Process Modelling

Objectives

- Define systems modeling and differentiate logical and physical models.
- Define process modeling and explain its benefits.
- Recognize and understand basic concepts and constructs of a process model.
- Read and interpret a data flow diagram.
- Explain when to construct process models and where to store them.
- Construct a context diagram to illustrate a system's interfaces with its environment. Identify use cases, external and temporal business events.
- Perform event partitioning and organize events in a functional decomposition diagram.
- Draw event diagrams and merge them into a system diagram.
- Draw primitive data flow diagrams and describe the elementary data flows in terms of data structures and procedural logic.
- Document the distribution of processes to locations.
- Synchronize data and process models using a CRUD matrix.

Introduction

- A **model** is a pictorial representation of reality. Just as a picture is worth a thousand words, most models are pictorial representations of reality.
- **Logical model** is a nontechnical pictorial representation that depicts what a system is or does. Synonyms or *essential model*, *conceptual model*, and *business model*.
- **Physical model** is a technical pictorial representation that depicts what a system is or does and how the system is implemented. Synonyms are *implementation model* and *technical model*.

Importance of Logical System Models

- Logical models remove biases that are the result of the way the system is currently implemented, or the way that any one person thinks the system might be implemented.
- Logical models reduce the risk of missing business requirements because we are too preoccupied with technical results.
- Logical models allow us to communicate with end-users in nontechnical or less technical languages.

Process Modelling and Data Flow Diagrams (DFDs)

- **Process modeling** is a technique used to organize and document a system's processes.
- **Process modelin**g can also be defined as a technique for organizing and documenting the structure and flow of data through a system's PROCESSES and/or the logic, policies, and procedures to be implemented by a system's PROCESSES.
- There various tools used to model system processes such as program structure charts, logic flowcharts, decision tables and data flow diagrams.

Data flow diagram (DFD)

- DFD is a process model used to depict the flow of data through a system and the work or processing performed by the system.
- The DFD has also become a popular tool for business process redesign.

Differences between DFDs and Flowcharts

- Processes on DFDs can operate in parallel (at-the-same-time) while Processes on flowcharts execute one at a time
- DFDs show the flow of data through a system whereas flowcharts show the flow of control (sequence and transfer of control)
- Processes on a DFD can have dramatically different timing (daily, weekly, on demand) whereas processes on flowcharts are part of a single program with consistent timing

DFD Components

External Agents

An **external agent** or an external entity is an outside person, unit, system, or organization that interacts with a system.

- External agents define the "boundary" or scope of a system being modeled.
- As scope changes, external agents can become processes, and vice versa.
- An external agent can be any of the following: − Office, department, division.
  - An external organization or agency.
  - Another business or another information system.
  - One of system's end-users or managers
- External agents are named with descriptive, singular noun

Gane and Sarson shape

DeMarco/Yourdon shape

Data Stores

- **Data store** is stored data intended for later use. A data store can be a file or a database
- A data store is "data at rest" compared to a data flow that is "data in motion.
- Data stores depicted on a DFD store all instances of data entities (depicted on an ERD)
- A data store is named with plural noun

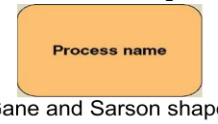
Gane and Sarson shape

DeMarco/Yourdon shape

Process Concepts

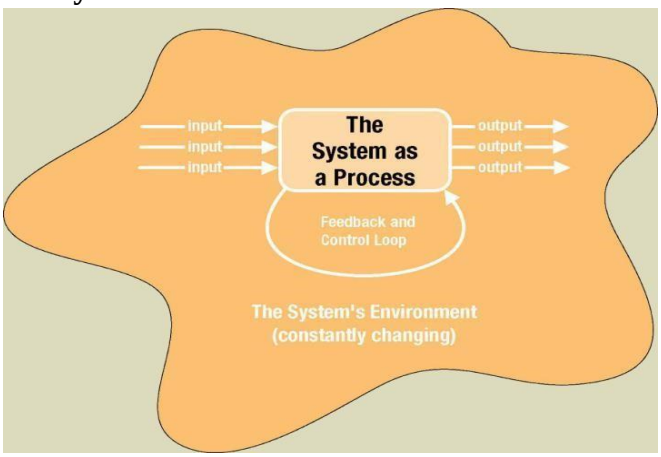
- **Process** is work performed by a system in response to incoming data flows or conditions.

- All information systems include processes
- Processes respond to business events and conditions and transform data into useful information
- Modeling processes helps us to understand the interactions with the system's environment, other systems, and other processes.
- Processes are named with a strong action verb followed by object clause describing what the work is performed on/for.



Gane and Sarson shape

The System is itself a Process

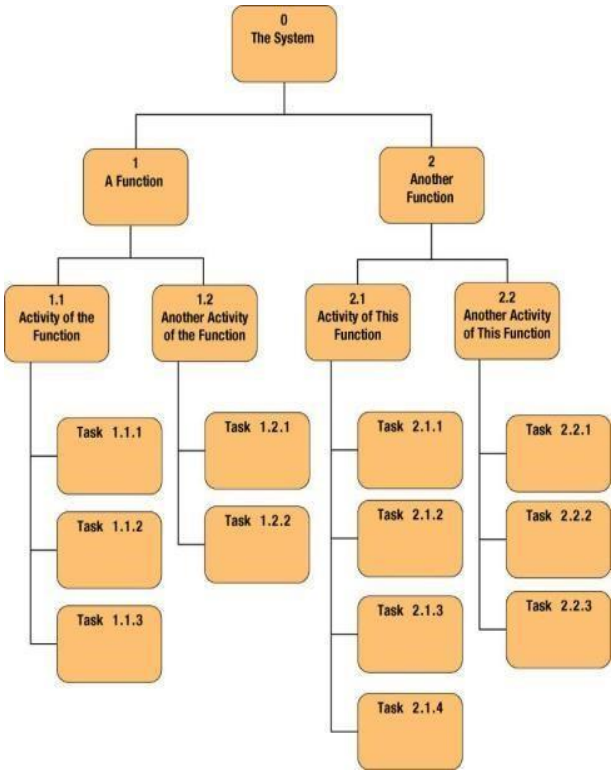


Process Decomposition

- Decomposition is a top-down problem-solving approach.
- **Decomposition** is the act of breaking a system into sub-components. Each level of abstraction reveals more or less detail.

Decomposition Diagram (Hierarchy Chart)

Decomposition diagram is a tool used to depict the decomposition of a system.

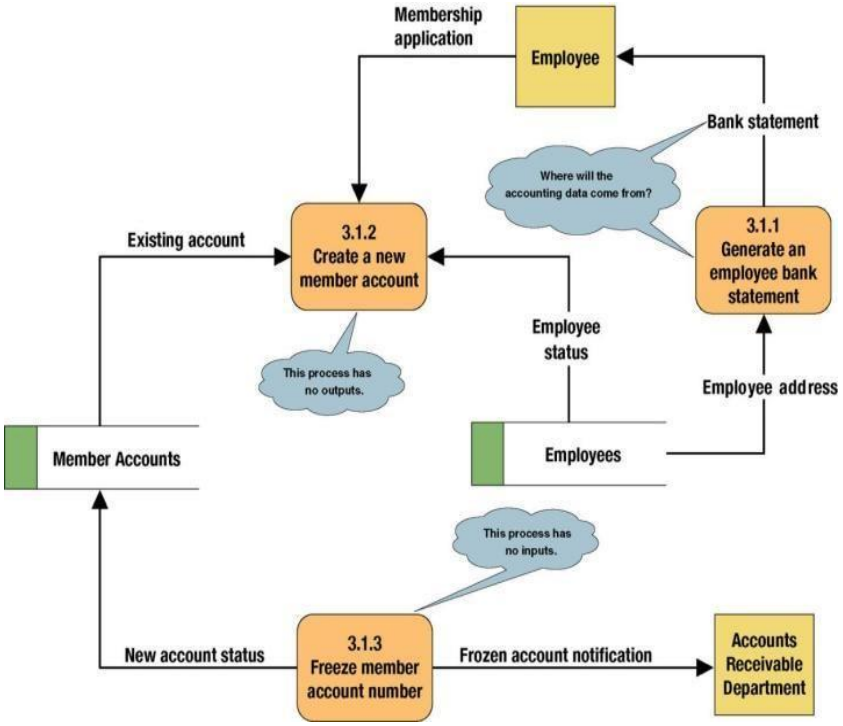


Types of Logical Processes

1. **Function** is a set of related and ongoing activities of a business. A function has no start or end.
2. **Event** or transaction is a logical unit of work that must be completed as a whole.
   - An event is triggered by a discrete input and is completed when process has responded with appropriate outputs.
   - Functions consist of processes that respond to events.
3. **Elementary process** or primitive event is a discrete, detailed activity or task required to complete the response to an event.

- Elementary process is the lowest level of detail depicted in a process model.

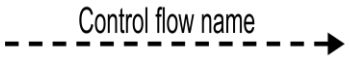
Common Process Errors on DFDs



Data Flows & Control Flows

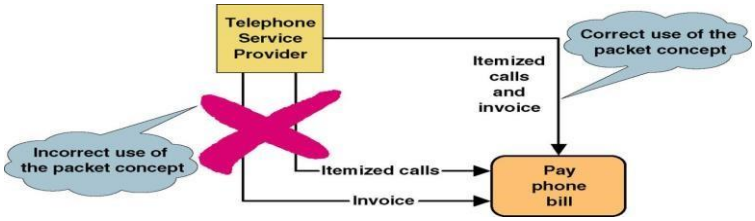
- **Data flow** is data that is input to or output from a process.
  - A data flow is data in motion
  - A data flow may also be used to represent the creation, reading, deletion, or updating of data in a file or database (called a data store).
- **Composite data flow** is a data flow that consists of other data flows.

- **Control flow** is a condition or non-data event that triggers a process. This is used sparingly on DFDs.

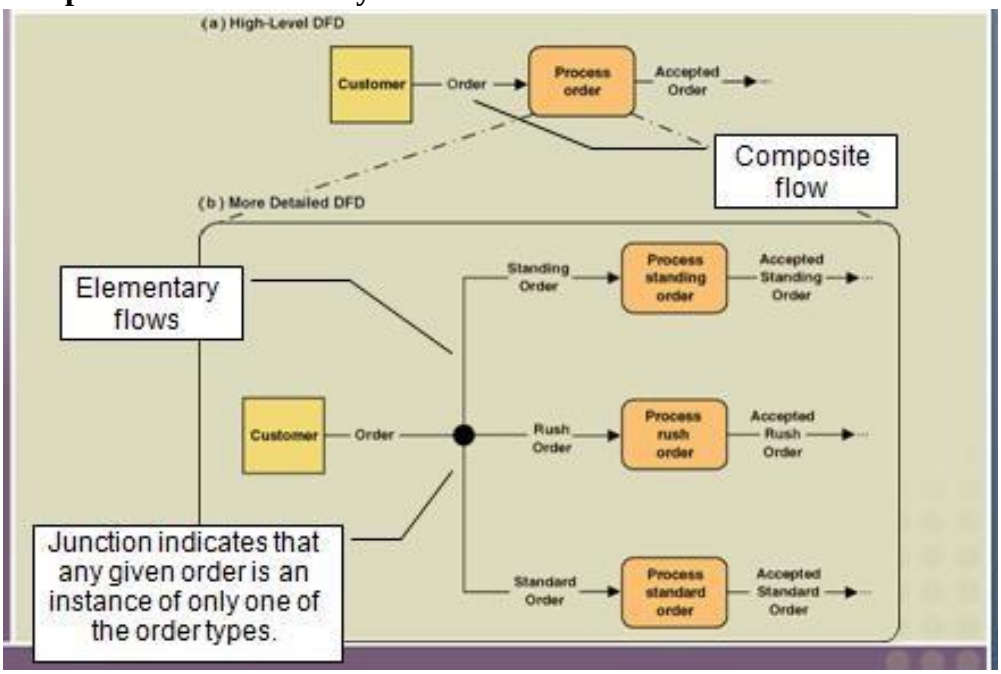


Data Flow Packet Concept

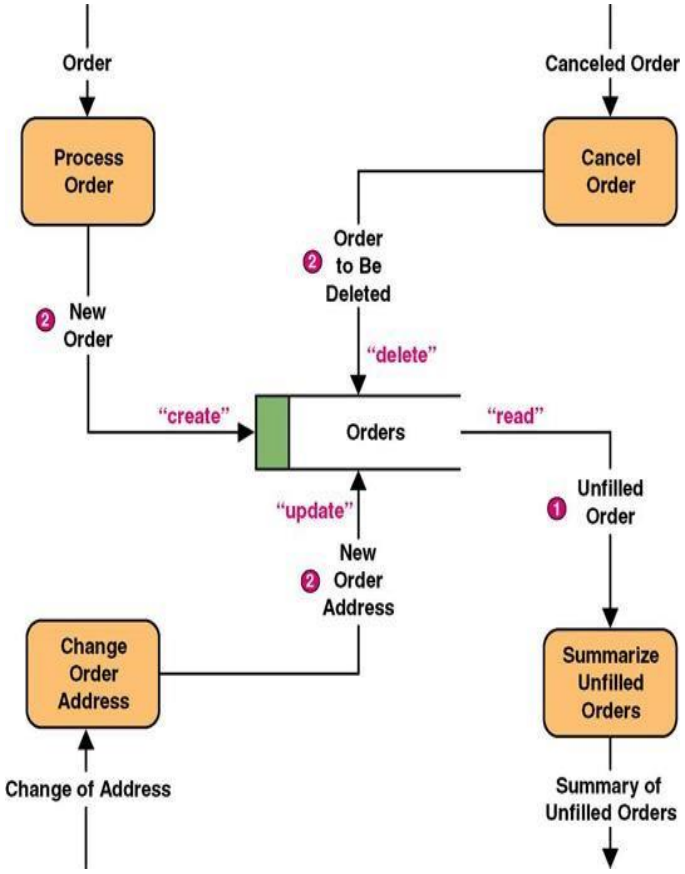
• Data that should travel together should be shown as a single data flow, no matter how many physical documents might be included.



Composite and Elementary Data Flows



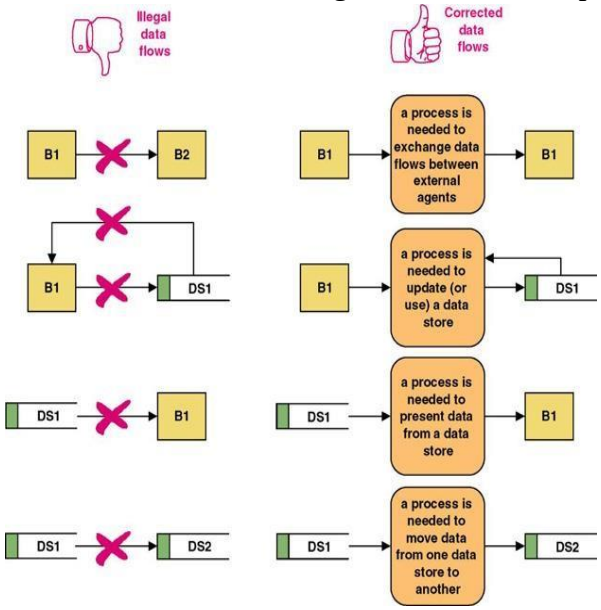
Data Flows to and from Data Stores



Rules for Data Flows

- A data flow should never go unnamed.

- In logical modeling, data flow names should describe the data flow without describing the implementation
- All data flows must begin and/or end at a process.



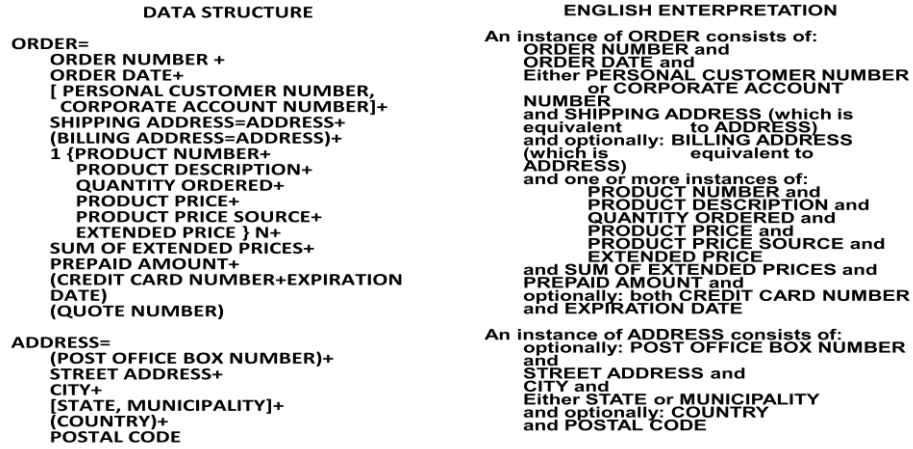
Data Conservation

- **Data conservation** (*starving the processes)* is the practice of ensuring that a data flow contains only data needed by the receiving process.
- The emphasis of data conservation on business process redesign is to identify and eliminate inefficiencies and to simplify the interface between those processes.
- You must precisely define the data composition of each data flow, expressed in the form of *data structures*.

Data Structures

- **Data attribute** is the smallest piece of data that has meaning to the users and the business.
- **Data structure** is a specific arrangement of data attributes that defines an instance of a data flow
- The data attributes that comprise a data flow are organized into data structures.
- Data flows can be described in terms of the following types of data structures: − A *sequence* or group of data attributes that occur one after another.
  - The *selection* of one or more attributes from a set of attributes.
  - The *repetition* of one or more attributes.

Data Structure for a Data Flow



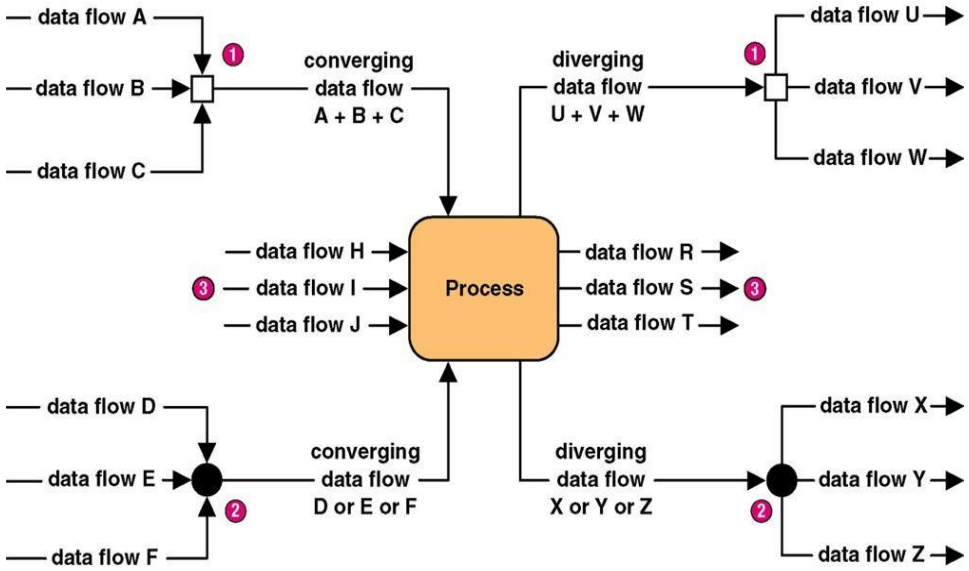
Data Structure Constructs

Data Structure	**Format by Example** (relevant portion is **boldfaced**	**English Interpretation** (relevant portion is **boldfaced**)
Sequence of Attributes - The sequence data structure indicates one or more attributes that may (or must) be included in a data flow.	WAGE AND TAX STATEMENT= **TAXPAYER IDENTIFICATION NUMBER+** **TAXPAYER NAME+** **TAXPAYER ADDRESS+** **WAGES, TIPS, AND COMPENSATION+** **FEDERAL TAX WITHHELD+…**	An instance of WAGE AND TAX STATEMENTS consists of: **TAXPAYER IDENTIFICATION NUMBER and** **TAXPAYER NAME and** **TAXPAYER ADDRESS and** **WAGES, TIPS AND COMPENSATION and** **FEDERAL TAX WITHHELD and…**
Selection of Attributes - The selection data structure allows you to show situations where different sets of attributes describe different instances of the data flow.	ORDER= **(PERSONAL CUSTOMER NUMBER,** **CORPORATE ACCOUNT NUMBER)+** ORDER DATE+…	An instance or ORDER consists of: **Either PERSONAL CUSTOMER NUMBER or CORPORATE ACCOUNT NUMBER;** and ORDER DATE and…
Repetition of Attributes - The repetition data structure is used to set off a data attribute or group of data attributes that may (or must) repeat themselves a specific number of time for a single instance of the data flow. The minimum number of repetitions is usually *zero* or *one*. The maximum number of repetitions may be specified as "n" meaning "many" where the actual number of instances varies for each instance of the data flow.	POLICY NUMBER+ POLICYHOLDER NAME+ POLICY HOLDER ADDRESS+ 0 **{DEPENDENT NAME+** **DEPENDENT'S RELATIONSHIP} N+** 1 **{EXPENSE DESCRIPTION+** **SERVICE PROVIDER+** **EXPENSE AMOUNT} N**	An instance of CLAIM consists of: POLICY NUMBER and POLICYHOLDER NAME and POLICYHOLDER ADDRESS and **zero or more instance of:** **DEPENDENT NAME and** **DEPENDENT'S RELATIONSHIP and** **one or more instances of:** **EXPENSE DESCRIPTION and** **SERVICE PROVIDER and** **EXPENSE ACCOUNT**

Optional Attributes - The optional notation indicates that an attribute, or group of attributes in a sequence or selection date structure may not	CLAIM= POLICY NUMBER+ POLICYHOLDER NAME+ POLICYHOLDER ADDRESS+	An instance of CLAIM consists of: POLICY NUMBER and POLICYHOLDER NAME and POLICYHOLDER ADDRESS
be included in all instances of a data flow. *Note: For the repetition data structure, a minimum of "zero" is the same as making the entire repeating group "optional."*	**(SPOUSE NAME+ DATE OF BIRTH)**+…	and **optionally, SPOUSE NAME and DATE OF BIRTH** and…
Reusable Attributes - For groups of attributes that are contained in many data flows, it is desirable to create a separate data structure that can be reused in other data structures.	DATE= MONTH+ DAY+ YEAR+	Then, the reusable structures can be included in other data flow structures as follows: ORDER=ORDER NUMBER…+DATE INVOICE=INVOICE NUMBER…+DATE PAYMENT=CUSTOMER NUMBER…+DATE

Diverging and Converging Data Flows

- **Diverging data flow** is a data flow that splits into multiple data flows. It indicates data that starts out naturally as one flow, but is routed to different destinations. It also useful for indicating multiple copies of the same output going to different destinations.
- **Converging data flow** is the merger of multiple data flows into a single packet. It indicates data from multiple sources that can (must) come together as a single packet for subsequent processing.



When to Draw Process Models

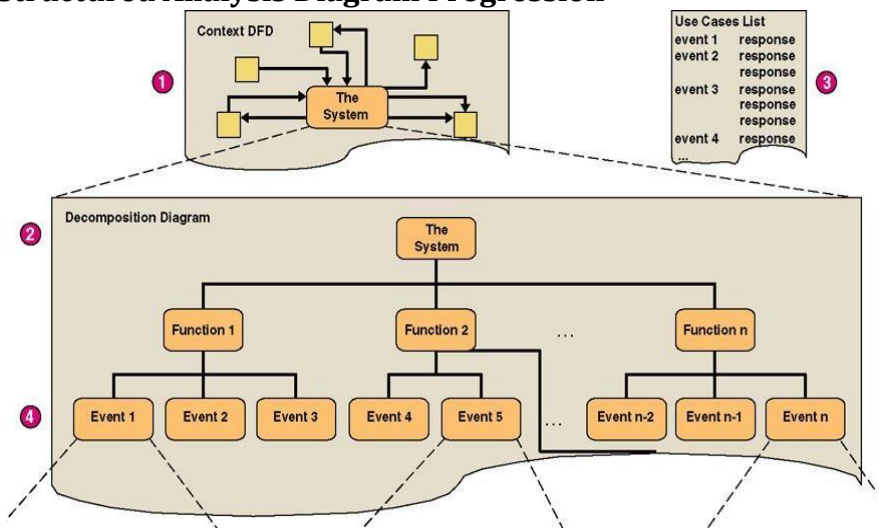
1. Strategic systems planning
    - Enterprise process models illustrate important business functions.

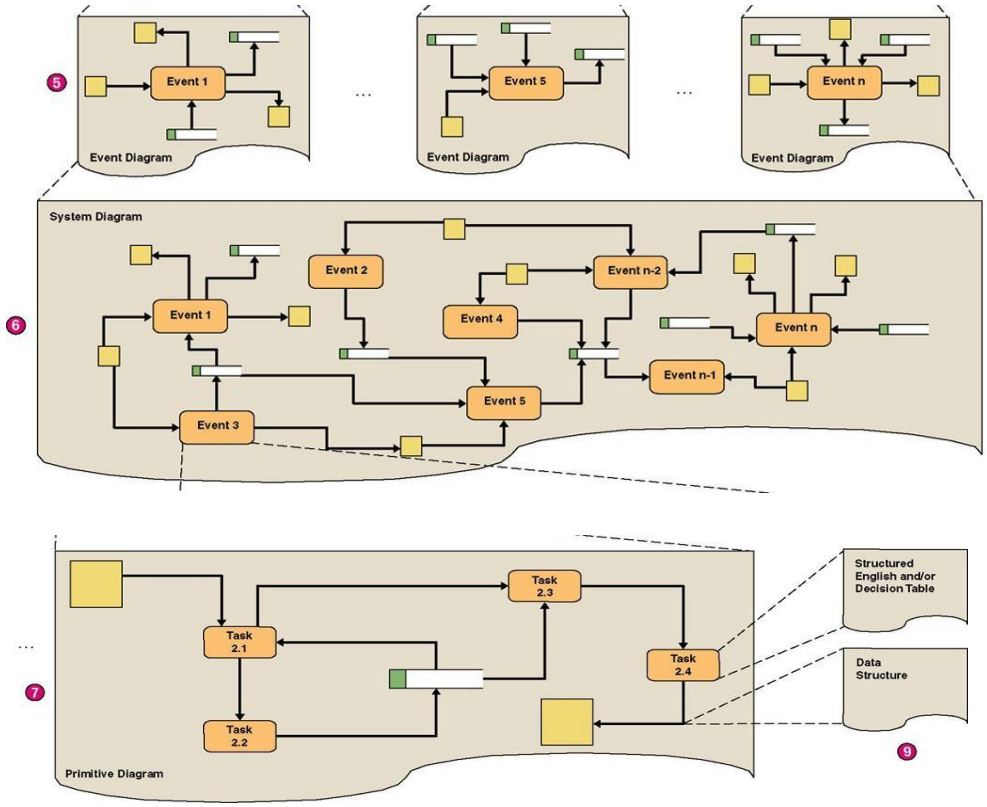
2. Business process redesign
    - "As is" process models facilitate critical analysis.
    - "To be" process models facilitate improvement.
3. Systems analysis
    - Model existing system including its limitations
    - Model target system's logical requirements
    - Model candidate technical solutions
    - Model the target technical solution

Modern Structured Analysis

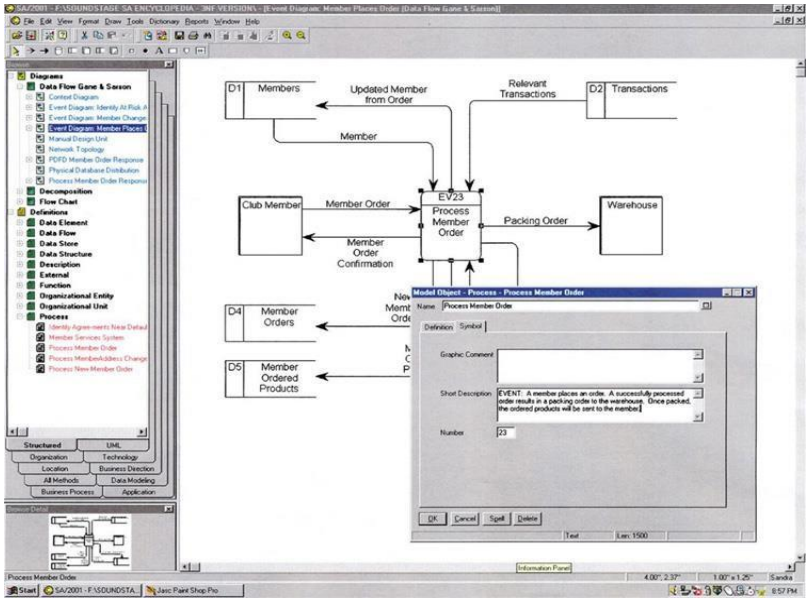
1. Draw context DFD to establish initial project scope.
2. Draw functional decomposition diagram to partition the system into subsystems.
3. Create event-response or use-case list for the system to define events for which the system must have a response.
4. Draw an event DFD (or event handler) for each event.
5. Merge event DFDs into a system diagram (or, for larger systems, subsystem diagrams).
6. Draw detailed, primitive DFDs for the more complex event handlers.
7. Document data flows and processes in data dictionary.

Structured Analysis Diagram Progression





CASE for Process Modelling

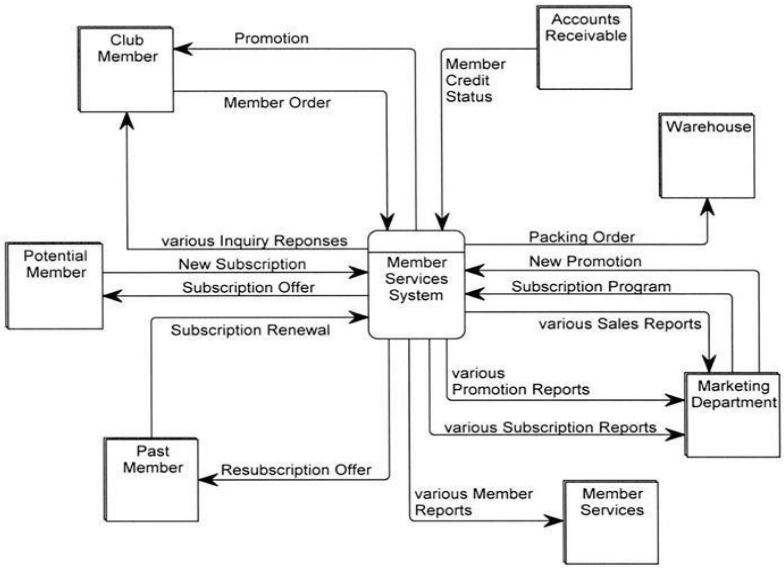


Context Data Flow Diagram

- **Context data flow diagram** (environmental model) is a process model used to document the scope for a system. Also called the. 1. Think of the system as a "black box."
  2. Ask users what business transactions the system must respond to. These are inputs, and the sources are external agents.

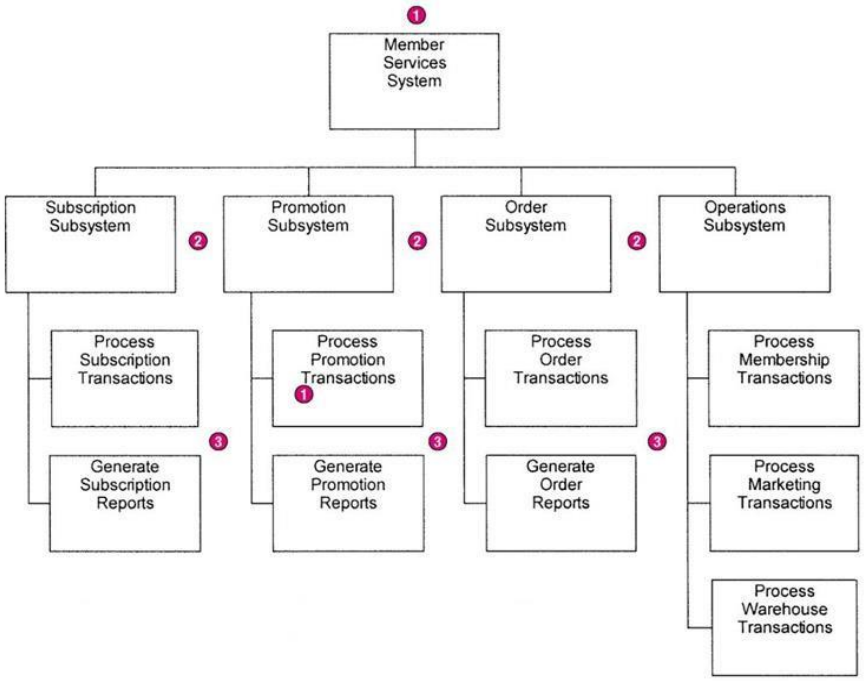
3. Ask users what responses must be produced by the system. These are outputs, and the destinations are external agents.
4. Identify any external data stores, if any.
5. Draw a context diagram.

Sample Context DFD



Sample Functional Decomposition Diagram

- Break system into sub-components to reveal more detail.
- Every process to be factored should be factored into at least two child processes.
- Larger systems might be factored into subsystems and functions.



Events and Use Cases

- Events are very similar to use cases in object-oriented analysis.

- Events are represented on DFDs as data flows (for external events) or control flows (for temporal and state events).
- **External events** are initiated by external agents. They result in an input transaction or data flow.
- **Temporal events** are triggered on the basis of time, or something that merely happens. They are indicated by a control flow.
- **State events** trigger processes based on a system's change from one state or condition to another. They are indicated by a control flow.
- **Use case** is an analysis tool for finding and identifying business events and responses.
- **Actor** is anything that interacts with a system.

Sample Use Case List

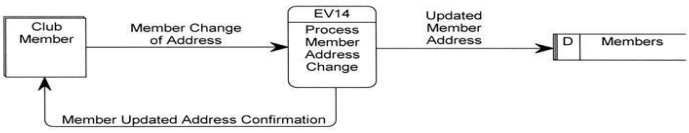
Actor/ External Agent	Event (or Use Case)	Trigger	Response
Marketing	Establishes a new membership subscription plan to entice new members.	New Member Subscription Program	Generate Subscription Plan Confirmation. Create Agreement in the database.
Marketing	Establishes a new membership resubscription plan to lure back former members.	Past Member Resubscription Program	Generate Subscription Plan Confirmation. Create Agreement in the database.
(time)	A subscription plan expires.	(current date)	Generate Agreement Change Confirmation. Logically delete Agreement in database.
Member	Joins club by subscribing.	New Subscription	Generate Member Directory Update Confirmation. Create Member in database. Create first Member Order and Member Ordered Products in database.

Event Diagrams

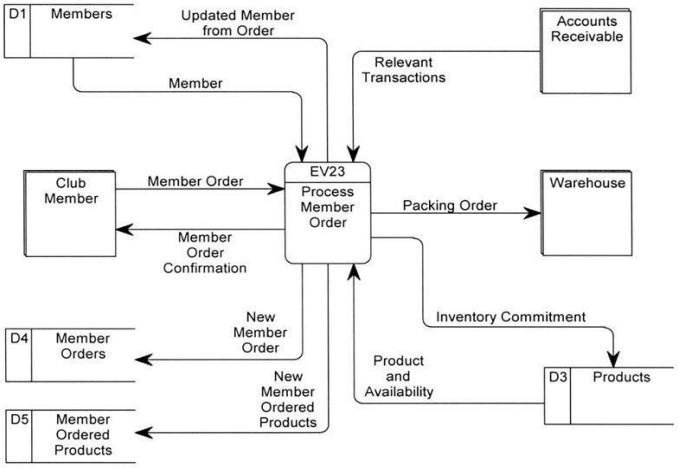
- **Event diagram** is a data flow diagram that depicts the context for a single event.
- One diagram for each event process depicts
  - Inputs from external agents
  - Outputs to external agents
  - Data stores from which records must be "read." Data flows should be added and named to reflect the data that is read.

- Data stores in which records must be created, deleted, or updated. Data flows should be named to reflect the update.

Sample Event Diagrams

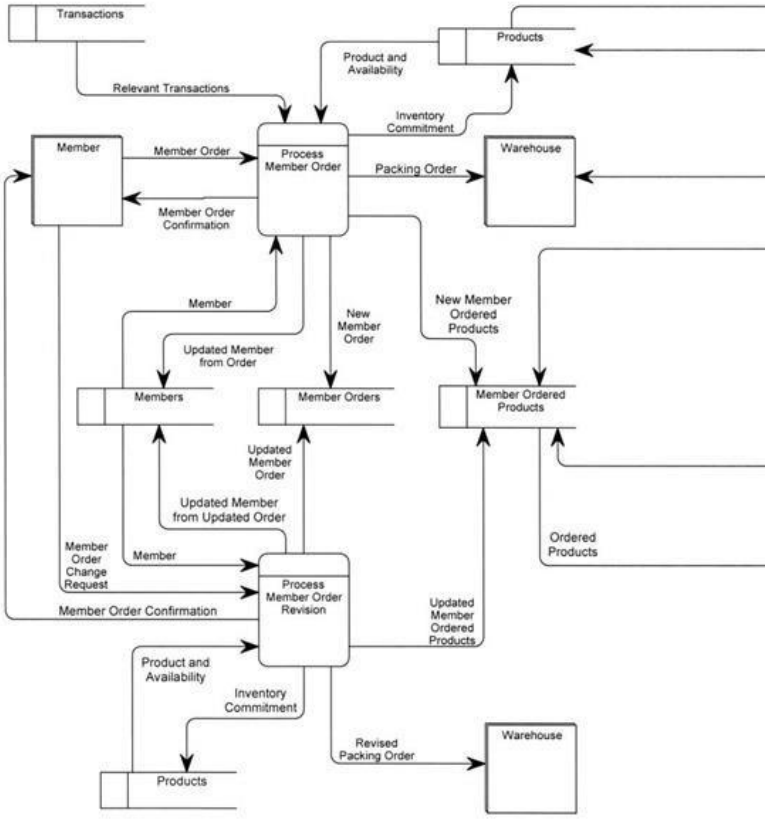


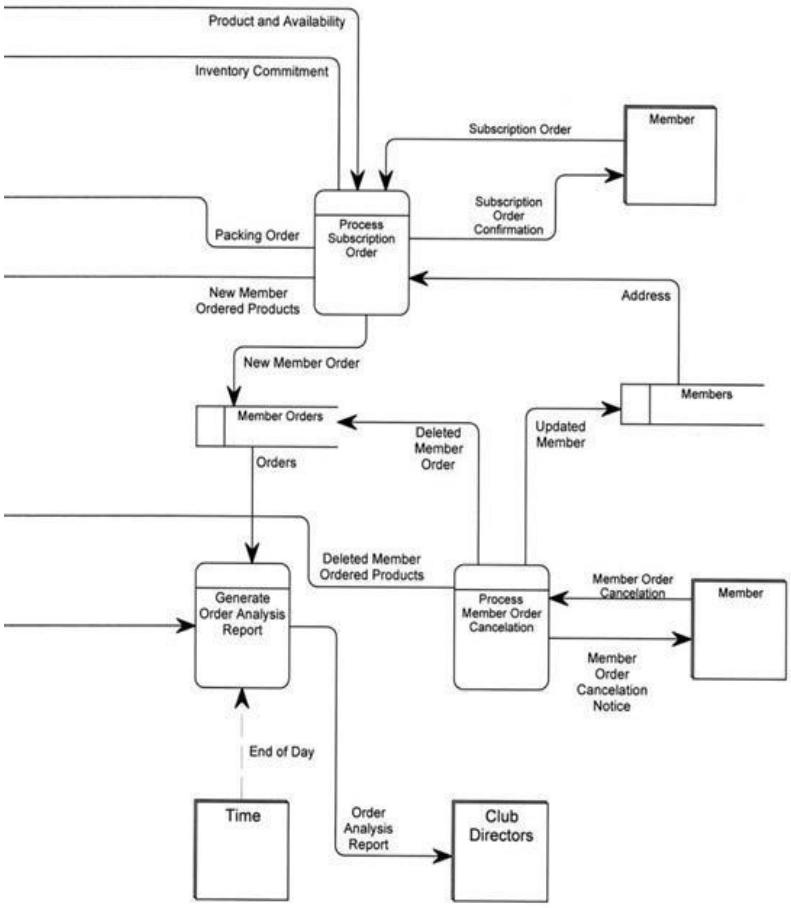
Simple Event Diagram



Complex Event Diagram

System DFD





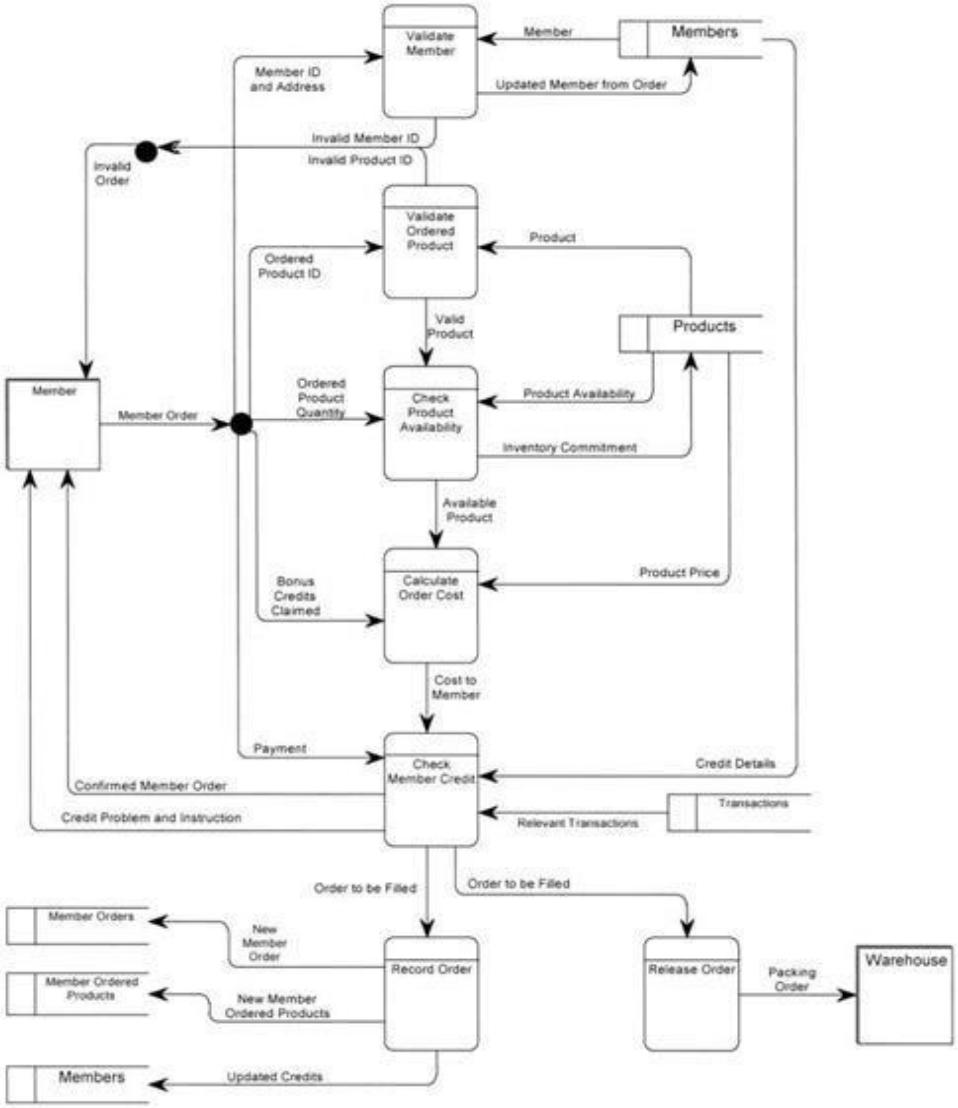
Balancing

• **Balancing** is a concept that requires that data flow diagrams at different levels of detail reflect consistency and completeness.

- Quality assurance technique requires that if you explode a process to another DFD to reveal more detail, you must include the same data flows and data stores.

Primitive Diagrams

- Some (not necessarily all) event processes may be exploded into primitive diagrams to reveal more detail.
- Complex business transaction processes
- Process decomposed into multiple elementary processes
- Each elementary process is cohesive - it does only one thing
- Flow similar to computer program structure



Process Logic

- Data Flow Diagrams are good for identifying and describing processes
- DFDs are not good at showing logic inside processes
- There is always need to specify detailed instructions for elementary processes. This can be done using:
  - Flowcharts & Pseudocode however most end users do not understand them
  - Natural English - imprecise and subject to interpretation

Problems with Natural English

- Many do not write well and do not question writing abilities. • Many too educated to communicate with general audience
- Some write everything like it was a program.
- Can allow computing jargon, acronyms to dominate language.
- Statements frequently have excessive or confusing scope.
- Overuse compound sentences.

- Too many words have multiple definitions.
- Too many statements use imprecise adjectives.
- Conditional instructions can be imprecise.
- Compound conditions tend to show up in natural English

Structured English

- **Structured English** is language syntax for specifying the logic of a process.
- It is based on the relative strengths of structured programming and natural English.

1. For each CUSTOMER NUMBER in the data store CUSTOMERS:
   a. For each LOAN in the data store LOANS that matches the above CUSTOMER NUMBER:
      1) Keep a running total of NUMBER OF LOANS for the CUSTOMER NUMBER.
      2) Keep a running total of ORIGINAL LOAN PRINCIPAL for the CUSTOMER NUMBER.
      3) Keep a running total of CURRENT LOAN BALANCE for the CUSTOMER NUMBER.
      4) Keep a running total of AMOUNTS PAST DUE for the CUSTOMER NUMBER.
   b. If the TOTAL AMOUNTS PAST DUE for the CUSTOMER NUMBER is greater than 100.00 then
      1) Write the CUSTOMER NUMBER and data in the data flow LOANS AT RISK.
      Else
      1) Exclude the CUSTOMER NUMBER and data from the data flow LOANS AT RISK.

Structured English Constructs

Construct	Sample Template
Sequence of steps – Unconditionally perform a sequence of steps.	[Step 1] [Step 2] … [Step n]
Simple condition steps – If the specified condition is true, then perform the first set of steps. Otherwise, perform the second set of steps. Use this construct if the condition has only two possible values. (Note: The second set of conditions is optional.)	**If** [truth condition] **then** [sequence of steps or other conditional steps] **else** [sequence of steps or other conditional steps] ~~**End If**~~
Complex condition steps – Test the value of the condition and perform the appropriate set of steps. Use this construct if the condition has more than two values.	**Do the following based on** [condition]: **Case 1: If** [condition] = [value] then [sequence of steps or other conditional steps] **Case 2: If** [condition] = [value] then [sequence of steps or other conditional steps] … **Case n: If** [condition] = [value] then [sequence of steps or other conditional steps] ~~**End Case**~~

Multiple conditions – Test the value of multiple conditions to determine the correct set of steps.

Use a decision table instead of nested if-then-else Structured English constructs to simplify the presentation of complex logic that involves combinations of conditions.

***A decision table** is a tabular presentation of complex logic in which rows represent conditions and possible actions and columns indicate which combinations of conditions result in specific actions.*

DECISION TABLE	Rule	Rule	Rule	Rule
[Condition]	value	value	value	value
[Condition]	value	value	value	value
[Condition]	value	value	value	value
[Sequence of steps or conditional steps]	X			
[Sequence of steps or conditional steps]		X	X	
[Sequence of steps or conditional steps]				X

Although it isn't a Structured English construct, a decision table can be named, and referenced within a Structured English procedure.

One-to-many iteration – Repeat the set of steps until the condition is false.

Use this construct if the set of steps must be performed at least once, regardless of the condition's initial value.

Repeat the following until [truth condition]:
[sequence of steps or conditional steps]
~~**End Repeat**~~

Zero-to-many iteration – Repeat the set of steps until the condition is false.

Use this construct if the set of steps is conditional based on the condition's initial value.

Do while [truth condition]:
[sequence of steps or conditional steps]
~~**End Do**~~

- OR -

For [truth condition]:
[sequence of steps or conditional steps]
~~**End For**~~

Structured English Restrictions on Process Logic •

Only strong, imperative verbs may be used.

- Only names that have been defined in project dictionary may be used.
- Formulas should be stated clearly using appropriate mathematical notations.
- Undefined adjectives and adverbs are not permitted.
- Blocking and indentation are used to set off the beginning and ending of constructs.
- User readability should always take priority.

Policies and Decision Tables

- **Policy** is a set of rules that govern show a process is to be completed.
- **Decision table** is a tabular form of presentation that specifies a set of conditions and their corresponding actions.
- **Decision tables** are required to implement a policy.

A SIMPLE POLICY STATEMENT

CHECK CASHING IDENTIFICATION CARD

A customer with check cashing privileges is entitled to cash personal checks of up to $75.00 and payroll checks from companies pre-approved by *LMART*. This card is issued in accordance with the terms and conditions of the application and is subject to change without notice. This card is the property of *LMART* and shall be forfeited upon request of *LMART*.

SIGNATURE *Charles C. Parker, Jr.*
EXPIRES **May 31, 2003**

THE EQUIVALENT POLICY DECISION TABLE

	Conditions and Actions	Rule 1	Rule 2	Rule 3	Rule 4
Condition Stubs	C1: Type of check	personal	payroll	personal	payroll
	C2: Check amount less than or equal to $75.00	yes	doesn't matter	no	doesn't matter
	C3: Company accredited by *LMART*	doesn't matter	yes	doesn't matter	no
Action Stubs	A1: Cash the check	X	X		
	A2: Don't cash the check			X	X
		Rules			

Data & Process Model Synchronization CRUD Matrix

Data-to-Process-CRUD Matrix

Entity . Attribute	Process Customer Application	Process Customer Credit Application	Process Customer Change of Address	Process Internal Customer Credit Change	Process New Customer Order	Process Customer Order Cancellation	Process Customer Change to Outstanding Order	Process Internal Change to Customer Order	Process New Product Addition	Process Product Withdrawal from Market	Process Product Price Change	Process Change to Product Specification	Process Product Inventory Adjustment
Customer	C	C			R	R	R	R					
.Customer Number	C	C			R	R	R	R					
.Customer Name	C	C	U		R		R	R					
.Customer Address	C	C	U		RU		RU	RU					
.Customer Credit Rating		C		U	R		R	R					
.Customer Balance Due					RU	U	R	R					
Order					C	D	RU	RU					
.Order Number					C		R	R					
.Order Date					C		U	U					
.Order Amount					C		U	U					
Ordered Product					C	D	CRUD	CRUD		RU			
.Quantity Ordered					C		CRUD	CRUD					
.Ordered Item Unit Price					C		CRUD	CRUD					
Product					R	R	R	R	C	D	RU	RU	RU
.Product Number					R	R	R	R	C			R	
.Product Name					R		R	R	C			RU	
.Product Description					R		R	R	C			RU	
.Product Unit of Measure					R		R	R	C		RU	RU	
.Product Current Unit Price					R		R	R			U		
.Product Quantity on Hand					RU	U	RU	RU					RU

C = create R = read U = update D = delete

Process Distribution

Process-to-Location-Association Matrix

Process	Customers	Kansas City	. Marketing	. Advertising	. Warehouse	. Sales	. Accounts Receivable	Boston	. Sales	. Warehouse	San Francisco	. Sales	San Diego	. Warehouse
Process Customer Application	X					X			X			X		
Process Customer Credit Application	X						X							
Process Customer Change of Address	X					X			X			X		
Process Internal Customer Credit Change							X							
Process New Customer Order	X					X			X			X		
Process Customer Order Cancellation	X					X			X			X		
Process Customer Change to Outstanding Order	X					X			X			X		
Process Internal Change to Customer Order						X			X			X		
Process New Product Addition			X											
Process Product Withdrawal from Market			X											
Process Product Price Change			X											
Process Change to Product Specification			X	X										
Process Product Inventory Adjustment					X					X				X

Data Modelling and Analysis

Objectives

- Define data modelling and explain its benefits.
- Recognize and understand the basic concepts and constructs of a data model.
- Read and interpret an entity relationship data model.
- Explain when data models are constructed during a project and where the models are stored.
- Discover entities and relationships.
- Construct an entity-relationship context diagram.
- Discover or invent keys for entities and construct a key-based diagram.
- Construct a fully attributed entity relationship diagram and describe data structures and attributes to the repository.
- Normalize a logical data model to remove impurities that can make a database unstable, inflexible, and nonscalable.
- Describe a useful tool for mapping data requirements to business operating locations.

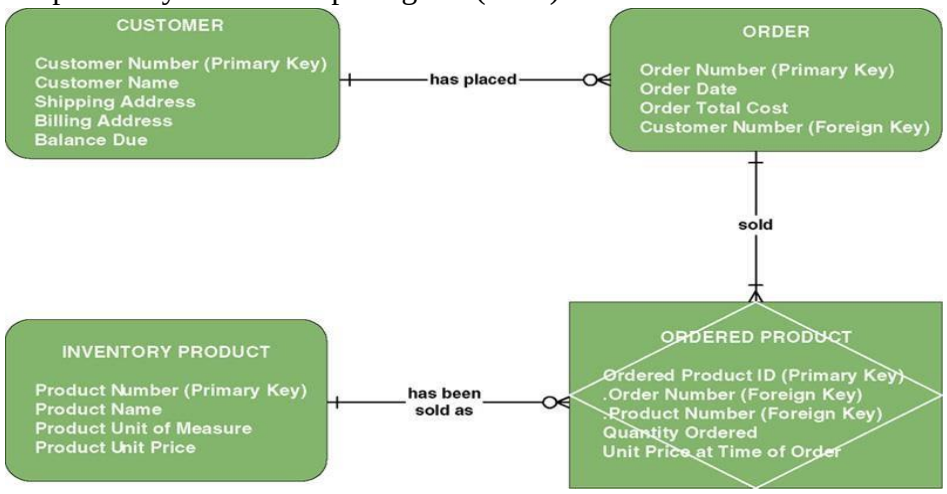
Data Modeling

Data modeling (database modeling) is a technique for organizing and documenting a system's data.

System Concepts for Data Modelling

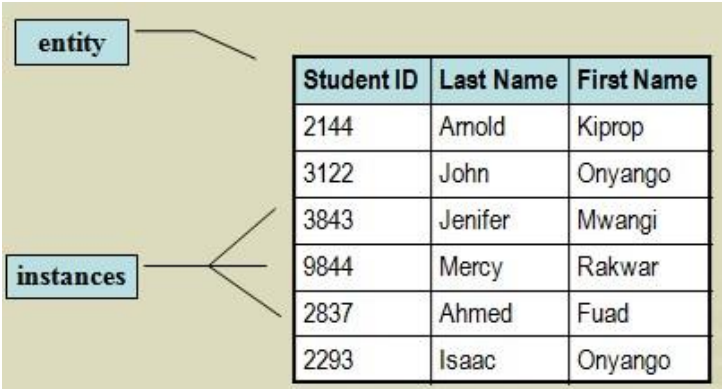
- There are several notations for data modelling. The actual model is frequently called an entity relationship diagram (ERD) because if defines data in terms of the entities and relationships described by the data.
- **Entity relationship diagram (ERD)** is a data model utilizing several notations to depict data in terms of the entities and relationships described by that data.

Sample Entity Relationship Diagram (ERD)



Entities

- Entity is a class of persons, places, objects, events, or concepts about which we need to capture and store data.
- Entity is named by a singular noun.
- Categories of include:
  - Persons: agency, contractor, customer, department, division, employee, instructor, student, supplier.
  - Places: sales region, building, room, branch office, campus.
  - Objects: book, machine, part, product, raw material, software license, software package, tool, vehicle model, vehicle.
  - Events: application, award, cancellation, class, flight, invoice, order, registration, renewal, requisition, reservation, sale, trip.
  - Concepts: account, block of time, bond, course, fund, qualification, stock. Entity instance is a single occurrence of an entity.



Attributes

- If an entity is something about which we want to store data, then we need to identify what specific pieces of data we want to store about each instance of a given entity. This pieces of data are called attributes
- **Attribute** is a descriptive property or characteristic of an entity. Synonyms include *element*, *property*, and *field*.
- Just as a physical student can have attributes, such as hair color, height, etc., data entity has data attributes
- Some attributes can be logically grouped into super attributes called **compound attributes**. For example, a student's NAME is a compound attribute that consists of LAST NAME, SURNAME, and MIDDLE NAME.
- **Compound attribute** is an attribute that consists of other attributes. Synonyms in different data modeling languages are numerous: concatenated attribute, composite attribute, and data structure.

Data Type

Data type is a property of an attribute that identifies what type of data can be stored in that attribute.

Representative Logical Data Types for Attributes	
Data Type	**Logical Business Meaning**
NUMBER	Any number, real or integer.
TEXT	A string of characters, inclusive of numbers. When numbers are included in a TEXT attribute, it means that we do not expect to perform arithmetic or comparisons with those numbers.
MEMO	Same as TEXT but of an indeterminate size. Some business systems require the ability to attach potentially lengthy notes to a give database record.
DATE	Any date in any format.
TIME	Any time in any format.
YES/NO	An attribute that can assume only one of these two values.
VALUE SET	A finite set of values. In most cases, a coding scheme would be established (e.g., FR=Freshman, SO=Sophomore, JR=Junior, SR=Senior).
IMAGE	Any picture or image.

Domain

Domain is a property of an attribute that defines what values an attribute can legitimately take on.

Representative Logical Domains for Logical Data Types		
Data Type	**Domain**	**Examples**
NUMBER	For integers, specify the range. For real numbers, specify the range and precision.	{10-99} {1.000-799.999}
TEXT	Maximum size of attribute. Actual values usually infinite; however, users may specify certain narrative restrictions.	Text(30)
DATE	Variation on the MMDDYYYY format.	MMDDYYYY MMYYYY
TIME	For AM/PM times: HHMMT For military (24-hour times): HHMM	HHMMT HHMM
YES/NO	{YES, NO}	{YES, NO} {ON, OFF}
VALUE SET	{value#1, value#2,...value#n} {table of codes and meanings}	{M=Male F=Female}

Default Value

Default value is the value that will be recorded if a value is not specified by the user.

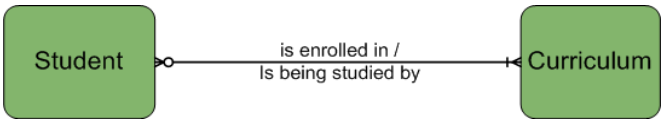
Permissible Default Values for Attributes		
Default Value	**Interpretation**	**Examples**
A legal value from the domain	For an instance of the attribute, if the user does not specify a value, then use this value.	0 1.00
NONE or NULL	For an instance of the attribute, if the user does not specify a value, then leave it blank.	NONE NULL
Required or NOT NULL	For an instance of the attribute, require that the user enter a legal value from the domain. (This is used when no value in the domain is common enough to be a default but some value must be entered.)	REQUIRED NOT NULL

Identification

- **Key** is an attribute, or a group of attributes, that assumes a unique value for each entity instance. It is sometimes called an *identifier*.
  - **Concatenated key** is group of attributes that uniquely identifies an instance. Synonyms: composite key, compound key.
  - **Candidate key** is one of a number of keys that may serve as the primary key. Synonym: *candidate identifier*.
  - **Primary key** is a candidate key used to uniquely identify a single entity instance.
  - **Alternate key** is a candidate key not selected to become the primary key. Synonym: secondary key.

Relationships

- Relationship is a natural business association that exists between one or more entities.
- The relationship may represent an event that links the entities or merely a logical affinity that exists between the entities.



Cardinality

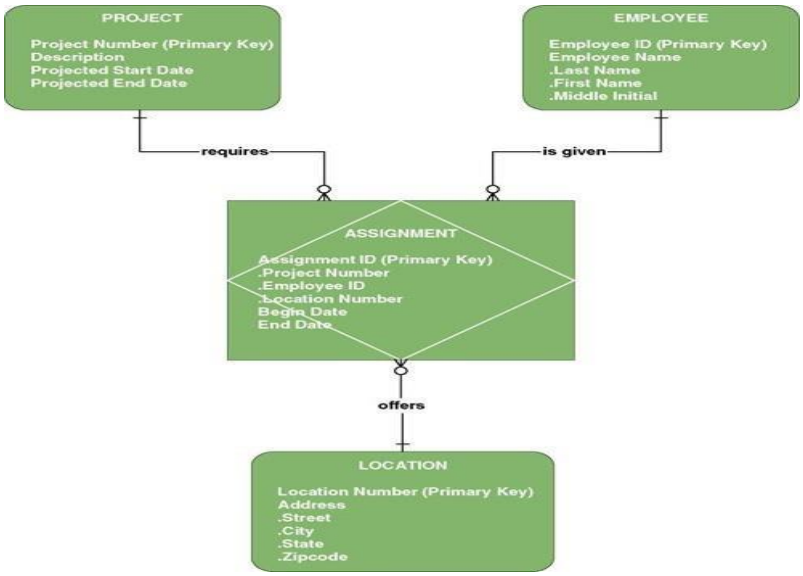
- **Cardinality** is the minimum and maximum number of occurrences of one entity that may be related to a single occurrence of the other entity.
- Because all relationships are bidirectional, cardinality must be defined in both directions for every relationship.

Cardinality Notations

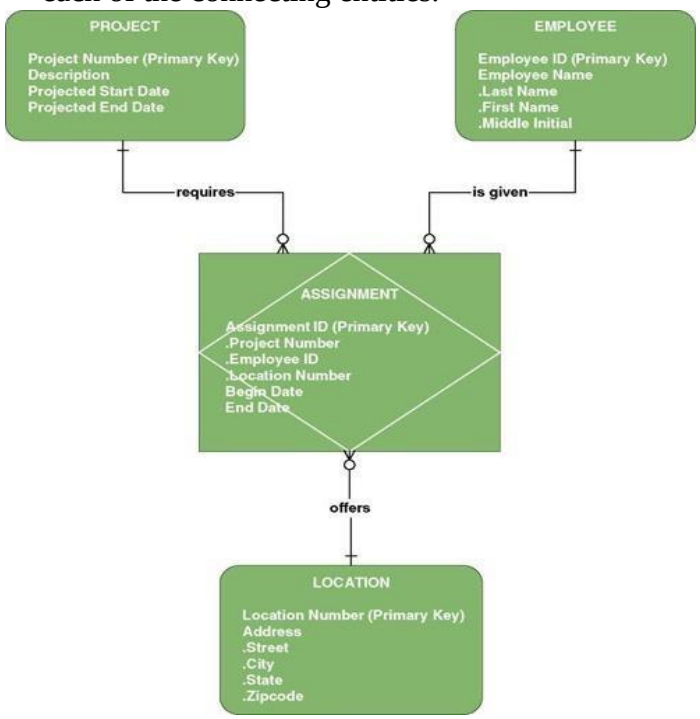
CARDINALITY INTERPRETATION	MINIMUM INSTANCES	MAXIMUM INSTANCES	GRAPHIC NOTATION
Exactly one (one and only one)	1	1	– or –
Zero or one	0	1	
One or more	1	many (>1)	
Zero, one, or more	0	many (>1)	
More than one	>1	>1	

Degree

- **Degree** is the number of entities that participate in the relationship.
- A relationship between two entities is called a *binary relationship*.
- A relationship between three entities is called a *3-ary* or *ternary relationship*.
- A relationship between different instances of the same entity is called a *recursive relationship.*
- Relationships may exist between more than two entities and are called *N-ary* relationships. The example ERD depicts a *ternary relationship.*

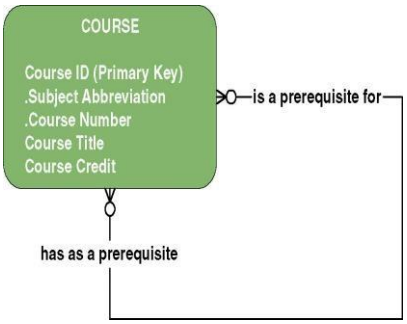


- **Associative entity** is an entity that inherits its primary key from more than one other entity (called parents). Each part of that concatenated key points to one and only one instance of each of the connecting entities.



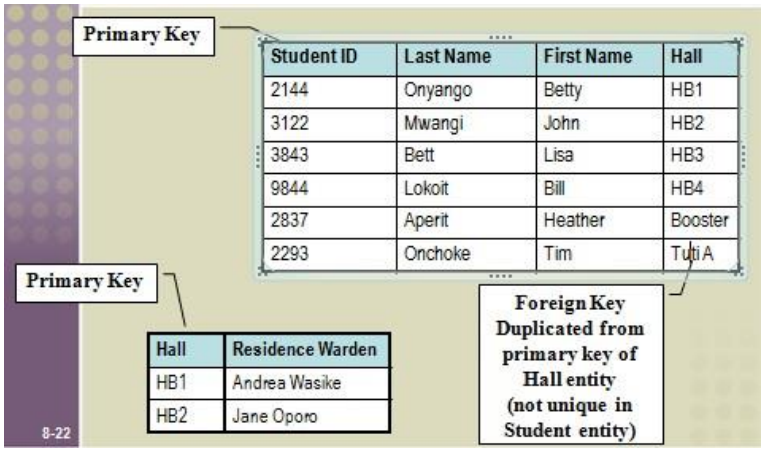
Recursive Relationship

- **Recursive relationship** is a relationship that exists between instances of the same entity
- A classic example of a recursive relationship is in an Employee entity with a Supervisor attribute that holds the identifier of the supervisor's instance of that same Employee entity.



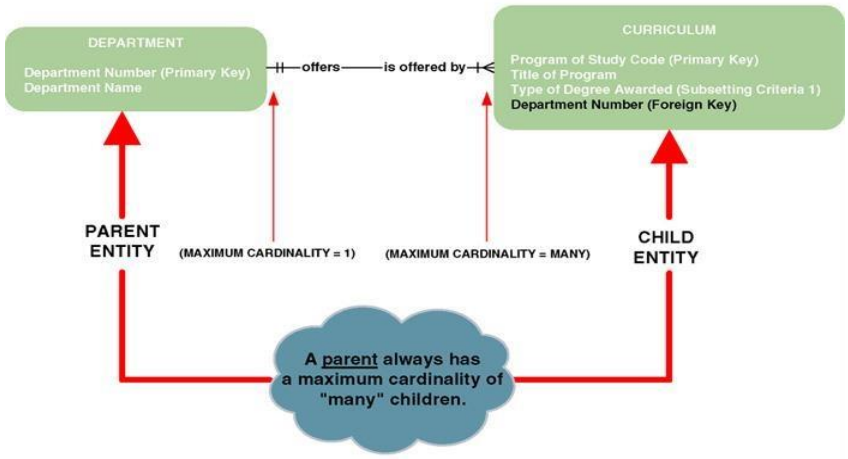
Foreign Keys

- **Foreign key** is a primary key of an entity that is used in another entity to identify instances of a relationship.
  - A foreign key is a primary key of one entity that is contributed to (duplicated in) another entity to identify instances of a relationship.
  - A foreign key always matches the primary key in the another entity − A foreign key may or may not be unique (generally not) − The entity with the foreign key is called the child.
  - The entity with the matching primary key is called the parent.
- **Parent entity** is a data entity that contributes one or more attributes to another entity, called the child. In a one-to-many relationship the parent is the entity on the "one" side.
- **Child entity** is a data entity that derives one or more attributes from another entity, called the parent. In a one-to-many relationship the child is the entity on the "many" side.

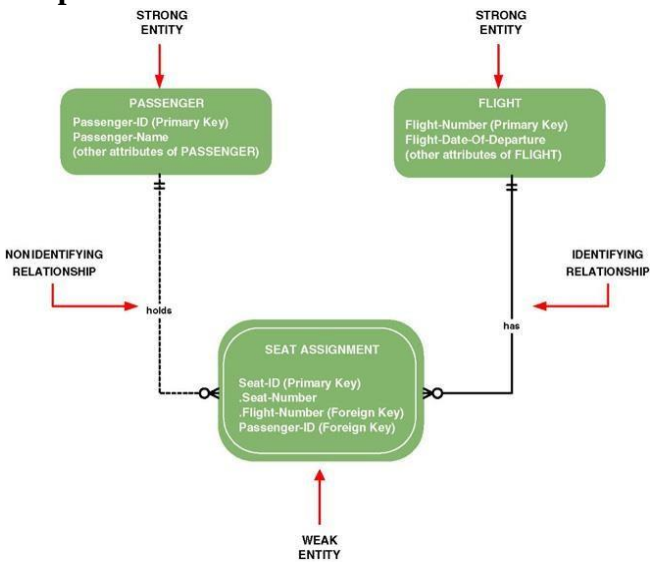


Nonidentifying Relationships

- **Non-identifying relationship** are relationship where each participating entity has its own independent primary key
- Primary key attributes are not shared.
- The entities are called *strong* entities

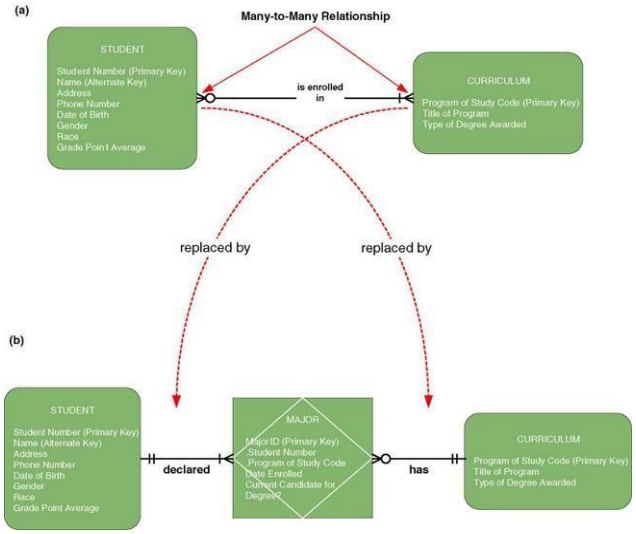


Sample CASE Tool Notations



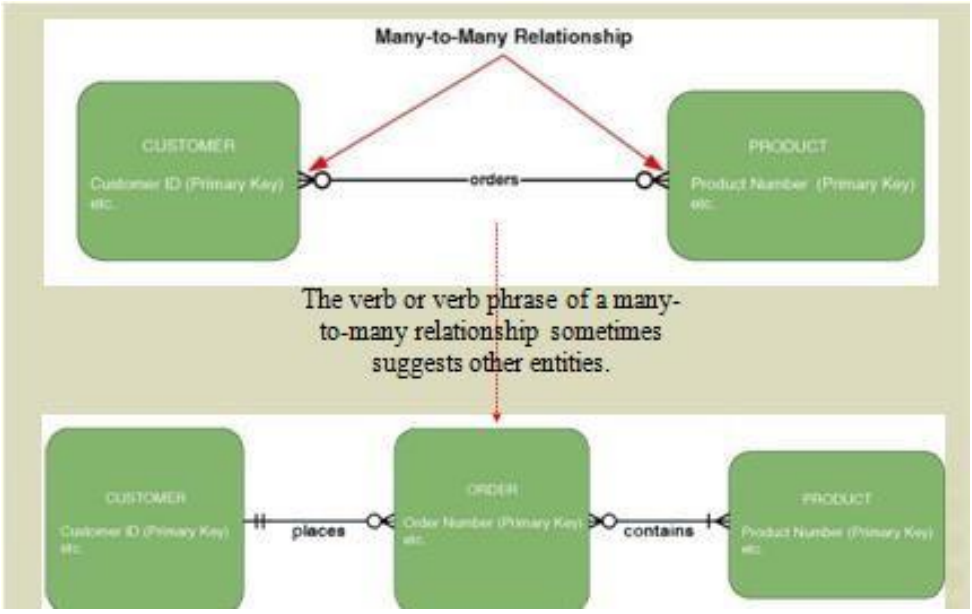
Nonspecific Relationships

- **Nonspecific relationship** is relationship where many instances of an entity are associated with many instances of another entity. Also called *many-to-many relationship*.
- *Nonspecific relationships must be resolved, generally by introducing an associative entity.*

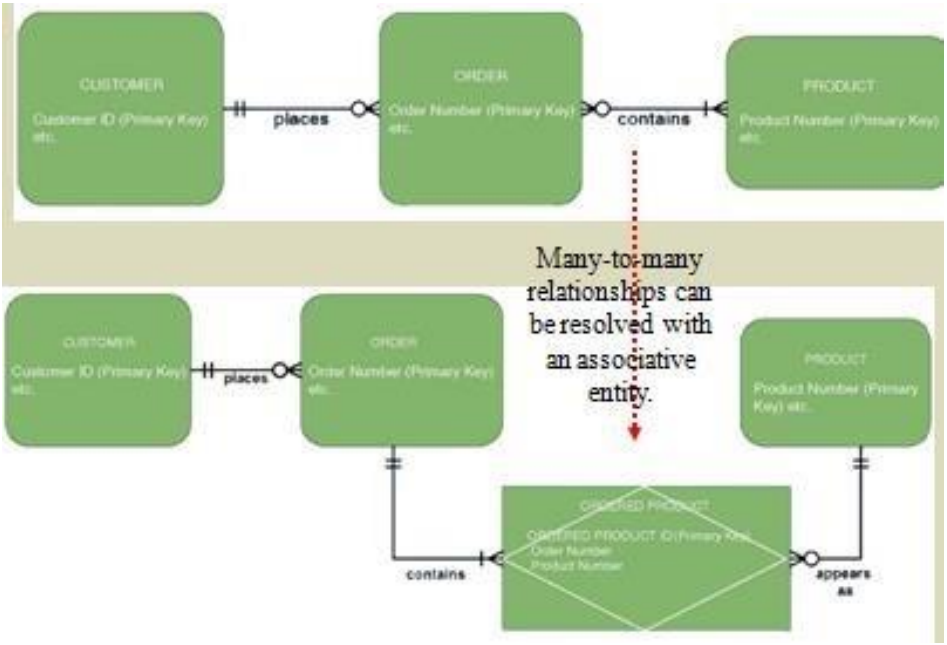


Resolving Nonspecific Relationships

- Many nonspecific relationships can be resolved into a pair of one-to-many relationships.

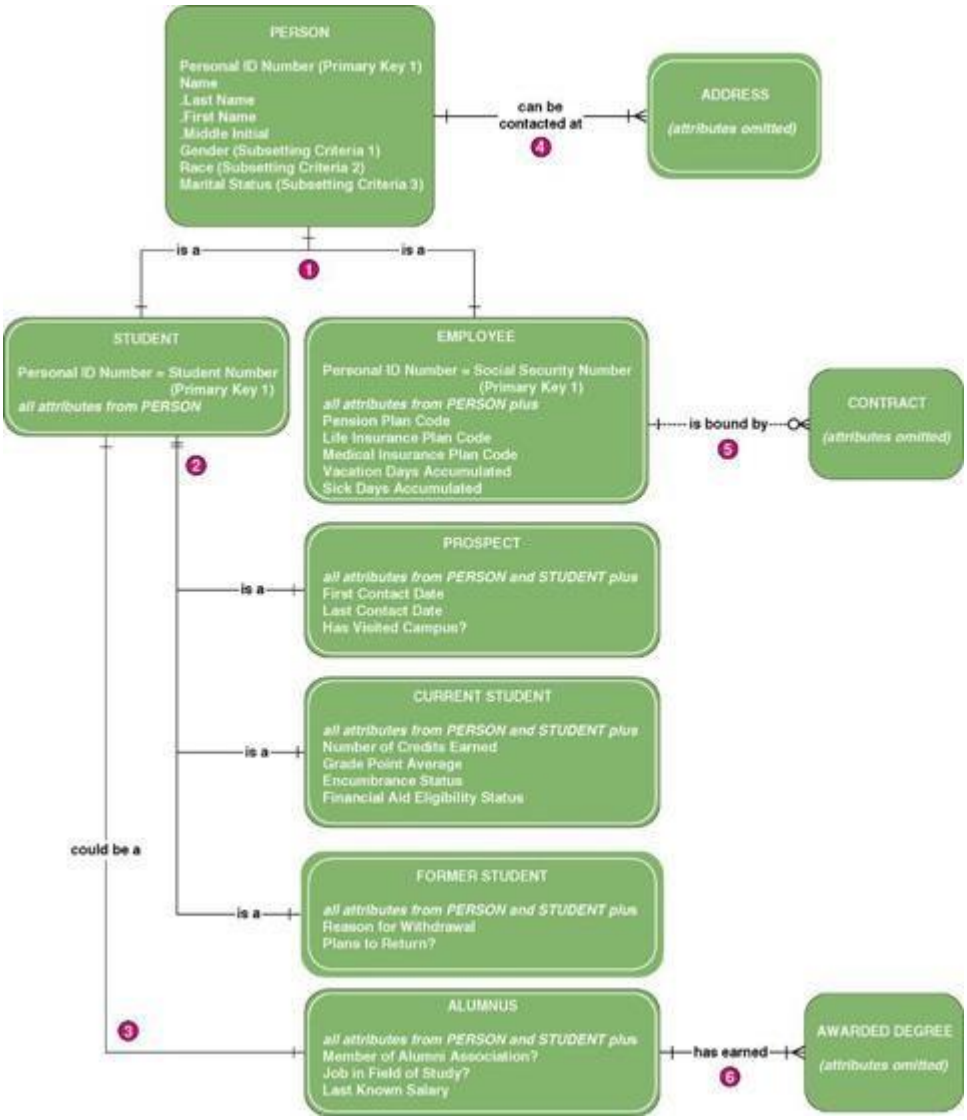


- A new, associative entity is introduced as the child of each parent.



Generalization

- **Generalization** is a concept wherein the attributes that are common to several types of an entity are grouped into their own entity.
- **Supertype** is an entity whose instances store attributes that are common to one or more entity subtypes.
- **Subtype** is an entity whose instances may inherit common attributes from its entity supertype and then add other attributes unique to the subtype.
- One common example is EMPLOYEE (supertype) with HOURLY EMPLOYEE (subtype) and SALARY EMPLOYEE (subtype).



Process of Logical Data Modeling

- Data modeling may be performed during various types of projects and in multiple phases of projects.
- Data models are progressive and should be considered a living document that will change in response to changing business needs.

Strategic Data Modeling

- Many organizations select IS development projects based on strategic plans. This is called **strategic data modeling.**
- It includes vision and architecture for information systems
- Identifies and prioritizes develop projects
- Includes enterprise data model as starting point for projects

Data modeling during system analysis

- Data model for a single information system is called an **application data model**.

Logical Model Development Stages

1. Context Data model
    - Includes only entities and relationships
    - To establish project scope
2. Key-based data model
    - Eliminate nonspecific relationships
    - Add associative entities
    - Include primary and alternate keys
    - Precise cardinalities
3. Fully attributed data model
    - All remaining attributes
    - Subsetting criteria
4. Normalized data model

JRP and Interview Questions for Data Modeling

Regardless of whether you use JRP, interviewing, or any other approach for information gathering, these are good questions to ask.

Purpose	Candidate Questions (see textbook for a more complete list)
Discover system entities	What are the subjects of the business?
Discover entity keys	What unique characteristic (or characteristics) distinguishes an instance of each subject from other instances of the same subject?
Discover entity subsetting criteria	Are there any characteristics of a subject that divide all instances of the subject into useful subsets?
Discover attributes and domains	What characteristics describe each subject?
Discover security and control needs	Are there any restrictions on who can see or use the data?
Discover data timing needs	How often does the data change?
Discover generalization hierarchies	Are all instances of each subject the same?
Discover relationships?	What events occur that imply associations between subjects?
Discover cardinalities	Is each business activity or event handled the same way, or are there special circumstances?

Automated Tools for Data Modelling

- Data models are stored in a repository. In a sense, the data model is metadata - data about the business's data. Computer-Aided systems engineering (CASE) provides the repository for storing the data model and its detailed descriptions.
- Most CASE products support t computer-assisted data modelling and database design.

How to Construct Data Models

Entity Discovery

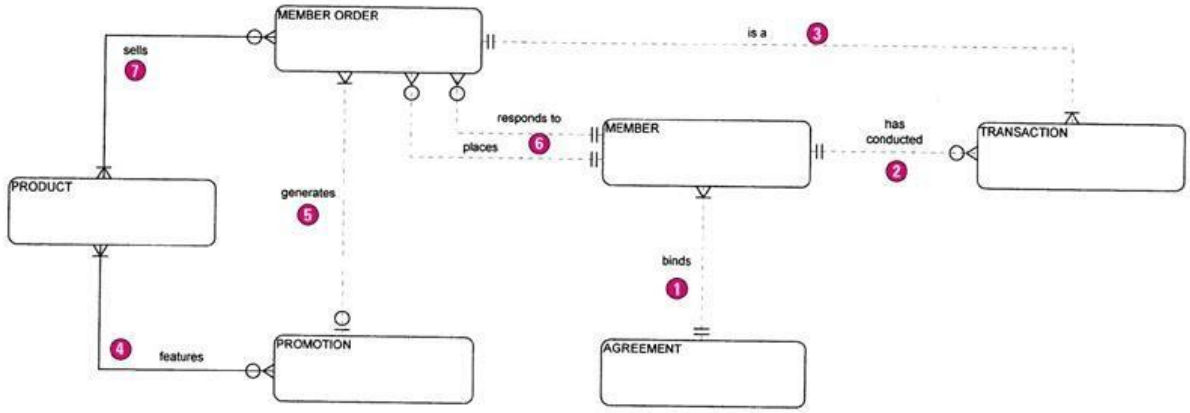
- In interviews or JRP sessions, pay attention to key words (i.e. "we need to keep track of

...").

- In interviews or JRP sessions, ask users to identify things about which they would like to capture, store, and produce information.
- Study existing forms, files, and reports.
- Scan use case narratives for nouns.
- Some CASE tools can reverse engineer existing files and databases.

The Context Data Model

The context data model should include the fundamentals business entities that were previously discovered *as* well as their natural relationships.



The Key-based Data Model

- Identify the keys of each entity.
- The following guidelines are suggested for keys:

1. The value of a key should not change over the lifetime of each entity instance. For example, NAME would be a poor key since a person's last name could change by marriage or divorce.
2. The value of a key cannot be null.
3. Controls must be installed to ensure that the value of a key is valid. This can be accomplished by precisely defining the domain and using the database management system's validation controls to enforce that domain.
4. Some experts (Bruce) suggest you avoid **intelligent keys.** An intelligent key is a business code whose structure communicates data about an entity instance (such as its classification, size, or other properties). A code is a group of characters and/or digits that identifies and describes something in the business system. Some experts argue that because those characteristics can change, it violates rule 1 above.

The Key-based Data Model with Generalization

At this time, it would be useful to identify any generalization hierarchies in the business domain.

Characteristics of a Good Data Model

1. A good data model is simple.
    - Data attributes that describe any given entity should describe only that entity.
    - Each attribute of an entity instance can have only one value.
2. A good data model is essentially nonredundant.
    - Each data attribute, other than foreign keys, describes at most one entity.
    - Look for the same attribute recorded more than once under different names.
3. A good data model should be flexible and adaptable to future needs.

Data Analysis & Normalization

- **Data analysis** is a technique used to improve a data model for implementation as a database. Goal of data analysis is a simple, nonredundant, flexible, and adaptable database.
- **Normalization** is a data analysis technique that organizes data into groups to form nonredundant, stable, flexible, and adaptive entities.

Normalization

- **Normalization** is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.
- There are two goals of normalization:
- Eliminating redundant data (for example, storing the same data in more than one table) Ensuring data dependencies make sense (only storing related data in a table).
- Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

The Normal Forms

- The database community has developed a series of guidelines for ensuring that databases are normalized.
- These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or (1NF) through five (fifth normal form or 5NF). In practical applications, you'll often see 1NF,2NF, and 3NF along with the occasional 4NF. Fifth normal form is very rarely seen.
- Therefore Normalization works through a series of stages: After the first stage, the data is said to be in first normal form, after the second, it is in second normal form, after the third, it is in third normal form etc. The highest level of normalization is not always desirable.
- Normalization Avoids
    - Duplication of Data – The same data is listed in multiple lines of the database
    - Insert Anomaly – A record about an entity cannot be inserted into the table without first inserting information about another entity – Cannot enter a customer without a sales order
    - Delete Anomaly – A record cannot be deleted without deleting a record about a related entity. Cannot delete a sales order without deleting all of the customer's information.

- Update Anomaly – Cannot update information without changing information in many places. To update customer information, it must be updated for each sales order the customer has placed.

Benefits of Normalization

- Facilitates data integration.
- Reduces data redundancy.
- Provides a robust architecture for retrieving and maintaining data.
- Compliments data modelling.
- Reduces the chances of data anomalies occurring.

Database Dependencies/Functional Dependencies

- A dependency occurs in a database when information stored in the same database table uniquely determines other information stored in the same table. You can also describe this as a relationship where knowing the value of one attribute (or a set of attributes) is enough to tell you the value of another attribute (or set of attributes) in the same table.
- Saying that there is a dependency between attributes in a table is the same as saying that there is a functional dependency between those attributes. If there is a dependency in a database such that attribute B is dependent upon attribute A, you would write this as "A -> B".
- For example, in a table listing employee characteristics including Social Security Number (SSN) and name, it can be said that name is dependent upon SSN (or SSN -> name) because an employee's name can be uniquely determined from their SSN. However, the reverse statement (name -> SSN) is not true because more than one employee can have the same name but different SSNs.

Trivial Functional Dependencies

- A **trivial functional dependency** occurs when you describe a functional dependency of an attribute on a collection of attributes that includes the original attribute. For example, "{A, B} -> B" is a trivial functional dependency, as is "{name, SSN} -> SSN". This type of functional dependency is called trivial because it can be derived from common sense
- It is obvious that if you already know the value of B, then the value of B can be uniquely determined by that knowledge.

Full Functional Dependencies

- A **full functional dependency** occurs when you already meet the requirements for a functional dependency and the set of attributes on the left side of the functional dependency statement cannot be reduced any farther.
- For example, "{SSN, age} -> name" is a functional dependency, but it is not a full functional dependency because you can remove age from the left side of the statement without impacting the dependency relationship.

Transitive Dependencies

• **Transitive dependencies** occur when there is an indirect relationship that causes a functional dependency. For example, ”A -> C” is a transitive dependency when it is true only because both “A -> B” and “B -> C” are true.

Multivalued Dependencies

- **Multivalued dependencies** occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.
- For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table. If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

Importance of Dependencies

Database dependencies are important to understand because they provide the basic building blocks used in database normalization. For example:

- For a table to be in second normal form (2NF), there must be no case of a non-prime attribute in the table that is functionally dependent upon a subset of a candidate key.
- For a table to be in third normal form (3NF), every non-prime attribute must have a nontransitive functional dependency on every candidate key.
- For a table to be in Boyce-Codd Normal Form (BCNF), every functional dependency (other than trivial dependencies) must be on a superkey.
- For a table to be in fourth normal form (4NF), it must have no multivalued dependencies.

Basic Rules for Normalization

- The attribute values in a relational table should be functionally dependent (FD) on the primary key value. ○ A relationship is functionally dependent when one attribute value implies or determines the attribute value for the other attribute.

  EM_SS_NUM --> EM_NAME
- **Corollaries** ○ **Corollary 1:** No repeating groups allowed in relational tables. ○ **Corollary 2:** A relational table should not have attributes involved in a transitive dependency

Before Normalization

1. Begin with a list of all of the fields that must appear in the database. Think of this as one big table.
2. Do not include computed fields
3. One place to begin getting this information is from a printed document used by the system.
4. Additional attributes besides those for the entities described on the document can be added to the database.

Before Normalization – Example

See Sales Order from below:

Sales Order

Fiction Company
202 N. Main
Mahattan, KS 66502

CustomerNumber:	*1001*	**Sales Order Number:**	405
Customer Name:	*ABC Company*	**Sales Order Date:**	2/1/2000
Customer Address:	*100 Points*	**Clerk Number:**	*210*
	Manhattan, KS 66502	**Clerk Name:**	*Martin Lawrence*

Item Ordered	Description	Quantity	Unit Price	Total
800	*widgit small*	*40*	*60.00*	*2,400.00*
801	*tingimajigger*	*20*	*20.00*	*400.00*
805	*thingibob*	*10*	*100.00*	*1,000.00*
	Order Total			*3,800.00*

Fields in the original data table will be as follows:

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName, ItemNo, Description, Qty, UnitPrice

Think of this as the baseline – one large table

Normalization: First Normal Form

1NF Definition

The term first normal form (1NF) describes the tabular format in which:

- All the key attributes are defined.
- There are no repeating groups in the table.
- All attributes are dependent on the primary key.

First normal form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

The first rule dictates that we must not duplicate data within the same row of a table. Within the database community, this concept is referred to as the atomicity of a table. Tables that comply with this rule are said to be atomic.

- Separate Repeating Groups into New Tables.
- ***Repeating Groups*** Fields that may be repeated several times for one document/entity
- Create a new table containing the repeating data
- The primary key of the new table (repeating group) is always a composite key: Usually document number and a field uniquely describing the repeating line, like an item number.

First Normal Form Example The
new table is as follows:
SalesOrderNo, ItemNo, Description, Qty, UnitPrice

The repeating fields will be removed from the original data table, leaving the following.

SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

These two tables are a database in first normal form

What if we did not Normalize the Database to First Normal Form?
Repetition of Data – SO Header data repeated for every line in sales order.

Normalization: Second Normal Form

A table is in 2NF if:
- It is in 1NF and
- It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.

Note:
It is still possible for a table in 2NF to exhibit transitive dependency; that is, one or more attributes may be functionally dependent on non-key attributes.

- Remove Partial Dependencies.
    - ***Functional Dependency***: The value of one attribute in a table is determined entirely by the value of another.
    - ***Partial Dependency***: A type of functional dependency where an attribute is functionally dependent on only part of the primary key (primary key must be a composite key).
- Create separate table with the functionally dependent data and the part of the key on which it depends. Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
- Create relationships between these new tables and their predecessors through the use of foreign keys.

2NF attempts to reduce the amount of redundant data in a table by extracting it, placing it in new table(s) and creating relationships between those tables.

Second Normal Form Example
The new table will contain the following fields:
ItemNo, Description

All of these fields except the primary key will be removed from the original table. The primary key will be left in the original table to allow linking of data:

SalesOrderNo, ItemNo, Qty, UnitPrice

Never treat price as dependent on item. Price may be different for different sales orders (discounts, special customers, etc.)

Along with the unchanged table below, these tables make up a database in second normal form:
SalesOrderNo, Date, CustomerNo, CustomerName, CustomerAdd, ClerkNo, ClerkName

What if we did not Normalize the Database to Second Normal Form?

- Repetition of Data – Description would appear every time we had an order for the item
- Delete Anomalies – All information about inventory items is stored in the SalesOrderDetail table. Delete a sales order, delete the item.
- Insert Anomalies – To insert an inventory item, must insert sales order. Update Anomalies – To change the description, must change it on every SO.

Normalization: Third Normal Form

3NF Definition
A table is in 3NF if:

- It is in 2NF and
- It contains no transitive dependencies.
- Remove transitive dependencies.

Transitive Dependency is a type of functional dependency where an attribute is functionally dependent on an attribute other than the primary key. Thus its value is only indirectly determined by the primary key.

- Create a separate table containing the attribute and the fields that are functionally dependent on it. Tables created at this step will usually contain descriptions of either resources or agents. Keep a copy of the key attribute in the original file.

Third Normal Form Example The
new tables would be:
CustomerNo, CustomerName, CustomerAdd

ClerkNo, ClerkName

All of these fields except the primary key will be removed from the original table. The primary key will be left in the original table to allow linking of data as follows:

SalesOrderNo, Date, CustomerNo, ClerkNo

Together with the unchanged tables below, these tables make up the database in third normal form.

ItemNo, Description

SalesOrderNo, ItemNo, Qty, UnitPrice

What if we did not Normalize the Database to Third Normal Form?

- Repetition of Data – Detail for Cust/Clerk would appear on every SO Delete Anomalies – Delete a sales order, delete the customer/clerk
- Insert Anomalies – To insert a customer/clerk, must insert sales order.
- Update Anomalies – To change the name/address, etc, must change it on every SO.

Completed Tables in Third Normal Form

Customers: CustomerNo, CustomerName, CustomerAdd

Clerks: ClerkNo, ClerkName

Inventory Items: *ItemNo*, Description

Sales Orders: SalesOrderNo, Date, CustomerNo, ClerkNo

SalesOrderDetail: SalesOrderNo, ItemNo, Qty, UnitPrice

Sample Question

A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise:

- Employee Number
- Employee Name
- Date of Birth
- Department Code
- Department Name
- Project Code
- Project Description
- Project Supervisor

Assume the following:

- Each employee number is unique.

- Each department has a single department code.
- Each project has a single code and supervisor.
- Each employee may work on one or more projects.
- Employee names need not necessarily be unique.
- Project Code, Project Description and Project Supervisor are repeating fields.

1. Normalise this data to Third Normal Form.
2. Create an ER diagram for the database