind obtaining system records and direct observation is the same, however, and that is to obtain more firsthand and objective measures of employee interaction with information systems. In some cases, behavioral measures will more accurately reflect reality than what employees themselves believe. In other cases, the behavioral information will substantiate what employees have told you directly. Although observation and obtaining objective measures are desirable ways to collect pertinent information, such methods are not always possible in real organizational settings. Thus, these methods are not totally unbiased, just as no one data-gathering method is unbiased. For example, observation can cause people to change their normal operating behavior.

Analyzing Procedures and Other Documents

Interviewing people who use a system every day or who have an interest in a system is an effective way to gather information about current and future systems. Observing current system users is a more direct way of seeing how an existing system operates. Both interviewing and observing have limitations. Methods for determining system requirements can be enhanced by examining system and organizational documentation to discover more details about current systems and the organization they support. We discuss several important types of documents that are useful in understanding system requirements, but our discussion is not necessarily exhaustive. In addition to the few specific documents we mention, other important documents need to be located and considered, including organizational mission statements, business plans, organization charts, business policy manuals, job descriptions, internal and external correspondence, and reports from prior organizational studies.

What can the analysis of documents tell you about the requirements for a new system? In documents you can find information about:

- Problems with existing systems (e.g., missing information or redundant steps)
- Opportunities to meet new needs if only certain information or information processing were available (e.g., analysis of sales based on customer type)
- Organizational direction that can influence information system requirements (e.g., trying to link customers and suppliers more closely to the organization)
- Titles and names of key individuals who have an interest in relevant existing systems (e.g., the name of a sales manager who has led a study of buying behavior of key customers)
- Values of the organization or individuals who can help determine priorities for different capabilities desired by different users (e.g., maintaining market share even if it means lower short-term profits)

- Special information-processing circumstances that occur irregularly that may not be identified by any other requirements determination technique (e.g., special handling needed for a few large-volume customers who require use of customized customer ordering procedures)
- The reason why current systems are designed as they are, which can suggest features left out of current software that may now be feasible and desirable (e.g., data about a customer's purchase of competitors' products not available when the current system was designed; these data now available from several sources)
- Data, rules for processing data, and principles by which the organization operates that must be enforced by the information system (e.g., each customer assigned exactly one sales department staff member as primary contact if customer has any questions)

Contemporary Methods for Determining System Requirements

Even though we called interviews, questionnaires, observation, and document analysis traditional methods for determining a system's requirements, all of these methods are still used by analysts to collect important information. Today, however, additional techniques are available to collect information about the current system, the organizational area requesting the new system, and what the new system should be like. In this section, you learn about two modern information-gathering techniques for analysis: **joint application design (JAD) and prototyping**. These techniques can support effective information collection and structuring while reducing the amount of time required for analysis.

Joint application design (JAD)

JAD started in the late 1970s at IBM as a means to bring together the key users, managers, and systems analysts involved in the analysis of a current system. Since the 1970s, JAD has spread throughout many companies and industries. For example, it is quite popular in the insurance industry. The primary purpose of using JAD in the analysis phase is to collect systems requirements simultaneously from the key people involved with the system. The result is an intense and structured, but highly effective, process. Having all the key people together in one place at one time allows analysts to see the areas of agreement and the areas of conflict. Meeting with all 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 in a location away from where the people involved normally work, in order to limit distractions and help participant's better 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 following is a list of typical JAD participants:

- **JAD session leader:** The JAD leader organizes and runs the JAD. This person has been trained in group management and facilitation as well as in systems analysis. 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 clearly understand 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:** As a major undertaking, because of its expense, a JAD must be sponsored by someone at a relatively high level in the company such as a vice president or chief executive officer. If the sponsor attends any sessions, it is usually only at the 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 on a personal computer or laptop.
- **IS staff:** Besides systems analysts, other IS staff, such as programmers, database analysts, IS planners and data-center personnel, may attend the session. Their purpose is to learn from the discussion and possibly to contribute their ideas on the technical feasibility of proposed ideas or on the technical limitations of current systems.

JAD sessions are usually held in special-purpose rooms where participants sit around horseshoe-shaped tables, as in Figure below. These rooms are typically equipped with whiteboards (possibly electronic, with a printer to make copies of what is written on the board). Other audiovisual tools may be used, such as magnetic symbols that can be easily rearranged on a whiteboard, flip charts, and computer-generated displays. Flip-chart paper is typically used for keeping track of issues that cannot be resolved during the JAD, or for those issues requiring additional information that can be gathered during breaks in the proceedings. Computers may be used to create and display form or report designs or to diagram existing or replacement systems. In general, however, most JADs do not benefit much from computer support. The end result of a completed JAD is a set of documents that detail the workings of the current system and the features of a replacement system. Depending on the exact purpose of the JAD, analysts may gain detailed information on what is desired of the replacement system.

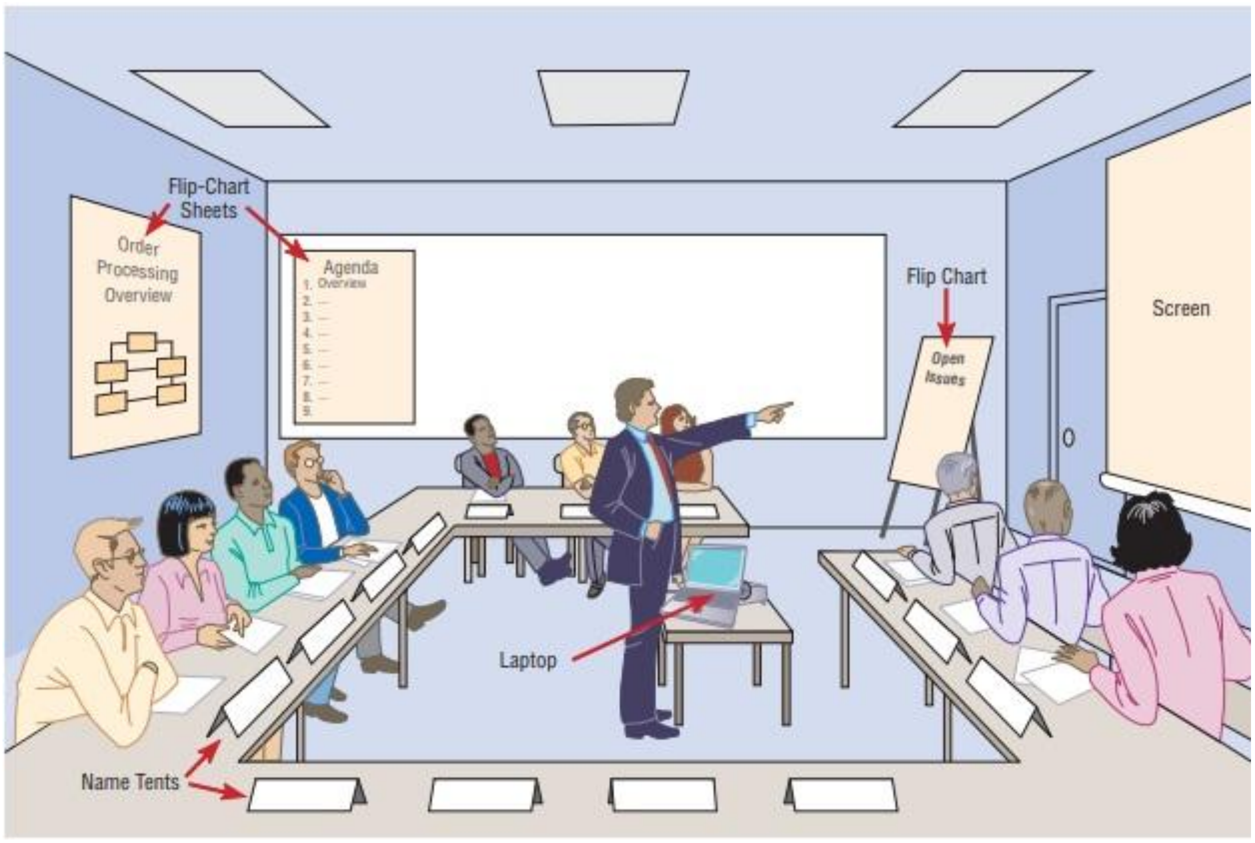


Figure: A typical room layout for a JAD session

Using Prototyping during Requirements Determination

Prototyping allows us to quickly convert basic requirements into a working, though limited, version of the desired information system. The user then views and tests the prototype. Typically, seeing verbal descriptions of requirements converted into a physical system prompts the user to modify existing requirements and generate new ones. For example, in the initial interviews, a user might have said he wanted all relevant utility billing information on a single computer display form, such as the client's name and address, the service record, and payment history. Once the same user sees how crowded and confusing such a design would be in the prototype, he might change his mind and instead ask for the information to be organized on several screens but with easy transitions from one screen to another. He might also be reminded of some important requirements (data, calculations, etc.) that had not surfaced during the initial interviews. You would then redesign the prototype to incorporate the suggested changes. Once modified, users would again view and test the prototype. Once again, you would incorporate their suggestions for change. Through such a repetitive process, the chances are good that you will be able to better capture a system's requirements. The goal with using prototyping to support requirements determination is to develop concrete specifications for the ultimate system, not to build the ultimate system.

Prototyping is most useful for requirements determination when:

- User requirements are not clear or well understood which is often the case for totally new systems or systems that support decision making.
- One or a few users and other stakeholders are involved with the system.
- Possible designs are complex and require concrete form to evaluate fully.
- Communication problems have existed in the past between users and analysts, and both parties want to be sure that system requirements are as specific as possible.
- Tools (such as form and report generators) and data are readily available to rapidly build working systems.

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

- 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 idiosyncratic 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.

- Checks in the SDLC are bypassed so that some more subtle, but still important, system requirements might be forgotten (e.g., security, some data-entry controls, or standardization of data across systems).

Radical Methods for Determining System Requirements

Whether traditional or modern, the methods for determining system requirements is apply to any requirements determination effort, regardless of its motivation. Yet, most of what you have learned has traditionally been applied to systems development projects that involve automating existing processes. Analysts use system requirements determination to understand current problems and opportunities, as well as what are needed and desired in future systems. Typically, the current way of doing things has a large impact on the new system. In some organizations, though, management is looking for new ways to perform current tasks. These 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 referred to as **business process reengineering (BPR)**

To better understand BPR, consider the following analogy. Suppose you are a successful European golfer who has tuned your game to fit the style of golf courses and weather in Europe. You have learned how to control the flight of the ball in heavy winds, roll the ball on wide-open greens, putt on large and undulating greens, and aim at a target without the aid of the landscaping common on North American courses. When you come to the United States to make your fortune on the U.S. tour, you discover that improving you're putting, driving accuracy, and sand shots will help, but the new competitive environment is simply not suited to your playing style. You need to reengineer your whole approach, learning how to aim at targets, spin and stop a ball on the green, and manage the distractions of crowds and press. If you are good enough, you may survive, but without reengineering, you will never become a winner. Just as the competitiveness of golf forces good players to adapt their games to changing conditions, the competitiveness of our global economy has driven most companies into a mode of continuously improving the quality of their products and services. Organizations realize that creatively using information technologies can significantly improve most business processes. The idea behind BPR is not just to improve each business process but, in a systems modeling sense, to reorganize the complete flow of data in major sections of an organization to eliminate unnecessary steps, combine previously separate steps, and become more responsive to future changes. Companies such as IBM, Procter &

Gamble, Wal-Mart, and Ford have had great success in actively pursuing BPR efforts. Yet, many other companies have found difficulty in applying BPR principles. Nonetheless, BPR concepts are actively applied in both corporate strategic planning and information systems planning as a way to improve business processes radically.

BPR advocates suggest that radical increases in the quality of business processes can be achieved through creatively applying information technologies. BPR advocates also suggest that radical improvement cannot be achieved by making minor changes in existing processes but rather by using a clean sheet of paper and asking, “If we were a new organization, how would we accomplish this activity?” Changing the way work is performed also changes the way information is shared and stored, which means that the results of many BPR efforts are the development of information system maintenance requests, or requests for system replacement. You likely have encountered or will encounter BPR initiatives in your own organization. A recent survey of IS executives found that they view BPR to be a top IS priority for the coming years.

Identifying Processes to Reengineer

A first step in any BPR effort is to understand what processes need to change, what are 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. The important aspect of this definition is that key processes are focused on some type of organizational outcome such as the creation of a product or the delivery of a service. Key business processes are also customer focused. In other words, key business processes would include all activities used to design, build, deliver, support, and service a particular product for a particular customer. BPR, therefore, requires you first to understand those activities that are part of the organization’s key business processes and then to alter the sequence and structure of activities to achieve radical improvements in speed, quality, and customer satisfaction. The same techniques you learned to use for system requirements determination can be applied to discovering and understanding key business processes: interviewing key individuals, observing activities, reading and studying organizational documents, and conducting JAD sessions. After identifying key business processes, the next step is to identify specific activities that can be radically improved through reengineering. Michael Hammer and James Champy, two academics who coined the term BPR, suggest systems analysts ask three questions to identify activities for radical change:

1. How important is the activity to delivering an outcome?
2. How feasible is changing the activity?

3. How dysfunctional is the activity?

The answers to these questions provide guidance for selecting which activities to change. Those activities deemed important, changeable, yet dysfunctional, are primary candidates for alteration. To identify dysfunctional activities, Hammer and Champy suggest you look for activities that involve excessive information exchanges between individuals, information that is redundantly recorded or needs to be rekeyed, excessive inventory buffers or inspections, and a lot of rework or complexity.

Disruptive Technologies

Once key business processes and activities have been identified, information technologies must be applied to improve business processes radically. Hammer and Champy suggest that organizations think "inductively" about information technology. Induction is the process of reasoning from the specific to the general, which means that managers must learn about the power of new technologies and think of innovative ways to alter the way work is done. This approach is contrary to deductive thinking, in which problems are first identified and solutions then formulated. Hammer and Champy suggest that managers especially consider disruptive technologies when applying deductive thinking. Disruptive technologies are those that enable the breaking of long-held business rules that inhibit organizations from making radical business changes. For example, Toyota is using production schedule databases and electronic data interchange (EDI) an information system that allows companies to link their computers directly to suppliers to work with its suppliers as if they and Saturn were one company. Suppliers do not wait until Saturn sends them a purchase order for more parts but simply monitor inventory levels and automatically send shipments as needed.

Structuring System Process Requirements

Here we understand the logical modeling of processes by studying examples of data flow diagrams:

Process Modeling

Process modeling involves graphically representing the processes, or actions, that capture, manipulate, store, and distribute data between a system and its environment and among components within a system. A common form of a process model is a **data-flow diagram (DFD)**. A data-flow diagram is a graphic that illustrates the movement of data between external entities and the processes and data stores within a system. Although several different tools have been developed for process modeling, we focus solely on data-flow diagrams because they are useful tools for process modeling. Data-flow diagramming is one of several structured analysis techniques used to increase software development productivity. Although not all organizations use each structured analysis technique, collectively, these techniques, like dataflow diagrams, have had a significant impact on the quality of the systems development process.

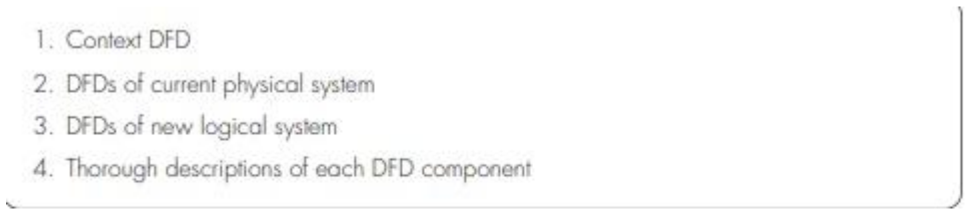
Modeling a System's Process

The analysis team begins the process of structuring requirements with an abundance of information gathered during requirements determination. As part of structuring, you and the other team members must organize the information into a meaningful representation of the information system that exists and of the requirements desired in a replacement system. In addition to modeling the processing elements of an information system and transformation of data in the system, you must also model the structure of data within the system. Analysts use both process and data models to establish the specification of an information system. With a supporting tool, such as a CASE tool, process and data models can also provide the basis for the automatic generation of an information system.

Deliverables and Outcomes

In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated data-flow diagrams. Table below lists the progression of deliverables that result from studying and documenting a system's process. First, a context data-flow diagram shows the scope of the system, indicating which elements are inside and outside the system. Second, data-flow diagrams of the current system specify which people and technologies are used in which processes to move and transform data, accepting inputs and producing outputs. The detail of these diagrams allows analysts to understand the current system and eventually to determine how to convert the current system into its replacement. Third, technology-independent, or logical, data-flow diagrams show the dataflow, structure, and functional requirements of the new system. Finally, entries for all of the objects in all diagrams are included in the project dictionary or CASE repository.

Table: Deliverables for Process Modeling


1. Context DFD
2. DFDs of current physical system
3. DFDs of new logical system
4. Thorough descriptions of each DFD component

This logical progression of deliverables helps you to understand the existing system. You can then reduce this system into its essential elements to show the way in which the new system should meet its information processing requirements, as they were identified during requirements determination. In later steps in the systems development life cycle, you and other project team members make decisions on exactly how the new system will deliver these new requirements in specific manual and automated functions. Because requirements determination and structuring are often parallel steps, data-flow diagrams evolve from the more general to the more detailed as current and replacement systems are better understood. Even though data-flow diagrams remain popular tools for process modeling and can significantly increase software development productivity, they are not used in all systems development methodologies. Some organizations, such as EDS, have developed their own type of diagrams to model processes. Some methodologies, such as rapid application development (RAD), do not model processes separately at all. Instead, RAD builds processes the work or actions that transform data so that they can be stored or distributed into the prototypes created as the core of its development life cycle. However, even if you never formally use data-flow diagrams in your professional career, they remain a part of systems development's history. DFDs illustrate important concepts about the movement of data between manual and automated steps and are a way to depict work flow in an organization. DFDs continue to benefit information systems professionals as tools for both analysis and communication. For that reason, we devote this entire chapter to DFDs.

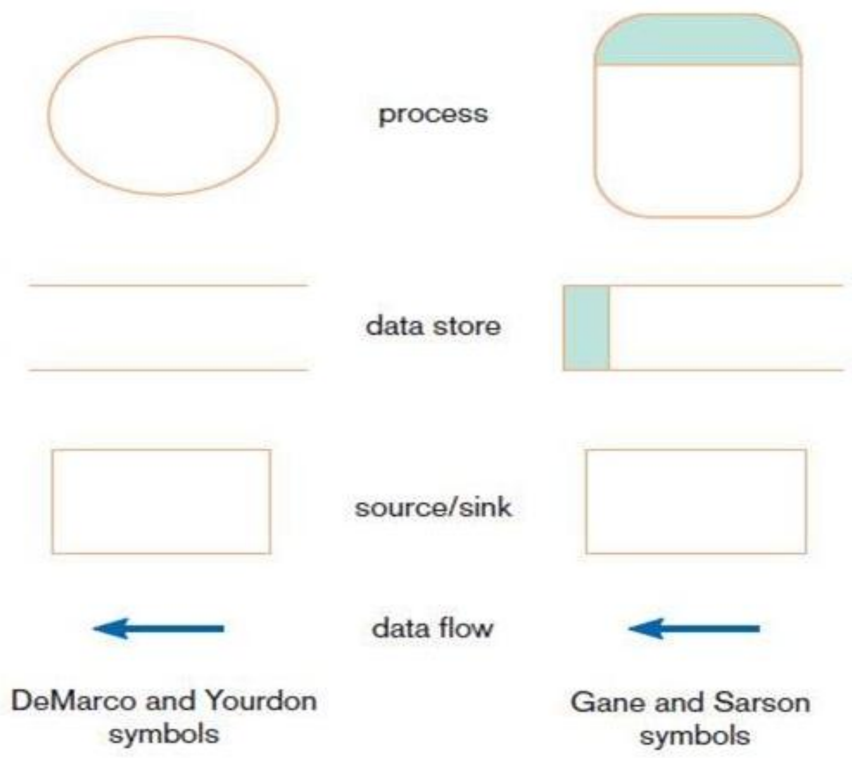
Data Flow Diagrams (DFD)

A DFD is a pictorial representation of the movement of data between external entities and the processes and data stores within a system. It is a common technique for creating process models. A DFD shows how data moves through an information system but does not show program logic or processing steps. DFDs can also be used for the visualization of data processing (Structured Design).

Data Flow Diagramming Mechanics

Four symbols are used to represent DFDs (See Figure below). Two different standard sets can be used as proposed by:

1. DeMarco and Yourdan
2. Gane and Sarson



Process: A process is the work or actions performed on data so that they are transformed, stored, or distributed. The symbol for a process is a rectangle with rounded corners. Inside the rectangle are written both the number of the process and a name, which indicates what the process does. For example, the process may generate paychecks, calculate overtime pay, or compute grade-point average.

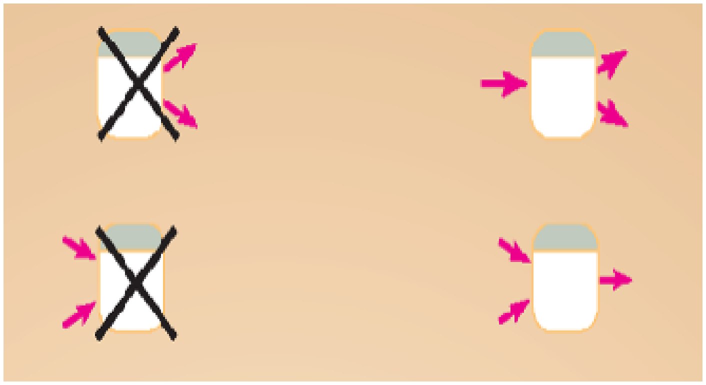
Data Store: A data store is data at rest. A data store may represent one of many different physical locations for data, including a file folder, one or more computer-based file(s), or a notebook. The symbol for a data store is a rectangle with the right vertical line missing. Its label includes the number of the data store (e.g., D1 or D2) and a meaningful label, such as student file, transcripts, or roster of classes.

Source/Sink (External Entities): Source/Sink is the origin and/or destination of the data (outside of the system). Source/sinks are sometimes referred to as external entities because they are outside the system and define the system's boundaries. Data must originate outside a system from one or more sources, and the system must produce information to one or more sinks. It is drawn as a square symbol. The name states what the external agent is. Because they are external, many characteristics are not of interest to us. A person, organization, or system that is external to the system but interacts with it.

Data Flow: A data flow is data that are in motion and moving as a unit from one place in a system to another. A data flow is depicted as an arrow. The arrow is labeled with a meaningful name for the data in motion; for example, customer order, sales receipt, or paycheck.

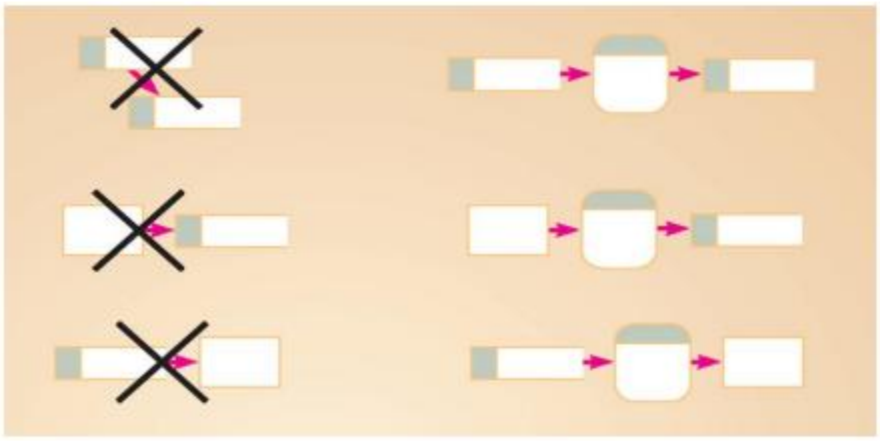
DFD Diagramming Rules:

Rules for Process:



- No process can have only outputs. It is making data from nothing (a miracle). If an object has only outputs, then it must be a source.
- No process can have only inputs (a black hole). If an object has only inputs, then it must be a sink

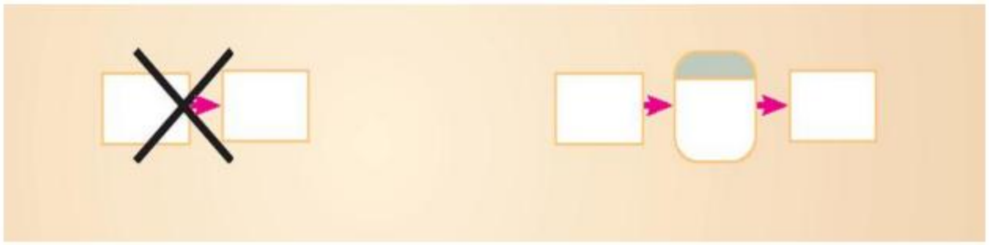
Rules for Data Store:



- 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.

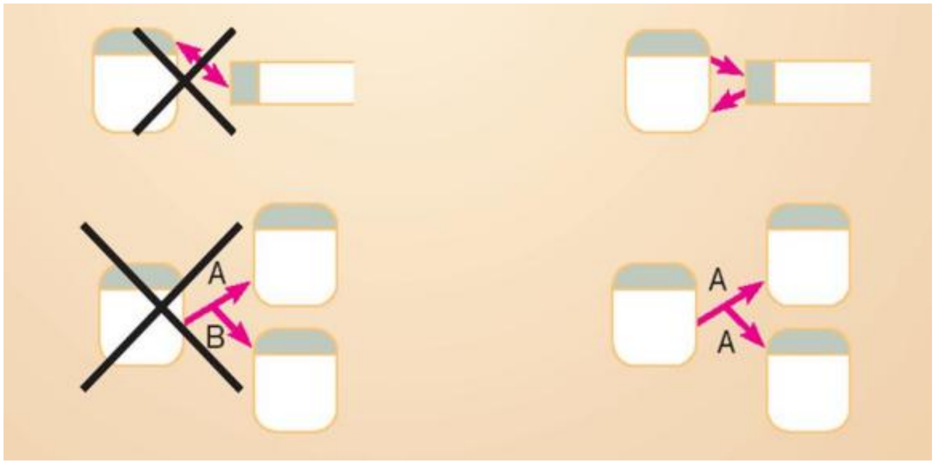
- Data cannot move directly to an outside sink from a data store. Data must be moved by a process.

Rules for Source/Sink (External Entities):



- Data cannot move directly from a source to a sink. They must be moved by a process if the data are of any concern to our system. Otherwise, the data flow is not shown on the DFD

Rules for Data Flow:



- A data flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated, however, by two separate arrows because the read and update usually happen at different times.

Functional Decomposition:

It is an iterative process of breaking a system description down into finer and finer detail, which creates a set of charts in which one process on a given chart is explained in greater detail on another chart. Functional decomposition is an act of going from one single system to many component processes. It is a repetitive procedure. Decomposition goes on until you have reached the point where no sub process can logically be broken down any further. The lowest level of DFDs is called a primitive DFD.

DFD Levels:

- **Context Diagram/ Level-0 DFD:** Overview of the organizational system.
- **Level-1 DFD:** Representation of system's major processes at high level of abstraction.
- **Level-2 DFD:** Results from decomposition of Level 0 diagram.
- **Level-n DFD**: Results from decomposition of Level n-1 diagram.

Level-0 DFD/ Context Diagram:

Level-0 shows the system boundaries, external entities that interact with the system, and major information flows between entities and the system. It is the first DFD in every business process. It shows the context into which the business process fits. It also shows the overall business process as just one process.

It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

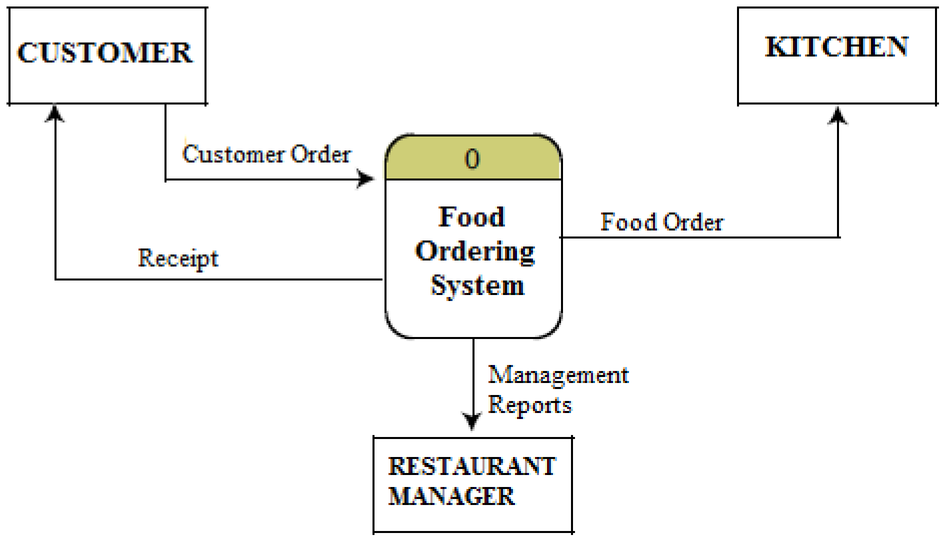


Figure: Context Diagram of Food Ordering System

Level-1 DFD:

Level-1 DFD shows the system's major processes, data flows, and data stores at a high level of abstraction. It shows all the major processes that comprise the overall system – the internal components of process 0. It also shows how the major processes are interrelated by data flows. It shows external entities and the major processes with which they interact. It adds data stores.

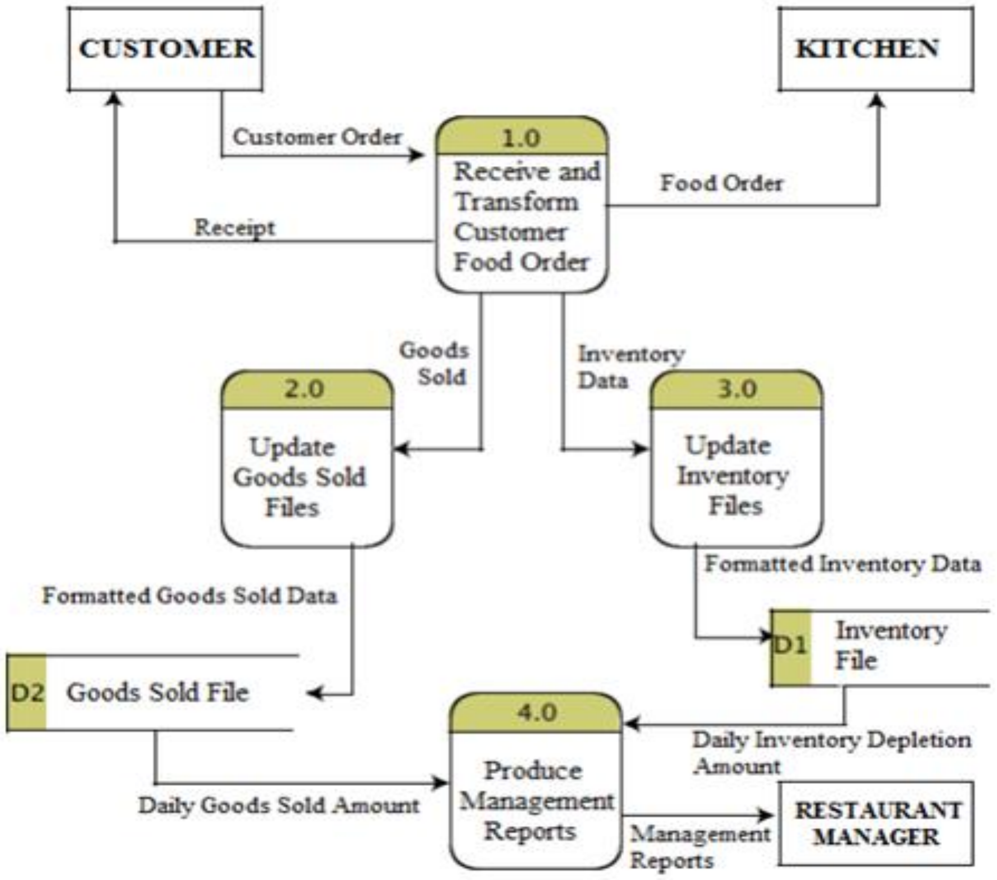


Figure: Level-1 DFD of food ordering system

Level-2 DFD:

Level-2 DFD shows the sub-processes of one of the processes in the Level-1 DFD. Generally, one level 1 diagram is created for every major process on the level 1 diagram. It shows all the internal processes that comprise a single process on the level 1 diagram. It also shows how information moves from and to each of these processes.

If a parent process is decomposed into, for example, three child processes, these three child processes wholly and completely make up the parent process.

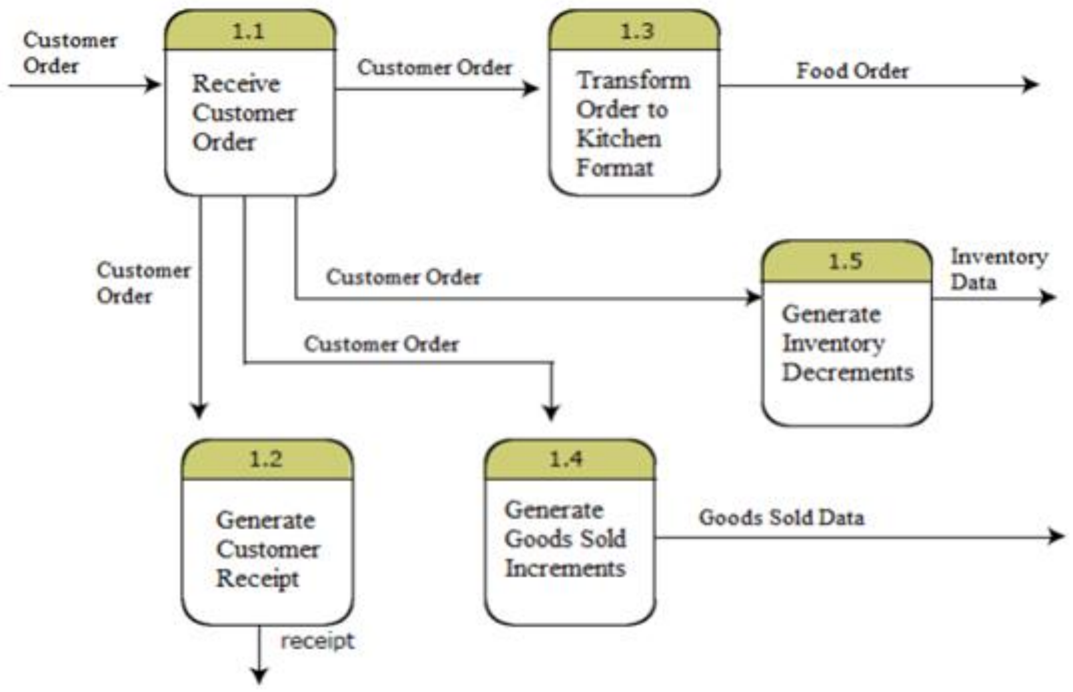


Figure: Level-2 diagram showing Decomposition of process 1.0

Level-n DFD:

Level-n DFD shows the sub-processes of one of the processes in the Level n-1 DFD. It shows all processes that comprise a single process on the previous level diagram. It also shows how information moves from and to each of these processes.

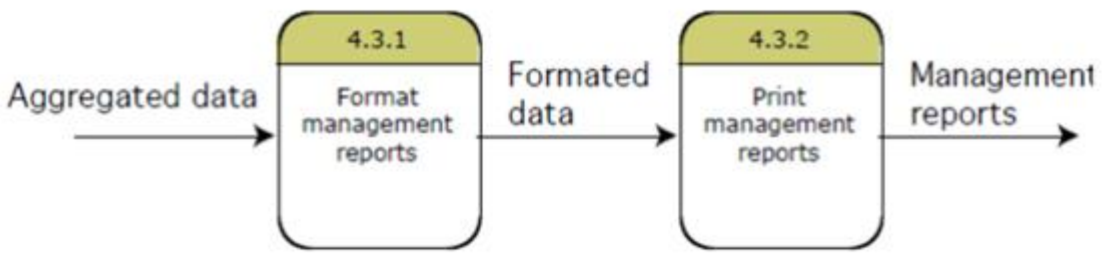


Figure: Level-n DFD showing the decomposition of process 4.3 from the level-2 Diagram

Logical DFD vs. Physical DFD

- Data flow diagrams are categorized as either logical or physical. A logical data flow diagram focuses on the business and how the business operates. It is not concerned with how the system will be constructed.
- A physical data flow diagram shows how the system will be implemented, including the hardware, software, files, and people in the system. It is developed such that the processes described in the logical data flow diagrams are implemented correctly to achieve the goal of the business.
- A logical DFD makes it easier to communicate for the employees of an organization, leads to more stable systems, allows for better understanding of the system by analysts, is flexible and easy to maintain, and allows the user to remove redundancies easily. On the other hand, a physical DFD is clear on division between manual and automated processes, gives detailed description of processes, identifies temporary data stores, and adds more controls to make the system more efficient and simple.
- A logical DFD captures the data flows that are necessary for a system to operate. It describes the processes that are undertaken, the data required and produced by each process, and the stores needed to hold the data. On the other hand, a physical DFD shows how the system is actually implemented, either at the moment (Current Physical DFD), or how the designer intends it to be in the future (Required Physical DFD). Thus, a Physical DFD may be used to describe the set of data items that appear on each piece of paper that move around an office, and the fact that a particular set of pieces of paper are stored together in a filing cabinet.

Logic Modeling

A logic model is a graphic depiction (road map) that presents the shared relationships among the resources, activities, outputs, outcomes and impact for your program. It depicts the relationship between your program's activities and its intended effects.

Data flow diagrams do not show the logic inside the processes - what occurs within a process? How input data is converted into output information. Logic modeling involves representing internal structure and functionality of processes depicted on a DFD. Processes must be clearly described before translating them into programming language. Logic modeling can also be used to show when processes on a DFD occur.

Logic modeling will be generic without taking syntax of a particular programming language. A logic model has four components:

1. **Needs** are about the problem and why it's important to address it. What issue are we trying to address?
2. **Inputs** are the things contribution to addressing the need (usually a combination of money, time, expertise and resources). What resources are we investing?
3. **Activities** describe the things that the inputs allow to happen. What difference are we these resources?
4. **Outcomes** are usually expressed in terms of measures of success. What difference are we hoping to make?

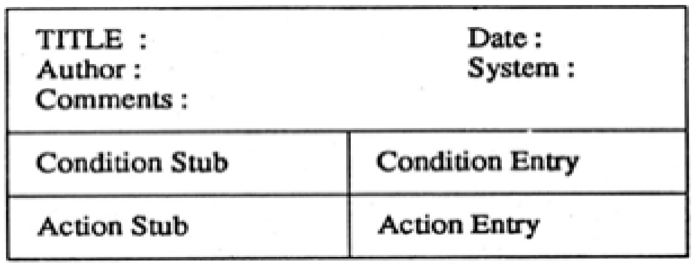
Each process on the lowest level DFD will be represented by one or more of the following

- Structured English
- Decision Tables
- Decision Trees
- State-transition diagrams
- Sequence diagrams
- Activity diagrams

Modeling Logic with Decision Tables

The decision table or a logic table is a chart with four sections listing all the logical conditions and actions. In addition the top section permits space for title, date, author, system and comment.

A decision table appears as a matrix of rows and columns that shows conditions and corresponding actions. Decision rules, included in a decision table, states what procedure is to follow when certain condition exist.



TITLE : Author : Comments :	Date : System :
Condition Stub	Condition Entry
Action Stub	Action Entry

Figure: Format of Decision Table

A decision table is divided into four sections:

1. Condition Stub
2. Condition Entries
3. Action Stub
4. Action Entries

Condition stub identifies total set of relevant test or condition. These conditions require yes or no answers. Combination of these conditions are then identified and expressed as rules or conditions entries. The condition stub contains a list of all the necessary tests in a decision table. In the lower left-hand corner of the decision table we find the action stub where one may note all the processes desired in a given module.

Condition entries provide all possible permutations of yes or no responses related to the condition statement. The upper right corner provides the space for the condition entry - all possible permutations of yes and no responses related to the condition stub. The yes and no possibilities are arranged as a vertical column called rules. Rules are numbered 1, 2, 3 and so on. We can determine the rules in a decision table by the formula:

Number of rules = 2^N = 2N where N represents the number of condition and ^ means exponentiation. Thus a decision table with four conditions has 16 ($2^4 = 2 \times 2 \times 2 \times 2 = 16$) rules one with six conditions has 64 rules and eight conditions yield 256 rules.

Action stub list the possible actions which can occur as a result of different condition combinations. Action Stub contains a list of all the processes involved in a decision table.

Action entries show what specific action in the set to take when selected or group of conditions are true. Action entry indicates via dot or X whether something should happen in a decision table.

Steps of Creating a Decision Table

1. Determine conditions and their values. The number of conditions becomes the number of rows in the top half of the decision table.

2. Determine possible actions that can be taken. This becomes the number of rows in the lower half of the decision table.

3. Determine condition alternatives for each condition In the simplest form two alternatives (Y or N), in an extended there may be many alternatives.

4. Calculate the maximum number of columns (Rules) in the decision table For example, say we have Conditions A, B, and C. Each condition has the following values:

Condition A: Y, N

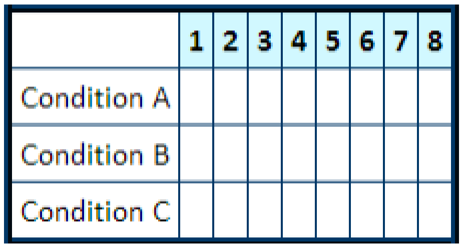
Condition B: 1, 2, 3

Condition C: Y, N

In this example, A and C have 2 values, and B has 3 values. Therefore, the number of rule columns is 2 * 3 * 2 = 12.

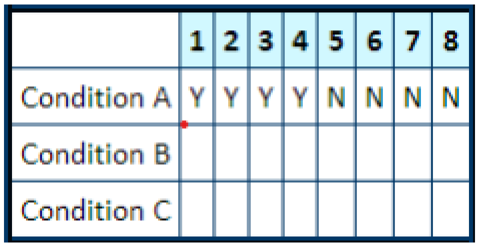
5. Fill in the condition alternatives It is done to ensure we get every possible combination of values in the table. Example: A table has 3 conditions: Condition A, B, and C. All three conditions have two possible values: Y (yes) and N (no). Draw the condition section and populate the condition alternatives.

Since each condition has 2 possible values, there will be 2 * 2 * 2 = 8 rule columns:



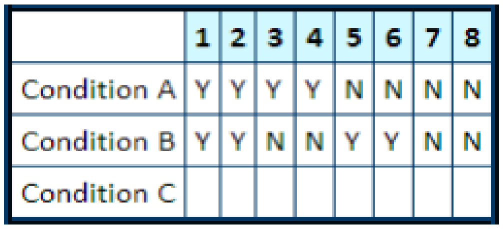
	1	2	3	4	5	6	7	8
Condition A								
Condition B								
Condition C								

To calculate the first row, divide the number of rules by 2. That gives us 4. The first row will have 4 Y's and 4 N's and that should fill the whole row:



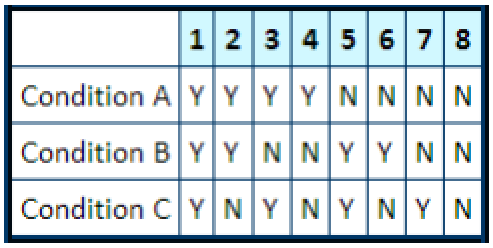
	1	2	3	4	5	6	7	8
Condition A	Y	Y	Y	Y	N	N	N	N
Condition B								
Condition C								

For the second row, we divide the value from the previous row (4) by 2: 4/2 = 2. So we now fill in the second row with two Y's, two N's, two Y's, two N's, until the row is full:



	1	2	3	4	5	6	7	8
Condition A	Y	Y	Y	Y	N	N	N	N
Condition B	Y	Y	N	N	Y	Y	N	N
Condition C								

The last row should work out such that the values just alternate, one of each across the row. To make sure, let's get the previous row's value and divide by 2: 2/2 = 1. So we should have the last row filled with one Y, one N, one Y, one N, until the row is full. Our final table now looks like this:



	1	2	3	4	5	6	7	8
Condition A	Y	Y	Y	Y	N	N	N	N
Condition B	Y	Y	N	N	Y	Y	N	N
Condition C	Y	N	Y	N	Y	N	Y	N

Advantages of Decision Table

- Simple and easy to understand by non-computer literate users and managers
- Good documentation of rules used in data processing.
- Simple representation of complex decision rules.
- Tabular representation allows systematic validation of specification detection of redundancy, incompleteness & inconsistency of rules
- Algorithms exist to automatically convert decision tables to equivalent computer programs.
- Allows systematic creation of test data

- Help the analysis ensure completeness
- Easy to check for possible errors
- Enhances readability
- Alternatives are shown side by side.
- Cause & effect relationship is shown, thus permitting easier user validation.
- Aids in analysis of structured decisions.

Decision Tree

Decision tree turns a decision table into a diagram. It consists of nodes (a square for actions and a circle for conditions) and branches.

A decision tree is a diagram that resembles a tree, with a root on the left hand side and branches representing each decision. It is read from left to right and the actions to be undertaken are recorded down the right hand side of the diagram. The root of the tree, on the left of the diagram is starting point of the decision sequence. The particular branch to be followed depends on the prevalent conditions and decision to be made. Progression from left to right along the particular branch is the result making a series of decisions. Following each decision point is the next set of decision to be considered. The nodes of the tree thus represent conditions and indicate that a determination must be node about which condition exist before next path can be chosen. The right side of the tree lists the action to be taken depending on the sequence of conditions that is followed.

Example: Let us consider the following example:

If order is from book store
And if order is for 6 copies or more
Then discount is 25%
Else (if order is for less than 6 copies)
No discount is allowed
Else (if order is from libraries)
If order is for 50 copies or more
Then discount is 15%

Else if order is for 20 to 49 copies

Then discount is 10%

Else if order is for 6 to 19 copies

Then discount is 5%

Else (order is for less than 6 copies)

No discount is allowed

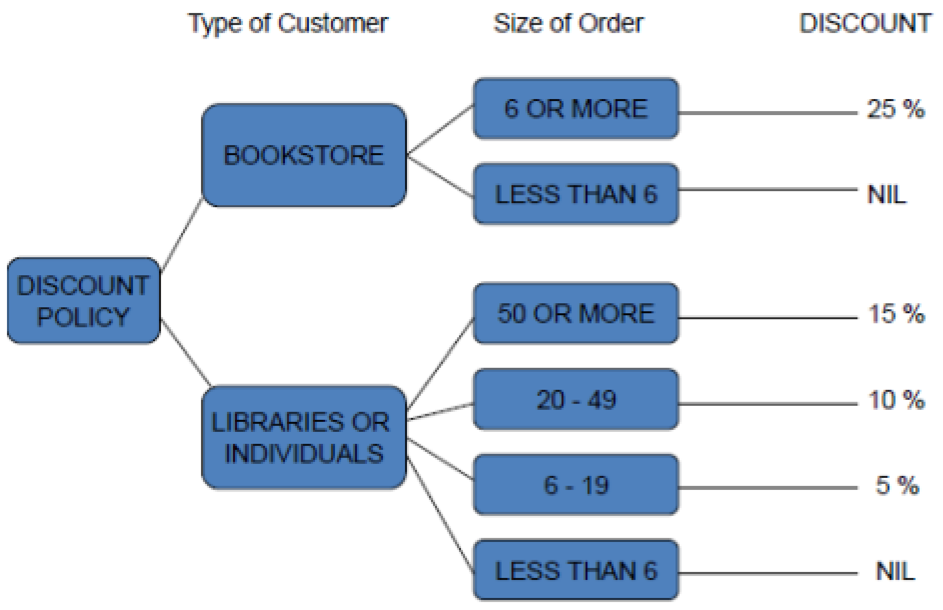


Figure: Decision Tree for Book Order

Developing a decision tree is beneficial to analyst as the need to describe condition and action compels the analyst to formally identify the actual decision that must be taken. A decision tree helps to show the paths that are possible in a design following an action or decision by the user.

Advantages of Decision Tree

- The order of checking conditions and executing actions is immediately noticeable
- It is easy to understand and interpret
- It can map nicely to a set of business rules
- It can be applied to any real world problems
- It makes no prior assumptions about the data
- It has ability to process both numerical and categorical data

• It can be combined with other decision techniques.

• Worst, best and expected values can be determined for different scenarios

Differences between Decision Table and Decision Tree

- Decision tables are used when the process is logically complex involving large number of conditions and alternate solutions. Use Decision tables when there are a large number of conditions to check and logic is complex. A major drawback of a decision tree is the lack of information in its format to tell us what other combinations of conditions to test. This is where the decision table is useful.
- Decision Trees are used when conditions to be tested must follow a strict time sequence. Decision trees are used to verify logic and in problems that involve a few complex decisions resulting in limited number of actions. Use Decision trees when sequencing of conditions is important and if there are not many conditions to be tested.
- Conditions and actions of decision trees are found on some branches but not on others, which contrasts with decision tables, in which they are all part of the same table. Those conditions and actions that are critical are connected directly to other conditions and actions, whereas those conditions that do not matter are absent. In other words it does not have to be symmetrical.
- Compared to decision tables, decision trees are more readily understood by others in the organization. Decision Table can create more queries; it is more of multipath/multiflow. Decision Tree follows single path.
- Decision tables are better than decision trees at portraying complex logic, are more compact, and are easier to manipulate. Decision trees are better than decision tables at portraying simple problems and at helping people make decisions in practice.

Pseudo Codes

Pseudo-code is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations. It is used for creating an outline or a rough draft of a program. Pseudo-code summarizes a program's flow, but excludes underlying details. System designers write pseudo-code to ensure that programmers understand a software project's requirements and align code accordingly.

It's simply an implementation of an algorithm in the form of annotations and informative text written in plain English. It has no syntax like any of the programming language and thus can't be compiled or interpreted by interpreted by the computer.

Advantages of Pseudo-code

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm
- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
- The main goals of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

Structuring System Data Requirements

Structuring system and database requirements concentrates on the definition, structure, and relationship within data. The characteristics of data captured during data modeling are crucial in the design of database, programs, computer screen and printed reports. The information is essential in ensuring data integrity in an information system. Moreover data are often the most complex aspects of many modern information systems and are reasonably permanent. The most common format used for data modeling is entity-relationship diagramming. Data modeling explains the characteristics and structure of data independent of how the data may be stored in computer memories and is usually developed iteratively.

Conceptual Data Modeling

Conceptual data model is a detailed model that captures the overall structure of data in an organization. It is independent of any database management system (DBMS) or other implementation considerations. Conceptual data modeling is the representation of organizational data. Its purpose is to show rules about the meaning and interrelationships among data. Entity-Relationship (E-R) diagrams are commonly used to show how data are organized. The main goal of conceptual data modeling is to create accurate E-R diagrams. Methods such as interviewing, questionnaires, and JAD are used to collect information. Consistency must be maintained among process flow, decision logic, and data modeling descriptions.

The Process of Conceptual Data Modeling

First step is to develop a data model for the system being replaced. Next, a new conceptual data model is built that includes all the requirements of the new system. In the design stage, the conceptual data model is translated into a physical design. Project repository links all design and data modeling steps performed during SDLC. Primary deliverable is the entity-relationship diagram.

Entity-Relationship Model

The E-R data models is based on a perception of real world that consist of a collection of basic objects called entities and relationship among these objects. In an E-R model a database can be modeled as a collection of entities, and relationship among entities.

Entity Relationship Model(ER Model) is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them. Creating an ER Model in DBMS is considered as a best practice before implementing your database.

ER Modeling helps you to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing your database.

ER Diagram

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

Components of the ER Diagram

This model is based on three basic concepts:

- Entities
- Attributes
- Relationships

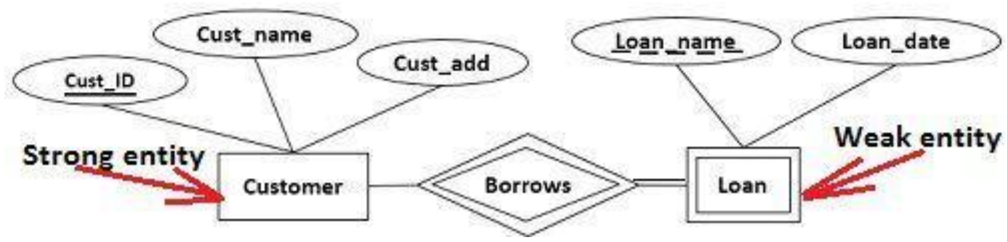
Entity

A real-world thing either living or non-living that is easily recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world. An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

There are two types of entity:

- Strong entity
- Weak entity

The **Strong Entity** is the one whose existence does not depend on the existence of any other entity in a schema. It is denoted by a **single rectangle**. A strong entity always has the **primary key** in the set of attributes that describes the strong entity. It indicates that each entity in a strong entity set can be uniquely identified.



A **Weak entity** is the one that depends on its owner entity i.e. a strong entity for its existence. A weak entity is denoted by the **double rectangle**. Weak entity do not have the **primary key** instead it has a partial key that uniquely discriminates the weak entities. The primary key of a weak entity is a composite key formed from the primary key of the strong entity and partial key of the weak entity**.**

Attributes:

The properties or characteristics of an entity are called **attributes**. For example, a customer entity can have customer-id, customer-name, customer-street, and customer-city as attributes.

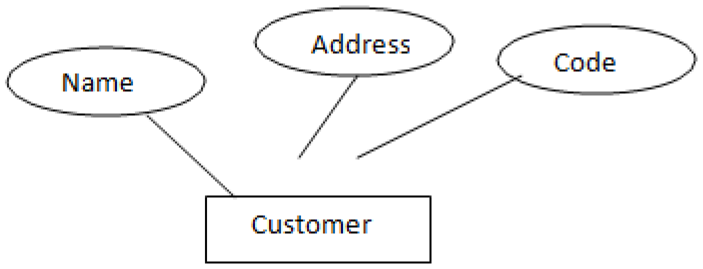


Fig: Entity with attributes

Attribute Types

An Attribute, as used in E-R model, can be characterized by the following attributes types:

Simple and composite attribute:

- **Simple:** The attributes that cannot be divided into subparts (ie. Into attributes) are called simple attributes. For example roll-number attribute of a student cannot be further divided into sub parts thus roll-number attribute of a student entity acts as a simple attribute.
- **Composite:** The attributes that can be divided into subparts (ie into attributes) are called composite attributes. For example name attribute of a particular student can be further vided into subparts first-name, middle-name, and last-name thus name attribute acts as a composite

.

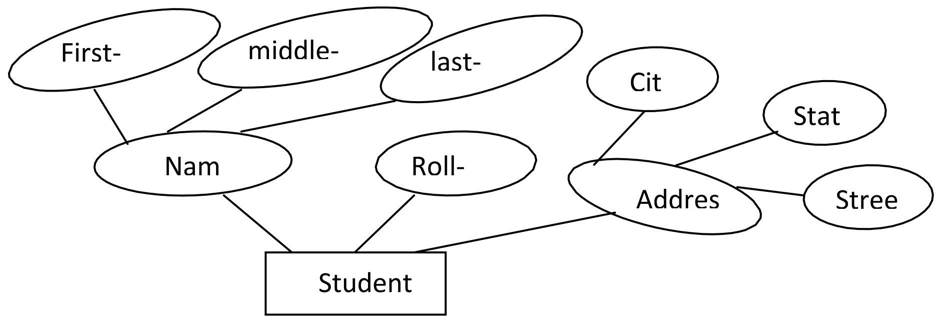
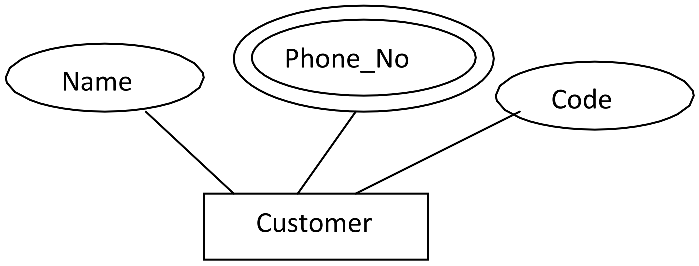


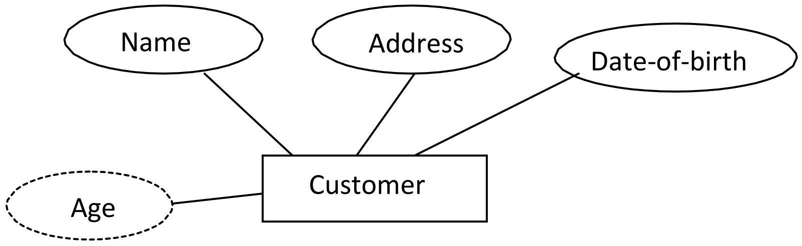
Fig:- Simple and composite attributes of Student entity

Single-valued and multi-valued attributes:

- **Single-valued:** The attributes which has a single value for a particular entity is called single-valued attributes. For example almost of our example has the single value attributes; loan-number specifies loan entity refers only one lone number.
- **Multi-valued:** If an attribute has a set of value for a specific entity is called multi-valued attributes. For example: multi-valued attribute: 'phone_number' of an employee may have zero, one or several phone numbers.



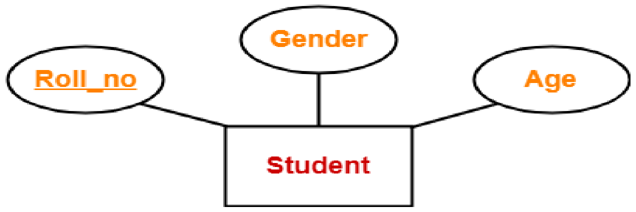
Derived attributes
The attribute whose value derived from the values of other related attributes or entities is called derived attribute. For example: age, given date of_birth.



Key Attributes

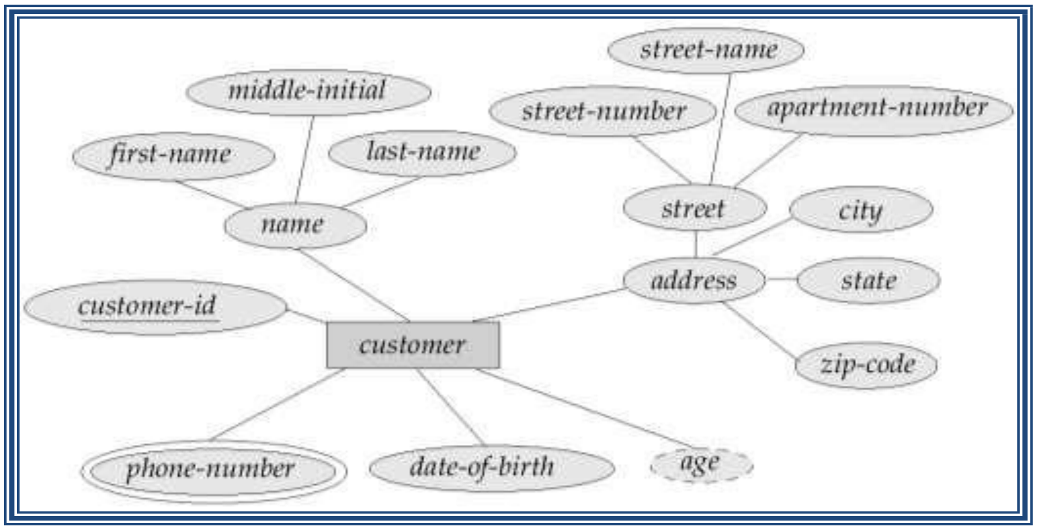
Key attributes are those attributes which can identify an entity uniquely in an entity set.

Example



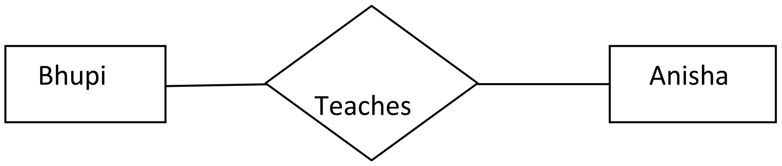
Here, the attribute "Roll_no" is a key attribute as it can identify any student uniquely.

Example: E-R diagram showing all types of attribute



Relationship and Relationship sets:

A **relationship** is an association among two or more entities. A relationship is represented by diamond shape in ER diagram. For example we may define a relationship which associates the teacher "Bhupi" with a student of name "Anisha". This specifies that Bhupi is a teacher who teaches a student of name "Anisha".

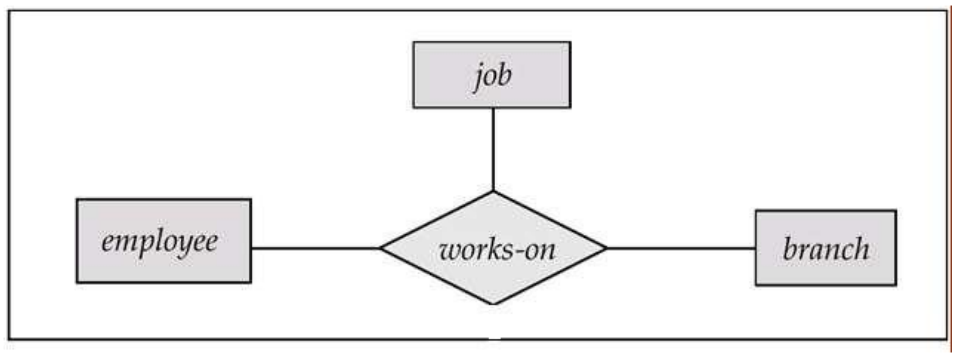


A **relationship set** is a set of relationships of the same type

Degree of a relationship

Degree of a relationship set refers to the number of entity sets that participate in a relationship set. Relationship sets that involve two entity sets are called **binary relationship** sets. Most relationship sets in a database system are binary.

Relationship sets may involve more than two entity sets called **n-ary** relationship sets butare rarer. For example, suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches. Then there is a ternary relationship set between entity sets employee, job and branch.



<u>**Constraints:**</u>

An entity relationship model may define certain constraints to which the contents of a database must conform. The most important constraints are: ***mapping cardinalities*** and ***participation constraints.***

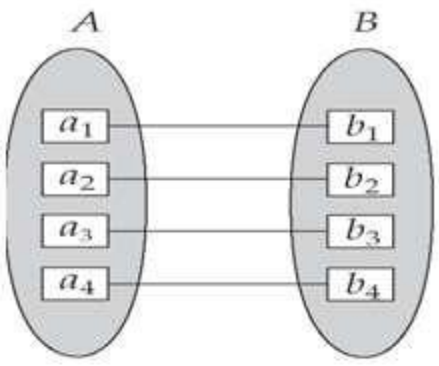
Mapping Cardinality Constraints:

Mapping cardinality or cardinality ratio express the number of entities to which another entity can be associated via a relationship set. The mapping cardinality is most useful in describing binary relationship sets. (mapping cardinality also used for other relationship that is ternary etc.) For a binary relationship set the mapping cardinality must be one of the following types:

<u>**One-to-One:**</u>

An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. The following fig shows one to one mapping cardinality of entity sets A and B.

Stud



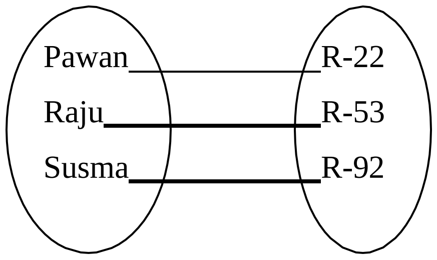
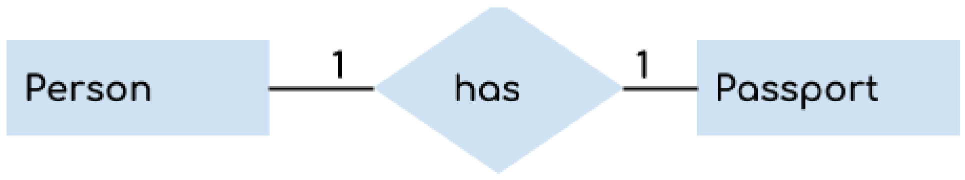
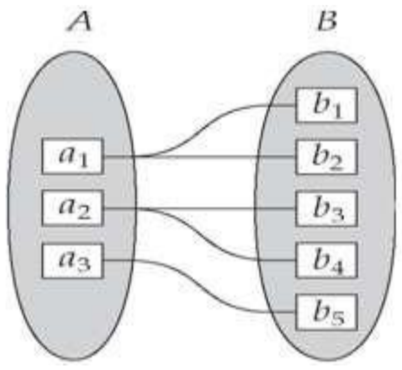


Fig: One-to-One



One-to-Many:

An entity in A is associated with any number (zero or more) of entities in B. An entity in B however can be associated with at most one entity in A. The following fig shows one-to-many mapping cardinality of entity sets A and B.



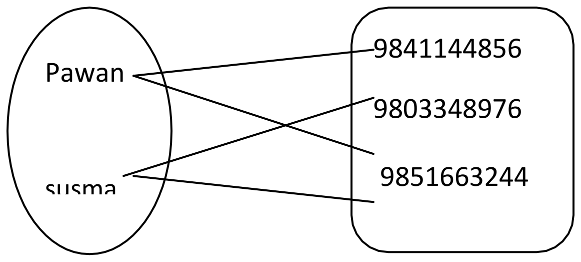
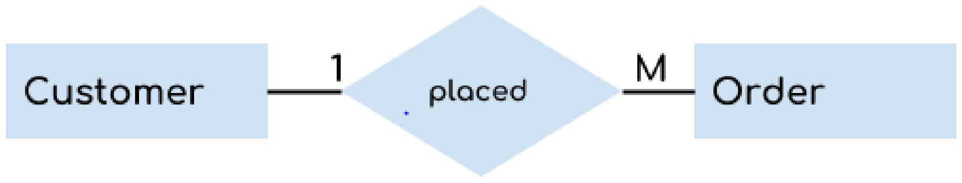
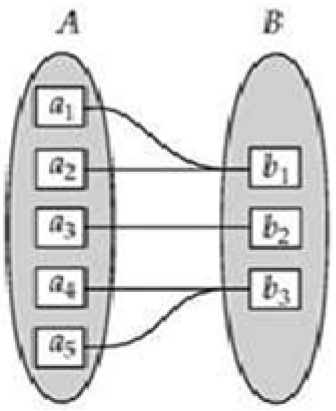


Fig: One-to-Many



Many-to-One:

An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A. The following fig clearly shows the many to one cardinality between entity sets A and B.



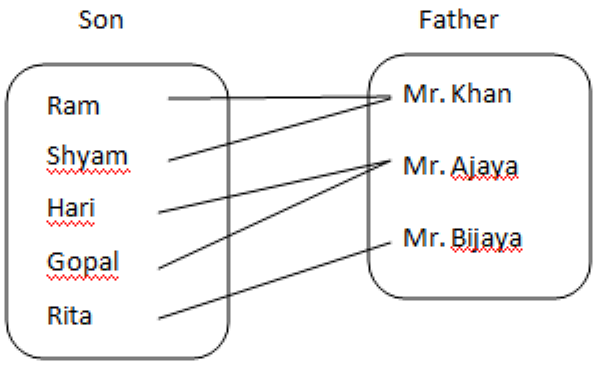
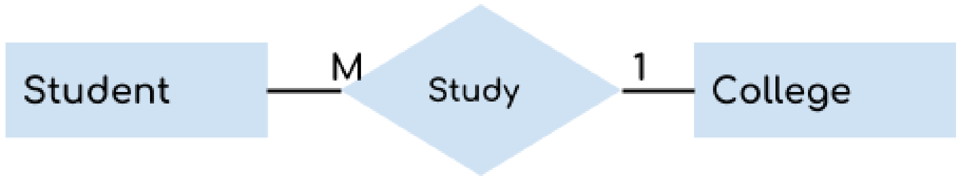
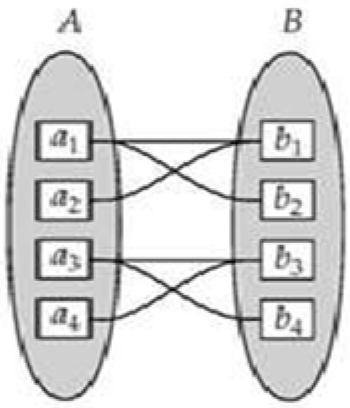


Fig: Many-to-One



Many-to-Many:

An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A. The following fig clearly shows the many-to-many cardinality between entity sets A and B.



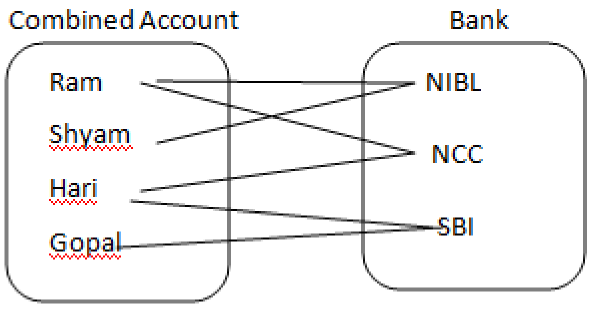
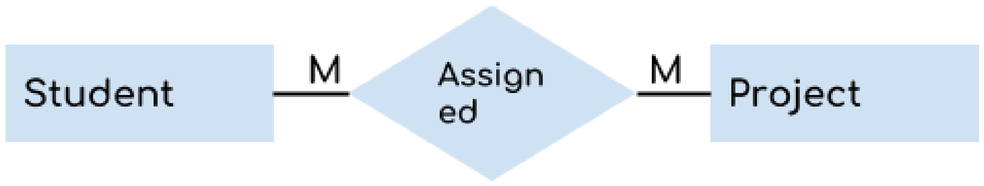


Fig: Many-to-Many



<u>Participation Constraints:</u>

The participation constraint specifies whether the existence of an entity depends on its beingrelated to another entity via the relationship type.

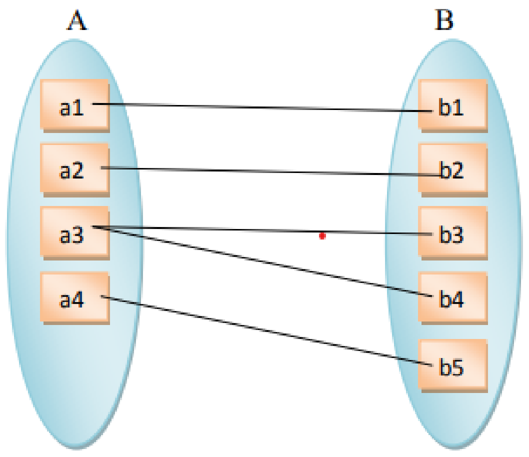
There are two types of participation constraints:

I. Total Participation Constraints

II. Partial Participation Constraints

<u>Total Participation Constraints:</u>

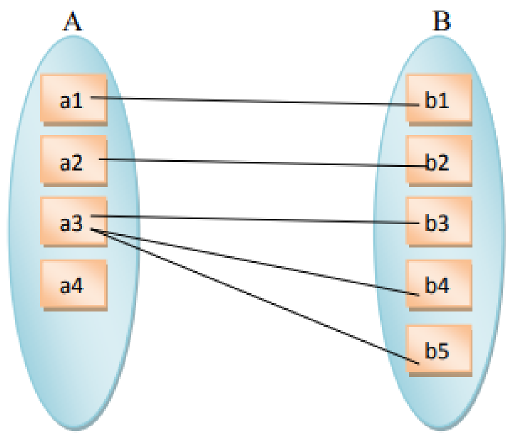
The Participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.



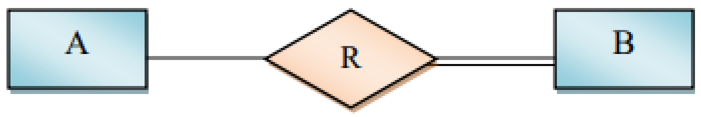
The participation of entity set A in the relationship set is total because every entity of A participates in the relationship set and The participation of entity set B in the relationship set is also total because every entity of B also participates in the relationship set.

<u>Partial Participation Constraints:</u>

The participation of an entity set E in relationship set R is said to be partial if only some entities in E participate in relationships in R.



In ERD, the total participation is denoted by **doubled-line** between entity set and relationship set and partial participation is denoted by **single line** between entity set and relationship set.

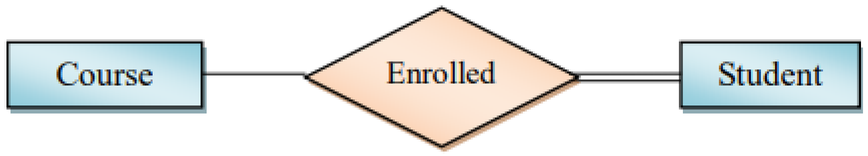


Participation in R: **Partial A**
Total B

Example: Suppose an entity set Student related to an entity set Course through Enrolled relationship set.

The participation of entity set course in enrolled relationship set is partial because a course may or may not have students enrolled in. It is possible that only some of the course entities are related to the student entity set through the enrolled relationship set.

The participation of entity set student in enrolled relationship set is total because every student is expect to relate at least one course through the enrolled relationship set.



Participation in Enrolled relationship set: **Partial** Course
Total Student

Keys:

Keys are used to distinguish the entities within a given entity set. Keys also help to uniquely identify relationships.

There are different types of keys which are as

- Super key
- Candidate key
- Primary key
- Foreign key

Super key

A super key is a set of one or more attributes which uniquely identifies an entity in entity set. For example: in customer relation single attribute customer_id is sufficient to uniquely identify one customer entity to another. So customer_id is a superkey in a customer relation. Since combination of customer_id and customer_name can also uniquely identifies one customer entity to another. So combination of attributes {customer_id,customer_name} is also superkey

in relation customer. But single attribute customer_name can not superkey in relation customer because customer name only can not uniquely identify one customer entity to another, there would be number of customers having same name. The above example of supekey shows that superkey may contains extraneous attributes. That is, if K is superkey then any superset of K is superkey.

Candidate key

The minimal superkey called candidate key. That is, candidate key is a superkey but its proper subset is not superkey. For example: customer_id is a candidate key in customer relation. Similarly account_id is a candidate key in account relation.

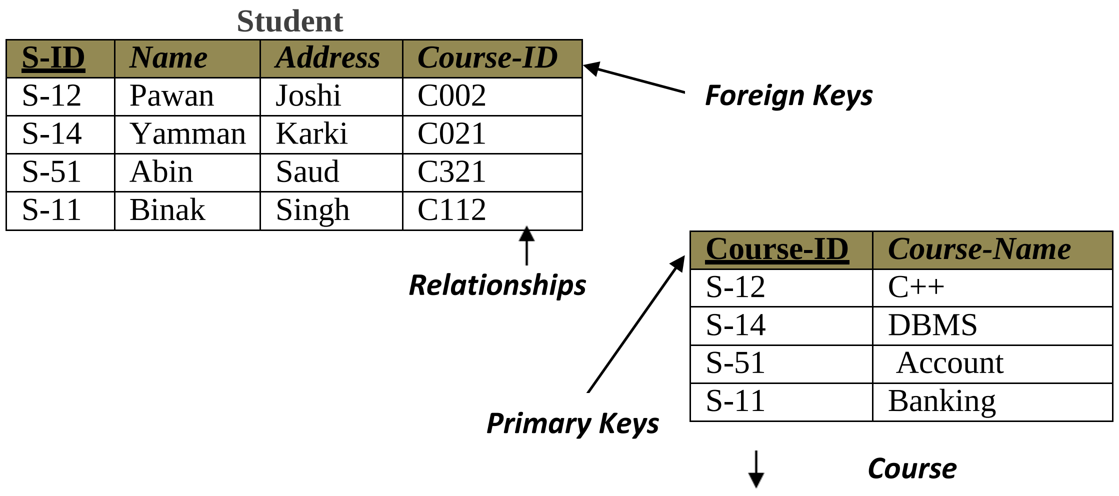
Primary key

In a relation, it is possible that we can choose distinct set of attributes as a candidate key. For example: in customer we can choose single attribute {custome_id} or set attributes {customer_name, customer_city} as candidate key. Candidate key chosen by database designer for particular relation known as primary key.

Foreign key:

A foreign key (FK) is an attribute or combination of attributes that is used to establish and enforce a link between the data in two tables. You can create a foreign key by defining a FOREIGN KEY constraint when you create or modify a table.

In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.



Student

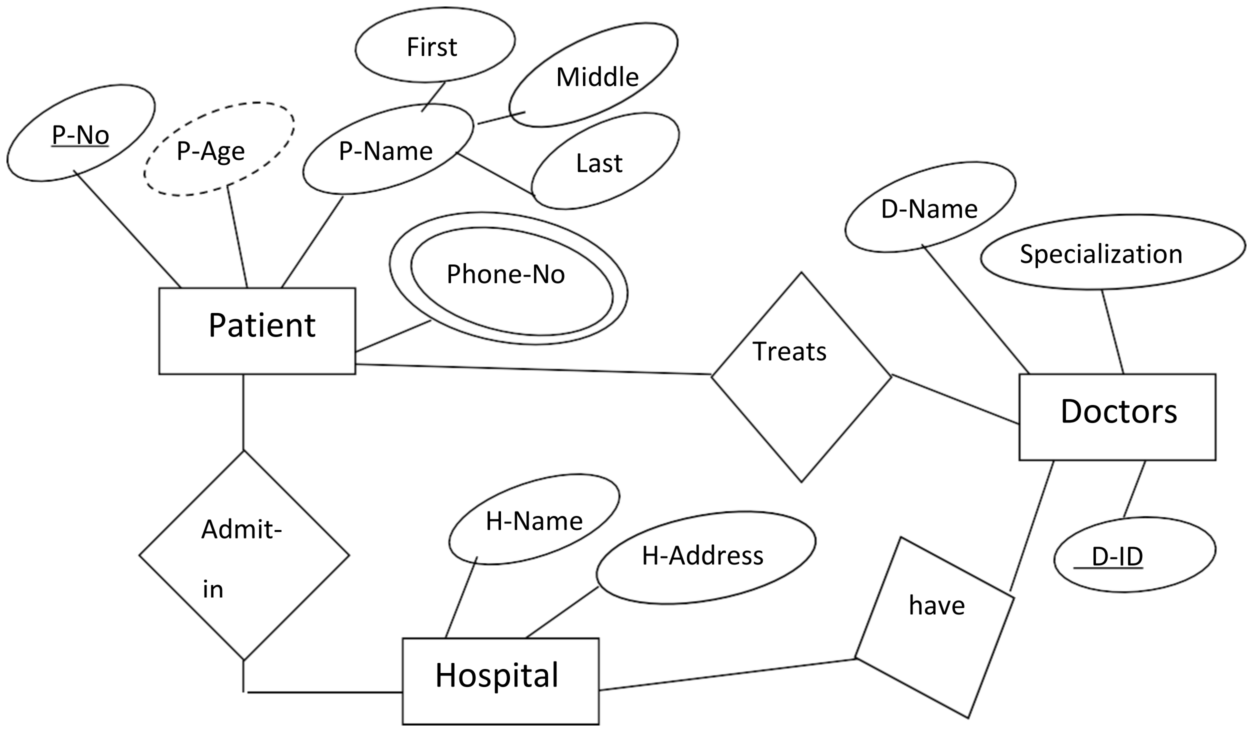
S-ID	Name	Address	Course-ID
S-12	Pawan	Joshi	C002
S-14	Yamman	Karki	C021
S-51	Abin	Saud	C321
S-11	Binak	Singh	C112

Course-ID	Course-Name
S-12	C++
S-14	DBMS
S-51	Account
S-11	Banking

Fig: Primary key and foreign key

<u>Some Example of E-R Diagram</u>

Example 1: E-R diagram for hospital with a set of patients and medical doctors..



Example 2: Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted

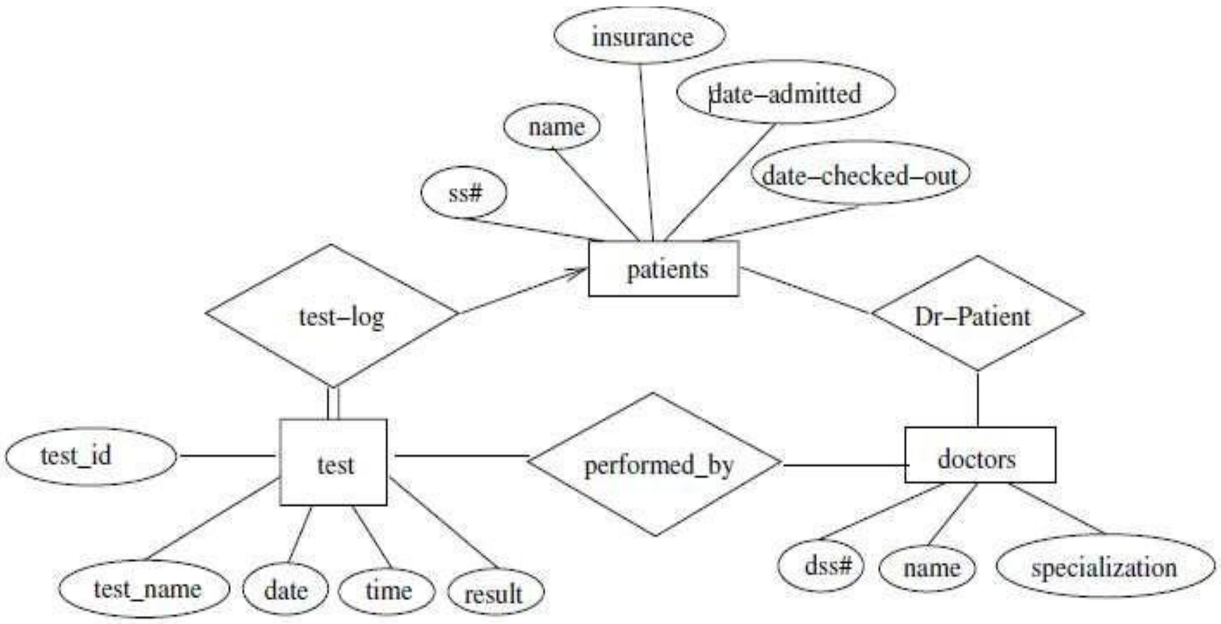
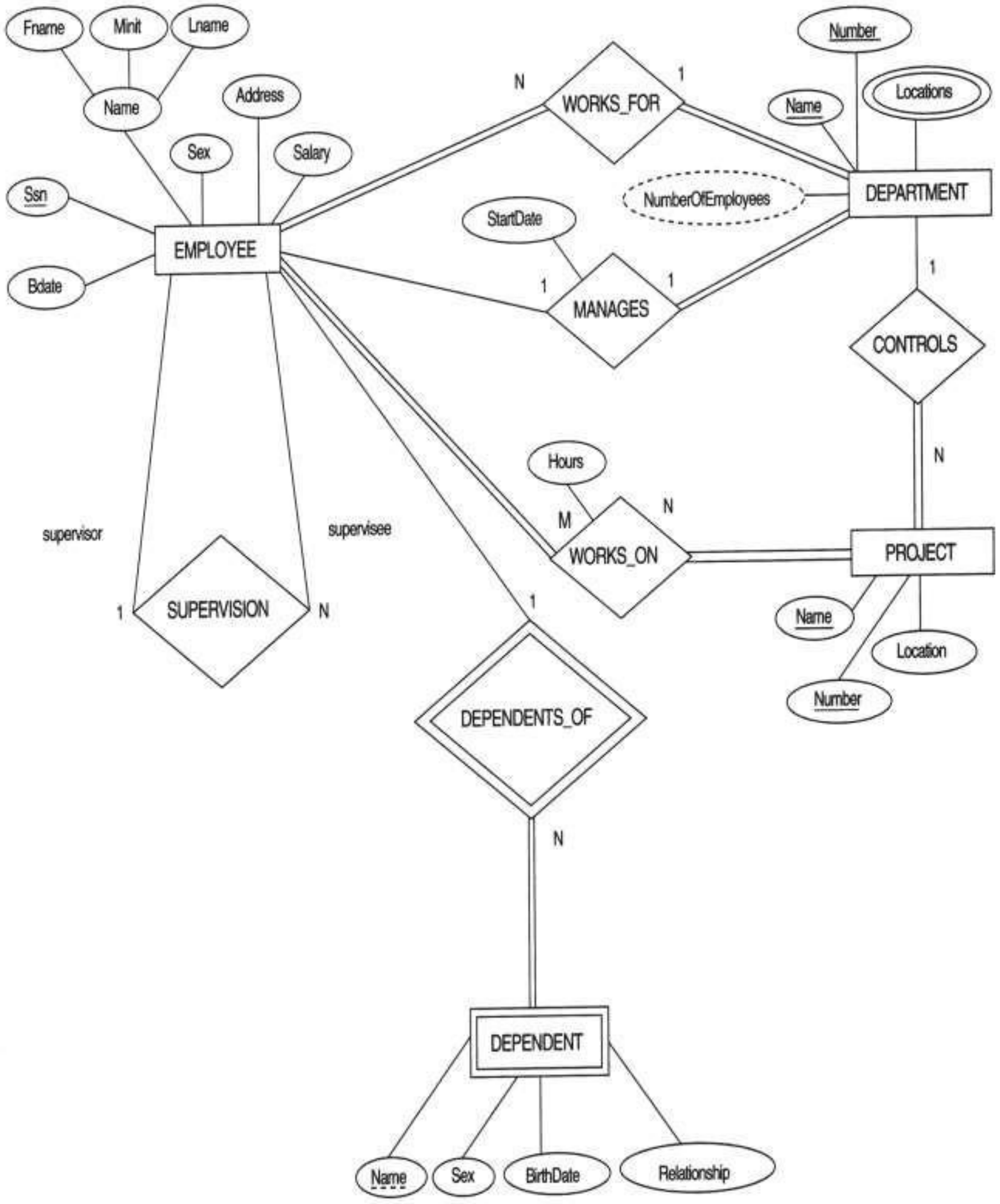
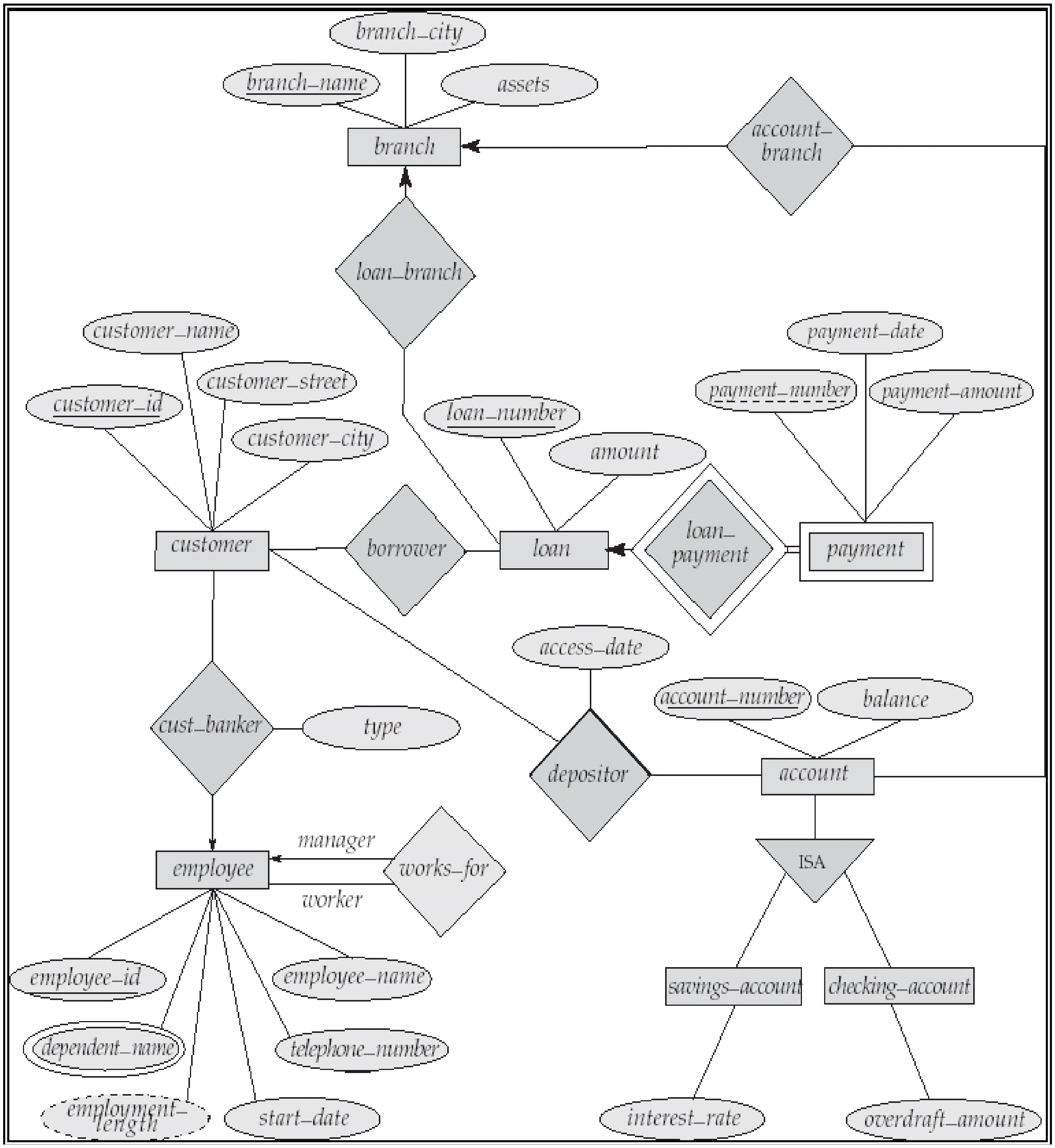


Figure E-R diagram for a hospital.

Example 3: ER diagram for Company database system



Example 4: E-R Diagram for a Banking Enterprise



Example 5: Construct an E-R diagram for a car-insurance company whose customers own one ormore cars each. Each car has associated with it zero to any number of recorded accidents.

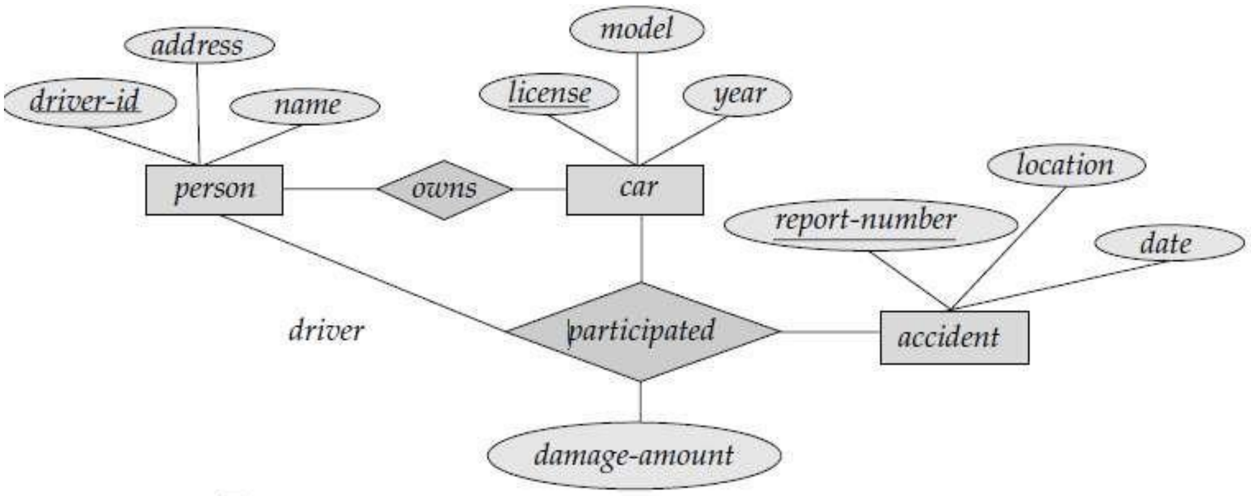
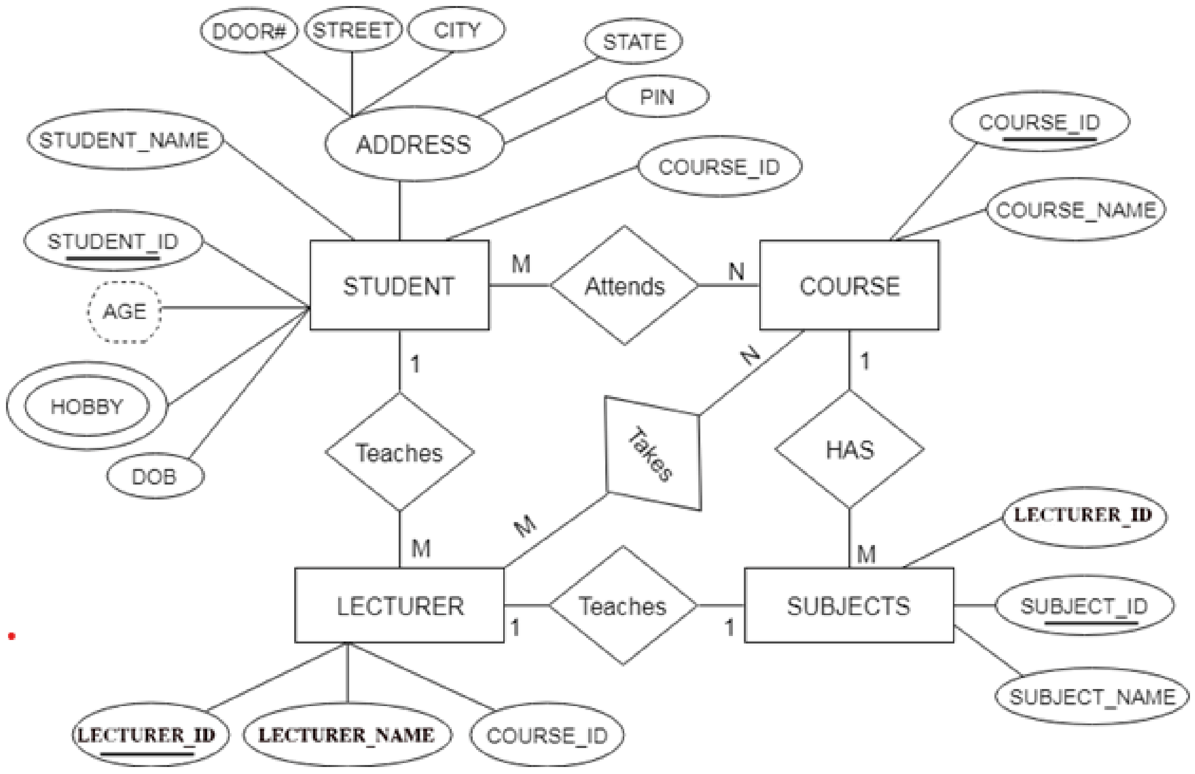


Figure E-R diagram for a Car-insurance company.

Example 6: Draw an E-R diagram for college management system


DOOR#
STREET
CITY
STATE
PIN
ADDRESS
STUDENT_NAME
COURSE_ID
STUDENT_ID
AGE
HOBBY
DOB
STUDENT
M
Attends
N
COURSE
COURSE_ID
COURSE_NAME
1
Teaches
N
Takes
1
HAS
M
M
M
LECTURER
1
Teaches
1
SUBJECTS
LECTURER_ID
SUBJECT_ID
SUBJECT_NAME
LECTURER_ID
LECTURER_NAME
COURSE_ID