



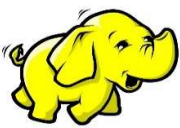

SNOWFLAKE

- I. Unit 1: Introduction
 - a. [Introduction](#)
 - b. [Self registration](#)
 - c. [Snowflake Architecture](#)
 - d. [Clustering](#)
 - e. [Virtual Warehouse](#)
 - f. [Performance Tuning](#)
 - g. [Snowsight](#)
 - h. [Query Acceleration](#)
 - i. [Search Optimization](#)
 - j. [Optimization - Outlook](#)
- II. Unit 2 : Loading and unloading data
 - a. [Data load/unload](#)
 - b. [Copy options](#)
 - c. [Loading Unstructured Data](#)
- III. Unit 3: Snowflake features
 - a. [Snowpipe](#)
 - b. [Data sharing](#)
 - c. [Time travel](#)
 - d. [Retention period](#)
 - e. [Fail safe](#)
 - f. [Clone](#)
 - g. [Sampling](#)
 - h. [Materialized Views](#)
 - i. [Dynamic Tables](#)
- IV. Unit 4: Snowflake Data Governance and Administration
 - a. [Data Masking](#)
 - b. [Row access policy.](#)
 - c. Tagging
 - d. Stored procedures
 - e. Access control
 - f. Snowflake ORG account
 - g. Disaster recovery and replication
 - h. Best practices
 - i. Snowpark

Note: This document is only for revision, to get the detailed view refer to the documentation.

Chapter 1 : Introduction

1.1. Evolution of Cloud Data Platform

				
	On premise data	1 st Gen Cloud EDW	Data Lake, Hadoop	Cloud platform data
All Data	X	X	X	✓
All Users	X	X	-	✓
Fast answers	✓	✓	✓	✓
SQL Databases	✓	✓	✓	✓

1.1.1. 1st Evolution

- ⇒ Traditional On – premise Enterprise Data warehouse.
- ⇒ They were able to answer the business questions (queries) very fast.
- ⇒ They were all SQL Databases.
- ⇒ Challenges:
 - Not designed for all kind of data – structured & unstructured.
 - Not designed for all the users who may have demand for that data.
 - Not scalable.

1.1.2. 2nd Evolution

- ⇒ 1st Generation Cloud EDW
- ⇒ They moved the same code into the public cloud.
- ⇒ They fast answer to queries & SQL Database but inherent problem still existed like:
 - Unable to onboard all type of data (structured and unstructured).
 - Access to all your different users was still unavailable.

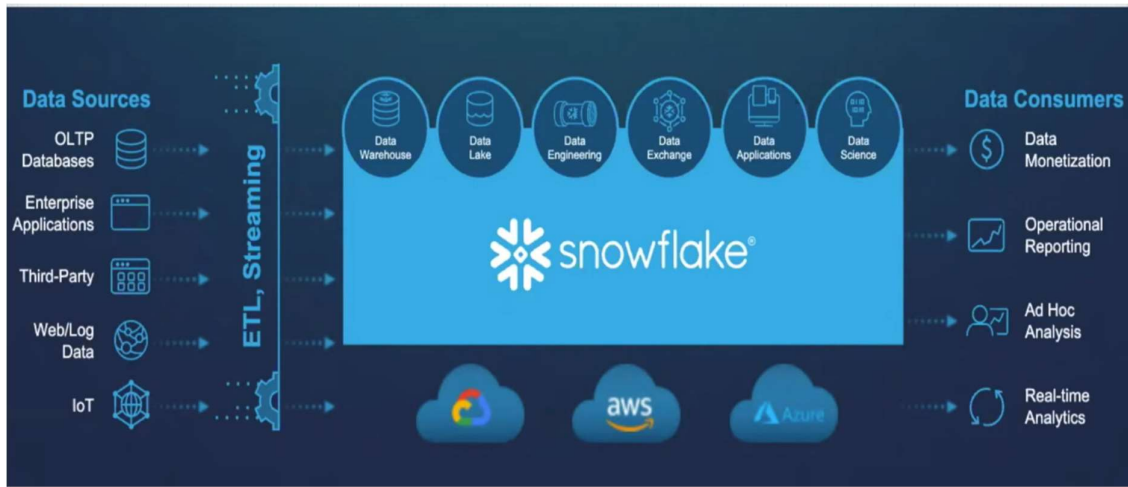
1.1.3. 3rd Evolution

- ⇒ Hadoop / Data Lake
- ⇒ You can load any kind of data.
- ⇒ A lot of users who have a demand for the data can access it to a certain extent.
- ⇒ Slow and no native support to SQL.
- ⇒ Performance & scalability is challenging (operational/manageability).

1.1.4. 4th Evolution – Cloud Data Platform

- ⇒ All users, all data, fast answer, SQL – all features are in a single platform.
- ⇒ Natively ingest structure/semi – structured data.
- ⇒ Concurrent access to as many users within organisation.
- ⇒ Standard SQL DB, hence easy to manage.

1.2. Modern Data Architecture with Snowflake



1.2.1. Build for the cloud

- ⇒ Build from scratch (cloud – native), optimise for cloud (AWS/Azure/GCP), storage & compute is decoupled.
- ⇒ All kind of use case are supported (Data warehouse, Data Lake, Data Engineering, Streaming, Data Application, Data Science, Data Exchange).
- ⇒ Share data to internal as well as external customers securely (regulator/partner, etc.) without data leaving your premise.

1.2.2. Software as a Service (SaaS)

- ⇒ No software, infrastructure, or upgrades to manage.
- ⇒ Any update, maintenance, releases are done by the snowflake, and it is available to all customers automatically.
- ⇒ In all the 3 major cloud provides.

1.2.3. Pay as you go

- ⇒ Storage and compute charged independently and only for usage (an issue in Hadoop).
- ⇒ If we store TBs of data and no processing is performed, we are charged only for the storage and not computing.

1.2.4. Scalable

- ⇒ Virtual warehouses enable computing scaling independently form storage.
- ⇒ The scalability is virtually unlimited and storage and compute both can scale as needed and independent from each other.

1.3. The Value of a Cloud Data Platform

- ⇒ Unlimited performance and scale
- ⇒ Near – zero maintenance, as a service.
- ⇒ One platform, one copy of data, many workloads.
- ⇒ Secure governed access to all data.
- **Snowflake supports all these types of use cases. Hence is it a true cloud data platform.**

Chapter 2 : Self registration

- ⇒ Simply go to [Snowflake Trial](#) and register yourself on the snowflake website (Choose Enterprise edition + AWS).
- ⇒ Snowflake supports 3 cloud service providers to host account:
 - AWS
 - Azure
 - GCP
- ⇒ Please make sure to choose the same region where the data is to be fetched from. E.g. If the data is in US East (Northern Virginia) in AWS S3 bucket then choose the same region in Snowflake to reduce extra cost.
 - S3 is a blob store with a relatively simple HTTP(S) – based PUT/GET/DELETE interface.
 - Objects i.e. files can only be over – written in full. It is not even possible to append data to the end of a file.
 - S3 does, however, support GET request for parts (ranges) of file.
- ⇒ When I host snowflake account on AWS it will only use AWS services to store data and utilize AWS compute resources.
- ⇒ When you create a database, two default schemas are created:
 - i. Public
 - ii. Information schema
- ⇒ Every query executed in Snowflake has its own query id.
- ⇒ Database consists of:
 - Tables
 - Views
 - Schema
 - Stages
 - Pipes
 - Sequences
 - File format

Chapter 3 : Snowflake architecture

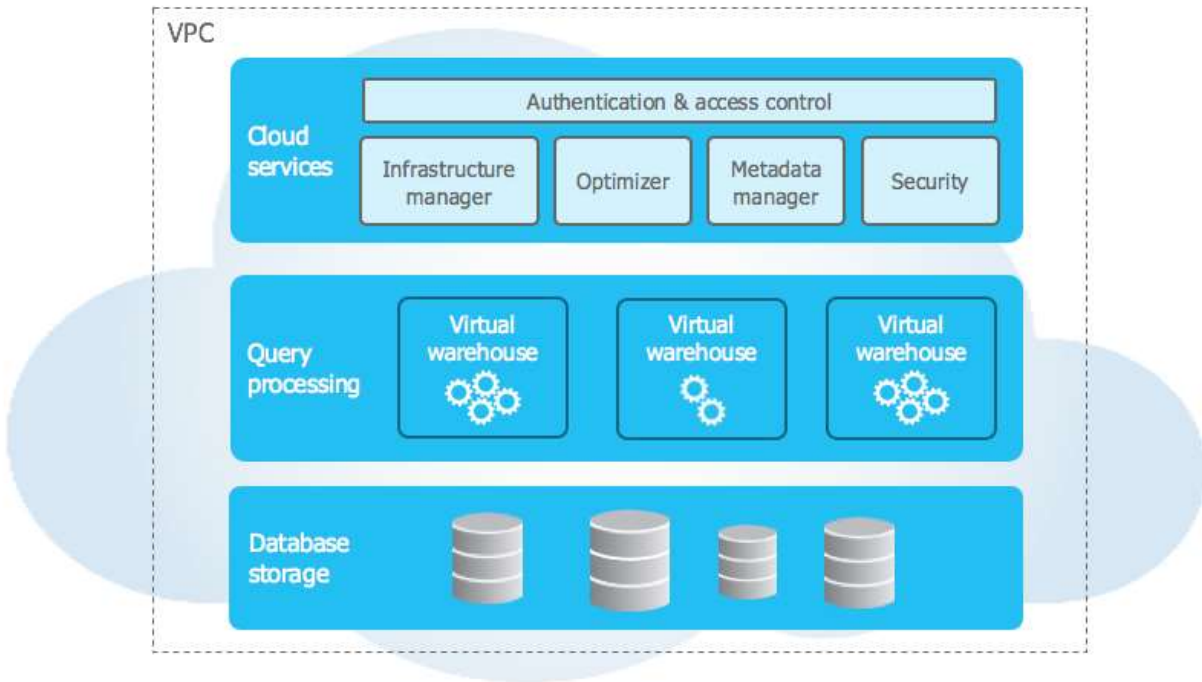
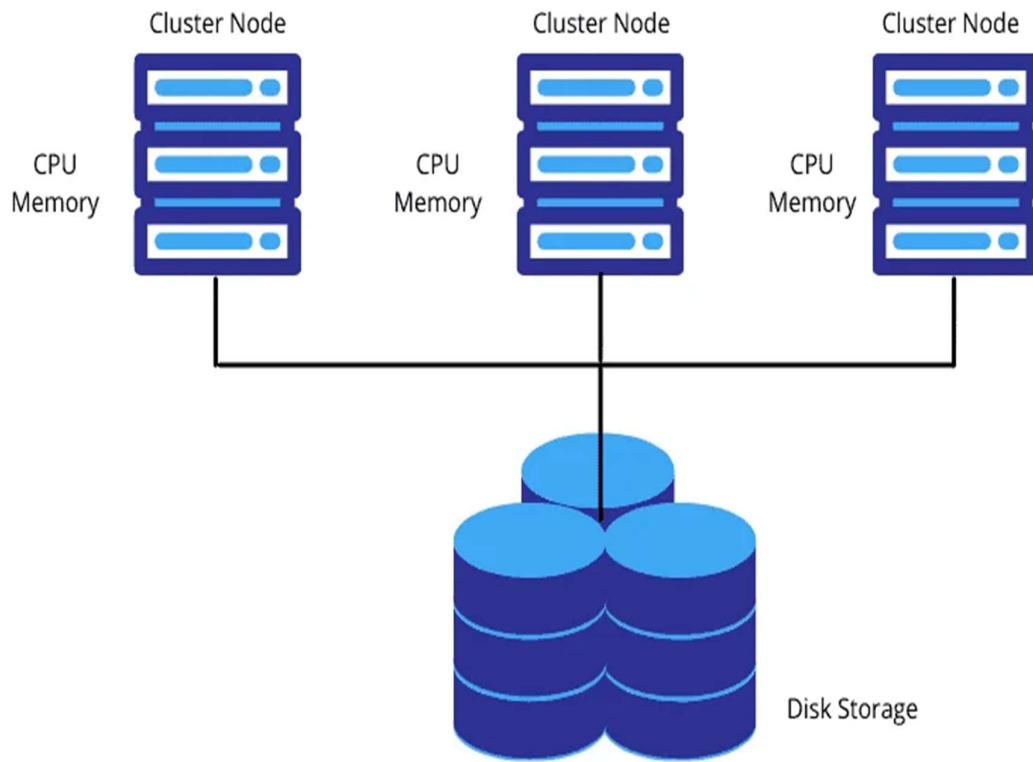


Figure 1. Multi-cluster, shared data architecture

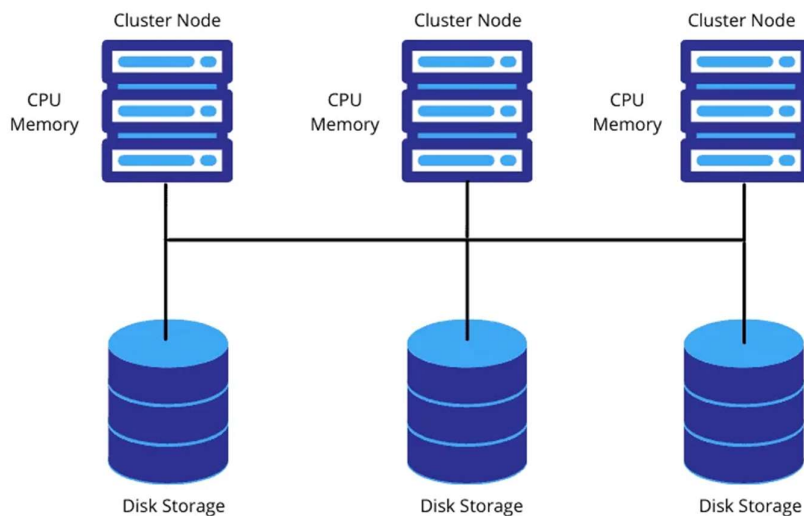
Note: Snow SQL is case-sensitive in case of S3 bucket files.

1. Data Storage Layer : This layer stores all table data and query results (AWS, Azure or GCP).
 2. Query processing : Handles query execution within elastic clusters of virtual machines. This is also called as **"muscle of the system"**.
 3. Cloud Services Layer : It is a collection of services used to manage virtual warehouses, queries we execute on Snowflake, transactions and metadata information. It provides access control, usage statistics, and query optimization and other services. Hence, it is called as **"brain of the system"**.
- For every query we execute, we'll be charged, hence we need to understand the architecture which helps us to save cost.
 - Let's understand other architectures :
 - Shared disk architecture : In this architecture, we'll have one shared disk which will be accessed by multiple database nodes. Every database node will try to read and write data on this shared disk.



○ Limitations:

- Scalability : With increase in number of nodes, it will become difficult to read and write data, as shared disk will become a bottleneck.
 - Hard to maintain data consistency across the cluster : This becomes an issue if two or more nodes are trying to read/write data at the same time.
 - Bottleneck of communication with shared disk.
- Shared Nothing architecture : To solve issues found in 'shared disk architecture', in this architecture, we assign storage for every compute node.



○ Advantages:

- It scales processing and compute together.
- Eliminates bottleneck issue.
- It moves data storage close to compute.

- Disadvantages:
 - With increase in data volumes, no. of nodes will increase. Each node is only responsible for the data it has stored on the disk. If a node wants data from a different node, data should get shuffled b/w the nodes.
 - What if a node fails? Data should get shuffled to other nodes. Such failures will impact the performance of the whole system. Performance is heavily dependent on how data is distributed across the nodes in the system.
 - Choosing right balance b/w storage and compute is difficult. If you wish to increase storage without adding compute, you can't do that as **data and compute are tightly coupled together** in this architecture. Compute can't be sized independently of storage.
 - Heterogeneous workload and homogenous hardware : During loading data activity, we will be doing "*bulk loading*", such process is only intended to copy data quickly inside the system. During such process **we will not be utilizing heavy compute, but we will be using high bandwidth**. In some other cases, we will be cleansing data, we do aggregation operations and other transformation activity on the data, such process requires **heavy compute and low I/O bandwidth**. In share nothing architecture, hardware is homogenous, hence we can't change/configure our hardware based on the activity we are performing. Hence, configuration needs to be a trade off with low average utilization.
 - Membership changes : If a node fails or user chooses to resize the system, larger amounts of data need to be reshuffled. If we want to increase storage size or process efficiency, change has to be made to the larger number of nodes. This limits elasticity and availability.
 - Upgrades : What if we want to upgrade software in every node ? If system hardware is heterogeneous, an upgrade in one node may not work in another. We expect all nodes to be homogenous and tightly coupled (i.e. identical hardware).
- Multi-cluster shared data architecture (Snowflake) : To solve the above problems, Snowflake separated data storage layer and compute layers. Now as they both are independent of each other (Query processing/ Compute layer, Storage Layer), both can grow and shrink according to the activity needs.
 - But the problem here is that how do we coordinate b/w these two layers? How data compute layer should know which data to process ? how data storage should know which data to return ?
 - To solve these problems, Snowflake created another layer, i.e. "Cloud Services Layer". This layer handles many things like authentication and access control, metadata storage, query optimization, security, etc.
 - So, how does the *Storage Layer* work?
 - Snowflake stores all data into databases and the database is a logical grouping of objects within a snowflake instance.
 - These objects are primarily Tables (Permanent, Temporary, Transient) and Views (Standard and Materialised).
 - These objects are part of one or more schemas.
 - You can store any structured relational data (standard SQL data types) or semi structured non-relational data (JSON Parquet, Avro, ORC XML) (variant data types).
 - Snowflake uses highly secure cloud storage to maintain your structured and semi structured data.
 - As data is loaded into the table following activities happen:
 - Snowflake convert them into optimised columnar compressed format(proprietary to snowflake).
 - This optimised columnar compressed format brings a lot of data access efficiency (faster workload, low compute and storage cost).
 - Add Encrypt AES 256 Strong Encryption.
 - Based on the cloud platform data is loaded to the cloud storage (S3/Azure Blob/GCP Bucket), however, this is not visible to the user how it is stored and retrieved and overhead is taken care of by the snowflake.

- These compressed and secure data is accessible only via SQL and there are no other means that it is accessible.
 - Data storage cost is calculated on the daily average amount of data (in bytes) (short lived or long lived tables).
 - If the time travel feature is enabled it is also part of the data storage cost.
- Snowflake data storage costs are calculated based on **compressed size and amount stored – daily average**.
- And how does *Compute Layer* work?
 - Compute layer is where queries (select queries, join queries, data loading, stored procedure, etc.) close are executed.
 - Before any query is executed, compute machines need to be provisioned and in Snowflake, they are called virtual warehouse (VWH).
 - This virtual warehouse (compute resource) has access to same data storage or data layer.
 - You can choose a virtual warehouse as per the workload required without any contention or performance degradation.
 - To create a virtual warehouse, you simply give a name and size (bigger the size more the computer power) and Snowflake handles all the provisioning and configuration of the underlying computer resources (in the case of AWS, that is EC2 instance and for Azure it is Azure VM).
 - The virtual warehouse can be scaled up and down at any time during the query execution without any hiccups. When rescaling is done, all the subsequent queries take the advantage of new size of the warehouse.
 - One multiple virtual warehouses (of different sizes) are running in parallel, Snowflake takes care of concurrency.
- Let's explore how *Cloud Service Layer* works:
 - It manages and coordinates the entire system.
 - This layer ensures and takes care of:
 - Authentication and authorization (via WebUI or Web connector or SnowSQL or native connector or MFA etc).
 - User and session management.
 - Query compilation, optimization and data caching.
 - Virtual warehouse management, coordinate data storage / updates and transaction management.
 - Metadata management (one of the core activity) and this feature supports:
 - Zero copy cloning.
 - Time travel
 - Data sharing
 - Caching.
 - Management maintain the life cycle of a query.
 - The service layer is highly scalable and distributed across multiple Availability Zones.
- [Snowflake caching :](#)
 - When we first run a query, it first reaches "*Cloud services layer* " where any optimizations needed are checked. Then it goes to the "*Compute layer* " where the query is processed and then finally it reaches the last layer i.e. "*Storage Layer* " form where data is processed.
 - But if we run the same query second time, it won't reach the third layer, instead it will directly return the result from the "*caching area*" of the "*Cloud services layer*" (not virtual warehouse).
 - The above two statements can be confirmed by checking the activity of the warehouse, in the first case it will get active while in the second it won't.
 - We can also use the "*caching area*" of the virtual warehouse by altering the session cache by this query " ALTER SESSION SET use_cache_result = false; ". This will stop using the caching area of the "

cloud services layer “ and start using the caching area of the virtual warehouse layer. NOTE : If the VWH is set to auto suspend it will purge the cache memory upon hitting the limit of suspension.

- Lesson learnt:
 - Always use limit clause while using ‘ *Select ** ’ queries in Snowflake to avoid unnecessary wastage of credits.
 - During development activity, always keep auto suspend to a higher time limit, at least 15 minutes.
 - Share your virtual warehouse when a group of users are working on the common tables to use shared caching area.
 - **Never disable your cloud services result cache.**
 - Reusing query result is free in Snowflake.
- If there are any changes in the table, Snowflake will not use the cache memory to show the result.
- This query will be cached (result cache) 24 hours from the time of last execution. If we execute the query on the 23rd hour, it will again get cached for the next 24 hours.
- If two users use the same query, Snowflake will intelligently identify it and show the same result without processing the query once again. Snowflake leverages cached result across multiple users.
- Snowflake will not charge to store the cache result.
- Following are Snowflake cache layers:
 - Result cache – belongs to service layer. Can be disabled.
 - Local disk cache – belongs to compute layer (VWH). Can’t be disabled.
 - Remote cache – blob storage area like AWS S3. Can’t be disabled.
- If there are manipulations in the query where we need to fetch a subset of the query result (stored in Cloud service layer cache), then it is fetched from the cache area of local disk i.e. virtual warehouse cache area.

Chapter 4 : Clustering

- ⇒ The process of grouping micro partitions is called '**CLUSTERING**'.
- ⇒ We can also provide clustering key in the query to customize clustering according to our needs:
 - E.g.: CREATE TABLE EMPLOYEE (type, name, country, date) CLUSTER BY (date);
 - In the above query the clustering will be performed according to the date.
- ⇒ What are micro partitions?
 - When you load the files in Snowflake table, tables are horizontally partitioned into large immutable files, which are equivalent to blocks or pages in a traditional database system.
 - Immutable table files which got generated are called as micro partitions or table file.
 - Micro-partitions are automatically performed on all Snowflake tables.
 - Table are transparently partitioned according to the ordering of the data it is inserted/loaded.
 - While data loading, I will order data by columns on which I filter data frequently. Micro-partitions in back-end will remain well grouped. Remember it is not like applying clustering key. Clustering will re-group micro partitions based on recently loaded data. But doing an order by, will not re-group old micro partitions but it will only ensure better grouping while loading data to table.
 - Within each file, the values of columns or attributes are grouped together and stored independently. This is referred to as **columnar storage**, and columns are heavily compressed using a scheme called PACs or hybrid columnar.
 - Each table file will be having a header file which contains the offset of each the table file. And it will hold other metadata information like **min max, distinct column offset**, etc.
 - Size of micro – partition when uncompressed ranges b/w **50-500 MB**.
- ⇒ What is micro partition depth?
 - There will be overlap of data in micro partitions when thousands of tables are created.
 - The degree of overlap of micro partition is represented by micro partition depth in Snowflake.
 - There are certain micro partitions which are not overlapping with each other, such micro partitions are called constant micro partitions. Their depth is always one.
- ⇒ Pre – cautions before applying clustering keys:
 - Clustering keys are not intended for all tables.
 - The size of the table, as well as the query performance of the table should dictate weather to define a clustering key for the table.
 - Table must be large enough to consist of a sufficiently large number of micro partitions, and the column(s) defined in the clustering key must provide sufficient filtering to select a subset of these micro – partitions.
 - In general, tables in the multi – terabyte (TB) range will experience the most benefit from clustering, particularly if DML commands are performed regularly on the tables.
- ⇒ **Parsing, object resolution, access control, and plan optimization** are part 'query life cycle ' in cloud services layer.
- ⇒ Query execution plan will be submitted to 'worker nodes in virtual warehouse layer'.
- ⇒ All query information and statistics are stored for audits and performance analysis in **cloud services layer**.
- ⇒ Based on the header file information, only required columns will be downloaded from remote disk.
- ⇒ Snowflake scans micro partitions well if you use numeric values as filter than string value. Over numeric values snowflake will maintain stats internally about min, max values. Which helps to scan micro partitions easily.
- ⇒ We can check the clustering information by running below command:
 - Select SYSTEM\$CLUSTERING_INFORMATION('tableName', '(cluster_key_name)');

- Note : We can also give multiple cluster_keys, also we can run this query to check how our clustering will look like if we want to perform clustering using same keys, although the result will not be always accurate (we do this to check the cardinality of columns to be put as cluster_key).
- ⇒ We can manually re-cluster the Snowflake tables using:
 - ALTER TABLE *table_name* RECLUSTER;
- ⇒ How to choose **Clustering key**?
 - Columns which are more often used in 'where' clause.
 - Columns which are more often used in 'join' conditions.
 - The order in which you specify clustering key is important. Snowflake recommends ordering the columns from lowest cardinality to highest cardinality.
- ⇒ It is not a good idea to apply clustering on the table, which frequently changes. Every time data loads in to table. Micro partitions will re group based on the clustering key. If data size is huge, re clustering cost will increase.
- ⇒ We can use 'substring' function on clustering keys to get better cardinality value over a column.
- ⇒ Before explicitly choosing to cluster a table, Snowflake strongly recommends that you test a representative set of queries on the table to establish some performance baselines. E.g. My table size is 5 TB. Daily I am inserting data into table, of around 500 MB. My table performance is slow. I want to apply clustering on the table. In this case, compare performance vs cost first.

Chapter 5: Virtual Warehouse

⇒ Used to process data.

Warehouse Size	Credits / Hour	Credits / Second	Notes
X-Small	1	0.0003	Default size for warehouses created using <code>CREATE WAREHOUSE</code> .
Small	2	0.0006	
Medium	4	0.0011	
Large	8	0.0022	
X-Large	16	0.0044	Default for warehouses created in the web interface.
2X-Large	32	0.0089	
3X-Large	64	0.0178	
4X-Large	128	0.0356	
5X-Large	256	0.0711	Preview feature.
6X-Large	512	0.1422	Preview feature.

Figure 2. Virtual warehouse - cost estimation

- ⇒ We can manager virtual warehouse/es, in case of different cases, like if there are queries getting queued for a longer duration, either we can increase the number of small warehouse or we can have one warehouse of bigger size (whichever is cost effective).
- ⇒ We can create warehouses from 'warehouse' tab or using SQL commands.
- ⇒ AUTO-SCALE mode: This mode is enabled by specifying different values for maximum and minimum clusters. In this mode, Snowflake starts and stops clusters as needed to dynamically manage the load on warehouse. This mode is statically effective to manage the resources (i.e. servers), particularly if you have large number of sessions and queries and the numbers do not fluctuate frequently. This modes comes with scaling policy, where 'Standard' is the default policy.
- ⇒ MAXIMIZED mode: This mode is enabled by specifying the same values for maximum and minimum clusters (note that the specified value must be less than 1). In this mode, when the warehouse is started snowflake starts all the starts all the clusters so that maximum resources are available while the warehouse is running.
- ⇒ Scaling policy: There are 2 types of scaling policy - Standard and Economy.
 - How many queries does a snowflake queue before it spins up additional cluster ?
 - Standard: Immediately when a query is queued, or if the system detects that There is one more query than the currently running clusters can currently execute.
 - Economy: only if the system that there is enough query load to keep the cluster busy for at least 6 minutes.
 - What is the trigger to suspend a cluster if no load is present?
 - Standard: After 2-3 consecutive successful checks, (Performed within one minute), which determine whether the load on the least-loaded cluster can be redistributed to other clusters without spinning the clusters again.
 - Economy: After 5-6 consecutive successful checks, (Performed within one minute), which determine whether the load on the least-loaded cluster can be redistributed to other clusters without spinning the clusters again.

Chapter 6: Performance Tuning

- ⇒ It is performed to improve the functioning of Snowflake to get optimized results in cost effective manner.
 - ⇒ E.g. 1) `SELECT * FROM emp;` 2) `SELECT emp_id, emp_name FROM emp;`
 - In both of these queries, the second query is better tuned/optimized.
 - ⇒ Sharing virtual warehouse during dev activities.
 - ⇒ Order your data by filter columns during data loading process.
 - ⇒ Use multi cluster warehouse instead of spinning up existing cluster to bigger size.
 - ⇒ Basically, in a snowflake, performance tuning is nothing but cost tuning.
 - ⇒ There is no concept of index.
 - ⇒ No concept of primary key, foreign key constraints.
 - ⇒ No need of transaction management.
 - ⇒ There is no buffer pool.
 - ⇒ You will never encounter out of memory exception.
-
- ⇒ How is Snowflake achieving the ACID properties?
 - QUERY MANAGEMENT AND OPTIMIZATION: When you submit a snowflake query it will go through following phases:
 - Parsing.
 - Object resolution.
 - Access control.
 - Plan optimization.
 - Snowflake won't use indexes. Why?
 - Storage medium in Snowflake is S3 and data format is compressed files.
 - Maintaining indexes significantly increases data volume and data loading time.
 - User needs to explicitly create indexes, which goes against snowflake philosophy of SAS.
 - Maintaining indexes can be complex, expensive and rescue process.
 - The way Snowflake processes the data, we don't need indices.
 - Snowflake follows all ACID properties using 'snapshot isolation'.
 - ⇒ How update works?
 - Update works as combination of Delete + Insert.
 - Hence, before executing update, check how many records will be impacted. If more than 80% of the records are getting impacted, then instead of executing update, you can think of creating the whole table or you can execute delete and insert statement separately.
 - When you are trying to update or delete, try considering the numeric columns as it is easy to scan micro-partitions.
 - ⇒ Snowflake doesn't have the concept of 'PRIMARY/FOREIGN KEY', but this doesn't mean we won't be defining them. We will be needing them in case of data migration, and other platform related activities to help us understand the relationship between the table columns.
 - ⇒ Ques. What if I abort a query while executing? Will it rollback partially updated data?
 - Ans. Yes, it follows ACID properties.

Chapter 7: Snowsight

- ⇒ Initially this feature was in 'beta' phase, but now it is the main UI of Snowflake.
- ⇒ This is a feature of Snowflake which allows you to:
 - Write query with version control enabled.
 - Create folders and organize your worksheets.
 - Capability to share worksheets.
 - Visualize data.
 - Profile data.
 - Create sample dashboards.
 - Add dynamic filter to query.
 - Interface to visualize roles hierarchy.
 - Billing Information, users' information.
 - Browse through query history, warehouse usage and resource monitors.
 - Integrates well with Snowflake marketplace data.

Chapter 8: Query Acceleration

- ⇒ The Query Acceleration service can accelerate the parts of query workload in a warehouse.
- ⇒ When it is enabled for a warehouse, it can improve the overall warehouse performance by reducing the impact of outlier queries, which are queries that use more resources than the typical query. The query acceleration does this by offloading portions of the query processing work to shared compute resources that are provided by the service.
- ⇒ The Query Acceleration service can handle the workloads more efficiently, by performing more work in parallel and reducing the wall clock time spent in scanning and filtering.
- ⇒ Scale Factor: It is a cost control mechanism which allows you to set up an upper bound the amount of computer resources a warehouse can lease for query acceleration. This value is used as a multiplier based on the warehouse size and the cost.
 - E.g. Suppose you have set a scale factor of 5 for a medium warehouse. This means that:
 - The warehouse can lease computer resources up to 5 times the size of a medium warehouse.
 - Since medium size warehouse costs 4 credits per hour, leasing these resources can cost up to an additional 20 credits per hour (4 X 5).
- ⇒ To check if a query is eligible for query acceleration, We need to perform some tasks having the role of Account admin:
 - Copy the query, add of the query.
 - Run the below query:
 - `SELECT parse_json(system$estimate_query_acceleration('query_id');`
 - We Can check the 'status' of query acceleration from the result.
 - We can also check the eligibility by following query:
 - `SELECT * FROM snowflake.account_usage.query_acceleration_eligible ORDER BY eligible_query_acceleration_time DESC;`
 - This will display all the queries eligible for query acceleration.
 - Note : Running the above query also requires the role of Account Admin.

Chapter 9: Search optimization

- ⇒ The Search Optimization Service can significantly improve the performance of such type of lookup and analytical queries that use extensive set of predicates for filtering.
- ⇒ Selective point queries on lookup tables. A point query returns only one or a small number of distinct rows. Once you identify the queries which can benefit from search optimization, you can configure columns and tables used in those queries.
- ⇒ Problem: Suppose we need to fetch 50 records from 20,000,000 records.
 - In this scenario, we can scale up virtual warehouse, but this is not possible as a snowflake will spin a new warehouse when the query starts queuing. But in our scenario, we are not going to scale it horizontally. Also, we cannot scale it up vertically because that will add up to additional cost and will not be using all the warehouse resources (underutilization).
- ⇒ To enable SOS (Search Optimization Service), on a table, use below query:
 - `ALTER TABLE table_name ADD SEARCH OPTIMIZATION ON PREDICATE_NAME(column_name);`
 - Note: *PREDICATE_NAME* here can be 'EQUALITY, IN, ARRAY_CONTAINS, ARRAYS_OVERLAP,' etc.
- ⇒ SOS is used in places where we need fast results.
- ⇒ How search optimization works?
 - Snowflake belts a separate space (extra storage) for search optimization columns(bytes), these act as indices for columns.

Chapter 10: Optimization - Outlook

- ⇒ As the volume of data increases, you need to scale up (increase the size of the cluster).
- ⇒ As the number of queries in queue increases, you need to scale out (increase no. of clusters).
- ⇒ Optimise your query to get the best outcome. Bad queries need to be optimised. Make sure not to explode records while making joins.
- ⇒ Always follow standard syntax to get optimised query and avoid record explosion and manage cost.
- ⇒ Snowflake provides 3 types of storage considerations:
 - Automatic clustering.
 - Search optimization.
 - Materialized Views.
- ⇒ Warehouse monitor:
 - We can set limit to monitor the cost incurred by warehouses (and cloud services) which will send notification upon reaching limit and suspend the warehouses.
 - To create resource monitor, run below query (taken from documentation):
 - ```
USE ROLE ACCOUNTADMIN;
CREATE OR REPLACE RESOURCE MONITOR limit1 WITH CREDIT_QUOTA=1000
TRIGGERS ON 100 PERCENT DO SUSPEND;
ALTER WAREHOUSE wh1 SET RESOURCE_MONITOR = limit1;
```
  - We can add more features, like we can trigger suspension/notification on 90% usage too.
  - We can also create resource monitor by going to: Admin -> Cost management -> Resource monitor.
  - We can configure the email id where notification has to be sent by going to: Admin -> Contacts.
- ⇒ Budget: Budget object is used to monitor the cost of different resources like:
  - Tables.
  - Materialised views.
  - Schemas.
  - Databases.
  - Warehouses.
  - Pipelines.
  - Tasks.
    - We can go through the documentation to know how to create the budget object.
    - This feature is not available in the trial account.

## Chapter 11: Data load/unload in Snowflake

- ⇒ Before moving forth, we need to install Snow CLI and AWS CLI.
  - ⇒ Use this link for AWS CLI: <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-windows.html>
  - ⇒ Use this link for Snow CLI: Download it from Snowflake.
- Note: I couldn't download anything on company laptop.*

### 1) **Introduction to Stages in Snowflake:**

- a. Snowflake staging area is not the same as warehouse staging area.
- b. Staging area and Snowflake is a blob storage area where you load all your raw files before loading them into Snowflake database. There are two types of staging area in Snowflake:
  - i. External Staging area.
  - ii. Internal Staging area.
- c. Blob storages like: AWS S3, Azure Blob S3, Google Cloud Storage (GCS). These staging areas are called Snowflake external staging area. If we don't have subscription to these staging areas, we can use Snowflake internal staging areas.
- d. **External staging – Snowflake:**
  - i. We store how are row files in external stage before loading them to Snowflake.
  - ii. In order to make a connection with this staging area, we can use a staging object. But connection with external staging area will not be secure. To maintain a secure connection with staging area, we can use an integration object. Because stage is an object, and it will have properties like how to load data in an area.
  - iii. File format object describes file properties like comma delimited, pipe delimited, etc. We can configure file properties before loading them to stage.
  - iv. Combined all the above objects will be used by copy command to load data into Snowflake.
- e. **Internal staging – Snowflake:**
  - i. All the components will remain same as external staging. Only if we are using internal staging area, Integration object is not required as the staging area (internal) is managed by Snowflake, the connection created will be secure.
  - ii. There are 3 types of internal stages:
    - 1. **User stage –**
      - a. Represented by “@~”.
      - b. Each user has a snowflake stage allocated to them by default for storing files. This stage is convenient if your file is to be used by a single user but has to be copied to multiple tables.
      - c. Constraints:
        - i. Multiple users have to be given access to the files.
        - ii. The current user does not have insert privileges on the table where data has to be loaded into.
    - 2. **Table stage-**
      - a. Represented by “@%”.
      - b. Each table has a snowflake stage allocated to it by default for storing files.
      - c. This stage is convenient option if your file has to be accessed by multiple users, but has to be copied into a single table.
      - d. Constraints:
        - i. Multiple users have to be given access to the files.
        - ii. Unlike named stages, table stages cannot be altered or dropped.
        - iii. Table stages do not support setting file format options. Instead, you must specify file format and copy options as part of the COPY INTO <table> command.

- iv. Table stages do not perform transforming data while loading it (i.e. using a query as a source of the copy command).

### 3. Named stage –

- a. Represented by "@".
- b. Internal stages are named database objects that provide the greatest degree of flexibility for data loading. Because they are database objects, the security/access rules that apply to all objects apply.
- c. Users with appropriate privileges on the stage can load data into any table. Ownership can be transferred to another role, and privileges granted to use the stage can be modified to add or remove roles.

## 2) Stage object, file object and Copy Command

### a.

#### i. External\_stage :

- 1. CREATE OR REPLACE SCHEMA *schema\_name*;
- 2. CREATE OR REPLACE STAGE *stage\_name* url = '*url\_s3\_bucket*'  
credentials=(aws\_key\_id='key' aws\_secret\_key='secret\_key');

#### ii. Internal\_stage:

- 1. CREATE OR REPLACE SCHEMA *schema\_name*;
- 2. CREATE OR REPLACE STAGE *database\_name.schema\_name.stage\_name*;

#### iii. We can use command 'DESC' to describe any object, as these both are objects we can use it to describe their properties.

### b. Similarly we can create 'file format' using query. Later we can use this file format object in other queries.

- i. CREATE OR REPLACE SCHEMA *schema\_name*;
- ii. Use query: CREATE OR REPLACE FILE FORMAT *database\_name.schema\_name.format\_name*  
type = csv field\_delimiter = ',' skip\_header = 1 null\_if = ('NULL','null') empty\_field\_as\_null=  
true compression =gzip;

### c. Copy command is used to copy data of a stage into table.

- i. E.g. COPY INTO sales FROM s3:url= (type= csv field\_optionally\_enclosed\_by='');

## 3) Load Data Using Internal Stage Area

- a. Please refer to the documentation for this.
- b. For small sized data, we can use web console to load data (in older version), but for large sized data, we need to use Snow SQL CLI.
- c. To store the data into staging area (internal), you'll be charged by Snowflake. Hence, once all activities are completed, it is always a good practice to remove the files from the staging area.
- d. Columns from internal stage are referred by '\$' sign. E.g. The first column will be referred as 't.\$1' from internal stage 't'. The query will be:
  - i. SELECT t.\$1 FROM @db\_name.%stage\_name t;
- e. We cannot copy the files of one table staging area to another table.
- f. To upload data to Snowflake staging area, we use 'put' command, and to download data from snowflake staging area we use 'get' command. But make sure to add the files to be downloaded to staging area first.
- g. To get the file name of a file, use 'metadata\$filename'.

## 4) Load Data Using External Stage Area

### a. AWS

- i. You need an AWS Free tier account. Register for this account.
- ii. Next Create an S3 bucket (go to services -> S3 -> Create bucket -> *configure it*). Make sure to choose the same region for the bucket where your snowflake account is situated in order to save the transfer cost. Keep the defaults as it is.

- iii. Create policy on 'IAM' page (*search it on AWS*). This will grant permission to upload or download data from S3 bucket.
- iv. Create Role (*copy account id in clipboard before doing this*). Go to roles -> Choose AWS Account (Snowflake is hosted on AWS) -> Choose 'require external ID' (put 0000 for now) -> Click next and select the policy -> Fill in the name of the role -> Create Role.

**b. Next create an integration object in Snowflake.**

- i. Run below query to create AWS external stage:

1. CREATE OR REPLACE STORAGE INTEGRATION *stage\_name*  
type = external stage  
storage\_provider = s3  
Enabled = true  
storage\_aws\_role\_arn = 'arn:aws:iam::\*\*\*\*\*:role/snowflake'  
storage\_allowed\_locations = ('s3://snowflakecrctest/emp/');

2. *Note: Refer to snowflake documentation for updated query. The 'arn' can be copied from AWS along with S3 path.*
3. *Note: Recreating integration object requires reconfiguration of trust relationship with s3. Snowflake external id will change once you re-create external object. Hence re configuration is required.*

4. Get the description/property of this object by running below query:

- a. DESC INTEGRATION *stage\_name*;

5. Copy the 'STORAGE\_AWS\_IAM\_USER\_arn' 'property value' (this is arn used by snowflake) and paste it in 'trust relationship' in AWS (present in 'Roles' section). Paste it in JSON (Statement: Principal: AWS). Also copy 'STORAGE\_AWS\_EXTERNAL\_ID' 'property value' and paste it in the same JSON (Statement:Action:Condition:sts:ExternalId:).

6. Update the trust.

7. Now we are ready to query the data from S3.

- c. We can query the data using normal queries used to fetch data. (Refer documentation).
- d. You can query data directly from S3 bucket and Snowflake. It is not important to create integration object.
- e. You can filter data and create views on the top of it.
- f. To load data from S3 bucket, the copy command will be same as we did in the internal stage and vice versa.
- g. After you unload data compare record counts between table and staging area.
- h. When you are unloading data into external staging area, always use a dedicated warehouse.
- i. *Note: Refer to different copy commands from documentation.*
- j. The only way to copy a zip file from S3 bucket is to manually unzip it and then copy the files into Snowflake. If we try to copy the zip file, it will not be loaded correctly.
- k. **Upload data to AWS 3 using AWS web console:**
  - i. If you have few files to upload, you can choose AWS web console to upload the files. You can directly visit the AWS Web Console and choose the folder from where you want to upload it and then simply click the upload button, choose the file and upload it.
  - ii. If you have files greater in capacity or size (For example in GB or TB), this approach will not work. We need to use AWS command line tool to upload data (*Refer documentation for CLI*).

**l. Create external table for S3 in snowflake:**

- i. To do this perform the below query:

1. CREATE OR REPLACE external table *table\_name* WITH LOCATION =  
@*db\_name.ext\_stage\_schema\_name.stage\_name* FILE\_FORMAT = (TYPE = CSV);
2. Before creating this table, make sure to be ready with the integration object and the external stage.

3. We can display the stored JSON data in tabular form by using this below query (sample query):

a. `SELECT value:c1::varchar as ID,  
value:c2::varchar as name,  
Value:c3::varchar as dept  
FROM ext_table;`

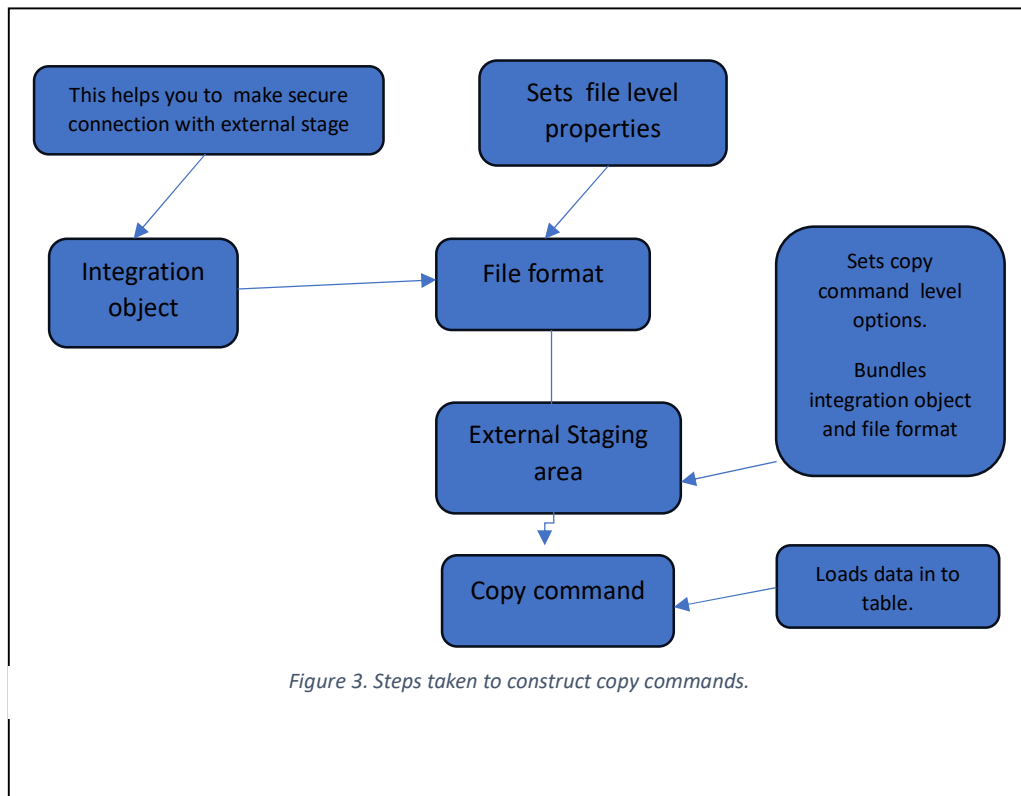
b. Note : '::' means type cast in Snowflake.

- ii. Why do we need external tables?
- iii. Suppose we want to query a file or table from S3 bucket, whenever we need this, we have to run the query '`SELECT * from view_name;`'. Every time we run this query this will scan the whole view and then return the table whatever we need. Even if we have applied some filter condition it will scan whole view. Now this will create an issue of cost. We need to handle this. Also, whenever we will need the metadata of the tables, we won't be getting it. Hence, we need external tables to give us the metadata (information schema).
- iv. When we insert data/table in S3 bucket, we need to refresh the table schema in snowflake in order to view that updated table there. To do this use below query:
  - 1. `ALTER EXTERNAL TABLE table_name REFRESH;`
- v. Same is the scenario with deleting data.
- vi. But in the case of update, Snowflake will recognize the change in hash values and mark it as a new entry. Here, we don't need to refresh the meta data table.
- vii. We can dynamically make partitions in the external table using SQL queries. This will optimise the scanning and give results quickly. When volume of data grows, we need to do this partitioning. (Refer documentation).
- viii. We also will need to do the partitioning manually when needed. We cannot use the 'refresh' feature when we have created the partitioning manually (`PARTITION_TYPE = USER_SPECIFIED`)

**m. Auto refresh external tables**

- i. We can't every time Run this query to refresh the Table, schema and Snowflake, hence we need to use some services provided by AWS (use anyone):
  - 1. SQS queue.
  - 2. SNS topic.
- ii. Go to AWS S3 bucket -> properties -> Create event notification -> enter event name -> choose 'all object create events' in object creation -> choose 'all object removal events' in object removal -> use 'SQS queue' in destination -> choose 'enter SQS queue ARN' in specify SQS queue -> (We need to get the queue ARN from snowflake by running the query – `SHOW EXTERNAL TABLES`, then copy ARN number under notification\_channel from result) -> paste the ARN number in 'SQS queue'. -> Save changes.
- iii. Snowflake designates only 1 SQS queue per S3 bucket. We can share the same SQS queue across multiple buckets in the same AWS account. **When a new or updated file is uploaded into the bucket, all external table definitions that match the stage directory path read the file details into their metadata.**

5)



## Chapter 12: Copy Options

### I. **Validate Mode**

- a. String (constant) that instructs the COPY command **to validate the data files instead of loading them into the specified table**, i.e. the COPY command tests the files for errors but does not load them. The command validates the data to be loaded and returns results based on the validation option specified.
- b. `VALIDATION_MODE = RETURN_N_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS`
  - i. We can use any one of the 3 options as per requirements.
- c. Do not use any transformation while using validation mode as that will nullify validation mode.

### II. **File Option**

- a. File option can load specific files into snowflake table.
- b. We can't use negation symbol with file option.
- c. Pattern option.

### III. **ON ERROR**

- a. You can easily reject records without failing copy commands.
- b. **When building data pipelines, using this option is a must.**
- c. You can collect the rejected records in separate table using below query:
  - i. `SELECT * FROM TABLE (validate (table_name, job_id => 'query_id'));`
  - ii. If you want to store above result in separate table, you can do so by using below query:
    - 1. `CREATE OR REPLACE TABLE table_name`  
`AS`  
`SELECT * FROM TABLE (validate (table_name, job_id => 'query_id'));`

### IV. **ENFORCE and TRUNCATE COLUMN Option**

- a. `TRUNCATE COLUMN = TRUE`, option will truncate all columns exceeding the length of defined value.
- b. `ENFORCE_LENGTH = TRUE`, option will 'fail' the COPY command if the length of any column value is exceeding the defined value. If this is set to 'FALSE', then it will only truncate the columns exceeding the length of defined value.

### V. **FORCE option.**

- a. `FORCE = TRUE`, this command will force a snowflake to load the data without rechecking or rescanning the preloaded data. Please note, this can make duplicate records to load in the table.

### VI. **PURGE option.**

- a. `PURGE = TRUE`, this command automatically deletes the table from the staging area once it is loaded into the Snowflake Database or table.
- b. We can use this option to manage our cost.

### VII. **LOAD HISTORY AND COPY HISTORY**

- a. Load History View - This INFORMATION SCHEMA view enables you to load the history of the data loaded using `COPY INTO table_name` command. The view displays one row for each file loaded.
- b. Snowflake retains the data of previously loaded files (loaded using COPY INTO command) for up to 14 days. We can retrieve these files using Load history view.
- c. The limitation of this view is that it returns an upper limit of only 10,000 rows/records.
- d. `SELECT * INFORMATION_SCHEMA.LOAD_HISTORY WHERE SCHEMA_NAME = schema_name AND TABLE_NAME = table_name;` This command will list all the historical information about the files which got loaded into the table using the COPY command.
- e. You can also use one more method. You can consider this method as the parent method of the above, load history method.
- f. COPY HISTORY – Use below query to retrieve historical data:
  - i. `SELECT * FROM TABLE (information_schema.copy_history(table_name => table_name, start_time => dateadd(hours, -42, current_timestamp())) WHERE error_count > 0;`

- ii. Note: -42 is sample time in hours. We are retrieving historical data of the Table loaded 42 hours before the current time and has anything which got failed while loading data in this table.
- g. Copy history view will give more detailed information (Pipe (snow) data) as compared to load history. So, if you are going for an audit purpose you need to use copy history command.
- h. At global level there is one more option which is accessible only to the admin user to retrieve data (Historical) i.e., using Snowflake database (shared). Under this database also you can view load history view under account usage. You can view the Historical data of the tables loaded since the past 365 days (1 year). Copy history view is also available in the account usage section.
- i. Note: Copy history view, avoids the 10,000 rows limitation of load history view.
- j. We can use snowflake database in snowflake itself to get data from snowflake database, only the difference will be 'information\_schema' will be replaced by 'account\_usage' in SQL query.
- k. If you are selecting columns from staging area Validation\_mode will not work.
  - i. COPY into *table\_name*  
 FROM (SELECT t.\$1, t.\$2, t.\$3 FROM @my\_s3\_stage/t)  
 File\_format = (type = csv field\_optionally\_enclosed\_by = "")  
 Validation\_mode = 'return\_errors';
- l. If your source file has large amount of data/file, break it into smaller chunks of files (by running 'split' command, e.g. split -b 500000 *table\_name location*), then load it into stage(using snow sql cli) and then using COPY command (in snowflake), upload it in snowflake from stage. This will improve the time taken to load the data. Instead of using larger warehouse we can do this and copy the data from stage (internal) to snowflake using smaller warehouse.



## Chapter 13: Loading Unstructured Data (JSON, XML)

### 1. Loading JSON

- a. Create a table specifying its column name as 'v' and its data type as 'variant'.

i. `CREATE OR REPLACE TABLE table_name (v variant);`

- b. Perform an insert.

i. `INSERT INTO table_name`

`SELECT`

`PARSE_JSON (`

`'{`

`"name": "Fugaku",`

`"age": 54,`

`"gender": "Male"`

`"phoneNumber": {`

`"subscriberNumber": 8920XXX190`

`}`

`"children": [`

`{`

`"name": "Sasuke",`

`"age": 23`

`"gender": "Male"`

`},`

`{`

`"name": "Itachi",`

`"age": 28`

`"gender": "Male"`

`},`

`{`

`"name": "Haruno",`

`"age": 29`

`"gender": "Female"`

`}`

`]`

`}'`

`);`

- c. Next step is to parse the json data:

i. `SELECT v: name::string as Name FROM table_name;`

ii. `SELECT v: phoneNumber.subscriberNumber::string as subscriber_number FROM table_name;`

iii. `SELECT v: children[0].name :: string FROM table_name`

`Union all`

`SELECT v: children[1].name :: string FROM table_name`

`Union all`

`SELECT v: children[2].name :: string FROM table_name;`

- d. *Refer documentation for more, learn about 'flatten' function.*

### 2. Loading XML

- a. Similar to loading JSON, create a table first:

i. `CREATE OR REPLACE TABLE table_name (v variant);`

- b. Insert the data into the table:

i. `INSERT INTO table_name`

```
SELECT parse_xml
(' <xml/> ');
```

- c. You can now view table by:
  - i. `SELECT v FROM table_name;`
  - ii. `SELECT v:"@" FROM table_name;` (this will display the root)
  - iii. `SELECT v:"$" FROM table_name;` (this will display the root and all its children/sub nodes).
- d. You can use LATERAL FLATTEN function to convert this xml to JSON.
- e. *THERE is more to learn!*

## Chapter 14: Snowpipe

⇒ Snow pipe is a feature provided by Snowflake to load data continuously.

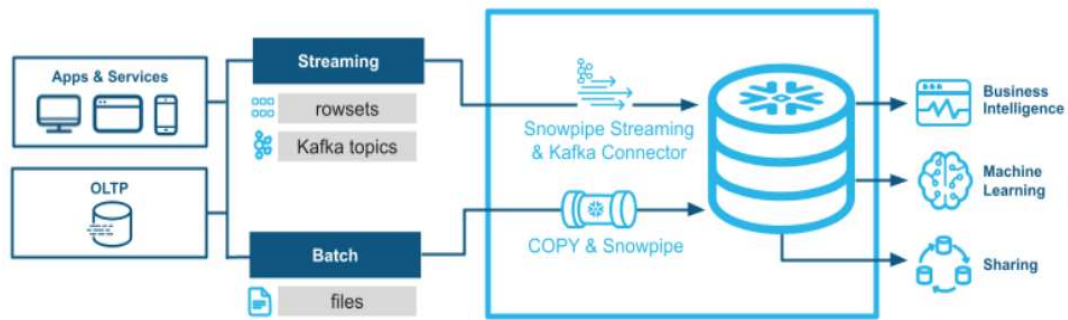


Figure 4. Snowpipe process flow.

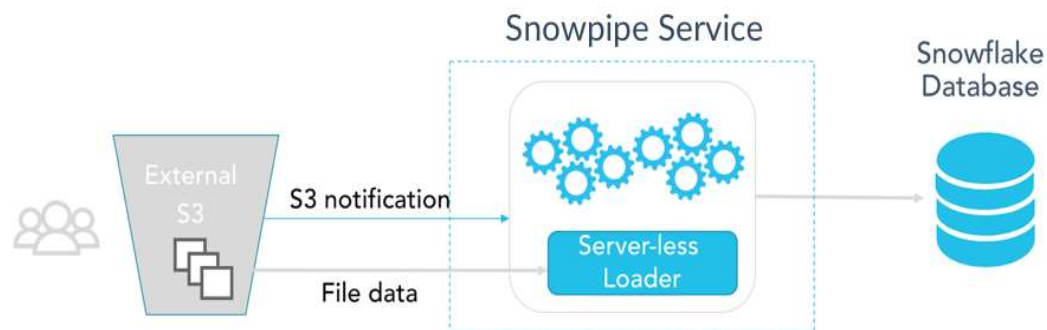


Figure 5. Snowpipe - serverless loader

- ⇒ Whenever any external bucket like S3 bucket, or streaming services like Kafka or Firehose, will send a notification regarding some data to snow pipe, it will continuously load the data in snowflake database.
- ⇒ Snow Pipe is basically a wrapper around the copy command, but it is continuously listening to the notifications.
- ⇒ Create a snowpipe:
  - Create a database.
  - Create an integration object.
  - Create a file format.
  - Create an external stage (give url to the S3 bucket if storage is S3).
  - Create a table where data has to be loaded.
  - Create a pipe to ingest JSON data, use below query:
    - `CREATE OR REPLACE pipe db_name.schema_name.pipe_name auto_ingest = TRUE as COPY INTO db_name.schema_name.table_name FROM @db_name.schema_name.stage_name file_format = (type = format_type);`
    - Now, connect it to the bucket:
      - Run query, 'SHOW pipes';
      - Copy the notification\_channel value.
      - Go to the AWS S3 bucket -> go to 'properties' -> go to 'event'.
      - Set up the event.
      - Select 'put' under 'events'. Choose 'SQS queue' under 'send to'. Choose 'Add SQS queue ARN' under 'SQS'. Paste the copied value into 'SQS queue ARN'. Save it.
    - It is done!

⇒ How will snow pipe behave on errors?

- We can't directly validate the snow pipe. That is why we need to run a query to check the status of the pipe:
  - `SELECT SYSTEM$PIPE_STATUS('pipe_name');`
  - Now, we need to run the query:
    - `SELECT * FROM TABLE (validate_pipe_load(Pipe_name => 'pipe_name', Start_time => dateadd(hour, -4, current_timestamp())));`
  - This will give the error.
- Snowflake will not directly throw any error. That is why we need to check it manually.
- Also, the copy command of snow pipe object, we won't be able to see the copy command in the query history of the history information schema table. But we can still that copy command getting registered in the copy history information schema table.
  - `SELECT * FROM TABLE (information_schema_copy.history(table_name => 'table_name', starttime => dateadd(hour, -1, current_timestamp())));`
  - We can see the failure history from the above command.
- You can't capture error records when are executing copy commands through snow pipe.
- You cannot alter or update the copy command within the pipe object. You can only recreate the pipe.
- Do not create multiple pipe objects on the same bucket.
- Snow pipe works on the name of the file and the copy command works on the hash value of the file. Hence if the file is replaced by the same name having updated values, snowpipe won't update it.
- It's a good practice to define distinct snowpipe object for each aws s3 bucket. If you define multiple pipe object on same bucket, all snowpipe objects defined on bucket will be notified.

## Chapter 15: Data Sharing

- ⇒ In the snowflake, we can directly share the metadata of a table or a database. As in snowflake architecture, all the 3 layers (cloud services, compute layer, storage layer) are separated from each other.
- ⇒ We can give some permissions so that the other party can access the storage layer of the current database or table which has to be shared.
- ⇒ Follow the below procedure to grant access to other account:
  - Create database and table which has to be shared.
  - Create an empty shell and grant it permission to access the table:
    - `CREATE SHARE share_name;`
    - `GRANT USAGE ON DATABASE db_name TO SHARE share_name;`
    - `GRANT USAGE ON SCHEMA db_name.schema_name TO SHARE share_name;`
    - `GRANT SELECT ON TABLE db_name.schema_name.table_name TO SHARE share_name;`
  - We can verify the grants given by below query:
    - `SHOW GRANTS TO SHARE share_name;`
  - Now, add the account which we want to have access to this:
    - `ALTER SHARE share_name ADD ACCOUNTS = account_id;`
  - After running this query, the given account will have the shared meta data of the table we made.
  - We can add more accounts to have the share if we want to, by using the same query.
  - Note: We only have access to the metadata information of the table, hence we cannot perform DDL/DML commands on it. We can't even clone it. **It is read-only, even if we GRANT all permissions, it won't have DDL/DML privileges.**
- ⇒ Suppose if you want to share only few columns which we want to share, we can create a secure view using below query:
  - `CREATE OR REPLACE SECURE VIEW db_name1.schema_name1.table_name1`  
`AS`  
`SELECT col1, col2, col3 FROM db_name2.schema_name2.table_name2;`
- ⇒ We can also create a Snowflake reader account for the other party if it doesn't have a snowflake account. It is marked as the child account of the account by which it was created and all bill of reader account is charged on the parent account.
- ⇒ Reader account only has the ability to read data shared to it.
- ⇒ The disadvantage of secure view is when data volume is large, snowflake won't be applying its optimization techniques hence performance would be slow.
- ⇒ You cannot share normal view to another snowflake account using shared object. You'll require a secure view for that.

## Chapter 16: Time travel

- ⇒ Time travel features used to travel back to an older version of the table.
- ⇒ Query:
  - `SELECT * FROM table_name AT(offset => -timeInSeconds);`
- ⇒ We can 'CREATE OR REPLACE' from this query result. We can also use this feature to create backup tables.
- ⇒ Note: Doing 'CREATE OR REPLACE' on the same table will change/drop the hidden ID and you will not be able to travel back in time on that table. Hence, it is advisable to create a backup table using this feature instead.
- ⇒ To insert data into production table, first truncate it (no drop), (in case no backup table is present).
- ⇒ Different versions of a file are maintained in AWS S3, and metadata will point to a specific version of file. If you want to travel back to a certain version of file, the pointer (metadata) will change to that specific version. That's how time travel works in backend.
- ⇒ A file version is only accessible till the table retention period.
- ⇒ Table retention period – The point when you did a change on a file, to a certain period (24 hrs for transient tables and 90 days for permanent tables in Enterprise edition) of time cycle.
- ⇒ We can time travel an entire database by using below query:
  - `create database restored_db clone my_db before(statement => 'query_id');`
- ⇒ We can time travel a schema using below query:
  - `create schema restored_schema clone my_schema at(offset => -timeInSeconds);`
- ⇒ Select historical data from a table using a specific timestamp:

1. `select * from my_table at(timestamp => 'Mon, 01 May 2015 16:20:00 - 0700':timestamp);`
2. `select * from my_table at(timestamp => to_timestamp(1432669154242, 3));`

- ⇒ Select historical data from a table as of 5 minutes ago:

1. `select * from my_table at(offset => -60*5) as t where t.flag = 'valid';`

- ⇒ Select historical data from a table up to, but not including any changes made by the specified transaction:

1. `select * from my_table before(statement => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');`

- ⇒ Return the difference in table data resulting from the specified transaction:

1. `select oldt.* ,newt.*  
from my_table before(statement => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726')  
as oldt  
full outer join  
my_table at(statement => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726')  
as newt  
on oldt.id = newt.id  
where oldt.id is null or newt.id is null;`

## Chapter 17: Fail Safe

- ⇒ Fail safe feature takes backup of the table automatically when the retention the is over.

### Continuous Data Protection Lifecycle



Figure 6. Life cycle of Fail Safe

- ⇒ Once data gets into the field safe zone, it is only recoverable by Snowflake.
- ⇒ Fail safe maintains permanent table data up to 7 days.
- ⇒ This feature is only available for permanent tables (automatically).
- ⇒ Post retention period the updated data is sent to fail safe zone. Note: Data is sent in the fail-safe zone only if there has been a change in the data. If there is no change, then the data will not be sent to the fail-safe zone.
- ⇒ There are 3 types of tables in Snowflake:
  - Temporary: No retention period, no fail safe. (CREATE TEMPORARY TABLE *table\_name*).
  - Transient: Retention period is present, no fail safe. (CREATE TRANSIENT TABLE *table\_name*).
  - Permanent: Both retention period and fail safe are present. (CREATE TABLE *table\_name*).
- ⇒ The issue with fail safe is that it may take unnecessary backup for those data which can be easily reproduced as permanent table, those data will go to the fail-safe zone. That's unnecessary backup. Snowflake will bill you for storing that data in the failsafe zone.
- ⇒ It is recommended to use permanent tables wherever the data is not being reproduced. Areas like staging or loading the data into stages requires frequent reproduction of the data. Hence there it is recommended to use transient or temporary tables as per the requirement.
- ⇒ Fail safe zone is not accessible to data base developers.
- ⇒ Please go through below document,
  - [ALTER DATABASE | Snowflake Documentation](#).

## Chapter 18: Cloning in Snowflake

- ⇒ In cloning we take a pointer of metadata of the same table and point it to the table. But after cloning tables will remain independent of each other.
- ⇒ Basically, a user cloning a table takes the metadata of the table into another. It will point to the same storage area in the backend.

### CLONING

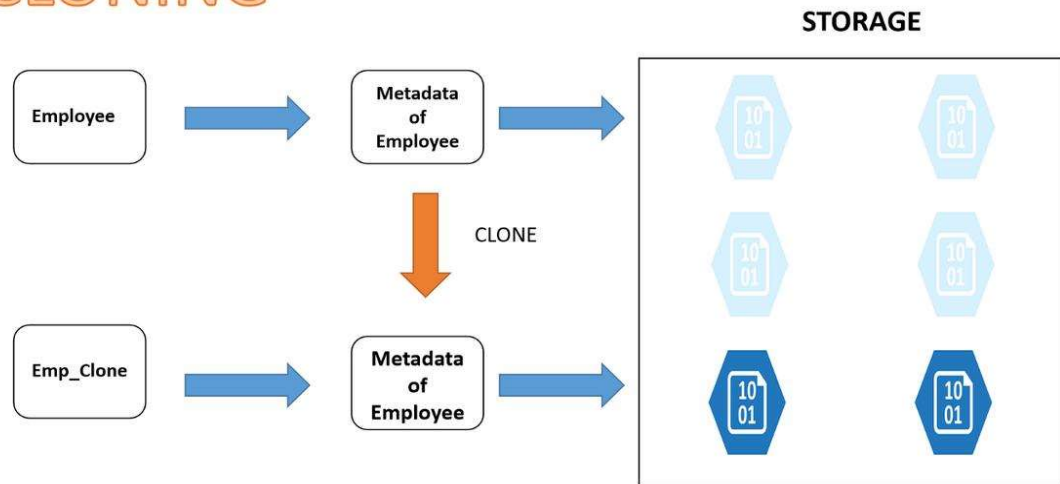


Figure 7. Cloning

- ⇒ Even when the pointers point to the same storage, any change made from or in one table will not affect the other table. But how is it possible?
  - Well, the storage area S3 keeps the data, but Snowflake intelligently keeps different versions for different clones. Only the data which is common between the 2 tables will be shared together and the rest will be kept separate.
- ⇒ Table Swap: This is the process in which we swap the metadata information of 2 tables where data needs to be swapped instead of swapping the whole data. In this case the pointers or the metadata information is exchanged or swapped.
  - Use below query to do so:
    - `ALTER TABLE table_name1 SWAP WITH table_name2;`
  - This can be helpful in scenarios like when you want to swap the data of production environment to the development environment or vice versa.



## Chapter 19: Data Sampling

- ⇒ Data sampling is a technique in which developer takes a small sample/portion of data from a large set of data to test it and do development activities. This is usually done to avoid the errors which we can make on the large set of data and for which Snowflake will bill us. That is why we take small portions or samples of data so that we can do all the trial and error activities on this small set of data.
- ⇒ There are two sampling methods in Snowflake:
  - System / Block method: In this case, the sample is formed of randomly selected blocks of data rows forming the micro-partitions of the FROM table, each partition chosen with a  $\langle P \rangle / 100$  probability.
  - Bernoulli / Row method: Snowflake selects each row of the FROM table with  $\langle P \rangle / 100$  probability. The resulting sample size is approximately of  $\langle P \rangle / 100 * \text{number of rows on the FROM expression}$ .
- ⇒ Which one to choose?
  - If it is bigger data set, use System method.
  - If data size is smaller, use Bernoulli method.
- ⇒ E.g. of data sampling by using 'system method':
  - CREATE OR REPLACE TABLE *table\_name1*  
AS  
SELECT \* FROM *table\_name2* sample system (*int\_value\_of\_* $\langle p \rangle$ ) seed (82);
  - Seed (82) is a name given if someone has to create the same result with probability, they can use 'seed (82)'. We can tell other developer, to use 'seed (82)' to create the same sample which we created.
- ⇒ E.g. of data sampling by using 'Row method':
  - CREATE OR REPLACE TABLE *table\_name1*  
AS  
SELECT \* FROM *table\_name2* sample row (*int\_value\_of\_* $\langle p \rangle$ ) seed (82);
- ⇒ Sampling v/s cloning: It is always better to use sampling for development purposes instead of cloning because in cloning we will apply all the development activities on the whole table which can be large. This will cost us more than what we require to do in sampling.
- ⇒ Please go through this blog,
  - <https://sonra.io/2018/08/02/sampling-in-snowflake-approximate-query-processing-for-fast-data-visualisation/>
- ⇒ Please go through below documentation,
  - <https://docs.snowflake.net/manuals/sql-reference/constructs/sample.html>
- ⇒ Also go through different methods to take sample,
- ⇒ Ref(<https://docs.snowflake.com/en/sql-reference/constructs/sample.html>)
- ⇒ ***Fraction-based Row Sampling***
- ⇒ Return a sample of a table in which each row has a 10% probability of being included in the sample:
  - 1. `select * from testtable sample (10);`
- ⇒ Return a sample of a table in which each row has a 20.3% probability of being included in the sample:
  - 1. `select * from testtable table sample bernoulli (20.3);`
- ⇒ Return an entire table, including all rows in the table:

```
1. select * from testtable table sample (100);
```

⇒ Return an empty sample:

```
1. select * from testtable sample row (0);
```

⇒ This example shows how to sample multiple tables in a join:

```
1. select i, j
 from
 table1 as t1 sample (25) -- 25% of rows in table
 inner join
 table2 as t2 sample (50) -- 50% of rows in table2
 where t2.j = t1.i;
```

⇒ The **SAMPLE** clause applies to only one table, not all preceding tables or the entire expression prior to the **SAMPLE** clause. The following **JOIN** operation joins all rows of t1 to a sample of 50% of the rows in table2; it does not sample 50% of the rows that result from joining all rows in both tables:

```
1. select i, j
 from table1 as t1 inner join table2 as t2 sample (50)
 where t2.j = t1.i;
```

⇒ To apply the **SAMPLE** clause to the result of a **JOIN**, rather than to the individual tables in the **JOIN**, apply the **JOIN** to an inline view that contains the result of the **JOIN**. For example, perform the **JOIN** as a subquery, and then apply the **SAMPLE** to the result of the subquery. The example below samples approximately 1% of the rows returned by the **JOIN**:

```
1. select * from (
 select *
 from t1 join t2
 on t1.a = t2.c
) sample (1);
```

⇒ **Fraction-based Block Sampling (with Seeds)**

⇒ Return a sample of a table in which each block of rows has a 3% probability of being included in the sample, and set the seed to 82:

```
1. select * from testtable sample system (3) seed (82);
```

⇒ Return a sample of a table in which each block of rows has a 0.012% probability of being included in the sample, and set the seed to 99992:

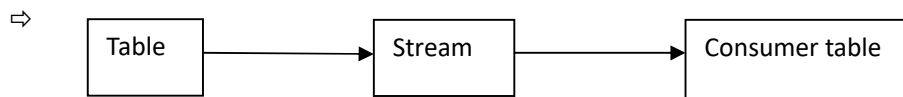
```
1. select * from testtable sample block (0.012) repeatable (99992);
```

## Chapter 20: Scheduling Tasks

- ⇒ Tasks in Snowflake are used to schedule query execution. We can create simple tree of tasks to execute queries by creating dependencies.
- ⇒ Tasks can be combined with table streams for continuous ELT workflows to process recently changed table rows.
- ⇒ Query to create a task (example):
  - CREATE OR REPLACE TASK *task\_name*  
WAREHOUSE = *warehouse\_name*  
SCHEDULE = '1 MINUTE'  
AS  
INSERT INTO *table\_name(ts)* VALUES(CURRENT\_TIMESTAMP);
  - CREATE OR REPLACE TASK *task\_name*  
WAREHOUSE = *warehouse\_name*  
SCHEDULE = 'USING CRON 0 9-17 \* \* SUN AMERICA/Los\_Angeles'  
TIMESTAMP\_INPUT\_FORMAT = 'YYYY-MM-DD HH24'  
AS  
INSERT INTO *table\_name(ts)* VALUES(CURRENT\_TIMESTAMP);
- ⇒ Use query, 'SHOW TASKS' to see created tasks (initially suspended).
- ⇒ When you create a task, it won't get automatically scheduled. In order to schedule them, we need to run the below query:
  - ALTER TASK *task\_name* RESUME;
- ⇒ To suspend a task use below query:
  - ALTER TASK *task\_name* SUSPEND;
- ⇒ To see all the scheduled tasks run below query:
  - SELECT \* FROM TABLE(information\_schema.task\_history()) ORDER BY scheduled\_time;
- ⇒ And snowflake, we have dependency on one task to the other. A task having no dependency cannot run. Tasks run according to their dependency. If Task A is scheduled to be run before task B, then task A is dependent on task B.
- ⇒ To create a task dependency, use 'AFTER' keyword.
  - E.g.: CREATE TASK *task2* WAREHOUSE = *compute\_wh* AFTER *task1*  
AS INSERT INTO TASK2 select '2';
- ⇒ We can see task dependency by using below command:
  - SHOW TASKS LIKE 'TASK%';
- ⇒ We need to resume tasks to run. A task in suspended cannot run even after having dependency.
- ⇒ On task failure, there is no mechanism to send notification/email.

## Chapter 21: Streams

- ⇒ A stream object records, DML or data manipulation changes to a table, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data.
- ⇒ Note that stream object itself does not carry any data. A stream only stores the offset(a point of time from where all the DML operations are recorded) of the source table and returns changed data capture records by leveraging the versioning history of the source table (Know that in storage layers like a AWS, GCP or Azure, Files are fully overwritten and not changed as they are immutable and thus these storages maintain versions of the files.).
- ⇒ There are 2 types of streams:
  - Standard stream(default):
    - Captures all data change operation on source table.
  - Append only stream:
    - Only tracks insert operation on source table.
- ⇒ To create a stream object over a table 'myTable', use below query:
  - CREATE OR REPLACE STREAM *stream\_name* OVER *myTable*.
- ⇒ To check the stream object:
  - SELECT \* FROM *stream\_name*;
- ⇒ To check offset of the current stream object:
  - SELECT SYSTEM\$STREAM\_GET\_TABLE\_TIMESTAMP('stream\_name');
  - We'll get a timestamp as a result from this query.
  - Paste it in: SELECT TO\_TIMESTAMP(*timestamp\_we\_got*);
- ⇒ We need to create multiple stream objects, if we have multiple consumer tables.
- ⇒ You can tie streams with task.



- ⇒ We can use a 'changes' clause to track all the changes which happened without using streams.
  - E.g. SELECT \* FROM *t1* changes(information => default) at(timestamp => *\$timeStamp1*);
    - Note: *timeStamp1* variable needs to be initialized using: SET timestamp = (SELECT current\_timestamp());
  - We don't have the concept of offset in this.
  - We need to allow change tracking before using this:
    - ALTER TABLE *t1* SET change\_tracking = true;

## Chapter 21: Continuous Data load

⇒ We can tie Snowpipe, streams and tasks to build continuous data load in snowflake.

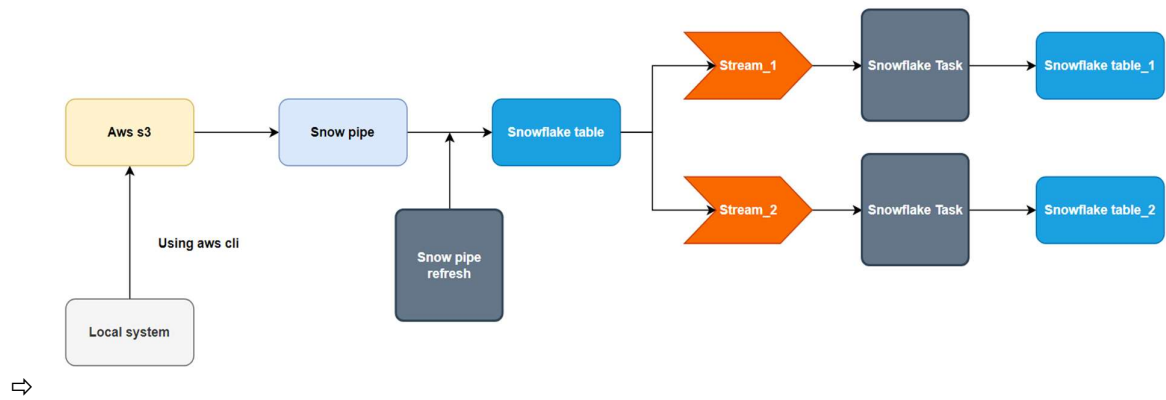


Figure 8. Process flow of continuous data load

⇒ If it is a large file, we need to split it into smaller files.

## Chapter 22: Materialized Views

- ⇒ A database object that holds the results of a query is called materialized view.
- ⇒ Basically, **A view can capture a complex select statement and make it simple to run repeatedly.**
- ⇒ Unlike a view, it's not a window into a database.
- ⇒ Rather, it is a separate object holding query results with data refreshed periodically.
- ⇒ Problems with conventional MVs:
  - Refreshing data periodically can lead to inconsistent or out-of-date results when you access MVs.
  - DML operations traditionally experienced slow-downs when they used MVs.
- ⇒ Properties of Snowflake MVs:
  - Ensure optimal speed.
  - Deliver query results via MVs that are always current and consistent with the main data table.
  - Provide exceptional ease of use via a maintenance service that continuously runs and updates MVs in the background.
  - Snowflake MVs enhance data performance by helping you filter data so you can perform resource-intensive operations and store the results, eliminating the need to continuously or frequently perform resource-intensive operations.
  - Snowflake performs automatic background maintenance of MVs.
  - When a base table changes, a background service automatically updates all MVs defined on the table.

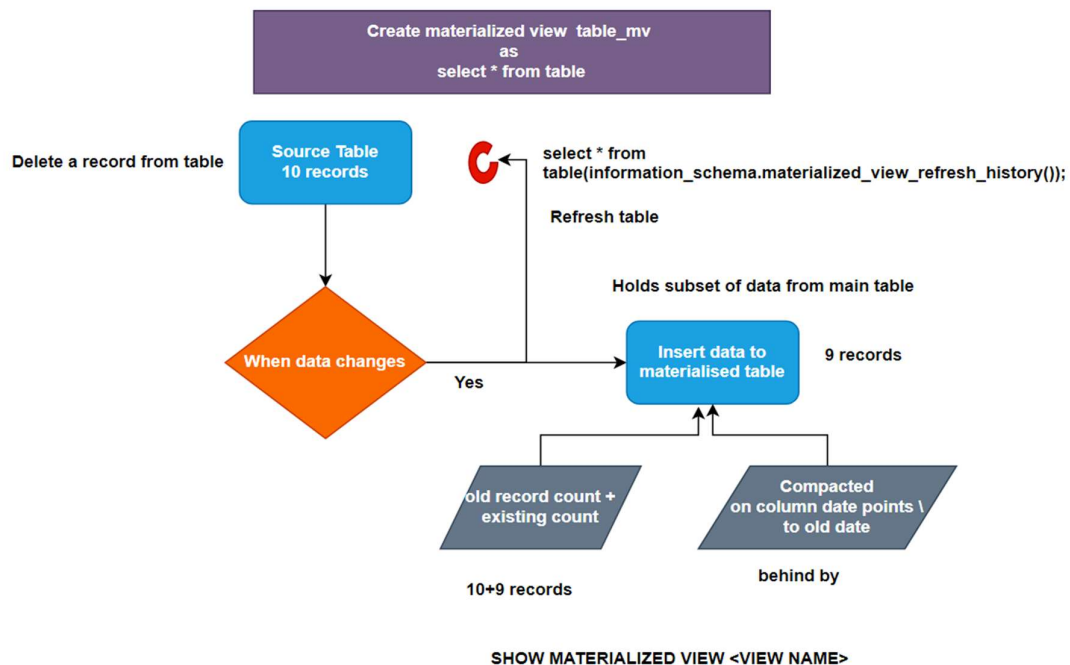


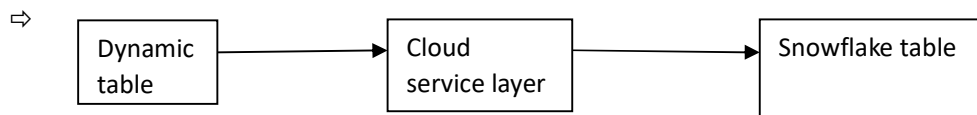
Figure 9. MVs under the hood.

- ⇒ You can check the cost associated with the MVs in the usage section.
- ⇒ Once data is changed in the main table, changed data will be pulled to the materialized table.
- ⇒ If underlying data has changed, if you execute select query on materialized table there will be a refreshing cost.
- ⇒ Don't create materialized view on large tables which has frequent data changes.
- ⇒ If we can achieve the same business requirements using other Snowflake features, do it.

- ⇒ The automatic maintenance of materialized views is not free of cost.
- ⇒ MVs use storage (not cache).
- ⇒ *Refer documentation.*
  
- ⇒ The following limitations apply to creating materialized views:
  - A materialized view can query only a single table.
  - Joins, including self-joins, are not supported.
  - A materialized view cannot query:
    - A materialized view.
    - A non-materialized view.
    - A UDTF (user-defined table function).
  - A materialized view cannot include:
    - UDFs (this limitation applies to all types of user-defined functions, including external functions).
    - Window functions.
    - HAVING clauses.
    - ORDER BY clause.
    - LIMIT clause.
  - GROUP BY keys that are not within the SELECT list. All GROUP BY keys in a materialized view must be part of the SELECT list.
  - GROUP BY GROUPING SETS.
  - GROUP BY ROLLUP.
  - GROUP BY CUBE.
  - Nesting of subqueries within a materialized view.
- ⇒ Read more at,
  - [Working with Materialized Views | Snowflake Documentation](#)

## Chapter 23: Dynamic tables

- ⇒ As we know in normal views, when the source table has some changes, the view gets updated accordingly. But the data is not persistent on the view. This issue is resolved in materialized view.
- ⇒ In materialized view, the data is persistent on the view. But MVs have their own disadvantages too:
  - You can't write complex queries.
  - Persisted result refresh is not in your control.
- ⇒ Hence, we need to have a solution for both the problems, say:
  - Persist the query result.
  - Should be able to write complex queries.
  - Should be able to determine how often the persistent result should be refreshed.
- ⇒ So the solution to these problems is “**Dynamic tables**”.
- ⇒ You can create dynamic tables using query:
  - ```
CREATE OR REPLACE DYNAMIC TABLE table_name
  TARGET_LAG = '20 minutes'
  WAREHOUSE = warehouse_name
  AS
  .....<any simple or complex logic>.....
```
- ⇒ We can use below query to manually refresh the dynamic table in case the target_lag time period is not hit, and we want to refresh it:
 - `ALTER DYNAMIC TABLE table_name REFRESH;`
- ⇒ Dynamic tables are materialised views that are maintained by Snowflake. And because they are maintained by Snowflake, you cannot directly perform any query or operations on them. You can only perform those actions on the source table and the rest will be taken care by Snowflake.



- ⇒ Above is shown the process followed by a dynamic table when there is a change in the source table.
- ⇒ There are 2 types of dynamic refresh in dynamic table:
 - Full refresh.
 - Incremental refresh.
- ⇒ Downstream property: When the table in downstream gets updated, then the tables in upstream will also get updated. (Upstream tables are like tables which come before downstream tables in dependency). [TARGET_LAG = downstream].
- ⇒ *Note : Refer to documentation, many things aren't covered here.*
- ⇒ The following results are possible, depending on how each dynamic table specifies its lag:

Dynamic Table 1 (DT1)	Dynamic Table 2 (DT2)	Refresh results
TARGET_LAG = DOWNSTREAM	TARGET_LAG = 10 minutes	DT2 is updated at most every 10 minutes. DT1 gets, or infers, its lag from DT2 and is updated every time DT2 requires updates.
TARGET_LAG = 10 minutes	TARGET_LAG = DOWNSTREAM	This scenario should be avoided. The report query will not receive any data. DT2 is not refreshed as no DT is built on top of DT2. Furthermore, DT1 will be frequently refreshed.
TARGET_LAG = 5 minutes	TARGET_LAG = 10 minutes	DT2 is updated approximately every 10 minutes with data from DT1 that is at most 5 minutes old.

Dynamic Table 1 (DT1)	Dynamic Table 2 (DT2)	Refresh results
TARGET_LAG = DOWNSTREAM	TARGET_LAG = DOWNSTREAM	DT2 is not refreshed periodically because DT1 has no downstream children with a defined lag.

Chapter 24: Data Masking

- ⇒ Data masking is performed to hide sensitive data. E.g. A salary report made for manager can be shared with an analyst who is not supposed to view the salaries of other employees. For this purpose we need data masking.
- ⇒ Data masking is also applied to comply with HIPAA rules (rules of making certain data private in medical records).
- ⇒ We can create a view to perform data masking. But we'll be facing some issues:
 - Managing view for large no. of tables (say 100), is a big issue in case of DML operations.
 - Possibility of multiple views creation in the same table.
 - Owner of table has full control.
 - Masking on columns is code dependent.
 - Masking function can be manipulated.
- ⇒ Snowflake has come up with a solution to this issue, it has 'access control' feature which controls the access to the table.
- ⇒ To create masking policy use below query:
 - ```
CREATE MASKING POLICY policy_name AS (val string) RETURNS string ->
CASE
 WHEN current_role() IN ('role_name') THEN val
 ELSE '*****'
END;
```
  - '*val string*' is the type of column on which this policy is going to be applied.
  - Now create the table on which the policy is going to be applied:
    - ```
CREATE OR REPLACE TABLE table_name(
    Column_name data_type MASKING POLICY policy_name, col2 data_type2
);
```
- ⇒ We can add masking policy to an existing table:
 - ```
ALTER TABLE IF EXISTS table_name MODIFY COLUMN column_name SET MASKING POLICY policy_name;
```
- ⇒ Issues with masking:
  - We can only apply masking to 1 argument(column).
  - To solve this issue we can use function and views.
- ⇒ To create a function, use below query:
  - ```
CREATE OR REPLACE FUNCTION "function_name"("col1" data_type1, "col2" data_type2)
RETURNS data_type
LANGUAGE SQL
AS
SELECT
    CASE WHEN current_role() IN ("role1", "role2") THEN
        .....
    END;
```
- ⇒ We can create secure view which can be seen by them.

Chapter 25: Row accessing policy

- ⇒ Row access policy implement row-level security to determine which rows are visible in the query result.
- ⇒ Row access policies can include conditions and functions in the policy expression to transform the data at query runtime when those conditions are met.
- ⇒ The policy-driven approach supports segregation of duties to allow governance teams to define policies that can limit sensitive data exposure.
- ⇒ To create a row access policy, use below query:
 - `CREATE OR REPLACE ROW ACCESS POLICY policy_name`
`AS`
`(col_name data_type) RETURNS data_type ->`
`CASE WHEN col_name = 'value' THEN value ELSE value END;`
- ⇒ Now modify the row in the table using below query:
 - `ALTER TABLE table_name ADD ROW ACCESS POLICY policy_name ON col_name;`
- ⇒ Object owner also cannot see the records after applying that policy.
- ⇒ You can apply only one policy at a time on the table.
- ⇒ You can use multiple columns as input parameter for policy.

Note: Further lessons were easy and theoretical, so best way to learn will be documentation. Refer to [->](#)