**Snowflake**

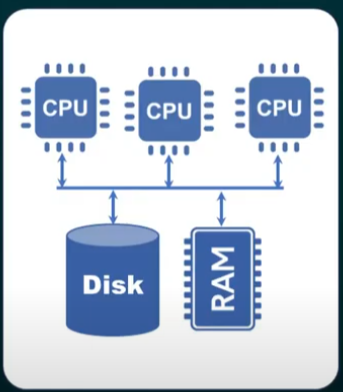
**Nivel 1**

Cloud base data platform offered as a software as a service solution.

* Cloud base 🡪 Born in the cloud / Integrate with more cloud services
* Data platform 🡪 ACID transactions / SQL / Database schema / Structured and semi-structured / Multiple data source.
* Service as a solution 🡪 No infrastructure planning / OS / Software updated

**Common parallel architecture (No Snowflake):**

* Share Disk 🡪 1 disk and multiple CPU processors connected



* Share Nothing 🡪 multiple disks with single processor for each one, but systems connected / Scale multiple but not scaled independently, everything must be scaled (problem)

Diagrama

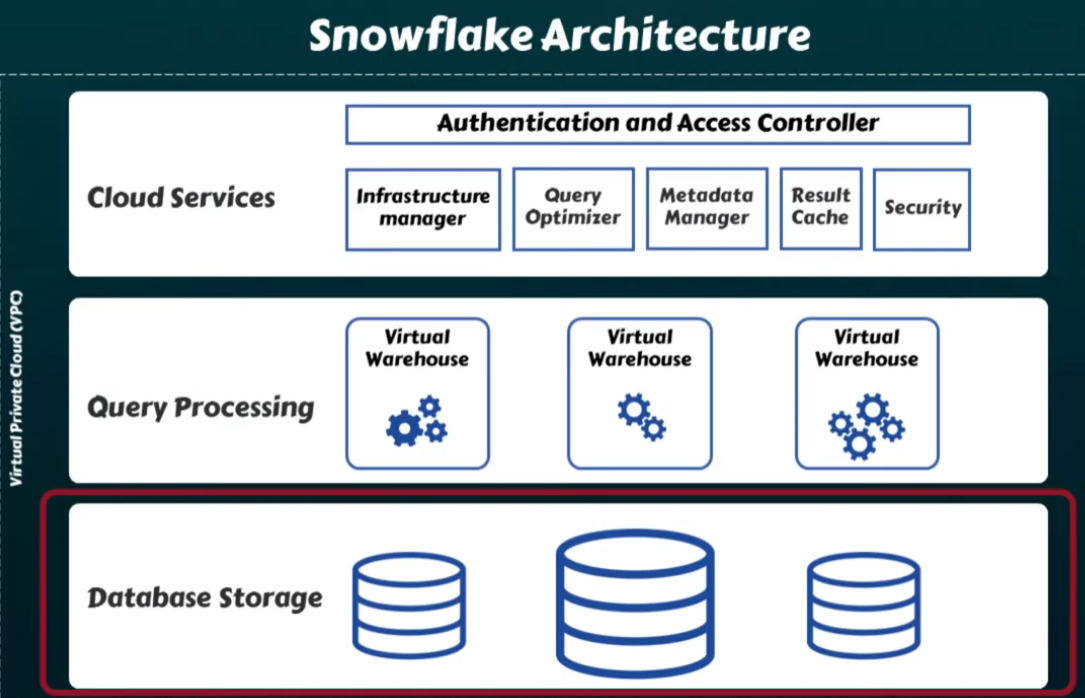
El contenido generado por IA puede ser incorrecto.

**Snowflake architecture**

* Processing layer 🡪 Just have cache memory / can be used as needed
* Storage layer 🡪 Can be scaled / can be scaled as needed

**Layers:**

* Cloud services 🡪 Coordinate activities to process queries
* Query processing 🡪 Only way to access information / Virtual warehouse / Independent
* Database storage 🡪 Automatically optimized in S3 / Partitioned



**Download SnowSQL**

* Connect to your Snowflake account by:

Snowsql -a account\_name -u user\_name

* Se puede conectar automatic a tu cuenta editando el 🡪user 🡪snowsql🡪config

ymqwzzs-uac34472

Waterpolo\_2025

**Permisions and pricing**

* Standard edition 🡪 SQL datawarehouse / Security / 7 days fail safe
* Enterprise 🡪 Standard + Multicluster / Column-row security / Optimization
* Business 🡪 Enterprise + Support / Regulatory support for sensitive information
* Virtual private 🡪 Business + Dedicated computing resources / Dedicated store

**Pricing**

* Storage 🡪 Consumption with flat rate per TB per Month
  + On demand 🡪 Immediate access
  + Commited capacity 🡪 Long term negotiations
* Compute 🡪 Consumed with credits per hour
  + Serveless computing 🡪 Search optimization / Replication / Clustering
  + Virtual warehouse compute
  + Cloud service compute
* Data transfer 🡪 Move data between regions, charged by TB of transported data

**Nivel 2**

* Snowflake uses cloud-based object storage between Amaxon S3, Azure Blob and google cloud.
  + Unilimited capacity
  + High durability

**First Snowflake steps:**

* Create database:

CREATE DATABASE \_\_\_\_\_\_\_\_\_\_\_

* To use a database:

USE DATABASE \_\_\_\_\_\_

* Create the schema: allows organization of objects in a modular way, control access through permissions and separate logic environments, It saves tables, procedures, views, tasks, stages and sequences, each object is named as [DATABASE].[SCHEMA].[ONJECT]

CREATE SCHEMA \_\_\_\_\_\_\_\_

* Define an stage: is a intermediate storage to load data to snowflake or download from it, can be reusable

CREARE STAGE \_\_\_\_\_

* By default there is a personal stage for each user, you can upload files with, by default the file is compresed, to avoid this use AUTO\_COMPRESS=FALSE:

Snowsql -q “PUT <file://file.csv> @~”

* To see the files in the personal stage:

LIST @~

* To upload a file to a general stage from local files:

PUT FILE: //mydata.csv @stage\_\_name

* To upload a file from S3, if the file is compressed just end the file name as -gz:

CREATE STAGE s3\_clientes\_stage

URL = 's3://mi-bucket/datos/'

CREDENTIALS = (

AWS\_KEY\_ID = '<TU\_AWS\_KEY>'

AWS\_SECRET\_KEY = '<TU\_AWS\_SECRET>'

)

FILE\_FORMAT = (

TYPE = 'CSV'

FIELD\_DELIMITER = ','

SKIP\_HEADER = 1

);

* Copy the info to a table:

COPY INTO Table

FROM @stage\_name/file

**Concept of the process to upload files to storage layer:**

* The file will be transformed in the uploading process
  + First is going to be converted into Column format
  + Data is partitioned in chunks
  + Each partition is compressed
* Snowflake partitions automatically
* Uses micropartitions 50-500MB
* Saves metadata about each partition to improve the query process
* Partitions are immutable, when there is an update, it creates a new partition.

**Processing layer:**

* 1-3 Virtual warehouses: each warehouse has two or more nodes, each node has CPU+RAM+Cache memory.
* Each node compute a single query
* Warehouses operates: SQL SELECT / DML Operations / Loading – Uploading data
* Local cache performed a faster retrieve in the processing layer
* Metadata cache helps for specific queries in cloud layer
* We can set within the warehouse: Number of clusters, if is suspended, if is on.

**Cloud services:**

* Tie all the snowflake resources
* Authentication 🡪 Permissions
* Infrastructure manager 🡪 Storage and Compute
* Metadata manager 🡪 Main information about tables and objects, partitions and all
* Security manager 🡪 IP control / IAM manager / Deta encryption
* Transaction manager 🡪 Visible transactions / Locking transactions statement (INSERT, DELETE, MERGE) / No-lock (INSERT, COPY, SELECT) to avoid inconsistency
* Query optimizer 🡪 Very fast and optimized by using metadata cache and results cache.

**Nivel 3**

**Stages:** The COPY command works with SQL operations: Reordering, Columns omission, Type cast, Truncate text columns.

* **Internal 🡪** Assosiated to the snowflake account / within snowflake
  + **User stages** 🡪 Files by single user
    - Automatically created with the user
    - @~ / just the user can access
    - PUT to upload
    - COPY INTO table\_name
  + **Table stages 🡪** Many users and single table
    - Automatically created with the table
    - @%table\_name
    - OWNER of the table can grant access
    - PUT to upload
    - COPY INTO table\_name
  + **Named stages 🡪** Many users and many tables
    - when using CREATE STAGE DDI
    - @int\_stage\_name
    - PUT to upload
    - COPY INTO table\_name
* **External 🡪** Assosiated to the company storage service / outside snowflake / @ext\_stage\_name
  + Multiple tables and users
  + No storage charges
  + Can NOT use PUT to upload, needs cloud service provider API
  + COPY INTO table\_name

**Subir archivos desde el disco local / AWS:**

* Especificar el rol:

USE ROLE NAME

* Especificar warehouse:

USE WAREHOUSE NAME

* Crear la tabla si aún no está creada

CREATE TABLE IF NOT EXISTS TABLA ()

* Verificar que hay en cada una de las stages, sean personales o de tablas

ls @~

ls @%TABLA

* Si es necesario crear una stage interna

CREATE OR REPLACE STAGE internal\_stage\_name

ls@internal\_stage\_name

* Para subir archivos mediante stage externa
  + Option 1: créate IAM user / create access key / generate access for application outside AWS

CREATE OR REPLACE STAGE external\_stage\_name

URL=’S3://bucket/’

CREDENTIALS=(AWS\_KEY\_ID=’\_\_\_\_’ AWS\_SECRET\_KEY=’\_\_\_\_\_\_\_\_’)

Para el ejemplo:

KEY\_ID= AKIA4QKUKVXAZLIGPIMR

AWS\_SECRET\_KEY= Dmy0QjeB4kTBspnyRdSQFVVlmWN4z9LTZW6/VfCu

* + Option 2: create IAM role in AWS account / 000 for external ID field / assign permissions / copy the ARN and use as:

CREATE STORAGE INTEGRATION s3\_integration

TYPE=EXTERNAL\_STAGE

STORAGE\_PROVIDER=’S3’

STORAGE\_AWS\_ROLE\_ARN=’ARN’

ENABLED=TRUE

STORAGE\_ALLOWED\_LOCATIONS=(‘s3://s3\_bucket/’)

* + - Then snowflake creates an IAM user, then in roles, navegate in relationships and put the info from snowflake and then you can create the stage

CREATE STAGE external\_stage\_name

URL=’’

STORAGE\_INTEGRATION=s3\_integration

* Para eliminar:

DROP STAGE external\_stage

* Para cargar los datos desde el stage para la tabla se usa con archivos internos

PUT ‘file://C:/Users/mamba/Desktop/customer.csv’ @~

* Luego para cargarlo a la tabla se usa

COPY INTO Table\_name

FROM @~

FILE\_FORMAT=(TYPE=CSV,FIELD\_OPTIONALLY\_ENCLOSED\_BY=’”’,SKIP\_HEADER=1)

PURGE=TRUE

* El file format puede setearse desde antes así:

CREATE OR REPLACE FILE FORMAR csv

TYPE=’CSV’

FIELD\_OPTIONALLY\_ENCLOSED\_BY=’”’

SKIP\_HEADER=1

* Y solo se referencia al subir el archivo

COPY INTO table

FROM @~

FILE\_FORMAT=(FORMAT\_NAME=csv)

* Los archivos se pueden eliminar solos con el PURGE una vez se suben o con este comando:

REMOVE @~

**Bajar archivos de snowflake**

* Copiar los archivos de las tablas a los stages, se puede hacer como json o mas formatos

COPY INTO @~/csv\_export/

FROM table

FILE\_FORMAT=(TYPE=’csv’,COMPRESSION=’GZIP’)

* Solo se descarga el archivo del stage

GET @~/csv\_export ‘file://C:user/directory/’

**Nivel 4**

**Table types:**

* Permanent
  + Recover the table data as existed before 90 days
  + Failure safe 7 days
  + Persist until drop
* Temporary
  + Only during the session
  + Preparing or intermediate processes, 1 day time travel
  + No safe period
* Transient
  + Persist until drop
  + No safe period
  + Time travel 1 day

**Table types creation:**

* Create tables

CREATE TRANSIENT TABLE table\_name

CREATE TEMPORARY TABLE table\_name

CREATE TABLE table\_name (para las permanentes)

* Como Volver en el tiempo una tabla (la fecha tiene que tener formato timestamp)

SELECT \* FROM table\_name AT(TIMESTAMP=>’’::timestamp\_tz)

* Configurar la duracion de recuperación

ALTER TABLE table\_name SET DATA\_RETENTION\_TIME\_IN\_DAYS=90

* Como ver detalles de las tablas, usar

SHOW TABLES LIKE ‘%’ 🡪 Todas las tablas

SELECR “name”, “database\_name”, “kind”,”is\_external”,”is\_hybrid”

FROM TABLE (result\_scan(last\_query\_id()))

**Additional table types:**

* **External tables 🡪** Query in external stage, just read / slower
* **Iceberg tables 🡪** Available for multiple formats
* **Dynamic tables 🡪** Replacement of ETL pipelines
* **Hybrid tables 🡪** To optimize updates and OLTP write
* **Event tables 🡪** For store procedures
* **Directory tables 🡪** Metadata tables

**Nivel 5**

**Snowflake views:** Objects that can be accessed as tables no (Insert, Update, Drop)

* Can restrict sensitive information from a table
* Reduce complexity to end users

**Types of views:**

* **Non materialized views 🡪** No additional cost but commutated each time

CREATE OR REPLACE VIEW employee AS

SELECT nombre, edad

FROM employees

* **Materialized view 🡪** Precomputed result from a query, updated each time, faster and additional cost usually over external tables, for data not updates frecuently but with many queries

CREATE MATERIALIZED VIEW salary AS

SELECT \*

FROM employees;

* **Secure view 🡪** People can access to the original table from the other views with the DDL

CREATE SECURE VIEW employee AS