UNIT-II

System Design: Interface design tools, user interface- evaluations, Introduction to Process Modeling, Introduction to Data Modeling.

System Design Techniques, Document Flow Diagrams, Documents, Physical Movement of documents, Usefulness of Document Flow diagram, Data Flow

Diagrams. DFD notation, Context diagram DFD leveling, Process deseripiioas structured English, Decision Trees and Decision Tables, Entity Relationship Diagrams, Kntitics. Attributes, Relationship, Degree, Optionality, Resolving many to many relationship. Exclusive relationship, Structure Charts, Modules, Parameter passing, Execution sequence, Structured Design, Conversion from Data Flow Diagrams to Structure Charts.

Unit - II

7

FUNCTIONAL MODELING

Objectives

1. Functional Requirements :

After the thorough analysis of the system is done, the design process for the proposed system starts. Designing of the system includes making the conceptual layout of the system using various modeling techniques available and coding the proposed system for actual implementation.

The first step after the analysis of the system is making the conceptual design. Conceptual design consists of two elements: data model and functional models.

Data modeling is a technique for organizing and documenting system's data. Process model is a technique for organizing and documenting a system's processes, inputs, outputs and data stores. Generally, these models give the following information.

What all processes make up a system?

What data are used in each process?

What data are stored?

What data enter and leave the system?

Information is transformed as it flows through a computer-based system. As we already know, information transformation basically consists of: input, process and output. The system accepts input in variety of forms; applies hardware, software, and human elements to transform input into output; and produces output in a variety of forms. The transform(s) may comprise a single logical comparison, a complex numerical algorithm or rule-inference approach of an expert system. We can create a flow model for any computer based system, regardless of size and complexity. By flow models, we get to know the functionality of a system.

Data flow diagram is one of the tools used in the analysis phase. Data flow diagram is a graphical tool used to analyze the movement of data through a system-manual or automated-including the processes, stores of data, and delay in the system. The transformation of data from input to output, through processes, may be described logically and independently of the physical components (for example, computers, file cabinets, disk units, and word processors).

Structure chart is another tool used in designing phase of the life cycle.

2. Design Elements :

This section describes the various design elements. These include modules, processes, input, output, files and databases.

2.1. Modules :

A large system actually consists of various small independent subsystems that combine together to build up the large systems. While designing the system too, the complete system is divided into small independent modules which may further be divided if the need be. Such independent modules are designed and coded separately and are later combined together to make the system functional.

For the better understanding and design of the system, it should be made as a hierarchy of modules. Lower level modules are generally smaller in scope and size compared to higher level modules and serve to partition processes into separate functions. Following factors should be considered while working on modules:

Size: The number of instructions contained in a module should be limited so that module size is generally small.

Shared use: Functions should not be duplicated in separate modules, but established in a single module that can be invoked by any Other module when needed.

2.2. Processes :

As already discussed, a system consists of many subsystems working in close coordination to achieve a specific objective. Each of these subsystems carries out a specific function and each of these functions may in turn be consisting of one or more processes. Thus the system's functions can be subdivided into processes, as depicted by fig. 1. A process is a specific act that has definable beginning and ending points. A process has identifiable inputs and outputs. Create purchase requisition, follow up order etc. are few examples of processes. For designing of any system, these processes need to be identified as a part of functional modeling. Every process may be different from the other but each of them has certain common characteristics, as:

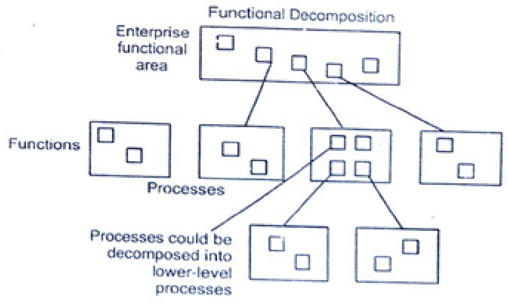

Functional Decomposition
Enterprise functional area
Functions
Processes
Processes could be decomposed into lower-level processes

Fig. 1 : Functional Decomposition

- A process is a specified activity in an enterprise that is executed repeatedly. This means that the processes are ongoing, for example, generation of bills may be labeled as a process for a warehouse as it is repeatedly carried out
- A process can be described in terms of inputs and outputs. Every process would have certain input required which are transformed into a certain output. For example, in case of a warehouse, information related to the sale of various items is required for generation of bills. This information is taken as input and the bills generated are the output of the process.
- A process has definable starting and ending points;
- A process is not based on organizational structures and is carried out irrespective of this structure.
- A process identifies what is done, not how

2.3. Input(s) and Output(s) :

As discussed earlier, inputs and outputs are an important part of any system, so while designing a system inputs and outputs of the system as a whole need to be identified and the inputs and outputs for the various processes of the system need to be listed down. During design of input, the analyst should decide on the following details:

- What data to input
- What medium to use
- How data should be arranged
- How data should be coded i.e. data representation conventions
- The dialogue to guide users in providing input i.e. informative messages that should be provided when the user is entering data. Like saying, "It is required. Don't leave it blank."
- Data items and transactions needing validation to detect errors
- Methods for performing input validation and steps to follow when errors occur

The design decisions for handling input specify how data are accepted for computer processing. The design of inputs also includes specifying the means by which end-users and system operators direct the system in performing actions.

Output refers to the results and information that are generated by the system. In many cases, output is the main reason for developing the system and the basis on which the usefulness of the system is evaluated. Most end-users will not actually operate the information system or enter data through workstations, but they will use the output from the system.

While designing the output of system, the following factors should be considered:

- Determine what information to present
- Decide on the mode of output, i.e. whether to display, print, or "speak" the information and select the output medium

- Arrange the presentation of information in an acceptable format
- Decide how to distribute the output to intended recipients

These activities require specific decisions, such as whether to use preprinted forms when preparing reports and documents, how many lines to plan on a printed page, or whether to use graphics and color. The output design is specified on layout forms, sheets that describe the location characteristics (such as length and type), and format of the column heading, etc.

2.4. Design of Databases and Files :

Once the analyst has decided onto the basic processes and inputs and outputs of the system, he also has to decide upon the data to be maintained by the system and for the system. The data is maintained in the form of data stores, which actually comprise of databases. Each database may further be composed of several files where the data is actually stored. The analyst, during the design of the system, decides onto the various file-relating issues before the actual development of the system starts.

The design of files includes decisions about the nature and content of the file itself such as whether it is to be used for storing transaction details, historical data, or reference information.

Following decisions are made during file design:

- Which data items to include in a record format within the file?
- Length of each record, based on the characteristics of the data items
- The sequencing or arrangement of records within the file (the storage structure, such as sequential, indexed, or relative)

In database design, the analyst decides upon the database model to be implemented. Database model can be traditional file based, relational, network, hierarchical, or object oriented database model.

2.5. Interfaces :

Systems are designed for human beings to make their work simpler and faster. Hence interaction of any system with the human being should be an important area of concern for any system analyst. The analyst should be careful enough to design the human element of the system in such a manner that the end user finds the system friendly to work with. Interface design implies deciding upon the human computer interfaces. How the end user or the operator will interact with the system. It includes designing screens, menus, etc.

The following factors should be considered while working on interfaces.

- Use of a consistent format for menu, command input, and data display.
- Provide the user with visual and auditory feedback to ensure that two-way communication is established.
- Provide undo or reversal functions.
- Reduce the amount of information that must be memorized between actions.

- Provide help facilities that are context sensitive.
- Use simple action verbs or short verb phrases to name commands.
- Display only that information that is relevant to the current context.
- Produce meaningful error messages.
- Use upper and lower case, indentation, and text grouping to aid in understanding.
- Produce meaningful error messages.

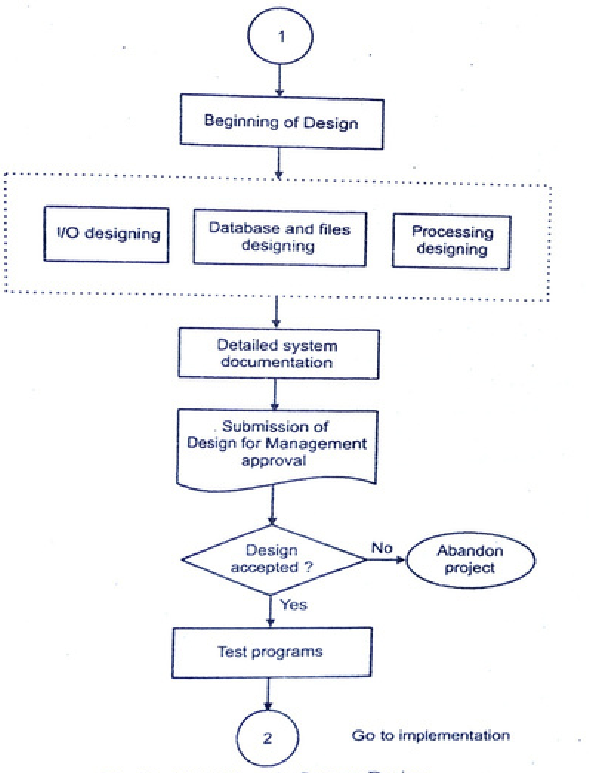


Fig. 2 : Basic Steps in System Design

Maintain consistency between information display and data input. The visual characteristics of the display (e.g., text size, color, and placement) should be carried over to the input domain.

Interaction should be flexible but also tuned to user's preferred mode of input.

Deactivate commands that are inappropriate in the context of current actions.

Provide help to assist with all input actions.

3. Functional Modeling Techniques :

Now that we are familiar with the various design elements, let us take a look at the modeling techniques that are used for designing the systems. Data Flow Diagrams are used for functional modeling. As the name suggests, it is a diagram depicting the flow of data through the system. In the next section, we'll explore this technique in detail.

3.1. Data Flow Diagrams :

A Data Flow Diagram (DFD) is used to describe operation of a system, i.e. what a system does. DFD shows the flow of data through a system and the work or processing performed by that system

3.1.1. Elements of Data Flow Diagrams :

Data Flow Diagrams are composed of the four basic symbols shown below :

- The External Entity symbol represents sources of data to the system or destinations of data from the system.
- The Data Flow symbol represents movement of data.
- The Data Store symbol represents data that is not moving (delayed data at rest).
- The Process symbol represents an activity that transforms or manipulates the data (combines, reorders, converts, etc.).

Any system can be represented at any level of detail by these four symbols.

3.1.1.1. Processes : Processes are work or actions performed on incoming data flows to produce outgoing data flows. These show data transformation or change. Data coming into a process must be "worked on" or transformed in some way. Thus, all processes must have inputs and outputs. In some (rare) cases, data inputs or outputs will only be shown at more detailed levels of the diagrams. Each process in always "running" and ready to accept data.

Major functions of processes are computations and making decisions. Each process may have dramatically different timing: yearly, weekly, daily.

Naming Processes : Processes are named with one carefully chosen verb and an object of the verb. There is no subject. Name is not to include the word "process". Each process should represent one function or action. If there is an "and" in the name, you likely have more than one function (and process). For example, get invoice ,update customer and create Order

Processes are numbered within the diagram as convenient. Levels of detail are shown by decimal notation. For example, top level process would be Process 14, next level of detail Processes 14.1-14.4, and next level with Processes 14.3.1-14.3.6. Processes should generally move from top to bottom and left to right.

3.1.1.2. External Entities : External entities determine the system boundary. They are external to the system being studied. They are often beyond the area of influence of the developer. These can represent another system or subsystem. These go on margins/edges of data flow diagram. External entities are named with appropriate name.

3.1.1.3. Data Flow : Data flow represents the input (or output) of data to (or from) a process ("data in motion"). Data flows only data, not control. Represent the minimum essential data the process needs. Using only the minimum essential data reduces the dependence between processes. Data flows must begin and/or end at a process.

Data flows are always named. Name is not to include the word "data". Should be given unique names. Names should be some identifying noun. For example, order, payment, complaint.

3.1.1.4. Data Stores : Data Stores are repository for data that are temporarily or permanently recorded within the system. It is an "inventory" of data.

These are common link between data and process models. Only processes may connect with data stores.

There can be two or more systems that share a data store. This can occur in the case of one system updating the data store, while the other system only accesses the data.

Data stores are named with an appropriate name, not to include the word "file", Names should consist of plural nouns describing the collection of data. Like customers, orders, and products. These may be duplicated. These are detailed in the data dictionary or with data description diagrams.

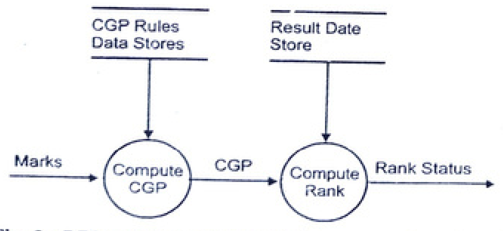


Fig. 3 : DFD showing rank calculation process of a university

3.2. Different Levels of DFDs :

The DFD may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional details.

A 0 level DFD, also called a context model, represents the entire software elements as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes and information flow paths are

represented as the 0 level DFD is partitioned to reveal more information. For example, a I level DFD might contain five or six bubbles with interconnecting arrows.

3.3. Making DFDs :

Data Flow Diagramming is a means of representing a system at any level of detail with a graphic network of symbols showing 'data flows, data stores, data processes, anti data sources/destinations.

The goal of data flow diagramming is to have a commonly understood model of a system. The diagrams are the basis of structured systems analysis. Data flow diagrams are supported by other techniques of structured systems analysis such as structure diagrams, data dictionaries, and procedure-representing techniques such as decision tables, decision trees, and structured English.

The purpose of data flow diagrams is to provide a semantic bridge between users and systems developers. The diagrams are graphical, eliminating thousands of words. These are logical representations, modeling what a system does, rather than physical models showing how it does it. DFDs are hierarchical, showing systems at any level of detail. Finally, it should be jargonless, allowing user understanding and reviewing. Also, data flow diagrams have the objective of avoiding the cost of:

- user/developer misunderstanding of a system, resulting in a need to redo systems or in not using the system.
- having to start documentation from scratch when the physical system changes since the logical system, WHAT gets done, often remains the same when technology changes.
- systems inefficiencies because a system gets "computerized" before it gets "systematized".
- being unable to evaluate system project boundaries or degree of automation, resulting in a project of inappropriate scope.

Notation : Data flow analysis was developed and promoted simultaneously by two organizations. Yourdon Inc., a consulting firm, has vigorously promoted the method publicly. Mc Donnell-Douglas, through the work and writings of Gane and Sarson, has also influenced the popularity of data flow analysis.

Data flow diagram can be completed using only four simple notations. The use of specific icons associated with each element depends on whether the Yourdon or Gane and Sarson approach is used.

1) People, procedures, or devices that use or product (transform) data. The physical component is not identified

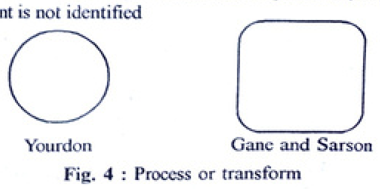


Fig. 4 : Process or transform

2) Data flow: Data move in a specific direction from origin to a destination in the form of a document, letter, telephone call, or virtually any other medium. The data flow is a "packet" of data.

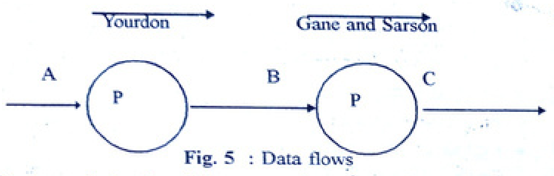


Fig. 5 : Data flows

3) Source or destination of data: external sources or destination of data, which may be people, programs, organizations, or other entities, interact with the system but are outside its boundary. The term source and sink are interchangeable with origin and destination.

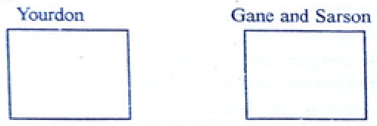


Fig. 6 : Source or destination of data (external)

4) Here data are stored or referenced by a process in the system. The data store may represent computerized or non-computerized devices.

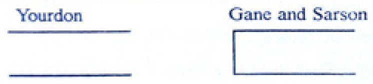


Fig. 7 : Data stores

Each component in a data flow diagram is labeled with a descriptive name. Process names are further identified with a number that will be used for identification purposes. The number assigned to a specific process does not represent the sequence of processes.

Multiple input data streams and multiple data output streams are possible.

If two adjacently placed input are both required then a star sign (*) is placed between these.

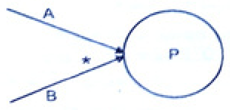


Fig. 8 Star Sign when two inputs are required for a transform

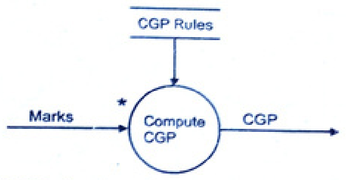


Fig. 9 :DFD showing when two inputs are both required.

If either of two adjacent .placed inputs then a ring plus is placed between these.

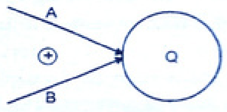


Fig. 10 : Ring Plus when either of two inputs is required

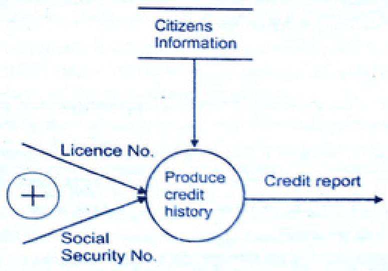


Fig. 11 : DFD showing when either of two inputs is required.

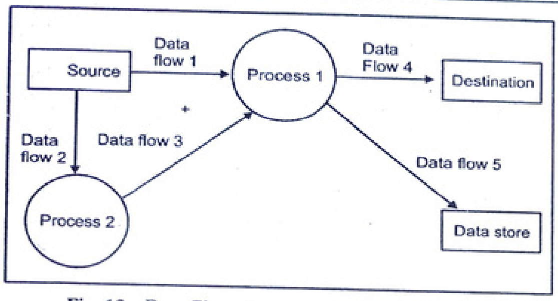


Fig. 12 : Data Flow diagram using Yourdon notation

The procedure for producing a data flow diagram is to:

- identify and list external entities providing inputs/receiving outputs from system;
- identify and list inputs from/outputs to external entities;
- Draw a context DFD

  Defines the scope and boundary for the system and project
  1. Think of the system as a container (black box)
  2. Ignore the inner workings of the container
  3. Ask end-users for the events the system must respond to
  4. For each event, ask end-users what responses must be produced by the system
  5. Identify any external data stores
  6. Draw the context diagram—

     Use only one process

     Only show those data flows that represent the main objective or most common inputs/outputs
- identify the business functions included within the system boundary;
- identify the data connections between business functions;
- confirm through personal contact sent data is received and vice-versa;
- trace and record what happens to each of the data flows entering the system (data movement, data storage, data transformation/processing.
- Draw an overview DFD :

  Shows the major subsystems and how they interact with one another

  Exploding processes' should add detail while retaining the essence of the details from the more general diagram

  Consolidate all data stores into a composite data store

- Draw middle-level DFDs

  Explode the composite processes
- Draw primitive-level DFDs

  Detail the primitive processes

  Must show all appropriate primitive data stores and data flows
- verify all data flows have a source and destination;
- verify data coming out of a data store goes in;
- review with "informed".
- explode and repeat above steps as needed.

Balancing DFDs :

- Balancing: child diagrams must maintain a balance in data content with their parent processes
- Can be achieved by either:
  - ❑ exactly the same data flows of the parent process enter and leave the child diagram, or
  - ❑ the same net contents : from the parent process serve as the initial inputs and final outputs for the child diagram or
  - ❑ the data in the parent diagram is split in the child diagram

Rules for Drawing DFDs :

- A process must have atleast one input and one output data flow
- A process begins to perform its tasks as soon as it receives the necessary input data flows
- A primitive process performs a single well-defined function
- Never label a process with an IF-THEN statement
- Never show time dependency directly on a DFD
- Be sure that data stores, data flows, data processes have descriptive titles. Processes should use imperative verbs to project action.
- All processes receive and generate at least one data flow.
- Begin/end data flows with a bubble.

Rules for Data Flows :

1. A data store must always be connected to a process
2. Data flows must be named
3. Data flows are named using nouns
   - Customer ID, Student information
4. Data that travel together should be one data flow
5. Data should be sent only to the processes that need the data

Use the Following Additional Guidelines when Drawing DFDs. :

Identify the key processing steps in a system. A processing step is an activity that transforms one piece of data into another form. .

Process bubbles should be arranged from top left to bottom right of page.

Number each process (1 0, 2.0, etc.). Also name the process with a verb that describes the information processing activity.

Name each d flow with a noun that describes the information going into and out of a process. What goes in should be different from what comes out.

Data stores, sources and destinations are also named with nouns.

Realize that the highest level DFD is the context diagram. It summarizes the entire system as one bubble and shows the inputs and outputs to a system

Each lower level DFD must balance with its higher level DFD. This means that no inputs and outputs are changed.

Think of data flow not control flow. Data flows are pathways for data. Think about what data is needed to perform a process or update a data store. A data flow diagram is not a flowchart and should not have loops or transfer of control. Think about the data flows, data processes, and data storage that are needed to move a data structure through a system.

Do not try to put everything you know on the data flow diagram. The diagram should serve as index and outline. The index/outline will be "fleshed out" in the data dictionary, data structure diagrams, and procedure specification techniques.

Examples :

1. Students wish to register for courses. Some courses, have a prerequisite, which must be satisfied. A student must take the compulsory courses of her/his 2. A student can withdraw within 21 days of registration. Enhance the above DFD.

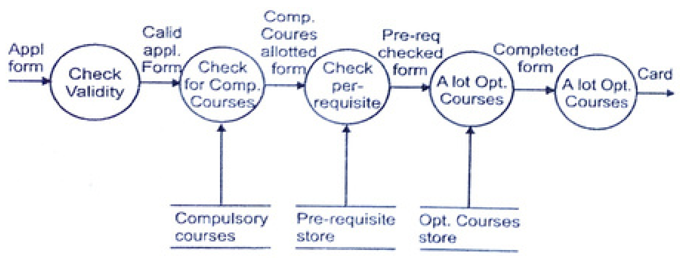


Fig. 13 : DFD for a registration for university's new semester

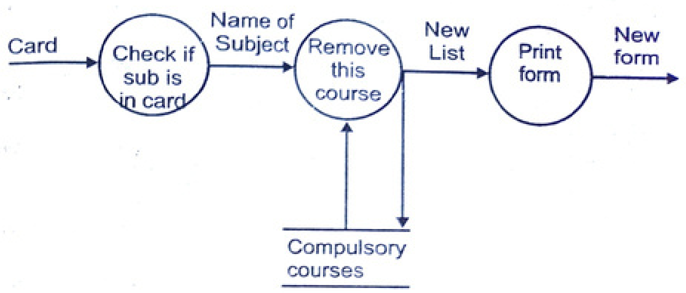


Fig. 14 : DFD when the student has the option of withdrawing within 21 days

Exercise: Railway Reservation System :

Railway caters to the need of passenger. Tickets can be booked for different classes. Some concessions are available for categories like students, old people, etc. there is a special category for reserving tickets for handicapped people, military, MPs. There is a certain surcharge levied for special trains. Fare computation depends on the class, category, train, and distance. The passenger is issued a confirmed ticket if seat is available. She/he gets a wait listed/RAC ticket if she/he so desires.

DFDs pay a major role in designing of the software and also provide the basis for other design-related issues. All the basic elements of DFD are further addressed in the designing phase of the development procedure.

A data model organizes data elements and standardizes how the data elements relate to one another. Since data elements document real life people, places and things and the events between them, the data model represents reality, for example a house has many windows or a cat has two eyes. Computers are used for the accounting of these real life things and events and therefore the data model is a necessary standard to ensure exact communication between human beings.

A data model is a set of symbols and text used for communicating a precise representation of an information landscape. As with a model of any landscape, such as a map that models a geographic landscape, certain content is included and certain content excluded to facilitate understanding.

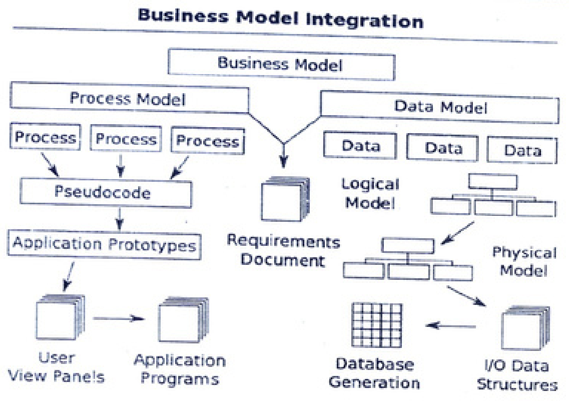


Fig. : 15

Data model is based on Data, Data relationship, Data semantic and Data constraint. A data model provides the details of information to be stored, and is of primary use when the final product is the generation of computer software code for an application or the preparation of afunctional specification to aid a computer software make-or-buy decision. The figure is an example of the interaction between process and data models.

Data models are often used as an aid to communication between the business people defining the requirements for a computer system and the technical people defining the design in response to those requirements. They are used to show the data needed and created by business processes.

According to **Hoberman** (2009), "A data model is a **way finding** tool for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment."

A data model explicitly determines the structure of data. Data models are specified in a **data modeling** notation, which is often graphical in form A data model can be sometimes referred to as a **data structure**, especially in the context of **programming languages**. Data models are often complemented by function models, especially in the context of **enterprise models.**

Types of data models

Database model

A database model is a specification describing how a database is structured and used. Several such models have been suggested. Common models include:

Flat model

This may not strictly qualify as a data model. The flat (or table) model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another.

Hierarchical model

In this model data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.

Network model

This model organizes data using two fundamental constructs, called records and sets. Records contain fields, and sets define one-to-many relationships between records: one owner, many members.

Relational model

is a database model based on first-order predicate logic. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values.

Object-relational model

Similar to a relational database model, but objects, classes and inheritance are directly supported in **database schemas** and in the query language.

Star schema

The simplest style of data warehouse schema. The star schema consists of a few "fact tables" (possibly only one, justifying the name) referencing any number of "dimension tables". The star schema is considered an important special case of the **snowflake schema.**

DFD

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an **information system**, modelling *its process* aspects.DFDs can also be used for the **visualization** of data processing (structured design).

A two-dimensional diagram that graphically representation of the "flow" of data through an **information system**, modelling *its process* aspects and explains how data is processed and transferred in a system. The graphical depiction identifies each source of data and how it interacts with other data sources to reach a common output. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated .

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored While making DFD we should

(1) identify external inputs and outputs, (2) determine how the inputs and outputs relate to each other, and (3) explain with graphics how these connections relate and what they result in.

The Food Ordering System Example

Context DFD

The figure below shows a context Data Flow Diagram that is drawn for a Food Ordering System. It contains a process (shape) that represents the system to model, in this case, the *"Food Ordering System"*. It also shows the participants who will interact with the system, called the external entities. In this example, *Supplier, Kitchen, Manager and Customer are* the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.

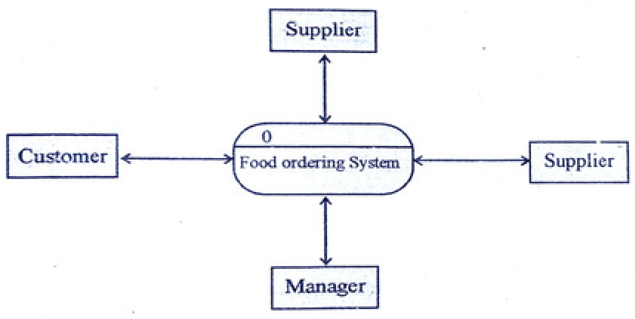


Fig. 16

Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the Food Ordering System process shown in the context DFD. Read through the diagram and then we will introduce some of the key concepts based on this diagram.

The Food Order System Data Flow Diagram example contains three processes, four external entities and two data stores.

Based on the diagram, we know that a *Customer can* place an *Order*. The *Order Food* process receives the *Order*, forwards it to the *Kitchen*, store it in the Order data store, and store the updated *Inventory details* in the *Inventory* data store. The process also deliver a *Bill* to the *Customer*.

Manager can receive *Reports* through the *Generate Reports* process, which takes *Inventory details* and *Orders* as input from the *Inventory and Order data* store respectively.

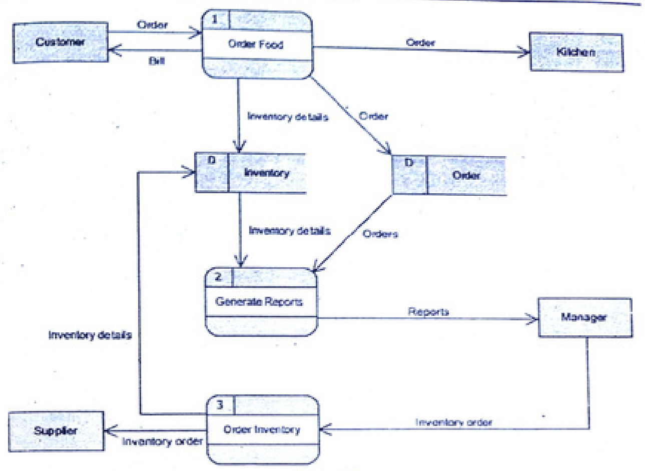


Fig. 17

Manager can also initiate the *Order Inventory process* by providing *Inventory order*. The process forwards the *Inventory orderio* the *Supplier and* stores the updated *Inventory details* in the *Inventory* data store.

4. Process Specification :

A process specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. The process specification describes the input to a function, the algorithm, the PSPEC indicates restrictions and limitations imposed on the process (function), performance characteristics that are relevant to the process, and design constraints that may influence the way in which the process will be implemented. In other words, process specification is used to describe the inner workings of a process represented in a flow diagram.

5. Control Flow Model :

The Hatley and Pirbhai extensions focus on the representation and specification of the control-oriented aspects of the software. Moreover, there exists a large class of applications that are driven by events rather than data that produce control information rather than reports or displays, and that process information with heavy concern for time and performance. Such an application requires the use of control flow modeling in addition to data flow modeling. For this purpose, control flow diagram is created. The CFD contain the same processes as the DFD, but shows control rather than data flow. Control flow diagrams show how events flow among processes and illustrate those external events that cause various processes to be activated. The relationship between process and control model is shown in Fig. 15.

Drawing a control flow model is similar to drawing a data flow diagram. A data flow model is stripped of all data flow arrows. Events and control items are then

added to the diagram a "window" (a vertical bar) into the control specification is shown.

6. Control Specifications :

The CSPEC is used to indicate (I) how the software behaves when an event or control signal is sensed and (2) which processes are invoked as a consequence of the occurrence of the event. The control specification (CSPEC) contains a number of important modeling tools.

The control specification represents the behavior of the system in two ways. The CSPEC contains a state transition diagram that is sequential specification of behavior. It also contains a process activation table (P AT) -a combinatorial specification of behavior.

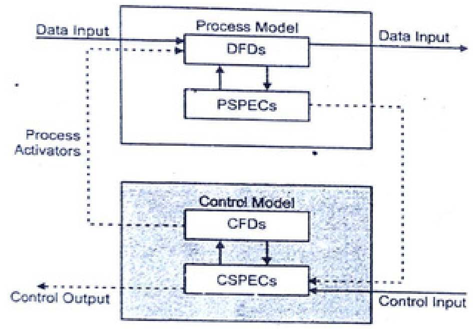


Fig. 18 : The relationship between data and control models

7. Structure Charts :

Once the flow of data and control in the system is decided using tools like DFDs and CFDs, the system is given shape through programming. Prior to this, the basic infrastructure of the program layout is prepared based on the concepts of modular programming. In modular programming, the complete system is coded as small independent interacting modules. Each module is aimed at doing one specific task. The design for these modules is prepared in the form of structure charts.

A structure chart is a design tool that pictorially shows the relation between processing modules in computer software. Describes the hierarchy of components modules and the data are transmitted between them. Includes analysis of input-to-output transformations and analysis of transaction.

Structure charts show the relation of processing modules in computer software. It is a design tool that visually displays the relationships between program modules. It shows which module within a system interacts and graphically depicts the data that are communicated between various modules.

Structure charts are developed prior to the writing of program code. They identify the data passes existing between individual modules that interact with one another.

They are not intended to express procedural logic. This task is left to flowcharts and pseudocode. They don't describe the actual physical interface between processing functions.

Notation : Program modules are identified by rectangles with the module name written inside the rectangle. .

Arrows indicate calls, which are any mechanism used to invoke a particular module.

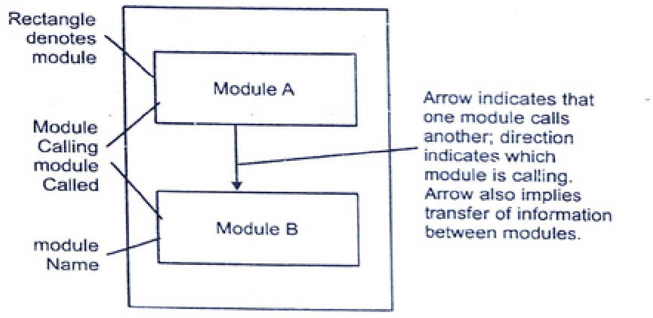

Rectangle denotes module
Module Calling
module Called
module Name
Module A
Module B
Arrow indicates that one module calls another; direction indicates which module is calling. Arrow also implies transfer of information between modules.

Fig. 19 : Notation used in structure charts.

Annotations on the structure chart indicate the parameter that are passed and the direction of the data movement. In Fig. 17, we see that modules A and B interact. Data identified as X and A are passed to module B, which in turn passes back Z.

A calling module can interact with more than one subordinate module. Fig. 17 also shows module L calling subordinate modules M and N. M is called on the basis of a decision point in L (indicated by the diamond notation), while N is called on the basis of the iterative processing loop (noted by the arc at the start of the calling arrow.

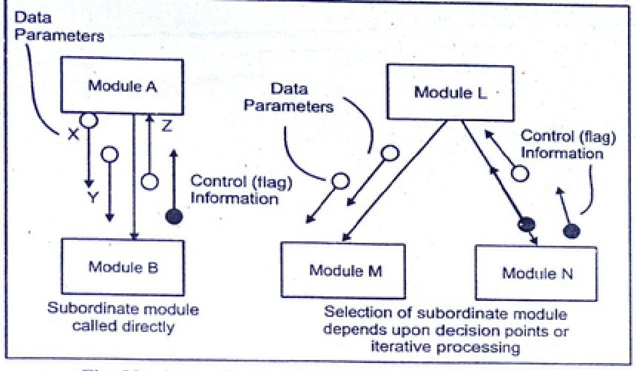

Data Parameters
Module A
X
Y
Z
Module B
Subordinate module called directly
Data Parameters
Control (flag) Information
Module L
Control (flag) Information
Module M
Module N
Selection of subordinate module depends upon decision points or iterative processing

Fig. 20 : Annotations and data passing in structure charts

Data Passing : When one module calls another, the calling module can send data to the called module so that it can perform the function described in its name. The called module can produce data that are passed back to the calling module.

Two types of data are transmitted. The first, parameter data, are items of data needed in the called module to perform the necessary work. A small arrow with an open circle at the end is used to note the passing of data paraméters. In addition, control information (flag data) is also passed. Its purpose is to assist in the control of processing by indicating the occurrence of, say, errors or end-of-conditions. A small arrow with a closed circle indicates the control information. A brief annotation describes the type of information passed.

Structure chart is a tool to assist the analyst in developing software that meets the objectives of good software design.

8. Structure of Modules :

We have discussed in the previous chapter that a system may be seen as a combination of several small independent units. So, while designing software also, it is designed as a collection of separately named and addressable components called modules. This property of software is termed as modularity. Modularity is a very important feature of any software and allows a program to be intellectually manageable. For instance, while coding small programs in 'c' also, we make a program as a collection of small functions. A program for finding average of three numbers may make use of a function for calculating the sum of the numbers. Each of these can be called as a separate module and may be written by a different programmer. But once such modules are created, different programs may use them.

Thus modularity in software provides several advantages apart from making the program more manageable.

While designing the modular structure of the program, several issues are to be paid attention. The modular structure should reflect the structure of the problem. It should have the following properties.

1. Intra-module property : Cohesion
   Modules should be cohesive.
2. Inter module property : Coupling
   Modules should be as loosely interconnected as possible.
   — Highly coupled modules are strongly interconnected.
   — Loosely coupled modules are weakly connected.
   — De-coupled modules exhibit no interconnection.
3. A module should capture in it strongly related elements of the problem.

8.1. Cohesion :

Cohesion, as the name suggests, is the adherence of the code statements within a module. It is a measure of how tightly the statements are related to each other in a module. Structures that tend to group highly related elements from the point of view of the problem tend to be more modular. Cohesion is a measure of the amount of such grouping.

Cohesion is the degree to which module serves a single purpose. Cohesion is a natural extension of the information-hiding concept. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program. There should be always high cohesion.

Cohesion: modules should interact with and manage the functions of a limited number of lower-level modules.

There are various types of cohesion.

Functional Cohesion : A module performs just one function.

Examples:

1. READ-RECORD.
2. EDIT-TRANSACTION

This is acceptable. But it breaks modules down into very small parts.

Sequential Cohesion : Module consisting of those processing elements, which has the output of one as the input of the next, is known to be sequentially cohesive.

Module consisting of P and Q.

In terms of DFD, this combines a linear chain of successive transformations. This is acceptable.

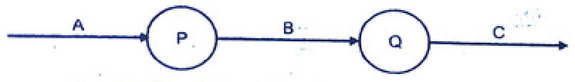


Fig. 21 : P and Q modules show sequential cohesion

Example:

1. READ-PROCESS; WRITE RECORD
2. Update the current inventory record and write it to disk.

Communicational Cohesion : Module consists of ail processing elements, which act upon the same input data set and/or produce the same output data set.

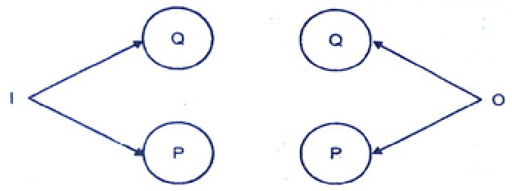


Fig. 22 : Communicational cohesion

P and Q form a single module.

Module is defined in terms of the problem structure as captured in the DFD. It is commonly found in business or commercial applications where one asks what are the things that can be done with this data set.

This is acceptabie.

Example:

1. Update master time clock record, the employee time and the current pay entry- all from same record.

Procedural Cohesion : Module formation associates processing elements together since these are found in the same procedural unit.

Often found when modules are defined by cutting up flowcharts or other procedural artifacts. There is no logical reasoning behind this. Fig 20 illustrates this. In this all the elements that are being used in the procedure 2 are put in the same module. It is not acceptable. Since elements of processing shall be found in various modules in a poorly structured way.

Temporal Cohesion : Temporal Cohesion is module formation by putting together all those functions, which happen at the same time.

Example: Before sorting, write a proof tape and check totals. So put functions for writing proof tape and checking totals in the same module.

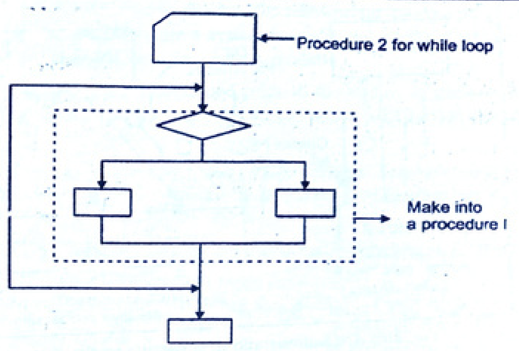


Fig. 23 : Procedural cohesion

Logical Cohesion : Logical Cohesion-is module formation by putting together a class of functions to be performed. It should be avoided.

For example, Display error on file, terminal, printer, etc.

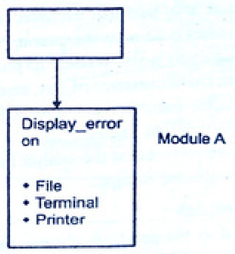


Fig. 24 :Logical Cohesion

Fig 21 shows logical cohesion. Here function Display error is for files, terminals, and printers. Since the function is to display error, all three functions are put into same modules.

Another example, produce job control reports, library file listings, and customer run support.

Coincidental Cohesion : Coincidental Cohesion is module formation by coincidence. Same code is recognized as occurring in some other module. Pull that code into a separate module. This type of cohesion should be avoided since it does not reflect problem structure.

8.2 Coupling :

Coupling is a measure of interconnection among modules in a program structure. Coupling depends on the interface complexity between modules the point at which entry or reference is made to a module, and what data pass across the interface. Or, simply it is Strength of the connections between modules Coupling can be represented, as a spectrum as shown is figure below. As the number of different items being passed (not amount of data) rises the module interface complexity rises. It is number of different items that increase the complexity not the amount of data.

There are four type of coupling

Data coupling: In this argument list data that is passed to the module. In this type of coupling only data flow across modules. Data coupling is minimal

Stamp coupling : In this type of coupling, a portion of the structure is argument

Control coupling : If there is control flag, that is, control decisions in subordinate module then it is control coupling.

Common Coupling : Common coupling occurs when there are common data areas. That is there are modules using data that are global. It should be avoided.

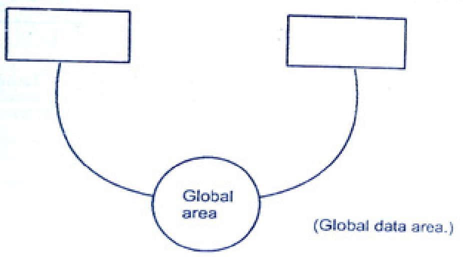


Fig. 25 : Common coupling

Content coupling: if there is data access within the boundary of another. For example, passing pointer can be considered as content coupling or branch into the middle of a module.

9. Coding :

Structure charts provide the framework for programmers to code the various modules of the system by providing all the necessary information about each module of the system. From here the system analyst takes a back seat and programmer comes into action. Programmer codes each and every module of the system, which gives a physical shape to the system.

Coding phase is aimed at converting the design of the system prepared during the design phase onto the code in a programming language, which performs the tasks as per the design requirements and is executable by a computer. The programs should be written such that they are simple, easy to read, understand and modify. Program should also include comment lines to increase the readability and modifiability of the program. Tools like flow charts and algorithms are used for coding the programs. In some cases, this phase may involve installation and modification of the purchased software packages according to the system requirements.

From here, the system goes for testing.

10. Data Dictionary :

The analysis model encompasses representations of data objects, functions, and control. In each representation data objects and/or control items playa role. Therefore, it is necessary to provide an organized approach for representing the characteristics of each data objects and control items. This is accomplished with the data dictionary.

Formally, Data Dictionary can be Defined as: The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and even intermediate calculations.

Data dictionary contains the following information :

l Name-the primary name of data or control item, the data store, or an external entity.

l Alias-other names used for first entry.

l Where-used/how-used-a listing of the processes that use the data or control item and how it is used. For example, input to a process, output from the process, as a store, as an external entity.

Central description- a notation for representing content.

Supplementary information - other information about data types, preset values, restrictions or limitations, etc.

The logical characteristics of current data stores, including name, description, aliases, contents, and organization. Identifies processes where the data are used and where immediate access to information is needed. Serves as the basis for identifying database requirements during system design.

11. Other Notation :

11.1. System Flowcharts :

Systems flowcharts are graphic illustrations of the physical flow of information through the entire accounting system. A systems flowchart is commonly used in analysis and design. Flowlines represent the sequences of processes, and other symbols represent the inputs and outputs to a process. Accountants use system flowcharts to describe the computerized processes, manual operations, and inputs and outputs of an application system. Auditors use system flowcharts to identify key control points in an accounting system's internal control structure. Figure shows the basic flowchart symbols. An example of a systems flowchart is shown below

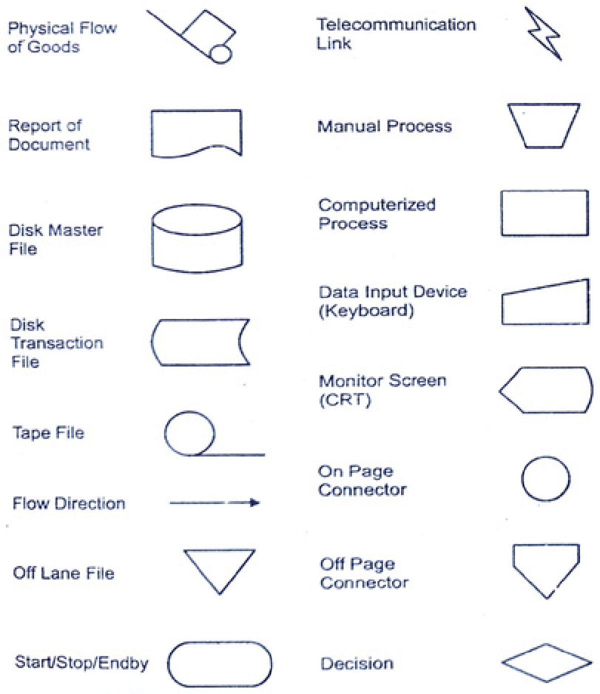


Fig. 26 Flowchart Symbols

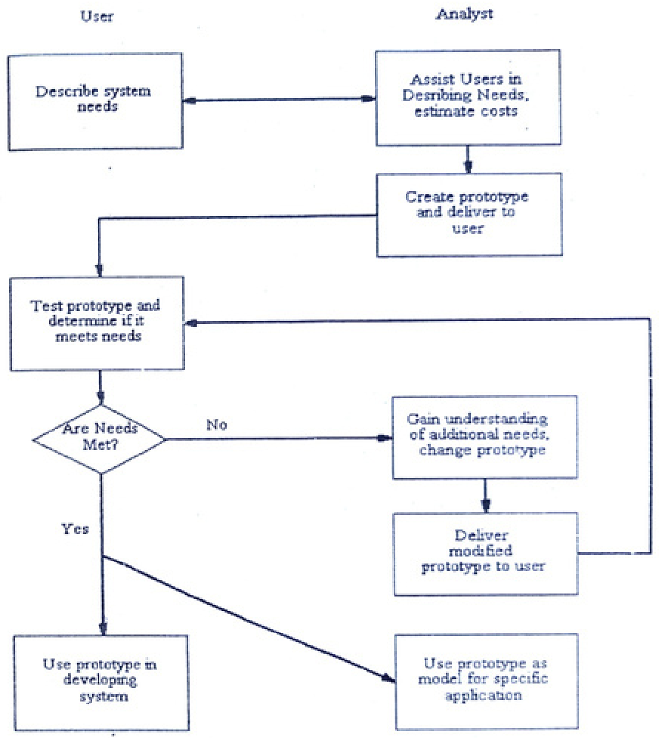

User
Analyst
Describe system needs
Assist Users in Desribing Needs, estimate costs
Create prototype and deliver to user
Test prototype and determine if it meets needs
Are Needs Met?
No
Gain understanding of additional needs, change prototype
Yes
Deliver modified prototype to user
Use prototype in developing system
Use prototype as model for specific application

Fig. 27 : System Flowchart of Sales Transaction Processing

11.2. Program Flowcharts :

Program flowcharts illustrate how individual computer programs work. That is, a program flowchart will show in detail each processing step of a computer program. Exhibit 4 depicts the master file update process in a batch processing system. As shown, the transaction record number is compared to the master file record number. When the two numbers match, then the master file account balance (MF_AMT) is updated. This process continues until all master file transaction file records are read.

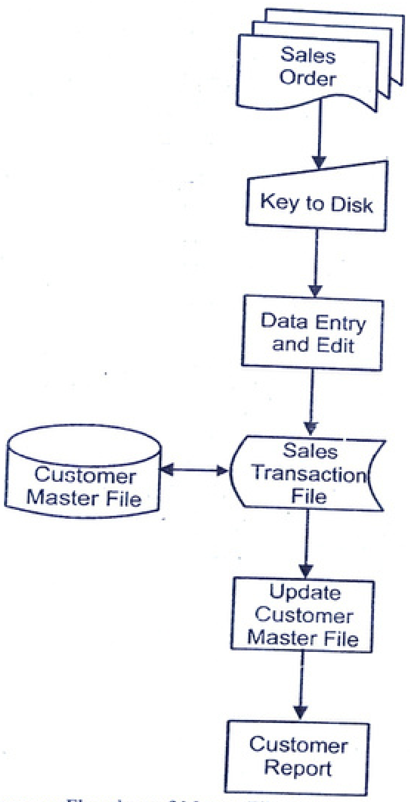


Fig. 28 : Programm Flowchart of Master File Update process

11.3. Document Flowcharts :

A document flowchart displays the flow of documents between organizational units. The chart is divided into several columns separated by vertical lines. Each column represents an organizational unit, such as a department, section, or employee. The flowchart shows the movement of a document from one department to another by a flowline connecting the document symbol in each department.

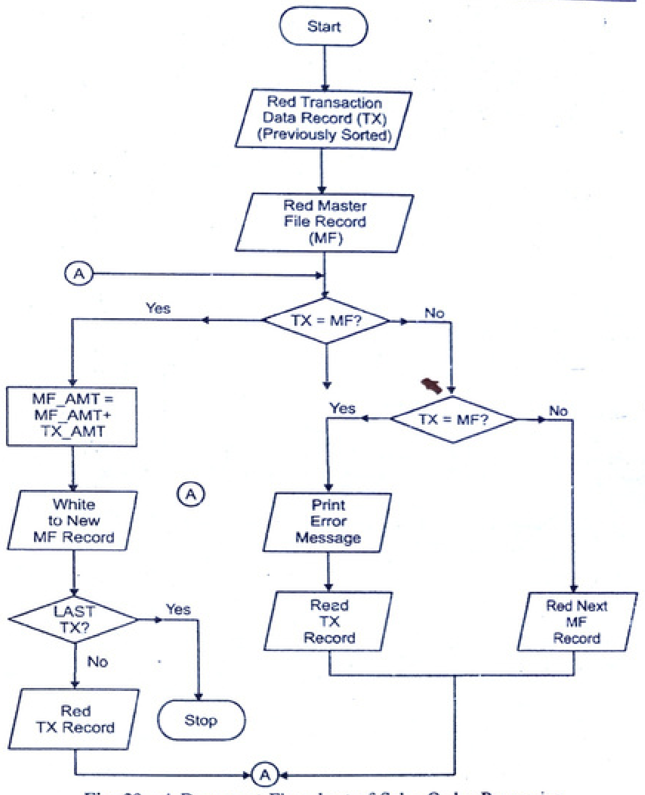


Fig. 29 : A Document Flowchart of Sales Order Processing

11.4. Prototyping :

Prototyping is the creation of a shell template of a system. In prototyping an information system, only sections of the system are modeled with emphasis on user interfaces such as screens, menus, source documents, and reports. This emphasis ensures that the user approves of the output. It is important that users understand that the modeling and building of the data underlying the shell is a time-consuming and critical portion of system development.

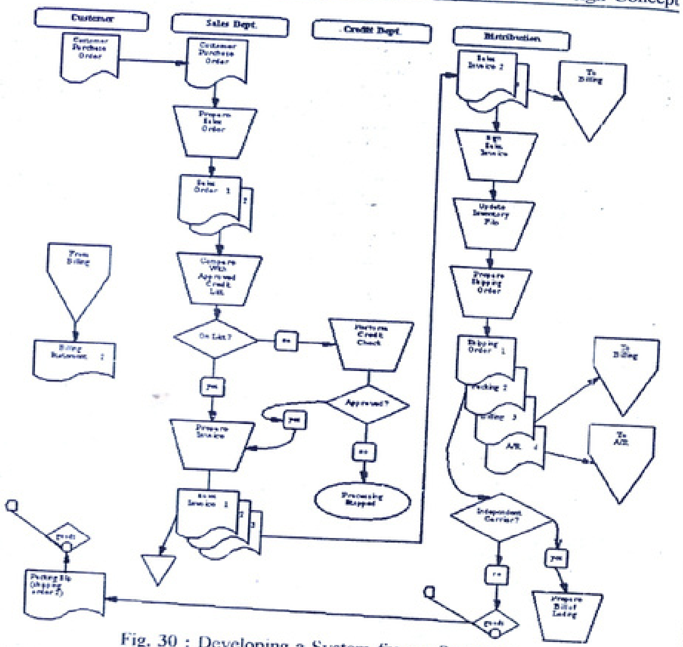


Fig. 30 : Developing a System from a Prototype

11.5. Structured Walkthroughs :

As a new system is developed, structured walkthrough is the meeting together of programmers and/or analysts on a regular basis to evaluate (i.e. walkthrough) their designs or codes. These walkthroughs provide constructive criticism and the opportunity to detect and correct logic errors before the testing phase of systems development occurs.

11.6. Top-Down Analysis :

In top-down analysis, the analyst begins with an overview of the entire system and gradually progresses until details at the lowest level are understood. This is an interactive process such that the analysis, design, coding, testing, and installation steps occur at each level. The greatest benefit to top-down analysis is that the difficult interface bugs are found early in the development process rather than at the end.

11.7. HIPO Charts :

HIPO stands for Hierarchy plus Input, Process, Output. The first part, the hierarchy, is a visual table of contents that displays the modules in a hierarchy much like the appearance of an organization chart. The second part is a diagram that lists all input, all processes, and all output, and is often called an "IPO" chart.

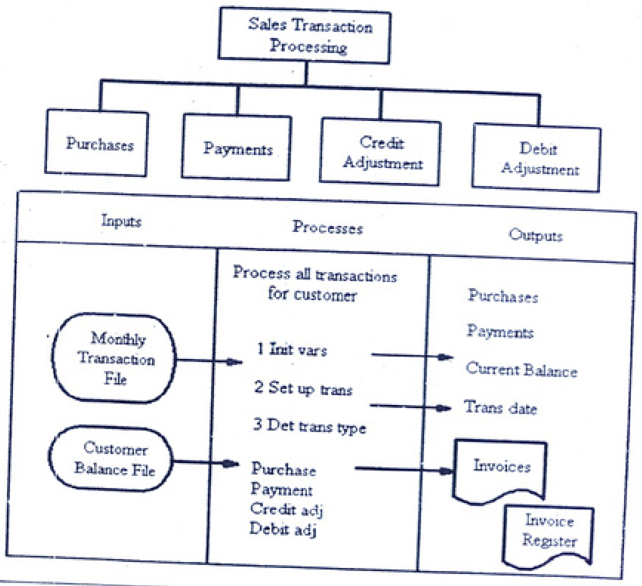


Fig. 31 : HIPO Chart of Sales Transaction Processing

11.8. Warnier-Orr Diagrams :

Warnier-Orr diagrams contain braces that identify each level of modules. The output or detail modules are shown on the right side of the braces and the control modules on the left side. Figure shows how the payroll cycle can be illustrated with a Warnier-Orr Diagram

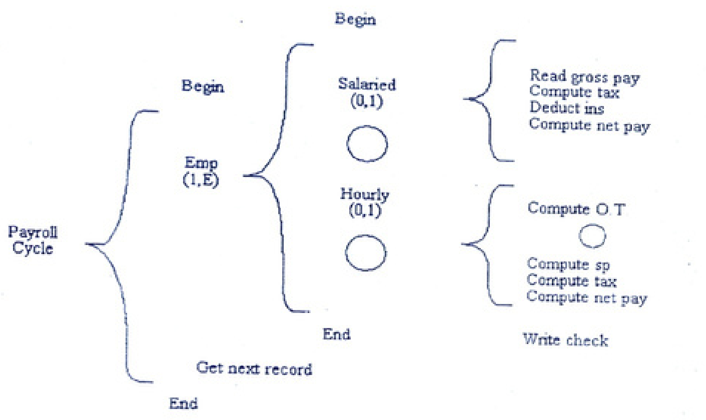


Fig. 32: Warnier-Orr Diagram for the Payroll Cycle

11.9. Nassi-Schneiderman Diagrams :

The Nassi-Schneiderman diagram is a graphic logic aid tool that causes the analyst to work in a modular, top-down mode. The three basic elements of process, decision, and iteration are contained within a box structure that represents the entire module. Figure provides the layout of a Nassi-Schneiderman diagram.

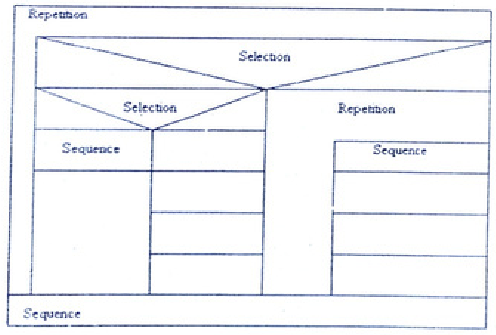


Fig. 33 : Layout for the Nassi-Schneiderman Diagram

12. Decision Making and Documentation :

Decision-making is an integral part of any business no matter how small, simple or big and complex it may be. Thus decisions have to be made and set procedures are to be followed as the subsequent actions. Thus while analyzing and designing a business system, analyst also needs to identify and document any decision policies or business rules of the system being designed. There are various tools and techniques available to the analyst for this, like decision trees, decision tables or structured English.

To analyze procedures and decisions the first step to be taken is to identify conditions and actions of all possible activities. Conditions are the possibilities or the possible states of any given entity, which can be a person, place, thing, or any event. Conditions are always in a flux i.e. they keep on varying time to time and object to object and based only on these conditions the decisions are made therefore conditions are also put as decision variables.

Documentation comes as an aid in this condition based decision process. As the whole web of all the possible combination of conditions and decisions is usually very large and cumbersome it becomes extremely important to document these so that there are no mistakes committed during the decision process.

Here comes the role of documenting tools, which are available to the analyst. Tools, which are usually used, are decision trees, decision tables, Structured English and the various CASE tools. The basic role of these tools is to depict the various conditions, their possible combinations and the subsequent decisions.

This has to be done without harming the logical structure involved. Once all of the parameters are objectively represented the decision process becomes much simpler, straightforward and almost error free

12.1. Decision Trees :

Decision tree is a tree like structure that represents the various conditions and the subsequent possible actions. It also shows the priority in which the conditions are to be tested or addressed. Each of its branches stands for anyone of the logical alternatives and because of the branch structure, it is known as a tree

The decision sequence starts from the root of the tree that is usually on the left of the diagram. The path to be followed to traverse the branches is decided by the priority of the conditions and the respectable actions. A series of decisions are taken, as the branches are traversed from left to right. The nodes are the decision junctions. After each decision point there are next set of decisions to be considered. Therefore at every node of the tree represented conditions are considered to determine which condition prevails before moving further on the path.

This decision tree representation form is very beneficial to the analyst. The first advantage is that by using this form the analyst is able to depict all the given parameters in a logical format which enables the simplification of the whole decision process as now there is a very remote chance of committing an error in the decision process as all the options a/c clearly specified in one of the most simplest manner.

Secondly it also aids the analyst about those decisions, which can only be taken when couple or more conditions should hold true together for there may be a case where other conditions are relevant only if one basic condition holds true.

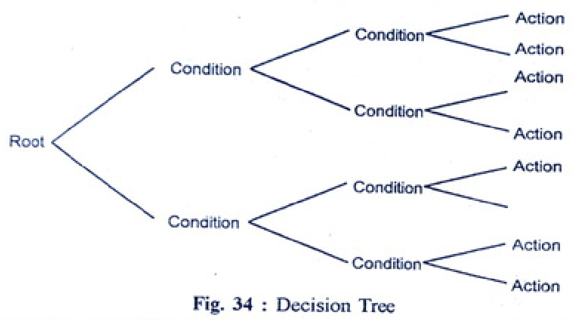


Fig. 34 : Decision Tree

In our day-to-day life, many a times we come across complex cases where the most appropriate action under several conditions is not apparent easily and for such a case a decision tree is a great aid. Hence this representation is very effective in describing the business problems involving more then one dimension and parameters.

They also point out the required data, which surrounds the decision process. All the data used in the decision making should be first described and defined by the analyst so that the system can be designed to produce correct output data.

Consider for example the discount policy of a saree manufacturer for his customers. According to the policy the saree manufacturer give discount to his customers based on the type of customer and size of their order. For the individual, only if the order size is 12 or more, the manufacturer gives a discount of 50% and for less than 12 sarees the discount is 30%. Whereas in case of shopkeeper or retailers, the discount policy is different. If the order is less than 12 then there is 15% discount. For 13 to 48 sarees order, the discount is 30%, for 49 to 84 sarees 40% and for more than 85 sarees the discount is 50%. The decision policy for discount percentage can be put in the form of a decision tree displayed in Fig. 2

The decision trees are not always the most appropriate and the best tool for the decision making process. Representing a very complex system with this tool may lead to a huge number of branches with a similar number of possible paths and options

For a complex problem, analyzing various situations is very difficult and can confuse the analyst.

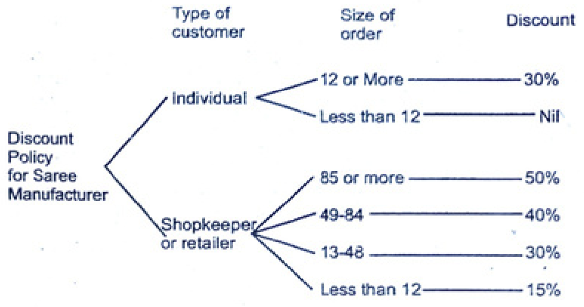


Fig. 35 : A Sample Decision Tree

12.2. Decision Tables :

A decision table is a table with various conditions and their corresponding actions.

Decision tree is a two dimensional matrix. It is divided into four parts, condition stub, action stub, condition entry, and action entry. See fig. 3. Condition stub shows the various possible conditions. Condition entry is used for specifying which condition is being analyzed. Action stub shows the various actions taken against different conditions. And action entry is used to find out which action is taken corresponding to a particular set of conditions.

The steps to be taken for a certain possible condition are listed by action statements. Action entries display what specific actions to be undertaken when selected conditions or combinations of conditions are true. At times notes are added below the table to indicate when to use the table or to distinguish it from other decisions tables.

The right side columns of the table link conditions and actions and form the decision rules hence they state the conditions that must be fulfilled for a particular set of actions to be taken. In the decision trees, a fixed ordered sequence is followed in which, conditions are examined. But this is not the case here as the decision rule incorporates all the required conditions, which must be true.

Developing Decision Tables : Before describing the steps involved in building the decision table it is important to take a note of few important points. Every decision should be given a name and the logic of the decision table is independent of the sequence in which condition rules are written but the action takes place in the order in which events occur. Wherever possible, duplication of terms and meaning should be avoided and only the standardized language must be used. The steps of building the concerned tables are given below.

1. Firstly figure out the most essential factors to be considered in making a decision. This will identify the conditions involved in the decision. Only those conditions should be selected which have the potential to either occur or not but partial occurrences are not permissible.

2. Determine the most possible steps that can take place under varying conditions and not just under current condition. This step will identify the actions.

3. Calculate all the possible combinations of conditions. For every N number of conditions there are 2*2*2..... (N times) combinations to be considered.

4. Fill the decision rules in the table.

Entries in a decision table are filled as Y/N and action entries are generally marked as "X". For the conditions that are immaterial a hyphen "-" is generally put. Decision table is further simplified by eliminating and consolidating certain rules. Impossible rules are eliminated. There are certain conditions whose values do not affect the decision and always result in the same action. These rules can be consolidated into a single rule.

Example: Consider the recruitment policy of ABC Software Ltd.

It the applicant is a BE then recruit otherwise not. If the person is from Computer Science, put him/her in the software development department and if the person is from non-computer science background put him/her in HR department. If the Person is from Computer Science and having experience equal to or greater than three years, take him/her as Team leader and if the experience is less than that then take the person as Team member. If the person recruited is from non Computer Science background, having experience less than three years, make him/her Management Trainee otherwise Manager.

Condition stub	Condition entry					
	1	2	3	4	5	6
Customer is individual	Y	Y				
Customer shopkeeper or retailer?			Y	Y	Y	Y
Order-size 85 copies or more?			Y			
Order-size 49-84 sarees ?				Y		
Order-size 13-48 copies?					Y	
Order-size 12 or more?		Y				
Order-size less than 12?	Y					Y
Allow 50% discount		X	X			
Allow 40% discount				X		
Allow 30% discount	X				X	
Allow 15% discount						X

Action Stub Action Entry

Fig. 36 : Decision Table -Discount Policy

The first decision table for the problem stated above can be drawn as shown in Fig. 4

Condition Stub	Condition Entry							
	1	2	3	4	5	6	7	8
Is person BE ?	Y	N		N			Y	
Is person Comp	Y	Y		N			N	
Work Experience >=3	Y	Y		Y			N	
Recruit	X		X		X		X	
Don't Recruit		X		X		X		X
S/W Deptt	X				X			
HRDeptt			X				X	
Team Leader	X							
Team Mem					X			
MgtTrainee							X	
Manage			X					

Action Stub Action Entry

Fig. 37 : Decision Table

This table can further be refined by combining condition entries 2, 4, 6, and 8. The simplified table is displayed in fig. 5.

BE?	Y	Y	Y	Y	N
CS?	Y	N	Y	N	-
Work >=	Y	Y	N	N	-
Recruit	Y	Y	Y	Y	
Don't recruit					X
S/W Deptt	Y		Y		
HR Deptt		Y		Y	
Team Leader	Y				
Team Mem			Y		
Mgt Trainee				Y	
Manager		Y			

Fig. 38 : Simplified Decision Table

12.3 Case Study Example :

Now we'll look at the techniques that the analyst employed to document the various business rules of the library. Analyst identified the following business rules.

(1) Procedure for Becoming a Member of Library : Anyone whose age is 18 or more than that can become a member of library. There are two types of memberships depending upon the duration of membership. First is for 6 months and other is for 1 year. 6 months membership fee is Rs 900 and 1 year membership fee is Rs 1500. The decision tree illustrating the business rule is given below.

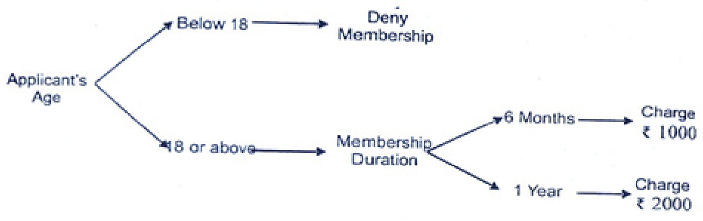


Fig. 39 : Decision Tree for Membership Rule

Is Age < 18	Y		
Age >= 18		Y	Y
Is Membership For 6 Months?		Y	
Is Membership For 12 months			Y
Grant Membership		X	X
Deny membership	X		
Charge membership fees Rs 1000		X	
Charge membership fees Rs 2000			X

Fig. 40 : Decision Table for Membership Rule

Rule for Issuing Books : If the number of books already issued is equal to 5 then no more books is issued to that member. If it is less than 5 then that book is issued.

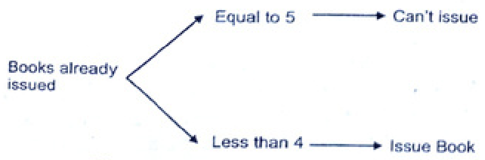


Fig. 41 : Decision Tree for Issue of Books

Are Book Already Issued= 4	Y	
Are books Already issued <4		Y
Don't Issue	X	
Issue		X

Fig. 42 : Decision table for Book Issue rule

Returning Books : Whenever a member returns a book, it is checked if the book is being returned after the due return date. If this is the case, then a fine of Rs 2 per day after the return date is charged. If the book is returned on the due date or before that, then no fine is charged.

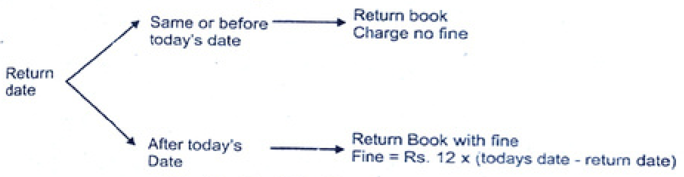

Return date
Same or before today's date
Return book Charge no fine
After today's Date
Return Book with fine Fine = Rs. 12 x (todays date - return date)

Fig. 43 : Decision Tree for Return of Books

Is Return Date <= Today's date	Y	
Is Return Date> Today's date		Y
Return book without charging any fine	X	
Return book with fine Fine = Rs 5 x [Today's date - Return Date]		X

Fig. 44 : Decision table for return of books

Now the analyst has a good understanding of the requirements for the new system, we can move to the designing Design of the system will be discussed in the later chapters.

12.4. Structured English :

Structured English is one more tool available to the analyst. It comes as an aid against the problems of ambiguous language in stating condition and actions in decisions and procedures. Here no trees or tables are employed, rather with narrative statements a procedure is described. Thus it does not show but states the decision rules. The analyst is first required to identify the conditions that occur in the process, subsequent decisions, which are to be made and the alternative actions to be taken.

Here the steps are clearly listed in the order in which they should be taken. There are no special symbols or formats involved unlike in the case of decision trees and tables, also the entire procedure can be stated quickly as only English like statements are used. Structured English borrows heavily from structured programming as it uses logical construction and imperative statements designed to carry out instructions for actions. Using "IF", "THEN", "ELSE" and "So" statement decisions are made. In this structured description terms from the data dictionary are widely used which makes the description compact and straight.

Developing Structured Statements: Three basic types of statements are employed to describe the process.

1. **Sequence Structures** - A sequence structure is a single step or action included in a process. It IS independent of the existence of any condition and when encountered it is always taken. Usually numerous such instructions are used together to describe a process.

2. **Decision Structures** - Here action sequences described are often included within decision structures that identify conditions. Therefore these structures occur when two or more actions can be taken as per the value of a specific condition. Once the condition is determined the actions are unconditional.

```
COMPUTE DISCOUNT
See Number of Sarees
IF order is from individual
        and-if order is for 12 or more sarees
                THEN: Discount is 50%
        ELSE order is for fewer than 12 arees
                SO: Discount is 30%
ELSE order is from shopkeeper or retailer
        SO-IF order is for 85 sarees or more
                Discount is 50%
        ELSE IF order is for 49 to 84 sarees
                Discount is 40%
        ELSE IF order is for 48 to 13 sarees
                Discount is 30%
        ELSE order is for less than 12 sarees
        SO:     Discount is 15%
```

Fig. 45 : An example of Structured English

3. **Iteration Structures**- these are those structures, which are repeated, in routing - operations such as DO WHILE statements.

The decision structure of example discussed in previous sections (see decision table in fig 3) may be given in structured English as in fig 6.

Data Dictionary :

As the name suggests the data dictionary is a catalog or repository of data terms such as data elements, data structures etc. Data dictionary is a collection of data to be captured and stored in the system, inputs to the systems and outputs generated by the systems. So we first know more about what are these data elements and structures.

Data element : The smallest unit of data, which can not be further decomposed, is known as a data element. For example any number digit or an alphabet will qualify

to be data elements. Data element is the data at the most fundamental level. These elements are used to as building blocks for all other data in the system. At times data elements are also referred as data item, elementary item or just as field. There is a very little chance that only by them data element can convey some meaning.

Data Structure : Data elements when clubbed together as a group make up a data structure. These data elements are related to one another and together they stand for some meaning. Data structures are used to define or describe the system's components.

Data dictionary entries consist of a set of details about the data used or reduced in the system such as data flows, data star processes. For each item the dictionary records its name, description, alias and its length. The data dictionary takes its shape during the data flow analysis and its contents are used even till the system design. It very reasonable to know why the data dictionary is so essential. There are numerous important reasons.

In a system there is data volume flow in the form of reports, documents etc. In these transactions either the existing data is used or new data items are created. This poses a potential problem for the analyst and thus developing and using a well-documented dictionary can be a great help.

Now consider a case where everyone concerned with the system derives different meanings for the same data items. This problem can continue until the meaning of all data items and others are well documented so that everyone can refer to the same source and derive the same common meaning.

Documentation in data dictionary is further carried on to record about the circumstances of the various process of the system. A data dictionary is always an added advantage for the system analysis. From the data dictionary one can determine about the need of the new features or about the changes required. Thus they help in evaluating the system and in locating the errors about the system description, which takes place when the contents of the dictionary are themselves not up to the mark.

CASE Tools :

A tool is any device, object or a kind of an operation used to achieve a specific task. The complete and correct description of the system is as important as the system itself. The analyst uses case tool to represent and assemble all the information and the data gathered about the system.

Most of the organizations have to follow some kind of procedures and they are required to make all sorts of decisions from time to time, For the job of the analyst both these procedures and decision-making processes of the business system under investigation are equally important

Expressing business processes and rules in plain text is very cumbersome and difficult process, It requires a lot of effort. Moreover, it does not guarantee if the reader will understand it well. So representing these things graphically is a good choice. So CASE Tools are useful in representing business rules and procedures of organization in graphical way. It requires less effort and it is easy to understand.

Some CASE tools are designed for creating new applications and not for maintaining or enhancing existing ones hence if an organization is in a maintenance mode, it must have a CASE tool that supports the maintenance aspect of the software developed. Many a times the large projects are too big to be handled by a single analyst thus here the CASE tool must be compatible enough to allow for partitioning of the project.

Efficient and better CASE tools collect a wide range of facts, diagrams, and rules, report layouts and .screen designs, The CASE tool must format the collected data into a meaningful document ready to use.

Questions

Very short Questions:

1. What is module
2. What is data dictionary
3. What is Program flow chart
4. What is HIPO Chart
5. What is Warnier Orr Diagram

Short Questions

1. What are different design elements
2. What is DFD ? What are the symbols used in DFD
3. What do you mean by cohesion and coupling ?
4. What are the steps involved in making DFD ?
5. What do you mean by control flow model ?

Long Question :

1. Draw a context level model (level O DFD) for five systems with which you are familiar. The system need not be computer based.
2. Using the Systems described in previous problem, refine each in to a level 1 and level 2 DFD.
3. What would a simple flow diagram for a payroll system look like ?
4. Discuss between logical and physical views of the system. Which view is included in a data flow diagram ? Why ?
5. What role does observation play in system investigation ?
6. What are conditions and actions ? What are their roles in decision analysis?
7. In what way do decision trees assist indecision analysis ? Explain how an analyst should develop a decision tree.
8. What advantages do decision trees present for analysis ?
9. How does the purpose of decision tables differ from that of decision trees ? What components make up a decision tables ?
10. How do analysis develop a decision table ?
11. How a decision table is developed ?
12. How does structured English differ from the decision tree and decision table ? What advantage does it offer over the other two methods ?

❑❑❑

8

DATA REQUIREMENT & DATA MODELS

Objectives

1. Data Requirements :

Last chapter discusses about one part of the conceptual design process, the functional model. The other is the data model, which discusses the data related design issues of the system. See Fig 1. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. In this chapter, we'll look into details of data modeling.

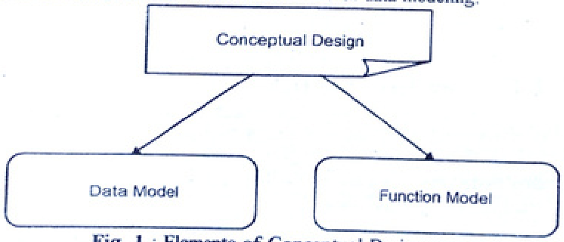
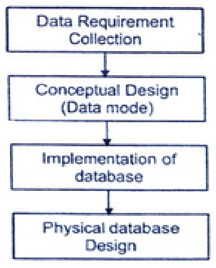


Fig. 1 : Elements of Conceptual Design

We have already discussed the Data Flow Diagrams, which make the foundation of the system under development. While the system is being studied, the physical DFDs are prepared whereas at the design phase, the basic layout of the proposed system is depicted in the form of a logical DFD. Taking this DFD as the basis the system is further developed. Even at the Data Modeling phase, the DFD can provide the basis in the form of the data flows and the Data Stores depicted in the DFD of the proposed system. The Data Stores from the DFD are picked up and based on the data being stored by them the Data Model of the system is prepared.

Prior to data modeling, we'll talk of basics of database design process. The database design process can be described as a set following steps. (Also see fig. 2)

- Requirement collection: Here the database designer interviews database users. By this process they are able to understand their data requirements. Results of this process are clearly documented. In addition to this, functional requirements are also specified. Functional requirements are user defined operations or transaction like retrievals, updates, etc., that are applied on the database.
- Conceptual schema: Conceptual schema is created. It is the description of data requirements of the users. It includes description of data types, relationships and constraints.
- Basic data model operations are used to specify user functional requirements.
- Actual implementation of database.
- Physical database design. It includes design of internal storage structures and files.



Data Requirement Collection
Conceptual Design (Data mode)
Implementation of database
Physical database Design

Fig. 2 : Overall database design process

In this chapter, our main concern is data model. There are various data models available. They fall in three different groups.

- Object-based logical models
- Records-based logical models
- Physical-models

Object-Based Logical Models : Object-based logical models are used in describing data at the logical and view levels. The main characteristic of these models is that they provide flexible structuring capabilities and allows data constraints to be specified explicitly. Many different models fall into this group. They are following :

- Entity-relationship model
- Object-oriented model

In this chapter, we'll discuss Entity-Relationship model in detail. The object-oriented model is covered in the next chapter.

Record-Based Logical Models : Records-based logical models are used in describing data at the logical and view levels. They are used to specify the overall logical structure of the database and to provide a higher-level description of the implementation.

In record-based models, the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length. The use of fixed-length records simplifies the physical-level implementation of the database. The following models fall in this group.

- Relational model
- Network model
- Hierarchical model

Relational Model : This model uses a collection of tables to represent data and relationship among those data. Each table has multiple columns, and each column has a unique name. Figure shows a simple relational database.

Emp. No	Emp. Name	Age
1000	Ashish	23
2000	Saurabh	21
3500	Vikram	45
4000	Nagender	25

Emp. No	DOJ
1000	15-Jul-2002
2000	11-May-2003
3500	21-Jun-2007
4000	15-Jun-2008

Fig. 3 : A sample relational model

Network Model : In network database, data is represented by collection of records, and relationships among data are represented by links. The records are organized as a collection of arbitrary graphs. Figure :4 represent a simple network database.

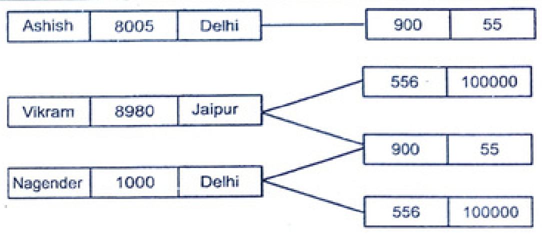


Fig. 4 : A sample network model

Hierarchical Model : The hierarchical model is similar to the network model. Like network model, records and links represent data relationships among data respectively. It differs from the network mode in that the records are organized as collections trees rather than arbitrary graphs. Fig. 5 represents a simple database.

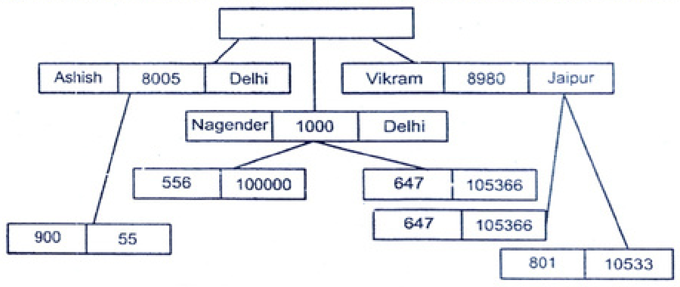


Fig. 5 : A sample hierarchical database

Physical Data Models : Physical data models are used to describe data at the lowest level. A few physical data models are in use. Two such models are:

- Unifying model
- Frame-memory model

Physical data models capture aspects of database-system implementation.

2. E-R Data Modeling Technique :

Now we know various data models available. To understand the process of data modeling we'll study Entity Relationship model. Peter P. Chen originally proposed the Entity Relationship (ER) model in 1976. The ER model is a conceptual data model that views the real world as a construct of entities and associations or relationships between entities.

A basic component of the model is the Entity-Relationship diagram, which is used to visually represent data objects. The ER modeling technique is frequently used for the conceptual design of database applications and many database design tools employ its concepts.

ER model is easy to understand. Moreover it maps easily to relational model. The constructs used in ER model can easily be transformed into relational tables. We will look into relational model in the next chapter, where other data models are discussed. In the following section, we'll look at E-R model concepts.

We can compare ER diagram with a flowchart for programs. Flow chart is a tool for designing a program; similarly ERD is a tool for designing databases. Also an ER diagram shows the kind and organization of the data that will be stored in the database in the same way a flowchart chose the way a program will run.

3. E-R Model Concept :

The ER data modeling techniques is based on the perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects.

In ER modeling, data is described as entities, relationships, and attributes. In the following section, entities and attributes are discussed. Later, entity types, their key attributes, relationship types, their structural constraints, and weak entity types are discussed. In the last, we will apply ER modeling to our case study problem "Library management system".

3.1. Entities and Attributes :

One of the basic components of ER model is entity. An entity is any distinguishabie object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. Each entity is distinct.

An entity may be physical or abstract. A person, a book, car, house, employee etc. are all physical entities whereas a company, job, or a university course, are abstract entities.

Employee
Loan
Physical
Abstract

Fig. 6 : Physical and Abstract Entity

Another classification of entities can be independent or dependent (strong or weak) entity.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one, which does not rely on another entity for identification. A dependent entity is one that relies on another' entity for Identification. An independent entity exists on its own whereas dependent entity exists on the existence of some other entity. For example take an organization scenario. Here department is independent entity.

Department manager is a dependent entity. It exists for existing depts. There won't be any department manager for which there is no dept.

Some entity types may not have any key attributes of their own. These are called weak entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. For example, take the license entity. It can't exist unless It IS related to a person entity.

Attributes: After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities. For example, Dept entity can have Dept. Name, Dept. Id, and Dept. Manager as its attributes. A car entity can have model no., brand name, and color as its attributes.

A particular instance of an attribute is a value. For example, "Bhaskar" is one value of the attribute Name. Employee number 8005 uniquely identifies an employee in a company.

The value of one or more attributes can uniquely identify an entity.

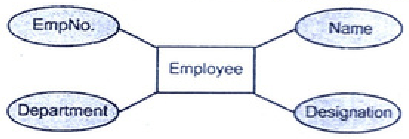


Fig. 7 - Entity and its attributes

In the above figure, employee is the entity. EmpNo., Name, Designation and Department are its attributes.

An entity set may have several attributes. Formally each entity can be described by set of <attribute, data value> pairs.

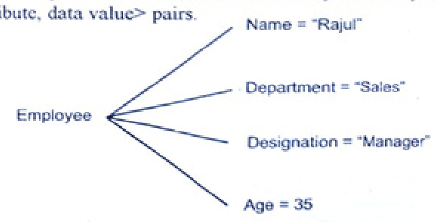


Fig. 8 : Employee entity and its attribute values

3.2. Types of Attributes :

Attributes can be of various types. In the section, we'll look at different types of attributes Attributes can be categorized as:

- Key or non key attributes
- Required or optional
- Simple or composite
- Stored or derived

Key or Non-key Attributes : Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys or key attributes uniquely identify an instance of an entity. If such an attribute doesn't exist naturally, a new attribute is defined for that purpose, for example an ID number or code. A descriptor describes a non-unique characteristic of an entity instance.

An entity usually has an attribute whose values are distinct for each individual entity. This attribute uniquely identifies the individual entity. Such an attribute is called a key attribute. For example, in the Employee entity type, EmpNo is the key attribute since no two employees can have same employee number. Similarly, for Product entity type, Product is the key attribute.

There may be a case when one single attribute is not sufficient to identity entities. Then a combination of attributes can solve this purpose. We can form a group of more than one attribute and use this combination as a key attribute. That is known as a composite key attribute. When identifying attributes of entities, identifying key attribute is very important.

Attributes that describe an entity are called non-key attributes.

Required or Optional : An attribute can be required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence. For example, there is an attribute EmpNo (for employee no.) of entity employee. This is required attribute since here would be no employee having no employee no. Employee's spouse is optional attribute because an employee mayor may not have a spouse.

Simple and Composite : Composite attributes can be divided into smaller subparts. These subparts represent basic attributes with independent meanings of their own. For example, take Name attributes. We can divide it into sub-parts like First_name, Middle_name, and Last_name.

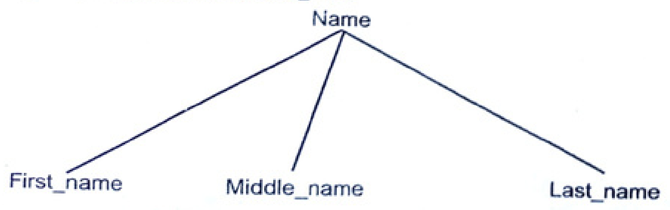


Fig. 9 : Composite attributes

Attributes that can't be divided into subparts are called Simple or Atomic attributes. For example, EmployeeNumber is a simple attribute. Age of a person is a simple attribute.

Single-valued and Multi-valued : Attributes that can have single value at a particular instance of time are called single valued. A person can't have more than one age value. Therefore, age of a person is a single-values attribute. A multi-valued attribute can have more than one value at one time. For example, degree of a person is a multi-valued attribute since a person can have more than one degree. Where appropriate, upper and lower bounds may be placed on the number of values in a multi-valued attribute. For example, a bank may limit the number of addresses recorded for a single customer to two.

Stored, Coded, or Derived : There may be a case when two or more attributes values are related. Take the example of age. Age of a person can be can be calculated from person's date of birth and present date. Difference between the two gives the value of age. In this case, age is the derived attribute. The attribute from which another attribute value is derived is called stored attribute. In the above example, date of birth is the stored attribute. Take another example, if we have to calculate the interest on some principal amount for a given time, and for a particular rate of interest, we can simply use the interest formula

Interest = NPR/1OO;

In this case, interest is the derived attribute whereas principal amount(P), time(N) and rate of interest(R) are all stored attributes.

Derived attributes are usually created by a formula or by a summary operation on other attributes.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female".

The attributes reflect the need for the information they provide. In the analysis meeting, the participants should list as many attributes as possible. Later they can weed out those that are not applicable to the application, or those clients are not prepared to spend the resources on to collect and maintain. The participants come to an agreement, on which attributes belong with an entity, as well as which attributes are required or optional.

3.3. Entity Types :

An entity set is a set or entities or the same type that share the same properties, or attributes. For example, all software engineers working in the department involved in the Internet projects can be defined as the entity set InternetGroup. The individual entities that constitute a set are called extension of the entity set. Thus, all individual software engineers of in the Internet projects are the extensions of the entity set InternetGroup.

Entity sets don't need to be disjointed. For example, we can define an entity set Employee. An employee mayor may not be working on some Internet projects. In InternetGroup we will have some entries that are there in Employee entity set. Therefore, entity sets Employee and InternetGroup are not disjoint.

A database usually contains groups of entities that are similar. For example, employees of a company share the same attributes. However, every employee entity has its own values for each attribute. An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Fig. 10 shows two entity types Employee and Product. Their attribute list is also shown. A few members of each entity type are shown.

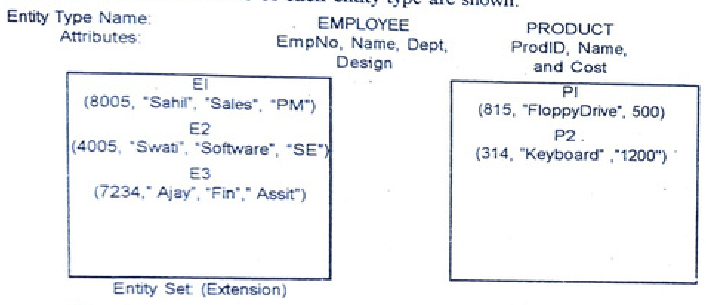

Entity Type Name:
Attributes:
EMPLOYEE
EmpNo, Name, Dept, Design
PRODUCT
ProdID, Name, and Cost
EI
(8005, "Sahil", "Sales", "PM")
E2
(4005, "Swati", "Software", "SE")
E3
(7234," Ajay", "Fin"," Assit")
PI
(815, "FloppyDrive", 500)
P2
(314, "Keyboard" ,"1200")
Entity Set (Extension)

Fig. 10 : Two entity types and some of the member entities of each

An entity type is represented in ER diagrams as rectangular box and the corresponding attributes are shown in ovals attached to the entity type by straight lines. See Fig. 7.

An entity type is basically the schema or intension or structure for the set of entities that share the same structure whereas the individual entities of a particular entity type are collectively called entity set. The entity set is also called the extension of the entity type.

3.4. Value Sets (Domain) of Attributes :

Each attribute of an entity type is associated with a value set. This value set is also called domain. The domain of an attribute is the collection of all possible values an attribute can have.

The value set specifies the set of values that may be assigned for each individual entity. For example, we can specify the value set for designation attribute as <"PM", "Assit", "DM", "SE">. We can specify "Name" attribute value set as <strings of alphabetic characters separated by blank characters>. The domain of Name is a character string.

3.5. Relationships :

After identification of entities and their attributes, the next stage in ER data modeling is to identify the relationships between these entities.

We can say a relationship is any association, linkage, or connection between the entities of interest to the business. Typically, a relationship is indicated by a verb connecting two or more entities. Suppose there are two entities of our library system, member and book, then the relationship between them can be "borrows".

Member borrows book

Each relationship has a name, degree and cardinality. These concepts will be discussed next.

3.6. Degree of a Relationship Type :

Relationships exhibit certain characteristics like degree, connectivity, and cardinality. Once the relationships are identified their degree and cardinality are also specified.

Degree: The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities are the most common type in the real world.

Fig. 11 shows a binary relationship between member and book entities of library system

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decompesed into two or more binary relationships.

Connectivity and Cardinality : By connectivity we mean how many instances of one entity are associated with how many instances of other entity in a relationship. Cardinality is used to specify such connectivity.

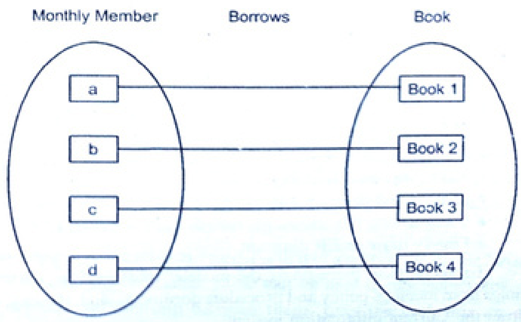


Fig. 11 : Binary Relationship

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

A one-to-one (I: I) relationship is when at most one instance of an entity A is associated with one instance of entity B. For example, each book in a library is issued to only one member at a particular time.

A one-to-many (1 :N) relationship is when for one instance of entity A, there are zero, one, or many instances of entity B but for one instance of entity B, there is only one instance of entity A. An example of a I:N relationships is a department has many employees; each employee is assigned to one department.

A many-to-many (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example,

An employee may be assigned to a project. Mandatory relationships, on the other hand, are indicated by words such as must have. For example, a student must register for at least three courses in each semester.

4. Designing Basic Model and E-R Diagrams :

E-R diagrams represent the schemes or the overall organization of the system. In this section, we'll apply the concepts of E-R modeling to our "Library Management System" and draw its E-R diagram.

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirement analysis for the purpose of: and

- classifying data objects as either entities or attributes,
- identifying and defining relationships between entities,
- naming and defining identified entities, attributes, and relationships,
- documenting this information in the data document.
- Finally draw its ER diagram.

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the Current information system.

E-R diagrams Constructs : In E-R diagrams, entity types are represented by square See fig 12. Relationship types are shown in diamond shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type or relationship type by a straight line. Multivalued attributes are shown in double ovals. Key attributes have their names underlined. Derived attributes are shown in dotted ovals.

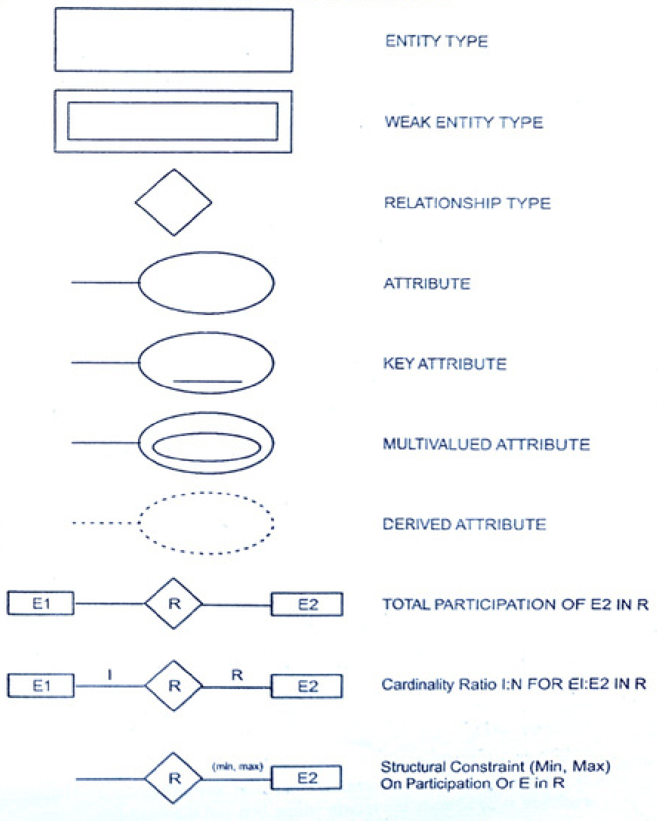


Fig. 12 : Summary of ER diagram notation

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds.

Attaching a I, M, or N on each participating edge specifies cardinality ratio of each binary relationship type. The participation constraint is specified by a single line for partial participation and by double lines for total participation. The participation constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. If every entity of an entity set is related to some other entity set via a relationship type, then the participation of the first entity type is total. If only few member of an entity type is related to some entity type via a relationship type, the participation is partial.

Naming Data Objects : The names should have the following properties:

- unique.
- have meaning to the end-user.
- contain the minimum number of words needed to uniquely and accurately
- describe the object.

For entities and attributes, names are singular nouns while relationship names are typically verbs

E-R Diagram for Library Management System : In the library Management system, the following entities and attributes can be identified.

- Book -the set all the books in the library. Each book has a Book-id, Title, Author, Price, and Available (y or n) as its attributes.
- Member-the set all the library members. The member is described by the attributes Member _id, Name, Street, City, Zip_code, Mem_type, Mem_date (date of membership), Expiry_date.
- Publisher-the set of all the publishers of the books. Attributes of this entity are Pub_id, Name, Street, City, and ZipJode.
- Supplier-the set or all the Suppliers of the books. Attributes of this entity are Sup_id, Name, Street, City, and Zip_code.

Assumptions: a publisher publishes a book. Supplier supplies book to library. Members borrow the book (only issue).

Return of book is not taken into account.

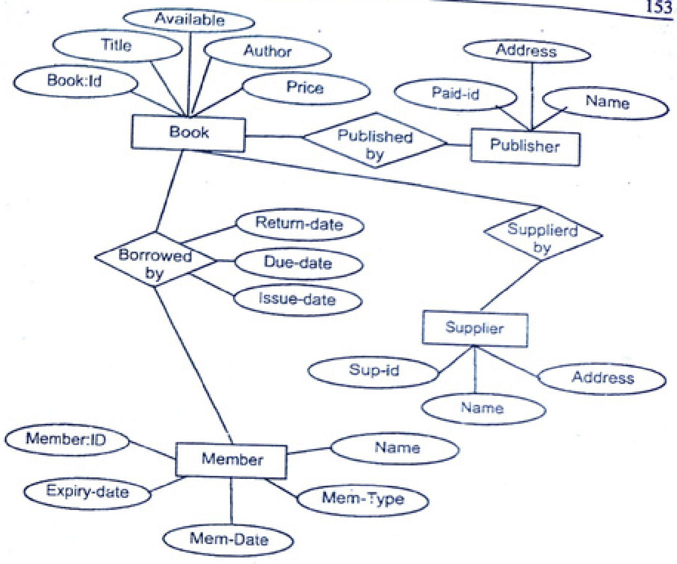


Fig. 13 : E-R Diagram of Library Management System.

5. Relation Model :

E. F. Codd proposed this model in the year 1970. The relational database model is the most popular data model. It is very simple and easily understandable by information systems professionals and end users.

Understanding a relational model is very simple since it is very similar tc Entity Relationship Model. In ER model data is represented as entities similarly here data in represented in the form of relations that are depicted by use of two-dimensional tables. Also attributes are represented as columns of the table. These things are discussed in detail in the following section.

The basic concept in the relational model is that of a relation. In simple language, a relation is a two-dimensional table. Table can be used to represent some entity information or some relationship between them. Even the table for an entity information and table for relationship information are similar in form. Only from the type of information given in the table can tell if the table is for entity or relationship. The entities and relationships, which we studied in the ER model, are similar to relations in this model. In relational model, tables represent all the entities and relationships identified in ER model. Rows in the table represent records; and columns show the attributes of the entity.

Attributes

Data item 1	Data item2	Data item3
Record 1		
Record 2		
Record 3		

Table 1 : Structure of a relation.

A table exhibits certain properties. It is a column homogeneous. That is, each item in a particular column is of same type. See fig 2. It shows two columns for EmpNo and Name. In the EmpNo column it has only employee numbers that is a numeric quantity. Similarly in Name column it has alphabetic entries. It is not possible for EmpNo to have some non-numeric value (like alphabetic value). Similarly for Name column only alphabetic values are allowed.

EmpNo	Name	....	....
8005	Ashish		
3098	Vikram		
4567	Nagendra		
8796	Rahul		

Table 2 : Columns are homogeneous

Another important property of table is each item value is atomic. That is, item can't be further divided. For example, take a name Item. It can have first name, middle name, or last name. Since these would be three different strings so they can't be placed under one column, Name. All the three parts are placed in three different columns. In this we can place them under, FirstName, MiddleName, and LastName. See fig. 3.

FirstName	MiddleName	LastName	
Jai	Prakash	Saini	....
Vijay	Pratap	Rawat	....
Deep	Chaanna	Singh	....

Table 3 : Table columns can have atomic values

Every table must have a primary key. Primary key is some column of the table whose values are used to distinguish the different records of the table. We'll take up primary key later in the session. There must be some column having distinct value in all rows by which one can identify all rows. That is all rows should be unique. See fig. 4.

EmpNo	Ename	DOJ
8005	Gargi	15-Jul-1998
8012	Rohit	01-Jul-1998
8075	Rohit	15-Jul-1998
8045	Sachin	15-Jul-1998

Table 4 : Table with primary key "EmpNo" and degree "3"

In this table, EmpNo can be used as a primary key. Since it is the only column where the values are all distinct. Whereas in Ename there are two Rohit and in DOJ column, 15-Jul1998 is same for three row no 1,3, and 4. If we use DOJ as primary key then there would be three records that have same DOJ so there won't be any way to distinguish these three records. For this DOJ can't be a primary key for this table. For similar reasons, Ename cannot be used as primary key.

Next property we are going to discuss is for ordering of rows and columns within a table. Ordering is immaterial for both rows and columns. See fig. 5. Table (a) and (b) represent the same table.

DName	DeptId	Manager
SD	1	GARGI
HR	2	ROHIT
FIN	4	SAURAB
EDU	7	SWATI

(a)

Manager	DName	DeptId
ROHIT	HR	2
SWATI	EDU	7
SAURAB	FIN	4
GARGI	SD	1

(B)

Table 5 : Ordering of rows and columns in a table is immaterial

Names of columns are distinct. It is not possible to have two columns having same name in a table. Since a column specifies a attribute, having two columns with same name mean that we are specifying the same property in two columns, which is not acceptable.

Total number of columns in a table specifies its degree. A table with n columns is said to have degree n. See fig. 1. Table represented there is of degree 3.

Domain - Domain is set of all possible values for an attribute. For example there is an Employee table in which there is a Designation attribute. Suppose, Designation attribute can take "PM", "Trainee", "AGM", or "Developer". Then we can say all these values make the domain for the attribute Designation. An attribute represents the use of a domain within a relation. Similarly for name attribute can take alphabetic strings. So domain for name attribute will be set of all possible valid alphabetic strings.

Keys - Now we'll take up another feature of relational tables. That is different type of keys. There are different types of keys, namely Primary keys, alternate keys, etc. The different types of keys are described below.

Primary Key : Within a given relation, there can be one attribute with values that are unique within the relation that can be used to identify the tuples of that relation. That attribute is said to be primary key for that relation.

Composite Primary Key - Not every relation will have single-attribute primary key. There can be a possibility that some combination of attribute when taken together have the unique identification property. These attributes as a group is called composite primary key. A combination consisting of a single attribute is a special case.

Existence of such a combination is guaranteed by the fact that a relation is a set. Since sets don't contain duplicate elements, each tuple of a relation is unique with respect to that relation. Hence, at least the combination of all attributes has the unique identification property.

In practice it is not usually necessary to involve all the attributes-some lesser combination is normally sufficient. Thus, every relation does have a primary (possibly composite) key.

Tuples represent entities in the real world. Primary key serves as a unique identifier for those entities.

Candidate key - In a relation, there can be more than one attribute combination possessing the unique -identification property. These combinations, which -can act as primary key, are called candidate keys.

EmpNo	SocSecurityNo	Name	Age
8005	1000076	Gargi	16
1000	8907769	Saurabh	24
3000	7654444	Bhaskar	25

Table 6 : Table having "EmpNo" and :SocSecurityNo" as candidate keys

Alternate Key - A candidate key that is not a primary key is called an alternate key. In fig. 6 if EmpNo is primary key then SocSecurityNo is the alternate key.

Integrity Rules :

1. Entity Integrity - It says that no component of a primary key may be null. All entities must be distinguishable. That is, they must have a unique identification of some kind. Primary keys perform unique identification function in a relational database. An identifier that was wholly null would be a contradiction in terms. It would be like there was some entity that did not have any unique identification. That is, it was not distinguishable from other entities. If two entities are not

distinguishable from each other, then by definition there are not two entities but only one.

Integrity Rule 2: Referential Integrity - The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations.

Suppose we wish to ensure that value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another. This is referential integrity.

The referential integrity constraint states that, a tuple in one relation that refers to another relation must refer to the existing tuple in that relation.

This means that the referential integrity is a constraint specified on more than one relation. This ensures that the consistency is maintained across the relations.

Table: A

DeptId	DName	DManager
S-10	Software	Gargi
H-09	HR	Rajat
P-44	Personal	Swami

Table: B

EmpNo	DeptId	EName
1000	S-IO	Swati
9098	S-IO	Saurabh
6578	H-09	Rajiv
5555	P-44	Akash

Table 7 : Table B as DeptId as foreign key, as it acts as primary key in table A

Extensions and Intensions - A relational in a relational database has two components, an extension and an intension.

Extension - The extension of a given relation is the set of tuples appearing in that relation at any given instance. The extension thus varies with time. It changes as tuples are created, destroyed, and updated.

Relation: Employee at time= t1

EmpNo	EName	Age	Dept
8005	Gargi	22	SO
9000	Rajiv	25	HR
2340	Swati	30	Fin
7689	Ashish	40	Fin

Relation: Employee at time= t2 after adding more records

EmpNo	EName	Age	Dept
8005	Gargi	22	SO
9000	Rajiv	25	HR
2340	Swati	30	Fin
7689	Ashish	40	Fin
7999	Ankita	20	SO
8000	Mukesh	23	SO

Relation: Employee at time= t3 after deleting more records

EmpNo	EName	Age	Dept
8005	Gargi	22	SO
7999	Ankita	20	SO

Table 8 : Extensions of relation Employee

Intension - The intension of a given relation is independent of time. It is the permanent part of the relation. It corresponds to what is specified in the relational schema. The intension thus defines all permissible extensions. The intension is a combination of two things: a structure and a set of integrity constraints.

- The naming structure consists of the relation name plus the names of the attributes (each with its associated domain name).
- The integrity constraints can be subdivided into key constraints, referential constraints, and other constraints.

For example :

Employee (EmpNo Number(4) Not NULL, EName Char(20), Age Number(2), Dept Char(4))

This is the intension of Employee relation.

Key Constraints - Key constraint is implied by the existence of candidate keys. The intension includes a specification of the attribute(s) consisting the primary key and specification of the attribute(s) consisting alternate keys, if any. Each of these specifications implies a uniqueness constraint (by definition of candidate key); in addition primary key specification implies a no-nulls constraint(by integrity rule I).

Referential Constraints - Referential constraints are constraints implied by the existence of foreign keys. The intension includes a specification of all foreign key in the relation. Each of these specifications implies a referential constraint (by integrity rule 2).

Other constraints - Many other constraints are possible in theory.

Examples - salary>= 10000.

Relational Algebra - Once the relationships are identified, then operations that are applied on the relations are also identified. In relational model, the operations are performed with the help of relational algebra. Relational algebra is a collection of operations on relations. Each operation takes one or more relations as its operand(s) and produces another relation as its result. We can compare relational algebra with traditional arithmetic algebra. In arithmetic algebra, there are operators that operate on operands(data values) and produce some result. Similarly in relational algebra there are relational operators that operate upon relations and produce relations as results.

Relational Algebra can be Divided into Two Groups:

1. Traditional set operators that include union, intersection, difference, and Cartesian product.

2. Special relational operators that include selection, projection, and division.

Traditional Set Operators:

Union :- The union of two relations A and B is the set of all tuples belonging to either A or B (or both).

Example :

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.

A UNION B = The set of employees whose are either in S/W development department or having age less than 30 years.

Intersection : The intersection of two relations A and B is the set of all tuples t belonging to both A and B.

Example :

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.

A INTERSECTION B = The set of employees whose are in S/W development department having age less than 30 years.

Difference : The difference between two relations A and B(in that order) is the set of all tuples belonging to A and not to B.

Example :

A = The set of employees whose department is S/W Development
B = The set of employee whose age is less than 30 years.

A MINUS B = The set of employees whose department is S/W development and not having age less than 30 years.

Cartesian Product:- The Cartesian product of two relations A and B is the set of all tuples *t* such that t is the concatenation of a tuple a belonging to A and a tuple b belonging to B.

The concatenation of a tuple $a = (al \ldots\ldots\ldots\ldots am)$ and tuple $b=(bm+l \ldots\ldots\ldots\ldots bm+n)$- **in that order- is the tuple** $t = (al, \ldots\ldots\ldots\ldots, am, bm+l, \ldots\ldots\ldots\ldots bm+n)$.

Example :

A = The set of employees whose department is S/W Development

B = The set of employee whose age is less than 30 years.

A TIMES B = is the set of all possible employee no/department ids pairs.

Special Relational Operators :

Selection: The selection operator yields a 'horizontal' subset of a given relation- that is, the subset of tuples within the given relation for which a specified predicate is satisfied.

The predicate is expressed as a Boolean combination of terms, each term being a simple comparison that can be established as true or false for a given tuple by inspecting that tuple in isolation.

Book WHERE Author = 'Kruse'

BookId	BName	Author
A-IOO	DataStructure	Kruse
C-12	Software Engg	Kruse
0-99	Compiler	Kruse

Employee WHERE Desig='Manager' AND Dept ='SD'

EmpNo	EName	Desig	Dept
2000	Swati	Manager	SO
4353	Gargi	Manager	SO
6666	Saurabh	Manager	SD

Table 9 : Selection Operation

Projection: The projection yields a 'vertical' subset of a given relation- that is, the subset obtained by selecting specified attributes, in a specified left-to-right order, and then eliminating duplicate tuples within the attributes selected.

Example: Issue [BookId ReturnDate]

BookId	ReturnDate
S-100	20-May-99
S-97	31-Aug-99
0-10	31-Aug-99
0-99	02-Sep-99
E-04	15-Sep-99

Book [Book Name]

Book Name
Software Concepts
Data Structures
Programming
Assembly Language
SSAD
PC-Troubleshooting
Compiler Design

Table 10 : Projection operation

Division : The division operator divides a dividend relation A of degree m+n by a divisor relation B of degree n, and produces a result relation of degree m.

Let A be set of pairs of values <x, y> and B a set of single values, <y>. Then the result of dividing A by B - that is A DIVIDEDBY B- is the set of values x such that the pair- <x, y> appears in A for all values y appearing in B.

OENO

Memberld	BookId
P-100	S-10
P-IOO	S-12
P-100	0-07
A-OO	S-JO
M-09	S-IO
M-09	S-12
A-OO	S-12
M-09	0-07

OOR

BookId
S-10
S-12 0-07

DEND DIVIDED BY DOR

MemberID
P-100
M-09

Table 11 : Sample divisions

Now we know about the constructs of relational data model. We also know how to specify constraints and how to use relational algebra for illustrating various functions. We now take up another data model that is entirely different from relational model.

6. Object Oriented Model :

Now we know about the constructs of relational data model. We also know how to specify constraints and how to use relational algebra for illustrating various functions. We now take up another data model that is entirely different from relational model.

Now days people are moving towards the object oriented approach in many fields. These fields include programming, software engineering, development technologies, implementation of databases, etc. Object oriented concepts have their roots in the object oriented programming languages. Since programming languages were the first to use these concepts in practical sense. When these concepts became widely popular and accepted, other areas started to implement these ideas in them. It became possible due to the fact the object-oriented concepts try to model things as they are. So today it is a common practice to use object-oriented concepts in data modeling.

Object Oriented Data Modeling Concepts : As discussed earlier Object Oriented Model (OOM) has adopted many features that were developed for object oriented programming languages. These include objects, inheritance polymorphism, and encapsulation.

In object-oriented model main construct is an object. (As in E-R model we have entities and in relational model then are relations similarly we have objects in 00 data modeling. So first thing that is done in OOM is to identify the objects for the systems. Examining the problem statement can do it. Other important task is to identify the various operations for these objects. It is easy to relate the objects to the real world, In the section that follows we will try to understand the basic concepts of OOM data model.

Objects and Object Identity : In this model, everything is modeled as objects. An object can be any physical or abstract thing. It can be a person, placed thing, or a concept. An object can be -used to model the overall structure not just a part of it. Also the behavior of the thing that is being modeled is also specified in the object. This feature is called encapsulation. Encapsulation is discussed later in the chapter. Only thing we need to know at this stage is object can store information and behavior in the same entity i.e. an object. Suppose a car is being modeled using this method. Then we can have an object 'Car' that has the following information,

Car: Color, Brand, ModelNo, Gears, EngineCylinders, Capacity, No of gates. All this information is sufficient to model any car.

All the objects should be unique. For this purpose, every object is given an identity. Identity is the property of an object, which distinguishes it from all other object. In OOM databases, each object has a unique identity. This unique identity is implemented via a unique, system generated object identifier (OID). OID is used internally by the system to identity each object uniquely and to create and manage inter-object references. But its value is not visible to the user.

The main properties of OID :

1) It is Immutable: The value of OID for a particular object should not change. This preserves the identity of the real world object being represented.

2) It is Used Only Once: Even if an object is removed its **OID is not assigned** to other objects.

The value of OID doesn't depend upon any attributes of the object since the values of attributes may change OID should not base on physical address of the object in memory since physical reorganization of the database could change the OID. There should be some mechanism for generating OIDs.

Another feature of OO databases is that objects may have a complex structure is due to contain all of the significant information that describes the object. In contrast, in traditional database systems, information about a complex object is often scattered over many relations or records. See fig. 14. It leads to loss of direct correspondence between a real-world object and its database representation.

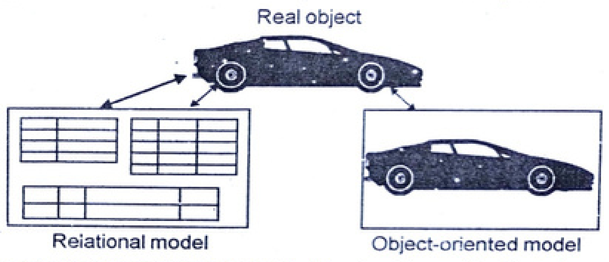


Fig. 14 : Databases store objects 'whole' not as disassembled data elements

The internal structure of an object includes the specification of instance variables, which hold the values, that defines the internal state of the object.

In OO databases, the values (or states) of complex objects may be constructed from other objects. These objects may be represented as a triple t(i, c, v), where i is a unique object identifier, c is a constructor (that is, an indication of how the object value is constructed), and v is the object value (or state).

There can be several constructors, depending upon or the OO system. The basic constructors are the atom, tuple, set, list, and array constructors. There is also a domain D that contains all basic atomic values that are directly available in the system. These include integers, real numbers, character strings, Boolean, dates, and any other data types that the system supports directly.

An object value v is interpreted on the basis of the value of the constructor c in the triple (i,c,v) that represents the object.

If c = atom, the value is an atomic value from domain D of basic values supported by the system.

If c = set, the value v is a set of objects identifiers{il,i2, ... ,in), which are the Identifiers (OIDs) for a set of objects that are typically of the same type.

If c= tuple, the value v is a tuple of the form <al:il, ... , an:in), where each aj is an attribute name (sometimes called an instance variable name in 00 terminology) and each it is an object identifier (OID).

If c = list, the value v is an ordered list of object identifiers [il ,i2, ... , in] of the same type. For c = array, the value v, is an array of object identifier.

Consider the following example:

01 = (i1, atom, Rohit)
02 = (i2, atom, Jai)
03 = (i3, atom, Gargi)
04 = (i4, set, {i1,i2,i3})
05 = (is, atom, SE)
06 = (i6, atom, NEPZ)
07 = (i7,tuple,<DNAME:i5,DNUMBER:i8,LOCATION:i6,ENAME:i3"
08 = (i8,atom,1)

Here value of object 4 is constructed from object values of objects 01, 02, and 03. Similarly value of object 7 is constructed from the value of 01, 03, 06, and 08.

These constructors can be used to define the data structures for an 00 database schema.

Encapsulation of Operations, Methods and Persistence : Encapsulation is related to the concepts of abstract data types and information hiding in programming languages. Here the main idea is to define the behavior of a type of object based on the operations that can be externally applied to objects of that type. The internal structure of the object is hidden, and the object is only accessible through a number of predefined operations. Some operations may be used to create or destroy objects; other operations may update the object value and other may be used to retrieve parts of the object value or to apply some calculations to the object value.

The external users of the object are only made aware of the interface of the object, which defines the names and arguments of each operation. The implementation of the object is hidden from the external users; it includes the definition of the internal data structure of the object and the implementation of the operations that access these structures.

In OO terminology, the interface part of each operation is called the signature, and the operation implementation is called a method, A method is invoked by sending a message to the object to execute the corresponding method,

Not all objects are meant to be stored permanently in the database. Transient objects exist in the executing program and disappear once the program terminates.

Persistent objects are stored in the database and persist after program terminates. The typical mechanism for persistence involves giving an object a unique persistent name through which it can be retrieved.

Inheritance : Inheritance is deriving objects from existing objects. The derived objects inherit properties from their parent object. Parent objects are those objects from which other objects are derived. Inheritance is a way of reusing the existing code.

Polymorphism : Polymorphism concept allows the same operator name or symbo I to be bound to two or more different implementation of the operator, depending on the type of objects to which the operator is applied.

Major features of OO databases can be summarized as following

OO databases store persistent objects permanently on secondary storage, and allow the sharing of these objects among multiple programs and applications.

OO databases provide a unique system-generated object identifier for each object. OO databases maintain a direct correspondence between real-world and database objects so that objects don't lose their integrity and identify and can be easily be identified and operated upon.

In OO databases, objects can be very complex in order to contain all significant information that may be required to describe the object completely.

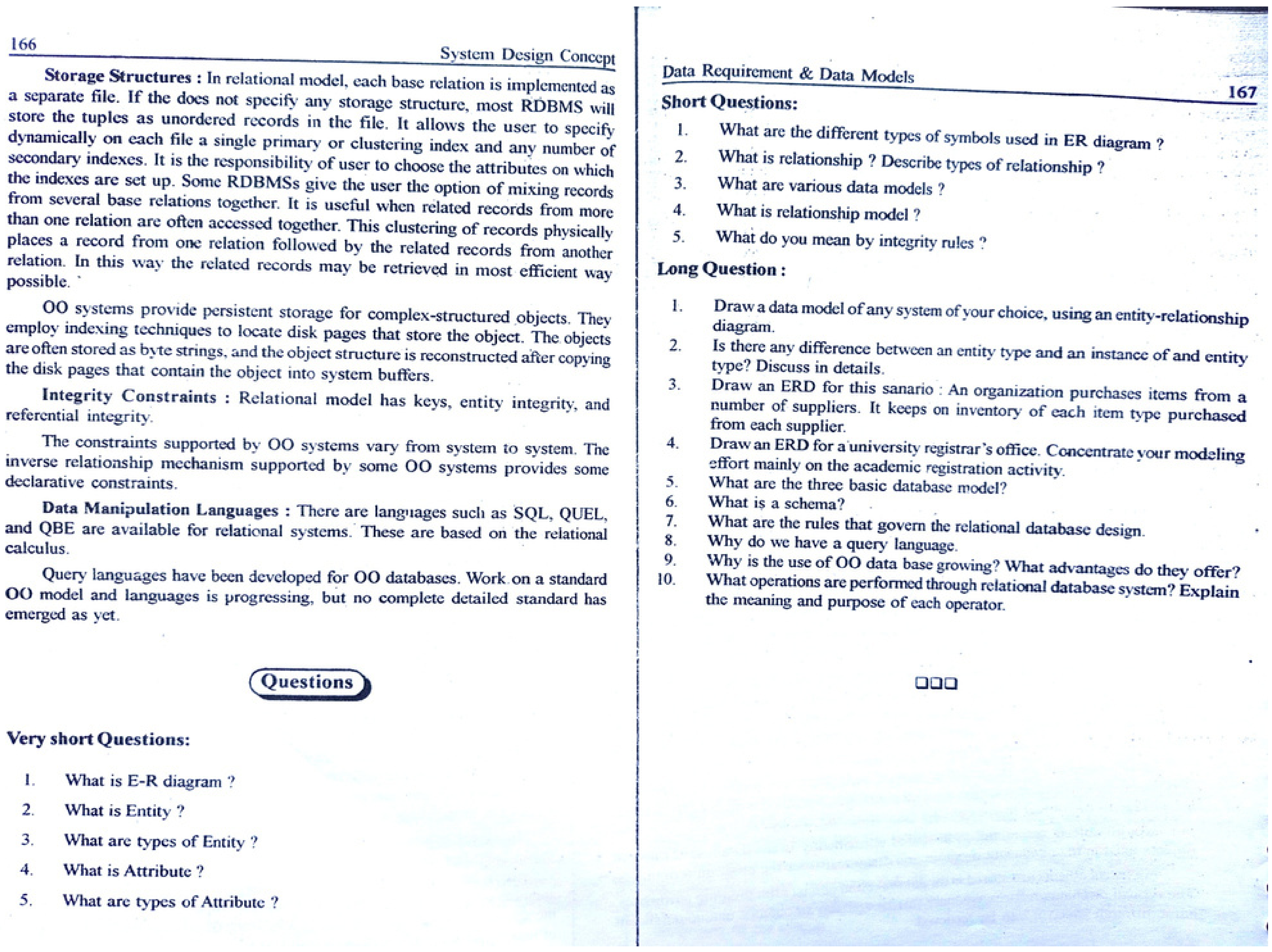
OO databases allow us to store both the class and state of an object between programs. They take the responsibility for maintaining the links between stored object behavior and state away from the programmer, and manage objects outside of programs with their public and private elements intact. They also simplify the whole process of rendering objects persistent by performing such tasks invisibly.

Persistence has to do with time i.e. a persistent object can exist beyond the program that created it. It also has to do with space (the location of the object may vary between processors, and even change its representation in the process).

Comparison : Now we know about both relational and object oriented approach, we can now compare these two models. In this session, we compare the relational model and object oriented model. We compare model representation capabilities, languages, system storage structures, and integrity constraints.

Data Model Representation : Different database models differ in their representation of relationships. In relational model, connections between two relations are represented by foreign key attribute in one relation that reference the primary key of another relation. Individual tuples having same values in foreign and primary key attribute are logically related. They are physically not connected. Relational model uses logical references.

In object oriented model, relationships are represented by references via the object identifier (OID). This is in a way similar to foreign keys but internal system identifiers are used rather than user-defined attributes. The OO model supports complex object structures by using tuple, set, list, and other constructors. It supports the specification of methods and the inheritance mechanism that permits creation of new class definitions from existing ones.



Storage Structures : In relational model, each base relation is implemented as a separate file. If the does not specify any storage structure, most RDBMS will store the tuples as unordered records in the file. It allows the user to specify dynamically on each file a single primary or clustering index and any number of secondary indexes. It is the responsibility of user to choose the attributes on which the indexes are set up. Some RDBMSs give the user the option of mixing records from several base relations together. It is useful when related records from more than one relation are often accessed together. This clustering of records physically places a record from one relation followed by the related records from another relation. In this way the related records may be retrieved in most efficient way possible.

OO systems provide persistent storage for complex-structured objects. They employ indexing techniques to locate disk pages that store the object. The objects are often stored as byte strings, and the object structure is reconstructed after copying the disk pages that contain the object into system buffers.

Integrity Constraints : Relational model has keys, entity integrity, and referential integrity.

The constraints supported by OO systems vary from system to system. The inverse relationship mechanism supported by some OO systems provides some declarative constraints.

Data Manipulation Languages : There are languages such as SQL, QUEL, and QBE are available for relational systems. These are based on the relational calculus.

Query languages have been developed for OO databases. Work on a standard OO model and languages is progressing, but no complete detailed standard has emerged as yet.

Questions

Very short Questions:

1. What is E-R diagram ?
2. What is Entity ?
3. What are types of Entity ?
4. What is Attribute ?
5. What are types of Attribute ?

Short Questions:

1. What are the different types of symbols used in ER diagram ?
2. What is relationship ? Describe types of relationship ?
3. What are various data models ?
4. What is relationship model ?
5. What do you mean by integrity rules ?

Long Question :

1. Draw a data model of any system of your choice, using an entity-relationship diagram.
2. Is there any difference between an entity type and an instance of and entity type? Discuss in details.
3. Draw an ERD for this sanario : An organization purchases items from a number of suppliers. It keeps on inventory of each item type purchased from each supplier.
4. Draw an ERD for a university registrar's office. Concentrate your modeling effort mainly on the academic registration activity.
5. What are the three basic database model?
6. What is a schema?
7. What are the rules that govern the relational database design.
8. Why do we have a query language.
9. Why is the use of OO data base growing? What advantages do they offer?
10. What operations are performed through relational database system? Explain the meaning and purpose of each operator.

❑❑❑