



BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI WORK INTEGRATED LEARNING PROGRAMMES

COURSE HANDOUT

Part A: Content Design

Course Title	Data Warehousing
Course No(s)	SS ZG515 / SE* ZG515 / CSI* ZG514/DSE* ZG515/ CC* ZG515
Credit Units	5 (2: Class Room Hours; 2: Students Preparation; 1: Case Studies) <i>(1 credit unit translates to approximately 32 hours)</i>
Course Author	Swarna Chaudhary
Version No	2.0
Date	

Course Description

Corporate decision makers require access to all the organization's data, wherever it is located. To provide comprehensive analysis of the organization, its business, its requirements and any trends, require access to not only the current data in the database but also to historical data. To facilitate this type of analysis, data warehouses have been created to contain data drawn from several sources, maintained by different departments of the organization. This course will involve an in-depth study of various concepts needed to design, develop, and maintain a data warehouse. It will also orient the students towards various approaches used for the deployment of the data warehouses including hybrid and cloud native architectures. It also provides an introduction to end user access tools like OLAP and reporting.

Course Objectives

CO1	Understand the importance of data warehouse and the business intelligence that can be gained with the help of data warehouse
CO2	Understand schema designs, information delivery techniques and architectures that are appropriate for data warehouse
CO3	Understand processes, management, and infrastructure necessary for building data warehouse
CO4	Understand the evolution of data warehouses with the presence of big data and cloud environments.

Teaching methodology

The teaching methodology to be used are:

- a) Lectures
- b) Schema Design Exercises
- c) Exploration of DW trends

Text Books

T1	Ponniah P, " Data Warehousing Fundamentals ", Wiley Student Edition, 2012
T2	Kimball R, " The Data Warehouse Toolkit ", 3e, John Wiley, 2013

Reference Books

R1	Anahory S, & Dennis M, " Data Warehousing in the Real World ", Pearson Education, 2008.
R2	Kimball R, Reeves L, Ross M, & Thornthwaite, W, " The Data Warehouse Lifecycle Toolkit ", John Wiley, 2e, 2012.
R3	Jiawei Han, Micheline Kamber and Jian Pei, " Data Mining: Concepts and Techniques ", Morgan Kaufmann Publishers 2012
R4	Amazon Redshift Database Developer Guide http://docs.aws.amazon.com/redshift - online resource for the DW on the Cloud
R5	Krish Krishnan, " Data Warehousing in the Age of Big Data ", Morgan Kaufmann Publishers 2013
R6	William H Inmon, et al., " DW 2.0 : The Architecture for the Next Generation of Data Warehousing ", Morgan Kaufmann 2012
R7	Cloud Data Warehousing for Dummies by Joe Kraynak, David Baum
R8	Automating the Modern Data Warehouse A Comprehensive Guide for Optimal Data Management by Steve Swoyer

Modular Content Structure

1. Introduction to Database Warehousing
 - 1.1. Evolution of Data Warehousing
 - 1.2. Data Warehouse definitions
 - 1.3. Business need for Data Warehouse
 - 1.4. Comparison of Data Warehouse with other business software
2. Data Warehouse Architecture
 - 2.1. DW architectural components
 - 2.2. Data Mart vs DW vs ODS
 - 2.3. DW architectural types
 - 2.4. Top-down vs. Bottom-up approaches to Data Warehousing
 - 2.5. DW vs. Hadoop
3. Introduction to Dimensional Modelling
 - 3.1. Dimensional Modeling
 - 3.2. ER Modelling vs Dimensional Modelling
 - 3.3. Star, Snowflake, Starflake schemas
 - 3.4. Data Warehouse design steps
 - 3.5. Slowly changing dimensions
 - 3.6. Types of facts, dimensions
 - 3.7. Retail, Inventory Case Study
4. Extraction, Transformation, Load
 - 4.1. ETL overview
 - 4.2. Data Extraction
 - 4.3. Data Transformation
 - 4.4. Data Loading

- 4.5. Data Quality
 - 4.6. ELT, EtLT
- 5. OLAP & Multidimensional Databases (MDDBs)
 - 5.1. Limitations of spreadsheets and SQL
 - 5.2. Major OLAP Features and functions
 - 5.3. Multidimensional databases
 - 5.4. ROLAP, MOLAP, and HOLAP
 - 5.5. OLAP operations using MDDBs
- 6. Query performance enhancement techniques
 - 6.1. Aggregations
 - 6.2. Shrunken, Lost & Collapsed Dimension
 - 6.3. Aggregate Navigator
 - 6.4. Partitioning
 - 6.5. View materialization
 - 6.6. View Maintenance Strategies
 - 6.7. Indexing techniques
 - 6.8. Row vs Column-oriented storage
- 7. Metadata
 - 7.1. Role of metadata in DW
 - 7.2. Types of metadata
 - 7.3. Metadata Design & Implementation
- 8. DW infrastructure
 - 8.1. Capacity Planning
 - 8.2. Security for DW
 - 8.3. Hardware for DW
 - 8.4. Deployment: on-premise, cloud, hybrid
- 9. DW Project Management
 - 9.1. DW development lifecycle
 - 9.2. Agile techniques for DW
- 10. DW on web
 - 10.1. Web-enabling DW
 - 10.2. Web as data source
 - 10.3. DW on cloud
- 11. Cloud Data Warehousing
 - 11.1. Topologies
 - 11.2. Provider Selection
 - 11.3. Configuration, Management
 - 11.4. Migration from on-prem to cloud DW
- 12. Real Time data warehousing (RTDWH)
 - 12.1. Business Need for RTDWH
 - 12.2. Real-time ETL
 - 12.3. Use cases
 - 12.4. Benefits and Challenges
 - 12.5. Big Data Analytics
 - 12.6. Extended RDBMS Architecture
 - 12.7. MapReduce/Hadoop Architecture

Learning Outcomes:

No	Learning Outcomes
LO1	Knowledge of business intelligence that can be gained from data warehouse.
LO2	Knowledge of schema designs, information delivery techniques and architectures for data warehouse.
LO3	Knowledge of processes, management, and infrastructure for building data warehouse.
LO4	Evolution of data warehouse with the presence of big data and cloud environments.

Part B: Contact Session Plan

Academic Term	
Course Title	Data Warehousing
Course No	
Lead Instructor	

Course Contents

Contact Hours(#)	List of Topic Title (from content structure in Part A)	Topic # (from content structure in Part A)	Text/Ref Book/external resource
1	<ul style="list-style-type: none"> ● Introduction to Database Warehousing <ul style="list-style-type: none"> ○ Evolution of Data Warehousing ○ Data Warehouse definitions ○ Business need for Data Warehouse ○ Comparison of Data Warehouse with other business software 	1	T1: Ch-1
2			
3	<ul style="list-style-type: none"> ● Data Warehouse Architecture <ul style="list-style-type: none"> ○ DW architectural components ○ Data Mart vs DW vs ODS ○ DW architectural types ○ Top-down vs. Bottom-up approaches to Data Warehousing ○ DW vs. Hadoop 	2	T1: Ch-2,7 T2: Ch 1
4			
5	<ul style="list-style-type: none"> ● Introduction to Dimensional Modelling <ul style="list-style-type: none"> ○ Dimensional Modelling ○ ER Modelling vs Dimensional Modelling ○ Star, Snowflake, Starflake schemas 	3	T1: Ch-10, 11 T2: Ch-1 to 5
6			
7			

8	<ul style="list-style-type: none"> ○ Data Warehouse Design steps ○ Slowly changing dimensions ○ Types of facts, dimensions ○ Retail, Inventory case study 		
9			
10			
11	<ul style="list-style-type: none"> ● Extraction, Transformation, Load ○ ETL overview ○ Data Extraction ○ Data Transformation ○ Data Loading ○ Data Quality ○ ELT, EtLT 	4	T1: Ch 12, 13
12			
13	<ul style="list-style-type: none"> ● OLAP & Multidimensional Databases (MDDB) ○ Limitations of spreadsheets and SQL ○ Major OLAP Features and functions ○ Multidimensional databases ○ ROLAP, MOLAP, and HOLAP ○ OLAP operations using MDDBs 	5	T1: Ch 15
14			
15	Review and case studies		T2
16			
17	<ul style="list-style-type: none"> ● Query performance enhancement techniques ○ Aggregations ○ Shrunk, Lost & Collapsed Dimension ○ Aggregate Navigator ○ Partitioning ○ View materialization ○ View Maintenance Strategies ○ Indexing techniques ○ Row vs Column-oriented storage 	6	T1: Ch 18
18			
19			
20			
21	<ul style="list-style-type: none"> ● Metadata ○ Metadata management, Standards ○ Role of metadata in DW ○ Types of metadata, implementation practices 	7	T1: Ch 9
22			
23	<ul style="list-style-type: none"> ● DW infrastructure ○ Capacity Planning ○ Security for DW ○ Hardware for DW ○ Deployment: on-premise, cloud, hybrid 	8	T1: Ch 8,19,20 R7: Ch 4
24			
25	<ul style="list-style-type: none"> ● DW Project Management ○ DW development lifecycle ○ Agile techniques for DW 	9	T1: Ch 4
26			
27	<ul style="list-style-type: none"> ● DW on web ○ Web-enabling DW 	10	T1: Ch 16 R4

28	<ul style="list-style-type: none"> ○ Web as data source ○ DW on cloud 		
29	<ul style="list-style-type: none"> ● Cloud Data Warehousing ○ Topologies ○ Provider Selection ○ Configuration, Management ○ Migration from on-prem to cloud DW 		T2, R7
30			
31	<ul style="list-style-type: none"> ● Real Time data warehousing (RTDW) ○ Business Need for RTDWH ○ Real-time ETL ○ Use cases ○ Benefits and Challenges ○ Big Data Analytics ○ Extended RDBMS Architecture ○ MapReduce/Hadoop Architecture ○ Traditional DW vs RTDW 		T2-Ch 20, 21
32			Class Notes

The above contact hours and topics can be adapted for non-specific and specific WILP programs depending on the requirements and class interests.

Evaluation Scheme

Legend: EC = Evaluation Component; AN = After Noon Session; FN = Fore Noon Session

No	Name	Type	Duration	Weight	Day, Date, Session, Time
EC-1	Quiz-I	Online		5%	
	Assignment-I	Take Home		13%	
	Assignment-II	Take Home		12%	
EC-2	Mid-Semester Test	Closed Book	1.5 Hrs	30%	
EC-3	Comprehensive Exam	Open Book	2.5 Hrs	40%	

Important Information

Syllabus for Mid-Semester Test (Closed Book): Topics in Weeks 1-7

Syllabus for Comprehensive Exam (Open Book): All topics given in plan of study

Evaluation Guidelines:

1. EC-1 consists of either two Assignments or three Quizzes. Announcements regarding the same will be made in a timely manner.
2. For Closed Book tests: No books or reference material of any kind will be permitted. Laptops/Mobiles of any kind are not allowed. Exchange of any material is not allowed.
3. For Open Book exams: Use of prescribed and reference text books, in original (not photocopies) is permitted. Class notes/slides as reference material in filed or bound form is permitted. However, loose sheets of paper will not be allowed. Use of calculators is permitted in all exams. Laptops/Mobiles of



BITS Pilani

Pilani Campus

Cloud Data Warehousing

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Course Objective

1. Comprehend the significance of data warehousing in leveraging business intelligence (BI) and its potential for driving informed decision-making.
2. Explore various schema designs, information delivery methods, and suitable architectures tailored for effective data warehousing.
3. Gain insight into the essential processes, management strategies, and infrastructure required to construct a robust and functional data warehouse.
4. Analyze the transformative journey of data warehouses within the context of big data and cloud environments, recognizing their impact and evolving role in contemporary data management.

Text Books

T1	Ponniah P, "Data Warehousing Fundamentals", Wiley Student Edition, 2012
T2	Kimball R, "The Data Warehouse Toolkit", 3e, John Wiley, 2013

Reference Books

R1	Anahory S, & Dennis M, "Data Warehousing in the Real World", Pearson Education, 2008.
R2	Kimball R, Reeves L, Ross M, & Thornthwaite, W, "The Data Warehouse Lifecycle Toolkit", John Wiley, 2e, 2012.
R3	Jiawei Han, Micheline Kamber and Jian Pei, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers 2012
R4	Amazon Redshift Database Developer Guide http://docs.aws.amazon.com/redshift - online resource for the DW on the Cloud
R5	Krish Krishnan, "Data Warehousing in the Age of Big Data", Morgan Kaufmann Publishers 2013
R6	William H Inmon, et al., "DW 2.0 : The Architecture for the Next Generation of Data Warehousing", Morgan Kaufmann 2012
R7	Cloud Data Warehousing for Dummies by Joe Kraynak, David Baum
R8	Automating the Modern Data Warehouse A Comprehensive Guide for Optimal Data Management by Steve Swoy

Module 1

Module 1 – Introduction to Data Warehousing

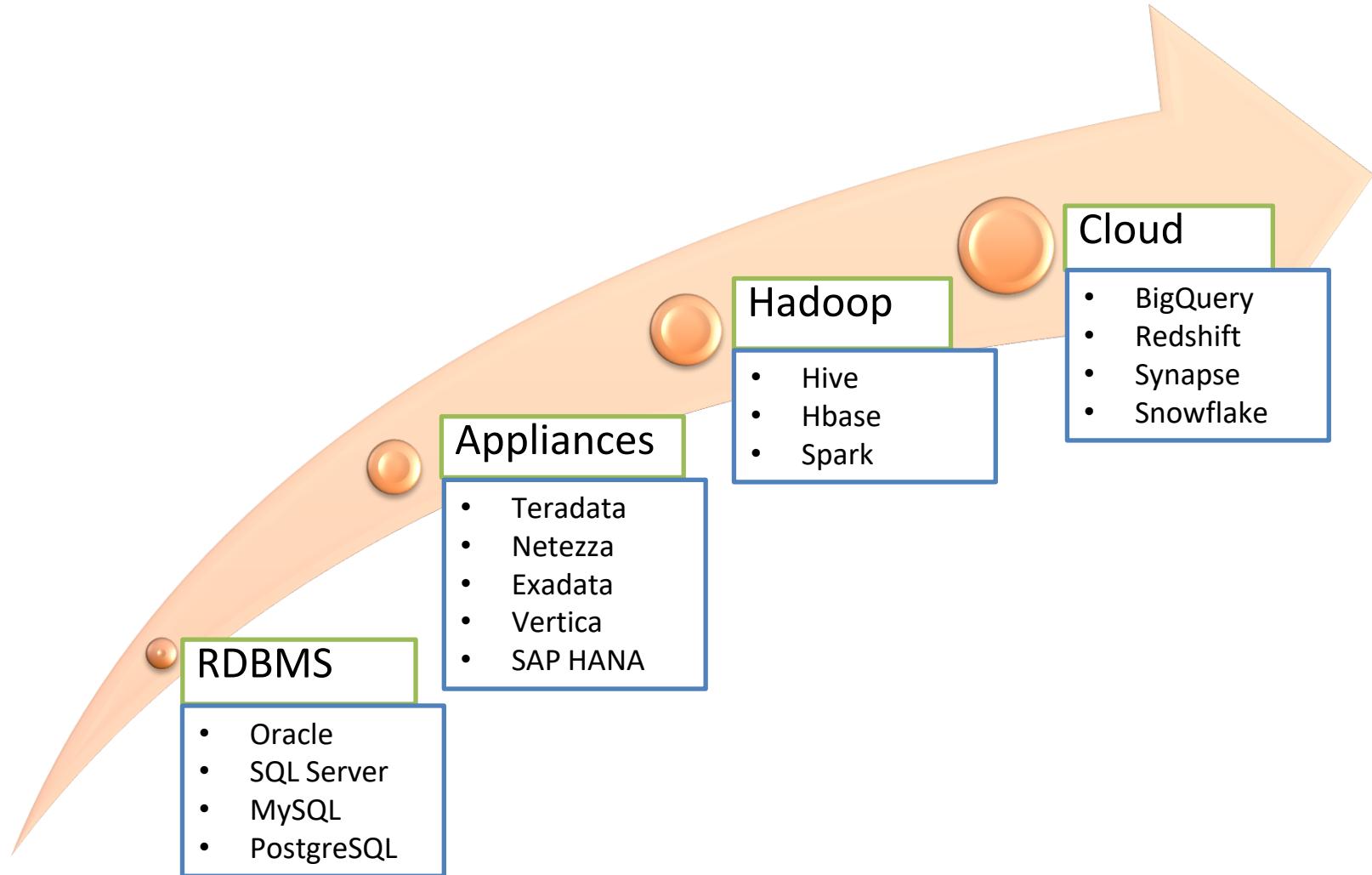
1.1 Evolution of Data Warehousing

1.2 Data Warehousing Definition

1.3 Business Need for Data Warehouse

1.4 Comparison of Data Warehouse with other business software

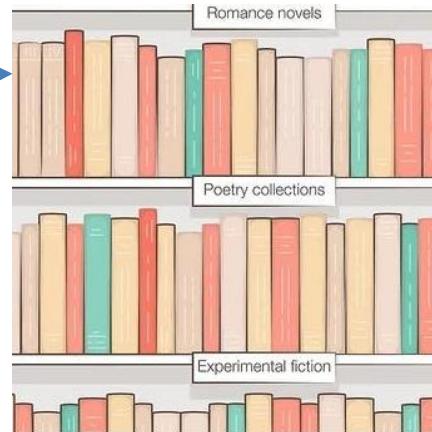
Evolution of Data Warehouse (DW)



What is Data Warehouse (DW)?

Understanding Data Warehouse - Library Analogy

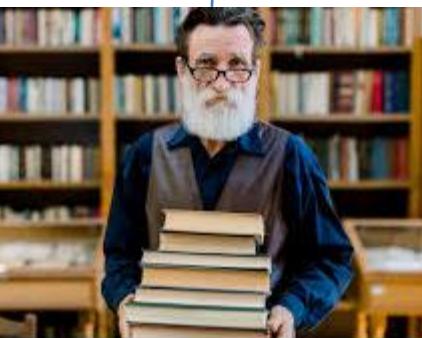
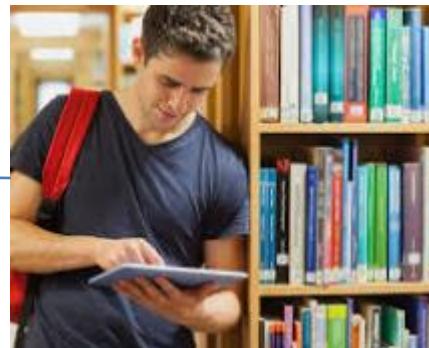
Multiple Type of Books



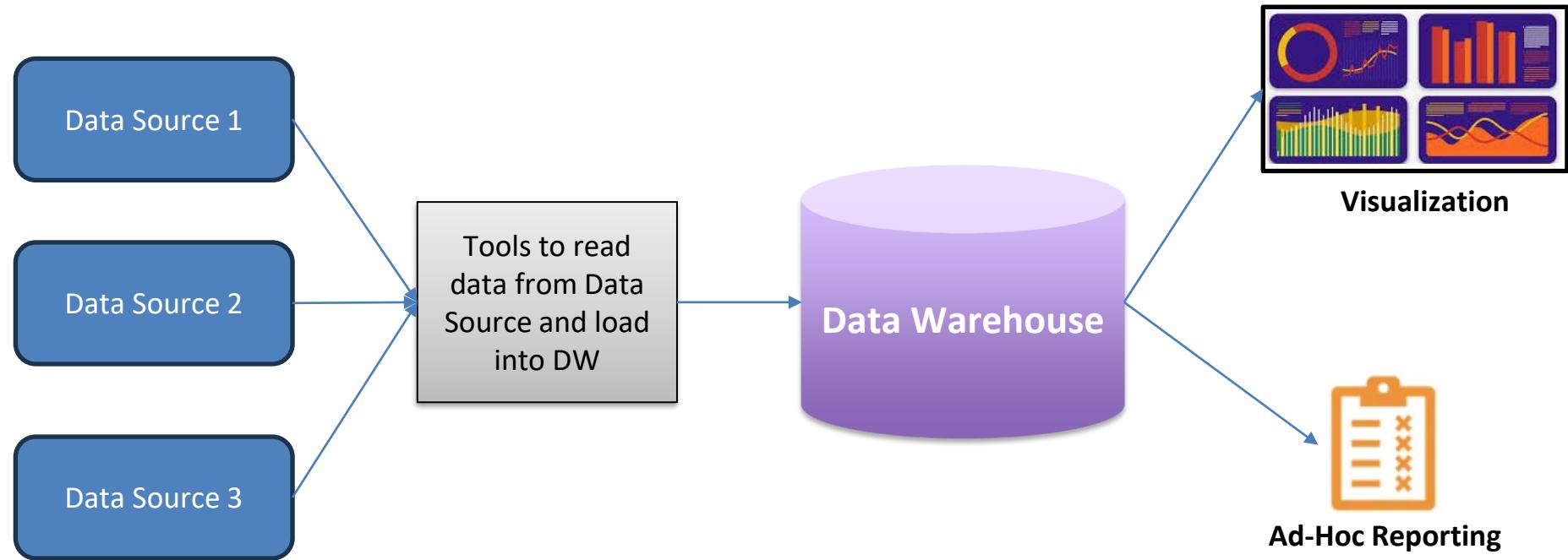
Library



Library User

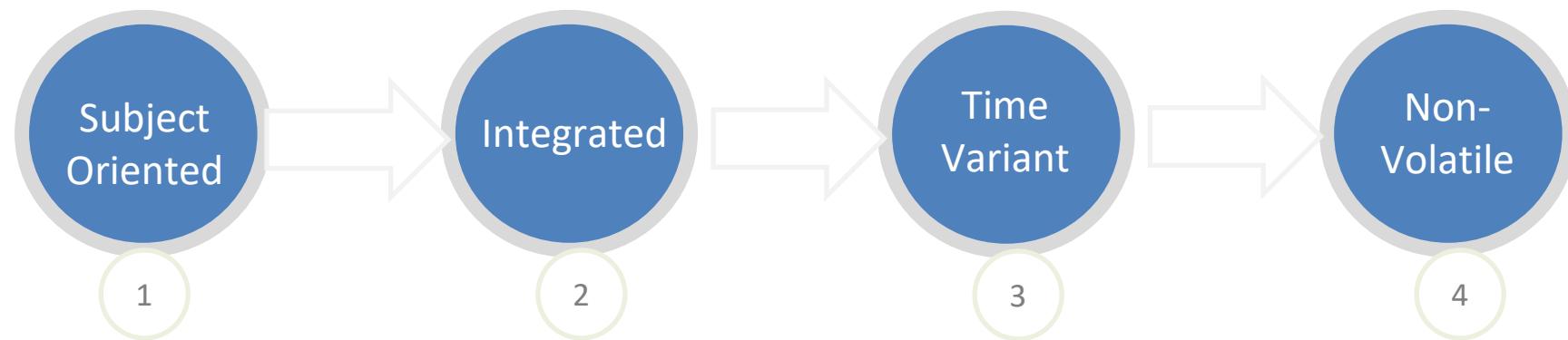


Data Warehouse (DW)



A data warehouse is a system for storing and managing data from multiple sources in a single, central location. It is designed to support complex analytics and decision-making.

Characteristics of Data Warehouse (DW)



Characteristics of Data Warehouse (DW)

Subject-Oriented

A data warehouse is organized around specific subjects or areas of interest. These subjects or areas of interest can be sales, customers, or products. This subject orientation allows data to be organized. It also allows data to be analyzed in a relevant way for business users.

Integrated

A data warehouse integrates data from various sources. These sources may involve a cloud, relational databases, structured and semi-structured data, etc. The sources are integrated in a sequential manner. They are consistent, relatable, and ideally certifiable. They provide a business with confidence in the data's quality.

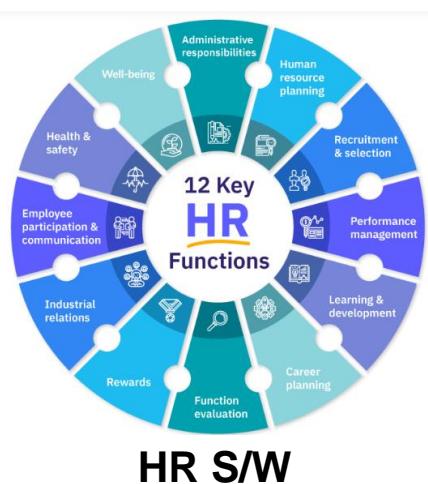
Time-Variant

An organization keeps historical data over time in a data warehouse. This makes it possible to spot patterns and trends. Based on what has previously occurred, it helps to make better decisions. Organizations can spot trends and make better decisions for the future by examining patterns throughout time.

Non-Volatile

A data warehouse is non-volatile. It means that once data is loaded into the warehouse, it cannot be modified or deleted. This helps to ensure the accuracy and consistency of the data. It maintains a historical record of changes over time.

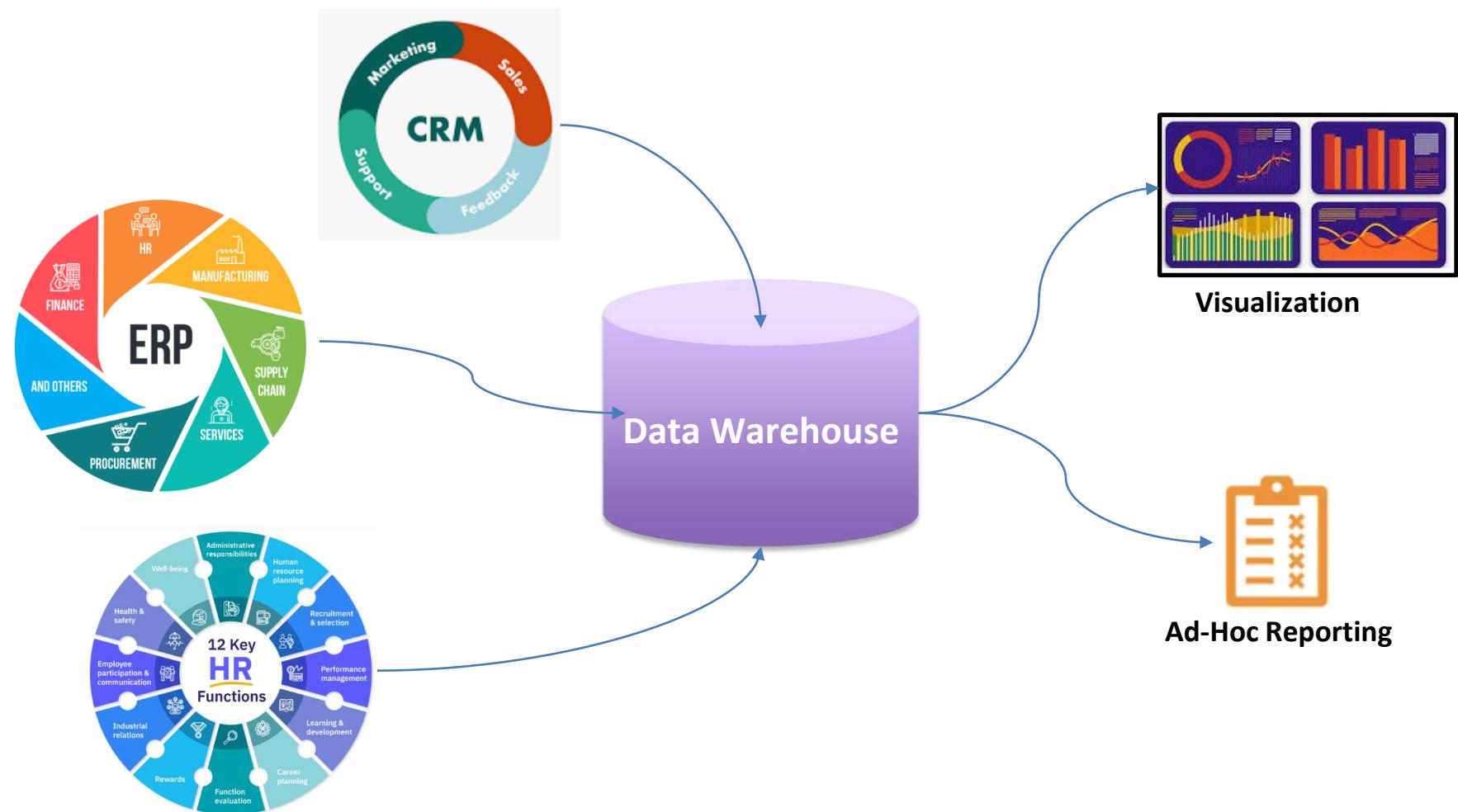
Business need of DW – A Real Life Example



**Prime Tyre
Manufacturing
(A Fictitious Company)**



Business need of DW



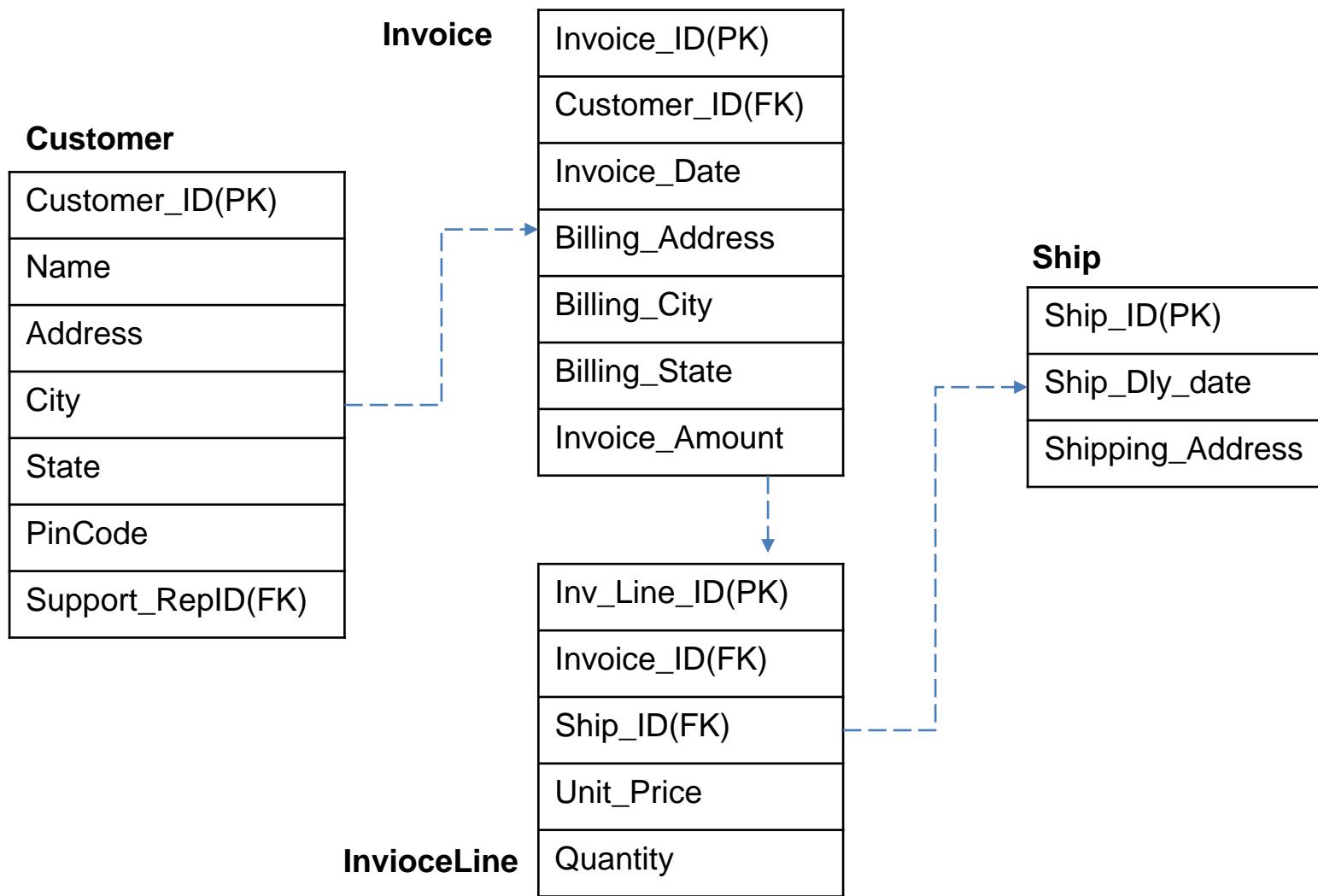
Few Terminologies & Comparisons

Data Modeling

Definition - Data modeling is the process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures.

Purpose: The goal of data modeling to illustrate the types of data used and stored within the system, the relationships among these data types, the ways the data can be grouped and organized and its formats and attributes..

Data Modeling – An Example



Data Mining

Definition - Data mining is the process of using statistical analysis and machine learning to discover hidden patterns, correlations, and anomalies within large datasets.

Purpose: This information can aid you in decision-making, predictive modeling, and understanding complex phenomena.

Data Mining - Process



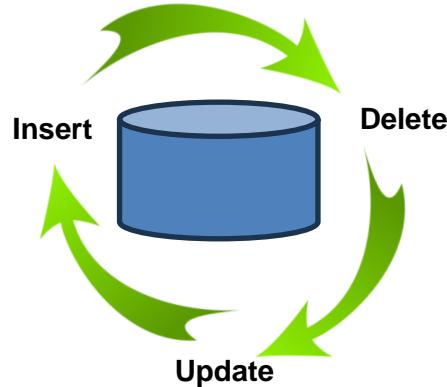
Data Mining - Process

- **Step 1: Collection** – First data is collected, organized, and filled into a data warehouse. The data is stored and managed either in the cloud or in-house servers.
- **Step 2: Understanding** – In this step, data scientists and business analysts examine the properties of the data and conduct an in-depth analysis from the context of a particular problem statement as defined by the company. This is addressed using querying, visualization, and reporting.
- **Step 3: Preparation** – Once the data sources of the available data are confirmed, the data is cleared, constructed, and formatted into the required form. In this process, additional data can also be explored at a greater depth, which is well informed by the insights and uncovered in the previous stage.
- **Step 4: Modeling** – In this stage, for the prepared dataset, modeling techniques are selected. A data model is just like a diagram that reflects and describes the relationships between different types of information that are stored in the database. Common techniques include decision trees, regression, clustering, classification, association rule mining, and neural networks.
- **Step 5: Evaluation** – In the context of the business objectives, the model results are evaluated. In this phase, due to new patterns that are discovered in the model results or other factors, new business requirements may be raised.

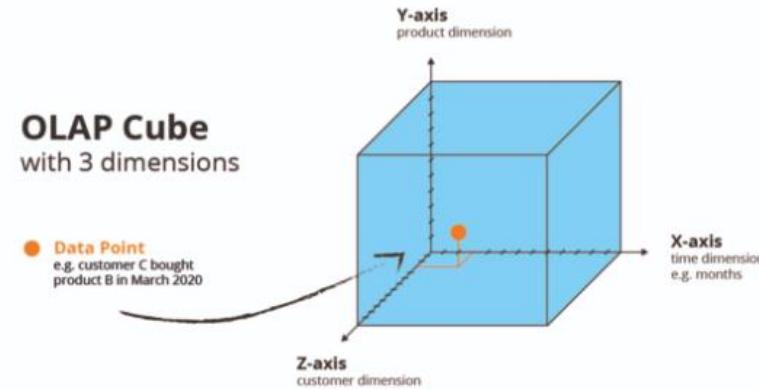
Data Warehouse Vs. Data Mining

Aspect	Data Warehousing	Data Mining
Purpose	To store and manage data for analysis, reporting, and decision making	To discover patterns, trends, and insights within data that may not be immediately apparent
Function	Storage and organization of structured data	Analysis and extraction of hidden patterns from data
Objective	Provides a unified view of an organization's data for analysis	Discovers hidden patterns or relationships within the data
Techniques	Structuring, storage, and optimization of data	Statistical analysis, machine learning, AI algorithms
Usage	Supports reporting, querying, and analytics	Identifies trends, predictions, and decision support
Benefit	Provides a centralized data repository for business intelligence	Uncovers insights for making informed decisions
Examples	Data warehouses like Oracle, SQL Server	Techniques like clustering, regression, classification

OLTP Vs. OLAP



OLTP



OLAP

Let's say your data set has figures for various products entered, including customers and time of purchase. In cube form, for example, the X-axis in the coordinate system represents the time dimension, the Z-axis represents the customer dimension, and the Y-axis represents your product dimension. The size of the cube is therefore determined by the existing data and is therefore dynamic from the ground up. Each combination of dimension data is assigned values, each of which represents a coordinate in the cube.

Here's an example: In March 2020, customer C purchased product B. The result of this relationship are quantifiable values, such as the sales revenue or the amount purchased, which are stored, so to speak, at the coordinate that can now be exactly determined (data point).

OLTP Vs. OLAP

OLAP	Operational Database(OLTP)(Database)
It involves historical processing of information.	It involves day-to-day processing.
OLAP systems are used by knowledge workers such as executives, managers, and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
It is used to analyze the business.	It is used to run the business.
It focuses on Information out.	It focuses on Data in.
It is based on Star Schema, Snowflake Schema, and Fact Constellation Schema.	It is based on Entity Relationship Model.
It contains historical data.	It contains current data.
It provides summarized and consolidated data.	It provides primitive and highly detailed data.
It provides summarized and multidimensional view of data.	It provides detailed and flat relational view of data.
The number of users is in hundreds.	The number of users is in thousands.
The number of records accessed is in millions.	The number of records accessed is in tens.

Integrating Heterogenous Databases

Heterogenous Databases

Heterogeneous databases are databases that consist of data from multiple, dissimilar sources. These sources may include different types of databases, such as relational databases, NoSQL databases, and flat files.

Integrating Heterogenous

- Integration of heterogeneous databases in data warehousing refers to the process of combining data from multiple, disparate databases into a central repository, known as a data warehouse. This process involves extracting data from different sources, such as relational databases, NoSQL databases, and flat files, and then transforming, cleaning, and loading the data into the data warehouse. This is commonly called ETL or ELT in Data Engineering World.
- Refer to “Business Need of DW section which talks about the same thing at a high level.

Approach for Integration of Heterogenous Databases

Approaches for Integration of Heterogeneous Databases

There are two different approaches to integrating heterogeneous databases

- 1. Query Driven Approach:** This is the traditional approach to integrate heterogeneous databases. The integration of data occurs dynamically at the time of the query execution rather than through a pre-defined, static integration process.
- 2. Update Driven Approach:** The information from multiple heterogeneous sources are integrated in advance and are stored in a warehouse. This information is available for direct querying and analysis.

Query Driven Vs Update Driven Approach

	Query Driven	Update Driven
Approach	<ul style="list-style-type: none"> When a query is issued on a client side, a metadata dictionary translates the query into an appropriate form for individual heterogeneous sites involved. Now these queries are mapped and sent to the local query processor. The results from heterogeneous sites are integrated into a global answer set 	<ul style="list-style-type: none"> Information from multiple heterogeneous sources are integrated in advance and are stored in a warehouse. This information is available for direct querying and analysis.

Query Driven Vs Update Driven Approach – Pros & Cons

	Query Driven	Update Driven
Advantages	<ul style="list-style-type: none"> • Data does not need to be stored in a dedicated DW 	<ul style="list-style-type: none"> • This approach provide high performance. • The data is copied, processed, integrated, annotated, summarized and restructured in semantic data store in advance. • Query processing does not require an interface to process data at local sources.
Disadvantages	<ul style="list-style-type: none"> • Query-driven approach needs complex integration and filtering processes. • This approach is very inefficient. • It is very expensive for frequent queries. • This approach is also very expensive for queries that require aggregations. 	<ul style="list-style-type: none"> • Need to have a DW with ample storage to store the data.

Query Driven Vs Update Driven Approach – Summary

Characteristic	Query-Driven Approach	Update-Driven Approach
Data Integration Process	Data is integrated at the time of query execution.	Data is integrated in advance and stored in the warehouse.
Data Freshness	Real-time or near real-time data availability.	Periodic updates with a potential latency between updates.
Query Performance	Can be impacted by the need to integrate data in real-time.	Generally faster querying as data is preintegrated.
Complexity of Implementation	Complex, especially for real-time integration requirements.	Easier to implement and maintain as integration is scheduled.
Consistency and Standardization	May result in varied data formats and structures.	Provides a consistent and standardized view of integrated data.
Resource Utilization	Resources are used at the time of query execution.	Resources are used during scheduled update processes.
Use Case Suitability	Suitable for scenarios requiring real-time data access.	Suitable for scenarios where periodic updates are acceptable.
Examples	Real-time analytics, dashboards, operational reporting.	Data warehousing with nightly or weekly batch update

Knowledge Check

Q1. Which of the following is a primary goal of a Data Warehouse?

- A. Real-time data processing
- B. Storing raw data without transformation
- C. Supporting online transaction processing
- D. Providing a centralized and unified view of data for analysis

Answer: D. Providing a centralized and unified view of data for analysis

Q2. Which of the following is NOT a characteristic of a Data Warehouse?

- A. Subject-oriented
- B. Integrated
- C. Online Transaction Processing (OLTP)
- D. Time-variant

Answer: C. Online Transaction Processing (OLTP)

Knowledge Check

Q3. What is the role of metadata in a Data Warehouse?

- A. Storing primary data
- B. Managing data security
- C. Providing information about data characteristics and relationships
- D. Executing data transformations

Answer: C. Providing information about data characteristics and relationship

Q4. How does a Data Warehouse contribute to data consistency and accuracy?

- A. By storing raw data without transformation
- B. By providing real-time data updates
- C. By integrating data from various sources into a unified view
- D. By limiting access to historical data

Answer: C. By integrating data from various sources into a unified view

Knowledge Check

Q6. Why is time-variant data important in a Data Warehouse?

- A. Time-variant data is not relevant in Data Warehousing.
- B. Time-variant data allows for the analysis of historical trends and changes over time.
- C. Time-variant data focuses on real-time data processing.
- D. Time-variant data increases data complexity.

Answer: B. Time-variant data allows for the analysis of historical trends and changes over time.

Q7. What is the significance of business intelligence tools in the context of Data Warehousing?

- A. Business intelligence tools are not applicable to Data Warehousing.
- B. Business intelligence tools help in real-time data processing.
- C. Business intelligence tools enable the analysis and visualization of data stored in the Data Warehouse.
- D. Business intelligence tools are only used for data extraction.

Answer: C. Business intelligence tools enable the analysis and visualization of data stored in the Data Warehouse



BITS Pilani

Pilani Campus

Data Warehouse - Architecture

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Module 2

Module 2 – Data Warehouse Architecture

2.1 Data Mart Vs Data Warehouse Vs Operational Data Store

2.2 Data Warehousing Architecture Types

2.3 Top-Down Vs Bottom-up approach to Data Warehousing

2.4 Snowflake Cloud Data Warehouse - Architecture

Types of Data Warehouse



Enterprise Data Warehouse (EDW)



Operational Data Store (ODS)



Data Mart

Recap – DW Vs. ODS Vs. Data Mart

Criteria	Enterprise Data Warehouse (EDW)	Data Mart	Operational Data Store (ODS)
Scope	Organization-wide	Department or business function-specific	Real-time or near-real-time operational data
Purpose	Comprehensive analysis across the enterprise	Focused analysis for specific business area	Real-time transactional processing
Data Size	Large volumes of data	Smaller subset of data from EDW	Current snapshot of operational data
Integration Level	High integration of data from various sources	Integration focused on specific business area	Real-time integration with operational systems
Granularity	Fine-grained data for enterprise-level insights	Can be fine-grained or summarized based on departmental needs	Real-time, detailed transactional data
Users	Broad range of users across the organization	Specific departmental or team users	Users requiring real-time operational data
Data Latency	Generally batch-oriented; may have longer update cycles	Can have shorter update cycles based on specific needs	Near-real-time or real-time updates
Example	An EDW for a multinational corporation integrating data from finance, sales, HR, etc.	A sales data mart for a retail company, focused on sales-related data	An ODS for a telecommunications company tracking real-time network performance.

Data Warehouse Architecture

Data warehouses and their architectures vary depending upon the elements of an organization's situation.

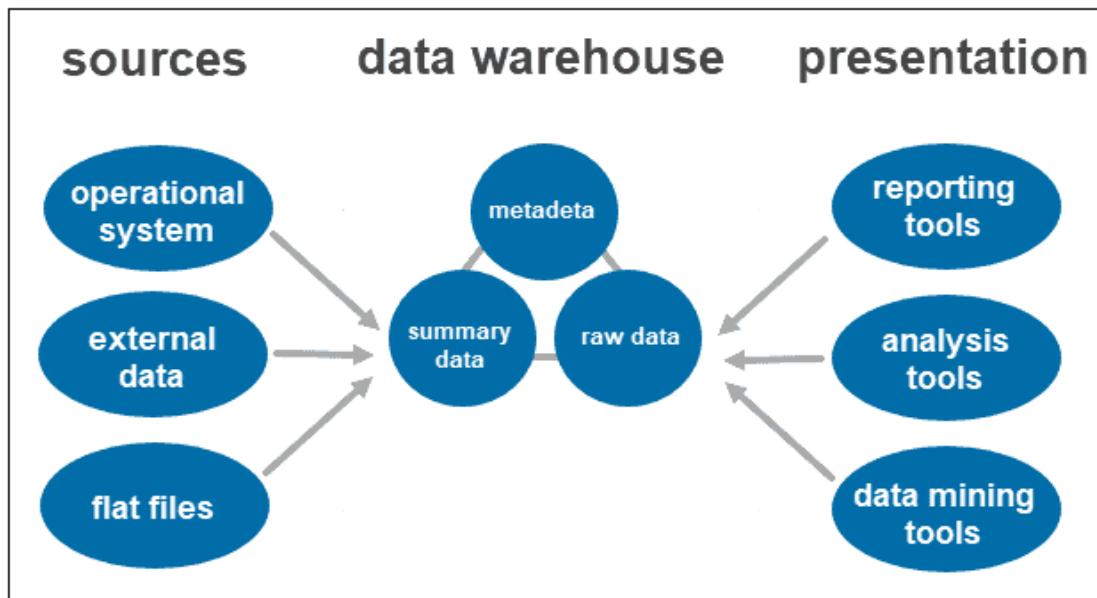
Three common architectures are:

- Data Warehouse Architecture: Single Tier
- Data Warehouse Architecture: Two Tier
- Data Warehouse Architecture: Three Tier

Data Warehouse Architecture – Single Tier

Operational System: An **operational system** refer to a **system** that is used to process the day-to-day transactions of an organization. Operational systems are the source systems that provide raw data to a data warehouse.

Flat Files: A Flat file system is a system of files in which transactional data is stored, and every file in the system must have a different name.



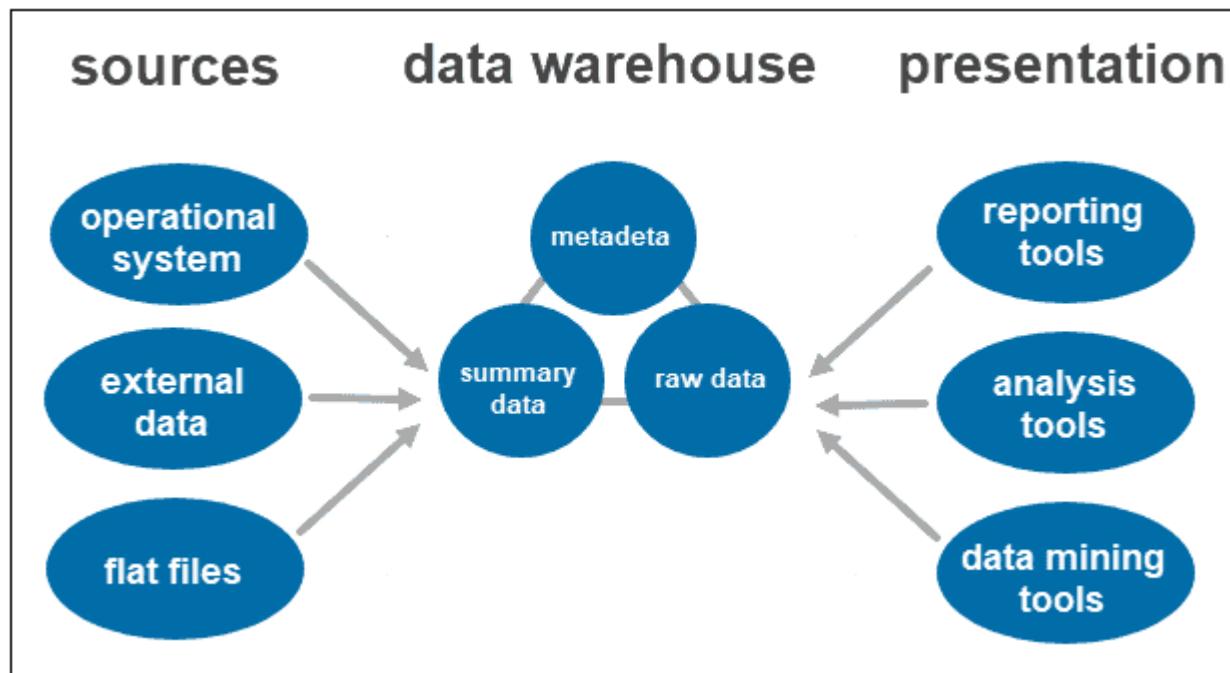
This warehouse architecture has a single layer – the source layer – and aims to reduce the volume of data.

This data warehouse architecture has no staging area or data marts and is not implemented in real-time systems.

This architecture type works best for processing organization operational data

Data Warehouse Architecture – Two Tier

A two-tiered architecture format separates the business layer from the analytical area, thereby giving the warehouse more control and efficiency over its processes. The two-tiered architecture contains a source layer and data warehouse layer and follows a 2-layer data flow process: 1. ETL 2. Data Warehouse

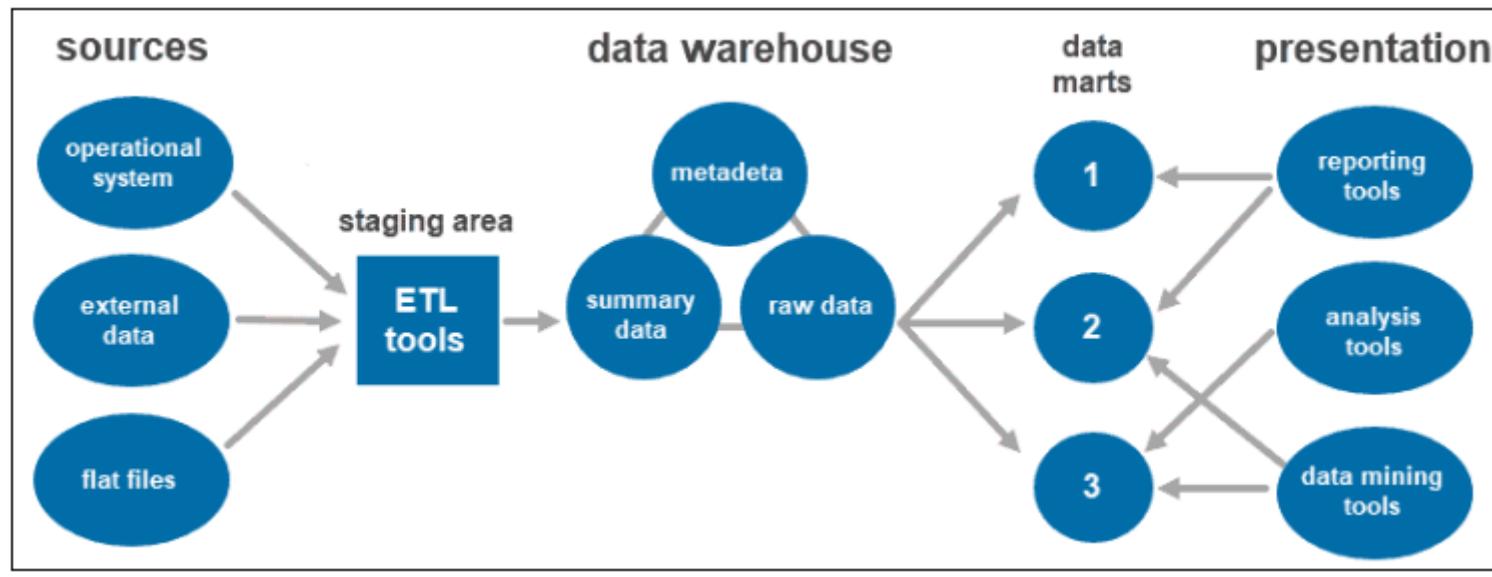


Data Warehouse Architecture – Three Tier

This architecture has three layers: the source, ETL, and data warehouse layer w/ a layer of data which is consumable.

The reconciled layer in this architecture sits between the source and data warehouse layer and acts as a standard reference for an enterprise data model.

However, although this layer introduces better data quality for warehouses, the additional storage space incurs extra costs.



Data Warehouse – ETL

Aspect	Single-Layer Architecture	Two-Layer Architecture	Three-Layer Architecture
Number of Layers	1	2	3
Data Storage	Integrated into a single database	Separated from processing layer	Separated from processing and presentation layers
Data Processing	Combined with data storage in a single layer	Dedicated processing layer	Separate processing layer
Data Presentation	Combined with data storage in a single layer	Integrated with data processing layer	Separate presentation layer
Scalability	Limited scalability due to the single layer	Improved scalability compared to single-layer	Modular design supports better scalability
Performance	May face performance issues as data grows	Can provide better performance optimization	Balanced performance due to modular design
Flexibility	Less flexible to changes in data sources	Moderate flexibility with separation of layers	More flexible and adaptable to changes
Complexity	Simple and easy to understand	Moderate complexity with a separation of layers	More complex due to the presence of three layers
Cost-Effectiveness	More cost-effective for smaller-scale projects	Moderate cost, can be higher than single-layer	Potentially higher implementation costs
Security	Limited control over security measures	Can have better control over security measures	More granular control over security measure

ETL

- ETL, which stands for “extract, transform, load,” are the three processes that move data from various sources to a unified repository—typically a Data Warehouse. It enables data analysis to provide actionable business information, effectively preparing data for analysis and business intelligence processes.
- As data engineers are experts at making data ready for consumption by working with multiple systems and tools, data engineering encompasses ETL.
- Data engineering involves ingesting, transforming, delivering, and sharing data for analysis.
- These fundamental tasks are completed via data pipelines that automate the process in a repeatable way.
- A data pipeline is a set of data-processing elements that move data from source to destination, and often from one format (raw) to another (analytics-ready).

Purpose of ETL

- **Purpose:** ETL allows businesses to consolidate data from multiple databases and other sources into a single repository with data that has been properly formatted and qualified in preparation for analysis. This unified data repository allows for simplified access for analysis and additional processing. It also provides a single source of truth, ensuring that all enterprise data is consistent and up-to-date.

Extraction (E)

- **Extraction**, in which raw data is pulled from a source or multiple sources. Data could come from transactional applications, such as customer relationship management (CRM) data from Salesforce or enterprise resource planning (ERP) data from SAP, or Internet of Things (IoT) sensors that gather readings from a production line or factory floor operation. **For example.** To create a data warehouse, extraction typically involves combining data from these various sources into a single data set and then validating the data with invalid data flagged or removed. Extracted data may be several formats, such as relational databases, XML, JSON, and others.
- **Advantage:**
 - ✓ Isolates the operational systems from the analytical systems, preventing impact on performance.
 - ✓ Enables the extraction of only relevant data, optimizing resources and reducing unnecessary load on the data warehouse.

Transform (T)

- **Transformation**, in which data is updated to match the needs of an organization and the requirements of its data storage solution. Transformation can involve standardizing (converting all data types to the same format), cleansing (resolving inconsistencies and inaccuracies), mapping (combining data elements from two or more data models), augmenting (pulling in data from other sources), and others. During this process, rules and functions are applied, and data cleansed to prevent including bad or non-matching data to the destination repository. Rules that could be applied include loading only specific columns, deduplicating, and merging, among others.
- **Advantage:**
 - ✓ Ensures data consistency and quality by applying business rules and validation checks.
 - ✓ Allows for the integration of data from multiple sources into a unified and standardized format.

Loading (L)

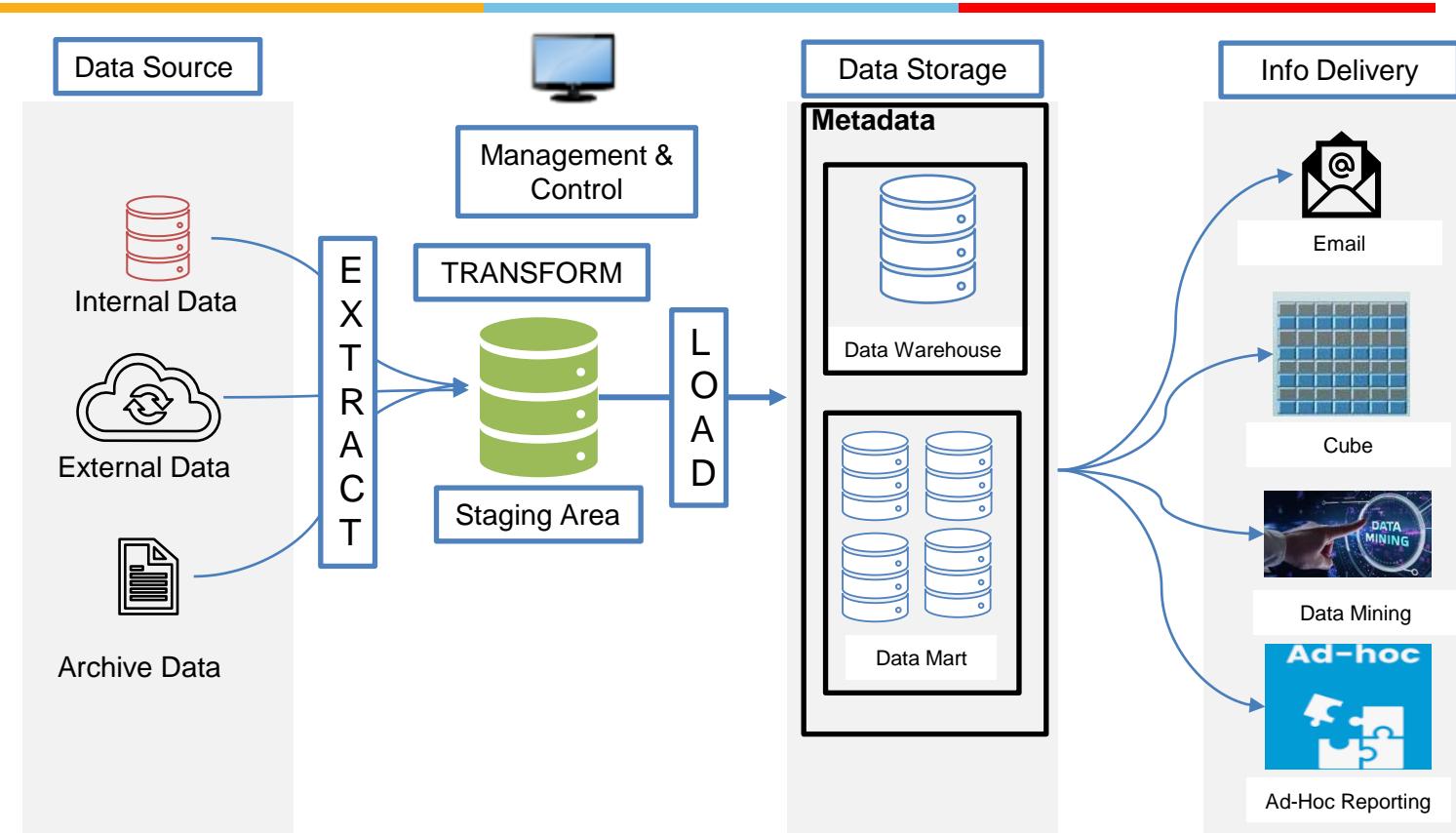
- **Loading**, in which data is delivered and secured for sharing, making business-ready data available to other users and departments, both within the organization and externally. This process may include overwriting the destination's existing data.
- **Advantage:**
 - ✓ Populates the data warehouse with organized, high-quality data for reporting and analysis.
 - ✓ Supports incremental loading, allowing only new or changed data to be loaded, reducing processing time.

ETL Vs ELT

ELT (Extract, Load & Transform) is a variation in which:

- Data is extracted, first loaded and then transformed.
- This sequence allows businesses to preload raw data to a place where it can be modified.
- ELT is more typical for consolidating data in a data warehouse, as cloud-based data warehouse solutions are capable of scalable processing.
- Extract, transform, load is especially conducive to advanced analytics. For example, data scientists commonly load data into a data lake and then combine it with another data source or use it to train predictive models. Maintaining the data in a raw (or less processed) state allows data scientists to keep their options open. This approach is quicker as it leverages the power of modern data processing engines and cuts down on unnecessary data movement.

Data Pipeline - A Complete Picture



Popular ETL Tools: Fivetran, Matillion, AWS Glue, Azure Data Factory, Google Dataflow, Airbyte (open source), Apache Nifi (Open source) AbInitio etc.

Data Pipeline - A Complete Picture

- **Source Data Component:** Source data coming into the data warehouses may be grouped into four broad categories:

Example: Transactional databases, CRM systems, ERP systems, external data feeds.

Production Data: This type of data comes from the different operating systems of the enterprise.

Internal Data: In each organization, the client keeps their "private" spreadsheets, reports, customer profiles, and sometimes even department databases. This is the internal data, part of which could be useful in a data warehouse.

Archived Data: Operational systems are mainly intended to run the current business. In every operational system, we periodically take the old data and store it in achieved files.

External Data: Most executives depend on information from external sources for a large percentage of the information they use. They use statistics associating to their industry produced by the external department.

Data Pipeline - A Complete Picture

• Data Staging Component

- After we have been extracted data from various operational systems and external sources, we have to prepare the files for storing in the data warehouse.
- The extracted data coming from several different sources need to be **changed, converted, and made ready in a format** that is relevant to be saved for querying and analysis.

- **Data Storage Components:** Data storage for the data warehousing is a split repository. The data repositories for the operational systems generally include only the current data. Also, these data repositories include the data structured in highly normalized for fast and efficient processing

- **Information Delivery Component:** The information delivery element is used to enable the process of subscribing for data warehouse files and having it transferred to one or more destinations according to some customer-specified scheduling algorithm.

- **Management and Control Component:**

- The management and control elements coordinate the services and functions within the data warehouse.
- Control the data transformation and the data transfer into the data warehouse storage.
- Moderates the data delivery to the clients.
- Authorizes data to be correctly saved in the repositories.
- It monitors the movement of information into the staging method and from there into the data warehouses storage itself.

Data Pipeline - A Complete Picture

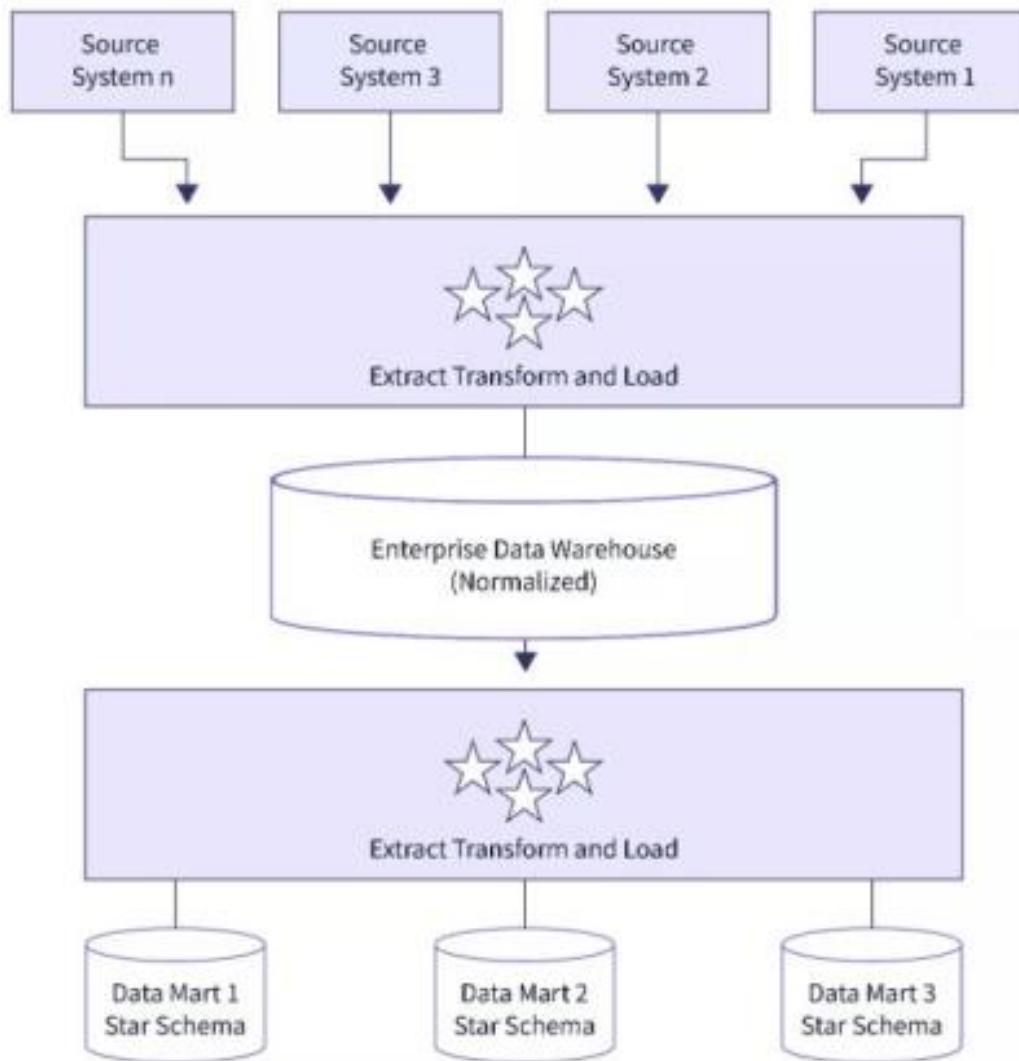
The following are the functions of data warehouse tools and utilities –

- **Data Extraction** - Involves gathering data from multiple heterogeneous sources.
- **Data Cleansing** - Involves finding and correcting the errors in data.
- **Data Transformation** - Involves converting the data from legacy format to warehouse format.
- **Data Loading** - Involves sorting, summarizing, consolidating, checking integrity, and building indices and partitions.
- **Refreshing** - Involves updating from data sources to warehouse.

Approach to Data Warehouse

- There can be two approaches in Data Warehouse:
 - Top-Down Approach
 - Bottom-Up Approach

Top-Down Approach



Top-Down Approach

Characteristic

- This approach was coined by Inmon, and data warehouse in this approach acts as a central information repository for the complete enterprise, and then the data marts are created from it after the complete data warehouse has been set up.
- A global data model is designed to cater to the common information needs of the organization.
- Data Marts: Data marts are then derived from the global data warehouse to address the specific needs of individual departments.

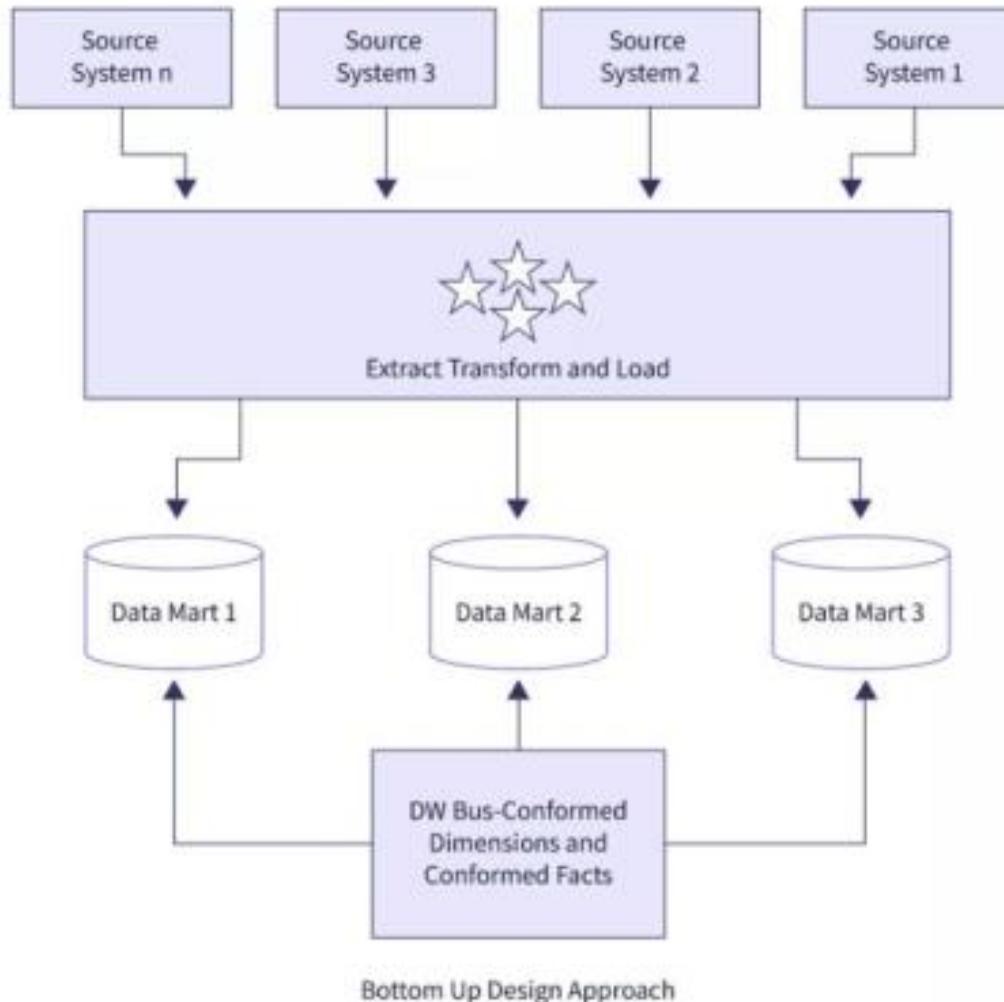
Advantages

- Consistency and Integration: Promotes consistency and integration across the entire organization.
- Centralized Control: Ensures centralized control over data definitions, standards, and security.
- Holistic Perspective: Provides a holistic view of the organization's data, facilitating enterprise-wide reporting and analysis.

Top-Down Approach - Challenges

- Longer Implementation Timelines: May have longer implementation timelines due to the comprehensive nature of designing a centralized data warehouse.
- Resistance from Departments: May face resistance from individual departments that prefer autonomy in managing their data.

Bottom-Up Approach



Bottom-Up Approach

Characteristic

- This approach was coined by Kimball as it can be defined as data after extraction from the source is cleansed and transformed by the staging area, after which the data is sent to the data marts of each theme/subject, and then it is loaded up in the data warehouse.
- In a bottom-up approach, the focus is on creating smaller data marts to address the specific needs of individual departments or business units.
- Over time, these departmental data marts may be integrated to form a larger, enterprise-wide data warehouse.

Advantages

- **Faster Implementation:** Allows for faster implementation as it addresses the immediate needs of individual departments.
- **Departmental Autonomy:** Departments have greater control over their data and can implement solutions more quickly.

Bottom-Up Approach - Challenges

- **Potential for Data Redundancy:** May lead to data redundancy and inconsistency across different data marts.
- **Integration Challenges:** Integration challenges may arise when trying to combine data marts into an enterprise-wide data warehouse.

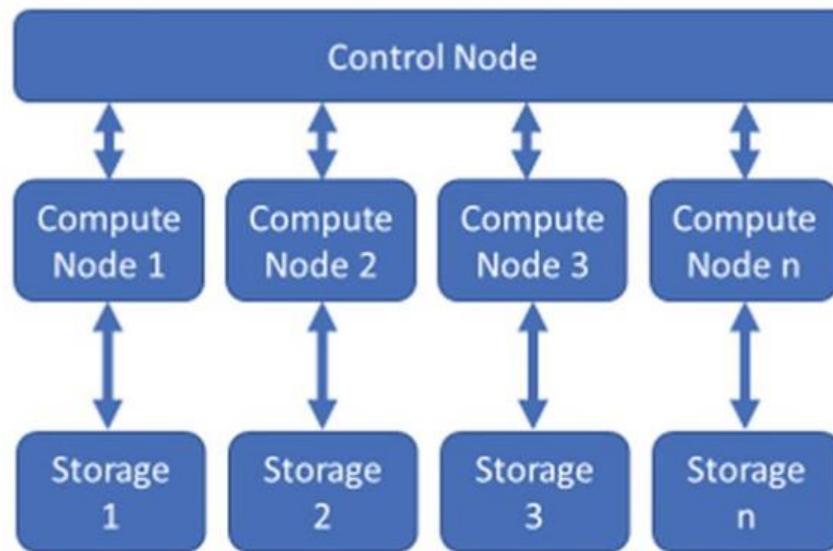
What is MPP System?

- MPP stands for Massive Parallel Processing

In order to understand popular data warehouses like Snowflake, you first need to understand their underlying architecture and the core principles upon which they are built. Massively Parallel Processing (or MPP for short) is this underlying architecture. Here, we'll dive into what an MPP Database is, how it works, and the strengths and weaknesses of Massively Parallel Processing.

MPP Architecture

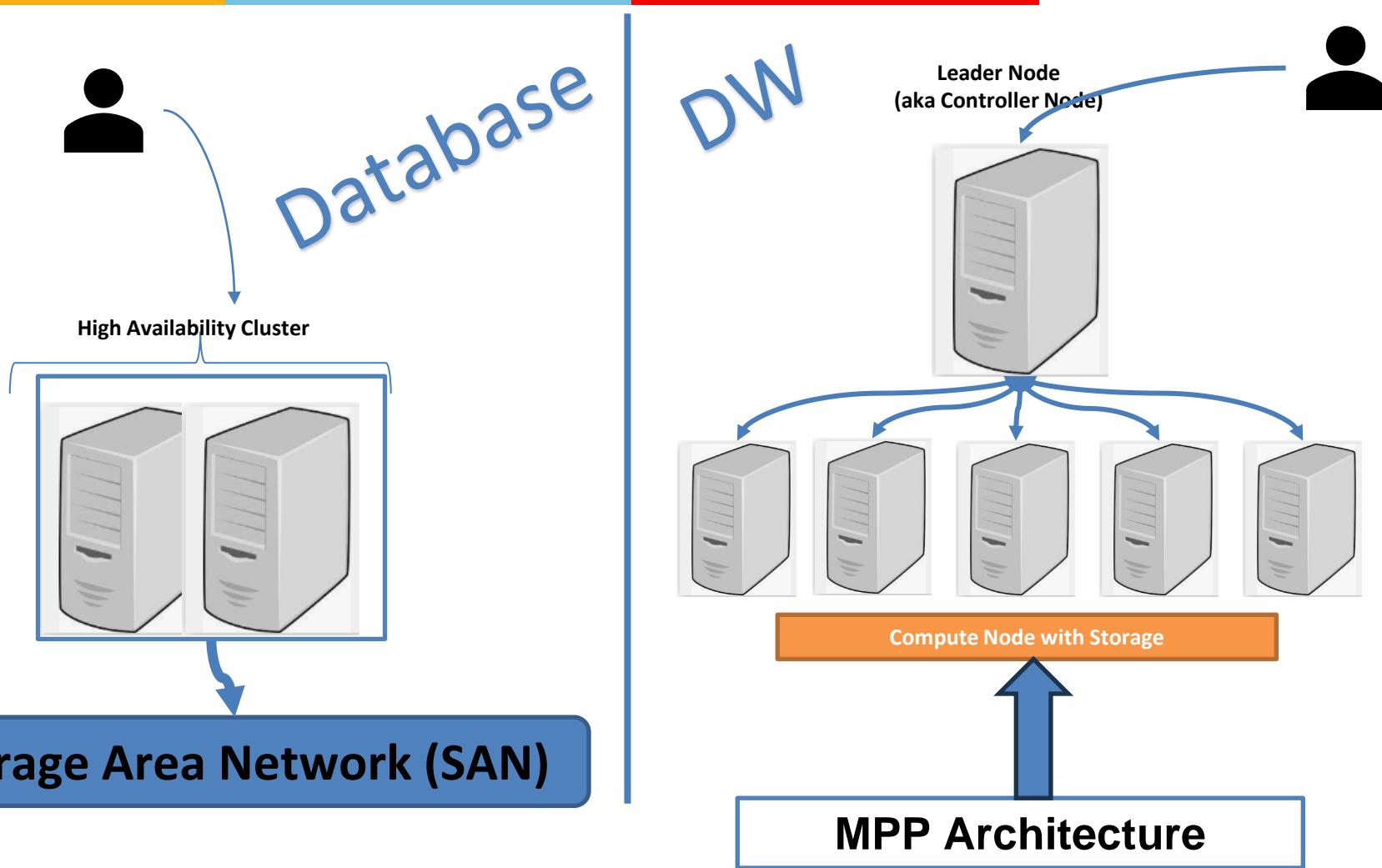
Massively Parallel Processing (MPP)



MPP Architecture

- An MPP database is a type of database or data warehouse where the data and processing power are split up among several different nodes (servers), with one leader node and one or many compute nodes.
- In MPP, the leader (you) would be called the **leader node** - you're the telling all the other people what to do and sorting the final tally.
- The library employees, your helpers, would be called **compute nodes** - they're dealing with all the data, running the queries and counting up the words. MPP databases can *scale horizontally* by adding more compute resources (nodes), rather than having to worry about upgrading to more and more expensive individual servers (scaling vertically).
- Adding more nodes to the cluster allows the data and processing to be spread across more machines, which means the query will be completed sooner

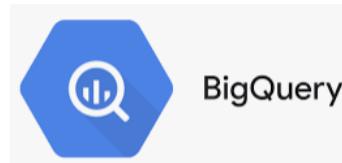
Database (DB) Vs. Data Warehouse (DW)



Key Takeaway

- ✓ MPP is a Data Warehouse system that concentrates on parallel processing software and hardware
- ✓ Query processing is divided into numerous smaller parallel jobs done concurrently across multiple servers
- ✓ Significantly reduces query and ingestion times

Popular MPP systems



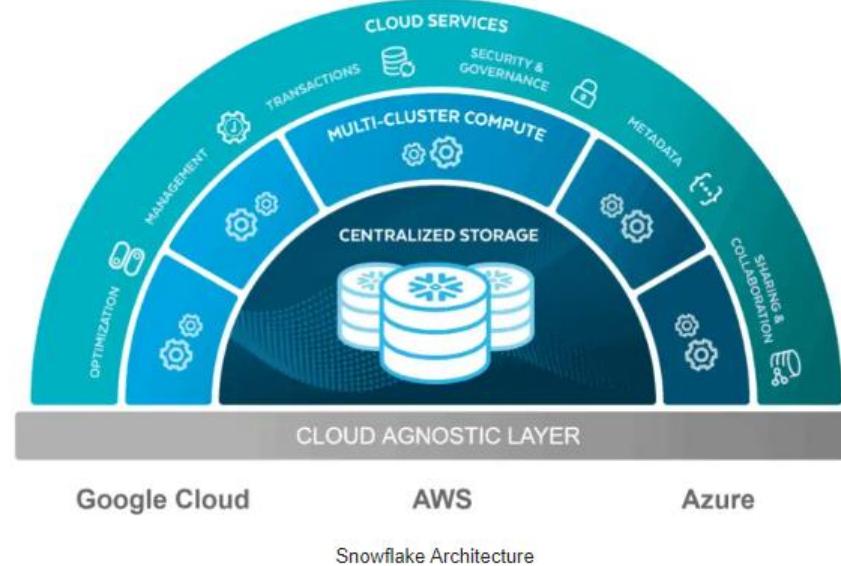
teradata.

Introduction to Snowflake

- Snowflake was founded by few Employees of Oracle, who launched Snowflake in 2012.
- Snowflake's Data Cloud is powered by an advanced data platform provided as Software-as-a-Service (SaaS)
- Snowflake enables data storage, processing, and analytic solutions that are faster, easier to use, and far more flexible than traditional offerings.
- The Snowflake data platform is NOT built on any existing database technology or “big data” software platforms such as Hadoop – it has been developed from scratch
- Snowflake combines a completely new SQL query engine with an innovative architecture natively designed for the cloud.



Snowflake Architecture - Introduction



Cloud Agnostic Layer

- Unlike many cloud data warehouses, Snowflake doesn't run on its own cloud.
- It uses the storage and compute resources from popular cloud computing platforms – Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP).
- Snowflake cannot run on a private cloud infrastructure, either on-premises or hosted.

BENEFITS



- There is no hardware (virtual or physical) to select, install, configure, or manage.
- There is virtually no software to install, configure, or manage.
- Ongoing maintenance, management, upgrades, and tuning are handled by Snowflake.

Snowflake Editions

STANDARD



- Complete SQL data warehouse
- Secure Data Sharing across regions / clouds
- Premier Support 24 x 365
- 1 day of time travel
- Always-on enterprise grade encryption in transit and at rest
- Customer-dedicated virtual warehouses
- Federated authentication
- Database replication
- External Functions
- Snowsight
- Create your own Data Exchange
- Data Marketplace access

ENTERPRISE



- Standard +**
- Multi-cluster warehouse
- Up to 90 days of time travel
- Annual rekeying of all encrypted data
- Materialized views
- Search Optimization Service
- Dynamic Data Masking
- External Data Tokenization

BUSINESS CRITICAL



- Enterprise +**
- HIPAA support
- PCI compliance
- Tri-Secret Secure using customer-managed keys
- AWS PrivateLink support
- Azure Private Link support
- Google Cloud Private Service Connect support
- Database failover and fallback for business continuity
- External Functions - AWS API Gateway Private Endpoints support

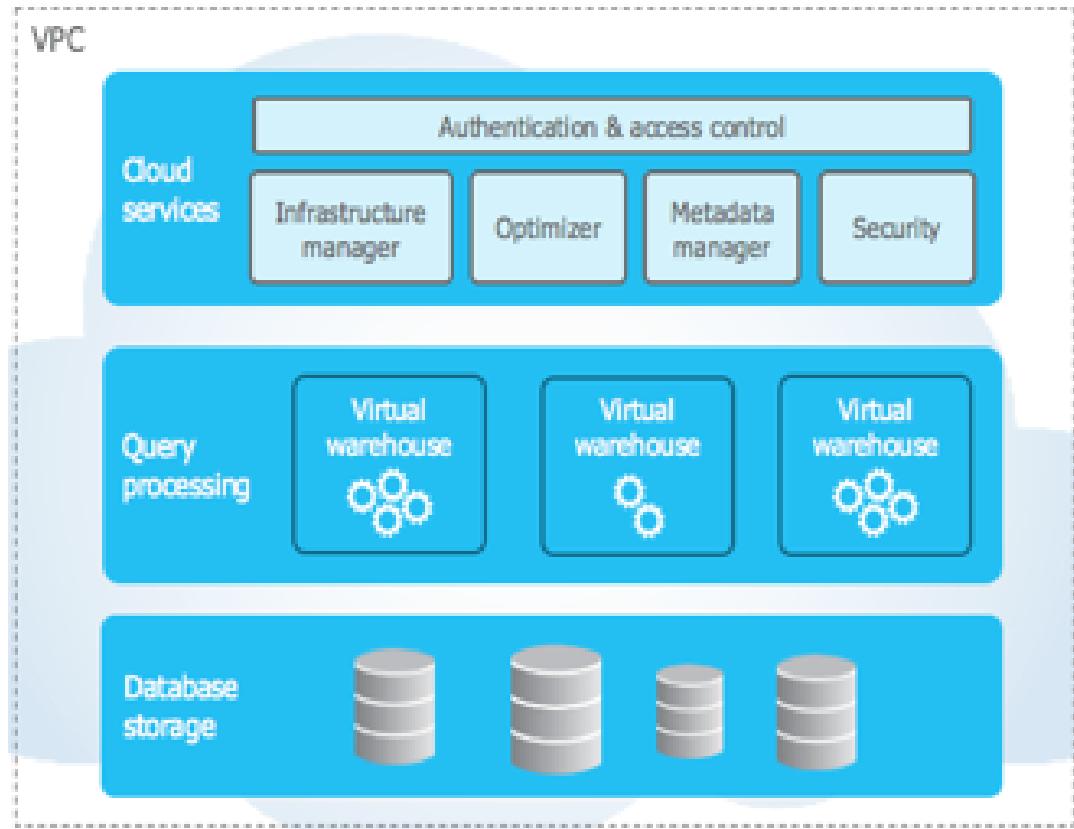
VIRTUAL PRIVATE SNOWFLAKE (VPS)



Business Critical +

- Customer-dedicated virtual servers wherever the encryption key is in memory
- Customer-dedicated metadata store

Snowflake Architecture



- Snowflake's architecture is a hybrid of traditional shared-disk and shared-nothing database architectures. Similar to shared-disk architectures, Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the platform.
- But similar to shared-nothing architectures, Snowflake processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally.
- This approach offers the data management simplicity of a shared-disk architecture, but with the performance and scale-out benefits of a shared-nothing architecture.

Snowflake Architecture - Storage

Database Storage

- When data is loaded into Snowflake, Snowflake reorganizes that data into its internal optimized, compressed, columnar format. Snowflake stores this optimized data in cloud storage.
- Snowflake manages all aspects of how this data is stored — the organization, file size, structure, compression, metadata, statistics, and other aspects of data storage are handled by Snowflake.
- The data objects stored by Snowflake are not directly visible nor accessible by customers; they are only accessible through SQL query operations run using Snowflake.
- Snowflake stores this optimized data in cloud storage
- Data stored in Storage is immutable

Snowflake Architecture – Virtual Warehouses

Query Processing

- Query execution is performed in the processing layer.
- Snowflake processes queries using “virtual warehouses”.
- Each virtual warehouse is an independent compute cluster that does not share compute resources with other virtual warehouses. As a result, each virtual warehouse has no impact on the performance of other virtual warehouses.
- Sizing of Warehouse is done in form of T-Shirt Sizing.
- Multiple Warehouse can access the same data in parallel
- Scale up/down or Scale out can be done in a second.

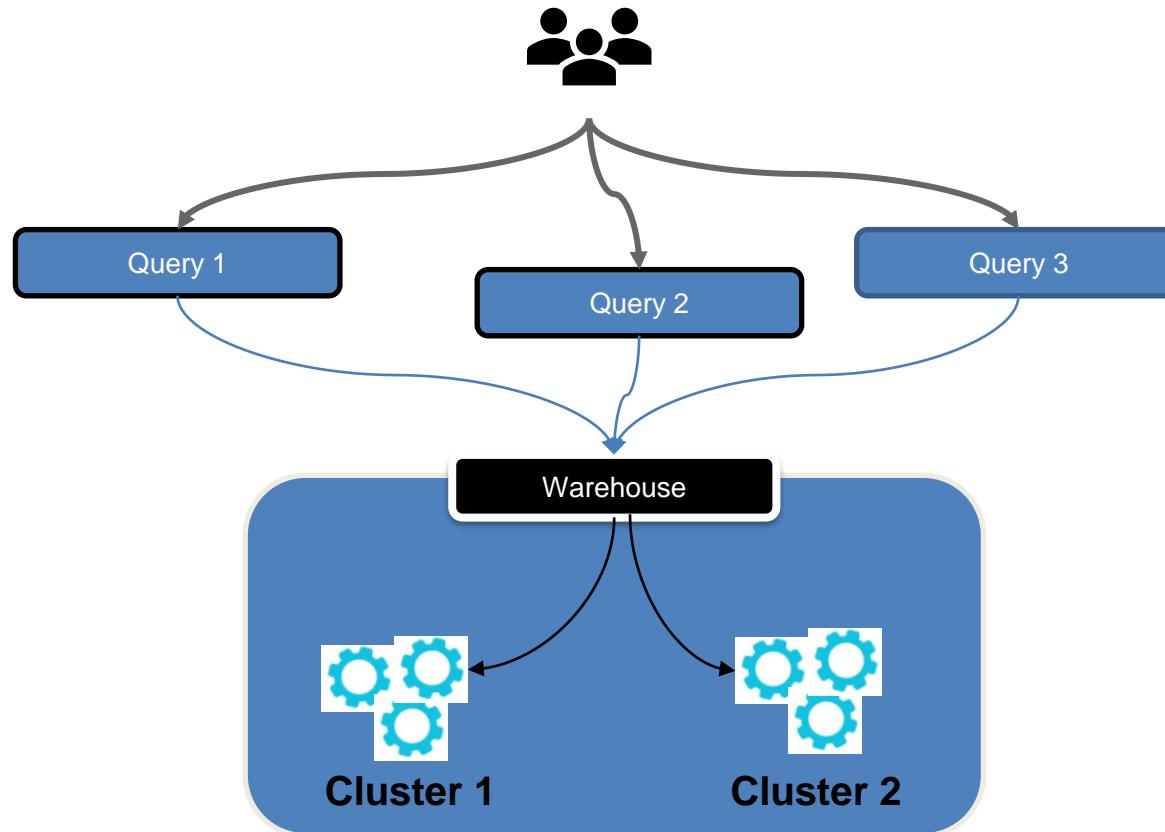
Snowflake Architecture – Cloud Services

- This is “MAGIC” Layer of Snowflake, which acts like “Brain”
- Cloud services layer is completely managed by Snowflake
- It is a collection of services that coordinates various activities in Snowflake
- Collection of Services include:
 - ❑ Authentication
 - ❑ Infrastructure Management
 - ❑ Metadata Management
 - ❑ Query parsing and optimization
 - ❑ Access Control
- The cloud services layer also runs on compute instances provisioned and managed by Snowflake.

Warehouse

- Warehouse (WH) are compute engine of Snowflake
- WH are required for running any queries as well as DML operations
- WH is defined by its Size, like XS, S, M, L, XL etc.
- WH can be stopped or started at any time
- Charges to WH only applies when they are running
- Recommendation is to enable Auto-Resume & Auto-Pause configuration on WH to optimize the cost.
- WH can be resized on the fly (w/o any downtime)

Warehouse Architecture – Multi-Cluster WH Example



Warehouse size & Credits

Warehouse Size	Credits / Hour	Credits / Second	Notes
X-Small	1	0.0003	Default size for warehouses created using CREATE WAREHOUSE .
Small	2	0.0006	
Medium	4	0.0011	
Large	8	0.0022	
X-Large	16	0.0044	Default for warehouses created in the web interface.
2X-Large	32	0.0089	
3X-Large	64	0.0178	
4X-Large	128	0.0356	
5X-Large	256	0.0711	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.
6X-Large	512	0.1422	Generally available in Amazon Web Services regions, and in preview in US Government and Azure regions.

Region Considerations

- Users Proximity
- Network Latency
- Cost Implications – Cost is NOT same across all regions (Link: [Snowflake Pricing & Cost Structure | Snowflake Data Cloud](#))
- Cloud Service Provider Region from where data will be loaded into Snowflake
- Regional Laws

As of Jun'2024, there are no egress charges for Snowflake

Account Identifier

-
- A hostname for a Snowflake account starts with an *account identifier* and ends with the Snowflake domain (snowflakecomputing.com).
 - Snowflake supports two formats to use as the account identifier in your hostname :
 - Account Name (Recommended)
 - Account Locator (Legacy)

Example:

Account Name:

`https://<orgname>-<account_name>.snowflakecomputing.com`

Account Locator: `https://<accountlocator>.<region>.<cloud>.snowflakecomputing.com`

Knowledge Check

Q1. Which Data Warehouse component is responsible for supporting end-user reporting and querying?

- A. Staging Area
- B. Data Mart
- C. OLAP Server
- D. Metadata Repository

Answer: C

Q2. What is the purpose of a Staging Area in Data Warehouse architecture?

- A. Real-time data storage
- B. Temporary storage for raw data before transformation
- C. Final storage for analytical data
- D. Query optimization

Answer: B

Knowledge Check

Q3. What is the primary advantage of using a Data Mart in a Data Warehouse architecture?

- A. Centralized storage for all organizational data
- B. Improved query performance for specific business units
- C. Real-time data processing capabilities
- D. Efficient support for complex ETL processes

Answer: B

Q4. Which of the following is one of the function of Cloud Services in Snowflake Cloud Data Warehouse architecture?

- A. Process the query
- B. Ingest Data into Warehouse
- C. Authentication & Access
- D. Persists Data

Answer: C

Knowledge Check

Q5. How many Warehouses can be created in Snowflake Cloud Data Warehouse

- A. 4
- B. 2
- C. 1
- D. No Limit

Answer: D

Q6. Which of the following is NOT a valid size of Snowflake Warehouse

- A. Large
- B. 2XL
- C. XS
- D. 2XS

Answer: D



BITS Pilani

Pilani Campus

Dimensional Modelling

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Module 3

Module 3 – Introduction to Dimensional Modelling

- 3.1 - Dimensional Modelling
- 3.2 - ER Modelling vs Dimensional Modelling
- 3.3 - Star, Snowflake, Starflake schemas
- 3.4 - Data Warehouse design steps
- 3.5 - Slowly changing dimensions
- 3.6 - Types of facts, dimensions
- 3.7 – Retail, Inventory Case Study

Dimensional Modelling

Dimensional Modelling

- **Dimensional Modeling (DM)** is a data structure technique optimized for data storage in a Data warehouse.
- The purpose of dimensional modeling is to optimize the database for faster retrieval of data.
- A dimensional model in data warehouse is designed to read, summarize, analyze numeric information like values, balances, counts, weights, etc. in a data warehouse.
- In contrast, relation models are optimized for addition, updating and deletion of data in a real-time Online Transaction System.

Elements of Dimensional Modelling

Fact:

- Facts are the measurements/metrics or facts from your business process.
- Facts are usually numeric and represent key performance indicators (KPIs) or business metrics.
- Example: In a sales data warehouse, a fact might be the **total sales amount**, **the quantity of products sold**, or the **number of orders placed**.

Key Characteristics:

- Numeric and measurable.
- Represent the business metrics or KPIs.
- Often involve aggregations and calculations.

Elements of Dimensional Modelling

Fact:

- **Measurable Metrics:** In a retail data warehouse, a fact could be the "Total Sales Amount" for a specific day, week, or month.
- **Aggregated Values:** A fact might represent the "Total Quantity Sold" of a particular product across multiple stores.
- **Count-based Metrics:** In a call center data warehouse, a fact could be the "Number of Calls Received" during a specific time period.
- **Duration-based Metrics:** In a service-level agreement (SLA) monitoring system, a fact might be the "Total Downtime" of a system in hours.
- **Financial Metrics:** For a financial institution, a fact could be the "Total Assets Under Management" for a specific portfolio.

Elements of Dimensional Modelling

Dimension:

- A dimension provides context and additional descriptive information about the data in a data warehouse.
- Dimensions are the descriptive data elements that are used to categorize or classify the data.
- Dimensions are typically **non-numeric** attributes that help to **categorize, filter, and analyze** the facts.

Example: In the same sales data warehouse, dimensions might include information about products (e.g., product category, brand), customers (e.g., customer name, region), and time (e.g., date, month, year).

Customer Dimension: Customer Name, Address, and Customer Segment.

Geographical Dimension: City, State, and Country.

Employee Dimension: Employee Name, Department, and Job Title.

Elements of Dimensional Modelling

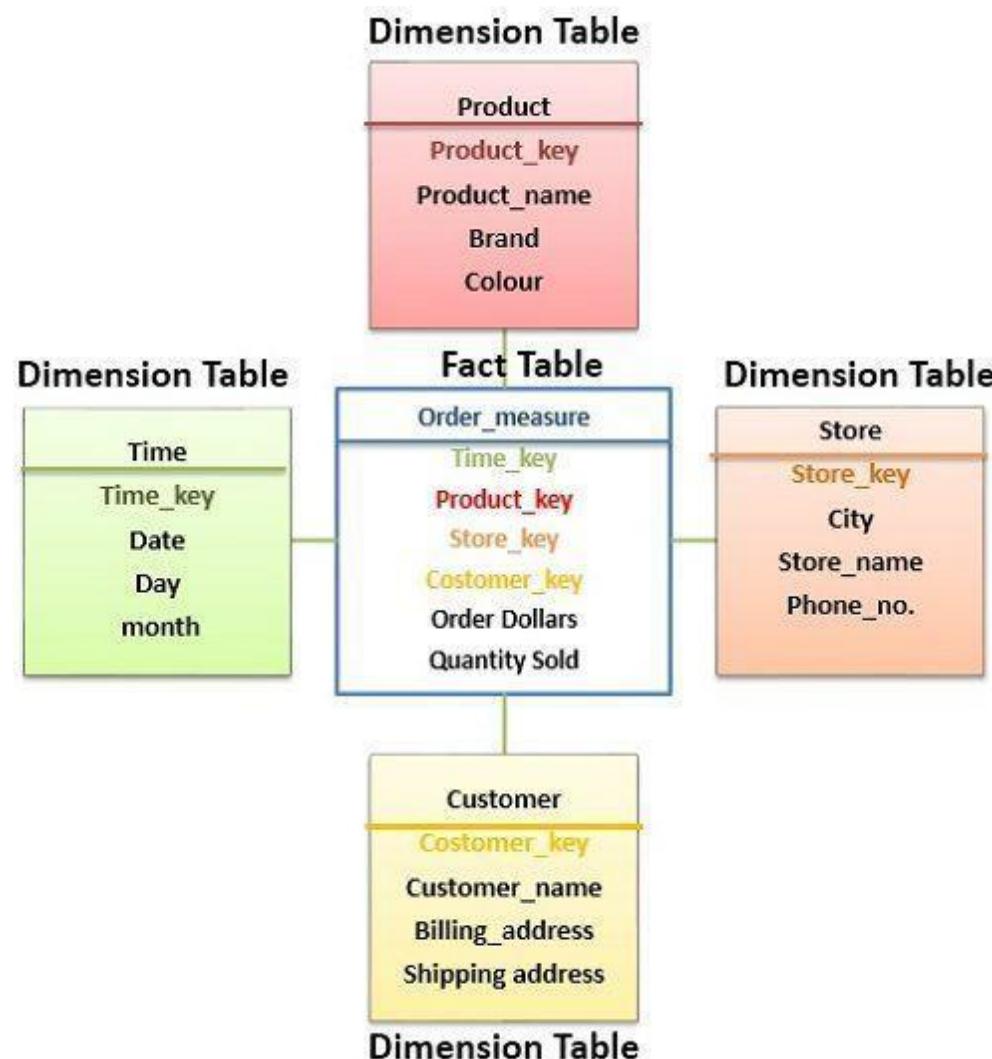
Fact Table

- Stores the performance measurements resulting from a business process.
- In a dimensional data model, the fact table is the central table that contains the **measures** or **metrics of interest**, surrounded by the dimension tables that describe the attributes of the measures.

Dimension Table

- Dimension tables contain the details about the facts.
- The dimension tables are related to the fact table through **foreign key** relationships.
- Dimension tables are simply denormalized tables.
- The dimensions can have one or more relationships.

Dimensional Modelling - Example



Dimension Tables

- These are the tables that are joined to fact tables.
- It describes the “who, what, where, when, how, and why” associated with the business event.
- It contains the descriptive attributes used for grouping and filtering the facts.
- Dimensions describe the measurements of the fact table.
- For example, customer id is a measurement, but we can describe its attributes further, more as what is the name of the customer, the address of the customer, gender, etc.

Dimension Tables - Example

Date_Dimension
Date Key (PK)
Date
Day
Month
Year
Quarter1
Quarter2
Quarter3
Quarter4

Product_Dimension
Product Key(PK)
Product id
Product Description
Brand
Department Number
Department Description
Weight

Order_Dimension
Order key (PK)
Order id
Payment Type
Expected Delivery Date
Deliver Address
#Products Ordered

Customer_Dimension
Customer Key(PK)
Customer id
Customer Name
Gender
Address
Contact Number

Promotion_Dimension
Promotion Key (PK)
Promotion Code
Promotion Name
Price Reduction Type
Promotion Begin Date
Promotion End Date

Warehouse_Dimension
Warehouse key (PK)
Warehouse Number
Warehouse Name
Address
State
District
Warehouse Manager

Dimension Tables - Example

Fact Table

Time ID	Product ID	Customer ID	Unit Sold
4	17	2	1
8	21	3	2
8	4	1	1

Dimension Tables

Customer ID	Name	Gender	Income	Education	Region
1	Rohan	Male	2	3	4
2	Sandeep	Male	3	5	1
3	Gaurav	Male	1	7	3

Dimension Tables Vs Fact Tables

Aspect	Fact Table	Dimension Table
Content	Contains quantitative, numeric data (facts). Examples include sales amounts, quantities, and metrics.	Contains descriptive, non-numeric attributes that provide context to the facts. Examples include customer names, product details, and time attributes.
Granularity	Typically has a finer granularity, representing the lowest level of detail in the data warehouse.	Represents aggregated and summarized data at various levels.
Primary Key	Usually has a composite primary key composed of foreign keys from dimension tables.	Has a single primary key, often a surrogate key, which is unique to each dimension record.
Foreign Keys	Contains foreign keys linking to various dimension tables, establishing relationships.	May have foreign keys linking to higher level dimensions or to other related dimensions.

Dimension Tables Vs Fact Tables

Aspect	Fact Table	Dimension Table
Measures	Contains measures or metrics that are subject to analysis, aggregation, and reporting.	Does not contain measures; instead, it contains descriptive attributes for categorization and filtering.
Aggregation	Data is pre-aggregated to support efficient query performance for analytical reporting.	Typically does not involve pre-aggregated data; instead, aggregations are performed during querying.
Changes Over Time	May incorporate historical data through techniques like slowly changing dimensions (SCD).	Historical changes are less common; dimensions often reflect the current state of attributes.
Size and Volume	Tends to have a larger volume of data due to detailed and fine-grained information.	Generally has a smaller volume compared to fact tables due to its descriptive nature.
Indexes	Indexing strategies are critical for performance optimization, especially for large fact tables.	Indexing is still important but may not be as critical as in fact tables for query performance.
Example Use Cases	Sales fact table, containing daily sales transactions with measures like sales amount and quantity sold.	Customer dimension table, containing attributes such as customer name, address, and demographic details.
Star Schema	Central component in a star schema or snowflake schema, connecting to multiple dimensions.	Forms the outer layer of a star schema, providing context to the measures in fact tables

Types of Dimensions

Conformed Dimension:

Definition: A conformed dimension is a dimension that is shared and consistent across multiple data marts or data warehouses within an organization.

Use Case: For example, a "**Time**" dimension with consistent date hierarchies and attributes is conformed across various business units or departments.

Outrigger Dimension:

Definition: An outrigger dimension is an additional set of attributes associated with an existing dimension table. It extends the information available in the primary dimension.

Use Case: Adding supplementary attributes like additional customer details (e.g., social media handles) to an existing "Customer" dimension.

Shrunken Dimension:

Definition: A shrunken dimension is a subset of a larger dimension that is specific to a particular department or business process. It's derived from a larger, enterprise-wide dimension.

Use Case: A "Product" dimension that is a subset focused on a specific product category or line within a larger enterprise-wide "Product" dimension

Types of Dimension

Role-Playing Dimension:

Definition: A role-playing dimension is a single dimension table that is used in multiple ways (roles) within the same fact table. Each role represents a different perspective or context.

Use Case: Using a "Date" dimension for order date, ship date, and delivery date within the same fact table.

Dimension to Dimension Table:

Definition: A dimension-to-dimension table, also known as a bridge table, is used to represent a many-to-many relationship between two dimensions in a fact table.

Use Case: Representing the relationship between "Employees" and "Projects" where an employee can work on multiple projects, and a project involves multiple employees.

Degenerate Dimension:

Definition: A degenerate dimension is a dimension that is derived from a fact table and does not have its own dedicated dimension table. It's a grouping of attributes without a separate entity.

Use Case: Using an invoice number as a degenerate dimension in a sales fact table

Types of Dimension

Swappable Dimension:

Definition: A swappable dimension allows for the substitution of one dimension for another without impacting the structure or integrity of the data warehouse.

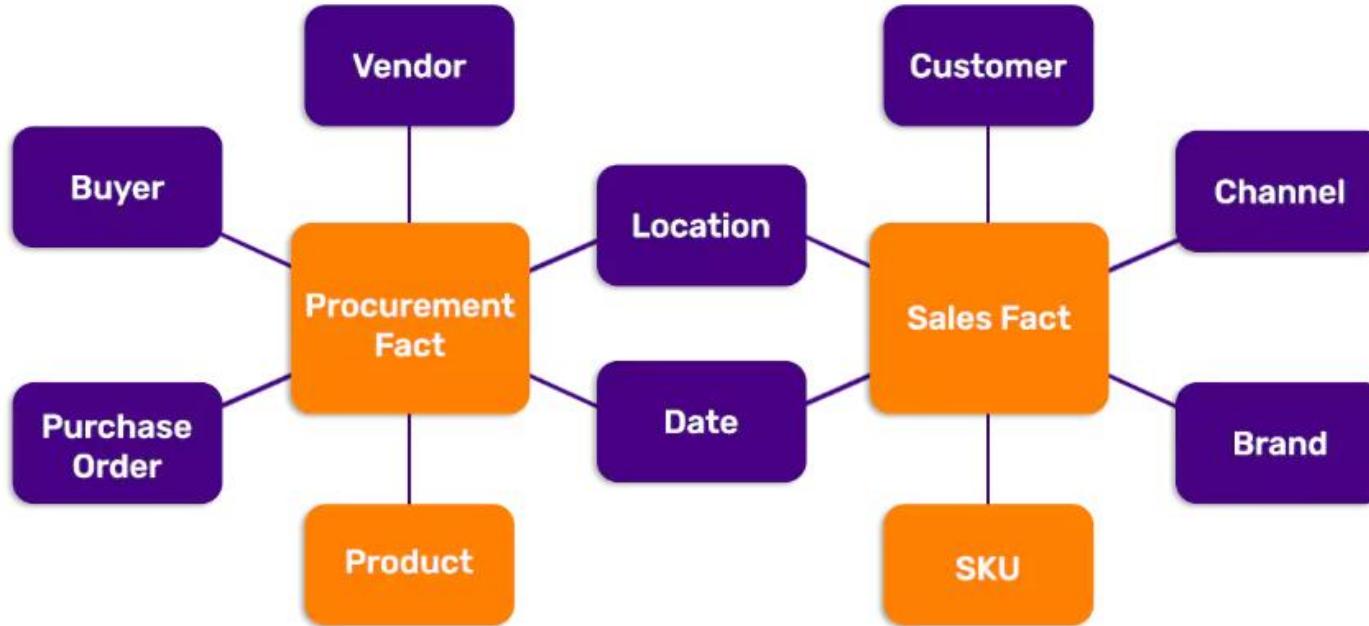
Use Case: Substituting a "Product" dimension with a "Promotional Event" dimension for a specific analysis without modifying the schema.

Step Dimension:

Definition: A step dimension represents a sequence or steps in a process, often used to track the progress of an entity through different stages.

Use Case: Modeling the stages of a customer's journey, where each stage represents a step in the dimension.

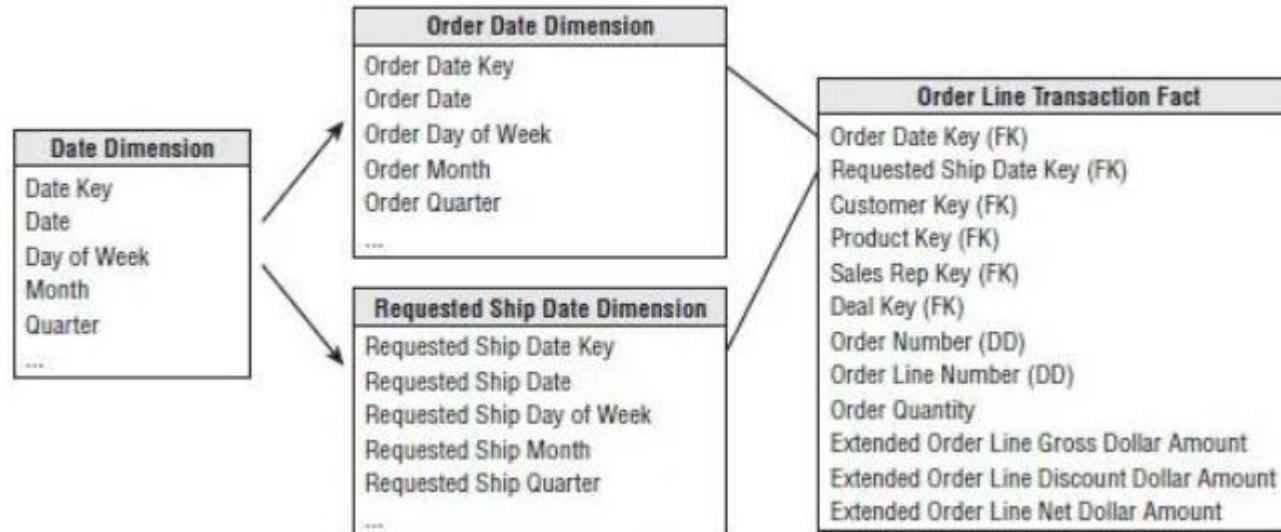
Few Examples



Conformed Dimension

A conformed dimension is any dimension that is shared across multiple fact tables or subject areas. The diagram below shows a simple conceptual model of conformed dimension.

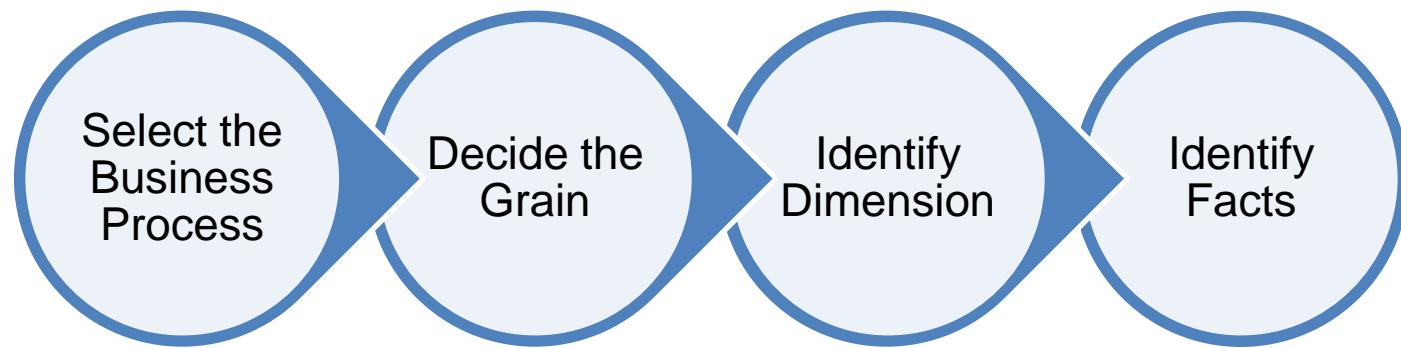
Few Examples



Role-Playing Dimension

A **role-playing dimension** is a dimension that can filter related facts differently. For example, the date dimension table has two relationships to the online transaction facts. The same dimension table can be used to filter the facts by ship date or order date.

Steps to Create Dimensional Data Modelling



Step 1: Identify Business Process

Step 1: Select the Business Process

The first step involves selecting the business process, and it should be an action resulting in output.

Business Process #1: The e-Commerce industry is widely known for selling and buying goods over the internet, so our first business process will be the **products** bought by the customers.

Business Process #2: Delivery status is also one of the most important business processes in this industry. It tells us where the product is currently from. It's dispatched from the warehouse to the customer's given address.

Business Process #3: Maintaining the **inventory** in order to ensure that items don't run out of stock, how sales are going on etc.

Step 2: Decide the Grain

- Identifying the grain, or the level of detail, is a crucial step in dimensional modeling.
- The grain defines what each row in a fact table represents
- A grain is a business process at a specified level.
- All the rows in a fact table should result from the same grain.
- Each fact table is the result of a different grain selected in a business process.
- The grain should be as granular (at the lowest level) as possible.
- To identify the grain, you need a clear understanding of the business processes and the specific information that stakeholders want to analyze.

Grains for the above business processes are

Grain 1: We can have the grain as the **products purchased by the customer**, i.e., each row of the fact table will represent all the products checked out by the customer from the cart but suppose a customer ordered 100 products, so this will be represented as a single row. Grain will be an individual product ordered by a customer, i.e., one product per row. This will make the data simple and easy to query. Similarly, we will select the most granular grains for the remaining processes.

Grain 2: Here also, the grain will be the status of an individual product shipped from the warehouse to the delivery location.

Grain 3: Here, each row will represent the daily inventory for each product in each store., it will tell the stock of that product left in the inventory and how many products have already been sold.

Step 2: Decide the Grain - Examples

Sales Data:

Business Process: Recording sales transactions.

Grain: Each row in the fact table represents a unique sales transaction, capturing details such as the product sold, quantity, price, and customer.

Customer Interaction:

Business Process: Capturing customer interactions in a call center.

Grain: Each row represents a specific customer interaction, including details such as the call duration, customer, agent, and date.

Inventory Management:

Business Process: Managing inventory levels.

Grain: Each row represents the inventory level for a specific product at a specific location on a specific date.

Web Analytics:

Business Process: Tracking user interactions on a website.

Grain: Each row represents a specific user session, capturing details such as page views, time spent, and actions taken.

Employee Time Tracking:

Business Process: Recording employee work hours.

Grain: Each row represents the work hours logged by an employee for a specific date.

Healthcare Data:

Business Process: Recording patient encounters in a healthcare system.

Grain: Each row represents a patient encounter, capturing details such as the patient, healthcare provider, date, and services provided.

Step 3: Identify the Dimensions for the Dimensional Table

- It describes the “who, what, where, when, how, and why” associated with the business event. It contains the descriptive attributes used for grouping and filtering the facts.

Consider following dimension

Date_Dimension

Date Key (PK)
Date
Day
Month
Year
Quarter1
Quarter2
Quarter3
Quarter4

Product_Dimension

Product Key(PK)
Product id
Product Description
Brand
Department Number
Department Description
Weight

Order_Dimension

Order key (PK)
Order id
Payment Type
Expected Delivery Date
Deliver Address
#Products Ordered

Customer_Dimension

Customer Key(PK)
Customer id
Customer Name
Gender
Address
Contact Number

Promotion_Dimension

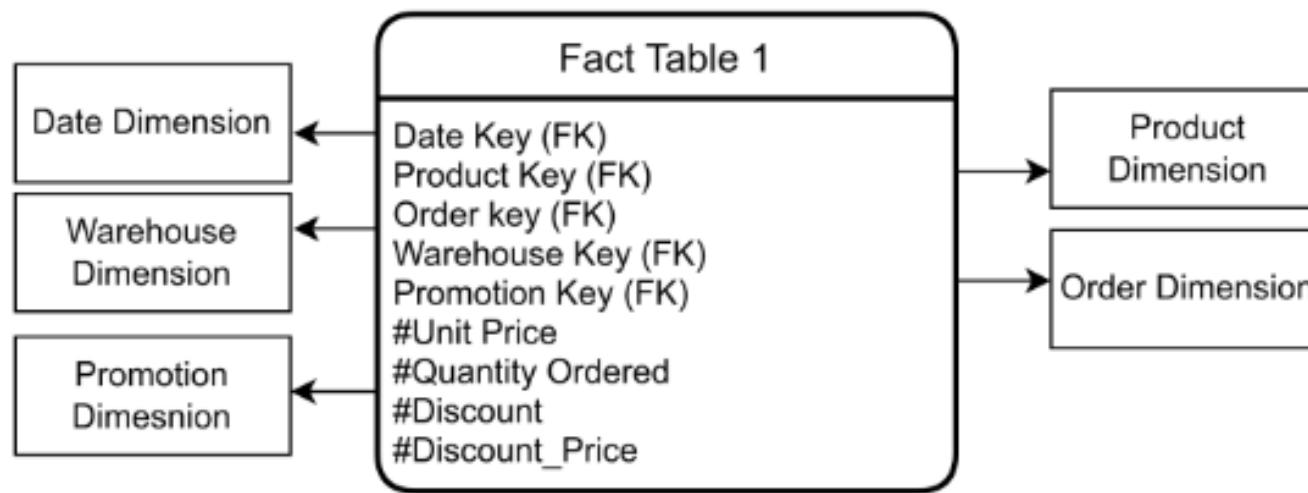
Promotion Key (PK)
Promotion Code
Promotion Name
Price Reduction Type
Promotion Begin Date
Promotion End Date

Warehouse_Dimension

Warehouse key (PK)
Warehouse Number
Warehouse Name
Address
State
District
Warehouse Manager

Step 4: Identify the Facts for the Dimensional Table

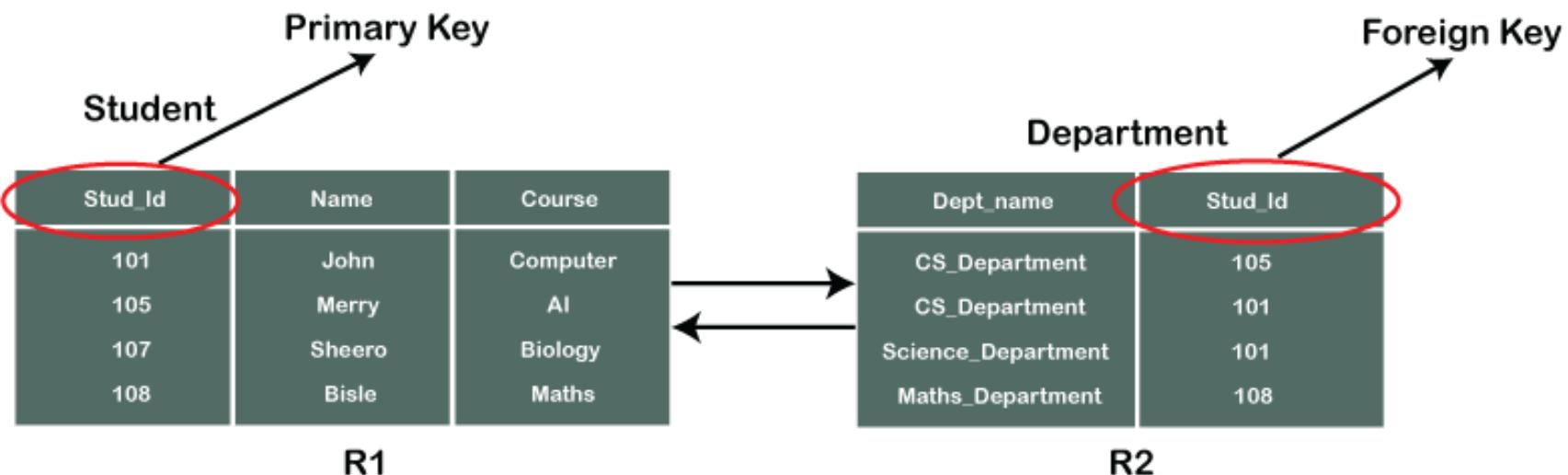
- The term fact represents a business measure; therefore, a fact table in dimensional modeling stores the performance measurements resulting from a business process.
- These performance measurements measure the business, i.e., these are the metrics through which we can infer whether our business is in profit or loss.
- Different business measurements can be unit price, number of goods sold, etc.



Grain: Individual product of the order per row.

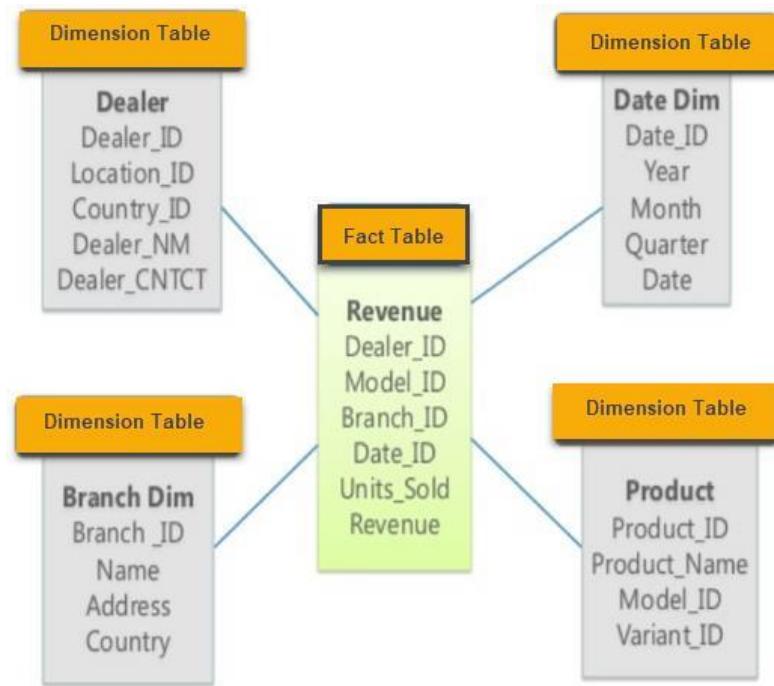
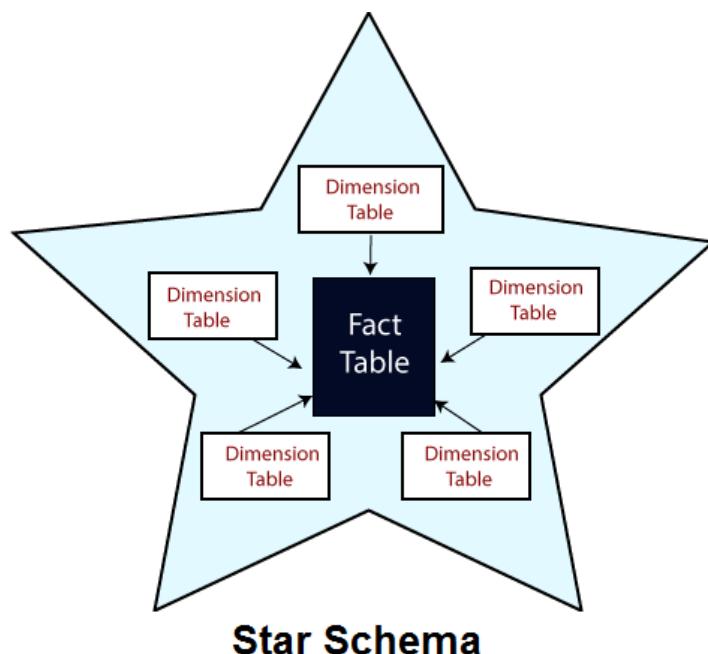
Schema

- Schema means the logical description of the entire database.
- It gives us a brief idea about the link between different database tables through keys and values.
- A data warehouse also has a schema like that of a database.
- In database modeling, we use the relational model schema.
- Whereas in the data warehouse, we use modeling Star, Snowflake, and Galaxy schema.



Star Schema

- A star schema is a type of data warehouse schema where a central fact table is connected to one or more dimension tables through foreign key relationships.
- This structure resembles a star when visualized, with the fact table at the center and dimension tables surrounding it like points on a star.
- The star schema is a popular design for data warehouses due to its simplicity and efficiency in querying and reporting.



Star Schema – Advantages & Disadvantages

ADVANTAGES:

1. Most Suitable for Query Processing: View-only reporting applications show enhanced performance.
2. Simple Queries: Optimized Navigation through the database. It is because the star join schema logic is much simpler.
3. Simplest and Easiest to design.

DISADVANTAGES:

1. They don't support many to many relationships[#] between business entities.
2. It is a result of each dimension having only one dimension table which may cause data redundancy. For example, a star schema would repeat the values in field customer_address_country for each order from the same country.

A many-to-many relationship occurs when multiple records in a table are associated with multiple records in another table.

117

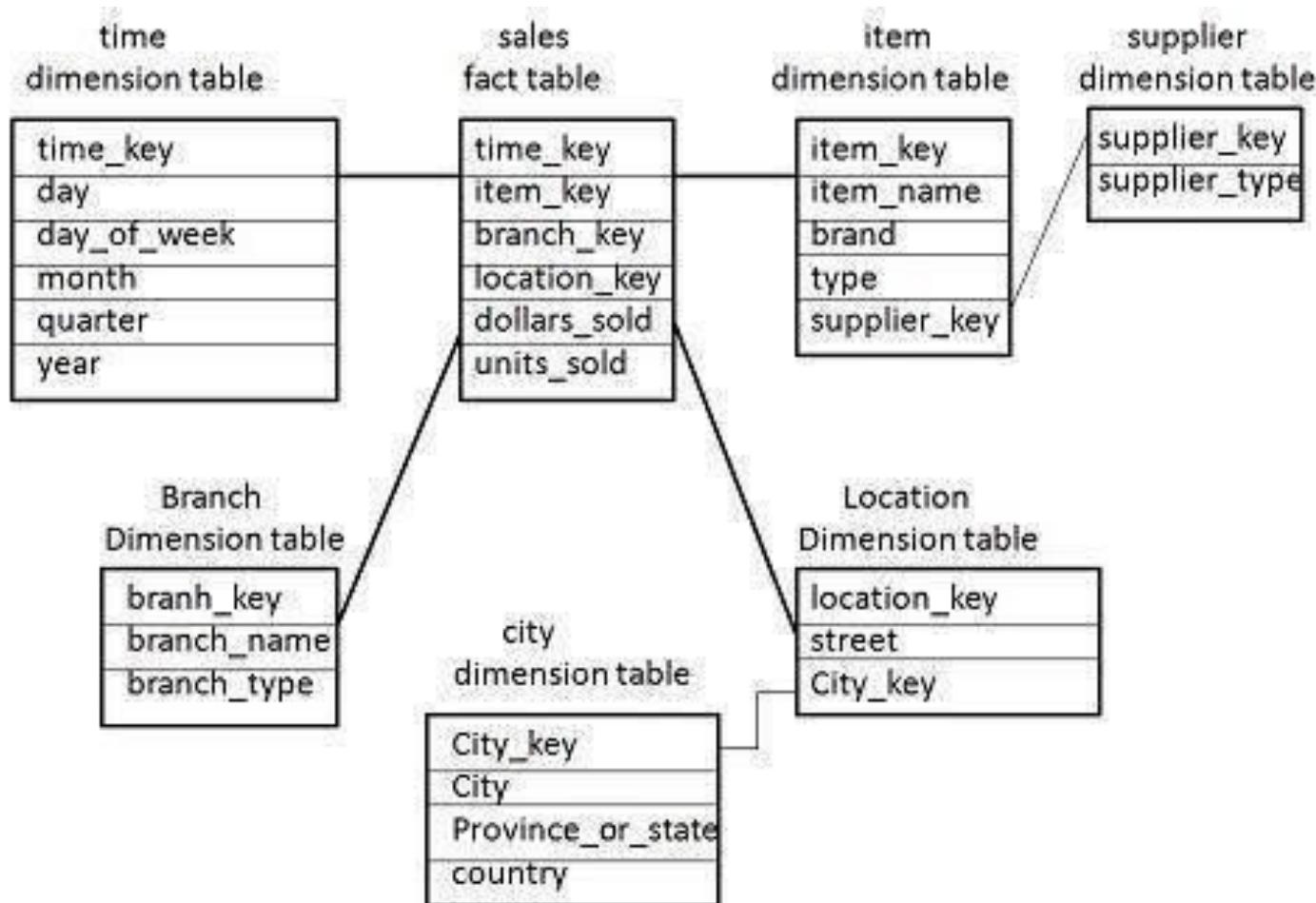
Snowflake Schema

- A snowflake schema is equivalent to the star schema.
- The snowflake schema consists of one fact table which is linked to many dimension tables, which can be linked to other dimension tables through a many-to-one relationship.
- It is called snowflake schema because the diagram of snowflake schema resembles a snowflake.
- Snowflaking is a method of normalizing the dimension tables in a STAR schemas.
- When we normalize all the dimension tables entirely, the resultant structure resembles a snowflake with the fact table in the middle.
- Some dimension tables in the Snowflake schema are normalized.
- The normalization splits up the data into additional tables.
- Unlike Star schema, the dimensions table in a snowflake schema are normalized. For example, the item dimension table in star schema is normalized and split into two dimension tables, namely item and supplier table.



[Click here to learn more about normalization](#)
118

Snowflake Schema - Example

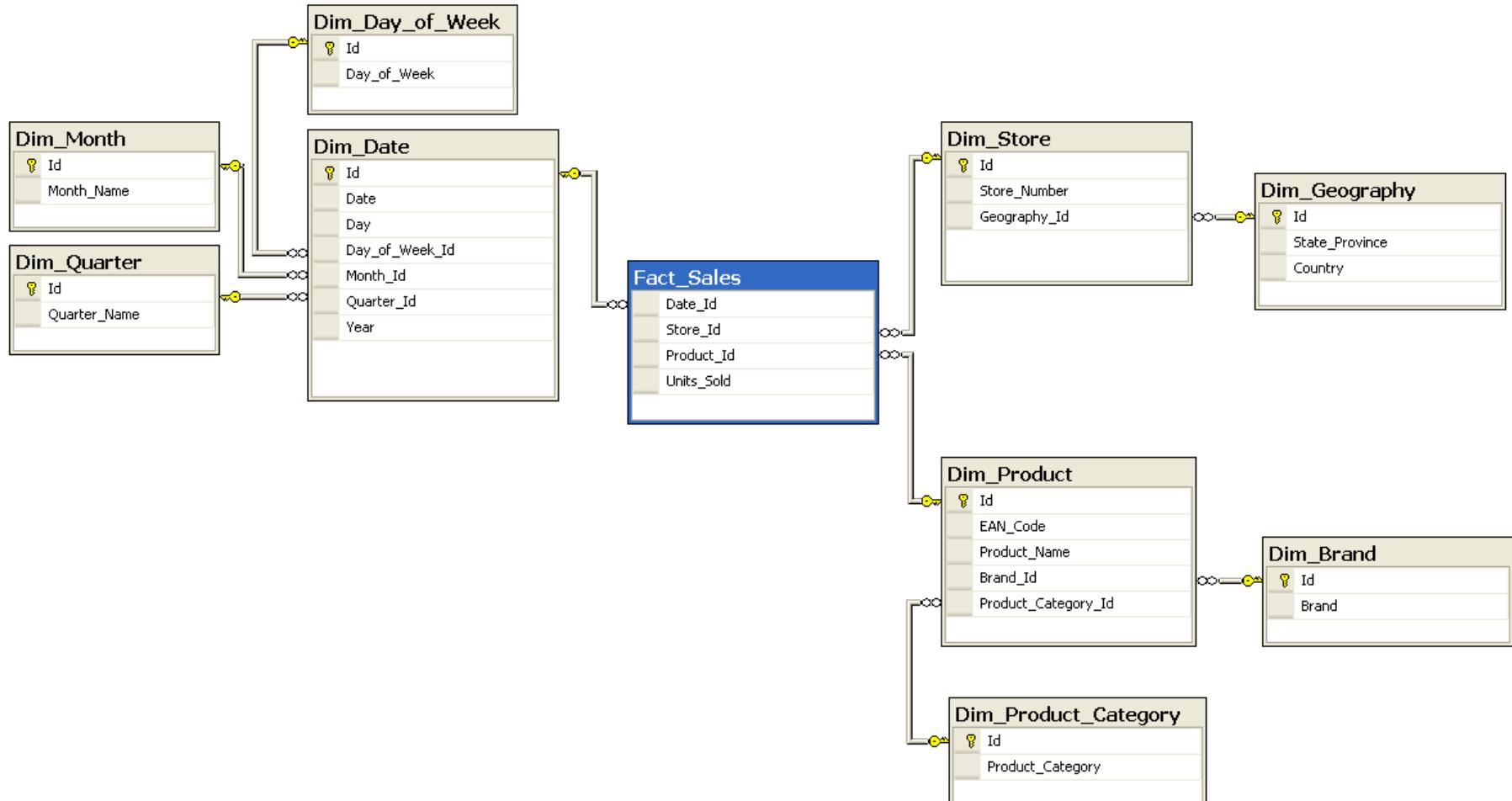


Snowflake Schema

- Central Fact Table: Similar to the star schema, there is a central fact table that contains the primary metrics and measures.
- Normalized Dimension Tables: Unlike the star schema, the dimension tables in a snowflake schema are normalized, meaning that they are divided into multiple related tables.
- Normalization involves breaking down large tables into smaller ones to reduce redundancy and improve data integrity.
- Hierarchical Structure: The normalized dimension tables are organized into a hierarchy, forming a structure that resembles a snowflake. Each level of the hierarchy represents a subset of the data.

The primary advantage of the snowflake schema is its ability to reduce data redundancy and improve data integrity through normalization

Snowflake Schema – An Example



Snowflake Schema – Downsides

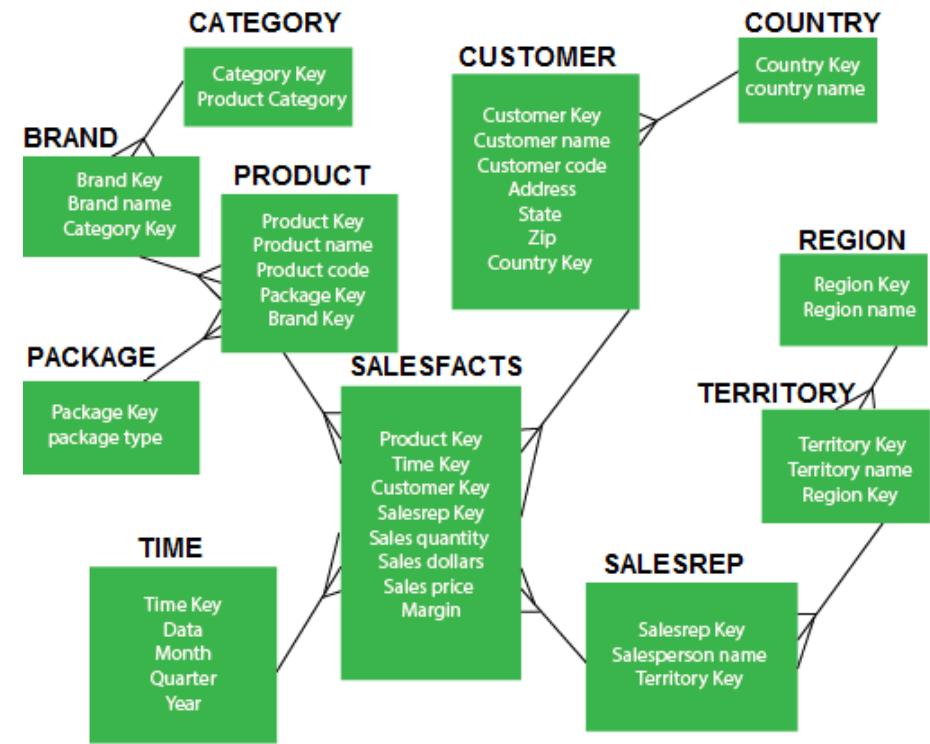
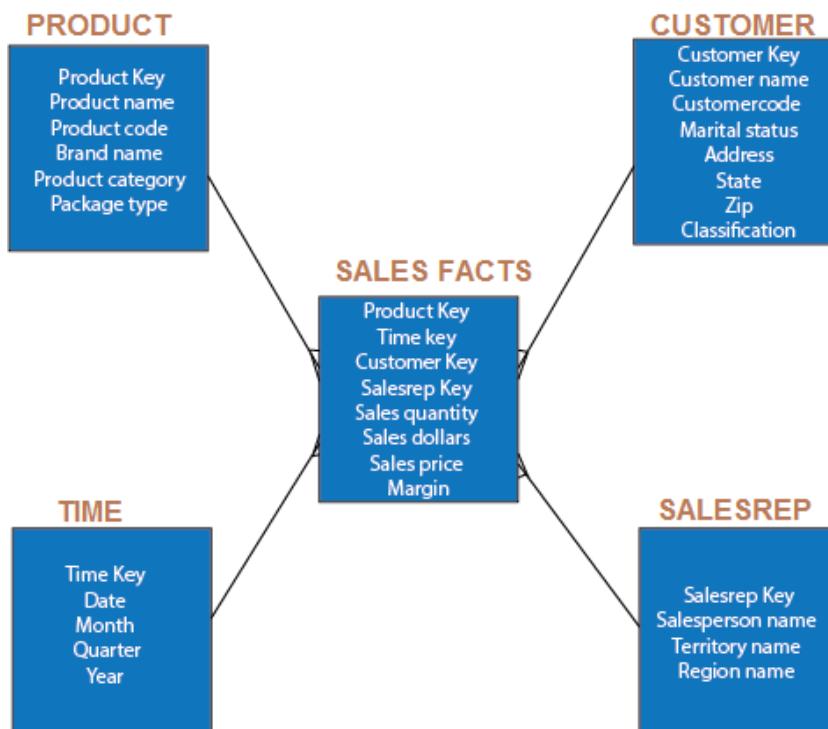
There are trade-offs associated with this approach, which are:

- **Query Performance:** While normalization can improve data integrity, it can also result in more complex queries, as analysts may need to join multiple tables to retrieve the required information. This can impact query performance.
- **Maintenance Complexity:** The snowflake schema can be more complex to design and maintain compared to the star schema. Changes to the schema structure may require modifications to multiple tables.

The choice between a star schema and a snowflake schema depends on the specific requirements of the data warehouse and the trade-offs that the organization is willing to make

Star Schema Vs Snowflake Schema

- STAR schema for sales in a manufacturing company.
- The sales fact table include quantity, price, and other relevant metrics. SALESREP, CUSTOMER, PRODUCT, and TIME are the dimension tables.

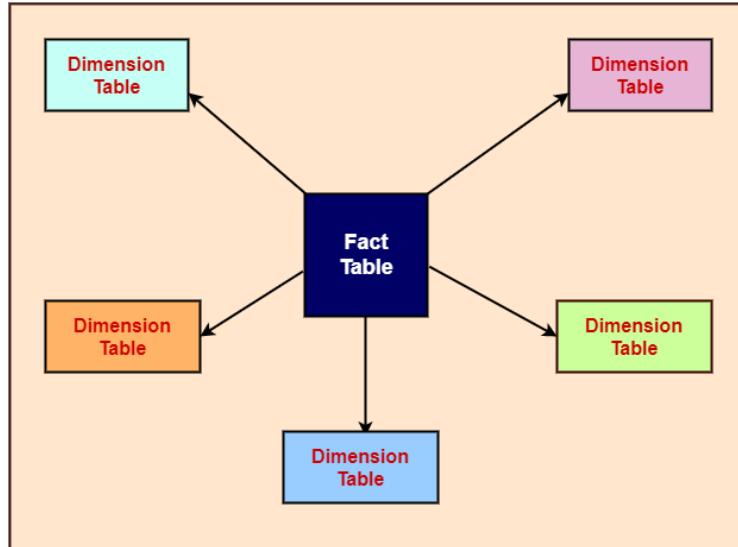


Star Schema Vs Snowflake Schema – An Example

- The STAR schema for sales, as shown in the example, contains only five tables, whereas the normalized version now extends to eleven tables.
- Notice that in the snowflake schema, the attributes with low cardinality in each original dimension tables are removed to form separate tables.
- These new tables are connected back to the original dimension table through artificial keys.

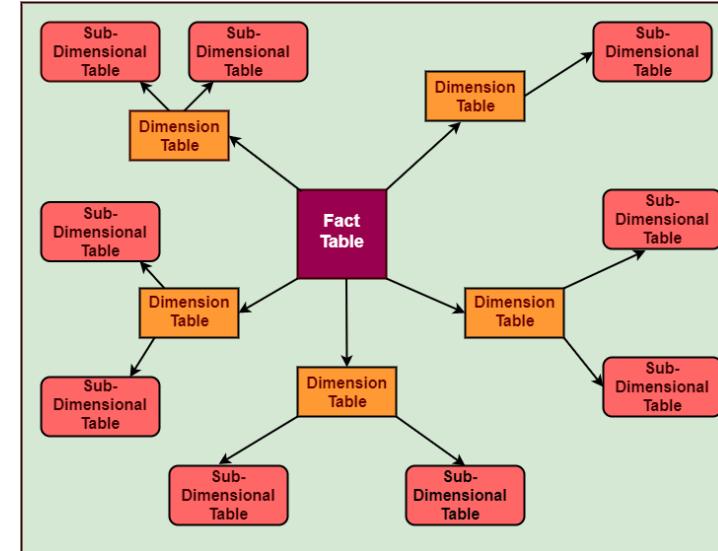
Low-cardinality refers to columns with few unique values

Star Schema Vs Snowflake Schema – Summary



Star Schema

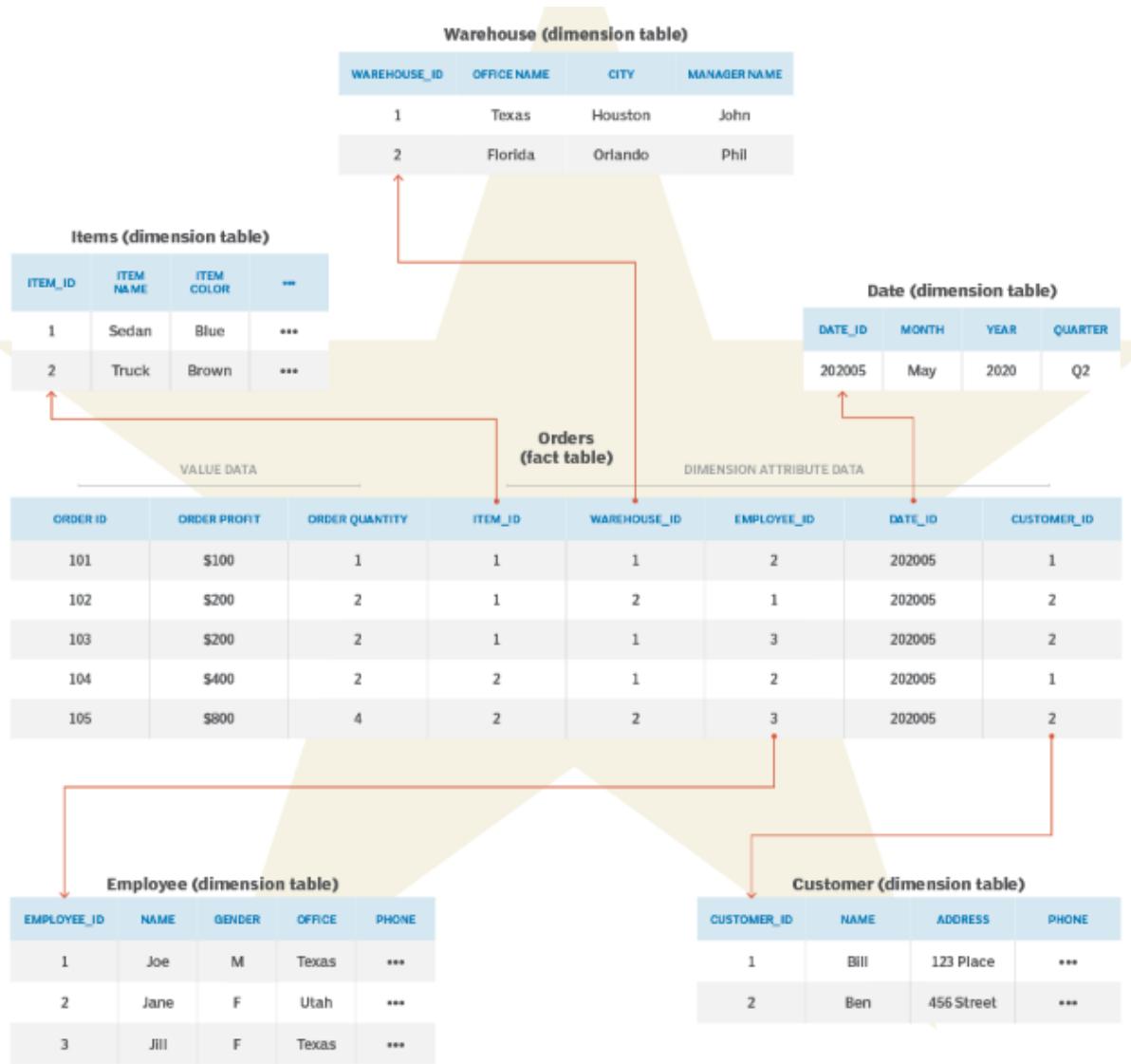
- In a star schema, the fact table will be at the center and is connected to the dimension tables.
- The tables are completely in a denormalized structure.
- SQL queries performance is good as there is less number of joins involved.
- Data redundancy is high and occupies more disk space



Snowflake Schema

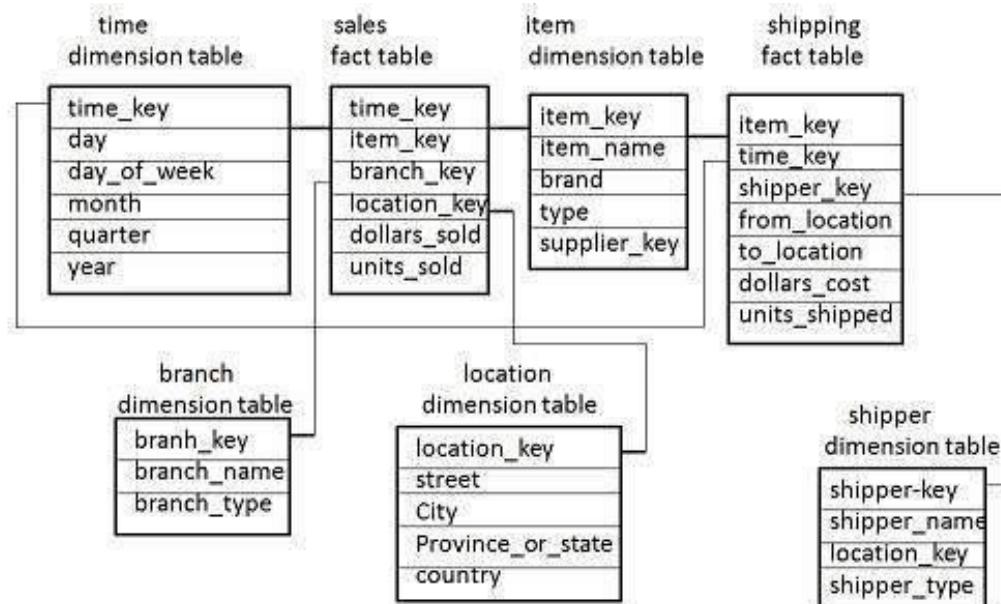
- A snowflake schema is an extension of star schema where the dimension tables are connected to one or more dimensions.
- The tables are partially denormalized in structure.
- The performance of SQL queries is a bit less when compared to star schema as more number of joins are involved.
- Data redundancy is low and occupies less disk space when compared to star schema

Star Schema – Demo



Fact Constellation Schema or Galaxy Schema

- A fact constellation has multiple fact tables. It is also known as galaxy schema.
- The following diagram shows two fact tables, namely sales and shipping.
- The sales fact table is same as that in the star schema.
- The shipping fact table has the five dimensions, namely item_key, time_key, shipper_key, from_location, to_location.
- The shipping fact table also contains two measures, namely dollars sold and units sold.
- It is also possible to share dimension tables between fact tables. For example, time, item, and location dimension tables are shared between the sales and shipping fact table.

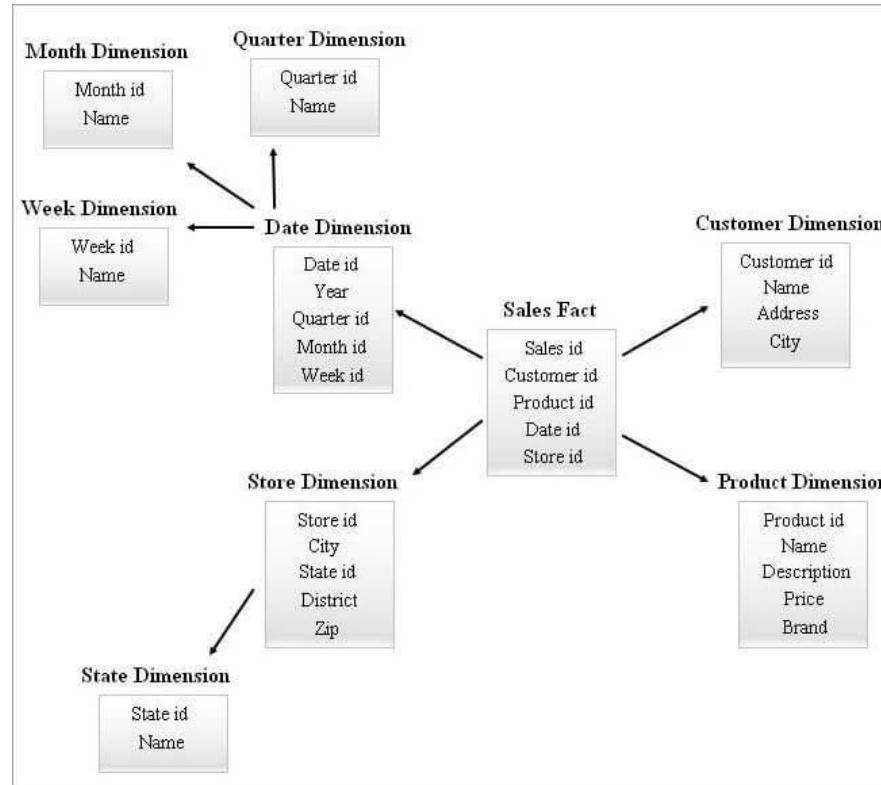


Fact Constellation Schema

- The primary disadvantage of the fact constellation schema is that it is a more challenging design because many variants for specific kinds of aggregation must be considered and selected.

Starflake Schema Or Star Cluster Schema

- A Snowflake schema with many dimension tables may need more complex joins while querying. A star schema with fewer dimension tables may have more redundancy. Hence, a star cluster schema came into the picture by combining the features of these two schemas.
- Star schema is the base to design a star cluster schema and few essential dimension tables from the star schema are snowflaked and this, in turn, forms a more stable schema structure.



Slowly Changing Dimensions (SCDs)

- Dimensions in a data warehouse are the descriptive information about business entities, and they often change over time.
- Slowly Changing Dimensions (SCDs) are a concept in data warehousing that deals with how to handle changes in dimension data over time.
- SCDs provide a way to manage these changes so that historical data remains accurate and meaningful.

Slowly Changing Dimensions (SCDs) - Types

Type 1 - Overwrite:

- In a Type 1 SCD, when a change occurs, the existing dimension record is simply updated with the new information.
- There is no tracking of historical changes, so the historical perspective is lost.
- This method is suitable when historical changes are not important for analysis, and the most recent information is sufficient.
- A good example of this is customer addresses. You don't need to keep track of how a customer's address has changed over time, you just need to know you are sending an order to the right place.

Type 2 – Add New Row:

- In a Type 2 SCD, when a change occurs, a new row is added to the dimension table with the updated information.
- The existing row remains unchanged, and it is marked as inactive or outdated.
- This approach preserves the historical changes and allows for tracking the history of the dimension over time.
- This method is suitable when historical changes need to be maintained for reporting and analysis.
- You can also handle type 2 dimensions by adding a timestamp column or two to show when a new record was created or made active and when it was made ineffective. Instead of checking for whether a record is active or not, you can find the most recent timestamp and assume that is the active data row. You can then piece together the timestamps to get a full picture of how a row has changed over time.

Slowly Changing Dimensions (SCDs) - Types

Type 2 – Add New Row - Example

ProductID	ProductPackName	Products	EffectiveDate	EndDate
P001	Summer Fun	[12345, 1245, 1456]	2022-01-01 00:00:00	2022-06-30 23:59:59
P001	Summer Fun	[12345, 1456, 3246]	2023-07-01 00:00:00	NULL
P002	Kid's Play	[1296, 3470]	2022-01-01 00:00:00	2022-03-31 23:59:59
P002	Kid's Play	[2969, 3470]	2022-04-01 00:00:00	NULL
P003	Party Pack	[12345, 6713]	2022-01-01 00:00:00	2022-07-31 23:59:59
P003	Party Pack	[6713, 34701]	2022-08-01 00:00:00	2022-11-30 23:59:59
P003	Party Pack	[12345, 7823]	2022-12-01 00:00:00	NULL

Slowly Changing Dimensions (SCDs) - Types

Type 3 – Add Columns

- Type 3 dimensions track changes in a row by adding a new column.
- Instead of adding a new row with a new primary key like with type 2 dimensions, the primary key remains the same and an additional column is appended.
- This is good if you need your primary key to remain unique and only have one record for each natural key.
- However, you can really only track one change in a record rather than multiple changes over time.
- Think of this as a dimension you'd want to use for one-time changes.
- For example, let's say your warehouse location is changing. Because you don't expect the address of your warehouse to change more than once, you add a `current_address` column with the address of your new warehouse. You then change the original address column name to be `previous_address` and store your old address information.

Slowly Changing Dimensions (SCDs) - Types

Type 4 – Add New Table

- Type 4 dimensions exist as records in two different tables- a current record table and a historical record table.
- All of the records that are active in a given moment will be in one table and then all of the records considered historical will exist in a separate history table.
- This is a great way of keeping track of records that have many changes over time.
- .



BITS Pilani

Pilani Campus

Data Warehouse - ETL

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Module 4

Module 4 – ETL

4.1.	ETL Overview
4.2	Data Extraction
4.3.	Data Transformation
4.4.	Data Loading
4.5.	Data Quality
4.6	Snowflake Cloud Datawarehouse – Data Loading

ETL (Extract, Transform, Load)

ETL (Extract, Transform, Load) is a process in data warehousing and data integration that involves the following key steps:

Step 1: Extract

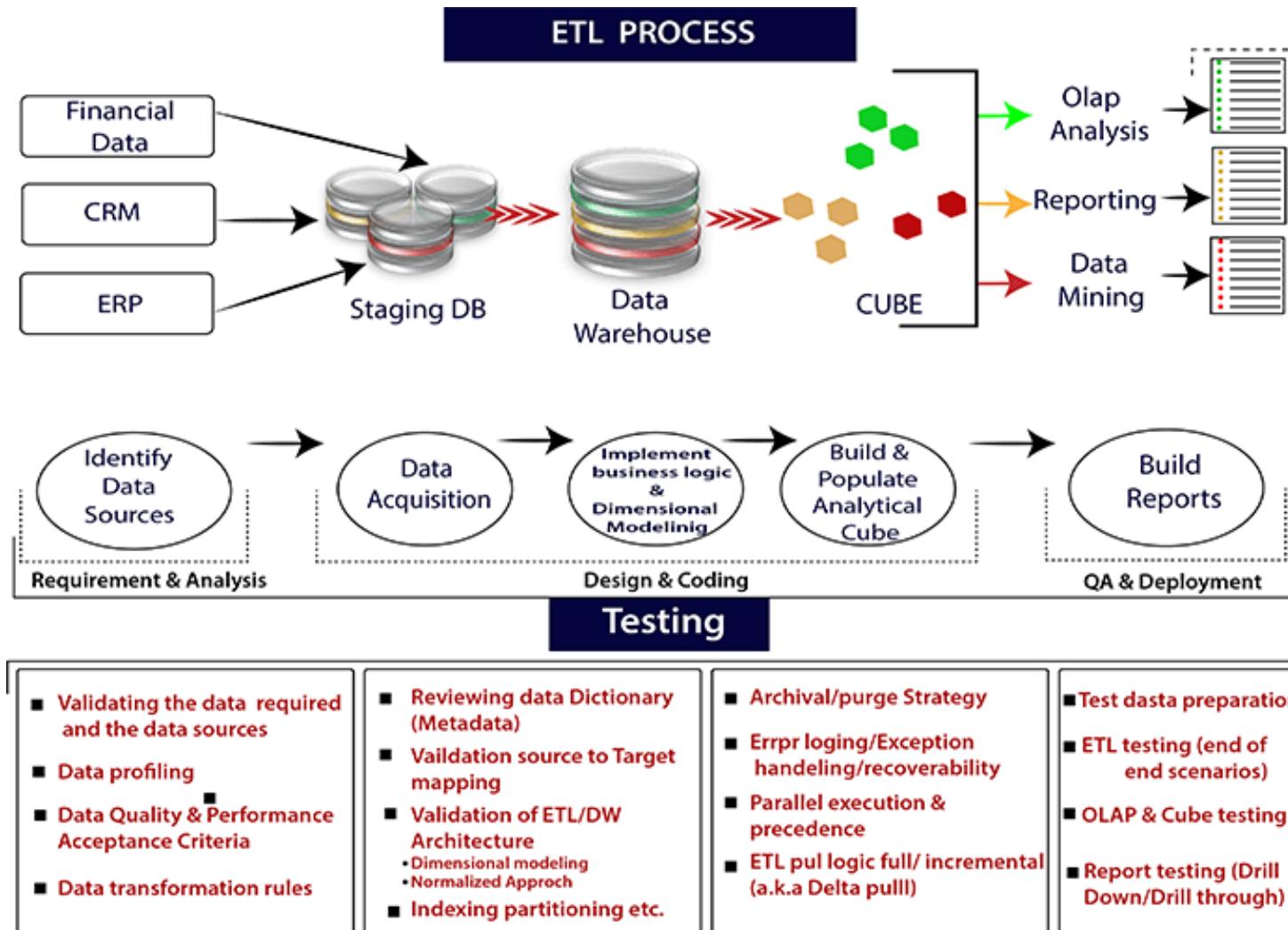
The extraction process involves copying or exporting raw data from multiple locations called source locations and storing them in a staging location for further processing.

Source locations can consist of any type of data, including SQL or NSQL servers, flat files, emails, logs, web pages, CRM, ERP systems, spreadsheets, logs, etc.

Common data extraction methods are:

- Partial extraction with update notification
- Partial extraction without update notification
- Full extraction

ETL (Extract, Transform, Load)



ETL (Extract, Transform, Load)

Step 2: Transform

In the transformation stage of the ETL process, data in the staging area is transformed through the data processing phase to make it suitable for use for analytics. Raw data is converted to a consolidated, meaningful data set.

Several tasks are performed on the data like:

- Cleaning and Standardization
- Verification and Validation
- Filtering and Sorting
- De-duplication
- Data audits
- Calculations, Translations
- Formatting
- Data encryption, protection

ETL (Extract, Transform, Load)

Step 3: Load

In this final step of the ETL process, the transformed data is loaded onto its target destination, which can be a simple database or even a data warehouse. The size and complexity of data, along with the specific organizational needs, determine the nature of the destination.

The load process can be:

Full loading – occurs only at the time of first data loading or for disaster recovery

Incremental loading – loading of updated data

ETL (Extract, Transform, Load)

ETL Tools - Some prominent ETL software tools are:

- Talend
- Oracle Data Integrator
- Amazon RedShift
- AWS Glue
- Matillion
- Azure Data Factory
- Fivetran

Process for ETL Testing (aka Data Quality Check)

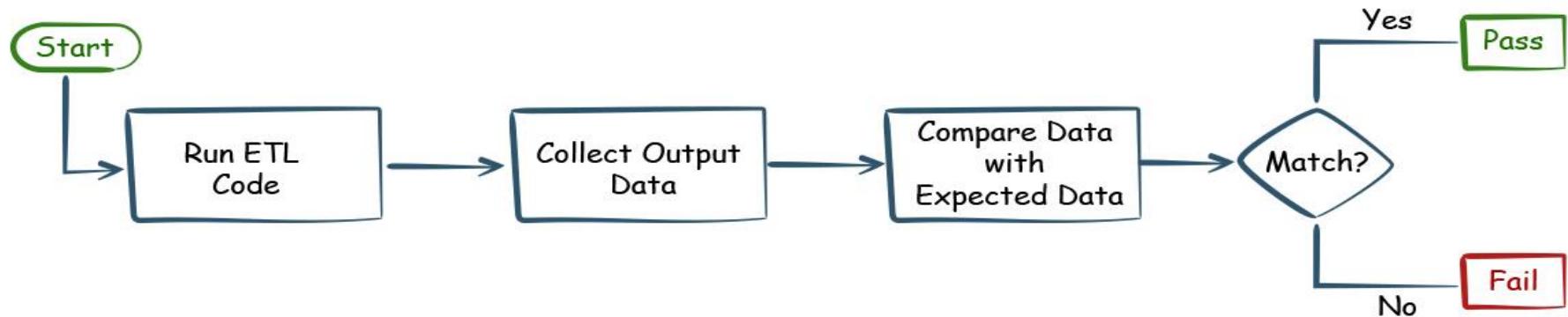
ETL testing is unique since:

- ETL processes are background processes and don't have user screens.
- ETL testing involves a large amount of data.
- ETL processes are like functions where testing requires execution of the ETL process and then the comparison of input and output data.
- The defects in the ETL processes cannot be detected by simply reviewing the ETL code

How to do ETL Testing?

ETL processes are evaluated indirectly through black box testing approach, wherein the ETL process is first executed to create the output data and then by verifying the output data the quality of the ETL process is determined.

ETL testing process is summarized in the following three steps



1. First, the ETL code is executed to generate the output data.
2. Then the output data is compared with the predetermined expected data.
3. Based on the comparison results, the quality of the ETL process is determined.

DQ Industry Tools

Here are few DQ tools available in the industry today:

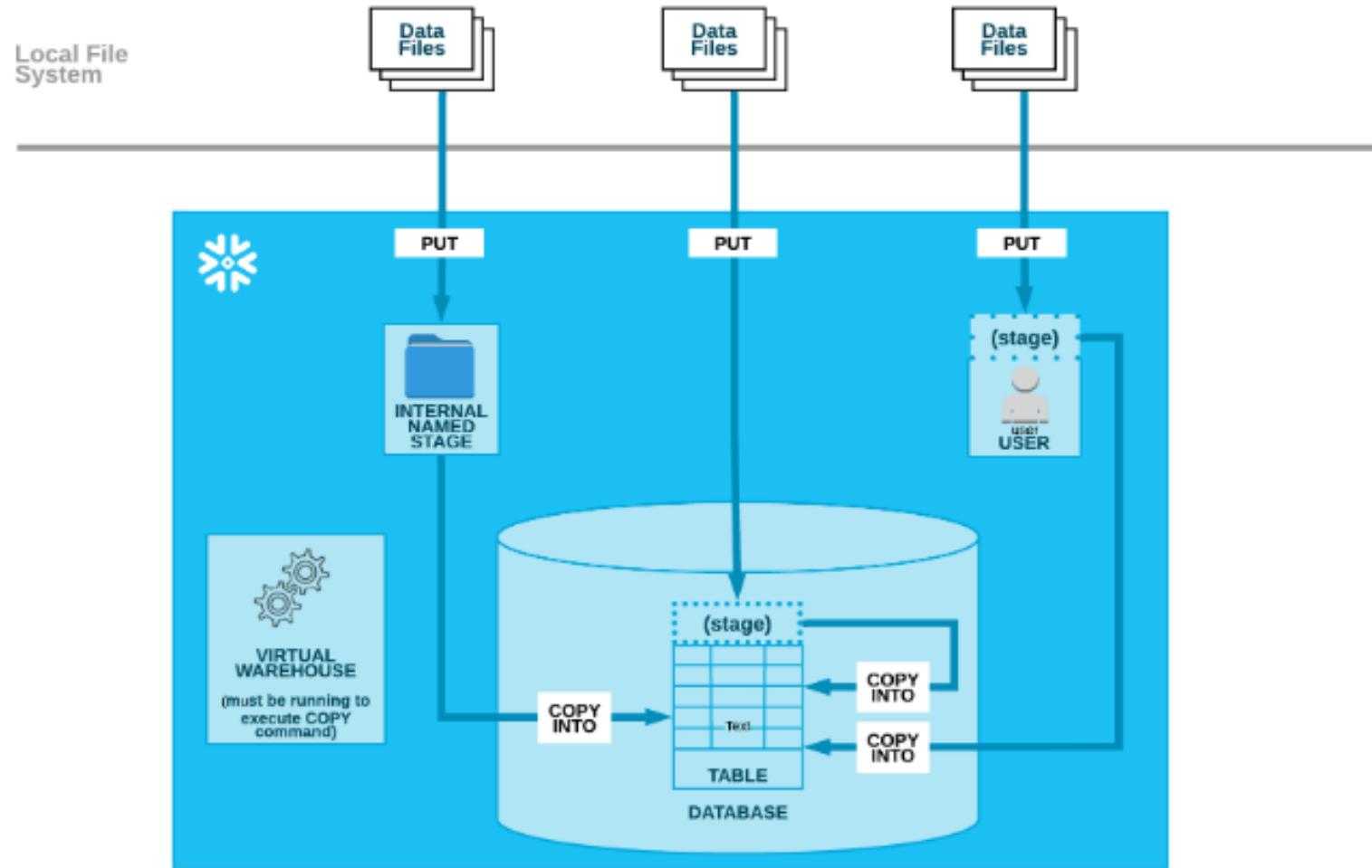
- Informatica Data Quality
- Talend
- Ataccama
- SAP Information Steward
- Collibra

Loading Data into Snowflake

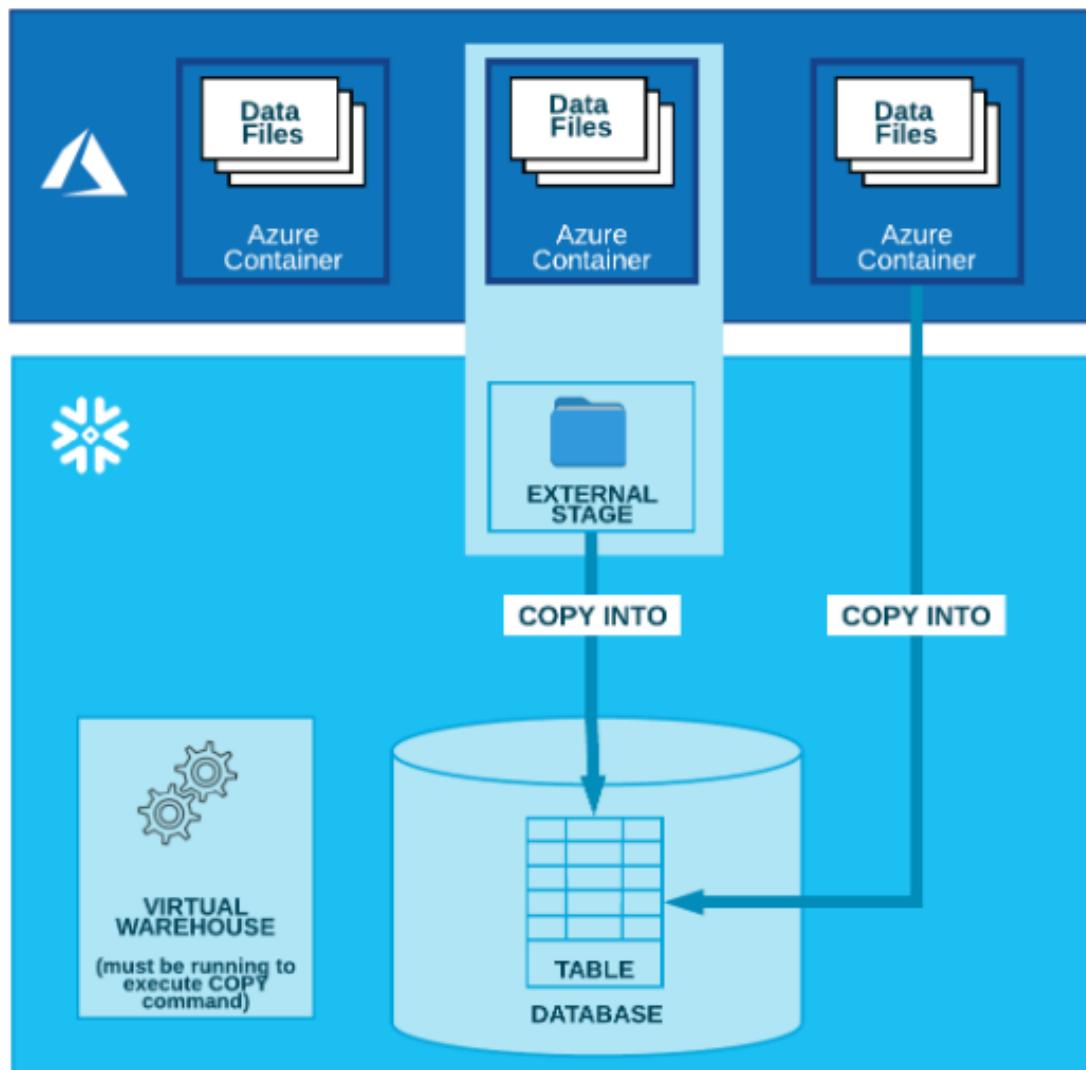
- Snowflake refers to the location of data files in cloud storage as a *stage*. The [COPY INTO <table>](#) command used for both bulk and continuous data loads (i.e. Snowpipe) supports cloud storage accounts managed by your business entity (i.e. *external stages*) as well as cloud storage contained in your Snowflake account (i.e. *internal stages*).
- **External stages:** Loading data from any of the following cloud storage services is supported regardless of the [cloud platform](#) that hosts your Snowflake account: Amazon S3, Google Cloud Storage, Microsoft Azure Data Lake Storage (ADLS)
- **Internal stages** Snowflake maintains the following stage types in your account:
 - **User:** A user stage is allocated to each user for storing files.
 - **Table:** A table stage is available for each table created in Snowflake. This stage type is designed to store files that are staged and managed by one or more users but only loaded into a single table.
 - **Named:** This stage type can store files that are staged and managed by one or more users and loaded into one or more tables.

Upload files to any of the internal stage types from your local file system using the [PUT](#) command.

Snowflake - Data Loading Local File System



Snowflake - Data Loading Cloud Storage (Azure Example)



Loading Data into Snowflake Using Internal Stage

1. Create Snowflake Internal Stage “`CREATE STAGE test_db.schema1.my_int_stage;`”
2. Create a table into which data will be loaded from sample file.
3. Install SnowSQL client
4. Connect to Snowflake using SnowSQL Client “`snowsql -a <account name> -u <username>`”
5. Load a sample file from your laptop to Internal stage via PUT command: `put file://c:/samplefile.csv @my_int_stage;` (this syntax is for Windows client only; For Linux or Mac, refer to documentation: <https://docs.snowflake.com/en/sql-reference/sql/put>)
6. *Load data into the table using the command as below:*

```
copy into <DB_Name>.<Schema_Name>.<Table_Name> from @ <DB_Name>.<Schema_Name>.<Stage_Name>
/<FileName> FILE_FORMAT = ( TYPE = <FORMAT OF FILE>, <additional parameters as needed>);
```

Example:

```
copy into test_db.schema1.st_emp
from @test_db.schema1.my_int_stage/samplefile.csv
FILE_FORMAT = ( TYPE = CSV, skip_header=1);
```

File Format Support by Snowflake

File formats supported: CSV | JSON | AVRO | ORC | PARQUET | XML

Data Loading Options for each file format are available here: [CREATE FILE FORMAT | Snowflake Documentation](#)

Example:

```
CREATE OR REPLACE FILE FORMAT my_csv_format
  TYPE = CSV
  FIELD_DELIMITER = '|'
  SKIP_HEADER = 1
  NULL_IF = ('NULL', 'null')
  EMPTY_FIELD_AS_NULL = true
  COMPRESSION = gzip;
```

Loading Data into Snowflake Using External Stage

1. Create Snowflake Internal Stage “**CREATE STAGE my_azure_stage STORAGE_INTEGRATION = azure_int URL = 'azure://myaccount.blob.core.windows.net/mycontainer/load/files/' FILE_FORMAT = my_csv_format;**”
2. Create a table into which data will be loaded from sample file.
3. *Load data into the table using the command as below:*

```
copy into <DB_Name>.<Schema_Name>.<Table_Name> from @ <DB_Name>.<Schema_Name>.<Stage_Name>
/<FileName> FILE_FORMAT = ( TYPE = <FORMAT OF FILE>, <additional parameters as needed>);
```

Example:

```
copy into test_db.schema1.st_emp
from @test_db.schema1.my_azure_stage/samplefile.csv
FILE_FORMAT = ( TYPE = CSV, skip_header=1);
```



Data Warehouse – OLAP & Multi-Dimensional Databases

Instructor-in-Charge:
Sachin Arora, Guest faculty

BITS Pilani

Pilani Campus

BITS Pilani

Disclaimer

Content in this deck have been curated from multiple online sources, including vendor's documentation

Module 5

Module 5 – OLAP & Multi-Dimensional Databases (MDD)

5.1	Limitations of spreadsheets and SQL
5.2	Major OLAP Features and functions
5.3	Multidimensional databases
5.4	ROLAP, MOLAP, and HOLAP
5.5	OLAP operations using MDDBs

Limitations of Spreadsheets

Data Size and Performance: Spreadsheets may struggle with handling large datasets, leading to performance issues and slower calculations.

Impact: As data warehouses often store massive amounts of data, using spreadsheets for extensive analysis can be impractical and inefficient.

Limited Scalability: Issue: Spreadsheets are typically designed for individual or small team use, and scaling them for enterprise-level data analysis can be challenging.

Impact: In large organizations with complex data structures, spreadsheets may lack the scalability needed for collaborative and comprehensive analytics.

Data Integration Challenges: Integrating data from various sources into a spreadsheet can be cumbersome, requiring manual effort and making it prone to errors.

Impact: Data warehouses often aggregate data from multiple sources, and the complexity of integrating this data into a spreadsheet may result in inaccuracies.

Limited Version Control: Spreadsheets may lack robust version control mechanisms, making it difficult to track changes and maintain data integrity.

Impact: In data warehouse environments where data is frequently updated, version control becomes crucial for maintaining accurate and auditable records.

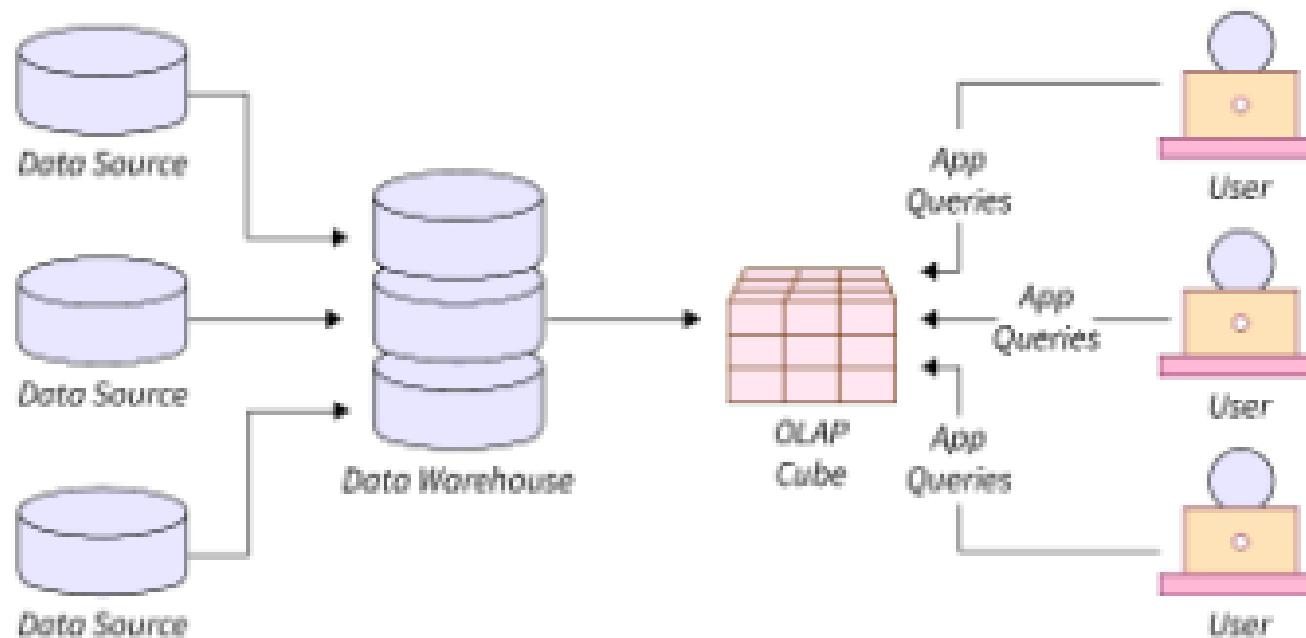
Security Concerns: Spreadsheets may lack sophisticated security features, making it challenging to control access to sensitive data.

Impact: In data warehouses, security is paramount, and spreadsheets may not provide the level of access control and encryption needed to protect critical information.

Limitations of SQL

SQL Limitations	Description	Impact in Data Warehouses
Steep Learning Curve	SQL can have a steep learning curve for non-technical users, making it challenging for widespread adoption.	Limits accessibility for business users and analysts.
Limited Graphical Representation	SQL is primarily a text-based language, making it challenging to visually represent complex relationships and hierarchies.	Hinders understanding of intricate data structures.
Performance Challenges with Complex Queries	Complex SQL queries, especially those involving multiple joins and subqueries, may suffer from performance issues.	Impacts query performance in large and complex data warehouses.
Difficulty in Handling Unstructured Data	SQL is designed for structured data and may struggle with handling unstructured or semi-structured data.	Hampers analysis of diverse data types in modern data warehouses.
Limited Support for Machine Learning and Advanced Analytics	SQL may lack native support for advanced analytics and machine learning, requiring integration with additional tools.	Challenges incorporating advanced analytics techniques.

OLAP



What is OLAP?

- It is a computing method that organizes data into multidimensional structures to facilitate users to get information and analyze data from multiple viewpoints.
- It enables business users to run ad hoc queries, run reports faster, and perform OLAP operations such as slicing, dicing, drill down data to uncover patterns, find relationships between data sets, find anomalies, etc.

OLAP – An Example

Example — A regional company is trying to break into the US market. They are selling four flavors of fruit drinks – mango, cherry, strawberry, and grapes. And they particularly want to know what's going on in four specific markets of the US – Los Angeles, Chicago, New York, Phoenix.

Lots of individuals are buying these fruit drinks in the US. Every purchase is stored in a database of the store from where it was bought. To perform business analysis on this data, they begin by analyzing sales of their products. The below table shows their sales information for the last two quarters of 2020

Time	Sales
Q3	\$20000
Q2	\$20000
Total	\$40000

OLAP – An Example

- Both quarters have the same figures which means that the company has identical sale performance in both quarters.
- Next, they analyze the same sales data from a few other perspectives like sales of individual flavors and the sales according to the region.

Product	Sales	Region	Sales
Mango	\$10000	Los Angeles	\$10000
Cherry	\$10000	Chicago	\$10000
Strawberry	\$10000	New York	\$10000
Grapes	\$10000	Phoenix	\$10000
Total	\$40000	Total	\$40000

OLAP – An Example

- As you can observe, all the three tables show the exact figures – \$40000 for all the three views.
- This means you can be assured that you are looking at the same information – the fruit drink sales of the company with multiple views, each broken into three different categories – time, geography, and product. **These categories are called dimensions.**
- There isn't much to analyze if we look at data this way. Since the data is the same in all three dimensions.

Product	Sales	Region	Sales
Mango	\$10000	Los Angeles	\$10000
Cherry	\$10000	Chicago	\$10000
Strawberry	\$10000	New York	\$10000
Grapes	\$10000	Phoenix	\$10000
Total	\$40000	Total	\$40000

OLAP – An Example

- Let's see what happens if we squeeze the existing three dimensions together to create a single multidimensional view of the data. This is where OLAP comes into the picture.

		Los Angeles	Chicago	New York	Phoenix	Total
Q3	Mango	-	-	\$5500	\$4500	\$10000
	Cherry	-	-	\$3000	\$7000	\$10000
	Strawberry	\$5300	\$4700	-	-	\$10000
	Grapes	-	\$6400	-	\$3600	\$10000
	Total Q3	\$5300	\$11100	\$8500	\$15100	\$40000
<hr/>						
Q4	Mango	\$10000	-	-	-	\$10000
	Cherry	\$6000	-	-	\$4000	\$10000
	Strawberry	-	-	\$5000	\$5000	\$10000
	Grapes	-	\$5680	\$4320	-	\$10000
	Total Q3	\$16000	\$5680	\$9320	\$9000	\$40000
Total		\$21300	\$16780	\$17820	\$24100	\$80000

OLAP – An Example

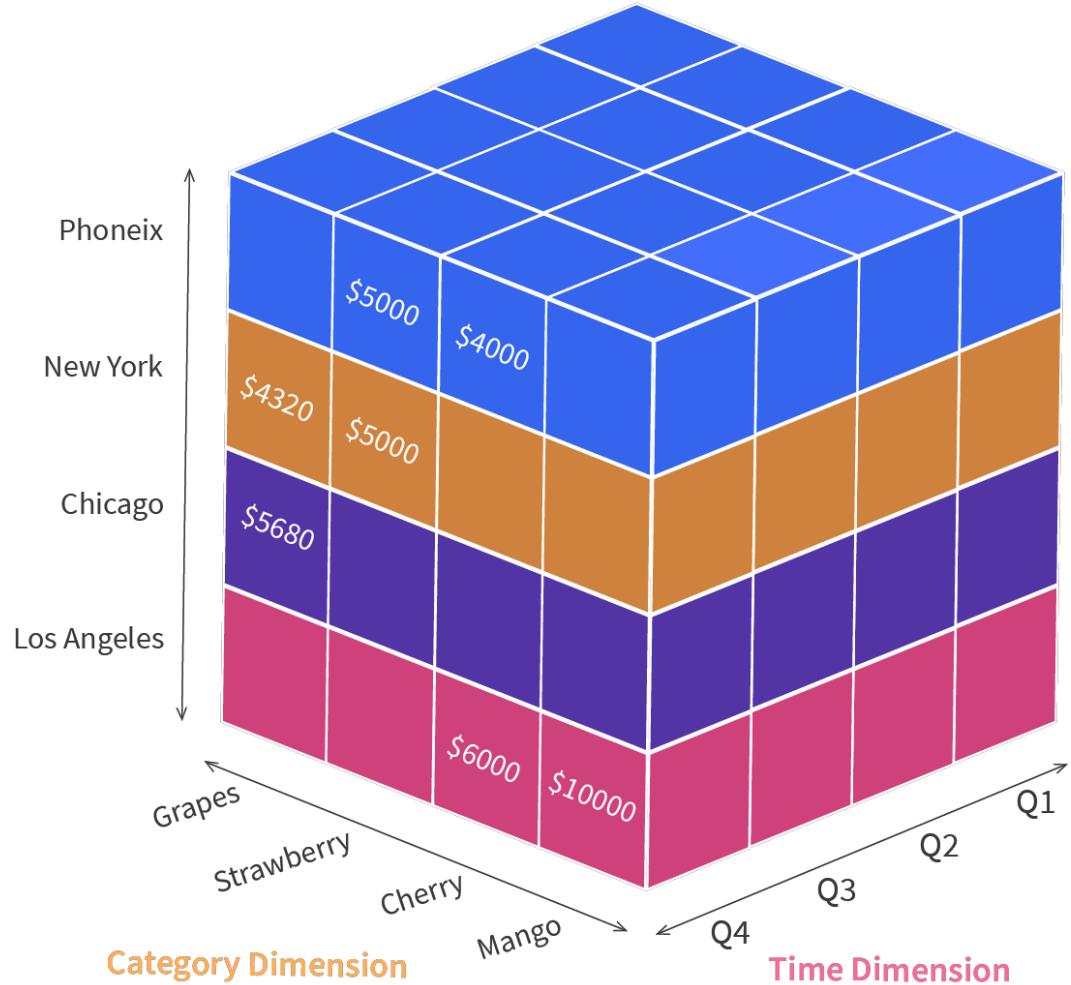
Now, if you observe this table, you will see the patterns and anomalies that lie in your data. For example –

- Mango and cherry flavor fruit drinks don't sell in Los Angeles and Chicago in the Q3.
- Strawberry flavor doesn't sell in New York and Phoenix in the Q3.
- On the other hand, It's vice versa in Q4.
- The grape flavor doesn't sell in Los Angeles in both Q3 & Q4.
- These are some patterns, and you can observe more patterns by interacting with your data from multiple viewpoints.
- This kind of data interaction reveals new and interesting information instead of isolated, single dimension data lists.

		Los Angeles	Chicago	New York	Phoenix	Total
Q3	Mango	-	-	\$5500	\$4500	\$10000
	Cherry	-	-	\$3000	\$7000	\$10000
	Strawberry	\$5300	\$4700	-	-	\$10000
	Grapes	-	\$6400	-	\$3600	\$10000
	Total Q3	\$5300	\$11100	\$8500	\$15100	\$40000
Q4	Mango	\$10000	-	-	-	\$10000
	Cherry	\$6000	-	-	\$4000	\$10000
	Strawberry	-	-	\$5000	\$5000	\$10000
	Grapes	-	\$5680	\$4320	-	\$10000
	Total Q3	\$16000	\$5680	\$9320	\$9000	\$40000
	Total	\$21300	\$16780	\$17820	\$24100	\$80000

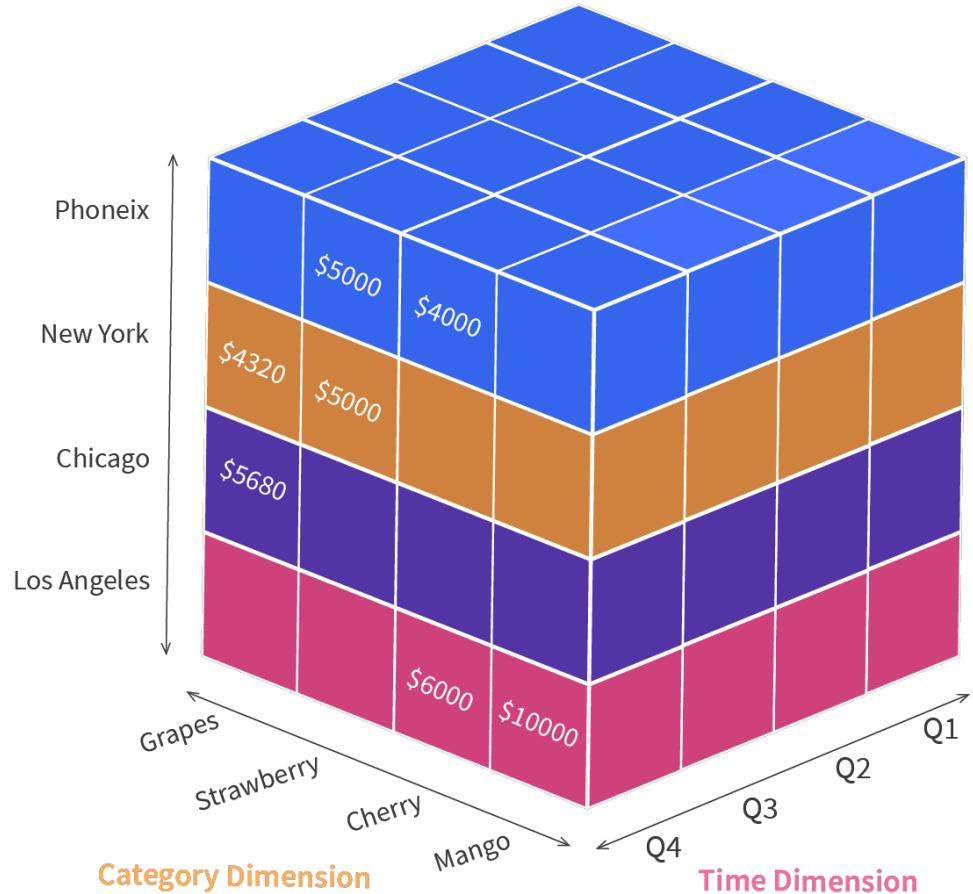
OLAP – An Example

- Diagram shows the pre-aggregated and pre-calculated data stored in an OLAP database or OLAP cube.
- Each cube cell holds a sales value for the intersection of each unique combination of time, geography, and product.**
- Now, users can perform ad-hoc analysis, slice and dice their data by different dimensions, pivot rows, and columns, and drill down through hierarchies to create on-the-fly specialized reports.



OLAP

- Since data is pre-aggregated, the results are retrieved faster, enabling analysts to ask as many questions as they want and gain insights by exploring every interesting pattern and valuable information in a database.
- The above example is an over-simplified, hypothetical use case.
- Real business data is much more complex, and the questions are tough.
- You have to deal with a large number of dimensions, perform heavy calculations on the fly, as well as deal with large datasets



What is OLAP?

- An OLAP cube is a multidimensional data structure that enables quicker and more efficient processing and analysis.
- With a data cube, different operations can be performed to explore and manipulate the data, such as slicing, dicing, drilling down, rolling up and pivoting.
- These operations help answer various business questions in a time-efficient manner.
- OLAP cubes can enable users to discover patterns from data to make informed decisions, hence making it an important tool for business intelligence and data analysis.

OLAP or Multi-Dimension Databases (MDD)

- OLAP, which stands for Online Analytical Processing, is a category of software tools that enables users to interactively analyze multidimensional data stored in a data warehouse.
- Multidimensional databases, also known as OLAP (Online Analytical Processing) databases, organize data into a multidimensional structure called a cube.
- OLAP systems are designed to facilitate complex and ad-hoc querying for business intelligence purposes.
- The primary goal of OLAP is to provide a fast and efficient way for users to explore and analyze large volumes of data from multiple dimensions.
- OLAP cubes have two main purposes - The first is to provide business users with a data model more intuitive to them than a tabular model. This model is called a Dimensional Model; The second purpose is to enable fast query response that is usually difficult to achieve using tabular models.
- Microsoft SQL Server provides a robust OLAP solution known as SQL Server Analysis Services (SSAS). SSAS is a multidimensional and data mining analysis tool that enables users to design, create, and manage multidimensional data models for efficient and interactive data analysis.

Key features of OLAP

Multidimensional Data Model: OLAP systems organize data into a multidimensional model, where data is represented as a cube. Each cell in the cube contains a data value, and the dimensions of the cube represent different aspects or perspectives of the data.

Cubes and Dimensions: A cube is the fundamental unit of OLAP, and it is defined by dimensions. Dimensions are the categorical attributes by which data is analyzed. For example, in a sales data cube, dimensions might include time, geography, products, and customers.

Measures: Measures are the numeric values that users want to analyze. In a sales data cube, a measure might be the total sales revenue. Measures are typically placed at the intersections of dimensions within the cube.

Key features of OLAP

Slicing, Dicing, and Pivoting: OLAP allows users to slice, dice, and pivot the data cube to view it from different perspectives. Slicing involves selecting a single plane of the cube, dicing involves selecting a sub-cube by fixing values for one or more dimensions, and pivoting involves rotating the cube to view it from a different angle.

Aggregation: OLAP systems provide fast and efficient aggregation capabilities, allowing users to view summarized data at different levels of granularity. This is crucial for analyzing data at various hierarchies within dimensions.

Interactive Analysis: OLAP systems support interactive querying, allowing users to explore and analyze data in real-time. Users can drill down into detailed data, roll up to higher levels of aggregation, and perform various analyses on the fly.

What is Data Cube?

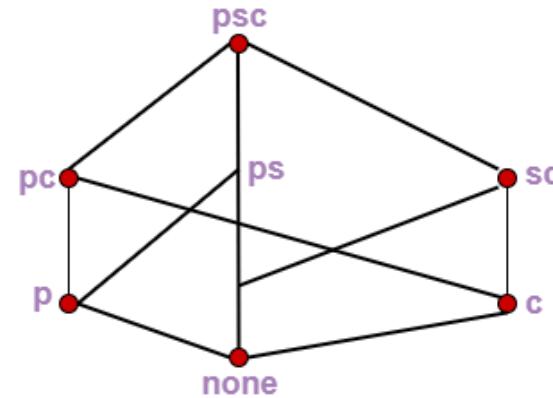
A data cube enables data to be modeled and viewed in multiple dimensions. A multidimensional data model is organized around a central theme, like sales and transactions.

- A data is grouped or combined in multidimensional matrices called Data Cubes.
- General idea of this approach is to materialize certain expensive computations that are frequently required.

What is Data Cube?

For example, a relation with the schema sales (part, supplier, customer, and sale-price) can be materialized into a set of eight views as shown in fig, where psc indicates a view consisting of aggregate function value (such as total-sales) computed by grouping three attributes **part**, **supplier**, and **customer**, p indicates a view composed of the corresponding aggregate function values calculated by grouping part alone, etc.

- A data cube is created from a subset of attributes in the database. Specific attributes are chosen to be measure attributes, i.e., the attributes whose values are of interest.
- Another attributes are selected as dimensions or functional attributes. The measure attributes are aggregated according to the dimensions.



Eight views of data cubes for sales information.

Data Cube

In the 2-D representation, we will look at the All Electronics sales data for items sold per quarter in the city of Vancouver. The measured display in dollars sold (in thousands).

2-D view of Sales Data

location = "Vancouver"					
time (quarter)	item (type)				
	home entertainment	computer	phone	security	
Q1	605	825	14	400	
Q2	680	952	31	512	
Q3	812	1023	30	501	
Q3	927	1038	38	580	

Let suppose we would like to view the sales data with a third dimension.

For example, suppose we would like to view the data according to time, item as well as the location for the

3-D view of Sales Data

location = "Chicago"			location = "New York"			location = "Toronto"		
item			item			item		
home	time	ent.	home	time	comp.	phone	sec.	home
time	ent.	comp.	phone	sec.				ent.
Q1	854	882	89	623	1087	968	38	872
Q2	943	890	64	698	1130	1024	41	925
Q3	1032	924	59	789	1034	1048	45	1002
Q4	1129	992	63	870	1142	1091	54	984
					818	746	43	591
					894	769	52	682
					940	795	58	728
					978	864	59	784

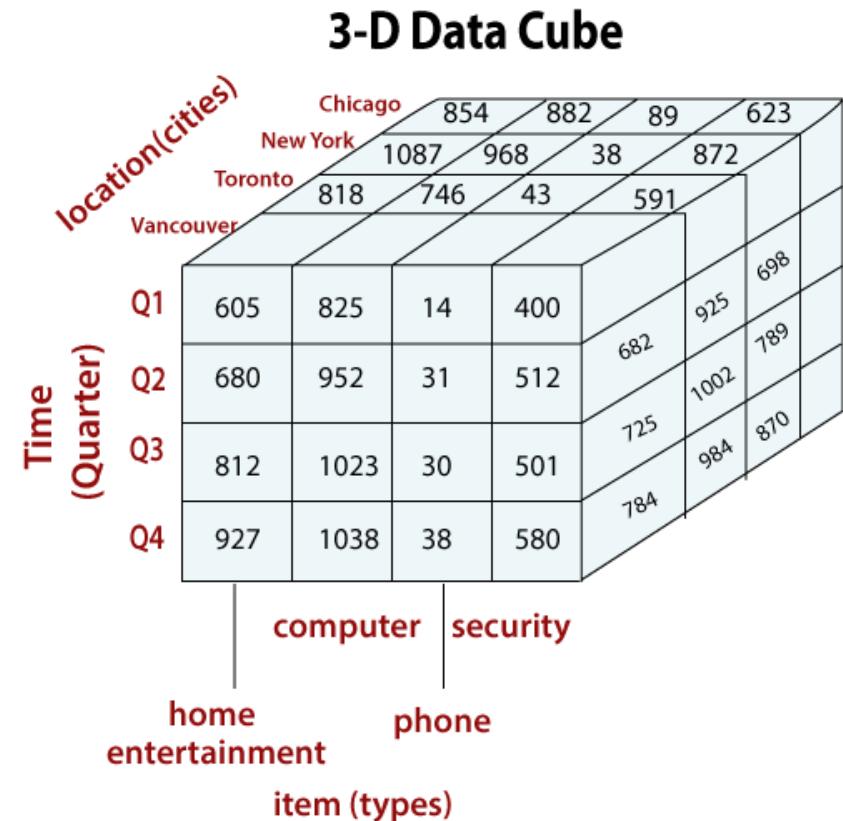
Data Cube

Let suppose we would like to view the sales data with a third dimension.

For example, suppose we would like to view the data according to time, item as well as the location for the cities Chicago, New York, Toronto, and Vancouver.

3-D view of Sales Data

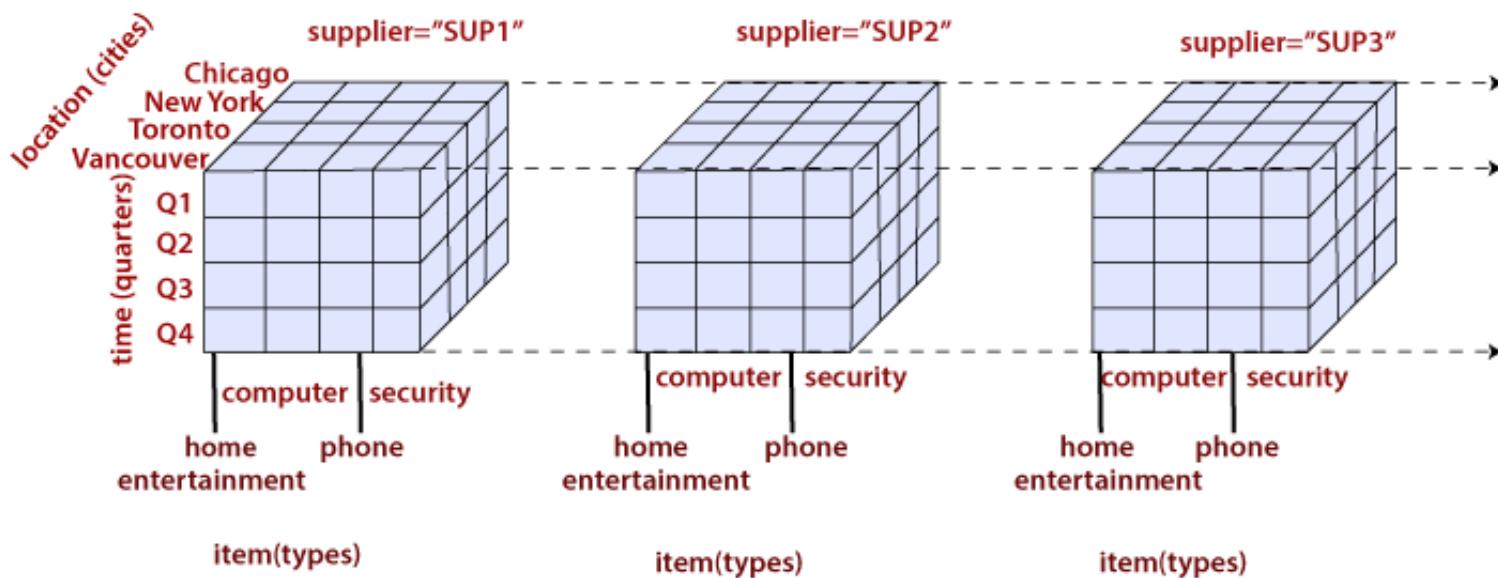
location = "Chicago"		location = "New York"		location = "Toronto"	
item		item		item	
home		home		home	
time	ent.	comp.	phone	sec.	
Q1	854	882	89	623	854
Q2	943	890	64	698	1087
Q3	1032	924	59	789	1130
Q4	1129	992	63	870	1048



Data Cube

In data warehousing, the data cubes are n-dimensional. The cuboid which holds the lowest level of summarization is called a base cuboid.

For example, the 4-D cuboid in the figure is the base cuboid for the given time, item, location, and supplier dimensions.



Data Cube – An Example

- Consider a retail company based in the United States that offers 4 types of clothing – t-shirts, shirts, jeans, and jackets.
- The in-house data engineers have gathered information about last year's revenue (displayed as a one dimensional array).
- The company, however, has stores in 4 states – California, Hawaii, Indiana, and Michigan.
- Therefore, the revenue can be split further by products and by states, resulting in a 2D table:
- Sum of the numbers along each of the columns gives the total revenue by product.
- We can sum along the rows as well, such as the annual revenue coming from California.

Products —→

T-Shirts	Shirts	Jeans	Jackets
\$12 000	\$10 500	\$16 300	\$15 200

Data Cube – An Example

- What if we wanted to split this revenue by date? For simplicity, we'll divide the year into quarters. One way to store the information for all 4 is by constructing separate tables for each:

Company Revenue by Products and States

Products →

States ↓

	T-Shirts	Shirts	Jeans	Jackets
California	\$4 100	\$2 300	\$4 700	\$1 300
Hawaii	\$2 900	\$4 600	\$3 200	\$3 700
Indiana	\$1 500	\$1 200	\$5 100	\$4 900
Michigan	\$3 500	\$2 400	\$3 300	\$5 300

Data Cube – An Example

Products →

States ↓

Q1	T-Shirts	Shirts	Jeans	Jackets
California	\$1 200	\$500	\$900	\$300
Hawaii	\$600	\$1 200	\$750	\$700
Indiana	\$600	\$650	\$800	\$700
Michigan	\$900	\$750	\$550	\$700

Products →

States ↓

Q2	T-Shirts	Shirts	Jeans	Jackets
California	\$1 100	\$400	\$1 200	\$400
Hawaii	\$550	\$1 100	\$820	\$600
Indiana	\$550	\$350	\$900	\$850
Michigan	\$1 000	\$850	\$450	\$900

Products →

States ↓

Q3	T-Shirts	Shirts	Jeans	Jackets
California	\$800	\$620	\$1 100	\$250
Hawaii	\$1 000	\$980	\$900	\$650
Indiana	\$200	\$100	\$1 100	\$750
Michigan	\$1 200	\$650	\$1 500	\$1 200

Products →

States ↓

Q4	T-Shirts	Shirts	Jeans	Jackets
California	\$1 000	\$780	\$1 500	\$350
Hawaii	\$750	\$1 320	\$730	\$1 750
Indiana	\$150	\$100	\$2 300	\$2 600
Michigan	\$400	\$150	\$800	\$2 500

Data Cube – An Example

As an alternative, we can create a third dimension called ‘Date’ by stacking all 4 tables

There is now one more dimension that we can sum over. As an example, we’ll calculate the revenue for a full year of t-shirt purchases in Michigan:

Similar to the ‘ALL’ fields for ‘Products’ and ‘States’, we can calculate the total revenue by ‘Date’ as well. In this way, all 3 data cube dimensions will meet in a single data cell, storing the total revenue for a full year.

Company Revenue Represented as a Data Cube

Products →

↓ States

Q1	T-Shirts	Shirts	Jeans	Jackets
California	\$1 200	\$500	\$900	\$300
Hawaii	\$600	\$1 200	\$750	\$700
Indiana	\$600	\$650	\$800	\$700
Michigan	\$900	\$750	\$550	\$700
Michigan	\$1 000	\$850	\$450	\$900
Michigan	\$1 200	\$650	\$1 500	\$1 200
Michigan	\$400	\$150	\$800	\$2 500

$$\$900 + \$1,000 + \$1,200 + \$400 = \$3,500$$

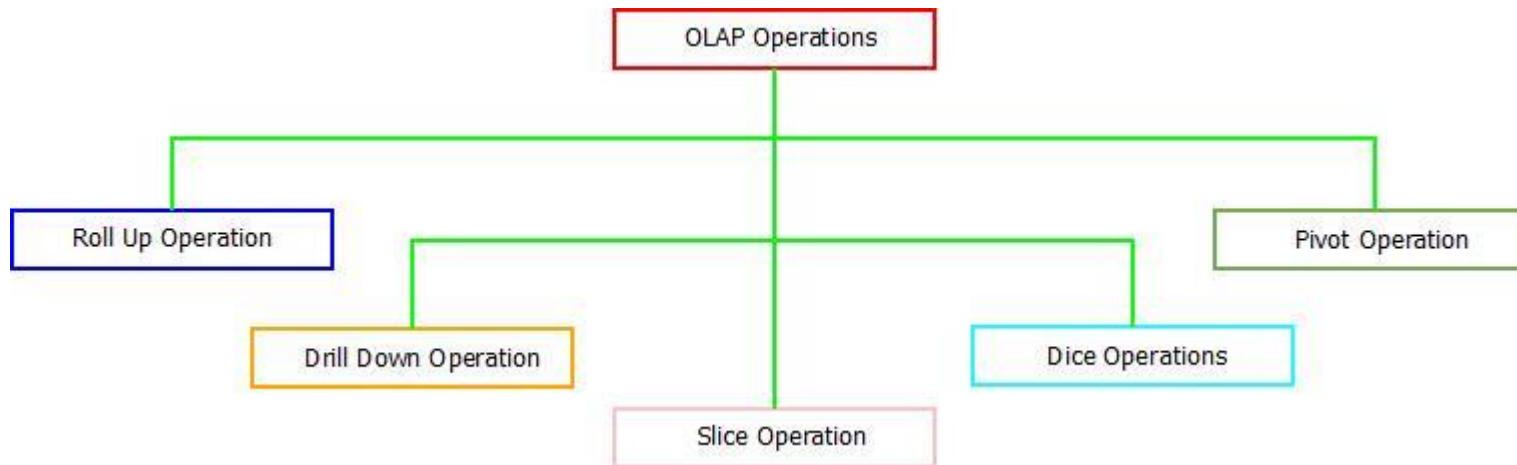
Elements of Data Cube

- A data cube is a multi-dimensional data structure.
- A data cube is characterized by its dimensions (e.g., Products, States, Date).
- Each dimension is associated with corresponding attributes (for example, the attributes of the Products dimension are T-Shirt, Shirt, Jeans and Jackets).
- The dimensions of a cube allow for a concept hierarchy (e.g., the T-shirt attribute in the Products dimension can have its own, such as T-shirt Brands).
- All dimensions connect in order to create a certain fact – the finest part of the cube.
- A fact has a corresponding measure in the data cube. Typically, the fact measure in a data cube for a chain retail business is the revenue (such as the \$900 revenue from jeans purchases in Indiana during the second quarter).

What are the Data Cube Operations?

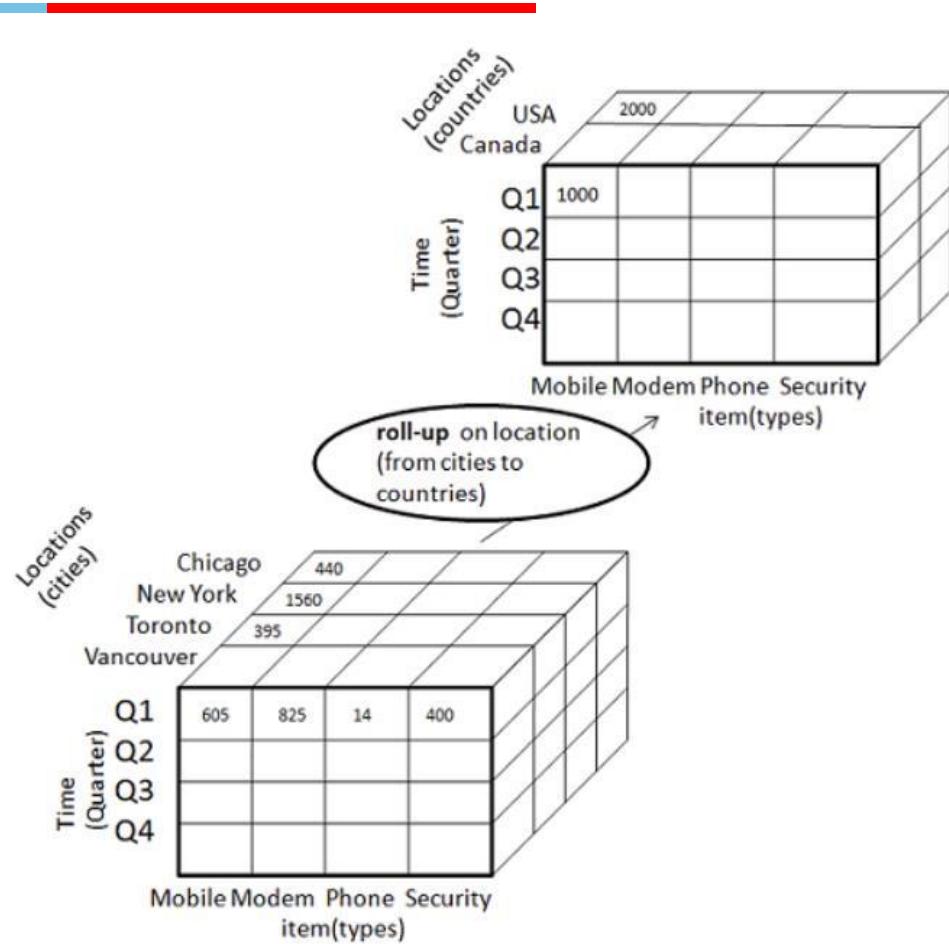
Data cubes are a very convenient tool whenever one needs to build summaries or extract certain portions of the entire dataset.

- Rollup – decreases dimensionality by aggregating data along a certain dimension
- Drill-down – increases dimensionality by splitting the data further
- Slicing – decreases dimensionality by choosing a single value from a particular dimension
- Dicing – picks a subset of values from each dimension
- Pivoting – rotates the data cube



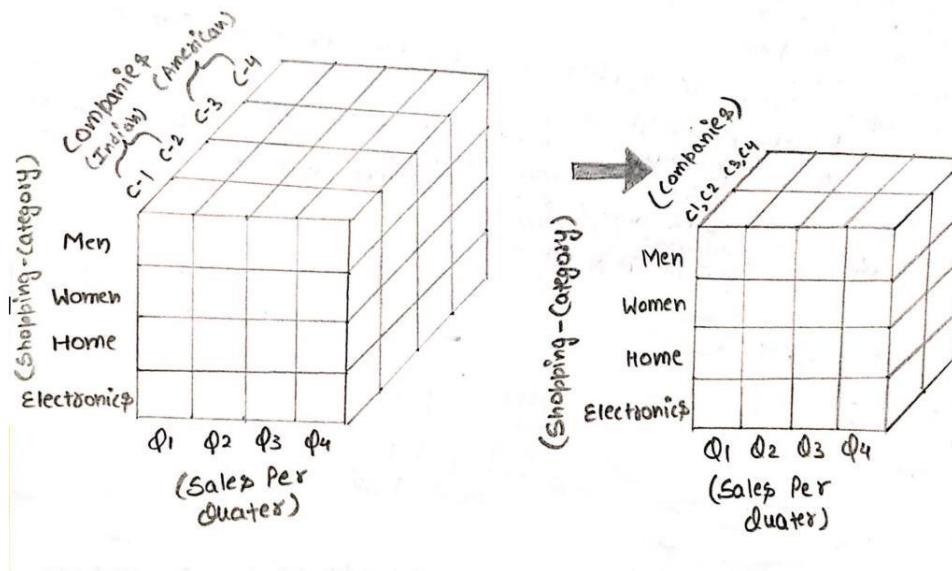
Rollup

- An important feature of data cubes is aggregating data, which results in dimensionality reduction.
- We do this by grouping the values across a certain dimension, either by taking their sum, average, standard deviation, or by applying any other *aggregate function*.
- This dimensionality reduction procedure is known as a *rollup*.
- Imagine it as a zooming out on a data cube.
- Retail example: summing across the ‘Date’ dimension in the 3D cube returns the 2D table displaying only ‘Products’ and ‘States’.
- Example: **GROUP BY** clause.



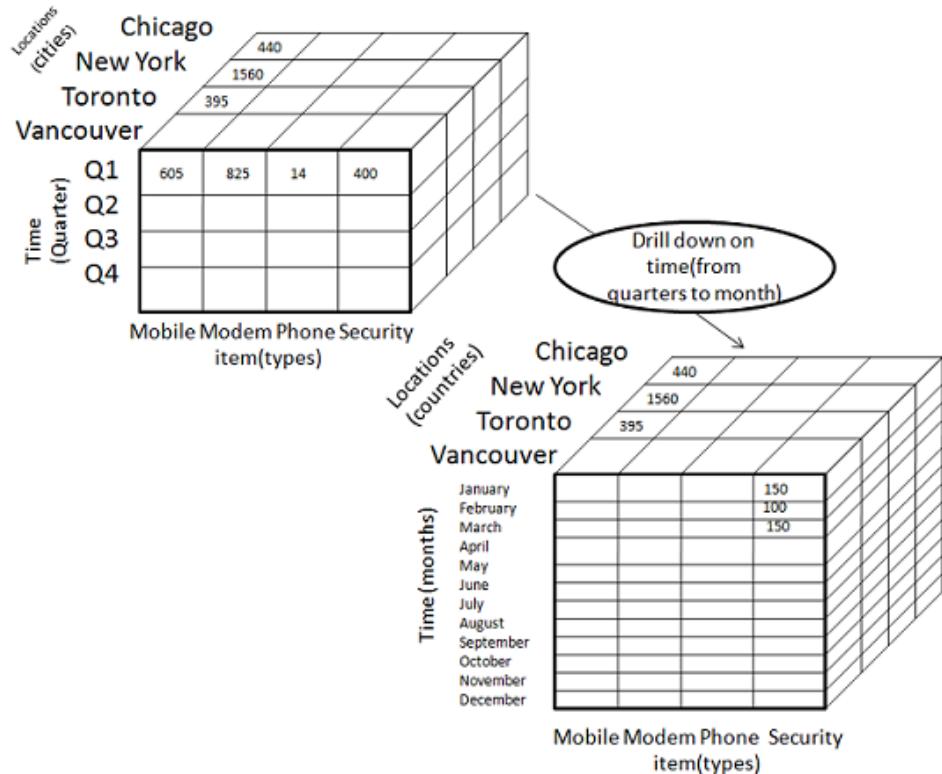
Rollup

Consider the example of sales of four companies C1, C2, C3 & C4 per quarter on the basis of product category(Men's, Women's, Electronics &Home). Out of the four companies, two companies are from India(C1 & C2) and two are from America(C3 & C4). So, if we want to perform the Roll-Up operation on the given data cube, we can do it by combining Indian companies together and American.



Drill-Down

- In contrast, a *drill-down* of a data cube refers to dimensionality increase.
- In laymen terms, this would be similar to zooming in on a cube.
- We split the revenue not only by products, but also by states; or make a split not only by products, but also by product brands.
- By stepping down a concept hierarchy for a dimension.
- By introducing a new dimension.

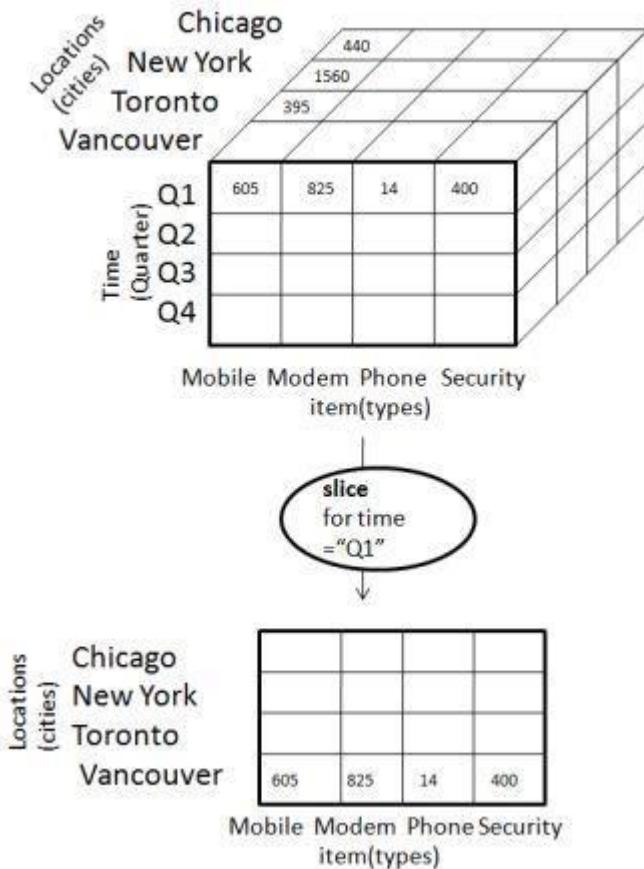


Slice & Dicing

- To slice and dice is to break a body of information down into smaller parts or to examine it from different viewpoints so that you can understand it better.
- Slice and Dice Analysis, also known as multidimensional analysis, is a method used to analyze data from various angles or dimensions. It involves breaking down a large dataset into smaller subsets, or "slices," based on specific criteria and then examining each slice to gain insights.
- To slice means to cut and to dice means to cut into very small uniform sections and the two actions are often performed sequentially. For example, a chef may first cut an onion into slices and then cut the slices up into dices
- In data analysis, the term generally implies a systematic reduction of a body of data into smaller parts or views that will yield more information. The term is also used to mean the presentation of information in a variety of different and useful ways.
- Pivot Tables are a popular self-service BI tool for slicing and dicing data. Essentially, a pivot table sorts, counts and totals the data stored in one database table or spreadsheet and creates a second table – the actual pivot table – that summarizes the data.
- Slice and dice contrasts with the terms drill down, drill across and roll up. To drill down is to look at more detailed data in progressively deeper levels of a body of information's hierarchy. To drill across is to compare data in similar levels of a body of information's hierarchy, and to roll up is to aggregate data by removing detail levels from the hierarchy.

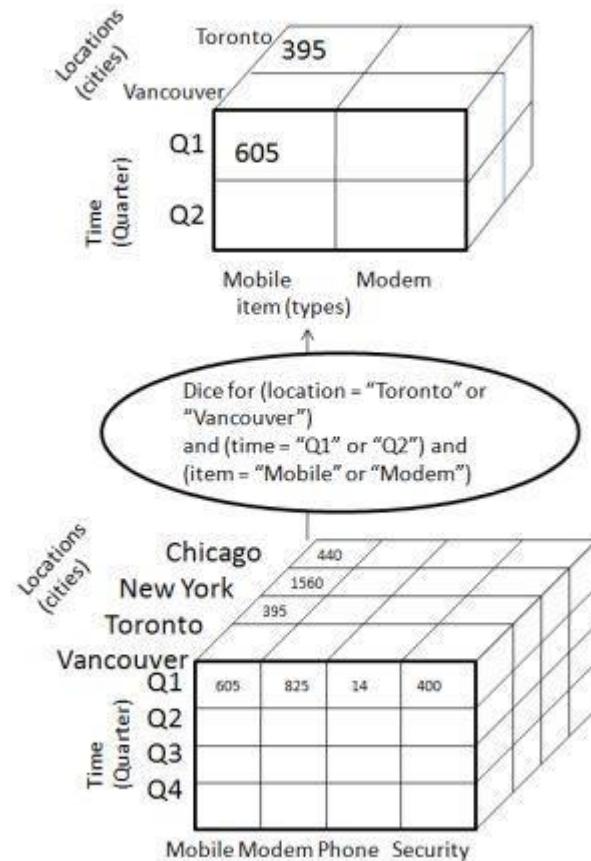
Slice

- Data cube can also be *sliced*.
- Reduce the number of dimensions by keeping one of them fixed.
- For example, we can slice our 3D data cube and obtain a two-dimensional one by choosing to study the revenue by products and states only for the first quarter of the year.



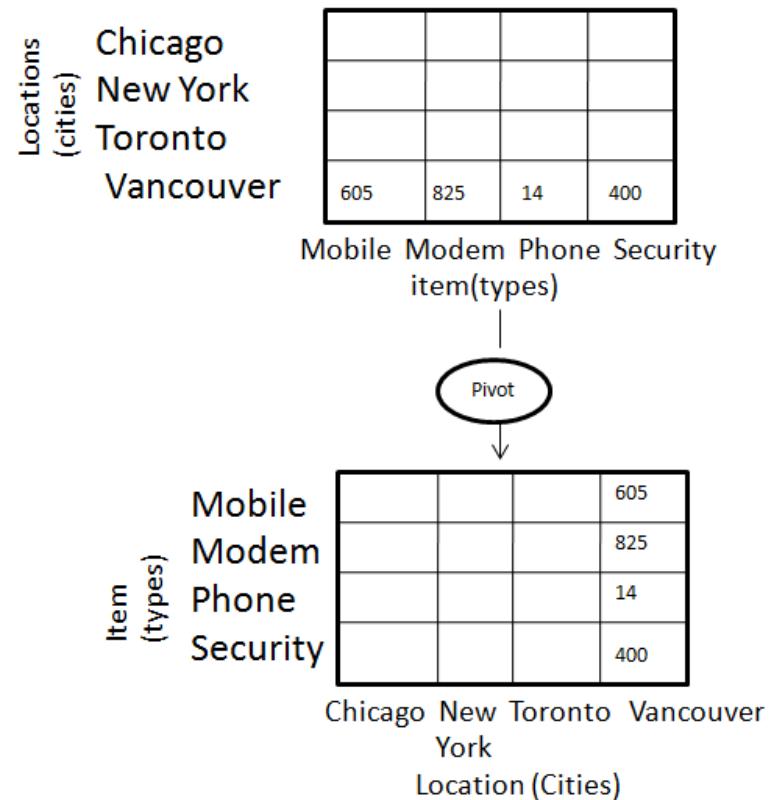
Dice

- Dice selects two or more dimensions from a given cube and provides a new sub-cube
- Rather than extracting a slice from a cube, we can also create a die through *dicing*.
- This is done by picking certain values from the dimensions.



Pivoting

- The pivot operation is also known as rotation. It rotates the data axes in view in order to provide an alternative presentation of data.
- Pivoting is when we rotate a data cube, allowing for a dimension to be displayed either along its length, width, or depth.



Examples of MDD

OLAP (Online Analytical Processing):

Example: Microsoft SQL Server Analysis Services (SSAS), Oracle OLAP, IBM Cognos TM1.

Characteristics: OLAP databases are optimized for complex queries and aggregations. They support operations like roll-up, drill-down, slicing, and dicing.

OLAP Guidelines

Dr E.F. Codd, the "father" of the relational model, has formulated a list of 12 guidelines and requirements as the basis for selecting OLAP systems

Information Principle:

Rule:

- All information in the database should be accessible through the multidimensional structure.
- It emphasizes that the data in the OLAP system should be organized in a way that allows users to access any piece of information through the dimensions of the data cube.
- The focus is on the accessibility and availability of information, ensuring that the multidimensional structure provides a comprehensive and integrated view of the data.

Example: In a sales data cube, all relevant information about sales, such as revenue, quantity sold, and profit margin, should be accessible through the dimensions like Time, Product, and Region.

OLAP Operations Rule

Rule: Emphasizes that OLAP operations should be inherently multidimensional, allowing users to analyze and manipulate data across multiple dimensions.

Example: Users should be able to perform operations like slicing, dicing, drilling down, and rolling up on a sales data cube to analyze sales performance from different perspectives.

OLAP Guidelines

Dynamic, Intuitive and High-Performance Response:

Rule: OLAP systems should provide dynamic and high-performance responses to user queries.

Example: Users should experience fast response times when exploring and analyzing large volumes of data, ensuring a smooth interactive experience

Comprehensive Data Independence:

Rule: The end-user should be shielded from the details of how data is stored and retrieved.

Example: Users should be able to query and analyze data without needing to know the underlying database schema or implementation details.

Support for Complex Objects and Relationships:

Rule: OLAP systems should support complex objects and relationships beyond simple data relationships.

Example: If a sales cube includes complex objects like sales promotions or discounts, the OLAP system should support the analysis of these complex relationships.

OLAP Guidelines

Multi-User Support:

Rule: OLAP systems should support multiple users accessing and manipulating the data simultaneously.

Example: Different users within an organization should be able to run concurrent queries and analyses on the same OLAP database.

Unlimited Dimensions and Aggregation Levels:

Rule: There should be no limit to the number of dimensions or levels of aggregation in an OLAP system.

Example: An OLAP system should support cubes with various dimensions like Time, Geography, Product, and more, with flexibility in the level of granularity for each.

Flexible Dimensionality:

Rule: Users should be able to define new dimensions and hierarchies easily.

Example: If a business wants to introduce a new category for products, the OLAP system should allow for the easy addition of this new dimension without major disruptions.

OLAP Guidelines

Handling of Missing Information:

Rule: OLAP systems should handle missing or incomplete information gracefully.

Example: If certain data points are missing, the OLAP system should still allow users to perform analyses, filling in missing values through interpolation or other methods.

User-Defined Calculations:

Rule: Users should be able to define custom calculations and formulas.

Example: Users may want to create custom measures such as profit margin percentage, which can be defined based on existing measures like revenue and cost.

Support for Non-Additive Aggregation:

Rule: OLAP systems should support non-additive aggregation functions.

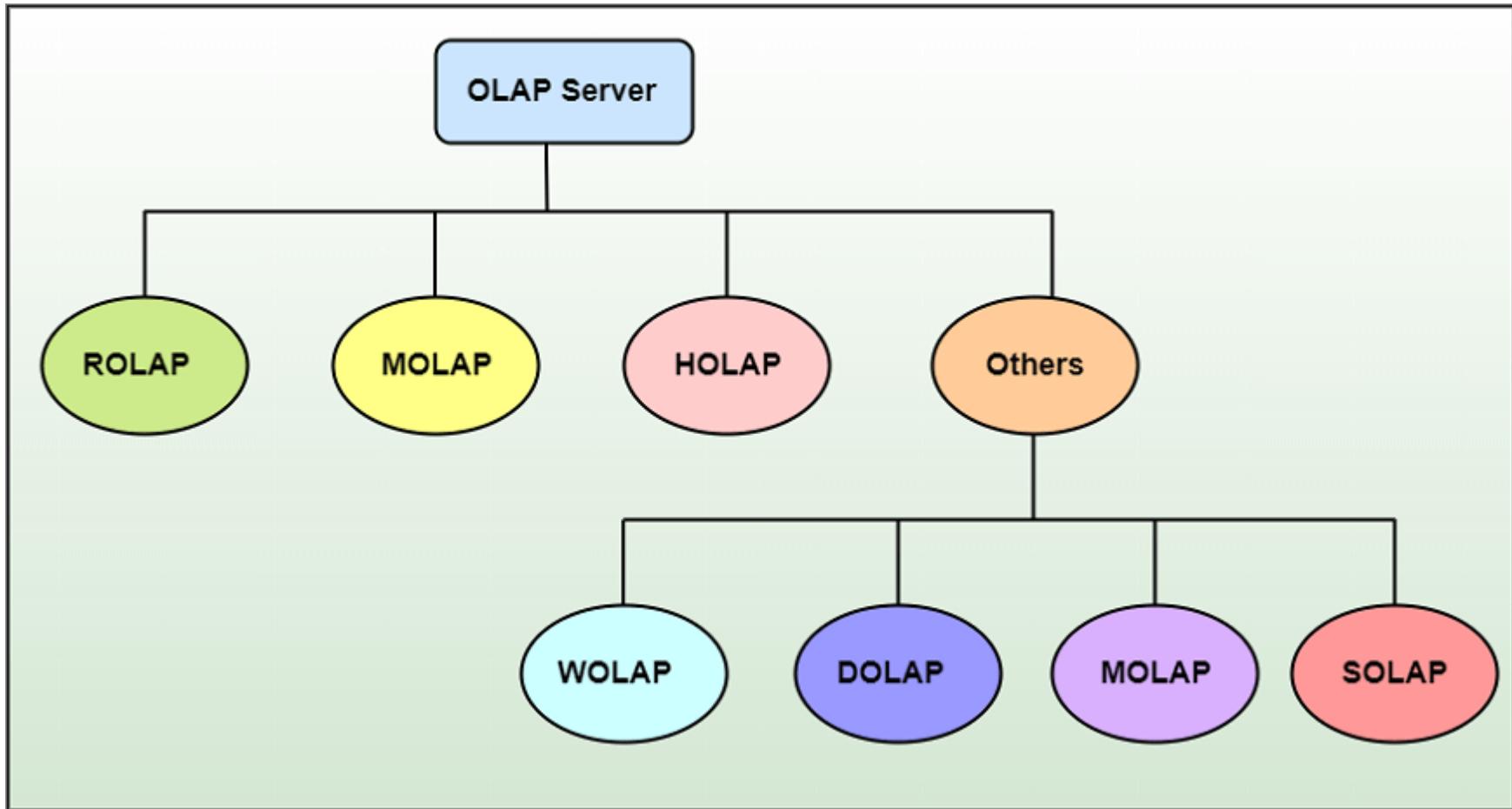
Example: Non-additive aggregations include calculations like average or percentage, which may not follow traditional summation rules.

Reliable Replication:

Rule: OLAP systems should support reliable replication of data for distributed databases.

Example: In a scenario where OLAP data is distributed across multiple servers, the system should ensure that the replicated data is consistent and reliable for analysis

Types of OLAP



Types of OLAP

- ROLAP stands for Relational OLAP, an application based on relational DBMSs.
- MOLAP stands for Multidimensional OLAP, an application based on multidimensional DBMSs.
- HOLAP stands for Hybrid OLAP, an application using both relational and multidimensional techniques.

Relational OLAP (ROLAP) Server

- Relational On-Line Analytical Processing (ROLAP) is primarily used for data stored in a relational database, where both the base data and dimension tables are stored as relational tables.
- These are intermediate servers which stand in between a relational back-end server and user frontend tools.
- This method allows multiple multidimensional views of two-dimensional relational tables to be created, avoiding structuring record around the desired view.
- They use a relational or extended-relational DBMS to save and handle warehouse data, and OLAP middleware to provide missing pieces.
- ROLAP servers contain optimization for each DBMS back end, implementation of aggregation navigation logic, and additional tools and services.
- ROLAP technology tends to have higher scalability than MOLAP technology.
- This technique relies on manipulating the data stored in the relational database to give the presence of traditional OLAP's slicing and dicing functionality.
- In essence, each method of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement

Relational OLAP (ROLAP) Server

Role: Central to ROLAP architecture is the underlying Relational Database Management System (RDBMS), such as Oracle, SQL Server, or MySQL.

Functionality:

- Stores both detailed and aggregated data in relational tables.
- Supports SQL queries for data retrieval and manipulation.

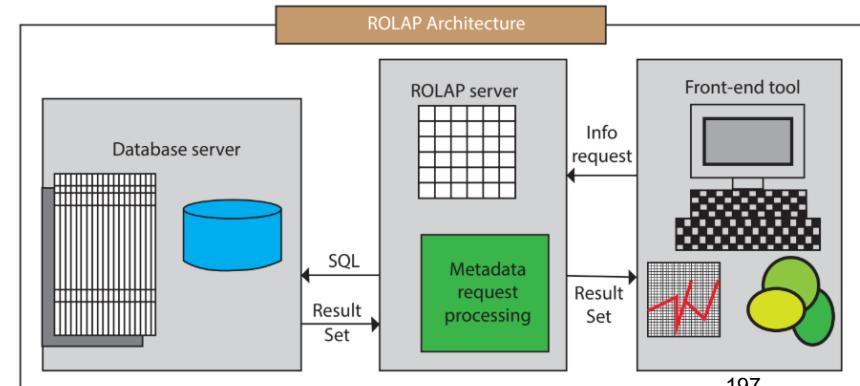
Star or Snowflake Schema: ROLAP systems typically use a star schema or a snowflake schema to organize data within the relational database.

ETL Processes (Extract, Transform, Load):

Aggregation Management: ROLAP systems often perform aggregations to pre-calculate and store summarized data at different levels of granularity.

Functionality:

- Aggregated data is stored in the fact table or separate aggregate tables.
- Improves query performance for frequently accessed data



Relational OLAP (ROLAP) – Architecture

SQL Query Interface: Users interact with ROLAP systems through SQL (Structured Query Language) queries. Functionality:

- SQL queries are used for data retrieval, analysis, and reporting.
- Supports complex queries involving joins, aggregations, and filtering.

OLAP Engine: The OLAP engine is responsible for processing analytical queries and translating them into SQL queries that can be executed against the relational database. Functionality:

- Dynamically generates SQL queries based on user requests.
- Optimizes query performance by leveraging indexes and aggregations.

Client Tools: Users interact with ROLAP systems through various client tools, such as reporting tools, dashboards, and BI applications. Functionality:

- Provides a user-friendly interface for data exploration and visualization.
- Supports the creation of custom reports and dashboards.

Security and Access Control: ROLAP systems implement security measures to control user access to sensitive data. Functionality:

- Role-based access control (RBAC) to restrict data access.
- Encryption and authentication mechanisms for data security.

Relational OLAP (ROLAP)

Advantages

- Can handle large amounts of information: The data size limitation of ROLAP technology depends on the data size of the underlying RDBMS. So, ROLAP itself does not restrict the data amount.
- RDBMS already comes with a lot of features. So ROLAP technologies, (works on top of the RDBMS) can control these functionalities.

Disadvantages

- Performance can be slow: Each ROLAP report is a SQL query (or multiple SQL queries) in the relational database, the query time can be prolonged if the underlying data size is large.
- Limited by SQL functionalities: ROLAP technology relies on upon developing SQL statements to query the relational database, and SQL statements do not suit all needs.

Relational OLAP (ROLAP) – Key Characteristics

Characteristic/Feature	Description
Storage Approach	Stores data in traditional relational databases (e.g., SQL Server, Oracle, MySQL).
Aggregation Method	Aggregations are calculated dynamically at query time using SQL queries.
Query Processing	Capable of handling complex queries involving multiple joins, filtering conditions, and aggregations.
Scalability	Can scale well with large datasets, leveraging the scalability features of relational databases.
Flexibility in Data Modeling	Allows for a flexible data modeling approach, accommodating changes in data structures and relationships.
Integration with BI Tools	Integrates seamlessly with various Business Intelligence (BI) tools for reporting and visualization.
Examples	<ul style="list-style-type: none"> - Microsoft SQL Server Analysis Services (ROLAP mode): Part of the Microsoft SQL Server BI stack. Allows ROLAP-style multidimensional analysis. - Oracle OLAP Option: Integrates with Oracle Database to provide ROLAP capabilities.
Storage Efficiency	May be less storage-efficient compared to MOLAP systems, as data is stored in a normalized form in relational tables.
Indexing and Compression	Can leverage indexing and compression techniques to optimize storage and query performance.

Multi-Dimensional OLAP

- MOLAP systems are designed to store and process data in a multidimensional cube structure, providing an efficient and intuitive way to analyze and explore complex datasets.
- This technology is particularly well-suited for scenarios where data has a natural hierarchical structure and requires quick and interactive querying.
- Data is pre-computed, re-summarized, and stored in a MOLAP (a major difference from ROLAP).
- Using a MOLAP, a user can use multidimensional view data with different facets.
- MOLAP has all possible combinations of data already stored in a multidimensional array. MOLAP can access this data directly.
- MOLAP architectures are optimized for fast query performance and interactive data exploration.

Multi-Dimensional OLAP – Architecture

MOLAP architecture mainly reads the precompiled data. MOLAP architecture has limited capabilities to dynamically create aggregations or to calculate results that have not been pre-calculated and stored.

Multidimensional Cube: The central concept in MOLAP architecture is the multidimensional cube, which organizes data along multiple dimensions.

Structure:

- Each cell in the cube contains a data value or a measure.
- Dimensions represent various aspects of the data, such as time, geography, or product categories.
- Hierarchies within dimensions provide different levels of granularity.

Cube Storage: The multidimensional cube is stored in memory, allowing for fast access and retrieval of data during analytical queries.

In-Memory Storage:

- Data is pre-aggregated and stored in memory for quick access.
- Aggregations are calculated for different levels and combinations of dimensions.

Multi-Dimensional OLAP – Architecture

Aggregation Levels: MOLAP systems pre-calculate aggregations at various levels of granularity, optimizing query performance.

Functionality:

- Aggregated data is stored for dimensions at different hierarchy levels.
- Allows for quick retrieval of summarized data during queries.

OLAP Engine: The OLAP engine is a critical component that manages and processes queries against the multidimensional cube.

Functionality:

- Translates user queries into efficient operations on the cube.
- Utilizes pre-calculated aggregations to respond to queries rapidly.

Dimension Hierarchies: Dimensions are organized in hierarchies, providing a structured way to view data at different levels.

Functionality:

- Users can drill down into more detailed levels or roll up to higher levels in the hierarchy.
- Hierarchies enable users to navigate through data intuitively.

MDX (Multidimensional Expressions): MDX is a query language specifically designed for querying multidimensional data structures.

Functionality:

- Allows users to express complex analytical queries against the multidimensional cube.
- Supports slicing, dicing, drilling down, and rolling up operations.

Multi-Dimensional OLAP – Architecture

Data Storage: Data is stored in a compressed and optimized format within the multidimensional cube.

Functionality:

- Efficient storage allows for quick retrieval of data during analytical queries.
- Data is organized to support multidimensional analysis.

Offline Analysis: Some MOLAP systems provide offline capabilities, allowing users to work with cube data even when disconnected from the network.

Functionality:

- Users can perform analysis and make decisions using cached cube data.

MOLAP architectures are particularly effective in scenarios where users require fast and interactive analysis of multidimensional data. The in-memory storage and pre-aggregated nature of MOLAP systems contribute to their high performance in handling complex analytical queries.

Implementation Considerations in MOLAP

- Selection of the right business dimensions
- Selection of the right filters for loading the data from the data warehouse
- Methods and techniques for moving data into the OLAP system (MOLAP model)
- Choosing the aggregation, summarization, and pre-calculation
- Developing application programs using the proprietary software of the OLAP vendor
- Size of the multidimensional database
- Handling of the sparse-matrix feature of multidimensional structures
- Drill down to the lowest level of detail
- Drill through to the data warehouse or to the source systems
- Drill across among OLAP system instances
- Access and security privileges
- Backup and restore capabilities

MOLAP - Examples

Microsoft SQL Server Analysis Services (SSAS):

Vendor: Microsoft

Description:

- SSAS is a powerful MOLAP solution integrated with the Microsoft SQL Server database.
- It allows users to create multidimensional cubes, define measures, and perform advanced analytics using MDX (Multidimensional Expressions).

Key Features:

- Cube design and processing.
- Support for calculated members and KPIs.
- Integration with Microsoft Excel for data analysis.

MOLAP - Examples

IBM Cognos TM1:

Vendor: IBM

Description:

- IBM Cognos TM1 is a multidimensional planning and analytics platform.
- It enables users to create and interact with multidimensional cubes, conduct scenario planning, and perform what-if analysis.

Key Features:

- In-memory OLAP technology for fast query performance.
- Hierarchical data representation.
- Dynamic real-time analytics.

MOLAP - Examples

Oracle OLAP:

Vendor: Oracle

Description: Oracle OLAP is part of the Oracle Database offering and provides MOLAP capabilities. It allows users to create analytic workspaces, define dimensions, and perform multidimensional analysis using SQL and MDX.

Key Features:

- Integration with Oracle Database.
- Support for aggregations and calculations.
- Seamless integration with Oracle BI tools.

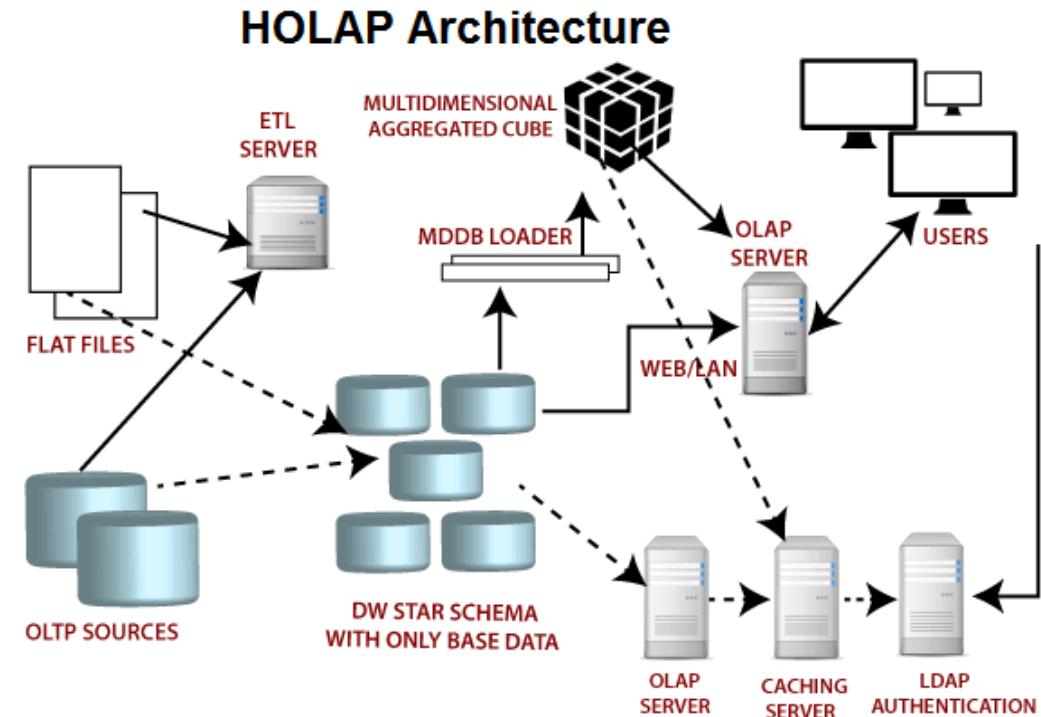
Hybrid OLAP (HOLAP)

Hybrid OLAP (HOLAP) is a combination of both Relational OLAP (ROLAP) and Multidimensional OLAP (MOLAP) architectures, aiming to leverage the advantages of both approaches.

- HOLAP systems are designed to provide the flexibility of ROLAP for handling large volumes of detailed data while taking advantage of the fast query performance and intuitive data representation of MOLAP for certain subsets of the data.
- The architecture integrates features from both ROLAP and MOLAP to optimize analytical processing.
- HOLAP systems save more substantial quantities of detailed data in the relational while the aggregations are stored in the pre-calculated cubes

Hybrid OLAP (HOLAP)

- Similar to ROLAP, HOLAP systems use an underlying relational database management system for storing detailed data.
- Detailed data, which may not fit into the in-memory structure, is stored in the relational database.
- Standard SQL queries are used for retrieving and managing detailed data.
- HOLAP systems create multidimensional cubes for specific subsets of data or for aggregations that benefit from in-memory processing.
- Aggregated and relevant data is loaded into the multidimensional cube.
- Fast and efficient queries are performed on the cube for certain analytical scenarios.



Hybrid OLAP (HOLAP)

OLAP server:

HOLAP balances the disk space requirement, as it only stores the aggregate information on the OLAP server and the detail record remains in the relational database. So no duplicate copy of the detail record is maintained.

Aggregation Management:

- HOLAP systems use a combination of pre-aggregation in the cube and on-the-fly aggregation in the relational database.
- Some aggregations are pre-calculated and stored in the cube for quick access.
- Aggregations not stored in the cube are calculated in real-time in the relational database when needed.

OLAP Engine:

- The OLAP engine in HOLAP systems is responsible for coordinating and managing queries that involve both the multidimensional cube and the relational database.
- Determines when to use the multidimensional cube or the relational database based on the nature of the query.
- Coordinates the integration of results from both sources to provide a unified view.



BITS Pilani

Pilani Campus



Query Performance Enhance Technique

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda

-
- 6.1. Snowflake – Query optimization Techniques
 - 6.2. Caching in Snowflake
 - 6.3. Clustering in Snowflake
 - 6.4. Micro-Partitions
 - 6.5. Materialized Views
 - 6.6. Query Profiling in Snowflake
 - 6.7. Aggregations

Query performance is crucial in data warehousing environments, where large volumes of data are stored and queried.

Several techniques are employed to enhance query performance in Snowflake Cloud data warehousing.

Automatic Clustering: Creating appropriate cluster on columns used frequently in WHERE clauses can significantly improve query performance. Clustering allows Snowflake to locate and retrieve data more efficiently.

Search Optimization: The search optimization service aims to significantly improve the performance of Selective point lookup queries on tables

Materialized Views: Materialized views store precomputed results of queries, enabling faster retrieval when the same or similar queries are executed. These views are periodically refreshed to keep the data up to date.

Query performance enhancement techniques

Aggregation and Summary Tables: Creating summary tables or aggregating data at different levels can accelerate queries that require aggregated results. This is particularly useful for reports and dashboards.

Star and Snowflake Schema: Designing a data warehouse using star or snowflake schema can enhance query performance. These schemas involve structuring the data in a way that optimizes joins and simplifies the query execution plan.

Optimizing Warehouse for performance: Strategies to fine-tune computing power in order to improve the performance of a query or set of queries running on a warehouse, including enabling the Query Acceleration Service.

Query performance enhancement techniques

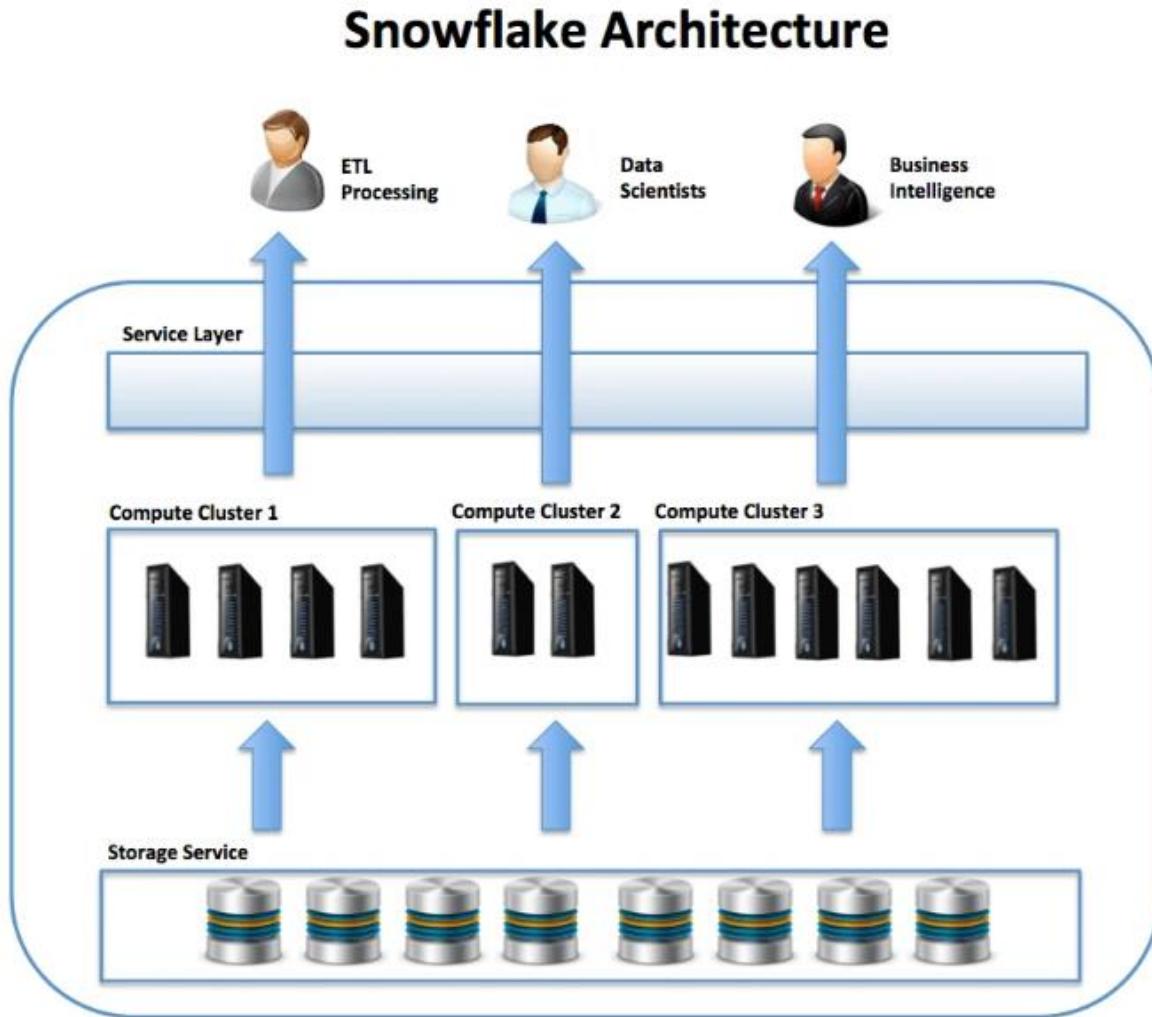
Caching: Caching involves storing the results of frequently executed queries in memory. When a similar query is issued, the system can retrieve the results from the cache, reducing the need to recompute the result set.

Query Profile: Snowflake provides query profile which shows details of the steps executed to complete the query and time spent on each step.

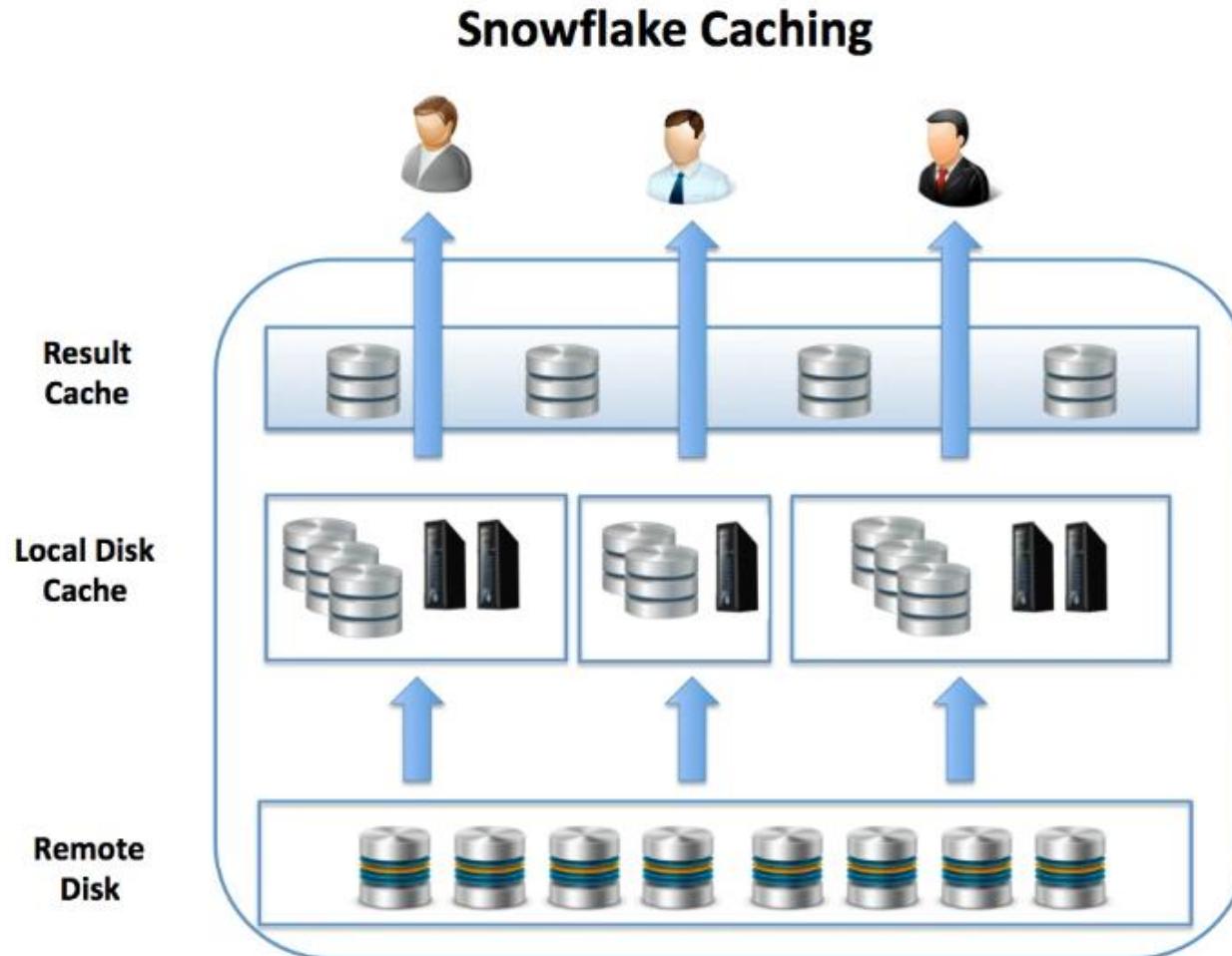
Implementing a combination of these techniques, based on the specific characteristics of the data and queries in your data warehouse, can significantly enhance query performance. It's essential to regularly monitor and analyze performance to identify areas for improvement and adjust strategies accordingly.

Caching in Snowflake

Recap – Snowflake Architecture



Snowflake Caching



Snowflake Caching

Snowflake Cache Layers

The diagram below illustrates the levels at which data and results are cached for subsequent use. These are:-

1. Result Cache: Which holds the results of every query executed in the past 24 hours. These are available across virtual warehouses, so query results returned to one user is available to any other user on the system who executes the same query, provided the underlying data has not changed.

2. Local Disk Cache: Which is used to cache data used by SQL queries. Whenever data is needed for a given query it's retrieved from the *Remote Disk* storage, and cached in SSD and memory.

3. Remote Disk: Which holds the long term storage. This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.999999999% durability. Even in the event of an entire data center failure.

Automatic Clustering

What are Micro-Partitions?

- All data in Snowflake tables is automatically divided into micro-partitions, which are contiguous units of storage.
- Each micro-partition contains between 50 MB and 500 MB of uncompressed data (note that the actual size in Snowflake is smaller because data is always stored compressed).
- Groups of rows in tables are mapped into individual micro-partitions, organized in a columnar fashion.
- This size and structure allows for extremely granular pruning of very large tables, which can be comprised of millions, or even hundreds of millions, of micro-partitions.
- Snowflake stores metadata about all rows stored in a micro-partition, including:
 - The range of values for each of the columns in the micro-partition.
 - The number of distinct values.
 - Additional properties used for both optimization and efficient query processing.

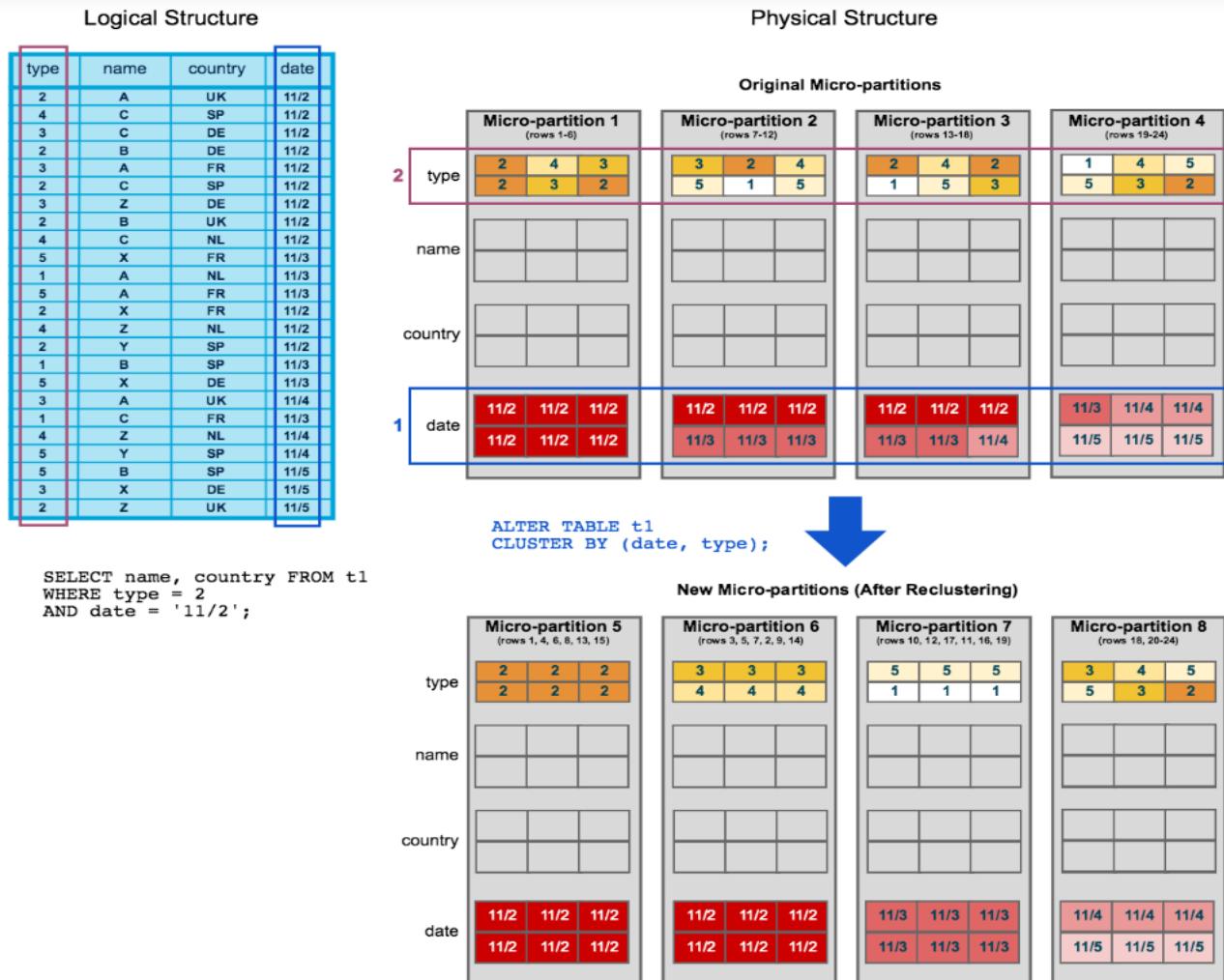
Note: Micro-partitioning is automatically performed on all Snowflake tables. Tables are transparently partitioned using the ordering of the data as it is inserted-loaded.

Automatic Clustering

What are Clustered Tables

- A clustered table is one where the underlying data is physically stored in the order of a user-defined set of key(s), rather than by natural ingestion order
- Selecting proper clustering keys is critical and requires an in-depth understanding of the common workloads and access patterns against the table in question

Automatic Clustering



```
SELECT name, country FROM t1
WHERE type = 2
AND date = '11/2';
```

Creating Clustered tables

--Adding Clustering key at the time of table creation

```
create or replace table t1 (c1 date, c2 string, c3 number) cluster by (c1, c2);
```

--Adding Clustering key to existing table

```
alter table t1 cluster by (c1, c2);
```

--Dropping Cluster key

```
alter table t1 drop clustering key;
```

--Suspend ReClustering

```
alter table t1 suspend recluster;
```

Materialized Views

What is Materialized View?

- A materialized view is a pre-computed data set derived from a query specification (the SELECT in the view definition) and stored for later use.
- Querying a materialized view is faster than executing a query against the base table of the view
- Materialized views can speed up expensive aggregation, projection, and selection operations, especially those that run frequently and that run on large data sets.

Guidelines - When to use Materialized View?

- Query results contain a small number of rows and/or columns relative to the base table (the table on which the view is defined).
- Query results contain results that require significant processing
- The query is on an external table (i.e., data sets stored in files in an external stage), which might have slower performance compared to querying native database tables.
- The view's base table does not change frequently.

Materialized views

Example

```
create materialized view mv1 as select * from table1 where c1>=1000 and  
c1<=40000;
```

```
create materialized view mv2 as select * from table1 where c2 in ('A','B');
```

Materialized views - Limitations

- A materialized view can query only a single table.
- Joins, including self-joins, are not supported.
- A materialized view cannot query:
 - A materialized view.
 - A non-materialized view.
 - A UDTF (user-defined table function).
- A materialized view cannot include:
 - UDFs (this limitation applies to all types of user-defined functions, including external functions).
 - Window functions.
 - HAVING clauses.
 - ORDER BY clause.
 - LIMIT clause.
 - GROUP BY keys that are not within the SELECT list. All GROUP BY keys in a materialized view must be part of the SELECT list.
 - <Few More...>
- Many aggregate functions are not allowed in a materialized view definition.
 - The aggregate functions that are ***supported*** in materialized views are:
 - [APPROX_COUNT_DISTINCT \(HLL\)](#).
 - [AVG](#) (except when used in [PIVOT](#)).
 - [BITAND AGG](#).
 - [BITOR AGG](#).
 - [BITXOR AGG](#).
 - [COUNT](#).
 - [MIN](#).
 - [MAX](#).
 - [STDDEV](#).
 - [STDDEV_POP](#).
 - [STDDEV_SAMP](#).
 - [SUM](#).
 - [VARIANCE \(VARIANCE_SAMP, VAR_SAMP\)](#).
 - [VARIANCE_POP \(VAR_POP\)](#).
 - The other aggregate functions are ***not supported*** in materialized views.

Complete List of Limitations is available here: <https://docs.snowflake.com/en/user-guide/views-materialized.html>

Materialized views

Refresh Process

- Maintenance of materialized views is performed by a background process and the timing is not predictable by the user.
- If maintenance falls behind, queries might run more slowly than they run when the views are up-to-date.
- Results will always be correct; if some micro-partitions of the materialized view are out of date, Snowflake skips those partitions and looks up data in the base table
- To see the last time that a materialized view was refreshed, check the “*Refreshed_On*” and “*Behind_By*” columns in “*Show Materialized Views*”

Cost

- Storage: Each materialized view stores query results, which adds to the monthly storage usage for your account.
- Compute resources: In order to prevent materialized views from becoming out-of-date, Snowflake performs automatic background maintenance of materialized views. When a base table changes, all materialized views defined on the table are updated by a background service that uses compute resources provided by Snowflake.
- These updates can consume significant resources, resulting in increased credit usage. However, Snowflake ensures efficient credit usage by billing your account only for the actual resources used. Billing is calculated in 1-second increments.

Snowflake – Query Profiling

- The Snowflake Query Profile is a comprehensive breakdown of the execution steps and performance statistics for a specific query.
- It offers insights into various aspects of query execution, including resource consumption, data movement across nodes, and potential bottlenecks.
- Analyzing the Query Profile can help identify areas for optimization and improve overall query performance.

Snowflake – Query Profiling

Purpose of the Query Profile

The Query Profile is an indispensable tool for improving query performance for several reasons:

- **Performance Optimization:** It helps identify inefficient query steps, allowing us to optimize and streamline the query's execution plan.
- **Troubleshooting:** When queries run slowly, the Query Profile provides crucial insights to troubleshoot performance issues.
- **Cost Efficiency:** By understanding resource consumption for each query, we can fine-tune it to reduce costs associated with query execution.

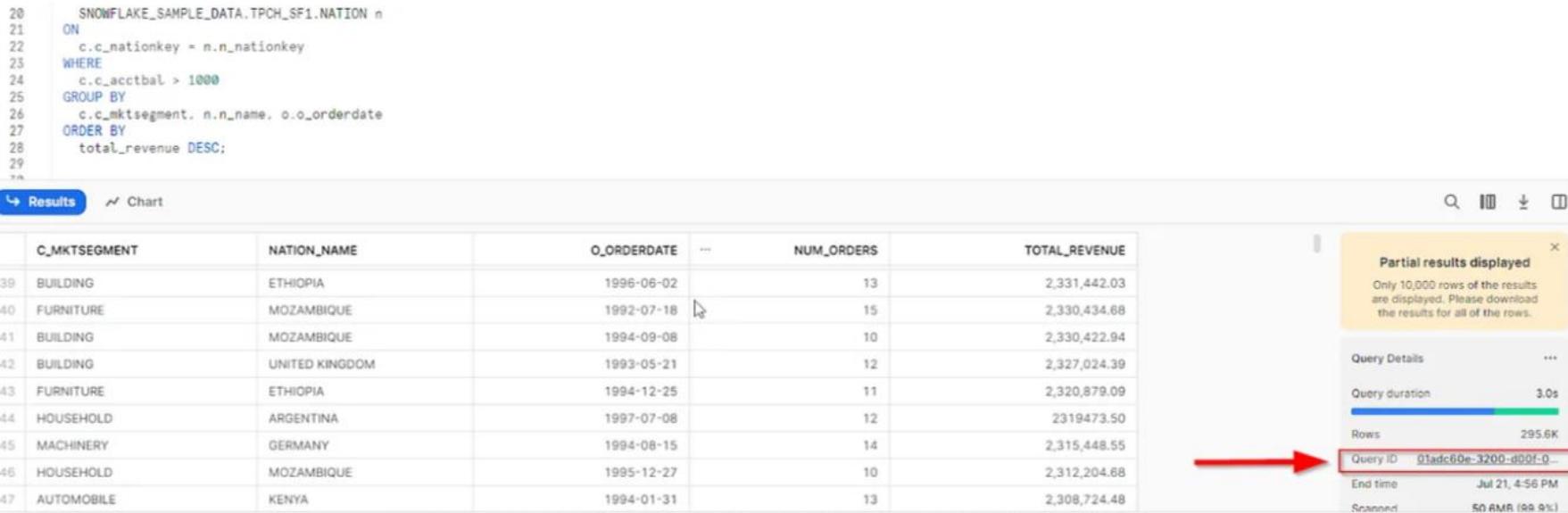
Snowflake – Query Profiling

Accessing Query Profile

Accessing the Query Profile is a straightforward process and can be done using *Snowsight UI* (web interface of Snowflake)

Using Snowsight:

Clicking on the **Query ID**(a unique identifier assigned to each executed query in Snowflake) provides instant access to the corresponding Query Profile



The screenshot shows the Snowsight UI interface. On the left, a code editor displays a SQL query:

```

20   SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.NATION n
21   ON
22     c.c_nationkey = n.n_nationkey
23   WHERE
24     c.c_acctbal > 1000
25   GROUP BY
26     c.c_mktsegment, n.n_name, o.o_orderdate
27   ORDER BY
28     total_revenue DESC;
29

```

Below the code editor is a results table with the following data:

C_MKTSEGMENT	NATION_NAME	O_ORDERDATE	NUM_ORDERS	TOTAL_REVENUE
BUILDING	ETHIOPIA	1996-06-02	13	2,331,442.03
FURNITURE	MOZAMBIQUE	1992-07-18	15	2,330,434.68
BUILDING	MOZAMBIQUE	1994-09-08	10	2,330,422.94
BUILDING	UNITED KINGDOM	1993-05-21	12	2,327,024.39
FURNITURE	ETHIOPIA	1994-12-25	11	2,320,879.09
HOUSEHOLD	ARGENTINA	1997-07-08	12	2,319,473.50
MACHINERY	GERMANY	1994-08-15	14	2,315,448.55
HOUSEHOLD	MOZAMBIQUE	1995-12-27	10	2,312,204.68
AUTOMOBILE	KENYA	1994-01-31	13	2,308,724.48

On the right, a sidebar displays "Partial results displayed" (Only 10,000 rows of the results are displayed. Please download the results for all of the rows.) and "Query Details" including "Query duration" (3.0s), "Rows" (295.6K), and "Scanned" (50 AMR (99.9%)). A red arrow points to the "Query ID" field, which contains the value `01adc60e-3200-d00f-0`.

Snowflake – Query Profiling

Query - 01adc60e-3200-d00f-0005-155a0004a25a

COMPUTE_WH A, SNOWFLAKEDEMOACCOUNT

Query Details **Query Profile**



Most Expensive Nodes (7 of ...)

TableScan [12]	31.7%
Aggregate [3]	8.3%
Result [0]	6.7%
TableScan [10]	5.0%
JoinFilter [11]	3.3%
Aggregate [2]	3.3%
Join [4]	1.7%

Profile Overview (Finished)

Total Execution Time	(2.0s) 100.0%
Processing	23.3%
Remote Disk I/O	36.7%
Initialization	40.0%

Statistics

Scan progress	100.00%
Bytes scanned	15.77MB
Percentage scanned from cache	0.00%
Bytes written to result	2.85MB

Search Optimization

What is Search Optimization?

The search optimization service aims to significantly improve the performance of Selective point lookup queries on tables

What are “Point Lookup” Queries?

- In a point lookup, users are typically looking for a handful of records, and they want to find them very quickly.
- Point lookups are very popular in interactive dashboards and data applications trying to conduct “needle in a haystack”-style queries.

Example: select * from log where msg = 'gnXHoNsU9WSROLegmbfDffHSDb0';

Search Optimization - Considerations

- Table Size is Large in GBs
- At least one of the columns accessed through the query filter operation has at least 100k-200k distinct values.
- The query uses equality predicate or predicates that use IN.
- The query returns a few rows with highly selective filters.
- The query typically runs for at least tens of seconds or minutes to fetch (example) 2 records.

Search Optimization

Example – How to enable Search Optimization

```
alter table table_name add search optimization;
```

Note: Once SEARCH OPTIMIZATION is enabled on a table, it takes sometime to populate the search access path for the table. To check if SEARCH OPTIMIZATION is 100% enabled on a table, "SHOW TABLES LIKE '%TABLE_NAME%'". Refer below for example

Example: SEARCH OPTIMIZATION enable but not 100% enabled

search_optimization	search_optimization_	search_optimization
ON	0	0

Example: SEARCH OPTIMIZATION 100% enabled

search_optimization	search_optimization_	search_optimization
ON	100	24032315392

Aggregations

Aggregations refer to the process of summarizing and condensing large volumes of data to provide meaningful insights at a higher level of granularity. Aggregations are crucial for optimizing query performance, reducing response times, and managing the complexity of analyzing large datasets. They are particularly essential in data warehousing environments where analytical queries often involve extensive datasets.

Common Aggregation Functions:

- Sum: Calculates the total sum of numeric values.
- Average (Mean): Computes the average value of a set of numeric values.
- Count: Counts the number of rows in a dataset.
- Min/Max: Determines the minimum or maximum value in a dataset.
- Group By: Aggregates data based on specified grouping criteria, creating summaries for each group.

Aggregations



The materialized view allows for faster query responses, especially when dealing with large datasets, by avoiding the need to perform expensive aggregations on the raw data for each query.

Materialized Views Example:

Consider a company's data warehouse that stores information about sales transactions. The raw data includes details such as product IDs, sales amounts, and transaction dates. We want to create a materialized view that pre-calculates the total sales amount for each product category to improve query performance.

Raw Sales Data:

TransactionID	ProductID	Category	SalesAmount	TransactionDate
1	101	Electronics	500	2022-01-01
2	102	Clothing	300	2022-01-02
3	103	Electronics	700	2022-01-03
4	104	Furniture	450	2022-01-04
...

Query Optimization:

Now, if a user wants to know the total sales for the "Electronics" category, instead of scanning the entire raw sales data, the system can quickly retrieve the result from the precomputed materialized view.

Total Sales by Category

Assume we create a materialized view that calculates the total sales amount for each product category.

Category	TotalSales
Electronics	1200
Clothing	300
Furniture	450

Query without Materialized View

```
SELECT Category, SUM(SalesAmount)
AS TotalSales
FROM RawSalesData
WHERE Category = 'Electronics'
GROUP BY Category;
```

Query with Materialized View:

```
SELECT Category, TotalSales
FROM
MaterializedView_TotalSalesByCategory
WHERE Category = 'Electronics';
```

Aggregations

Aggregations play a vital role in multidimensional data analysis through OLAP cubes.

Roll-up and Drill-down:

- **Roll-up:** Aggregating data from a detailed level to a higher level of summarization. For example, rolling up daily sales data to monthly sales.
- **Drill-down:** Moving from a higher level of summarization to a more detailed level. For example, drilling down from yearly revenue to quarterly revenue.

Query Performance Optimization:

Role: Aggregations help reduce the need for extensive scanning of raw data, improving query response times.

Example: Instead of calculating the sum of sales from detailed transaction records every time, precomputed aggregations provide faster access to summarized results.



BITS Pilani

Pilani Campus

Data Warehouse - Metadata

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda

Metadata

1. Role of metadata in DW
2. Metadata in Snowflake
3. Metadata Design & Implementation

Metadata

- Information about the data stored in the data warehouse.
- It provides descriptive details about the **structure, content, and context** of the data, enabling users to understand, manage, and utilize the data effectively
- For example, the index of a book serves as a metadata for the contents in the book.
- In other words, metadata is the **summarized data** that leads us to detailed data.

In terms of data warehouse, we can define metadata as follows:

- Metadata is the road-map to a data warehouse.
- Metadata in a data warehouse defines the warehouse objects.
- Metadata acts as a directory. This directory helps the decision support system to locate the contents of a data warehouse

Examples of Metadata

- File metadata: This includes information about a file, such as its name, size, type, and creation date.
- Image metadata: This includes information about an image, such as its resolution, color depth, and camera settings.
- Music metadata: This includes information about a piece of music, such as its title, artist, album, and genre.
- Video metadata: This includes information about a video, such as its length, resolution, and frame rate.
- Document metadata: This includes information about a document, such as its author, title, and creation date.
- Database metadata: This includes information about a database, such as its structure, tables, and fields.
- Web metadata: This includes information about a web page, such as its title, keywords, and description.

Role of Metadata

Metadata acts as a directory:

- A metadata repository or catalog contains information about the structure, content, and location of data assets in the data warehouse.
- This directory helps users and systems navigate and locate relevant data.

Mapping of data in transformation:

- Metadata includes mappings between source data elements from operational systems and their corresponding target data elements in the data warehouse.
- For instance, it specifies how fields like "CustomerName" from a CRM system map to "Customer_Name" in the data warehouse.

-- Create a table to store the mapping between source and target data elements

```
CREATE TABLE MetadataMapping ( SourceField NVARCHAR(50), TargetField NVARCHAR(50) );
```

-- Insert some sample data into the MetadataMapping table

```
INSERT INTO MetadataMapping (SourceField, TargetField) VALUES ('CustomerName', 'Customer_Name'),
```

```
('OrderID', 'Order_ID'), ('OrderDate', 'Order_Date');
```

-- Query the MetadataMapping table to retrieve mappings
SELECT *
FROM MetadataMapping

Snowflake - Metadata

- Snowflake keeps track of all the defined objects within it and their associated metadata.
- In order to make it simple for users to inspect some of the information about the databases, schemas, and tables in the Snowflake, the metadata information is supplied as a collection of views against the metadata layer.
- In a Simple words you can say two type of data dictionary is available in Snowflake where you can find various metadata information:
 1. Information Schema
 2. Account Usage

Information Schema

- The Snowflake Information Schema (aka “Data Dictionary”) consists of a set of system-defined views and table functions that provide extensive metadata information about the objects created in your account.
- Whenever you create any new Database, this Schema is auto created.
- The schema contains the following objects:
 - Views for all the objects contained in the database
 - Views for account-level objects (non-database objects such as roles, warehouses, and databases) e.g., TABLES, VIEWS, PIPES etc.
 - Table functions for historical and usage data across your account. e.g COPY History, Query History etc.
- Queries on INFORMATION_SCHEMA views do not guarantee consistency with respect to concurrent DDL. For example, if a set of tables are created while a long-running INFORMATION_SCHEMA query is being executed, the result of the query may include some, none, or all of the tables created.
- The output of a view or table function depend on the privileges granted to the user’s current role.
- When querying an INFORMATION_SCHEMA view or table function, only objects for which the current role has been granted access privileges are returned.

Information_Schema Commonly Used Views

TABLES View

- This Information Schema view displays a row for each table in the specified (or current) database.
- Some of the information contained in this view (not a complete list)

Column Name	Data Type	Description
TABLE_CATALOG	TEXT	Database that the table belongs to.
TABLE_SCHEMA	TEXT	Schema that the table belongs to.
TABLE_NAME	TEXT	Name of the table.
TABLE_OWNER	TEXT	Name of the role that owns the table.
TABLE_TYPE	TEXT	Indicates the table type. Valid values are BASE TABLE, TEMPORARY TABLE, EXTERNAL TABLE, EVENT TABLE, VIEW, or MATERIALIZED VIEW.
IS_TRANSIENT	TEXT	Indicates whether this is a transient table.
CLUSTERING_KEY	TEXT	Clustering key for the table.
ROW_COUNT	NUMBER	Number of rows in the table.
BYTES	NUMBER	Number of bytes accessed by a scan of the table.

VIEWS View

- This Information Schema view displays a row for each view in the specified (or current) database, including the INFORMATION_SCHEMA views for the database.
- Some of the information contained in this view (not a complete list)

Column Name	Data Type	Description
TABLE_CATALOG	TEXT	Database that the view belongs to.
TABLE_SCHEMA	TEXT	Schema that the view belongs to.
TABLE_NAME	TEXT	Name of the view.
TABLE_OWNER	TEXT	Name of the role that owns the view.
VIEW_DEFINITION	TEXT	Text of the view's query expression.
CHECK_OPTION	TEXT	Not applicable for Snowflake.
IS_UPDATABLE	TEXT	Not applicable for Snowflake.
INSERTABLE_INTO	TEXT	Not applicable for Snowflake.
IS_SECURE	TEXT	Specifies whether the view is secure.
CREATED	TIMESTAMP_LTZ	Creation time of the view.
LAST_ALTERED	TIMESTAMP_LTZ	Date and time the object was last altered. See Usage

PROCEDURES View

- This Information Schema view displays a row for each stored procedure defined in the specified (or current) database.
- Some of the information contained in this view (not a complete list)

Column Name	Data Type	Description
PROCEDURE_CATALOG	TEXT	Database which the stored procedure belongs to
PROCEDURE_SCHEMA	TEXT	Schema which the stored procedure belongs to
PROCEDURE_NAME	TEXT	Name of the stored procedure
PROCEDURE_OWNER	TEXT	Name of the role that owns the stored procedure
DATA_TYPE	TEXT	Return value data type
CHARACTER_MAXIMUM_LENGTH	NUMBER	Maximum length in characters of string return value
CHARACTER_OCTET_LENGTH	NUMBER	Maximum length in bytes of string return value
NUMERIC_PRECISION	NUMBER	Numeric precision of numeric return value
NUMERIC_PRECISION_RADIX	NUMBER	Radix of precision of numeric return value
NUMERIC_SCALE	NUMBER	Scale of numeric return value
PROCEDURE_LANGUAGE	TEXT	Language of the stored procedure

Account Usage

- In the SNOWFLAKE database, the ACCOUNT_USAGE schemas enable querying object metadata, as well as historical usage data, for your account.
- In general, these views mirror the corresponding views and table functions in the Snowflake Information Schema, but with the following differences:
 - Records for dropped objects included in each view.
 - Longer retention time for historical usage data.
 - Data latency - AC has latency - it can take from 45 minutes up to 3 hours to have data available in the views. There is no latency for information schema.
 - Apart from Account usage it contains other schemas with account related usage information (ORGANIZATION_USAGE etc.)

Important note: SNOWFLAKE DB and Account usage schema is by default available only to ACCOUNTADMIN role. It can be of course granted to other roles. You need to grant IMPORTED PRIVILEGES to desired role.

Dropped object records

- Account usage views include records for all objects that have been dropped.
- Many of the views for object types contain an additional "DELETED" column that displays the timestamp when the object was dropped.
- Objects can be dropped and recreated with the same name, to differentiate between objects records that have the same name, the account usage views include ID columns, where appropriate, that display the internal IDs generated and assigned to each record by the system.
- If a column for an object name (e.g. the `TABLE_NAME` column) is NULL, that object has been dropped.
- In this case, the columns for the names and IDs of the parent objects (e.g. the `DATABASE_NAME` and `SCHEMA_NAME` columns) are also NULL.
- Note that in some views, the column for the object name might still contain the name of the object, even if the object has been dropped.

Data Latency

- Due to the process of extracting the data from Snowflake's internal metadata store, the account usage views have some natural latency:
- For most of the views, the latency is 2 hours (120 minutes).
- For the remaining views, the latency varies between 45 minutes and 3 hours.
- For details, see the list of views for each schema (in this topic).
- Also, note that these are all maximum time lengths; the actual latency for a given view when the view is queried may be less.
- In contrast, views in the [Snowflake Information Schema](#) do not have any latency.

Historical Data Retention

- Certain account usage views provide historical usage metrics.
- The retention period for these views is 1 year (365 days).
- In contrast, the corresponding views in the [Snowflake Information Schema](#) have much shorter retention periods, ranging from 7 days to 6 months, depending on the views.

[Click here for detailed information](#)

Enabling other roles to use schemas in the SNOWFLAKE database

- By default, the SNOWFLAKE database is visible to all users
- Access to schemas in this database can be granted by a user with the ACCOUNTADMIN role by granting privilege as below:

Grant IMPORTED PRIVILEGES on the SNOWFLAKE database

For example, to grant IMPORTED PRIVILEGES on the SNOWFLAKE database to additional role:

```
USE ROLE ACCOUNTADMIN;  
GRANT IMPORTED PRIVILEGES ON DATABASE SNOWFLAKE TO ROLE customrole1;
```

A user with that is granted the customrole1 role can query a view as follows:

```
USE ROLE customrole1;  
SELECT database_name, database_owner FROM SNOWFLAKE.ACCOUNT_USAGE.DATA
```

Account_Usage – Frequently Used Views

View

[AUTOMATIC CLUSTERING HISTORY](#)

[COLUMNS](#)

[COPY HISTORY](#)

[DATABASES](#)

[DATABASE REPLICATION USAGE HISTORY](#)

[DATABASE STORAGE USAGE HISTORY](#)

[LOGIN HISTORY](#)

[MATERIALIZED VIEW REFRESH HISTORY](#)

[METERING DAILY HISTORY](#)

[METERING HISTORY](#)

[PROCEDURES](#)

[QUERY HISTORY](#)

[ROLES](#)

[SEARCH OPTIMIZATION HISTORY](#)

[STAGE STORAGE USAGE HISTORY](#)

[STORAGE USAGE](#)

[TABLES](#)

[USERS](#)

[VIEWS](#)

[WAREHOUSE METERING HISTORY](#)

QUERY_HISTORY View

- This Account Usage view can be used to query Snowflake query history by various dimensions (time range, session, user, warehouse, etc.) within the last 365 days (1 year).
- It can show information like: Query Id, Query text, Database Name, Schema Name, User Id, Role Name, Warehouse Name, Start Time, End Time etc.

COLUMNS View

- This Account Usage view displays a row for each column in the tables defined in the account.
- It can show information like: Column Name, Table Name, Table Schema, Data Type etc.

Login_History View

- This Account Usage view can be used to query login attempts by Snowflake users within the last 365 days (1 year).
- Latency for the view may be up to 120 minutes (2 hours).
- Information that can be seen in this view: User Name, Client IP address, Login successful or not, Error codeLogin time etc.

WAREHOUSE_METERING_HISTORY view

- This Account Usage view can be used to return the hourly credit usage for a single warehouse (or all the warehouses in your account) within the last 365 days (1 year).

Column Name	Data Type
READER_ACCOUNT_NAME	TEXT
START_TIME	TIMESTAMP_LTZ
END_TIME	TIMESTAMP_LTZ
WAREHOUSE_ID	NUMBER
WAREHOUSE_NAME	TEXT
CREDITS_USED	NUMBER
CREDITS_USED_COMPUTE	NUMBER
CREDITS_USED_CLOUD_SERVICES	NUMBER

Information_Schema Vs Account_Usage

Difference	Account Usage	Information Schema
Includes dropped objects	Yes	No
Latency of data	From 45 minutes to 3 hours (varies by view)	None
Retention of historical data	1 Year	From 7 days to 6 months (varies by views)

Categories of Metadata

Technical Metadata:

- **Database System Names:** Technical metadata includes information about the Datawarehouse systems used, such as Snowflake.
- **Table and Column Names and Sizes:** This includes details about the names and sizes of tables and columns within the Datawarehouse. For example, the "Customers" table may have columns like "CustomerID" and "CustomerName" with specific data types and lengths.
- **Data Types and Allowed Values:** Technical metadata specifies the data types and allowed values for each column. For instance, a "Gender" column may have the data type VARCHAR(1) and only allow values 'M' or 'F'.
- **Structural Information:** Technical metadata includes details about the database/schema/table structure, such as primary and foreign key attributes, indices, and constraints in a table. For example, it specifies that the "Orders" table has a primary key on the "OrderID" column and a foreign key referencing the "Customers" table

Categories of Metadata

Operational Metadata:

- **Currency of Data:** Operational metadata includes information about the currency of data, indicating whether it is active, archived, or purged. For example, it may specify that sales data for the current year is active, while data from previous years is archived.
- **Data Lineage:** Operational metadata tracks the history of data movement and transformations applied to it. For instance, it may document that sales data was extracted from the transactional system, transformed to calculate monthly sales totals, and loaded into the data warehouse.

Data currency is a term that refers to the real-time value and relevance of data in a business setting. Data currency is important for businesses that depend on data to make accurate decisions and gain insights

Designing Metadata

- Let's consider a real-world case study of a retail company that operates multiple brick and-mortar stores and an online e-commerce platform.
- The company wants to design metadata for its data warehouse to support various business intelligence (BI) and analytics initiatives.

Designing Metadata

- Designing metadata involves creating a framework and processes to capture, organize, manage, and utilize metadata effectively throughout the data lifecycle.
- **Step 1: Identify Metadata Requirements:**
 - Understand the needs of stakeholders, including business users, analysts, developers, and administrators, to determine what metadata is required.
 - Define the purpose and scope of metadata, considering factors such as data governance, data lineage, data quality, and regulatory compliance.

Let's break down the metadata requirements based on different stakeholders and their needs:

Designing Metadata



Step 1: Identify Metadata Requirements:

Let's apply this step to the case study of our retail company:

Business Users:

- Sales managers need metadata to analyze sales performance by region, product category, and customer segment.
- Marketing analysts need metadata to track the effectiveness of marketing campaigns and promotions.
- Finance executives need metadata to monitor revenue, expenses, and profitability metrics.

Technical Users:

- Data engineers need metadata to understand the data model, including tables, columns, primary keys, and foreign keys.
- Data scientists need metadata to assess data quality issues, such as missing values, duplicates, and inconsistencies.
- Data architects need metadata to document data lineage and transformations across different stages of the data pipeline.

Step 1: Identify Metadata Requirements:

Data Stewards and Administrators:

- Data stewards, who are responsible for implementing policies, processes, and standards to ensure data is managed efficiently and in compliance with regulatory guidelines, need metadata to manage access controls and data governance policies, ensuring compliance with regulatory requirements.
- Database administrators need metadata to monitor data loading schedules, data backups, and system performance metrics.

Compliance and Regulatory Requirements:

- Metadata should include information about data privacy measures, such as encryption, anonymization, and access controls, to comply with GDPR and other regulations.
- Metadata should document data lineage to trace the source of sensitive data and demonstrate compliance with regulatory audits

Designing Metadata



Step 2: Defining Metadata Categories:

Categorize metadata into logical groups based on their purpose and usage.

Common metadata categories include:

- **Structural Metadata:** Describes the structure of the data warehouse, including tables, columns, keys, relationships, etc.
- **Descriptive Metadata:** Provides information about the content and context of the data, such as data definitions, business rules, data lineage, etc.
- **Administrative Metadata:** Includes information about data warehouse administration, such as data loading schedules, access controls, user permissions, etc.
- **Business Metadata:** Represents business-specific information related to data semantics, data ownership, data usage policies, etc.

Case study example:

Examples of structural metadata for our retail company include:

- Table Name: sales_transactions
- Column Name: transaction_id, customer_id, product_id, transaction_date, quantity, unit_price, total_amount
- Primary Key: transaction_id
- Foreign Key: customer_id, product_i

Designing Metadata



Step 2: Defining Metadata Categories:

Descriptive Metadata: Examples of descriptive metadata for our retail company include:

- **Data Definition:** Sales transaction details including customer ID, product ID, transaction date, quantity, and amount.
- **Business Rule:** Each transaction must have a unique transaction ID and valid customer and product IDs.
- **Data Source:** Transactional databases of the e-commerce platform.

Administrative Metadata: Examples of administrative metadata for our retail company include:

- **Data Loading Schedule:** Daily data loads from transactional databases to the data warehouse.
- **Data Access Permissions:** Only authorized users can access sales data, following role-based access control (RBAC) policies.
- **Data Retention Policy:** Sales data is retained for seven years for compliance and historical analysis purposes.

Designing Metadata



Step 2: Defining Metadata Categories:

Business Metadata: Examples of business metadata for our retail company include:

- **Data Owner:** Sales department
- **Data Usage Guidelines:** Sales data can be used for performance analysis, forecasting, and customer segmentation.
- **Data Classification:** Sales data is classified as confidential and should be handled with appropriate security measures

Designing Metadata



Step 3: Defining Metadata Attributes:

For each metadata category, define the specific attributes or properties that need to be captured.

Structural Metadata:

- Table Name: sales_transactions
- Column Names: transaction_id, customer_id, product_id, transaction_date, quantity, unit_price, total_amount
- Data Types: Integer, Integer, Integer, Date, Integer, Decimal, Decimal
- Primary Key: transaction_id
- Foreign Keys: customer_id (references customer table), product_id (references product table)

Descriptive Metadata:

- Data Definition: Sales transaction details including customer ID, product ID, transaction date, quantity, and amount.
- Business Rule: Each transaction must have a unique transaction ID and valid customer and product IDs.
- Data Source: Transactional databases of the e-commerce platform.

Step 3: Defining Metadata Attributes:

Administrative Metadata:

- Data Loading Schedule: Daily data loads from transactional databases to the data warehouse.
- Data Access Permissions: Only authorized users (sales analysts, marketing managers) can access the sales_transactions table with read and write permissions.
- Data Retention Policy: Sales data is retained for seven years for compliance and historical analysis purposes.

Business Metadata:

- Data Owner: Sales department, including defining specific point of contact in Sales department.
- Data Usage Guidelines: Sales data can be used for performance analysis, forecasting, and customer segmentation, but should not be shared outside the company without proper authorization.
- Data Classification: Sales data is classified as confidential and should be handled with appropriate security measures. There could be Top Secret data category as well and that needs to be tagged appropriately.

Designing Metadata



Step 4: Defining Metadata Repository:

Choose a suitable repository or database to store metadata.

- Design the schema of the metadata repository to accommodate different metadata categories and attributes.
 - Ensure that the metadata repository supports efficient querying and retrieval of metadata.
 - Implement Metadata Capture Mechanisms.
-
- ✓ For our retail company, we can choose a relational database management system (RDBMS) such as PostgreSQL or MySQL to store metadata. These databases offer flexibility, scalability, and robust querying capabilities required for managing metadata effectively.
 - ✓ We can design the schema of the metadata repository to accommodate different metadata categories and attributes using tables. Each table corresponds to a metadata category, and columns represent metadata attributes.
 - ✓ In an Enterprise Setup, specialized Software are also available in the industry, like Informatica, Collibra etc.

Step 5: Implement Metadata Capture Mechanisms:

- Develop processes and tools to capture and populate metadata automatically wherever possible.
- Implement metadata extraction routines to extract metadata from source systems, ETL processes, data models, etc.
- Allow manual entry of metadata when automation is not feasible.

Develop Processes and Tools for Automatic Metadata Capture:

We can develop processes and tools to automatically capture and populate metadata wherever possible. This can include:

- **Automatic extraction of metadata from source systems:** When new data is ingested into the data warehouse from source systems (such as transactional databases), metadata about the data structure, attributes, and relationships can be automatically extracted.
- **Integration with ETL processes:** Metadata capture mechanisms can be integrated into the Extract, Transform, Load (ETL) processes to capture information about data transformations, mappings, and data lineage. Or scan of metadata can be done on regular cadence to capture delta metadata from previous runs
- **Utilizing data modeling tools:** Metadata about data models, schemas, and relationships can be automatically captured using data modeling tools such as ERwin or Lucidchart.

Example: A script can be developed to automatically scan the structure of new tables or files loaded into the data warehouse and populate the structural metadata table with information about table names, column names, data types, and primary/foreign key relationships

Step 5: Implement Metadata Capture Mechanisms:

Allow Manual Entry of Metadata:

In cases where automation is not feasible or where additional metadata needs to be provided by users, manual entry of metadata can be allowed. This can include:

- **Providing user-friendly interfaces:** Develop interfaces or forms where users can manually enter metadata attributes and descriptions.
- **Metadata approval workflows:** Implement workflows for metadata submission, review, and approval to ensure consistency and accuracy of manually entered metadata.

Example: Users can use a web-based form to manually enter descriptive metadata such as data definitions, business rules, and data usage guidelines for specific tables or data sources. Metadata stewards or administrators can review and approve the submitted metadata before it is stored in the metadata repository.

Designing Metadata

Step 6: Establish Metadata Governance:

1. Define metadata standards, policies, and procedures for metadata management.
2. Establish roles and responsibilities for metadata stewardship and ownership.
3. Implement mechanisms for metadata validation, review, and approval.
4. Integrate Metadata with Data Warehouse systems.

Define Metadata Standards, Policies, and Procedures:

- Metadata standards define the structure, format, and quality requirements for metadata.
- Policies outline the rules and guidelines for metadata management, including data naming conventions, metadata ownership, and data classification.
- Procedures specify the steps and processes for capturing, storing, and managing metadata.

Example:

- The retail company defines a metadata standard that specifies naming conventions for tables and columns, data types, and documentation requirements for descriptive metadata.
- Policies are established to ensure that all new data assets undergo metadata capture before being deployed in the data warehouse.
- Procedures are documented for metadata validation and review processes²⁷⁴

Step 6: Establish Metadata Governance:

Establish Roles and Responsibilities:

- Metadata stewardship involves assigning roles and responsibilities for managing and maintaining metadata.
- Metadata owners are responsible for overseeing metadata assets and ensuring compliance with metadata policies.
- Metadata stewards are responsible for day-to-day management tasks, such as metadata capture, validation, and documentation.

Example: The retail company assigns the data architect as the metadata owner, responsible for defining metadata standards and policies. Data stewards are appointed within each department (e.g., sales, marketing, finance) to manage metadata for their respective data assets.

Designing Metadata

Step 7: Integrate Metadata with Data Warehouse Tools:

Define Integration Requirements:

- Identify data warehouse tools and platforms where metadata integration is needed, such as data modeling tools, ETL tools, and BI platforms.
- Determine the types of metadata to be integrated with each tool and the frequency of synchronization.

Example: The retail company identifies integration points with tools such as ERwin for data modeling, Informatica for ETL, and Tableau for BI. Metadata integration requirements include synchronizing structural metadata with data modeling tools, capturing transformation metadata from ETL processes, and exposing metadata to BI tools for data discovery and lineage analysis.

Implement Integration Mechanisms:

- Develop connectors or APIs to facilitate metadata exchange between the data warehouse and other tools.
- Configure metadata repositories to support seamless integration with data warehouse tools and platforms.

Example: The retail company deploys metadata management software that provides connectors for popular data warehouse tools. APIs are used to extract metadata from ETL workflows and load it into the metadata repository. Integration configurations are set up to synchronize metadata changes bidirectionally between the data warehouse and connected tool

Designing Metadata



Step 8: Monitor and Maintain Metadata

Implement processes for ongoing monitoring and maintenance of metadata.

- Example: The retail company sets up automated monitoring processes that track changes in metadata usage patterns and detect any deviations from predefined thresholds. Metadata usage reports are generated periodically to assess the effectiveness of metadata in supporting business analytics and decision-making.

Regularly update metadata as changes occur in the data warehouse environment.

- Example: Whenever a new table is added to the data warehouse or existing schema undergoes modifications, the corresponding metadata is automatically updated through predefined extraction routines. Metadata stewards are notified of these changes and responsible for reviewing and approving metadata updates before they are applied.

Perform periodic audits to ensure metadata consistency, accuracy & completeness.

- Example: The retail company conducts quarterly audits of metadata by comparing metadata records with actual data assets stored in the data warehouse. Any inconsistencies or discrepancies are documented and investigated by the metadata governance team. Corrective actions are taken to reconcile metadata discrepancies and ensure data integrity.



BITS Pilani

Pilani Campus



Data Warehousing Infrastructure

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda

- DW infrastructure
 - Capacity Planning
 - Security for DW
 - Hardware for DW
- Deployment: on-premise, cloud
- IaaS Vs PaaS Vs SaaS
- Snowflake Implementation Aspects

DW Infrastructure

- Data Warehouses are central repositories of data coming from one or more disparate sources.
- Data warehouses have also evolved tremendously in the last 2 decades with technologies from Teradata, Hadoop to AWS Redshift, Azure Synapse & Snowflake.
- Data warehouses can be of various configurations based on an organization's requirement, policies and budget but primarily there are two types of data warehouses in terms of infrastructure:
 - On-premise data warehouses/Traditional data warehouses
 - Cloud data warehouses.

On-Premise Data Warehouses

Traditional Data Warehouse:

- Traditional data warehouses are built using relational database management systems (RDBMS) such as Oracle, SQL Server, or IBM Db2.
- They typically follow the extract, transform, load (ETL) process to collect, cleanse, and integrate data from various operational systems into a centralized repository.
- Data in traditional data warehouses is structured and stored in tables with predefined schemas.

Purpose of On-Premise Data Warehouses:

- **Centralized Data Storage:** On-premise data warehouses serve as centralized repositories for storing and managing structured and sometimes semi-structured data from multiple sources within an organization.
- **Decision Support and Analytics:** They support decision-making processes by providing business users with access to consistent, integrated, and timely information for analysis, reporting, and business intelligence.
- **Data Integration and Transformation:** On-premise data warehouses facilitate the integration, transformation, and consolidation of data from disparate sources, enabling organizations to derive insights from a unified view of their data.
- **Control over Data Infra:** By hosting data warehouses on-premise, organizations have more control over their data infrastructure, allowing them to tailor hardware resources to meet specific workload requirements.

On-Premise Data Warehouses

Examples of On-Premise Data Warehouses:

- **Retail Industry:** A retail company may deploy an on-premise data warehouse to consolidate sales data from various channels (in-store, online, mobile) and analyze customer behavior, inventory management, and sales performance.
- **Financial Services:** Banks and financial institutions often use on-premise data warehouses to analyze transactional data, detect fraud, assess risk, and comply with regulatory reporting requirements.
- **Healthcare Sector:** Healthcare organizations may implement on-premise data warehouses to integrate electronic health records (EHR), patient demographics, medical billing data, and clinical research data for population health management, predictive analytics, and improving patient outcomes.

Use of On-Premise Data Warehouses:

Business Intelligence (BI) and Reporting: On-premise data warehouses enable organizations to generate standard and ad-hoc reports, dashboards, and visualizations to monitor KPIs, track performance metrics, and gain insights into business operations.

Advanced Analytics: Data scientists and analysts can leverage on-premise data warehouses to perform advanced analytics tasks such as data mining, predictive modeling, and machine learning to uncover hidden patterns, trends, and correlations in data.

Operational Decision Support: On-premise data warehouses support operational decision-making by providing real-time or near-real-time access to relevant data for frontline employees, managers, and executives across various departments.

Data Governance and Security: Organizations can implement robust data governance policies and security measures to protect sensitive data stored in on-premise data warehouses, ensuring compliance with industry regulations and safeguarding against data breaches.

Cloud Data Warehouses

- Cloud data warehouses are data warehousing solutions that are hosted, managed, and accessed via cloud computing infrastructure provided by third-party service providers.
- Unlike on-premise data warehouses, which require organizations to own and maintain hardware and software on-site, cloud data warehouses leverage the **scalability, flexibility, and cost-effectiveness of cloud computing platforms** to deliver data storage and analytical capabilities as a service over the internet.

Python for Cloud Databases

Python provides libraries and frameworks to interact with these databases. Here are a few examples:

- **Boto3:** Boto3 is the AWS SDK for Python. You can use it to interact with AWS services including RDS. It allows you to manage RDS instances, create, read, update, and delete databases, and perform various other tasks.
- **google-cloud-python:** This is the official Python client library for Google Cloud. It provides APIs for various Google Cloud services including Cloud SQL. You can use it to manage Cloud SQL instances, perform database operations, and more.
- **PyODBC:** PyODBC is a Python module that allows you to connect to ODBC-compliant databases, which include many cloud databases like Azure SQL Database.
- **pymongo:** If you're working with MongoDB, pymongo is the official Python driver for MongoDB. It allows you to interact with MongoDB databases hosted on MongoDB Atlas or any other MongoDB instance.
- **SQLAlchemy:** SQLAlchemy is a SQL toolkit and Object-Relational Mapping (ORM) library for Python. It supports a variety of database backends including many cloud databases. It provides a higher-level abstraction for working with databases, allowing you to write Python code rather than SQL queries directly.

Types of Cloud Data Warehouses:

SQL-based Cloud Data Warehouses:

- These cloud data warehouses are built on relational database management systems (RDBMS) and offer SQL-based query interfaces for data access and analysis.
- Examples include Amazon Redshift, Google BigQuery, Snowflake, and Microsoft Azure Synapse Analytics (formerly known as Azure Synapse).
- They support traditional data warehousing concepts such as star schemas, dimensional modeling, and SQL-based analytics.

NoSQL-based Cloud Data Warehouses:

- Some cloud data warehouses are built on NoSQL (Not Only SQL) databases or distributed computing platforms, offering schema-less storage and flexible data models.
- Examples include Amazon DynamoDB, Google Cloud Datastore, and Azure Cosmos DB.
- These platforms are suitable for storing and analyzing semi-structured data types, such as JSON, XML, and key-value pairs.
- However, modern cloud data warehouses are now evolving to handle semi-structured data as well. For example, Snowflake supports loading JSON data into structured form or loading JSON or XML document in as-is form.

Purpose of Cloud Data Warehouses

- **Scalability and Elasticity:** Cloud data warehouses offer on-demand scalability, allowing organizations to easily scale their storage and compute resources up or down based on workload requirements. This elasticity ensures optimal performance and cost efficiency.
- **Cost-effectiveness:** By leveraging pay-as-you-go pricing models and eliminating the need for upfront hardware investments, cloud data warehouses offer cost-effectiveness, particularly for organizations with fluctuating data processing needs.
- **Global Accessibility:** Cloud data warehouses enable users to access and analyze data from anywhere with an internet connection, making them suitable for distributed teams, remote work environments, and global operations.
- **Integration with Cloud Services:** Cloud data warehouses seamlessly integrate with other cloud services and tools, such as data lakes, machine learning platforms, and business intelligence (BI) tools, enabling organizations to build end-to-end data pipelines and analytical workflows.

Examples of Cloud Data Warehouses

Amazon Redshift:

- Amazon Redshift is a fully managed, petabyte-scale data warehouse service offered by Amazon Web Services (AWS).
- It is based on PostgreSQL and optimized for online analytical processing (OLAP) workloads.
- Redshift offers features such as automatic scaling, columnar storage, and integration with AWS services like S3 and Lambda.

Google BigQuery:

- Google BigQuery is a serverless, highly scalable, and fully managed data warehouse service provided by Google Cloud Platform (GCP).
- It offers a familiar SQL interface for querying and analyzing large datasets in real-time.
- BigQuery features automatic scaling, data encryption, and integration with Google Cloud Storage and Google Data Studio.

Snowflake:

- Snowflake is a cloud-native data warehouse built for the cloud era, offering a unique architecture that separates storage, compute, and services layers.
- It supports both structured and semi-structured data, with native support for JSON, Avro, and Parquet formats.
- Snowflake provides features such as automatic scaling, data sharing, and multi-cluster compute.

Use of Cloud Data Warehouses

Data Analytics and Business Intelligence: Cloud data warehouses enable organizations to perform advanced analytics, generate reports, and visualize insights using BI tools and analytics platforms.

Data Integration and Consolidation: Cloud data warehouses facilitate the integration of data from various sources, including transactional databases, IoT devices, and external APIs, into a centralized repository for analysis and reporting.

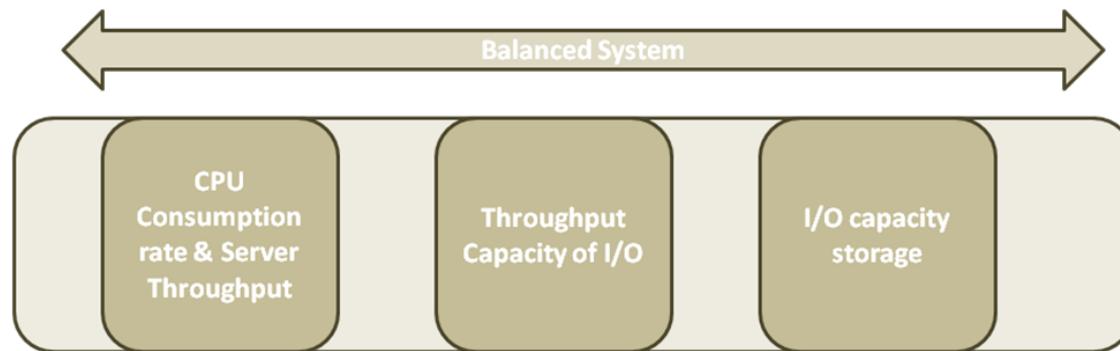
Real-time Data Processing: Some cloud data warehouses support real-time data ingestion and processing, allowing organizations to analyze streaming data and make data-driven decisions in near real-time.

Machine Learning and AI: Cloud data warehouses integrate with machine learning and AI platforms, enabling organizations to build predictive models, perform data mining, and extract actionable insights from large datasets.

Data Sharing and Collaboration: Cloud data warehouses support data sharing and collaboration across teams and organizations, allowing users to securely share data and insights with internal and external stakeholders.

Data Warehouse Hardware

- Data warehouse designers and administrators should always have forethought about the Input/Output performance while implementing a data warehouse.
- The data warehouse operations mainly consist of huge data loads and index builds, generation of materialized views, and queries over large volumes of data.
- The elemental I/O system of a data warehouse should be built to meet these heavy requirements.



Number of CPUs

- CPUs are responsible for the calculation abilities of a data warehouse
- Parallel operations are CPU- intensive when compared to the serial operations
- The number of CPUs is based on the highest throughput.
- Vendors would usually provide benchmark numbers based on concurrent load for specific number of cores.

Data Warehouse Hardware

Memory of data warehouse

- Large sorts is an example of a memory-intensive operation
- The memory requirements of a data warehouse are not the same when compared to mission-critical OLTP applications
- The amount of memory is calculated based on the number of CPUs as stated below:
- Amount of memory in GB = 2 * Number of CPUs

Number of Disks

- The number of disks is based on the maximum throughput.
- The storage provider's specifications should be used to find out the throughput a disk array can withstand
- The number of disks is calculated as stated below:
- Number of disks = Throughput in MB/s / Individual controller throughput in MB/s

Disk Redundancy

- Data warehouses are the largest storage systems used widely across several enterprises. They have many disks which are liable to failure
- Disk redundancy is important to avoid the failure of the entire system in case of hardware malfunctioning
- Redundancy should be achieved based on the cost constraints and performance of a data warehouse. In case of failure of one disk the data is stored in another disk is always critical for a data warehouse

Data Warehouse Hardware

Plan for expansion

- The data in a data warehouse keeps on growing. The data warehouse designer should focus on the growth of the I/O system without hindering the I/O bandwidth.
- In order to prevent unwanted data from burdening the systems businesses should pay attention to the age and overall quality of the archived data. Depending on the needs of the businesses the archiving methods should be effectively performed periodically.

Data Warehouse Software

Data warehousing demands the following prominent features when deciding a platform for functioning that are listed below:

- A prospect of combining various management systems
- A possibility of enhancing the structure of the queries
- A possibility of improving the load processes

The below table states the software with the versions considered for a data warehouse

Vendor Name	Products and Version
IBM	InfoSphere Balanced Warehouse
SAP	NetWeaver BI (Business Warehouse) / SAP HANA
Teradata	Active Enterprise Data Warehouse, Data Warehouse Appliance, Data Mart Appliance, Teradata Vantage (Cloud offering)
Microsoft	SQL Server
Oracle	Optimized Warehouses, Database, Warehouse Builder
Sybase	Analytic Appliance, IQ (Sybase was acquired by SAP)
Netezza	Performance Server 1000 Series Data Warehousing Appliance 4.5

Data Warehouse Software

IBM

- IBM is extensively used for its customer care, offering solutions according to the client's needs. The distinguishing features of IBM are
- DB2 a product of IBM is well-known for its efficient enterprise database
- Complete integration with WebSphere, Cognos and Infosphere
- IBM appliances are adaptable for supporting mixed workloads - OLAP, ETL, and Ad-hoc queries

SAP

- SAP is another leading ERP solution provider. The features of SAP are:
 - Row-oriented storage capabilities
 - Wide range of EDW services by SAP itself or significant partnerships
 - Complete integration with service-based architecture capabilities, BI, middleware, and desktop software

Teradata

- Among all the vendors Teradata wins hands down.
- The following are the other significant reasons:
 - EDW packaging, pricing, and licensing options
 - Solutions prepared by the Teradata designers are well understood and supportive for the external applications and middleware
 - Extensive range of products and services
 - Extraordinary scalability allows the Teradata customers to expand data warehouses through Massively Parallel Processors(MPP)

Microsoft

- SQL Server is a platform worth observing. It can be easily deployed in different BI and EDW technologies
- Cost-based query optimization, indexing, partitioning and caching
- Efficiently adjustable database that can be used by businesses for various size and trade
- May not be suitable to meet Enterprise Data Warehouse needs

Oracle

- Oracle Database and Oracle Warehouse Builder are the tools that have let Oracle achieve its position among the vendors. Its features are
- The EDWs can be expanded to various nodes that can sustain hundreds of terabytes
- Provides its customers a range of choice among different Optimized Warehouse EDW appliances
- Oracle solutions can be deployed on different hardware and software platforms, they are not restrained by a standard configuration

Netezza

- The constant growth of Netezza's portfolio has placed it among the top data warehousing platforms. Its features are
- Exclusive model of physical data storage implementing.
- Powerful in-database analytic framework
- Efficient BI, OLAP, query, and advanced analytic frameworks

Data warehouse Appliances

Large and small enterprises need to meet the demands of the ever-expanding workload requirements and continuous delivery of ROI while reducing the budgets. Appliances that are purpose-built devices that pre-integrate hardware and software to address particular workloads are promptly being considered

Whole Technology Stack Appliances

- Netezza was the first vendor to provide a data warehouse appliance in 2002. The Netezza performance server consolidates database and operating system software with server and storage hardware in a complete data warehouse platform
- DATAAllegro started in mid-2005 was similar to Netezza. Microsoft acquired DATAAllegro in 2008 combined DATAAllegro's MPP architecture into SQL Server, which runs on commodity hardware.
- SAP HANA (High-performance ANalytic Appliance) is a multi-model database that stores data in its memory instead of keeping it on a disk providing excellent performance.

Partial-Technology Stack Appliances

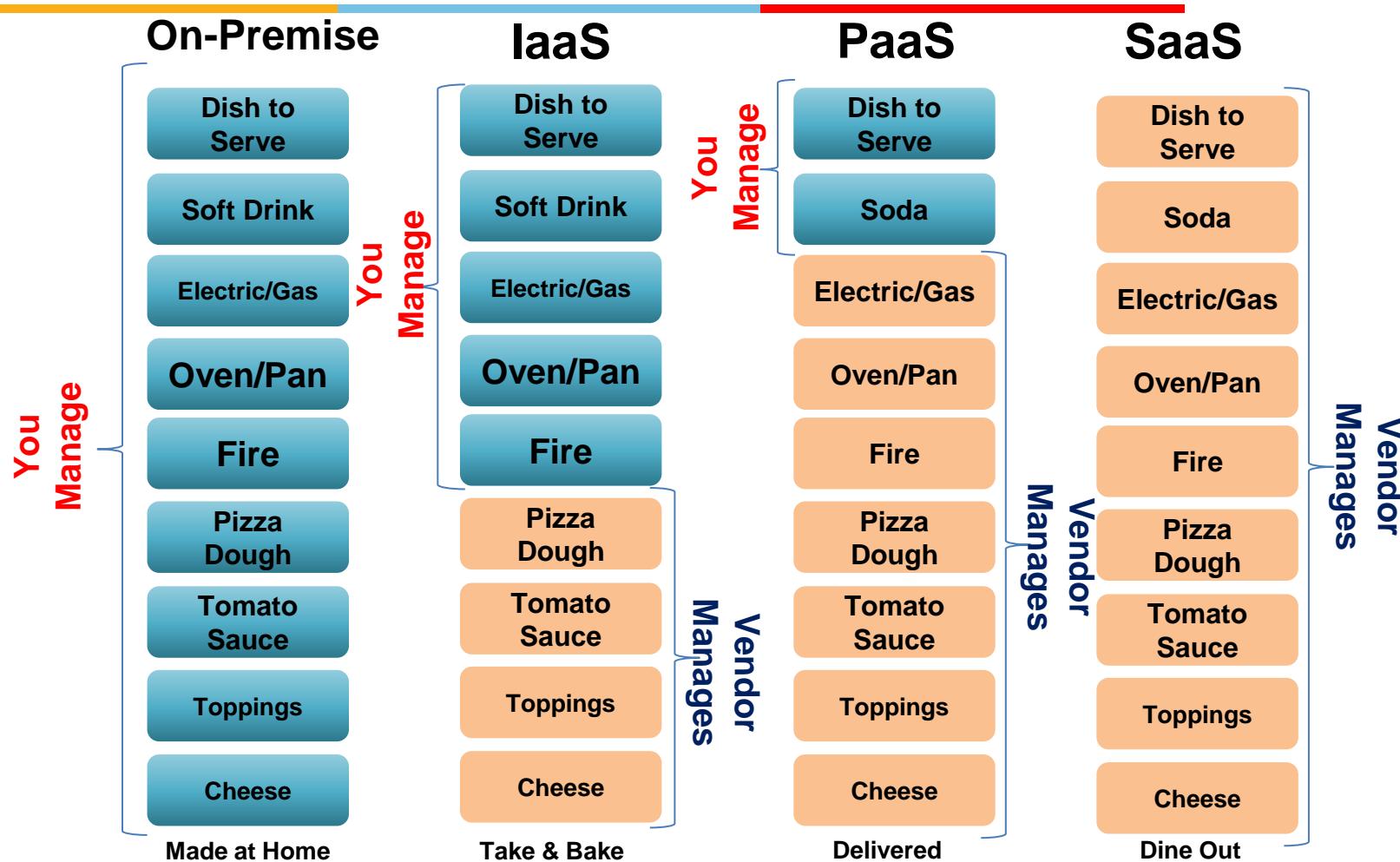
- Launched in 2006, new vendors evolved with database management systems. These include DBMS based on the relational model(Greenplum & Kognitio)
- These appliances typically call themselves software appliances since they focus on the database software

Deployment Service Models



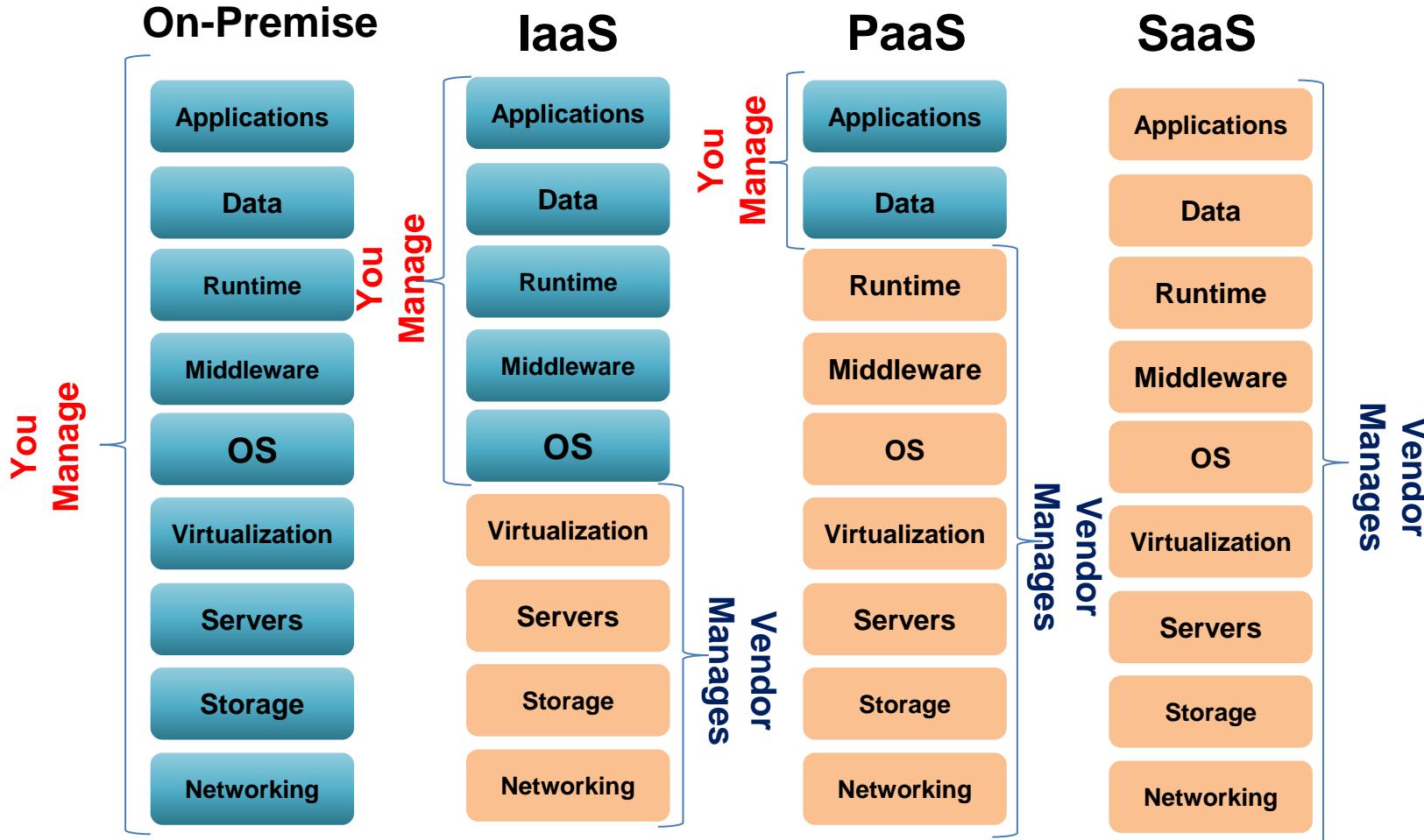
Understanding the “Pizza” Way....

Deployment Service Models



Understanding the “Pizza” Way....

Deployment Service Models



Deploying an on-premise Data Warehouse

Deploying an on-premise data warehouse involves several steps to ensure a smooth and efficient implementation.

1. Planning and Preparation:

- Define the objectives and scope of the data warehouse project, including the data sources, analytical requirements, and business goals.
- Assess hardware requirements, including server specifications, storage capacity, and network infrastructure.
- Determine the architecture and design of the data warehouse, considering factors such as data modeling, ETL processes, and security requirements.

2. Installation of SQL Server 2022 (Example):

- Install SQL Server 2022 on the designated server(s) following the recommended installation guidelines provided by Microsoft.
- Configure the SQL Server instance, including database engine, analysis services, integration services, and reporting services, based on the requirements of the data warehouse.

Deploying an on-premise Data Warehouse

3. Database Design and Configuration:

- Design the database schema for the data warehouse, including dimension tables, fact tables, and any additional structures required for data storage and analysis.
- Configure database settings, such as collation, filegroups, and partitioning, to optimize performance and scalability.
- Set up security permissions and access controls to ensure data confidentiality and integrity.

4. Data Integration and ETL:

- Develop Extract, Transform, Load (ETL) processes to extract data from source systems, transform it into the desired format, and load it into the data warehouse.
- Use SQL Server Integration Services (SSIS) or other ETL tools to automate data extraction, transformation, and loading tasks.
- Implement error handling and data quality checks to ensure the accuracy and completeness of the data in the warehouse.

Deploying an on-premise Data Warehouse

5. Indexing and Optimization:

- Create indexes on tables and views to improve query performance and optimize data retrieval.
- Monitor and analyze query execution plans to identify performance bottlenecks and optimize SQL queries for better performance.
- Implement partitioning and compression techniques to manage large volumes of data efficiently and reduce storage costs.

6. Backup and Recovery:

- Set up regular backup schedules to protect data against loss or corruption, including full backups, differential backups, and transaction log backups.
- Test backup and recovery procedures to ensure data integrity and recoverability in case of system failures or disasters.

7. Monitoring and Maintenance:

- Implement monitoring tools and alerts to track system performance, resource utilization, and data integrity issues.
- Schedule regular maintenance tasks, such as index rebuilds, statistics updates, and database reorganizations, to optimize performance and ensure data consistency.
- Review and analyze system logs and performance metrics to identify trends, troubleshoot issues, and proactively address potential problems.

Deploying Cloud Data Warehouse

1. Cloud Provider Selection:

- Choose a cloud provider that offers SQL Server 2022 as a managed service, such as Microsoft Azure SQL Database or Amazon Redshift or Snowflake.
- Consider factors like performance, scalability, security features, pricing, and integration with other cloud services.

2. Provisioning Resources:

- Create a new SQL Server instance or database in the chosen cloud environment by selecting appropriate offering (single Database instance, Managed Instance or Deploy IaaS)
- Configure the instance or database with the appropriate specifications, including compute capacity, storage size, and performance tiers.

3. Database Design and Configuration:

- Design the database schema for the cloud data warehouse, considering factors like data modeling, partitioning, and indexing.
- Configure database settings, such as collation, filegroups, and security options, based on best practices and compliance requirements.

Deploying Cloud Data Warehouse

4. Data Migration and ETL:

- Migrate existing data from on-premise databases or other sources to the cloud data warehouse using SQL Server Data Migration Service or other data migration tools.
- Develop Extract, Transform, Load (ETL) processes to automate data ingestion, transformation, and loading tasks into the cloud data warehouse.
- Utilize SQL Server Integration Services (SSIS) or cloud-native ETL services for data integration and transformation.

5. Security and Access Control:

- Implement security measures to protect data in transit and at rest, including encryption, access control policies, and multi-factor authentication.
- Configure role-based access control (RBAC) to restrict access to sensitive data and grant permissions based on user roles and responsibilities.

6. Performance Optimization:

- Optimize database performance by creating indexes, partitioning tables, and optimizing query execution plans.
- Monitor resource utilization, query performance, and system metrics using built-in monitoring tools or third-party monitoring solutions.
- Scale resources up or down dynamically based on workload demands using auto-scaling or manual scaling features.

Deploying Cloud Data Warehouse

7. Backup and Disaster Recovery:

- Set up regular backups and snapshots to protect data against loss or corruption, leveraging built-in backup and restore capabilities of the cloud platform.
- Implement disaster recovery solutions, such as geo-replication or failover clustering, to ensure business continuity in case of outages or disasters.

8. Continuous Monitoring and Maintenance:

- Monitor the health and performance of the cloud data warehouse continuously using cloud-native monitoring tools, alerts, and dashboards.
- Schedule regular maintenance tasks, such as index maintenance, database reorganizations, and software updates, to optimize performance and reliability.

9. Integration with Other Cloud Services:

- Integrate the cloud data warehouse with other cloud services and applications, such as analytics platforms, business intelligence tools, and data visualization tools.
- Leverage APIs, connectors, and data pipelines for seamless data integration and interoperability across the cloud ecosystem.

Snowflake Implementation Aspects*

* These aspects are based on experience only and may not be exhaustive list of things to be factored in.

Process & Solution Architecture Aspects

- Defining R&R for each role/groups
- Purpose and usage - Where does Snowflake fits into solution Architecture
- Type of Use cases – Batch/Real-Time
- Data Pipeline Orchestration
- Data Pipeline Monitoring/Alerting
- Funding Model (Centralized Funding vs. Distributed Model)
- Bill back model (in case of distributed funding)
- Snowflake Region for Production and DR
- DR – RPO/RTO Requirements
- Path-to-Production Landscape
- CI/CD Tools integrations

Technology Aspects

- Snowflake Account Name Standard
- Database/Schema Hierarchy and Naming Standard (**Remember:** Replication can be done at DB level only)
- Where will Table and Views reside (same schema/database or different schema/database)
- Logical Data Models (assuming Snowflake being used for Transformation as well)
- External Stage Standard
- Time Travel Configuration
- Data Sharing requirements (Internal & External)
- Access Security Model
- Network security
- Warehouse Segmentation & Naming Standard
- Credits/Cost Monitoring
- (If migrating to Snowflake from existing solution) Downstream application changes

Microsoft Azure Cloud Database

Microsoft Azure Cloud Database

Microsoft Azure offers a variety of cloud database services designed to meet different needs, including relational databases, NoSQL databases, data warehousing, and more.

Azure SQL Database:

- Azure SQL Database is a fully managed relational database service based on the SQL Server database engine.
- It offers built-in high availability, automated backups, and security features such as data encryption and threat detection.
- Azure SQL Database supports both traditional T-SQL queries and modern development practices like microservices and containers.
- It provides scalability options ranging from single databases to elastic pools for managing multiple databases with varying resource requirements.

Azure Cosmos DB:

- Azure Cosmos DB is a globally distributed, multi-model database service designed for building highly responsive and scalable applications.
- It supports multiple data models including document, key-value, graph, and column-family.
- Cosmos DB offers guaranteed low latency and high availability.
- It provides automatic and instant scaling across regions, enabling developers to reach global audiences with minimal effort.
- Cosmos DB is suitable for a wide range of use cases including IoT, gaming, retail, and finance where low latency and global reach are critical requirements.

Microsoft Azure Cloud Database

Azure Database for MySQL/PostgreSQL/MariaDB:

- Azure offers managed database services for popular open-source databases including MySQL, PostgreSQL, and MariaDB.
- These services provide fully managed, enterprise-ready databases with built-in high availability, automated backups, and security features.
- Azure Database for MySQL/PostgreSQL/MariaDB supports flexible scaling options to accommodate changing workloads.
- Developers can leverage familiar tools and frameworks to build applications on these platforms while benefiting from the scalability and reliability of Azure cloud infrastructure.

Azure Synapse Analytics:

- Formerly known as Azure SQL Data Warehouse, Azure Synapse Analytics is a cloud-based analytics service designed for processing and analyzing large volumes of data.
- It offers integration with various data sources including relational databases, data lakes, and streaming data.
- Azure Synapse Analytics provides built-in data integration, data warehousing, and big data analytics capabilities in a single service.
- It supports both serverless and provisioned resources to meet different performance and cost requirements.
- Microsoft is now transitioning to Fabric overhauling complete Data Platform architecture.

Azure Data Studio

- Azure Data Studio is a cross-platform database tool that provides a modern and lightweight environment for database development and management.
- It supports various database systems, including Microsoft SQL Server, Azure SQL Database, PostgreSQL, MySQL, and more.
- Cross-Platform: Azure Data Studio is available for Windows, macOS, and Linux operating systems, making it accessible to a wide range of users.
- Supports Multiple Database Systems: While originally focused on Microsoft SQL Server, Azure Data Studio has expanded its support to other database systems including Azure SQL Database, PostgreSQL, MySQL, and more.
- Modern Interface: Azure Data Studio features a modern user interface with customizable themes, intuitive navigation, and support for high DPI displays.
- Extensible: It supports extensions that enhance its functionality, allowing users to customize their experience by adding features, integrations, and tools.

Azure Data Studio-Key Features

Query Editor: Azure Data Studio provides a powerful SQL query editor with features like syntax highlighting, IntelliSense (code completion), code snippets, and query execution.

Object Explorer: The Object Explorer allows users to browse database objects, such as tables, views, stored procedures, and more. It provides a hierarchical view of the database schema and allows for easy navigation and management.

Integrated Terminal: It includes an integrated terminal that allows users to run shell commands and scripts directly within Azure Data Studio. This can be particularly useful for tasks like version control, database migrations, and running administrative commands.

Source Control Integration: Azure Data Studio integrates with source control systems like Git, enabling version control for database scripts and other artifacts. Users can commit changes, view history, and manage branches directly from within the tool.

Notebooks: It supports Jupyter Notebooks, allowing users to create and run interactive notebooks containing code, text, images, and visualizations. This feature is useful for data exploration, documentation, and collaboration.

Azure Data Studio-Key Features

Task Automation: Azure Data Studio supports task automation through the use of customizable dashboards and notebooks. Users can create and schedule tasks, such as backups, maintenance jobs, and data imports, using built-in or custom scripts.

Extensions Marketplace: The Extensions Marketplace offers a wide range of extensions developed by the community and Microsoft. These extensions add functionality such as additional language support, database management tools, productivity enhancements, and more.

Data Visualization: Azure Data Studio includes built-in charting capabilities for visualizing query results. Users can create bar charts, line charts, pie charts, and other visualizations to analyze data directly within the tool.

Connection Management: It provides flexible connection management options, allowing users to connect to multiple database servers simultaneously. Connections can be saved for quick access, and connection profiles can be customized with specific settings and preferences.

Azure Synapse Analytics (formerly call Azure SQL Data Warehouse)

- Azure Synapse is a fully managed and scalable cloud-based data warehousing service provided by Microsoft Azure.
- It is designed to handle large volumes of data and enable businesses to analyze and derive insights from their data effectively.

Key Features are:

Massive Scalability

- Azure Synapse is built for scalability, allowing users to resource based on workload requirements.
- Users can allocate resources to handle varying workloads, scaling up or down as needed downtime though.

Columnar Storage:

- Data in Azure Synapse is stored in a columnar format, which improves query performance by reducing the amount of data read from disk and optimizing data compression.
- Columnar storage is particularly well-suited for analytical queries that involve scanning large datasets.

Azure Synapse

Distributed Query Processing:

- Azure Synapse uses a distributed architecture to parallelize query processing across multiple compute nodes.
- Queries are divided into smaller tasks and executed in parallel across compute nodes, enabling fast query performance even for complex analytical queries.

Integration with Azure Ecosystem:

- Azure Synapse integrates seamlessly with other Azure services such as Azure Data Lake Storage, Azure Data Factory, Azure Analysis Services, and Power BI.
- Users can leverage these services to build end-to-end data analytics solutions and extract maximum value from their data.

Security and Compliance:

- Azure Synapse provides robust security features including data encryption, role-based access control (RBAC), and auditing.
- It supports compliance with various industry standards and regulations such as GDPR, HIPAA, SOC, and ISO.

Built-in Management Tools:

- Azure Synapse includes built-in management tools for monitoring and optimizing performance.
- Users can monitor resource utilization, query performance, and data distribution using dynamic management views (DMVs) and performance monitoring dashboards.

PolyBase Integration:

- Azure Synapse integrates with PolyBase, a technology that enables seamless data integration between relational databases and big data sources such as Hadoop and Azure Blob Storage.



BITS Pilani

Pilani Campus

DW Project Management

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda



DW Project Management:

- 9.1. DW development lifecycle
- 9.2. Agile techniques for DW

Development lifecycle



- Data Warehousing is a flow process that gathers and manages structured and unstructured data from numerous sources into a Centralized repository.
- It makes actionable business decisions.
- With all of your data in one location, you can perform analysis, reporting, and discover important insights at a variety of combination levels.
- The phases (and their relationships) that a data warehouse system goes through from the time it is conceptualized until it is no longer usable are referred to as the data warehouse life-cycle.

Data warehouse Development Lifecycle:

- Requirements Gathering: Understand the business requirements and objectives for the data warehouse. This involves working closely with stakeholders to identify the data sources, data types, frequency of data updates, and desired analytics and reporting capabilities.
- Data Sourcing and Integration: Identify and extract data from various source systems such as transactional databases, spreadsheets, flat files, etc. This data is then transformed and integrated to ensure consistency, quality, and compatibility with the data warehouse schema.
- Data Modeling: Design the structure of the data warehouse including dimensions, facts, relationships, and hierarchies. This involves creating entity-relationship diagrams, dimensional models (e.g., star schema or snowflake schema), and defining metadata.
- ETL (Extract, Transform, Load): Develop and implement ETL processes to extract data from source systems, transform it according to the data warehouse schema and business rules, and load it into the data warehouse. This step also involves data cleansing, normalization, and aggregation.

Data warehouse Development Lifecycle:

- Data Storage: Determine the appropriate storage infrastructure for the data warehouse, considering factors such as scalability, performance, and cost. This may involve relational databases, data lakes, or cloud-based storage solutions.
- Data Quality Assurance: Implement processes to monitor and ensure the quality of data in the data warehouse. This includes data profiling, validation, cleansing, and error handling to maintain data integrity and accuracy.
- Metadata Management: Establish a metadata repository to document and manage the metadata associated with the data warehouse, including data definitions, data lineage, transformations, and business rules.

Data warehouse Development Lifecycle:

- Indexing and Partitioning: Optimize the performance of the data warehouse by creating indexes, partitions, clustering and other optimization techniques to facilitate efficient data retrieval and analysis.
- Security and Access Control: Implement security measures to protect sensitive data and regulate access to the data warehouse. This includes user authentication, authorization, encryption, and auditing.
- Data Governance: Define policies and procedures for managing and governing the data within the data warehouse. This involves establishing roles and responsibilities, data ownership, compliance requirements, and data stewardship processes.
- Testing and Validation: Conduct thorough testing of the data warehouse to ensure that it meets the functional and performance requirements. This includes unit testing, integration testing, regression testing, and user acceptance testing.
- Deployment and Rollout: Deploy the data warehouse into production and make it available to end-users. This may involve data migration, system configuration, and user training.
- Monitoring and Maintenance: Establish ongoing monitoring and maintenance processes to ensure the continued performance, reliability, and security of the data warehouse. This includes monitoring system metrics, resolving issues, applying patches and updates, and optimizing performance as needed.
- Evolution and Enhancement: Continuously evolve and enhance the data warehouse to meet changing business needs and technological advancements. This involves gathering feedback from users, identifying opportunities for improvement, and implementing enhancements through iterative development cycles.

Requirement Gathering

- This is one of the tough phase and is very critical to the success of your data warehouse project development.
- Complete and clear requirement is must for success of any data warehouse project.
- Requirement gathering can happen as one-to-one meetings with end-users or client.
- This involves working closely with stakeholders to identify the data sources, data types, frequency of data updates, and desired analytics and reporting capabilities
- In this process, you will be preparing **Business requirement document (BRD)**.
- Finally, after gathering the requirements, the data modeler begins to identify the dimensions, facts, and combinations depending on the needs.
- We can describe this as the data warehouse's general design.

Requirement Gathering

Business requirement document (BRD)- sample format

Table of Contents:

- Introduction
- Purpose
- Scope
- Business Objectives
- Stakeholders
- Current System Overview
- Business Requirements
- Data Requirements
- Reporting and Analysis Requirements
- User Interface Requirements
- Functional Requirements
- Non-Functional Requirements
- Constraints
- Assumptions
- Risks
- Project Timeline
- Sign-off

Requirement Gathering

The requirements gathering includes but not limited to following steps:

- Understand the existing models and prepare questionnaire for new data warehouse users
- Ask specific questions regarding project rather than high level
- Get information on the different data sources for data warehouse
- Collect information on the frequency of data loading and incremental load details
- Get information about list of reports that are needs to be built as part of this data warehouse projects.

Requirement Gathering- A Sample questionnaire for new data warehouse users

1. Understanding Existing Models:

- Are you currently using any data warehouse or reporting systems?
- What are the limitations or challenges you face with the existing systems?
- Are there any specific features or functionalities from existing systems that you would like to see in the new data warehouse?

2. Project-Specific Questions:

- What are the primary objectives or goals of your project?
- What specific business processes or functions will the data warehouse support?
- Are there any regulatory or compliance requirements that the data warehouse must adhere to?

3. Data Sources:

- Can you provide a list of data sources that need to be integrated into the data warehouse?
- Are there any third-party systems or external data sources that need to be accessed?
- Do you have any preferences or requirements regarding data formats or structures?

4. Data Loading Frequency:

- How frequently do you expect data to be loaded into the data warehouse (e.g., daily, weekly, monthly)?
- Are there any specific time windows or schedules for data loading that need to be considered?
- Do you anticipate any challenges or constraints related to data loading processes?

5. Incremental Load Details:

- Are there any incremental data updates or changes that need to be captured?
- What criteria or key indicators will be used to identify new or updated data?
- Do you have any requirements or preferences regarding data deduplication or data reconciliation processes?

6. Reports Requirements:

- Can you provide a list of reports that are essential for your project?
- What specific metrics, KPIs, or dimensions do you need to include in these reports?
- Are there any specific visualization or formatting preferences for the reports?

Step 1: Requirement Gathering

Example Problem: Retail Sales Analysis and Forecasting

1. Understanding Business Requirements:

- Objective: The company aims to increase sales revenue by analyzing past sales data, identifying trends, and predicting future sales.
- Stakeholders: Sales managers, marketing team, finance department, IT department.
- Data Sources: Transactional databases (point of sale systems), customer relationship management (CRM) software, inventory management systems.
- Data Types: Sales transactions, customer demographics, product information, inventory levels.
- Frequency of Data Updates: Daily updates for sales transactions, weekly updates for inventory levels, monthly updates for customer demographics.
- Desired Analytics and Reporting Capabilities: Sales trend analysis, product performance analysis, customer segmentation, sales forecasting, dashboards for real-time monitoring.

DW Project Management

Requirement Gathering

Example Problem: Retail Sales Analysis and Forecasting

1) Understand the existing models and prepare questionnaire for new data warehouse users:

Existing Models: The retail company may already have databases or systems storing sales transactions, customer information, and product data.

Questionnaire: Prepare a questionnaire to gather insights from existing users or stakeholders.

Questions may include:

- What data sources are currently used for sales analysis?
- What are the limitations or challenges with existing systems?
- What additional data or functionalities would be beneficial for sales analysis and forecasting?

2) Ask specific questions regarding the project rather than high level:

During discussions with stakeholders, it's crucial to ask specific questions to gather detailed requirements and avoid ambiguity. In our example:

Specific Questions:

- How do you define a "successful" sales analysis and forecasting system?
- What specific metrics or KPIs are most important for monitoring sales performance?
- Can you provide examples of past scenarios where accurate sales forecasting would have been beneficial?

DW Project Management

Requirement Gathering

Example Problem: Retail Sales Analysis and Forecasting

3) Get information on the different data sources for the data warehouse:

Identifying and understanding the various data sources is fundamental for designing an effective data warehouse. In our example:

Data Sources:

- Transactional databases: Point of sale systems, online sales platforms.
- CRM software: Customer demographics, purchase history.
- Inventory management systems: Product information, inventory levels.

4) Collect information on the frequency of data loading and incremental load details:

Understanding the frequency of data updates and the process for incremental loads helps in designing efficient ETL processes. In our example:

Data Loading Frequency:

- Daily updates for sales transactions.
- Weekly updates for inventory levels.
- Monthly updates for customer demographics.

Incremental Load Details: Determine how new data is identified and loaded into the data warehouse without duplicating existing records.

DW Project Management

Requirement Gathering

Example Problem: Retail Sales Analysis and Forecasting

5) Get information about the list of reports that are needed to be built as part of this data warehouse project:

Identifying the required reports and analytics upfront ensures that the data warehouse is designed to meet specific business needs. In our example:

List of Reports:

- Sales trend analysis by store, product category, and time period.
- Product performance analysis based on sales volume and revenue.
- Customer segmentation for targeted marketing.
- Sales forecasting reports using historical data and predictive models.

Additional Requirements: Determine any specific formatting, visualization preferences, or interactive features desired for the reports.

Requirement Gathering

Example Problem: Developing a Healthcare Analytics Data Warehouse

- The healthcare industry generates vast amounts of data from various sources such as electronic health records (EHRs), medical devices, insurance claims, and research studies.
- However, this data is often siloed and underutilized, hindering opportunities for improving patient care, operational efficiency, and healthcare outcomes.
- To address this challenge, the healthcare organization aims to develop a healthcare analytics data warehouse.
- This data warehouse will integrate and analyze diverse healthcare data to support clinical decision-making, patient care management, resource allocation, and healthcare policy formulation.

1) Understand the existing models and prepare questionnaire for new data warehouse users:

Existing Models: Review existing healthcare information systems such as EHRs, laboratory information systems, and billing systems.

Questionnaire:

- What are the current data sources used for clinical decision-making and healthcare management?
- What challenges do healthcare providers face in accessing and analyzing patient data?
- What additional data elements or analytics capabilities would enhance patient care and operational efficiency?

DW Project Management

Requirement Gathering

Example Problem: Developing a Healthcare Analytics Data Warehouse

2) Ask specific questions regarding the project rather than high level:

Specific Questions:

- What specific clinical metrics or indicators are critical for measuring patient outcomes?
- How do healthcare providers currently identify and address gaps in patient care?
- Can you provide examples of past scenarios where timely access to patient data would have improved clinical decision-making?

3) Get information on the different data sources for the data warehouse:

Data Sources:

- Electronic Health Records (EHRs): Patient demographics, medical history, diagnoses, medications.
- Medical Devices: Vital signs, imaging data, wearable device data.
- Insurance Claims: Billing and reimbursement data.
- Research Studies: Clinical trial data, outcomes research data.

Requirement Gathering

Example Problem: Developing a Healthcare Analytics Data Warehouse

4) Collect information on the frequency of data loading and incremental load details:

Data Loading Frequency:

- Real-time updates for patient vitals and telemetry data.
- Daily or weekly updates for EHR data.
- Periodic updates for insurance claims and research studies.

Incremental Load Details: Understand how new patient data is captured and integrated into the data warehouse without duplicating existing records.

5) Get information about the list of reports that are needed to be built as part of this data warehouse project:

List of Reports:

- Patient health status dashboards for clinicians.
- Clinical performance indicators for quality improvement.
- Population health analytics for disease management and preventive care.
- Financial analytics for revenue cycle management and cost containment.

Additional Requirements: Identify any specific regulatory reporting requirements or clinical guidelines that need to be addressed in the reports.

Step 2: Data Sourcing and Integration

Example: developing a Healthcare Analytics Data Warehouse

- Identify and extract data from various source systems such as transactional databases, spreadsheets, flat files, etc.
- This data is then transformed and integrated to ensure consistency, quality, and compatibility with the data warehouse schema.

1. Identify Data Sources:

- **Electronic Health Records (EHRs):** These systems contain comprehensive patient information including demographics, medical history, diagnoses, medications, lab results, and treatment plans.
- **Medical Devices:** Data from medical devices such as vital signs monitors, imaging equipment, and wearable devices provide real-time patient data such as heart rate, blood pressure, and activity levels.
- **Insurance Claims:** Billing and reimbursement data from insurance claims provide insights into healthcare services rendered, costs, and payments.
- **Research Studies:** Data from clinical trials, outcomes research, and epidemiological studies contribute to understanding disease patterns, treatment effectiveness, and healthcare outcomes.

DW Project Management



Data Sourcing and Integration - developing a Healthcare Analytics Data Warehouse

2. Extract Data:

- Utilize SQL queries to extract patient demographics and medical history from the EHR database.
- Use an API to extract real-time vital signs data from medical devices connected to the hospital's network.
- Extract billing data from insurance claims spreadsheets stored in a secure file share.
- Pull clinical trial data using a web scraping tool to extract information from research study websites.

3. Data Transformation:

- **Data Cleansing:** Identify and rectify inconsistencies, errors, and missing values in the extracted data. This may involve techniques such as deduplication, standardization, and validation.
 - Remove duplicate patient records and correct misspelled diagnoses in the extracted EHR data
- **Data Integration:** Integrate data from multiple source systems into a unified format. This may require mapping data elements to a common schema or data model.
 - Integrate patient demographics from the EHR system with billing data from insurance claims to create a comprehensive patient profile.
- **Data Enrichment:** Enhance the extracted data with additional contextual information or derived attributes. For example, aggregating patient encounters to calculate average length of stay or deriving patient age from birthdates.
 - Calculate body mass index (BMI) from patient height and weight data to enrich the patient demographics.
- **Data Standardization:** Standardize data formats, units, and terminology to ensure consistency and interoperability across different data sources.
 - For instance, standardizing medication names or diagnosis codes using industry-standard terminologies like SNOMED CT or ICD-10.

DW Project Management

Data Sourcing and Integration - developing a Healthcare Analytics Data Warehouse

4. Ensure Data Quality:

- Establish data quality rules and metrics to measure the accuracy, completeness, consistency, and timeliness of the transformed data.
 - ***Eg: ensure that all patient records have a valid date of birth and gender.***
 - Implement data quality checks and validation processes to identify and address anomalies or discrepancies.
 - ***Validate that vital signs data falls within clinically reasonable ranges to detect potential errors.***
 - Monitor data quality over time and implement data governance practices to maintain data integrity.
 - ***Monitor data completeness by ensuring that all required fields are populated for each patient encounter.***

DW Project Management

Data Sourcing and Integration - developing a Healthcare Analytics Data Warehouse

5. Compatibility with Data Warehouse Schema:

- Map the transformed data to the data warehouse schema, which typically includes dimensions, facts, and relationships.
- **Map the transformed patient demographics and medical history data to dimension tables in the data warehouse schema.**
- **Populate fact tables with aggregated billing data to support financial analytics.**
- Ensure that all data elements are represented in a consistent format compatible with the target data warehouse platform (e.g., PostgreSQL or Amazon Redshift).
- Validate data integrity and schema compatibility to ensure that the data warehouse can effectively support analytics and reporting requirements.

Data Modelling- developing a Healthcare Analytics Data Warehouse

- Design the structure of the data warehouse including dimensions, facts, relationships, and hierarchies.
- This involves creating entity-relationship diagrams, dimensional models (e.g., star schema or snowflake schema), and defining metadata.

1. Entity-Relationship Diagrams (ERDs):

Entity-Relationship Diagrams help visualize the entities (data tables) and their relationships in the data warehouse.

Example:

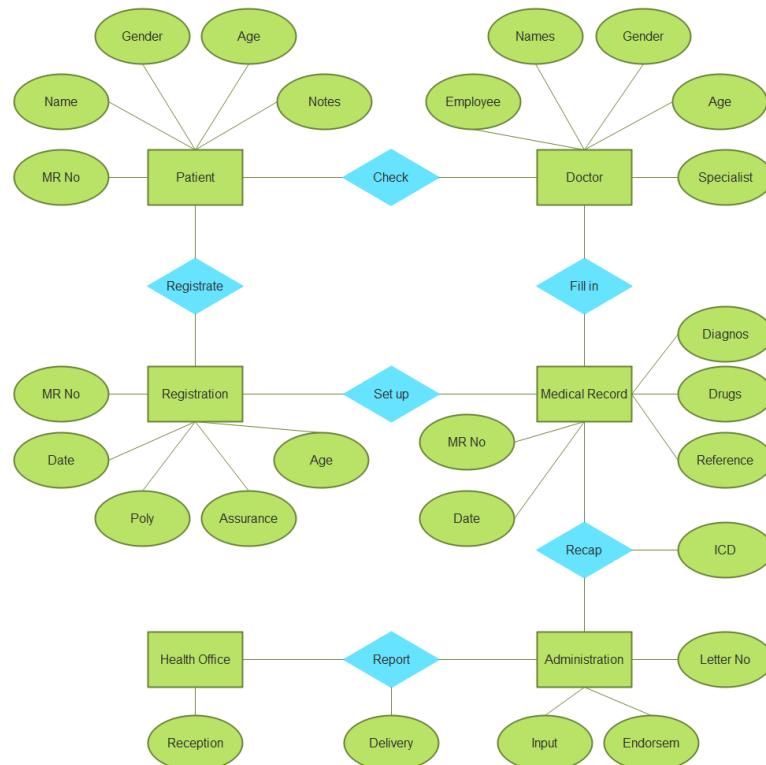
Entities: Patient, Encounter, Diagnosis, Medication, Procedure.

Relationships:

- A Patient can have multiple Encounters.
- An Encounter can have multiple Diagnoses, Medications, and Procedures.

Attributes: Each entity has specific attributes (e.g., PatientID, EncounterDate, DiagnosisCode) that describe the data being captured.

ER Diagram of Hospital Information System



DW Project Management



Data Modelling- developing a Healthcare Analytics Data Warehouse

2. Dimensional Models (Star Schema or Snowflake Schema):

Dimensional modeling involves organizing data into fact tables and dimension tables to support efficient querying and analysis.

Example (Star Schema):

Fact Table: Encounter_Fact

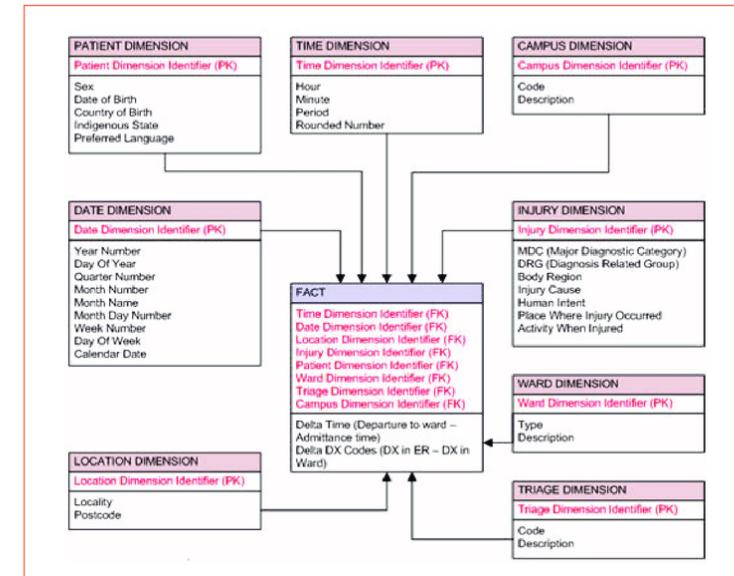
- Contains metrics such as EncounterID, PatientID, EncounterDate, EncounterCost.

Dimension Tables:

- Patient_Dim: Contains attributes related to patients (e.g., PatientID, Gender, Age).
- Diagnosis_Dim: Contains attributes related to diagnoses (e.g., DiagnosisCode, DiagnosisDescription).
- Medication_Dim: Contains attributes related to medications (e.g., MedicationCode, MedicationName).
- Procedure_Dim: Contains attributes related to procedures (e.g., ProcedureCode, ProcedureDescription).

Relationships:

- Fact table (Encounter_Fact) connects to dimension tables (Patient_Dim, Diagnosis_Dim, Medication_Dim, Procedure_Dim) via foreign key relationships.



Data Modelling- developing a Healthcare Analytics Data Warehouse

3. Hierarchies:

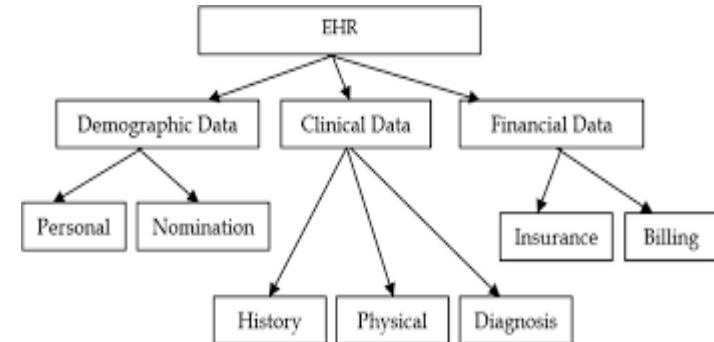
Hierarchies define the relationships between data elements at different levels of granularity, allowing for drill-down and roll-up analysis.

Time Hierarchy: Year > Quarter > Month > Day

Location Hierarchy: Country > Region > City > Facility

Diagnosis Hierarchy: Disease Category > Disease

Subcategory > Specific Diagnosis



4. Defining Metadata:

Metadata provides descriptive information about the data elements in the data warehouse, including their definitions, source systems, and usage.

Patient Dimension Metadata:

- Name: Patient_Dim
- Description: Contains attributes related to patients.
- Attributes: PatientID, Gender, Age, etc.
- Source System: EHR (Epic, Cerner).
- Usage: Used for patient demographics analysis, population health management.

ETL (Extract, Transform, Load): Healthcare Analytics Data Warehouse

1. Extract:

- Extracting Data from Source Systems: Use ETL tools or scripts to extract data from source systems such as Electronic Health Records (EHRs), medical devices, and insurance claims databases.
- **Example: Extract patient demographics, diagnoses, and procedures data from the hospital's EHR system using SQL queries.**

2. Transform:

- Data Cleansing: Cleanse the extracted data to remove inconsistencies, errors, and duplicates.
- **Example: Remove duplicate patient records and correct misspelled diagnoses in the extracted EHR data.**
- Data Normalization: Standardize data formats, units, and terminologies to ensure consistency and interoperability across different data sources.
- **Example: Standardize diagnosis codes using the International Classification of Diseases (ICD) terminology across all clinical and billing datasets.**
- Data Aggregation: Aggregate data at different levels of granularity to support analysis and reporting.
- **Example: Aggregate patient encounter data to calculate average length of stay or total cost per patient.**

DW Project Management

3. Load:

Loading Data into the Data Warehouse: Load the transformed data into the appropriate tables in the data warehouse, following the data warehouse schema.

Example: Load the cleaned and standardized patient demographics, diagnoses, procedures, and billing data into respective dimension and fact tables in the data warehouse using ETL tools or scripts.

DW Project Management

Data Storage

Data Storage: Determine the appropriate storage infrastructure for the data warehouse, considering factors such as scalability, performance, and cost. This may involve relational databases, data lakes, or cloud-based storage solutions.

Evaluate Data Storage Requirements:

- Determine the storage capacity needed to accommodate the growing volume of patient records, medical images, lab results, and other healthcare data.

Choose Storage Technologies:

- Consider various storage technologies such as relational databases, data lakes, or cloud-based storage solutions based on the data storage requirements and use cases.
- Relational databases (e.g., MySQL, PostgreSQL) are suitable for structured data with predefined schemas and complex querying requirements.
- Data lakes (e.g., Amazon S3, Azure Data Lake Storage) are ideal for storing large volumes of diverse data types in their native format, allowing for flexible data exploration and analysis.
- Cloud-based storage solutions offer scalability, reliability, and cost-effectiveness, making them well-suited for healthcare organizations with dynamic data storage needs.
- Example: Select Amazon S3 as the primary storage solution for the healthcare analytics data warehouse due to its scalability, durability, and integration with analytics services.

Data Storage

Data Lakes:

- Description: Data lakes are repositories that store vast amounts of raw data in its native format until it's needed for analysis.

Example:

- **Scenario: The healthcare organization wants to store a variety of healthcare data types including structured EHR data, unstructured medical imaging data, and streaming telemetry data from medical devices.**

Design Data Storage Architecture:

- Define the data storage architecture, including the logical and physical structure of the data warehouse.
- Determine the data modeling approach (e.g., dimensional modeling, star schema) to organize and optimize the data for analytics and reporting.
- Design data partitioning, indexing, and compression strategies to improve data retrieval performance and minimize storage costs.
- **Example: Implement a dimensional modeling approach with a star schema design, where healthcare data is organized into fact tables (e.g., patient encounters, procedures) and dimension tables (e.g., patients, providers, diagnoses) for efficient querying and analysis.**

DW Project Management

Data Storage

Implement Data Storage Solution:

- Set up and configure the chosen data storage solution, including database servers, data lakes, or cloud storage accounts.
- Define data access controls and security policies to protect sensitive healthcare information and ensure compliance with regulatory requirements such as HIPAA.
- Load initial data into the data warehouse from various source systems, ensuring data integrity and consistency.
- **Example: Deploy Amazon Redshift as the data warehouse platform, configure security groups and IAM roles to control access to healthcare data, and use AWS Glue for data ingestion and ETL processes.**

Test and Optimize Data Storage Performance:

- Conduct performance testing to evaluate the responsiveness and scalability of the data storage infrastructure under different workloads.
- Monitor and analyze data storage metrics such as latency, throughput, and storage utilization to identify performance bottlenecks and optimize resource allocation.
- Fine-tune database configurations, query optimizations, and indexing strategies to improve data retrieval performance and reduce query latency.
- **Example: Use AWS CloudWatch to monitor Redshift cluster performance metrics, identify slow-performing queries using Amazon Redshift Advisor, and optimize query execution plans using query optimization techniques.**

Data Quality Assurance

Data Profiling:

- Data profiling involves analyzing the structure, content, and quality of the data to understand its characteristics and identify potential issues.
- In the healthcare analytics data warehouse, data profiling can be performed on various sources such as electronic health records (EHRs), medical devices, insurance claims, etc.
- **Example: Profiling the EHR data to identify missing values, outliers, inconsistencies, and patterns in patient demographics, diagnoses, procedures, medications, etc.**

Data Validation:

- Data validation ensures that the data conforms to predefined rules, standards, and constraints.
- Define validation rules based on the requirements and regulations in the healthcare domain, such as HIPAA regulations for patient privacy and data security.
- **Example: Validating insurance claim data to ensure that the patient information, diagnosis codes, procedure codes, and billing amounts are accurate and comply with regulatory standards.**

DW Project Management



Data Quality Assurance

Data Cleansing:

- Data cleansing involves identifying and correcting errors, inconsistencies, and inaccuracies in the data to improve its quality.
- Develop automated processes or scripts to cleanse the data, such as removing duplicate records, correcting misspellings, standardizing formats, and resolving inconsistencies.
- **Example: Cleansing the medication data by standardizing drug names, removing special characters, and resolving discrepancies between different data sources.**

Error Handling:

- Implement robust error handling mechanisms to detect and handle data errors and exceptions during data processing.
- Define error handling procedures to log, report, and resolve data issues in a timely manner to prevent downstream impacts.
- **Example: Implementing error handling routines to handle data loading failures, data transformation errors, and data validation failures, ensuring that erroneous data is flagged for review and correction.**

Data Integrity and Accuracy:

- Ensure data integrity and accuracy by implementing measures to prevent data corruption, loss, or unauthorized modifications.
- Establish data governance policies, access controls, and audit trails to enforce data integrity and maintain data quality over time.
- **Example: Implementing role-based access controls to restrict access to sensitive patient data, ensuring that only authorized users can view or modify patient records.**

Indexing and partitioning/clustering

Data Analysis and Profiling:

- Conduct a comprehensive analysis and profiling of the healthcare data to identify key attributes, data distribution patterns, and access patterns.
- For example, analyze the frequency of access to patient demographics, diagnosis codes, treatment procedures, and medication records in the electronic health records (EHRs).

Identify Indexing Candidates:

- Identify the attributes and columns in the healthcare datasets that are frequently queried or used for filtering, grouping, or joining operations.
- For instance, attributes such as patient ID, admission date, diagnosis code, and provider ID are likely candidates for indexing in the healthcare analytics data warehouse.

Select Appropriate Indexing Strategies:

- Select the appropriate indexing strategies based on the query patterns and workload characteristics observed during data analysis.
- For example, consider creating clustered indexes on columns frequently used in range queries (e.g., admission date for time-based analysis) and non-clustered indexes on columns used in equality or join operations (e.g., patient ID for patient-level analysis).

DW Project Management

Indexing and partitioning/clustering

Partitioning Design:

- Design the partitioning strategy to distribute and manage data across multiple storage units or partitions efficiently.
- In the healthcare analytics data warehouse, partitioning can be based on time intervals (e.g., monthly or quarterly partitions for historical patient data), geographical regions (e.g., partitions for different hospitals or clinics), or functional domains (e.g., partitions for clinical, administrative, and financial data).

Partitioning Key Selection:

- Select the appropriate partitioning key(s) based on the query patterns, data distribution, and partitioning strategy.
- For example, partitioning by admission date may be suitable for time-based analysis, while partitioning by hospital ID may be appropriate for distributed query processing across different healthcare facilities.

Implement Indexing and Partitioning:

- Implement indexing and partitioning based on the selected strategies and design decisions. This involves creating indexes on the identified columns and defining partitions according to the chosen partitioning key(s).
- For example, create clustered and non-clustered indexes on patient ID, admission date, diagnosis code, and provider ID columns, and partition the data warehouse tables by admission date and hospital ID.

Performance Tuning and Monitoring:

- Continuously monitor and tune the indexing and partitioning strategies based on workload changes, query performance metrics, and data growth trends.
- For example, periodically review index fragmentation, partition distribution, and query execution plans to optimize the performance and scalability of the healthcare analytics data warehouse.

DW Project Management

Security and Access Control

Data Security Assessment:

- Conduct a comprehensive assessment of the security requirements and risks associated with healthcare data.
- Identify sensitive information such as patient demographics, medical history, and treatment records that require protection to ensure compliance with regulations like HIPAA (Health Insurance Portability and Accountability Act) and GDPR (General Data Protection Regulation).

Define Security Policies and Procedures:

- Establish security policies and procedures that govern access to healthcare data within the data warehouse.
- Define roles and responsibilities for data stewards, administrators, and end-users, and outline protocols for data encryption, authentication, authorization, and auditing.

Access Control Design:

- Design access control mechanisms to enforce security policies and regulate user access to healthcare data. Implement role-based access control (RBAC) to assign permissions based on user roles and responsibilities.
- For example, clinicians may have access to patient records for treatment purposes, while administrators may have access to system configuration settings.

Encryption and Data Masking:

- Implement encryption techniques to protect sensitive healthcare data both at rest and in transit. Use encryption algorithms to secure data stored in the data warehouse, and implement secure communication protocols (e.g., SSL/TLS) for data transmission between client applications and the data warehouse environment.
- Additionally, consider data masking techniques to anonymize or pseudonymize sensitive information in non-production environments.

DW Project Management



Security and Access Control

Authentication and Authorization:

- Implement robust authentication mechanisms to verify the identity of users accessing the healthcare analytics data warehouse.
- Utilize strong authentication methods such as multi-factor authentication (MFA) to enhance security.
- Implement authorization controls to restrict access to specific data based on user roles, privileges, and data sensitivity levels.

Auditing and Monitoring:

- Establish auditing and monitoring processes to track user activities and detect potential security breaches or unauthorized access attempts.
- Implement audit trails to log user actions, data modifications, and system events within the data warehouse environment.
- Regularly review audit logs and perform security audits to ensure compliance with regulatory requirements and security best practices.

Data Governance and Compliance:

- Integrate security controls into the overall data governance framework to ensure compliance with regulatory requirements and industry standards.
- Establish data governance policies and procedures for managing data access, security incidents, data quality, and privacy concerns within the healthcare analytics data warehouse.

DW Project Management



Testing and Validation:

Unit Testing:

- Example: In a healthcare data warehouse, unit testing involves testing individual components or modules responsible for data extraction, transformation, loading (ETL), and storage. For instance, a unit test might verify that the data transformation logic accurately converts diagnosis codes from ICD-9 to ICD-10 format without loss of information.

Integration Testing:

- Example: Integration testing in a healthcare data warehouse involves validating the end-to-end flow of data from source systems to the data warehouse and ensuring seamless integration with downstream analytics applications. For example, integration testing might verify that patient demographics, lab results, and medication records from disparate source systems are successfully integrated and available for analysis within the data warehouse.

Regression Testing:

- Example: In a healthcare data warehouse, regression testing ensures that new changes or enhancements to the data warehouse do not adversely impact existing functionality. For instance, after implementing a software update or modifying ETL processes, regression testing might be conducted to verify that previously validated reports and analytics queries still produce accurate results without any unintended side effects.

DW Project Management

Testing and Validation:

User Acceptance Testing (UAT):

- Example: User acceptance testing involves validating the healthcare data warehouse from the perspective of end-users, such as clinicians, administrators, and analysts.
- For example, clinicians might conduct UAT to verify that they can easily access and analyze patient data using predefined reports and dashboards within the data warehouse to support clinical decision-making and quality improvement initiatives.

Performance Testing:

- Example: Performance testing evaluates the responsiveness, scalability, and throughput of the healthcare data warehouse under various load conditions.
- For example, performance testing might involve simulating concurrent user access and executing complex analytical queries against large healthcare datasets to assess query response times, system resource utilization, and overall system performance.

Security Testing:

- Example: Security testing ensures that the healthcare data warehouse is adequately protected against unauthorized access, data breaches, and security vulnerabilities.
- For instance, security testing might involve assessing user authentication mechanisms, access controls, encryption protocols, and audit logging functionality to verify compliance with regulatory requirements such as HIPAA and GDPR.

DW Project Management

Data Migration:

- During deployment, data migration involves transferring data from legacy systems, such as electronic health records (EHRs) or claims processing systems, to the new data warehouse.
- For instance, migrating patient demographics, medical histories, treatment records, and billing data from existing databases to the new data warehouse ensures that historical data is available for analysis and reporting.

Performance Tuning:

- Performance tuning involves optimizing the performance and scalability of the healthcare data warehouse to meet the demands of end-users and analytical workloads.
- For example, tuning database indexes, optimizing query execution plans, and fine-tuning system parameters based on performance benchmarks and workload patterns ensures that the data warehouse can handle concurrent user access and execute complex analytical queries efficiently.

Change Management:

- Example: Change management processes ensure smooth deployment and rollout of the healthcare data warehouse while minimizing disruption to operations.
- For example, coordinating communication and collaboration between IT teams, end-users, and stakeholders, managing software releases and updates, and providing support and assistance during the transition phase helps facilitate the adoption of the new data warehouse and mitigate resistance to change.



BITS Pilani

Pilani Campus

Cloud Data Warehousing

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda

Cloud Data Warehousing

1. Topologies
2. Provider Selection
3. Configuration, Management
4. Loading & Unloading Data from/into Snowflake
5. Snowflake Time Travel Capability
6. Stored Procedure in Snowflake
7. Snowflake Tasks

Cloud Data Warehousing

- Cloud data warehousing refers to the practice of storing and managing data in a cloud-based environment, typically using cloud computing services and infrastructure.
- Unlike traditional data warehouses that rely on on-premises hardware and software, cloud data warehouses leverage the scalability, flexibility, and cost-effectiveness of cloud platforms.
- In a cloud data warehousing setup, data is stored in distributed storage systems offered by cloud providers, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.
- These platforms offer various services tailored for data warehousing, including scalable storage solutions, managed database services, and advanced analytics tools.

Cloud data warehousing: Key features and benefits

Scalability: Cloud data warehouses can scale up or down based on demand, allowing organizations to handle large volumes of data and accommodate fluctuating workloads.

RetailCo experiences seasonal spikes in sales, especially during holidays. With a cloud data warehouse like Google BigQuery or Snowflake, RetailCo can seamlessly scale their data storage and processing capacity to accommodate increased data volumes and analytical workloads during peak periods. For example, during Black Friday sales, RetailCo can easily scale up their cloud data warehouse to handle the surge in transaction data.

Flexibility: Cloud data warehouses support diverse data types and sources, enabling organizations to integrate structured and unstructured data from multiple sources, including databases, streaming platforms, and IoT devices.

RetailCo collects data from various sources, including online transactions, in-store purchases, customer reviews, and social media interactions. With a cloud data warehouse, RetailCo can integrate these diverse data sources into a unified data repository. They can then perform advanced analytics, such as customer segmentation, product recommendations, and sentiment analysis, to gain insights into customer behavior and preferences.

Cloud data warehousing: Key features and benefits

Cost-effectiveness:

- Cloud data warehouses operate on a pay-as-you-go model, where organizations only pay for the resources they use.
- This eliminates the need for upfront hardware investments and allows for cost optimization based on usage patterns.
- RetailCo leverages a pay-as-you-go pricing model with their cloud data warehouse provider. They only pay for the storage and processing resources they use, allowing them to optimize costs based on their data usage patterns.
- For example, RetailCo can scale down their data warehouse during periods of low activity to reduce costs, and scale up as needed during busy periods.

Performance:

- Cloud data warehouses leverage distributed computing architectures and parallel processing techniques to deliver high-performance analytics and query processing capabilities.
- RetailCo needs fast and responsive analytics to support real-time decision-making. With a cloud data warehouse's distributed computing architecture and parallel processing capabilities, RetailCo can execute complex queries and analytical workloads with high performance.
- For instance, RetailCo can quickly analyze sales trends, identify top-selling products, and forecast demand using advanced analytics algorithms.

Cloud data warehousing: Key features and benefits

Integration with other cloud services: Cloud data warehouses seamlessly integrate with other cloud services, such as data lakes, machine learning platforms, and business intelligence tools, enabling organizations to build end-to-end data pipelines and extract actionable insights from their data.

RetailCo integrates their cloud data warehouse with other cloud services, such as Google Analytics for web traffic analysis, Salesforce for customer relationship management (CRM), and Tableau for data visualization. This integration enables RetailCo to build end-to-end data pipelines, automate data workflows, and visualize insights in interactive dashboards.

Security and compliance: Cloud data warehouses provide robust security features, including encryption, access controls, and compliance certifications, to ensure the confidentiality, integrity, and availability of data.

RetailCo ensures the security and compliance of their data warehouse by implementing encryption, access controls, and auditing mechanisms provided by their cloud provider. They comply with industry regulations such as GDPR (General Data Protection Regulation) and PCI DSS (Payment Card Industry Data Security Standard) to protect customer data and maintain trust.

Cloud Data Warehousing: Topologies

Single Cloud Data Warehouse:

- In this topology, all data storage and processing are centralized within a single cloud data warehouse platform offered by a single cloud provider.
- This approach simplifies management and integration but may limit flexibility and vendor lock-in risks.
- Example: Amazon Redshift, Google BigQuery, Snowflake

Multi-Cloud Data Warehousing:

- In this topology, organizations leverage data warehousing solutions from multiple cloud service providers simultaneously.
- By distributing workloads across different cloud platforms, organizations can mitigate the risk of vendor lock-in and take advantage of each provider's unique features and pricing models.
- However, managing data across multiple clouds introduces complexities related to data integration, security, and governance.

Cloud Data Warehousing: Topologies

Hybrid Data Warehousing:

1. Hybrid Data Warehousing combines on-premises infrastructure with cloud-based data warehouse solutions.
2. Organizations can retain sensitive or legacy data on-premises while leveraging the scalability and flexibility of cloud data warehouses for newer, less sensitive data.
3. This topology offers greater flexibility and control over data placement but requires robust integration and synchronization mechanisms between on-premises and cloud environments.

Federated Data Warehousing:

1. In a Federated Data Warehousing architecture, data remains distributed across multiple disparate systems, and queries are executed across these distributed data sources in a federated manner.
2. Organizations can maintain data sovereignty and autonomy over their data sources while still enabling unified query processing and analysis.
3. Federated data access requires specialized query engines or middleware to coordinate and optimize query execution across distributed data sources.

Cloud Data Warehousing: Topologies

Edge Data Warehousing:

1. Edge Data Warehousing brings data processing and analytics closer to the data sources, such as IoT devices, edge computing nodes, or branch offices.
2. This topology reduces latency by processing data locally and sending only aggregated or summarized data to the centralized data warehouse for further analysis.
3. Edge Data Warehousing is suitable for use cases that require real-time or low-latency analytics, such as industrial IoT, retail analytics, and remote monitoring.

Data Virtualization:

1. Data Virtualization enables organizations to access and query data from multiple disparate sources, including databases, data lakes, and SaaS applications, without physically consolidating the data into a single repository.
2. Virtualized data views are created on-the-fly, providing a unified and consistent view of the data across sources.
3. This topology offers agility and flexibility in accessing and integrating diverse data sources but may introduce performance overhead and complexity in query optimization.

Cloud Data Warehousing: Topologies

Decentralized Data Warehousing:

- Decentralized Data Warehousing distributes data warehousing capabilities across multiple autonomous units or business units within an organization.
- Each unit maintains its own data warehouse instance tailored to its specific requirements and analytics needs.
- This topology offers autonomy and customization at the departmental level but may result in data silos and challenges in data governance and standardization.

Cloud Computing Deployment Models

There are four main deployment models for cloud computing: public, private, hybrid, and community.

- Public clouds are owned and operated by a third-party service provider, which delivers its services over the Internet. Here, the vendor offers cloud services to numerous clients in the public cloud who, in the back end, share the same hardware. Private clouds are owned and operated by a single organization, which can be either an enterprise or a service provider.
- Hybrid clouds are a combination of public and private clouds that are connected using technology like VPNs or data encryption. Community clouds are shared by a group of organizations with similar requirements. When governments from many nations establish a shared services division that houses all of the government's IT, we frequently see an example of a community cloud.
- The type of deployment model you choose will depend on your specific needs and requirements. For example, if you require high levels of security and privacy, then you would likely choose a private or hybrid cloud deployment model. If cost is a primary concern, then you would likely choose a public cloud deployment model.

Cloud Vendors

The three top cloud vendors are Microsoft Azure, AWS, and Google Cloud Platform. Each one offers a different set of features and services, so it's important to choose the right one for your needs.

- Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. Azure was announced in October 2008 and released on February 1, 2010, as Windows Azure, before being renamed to Microsoft Azure on March 25th 2014.
- AWS, on the other hand, is a subsidiary of Amazon that provides on-demand cloud computing platforms to individuals, companies, and governments on a paid subscription basis with a free tier option available for 12 months. AWS's services include computing, storage, database, security, and identity management among others.
- Finally, Google Cloud Platform is a provider of cloud computing that offers both Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Some of their IaaS offerings include preconfigured virtual machines, storage options (such as SQL or NoSQL databases), DNS serving, and content delivery.



Snowflake – Data Loading

External Stage & Internal Stage

- Snowflake refers to the location of data files in cloud storage as a Stage.
- There are two types of Stage in Snowflake:
 - **External Stage:** This is Cloud Storage managed by your Enterprise
 - **Internal Stage:** This is Cloud Storage contained within your Snowflake Account
- Supported Cloud Storage Account:
 - AWS S3
 - Google Storage Account
 - Azure Storage Account
- Irrespective of CSP where Snowflake account is hosted, data can be loaded from either of supported CSPs. For example: If your Snowflake account is hosted on Azure and data needs to be loaded from AWS S3, this is fully supported.

Note: You cannot access data held in archival cloud storage classes that requires restoration before it can be retrieved

Internal Stage

➤ Snowflake maintains the following stage types in your account:

- **User:**

- A user stage is allocated to each user for storing files
- This stage type is designed to store files that are staged and managed by a single user but can be loaded into multiple tables
- User stages cannot be altered or dropped.

- **Table:**

- A table stage is available for each table created in Snowflake
- This stage type is designed to store files that are staged and managed by one or more users but only loaded into a single table
- Table stages cannot be altered or dropped.

- **Named:**

- A named internal stage is a database object created in a schema
- This stage type can store files that are staged and managed by one or more users and loaded into one or more tables
- Ability to create/drop/modify Named stage is controlled by security access

Defining Internal Stage

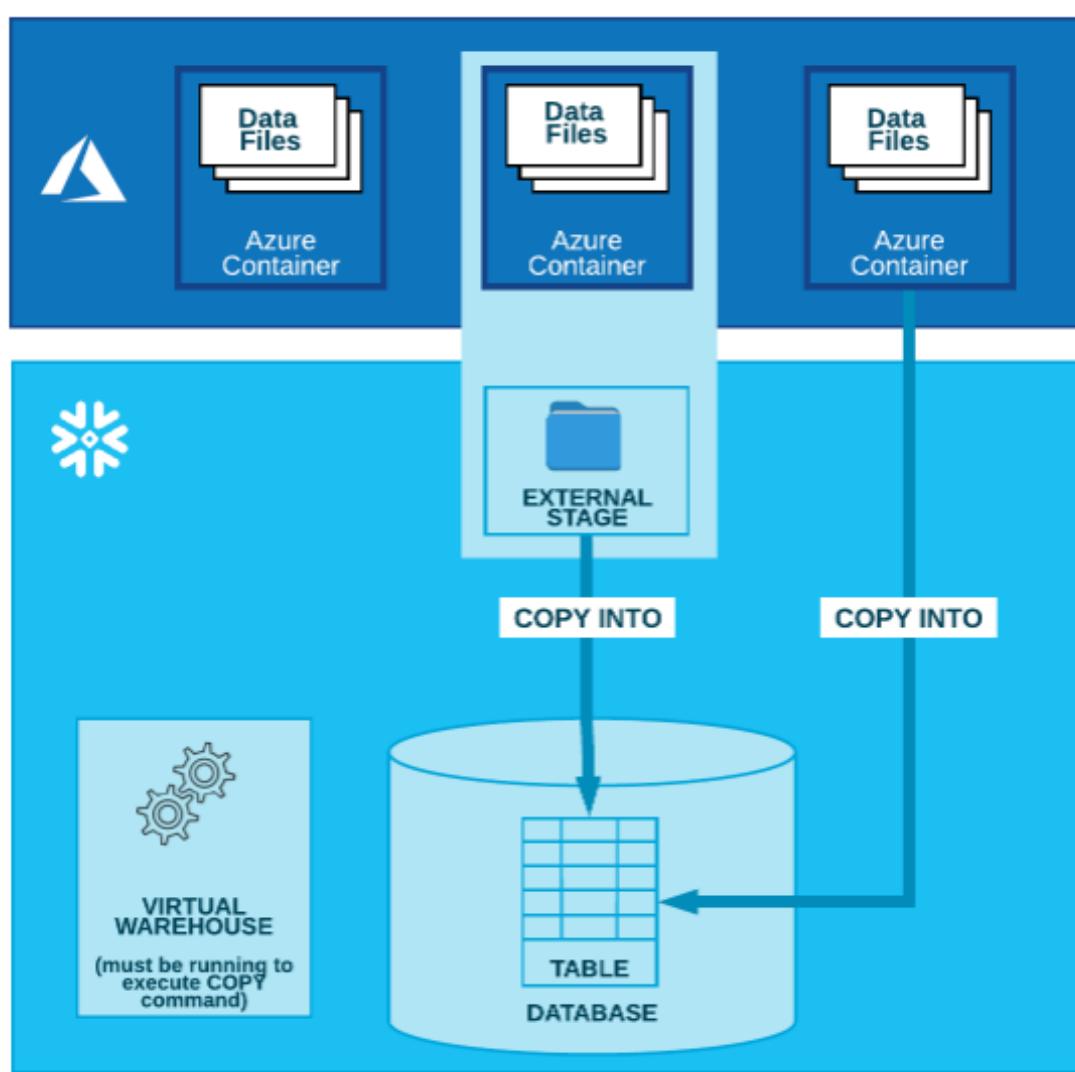
- In order to load data from internal stage, first stage has to be defined

```
create or replace stage <stage name>
file_format = (type = 'CSV'
field_delimiter = '|' skip_header = 1);
```

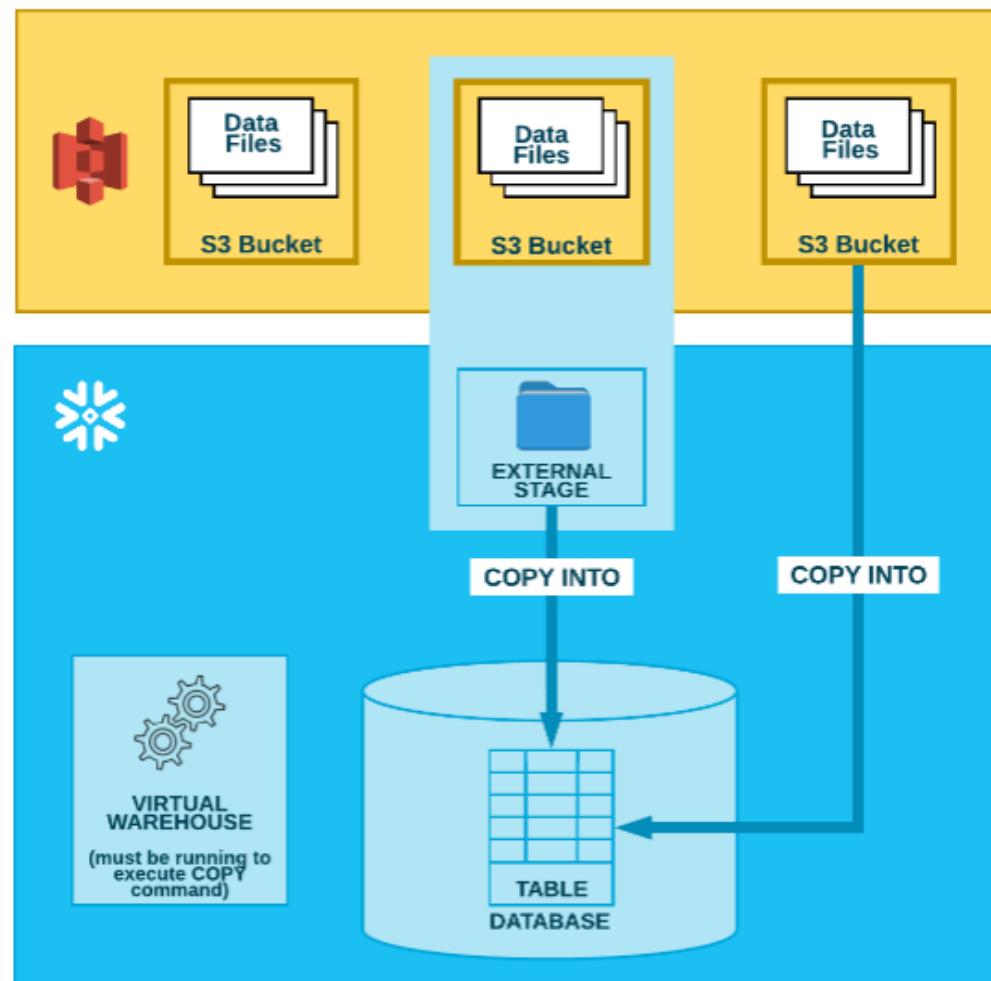
<upload file into internal stage>

```
copy into <table name> from @<stage
name>;
```

External Stage



External Stage



External Stage

- There are 2 options to load data via External Stage from Azure
 - **Option#1:** Using Storage Integration (Secured & Recommended for Enterprise)
 - Requires to create Storage Integration Configuration
 - Providing Access via Service Principal
 - Secured way to load data
 - **Option#2:** Using SAS Token (Recommended only for trials/learning/PoC etc.)
 - Requires to create Shared Access Signature (SAS) Token.
 - SAS Tokens have validity period.
 - Not Secured way to load data

Defining External Stage

- In order to load data from internal stage, first stage has to be defined

```
create or replace stage my_azure_stage
url='azure://<path to Storage Account>'
credentials=(azure_sas_token='<Token>')
file_format = (type = 'CSV' field_delimiter = '|' skip_header
= 1);
```

Supported File formats

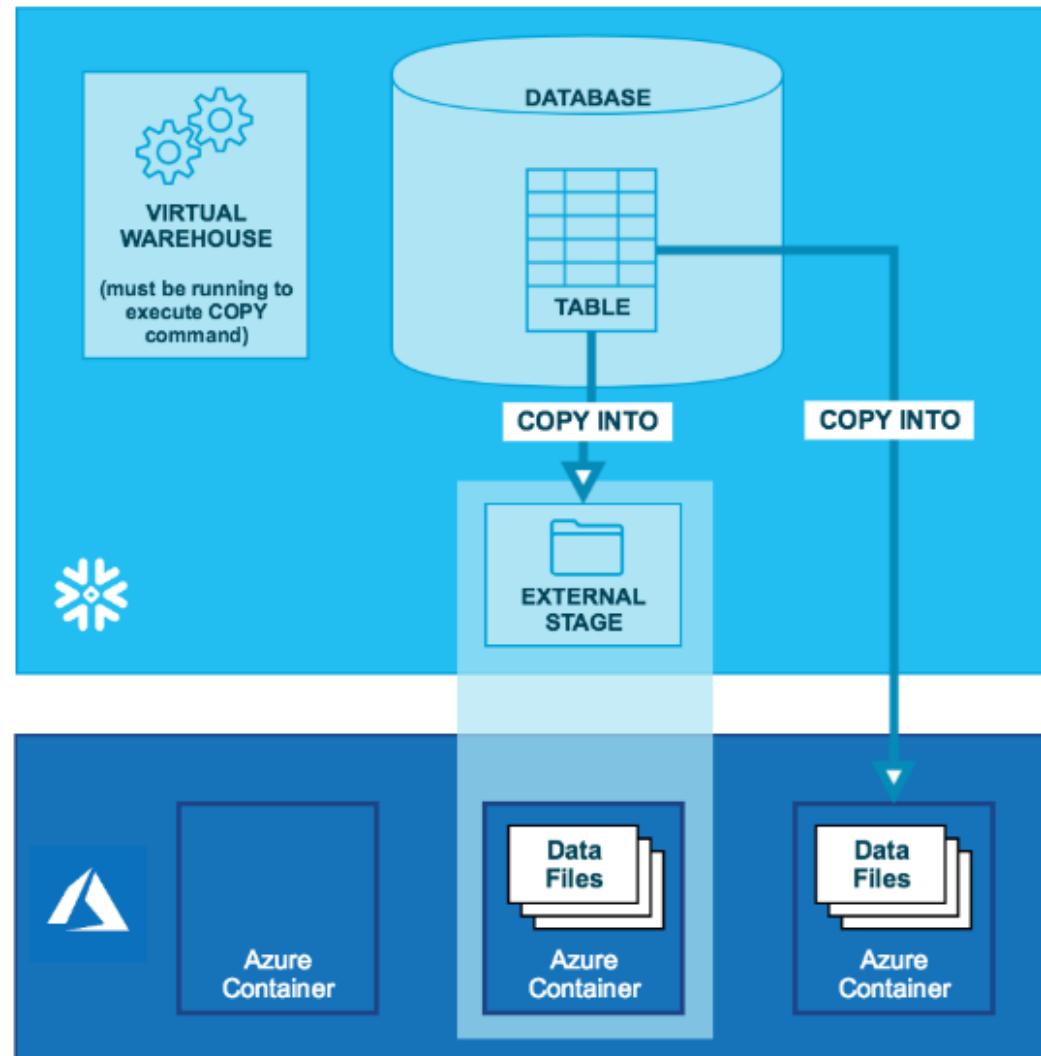
Structured/Semi-structured	Type	Notes
Structured	Delimited (CSV, TSV, etc.)	Any valid singlebyte delimiter is supported; default is comma (i.e. CSV).
Semi-structured	JSON	
	Avro	Includes automatic detection and processing of compressed Avro files.
	ORC	Includes automatic detection and processing of compressed ORC files.
	Parquet	Includes automatic detection and processing of compressed Parquet files. Currently, Snowflake supports the schema of Parquet files produced using the Parquet writer v1. Files produced using v2 of the writer are not supported.
	XML	Supported

Create File format

```
Create file format <format name>
type = csv field_delimiter = ',' skip_header = 1
```

[CREATE FILE FORMAT – Snowflake Documentation](#)

Unloading Data



Cloning

Zero-copy Cloning in Snowflake



- Clone Data in seconds
- Only pay for the unique data you store
- Very useful when refreshing TEST/DEV data with Production
- Cloning can be done at Database or Schema or Table level.

Time travel

- Helps query data in the past
- Creates clones of entire tables, schemas and databases at or before specific points
- Restore tables, schemas and databases that have been dropped

What happens once Time Travel Period is Elapses?

- Once the defined period of time has elapsed, the data is moved into Snowflake Fail-safe and these actions can no longer be performed.

Time Travel – Retention Period

- The standard retention period is 1 day (24 hours) and is automatically enabled for all Snowflake accounts
- For Snowflake Standard Edition, the retention period can be set to 0 (or unset back to the default of 1 day) at the account and object level (i.e. databases, schemas, and tables)
- For Snowflake Enterprise Edition (and higher):
 - For transient databases, schemas, and tables, the retention period can be set to 0 (or unset back to the default of 1 day). The same is also true for temporary tables.
 - For permanent databases, schemas, and tables, the retention period can be set to any value from 0 up to 90 days.

Time travel – Key Points

- Time Travel configuration can be done either at account level or database or schema or table level
- To see Time Travel Configuration at Account level, below command can be used

"show parameters like 'DATA_RETENTION_TIME_IN_DAYS' in account;"

- To change TIME TRAVEL configuration for an object, run below command

"alter table <table name> set data_retention_time_in_days=<number of days>"

- Or Time Travel can be enabled on an object at the time of creation as well

*create table <table name>(col1 number, col2 date)
data_retention_time_in_days=90*

- If a Database or Schema has a Retention Period, that duration is inherited by default for all objects created in the Database/Schema.
- Account level configuration is overridden by changing data retention configuration at either Database/Schema/Table level

Time travel - SQL extensions

- **SELECT** statements and **CREATE... CLONE** commands both have an **AT | BEFORE** Clause that can be provided (immediately after the object name).
- To pinpoint the particular **Historical Data** that you want to view, the Clause employs one of the following parameters:
 - **TIMESTAMP**
 - **OFFSET** (time difference in seconds from the present time)
 - **STATEMENT** (identifier for statement, e.g. query ID)
- To view historical data at specific time stamp:

```
select * from <table name> at(timestamp => 'Fri, 18 Nov 2022  
16:20:00'::timestamp_tz);
```

- To view historical data from a table 10mins ago:

```
select * from <table name> at(offset => -60*10);
```

- To view historical data prior execution of specific statement

```
select * from <table name> before(statement => '<query id>');
```

Restoring objects

- Calling UNDROP restores the object to its most recent state before the DROP command was issued

For example:

```
undrop table <table name>;  
undrop schema <schema name>;  
undrop database <database name>;
```

Stored Procedure

- Stored Procedures allows you to write procedural code that executes SQL
- You can use programmatic constructs to perform branching and looping.
- You can write Stored Procedure in one of the following languages:
 - Java (using Snowpark)
 - JavaScript
 - Python (using Snowpark)
 - Scala (using Snowpark)
 - Snowflake Scripting

Learning through an example - Javascript & Snowflake Scripting

JAVASCRIPT

```
CREATE OR REPLACE PROCEDURE JS_COPY_SP(stg
varchar(75))
RETURNS VARIANT
LANGUAGE JAVASCRIPT
EXECUTE AS CALLER
AS
$$

var copy_query = "copy into case1 from @" + STG + ";"
var copy_stmt = snowflake.createStatement({sqlText:
copy_query});
var copy_rs = copy_stmt.execute();

$$;
```

Snowflake Scripting (aka SQL SP)

```
create or replace procedure
SQL_COPY_SP(stg varchar(75))
returns varchar
language sql
as
declare
    v_query varchar:='copy into case1 from
@'||:stg;
begin
    execute immediate :v_query;
end;
```

TASK

A task can execute any one of the following types of SQL code:

- ✓ Single SQL statement
- ✓ Call to a stored procedure
- ✓ Tasks can be combined with table streams for continuous ELT workflows to process recently changed table rows
- ✓ Or any other type of work that needs to run on a specific cadence.

CREATING TASK in Snowflake (examples)

```
create task my_copy_task warehouse = mywh
schedule = '60 minute'
as
call my_unload_sp();
```

```
create task my_copy_task warehouse = mywh
schedule = 'USING CRON 0 * * * * America/Los_Angeles'
as
call my_unload_sp();
```

CREATING TASK in Snowflake (examples)

```
CREATE TASK t1
    USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE = 'XSMALL'
    SCHEDULE = '2 minute'
    AS
        EXECUTE IMMEDIATE
        $$

        DECLARE
            radius_of_circle float;
            area_of_circle float;

        BEGIN
            radius_of_circle := 3;
            area_of_circle := pi() * radius_of_circle * radius_of_circle;
            return area_of_circle;
        END;
        $$;
```

CREATING TASK in Snowflake (examples)

```
!set sql_delimiter=/
CREATE OR REPLACE TASK test_logging
    USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE = 'XSMALL'
    SCHEDULE = 'USING CRON 0 * * * * America/Los_Angeles'
AS
BEGIN
    ALTER SESSION SET TIMESTAMP_OUTPUT_FORMAT = 'YYYY-MM-DD
HH24:MI:SS.FF';
    SELECT CURRENT_TIMESTAMP;
END;/
!set sql_delimiter=";"
```



BITS Pilani

Pilani Campus

Real Time Data Warehousing

Instructor-in-Charge:
Sachin Arora, Guest faculty
BITS Pilani

Agenda



-
1. Business Need for Real-Time Data Warehousing
 2. Real-time ETL
 3. Use cases
 4. Benefits and Challenges
 5. Snowflake Real-Time Solution Architecture

Real-Time Data warehousing

- Real-time data warehousing refers to the process and technology involved in the immediate (or near-immediate) capture, cleansing, integration, and accessibility of data from various sources into a data warehouse environment for the purpose of timely analysis and reporting.
- Unlike traditional data warehousing, which often relies on batch processing of data at scheduled intervals (such as nightly or weekly), real-time data warehousing aims to minimize the latency between data generation and data availability for decision-making.
- The goal is to give businesses a competitive advantage by enabling faster responses to market changes, customer behavior, and other critical events

Business Need

- This approach supports operational and strategic business processes that require current data, such as dynamic pricing, real-time inventory management, immediate fraud detection, and up-to-the-minute financial reporting
 - The need for real-time data warehousing in modern business environments stems from several factors that are driven by the competitive, operational, and strategic demands of today's marketplaces
 - In today's volatile business environment, being able to respond to market conditions in real-time is not just an advantage—it's a necessity.
 - Imagine a supply chain application that can instantly reroute deliveries in response to sudden weather changes, or a financial system that can execute millisecond-sensitive trading strategies.
 - Real-time analytics is also revolutionizing customer engagement. Organizations can now track customer behavior in real-time, allowing them to offer more personalized experiences.
 - In an environment where customer preferences can change overnight, the ability to adapt marketing strategies in real-time can be the difference between market leadership and obsolescence.
-

Business Use Cases

1. Enhanced Decision Making: Retail Inventory Management

A retail chain uses real-time data warehousing to monitor sales data across all stores. By analyzing sales in real time, the company can quickly identify which products are selling faster than anticipated and adjust inventory levels or reorder stock immediately, avoiding stockouts and lost sales.

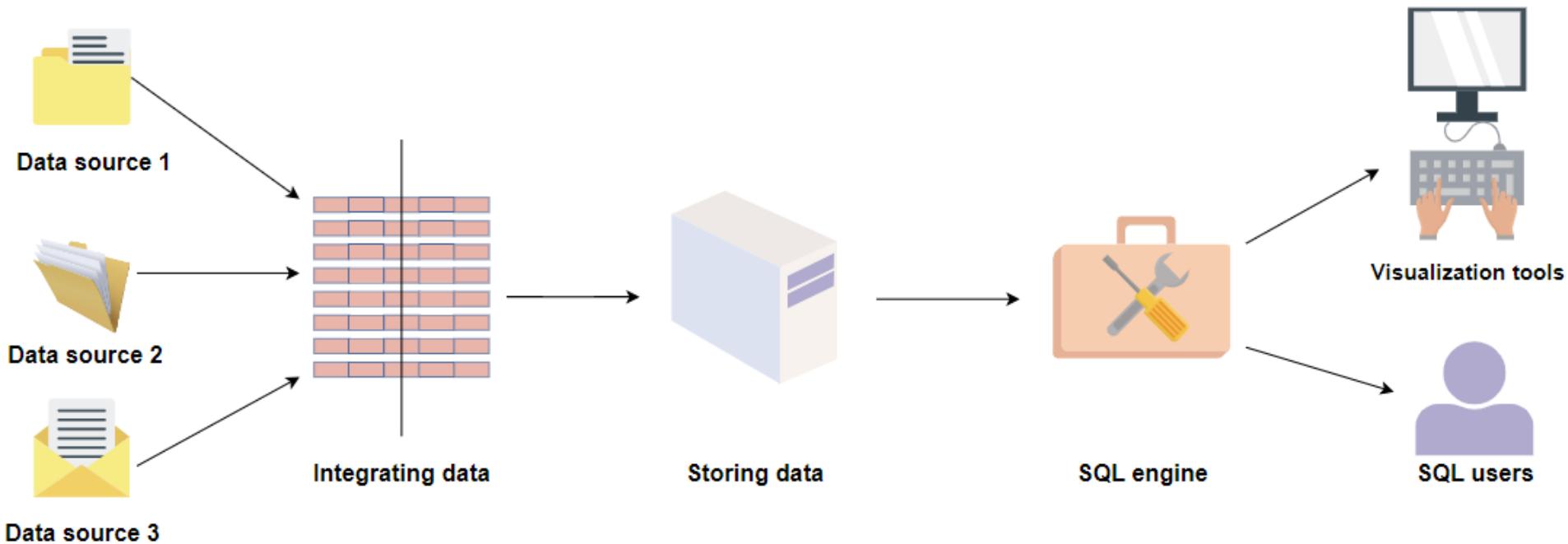
2. Improved Customer Experiences: E-commerce Personalization

An e-commerce platform leverages real-time data to personalize the shopping experience for each visitor. By analyzing current browsing behavior, past purchases, and real-time interactions, the platform can recommend products that the customer is likely to be interested in, increasing the likelihood of a purchase.

3. Operational Efficiency: Manufacturing Process Optimization

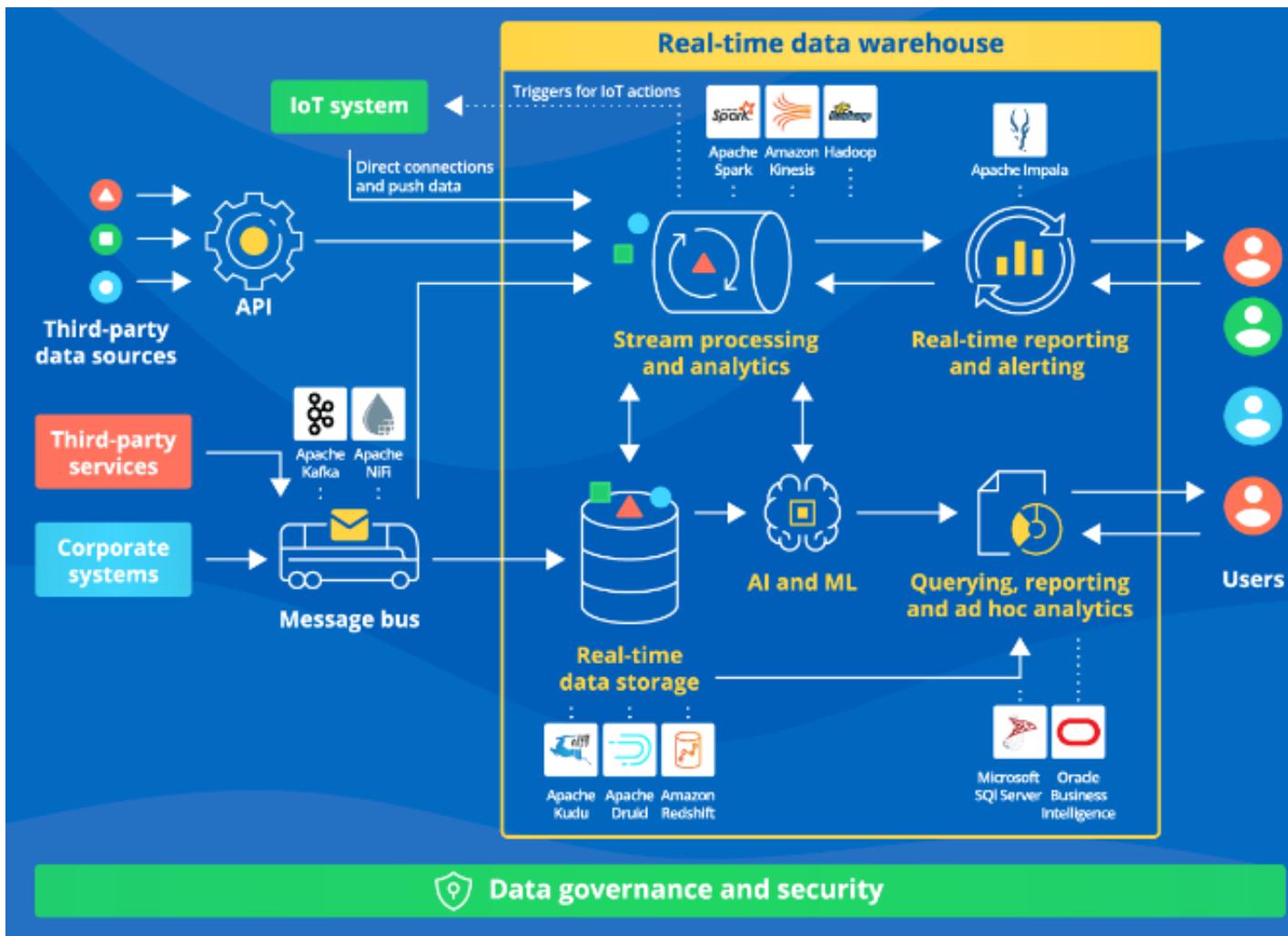
A manufacturing company uses sensors on its equipment to collect data in real time, which is then processed in a real-time data warehouse. This enables the company to monitor production processes closely, identify inefficiencies or issues as they arise, and take immediate corrective action to maintain optimal production levels and reduce downtime.

Process of Real Time Data Warehousing (RTDW)



- Real-time data warehousing is used as an information retrieval framework as soon as the data becomes available.
- In real-time data warehousing, the warehouse updates each time the system executes a transaction.
- It means that when a query triggers in the warehouse, it will return the company's status at that time

Real Time Data Warehousing Architecture



Real Time Data Warehousing Architecture

Real-time data warehouse solution.

- Today, majority of cloud-based platforms provide fully functional RTDWs out of the box – AWS, Azure & GCP
- These solutions offer built-in real-time analytics tools and their own set of real-time data integrations for specific data sources. When you choose one, make sure it will connect to the other data storage systems and tools you use.

Upgrade a traditional cloud data warehouse with a streaming pipeline.

- A streaming pipeline can turn a traditional cloud data warehouse into an RTDW by ingesting and transforming data in real time.
- For the warehouse itself, Snowflake and BigQuery are great choices.
- You can use a GUI-based platform like Estuary Flow or build your own pipeline with Apache Kafka.

Real Time Data Warehousing Architecture

Data Integration Tools:

- These are used for extracting, transforming, and loading (ETL) data.
- In real-time scenarios, traditional ETL is often replaced or augmented with real-time ETL processes (also known as streaming ETL) that handle data in motion.

Examples of Data Integration Tools:

- Apache NiFi: An open-source tool designed for automated and efficient movement of data between systems. It supports real-time data streaming and processing, making it suitable for real-time data integration tasks.
- StreamSets: A platform that enables continuous data ingestion, processing, and delivery, supporting a design for real-time streaming and batch data flows, thereby facilitating the integration of data into a data warehouse in real-time.

Real Time Data Warehousing Architecture

Data ingestion

An RTDW ingests real-time data with high throughput performance. Depending on the data source type and the physical distance between the data source and the analytics software, data can be ingested into the processing block by several means:

- Direct connections: for IoT systems.
- APIs: for third-party data sources (e.g., payment gateways, messaging services, authentication services).
- A message bus: for corporate systems (ERP, CRM, accounting software, etc.) and third-party services (e.g., customer data from an ecommerce platform, telematics data from a third-party device provider)

Real Time Data Warehousing Architecture

Change Data Capture (CDC):

- CDC is a technology used to capture changes made at the data source level (such as inserts, updates, and deletes) and to replicate those changes in the data warehouse in real-time or near-real-time, reducing the latency in data reporting and analysis.
- **Debezium:** An open-source distributed platform that turns your existing databases into event streams, so applications can see and respond almost instantly to each committed change. This is particularly useful for updating data warehouses in real-time without impacting the performance of the source databases.
- **Oracle GoldenGate:** A comprehensive software package for real-time data integration and replication in heterogeneous IT environments. It enables high availability solutions, real-time data integration, transactional change data capture, data replication, transformations, and verification between operational and analytical enterprise systems.

Real Time Data Warehousing Architecture

Real time storage & Data Streaming:

- Data streaming technologies allow for the continuous flow and processing of data.
- The real-time storage acts as a buffer that ensures reliable queuing logic, e.g., record ordering, scaling resources, delivering messages with minimal latency. This location also enables pre-analytics processing (ETL/ELT).
- Tools and platforms like Apache Kafka, Amazon Kinesis, and Google Pub/Sub enable real-time data processing and analytics by handling large streams of live data.
- **Apache Kafka:** A distributed streaming platform that can publish, subscribe to, store, and process streams of records in real time. Kafka is widely used for building real-time streaming data pipelines that reliably get data between systems or applications.
- **Amazon Kinesis:** A platform on AWS to collect, process, and analyze real-time, streaming data. It enables developers to build applications that can continuously ingest and process large streams of data records in real time.

Real Time Data Warehousing Architecture

Advanced Analytics and Machine Learning:

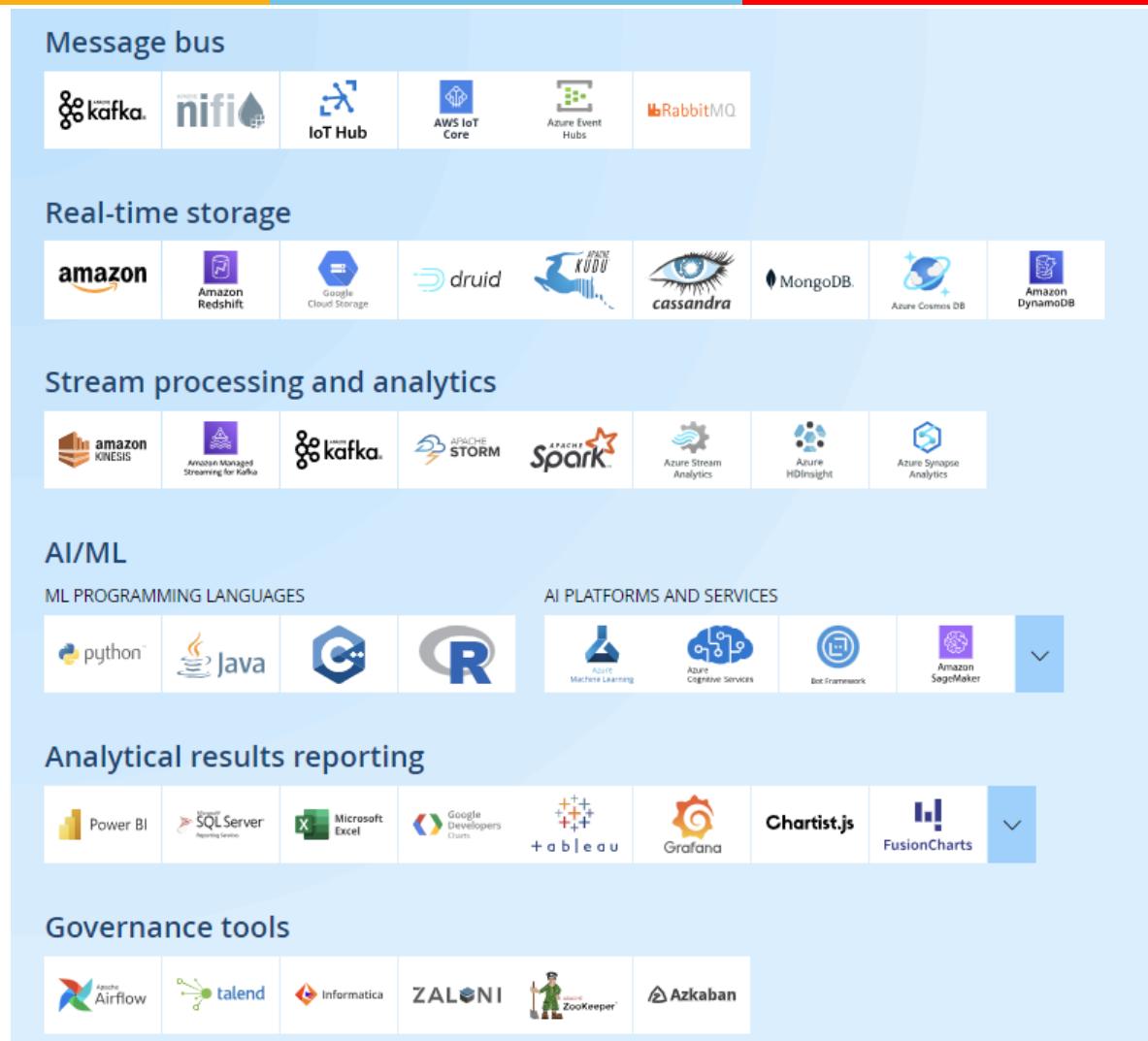
- Real-time data warehousing often leverages advanced analytics and machine learning models to instantly analyze incoming data streams for insights, predictions, and anomaly detection.
- **Apache Spark:** An open-source, distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark's in-memory processing capabilities make it well suited for machine learning algorithms requiring real-time analytics.
- **TensorFlow with TensorFlow Serving:** TensorFlow is an open-source framework for machine learning. TensorFlow Serving is designed to serve machine learning models with high performance and in a scalable way, enabling real-time processing of data and predictions in production environments.

Real Time Data Warehousing Architecture

Data access and reporting

- An RTDW makes the processed data immediately available as short-term insights and event-based alerts or automated action triggers.
- But in addition, such solutions enable comprehensive analytics of the accumulated historical data and ad hoc generation of custom reports.

Real Time Data Warehousing Architecture



Real Time Data Processing

There are different levels of real-time systems

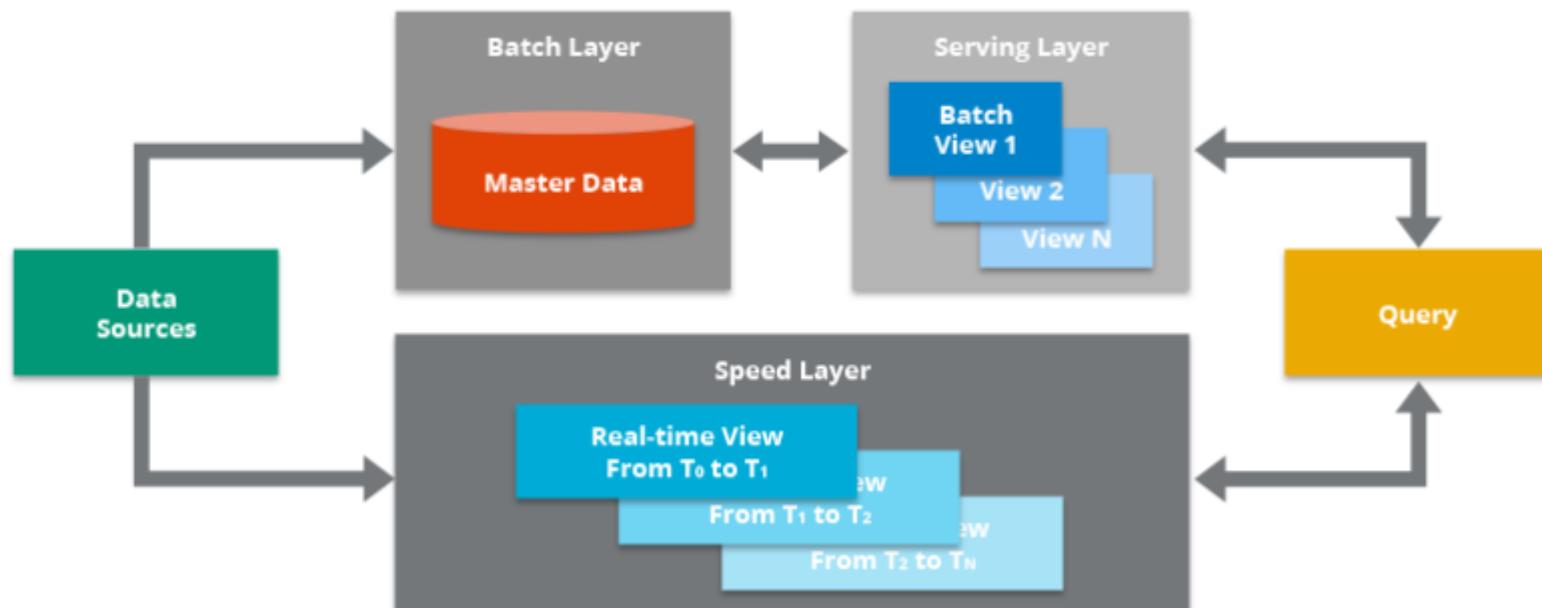
1. Microsecond systems (e.g., spaceships that require custom hardware)
2. Milli second systems (e.g., real-time auctions)
3. second or minute systems (real-time or Near-Real-Time (NRT) data processing pipeline)

Determine Data SLA

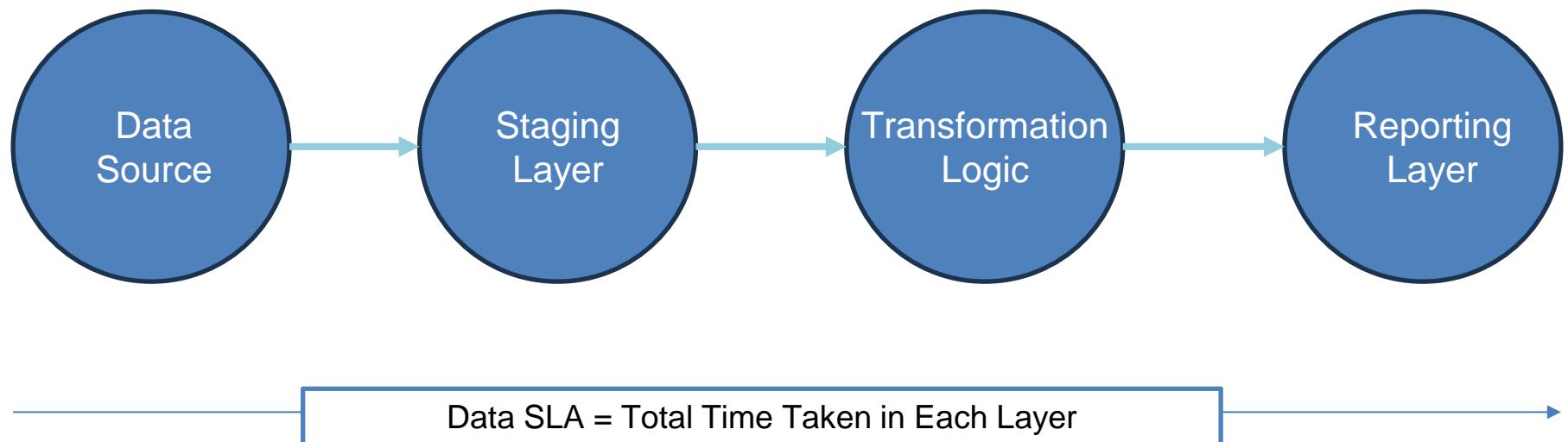
- The first step is deciding what data needs to be made available in real time.
- This is usually driven by business requirements.
- Most real-time data required for analysts or data scientists would be ok with about a 1h delay.
- When the consumer of your real-time data is another service or application then a lower time delay would be ideal since this data may be used for automatic consumption (e.g., anomaly detection).
- Usually, in most companies, the data that is processed in real-time is a very small subset of the incoming data.
- A trade-off between the business requirements and software development costs needs to be made, this will depend on the individual business requirement

Lambda Architecture (aka Speed Layer)

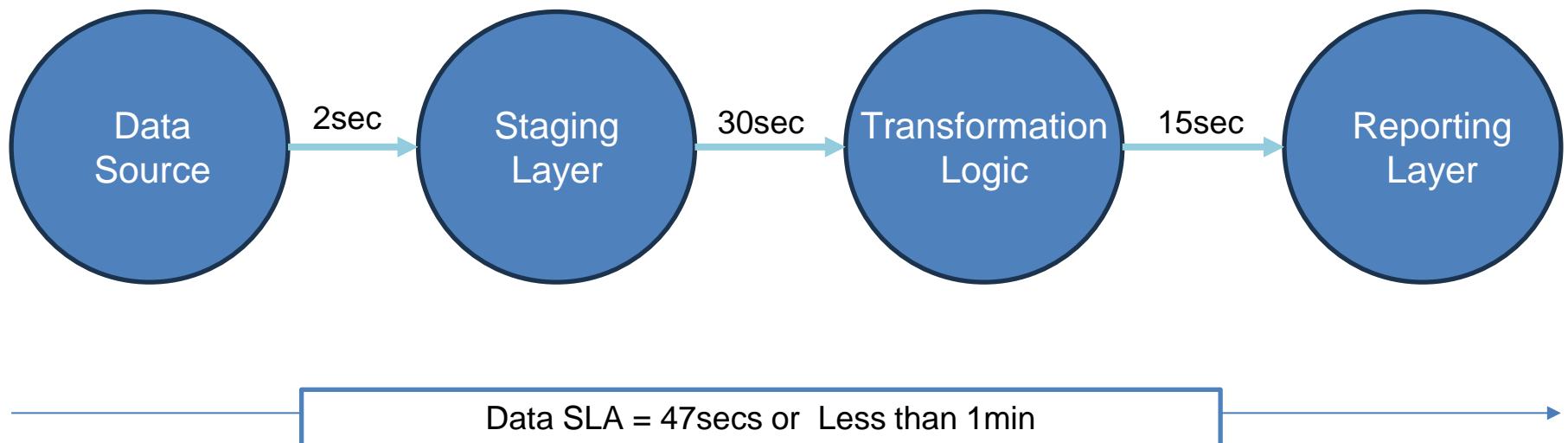
- Most companies populate their data warehouses using a combination of real-time and batch processing.
- The consumer of the data will need to know when and how to use the real-time data v the batch data that is available.
- This is called lambda architecture.



Determine Data SLA for Real-Time



Determine Data SLA for Real-Time - Example



Note: You can bring in Speed Layer to reduce the time

Real-Time ETL

- Real-time ETL (Extract, Transform, Load) is a process that involves continuously capturing and processing data as it becomes available, transforming it into a format suitable for analysis, and then loading it into a destination system such as a data warehouse or database in real-time or near-real-time.
- Unlike traditional ETL, which operates on a batch basis (processing data at set intervals, such as nightly or weekly or daily or hourly), real-time ETL aims to minimize the time between data creation and availability for analysis, enabling businesses to make decisions based on the most current data.

How Real-time ETL Works:

- **Extract:** Data is continuously captured from various sources as it is generated. This can include databases, sensors, software applications, and online transactions.
- **Transform:** The streaming data is cleaned, validated, aggregated, or otherwise processed in real time. This step ensures the data is in the correct format and quality for analysis.
- **Load:** The transformed data is then loaded into the target system (e.g., a real-time analytics platform, data warehouse, or operational datastore) almost immediately after it is received.

Scenario: Real-time Fraud Detection in Credit Card Transactions

A major financial institution processes millions of credit card transactions daily. To protect its customers and itself from fraudulent activities, it needs to detect and act on suspicious transactions in real-time.

Challenge:

Fraudulent transactions can happen at any time, and patterns may evolve rapidly. Traditional batch processing ETL, which might run every few hours or once a day, is not sufficient to catch fraud as it happens. The institution requires a system that can analyze transactions instantly and flag or block suspicious activity before significant damage is done

Scenario: Real-time Fraud Detection in Credit Card Transactions

- The financial institution implements a real-time ETL pipeline to analyze credit card transactions as they occur.
- Extract: Transaction data is captured in real-time from point-of-sale terminals, online purchases, and ATM withdrawals. This includes details like transaction amount, location, merchant type, and time of transaction.
- Transform: Each transaction is immediately passed through a series of transformations.
- Validation checks for any anomalies in the transaction data (e.g., invalid card numbers or expiration dates).
- Enrichment adds context to the transaction by including data from other sources, such as the customer's purchase history, current account balance, and recent locations based on past transactions.
- Scoring applies machine learning models that analyze the transaction in the context of the enriched data to assess the likelihood of fraud. These models consider factors like unusual spending patterns, transactions in uncommon locations, or purchases at high-risk merchants.
- Load: Transactions flagged as potentially fraudulent are instantly loaded into a fraud management system where they can be reviewed by fraud analysts. Simultaneously, the transaction data (both flagged and non-flagged) is loaded into the institution's data warehouse for longer-term analysis and model refinement.

Scenario: Real-time Fraud Detection in Credit Card Transactions

Outcome:

- Immediate Action: Transactions identified as high risk can be blocked or flagged for follow-up, often within milliseconds to seconds. Customers may receive an instant notification to confirm the transaction if it's flagged as suspicious.
- Customer Trust: By effectively minimizing fraud, the financial institution maintains and enhances customer trust. Customers feel secure using their credit cards, knowing that their bank has robust measures in place to protect them.
- Adaptive Techniques: Continuous analysis of transactions and outcomes feeds back into refining the machine learning models, making the system smarter and more adept at identifying fraud over time.
- Operational Efficiency: Real-time processing reduces the need for retrospective fraud investigation and the associated operational overhead. It also decreases the financial losses associated with fraud

Real-time Data Warehouse Modelling

- Real-time data warehouse modeling is the process of applying data modeling principles and techniques to create a data warehouse that can ingest, store, and analyze streaming data in real time or near real time.
- It involves different types of data modeling, such as conceptual, logical, and physical.
- Conceptual data modeling focuses on the high-level business requirements and objectives of the data warehouse, while logical data modeling translates the conceptual model into a more detailed and normalized representation of the entities and attributes.
- Physical data modeling implements the logical model into a specific database or data warehouse platform.
- However, when dealing with streaming data, there are some specific considerations and adaptations to take into account such as its high volume and velocity, variable format and structure, uncertain quality and consistency, and time-sensitivity

Real-time Data Warehouse Modelling

- Understand the fundamentals of data modeling and data warehousing, such as entities, attributes, relationships, keys, normalization, denormalization, dimensions, facts, measures, hierarchies, star schema, snowflake schema, and data marts.
- Additionally, you should also learn the tools and technologies that enable you to handle streaming data sources like sensors or messaging systems.
- This includes streaming data platforms like Apache Kafka or Google Cloud Pub/Sub and streaming data processing frameworks like Apache Spark.
- There are also streaming data warehouse solutions such as Snowflake or Amazon Redshift.

Challenges with Real-Time Data Warehousing

Traditional data warehouses are not designed to handle the concurrency and scalability needed for high-speed, real-time analytics.

Architectural Limitations

- Traditional data warehouses are monolithic structures optimized for batch processes.
- The complex, pre-aggregated tables and pre-defined schemas are not conducive for ad-hoc, real-time queries.
- While these designs were effective for periodic reports, they fail to meet the low-latency requirements of real-time analytics.

Latency Concerns

- In a real-time setup, latency is the enemy. The time it takes to ingest, process, query, and visualize the data can mean the difference between seizing an opportunity and missing it.
- Round-trip time for querying and data retrieval becomes a significant bottleneck in traditional systems.

Concurrency & Stability

- Another challenge with traditional data warehouses is concurrency—the ability to handle multiple queries simultaneously without degrading performance.
- As the volume of real-time analytics grows, scaling becomes imperative. Unfortunately, monolithic data warehouses are often ill-equipped to scale horizontally, exacerbating the latency issue.

ARCHITECTURAL CONSIDERATIONS FOR REAL-TIME ANALYTICS

Stream Processing

- Stream processing technologies like Kafka and Spark Streaming offer a bridge to fill the real time gap in data warehouses.
- Kafka's publish-subscribe model can continuously ingest streams of data into the warehouse.
- On the other hand, Spark Streaming can process this data in real-time before it even lands in the warehouse.
- These technologies essentially transform the data warehouse into an active participant in the data pipeline, rather than a passive recipient of batch uploads.

Data Partitioning

- Data partitioning, either horizontal or vertical, can significantly mitigate latency and concurrency issues
- Sharding strategies divide the dataset across multiple servers, enabling parallel query execution and reducing round-trip times.
- These partitioning methods offer an architectural shift that aligns more closely with real-time analytics.

Event-Based Processing

- Event-based processing allows for a more reactive system. Utilizing microservices architectures, event-based models enable real-time data ingestion and analytics.

ARCHITECTURAL CONSIDERATIONS FOR REAL-TIME ANALYTICS

Security & Compliance in Real-Time Analytics

- As data warehouses evolve to support real-time analytics, the security and compliance challenges compound exponentially.
- Real-time analytics, by its nature, demands a more porous boundary to enable quicker access to data. However, this velocity can sometimes lead to vulnerabilities, making the architecture susceptible to unauthorized access.
- Beyond the technical dimensions, compliance represents another convoluted maze.
- Regulations like GDPR, CCPA, and HIPAA have stringent data handling requirements.
- Meeting these demands within a real-time analytics framework often means implementing complex access controls, robust encryption algorithms, and comprehensive auditing mechanisms.
- Contrary to traditional batch processing systems, real-time analytics systems must meet these requirements at all times, as they are consistently interacting with sensitive data.

Benefits of Real-Time Data Warehousing

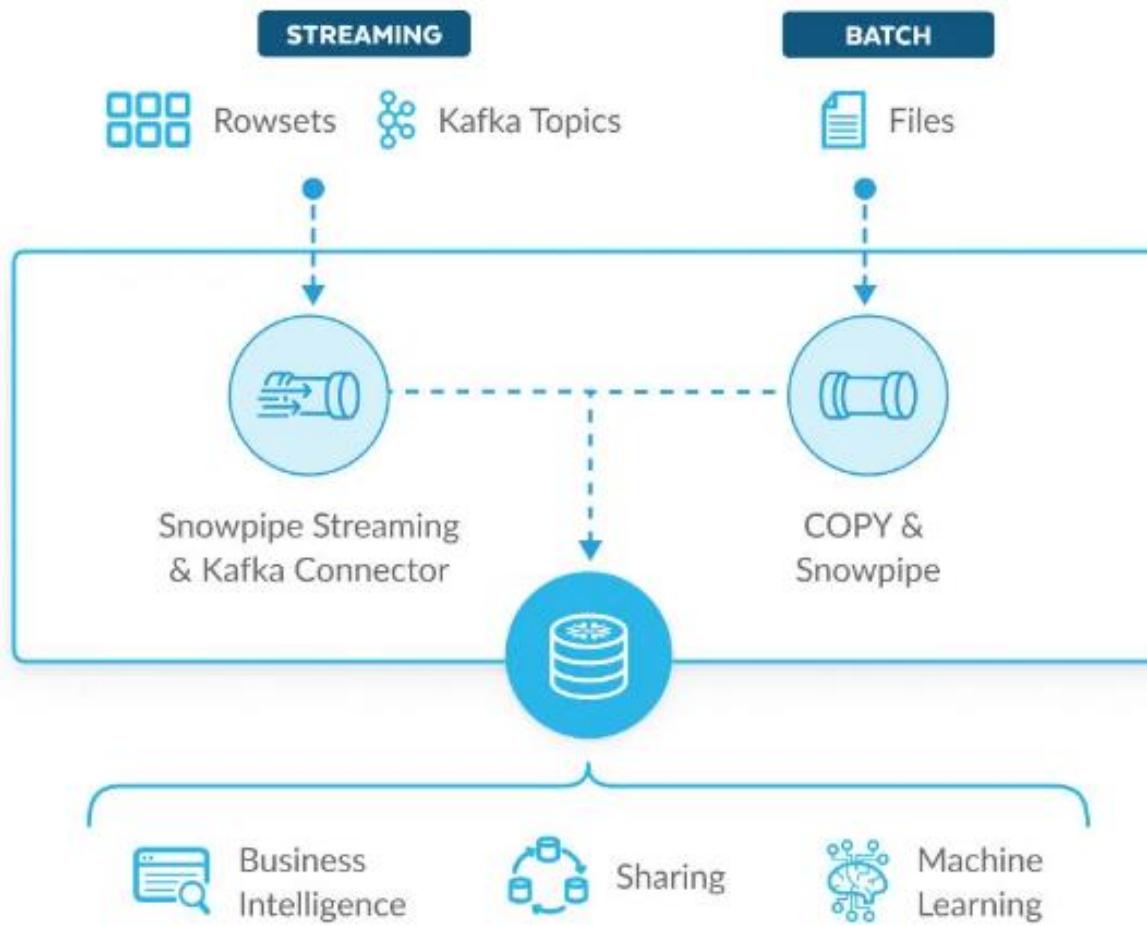
Real-time data warehousing has many advantages, including:

- It makes faster decisions based on more up-to-date, accurate, and transactionally consistent data.
- It reduces the load on the data source. Many organizations struggle to identify the perfect data loading window due to low-impact log-based change data capture (CDC).
- It accelerates recovery from data conversion or load issues.
- It removes the batch window that needs to idle the source database and possibly the data warehouse during loading. It prevents inconsistent data from being reflected in the query.

Real Time Analytics with Snowflake

- Snowflake helps you handle both streaming and batch data ingestion and transformation in a single system.
- Stream row-set data in near real time with single-digit latency using [**Snowpipe Streaming**](#), or auto-ingest files with [**Snowpipe**](#).
- Both options are serverless so you can scale more easily and manage costs more effectively.

Real Time Analytics with Snowflake



Snowpipe

- Snowpipe enables loading data from files as soon as they're available in a stage. This means you can load data from files in micro-batches, making it available to users within minutes, rather than manually executing COPY statements on a schedule to load larger batches.
- A pipe is a named, first-class Snowflake object that contains a COPY statement used by Snowpipe.
- The COPY statement identifies the source location of the data files (i.e., a stage) and a target table.
- All data types are supported, including semi-structured data types such as JSON and Avro

Snowpipe

- Automated data loads leverage event notifications for cloud storage to inform Snowpipe of the arrival of new data files to load.
- Snowpipe polls the event notifications from a queue.
- By using the metadata in the queue, Snowpipe loads the new data files into the target table in a continuous, serverless fashion based on the parameters defined in a specified pipe object.

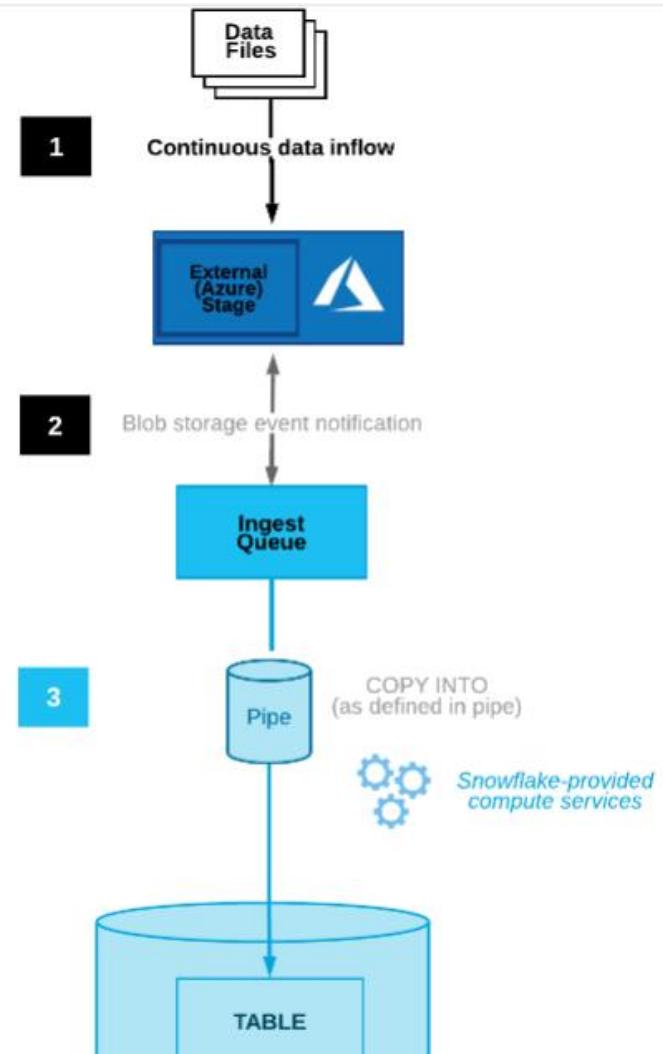
Snowpipe

- Automated data loads leverage event notifications for cloud storage to inform Snowpipe of the arrival of new data files to load.
- Snowpipe copies the files into a queue, from which they are loaded into the target table in a continuous, serverless fashion based on parameters defined in a specified pipe object

Snowpipe – Integrating with Azure Storage

[Microsoft Azure Event Grid](#) notifications for an Azure container trigger Snowpipe data loads automatically.

1. Data files are loaded in a stage.
2. A blob storage event message informs Snowpipe via Event Grid that files are ready to load. Snowpipe copies the files into a queue.
3. A Snowflake-provided virtual warehouse loads data from the queued files into the target table based on parameters defined in the specified pipe.

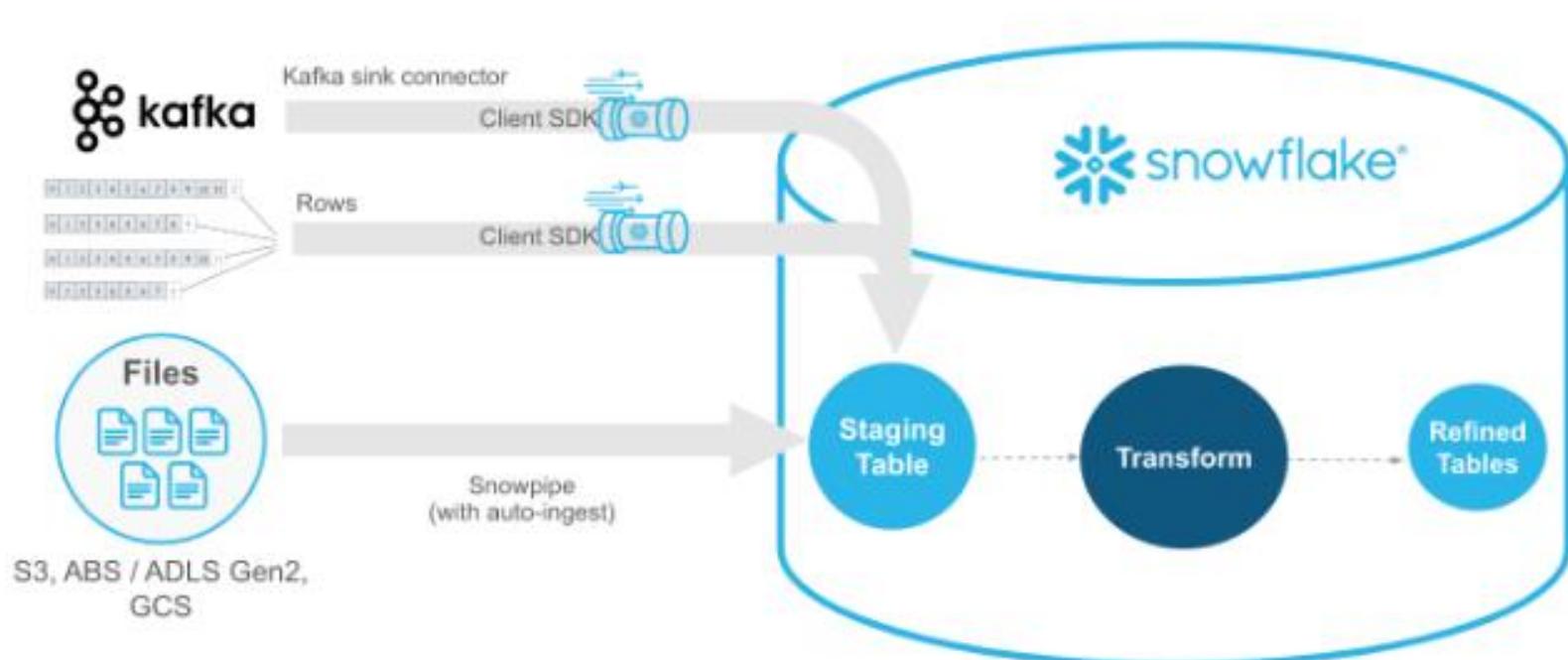


Note: [Automating Snowpipe for Microsoft Azure Blob Storage](#)
[Snowflake Documentation](#)

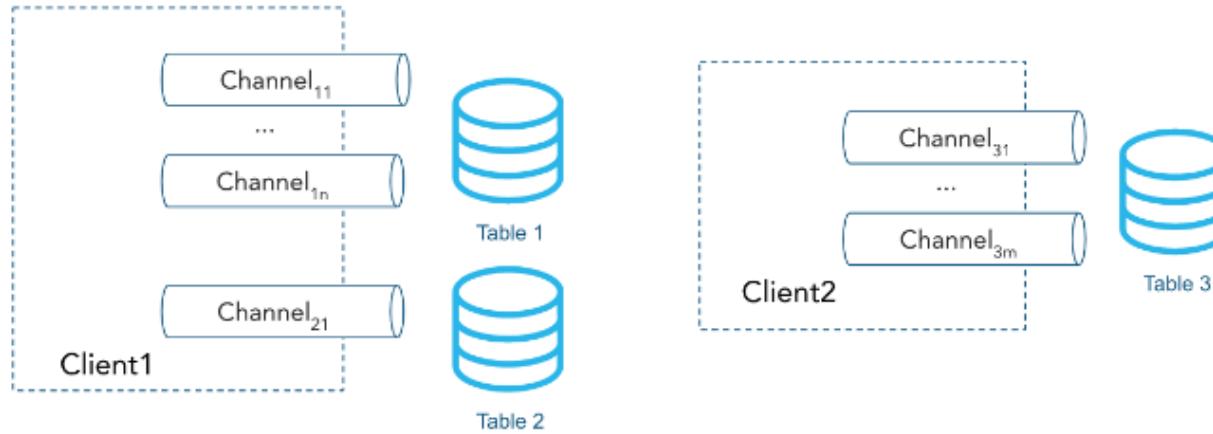
Snowflake Streaming

- Calling the Snowpipe Streaming API ("API") prompts the low-latency loading of streaming data rows using the Snowflake Ingest SDK and your own managed application code.
- The streaming ingest API writes rows of data to Snowflake tables, unlike bulk data loads or Snowpipe, which write data from staged files. This architecture results in lower load latencies with corresponding lower costs for loading similar volumes of data, which makes it a powerful tool for handling real-time data streams.
- The API is currently limited to inserting rows. To modify, delete, or combine data, write the "raw" records to one or more staging tables. Merge, join, or transform the data by using continuous data pipelines to insert modified data into destination reporting tables.

Snowflake Streaming



Channels



The API ingests rows through one or more *channels*. A channel represents a logical, named streaming connection to Snowflake for loading data into a table. A single channel maps to exactly one table in Snowflake; however, multiple channels can point to the same table. The Client SDK can open multiple channels to multiple tables; however the SDK cannot open channels across accounts. The ordering of rows and their corresponding offset tokens are preserved within a channel but not across channels that point to the same table.

Channels are meant to be long lived when a client is actively inserting data and should be reused as offset token information is retained. Data inside the channel is automatically flushed every 1 second by default and do not need to be closed.

Snowpipe Streaming API versus Snowpipe

- The API is intended to complement Snowpipe, not replace it.
- Use the Snowpipe Streaming API in streaming scenarios where data is streamed via rows (for example, Apache Kafka topics) instead of written to files.
- The API fits into an ingest workflow that includes an existing custom Java application that produces or receives records.
- The API removes the need to create files to load data into Snowflake tables and enables the automatic, continuous loading of data streams into Snowflake as the data becomes available.

Snowpipe Streaming API versus Snowpipe

Category	Snowpipe Streaming	Snowpipe
Form of data to load	Rows	Files. If your existing data pipeline generates files in blob storage, we recommend using Snowpipe instead of the API.
Third-party software requirements	Custom Java application code wrapper for the Snowflake Ingest SDK	None
Data ordering	Ordered insertions within each channel	Not supported. Snowpipe can load data from files in an order different from the file creation timestamps in cloud storage.
Load history	Load history recorded in SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY view (Account Usage)	Load history recorded in LOAD_HISTORY view (Account Usage) and COPY_HISTORY function (Information Schema)
Pipe object	Does not require a pipe object: the API writes records directly to target tables.	Requires a pipe object that queues and loads staged file data into target tables.

Snowflake Streams

- A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data.
- This process is referred to as change data capture (CDC). This helps to enable change data capture (CDC) in Snowflake using streams.
- An individual table stream tracks the changes made to rows in a *source table*.
- A table stream (also referred to as simply a “stream”) makes a “change table” available of what changed, at the row level, between two transactional points of time in a table. This allows querying and consuming a sequence of change records in a transactional fashion.

Snowflake Streams - Offset

- A stream logically takes an initial snapshot of every row in the source object by initializing a point in time (called an *offset*) as the current transactional version of the object.
- The change tracking system utilized by the stream then records information about the DML changes after this snapshot was taken.
- Change records provide the state of a row before and after the change.
- Change information mirrors the column structure of the tracked source object and includes additional metadata columns that describe each change event.
- Stream itself does **not** contain any table data. A stream only stores an offset for the source object and returns CDC records by leveraging the versioning history for the source object.
- When the first stream for a table is created, several hidden columns are added to the source table and begin storing change tracking metadata.
- These columns consume a small amount of storage.
- Note that for streams on views, change tracking must be enabled explicitly for the view and underlying tables to add the hidden columns to these tables.