CHP-1

30 August 2024 18:32

Define Datawarehouse. Why datawarehouse is required explain with an example.

What is a Data Warehouse?

A **Data Warehouse** is a centralized repository that stores large volumes of structured and sometimes unstructured data from multiple sources. It is designed for query and analysis, and it enables organizations to store historical data, which can then be used for business intelligence (BI), reporting, and data analysis.

Data warehouses typically consolidate data from various operational systems like databases, ERP systems, CRM systems, and other sources into a single, cohesive data model that supports analytical processing and decision-making. The data in a data warehouse is often organized in a way that makes it easy to retrieve, query, and analyze, typically in the form of tables, schemas, and sometimes multidimensional data structures.

Why is a Data Warehouse Required?

Data warehouses are essential for organizations that need to perform complex queries and analyses on large datasets. Here are some reasons why a data warehouse is required:

a. **Consolidation of Data**:
   - In many organizations, data is scattered across multiple systems, such as sales databases, customer relationship management (CRM) systems, financial systems, and more. A data warehouse brings together this disparate data into a single, unified source, allowing for a comprehensive view of the organization's operations.

b. **Historical Data Analysis**:
   - Operational databases are typically designed for real-time transaction processing, and they may not retain historical data for long periods. A data warehouse stores historical data, enabling trend analysis, time-series analysis, and the ability to see how metrics have changed over time.

c. **Improved Query Performance**:
   - Data warehouses are optimized for read-heavy workloads, such as running complex queries, aggregations, and reports. This contrasts with operational databases, which are optimized for write-heavy workloads (inserts, updates). The architecture of a data warehouse allows for faster query performance, even on large datasets.

d. **Decision Support**:
   - By providing a centralized repository of clean, consistent, and reliable data, data warehouses support better decision-making processes. Business users can perform queries, generate reports, and conduct analyses without impacting the performance of operational systems.

e. **Data Quality and Consistency**:
   - Data warehouses often involve processes like data cleaning, transformation, and integration to ensure that the data is accurate and consistent. This helps in maintaining high data quality across the organization, which is crucial for reliable analysis and decision-making.

Example: Why a Data Warehouse is Needed

Scenario: Imagine a retail company that has multiple departments: Sales, Marketing, Finance, and Supply Chain. Each department uses its own system to store data:

- **Sales** might use a sales database to track transactions and customer purchases.
- **Marketing** might use a CRM system to manage campaigns and customer interactions.
- **Finance** might use an accounting system to track revenue, expenses, and other financial data.
- **Supply Chain** might use an inventory management system to track stock levels and supplier data.

Explain the features of data-warehouse

A Data Warehouse has several key features that distinguish it from other data storage systems, particularly operational databases. These features make data warehouses ideal for supporting business intelligence, analytics, and decision-making processes. Here are the main features of a data warehouse:

1. Subject-Oriented

- **Definition**: A data warehouse is organized around key subjects or business areas of an organization, such as customers, sales, products, or finance.
- **Explanation**: Instead of focusing on day-to-day operations, the data warehouse focuses on areas that are important for decision-making. This subject-oriented nature allows users to analyze data in a way that aligns with business objectives.

2. Integrated

- **Definition**: A data warehouse integrates data from multiple, often heterogeneous, sources into a cohesive and consistent format.
- **Explanation**: Data from various systems (e.g., sales, marketing, finance) is standardized and combined into a unified database. This integration process resolves issues such as naming conflicts, differing data formats, and inconsistencies, ensuring that the data is accurate and reliable across the organization.

3. Non-Volatile

- **Definition**: Data in a data warehouse is stable and does not change frequently.
- **Explanation**: Once data is entered into the warehouse, it is not typically updated or deleted; instead, new data is added as time progresses. This feature allows for consistent historical analysis since past data remains unchanged, making it easier to track trends and patterns over time.

4. Time-Variant

- **Definition**: Data in a data warehouse is stored with a time dimension, which allows for analysis over different time periods.
- **Explanation**: Data warehouses store snapshots of data over time, such as daily, weekly, monthly, or yearly records. This time-based perspective enables users to analyze trends, changes, and comparisons over specific periods, which is crucial for business forecasting and performance tracking.

5. Data Granularity

- **Definition**: Data warehouses store data at different levels of detail, ranging from highly detailed (transaction-level) data to summarized or aggregated

data.
   - **Explanation**: The ability to store and access data at varying levels of granularity allows users to perform both detailed and high-level analyses. For example, users can drill down from yearly sales data to specific daily transactions or aggregate data to see broader trends.

6. Optimized for Query Performance

- **Definition**: Data warehouses are designed to handle complex queries and large-scale data analysis efficiently.
- **Explanation**: Unlike operational databases that are optimized for fast transaction processing, data warehouses are optimized for reading and retrieving large amounts of data quickly. Techniques such as indexing, partitioning, and parallel processing are often used to speed up query performance.

7. Historical Data Storage

- **Definition**: Data warehouses store large amounts of historical data, often spanning years or even decades.
- **Explanation**: The historical nature of the data warehouse allows organizations to conduct trend analysis, compare past and present data, and perform time-series analysis. This long-term perspective is critical for strategic planning and forecasting.

8. Separation of Operational and Analytical Processing

- **Definition**: Data warehouses separate the data used for operational tasks from the data used for analytical tasks.
- **Explanation**: This separation ensures that heavy analytical queries do not interfere with the performance of day-to-day operational systems, such as transaction processing systems. As a result, users can run complex analyses without impacting the performance of the organization's primary business activities.

9. Support for Multidimensional Analysis

- **Definition**: Data warehouses often support multidimensional data models, which allow for complex analyses involving multiple dimensions of data (e.g., time, geography, product).
- **Explanation**: Multidimensional analysis is enabled by structures like data cubes, which allow users to explore data across various dimensions, such as analyzing sales data by region, product line, and time period simultaneously.

10. Data Cleansing

- **Definition**: Data warehouses include processes for data cleansing, ensuring that the data is accurate, consistent, and free from errors.
- **Explanation**: During the ETL (Extract, Transform, Load) process, data is cleaned and transformed to ensure that it meets the quality standards required for analysis. This involves correcting inconsistencies, removing duplicates, and handling missing data.

11. Scalability

- **Definition**: Data warehouses are designed to scale as the volume of data grows.
- **Explanation**: As organizations collect more data over time, data warehouses can expand to accommodate increasing amounts of data and users. They are often built on scalable architectures that can handle growing data storage and processing needs.

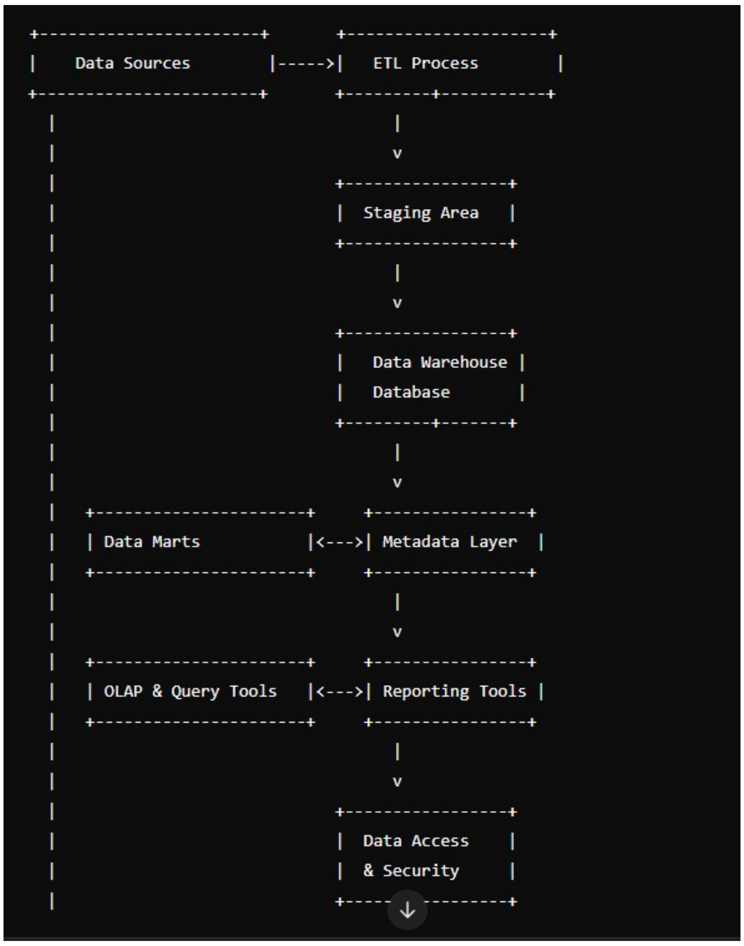
12. Accessibility

- **Definition**: Data in a data warehouse is made accessible to users through various tools and interfaces.
- **Explanation**: Business users, analysts, and decision-makers can access and query the data warehouse using reporting tools, dashboards, and SQL queries. The data is typically presented in a user-friendly format, making it easier for non-technical users to extract insights.

13. Security

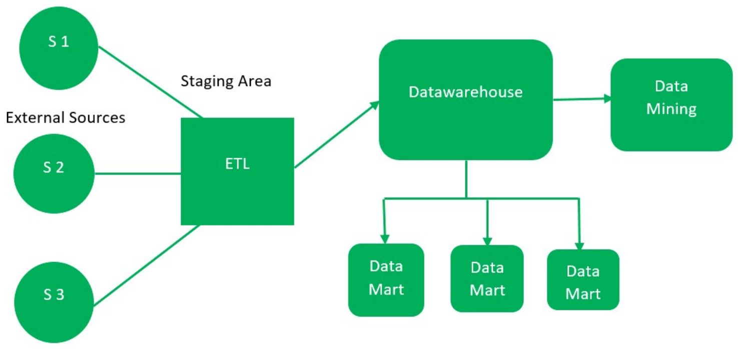
- **Definition**: Data warehouses include robust security features to protect sensitive data.
- **Explanation**: Security measures such as access controls, encryption, and user authentication are implemented to ensure that only authorized users can access or modify the data. This is especially important for protecting sensitive business information and complying with regulatory requirements.

Architecture of data-warehouse



```
+-----------------------+      +---------------------+
|    Data Sources       |----->|   ETL Process       |
+-----------------------+      +---------+-----------+
  |                                      |
  |                                      v
  |                            +----------------+
  |                            |  Staging Area  |
  |                            +----------------+
  |                                      |
  |                                      v
  |                            +----------------+
  |                            |   Data Warehouse |
  |                            |   Database       |
  |                            +---------+-------+
  |                                      |
  |                                      v
  |  +---------------------+     +---------------+
  |  | Data Marts          |<--->| Metadata Layer  |
  |  +---------------------+     +---------------+
  |                                      |
  |                                      v
  |  +---------------------+     +---------------+
  |  | OLAP & Query Tools  |<--->| Reporting Tools |
  |  +---------------------+     +---------------+
  |                                      |
  |                                      v
  |                            +----------------+
  |                            |  Data Access    |
  |                            |  & Security     |
  |                            +----------------+
```

Top Down Approach:



1. **External Sources –**
External source is a source from where data is collected irrespective of the type of data. Data can be structured, semi structured and unstructured as well.

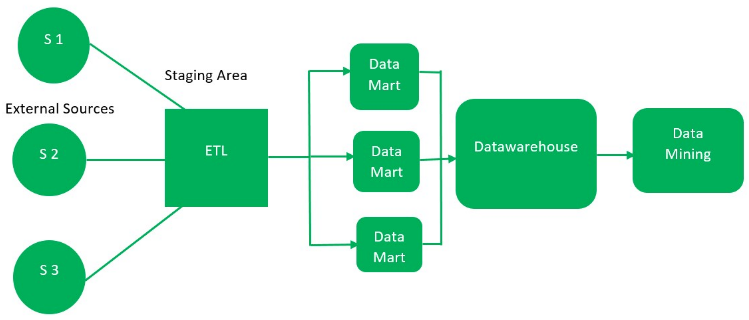
2. **Stage Area –**
Since the data, extracted from the external sources does not follow a particular format, so there is a need to validate this data to load into datawarehouse. For this purpose, it is recommended to use **ETL** tool.
   - **E(Extracted):** Data is extracted from External data source.
   - **T(Transform):** Data is transformed into the standard format.
   - **L(Load):** Data is loaded into datawarehouse after transforming it into the standard format.

3. **Data-warehouse –**
After cleansing of data, it is stored in the datawarehouse as central repository. It actually stores the meta data and the actual data gets stored in the data marts. **Note** that datawarehouse stores the data in its purest form in this top-down approach.

4. **Data Marts –**
Data mart is also a part of storage component. It stores the information of a particular function of an organisation which is handled by single authority. There can be as many number of data marts in an organisation depending upon the functions. We can also say that data mart contains subset of the data stored in datawarehouse.

5. **Data Mining –**
The practice of analysing the big data present in datawarehouse is data mining. It is used to find the hidden patterns that are present in the database or in datawarehouse with the help of algorithm of data mining.
This approach is defined by **Inmon** as – datawarehouse as a central repository for the complete organisation and data marts are created from it after the complete datawarehouse has been created.

Bottom Up Approach

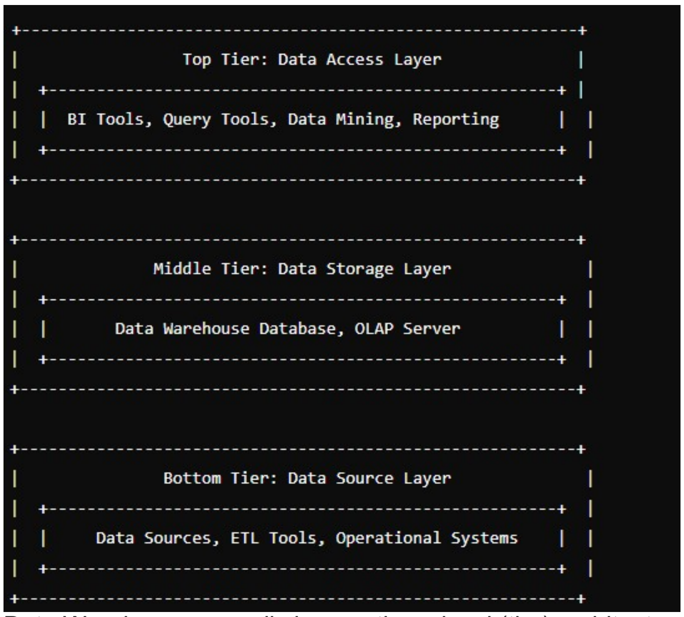


6.
7. First, the data is extracted from external sources (same as happens in top-down approach).

8. Then, the data go through the staging area (as explained above) and loaded into data marts instead of datawarehouse. The data marts are created first and provide reporting capability. It addresses a single business area.

9. These data marts are then integrated into datawarehouse.

Three tier Architecture of Data warehouse

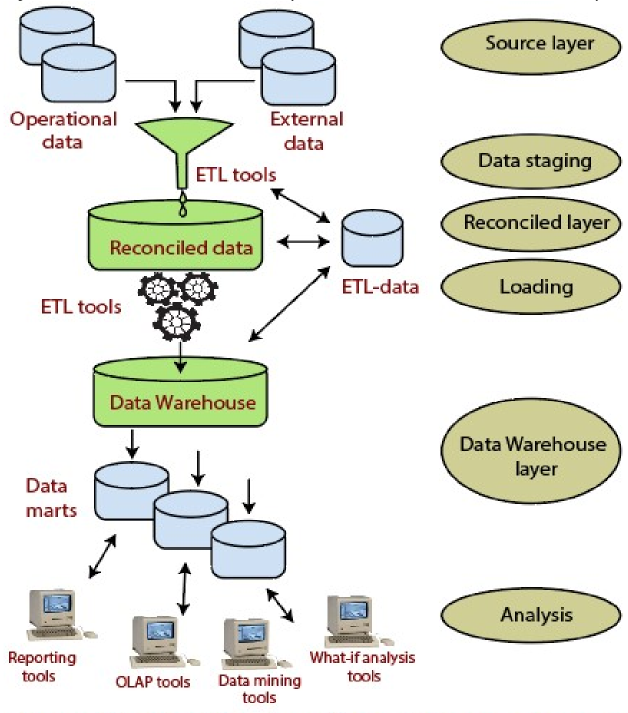


```
+-----------------------------------------------------+
|              Top Tier: Data Access Layer            |
|  +-----------------------------------------------+  |
|  |  BI Tools, Query Tools, Data Mining, Reporting |  |
|  +-----------------------------------------------+  |
+-----------------------------------------------------+


+-----------------------------------------------------+
|           Middle Tier: Data Storage Layer            |
|  +-----------------------------------------------+  |
|  |      Data Warehouse Database, OLAP Server      |  |
|  +-----------------------------------------------+  |
+-----------------------------------------------------+


+-----------------------------------------------------+
|            Bottom Tier: Data Source Layer            |
|  +-----------------------------------------------+  |
|  |    Data Sources, ETL Tools, Operational Systems |  |
|  +-----------------------------------------------+  |
+-----------------------------------------------------+
```

Data Warehouses usually have a three-level (tier) architecture that includes:

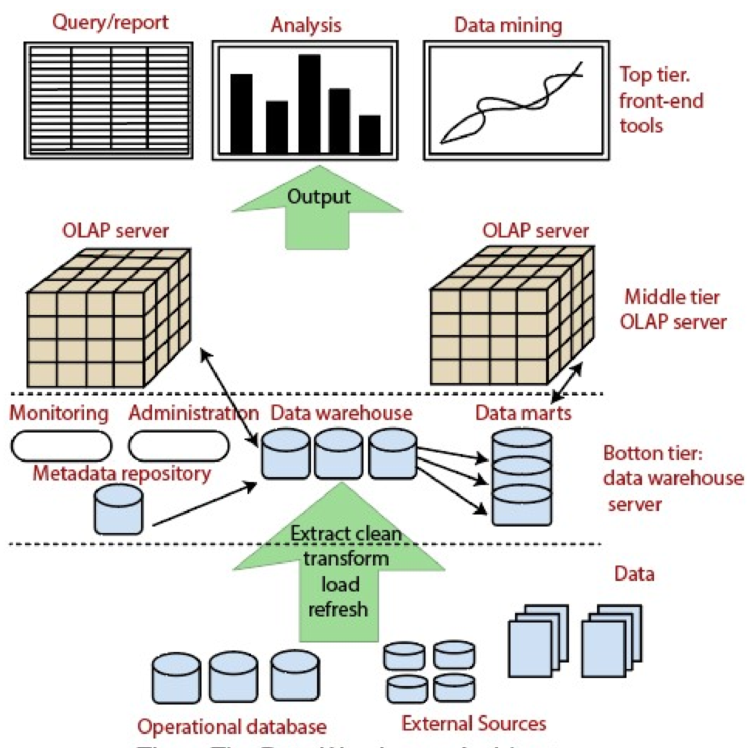
1. Bottom Tier (Data Warehouse Server)
2. Middle Tier (OLAP Server)
3. Top Tier (Front end Tools).

A **bottom-tier** that consists of the **Data Warehouse server**, which is almost always an RDBMS. It may include several specialized data marts and a metadata repository.
Data from operational databases and external sources (such as user profile data provided by external consultants) are extracted using application program interfaces called a gateway. A gateway is provided by the underlying DBMS and allows customer programs to generate SQL code to be executed at a server.
Examples of gateways contain **ODBC** (Open Database Connection) and **OLE-DB** (Open-Linking and Embedding for Databases), by **Microsoft**, and **JDBC** (Java Database Connection).



Three-Tier Architecture for a data warehouse system

A **middle-tier** which consists of an **OLAP server** for fast querying of the data warehouse.
The OLAP server is implemented using either
(1) A **Relational OLAP (ROLAP) model**, i.e., an extended relational DBMS that maps functions on multidimensional data to standard relational operations.
(2) A **Multidimensional OLAP (MOLAP) model**, i.e., a particular purpose server that directly implements multidimensional information and operations.
A **top-tier** that contains **front-end tools** for displaying results provided by OLAP, as well as additional tools for data mining of the OLAP-generated data.
The overall Data Warehouse Architecture is shown in fig:



Three-Tier Data Warehouse Architecture

The **metadata repository** stores information that defines DW objects. It includes the following parameters and information for the middle and the top-tier applications:

1. A description of the DW structure, including the warehouse schema, dimension, hierarchies, data mart locations, and contents, etc.
2. Operational metadata, which usually describes the currency level of the stored data, i.e., active, archived or purged, and warehouse monitoring information, i.e., usage statistics, error reports, audit, etc.
3. System performance data, which includes indices, used to improve data access and retrieval performance.
4. Information about the mapping from operational databases, which provides source **RDBMSs** and their contents, cleaning and transformation rules, etc.
5. Summarization algorithms, predefined queries, and reports business data, which include business terms and definitions, ownership information, etc.

Difference between dimesion modelling and ER Modelling

Dimension Modeling and ER (Entity-Relationship) Modeling are two distinct approaches used in database design, each suited to different types of database systems and applications. Here's a comparison of the two:

1. Purpose

- **Dimension Modeling**:
  - **Use Case**: Primarily used for designing data warehouses and OLAP (Online Analytical Processing) systems where the focus is on enabling efficient querying and reporting.
  - **Goal**: To simplify complex queries and optimize the performance of data retrieval for analytical purposes.
- **ER Modeling**:
  - **Use Case**: Used for designing operational or transactional databases where the focus is on managing day-to-day operations like customer transactions, order processing, etc.
  - **Goal**: To accurately represent the data and relationships within the system, ensuring data integrity and supporting various business processes.

2. Structure

- **Dimension Modeling**:
  - **Star Schema**: The most common structure, consisting of a central fact table (containing quantitative data) connected to multiple dimension tables (containing descriptive attributes).
  - **Snowflake Schema**: A variant of the star schema where dimension tables are normalized, leading to more tables but reduced data redundancy.
  - **Fact Tables**: Store transactional data and numeric measurements.
  - **Dimension Tables**: Store descriptive attributes related to facts (e.g., time, geography, products).
- **ER Modeling**:
  - **Entities and Relationships**: Uses entities to represent real-world objects or concepts and relationships to show how entities are related.
  - **Normalization**: Emphasizes normalization to reduce redundancy and ensure data integrity, resulting in multiple related tables.
  - **Attributes**: Each entity has attributes (fields) that describe its properties.

    - **Relationships**: Defines how entities relate to each other (e.g., one-to-many, many-to-many).

3. Data Redundancy

- **Dimension Modeling**:
    - **Redundancy**: Dimension tables in star schema can have some level of redundancy, which is acceptable and even desired to optimize query performance.
    - **Denormalization**: Often denormalized to allow for simpler queries and faster data retrieval.
- **ER Modeling**:
    - **Redundancy**: Strives to eliminate redundancy through normalization, ensuring that each piece of data is stored in one place.
    - **Normalization**: Typically involves normalizing data into multiple tables to minimize duplication and ensure consistency.

4. Query Performance

- **Dimension Modeling**:
    - **Optimized for Queries**: Designed to make complex queries, especially those involving aggregates (e.g., sums, averages), as fast as possible.
    - **Query Simplicity**: Queries tend to be simpler because of the star schema structure, where all dimensions are directly linked to the fact table.
- **ER Modeling**:
    - **General-Purpose Queries**: Suitable for a wide range of queries, including CRUD (Create, Read, Update, Delete) operations, but complex queries may require multiple joins.
    - **Complexity**: Queries can become complex due to the need to join multiple normalized tables.

5. Use Cases

- **Dimension Modeling**:
    - **Analytics**: Best suited for environments where data analysis, reporting, and business intelligence are the primary objectives.
    - **Data Warehouses**: Commonly used in data warehouses where historical data is analyzed for trends, patterns, and decision-making.
- **ER Modeling**:
    - **Transactional Systems**: Ideal for operational systems like ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), and other transactional applications.
    - **Data Integrity**: Used in systems where maintaining data integrity and supporting complex business rules is critical.

6. Example

- **Dimension Modeling**:
    - **Star Schema Example**:
        - **Fact Table**: Sales (containing SalesAmount, QuantitySold)
        - **Dimension Tables**: Time, Product, Customer, Geography
- **ER Modeling**:
    - **ER Diagram Example**:
        - **Entities**: Customer, Order, Product
        - **Relationships**: Customer places Order, Order contains Product

7. Flexibility

- **Dimension Modeling**:
    - **Less Flexible**: Since it's designed for specific analytical queries, changes to the schema might require significant redesign.
- **ER Modeling**:
    - **More Flexible**: Easier to accommodate changes in business processes or rules due to its detailed and normalized structure.

Summary

- **Dimension Modeling** is best suited for data warehousing and analytical applications where the primary goal is to optimize performance for querying and reporting.
- **ER Modeling** is ideal for operational databases where the focus is on maintaining data integrity, supporting complex relationships, and handling day-to-day transactions.

Diffference between data warehouse and data mart

A **Data Warehouse** and a **Data Mart** are both types of data storage systems used in business intelligence and data management, but they differ in scope, purpose, and complexity. Here's a comparison of the two:

1. Scope

- **Data Warehouse**:
    - **Broad Scope**: A data warehouse is a large, centralized repository that stores data from multiple sources across an entire organization. It encompasses a wide range of subject areas (e.g., sales, finance, HR) and integrates data from various operational systems.
    - **Enterprise-Level**: Designed to serve the entire organization, providing a unified view of all business data.
- **Data Mart**:
    - **Narrow Scope**: A data mart is a smaller, more focused version of a data warehouse. It typically contains data for a specific department, business unit, or subject area (e.g., marketing, sales, finance).
    - **Department-Level**: Tailored to meet the needs of a specific group of users within the organization.

2. Purpose

- **Data Warehouse**:
    - **Comprehensive Analysis**: Used for enterprise-wide data analysis and reporting. It supports complex queries and allows for the analysis of data across multiple subject areas.
    - **Strategic Decision-Making**: Helps senior management and analysts in making strategic decisions by providing a holistic view of the organization's data.
- **Data Mart**:
    - **Targeted Analysis**: Focused on providing data relevant to a specific department or function. It allows for more detailed analysis within a specific area

    of the business.
    - **Operational Decision-Making**: Supports day-to-day decision-making processes within a specific department or business unit.

3. Data Sources

- **Data Warehouse**:
    - **Multiple Sources**: Integrates data from various sources, including operational databases, external data, and other data marts.
    - **Heterogeneous Data**: Can handle diverse data types and formats, aggregating them into a unified structure.
- **Data Mart**:
    - **Fewer Sources**: Typically draws data from a few sources, which may include specific tables or subsets of data from the data warehouse or directly from operational systems.
    - **Homogeneous Data**: The data is usually more homogeneous, focused on a specific business function or area.

4. Design and Complexity

- **Data Warehouse**:
    - **Complex Design**: Requires a comprehensive design process, including data modeling (e.g., star schema, snowflake schema), ETL processes, and ongoing maintenance.
    - **High Complexity**: Involves significant resources and time to develop and maintain due to its large scale and the need to integrate data from multiple sources.
- **Data Mart**:
    - **Simpler Design**: Easier to design and implement than a data warehouse. The data structure is usually simpler, and the ETL process is less complex.
    - **Lower Complexity**: Quicker to develop and deploy, with fewer resources needed compared to a full data warehouse.

5. Implementation Time

- **Data Warehouse**:
    - **Longer Timeframe**: Due to its scale and complexity, implementing a data warehouse can take months or even years.
- **Data Mart**:
    - **Shorter Timeframe**: Data marts can be implemented relatively quickly, often in weeks or a few months, since they are smaller and less complex.

6. Cost

- **Data Warehouse**:
    - **Higher Cost**: More expensive to build, maintain, and manage due to its scale, complexity, and the infrastructure required.
- **Data Mart**:
    - **Lower Cost**: Less costly to implement and maintain, making it a more affordable option for smaller projects or specific departmental needs.

7. Data Integration

- **Data Warehouse**:
    - **Integrated Data**: Combines data from various sources, ensuring a consistent and consolidated view of the entire organization's data.
- **Data Mart**:
    - **Subset of Data**: Contains a subset of the data warehouse’s data or directly from specific operational systems. Data marts may or may not be integrated with other data marts or the data warehouse.

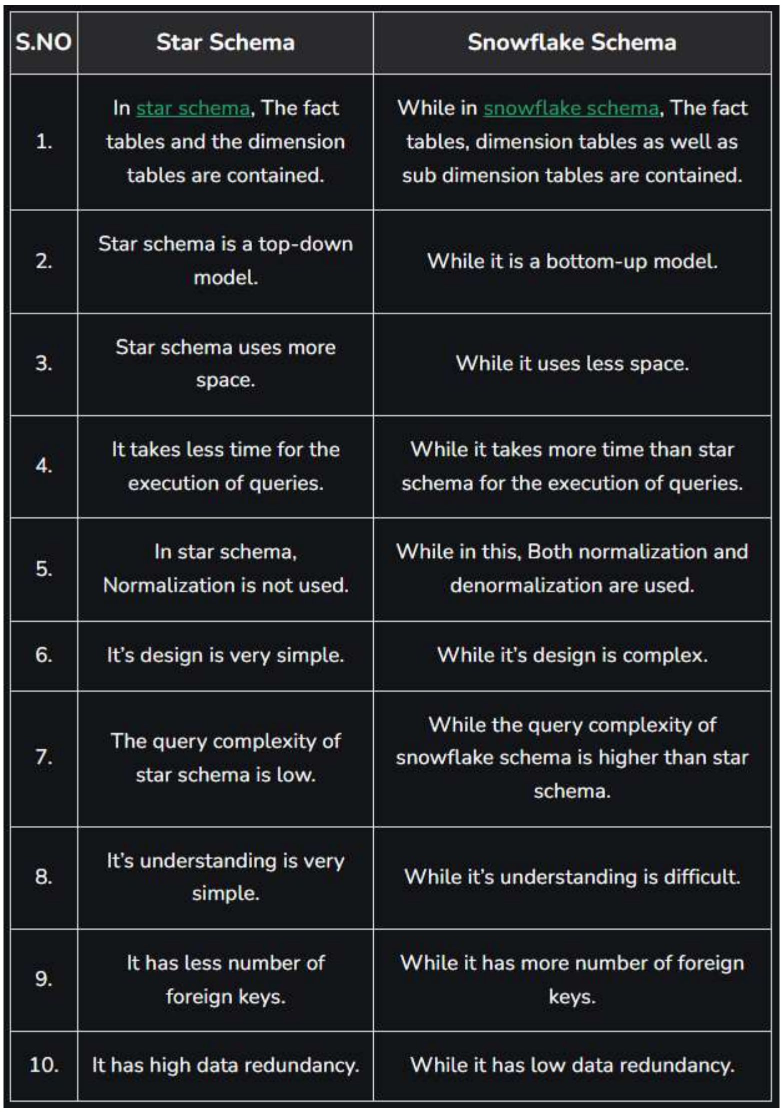
8. User Base

- **Data Warehouse**:
    - **Enterprise-Wide Users**: Used by a wide range of users across the organization, including executives, analysts, and various department heads.
- **Data Mart**:
    - **Department-Specific Users**: Primarily used by specific departments or business units with a focus on particular areas of the business.

9. Example

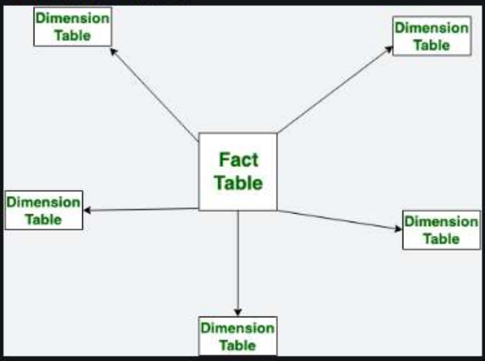
- **Data Warehouse**:
    - **Example**: A large multinational company’s data warehouse that integrates data from finance, sales, HR, and operations, providing a complete view of the organization’s performance.
- **Data Mart**:
    - **Example**: A marketing department’s data mart that contains detailed customer data, campaign performance metrics, and sales data relevant to marketing strategies.

star schema and Snowflake schema

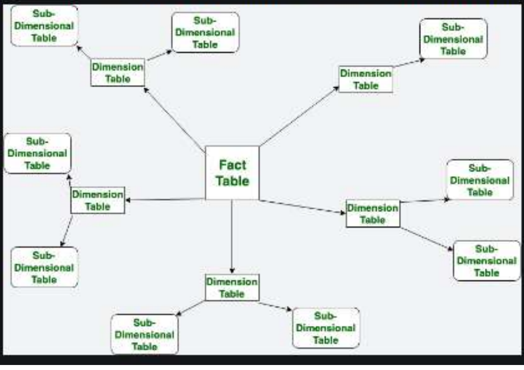


S.NO	Star Schema	Snowflake Schema
1.	In star schema, The fact tables and the dimension tables are contained.	While in snowflake schema, The fact tables, dimension tables as well as sub dimension tables are contained.
2.	Star schema is a top-down model.	While it is a bottom-up model.
3.	Star schema uses more space.	While it uses less space.
4.	It takes less time for the execution of queries.	While it takes more time than star schema for the execution of queries.
5.	In star schema, Normalization is not used.	While in this, Both normalization and denormalization are used.
6.	It's design is very simple.	While it's design is complex.
7.	The query complexity of star schema is low.	While the query complexity of snowflake schema is higher than star schema.
8.	It's understanding is very simple.	While it's understanding is difficult.
9.	It has less number of foreign keys.	While it has more number of foreign keys.
10.	It has high data redundancy.	While it has low data redundancy.

Star Schema: Star schema is the type of multidimensional model which is used for data warehouse. In star schema, The fact tables and the dimension tables are contained. In this schema fewer foreign-key join is used. This schema forms a star with fact table and dimension tables.



Snowflake Schema: Snowflake Schema is also the type of multidimensional model which is used for data warehouse. In snowflake schema, The fact tables, dimension tables as well as sub dimension tables are contained. This schema forms a snowflake with fact tables, dimension tables as well as sub-dimension tables.



Explain Factless fact table with an example

A **factless fact table** is a type of fact table in a data warehouse schema that does not contain any measures or numeric data. Instead, it captures the relationships between different dimensions by recording events or occurrences. Factless fact tables are useful for tracking the occurrence of events or for capturing details about the absence of events.

Purpose of Factless Fact Tables

1. **Tracking Events**: They record the occurrence of specific events or activities without measuring any quantitative metrics.
2. **Capturing Relationships**: They help capture the relationships between dimensions, which can be important for analysis, even if no quantitative data is associated.
3. **Handling Absence**: They can also be used to record the absence of certain events or activities.

Example of Factless Fact Table

Scenario: Suppose you are managing a data warehouse for a university that tracks student attendance in various classes. You want to capture when students attended classes without recording any numerical measure.

Dimensions

1. **Student**: Contains information about students.
   - Attributes: Student_ID, Student_Name
2. **Class**: Contains information about classes.
   - Attributes: Class_ID, Class_Name, Instructor_ID
3. **Date**: Contains information about the date of the class.
   - Attributes: Date_ID, Date, Month, Year

Factless Fact Table

- **Fact Table**: Attendance
  - Dimensions: Student_ID, Class_ID, Date_ID
  - Measures: None (no numeric data)

Factless Fact Table: Attendance

Student_ID	Class_ID	Date_ID
101	201	20230901

102	201	20230901
101	202	20230902
103	201	20230902

Explanation

1. **Tracking Attendance**: The Attendance factless fact table tracks which students attended which classes on which dates. It does not measure any quantities but records the presence of students.
2. **Analyzing Data**: You can analyze this factless fact table to answer questions such as:
   - How many students attended a particular class on a specific date?
   - Which classes did a particular student attend?
3. **Aggregations and Analysis**: Even without numeric measures, you can aggregate this table to count occurrences, such as the number of students attending each class, or to track attendance patterns over time.

Example Queries

```
1. Count of Students per Class
    sql
    Copy code
    SELECT Class_ID, COUNT(Student_ID) AS NumberOfStudents
    FROM Attendance
    GROUP BY Class_ID;
    This query counts the number of students attending each class.
 2. Attendance by Student:
    sql
    Copy code
    SELECT Student_ID, COUNT(Class_ID) AS NumberOfClassesAttended
    FROM Attendance
    GROUP BY Student_ID;
```

This query counts how many classes each student attended.

Factless fact tables are useful for capturing and analyzing relationships and events that do not involve numeric measures, providing valuable insights into the structure and interactions within a data warehouse.

Explain Fact Constellation Schema (Galaxy of star Families of star) with an example

A **Fact Constellation Schema**, also known as a **Galaxy Schema**, is a complex data warehouse schema that involves multiple fact tables sharing dimension tables. It combines multiple star schemas or snowflake schemas, allowing for a more flexible and detailed representation of data, often reflecting different business processes or analytical perspectives.

Structure of a Fact Constellation Schema

1. **Fact Tables**: Multiple fact tables capture different aspects of the business process. Each fact table stores measures related to a specific subject area.
2. **Shared Dimension Tables**: Dimensions are shared among different fact tables. These dimensions provide context and allow for multi-dimensional analysis across various fact tables.

Example of a Fact Constellation Schema

Scenario: Consider a retail business that wants to analyze sales, inventory, and shipments. The business needs to track different metrics related to sales transactions, inventory levels, and shipment details.

Dimensions

1. **Product Dimension**:
   - Attributes: Product_ID, Product_Name, Category, Brand
2. **Time Dimension**:
   - Attributes: Date_ID, Date, Month, Quarter, Year
3. **Store Dimension**:
   - Attributes: Store_ID, Store_Name, Location
4. **Supplier Dimension**:
   - Attributes: Supplier_ID, Supplier_Name, Contact_Info

Fact Tables

1. **Sales Fact Table**:
   - Measures: Sales_Amount, Quantity_Sold
   - Foreign Keys: Product_ID, Date_ID, Store_ID
2. **Inventory Fact Table**:
   - Measures: Inventory_Level
   - Foreign Keys: Product_ID, Date_ID, Store_ID
3. **Shipment Fact Table**:
   - Measures: Shipment_Cost, Quantity_Shipped
   - Foreign Keys: Product_ID, Date_ID, Supplier_ID, Store_ID

Fact Constellation Schema Diagram:

Explanation

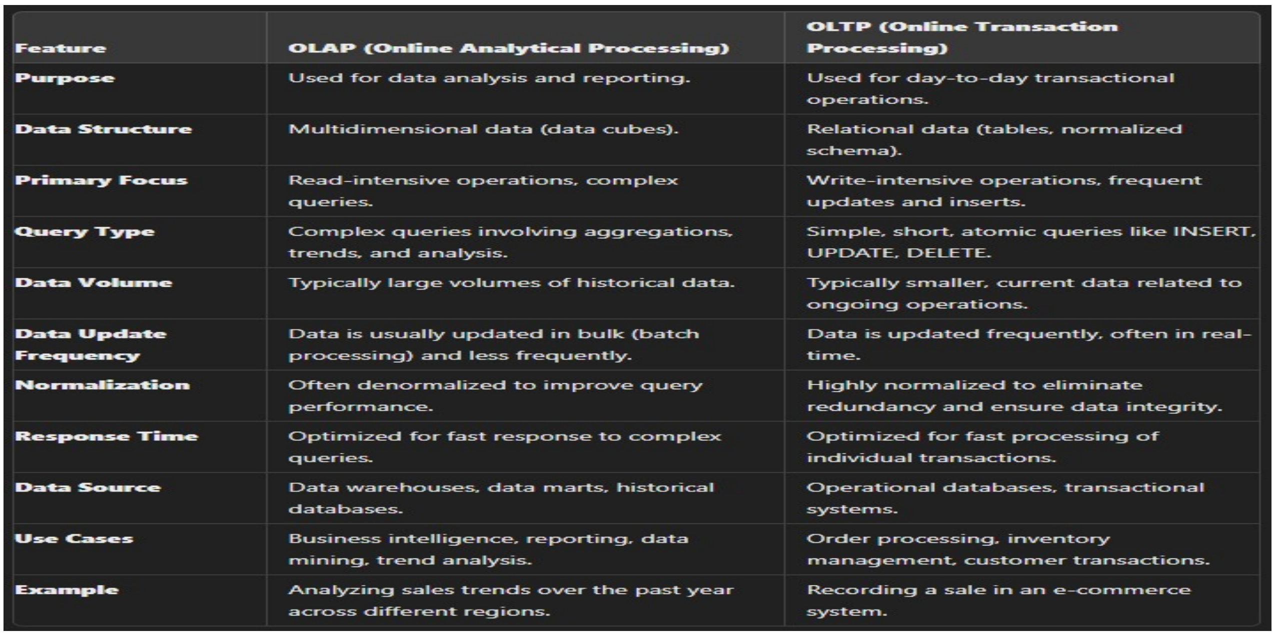
1. **Shared Dimensions**: The Product, Time, and Store dimensions are shared among the Sales, Inventory, and Shipment fact tables. This allows for comprehensive multi-dimensional analysis across different aspects of the business.
2. **Multiple Fact Tables**:
   - The **Sales Fact Table** tracks sales transactions, including sales amounts and quantities sold.
   - The **Inventory Fact Table** monitors inventory levels.
   - The **Shipment Fact Table** records details about shipments, including shipment costs and quantities shipped.
3. **Multi-dimensional Analysis**: This schema supports complex queries and analyses, such as:
   - Total sales by product and store for each month.
   - Inventory levels and changes over time.
   - Shipment costs and quantities by supplier and product.

```
Example Queries
    1. Total Sales by Product and Store:
sql
Copy code
SELECT Product_Name, Store_Name, SUM(Sales_Amount) AS Total_Sales
FROM Sales
JOIN Product ON Sales.Product_ID = Product.Product_ID
JOIN Store ON Sales.Store_ID = Store.Store_ID
GROUP BY Product_Name, Store_Name;
    2. Inventory Levels by Product and Store Over Time:
sql
Copy code
SELECT Product_Name, Store_Name, Date, SUM(Inventory_Level) AS Total_Inventory
FROM Inventory
JOIN Product ON Inventory.Product_ID = Product.Product_ID
JOIN Store ON Inventory.Store_ID = Store.Store_ID
JOIN Time ON Inventory.Date_ID = Time.Date_ID
GROUP BY Product_Name, Store_Name, Date;
    3. Shipment Costs by Supplier and Product:
sql
Copy code
SELECT Supplier_Name, Product_Name, SUM(Shipment_Cost) AS Total_Shipment_Cost
FROM Shipment
JOIN Product ON Shipment.Product_ID = Product.Product_ID
JOIN Supplier ON Shipment.Supplier_ID = Supplier.Supplier_ID
GROUP BY Supplier_Name, Product_Name;
```

The Fact Constellation Schema provides a flexible and powerful way to model complex data relationships and supports detailed and multi-faceted analytical queries.

Explain OLAP. Differentitate between OLAP and OLTP

- **OLAP** is designed for analyzing large volumes of historical data, allowing users to perform complex queries and derive insights that support decision-making.
- **OLTP** is focused on managing and processing day-to-day transactions in real-time, ensuring data integrity and efficient transaction processing.



Feature	OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
Purpose	Used for data analysis and reporting.	Used for day-to-day transactional operations.
Data Structure	Multidimensional data (data cubes).	Relational data (tables, normalized schema).
Primary Focus	Read-intensive operations, complex queries.	Write-intensive operations, frequent updates and inserts.
Query Type	Complex queries involving aggregations, trends, and analysis.	Simple, short, atomic queries like INSERT, UPDATE, DELETE.
Data Volume	Typically large volumes of historical data.	Typically smaller, current data related to ongoing operations.
Data Update Frequency	Data is usually updated in bulk (batch processing) and less frequently.	Data is updated frequently, often in real-time.
Normalization	Often denormalized to improve query performance.	Highly normalized to eliminate redundancy and ensure data integrity.
Response Time	Optimized for fast response to complex queries.	Optimized for fast processing of individual transactions.
Data Source	Data warehouses, data marts, historical databases.	Operational databases, transactional systems.
Use Cases	Business intelligence, reporting, data mining, trend analysis.	Order processing, inventory management, customer transactions.
Example	Analyzing sales trends over the past year across different regions.	Recording a sale in an e-commerce system.

OLAP operations with an example

OLAP (Online Analytical Processing) operations are designed to enable multidimensional data analysis and support complex queries for decision-making and business intelligence. These operations allow users to interact with and analyze data from different perspectives. Here's an overview of the main OLAP operations with examples:

1. Drill-Down

Definition: Drill-down is an OLAP operation that allows users to navigate from more aggregated data to more detailed data. It involves breaking down data into finer levels of granularity.

Example:

- **Scenario**: A retail manager wants to analyze monthly sales data and then drill down to view daily sales data.
- **Operation**: Starting with a view of total sales by month, the manager can drill down to see sales figures for each day within a specific month. This helps in identifying specific days with unusual sales patterns.

2. Roll-Up

Definition: Roll-up is the opposite of drill-down. It involves aggregating detailed data into higher levels of summary. This operation consolidates data to a more summarized level.

Example:

- **Scenario**: A financial analyst reviews daily transaction data and wants to see a monthly summary.
- **Operation**: By rolling up, the analyst consolidates daily transaction data into monthly totals, which provides a higher-level view of overall trends and performance for the month.

3. Slice

Definition: Slice refers to the operation of selecting a single dimension from a multidimensional cube to analyze a specific aspect of the data.

Example:

- **Scenario**: A sales analyst wants to examine sales data for a specific region.
- **Operation**: By slicing the sales cube along the "Region" dimension to include only data for the "North America" region, the analyst can focus on sales performance in that particular region, ignoring other regions.

4. Dice

Definition: Dice is an OLAP operation that involves selecting specific values from multiple dimensions to create a subcube. It filters data based on criteria across multiple dimensions.

Example:

- **Scenario**: A marketing manager wants to analyze sales data for a specific product category and time period.
- **Operation**: By dicing the sales cube to include only data for "Electronics" as the product category and the "Q1 2024" time period, the manager creates a subcube that provides detailed insights into sales performance for that category and period.

5. Pivot (Rotate)

Definition: Pivot, or rotate, is an OLAP operation that changes the orientation of the multidimensional view to provide different perspectives on the data. It involves reconfiguring the dimensions and measures in the report.

Example:

- **Scenario**: A business analyst wants to view sales data by product and region, but also wants to see the data by region and product.
- **Operation**: By pivoting the data, the analyst can switch the dimensions of the view. Initially, the report may show sales by product (rows) and region (columns). After pivoting, the report displays sales by region (rows) and product (columns), providing a different perspective on the data.

6. Aggregation

Definition: Aggregation involves summarizing data by applying functions like sum, average, count, etc., across one or more dimensions. It is often used in combination with other operations to provide summarized views of data.

Example:

- **Scenario**: A company wants to analyze the average sales per product category across different regions.
- **Operation**: By aggregating the sales data, the system calculates the average sales for each product category, providing a summary of how well each category performs on average.

OLTP Operations

OLTP (Online Transaction Processing) systems are designed for managing day-to-day transactional data and are optimized for quick query processing and transaction handling. OLTP operations are primarily concerned with inserting, updating, and deleting data in real-time. Here are some key OLTP operations with examples:

1. Insert

Definition: The insert operation adds new records to the database. It is used to capture new transactional data or add new entries into the system.
Example:

- **Scenario**: A customer places a new order on an e-commerce website.
- **Operation**: The OLTP system performs an insert operation to add a new record into the Orders table with details such as order ID, customer ID, product ID, quantity, and order date.

```
INSERT INTO Orders (OrderID, CustomerID, ProductID, Quantity, OrderDate)
VALUES (101, 1001, 2002, 3, '2024-09-04');
```

2. Update

Definition: The update operation modifies existing records in the database. It is used to change data based on new information or updates in the system.
Example:

- **Scenario**: A customer updates their shipping address on an e-commerce site.
- **Operation**: The OLTP system performs an update operation to change the shipping address for a specific customer in the Customers table.

```
UPDATE Customers
SET ShippingAddress = '123 New Address, City, State, ZIP'
WHERE CustomerID = 1001;
```

3. Delete

Definition: The delete operation removes existing records from the database. It is used to delete data that is no longer needed or to handle data cleanup.
Example:

- **Scenario**: A product is discontinued and needs to be removed from the inventory system.
- **Operation**: The OLTP system performs a delete operation to remove the record of the discontinued product from the Products table.

```
DELETE FROM Products
WHERE ProductID = 2002;
```

4. Select

Definition: The select operation retrieves data from the database. It is used to query and display information based on specified criteria.
Example:

- **Scenario**: A cashier needs to view all orders placed by a particular customer.
- **Operation**: The OLTP system performs a select operation to retrieve order details for the specified customer from the Orders table.

```
SELECT * FROM Orders
WHERE CustomerID = 1001;
```

5. Transaction Management

Definition: OLTP systems support transaction management to ensure that multiple operations are completed successfully and consistently. Transactions typically involve a series of operations that are treated as a single unit of work.
Example:

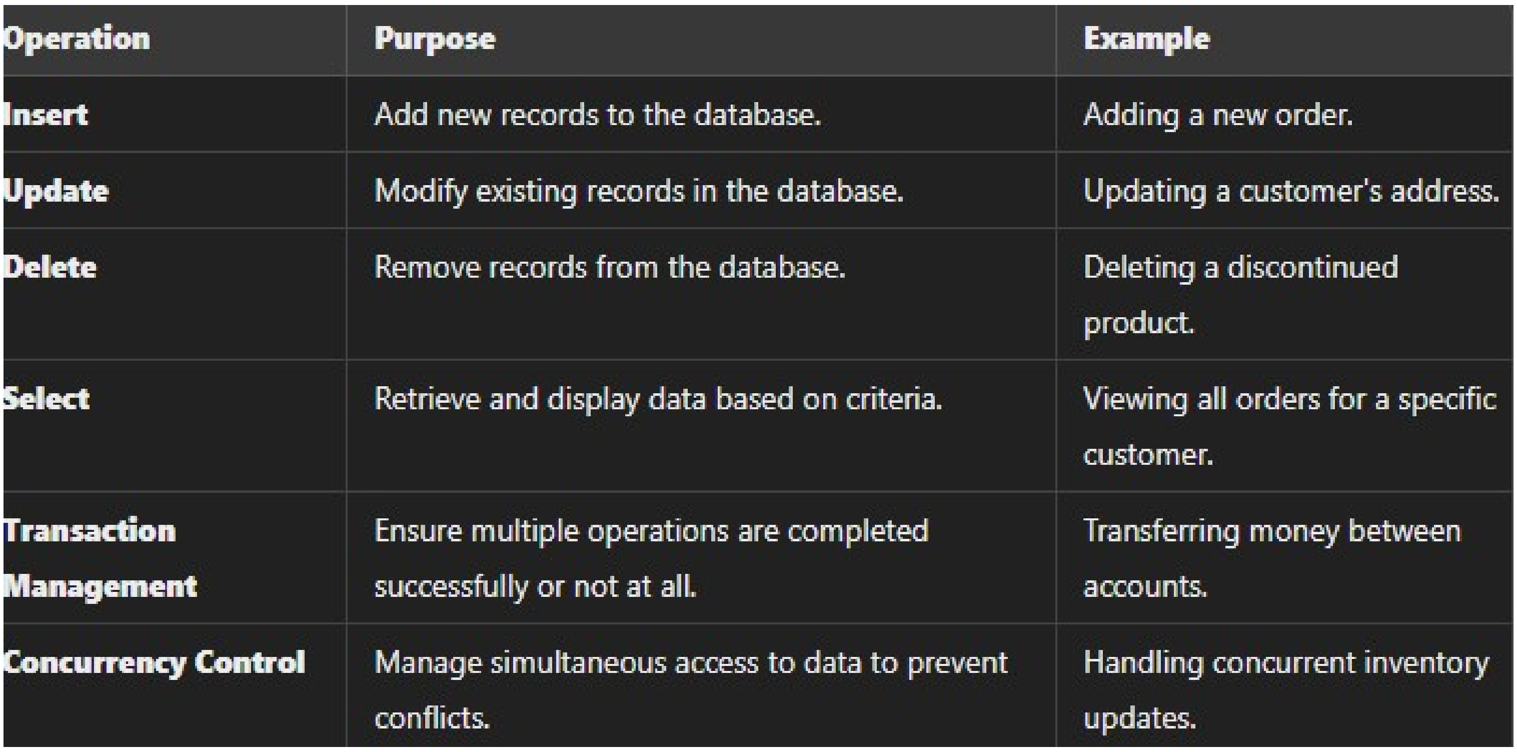
- **Scenario**: A customer transfers money from one bank account to another.
- **Operation**: The OLTP system performs a transaction involving multiple operations: deducting money from the source account, adding it to the destination account, and recording the transaction. The system ensures that all operations are completed successfully, or none are applied if an error occurs.

```
BEGIN TRANSACTION;
UPDATE Accounts
SET Balance = Balance - 100
WHERE AccountID = 12345;
UPDATE Accounts
SET Balance = Balance + 100
WHERE AccountID = 67890;
INSERT INTO Transactions (TransactionID, FromAccountID, ToAccountID, Amount, Date)
VALUES (1, 12345, 67890, 100, '2024-09-04');
COMMIT TRANSACTION;
```

6. Concurrency Control

Definition: Concurrency control mechanisms ensure that multiple users or processes can interact with the database simultaneously without causing conflicts or

data inconsistencies.
Example:
- **Scenario**: Two cashiers process transactions at the same time, updating inventory levels.
- **Operation**: The OLTP system uses concurrency control techniques like locking and isolation levels to ensure that inventory updates by one cashier do not interfere with updates by the other, maintaining data consistency and accuracy.



Operation	Purpose	Example
Insert	Add new records to the database.	Adding a new order.
Update	Modify existing records in the database.	Updating a customer's address.
Delete	Remove records from the database.	Deleting a discontinued product.
Select	Retrieve and display data based on criteria.	Viewing all orders for a specific customer.
Transaction Management	Ensure multiple operations are completed successfully or not at all.	Transferring money between accounts.
Concurrency Control	Manage simultaneous access to data to prevent conflicts.	Handling concurrent inventory updates.

Explain various OLAP models(OLAP servers)

OLAP (Online Analytical Processing) systems are designed for complex data analysis and multidimensional querying. There are several OLAP models (also known as OLAP servers) that offer different ways to store, process, and analyze data. Each model has its own strengths and use cases. Here's a detailed explanation of the various OLAP models:

1. MOLAP (Multidimensional OLAP)

Overview: MOLAP systems store data in multidimensional cubes. These cubes are pre-aggregated and optimized for fast query performance. MOLAP is known for its ability to quickly perform complex queries and provide a multidimensional view of data.
Key Features:
- **Pre-Aggregated Data**: Data is pre-summarized and stored in multidimensional cubes, which allows for fast retrieval of aggregated information.
- **Multidimensional Storage**: Data is organized along multiple dimensions, such as time, geography, and product.
- **High Performance**: Optimized for high-speed query performance due to the pre-computed aggregations.

Example:
- **Sales Analysis Cube**: A MOLAP system might store sales data in a cube with dimensions like time (months, quarters), geography (regions), and products (categories). Queries can quickly aggregate data, such as total sales by region for the last quarter.

Advantages:
- Fast query performance due to pre-aggregation.
- Rich support for complex calculations and multidimensional analysis.

Disadvantages:
- Can be less scalable compared to other models.
- Data loading and cube processing can be resource-intensive.

2. ROLAP (Relational OLAP)

Overview: ROLAP systems use relational databases to store data and generate multidimensional views dynamically. ROLAP systems leverage SQL queries to retrieve and aggregate data from relational tables.
Key Features:
- **Relational Storage**: Data is stored in traditional relational database tables.
- **On-the-Fly Aggregation**: Aggregations and multidimensional views are computed at query time.
- **Scalability**: Generally more scalable than MOLAP due to the use of relational databases.

Example:
- **Inventory Management System**: A ROLAP system might store detailed inventory data in relational tables. Users can query the system to see aggregated data, such as total inventory value by category and time period, with the aggregation done dynamically.

Advantages:
- More scalable and flexible with large datasets.
- Can work with existing relational databases.

Disadvantages:
- Query performance may be slower compared to MOLAP due to on-the-fly aggregation.
- More complex to optimize for performance.

3. HOLAP (Hybrid OLAP)

Overview: HOLAP systems combine features of both MOLAP and ROLAP. They use relational databases for detailed data storage and multidimensional cubes for pre-aggregated data. This hybrid approach aims to balance performance and scalability.

Key Features:

- **Hybrid Storage**: Detailed data is stored in relational tables, while aggregated data is stored in multidimensional cubes.
- **Flexible Analysis**: Provides both fast query performance for pre-aggregated data and detailed analysis through relational tables.
- **Optimized Performance**: Attempts to offer the best of both MOLAP and ROLAP.

Example:

- **Customer Analytics System**: A HOLAP system might store detailed customer transaction data in a relational database, while using multidimensional cubes to provide summarized views, such as total spending by customer segment and time period.

Advantages:

- Balances the performance benefits of MOLAP with the scalability of ROLAP.
- Provides both detailed and summarized data access.

Disadvantages:

- Complexity in managing and maintaining both relational and multidimensional data stores.
- Can be more challenging to implement and optimize.

4. DOLAP (Desktop OLAP)

Overview: DOLAP systems are designed for desktop-based analysis, allowing users to perform OLAP operations on local machines. DOLAP solutions are typically used for individual or small-scale analytical needs.

Key Features:

- **Local Data Storage**: Data is stored locally on the user's desktop or device.
- **Limited Scalability**: Suitable for small datasets and individual analysis rather than enterprise-level data.
- **Offline Capability**: Users can analyze data without requiring a constant connection to a central server.

Example:

- **Personal Finance Analysis**: A DOLAP tool might be used by individuals to analyze personal budget data, providing a local multidimensional view of income and expenses.

Advantages:

- Allows for offline analysis and local data manipulation.
- Easy to use for small-scale analytical tasks.

Disadvantages:

- Limited scalability and performance compared to server-based OLAP solutions.
- Not suitable for large-scale enterprise data analysis.

OLAP Model	Data Storage	Aggregation Method	Use Cases	Advantages	Disadvantages
MOLAP	Multidimensional cubes	Pre-aggregated data	Complex queries, multidimensional analysis	Fast query performance, rich multidimensional views	Less scalable, resource-intensive data loading
ROLAP	Relational database tables	On-the-fly aggregation	Large datasets, dynamic analysis	Scalable, flexible with relational databases	Slower query performance, complex optimization
HOLAP	Hybrid of relational and cubes	Mixed (pre-aggregated and on-the-fly)	Balanced performance and scalability	Combines strengths of MOLAP and ROLAP	Complexity in managing hybrid storage
DOLAP	Local desktop storage	Local data manipulation	Individual or small-scale analysis	Offline capability, easy for small datasets	Limited scalability and performance

Each OLAP model has its specific strengths and is suited to different types of analytical tasks and data sizes. The choice of OLAP model depends on the performance requirements, data scalability needs, and the complexity of queries and analyses.

Explain ETL process in detail

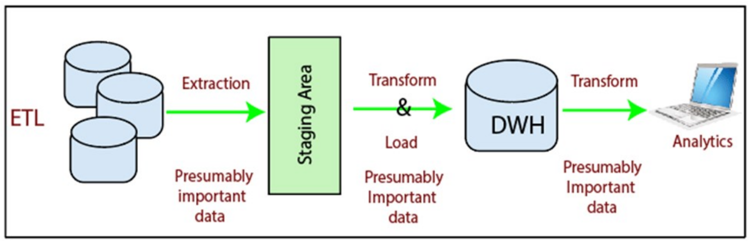
ETL (Extract, Transform, and Load) Process

What is ETL?

The mechanism of extracting information from source systems and bringing it into the data warehouse is commonly called **ETL**, which stands for **Extraction, Transformation and Loading**.

The ETL process requires active inputs from various stakeholders, including developers, analysts, testers, top executives and is technically challenging.

To maintain its value as a tool for decision-makers, Data warehouse technique needs to change with business changes. ETL is a recurring method (daily, weekly, monthly) of a Data warehouse system and needs to be agile, automated, and well documented.


ETL
Extraction
Presumably important data
Staging Area
Transform & Load
Presumably Important data
DWH
Transform
Presumably Important data
Analytics

Extraction

- Extraction is the operation of extracting information from a source system for further use in a data warehouse environment. This is the first stage of the ETL process.
- Extraction process is often one of the most time-consuming tasks in the ETL.
- The source systems might be complicated and poorly documented, and thus determining which data needs to be extracted can be difficult.
- The data has to be extracted several times in a periodic manner to supply all changed data to the warehouse and keep it up-to-date.

Cleansing

The cleansing stage is crucial in a data warehouse technique because it is supposed to improve data quality. The primary data cleansing features found in ETL tools are rectification and homogenization. They use specific dictionaries to rectify typing mistakes and to recognize synonyms, as well as rule-based cleansing to enforce domain-specific rules and defines appropriate associations between values.
The following examples show the essential of data cleaning:
If an enterprise wishes to contact its users or its suppliers, a complete, accurate and up-to-date list of contact addresses, email addresses and telephone numbers must be available.
If a client or supplier calls, the staff responding should be quickly able to find the person in the enterprise database, but this need that the caller's name or his/her company name is listed in the database.
If a user appears in the databases with two or more slightly different names or different account numbers, it becomes difficult to update the customer's information.

Transformation

Transformation is the core of the reconciliation phase. It converts records from its operational source format into a particular data warehouse format. If we implement a three-layer architecture, this phase outputs our reconciled data layer.
The following points must be rectified in this phase:

- Loose texts may hide valuable information. For example, XYZ PVT Ltd does not explicitly show that this is a Limited Partnership company.
- Different formats can be used for individual data. For example, data can be saved as a string or as three integers.

Following are the main transformation processes aimed at populating the reconciled data layer:

- Conversion and normalization that operate on both storage formats and units of measure to make data uniform.
- Matching that associates equivalent fields in different sources.
- Selection that reduces the number of source fields and records.

Cleansing and **Transformation** processes are often closely linked in ETL tools.

Loading

The **Load** is the process of writing the data into the target database. During the load step, it is necessary to ensure that the load is performed correctly and with as little resources as possible.
Loading can be carried in two ways:

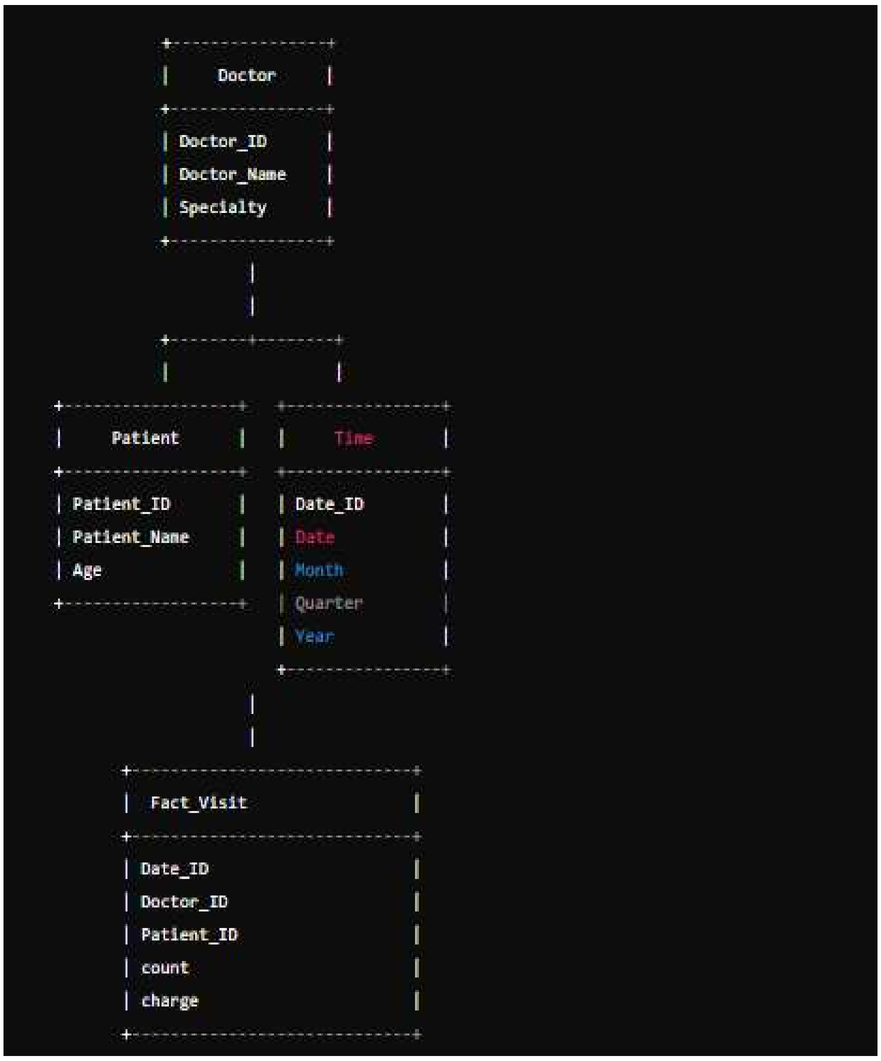
1. **Refresh:** Data Warehouse data is completely rewritten. This means that older file is replaced. Refresh is usually used in combination with static extraction to populate a data warehouse initially.
2. **Update:** Only those changes applied to source information are added to the Data Warehouse. An update is typically carried out without deleting or modifying preexisting data. This method is used in combination with incremental extraction to update data warehouses regularly.

Suppose that a data warehouse consists of the three dimensions time, doctor and patient, and the two measures count and charge, where charge is the fee that a doctor charges a patient for a visit. (i) Draw a star schema diagram for the above data warehouse. (ii) Starting with the base cuboid [day, doctor, patient], what specific OLAP operations should be performed in

order to list the total fee collected by each doctor in 2010?

i. Star Schema Diagram

A star schema consists of a central fact table connected to several dimension tables. In this scenario, the data warehouse has the dimensions **time**, **doctor**, and **patient**, and measures **count** and **charge**.



```
                +----------------+
                |     Doctor     |
                +----------------+
                | Doctor_ID      |
                | Doctor_Name    |
                | Specialty      |
                +----------------+
                         |
                         |
                +--------+--------+
                |                 |
     +-------------------+  +-----------------+
     |     Patient       |  |      Time       |
     +-------------------+  +-----------------+
     | Patient_ID        |  | Date_ID         |
     | Patient_Name      |  | Date            |
     | Age               |  | Month           |
     +-------------------+  | Quarter         |
                            | Year            |
                            +-----------------+
                         |
                         |
           +------------------------------+
           |  Fact_Visit                  |
           +------------------------------+
           | Date_ID                      |
           | Doctor_ID                    |
           | Patient_ID                   |
           | count                        |
           | charge                       |
           +------------------------------+
```

- **Fact Table**: Fact_Visit
  - Measures: count, charge
  - Foreign Keys: Date_ID, Doctor_ID, Patient_ID
- **Dimension Tables**:
  - **Doctor**: Contains details about doctors.
    - Attributes: Doctor_ID, Doctor_Name, Specialty
  - **Patient**: Contains details about patients.
    - Attributes: Patient_ID, Patient_Name, Age
  - **Time**: Contains details about time.
    - Attributes: Date_ID, Date, Month, Quarter, Year

ii. OLAP Operations to List the Total Fee Collected by Each Doctor in 2010

To list the total fee collected by each doctor in 2010, starting from the base cuboid [day, doctor, patient], you would perform the following OLAP operations:

1. **Roll-up from day to year**:
   - **Operation**: Aggregate data from the day level to the year level.
   - **Purpose**: To aggregate the data at the year level, focusing on the year 2010.
2. **Select (Filter)**:
   - **Operation**: Filter the data to include only records where Year = 2010.
   - **Purpose**: To focus on the specific year of interest, 2010.
3. **Aggregation**:
   - **Operation**: Compute the total fee (charge) for each doctor.
   - **Purpose**: To get the sum of charges collected by each doctor.

OLAP Operations in Sequence:

1. **Start with base cuboid**: [day, doctor, patient].

2. **Roll-up** from day to year to aggregate data by year.
3. **Filter** the data to include only those records where Year = 2010.
4. **Aggregation** to compute the total fee (charge) for each doctor.

```
SELECT Doctor_ID, SUM(charge) AS TotalFee
FROM Fact_Visit
JOIN Time ON Fact_Visit.Date_ID = Time.Date_ID
WHERE Time.Year = 2010
GROUP BY Doctor_ID;
```

This query computes the total fee collected by each doctor in the year 2010. It joins the Fact_Visit table with the Time dimension to filter records for the year 2010 and then aggregates the charge measure by Doctor_ID

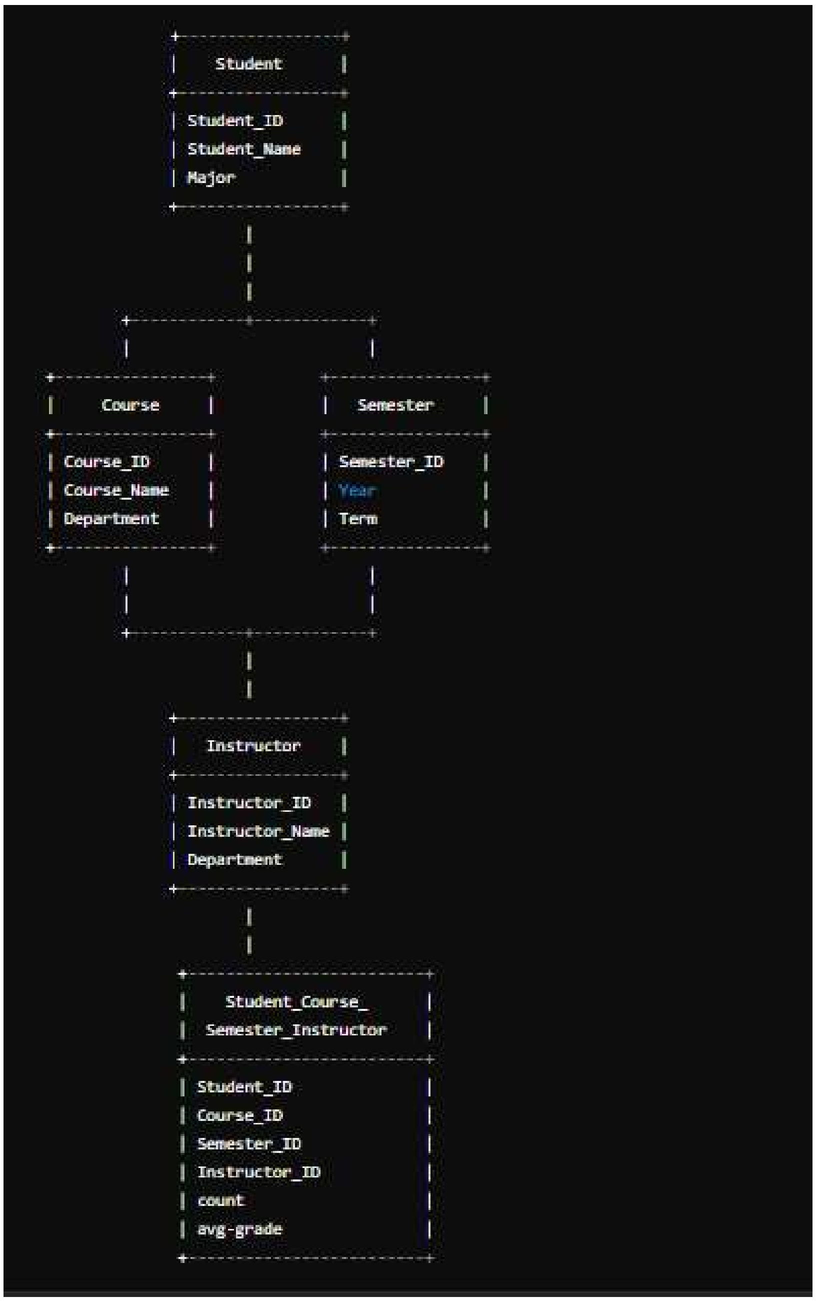
Suppose that a data warehouse for DB-University consists of the four dimensions student, course, semester, and instructor, and two measures count and avg-grade. At the lowest conceptual level (e.g., for a given student, course, semester, and instructor combination), the avg-grade measure stores the actual course grade of the student. At higher conceptual levels, avg-grade stores the average grade for the given combination. i. Draw a snowflake schema diagram for the data warehouse. ii. Starting with the base cuboid [student, course, semester, instructor], what specific OLAP operations (e.g., roll-up from semester to year) should you perform in order to list the average grade of CS courses for each DB-University student.

1. Fact Table

- **Fact Table**: Fact_Student_Course_Semester_Instructor
  - Measures: count, avg-grade
  - Foreign Keys: Student_ID, Course_ID, Semester_ID, Instructor_ID

2. Dimension Tables

- **Student Dimension**
  - Attributes: Student_ID, Student_Name, Major
- **Course Dimension**
  - Attributes: Course_ID, Course_Name, Department
- **Semester Dimension**
  - Attributes: Semester_ID, Year, Term
- **Instructor Dimension**
  - Attributes: Instructor_ID, Instructor_Name, Department



```
                +----------------+
                |    Student     |
                +----------------+
                | Student_ID     |
                | Student_Name   |
                | Major          |
                +----------------+
                        |
                        |
                        |
           +------------+------------+
           |                         |
+---------------+            +---------------+
|    Course     |            |   Semester    |
+---------------+            +---------------+
| Course_ID     |            | Semester_ID   |
| Course_Name   |            | Year          |
| Department    |            | Term          |
+---------------+            +---------------+
           |                         |
           |                         |
           +------------+------------+
                        |
                        |
                +----------------+
                |   Instructor   |
                +----------------+
                | Instructor_ID  |
                | Instructor_Name|
                | Department     |
                +----------------+
                        |
                        |
                 +-----------------------+
                 |    Student_Course_    |
                 |  Semester_Instructor  |
                 +-----------------------+
                 | Student_ID            |
                 | Course_ID             |
                 | Semester_ID           |
                 | Instructor_ID         |
                 | count                 |
                 | avg-grade             |
                 +-----------------------+
```

ii. OLAP Operations

To list the average grade of CS courses for each DB-University student starting from the base cuboid [student, course, semester, instructor], follow these OLAP operations:

1. **Roll-up from course to department**:
   - **Operation**: Aggregate data by course department.
   - **Purpose**: To group grades by course departments, allowing you to focus on the CS department.
2. **Select (Filter)**:
   - **Operation**: Filter the data to include only records where the Department is "CS".
   - **Purpose**: To narrow down the data to only CS courses.
3. **Roll-up from semester to year** (optional):
   - **Operation**: Aggregate data by year if needed for broader analysis.
   - **Purpose**: This step may be skipped if semester-level details are sufficient for your analysis.
4. **Aggregation**:
   - **Operation**: Compute the average grade (avg-grade) for each student in the CS department.
   - **Purpose**: To get the average grade for each student across the CS courses.

OLAP Operations in Sequence:

1. **Start with base cuboid**: [student, course, semester, instructor].
2. **Roll-up** from course to department to aggregate data by department.
3. **Filter** the data to include only those where Department = 'CS'.
4. **(Optional) Roll-up** from semester to year if broader time aggregation is needed.
5. **Aggregation** to compute the average grade (avg-grade) for each student in the CS department.

```
SELECT Student_ID, AVG(avg-grade) AS AverageGrade
FROM Fact_Student_Course_Semester_Instructor
JOIN Course ON Fact_Student_Course_Semester_Instructor.Course_ID = Course.Course_ID
WHERE Course.Department = 'CS'
GROUP BY Student_ID;
```
This query computes the average grade for each student in CS courses by joining the fact table with the course dimension and filtering for the CS department.