Introduction to Data Warehouse

Second release

30th Jan, 2025

Contents

Data Warehouse

- A **Data Warehouse** is a centralized repository that stores data from various sources in a single location.
- It is designed to support business intelligence (BI) activities, such as data analysis, reporting, and visualization.
- A **Data Warehouse** typically contains a large amount of historical data, which is used to analyze trends, patterns, and relationships.
- According to **Inmon&Codd** Operational applications (OLTP) and Decision support applications (OLAP) cannot coexist efficiently in the same database

Characteristics of Data Warehouse

- Centralized repository
- Stores data from multiple sources
- Designed for business intelligence and analytics
- Contains historical data
- Supports complex queries and analysis

Data Mart

- A **Data Mart** is a subset of a **Data Warehouse** that contains a specific set of data for a particular business area or department.

- It is designed to support fast query performance and data analysis for a specific business need.

- A **Data Mart** typically contains a smaller amount of data than a **Data Warehouse** and is optimized for a specific business function.

Characteristics of Data Mart

- Subset of a Data Warehouse
- Contains a specific set of data for a business area or department
- Designed for fast query performance and data analysis
- Optimized for a specific business function
- Typically contains a smaller amount of data than a Data Warehouse

Operational Data Store (ODS)

- An **Operational Data Store** (ODS) is a database that stores current and near-real-time data from various operational systems.
- It is designed to support operational reporting and analysis, rather than strategic decision-making.
- An ODS typically contains a small amount of historical data and is optimized for fast data ingestion and query performance.

Characteristics of ODS

- Stores current and near-real-time data
- Designed for operational reporting and analysis
- Contains a small amount of historical data
- Optimized for fast data ingestion and query performance
- Supports real-time or near-real-time data integration

Key Differences

- A **Data Warehouse** is a centralized repository that stores data from multiple sources, while a **Data Mart** is a subset of a **Data Warehouse** that contains a specific set of data for a business area or department.

- An **ODS**, on the other hand, stores current and near-real-time data from operational systems.

- A **Data Warehouse** is designed for business intelligence and analytics, while a **Data Mart** is designed for fast query performance and data analysis for a specific business function.

- An **ODS** is designed for operational reporting and analysis.

Approaches of Data Warehouse

- Inmon Approach
- Kimball Approach

Inmon Approach

- The Inmon approach is also known as the "Top-Down" approach.
- It emphasizes a centralized, enterprise-wide data warehouse that integrates data from various sources.

The key characteristics of the Inmon approach

- **Enterprise-wide data warehouse:** A single, centralized repository that stores data from all parts of the organization.
- **Normalized data:** Data is stored in a normalized form to minimize data redundancy and improve data integrity.
- **Data is transformed:** Data is transformed and cleansed before loading into the data warehouse.
- **Focus on data integration:** The primary focus is on integrating data from various sources to provide a unified view of the organization.

Kimball Approach

- The Kimball approach is also known as the "Bottom-Up" approach.
- It focuses on building a series of smaller, independent data marts that are optimized for specific business areas or departments.

The key characteristics of the Kimball approach

- **Data marts:** A collection of smaller, independent data repositories that are optimized for specific business areas or departments.
- **Denormalized data:** Data is stored in a denormalized form to improve query performance and simplify data access.
- **Data is not transformed:** Data is loaded into the data mart in its original form, with minimal transformation.
- **Focus on business needs:** The primary focus is on meeting the specific business needs of each department or business area.

Key Differences

- **Inmon** focuses on an enterprise-wide data warehouse, while **Kimball** focuses on smaller, independent data marts.

- **Inmon** uses a normalized data structure, while **Kimball** uses a denormalized data structure.

- **Inmon** transforms data before loading, while **Kimball** loads data in its original form.

- **Inmon** focuses on data integration, while **Kimball** focuses on meeting specific business needs.

Dimensions & Facts

- **Fact Tables:**
  - Record a single measurement of a real-world observation.
  - It is almost always numerical.
  - Typically involve money or financial transactions.
  - Atomic facts record a single value.
  - Snapshot or aggregate facts records summary information.
- **Dimension Tables:**
  - Store the descriptive information about the fact.
  - Used for filtering, grouping, and sorting.
  - Time and location are common dimensions.

Types of Data Warehouse Schemas

- Star Schema
- Snowflake Schema
- Galaxy Schema / Fact Constellation

Star Schema

- A **star schema** is a type of **data warehouse** modeling that consists of a central fact table surrounded by dimension tables.

- The **fact table** contains measures or facts, while the **dimension tables** contain descriptive attributes.

- **Star schemas** are simple, easy to maintain, and support fast query performance.

- can contain redundant data, especially for attributes that are common across multiple dimensions.

Components of a Star Schema

- **Fact Table:**
  - Central Table: Contains measurements of a specific business process.
  - Foreign Keys: References to dimension tables to establish relationships.
  - Example: A sales fact table might include columns for {Order ID, Product ID, Customer ID, Sales Amount, and Sales Date}.

- **Dimension Tables:**
  - Descriptive Table: Provide characteristics that describe the data in the fact table.
  - Hierarchical Structure: Often have a hierarchal structure, such as Customer dimension with levels like Customer ID, Country, Region, and City.

- Example: A customer dimension might include columns for {Customer ID, Customer Name, Phone Number, and Customer Type}.

- Attributes of **Date Dimension Table:**
  Date Key (Primary Key),
  Date,
  Day of Week,
  Month Name,
  Month,
  Quarter,
  Year,
  Holiday Flag,
  Weekend Flag

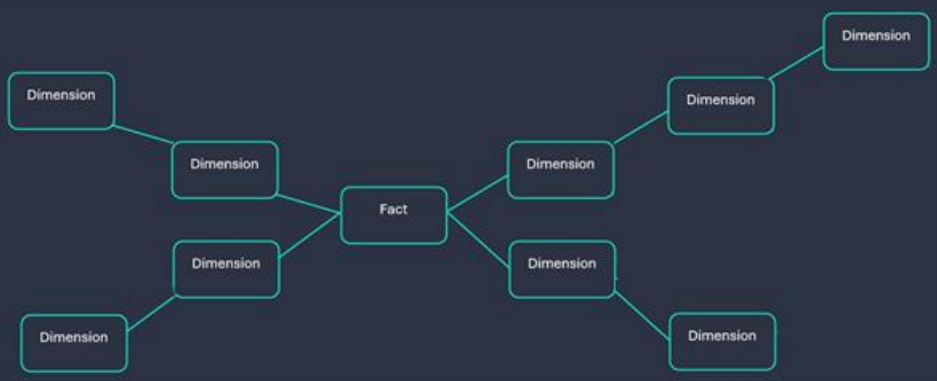
Snowflake Schema

- A **snowflake** schema is a variation of the **star schema**, where each **dimension table** is further divided into multiple related tables.

- **Snowflake Schema** normalizes **dimension tables** to reduce data redundancy and improve data integrity.

- **Snowflake schema** can be more efficient for certain types of queries, but can also introduce additional complexity that might impact performance.

- **Snowflake Schema** ensures data consistency and avoids redundancy.



Components of a Snowflake Schema

- **Fact Table:**
  Remains the same as in a star schema, containing measurements or metrics related to a specific business process.
- **Dimension Tables:**
  - Hierarchical Structure: Dimension tables can be further divided into sub-dimension tables, creating a hierarchical structure.
  - Example: A customer dimension might have sub -dimensions like address, and demographic information

  - Attributes of **Date Dimension Tables:**
    Date Table:
    Date Key (Primary Key),
    Date,
    Day of Week,
    Month Key (Foreign Key),

Year

Month Table:

Month Key (Primary Key),
Short Month Name,
Long Month Name,
Quarter

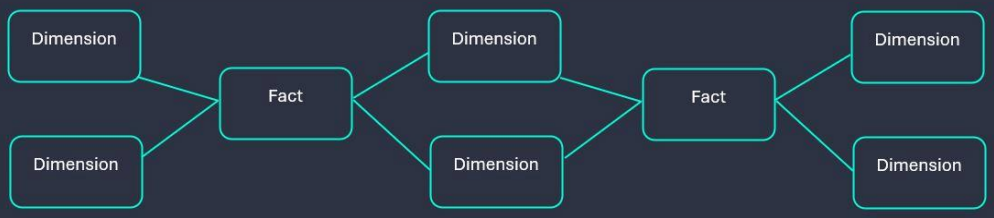
Galaxy Schema

- A **galaxy schema** is a type of **data warehouse** modeling that consists of multiple **fact tables** and **dimension tables.**

- **Galaxy schemas** are used to model complex business processes and support advanced analytics.

- **Galaxy schemas** can model extremely complex relationships and hierarchies within **dimension tables**.

- **Galaxy schema** offers a high level of granularity compared to the **snowflake schema.**



Components of a Galaxy Schema

- **Fact Table:**
  Remains the same as in a star and snowflake schemas, containing measurements or metrics related to a specific business process.
- **Dimension Tables:**
  - Multiple Level of Granularity:
    Dimension tables can be divided into sub-dimension and sub-sub dimensions, creating a multi-level hierarchy.
  - Example: A customer dimension might have sub -dimensions like address, and demographic information and each sub-dimension could be further divided into more granular components.

Dimension Tables in a Data Warehouse

- In a **data warehouse**, **dimension tables** are used to describe the data in the **fact tables**.

- There are several types of **dimension tables:**
    1. Conformed Dimension
    2. Non-Conformed Dimension
    3. Degenerate Dimension
    4. Role-Playing Dimension
    5. Junk Dimension
    6. Factless Fact Table
    7. Bridge Table
    8. Slowly Changing Dimension (SCD)

Conformed Dimensions

- A **Conformod dimension** is a dimension table that is used across multiple fact tables, ensuring consistency and data integrity.
- The same dimension attributes are used in all associated fact tables.
- The values in the dimension table are consistent across different fact tables.
- The level of detail in the dimension table is appropriate for all fact tables.
- Example: Consider a data warehouse with two fat tables: Sales and Returns.
- **Conformed dimension: Customer**

  Attributes: {Customer ID, Customer Name, Address, City, Country} Ensures consistent customer information across both fact tables.

Non-Conformed Dimensions

- A **Non-conformed dimension** is a dimension table that has different attributes or granularity when used in different fact tables.
- It provides a unique view of the data for a specific business process.
- The values in the dimension table may not be consistent across all fact tables.
- The level of detail in the dimension table may vary between fact tables.
- Example: Consider Sales Fact Table might include Product Price, while Returns might include Return Reason.
  Sales might have a product-level granularity, while Returns might have a product- variant level granularity.

Degenerate Dimensions

- A **Degenerate dimension** is a dimension table that is entirely composed of keys from other dimension tables.
- It doesn't have its own unique attributes or data, but rather serves as a bridge between other dimension tables and the fact table.
- Its primary key is typically composed of foreign keys from other dimension tables.
- Example: Consider a Sales fact table with columns like {Order ID, Product ID, Customer ID, and Sales Amount}.
  If the Product Category is not a significant dimension for analysis, it can be represented as a degenerate dimension.
  Degenerate dimension: Product Category
  Attributes: {Product ID (Primary Key), Category ID (Foreign key to a Category dimension)}

Role-Playing Dimensions

- A **Role-playing dimension** is dimension table that plays multiple roles in a data warehouse
- Reduces the number of dimension tables required.
- It can provide different perspectives on the same data.
- The level of detail in the dimension can be adjusted to suit different analytical needs.
- Example: Date Dimension.
  A Date Dimension can be used in multiple roles:
  - Date: For analyzing data by date.
  - Month: For analyzing data by month.
  - Quarter: For analyzing data by quarter.
  - Year: For analyzing data by year.

Junk Dimensions

- **Junk dimensions** Are used to store low-cardinality, categorical data that is not suitable for inclusion in other dimensions.
- These dimensions often contain attributes that are frequently filtered or used in reporting, but do not have a significant impact on the overall data model.
- They contain categorical data, such as flags, indicators, or codes.
- Can simplify the data model by consolidating multiple attributes into a single dimension.
- Example: A Customer dimension might include attributes like {Customer ID, Customer Name, Address, and City}. If there are a few common customer types, such as “Retail”, “Wholesale”, and “Government”, they can stored in a junk dimension called **Customer Type**.

Factless Fact Tables

- A **factless fact table** is a dimension table that contains no measures or facts.
- These tables are often used in scenarios where the focus is on tracking the frequency or existence of events rather than quantifying their impact.
- Often used for frequency analysis or to identify patterns in events.
- Example: A Website Visit fact table might be a factless fact table. it could contain columns like { Visit ID, Customer ID, Visit Date, Page Views, and Time Spent}. While these columns provide valuable information about website visits, they do not directly measure a numerical quantity like sales or revenue.

Bridge Table

- **Bridge tables** in data warehouse serve a similar purpose to their counterparts in transactional databases: they facilitate many-to-many relationships between dimensions. However, their design and usage often differ due to the analytical nature of data warehouse.
- Bridge tables typically connect two or more dimensions.
- Example: In a sales data warehouse, a bridge table might connect the “Product” dimension, the “Customer” dimension, and the “Order” dimension to represent the products purchased by customers in different orders.

Slowly Changing Dimension (SCD)

- A **slowly changing dimension** (SCD) is a technique used in data warehousing to handle dimensions that change over time.
- These dimensions, such as customer, product, or employee, often requires tracking historical data to analyze trends, identify changes, and support decision-making. **Following are the types of SCD**
- Type 0 SCD: No Changes
- Type 1 SCD: Overwrite
- Type 2 SCD: Add New Row
- Type 3 SCD: Add New Column
- Type 4 SCD: Add New Table
- Type 5 SCD: Hybrid (Type 1 + Type 4) Mini-Dimension with Current Overwrite

Type 0 SCD: No Changes

- **Description**: The dimension attributes do not change over time. They remain static and are not updated.
- **Use Case**: Used for attributes that should never change, such as a birthdate or historical records that must remain unchanged.
- Example: A company has a dimension table for Employees with a column {Birthdate}.
- **Scenario**: An employee's birthdate is recorded as 1980-01-01.
  Even if the source system mistakenly updates it to 1985-01-01, the data warehouse retains the original value.
- Table:

EmployeeID	Name	Birthdate
1	John	1980-01-01

Type 1 SCD: Overwrite

- **Description:** The existing data is overwritten with new data. No history is preserved.
- **Use Case:** Suitable for correcting errors or when historical data is not important.
- Example: A company has a dimension table for Products with a column {ProductDescription}.
- **Scenario:** The product description for ProductID 101 changes from "Old Description" to "New Description".
- Before Change:

ProductID	ProductDescription
101	Old Description

- After Change:

ProductID	ProductDescription
101	New Description

Type 2 SCD: Add New Row

- **Description:** A new row is added to the dimension table to reflect the change, preserving the old data. This allows for full history tracking.
- **Use Case:** Ideal for tracking changes where history is important, such as customer address changes.
- Example: A company has a dimension table for Customers with columns {CustomerID, Name, Address, StartDate, EndDate, and Version}.
- **Scenario**: A customer with CustomerID 1 changes their address from "123 Old St" to "456 New St".
- Before Change:

CustomerID	Name	Address	StartDate	EndDate	Version	IsCurrent
1	John	123 Old St	2020-01-01	9999-12-31	1	True

- After Change:

CustomerID	Name	Address	StartDate	EndDate	Version	IsCurrent
1	John	123 Old St	2020-01-01	2023-01-01	1	False
1	John	456 New St	2023-01-01	9999-12-31	2	True

Type 3 SCD: Add New Column

- **Description:** A new column is added to the dimension table to store the changed value, while the old value is retained in another column. This allows for limited history tracking.
- **Use Case:** Useful when only the current and previous values need to be tracked.
- Example: A company has a dimension table for Employees with columns {EmployeeID, Name, CurrentRole, and PreviousRole}.
- **Scenario**: An employee with EmployeeID 1 changes their role from "Manager" to "Director".
- Before Change:

EmployeeID	Name	CurrentRole	PreviousRole
1	John	Manager	NULL

- After Change:

EmployeeID	Name	CurrentRole	PreviousRole
1	John	Director	Manager

Type 4 SCD: Add New Table

- **Description:** A separate history table is created to store the changes, while the main dimension table holds only the current data.
- **Use Case:** Suitable for scenarios where the main dimension table needs to remain small and only current data is frequently accessed.
  Example: A company has a main dimension table for Products and a separate history table for ProductPriceHistory.
- **Scenario:** The price for ProductID 101 changes from **$10** to **$15**.
- Main Table Before Change:

ProductID	ProductName	Price
101	Product A	10

- History Table Before Change: Empty
- Main Table After Change:

ProductID	ProductName	Price
101	Product A	15

- History Table After Change:

ProductID	Price	StartDate	EndDate
101	10	2020-01-01	2023-01-01

Type 5 SCD: Hybrid (Type 1 + Type4) Mini-Dimension with Current Overwrite

- **Description**: Combines Type 1 and Type 4 approaches. The main dimension table is updated with the latest data (Type 1), and a separate history table is used to track changes (Type 4).
- **Use Case:** Useful when both current data and historical changes need to be efficiently managed.
- Example: A company has a main dimension table for Customers and a separate history table for CustomerAddressHistory.
- **Scenario:** A customer with CustomerID 1 changes their address from "123 Old St" to "456 New St".
- Main Table Before Change:

CustomerID	Name	Address
1	John	123 Old St

- History Table Before Change:

AddressHistoryKey	CustomerID	Address	StartDate	EndDate
1	1	123 Old St	2020-01-01	9999-12-31

- Main Table After Change:

CustomerID	Name	Address
1	John	456 New St

- History Table After Change:

AddressHistoryKey	CustomerID	Address	StartDate	EndDate
1	1	123 Old St	2020-01-01	2023-01-01
2	1	456 New St	2023-01-01	9999-12-31

Types of Keys in Data Warehouse for SCD

- In a data warehouse, **Slowly Changing Dimensions (SCDs)** are used to manage changes to dimension data over time. To implement SCDs effectively, different types of keys are used to uniquely identify records and track historical changes.

- Below are the key types commonly used in SCDs:
  - Surrogate Key
  - Natural Key (Business Key)
  - Composite Key
  - Primary Key
  - Foreign Key
  - Version Key (for SCD Type 2)
  - Effective Date Key (for SCD Type 2)
  - Current Flag (for SCD Type 2)
  - Hash Key (for SCD Type 1 or 3)

Surrogate Key

- A **surrogate key** is a system-generated, unique identifier for each record in a dimension table.
- It is an artificial key (not derived from the source data) and is typically an integer.
- Used to maintain consistency and improve performance in joins.
- Example: Customer_ID (e.g., 101, 102, 103).

Natural Key (Business Key)

- A **natural key** is a unique identifier from the source system or business context.
- It represents the real-world identifier of a record (e.g., employee ID, product code).
- Used to link the data warehouse record back to the source system.
- Example: Employee_ID (e.g., E1234, E5678)

Composite Key

- A **composite key** is a combination of two or more columns that uniquely identify a record.

- Often used when no single column can serve as a natural key.

- Example: Order_ID + Product_ID to uniquely identify an order line item.

Primary Key

- The **primary key** is a column (or set of columns) that uniquely identifies each row in a table.

- In a dimension table, the primary key is often the surrogate key.

- Example: Customer_ID as the primary key in the Customer dimension table

Foreign Key

- A **foreign key** is a column in a fact table that references the primary key of a dimension table.

- Used to establish relationships between fact and dimension tables.

- Example: Customer_ID in the Sales fact table, referencing Customer_ID in the Customer dimension table.

Version Key (for SCD Type 2)

- In SCD Type 2, a **version key** (or version number) is used to track different versions of the same record over time.

- Each version of a record has a unique surrogate key but shares the same natural key.

- Example: Version_Number or Effective_Date to distinguish between historical and current records.

Effective Date Key (for SCD Type 2)

- In SCD Type 2, an **effective date** (or start date) is used to indicate when a record became active.

- Often paired with an expiration date (or end date) to indicate when the record was superseded.

- Example: Start_Date and End_Date columns in the Customer dimension table.

Current Flag (for SCD Type 2)

- A **current flag** is a boolean column used in SCD Type 2 to indicate the most recent version of a record.

- Helps simplify queries by identifying the active record.

- Example: Is_Current (e.g., True for current, False for historical).

Hash Key (for SCD Type 1 or 3)

- A **hash key** is a unique identifier generated by applying a hash function to a set of attribute values in a record.

- The hash function takes input data (e.g., the values of specific columns) and produces a fixed-size string of characters, which is typically a hexadecimal number. This output is known as the hash value or hash key.

- The primary purpose of using a hash key is to quickly determine whether a record has changed.

- By comparing the hash key of the current record with the hash key of the previous version of the record, you can easily identify if any of the relevant attributes have changed.

- Used to detect changes in SCD Type 1 (overwrite) or SCD Type 3 (add new column).

- Example: Hash_Value to compare changes in customer address.

Changing Data Capture (CDC)

- **Changing Data Capture** (CDC) is a set of techniques used in data warehousing and database management to identify and capture changes made to data in a source system.

- CDC allows organizations to track changes (inserts, updates, and deletes) in real-time or near real-time, enabling efficient data synchronization and replication to target systems, such as data warehouses or data lakes.

- CDC ensures that only the changed data is processed and transferred, rather than reloading the entire dataset, which improves efficiency and reduces processing time.

Why is CDC Important?

- **Change Tracking:** CDC focuses on capturing changes to data rather than the entire dataset. This means that only the modified records are identified and processed, which can significantly reduce the volume of data that needs to be transferred.

- **Real-Time or Near Real-Time:** CDC can operate in real-time, capturing changes as they occur, or in near real-time, where changes are captured at regular intervals. This flexibility allows organizations to choose the best approach based on their requirements.

- **Data Synchronization:** CDC is commonly used to synchronize data between source systems (like operational databases) and

target systems (like data warehouses). This ensures that the target system reflects the most current state of the source data.

- **Data Consistency:** Ensures that the target system stays synchronized with the source system.

- **Reduced Load on Source Systems**: Minimizes the impact on source systems by avoiding full-table scans or large data extracts.

How CDC Works

- **CDC** works by monitoring and capturing changes (inserts, updates, and deletes) in the source system.
- These changes are then applied to the target system.
- The process typically involves:
    - **Identifying Changes:** Detecting which records have been added, modified, or deleted.
    - **Capturing Changes**: Extracting the changed data.
    - **Storing Changes**: Storing the changes in a staging area or intermediate storage.
    - **Applying Changes**: Propagating the changes to the target system.

Implementation Methods of CDC

- There are several methods to implement **CDC**, including:
  - Database Triggers
  - Log-Based CDC
  - Timestamp-Based CDC
  - Change Data Tables
  - ETL Tools

Database Triggers

- Triggers can be set up on tables to capture changes (inserts, updates, deletes) and log them into a separate change table.
- This method is straightforward but can introduce overhead on the source database.

Log-Based CDC

- This method involves reading the database transaction logs to capture changes.
- It is often more efficient than triggers because it does not add overhead to the database operations.
- Log-based CDC can capture changes in real-time and is commonly used in enterprise-level solutions.

Timestamp-Based CDC

- In this approach, a timestamp column is added to the source tables to track when records were last modified.
- The CDC process queries the source tables for records with timestamps greater than the last processed timestamp.
- This method is simple but may not capture all changes if multiple updates occur within the same timestamp.

Change Data Tables

- Some databases provide built-in features for CDC, where changes are automatically tracked and stored in change data tables. For example, SQL Server has a feature called Change Data Capture that allows for easy tracking of changes.

ETL Tools

- Many Extract, Transform, Load (ETL) tools offer built-in CDC capabilities, allowing organizations to configure CDC as part of their data integration workflows.
- These tools can handle various methods of CDC and provide a user-friendly interface for managing data flows.

Use Cases for CDC

- **Data Warehousing:** Keeping data warehouses synchronized with source systems.
- **Data Replication:** Replicating data across distributed systems or databases.
- **Real-Time Analytics:** Enabling real-time reporting and analytics.
- **Data Migration:** Migrating data while ensuring changes are captured and applied.
- **Audit and Compliance:** Tracking changes for audit trails and compliance purposes.

Kerolos Alfons
@LinkedIn