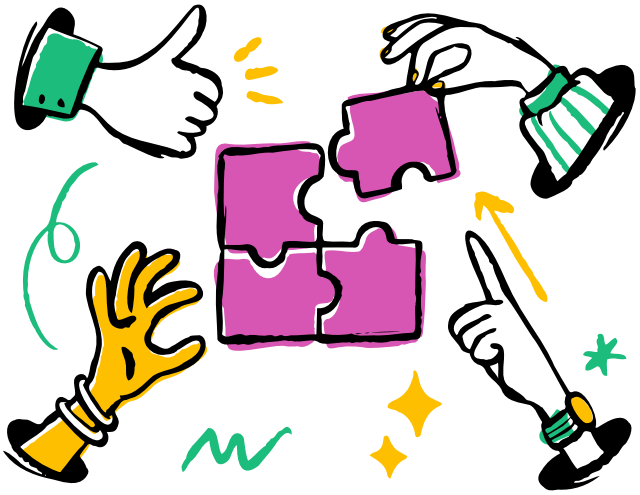
Comprehensive Guide For Data Modelling



Follow Me
https://www.linkedin.com/in/dhwanimehta05/

Data Modelling

What is data modeling?

Imagine your business's data as a big puzzle. Data modeling is like creating a guide that shows how all the pieces of that puzzle fit together. This guide helps everyone involved—whether they're data analysts, scientists, or engineers—see the big picture and work with the data in a unified way.

Data modeling involves taking a close look at all the different types of information your business collects and figuring out how they connect. Think of it as drawing a map that uses symbols, text, and diagrams to show how data flows through your business—from how it's captured to how it's stored and used.

As your business grows and evolves, data modeling becomes a powerful tool for understanding what data you need, how it should be organized, and how it can best be used. It's like having a playbook that guides everyone on how to manage and make sense of the data, making sure every piece of the puzzle falls into place perfectly.

Follow Me
https://www.linkedin.com/in/dhwanimehta05/

Why is data modeling important? and what are its' benefits?

Data modeling is Important because, in today's world, businesses gather a huge amount of data from various sources. But just having raw data isn't enough—you need to analyze it to get valuable insights that can help you make smart business decisions. For accurate analysis, data needs to be collected, stored, and processed efficiently. Different types of data require different tools, and data modeling helps you choose the right ones.

Think of data modeling like an architect designing a blueprint before building a house. Before your business sets up database systems, you create a data model. This model helps you understand your data, choose the best technology to manage it, and organize everything in a way that makes sense.
Here's why data modeling is so beneficial:

- **Fewer Mistakes** It reduces errors when developing database software, saving time and frustration.
- **Efficiency** It speeds up the process of designing and creating databases, making everything run smoother.
- **Consistency** It ensures that everyone in the organization is on the same page when it comes to how data is documented and systems are designed.

Better Communication It makes it easier for data engineers and business intelligence teams to work together and understand each other.

By modeling your data, you also gain a clear picture of what data you have, how you use it, and how it needs to be managed. This collaboration between IT and business teams opens up opportunities to improve business processes and save time and money by planning better.

Understanding Data Requirements

Let's walk through the process of **Understanding Data Requirements** in the context of a retail business, like a small online store.

1. Identify Entities

First, let's think about what key things (or entities) the business needs to keep track of. These are the things that have important information associated with them.

What Are Entities?

Entities are like the main categories or objects that the business needs to manage. In a retail business, some common entities would be:

- **Customers** These are the people who buy things from your store. You need to keep track of who they are, how to contact them, and what they buy.
- **Products** These are the items you sell. Each product has details like its name, price, and maybe a description.
- **Orders** These are the transactions that happen when a customer buys something. An order records what the customer bought, how much they paid, and when the purchase happened.

Example in a Retail Business:

Imagine you run a small online store selling books and other items. The main entities you would care about include:

- **Customers** Each person who buys from you. You'd store their name, email, and shipping address.
- **Products** Every book or item you sell. You'd keep details like the title of the book, its price, and how many copies you have in stock.
- **Orders** Each time a customer makes a purchase, you record what they bought, when they bought it, and how much they paid.

These entities are like the building blocks of your database. They represent the key parts of your business that you need to keep track of every day.

2. Define Relationships

Once you know what your entities are, the next step is to figure out how these entities relate to each other. In other words, how do these different parts of your business connect and interact?

Understanding Relationships:

Relationships show how entities are connected. There are a few common types of relationships:

One-to-One Relationship:

- This is where one record in an entity is linked to one record in another entity.

- Example: If you had a special profile for each customer with extra details like preferences or loyalty points, and each customer could only have one profile, this would be a one-to-one relationship.

One-to-Many Relationship:

- This is where one record in an entity can be related to many records in another entity.
- Example: A single customer can place multiple orders over time. So, one customer (in the Customers entity) can be linked to many orders (in the Orders entity). Each order, however, is linked to only one customer.

Many-to-Many Relationship:

- This is where many records in one entity can relate to many records in another entity.
- Example: An order can contain multiple products, and each product can appear in multiple orders. So, there's a many-to-many relationship between Orders and Products. To manage this, you need a middle table (sometimes called a junction table) that links them together.

Example in a Retail Business:

Continuing with our online bookstore example, let's define the relationships:

- **Customers and Orders One-to-Many Relationship)**:
  - Each customer can place multiple orders over time. For example, "John Doe" might buy something today and then come back next month to buy something else. But each order only belongs to one customer. You wouldn't want an order to be linked to multiple customers; that wouldn't make sense.
- **Orders and Products (Many-to-Many Relationship)**:
  - Each order can include several different products. For example, one order might include "Book A" and "Book B." At the same time, each product, like "Book A," could be part of many different orders placed by different customers. To handle this in the database, you'd create an extra table, let's call it "OrderDetails," that lists each product included in each order.

Putting It All Together:

Imagine this scenario:

- **John Doe** is a customer who buys from your store. He places an **Order** on August 13th. In this order, he buys **Book A** and **Book B**.

Here's how it looks in terms of entities and relationships:

- **Customer** John Doe Name, Email, Address)
- **Order** Order #001 Date: August 13th, Linked to John Doe)
- **Products**:
  - Book A Title: "Learning SQL", Price: $30
  - Book B Title: "Data Modeling 101", Price: $25

Relationships:

- John Doe Customer) placed Order #001 (One-to-Many).
- Order #001 includes Book A and Book B (Many-to-Many).

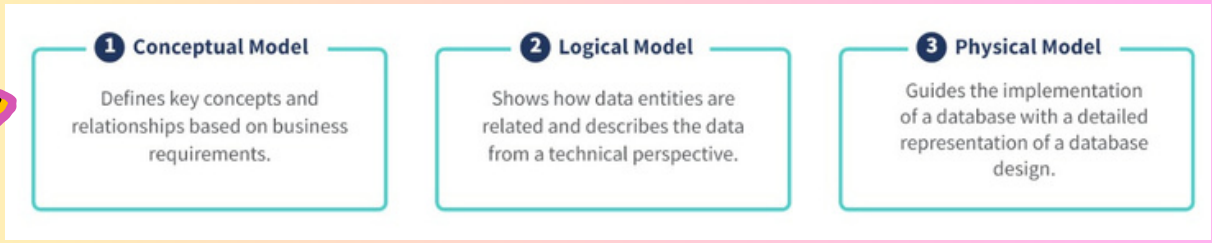
Why This Matters

Understanding entities and their relationships is crucial because it helps you build a database that reflects the real world of your business. If you don't get this part right, your database might not work the way you need it to.

- **For example** If you don't set up the relationship correctly between orders and products, you might find it difficult or even impossible to track what products were sold in each order.
- **Another example** If you didn't understand the one-to-many relationship between customers and orders, you might accidentally set up your database so that each customer could only place one order. That would be a big problem for any business!

By carefully identifying the entities and understanding how they relate to each other, you ensure that your database is well-organized, efficient, and capable of handling the day-to-day operations of your business without any issues.

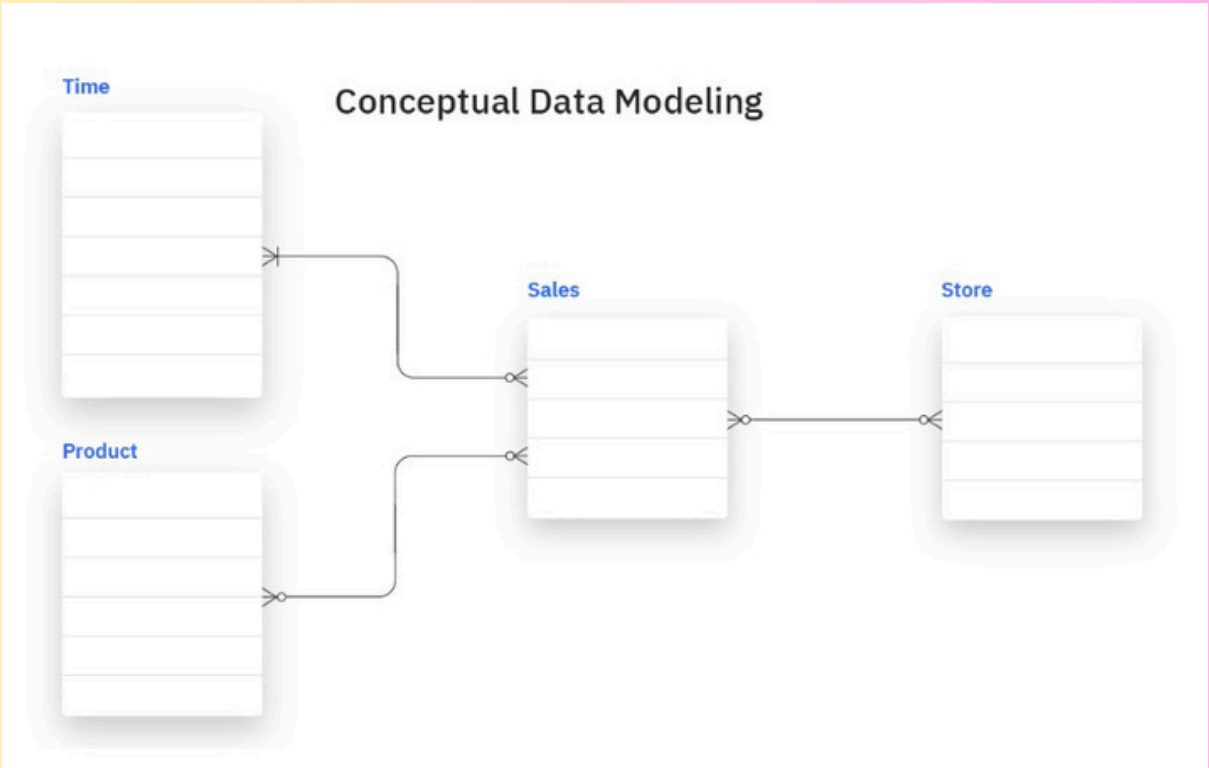
What are the types of data models?



Data modeling usually starts with creating a big-picture view of the data, known as a conceptual data model. This is like a high-level map that shows what data your business has and how it should be organized. It's a tool that helps everyone—from business stakeholders to technical teams—get on the same page about what the data project is all about.

What is a Conceptual Data Model?

A conceptual data model is the first step in understanding your data. It gives an overview of what data your system will handle, how that data is related, and what rules or conditions apply. This model doesn't dive into the technical details; instead, it's more about making sure everyone understands the basics and agrees on the direction of the project.



Follow Me
https://www.linkedin.com/in/dhwanimehta05/

Understanding Conceptual Data Modeling

The diagram above is an example of a **Conceptual Data Model**. This type of model gives a high-level view of the data your business deals with, showing the main entities (or data categories) and how they are related to each other.

What is the Conceptual Data Model Showing?

In this diagram, we see four key entities:

Time This reprsents any data related to dates and time periods. For instance, it could include information like specific dates, months, and years.

Product This entity captures details about the products your business offers. It could include attributes like product names, categories, and descr iptions.

Sales This entity ties together other entities like Product, Time, and Store. It represents the sales transactions, showing which products were sold, where they were sold, and when.

Store This represents the locations where the sales occur. It might include data like store names, locations, and other relevant details.

How Do These Entities Relate?

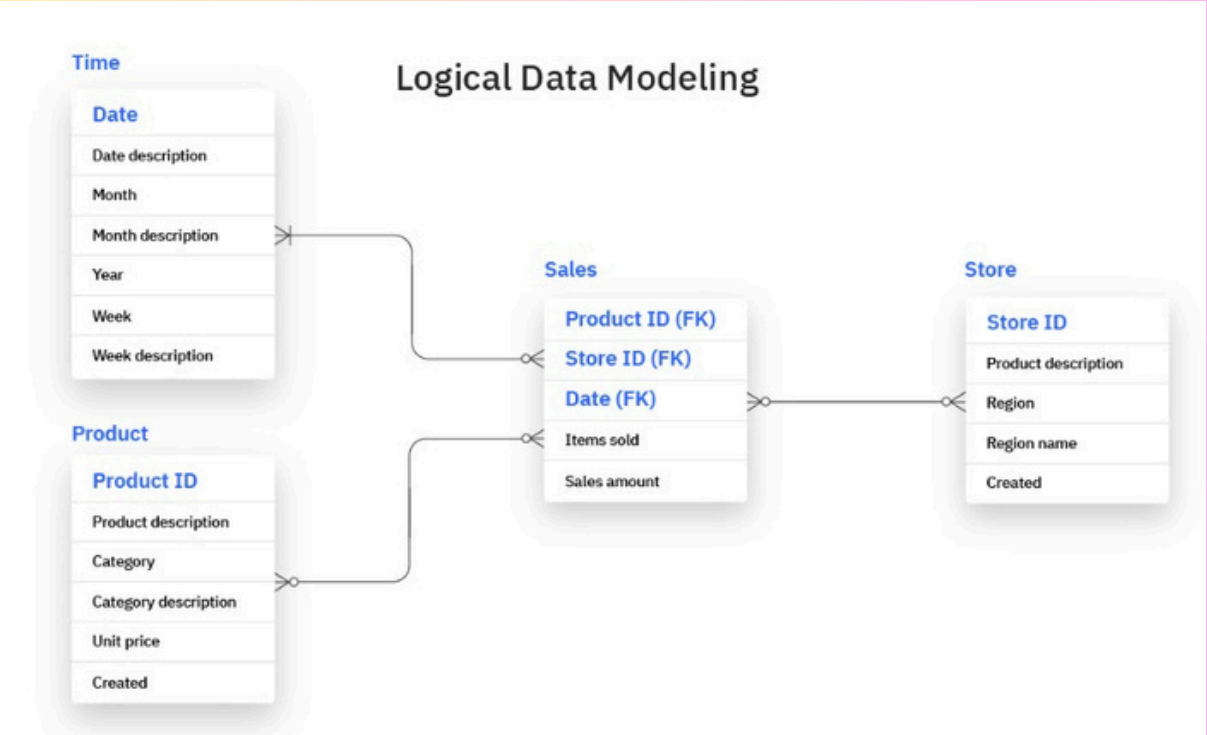
- **Sales to Product** The relationship here indicates that sales records are linked to specific products. For example, every sale involves a particular product.
- **Sales to Time** This shows that each sale is connected to a specific date or time period. This helps in tracking when the sales occur.
- **Sales to Store** This relationship signifies that each sale happens at a specific store. So, the sales entity links the product being sold, the time it was sold, and the store where the transaction took place.

Why is This Important?

The conceptual data model is crucial because it helps you visualize the overall structure of your business's data without getting into too many technical details. It's like a blueprint that ensures everyone—both technical and non-technical teams—understands what data is important and how it should be organized.

By mapping out the main entities and their relationships, the conceptual model provides a clear picture of how data will flow through your system, setting a solid foundation for more detailed data modeling down the line.

2 Understanding Logical Data Modeling



When it comes to data modeling, the logical data model is the next step after the conceptual data model. It takes the big-picture ideas and details from the conceptual model and adds more specifics, helping to map out how data should be structured technically.

What Does a Logical Data Model Do?

In a logical data model, you go deeper into the details:

- **Data Types:** You define what kind of data each piece of information is—like whether it's a string of text, a number, or a date.
- **Relationships:** You clarify how different pieces of data are connected, such as how a product is linked to a sale or how a date is tied to a store's records.

- **Key Fields:** You identify important data points, known as **primary keys**, which are essential for organizing and retrieving data efficiently.

How is it Created?

Data architects and analysts work together to build the logical data model. They use formal systems to ensure everything is clearly defined and organized. While some agile teams might skip this step and jump straight to the physical model, the logical model is particularly useful for designing complex databases like data warehouses or automatic reporting systems.

Example Using the Diagram

Let's look at the example from the diagram you provided:

- **Time (Date):** Includes details like the specific date, month, year, and even descriptions for each week.
- **Product:** Contains information about each product, like its description, category, and price.
- **Sales:** Shows the connection between products, stores, and dates—linking what was sold, where it was sold, and when it was sold.
- **Store:** Details about each store, such as its name, region, and creation date.
-

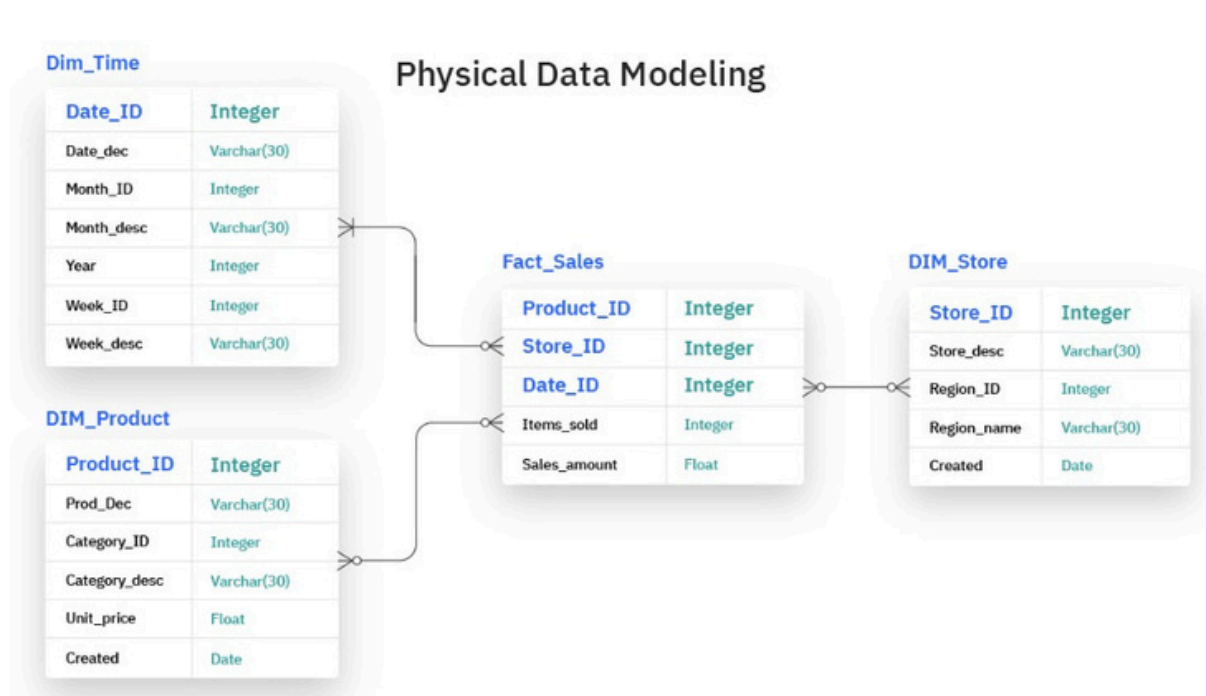
In this logical model, each entity (like Product or Store) is connected to others in a way that reflects real-world relationships. For instance, each sale is tied to a specific product and store, and it happens on a certain date. This structure is more detailed than the conceptual model and helps set the stage for the final step—creating the physical database.

Why is it Important?

The logical data model is crucial because it acts as a bridge between the big ideas (conceptual model) and the actual technology that will store and manage the data (physical model). It's flexible enough to be used with any database system, but it's specific enough to guide developers in building the database correctly.

In summary, the logical data model is all about adding more detail and structure to your data plan, ensuring that when it comes time to build the database, everything is clearly mapped out and ready to go.

Understanding Physical Data Modeling



The final step in the data modeling process is the **Physical Data Model**, which is where all the planning and designing from the conceptual and logical models come to life. This model is created by database administrators and developers to fit the specific database tools and storage technologies your business uses.

What Does the Physical Data Model Show?

In the diagram you've provided, the physical data model takes the logical structure and maps it directly to the specific technology your business will use. Let's break it down:

- **DIM_Time, DIM_Product, and DIM_Store** These represent different dimensions or categories of data. For example:
  - **DIM_Time** Manages all the time-related data, such as dates, months, and years. Each attribute is defined with a specific data type like `Integer` for numerical data or `Varchar` for text data.
  - **DIM_Product** Contains product-related information, such as product descriptions, categories, and prices.
  - **DIM_Store** Holds data about the stores, including store descriptions and regional details.

Follow Me
https://www.linkedin.com/in/dhwanimehta05/

- **Fact_Sales** This is the central table that ties everything together. It links to the DIM_Time, DIM_Product, and DIM_Store tables through fields like `Product_ID`, `Store_ID`, and `Date_ID`, representing the actual sales data, such as the number of items sold and the total sales amount.

How is it Created?

When creating a physical data model, data engineers and developers take the logical model and apply it to the chosen database management system (DBMS). This means:

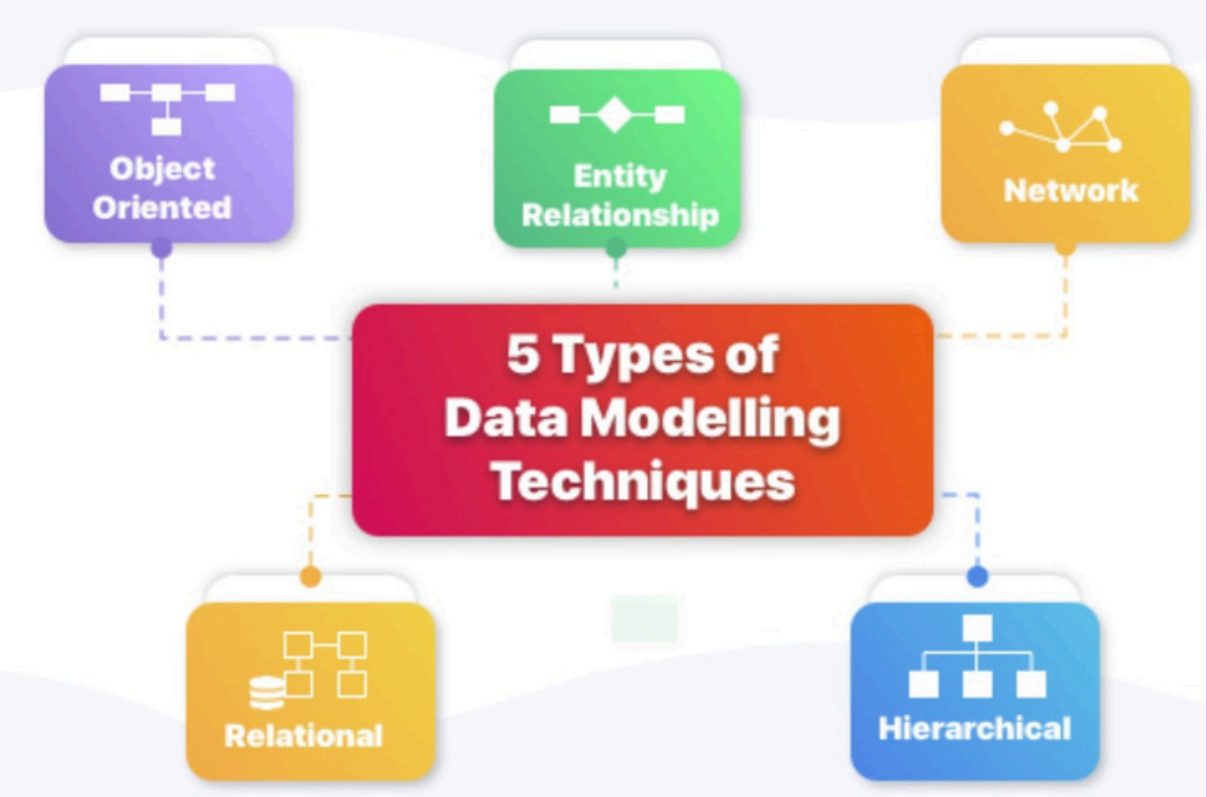
- Specifying exact data types that the DBMS supports, such as `Integer` for whole numbers, `Float` for decimal numbers, or `Date` for date fields.
- Defining the relationships between different data entities, ensuring that they reflect the actual data storage technology.
- Adding details for performance tuning, like indexing certain fields to make data retrieval faster.

Why is the Physical Data Model Important?

The physical data model is crucial because it's the actual implementation of your data structure. While the conceptual and logical models are all about planning and designing, the physical model is where those plans turn into reality. It ensures that the data is stored efficiently, is easily accessible, and works smoothly with your business systems.

For instance, in the diagram, the `Fact_Sales` table shows exactly how sales data is linked to specific products, stores, and dates. This precise implementation is what allows your business to track and analyze sales performance accurately.

5 Types of Data Modelling Techniques



Let’s break down the different types of data models using simple terms and examples in SQL to make them easy to understand.

1. Hierarchical Data Models

Imagine a family tree where each person (child) has only one parent, but a parent can have multiple children. This is what a hierarchical data model looks like—it's a treelike structure with one-to-many relationships.

2. Relational Data Models

Relational data models are like a set of interconnected tables. Each table represents an entity (like Customers, Products, Orders) and they are linked through keys (like `CustomerID`, `ProductID`, etc.).

This is the most common model in SQL databases, where tables are related to each other using foreign keys.
Example:

Follow Me
https://www.linkedin.com/in/dhwanimehta05/

```
CREATE TABLE Customers (

    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    Email VARCHAR(100)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY, OrderDate DATE, CustomerID
    INT, FOREIGN KEY (CustomerID) REFERENCES
    Customers(CustomerI

D
)
);
```

Here, each customer can place many orders, and each order is linked back to a customer using `CustomerID`. This model is used in everyday applications like online shopping or bank transaction systems.

3. Entity-Relationship (ER) Data Models

ER models are visual tools that help design databases by showing entities (like Customers, Products) and how they relate to each other. Think of it as a blueprint for your database.

In SQL, this is more of a planning stage, but let's imagine it like this:

- **Entities** Tables like `Customers`, `Products`.
- **Relationships**The links between these tables, such as an order connecting a customer to a product.

Example:

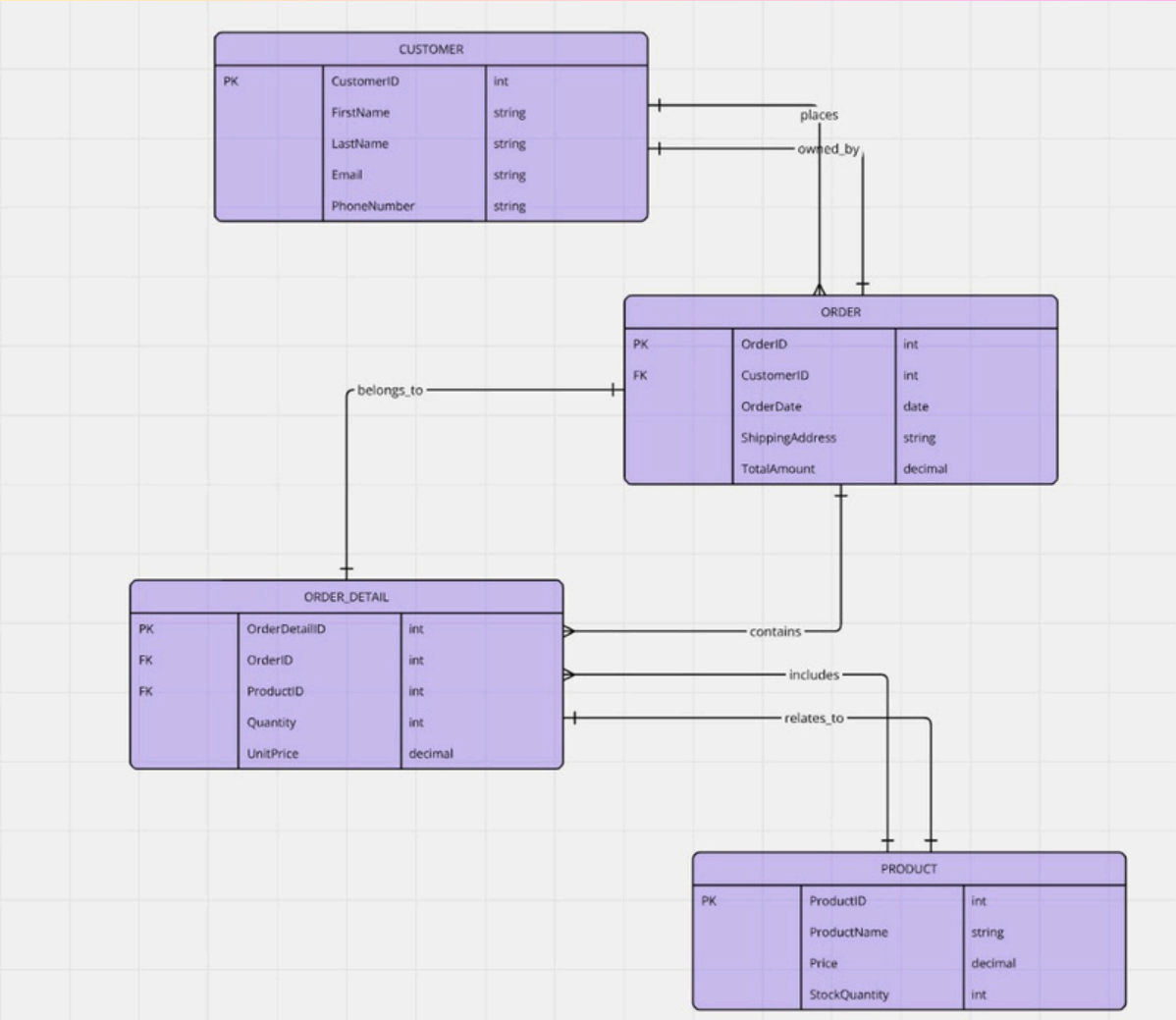
```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100)
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(100)
);
```


```
-- Relationship
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY, OrderDate DATE, CustomerID
    INT, ProductID INT, FOREIGN KEY (CustomerID)
    REFERENCES Customers(CustomerI


D),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

This structure allows you to visually map out and then implement the relationships in SQL.



Follow Me
https://www.linkedin.com/in/dhwanimehta05/

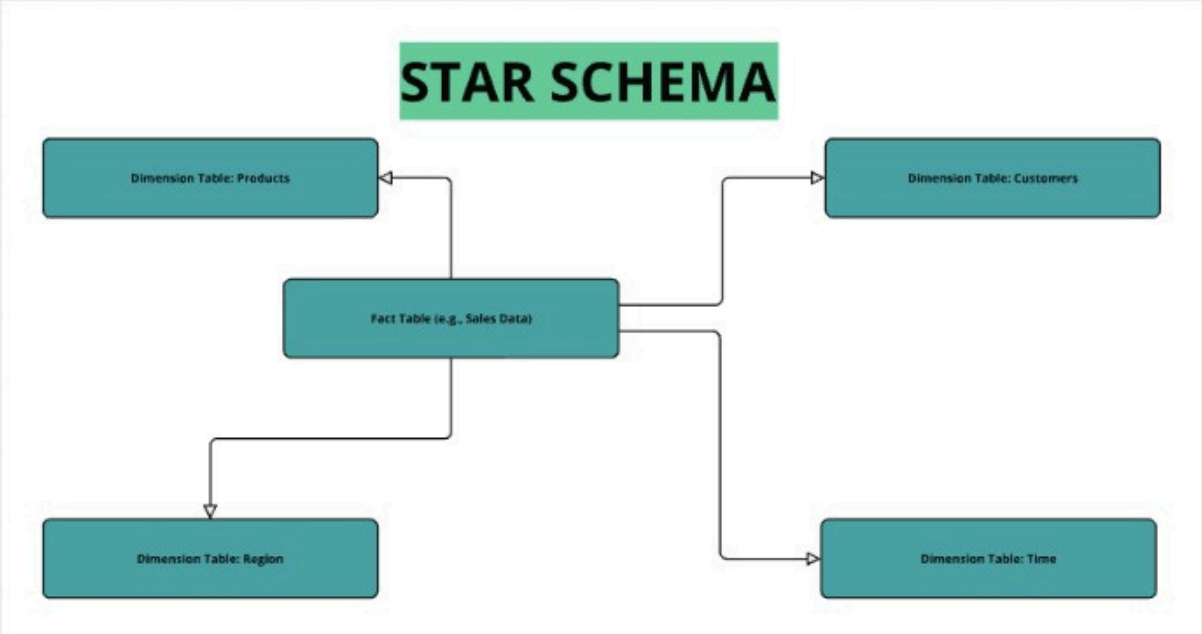
4. Object-Oriented Data Models

Object-oriented models treat data as “objects,”” similar to how object-oriented programming languages like Java or Python do. Each object represents a real-world entity, and these objects can contain both data and methods (actions).

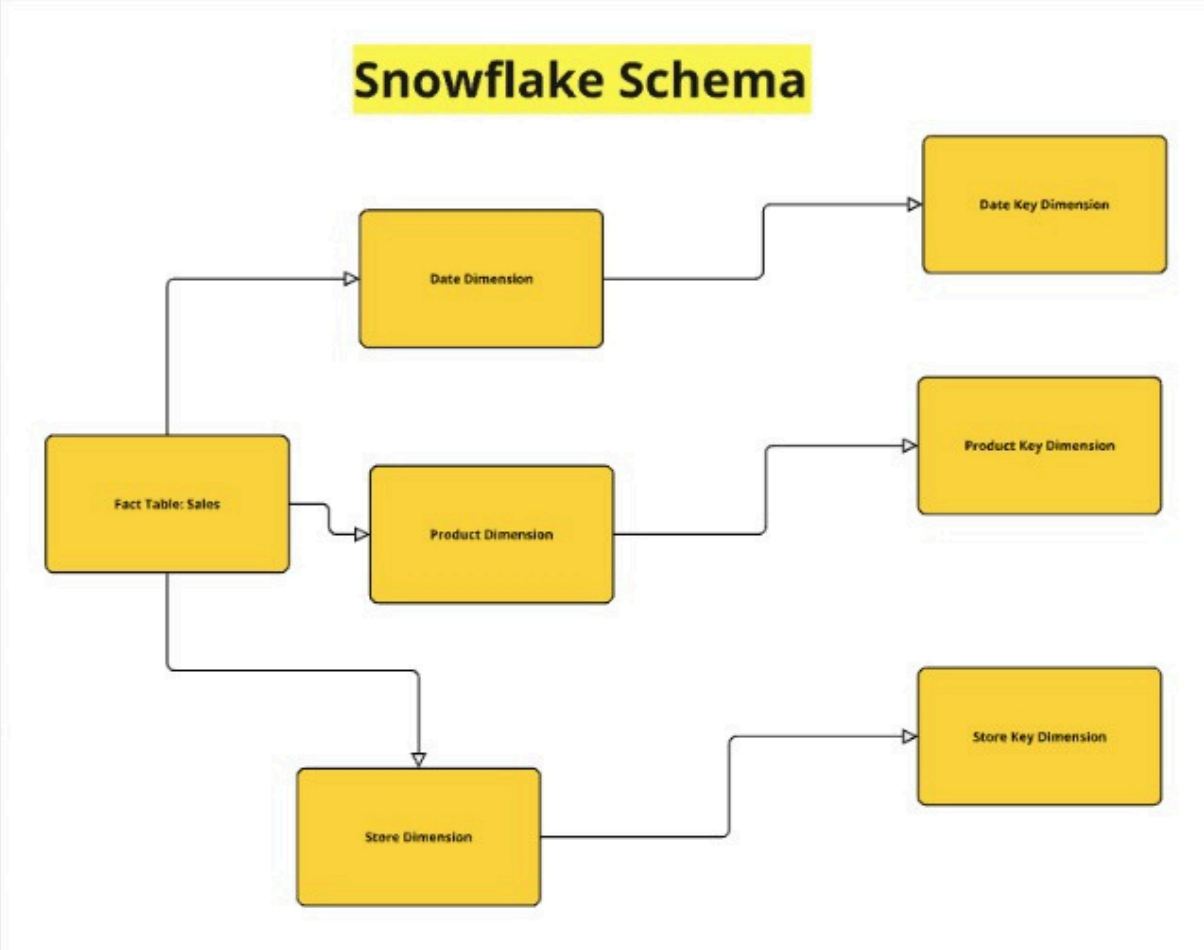
5. Dimensional Data Models

Dimensional models are designed to make data retrieval quick and easy, especially in data warehouses where large amounts of data are analyzed. These models often increase redundancy to speed up query performance.

- **Star Schema** The simplest form, where you have a central fact table (e.g., sales data) connected to dimension tables (e.g., products, customers).
- **Snowflake Schema** A more complex version of the star schema where dimension tables are normalized into multiple related tables, making the schema look like a snowflake.
-



Follow Me
https://www.linkedin.com/in/dhwanimehta05/



Data modeling in SQL involves creating a structured representation of a database, defining how data is stored, organized, and accessed. Here are the key concepts and steps involved:

Key Concepts

Entity-Relationship Model (ER Model):

- **Entities** Objects or concepts, often corresponding to real-world nouns (e.g., Customer, Order).
- **Attributes** Characteristics of an entity (e.g., CustomerName, OrderDate).

- **Relationships** Associations between entities (e.g., A customer places an order).

Normalization:

- **1NF (First Normal Form):** Ensuring that each column contains atomic (indivisible) values and that each column has a unique name.
- **2NF (Second Normal Form)** Achieving 1NF and ensuring that all non-key attributes are fully functional dependencies on the primary key.
- **3NF (Third Normal Form)** Achieving 2NF and ensuring that all attributes are not only fully functional dependencies on the primary key but also non-transitive (no indirect relationships).

Primary Key:

- A unique identifier for each record in a table (e.g., CustomerID).

Foreign Key:

- A field in one table that uniquely identifies a row of another table, creating a relationship between the two tables (e.g., Order.CustomerID).

Indexes:

- Structures that improve the speed of data retrieval operations on a database table at the cost of additional storage space and slower wr ites.

Joins:

- Methods of combining rows from two or more tables based on related columns (e.g., INNER JOIN, LEFT JOIN).

Data Types:

- Define the type of data that can be stored in each column (e.g., INTEGER, VARCHAR, DATE).

Constraints:

- Rules enforced on data columns, such as NOT NULL, UNIQUE, CHECK, and FOREIGN KEY constraints.

Steps in Data Modeling Using SQL

Requirement Gathering:

- Understand the business requirements, the scope of the database, and what data needs to be stored.

Conceptual Design:

- Create an Entity-Relationship Diagram (ERD) to visually represent entities, attributes, and relationships.
- Identify entities and their attributes.
- Define relationships between entities (one-to-one, one-to-many, many-to-many).

Logical Design:

- Convert the ERD into a relational schema.
- Define primary keys for each entity.
- Apply normalization to organize data efficiently and avoid redundancy.
- Define foreign keys to establish relationships between tables.
- Determine the data types for each attribute.

Physical Design:

- Create SQL scripts to define tables, relationships, and constraints.
- Define indexes to optimize query performance.
- Consider storage requirements and indexing strategies.

Implementation:

- Execute SQL scripts in the database management system (DBMS) to create tables and relationships.
- Populate the tables with data, if necessary.

Testing and Validation:

- Test the database model to ensure it meets the requirements.
- Run queries to validate the correctness of relationships and constraints.

Optimization:

- Analyze query performance and optimize by creating indexes or denormalizing certain tables if needed.
- Review and refine data types, indexes, and constraints.

Maintenance:

- Monitor the database for performance, scalability, and integrity issues.
- Update the model as business requirements evolve.

Let's go through the steps of data modeling for a retail store that make it easy to understand.

1. Understanding What the Store Needs (Requirement Gathering)

Imagine you're helping a small retail store set up a database. The store sells products, has customers, and takes orders. To organize this information, we need to know:

- What products are sold?
- Who are the customers?
- What orders have been placed?

2. Drawing a Simple Picture (Conceptual Design)

Next, we create a basic picture of how everything is connected. This is like a simple map showing the main parts of the store's data:

- **Products:** These are the items the store sells, like "T-shirts," "Jeans," etc.
- **Customers:** These are the people who buy things, like "John Doe" or "Jane Smith."
- **Orders:** These are records of what customers bought.

We also need to know how these parts are connected:

- A customer can place many orders.
- Each order can include many products.

3. Organizing the Information (Logical Design)

Now we take our picture and organize the information into tables:

- **Products Table :** We create a list of all the products. Each product has details like a name, price, and product ID (a unique number to identify each product).

ProductID	ProductName	Pr ice
1	T-shirt	$10
2	Jeans	$30

- **Customers Table:** We create a list of all customers. Each customer has a unique ID, name, and maybe an email.

CustomerID	CustomerName	Email
1	John Doe	john@example.com
2	Jane Smith	jane@example.com

- **Orders Table :** We track each order, including the customer who made it and the date of the order.

OrderID	CustomerID	OrderDate
1	1	2024-08-01
2	2	2024-08-02

- **OrderDetails Table** Since each order can have many products, we need another table to list the products in each order.

OrderID	ProductID	Quantit y
1 1 2	1	2 1 1
	2	
	1	

4. Putting It All Together (Physical Design)

We now use SQL Structured Query Language) to create these tables in a database. SQL helps us set up the structure so the database knows how to store and connect the information.

Here's an example of SQL to create the Products table:

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
```


```
    ProductName VARCHAR(50),
    Price DECIMAL(5, 2)
);
```

We do similar things for the Customers, Orders, and OrderDetails tables.

5. Adding the Data (Implementation)

Once the tables are ready, we start adding the store's data. For example, we add products, customers, and the orders they placed:

```
INSERT INTO Products (ProductID, ProductName, Price) VALUES
(1, 'T-shirt', 10);
INSERT INTO Customers (CustomerID, CustomerName, Email) VAL
UES (1, 'John Doe', 'john@example.com');
INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES
(1, 1, '2024-08-01');
```

6. Making Sure Everything Works (Testing and Validation)

We check that everything is working correctly. For example, we might run a test to see if we can find all the orders placed by John Doe:

```
SELECT * FROM Orders WHERE CustomerID = 1;
```

7. Speeding Things Up (Optimization)

If the store starts growing and there are a lot of orders, we might need to make things faster. We can create indexes to help the database quickly find the data it needs, like a table of contents in a book.

These are additional techniques that can make your database more powerful and easier to use.

- **Views:** A way to simplify complex queries by creating a virtual table.
    - Example: You could create a view that shows all the information about an order (customer name, product details) without needing to write a complex query every time.

- **Stored Procedures and Functions:** Pre-written SQL code that can be reused to perform common tasks.
    - Example: You might write a stored procedure to automatically calculate the total cost of an order including taxes and discounts.
- **Data Partitioning:** Splitting large tables into smaller pieces to improve per for mance.
    - Example: If you have millions of orders, you might partition the Orders table by year so that the database can quickly access the relevant part.

Once your database is running, you need to ensure it works efficiently, especially as it grows.

- **Indexing:** Creating indexes on the most commonly searched columns can speed up query performance.
    - Example: If users frequently search for orders by date, creating an index on the OrderDate column will make those searches faster.
- **Query Optimization:** Fine-tuning SQL queries to run faster.
    - Example: Instead of retrieving all columns in a table, retrieve only the ones you need to reduce the amount of data processed.

8. Keeping It Running Smoothly (Maintenance)

Finally, we keep an eye on the database to make sure it continues to work well as the store grows. If the store starts selling new products or changes how it operates, we update the database to match.

For that , you need to document what you've done and keep improving it.

- **Document the Data Model:** Write down details about your tables, relationships, and any special rules or logic.
    - Example: Create a document that explains what each table and column represents, so others can understand and use the database correctly.
- **Review and Iterate:** Regularly check the database design to ensure it still meets the business needs and make changes as needed.
    - Example: If the business starts selling new types of products, you might need to add new tables or columns to handle the new data.