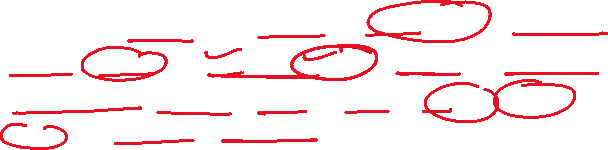
* 1. **Data Modelling Basics, Types of data models, their use, and how they interrelate**



**Data Modeling** is defined in **DAMA DMBOK V2** as the process of defining and analyzing data requirements to support business processes and information systems. This involves creating abstract models that represent data structures, relationships, and business rules.



The purpose of data modeling is to provide a clear and understandable representation of the data landscape, ensuring data consistency, efficiency, and alignment with business objectives. Data modeling helps ensure that data is accurate, organized, and available for use across the organization.

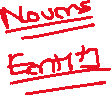


**Key Data Modeling Concepts in DAMA DMBOK V2:**

1. **Data Models**: Represent the structure of data and the relationships between data elements.



1. **Entities and Attributes**: Entities are the primary objects or concepts in the model (e.g., Customer, Order), and attributes are the specific data elements that describe them (e.g., Name, OrderID).



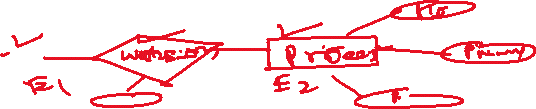
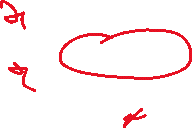
1. **Relationships**: Define how entities interact or are related to one another (e.g., Customer places Order, Product is part of Order).



1. **Normalization**: The process of organizing data to reduce redundancy and improve data integrity.



1. **Business Rules**: These are rules that govern how data behaves and is processed (e.g., "An Order must have at least one Product").



**1. Conceptual Data Model (CDM)**



**Definition**:  
A **Conceptual Data Model (CDM)** is a high-level representation of organizational data. It highlights the core entities and their relationships without getting into implementation details. This model is intended to be understandable by business stakeholders, IT professionals, and anyone needing a broad overview of the data.



**Key Features**:



* **Abstract view**: Focuses on what data is important and how entities relate to each other.



* **Entity-focused**: Describes entities (e.g., Customer, Order, Product) and relationships (e.g., Customer places Order).



* **High-level**: Does not include data types, attributes, or detailed technical specifications.



**Use Case**:

* **Scenario**: An organization wants to develop a new customer relationship management (CRM) system.



* + The conceptual model might identify core entities like **Customer**, **Order**, **Product**, **Payment**, and their relationships (e.g., Customer places Orders, Payment is made for Orders).



* + This is the phase where you are just capturing business needs and the high-level data interactions, without worrying about how this will be implemented in a database.



**Example**:

Entities:

- Customer



- Order



- Payment



Relationships:

- A Customer places an Order.



- An Order has a Payment.



**2. Logical Data Model (LDM)**



**Definition**:  
A **Logical Data Model (LDM)** provides more detail than the conceptual model but is still independent of database technologies and physical constraints. It defines entities, attributes, and relationships in a more structured way, along with business rules, without worrying about specific DBMS requirements.



**Key Features**:



* **Entity relationships**: Details the attributes of entities and their relationships.



* **Normalization**: Often includes the normalization of data to eliminate redundancy.



* **Business rules**: Includes rules such as constraints, default values, or data validation.

**Use Case**:

* **Scenario**: A retail company needs to create a database to store customer information, orders, and inventory. After understanding the high-level business requirements, the company moves into the logical modeling phase.



* + The logical model will define specific attributes like Customer ID, Name, Email, Order ID, Order Date, etc., and the relationships between these entities.



* + The logical model would also specify things like:
    - "Each Customer must have a unique Customer ID."



* + - "Each Order must be associated with one Customer."



**Example**:



Entities:



- Customer (CustomerID, Name, Email)



- Order (OrderID, OrderDate, CustomerID)



- Payment (PaymentID, Amount, PaymentDate, OrderID)



Relationships:



- A Customer can have multiple Orders (1-to-many).



- An Order can have one Payment (1-to-1).



**3. Physical Data Model (PDM)**

**Definition**:  
A **Physical Data Model (PDM)** translates the logical model into a blueprint for physical database implementation. This model defines how data will be stored and accessed in a specific database system (e.g., SQL Server, Oracle, MySQL).



**Key Features**:



* **Storage specifics**: Defines tables, indexes, foreign keys, data types, and other physical properties.



* **Performance optimization**: Focuses on storage, indexing, and performance tuning.



* **Specific to DBMS**: Dependent on the technology being used for implementation.

**Use Case**:

* **Scenario**: The retail company now wants to implement the database on an SQL Server system.
  + The physical model will define how data is stored in tables (e.g., Customers table, Orders table), what data types are used (e.g., VARCHAR for Name, DATE for Order Date), and how to index frequently queried columns (e.g., indexing OrderID for faster search).
  + It would also define things like relationships between tables using foreign keys and any constraints (e.g., NOT NULL, UNIQUE).

**Example**:

Table: Customer

- CustomerID (INT, PRIMARY KEY)

- Name (VARCHAR(100))

- Email (VARCHAR(150), UNIQUE)

Table: Order

- OrderID (INT, PRIMARY KEY)

- OrderDate (DATE)

- CustomerID (INT, FOREIGN KEY REFERENCES Customer(CustomerID))

Table: Payment

- PaymentID (INT, PRIMARY KEY)

- Amount (DECIMAL(10,2))

- PaymentDate (DATE)

- OrderID (INT, FOREIGN KEY REFERENCES Order(OrderID))

Indexes:

- Index on Customer.Email for fast lookup.

- Index on Order.OrderDate for performance in reports.

**4. Dimensional Data Model (DDM)**

**Definition**:  
A **Dimensional Data Model (DDM)** is designed for use in data warehousing and analytics systems. It structures data into **facts** (quantitative data) and **dimensions** (descriptive data) to facilitate efficient querying and reporting, especially for analytical purposes.

**Key Features**:

* **Fact tables**: Contain the metrics or measurements that are the focus of analysis (e.g., sales, revenue).
* **Dimension tables**: Provide descriptive context to the facts (e.g., time, product, location).
* **Denormalization**: Often denormalized for faster query performance.

**Use Case**:

* **Scenario**: The retail company wants to create a data warehouse to analyze sales performance across different regions and time periods.
  + The **fact table** could store sales data (e.g., total sales, quantity sold), and the **dimension tables** would provide context (e.g., Time Dimension, Product Dimension, Region Dimension).
  + Analysts will use this structure to generate reports such as "Sales by Product Category" or "Sales by Region over Time."

**Example**:

Fact Table: Sales

- SaleID (INT, PRIMARY KEY)

- ProductID (INT, FOREIGN KEY to Product Dimension)

- DateID (INT, FOREIGN KEY to Date Dimension)

- RegionID (INT, FOREIGN KEY to Region Dimension)

- QuantitySold (INT)

- TotalSales (DECIMAL(10,2))

Dimension Table: Product

- ProductID (INT, PRIMARY KEY)

- ProductName (VARCHAR(100))

- Category (VARCHAR(50))

Dimension Table: Date

- DateID (INT, PRIMARY KEY)

- Date (DATE)

- Month (VARCHAR(20))

- Year (INT)

Dimension Table: Region

- RegionID (INT, PRIMARY KEY)

- RegionName (VARCHAR(50))

**Summary: Interrelationships Between Models**

* **Conceptual Model**: Starts by providing a high-level view of the system, focusing on entities and their relationships, without worrying about technical details. It serves as the foundation for the other models.
* **Logical Model**: Refines the conceptual model by specifying more detailed data structures and rules but remains independent of physical constraints. It includes attributes and relationships in more detail.
* **Physical Model**: Takes the logical model and maps it to a specific database system. It defines how data is stored and accessed, considering performance, indexing, and storage.
* **Dimensional Model**: A specialized form of logical modeling used for data warehousing, optimized for reporting and analytics.
* **Ontology Model**: Represents knowledge about a domain, used in AI and knowledge-based systems.

**SUMMARY**

*Data modeling, a process that supports efficient database design and management, involves three stages: conceptual, logical, and physical. The first stage, the conceptual model, defines high-level entities and relationships. The second stage, the logical model, adds technical detail like attributes, data types, and structures, and the final stage, the physical model, implements the design in a specific database management system (DBMS). Each model serves a unique purpose, from clarifying business requirements to guiding system implementation. These models are crucial to maintaining data integrity and can evolve as your business needs change.*

Continue reading part one of our three-part series on data modeling to gain an even greater understanding of conceptual, logical, and physical models.

**Conceptual vs. logical vs. physical data modeling**

Conceptual, logical, and physical data models each serve a key function in the data modeling process. A conceptual model provides a high-level system overview, focusing on the main entities and relationships without going into technical details. A logical model adds more structure by defining data elements, attributes, and their relationships. A physical model addresses implementation details, specifying how data will be stored in a database’s tables, columns, and indexes.

Here’s how to think of each model in the simplest terms:

* 1. Conceptual – the “what” model
  2. Logical – the “how” of the details
  3. Physical – the “how” of the implementation

Each level of conceptual, physical, and logical data models can involve different roles from your team.

**Conceptual data model**

The conceptual data model can be considered the “whiteboard” data model. It does not address the “how.”

For this model, it’s important to focus on capturing all the types of data (or “entities”) that the system will need. In addition to entities, a conceptual data model will also capture:

* + **Attributes:** Individual properties of an entity. For instance, a “person” entity may have “name” and “shoe size.” An “address” entity may have “zip code” and “city.”
  + **Relationships:** How an entity connects to other entities. For instance, a “person” entity may have one or more “addresses.”

Along with the entities, their attributes, and relationships, a conceptual model can also:

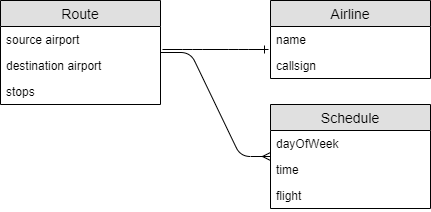
* + **Organize scope:** Details which entities are included and which are **not** included.
  + **Define business rules:** For instance, are person entities allowed to have multiple addresses? What about multiple emails? Do they need to have a unique identifier?

Architects often create the conceptual data model with business stakeholders and domain experts.

*Conceptual data model example*

There are many “languages” for describing a conceptual data model. But as long as it’s documented in an accessible way, it can be as easy as boxes and arrows.

Here’s a diagram of a conceptual data model that involves two core entities, travel routes (and their associated schedules), and *airlines*:



While these may look like tables in a relational database, the conceptual modeling stage is too early to determine how the data will be stored. That determination comes later: it could be tables, JSON documents, graph nodes, CSV files, blockchain, or any other number of storage mediums.

**Logical data model**

A logical data model is decided upon after stakeholders agree on a conceptual model.

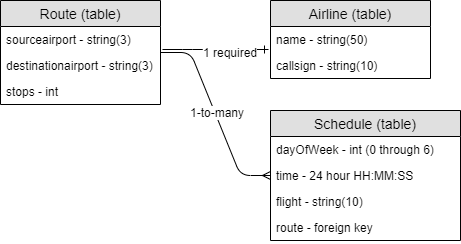
This stage involves filling in the details of the conceptual model. It’s still too early to pick a database management system (DBMS), but this stage can help you decide which database to use (relational, document, etc.). For instance, if you choose **relational**, you’ll need to decide which tables to create. If you choose **document**, you’ll need to define the collections.

During this step, you should also decide the details of each field or column and relationship. These details include data types, sizes, lengths, arrays, nested objects, etc.

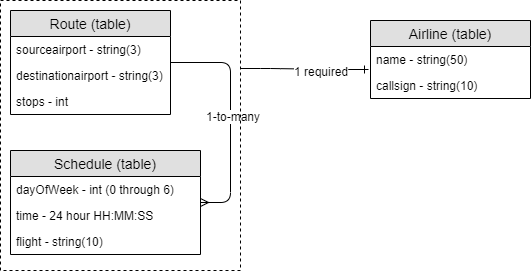
Architects and business analysts typically create the logical model.

*Logical data model example*

For instance, if you’re going with a relational model, the logical model might look like this:



However, with a document database, the schedule can be modeled directly as part of the route. There’s no need for a foreign key, but it’s still helpful to consider it its own sub-entity. So that logical model might look like this:



A schedule, which has a fairly small, finite footprint, should be embedded in the same collection. Social media posts, which are unbounded, should be modeled in separate collections.

**Physical data model**

Once you’ve defined a logical model, it’s time to implement it into a database.

If you decide on a relational model, some options you can choose from include Microsoft SQL Server, Oracle, PostgreSQL, or MySQL. However, if your modeling process reveals that your data model will likely change frequently to adapt to new requirements, you should consider using a document database. Couchbase, a NoSQL document database, supports relational concepts like JOINs, ACID transactions, and flexible JSON data.

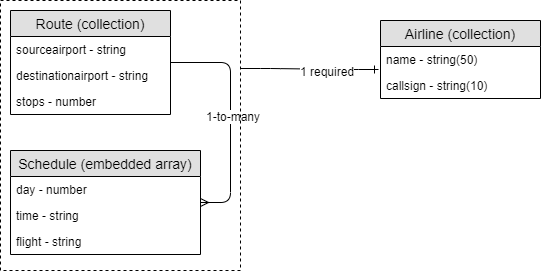
The physical data model should include:

* + A specific DBMS
  + Specifications for storing data (e.g., on disk, RAM, or hybrid)
  + Instructions for accommodating replications, shards, partitions, etc.

Database administrators (DBAs) and developers typically create the physical data model.

*Physical data model example*

Here’s an example of a physical model for Couchbase:



It’s usually helpful to show sample data along with the physical model.