Chapter 4: The MERISE Method

Part I: General Overview of the MERISE Methodology

Introduction

MERISE was developed around 1978-1979 by a team led by J.L. Lemoigne at the University of Aix-en-Provence. It emerged in response to the limitations of existing methods, such as MINOS and CORIG, amidst the technological transformations of the 1970s and numerous studies on database systems. MERISE stands for "MEthode d'étude et de Réalisation Informatique pour les Systèmes d'Entreprises" (Method for Studying and Realizing Information Systems for companies).

MERISE is based on:

- A holistic view of the enterprise: Setting up an Information System (IS) is tied to reorganizing the enterprise.
- A systemic view of the enterprise, represented by the concept of a "macroscope."

The MERISE method establishes both principles and an approach. This approach involves a series of stages, incorporating various disciplines.

"MERISE": MEthodes pour Rassembler les Idées Sans Effort (Methods for Assembling Ideas Effortlessly).

One of the key aspects of MERISE is to conduct separate studies of data and processes. Initially, these two studies are handled independently. MERISE also thoroughly addresses organizational aspects.

Philosophy of the Method

A. The Three Cycles

The MERISE method is structured around three cycles, or axes:

1. **The Life Cycle**: This cycle defines how each stage follows from the previous one. It includes three main phases:
    - **Conception**: Studying the current system and designing the future system.
    - **Implementation**: Covers the setup and operational phase of the system.
    - **Maintenance**: Ensures that the system evolves and adapts to environmental changes and new objectives throughout its lifetime, eventually leading to its replacement.

2. **The Abstraction Cycle**: This cycle focuses on how to specify an Information System (IS):
    - **Data memory** is described at conceptual, logical, and physical levels.
    - **Process operations** are described conceptually, organizationally, and operationally.

    Each layer is represented through a model. Changes in lower layers don't affect higher layers unless they directly impact the parameters of the layer in question. Each model is defined using formalized rules, principles, vocabulary, and syntax. Transition rules allow for automatic or semi-automatic movement from one model to another.

3. **The Decision Cycle**: Throughout the study and maintenance phases, decisions must be made. These decisions start broadly at the general management level and become increasingly detailed. Global decisions are handled by senior management, while consultation at each level is required for more specific choices.

Example: A decision on how to organize a user's screen layout should not be made without consulting the individual who will be using the screen regularly.

B. The Stages of MERISE

The design and development process of a system in MERISE is divided into several stages:

1. **Master Plan**: Defines the overall strategic approach and objectives for the system.
2. **Preliminary Study**: Conducts an initial analysis to assess the current situation and system needs.
3. **Detailed Study**: Provides an in-depth examination of system requirements and specifications.
4. **Technical Study**: Outlines the technical aspects required for implementation, including hardware and software specifications.
5. **Software Production**: Involves developing the necessary software components according to the design specifications.
6. **Implementation**: Encompasses the deployment and operationalization of the system.

Each stage plays a crucial role in building a coherent system aligned with the organization's goals and resources.

C. A Multi-Level Approach

MERISE addresses various challenges at different levels, acknowledging issues like hardware or software changes, new regulations, and other contextual shifts. This results in three primary levels, each with distinct concerns:

1. **Conceptual Level**: Defines the enterprise's main objectives and management rules, reflecting the organization's goals and constraints. This level is generally the most stable and includes functions such as personnel management, accounting, etc.
2. **Logical Level**: Specifies the organizational structure required to meet objectives, including job roles, operational sequences, and the nature of various processes.
3. **Technical Level**: Details the technical resources (hardware/software) necessary to implement the project. This level is subject to frequent changes.

Data and Process Models by Level

Each level consists of data and process models:

- **Conceptual Level**: Conceptual Data Model (CDM) and Conceptual Process Model (CPM).
- **Logical Level**: Logical Data Model (LDM) and logical Process Model (LPM).
- **Technical (Physical) Level**: Physical Data Model (PDM) and physical Process Model (PPM).

Additional stages include:

- **Study of the Existing System**: A necessary preliminary analysis for designers unfamiliar with the domain.
- **Validation**: Ensures that each level produces documentation and project files required for review and future reference.

D. Study of the Existing System

For a designer new to the area being studied, examining the existing system is essential. This involves:

1. **Understanding the Domain in Detail**: Gaining a comprehensive knowledge of the study area.
2. **Identifying All Objectives**: Listing all goals pursued by the enterprise in this domain.

If a master plan exists, some of this work may already be done.

During the study of the current system, two types of entities provide input:

- **Workstation**: Offers detailed knowledge of specific tasks.
- **Management**: Provides a broad overview and lists all goals within the domain.

Information Collection Techniques:

Several techniques may be used to gather information, including:

- **Interviews**: Direct contact facilitates a deep understanding.
- **Questionnaires**
- **Surveys**
- Other methods as needed.

The information gathered is structured for analysis. Interviews vary according to the type of entity:

1. **Management Interviews**:
    - Initial understanding of the issue.
    - Listing objectives.
    - Identifying key workstations.
    - Defining interfaces with other projects.
    - Setting boundaries for the study area.

    Results:

    - Main objectives.
    - List of workstations.
    - General quantifications.
    - Study boundaries.
    - Constraints in terms of:
        - **Resources** (material, human, financial).
        - **Timelines** (desired completion times).
        - **Regulations** (labor law, general accounting rules, etc.).
2. **Workstation Interviews**:
    - Document and describe tasks performed.
    - Observe information flow.
    - Learn the organization's terminology.

    Results:

    - Task documentation.
    - Data collection.
    - List of business rules.

Part II. Data Modeling (Conceptual Level)

This phase involves creating the Conceptual Data Model (CDM), which is a graphical and structured representation of the information stored by an IS. The CDM is based on two main elements: entities and associations, which give it the alternative name "Entity-Association Diagram."

The process of creating the CDM includes:

1. Establishing business rules (if they haven't been provided).
2. Developing the data dictionary.
3. Identifying functional dependencies among data items.
4. Creating the CDM (defining entities, associations, and cardinalities).

A. Business Rules

Before defining tables (or entities and associations in conceptual terms), it's necessary to gather the future users' requirements. From these, the developer should be able to establish the business rules governing the data to be stored.

Example: Suppose a developer needs to digitize the information system of a library.

- For each book, the title, publication year, summary, and genre (e.g., novel, poetry, science fiction) must be known.
- A book may have zero (for anonymous works), one, or multiple authors, with details such as name, first name, date of birth, and country of origin recorded.
- Each book copy is identified by a unique reference code of letters and numbers, and each belongs to a single edition.
- Each registered user has a unique ID, along with their name, address, phone number, and email.
- Each registered user can borrow zero, one, or more books, with each loan covering one specific copy. The loan date and allowed duration (in days) are recorded.

The business rules of this system are defined as follows:

- A book may have zero (for anonymous works), one, or multiple authors
- Each book copy belongs to a single edition
- Each registered user can borrow zero, one, or more books,
- Each loan covering one specific copy.

If users do not provide clear rules, it's often the developer's responsibility to establish them. Developers may need to guide users through this process, as required.

B. Data Dictionary

This is an intermediate step, especially useful when multiple people work on the same large database. The data dictionary is a document listing all data items to be stored in the database (and that will appear in the CDM). For each data item, it indicates:

- **Mnemonic Code**: A label for the data (e.g., “title_b” for book title).
- **Designation**: A description of the data item (e.g., “title of the book”).
- **Data Type**:
  - **A (Alphabetic)**: Letters only.
  - **N (Numeric)**: Numbers only.
  - **AN (Alphanumeric)**: Letters and numbers.
  - **Date**: Date format (YYYY-MM-DD).
  - **Boolean**: True or False.
- **Size**: Expressed in characters or digits. For a date (e.g., YYYY-MM-DD), it would be 10 characters.
- **Remarks**: Additional comments, such as if a value must be positive.

Mnemonic code	Designation	Data type	Size	Remarks
Book_ID	Numeric identifier of a book	N		
Book_Title	Title of a book	AN	50	
Book_year	Year of publication of a book	N	4	
Book_Summ	Summary of a book	AN	1000	
Au_ID	Identifier of an author	N		
Au_Name	Name of an author	A	30	
Au_FName	First name of an author	A	30	
Au_Date	Date of birth of an author	Date		

C. Functional Dependencies

A functional dependency (FD) exists when two properties (data items) P1 and P2 are related, such that each P1 value corresponds to exactly one P2 value. This dependency is represented as follows:

$$P1 \rightarrow P2$$

It is said that P1 is the source of the DF and that P2 is its goal. Moreover, multiple properties can be the source, just as multiple properties can be the goal of a DF.

$$P1, P2 \rightarrow P3 \quad P1 \rightarrow P2, P3 \quad P1, P2 \rightarrow P3, P4, P5$$

By referring to the previous data dictionary, the following DFs can be established:

Book_ID➔ Book_Title, Book_year, Book_Summ

Au_ID➔ Au_Name, Au_FName, Au_Date

Functional dependencies allow the identification of business rules, such as:

- From a book ID, retrieve title, publication year, summary, genre ID, and edition.
- From a author ID, retrieve name, first name and date of birth of this author.

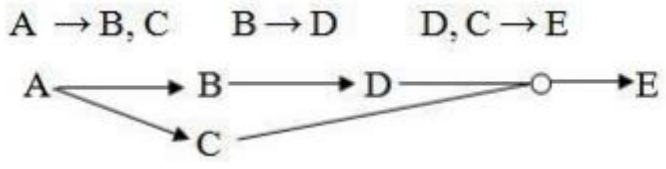
Functional dependency properties include reflexivity, augmentation, transitivity, union, decomposition, and pseudo-transitivity.

Reflexivity	If $Y \subset X$, then $X \rightarrow Y$
Augmentation	$X \rightarrow Y \Rightarrow X, Z \rightarrow Y, Z$
Transitivity	$X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$
Union	$X \rightarrow Y$ et $X \rightarrow Z \Rightarrow X \rightarrow Y, Z$
Decomposition	$X \rightarrow Y, Z \Rightarrow X \rightarrow Y$ et $X \rightarrow Z$
Pseudo-transitivity	$X \rightarrow Y$ et $Y, W \rightarrow Z \Rightarrow X, W \rightarrow Z$

A Functional Dependency Graph (FDG)

FDG can also be used to represent a functional dependency.

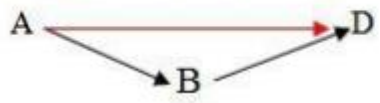
Example:



Transitive Closure

Let F be a set of functional dependencies. The transitive closure of F, denoted F+, is obtained by adding all the transitive functional dependencies to F.

$F = \{A \rightarrow B \quad B \rightarrow D\} \qquad F^{+} = \{A \rightarrow B \qquad B \rightarrow D \qquad A \rightarrow D\}$



A minimal cover, denoted as F*, is the set of essential and direct functional dependencies needed for the system.

Part III: Conceptual Data Model (CDM)

Introduction

The main objective is the creation of a physical database (internal model) containing information related to the organization's activities. The data to be used within the organization constitutes the perceived reality.

The transition from perceived reality to the internal model is achieved through an intermediate level: the conceptual level. The conceptual model is a synthesis of the perceptions of reality by the various actors within the organization.
It is unique and independent of any technological constraints.
An actor's perception will be represented as an external model.

A database design method must:

- Specify the formalisms to be used to describe the models at each level;
- Define the interfaces between the levels;
- Define tools to design each of the models.

Basic Concepts

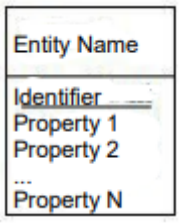
An **entity** is a fundamental concept in database design that represents a distinct object or concept in the real world that is relevant to the system being modeled. It serves as a container for storing data about an object or concept, which is described by its **attributes** (or properties).

Key Characteristics of an Entity:

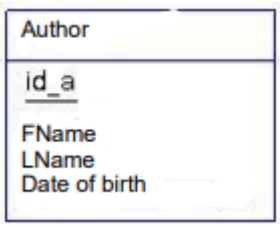
1. **Uniqueness**: Each entity must be uniquely identifiable within its context, typically through a unique attribute called the **identifier** or **primary key**.
2. **Attributes**: An entity is described by a set of attributes, which are characteristics or properties of the entity.
3. **Occurrences**: An entity can have multiple instances (or occurrences), each representing a specific instance of the object it models.
4. **Relationships**: Entities can be linked to one another through **associations** or **relationships** to define interactions between them.

Formal Structure of an Entity:

- **Entity Name**: The label that identifies the type of object (e.g., "Author," "Book").
- **Identifier**: A unique property or combination of properties that distinguishes each occurrence.
- **Attributes**: Properties that describe the entity.



Example of an Entity: Author



Occurrence Example:

If the entity "Author" contains three occurrences, the table might look like this:

ID_A	NAME	FIRST NAME	DATE OF BIRTH
001	Smith	John	1970-05-15
002	Johnson	Emily	1982-10-23
003	Brown	Michael	1995-07-08

Association

An **association** is a concept in database design that represents a **semantic link** or relationship between two or more entities. It describes how the entities interact with one another and helps translate additional business rules that cannot be captured through the individual definition of entities.

Key Characteristics of an Association:

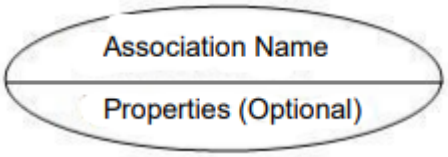
1. **Entities Involved**: Associations link one or more entities together.
2. **Cardinality**: Defines the number of instances of one entity that can be associated with instances of another entity (e.g., one-to-one, one-to-many, many-to-many).

3. **Attributes**: Some associations have their own properties that describe the relationship.

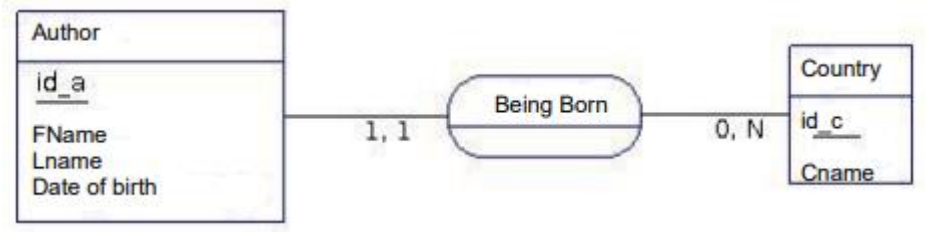
Formal Structure of an Association:

- **Association Name**: Describes the relationship (e.g., "Writes," "Borrows").
- **Entities**: The entities involved in the relationship.
- **Attributes (Optional)**: Properties specific to the association.
- **Cardinality**: Specifies the rules of the relationship (e.g., 1, M).
- **"Author writes Book"** This association links the "Author" and "Book" entities and captures the business rule that an author can write one or more books.

Structure:



Example of an Association:



Here, the "being born" association translates to the following two business rules:

- An author is born in exactly one country.
- In a country, there are no, one, or several authors born.

You will notice that this association is characterized by these annotations (1,1) and (0,N), which allowed us to define the previous business rules. These annotations are called **cardinalities.**

Cardinalities in the context of databases, entity-relationship (ER) diagrams, or data modeling refer to the number of instances of one entity that can or must be associated with an instance of another entity. Cardinality defines the allowable number of relationships between two entities.

Cardinalities are expressed as pairs of numbers: **(minimum, maximum)**, and they specify the rules of participation in the relationship.

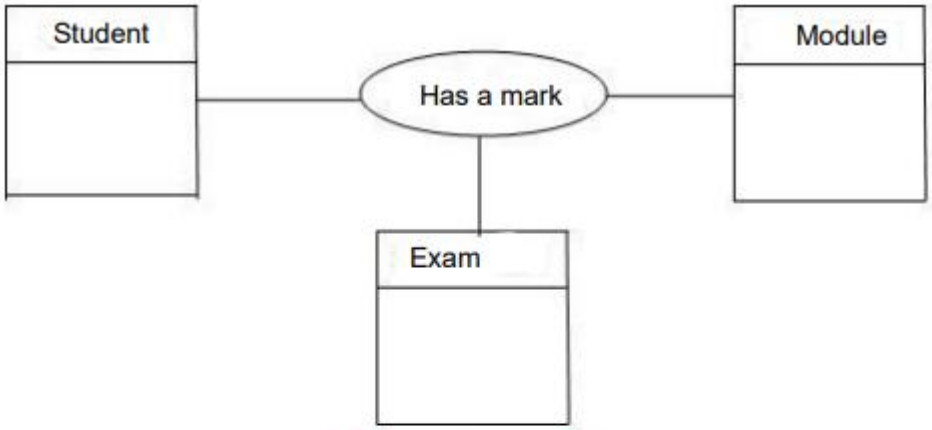
Here are the common types of cardinalities:

1. **(1,1)**:
    - **One-to-One** relationship. Each instance of an entity is associated with exactly one instance of another entity.
    - Example: Each **person** has exactly one **passport**.
2. **(0,1)**:
    - **Zero or One** relationship. An entity instance can either be associated with no instance or exactly one instance of another entity.
    - Example: A **student** may or may not have a **scholarship**.
3. **(0,N)**:
    - **Zero or Many** relationship. An entity instance can be associated with zero or multiple instances of another entity.
    - Example: A **book** can have zero or more **authors**.
4. **(1,N)**:
    - **One or Many** relationship. An entity instance must be associated with at least one, but possibly many, instances of another entity.
    - Example: A **teacher** must teach at least one **course**.
5. **(M,N)** (less common):
    - **Many-to-Many** relationship. Both entities can be associated with multiple instances of the other entity.
    - Example: **Students** can enroll in multiple **courses**, and each **course** can have multiple **students**.

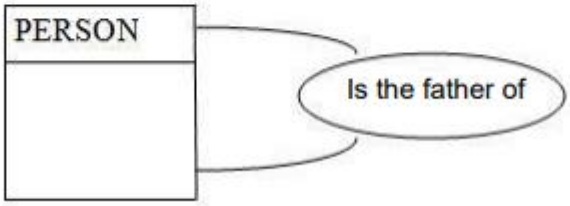
These cardinalities help define the structure and rules of data relationships, ensuring the integrity of the system and accurate representation of real-world constraints.

Dimension of an Association

A relation can be unary (connecting an entity to itself), binary (connecting two entities), or ternary (connecting three entities).



Ternary association



Unary association