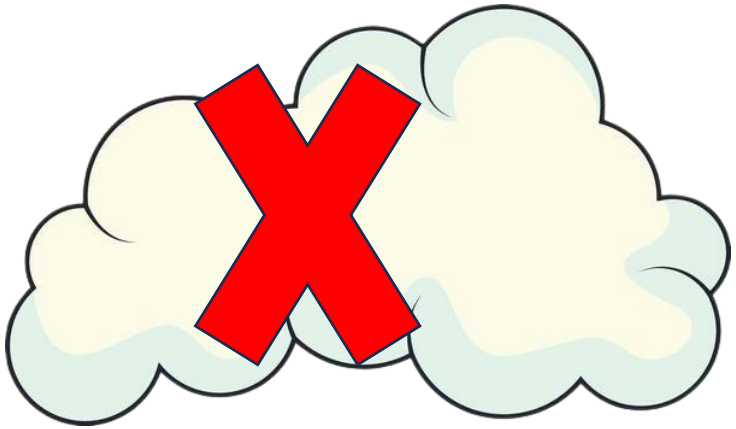
Why Snowflake and problems its solving?

Rigid

Expensive



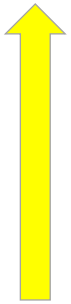
Why Snowflake and problems its solving?

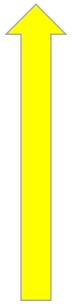
The "Decoupling" Revolution

Traditional Data Warehouses

Data-2-Dollar$

Snowflake Fix





Why Snowflake and problems its solving?

No Resource War

Traditional

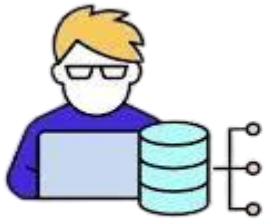
Data-2-Dollar$

Snowflake

Why Snowflake and problems its solving?

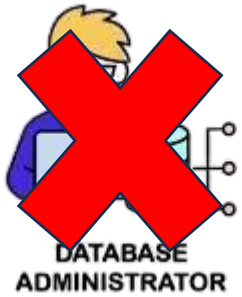
Zero Management

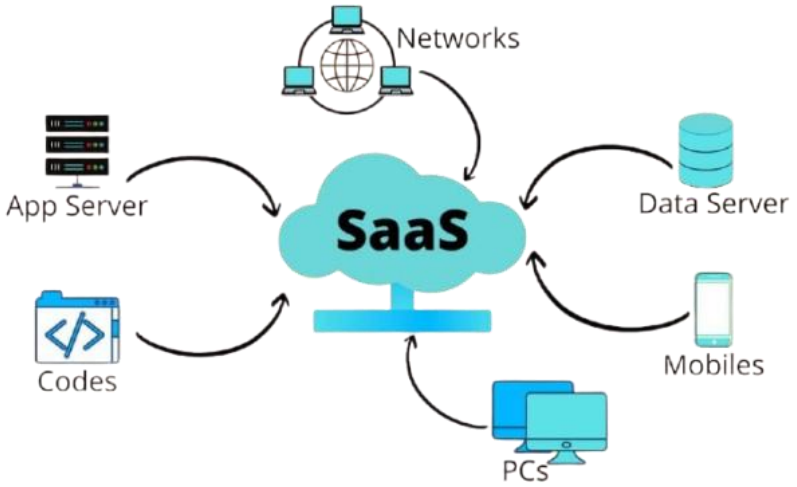
Traditional



Data-2-Dollar$

Snowflake



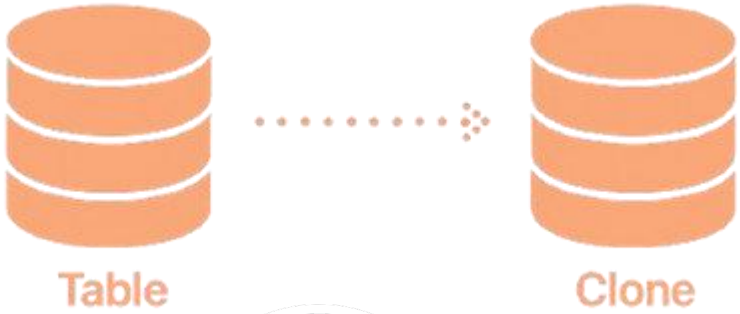


Breaking Data Silos

Secure Data Sharing

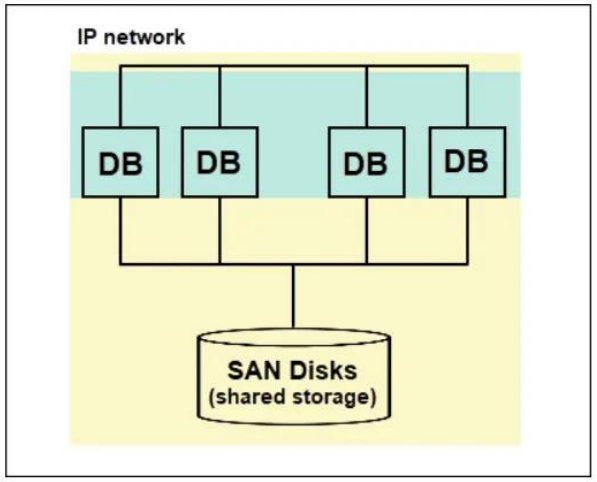
Data-2-Dollar$

Zero Copy Cloning



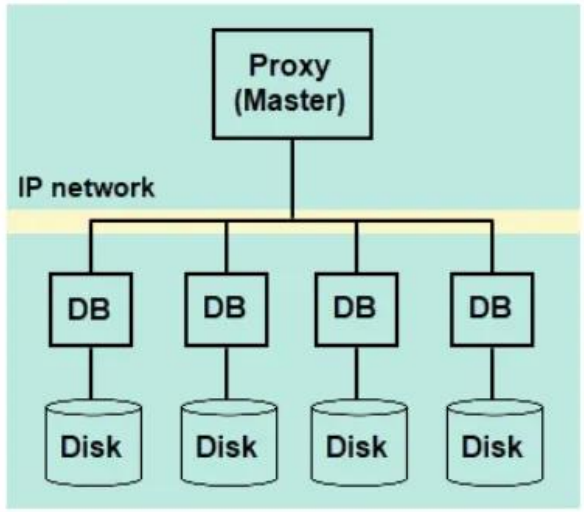
Time Travel

Shared Disk



1 Disk => Multiple nodes

Shared Nothing



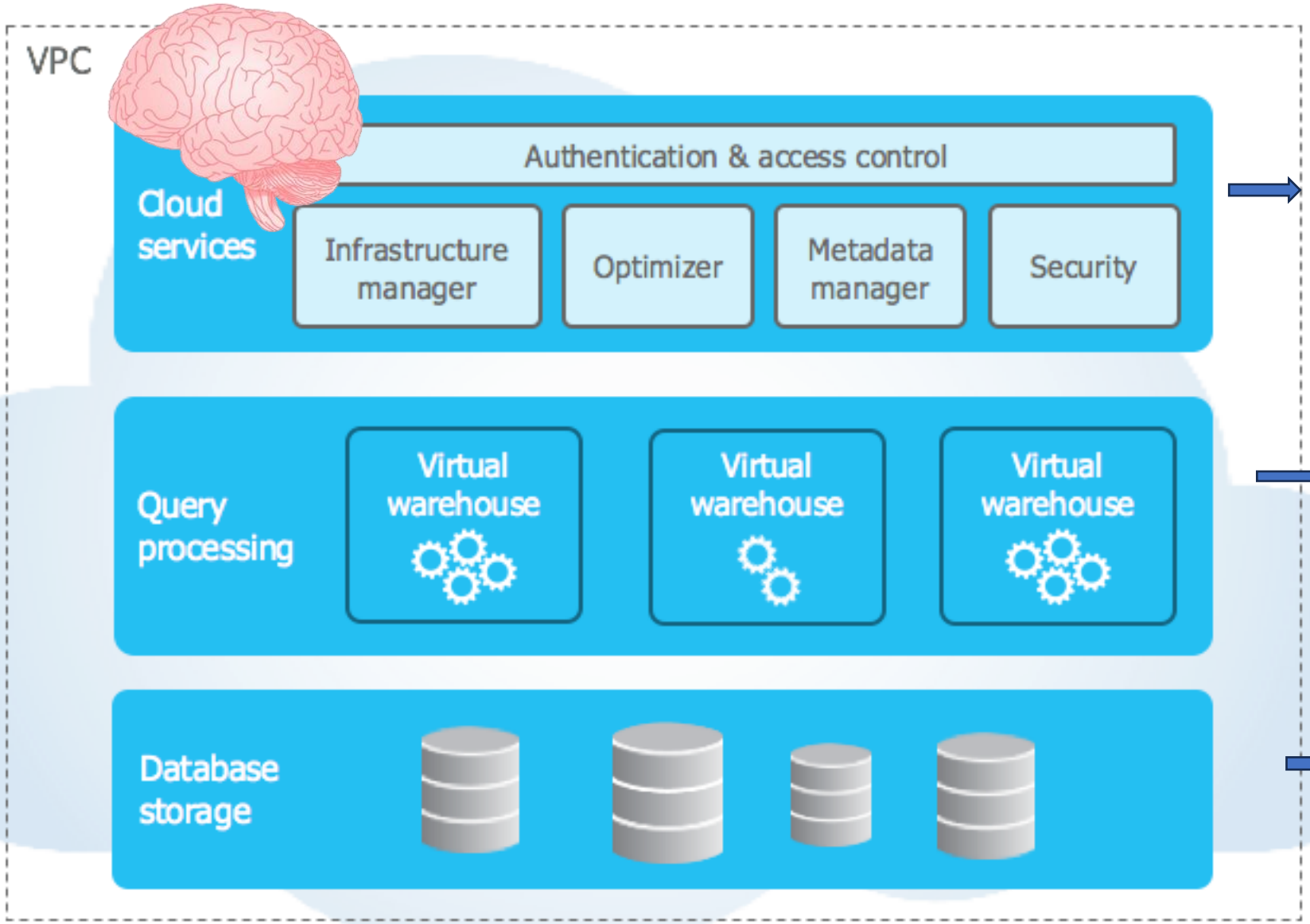
1 Disk => 1 nodes

Limitation

Harder to grow very large

Harder to swap/share info b/w nodes

Snowflake AI Data Cloud Architecture



- Security, authentication & access control
- Snowflake Horizon Catalog
- Infrastructure management
- Metadata management
- Query parsing and optimization
- Regulatory compliance

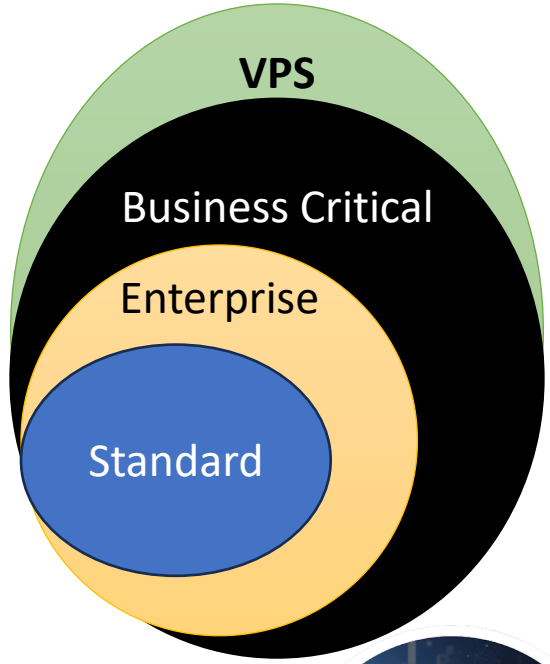
Snowflake processes queries using (MPP) compute clusters, where each node in the cluster stores a portion of the entire data set locally.

Snowflake uses a central data repository for persisted data that is accessible from all compute nodes in the platform.

Data-2-Dollar$

Snowflake Editions

STANDARD	ENTERPRISE	BUSINESS CRITICAL	VIRTUAL PRIVATE SNOWFLAKE (VPS)
Complete SQL data warehouse	**Premier +**	**Enterprise +**	**Business Critical +**
Secure Data Sharing across regions/clouds	Multi-cluster warehouse	HIPAA support	Customer-dedicated virtual servers wherever the encryption key is in memory
Business hour support M-F	Up to 90 days of time travel	PCI compliance	Customer-dedicated metadata store
1 day of time travel	Annual rekey of all encrypted data	Data encryption everywhere	Additional operational visibility
Always-on enterprise grade encryption in transit and at rest	Materialized views	Tri-Secret Secure using customer-managed keys	
Customer-dedicated virtual warehouses	AWS PrivateLink available for extra fee	AWS PrivateLink support	
Federated authentication		Enhanced security policy	
Database replication		Database failover and failback for business continuity	



	Feature Name	Minimum Edition
SCALING & PERFORMANCE		
...to handle 100s of users simultaneously	**Multi-cluster Warehouses**	**Enterprise**
...to speed up specific point-lookup queries	**Search Optimization Service**	**Enterprise**
...to speed up frequent, complex aggregations	**Materialized Views**	**Enterprise**
...to automatically boost power for heavy queries	**Query Acceleration Service**	**Enterprise**
DATA PROTECTION (HISTORY)		
...only 24 hours of "undo" history	**Standard Time Travel**	**Standard**
...up to **90 days** of "undo" history	**Extended Time Travel**	**Enterprise**
...to recover data after Time Travel expires	**Fail-safe (7 days)**	**All Editions**
GOVERNANCE & SECURITY		
...to hide sensitive data (SSNs, Emails)	**Column-level Security (Masking)**	**Enterprise**
...to restrict rows based on user role	**Row-level Security**	**Enterprise**
...to see *who* accessed *what* data and when	**Access History (Account Usage)**	**Enterprise**
HIGH SECURITY & COMPLIANCE		
...to store medical (PHI) or credit card (PCI) data	**HIPAA / PCI-DSS Support**	**Business Critical**
...to use **Private Link** (AWS/Azure/GCP)	**Private Connectivity**	**Business Critical**
...to control encryption with your own keys	**Tri-Secret Secure**	**Business Critical**
BUSINESS CONTINUITY		
...to sync databases across regions	**Database Replication**	**Standard**
...to **Failover** your whole account if a region dies	**Account Failover & Failback**	**Business Critical**
...to keep client connections alive during failover	**Client Redirect**	**Business Critical**

Domain 1.0: Snowflake AI Data Cloud Features and Architecture

1.1 Describe and use the Snowflake architecture

- Cloud Services layer
- Compute layer
- Database Storage layer
- Compare and contrast the different Snowflake editions

1.2 Use Snowflake Interfaces and tools

- Snowsight
- Snowflake CLI
- IDE integrations (e.g., Visual Studio Code)

1.3 Differentiate Snowflake object hierarchy and types

- Organization and account objects
- Database objects
  - Stages
  - Schemas
  - Tables
  - Views
  - User-Defined Functions (UDFs)
  - File formats
  - Stored procedures
  - Pipes

1.4 Configure virtual warehouses

- Types
  - Snowpark Optimized
  - Standard (Gen 1 and Gen 2)
  - Default warehouse for Notebooks* (Feature will not be tested until it's globally GA)
- Scaling policies
- Warehouse type and configurations based on use cases
  - Ad-hoc queries
  - Data loading
  - BI and reporting
- Best practices
  - Sizing (up, down)
  - Scaling (in, out)
  - Auto-Suspend
  - Workloads
    - Different teams
    - High concurrency
    - Complex queries

Snowflake CLI

Snowflake CLI's open-source nature means that developers can leverage the community's collective knowledge and contributions to improve and enhance the tool.

In a very general words,

Instead of clicking through the web interface (the Snowsight UI) to move files or manage tasks, you type specific commands into your terminal or command prompt. It's designed to be fast, scriptable, and efficient for developers and data engineers.

As a command-line interface (CLI), Snowflake CLI provides several benefits for developers, such as:

Speed and efficiency

- executing commands from terminal
- No repetitive or complex tasks.

Automation

- automate tasks and workflows.
- reduce the risk of errors or inconsistencies

Portability

- platform-independent

Customization

- using any modules and scripts
- tailor it as per your needs

Accessibility

- Access remotely

Version control

- integrate with Git
- collaborate more effectively.
- resolve conflicts

Snowflake object hierarchy and types

Organization handles the bill and the accounts.

ORGANIZATION
ACCOUNT
RESOURCE INTEGRATION
SECURITY INTEGRATION
SECURITY & MANAGEMENT

ORGANIZATION
NETWORK POLICY
REPLICATION
DATABASE GROUP
SHARE
EXTERNAL VOLUME
Account
SHARE
USER
USER

Account handles the security and the compute power.

Databases & Schemas handle the organization.

VIRTUAL WARHOUSE
Database
Database
Role
User
SECURITY & MANAGEMENT

Database Role
Schema

Schema Objects handle the actual data.

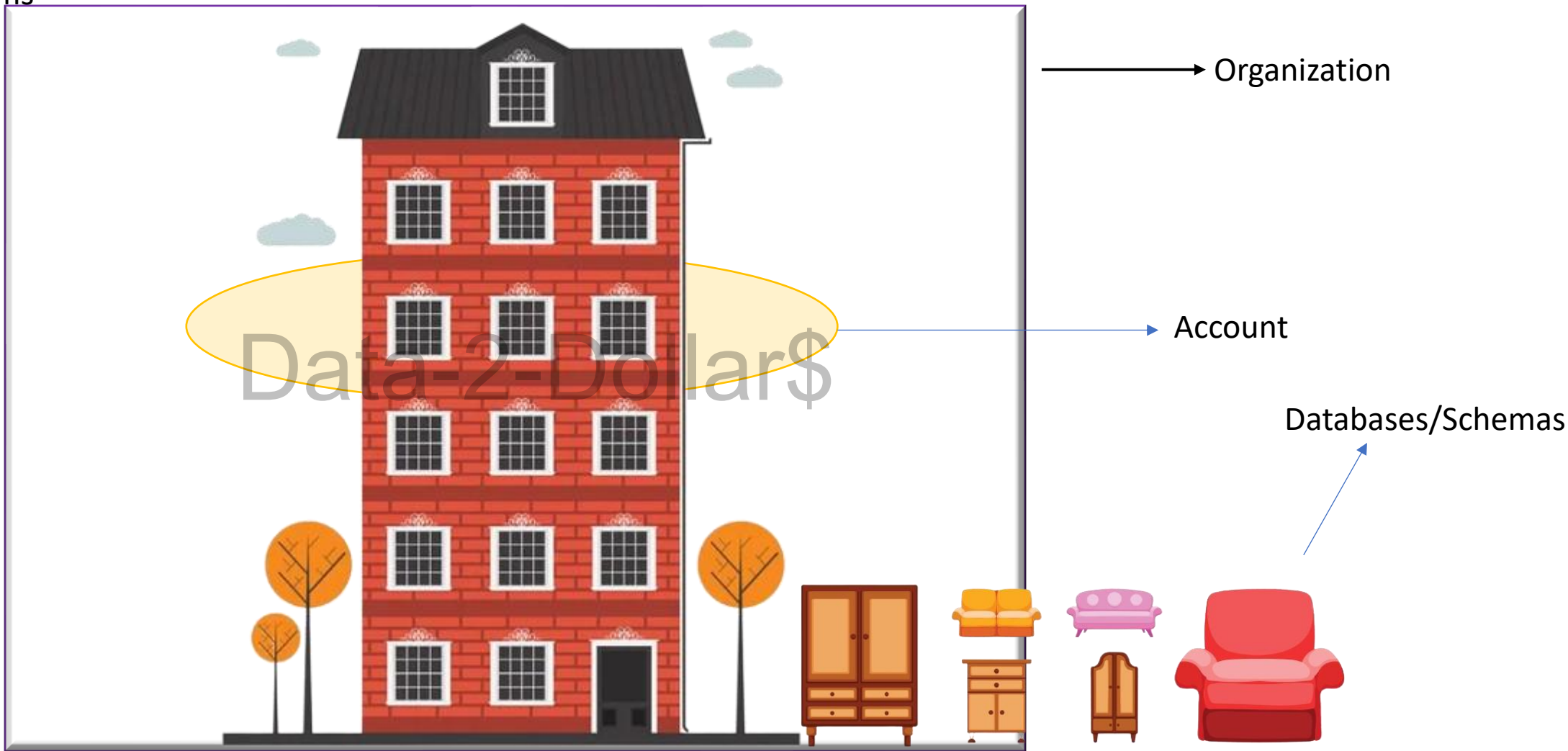
Stuff

Table
View
View
Stored Procedure
UDF
MATERIALIZED VIEW)

Organization and account objects

An organization is a **first-class Snowflake object** that links the accounts owned by your business entity. Organizations simplify account management and billing, Replication and Failover/Failback, Snowflake Secure Data Sharing, and other account administration tasks.

This feature allows organization administrators to view, create, and manage all of your accounts across different regions and cloud platforms



Snowflake Accounts

Account Scalability: Anyone can have any number of accounts.

Unique Access: Each account has its own unique URL.

Infrastructure: Each account is deployed on a single Cloud Service Provider (CSP)—either AWS, Azure, or GCP.

Geography: Each account exists in a single geographic region.

Edition: Each account exists in a single Snowflake edition (e.g., Standard or Enterprise).

Account Usage Tracking: Every account comes with a built-in, read-only database called SNOWFLAKE.

It contains the ACCOUNT_USAGE schema.This is where you find views to track how many credits you've spent, who logged in, and which tables are taking up the most storage.

Account Name Format: This is the current standard, which uses your Organization name.

Format: https://[org_name]-[account_name].snowflakecomputing.com
Example: https://my-org-account123.snowflakecomputing.com

Database Objects

- Stages
- Schemas
- Tables
- Views
- User-Defined Functions (UDFs)
- File formats
- Stored procedures
- Pipes
- Shares
- Sequences
- ML models
- Applications

Stages in Snowflake

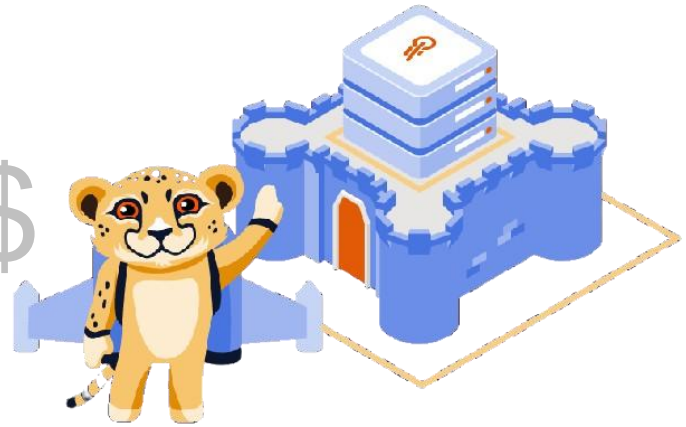
Snowflake Stages are locations where data files are stored (“staged”) which helps in loading data into and unloading data out of database tables. The stage locations could be internal or external to Snowflake environment.

Types of Snowflake Stages

Snowflake supports two types of stages for storing data files used for loading/unloading.

Internal stages store the files internally within Snowflake.

External stages store the files in an external location (AWS S3 bucket or Azure Containers or GCP Cloud storage) that is referenced by the stage.



Tables

Table Type	Persistence	Time Travel	Fail-safe	Use Case
Permanent	Persistent until dropped	Up to 90 days	7 days	Long-term data storage
Temporary	Session duration	None	None	Short-term data processing
Transient	Persistent until dropped	0-1 day	None	Intermediate or staging data
External	Data stored externally	None	None	Querying data in external storage
Dynamic	Virtual, data generated on-the-fly	N/A	N/A	Real-time data generation
Hybrid	Combines local and external data	Varies	Varies	Flexible data management
Iceberg	Persistent, open table format	Configurable	Configurable	Large-scale data analytics

What Snowflake Documentation says?

Transient Tables

Snowflake supports creating transient tables **that persist until explicitly dropped** and are available to all users with the appropriate privileges.

Transient tables are similar to permanent tables with the key **difference that they do not have a Fail-safe period.**

External Tables

An *external table* is a Snowflake feature that you can use to query data stored in an external stage as if the data were inside a table in Snowflake. **The external stage is not part of Snowflake, so Snowflake doesn't store or manage the stage.** External tables let you store (within Snowflake) **certain file-level metadata, including filenames, version identifiers, and related properties.**

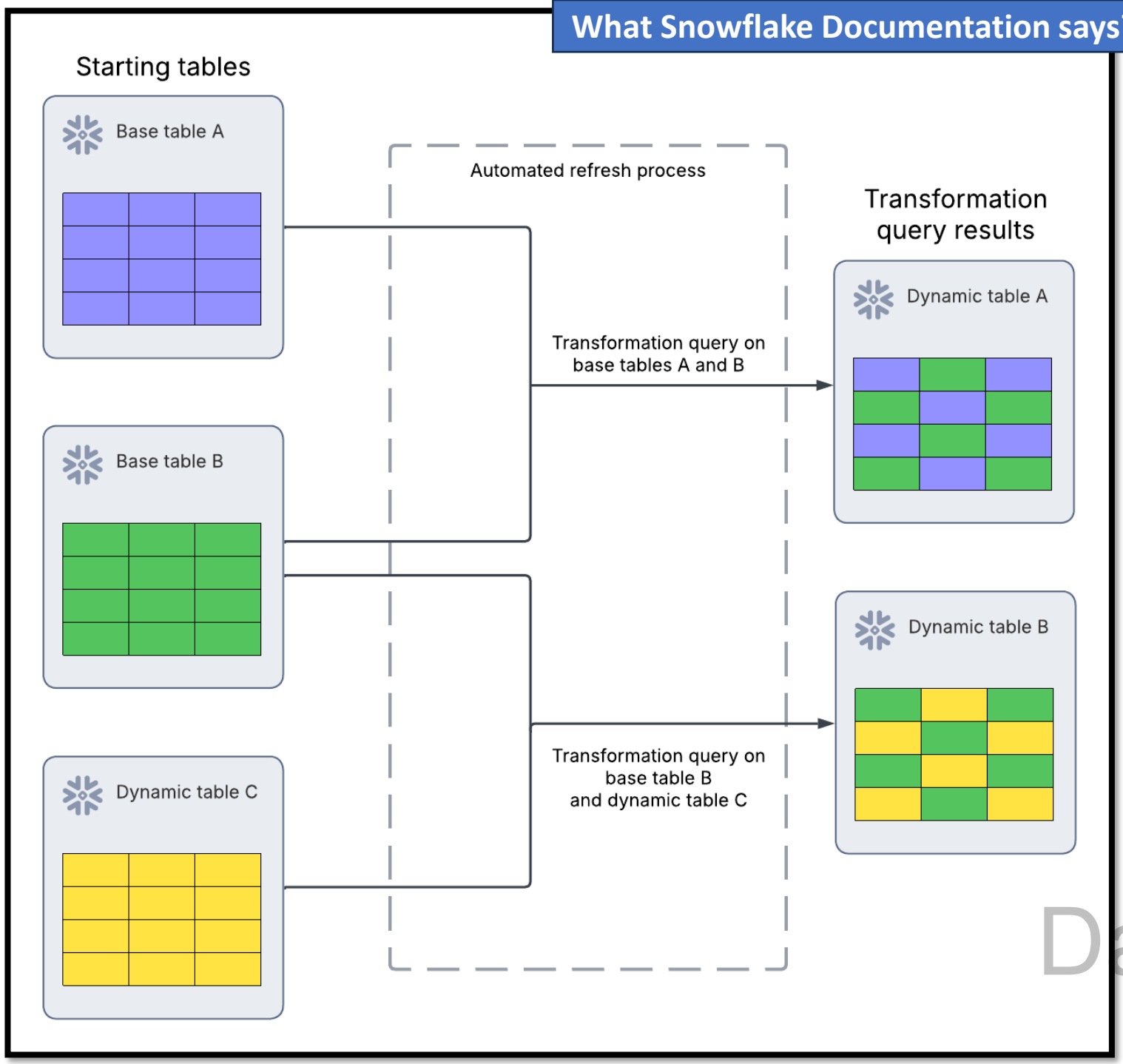
External tables can access data stored in any format that the COPY INTO <table> command supports, **except XML.**

External tables are read-only. You can't perform data manipulation language (DML) operations on external tables. Querying data in an external table might be slower than querying data that you store natively in a table within Snowflake. For **optimal query performance when you work with Parquet files**, consider using Apache Iceberg™ tables instead.

Dynamic tables

Dynamic tables are tables that **automatically refresh based on a defined query** and target freshness, simplifying data transformation and pipeline management **without requiring manual updates or custom scheduling.**

What Snowflake Documentation says?



Dynamic tables aim to refresh within the **target lag** you specify.
For example, a target lag **of five minutes ensures that the data in the dynamic table is no more than five minutes behind** data updates to the base table.

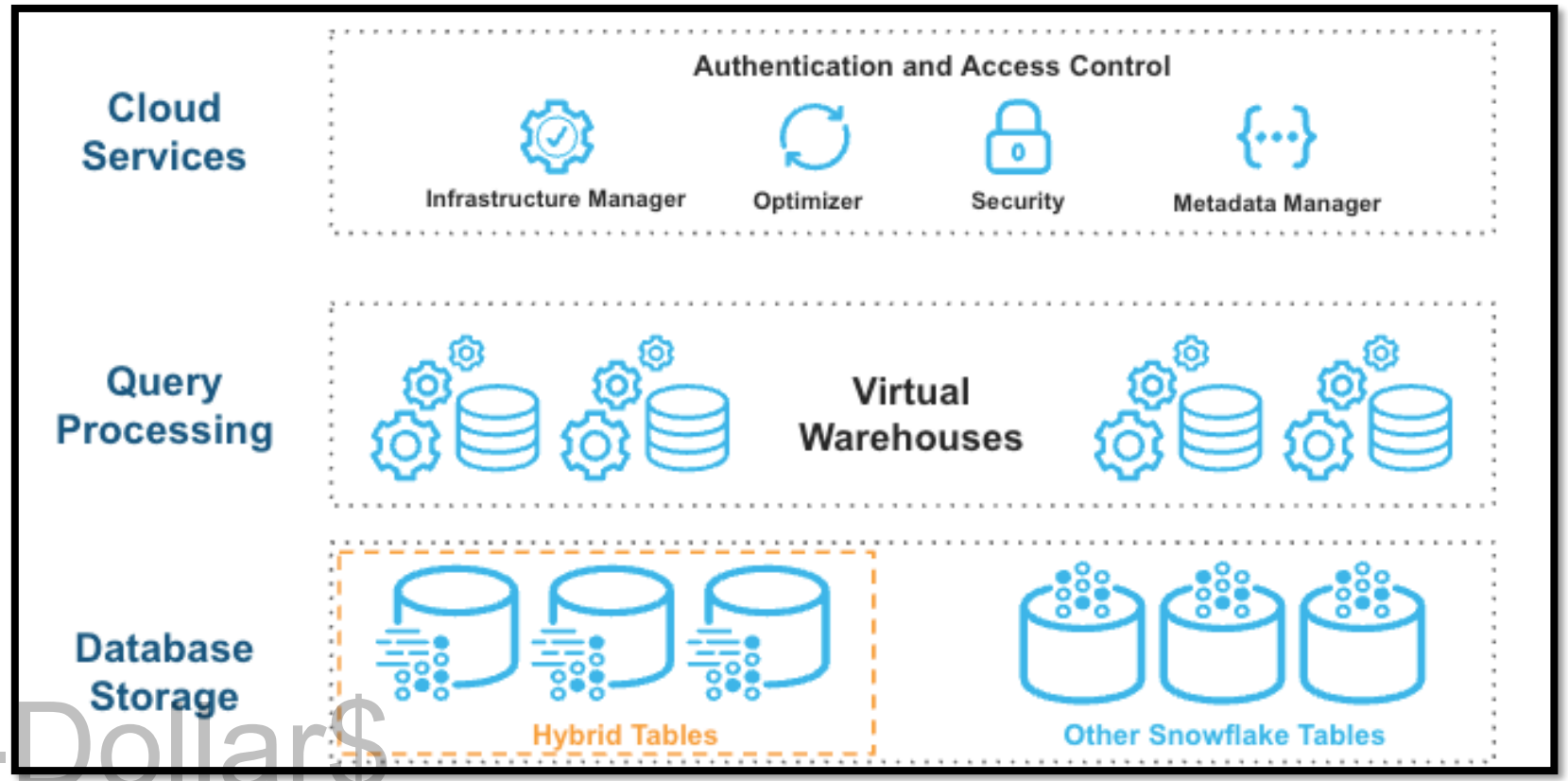
When to use dynamic tables

- You want to **materialize query results** without writing custom code.
- You want to avoid manually tracking data dependencies and **managing refresh schedules.**
- You want to **chain together multiple tables** for data transformations in a pipeline.

What Snowflake Documentation says?

A hybrid table is a Snowflake table type that **is optimized for low latency and high throughput using index-based random reads and writes**. Hybrid tables provide a **row-based storage engine that supports row locking for high concurrency.**
Hybrid tables also **enforce unique and referential integrity constraints**, which are critical for transactional workloads.

Hybrid tables **leverage a row store** as the primary data store to provide excellent operational query performance.
When you write to a hybrid table, the data is written directly into the row store.
Some data may also be cached in columnar format on your warehouse in order to provide better performance for analytical queries



What Snowflake Documentation says?

Apache Iceberg™ tables combine the **performance and query semantics of typical Snowflake tables with external cloud storage** that you manage. They **are ideal for existing data lakes that you cannot, or choose not to, store in Snowflake.**

Iceberg tables use the Apache Iceberg™ **open table format** specification, which provides an abstraction layer on data files stored in open formats .

You are responsible for all management of the external cloud storage location, including the configuration of data protection and recovery. Snowflake **does not provide Fail-safe storage for Iceberg tables**.

There is an Iceberg catalog which enables a compute engine to manage and load Iceberg tables.

The 3-Layer Architecture

Iceberg doesn't just store data; it organizes it into three distinct layers of files on S3:

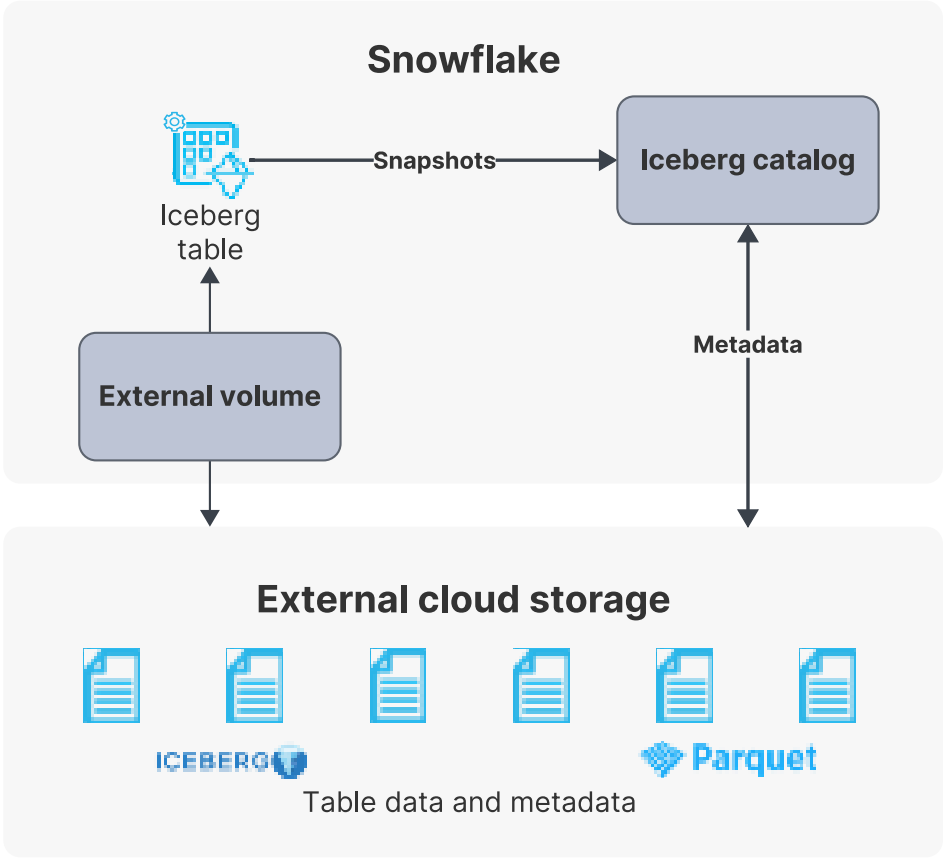
Data Files: These are the actual records, stored in Parquet format.

Manifest Files: These act as an index**. They track which data files belong to a specific version of the table** and store "min/max" stats for each column (used for data pruning).

Metadata Files: This is the **"Source of Truth."** It tracks the table's schema, partitioning, and "snapshots" (history).
When you create an Iceberg table, you have to choose an **Iceberg Catalog**. This is the **"brain" that tracks where the latest metadata** file is located.

Why use Iceberg?

No Vendor Lock-in: If you decide to leave Snowflake, your data is already in S3 in a standard format. **You don't have to "export" it.**
Interoperability: You can query the same S3 files with Snowflake for BI and Spark for Machine Learning at the same time.
Performance: Unlike standard External Tables (which can be slow), **Iceberg tables use the metadata layer to skip unnecessary files, making them nearly as fast as native Snowflake tables.**



Views

Snowflake supports two types of views:

- Non-materialized views (usually simply referred to as “views”)
- Materialized views.

Secure Views:

Both non-materialized and materialized views can be defined as *secure*. Secure views have advantages over standard views, including improved data privacy and data sharing; however, **they also have some performance impacts to take into consideration.**

However, Snowflake recommends to create a materialized view when *all* of the following are true:

1. The query results from the view don’t change often.
2. The results of the view are used often.
3. The query consumes a lot of resources.

User Defined Functions

- UDF allows you to perform operations that are not available through the built-in, system-defined functions.
- Create UDF whenever there is a need to repeat the same functionality.
- Snowflake supports 4 languages for writing UDFs.
  - SQL
  - Java Script
  - Java
  - Python
- Snowflake UDFs can return scalar(Just a value or a string) and tabular results.
- Snowflake supports UDF overloading means support functions with same name but different params.

Sample UDFs

SCALAR: Returns output for each input we are passing.

```
create function area_of_circle(radius float)
  returns float
  as
  $$
     pi() * radius * radius
  $$
  ;

Select area_of_circle(4.5);
```


TABULAR: Can return zero, one or multiple rows.

```
create function t()
    returns table(name varchar, age number)
    as
    $$
        select ‘Ravi’ , 34
        union
        select ‘Latha’ , 27
        union
        select ‘Madhu’, 25
    $$
    ;
```

Stored Procedures

- Stored procedures allow you to write procedural code that includes sql statements, conditional statements, looping statements and cursors.
- Snowflake supports 5 languages for writing procedures.
  - Snowflake Scripting(SQL)
  - Java Script
  - Java
  - Scala
  - Python
- From a stored procedure, you can return a single value or tabular data.
- Supports branching and looping.
- Supports dynamically creating a SQL statement and execute it.

Sample Procedure

```
CREATE or REPLACE PROCEDURE  LOAD_TABLE1()
 RETURNS VARCHAR
  LANGUAGE javascript
  AS
  $$
          var rs = snowflake.execute( { sqlText:
                    `INSERT INTO table1 ("column 1")   SELECT 'value 1' AS "column 1" ;`
                   }
          );

          return 'Done' ;

  $$
;
```

Data-2-Dollar$

UDFs Vs Procedures

Stored Procedures	UDF
Stored procedure may or may not return results	Functions must return results (A Value or Table)
Procedures Are Called as independent statements	Functions are called in SQL statements
Values returned by Procedures are not directly usable in SQL	Values returned by Functions can be directly usable in SQL
Single procedure per CALL statement	Single SQL statement can call multiple functions
Procedures can execute DDL and DML operations	UDFs do not have access to perform database operations

Data-2-Dollar$

File Formats

formatTypeOptions

-- If TYPE = CSV (Flat Files)

COMPRESSION = AUTO | GZIP | SNAPPY | NONE
RECORD_DELIMITER = '<string>'
FIELD_DELIMITER = '<string>'
FILE_EXTENSION = '<string>'
SKIP_HEADER = <integer>
TIMESTAMP_FORMAT = '<string>' | AUTO
TRIM_SPACE = TRUE | FALSE
FIELD_OPTIONALLY_ENCLOSED_BY = '<character>' | NONE
NULL_IF = ('<string>' [, '<string>' ...])
EMPTY_FIELD_AS_NULL = TRUE | FALSE
ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE | FALSE

-- If TYPE = JSON (Semi-Structured)

COMPRESSION = AUTO | GZIP | NONE
DATE_FORMAT = '<string>' | AUTO
TIMESTAMP_FORMAT = '<string>' | AUTO
STRIP_OUTER_ARRAY = TRUE | FALSE
STRIP_NULL_VALUES = TRUE | FALSE
IGNORE_UTF8_ERRORS = TRUE | FALSE
REPLACE_INVALID_CHARACTERS = TRUE | FALSE

SAMPLE SYNTAX: CREATE OR ALTER FILE FORMAT

-- If TYPE = PARQUET (Columnar/Big Data)

COMPRESSION = AUTO | SNAPPY | LZO | NONE
BINARY_AS_TEXT = TRUE | FALSE
USE_LOGICAL_TYPE = TRUE | FALSE
TRIM_SPACE = TRUE | FALSE
NULL_IF = ('<string>' [, '<string>' ...])

-- If TYPE = AVRO / ORC

COMPRESSION = AUTO | GZIP | ZSTD | NONE
TRIM_SPACE = TRUE | FALSE
REPLACE_INVALID_CHARACTERS = TRUE | FALSE
NULL_IF = ('<string>' [, '<string>' ...])

-- If TYPE = XML

COMPRESSION = AUTO | GZIP | NONE
STRIP_OUTER_ELEMENT = TRUE | FALSE
DISABLE_AUTO_CONVERT = TRUE | FALSE
REPLACE_INVALID_CHARACTERS = TRUE | FALSE

Snowpipe

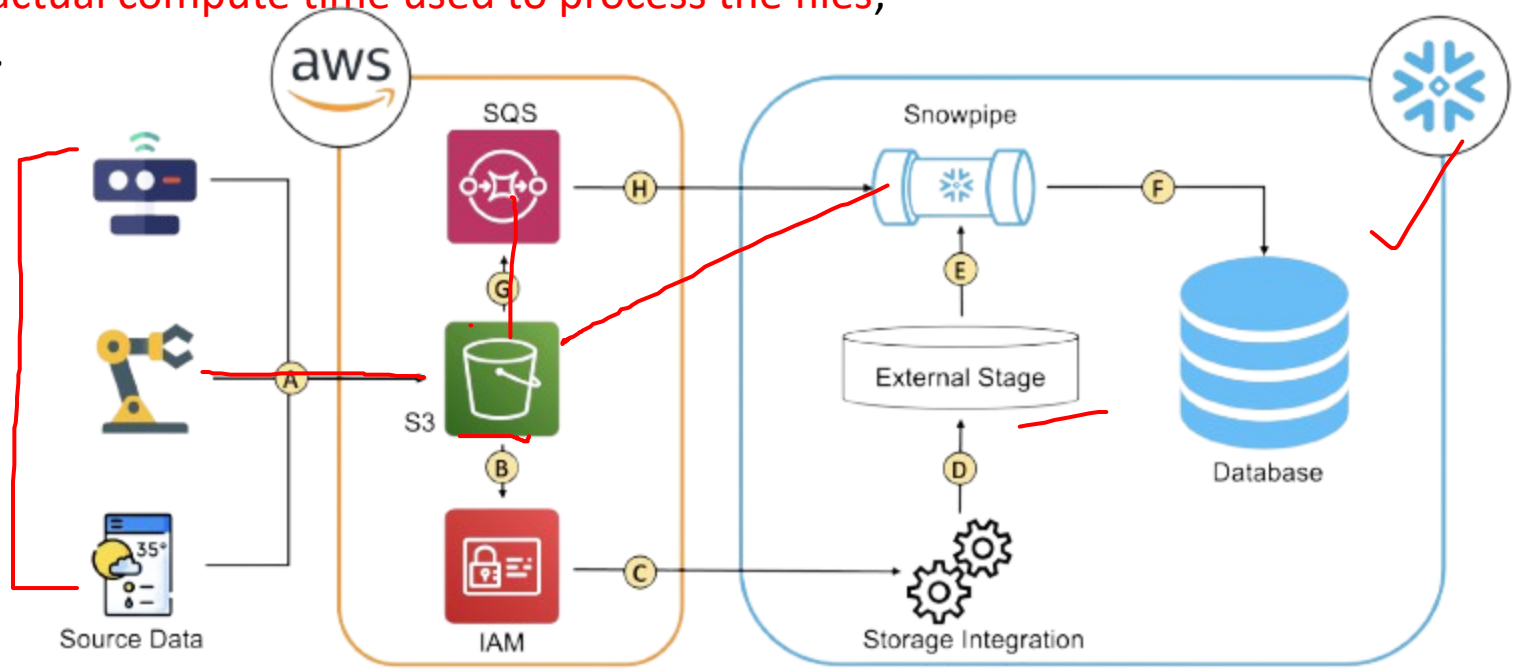
Snowpipe is Snowflake's **automated, continuous data ingestion service**. Unlike traditional "bulk loading" (which uses the COPY command on a schedule**), Snowpipe loads data in micro-batches as soon as it arrives in your cloud storage** (S3, Azure Blob, or GMS).

Characteristics

Serverless: You don't need to manage a virtual warehouse. Snowflake provides and manages the computing power automatically.

Micro-batching: It is designed to load small amounts of data frequently, making data available for analysis in minutes rather than hours.

Cost-Efficient: You are charged based on the actual compute time used to process the files, rather than a running warehouse hourly rate.



Shares

Shares are named Snowflake **objects that encapsulate all of the information required to share a database.**
Data providers add Snowflake objects (databases, schemas, tables, secure views, etc.) to a share using *either or both* of the following options:

Option 1: Grant privileges on objects to a share via a database role.
Option 2: Grant privileges on objects directly to a share.

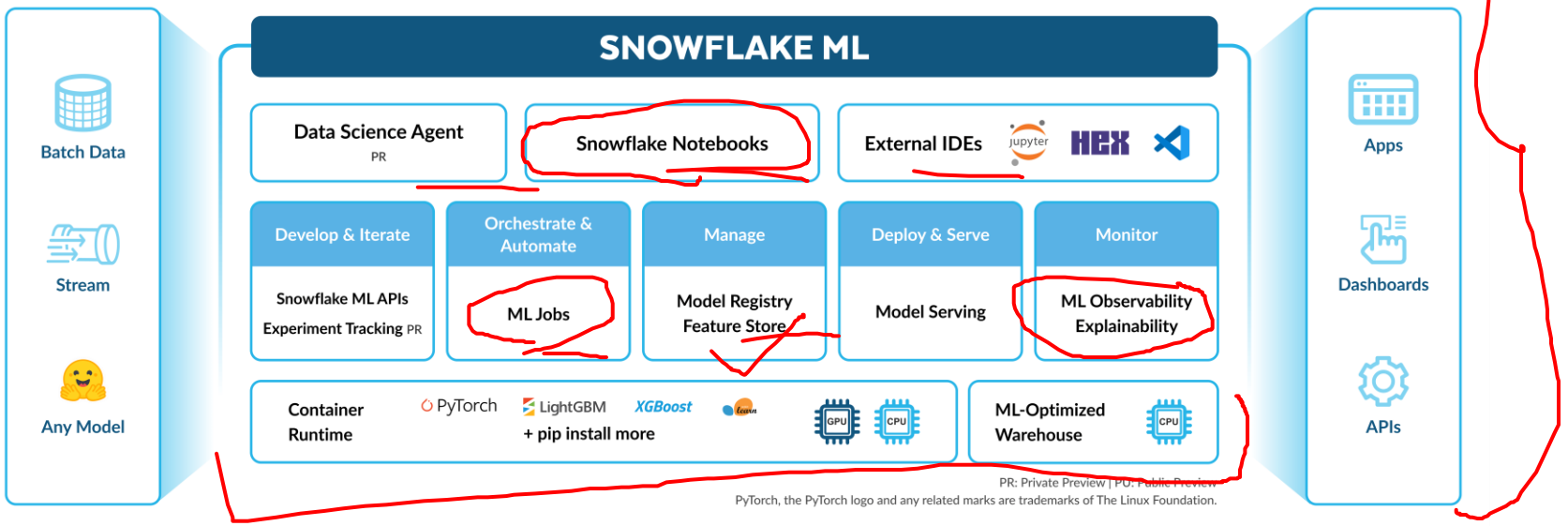
Shares are secure, configurable, and controlled completely by the provider account:

- **New objects added to a share become immediately available to all consumers, providing real-time access to imported data**.
- Updates to existing objects in a share become immediately available to all consumers.
- Access to a share (or any of the objects in a share) can be revoked at any time.

Sequences

Sequences are used to **generate unique numbers across sessions and statements**, including concurrent statements.
They can be used to generate values for a primary key or any column that requires a unique value.

Snowflake ML



Scaling end-to-end ML workflows in Snowflake is seamless. You can do the following:

- Prepare data
- **Create and use features with the Snowflake Feature Store**
- Train models with CPUs or GPUs using any open-source package from Snowflake Notebooks on Container Runtime
- Create experiments to evaluate your trained models against set metrics
- **Operationalize your pipelines using Snowflake ML Jobs**
- **Deploy your model for inference at scale with the Snowflake Model Registry**
- Monitor your production models with ML Observability and Explainability

Snowflake ML is **also flexible and modular. You can deploy the models that you've developed in Snowflake outside of Snowflake and externally-trained models can easily be brought into Snowflake for inference**.

Applications

Snowflake provides several different applications and tools that you can use to access databases in Snowflake.

User interface

Snowsight: The Snowflake web interface

Command-line clients

Snowflake CLI

SnowSQL (CLI client)

Extensions for code editors

Snowflake Extension for Visual Studio Code

Integrating with third-party systems

Snowflake Connectors

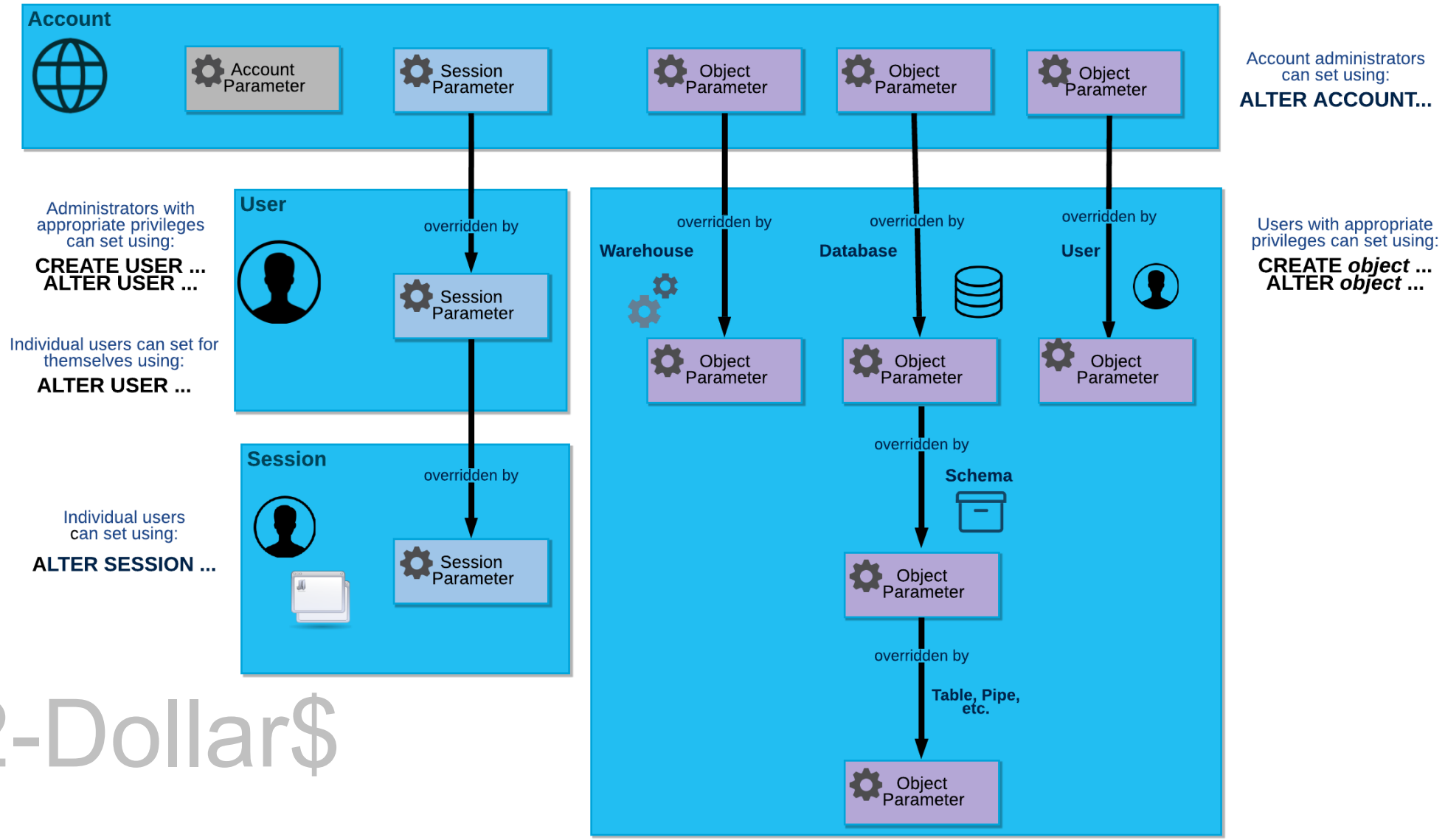
Third-party software

Snowflake Ecosystem

Parameter Hierarchy

This section describes the different types of parameters and the levels at which each type can be set. There are three types of parameters:

- Account parameters
- Session parameters
- Object parameters



Parameter Precedence

In Snowflake, parameters have a clear hierarchy where settings at a more specific level override those at a broader level. The order of parameter precedence, from lowest to highest, is:

- **Default Value (System Level):** Snowflake provides a default value for every parameter.
- **Account Level:** Account administrators can set a default value for the entire account using ALTER ACCOUNT. This value overrides the Snowflake default.
- **Object Level:** Parameters can be set for specific objects like warehouses, databases, schemas, or tables using commands like ALTER WAREHOUSE, ALTER DATABASE, etc. This overrides the account-level setting for that specific object.
- **User Level:** Administrators (or the user themselves) can set parameter defaults for a specific user using ALTER USER. This value overrides the object and account settings for that user.
- **Session Level:** Within an active session, a user can explicitly set parameters using ALTER SESSION. This overrides all lower-level settings for the duration of that session.
- **Statement/Query Level:** Certain parameters (like STATEMENT_TIMEOUT_IN_SECONDS) can be applied to an individual SQL statement or passed as arguments in functions/methods, which takes the highest precedence for that specific operation.

Schema Objects:

TABLES

- ✓ Permanent Table.
- ✓ Temporary Table.
- ✓ Transient Table.
- ✓ External Table.
- ✓ Iceberg Table(Preview).
- ✓ Hybrid Table(Preview).

VIEWS

- ✓ Standard View a.k.a view.
- ✓ Secured View.
- ✓ Materialized View.

STAGES

- ✓ Table stage.
- ✓ User Stage.
- ✓ Named Stage.
  - ✓ Named Internal Stage.
  - ✓ Named External Stage.
    - ✓ AWS
    - ✓ Azure
    - ✓ GCP

- ✓ File Formats.
- ✓ Sequences.
- ✓ Stored Procedures.
- ✓ User Defined Functions.
- ✓ Streams.
- ✓ Tasks.
- ✓ Pipes

Virtual Warehouses

A virtual warehouse, often referred to simply as a “warehouse”, is **a cluster of compute resources** in Snowflake.
A virtual warehouse is available in two types:

- Standard
- Snowpark-optimized

Snowpark-optimized warehouses let you configure the available memory resources and CPU architecture on a single-node instance for your workloads.
The default configuration for a **Snowpark-optimized warehouse provides 16x memory per node compared** to a standard warehouse. You can optionally configure additional memory per node and specify CPU architecture using the resource_constraint property of the CREATE WAREHOUSE or ALTER WAREHOUSE command. The following options are available.

Memory (up to)	CPU architecture	RESOURCE_CONSTRAINT values	Minimum warehouse size
16GB	Default or x86	MEMORY_1X, MEMORY_1X_x86	XSMALL
256GB	Default or x86	MEMORY_16X, MEMORY_16X_x86	M
1TB	Default or x86	MEMORY_64X, MEMORY_64X_x86	L

Standard warehouses comes with 2 options:

- Gen1
- Gen2

GENERATION = '1' represents Snowflake’s original, industry-leading standard virtual warehouses.
GENERATION = '2' represents the next generation of Snowflake’s standard virtual warehouses.

You can alter a standard warehouse and specify a different GENERATION clause or RESOURCE_CONSTRAINT clause to change it from generation 1 to generation 2, or from generation 2 to generation 1. You can make that change whether the warehouse is running or suspended.

```
CREATE OR REPLACE WAREHOUSE next_generation_size_small
  RESOURCE_CONSTRAINT = STANDARD_GEN_2
  WAREHOUSE_SIZE = SMALL;
```


Key Differences

Feature	How Gen1 Works	How Gen2 Works
Data Processing	Optimized for Row/Columnar SQL.	Optimized for Memory-heavy Objects.
Spilling	High risk of "Remote Spilling" on large sorts.	Designed to eliminate remote spilling.
Concurrency	Great for many small queries.	Great for fewer, massive, complex tasks.
Cost Logic	1 Credit per hour (Small).	~1.5 - 2 Credits per hour (Small).

Scaling policies

Snowflake employs scaling policies as a feature of multi-cluster warehouses (Enterprise Edition or higher) running in auto-scale mode. These policies determine how quickly Snowflake adds or removes clusters to manage fluctuating query loads and balance performance with cost.

Snowflake offers two main scaling policies for auto-scale multi-cluster warehouses:

Standard (Default): This policy prioritizes performance by quickly starting new clusters when queries are queued or current resources are deemed insufficient. It's suitable for interactive workloads like BI dashboards. New clusters are added quickly, with a 20-second wait between adding additional clusters for warehouses with 10 or fewer max clusters. Clusters are shut down after a period of low activity once running queries complete.

Economy: This policy prioritizes cost by aiming to keep existing clusters busy before adding new ones, potentially leading to longer query queues. It's recommended for non-urgent background tasks like ETL/ELT jobs. A new cluster is started only if there's enough estimated load.

```
ALTER WAREHOUSE my_warehouse SET SCALING_POLICY = 'ECONOMY';
-- or
ALTER WAREHOUSE my_warehouse SET SCALING_POLICY = 'STANDARD';
```

Warehouse type and configurations based on use cases

1. Ad-Hoc Queries

Goal: Balanced performance for unpredictable, highly unique queries.

- **Warehouse Type: Standard**.
- **Size:** Start with **X-Small (XS) or Small (S)**.
- **Configuration:**
  - **Auto-Suspend:** 60 seconds (Ad-hoc users often leave sessions open; aggressive suspension saves money).
  - **Scaling Policy: Standard** (Prioritizes starting new clusters immediately to prevent user frustration).
  - **Multi-cluster:** Max clusters 2–5 (depending on team size).
- **Why:** Ad-hoc queries are rarely repetitive. Keeping a "warm cache" is less important because users usually query different data each time, so short auto-suspend times are ideal.

2. Data Loading (ETL/ELT)

Goal: High throughput and stability for batch processing.

- **Warehouse Type:** * Use **Standard** for typical SQL-based `COPY INTO` or `INSERT`.
  - Use **Snowpark-Optimized** if your ETL involves heavy Python/Java logic or ML pre-processing that spills to disk.
- **Size: Medium (M) to Large (L)**. (Larger files benefit from larger warehouses).
- **Configuration:**
  - **Auto-Suspend:** 0–60 seconds (Suspend immediately after the batch job finishes).
  - **Scaling Policy: Economy** (Prevents spinning up new clusters for minor spikes, keeping costs predictable for background tasks).
  - **Multi-cluster:** Usually **1 cluster** (Max=Min=1). ETL is typically managed by a scheduler; if you need more speed, "Scale Up" (bigger size) rather than "Scale Out" (more clusters).
- **Why:** Data loading is often a "heavy lifting" task. You want raw power (Vertical Scaling) to handle large files efficiently.

Warehouse type and configurations based on use cases

BI and Reporting

Goal: Low latency and high concurrency for many simultaneous dashboard users.

- **Warehouse Type: Standard**.
- **Size: Small (S) or Medium (M)**.
- **Configuration:**
  - **Auto-Suspend: 10 minutes** (Crucial: This keeps the "Local Data Cache" warm so that different users viewing the same dashboard get near-instant results).
  - **Scaling Policy: Standard**.
  - **Multi-cluster:** Max clusters 3–10.

- **Why:** BI tools (Tableau, Power BI) send many small, concurrent queries. You need "Horizontal Scaling" (Multi-cluster) to prevent "Queueing," where one user's heavy report makes everyone else's dashboard spin.

Use Case	Recommended Type	Size	Auto-Suspend	Scaling Policy
Ad-Hoc	Standard	XS - S	60 sec	Standard
Data Load	Standard / Optimized	M - XL	0-60 sec	Economy
BI / Reports	Standard	S - M	10 min	Standard

Best practices

1. Sizing: Vertical Scaling (Up/Down)

Sizing refers to changing the size of a single warehouse (e.g., Small to Large). Each step up doubles the compute power and the credit cost.

- **Best Practice: Start Small.** Always start with an **X-Small** and monitor the `QUERY_HISTORY` for "spilling."
- **Scale Up (Vertical):** Do this for **Complex Queries**. If a query is taking too long because it's processing massive data volumes, a larger warehouse provides more memory and more threads to handle the task.
- **Scale Down:** If your `Warehouse Load Monitoring` chart shows a lot of white space (idle time) or if queries finish in sub-seconds, you are likely over-provisioned.
- **Note:** You can resize a warehouse instantly with zero downtime, even while queries are running.

2. Scaling: Horizontal Scaling (Out/In)

Scaling refers to adding more "clusters" of the same size to a warehouse (Multi-cluster Warehouse).

Best Practice: Scale Out (Horizontal) for **High Concurrency**. If you have 50 people opening a dashboard at 9:00 AM, scaling *up* to a Large won't help; you need to scale *out* by adding clusters so users don't have to wait in a queue.

Scaling Policies:

Standard: Adds clusters immediately when a queue is detected. Best for user-facing apps.

Economy: Only adds clusters if the system is sure they will be fully utilized for 6 minutes. Best for background/batch tasks.

3. Auto-Suspend & Auto-Resume

These are the most important settings for cost control.

- **Best Practice (Task-Based):** Set Auto-Suspend to **60 seconds** for ETL and Ad-hoc.
- **Best Practice (BI-Based):** Set Auto-Suspend to **10 minutes** for BI tools. Frequent suspending/resuming clears the **Data Cache**, meaning the next user has to pull data from cold S3 storage instead of warm local memory.
- **Auto-Resume:** Always keep this **ON**. There is almost no reason to manually manage warehouse starts.

4. Workload Management Strategy

Don't let your Data Scientists, Finance team, and Automated ETL share the same "General" warehouse.

Different Teams (Isolation)

Create separate warehouses for different departments (e.g., `FINANCE_WH`, `DEV_WH`, `DATA_SCI_WH`).

Benefit: Provides **Resource Isolation** (so a Finance user's runaway query doesn't crash the CEO's dashboard) and **Cost Attribution** (you can see exactly which department is spending the most).

High Concurrency: For workloads with many small queries (Reporting/BI):

- Use a **Multi-cluster Warehouse**.
- Keep the size **Small** but set **Max Clusters** to a higher number (e.g., 5 or 10).

Complex Queries

For massive "Year-over-Year" joins or data science transformations:

- Use a **Large or X-Large** warehouse.
- Consider **Snowpark-Optimized (Gen2)** if you see "Remote Disk Spilling" in the Query Profile.

Micro Partitions

In Snowflake, all data are automatically loaded into partitions, called Micro Partitions. Each micro partition contains 50–500 MB of data in uncompressed format and organized in a columnar fashion.

Snowflake maintains the metadata of each micro partition, such as:

- 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.

Micro-partitioning is automatically performed on all Snowflake tables. Tables are transparently partitioned using the ordering of the data as it is inserted/loaded . This is also known a natural clustering of table.

Emp Id	Emp Name	Salary	Dept Id
1	Tom	1000	1
2	Sam	600	4
3	Sandy	500	2
4	Ram	1100	5
5	Rommy	700	3
6	Daniel	1000	1
7	Somen	600	4
8	Supra	500	2
9	Raj	1100	5
10	Sammy	700	3
11	Paul	500	1

MICRO-PARTITION 1

Emp Id	1	2	3
Emp Name	Tom	Sam	Sandy
Salary	1000	600	500
Dept id	1	4	2

MICRO-PARTITION 2

Emp Id	4	5	6
Emp Name	Ram	Rommy	Daniel
Salary	1100	700	1000
Dept id	5	3	1

MICRO-PARTITION 3

Emp Id	7	8	9
Emp Name	Somen	Supra	Raj
Salary	600	500	1100
Dept id	4	2	5

MICRO-PARTITION 4

Emp Id	10	11
Emp Name	Sammy	Paul
Salary	700	500
Dept id	3	1

What Snowflake documentation says?

Benefits of Micro-partitioning

The benefits of Snowflake's approach to partitioning table data include:

•In contrast to traditional static partitioning, Snowflake micro-partitions are derived automatically; they don't need to be explicitly defined up-front or maintained by users.

•As the name suggests, micro-partitions are small in size (50 to 500 MB, before compression), which enables extremely efficient DML and fine-grained pruning for faster queries.

•Micro-partitions can overlap in their range of values, which, combined with their uniformly small size, helps prevent skew.

•Columns are stored independently within micro-partitions, often referred to as *columnar storage*. This enables efficient scanning of individual columns; only the columns referenced by a query are scanned.

•Columns are also compressed individually within micro-partitions. Snowflake automatically determines the most efficient compression algorithm for the columns in each micro-partition.

•The micro-partition metadata maintained by Snowflake enables precise pruning of columns in micro-partitions at query run-time, including columns containing semi-structured data.

Table Clustering

Table Clustering is the process of physically organizing table data within its micro-partitions to align with specific column values.

How it Works?
Snowflake automatically stores data in encrypted, cloud-optimized micro-partitions (usually between 50 MB and 500 MB of uncompressed data).

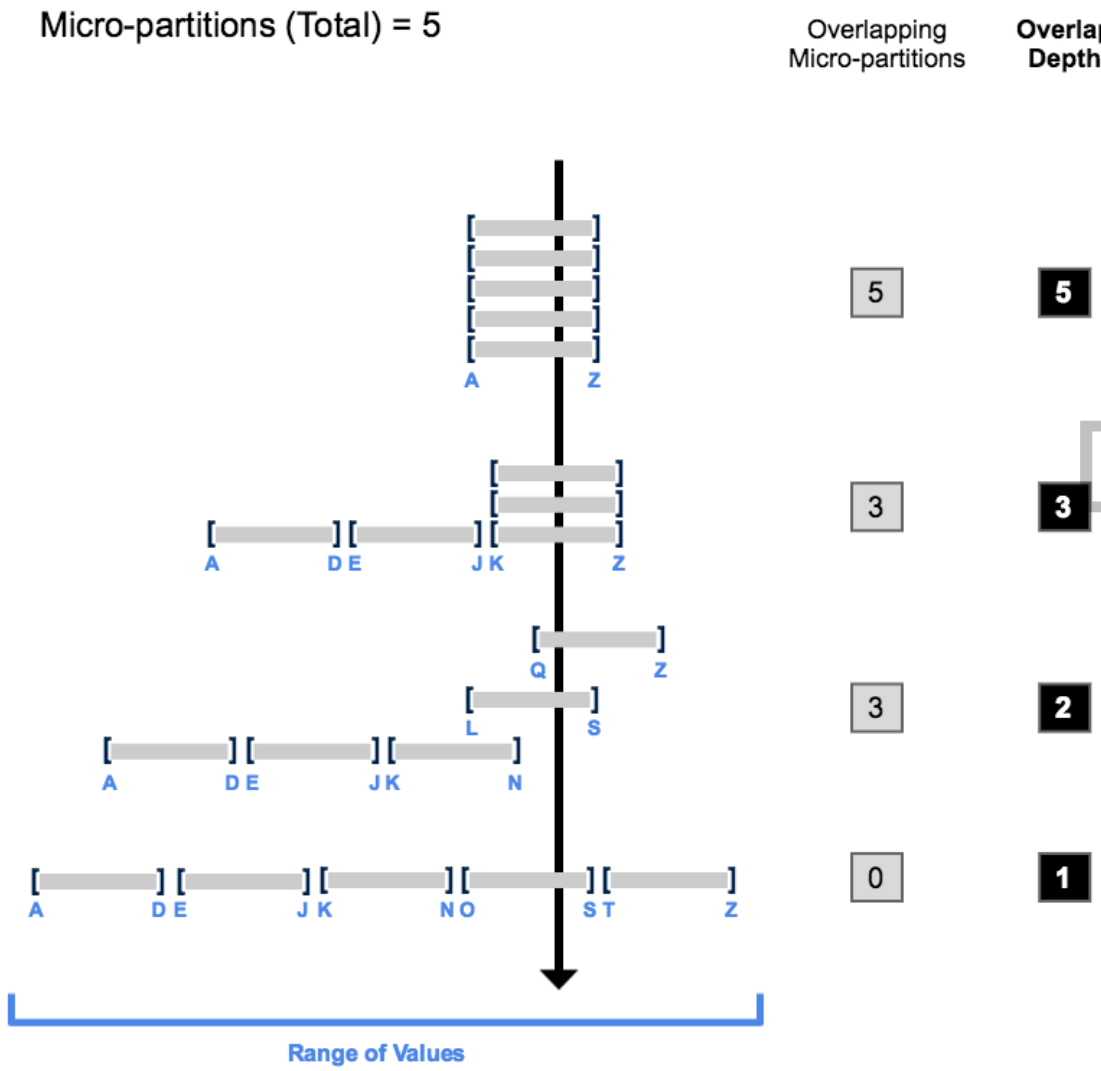
Natural Clustering: When you load data, Snowflake naturally clusters it by the order it arrives (usually by timestamp).

The Problem: Over time, as data is deleted, updated, or merged, the "natural" order breaks down. This might leads to data skew, where a specific range of values (like a Date or ID) is scattered across thousands of micro-partitions.

The Solution: Clustering allows you to define a Clustering Key—one or more columns that Snowflake uses to co-locate related data.

When to use?
Although, snowflake's default "Natural Clustering" is incredibly efficient for most small-to-medium datasets. Consider manual clustering only if:

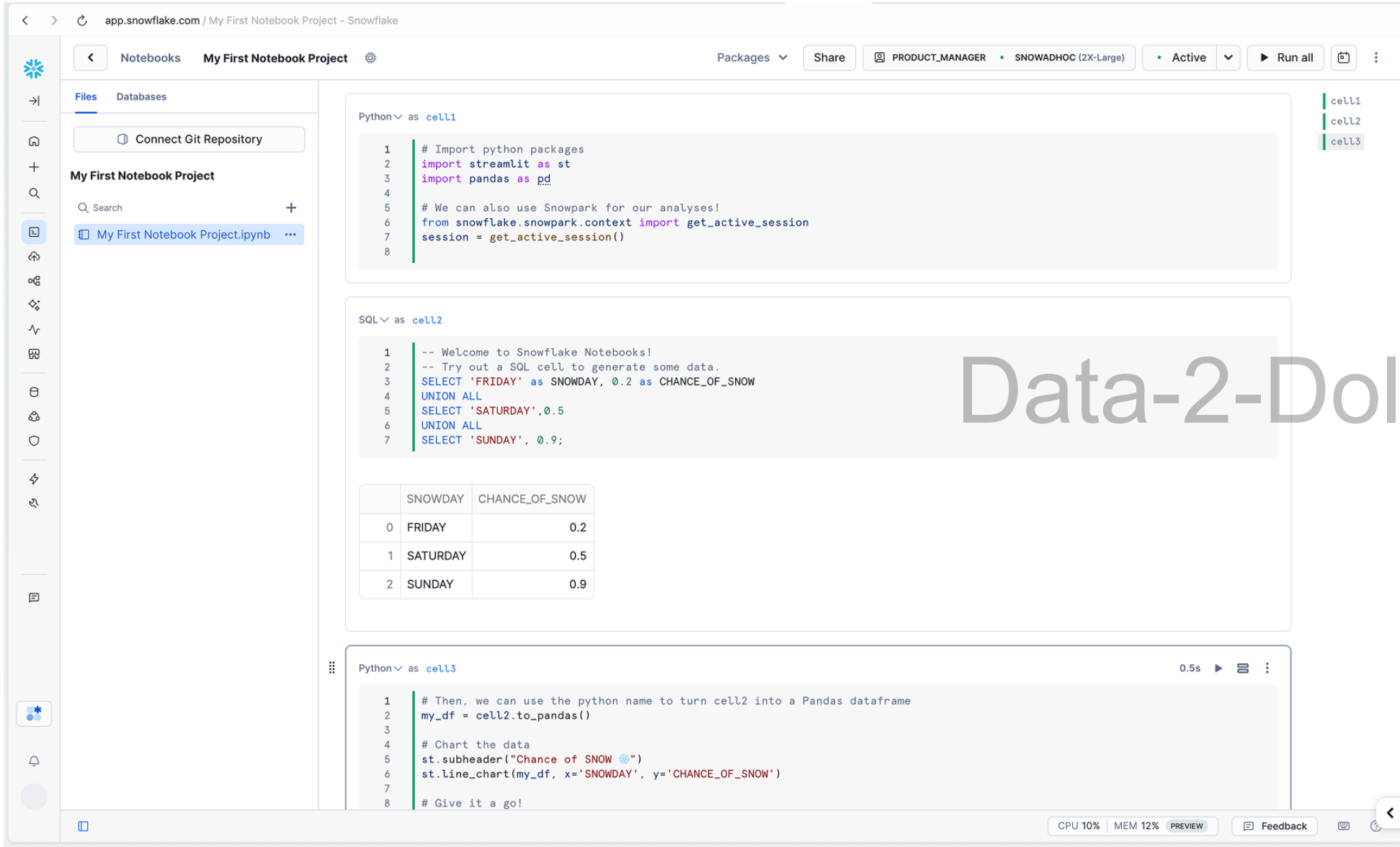
- The table is multi-terabyte in size.
- Query performance has noticeably degraded over time.
- The Clustering Depth is consistently high for your most frequent filter columns.



The clustering depth for a populated table measures the average depth (1 or greater) of the overlapping micro-partitions for specified columns in a table. The smaller the average depth, the better clustered the table is with regards to the specified columns.

Snowflake Notebooks

Snowflake Notebooks is a unified development interface in Snowsight that offers an interactive, cell-based programming environment for Python, SQL, and Markdown. In Notebooks, you can leverage your Snowflake data to perform exploratory data analysis, develop machine learning models, and perform other data science and data engineering workflows, all within the same interface.



Snowflake Notebook Limitations

One File Only: Each Snowflake Notebook can only have one "runnable" file inside it. You can't have a folder full of multiple active notebooks within a single project.

No "Saving" Your Progress on Widgets: If you use a slider or a dropdown (Streamlit) and then refresh your page or close the tab, it resets. It won't remember what you selected.

No Moving or Renaming: If you rename your notebook or move it to a different folder, the link (URL) you saved or shared will break.

No "Copies" for Backup: Notebooks don't support "Replication." This means you can't automatically sync them to a different Snowflake account in a different region for backup.

Repositories (GitHub/Git): If you connect your notebook to a code repository, only the main file you picked will actually run. You can see the other files, but you can't hit "Play" on them.

Restricted Roles: Specific "Snowflake Database Roles" cannot create or run notebooks.

Streamlit in Snowflake

Streamlit in Snowflake is a fully managed service that allows developers to securely build, deploy, and share interactive data applications using pure Python directly within the Snowflake Data Cloud. This integration enables data scientists and developers to create powerful applications on top of their data without moving it to external systems.

Key Features and Benefits

Integrated Platform: Applications run entirely within Snowflake's secure infrastructure, leveraging existing role-based access control (RBAC) and governance policies.

Pure Python Development: Developers can use the familiar, open-source Streamlit library and the Snowpark library to build apps without needing front-end web development experience.

Data Proximity: Apps process data directly where it lives in Snowflake, which eliminates the latency, security risks, and complexity associated with moving large datasets.

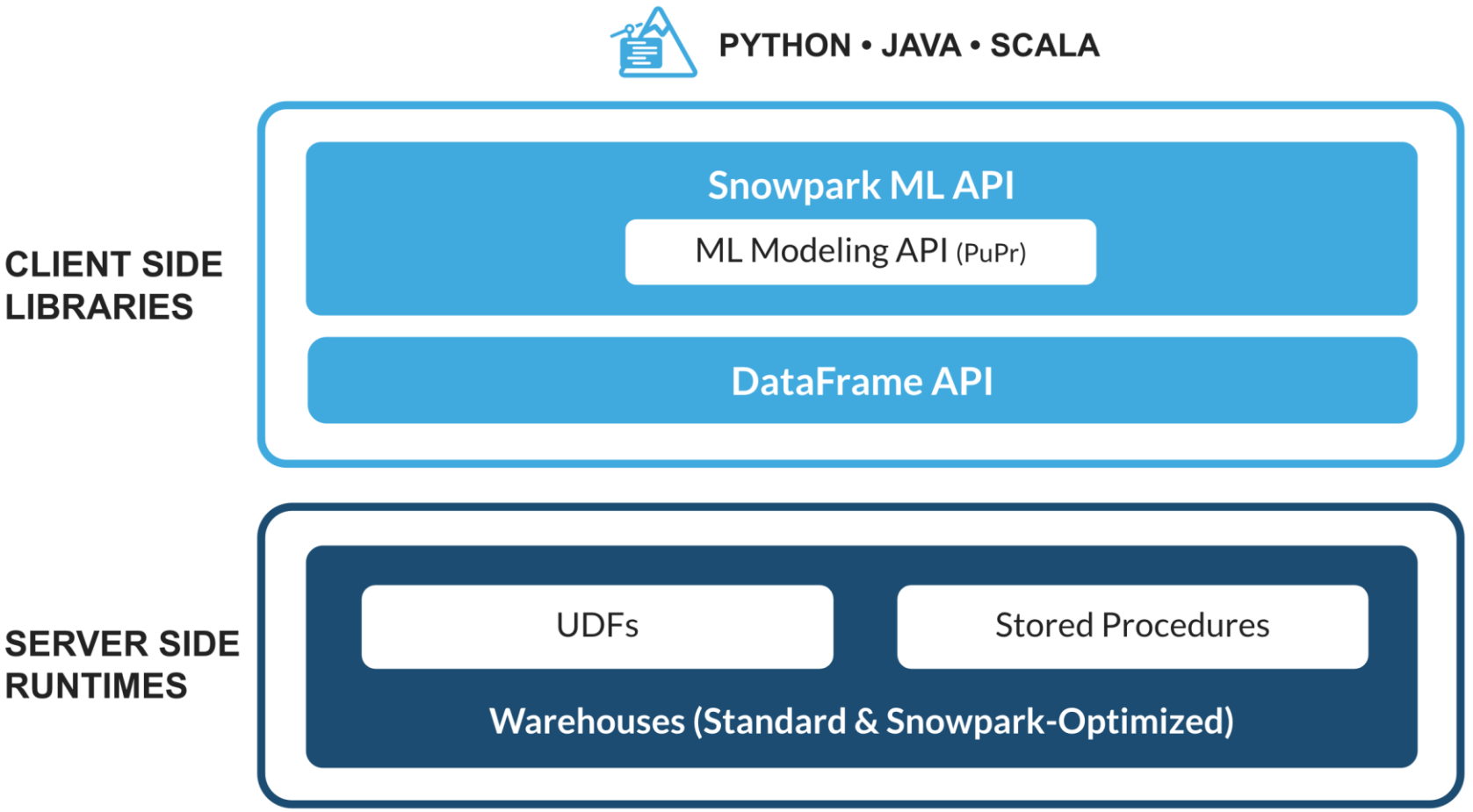
Scalability and Performance: Snowflake manages the underlying compute and storage resources (warehouses or compute pools) for the apps, ensuring they can scale with your data and user needs.

Seamless Deployment: Apps can be created and managed using Snowsight (Snowflake's web interface), SQL commands, or the Snowflake CLI.

AI and ML Integration: The platform works seamlessly with Snowflake Cortex AI and various Python ML/AI libraries, allowing the building of sophisticated data and AI applications, including AI chatbots.

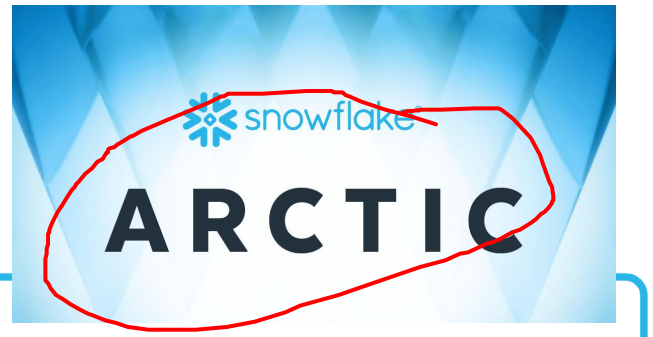
Snowpark

Snowpark is a developer framework for Snowflake that allows you to write code in non-SQL languages—specifically Python, Java, and Scala—and execute it directly within Snowflake's engine.



Snowflake Cortex

Snowflake Cortex is a fully managed service that provides built-in **Artificial Intelligence (AI)** and **Large Language Models (LLMs)** directly within Snowflake.
Essentially, it gives you "AI in a box." You don't need to manage complex GPU infrastructure or move your data to an external AI provider (like OpenAI or Anthropic).
Instead, you can call powerful models using simple **SQL functions** or **Python code**



Data-2-Dollar$

Snowflake Cortex AI

AI SQL functions

AGENTS & INSIGHTS

- Agent Apps — **Snowflake Intelligence**
- Agent APIs — **Cortex Agents**
- Multimodal AI-powered SQL — **Cortex AI Functions**

Cortex Search

PROCESSING & RETRIEVAL

- Unstructured Data Retrieval — **Cortex Search**
- Structured Data Retrieval — **Cortex Analysts**
- Document Processing — **Document AI, Parse, Embed**

Document AI

DATA

- Data Connectors — **Openflow**
- AI Ready Data & Apps — **Snowflake Marketplace**
- Multimodal Data — **Snowflake, External**

MODELS

- OpenAI
- ANTHROP\C
- Meta
- MISTRAL AI_
- deepseek
- snowflake
- **Inference & Fine-Tuning APIs**
- **Embedding Models**

GOVERNANCE

- **RBAC**
- **Guardrails**
- **Evaluations**
- **Monitoring**
- **AI Gateway**

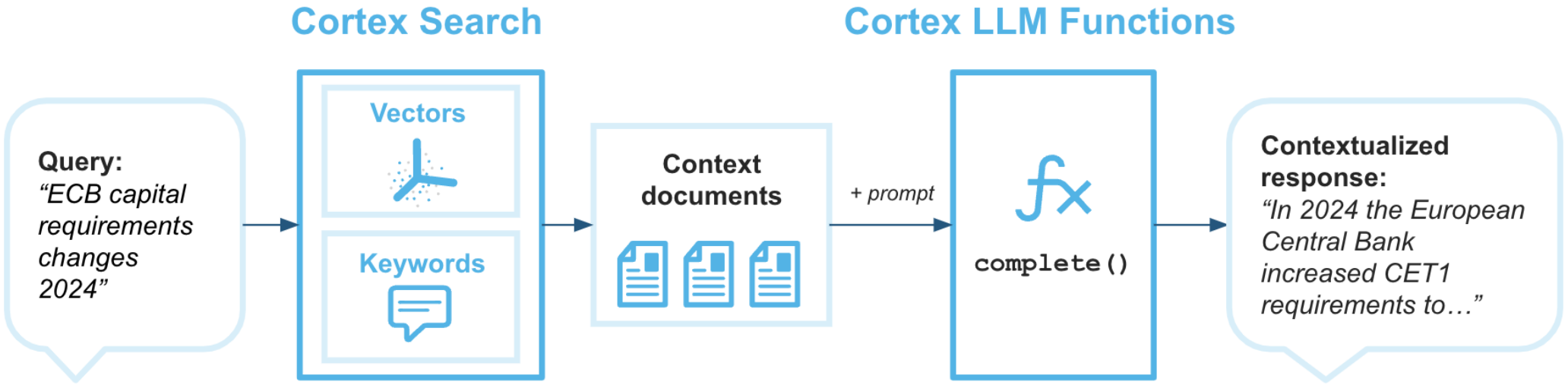
AI SQL functions

- **AI_COMPLETE:** Generates a completion for a given text string or image using a selected LLM.
- **AI_CLASSIFY:** Classifies text or images into user-defined categories.
- **AI_FILTER:** Returns True or False for a given text or image input, allowing you to filter results in SELECT, WHERE, or JOIN clauses.
- **AI_EMBED:** Generates an embedding vector for a text or image input, which can be used for similarity search, clustering, and classification tasks.
- **AI_EXTRACT:** Extracts information from an input string or file, for example, text, images, and documents.
- **AI_SENTIMENT:** Extracts sentiment from text.
- **AI_SUMMARIZE_AGG:** Aggregates a text column and returns a summary across multiple rows.
- **AI_SIMILARITY:** Calculates the embedding similarity between two inputs.
- **AI_TRANSCRIBE:** Transcribes audio and video files stored in a stage, extracting text, timestamps, and speaker information.
- **AI_PARSE_DOCUMENT**:
- **AI_TRANSLATE**

AI_REDACT: Redact personally identifiable information (PII) from text.

SUMMARIZE (SNOWFLAKE.CORTEX): Returns a summary of the text that you've specified.

Cortex Search




```
CREATE OR REPLACE CORTEX SEARCH SERVICE transcript_search_service
  ON transcript_text
  ATTRIBUTES region
  WAREHOUSE = cortex_search_wh
  TARGET_LAG = '1 day'
  EMBEDDING_MODEL = 'snowflake-arctic-embed-l-v2.0'
  AS (
    SELECT
        transcript_text,
        region,
        agent_id
    FROM support_transcripts
);
```

What Snowflake Documentation says?

Document AI is a Snowflake AI feature that uses Arctic-TILT, a proprietary large language model (LLM), to extract data from documents. Document AI processes documents of various formats and extracts information from both text-heavy paragraphs and the content in a graphical form, such as logos, handwritten text (signatures), tables, or checkmarks.

When to use Document AI

Document AI is best used when:

- You want to turn unstructured data from documents into structured data in tables.
- You want to create pipelines for continuous processing of new documents of a specific type.
- Business users with domain knowledge prepare the model, and the data engineers working with SQL prepare pipelines to automate the processing of new documents.