UNIT 3
Analysis

LH- 13 HRS

Er. Rolisha Sthapit

SYSTEM ANALYSIS AND DESIGN (SAD)

Contents

3.1 System requirements

3.1.1 Introduction, 3.1.2 Performing Requirements Determination, 3.1.3 Traditional Methods For Determining Requirements, 3.1.4 Contemporary Methods For Determining System Requirements, 3.1.5 Radical Methods For Determining System Requirements, 3.1.6 Requirements Management Tools, 3.1.7 Requirement Determination Using Agile Methodologies

3.2 System Process Requirements

3.2.1 Introduction, 3.2.2 Process Modeling, 3.2.3 Data Flow Diagramming Mechanics, 3.2.4 Using DFD In The Analysis Process, 3.2.5 Modeling Logic With Decision Tables

3.3 System Data Requirements

3.3.1 Introduction, 3.3.2 Conceptual Data Modeling, 3.3.3 Gathering Information For Conceptual Data Modeling, 3.3.4 Introduction To ER Modeling, 3.3.5 Conceptual Data Modeling And The ER Model, 3.3.6 Representing SuperTypes And Sub-Types, 3.3.7 Business Rules, 3.3.8 Role Of Packaged Conceptual Data Models- Database Patterns

Analysis

- Analysis is the first systems development life cycle (SDLC) phase where you 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 systems development project under consideration has merit and should be pursued through this phase.
- 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.
- The purpose of analysis is to determine what information and information processing services are needed to support selected objectives and functions of the organization. Gathering this information is called requirements determination.

- The results of the requirements determination can be structured according to three essential views of the current and replacement information systems:
- **Process:** The sequence of data movement and handling operations within the system.
- **Logic and timing:** The rules by which data are transformed and manipulated and an indication of what triggers data transformation.
- **Data:** The inherent structure of data independent of how or when they are processed.

3.1 System requirements

3.1.1 Introduction,

3.1.2 Performing Requirements Determination,

3.1.3 Traditional Methods For Determining Requirements,

3.1.4 Contemporary Methods For Determining System Requirements,

3.1.5 Radical Methods For Determining System Requirements,

3.1.6 Requirements Management Tools,

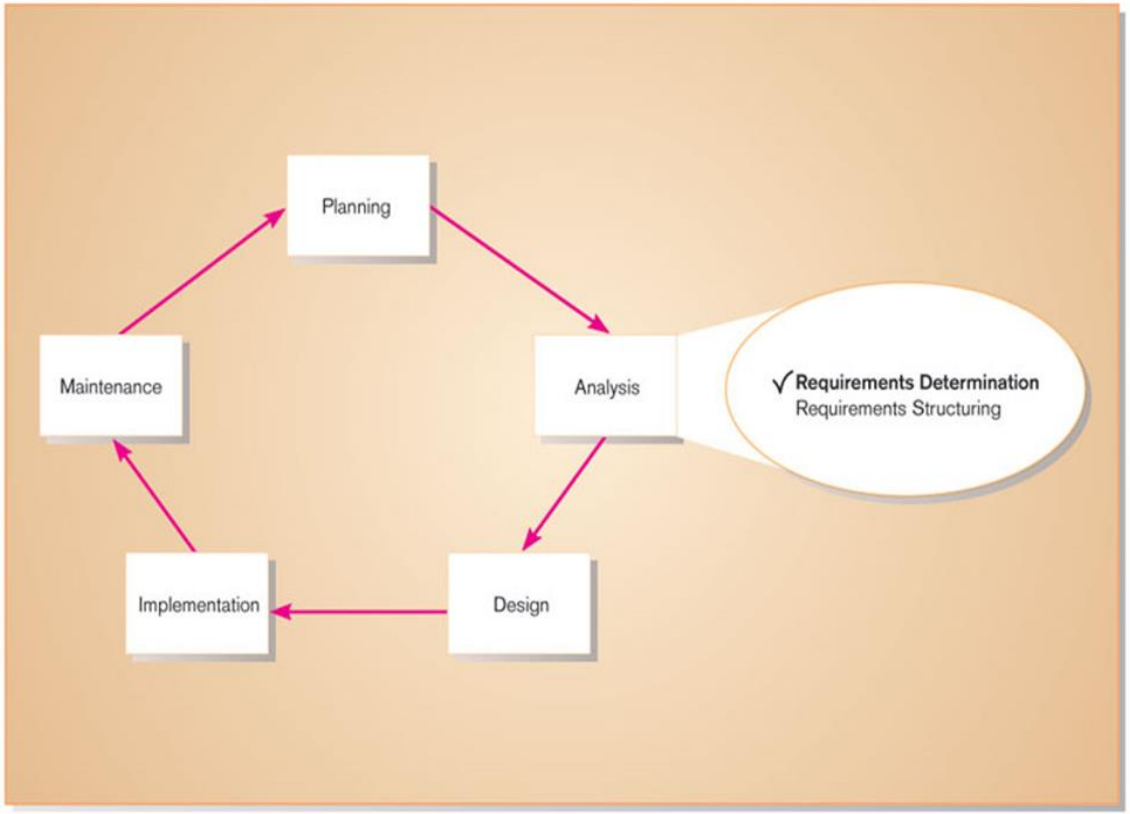
3.1.7 Requirement Determination Using Agile Methodologies

3.1SystemRequirements

3.1.1 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.
- Analysis has two sub phases: **requirements determination and requirements structuring.**
- **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.

Figure 6-1 Systems development life cycle with analysis phase highlighted



3.1.2 Performing Requirements Determination

- Once management has granted permission to pursue(follow or start) development of a new system (this was done at the end of the project identification and selection phase of the SDLC) and a project is initiated and planned, you begin 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.
- We can detect some similar characteristics for a good systems analysts during the requirements determination subphase.

- These characteristics include:
- **Impertinence:** You should question everything. You need to ask questions such as: Are all transactions processed the same way? Could anyone be charged something other than the standard price? Might we someday want to allow and encourage employees to work for more than one department?
- **Impartiality:** Your role is to find the best solution to a business problem or opportunity. It is not, for example, to find a way to justify the purchase of new hardware or to insist on incorporating what users think they want into the new system requirements. 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. For example, do not accept this statement: “We’ve always done it that way, so we have to continue the practice.” Traditions are different from rules and policies. Traditions probably started for a good reason but, as the organization and its environment change, they may turn into habits rather than sensible procedures

- **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. For example, an imprecise (not accurate or exact) definition of who a customer is may mean that you purge (remove) customer data when a customer has no active orders, yet these past customers may be vital contacts for future sales.
- **Reframing:** Analysis is, in part, a creative process. You must challenge yourself to look at the organization in new ways. You must consider how each user views his or her requirements. You must be careful not to jump to the following conclusion: “I worked on a system like that once—this new system must work the same way as the one I built before.”

Deliverables and Outcomes

- The primary deliverables from requirements determination are the various forms of information gathered during the determination process: transcripts of interviews; notes from observation and analysis of documents; sets of forms, reports, job descriptions, and other documents; and computer-generated output such as system prototypes. In short, anything that the analysis team collects as part of determining system requirements is included in the deliverables resulting from this subphase of the systems development life cycle. Table 6-1 lists examples of some specific information that might be gathered during requirements determination.

TABLE 6-1 Deliverables for Requirements Determination

1. Information collected from conversations with or observations of users: interview transcripts, notes from observation, meeting minutes
2. 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
3. Computer-based information: results from JAD sessions, CASE repository contents and reports of existing systems, and displays and reports from system prototypes

- We could group all this information in three groups:
- Information collected from conversations with or observations of users (interview transcripts, notes from observations etc.)
- Existing written information (business mission or strategy, forms, reports, training manuals, flow charts and documentation of existing system etc.)
- Computer-based information (results from JRP sessions, CASE repository contents).
- Too much analysis is not productive, and the term analysis paralysis has been coined (invented) to describe a systems development project that has become bogged down (to be/become so involved in something difficult or complicated that you cannot do anything else) in an abundance of analysis work. Because of the dangers of excessive analysis, today's systems analysts focus more on the system to be developed than on the current system. The techniques you will learn about later in this chapter, JAD and prototyping, were developed to keep the analysis effort at a minimum yet still keep it effective. Newer techniques have also been developed to keep requirements determination fast and flexible, including continual user involvement, usage centered design, and the Planning Game from eXtreme Programming.

3.1.3 Traditional Methods For Determining Requirements

- At the core of systems analysis is the collection of information. At the outset, you must 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.
- We will learn about various ways to get information directly from stakeholders: interviews, group interviews, the Nominal Group Technique, and direct observation. You will 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 6-2.

TABLE 6-2 Traditional Methods of Collecting System Requirements

- 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

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 non routine aspects of organizational operations.
- During interviewing you will gather facts, opinions, and speculation(Guess, when you guess possible answers to a question without having enough information to be certain) and observe body language, emotions, and other signs of what people want and how they assess current systems.
- There are many ways to effectively interview someone, and no one method is necessarily better than another. Some guidelines you should keep in mind when you interview, summarized below:

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

- First, you should prepare thoroughly before the interview.
- Set up an appointment at a time and for a duration convenient for the interviewee. 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.
- You should 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 spontaneously direct the interview as you discover what expertise the interviewee brings to the session.

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: Introduction Background on Project Overview of Interview Topics to Be Covered Permission to Record Topic 1 Questions Topic 2 Questions ... Summary of Major Points Questions from Interviewee Closing	Approximate Time: 1 minute 2 minutes 1 minute 5 minutes 7 minutes ... 2 minutes 5 minutes 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 1999. He raised the issue of how to handle returned goods, but we did not have time to discuss.

Interviewee:	Date:
Questions:	Notes:
When to ask question, if conditional *Question: 1* Have you used the current sales tracking system? If so, how often?	*Answer* Yes, I ask for a report on my product line weekly. *Observations* Seemed anxious—may be overestimating usage frequency.
If yes, go to Question 2	
Question: 2 What do you like least about the system?	*Answer* Sales are shown in units, not dollars. *Observations* System can show sales in dollars, but user does not know this.

Figure: A typical Interview Guide

Choosing Interview Questions

- You need to decide what mix and sequence of open-ended and closed-ended questions you will use.
- **Open-ended questions** 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.
- An example is, “What would you say is the best thing about the information system you currently use to do your job?” or “List the three most frequently used menu options.” You must react quickly to answers and determine whether or not any follow-up questions are needed for clarification or elaboration. Sometimes body language will suggest that a user has given an incomplete answer or is reluctant to divulge (to make something secret known) some information; a follow-up question might yield additional insight.

- 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 (move) 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. Here is an example:
- Which of the following would you say is the one best thing about the information system you currently use to do your job (pick only one)?

a. Having easy access to all of the data you need

b. The system’s response time

c. The ability to access the system from remote locations

- 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. Closed-ended questions can also be an easy way to begin an interview and to determine which line of open-ended questions to pursue. You can include an “other” option to encourage the interviewee to add unanticipated 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 put his or her true opinion and perspective and that his or her idea will be considered equally with those of others. 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?”.
- 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 (be sure to ask permission first!). The answers may contain extremely important information for the project.
- **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 (a temporary failure) 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. 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 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 the interviewee know that there are many steps to the project and the perspectives of many people need to be considered, along with what is technically possible. 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. Find out what potential users of the system, users of other systems that might be affected by changes, managers and superiors, information systems staff who have experience with the current system, and others think the current problems and opportunities are and what new information services might better serve the organization.

Interviewing groups:

- One drawback to using interviews to collect systems requirements is the need for the analyst to reconcile (reconsider) apparent contradictions in the information collected. A series of interviews may turn up inconsistent information about the current system or its replacement.
- You must work through all of these inconsistencies to figure out what might be the most accurate representation of current and future systems. Such a process requires several follow-up phone calls and additional interviews. Catching important people in their offices is often difficult and frustrating, and scheduling new interviews may become very time consuming.
- 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. For example, one analyst may listen for data requirements while another notes the timing and triggering of key events. 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 (although the time commitment of the interviewees may be more of a concern).
- **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 (the combined power of a group of things when they are working together which is greater than the total power achieved by each working separately) also often occur.

- For example, the comments of one person might cause another person to say, “That reminds me of” or “I didn’t know that was a problem.” You can benefit from such a discussion as it helps you identify issues on which there is general agreement and areas where view s diverge widely.
- 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. Group interviews are at the core of the JAD process.

Nominal Group Technique:

- Many different techniques have been developed over the years to improve the process of working with groups.
- One of the more popular techniques for generating ideas among group members is called Nominal Group Technique (NGT).
- NGT is exactly what the name indicates—the individuals working together to solve a problem are a group in name only, or nominally. Group members may be gathered in the same room for NGT, but they all work alone for a period of time. Typically, group members make a written list of their ideas.
- At the end of the idea-generation time, group members pool their individual ideas under the guidance of a trained facilitator. Pooling usually involves having the facilitator ask each person in turn for an idea that has not been presented before. As the person reads the idea aloud, someone else writes down the idea on a blackboard or flip chart.
- After all of the ideas have been introduced, the facilitator will then ask for the group to openly discuss each idea, primarily for clarification.

- Once all of the ideas are understood by all of the participants, the facilitator will try to reduce the number of ideas the group will carry forward for additional consideration. There are many ways to reduce the number of ideas.
- The facilitator may ask participants to choose only a subset of ideas that they believe are important. Then the facilitator will go around the room, asking each person to read aloud an idea that is important to him or her that has not yet been identified by someone else.
- Or the facilitator may work with the group to identify and either eliminate or combine ideas that are very similar to others. At some point, the facilitator and the group end up with a tractable set of ideas, which can be further prioritized.
- In a requirements determination context, the ideas being sought in an NGT exercise would typically apply to problems with the existing system or ideas for new features in the system being developed.

Directly observing users

- For example, observation can cause people to change their normal operating behavior. Employees who know they are being observed may be nervous and make more mistakes than normal, may be careful to follow exact procedures they do not typically follow, and may work faster or slower than normal.
- Moreover, because observation typically cannot be continuous, you receive only a snapshot image of the person or task you observe, which may not include important events or activities. Because observation is very time consuming, you will not only observe for a limited time, but also a limited number of people and a limited number of sites.
- Again, observation yields only a small segment of data from a possibly vast variety of data sources. Exactly which people or sites to observe is a difficult selection problem. You want to pick both typical and atypical people and sites, and observe during normal and abnormal conditions and times to receive the richest possible data from observation.

Analyzing Procedures And Other Documents

- By examining existing system and organizational documentation, system analyst can find out details about current system and the organization. In documents analyst can find information, such as problem with existing systems, opportunities to meet new needs if only certain information or information processing were available, organizational direction that can influence information system requirements, and the reason why current systems are designed as they are etc.
- However, when analyzing those official documentations analyst should pay attention to the different between the systems described on the official documentation and practical systems in real world. For the reason of inadequacies (not enough or small in amount) of formal procedures, individual work habits and preferences, resistance to control, and other factors, the difference between so called formal system and informal system universally exists.

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 led a study of the 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 that requires 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, which may now be feasible and more desirable (e.g., data about a customer’s purchase of competitors’ products were not available when the current system was designed; these data are 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 is assigned exactly one sales department staff member as a primary contact if the customer has any questions)

Useful document: Written work procedure

- For an individual or work group.
- Describes how a particular job or task is performed.
- Includes data and information used and created in the process.

Figure 6-3 Example of a procedure

GUIDE FOR PREPARATION OF INVENTION DISCLOSURE
(See FACULTY and STAFF MANUALS for detailed Patent Policy and routing procedures.)

(1) DISCLOSE ONLY ONE INVENTION PER FORM.

(2) PREPARE COMPLETE DISCLOSURE.

The disclosure of your invention is adequate for patent purposes ONLY if it enables a person skilled in the art to understand the invention.

(3) CONSIDER THE FOLLOWING IN PREPARING A COMPLETE DISCLOSURE:

(a) All essential elements of the invention, their relationship to one another, and their mode of operation.

(b) Equivalents that can be substituted for any elements.

(c) List of features believed to be new.

(d) Advantages this invention has over the prior art.

(e) Whether the invention has been built and/or tested.

(4) PROVIDE APPROPRIATE ADDITIONAL MATERIAL.

Drawings and descriptive material should be provided as needed to clarify the disclosure. Each page of this material must be signed and dated by each inventor and properly witnessed. A copy of any current and/or planned publication relating to the invention should be included.

(4) INDICATE PRIOR KNOWLEDGE AND INFORMATION.

Pertinent publications, patents or previous devices, and related research or engineering activities should be identified.

(5) HAVE DISCLOSURE WITNESSED.

Persons other than coinventors should serve as witnesses and should sign each sheet of the disclosure only after reading and understanding the disclosure.

(7) FORWARD ORIGINAL PLUS ONE COPY (two copies if supported by grant/contract) TO VICE PRESIDENT FOR RESEARCH VIA DEPARTMENT HEAD AND DEAN.

3.1.4 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 about the current system, the organizational area requesting the new system, and what the new system should be like.
- In this section, you will learn about several contemporary information gathering techniques for analysis (listed in Table 6-5): JAD, CASE tools to support JAD, and prototyping.
- As we said earlier, these techniques can support effective information collection and structuring while reducing the amount of time required for analysis.

TABLE 6-5 Contemporary Methods for Collecting System Requirements

- Bringing session users, sponsors, analysts, and others together in a JAD session to discuss and review system requirements
- Using *CASE tools* during a JAD to analyze current systems to discover requirements that will meet changing business conditions
- Iteratively developing system *prototypes* that refine the understanding of system requirements in concrete terms by showing working versions of system features

Joint Application Design (JAD)

- A structured process in which users, managers, and analysts work together for several days in a series of intensive meetings to specify or review system requirements.
- **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 in the analysis phase is to collect systems requirements simultaneously from the key people involved with the system. The result is an intense (extreme and forceful or (of a feeling) very strong) and structured, but highly effective, process.
- 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.

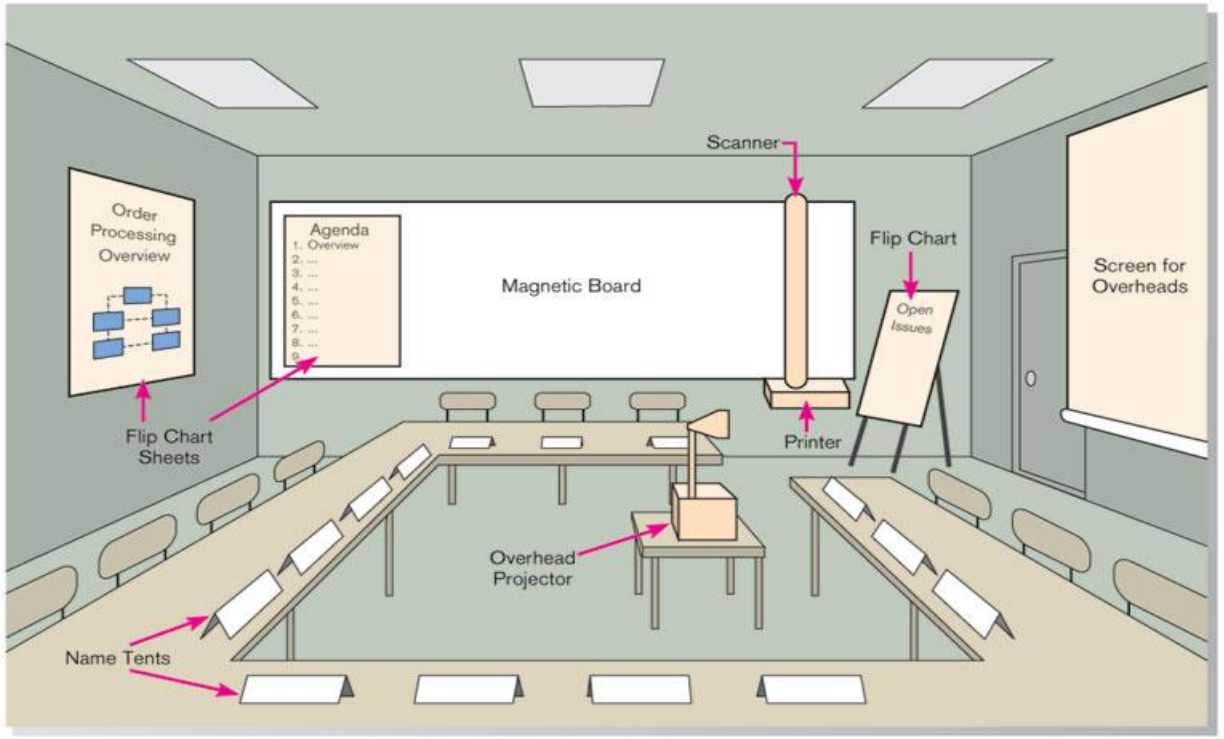
- JAD sessions are usually conducted at a location other than the place where the people involved normally work. The idea behind such a practice is to keep participants away from as many distractions as possible so that they can concentrate on systems analysis. A JAD may last anywhere from four hours 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. 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 (manage) 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:** As a major undertaking due to its expense, a 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. This is usually done on a laptop. Notes may be taken using a word processor, or notes and diagrams may be entered directly into a CASE tool.

- **IS staff:** Besides systems analysts, other information systems (IS) staff, such as programmers, database analysts, IS planners, and data center personnel, 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 horseshoe-shaped tables, as shown in Figure 6-6. These rooms are typically equipped with whiteboards. Other audiovisual tools may be used, such as magnetic symbols that can be easily rearranged on a whiteboard, flip charts, and computer-generated displays.

Figure 6-6 Illustration of the typical room layout for a JAD



Source: Adapted from Wood and Silver, 1995.

- 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, 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. Depending on the exact purpose of the JAD, analysts may also walk away from the JAD with some detailed information on what is desired of the replacement system.

- **CASE Tools During JAD**: For requirements determination and structuring, the most useful CASE tools are used for diagramming and form and report generation.
- Some observers advocate using CASE tools during JADs (Lucas, 1993). Running a CASE tool during a JAD enables analysts to enter system models directly into a CASE tool, providing consistency and reliability in the joint model-building process.
- The CASE tool captures system requirements in a more flexible and useful way than can a scribe or an analysis team making notes. Further, the CASE tool can be used to project menu, display, and report designs, so users can directly observe old and new designs and evaluate their usefulness for the analysis team.

Advantages of JAD

- JAD allows you to resolve difficulties more simply and produce better, error-free software.
- The joint collaboration between the company and the clients lowers all risks.
- JAD reduces costs and time needed for project development.
- Well-defined requirements improve system quality.
- Due to the close communication, progress is faster.
- JAD encourages the team to push each other to work faster and deliver on time.

Disadvantages of JAD

- Different opinions within the team make it difficult to align goals and maintain focus
- Depending on the size of the project , JAD may require a significant time commitment.

Using Prototyping During Requirements Determination

- **Prototyping:** An iterative process of systems development in which requirements are converted to a working system that is continually revised through close collaboration between an analyst and users.
- Prototyping will enable you to quickly convert basic requirements into a working, though limited, version of the desired information system. The prototype will then be viewed and tested by the user. Typically, seeing verbal descriptions of requirements converted into a physical system will prompt the user to modify existing requirements and generate new ones.
- For example, in the initial interviews, a user might have said that he wanted all relevant utility billing information (e.g., the client's name and address, the service record, and payment history) on a single computer display form. 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 to have the information 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.

- As the prototype changes through each iteration, more and more of the design specifications for the system are captured in the prototype. The prototype can then serve as the basis for the production system in a process called **evolutionary prototyping.**
- Alternatively, the prototype can serve only as a model, which is then used as a reference for the construction of the actual system. In this process, called **throwaway prototyping**, the prototype is discarded after it has been used.

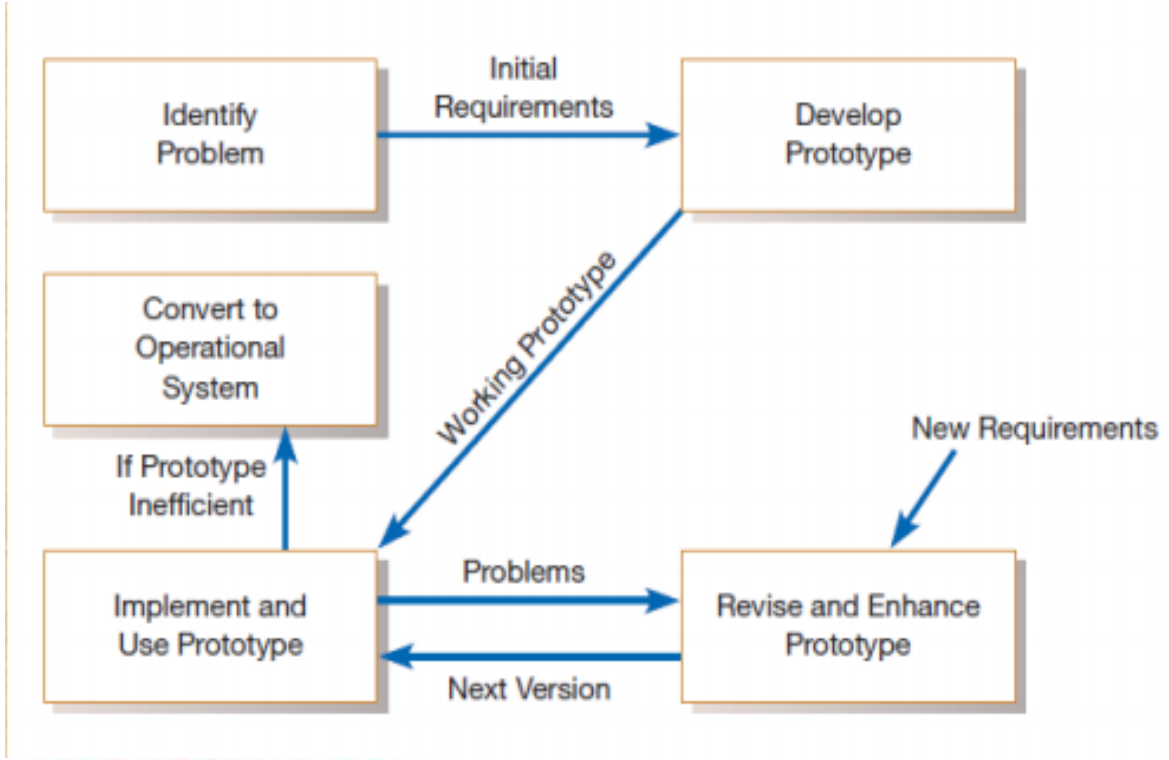


FIGURE 6-7
The prototyping methodology
(*Source:* Based on "Prototyping: The New Paradigm for Systems Development," by J. D. Naumann and A. M. Jenkins, *MIS Quarterly* 6(3): 29–44.)

Evolutionary Prototyping

- In evolutionary prototyping, you begin by modeling parts of the target system and, if the prototyping process is successful, you evolve the rest of the system from those parts (McConnell, 1996). 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 (Figure 6-8).
- Systems must be designed to support scalability, multiuser support, and multiplatform support. Few of these design specifications will be coded into a prototype. Further, as much as 90 percent of a system's functioning is devoted to handling exceptional cases (McConnell, 1996). Prototypes are designed to handle only the typical cases, so exception handling must be added to the prototype as it is converted to the production system. Clearly, the prototype captures only part of the system requirements.

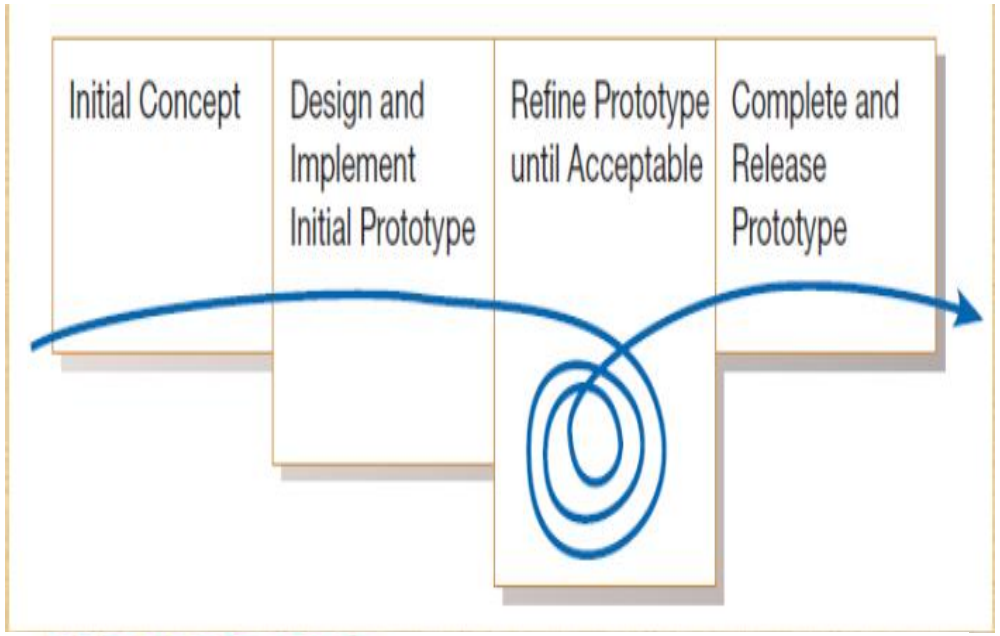


FIGURE 6-8
McConnell's evolutionary prototyping model

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 or interface characteristics. 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.

3.1.5 Radical Methods For Determining System Requirements

- The overall process by which current methods are replaced with radically new methods is generally referred to as **business process reengineering (BPR).**
- **Business Process Reengineering (BPR):** search for, and implementation of radical change in business processes to achieve breakthrough improvements in products and services.
- 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 US tour, you discover that incrementally improving your putting, driving accuracy, and sand shots will help, but the new competitive environment is simply not suited to your style of the game.

- 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 the press. If you are good enough, you may survive, but without reengineering, you will never be 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 (Dobyns and Crawford-Mason, 1991). Organizations realize that creatively using information technologies can yield significant improvements in most business processes.

Identifying Processes To Reengineer

- 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. 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.

- Three questions to identify activities for radical change are:
- How important is the activity to delivering an outcome?
- How feasible is changing the activity?
- How dysfunctional is the activity?
- The answers to these questions provide guidance for selecting which activities to change.
- Those activities deemed important, changeable, yet functional, are primary candidates for alteration.

Disruptive technologies

- Once key business processes and activities have been identified, information technologies must be applied to radically improve business processes.
- Disruptive technologies: are technologies that enable the breaking of long-held business rules that inhibit organizations from making radical business changes.
- To do this, Hammer and Champy (1993) 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 the power of new technologies and think of innovative (using new methods or ideas) ways to alter the way work is done.
- This is contrary to deductive thinking, where problems are first identified and solutions are then formulated.

3.1.6 Requirements Management Tools

- Organizations and developers have always been looking for more effective and creative ways to create and maintain requirements documents. One method that has been developed is computer-based requirements management tools.
- These tools make it easier for analyst to keep requirements, current documents and to define links between different parts of the overall specifications package. Requirements management tools are typically designed to work with many of the standards now available for requirements specification such as the Unified Modeling Language 2.0, System modeling language, Business process modeling notation etc.
- Requirements management tools is best to traditional, planning based systems development approaches. They do not work well with agile methodologies which tend to employ different approaches to requirements gathering.

3.1.7 Requirement Determination Using Agile Methodologies

- Three techniques are presented in this section.
- The **first is continual user involvement** in the development process, a technique that works especially well with small and dedicated development teams.
- The **second approach is a JAD-**like process called Agile Usage-Centered Design.
- The **third approach is the Planning Game**, which was developed as part of eXtreme Programming.

(1) Continual User Involvement

- We read about the criticisms of the traditional waterfall SDLC. One of those criticisms was that the waterfall SDLC allowed users to be involved in the development process only in the early stages of analysis.
- Once requirements had been gathered from them, the users were not involved again in the process until the system was being installed and they were asked to sign off on it.
- Typically, by the time the users saw the system again, it was nothing like what they had imagined. The system most likely did not adequately address user needs.
- One approach to the problem of limited user involvement is to involve the users continually, throughout the entire analysis and design process. Such an approach works best when development can follow the analysis–design–code–test cycle favored by the Agile Methodologies (Figure 6-9), because the user can provide information on requirements and then watch and evaluate as those requirements are designed, coded, and tested.

- This iterative process can continue through several cycles, until most of the major functionality of the system has been developed. Extensive involvement users in the analysis and design process is a key part of many Agile approaches, but it was also a key part of Rapid Application Development.

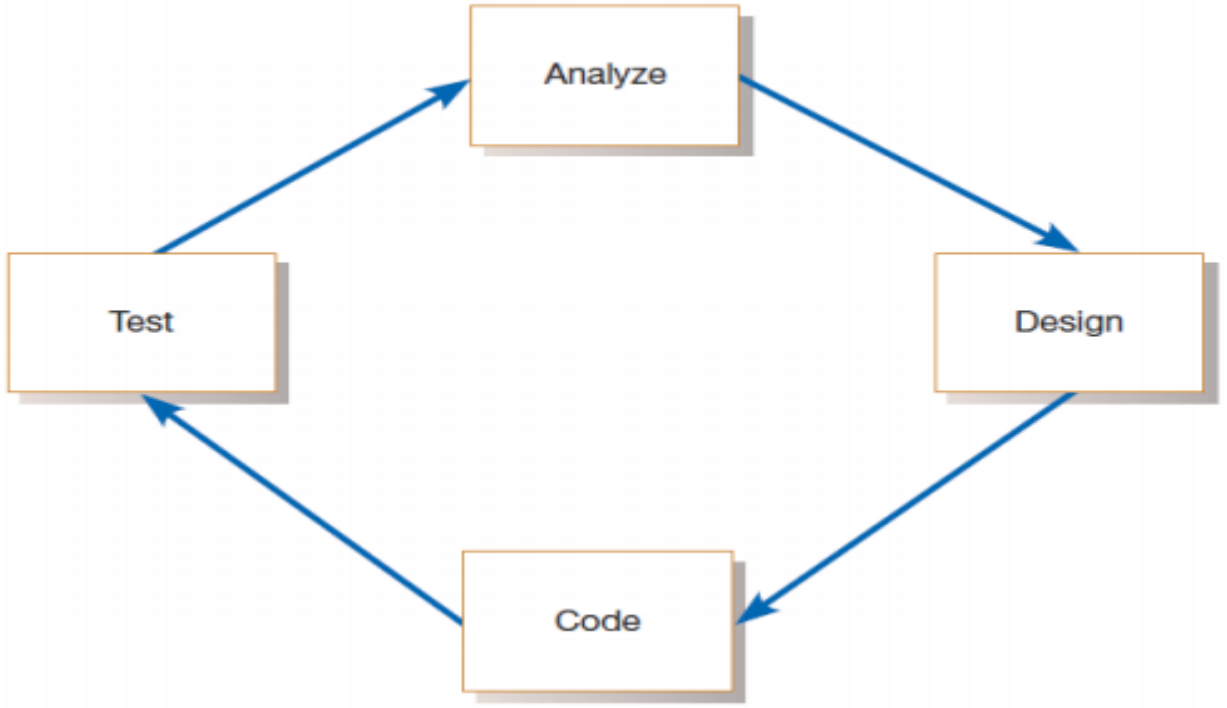


FIGURE 6-9
The iterative analysis–design–code–test cycle

- Continual user involvement was a key aspect of the success of Boeing’s Wire Design and Wire Install system for the 757 aircraft (Bedoll, 2003).
- The system was intended to support engineers who customize plane configurations for customers, allowing them to analyze all 50,000 wires that can possibly be installed on a 757. A previous attempt at building a similar system took over three years, and the resulting system was never used. The second attempt, relying on Agile Methodologies, resulted in a system that was in production after only six weeks.
- One of the keys to success was a user liaison (communication between people who work with each other) who spent half of his time with the small development team and half with the other end users. In addition to following the analysis–design–code–test cycle, the team also had weekly production releases.
- The user liaison was involved every step of the way. Obviously, for such a requirements determination to succeed, the user who works with the development team must be very knowledgeable, but he or she must also be in a position to give up his or her normal business responsibilities in order to become heavily involved in the system’s development.

(2) Agile Usage-Centered Design

- Continual user involvement in systems development is an excellent way to ensure that requirements are captured accurately and immediately implemented in system design. However, such constant interaction works best when the development team is small, as was the case in the Boeing example.
- Also, it is not always possible to have continual access to users for the duration of a development project. Thus, Agile developers have come up with other means for effectively involving users in the requirements determination process. One such method is called Agile Usage-Centered Design, originally developed by Larry Constantine (2002) and adapted for Agile Methodologies by Jeff Patton (2002).

Agile Usage-Centered Design Steps

Gather group of programmers, analysts, users, testers, facilitator.

- Document complaints of current system.
- Determine important user roles.
- Determine, prioritize, and describe tasks for each user role.
- Group similar tasks into interaction contexts.
- Associate each interaction context with a user interface for the system, and prototype the interaction context.
- Step through and modify the prototype.

(3) The Planning Game From eXtreme Programming

- eXtreme Programming is an approach to software development put together by Kent Beck (Beck and Andres, 2004).
- It is distinguished by its short cycles, its incremental planning approach, its focus on automated tests written by programmers and customers to monitor the process of development, and its reliance on an evolutionary approach to development that lasts throughout the lifetime of the system.
- One of the key emphases of eXtreme Programming is its use of two-person programming teams and having a customer on-site during the development process. The relevant parts of eXtreme Programming that relate to requirements determination are

(1) how planning, analysis, design, and construction are all fused together into a single phase of activity and

(2) its unique way of capturing and presenting system requirements and design specifications. All phases of the life cycle converge into a series of activities based on the basic processes of coding, testing, listening, and designing.

3.2 System Process Requirements

3.2.1 Introduction,

3.2.2 Process Modeling,

3.2.3 Data Flow Diagramming Mechanics,

3.2.4 Using DFD In The Analysis Process,

3.2.5 Modeling Logic With Decision Tables

3.2 System Process Requirements

3.2.1 Introduction

- We will be focusing on one tool that is used to coherently represent the information gathered as part of **requirements determination—data flow diagrams.**
- 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**.
- **Decision tables** allow you to represent the conditional logic that is part of some data flow diagram processes.

3.2.2 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).** DFDs, the traditional process modeling technique of structured analysis and design and one of the techniques most frequently used today for process modeling.
- Data-flow diagramming is one of several structured analysis techniques used to increase software development productivity.

Modeling a system's Process for structured Analysis

- As Figure: shows, the analysis phase of the systems development life cycle has two subphases : requirements determination and requirements structuring. 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.

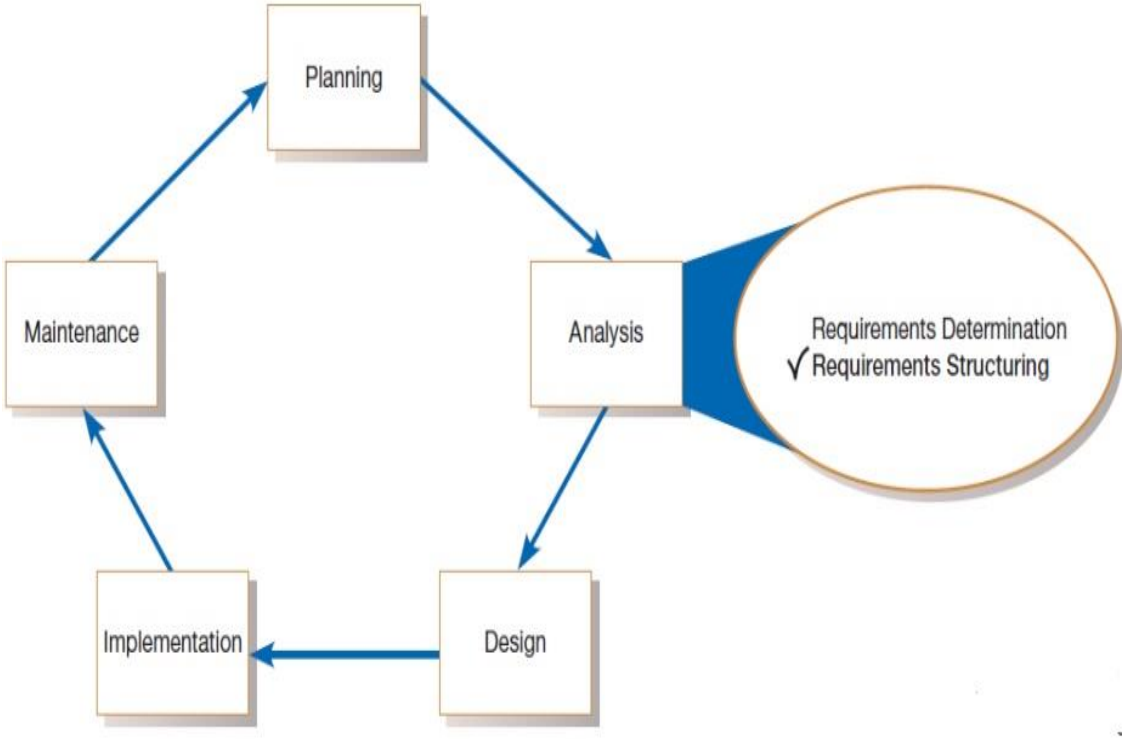


Figure:
Systems development life cycle with the analysis phase highlighted

Deliverables and Outcomes

- In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated DFDs. Table 7-1 provides a more detailed list of the deliverables that result when DFDs are used to study and document a system's processes.
- 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.
- Finally, entries for all of the objects included in all of the diagrams are included in the project dictionary or CASE repository.

TABLE 7-1 Deliverables for Process Modeling

1. Context DFD
2. DFDs of the system (adequately decomposed)
3. Thorough descriptions of each DFD component

3.2.3 Data Flow Diagramming Mechanics

- DFDs are versatile diagramming tools. With only four symbols, you can use DFDs to represent both physical and logical information systems. DFDs are not as good as flowcharts for depicting (to represent or show something in a picture or story) the details of physical systems.
- There are two different standard sets of DFD symbols; each set consists of four symbols that represent the same things: data flows, data stores, processes, and sources/sinks (or external entities).
- A **data store** is data at rest. A data store may represent one of many different physical locations for data; for example, a file folder, one or more computer-based file(s), or a notebook. A data store might contain data about customers, students, customer orders, or supplier invoices.
- A **process** is the work or actions performed on data so that they are transformed, stored, or distributed. When modeling the data processing of a system, it does not matter whether a process is performed manually or by a computer

- Finally, a **source/sink** is the origin and/or destination of the data. Sources/sinks are sometimes referred to as external entities because they are outside the system. Once processed, data or information leave the system and go to some other place. Sources and sinks are outside the system we are studying.
- The symbols for each set of DFD conventions are presented in Figure 7-2. In both conventions, a data flow is represented as an arrow. The arrow is labeled with a meaningful name for the data in motion; for example, Customer Order, Sales Receipt, or Paycheck.

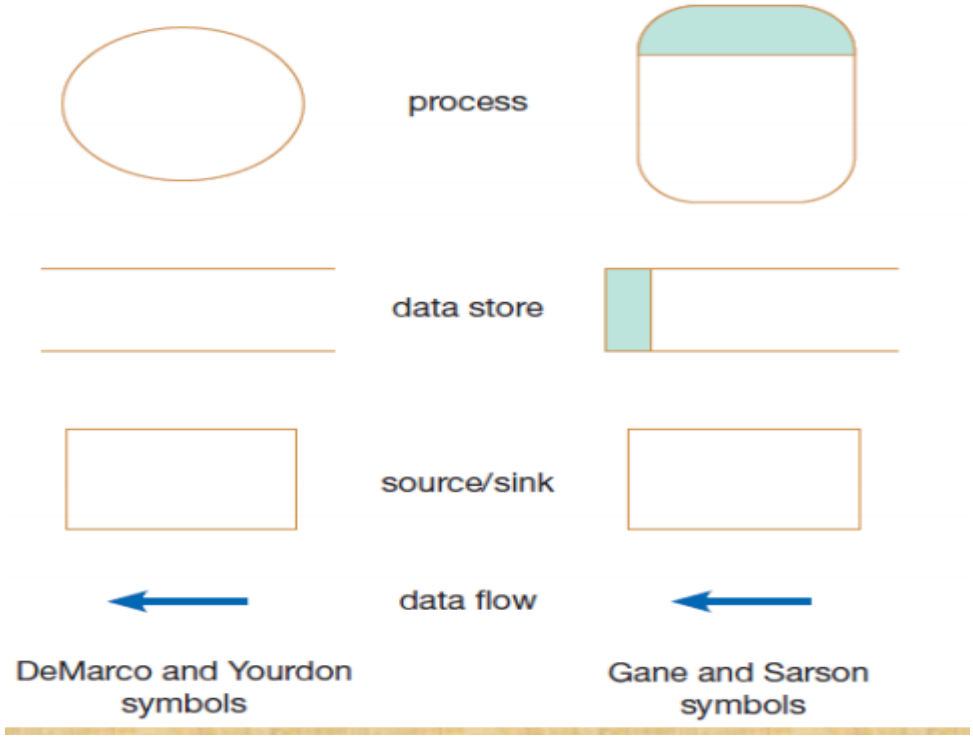
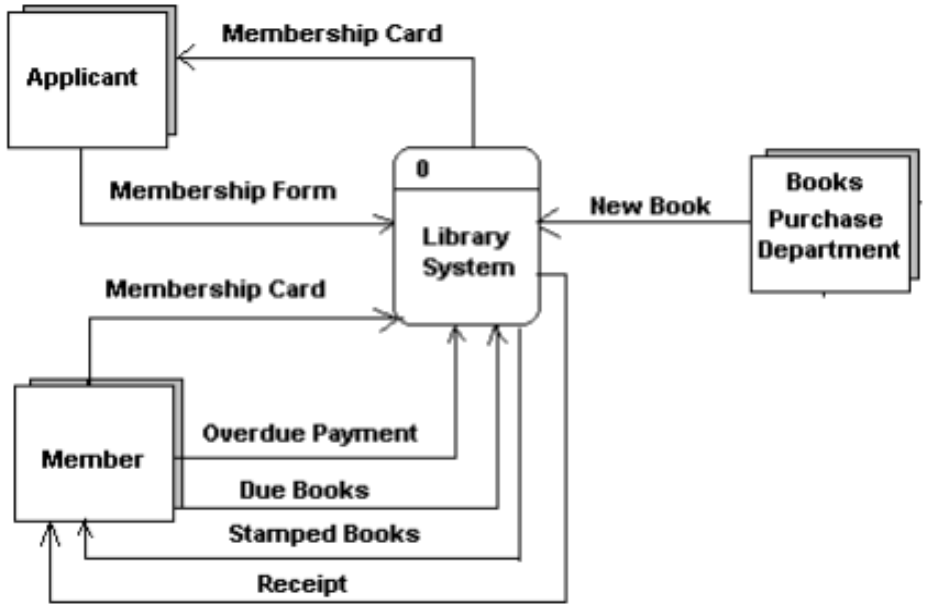


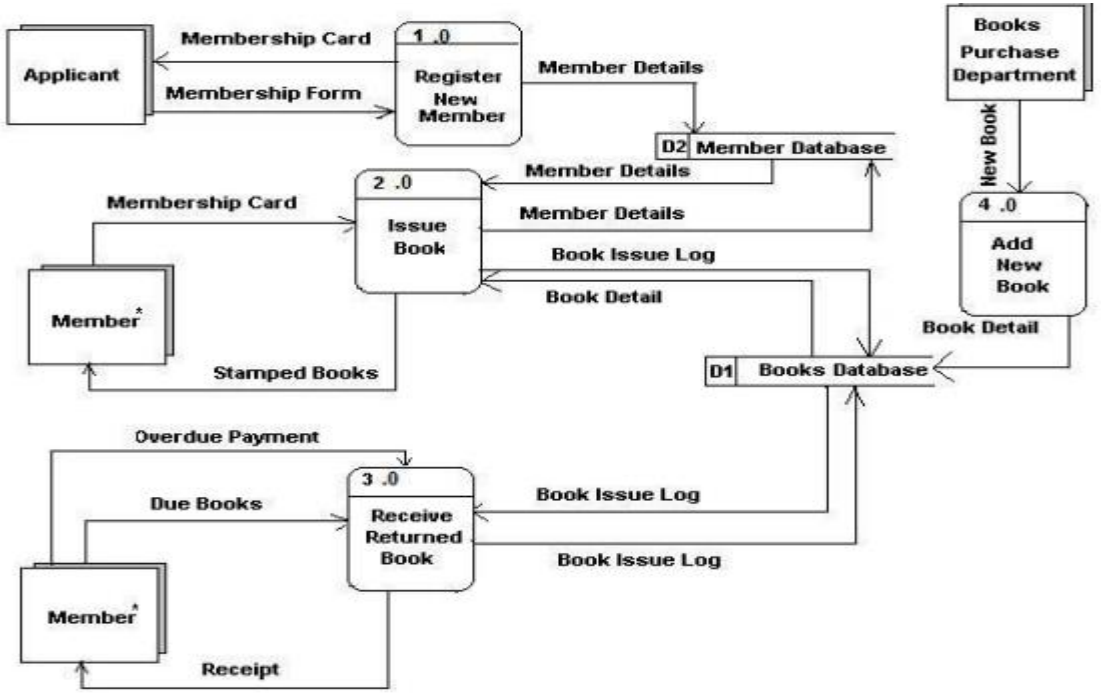
FIGURE 7-2
Comparison of DeMarco and Yourdon and Gane and Sarson DFD symbol sets

- The name represents the aggregation of all the individual elements of data moving as part of one packet, that is, all the data moving together at the same time.
- Sources/Sinks are always outside the information system and define the boundaries of the system. Data must originate outside a system from one or more sources, and the system must produce information to one or more sinks (these are principles of open systems, and almost every information system is an open system).
- If any data processing takes place inside the source/sink, it is of no interest because this processing takes place outside the system we are diagramming.
- The figure above 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.



A sample Context Diagram



DFD

Developing DFD

- The information system is represented as a DFD in Figure 7-4. The highest-level view of this system, shown in the figure, is called a **context diagram**. You will notice that this context diagram contains only one process, no data stores, four data flows, and three sources/sinks. The single process, **labeled 0,** represents the entire system; all context diagrams have only one process, labeled 0. The sources/sinks represent the environmental boundaries of the system. Because the data stores of the system are conceptually inside one process, data stores do not appear on a context diagram.

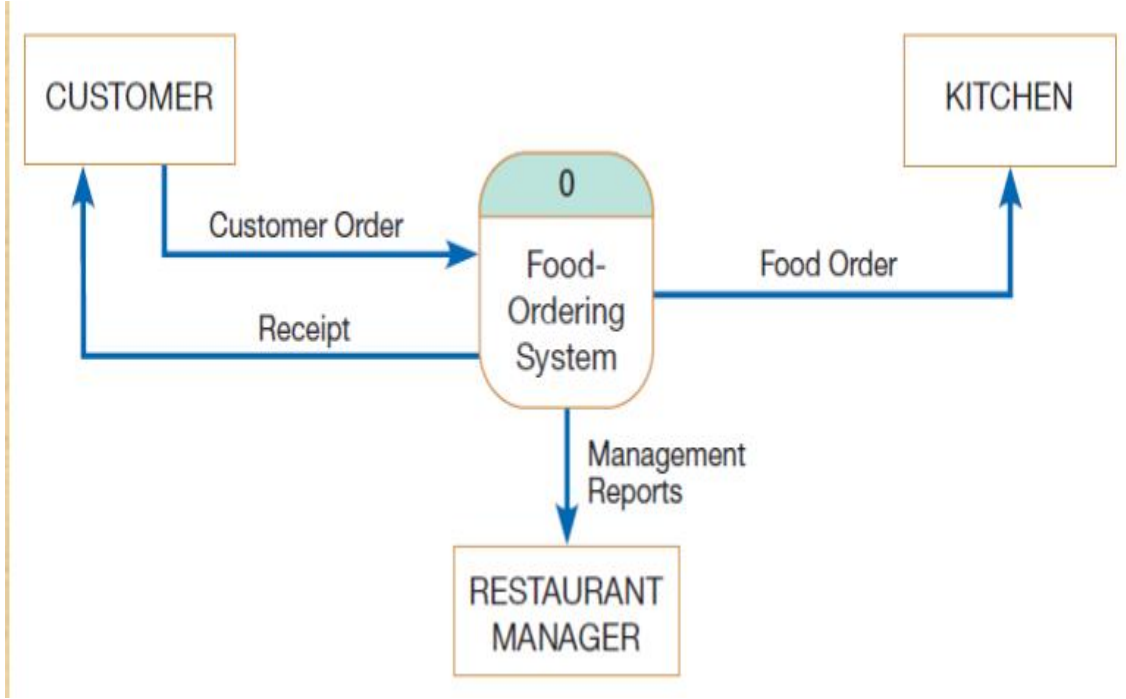


FIGURE 7-4
Context diagram of Hoosier Burger's food-ordering system

- As you can see in Figure 7-5, we have identified four separate processes. The main processes represent the major functions of the system, and these major functions correspond to actions such as the following:

1. Capturing data from different sources (e.g., Process 1.0)
2. Maintaining data stores (e.g., Processes 2.0 and 3.0)
3. Producing and distributing data to different sinks (e.g., Process 4.0)
4. High-level descriptions of data transformation operations (e.g., Process 1.0)

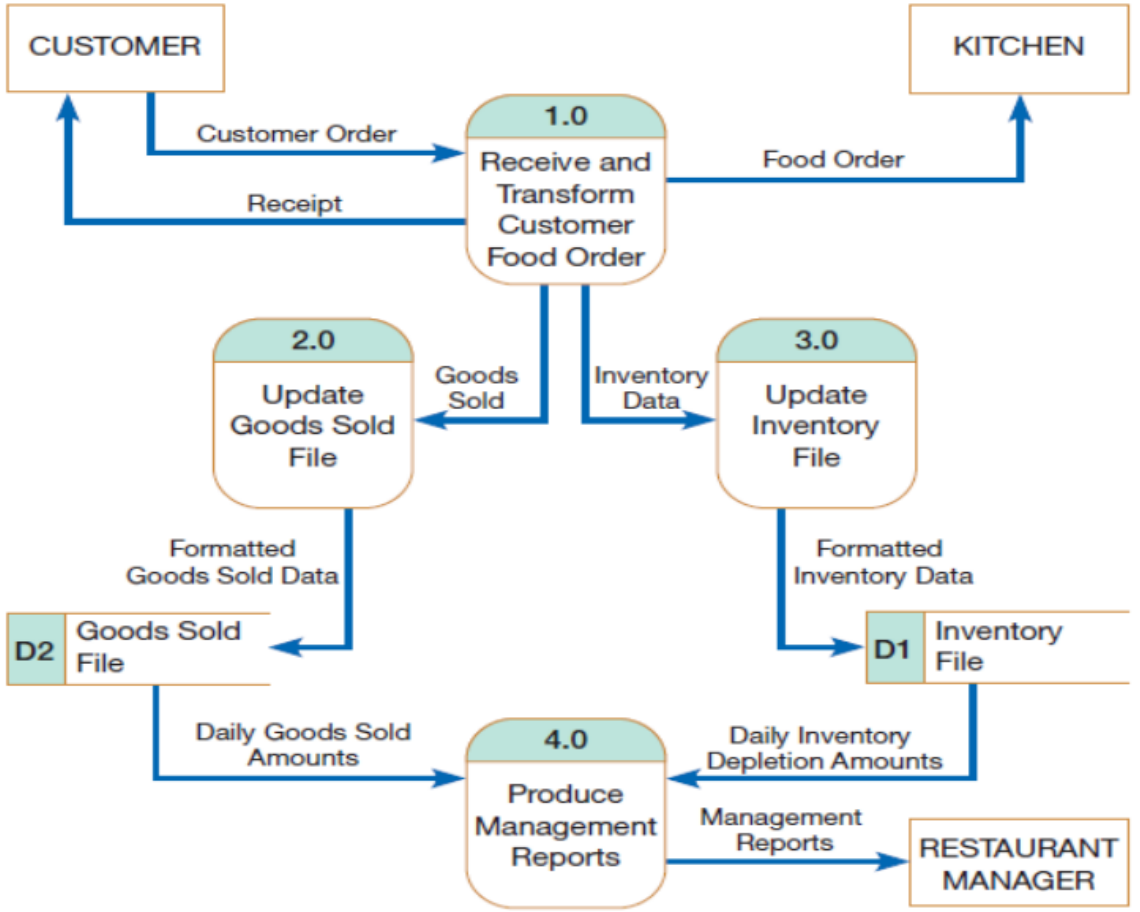


FIGURE 7-5
Level-0 DFD of Hoosier Burger's food-ordering system

- We see that the system begins with an order from a customer, as was the case with the context diagram. In the first process, labeled 1.0, we see that the customer order is processed. The result is four streams, or flows, of data:

(1) the food order is transmitted to the kitchen,

(2) the customer order is transformed into a list of goods sold,

(3) the customer order is transformed into inventory data, and

(4) the process generates a receipt for the customer.

- Notice that the sources/sinks are the same in the context diagram and in this diagram: the customer, the kitchen, and the restaurant's manager. This diagram is called a level-0 diagram because it represents the primary individual processes in the system at the highest possible level. Each process has a number that ends in .0 (corresponding to the level number of the DFD).

- Two of the data flows generated by the first process, Receive and Transform Customer Food Order, go to external entities, so we no longer have to worry about them. We are not concerned about what happens outside our system. Let’s trace the flow of the data represented in the other two data flows.
- First, the data labeled Goods Sold go to Process 2.0, Update Goods Sold File. The output for this process is labeled Formatted Goods Sold Data. This output updates a data store labeled Goods Sold File. If the customer order was for two cheeseburgers, one order of fries, and a large soft drink, each of these categories of goods sold in the data store would be incremented appropriately. The Daily Goods Sold Amounts are then used as input to Process 4.0, Produce Management Reports. Similarly, the remaining data flow generated by Process 1.0, Inventory Data, serves as input for Process 3.0, Update Inventory File.
- This process updates the Inventory File data store, based on the inventory that would have been used to create the customer order. For example, an order of two cheeseburgers would mean that Hoosier Burger now has two fewer hamburger patties, two fewer burger buns, and four fewer slices of American cheese.

- The Daily Inventory Depletion Amounts are then used as input to Process 4.0. The data flow leaving Process 4.0, Management Reports, goes to the sink Restaurant Manager.
- Figure 7-5 illustrates several important concepts about information movement. Consider the data flow Inventory Data moving from Process 1.0 to Process 3.0. We know from this diagram that Process 1.0 produces this data flow and that Process 3.0 receives it. However, we do not know the timing of when this data flow is produced, how frequently it is produced, or what volume of data is sent. Thus, this DFD hides many physical characteristics of the system it describes. We do know, however, that this data flow is needed by Process 3.0 and that Process 1.0 provides these needed data.

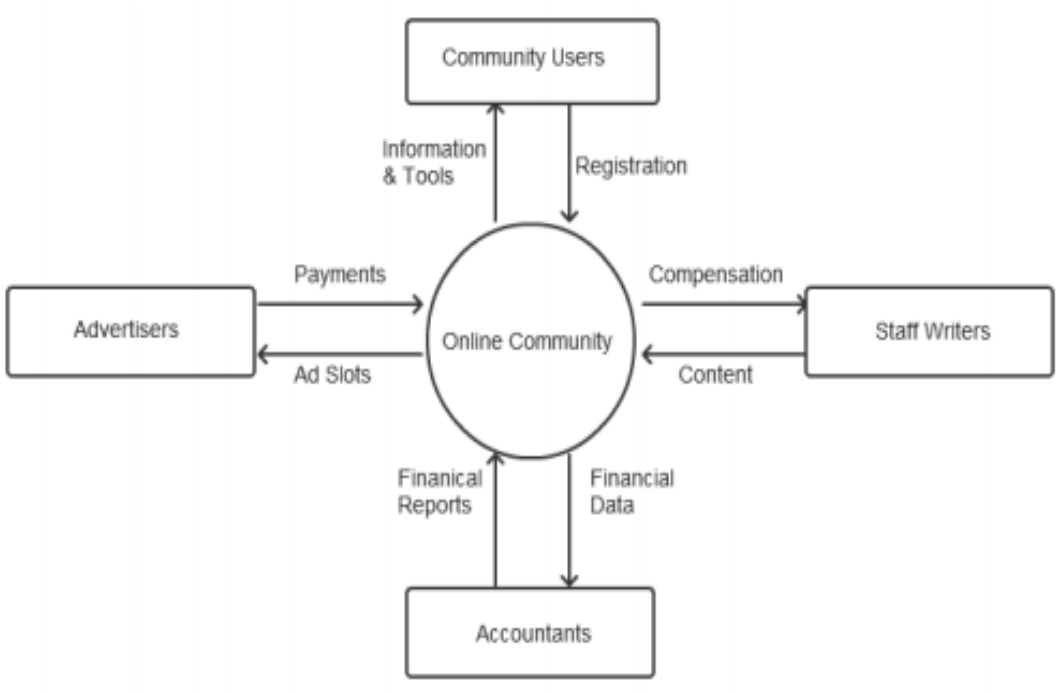
- Also implied by the Inventory Data flow is that whenever Process 1.0 produces this flow, Process 3.0 must be ready to accept it. Thus, Processes 1.0 and 3.0 are coupled with each other.
- In contrast, consider the link between Process 2.0 and Process 4.0. The output from Process 2.0, Formatted Goods Sold Data, is placed in a data store and, later, when Process 4.0 needs such data, it reads Daily Goods Sold Amounts from this data store. In this case, Processes 2.0 and 4.0 are decoupled by placing a buffer, a data store, between them. Now, each of these processes can work at their own pace, and Process 4.0 does not have to be ready to accept input at any time. Further, the Goods Sold File becomes a data resource that other processes could potentially draw upon for data.

- A **System Context Diagram (SCD) in** software engineering and systems engineering is a diagram that represents the actors outside a system that could interact with that system. This diagram is the highest level view of a system. It is similar to a Block diagram. SCDs show a system, often software-based, as a whole and its inputs and outputs from/to external factors.
- A System Context Diagram (SCD) in software engineering and systems engineering is a diagram that represents the actors outside a system that could interact with that system. This diagram is the highest level view of a system. It is similar to a Block diagram. SCDs show a system, often software-based, as a whole and its inputs and outputs from/to external factors.
- The Context Diagram shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores, etc.).
- Another name for a Context Diagram is **a Context-Level Data-Flow Diagram or a Level-0 Data Flow Diagram**. Since a Context Diagram is a specialized version of Data-Flow Diagram, understanding a bit about Data-Flow Diagrams can be helpful.

- Each DFD may show a number of processes with data flowing into and out of each process. If there is a need to show more detail within a particular process, the process is decomposed into a number of smaller processes in a lower level DFD. In this way, the Content Diagram or Context-Level DFD is labeled a “Level-0 DFD” while the next level of decomposition is labeled a “Level-1 DFD”, the next is labeled a “Level-2 DFD”, and so on.
- Context Diagrams and Data-Flow Diagrams were created for systems analysis and design. But like many analysis tools they have been leveraged for other purposes. For example, they can also be leveraged to capture and communicate the interactions and flow of data between business processes. So, they don’t have to be restricted to systems analysis.
- A Context Diagram (and a DFD for that matter) provides no information about the timing, sequencing, or synchronization of processes such as which processes occur in sequence or in parallel. Therefore it should not be confused with a flowchart or process flow which can show these things.

Some of the benefits of a Context Diagram are:

- Shows the scope and boundaries of a system at a glance including the other systems that interface with it
- No technical knowledge is assumed or required to understand the diagram
- Easy to draw and amend due to its limited notation
- Easy to expand by adding different levels of DFDs
- Can benefit a wide audience including stakeholders, business analyst, data analysts, developers



A sample Context Diagram

Data Flow Diagramming Rules

- There are two DFD guideline s that apply:
- The inputs to a process are different from the outputs of that process.

Processes purpose is to transform inputs into outputs.

- Objects on a DFD have unique names. Every process has a unique name.

TABLE 7-2 Rules Governing Data Flow Diagramming

Process:

A. No process can have only outputs. It would be making data from nothing (a miracle). If an object has only outputs, then it must be a source.

B. No process can have only inputs (a black hole). If an object has only inputs, then it must be a sink.

C. A process has a verb phrase label.

Data Store:

D. Data cannot move directly from one data store to another data store. Data must be moved by a process.

E. 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.

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

G. A data store has a noun phrase label.

Source/Sink:

H. Data cannot move directly from a source to a sink. It 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.

I. A source/sink has a noun phrase label.

Data Flow:

J. 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 these happen at different times.

K. 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 (this usually indicates different copies of the same data going to different locations).

L. 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.

M. 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.

N. A data flow to a data store means update (delete or change).

O. A data flow from a data store means retrieve or use.

P. 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.

(*Source:* Based on Celko, 1987.)

•Decomposition of DFDs

- In the earlier example of Hoosier Burger's food-ordering system, we started with a high-level context diagram. Upon thinking more about the system, we saw that the larger system consisted of four processes. The act of going from a single system to four component processes 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. This process creates a set of hierarchically related charts in which one process on a given chart is explained in greater detail on another chart. For the Hoosier Burger system, we broke down, or decomposed, the larger system into four processes.
- Each resulting process (or subsystem) is also a candidate for decomposition. Each process may consist of several sub processes. Each sub process may also be broken down into smaller units.
- Decomposition continues until you have reached the point at which no sub process can logically be broken down any further. The lowest level of a DFD is called a **primitive DFD.**

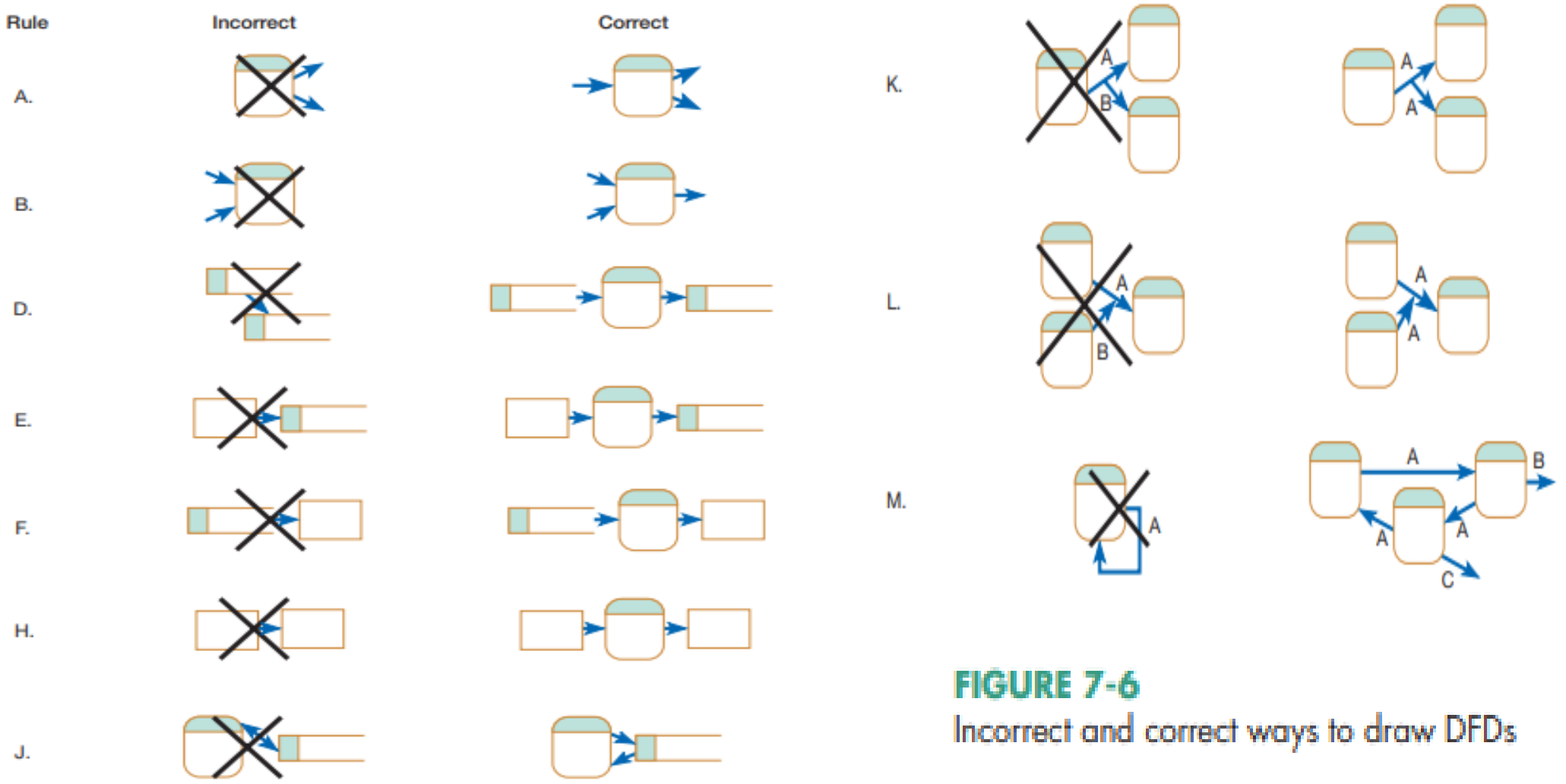
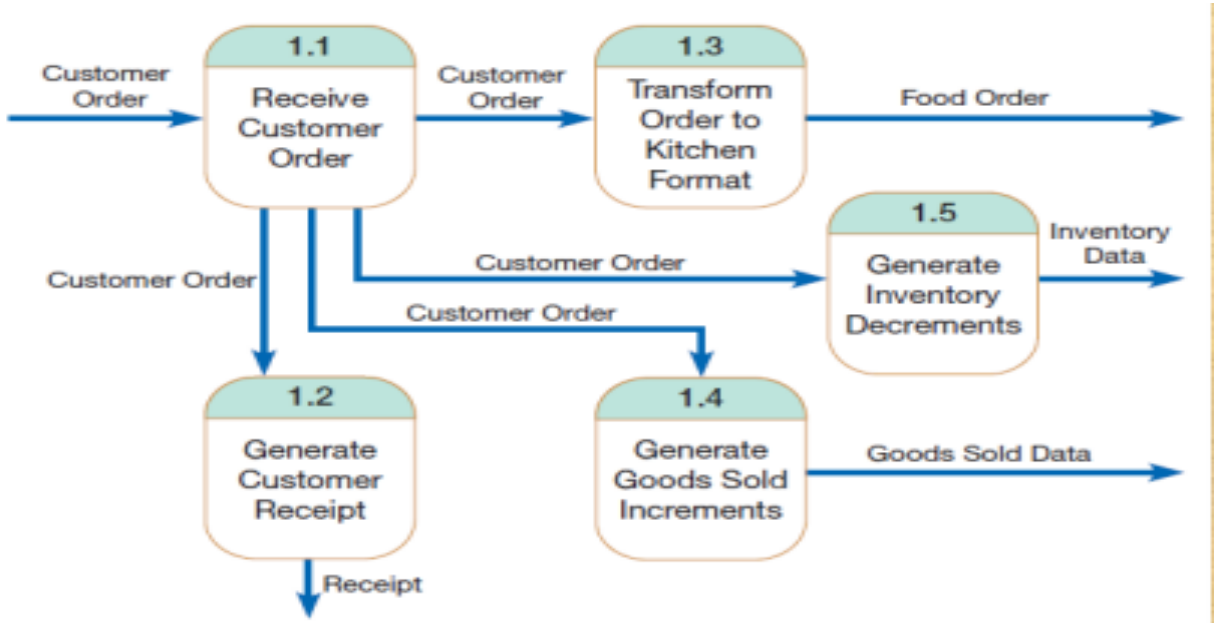


FIGURE 7-6
Incorrect and correct ways to draw DFDs

FIGURE 7-7
Level-1 diagram showing the decomposition of Process 1.0 from the level-0 diagram for Hoosier Burger's food-ordering system



Note that each of the five Processes in Figure 7-7 is labeled as a sub process of 1.0

Process 1.0: Process 1.1, Process 1.2, and so on. Also note that, just as with the other DFDs we have looked at, each of the processes and data flows is named. You will also notice that no sources or sinks are represented. Although you may include sources and sinks, the context and level-0 diagrams show the sources and sinks. The DFD in Figure 7-7 is called a level-1 diagram. If we should decide to decompose Processes 2.0, 3.0, or 4.0 in a similar manner, the DFDs we would create would also be level-1 diagrams. In general, a level-n diagram is a DFD that is generated from n nested decompositions from a level-0 diagram.

- Processes 2.0 and 3.0 perform similar functions in that they both use data input to update data stores. Because updating a data store is a singular logical function, neither of these processes needs to be decomposed further. We can, however, decompose Process 4.0, Produce Management Reports, into at least three sub processes: Access Goods Sold and Inventory Data, Aggregate Goods Sold and Inventory Data, and Prepare Management Reports. The decomposition of Process 4.0 is shown in the level-1 diagram of Figure 7-8.

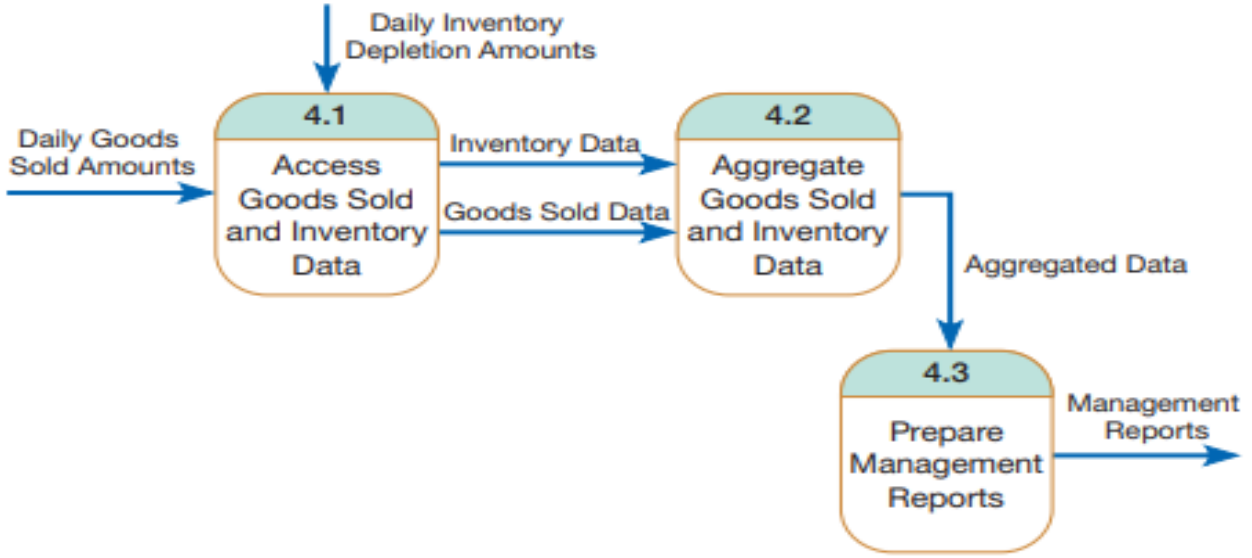
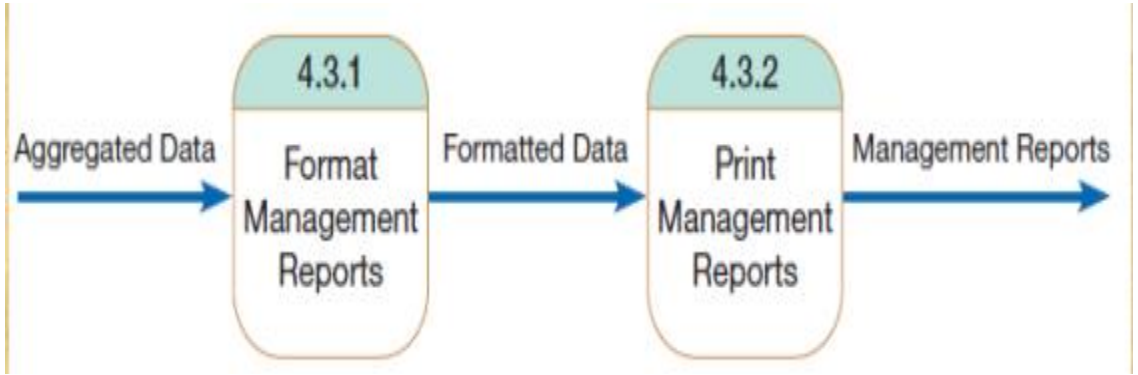


FIGURE 7-8
Level-1 diagram showing the decomposition of Process 4.0 from the level-0 diagram for Hoosier Burger's food-ordering system

- Each level-1, -2, or -n DFD represents one process on a level-n-1 DFD; each DFD should be on a separate page.

FIGURE 7-9
Level-2 diagram showing the decomposition of Process 4.3 from the level-1 diagram for Process 4.0 for Hoosier Burger's food-ordering system



No DFD should have more than about seven processes because too many processes will make the diagram too crowded and difficult to understand. Typically, process names begin with an action verb, such as Receive, Calculate, Transform, Generate, or Produce. Process names often are the same as the verbs used in many computer programming languages. Example process names include Merge, Sort, Read, Write, and Print.

Balancing DFDs

- **Conservation Principle:** conserve inputs and outputs to a process at the next level of decomposition.
- **Balancing:** conservation of inputs and outputs to a data flow diagram process when that process is decomposed to a lower level.
- **Balanced means:**

  - Number of inputs to lower level DFD equals number of inputs to associated process of higher level DFD

  - Number of outputs to lower level DFD equals number of outputs to associated process of higher level DFD

- 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.

Figure 7-10a An unbalanced set of data flow diagrams - Context diagram

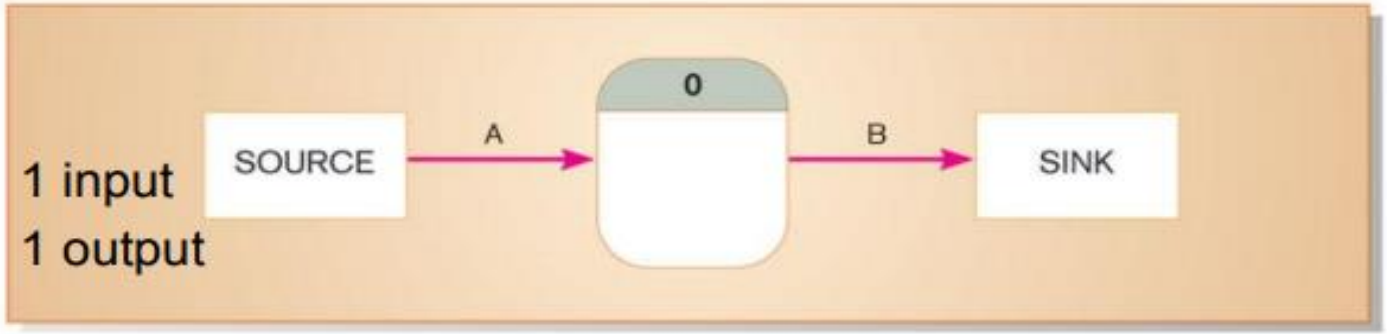
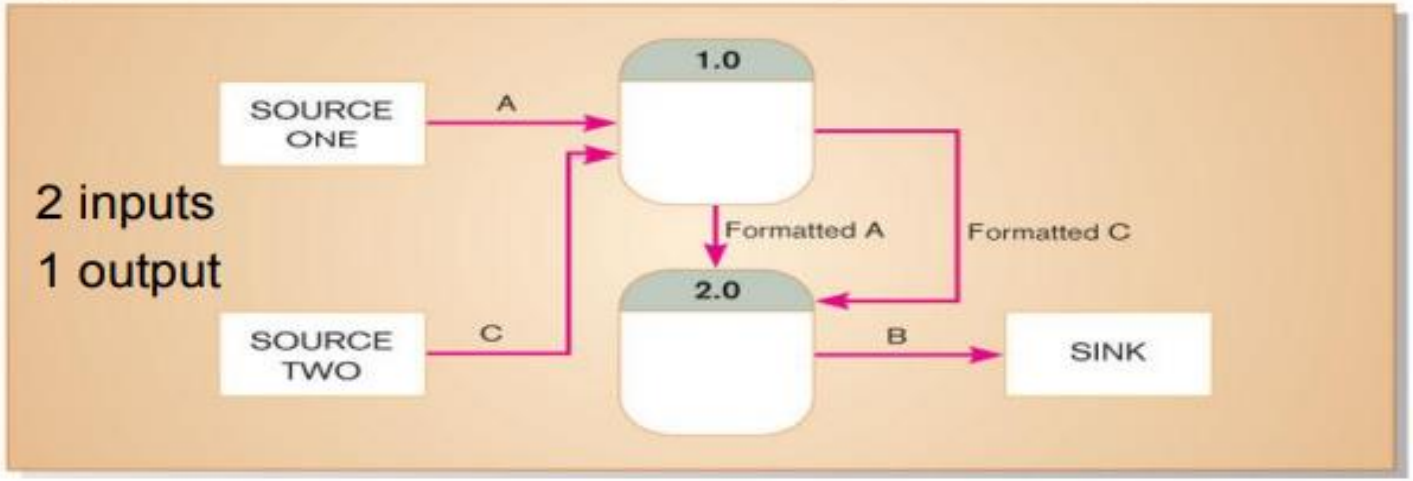


Figure 7-10b An unbalanced set of data flow diagrams - Level-0 diagram



This is unbalanced because the process of the context diagram has only one input but the Level0 diagram has two inputs.

- Data flow splitting is when a composite data flow at a higher level is split and different parts go to different processes in the lower level DFD.
- The DFD remains balanced because the same data is involved, but split into two parts.

Figure 7-11a Example of data flow splitting - Composite data flow

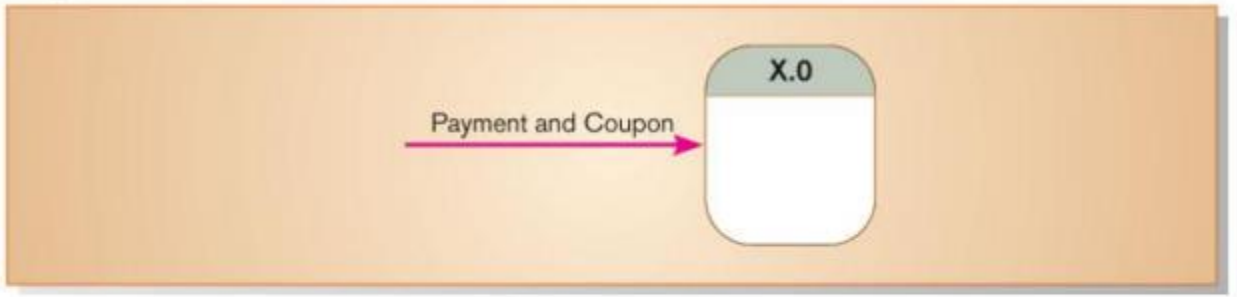
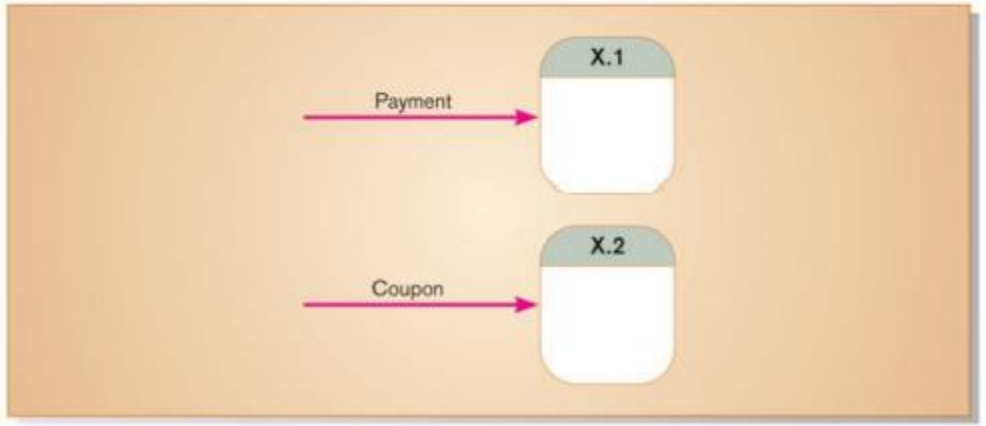


Figure 7-11b Example of data flow splitting - Disaggregated data flows



Let’s look at an example of balancing a set of DFDs. Look back at Figure 7-4. This is the context diagram for Hoosier Burger’s food-ordering system. Notice that there is one input to the system, the customer order, which originates with the customer. Notice also that there are three outputs: the customer receipt, the food order intended for the kitchen, and management reports. Now look at Figure 7-5. This is the level-0 diagram for the food-ordering system. Remember that all data stores and flows to or from them are internal to the system. Notice that the same single input to the system and the same three outputs represented in the context diagram also appear at level 0. Further, no new inputs to or outputs from the system have been introduced. Therefore, we can say that the context diagram and level-0 DFDs are balanced

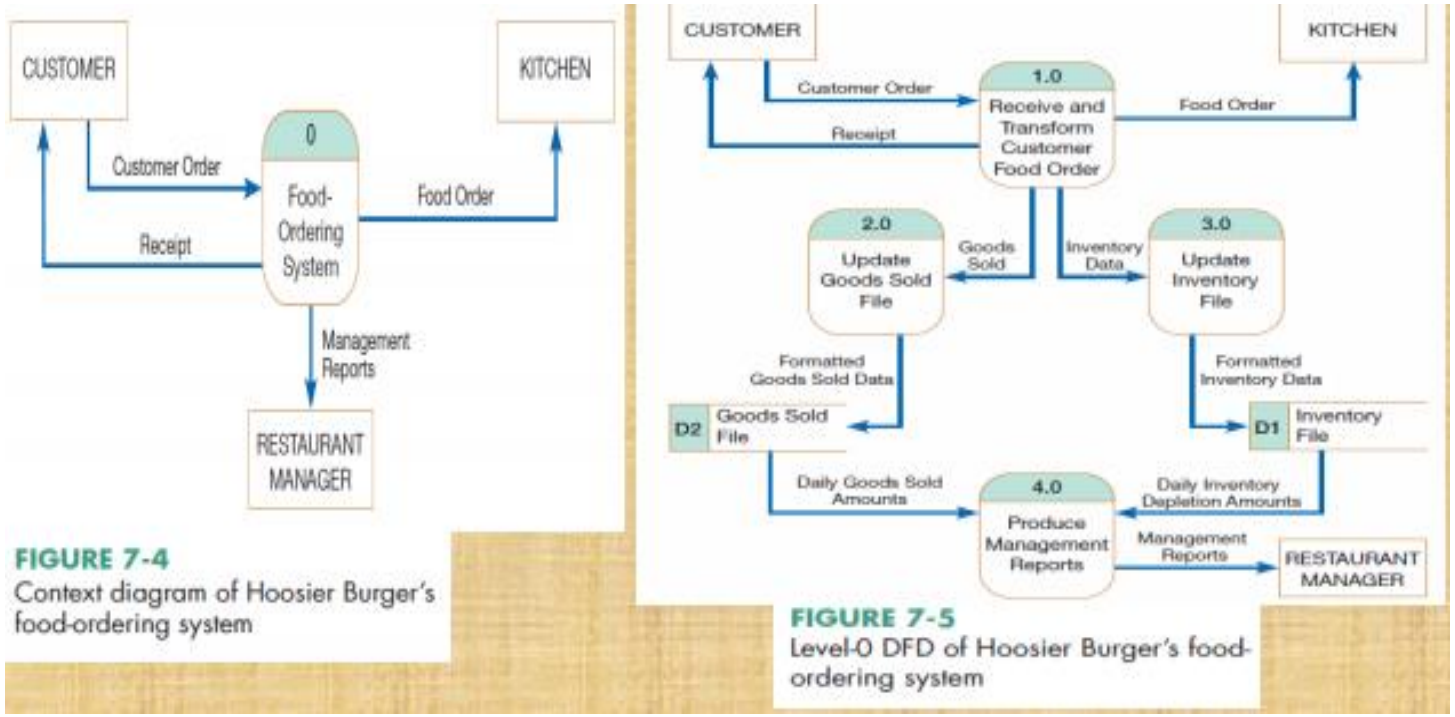


FIGURE 7-4
Context diagram of Hoosier Burger’s food-ordering system

FIGURE 7-5
Level-0 DFD of Hoosier Burger’s food-ordering system

Table 7-3 Advanced Rules Governing Data Flow Diagramming

Q.	A composite data flow on one level can be split into component data flows at the next level, but no new data can be added and all data in the composite must be accounted for in one or more subflows.
R.	The inputs to a process must be sufficient to produce the outputs (including data placed in data stores) from the process. Thus, all outputs can be produced, and all data in inputs move somewhere: to another process or to a data store outside the process or onto a more detailed DFD showing a decomposition of that process.
S.	At the lowest level of DFDs, new data flows may be added to represent data that are transmitted under exceptional conditions; these data flows typically represent error messages (e.g., "Customer not known; do you want to create a new customer"?) or confirmation notices (e.g., "Do you want to delete this record"?).
T.	To avoid having data flow lines cross each other, you may repeat data stores or sources/sinks on a DFD. Use an additional symbol, like a double line on the middle vertical line of a data store symbol or a diagonal line in a corner of a sink/source square, to indicate a repeated symbol.

(*Source:* Adapted from Celko, 1987.)

Four Different Types of DFDs

- **Current Physical**
  - Process labels identify technology (people or systems) used to process the data.
  - Data flows and data stores identify actual name of the physical media.
- **Current Logical**
  - Physical aspects of system are removed as much as possible.
  - Current system is reduced to data and processes that transform them.
- **New Logical**
  - Includes additional functions.
  - Obsolete functions are removed.
  - Inefficient data flows are reorganized.
- **New Physical**
  - Represents the physical implementation of the new system.

Physical & Logical DFD

- Consider the figure 1 It is clear from the figure that orders are placed, orders are received, the location of ordered parts is determined and delivery notes are dispatched along with the order.

Fig 1

It does not however tell us how these things are done or who does them. Are they done by computers or manually and if manually who does them ? A logical DFD of any information system is one that models what occurs without showing how it occurs

- A physical DFD shows, how the various functions are performed? Who does them? Consider the following figure:

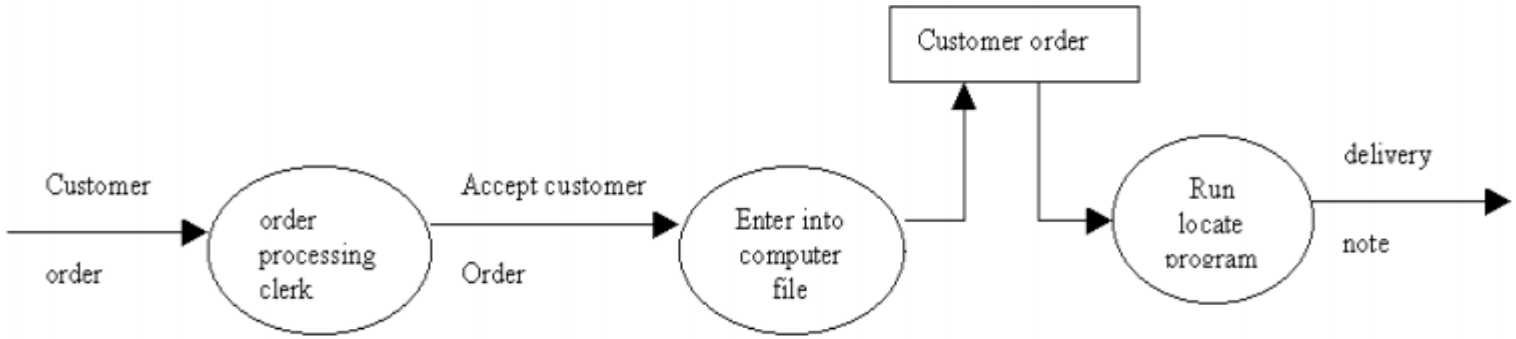


Fig 2

The figure 2 is opposite, it shows the actual devices that perform the functions. Thus there is an "order processing clerk", an "entry into computer file" process and a "run locate program" process to locate the parts ordered. DFD(s) that shows how things happen, or the physical components are called physical DFD(s). Typical processes that appear in physical DFDs are methods of data entry, specific data transfer or processing methods

- Data flow diagrams (DFDs) are categorized as either logical or physical. A logical DFD focuses on the business and how the business operates. It describes the business events that take place and the data required and produced by each event. On the other hand, a physical DFD shows how the system will be implemented. Here are the main differences between logical and physical DFD:

Logical vs Physical DFD

Logical DFD

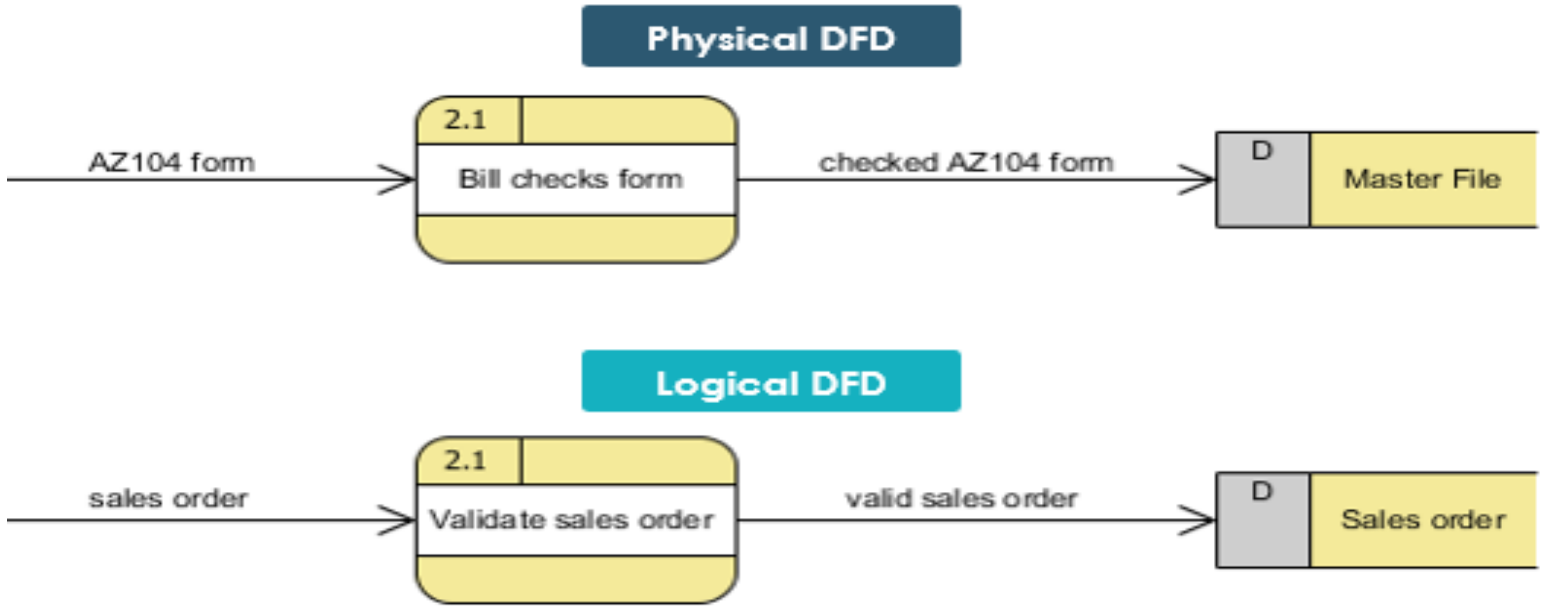
- Logical DFD depicts how the business operates.
- The processes represent the business activities.
- The data stores represent the collection of data regardless of how the data are stored.
- It s how business controls.

Physical DFD

- Physical DFD depicts how the system will be implemented (or how the current system operates).
- The processes represent the programs, program modules, and manual procedures.
- The data stores represent the physical files and databases, manual files.
- It show controls for validating input data, for obtaining a record, for ensuring successful completion of a process, and for system security.

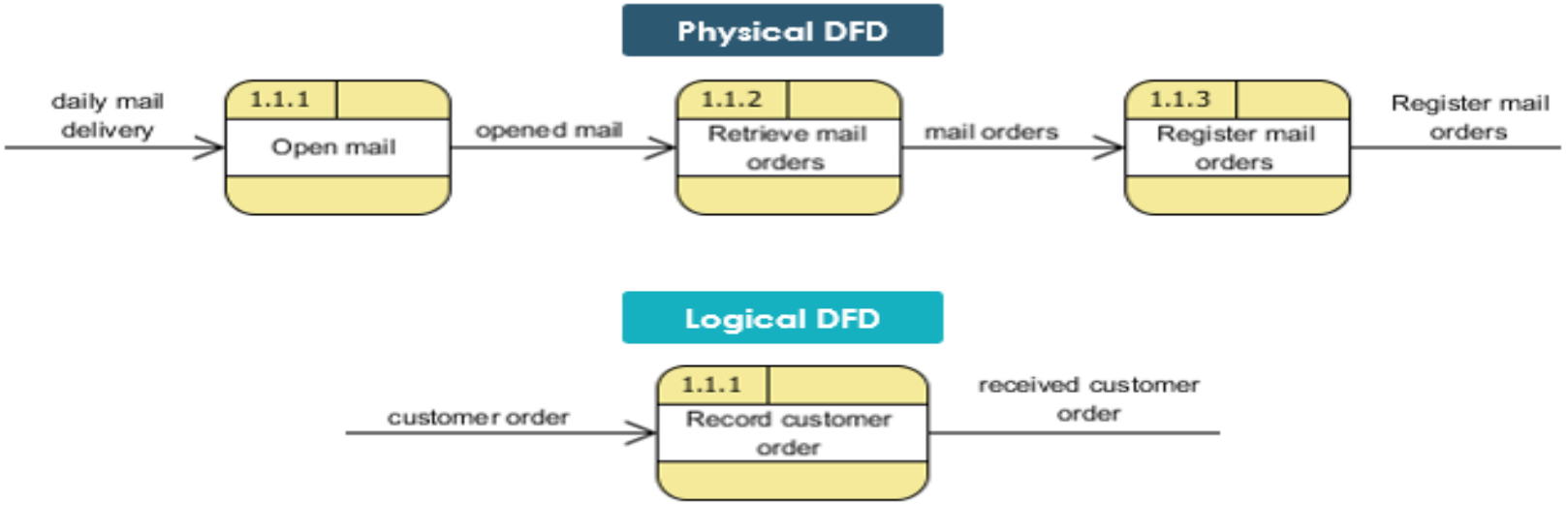
Physical and Logical DFD: Example 1

- Physical DFD specifies actual flow of physical documentation, while logical DFD only focus on the information flow in business term.



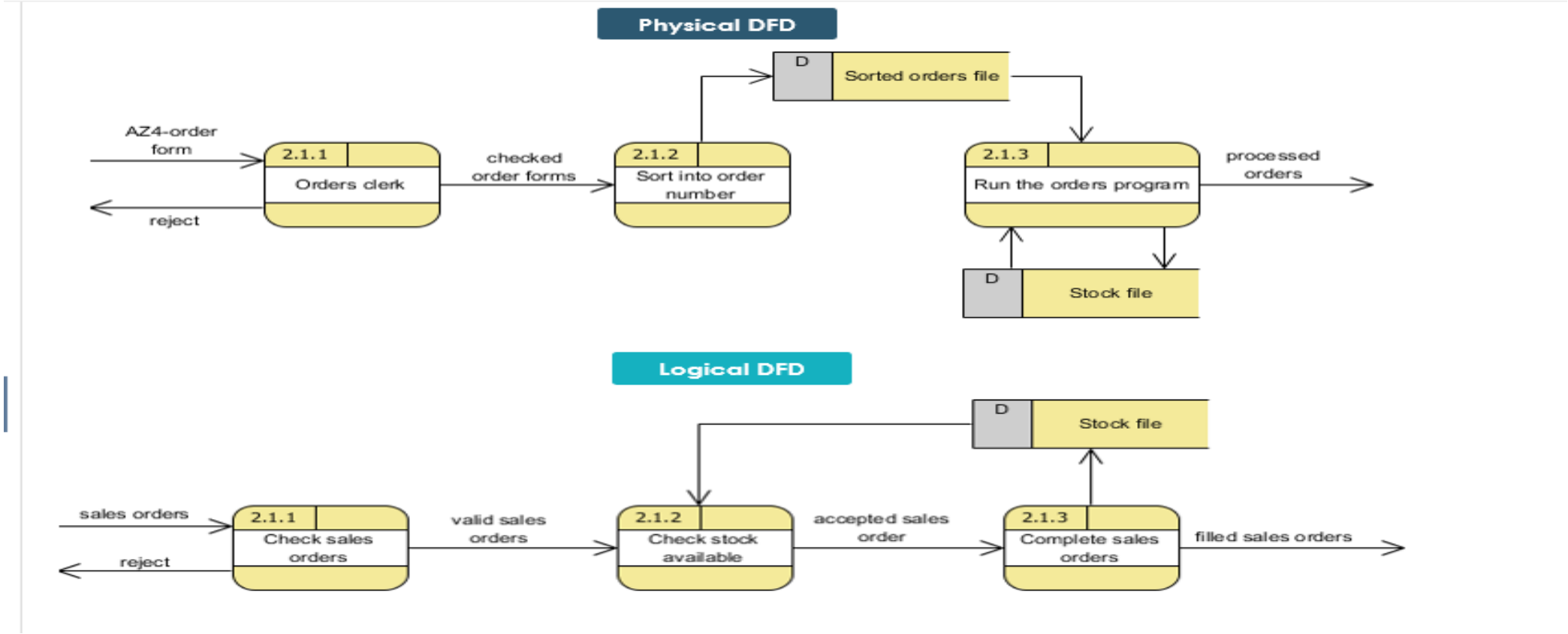
Physical and Logical DFD: Example 2

- Logical DFD eliminates physical processes that refer to physical activities only and do not transform data.



Physical and Logical DFD: Example 3

- The logical DFD describes what the system does by including the essential sequence of business activities. It model the business data and activities instead of actual forms, location and roles.

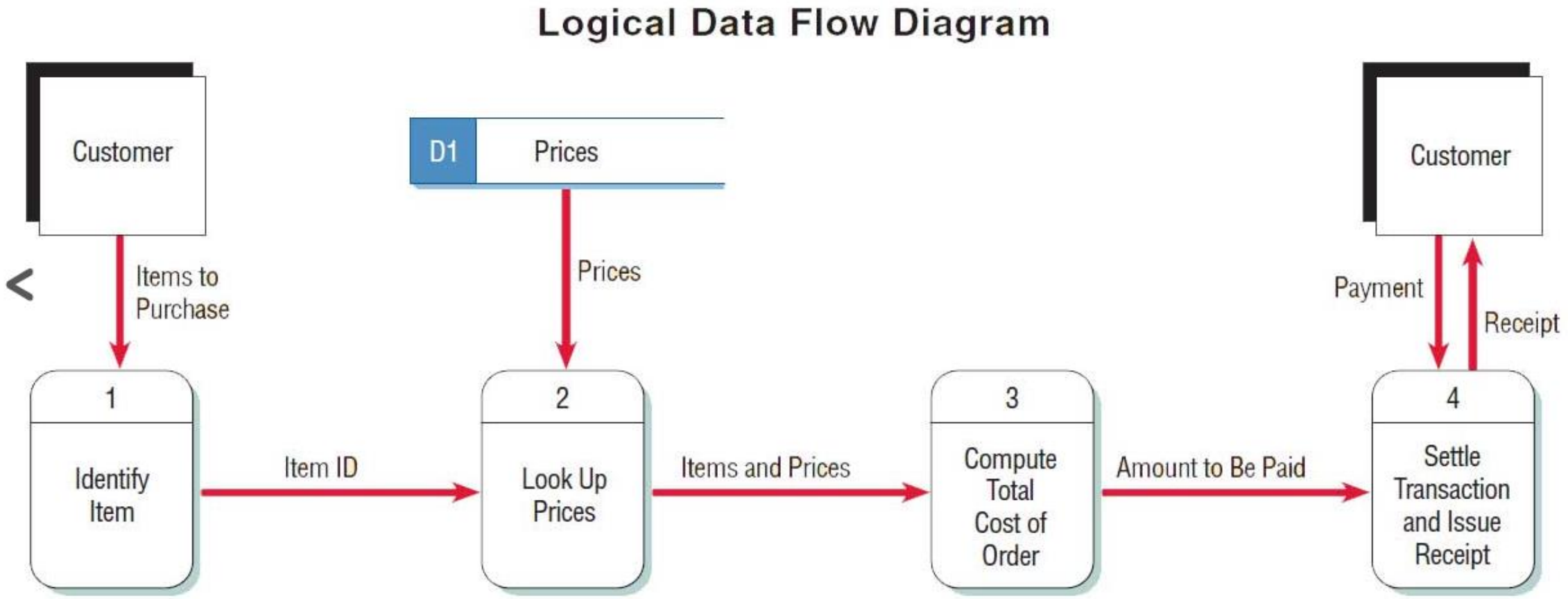


A Larger Example - Grocery Store System

- The example below shows a logical DFD and a physical DFD for a grocery store cashier:
- The CUSTOMER brings the ITEMS to the register;
- PRICES for all ITEMS are LOOKED UP, and then totaled;
- Next, PAYMENT is given to the cashier finally, the CUSTOMER is given a receipt.

Logical DFD Example - Grocery Store

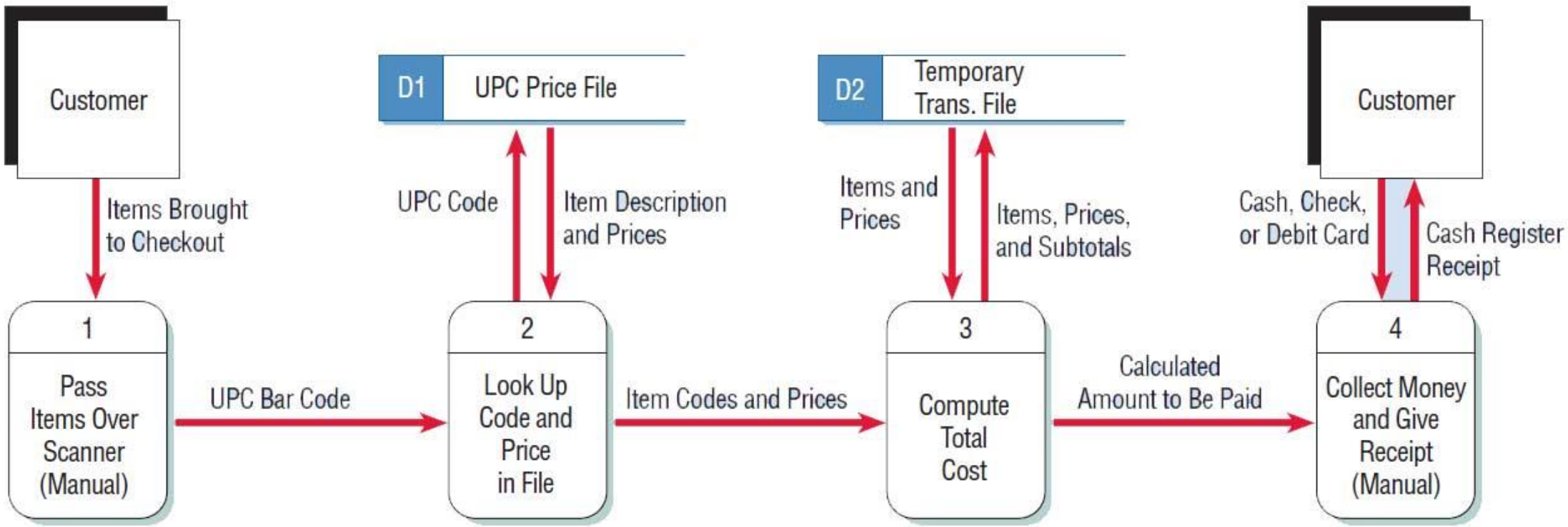
- The logical DFD illustrates the processes involved without going into detail about the physical implementation of activities.



Physical DFD Example - Grocery Store

- The physical DFD shows that a bar code-the UPC PRICE code found on most grocery store items is used
- In addition, the physical DFD mentions manual processes such as scanning, explains that a temporary file is used to keep a subtotal of items
- The PAYMENT could be made by CASH, CHECK, or DEBIT CARD
- Finally, it refers to the receipt by its name, CASH REGISTER RECEIPT

Physical Data Flow Diagram



3.2.4 Using Data flow Diagramming in the Analysis Process

- Learning the mechanics of drawing DFDs is important because DFDs have proven to be essential tools for the structured analysis process. Beyond the issue of drawing mechanically correct DFDs, there are other issues related to process modeling with which an analyst must be concerned. Such issues, including whether the DFDs are complete and consistent across all levels, which covers guidelines for drawing DFDs.
- Another issue to consider is how you can use DFDs as a useful tool for analysis. • Guidelines for drawing DFDs

1. **Completeness :** The concept of DFD completeness refers to whether you have included in your DFDs all of the components necessary for the system you are modeling. If your DFD contains data flows that do not lead anywhere or data stores, processes, or external entities that are not connected to anything else, your DFD is not complete.
2. **Consistency :** The concept of DFD consistency refers to whether or not the depiction of the system shown at one level of a nested set of DFDs is compatible with the depictions of the system shown at other levels. A gross violation of consistency would be a level-1 diagram with no level-0 diagram. Another example of inconsistency would be a data flow that appears on a higher-level DFD but not on lower levels (also a violation of balancing).

3. Timing: You may have noticed in some of the DFD examples we have presented that DFDs do not do a very good job of representing time. On a given DFD, there is no indication of whether a data flow occurs constantly in real time, once per week, or once per year. There is also no indication of when a system would run.

4. Iterative Development: The first DFD you draw will rarely capture perfectly the system you are modeling. You should count on drawing the same diagram over and over again, in an iterative fashion. With each attempt, you will come closer to a good approximation of the system or aspect of the system you are modeling. One rule of thumb is that it should take you about three revisions for each DFD you draw.

5. Primitive DFDs : One of the more difficult decisions you need to make when drawing DFDs is when to stop decomposing processes. One rule is to stop drawing when you have reached the lowest logical level; however, it is not always easy to know what the lowest logical level is.

Rules for stopping decomposition

- When each process has been reduced to a single decision, calculation or database operation.
- When each data store represents data about a single entity.
- When the system user does not care to see any more detail.
- When every data flow does not need to be split further to show that data are hand led in various ways.
- When you believe that you have shown each business form or transaction, online display and report as a single data flow.
- When you believe that there is a separate process for each choice on all lowest level menu options.

3.2.5 Modeling Logic With Decision Tables

- Logic Modeling:

➢ Data flow diagrams do not show the logic inside the processes.

➢ Logic modeling involves representing internal structure and functionality of processes depicted on a DFD.

➢ Logic modeling can also be used to show when processes on a DFD occur.

➢ A logic model has four components:

1. **Needs:** are about the problem and why its important to address it. What issue are we trying to address?
2. **Inputs:** are the things that contribute 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 are we doing with these resources.
4. **Outcomes:** are usually expressed in terms of measures of success. What differences are we hoping to make?

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

- Structured English representation of process logic.
- Decision Tables representation.
- Sequence diagram.
- Activity diagram
- Decision Trees
- State-transition diagrams

- A decision table is a diagram of process logic where the logic is reasonably complicated. All of the possible choices and the conditions the choices depend on are represented in tabular form, as illustrated in the decision table in Figure 7-18. The decision table in Figure 7-18 models the logic of a generic payroll system.
- The table has three parts: the condition stubs, the action stubs, and the rules. The condition stubs contain the various conditions that apply to the situation the table is modeling.

Decision table
A matrix representation of the logic of a decision; it specifies the possible conditions for the decision and the resulting actions.

Condition stubs
The part of a decision table that lists the conditions relevant to the decision.

Action stubs
The part of a decision table that lists the actions that result for a given set of conditions.

Rules
The part of a decision table that specifies which actions are to be followed for a given set of conditions.

- In Figure 7-18, there are two condition stubs for employee type and hours worked. Employee type has two values: “S,” which stands for salaried, and “H,” which stands for hourly. Hours worked has three values: less than 40, exactly 40, and more than 40. The action stubs contain all the possible courses of action that result from combining values of the condition stubs. There are four possible courses of action in this table: Pay Base Salary, Calculate Hourly Wage, Calculate Overtime, and Produce Absence Report. You can see that not all actions are triggered by all combinations of conditions. Instead, specific combinations trigger specific actions. The part of the table that links conditions to actions is the section that contains the rules.

	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				

FIGURE 7-18
Complete decision table for payroll system example

- To read the rules, start by reading the values of the conditions as specified in the first column: Employee type is "S," or salaried, and hours worked is less than 40. When both of these conditions occur, the payroll system is to pay the base salary.
- In the next column, the values are "H" and "<40," meaning an hourly worker who worked less than 40 hours. In such a situation, the payroll system calculates the hourly wage and makes an entry in the Absence Report. Rule 3 addresses the situation when a salaried employee works exactly 40 hours. The system pays the base salary, as was the case for rule 1. For an hourly worker who has worked exactly 40 hours, rule 4 calculates the hourly wage. Rule 5 pays the base salary for salaried employees who work more than 40 hours. Rule 5 has the same action as rules 1 and 3 and governs behavior with regard to salaried employees. The number of hours worked does not affect the outcome for rules 1, 3, or 5. For these rules, hours worked is an **indifferent condition in that its value does not affect the action taken.** Rule 6 calculates hourly pay and overtime for an hourly worker who has worked more than 40 hours.

	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				

- indifferent condition
- In a decision table, a condition whose value does not affect which actions are taken for two or more rules.

Because of the indifferent condition for rules 1, 3, and 5, we can reduce the number of rules by condensing rules 1, 3, and 5 into one rule, as shown in Figure 7-19. The indifferent condition is represented with a dash. Whereas we started with a decision table with six rules, we now have a simpler table that conveys the same information with only four rules.

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		

FIGURE 7-19
Reduced decision table for payroll system example

- Decision tables are a concise / short / brief visual representation for specifying which actions to perform depending on given conditions.
- They are algorithms whose output is a set of actions.
- A **decision table** is an excellent tool to use in both testing and requirements management.
- **Decision Table** is a structured exercise to formulate requirements when dealing with complex business rules.
- **Decision Tables** are used to model complicated logic.
- Lets take an example scenario for an ATM where a decision table would be of use.
- A customer requests a cash withdrawal. One of the business rules for the ATM is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted.

- This simple example of a business rule is quite complicated to describe in text. A decision table makes the same requirements clearer to understand:

Conditions	R1	R2	R3
Withdrawal amount <= balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

In a decision table, conditions are usually expressed as true (T) or false (F). Each column in the table corresponds to a rule in the business logic that describes the unique combination of circumstances that will result in the actions.

- Decision tables can be used in all situations where the outcome depends on the combinations of different choices, and that is usually very often.
- In many systems there are tons of business rules where decision tables add a lot of value.
- **Steps to create decision tables**
- **Step 1 – Analyze the requirement and create the first column**
- Requirement: “Withdrawal is granted if requested amount is covered by the balance or if the customer is granted credit to cover the withdrawal amount”.
- Express conditions and resulting actions in a list so that they are either TRUE or FALSE.
- In this case there are two conditions, “withdrawal amount <= balance” and “credit granted”.

- There is one result, the withdrawal is granted.

Conditions
Withdrawal amount <=balance
Credit granted
Actions
Withdrawal granted

- **Step 2: Add Columns**
- Calculate how many columns are needed in the table
- The number of columns depends on the number of conditions and the number of alternative for each condition.
- Number of column that is needed:

Number of conditions	Number of Columns
1	2
2	4
3	8
4	16
5	32

- **Step 2: Add Columns**
- To fill True (T) and False (F) for the conditions, the simplest way is:
- Row 1: TF
- Row 2: TTFF
- Row 3: TTTTFFFF
- For each row, there is twice as many T and F as the previous line.

Conditions				
Withdrawal Amount <= Balance	T	F	T	F
Credit granted	T	T	F	F
Actions				
Withdrawal granted				

- **Step 3: Reduce the table**
- Make insignificant values with “-”. If the requested amount is less than or equal to the account balance it does not matter if credit is granted.
- In the next step, you can delete the column that have become identical.

Conditions				
Withdrawal amount<=balance	T	F	T	F
Credit granted	-	T	-	F
Actions				
Withdrawal granted				

- Check for invalid combinations.
- Invalid combinations are those that cannot happen, for example, that someone is both an infant and senior. Mark them somehow, e.g. with “X”. In this example, there are no invalid combinations.
- Finish by removing duplicate columns. In this case, the first and third column are equal therefore one of them is removed.

- **Step 4: Determining actions**
- Enter actions for each column in the table. Name the columns (the rules) be named R1/ Rule 1, R2/Rule 2 and so on, but you can also give them descriptive names.

Conditions			
Withdrawal amount<=balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

- **Step 5: Write test cases**
- Write test cases based on the table. At least one test case per column gives full coverage of all business rules.
- **Test Case for R1:** Balance = 200, requested withdrawal = 200. Expected result: Withdrawal granted.
- **Test Case for R2:** Balance = 100, requested withdrawal = 200. Credit granted. Expected result: Withdrawal granted.
- **Test Case for R3:** Balance = 100, requested withdrawal = 200. No credit. Expected result: Withdrawal denied.

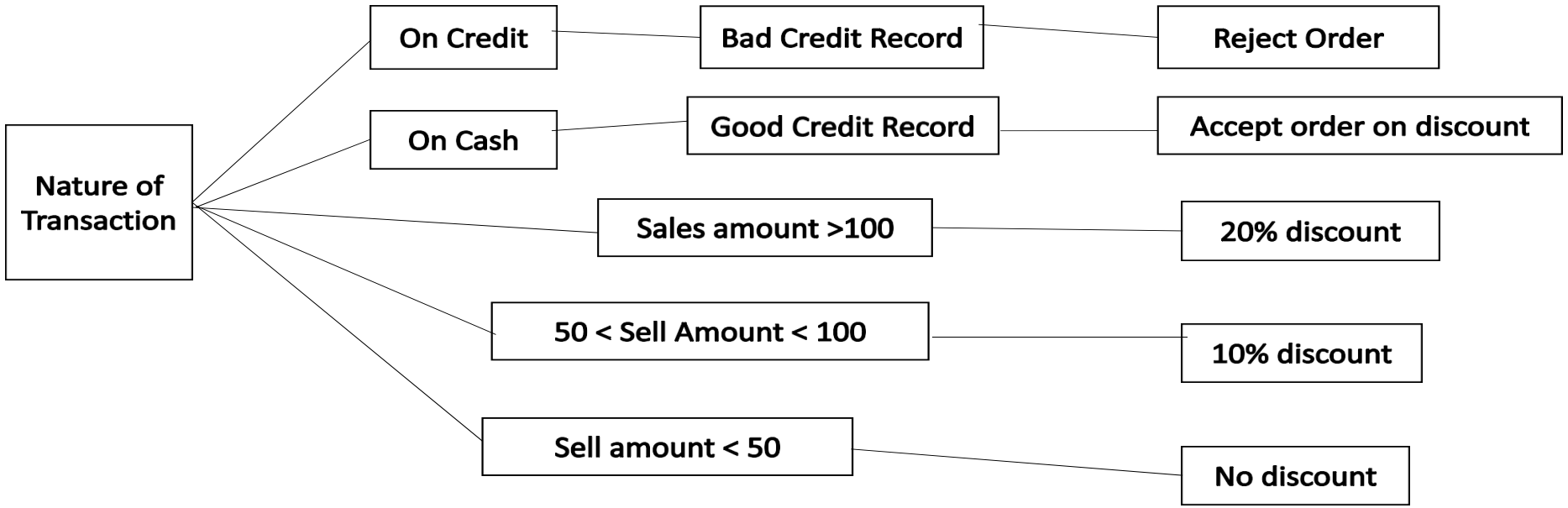
Advantages and disadvantage of decision tables

- One advantage of using decision table is that they make it possible to detect combinations of decisions that would otherwise not have been found and therefore not tested or developed.
- The requirements become much clearer and you often realize that some requirements are illogical, something that is hard to see when the requirements are only expressed in text.
- A disadvantage of the technique is that a decision table is not equivalent to complete test cases containing step by step instructions of what to do in what order. When this level of data is required, the decision table has to be further detailed into test cases.

Decision Trees

- It is the most powerful and popular tool for classification and prediction.
- A **Decision Tree** is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.
- Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication.
- A **Decision Tree** is a diagram that shows alternative actions and conditions within horizontal tree framework.
- A square node indicates an action and a circle indicates a condition.
- It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.

- The table below shows the decision tree:



Advantages of decision tree:

1. It expresses the logic of if then else in pictorial form

2. It is useful to express the logic when a value is variable or an action is dependent.

3. It helps the analyst to identify the actual decision to be made.

4. It is used to verify logic and problems that involve a few complex decision and limited number of actions.

Disadvantages of decision tree:

1. There is absent of information in its format to take for conditions to take.

2. Large number of branches with many paths will confuse rather than help in analysis.

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

Example of program to print Fibonacci up to n numbers.

Void function Fibonacci

Get value of n;

Set value of a to 1;

Set value of b to 1;

Initialize I to 0;

For(i=0; i<n; i++)

{

if a greater than b

{

increase b by a;

print b;

}

Else if b greater than a

{

increase a by b;

print a;

}

}

Advantages of Pseudo-code

1. Improves the readability of any approach.

2. It is one of the best approach to start implementation of an algorithm.

3. Acts as a bridge between the program and the algorithm or flowchart.

4. The main goal 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.

3.3 System Data Requirements

3.3.1 Introduction,

3.3.2 Conceptual Data Modeling,

3.3.3 Gathering Information For Conceptual Data Modeling,

3.3.4 Introduction To ER Modeling,

3.3.5 Conceptual Data Modeling And The ER Model,

3.3.6 Representing Super Types And Sub-Types,

3.3.7 Business Rules,

3.3.8 Role Of Packaged Conceptual Data Models- Database Patterns

3.3 System Data Requirements
3.3.1 Introduction

- Structuring system and database requirements concentrates on the definition, structure and relationships within data.
- The characteristics of data captured during data modeling are crucial in the design of databases, programs, computer screens and printed reports. This information is essential in ensuring data integrity in an information system.
- The most common format used for data modeling in 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.

- The most common format used for data modeling is **entity-relationship (E-R) diagramming.**
- A similar format used with object-oriented analysis and design methods is class diagramming, which is included in a special section at the end of this chapter on the object-oriented development approach to data modeling.
- Data models that use E-R and class 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

3.3.2 Conceptual Data Modeling

- 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.
- Analysts develop (or use from prior systems development) a conceptual data model for the current system and then build or refine a purchased conceptual data model that supports the scope and requirements for the proposed or enhanced system.
- The work of all team members is coordinated and shared through the project **dictionary or repository.** This repository is often maintained by a common Computer- Aided Software Engineering (CASE) or data modeling software tool.

- **The Conceptual Data Modeling process:** The process of conceptual data modeling begins with developing a conceptual data model for the system being replaced, if a system already exists. This is essential for planning the conversion of the current files or database into the database of the new system. Further, this is a good, but not a perfect, starting point for your understanding of the data requirements of the new system. Then, a new conceptual data model is built (or a standard one is purchased) that includes all of the data requirements for the new system.
- **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. Because of the rapidly increasing interest in object-oriented methods, class diagrams using unified modeling language (UML) drawing tools such as IBM's Rational products or Microsoft Visio are also popular.

1. The primary deliverable from the conceptual data modeling step within the analysis phase is an E-R diagram.
2. 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. The repository is the mechanism to link data, process and logic models of an information system.

3.3.3 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. For example, during interviews with potential system users—during Joint Application Design (JAD) sessions or through requirements interviews—you must ask specific questions in order to gain the perspective on data that you need to develop or tailor a purchased data model.
- You typically do data modeling from a combination of perspectives. The first perspective is generally called the **top-down approach**. This perspective derives the business rules for a data model from an intimate understanding of the nature of the business, rather than from any specific information requirements in computer displays, reports, or business forms.

1. *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**
2. *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**
3. *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**
4. *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**
5. *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**
6. *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**
7. *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**
8. *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**

TABLE 8-1 Requirements Determination Questions for Data Modeling

- 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. This process of gaining an understanding of data is often called a **bottom-up approach.** These items will appear as data flows on DFDs and will show the data processed by the system and, hence, probably the data that must be maintained in the system's database. Consider, for example, Figure 8-4, which shows a customer order form used at Pine Valley Furniture (PVF). From this form, we determine that the following data must be kept in the database:

ORDER NO	CUSTOMER NO
ORDER DATE	NAME
PROMISED DATE	ADDRESS
PRODUCT NO	CITY-STATE-ZIP
DESCRIPTION	
QUANTITY ORDERED	
UNIT PRICE	

PVF CUSTOMER ORDER

ORDER NO: 61384 CUSTOMER NO: 1273

NAME: Contemporary Designs
ADDRESS: 123 Oak St.
CITY-STATE-ZIP: Austin, TX 28384

ORDER DATE: 11/04/2014 PROMISED DATE: 11/21/2017

PRODUCT NO	DESCRIPTION	QUANTITY ORDERED	UNIT PRICE
M128	Bookcase	4	200.00
B381	Cabinet	2	150.00
R210	Table	1	500.00

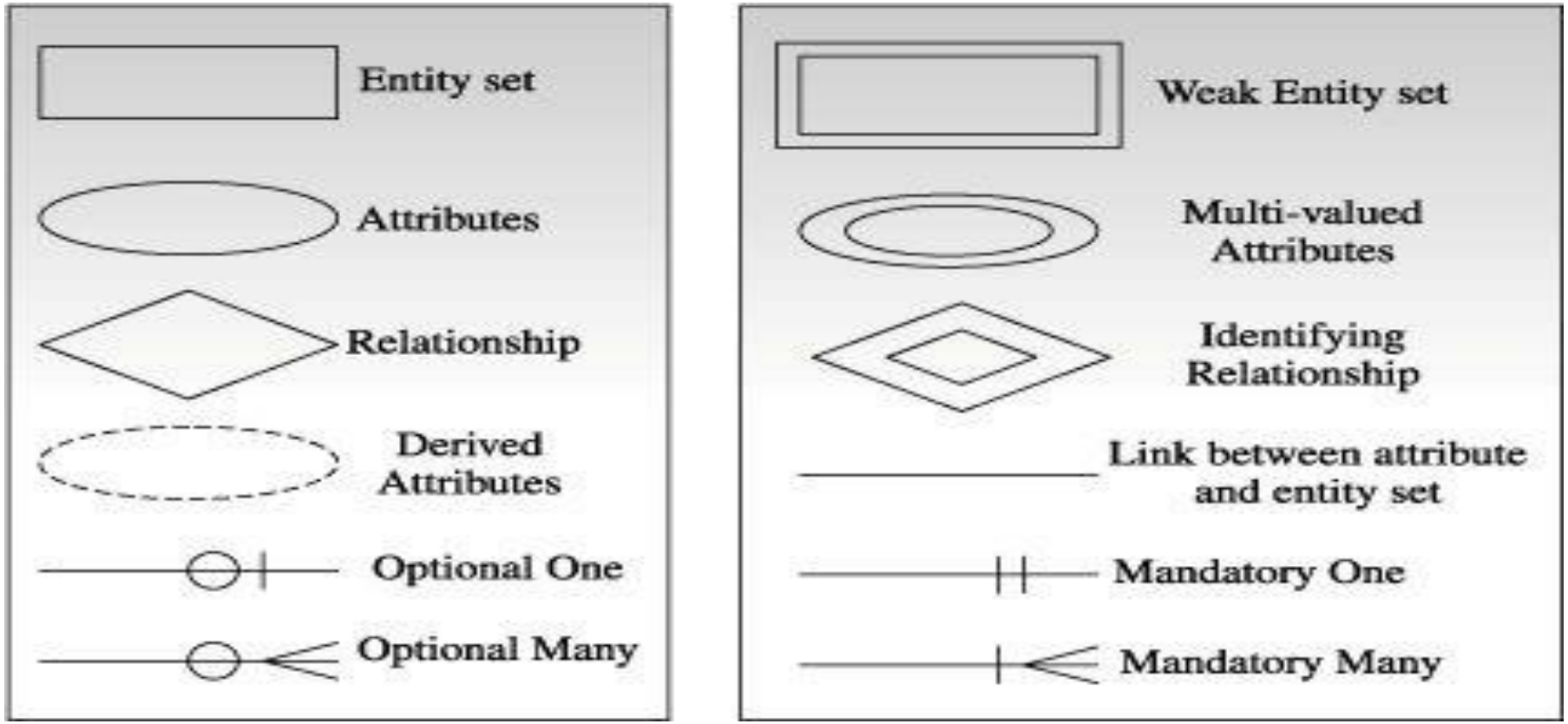
FIGURE 8-4
Sample customer form

3.3.4 Introduction To ER Modeling

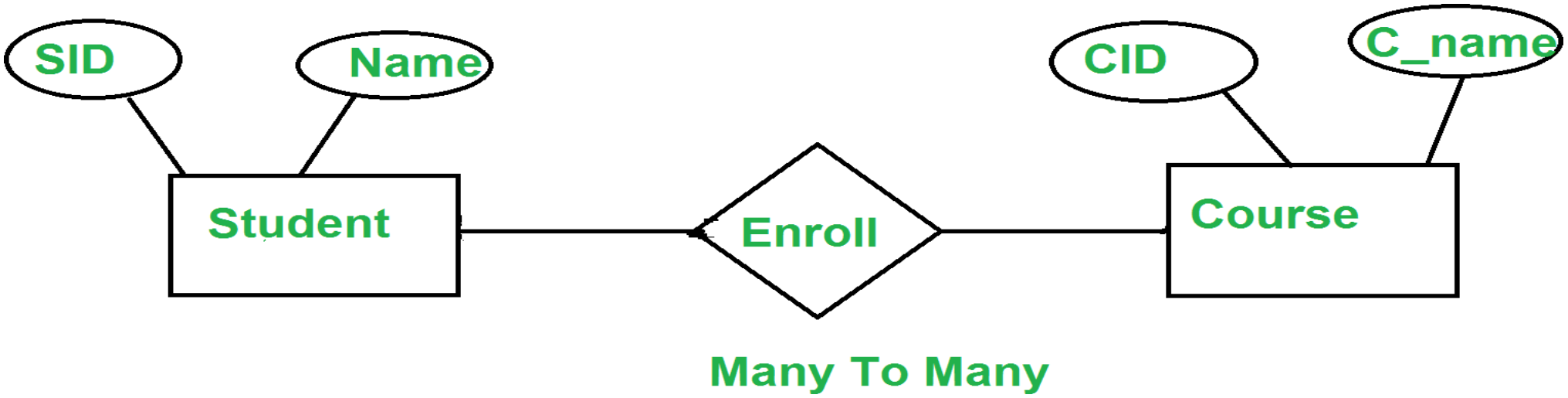
- An E-R data model is a detailed, logical representation of the data for an organization or for a business area.
- An entity is a person, place, object, event, or concept in the user environment about which the organization wishes to maintain data.
- An entity has its own identity, which distinguishes it from other entities.
- There is an important distinction between **entity types** and **entity instances**.
- An **entity type** is a single occurrence of an entity type.
- For example, there is usually one EMPLOYEE type but there may be hundreds of instances of this entity type stored in a database.
- Each entity type has a set of attributes associated with it.
- For example, for an entity STUDENT we have such attributes like: STUDENT NO, NAME, ADDRESS, PHONE NO.
- Every entity must have an attribute or set of attributes that distinguishes one instance from other instances of the same type – and that is called a **candidate key**.
- A **primary key** is a candidate key that has been selected to be used as the identifier for the entity type.
- For each entity the name of the primary key is underlined on an E-R diagram.

- **Entity Relationship Modeling (ER Modeling)** is a graphical approach to database design.
- It uses Entity/Relationship to represent real world objects.
- An entity is a thing or object in real world that is distinguishable from surrounding environment. For example each employee of an organization is a separate entity.
- It is a technique used in database design that helps describe the relationship between various entities of an organization. Terms used in E-R model are:
- **Entity** – It specifies distinct real world items in an application. For example: vendor, item, student course, teachers etc.
- **Relationship** – They are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and teachers are relationships.
- **Attributes** – It specifies the properties of relationships. For example, vendor code, student name.

- **Entity-relationship diagram (ERD)** are essential to modeling anything from simple to complex databases, but the shapes and notations used can be very confusing.
- There are various types of symbols used for ER diagram, some of well-defined symbols are listed below;



- **Entity-relationship diagram (ERD)** are essential to modeling anything from simple to complex databases, but the shapes and notations used can be very confusing.
- There are various types of symbols used for ER diagram, some of well-defined symbols are listed below;



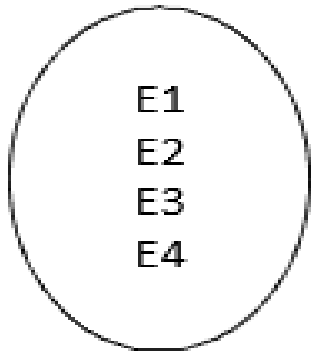
Elements of ER Diagram

- **ENTITY:**
- An entity is a real-world thing either living or non-living that is easily recognizable and non-recognizable.
- 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.
- For example: Tuple1/ row1 / record1 contains information about Hari (id, name, age, address) which has existence in real world. So, the tuple1 is an entity. So, we may say each tuple is an entity. Let “Hari” is a particular member of entity type student.

Entity

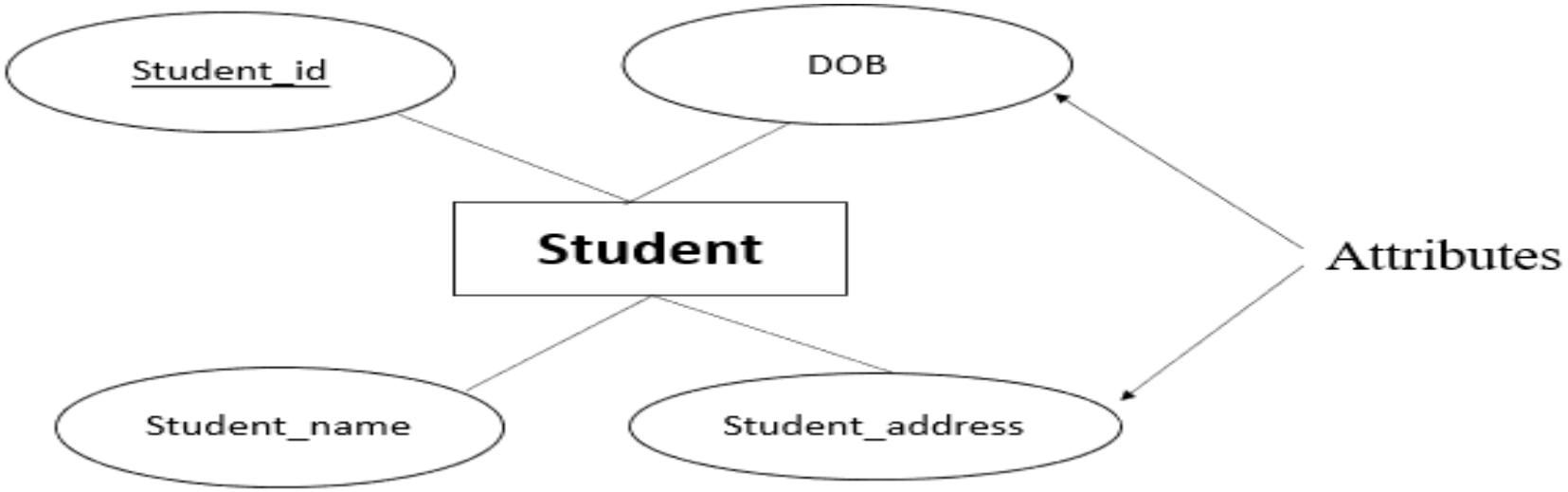
- **ENTITY TYPE:**
- It is collection of entities having common attribute.
- As in student table, each row is an entity and has common attributes.
- So, Student is a entity type which contains entities having attributes id, name and age.
- Also each entity type in a database is described by a name and a list of attribute.
- For example: Collection of entities with similar properties.

- **ENTITY SET:**
- It is same as entity type, but defined at a particular point in time such as students enrolled in a class on the first day.
- Other example: customer who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an instance of the entity set.
- For example: let E1 is an entity having entity type student and set of all student is called entity set.



Entity set

- **Attributes:**
- Attributes are the properties which define the entity type. For example, student_id, student_name, DOB, age, address, mobile_no etc. are the attributes which define entity type student.
- In ER diagram, attribute is represented by an oval.



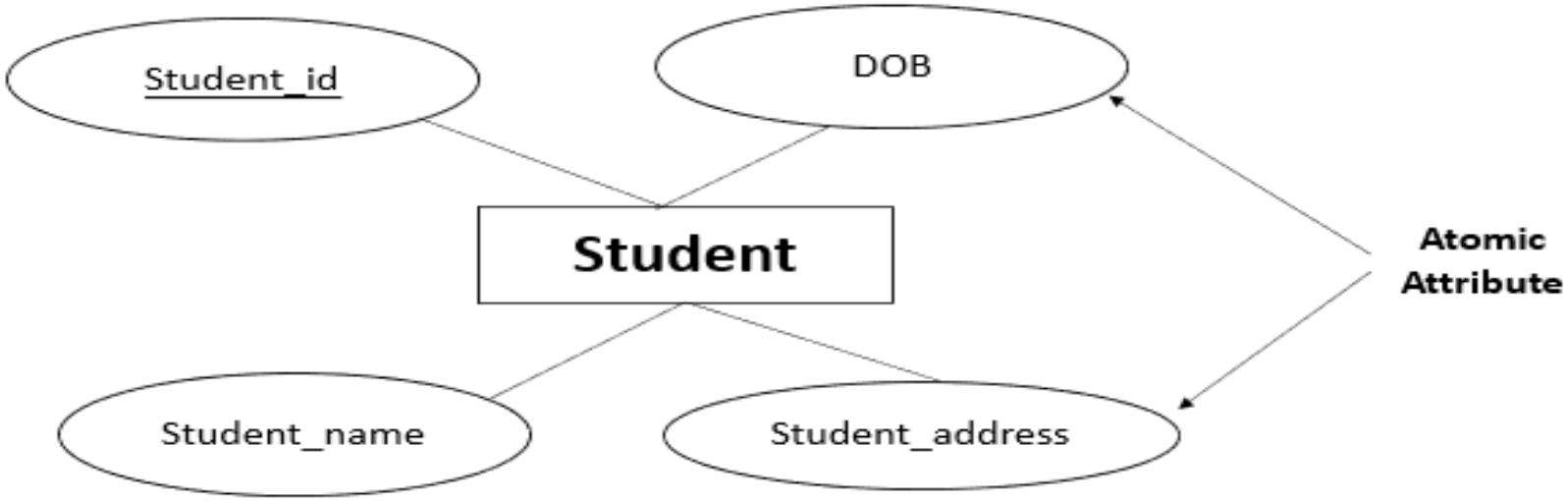
Types of Attributes:

- Attributes of an entity type can be further divided into following types.
- A. Atomic Vs. Composite attributes
- B. Single valued Vs. Multi valued attributes
- C. Stored Vs. Derived attributes
- D. NULL value attribute
- E. Key attribute

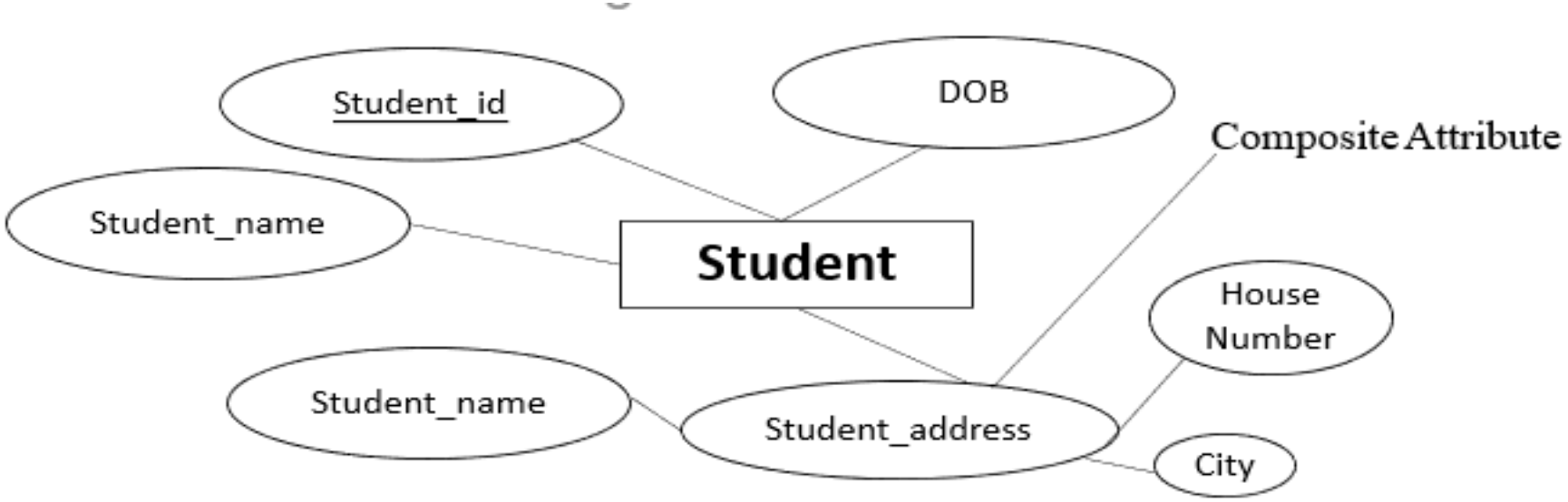
Types of Attributes

A. Atomic Vs. Composite attributes

- An attribute that cannot be divided into smaller independent attribute is known as **atomic attribute**. For example: Assume, student is an entity and its attributes are name, age, DOB, address and phone number.
- Here the student_id, DOB attribute of students (entity) cannot further divide. In this example, Student_id and DOB are atomic attribute.

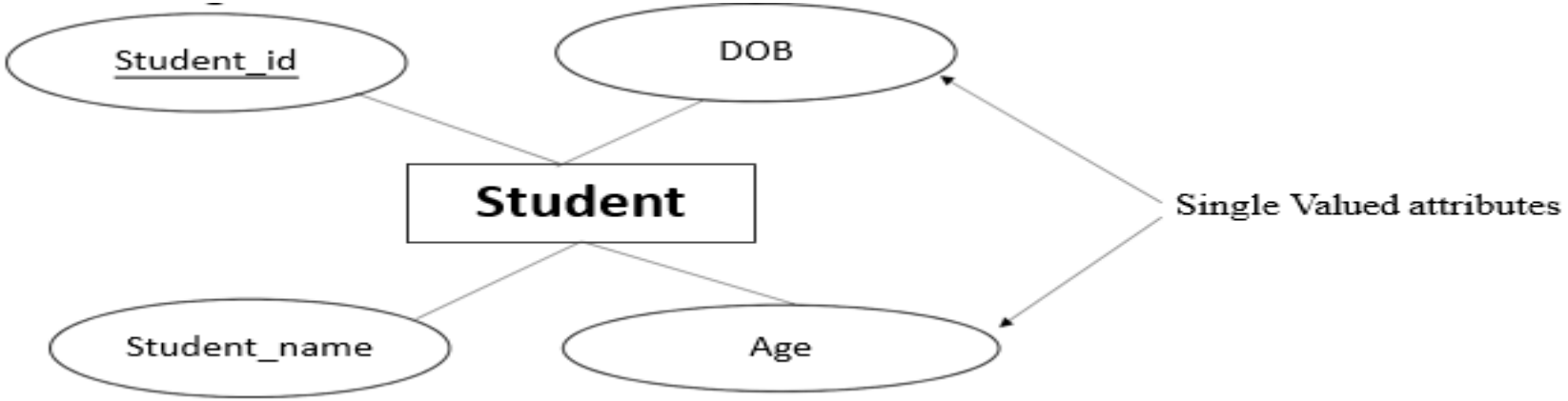


- An attribute that can be divided into smaller independent attribute is known as **composite attribute**. For example: Assume, student is an entity and its attributes are student_id, name, age, DOB, address and phone number.
- Here the address (attribute) of student (entity) can be further divided into house number, city and so on. In this example, address is composite attribute.

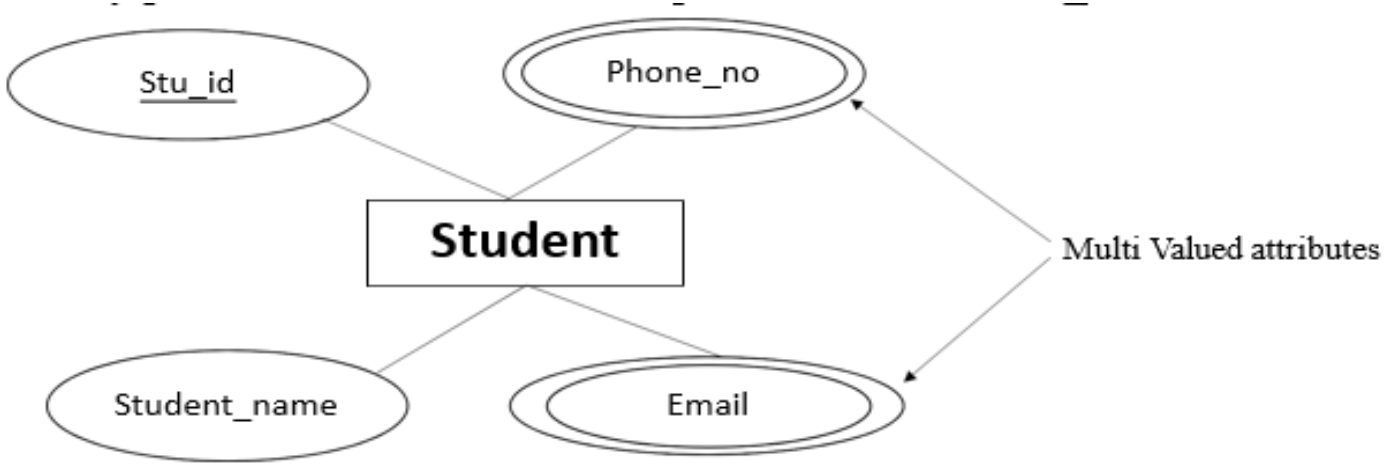


B. Single Valued Vs. Multi Valued attributes

- An attribute that has only single value for an entity is known as single valued attribute. For example: Assume, student is an entity and its attributes are Stu_id, Name, age, DOB, address and phone number.
- Here the age and DOB(attribute) of student (entity) can have only one value. In this example, age and DOB are single valued attribute.

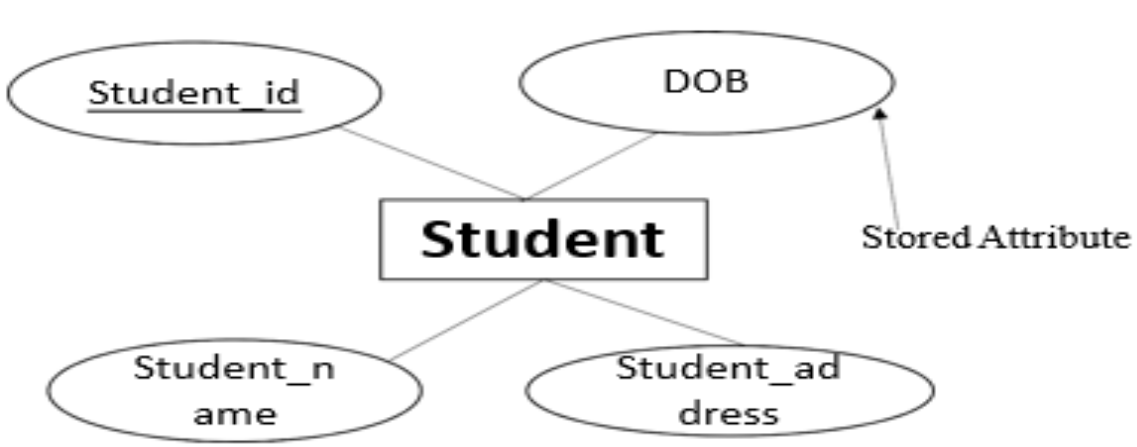


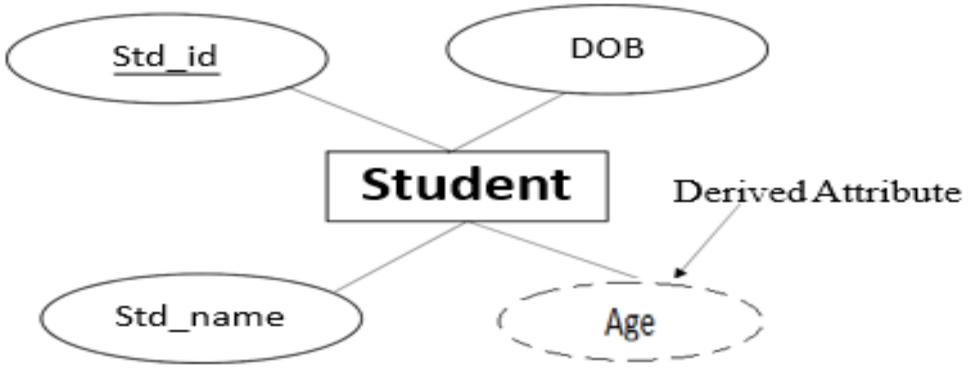
- An attribute that can have multiple values for an entity is known as **multi valued attribute.** For example: Assume, Student is an entity and its attributes are Stu_id, Name, age, DOB, address, email and phone no.
- Here the phone no and Email (attribute) of Student (entity) can have multiple values because a student may have many phone numbers. In this example, Email and Phone_no are multi valued attributes.



C. Stored Vs. Derived attributes

- An attribute that cannot be derived from another attribute and we need to store their value in database is known as **stored attribute**. For example, DOB cannot derive from age of student.
- An attribute that can be derived from another attribute and we do not need to store their value in the database due to dynamic nature is known as **derived attribute.**
- It is denoted by dotted oval. For example, age can be derived from birth date of student.





D. NULL value attributes

- An attribute, which has not any value for an entity is known as null value attribute. For example, assume Student is an entity and its attributes are Name, Age, Address and Phone_no.
- There may be chance when a student has no phone no. In this case, phone number is called NULL valued attributes.

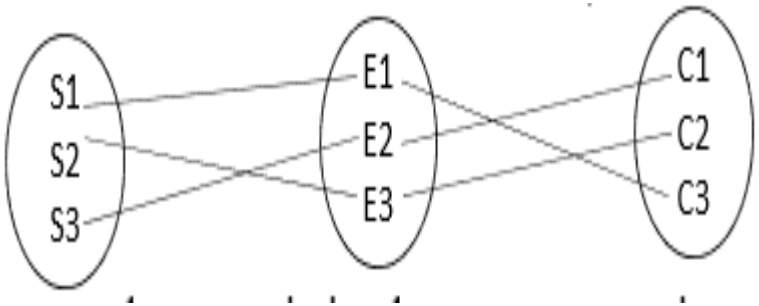
E. Key attributes

- An attribute that has unique value of each entity is known as key attribute. For example, every student has unique roll no. Here roll no is key attribute.

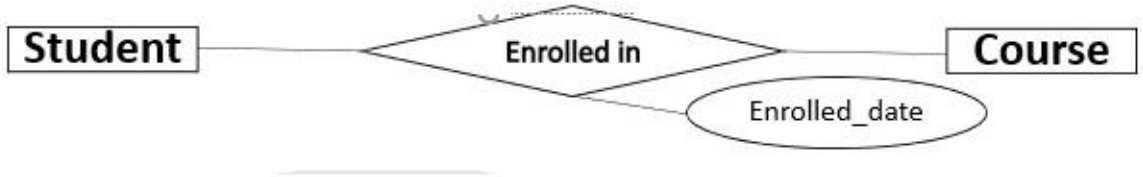
Relationship Type and Relationship Set

- A relationship type represents the association between entity types. For example, “Enrolled in” is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.

- A set of relationships of same type is known as relationship set. The following relationship set depicts / represents / illustrates S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled C3.



- In another way, we can say that association between two entity sets is called relationship set. A relationship set may also have attributes called descriptive attributes. For example, the enrolled_in relationship set between entity sets student and course may have the attribute enrolled_date.

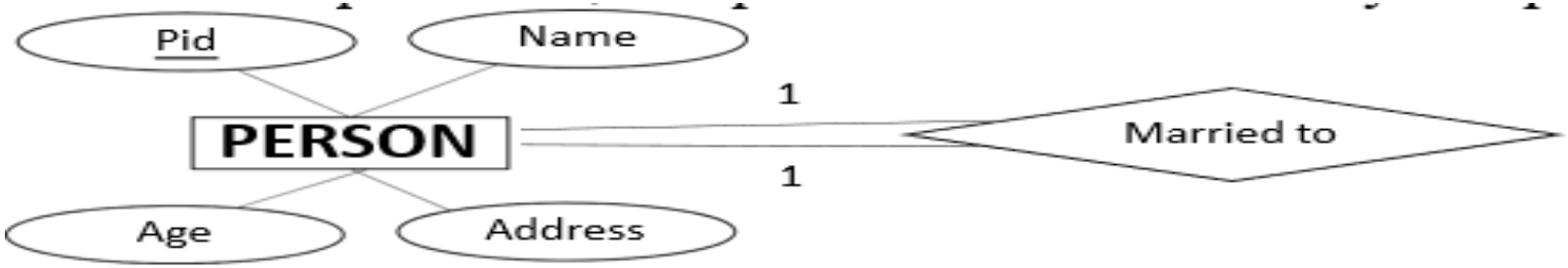


3.3.5 Conceptual Data Modeling And The E-R Model: Degree of a Relationship

- Number of entity sets that participate in a relationship set is called degree of the relationship set. On the basis of degree, relationships can be divided as below:
- Unary Relationship
- Binary Relationship
- N-ray Relationship
- **Unary Relationship**
- If only one entity set participates in a relation, the relationship is called as unary relationship. Here same entity set participates in relationship twice with different roles. Role names are specified above the link joining entity set and relationship set. This type of relationship set is sometimes called a recursive relationship set. There are three types of unary relationships.
- 1:1 unary relationship
- 1:M unary relationship
- M:N unary relationship

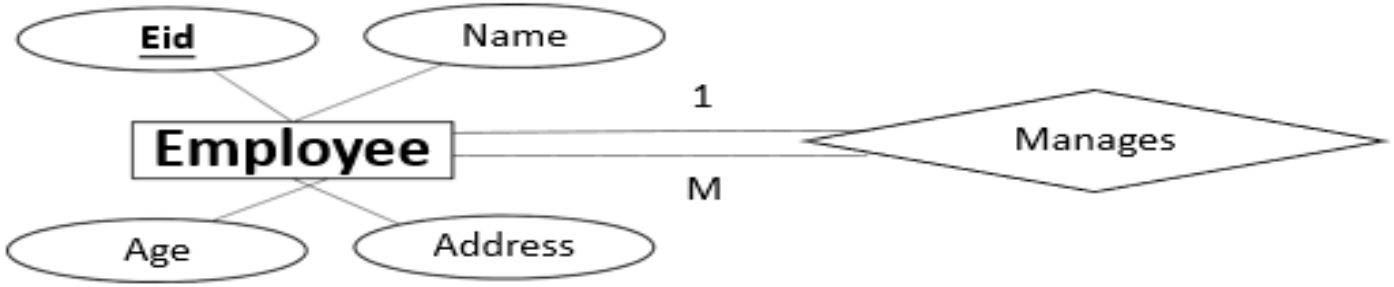
One to One (1:1) Unary Relationship

- In the example below, one person is married to only one person.



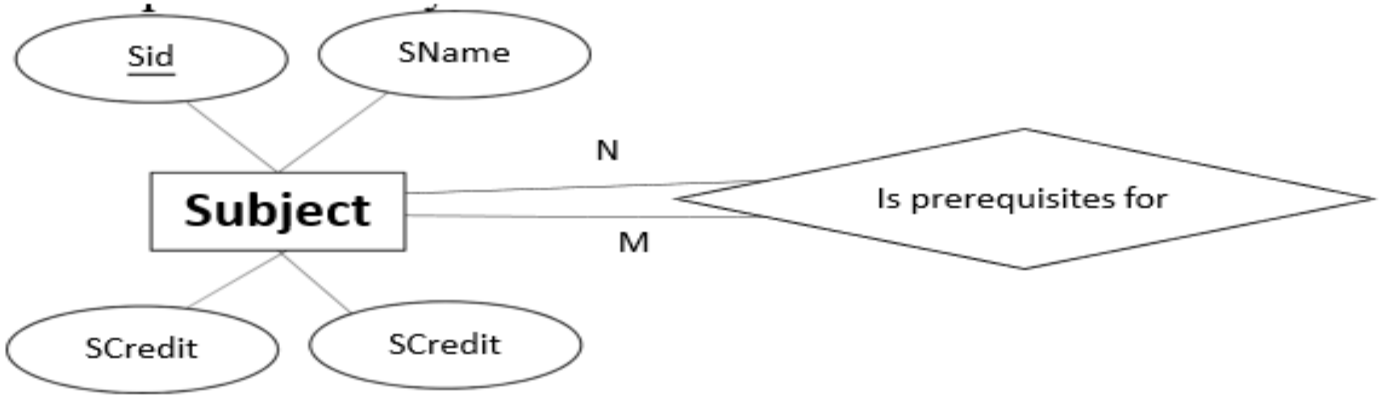
One to Many (1:M) Unary Relationship

- An employee may manage many employee but an employee is managed by only employee. This types of relationship with employee relationship set itself is called 1:M unary relationship as shown in below:



Many to Many (M:M) Unary Relationship

- A subject may have many other subjects as prerequisites and each subject may be a prerequisite to many other subjects.

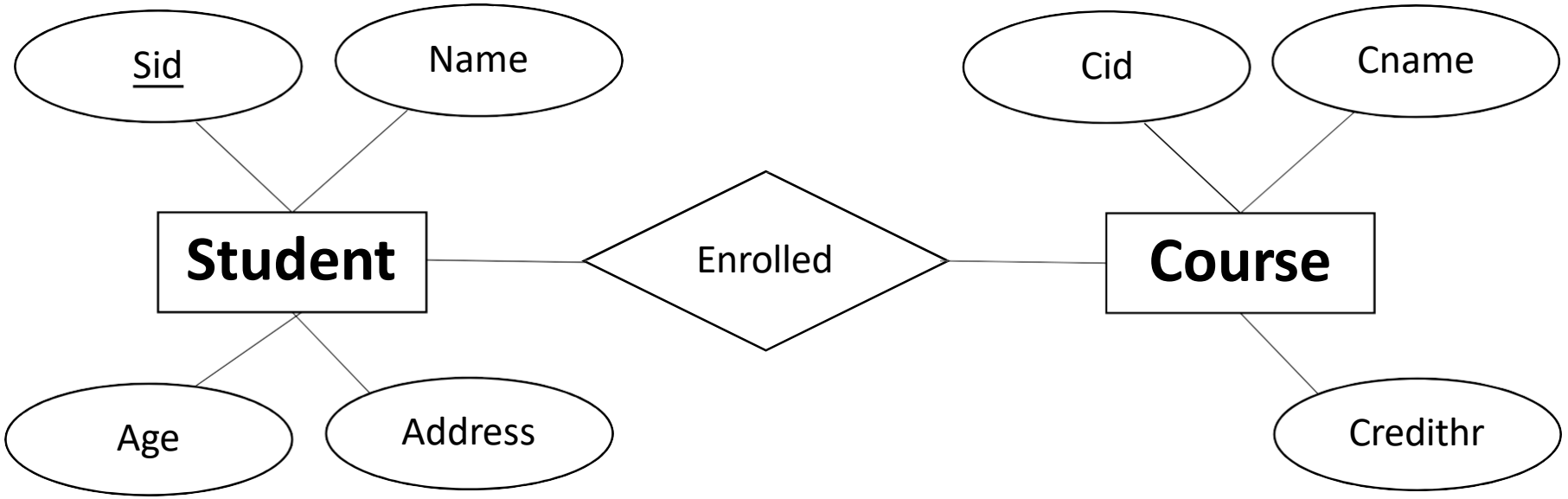


Binary Relationship

- When there are two entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course. This is the most common type of relationship in database systems.

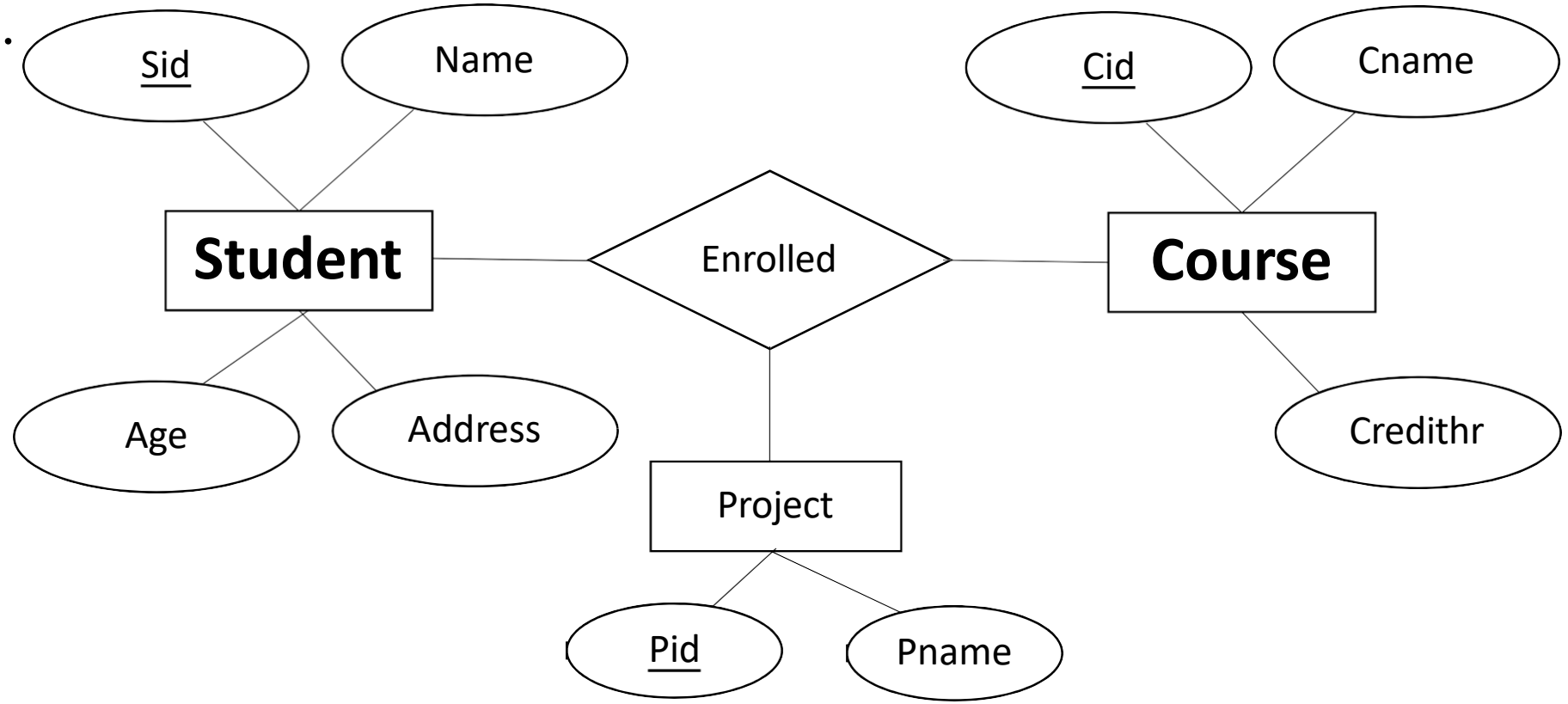
Binary Relationship

- When there are two entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course. This is the most common type of relationship in database systems.



N-ary Relationship

- When there are n entities set participating in a relation, the relationship is called n-ary relationship. For example, if n = 1 then it is called unary relationship, if n=2 then it is called binary relationship.
- Generally in N-ary relationship there are more then two entities participating with a single relationship i.e. n>2.

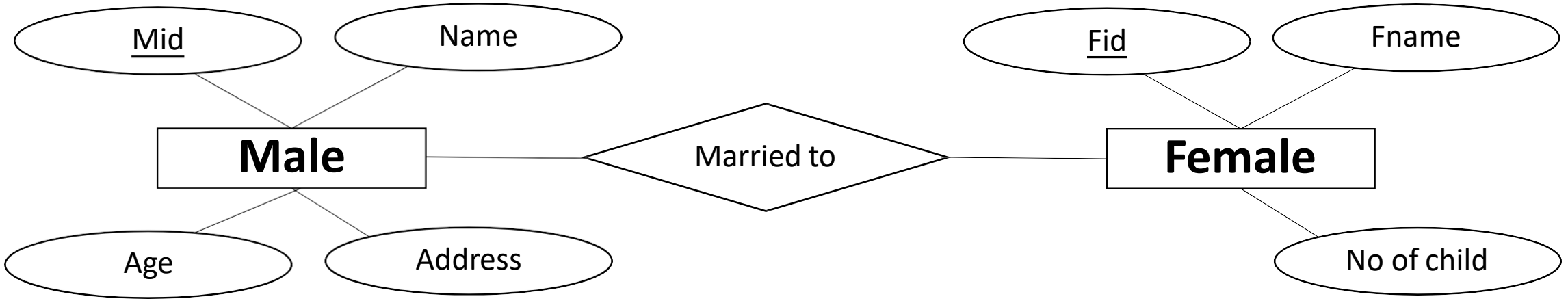


Constraints on ER Model/Structural Constraints in ER

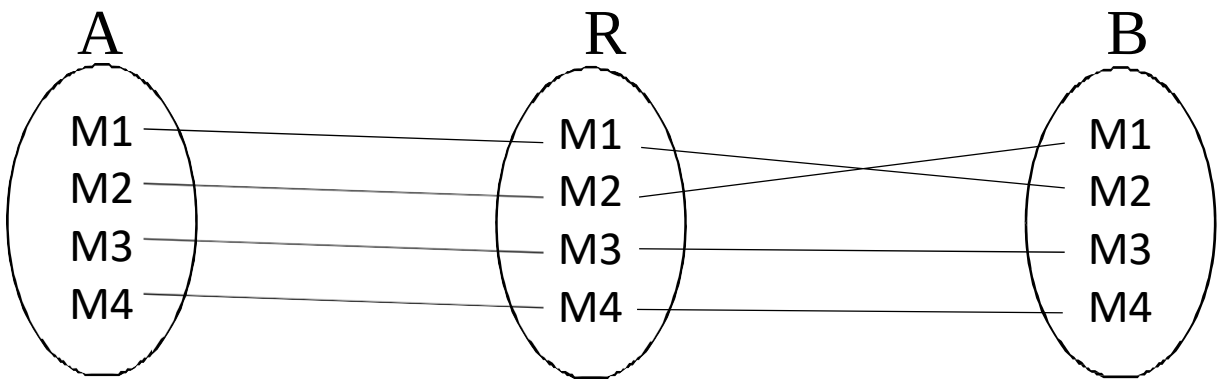
- Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involve in the corresponding relationship set. Database content must confirm these constraints. The most important structural constraints in ER are listed below:
- Mapping cardinalities and
- Participation constraints
- **Mapping Cardinality Constraints**
- The number of times an entity of an entity set participated in a relationship set is known as cardinality. Cardinality can be of different types:
- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many
- We express cardinality constraints by drawing either a directed line(→), signifying “one”, or an undirected line (-), signifying “many”, between the relationship set and the entity set.

One-to-One Mapping Cardinality Constraints

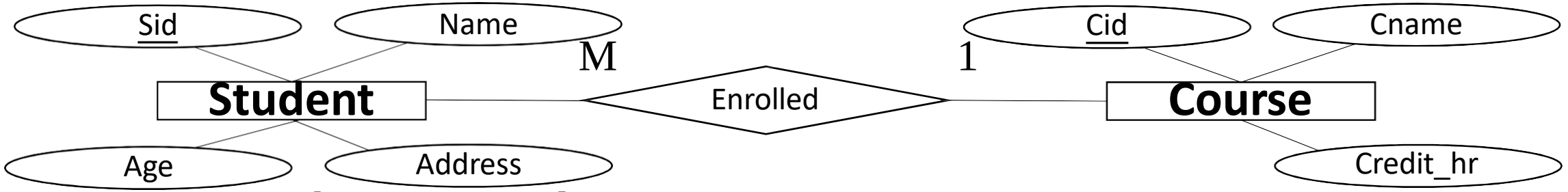
- In One-to-One mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.



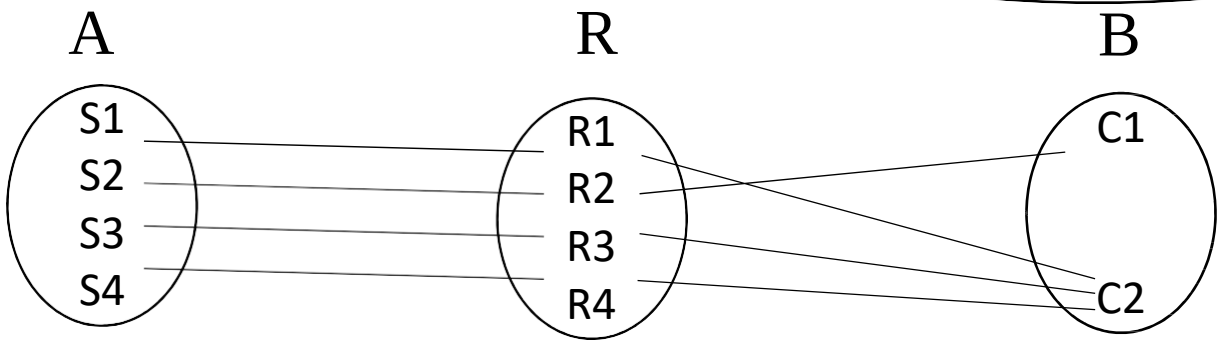
- Using Sets, it can be represented as:



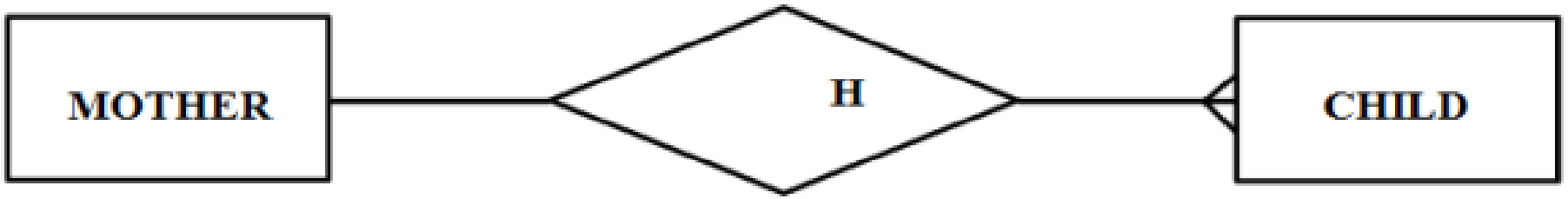
One-to-Many or Many-to-One Mapping Constraints

- In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.
- In this mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.
- Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.
- 

- Using Sets, it can be represented as:

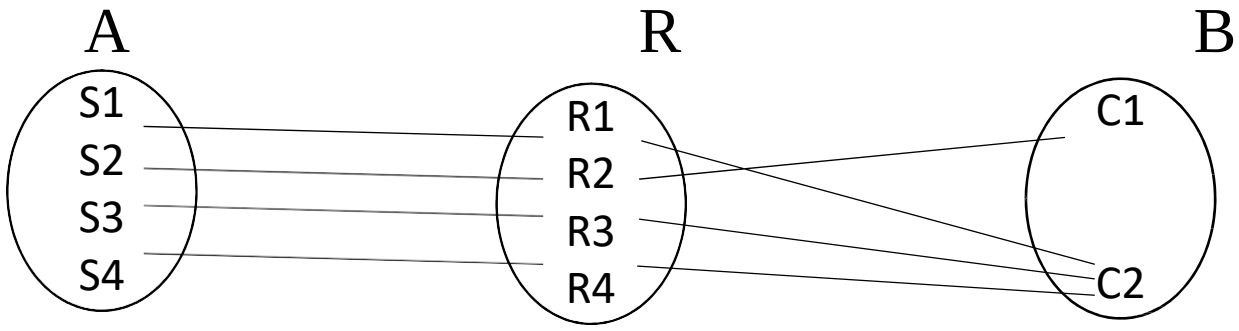


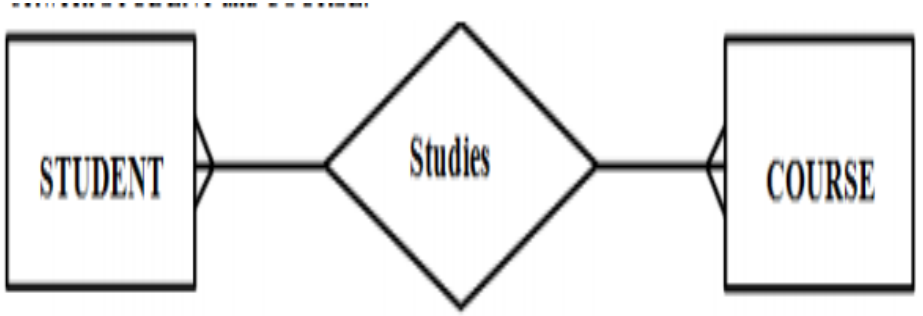
- One to Many



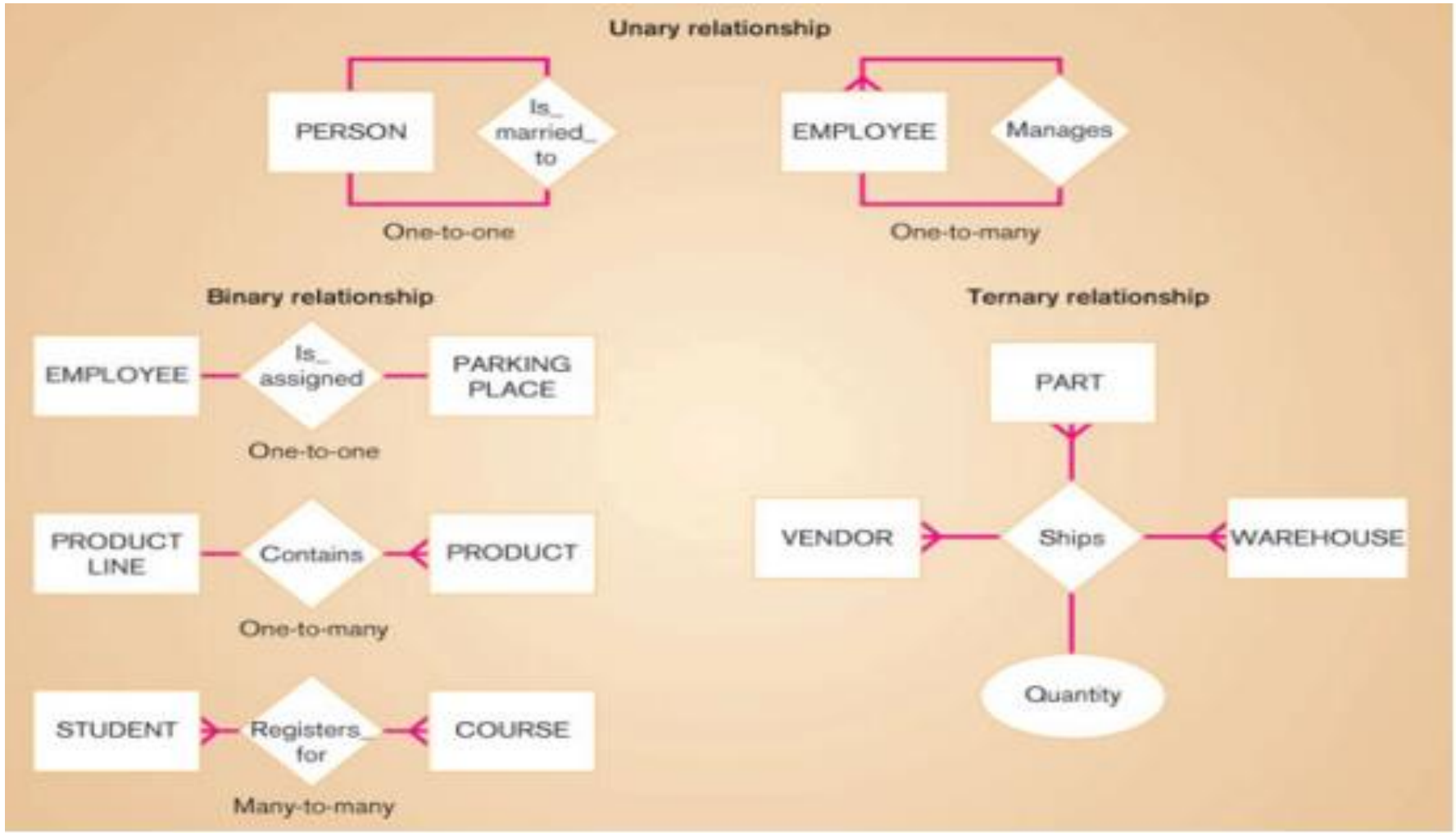
- Many to One

Many-to-Many Mapping Constraints

- In Many-to-Many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1. Let us assume that a student can take more than one course and one course can be taken by many students. So, the relationship will be many to many.
- Using Sets, it can be represented as:
- 



- In this example, student S1 is enrolled in C1 and C3 and Course C2 is enrolled by S1, S2 and S4. So, it is many to many relationships.

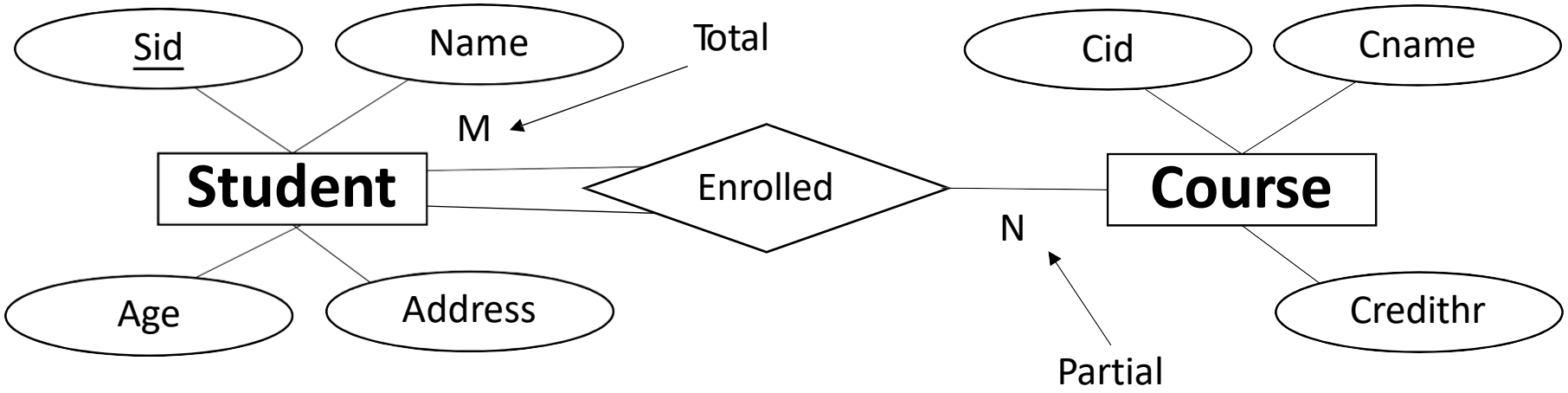

Unary relationship
PERSON
Is_ married_ to
One-to-one
EMPLOYEE
Manages
One-to-many
Binary relationship
EMPLOYEE
Is_ assigned
PARKING PLACE
One-to-one
PRODUCT LINE
Contains
PRODUCT
One-to-many
STUDENT
Registers_ for
COURSE
Many-to-many
Ternary relationship
PART
VENDOR
Ships
WAREHOUSE
Quantity

Participation Constraints

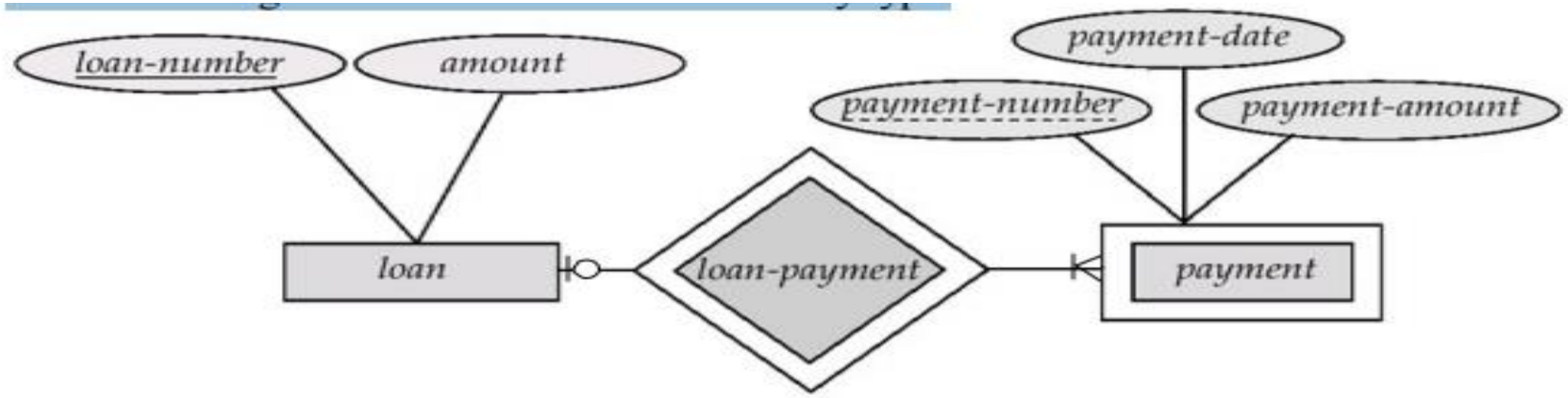
- Participation Constraint is applied on the entity participating in the relationship set. Constraint on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint.
- It specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- There are two types of participation constraints:
- Total Participation Constraints and
- Partial Participation Constraints.
- **Total Participation Constraints**
- It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why; it is also called as mandatory participation. Total participation is represented using a double line between the entity set and relationship set in ER diagram. If each student must enroll in a course, the participation of student will be total.

Participation Constraints

- **Partial Participation Constraints**
- It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why; it is also called as optional participation. Partial participation is represented using a single line between the entity set and relationship set in ER diagram. If some courses, are not enrolled by any of the student, the participation of course will be partial.
- The diagram depicts the ‘Enrolled in’ relationship set with Student Entity set having total participation and Course Entity set having partial participation.
-



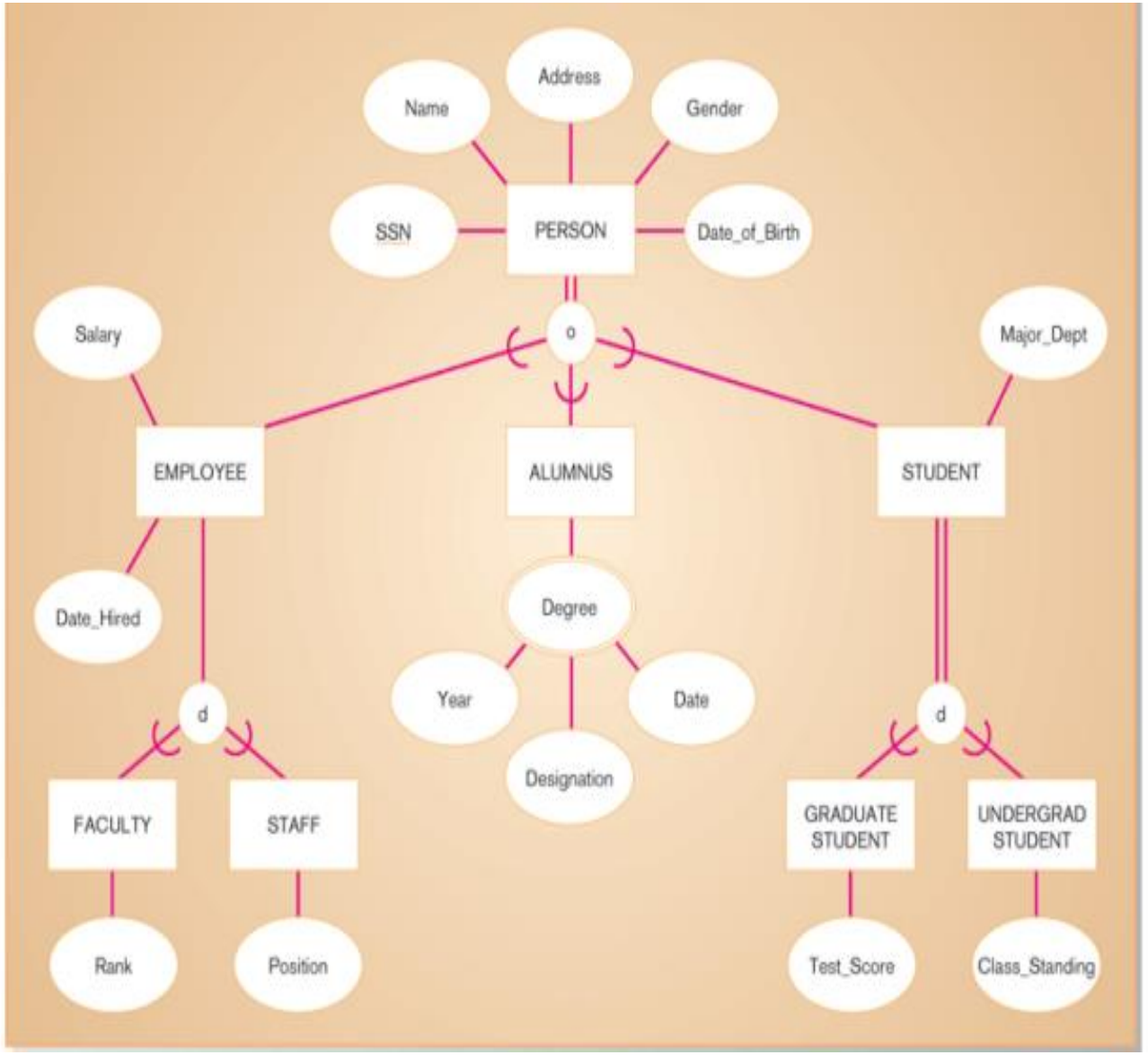
- Weak Entity Type: An entity type may not have sufficient attributes to form a primary key. Such an entity type is termed as a weak entity type. An entity type that has a primary key is termed as a strong entity type. For a weak entity type to be meaningful, it must be associated with another entity 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.



3.3.6 Representing Supertypes And Subtypes

- Often two or more entity types seem very similar (maybe they have almost the same name), but there are a few differences. That is, these entity types share common properties but also have one or more distinct attributes or relationships. To address this situation, the E-R model has been extended to include supertype/subtype relationships.
- A **subtype** is a sub grouping of the entities in an entity type that is meaningful to the organization. For example, STUDENT is an entity type in a university. Two subtypes of STUDENT are GRADUATE STUDENT and UNDERGRADUATE STUDENT. A **supertype** is a generic entity type that has a relationship with one or more subtypes.

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



3.3.7 Business Rules

Conceptual data modeling is a step-by-step process for documenting information requirements, and it is concerned with both the structure of data and with rules about the integrity of those data. Business rules are specifications that preserve the integrity of the logical data model. Four basic types of business rules are as follows:

1. **Entity integrity:** Each instance of an entity type must have a unique identifier that is not null.

2. Referential integrity constraints: Rules concerning the relationships between entity types.

3. Domains: Constraints on valid values for attributes.

4. Triggering operations: Other business rules that protect the validity of attribute values.

- **Domains:** A domain is the set of all data types and ranges of values that attributes may assume.
- **Triggering operations:** A triggering operation (also called a trigger) is an assertion or rule that governs the validity of data manipulation operations such as insert, update, and delete. The scope of triggering operations may be limited to attributes within one entity or it may extend to attributes in two or more entities.

3.3.8 Role of Packaged Conceptual Data Models

- There are two principal types of packaged data models: universal data models applicable to nearly any business or organization and industry-specific data models.
- **Universal Data Models :**

- Numerous core subject areas are common to many (or even most) organizations, such as customers, products, accounts, documents, and projects. Although they differ in detail, the underlying data structures are often quite similar for these subjects. Further, there are core business functions such as purchasing, accounting, receiving, and project management that follow common patterns.

- Universal data models are templates for one or more of these subject areas and/or functions. All of the expected components of data models are generally included: entities, relationships, attributes, primary and foreign keys, and even sample data.

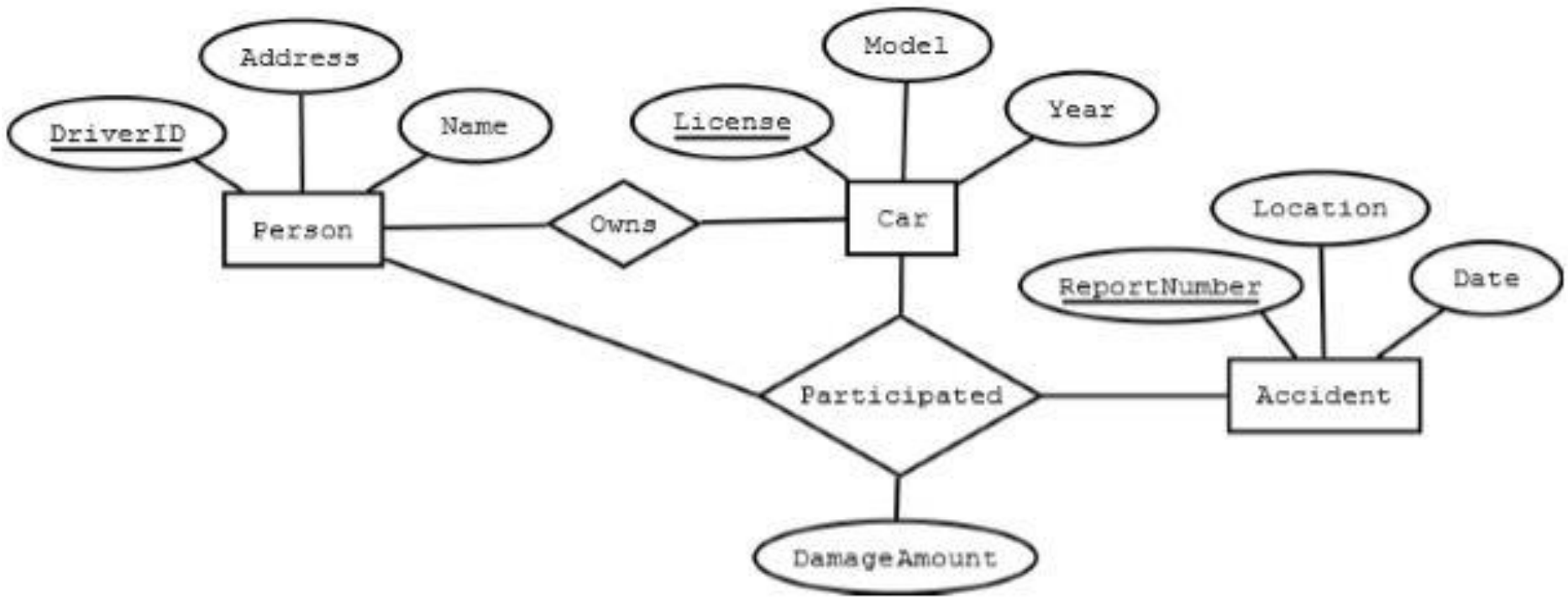
- **Industry-Specific Data Models**

  - Industry-specific data models are generic data models that are designed to be used by organizations within specific industries. Data models are available for nearly every major industry group, including health care, telecommunications, discrete manufacturing, process manufacturing, banking, insurance, and higher education.

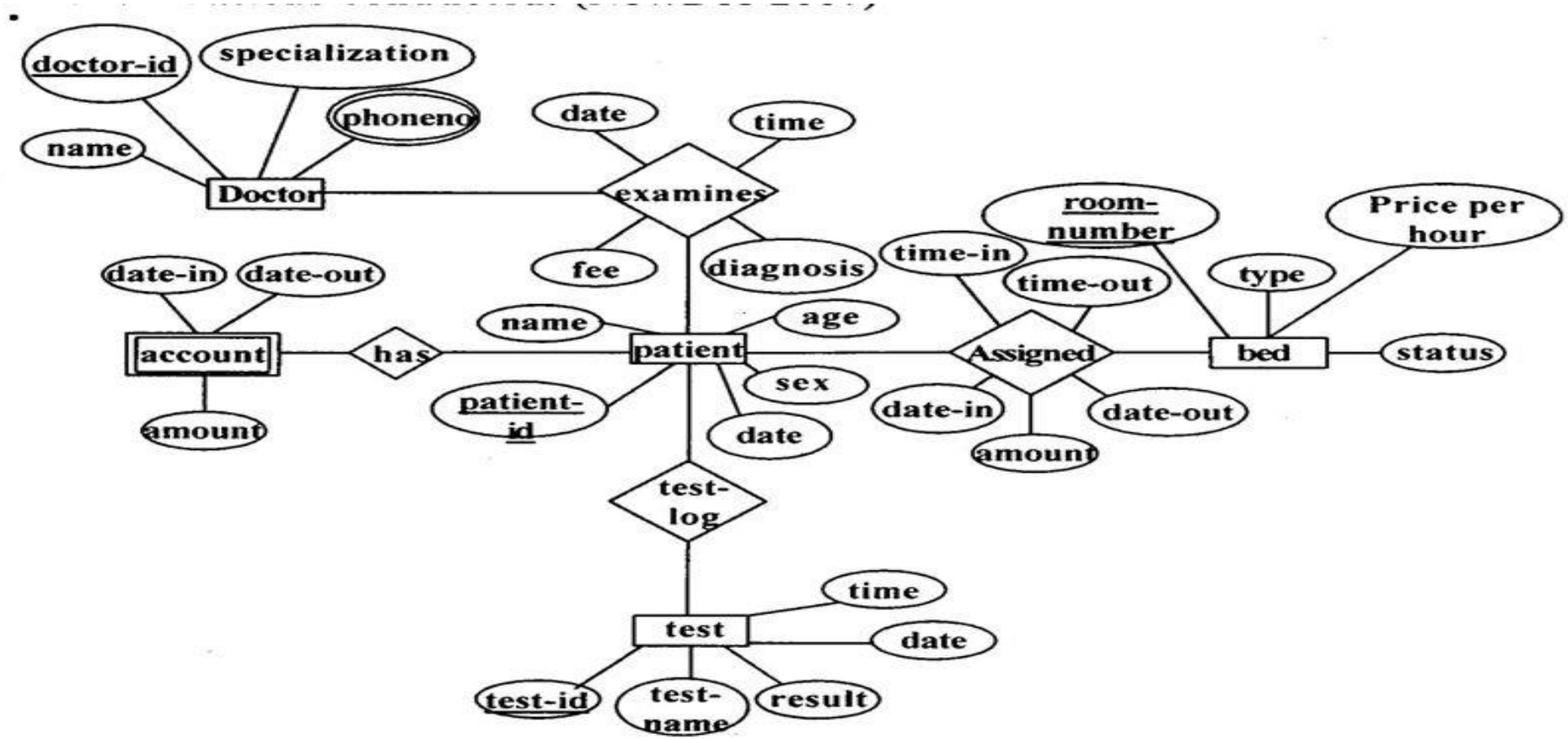
  - These models are based on the premise that data model patterns for organizations are very similar within a particular industry (“a bank is a bank”). However, the data models for one industry (such as banking) are quite different from those for another (such as hospitals).

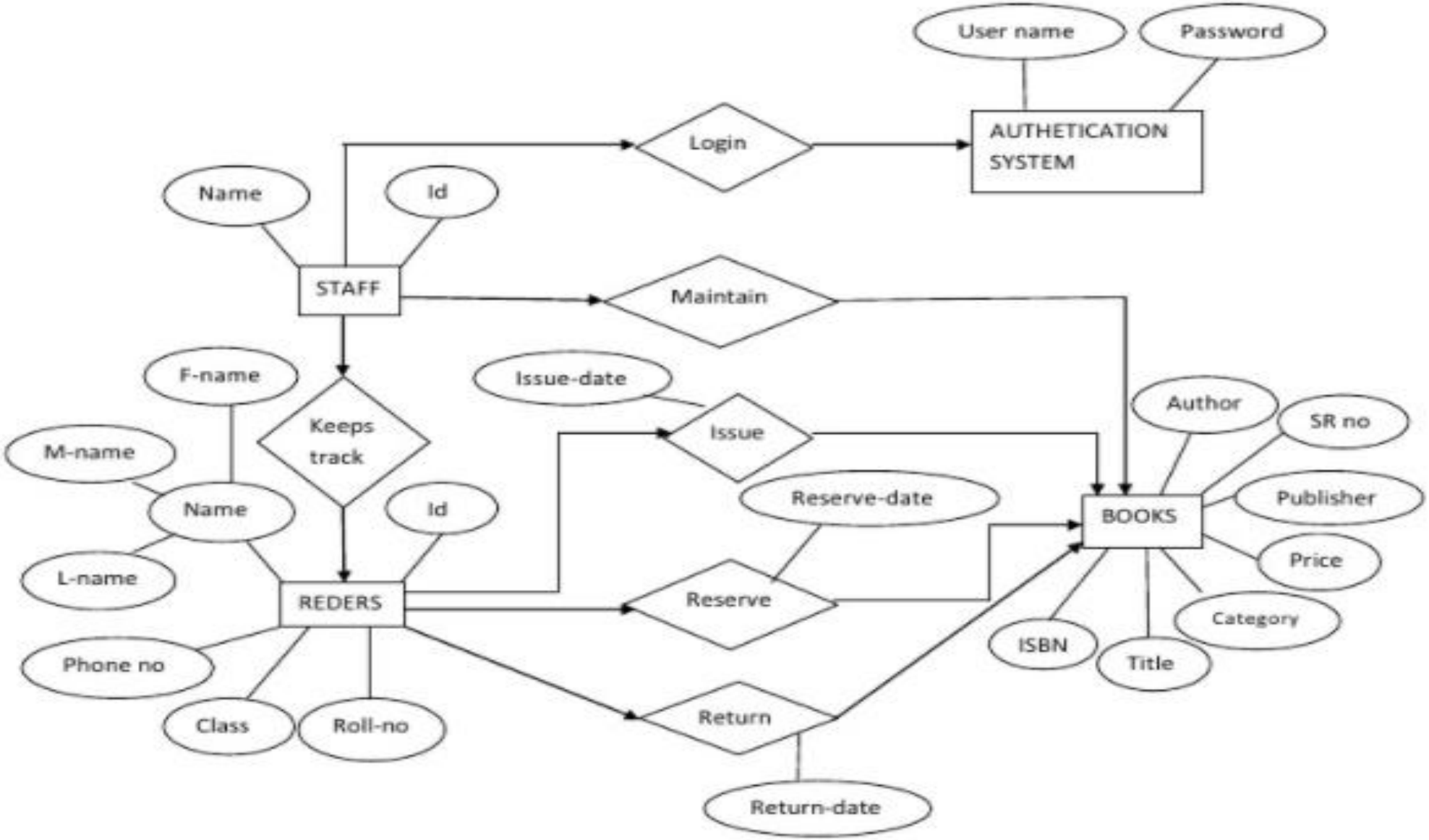
Some Exercises

1. Construct an ER diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
2. Construct an ER diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examinations conducted.
3. Construct an ER diagram of the library system in your college.


Address
DriverID
Name
Person
Owns
License
Model
Year
Car
Location
ReportNumber
Date
Participated
Accident
DamageAmount

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 various tests and examinations conducted.




User name
Password
AUTHETICATION SYSTEM
Login
Name
Id
STAFF
Maintain
F-name
Keeps track
Issue-date
Issue
Author
SR no
M-name
Name
Id
Reserve-date
BOOKS
Publisher
L-name
REDERS
Reserve
Price
Category
Phone no
ISBN
Title
Class
Roll-no
Return
Return-date

Video library management system

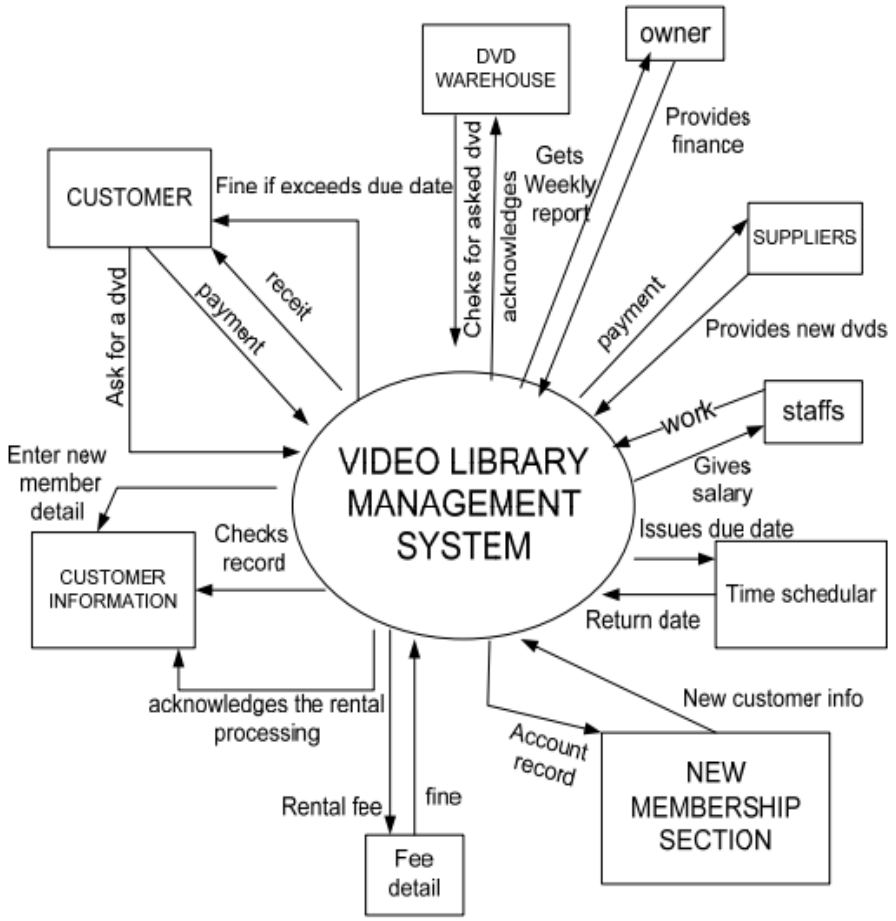


Fig: context diagram of video library management system

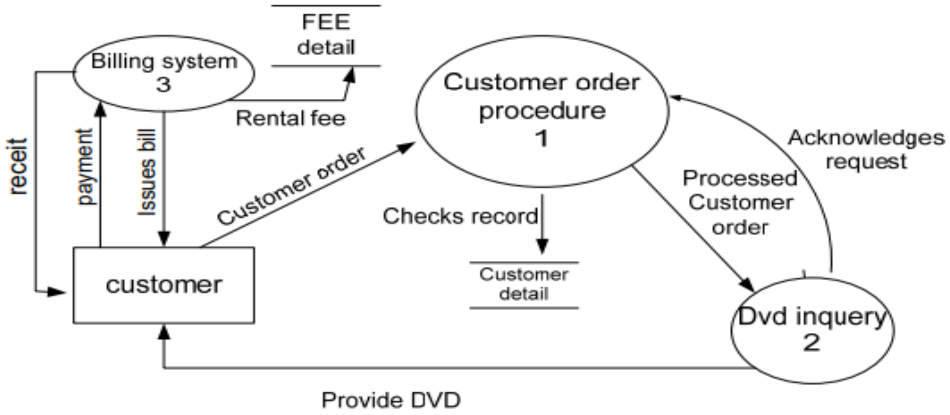


Fig : 0 level dfd of video library management system.

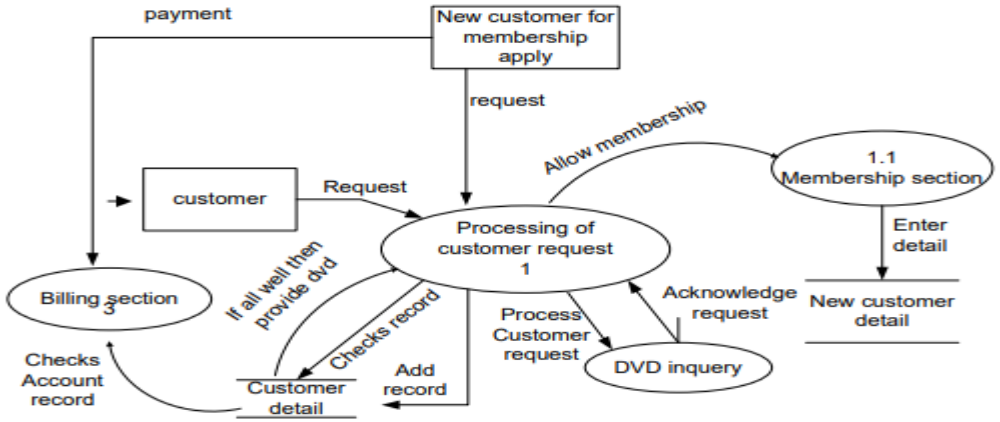
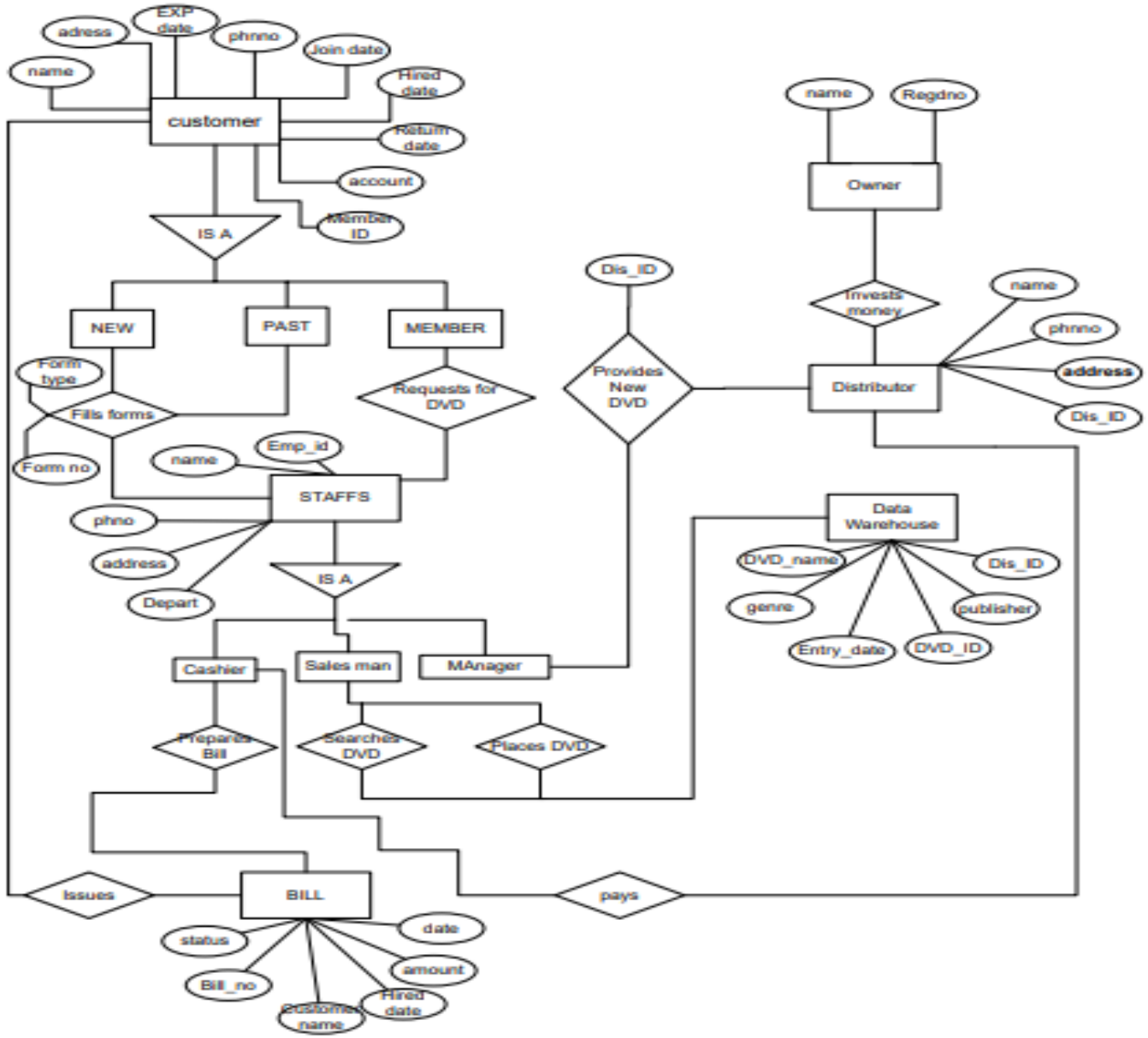


fig: level 1 dfd diagram of video library management system



ER diagram of Video library management system

RESULT MANAGEMENT SYSTEM

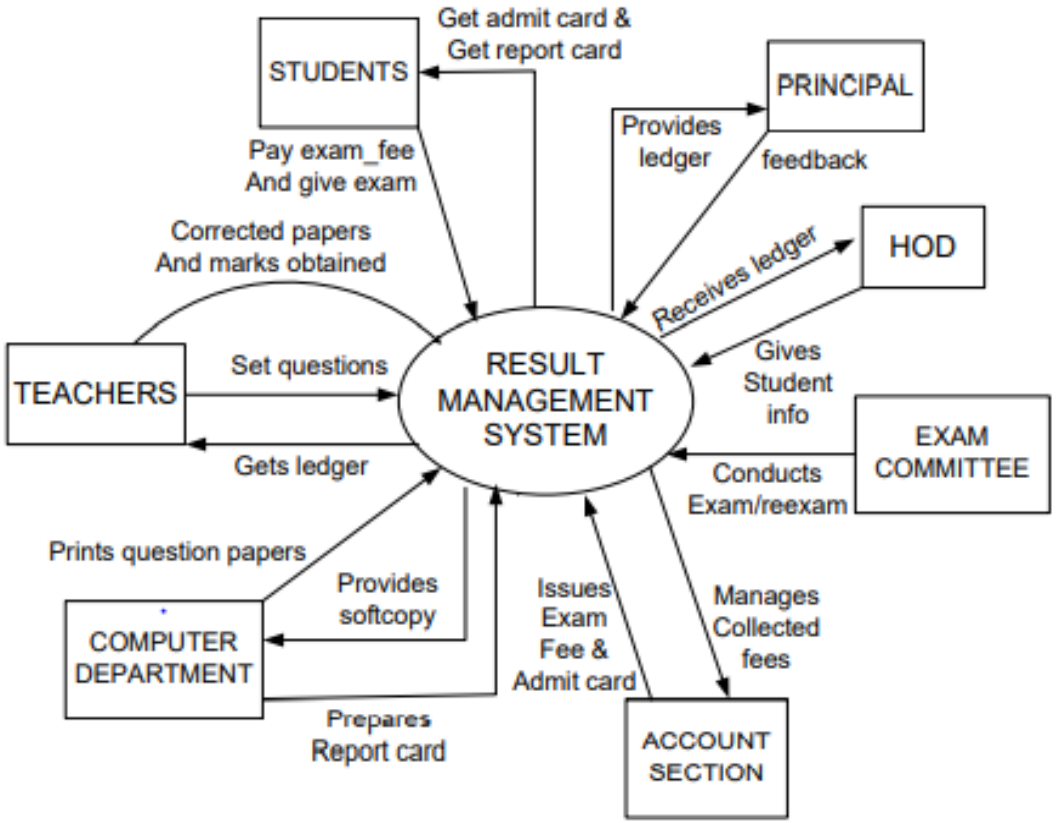


Fig: Context diagram of result management system

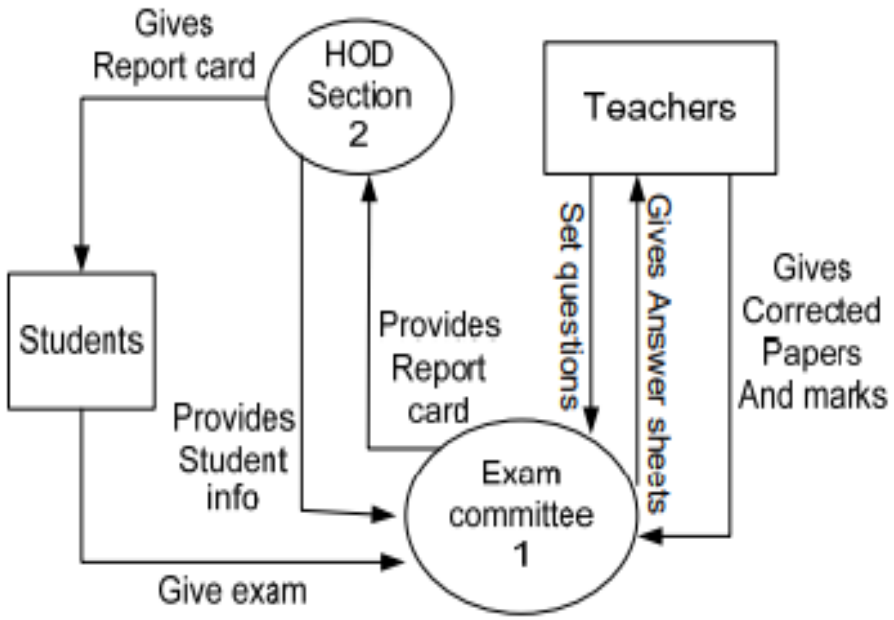


Fig: level 0 DFD for Result Management System

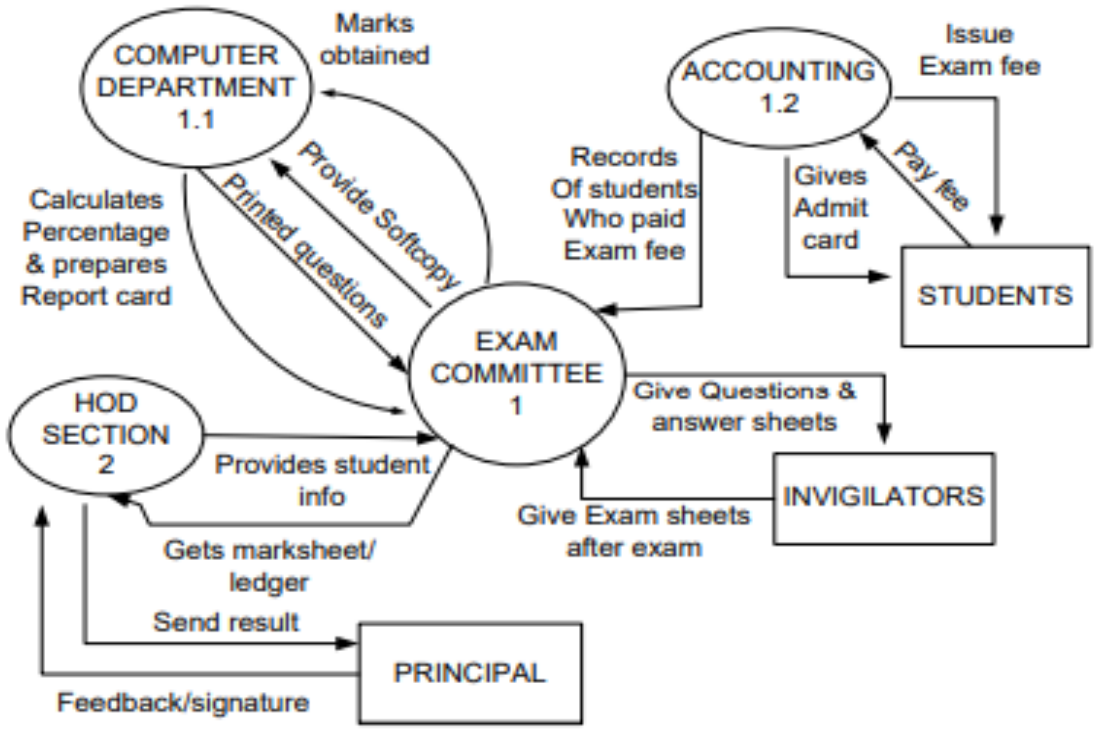


Fig: level 1 DFD for Result Management System

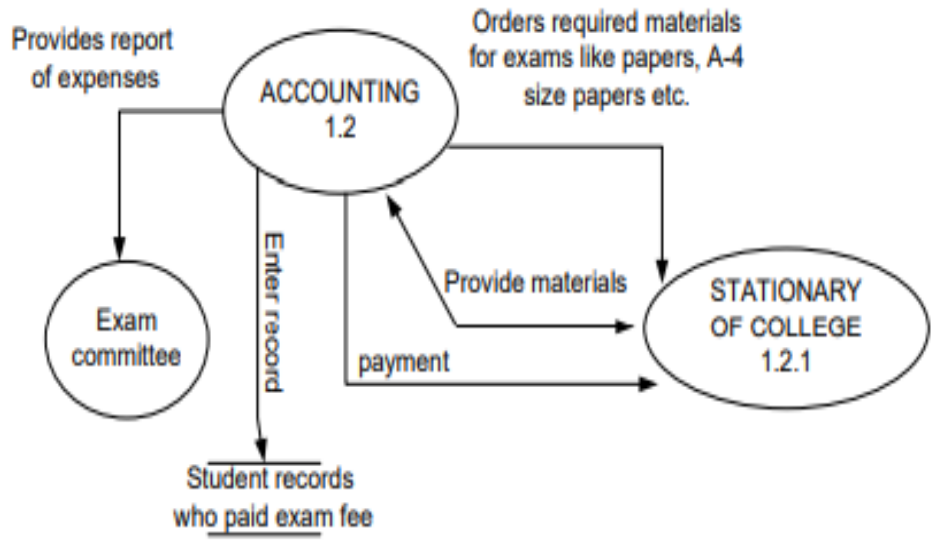
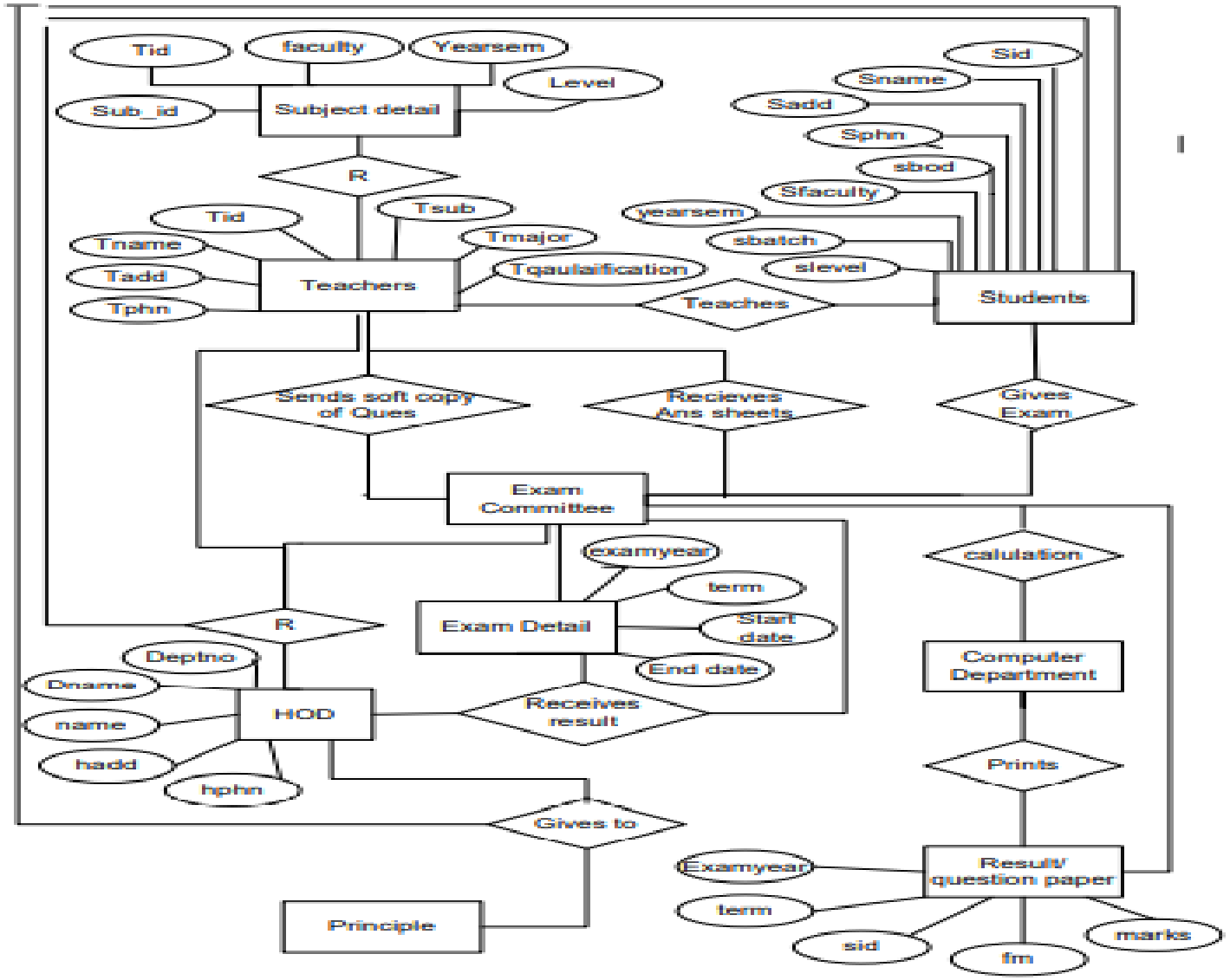


Fig: level 2 DFD of result management system



ER diagram of result management system