# Data Modelling

* Data modeling is a crucial step in the process of designing a database.
* It involves creating a conceptual representation of the data to be stored, organized, and managed in a database system.
* The goal of data modeling is to ensure that the database is well-structured, efficient, and capable of supporting the required business processes and data analysis.
* Data modeling is a critical skill for database developers, administrators, and analysts, as it lays the foundation for building robust and efficient database systems that meet the needs of the organization.

## From Requirements to Database: Crafting 3 Essential Models

### Conceptual Data Modeling:

* Conceptual data modeling is the first step in designing a database. It involves creating a high-level representation of the data to be stored in the database, without considering the specific details of how the data will be stored or implemented.
* The focus is on identifying the key entities (objects or concepts about which data is stored) and their relationships, along with the attributes (properties or characteristics) of each entity.
* The resulting conceptual model provides a clear, visual representation of the data requirements and serves as a basis for further refinement.

#### **Here are some key insights into conceptual data modeling:**

* **Focus on Business Concepts:**
  + Conceptual data modeling is centered around the business perspective rather than technical details.
  + It aims to represent the data requirements of the organization in a way that is easily understood by both technical and non-technical stakeholders.
* **Key Elements:**
  + The main elements of a conceptual data model include entities, attributes, and relationships.
  + Entities represent the main objects or concepts in the business domain, attributes describe the properties or characteristics of entities, and relationships define how entities are related to each other.
* **Abstraction:**
  + The conceptual data model is abstract and does not include specific details such as data types, keys, or constraints.
  + It provides a high-level overview of the data requirements without getting into the technical implementation details.
* **Communication Tool:**
  + Conceptual data models serve as a communication tool between business stakeholders and data modelers.
  + They help to clarify and document the data requirements and ensure that all stakeholders have a common understanding of the data.
* **Requirements Gathering:**
  + Conceptual data modeling is often used in the early stages of a project to gather and document the business requirements.
  + It helps to identify the key entities and relationships that need to be included in the database design.
* **Iterative Process:**
  + Conceptual data modeling is an iterative process that evolves as the understanding of the business domain grows.
  + It may involve multiple iterations of refining the model based on feedback from stakeholders.
* **Tool Support:**
  + There are several tools available for creating conceptual data models, such as ER (Entity-Relationship) diagrams.
  + These tools provide a graphical representation of the data model, making it easier to understand and communicate.
* **Foundation for Logical Design:**
  + The conceptual data model serves as the foundation for the logical data model, which translates the conceptual model into a more detailed representation that can be implemented in a database management system.

### Logical Data Modeling:

* Once the conceptual model is in place, the next step is to translate it into a logical data model that can be implemented in a database management system (DBMS).
* The logical data model defines the structure of the database in terms of tables, columns, keys, and relationships between tables. It also includes data types, constraints, and other details necessary for implementation.
* Normalization is an important aspect of logical data modeling, which involves organizing the data in such a way that redundancy is minimized, and data integrity is ensured.

#### **Here are some key insights into logical data modeling:**

* **Translating Conceptual Model:**
  + The logical data model translates the entities, attributes, and relationships from the conceptual model into tables, columns, and keys that can be implemented in a database.
  + It adds more detail and specificity to the data model.
* **Normalization:**
  + A key aspect of logical data modeling is normalization, which is the process of organizing the data in a database to minimize redundancy and dependency.
  + Normalization helps to ensure data integrity and efficient storage.
* **Entity-Relationship Diagrams (ERDs):**
  + ERDs are commonly used to represent the logical data model.
  + They visually depict the entities, attributes, and relationships in the data model, making it easier to understand and communicate.
* **Data Types and Constraints:**
  + In the logical data model, data types and constraints are specified for each attribute.
  + Data types define the kind of data that can be stored in a column (e.g., integer, string), while constraints enforce rules for the data (e.g., unique, not null).
* **Keys:**
  + Keys are used to uniquely identify rows in a table.
  + In the logical data model, primary keys and foreign keys are defined to establish relationships between tables.
* **Normalization Levels:**
  + There are several levels of normalization (e.g., 1NF, 2NF, 3NF) that a logical data model can achieve.
  + Each level represents a higher degree of normalization and reduces redundancy in the database design.
* **Performance Considerations:**
  + While designing the logical data model, performance considerations such as indexing, partitioning, and denormalization may be taken into account to optimize query performance and data retrieval.
* **Data Integrity:**
  + Ensuring data integrity is a key goal of logical data modeling.
  + By defining constraints and relationships between tables, data integrity can be enforced at the database level.
* **Iteration and Refinement:**
  + Like conceptual data modeling, logical data modeling is an iterative process.
  + It may involve multiple iterations of refining the model based on feedback from stakeholders and changes in the business requirements.

### Physical Data Modeling:

* Physical data modeling is the process of implementing the logical data model in a specific DBMS. It involves translating the logical model into a physical database schema that can be used to create the actual database.
* This step includes defining storage structures, indexing strategies, partitioning schemes, and other physical aspects of the database that affect performance and storage efficiency.
* The physical data model is optimized for the specific DBMS being used and takes into account factors such as performance requirements, scalability, and hardware constraints.

#### **Here are some key insights into physical data modeling:**

* **DBMS-specific Design:**
  + Physical data modeling is specific to the DBMS being used (e.g., Oracle, SQL Server, MySQL).
  + It takes into account the features and limitations of the DBMS to optimize the database design for that system.
* **Storage Structures:**
  + Physical data modeling defines the storage structures used to store data in the database, such as tables, indexes, and partitions.
  + It includes decisions about how data is stored on disk and in memory.
* **Indexes:**
  + Indexing is an important aspect of physical data modeling.
  + Indexes are used to speed up data retrieval by creating efficient access paths to the data.
  + Physical data modeling determines which columns should be indexed and the type of indexes to use (e.g., B-tree, hash).
* **Partitioning:**
  + Partitioning involves dividing large tables into smaller, more manageable parts called partitions.
  + Physical data modeling determines the partitioning strategy based on factors such as data volume, query patterns, and performance requirements.
* **Denormalization:**
  + In some cases, physical data modeling may involve denormalization, which is the process of adding redundant data to improve query performance.
  + Denormalization should be carefully considered to avoid data redundancy and maintain data integrity.
* **Performance Considerations:**
  + Physical data modeling considers various performance factors, such as query optimization, disk I/O, and memory usage.
  + It aims to design the database in a way that maximizes performance and minimizes resource consumption.
* **Data Integrity and Security:**
  + Physical data modeling includes considerations for enforcing data integrity and security.
  + This may involve defining constraints, permissions, and encryption methods to protect the data.
* **Data Migration and Upgrades:**
  + Physical data modeling should consider how data will be migrated from existing systems and how the database will be upgraded in the future.
  + This includes ensuring compatibility with future versions of the DBMS.
* **Documentation:**
  + Physical data modeling should be well-documented to provide guidelines for database administrators and developers.
  + It should include details about table structures, indexes, partitions, and other physical aspects of the database design.
* **Maintenance and Monitoring:**
  + After the database is implemented, physical data modeling continues with ongoing maintenance and monitoring.
  + This includes monitoring performance, tuning the database for optimal performance, and making adjustments as needed.

## Normalization Types:

* Normalization is the process of organizing data in a database to reduce redundancy and dependency. There are several different levels of database normalization. Let’s apply normalization to the example of a company’s employee management system, which includes departments, employees, and employee salary payments.

### **Entities:**

* Department (DepartmentID, DepartmentName)
* Employee (EmployeeID, EmployeeName, DateOfBirth, DepartmentID)
* Employee\_Salary\_Payments (PaymentID, PaymentDate, Amount, EmployeeID)

### **First Normal Form (1NF):**

* Ensure that each table has a primary key.
* Ensure that each column contains atomic values.
* Example:
  + Employee table (1NF): EmployeeID (PK), EmployeeName, DateOfBirth, DepartmentID (FK)
  + Department table (1NF): DepartmentID (PK), DepartmentName
  + Employee\_Salary\_Payments table (1NF): PaymentID (PK), PaymentDate, Amount, EmployeeID (FK)

### **Second Normal Form (2NF):**

* Meet the requirements of 1NF.
* Ensure that no partial dependencies exist; all non-key attributes depend on the entire primary key.
* Example:
  + Employee table (2NF): EmployeeID (PK), EmployeeName, DateOfBirth, DepartmentID (FK)
  + Department table (2NF): DepartmentID (PK), DepartmentName
  + Employee\_Salary\_Payments table (2NF): PaymentID (PK), PaymentDate, Amount, EmployeeID (FK)

### **Third Normal Form (3NF):**

* Meet the requirements of 2NF.
* Ensure that no transitive dependencies exist; all non-key attributes depend only on the primary key.
* Example:
  + Employee table (3NF): EmployeeID (PK), EmployeeName, DateOfBirth, DepartmentID (FK)
  + Department table (3NF): DepartmentID (PK), DepartmentName
  + Employee\_Salary\_Payments table (3NF): PaymentID (PK), PaymentDate, Amount, EmployeeID (FK)

### **Explanation:**

* In 1NF, we ensure that each column contains atomic values and there are no repeating groups.
* In 2NF, we remove partial dependencies by moving non-key attributes to separate tables.
* In 3NF, we eliminate transitive dependencies by moving attributes that do not depend on the primary key to separate tables.
* Example:
  + Suppose we have the following data:
    - Department: {1, ‘Sales’}, {2, ‘Marketing’}
    - Employee: {1, ‘Alice’, ‘1990-01-01’, 1}, {2, ‘Bob’, ‘1995-05-05’, 2}
    - Employee\_Salary\_Payments: {1, ‘2022-01-01’, 3000, 1}, {2, ‘2022-02-01’, 3500, 2}
  + After normalization, the tables would be:
    - Department: {1, ‘Sales’}, {2, ‘Marketing’}
    - Employee: {1, ‘Alice’, ‘1990-01-01’}, {2, ‘Bob’, ‘1995-05-05’}
    - Employee\_Department: {1, 1}, {2, 2}
    - Salary\_Payments: {1, ‘2022-01-01’, 3000, 1}, {2, ‘2022-02-01’, 3500, 2}

## Denormalization Types:

* Denormalization is the process of intentionally introducing redundancy into a database schema to improve performance by reducing the need for joins and simplifying query processing. Here’s how denormalization can be applied to the entities Department, Employee, and Employee\_Salary\_Payments:

### **Flattening Hierarchical Data:**

* Flattening hierarchical data refers to the process of transforming nested or hierarchical data structures into a flat representation.
* Original Schema:
  + Department (DepartmentID, DepartmentName)
  + Employee (EmployeeID, EmployeeName, DepartmentID)
  + Employee\_Salary\_Payments (PaymentID, PaymentDate, Amount, EmployeeID)
* Denormalized Schema:
  + Employee\_Salary\_Payments\_Denorm (PaymentID, PaymentDate, Amount, EmployeeID, EmployeeName, DepartmentID, DepartmentName)

### **Aggregating Data:**

* Aggregating data refers to the process of combining and summarizing multiple data records into a single value.
* Original Schema:
  + Employee\_Salary\_Payments (PaymentID, PaymentDate, Amount, EmployeeID)
* Denormalized Schema:
  + Employee\_Total\_Payments (EmployeeID, EmployeeName, TotalAmount)

### **Duplicating Data for Performance:**

* Duplicating data for performance involves intentionally creating redundant copies of data in a database to improve query performance.
* Original Schema:
  + Employee\_Salary\_Payments (PaymentID, PaymentDate, Amount, EmployeeID)
* Denormalized Schema:
  + Employee\_Salary\_Payments\_Cached (PaymentID, PaymentDate, Amount, EmployeeID, EmployeeName, DepartmentID)

### **Example:**

* Suppose we have the following data:
  + Department: {1, ‘Sales’}, {2, ‘Marketing’}
  + Employee: {1, ‘Alice’, 1}, {2, ‘Bob’, 2}
  + Employee\_Salary\_Payments: {1, ‘2022-01-01’, 3000, 1}, {2, ‘2022-02-01’, 3500, 2}
* Flattening Hierarchical Data Example (1):
  + Employee\_Salary\_Payments\_Denorm: {1, ‘2022-01-01’, 3000, 1, ‘Alice’, 1, ‘Sales’}, {2, ‘2022-02-01’, 3500, 2, ‘Bob’, 2, ‘Marketing’}
* Aggregating Data Example (2):
  + Employee\_Total\_Payments: {1, ‘Alice’, 3000}, {2, ‘Bob’, 3500}
* Duplicating Data for Performance Example (3):
  + Employee\_Salary\_Payments\_Cached: {1, ‘2022-01-01’, 3000, 1, ‘Alice’, 1}, {2, ‘2022-02-01’, 3500, 2, ‘Bob’, 2}

## OLTP

* OLTP (Online Transaction Processing) data modeling is the process of designing a database to support transaction-oriented applications, typically for day-to-day operations in businesses.
* The goal is to create a structure that efficiently handles a large volume of transactions while ensuring data integrity and supporting concurrent access by multiple users.
* Let’s illustrate OLTP operations using the example of a company’s department, employee, and employee salary payments.

### OLTP Operations:

* **Insert:** Adding a new employee record to the Employee table when a new employee is hired.
* **Update:** Modifying an employee’s salary in the Employee\_Salary\_Payments table when they receive a raise.
* **Delete:** Removing an employee record from the Employee table when they leave the company.
* **Select:** Retrieving information such as an employee’s salary history or department details for reporting or analysis.

## OLAP

* OLAP (Online Analytical Processing) is a technology used to organize and analyze large volumes of data from multiple perspectives.
* It allows users to perform complex analysis, such as trend analysis, forecasting, and data mining, on data stored in a data warehouse or data mart.

### OLAP Models:

* **Multidimensional Model:** In this model, data is organized into dimensions (e.g., time, product, geography) and measures (e.g., sales, profit). Users can analyze data by selecting dimensions and measures of interest.
* **Relational Model:** This model uses a relational database to store data and SQL queries to perform OLAP operations. It is suitable for smaller datasets or scenarios where a separate OLAP server is not feasible.
* **Hybrid Model:** Combines elements of both the multidimensional and relational models to provide flexibility and scalability.

### OLAP Operations:

* **Slice:** Selecting a subset of data from a single dimension. For example, viewing salary payments for a specific department.
* **Dice:** Selecting a subset of data from multiple dimensions. For example, viewing salary payments for a specific department and year.
* **Pivot:** Rotating the data axes to view data from a different perspective. For example, pivoting the data to compare salary payments by department and year.
* **Drill Down/Up:** Navigating through the data hierarchy to view data at different levels of detail. For example, drilling down from department-level to employee-level salary payments.

## ODS

* ODS stands for Operational Data Store. It is a type of database that is used for operational reporting and decision-making.
* An ODS collects, cleanses, and integrates data from multiple sources in real-time or near-real-time, providing a consistent and integrated view of operational data.

### Characteristics of an ODS:

* **Real-time Data Integration:** An ODS integrates data from various operational systems in real-time or near-real-time, providing up-to-date information.
* **Data Cleansing:** ODS typically performs data cleansing and transformation to ensure data quality and consistency.
* **Support for Operational Reporting:** An ODS is optimized for operational reporting and analysis, providing users with access to current and historical data.
* **Integration with Data Warehouses:** An ODS can serve as a staging area for data warehouses, providing cleansed and integrated data for further analysis.
* **Transactional and Analytical Processing:** An ODS supports both transactional and analytical processing, allowing users to perform ad-hoc queries and analysis on operational data.
* **Incremental Data Updates:** ODS supports incremental updates, allowing it to capture and process only the changes that occur in the source systems.

### Example of an ODS:

In a retail business, an ODS can be used to integrate data from various operational systems such as point-of-sale (POS) systems, inventory management systems, and customer relationship management (CRM) systems. The ODS can provide a consolidated view of sales data, inventory levels, and customer information, which can be used for operational reporting and analysis.

### Benefits of an ODS:

* Provides a unified view of operational data from multiple sources.
* Supports real-time or near-real-time data integration.
* Improves data quality and consistency through data cleansing and transformation.
* Supports operational reporting and decision-making.

## Staging Area

* A staging area, in the context of data management and data warehousing, is a temporary storage area where data is prepared, processed, and transformed before being loaded into a data warehouse or operational data store (ODS).
* The staging area serves as an intermediary step between the source systems and the target database, allowing for data cleansing, validation, and transformation.

### Characteristics of a Staging Area:

* **Data Transformation:** Data in the staging area is transformed to meet the requirements of the target database, such as standardizing formats, cleaning up data inconsistencies, and deriving new fields.
* **Data Aggregation:** Staging areas can aggregate data from multiple sources before loading it into the target database, reducing the complexity of queries in the target database.
* **Data Quality Checks:** Staging areas often perform data quality checks to ensure that the data being loaded into the target database is accurate and complete.
* **Data Integration:** Staging areas integrate data from various source systems, allowing for a unified view of the data before loading it into the target database.
* **Data Storage:** Staging areas provide temporary storage for data, typically in a raw or semi-structured format, before it is processed and loaded into the target database.

### Example of a Staging Area:

In a retail business, a staging area may be used to collect data from various sources such as point-of-sale (POS) systems, online sales platforms, and customer databases. The data in the staging area may undergo transformations to standardize product codes, clean up customer information, and aggregate sales data before being loaded into the data warehouse for analysis.

## Data Injestion

* Data ingestion is the process of collecting, importing, and processing data for storage or analysis.
* It involves moving data from various sources to a destination where it can be stored and used.
* The goal of data ingestion is to make data available for further processing, such as querying, analysis, or reporting.

### Key aspects of data ingestion:

* **Data Collection:** Data is collected from various sources, which can include databases, files, streaming sources, APIs, and other systems.
* **Data Transformation:** Data may undergo transformation to convert it into a suitable format for storage or analysis. This can include cleaning, enriching, and aggregating the data.
* **Data Loading:** The transformed data is loaded into a storage or processing system. This can be a data warehouse, data lake, database, or another type of storage system.
* **Data Validation:** Data may be validated to ensure its quality, consistency, and correctness before loading it into the destination system.
* **Data Processing:** In some cases, data may undergo further processing or analysis after it is ingested, such as running algorithms or generating reports.

### Example of data ingestion:

In a retail business, data ingestion may involve collecting sales data from multiple stores, online sales platforms, and inventory systems. This data is then transformed to standardize formats and clean up inconsistencies. Finally, the transformed data is loaded into a data warehouse for analysis.

### Tools for data ingestion:

* There are various tools and technologies available for data ingestion, including Apache Kafka, Apache NiFi, AWS Glue, Google Cloud Dataflow, and others.
* These tools help automate the data ingestion process and provide features for data transformation, validation, and loading.

## ETL

* ETL stands for Extract, Transform, Load. It refers to the process of extracting data from various sources, transforming it into a suitable format, and loading it into a target destination, such as a data warehouse, data lake, or database, for analysis and reporting.

### Key components of ETL:

* **Extract:** Data is extracted from one or more source systems, which can include databases, files, APIs, and other sources.
* **Transform:** Extracted data is transformed into a format that is suitable for analysis. This can include cleaning, aggregating, and enriching the data.
* **Load:** Transformed data is loaded into a target destination, such as a data warehouse, where it can be stored and analyzed.

### ETL Process:

* **Extraction:** Data is extracted from the source systems using ETL tools or scripts. The extracted data can be full or incremental, depending on the requirements.
* **Transformation:** Extracted data is transformed to meet the requirements of the target system. This can involve cleaning up data, converting data types, and aggregating data.
* **Loading:** Transformed data is loaded into the target destination. This can be a batch process or real-time streaming, depending on the needs of the organization.

### Example of ETL:

In a retail business, ETL may involve extracting sales data from point-of-sale systems, transforming it to standardize product codes and clean up customer information, and loading it into a data warehouse for analysis.

### Tools for ETL:

* There are various ETL tools available, both open-source and commercial, that help automate the ETL process.
* Some popular ETL tools include Apache NiFi, Talend, Informatica, and Microsoft SSIS (SQL Server Integration Services).

## ELT

* ELT stands for Extract, Load, Transform. It is a data processing approach similar to ETL, but with the transformation step occurring after the data is loaded into the target destination.
* ELT is often associated with big data and data lake architectures, where the data is initially loaded into the destination in its raw form and then transformed as needed for analysis.

### Key components of ELT:

* **Extract:** Data is extracted from various sources, such as databases, files, APIs, and streaming sources.
* **Load:** Extracted data is loaded into a target destination, such as a data lake, data warehouse, or database, in its raw form.
* **Transform:** Data is transformed within the target destination as needed for analysis. This can include cleaning, aggregating, and enriching the data. ### ELT Process:
* **Extraction:** Data is extracted from source systems using ETL tools or scripts.
* **Loading:** Extracted data is loaded into the target destination in its raw form.
* **Transformation:** Data is transformed within the target destination using tools and technologies that are optimized for big data processing, such as Hadoop, Spark, or data warehouse technologies.

### Example of ELT:

In a big data environment, ELT may involve extracting data from various sources, loading it into a data lake, and then transforming the data using technologies like Spark or Hadoop. This allows for flexible and scalable processing of large volumes of data.

### Tools for ELT:

* Popular ELT tools include Apache Spark, AWS Glue, Talend, Matillion, Informatica, Azure Data Factory, and Pentaho Data Integration.
* These tools vary in capabilities and are used for processing large volumes of data in different environments.

## Star Schema

* The star schema is a data modeling technique used in data warehousing to organize data into a central fact table surrounded by denormalized dimension tables.
* It is called a star schema because the diagram of the schema resembles a star, with the fact table at the center and dimension tables radiating outwards like the arms of a star. ### Key components of a star schema:
* **Fact Table:** The central table in the star schema that contains quantitative data (facts) that can be analyzed. It typically contains foreign keys to the dimension tables.
* **Dimension Tables:** Tables that contain descriptive attributes related to the facts in the fact table. Each dimension table corresponds to a specific aspect or dimension of the business (e.g., time, product, location).
* **Primary and Foreign Keys:** Dimension tables have primary keys that uniquely identify each record, and fact tables have foreign keys that reference the corresponding dimension tables.

### Advantages of a star schema:

* **Simplicity:** The star schema is simple and easy to understand, making it ideal for business users and analysts.
* **Query Performance:** The denormalized structure of the schema enables fast query performance, as it reduces the number of joins required to retrieve data.
* **Flexibility:** The star schema is flexible and can easily accommodate changes in the data model or business requirements.
* **Aggregation:** Aggregations can be precomputed and stored in the fact table, further improving query performance for summary queries.

### Example of a star schema:

* Consider a sales database with a fact table Sales and dimension tables Product, Time, and Location.
* The Sales table contains sales data such as sales amount and quantity, with foreign keys to the Product, Time, and Location tables, which contain details about the products, time periods, and locations respectively.

-- Total Sales Amount by Product Category:  
SELECT   
 Product.Category,  
 SUM(Sales.Amount) AS TotalSalesAmount  
FROM  
 Sales  
 JOIN Product ON Sales.ProductID = Product.ProductID  
GROUP BY  
 Product.Category;  
  
-- Total Sales Amount by Month:  
SELECT   
 Time.Month,  
 SUM(Sales.Amount) AS TotalSalesAmount  
FROM  
 Sales  
 JOIN Time ON Sales.DateID = Time.DateID  
GROUP BY  
 Time.Month;

## Snowflake Schema

* The snowflake schema is a data modeling technique used in data warehousing where a centralized fact table is surrounded by normalized dimension tables.
* Unlike the star schema, the snowflake schema’s dimension tables are normalized, meaning they are further broken down into sub-dimension tables.
* This results in a more complex schema but can lead to more efficient use of storage and better data integrity.

### Key components of a snowflake schema:

* **Fact Table:** Central table containing quantitative data (facts) that can be analyzed. It typically contains foreign keys to the dimension tables.
* **Dimension Tables:** Tables containing descriptive attributes related to the facts in the fact table. Dimension tables are normalized, meaning they may be broken down into sub-dimension tables.
* **Primary and Foreign Keys:** Dimension tables have primary keys that uniquely identify each record, and fact tables have foreign keys that reference the corresponding dimension tables.

### Advantages of a snowflake schema:

* **Normalisation:** Reduces redundancy and improves data integrity by eliminating duplicate data in dimension tables.
* **Storage Efficiency:** Uses storage more efficiently compared to denormalized schemas like star schema, especially when dealing with large dimension tables.
* **Easier Maintenance:** Changes to dimension tables only require updates to a single table, improving maintainability.

#### Example of a snowflake schema in a sales database:

* In a sales database, a snowflake schema may include a fact table Sales and dimension tables Product, Time, and Location, each of which can be further normalized if needed (e.g., Product\_Category, Product\_Subcategory).

SELECT   
 p.Product\_Category,  
 t.Month,  
 SUM(s.Amount) AS TotalSalesAmount  
FROM  
 Sales s  
 JOIN Time t ON s.TimeID = t.TimeID  
 JOIN Product p ON s.ProductID = p.ProductID  
GROUP BY  
 p.Product\_Category,  
 t.Month;

## Facts Dimensions

In data warehousing, the terms “facts” and “dimensions” are fundamental concepts in designing a schema for storing and analyzing data.

### Facts:

* Facts are numerical measures or metrics that represent specific business aspects, such as sales revenue, quantity sold, or profit margin.
* In a data warehouse, facts are typically stored in a fact table, which contains the quantitative data that can be analyzed.
* Each record in a fact table is associated with one or more dimension keys that relate to the dimensions in the data model.

### Dimensions:

* Dimensions are the descriptive attributes or categories by which the facts are analyzed.
* They provide the context for the facts and include entities such as time, product, location, and customer.
* Dimension tables contain the attributes related to each dimension, and each record in a dimension table represents a unique value or member of that dimension.

### Example:

In a sales data warehouse, the fact table might contain measures such as sales amount, quantity sold, and profit margin. The dimension tables would include tables for product (containing product details), time (containing date-related information), and location (containing location details).

### Facts and Dimensions Relationship:

* Facts are typically associated with one or more dimensions through foreign key relationships.
* The combination of dimensions used to analyze a set of facts is often referred to as a “slice” of the data.

### Factless Fact Tables:

* Sometimes, a fact table may not contain any measurable facts but serves as a bridge between dimensions. This is called a “factless fact table” and is used to represent events or relationships between dimensions.

## Data Mart

* Data marts are subsets of a data warehouse that are designed for specific business lines or departments within an organization.
* They are smaller, more focused data repositories that contain a subset of data from the larger data warehouse.
* Data marts are typically created to meet the specific reporting and analysis needs of a particular group or department, providing them with easy access to relevant data without having to navigate the entire data warehouse.

### Key characteristics of data marts:

* **Focused Data:** Data marts contain a subset of data from the data warehouse that is relevant to a specific business area or department.
* **Subject-Oriented:** Data marts are organized around specific subjects or business areas, such as sales, marketing, or finance.
* **Summarized Data:** Data in data marts is often summarized and aggregated to support specific reporting and analysis requirements.
* **User-Friendly:** Data marts are designed to be user-friendly and easily accessible to business users, allowing them to perform ad-hoc queries and generate reports without technical expertise.
* **Independent:** While data marts are typically derived from the data warehouse, they can also be created independently to meet the specific needs of a department or business unit.

### Types of data marts:

* **Dependent Data Mart:** A data mart that is derived directly from the data warehouse, using the same data and structures.
* **Independent Data Mart:** A data mart that is created independently of the data warehouse, using data from various sources specific to the business area.

### Examples

* Here’s an example of how data marts could be structured for a sales database
* **Sales Data Mart:**
  + **Focus:** Analyzing sales performance.
  + **Data:** Contains aggregated sales data by product, region, time period, and other relevant dimensions.
  + **Purpose:** Provides insights into sales trends, product performance, and regional sales analysis.
* **Customer Data Mart:**
  + **Focus:** Analyzing customer behavior and preferences.
  + **Data:** Contains customer demographics, purchase history, and other customer-related information.
  + **Purpose:** Helps in understanding customer segments, lifetime value, and personalized marketing campaigns.
* **Product Data Mart:**
  + **Focus:** Analyzing product performance.
  + **Data:** Contains product details, sales history, inventory levels, and product categories.
  + **Purpose:** Provides insights into best-selling products, inventory management, and product profitability.
* **Regional Sales Data Mart:**
  + **Focus:** Analyzing sales performance by region.
  + **Data:** Contains sales data segmented by geographical regions, such as country, state, or city.
  + **Purpose:** Helps in understanding regional sales trends, market penetration, and regional marketing strategies.

### Example Queries

* Assuming we have the following tables in our data warehouse:
* **Fact Table:** sales
  + sales table columns: sale\_id, product\_id, customer\_id, sale\_date, quantity, amount
* **Dimension Tables:** products, customers, dates
  + products table columns: product\_id, product\_name,category
  + customers table columns: customer\_id, customer\_name, city, state
  + dates table columns: date\_id, date, day\_of\_week, month, year

### **Sales Data Mart Creation:**

* Sales Analysis by Product Category:
  + Create a new table in the data mart for sales analysis by product category.
  + Aggregate sales data by product category for each year.
* CREATE TABLE sales\_by\_product\_category AS  
  SELECT  
   p.category,  
   EXTRACT(year FROM s.sale\_date) AS year,  
   SUM(s.amount) AS total\_sales\_amount  
  FROM  
   sales s  
  JOIN  
   products p ON s.product\_id = p.product\_id  
  GROUP BY  
   p.category, EXTRACT(year FROM s.sale\_date);
* Sales Analysis by Customer City:
  + Create a new table in the data mart for sales analysis by customer city.
  + Aggregate sales data by customer city for each year.
* CREATE TABLE sales\_by\_customer\_city AS  
  SELECT  
   c.city,  
   EXTRACT(year FROM s.sale\_date) AS year,  
   SUM(s.amount) AS total\_sales\_amount  
  FROM  
   sales s  
  JOIN  
   customers c ON s.customer\_id = c.customer\_id  
  GROUP BY  
   c.city, EXTRACT(year FROM s.sale\_date);
* Sales Analysis by Month:
  + Create a new table in the data mart for sales analysis by month.
  + Aggregate sales data by month for each year.
* CREATE TABLE sales\_by\_month AS  
  SELECT  
   EXTRACT(year FROM s.sale\_date) AS year,  
   EXTRACT(month FROM s.sale\_date) AS month,  
   SUM(s.amount) AS total\_sales\_amount  
  FROM  
   sales s  
  GROUP BY  
   EXTRACT(year FROM s.sale\_date), EXTRACT(month FROM s.sale\_date);