

*******Snowflake Topics*******

- 1. What is Snowflake & Snowflake features?**
- 2. Architecture & Key Concepts?**
- 3. Virtual warehouse creation & scaling methods (Maximized, Auto scale)?**
- 4. Caching in snowflake & Types of Chasings?**
- 5. Clustering in snowflake?**
- 6. Data sharing in snowflake (Reader Account)?**
- 7. Time travel & Fail safe in snowflake?**
- 8. Stages & Types of Stages and Creation of Stages?**
- 9. File formats & File types and Options (Force, Purge, etc...)?**
- 10. Database creation & Tables & Types of tables?**
- 11. Clone & Swap creation in snowflake?**
- 12. Views & Types of views and MV maintenance cost and Limitations?**
- 13. Copy command & Options and Data loading (Local file system, Could)?**
- 14. Snow pipe in snowflake?**
- 15. Streaming in snowflake & Types of Streaming, SCD's, Change Clause?**
- 16. Scheduling-TASKS in snowflake?**
- 17. Loading unstructured data (Jason, XML, and Parquet)?**
- 18. Data sampling in snowflake (Row, Block, Clone)?**
- 19. Performance tuning?**
- 20. AWS s3 account creation (S3 bucket, Policy, Role) & Creating Integration.**
- 21. Snow CLI, AWS CLI?**

1. What is Snowflake & Snowflake features?

- **Snowflake** : Snowflake is an analytic data warehouse database as Software-as-a-Service (SAAS)

.SAAS:

- There is no hardware, software (virtual or physical) to install, configure, or manage.
- Ongoing maintenance, management, and tuning is handled by Snowflake (software installation and updates).
- Snowflake runs completely on cloud infrastructure and cannot be run on private cloud infrastructures (on-premises or hosted).

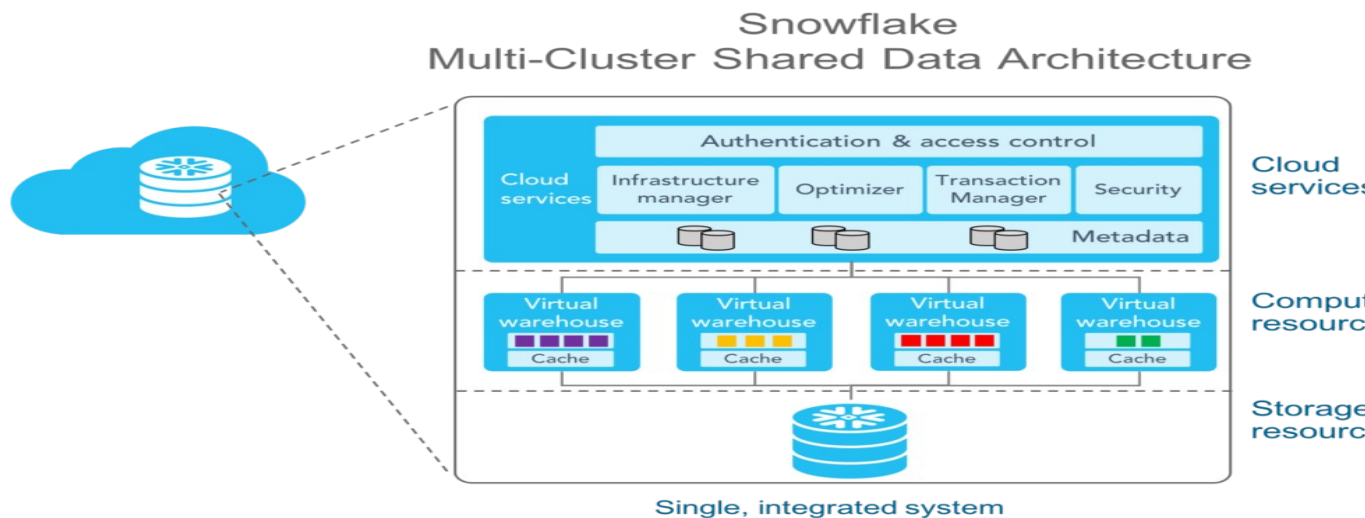
.Snowflake Features:

1. Caching
2. Clustering
3. Data sharing
4. Time travel & Fail safe
5. Clone & Swap
6. Snow pipe
7. Streaming & Change Clause
8. Scheduling-TASKS

2. Architecture & Key Concepts?

- **Snowflake Architecture**
 - Snowflake's architecture is a shared-disk and shared-nothing processing architectures.
 - Shared-disk: Snowflake uses a central data repository for persisted data that is accessible from all compute nodes.
 - Shared-nothing processing: Snowflake processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally.
- **Key Concepts**
 1. Database Storage
 2. Query Processing

3. Cloud Services



- **Database Storage:** Snowflake loaded or stored the data as optimized, compressed, columnar format in cloud database storage.

The data objects stored by Snowflake are not directly visible nor accessible by customers, they are only accessible through SQL query operations run using Snowflake.

- **Query Processing:** Snowflake processes queries using “virtual warehouses”.

Each virtual warehouse is an independent compute cluster that does not share compute resources with other virtual warehouses

- **Cloud Services:** The cloud services layer is a collection of services that coordinate all to process user requests, from login to query dispatch using below collection of services.
 - Authentication
 - Infrastructure management
 - Metadata management
 - Query parsing and optimization
 - Access control

3. Virtual warehouse creation & scaling methods (Maximized, Auto scale)?

```
CREATE OR REPLACE WAREHOUSE KNOW_ARCHITECTURE_1 WITH
```

```
WAREHOUSE_SIZE='X-SMALL'
```

```
AUTO_SUSPEND = 180
```

```
AUTO_RESUME = TRUE
```

`INITIALLY_SUSPENDED=TRUE;`

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 the warehouse this mode.

MAXIMIZED MODE: This mode is enabled by specifying the same value for both maximum and minimum clusters. In this mode, when the warehouse is started, Snowflake starts all the clusters so that maximum resources are available while the warehouse is running.

Scale Up: Doing server scale up like XS to S.

Scale Down: Doing server scale down like S to XS.

Scale Out: Increasing cluster size like 2 serve to 3 servers.

Scale In: Decreasing cluster size like 3 server to 2 server.

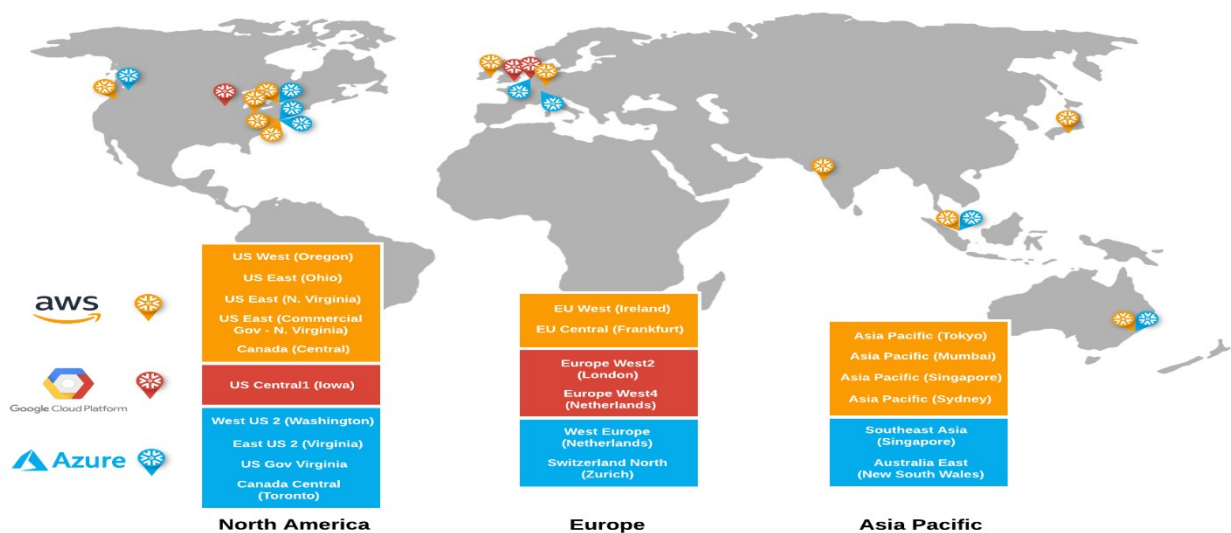
1. X-Small (1 cluster, 1Server,8 VW(8 CPU's or 8 threads).
2. Small (1 cluster, 2Server,16 VW(16 CPU's or 16 threads)
3. Medium (1 cluster, 4Server,32 VW(32 CPU's or 32 threads)
4. Large (1 cluster, 8Server,64 VW(64 CPU's or 64 threads)
5. X-Large (1 cluster, 16Server,128 VW(128 CPU's or 128 threads)
6. 2x-Large (1 cluster, 32Server,256 VW(256CPU's or 256 threads)
7. 3x-Large (1 cluster, 64Server,512 VW(512 CPU's or 512 threads)
8. 4x-Large (1 cluster, 128Server,1024 VW(1024CPU's or 1024 threads)

Virtual Warehouse Sizes:




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

Running Time	Credits (X-Small)	Credits (X-Large)	Credits (3X-Large)
0-60 seconds	0.017	0.267	1.067
61 seconds	0.017	0.271	1.084
2 minutes	0.033	0.533	2.133
10 minutes	0.167	2.667	10.667
1 hour	1.000	16.000	64.000

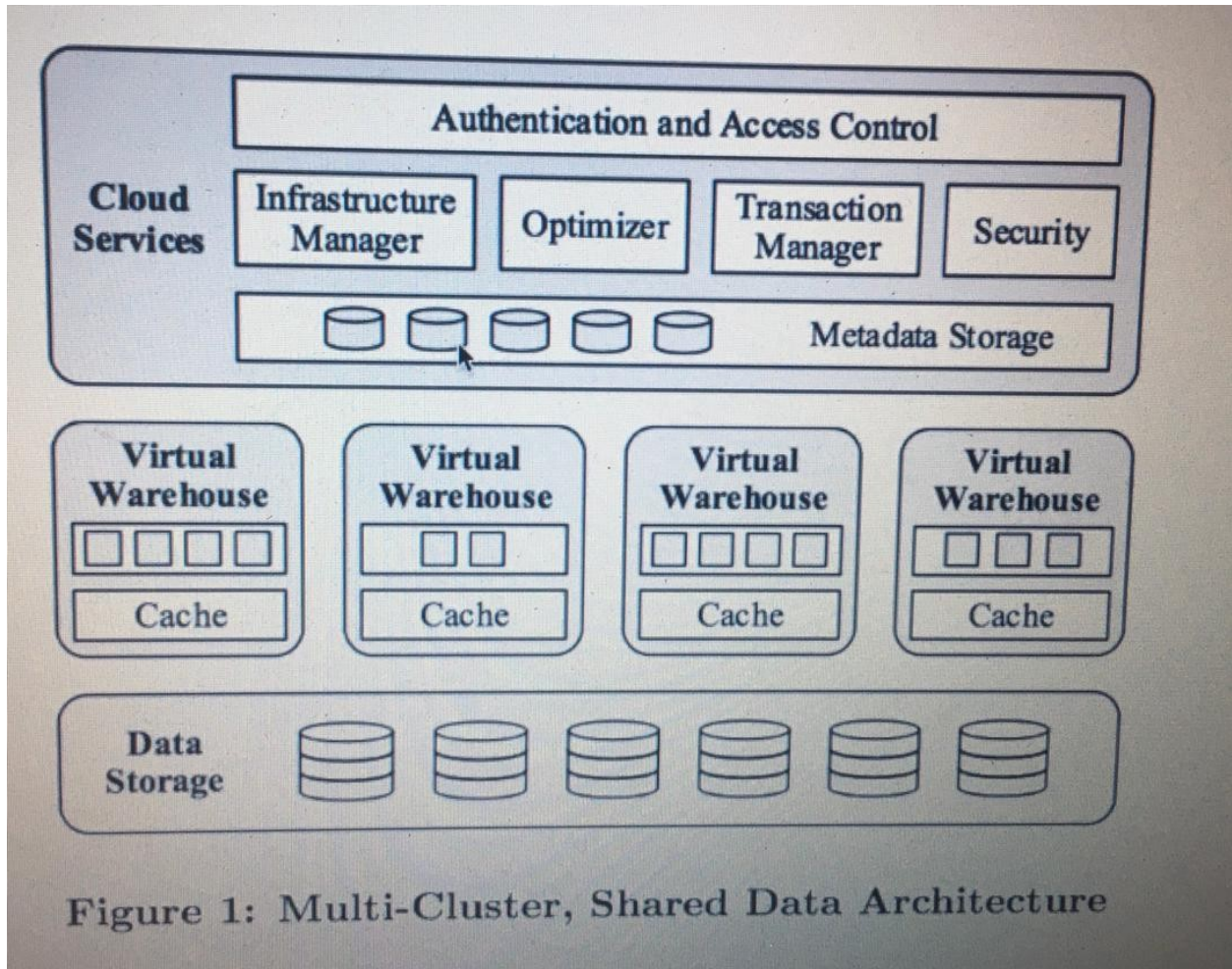
Snowflake Regions:



Snowflake Editions:

STANDARD	ENTERPRISE	BUSINESS CRITICAL	VIRTUAL PRIVATE SNOWFLAKE (VPS)
 <p>Complete SQL data warehouse Secure Data Sharing across regions / clouds Premier Support 24 x 365 1 day of time travel Built-in enterprise grade encryption in transit and at rest Customer-dedicated virtual warehouses Federated authentication Database replication External Functions Snowsight Create your own Data Exchange Data Marketplace access</p> <p>\$2.20 cost per credit</p> <p>GET STARTED</p>	 <p>Standard +</p> <p>Multi-cluster warehouse Up to 90 days of time travel Annual rekey of all encrypted data Materialized views Search Optimization Service Dynamic Data Masking External Data Tokenization</p> <p>\$3.30 cost per credit</p> <p>GET STARTED</p>	 <p>Enterprise +</p> <p>HIPAA support PCI compliance Data encryption everywhere Tri-Secret Secure using customer-managed keys AWS PrivateLink support Database failover and fallback for business continuity</p> <p>\$4.40 cost per credit</p> <p>GET STARTED</p>	 <p>Business Critical +</p> <p>Customer-dedicated virtual servers wherever the encryption key is in memory Customer-dedicated metadata store</p> <p>CONTACT US</p>

4. Caching in snowflake & Types of Chasings?



Profile Overview (Finished)

Total Execution Time (1m 38.992s)



Processing	62 %
Local Disk IO	0 %
Synchronization	0 %
Initialization	38 %

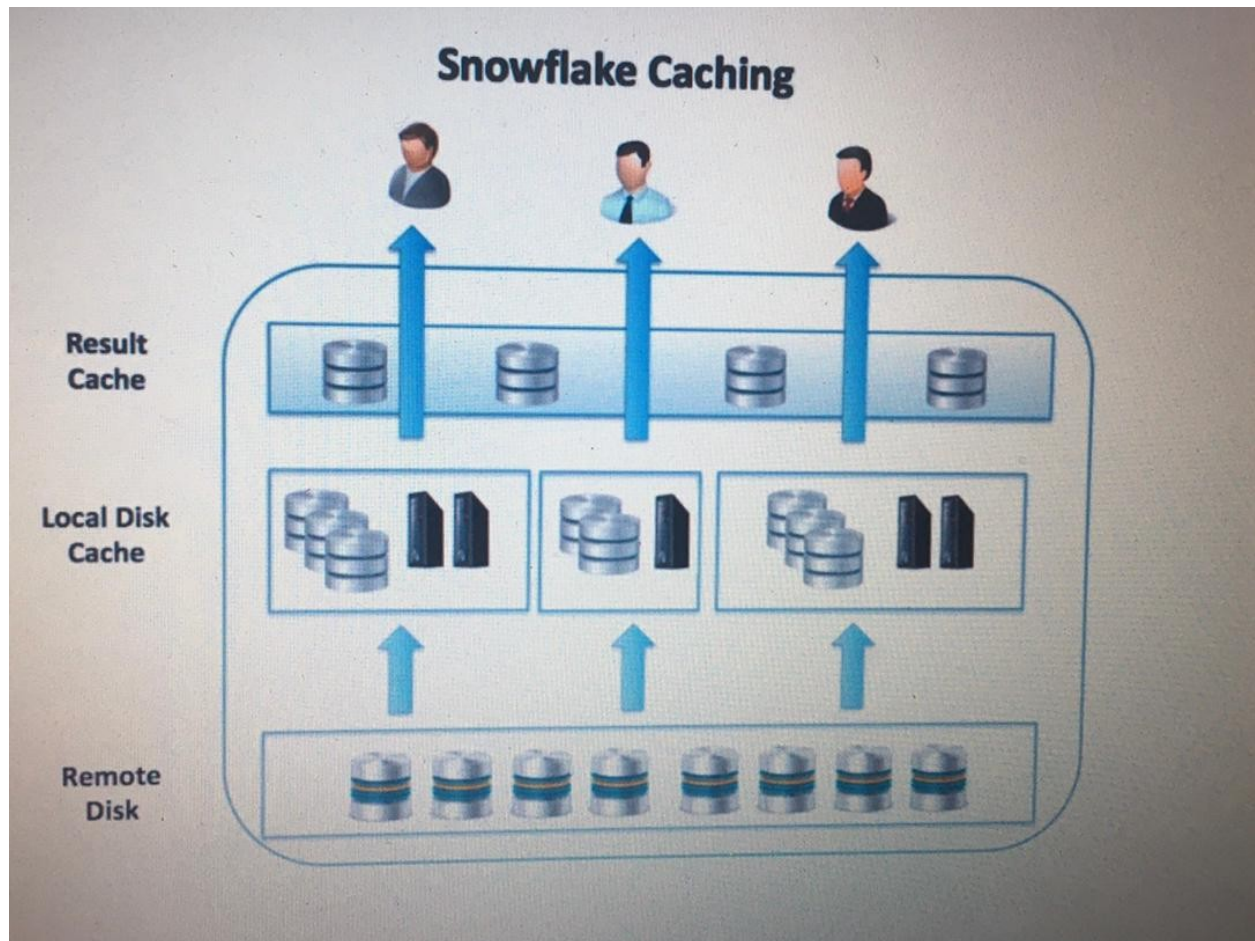
Total Statistics

IO

Scan progress	100.00 %
Bytes scanned	6.47 GB
Percentage scanned from cache	100.00 %
Bytes written to result	5.33 GB

Pruning

Partitions scanned	432
Partitions total	432



When user given query request, the Cloud services take that request and submit to the VWH then VWH will go and take the data form data storage and give to Cloud service.

Finally the cloud service give result to user and store the result in Result cash.

Types of Caching's:

1. **Result Cashing:** The result will store in Cloud services.
 2. **Local disk chasing:** The query result will store in VWH cash.
 3. **Remote disk chasing:** It will store in cloud caching.
- Query Results are Re-Used Based on the below Criteria
 - Caching the Query Results for result set re-use
 - Retention Period – 24 Hours
 - The user executing the query has the necessary access privileges for all the tables used in the query.

- The new query syntactically matches the previously-executed query.
 - The table data contributing to the query result has not changed.
 - The persisted result for the previous query is still available.
 - The query does not include functions that must be evaluated at execution (e.g. [CURRENT_TIMESTAMP](#)).
 - The table's micro-partitions have not changed (e.g. been re clustered or consolidated) due to changes to other data in the table.
- Session Parameter `USE_CACHED_RESULT = true;`
`Alter session set USE_CACHED_RESULT = TRUE;`
 - Result_Scan Function for getting the Query Results from Cache →
`RESULT_SCAN ({ '<query_id>' | LAST_QUERY_ID() })`
 - Retrieve All the Values from first Query in the session
 - `select * from table(result_scan(last_query_id(1)));`
 - Retrieve All the Values from second most recent query in the session
 - `select * from table(result_scan(last_query_id(-2)));`

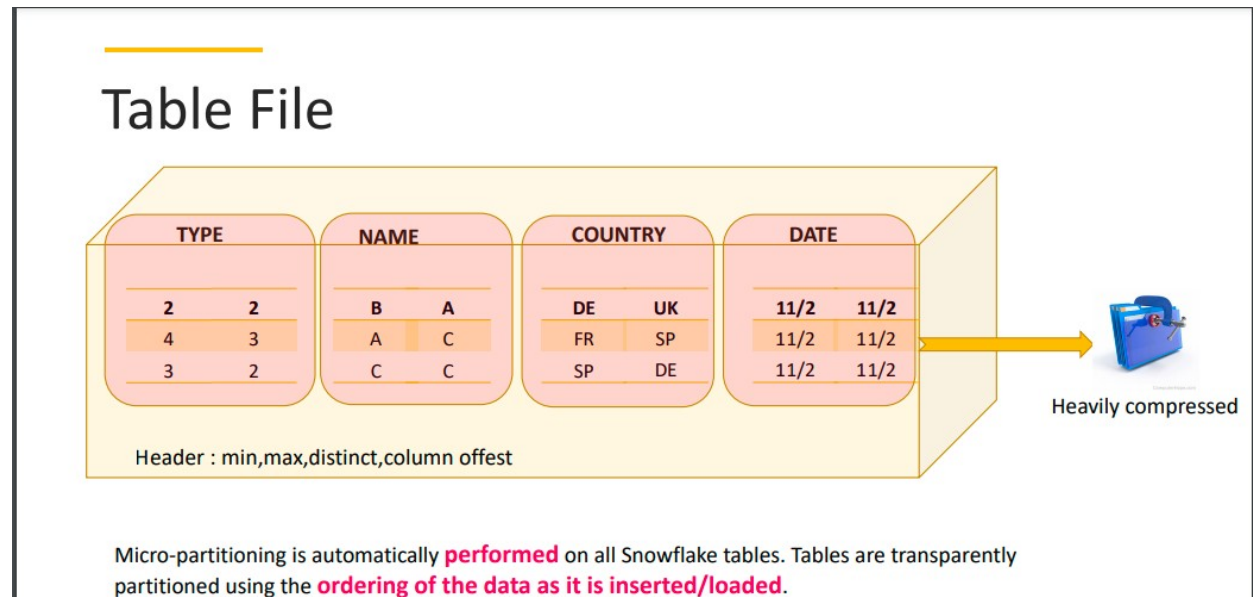
LESSONS LEARNED

- You always need a virtual warehouse to execute queries.
- Always use limit clause with `select * from` queries.
- During development activity always keep auto suspend time limit to a higher value at least 15 mins.
- Share your virtual warehouse when group of users are working on the common tables.
- Never disable your cloud service layer result cache.
- Reusing query result in snowflake is free.

5. Clustering in snowflake?

Clustering: Clustering is to co-locate similar rows in the same micro-partitions.

In snowflake the table file data will be stored as micro partitions in database storage. If we apply cluster after storage data the cluster will be happen so processing cost will be there. Better to add order by clause while loading data into table.



Select Name, Country From employee Where date='11/2'

This query will get submitted to cloud services layer.

Cloud service layer will do query optimization, creates execution plan and submits this plan to all virtual warehouse nodes.

Nodes will first download table file HEADER from all table files.

Based on the metadata information in the HEADER file, table files (Micro partitions) will be scanned.

```

Select
Name,
Country
From employee
Where date='11/2'

```

TYPE	NAME	COUNTRY	DATE
2 A	UK		11/2
4 C	SP		11/2
3 C	DE		11/2
2 B	DE		11/2
3 A	FR		11/2
2 C	SP		11/2

TYPE	NAME	COUNTRY	DATE
3 Z	DE		11/2
2 B	UK		11/2
4 C	NL		11/2
5 X	FR		11/3
1 A	NL		11/3
5 A	FR		11/3

TYPE	NAME	COUNTRY	DATE
2 X	FR		11/2
4 Z	NL		11/2
2 Y	SP		11/2
1 B	SP		11/3
5 X	DE		11/3
3 A	UK		11/4

Virtual warehouse will request AWS s3(data storage layer in our case) to download only this part of the files or range of file.

/ CREATE TABLE WITH CLUSTERING **/**

CREATE TABLE EMPLOYEE (TYPE,NAME,COUNTRY,DATE) CLUSTER BY (DATE);

/ IF YOU HAVE ALREADY LOADED DATA **/**

ALTER TABLE EMPLOYEE CLUSTER BY (DATE);

```

Select
Name,
Country
From employee
Where date='11/2'

```

TYPE	NAME	COUNTRY	DATE
2 A	UK		11/2
4 C	SP		11/2
3 C	DE		11/2
2 B	DE		11/2
3 A	FR		11/2
2 C	SP		11/2

TYPE	NAME	COUNTRY	DATE
3 Z	DE		11/2
2 B	UK		11/2
4 C	NL		11/2
5 X	FR		11/3
1 A	NL		11/3
5 A	FR		11/3

TYPE	NAME	COUNTRY	DATE
2 X	FR		11/2
4 Z	NL		11/2
2 Y	SP		11/2
1 B	SP		11/3
5 X	DE		11/3
3 A	UK		11/4

TYPE	NAME	COUNTRY	DATE
1 C	FR		11/3
4 Z	NL		11/4
5 Y	SP		11/4
5 B	DE		11/5
3 X	DE		11/5
2 Z	UK		11/5

Selected

Pruned

Activate Windows

File 1	File 2	File 3	File 4
TYPE	NAME	COUNTRY	DATE
2	A	UK	11/2
4	C	SP	11/2
3	C	DE	11/2
2	B	DE	11/2
3	A	FR	11/2
2	C	SP	11/2

Each column will be stored as file.

Columns are stored independently within micro partition this is also called as columnar storage.

HOW TO CHOOSE CLUSTERING KEY

- Columns which are more often used in where clause.
- Columns which are more often used in join conditions.
- ```
SELECT A.EMP_DEPT, B. EMP_NAME
FROM EMP B, DEPT A
WHERE B.DEPT_ID = A.DEPT_ID
```
- Order you specify clustering key is important. As general rule, Snowflake recommends ordering the columns from lowest cardinality to highest cardinality.

**/\*\*\*\*\* Cardinality of columns \*\*\*\*\*/**

```
SELECT DISTINCT C_ADDRESS FROM
SAMPLE_DATABASE.PUBLIC.CUSTOMER_NOCLUSTER CLUSTER
SELECT 238609294/1500000000
SELECT DISTINCT C_MKTSEGMENT FROM
SAMPLE_DATABASE.PUBLIC.CUSTOMER_NOCLUSTER CLUSTER
```



```
SELECT 5/1500000000
```

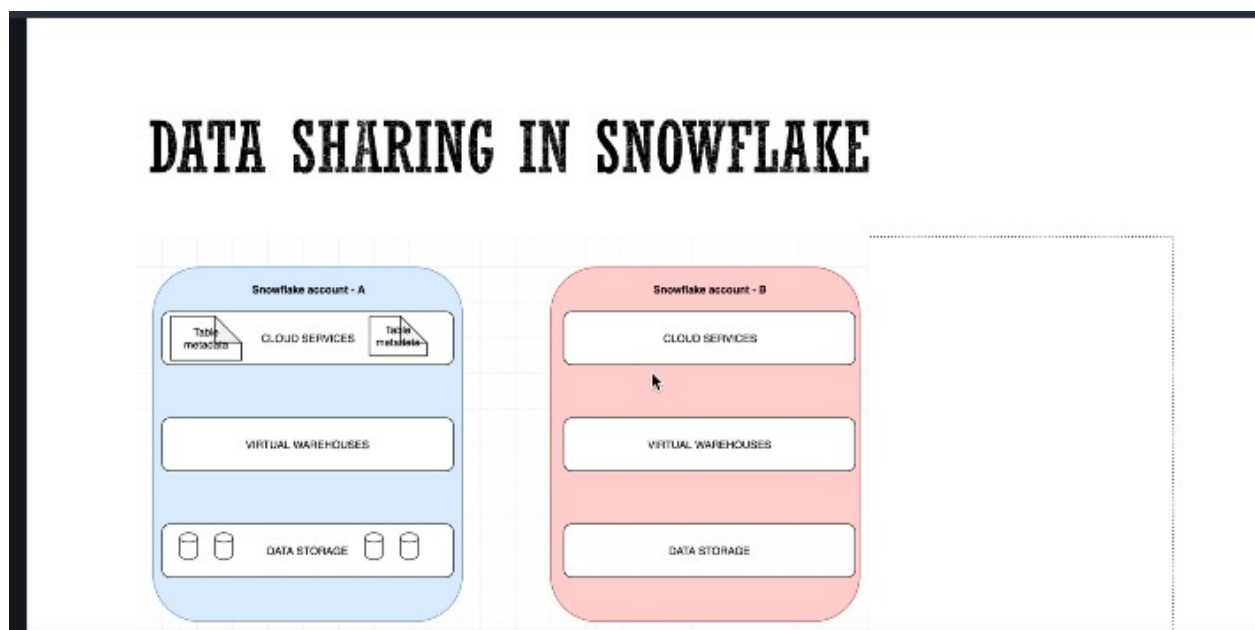
```
/*** Improve performance without applying clustering *****/
```

```
CREATE OR REPLACE TRANSIENT TABLE
DEMO_DB.PUBLIC.CUSTOMER_ORDERBY
```

```
AS
```

```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10000.CUSTOMER
ORDER BY C_MKTSEGMENT
```

## 6. Data sharing in snowflake (Reader Account)?



Data Sharing enables us sharing the selected objects in a database in your account with other Snowflake accounts.

Here we are going to share data table metadata from one snowflake account to other snowflake account.

The consumer going connect to our cloud service for accessing the table's data using their VWH.

**Note:** Here the data is not copying from one account to another account just sharing data only with cloud service metadata.

**Providers:** A data provider is any Snowflake account that **creates shares** and makes them available to other Snowflake accounts to consume

**Consumers:** A data consumer is any account that chooses to create a database from a share made available by a data provider

**Reader Account:** Share data with a consumer who does not already have a Snowflake account and/or is not ready to become a licensed Snowflake customer

- **Sharing Data**
  - Web UI – Inbound , Outbound
- **Outbound Shares (Providers)**
  - Shares that you have created and you have access to
  - Create a Share
  - Edit Shares
- **Inbound Shares (Consumers)**
  - View Shares from providers that you are granted to
  - Create a Database for the share
- **Sharing Considerations**
  - Currently, consumer accounts must be in the same Snowflake Region as your account; i.e. you can only share with other accounts in your Snowflake Region.
  - A share can include data from multiple databases.
  - For data security and privacy reasons, only secure views are supported in shares at this time. If a standard view is added to a share, Snowflake returns an error.
  - Adding accounts to a share immediately makes the share available to consume by the accounts.

- New and modified rows in tables in a share (or in tables referenced by a view in a share) are available immediately to all consumers who have created a database from the share. Keep this in mind when updating these tables.
- A new object created in a database in a share is not automatically available to consumers.
- **General Limitations for Shared Databases**
  - Shared databases are read-only. Users in a consumer account can view/query data, but cannot insert or update data, or create any objects in the database.
  - The following actions are not supported:
    - Creating a clone of a shared database or any schemas/tables in the database.
    - Time Travel for a shared database or any schemas/tables in the database.
    - Editing the comments for a shared database

### Share DDLs

- Create Share
- Alter Share
- Drop Share
- Describe Share
- Show Shares
- Grant <Privileges> to Share
- Revoke <Privileges> to Share
- Show Grants to Share
- Show Grants of Share

**create database reader\_sales;**

**create table customer as**

**select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1000.CUSTOMER;**

```
create share sales_s;
```

```
grant usage on database reader_sales to share sales_s;
```

```
grant usage on schema reader_sales.public to share sales_s;
```

```
grant select on table reader_sales.public.customer to share sales_s;
```

```
grant usage on database reader_sales to role public;
```

```
grant usage on schema reader_sales.public to role public;
```

```
grant select on table reader_sales.public.customer to role public;
```

```
create or replace secure view sales.public.customer_data
```

```
as select C_NAME, C_MKTSEGMENT, C_ACCTBAL
```

```
from sales.public.customer;
```

```
create database reader_sales;
```

```
create table customer as
```

```
select * from SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.CUSTOMER;
```

## 7. Time travel & Fail safe in snowflake?

**Time travel:** SF Time Travel Enables access to Historical Data ( Modified or Dropped / Deleted).

- Objects can be restored which were deleted or dropped

**Data Retention Period:** The Standard Retention Period is 1 day / 24 Hours

--- Set retention time

```
create or replace table emp_retention (
```

```
 first_name varchar(10) ,
```



```

 last_name varchar(10) ,
 email varchar(10) ,
 streetaddress varchar(100) ,
 city varchar(100) ,
 start_date string
);

copy into emp_retention
from @my_s3_stage
file_format = (type = csv field_optionally_enclosed_by='')
pattern = '.*employees0[1-5].csv'
ON_ERROR='CONTINUE'

-- Check retention period.

SHOW TABLES LIKE 'emp' in sfnowflake_tutorial.public

-- Retention period is unset.

ALTER TABLE emp_retention SET DATA_RETENTION_TIME_IN_DAYS = 0;

```

Examples :

```

select * from my_table at(timestamp => 'Mon, 01 May 2015 16:20:00 -
0700'::timestamp);

select * from my_table at(offset => -60*5);

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

create table restored_table clone my_table at(timestamp => 'Mon, 09 May 2015
01:01:00 +0300'::timestamp);

create schema restored_schema clone my_schema at(offset => -3600);

create database restored_db clone my_db before(statement => '8e5d0ca9-005e-
44e6-b858-a8f5b37c5726')

```

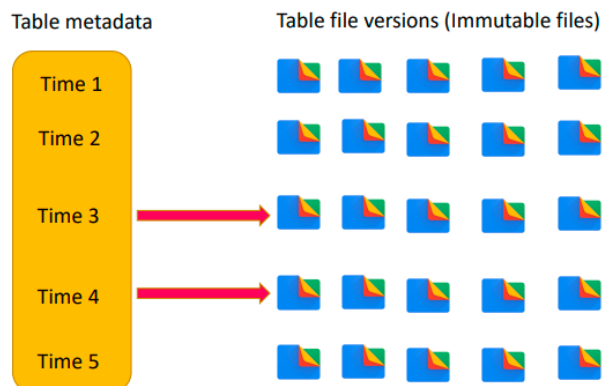
# How time travel works

- 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 requests for parts (ranges) of a file.

## TIME TRAVEL IN SNOWFLAKE

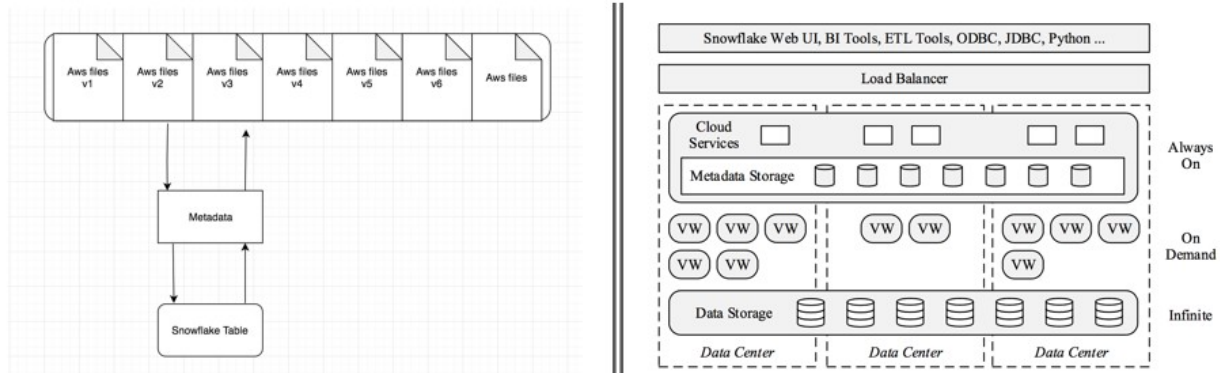


## TIME TRAVEL IN SNOWFLAKE



SF just remove the metadata points. Once we did time travel again SF will attached the those particle files.

## File versioning and metastore



## TIME TRAVEL

- Create backup table by travelling to older version of table.
- Truncate production table.
- Insert data from backup table to production table.

### Fail Safe:

## Continuous Data Protection Lifecycle

**Standard operations allowed:**  
Queries, DDL, DML, etc.

**Time Travel allowed:**  
SELECT ... AT|BEFORE ...  
CLONE ... AT|BEFORE ...  
UNDROP ...


**No user operations allowed**  
(data recoverable only by  
Snowflake)



Fail safe provides 7-day period during which historical data is recoverable by Snowflake.

Fail-safe is not provided access us to get historical data after the Time Travel retention period has ended.

It is for use only by Snowflake to recover data that may have been lost or damaged due to extreme operational failures

| SYNTAX                                                                                             | TABLE TYPE | RETENTION PERIOD | FAIL SAFE |
|----------------------------------------------------------------------------------------------------|------------|------------------|-----------|
| CREATE TABLE EMP  | PERMANENT  | YES              | YES       |
| CREATE TRANSIENT TABLE EMP                                                                         | TRANSIENT  | YES              | NO        |
| CREATE TEMPORARY TABLE EMP                                                                         | TEMPORARY  | NO               | NO        |

### Why Fail Safe Instead of Backup

- Data corruption or loss can happen with any database management
- Time required to reload lost data for us.
- Fail-safe provides an efficient and cost-effective alternative to backup that eliminates the remaining risk

## 8. Stages & Types of Stages and Creation of Stages?

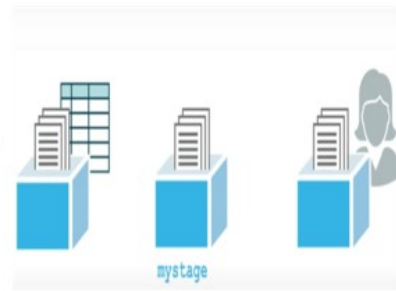
**Stage:** Staging area in snowflake is a blob storage area where you load all your raw files before loading them into snowflake database.



# Snowflake Stages

## • Stages

- Table Stage
- Named Stage
  - Permanent
  - Temporary
- User Stage
- External



# Snowflake Stages

## Table Stage

- Each table has a Snowflake stage allocated to it by default for storing files
- This stage is a convenient option if your files need to be accessible to multiple users and only need to be copied into a single table

## User Stage

- Each User has a stage allocated by default for storing Files
- All the worksheets are stored in this stage
- Will be accessed by single user
- Will be useful if the same file needs to be copied to multiple tables
- No Setting of File Format Options

## Internal Named Stage

- Internal stages are named database objects that provide the greatest degree of flexibility for data loading
- Users with the appropriate privileges on the stage can load data into any table.
- Explicitly grant privileges on the stage to one or more roles before users with those roles can use the stage.
- Internal Temporary Stage – Stays for 24 Hours

## External Stage

- Stage to connect to AWS S3, Azure Blob, GCP Storage

## Viewing Stages



User Stage

List @~;



Table Stage

List @%mytable



Internal Named Stage

List @my\_stage\_name



External Stage

List @external\_stage

# STAGES IN SNOWFLAKE

- What if I don't have subscription to these cloud storage areas ?



# STAGES IN SNOWFLAKE

- What if I don't have subscription to these cloud storage areas ?



Internal staging area

- Remember this is also a blob storage, which is managed by snowflake.

## -- Create external stage

```
create or replace stage control_db.external_stages.my_ext_stage
url='s3://snowflake067/test/'

credentials=(aws_key_id='AKIAUIIPUVJBJMSPABKO'
aws_secret_key='bgQb6b816dzQdGkT+JPVqeiQ561B');

DESC STAGE control_db.external_stages.my_ext_stage

alter stage control_db.external_stages.my_ext_stage set
credentials=(aws_key_id='d4c3b2a1' aws_secret_key='z9y8x7w6');
```

## -- Create internal stage

```
create or replace stage control_db.internal_stages.my_int_stage

DESC STAGE control_db.internal_stages.my_int_stage
```

## 9. File formats & File types and Options (Force, Purge, etc...)?

Create file format:

```
create or replace file format control_db.file_formats.my_csv_format

type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
empty_field_as_null = true compression = gzip;
```

| FILE FORMAT OBJECT             |               |                  |                  | STAGE OBJECT       |                                |               |                   |                   |
|--------------------------------|---------------|------------------|------------------|--------------------|--------------------------------|---------------|-------------------|-------------------|
| property                       | property_type | property_value   | property_default | parent_property    | property                       | property_type | property_value    | property_default  |
| TYPE                           | String        | CSV**            | CSV**            | STAGE_FILE_FORMAT  | TYPE                           | String        | CSV**             | CSV**             |
| RECORD_DELIMITER               | String        | \n**             | \n**             | STAGE_FILE_FORMAT  | RECORD_DELIMITER               | String        | \n**              | \n**              |
| FIELD_DELIMITER                | String        | **               | **               | STAGE_FILE_FORMAT  | FIELD_DELIMITER                | String        | **                | **                |
| FILE_EXTENSION                 | String        | .                | .                | STAGE_FILE_FORMAT  | FILE_EXTENSION                 | String        | .                 | .                 |
| SKIP_HEADER                    | Integer       | 1                | 0                | STAGE_FILE_FORMAT  | SKIP_HEADER                    | Integer       | 0                 | 0                 |
| DATE_FORMAT                    | String        | AUTO**           | AUTO**           | STAGE_FILE_FORMAT  | DATE_FORMAT                    | String        | AUTO**            | AUTO**            |
| TIME_FORMAT                    | String        | AUTO**           | AUTO**           | STAGE_FILE_FORMAT  | TIME_FORMAT                    | String        | AUTO**            | AUTO**            |
| TIMESTAMP_FORMAT               | String        | AUTO**           | AUTO**           | STAGE_FILE_FORMAT  | TIMESTAMP_FORMAT               | String        | AUTO**            | AUTO**            |
| BINARY_FORMAT                  | String        | HEX**            | HEX**            | STAGE_FILE_FORMAT  | BINARY_FORMAT                  | String        | HEX**             | HEX**             |
| ESCAPE                         | String        | NONE**           | NONE**           | STAGE_FILE_FORMAT  | ESCAPE                         | String        | NONE**            | NONE**            |
| ESCAPE_UNENCLOSED_FIELD        | String        | \\**             | \\**             | STAGE_FILE_FORMAT  | ESCAPE_UNENCLOSED_FIELD        | String        | \\**              | \\**              |
| TRIM_SPACE                     | Boolean       | FALSE            | FALSE            | STAGE_FILE_FORMAT  | TRIM_SPACE                     | Boolean       | FALSE             | FALSE             |
| FIELD_OPTIONALLY_ENCLOSED_BY   | String        | NONE**           | NONE**           | STAGE_FILE_FORMAT  | FIELD_OPTIONALLY_ENCLOSED_BY   | String        | NONE**            | NONE**            |
| NULL_IF                        | List          | ["NULL", "null"] | ["\\N", "N"]     | STAGE_FILE_FORMAT  | NULL_IF                        | List          | ["\\N", "N"]      | ["\\N", "N"]      |
| COMPRESSION                    | String        | gzip**           | AUTO**           | STAGE_FILE_FORMAT  | COMPRESSION                    | String        | AUTO**            | AUTO**            |
| ERROR_ON_COLUMN_COUNT_MISMATCH | Boolean       | TRUE             | TRUE             | STAGE_FILE_FORMAT  | ERROR_ON_COLUMN_COUNT_MISMATCH | Boolean       | TRUE              | TRUE              |
| VALIDATE_UTF8                  | Boolean       | TRUE             | TRUE             | STAGE_FILE_FORMAT  | VALIDATE_UTF8                  | Boolean       | TRUE              | TRUE              |
| SKIP_BLANK_LINES               | Boolean       | FALSE            | FALSE            | STAGE_FILE_FORMAT  | SKIP_BLANK_LINES               | Boolean       | FALSE             | FALSE             |
| REPLACE_INVALID_CHARACTERS     | Boolean       | FALSE            | FALSE            | STAGE_FILE_FORMAT  | REPLACE_INVALID_CHARACTERS     | Boolean       | FALSE             | FALSE             |
| EMPTY_FIELD_AS_NULL            | Boolean       | TRUE             | TRUE             | STAGE_FILE_FORMAT  | EMPTY_FIELD_AS_NULL            | Boolean       | TRUE              | TRUE              |
| SKIP_BYTE_ORDER_MARK           | Boolean       | TRUE             | TRUE             | STAGE_FILE_FORMAT  | SKIP_BYTE_ORDER_MARK           | Boolean       | TRUE              | TRUE              |
| ENCODING                       | String        | UTF8**           | UTF8**           | STAGE_FILE_FORMAT  | ENCODING                       | String        | UTF8**            | UTF8**            |
|                                |               |                  |                  | STAGE_COPY_OPTIONS | ON_ERROR                       | String        | ABORT_STATEMENT** | ABORT_STATEMENT** |
|                                |               |                  |                  | STAGE_COPY_OPTIONS | SIZE_LIMIT                     | Long          |                   |                   |
|                                |               |                  |                  | STAGE_COPY_OPTIONS | PURGE                          | Boolean       | FALSE             | FALSE             |
|                                |               |                  |                  | STAGE_COPY_OPTIONS | RETURN_FAILED_ONLY             | Boolean       | FALSE             | FALSE             |
|                                |               |                  |                  | STAGE_COPY_OPTIONS | ENFORCE_LENGTH                 | Boolean       | TRUE              | TRUE              |

**Types and Options:** Can refer below copy command topic.

1. Purge
2. Overwrite
3. Force

## 10. Database creation & Tables & Types of tables?

**Database:**

create or replace database reader\_sales;

create or replace transient database control\_db;

| SYNTAX                     | TABLE TYPE | RETENTION PERIOD | FAIL SAFE |
|----------------------------|------------|------------------|-----------|
| CREATE TABLE EMP           | PERMANENT  | YES              | YES       |
| CREATE TRANSIENT TABLE EMP | TRANSIENT  | YES              | NO        |
| CREATE TEMPORARY TABLE EMP | TEMPORARY  | NO               | NO        |

**Permanent Table:**

CREATE OR REPLACE TABLE READER\_SALES.PUBLIC.CUSTOMER

AS

SELECT \* FROM SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10000.CUSTOMER;

**TRANSIENT:**

CREATE OR REPLACE TRANSIENT TABLE READER\_SALES.PUBLIC.CUSTOMER



AS

```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10000.CUSTOMER;
```

TEMPORARY:

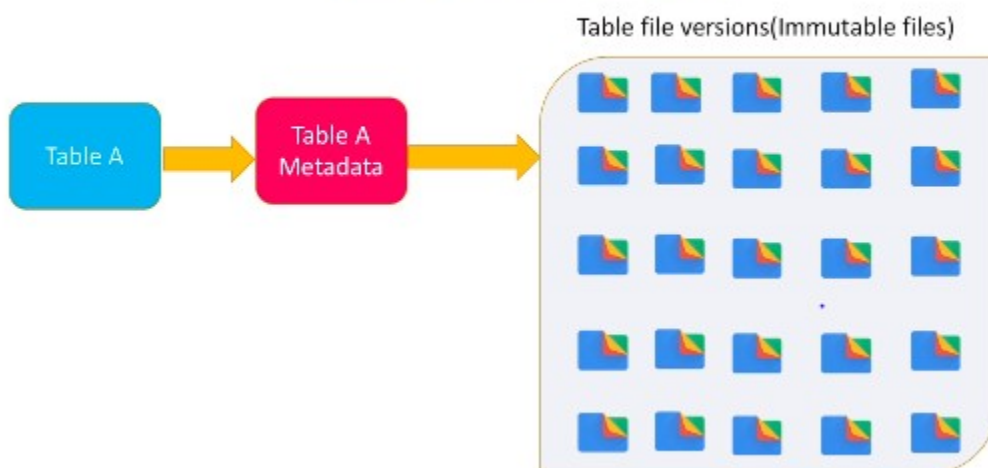
```
CREATE OR REPLACE TEMPORARY TABLE
READER_SALES.PUBLIC.CUSTOMER_TEMP
```

AS

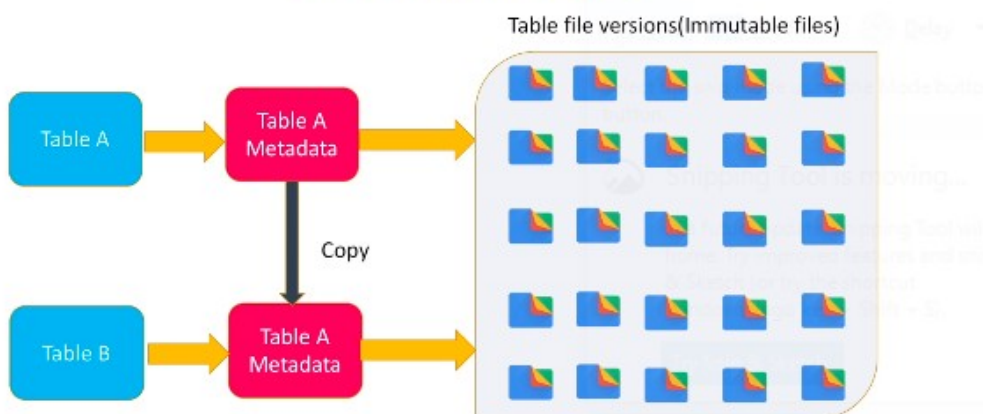
```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10000.CUSTOMER;
```

### 11. Clone & Swap creation in snowflake ?

## CLONE IN SNOWFLAKE



## CLONE IN SNOWFLAKE

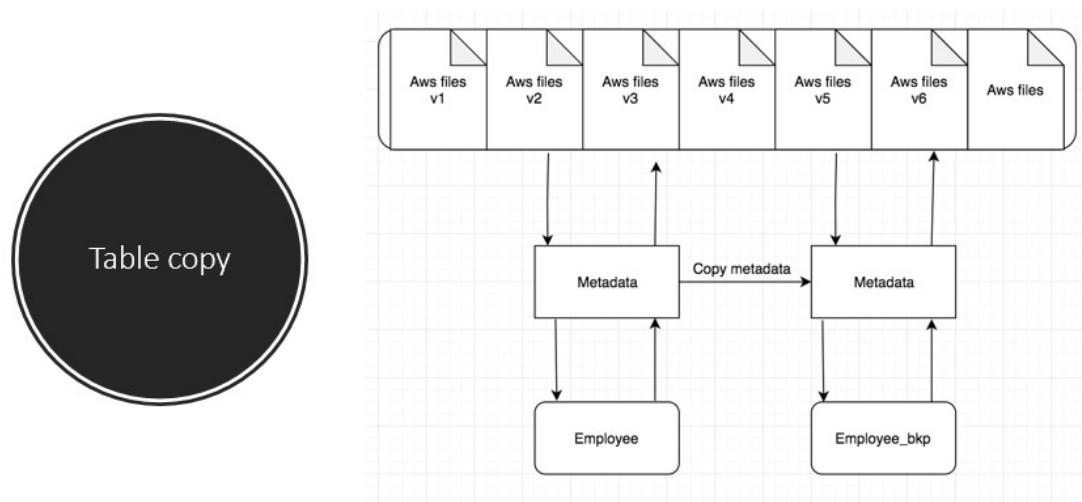


**Clone:** Which means taking metadata copy of the table. Then new copy of the metadata pointing to same storage in backend. But after cloning the tables will be independently each other.

If add any data into any tables, the files will be added in separately (S3 storage).

## CLONE IN SNOWFLAKE

- You are not making a new copy of storage. You are only making a copy of metadata



-- CLONE TABLE

```
CREATE OR REPLACE TRANSIENT TABLE EMP_CLONE CLONE EMP;
```

```
SELECT * FROM EMP_CLONE;
```

-- Make changes in EMP\_CLONE it should not affect EMP Table.

```
DELETE FROM EMP_CLONE WHERE FIRST_NAME='Arlene';
```

```
SELECT * FROM EMP_CLONE WHERE FIRST_NAME='Violette';
```

---Checking clone id's,(ID and clone id will be same EMP, But clone id will same as emp but ID will be unique).

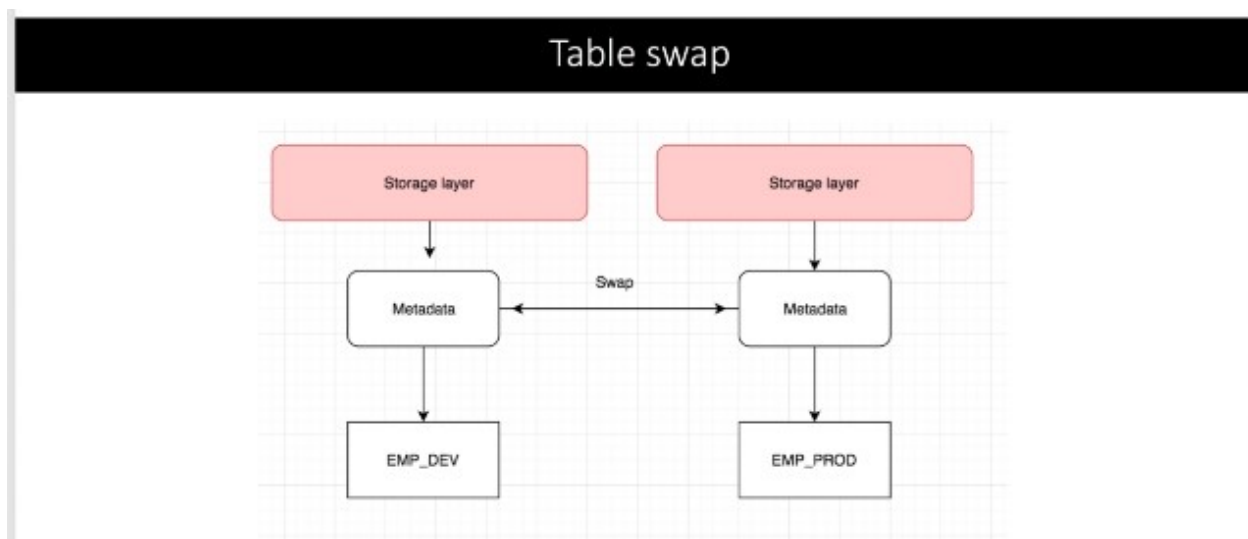
```
SELECT * FROM INFORMATION_SCHEMA.TABLE_STORAGE_METRICS WHERE
TABLE_NAME LIKE 'EMP'
```

```
AND TABLE_CATALOG='SFNOWFLAKE_TUTORIAL' AND TABLE_DROPPED IS NULL;
```

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_STORAGE_METRICS WHERE
TABLE_NAME LIKE 'EMP_CLONE'
```

```
AND TABLE_CATALOG='SFNOWFLAKE_TUTORIAL' AND TABLE_DROPPED IS NULL;
```

**Swap:** Swapping the one table metadata to another table. Once swap emp\_dev to Emp\_prod, then emp\_dev data will not there. The records will be there in EMP\_Prod.



#### ---- Table swap

```
create or replace table emp_dev (
```

```
 first_name string ,
```

```
 last_name string ,
```

```
 email string ,
```

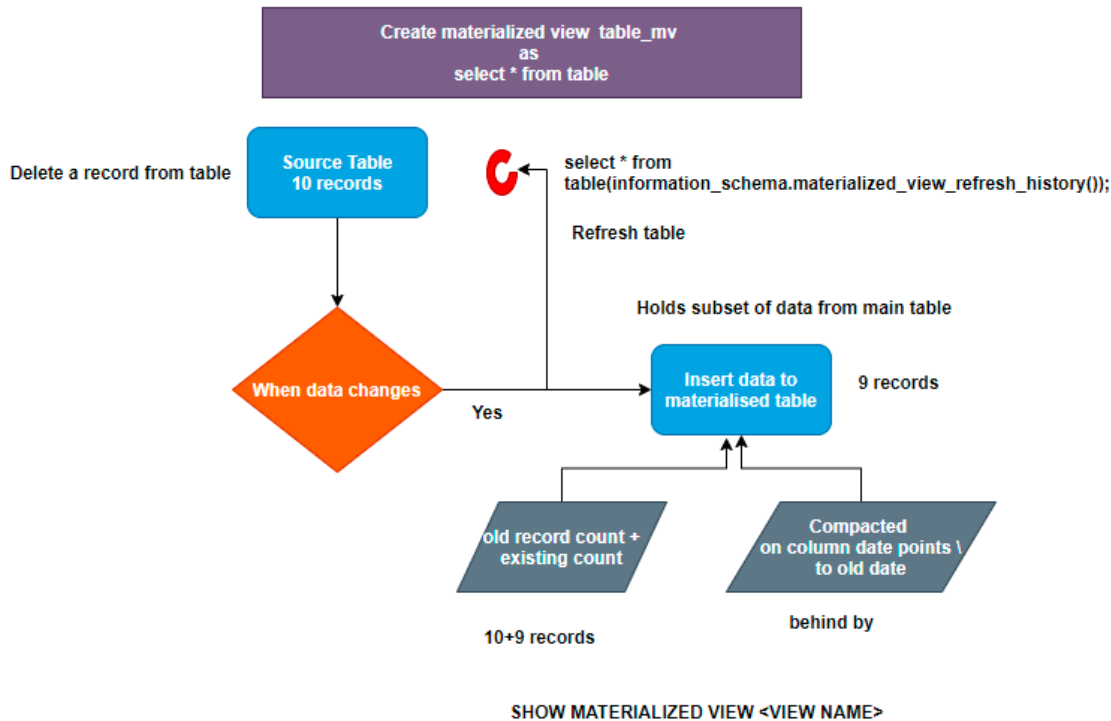
```
 streetaddress string ,
```

```
 city string ,
 start_date date
);
copy into emp_dev
from @my_s3_stage
file_format = (type = csv field_optionally_enclosed_by='')
pattern = '.*employees0[1-5].csv'
ON_ERROR='CONTINUE'
create or replace table emp_prod (
 first_name string ,
 last_name string ,
 email string ,
 streetaddress string ,
 city string ,
 start_date date
);
copy into emp_prod
from @my_s3_stage
file_format = (type = csv field_optionally_enclosed_by='')
pattern = '.*employees0[1-2].csv'
ON_ERROR='CONTINUE'
ALTER TABLE emp_prod SWAP WITH emp_dev;
select * from emp_prod;
select * from emp_dev;
```

## 12. Views & Types of views and MV maintenance cost and Limitations?

- **Views**

- Regular View - Non-Materialized Views
- Secured Views
- Materialized Views



## MATERIALIZED VIEWS

- Wikipedia defines a materialized view as “a database object that contains the **results of a query**”.
- Unlike a view, it’s not a window into a database.
- Rather, it is a separate object holding **query results** with data refreshed periodically.

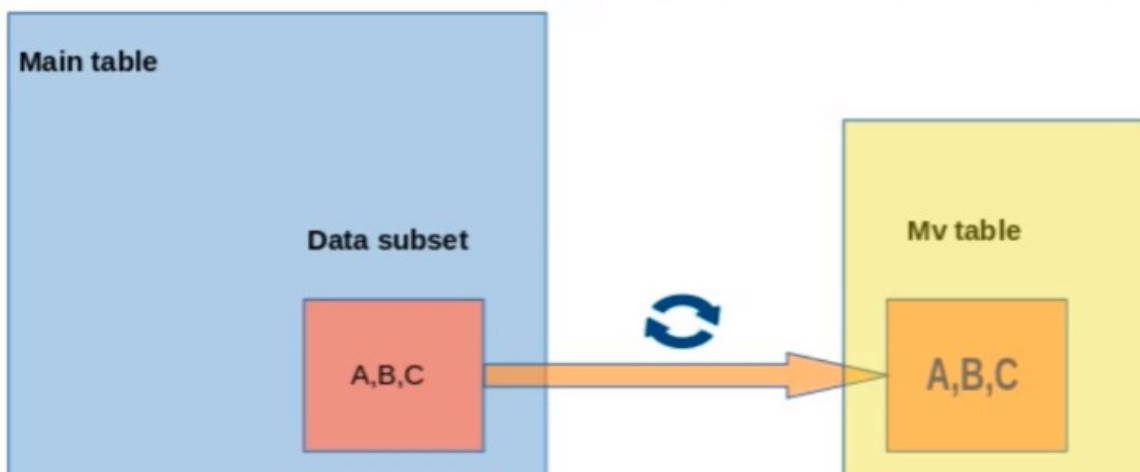
## Properties of snowflake MV

- 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.**

## Properties of snowflake MV

- Snowflake MVs enhances 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.

## Properties of snowflake MV





# Problem with MV

- Refreshing data periodically can lead to inconsistent or out-of-date results when you access Mvs.
- Data manipulation language (DML) operations (for adding, deleting, and modifying data) traditionally experienced slow-downs when they used MVs

**/\*\*\*\*\* How data might get exposed in normal view \*\*\*\*\*/**

```
create or replace view sales.public.customer_data_normal
as select C_NAME, C_MKTSEGMENT, C_ACCTBAL
from sales.public.customer where c_mktsegment='AUTOMOBILE';
```

**/\*\*\*\*\*Secure view normal view difference \*\*\*\*\*/**

```
create or replace secure view sales.public.customer_data_secure
as select C_NAME, C_MKTSEGMENT, C_ACCTBAL
from sales.public.customer;
```

**/\*\*\*\*\*materialize view\*\*\*\*\*/**

```
create or replace materialized view call_center_M_view
as
select * from
CALL_CENTER;

select * from table(information_schema.materialized_view_refresh_history());

SHOW MATERIALIZED VIEWS LIKE 'call_center_M_view' -- -- observe behind_by
column
```

**select 60+59 =119**

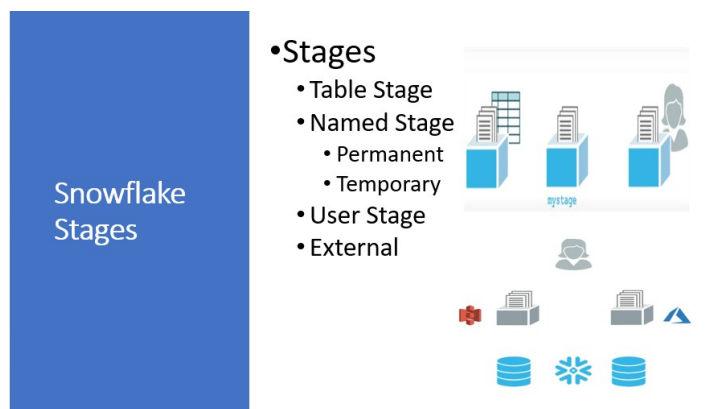
**select \* from call\_center\_M\_view -- still 59 records**

**select \* from**

**table(information\_schema.materialized\_view\_refresh\_history());--- It will start to refresh**

### **13. Copy command & Options and Data loading (Local file system, Could)?**

**COPY:** Copy command is using for load the files of stage data into existing table in SK.



```
-- Stages
copy into mytable from '@mystage/path 1/file 1.csv';
copy into mytable from '@%mytable/path 1/file 1.csv';
copy into mytable from '@~/path 1/file 1.csv';

-- S3 bucket
copy into mytable from 's3://mybucket 1/prefix 1/file 1.csv';

-- Azure container
copy into mytable from 'azure://myaccount.blob.core.windows.net/mycontainer/encrypted_files/file 1.csv';
```

## COPY OPTIONS

- VALIDATE\_MODE
- LOAD SELECTED FILES (FILES)
- OR\_ERROR
- ENFORCE\_LENGTH AND , TRUNCATECOLUMNS
- FORCE
- PURGE
- LOAD HISTORY VIEW AND COPY HISTORY FUNCTION.

## VALIDATE MODE

- VALIDATION\_MODE = RETURN\_n\_ROWS | RETURN\_ERRORS | RETURN\_ALL\_ERRORS
- 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

-- Create table

create or replace table emp (

first\_name string ,

last\_name string ,

email string ,

streetaddress string ,

city string ,

start\_date date

);

-- We will be copying the data where we have error values for date columns.

copy into emp

from @my\_s3\_stage

file\_format = (type = csv field\_optionally\_enclosed\_by='')

--pattern = '\*.employees0[1-5].csv'

```
validation_mode = 'RETURN_ERRORS';
```

-- Recreate the table with start\_date as string.

```
create or replace table emp (
```

```
 first_name string ,
```

```
 last_name string ,
```

```
 email string ,
```

```
 streetaddress string ,
```

```
 city string ,
```

```
 start_date string
```

```
);
```

-- We will change the data type of date column and try to load it again.

```
copy into emp
```

```
from @my_s3_stage
```

```
file_format = (type = csv field_optionally_enclosed_by='')
```

```
--pattern = '.*employees0[1-5].csv'
```

```
validation_mode = 'RETURN_ERRORS';
```

## LESSONS LEARNED

- VALIDATION\_MODE will not load data into table.
- Define proper DDL while creating tables.

File Options:

**-- Create emp table.**

```
create or replace table emp (
 first_name string ,
 last_name string ,
 email string ,
 streetaddress string ,
 city string ,
 start_date date
```

```
);
```

**-- Copy data using filter condition on file name.**

```
copy into emp
```

```
from @my_s3_stage
```

```
file_format = (type = csv field_optionally_enclosed_by='')
```

```
pattern = '.*employees0[1-5].csv'
```

```
-- ON_ERROR='CONTINUE'
```

```
files=('employees01.csv','employees_error_file0.csv','employees_error_file1.csv'
)
```

```
desc stage my_s3_stage;
```

**-- Remove error files and load data**

```
copy into emp
```

```
from @my_s3_stage
```

```
file_format = (type = csv field_optionally_enclosed_by='')
```

```
pattern = '.*employees0[1-5].csv'
```

```
-- ON_ERROR='CONTINUE'

files=('employees01.csv','employees_error_file0.csv','employees_error_file1.csv'
')

copy into emp

from @my_s3_stage

file_format = (type = csv field_optionally_enclosed_by='')

pattern = '.*employees0[1-5].csv'

-- ON_ERROR='CONTINUE'

files!

=('employees01.csv','employees_error_file0.csv','employees_error_file1.csv')
```

## LESSONS LEARNED

- Using FILE option you can load specific files into snowflake table.
- You can't use negation symbol with FILE option.
- How to use pattern option.

### ON ERROR:

## On Error

- You can easily reject records without failing the copy commands.
- When you are building data pipelines, using this option is must.
- You can collect rejected records in a separate table.

-- Load data into table ignoring rejected records.



```

copy into emp
from @my_s3_stage
file_format = (type = csv field_optionally_enclosed_by='')
--pattern = '.*employees0[1-5].csv'
ON_ERROR='CONTINUE';

-- Check for rejected records.

select * from table(validate(emp, job_id=>'018ff817-0317-f8b7-0000-
e5fd0001551e'));

-- Save rejected records to reprocess in future.

create table copy_rejects
as

select * from table(validate(emp, job_id=>'018ff817-0317-f8b7-0000-
e5fd0001551e'));

-- Lessons learned.

1. How to load ignore records while loading data into table.
2. How to retrieve rejected records.

```

**ENFORCE\_LENGTH & TRUNCATECOLUMNS:** If column values length is more using these option we can load data into table.

**ENFORCE\_LENGTH = TRUE | FALSE**

**TRUNCATECOLUMNS = TRUE | FALSE**

**DESC STAGE MY\_S3\_STAGE;**

**COPY INTO EMP**

**FROM @MY\_S3\_STAGE**

**FILE\_FORMAT = (TYPE = CSV FIELD\_OPTIONALLY\_ENCLOSED\_BY='')**

```
--PATTERN = '.*EMPLOYEES0[1-5].CSV'
```

```
ON_ERROR='CONTINUE'
```

```
--TRUNCATECOLUMNS = TRUE
```

```
ENFORCE_LENGTH = FALSE;
```

**FORCE:** Using this option we can load already loaded file again into table. Without this option we cannot same records again into table bas SK checking records with MD5 hash algorithm.

-- Remove files from staging area.

```
copy into emp
```

```
from @my_s3_stage
```

```
file_format = (type = csv field_optionally_enclosed_by='')
```

```
pattern = '.*employees0[1-5].csv'
```

```
ON_ERROR='CONTINUE'
```

```
FORCE = TRUE;
```

**PURGE:** Once file loaded into table then file will be deleted form stages.

-- Remove files from staging area.

```
copy into emp
```

```
from @my_s3_stage
```

```
file_format = (type = csv field_optionally_enclosed_by='')
```

```
pattern = '.*employees0[1-5].csv'
```

```
ON_ERROR='CONTINUE'
```

```
PURGE = TRUE;
```

**Load History:** Copy command history we can get till 14 days (What are the files loaded by copy command).It will give only 10k records.

## LOAD HISTORY

- This Information Schema view enables you to retrieve the history of data loaded into tables using the `COPY INTO <table>` command. The view displays one row for each file loaded.
- Snowflake retains historical data for `COPY INTO` commands executed within the previous 14 days only.
- This view returns an upper limit of 10,000 rows

```
SELECT * FROM INFORMATION_SCHEMA.LOAD_HISTORY
```

```
ORDER BY LAST_LOAD_TIME DESC;
```

```
SELECT * FROM INFORMATION_SCHEMA.LOAD_HISTORY
```

```
WHERE SCHEMA_NAME='PUBLIC' AND
```

```
TABLE_NAME='EMP';
```

**Copy History:** To overcome Load history command 10k records we can use this copy history option.

```
SELECT *
```

```
FROM TABLE(INFORMATION_SCHEMA.COPY_HISTORY(TABLE_NAME=>'EMP'));
```

```
SELECT *
```

```
FROM TABLE(INFORMATION_SCHEMA.COPY_HISTORY(TABLE_NAME=>'EMP',
START_TIME=> DATEADD(HOURS, -5, CURRENT_TIMESTAMP()))
```

```
WHERE ERROR_COUNT >0;
```

**D/w Load history & Copy history view:**

| A                              | B                         |
|--------------------------------|---------------------------|
| LOAD HISTORY VIEW              | COPY HISTORY VIEW         |
| TABLE ID                       | FILE_NAME                 |
| TABLE NAME                     | STAGE_LOCATION            |
| SCHEMA ID                      | LAST_LOAD_TIME            |
| SCHEMA NAME                    | ROW_COUNT                 |
| CATALOG ID                     | ROW_PARSED                |
| CATALOG NAME                   | FILE_SIZE                 |
| FILE_NAME                      | FIRST_ERROR_MESSAGE       |
| LAST_LOAD_TIME                 | FIRST_ERROR_LINE_NUMBER   |
| STATUS                         | FIRST_ERROR_CHARACTER_POS |
| ROW_COUNT                      | FIRST_ERROR_COLUMN_NAME   |
| ROW_PARSED                     | ERROR_COUNT               |
| FIRST_ERROR_MESSAGE            | ERROR_LIMIT               |
| FIRST_ERROR_LINE_NUMBER        | STATUS                    |
| FIRST_ERROR_CHARACTER_POSITION | TABLE_CATALOG_NAME        |
| FIRST_ERROR_COL_NAME           | TABLE_SCHEMA_NAME         |
| ERROR_COUNT                    | TABLE_NAME                |
| ERROR_LIMIT                    | PIPE_CATALOG_NAME         |
|                                | PIPE_SCHEMA_NAME          |
|                                | PIPE_NAME                 |
|                                | PIPE_RECEIVED_TIME        |

**14. Snow pipe in snowflake?**

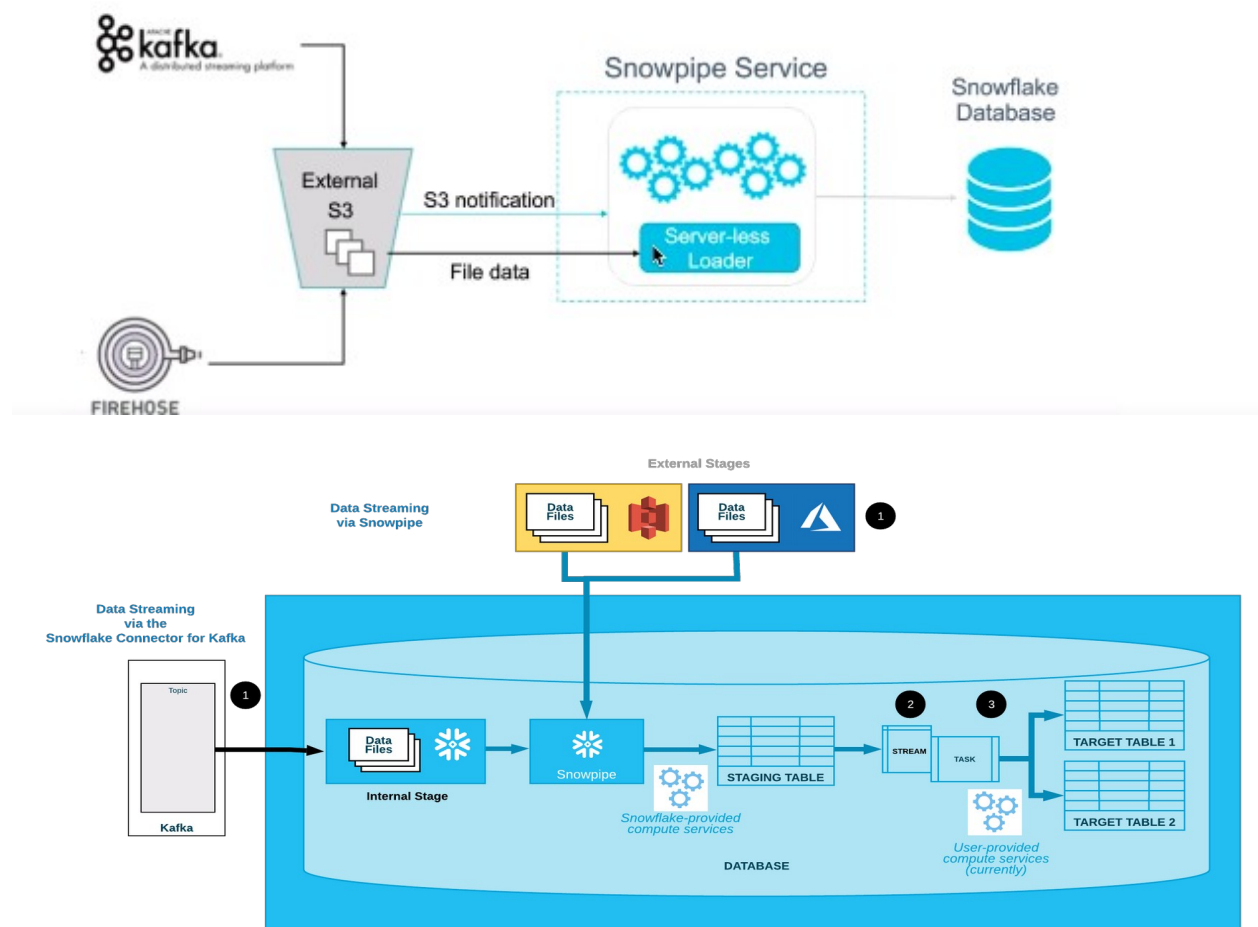
**Snow Pipe:** Continuous Data Loading (stream data) in SK.

Once file loaded in S3 bucket then AWS S3 notify the Snow pipe service then copy command will execute and load data form S3 bucket to Stage tables.

**Amazon SQS(ADD SQS QUEUE ARN):** Amazon SQS is a message queue service used by distributed applications to exchange messages through a polling model, and can be used to decouple sending and receiving components.

**Event:** Need to open bucket properties in S3 and create Event with show pipe SQS(Show pipe pipename;).

## SNOW PIPE



-- Create a pipe to ingest csv data

```
create or replace pipe snowpipe.public.snowpipe auto_ingest=true as
 copy into snowpipe.public.emp_snowpipe
 from @snowpipe.public.snow_stage
 file_format = (type = 'csv');
```

```
alter pipe snowpipe refresh;
show pipes;
```

**Note1:** If there is an error snow pipe will not give you any notification. It will remain silent. It's your responsibility to check for errors.

-- Validate

```
select SYSTEM$PIPE_STATUS('snowpipe');
```

```
select * from table(validate_pipe_load(
 pipe_name=>'DEMO_DB.PUBLIC.snowpipe',
 start_time=>dateadd(hour, -4, current_timestamp())));
```

```
select * from
table(information_schema.copy_history(table_name=>'emp_snowpipe',
start_time=> dateadd(hours, -1, current_timestamp())));
```

```
select * from table(information_schema.query_history()
order by start_time desc;
```

**Note2:** You can't update copy command. You can only recreate pipe. Recreating your pipe will not change your notification channel. Recreating pipe object is not dropping pipe metadata (file sent). Snow pipe will work based on filename and Copy command work based on records (MD5 hash algorithm).

```
create or replace pipe demo_db.public.snowpipe auto_ingest=true as
copy into demo_db.public.emp_snowpipe
from @demo_db.public.snow_stage
file_format = (type = 'csv', FIELD_OPTIONALLY_ENCLOSED_BY='');
```

```
alter pipe snowpipe refresh;
```

```
--arn:aws:sqs:us-east-1:628993367716:sf-snowpipe-AIDAZE4XND2SCZEJXXYBR-
gmNjf-iFPpApjNiM7VwsSQ
```

```
--arn:aws:sqs:us-east-1:628993367716:sf-snowpipe-AIDAZE4XND2SCZEJXXYBR-
gmNjf-iFPpApjNiM7VwsSQ
```

**Integration:** The AWS & Sreat key can integrate instead of giving directly in copy command, Because others can see the keys' and we can do changes in future in once place instead of changing copy commands.

```
create or replace storage integration s3_int
type = external_stage
storage_provider = s3
enabled = true
storage_aws_role_arn = 'arn:aws:iam::579834220952:role/snowflake_role'
storage_allowed_locations = ('s3://hartfordstar/');
```

```
alter storage integration s3_int
```



```
set STORAGE_ALLOWED_LOCATIONS =
('s3://hartfordstar/', 's3://hartfordstar/snowpipe/', 's3://hartfordstar/
snowpipe2/');
```

```
create or replace file format my_csv_s3_format
```

```
type = csv field_delimiter = ',' skip_header = 0 null_if = ('NULL', 'null')
```

```
empty_field_as_null = true FIELD_OPTIONALLY_ENCLOSED_BY='\"';
```

```
create or replace stage snow_stage
```

```
storage_integration = s3_int
```

```
url = 's3://hartfordstar/snowpipe'
```

```
file_format = my_csv_s3_format;
```

## LESSONS LEARNED

- You have to validate Snowpipe object in error scenarios.
- You can't capture rejected records when you are executing copy command through snowpipe.

---

## LESSONS LEARNED

- You can't alter the copy command in snowpipe object.
- Don't create multiple pipe objects on same bucket.

## LESSONS LEARNED

- Snowpipe works based on the names of the file.
- Direct execution of copy command will look for hash values.
- Direct execution of copy commands is more reliable if you are doing batch loads.

### 15. Streaming in snowflake & Types of Streaming, SCD's, Change Clause?

**Stream:** Stream is an object which is created on top of the table. It maintains offsets when performing any DML operation on the table. The offsets will be recorded by the stream.

Stream itself does not contain any table data. A stream only stores the offset for the source table and returns CDC (Change Data Capture) records by leveraging the versioning history for the source table.

- **Stream Columns –**
  - METADATA\$ACTION
  - METADATA\$ISUPDATE
  - METADATA\$ROW\_ID
- Data Retention Period and Staleness
  - If a stream is created on a table, the data retention period is 14 days.
  - Additional Storage Charges are applied.

**Types of Stream's:**

## TYPES IN STREAMS

- Standard streams.
- Append only streams.

-- Create a standard stream on the source table.

```
create or replace stream delta_s on table t;
```

-- Create an append-only stream on the source table.

```
create or replace stream append_only_s on table t append_only=true;
```

-- Create consumer table

```
create or replace
```

```
table target_t(id int, name string, stream_type string default null, rec_version
number default 0, REC_DATE TIMESTAMP_LTZ);
```

--consuming the delta\_s table data to Target table t

```
merge into target_t t
```

```
using delta_s s
```

```
on t.id=s.id and (metadata$action='DELETE')
```

```
when matched and metadata$isupdate='FALSE' then update set
rec_version=9999, stream_type='DELETE'
```

```
when matched and metadata$isupdate='TRUE' then update set
rec_version=rec_version-1
```

```
when not matched then insert (id,name,stream_type,rec_version,REC_DATE)
values(s.id, s.name, metadata$action,0,CURRENT_TIMESTAMP())
```

---

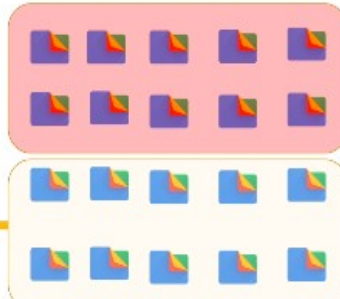
## LESSONS LEARNED

- Using append streams we can only track insert operation on source table.
- Standard stream captures all data change operation on source table.

# STREAMS IN SNOWFLAKE

UPDATE TABLE EMPLOYEE  
WHERE NAME='JOHN'

STREAMS



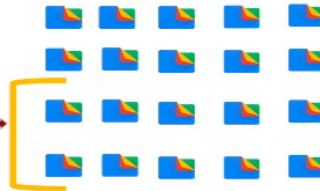
OLD RECORD

NEW RECORD

## STREAMS IN SNOWFLAKE

UPDATE TABLE EMPLOYEE  
WHERE NAME='JOHN'

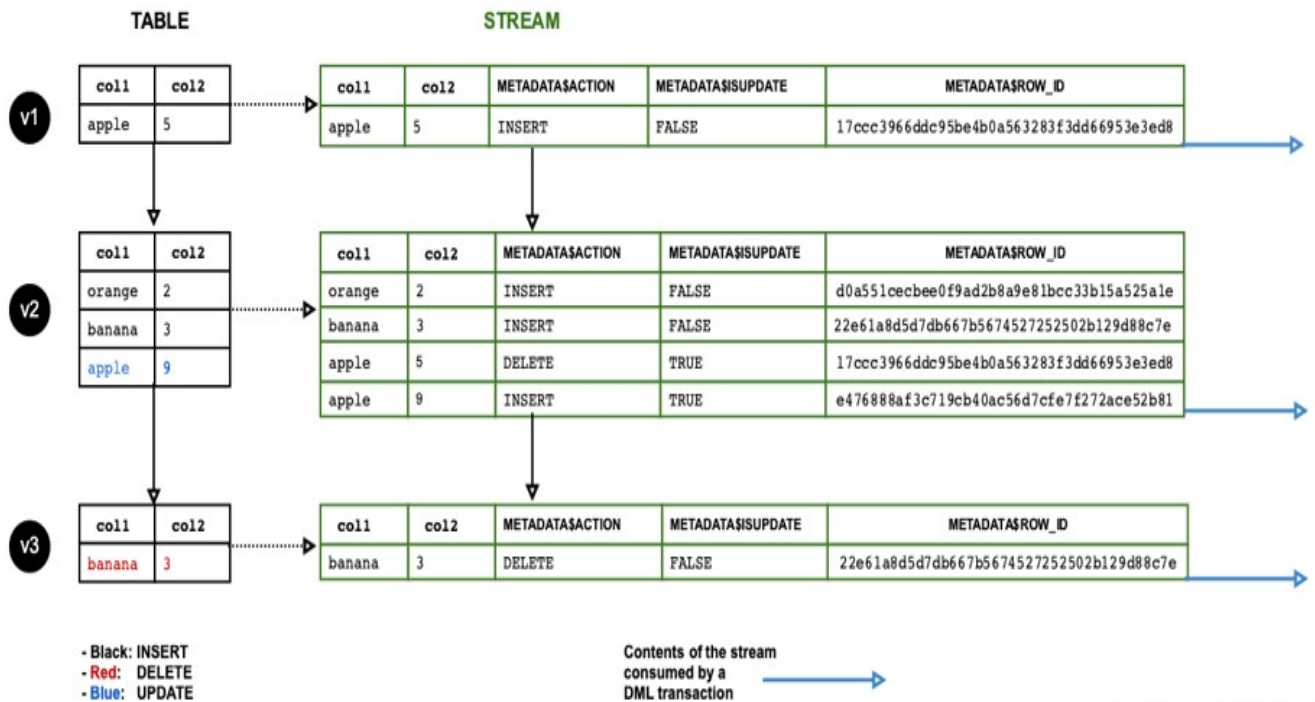
STREAM  
OBJECT



Offset 1(10:51)

Offset 2(10:55)

### • Data Flow



# STREAMS IN SNOWFLAKE

- A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data. LOAD SELECTED FILES (FILES)
- Note that a stream itself does not contain any table data. A stream only stores the offset for the source table and returns CDC records by leveraging the versioning history for the source table.

# STREAMS IN SNOWFLAKE



## LESSONS LEARNED

- Stream object created will point to most recent offset.
- Stream object will maintain offset.
- Offset will increment once the stream is consumed.
- You can create multiple stream object on a table.
- Tie stream with task.

**CHANGE Clause:** when use this changes clause we can track changes on table for particular period like stream.

-- Enable change tracking on the table.

```
alter table t1 set change_tracking = true;
```

-- Initialize a session variable for the current timestamp.

```
set ts1 = (select current_timestamp());
```

-- Query the change tracking metadata in the table during the interval from \$ts1 to the current time.

-- Return the full delta of the changes.

```
select *
from t1
 changes(information => default)
 at(timestamp => $ts1);
```

SCD:

-- updating task

CREATE TASK tgt\_merge

WAREHOUSE = compute\_wh

SCHEDULE = '1 minute'

WHEN

SYSTEM\$STREAM\_HAS\_DATA('delta\_s')

AS

merge into target\_t t

using delta\_s s

on t.id=s.id and (metadata\$action='DELETE')

when matched and metadata\$isupdate='FALSE' then update set  
rec\_version=9999, stream\_type='DELETE'

when matched and metadata\$isupdate='TRUE' then update set  
rec\_version=rec\_version-1

when not matched then insert (id,name,stream\_type,rec\_version,REC\_DATE)  
values(s.id, s.name, metadata\$action,0,CURRENT\_TIMESTAMP() )

**16. Scheduling-TASKS in snowflake?**



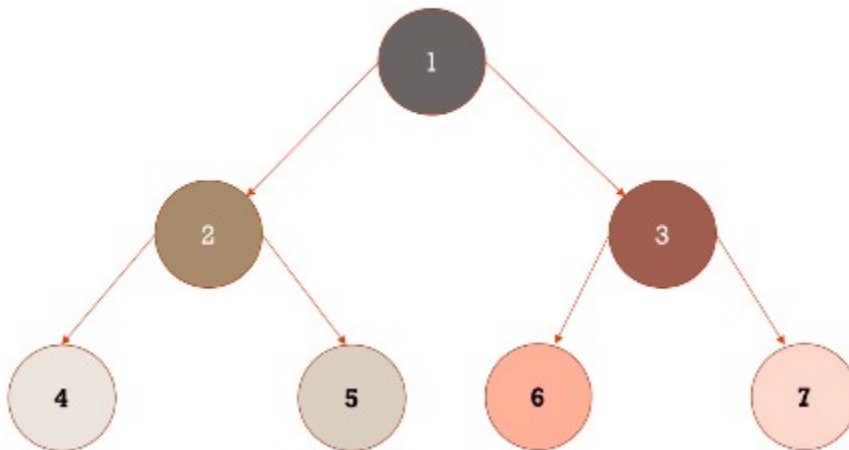
Task:

## TASKS IN SNOWFLAKE

- Task in snowflake is 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.

.Execute Single DML Statement or Procedure(Multiple DML).

**Task Dependency:** A task can be trigger by only one parent task. This is one constraint currently in Snowflake.

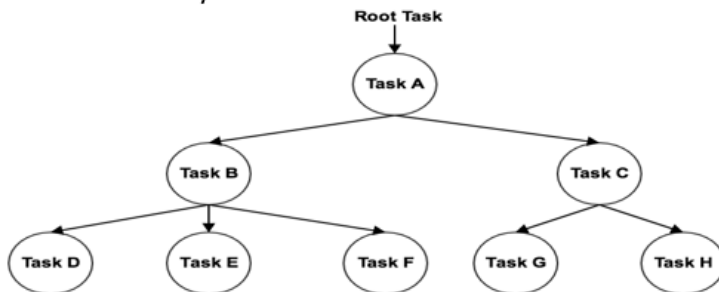


- Tasks DDL

- Create Task
- Alter Task
- Drop Task
- Describe Task
- Show Tasks
- SYSTEM\$CURRENT\_USER\_TASK\_NAME
- TASK\_HISTORY
- TASK\_DEPENDENTS
- SYSTEM\$TASK\_DEPENDENTS\_ENABLE

- Tasks in Hierarchy

- 



Root Task Must Have a Schedule

A Task is limited to 1 Predecessor Task

Max 100 Child Tasks – Task B

Activate \

## -- Creating task.

**CREATE OR REPLACE TASK mytask\_minute**

**WAREHOUSE = COMPUTE\_WH,**

**SCHEDULE = '1 MINUTE'**

**AS**

**INSERT INTO mytable(ts) VALUES(CURRENT\_TIMESTAMP);**

**CREATE TASK mytask\_hour**

**WAREHOUSE = mywh**

**SCHEDULE = 'USING CRON 0 9-17 \* \* SUN America/Los\_Angeles'**

**TIMESTAMP\_INPUT\_FORMAT = 'YYYY-MM-DD HH24'**

**AS**

**INSERT INTO mytable(ts) VALUES(CURRENT\_TIMESTAMP);**

-- SCHEDULE AT SPECIFIC TIME.

```
_____ minute (0-59)
| _____ hour (0-23)
| | _____ day of month (1-31, or L)
| | | _____ month (1-12, JAN-DEC)
| | | | _ day of week (0-6, SUN-SAT, or L)
| | | | |
| | | | |
* * * * *
```

-- Check sheduled tasks.

**SHOW TASKS**

-- Put task in the shedule.

```
alter task mytask_minute resume;
alter task mytask_minute SUSPEND
```

-- Check task history.

```
select *
from table(information_schema.task_history())
order by scheduled_time;
```

## 17. Loading unstructured data (Jason, XML, and Parquet)?

**Loading unstructured data:** We can load Jason, XML, and Parquet data using variant data type.

```
create or replace table json_demo (v variant);
```



**Json:**



**XML:**

## 18. Data sampling in snowflake (Row, Block, Clone)?

**Data Sampling:** Returns a subset of rows sampled randomly from the specified table instead of taking entire data from prod data base for analysis.

### Sampling in snowflake

- Approximate Query Processing (AQP) is a data querying method to provide approximate answers to queries at a fraction of the usual cost – both in terms of time and processing resource: big data and fast but not entirely accurate.
- It's similar to viewing pictures. You can take that great shot with a professional 30 Megapixel camera. Viewing it on your tablet, you barely see a difference if it had been taken with a compact camera. At least as long as you don't try to zoom in deeply, which you'd probably only do in case you found something interesting to drill into. So, to decide if your data is worth to be looked at in more detail you don't need to see each and every data point.

### Sampling method

---

- SYSTEM method.
- BERNOULLI method.

## Sampling method SYSTEM

- SYSTEM or BLOCK: 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  $p/100$  probability

## Sampling method BERNOULLI

- BERNOULLI or ROW: this is the simplest way of sampling, where Snowflake selects each row of the FROM table with  $p/100$  probability. the resulting sample size is approximately of  $p/100 * \text{number of rows on the FROM expression}$ .

## Advantages

- Development life cycle will be quick.
- We can test all scenarios on sample data with 80% accuracy.
- It will reduce development cost.
- Analytics team can test their queries on sample data.
- If they are getting 80 % accuracy then they can execute queries on prod data base.

--Sample Block wise

CREATE OR REPLACE TRANSIENT TABLE SUPPLIER\_BLOCK\_ALG

AS

select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10000.SUPPLIER sample  
system (3) seed (82);

--Sample Row wise.

**CREATE OR REPLACE TRANSIENT TABLE SUPPLIER\_ROW\_ALG**

**AS**

**select \* from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10000.SUPPLIER sample row  
(3) seed (82);**

-- CLONE sample: Clone also will take much time for querying the data for  
analysis better to go sample above.

**CREATE OR REPLACE TRANSIENT TABLE SUPPLIER\_CLONE CLONE SUPPLIER**

**CREATE OR REPLACE TRANSIENT TABLE SUPPLIER as select \* from  
SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF10000.SUPPLIER**

## **19. Performance tuning?**

- 1)absolute path while copy into from s3
- 2)break the bigger files like 10gb into small 100 mbs file so the parallel execution takes place to load and performance will be better(split -b <bytes>).
- 3)since snowflake is a columnar storage utilise by selecting only the requires columns instead of select \* from
- 4)Create multicluster warehouses for parallel tasks, which autosuspends after 60 secs
- 5)Decide Warehouse Size based on Environment
- 6)Separate Warehouse for Data Loading and Query Execution
- 7)Warehouse for Data Loading
- 8)Enable Auto-scaling
- 9)Optimize Insert Statements
- 10)Refrain from Executing Simple Queries
- 11)Enable Warehouse Auto-suspension

## CLUSTERING (lowest possible cardinality)

1)select system\$clustering\_information('test2', '(col1, col3)');

2)cluster ratio

SELECT e.ename, e.empno, m.ename as manager, e.mgr

FROM

emp e

LEFT OUTER JOIN emp m

ON e.mgr = m.empno

## 20. AWS s3 account creation (S3 bucket, Policy, Role) & Creating Integration.

**AWS S3:** It is a simple storage service that we can use to store and retrieve any amount data, at any time, from anywhere on the web.

### **S3 Basics:**

#### ***Pricing/Cost Overview:***

**Free Tier** use is available for S3.

#### **How are you charged for using S3?**

##### **(1) Storage Cost:**

- Applies to data at rest in S3
- Charged per GB used
- Price per GB varies based on region and storage class

##### **(2) Request Pricing - moving data in/out of S3:**

- PUT
- COPY
- POST
- LIST
- GET
- Lifecycle Transitions Request
- Data Retrieval
- Data Archive
- Data Restore

**NOTE:** Before doing any major usage of S3, you should make sure to review AWS's current pricing model to make sure you understand how much you will be required to pay.

**Detailed S3 pricing based on storage class:**

<https://aws.amazon.com/s3/pricing/>



# S3 Permissions

## What are S3 Permissions?

S3 permissions are what allow you to have **granular control** over who can view, access and use specific buckets and objects.

(1) Permissions functionality can be found on the bucket **AND** object level.

(2) On the bucket level you can control (for each bucket individually):

- List**: Who can see the bucket name.
- Upload/Delete**: Objects to (upload) or in the bucket (delete).
- View Permissions**
- Edit Permissions**: Add/edit/delete permissions

**NOTE: Bucket level permission are generally used for "internal" access control**

(3) On the Object level, you can control: (for each object individually)

- Open/Download**
- View Permissions**
- Edit Permissions**

**NOTE: You can share specific objects (via a link) with the anyone in the world.**

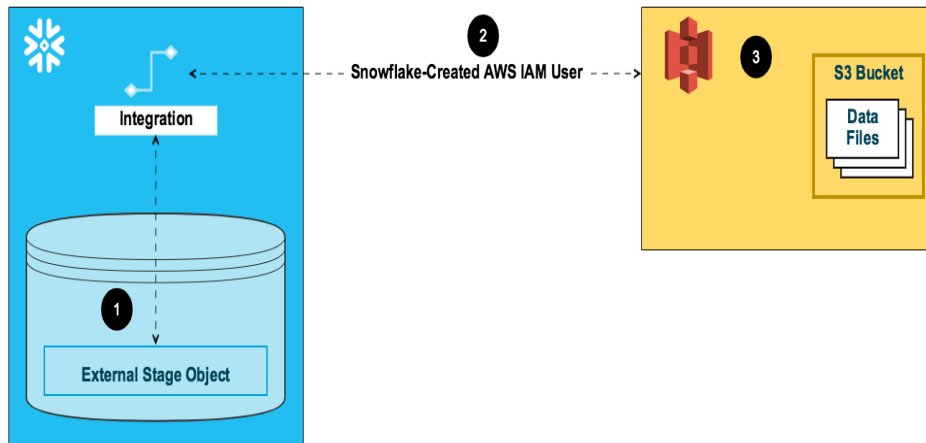
## How to connect S3 bucket & Integration from SK:

- Step 1: Create S3 Bucket / Access Permissions for the S3 Bucket
- Step 2: Snowflake related AWS IAM Role creation
- Step 3: Creation of Snowflake Cloud Storage Integration
- Step 4: AWS IAM User for your Snowflake Account updation with Snowflake Integration
- Step 5: Granting the IAM User Permissions to Access Bucket Objects
- Step 6: Creating an External Stage
- Step 7: Load Data into Table

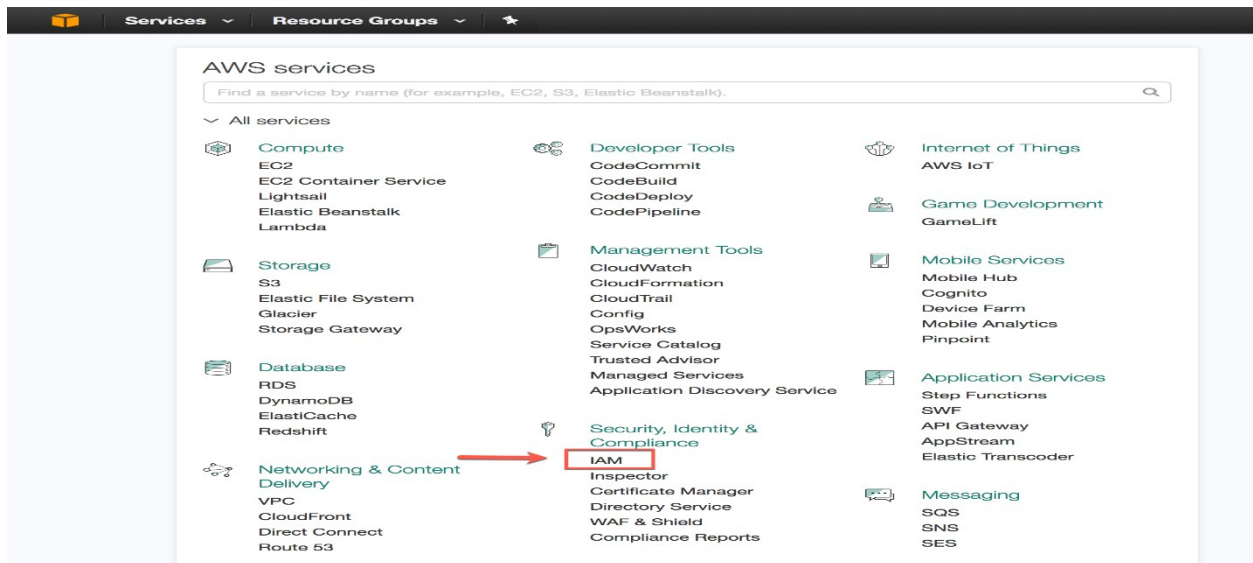
## Integration:

- Integration With S3 can be done through Below Options
- Option 1: Configuring a Snowflake Storage Integration
- Option 2: Configuring an AWS IAM Role

- Option 3: Configuring AWS IAM User Credentials



## Create IAM Policy:



Search IAM

Dashboard

Groups

Users

Roles

**Policies**

Identity providers

Account settings

Credential report

Encryption keys

Create Policy Policy Actions

Filter: Policy Type Filter

Showing 242 results

|                          | Policy Name                       | Attached Entities | Creation Time        | Edited Time          |
|--------------------------|-----------------------------------|-------------------|----------------------|----------------------|
| <input type="checkbox"/> | AmazonS3FullAccess                | 2                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AmazonS3ReadOnlyAccess            | 2                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AdministratorAccess               | 1                 | 2015-02-06 13:39 EST | 2015-02-06 13:39 EST |
| <input type="checkbox"/> | IAMUserChangePassword             | 1                 | 2016-11-14 19:25 EST | 2016-11-15 18:18 EST |
| <input type="checkbox"/> | SelfPasswordChange                | 1                 | 2016-12-27 10:19 EST | 2016-12-27 10:19 EST |
| <input type="checkbox"/> | AmazonAPIGatewayAdministrator     | 0                 | 2015-07-09 13:34 EST | 2015-07-09 13:34 EST |
| <input type="checkbox"/> | AmazonAPIGatewayInvokeFullAccess  | 0                 | 2015-07-09 13:36 EST | 2015-07-09 13:36 EST |
| <input type="checkbox"/> | AmazonAPIGatewayPushToCloudWat... | 0                 | 2015-11-11 18:41 EST | 2015-11-11 18:41 EST |
| <input type="checkbox"/> | AmazonAppStreamFullAccess         | 0                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AmazonAppStreamReadOnlyAccess     | 0                 | 2015-02-06 13:40 EST | 2016-12-07 16:00 EST |
| <input type="checkbox"/> | AmazonAppStreamServiceAccess      | 0                 | 2016-11-18 23:17 EST | 2016-11-18 23:17 EST |

**Services** **Resource Groups**

Search IAM

**Create Policy** Policy Actions

Filter: Policy Type Filter Showing 242 results

|                          | Policy Name                       | Attached Entities | Creation Time        | Edited Time          |
|--------------------------|-----------------------------------|-------------------|----------------------|----------------------|
| <input type="checkbox"/> | AmazonS3FullAccess                | 2                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AmazonS3ReadOnlyAccess            | 2                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AdministratorAccess               | 1                 | 2015-02-06 13:39 EST | 2015-02-06 13:39 EST |
| <input type="checkbox"/> | IAMUserChangePassword             | 1                 | 2016-11-14 19:25 EST | 2016-11-15 18:18 EST |
| <input type="checkbox"/> | SelfPasswordChange                | 1                 | 2016-12-27 10:19 EST | 2016-12-27 10:19 EST |
| <input type="checkbox"/> | AmazonAPIGatewayAdministrator     | 0                 | 2015-07-09 13:34 EST | 2015-07-09 13:34 EST |
| <input type="checkbox"/> | AmazonAPIGatewayInvokeFullAccess  | 0                 | 2015-07-09 13:36 EST | 2015-07-09 13:36 EST |
| <input type="checkbox"/> | AmazonAPIGatewayPushToCloudWat... | 0                 | 2015-11-11 18:41 EST | 2015-11-11 18:41 EST |
| <input type="checkbox"/> | AmazonAppStreamFullAccess         | 0                 | 2015-02-06 13:40 EST | 2015-02-06 13:40 EST |
| <input type="checkbox"/> | AmazonAppStreamReadOnlyAccess     | 0                 | 2015-02-06 13:40 EST | 2016-12-07 16:00 EST |
| <input type="checkbox"/> | AmazonAppStreamServiceAccess      | 0                 | 2016-11-18 23:17 EST | 2016-11-18 23:17 EST |

## Create IAM Role:

OpenShift Web Console S3 Management Console Amazon S3 Server Access Li IAM Management Cons New tab

https://console.aws.amazon.com/iam/home?region=ap-south-1#/roles

**Services** **Resource Groups**

**Identity and Access Management (IAM)**

▼ AWS Account (662291337917)

Dashboard

Groups

Users

**Roles**

Policies

Identity providers

Account settings

Credential report

Encryption keys

Search IAM

**Create role** Delete role

Search Showing 4 results

| Role name                                                | Description                                                                    | Trusted entities                               |
|----------------------------------------------------------|--------------------------------------------------------------------------------|------------------------------------------------|
| <input type="checkbox"/> AWSServiceRoleForSupport        | Enables resource access for AWS to provide billing, administrative and supp... | AWS service: support (Service-Linked role)     |
| <input type="checkbox"/> AWSServiceRoleForTrustedAdvisor | Access for the AWS Trusted Advisor Service to help reduce cost, increase p...  | AWS service: trustedadvisor (Service-Linked... |
| <input type="checkbox"/> snowflake-access                | snowflake-access                                                               | Account: 418734513476                          |
| <input type="checkbox"/> snowflakeRole                   | Snowflake Role                                                                 | Account: 418734513476                          |

Feedback English (US)

© 2008 - 2019, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Type here to search

6:34 PM 8/1/2019

**Create role**

Select type of trusted entity

☐ AWS service  
EC2, Lambda and others
 ☒ **Another AWS account**  
Belonging to you or 3rd party
 ☐ Web identity  
Cognito or any OpenID provider
 ☐ SAML 2.0 federation  
Your corporate directory

Allows entities in other accounts to perform actions in this account. [Learn more](#)

Specify accounts that can use this role

Account ID\*

Options ☒ Require external ID (Best practice when a third party will assume this role)

You can increase the security of your role by requiring an optional external identifier, which prevents "confused deputy" attacks. This is recommended if you do not own or have administrative access to the account that can assume this role. The external ID can include any characters that you choose. To assume this role, users must be in the trusted account and provide this exact external ID. [Learn more](#)

External ID

**Important:** The console does not support using an external ID with the Switch Role feature. If you select this option, entities in the trusted account must use the API, CLI, or a custom federation proxy to make cross-account `iam:AssumeRole` calls. [Learn more](#)

\* Required Cancel Next: Permissions

## AWS S3 Configuration with integration.

create or replace storage integration s3\_int

type = external\_stage

storage\_provider = s3

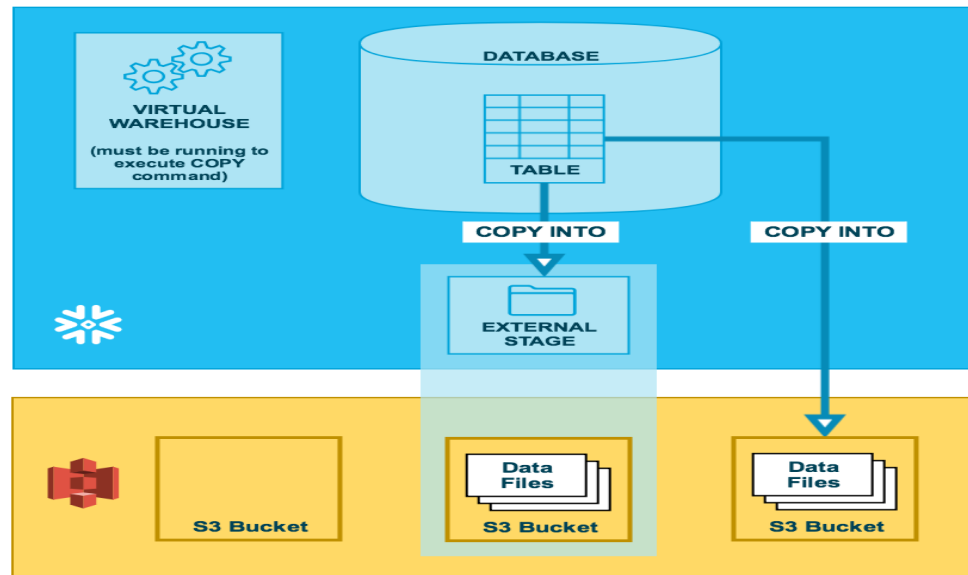
enabled = true

storage\_aws\_role\_arn = 'arn:aws:iam::\*\*\*\*\*:role/snowflake'

storage\_allowed\_locations = ('s3://snowflake069/employee/');

DESC INTEGRATION s3\_int;

Unloading data from SK to S3 bucket:



**Aws commands:**

-- Copy data from local to aws s3.

```
aws s3 cp /GITHUB/Snowflake/Snowflake/getting-started/
s3://snowflake069/upload_cli --recursive
```

-- Copy data from aws s3 to local.

```
aws s3 cp s3://snowflake069/upload_cli
/GITHUB/Snowflake/Snowflake/getting-started_1/ --recursive
```

-- Command to list all files in aws s3.

```
aws s3 ls s3://snowflake069/ --recursive
```

--- AWS S3 Configuration.

create or replace storage integration s3\_int

type = external\_stage

storage\_provider = s3

enabled = true

```
storage_aws_role_arn = 'arn:aws:iam::*****:role/snowflake'

storage_allowed_locations =
('s3://snowflake069/employee/', 's3://bucky2018/');
```

```
DESC INTEGRATION s3_int;
```

```
create or replace file format control_db.file_formats.my_csv_format
type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
empty_field_as_null = true;
```

```
desc file format my_csv_format;
```

```
create or replace stage control_db.external_stages.my_s3_stage
storage_integration = s3_int
url = 's3://bucky2018/'
file_format = control_db.file_formats.my_csv_format;
```

```
-- Query data directly
```

```
select t.$1 as first_name, t.$2 last_name, t.$3 email
from @control_db.external_stages.my_s3_stage/ t
```

```
-- filter data directly
```

```
select t.$1 as first_name, t.$2 last_name, t.$3 email
from @control_db.external_stages.my_s3_stage/ t
where t.$1 in ('Di', 'Carson', 'Dana')
```

-- you can write join condition.

```
select t.$1 as first_name,t.$2 last_name,t.$3 email
from @control_db.external_stages.my_s3_stage/ t,
 @control_db.external_stages.my_s3_stage/ d
where t.$1 =d.$1
```

-- You can also create views & Tables.

```
create or replace view demo_db.public.query_from_s3
as
```

```
select t.$1 as first_name,t.$2 last_name,t.$3 email
from @control_db.external_stages.my_s3_stage/ t
```

```
create or replace table demo_db.public.query_from_s3_table
as
```

```
select t.$1 as first_name,t.$2 last_name,t.$3 email
from @control_db.external_stages.my_s3_stage/ t
```

```
select * from demo_db.public.query_from_s3_table;
show tables;
```

```
select * from demo_db.public.query_from_s3
where First_name='Ivett';
select count(*) from demo_db.public.query_from_s3;
```

**/\*\*\*\*\* LOAD DATA FROM S3 TO SNOWFLAKE \*\*\*\*\*/**



```
create or replace table demo_db.public.emp_ext_stage (
 first_name string ,
 last_name string ,
 email string ,
 streetaddress string ,
 city string ,
 start_date date
);
```

```
create or replace stage control_db.external_stages.my_s3_stage
 storage_integration = s3_int
 url = 's3://snowflake069/employee/'
 file_format = control_db.file_formats.my_csv_format;
```

```
copy into demo_db.public.emp_ext_stage
from (select t.$1 , t.$2 , t.$3 , t.$4 , t.$5 , t.$6 from
@control_db.external_stages.my_s3_stage/ t)
--pattern = '.*employees0[1-5].csv'
on_error = 'CONTINUE';
```

```
copy into demo_db.public.emp_ext_stage
from (select metadata$filename,t.$1 , t.$2 , t.$3 , t.$4 , t.$5 , t.$6 from
@control_db.external_stages.my_s3_stage/ t)
```

```
pattern = '.*employees0[1-5].csv'
```

```
on_error = 'CONTINUE';
```

```
TRUNCATE TABLE emp_ext_stage;
```

```
SELECT * FROM emp_ext_stage
```

```
select * from table(validate(emp_ext_stage, job_id=>'01931c98-02b2-9c2c-000f-73030001a0e2'));
```

```
/****** UNLOAD DATA FROM SNOWFLAKE TO S3 *****/
```

```
create or replace file format my_csv_unload_format
```

```
type = csv field_delimiter = ',' skip_header = 1 null_if = ('NULL', 'null')
```

```
empty_field_as_null = true compression = gzip;
```

```
alter storage integration s3_int set
```

```
storage_allowed_locations=('s3://snowflake069/employee/', 's3://snowflake069/emp_unload/', 's3://snowflake069/zip_folder/')
```

```
desc integration s3_int
```

```
create or replace stage my_s3_unload_stage
```

```
storage_integration = s3_int
```

```
url = 's3://snowflake069/emp_unload/'
```

```
file_format = my_csv_unload_format;
```

**copy into @my\_s3\_unload\_stage**

**from**

**emp\_ext\_stage**

**copy into @my\_s3\_unload\_stage/select\_**

**from**

**(**

**select**

**first\_name,**

**email**

**from**

**emp\_ext\_stage**

**)**

**copy into @my\_s3\_unload\_stage/parquet\_**

**from**

**emp\_ext\_stage**

**FILE\_FORMAT=(TYPE='PARQUET' SNAPPY\_COMPRESSION=TRUE)**

Create| staging  
-----

```
create or replace stage my_ext_stage2
 url='s3://load/encrypted_files/'
 credentials=(aws_key_id='1a2b3c' aws_secret_key='4x5y6z')
 encryption=(master_key = 'eSxX0jzYfIamtnBK0EOwq80Au6NbSgPH5r4BDDw0a08=');
```

Direct copy options  
-----

```
COPY into 's3:<path>'
from <TABLE>
HEADER=TRUE credentials=(AWS_KEY_ID='aws_key_value' AWS_SECRET_KEY='aws_secret_value')
FILE_FORMAT=(TYPE='PARQUET' SNAPPY_COMPRESSION=TRUE) MAX_FILE_SIZE = 4000000000;
```

```
COPY into <TABLE>
from 's3:<path>'
HEADER=TRUE credentials=(AWS_KEY_ID='aws_key_value' AWS_SECRET_KEY='aws_secret_value')
FILE_FORMAT=(TYPE='PARQUET' SNAPPY_COMPRESSION=TRUE) MAX_FILE_SIZE = 4000000000;
```

**\*\*\*\*\* Copy zip files\*\*\*\*\***

**create or replace stage my\_s3\_zip\_stage**

**storage\_integration = s3\_int**

**url = 's3://snowflake069/zip\_folder/'**

**file\_format = my\_csv\_format;**

**copy into emp\_ext\_stage**

**from (select t.\$1 , t.\$2 , t.\$3 , t.\$4 , t.\$5 , t.\$6 from @my\_s3\_zip\_stage/ t)**

**--pattern = '.\*employees0[1-5].csv'**

**on\_error = 'CONTINUE';**

## **21. Snow CLI, AWS CLI?**

**Snow CLI:** Using command line we can connect snowflake for writing snow sql.

Snowsql – a <snowflake account id>

Password

**AWS CLI:** Using command line we can connect AWS for working S3 bucket files like splitting the files and etc..., we need to some AWS commands.

-- Aws commands

-- Copy data from local to aws s3.

```
aws s3 cp /GITHUB/Snowflake/Snowflake/getting-started/
s3://snowflake069/upload_cli --recursive
```

-- Copy data from aws s3 to local.

```
aws s3 cp s3://snowflake069/upload_cli
/GITHUB/Snowflake/Snowflake/getting-started_1/ --recursive
```

-- Command to list all files in aws s3.

```
aws s3 ls s3://snowflake069/ --recursive
```

=====END=====

This Notes Prepared by

Chandu.