Data warehousing is the process of collecting, storing, and managing large volumes of structured data from multiple sources in a central repository. It enables businesses to perform data analysis, reporting, and decision-making efficiently.

A data warehouse is a centralized repository that stores data from various sources, such as transactional systems, applications, and external sources. It is designed to support business intelligence activities, such as reporting, analysis, and data mining. Unlike transactional databases, which are designed for day-to-day operations, data warehouses are designed for complex queries and analysis.

**Key Features of Data Warehousing:**

* **Centralized Storage**: Stores data from various sources in one place.
* **ETL Process (Extract, Transform, Load)**: Data is extracted from multiple sources, transformed into a consistent format, and loaded into the data warehouse.
* **Optimized for Querying**: Designed for fast and efficient analytical queries.
* **Historical Data Storage**: Retains historical data for trend analysis.
* **Supports Business Intelligence (BI)**: Used for reporting, dashboards, and analytics.

**Popular Data Warehousing Solutions:**

* **Cloud-based**: Snowflake, Amazon Redshift, Google BigQuery, Microsoft Azure Synapse
* **On-premises**: Teradata, Oracle Exadata

Since you're learning **Snowflake**, you might be particularly interested in how it differentiates itself from traditional data warehouses by offering **scalability, cloud-native architecture, and cost efficiency**. Let me know if you want a deeper dive into Snowflake’s data warehousing capabilities!

**Characteristics of Data Warehousing**

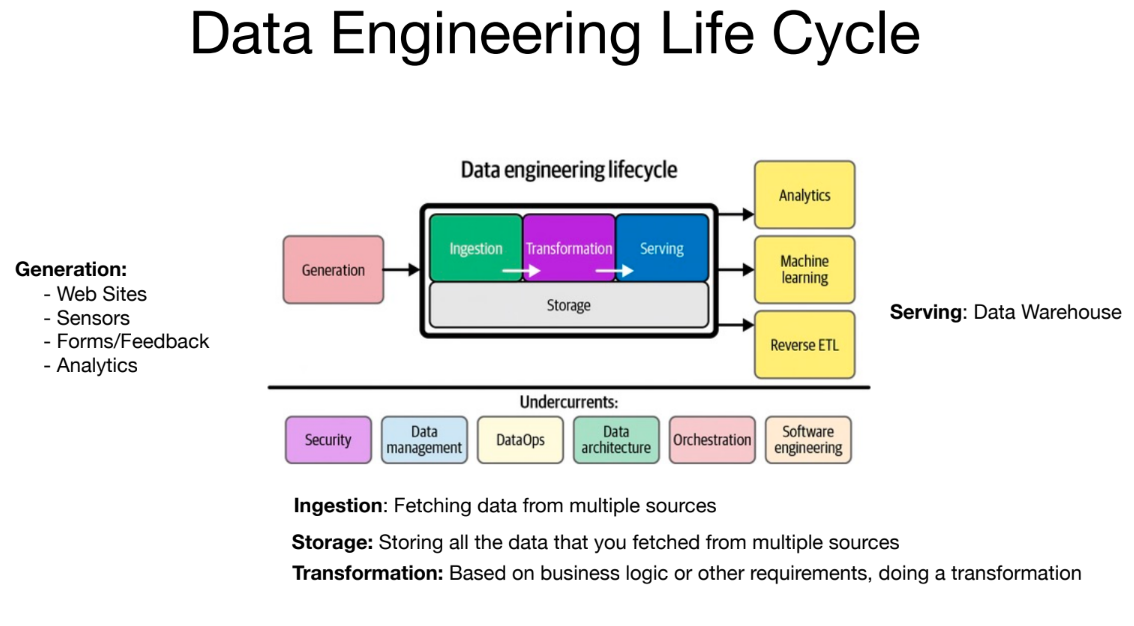
1. **Subject-Oriented** – Organized around specific business domains like sales, finance, or HR rather than transactional processes.
2. **Integrated** – Combines data from multiple sources (databases, flat files, APIs) into a unified format.
3. **Time-Variant** – Stores historical data over time, allowing for trend analysis and forecasting.
4. **Non-Volatile** – Data is read-only and does not frequently change, ensuring consistency.
5. **Optimized for Analysis** – Designed for complex queries, reporting, and Business Intelligence (BI) rather than transaction processing.
6. **Scalable and High Performance** – Supports large data volumes and concurrent queries efficiently.

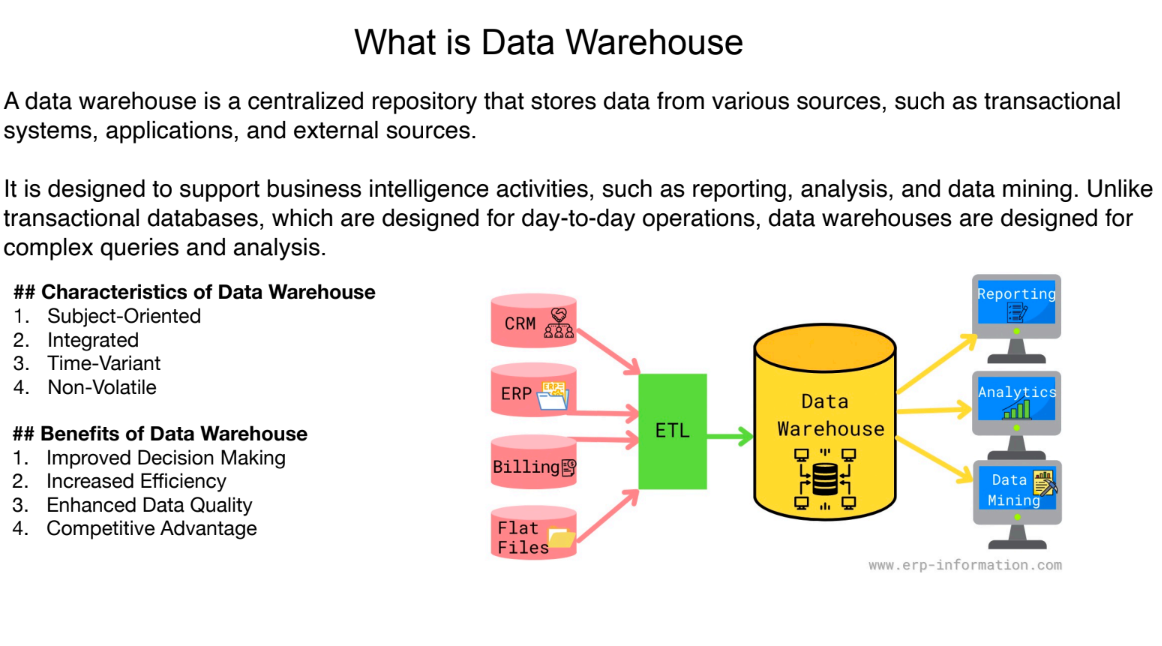
**Benefits of Data Warehousing**

✅ **Improved Decision-Making** – Provides reliable and structured data for analytics and reporting.  
✅ **Faster Query Performance** – Optimized for quick data retrieval compared to operational databases.  
✅ **Data Consistency & Quality** – Cleansed and standardized data improves accuracy.  
✅ **Historical Data Analysis** – Supports long-term trend analysis and predictive analytics.  
✅ **Business Intelligence & Reporting** – Enhances insights through dashboards and visualizations.  
✅ **Security & Compliance** – Centralized governance ensures better data control and regulatory compliance.

Since you're learning **Snowflake**, its cloud-native architecture provides additional benefits like **automatic scaling, cost efficiency, and seamless data sharing**. Would you like a comparison between Snowflake and traditional data warehouses?

**NOTE:** [**https://datawithdarshil.notion.site/What-is-Data-Warehouse-and-Why-Do-We-Need-It-247480bb68554653bf08f1fe1aecfaa5**](https://datawithdarshil.notion.site/What-is-Data-Warehouse-and-Why-Do-We-Need-It-247480bb68554653bf08f1fe1aecfaa5)





Kimball’s approach to data warehousing, known as the **Dimensional Modeling approach**, is a bottom-up methodology that focuses on creating a data warehouse optimized for ease of reporting and business intelligence.

**Key Principles of Kimball’s Approach**

1. **Bottom-Up Design** – The data warehouse is built by first creating smaller, business-specific **Data Marts**, which are later integrated into a central Data Warehouse.
2. **Dimensional Modeling** – Uses **Star Schema** and **Snowflake Schema** to organize data into fact and dimension tables for fast querying.
3. **ETL Process (Extract, Transform, Load)** – Data is cleaned, transformed, and loaded into dimensional models before being queried.
4. **Business-Centric** – The design focuses on making data accessible and intuitive for business users.
5. **Conformed Dimensions** – Ensures that common business dimensions (e.g., Date, Product, Customer) remain consistent across multiple data marts.

**Kimball’s Data Warehouse Architecture**

* **Fact Tables** – Store transactional data (e.g., sales, revenue, orders).
* **Dimension Tables** – Contain descriptive attributes for facts (e.g., customer details, product categories).
* **Data Marts** – Smaller, department-specific data repositories (e.g., Sales Data Mart, HR Data Mart).
* **Enterprise Data Warehouse (EDW)** – An integrated collection of data marts that provides a holistic view of the organization.

**Advantages of Kimball’s Approach**

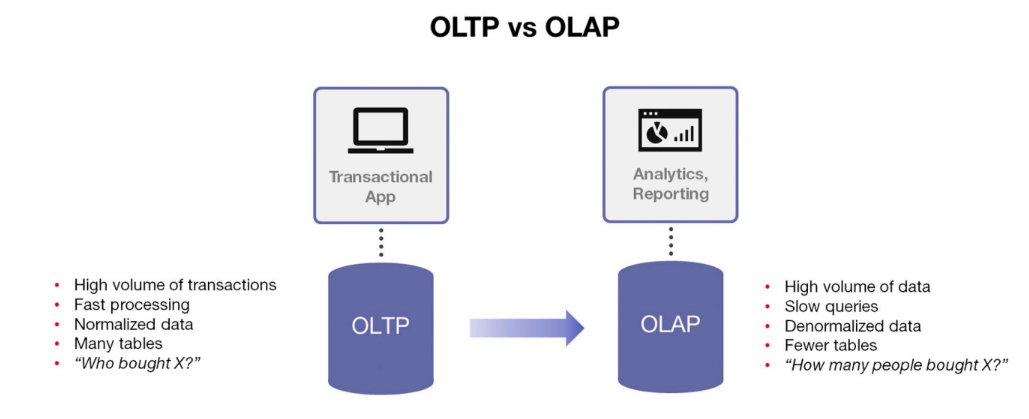
✅ Faster implementation compared to a centralized approach.  
✅ Business users can access data quickly for analysis.  
✅ Simplifies querying through well-structured schemas.  
✅ Ensures consistency through conformed dimensions.

**Kimball vs. Inmon (Alternative Approach)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Aspect** | **Kimball (Bottom-Up)** | **Inmon (Top-Down)** | | --- | --- | --- | | Design | Data marts first, then integrated into EDW | Centralized EDW first, then data marts | | Data Model | Dimensional (Star/Snowflake Schema) | Normalized (3NF) | | Query Performance | Faster for analysis | May require more joins | | Best for | BI and reporting-heavy environments | Data consistency and integration | |

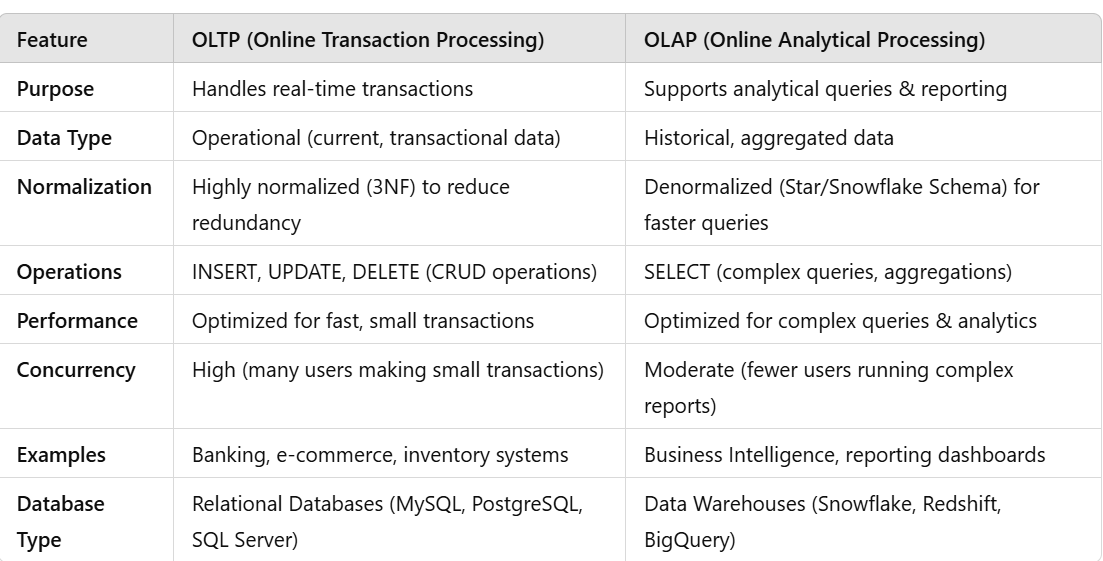
Since I’m learning **Snowflake**, it aligns well with Kimball’s approach because Snowflake supports **fast querying, scalability, and data sharing across data marts**.





**OLTP vs. OLAP: Key Differences**

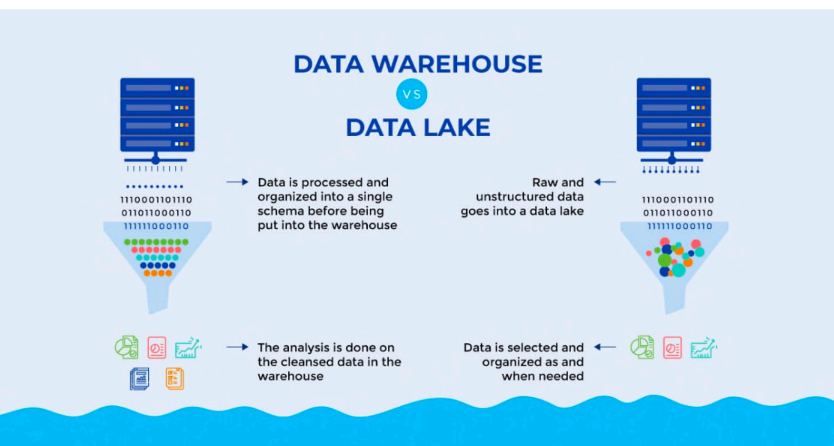
OLTP (**Online Transaction Processing**) and OLAP (**Online Analytical Processing**) are two distinct types of database systems designed for different purposes.



**Which One to Use?**

* Use **OLTP** for **day-to-day business operations** (e.g., processing orders, banking transactions).
* Use **OLAP** for **business intelligence and decision-making** (e.g., analyzing sales trends over time).

Since I’M learning **Snowflake**, it is primarily an **OLAP system** optimized for analytics, reporting, and large-scale data queries.

****

**Data Warehouse:** A Data Warehouse (DW or DWH) is a centralized repository that stores structured and processed data from multiple sources to support business intelligence (BI), reporting, and analytics. It is optimized for fast queries and complex analytical workloads rather than real-time transactions.

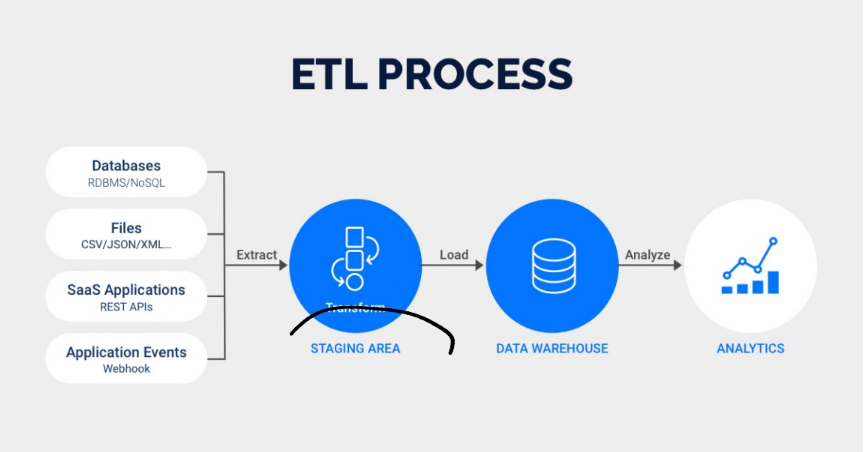
**Data Lake:** A Data Lake is a centralized storage system that allows you to store large volumes of raw data in its original format, including structured, semi-structured, and unstructured data. It is designed to handle big data, machine learning, and advanced analytics.

**Data Lakehouse:** A Data Lakehouse is a hybrid data architecture that combines the scalability and flexibility of a Data Lake with the structured data management and performance of a Data Warehouse. It allows organizations to store both raw and processed data in one system while enabling fast analytics, machine learning (ML), and business intelligence (BI).

**Which One Should You Use?**

* Use a **Data Warehouse** for **structured business analytics** (e.g., finance, reporting).
* Use a **Data Lake** for **big data, machine learning, and unstructured data storage**.
* Use a **Data Lakehouse** if you need **both analytics and big data processing in one system**.

Since you’re learning **Snowflake**, it supports both **Data Warehousing** and **Lakehouse architectures** through **Snowflake Unistore** and **external table support for Data Lakes**.



**ETL Process (Extract, Transform, Load)**

The **ETL process** is a fundamental data integration technique used to extract data from various sources, transform it into a usable format, and load it into a **Data Warehouse** or other storage system for analysis.

**Three Main Steps of ETL:**

1. **Extract** – Retrieve raw data from multiple sources.
2. **Transform** – Clean, structure, and process the data.
3. **Load** – Store the transformed data in a target system (Data Warehouse, Data Lake, or Data Lakehouse).

**1️⃣ Extract (E) – Getting the Data**

✔ Pulls data from sources like databases, APIs, flat files (CSV, JSON, XML), IoT devices, and logs.  
✔ Can be **batch-based** (scheduled) or **real-time** (streaming data).

🔹 **Example Data Sources:**

* SQL databases (MySQL, PostgreSQL, SQL Server)
* Cloud storage (AWS S3, Google Drive)
* APIs (REST, SOAP)
* CRM systems (Salesforce, SAP)

**2️⃣ Transform (T) – Processing & Cleaning the Data**

✔ Applies business rules, formatting, and validation.  
✔ Converts raw data into a structured format for analysis.  
✔ Common transformation tasks:

* **Data Cleaning** (handling missing values, duplicates)
* **Data Aggregation** (sum, average, count)
* **Data Standardization** (date formats, naming conventions)
* **Joining & Merging** (combining multiple datasets)

🔹 **Example:** Converting dates from MM/DD/YYYY to YYYY-MM-DD, removing duplicates, and categorizing product sales by region.

**3️⃣ Load (L) – Storing the Data**

✔ Loads transformed data into a **Data Warehouse, Data Lake, or Data Lakehouse**.  
✔ Can be **full loads** (entire dataset) or **incremental loads** (only new/updated data).

🔹 **Example Target Systems:**

* **Data Warehouse** (Snowflake, Redshift, BigQuery)
* **Data Lake** (AWS S3, Azure Data Lake)
* **Hybrid Lakehouse** (Databricks, Snowflake)

**ETL vs. ELT (Extract, Load, Transform)**

| **Feature** | **ETL** | **ELT** |
| --- | --- | --- |
| **Process** | Extract → Transform → Load | Extract → Load → Transform |
| **Transformation** | Before loading (structured) | After loading (flexible) |
| **Best For** | Traditional Data Warehouses | Big Data, Data Lakes |
| **Performance** | Slower (pre-processing needed) | Faster (parallel processing in cloud) |
| **Tools** | Informatica, Talend, SSIS | Snowflake, Databricks, BigQuery |

**Popular ETL Tools**

* **Cloud-Based ETL:** AWS Glue, Google Dataflow, Azure Data Factory
* **Open Source:** Apache NiFi, Talend, Airbyte
* **Enterprise:** Informatica PowerCenter, IBM DataStage
* **Modern ELT:** dbt (Data Build Tool), Fivetran, Matillion

**ETL in Snowflake**

Since I’m learning **Snowflake**, it supports **ELT** instead of traditional ETL. Snowflake’s compute engine can handle transformations **after loading** the data, improving performance and scalability.



**What is Dimensional Modeling?**

**Dimensional Modeling (DM)** is a database design technique used in **Data Warehousing** to optimize data for **fast querying and reporting**. It structures data in a way that is intuitive and easy to understand for business users, making it ideal for **Business Intelligence (BI) and analytics**.

Dimensional Modelling: <https://datawithdarshil.notion.site/Dimensional-Modelling-4bfe924d371c4ea2abcb02c13d8f2e73>

Fact tables and their types: <https://datawithdarshil.notion.site/Fact-Tables-and-Their-Types-140336c6eca3471e9d43d2a0354dca22>

Dimension tables and their types: <https://datawithdarshil.notion.site/Dimension-Tables-and-Types-922d121415314d6dbaa70c76fe3f7611>

**Key Concepts in Dimensional Modeling**

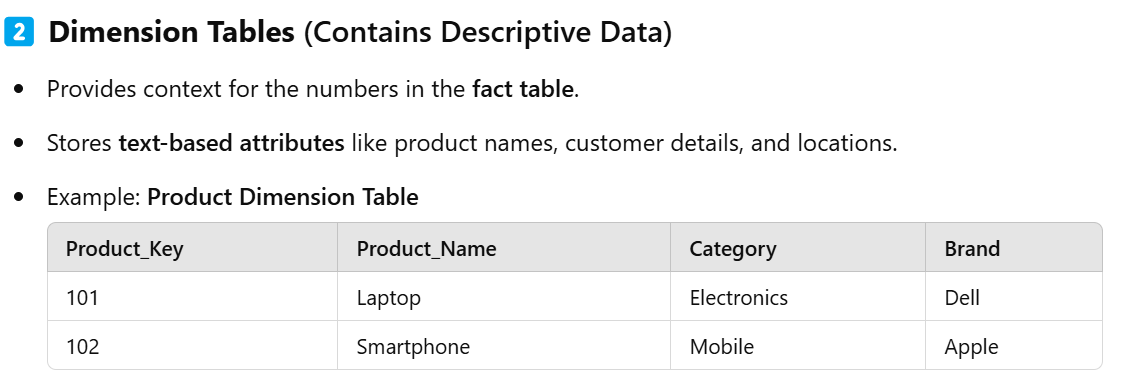
Dimensional modeling follows a **star schema or snowflake schema**, consisting of:

1. **Fact Table** – Stores measurable business events (e.g., sales, revenue, transactions).
2. **Dimension Tables** – Provide descriptive details (e.g., customer info, product details, time data).

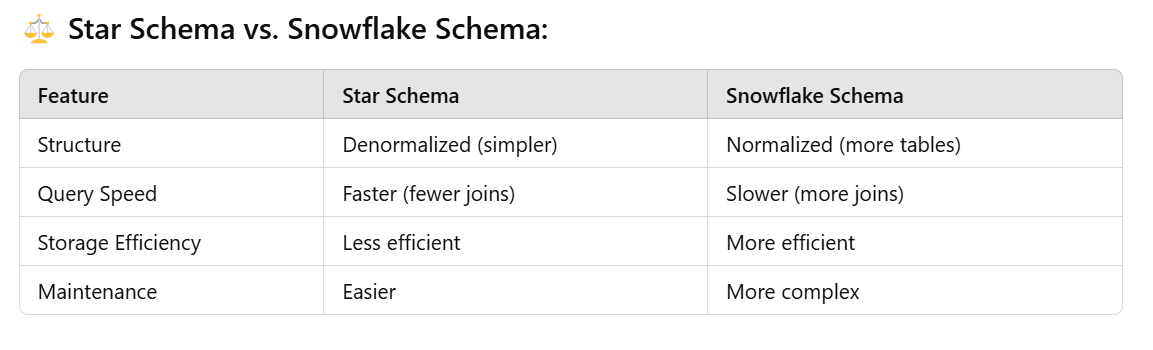
**Fact Table (Contains Numeric Data)**

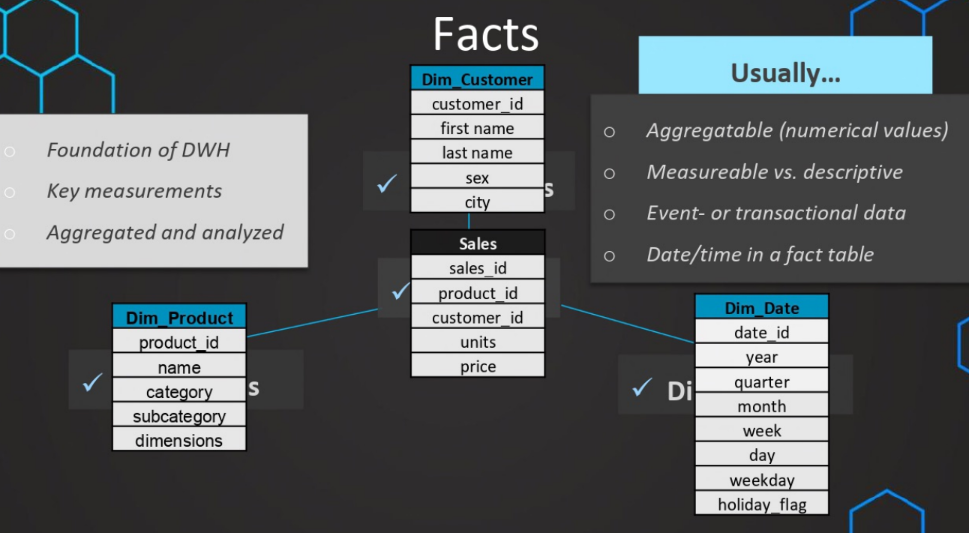
* Holds business **metrics** or **measures** (e.g., sales amount, quantity sold, profit).
* Contains **foreign keys** to connect to dimension tables.
* Example: **Sales Fact Table**

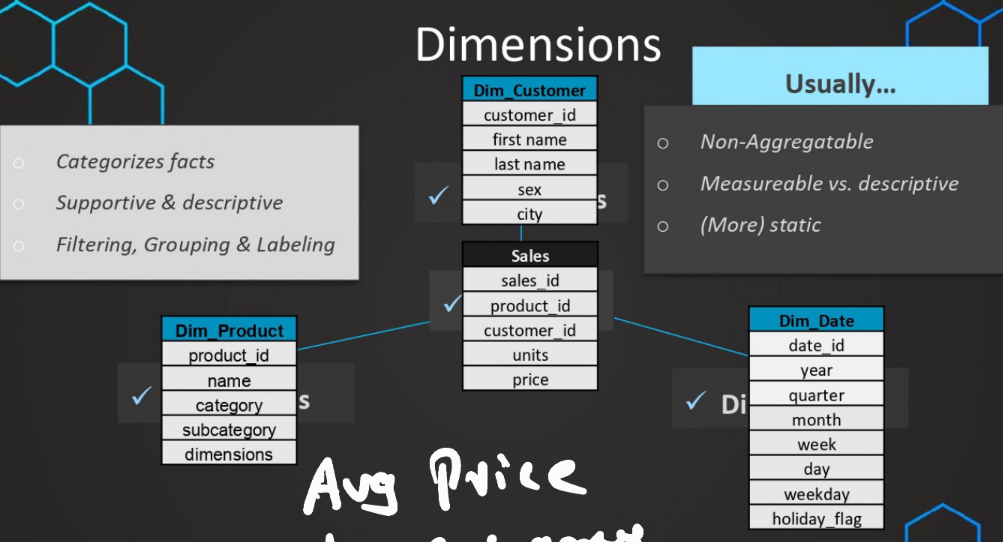
|  |
| --- |
|  |



Star schema vs snowflake schema









* A **star schema** is a **simple and fast** way to organize data in a **data warehouse** for **efficient querying**—especially for **reporting and business intelligence (BI)** tools.

Natural Key vs Surrogate Key

Great topic! Understanding **natural vs surrogate keys** is super important when designing **star schemas** (or any data model, really). Let’s break it down:

**🔑 Natural Key**

A **natural key** is a **real-world identifier** that already exists in your data.

**✔️ Example:**

* email\_address for a customer.
* product\_code or SKU for a product.
* SSN for a person (not recommended due to privacy issues!).

**✅ Pros:**

* Already exists — no need to create it.
* Makes sense to business users.

**❌ Cons:**

* Can change (e.g., customer changes their email).
* Might be **long or complex** (bad for joins).
* Potential **privacy issues**.
* Not always unique across systems.

**🆔 Surrogate Key**

A **surrogate key** is a **system-generated, unique ID**, often just an **integer**.

**✔️ Example:**

* customer\_id = 101, 102, 103...
* product\_id = 5001, 5002...
* Generated via sequences or auto-increment.

**✅ Pros:**

* **Simple, fast joins** (especially in large tables).
* **Never changes** — stable over time.
* Keeps **natural keys private** (useful in data warehouses).
* Helps with **slowly changing dimensions (SCD)** for tracking history.

**❌ Cons:**

* Adds extra step (need to generate & manage keys).
* Less meaningful to users (need to join to see actual data).

**🧱 In Star Schema – What Should You Use?**

**Fact Tables:**

* Always use **surrogate keys** to join to dimension tables — it’s more efficient.

**Dimension Tables:**

* Use **surrogate keys as primary key**.
* Store **natural keys** too (as a column) for reference and uniqueness checking.

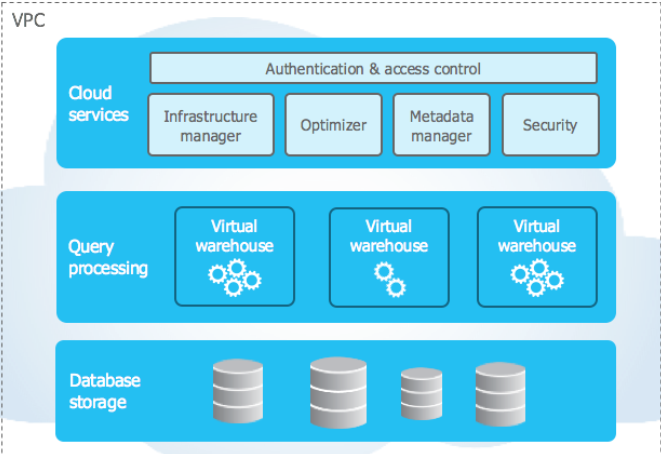
**📊 Example: Customer Dimension Table**

| **customer\_id (Surrogate)** | **email\_address (Natural)** | **name** | **city** |
| --- | --- | --- | --- |
| 101 | [alice@example.com](mailto:alice@example.com) | Alice Lee | New York |
| 102 | [bob@example.com](mailto:bob@example.com) | Bob Smith | Chicago |

| **Feature** | **Natural Key** | **Surrogate Key** |
| --- | --- | --- |
| Origin | Real-world data | System-generated |
| Stability | May change | Never changes |
| Meaningful | Yes | No |
| Performance | Slower joins | Faster joins |
| Recommended in DW | No | **Yes** |

Want to see how to generate surrogate keys in **Snowflake** or SQL? Or when to use them with **slowly changing dimensions (SCD)**? Let me know!

Snowflake architecture:



Snowflake’s unique architecture consists of three key layers:

* [Database Storage](https://docs.snowflake.com/en/user-guide/intro-key-concepts#database-storage)
* [Query Processing](https://docs.snowflake.com/en/user-guide/intro-key-concepts#query-processing)
* [Cloud Services](https://docs.snowflake.com/en/user-guide/intro-key-concepts#cloud-services)

**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.

**Query Processing**

Query execution is performed in the processing layer. Snowflake processes queries using “virtual warehouses”. Each virtual warehouse is an MPP compute cluster composed of multiple compute nodes allocated by Snowflake from a cloud provider.

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.

For more information, see [Virtual warehouses](https://docs.snowflake.com/en/user-guide/warehouses).

**Cloud Services**

The cloud services layer is a collection of services that coordinate activities across Snowflake. These services tie together all of the different components of Snowflake in order to process user requests, from login to query dispatch. The cloud services layer also runs on compute instances provisioned by Snowflake from the cloud provider.

Services managed in this layer include:

* Authentication
* Infrastructure management
* Metadata management
* Query parsing and optimization
* Access control

Absolutely! Snowflake has a **unique and powerful architecture** that sets it apart from traditional databases and even other cloud data platforms.

Let’s break down **Snowflake’s architecture** in a clear, visual way. Think of it as **3 layers** that work together but scale independently.

**🧊 Snowflake Architecture Overview**

**🔷 1. Storage Layer**

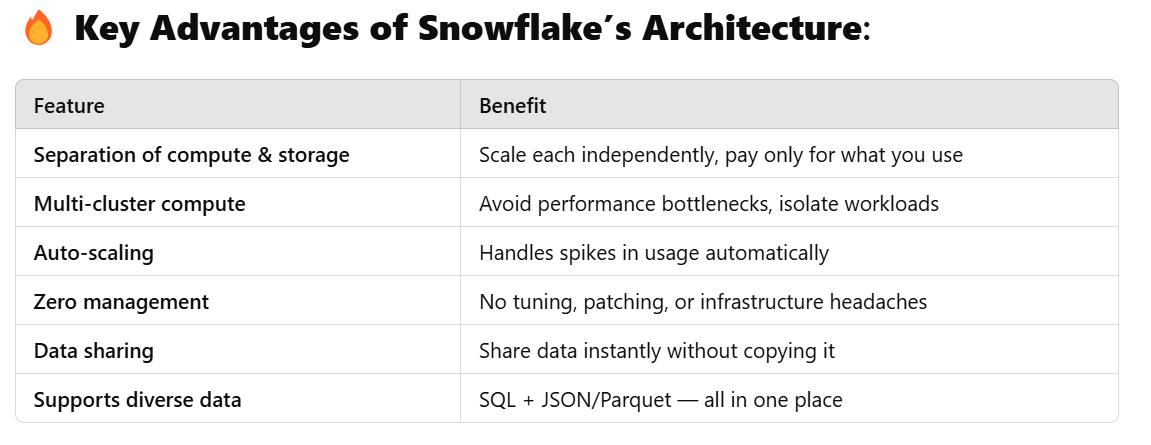
* Stores **all your data** — structured, semi-structured (JSON, Parquet), etc.
* Automatically **compressed**, **encrypted**, and stored in **cloud object storage** (AWS S3, Azure Blob, GCP Storage).
* **Fully managed** → You don’t worry about disks, partitions, etc.
* Supports **Time Travel** and **Fail-safe** (data recovery features).

**⚙️ 2. Compute Layer (Virtual Warehouses)**

* Runs all your **queries, data loads, transformations**.
* **Virtual Warehouses** = **independent compute clusters**.
* Each warehouse can scale up/down (bigger/smaller) or in/out (more/less parallelism).
* Multiple warehouses can run **concurrently** without affecting each other.  
  → Great for **isolation** (e.g., ETL on one, BI dashboards on another).

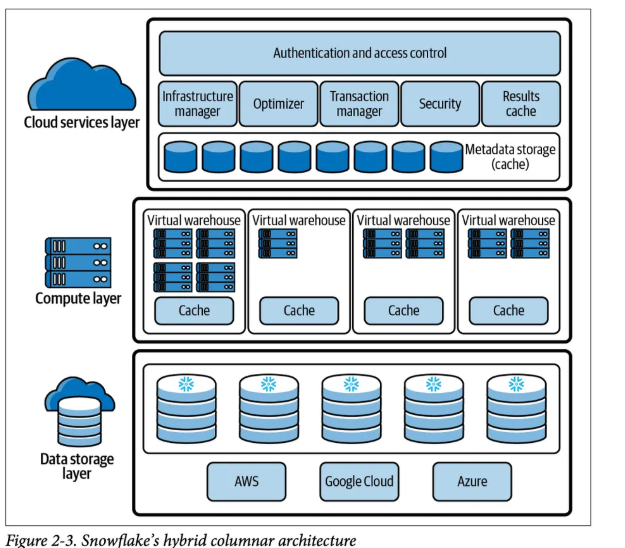
**🧠 3. Cloud Services Layer**

* Handles **metadata, authentication, query optimization**, security, and more.
* Examples:
  + User access control (RBAC)
  + Query parsing and optimization
  + Transaction management
  + Metadata: stats, query history, table info
* This layer is **always on**, but Snowflake manages it—you don’t pay separately for it.



**Snowflake Architecture: A Beginner's Guide**

If you're new to Snowflake, understanding its architecture can be confusing. But don't worry, this guide will break down Snowflake's architecture into easy-to-understand terms.



**What is Snowflake?**

Snowflake is a cloud-based data warehousing platform that allows users to store, process, and analyze large amounts of structured and semi-structured data. It was designed from the ground up to be a cloud-native solution, meaning it was built with the cloud in mind.

**How Does Snowflake Architecture Work?**

At a high level, Snowflake's architecture consists of three layers: the storage layer, the compute layer, and the services layer.

**Storage Layer**

The storage layer is where all data is stored in Snowflake. Snowflake uses a unique architecture called "multi-cluster shared data architecture," which allows all users to access the same data simultaneously without any interference. This is made possible by separating compute and storage, which means that compute resources can be scaled up or down independently of storage resources.

Data is stored in Snowflake as "micro-partitions." Each micro-partition is a small, self-contained unit of data that is compressed and encrypted. This allows Snowflake to quickly and efficiently read and write data, as well as ensure the security of your data.

**Compute Layer**

The compute layer is responsible for executing queries and processing data in Snowflake. When a query is executed, Snowflake automatically determines the optimal amount of compute resources needed to complete the query based on the size and complexity of the data being queried.

Compute resources in Snowflake are provided by virtual warehouses. A virtual warehouse is a cluster of compute resources that can be scaled up or down as needed. This allows Snowflake to provide near-instantaneous elasticity, meaning you can quickly and easily scale up or down your compute resources based on demand.

**Services Layer**

The services layer is responsible for managing user authentication, security, and metadata. It includes services such as the Snowflake web interface, which allows users to interact with Snowflake, as well as the Snowflake REST API, which allows developers to programmatically interact with Snowflake.

**Example**

Let's say you have a large dataset that you want to analyze. You upload the dataset to Snowflake, which stores it in the storage layer as micro-partitions. You then execute a query using the Snowflake web interface, which sends the query to the compute layer. The compute layer determines the optimal amount of compute resources needed to execute the query and spins up a virtual warehouse to handle the query. Once the query is complete, the results are sent back to you via the web interface.

**Advantages of Snowflake Architecture**

1. Scalability: Snowflake's architecture allows for the separation of compute and storage, which means that compute resources can be scaled up or down independently of storage resources. This provides near-instantaneous elasticity, allowing you to quickly and easily scale up or down your compute resources based on demand.
2. Efficiency: Snowflake's unique architecture allows for data to be stored as "micro-partitions," which are small, self-contained units of data that are compressed and encrypted. This allows Snowflake to quickly and efficiently read and write data, as well as ensure the security of your data.
3. Security: Snowflake's architecture ensures the security of your data by storing it in encrypted micro-partitions. Additionally, Snowflake's services layer is responsible for managing user authentication, security, and metadata.
4. Flexibility: Snowflake's architecture is designed to be cloud-native, meaning it was built with the cloud in mind. This allows for flexibility in deployment and integration with other cloud-based services.

**Disadvantages of Snowflake Architecture**

1. Cost: Snowflake is a cloud-based service, which means that there are ongoing costs associated with using it. Additionally, the cost of virtual warehouses can add up if not managed properly.
2. Complexity: While Snowflake's architecture is designed to be scalable, efficient, and secure, it can also be complex. This complexity may require additional training or expertise to fully utilize Snowflake's capabilities.

**Conclusion**

Snowflake's architecture may seem complex at first, but it's designed to be scalable, efficient, and secure. By separating compute and storage, Snowflake is able to provide near-instantaneous elasticity, which allows you to quickly and easily scale up or down your compute resources based on demand. With this guide, you should now have a better understanding of how Snowflake's architecture works, as well as its advantages and disadvantages.

**Stages & Copy**

SnowflakeDB is a cloud-based data warehousing system that allows users to store, manage, and analyze large amounts of data. One of the key features of SnowflakeDB is the ability to load data from external sources into tables using stages and the COPY command. In this tutorial, we'll cover how to create stages and load data into tables using the COPY command in SnowflakeDB.

**What are Stages in SnowflakeDB?**

Stages are named locations in an internal or external location that provide a way to access files from SnowflakeDB. Stages can be internal or external. Internal stages are created in SnowflakeDB and managed by the system. External stages are locations outside of SnowflakeDB that are accessed using an external location.

Stages provide a way to access and load data from external sources into SnowflakeDB tables. This is useful when you need to load data from sources such as S3, Azure Blob Storage, or Google Cloud Storage.

**Creating a Stage**

To create a stage, use the CREATE STAGE command. Here's an example of how to create an internal stage:

CREATE STAGE my\_internal\_stage;

This creates an internal stage named my\_internal\_stage.

Here's an example of how to create an external stage:

CREATE STAGE my\_external\_stage

URL='s3://my-bucket/my-prefix'

CREDENTIALS=(AWS\_KEY\_ID='<aws-key-id>' AWS\_SECRET\_KEY='<aws-secret-key>');

This creates an external stage named my\_external\_stage that points to an S3 bucket. The CREDENTIALS parameter specifies the AWS access key and secret access key needed to access the bucket.

**Loading Data with COPY**

Once you've created a stage, you can use the COPY command to load data into a table from the stage. Here's an example:

COPY INTO my\_table

FROM '@my\_internal\_stage/file.csv'

FILE\_FORMAT = (TYPE = 'CSV');

This loads data from a CSV file located in the my\_internal\_stage stage into a table called my\_table. The FILE\_FORMAT parameter specifies the format of the file being loaded.

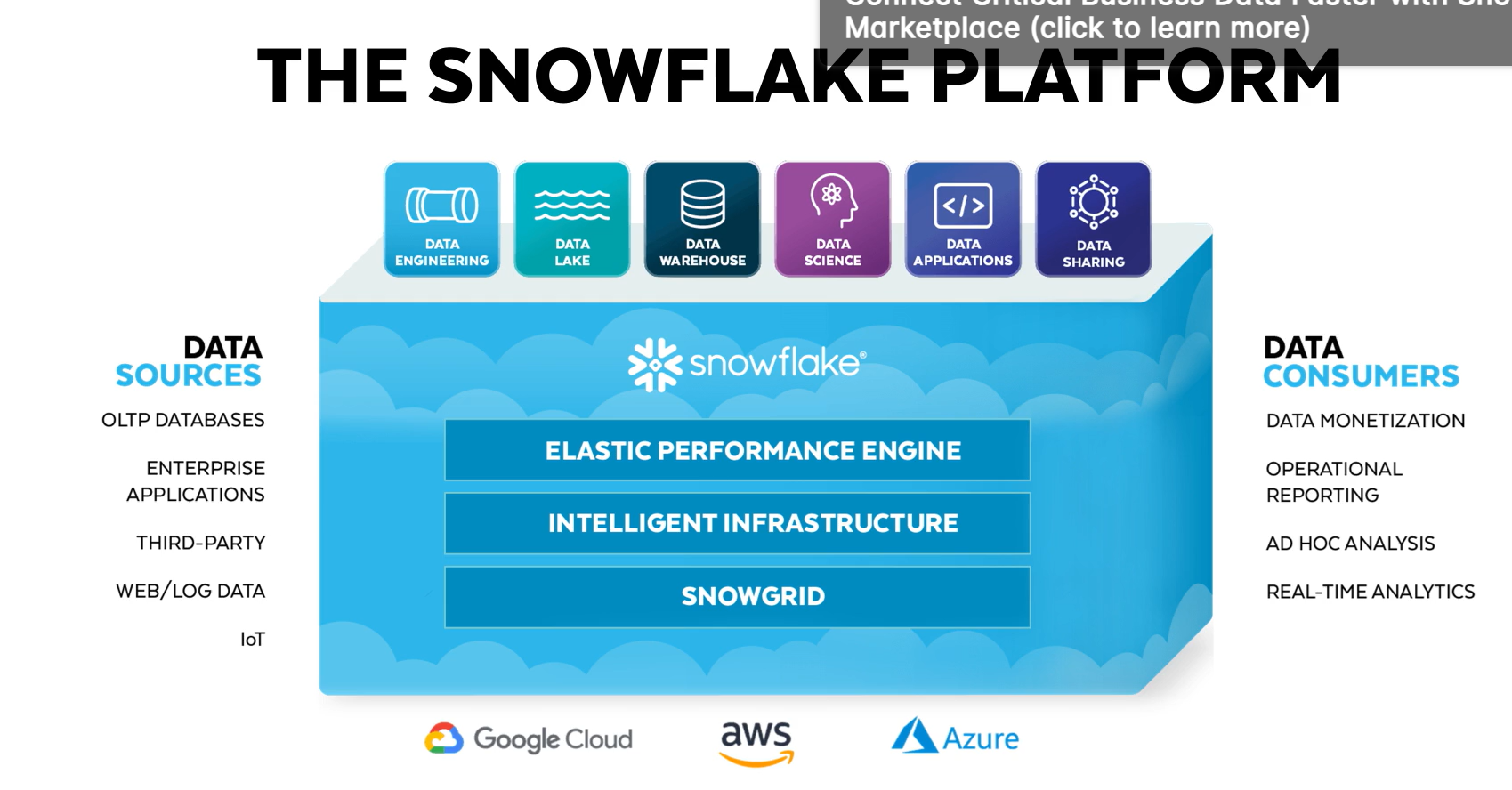
Here's an example of loading data from an external stage:

COPY INTO my\_table

FROM '@my\_external\_stage/file.csv'

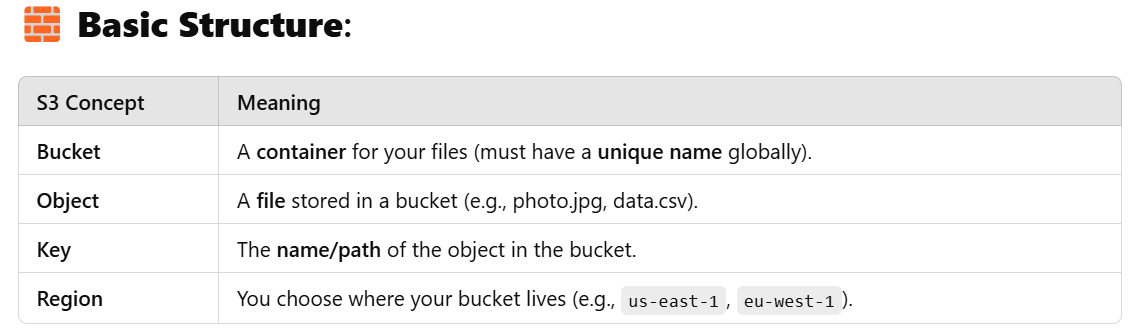
FILE\_FORMAT = (TYPE = 'CSV');

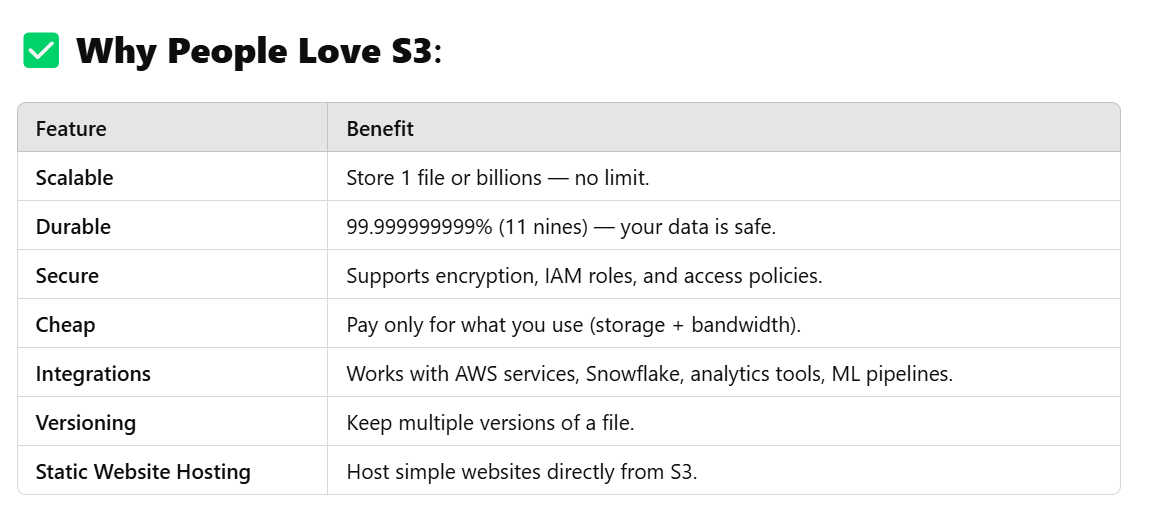
This loads data from a CSV file located in the my\_external\_stage stage into my\_table.



**E-COMMERCE DIMENSION MODELLING.**

* + First created aws s3 bucket and uploaded all the files in the bucket.
  + To get access key credentials. Created one IAM user with admin permissions. The credentials are downloaded in the csv file and saved in local download folder.
  + Snowflake account is created.
  + In snowflake database is create using the command “ CREATE DATABASE DW\_COURSE\_DB;”
  + In that data base one schema is created using the sql statement “CREATE SCHEMA DW\_COURSE\_DB.instacart”
  + **AWS S3:** **Amazon S3** is a **cloud storage service** that lets you store and retrieve **any amount of data** (files, images, videos, backups, logs, etc.) from anywhere on the web.
  + Think of it like a **super secure, highly scalable USB drive** in the cloud.

****

****

**🔄 Common Use Cases:**

* **Backup & disaster recovery**
* **Data lakes (for big data & analytics)**
* **Serving files (e.g., media, software downloads)**
* **Static website hosting**
* **Staging files for data warehousing (e.g., Snowflake COPY INTO)**

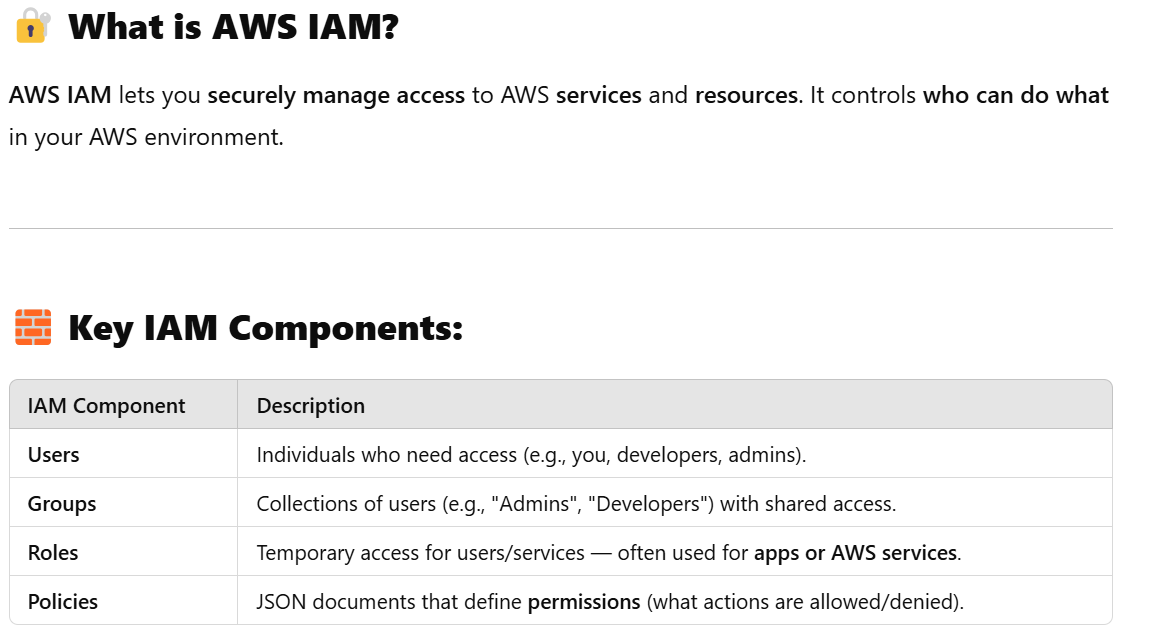
**🛠️ Example – Uploading a File to S3:**

1. **Go to AWS Console > S3 > Create a Bucket.**
2. **Upload files via:** 
   * **Web UI**
   * **AWS CLI (aws s3 cp file.csv s3://my-bucket/)**
   * **Programmatically (Python, Java, etc.)**

**🔐 Security:**

* **Use IAM policies to control who can access your buckets.**
* **Use bucket policies for public/private access.**
* **Enable encryption at rest and in transit.**

**IDENTITY AND ACCESS MANAGEMENT(IAM):**

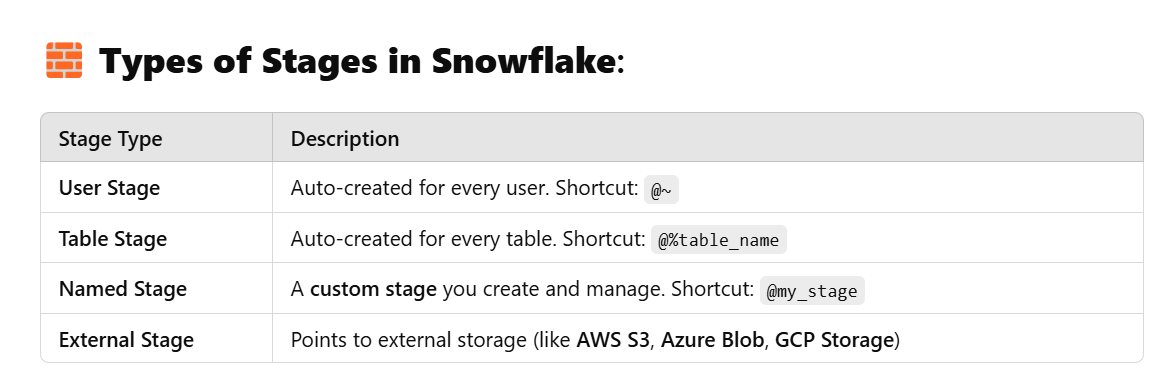
****

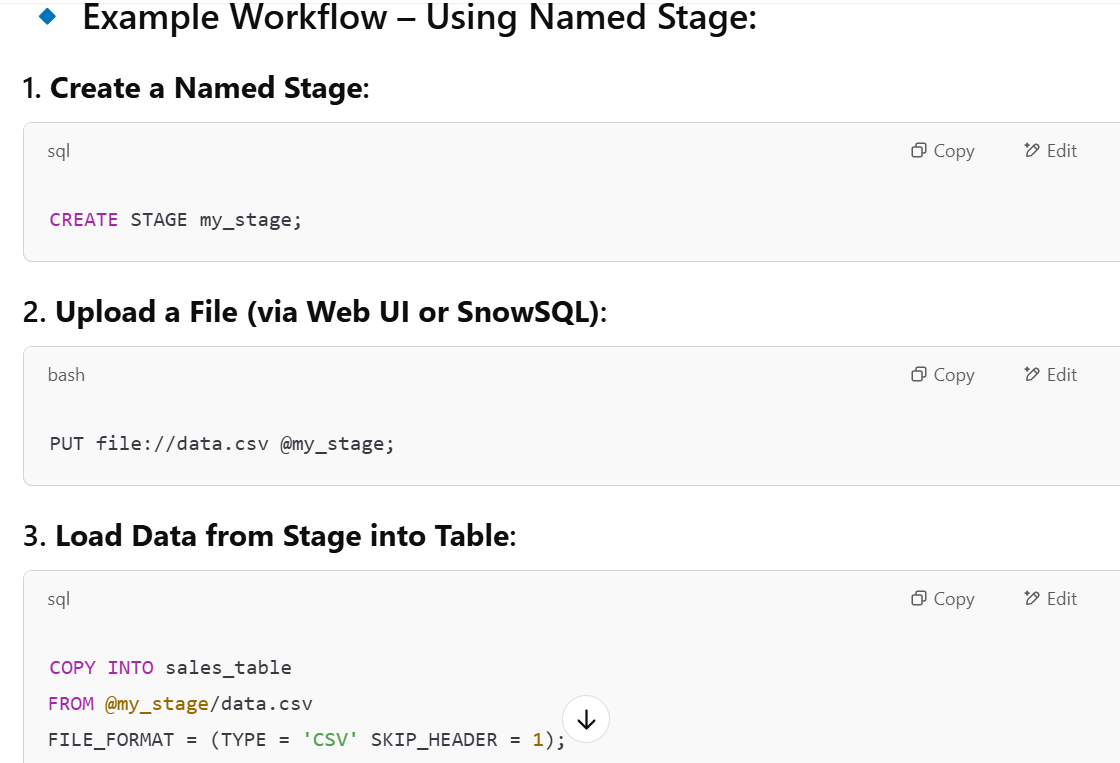
****

**What is a Stage in Snowflake?**

A **Stage** is a **temporary or permanent storage area** where you can **upload files** **before loading them into tables** — or **unload data** from Snowflake to files.

Think of it as a **staging area** (like a "waiting room") for data files **before** they go into your tables.







.

**🚚 Why Stage Data Instead of Loading Directly?**

**1. 🔄 Bulk File Uploads (Not Direct Input)**

Snowflake can’t directly load files from your local machine or S3 **into a table**.  
👉 You **must upload files to a stage first**, then use COPY INTO to load them.

**2. 📂 Support for Big Data Files**

Stages handle **large volumes** of data efficiently.  
Snowflake’s COPY INTO command is **optimized** for **bulk loading from staged files**.

**4. ♻️ Reusability**

Staged files can be **reused**:

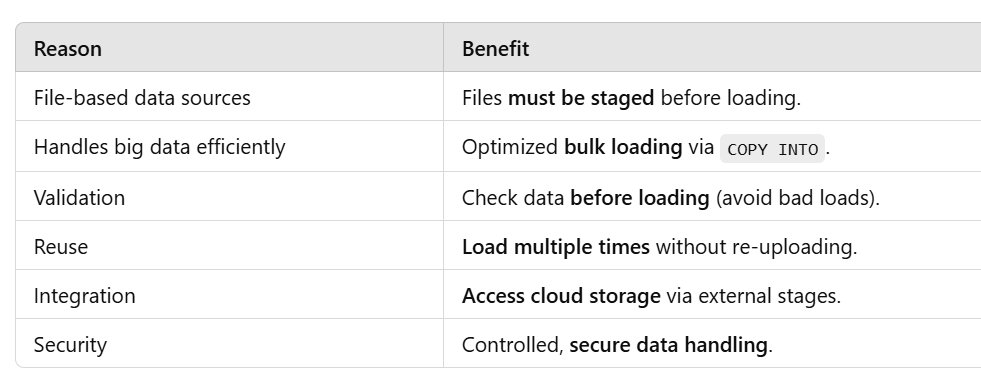
* Load into **multiple tables**.
* Reload if something fails (without re-uploading).
* Useful for **data pipelines**.

**5. 🌐 External Storage Integration**

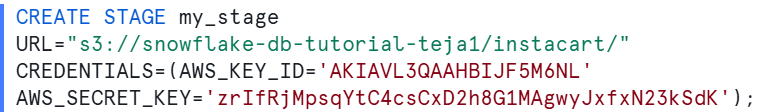
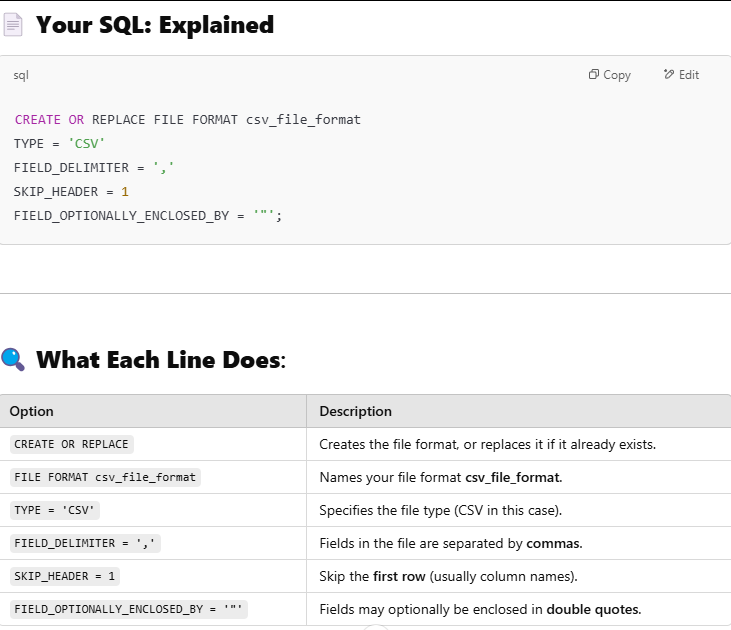
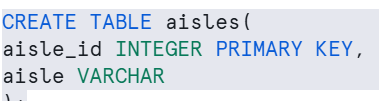
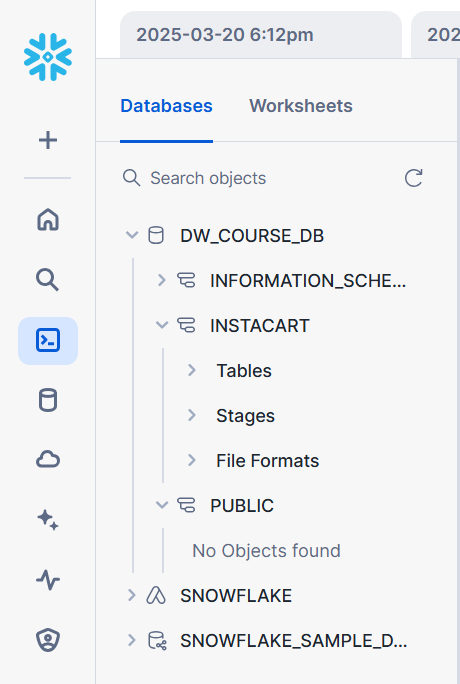
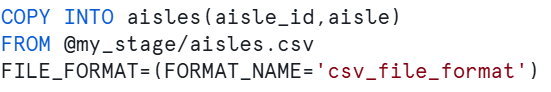
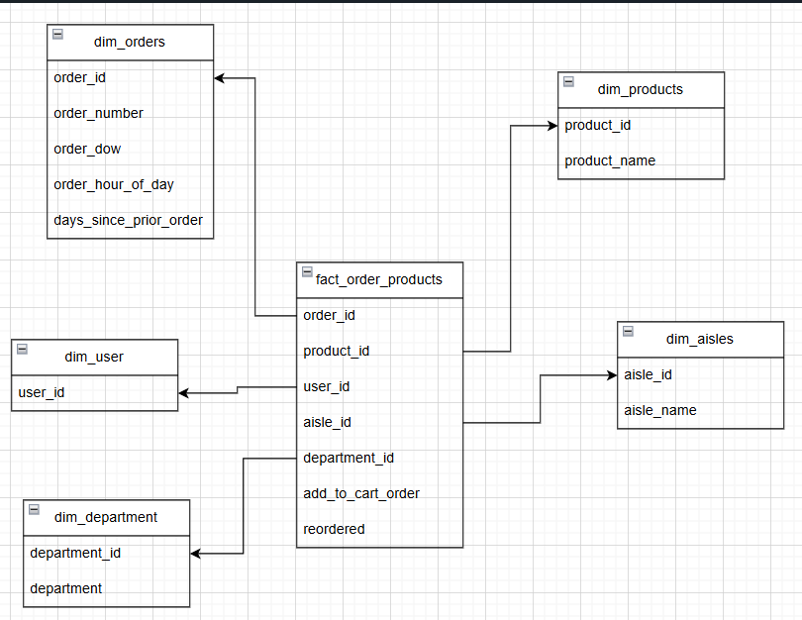
Want to load data from **AWS S3, Azure, GCP**?  
You need to create an **external stage** pointing to the cloud storage — then load data from it.

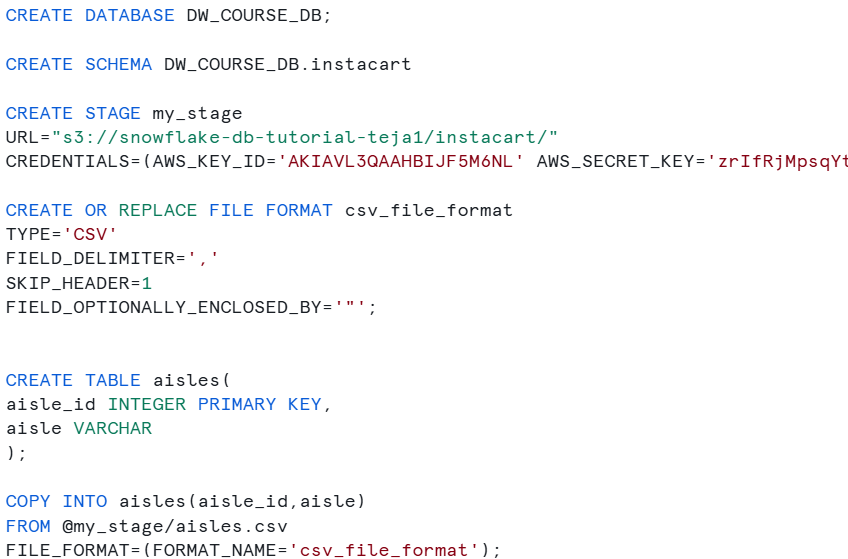
**6. 🔐 Secure & Auditable Process**

Stages support **access control**, encryption, and **logging**.  
You can control **who can upload/download**, and **track activity**.

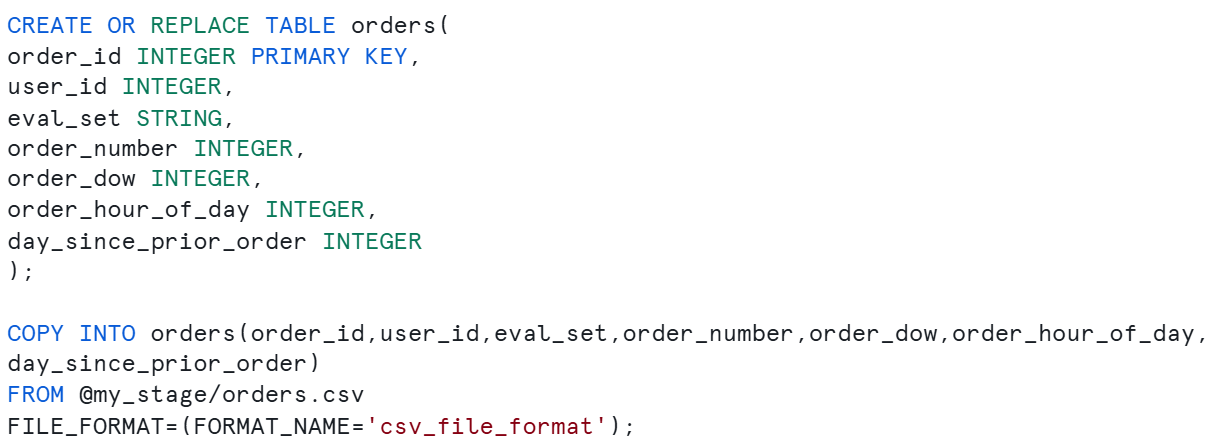


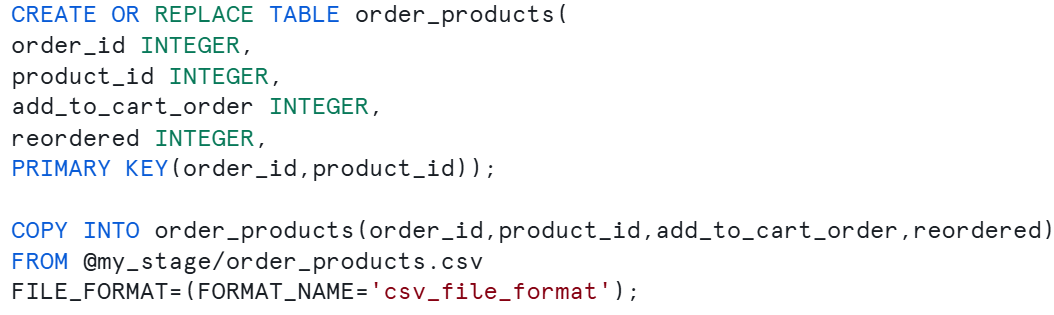
Continue to the project:

* + Then create a stage in the Instacart files. Stage means the place where the files are stored before loading into the warehouse.
  + 
  + In this the stage is created with name my\_stage in the Instacart schema. url also specified from where we need to get the data. The URL is from the aws s3 bucket where we stored data. And credentials are provided to do authentication.
  + We need to tell snowflake on which files we are working. So tell using the command 
  + We need to create tables next using sql statement. 
  + Aisle table is created containing aisle\_id is integer and it is primary key. And one more column is aisle varchar.
  + The created stages, tables, fileformats all are visible under isntacart shema. 
  + We just created table. We need to load data into that table from stage.
  + Sql statement 
  + Copy into command is used to copy data from the stage location to the table. Here specified stage “@my\_stage” means it contains path of the s3 bucket. So in from statement it says at my\_stage location load the aisles.csv file into aisles created table and finally file\_format is specified.
  + Using draw.io tool dimensional modelling diagram is drawn.
  + 
  + The tables are created and the data is loaded into the tables using copy into command.

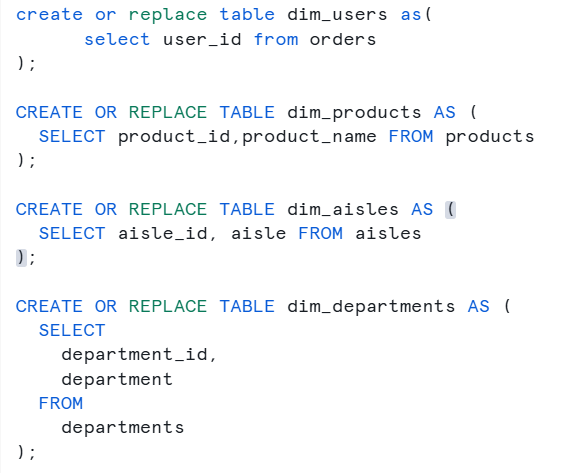


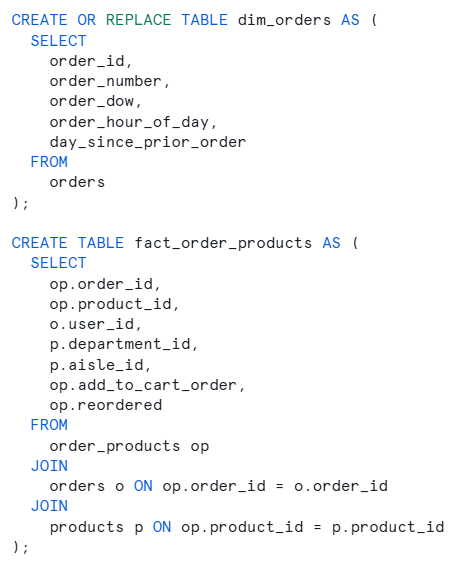






FACT & DIMENSION TABLES





ANALYTICS(Some analysis using SQL queries)



SQL DEEP DIVE:

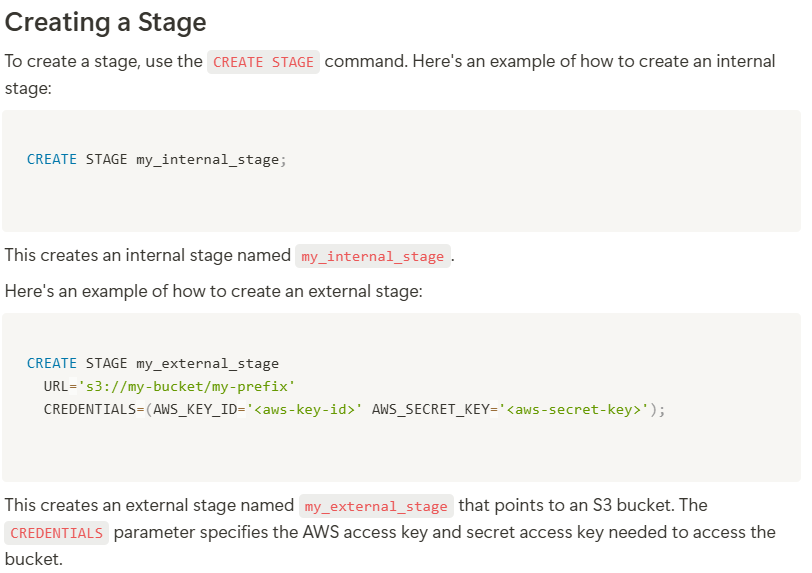
COPY & STAGE:

SnowflakeDB is a cloud-based data warehousing system that allows users to store, manage, and analyze large amounts of data. One of the key features of SnowflakeDB is the ability to load data from external sources into tables using stages and the COPY command.

**What are Stages in SnowflakeDB?**

Stages are named locations in an internal or external location that provide a way to access files from SnowflakeDB. Stages can be internal or external. Internal stages are created in SnowflakeDB and managed by the system. External stages are locations outside of SnowflakeDB that are accessed using an external location.

Stages provide a way to access and load data from external sources into SnowflakeDB tables. This is useful when you need to load data from sources such as S3, Azure Blob Storage, or Google Cloud Storage.



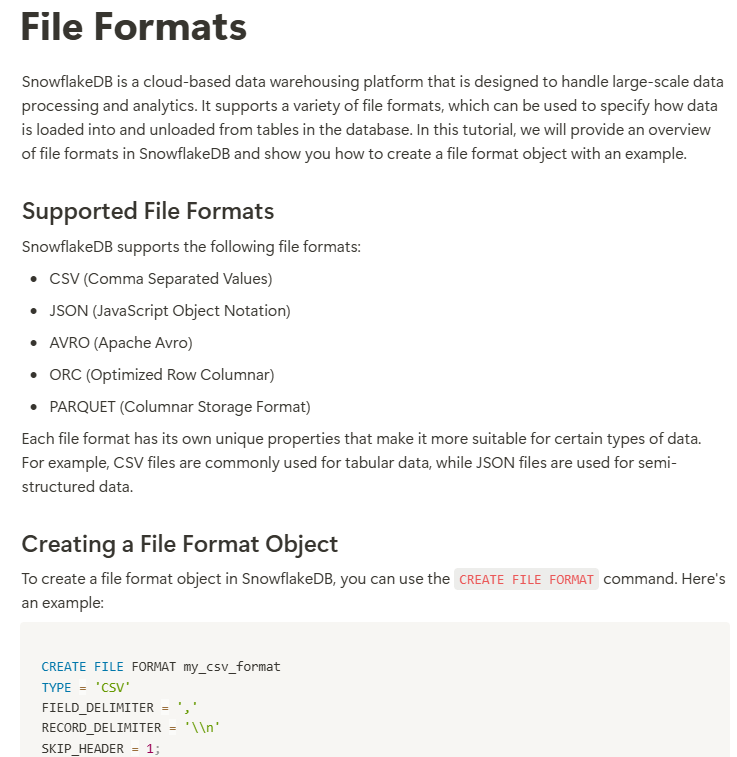


**NOTE:** [**https://github.com/darshilparmar/data-warehouse-snowflake-for-data-engineering/tree/main/SQL%20Code**](https://github.com/darshilparmar/data-warehouse-snowflake-for-data-engineering/tree/main/SQL%20Code)

**Refer to the above repository for sql codes for snowflake deep dive concepts.**

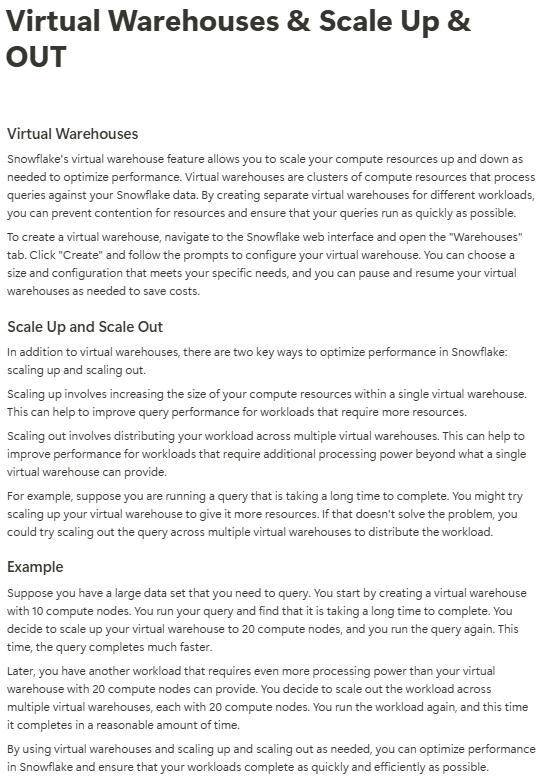
**FILE FORMATS :**

**Descriptive notes for file formats:** [**https://datawithdarshil.notion.site/File-Formats-29339ba303d64b4b8093a87dbd44728f**](https://datawithdarshil.notion.site/File-Formats-29339ba303d64b4b8093a87dbd44728f)

****

**PERFORMANCE OPTIMIZATION:**

* + Creating roles and creating data warehouses and granting permissions to access warehouses.
  + The users are created to use data warehouses independently. This is done to do performance optimization.
  + The snowflake sql document is provided here: <https://github.com/darshilparmar/data-warehouse-snowflake-for-data-engineering/blob/main/SQL%20Code/4.%20Perfromance%20Optimization.sql>

****