Analysis

Unit 3

3.1 Determining System Requirements

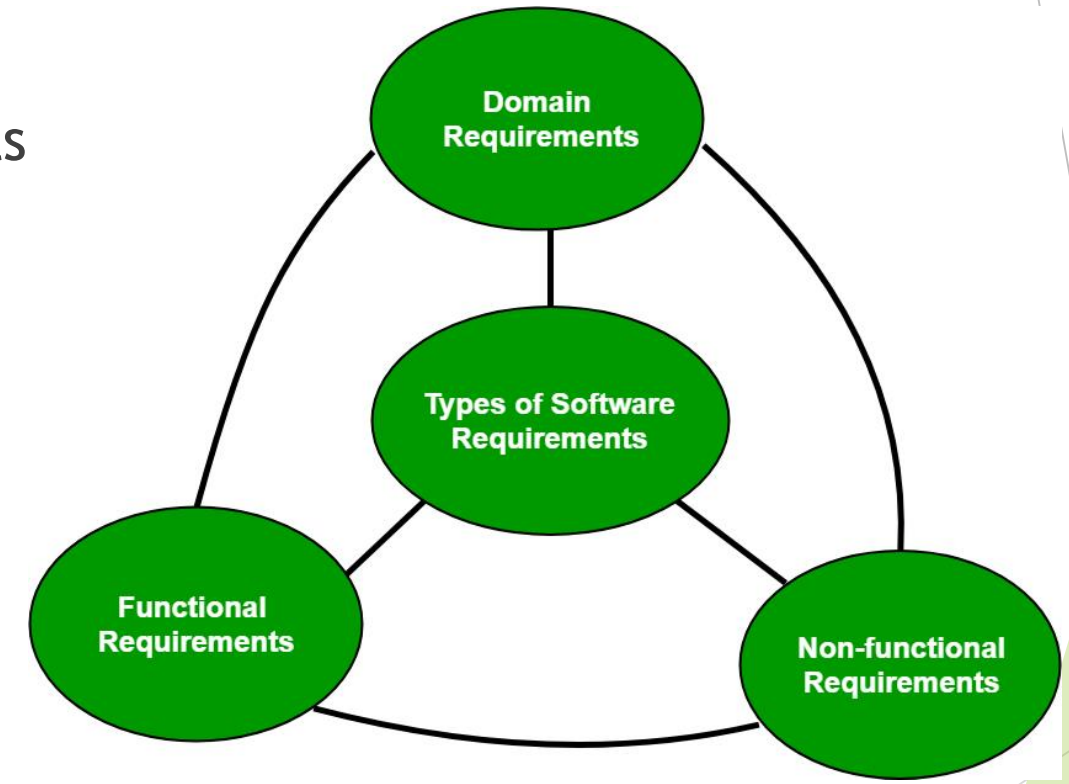
- Introduction
- Performing Requirements Determination (Process and Deliverables)
- Traditional Methods for Determining Requirements(Interviewing and Listening, Interviewing Groups, Directly Observing Users, Analyzing Procedures and other Documents)
- Contemporary Methods for Determining System Requirements (Joint Application Design, Prototyping)
- Radical Methods for Determining System Requirements (Business Process Reengineering, Identifying Process to Reengineer, Disruptive Technology)

Requirement analysis

- is the process of determining user expectations for a new or modified product
- must be quantifiable, relevant and detailed
- involves frequent communication with the stakeholders and end-users

Types of Requirements

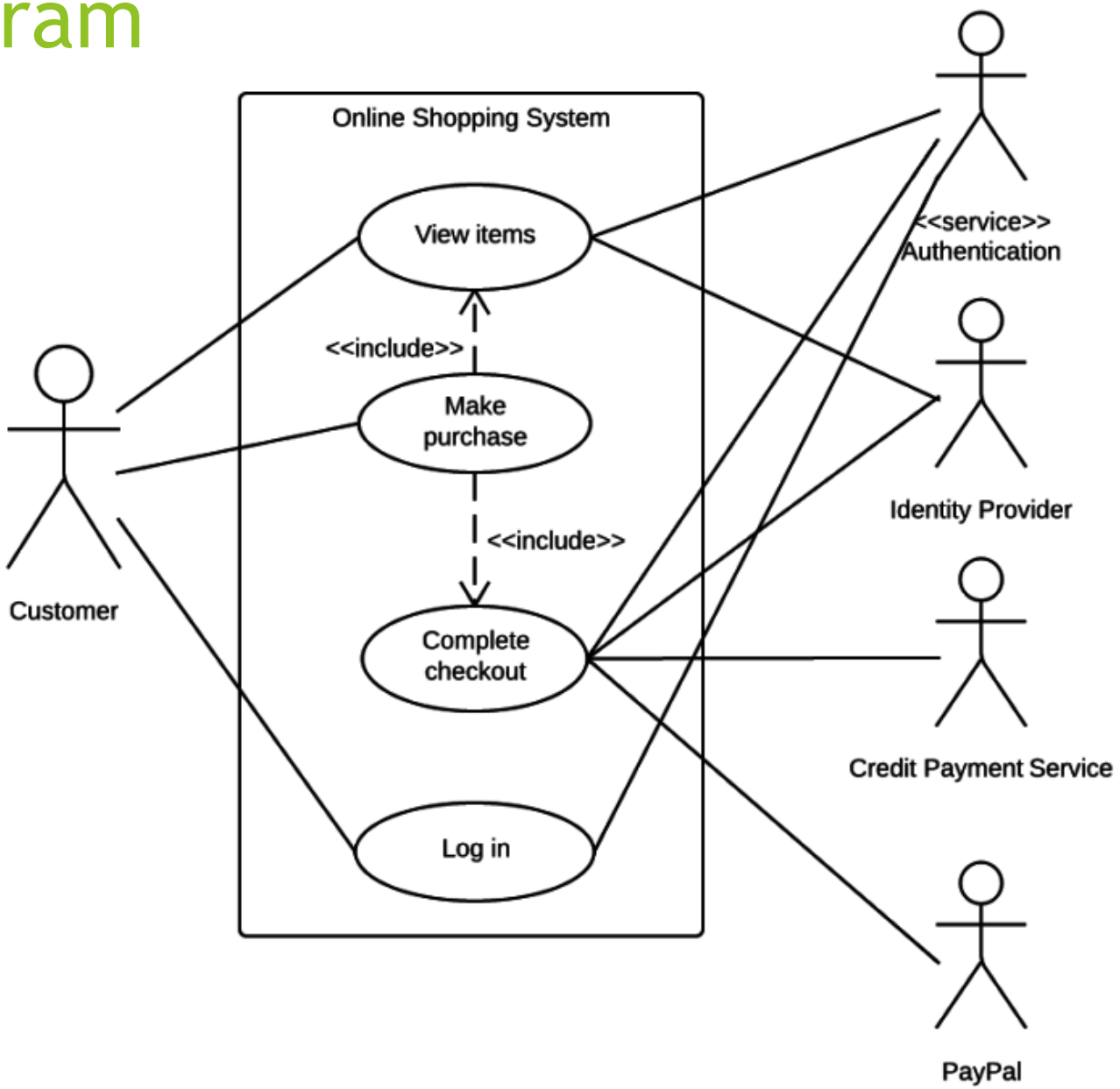
- Functional requirements
- Non-functional requirements
- Domain requirements



Functional Requirements

- are the requirements that the end user specifically demands as basic facilities that the system should offer.
- All these functionalities need to be necessarily incorporated into the system as a part of the contract.
- These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
- For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

Use case diagram



Non-functional requirements

- are basically the quality constraints that the system must satisfy according to the project contract.
- priority or extent to which these factors are implemented varies from one project to other.
- are also called non-behavioral requirements.
- They basically deal with issues like:
  - Portability
  - Security
  - Maintainability
  - Reliability
  - Scalability
  - Performance
  - Reusability
  - Flexibility

Non-functional requirements (contd..)

- The advantage of Non-functional requirement is that it helps you to ensure good user experience and ease of operating the software
- The biggest disadvantage of Non-functional requirement is that it may affect the various high-level software subsystems.
- **Examples of Non-functional requirements**
  - Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
  - Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
  - Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
  - A website should be capable enough to handle 20 million users with affecting its performance
  - The software should be portable. So moving from one OS to other OS does not create any problem.
  - Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited

Parameters	Functional Requirement	Non-Functional Requirement
What is it?	Verb	Attributes
Requirement	It is mandatory	It is non-mandatory
Capturing type	It is captured in use case.	It is captured as a quality attribute.
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties

Functional Requirements	Non functional requirements
• explain how the system must work • specified by User • It is mandatory. • It is captured in use case. • Defined at a component level. • Helps you verify the functionality of the software. • Example1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	• explain how the system should perform • specified by technical peoples e.g. Architect, Technical leaders and software developers • It is not mandatory. • It is captured as a quality attribute. • Applied to a system as a whole. • Helps you to verify the performance of the software. • Example1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

Domain requirements

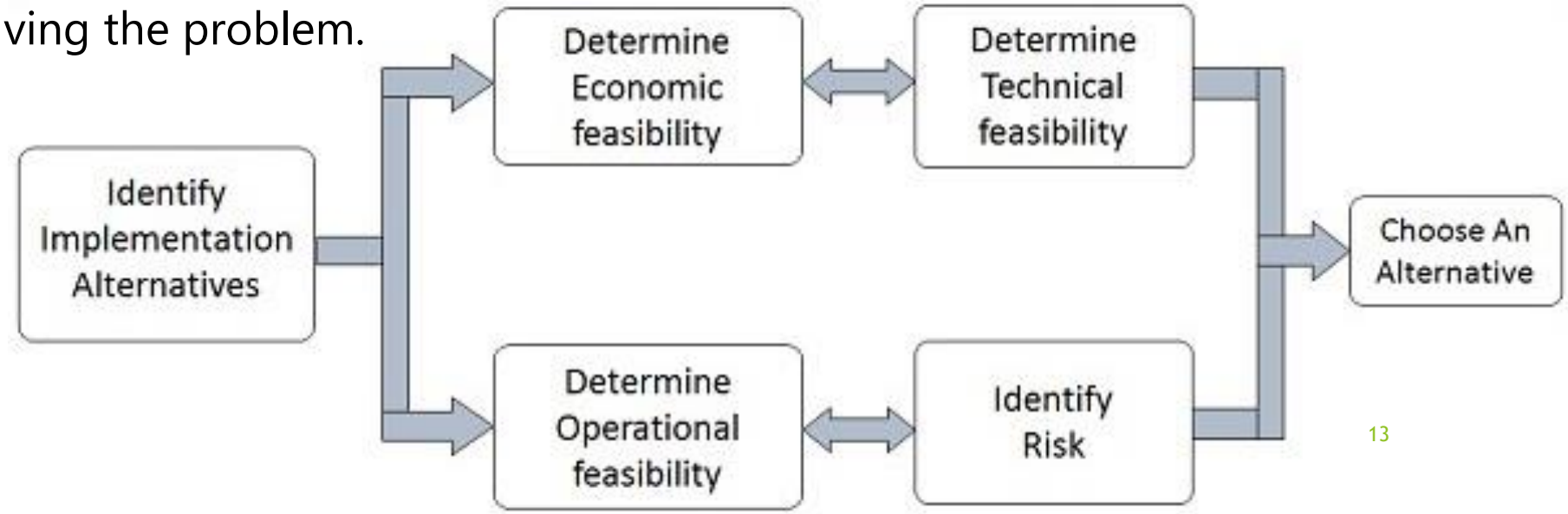
- are the requirements which are characteristic of a particular category or domain of projects.
- The basic functions that a system of a specific domain must necessarily exhibit come under this category.
- For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

Some other types

- **User requirement:** Statements in natural language plus diagrams of the system provides and its operational constraints. Written for customers
- **System requirements:** A structured document setting out detailed descriptions of the system services. Written a contract between client and contractor
- **Software specification:** A detailed software description which can serve as a basis for design or implementation. Written for developers

Feasibility study

- can be considered as preliminary investigation
- Is a study to reveal whether a project is feasible (possible to do) or not.
- It is conducted in order to find answer to following question
  - Do we have required resources and technologies to build the project
  - Do we receive profit from the project
- It tells us whether a project is worth the investment
- The main objective of a feasibility study is to acquire problem scope instead of solving the problem.



Types of Feasibility:

- **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
- **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
- **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.
- **Scheduling Feasibility-** we estimate the time necessary to complete the project, also consider the organization's a capabilities and determine whether that amount of time is available or not
- **Legal Feasibility**

Importance of feasibility

- Get a clear-cut idea of whether the project is likely to be successful, before allocating budget, manpower and time.
- Improves project teams focus
- Provides valuable information for a decision whether to accept or not
- Identifies a valid reason to undertake the project
- Enhances the success rate by evaluating multiple parameters

What is Requirements Determination?

- A requirement is a vital feature of a new system which may include processing or capturing of data, controlling the activities of business, producing information and supporting the management
- involves studying the existing system and gathering details to find out what are the requirements, how it works, and where improvements should be made
- to understand current problems and opportunities, as well as what is needed and desired in future systems

Major Activities in requirement Determination

- **Requirements Anticipation :** It predicts the characteristics of system based on previous experience which include certain problems or features and requirements for a new system.
- **Requirements Investigation :** It is studying the current system and documenting its features for further analysis. It is at the heart of system analysis where analyst documenting and describing system features using fact-finding techniques, prototyping, and computer assisted tools.
- **Requirements Specifications :** It includes the analysis of data which determine the requirement specification, description of features for new system, and specifying what information requirements will be provided.

The Process of Determining Requirements

- During requirements determination, you and other analysts gather information on what the system should do from as many sources as possible: from developers, administrators and users of the current system, from observing users, from reports, forms and procedures.
- Several characteristics of a good system analyst :
  1. **Impertinence:** You should question everything.
  2. **Impartiality :** Your role is to find the best solution to a business problem or opportunity.
  3. **Relax constrains :** Assume anything is possible and eliminate the infeasible.
  4. **Attention to details :** Every fact MUST fit with every other fact.
  5. **Reframing :** You MUST challenge your self to look at the organization in different ways.

Process and Deliverables

- **Process:** a set of elements arranged in an orderly manner to accomplish an objective
- **Deliverables:**
  - is a final deadline or project milestone that can be provided to external or internal customers. It is the end result or one of many end results in a project plan that can be quantifiable
  - is any product, service, or result that must be completed to finish a project.
  - Examples of deliverables include an initial project strategy report, the budget report, a progress report, a beta product, a test result report, and any other quantifiable aspects of a project that mark a completion.

Types of Deliverables

- **Tangible vs. Intangible Deliverables:**
  - An example of a tangible deliverable would be the construction of a new office to place new workers that don't fit in the old office or a new manufacturing plant that needs to be built to meet increased production levels.
  - An example of an intangible deliverable would be a training program for employees to teach them how to use new software that will be used at the company.
- **Internal Deliverables vs. External Deliverables:**
  - Internal deliverables are those deliverables that are in-house and required to complete a project, deliver a good, or provide a service. Internal deliverables are not seen by the customer and are not considered final.
  - External deliverables, on the other hand, are final and provided to the customer. In the example above, the external deliverable would be the final good that comes out of the new factory that the customer will purchase and use. In project management, external deliverables are commonly known as product deliverables.

Traditional Methods for Determining Requirements

- Interviewing and Listening
- Interviewing Groups
- Questionnaires
- Directly Observing Users
- Analyzing Procedures and other Documents

Contemporary Methods for Determining System Requirements

- **Joint Application Design:** Brings together key users, managers, and systems analysts
  - Purpose: collect system requirements simultaneously from key people
  - Conducted off-site
  - Group Support Systems: Facilitate sharing of ideas and voicing of opinions about system requirements
- **Prototyping:** Iterative development process
  - Rudimentary working version of system is built
  - Refine understanding of system requirements in concrete terms
- CASE tools: Used to analyze existing systems
  - Help discover requirements to meet changing business conditions

Joint Application Design (JAD)

- Intensive group-oriented requirements determination technique
- is to bring together the key users, managers, and system analysts involved in the analysis of a current system
- is to collect systems requirements simultaneously from the key people involved with the system
- Team members meet in isolation for an extended period of time
- Highly focused
- Resource intensive
- Started by IBM in 1970s
- End Result
  - Documentation detailing existing system
  - Features of proposed system

JAD Participants

- Session Leader: facilitates group process
- Users: active, speaking participants
- Managers: active, speaking participants
- Sponsor: high-level champion, limited participation
- Systems Analysts: should mostly listen
- Scribe: record session activities
- IS Staff: should mostly listen

CASE Tools During JAD

- For requirements determination and structuring, the most useful CASE tools are for diagramming and form and report generation
- analyst can use diagramming and prototyping tools to give graphic form to system requirements, show the tools to users, and make changes based on the users' reactions.
- The same tools are very valuable for requirements structuring as well.
- Using common CASE tools during requirements determination and structuring makes the transition between these two subphases easier and reduces the total time spent

Prototyping

- is a means of exploring ideas before you invest in them
- allow system analysts quickly show users the basic requirement into a working version of the desired information system
- After viewing and testing the prototype, the users usually adjust existing requirements to new ones
- The goal with using prototyping to support requirement determination is to develop concrete specification for the ultimate system, not to build the ultimate system from prototyping
- is most useful for requirements determination when user requirements are not clear or well understood
- one or a few users and other stakeholders are involved with the system, possible designs are complex and require concrete form to fully evaluate

Using Prototyping During Requirements Determination

- Quickly converts requirements to working version of system
- Once the user sees requirements converted to system, will ask for modifications or will generate additional requests
- Most useful when:
  - User requests are not clear.
  - Few users are involved in the system.
  - Designs are complex and require concrete form.
  - There is a history of communication problems between analysts and users
  - Tools are readily available to build prototype.

Characteristic	Interviews	Questionnaires	Observation	Document analysis	JAD	Prototyping
Information Richness	High	Medium to low	High	Low (passive) and old	High	Medium to High
Time Required	Can be extensive	Low to moderate	Can be extensive	Low to moderate	Dedicated period of time of all kinds of involved people	Moderate and can be extensive
Expense	Can be high	Moderate	Can be high	Low to moderate	High	High
Chance for Follow-up and probing	Good	Limited	Good	Limited	Good	Good
Confidentiality	Interviewee is known to interviewer	Respondent can be unknown	Observee is known to interviewer	Depends on nature of document	All the people know each other	Usually know each other
Involvement of Subject	Interviewee is involved and committed	Respondent is passive, no clear commitment	Interviewees may or may not be involved and committed depending on whether they know if they are being observed	None, no clear commitment	All kinds of people are involved and committed	Users are involved and committed
Potential Audience	Limited numbers, but complete responses from those interviewed	Can be quite large, but lack of response from some can bias results	Limited numbers and limited time of each	Potentially biased by which documents were kept or because document not created for this purpose	Potentially biased by the subordinator intentionally don't want to directly point out his superior's errors.	Limited numbers; it is difficult to diffuse or adapt to other potential users

Radical Methods for Determining System Requirements

- **Business Process Reengineering(BPR):** search for and implementation of radical change in business processes to achieve breakthrough improvements in products and services
  - Fewer people may be needed to do the same work
  - relationships with customers may improve dramatically
  - processes may become much more efficient and effective
  - all of which can result in increased profits.

  The overall process by which current methods are replaced with radically new methods is referred to as **business process reengineering (BPR)**

Five steps of business process reengineering (BPR)

1. Map the current state of your business process
2. Analyze them and find any process gaps or disconnects
3. Look for improvement opportunities and validate them
4. Design a cutting-edge future-state process map
5. Implement future state changes and be mindful of dependencies

- **Identifying Processes to Reengineer:**
  - A first step in any BPR effort is to understand what processes need to change, what are the **key business processes** for the organization.
  - Key business processes are the structured set of measurable activities designed to produce a specific output for a particular customer or market.
  - The important aspect of this definition is that key processes are focused on some type of organizational outcome such as the creation of a product or the delivery of a service.
  - Key business processes are also customer focused.
  - In other words, key business processes would include all activities used to design, build, deliver, support, and service a particular product for a particular customer.

- **Disruptive Technology:**
  - Once key business processes and activities have been identified, information technologies must be applied to improve business processes radically
  - are those that enable the breaking of long-held business rules that inhibit organizations from making radical business changes

3.2 Structuring System Process Requirements

- Introduction
- Process Modeling (Modeling a System's Process for Structured Analysis, Deliverables and Outcomes)
- Data Flow Diagrams(Context Diagram and DFD, Data Flow Diagramming Rules, Decomposition and Balancing DFDs)
- Modeling Logic with Decision Tables, Decision Trees, and Pseudocodes

System Analyst

- is an IT professional who specializes in analyzing, designing and implementing information systems.
- carry the responsibilities of researching problems, finding solutions, recommending courses of actions and coordinating with stakeholders in order to meet specified requirements
- study the current system, procedures and business processes of a company and create action plans based on the requirements set
- can be involved starting from the analysis phase of the project until the post deployment assessment review

What does a Systems Analyst do?

- Communicate with customers and stakeholders to learn and document requirements in order to create a technical specification
- Interact and coordinate with developers and implementers
- document technology systems in order to understand, change, improve, and help rebuild these systems
- Help perform system testing
- Deploy the system
- Help with technical documentation like manuals
- Deliberate over post-project assessment

System Analyst Roles

- They act as liaisons between business users on one hand and technical personnel on the other.
- **Defining user requirements**
- **Prioritizing requirements**
- **Problem analysis**
- **Drawing specifications**
- **System design and evaluation**
- **Keeping up to date with technological advancements**

Skills of a systems analyst

- Knowledge of organization
- Knowledge of computer hardware and software
- Good understanding Communication and teaching abilities
- Problem solving and Critical thinking
- Analytical skills
- Motivator skill
- Management

Process Modeling

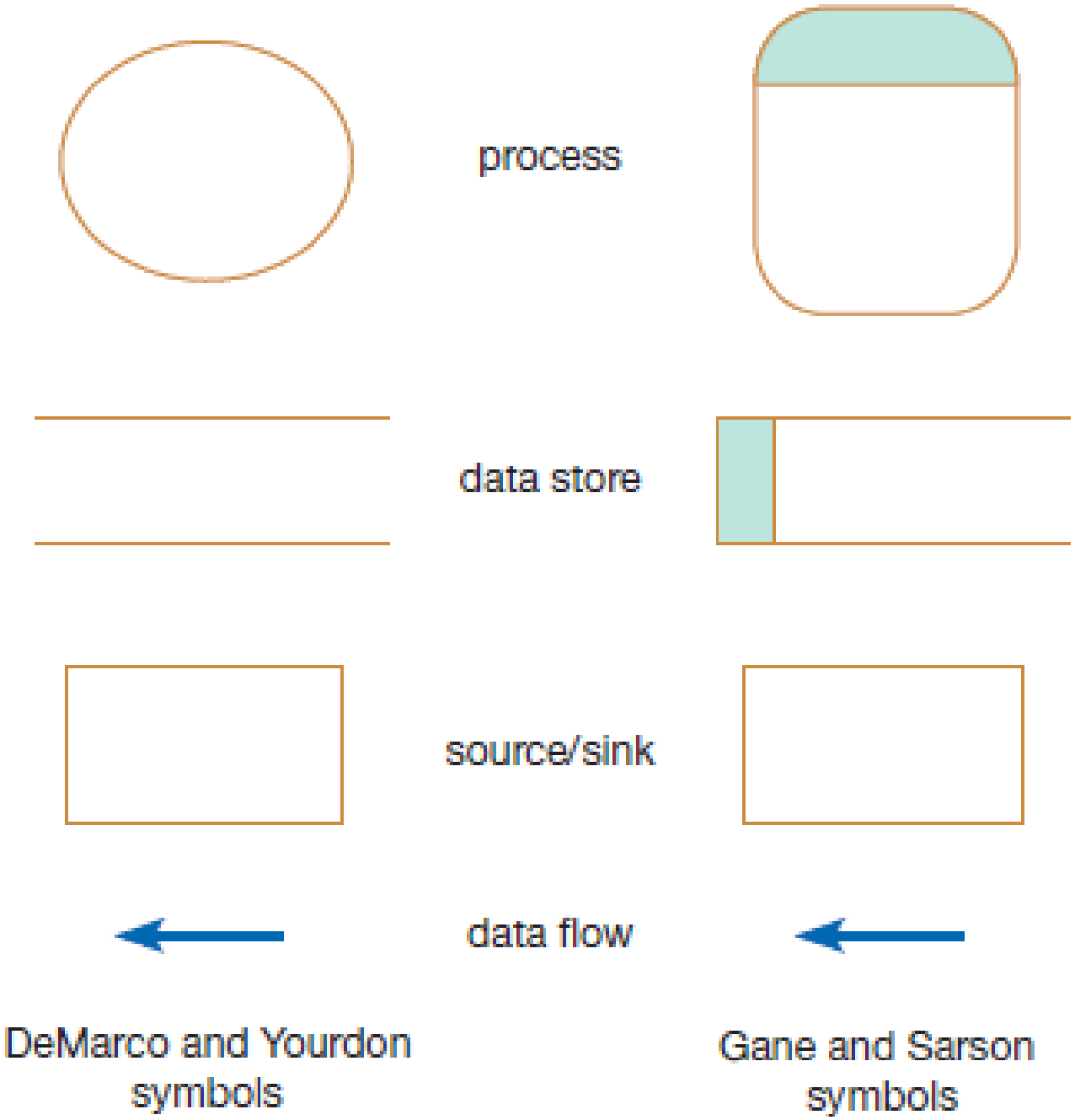
- involves graphically representing the functions, or processes, that capture, manipulate, store, and distribute data between a system and its environment and between components within a system
- Common form of a process model is a **data flow diagram (DFD)**

Deliverables and outcomes

- In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated DFDs
- DFDs are used to study and document a system's processes.
- First, a context diagram shows the scope of the system, indicating which elements are inside and which are outside the system.
- Second, DFDs of the system specify which processes move and transform data, accepting inputs and producing outputs.
- These diagrams are developed with sufficient detail to understand the current system

Data Flow Diagram (DFD)

- is a tool that depicts the flow of data through a system and the work or processing performed by that system.
- It is also called bubble chart, transformation graph, or process model.
- There are two different sets of data flow diagram symbols, but each set consists of four symbols that represent the same things: data flows, data stores, processes, and sources/sinks (or external entities).


process
data store
source/sink
data flow
DeMarco and Yourdon symbols
Gane and Sarson symbols

Contd..

- **Process** is the work or actions performed on data so that they are transformed, stored or distributed.
- **Data store** is the data at rest (inside the system) that may take the form of many different physical representations.
- **External entity (source/sink)** is the origin and/or destination of data.
- **Data flow** represents data in motion, moving from one place in a system to another

Developing DFD

- The highest-level view of a system is called context diagram or context DFD.
- A context DFD is used to document the scope
- the scope of any project is always subject to change, the context diagram is also subject to constant change.
- The context DFD contains one and only one process and it has no data storage.
- Sometimes this process is identified by the number “0”. The figure below shows the context diagram of library system.

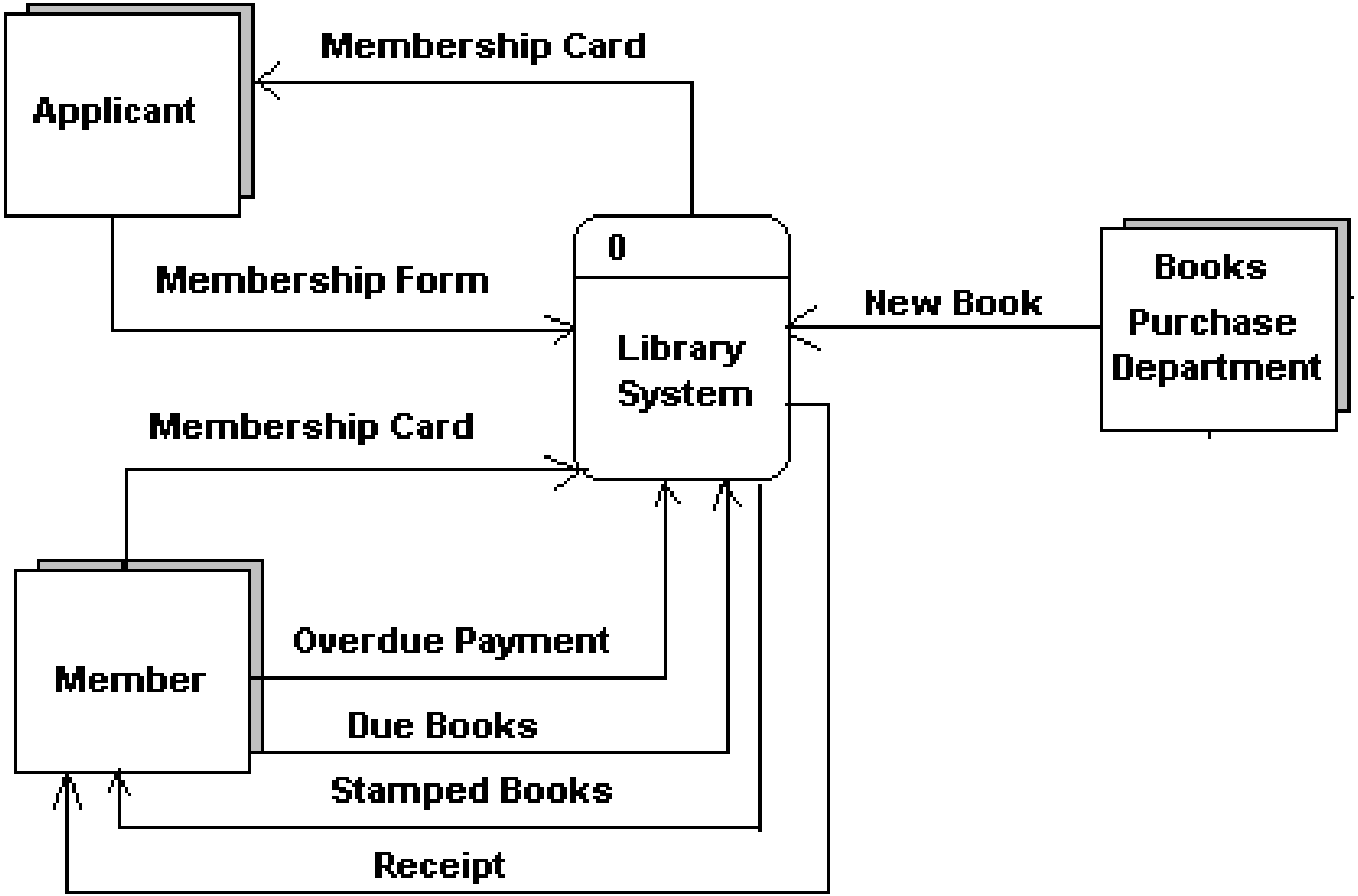


Fig: Context diagram of library system

Contd..

- The next step is to draw a DFD with major processes that are represented by the single process in the context diagram
- These major processes represent the major functions of the system.
- This diagram also includes data stores and called **level-0 diagram** (or **level-0 DFD**).
- A level-0 diagram is a DFD that represents a system's major processes, data flows, and data stores at a high level of detail.
- In this diagram, sources/sinks should be same as in the context diagram. Each process has a number that ends in .0. The figure below shows level-0 DFD.

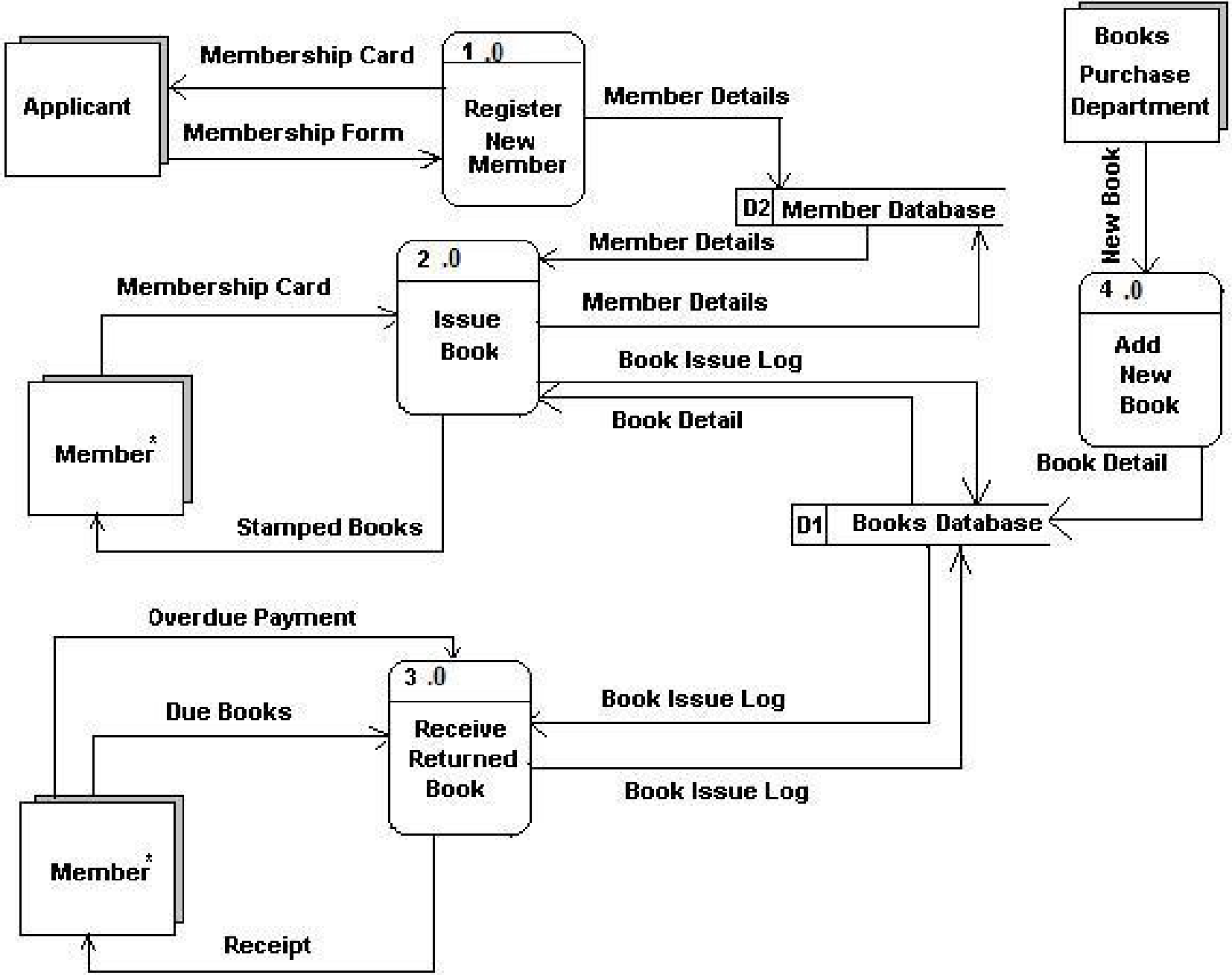


Fig: Level-1 DFD

Decision tables

- is a diagram of process logic where the logic is reasonably complicated.
- One of the ways to describe the *logic and timing* of what goes on inside the process boxes in data flow diagrams
- allow you to represent the conditional logic that is part of some data flow diagram processes
- shows conditions and actions in a simplified and orderly manner
- displays all conditions affecting a particular situation and the appropriate action or actions to be taken for each set of conditions
- has four parts: the condition stubs, the action stubs, action entry and condition entry .

Contd..

- **Action entry:** It indicates the actions to be taken.
- **Condition entry:** It indicates conditions which are being met or answers the questions in the condition stub.
- **Action stub:** It lists statements described all actions that can be taken.
- **Condition stub:** It lists all conditions to be tested for factors necessary for taking a decision.

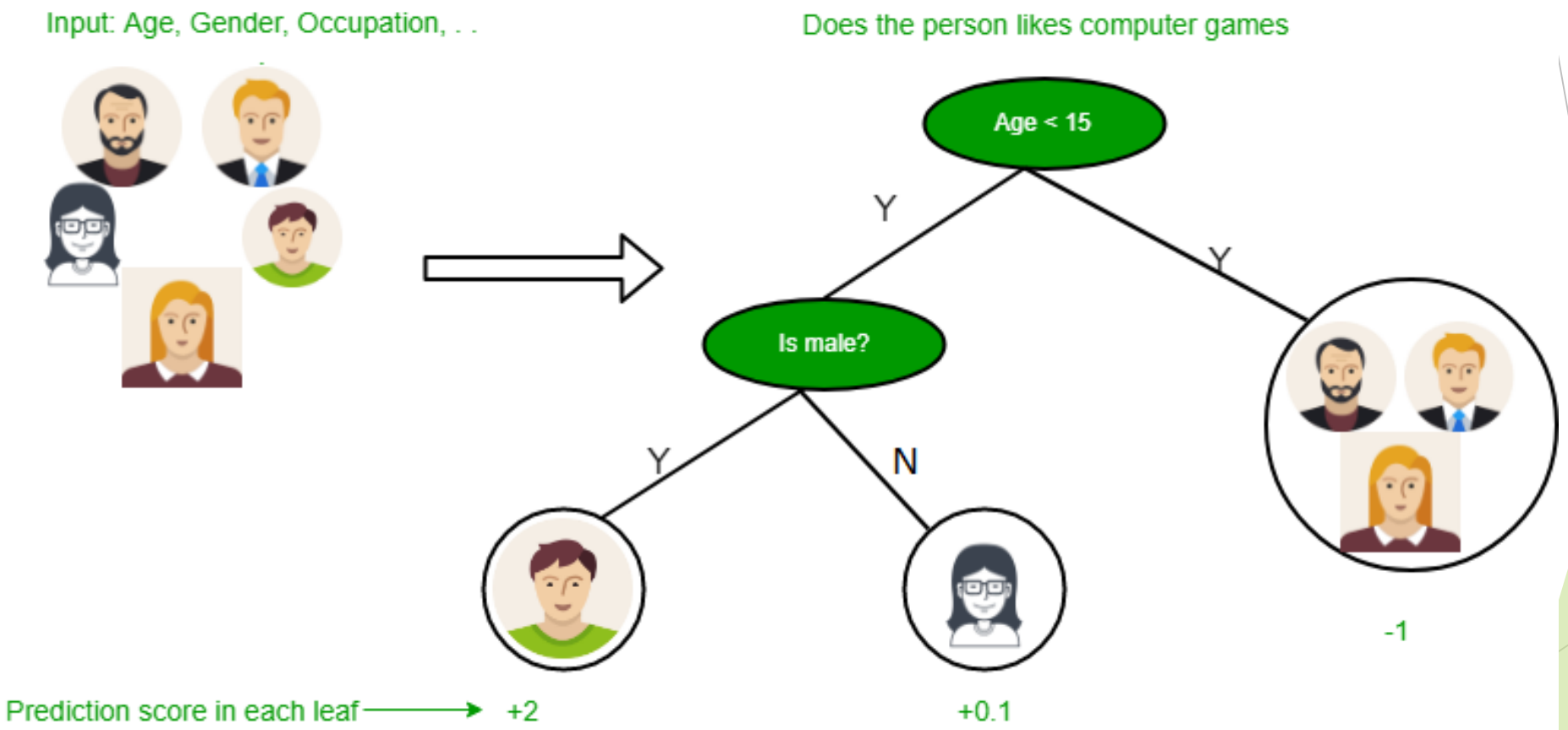
Examples:

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

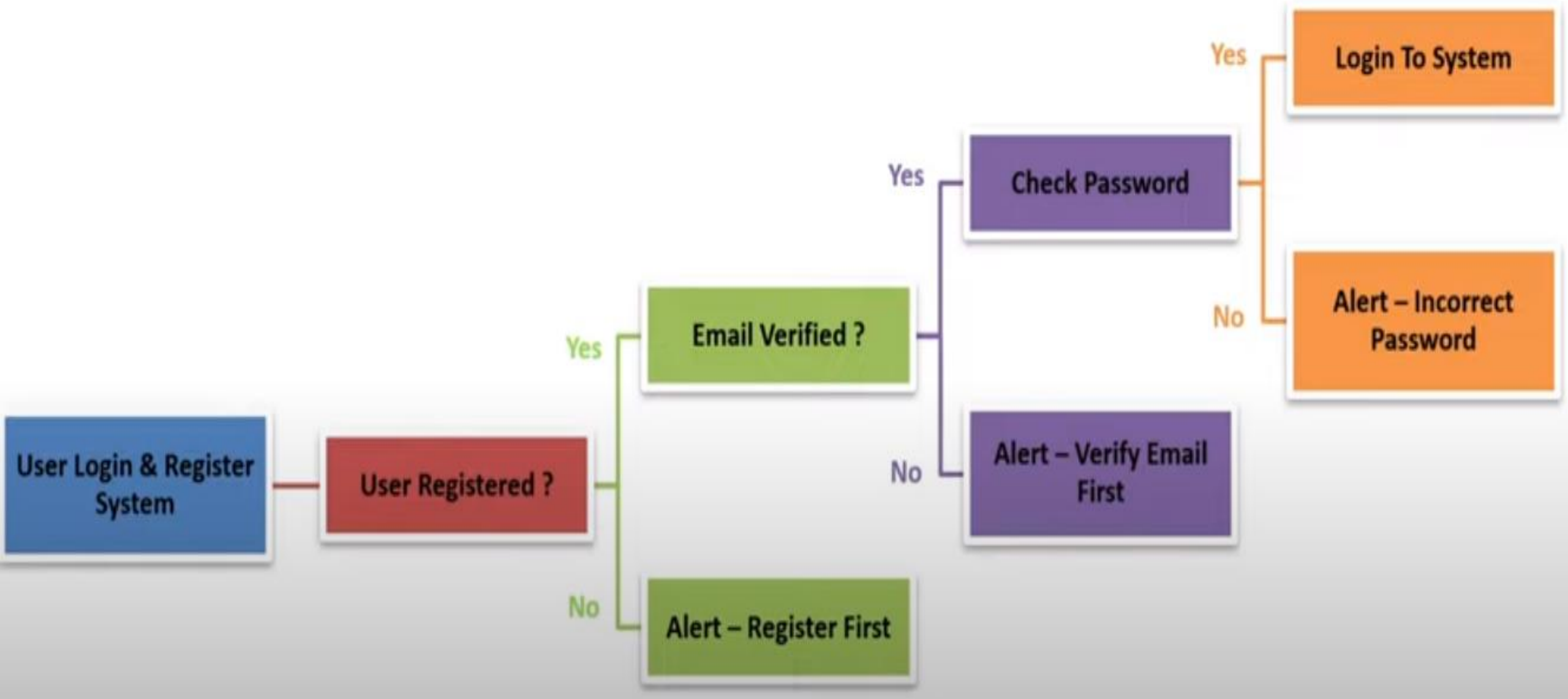
Decision Tree

- is a graph that always uses a branching method in order to demonstrate all the possible outcomes of any decision.
- are graphical and show a better representation of decision outcomes.
- It consists of three nodes namely Decision Nodes, Chance Nodes, and Terminal Nodes

Example


Input: Age, Gender, Occupation, . .
Does the person likes computer games
Age < 15
Y
Y
Is male?
Y
N
-1
Prediction score in each leaf
+2
+0.1

Technical example



S. No.	Decision Table	Decision Tree
1.	Decision Tables are a tabular representation of conditions and actions.	Decision Trees are a graphical representation of every possible outcome of a decision.
2.	We can derive a decision table from the decision tree.	We can not derive a decision tree from the decision table.
3.	It helps to clarify the criteria.	It helps to take into account the possible relevant outcomes of the decision.
4.	In Decision Tables, we can include more than one ‘or’ condition.	In Decision Trees, we can not include more than one ‘or’ condition.
5.	It is used when there are small number of properties.	It is used when there are more number of properties.
6.	It is used for simple logic only.	It can be used for complex logic as well.
7.	It is constructed of rows and tables.	It is constructed of branches and nodes.
8.	The goal of using a decision table is the generation of rules for structuring logic on the basis of data entered in the table.	A decision tree’s objective is to provide an effective means to visualize and understand a decision’s available possibilities and range of possible outcomes.

Pseudocodes

- is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations.
- is used for creating an outline or a rough draft of a program.
- summarizes a program's flow, but excludes underlying details

Unit 3.3 Structuring System Data Requirements

- Introduction
- Conceptual Data Modeling (Process, Deliverables and Outcomes)
- Gathering Information for Conceptual Data Modeling; Introduction to E-R Modeling (Entities, Attributes, Keys and Identifiers, Relationships: Degree, Cardinality, Naming and Defining Relationships, Associative Entity)

Entity Relationship Diagram (E-R Diagram)

- During analysis phase, a systems analyst uses entity relationship data model (E-R model) as a conceptual data model.
- A conceptual data model is a detailed model that captures the overall structure of organizational data while being independent of any database management system or other implementation consideration.
- E-R model is a detailed, logical representation of the data for an organization or for a business area.
- The E-R model is expressed in terms of entities in the business environment, the relationships or associations among those entities, and the attributes or properties of both the entities and their relationships.

Contd..

- An E-R model is normally expressed as an entity relationship diagram (E-R diagram), which is a graphical representation of an E-R model.
- It has three basic concepts: entities, attributes, and relationships.
- **Entities:** An entity is a person, place, object, event, or concept in the user environment about which the organization wishes to capture and store data.
- An **entity type** (sometimes called an **entity class** or **entity set**) is a collection of entities that share common properties or characteristics.
- An **entity instance** (or **instance**) is a single occurrence of an entity type.
- Each entity type in E-R diagram is given a name.

When naming entity types, we should use the following guidelines:

- An entity type name is a *singular noun* like CUSTOMER, STUDENT, or AUTOMOBILE.
- An entity type name should be *descriptive and specific to the organization* like PURCHASE ORDER for orders placed with suppliers to distinguish it from CUSTOMER ORDER for orders placed by customers.
- An entity type name should be *concise* like REGISTRATION for the event of a student registering for a class rather than STUDENT REGISTRATION FOR CLASS.
- *Event entity types* should be named for the *result of the event*, not the activity or process of the event like the event of a project manager assigning an employee to work on a project results in an ASSIGNMENT.

When defining entity types, we should use the following guidelines:

- An entity type definition should include a statement of what the unique characteristic(s) is (are) for each instance.
- An entity type definition should make clear what entity instances are included and not included in the entity type.
- An entity type definition often includes a description of when an instance of the entity type is created or deleted.
- For some entity types the definition must specify when an instance might change into an instance of another entity type. For example, a bid for a construction company becomes a contract once it is accepted.
- For some entity types the definition must specify what history is to be kept about entity instances

Contd..

- An entity type in E-R diagram is drawn using **rectangle**.
- This shape represents all instances of the named entity. We place entity type name inside the rectangle.
- For example, the figure below shows STUDENT entity type.

Attributes

- Each entity type has a set of attributes associated with it.
- An attribute is a property or characteristic of an entity that is of interest to the organization.
- For example, STUDENT entity type may have Student_ID, Student_Name, Home_Address, Phone_Number, and Major as its attributes.
- Similarly, EMPLOYEE entity type may have Employee_ID, Employee_name, and Skill, Address as its attributes.
- Each attribute in E-R diagram is given a name.

When naming attributes, we should use the following guidelines:

- An attribute is a noun like Customer_ID, Age, or Skill.
- An attribute name should be unique.
- No two attributes of the same entity type may have the same name, and it is desirable, for clarity purposes, that no two attributes across all entity types have the same name.
- To make an attribute unique and clear, each attribute name should follow a standard form.
- For example, Student_GPA as opposed to GPA_of_Students.
- Similar attributes of different entity types should use the similar but distinguishing names.
- For example, Student_Residence_City_Name for STUDENT entity type and Faculty_Residence_City_Name for FACULTY entity type.

When defining attributes, we should use the following guidelines:

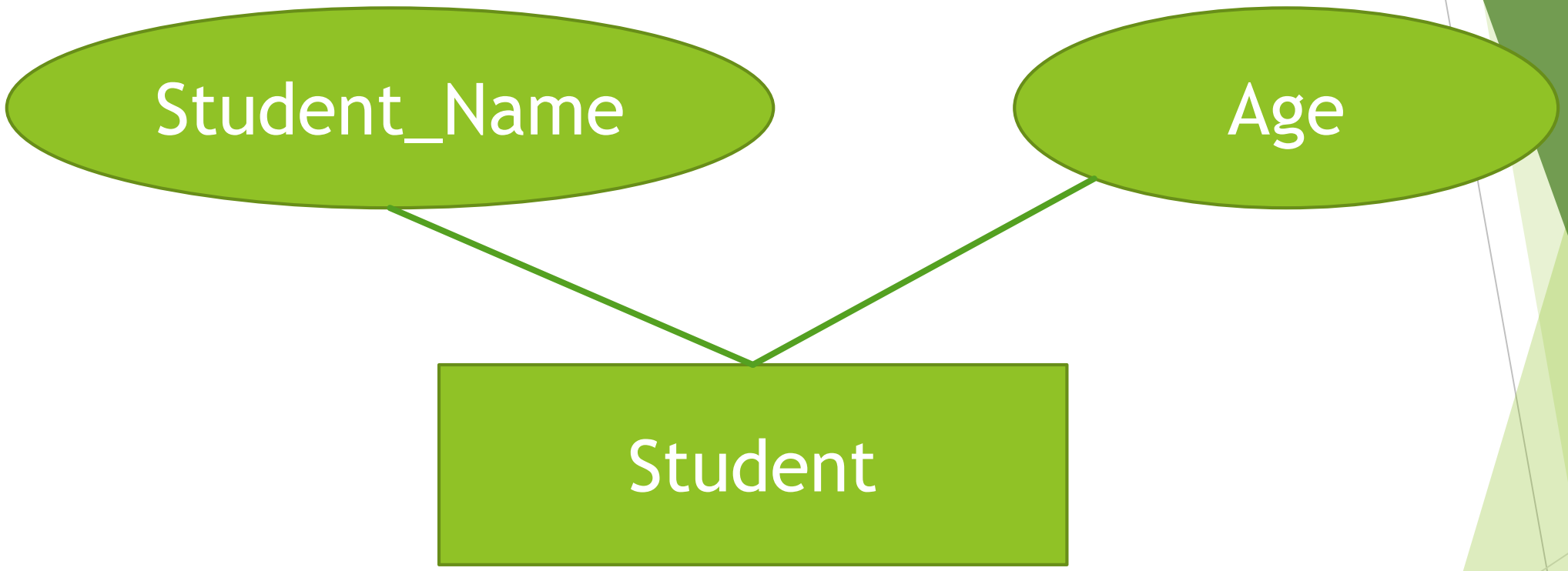
- An attribute definition states what the attribute is and possibly why it is important.
- An attribute definition should make it clear *what is included and what is not included* in the attributes value. For example, Employee_Monthly_Salary_Amount is the amount paid each month exclusive of any benefits, bonuses, or special payments.
- An attribute definition may contain any *aliases* or alternative names.
- An attribute definition may state *the source of values for the attribute to make the meaning clearer*.
- An attribute definition should indicate *if a value for the attribute is required or optional* (to maintain data integrity).
- An attribute definition may indicate *if a value for the attribute may change* (to maintain data integrity).
- An attribute definition may also indicate any *relationships that an attribute has with other attributes*. For example, Age is determined from for Date_of_Birth.

Contd..

- An attribute in E-R diagram is drawn using an **ellipse**.
- We place attribute name inside the ellipse with a line connecting it to the associated entity type.
- For example, the figure below shows Student_Name and Age attributes of STUDENT entity type.

Contd..

- Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type.
- A candidate key is an attribute or combination of attributes that uniquely identifies each instance of an entity type.
- For example, a candidate key for a STUDENT entity type might be Stident_ID.
- Some entity type may have more than one candidate key. In such a case, we must choose one of the candidate keys as the identifier.
- An identifier (or primary key) is a candidate key that has been selected to be used as the unique characteristic for an entity type.

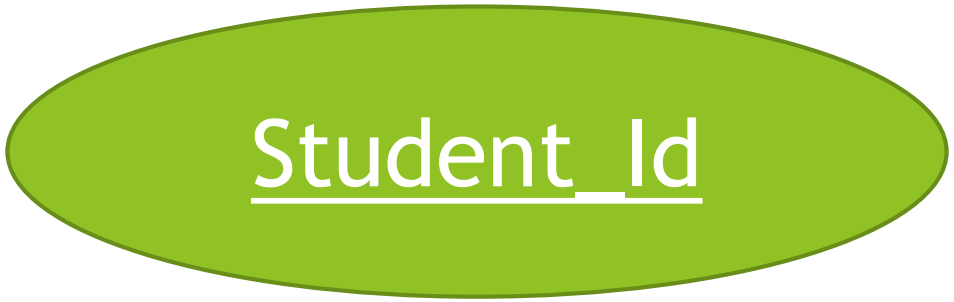

Student_Name
Age
Student

We can use the following selection rules to select identifiers:

- Choose a candidate key that will not change its value over the life of each instance of the entity type.
- Choose a candidate key that will never be null.
- Avoid using intelligent keys.
- Consider substituting single value surrogate keys for large composite keys.
- The name of the identifier is underlined on an E-R diagram. For example, the figure below shows Student_Id as an identifier for a STUDENT entity type.

Contd..

- The name of the identifier is underlined on an E-R diagram.
- For example, the figure below shows Student_Id as an identifier for a STUDENT entity type.

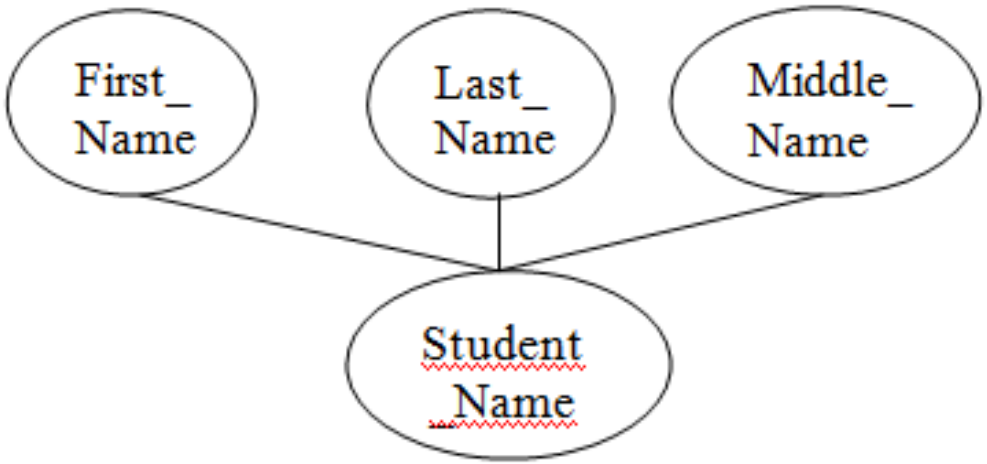


Contd..

- A **multivalued attribute** may take more than one value for each entity instance.
- For example Phone_Number attribute of STUDENT entity type.
- We use a double-lined ellipse to represent multivalued attribute.

Composite attribute

- An attribute that has meaningful component parts
- For example, Student_Name attribute of STUDENT entity type has First_Name, Middle_Name, and Last_Name as its component parts.



Derived attribute

- An attribute whose value can be computed from related attribute values.
- For example, value of Age attribute is computed from Date_of_Birth attribute.
- We use dashed ellipse to denote derived attribute.

Relationships

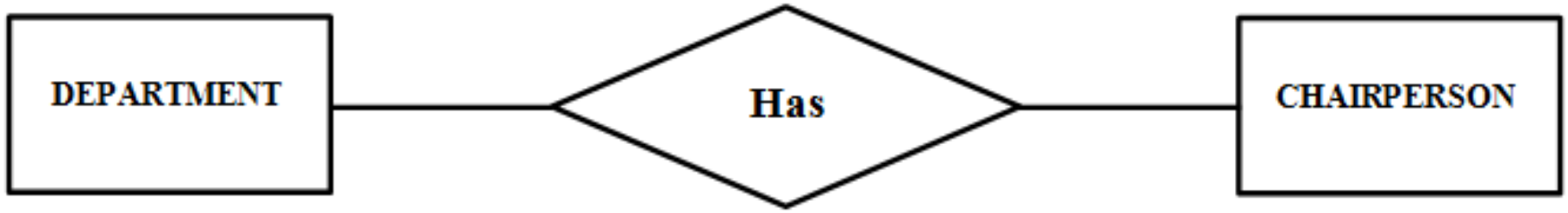
- is an association between the instances of one or more entity types that is of interest to the organization.
- We use diamond to denote relationships. Relationships are labeled with verb phrases.
- For example, if a training department in a company is interested in tracking with training courses each of its employees has completed, this leads to a relationship (called Completes) between the EMPLOYEE and COURSE entity types as shown in the figure below.

Cardinality of a relationship

- As indicated by arrows, the cardinality of this relationship is many-to-many since each employee may complete more than one course, and each course may be completed by more than one employee.
- The cardinality of a relationship is the number of instances of one entity type that can (or must) be associated with each instance of another entity type.
- The cardinality of a relationship can be in one of the following four forms: one-to-one, one-to-many, many-to- one, and many-to-many.

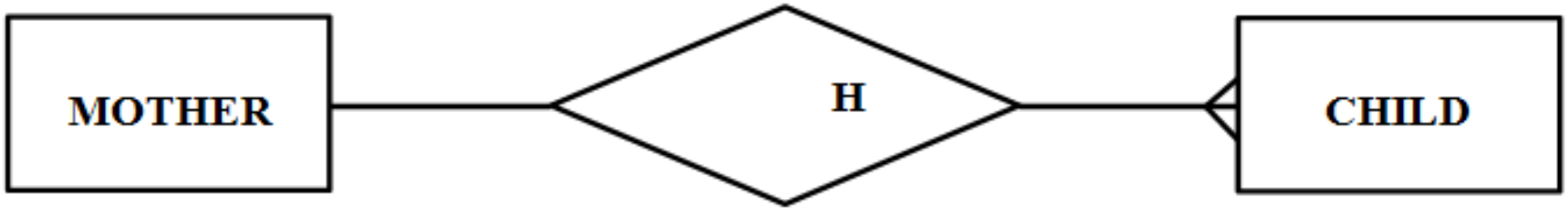
One-to-one:

- An instance in entity type A is associated with at most one instance in entity type B, and an instance in entity type B is associated with at most one instance in entity type A.
- For example, cardinality between DEPARTMENT and CHAIRPERSON.



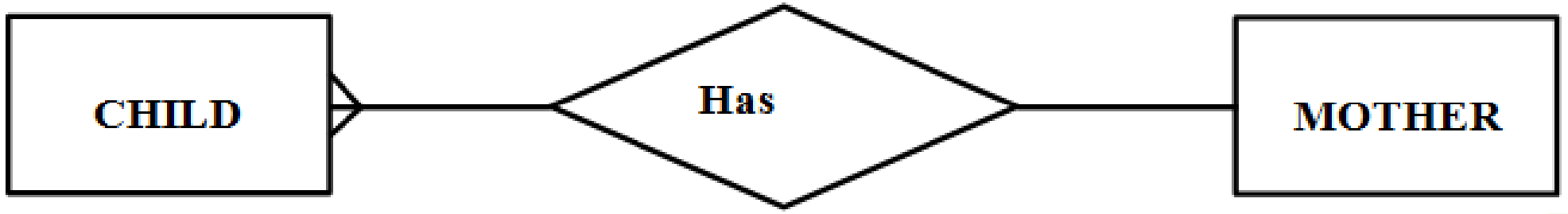
One-to-many:

- An instance in entity type A is associated with any number (zero or more) of instances in entity type B, and an instance in entity type B, however, can be associated with at most one instance in entity type A.
- For example, cardinality between MOTHER and CHILD.



Many-to-one:

- An instance in entity type A is associated with at most one instance in entity type B, and an instance in entity type B, however, can be associated with any number (zero or more) of instances in entity type A.
- For example, cardinality between CHILD and MOTHER.

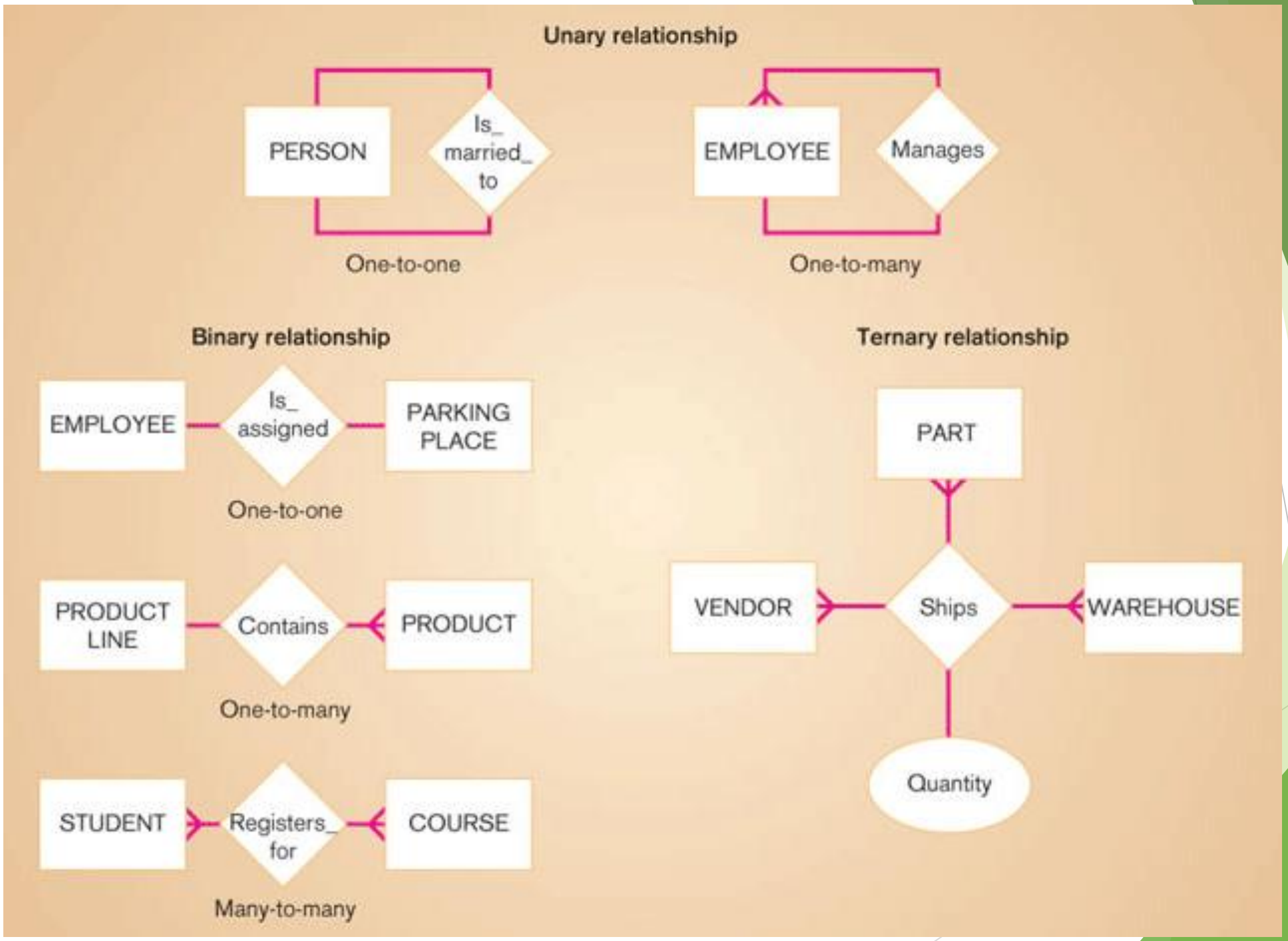


Many-to-many:

- An instance in entity type A is associated with any number (zero or more) of instances in entity type B, and an instance in entity type B is associated with any number (zero or more) of instances in entity type A.
- For example, cardinality between STUDENT and COURSE.

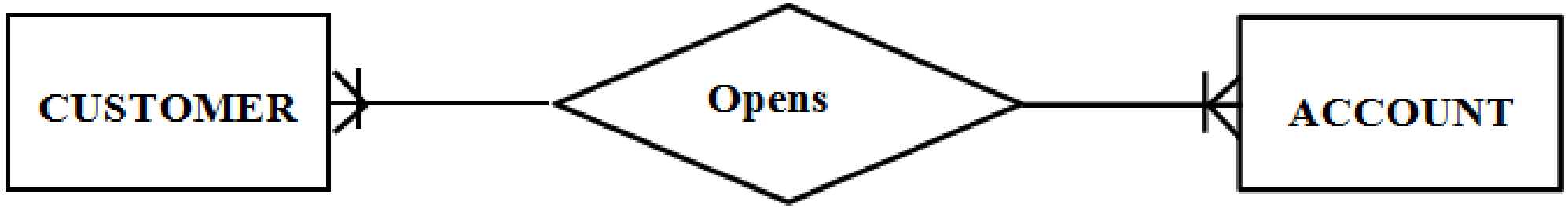
Degree of a relationship

- is the number of entity types that participate in the relationship.
- The three most common relationships in E-R models are
  - unary (degree one): is a relationship between the instances of one entity type
  - binary (degree two): is a relationship between instances of two entity types
  - ternary (degree three): is a simultaneous relationship among the instances of three entity types
- Higher degree relationships are also possible, but they are rarely encountered.


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

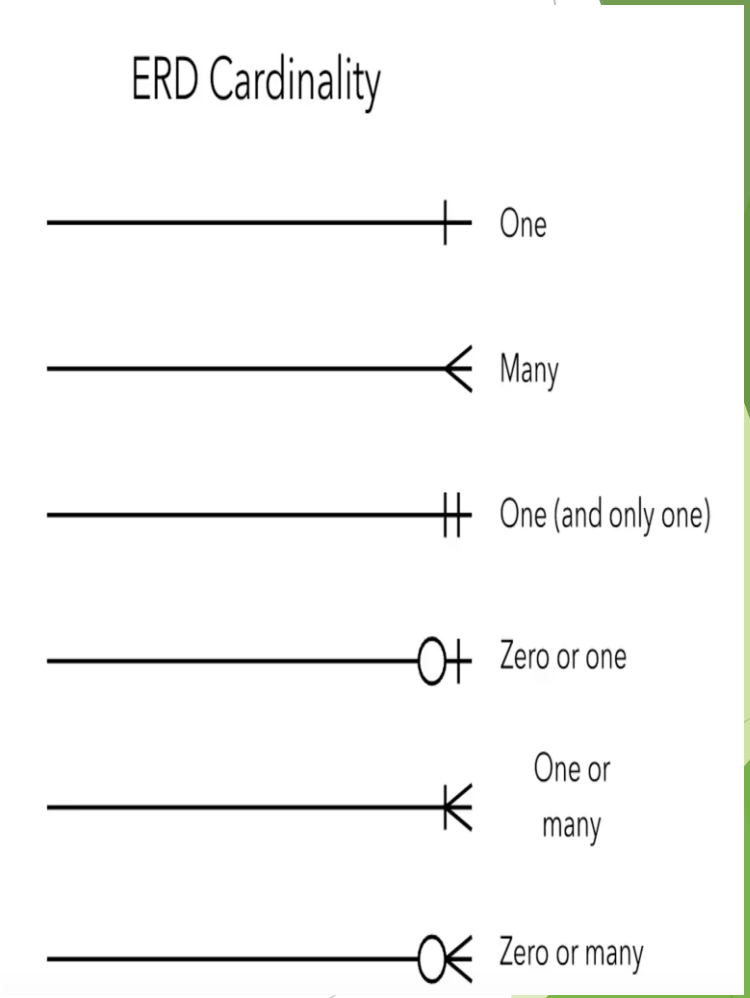
Cardinality

- We can also use the concept minimum and maximum cardinality when we draw E-R diagrams.
- The ***minimum cardinality*** of a relationship is the minimum number of instances of an entity type that may be associated with each instance of another entity type.
- The ***maximum cardinality*** of a relationship is the maximum number of instances of an entity type that may be associated with each instance of another entity type.
- For example, suppose a portion of banking database as shown in the figure below. Here, the minimum number of accounts for a customer is one and the maximum number of accounts is many (unspecified number greater than one). Similarly, the minimum number of customers associated with an account is one and the maximum number of customers is many.


CUSTOMER
Opens
ACCOUNT

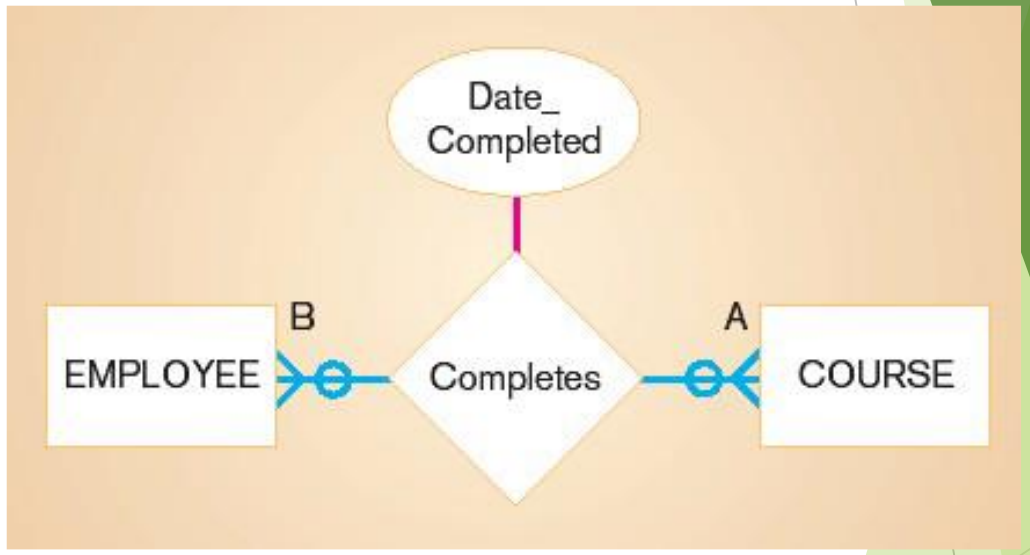
Cardinality

- Defines the numerical attributes of the relationship between two entities or entity sets.
- Different types of cardinal relationships are:
  - One-to-One Relationships
  - One-to-Many Relationships
  - Many to One Relationships
  - Many-to-Many Relationships



Other E-R Notations

1. **Associative Entity:** An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances is called associative entity. Associative entity is also called a gerund. For example, the first figure below shows a relationship with an attribute and the second figure shows an associative entity.



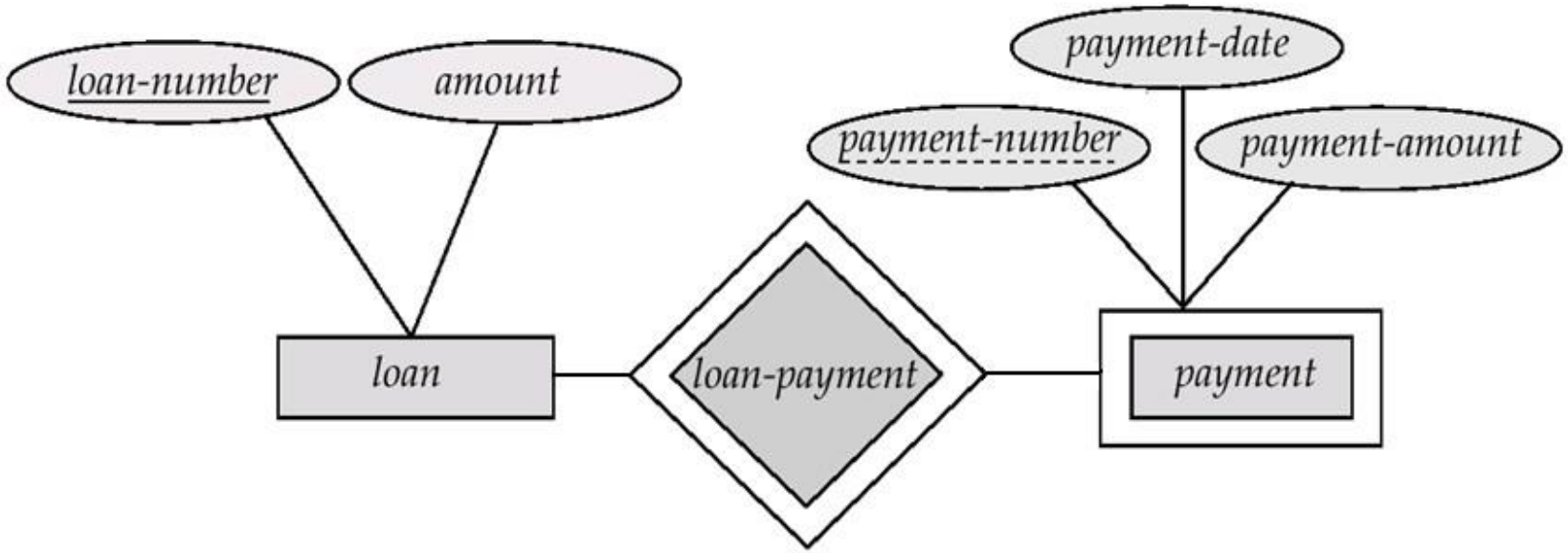
2. **Weak Entity Type:** An entity type may not have sufficient attributes to form a primary key. Such an entity type is termed as a weak entity type.

An entity type that has a primary key is termed as a strong entity type.

For a weak entity type to be meaningful, it must be associated with another entity type, called the identifying or owner entity type, using one of the key attribute of owner entity type.

The weak entity type is said to be existence dependent on the identifying entity type.

Although a weak entity type does not have a primary key, we use discriminator (or partial key) as a set of attributes that allows the distinction to be made among all the entities in the weak entity type.



- **Supertypes and Subtypes**
- **Subtype:** Subtype is a subgrouping of the entities in an entity type that is meaningful to the organization and that shares common attributes or relationships distinct from other subgroupings. For example, an entity type STUDENT to GRADUATE STUDENT and UNDERGRADUATE SUTDENT.
- **Supertype** is a generic entity type that has a relationship with one or more subtypes. For example PATIENT with OUTPATIENT and RESIDENTPATIENT. There are several important business rules for supertype/subtype relationships.
- **Total specialization rule:** Specifies that each entity instance of the supertype must be a member of some subtype in the relationship.
- **Partial specialization rule:** Specifies that an entity instance of the supertype does not have to belong to any subtype. May or may not be an instance of one of the subtypes.
- **Disjoint rule:** Specifies that if an entity instance of the supertype is a member of one subtype, it cannot simultaneously be a member of any other subtype.
- **Overlap rule:** Specifies that an entity instance can simultaneously be a member of two (or more) subtypes.

Assignment 3.3

1. Discuss the use of interviewing technique for information gathering.
2. Explain JAD method for determining requirements? What are the benefits of using JAD?
3. Draw a DFD for your project I upto level 2.
4. What are the key steps for designing E-R diagram? Explain with example.
5. What are the three relationship types of E-R diagrams? How are these relationships paired to build an ER-diagram?
6. Design the ER-diagram of the following.
   - Customer with draws money from his account
   - Student attends classes.