Snowflake

* Snowflake **enables data storage, processing, and analytic solutions** that are faster and easy to use.
* The Snowflake architecture allows storage and compute to scale independently, so customers can use and pay for storage and computation separately.
* Sharing functionality makes it easy for organizations to quickly share governed and secure data in real time.

Snowflake decouples the storage and compute functions, which means organizations that have high storage demands but less need for CPU cycles, or vice versa, don’t have to pay for an integrated bundle that requires them to pay for both. Users can scale up or down as needed and pay for only the resources they use. Storage is billed by terabytes stored per month, and computation is billed on a per-second basis.

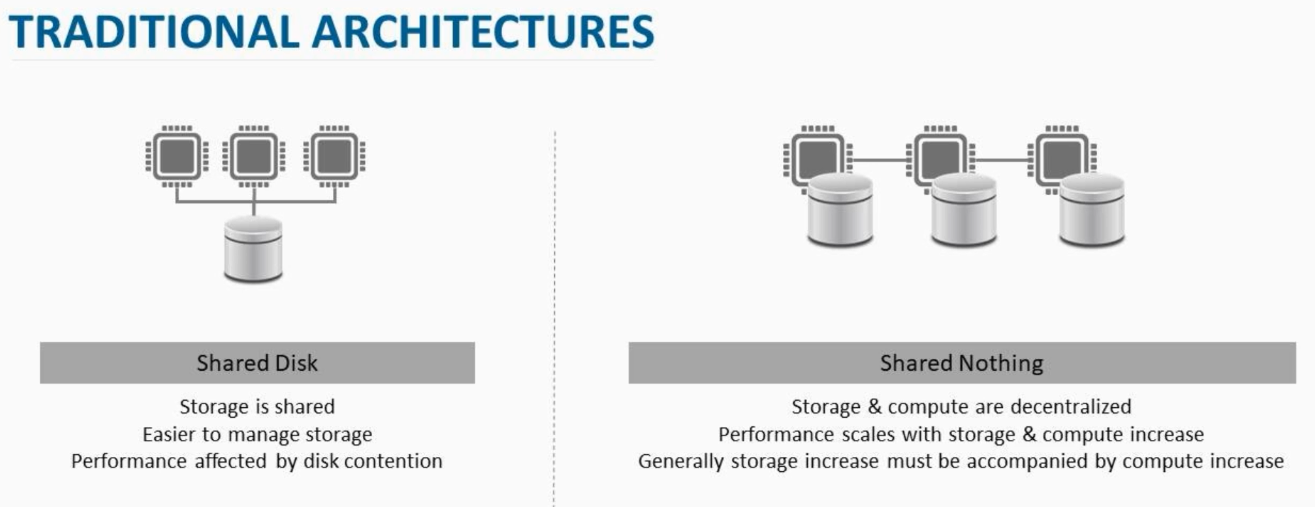


Benefits

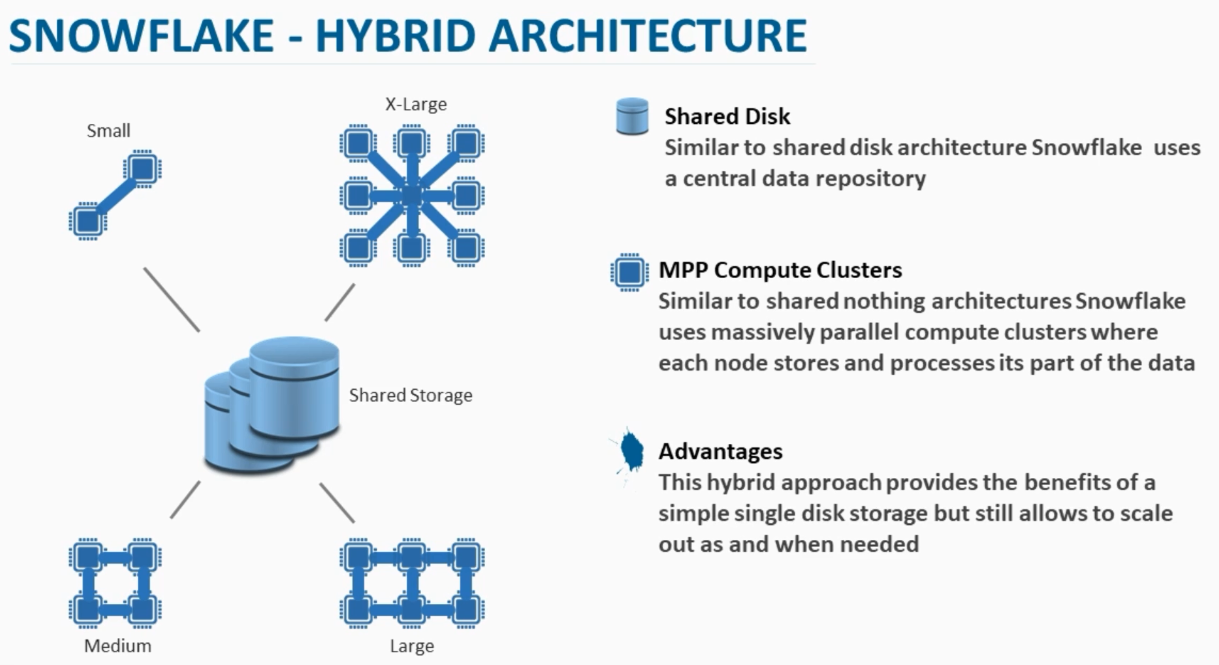
* Performance and speed
* Storage and support for structured and semi-structured data
* Concurrency and accessibility
* Seamless data sharing
* Availability and security

**Snowflake Architecture**

Most databases followed the traditional architectures as shown in the picture.



Snowflake follows a hybrid architecture where it uses parallel compute clusters while keeping a central data repository. It provides an option to scale out as required by business.



Snowflake architecture consists of three layers, each of which is independently scalable.

* **Database storage**

Snowflake uses cloud platform storage. It internally stores data in columnar format which provides efficiency, compression and performance.The database storage layer holds all data loaded into Snowflake, including structured and semi-structured data. Snowflake automatically manages all aspects of how the data is stored: organization, file size, structure, compression, metadata, and statistics. This storage layer runs independently of compute resources.

* **Compute layer**

The compute layer is made up of virtual warehouses that execute data processing tasks required for queries. Each virtual warehouse (or cluster) can access all the data in the storage layer, work independently, so the warehouses do not share, or compete for, compute resources. This enables non-disruptive, automatic scaling, which means that while queries are running, compute resources can scale without the need to redistribute or rebalance the data in the storage layer.

No data is stored within compute nodes. It just performs the tasks and returns the results.

* **Cloud services**

The cloud services layer uses ANSI SQL and coordinates the entire system. It eliminates the need for manual data warehouse management and tuning. Services in this layer include:

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

The virtual warehouse can be turned off at any moment when not in use which means you pay only for storage of data and not compute. It also provides the option to create different warehouses of different sizes which can be used for different purposes but all work on single data store.

Data can be used on cloud like an S3 bucket or Azure blob storage and can be loaded into snowflake using any ETL tool or using bulk load method (Copy command).

**Connecting to Snowflake**

Snowflake supports multiple ways of connecting to the service:

* A web-based user interface from which all aspects of managing and using Snowflake can be accessed.
* Command line clients (e.g. SnowSQL) which can also access all aspects of managing and using Snowflake.
* ODBC and JDBC drivers that can be used by other applications (e.g. Tableau) to connect to Snowflake.
* Native connectors (e.g. Python, Spark) that can be used to develop applications for connecting to Snowflake.
* Third-party connectors that can be used to connect applications such as ETL tools (e.g. Informatica) and BI tools (e.g. ThoughtSpot) to Snowflake.

**Types of tables Supported**

* **Permanent tables**- Default type of table. If you don’t specify the type, the table is created as permanent
* **Temporary tables**- Temporary tables only exist within the session in which they were created and persist only for the remainder of the session. Once the session ends, data stored in the table is purged completely from the system and, therefore, is not recoverable.
* **Transient tables**- Transient tables 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. As a result, transient tables are specifically designed for transitory data that needs to be maintained beyond each session but does not need the same level of data protection and recovery provided by permanent tables.
* **External tables-**In an external table, the data is stored in files in an external stage. External tables store file-level metadata about the data files, such as the filename, a version identifier and related properties. This enables querying data stored in files in an external stage as if it were inside a database. External tables can access data stored in any format supported by [COPY INTO <table>](https://docs.snowflake.com/en/sql-reference/sql/copy-into-table.html) statements. External tables are read-only, therefore no DML operations can be performed but can be used for query and join operations.

**Data Loading Options**

* **Bulk Load**
  + Copy command for batch loading which uses VM’s compute resources
  + Data should be available in cloud storage or an internal location
  + Allows basic transformations
* **Continuous Load**
  + Snowpipe is used to load streaming data
  + Uses a serverless approach like scaling up or down automatically
  + Doesn’t use VM’s compute resources

For loading the data into snowflake, we need to stage the data into S3 bucket or azure blob storage. From there we can use copy command to load data into snowflake table.

**Bulk Load Option Sample Code**

--Create a new database

CREATE DATABASE NEW\_DATABASE;

--Create a new table

CREATE TABLE NEW\_TABLE (

first\_name STRING ,

last\_name STRING ,

address string ,

city string ,

state string

);

--sample data for the table above is present in AWS S3 at

https://s3.ap-southeast-2.amazonaws.com/snowflake-essentials/our\_first\_table\_data.csv

create or replace stage my\_s3\_stage url='s3://snowflake-essentials/';

LIST @my\_s3\_stage;

copy into NEW\_TABLE

from s3://snowflake-essentials/our\_first\_table\_data.csv

file\_format = (type = csv field\_delimiter = '|' skip\_header = 1);

**or**

copy into NEW\_TABLE

from @my\_s3\_stage/our\_first\_table\_data.csv

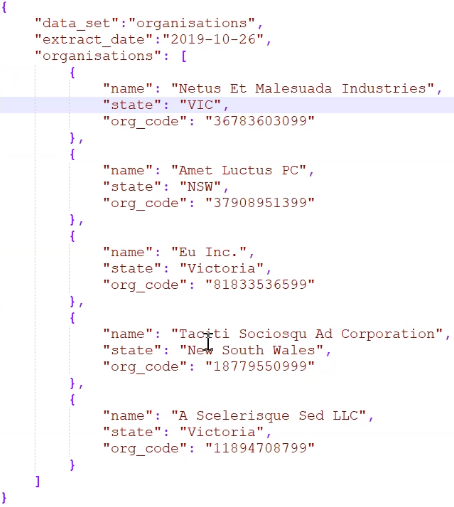
file\_format = (type = csv field\_delimiter = '|' skip\_header = 1);

SELECT \* FROM NEW \_TABLE;

SELECT COUNT(\*) FROM NEW \_TABLE;

**Loading JSON Data**

Suppose a JSON file is available as mentioned in the screenshot, then it can be loaded as mentioned below.

****

-- create a table in which we will load the raw JSON data

CREATE TABLE organisations\_json\_raw (

json\_data\_raw VARIANT

);

-- create an external stage using the S3 bucket that contains JSON data

CREATE OR REPLACE STAGE json\_example\_stage url='s3://snowflake-essentials/json\_data';

-- list the files in the bucket. It is just to check if the file is visible or not.

LIST @json\_example\_stage;

-- copy the example\_json\_file.json into the raw table

COPY INTO organisations\_json\_raw

FROM @json\_example\_stage/example\_json\_file.json

file\_format = (type = json);

-- validate that the JSON has been loaded into the raw table. All the data gets into a single row.

SELECT \* FROM organisations\_json\_raw;



-- use flatten table function to convert the JSON data into column and insert it into a table

SELECT

value:name::String,

value:state::String,

value:org\_code::String,

json\_data\_raw:extract\_date

FROM

organisations\_json\_raw

, lateral flatten( input => json\_data\_raw:organisations );



**Scaling Options**

Snowflake supports two ways to scale warehouses:

* Scale up by resizing a warehouse.
* Scale out by adding warehouses to a multi-cluster warehouse (requires Snowflake Enterprise Edition or higher).

**Caching**

* Caching is automatic in snowflake
* If you use same query multiple times, results get cached and data retrieval is faster
* Snowflake knows when the underlying data has changed therefore re-executes the query
* To maximize cache usage, similar queries should go to the same VM.

**Clustering**

* Snowflake produces well-clustered data in tables however over time the data in some table rows might no longer cluster optimally on desired dimensions.
* A clustering key is a subset of columns in a table that are explicitly designated to co-locate the data in the table in the same [micro-partitions](https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions.html).
* Improves scan efficiency in queries by skipping data that does not match filtering predicates

**Snowflake Emergency Data Retrieval Options**

* **Time Travel**

Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It serves as a powerful tool for performing the following tasks:

* + Restoring data-related objects (tables, schemas, and databases) that might have been accidentally or intentionally deleted.
  + Duplicating and backing up data from key points in the past.
  + Analyzing data usage/manipulation over specified periods of time.

AT | BEFORE Clause

The AT or BEFORE clause is used for Snowflake Time Travel. In a query, it is specified in the [FROM](https://docs.snowflake.com/en/sql-reference/constructs/from.html) clause immediately after the table name and it determines the point in the past from which historical data is requested for the object:

* + The AT keyword specifies that the request is inclusive of any changes made by a statement or transaction with timestamp equal to the specified parameter.
  + The BEFORE keyword specifies that the request refers to a point immediately preceding the specified parameter.

Example:

-- time travel to a time just before the update was run

select \* from CUSTOMER before(timestamp => '2019-05-26 03:20:15.855'::timestamp);

-- time travel to 10 minutes ago (i.e. before we ran the update)

select \* from CUSTOMER AT(offset => -60\*10);

-- time travel to the time before the update query was run

select \* from CUSTOMER before(statement => '018c6f1f-00fd-b06c-0000-00000e99c991');

* **UNDROP**
  + UNDROP restores the most recent version of a dropped table
  + In addition to UNDROP TABLE, UNDROP SCHEMA and UNDROP DATABASE commands are also available.
  + Option available to purchase even longer retention periods if needed

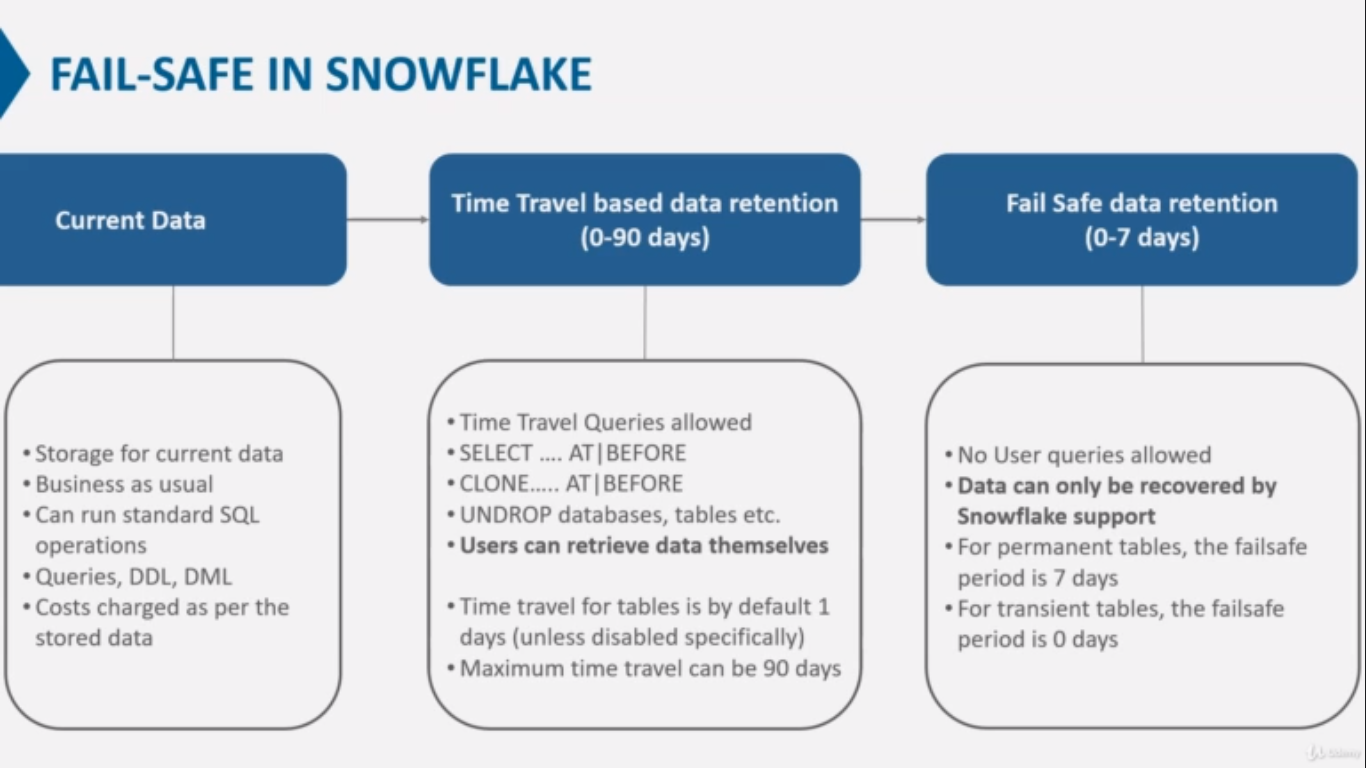
Syntax:

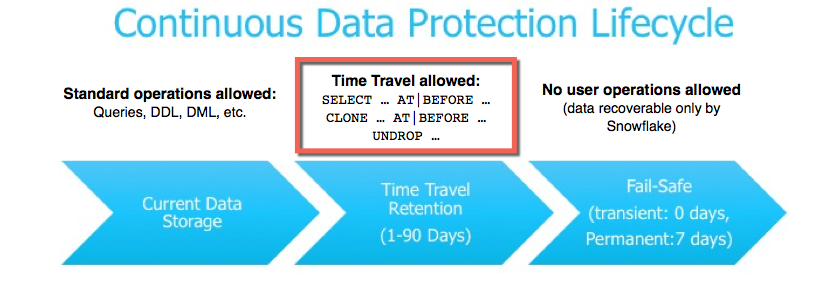
UNDROP TABLE <tablename>

UNDROP SCHEMA <schema name>

UNDROP DATABASE <db name>

* **Fail-Safe**
  + Fail-safe provides a (non-configurable) 7-day period during which historical data may be recoverable by Snowflake
  + This period starts immediately after the Time Travel retention period ends





**Zero copy clones**

* Clone lets you create copy of your tables, schemas, databases without copying the actual data.
* Clone utilizes no data storage because it shares all the existing micro-partitions of the original table at the time it was cloned, however rows can then be added, deleted, or updated in the clone independently from the original table.
* Each change to the clone results in new micro-partitions that are owned exclusively by the clone and are protected through CDP.

References

https://www.snowflake.com/

<https://www.stitchdata.com/resources/snowflake/>

https://docs.snowflake.com/en/user-guide/