

Unit 2

Software Engineering

Prepared By

Abhishek Kesharwani

Assistant Professor ,United College of Engineering and Research

Index

- Data Flow Diagrams
- Entity Relationship Diagrams
- Decision Tables
- SRS Document
- IEEE Standards for SRS


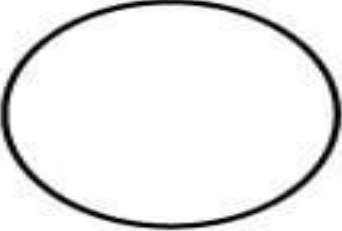

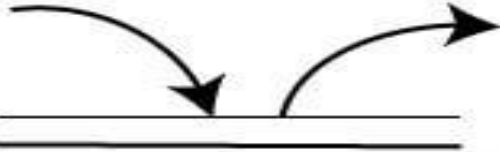
Data Flow Diagrams

- 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

- All names should be unique. This makes it easier to refer to elements in the DFD.
- Remember that DFD is not a flow chart. Arrows in a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
- 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 represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
- Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te 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.

- **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:

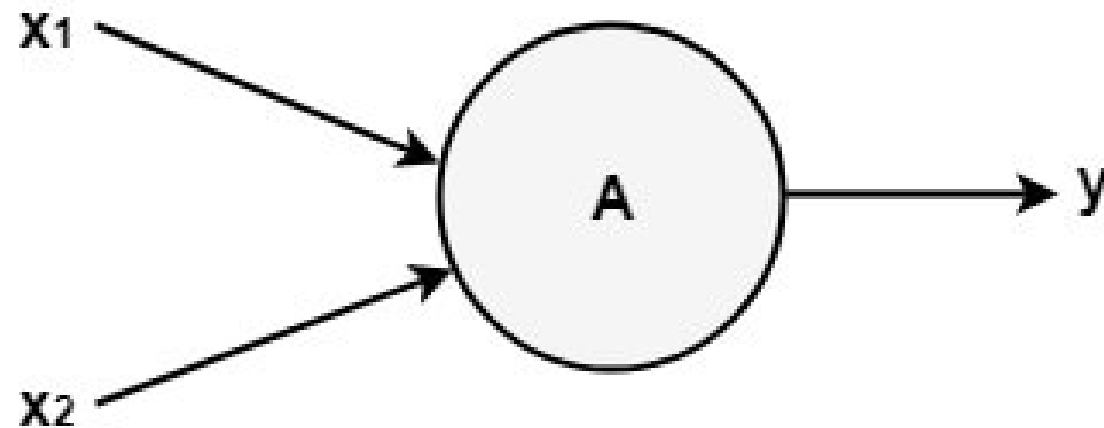
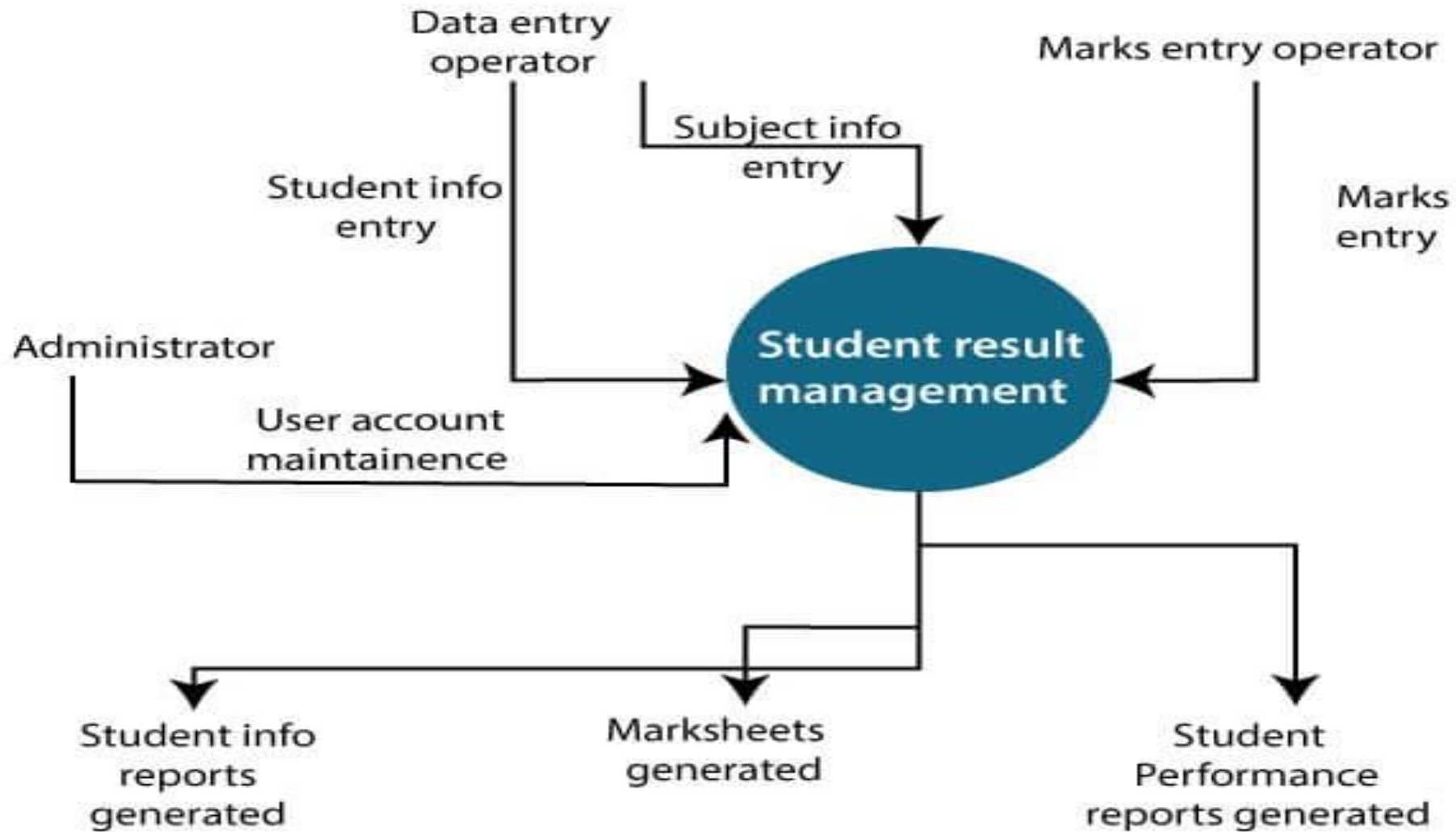


Fig: Level-0 DFD.

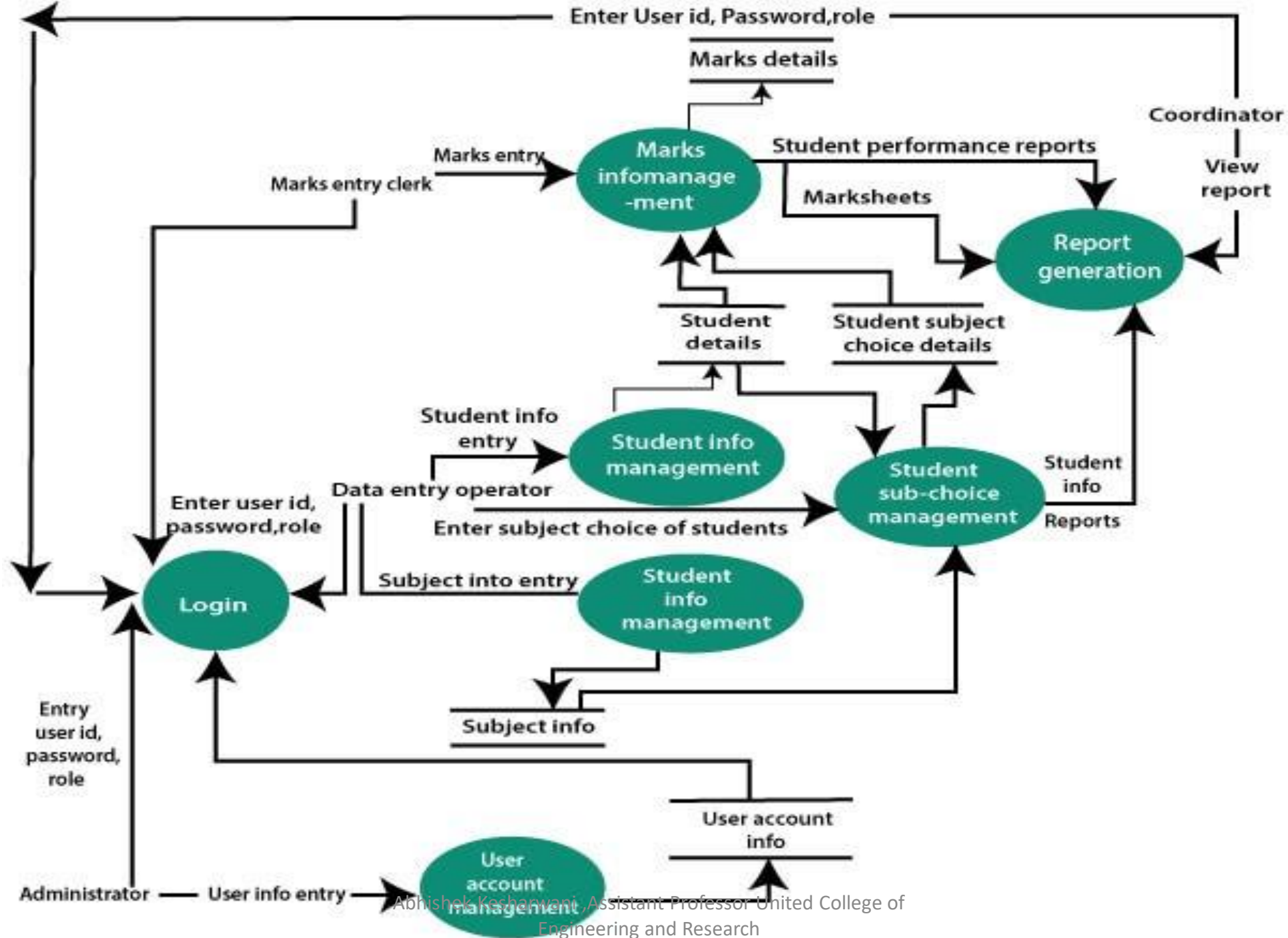
The Level-0 DFD, also called context diagram of the result management system is shown in fig.



Abhishek Kesharwani Assistant Professor United College of Engineering and Research
Fig: Level-0 DFD of result management system

1-level DFD

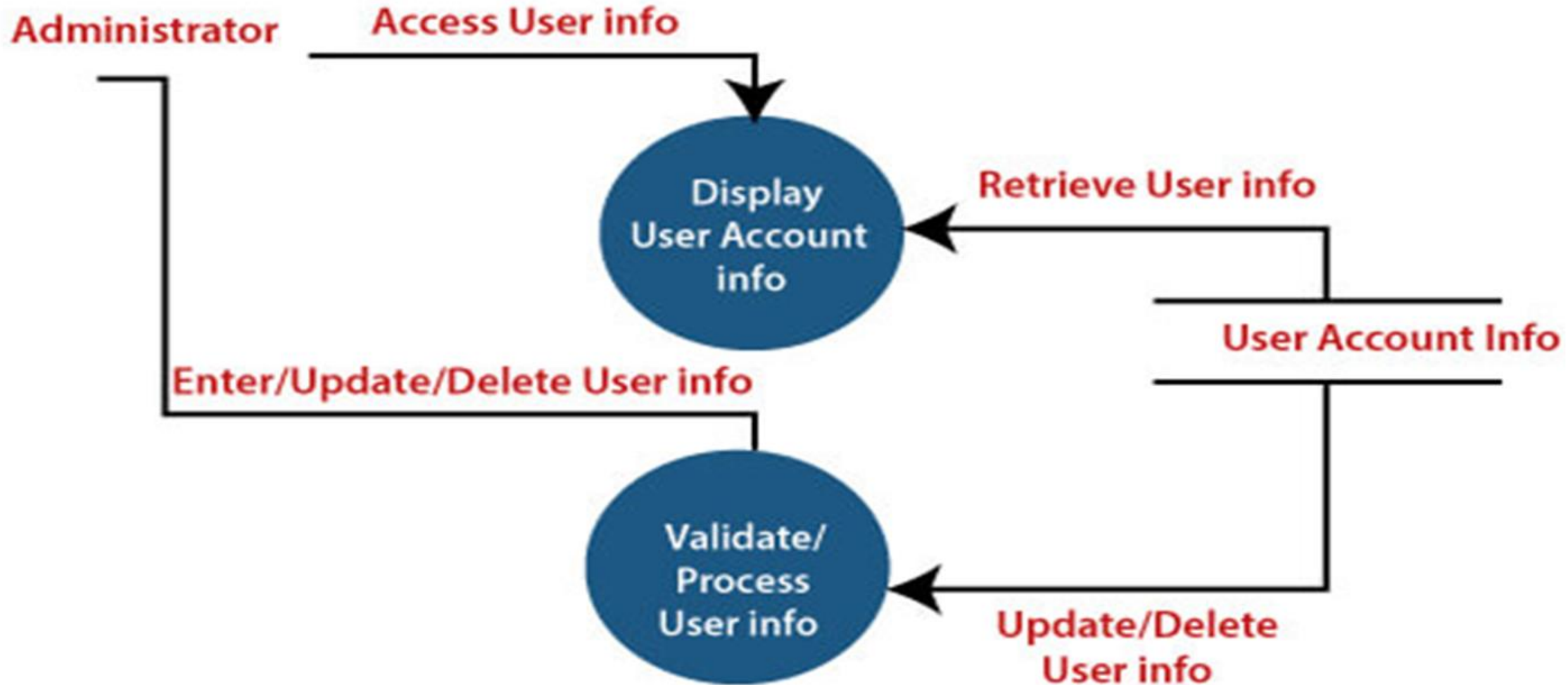
- In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes.
- In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.



2-Level DFD

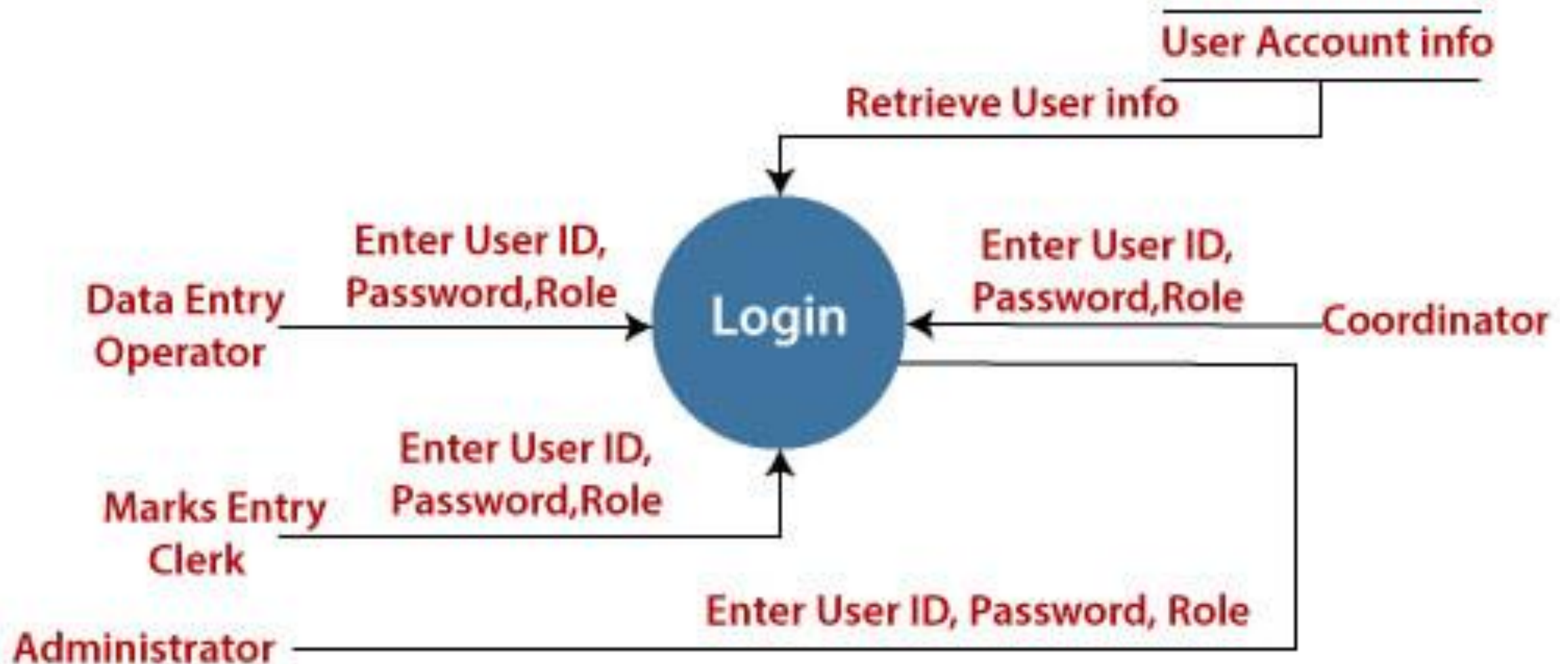
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

1. User Account Maintenance

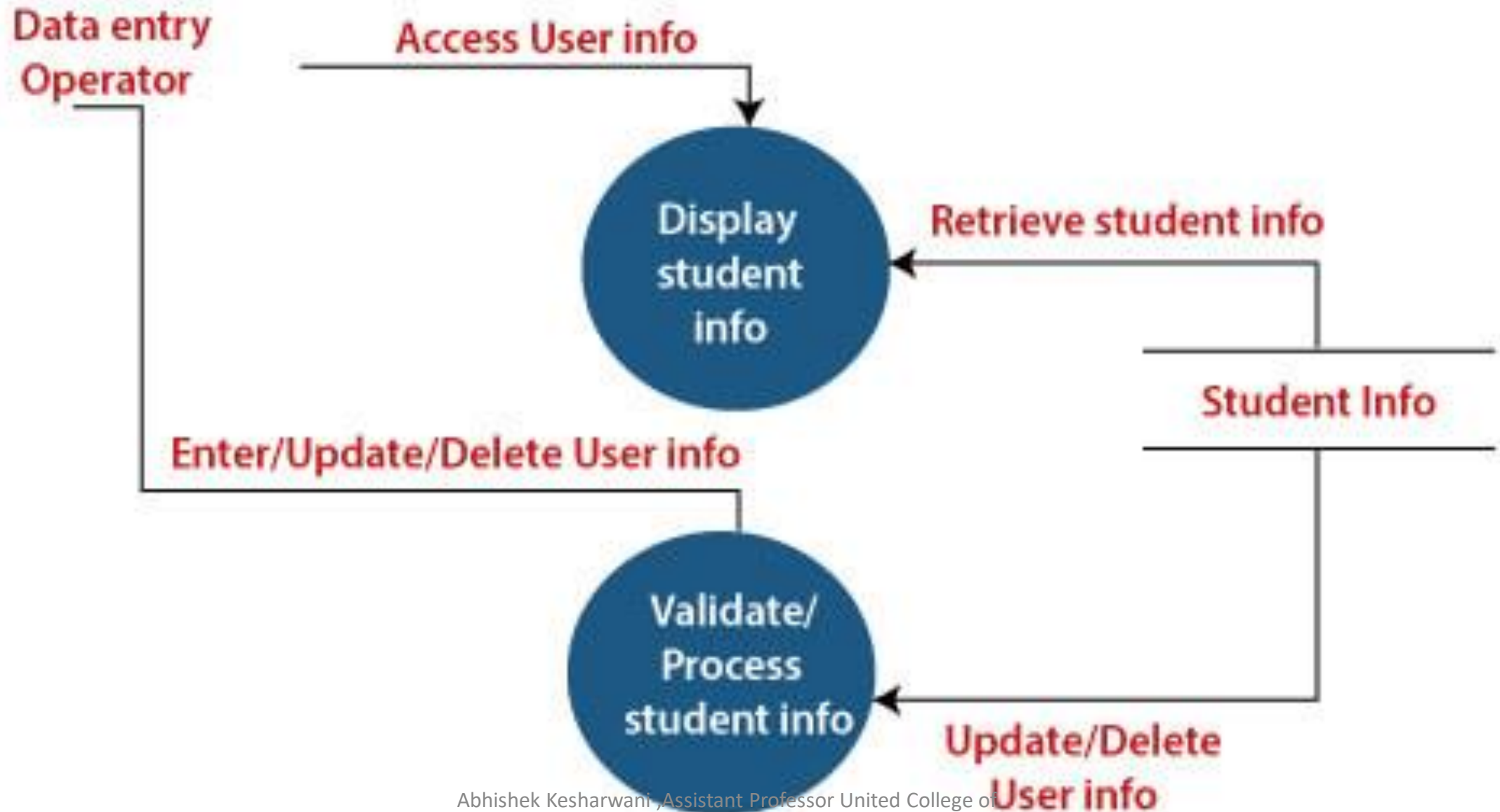


2. Login

The level 2 DFD of this process is given below:

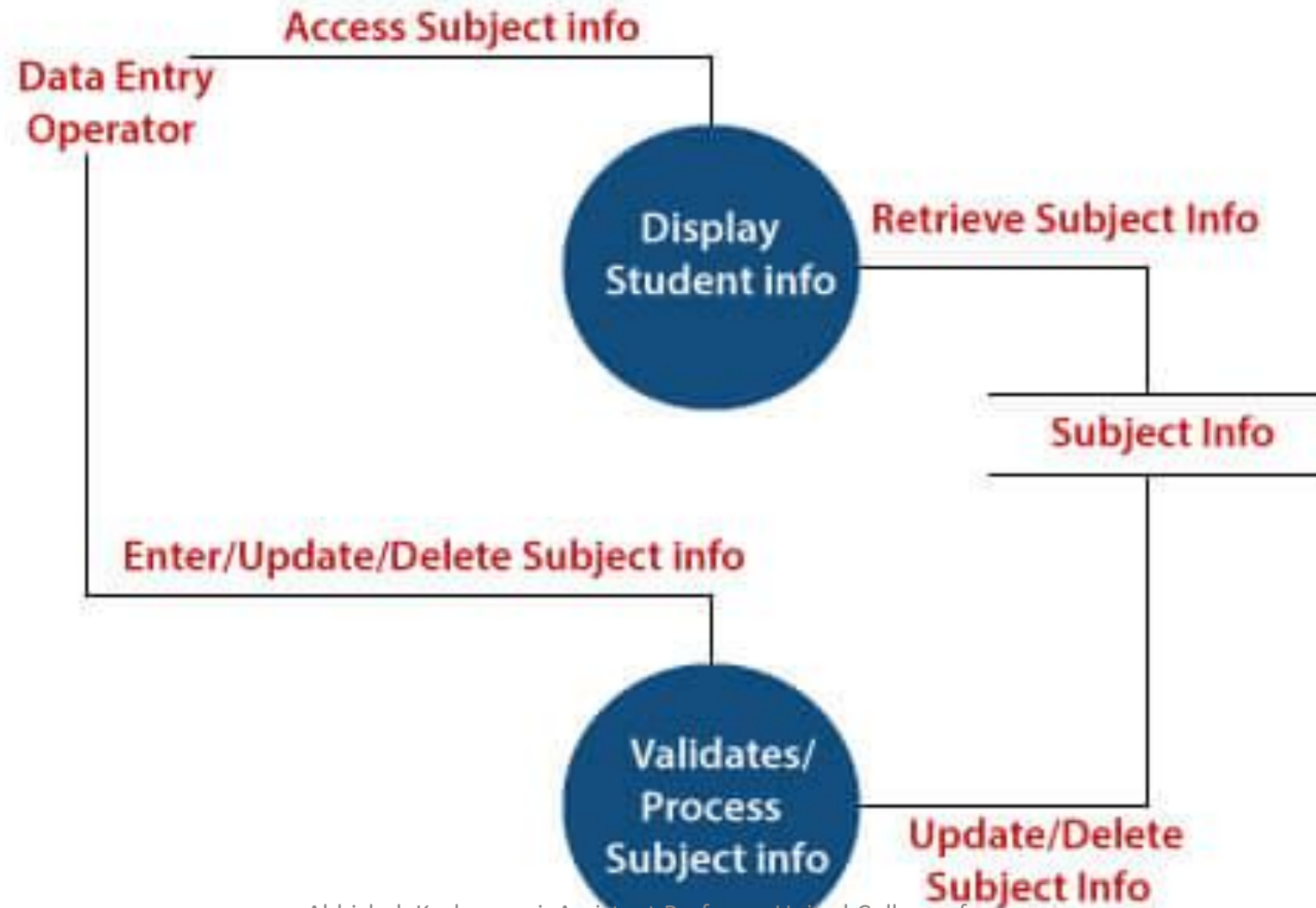


3. Student Information Management



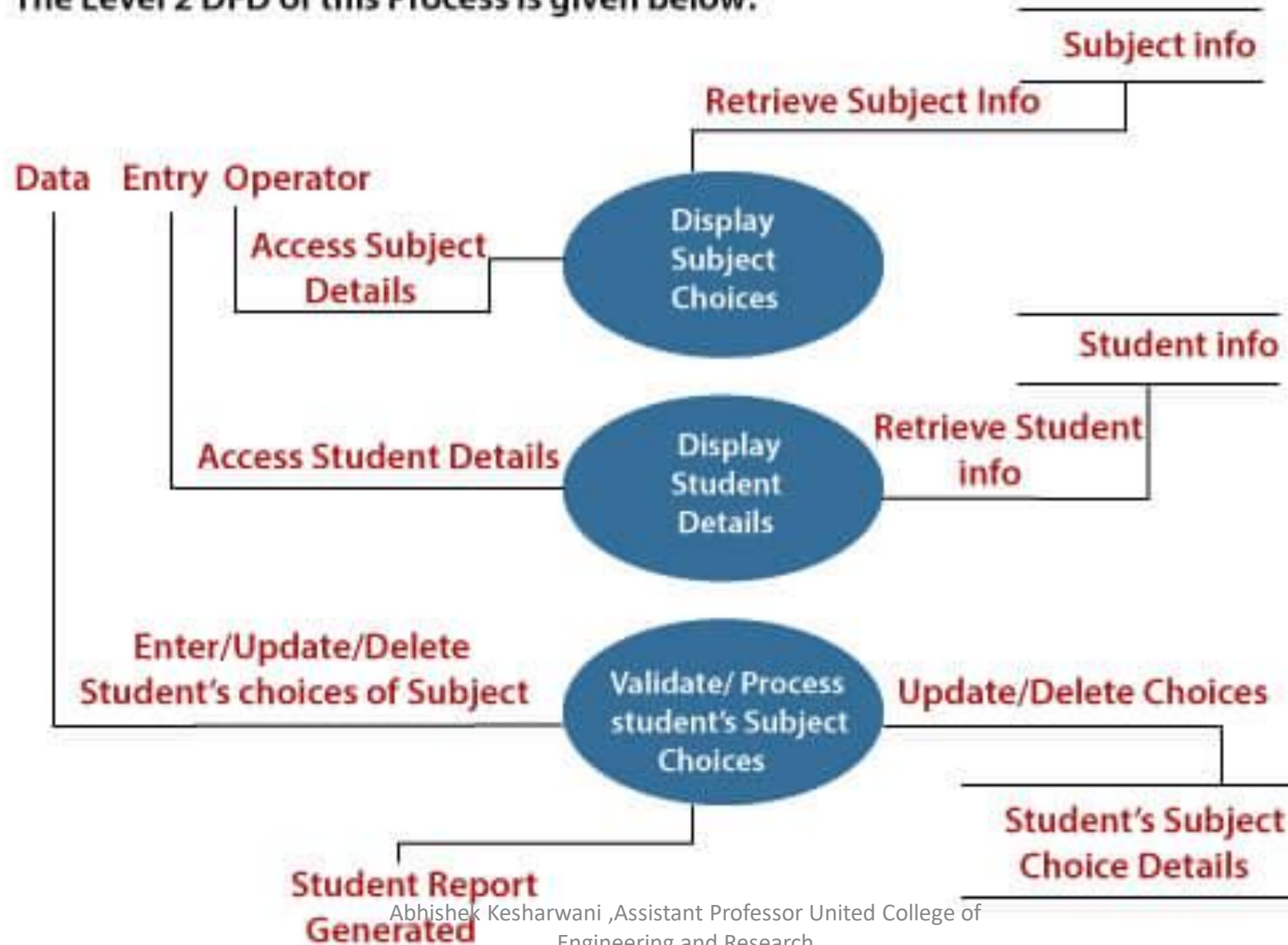
4. Subject Information Management

The level 2 DFD of this process is given below:



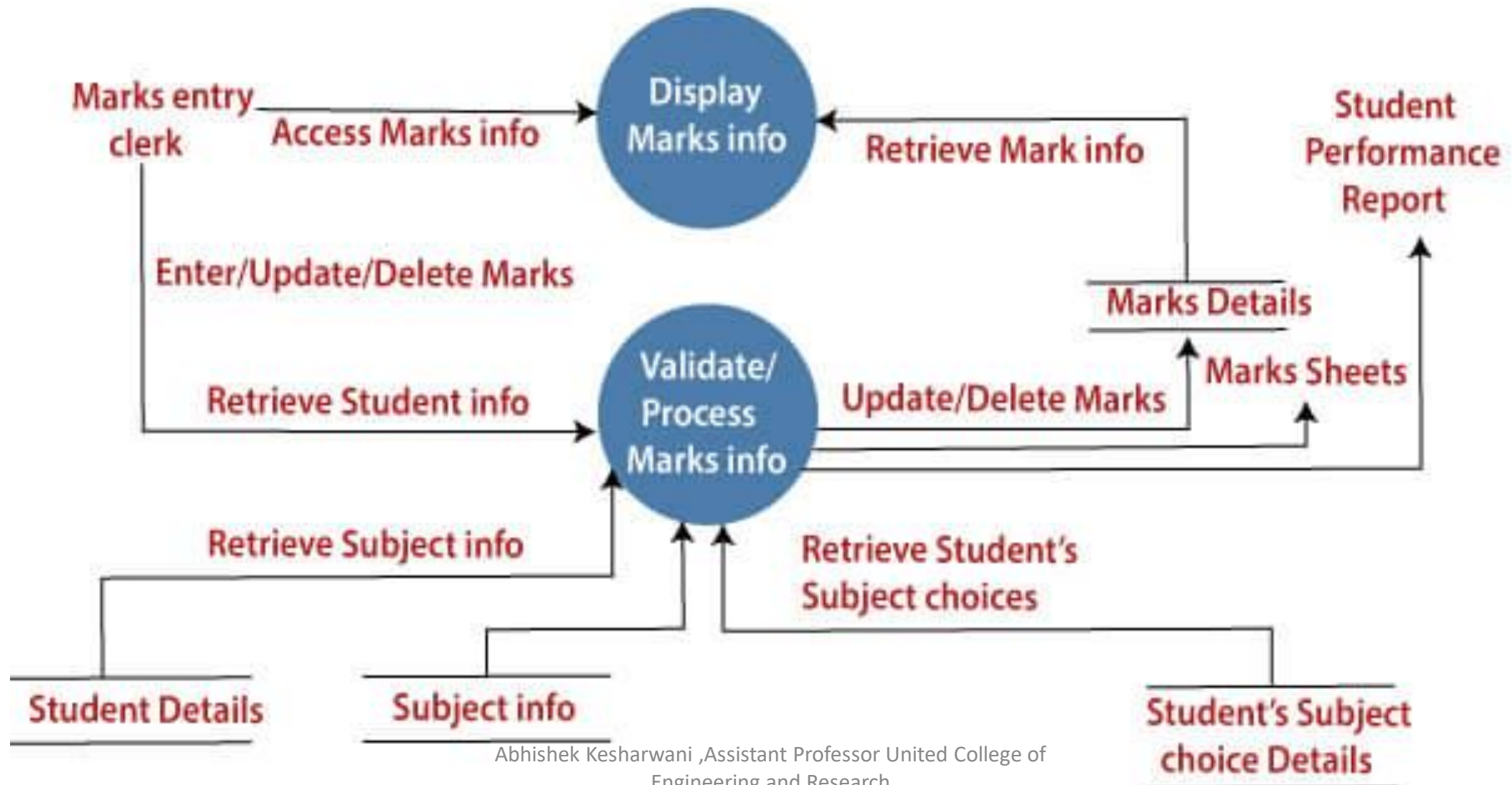
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Management

The Level 2 DFD of this Process is given below:



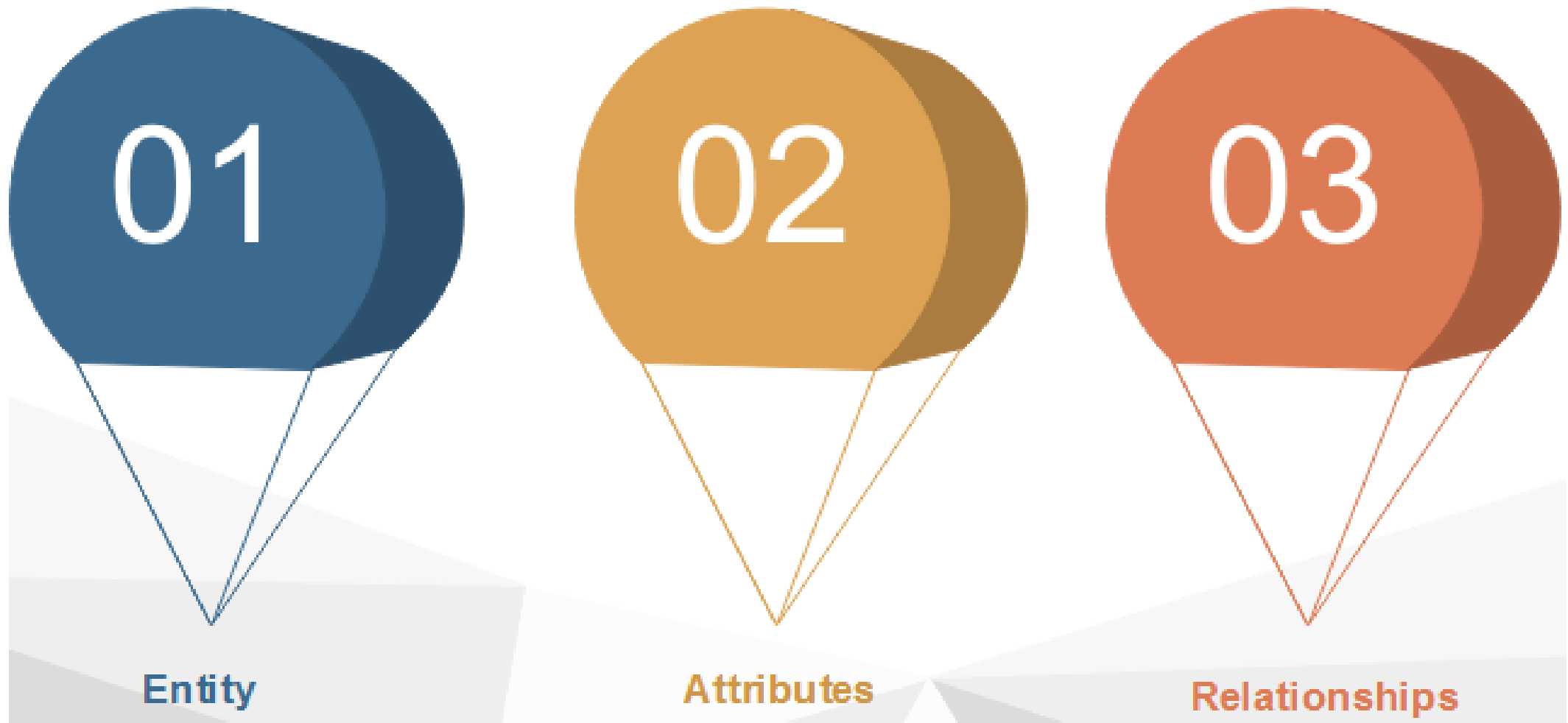
Entity-Relationship Diagrams

- 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 a ER Diagram



1. Entity

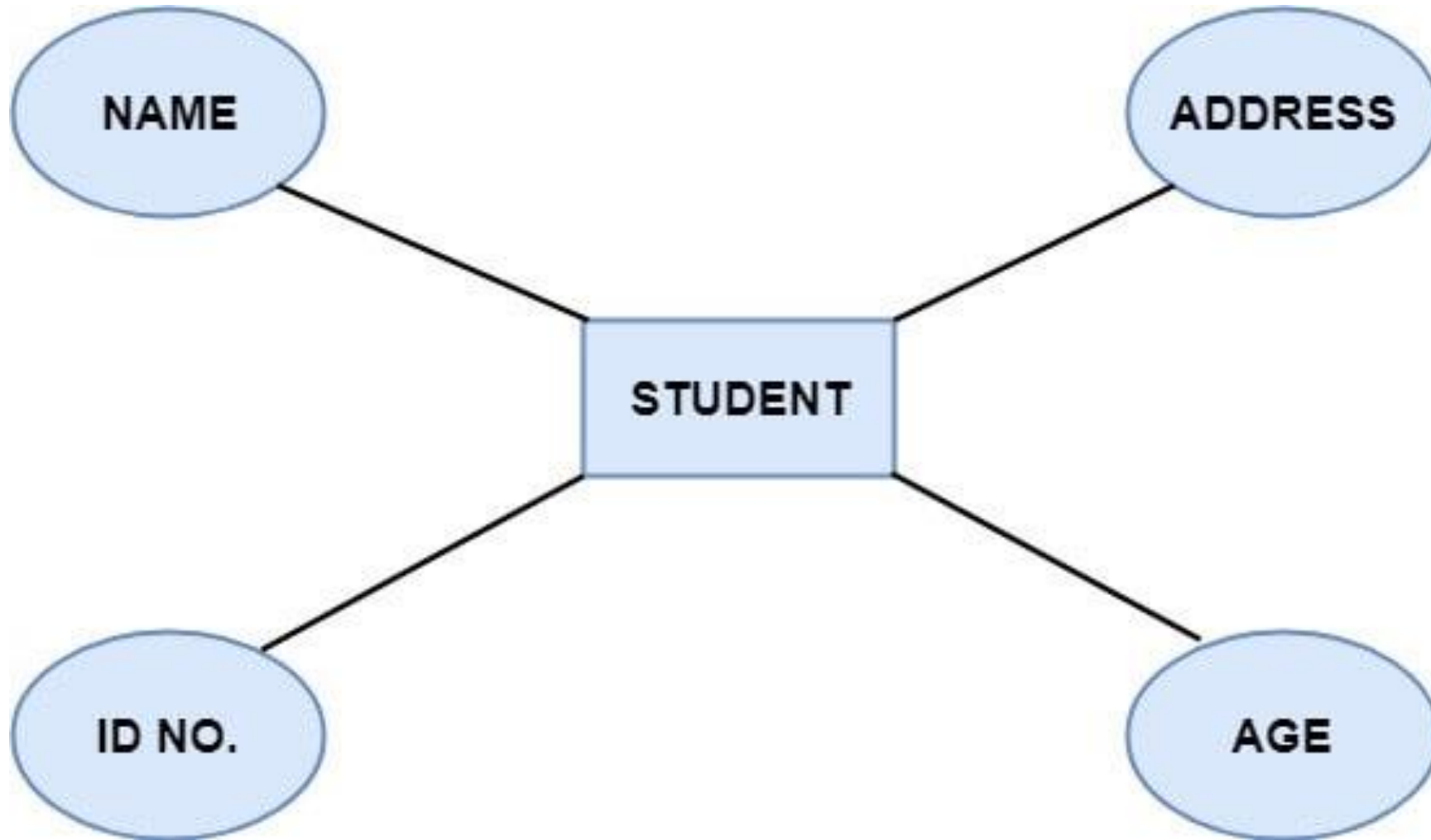
An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes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

2. Attributes

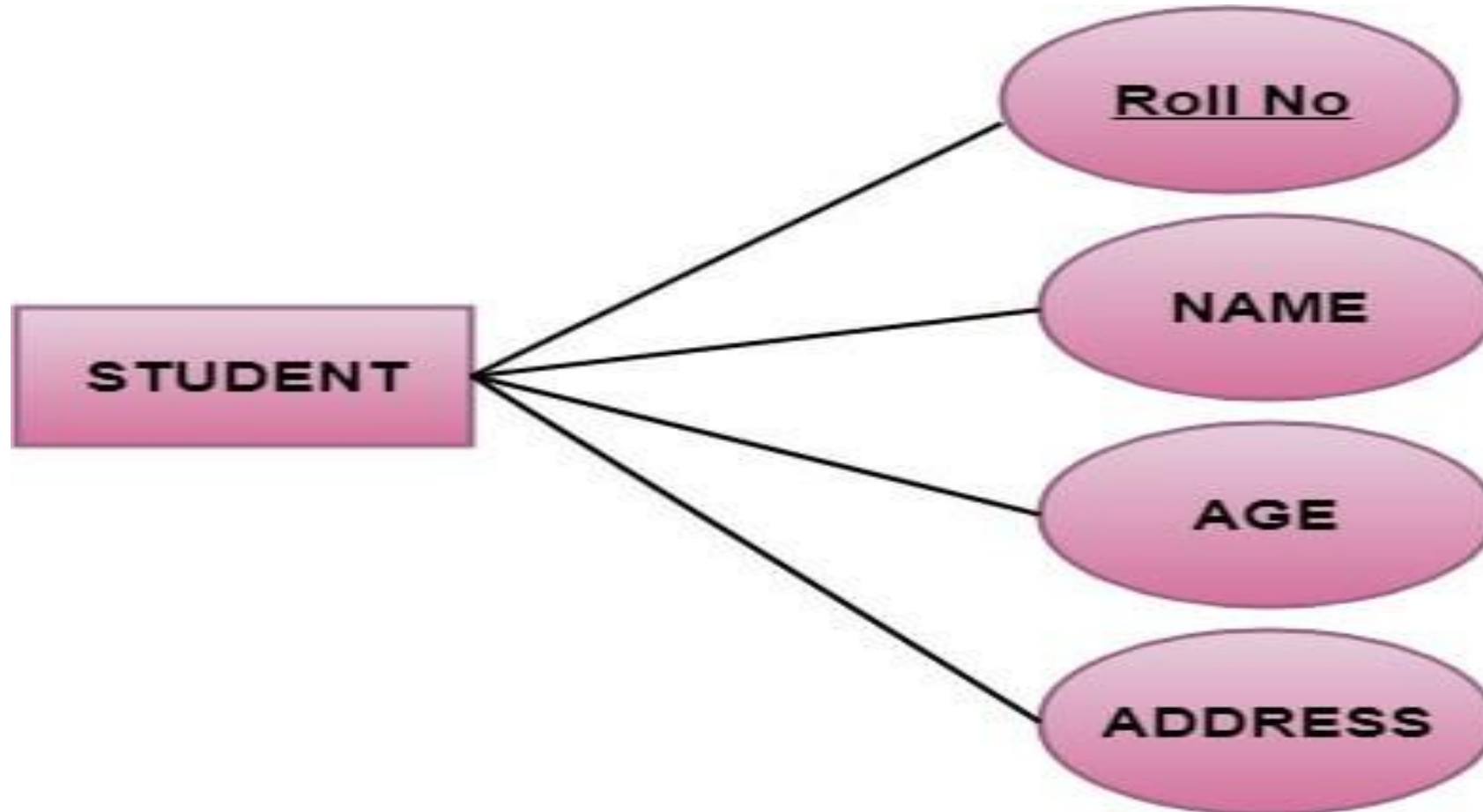
- Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.
- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

- Key attribute
- Composite attribute
- Single-valued attribute
- Multi-valued attribute
- 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.

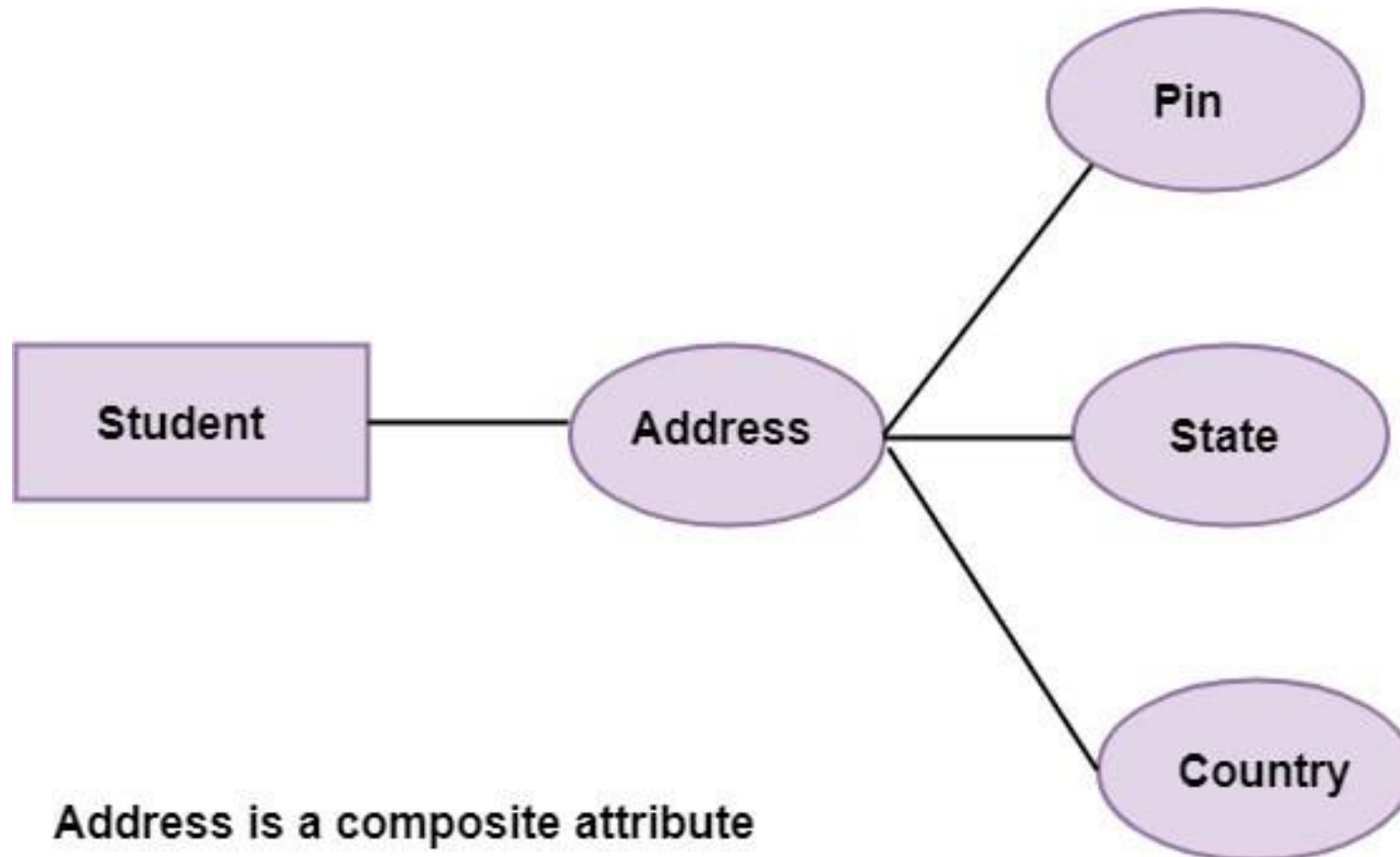


There are mainly three types of keys:

- **Super key:** A set of attributes that collectively identifies an entity in the entity set.
- **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.
- **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.

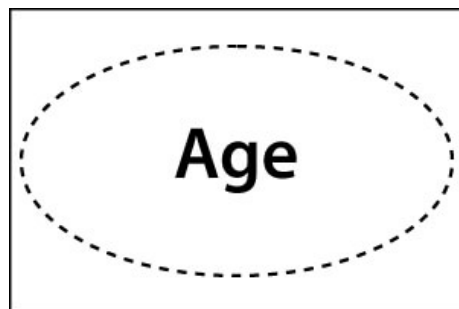
For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.



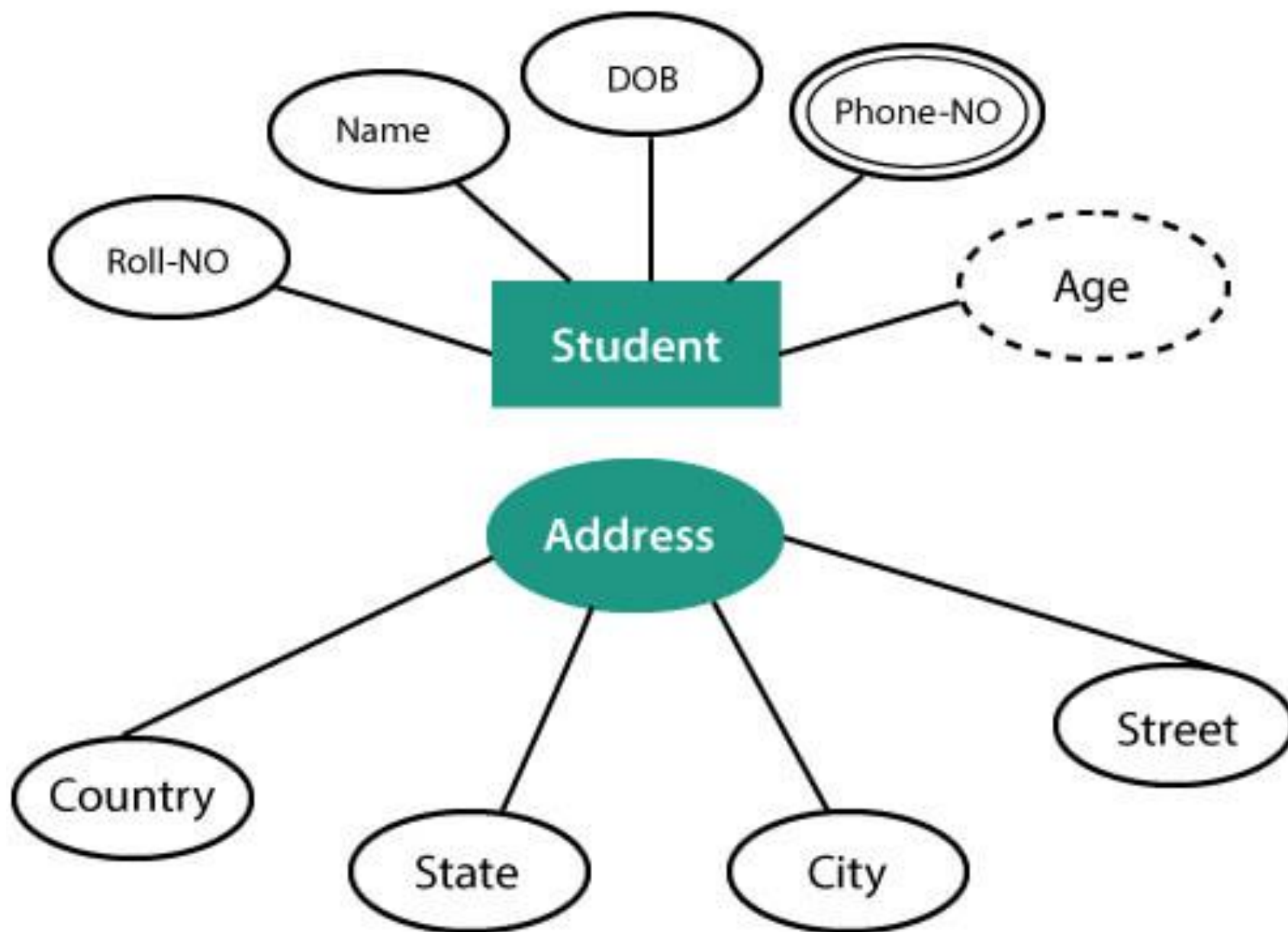
3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

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:

- Unary (degree1)
- Binary (degree2)
- 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.

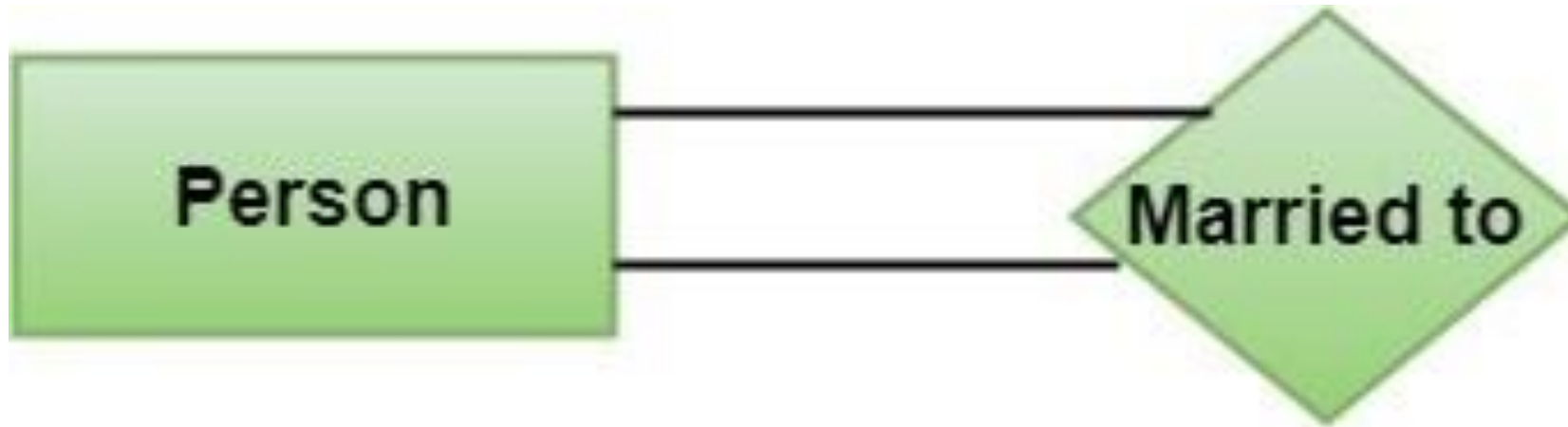


Fig: Unary Relationship

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.

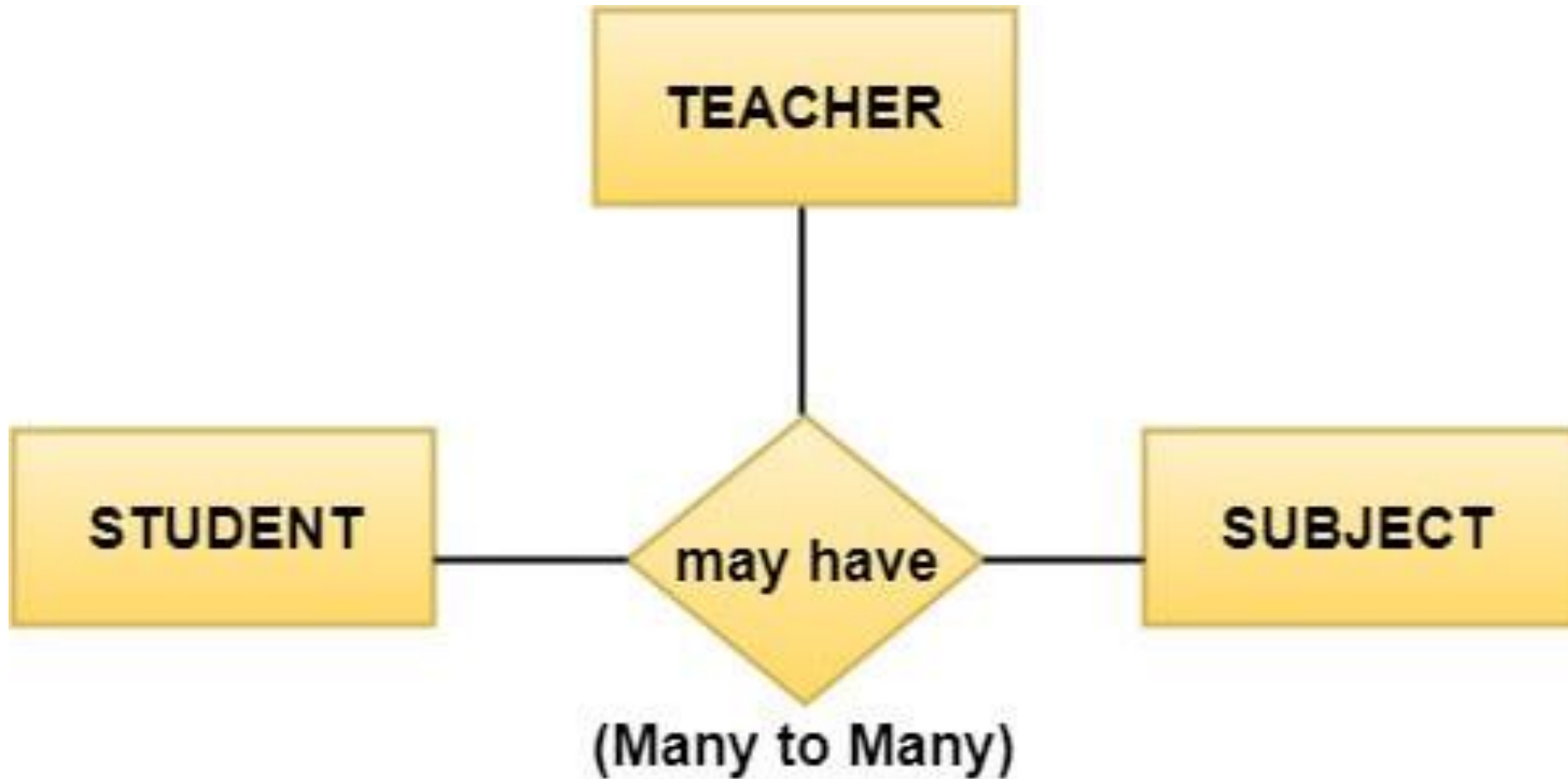


Fig: Ternary Relationship

Cardinality

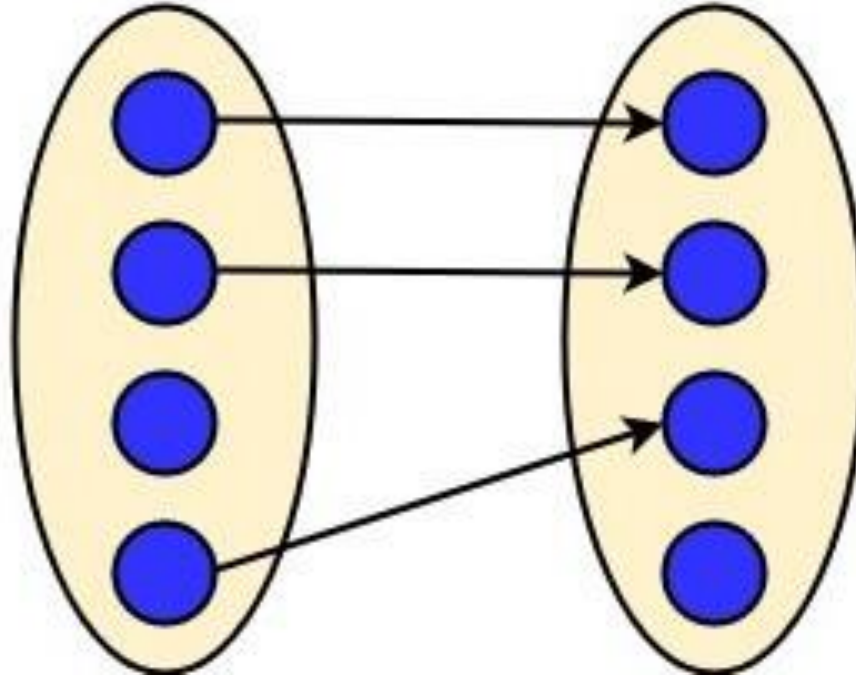
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

Types of Cardinalities

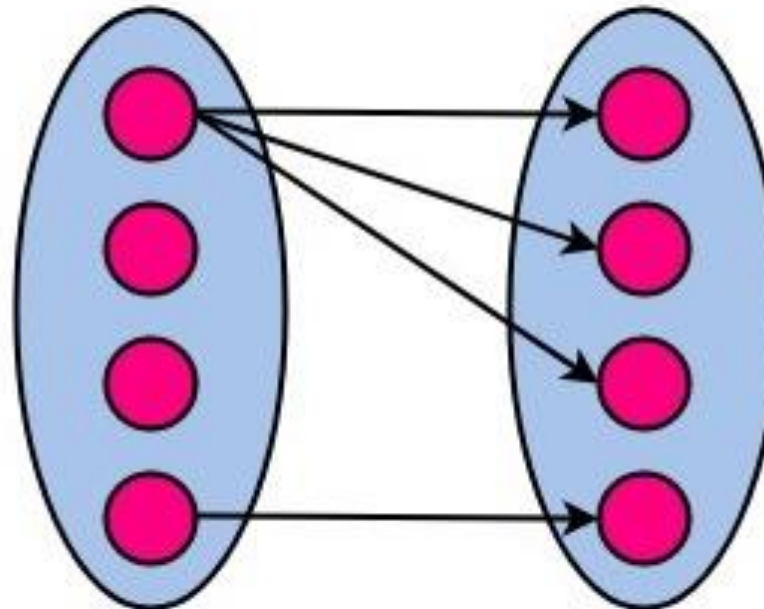
1. One to One: One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.



Using Sets, it can be represented as:

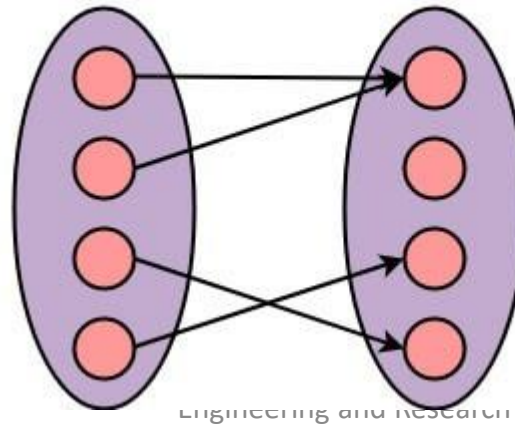


2. One to many: When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.

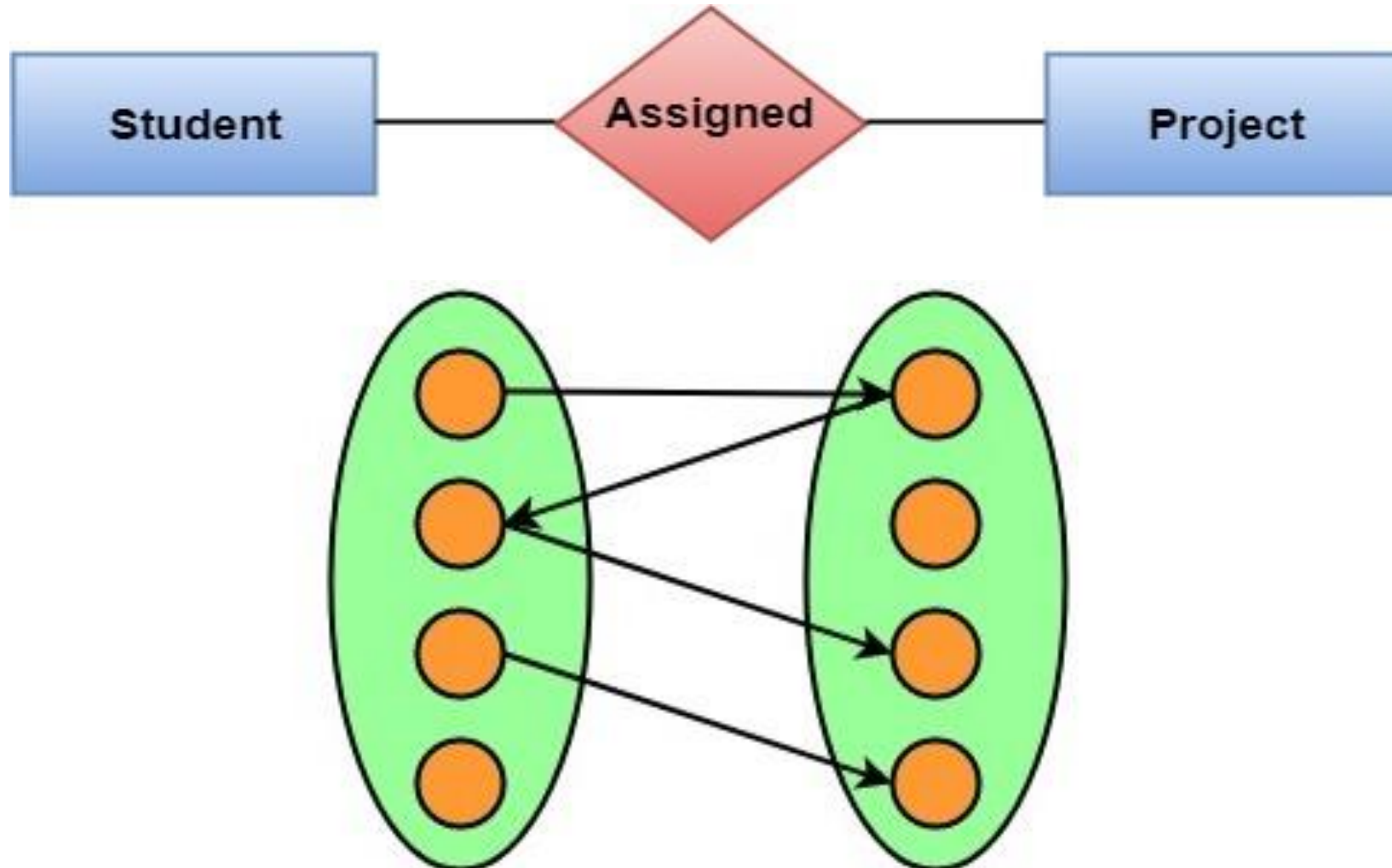


e of

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.



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.



Decision Table

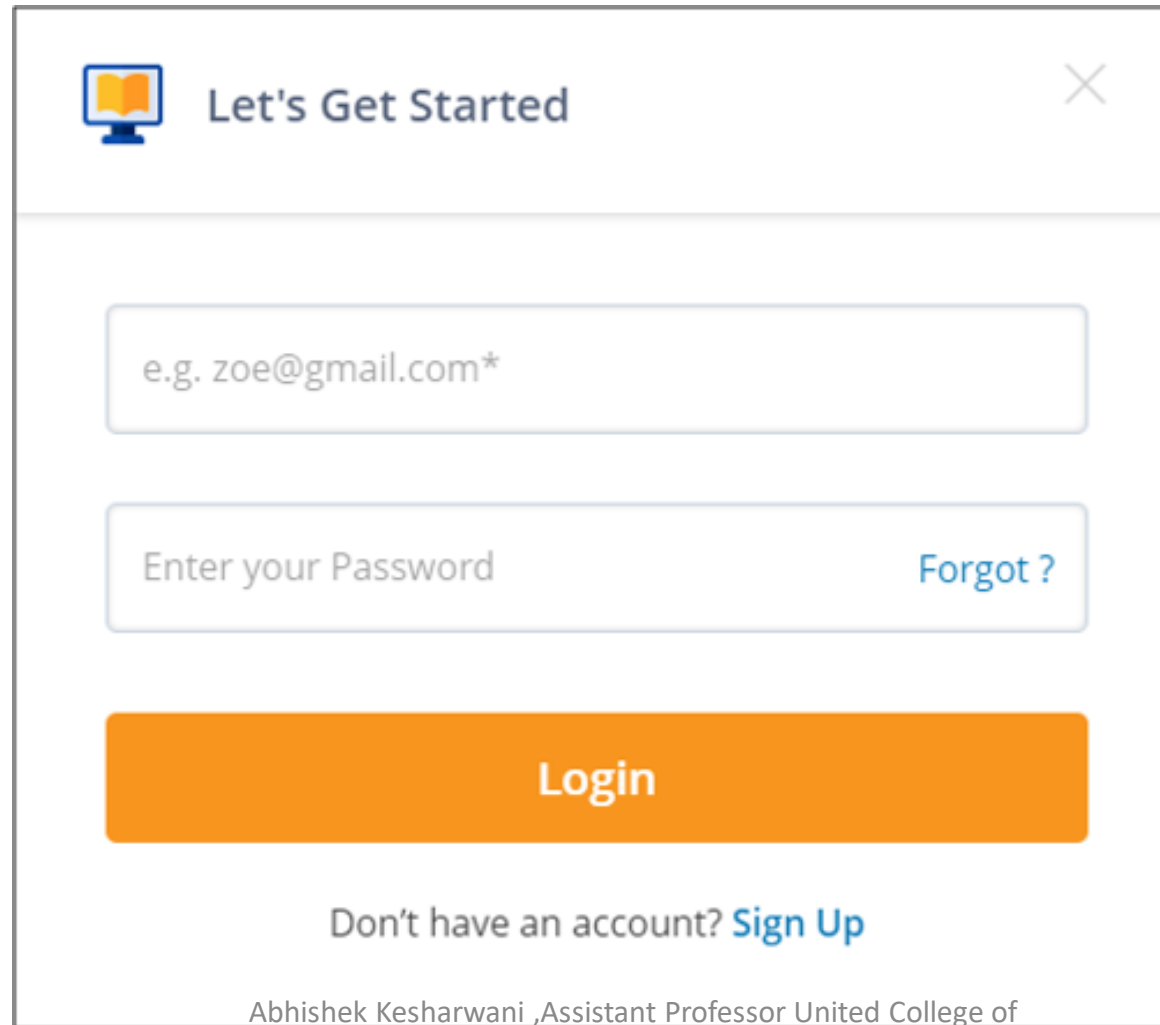
- A decision table is a good way to deal with different combination inputs with their associated outputs. It is a black box test design technique to determine the test scenarios for complex business logic.
- The decision table is a software testing technique which is used for testing the system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior are captured in a tabular form.

- This table helps you deal with different combination inputs with their associated outputs. Also, it is known as the cause-effect table because of an associated logical diagramming technique called cause-effect graphing that is basically used to derive the decision table.

Why is Decision Table Important?

- Decision tables are very much helpful in test design technique.
- It helps testers to search the effects of combinations of different inputs and other software states that implement business rules.
- It provides a regular way of stating complex business rules which benefits the developers as well as the testers.
- It assists in the development process with the developer to do a better job. Testing with all combination might be impractical.
- It the most preferable choice for testing and requirements management.
- it is a structured exercise to prepare requirements when dealing with complex business rules.

Let's take an example and see how to create a decision table for a login screen:



Let's Get Started

e.g. zoe@gmail.com*

Enter your Password [Forgot ?](#)

Login

Don't have an account? [Sign Up](#)

The condition states that if the user provides the correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username	F	T	F	T
Password	F	F	T	T
Output	E	E	E	H

In the above example,

T – Correct username/password

F – Wrong username/password

E – Error message is displayed

H – Home screen is displayed

Now let's understand the interpretation of the above cases:

Case 1 – Username and password both were wrong. The user is shown an error message.

Case 2 – Username was correct, but the password was wrong. The user is shown an error message.

Case 3 – Username was wrong, but the password was correct. The user is shown an error message.

Case 4 – Username and password both were correct, and the user is navigated to the homepage.

SRS Document

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

An SRS can be simply summarized into four Ds:

- **Define your product's purpose.**
- **Describe what you're building.**
- **Detail the requirements.**
- **Deliver it for approval.**

IEEE Standards for SRS

- IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces.'
- Each requirement is defined in such a way that its achievement can be objectively verified by a prescribed method, for example, inspection, demonstration, analysis or test.'
- Note that requirements specification can be in the form of a written document, a mathematical model, a collection of graphical models, a prototype, and so on.

Structure of SRS

- The requirements document is devised in a manner that is easier to write, review, and maintain.
- It is organized into independent sections and each section is organized into modules or units.
- Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development.
- **For example**, if a system is to be developed by an external contractor, then critical system specifications need to be precise and detailed. Similarly, when flexibility is required in the requirements and where an in-house development takes place, requirements documents can be less detailed.

Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner. For this, certain guidelines are followed while preparing SRS.

Guidelines are followed while preparing SRS

Functionality: It should be separate from implementation.

Analysis model: It should be developed according to the desired behavior of a system. This should include data and functional response of a system to various inputs given to it.

Cognitive model: It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.

The content and structure of the specification: It should be flexible enough to accommodate changes.

Specification: It should be robust. That is, it should be tolerant towards incompleteness and complexity.

SRS document comprises the following sections.

- **Introduction:** This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.
- **Overall description:** It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.
- **Product perspective:** It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.

- **Product functions:** It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:
- **User characteristics:** It determines general characteristics of the users.
- **Constraints:** It provides the general description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.
- **Assumption and dependency:** It provides a list of assumptions and factors that affect the requirements as stated in this document.
- **Apportioning of requirements:** It determines the requirements that can be delayed until release of future versions of the system.
- **Specific requirements:** These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the following subsections.

- **External interface:** It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.
- **Functions:** It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

- **Performance requirements:** It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and dynamic requirements. **Static requirements** (also known as **capacity requirements**) do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. **Dynamic requirements** determine the constraints on the execution of the behavior of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).
- **Logical database of requirements:** It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.

- **Design constraint:** It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.
- **Software system attributes:** It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.
- **Organizing Specific Requirements:** It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.

- **Change management process:** It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.
- **Document approvals:** These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.
- **Supporting information:** It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects.