# Data Engineering Interview: Basic Concepts in Data Modeling

Ankur Bhattacharya

# What is a Data Warehouse

A data warehouse is a large, centralized, and integrated repository that stores and manages data from various sources within an organization. It is designed to facilitate business intelligence (BI) activities, data analysis, and reporting. The main purpose of a data warehouse is to provide a single, consistent, and reliable source of data for decision-making and analysis by bringing together data from disparate sources into a structured and optimized format.

## Key characteristics of a data warehouse include:

1. **Data Integration**: Data warehouses collect and integrate data from different operational systems, such as transactional databases, spreadsheets, and other sources. This integration process ensures that data is consistent and accurate.

2. **Time-variant Data Storage**: A data warehouse stores historical data over time, allowing users to analyze trends and changes in data over specific periods.

3. **Subject-Oriented**: Data in a data warehouse is organized around key subjects or areas relevant to the organization's business, such as sales, customers, products, etc.

4. **Non-volatile**: Unlike operational databases, which are constantly updated with real-time data, a data warehouse is non-volatile. It means that once data is loaded into the warehouse, it remains unchanged, ensuring consistency for analytical purposes.

5. **Query and Reporting Performance**: Data warehouses are optimized for fast querying and reporting, enabling users to perform complex analytical operations on large datasets efficiently.

The data stored in a data warehouse is typically used for business analysis, trend analysis, data mining, decision support, and generating reports. By providing a centralized and structured view of data, data warehouses help organizations make informed business decisions based on historical and current data insights.

# Difference Between Database/DataLake/DataWarehouse

**DataBase:**

- **Purpose:** A database is designed primarily for transactional processing, supporting day-to-day operations of an organization. It handles real-time data processing and CRUD operations (Create, Read, Update, Delete).
- **Data Structure:** Databases are typically structured, adhering to a fixed schema that defines the data types, relationships, and constraints.
- **Data Use:** Databases are best suited for managing operational data, such as online transactional data in applications like customer orders, inventory, financial transactions, etc.
- **Performance:** They are optimized for handling concurrent transactions with high throughput and low-latency response times.
- **Examples:** MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL, etc.

**Data Warehouse:**

- **Purpose**: A data warehouse is specifically designed for analytical processing and business intelligence. It serves as a centralized repository for historical and aggregated data from various sources.
- **Data Structure**: Data warehouses use a denormalized schema, often in a star or snowflake schema, to optimize query performance and analytical reporting.
- **Data Use**: Data warehouses are used for data analysis, complex queries, and generating reports to support decision-making and trend analysis.
- **Time Variance**: Data warehouses store historical data over time, allowing users to analyze trends and changes.
- **Examples**: Amazon Redshift, Google BigQuery, Snowflake, Microsoft Azure Synapse Analytics, etc.

**Data Lake:**

- **Purpose:** A data lake is a storage system that holds a vast amount of raw, unstructured, and structured data in its native format. It acts as a central repository for diverse data sources, providing a single location for data storage and exploration.
- **Data Structure**: Data lakes are schema-on-read, meaning that the data's structure is only imposed when it is read or queried, allowing for flexibility and the inclusion of various data types.
- **Data Use**: Data lakes are used for exploratory data analysis, data science, and big data processing. They support data transformation and preparation for analysis.
- **Data Ingestion**: Data lakes can ingest both structured data (e.g., databases) and unstructured data (e.g., log files, social media data, etc.).
- **Examples**: Amazon S3, Hadoop Distributed File System (HDFS), Azure Data Lake Storage, Google Cloud Storage, etc.

# Introduction to Data Mart

A data mart is a smaller, specialized subset of a data warehouse that focuses on providing data for a specific group of users or a particular business function within an organization. It serves as a decentralized, departmentalized version of a data warehouse, catering to the needs of individual teams or departments. Data marts are designed to offer a more targeted and efficient approach to data analysis, enabling users to access relevant information swiftly and make informed decisions based on their specific requirements.

## Difference between Data Warehouse and Data Mart:

| Aspect | Data Warehouse | Data Mart |
|---|---|---|
| Purpose | Serves as a centralized repository that consolidates data from various sources across the organization. | Focuses on meeting the specific needs of a particular business unit, department, or user group. |
| Scope | Covers the entire organization's data and integrates data from different sources into a unified view. | Contains a subset of data relevant to a specific user group or department. |
| Data Size | Generally, data warehouses store large volumes of historical and current data over extended periods. | Contains a smaller dataset, limited to the requirements of the targeted users or department. |
| Schema Design | Uses a global, enterprise-wide schema, often denormalized to optimize query performance for complex analysis. | Utilizes a local schema designed to cater to the specific analytical needs of the data mart's users. |

| Aspect | DataWarehouse | Data Mart |
| --- | --- | --- |
| Implementation Time | Building a data warehouse is a time-consuming and resource-intensive process, involving data integration, transformation, and modeling. | Data marts are quicker to implement, as they focus on a narrower scope and leverage the existing data warehouse's data. |
| Flexibility | Offers a broad and flexible range of data for cross-functional analysis and enterprise-wide decision-making. | Provides tailored data structures and content, offering agility and rapid response to specific business requirements. |
| Accessibility | Access may require a higher level of clearance and involvement from IT teams due to its enterprise-wide nature. | Data marts are more accessible to business users, allowing them to control and access their data independently. |
| Business User Control | Business users may have limited control over data definitions and structures, as it caters to the organization as a whole. | Business users have greater control over data definitions and structures, aligning with their specific needs and preferences. |
| Scalability | Designed to handle large-scale data and complex queries across the entire organization. | Scales easily to accommodate the specific needs of a single department or team, often requiring less computational resources. |

# Difference Between ETL & ELT

**ETL (Extract, Transform, Load): (Useful for datawarehouse ingestion)**

ETL follows a traditional approach to data integration. It begins by extracting data from various source systems, such as databases, applications, and external data feeds. The extracted data is then transformed into a structured and consistent format to match the destination data warehouse's schema

**Key characteristics of ETL:**

**Schema on Read:** ETL relies on a fixed schema approach, meaning data is transformed and conformed to the warehouse schema before loading. This ensures that data is ready for immediate analysis once loaded into the data warehouse.

**Use in Traditional Warehouses:** ETL is commonly employed in traditional data warehousing setups, where data is processed and structured before entering the warehouse, ensuring consistency and data integrity.

**ELT (Extract, Load, Transform):**

ELT, on the other hand, takes a modern and more flexible approach to data integration. It begins with data extraction from various sources, much like ETL. However, instead of immediately transforming the data, it is loaded into the data storage layer, often a Data Lake, in its raw and unstructured form. This storage approach is commonly referred to as "Schema on Write."

**Key characteristics of ELT(Useful for dataleke ingestion ):**

**Schema on Write:** ELT allows the data lake to accept data in its original schema, providing greater flexibility to analyze diverse data types without upfront transformation.

**Use in Big Data and Data Lakes:** ELT is well-suited for big data environments and data lakes, where vast amounts of raw and unstructured data can be ingested and processed, allowing for dynamic and agile data analysis.

# Two Types of ETL: Initial and Incremental

Within the realm of Extract, Transform, Load (ETL) processes, there are two primary approaches employed to manage data integration and synchronization: Initial ETL and Incremental ETL. These methodologies cater to specific data management needs and play pivotal roles in ensuring data accuracy and timeliness in various systems.

- **Initial ETL:**

Initial ETL, also known as Full ETL, is the first step in populating a data warehouse or data storage system with data from diverse source systems. In this approach, the entire dataset is extracted from the source systems, without considering previous data loading history. All relevant data is pulled from the sources, transformed into a consistent format, and loaded into the target destination.

**Key characteristics of Initial ETL:**

**Comprehensive Data Load**: During an initial ETL process, all data required for analysis or reporting is extracted from the source systems. This approach ensures that the data warehouse starts with a complete and up-to-date dataset.

**Time-Consuming:** As it extracts and loads all data from scratch, the initial ETL process can be time-consuming, especially when dealing with large volumes of data.

**Overwriting Existing Data:** With Initial ETL, the existing data in the target destination is usually truncated or overwritten, ensuring a fresh start with the latest data.

- **Incremental ETL:**

Incremental ETL, also known as Delta ETL or Change Data Capture (CDC), focuses on extracting and loading only the changes or updates that have occurred in the source data since the last ETL process. Rather than processing the entire dataset, Incremental ETL identifies new or modified data and applies only these changes to the target destination.

**Key characteristics of Incremental ETL:**

**Efficient Data Updates:** Incremental ETL significantly reduces processing time and resources by extracting and processing only the changed data, making it ideal for dealing with large datasets.

**Maintaining Historical Data:** By selectively applying changes, Incremental ETL preserves historical data in the target destination, allowing for trend analysis and historical reporting.

**Continuous Updates**: Incremental ETL processes can be scheduled at regular intervals to ensure the data warehouse remains up-to-date with the latest changes from the source systems.

# Data Warehouse: Three Layer Architecture

A 3-layer architecture in the context of a data warehouse refers to the structure of the data warehousing system, which is designed to help organizations efficiently manage and analyze their data. These three layers are:

**1.Data Source Layer**: This is the first layer of the architecture, where data is collected from various source systems. These sources can include databases, flat files, external data feeds, and more. The data is extracted from these sources and transformed into a format suitable for storage and analysis in the data warehouse. The transformation process can involve data cleansing, integration, and the application of business rules.

Source data can be production data,internal data,Archived data,External Data,Example: Data which comes to your datalake can be source data for your warehouse system

**2.Data Warehouse Layer**: This is the core of the data warehousing architecture. In this layer, data from the data source layer is stored in a structured, organized, and optimized manner. The data is typically stored in a format that supports efficient querying and reporting, such as a star schema or snowflake schema. The data warehouse layer may include one or more data warehouses, depending on the complexity of the organization's data needs.

In the data warehouse layer, data is typically stored in a specialized database management system (DBMS) that comes with its own built-in metadata repository. This layer can be broadly categorized into three key components:

**A.Data Mart**: Data marts are subsets of the data warehouse specifically tailored to meet the data delivery requirements of specific user groups or business units. They serve as focused, smaller databases within the broader data warehouse, providing granular access to data for specific analytical needs.

**B.Data Warehouse DBMS and Metadata**: The data warehouse DBMS is the core storage system where data from various sources is consolidated, transformed, and organized for efficient querying and reporting. This DBMS often includes an integrated metadata repository, which stores essential information about the data, its structure, and its lineage, aiding in data management and governance.

**C.Dimensional Modeled DBMS**: In alignment with specific business requirements, the data warehouse layer may incorporate dimensional modeling within the DBMS. Dimensional modeling is a design technique that structures data in a way that is optimized for analytical queries. It typically involves the creation of star schemas or snowflake schemas, which simplify data retrieval for reporting and analysis.

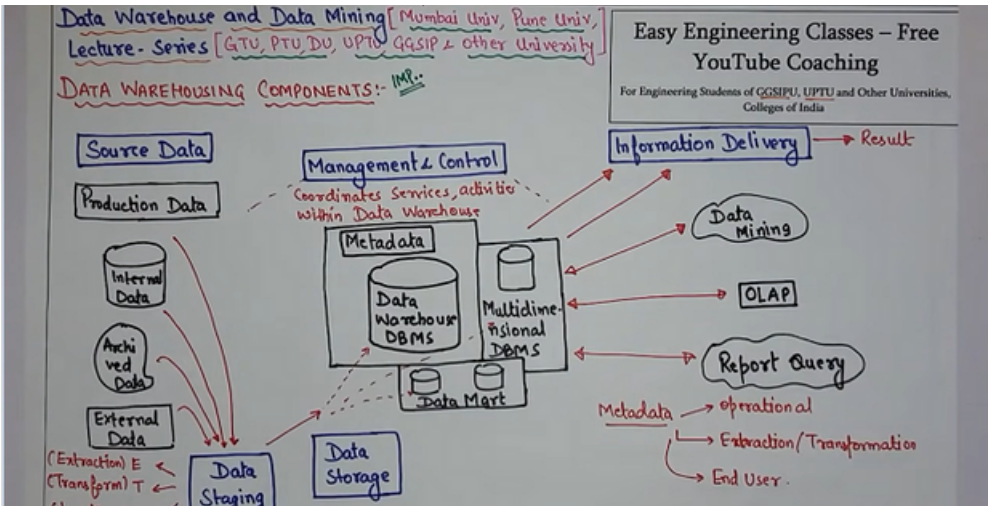



diagram:1

**3. Data Access Layer:**The data access layer is where end-users, typically business analysts, data scientists, and other stakeholders, access the data stored in the data warehouse. This layer includes tools and interfaces for querying, reporting, and analyzing the data. Users can interact with the data using various BI (Business Intelligence) tools, SQL queries, or custom applications. The goal is to make it as easy as possible for users to retrieve and analyze the data for decision-making and business intelligence purposes.

# OLTP vs OLAP

| Aspect | OLAP (Online Analytical Processing) | OLTP (Online Transaction Processing) |
|---|---|---|
| Purpose | Designed for complex data analysis and reporting | Geared towards real-time transaction processing and data updates. |
| Data Type | Handles historical and aggregated data. Typically used for read-heavy operations. | Manages current, operational data. Primarily supports write-heavy operations. |
| Data Volume | Manages large volumes of data, typically for reporting purposes | Handles smaller subsets of data for day-to-day transactions. |
| Query Complexity | Supports complex ad-hoc queries involving aggregation and data slicing/dicing. | Primarily handles simple, predefined queries, and updates. |
| Performance | Read Optimized for query performance, with pre-aggregated data for faster reporting. | write Optimized for fast transaction processing and data integrity. |
| Users | Primarily used by business analysts, data scientists, and decision-makers. | Accessed by operational staff, including clerks, customer service, and administrators. |
| Examples | Business intelligence systems, data warehouses, and reporting tools. | Order processing systems, inventory management, and customer relationship management (CRM) systems. |

# Dimensional Modeling

Dimensional modeling is a design approach in data warehousing that structures data for easy and efficient analytical querying. It uses dimensions (attributes) and facts (quantitative data) organized in a way that simplifies data analysis, often using star or snowflake schemas for better query performance.

# What is Fact Table

A **fact table** is a critical component of a data warehouse that stores quantitative data related to business events, allowing analysts to perform in-depth analysis and gain valuable insights into the performance and trends of the business processes being measured.

For example, in a sales data scenario, a fact table might contain measures like sales amount, quantity sold, and profit. The foreign keys in the fact table would link to dimension tables such as date, product, and store, providing additional information about the sales events, such as the date of the sale, the product sold, and the store where the sale occurred.

**FACT HOLD TWO KIND OF DATA**
foreign key of dimension and Measures

**Grain Of Fact Table**: Whats the level of uniquely identify the fact table

# Types Of Fact Table

**A. Transactional Fact Table:**
This type of fact table captures individual business events or transactions at a detailed level. It stores data for each occurrence of an event, such as a sales transaction, a customer order, or a service request. Transactional fact tables are used to track specific events and are suitable for granular analysis.

**Example**:

**Sales_Fact Table**:

Here we are capturing **Quantity_Sold** ,**Sales_Amount** ,**Profit** as part of our Transactional Fact Table

| Column Name | Data Type | Description |
|---|---|---|
| Sales_ID | INTEGER | Primary key, unique identifier for each sales transaction. |
| Date_ID (FK) | INTEGER | Foreign key referencing the Date_Dimension table. |
| Product_ID (FK) | INTEGER | Foreign key referencing the Product_Dimension table. |
| Customer_ID (FK) | INTEGER | Foreign key referencing the Customer_Dimension table. |
| **Quantity_Sold** | INTEGER | The quantity of products sold in the transaction. |
| **Sales_Amount** | DECIMAL | The total sales amount for the transaction. |
| **Profit** | DECIMAL | The profit generated from the sales transaction. |

**B. Periodic Snapshot Fact Table:**

A periodic snapshot fact table stores aggregated data at specific intervals or snapshots, usually at regular time periods (e.g., daily, weekly, monthly). It represents the state of metrics or KPIs at a specific point in time, providing a snapshot of business performance over time. This type of fact table is useful for trend analysis and tracking changes in business metrics over different periods.

**Example:**

**Periodic Snapshot Fact Table (Monthly_Sales_Fact):**

Here we are capturing Monthly_Sales as part f our Periodic Snapshot Fact Table

| Column Name | Data Type | Description |
|---|---|---|
| Snapshot_Date | DATE | The date representing the end of each calendar month. |
| Store_ID (FK) | INTEGER | Foreign key referencing the Store_Dimension table. |
| Product_ID (FK) | INTEGER | Foreign key referencing the Product_Dimension table. |
| **Monthly_Sales** | DECIMAL | The total sales amount for the store and product in the month. |

**C. Accumulating Snapshot Fact Table:**

An accumulating snapshot fact table tracks the progress or status of a process over time, recording **key milestones** or events as they occur. It stores data related to specific process stages, and as the process progresses, new information is added to the fact table to reflect the current state. Accumulating snapshot fact tables are often used in process-oriented analysis, such as order fulfillment or project management.

| Order_ID | Customer_ID | Product_ID | Order_Date | Initial_Review_Date | Final_Review_Date | Approved_Date | Shipped_Date | Delivered_Date | Status |
|---|---|---|---|---|---|---|---|---|---|
| 1001 | 201 | 301 | 2023-07-01 | 2023-07-02 | 2023-07-03 | 2023-07-05 | 2023-07-08 | 2023-07-10 | Delivered |
| 1002 | 202 | 302 | 2023-07-02 | 2023-07-03 | 2023-07-04 | 2023-07-06 | 2023-07-09 | | Shipped |
| 1003 | 203 | 303 | 2023-07-03 | 2023-07-04 | | | | | In Progress |
| 1004 | 204 | 304 | 2023-07-04 | | | | | | Pending |

In above example , this accumulating snapshot fact is capturing different moment of order fulfilling history in proper way and keep updating each record accordingly

- **Order_ID** uniquely identifies each order.
- **Customer_ID** and **Product_ID** are foreign keys referencing the **Customer_Dimension** and **Product_Dimension** tables, respectively, providing additional information about customers and products.
- **Order_Date** represents the date when the order was placed.
- **Initial_Review_Date**, **Final_Review_Date**, **Approved_Date**, **Shipped_Date**, and Delivered_Date represent the dates when specific milestones in the order fulfillment process occurred. If a milestone has not yet occurred, the corresponding cell is left empty.
- **Status** indicates the current status of each order in the fulfillment process.

.

Here's a breakdown of the status for each order:

1. Order with Order_ID 1001: This order has gone through all stages and is marked as "**Delivered**."
2. Order with Order_ID 1002: This order has gone through the initial and final reviews, approved for fulfillment, and has been "**Shipped**" to the customer.
3. Order with Order_ID 1003: This order is still "**In Progress**," as it has gone through the initial review, but the final review and approval are pending.
4. Order with Order_ID 1004: This order is still "**Pending**," as it has been placed, but the initial review is pending

**D. Factless Fact Table:**

A factless fact table contains foreign keys referencing various dimension tables but lacks any numeric measures. It captures events or occurrences without any quantitative data associated with them. Factless fact tables are useful for representing many-to-many relationships between dimensions or recording events where no measures are relevant.

| Column Name | Data Type | Description |
|---|---|---|
| Student_ID (FK | INTEGER | Foreign key referencing the Student_Dimension table |
| Course_ID (FK) | INTEGER | Foreign key referencing the Course_Dimension table. |
| Enrollment_Date | DATE | The date when the student enrolled in the course |

In this example, the Enrollment_Fact table is a factless fact table that captures enrollment events for students in courses. It stores the foreign keys referencing the Student_Dimension and Course_Dimension tables, indicating which student has enrolled in which course and when the enrollment occurred.

# What is Dimension Table

In the context of data warehousing and database management, a dimension table is a type of table that stores descriptive or textual information about business entities, often referred to as "dimensions." Dimension tables are a fundamental component of the star schema and snowflake schema data modeling techniques used in data warehousing. They help organize and provide context to the measures or facts stored in fact tables.

**Key characteristics of dimension tables include:**

****Descriptive Attributes:**** Dimension tables typically contain descriptive attributes that provide additional information about business entities. For example, in a sales data warehouse, dimension tables might include information about customers, products, time, geography, or sales representatives.

****Primary Key:**** Each dimension table has a primary key that uniquely identifies each record or row within the table. The primary key is used to establish relationships with fact tables.

****No Numeric Data:**** Unlike fact tables, dimension tables do not contain numeric measures or metrics. Instead, they store textual or categorical data, such as names, descriptions, codes, or labels.

****Hierarchical Structure:**** Dimension tables may have a hierarchical structure, allowing them to represent multiple levels of granularity. For example, a time dimension table might include levels for years, quarters, months, and days.

****Low Cardinality:**** Dimension tables often have a relatively low cardinality, meaning they contain a limited number of distinct values compared to fact tables. This characteristic helps improve query performance

# Types Of Dimension Table

There are several types of dimension tables, each serving a specific purpose in data warehousing:

1. **Conformed Dimension**:** A conformed dimension is a dimension in a data warehousing or business intelligence environment that has consistent and uniform attributes and definitions across all data marts and data sources within an organization. This consistency ensures that the same dimension can be used seamlessly in different parts of a data warehouse, data mart, or reporting system. Here's an example of a conformed dimension:

**Date Dimension:**
A date dimension is a common example of a conformed dimension. In most organizations, dates are used in various reports and analyses across different departments and systems. The date dimension contains attributes such as:

1. Date
2. Month
3. Quarter
4. Year
5. Week
6. Day of the week
7. Public Holidays
8. Fiscal period, etc.

In a data warehouse or data mart, different parts of the organization can utilize this date dimension without inconsistencies. For instance:

1. **Finance Department**: In the finance data mart, the date dimension is used to analyze financial data by various time periods (e.g., monthly financial statements).
2. **Sales Department**: In the sales data mart, the date dimension helps track sales performance over time (e.g., daily, weekly, monthly, and yearly sales figures).
3. **Human Resources Department**: In the HR data mart, the date dimension can be used for tracking employee records by hiring date, anniversary dates, or other HR-related timeframes.
4. **Marketing Department**: In the marketing data mart, the date dimension assists in analyzing campaign performance and customer behavior over time.

The key to a conformed dimension like the date dimension is that it's maintained consistently throughout the organization. Any updates or changes to the dimension are made in a centralized location, ensuring that everyone uses the same date attributes and definitions. This uniformity is vital for producing accurate and consistent reporting and analysis across the organization.

**2. Junk Dimension:** A junk dimension is a dimension table that consolidates multiple low-cardinality flags or attributes into a single table. This helps reduce the complexity of the schema and improve query performance

Suppose you are designing a data warehouse for a retail business, and you have several low-cardinality attributes that are not directly related to any specific dimension but are still useful for analysis and reporting. These attributes might include:

1. **Promotion Type**: Describes the type of promotion used (e.g., discount, free gift, buy-one-get-one).
2. **Payment Method**: Indicates how customers paid for their purchases (e.g., cash, credit card, debit card).
3. **Purchase Channel**: Specifies where the purchase was made (e.g., in-store, online, mobile app).
4. **Coupon Used**: Indicates whether a coupon was applied to the purchase (yes or no).

Instead of creating separate dimension tables for each of these attributes, you can create a junk dimension called "Junk_Dimension" or "Miscellaneous_Dimension." This dimension table might look like this:
Junk Dimension - Miscellaneous_Dimension:

- JunkKey: A unique identifier for each combination of attributes.
- Promotion Type: Attribute indicating the promotion type.
- Payment Method: Attribute indicating the payment method.
- Purchase Channel: Attribute specifying the purchase channel.
- Coupon Used: Attribute indicating whether a coupon was used.

The "JunkKey" serves as a unique key for each combination of attribute values. This allows you to reduce the number of dimension tables and keep these low-cardinality attributes in one place, making it easier to manage and query the data.

**3.Degenerate Dimension :** A degenerate dimension is a dimension that is derived from fact table data and does not have a separate dimension table. It is often used for grouping or aggregating facts.

Suppose you are designing a data warehouse for a retail business, and you have a fact table that records sales transactions. Within this fact table, you may find a transaction ID that is unique for each sale. This transaction ID can serve as a degenerate dimension.
Sales Fact Table:

- **Transaction ID (Degenerate Dimension)**: A unique identifier for each sales transaction.
- **Product Key**: Foreign key linking to the Product dimension table.
- **Date Key**: Foreign key linking to the Date dimension table.
- **Sales Amount**: The amount of the sale.
- **Quantity Sold**: The number of items sold.

In this example, the "Transaction ID" is a degenerate dimension because it's an attribute associated with each sale, but it doesn't have a corresponding dimension table. Instead, it's derived directly from the fact table.

**4.Role-Playing Dimension:**In some cases, a single dimension table can play multiple roles in a data model. For example, a date dimension can be used to represent both the order date and the ship date, each serving as a different role in the fact table.

Suppose you're working on a data warehouse for a retail company, and you have a sales fact table that captures daily sales. You want to analyze sales from different date-related perspectives.
Sales Fact Table:

- DateKey (Role-Playing Date Dimension): A foreign key linked to the "Date" dimension table.
- ProductKey: A foreign key linked to the "Product" dimension table.
- StoreKey: A foreign key linked to the "Store" dimension table.
- SalesAmount: The amount of sales for each transaction.

In this scenario, the "Date" dimension table is used in multiple roles:

1. Order Date Role: The "DateKey" represents the order date when a customer made a purchase.
2. Ship Date Role: The same "DateKey" can also represent the date when the order was shipped.
3. Delivery Date Role: It can be used to represent the date of actual delivery.

Each role-playing dimension has a distinct meaning and serves a specific analytical purpose, allowing you to answer questions like:

- "What were the sales on each product for their order date?"
- "How many products were delivered on time, based on their delivery date?"
- "How many products were shipped and delivered on the same day, based on their ship date and delivery date?"

In this example, the "Date" dimension table plays multiple roles within the same fact table, offering different date-related perspectives for analysis and reporting without the need for creating separate dimension tables for each date role.

5. **Slowly Changing Dimension**(SCD) :

SCDs are dimension tables that track changes to descriptive attributes over time. There are several types of SCDs, including Type 1 (overwrite existing data), Type 2 (add new records with historical data), and Type 3 (maintain limited history) , SCD 4 and SCD 6.

Lets understand each of them one by one.

# Types of SCD

1. **SCD Type 1 - Overwrite**:

In a Type 1 SCD, when a change occurs, the existing record is simply updated with the new data, overwriting the old data. Historical data is not preserved. This approach is suitable when historical changes are not significant or when tracking historical changes is not required.

**Pros**: Simple to implement and results in a compact dataset.

**Cons**: Loss of historical data, no tracking of changes.

2. **SCD Type 2 - Add New Row**:

In a Type 2 SCD, new rows are added to the dimension table to represent each change. Each row has its own unique identifier (e.g., surrogate key) and a valid time range during which it is applicable. This approach retains a full history of changes.

**Pros**: Full historical tracking, no data loss.

**Cons**: Increased storage requirements, more complex queries to handle history

3. **SCD Type 3 - Add New Columns:**

In a Type 3 SCD, additional columns are added to the dimension table to store limited historical changes. Typically, only a small subset of important attributes are tracked as historical columns. This approach is useful when a limited amount of historical data is sufficient.

**Pros**: Simpler structure than Type 2, less storage required.

**Cons**: Limited historical data tracking, not suitable for all scenarios.

4. **SCD Type 4 - Historical Table:**

In a Type 4 SCD, a separate historical table is created to store historical data. The main dimension table holds only the current data, while the historical table stores the previous versions of the records. This approach separates the current and historical data, maintaining data integrity.

**Pros**: Clear separation of current and historical data, no changes to the main table structure.

**Cons**: Requires additional table and maintenance.

**SCD Type 6 - Hybrid Approach:**

A Type 6 SCD combines aspects of Type 1, Type 2, and Type 3 SCDs. It maintains the current record in the main dimension table (Type 1), but also includes additional columns to track changes over time (Type 3). Additionally, it creates new rows to represent changes (Type 2). This approach is versatile but can be complex to manage.

**Pros**: Balances between history and current data, flexible for different scenarios.

**Cons**: Complexity in implementation and querying.

The choice of which SCD type to use depends on the specific requirements of your data warehousing project, including the importance of historical data, storage considerations, and query performance. Each SCD type has its advantages and trade-offs, and the decision should be based on your organization's data management needs

.

# Design of Database Schema

Schema design in the context of databases is a critical aspect of data organization and management, playing a pivotal role in the efficient storage, retrieval, and analysis of information. Two common approaches to schema design are the "**Snowflake**" and "**Star**" schemas, each with its own characteristics and use cases.

**The Need for Schema Design**: Schema design is crucial because it defines the structure of a database, impacting how data is stored, accessed, and analyzed. A well-designed schema ensures data accuracy, consistency, and efficiency. It enables businesses to make informed decisions, generate reports, and gain insights from their data. Whether using a Snowflake or Star schema, the choice depends on the specific requirements of the application and the balance between data integrity and query performance.

**The Snowflake Schema**: The Snowflake schema is a relational database design that arranges data in a structured and normalized manner. In this design, data is organized into multiple related tables, and relationships between these tables are established through foreign keys. The Snowflake schema often includes dimension tables, which store descriptive attributes, and fact tables, which contain numerical measures or metrics. One distinctive feature of the Snowflake schema is that dimension tables can further be broken down into sub-dimensions, leading to a tree-like structure. This normalization helps in reducing data redundancy and maintaining data integrity, making it suitable for scenarios where data accuracy and consistency are paramount.

**KeyPoint: SnowFlake**

- Data is structured and normalized.
- Multiple related tables are used.
- Relationships between tables are established through foreign keys.
- Dimension tables store descriptive attributes.
- Fact tables contain numerical measures or metrics.
- Dimension tables can be further broken down into sub-dimensions.
- This leads to a tree-like structure.
- Normalization reduces data redundancy and maintains data integrity.
- Suitable for scenarios where data accuracy and consistency are paramount.
- Fact tables remain the same as in the source data.
- Join complexity is increased due to the multiple levels of dimension tables

**The star Schema:** In contrast, the Star schema simplifies database design by denormalizing data, creating a centralized fact table at the center, surrounded by dimension tables. This design minimizes the number of joins required to retrieve data, which can lead to faster query performance. Star schemas are particularly well-suited for data warehousing and business intelligence applications where query speed is a priority. The fact table typically contains foreign keys referencing dimension tables, allowing for easy aggregation and analysis of data.

**KeyPoint: Star Schema**:

- Denormalizes data for a simplified database design.
- Features a centralized fact table at the center of the schema.
- Surrounded by dimension tables that provide descriptive attributes.
- Minimizes the number of joins required to retrieve data.
- Results in faster query performance due to reduced complexity.
- Well-suited for data warehousing and business intelligence applications.
- Prioritizes query speed, making it ideal for analytical and reporting purposes.
- The fact table contains foreign keys referencing dimension tables for data aggregation and analysis.

# Example: Snowflake Schema

In this example, a Snowflake Schema is achieved by structuring the database with a central fact table (Transaction_Fact_Table) surrounded by dimension tables that are linked through foreign key relationships. Here's how each table contributes to the Snowflake Schema:

**Dimension Tables:**

- **product_vendor_dim:**
  - Stores information about products and vendors.
  - Linked with the fact table through the "Product_id"  foreign key.

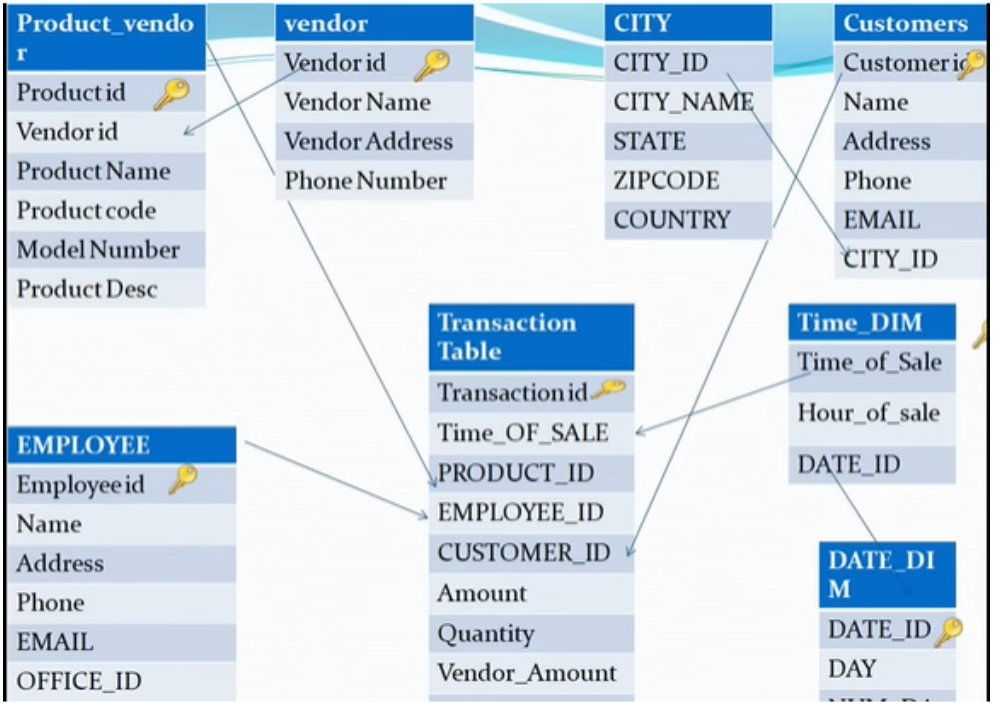
- **vendor_dim:**
  1. Contains vendor details.
  2. Connected to the product_vendor_dim table via the "Vendor_id" foreign key.

- **City_dim**:
  - Holds data about cities, including their names, states, zip codes, and countries.
  - Linked to the customer_dim table through the "CityID" foreign key.
- **Customer_dim**:
  - Stores customer information.
  - Connected to the City_dim table using the "CityID" foreign key.
- **Time_dim**:
  - Focuses on time-related data, such as the time of sale and the hour of sale.
  - Joined with the fact table via the "Time_of_sale" and linked to the date_dim table through the "Date_id" foreign key.
- **Date_Dim**:
  - Contains date-related information, including days, months, and years.
  - Connected to the Time_dim table through the "Date_id" foreign key.
  -

**Fact Table:**

- **Transaction_Fact_Table**:
  - Serves as the central fact table where transactional data is stored.
  - Contains transaction-related details such as Transaction ID, Time of Sale, Product ID, Employee ID, Customer ID, Amount, and Quantity.
  - Links to various dimension tables using their respective foreign keys, creating relationships with the product, vendor, city, customer, time, and date dimensions.

# ER Diagaram Of above example



# Example: Star Schema

In the below example, a star schema is implemented with a central fact table and multiple dimension tables. Here's how each component contributes to the schema:

**Dimension Tables**:**
- `**product_vendor**`: Stores details about products and vendors. It's linked to the fact table via the `product_id` foreign key.
- `**Employee**`: Contains employee information. Connected to the fact table through the `employee_id` foreign key.
- `**customers**`: Holds customer data, including personal and contact details. Linked to the fact table by the `customer_id` foreign key.
- `**Time_dim**`: Focuses on time-related data. It's joined with the fact table through the `time_id` foreign key.

**Fact Table**:**
- `**transaction_table**`: Serves as the central fact table, storing transactional data such as Transaction ID, Time ID, Product ID, Vendor ID, Employee ID, Customer ID, Amount, and Quantity. This table links to various dimension tables using their respective foreign keys, establishing relationships with product, vendor, employee, customer, and time dimensions.

Dimension tables in a star schema are denormalized, meaning they may contain redundant data to improve query performance. For instance, product_vendor contains both VendorId and Vendorname, which could be in a separate table in a normalized schema. Denormalization reduces the number of joins needed during queries, thus enhancing performance

# ER Diagaram Of above example