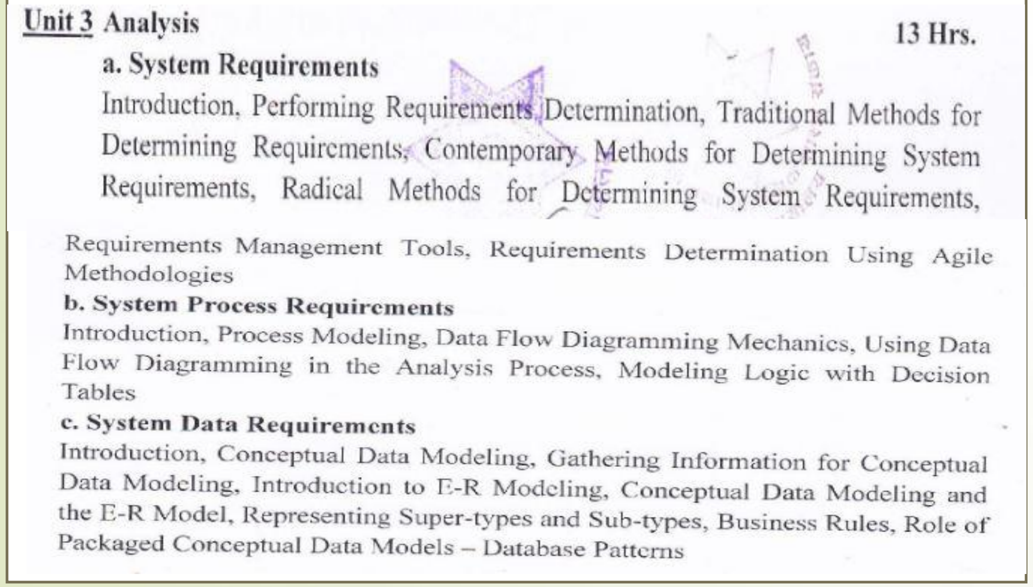


Unit 3 Analysis 13 Hrs.

a. System Requirements

Introduction, Performing Requirements Determination, Traditional Methods for Determining Requirements, Contemporary Methods for Determining System Requirements, Radical Methods for Determining System Requirements, Requirements Management Tools, Requirements Determination Using Agile Methodologies

b. System Process Requirements

Introduction, Process Modeling, Data Flow Diagramming Mechanics, Using Data Flow Diagramming in the Analysis Process, Modeling Logic with Decision Tables

c. System Data Requirements

Introduction, Conceptual Data Modeling, Gathering Information for Conceptual Data Modeling, Introduction to E-R Modeling, Conceptual Data Modeling and the E-R Model, Representing Super-types and Sub-types, Business Rules, Role of Packaged Conceptual Data Models – Database Patterns

SYSTEM ANALYSIS AND DESIGN(SAD)

Chapter 3: **Analysis**

Introduction

- Systems analysis is a critical phase in the **Systems Development Life Cycle (SDLC)**, where the functionality of the existing information system is evaluated, and user requirements for a new system are assessed. This stage ensures that the proposed system aligns with business needs and improves efficiency . Analysis has two sub phases: **requirements determination** and **requirements structuring.**
- The purpose of analysis is to determine what information and information processing services are needed to support selected objectives and functions of the organization. Gathering this information is called **requirements determination**.
- The results of the requirements determination can be structured according to three essential views of the current and replacement information systems:

Process: The sequence of data movement and handling operations within the system.

Logic and timing: The rules by which data are transformed and manipulated and an indication of what triggers data transformation.

Data: The inherent structure of data independent of how or when they are processed.

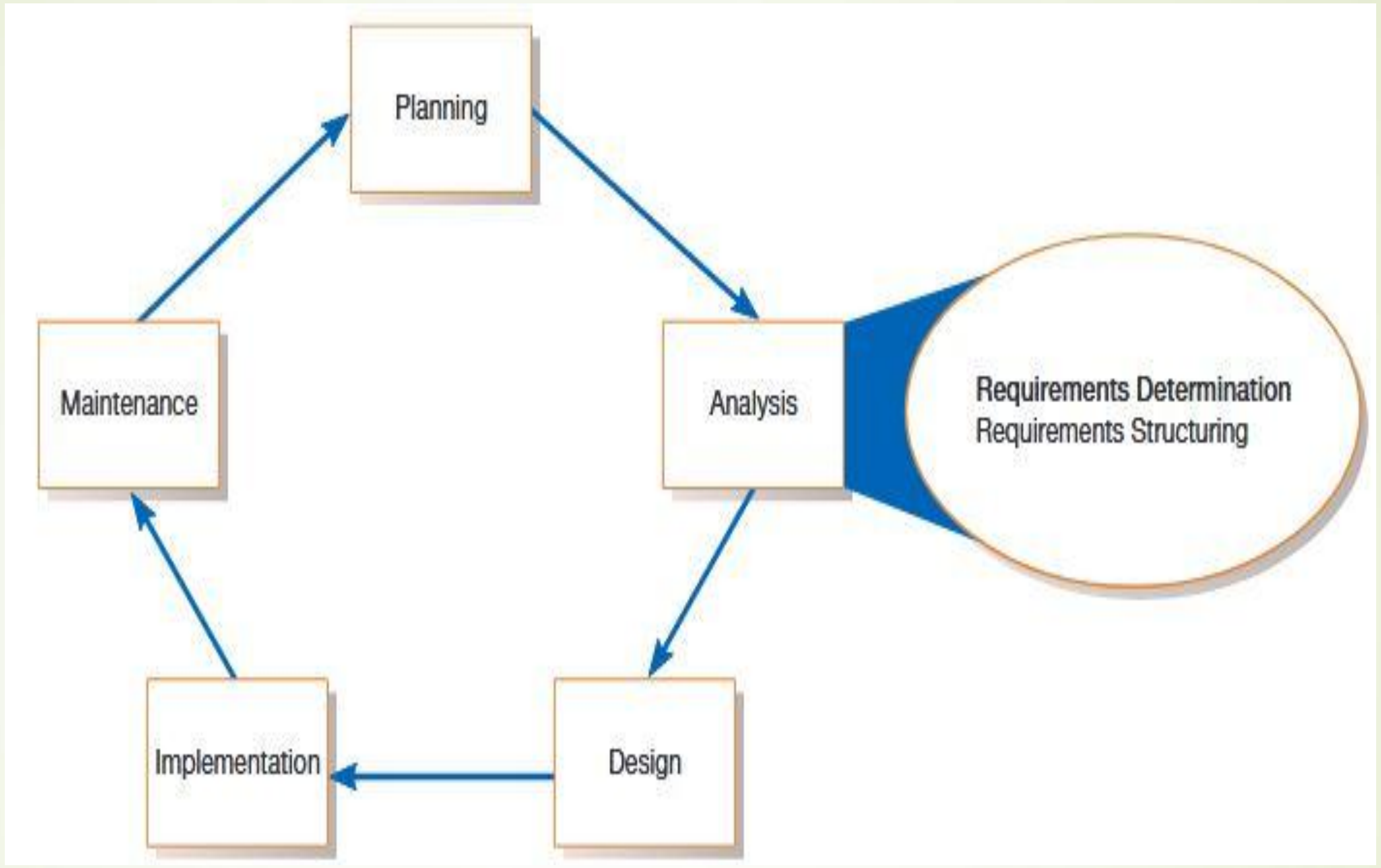


Fig: Systems development life cycle with analysis phase highlighted.

Determining System Requirements

- Determining system requirements is a key aspect of **systems analysis and design**, evolving from traditional methods to more structured, computer-supported approaches. Initially, requirements were gathered through **interviews, user observation, and document analysis**.
- Modern techniques, such as **Joint Application Design (JAD)**, promote collaboration between stakeholders, while **CASE tools** and **prototyping** enhance efficiency in requirement analysis. Agile methodologies, **business process redesign**, and **Internet-based application development** further refine the process.
- Regardless of the approach, effective **requirements analysis** ensures that the final system aligns with user needs and business goals.

Performing Requirements Determination

- Systems analysis consists of two key sub phases: **requirements determination** and **requirements structuring**. Although treated as separate steps, these processes are **parallel and iterative**. As analysts gather system requirements, they simultaneously structure the information or develop **prototypes** to visualize system behavior.
- Through **structuring and prototyping**, inconsistencies or gaps in the current and desired systems become evident, prompting further exploration of operational needs. Over time, this iterative approach results in a **comprehensive and accurate representation** of both existing operations and the requirements for the new system.

Characteristics for gathering requirements

A successful systems analyst must possess key traits to gather, analyze, and structure system requirements effectively. These include:

Impertinence (Question Everything)

- A good analyst challenges assumptions and seeks deeper insights.
- Key questions to ask:
  - ✓ *Are all transactions processed the same way?*
  - ✓ *Are there exceptions to standard pricing?*
  - ✓ *Should employees be allowed to work across multiple departments?*
- The goal is to identify gaps, inconsistencies, and opportunities for improvement.

Impartiality (Find the Best Solution, Not Just What Users Want)

- The analyst must focus on solving real business problems rather than just fulfilling user requests.
- Avoid biases, such as:
- Justifying the purchase of new hardware/software without valid reasons.
- Accepting all user demands without considering their feasibility or impact on the organization.
- The best solution should be based on organizational efficiency, cost-effectiveness, and long-term sustainability.

Relax Constraints (Encourage Innovation and Change)

- Analysts should **challenge existing constraints** and question traditional practices.
- Example: Instead of accepting *"We've always done it this way"*, analysts should:
    - Differentiate between **rules, policies, and traditions**.
    - Identify outdated practices that no longer serve a purpose.
    - Recommend **innovative solutions** that align with modern business needs.

Attention to Detail (Ensure Accuracy and Consistency)

- Every **fact, requirement, and assumption** must align correctly to avoid errors.
- Example of a **potential risk**:
    - An unclear **definition of a customer** could lead to **deleting past customer data** when they have no active orders.
    - This mistake may result in **losing important client records** needed for future sales and marketing.

Reframing (Thinking Creatively and Avoiding Bias**)**

- **Analysis is a creative process** that requires looking at the organization from multiple perspectives.
- A systems analyst must consider how **each user perceives their requirements**.
- Avoid jumping to conclusions based on past experiences, such as:

  "I worked on a system like this before, so this new system must work the same way."
- Every system has **unique challenges and requirements**, and it is important to analyze them independently.

Types of deliverables:

- From interviews and observations - interview transcript observation notes, meeting minutes
- From existing written documents - mission and strategy statements, business forms, procedure manuals, job descriptions, training manuals, system documentation, flowcharts
- From computerized sources – Joint Application Design, session results, CASE repositories, reports from existing systems, displays and reports from system prototype.
- Information collected from users
- Existing documents and files
- Computer-based information

Understanding of organizational components

- Business objective
- Information needs
- Rules of data processing
- Key events

Traditional Methods for Determining Requirements

1) Interviewing and listening: Interviewing and listening involves talking with users individually or as a group to discover their views about the current and target systems. It also involves careful preparing an interview outline and guidelines before conducting the interview. Interviews are best done when only a few people are involved.

Types of interview questions:

Open-ended questions: These are usually used to examine for which we cannot expect all possible responses.

Closed-ended questions: Provide a range of possible answers from which the interviewer may choose (True/False, Multiple choice, ranking items etc.).

2) Group Interviewing: Group interviewing consists of taking Interview of several key people, together. The advantages of group interviewing. are more. effective use of time, can hear agreements and disagreements at once etc. The disadvantage. as that it is more difficult to schedule than Individual interviews.

3) Directly observing users: Interviewing involves getting people. to recall and convey information they have about organizational processes and the information systems that support them. People, are not always reliable and say what they think is the truth. People often do not have a completely accurate appreciation of what they do or how they do it. Because people. cannot always be trusted to interpret and report their own actions reliably, we can supplement what people tell us by watching what they do in work situations.

4) Analyzing Procedures and Other Documents: Both interviewing and observing have some sort of limitations. Methods for determining system requirements can be enhanced by analyzing procedures and other documents of an organization. By examining existing system and organizational documentation, system analyst can find out details about current system and the organization. In In documents analyst can find Information such as problems with existing system, reasons why current systems are designed and many more.

Contemporary Methods for determining system requirements:

1. Joint Application Design (JAD)

- Joint application development is a methodology that involves the client or end user in the design and development of a software application through a succession of collaborative workshops called JAD sessions. Its aim is to involve the customer (or end user) in defining the business need and developing a solution based on that need.
- JAD is a unique approach to software and systems development because it involves clients, customers or end users throughout the product design and development lifecycle. It emphasizes a team-oriented development approach along with a group consensus-based problem-solving model.
- JAD techniques suited to projects that require engineering analysis and design of systems with emphasis on the development of participation between the system owners, users, designers, and builders.
- 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

- Upper CASE tools are used
- Enables analysts to enter system models directly into CASE during the JAD session
- Screen designs and prototyping can be done during JAD and shown to users

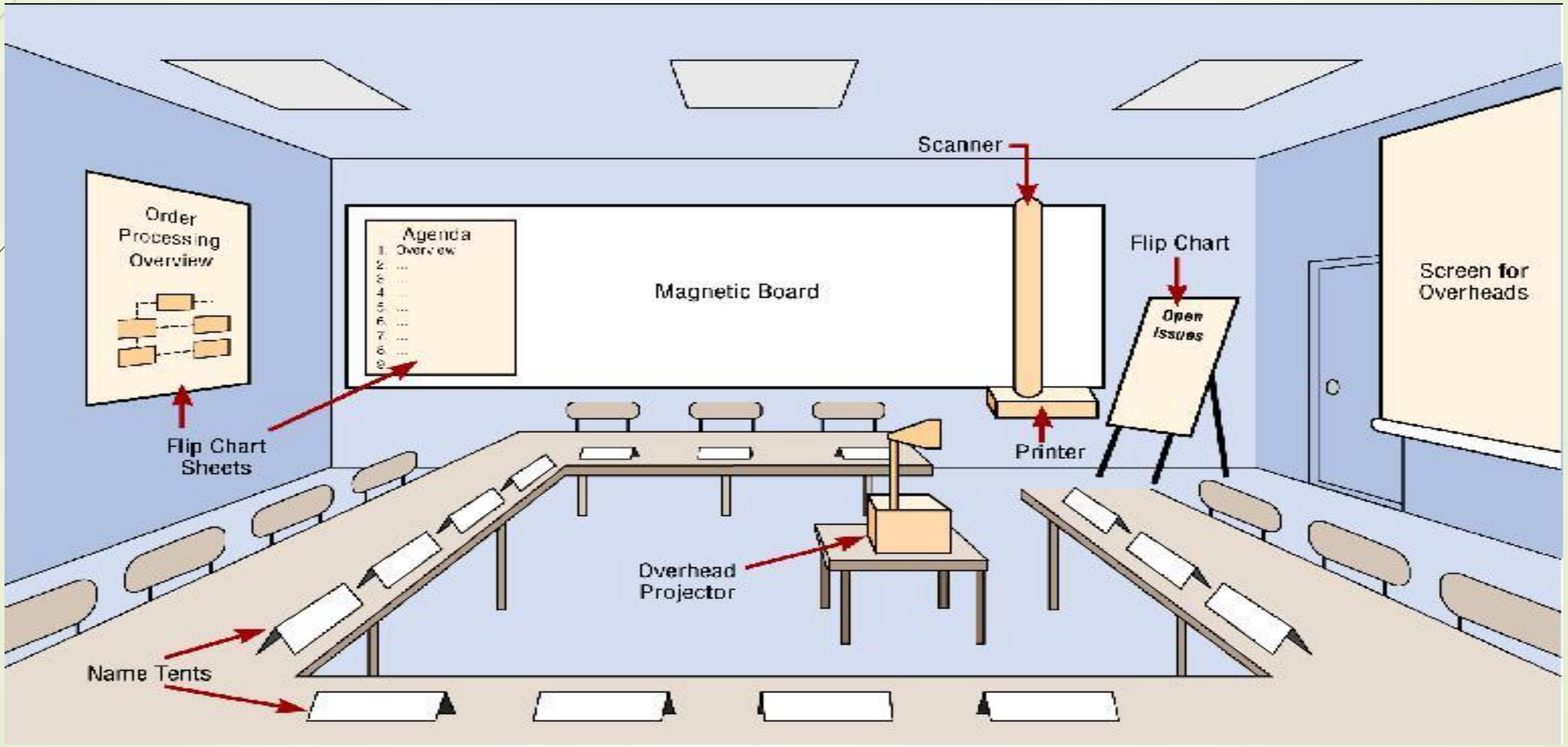


Fig: Illustration of typical room layout for a JAD

Advantages of JAD:

- JAD reduces costs and time for project development.
- JAD encourages the team to push each other to work faster and deliver on time.
- JAD allows us to produce better, error-free software.
- JAD lowers the risks.
- JAD Improves system quality.

2. Prototyping

- Prototyping is an iterative process involving analysis and users whereby a rudimentary version of an information system is built and rebuilt according to user feedback. Prototyping can replace the SDLC or augment it. Prototyping will allow you to quickly convert basic requirements into a working, though limited, version of the desired information system. Converting verbal descriptions of requirements into a physical system.
- 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.

Prototyping is most useful for requirements determination when:

- User requests are not clear
- Few users are involved in the system
- Designs are complex and require concrete form
- History of communication problems between analysts and users
- Tools are readily available to build prototype

Drawbacks

- Tendency to avoid formal documentation
- Difficult to adapt to more general user audience
- Sharing data with other systems is often not considered
- Systems Development Life Cycle (SDLC) checks are often bypassed

Radical methods for determining system requirements:

1. **Business Process Reengineering:** The overall process by which current methods are replaced with radically new methods is referred to as business process reengineering (BPR). Reorganize complete flow of data in major sections of organization, Eliminate, unnecessary steps, Combine steps, and Become more responsive to future change are it's main - goals. This method Search for implementation of radical change in business processes to achieve breakthrough improvements in products and services.
2. **Identification of process to reengineer:** The first step in any BPR effort as 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. Key business processes are customer focused. After identifying key business processes, the next step is to identify specific activities that can be radically improved through reengineering.
3. **Disruptive Technologies:** Technologies that enable breaking long held business rules that discourage organizations from making radical business changes, are called disruptive technologies. In this method problem are first identified and then solutions are formulated.

Structuring System Process Requirements:

Process Modeling: Process modeling involves graphically representing the processes, or actions, that capture, manipulate, store and distribute data between the system and it's environment and among components within the system. A common form of a process model is data-flow diagram (DFD).

Modeling of system's process: Analysts use both process and data models to establish the specification of an Information System. With a supporting tool, such as CASE tool, process and data models, can also provide the basis for the automatic generation of an information system.

Deliverables and Outcomes:

In structured analysis, the primary deliverables from process modeling are set of coherent, inter-related data-flow diagrams. Following are the deliverables for process modeling:

- ✓ Context DFD
- ✓ DFDs of current physical system.
- ✓ DFDs of new new logical system.
- ✓ Through description of each DFD component.

Data Flow Diagrams

- A Data Flow Diagram (DFD) is a visual representation of how data moves through a system or process, using standardized symbols to illustrate data flow, processes, data stores, and external entities, aiding in understanding and analysis.
- Data flow diagrams can be divided into **logical and physical**. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

Structure of DFD allows starting from a broad overview and expands it to a hierarchy of detailed diagrams, DFD has often been used due to the following reasons:

- Logical information flow of the system.
- Determination of physical system construction requirements.
- Simplicity of notation.
- Establishment of manual and automated systems requirements.

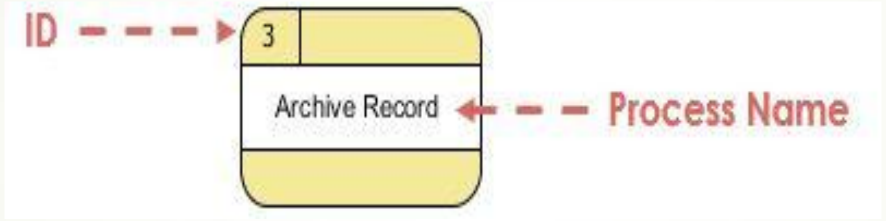
DFD Symbols

There are **four basic symbols** that are used to represent a data-flow diagram.

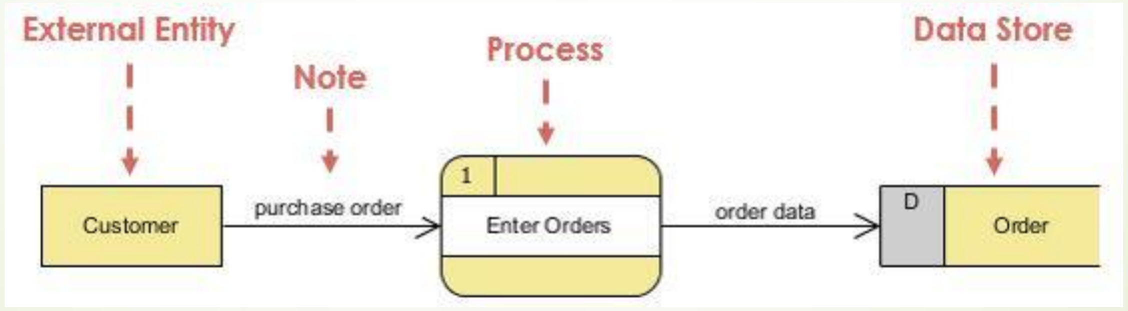
1. Process
2. Data Flow
3. Data Store
4. External Entity

Process

- A process receives input data and produces output.
- Every process has a name that identifies the function it performs. The name consists of a verb, followed by a singular noun.
- **Example:** Apply payment, Verify order etc.
- Notation: A rounded rectangle represents a process. Process are given IDs for easy referencing.



Process Example



Data Flow

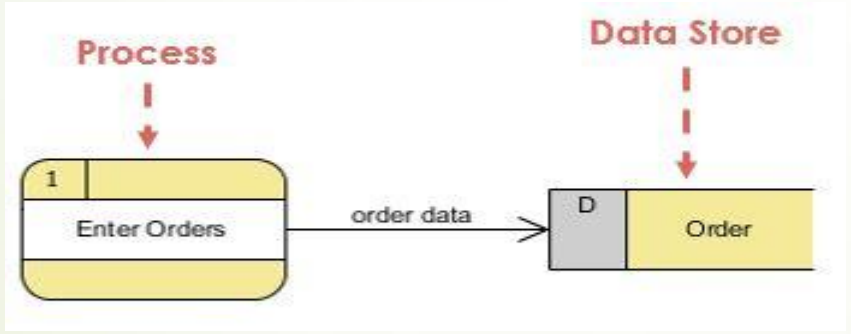
A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent single data element such as customer ID or It can represent a set of data element (or a data structure).

Notation:

→Straight lines with incoming arrows are input data flows.

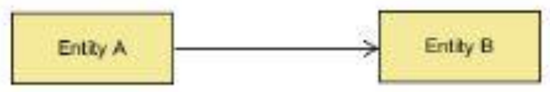
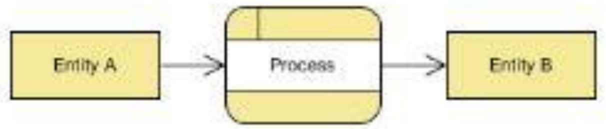
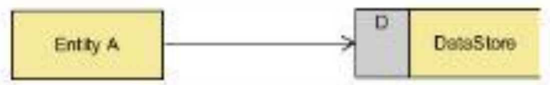
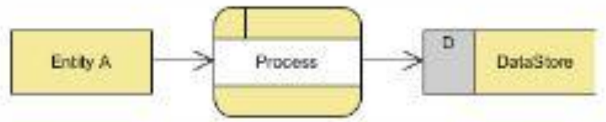
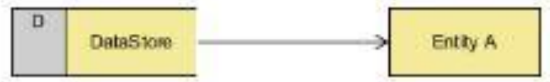
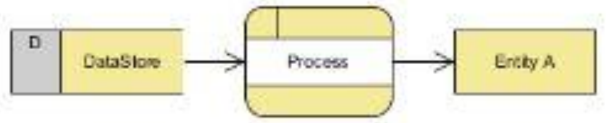
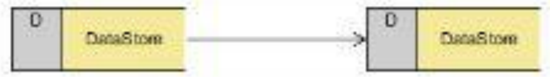
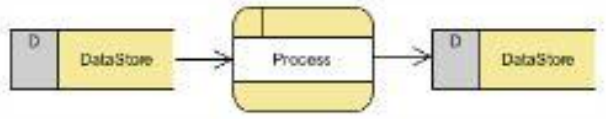
→Straight lines with outgoing arrows are output data flows.

Example



Rule of Data Flow

- One of the rule for developing DFD is that all flow must begin with and end at a processing step. This is quite logical, because data can't transform on its own with being process. By using the thumb rule, it is quite easily to identify the illegal data flows and correct them in a DFD.

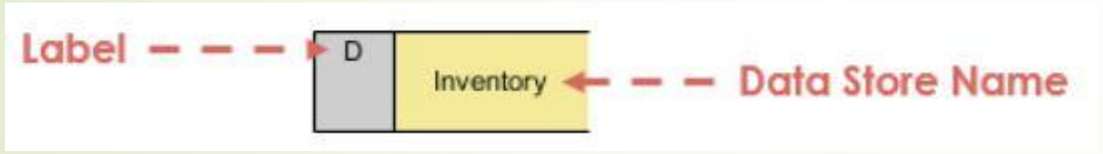
Wrong	Right	Description
		An entity cannot provide data to another entity without some processing occurred.
		Data cannot move directly from an entity to a data story without being processed.
		Data cannot move directly from a data store without being processed.
		Data cannot move directly from one data store to another without being processed.

Data Store

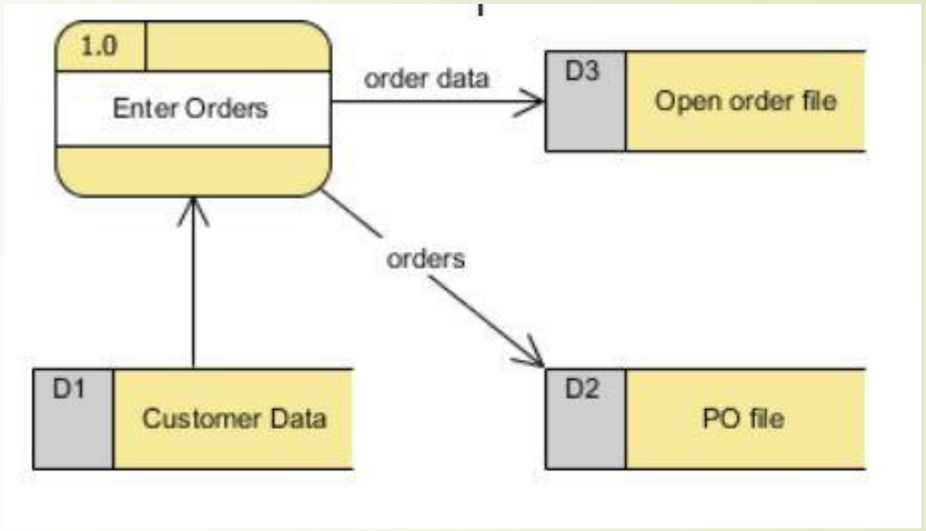
A data-store is used in DFD to represent a Situation when the system must retain data because one or more processes need to use the stored data in a later time. A data-store must be connected to a process with a data-flow. Each data store must have at least one input data-flow and at least one output data-flow.

Notation:

- ✓ Data can be written into data store, representing by outgoing arrow.
- ✓ Data can be read from data store, representing by an incoming arrow.



Example:



Note that:

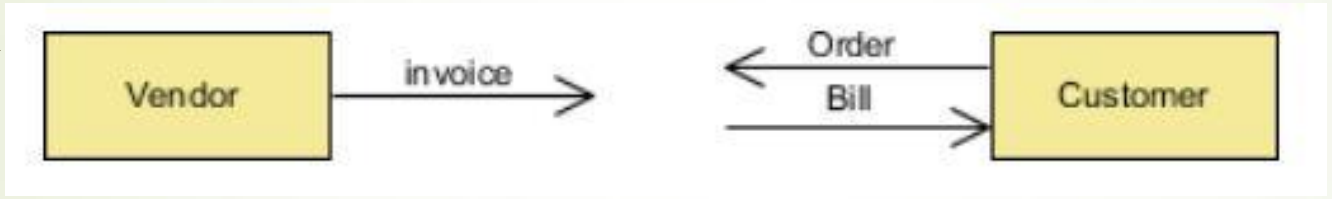
- ✓ A data store must be connected to a process with a data-flow.
- ✓ Each data store must have at least one input data-flow and at least one output data-flow (even if the output data-flow is a control or confirmation message).

External Entity:

An external entity is a person, department, outside organization, or other information System that provides data to the system or receives output from the system. External entities are also called terminators because they are data origins or final destinations. An external entity must be connected to a process through a data flow

Notation:

A rectangle represents an external entity. They either supply data or receive data but not process data.



- A customer submitting an order and then receive a bill from the system
- A vendor issue an invoice

Guideline for Developing Data-Flow Diagram

Context Diagram - Level 0

- ✓ The context diagram must fit in one page.
- ✓ The process name in the context diagram should be the name of the information system.

  For example, Grading System, Order Processing System, Registration System.
- ✓ The context level diagram gets the number 0 (level zero).

Unique Name for Levels

- ✓ Use unique names within each set of symbols.
- ✓ For example, there can be only one entity CUSTOMER in all levels of the data-flow diagrams; or here can be only one process named CALCULATE OVERTIME among all levels of data-flow diagrams.

No Cross Line in DFD

- ✓ One way to achieve this is to restrict the number of processes in a data-flow diagram.
- ✓ On lower-level data-flow diagrams with multiple processes, one should not have more than one process symbols.
- ✓ Another way to avoid crossing lines is to duplicate an external entity or data store. Use a special notation such as an asterisk, to denote the duplicate symbol.

Numbering Convention:

- ✓ Use a unique refrence number for each process symbol..
- ✓ Other process numbers ase in the hierarchy of:

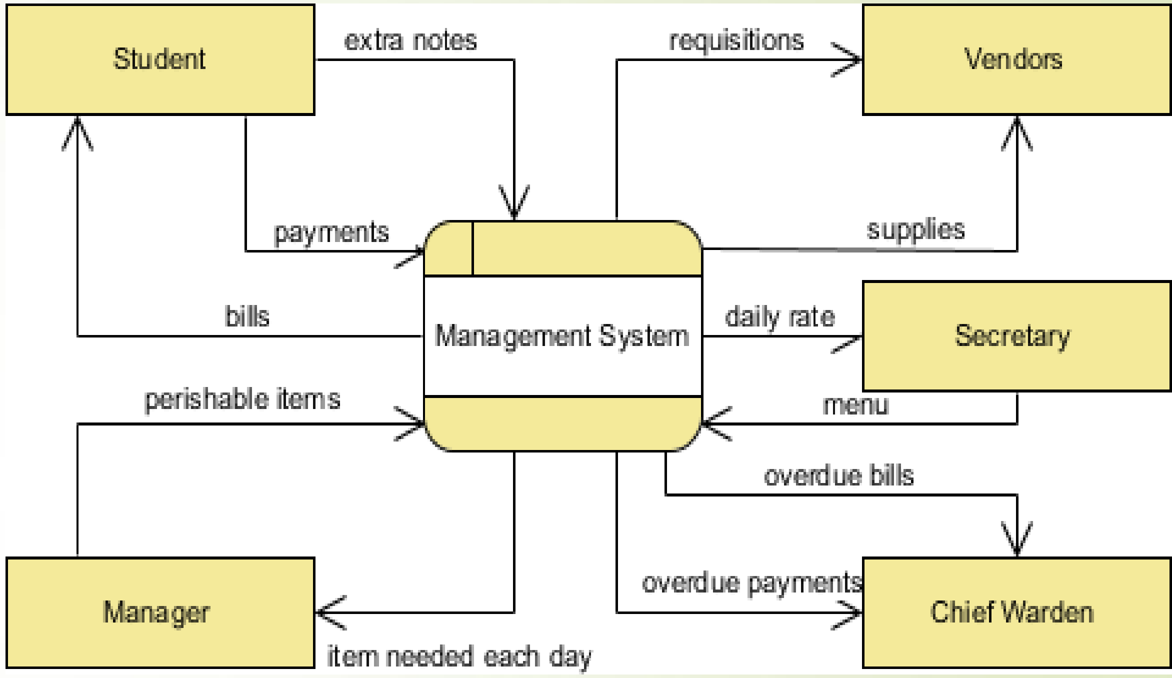
(1,2,3,...);

(1.1, 1.2,..., 2.1, 2.2, ...);

(1.1.1, 1.1.2,...);

Context-Level Diagram

- A context diagram gives an overview and it is the highest level in a data flow diagram, containing only one process representing the entire system. It should be split into major processes which give greater detail and each major process may further split to give more detail.
- All external entities are shown on the context diagram as well as major data flow to and from them.
- The diagram does not contain any data storage.
- The single process in the context-level diagram, representing the entire system, can be exploded to include the major processes of the system in the next level diagram, which is termed as diagram 0.

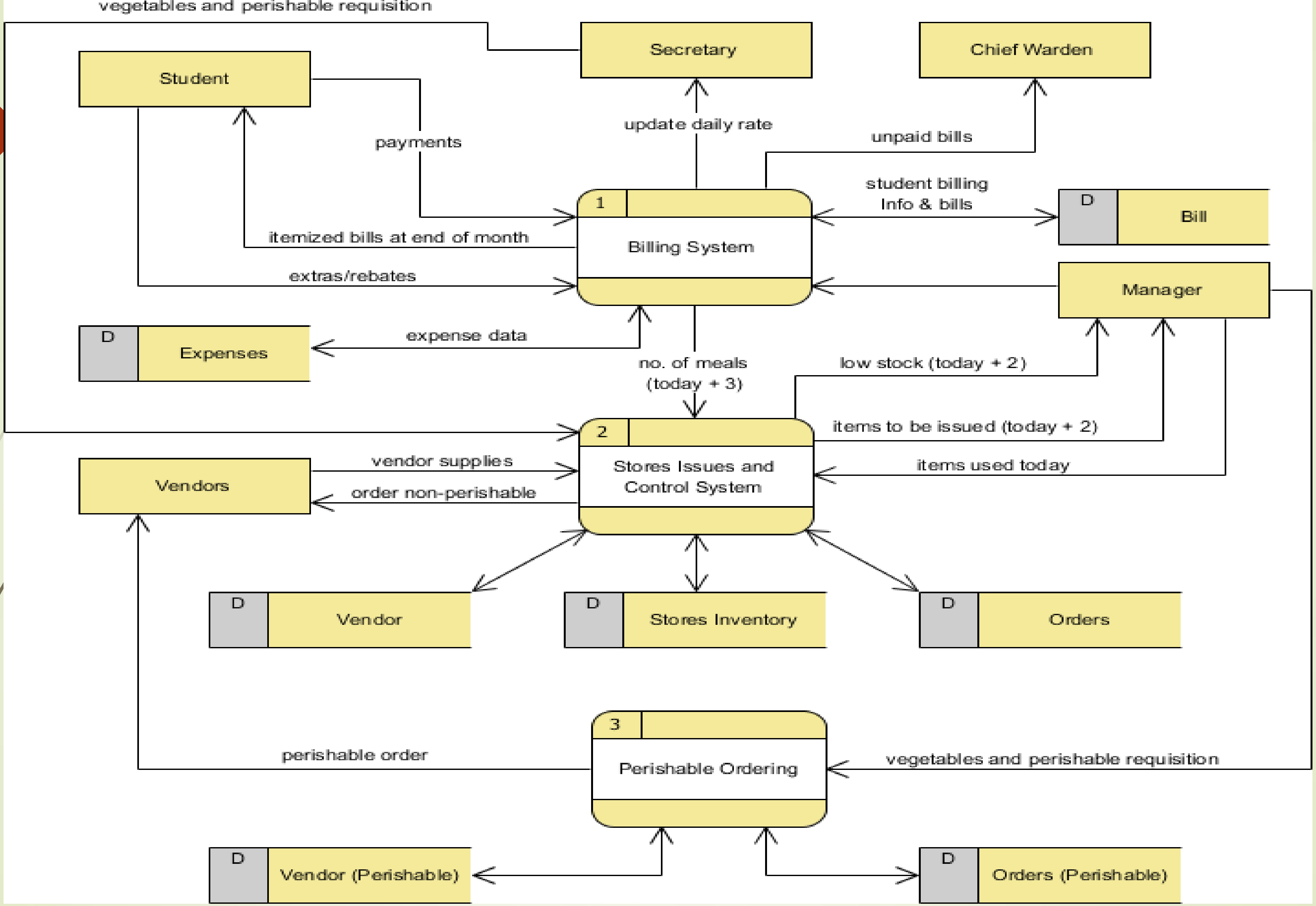


Level 1 DFD

- Processes in diagram 0 (with a whole number) can be exploded further to represent details of the processing activities. Example below shows the next level ((Diagram 1) of process explosion.

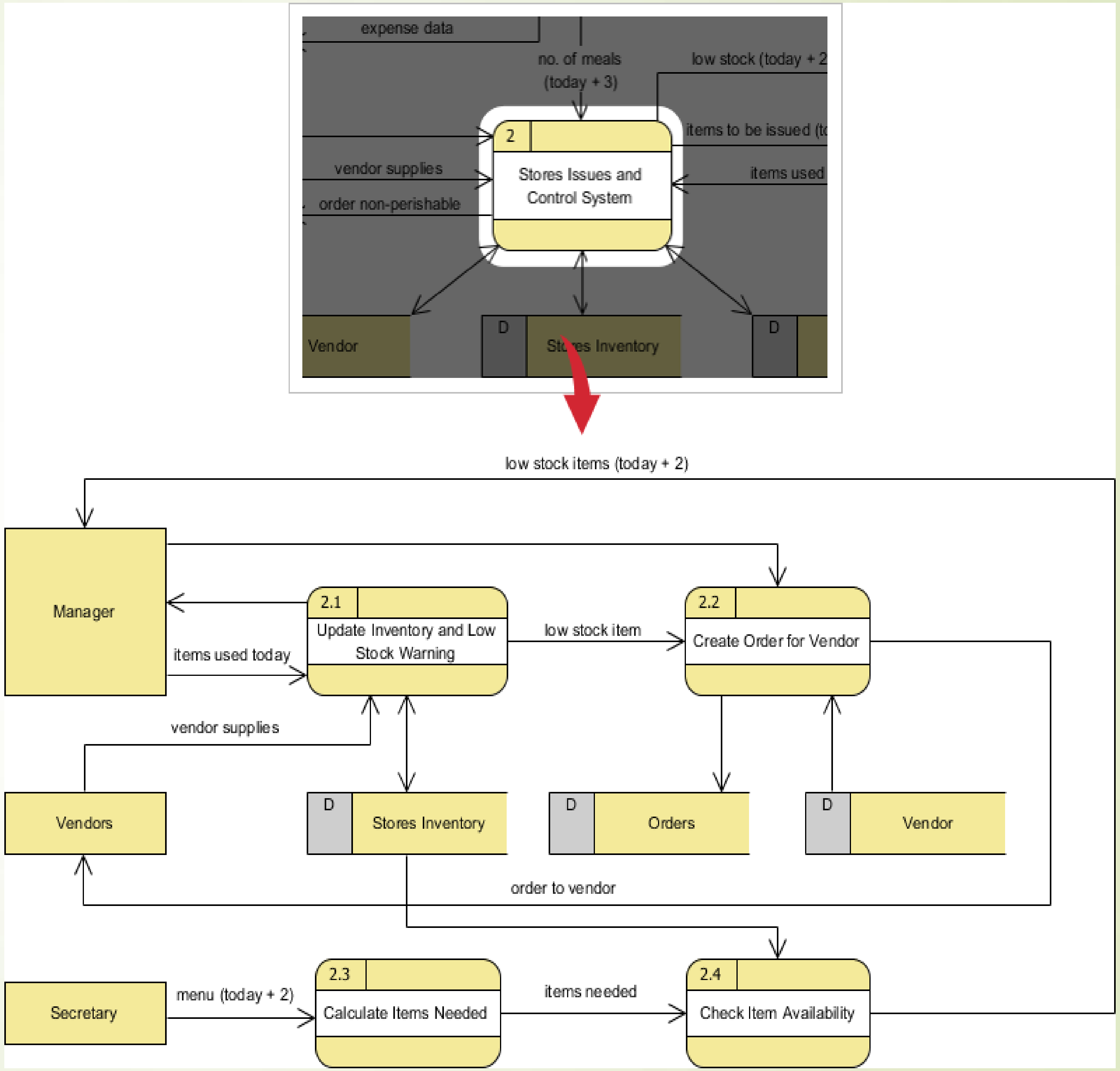
Note that:

- Although the following level 1 DFD only has three processes, there are quite a few input and input from the processes to the external entities and that could end up to be a few cross lines among them in the diagram; to avoid this problem, we could use (master and auxiliary view) multiple views of the same external entity in the DFD.


vegetables and perishable requisition
Secretary
Chief Warden
Student
payments
update daily rate
unpaid bills
student billing Info & bills
1
Billing System
D
Bill
itemized bills at end of month
extras/rebates
Manager
D
Expenses
expense data
no. of meals (today + 3)
low stock (today + 2)
items to be issued (today + 2)
2
Stores Issues and Control System
items used today
Vendors
vendor supplies
order non-perishable
D
Vendor
D
Stores Inventory
D
Orders
3
Perishable Ordering
perishable order
vegetables and perishable requisition
D
Vendor (Perishable)
D
Orders (Perishable)

Level 2 DFD

- If a process with a lot of data flow linking between a few external entities, we could first extract that particular process and the associated external entities into a separate diagram similar to a context diagram, before you refine the process into a separate level of DFD; and by this way you can ensure the consistency between them much easier.



Logical vs Physical Data Flow Diagrams

- Data flow diagrams are categorized as either logical or physical. A logical data flow diagram focuses on the business and how the business operates. It is not concerned with how the system will be constructed. We can ignore implementation specifics such as, computer configuration, data storage technology, communication or message passing methods by focusing on the functions performed by the system, such as, data collection, data to information transformation and information reporting.
- A physical data flow diagram shows how the system will be implemented, including the hardware, software, files, and people in the system. It is developed such that the processes described in the logical data flow diagrams are implemented correctly to achieve the goal of the business.

Benefits of Logical Data Flow Diagram

- A logical diagram is drawn present business information and centered on business activities, which makes it an ideal communication tool when use in communicating with project users.
- Logical DFD is based on business events and independent of particular technology or physical arrangement, which makes the resulting system more stable.
- Logical DFD allows analyst to understand the business being studied and to identify the reason behind implementation plans.
- Systems implemented based on logical DFD will be easier to maintain because business functions are not subject to frequent change.
- Very often, logical DFD does not contain data stores other than files or a database, making less complex than physical DFD and is easier to develop.
- Physical DFD can be easily formed by modifying a logical DFD.

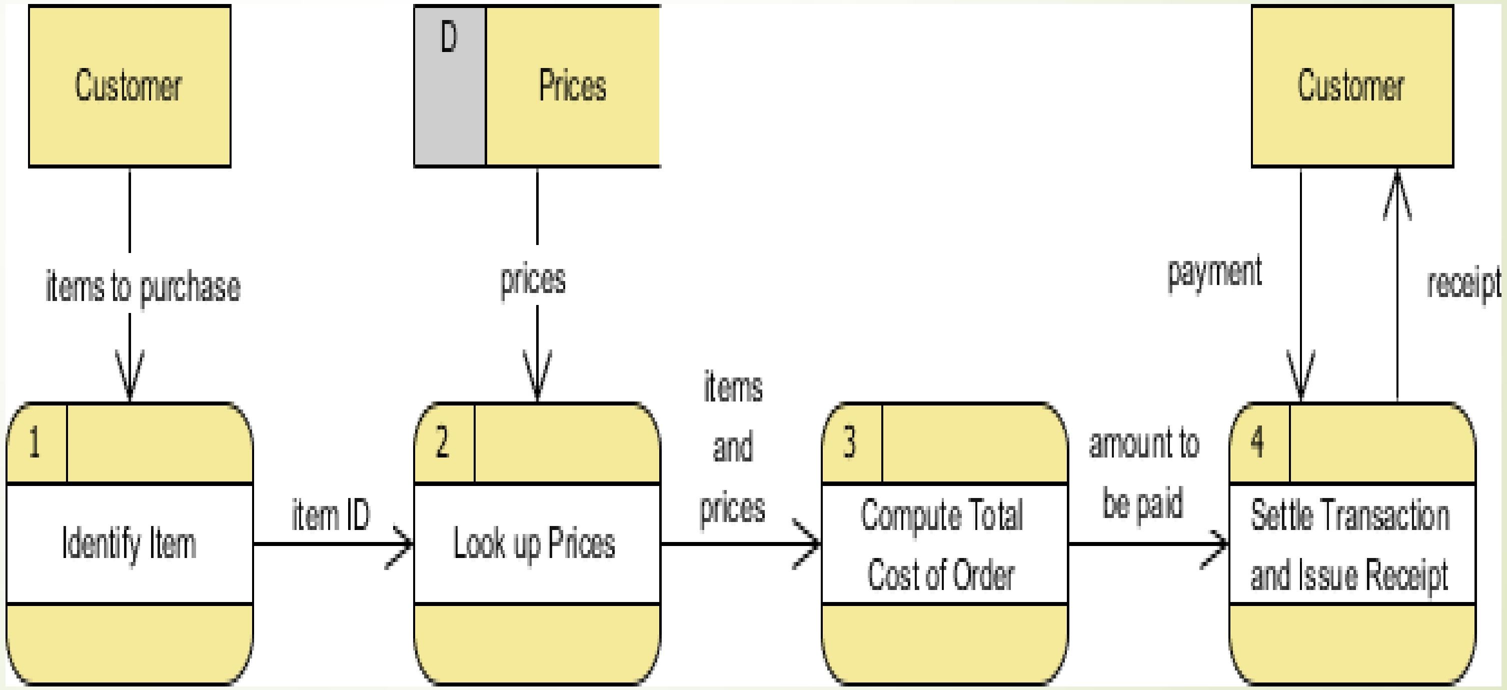
Benefits of Physical Data Flow Diagram

- Clarifying which processes are manual and which are automated: Manual processes require detailed documentation and automated process require computer programs to be developed.
- Describing processes in more detail than do logical DFDs: Describes all steps for processing of data.
- Sequencing processes that have to be done in a particular order: Sequence of activities that lead to a meaningful result are described. For example, update must be performed before a producing a summary report.
- Identifying temporary data storage: Temporary storage such as a sales transaction file for a customer receipt (report) in a grocery store, are described.
- Specifying actual names of files and printouts: Logical data flow diagrams describes actual filenames and reports, so that the programmers can relate those with the data dictionary during the developmental phase of the system.
- Adding controls to ensure the processes are done properly: These are conditions or validations of data that are to be met during input, update, delete, and other processing of data.

Refining Physical DFD for Logical DFD

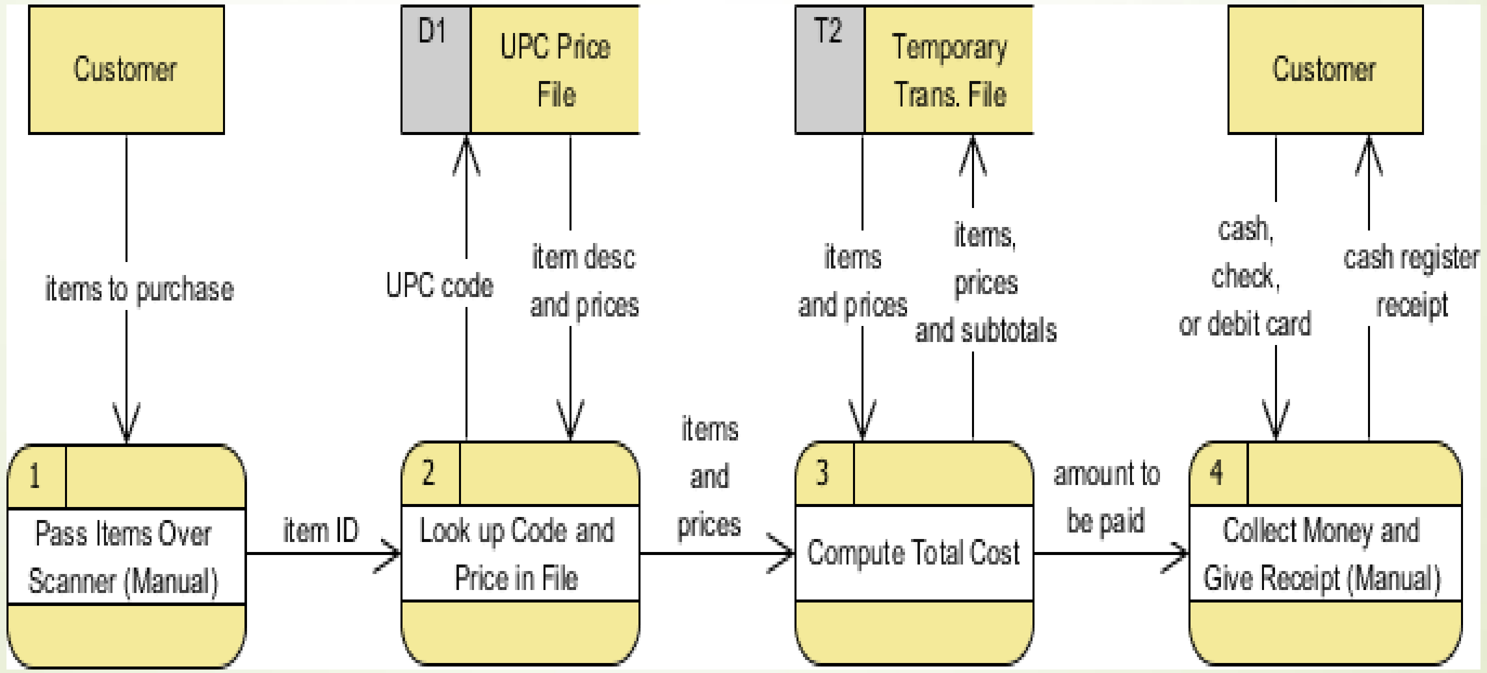
- The example below shows a logical DFD and a physical DFD for a grocery store cashier:
- The CUSTOMER brings the ITEMS to the register;
- PRICES for all ITEMS are LOOKED UP, and then totaled;
- Next, PAYMENT is given to the cashier finally, the CUSTOMER is given a receipt.

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



Physical DFD Example - Grocery Store

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



Logic Modeling:

Logic modeling is a graphic representation for the shared relationships among the resources, activities, outputs, outcomes, and impact for our program. Logic model has four components:

1. **Needs**: Needs are about the problem and why it's important to address at.

2. **Inputs**: These are the things that contribute to addressing the need.

3. **Activities**: These describe the things that the inputs allow to happen.

4. **Outcomes** :are usually expressed in terms of measures of success.

Modeling Logic With Decision Tables

- Decision tables are a concise/short/brief visual representation for specifying which actions to perform depending on given conditions.
- They are algorithms whose output is a set of actions.
- A decision table is an excellent tool to use in both testing and requirements management.
- Decision Table is a structured exercise to formulate requirements when dealing with complex business rules.
- Decision Tables are used to model complicated logic.

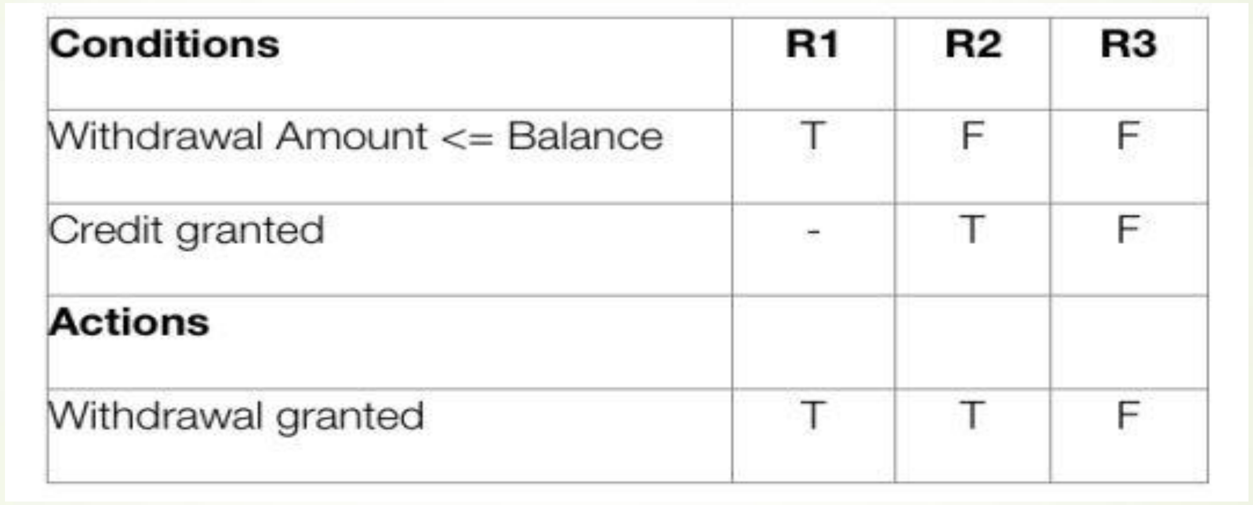
Example:

Lets take an example scenario for an ATM where a decision table would be of use.

A customer requests a cash withdrawal. One of the business rules for the ATM cash is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted.

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

- This simple example of a business rule is quite complicated to describe in text. A decision table makes the same requirements clearer to understand:
- In a decision table, conditions are usually expressed as true (T) or false (F). Each column in the table corresponds to a rule in the business logic that describes the unique combination of circumstances that will result in the actions.



Conditions	**R1**	**R2**	**R3**
Withdrawal Amount <= Balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

Steps to create decision tables:

Step1: Analyze the requirement and create the first column. Express conditions and resulting actions in a list so that they are either **TRUE** or **FALSE.**

Step2: Add Columns by calculating how many columns are needed in the table. The number of column's depends on number of conditions . Mathematically, no. of columns is $\mathbf{2^{condition}}$ **.**

Now, T (True) and F(False) for conditions are filled.

Steps 3: Reduce the table by deleting the columns that have become identical and mark Insignificant values with**"-"**.

Step4: Determine and enter actions for each column in the table.

Step5: White test cases based on the table.

Advantages of decision table:

- ✓ They are easier to draw.
- ✓ Decision table can be easily changed according to the situation.
- ✓ A small table can replace several pages of flow chart.
- ✓ The requirements become much clearer.
- ✓ Decision tables make it possible to detect combinations of conditions.

Disadvantages of decision table:

- ✓ When there are two many alternatives, decision table cannot list them all.
- ✓ It can not express complete sequence of operations to solve a problem
- ✓ They only present partial solution

Self Study

1. Structured English
2. Decision Trees
3. State-transition diagrams
4. Sequence diagrams
5. Activity diagrams
6. Pseudo code

Structuring System Data Requirements:

Structuring system and data requirements concentrates structure and relationships within data. The most Common format used for data modeling is entity-relationship diagramming.

Conceptual Data modeling:

- A conceptual data model is a map of concepts and their relationships used for databases. It is a representation of organizational data.
- Conceptual data modeling is typically done in parallel with other requirements analysis and structuring steps during system analysis. On larger systems a subset of project team concentrates on data modeling while other team members focus attention on process or logic modeling. Analysts develop a conceptual data model for current system.

Process: This process usually begins with developing a conceptual data model for the existing system. Then a new Conceptual Data model, which includes all of the data requirements for the new system is built. E-R diagrams can be translated into wide variety of technical architectures for data, such as relational, network and hierarchical.

Deliverables and Outcomes:

- The primary deliverable from the conceptual data modeling is ER diagram .Another deliverable from CDM is the full set of entries about data object to be stored in the project repository.

Gathering Information for Conceptual Data Modeling:

- A data model explains what the organization does and what rules govern how work is performed in the organization.
- The first perspective is generally called the **top-down** approach. This perspective derives the business rules for a data model from an intimate understanding of the nature of the business, rather than from any specific information requirements in computer displays, reports or business forms.
- The bottom-up approach is on the contrary a process of gaining an understanding of data by reviewing specific business documents handled within the system.

ER Diagram

- An Entity Relationship Diagram (ER Diagram or ERD) is a visual representation of the structure of a database, showing how entities (like tables) and their attributes (like columns) relate to each other.
- It is a high-level data model that defines data elements and their relationship for a specified software system. An ER model is used to represent real-world objects.

Elements of ER Diagram/ER Design Issues.

- ***Entity:*** Entity is a thing or object in the real world with an independent existence. An entity may be an object with a physical existence or it may be an object with conceptual existence.
- ***Entity type:*** It is a collection of entities having common attribute. Eg; student is an entity type.
- ***Entity set:*** It is same as an entity type, but defined, at a particular point in time, such as students enrolled on class on the first day.

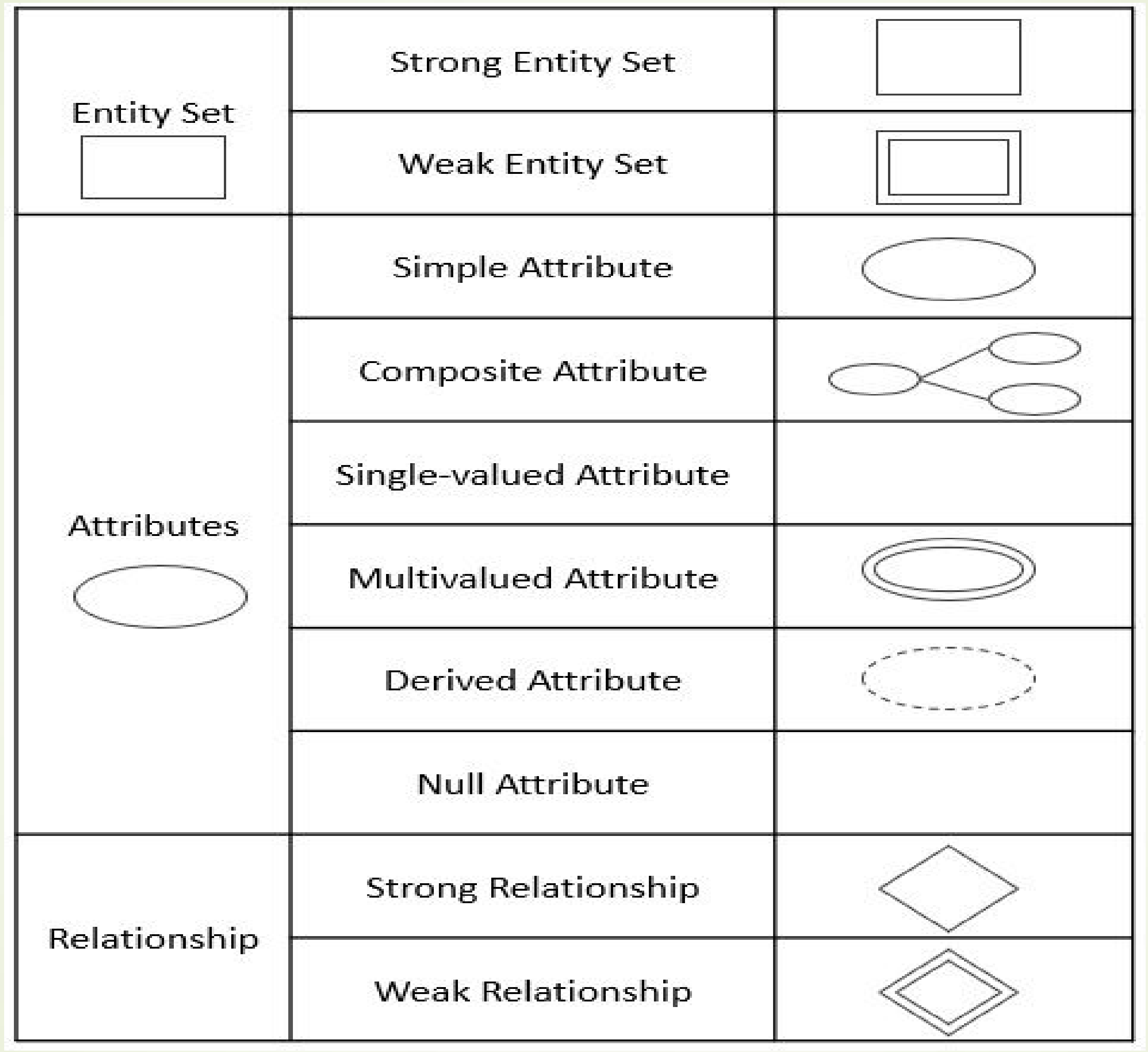
Attributes: The particular, properties that describe entity are known as attributes. For example., an. EMPLOYEE entity may be described by the employee's name, age, address, salary, job etc.

Types of attributes:

Atomic vs, composite :An attribute that cannot be divided into smaller independent attribute is known as atomic attribute and attribute that can be divided are composite.

Single vs. Multi valued :An attribute having only single value for an entity is single valued attribute and that have multiple value for entity is multivalued.

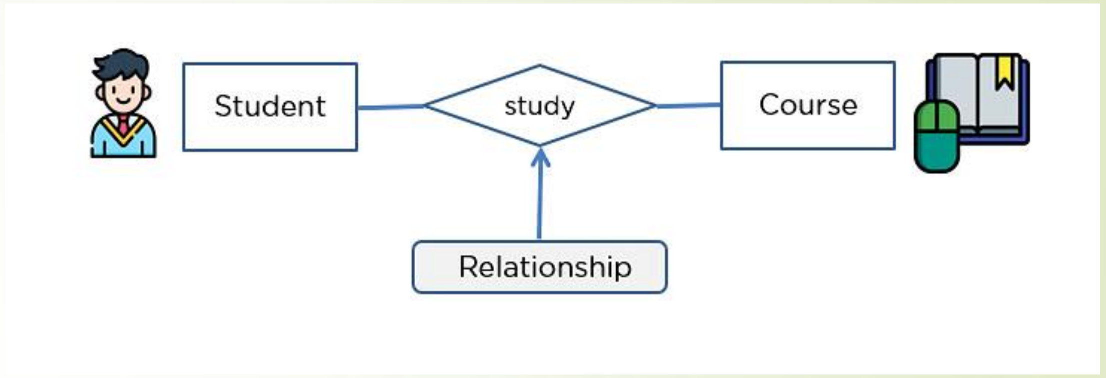
Key attribute :An attribute that has unique value of each enty is known as key attribute, For eg, every student has unique roll number in the class.

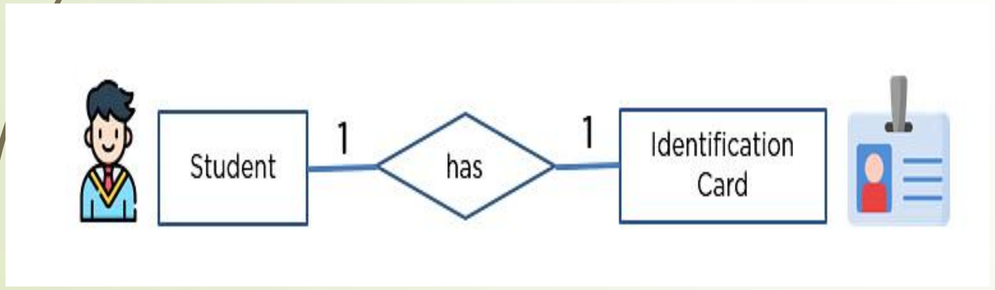


Entity Set ▭	Strong Entity Set	▭
	Weak Entity Set	⧈
Attributes ⬭	Simple Attribute	⬭
	Composite Attribute	⬭<⬭⬭
	Single-valued Attribute	
	Multivalued Attribute	◎
	Derived Attribute	⬯
	Null Attribute	
Relationship	Strong Relationship	◇
	Weak Relationship	◈

Relationship

- The diamond shape showcases a relationship in the ER diagram.
- It depicts the relationship between two entities.
- In the example below, both the student and the course are entities, and study is the relationship between them.



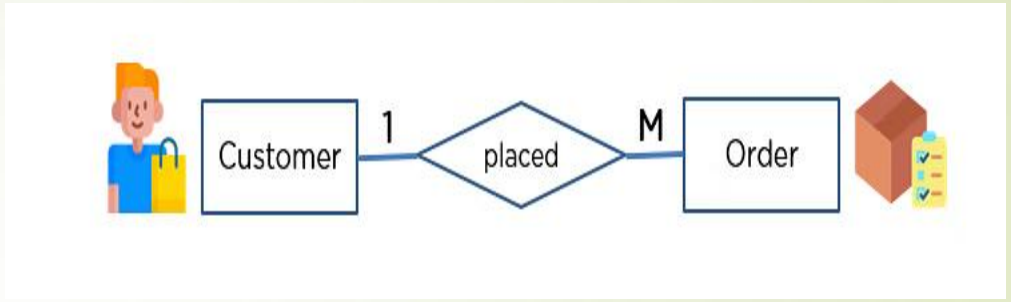


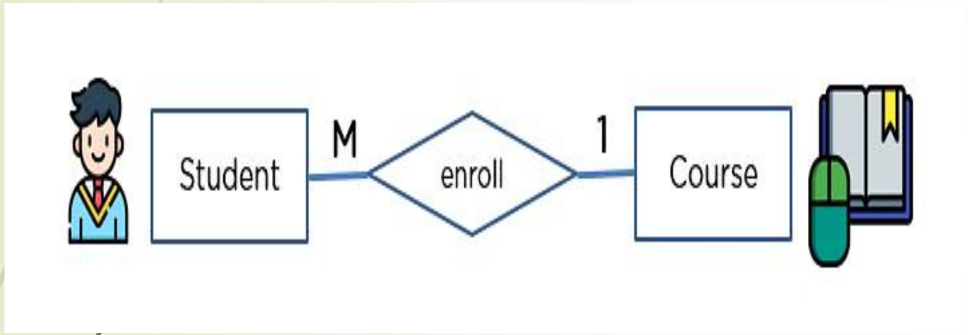
One-to-One Relationship

- When a single element of an entity is associated with a single element of another entity, it is called a one-to-one relationship.
- For example, a student has only one identification card and an identification card is given to one person.

One-to-Many Relationship

- ➢ When a single element of an entity is associated with more than one element of another entity, it is called a one-to-many relationship
- ➢ For example, a customer can place many orders, but an order cannot be placed by many customers.



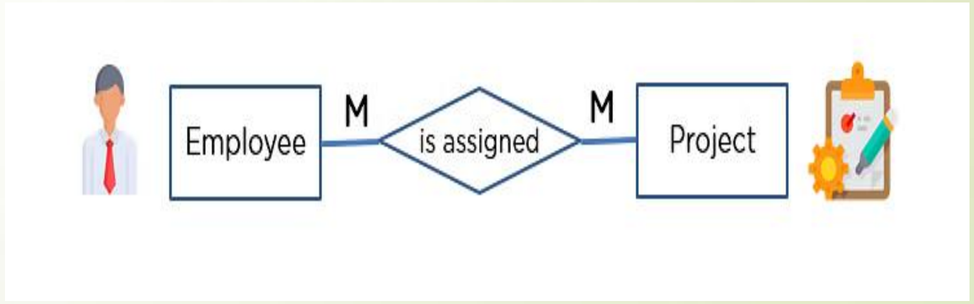


Many-to-One Relationship

- ✓ When more than one element of an entity is related to a single element of another entity, then it is called a many-to-one relationship.
- ✓ For example, students have to opt for a single course, but a course can have many students.

Many-to-Many Relationship

- ✓ When more than one element of an entity is associated with more than one element of another entity, this is called a many-to-many relationship.
- ✓ For example, you can assign an employee to many projects and a project can have many employees.



How to Draw an ER Diagram?

- ✓ First, identify all the Entities. Embed all the entities in a rectangle and label them properly.
- ✓ Identify relationships between entities and connect them using a diamond in the middle, illustrating the relationship. Do not connect relationships with each other.
- ✓ Connect attributes for entities and label them properly.
- ✓ Eradicate any redundant entities or relationships.
- ✓ Make sure your ER Diagram supports all the data provided to design the database.
- ✓ Effectively use colors to highlight key areas in your diagrams.

Practice different ER diagram from class work and homework

Representing Supertypes And Subtypes

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

.

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

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

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

3. Domains: Constraints on valid values for attributes.

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

Domains: A domain is the set of all data types and ranges of values that attributes may assume.

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

Role of Packaged Conceptual Data Models:

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

Universal Data Models :

Numerous core subject areas are common to many (or even most) organizations, such as customers, products, accounts, documents, and projects. Although they differ in detail, the underlying data structures are often quite similar for these subjects. Further, there are core business functions such as purchasing, accounting, receiving, and project management that follow common patterns. Universal data models are templates for one or more of these subject areas and/or functions. All of the expected components of data models are generally included: entities, relationships, attributes, primary and foreign keys, and even sample data.

Industry-Specific Data Models

Industry-specific data models are generic data models that are designed to be used by organizations within specific industries. Data models are available for nearly every major industry group, including health care, telecommunications, discrete manufacturing, process manufacturing, banking, insurance, and higher education. These models are based on the premise that data model patterns for organizations are very similar within a particular industry ("a bank is a bank"). However, the data models for one industry (such as banking) are quite different from those for another (such as hospitals).