

UNIT 3: Analysis

Structured and Object-Oriented Analysis

Analysis simple means to study or examine the structure of something, elements, and system requirements in detail, and methodical way. Structured analysis and Object-oriented analysis both are important for software development and are analysis techniques used in software engineering. But both are different from each other.

Structured Analysis and Object-Oriented Analysis (OOA) are two software development methodologies that are used to design and develop software systems. While they have some similarities, they differ in a number of key ways.

Structured Analysis:

1. Focus on processes: Structured Analysis focuses on the processes involved in a software system, modeling them as a series of connected steps.
2. Top-down approach: Structured Analysis follows a top-down approach, breaking down complex systems into smaller, simpler parts that can be more easily understood.
3. Data-centered: Structured Analysis focuses on the data that a software system manipulates, modeling it as data flows between processes.
4. Emphasis on functional decomposition: Structured Analysis emphasizes the functional decomposition of a software system into smaller, independent functions.

Object-Oriented Analysis (OOA):

1. Focus on objects: OOA focuses on the objects involved in a software system, modeling them as instances of classes that encapsulate both data and behavior.
2. Bottom-up approach: OOA follows a bottom-up approach, building complex systems from smaller, simpler objects that can be more easily understood.
3. Object-centered: OOA focuses on the objects that make up a software system, modeling their relationships and interactions.

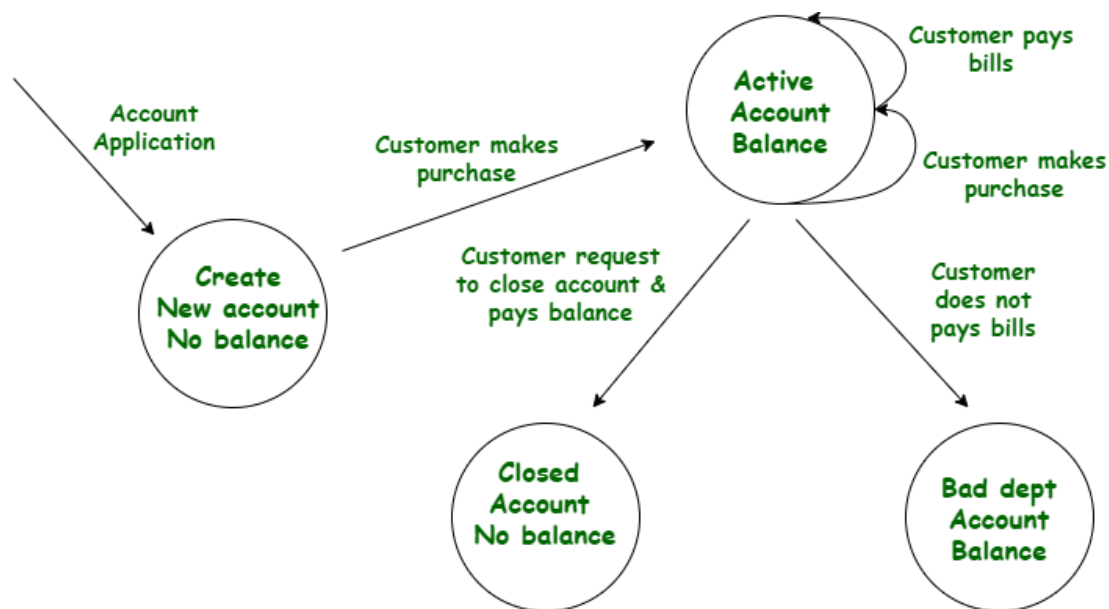
4. Emphasis on object-oriented design patterns: OOA emphasizes the reuse of objects and object-oriented design patterns, reducing the amount of code that needs to be written and improving the quality and consistency of the software.

In conclusion, Structured Analysis and OOA are both valuable software development methodologies, each with its own strengths and weaknesses. The choice of which to use depends on the particular requirements and constraints of a software project.

1. Structured Analysis :

Structured analysis is a method of development that allows and gives permission to the analyst to understand and know about the system and all of its activities in a logical way. It is simply a graphic that is used to specify the presentation of the application.

Example –

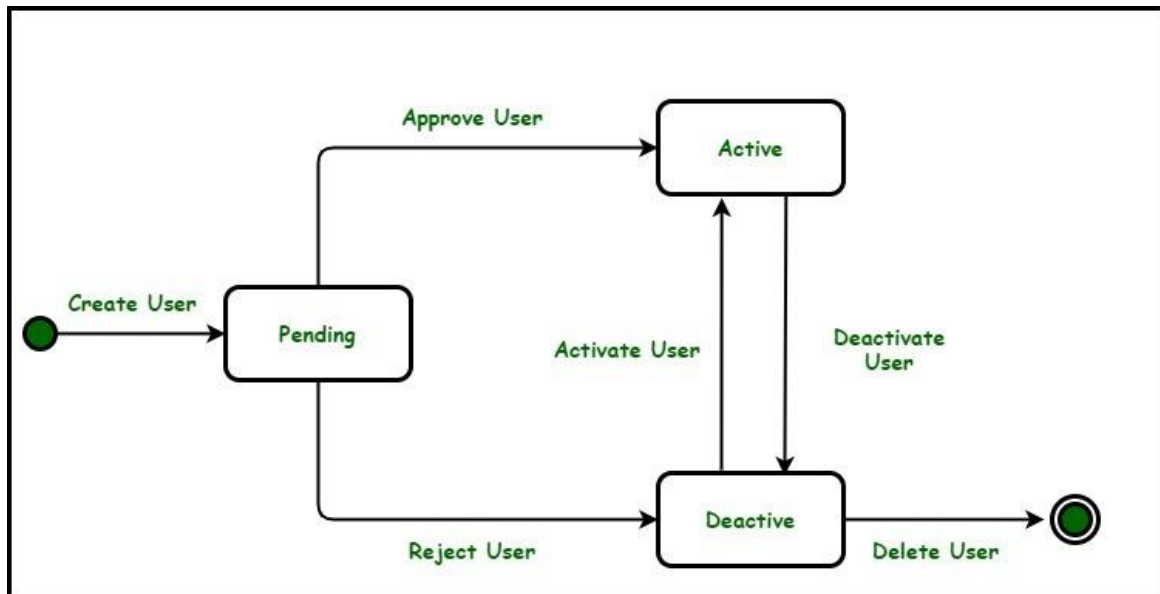


State Transition Diagram Example

2. Object-Oriented Analysis :

Object-Oriented Analysis (OOA) is a technical approach generally used for analyzing and application designing, system designing, or even business designing just by applying object-oriented programming even with the use of visual modeling throughout the process of development to just simply guide the stakeholder communication and quality of the product. It is actually a process of discovery where a team of developers understands and models all the requirements of the system.

Example –



Text

StateChart Diagram Example

Difference Between Structured and Object-oriented analysis :

Structured Analysis	Object-Oriented Analysis
The main focus is on the process and procedures of the system.	The main focus is on data structure and real-world objects that are important.
It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.	It uses Incremental or Iterative methodology to refine and extend our design.
It is suitable for well-defined projects with stable user requirements.	It is suitable for large projects with changing user requirements.

Risk while using this analysis technique is high and reusability is also low.	Risk while using this analysis technique is low and reusability is also high.
Structuring requirements include DFDs (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram.	Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), Statechart Diagram, and deployment diagram.
This technique is old and is not preferred usually.	This technique is new and is mostly preferred.

Advantages of Structured Analysis:

1. Clear and straightforward: Structured Analysis is a clear and straightforward methodology that is easy to understand and implement.
2. Top-down approach: The top-down approach of Structured Analysis makes it easy to identify the high-level functions and processes involved in a software system, and to break them down into smaller, more manageable components.
3. Emphasis on data: Structured Analysis places a strong emphasis on the data that a software system manipulates, making it easier to understand the relationships between data and processes.
4. Well-suited for small to medium-sized systems: Structured Analysis is well-suited for small to medium-sized systems, where the focus is on breaking down complex systems into simpler, more manageable components.

Disadvantages of Structured Analysis:

1. Limited scalability: Structured Analysis is limited in its scalability, and may become cumbersome when dealing with complex, large-scale systems.
2. Lack of object orientation: Structured Analysis does not provide the object orientation and encapsulation benefits of OOA, making it more difficult to manage and maintain large systems over time.

3. Limited ability to model complex relationships: Structured Analysis has a limited ability to model complex relationships between objects, making it less suitable for modeling large, complex systems.

Advantages of Object-Oriented Analysis (OOA):

1. Reusable code: OOA enables the creation of reusable objects and design patterns, reducing the amount of code that needs to be written and improving the quality and consistency of the software.
2. Scalability: OOA is more scalable than Structured Analysis, making it better suited for large, complex systems.
3. Object orientation: OOA provides the benefits of object orientation, including encapsulation, inheritance, and polymorphism, making it easier to manage and maintain large systems over time.
4. Better modeling of complex relationships: OOA enables better modeling of complex relationships between objects, making it better suited for modeling large, complex systems.

Disadvantages of Object-Oriented Analysis (OOA):

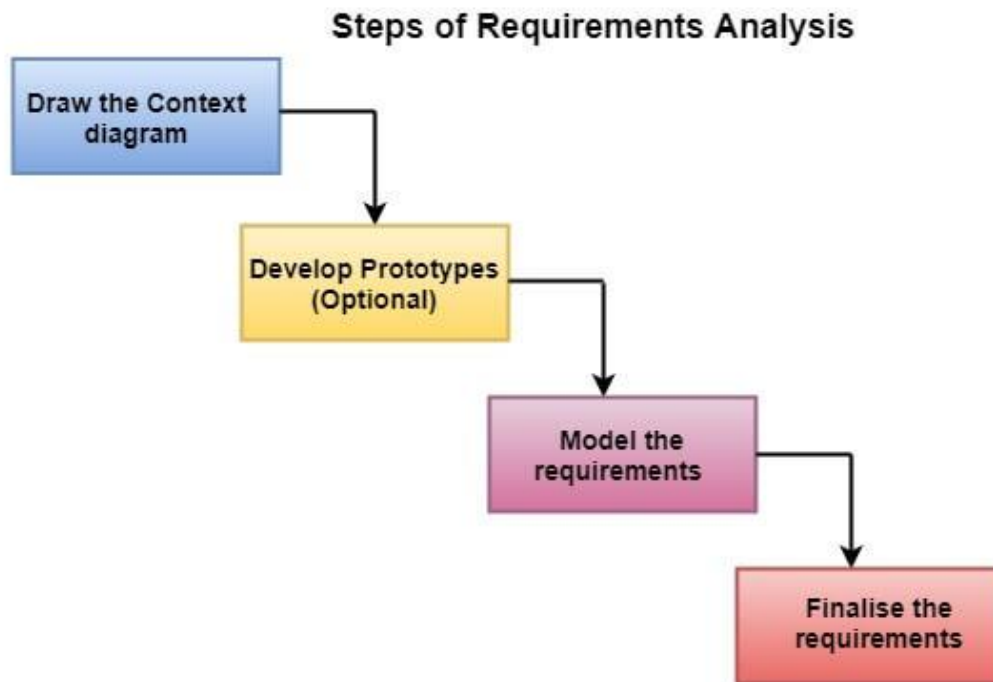
1. Steep learning curve: OOA can be more difficult to understand and implement than Structured Analysis, especially for those who are not familiar with object-oriented programming.
2. Bottom-up approach: The bottom-up approach of OOA can make it difficult to understand the high-level functions and processes involved in a software system, and to break them down into smaller, more manageable components.
3. Emphasis on objects: OOA places a strong emphasis on objects, making it more difficult to understand the relationships between data and processes.

Requirements Analysis

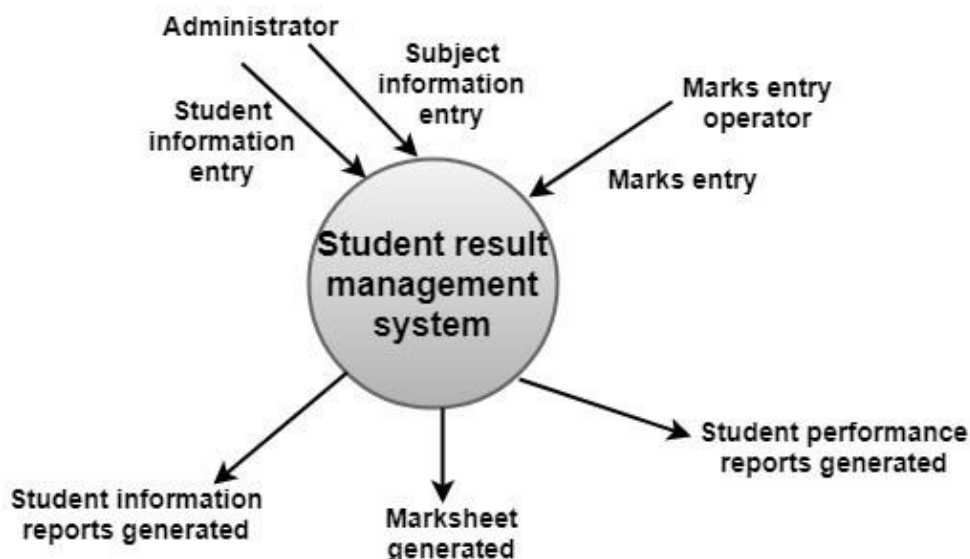
Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the

interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.


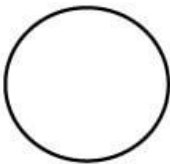

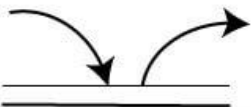
It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows in a flow chart represent the order of events; arrows in DFD represent flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represent decision points with multiple existing paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFDM

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y , then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

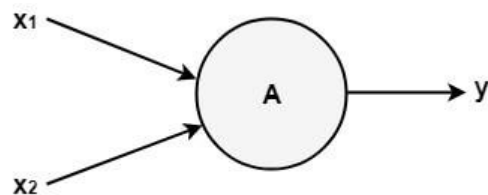


Fig: Level-0 DFD.

The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

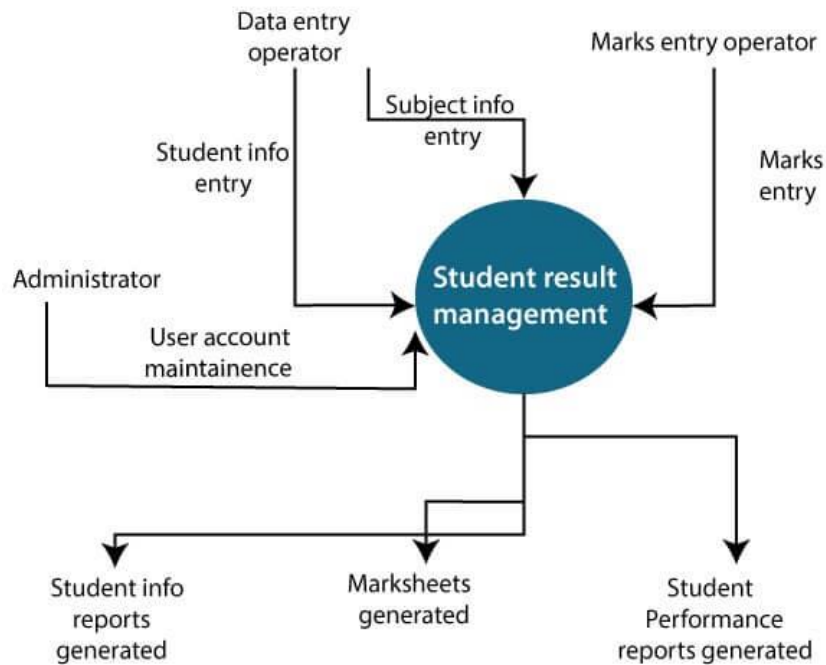


Fig: Level-0 DFD of result management system

1-level DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

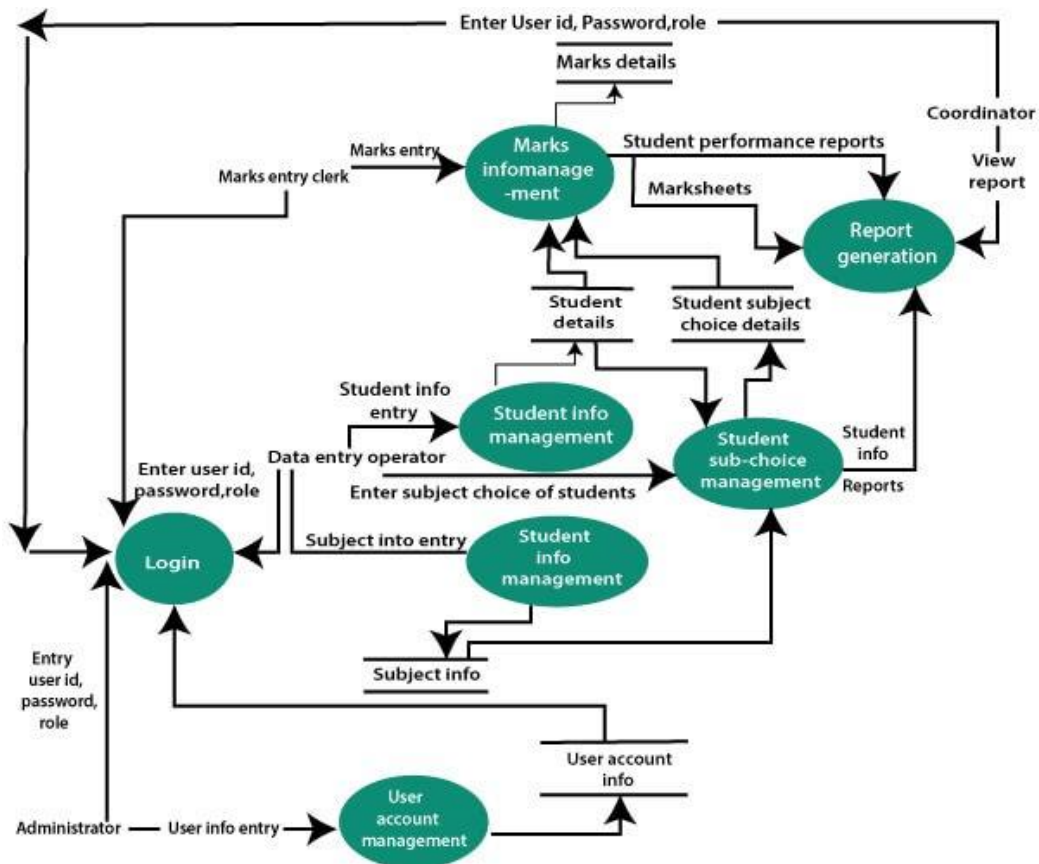
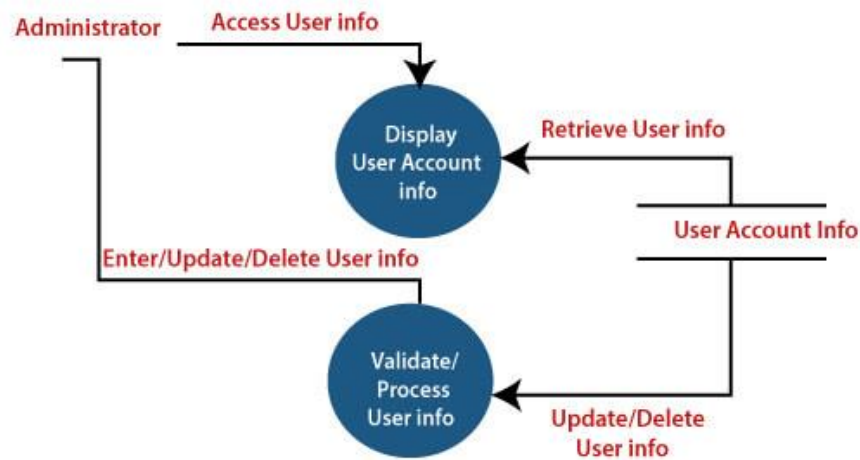


Fig: Level-1 DFD of result management system

2-Level DFD

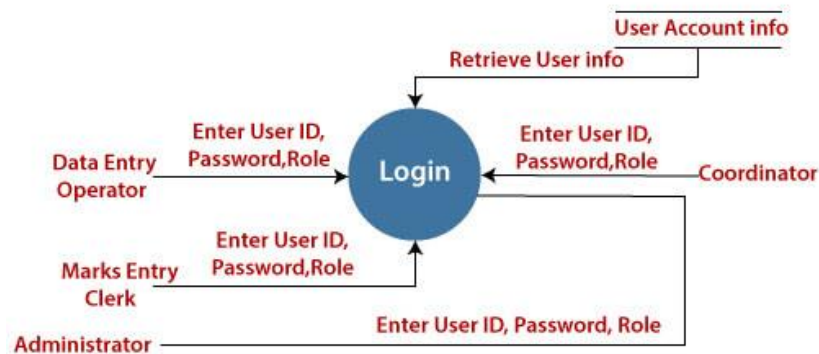
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

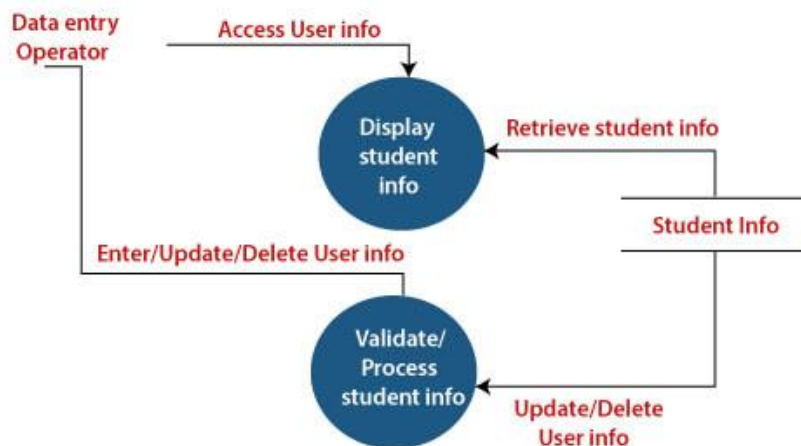


2. Login

The level 2 DFD of this process is given below:

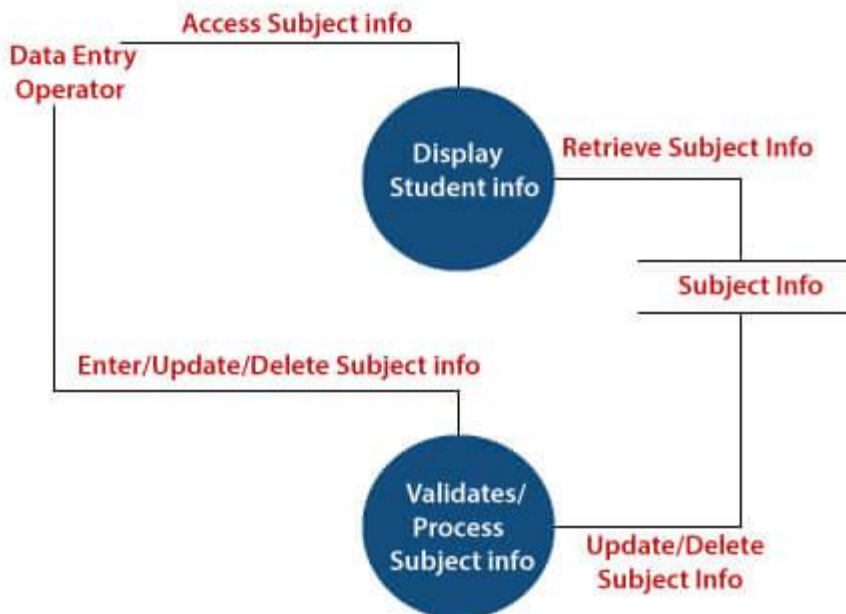


3. Student Information Management



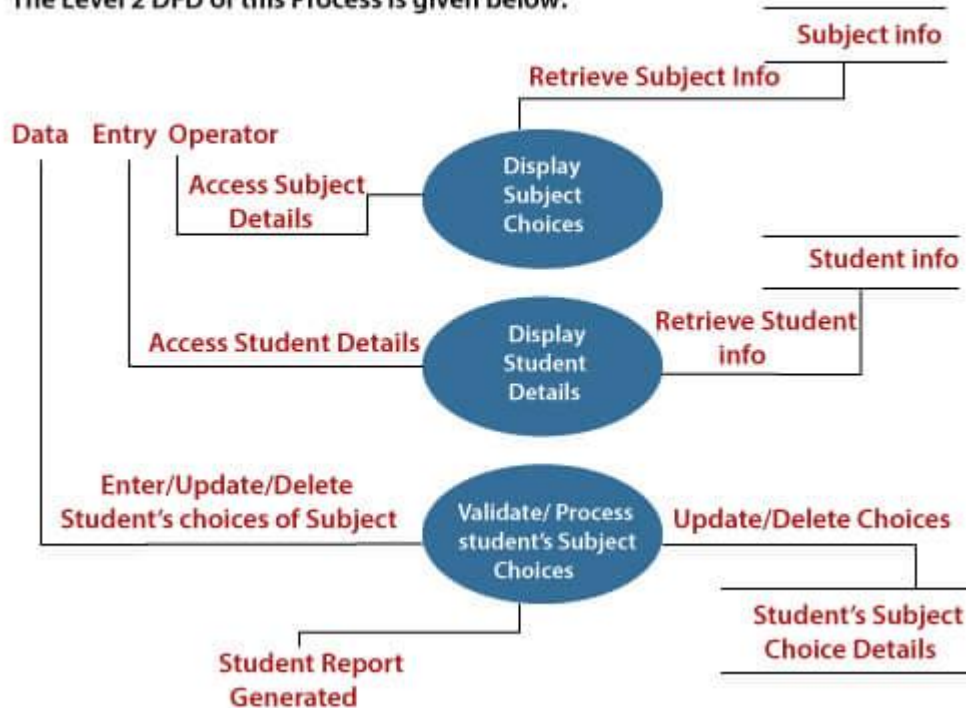
4. Subject Information Management

The level 2 DFD of this process is given below:



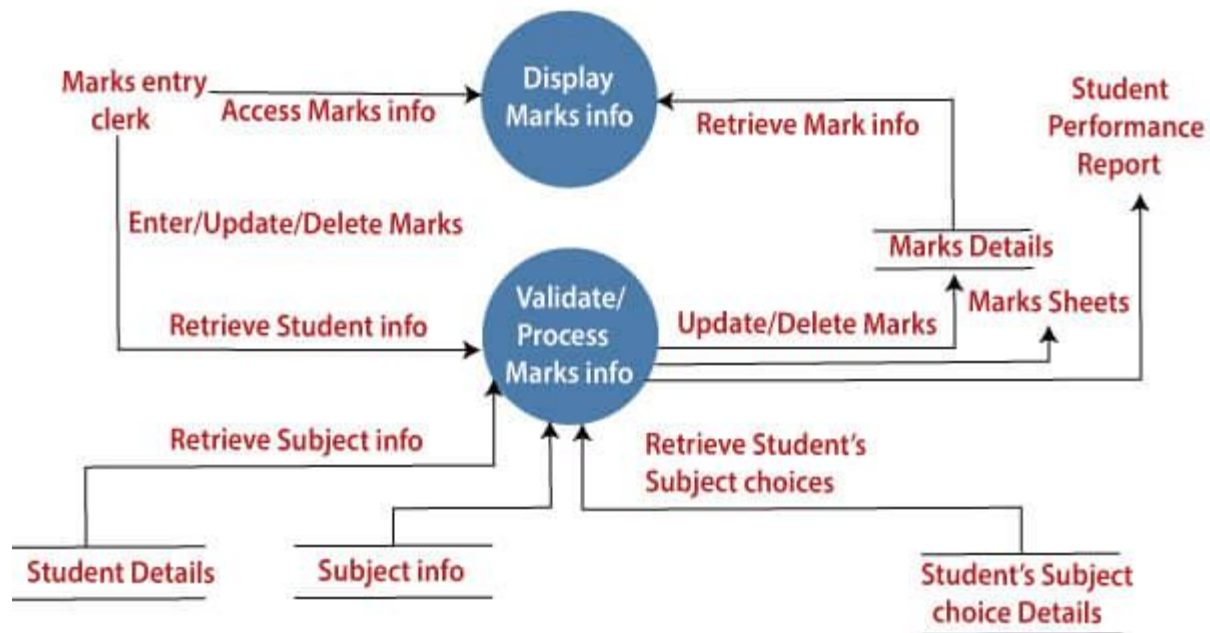
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



Entity-Relationship Diagrams

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD

- The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
- The ERD serves as a documentation tool.
- Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of an ER Diagrams

Components of a ER Diagram



1. Entity

An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

Entity Set

An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.

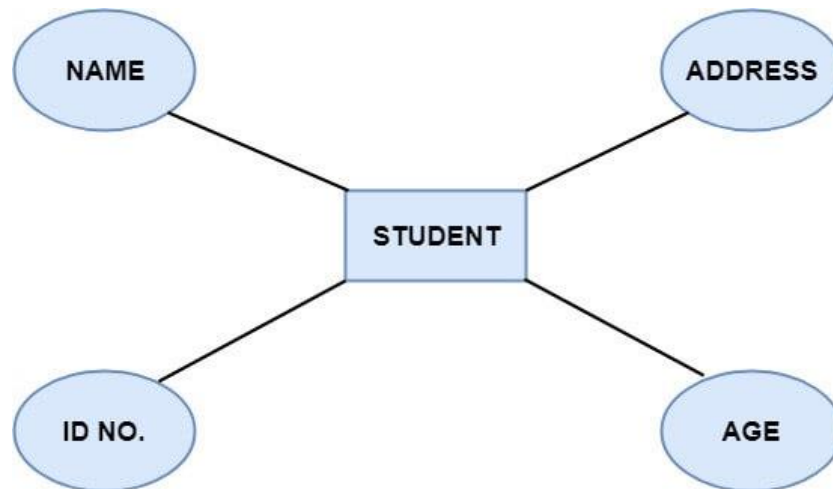
Advertisement



2. Attributes

Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

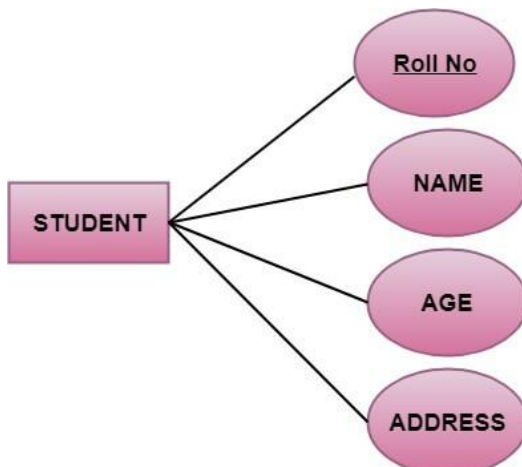
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

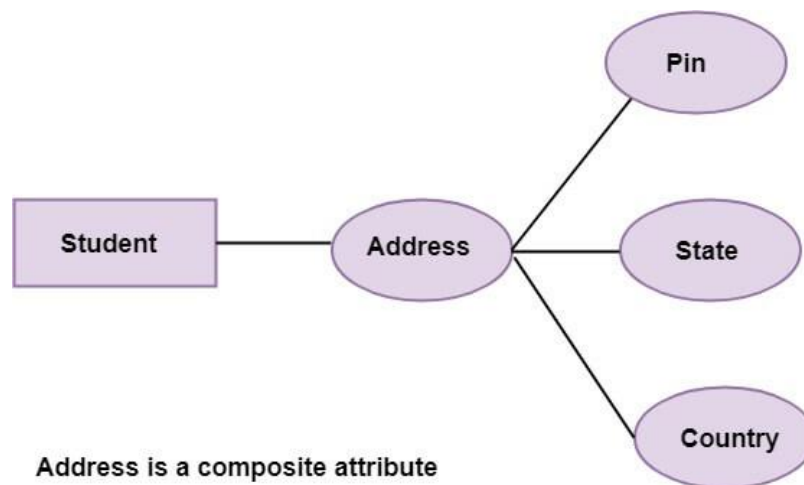
1. Key attribute: Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll_number of a student makes him identifiable among students.



There are mainly three types of keys:

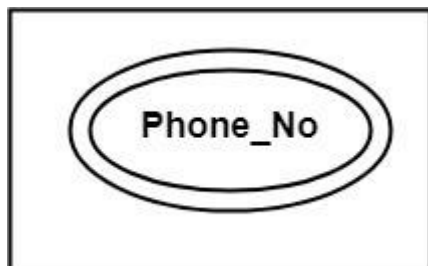
1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.
2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
3. **Primary key:** A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

2. Composite attribute: An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

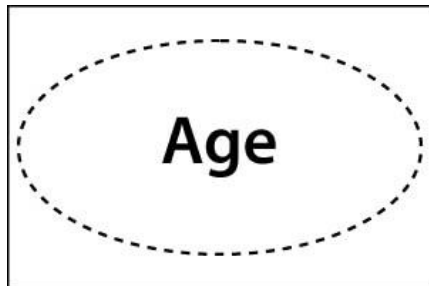


3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

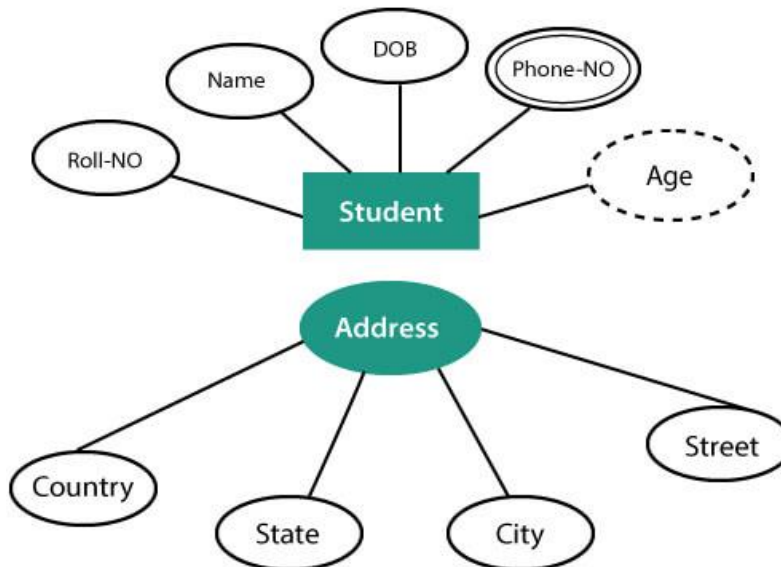
4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



5. Derived attribute: Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date_of_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.



The Complete entity type Student with its attributes can be represented as:



3. Relationships

The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.



Fig: Relationships in ERD

Relationship set

A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of a relationship set

The number of participating entities in a relationship describes the degree of the relationship. The three most common relationships in E-R models are:

1. Unary (degree1)
2. Binary (degree2)
3. Ternary (degree3)

1. Unary relationship: This is also called recursive relationships. It is a relationship between the instances of one entity type. For example, one person is married to only one person.

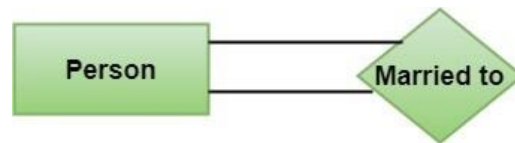


Fig: Unary Relationship

2. Binary relationship: It is a relationship between the instances of two entity types. For example, the Teacher teaches the subject.



Fig: Binary Relationship

3. Ternary relationship: It is a relationship amongst instances of three entity types. In fig, the relationships "**may have**" provide the association of three entities, i.e., TEACHER, STUDENT, and SUBJECT. All three entities are many-to-many participants. There may be one or many participants in a ternary relationship.

In general, "**n**" entities can be related by the same relationship and is known as **n-ary** relationship.

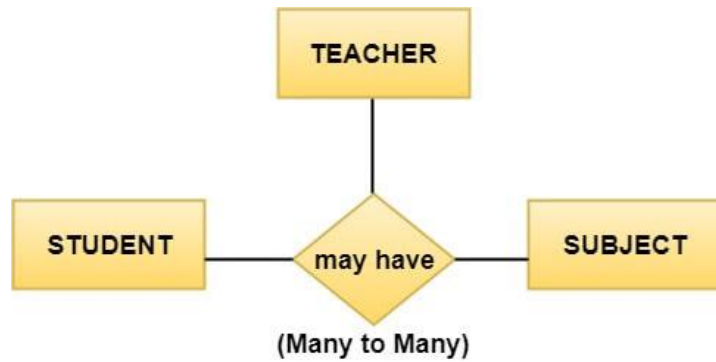


Fig: Ternary Relationship

Cardinality

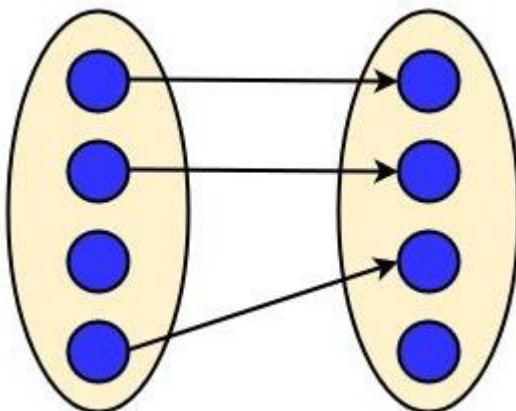
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

Types of Cardinalities

1. One to One: One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.



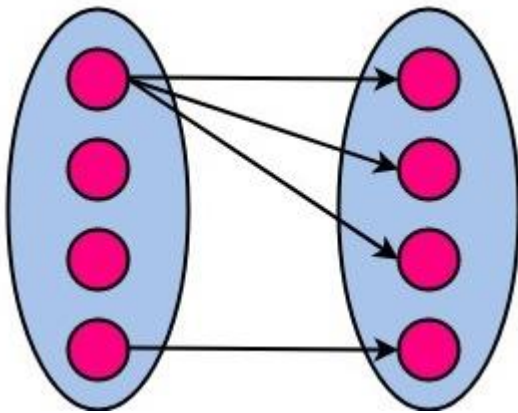
Using Sets, it can be represented as:



2. One to many: When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.



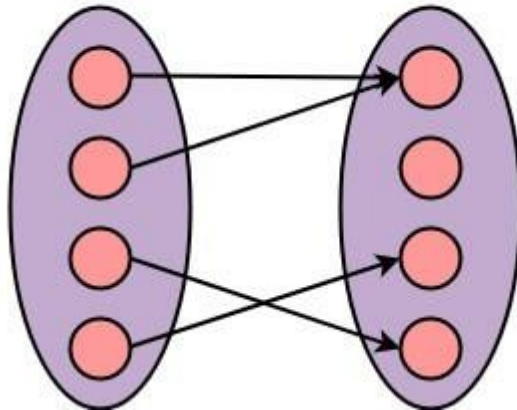
Using Sets, it can be represented as:



3. Many to One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. For example - many students can study in a single college, but a student cannot study in many colleges at the same time.



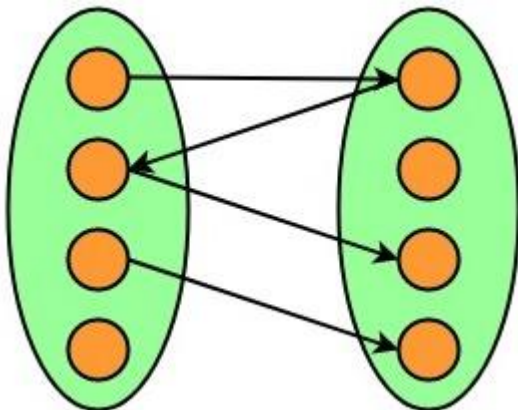
Using Sets, it can be represented as:



4. Many to Many: One entity from A can be associated with more than one entity from B and vice-versa. For example, the student can be assigned to many projects, and a project can be assigned to many students.



Using Sets, it can be represented as:



Data Dictionaries

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is an essential component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

The data dictionary, in general, includes information about the following:

- Name of the data item
- Aliases
- Description/purpose
- Related data items
- Range of values
- Data structure definition/Forms

The **name of the data item** is self-explanatory.

Aliases include other names by which this data item is called DEO for Data Entry Operator and DR for Deputy Registrar.

Description/purpose is a textual description of what the data item is used for or why it exists.

Related data items capture relationships between data items e.g., total_marks must always equal to internal_marks plus external_marks.

Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.

Data structure Forms: Data flows capture the name of processes that generate or receive the data items. If the data item is primitive, then data structure form captures the physical structures of the data item. If the data is itself a data aggregate, then data structure form capture the composition of the data items in terms of other data items.

The mathematical operators used within the data dictionary are defined in the table:

Notations	Meaning
$x=a+b$	x includes of data elements a and b.
$x=[a/b]$	x includes of either data elements a or b.
$x=a \ x$	includes of optimal data elements a.
$x=y[a]$	x includes of y or more occurrences of data element a
$x=[a]z$	x includes of z or fewer occurrences of data element a
$x=y[a]z$	x includes of some occurrences of data element a which are between y and z.

The data dictionary can be used to

