Cloud-Based Architecture: Snowflake is built on a cloud-based architecture, which means it leverages the power of cloud computing for data warehousing, allowing for scalability and flexibility.

Data Warehousing: Snowflake is primarily used for data warehousing, which involves the storage, processing, and analysis of large volumes of data.

Multi-Cluster, Shared-Data Architecture: Snowflake uses a multi-cluster, shared-data architecture. Multiple compute clusters can access the same data simultaneously, which enhances performance and concurrency.

Virtual Warehouses: In Snowflake, you create virtual warehouses to separate and manage your compute resources. This allows you to scale your resources up or down based on demand.

Data Storage: Snowflake separates data storage from compute resources. You pay for storage separately from compute, and this decoupling is a key feature for cost management.

SQL Interface: Snowflake uses SQL as its query language, making it accessible to SQL-savvy users.

Automatic Query Optimization: Snowflake handles query optimization automatically, so you don't need to manually tune queries for performance.

Data Sharing: Snowflake allows for easy and secure data sharing between organizations and within your organization. You can share data without copying it.

Data Security: Snowflake provides robust security features, including encryption, role-based access control, and auditing.

Elasticity: You can easily scale up or down based on your workload, which makes Snowflake cost-effective.

Data Loading: Snowflake provides multiple ways to load data, including bulk loading, streaming, and third-party integrations.

Snowpipe: Snowpipe is a service for automatically ingesting streaming data into Snowflake.

Semi-Structured Data: Snowflake supports semi-structured data formats like JSON, Avro, and Parquet, allowing you to work with a variety of data types.

Time Travel: Snowflake provides features like Time Travel and Fail-Safe, which allow you to recover from accidental data changes and historical data analysis.

Global Availability: Snowflake is available across multiple cloud providers, giving you the flexibility to choose your preferred cloud infrastructure.

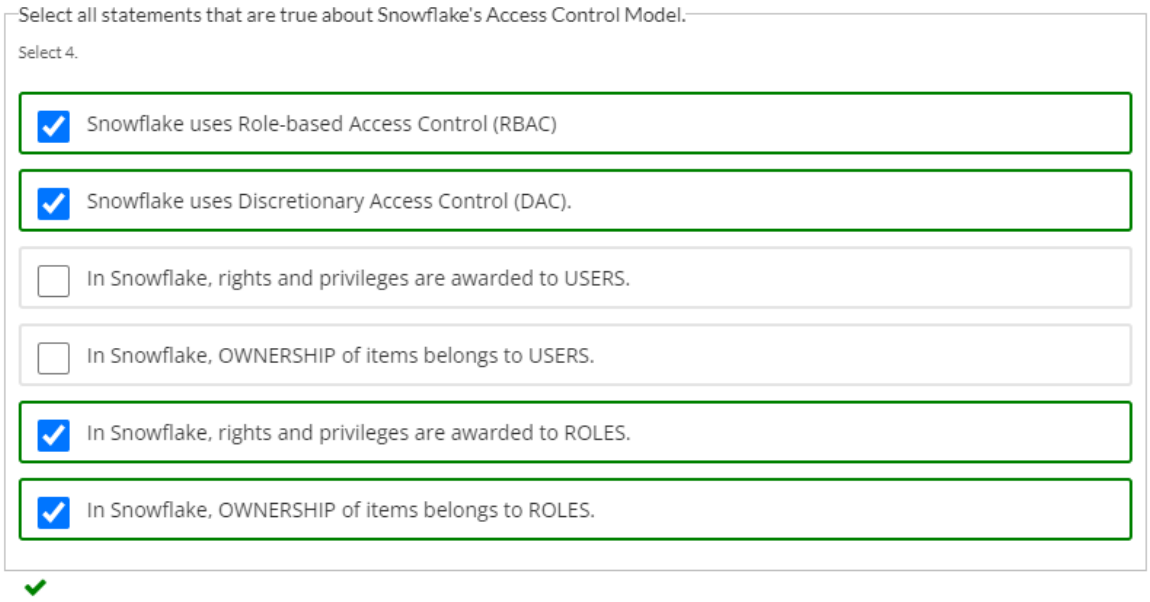
Snowflake Data Marketplace: You can access third-party data sets and data services through Snowflake's Data Marketplace.

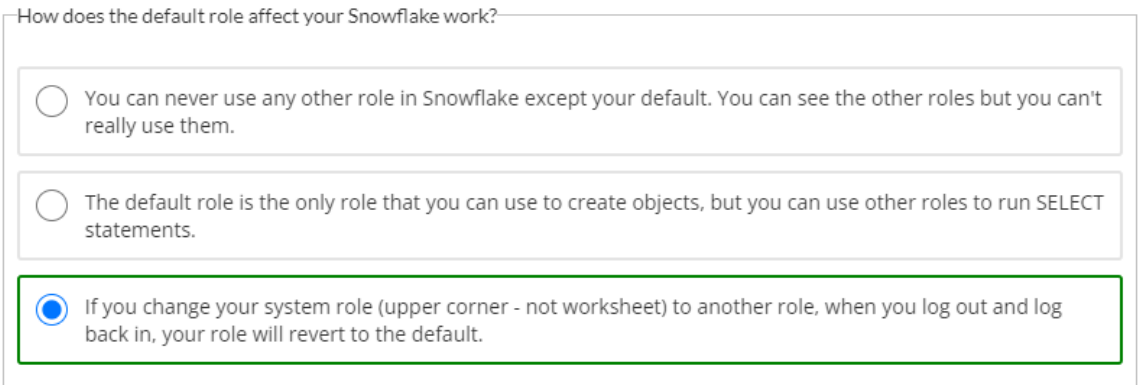
Query Performance: Snowflake is known for its excellent query performance, thanks to its architecture and optimizations.

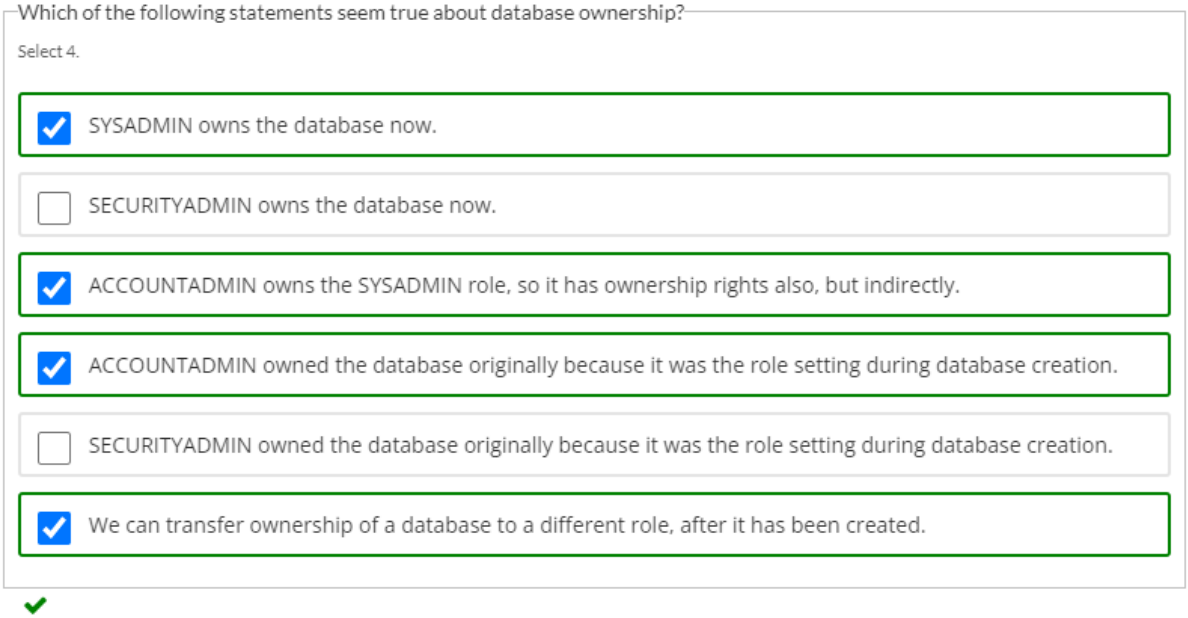
Integration: Snowflake can be easily integrated with popular data visualization and ETL tools.

Cost Management: With separate billing for storage and compute, you have better control over costs. Snowflake also offers features like Auto-Suspend and Auto-Resume to save on compute costs.

Compliance: Snowflake complies with various data security and compliance standards, making it suitable for a wide range of industries.







Follow these steps to add the Sample Database to your account:

1) Role set to **ACCOUNTADMIN**

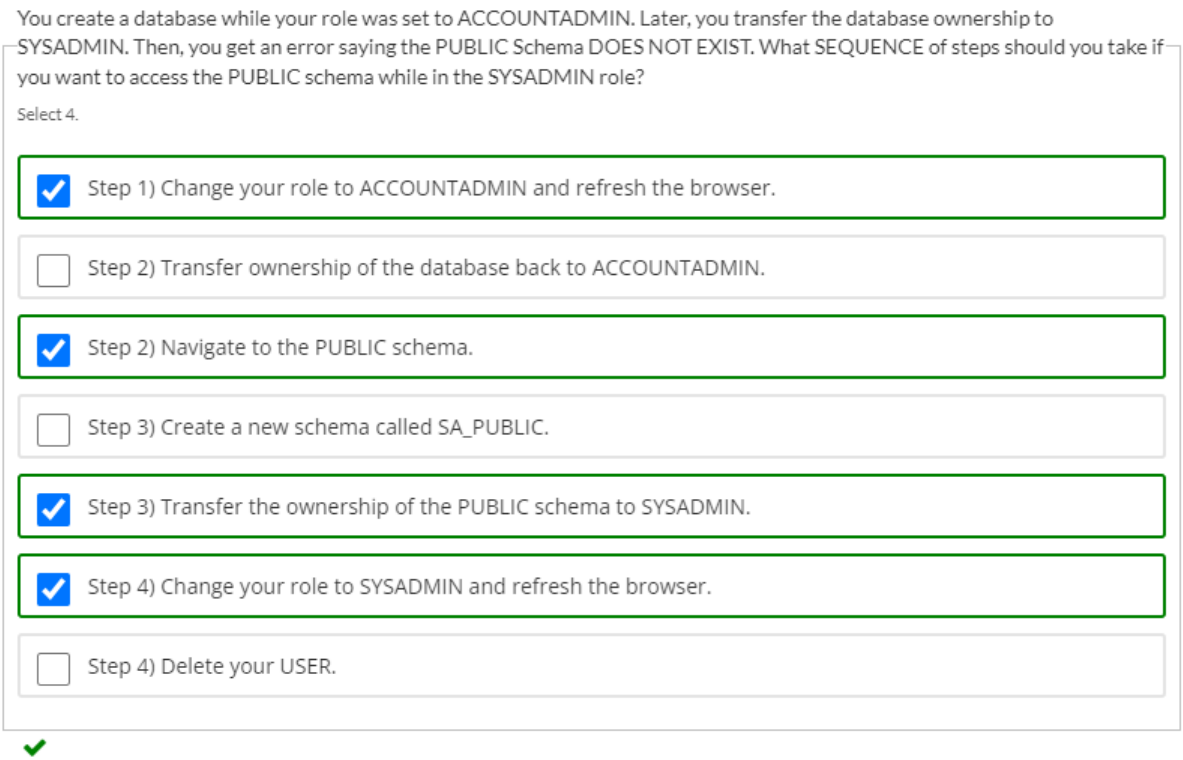
2) Click on side menu **DATA**

3) Click on **Private Sharing**

4) Look for **Direct Sharing**

5) Click on **SAMPLE_DATA**

6) In the dialog, edit the name to say **SNOWFLAKE_SAMPLE_DATA** and choose **SYSADMIN** as the other role that should have access.



There are four drop-menu settings or defaults for each worksheet. What are the four drop-menu worksheet context settings?

- [x] Database
- [x] Schema
- [ ] Account
- [x] Warehouse
- [x] Role
- [ ] Password
- [ ] Username

1/1 point (ungraded)

How does adding "IN ACCOUNT" to the SHOW SCHEMAS command change the result set?

- ( ) No schemas are shown. The command is not valid.
- (•) All schemas from all databases are shown (based on current role).
- ( ) Only schemas from databases owned by ACCOUNTADMIN are shown.

What is the difference between a regular lab and a Challenge Lab?

Select 2.

- [ ] Challenge labs are optional.
- [ ] Challenge labs are for advanced users only.
- [x] Challenge labs ask you to complete tasks without giving you step-by-step instructions.
- [x] Challenge labs don't teach new skills, they give you a chance to apply recently learned skills again.

Refer to the above image to answer this question. You created the schema called VEGGIES but when you run the SHOW SCHEMAS command it does not appear. Assuming your current worksheet role has access to the schema, what options below would make the schema appear in the results?

Select all that apply.

- [ ] Change your default role to SYSADMIN (and run again).
- [x] Set the worksheet database drop menu to GARDEN_PLANTS (and run again).
- [ ] Set the worksheet warehouse drop menu to VEGGIES_WH (and run again).
- [ ] Add "all" to the show schemas command (and run again).
- [x] Add "in account" to the show schemas command (and run again).
- [ ] Add "where schema = 'VEGGIES' " to the show schemas command (and run again).

✔

The data in the spreadsheets isn't quite right for database use. What are the two skills Tsai uses to re-structure the data to make it more ready for a database?

Select 2.

- [ ] Structured Query Language
- [ ] Visual Basic
- [x] Normalization
- [x] Data Modeling

1/1 point (ungraded)

How can you PREVIEW data in a table while in an AppUI Worksheet?

- (●) You can find the table in the Object Picker, click on the table name, then click on the magnifying glass symbol.
- ( ) You can set your worksheet context TABLE drop menu to PREVIEW, which is designed for data previewing.
- ( ) You can set your worksheet context ROLE drop menu to PUBLIC, which is designed for data previewing.
- ( ) You can right-click on the table name and choose the "Table Preview Now" option.

✔

Which Role should own your ROOT_DEPTH table?

- ( ) ACCOUNTADMIN
- (●) SYSADMIN
- ( ) PUBLIC

✔

What settings can you choose for each Worksheet (often called the Worksheet Context) that make writing and running code easier to type and simpler to read?

Select 2.

- [x] The "home" Database
- [x] The "home" Schema
- [ ] The "home" Table
- [ ] The "home" Account

✔

What settings MUST you choose for each Worksheet (also called the Worksheet Context) that, without them, code cannot run?

Select 2.

- [x] The warehouse to be used for all processing.
- [ ] The table to be used for compute.
- [ ] The account to be used for all processing.
- [x] The user role to carry out the commands.
- [ ] The user name to carry out the commands.
- [ ] The account to carry out the commands.

✔

Two of the context settings in a worksheet are in the upper-right corner, while two are in the middle, closer to the code. Why do you think they are positioned this way?

Select 2.

- [x] The set near the code is intended for convenience and simplicity of code.
- [ ] The set near the upper corner is intended for convenience and simplicity of code.
- [x] The set near the upper corner is NOT for convenience, it is required.
- [ ] The set near the code is NOT for convenience, it is required.

✔

You created a table and added some rows, later you try to query the table and Snowflake says the table doesn't exist? What is the first rule of Snowflake troubleshooting?

- ( ) Check your USER name.
- ( ) Check your default ROLE.
- (•) Check your current ROLE.
- ( ) Check your ACCOUNT.

✔

Find all the true statements about Snowflake Warehouses below and select them.

Select 3.

- [ ] Snowflake Warehouses are like Data Marts in that they hold subsections of an organization's data.
- [ ] Snowflake Warehouses were designed to hold data in structures based on Kimball and Inmon Data Warehousing theories.
- [x] Snowflake Warehouses do not hold data.
- [x] Snowflake Warehouses provide computing power to run queries, load data, and carry out other tasks.
- [ ] Functionally, a Snowflake Warehouse is more like a spreadsheet workbook than a laptop CPU.
- [x] Functionally, a Snowflake Warehouse is more like a laptop CPU than a spreadsheet workbook.

✔

Scaling OUT/IN and scaling UP/DOWN are easily confused. Check all the true statements about scaling, below.

Select 3.

- [x] Editing a XS Warehouse and making it a M is an example of scaling UP.
- [ ] Editing a XS Warehouse and making it a M is an example of scaling DOWN.
- [ ] Editing a XS Warehouse and making it a M is an example of scaling OUT.
- [ ] Editing a XL Warehouse and making it a M is an example of scaling UP.
- [x] Editing a XL Warehouse and making it a M is an example of scaling DOWN.
- [ ] Editing a XL Warehouse and making it a M is an example of scaling OUT.
- [ ] When an XS Warehouse automatically adds clusters to handle an increase workload, this is an example of scaling UP.
- [ ] When an XS Warehouse automatically adds clusters to handle an increase workload, this is an example of scaling DOWN.
- [x] When an XS Warehouse automatically adds clusters to handle an increase workload, this is an example of scaling OUT.

✔

Warehouses can be manually scaled UP or DOWN. They can be set up so that they automatically scale OUT. What did the video call the opposite of SCALING OUT?

- ○ De-Scaling
- ◉ Snapping Back
- ○ Shedding Clusters
- ○ Scaling DOWN

✔

Warehouse scaling involves both servers and clusters. Which statements are true about the role of servers and clusters in Warehouse sizing and scaling?

Select 7.

- ☑ Cluster just means a "group" of servers.
- ☐ Server just means a "group" of clusters.
- ☑ An XS-sized warehouse (not scaled out) has 1 cluster.
- ☑ An M-sized warehouse (not scaled out) has 1 cluster.
- ☐ An M-sized warehouse (not scaled out) has more clusters than an XS-sized Warehouse.
- ☑ The number of servers in a warehouse is different, based on size (XS, S, M, etc)
- ☑ An XS-sized Warehouse, when scaled out, has more than 1 cluster.
- ☑ An M-sized Warehouse, when scaled out, has more than 1 cluster.
- ☐ A server can hold multiple clusters.
- ☑ A cluster can hold multiple servers.

✔

You see this DOES NOT EXIST message when trying to run the select statement shown (see image above)? What is the issue?

- [ ] Your default role is wrong.
- [ ] Your current worksheet user is wrong.
- [ ] Your worksheet database setting is wrong.
- [ ] Your worksheet schema setting is wrong.
- [x] You have a typo.
- [ ] Your account is not activated.

✔

Why won't the CSV shown above load properly if Uncle Yer tries to load it?

- ( ) Some of the plant names have tabs in them.
- ( ) He shortened the Rooting Depth values to a single letter.
- ( ) Some words in the plant names are not capitalized
- (•) Some of the plant names have commas in them.

✔

Once you save a set of FILE FORMAT settings into a FILE FORMAT object, it will make it easier to load files because it tells Snowflake how the data will be structured when it arrives. In the next lab, you'll learn how to encapsulate a set of load settings into a file format object.

1/1 point (ungraded)

What is a File Format?

- (•) It's a way to tell Snowflake how your data will be structured when it arrives.
- ( ) It's a way to set margins and font for a table.
- ( ) It's a way to choose the background color of a web page.

✔

Where does Snowflake store some of its Metadata?

- ○ The INFORMATION_DB database of each account.
- ○ The METADATA_SCHEMA of each database.
- ○ The INFO_METADATA schema of each database.
- ◉ The INFORMATION_SCHEMA schema of each database.

✔

If we want to check for the schemas we created, why not just look with our own eyes to see if they are there?

Select 3.

- ☑ Because if we misspelled something, we might not notice the name issue, but checking via code might catch it.
- ☑ Because we may want to automate the check for some reason.
- ☑ Because coding things is fun.

When speaking of warehouses (real world or data) what is a stage?

- ◉ A temporary storage location.
- ○ A place where actors perform Shakespearean soliloquies.

✔

What happens if you try to hack the URL for the LU_SOIL_TYPE.tsv file?

- ○ It opens just like the .txt did!
- ○ I get a red message from Snowflake telling I've already downloaded this file.
- ◉ It doesn't open in a browser window, instead it just downloads the file.

✔

What happens if you try to hack the URL for the author_no_header.xml file?

- ( ) It opens just like the .txt did!
- ( ) It doesn't open in a browser window, instead it just downloads the file.
- (•) I get an error message.

✔

Why do you think we gave our Snowflake Stage Object the name LIKE_A_WINDOW_INTO_AN_S3_BUCKET?

- ( ) Because it's short so it's easy to type.
- ( ) Because Stages are easy to break, like windows can sometimes be.
- (•) Because "stage" sounds like a location, and in this case, the location is the S3 bucket, not the Snowflak object we created.

✔

UPPER CASE, lower case, miXEd CaSe? Who cares? Which of the statements below are true about Snowflake's case sensitivity?

Select 4.

- [x] Snowflake assumes you always meant to type UPPER_CASE, unless you put something in double-quotes.
- [x] If you create a table called [happy] (without double quotes) Snowflake will actually name it [HAPPY]
- [ ] If you create a table called [HAPPY] Snowflake will actually name it [happy]
- [x] If you create a table called ["hAppY"] (and use double quotes in the create table statement) Snowflake will actually name it [hAppY]
- [x] If you create a table called ["hAppY"], every time you query it, you will need to put it in double quotes like this: "hAppY"

✔

UPPER CASE, lower case, miXEd CaSe? Who cares? Which of the statements below are true about S3's case sensitivity?

- (●) S3 assumes you always meant to type exactly what you typed.
- ( ) If you are looking for a file named "haPPY.txt," you can type "HAPPY.TXT" and it will find it.
- ( ) As long as the file name part is the correct case and spelling, you can type the file extension case differntly (.txt vs .TXT) and S3 will find the file.

✔

Snowflake is cloud agnostic, especially when it comes to loading files from stage into Snowflake accounts. If truly agnostic, which of the below options SHOULD be true?

Select 3.

- [ ] A Snowflake Account on AWS can only have stage objects that point to, and load files from AWS S3 buckets.
- [x] A Snowflake Account on AWS can have stage objects that point to, and load files from AWS, Azure and/or GCP.
- [ ] A Snowflake Account on Azure can only have stage objects that point to, and load files from Azure Blob Storage.
- [x] A Snowflake Account on Azure can have stage objects that point to, and load files from AWS, Azure and/or GCP.
- [ ] A Snowflake Account on GCP can only have stage objects that point to, and load files from GCP buckets.
- [x] A Snowflake Account on GCP can have stage objects that point to, and load files from AWS, Azure and/or GCP.

✔

In each line below we have taken a line from the COPY INTO statement above and changed the format of the text. Some of the rows, if changed, would not cause any issues. Which changed row would cause problems because of case sensitivity?

- ( ) COPY INTO my_table_name
- ( ) from @LIKE_a_window_into_an_s3_bucket
- (●) files = ('if_i_had_a_file_like_this.txt')
- ( ) file_format = (format_name='EXAMPLE_FILEFORMAT');

✔

Which file format property, when set to TRUE, has the power to ignore the square brackets and load each author into a separate row?

- ENABLE_OCTAL
- ALLOW_DUPLICATE
- STRIP_OUTER_ARRAY
- STRIP_NULL_VALUES
- IGNORE_UTF8_ERRORS

1/1 point (ungraded)

Which of these looks like an Snowflake Account Locator?

- 0053246MMRxJAAT
- SLIMSHADY86
- MC46564

Answer

Correct: Yep! Account locators are 2-3 letters followed by 5-6 numbers.

If you run a SELECT statement that has worked in the past, but now you get an error message saying the table doesn't exist, what should you check?

- Your cloud provider.
- Your current region.
- Your worksheet context role.
- Your worksheet context warehouse.
- Your role under your name in the upper corner.

✓ 1/1 point (ungraded)

Which of these objects has the most compute power?

- A Snowflake Database.
- A Snowflake Schema.
- A Snowflake Sequence.
- ◉ A Snowflake Warehouse.
- A Snowflake Data Mart.

✓

Answer

Correct: ✓ Yep! Warehouses are what Snowflake calls it's compute resources. A little weird, but you'll get used to it.

Which of these objects is the lowest or smallest in the storage container hierarchy?

- A Snowflake Database.
- A Snowflake Schema.
- A Snowflake Region.
- A Snowflake Account.
- ◉ A Snowflake Table.

✓

Which of the statements below describes an external Stage in Snowflake?

Select 3.

- [x] You can use a COPY INTO statement to move data from a stage, into a table.
- [x] A stage can provide a "window" between Snowflake and a cloud folder.
- [x] A stage is often intended as a temporary location for data. Data is put there on it's way to longer term location.
- [ ] A stage can be used as a counter to generate unique ids for each new row in a table. We assign a starting value and an increment.
- [ ] A stage allows us to group database tables. For example, we created 3 stages in the GARDEN_PLANTS database. One was named VEGGIES. .

Udemy LEARNINGS on Snowflake

Introduction:

Architecture

Data loading: data aloding and unloding techniques

Performance tuning – performance otmiztion techniques. Time travel, cloning and data sharing.

Continuous data protection

Secure data sharing

Resource management

Security

Cloning

Snowflake architecture – decoupled storage and compute. How they all work together to deliver scalable and efficient data warehousing solution.

How to load data into snowflake from cloud and on-premises sources. Data loading approach via bulk loading approach and also continuous data loading using snowpy.

How to optimize the data loading process.

Performance optimization techniques include query re-writing , query profiling, materialized views clustering and cache.

Snowflake includes cloning , time travel and data sharing. Time travel allows users to query data as it appeared at specific point of time making it easy to track changes and analyse data discrepancies.

Cloning enables users to create copy of entire databases schemas or tables within a few mins. Which can help creating sandboxes or backups.

Data sharing allows users to share data securely and efficiently with other snowflake accounts enabling collaboration and data monetization.

Snowflake security features include multi factor authentication, RBAC, and data encryption in transit and rest.

Resource management in snowflake covers resource monitors used to monitor the cost of virtual warehouses and the use on information schema and account schema to view and monitor the compute and storage.

what is Snowflake?

Snowflake is a purpose built data warehousing platform built from scratch exclusively for public cloud platforms.

So it is available as a cloud only software as a service offer.

Since it has been designed for the cloud, the software is optimized for execution on the cloud and

takes advantage of cloud concepts like decoupling the storage and compute from each other scalability features.

We'll learn more about this decoupling and how it results in a better outcome for the customer.

So Snowflake is provided as a software, as a service offering, which means that there is no hardware to manage for the customer and no software to maintain or upgrade.

Any updates, optimizations and tunings are made by Snowflake itself and it's made available to all customers automatically.

A snowflake data warehouse is a paper used only, and because of the decoupled nature of storage and compute, a customer only pays for the actual storage used and the actual compute used.

This means that you could store terabytes of data, and if you're not processing that data regularly, you get charged only for the storage costs. So finally, Snowflakes design takes advantage of the storage and compute scalability that is offered by the underlying cloud platforms.

Since it uses object storage, the storage available to a snowflake customer is virtually unlimited.

It's highly fault tolerant and can be scaled up to any number. Similarly, on the processing front, Snowflake uses scalable compute clusters called virtual warehouses.

Virtual warehouses allow the processing power available to Snowflake to be scaled in many different ways.

Snowflake was ranked second on Forbes magazine's Cloud 100 list and first on LinkedIn's 2019 US top startup list.

So what does the future hold for Snowflake?

With all the unique features that Snowflake has, customers worldwide are moving to Snowflake as their platform of choice.

And slowly but surely, Snowflake is gaining traction and becoming a snowball.

And I believe this technology is seriously challenging both traditional data warehousing platforms and the big data platforms.

If you are working with data warehousing, big data or databases in any capacity.

Snowflake is undoubtedly one of the technology to master.

Private Snowflake.

So all the fundamentals of a cloud based data warehousing solution are included in Snowflake Standard Edition.

It includes complete SQL data, warehouse capabilities, data sharing, data encryption in transit and at rest and access to the Snowflake data marketplace.

Using the standard edition, you can perform database replication, federated authentication, multi-factor authentication, cloning, failsafe and time travel are also included.

However, in the Standard edition, the ability to travel back in time is limited to one day.

The Enterprise has all the capabilities of the standard edition but adds additional capabilities.

These include time travel up to 90 days Multi-cluster virtual warehouses, materialized views, dynamic masking, search optimization, external data, tokenization and annual rekeying of the data.

The business-Critical Edition enhances the enterprise edition with additional security features such

as a customer managed key payment card industry or PCI compliance and private connectivity support, as well as failover.

And then finally, the virtual private snowflake.

It builds on all these editions. So it has all the capabilities of Business Critical Edition, but also provides additional isolation by providing a customer specific metadata store and a customer specific pool of computing resources that are not shared with any other customer.

So what that means is you will get your own isolated version of Snowflake.

Snowsight: Snowflake's web interface.

Virtual private Snowflake edition provides dedicated compute resources and dedicated metadata store.

What is the minimum Snowflake edition which supports multi-factor authentication (MFA) – Standard edition.

What is the minimum Snowflake edition that supports private connectivity to Snowflake – Business critical

Snowflake architecture:

Hybrid architecture -multi cluster , shared data architecture.

Multi cluster, shared data – separation of storage and compute allows Unlimited scalability which is independent of each other.

Snowflake supports below clouds service providers storages:

AWS S3, Azure Blob storage or Google cloud storage to store its data. Since snowflake stores data on object

Currently, Snowflake supports AWS, S3 Storage, Azure Blob Storage or Google Cloud Storage to store its data.

Since Snowflake stores data on object storage on the cloud platform, the storage can scale indefinitely and independently of compute.
The cloud platform is responsible for providing the durability for these stored files.

Therefore, Snowflake can take advantage of the disaster recovery and fault tolerance provided by the underlying cloud platform.
Data that is loaded into Snowflake is stored as files on the object based cloud storage.
It is worth mentioning here that cloud based object storage is immutable, which means stored data cannot be updated once it is written, but it can only be appended to if updates are required to a file that was written to an object store.

You must remove the complete file, perform an update and write the new file back to the object store.
So this immutability of files on object store presents an interesting challenge that Snowflake solves through its unique micro partitioning approach.
These micro partitions also form the architectural basis of many exciting snowflake features such as time travel, secure data sharing and cloning.
So as this course progresses, we'll be talking a lot more about micro partitions, how they are managed, how data is inserted in those micro partitions, what is the maximum size of a micro partition and what are the immutability implications on micro partitions?
But for the time being, in the next lecture we'll be looking at the compute layer, which is the virtual warehouses in Snowflake.
Whenever you hear the term virtual warehouse, it actually refers to a compute cluster.
However, each of the virtual warehouse accesses the same shared data.
Next, we have the compute layer through which queries are executed on the stored data.
The compute engines in Snowflake are referred to as virtual warehouses.
The virtual warehouses are entirely independent of the storage.

each of the virtual warehouse accesses the same shared data.

Therefore, the architecture for Snowflake is often referred to as shared data.
Multi-cluster Architecture.
In the next lecture we will cover the cloud services, which is also a critical part of the snowflake architecture.

Storage layer – Micro partitions.

Each micro partition – 50 MB – 500 MB

So data in Snowflake is automatically organized into partitions known as micro partitions.
Compared to traditional static partitioning.
Micro partitions in Snowflake are managed automatically and don't require intervention by the user.
As the name suggests, micro partitions are relatively small and each micro partition will generally contain 50 MB to 500 MB of uncompressed data.
However, do note that the actual stored data is smaller as data in Snowflake is always stored with compression.

Micro partitions are added to a table in the order of how the data arrived in the table. So if additional data is added to a table, another micro partition or possibly multiple micro partitions depending on the size of the data, will be created to accommodate this data.
Snowflake.
Micro partitions are immutable, which means they cannot be changed once created. Any update to existing data or loading of new data into a table will result in new micro partitions being created.
Because micro partitions are immutable and any update or new data must be added into a new micro partition. Therefore, it is not necessary that similar partition values will always be in the same physical partition. So as an example, you'll see the data from the table on the left stored in multiple micro partitions
on the right.
You will also notice that the values in the two micro partitions overlap between the partitions.
You will see multiple partitions for the same column value and overlaps between partitions for different column values.
Because the data values are spread across multiple partitions.
Snowflake must keep track of what range of data is in which partitions so that it can use that information for efficient query processing.
Now Snowflake maintains several different kinds of metadata for a given table for this purpose.
It stores the range of column values in its metadata.
That is the maximum and minimum value for each column in each micro partition.
With this metadata information, Snowflake can intelligently decide which partitions to read when processing a query.

Similarly, it also stores the count of distinct values for each column in the metadata and certain other information to assist in query optimization.
Now, another important aspect is that within each micro partition, the data is stored in a columnar format.

So each column is stored compressed, and snowflake automatically determines the most appropriate and best compression algorithm.

Storing data in columnar format enables Snowflake to optimize queries even further when a subset of columns are accessed.

So consider this straightforward SQL example on the screen where we are querying the table on the left and only selecting the ID column.
We also added an ID equal to one where clause in the query.
So now Snowflake knows exactly which micro partitions contain the value one for the ID column.
And additionally, since we selected only one column, it will limit the processing to just that column
within its columnar format.
So the result is highly efficient access of a subset of stored data as highlighted on the screen.

So this was a bit of detail about the micro partitions in Snowflake.
We will come across micro partitions throughout this course, especially when we look into the cloning and the data sharing section.

Cloud Services Layer

So the cloud services is a highly available fault, tolerant, always on service.
So for any user connecting to Snowflake, whether via the Snowflake Web, UI or Snow SQL, their requests will go through the cloud services layer.
Metadata and management, security & governance, sharing, query parsing, query optimization, acid control.
Additionally, the Cloud Services layer provides transaction control or ACID compliance.
ACID stands for Atomicity, consistency, isolation and durability and a high level.
This term refers to the fact that a database system must allow for multiple transactions to execute in isolation, commit or rollback a transaction as a single unit, ensuring that a consistent state.

Compute layer – additional aspects of virtual warehouses.

A virtual warehouse in Snowflake is typically a multi node compute cluster.
A virtual warehouse provides resources such as CPU memory and temporary storage, which is used to process

Multiple virtual warehouses can be created for a given snowflake account.
But it is worth noting that each of them access the same shared data, hence the term multi clustered shared data.

Now virtual warehouse can be created in any of the available sizes.
For example, extra small, which is a one node cluster, small, which is a two node cluster, and then all the way up to four x large, which is a staggering 128 nodes.
Now Importantly, a customer can suspend a virtual warehouse if it is not in use or if it is not required for some time.
A suspended virtual warehouse does not consume any credits and therefore doesn't cost anything to the customer.
It is important to note here that when a virtual warehouse is requested to be suspended, it does not enter a suspended state until all active queries using that virtual warehouse have been completed.

A customer can also resize a virtual warehouse to meet the requirements of a changing workload.
For example, suppose the customer started out with a small size virtual warehouse.
In that case, it may be that the query workload complexity has increased significantly after a while.
So a small sized virtual warehouse is no longer enough to meet the demand.
In such cases, the virtual warehouse can be resized to a larger size to meet the increased workload complexity.

When a virtual warehouse is resized to a larger size, additional nodes are provisioned and added to the virtual warehouse compute cluster. It is important to note that the charge for the new size only takes effect. After all, the nodes in the virtual warehouse have been provisioned.

If the queries can execute efficiently and timely on a smaller virtual warehouse, keeping a larger sized virtual warehouse is basically wasting resources and incurring extra costs.
Therefore, scaling down the virtual warehouse size is a good option in such cases.

Snowflake pricing:

Storage cost
Compute cost
Serverless cost
Data transfer

So Snowflake charges a monthly fee to all customers for the data that they store.
The storage costs are calculated based on the average storage for each month.
And it's important to note that the storage costs are calculated after compression is applied on the Data.
Now with Snowflake, you can create a virtual warehouse of any size from extra small to four x large snowflake.
Credits are consumed based on the size of the virtual warehouse and are charged every second.
However, do note that there is a minimum of one minute charge when a virtual warehouse starts up.

How is Snowflake Costed:

Storage Cost – AWS S3 or Azure Blob storage based in actual usage , columnar compression and other techniques save cost. SF charges monthly fee to all the customers that they process data. The storage costs are calculated after the compression is applied.

Snowflake credits are consumed based on the size the virtual warehouse and are charged every sec.
Snowflake virtual warehouse can be created from extra small to 4x large.
Snowflake offers several serverless features for which the compute costs are charged. The serverless features doesn’t require any virtual wh so charged separately.

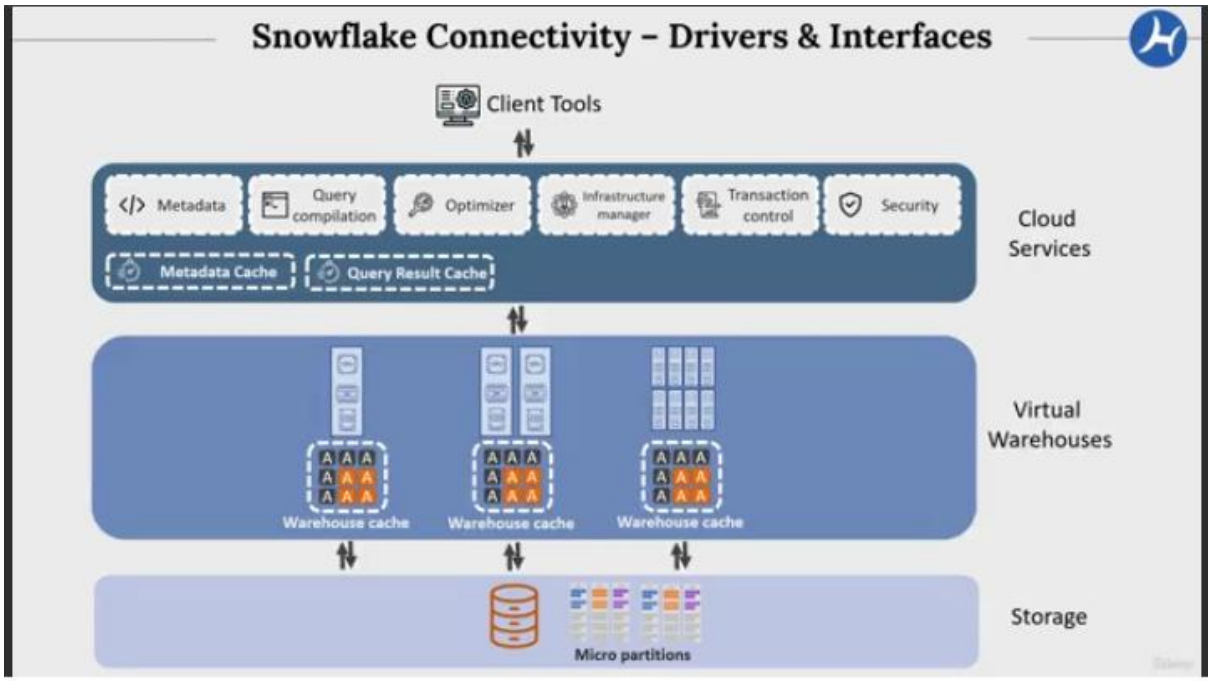
These serverless features includes: snowpipe, automatic clustering, database replication, materialized view, search optimization.

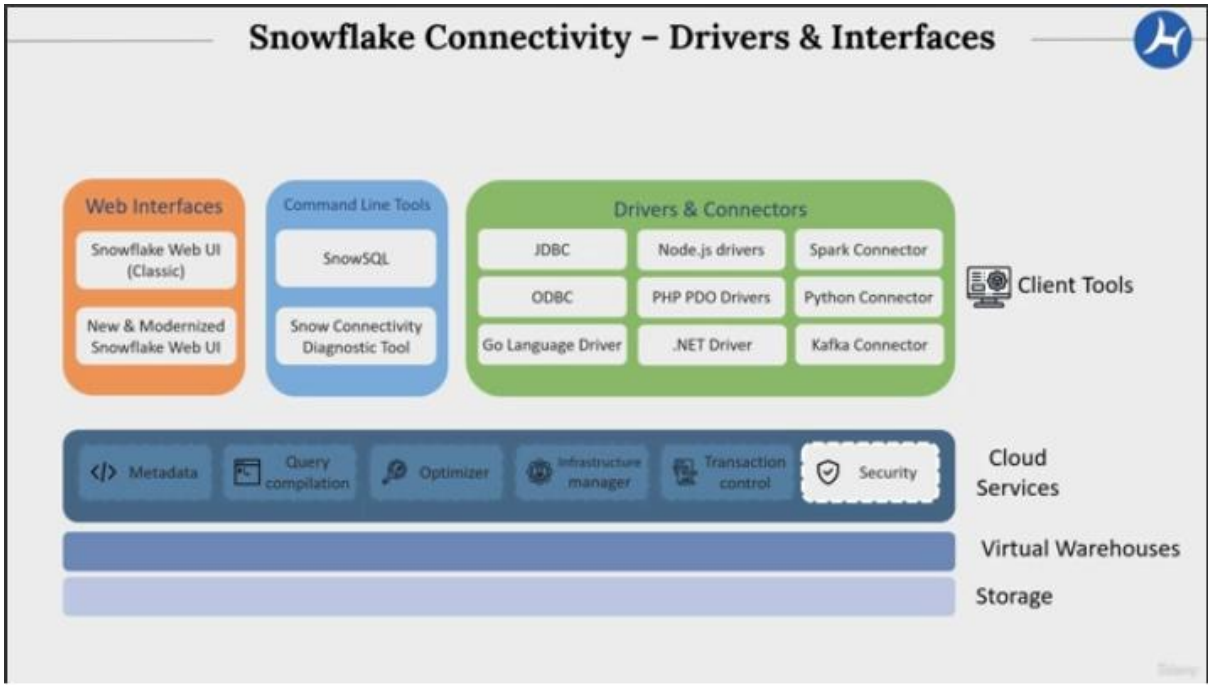
Another cost is data transfer costs: there are no costs for data transfer into the snowflake but its chargeable when its out to other regions and to other cloud platforms.

Cloud services:10% cloud services is free – database definitions, table definitions, users etc.,

Multi clustered shared data architecture -snowflake architecture.

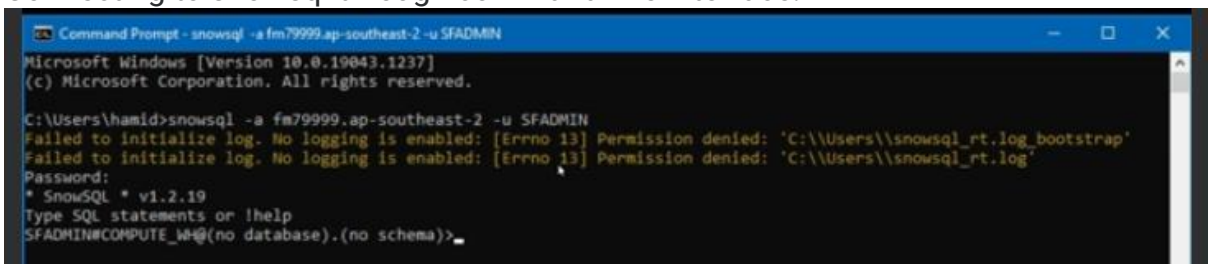
Snowflake interfaces and Tools: snowflake web interface will be discussed in this. The majority of tools and web interfaces.





Snowsql itself is available in linux, windows and mac OS.

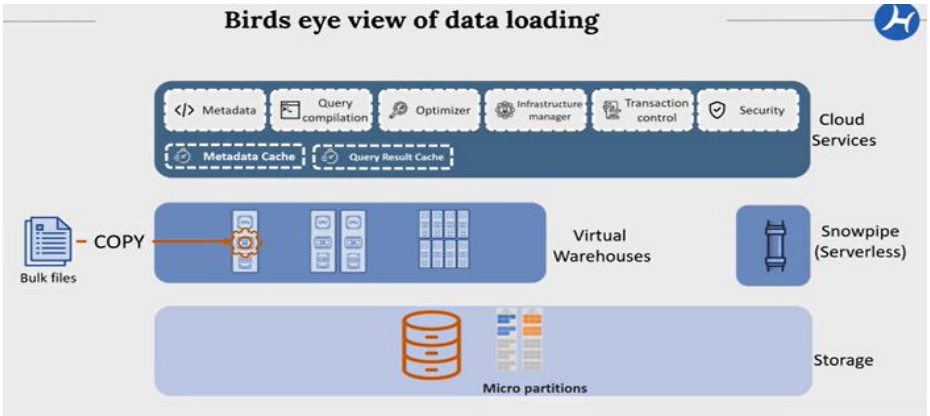
Connecting to snowsql through command line interface:



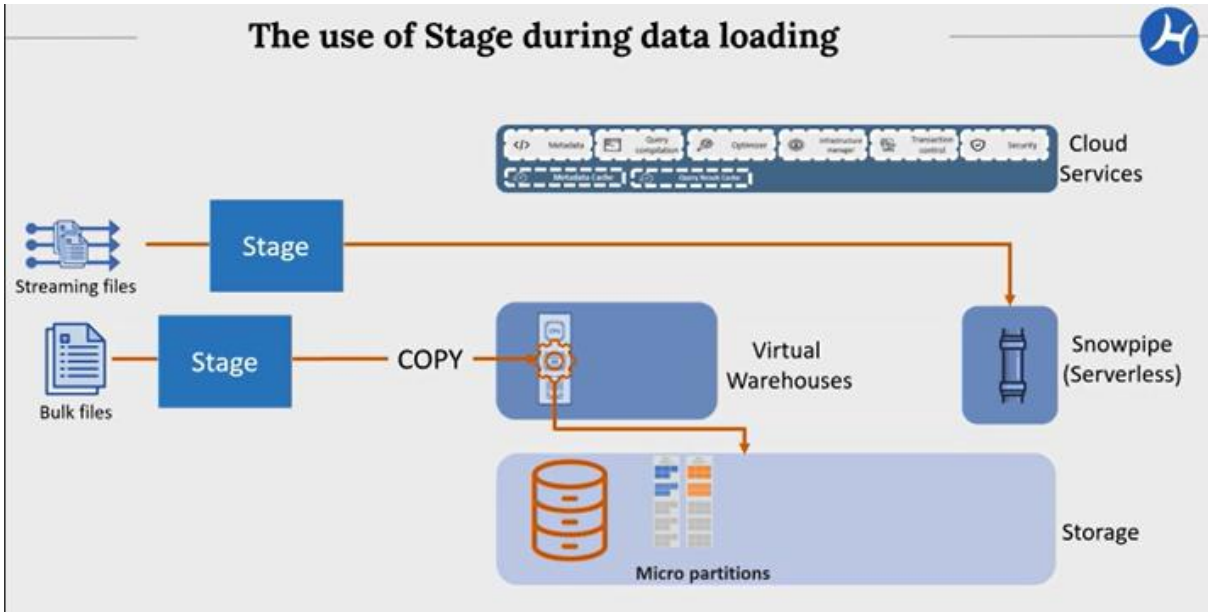
Snowsight is a modern and lightweight web interface using new technologies and is a primary method of interacting with your Snowflake instance.

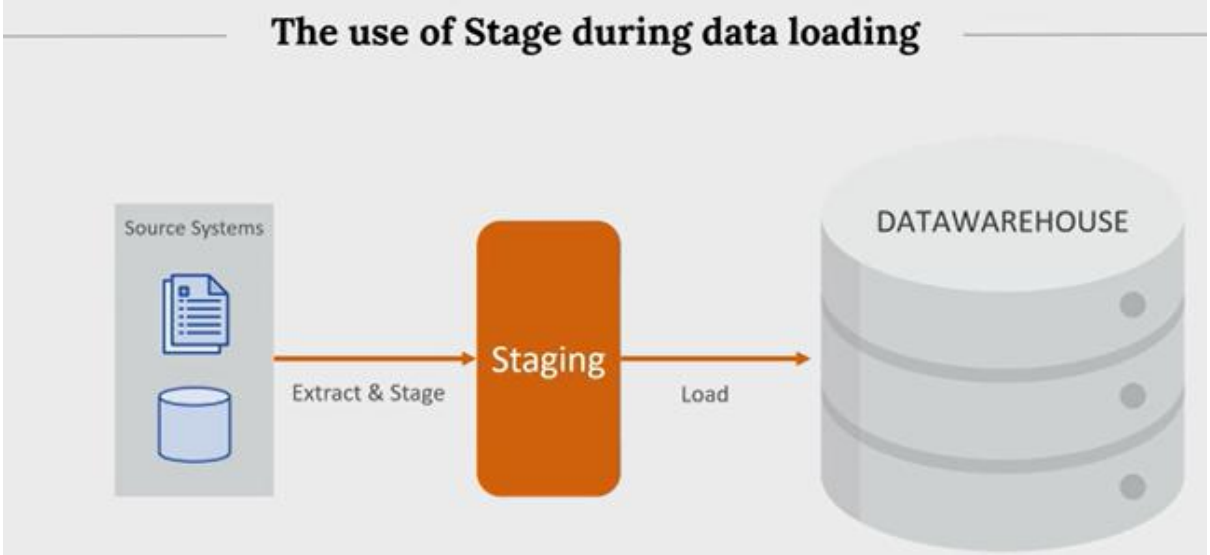
Snow SQL connects to Snowflake through the command line and executes SQL queries on your Snowflake instance. Snow SQL is available for Linux, Windows, and Mac OS.

High level view of data loading in Snowflake

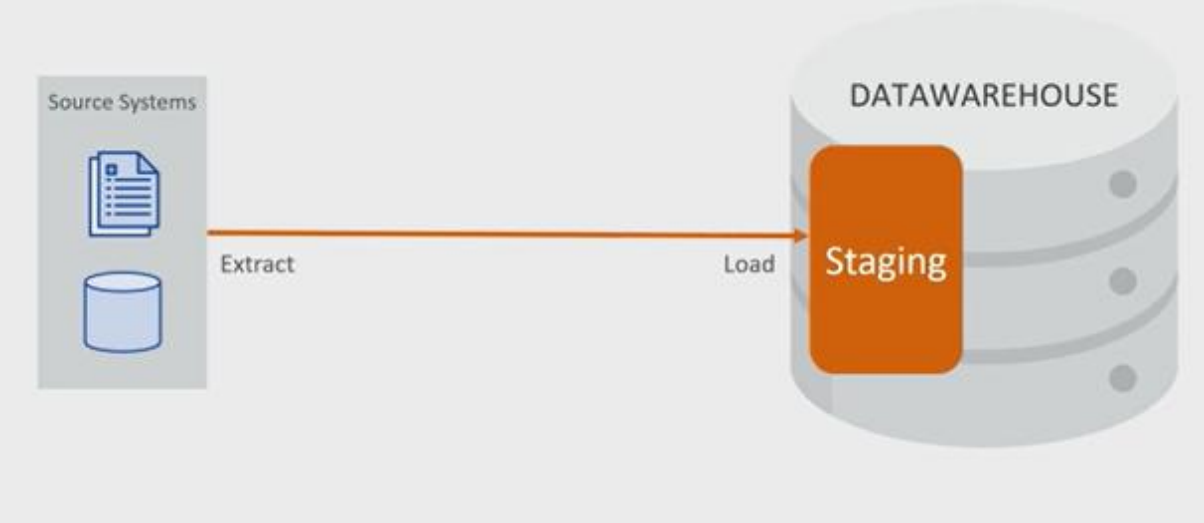


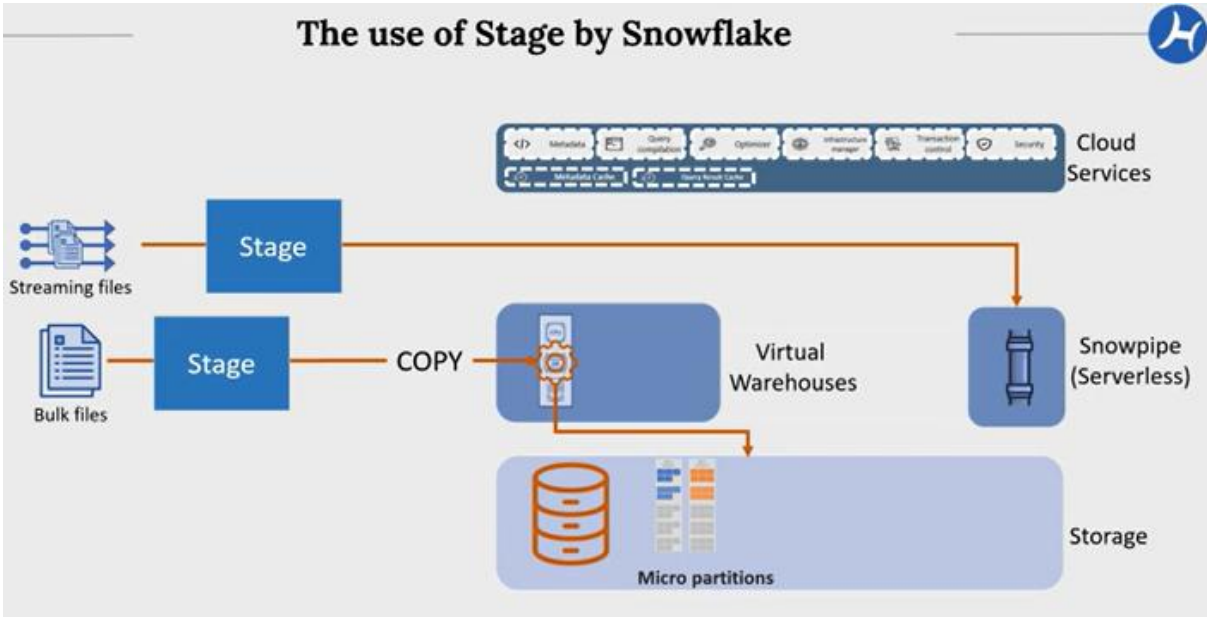
COPY and snowpipe are 2 methods to load the data into snowflake.





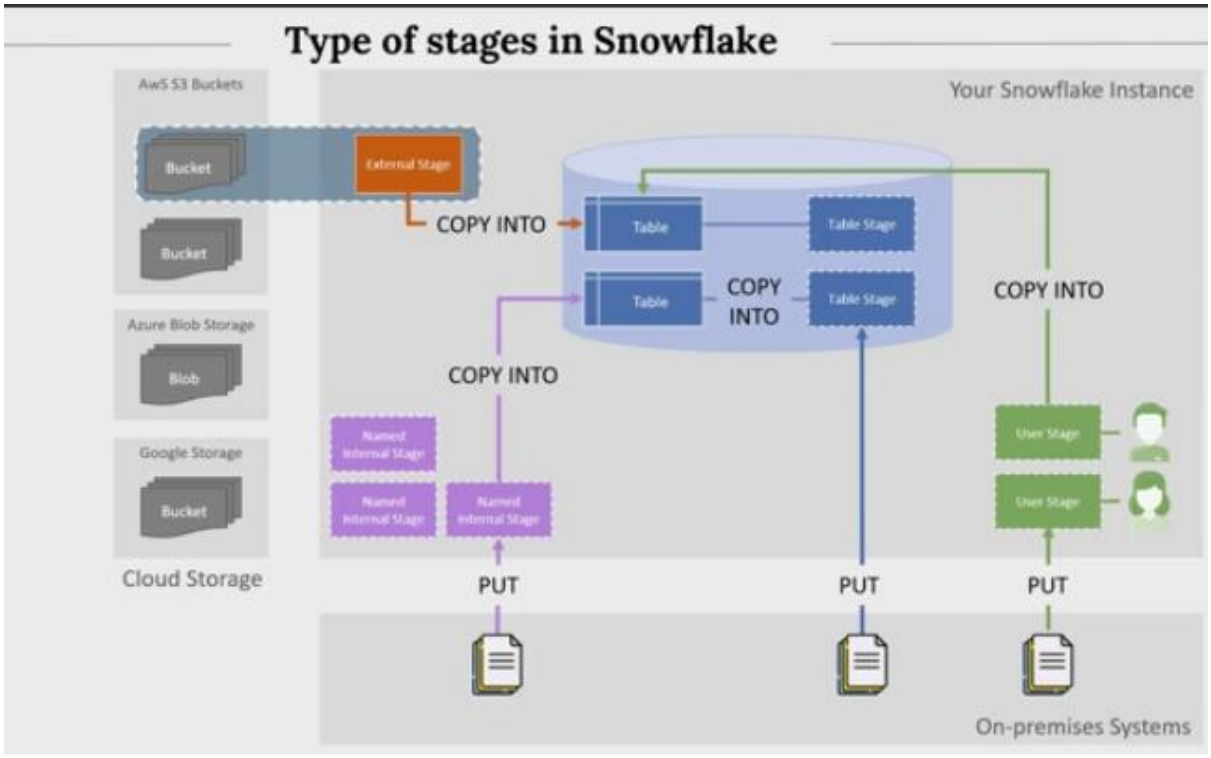
Alternatively, the staging can be inside the DWH as shown below:



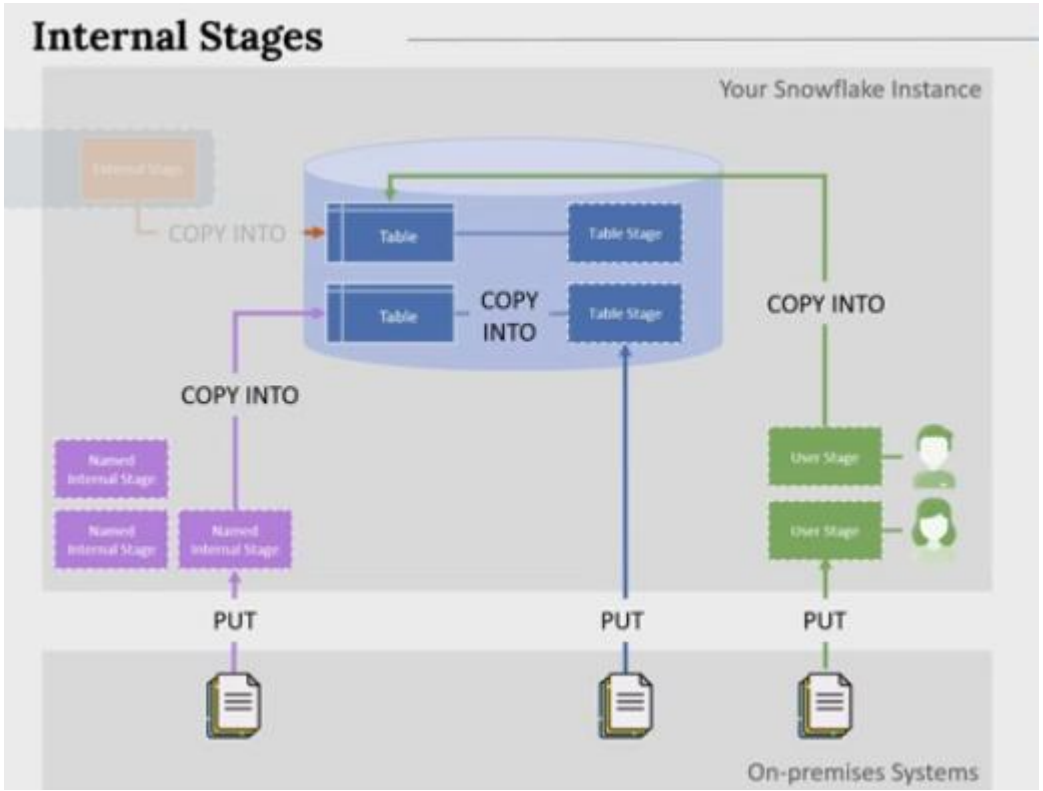


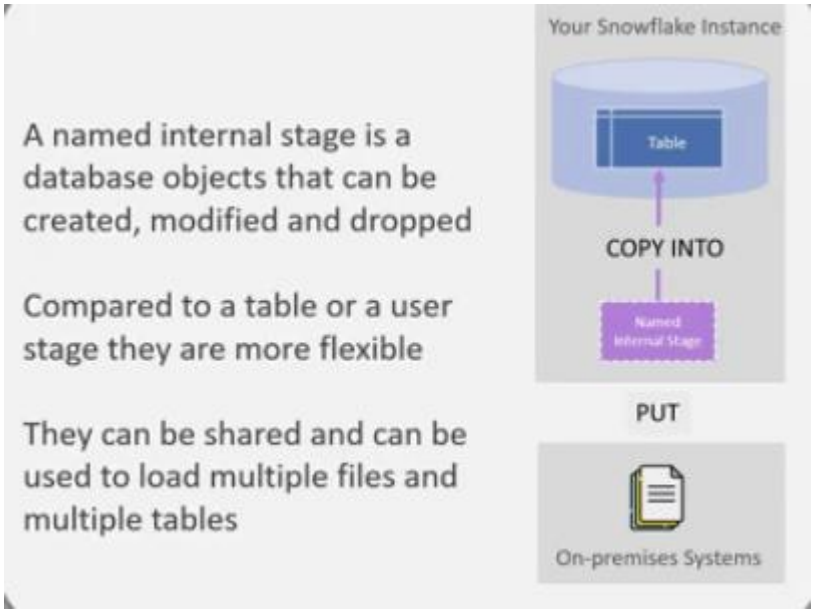
Role of STAGE in Data Loading:

Types of stages in snowflake: - external and internal stages.



External stage, named internal stage, user stage.

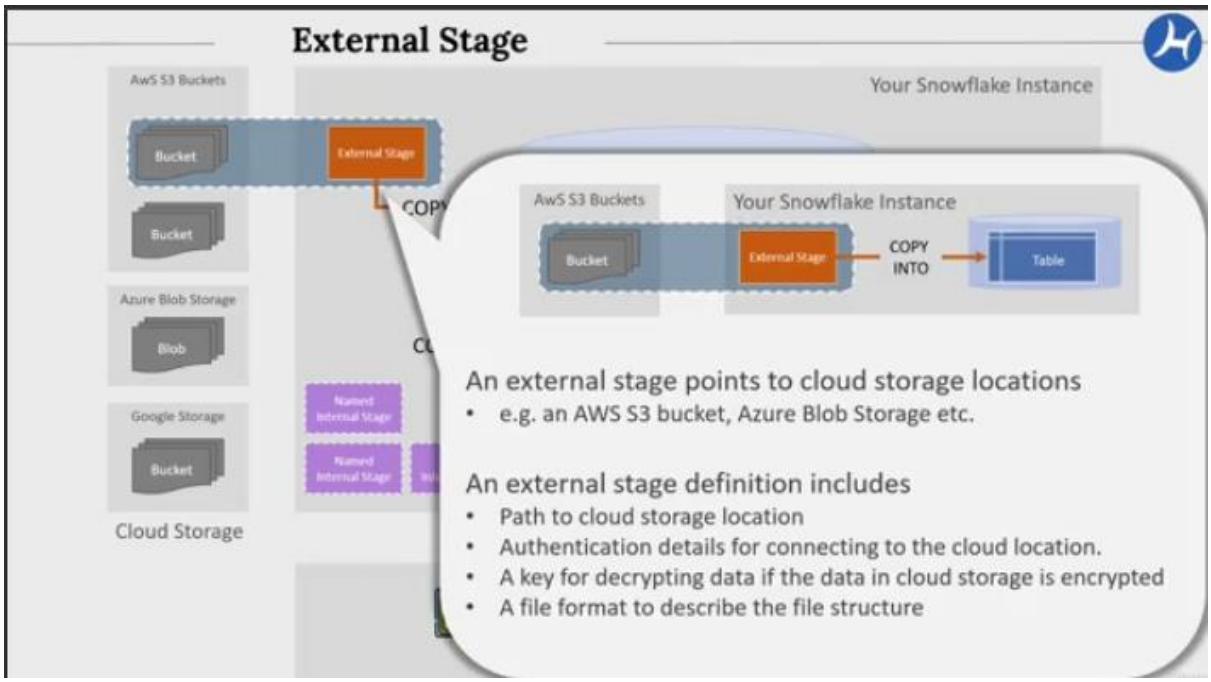


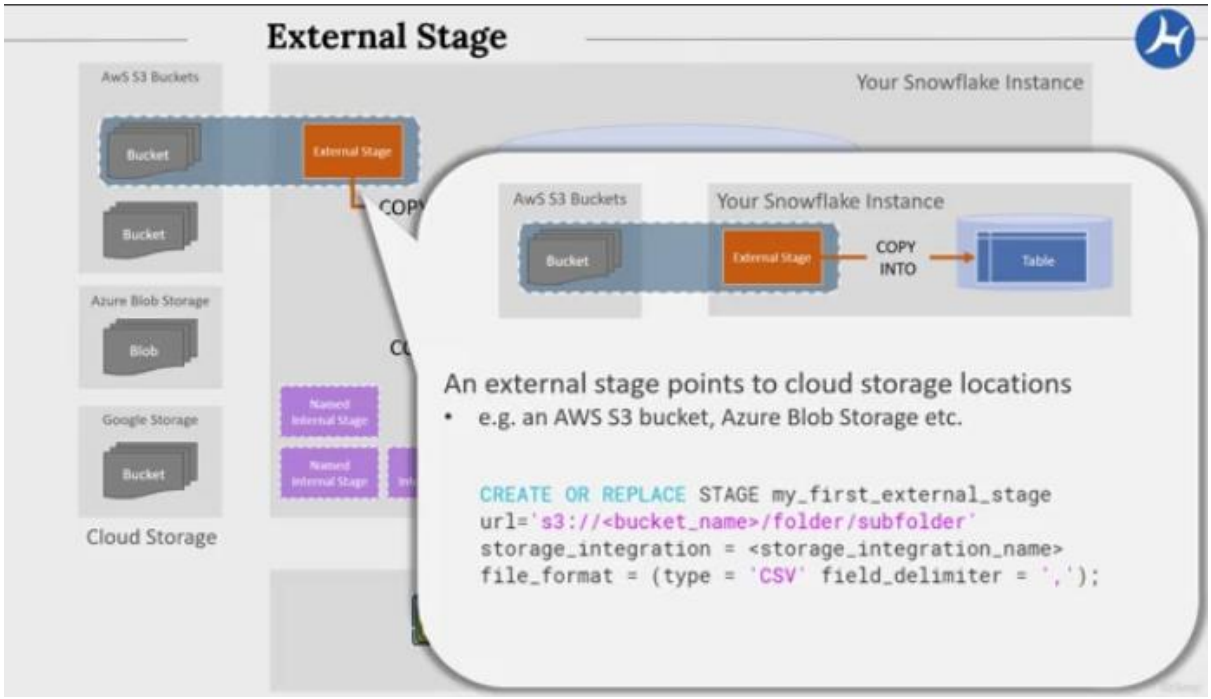


Loading on premises data via a named internal stage:

Hands on : LOADING DATA USING NAMED internal stage

External stage: points to cloud storage locations.



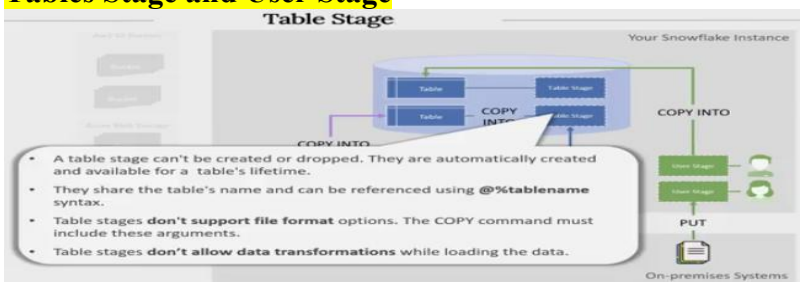


Snowflake can securely connect to a cloud storage location.

Hands on: Loading data using an external stage

http link doesn't work in external stage definition.
While creating the external stage we will have authentication and authorization as well.

Tables Stage and User Stage



Basic Data transformations:

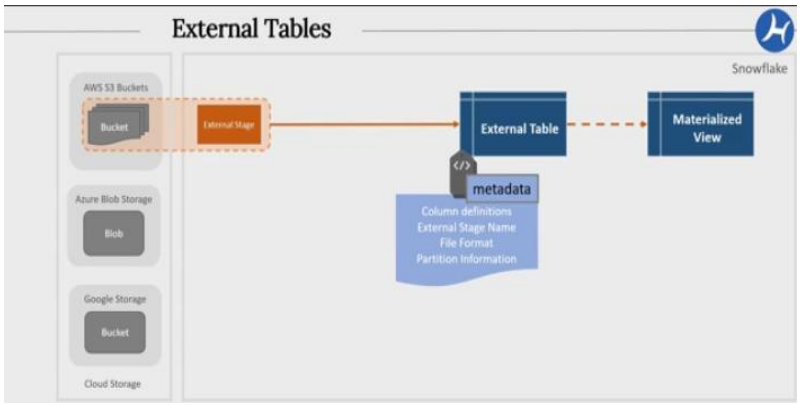
Basic Data Transformations

- Basic transformations allowed during ingestion using COPY
  - Omit column or columns
  - Change the order of columns
  - Cast columns into specific data types
  - Truncate
- Transformations not supported during COPY
  - Joins
  - Filters i.e. WHERE clause
  - Aggregation functions

External tables: an alternate to data loading

External Tables

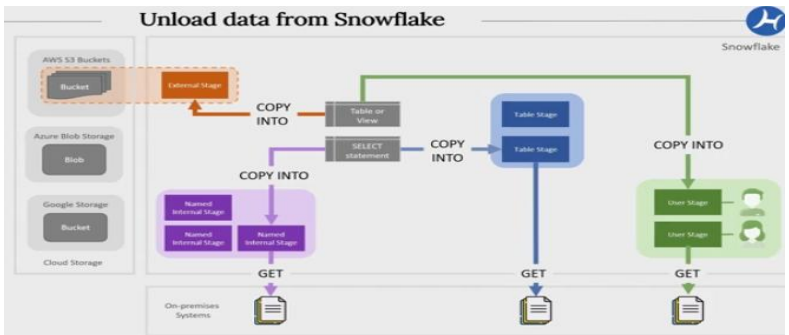
- External tables allow you to create tables on data stored external to Snowflake
  - Definition in Snowflake metadata
  - Data in external location
- External table functionality lets you query an external table like a regular table
  - Join with other tables
  - Create views on top of external tables
- External tables are read-only since they point to external storage.
  - DML operations cannot be performed on them.
- External table data does not contribute to Snowflake storage costs
  - the compute cost is charged when queries are executed



Materialized views can improve query performance for external tables.
MV's on external tables don't refresh automatically.

Unloading or exporting data from snowflake:

Unloading data or exporting data is almost the same as loading. It uses the same copy mechanism and the concept of stages. Unload to different file formats.
Compress and encrypt while unloading.



The GET command is used to download data from an internal stage to an on-premises system. The PUT command uploads data from an on-premises system to an internal stage. To download or upload data to an external stage, cloud provider utilities or other tools are used to interact with data in the cloud storage pointed to by the external stage.

The load metadata stores a variety of information, such as the name of every file that was loaded into that table and the time stamp corresponding to the time that a file was loaded. By utilizing this load metadata, Snowflake ensures that it will not reprocess a previously loaded file. The load metadata expires after 64 days. Snowflake skips over any older files for which the load status is undetermined.

The COPY command allows unloading or exporting data from a table or a view and also allows using queries (SELECT) to unload data.

When loading data into a table using the COPY command, Snowflake allows you to do simple transformations on the data as it is being loaded by using a SELECT statement. During the load process, the COPY command allows for modifying the order of columns, omitting one or more columns, and casting data into specified data types. It is also possible to truncate data using the COPY command if it is larger than the desired column width.

Snowflake offers an alternative approach for tables called external tables, which permits the creation of tables with data stored in external cloud storage. External tables remove the need for the data to be loaded into Snowflake. In the case of an External table, the definition of the table is still stored in Snowflake metadata and consists of table structure, file locations, filenames, and other attributes. However, the table's data is saved outside of Snowflake. The external table functionality enables you to query external data like a standard table. External tables may be joined to other

COPY command uses virtual warehouse resources. Snowpipe is billed separately and does not use virtual warehouse resources. Snowpipe is serverless and has its own computational capability; therefore, it does not rely on virtual warehouses for processing. Snowflake automatically manages the compute required by a Snowpipe. Snowflake also manages the scaling up and down of a Snowpipe as per the data load requirement. Since a Snowpipe is serverless, its costs are charged separately from virtual warehousing fees.

Snowflake allows continuous data loading using Snowpipe, a serverless service. Snowpipe enables you to load data in a micro-batch manner, loading small volumes of data on each execution. The micro-batch-based data loading is used when a continuous stream of data, such as transactions or events, must be loaded and made available to enterprises quickly. Snowpipe enables continuous data loading and can load data within a few minutes after it arrives in a stage. Snowpipe is serverless and has its own computational capability; therefore, it does not rely on virtual warehouses for processing.

Snowflake allows continuous data loading using Snowpipe, a serverless service. Snowpipe enables you to load data in a micro-batch manner, loading small volumes of data on each execution. The micro-batch-based data loading is used when a continuous stream of data, such as transactions or events, must be loaded and made available to enterprises quickly. Snowpipe enables continuous data loading and can load data within a few minutes after it

arrives in a stage. Snowpipe is serverless and has its own computational capability; therefore, it does not rely on virtual warehouses for processing.

When loading data into a table using the COPY command, Snowflake allows you to do simple transformations on the data as it is being loaded. During the load process, the COPY command allows for modifying the order of columns, omitting one or more columns, casting data into specified data types, and truncating values. While loading the data, complex transformations such as joins, filters, aggregations, and the use of FLATTEN are not supported as they are not essential data transformations. Therefore, joining, filtering, and aggregating the data are supported ONLY after the data has been loaded.

Continuous Data protection

This offers features that protect data in snowflake env without compromise.
This offers data encryption at rest and while in transit. Multi factor authentication is part of the authentication mechanism.

In addition, Snowflake is also equipped with a number of cutting edge features that assist in safeguarding of data and its subsequent recovery in the event that a human makes a mistake.

So you can restore data that has been mistakenly changed or deleted by utilizing two features time travel and UN drop.

In this section, we'll cover some aspects of continuous data protection in Snowflake, including time travel, failsafe, storage and drop functionality, and also covers the concept of transient and temporary

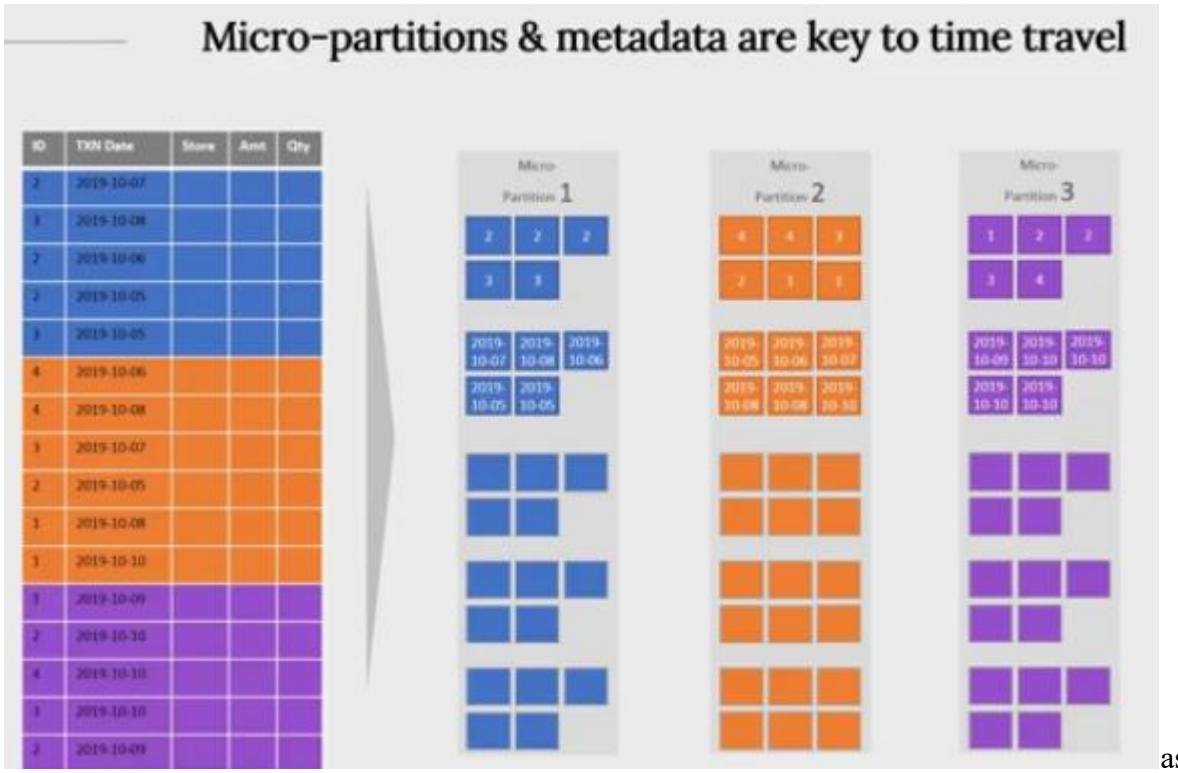
Back to the Future with Time travel

So time travel in Snowflake is a cutting edge data protection feature that enables users to query, retrieve and recover historical data from tables in case of loss of data when it has been mistakenly changed or deleted as a result of a human error. You want to restore data as painlessly and as quickly as possible.

Before the time travel functionality was introduced by Snowflake. The most common method for recovering from the inadvertent loss of data was to restore the lost information from a previous backup.

How time travel works in Snowflake:

Snowflake stores the data in its own format using the micro partitions.



as new data added to the table new micropartitions are added to the same. The micropartitions are immutable.

Micropartitions and metadata are key to time travel. And this immutability of the micro partition has implications for the updates and deletes performed on a table.

The example shows that two rows from a table were deleted because micro partitions are immutable.

Snowflake cannot simply update the data in micro partition number two, but rather it will create a new micro partition marked as partition number four on our example. So it is essential to note that deleted micro partitions that have been marked as deleted still exist.

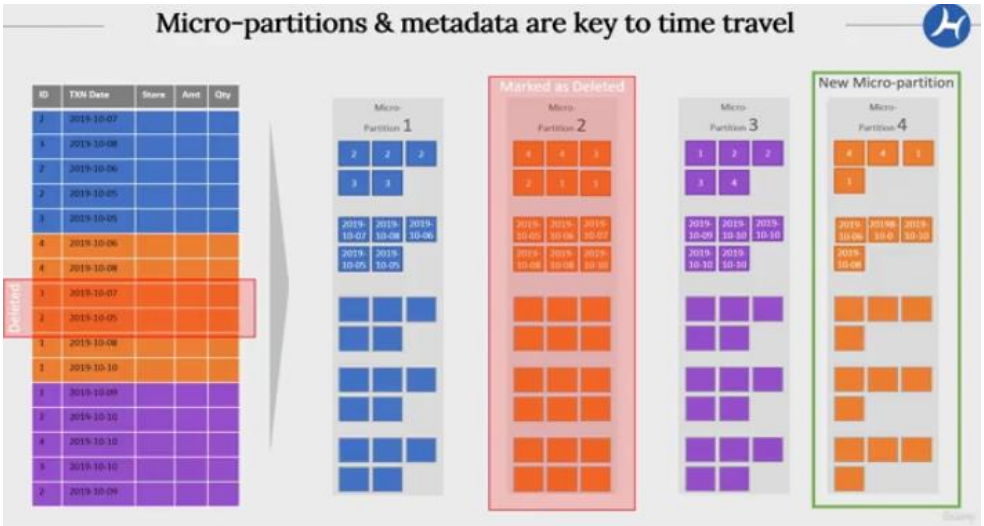
physically on the disk and they can be read if necessary. And this lays the groundwork for snowflake's time travel functionality.

So when users request to read data from a table using time travel extension and they want to read data as it existed at a specific point, Snowflake reads data from deleted historical partitions to fulfil those time travel queries.

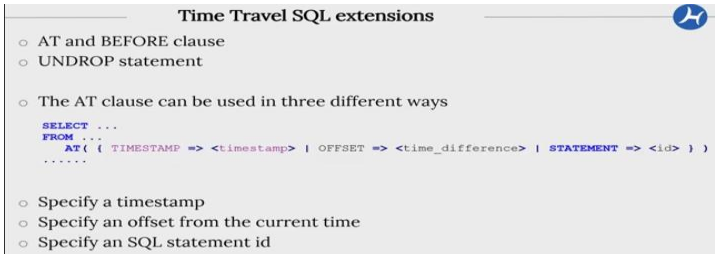
Snowflake retains these historical deleted partitions for a specific period of time before purging them altogether. Until these partitions are purged, any ordinary user can access the data contained in these historical partitions, and the period for which these historical partitions are retained is known as the time travel duration.

The time travel duration generally varies from one day to 90 days depending on the snowflake edition that you may have. So this was basically behind the scenes working of time travel. In the next lecture, we will discuss the time travel extensions and how to use them to modify existing queries to use time travel.

The new micro partition has all the data from the deleted micro partition except the two rows that were deleted. At the same time, the micro partition number two is marked as deleted. So it is essential to note that deleted micro partitions that have been marked as deleted still exist physically on the disk and they can be read if necessary.



Time travel SQL extensions:



Snowflake also supports UN drop statement, which can be used to recover tables, schemas or even complete databases after they have been dropped. So the add clause is usually used with select statements and supports three different ways of accessing historical data.

Back to the future with time travel:

Back to the Future with Time Travel

```
SELECT * FROM STORE
BEFORE(TIMESTAMP => '<timestamp>'::timestamp_ltz);

SELECT * FROM STORE
AT(TIMESTAMP => '<timestamp>'::timestamp_ltz);

DELETE FROM STORE;

SELECT COUNT(*) FROM STORE;

SELECT * FROM STORE
BEFORE(STATEMENT => '<query id>');

SELECT * FROM STORE
AT(STATEMENT => '<query id>');
```

Undrop with Timetravel:

Undo with Time Travel

```
UNDROP TABLE STORE;

SELECT COUNT(*) FROM STORE;

DROP DATABASE timetravel_demo;

USE DATABASE timetravel_demo;
SELECT COUNT(*) FROM timetravel_demo.public.STORE;

UNDROP DATABASE timetravel_demo;
```

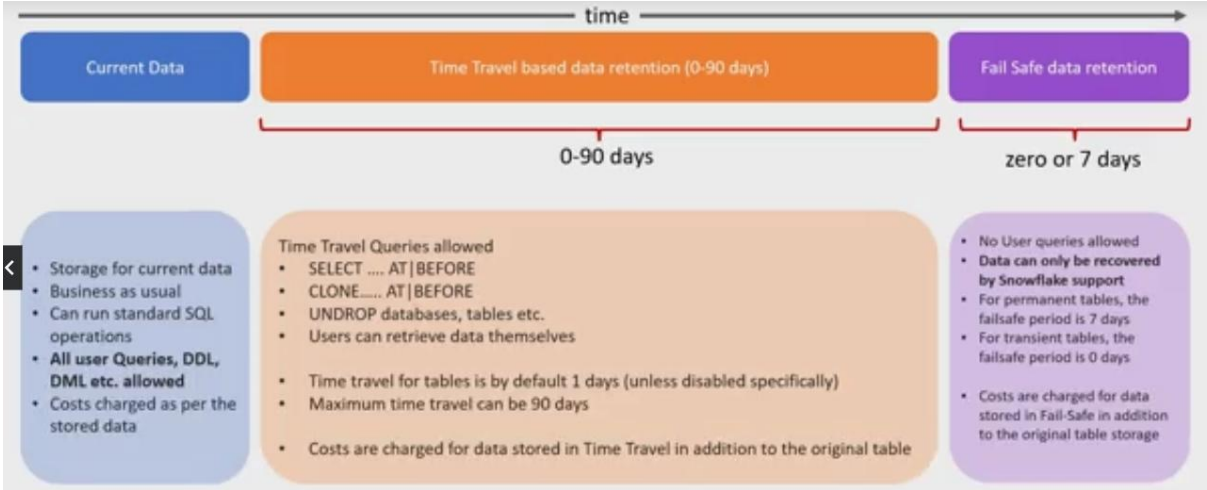
Undo is nothing but how we do ctrl+z. it's the same feature in snowflake.

Failsafe Storage:

- Failsafe provides a 7-day period during which historical data is recoverable by Snowflake
- The failsafe period starts immediately after the time travel period ends
- The failsafe data can be recovered only by the Snowflake support
- Provides a way to recover lost data in a relatively quick manner
- Snowflake's redundant & cheap data storage architecture reduces the need for backups, there still can be scenarios for data loss
- Failsafe Storage provides a safety cushion in such scenarios and allows you to recover data 7 days after even your time travel history is gone

The data in failsafe storage can be accessible by snowflake support team so we can contact the snowflake support team to recover the data from failsafe storage.

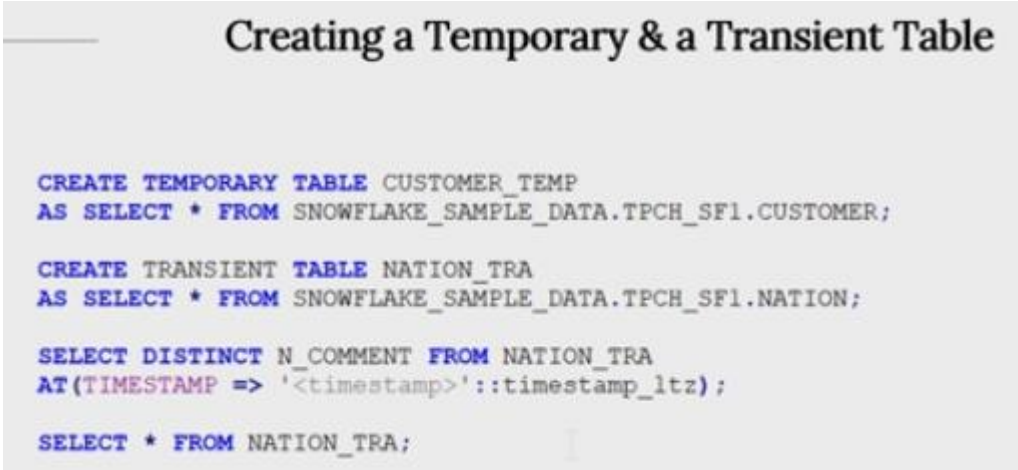
Time travel and failsafe durations:



Snowflake charges for data stored in failsafe and Time travel storage.

Types of Tables in Snowflake:

- **Temporary**
  - Exist only for the lifetime of a session.
  - They are not visible to other sessions and are removed immediately once the session ends
- **Transient**
  - Similar to temporary tables, but they persist between sessions. They are designed to hold temporary data that needs to be accessed across sessions e.g. ETL jobs
- **Permanent**
  - This is the default type of table in Snowflake, If you don't specify the type, a table is created as Permanent.



Depending on the Snowflake edition, the Time Travel duration might range from 1 to 90 days. The Standard edition allows for one day of Time Travel. Time Travel is possible for up to 90 days in the Enterprise version and above.

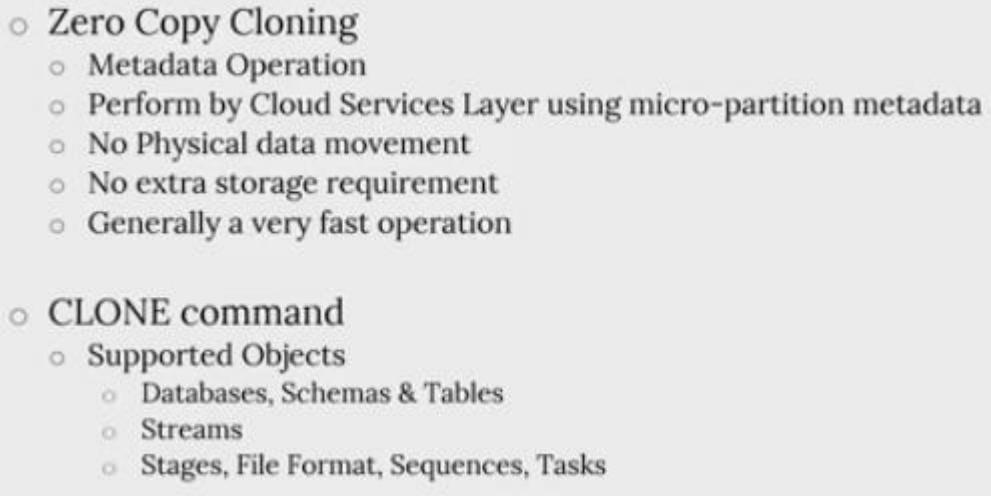
In addition to protection provided by Time Travel, data that has been modified also goes through a failsafe period. Failsafe storage is intended to provide an extra layer of protection against data loss caused by human error. Once the Time Travel period ends, Snowflake keeps the data for a further 7-day period as further protection. When data is in failsafe storage, ordinary users cannot access it; only Snowflake support employees can access and recover it if the customer requests it.

Time Travel SQL extensions allow you to see data as it existed before or at a particular time. It can also be used to see data before an SQL statement is executed or at the point when an SQL statement is run. Time Travel does not let you recover data for more than 90 days in the past.

To support Time Travel queries, Snowflake supports special SQL extensions. It supports the AT and BEFORE statements which can be used with SELECT statements or while cloning tables, schemas, and databases. Snowflake also supports the UNDROP statement, which can be used to recover tables, schemas, or even complete databases after they have been dropped.

Cloning in Snowflake:

Intro to zero copy cloning:

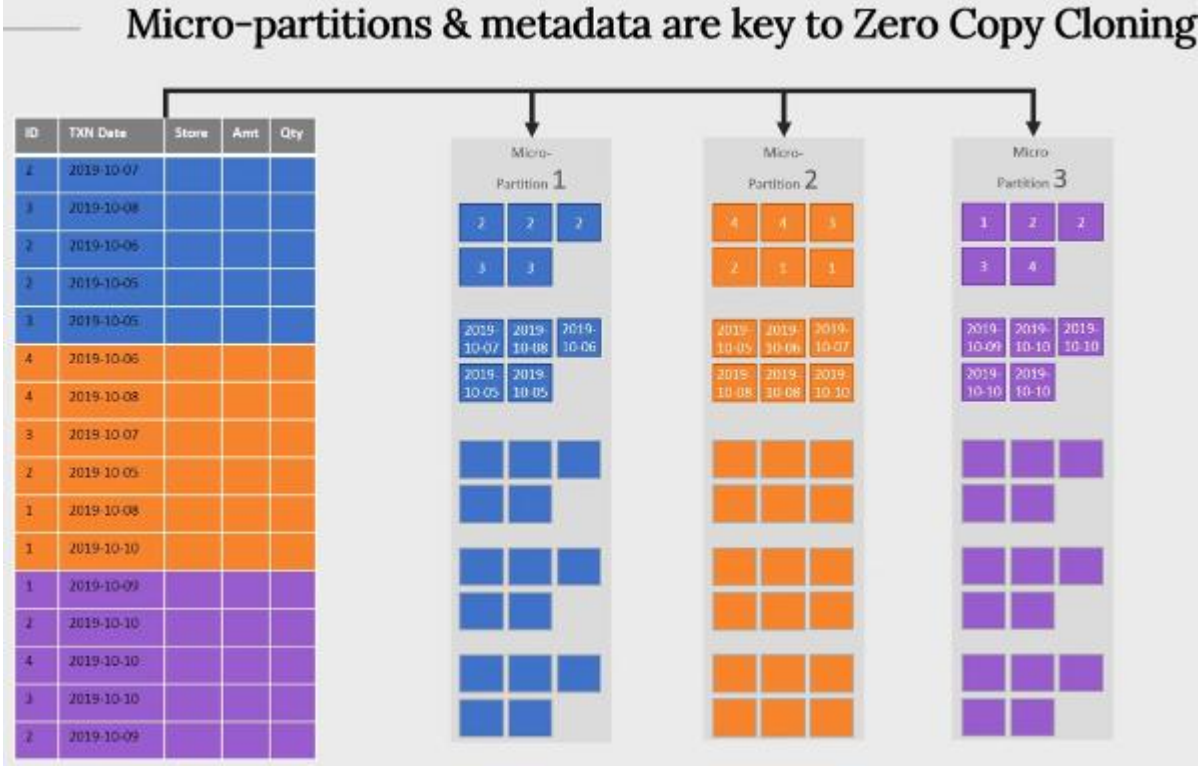


How cloning works in snowflake:

Micro partitions and metadata are key to zero copy cloning.

So Snowflake divides data in a table into several micro partitions, and the metadata in the cloud services layer tracks micro partitions corresponding to a table.

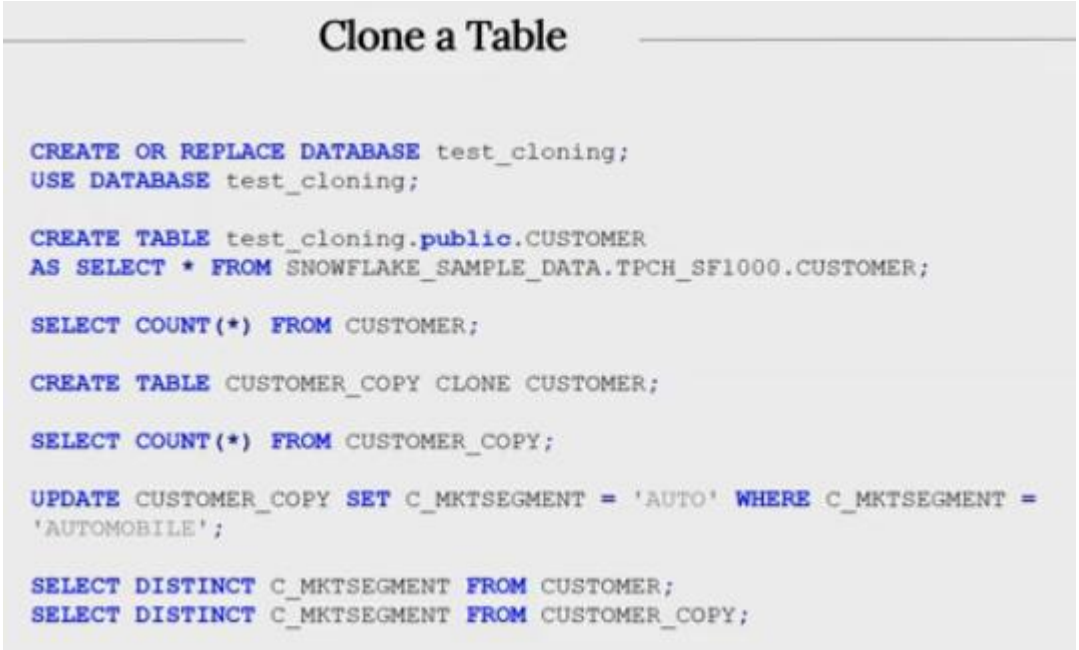
As new data is added to a table, new micro partitions are produced. So this metadata in Snowflake's Cloud services layer maintains information on which micro partitions belong to which table, and also other information such as if a micro partition is marked as deleted, or if it is in time travel storage or in fail safe storage.



Any updates to the table will result in addition of new micro partition.

Cloning of a table is bit faster than the copy operation.

Cloning a table in snowflake:



cloning of a table, demonstrating how much faster the operation is compared to a standard copy operation.

Sample queries for cloning a table:

```
--We start by creating a new database called test_cloning.
CREATE OR REPLACE DATABASE test_cloning;
USE DATABASE test_cloning;

--Next, we create a table that will be used to demonstrate th
we have purposedly chosen a table approximately 10GB in size.
the create table as (CTAS) expression to create this table wh
table with the same name in the test_cloning database.
CREATE TABLE test_cloning.public.CUSTOMER
AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.CUSTOMER;

--Let us find out what is the count of rows in this table. Yo
SELECT COUNT(*) FROM CUSTOMER;

--Now we clone the CUSTOMER table to another table called CUS
as a clone of another table. The following CLONE statement sh
CREATE TABLE CUSTOMER_COPY CLONE CUSTOMER;

--Let us find out the count of rows in the cloned table. You
same data.
SELECT COUNT(*) FROM CUSTOMER_COPY;
```

The distinct queries from both the tables will be different as we have updated some data in the copy table.

Clone a complete database and a schema:

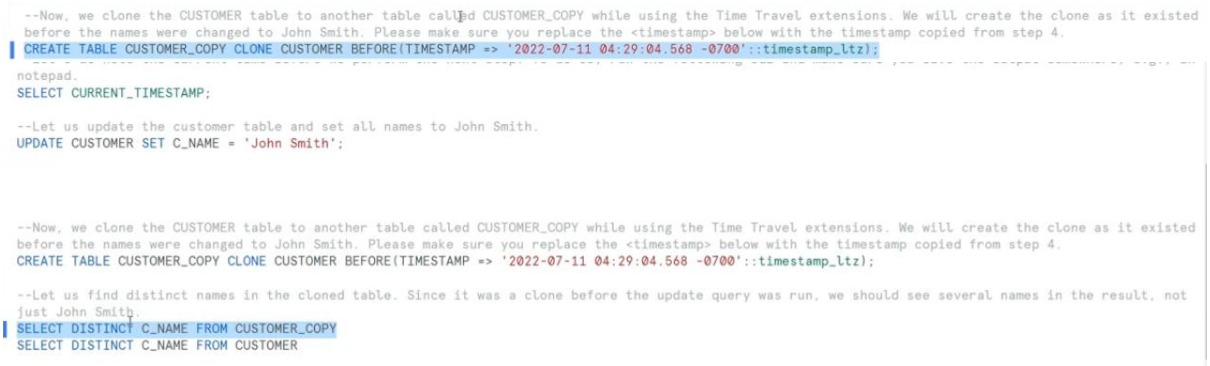
Clone whole schema & whole database

```
CREATE DATABASE SAMPLE_DATA_SUBSET;
USE DATABASE SAMPLE_DATA_SUBSET;

CREATE SCHEMA SAMPLE_SCHEMA1;
USE SCHEMA SAMPLE_SCHEMA1;
CREATE TABLE CUSTOMER AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER;
CREATE TABLE NATION AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.NATION;
CREATE TABLE REGION AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.REGION;

CREATE SCHEMA SAMPLE_SCHEMA2;
USE SCHEMA SAMPLE_SCHEMA2;
CREATE TABLE DATE_DIM AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.DATE_DIM;
CREATE TABLE STORE AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL.STORE;

CREATE DATABASE DEMO_SCHEMA_CLONING;
USE DATABASE DEMO_SCHEMA_CLONING;

CREATE SCHEMA MY_SAMPLE_SCHEMA CLONE SAMPLE_DATA_SUBSET.SAMPLE_SCHEMA1;

CREATE DATABASE DEMO_DATABASE_CLONING CLONE SAMPLE_DATA_SUBSET;
```

Cloning with Time travel:
This is how time travel can be done with cloning:

Cloning with Time Travel

```
CREATE OR REPLACE DATABASE test_cloning;
USE DATABASE test_cloning;

CREATE TABLE test_cloning.public.CUSTOMER
AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.CUSTOMER;

SELECT COUNT(*) FROM CUSTOMER;

CREATE TABLE CUSTOMER_COPY CLONE CUSTOMER;

SELECT COUNT(*) FROM CUSTOMER_COPY;

UPDATE CUSTOMER_COPY SET C_MKTSEGMENT = 'AUTO' WHERE C_MKTSEGMENT =
'AUTOMOBILE';

SELECT DISTINCT C_MKTSEGMENT FROM CUSTOMER;
SELECT DISTINCT C_MKTSEGMENT FROM CUSTOMER_COPY;
```



Cloning is a metadata operation in which no actual copying of the data occurs. A snapshot of the data in the object being cloned is captured and made available in the cloned object. The cloned table's metadata references the existing micro-partitions at the time of the snapshot.

Cloning is achieved through metadata operation performed in the cloud services layer. Data is not physically copied, nor are new micro-partitions created—instead, the cloned table points to the micro-partitions of the source table.

Combining Cloning and Time Travel can generate a clone of a table, database, or schema as it existed at a specific point in time. Because both Time Travel & Cloning are metadata operations, they can easily be combined.

The zero-copy cloning capability of Snowflake enables users to create clones of tables, schemas, and databases without physically copying the data. Cloning does not require additional storage space, and because cloning does not physically replicate data, it is far faster than the physical copying of data. Micro-partitions and metadata enable rapid and efficient zero-copy cloning because the cloned table's metadata references the existing micro-partitions.

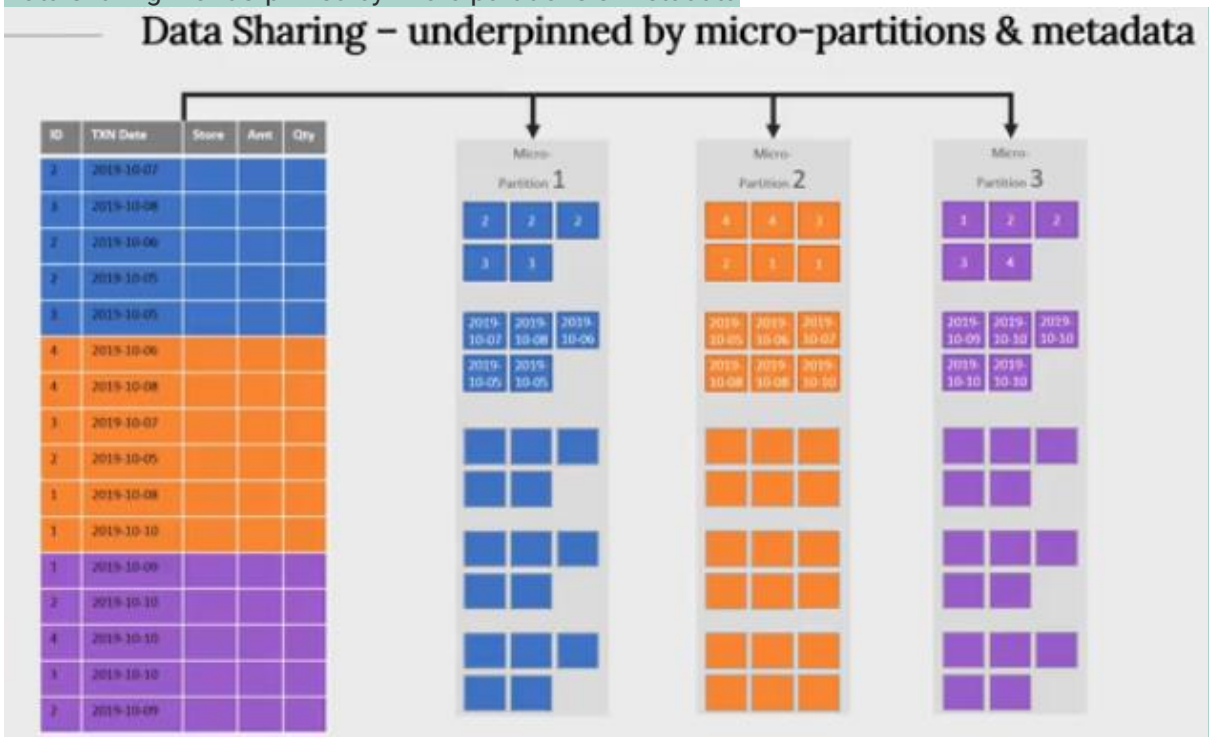
When tables, schemas, or databases are cloned, the cloned item does not contribute to total storage until data manipulation language (DML) operations are performed on the source or target, which modify or delete existing data or add additional data.

A cloned object does not contribute to overall storage until DML operations on the source or target object are done.

Data sharing in snowflake:
Secure data sharing in snowflake:

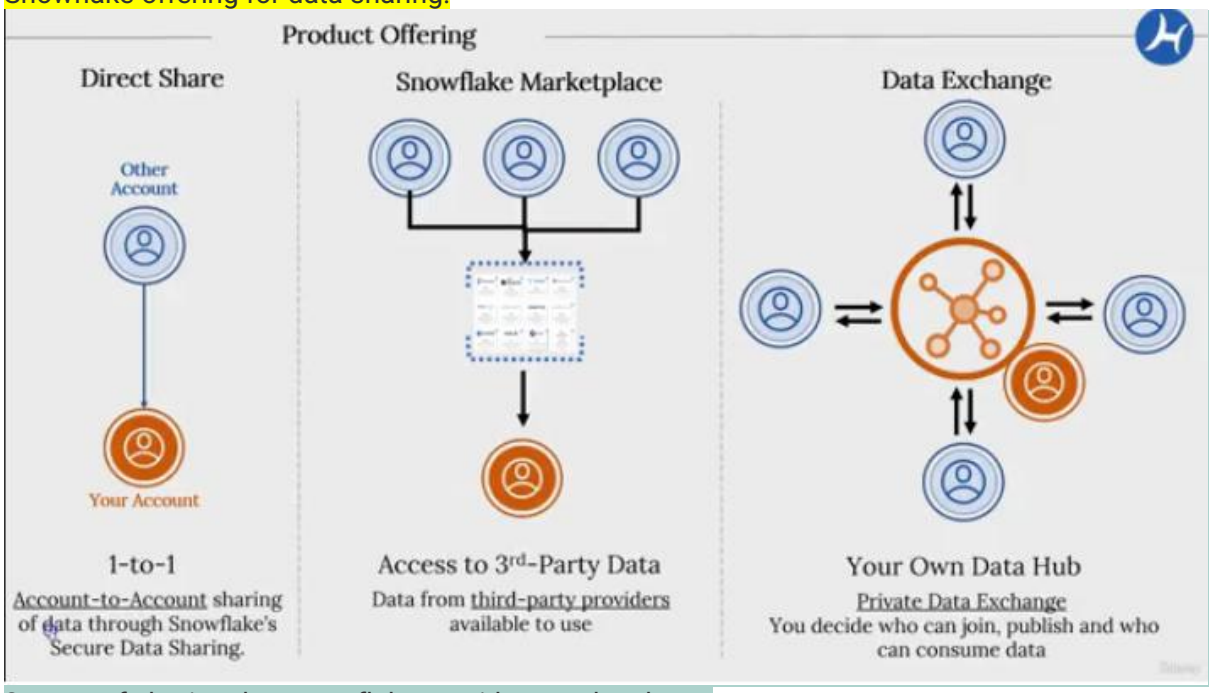
There is no storage fee for data sharing.

How data sharing works:
Data sharing – underpinned by micro partitions & metadata



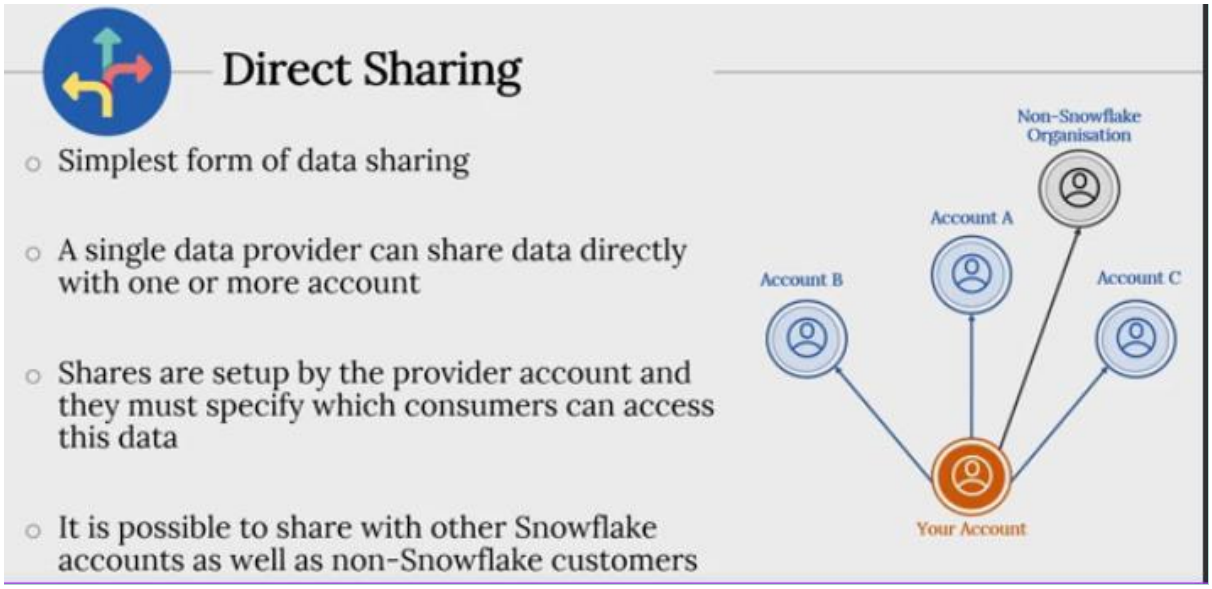
Shared table refers to the underlying table and its micro partition. Sharing is faster and any changes in data in table will be instantly reflected in the shared table.

Snowflake offering for data sharing:

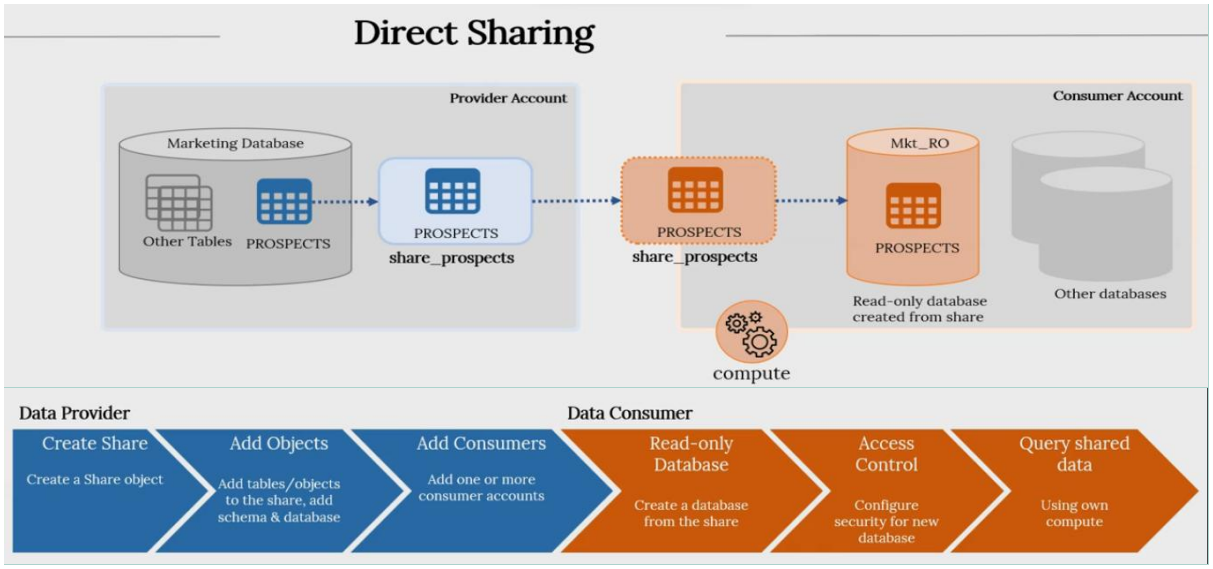


3 types of sharing that snowflake provides are the above:

Direct share, snowflake market place and data exchange.



Direct sharing:



Virtual private snowflake accounts (VPS) are exception for data sharing. They cant share the data.
Th e data sharing process requires steps to share with another consumer.

Steps in blue are performed by the data provider.
So in secure data sharing, the data sharing process starts with the provider account, creating a share object.
So one way to think about a share object is as a container that stores all of the information that is necessary to enable sharing.
So each object contains information on objects that are being shared, such as tables, and the addition of a table to a share is accomplished by granting the share object.
Select access to the table.
The shared object also contains information on the schema and database containing the shared item.
The shared object must be granted usage access to the schema in database.
Finally, the shared object contains information about one or more snowflake accounts with whom the data is shared.
These accounts are also referred to as consumers.
So after consumers account number is associated with a share, the share will begin to appear in the consumer's account.
The consumer can then create a read only database on the share object and is then able to view all of the shared object within that read only database.
Now it is important to note that the consumer account does not pay for the storage cost for the shared data.
However, any queries run by the consumer account on the shared data result in the usage of consumers compute.
Therefore, queries on shared data are charged to the consumer account.
Um, as I mentioned previously, reader accounts are an exception to this rule.
We will discuss them in coming lectures.
Finally, any snowflake account has the potential to become a data provider, allowing it to share a single item or several objects with other snowflake accounts.

Um, virtual private snowflake accounts or accounts are an exception to this, so they cannot share data.
So the process on the screen shows the steps required to share a table with another consumer.
The steps in blue are performed on the data provider.
So we start by creating a new share object.
Then we add the tables or any other supported objects such as views that we want to add to the share.
The add to the share is achieved by granting select access on the table to the share object.
We must also add the schema that contains the table and the corresponding databases to the share object.
This is achieved by granting usage access on the schema and the database to the share object.
Finally, on the data provider, we add one or more consumers to the share object, which basically results in the share being available to those consumers.
Now on the consumer account, we must create a database from the share.
There is a special syntax for that which we will see in the hands on lectures.
A database that is created from a share is automatically created as read only.
Then, optionally, you can grant access to other roles on the new database so that other users are also able to access shared data.
Now users in consumer accounts can query this shared data.
However, note that they'll be using the consumer accounts compute.

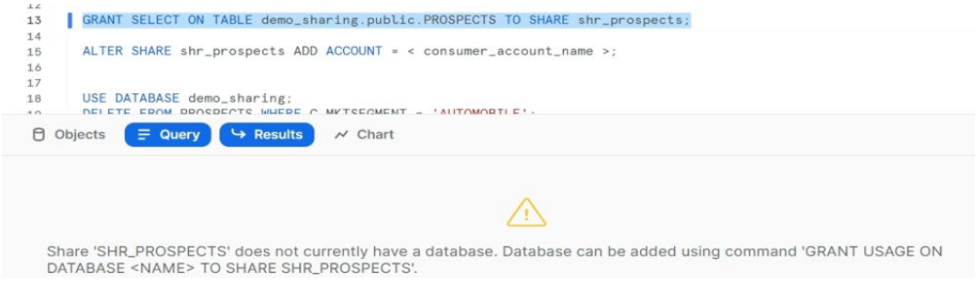
Share a table with another snowflake account:

2 snowflake accounts are required for this – one is data provider and another one for data consumer. Be on the same cloud provider and the same region.
It also covers across geographies.
Pls note that we use account admin role for sharing and managing the operation.

Please note that we will use the account admin role to create and manage the sharing operation.
It is possible to grant, create, share and import share privileges to another role which can then create these shares.
For setting up the sahre and sharing that with consumer account:

```
1   CREATE OR REPLACE DATABASE demo_sharing;
2   USE DATABASE demo_sharing;
3
4   CREATE TABLE PROSPECTS
5   AS SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER;
6
7   USE ROLE ACCOUNTADMIN;
8   CREATE SHARE shr_prospects;
9
10  GRANT USAGE ON DATABASE demo_sharing TO SHARE shr_prospects;
11  GRANT USAGE ON SCHEMA demo_sharing.public TO SHARE shr_prospects;
12
13  GRANT SELECT ON TABLE demo_sharing.public.PROSPECTS TO SHARE shr_prospects;
14
15  ALTER SHARE shr_prospects ADD ACCOUNT = < consumer_account_name >;
16
17
18  USE DATABASE demo_sharing;
19  DELETE FROM PROSPECTS WHERE C_MKTSEGMENT = 'AUTOMOBILE';
20
21  SELECT COUNT(*) FROM PROSPECTS;
```

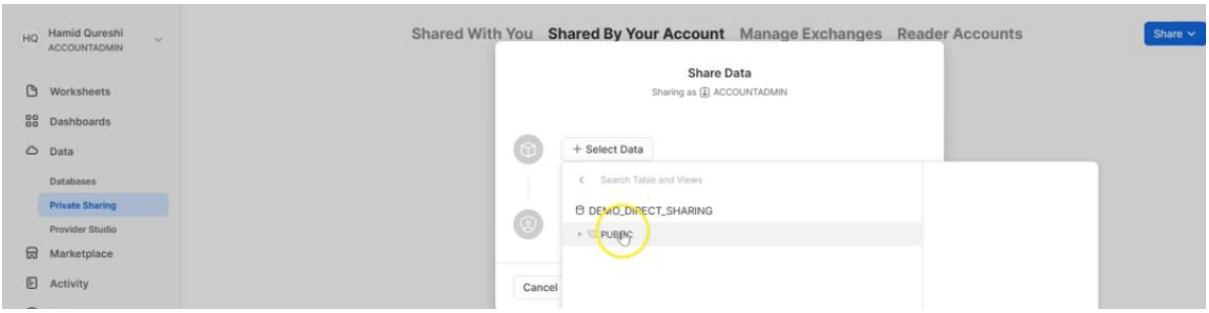
From the above, its clear that the share can be only created by a account admin. First we need to grant select on the database and schema and then the table else it will throw an error like below:



To find out the account name, below is the way we to find out the account name from the web url:

Share a table using snowflake WEB UI:

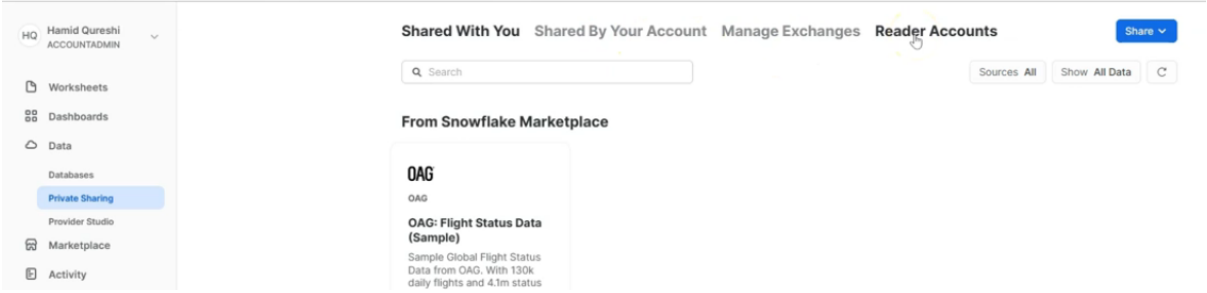
Provide acct and consumer accts are required for this as mentioned above already We need to have account admin privileges else we will not be able to share and the consumer acct also should be in the same role.



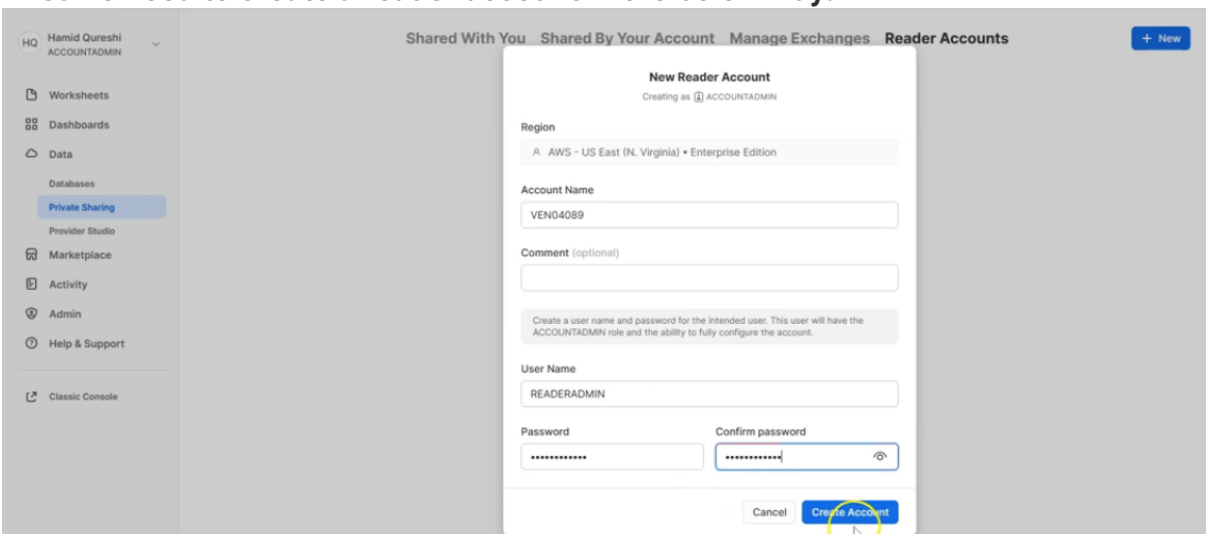
The consumer should be in the same region and the same cloud provider.

Share data with a non-snowflake customer with snowflake web UI:

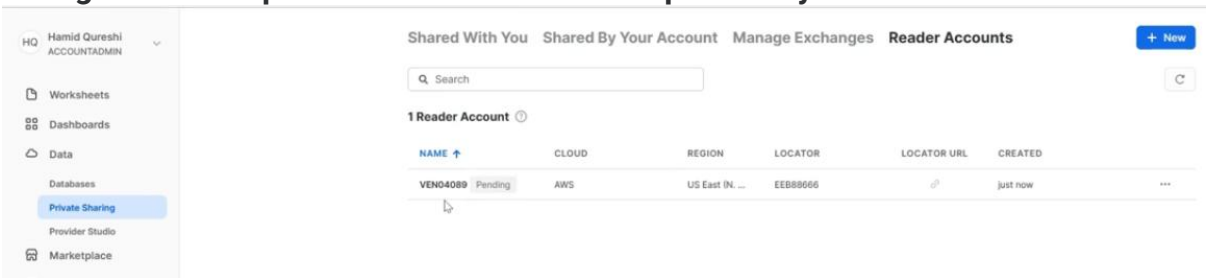
Here in this case, we are sharing the data here to a customer who doesn't have snowflake account. In this case we need to create a reader account for the user who to view the data on the other side.



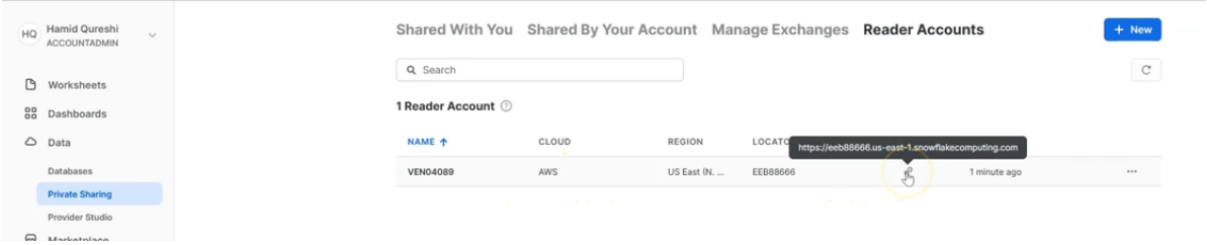
First we need to create a reader account in the below way:



Billing of this compute for the below is our responsibility:



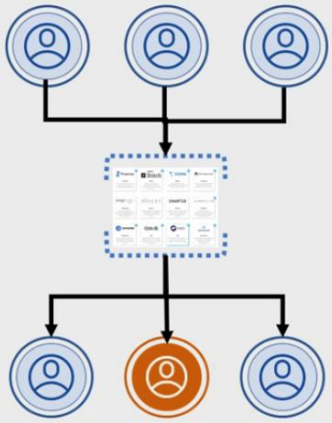
The account has been created:



Snowflake Market place:

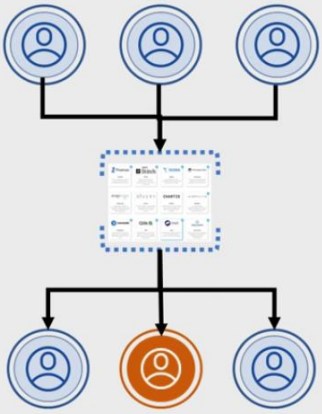
Snowflake Marketplace

- Discover and access (or purchase) 3rd party datasets
  - For a cost
  - For free
- Make 3rd party datasets available in your account
- Join the 3rd party data sets with your existing data to augment your data.
- Leverages the same Data Sharing technique as Direct Sharing
- Similar to Amazon or eBay in concept



Snowflake Marketplace

- Available only through the new Snowflake WebUI (Snowsight)
- Not available for Virtual Private Snowflake or VPS account
- Any Snowflake account can publish to Snowflake Marketplace
  - Must sign up as a partner and become an approved data provider
- To consume data from Marketplace the role used should be
  - Either ACCOUNTADMIN
  - Or have IMPORT SHARE privilege



Snowflake Marketplace

Snowflake marketplace listings can be

- Free
  - Free to access
  - Immediately available
  - May have trial or limited data
- Paid
  - Premium data
  - May allow personalisation

Snowflake Marketplace

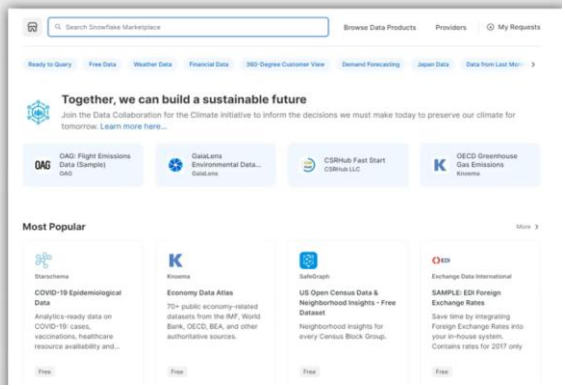
Snowflake marketplace listings can be

- Ready to Query
  - Immediately available
  - Generally free but can be for cost
- By Request & Personalized
  - Typically paid listings
  - Consumer Requests & producer evaluates request
  - Negotiate commercials
  - Personalize data if required

Exploring snowflake market place: **To explore this we need a user with account admin privilege or the import share privilege to consume the data.**

Snowflake Marketplace – Consume data

- Explore Snowflake Marketplace
- Discover various ways of searching for a data listings including categories, filters
- Import a dataset and query it



Snowsite web UI:

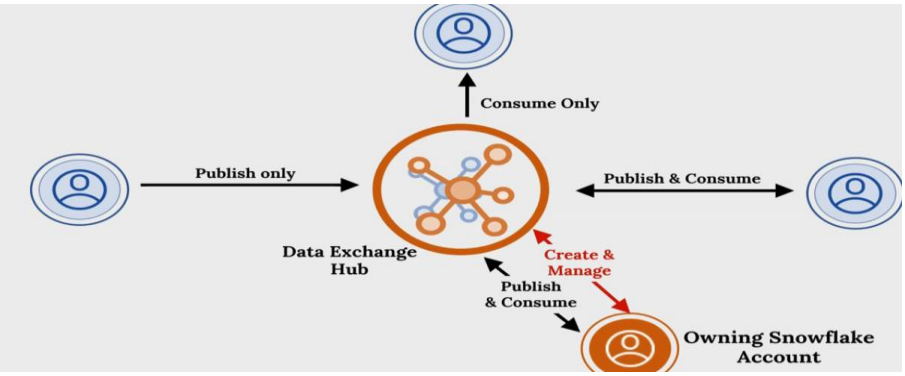
We usually will have different datasets un snowflake web UI in Market place. We need atleast admin account privileges or import and share privileges.

Data Exchange:

Data Exchange

- Private Data Sharing Hub
- Similar to Marketplace
- The Snowflake account owning the data exchange can enable sharing of data with
  - Partners
  - Suppliers
  - Vendors
  - Customers

Data exchange in snowflake is the own private data sharing hub. Its just similar to marketplace.



Quiz:

The cloud services layer facilitates data sharing through metadata operations.

When a Snowflake data provider shares data with another Snowflake account, the data consumer is charged for the compute charges for any queries they run.

Metadata operations in the cloud services layer allow data sharing without physically copying it. Since the provider account stores and pays for the data storage, the data consumer doesn't have to pay anything extra for storage. However, the data consumer pays for the compute used to run queries on shared data. When queries are run on shared data, the compute of the data consumer is used.

Sharing data with a non-Snowflake user or organization is possible by creating a reader account. This reader account is created by the data provider solely for sharing purposes.

Since the data provider creates and administers the reader account, all the reader account's compute expenses are invoiced to the provider account. Therefore, the reader account's use of the virtual warehouse compute is added to the provider account compute charges.

The Snowflake Marketplace is an online marketplace where you can purchase and sell datasets. You may import data from outside your company into your Snowflake instance and utilize it to enrich your data via the Snowflake Marketplace.

Data Exchange is your own private hub for sharing data with a small group of people or organizations who have been invited to join. The owner of the Data Exchange account is in charge of inviting members and specifying whether they can share, consume, or do both.

The consumer creates a database from the Share object as a read-only database.

Metadata operations in the cloud services layer allow data sharing without physically copying it. Since the provider account stores and pays for the data storage, the data consumer doesn't have to pay anything extra for storage. However, the data consumer pays for the compute used to run queries on shared data. When queries are run on shared data, the compute of the data consumer is used.

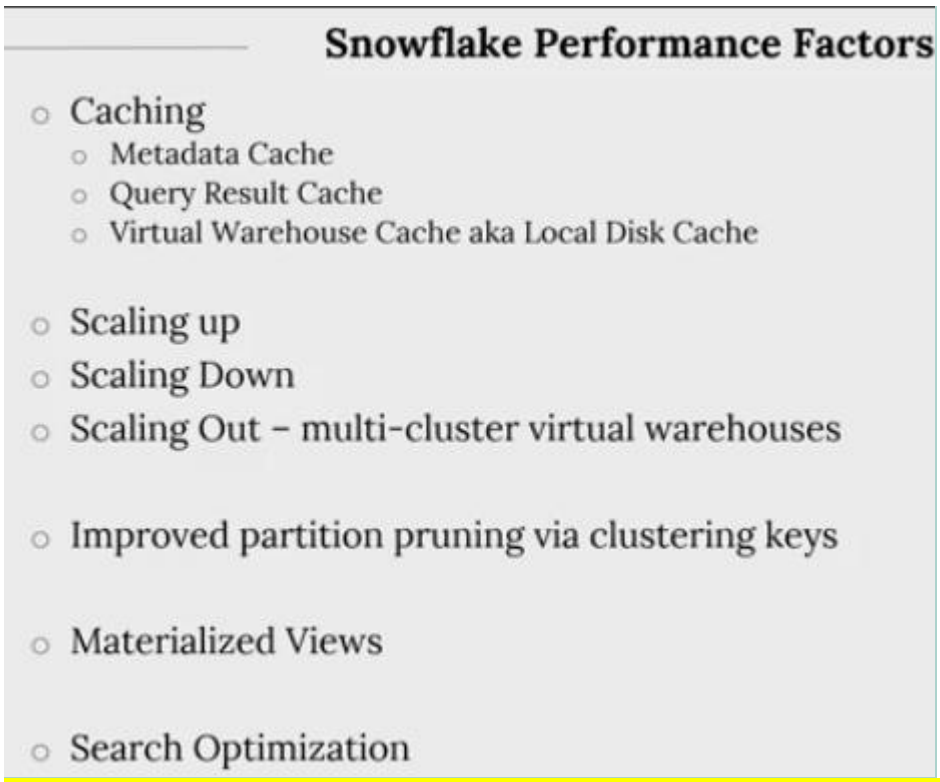
Except for Virtual private Snowflake accounts, the Snowflake Marketplace is available to all Snowflake accounts hosted on Amazon Web Services, Google Cloud Platform, and Microsoft Azure. Any Snowflake account (again, except for VPS accounts) can become a data provider and publish datasets to the Marketplace for a cost or for free. In addition, you are required to sign up as a partner first and become an approved data provider.

Virtual Private Snowflake (VPS) cannot use secure data sharing, Marketplace, etc., because VPS accounts have isolated metadata, compute, and storage and therefore don't have sharing capabilities.

Performance Optimization in Snowflake:

Some manual tuning may be required. scaling up and down of virtual warehouse is one of these features. There are some automatic features which work by default only thing is we need to take care that these features are working efficiently and as intended.

Partition cloning during query execution

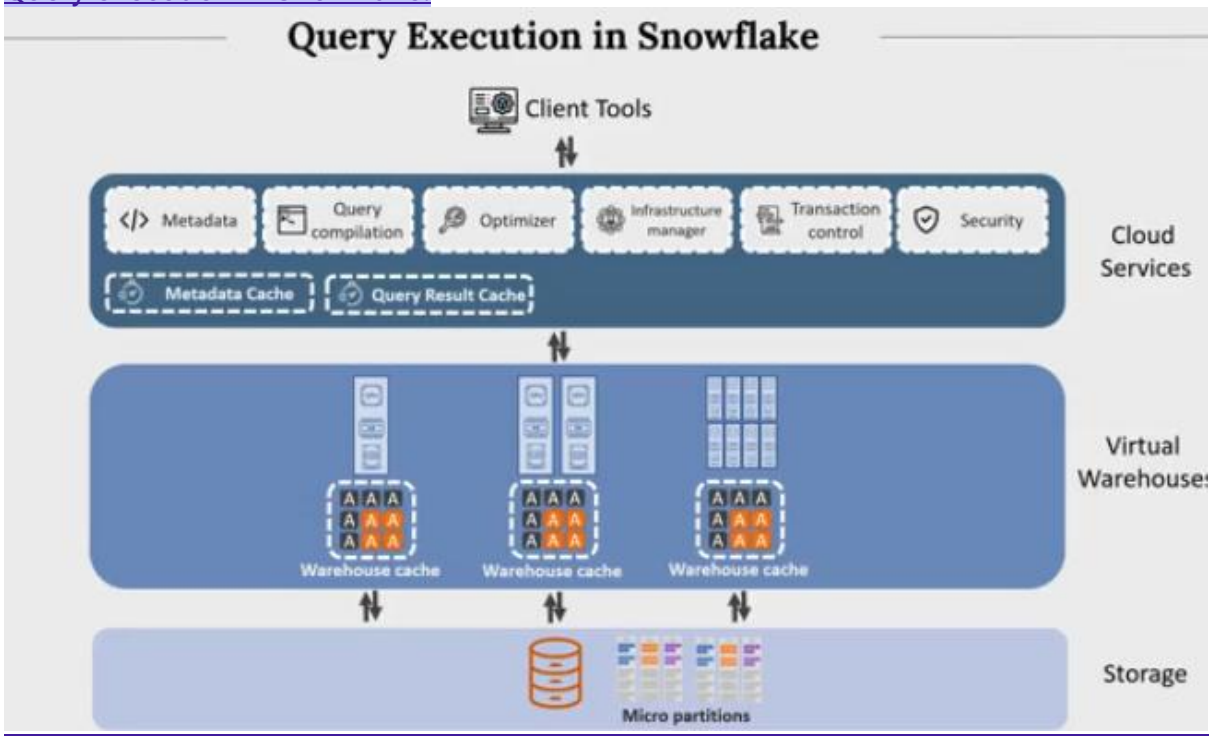


***In Snowflake, scaling up refers to increasing the size of a warehouse by adding more compute resources to it. This can be done by resizing the warehouse 1.
Scaling down refers to reducing the size of a warehouse by removing compute resources from it. This can be done by either suspending the warehouse or by manually downsizing it 1.
Scaling out refers to adding more clusters to a multi-cluster warehouse to increase the pool of compute resources available to it. This can be done either statically or dynamically, depending on the load on the warehouse ***

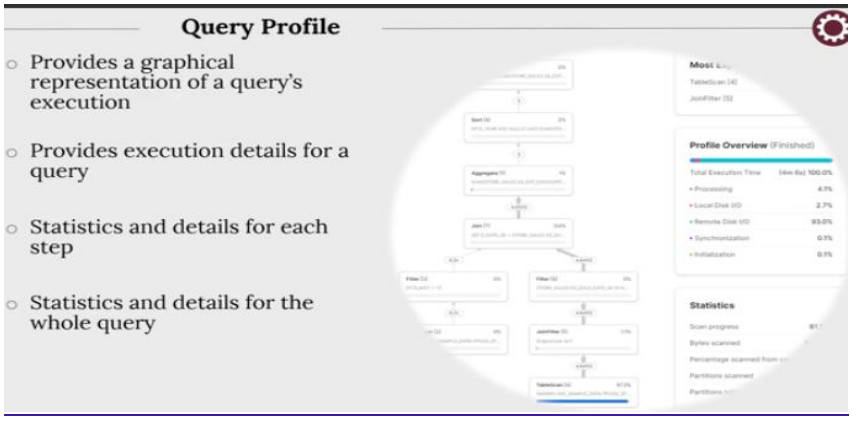
Some of the snowflake managed performance features include metadata, cache, query result, caching, caching at the virtual warehouse level and partition pruning during query execution.
While most of this work out of the box features such as partition pruning may work better if table partitions are aligned to the query patterns, so some manual tuning may be required.
In addition to these snowflake managed features, Snowflake also provides optional and configurable performance options that can be set manually to optimize performance.
Scaling up and down a virtual warehouse is one of these features, as is auto scaling a virtual warehouse to handle rising concurrency.

This section discusses the performance optimization features and approaches available for enhancing query performance and in some instances, even reducing costs.
For snowflakes, unique architecture and the underlying micro partitions.
Storage technology means that it is not required to perform much query tuning in most situations.
There are, however, several performance improvement approaches that are available and can be used to increase Snowflake's overall performance. So these options include internal caching mechanisms that operate transparently in the background to increase performance.
And scaling up or increasing the capacity of a virtual warehouse to allow for more processing power to be available for complex queries.
Then horizontal scaling, which is done by increasing the capacity, by using a multi-cluster virtual warehouse to handle a larger number of concurrent users and concurrent queries.
Automatic static and dynamic partition pruning can be used to reduce unneeded partitions while a query is being processed.
It is possible to accomplish better partition pruning by redistributing data in micro partitions using clustering keys.
Then pre-computing results of complex regularly executed queries by using materialized views.
And finally, search optimization, which can be used to improve the performance of specific types of lookup queries.

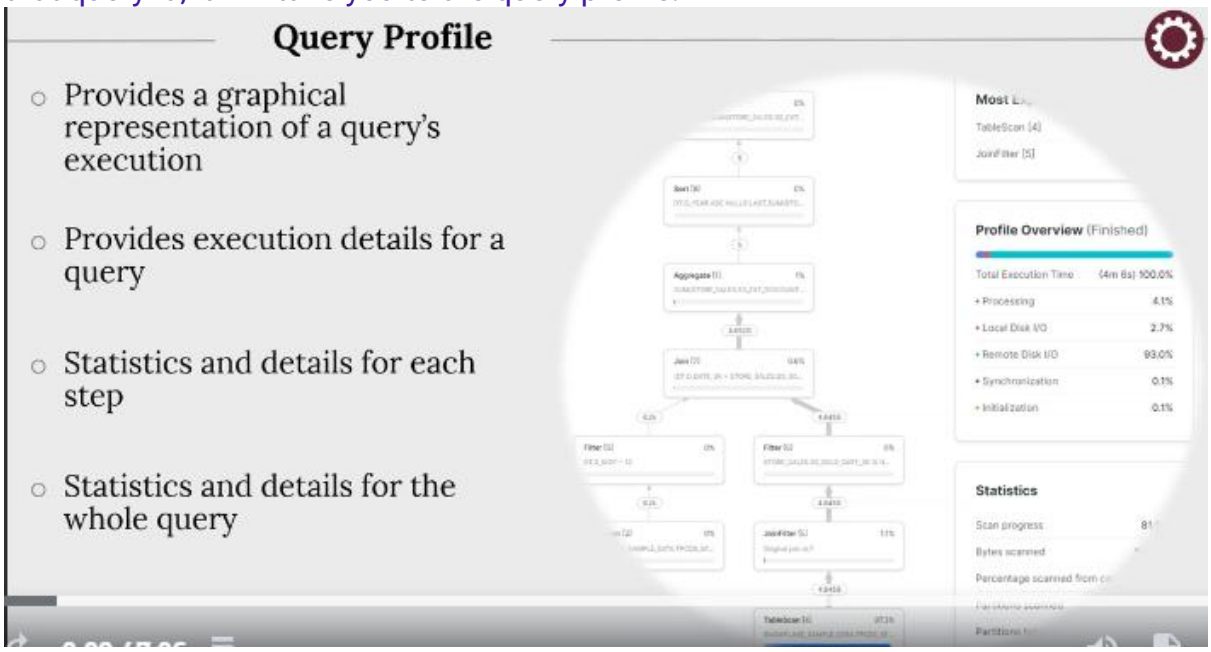
Query execution in Snowflake:



Query Profile:



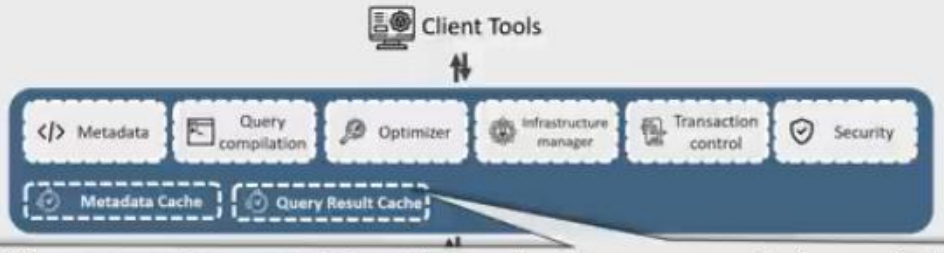
Once we submit the query , a query id appears in the out put screen. If we click on that query id, it will take you to the query profile.



Caching in Snowflake:
Metadata cache & Result cache – these are part of cloud services layer. These are available to all virtual warehouses.

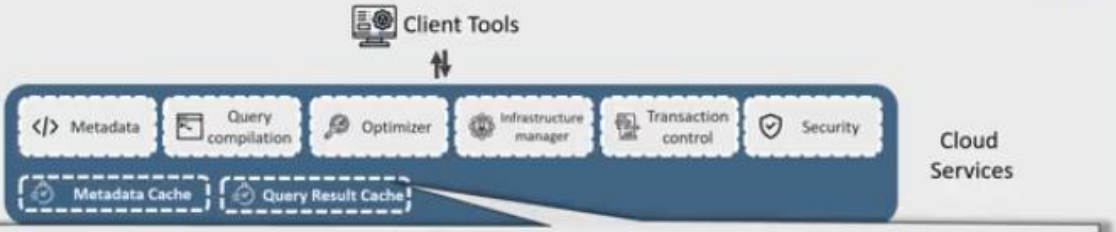
Query result cache:

Query Result Cache

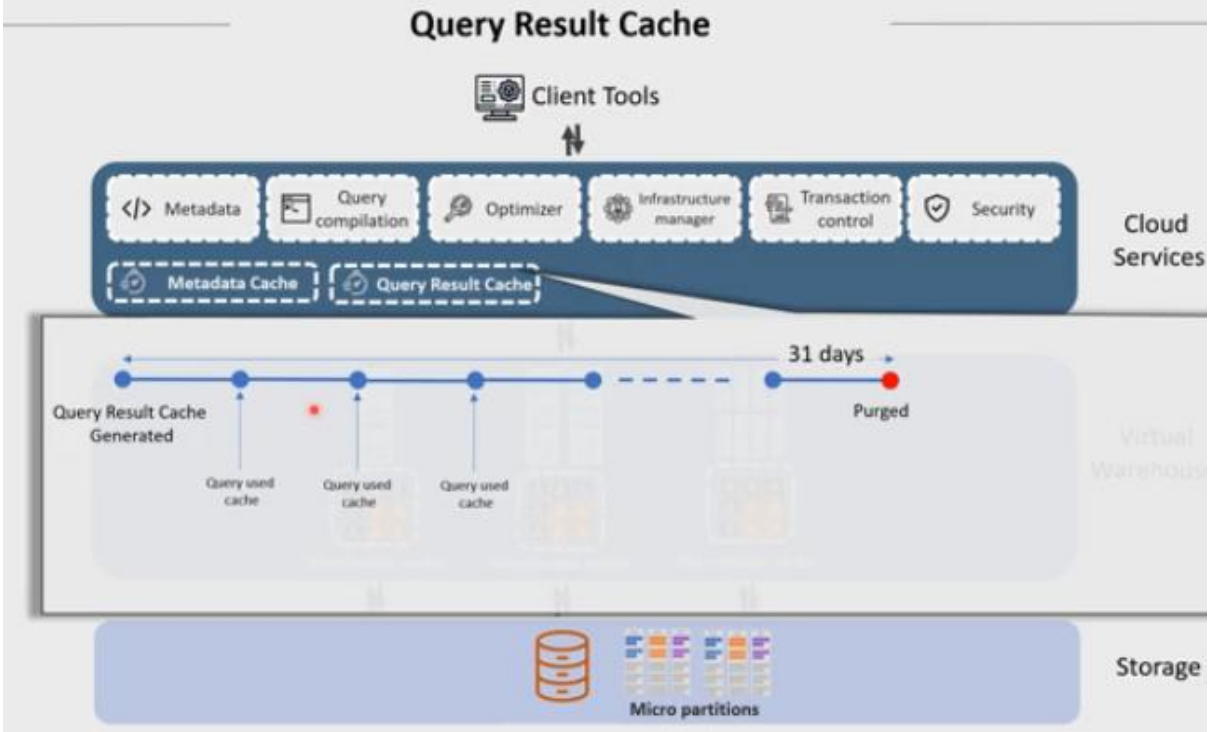


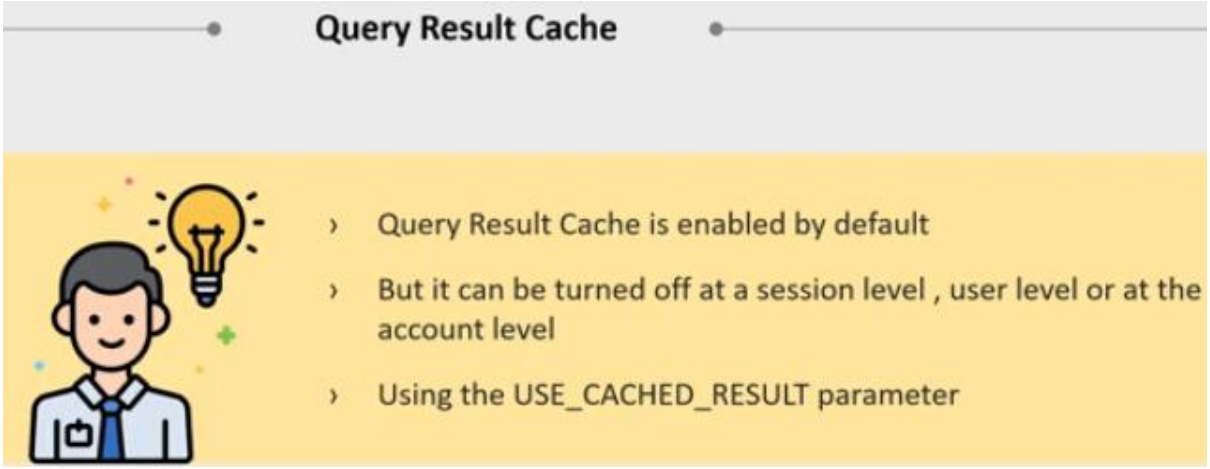
- When a query is executed, Snowflake caches the query results for a period.
- Snowflake uses the Query Result Cache to avoid re-executing queries when there are no changes to data
- Query Result Cache is used by a new query if
  - The query syntactically matches the previously executed query
  - The data in underlying table hasn't changed, and the micro-partitions haven't changed
  - The query doesn't use functions that are evaluated at runtime, or UDF or external functions
  - The security role running the query has the required privileges
  - (few other aspects also influence if Query Result Cache will be used)

Query Result Cache

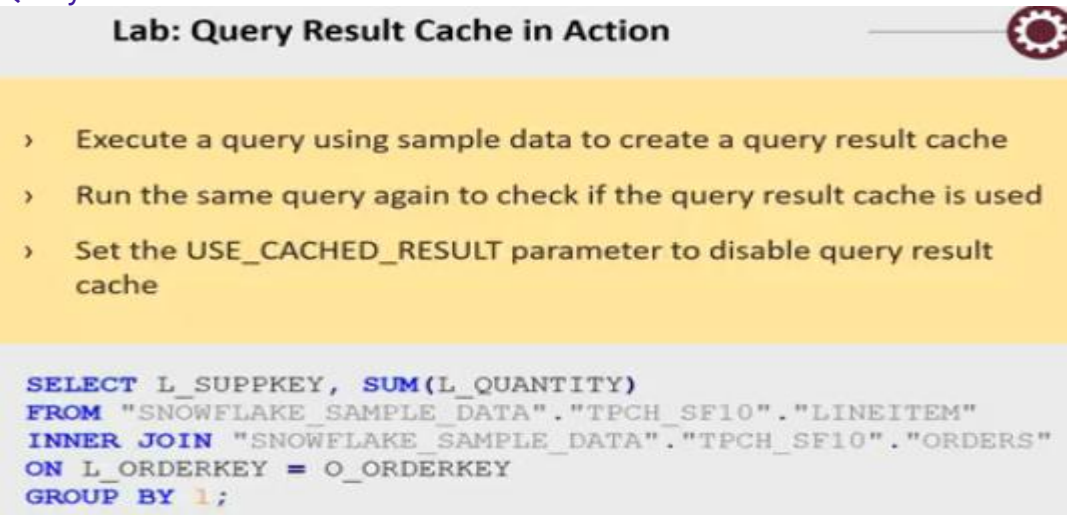


- Query result cache for a given query stays valid for 24 hours
- However, if a new query is run which uses the query result cache, the validity is reset for another 24 hours.
- Query Result Cache for a given query may stay valid up to a maximum of 31 days after which is purged





Query result cache in action:



Before returning the results pls check the below query plan when we checked using the query id: when we re-run the same query again the query plan looks different compared to initial execution.

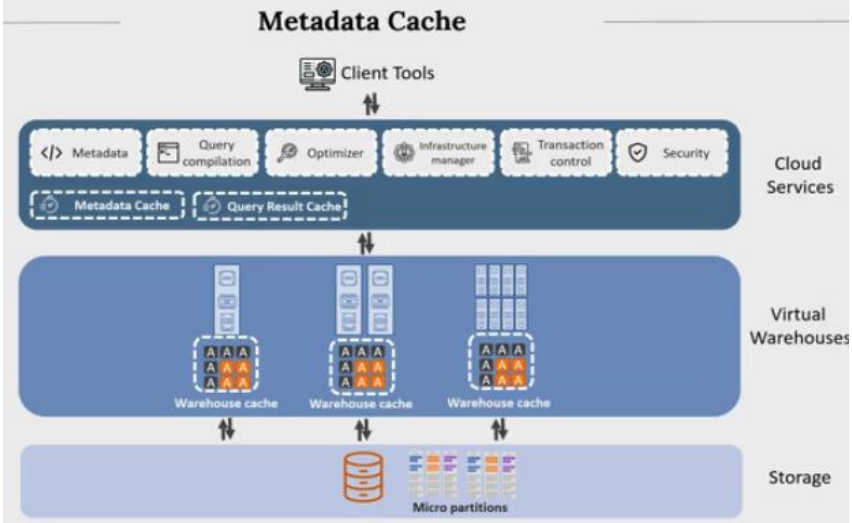
Below is the query plan returned when the same query is executed again:

We can also disable the query result cache. Below is the way to disable the query result cache for that particular session:

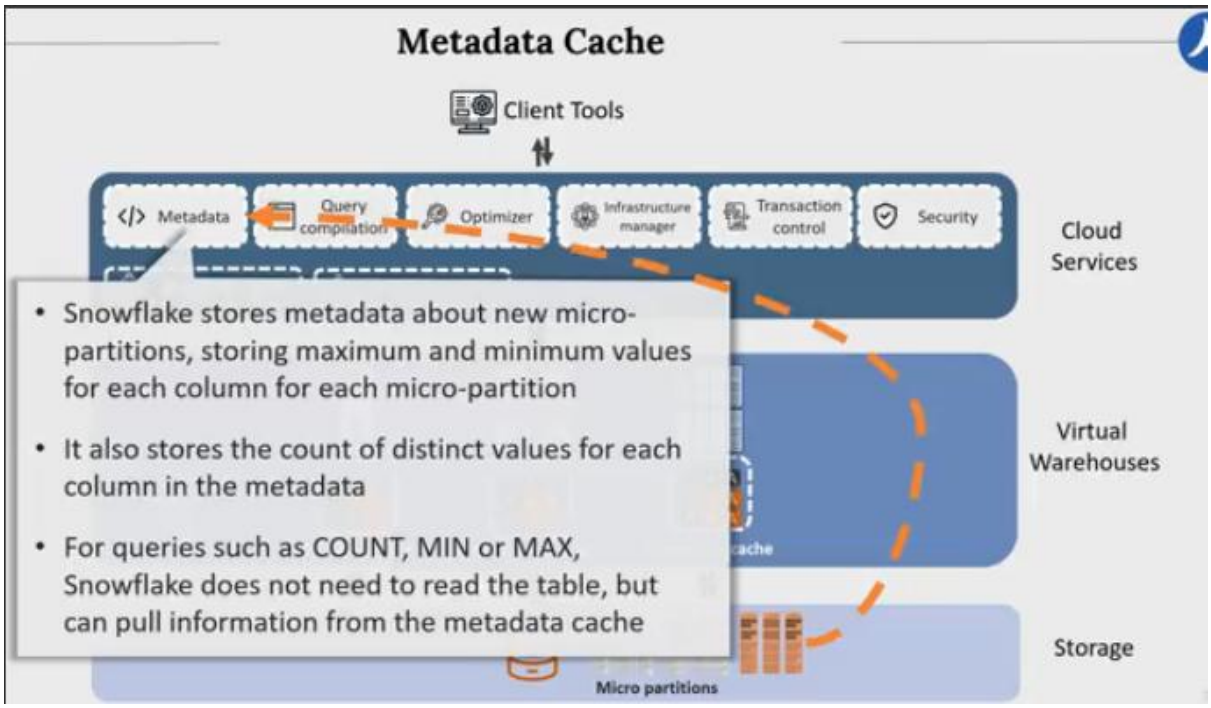
```
SELECT L_SUPPKEY, SUM(L_QUANTITY)
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF100"."LINEITEM"
INNER JOIN "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF100"."ORDERS"
ON L_ORDERKEY = O_ORDERKEY
GROUP BY 1;

ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

In this case there wont be any difference in the query plan as we disabled the query result cache. From this point it will not use the query result cache unless a new session is started or if the above is set to ‘TRUE’

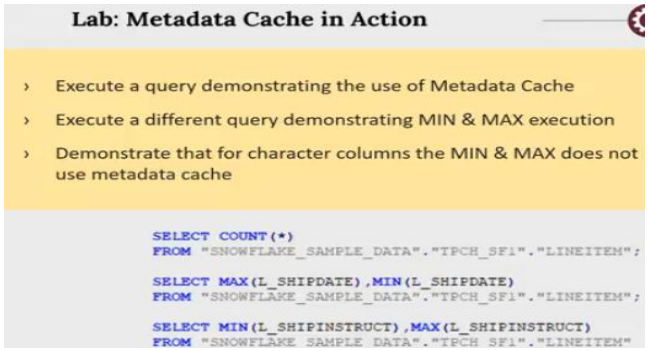
Metadata Cache:everytime when data is inserted, updated or deleted in snowflake table, new micro partitions are written. When new micro partitions are written snowflake stores that info in the metadata.



Continuation of above:

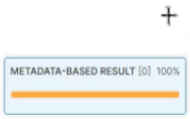


Metadata cache in Action:

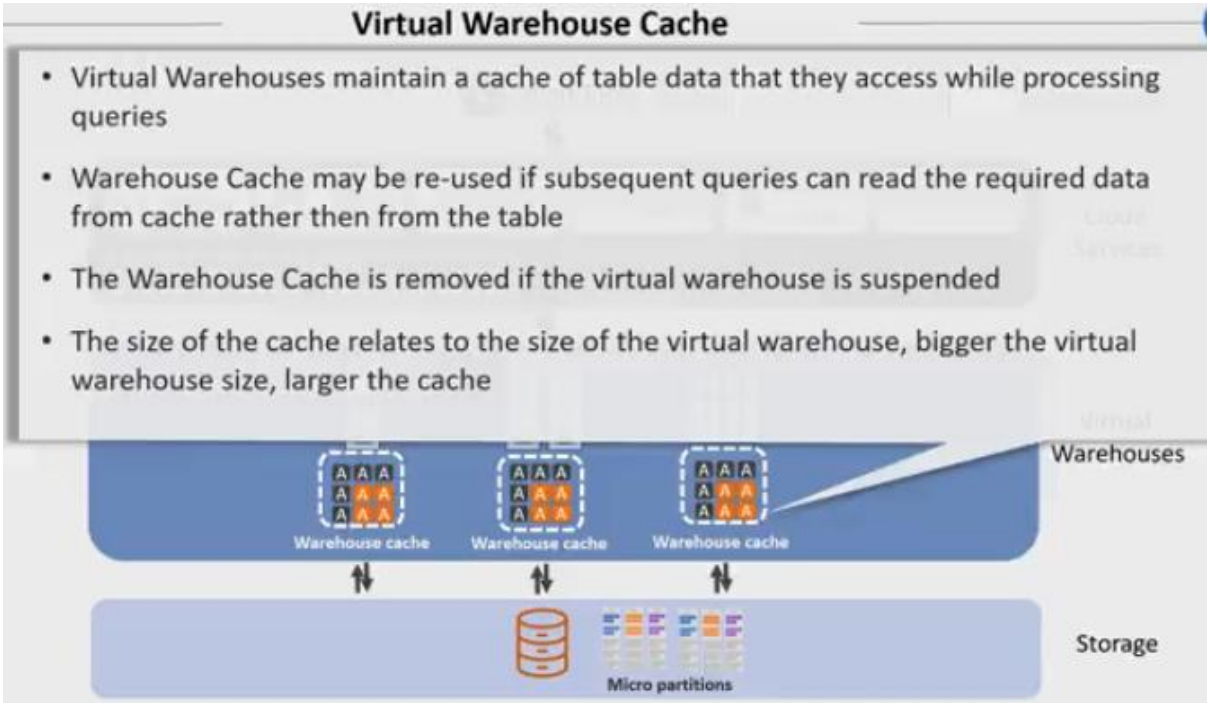


All the types of min, max and count related queries will go to metadata cache and fetches the result quickly. For character columns (for last query from the above 3 queries) the metadata doesn't store the min and max of those columns

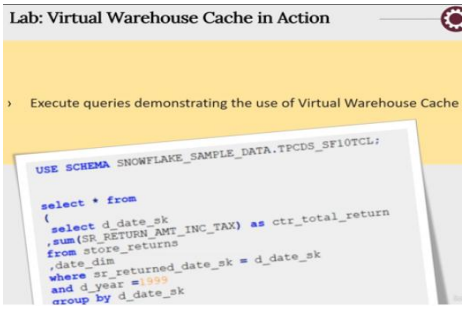
For any of the below queries executed the snowflake doesn't make use of the micro partitions but it would make use of only the cache as shown below:



Virtual warehouse cache or local disk cache:

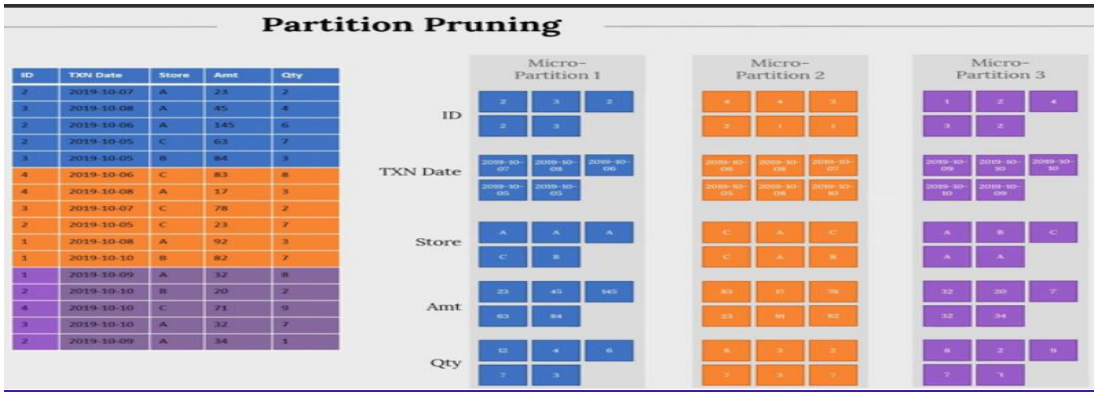


Virtual warehouse cache in action:

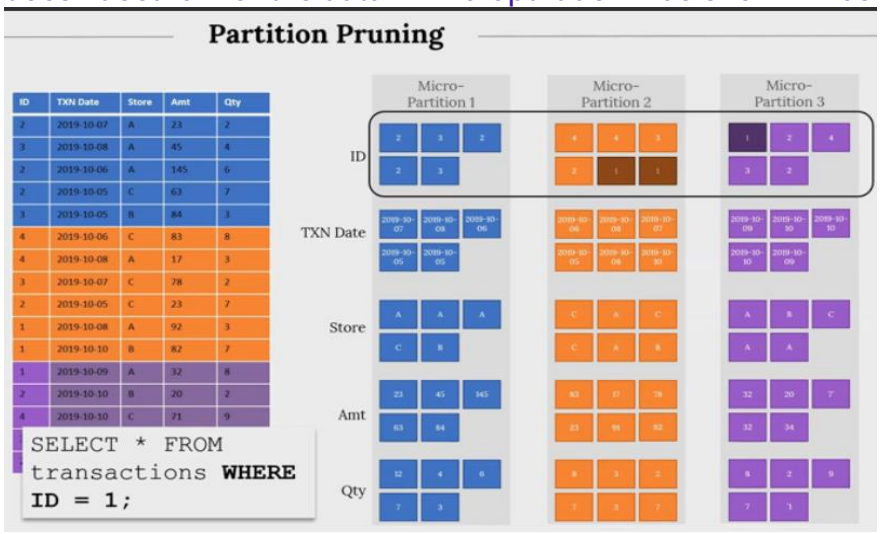


Once the virtual warehouse is suspended, all the data stored in cache will be removed and the query executed immediately after the warehouse suspended will not be using the data stored in cache as it doesn't exists any more.

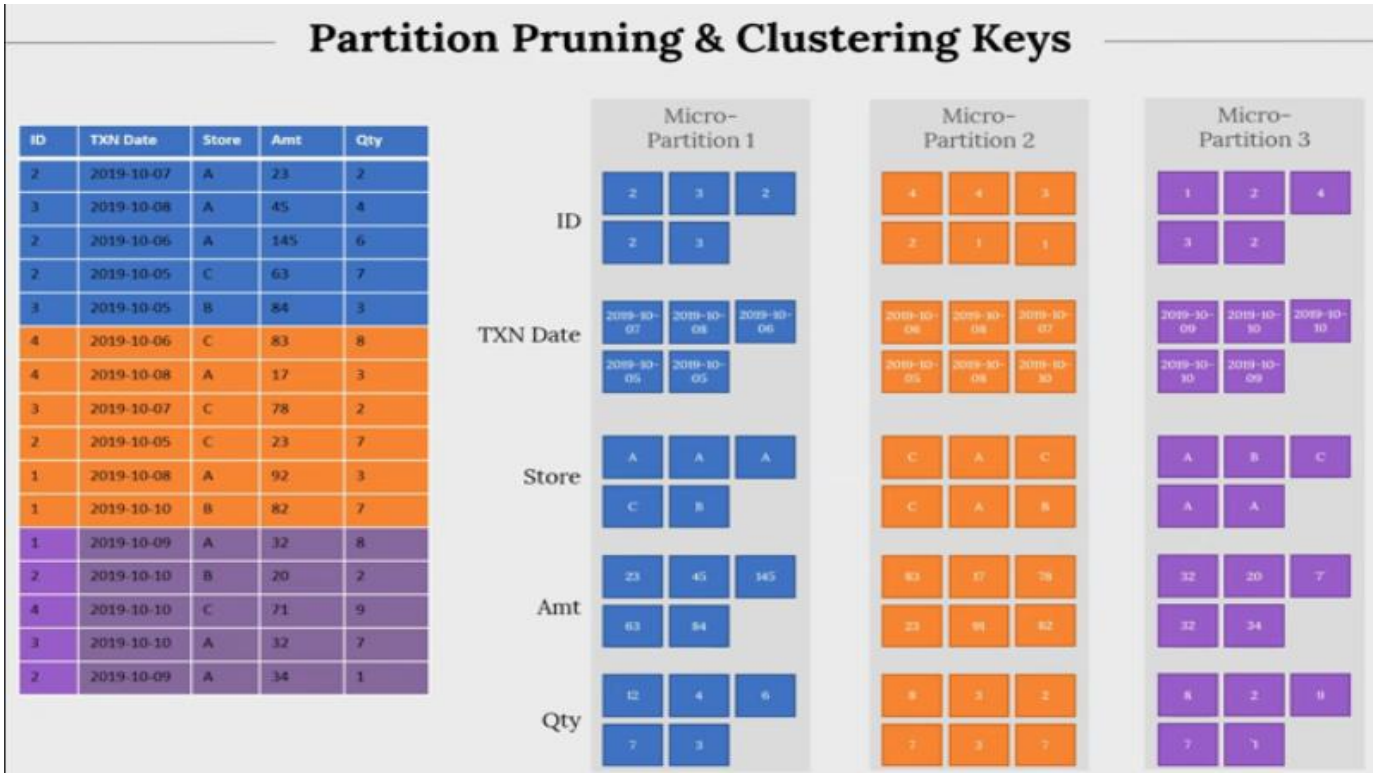
Partition pruning and clustering Keys:



Snowflake query optimizer knows the id value 1 is contained in 2 and 3 partitions, it doesn't search for the data in micropartition 1 as shown in below screen shot:

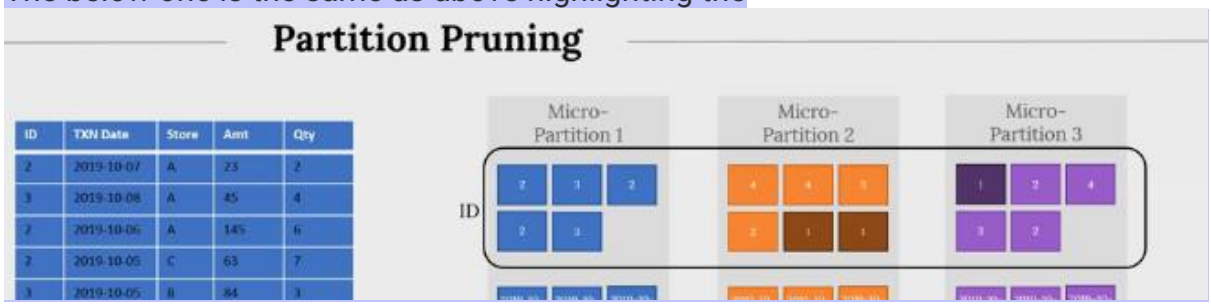


Additional partitions are added to the micro partitions are produced when data are added to a table. Because the column values are scattered across numerous micro partitions. Snowflake must keep track of what range of data is kept in which micro partition for each column. This metadata enables Snowflake to eliminate unnecessary micro partitions when running queries.
Therefore, boosting the overall query performance. This process of eliminating micro partitions is also known as partition pruning.



So consider the query shown on the screen. Because the Snowflake Query Optimizer knows that the data for ID equal to one is contained in micro partition two and three. It does not search micro partition one at all.
Therefore, reducing the amount of reads it needs to perform for processing this query. So for this simple example, it may not seem that big a deal, but for a really large table partition, pruning can eliminate a large number of partitions, therefore improving query performance significantly.

The below one is the same as above highlighting the



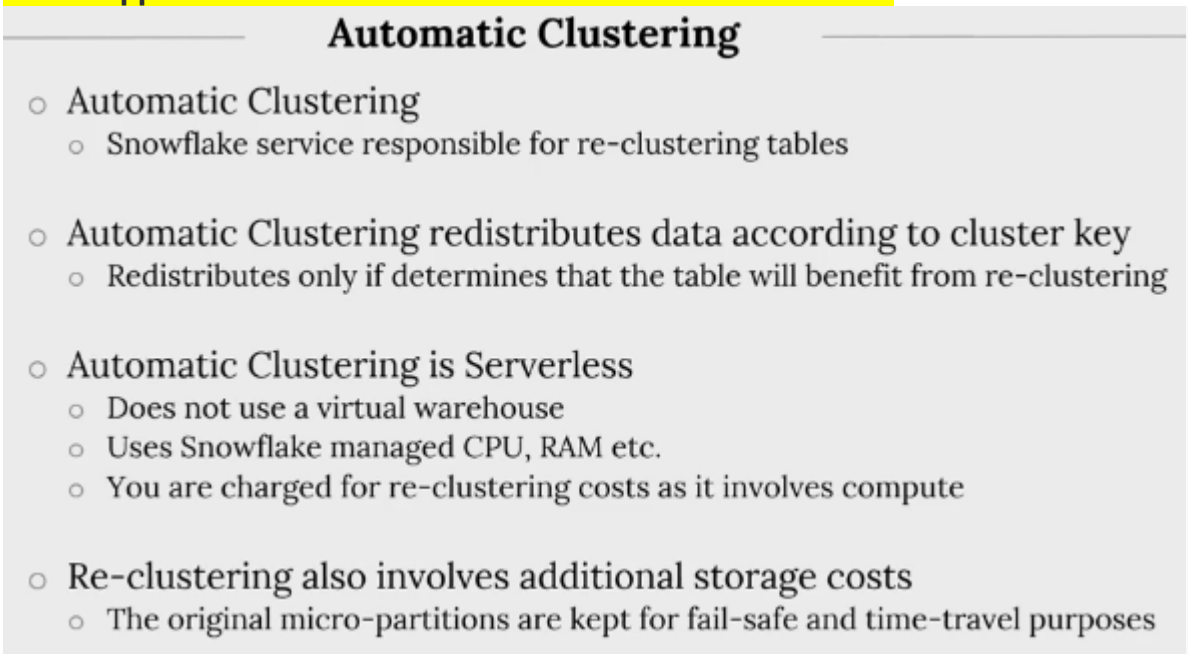
Since micro partitions are produced in the order of the arrival of the data over time, the data in micro partitions may not be optimally stored and may not support optimal partition pruning.
For example, the micro partitions shown on the below screen do not enable effective partition pruning if most queries are based on the store column.

If queries are mostly predicated on the store column as shown on the screen, queries may perform better if the table is clustered on the store column with no clustering in place and for the data shown on the screen.
Snowflake will need to scan all micro partitions to find all records for store A.
Similarly, if a query was trying to retrieve data for store B only, it also needs to scan all the micro partitions.

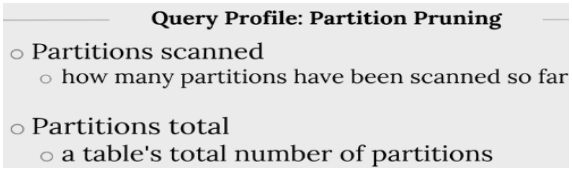
Similarly, if a query was trying to retrieve data for store B only, it also needs to scan all the micro partitions.
By clustering a table on a specific column, queries can be optimized by eliminating unneeded partitions from the query processing.

The advantages of clustering a table on a separate column may not be visible for tables with a small quantity of data, but as the table data and its micro partitions grow.
Clustering a table on the proper set of columns can bring significant gains in speed through partition pruning (This process of eliminating micro partitions is also known as partition pruning.)

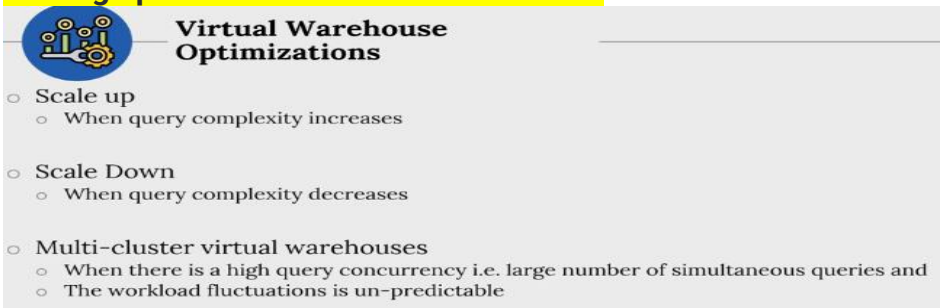
What happens behind the scenes when a table is re-clustered:

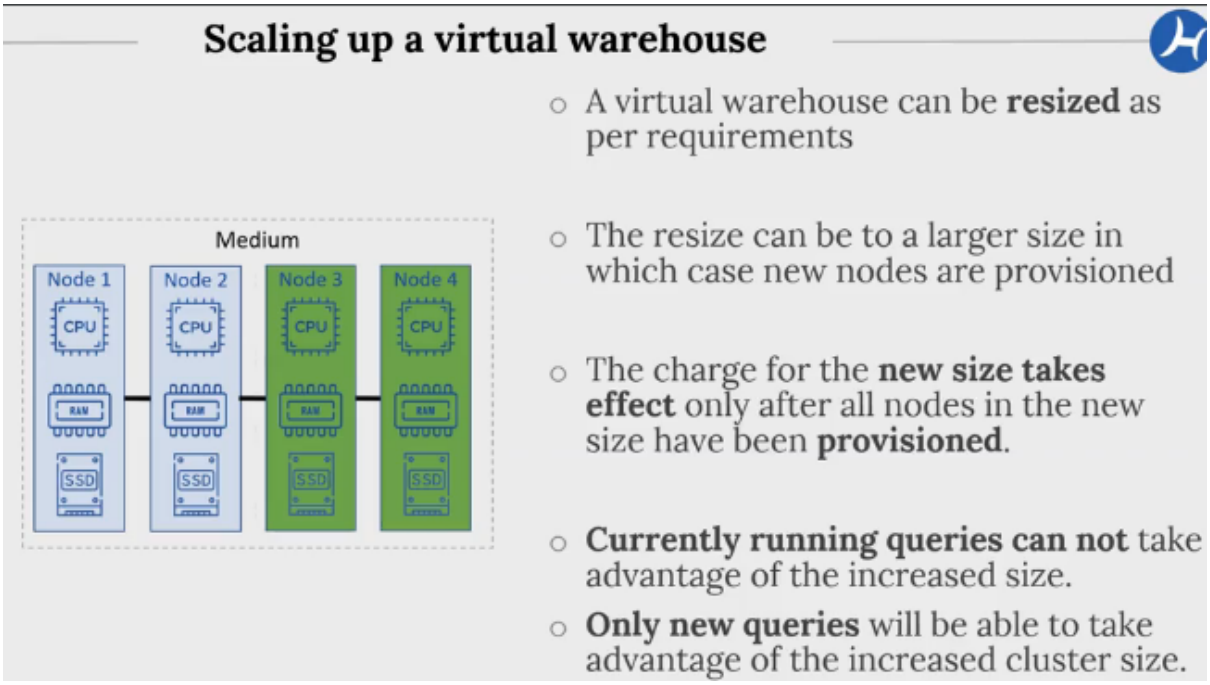


For tables with a clustering key defined automatic clustering, A snowflake service manages the clustering as needed distributing data according to the clustering key to achieve appropriate partition pruning.
So Snowflake internally maintains the cluster tables and any resource requirements that are associated with automatic clustering.
Automatic clustering only adjusts those micro partitions which benefit from the clustering process. Automatic clustering does not need a virtual warehouse but uses snowflake managed CPU and RAM.
Therefore it has a cost attached which would appear under serverless costs.
Clustering a table uses credits like any other data modification action in Snowflake, so clustering also adds extra storage when data is physically redistributed, and new micro partitions are created.
The original micro partitions in this case are kept for time travel and failsafe purposes, resulting in increased storage.
So Snowflake does not immediately update the table Micro partitions when we define a table clustering key.
Instead, Snowflake redistributes data according to the new clustering key only if it determines that the table will benefit from re clustering.

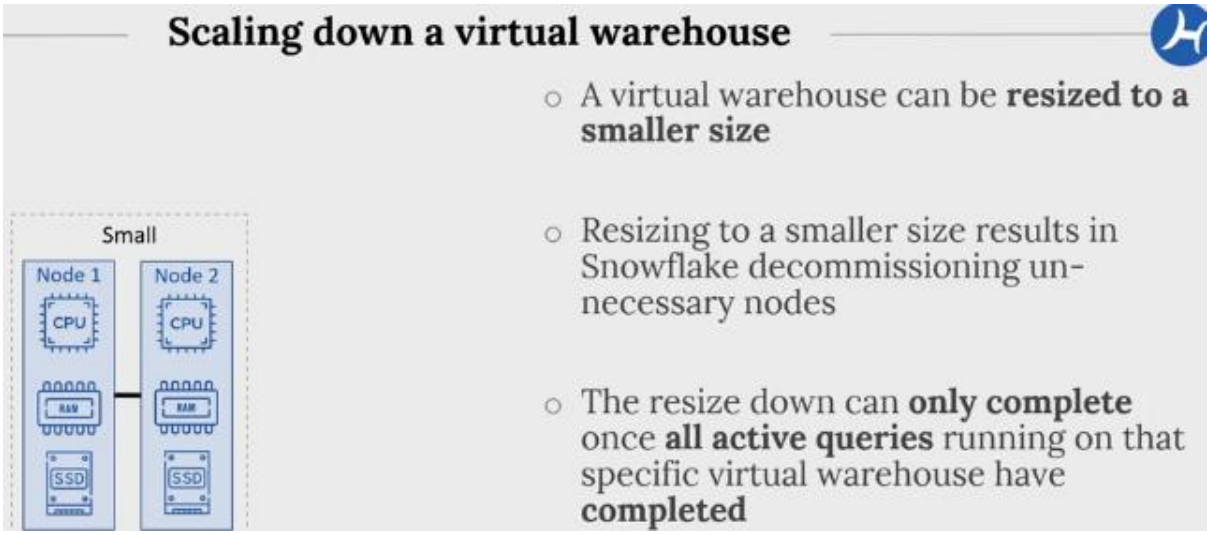


Scaling up and down a Virtual warehouse:





The virtual warehouse being scaled up or down is being done by system admins.



The syntax to scale up or down a Virtual warehouse:

```
ALTER WAREHOUSE <warehouse_name>
SET WAREHOUSE_SIZE = XSMALL | SMALL | MEDIUM | LARGE |
XLARGE | XXLARGE | XXXLARGE | X4LARGE | X5LARGE |
X6LARGE;
```

The larger size is used for scaling up and the smaller size is used for scaling down.

Multi-cluster virtual warehouse - Scaling out

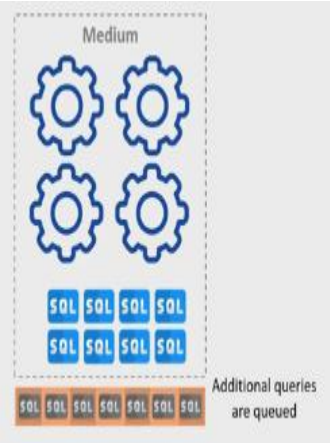
Scaling out / Multi-cluster virtual warehouse

- Typical virtual warehouse consists of a single cluster of compute resources
- Queuing occurs if there aren't enough compute resources

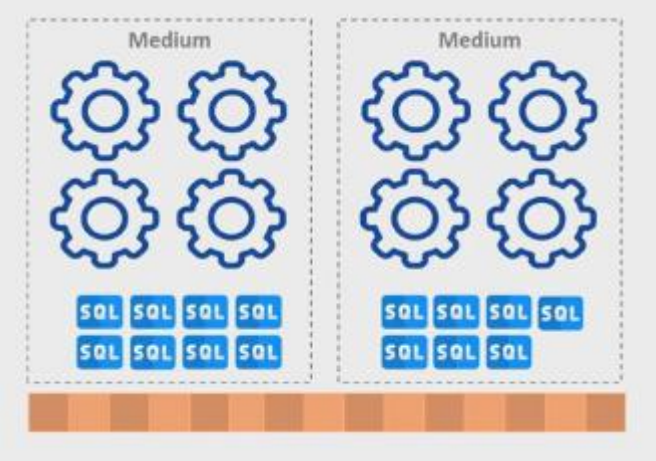
- Multi-cluster virtual warehouses provide a solution
  - When concurrent queries exceed the capacity of a single virtual warehouse
  - Spin up new virtual warehouses automatically
  - Decommission when demand decreases

- Scaling out is done in response to concurrency demands
  - On the other hand, scaling up/down is used to improve the performance of complex queries

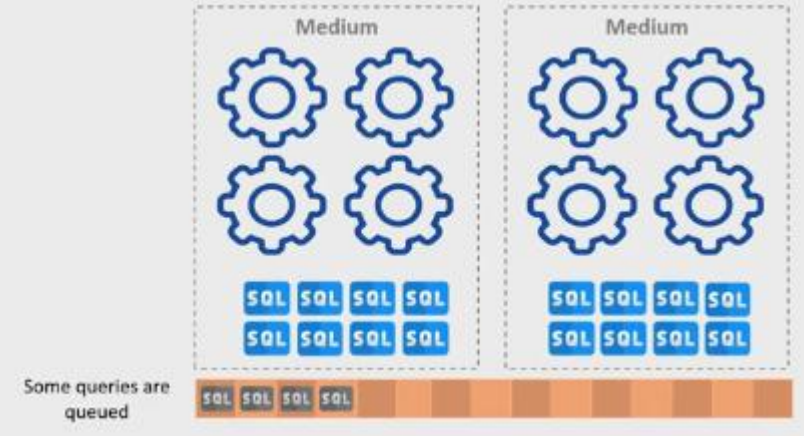
In the case of standard virtual warehouse:



However, in the case of a multi-cluster warehouse, Snowflake will spin up an additional instance of a medium sized virtual warehouse and execute the queued queries on the newly spawned virtual warehouse. Below is the way:



Incase if both the above are full and again the additional queries are queued up again as shown below:

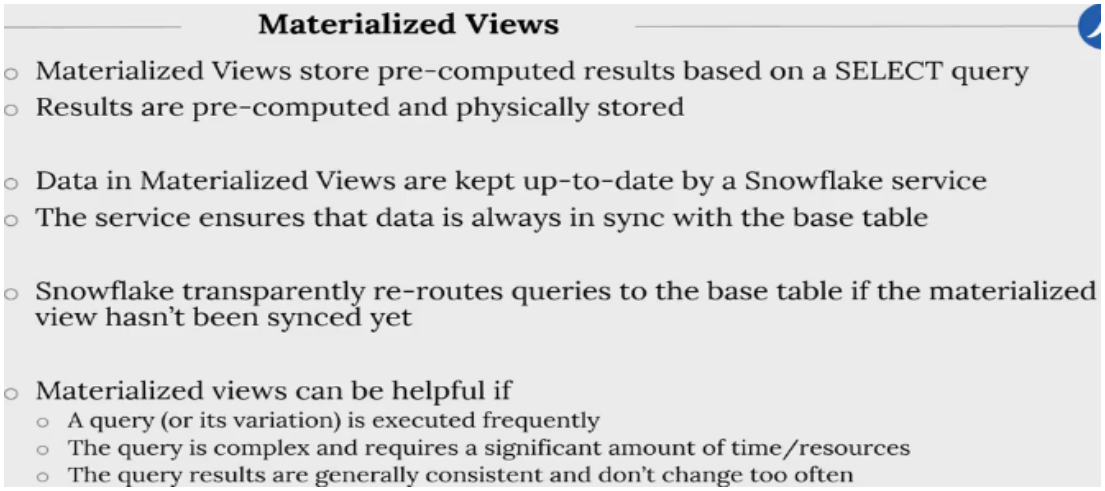


However, the process of scaling out automatically can continue up to the maximum configured value for the Multi-cluster virtual warehouse.
So if the Multi-cluster virtual warehouses were configured to allow for up to three or higher number of virtual warehouses, it would spawn additional virtual warehouses as more queries come in.
And if there are not enough compute resources to handle those queries.
Once the query workload decreases and the additional spawned virtual warehouses don't have any workload, Snowflake shuts them down until it gets to the configured minimum value.
In our case, the minimum value is one, so the multi-cluster virtual warehouses will scale back until there is only one medium sized virtual warehouse.
The syntax for creating a Multi-cluster virtual warehouse is similar to creating a standard virtual warehouse with a few additional options.
The syntax shown on the screen focuses on relevant parameters which are related to Multi-cluster virtual warehouses.
For the complete syntax, please refer to snowflake's documentation.

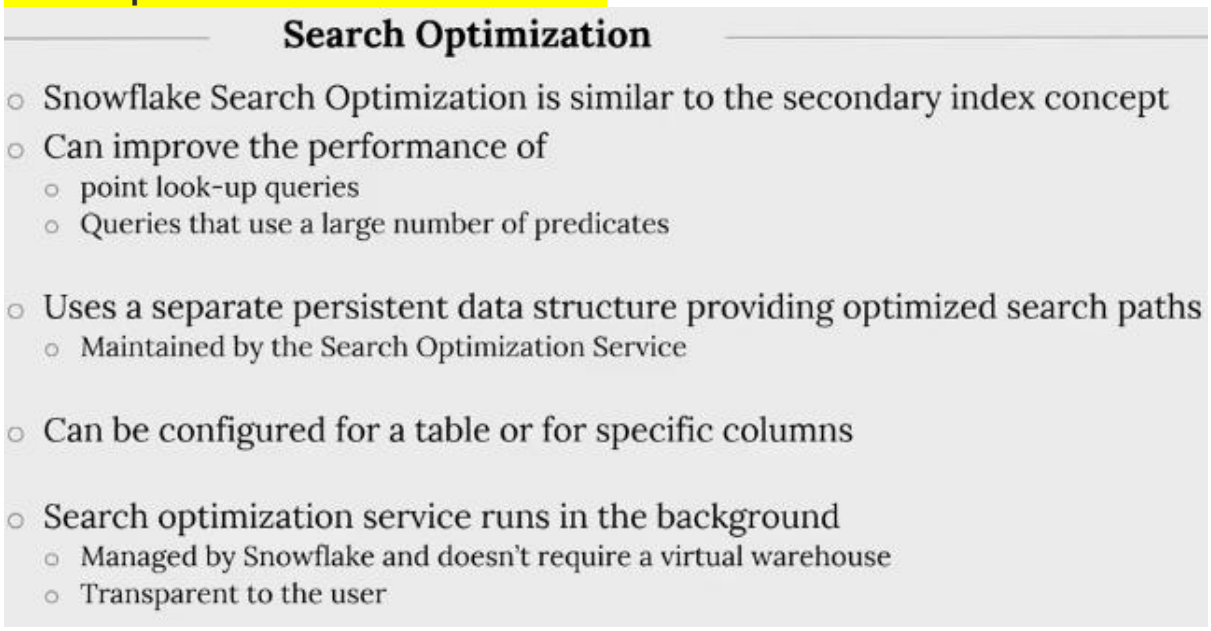
```
CREATE  WAREHOUSE <name>
WITH WAREHOUSE_SIZE = <size>
MAX_CLUSTER_COUNT = <num>
MIN_CLUSTER_COUNT = <num>
SCALING_POLICY = STANDARD | ECONOMY
AUTO_SUSPEND = <num> | NULL
AUTO_RESUME = TRUE | FALSE
INITIALLY_SUSPENDED = TRUE | FALSE;

MIN_CLUSTER_COUNT = starting number of virtual warehouses
MAX_CLUSTER_COUNT = maximum clusters that can be spun up
```

So when creating a multi-cluster virtual warehouse, the primary differentiation from a typical virtual warehouse is that you set a minimum and a maximum cluster count.
The maximum cluster count can be any value from 2 to 10, indicating the maximum number of virtual warehouses that this multi cluster virtual warehouse can spin up.
The minimum cluster count can be any value from one to the maximum, indicating the starting number of virtual warehouses.

Materialized Views:



Search optimization service in Snowflake:



Quiz Performance optimization in snowflake:

- Once a result cache is generated for a query stays valid for 24 hours. If another query that reuses the query result cache is executed within that 24-hour window, the result cache expiry is extended for another 24 hours from that point onwards. If the result cache for a query keeps getting used, it will stay valid for up to 31 days. After 31 days, the result cache for a query will be purged regardless of any other condition.

- Every time a virtual warehouse accesses data from a table, it caches that data locally. This data cache can improve the performance of subsequent queries if those queries can reuse the data in the cache instead of reading from the table in the cloud storage. The warehouse cache is local to a virtual warehouse and can not be shared with other virtual warehouses.

- Clustering a table on a specific column can optimize queries by eliminating unnecessary partitions from the query processing. A table can be re-clustered by defining a clustering key, which effectively redistributes the data into micro-partitions, ensuring optimal access to the clustered column.

- For tables with a clustering key defined, Automatic Clustering, a Snowflake service, manages the re-clustering as needed, distributing data according to the clustering key. Snowflake internally maintains the clustered tables and any resource requirements with Automatic Clustering. Automatic Clustering only adjusts those micro-partitions which benefit from the re-clustering process.

- For a populated table, the clustering depth is the average depth of overlapping micro-partitions for specific columns. The clustering depth starts at 1 (for a well-clustered table) and can be a larger number. For an unpopulated table, the clustering depth is zero.

- When defining clustering keys, the initial candidate clustering columns are those columns that are frequently used in the WHERE clause or other selective filters. Additionally, columns that are used for joining can also be considered.

- A materialized view is a view that pre-computes data based on a SELECT query. The query's results are pre-computed and physically stored to enhance performance for similar queries that are executed in the future. When the underlying table is updated, the materialized view refreshes automatically, requiring no additional maintenance. Snowflake-managed services perform the update in the background transparent to the user without interfering with the user's experience.

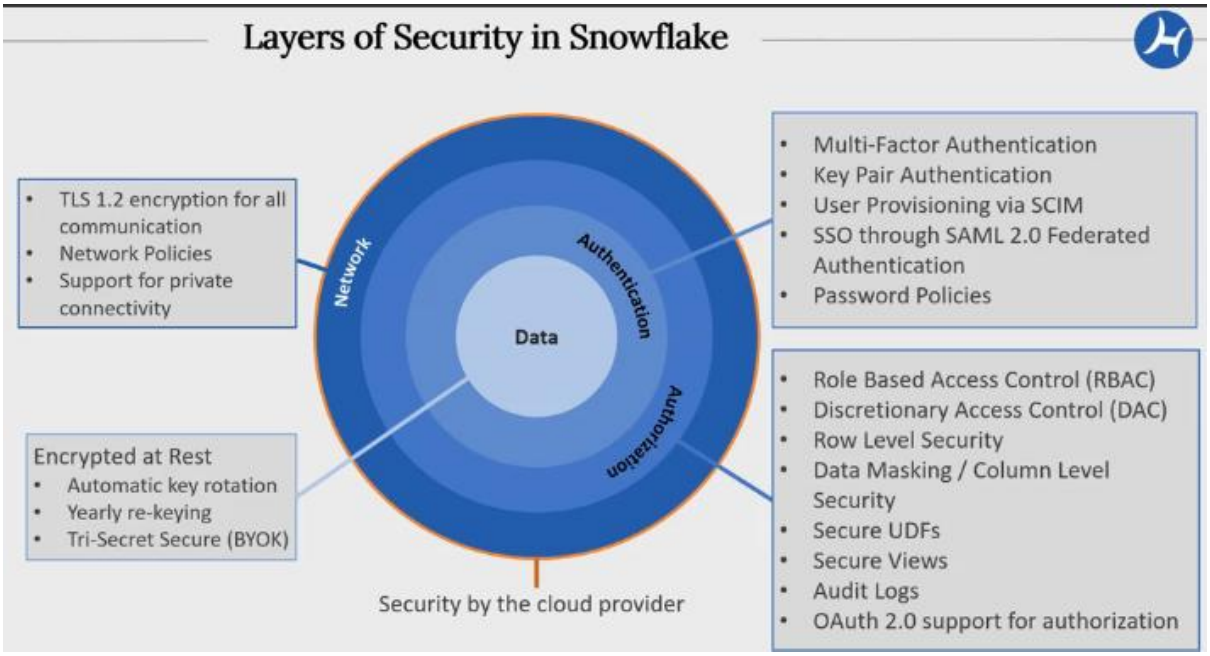
- Auto-Scaling mode is enabled by selecting different values for the multi-minimum clusters and maximum warehouse count. As a result, Snowflake starts and stops warehouses dynamically based on the workload needs. When a multi-cluster virtual warehouse using auto-scaling mode starts, the number of active virtual warehouses equals the minimum warehouse count. Snowflake spins up more warehouses according to the need, up to the maximum warehouse count. Snowflake shuts down virtual warehouses as the demand lowers until the number equals the minimum warehouse count.

- The Economy scaling policy attempts to conserve credits over performance and user experience. It doesn't spin up more virtual warehouses as soon as queuing is observed but instead applies additional criteria to ascertain whether to spin up new virtual warehouses. With the scaling policy set to Standard, Snowflake prefers to spin up extra virtual warehouses almost as soon as it detects that queries are starting to queue up. The Standard scaling policy aims to prevent or minimize queuing.

- The metadata in Snowflake allows the Snowflake query engine to eliminate partitions to optimize query execution. For example, if the query specifies a WHERE condition, partitions NOT containing the value matching that condition will NOT be scanned.

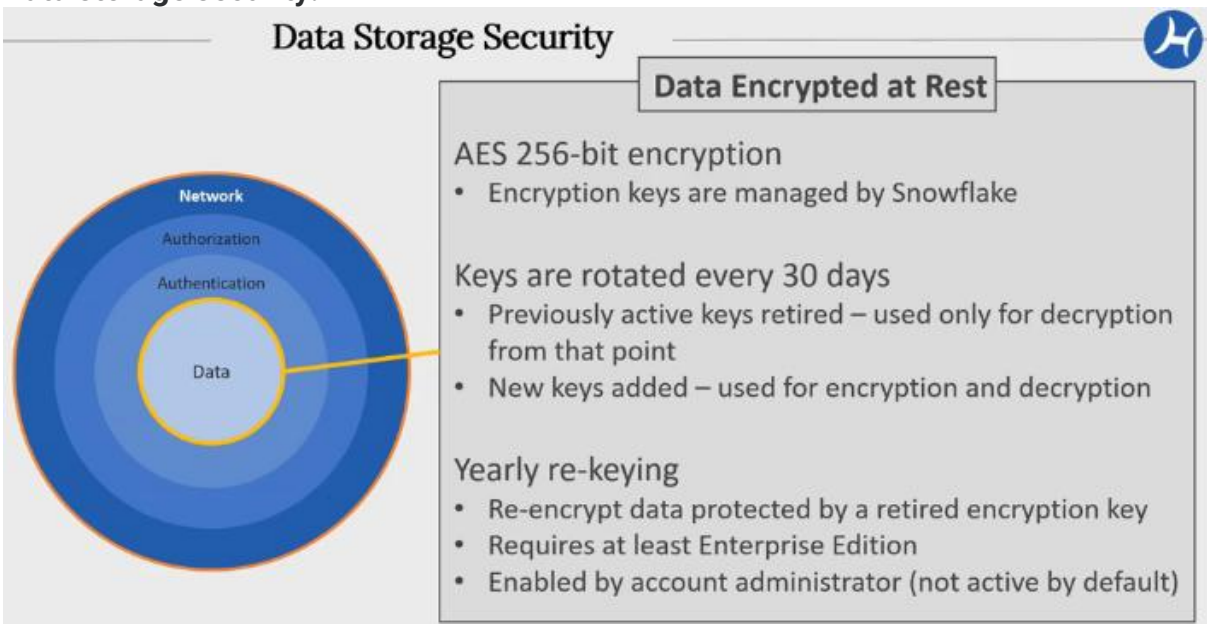
Security and Access Control in Snowflake

Security in snowflake: its implemented in multi layers.
Variety of security features available in snowflake security.
Layers of security in snowflake:



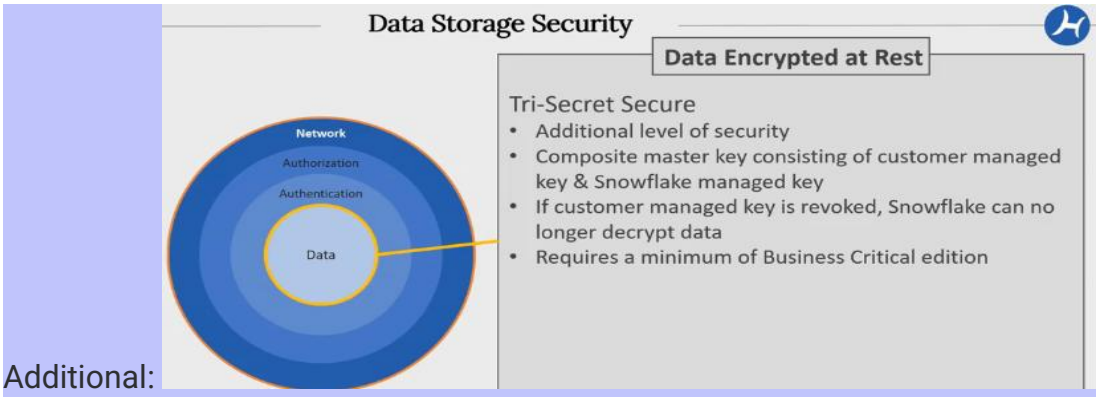
Data encryption at rest:

Data storage security.

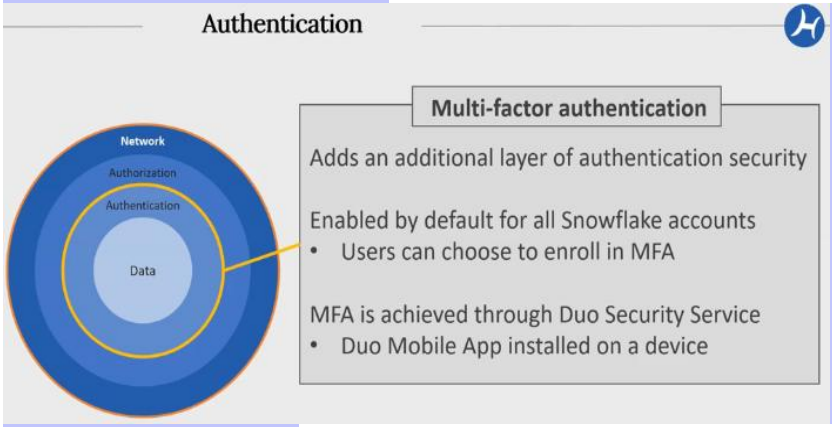


Try Secret secure refers to the combination of a snowflake managed key and a customer managed key that results in the creation of a composite master key to further protect your data.

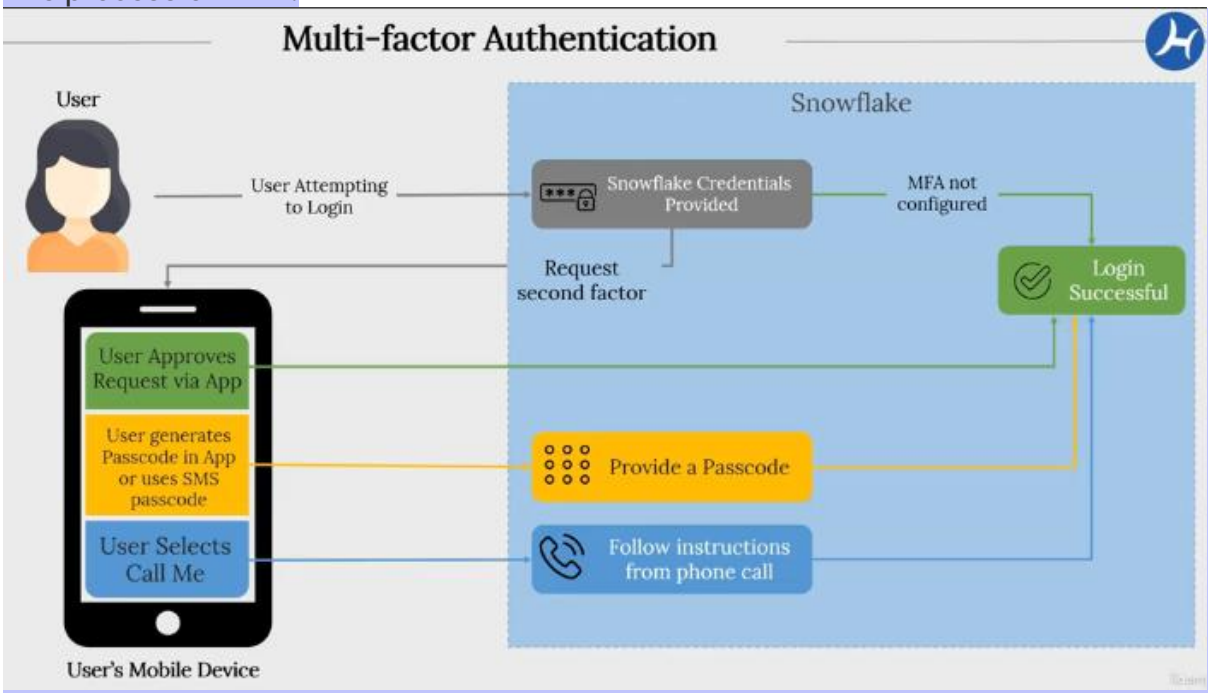
Additional:



Multi-factor authentication:



The process of MFA:

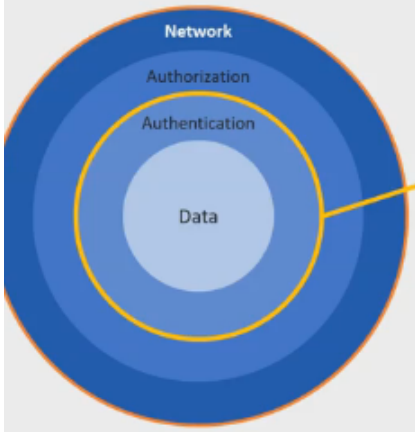


Multi-factor Authentication

- MFA is enabled for all accounts
- MFA is available in all Snowflake editions
  - So the minimum edition supporting MFA is Basic
- Snowflake recommends that users with ACCOUNTADMIN role should use MFA
- An administrator can disable MFA for a user
  - They must re-enrol if they require MFA capability
  - ALTER USER and set DISABLE_MFA = TRUE
- An administrator can also temporarily disable MFA for a user
  - The MFA for that user re-enables after the specified time period has passed
  - ALTER USER and set MINS_TO_BYPASS_MFA = <mins>

Other aspects in Authentication:

Authentication



Key Pair Authentication

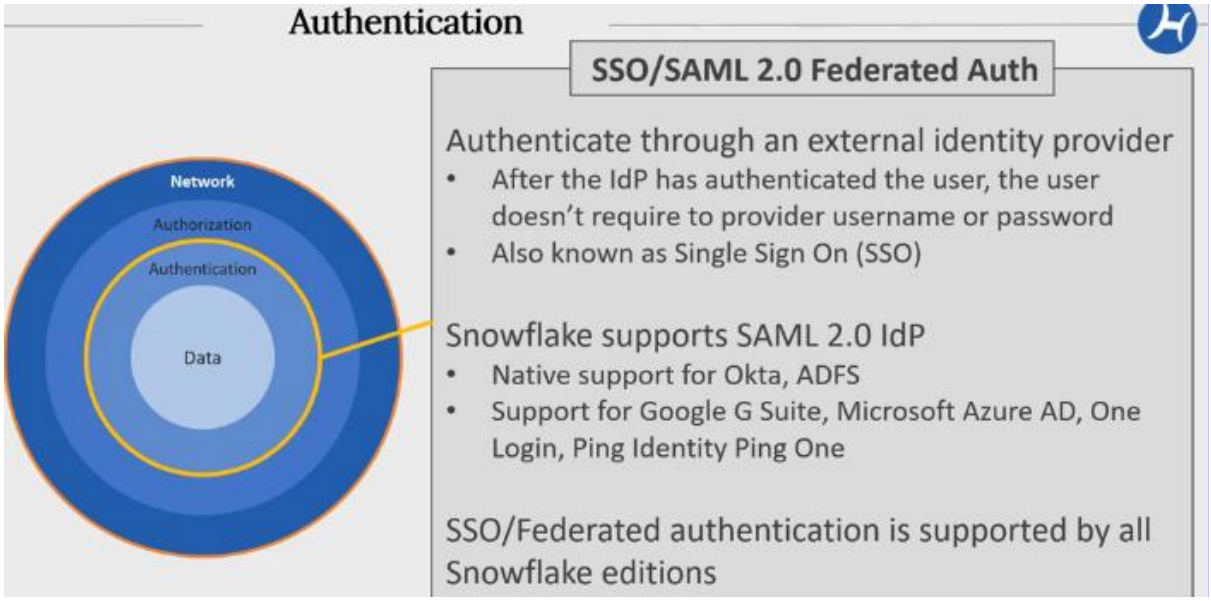
More secure alternative to username/password

Uses private & public keys

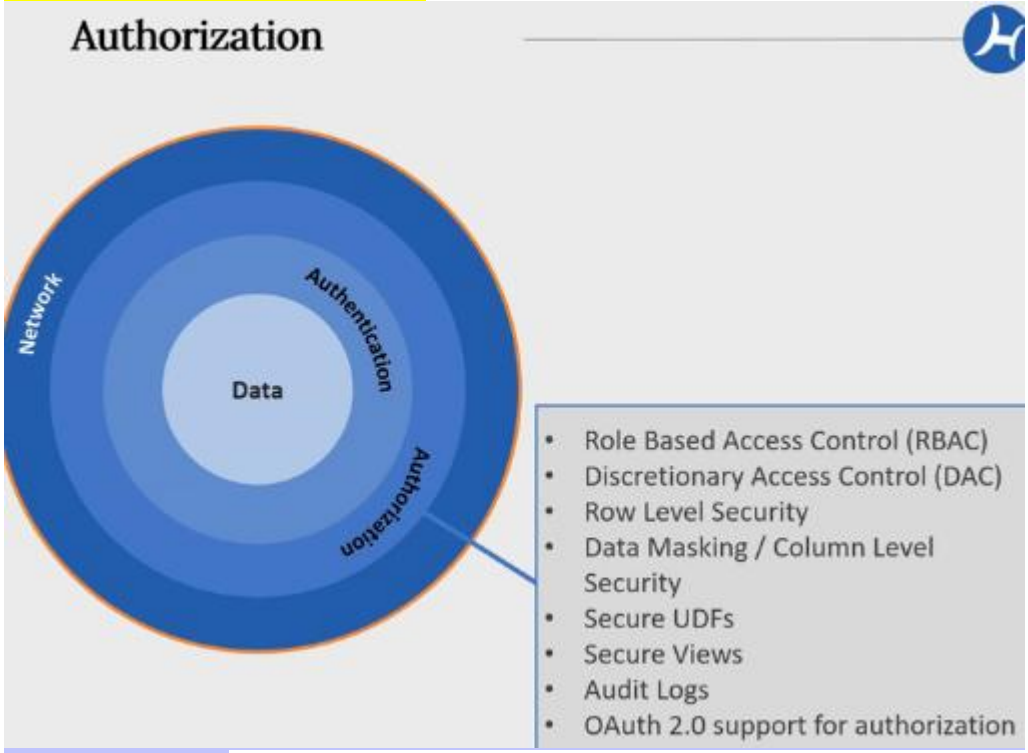
- Private key is assigned to a user in Snowflake
- Public key is issued to the user for assisting in authentication
- A user can have up to two public keys

Key-pair authentication is supported by all Snowflake editions

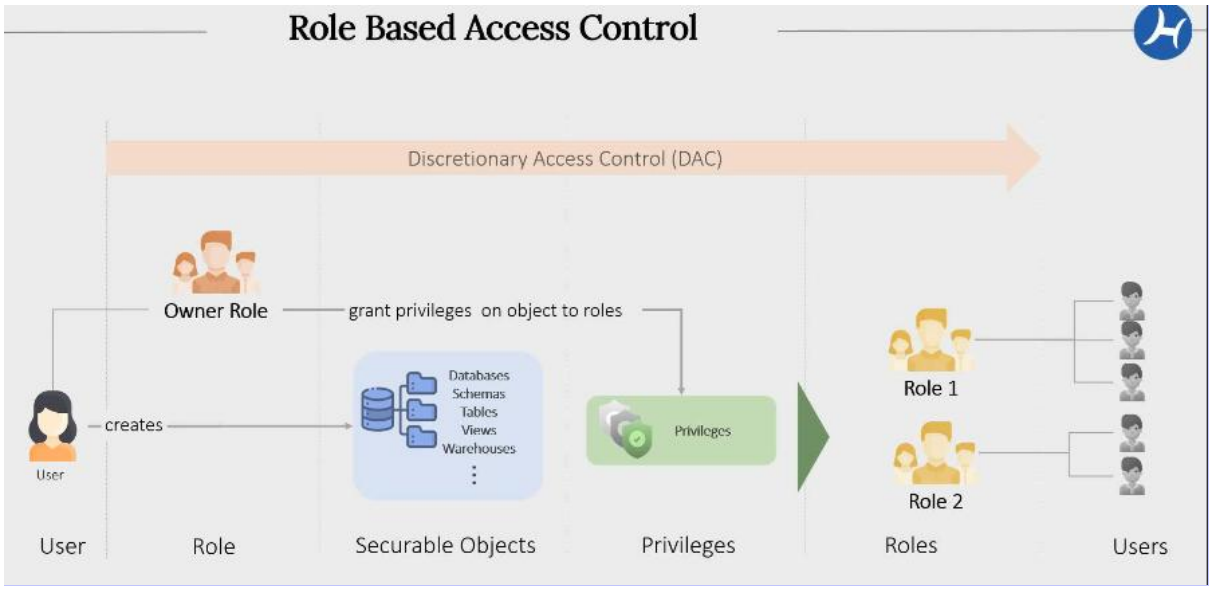
Key-pair authentication is supported by SnowSQL & all other connectors



Access Control in Snowflake:

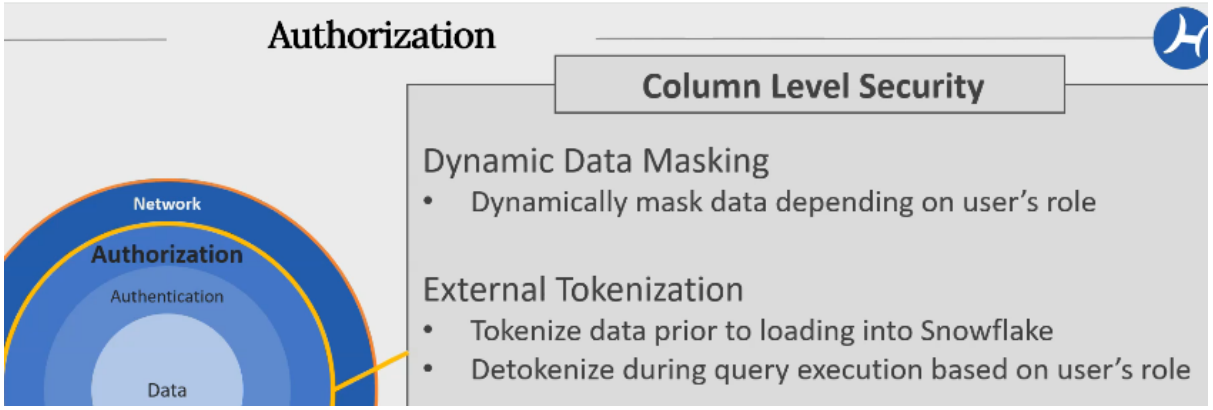


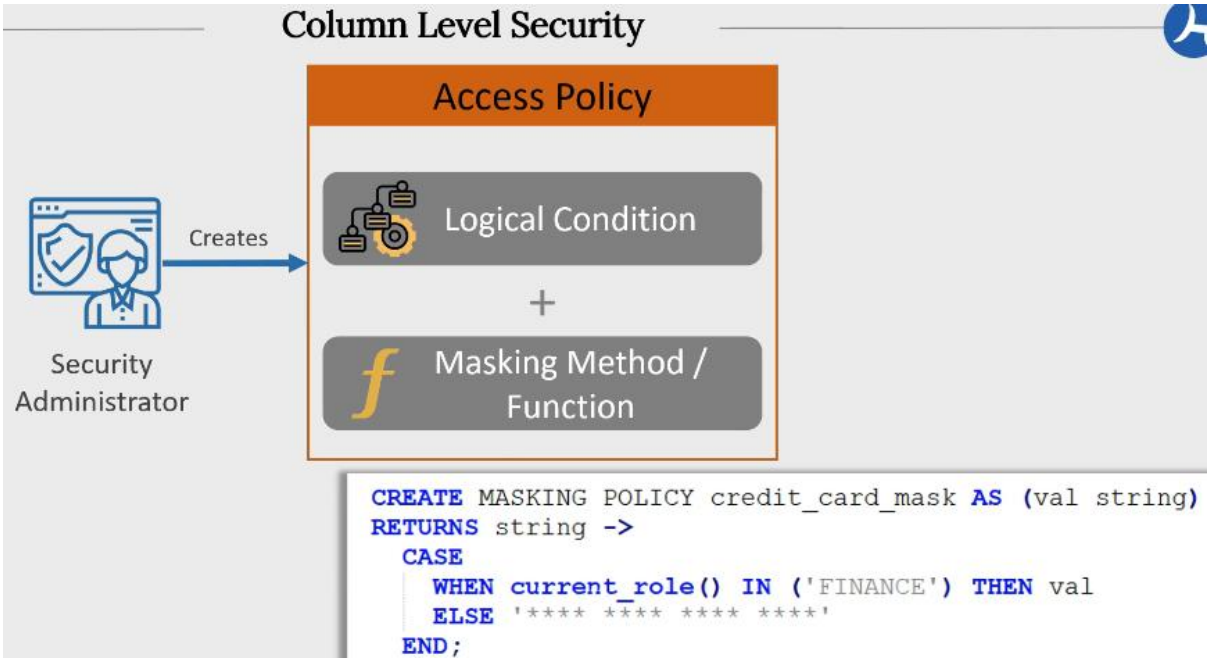
RBAC and DAC:

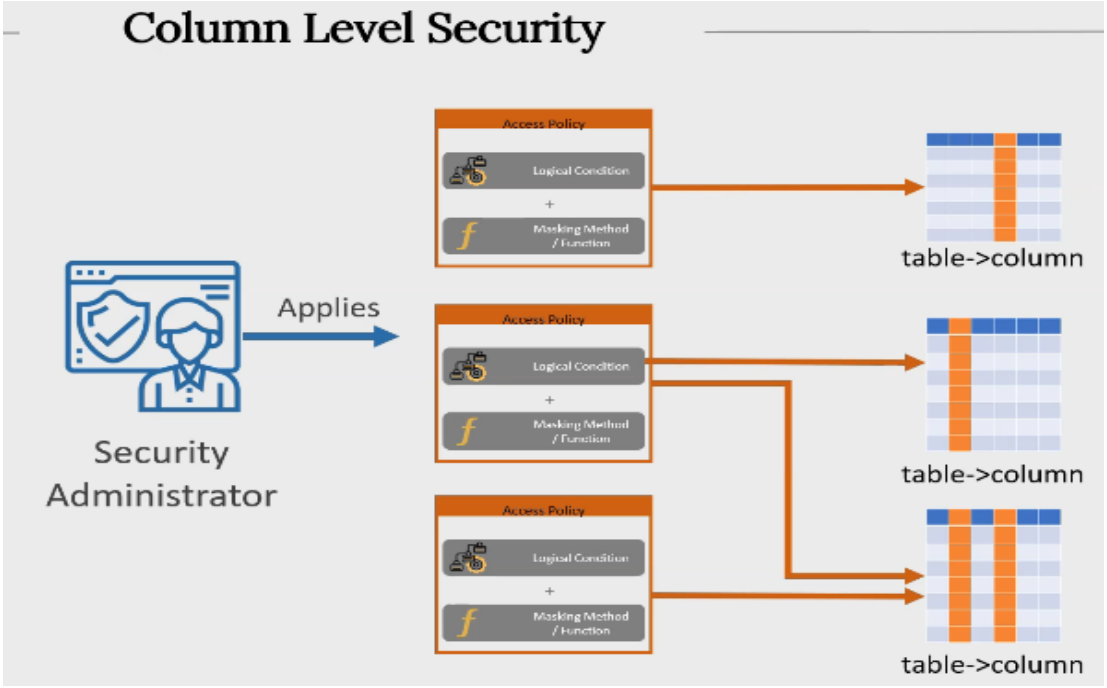


Authorization other aspects:

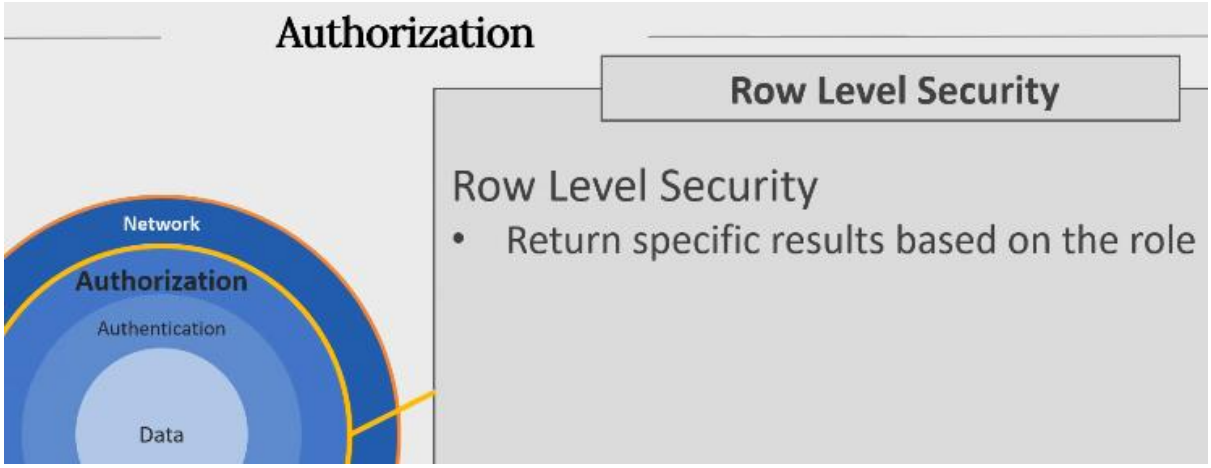
Row level and column level security.

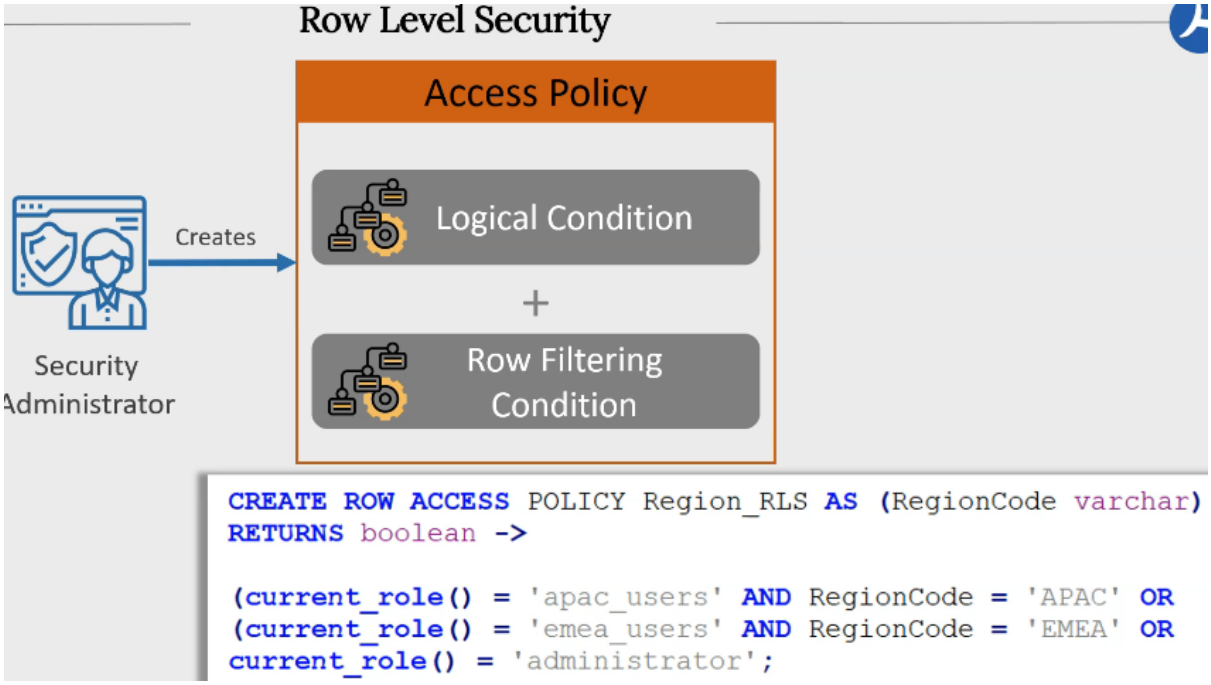




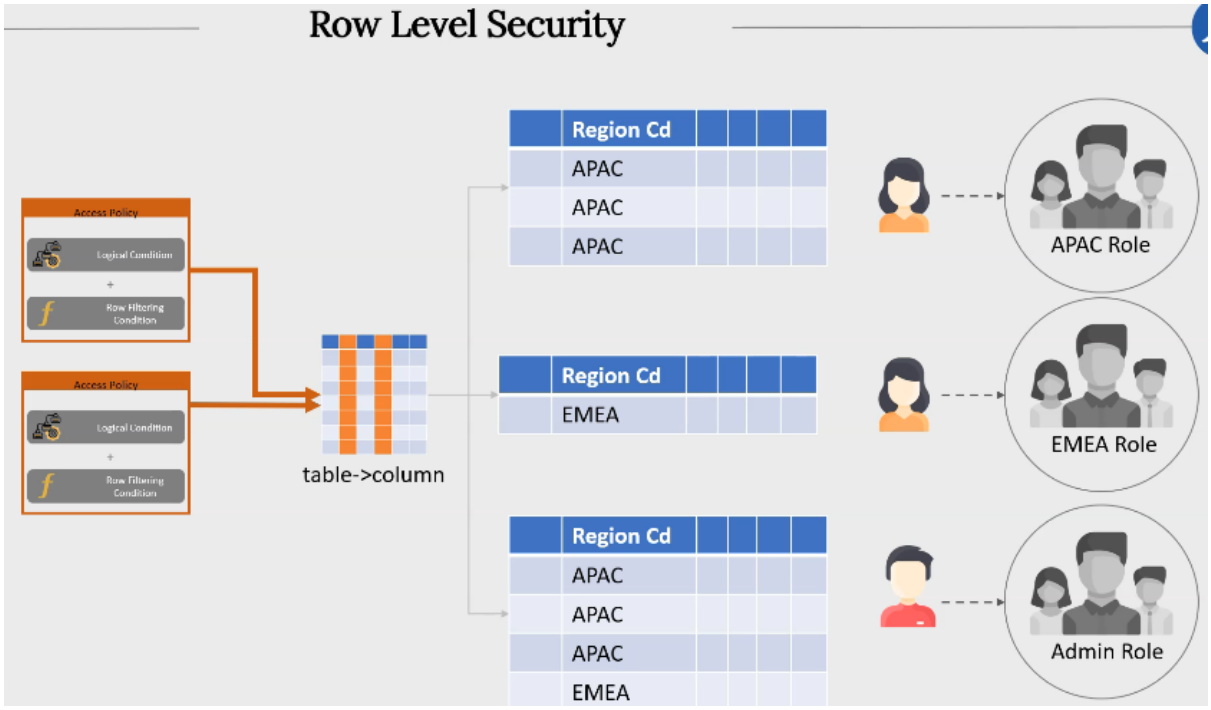


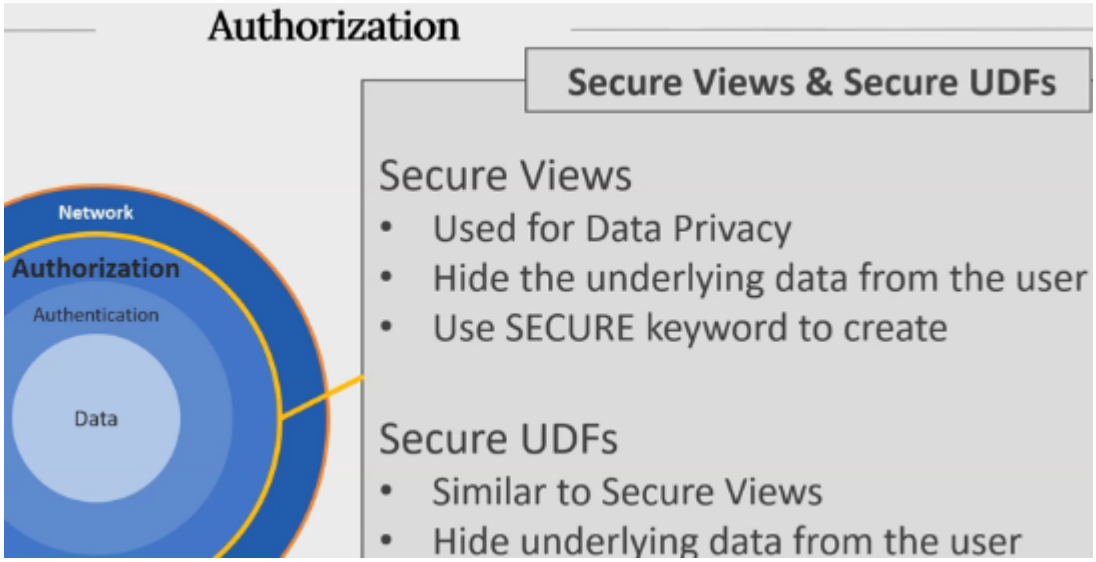
Row level security:



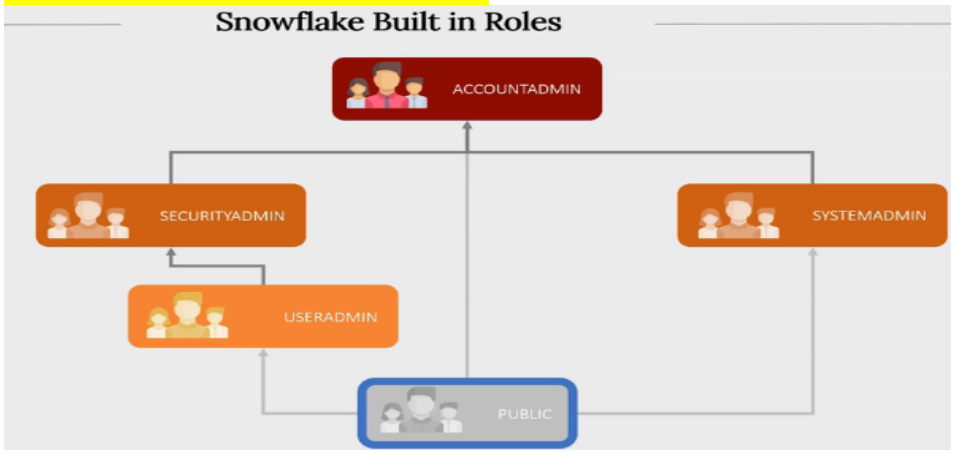



```
CREATE ROW ACCESS POLICY Region_RLS AS (RegionCode varchar)
RETURNS boolean ->

(current_role() = 'apac_users' AND RegionCode = 'APAC' OR
(current_role() = 'emea_users' AND RegionCode = 'EMEA' OR
current_role() = 'administrator';
```

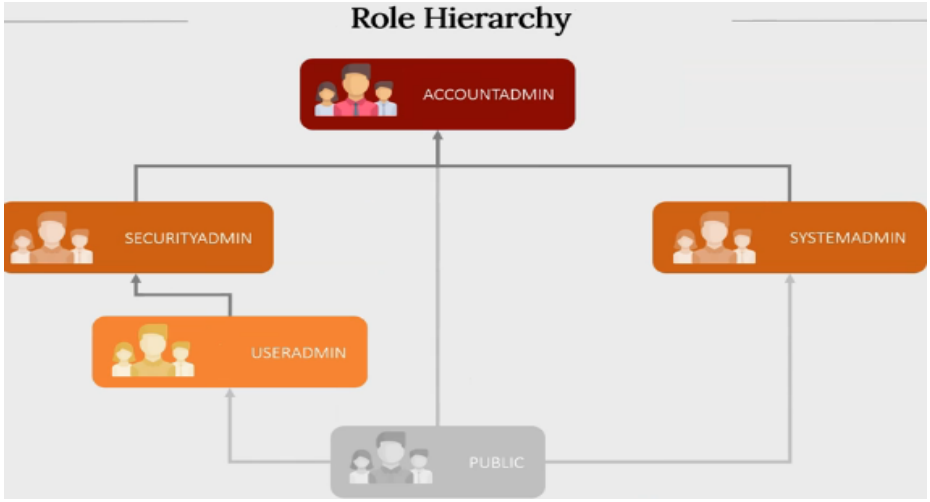




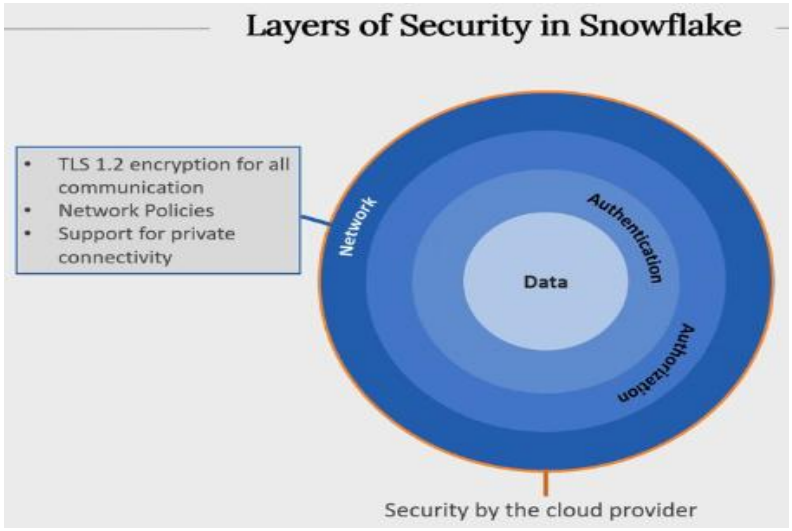
Out of the box roles in Snowflake:



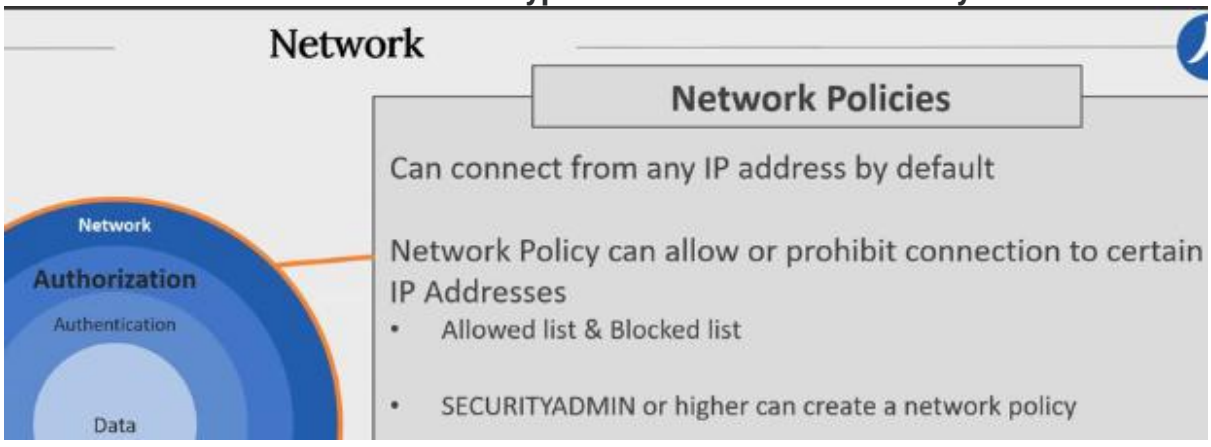
Roles in Snowflake:



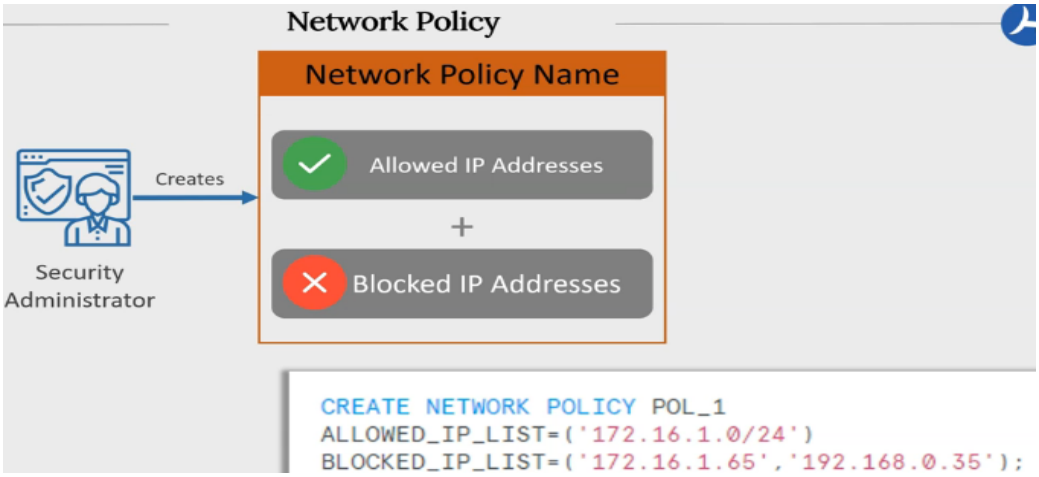
Network layer security: At the network level, Snowflake encrypts all communication by default using TLS 1.2.
The security is further enhanced through network policies, through which specific IP addresses may be allowed to connect and others may be blocked.
Additionally, Snowflake supports private connectivity, which means your connection to Snowflake can be via a private link to the cloud.



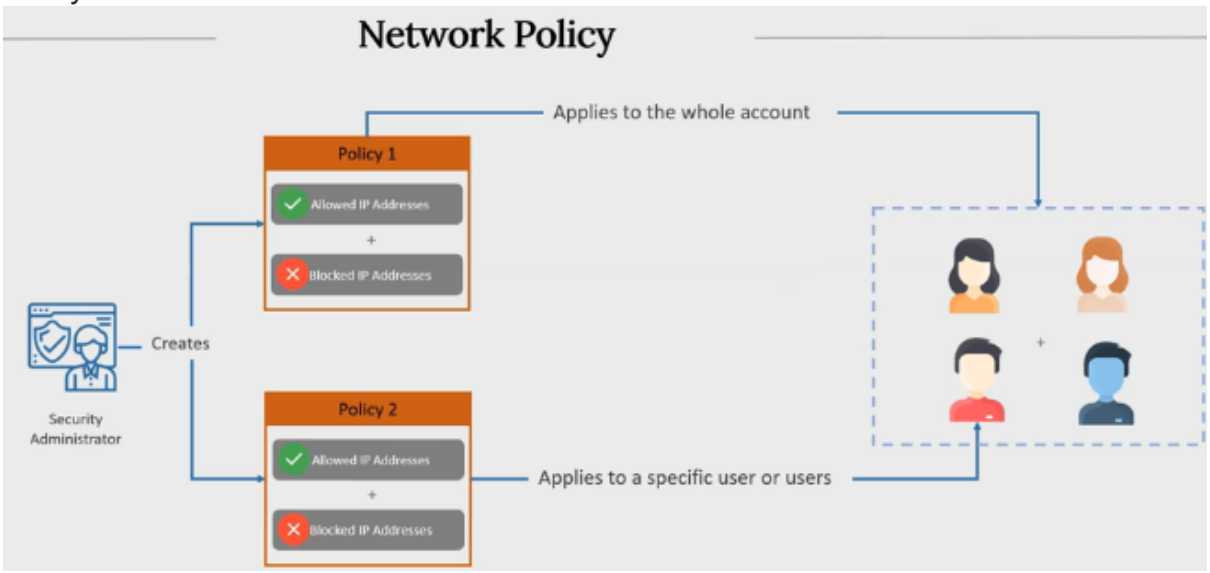
At the network level snowflake encrypts all the communications by default.



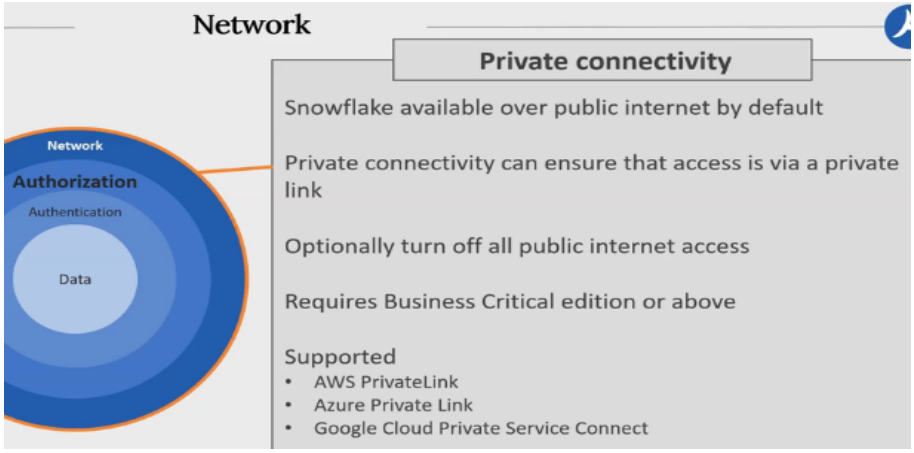
However, administrators can use network policies to allow only certain IP addresses to connect.
Or they can also deny access to specific IP addresses.
A network policy can only be created by a security administrator, a higher role or by a role with the create network policy privilege.
So network policy consists of the policy. Name a list of authorized IP addresses separated by commas, and a list of forbidden IP addresses, again separated by commas in the authorized or disallowed IP address list.



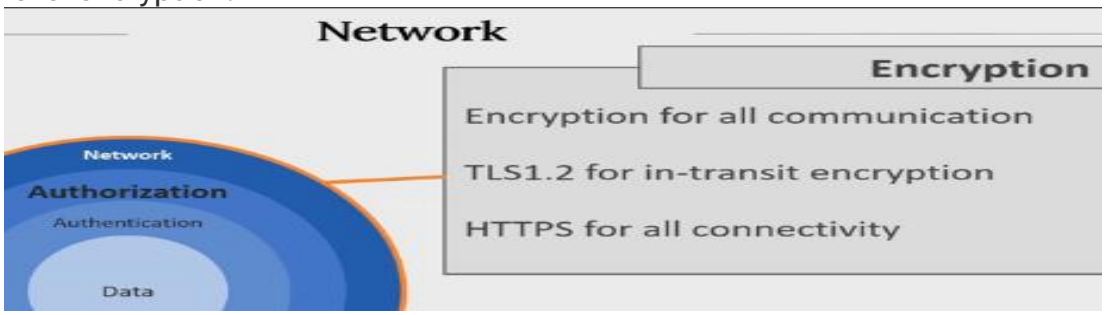
You can specify an individual IP address or an IP address range. However, do note that the network policies at present only support IP version four addresses.
If both the allowed and blocked IP lists are populated, Snowflake applies the blocked list first, followed by the allowed list.
So network policies can be created and applied at the account level, in which case the policy applies to the entire account.
Network policies can also be assigned to the individual user, in which case they apply solely to that user.

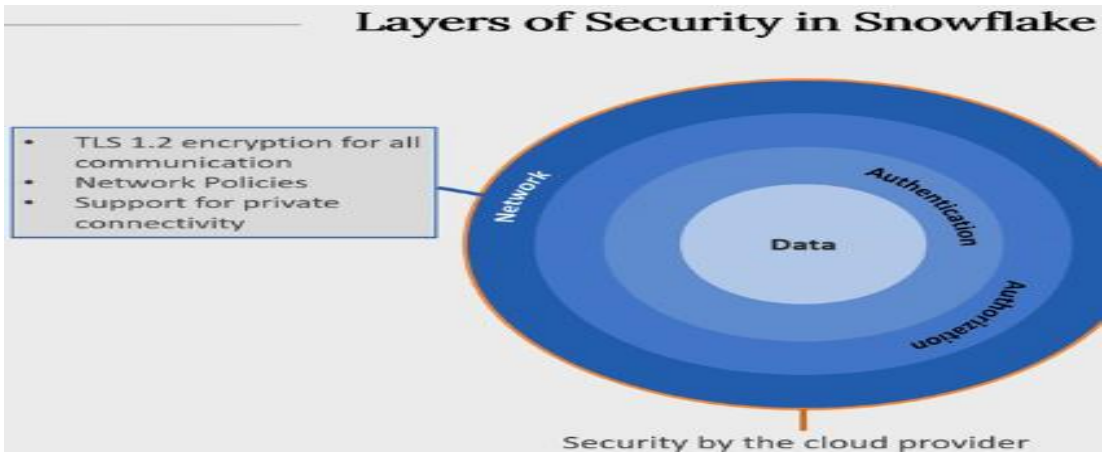


When a user level network policy is applied, that policy takes precedence over account level policy.
Now let's talk about private connectivity.
So by default, your snowflake instance is available over the public Internet with access protected by different security measures such as MFA, HTTPS and network rules.



Now, if your organization demands that your snowflake instance not be available through the Internet. Snowflake supports private connectivity through which you can ensure that access to your snowflake instance is via a private connection.
And then you can optionally block all Internet access. It's good to note that private connectivity to Snowflake requires at least the business critical edition.
So depending on your cloud provider, Snowflake provides private connectivity methods specific to that cloud. So it supports AWS PRIVATELINK. Azure Privatelink and Google Cloud Private Services Connect. Finally, let's talk about encryption in transit. So Snowflake encrypts all communication end to end. TLS 1.2 is used to encrypt data while it is in transit. Everything in Snowflake is connected over HTTPS, including connectivity to the Snowflake web UI connectivity via JDBC, via ODBC, as well as via the Python connector and other connection mechanisms. And then all access to snowflake services is accomplished through rest APIs, which are also invoked over the HTTPS protocol. So this lecture covered the network level security capabilities that Snowflake provides. We talked about TLS 1.2, which is that transport level encryption.





MFA is enabled by default for all Snowflake accounts and is available in all Snowflake editions.

Snowflake supports masking policies that may be applied to columns and enforced at the column level to provide column-level security. Column-level security is achieved by dynamic data masking or external Tokenization.

Row-level security is implemented by creating row access policies, which include conditions and functions that govern which rows are returned during query execution.

Secure views can be used to return only certain rows from a table. Additionally, secure views hide the underlying data by removing some of the internal Snowflake optimizations.

In Snowflake, all data at rest is encrypted using AES 256-bit encryption.

Snowflake encrypts all data in transit using Transport Layer Security (TLS) 1.2. This applies to all Snowflake connections, including those made through the Snowflake Web interface, JDBC, ODBC, and the Python connector

Administrators can configure the system to allow or deny access to specific IP addresses through network policies. A network policy consists of the policy name, a comma-separated list of allowed IP addresses, and a list of blocked IP addresses

Snowflake supports SCIM 2.0 and is compatible with Okta and Azure Active Directory. SCIM is an open standard that provides automatic user provisioning and role synchronization based on identity provider information. When a new user is created in the identity provider, the SCIM automatically provisions the user in Snowflake. Additionally, SCIM can sync groups defined in an identity provider with Snowflake roles.

Snowflake is pre-configured with the following roles. ACCOUNTADMIN is a full-privilege account administrator role. USERADMIN provides the ability to create USERS and ROLES. SECURITYADMIN receives privileges from USERADMIN and can govern global object grants. SYSADMIN can create and manage the majority of Snowflake objects. ORGADMIN manages the operations at an organizational level. There is also the PUBLIC role, which is automatically assigned to everyone.

ACCOUNTADMIN is the most powerful role in a Snowflake account. Due to the role hierarchy and privileges inheritance, the ACCOUNTADMIN inherits all the privileges that SECURITYADMIN & USERAMDIN has.

Snowflake's access control system is built on the RBAC idea, which means that privileges are issued to roles and roles to users. The privileges associated with a role are given to all users assigned to it. Snowflake also supports discretionary access control (DAC), which means that the role that created an object owns it and can provide access to other roles to that item.

Extending snowflake functionality:

Use Defined Functions

- User Defined Functions (UDFs) can be created to create custom functionality
- Supported Languages
  - SQL
  - Java
  - JavaScript
  - Python
- Scalar UDFs return one single value for each input
  - E.g. MAX(2,3) returns 3
- Table UDFs or Tabular UDFs (UDTFs) can return zero, one or several rows for an input. Each result row can have multiple columns
  - E.g. FLATTEN

External Functions

- Type of UDFs
- Doesn't have its own code
- Code is stored and executed outside of Snowflake
- Snowflake only maintains metadata
- External functions can be written in languages that standard UDFs don't support
- They can also access functions and libraries that typical UDF can't
- They can of course make calls to external API services.

Secure UDF's:

Secure UDF

- Typical UDF uses SQL execution optimizations which can enable access to underlying data.
- This may allow access to data to users indirectly
  - Security concern
- Secure UDFs solve this by not using the optimization
  - Thus, ensuring users have no access to underlying data, even indirectly.
- Users can also see SQL logic or JavaScript code behind a typical UDF
  - You may want to hide the code for security reasons
- Secure UDF solves this issue by hiding the definition
  - Only available to authorized users
- Performance vs. Security

Stored Procedures:

Stored Procedures

- Procedural code with if-else logic, looping, and other features
- Construct dynamic SQL statements and execute

- Stored procedures can return a single value or a tabular data
- Not mandatory to return a value

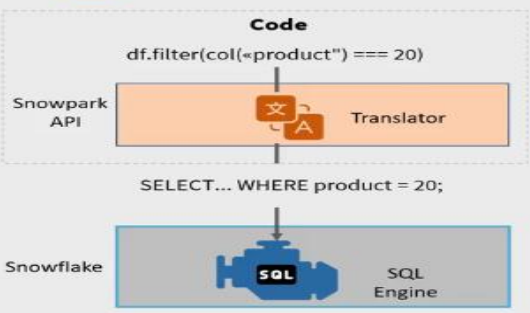
- UDFs & Stored Procedures appear similar but serve a different purpose
- UDF takes an input, performs computation on the input, and returns a value.
- Stored Procedures execute one or more SQL queries.
  - They are often used for performing administrative tasks

Stored Procedures

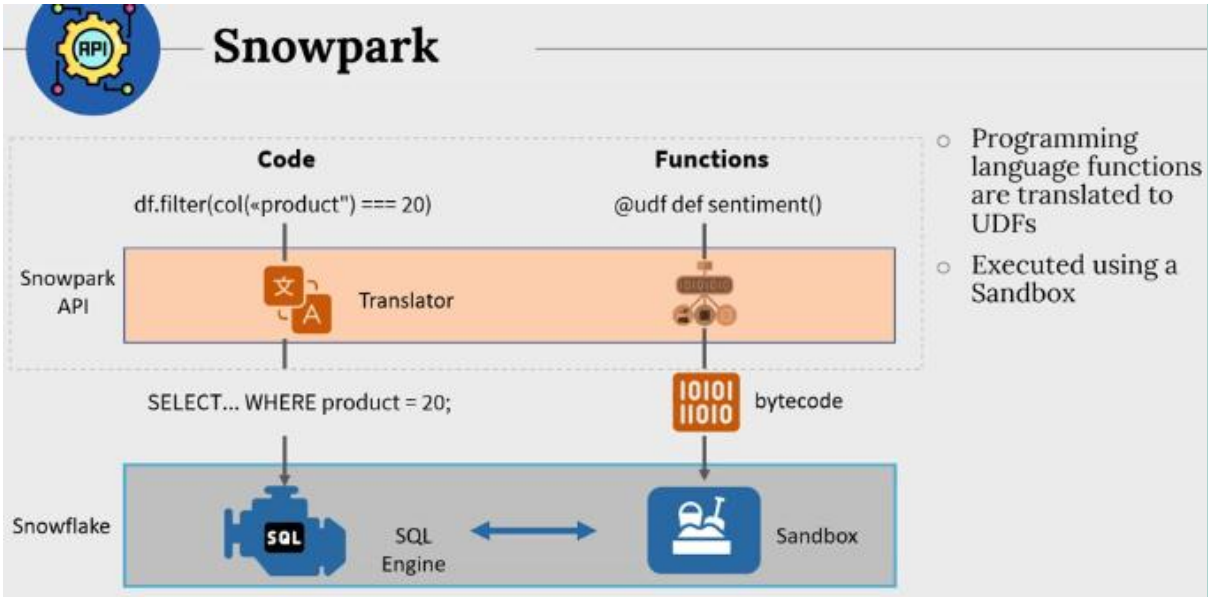
- Caller's Right
  - Use privileges of the caller
- Owner's Right
  - Default
  - Uses privileges of the creating role
- Stored Procs can be written in
  - SQL
  - JavaScript
  - Snowflake Scripting
  - & when using Snowpark, you can use Java, Python or Scala

Snowpark:

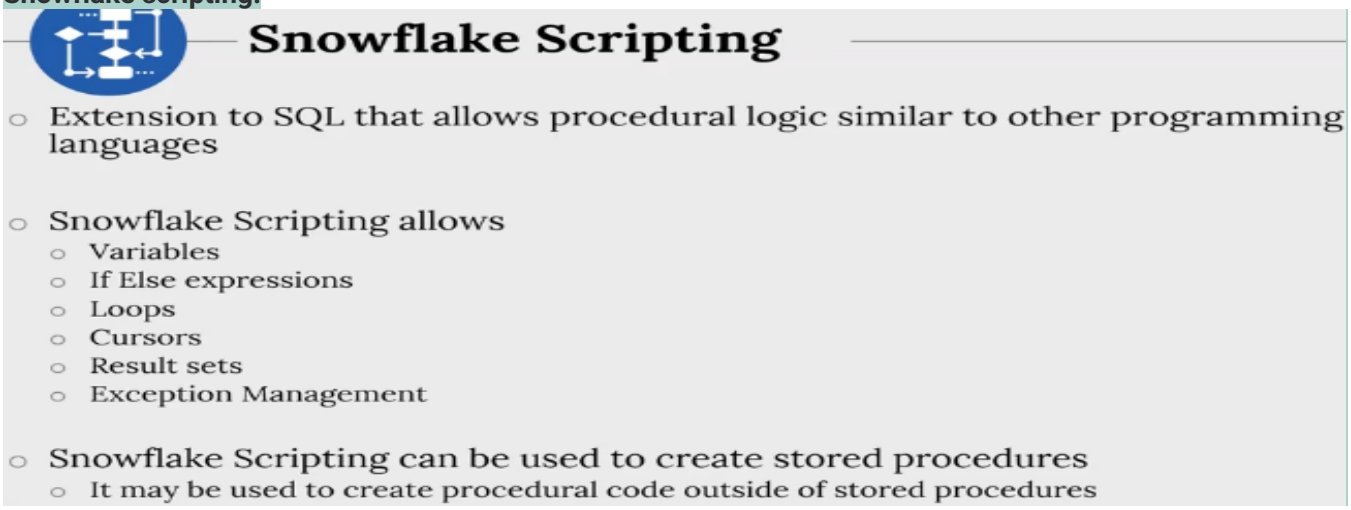
Snowpark

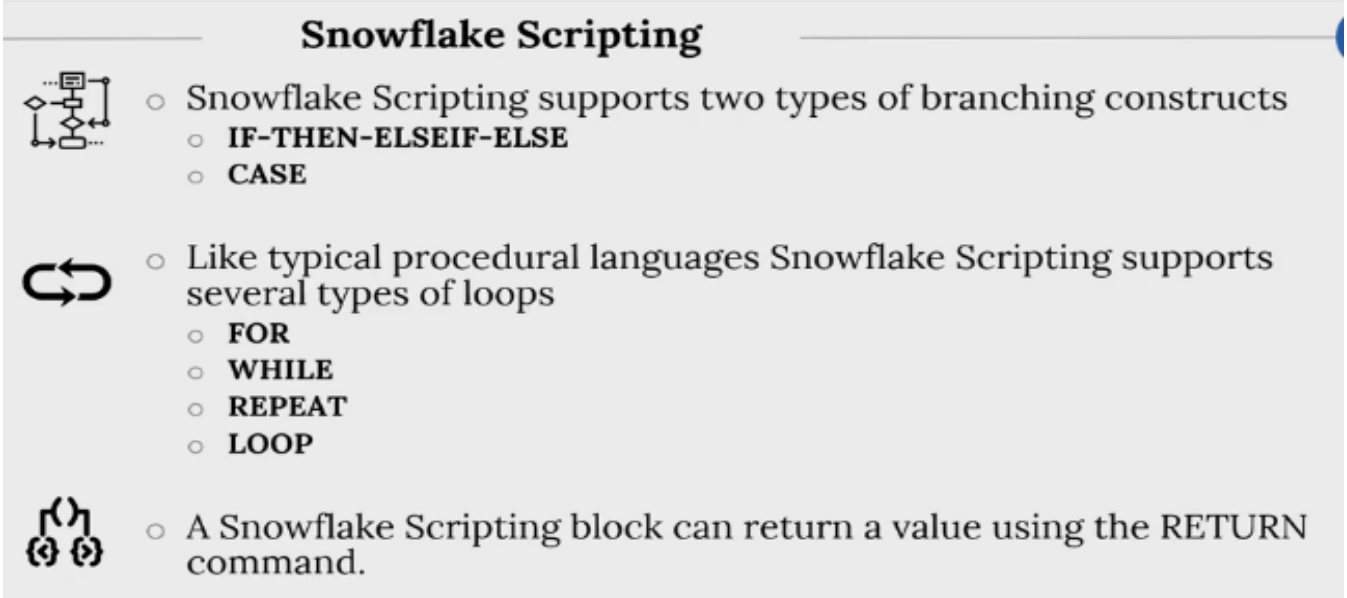


- Snowpark API
  - Provides programming language support for building SQL that is executed on the Server
  - Programmers can create code in a familiar language
- The code gets translated to SQL and is executed on Snowflake.
  - Takes advantage of the Snowflake scale & parallelism
- The Snowpark API functions are evaluated in a lazy manner, i.e. nothing is done until a function is called that performs an action



Snowflake scripting:





Extending Snowflake functionality Quiz:

An external function, unlike other UDFs, does not include its own code; instead, it invokes code that is stored and run outside of Snowflake. For an external function, the only thing that is kept inside Snowflake is information that Snowflake uses to invoke the remote service that contains the code.

Snowflake Scripting is an extension to SQL that allows you to use procedural logic similar to that found in programming languages. Snowflake Scripting allows you to use variables, if-else expressions, looping, cursors, manage result sets, and allows you to handle errors. Snowflake scripting is typically used to create stored procedures, but it may also be used to create procedural code outside of a stored procedure.

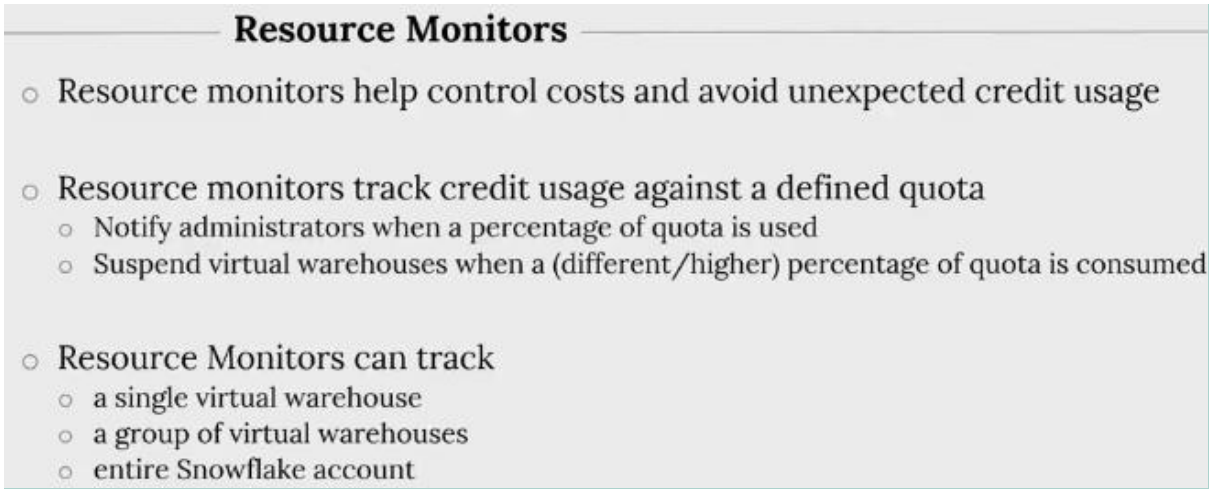
Snowpark is a library created by Snowflake that provides APIs for accessing and processing data in applications written in a programming language other than SQL. Snowpark allows programmers to utilize common programming languages such as Java, Scala, and Python to construct apps that handle data using standard programming structures. Snowpark automatically converts the data-processing programming constructs to SQL and pushes it down to Snowflake for execution. As a result, developers may utilize a familiar language while benefiting from Snowflake's scale and execution engine.

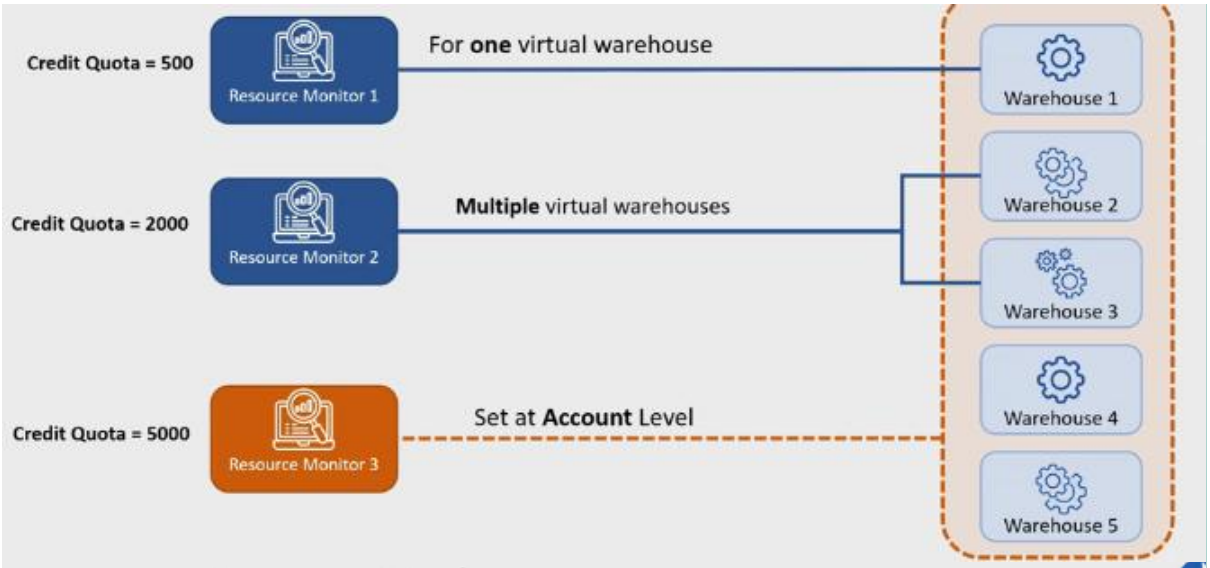
Stored procedures are often used to perform recurring administrative activities, e.g., in a particular organization setting up a new user on the system may require creating the user, granting them several roles, creating a private database from them, etc. These steps can easily be placed in a stored procedure, and then the stored procedure can be called whenever there is a requirement to create a new user.

You can create functions using typical programming languages such as Java, Python, or Scala, and those functions can be exposed in Snowflake as UDFs. So, you can use these UDFs in your SQL just like any other UDFs. To execute these UDFs, Snowflake creates a run-time environment sandbox within the virtual warehouse houses. The UDFs execute inside the sandbox. This approach also ensures default parallel execution of the UDFs because they will use Snowflake infrastructure to scale.

Resource Management in Snowflake:

Resource Monitors: These help to monitor the virtual warehouses.

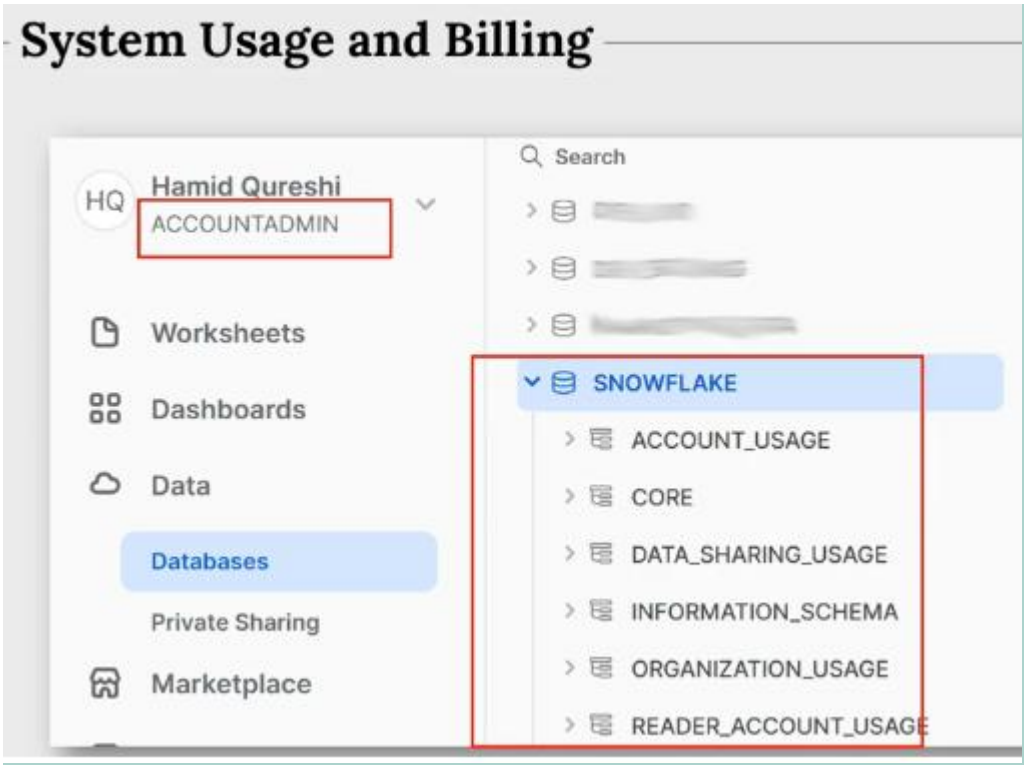




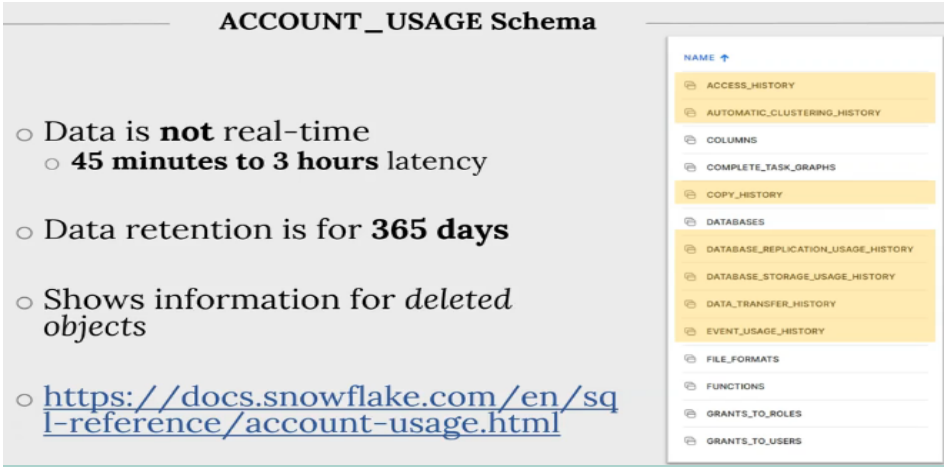
Resource Monitors

- It is not possible for more than one Resource Monitor to be monitoring the same virtual warehouse
  - Account Level monitoring is an exception
- Resource monitors do not control the credit usage by serverless features such as
  - Snowpipe
  - Automatic Reclustering
  - Materialized View maintenance
  - Etc.
- Resource monitors can only suspend user-managed virtual warehouses
- Resource monitors can not manage the cost of cloud services
- Only account administrators (users with the ACCOUNTADMIN role) can create resource monitors

Viewing Usage and Billing in Snowflake:
System usage and Billing –



Account Usage schema:



Information Schema:

The ACCOUNT USAGE schema consists of several views that provide usage metrics and metadata information at the account level. Data provided by the ACCOUNT_USAGE views is NOT real-time and refreshes typically with a lag of 45 minutes to 3 hours, depending on the view. The data in these views are retained for up to 365 days.

The data provided via the INFORMATION_SCHEMA views is real-time, and there is no latency in the information provided. So, if you are asked which schema should be used if there is a requirement to view real-time data, then the views in INFORMATION SCHEMA should be used as they contain real-time information.

A resource monitor cannot control the costs of cloud services. A warehouse-level resource monitor can monitor credit usage by Cloud Services, but the resource monitor cannot suspend the cloud services.

Resource monitors can track & manage a single virtual warehouse against a defined quota. Resource monitors can be created to track the credit usage of multiple virtual warehouses together. Resource Monitors can also be created at the account level, which means that such resource monitors track credit usage at the account level, considering the credit usage of all virtual warehouses.

Resource monitors help manage virtual warehouse costs and avoid unexpected credit usage. Credit usage can be controlled with resource monitors by monitoring credit usage against a defined upper limit, notifying administrators when a certain percentage of the limit is reached, and even suspending virtual warehouses if necessary.

Another one in Udemy

Snowflake Complete course

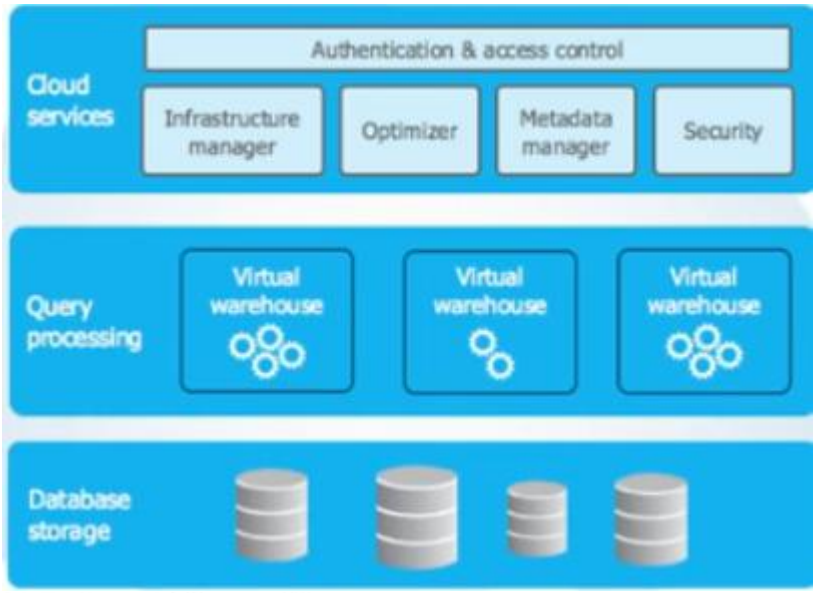
Learning Plan:

Day 1	Introduction to Snowflake	Day 14	Caching in Snowflake
Day 2	Snowflake Architecture and Virtual Warehouses	Day 15	Time Travel and Fail-Safe
Day 3	Trial Account Creation and Snowsight	Day 16	Zero-Copy Cloning
Day 4	Micro Partitions and Cluster Keys	Day 17	Table Types
Day 5	Snowflake Pricing(Cost)	Day 18	Role Based Access Control
Day 6	Loading and Transforming Data	Day 19	Roles and Users Creation
Day 7	Stages - External Stages	Day 20	Dynamic Data Masking
Day 8	AWS - Snowflake Integration	Day 21	Views and Materialized Views
Day 9	Internal Stages and SnowSQL	Day 22	Data Sharing
Day 10	Processing Semi-Structured Data	Day 23	Tasks for Scheduling
Day 11	SnowPipe	Day 24	Streams for CDC
Day 12	Troubleshooting SnowPipe	Day 25	External Tables
Day 13	Unloading Data	Day 26	UDFs and Stored Procedures

Traditional WH Vs Snowflake

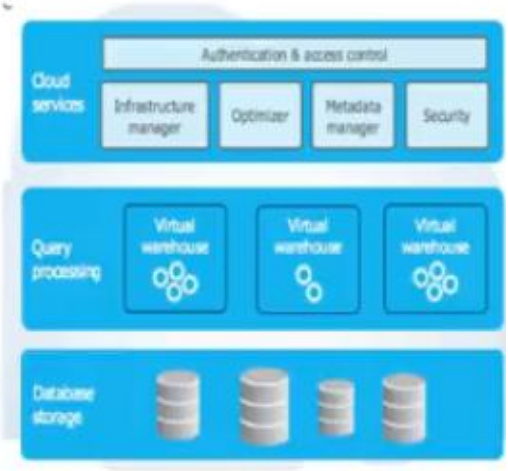
Feature	Traditional WH	Snowflake
Infrastructure cost	yes	No Infrastructure cost
Handle semi structure data	Need ETL tools	Snowflake can process
Data loading and unloading	Need ETL tools	Can be done by using "COPY"
Scalability	No	Highly Scalable (support Scale-up and Scale-out)
Database Administration	Highly Required	In-built performance optimization with its micro partitions and cluster keys
Data Backup	Need additional storage	Easy and no cost with "Cloning"
Data Recovery	Difficult	Very easy with "Time Travel"
Data Sharing	Difficult	Easy with Data Sharing feature
Change Data Capture	Need ETL tools	Can be done by using "Streams"
Scheduling	Tools required	Can schedule by using "Tasks"

Snowflake Architecture:
Layers in architecture and the importance of each layer.

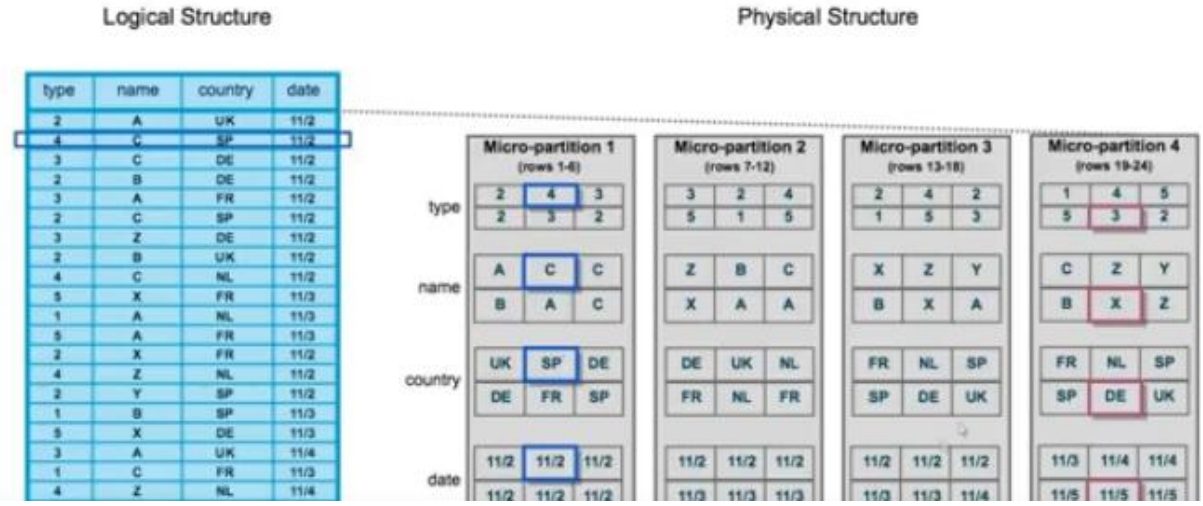


Database Storage Layer:

1. Stores table data and query results
2. Data will be stored in columnar format
3. Data will be stored in micro partitions
4. Snowflake manages all aspects of how this data is stored i.e. the data organization, file size, structure, compression, metadata, statistics
5. 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.
6. We can define cluster keys on large tables for better performance.



Since the data is stored in columnar format in snowflake the performance is good.

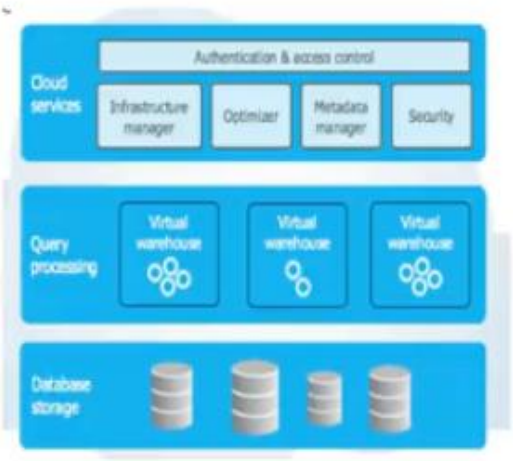


In snowflake the data is stored in micro partitions in columnar format.

Query processing layer→

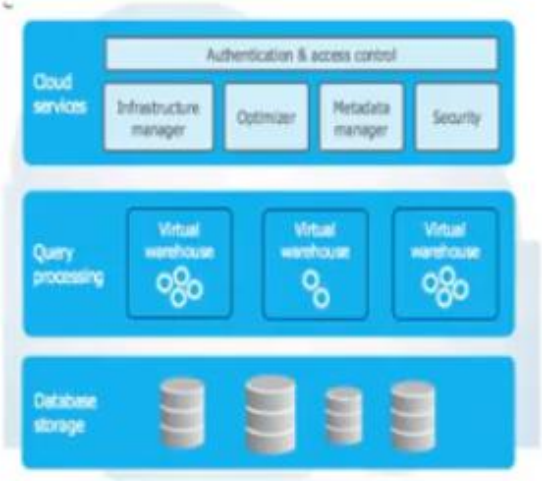
Query Processing Layer:

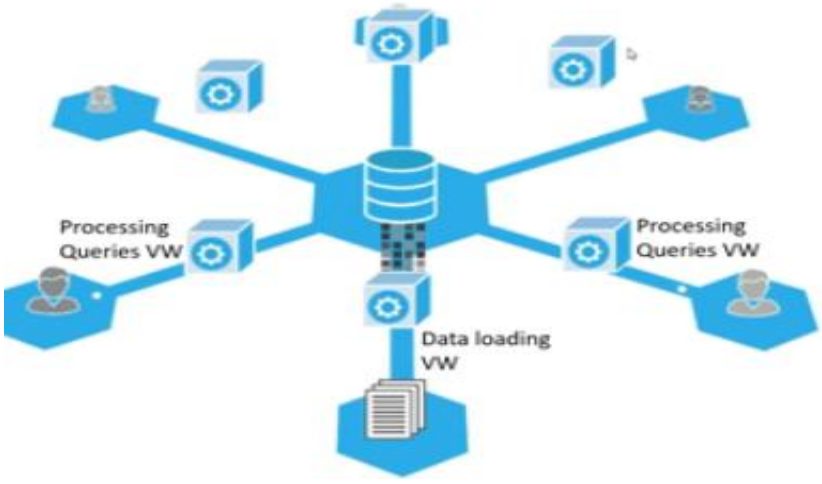
1. This is the actual processing unit of snowflake
2. Snowflake processes queries using "virtual warehouses"
3. Each virtual warehouse is composed of multiple compute nodes allocated by Snowflake from a cloud provider.
4. On AWS they are a group of EC2 instances and on AZURE a group of Virtual Machines
5. Compute cost will be calculated on the basis of query execution time on virtual warehouses
6. Virtual Warehouses are considered as the muscle of the system
7. Can scale up and scale down easily
8. Auto-Suspend and Auto-Resume is available



Cloud Services Layer:

1. Collection of services that coordinate activities across Snowflake
2. This is the Brain of the snowflake
3. Authentication and access control
4. Infrastructure management
5. Metadata Management
6. Security
7. Manages all Serverless tasks like Snowpipe, Tasks, Materialized view maintenance etc.





Requirement specific Warehouse choosing

Warehouses come in different sizes.

The size of warehouse determines the number of servers in each cluster in the warehouse.

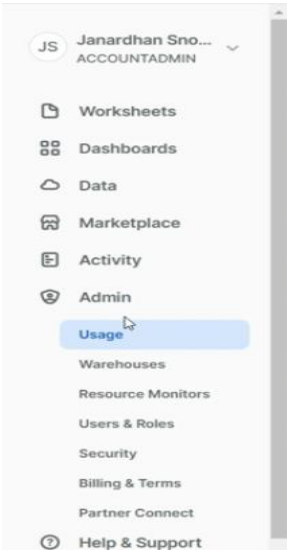
X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large
1	2	4	8	16	32	64	128

Snowflake provides 2 options for increasing the computer resources.

You can increase the VW anytime using the web UI or SQL Interface.
Scale up or vertical scaling – increase the size of the virtual Warehouse.
(warehouse resizing)- Increase in the size of the VW if your queries are taking too long or data loading is slow.
Scale out or horizontal scaling– increasing the no of clusters to await the queries going to que.
Snowflake offers upto 10 clusters to increase as part of scale out option.

The different roles in snowflake account we can see are below:
ACCOUNTADMIN, PUBLIC, SECURITYADMIN, SYSADMIN, USERADMIN, ORGADMIN

If we login with security admin or account admin then only the below option is visible to us:



Within the Warehouses we have an option called multi cluster warehouse.
Scaling policy available – standard and economy.
Standard meaning if we don't want to wait and don't want to make the queries to be in que, then we go for standard.

Clustering

- Clustering is a key factor in query performance, It reduces the scanning of micro partitions.
- A clustering key is a subset of columns in a table that are explicitly designated to co-locate the data in the table in the micro-partitions.
- Initially the data will be stored in micro partitions in the order of inserting/lopading records, then will be realigned based on the cluster keys.
- We have to choose proper Cluster keys.
- We can define cluster keys on multiple columns as well.
- We can modify the cluster keys based on our requirements, this is called as re-clustering.
- Re-clustering consumes credits, number of credits consumed depends on the size of the table.

- Load Types
- Bulk Loading v/s Continuous Loading
- Copy Command
- Transforming Data

Bulk load & continuous load.

Bulk Loading Using the COPY Command

- This option enables loading batches of data from files already available in cloud storage(External Stages)
- We have to create Storage Integration objects to extract data from these cloud storages.

(Or)

- Copying data files from a local machine to an Internal Stage(i.e. Snowflake) before loading the data into table.
- Bulk loading uses virtual warehouses
- Users are required to size the warehouse appropriately to accommodate expected loads using the COPY command.

COPY Command

```
COPY INTO TABLENAME
        FROM @STAGE
        file_format= (...)
        files = (filename1, filename2 ...)
        (or)
        pattern = '.*filepattern.*'
        other_optional_props ;
```

COPY Command

Feature	Supported	Notes
Location of files	Local environment	Files are first staged in a Snowflake stage, then loaded into a table.
	Amazon S3	Files can be loaded directly from any user-supplied S3 bucket.
	Google Cloud Storage	Files can be loaded directly from any user-supplied Cloud Storage container.
	Microsoft Azure	Files can be loaded directly from any user-supplied Azure container.
File formats	Delimited files (CSV, TSV, etc.)	Any valid delimiter is supported; default is comma (i.e. CSV).
	JSON	
	Avro	Includes automatic detection and processing of staged Avro files that were compressed using Snappy.
	ORC	Includes automatic detection and processing of staged ORC files that were compressed using Snappy or zlib.
	Parquet	Includes automatic detection and processing of staged Parquet files that were compressed using Snappy.
	XML	Supported as a preview feature.

Other ways to load data to snowflake tables:

Other ways to load Data

- By using ETL tools like
  - Matellion
  - Datastage
  - Informatica
  - Hevo
  - Azure Data Factory
  - Azure Synapse etc

Simple Transformations During Data Load

Snowflake supports transforming data while loading it into a table using the COPY command. Options include:

- Column reordering
- Column omission
- String operations
- Other functions
- Sequence numbers
- Auto increment fields

Copying the data from data lake to snowflake table:

```
//Loading the data from S3 bucket
COPY INTO PUBLIC.LOAN_PAYMENT
    FROM s3://bucketsnowflakes3/Loan_payments_data.csv
    file_format = (type = csv , field_delimiter = ',' , skip_header=1);
```

To keep all the stage objects in one schema we need to create a schema like external schema as shown below:

```
// Create a Schema for External Stages

CREATE OR REPLACE SCHEMA MYDB.external_stages;

// Publicly accessible staging area
CREATE OR REPLACE STAGE MYDB.external_stages.aws_ext_stage
    url='s3://bucketsnowflakes3';

// Listing the files in external stage
list @MYDB.external_stages.aws_ext_stage;

//Case 1: Just Viewing Data from ext stage
select $1, $2, $3, $4, $5, $6 from @MYDB.external_stages.aws_ext_stage/OrderDetails.csv;
```

If we want to specifically load the orders.csv file then we have to mention in the below way:

```
COPY INTO MYDB.PUBLIC.ORDERS_EX
    FROM (select s.$1, s.$2 from @MYDB.external_stages.aws_ext_stage s)
    file_format= (type = csv field_delimiter=',' skip_header=1)
    files=('OrderDetails.csv');
```

```
//Loading the data from S3 bucket
COPY INTO PUBLIC.LOAN_PAYMENT2("Loan_ID", "loan_status", "Principal", "terms", "effective_date", "due_date",
"paid_off_time", "past_due_days", "age", "education", "Gender")
    FROM s3://bucketsnowflakes3/Loan_payments_data.csv
    file_format = (type = csv  field_delimiter = ','  skip_header=1);
```

To generate the sequence number in the snowflake table, below is the way:

```
CREATE OR REPLACE TABLE MYDB.PUBLIC.LOAN_PAYMENT (
  "SEQ_ID" number default seq1.nextval,
```

Agenda

- What is a Stage?
- Stage Types
- External Stages
- Internal Stages
- Data load from Stages
- A stage specifies where data files are stored (i.e. "staged") so that the data in the files can be loaded into a table.
- Stage is a location of files, that can be internal or external to snowflake

Stage is nothing but the location of files. That can be internal or external to snowflake.

Stage Types:

1. External Stages
2. Internal Stages
    1. User Internal Stage
    2. Table Internal Stage
    3. Named Internal Stage

Stage object is a db object created in schema.

```
// Create Database and schema.

CREATE DATABASE IF NOT EXISTS MYDB;
CREATE SCHEMA IF NOT EXISTS external_stages;

// Publicly accessible staging area
CREATE OR REPLACE STAGE MYDB.external_stages.sample_stage
    url='s3://bucketsnowflakes3';

// Description of external stage
DESC STAGE MYDB.external_stages.sample_stage;

// List files in stage
LIST @external_stages.sample_stage;
```

To check the properties of the stage object:

```
// Description of external stage
DESC STAGE MYDB.external_stages.sample_stage;
```

To load the data from data lake s3 bucket storage to snowflake tables:

```
COPY INTO MYDB.PUBLIC.ORDERS
    FROM @MYDB.external_stages.sample_stage
    file_format= (type = csv field_delimiter=',' skip_header=1)
    files = ('OrderDetails.csv');
```

If we don't mention any file properties, then by default its .csv file.

```
// Creating schema to keep file formats
CREATE SCHEMA IF NOT EXISTS MYDB.file_formats;

// Creating file format object
CREATE file format MYDB.file_formats.csv_file_format;

// See properties of file format object
DESC file format MYDB.file_formats.scv_file_format;
```

To alter file format object:

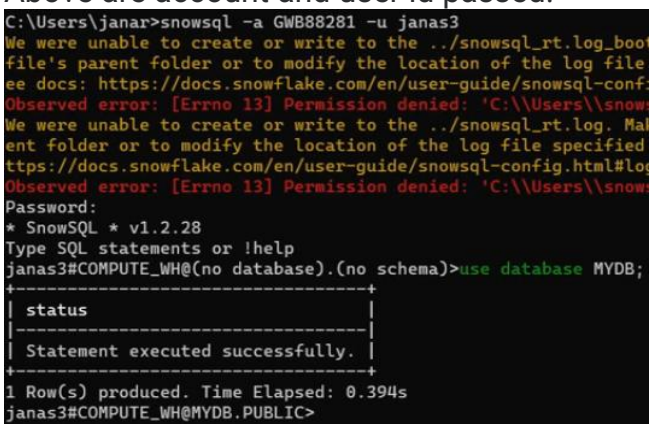
```
// Altering file format object
ALTER file format MYDB.file_formats.scv_file_format
    SET $KIP_HEADER = 1;
```

Snow SQL:

Below is the connection string to give to connect to snowsql:

```
snowsql -a GWB88281 -u janas3
```

Above are account and user id passed.



Internal stages and SnowSQL:

3 Types of Internal Stages:

1. User Stage
2. Table Stage
3. Named Internal Stage

User stage is used to store files in staging area allocated to that user.
This happens when user is created some space is allocated to user.

User Stage (@~)

- A user stage is allocated to each user for storing files.
- To store files that are staged and managed by a single user but can be loaded into multiple tables.
- User stages cannot be altered or dropped.
- This option is not appropriate if multiple users require access to the files.

Table Stage (@%)

- A table stage is available for each table created in Snowflake.
- Table stages have the same name as the table.

  e.g. a table named mytable has a stage referenced as @%mytable
- To store files that are staged and managed by one or more users but only loaded into a single table.
- Table stages cannot be altered or dropped.
- Note that a table stage is not a separate database object, rather it is an implicit stage tied to the table itself.

The particular staging area is allocated to one table. One or more users can load to this table.

Named Internal Stage (@)

- A named internal stage is a database object created in a schema
- To store files that are staged and managed by one or more users and loaded into one or more tables.
- Create stages using the CREATE STAGE command (similar to external stages)

```
CREATE OR REPLACE STAGE MYDB.internal_stages.sample_stage
```

Named stage is a db object.

Staging Data Files from a Local File System

Linux or macOS:

```
put file:///data/data.csv @~/staged;
put file:///data/data.csv @%mytable;
put file:///data/data.csv @my_stage;
```

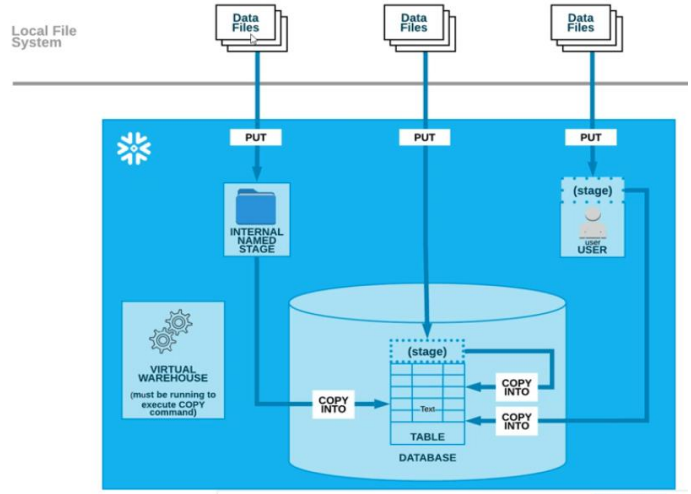
Windows:

```
put file://c:\data\data.csv @~/staged;
put file://c:\data\data.csv @%mytable;
put file://c:\data\data.csv @my_stage;
```

Listing Files:

```
list @~;
list @%mytable;
list @my_stage;
```

Difference between Copy & Put commands in snowflake:
Put → This is used to copy the files from local desktop or local server to snowflake internal staging area.
Copy→ to load the data from external stages or internal stages to snowflake tables.



Copying data from Internal stage

User Stage:

To Load all files prefixed with staged in your user stage,

copy into mytable from @~/staged
file_format = (type = csv field_delimiter = '|' skip_header = 1);

Table Stage:

To load all files in the stage for the customer table,

copy into mytable from @%customer
file_format = (format_name = my_csv_format);

Copying data from Internal stage

Named Stage:

To load all files from the my_stage named stage,

copy into mytable from @my_stage;

Note that a file format does not need to be specified because it is included in the stage definition.

We can mention the file format while creating the stage itself.
Below way to copy the files from desktop to User staging area:

```
// USER STAGE
//Put your files into user internal stage
//If your put command is failing with 403 forbidden error, practice this session after
AWS-Snowflake integration session(next video in the play list) then it will work.

put file://C:\Users\janar\OneDrive\Documents\Files\pets_data.json @~/staged;
```

Executed the put command to userstage using snowsql:

```
janas2#SAMPLE_WH@MYDB.PUBLIC>put file://C:\Users\janar\OneDrive\Documents\Files\customer_data_table.csv @%customer_data_table;
+---------------------+---------------------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source              | target                    | source_size | target_size | source_compression | target_compression | status   | message |
|---------------------+---------------------------+-------------+-------------+--------------------+--------------------+----------+---------|
| customer_data_table.csv | customer_data_table.csv.gz |        2752 |        1552 | NONE               | GZIP               | UPLOADED |         |
+---------------------+---------------------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 2.330s
janas2#SAMPLE_WH@MYDB.PUBLIC>list @%customer_data_table;
+-------------------------+------+----------------------------------+-------------------------------+
| name                    | size | md5                              | last_modified                 |
|-------------------------+------+----------------------------------+-------------------------------|
| customer_data_table.csv.gz | 1552 | 50aadc31125c3546fc5200d9511fbfd4 | Sat, 3 Dec 2022 12:29:10 GMT |
+-------------------------+------+----------------------------------+-------------------------------+
1 Row(s) produced. Time Elapsed: 0.340s
janas2#SAMPLE_WH@MYDB.PUBLIC>
```

Named staging area is a db object. we have to create it in the snowflake db.

```
janas2#SAMPLE_WH@MYDB.PUBLIC>CREATE SCHEMA IF NOT EXISTS mydb.internal_stages
                            ;
+----------------------------------------------------+
| status                                             |
|----------------------------------------------------|
| INTERNAL_STAGES already exists, statement succeeded. |
+----------------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.299s
janas2#SAMPLE_WH@MYDB.PUBLIC>CREATE OR REPLACE STAGE mydb.internal_stages.named_customer_stage;
+----------------------------------------------------+
| status                                             |
|----------------------------------------------------|
| Stage area NAMED_CUSTOMER_STAGE successfully created. |
+----------------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.359s
```

```
//Create a named stage
CREATE OR REPLACE STAGE mydb.internal_stages.named_customer_stage;

CREATE OR REPLACE STAGE mydb.internal_stages.named_orders_stage;

CREATE OR REPLACE STAGE mydb.internal_stages.named_product_stage;
```

If we want to check all the internal stages which are created above:

Copying the data from different stages to snowflake table:

```
COPY INTO mydb.public.customer_data_table
FROM @~/staged/customer_data_user.csv
file_format = (type = csv field_delimiter = '|' skip_header = 1);

COPY INTO mydb.public.customer_data_table
FROM @%customer_data_table/customer_data_table.csv
file_format = (type = csv field_delimiter = '|' skip_header = 1);

COPY INTO mydb.public.customer_data_table
FROM @mydb.internal_stages.named_customer_stage/customer_data_named.csv
file_format = (type = csv field_delimiter = '|' skip_header = 1);
```

Copy command Options:

Copy Command Syntax

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
CopyOptions
```

The different copy options available are:

- VALIDATION_MODE
- RETURN_FAILED_ONLY
- ON_ERROR
- FORCE
- SIZE_LIMIT
- TRUNCATECOLUMNS
- ENFORCE_LENGTH
- PURGE
- LOAD_UNCERTAIN_FILES

In detail about each Copy options:

Return_error:

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
VALIDATION_MODE = RETURN_n_ROWS|RETURN_ERRORS|RETURN_ALL_ERRORS;
```

→ Validate the data files instead of loading them into the table
→ RETURN_ERRORS gives all errors in the files
→ RETURN_ALL_ERRORS gives all errors from previously loaded files if we have used ON_ERROR = CONTINUE
→ RETURN_N_ROWS displays first N records and fails at the first error record

On_error:

ON_ERROR

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
ON_ERROR = CONTINUE | SKIP_FILE | SKIP_FILE_num |
           SKIP_FILE_num% | ABORT_STATEMENT
```

→ CONTINUE – To skip error records and load remaining records
→ SKIP_FILE – To skip the files that contain errors
→ SKIP_FILE_Num – Skip a file when the number of error rows found in the file is equal to or exceeds the specified number.
→ SKIP_FILE_Num% – Skip a file when the percentage of error rows found in the file exceeds the specified percentage.
→ Default is ABORT_STATEMENT, Abort the load operation if any error is found in a data file.

Force property:

FORCE

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
FORCE = TRUE | FALSE ;
```

→ To load all the files, regardless of whether they've been loaded previously
→ Default is False, if we don't specify this property Copy command will not fail but it skips loading the data

Size Limit:

SIZE_LIMIT

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
SIZE_LIMIT = <NUMBER> ;
```

→ Specify maximum size in bytes of data loaded in that command
→ When the threshold is exceeded, the COPY operation stops loading
→ In case of multiple files of same pattern also it will stop after the size limit, that means some files can be loaded fully and one file will be loaded partially

Truncate columns or enforce length:

TRUNCATE_COLUMNS or ENFORCE_LENGTH

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
TRUNCATECOLUMNS = TRUE | FALSE - Default is False ;
(Or) ENFORCE_LENGTH = TRUE | FALSE - Default is TRUE ;
```

→ Specifies whether to truncate text strings that exceeds the target column length

→ Default is FALSE, that means if we don't specify this option, and if text strings that exceeds the target column length, then Copy command will fail

Purge:

PURGE

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
PURGE = TRUE | FALSE
```

→ Specifies whether to remove the data files from the stage automatically after the data is loaded successfully.

→ Default is FALSE

Load Uncertain files:

LOAD UNCERTAIN FILES

```
COPY INTO <table_name>
FROM  @ExternalStage
FILES = ( '<file_name>','<file_name2>')
FILE_FORMAT = <file_format_name>
LOAD_UNCERTAIN_FILES = TRUE | FALSE
```

→specifies to load files for which the load status is unknown. The COPY command skips these files by default.

Note: The load status is unknown if ***all*** of the following conditions are true:

- The file's LAST_MODIFIED date is older than 64 days.
- The initial set of data was loaded into the table more than 64 days earlier.

If we want to list down the files in a stage→ list <stage name>

On_error, truncate_columns, validation_mode are the important properties/commands in the real time scenarios.

Snowflake-Azure Integration

Steps to Load Data from Azure

1. You should have Snowflake trial account
2. You should have Azure trial account
3. Create storage account and containers in Azure
4. Upload the source files to these containers
5. Create storage integration between Snowflake and Azure
6. Create stage objects using the storage integration object
7. Use copy commands to extract the data from files and load in snowflake tables.

For storage integration object we need to go to IAM(Access Control) and provide access to necessary snowflake related components.

Processing Semi structured data

Processing Json data:

If we have multiple elements in a list then e call it as an array. Json file can contain array of elements.

How to extract the data from json file and load into snowflake tables step by step:

To copy the JSON data into Stage Table:

```
--Copy the RAW data into a Stage Table
COPY INTO MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW
    FROM @MYOWN_DB.external_stages.STAGE_JSON
    file_format= MYOWN_DB.external_stages.FILE_FORMAT_JSON
    FILES=('pets_data.json');
```

How can we store avro or parquet or json file into snowflake

Or while processing the semi strucred data how can you store the entire file into snowflake → we can create a table with variant data type and we can load the entire file into that variant field.

Later by using the parsing method we can extract the data from the variant type and load it into rows and columns.

If we want to fetch the array data from that json file we may need to give the indexing position as below:

```
--Extracting Array data
SELECT raw_file:Name::string as Name,
       raw_file:Pets[0]::string as Pet
FROM MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW;
```

This case we will fetch only one lement from the array.

Incase if we want to fetch all the elements from the array:

TO get the array size:

```
SELECT MAX(ARRAY_SIZE(RAW_FILE:Pets)) as PETS_AR_SIZE
FROM MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW;
```

To find out how many no of elements in each row:

```
--Get the size of ARRAY
SELECT raw_file:Name::string as Name, ARRAY_SIZE(RAW_FILE:Pets) as PETS_AR_SIZE
FROM MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW;
```

Nested data in the JSON file:

```
{
    "Name": "Ravi",
    "Gender": "Male",
    "DOB": "1990-03-21",
    "Pets": ["Dog", "Cat"],
    "Address": {
        "House Number": "123/4",
        "City": "Hyderabad",
        "State": "Telangana"
    },
    "Phone": {
        "Work": 123456789,
        "Mobile": 987654321
    }
},
```

When we want to fetch the address part from the above:

```
--Extracting nested data
SELECT raw_file:Name::string as Name,
       raw_file:Address."House Number"::string as House_No,
       raw_file:Address.City::string as City,
       raw_file:Address.State::string as State
FROM MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW;
```

The output will be as shown below:

Row	NAME	HOUSE_NO	CITY	STATE
1	Ravi	123/4	Hyderabad	Telangana
2	Latha	567/8	Bangalore	Karnataka

Below is the way to extract entire data in single sql statement using union all:

```
--Parsing entire file
SELECT raw_file:Name::string as Name,
       raw_file:Gender::string as Gender,
       raw_file:DOB::date as DOB,
       raw_file:Pets[0]::string as Pets,
       raw_file:Address."House Number"::string as House_No,
       raw_file:Address.City::string as City,
       raw_file:Address.State::string as State,
       raw_file:Phone.Work::number as Work_Phone,
       raw_file:Phone.Mobile::number as Mobile_Phone
from MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW
UNION ALL
SELECT raw_file:Name::string as Name,
       raw_file:Gender::string as Gender,
       raw_file:DOB::date as DOB,
       raw_file:Pets[1]::string as Pets,
```

To avoid the duplicates, we will have to give in the where condition as below:

```
from MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW
WHERE Pets is not null;
```

By suing an approach called flatten approach, we can avoid eliminate writing multiple union all statements.

```
INSERT INTO MYOWN_DB.INTG_TBLS.PETS_DATA
select distinct
       raw_file:Name::string as Name,
       raw_file:Gender::string as Gender,
       raw_file:DOB::date as DOB,
       f1.value::string as Pet,
       raw_file:Address."House Number"::string as House_No,
       raw_file:Address.City::string as City,
       raw_file:Address.State::string as State,
       raw_file:Phone.Work::number as Work_Phone,
       raw_file:Phone.Mobile::number as Mobile_Phone
FROM MYOWN_DB.STAGE_TBLS.PETS_DATA_JSON_RAW,
table(flatten(raw_file:Pets)) f1;
```

When ever we do flattening we have to put .value for it. As shown above for pets field as f.value as we flattened pets field.

When we are flattening . it means that we are converting rows into columns.

Processing XML files:
Below is the sample xml file shown which contains the book details:

```xml
<?xml version="1.0"?>
<catalog>
   <book> <id>bk101</id>
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book> <id>bk102</id>
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book> <id>bk103</id>
```

If we don't give the file format in the below by default it would take xml format:

```
// Create file format object for xml files
CREATE OR REPLACE file format my_db.file_formats.xml_fileformat
    type = xml;

// Create variant table to load xml file
CREATE OR REPLACE TABLE my_db.stage_tbls.STG_BOOKS(xml_data variant);

// Load xml file to variant table
copy into my_db.stage_tbls.STG_BOOKS
from @my_db.external_stages.aws_s3_xml
files=('books_20230304.xml');

// Query stage table
select * from my_db.stage_tbls.STG_BOOKS;
```

```
// Get the content using xmlget function, index position
select xmlget(xml_data,'book',0):"$" from my_db.stage_tbls.STG_BOOKS;
select xmlget(xml_data,'book',1):"$" from my_db.stage_tbls.STG_BOOKS;

// Fetch actual data from file
SELECT
XMLGET(bk.value, 'id' ):"$" as "book_id",
XMLGET(bk.value, 'author' ):"$" as "author"
FROM my_db.stage_tbls.STG_BOOKS,
LATERAL FLATTEN(to_array(STG_BOOKS.xml_data:"$" )) bk;
```

```
// Fetch data and assign datatypes
SELECT
XMLGET(bk.value, 'id' ):"$" :: varchar as "book_id",
XMLGET(bk.value, 'author' ):"$" :: varchar as "author",
XMLGET(bk.value, 'title' ):"$" :: varchar as "title",
XMLGET(bk.value, 'genre' ):"$" :: varchar as "genre",
XMLGET(bk.value, 'price' ):"$" :: number(10,2) as "price",
XMLGET(bk.value, 'publish_date' ):"$" :: date as "publish_date",
XMLGET(bk.value, 'description' ):"$" :: varchar as "description"

FROM my_db.stage_tbls.STG_BOOKS,
LATERAL FLATTEN(to_array(STG_BOOKS.xml_data:"$" )) bk;
```

Snowflake Pricing: Cost calculation in Snowflake

Snowflake cost is dependent on

1. Snowflake Edition
   - Standard - $2.7/Credit
   - Enterprise - $4/Credit
   - Business Critical - $5.4/Credit
   - VPS – Depends on Org
2. Region where Snowflake account created
3. Cloud platform where Snowflake account hosted
4. Virtual Warehouse Size

Types of Cost:

1. Storage Cost
2. Compute Cost
3. Cloud Services Cost

Storage Cost:

- The charge for Storage is per terabyte after compressed, per month.
- 2 Storage Plans available in Snowflake.

1. On Demand Storage:
   - Most flexible and easiest model
   - Pay-as you Use / Pay-as you go
   - Customer are charged FIXED rate of $40/TB Per month (compressed)
2. Capacity Storage (or) Fixed Storage:
   - Option to Pre-purchase capacity
   - A Capacity purchase is a specific dollar commitment to snowflake
   - Snowflake charges $23 per month/TB (compressed)

How to choose storage type:

- When you are not sure about your data size, start with On Demand
- Once you are sure switch to Capacity Storage

After we analyze the data size we can switch to storage type to capacity size.

Compute Cost:

- Compute cost is calculated based on the Virtual Warehouse usage per month
- Compute cost is calculated in Snowflake Credits
- Billed per seconds with 1 minute minimum
- The different sizes of virtual warehouses consume snowflake credits at the following rates.
- If we choose Large size and if we used it for 30 minutes then it will be billed as 4 Credits

Size	X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large (in preview)	6X-Large (in preview)
Credit Usage per Hour	1	2	4	8	16	32	64	128	256	512

Snowflake Credit:

It's a unit of measure. The cost is calculates using the measure - credits.

- A snowflake credit is a unit of measure
- Snowflake credits are used to pay for the consumption of the resources on Snowflake
- It is consumed only when a customer is using resources, such as when using virtual warehouses.
- Users receives $400 worth of free usage upon creation of Snowflake free trial account

Server less features:

Serverless features use snowflake-managed compute resources and consume snowflake credits when they are used.
We can see the credit usage on the bill.
Below are the serverless features.

Functional Area	Feature	Usage
Tables	Automatic Clustering	Automated background maintenance of each clustered table, including initial clustering and reclustering as needed.
	Search Optimization Service	Automated background maintenance of the search access paths used by the search optimzation service.
External tables	Automatically Refreshing External Table Metadata	Automated refreshing of the external table metadata with the latest set of associated files in the external stage and path.
Views	Materialized views maintenance	Automated background synchronization of each materialized view with changes in the base table for the view.
Data loading	Snowpipe	Automated processing of file loading requests for each pipe object.
Database replication	Database Replication and Failover/Failback	Automated copying of data between accounts, including initial data replication and maintenance as needed.
Tasks	Executing SQL Statements on a Schedule Using Tasks	Snowflake-managed compute resources provided to execute SQL code, rather than a user-managed virtual warehouse.
Warehouses	Using the Query Acceleration Service	Snowflake-managed compute resources provided to execute portions of eligible queries. [1]

Types of Cost:

Storage Cost

Compute Cost

Cloud Services Cost

The storage cost depends on:

- Cost Depends On?
- Types of Costs
- Storage Types
- Compute Cost
- Snowflake Credit
- Serverless features

Snowpipe

- What is continuous loading?
- Snowpipe
- Steps in Creating Snowpipe
- Snowpipe syntax
- Snowpipe DDL
- Troubleshooting Snowpipe
- Managing Snowpipe

Continuous Loading:

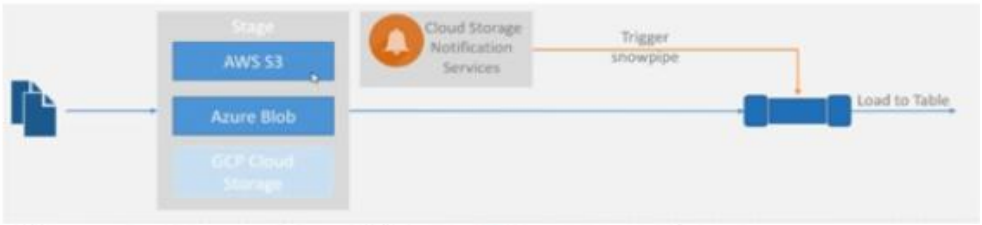
- Loading small volumes of data in continuous manner like for every hour or for every 10 minutes etc.
- Live or real time data.
- This ensures users have the latest data for business analysis.
- Snowflake uses snowpipes for continuous data loads into snowflake tables.

Snowpipe is a named DB object that contains copy command used to load the data.

- Snowpipe loads data within minutes after files are added to a stage and submitted for ingestion.
- Snowpipe uses compute resources provided by Snowflake, It is a serverless task.
- One time setup.
- Suggested micro file size is 100-250 MB.
- Snowpipe uses file loading metadata associated with each pipe object to prevent reloading the same files.

It's a serverless setup. As soon as the files are placed in aws s3 bucket or adls, they are immediately loaded into snowflake tables. For that we have some configuration settings needs to be done.

How Snowpipe works?



Steps in creating Snowpipe

1. Create Storage Integration object
2. Create a Stage object using storage integration object.
3. Create and test Copy command to load the data.
4. Create a Pipe by using the Copy command.
5. Setup event notifications at Cloud storage provider's end.

Snowpipe Syntax

```
CREATE OR REPLACE   PIPE   PIPE_NAME
    AUTO_INGEST = [ TRUE | FALSE ] ]
    AS
    <Copy_Statement>
```

Snowpipe DDL

- CREATE PIPE → To create a pipe
- ALTER PIPE → To alter pipe or to pause or resume pipe

  ALTER PIPE *PIPE_NAME* PIPE_EXECUTION_PAUSED = TRUE | FALSE
- DROP PIPE → To drop the pipe
- DESCRIBE PIPE → To describe the pipe properties, to get ARN
- SHOW PIPES → To see all the pipes

```
// Create a file format object of csv type
CREATE OR REPLACE file format mydb.file_formats.csv_fileformat
    type = csv
    field_delimiter = ','
    skip_header = 1
    empty_field_as_null = TRUE;

// Create a stage object using storage integration
CREATE OR REPLACE stage mydb.external_stages.stage_aws_pipes
    URL = 's3://awss3bucketjana/pipes/csv/'
    STORAGE_INTEGRATION = s3_int
    FILE_FORMAT = mydb.file_formats.csv_fileformat;

// List the files in Stage
LIST @mydb.external_stages.stage_aws_pipes;

// Create a table to load these files
CREATE OR REPLACE TABLE mydb.public.emp_data
(
  id INT,
  first_name STRING,
  last_name STRING,
  email STRING,
  location STRING,
  department STRING
);

// Create a schema to keep pipe objects
CREATE SCHEMA IF NOT EXISTS mydb.pipes;

// Create a pipe
CREATE OR REPLACE pipe mydb.pipes.employee_pipe
AUTO_INGEST = TRUE
AS
COPY INTO mydb.public.emp_data
FROM @mydb.external_stages.stage_aws_pipes
pattern = '.*employee.*';
```

How to trouble shoot issues in snow pipe:

Step 1: Check the Pipe Status

Step 2: View the COPY History for the Table

Step 3: Validate the Data Files

We have to check whether the pipe is up and running using the below:

```
SELECT SYSTEM$PIPE_STATUS('pipe_name');
```

Step 2 : we can check for the copy history:

→ Copy history shows the history of all file loads and errors if any.
→ View the copy history by using below query.

```
SELECT * FROM TABLE( INFORMATION_SCHEMA.COPY_HISTORY
      (table_name => 'table_name',
       START_TIME => 'timestamp or expression'))
);
```

Incase if we want to check the copy history of 10 hrs back then below is the way:

```
// Step2: Check the Copy History
SELECT * FROM TABLE( INFORMATION_SCHEMA.COPY_HISTORY
    (TABLE_NAME  =>  'myown_db.public.emp_data',
     START_TIME => DATEADD(HOUR, -10 ,CURRENT_TIMESTAMP()))
);
```

From the above we can understand that there is a zero count loaded into the table from the file for the last file since it shows row_count as zero.

```
// Step2: Check the Copy History
SELECT * FROM TABLE( INFORMATION_SCHEMA.COPY_HISTORY
    (TABLE_NAME  =>  'myown_db.public.emp_data',
     START_TIME => DATEADD(HOUR, -24 ,CURRENT_TIMESTAMP()))
);
```

Results | Data Preview | Open History

Query ID | SQL | 777ms | 3 rows

Filter result... | Copy | Columns

Row	FILE_NAME	STAGE_LOCATION	LAST_LOAD_TIM	ROW_COUNT	ROW_PARSED	FILE_SIZE	FIRST_ERROR_MESSAGE	FIRST_ERROR_LI
1	sp_employee_2.csv	s3://awss3bucketjana/pipes/csv/	2022-07-06 ...	100	100	6398	NULL	NULL
2	sp_employee_1.csv	s3://awss3bucketjana/pipes/csv/	2022-07-06 ...	100	100	6524	NULL	NULL
3	sp_employee_3.csv	s3://awss3bucketjana/pipes/csv/	2022-07-06 ...	0	100	6459	Number of columns in...	3

The error it has shown for the last file is below:

Details

```
1  Number of columns in file (1) does not match that of the corres
   table (6), use file format option error_on_column_count_mismatc
   to ignore this error
```

Copy

Validating the source files: This is not with us as its completely from source team.

→ If the load operation encounters errors in the data files, the COPY_HISTORY table function describes the first error encountered in each file.

→ To validate the data files, query the VALIDATE_PIPE_LOAD table.

```
SELECT * FROM TABLE(INFORMATION_SCHEMA.VALIDATE_PIPE_LOAD
      (PIPE_NAME => 'pipe_name',
   START_TIME => 'timestamp or expression'
));
```

In some cases there will be an issue with the file format object. In that case we can manually run the copy command that is used in the pipe as shown below by specifying the particular file name:

```
// Load the history files by running Copy command
COPY INTO myown_db.public.emp_data
FROM @myown_db.external_stages.stage_aws_pipes
FILES = ('sp_employee_3.csv');
```

We have to load the history files by running the copy command manually.
Incase if the delimiter was changed from | to , then below step need to be xecuted manually:

```
// Correct the delimiter to ','
CREATE OR REPLACE file format myown_db.file_formats.csv_fileforma
    type = csv
    field_delimiter = ','
    skip_header = 1
    empty_field_as_null = TRUE;
```

How can we manage the pipes:

Managing Pipes

- → Use DESC pipe_name command to see the pipe properties and the copy command
- → Use SHOW pipes command to see all the pipes
- → We can pause/resume pipes with PIPE_EXECUTION_PAUSED = true/false
- → It is a best practice to pause and resume pipes before and after performing below actions
  - → when modifying the stage object
  - → when modifying file format object if stage is using
  - → when modifying copy command
  - → when recreating the pipes (replace)
- → To modify the copy command, recreating pipe is the only possible way
- → When you recreate a pipe, all the load history will be dropped.

If we want to see the pipes listed in the DB, below are some of the ways:

```
// Validate the data in the table
SELECT * FROM myown_db.public.emp_data;

DESC PIPE employee_pipe;
SHOW PIPES;
SHOW PIPES like '%employee%';
SHOW PIPES in database myown_db;
```

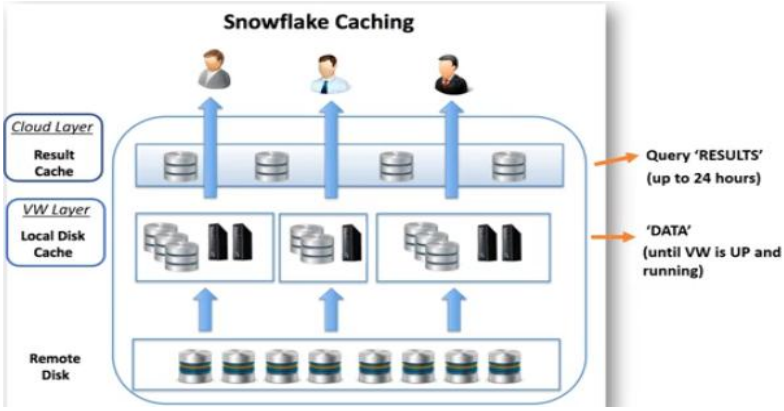
Caching in Snowflake:

Cache is a temporary storage location that stores files/copies of data so that they can be accessed faster in near future. It plays a vital role in saving costs and speeding up the results and improves query performance.

There are 2 types of cache → Query results cache & local disk cache

If we want to use those files in next 24 hrs or next 2 days which are stored in cache, we can access them in faster and easier way.

Architecture of snowflake: Results cache is located in the cloud services layer. Local disk cache is located in the virtual warehouse layer. When a query is executed first it checks in the result cache to give the output faster.If the desired data is not available in the query result cache, then it looks for the local disk cache and bring up the data from local disk cache.



In this case the data is available in the results cache when the VWH is up and running. In between when the VWH is suspended then the local disk cache is cleared.

Query results are available in cache for 24 hrs.

Results cache can be available across different virtual warehouses.

Results cache works as long as there is no change done for the underlying data.

Query results returned to one user is available to another user on the system who executed the same query.

The query should be the same without any changes in the underlying data. Not even the re-ordering of columns or subset of data.

It works when we query for subset of data that is available in local disk cache.

The **local disk cache** depends on the size of the virtual warehouse we are using.
For ex: X small VW cant hold Millions of records , but can fetch part of the data from local disk cache and remaining from Remote Disk.
As we know metadata management is done by cloud services layer in snowflake.

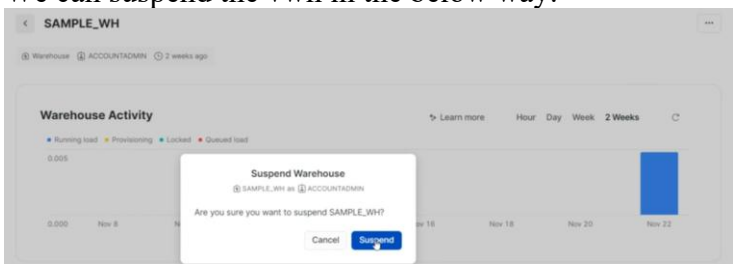
```
// Query is fetching results from Storage Layer(Remote Disk)
SELECT * FROM  TPCH_SF1000.CUSTOMER; -- 2min 20sec

// Fetching METADATA info is very fast, look at query profile
SELECT COUNT(*) FROM  TPCH_SF1000.CUSTOMER; -- 242 ms
SELECT MIN(C_CUSTKEY) FROM  TPCH_SF1000.CUSTOMER; -- 100 ms
SELECT MAX(C_CUSTKEY) FROM  TPCH_SF1000.CUSTOMER; -- 57 ms

// Run the same query again and observe time taken and query profile
SELECT * FROM  TPCH_SF1000.CUSTOMER; -- 241 ms
```

The difference in the time taken by re-ordering the columns but querying the same data:

```
// Fetching METADATA info is very fast, look at query profile
SELECT COUNT(*) FROM  TPCH_SF1000.CUSTOMER; -- 70ms
SELECT MIN(C_CUSTKEY) FROM  TPCH_SF1000.CUSTOMER; -- 68 ms
SELECT MAX(C_CUSTKEY) FROM  TPCH_SF1000.CUSTOMER; -- 64 ms

// Run the same query again and observe time taken and query profile
SELECT * FROM  TPCH_SF1000.CUSTOMER; -- 113 ms

// Try to fetch same data by changing queries little bit and observe query profile
SELECT C_CUSTKEY, C_NAME, C_ACCTBAL, C_ADDRESS FROM TPCH_SF1000.CUSTOMER; -- 53 sec
SELECT C_CUSTKEY, C_ADDRESS FROM TPCH_SF1000.CUSTOMER; -- 36 sec
SELECT C_ADDRESS, C_CUSTKEY FROM TPCH_SF1000.CUSTOMER; -- 32 sec
```

To disable the result cache→ ALTER SESSION SET USE_CACHED_RESULT=FALSE
We can set the auto suspend option for certain time for the vwh.

We can suspend the vwh in the below way:



Time Travel & Fail Safe Mechanisms

Agenda

- Time Travel
- Retention Period
- Querying historical data
- Restoring Objects
- Fail Safe
- Continuous Data Protection Life Cycle

There is no need to enable the time travel, its automatically enabled by default.

Retention Period

- Retention is the key component of Time Travel.
- It specifies the number of days for which this historical data is preserved.
- For Standard Edition, the retention period is 1 day, can set it to 0.
- For Enterprise and higher editions it is 90 days, can set it any thing between 0-90 days.
- Default is 1, we can set this period at the time of creation of object or can alter later.
- A retention period of 0 days for any object means disabling Time Travel for the object.
- We can change the retention period by using ALTER command.
- Higher retention period, the higher storage cost.

Higher the retention period higher will be the cost of storage.
Retention period can be set at the schema level or table level or db level or warehouse level.
To query historical data:

1. At specified timestamp
2. At some time ago
3. Before executing any statement/query

Querying Historical Data

1. Below query selects historical data from a table at the specified timestamp:

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

2. Below query selects historical data from a table as of 5 minutes ago:

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

3. Below query selects historical data that present in the table before executing the query specified by query id

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

How to restore objects:

Restoring Objects

- When a table, schema, or database is dropped, it is not immediately removed from the system.
- Instead, it is retained for the data retention period for the object, during which time the object can be restored.
- After retention period is completed, we can't restore the objects.

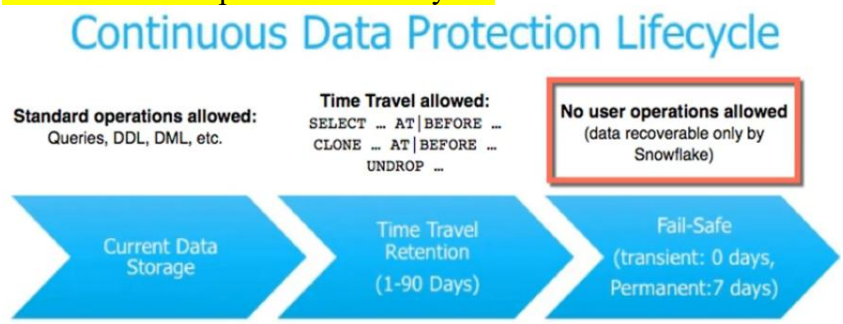
```
UNDROP TABLE 'TABLE_NAME'
UNDROP SCHEMA 'SCHEMA_NAME'
UNDROP DATABASE 'DATABASE_NAME'
```

With time travel we can retain the data for about 90 days after that fail safe period comes into picture when the time travel period expires. But the data can be retrieved from fail safe only through snowflake support.

Fail-safe

- Fail-safe provides a 7-day period during which historical data may be recoverable by Snowflake.
- This period starts immediately after the Time Travel retention period ends.
- We can't query or restore the Fail-safe data.
- We need to contact Snowflake Support to restore the data during Fail-safe period.
- Data recovery through Fail-safe may take several hours to several days to complete.
- After Fail-safe period is over, there is no other way to recover the data.

Continuous data protection Life cycle:



Hands on:

```
CREATE OR REPLACE TABLE myown_db.public.timetravel_ex(id number, name string);

SHOW TABLES like 'timetravel_ex%';

CREATE OR REPLACE TABLE myown_db.public.timetravel_ex(id number, name string)
DATA_RETENTION_TIME_IN_DAYS = 10;
```

It shows the 10 days retention period:

We can change the data retention period in the below way:

```
// how to alter retention period later?
ALTER TABLE myown_db.public.timetravel_ex
SET DATA_RETENTION_TIME_IN_DAYS = 15;

// Case1: retrieve history data by using AT OFFSET

SELECT * FROM myown_db.public.customer WHERE CUSTOMERID=1682100334099;

SELECT * FROM myown_db.public.customer AT (offset => -60*2)
WHERE CUSTOMERID=1682100334099;

// Case2: retrieve history data by using AT TIMESTAMP
SELECT * FROM myown_db.public.emp_data AT(timestamp => '2022-07-08 07:30:47.145'::timestamp)
WHERE id=1;
```

Checking the above table 25 hrs back what is the data:

```
SELECT * FROM myown_db.public.customer AT (offset => -60*60*25)
WHERE CUSTOMERID=1682100334099;
```

If data is not available during that retention period or if its beyond the time it has been created then below is the error its thrown:

```
Time travel data is not available for table CUSTOMER. The requested time is either beyond the allowed time travel period or before the object creation time.
```

If it is at certain point of time then we use at timestamp. If its before certain period of time then we use offset as mentioned below:

```
// Case1: retrieve history data by using AT OFFSET

SELECT * FROM myown_db.public.customer WHERE CUSTOMERID=1682100334099;

SELECT * FROM myown_db.public.customer AT (offset => -60*5)
WHERE CUSTOMERID=1682100334099;

// Case2: retrieve history data by using AT TIMESTAMP
SELECT * FROM myown_db.public.emp_data AT(timestamp => '2022-07-08 07:30:47.145'::timestamp)
WHERE id=1;
```

For semi structured data, when ever we are casting the column from one datatype to another datatype then we use double colon same as in last query in above ss.
The below is another kind ofd retrieval of data apart from offset and timestamp i.e. before

```
// Case3: retrieve history data by using BEFORE STATEMENT
SELECT * FROM myown_db.public.orders WHERE ORDER_ID='B-25601';

SELECT * FROM myown_db.public.orders
before(statement => '01a57b69-0004-25d4-0015-ab8700024536')
WHERE ORDER_ID='B-25601';
```

If we want to check the storage metrics we can see the below:

```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.TABLE_STORAGE_METRICS;
```

The below indicates that the time travel period is completed and currently in the fail safe zone:

ABLE_SCHEMA	TABLE_SCHEMA	TABLE_CATALO	TABLE_CATALO	CLONE_GROUP_	IS_TRANSIENT	ACTIVE_BYTES	TIME_TRAVEL_B	FAILSAFE_BYTE	RETAINED_FOR_	DELETED	TA
5	PUBLIC	5	MYOWN_DB	4100	NO	0	0	0	0	TRUE	
11	INTG_TBLS	5	MYOWN_DB	11266	NO	0	0	7168	0	FALSE	
5	PUBLIC	5	MYOWN_DB	8196	NO	34816	0	0	0	FALSE	
5	PUBLIC	5	MYOWN_DB	5122	NO	0	0	0	0	TRUE	
5	PUBLIC	5	MYOWN_DB	13314	NO	0	0	66560	0	FALSE	

The below query is to convert the no of bytes to GB's to see how much space the table is occupying.

```
SELECT  ID,
        TABLE_NAME,
        TABLE_SCHEMA,
        TABLE_CATALOG,
        ACTIVE_BYTES / (1024*1024*1024) AS STORAGE_USED_GB,
        TIME_TRAVEL_BYTES / (1024*1024*1024) AS TIME_TRAVEL_STORAGE_USED_GB
FROM SNOWFLAKE.ACCOUNT_USAGE.TABLE_STORAGE_METRICS
WHERE TABLE_NAME = 'CUSTOMER_LARGE'
ORDER BY STORAGE_USED_GB DESC,TIME_TRAVEL_STORAGE_USED_GB DESC;
```

To create a schema with certain data retention period:

```
CREATE SCHEMA abcxyz DATA_RETENTION_TIME_IN_DAYS = 10;
SHOW SCHEMAS like 'abcxyz'

SHOW TABLES like 'timetravel_ex%';

CREATE OR REPLACE TABLE myown_db.public.timetravel_ex(id number, name string)
```

Results | Data Preview | Open History

Query ID | SQL | 70ms | 1 rows

Filter result... | Copy | Columns

Row	created_on	name	is_default	is_current	database_name	owner	comment	options	retention_time
1	2022-07-08 2...	ABCXYZ	N	Y	MYOWN_DB	ACCOUNTADM...			10

The above will be applicable for all the underlying tables.

Zero Copy Cloning

- Snowflake allows you to create clones, also known as "zero-copy clones" of tables, schemas, and databases in seconds.
- We can maintain multiple copies of data with no additional cost, so call zero copy.
- A snapshot of data present in the source object is taken when the clone is created, and is made available to the cloned object.
- The cloned object is writable, and is independent of the clone source.
- Changes made to either the source object or the cloned object are not part of the other.
- Two popular use cases of cloning in real time are
  - Cloning Prod data into Dev/Test environment for our unit testing in the lower environment
  - Taking back up of data

***If source tabl;e has 1000 records and its cloned to a new table. The new cloned table is loaded with another 200 new records, then the cost will be incurred only on the new 200 records but not on the 1000 records which are pointed to the original source table. This is how the zero copy cloning works. ***

Why we use zero copy cloning→ when we want to do unit testing or integration testing and bring some data from prod table to dev table, then there will be no charge.

Cloning Syntax:

- CREATE OR REPLACE *'CLONED_OBJECT_TYPE' 'CLONED_OBJECT_NAME'*
  CLONE *'SOURCE_OBJECT';*

→ Here Clone object type can be

Database
Schema
Table
Stage
File Format
Task
Stream

Objects that can be cloned:

- Data Storage Objects
  - Databases
  - Schema
  - Tables
  - Streams
- Data Configuration Objects
  - File Formats
  - Stages
  - Tasks

Hands on:

```
// Cloning a Table
CREATE TABLE myown_db.public.customer_clone
CLONE myown_db.public.customer;

SELECT * FROM myown_db.public.customer;
SELECT * FROM myown_db.public.customer_clone;

// Cloning Schema
CREATE SCHEMA myown_db.copy_of_file_formats
CLONE myown_db.file_formats;


// Cloning Database
CREATE DATABASE myown_db_copy
CLONE myown_db;


//Update data in source and cloned objects and observer both the tables

select * from myown_db.public.customer where customerid=1684012735799;
UPDATE myown_db.public.customer SET CUSTNAME='ABCDEFGH' WHERE CUSTOMERID=1684012735799;
```

While cloning we cannot apply any filters and we will have to clone the entire object.
If we are updating something in the main table it should not affect the cloned table and vice versa.
The below way of creating the backup table also works in snowflake too but its costly. Whereas if we clone the table it doesn't cost as the data is pointing to the main table. Its wrong way of taking the backup of a table

```
// Wrong way to take backup
CREATE OR REPLACE TABLE myown_db.public.customer_clone2
as select * from myown_db.public.customer;
```

Merging time travel concept with cloning:
Until the time travel retention period is active there will be no cost for the cloned table. Once the time travel period is over, then in that case storage cost is associated with the cloned table.

Table Types:
3 types of tables – Permanent, Temporary, Transient Tables.
Permanent Tables: Default table type in Snowflake. These are the regular and common tables. Tables exists until we drop them explicitly. These are the tables which will have the time travel period – 90 days and fail safe - 7 days.

Transient Tables:
These are similar to Permanent tables but with 1 day retention period. There is no fail safe period. Tables exist until we drop them. These types are useful when data protection is not required.
Defining stage tables as transient is best practice. Its only 1 day the time travel period and there is no fail-safe period for this type.
SYNTAX:
CREATE TRANSIENT TABLE ,TABLE NAME>();

Temporary Tables: This type of tables exists only with session. Once the session ends, then the table gets dropped completely and is not recoverable. This will have the retention period as 1 day and the session should be active for 24 hrs, only then we can retain. Though multiple worksheets are opened, this type of table is accessed only for that sql worksheet and cannot be accessed in a different worksheet.

These atbles can be used for temporary processing like it can be used in procedures and drop at the end of the procedure. These are useful for intermediary storage.

SYNTAX:
CREATE TEMPORARY TABLE TABLE_NAME();

Key points to Remember→
We can't convert any type of table to other type.
We can create transient databases and transient schemas.
Tables created under transient databases/schemas are by default transient.
We can create a temporary table with same name as perm/tran table. If we query with that table name, it fetches data from temporary table in that session.
To find the table type → Look at the 'Kind' field in SHOW TABLES properties.

Comparison of Tables:

Table Type	Persistence	Time-Travel Retention period in days	Fail-Safe period in days
Temporary	Until session is active	0 or 1 (default is 1)	0
Transient	Until explicitly dropped	0 or 1 (default is 1)	0
Permanent (Standard Edition)	Until explicitly dropped	0 or 1 (default is 1)	7
Permanent (Enterprise and higher)	Until explicitly dropped	0 to 90 (default is 1, can be modified)	7

```
SHOW TABLES in SCHEMA PUBLIC;

CREATE OR REPLACE TRANSIENT TABLE myown_db.public.tran_table(id number);
CREATE OR REPLACE TEMPORARY TABLE myown_db.public.temp_table(name string);
```

Data Preview

ery ID SQL 57ms 9 rows

Copy

Row	created_on	name	database_name	schema_name	kind
3	2022-06-24 ...	CUSTOMER_...	MYOWN_DB	PUBLIC	TABLE
4	2022-07-10 ...	CUSTOMER_...	MYOWN_DB	PUBLIC	TABLE
5	2022-07-06 ...	EMP_DATA	MYOWN_DB	PUBLIC	TABLE
6	2022-06-28 ...	ORDERS	MYOWN_DB	PUBLIC	TABLE
7	2022-07-08 ...	ORDERS_TT	MYOWN_DB	PUBLIC	TABLE
8	2022-07-10 ...	TEMP_TABLE	MYOWN_DB	PUBLIC	TEMPORARY
9	2022-07-10 ...	TRAN_TABLE	MYOWN_DB	PUBLIC	TRANSIENT

Working with Transient Tables:

The tables which we create in transient schema are transient table type by default. We cannot alter the retention period from 1 to 2 for transient tables. Ex shown in below:

```
==========
// Create transient schema
CREATE OR REPLACE TRANSIENT SCHEMA myown_db.tran_schema;

SHOW SCHEMAS in DATABASE myown_db;

// Create trans table under this trans schema without trans keyword
CREATE OR REPLACE TABLE myown_db.tran_schema.tran_table1(name string);

SHOW TABLES in SCHEMA myown_db.tran_schema;

ALTER TABLE myown_db.tran_schema.tran_table1
SET DATA_RETENTION_TIME_IN_DAYS = 2;

// We can restore trans tables within 24 hours
DROP TABLE myown_db.tran_schema.tran_table1;

SHOW TABLES in SCHEMA myown_db.tran_schema;
```

Results Data Preview

Query ID SQL 55ms

SQL compilation error: invalid value [2] for parameter 'DATA_RETENTION_TIME_IN_DAYS'

To retain the transient table which was dropped a while ago can be re stored in below way:

```
UNDROP TABLE myown_db.tran_schema.tran_table1;

SHOW TABLES in SCHEMA myown_db.tran_schema;

==========
 Temporary Tables
==========
```

sults Data Preview

Query ID SQL 63ms 1 rows

Copy

Row	status
1	Table TRAN_TABLE1 successfully restored.

For Temp tables:

```
// Rename current table and undrop table - will get dropped table
ALTER TABLE myown_db.public.temp_table2 RENAME TO myown_db.public.temp_table3;

UNDROP TABLE myown_db.public.temp_table2;

SELECT * FROM myown_db.public.temp_table2;
```

The table though renamed to different table the actual table still can be retained using the undrop command. This is not only with temporary tables but can be done for permanent tables.
Two different table types can be created with the same name as shown below:

```
// Create a temp table with same name as Perm table
CREATE TEMPORARY TABLE myown_db.public.emp_data(id number);
INSERT INTO myown_db.public.emp_data VALUES(20);
INSERT INTO myown_db.public.emp_data VALUES(47);
INSERT INTO myown_db.public.emp_data VALUES(35);

// Execute this in two diff worksheets and observer results
SELECT * FROM myown_db.public.emp_data;
```

RBAC – Role Based Access Control – Access Control in Snowflake

In real time we may not be operating these, only the admins will operate these. It important to know the below concepts:

- What is Access Control
- Key Concepts of Access Control
- Objects Hierarchy
- Roles in Snowflake
- System Defined Roles
- Role Hierarchy
- Custom Roles

2 types of access controls – RBAC & DAC:

- What is Access Control?
  - Access control determines who can access database objects and perform operations on specific objects in Snowflake.
- Snowflake supports and combines both of below access control models.
  - **Discretionary Access Control (DAC):** Each object has an owner, who can in turn grant access to that object.
  - **Role-based Access Control (RBAC):** Access privileges are assigned to roles, which are in turn assigned to users.

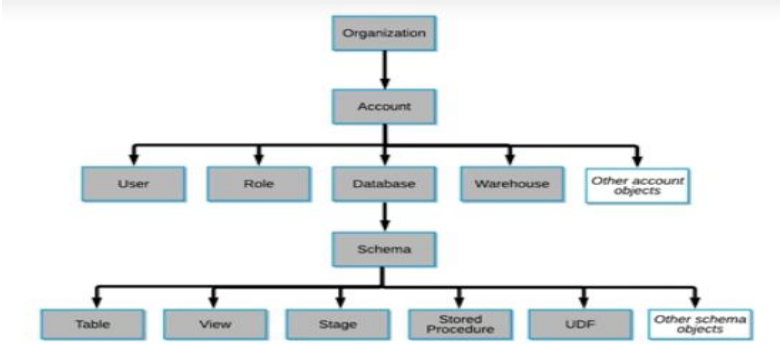
RBAC – access privileges are assigned to some roles and those roles will be assigned to the users.
Below are the key concepts of Snowflake Access Control mechanism.

- **Securable object:** An entity to which access can be granted. Tables, schemas, views etc.
- **Role:** An entity to which privileges can be granted. Roles are in turn assigned to users. Note that roles can also be assigned to other roles, creating a role hierarchy.
- **Privilege:** It is level of access that can be granted to any object. Like select, create, drop, insert etc.
- **User:** Specifies the people or system to whom the access granted.

Different types of Privileges:

Privilege	Usage
SELECT	Execute a SELECT statement on the table.
INSERT	Execute an INSERT query on the table.
UPDATE	Execute an UPDATE query on the table.
TRUNCATE	Execute a TRUNCATE query on the table.
DELETE	Execute a DELETE query on the table.
ALL [PRIVILEGES]	Grant all privileges, except OWNERSHIP, on the table.
OWNERSHIP	Grant full control over a table.

Object Hierarchy:



Roles in Snowflake: Roles are the entities to which privileges are on securable objects can be granted and revoked.
Roles are assigned to users to allow them to perform actions required for business functions.
A user can be assigned multiple roles. This allows users to switch roles.

2 types of Roles: → System defined roles & Custom roles.
Account admin, ORG Admin, Public, Security Admin, Sys Admin, User Admin are the system defined roles.

System Defined Roles:
Org Admin - Organization administrator→ Role that manages the operations at the organization level. Morer specifically, this role can:

- create accounts in the organization
- View all accounts in the organization as well as all regions enabled for the organization.
- View usage information across the organization.

Account Admin – Account Administrator→ It is the top-level role in the system.
Only account admin can see all the account related things like usage. Billing, users, roles, sessions, and reader account details.
This admin role should be granted to only a limited no of users.
AA can enable multi factor authentication.
Encapsulates the SYSADMIN and SECURITY ADMIN system-defined roles.

Security admin and User admin:
Security admin can grant access to roles and users. He will inherit the privileges of user admin role.

3. SECURITYADMIN (Security Administrator):

- Role that can manage any object grant globally,
- Can create, monitor, and manage users and roles.
- Limited access to Account tab.
- Inherits the privileges of the USERADMIN role via the system role hierarchy.

4. USERADMIN (User and Role Administrator):

- Role that is dedicated to user and role management only.
- This role is granted the CREATE USER and CREATE ROLE security privileges.
- Can create users and roles in the account.

System Defined Roles:

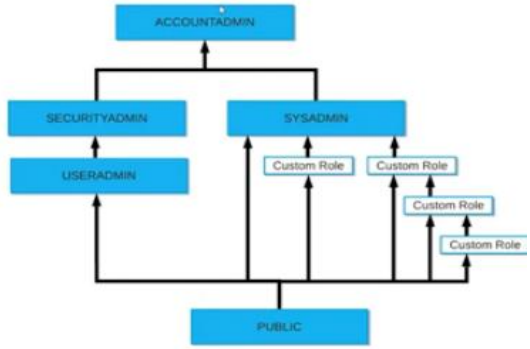
5. SYSADMIN (System Administrator):

- Role that has privileges to create warehouses and databases and other objects in an account.
- This role also has the ability to grant privileges on warehouses, databases, and other objects to other roles.
- All custom roles are assigned to the SYSADMIN role.

6. PUBLIC:

- Automatically granted to every user and every role in your account.
- The objects owned by the role are, by definition, available to every other user and role in the account.
- This role is typically used in cases where all users have all access to the objects.

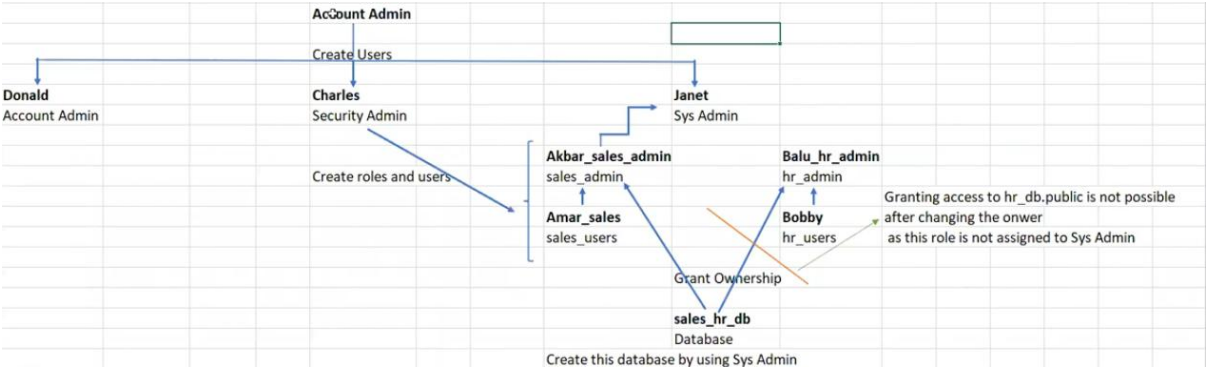
Role Hierarchy in Snowflake:



Account admin is the boss. He will have all access. He can do anything. What ever security admin and sys admin can do can be done by account admin. Whatever user admin can do - can be done by security admin as well. Security admin inherits the properties of user admin.
Public role is the least role. And account admin role is the top role.
The custom roles are user-defined roles.
To explain everything:

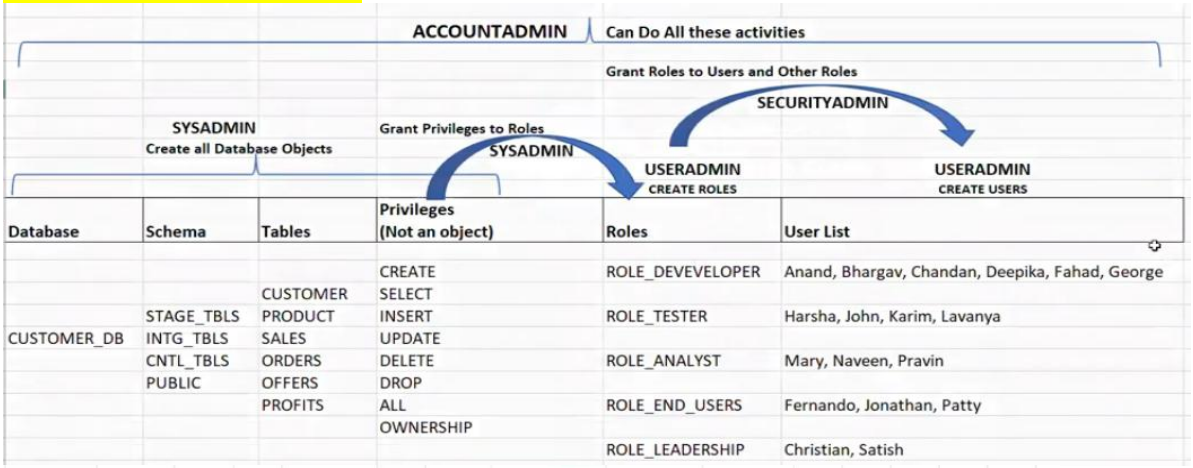
Database	Schema	Tables	Privileges	Roles	User List
			Privileges Set 1	ROLE_DEV	Anand, Bhargav, Chandan, Deepak, Fahad, George
		CUSTOMER			
	STAGE_TBLS	PRODUCT	Privileges Set 2	ROLE_TEST	Harsha, John, Karim, Lavanya,
CUSTOMER_DB	INTG_TBLS	SALES			
	CNTL_TBLS	ORDERS	Privileges Set 3	ROLE_BA	Mary, Naveen, Pravin
	PUBLIC	OFFERS			
		BTCH_TBL	Privileges Set 4	ROLE_END_USERS	Fernando, Jonathan, Patty
		PROFITS			
			Privileges Set 5	ROLE_LEADERSHIP	Christian, Satish

Databases, schemas, table and privileges will be done by the sys admin and roles and users will be done by the security admin.



- Custom roles can be created by the USERADMIN role (or a higher role) as well as by any role to which the CREATE ROLE privilege has been granted.
- By default, a newly-created role is not assigned to any user, nor granted to any other role.
- Snowflake recommends creating a hierarchy of custom roles, with the top-most custom role assigned to the system role SYSADMIN. This will allow system administrators to manage all objects in the account.
- If custom role is not assigned to SYSADMIN through a role hierarchy, the system administrators will not be able to manage the objects owned by that role. Only SECURITYADMIN can view the objects and modify their access grants.

Roles and Users Creation:



Database	Schema	Tables	Privileges (Not an object)	Roles	User List
			CREATE	ROLE_DEVEVELOPER	Anand, Bhargav, Chandan, Deepika, Fahad, George
		CUSTOMER	SELECT		
	STAGE_TBLS	PRODUCT	INSERT	ROLE_TESTER	Harsha, John, Karim, Lavanya
CUSTOMER_DB	INTG_TBLS	SALES	UPDATE		
	CNTL_TBLS	ORDERS	DELETE	ROLE_ANALYST	Mary, Naveen, Pravin
	PUBLIC	OFFERS	DROP		
		PROFITS	ALL	ROLE_END_USERS	Fernando, Jonathan, Patty
			OWNERSHIP		
				ROLE_LEADERSHIP	Christian, Satish

From the above what ever the sales user can do the sales admin also can do.

---- user 1 - Account Admins ---

CREATE USER RANJIT PASSWORD ='ABC123'

DEFAULT_ROLE = ACCOUNTADMIN

MUST_CHANGE_PASSWORD=TRUE;

Password changed to Raghurama123

GRANT ROLE ACCOUNTADMIN TO USER RANJIT;

---- user 2 - Security Admin ----

CREATE USER CHARLES PASSWORD ='ABC123'

DEFAULT_ROLE=SECURITYADMIN

MUST_CHANGE_PASSWORD =TRUE;

GRANT ROLE SECURITYADMIN TO USER CHARLES;

--- user 3 - sysadmin ----

CREATE USER JANET PASSWORD ='ABC123'

DEFAULT_ROLE = SYSADMIN

MUST_CHANGE_PASSWORD = TRUE;

GRANT ROLE SYSADMIN TO USER JANET;

Custom roles mostly assigned to SYSADMIN.

Security admin will not be able to create warehouses.
If we want to grant roles to users or users to objects then below is the way:

```
GRANT USAGE ON DATABASE sales_db TO ROLE sales_users;
GRANT USAGE ON SCHEMA sales_db.public TO ROLE sales_users;
GRANT SELECT ON TABLE sales_db.public.CUSTOMERS TO ROLE sales_users;
```

Views & Materialized Views:

- A View is a database object that contains SQL query built over one or multiple tables.
- A View can be considered as a virtual table that can be used almost anywhere that a table can be used (filters, joins, subqueries, etc.).
- Whenever you query a view, the underlying SQL query associated with the view gets executed dynamically and will fetch data from underlying tables.
- Views serve a variety of purposes like combining, segregating and protecting data.
- A view can be created with below syntax

```
CREATE OR REPLACE VIEW VIEW_NAME
AS
SELECT Statement;
```

In Snowflake we can see 3 types of Views:
Non materialized Views(Normal Views), Secure Views & Materialized Views
Secure views doesn't allow the user to view the definition of the view.

- A Secure View does not allow users to see the definition of the view.
- Users can't see the underlying sql query.
- The definition of a secure view is only exposed to authorized users i.e. users who have been granted the role that owns the view.
- 2 Advantages of secure views
  - Can protect the data by not exposing to other users.
  - I don't want the users to see underlying tables present in our database.
- Use SECURE keyword to create secure views

```
CREATE OR REPLACE SECURE VIEW VIEW_NAME
AS
SELECT Statement;
```

How to determine a view is secure view or not?

IS_SECURE column in the Information_Schema and Account_Usage tells us a view is secure or not.

```
SELECT table_catalog, table_schema, table_name, is_secure
  FROM mydb.information_schema.views
  WHERE table_name = 'VIEW_NAME';

SELECT table_catalog, table_schema, table_name, is_secure
  FROM snowflake.account_usage.views
  WHERE table_name = 'VIEW_NAME';
```

Materialized views:

- A materialized view stores pre-computed result set.
- Querying a materialized view gives better performance than querying the base tables.
- Can create on a single table, can't build on multiple tables.
- Designed for improved query performance when we are using same dataset repeatedly.
- Available in Enterprise edition and higher.
- Can be created with MATERIALIZED key word

```
CREATE OR REPLACE MATERIALIZED VIEW VIEW_NAME
AS
SELECT Statement;
```

Refresh of Materialized Views:

- No need to refresh the data manually.
- Snowflake performs automatic background maintenance of materialized views.
- When a base table changes, Snowflake runs a background process to keep the materialized views up-to-date, but takes a minute to refresh.

Costs of Materialized Views:

Materialized views impact your costs for both storage and compute resources:

- **Storage Cost:** Each materialized view stores query results, which adds to the monthly storage usage for your account.
- **Compute Cost:** In order to prevent materialized views from becoming out-of-date, Snowflake performs automatic background maintenance of materialized views. When a base table changes, all materialized views defined on the table are updated by a background service that uses compute resources provided by Snowflake. So there will be compute cost associated with it.

When to create a materialized view and when to use Normal View:

- Create a materialized view when ***all*** of the following are true:
  - The query results from the view don't change often
  - The results of the view are used often
  - The query consumes a lot of resources means query takes longer time for processing like aggregating data.

- Create a regular view when ***any*** of the following are true:
  - The results of the view change often
  - The results are not used often
  - The query is simple
  - The query contains multiple tables

Advantages of Materialized Views:

- Improves the performance
- No need of additional maintenance, auto refresh of results
- Data accessed through materialized views is always current, regardless of the amount of DML that has been performed on the base table.

Limitations of materialized Views:

Can query a single table only

Does not support Joins, including self-joins

Does not support all aggregate and windowing functions

When the base table is altered or dropped, the materialized view is suspended

Materialized View cannot query

- Another Materialized View
- A Normal View
- A UDF(User Defined Function)

Secure views & Materialized Views are created by Sysadmin.

Dynamic Data Masking Agenda:

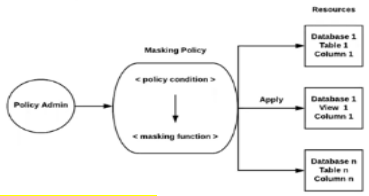
- Column level security
- Masking policies
- Dynamic Data masking
- Creating Mask Policies
- Applying Masking Policies
- Altering or Dropping Policies

Column level security is to protect the information of customer.
PHI – primary health information
PII – personal identifier

Column level security is composed of 2 features:
Dynamic data masking → process of hiding data by masking with other characters. We can create the masking policies to hide the data present in columns.
& External tokenization → process of hiding sensitive data by replacing it with cypher text. External tokenisation makes use of masking policies with external functions created at external cloud provider side.

Masking Policies:

- Snowflake supports masking policies to protect sensitive data from unauthorized access while allowing authorized users to access at query runtime.
- Masking policies are schema level objects.
- Masking policies can include conditions and functions to transform the data when those conditions are met.
- Same masking policy can be applied on multiple columns.



Dynamic data masking: The data is not changed in the storage or in any table, but when executed the output data will be masked dynamically and displayed.

- Sensitive data in Snowflake is not modified in an existing table. But when users execute a query it will apply the masking dynamically and displays the masked data, hence called dynamic data masking.
- The data can be masked, partially masked, obfuscated(unclear), or tokenized data.
- Unauthorized users can operate the data as usual but they can't view the data.
- Mostly masking policies applied based on the Roles.

Authorized role (i.e. SUPPORT)

ID	Phone	SSN
101	408-123-5534	387-78-3456
102	510-334-3564	226-44-8908
103	214-553-9787	359-9987-0098

Unauthorized role (i.e. ANALYST)

ID	Phone	SSN
101	***-**-5534	********
102	***-**-3564	********
103	***-**-9787	********

Creation of masking Policy:

```
// Based on the role
    create masking policy employee_ssn_mask as (val string) returns string ->
      case   when current_role() in ('PAYROLL') then val   else '******'   end;

// Based on some condition
    create masking policy email_visibility as
    (email varchar, visibility string) returns varchar ->
      case
        when current_role() = 'ADMIN' then email
        when visibility = 'Public' then email
        else '***MASKED***'
      end;
```

Applying Masking Policies:

- After creating masking policies, we can apply them wherever we have a requirement to protect the data or hide that data.
- This policy should be applied at column level.
- We can apply same policy on multiple columns from multiple tables and views.

```
// Setting or applying Masking Policy
ALTER TABLE public.employee MODIFY COLUMN ssn
        SET MASKING POLICY employee_ssn_mask;

ALTER TABLE public.employee
        MODIFY COLUMN ssn SET MASKING POLICY employee_ssn_mask,
        MODIFY COLUMN email SET MASKING POLICY email_visibility USING(email, visibility);
```

We can apply the masking policies at column level.

Removing masking policy:

```
// Unsetting or removing Masking Policy

ALTER TABLE public.employee
        MODIFY COLUMN ssn UNSET MASKING POLICY;

ALTER TABLE public.employee
        MODIFY COLUMN ssn UNSET MASKING POLICY,
        MODIFY COLUMN email UNSET MASKING POLICY;
```

We no need to mention the masking policy when we are removing as we can only apply one policy one a column.

For altering and dropping Policies:

```
// Altering or modifying
ALTER MASKING POLICY policy_name SET BODY -> <Case Statement>;

ALTER MASKING POLICY policy_name RENAME TO new_policy_name;

// Dropping
DROP MASKING POLICY policy_name;
```

Limitations:

1. Before dropping masking policies, we should unset them.
2. Data type of input and output values must be same.

We cannot clone the tables which are imported from share.

```
// Want to Hide Phone and Account Balance
CREATE OR REPLACE MASKING POLICY customer_phone
    as (val string) returns string->
CASE WHEN CURRENT_ROLE() in ('SALES_ADMIN', 'MARKET_ADMIN') THEN val
    ELSE '##-###-###-'||SUBSTRING(val,12,4)
    END;

    |
CREATE OR REPLACE MASKING POLICY customer_accbal
    as (val number) returns number->
CASE WHEN CURRENT_ROLE() in ('SALES_ADMIN', 'MARKET_ADMIN') THEN val
    ELSE '####'
    END;


CREATE OR REPLACE MASKING POLICY customer_accbal2
    as (val number) returns number->
CASE WHEN CURRENT_ROLE() in ('SALES_ADMIN', 'MARKET_ADMIN') THEN val
    ELSE 0
    END;
```

To check what are all the masking policies:

SHOW MASKING POLICIES;

if we want to see the particular masking policy:

DESC MASKING POLICY CUSTOMER_PHONE;

Before masking policy is dropped we have to unset them and then drop the policy.

```
105 // To see wherever you applied the policy
106 SELECT * FROM table(information_schema.policy_references(policy_name=>'CUSTOMER_PHONE'));
```

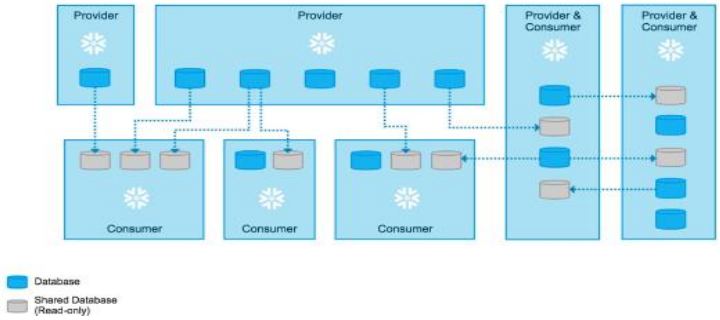
Masking policy applying on a view:

```
// Applying on views
ALTER VIEW MYVIEWS.VW_CUSTOMER MODIFY COLUMN C_PHONE
    SET MASKING POLICY customer_phone;
```

Secure Data Sharing

- Data can be shared securely.
- Can share data to other snowflake users and to non-snowflake users as well.
- For non-snowflake users, we have to create a reader account and share the data.
- Provider – Who is sharing the data by creating share object.
- Consumer – Who is consuming or using the shared data.
- Shared data can be consumed by consumers own compute resources.

For non-snowflake users we must create the reader account and then share it.



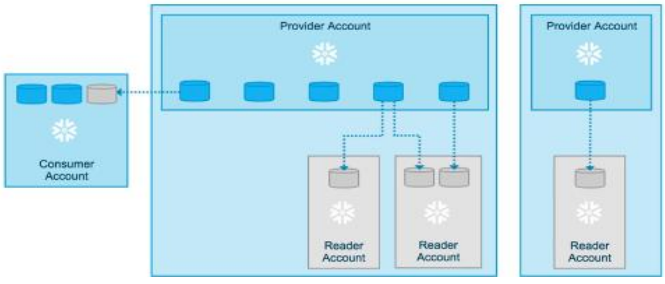
One consumer can get data from multiple providers and one provider can share data to multiple consumers.
A snowflake user can act as a both provider and consumer.
Objects can be shared are:

- Tables
- External tables
- Secure views
- Secure materialized views
- Secure UDFs

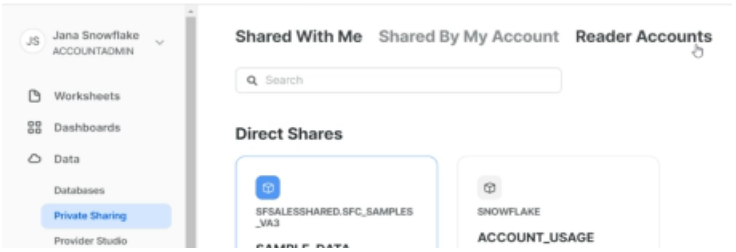
Data sharing is supported only between snowflake accounts.

- Data sharing is only supported between Snowflake accounts.
- If we want to share data with non-snowflake users, we can create reader accounts for them.
- Reader accounts (known as “read-only accounts”) provide a quick, easy, and cost-effective way to share data without requiring the consumer to become a Snowflake customer.
- Each reader account belongs to the provider account that created it.
- Readers can't perform any DML operations, they will have only select access.
- Billing will be calculated in Provider's account itself.

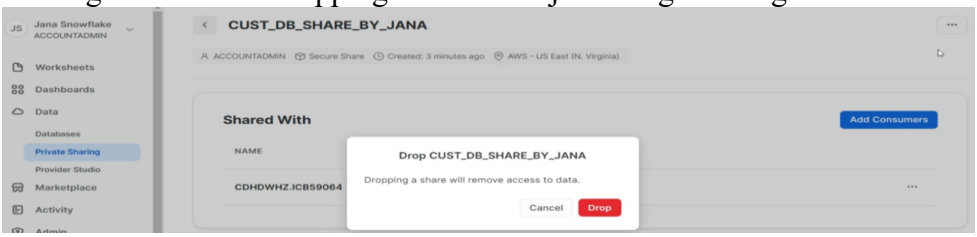
Here are total 3 snowflake accounts:



Consumer account, Reader account, Provider account. The reader account belongs to provider account.



Sharing the data and dropping the share object using snowsight window:



We can share the complete database or schema to other users.

To see the shares on an object we can execute the below:

```
DESC SHARE UAGZKRG.LEB24816.CUST_DATA_SHARE;

// Create a database to consume the shared data
CREATE DATABASE CUST_DB_SHARED FROM SHARE <share name>;

SELECT * FROM CUST_DB_SHARED.CUST_TBLS.CUSTOMER;
```

Objects | Editor | Results | Chart

	kind	name	shared_on
1	DATABASE	CUST_DATA_SHARED	2022-11-20 02:39:51.038 -0800
2	SCHEMA	CUST_DATA_SHARED.CUST_TBLS	2022-11-20 02:40:08.846 -0800
3	TABLE	CUST_DATA_SHARED.CUST_TBLS.CUSTOMER	2022-11-20 02:40:17.710 -0800
4	TABLE	CUST_DATA_SHARED.CUST_TBLS.ORDERS	2022-11-20 02:40:18.381 -0800
5	SCHEMA	CUST_DATA_SHARED.CUST_VIEWS	2022-11-20 02:40:34.063 -0800
6	MATERIALIZED_VIEW	CUST_DATA_SHARED.CUST_VIEWS.SEC_MAT_VW_ORDERS	2022-11-20 02:42:13.347 -0800
7	VIEW	CUST_DATA_SHARED.CUST_VIEWS.SEC_VW_CUST	2022-11-20 02:41:47.377 -0800

We can create a database from the shared:

```
// Create a database to consume the shared data
CREATE DATABASE CUST_DATABASE FROM SHARE UAGZKRG.LEB24816.CUST_DATA_SHARE;

SELECT * FROM CUST_DATABASE.CUST_TBLS.CUSTOMER;
```

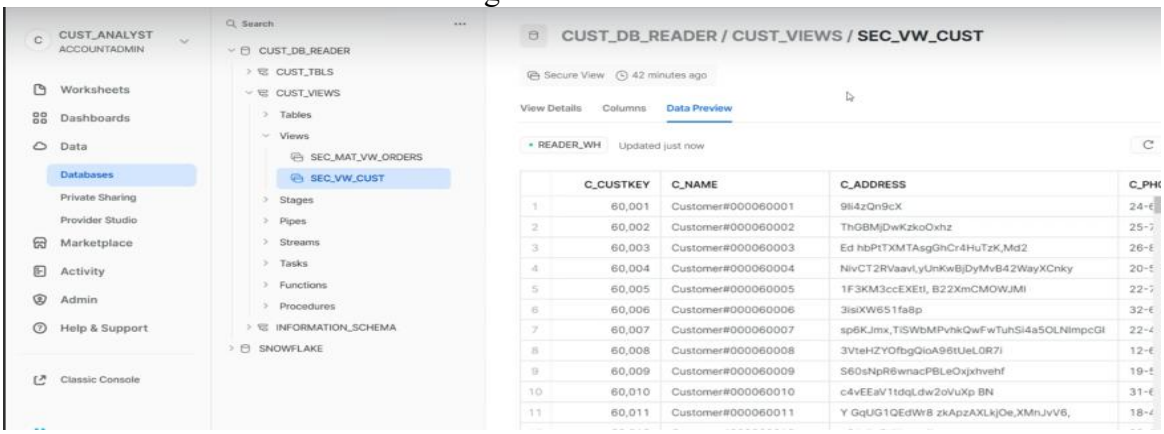
Creating a Reader Account:

```
// Create a reader account

CREATE MANAGED ACCOUNT CUSTOMER_ANALYST
ADMIN_NAME = cust_analyst,
ADMIN_PASSWORD = 'Abcd@123',
TYPE = READER;
```

How to see reader accounts:

SHOW MANAGED ACCOUNTS;

Creation of reader account and sharing that with non-snowflake reader account:



DATA UNLOADING:

Unloading process →The process of loading data into files is the same as loading process except in reverse.

Step1: Use the COPY INTO <location> command to copy the data from the snowflake table into one or more files in snowflake or external stage.

Syntax: COPY INTO @STAGE
FROM TABLE_NAME
<OPTIONS>

There are different options (5) for unloading process:

- OVERWRITE = TRUE | FALSE - Specifies to Overwrite existing files
- SINGLE = TRUE | FALSE - Specifies whether to generate a single file or multiple files
- MAX_FILE_SIZE = <num> - Maximum file size
- INCLUDE_QUERY_ID = TRUE | FALSE - Specifies whether to uniquely identify unloaded files by including a universally unique identifier
- DETAILED_OUTPUT = TRUE | FALSE - Shows the path and name for each file, its size. and the number of rows that were unloaded to the file.

The process for unloading data into files is the same as the loading process, except in reverse:

Step 1

- Use the COPY INTO <location> command to copy the data from the Snowflake database table into one or more files in a Snowflake or external stage.

Step 2

- Download the file from the stage:
  - From a Snowflake stage, use the GET command to download the data file(s).
  - From S3, use the interfaces/tools provided by Amazon S3 to get the data file(s).
  - From Azure, use the interfaces/tools provided by Microsoft Azure to get the data file(s).

Data Sampling:

Data sampling is selecting part of data or subset of records from the table.

This is to build and test the query whether the query is syntactically correct or not.

This is mainly used for query building & testing and also for Data analysis or understanding.

This useful in dev env where we use small wh's and occupy less storage.

We can sample a fraction or % of rows and also we can sample a fixed no of rows.

Sampling Methods

There are 2 sampling methods

1. Bernoulli or Row: - Where the probability of including a row is p/100
   - we can say this gives almost p% of data
   - Good for smaller tables
2. System or Block: - Where the probability of including a block is p/100
   - we can say this gives data from p% of blocks
   - Good for larger tables

For Example if a table contains 4 million records stored in 600 micro partitions and I need 10% of data for my testing.

Bernoulli or Row – It will fetch 10% of 4 mil = 4 lakh rows

System or Block – It will fetch data from 10% of 600 = 60 micro partitions

Syntax:

```
SELECT ... FROM TABLE_NAME
        { SAMPLE | TABLESAMPLE }
        [ samplingMethod ] ( { <probability> | <num> ROWS } )
        [ { REPEATABLE | SEED } ( <seed> ) ]
```

Where,

samplingMethod = { { BERNOULLI | ROW } | { SYSTEM | BLOCK } }

probability | num ROWS: Specifies whether to sample on a fraction of the table or a fixed number of rows in the table, where:

Ex: 10 – 10% data
10 ROWS – Exact 10 rows

REPEATABLE | SEED (seed)

Specifies a seed value to make the sampling deterministic. Seed Can be any integer between 0 and 2147483647. The samples will be the same, as long as the table hasn't been updated.

Some of the samples:

Examples

1. select * from tablename sample row(10);
Return a sample with 10% of rows

2. select * from tablename table sample block (20);
Return a sample with data from 20 blocks

3. select * from tablename table sample system (10) seed (111) ;
Return a sample with data from 10 blocks and guarantees same data set if we use seed 111 next time

4. select * from tablename tablesample (100);
Return an entire table, including all rows into the sample

5. select * from tablename sample row (0);
Return an empty sample

6. select * from tablename sample (10 rows);
Return a fixed-size sample of 10 rows

Why Sampling when Cloning is available

- If you query a cloned table, it will touch the actual big table(may be from Production) every time and so more compute cost
- Where as sampling creates smaller dataset and so less compute cost
- Obviously storage cost is cheaper than compute cost.
- So we can use sampling instead of cloning in scenarios like this where we need small dataset for our testing or dev work.

Sampling the data – Lab part:

```
DEV_DB.PUBLIC

CREATE DATABASE IF NOT EXISTS DEV_DB;

USE DATABASE DEV_DB;
USE SCHEMA PUBLIC;

// Creating tables with sample data

// Bernoulli or Row
CREATE TABLE CUST_SAMPLE_1 AS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE (10);

CREATE TABLE CUST_SAMPLE_2 AS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE ROW (10);

// Sample with fixed number of rows
CREATE TABLE CUST_SAMPLE_5 AS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE ROW (1000 rows);

SELECT COUNT(1) FROM CUST_SAMPLE_5; -- 1,000

// Check the seed is same or not
SELECT * FROM CUST_SAMPLE_4
MINUS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE SYSTEM (5) SEED (111); -- Same
```

In the below case it cannot guarantee the same data as output. Because we are not giving the seed number.

```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE SYSTEM (5)
MINUS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE SYSTEM (5);
```

Objects | Editor | Results | Chart

	C_CUSTKEY	C_NAME	C_ADDRESS
1	9,713	Customer#000009713	URTHBRZI7mGYBtyCFczdh08jmPqXqTRV
2	412,279	Customer#000412279	EvOHB6kU60II7QlON0WWmi8a6sl56eH5,
3	1,098,779	Customer#001098779	Ljha4c I7ASgfLEAFhT

In below case since we are fetching the data from the seed, in both cases the data is same.

```
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE SYSTEM (5) SEED(1000)
MINUS
SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF10.CUSTOMER
    SAMPLE SYSTEM (5) SEED(1000);
```

External Tables in Snowflake:

- Snowflake External tables allow you to query the files stored in external stage like a regular table, that means without moving that data from files to Snowflake tables.
- External tables access the files stored in external stage area such as Amazon S3, GCP bucket, or Azure blob storage.
- External tables store metadata about these data files, such as value(complete record), the filename and file row number.
- External tables are read-only, therefore no DML operations can be performed on them.
- But we can use external tables for query and join operations.
- Views and Materialized views can be created against external tables.
- Querying data from external tables is likely slower than querying database tables.
- Advantage of external table is you can analyze the data with out storing it in Snowflake

Metadata of External Tables:

External tables store below metadata information.

VALUE: A VARIANT type column that represents a single row in the external file.

METADATA$FILENAME: A pseudocolumn that identifies the name of each staged data file included in the external table, including its path in the stage.

METADATA$FILE_ROW_NUMBER: A pseudocolumn that shows the row number for each record in a staged data file.

Steps in creating External Tables:

For creating External tables,

1. Create file format object
2. Create stage object referring to cloud storage location.
3. Create the External table.

External Table Syntax

Syntax:

```
CREATE EXTERNAL TABLE <table_name>
( Columns )
WITH LOCATION = <external stage>
        FILE_FORMAT = <file format object>;
```

Example:

```
CREATE OR REPLACE EXTERNAL TABLE SAMPLE_EXT
(ID INT AS  (VALUE:C1::INT),
NAME VARCHAR(20) AS ( VALUE:C2::VARCHAR),
DEPT INT AS (VALUE:C3::INT)
)
WITH LOCATION = @MYS3STAGE
       FILE_FORMAT = MYS3CSV;
```

```
//Create External Tables

CREATE OR REPLACE EXTERNAL TABLE EXT_TABLES.ET_S3_CUSTOMER(
 CUST_ID NUMBER AS (value:c1::NUMBER),
 CUSTNAME VARCHAR AS (value:c2::VARCHAR),
 EMAIL VARCHAR AS (value:c3::VARCHAR),
 CITY VARCHAR AS (value:c4::VARCHAR),
 STATE VARCHAR AS (value:c5::VARCHAR),
 DOB DATE AS TO_DATE(value:c6::VARCHAR,'YYYY-MM-DD')
)
WITH
LOCATION = @MYOWN_DB.EXT_STAGES.MYS3_STAGE
PATTERN = '.*customer.*'
FILE_FORMAT = MYOWN_DB.FILE_FORMATS.CSV_FILEFORMAT
;

SELECT * FROM EXT_TABLES.ET_S3_CUSTOMER;
```

From an external table always the first field is value field. Which stores the entire record in variant datatype.

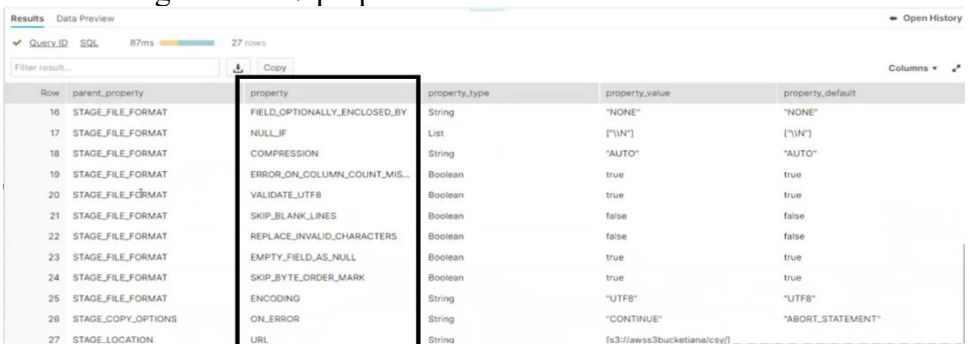
```
CREATE OR REPLACE EXTERNAL TABLE EXT_TABLES.ET_S3_ORDERS(
 CUST_ID NUMBER AS (value:c1::NUMBER),
 NUM_ORDERS NUMBER AS (value:c2::NUMBER)
)
WITH
LOCATION = @MYOWN_DB.EXT_STAGES.MYS3_STAGE
PATTERN = '.*orders.*'
FILE_FORMAT = MYOWN_DB.FILE_FORMATS.CSV_FILEFORMAT
;
```

The data is fetched as shown below:

Row	VALUE	CUST_ID	NUM_ORDERS
1	{ "c1": "1634013037799", "c2": "4" }	1634013037799	4
2	{ "c1": "1667111084599", "c2": "6" }	1667111084599	6
3	{ "c1": "1636040638499", "c2": "7" }	1636040638499	7
4	{ "c1": "1681100256899", "c2": "11" }	1681100256899	11
5	{ "c1": "1626051884899", "c2": "2" }	1626051884899	2
6	{ "c1": "1605082610199", "c2": "5" }	1605082610199	5
7	{ "c1": "1618042063399", "c2": "12" }	1618042063399	12
8	{ "c1": "1699111927399", "c2": "2" }	1699111927399	2
9	{ "c1": "1626012017699", "c2": "8" }	1626012017699	8

```
// To see external tables
DESC EXTERNAL TABLE EXT_TABLES.ET_S3_CUSTOMER TYPE = 'column';
DESC EXTERNAL TABLE EXT_TABLES.ET_S3_CUSTOMER TYPE = 'stage';
```

External stage stores 27 properties.



Row	parent_property	property	property_type	property_value	property_default
16	STAGE_FILE_FORMAT	FIELD_OPTIONALLY_ENCLOSED_BY	String	"NONE"	"NONE"
17	STAGE_FILE_FORMAT	NULL_IF	List	["\\N"]	["\\N"]
18	STAGE_FILE_FORMAT	COMPRESSION	String	"AUTO"	"AUTO"
19	STAGE_FILE_FORMAT	ERROR_ON_COLUMN_COUNT_MIS...	Boolean	true	true
20	STAGE_FILE_FORMAT	VALIDATE_UTF8	Boolean	true	true
21	STAGE_FILE_FORMAT	SKIP_BLANK_LINES	Boolean	false	false
22	STAGE_FILE_FORMAT	REPLACE_INVALID_CHARACTERS	Boolean	false	false
23	STAGE_FILE_FORMAT	EMPTY_FIELD_AS_NULL	Boolean	true	true
24	STAGE_FILE_FORMAT	SKIP_BYTE_ORDER_MARK	Boolean	true	true
25	STAGE_FILE_FORMAT	ENCODING	String	"UTF8"	"UTF8"
26	STAGE_COPY_OPTIONS	ON_ERROR	String	"CONTINUE"	"ABORT_STATEMENT"
27	STAGE_LOCATION	URL	String	[s3://awss3bucketiana/csv/]	

We built the external stages to analyse the data when the data is in source file itself (shown below) and to access the file in the form of table.

```
// Analyze the data

// Requirement: Get the list of customer who placed more than 10 orders
SELECT C.CUST_ID, SUM(O.NUM_ORDERS)
FROM EXT_TABLES.ET_S3_CUSTOMER C
INNER JOIN EXT_TABLES.ET_S3_ORDERS O
ON C.CUST_ID=O.CUST_ID
GROUP BY C.CUST_ID HAVING SUM(O.NUM_ORDERS) > 10;


// Requirement: Get the list of customer who did not placed any order
SELECT C.CUST_ID, SUM(O.NUM_ORDERS)
FROM EXT_TABLES.ET_S3_CUSTOMER C
INNER JOIN EXT_TABLES.ET_S3_ORDERS O
ON C.CUST_ID=O.CUST_ID
GROUP BY C.CUST_ID HAVING SUM(O.NUM_ORDERS) = 0;
```

We can also build the views on external tables.

```
// Views on External tables

// Create a schema for views
CREATE SCHEMA IF NOT EXISTS MYVIEWS;

// Create a secure view
CREATE OR REPLACE SECURE VIEW MYVIEWS.SECVW_ET_CUST
AS
SELECT C.CUST_ID, SUM(O.NUM_ORDERS) TOT_ORDERS
FROM EXT_TABLES.ET_S3_CUSTOMER C
INNER JOIN EXT_TABLES.ET_S3_ORDERS O
ON C.CUST_ID=O.CUST_ID
GROUP BY C.CUST_ID HAVING SUM(O.NUM_ORDERS) > 10;

// Query the secure view
SELECT * FROM MYVIEWS.SECVW_ET_CUST;
```

We can create secured view (top) and materialized view (below) on top of external tables.

```
// Create materialized view
CREATE OR REPLACE materialized VIEW MYVIEWS.MATVW_ET_CUST
AS
SELECT * FROM EXT_TABLES.ET_S3_CUSTOMER WHERE STATE='Victoria';
```

We can easily identify the type of view by looking at the icon symbol of the view as shown below:

If its secure it will be with lock symbol.

USER DEFINED FUNCTIONS

UDF's allows you to perform the operations that are not available through the built in system, defined functions.

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

As shown above we can create the function with same name but different no. of parameters.

```
// Create required schemas
CREATE SCHEMA IF NOT EXISTS MYFUNCTIONS;

************
Scalar UDFs
************

// Create funtion to calculate Tax

// SCENARIO 1 - Fixed Tax
CREATE OR REPLACE FUNCTION MYFUNCTIONS.CUST_TAX(PRICE FLOAT)
RETURNS FLOAT
AS
$$
    (PRICE * 10)/100
$$
;

GRANT USAGE ON FUNCTION MYFUNCTIONS.CUST_TAX(FLOAT) TO PUBLIC;
// Create a function to fetch country name of customer
CREATE OR REPLACE FUNCTION MYFUNCTIONS.GET_COUNTRIES_FOR_USER(ID NUMBER)
  RETURNS TABLE (USER_ID NUMBER, COUNTRY_NAME VARCHAR)
AS
$$
SELECT ID, C.COUNTRY_NAME FROM PUBLIC.USER_ADDRESSES A, PUBLIC.COUNTRIES C
     WHERE A.USER_ID = ID
     AND C.COUNTRY_CODE = A.COUNTRY_CODE
$$
;

// Fetch country name for specified user id
SELECT * from table(MYFUNCTIONS.GET_COUNTRIES_FOR_USER(100));

// Fetch country name for all users
SELECT F.* from PUBLIC.USER_ADDRESSES, table(MYFUNCTIONS.GET_COUNTRIES_FOR_USER(USER_ID)) F;
```

Stored Procedures:

SPs allows you to write procedural code which includes SQL statements, conditional statements, looping statements & cursors.
Snowflake supports 5 languages for writing procedures:
Snowflake scripting
Java
Scala
Python
From a stored procedure, you can return a single value or tabular data.
SP's supports branching and looping.
Dynamically SQL statements can be created and executed.

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

        return 'Done' ;

  $$
;
```

Differences between SP's & UDF's:

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

Stored procedure creation:

```
CREATE OR REPLACE PROCEDURE MYPROCS.CUST_TOT_PRICE(ID INT, CAT VARCHAR)
RETURNS FLOAT
LANGUAGE SQL
AS
$$
declare
total_spent float;
cur1 cursor for select CID, PROD_CAT, PRICE from MYOWN_DB.PUBLIC.CUST_SALES;

begin
total_spent := 0;

for rec in cur1
do
    if (rec.PROD_CAT = :CAT and rec.CID = :ID) then
        total_spent := total_spent+rec.PRICE;
    end if;
end for;
```

The above stored procedure contains like declaration of variables and execution of sql statements.
Need to be careful in writing the stored procedures. Even if we miss a single semicolon its difficult to find it out.

Resource Monitors:

A Virtual warehouse consumes snowflake credits while it runs. The no of credits consumed depends on the size of the warehouse and how long it runs.

A resource monitor can be used to monitor credit usage by virtual warehouses and the cloud services needed to support those warehouses.

Resource monitors helps in controlling costs and avoid unexpected credit usage.
In resource monitors we can set credit limits for a specified interval or date range.

When these limits are reached or approaching, the resource monitor can trigger various actions such as sending alerts and suspending warehouses.

Resource monitors can only be created by account administrators or with the role that has admin privileges.
Resource monitors will reduce the unexpected credit usage and they can help us to track the credit usage.

Credit Quota: This specifies the no. of snowflake credits allocated to the monitor for the specified frequency interval. In addition snowflake tracks the used credits/quota within the specified frequency interval by all warehouses assigned to the monitor. After specified interval this number resets back to 0.

Credit quota accounts for credits consumed by both user managed virtual warehouses and virtual warehouses used by cloud services.

Monitor type: A resource monitor can be created to monitor the credit usage both at account level & warehouse level (single or set of warehouses).
If this property is not set then the resource monitor doesn't monitor any credit usage.

Schedule: The default schedule for a resource monitor specifies that it starts monitoring credit usage immediately and the used credits reset back to 0 at the beginning of each calendar month (i.e. the start of the standard snowflake billing cycle)
We can customize the schedule for a resource monitor using the following properties:

1. Frequency: Daily, monthly, weekly, yearly

Never(Used credits never reset; assigned warehouses continue using credits until the credit quota is reached)

2. Start: Date & time when the resource monitor starts monitoring the assigned warehouses. It can be immediately or any future timestamp.
3. End: Date and Time when snowflake suspends the warehouses associated with the resource monitor, regardless of whether the used credits reached any of the thresholds defined. It can be any future timestamp. Need to be very careful with this property especially. We have to set this property as "never".

Actions:

- Specifies what action can be taken once it reaches to the credit threshold.
- Resource monitors support the following actions:

Notify & Suspend: Send a notification to all account administrators with notifications enabled and suspend all assigned warehouses after all statements being executed by the warehouses have completed.

Notify & Suspend Immediately: Send a notification and suspend all assigned warehouses immediately, which cancels any statements being executed by the warehouses at the time.

Notify: Perform no action, but send an alert notification

Note: We can specify actions up to

- One Suspend action.
- One Suspend Immediate action.
- Up to five Notify actions.

Warehouse suspension and Resumption:

If a monitor has a **Suspend** or **Suspend Immediately** action defined and its used credits reach the threshold for the action, any warehouses assigned to the monitor are suspended and cannot be resumed until one of the following conditions is met:

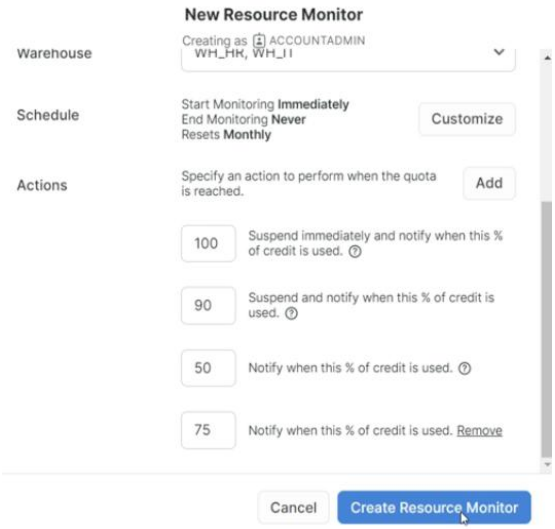
- The next interval, if any, starts, as dictated by the start date for the monitor.
- The credit quota for the monitor is increased.
- The credit threshold for the suspend action is increased.
- The warehouses are no longer assigned to the monitor.
- The monitor is dropped.

Creating Resource Monitors:

We can create monitors in 2 ways.

1. From Web UI
2. By using below sql

```
CREATE [ OR REPLACE ] RESOURCE MONITOR <name> WITH
  [ CREDIT_QUOTA = <number> ]
  [ FREQUENCY = { MONTHLY | DAILY | WEEKLY | YEARLY | NEVER } ]
  [ START_TIMESTAMP = { <timestamp> | IMMEDIATELY } ]
  [ END_TIMESTAMP = <timestamp> ]
  [ NOTIFY_USERS = ( <user_name> [ , <user_name> , ... ] ) ]
  [ TRIGGERS triggerDefinition [ triggerDefinition ... ] ]

Where,
triggerDefinition ::=
  ON <threshold> PERCENT DO { SUSPEND | SUSPEND_IMMEDIATE | NOTIFY }
```



Modifying the Monitors:

We can modify the properties of a monitor from Web UI or by using below Alter Statement.

```
ALTER RESOURCE MONITOR [ IF EXISTS ] <name>
[ SET { [ CREDIT_QUOTA = <num> ]
  [ FREQUENCY = { MONTHLY | DAILY | WEEKLY | YEARLY | NEVER } ]
  [ START_TIMESTAMP = { <timestamp> | IMMEDIATELY } ]
  [ END_TIMESTAMP = <timestamp> ] } ]
  [ NOTIFY_USERS = ( <user_name> [ , <user_name> , ... ] ) ]
  [ TRIGGERS triggerDefinition [ triggerDefinition ... ] ]
```

Creating a monitor at account level:

A resource monitor can be set for your account through either the web interface or using below SQL:

```
ALTER ACCOUNT SET RESOURCE_MONITOR = <Monitor_name>
```

Tasks & Streams:

- We use tasks for scheduling in Snowflake.
- We can schedule
  - SQL queries
  - Stored procedures
- Tasks can be combined with streams for implementing the continuous change data captures(CDC).
- We can maintain a DAG of tasks to keep the dependencies between tasks.
- Tasks require compute resources to execute SQL code, we can choose either of
  - Snowflake managed compute resources(Serverless)
  - User managed (Virtual warehouses)

Creation of Tasks:

By using CREATE TASK Command we can create tasks.

```
CREATE OR REPLACE TASK task_name
        WAREHOUSE = 'warehouse name'
        SCHEDULE = 'Time or Cron'
        AFTER dep_task_name
AS
SQL Statement;
```

SQL Statement can be a query or a call to stored procedure.

Alter the Task:

Altering a Task?

By using ALTER TASK command, you can

1. Add or remove dependency
2. Change the warehouse
3. Change the schedule
4. Modify the SQL statement
5. Add/Remove/Modify the comments
6. Suspend or Resume tasks

Examples:

```
ALTER TASK task_dept ADD AFTER task_emp;
ALTER TASK task_dept REMOVE AFTER task_emp;
ALTER TASK emp_task SET SCHEDULE = '5 minutes';
ALTER TASK emp_task SUSPEND;
```

Some of the sample Tasks:

```
// Task to run a query for every 10 minutes
CREATE OR REPLACE TASK CUSTOMER_INSERT
   WAREHOUSE = COMPUTE_WH
   SCHEDULE = '10 MINUTE'
 AS
 INSERT INTO CUSTOMERS(CREATE_DATE) VALUES(CURRENT_TIMESTAMP);

// Task to call a store proc every day at 9.30 UTC
CREATE OR REPLACE TASK CUSTOMER_INSERT
   WAREHOUSE = MYOWN_WH
   SCHEDULE = 'USING CRON 30 9 * * * UTC'
 AS
CALL PROC_EMPL_DATA_LOAD();
```

Using CRON we will schedule the tasks:

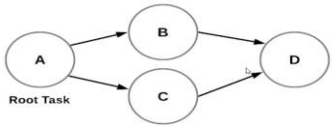
```
# ┌──────────── minute (0-59)
# │ ┌────────── hour (0-23)
# │ │ ┌──────── day of month (1-31, or L)
# │ │ │ ┌────── month (1-12, JAN-DEC)
# │ │ │ │ ┌──── day of week (0-6, SUN-SAT, or L)
# │ │ │ │ │
# │ │ │ │ │
  * * * * *
```

SCHEDULE Value	Description
`* * * * * UTC`	Every minute. UTC time zone.
`0 2 * * * UTC`	Every night at 2 AM. UTC time zone.
`0 5,17 * * * UTC`	Twice daily, at 5 AM and 5 PM (at the top of the hour). UTC time zone.
`30 2 L 6 * UTC`	In June, on the last day of the month, at 2:30 AM. UTC time zone.

DAG of tasks:

Directed acyclic graph

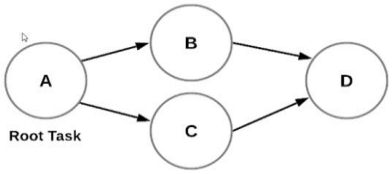
- DAG – Directed Acyclic Graph
- To maintain dependencies between tasks
- A root task followed by child tasks
- Just schedule root task, child tasks will be executed in order.



Creation of tasks setting the dependency between the tasks:

DAG of Tasks

```
CREATE OR REPLACE TASK TASK_A
  WAREHOUSE = COMPUTE_WH
   SCHEDULE = 'USING CRON 30 9 * * * UTC'
  AS    'SQL Query 1';

CREATE OR REPLACE TASK TASK_B
  WAREHOUSE = COMPUTE_WH
  AFTER TASK_A
  AS    'SQL Query 2';

CREATE OR REPLACE TASK TASK_C
  WAREHOUSE = COMPUTE_WH
   AFTER TASK_A
  AS    'SQL Query 3';

CREATE OR REPLACE TASK TASK_D
  WAREHOUSE = COMPUTE_WH
  AS    'SQL Query 4';

ALTER TASK TASK_D ADD AFTER TASK_B;
ALTER TASK TASK_D ADD AFTER TASK_C;
```



Task History:

Task History

- We can check task history from information schema table TASK_HISTORY.

```
// To see all tasks history with last executed task first
select  *   from table(information_schema.task_history())
  order by scheduled_time desc;

// To see results of a specific task in last 6 hours
select  *  from table(information_schema.task_history(
    scheduled_time_range_start =>dateadd('hour',-6,current_timestamp()),
    task_name => 'Task Name'));

// To see results in a given time period
select  *  from table(information_schema.task_history(
    scheduled_time_range_start=>to_timestamp_ltz('2022-07-17 10:00:00.000 -0700'),
    scheduled_time_range_end=>to_timestamp_ltz('2022-07-17 11:00:00.000 -0700')));
```

Some examples of scheduling the Tasks:

```
// Some example schedules

// Every day at 7AM UTC timezone
SCHEDULE = 'USING CRON 0 7 * * * UTC'

// Every day at 10AM and 10PM
SCHEDULE = 'USING CRON 0 10,22 * * * UTC'

// Every month last day at 11 PM
SCHEDULE = 'USING CRON 0 23 L * * UTC'

// Every Monday at 6.30 AM
SCHEDULE = 'USING CRON 30 6 * * 1 UTC'

// Frirst day of the month only from January to March
SCHEDULE = 'USING CRON 0 7 1 1-3 * UTC'
```

To create DAG of tasks:

```
======================
Creating DAG of Tasks
======================
CREATE TABLE MYOWN_DB.PUBLIC.CUST_ADMIN
(CUSTID INT AUTOINCREMENT START = 1 INCREMENT =1,
 CUST_NAME STRING DEFAULT 'JOHN',
 DEPT_NAME STRING DEFAULT 'SALES',
 LOAD_TIME TIMESTAMP
);

// Root task
CREATE OR REPLACE TASK MYTASKS.TASK_CUST_ADMIN
    WAREHOUSE = MYOWN_WH
    SCHEDULE = '1 MINUTE'
AS
INSERT INTO MYOWN_DB.PUBLIC.CUST_ADMIN(LOAD_TIME)
    VALUES(CURRENT_TIMESTAMP);
```

```
// Root task
CREATE OR REPLACE TASK MYTASKS.TASK_CUST_ADMIN
    WAREHOUSE = MYOWN_WH
    SCHEDULE = '1 MINUTE'
AS
INSERT INTO MYOWN_DB.PUBLIC.CUST_ADMIN(LOAD_TIME)
    VALUES(CURRENT_TIMESTAMP);


// Task for loading SALES Table
CREATE OR REPLACE TASK MYTASKS.TASK_CUST_SALES
    WAREHOUSE = MYOWN_WH
    AFTER MYTASKS.TASK_CUST_ADMIN
AS
CREATE OR REPLACE TABLE MYOWN_DB.PUBLIC.CUST_SALES
AS
SELECT * FROM MYOWN_DB.PUBLIC.CUST_ADMIN
    WHERE DEPT_NAME = 'SALES';
CREATE OR REPLACE TABLE MYOWN_DB.PUBLIC.CUST_MARKET
AS
SELECT * FROM MYOWN_DB.PUBLIC.CUST_ADMIN
    WHERE DEPT_NAME = 'MARKET';


// Add dependencies
ALTER TASK MYTASKS.TASK_CUST_MARKET ADD AFTER MYTASKS.TASK_CUST_SALES;
ALTER TASK MYTASKS.TASK_CUST_MARKET ADD AFTER MYTASKS.TASK_CUST_HR;

// Start the tasks - Child first then parent
ALTER TASK MYTASKS.TASK_CUST_MARKET RESUME;
ALTER TASK MYTASKS.TASK_CUST_SALES RESUME;
ALTER TASK MYTASKS.TASK_CUST_HR RESUME;
ALTER TASK MYTASKS.TASK_CUST_ADMIN RESUME;

SHOW TASKS;
```

First we have to resume the child tasks then parent tasks.
To check the tasks History and to check the specific tasks at a certain period of time. we can check the history as below:

```
// To see all tasks history with last executed task first
select  *   from table(information_schema.task_history())
  order by scheduled_time desc;

// To see history of a specific task
select  *  from table(information_schema.task_history(
    task_name => 'TASK_TRACK_TIME')
    );

// To see results of a specific task in last 6 hours
select  *  from table(information_schema.task_history(
    scheduled_time_range_start => dateadd('hour',-6,current_timestamp()),
    result_limit => 10,
    task_name => 'TASK_TRACK_TIME')
    );
// To see results in a given time period
select  *  from table(information_schema.task_history(
    scheduled_time_range_start => to_timestamp_ltz('2022-07-17 1:00:00.000 -0700'),
    scheduled_time_range_end => to_timestamp_ltz('2022-07-18 1:00:00.000 -0700'))
    );

select to_timestamp_ltz(current_timestamp);
```

Streams for CDC

A stream object records dml changes made to the tables like – deletes , updates and inserts.
It stores metadata about each change , so that actions can be taken using this metadata.
We call this process as change data capture.
Streams track all row level changes to source table using offset but doesn't store the changed data.
Once these changes are consumed by the target table, this offset moves to the next point.
Streams can be combined with tasks to set continuous data pipelines.

Snowpipe + Stream + Task → Continuous Data Load.

Metadata of Streams:

Along with the changes made to the source table, Streams maintain 3 metadata fields.

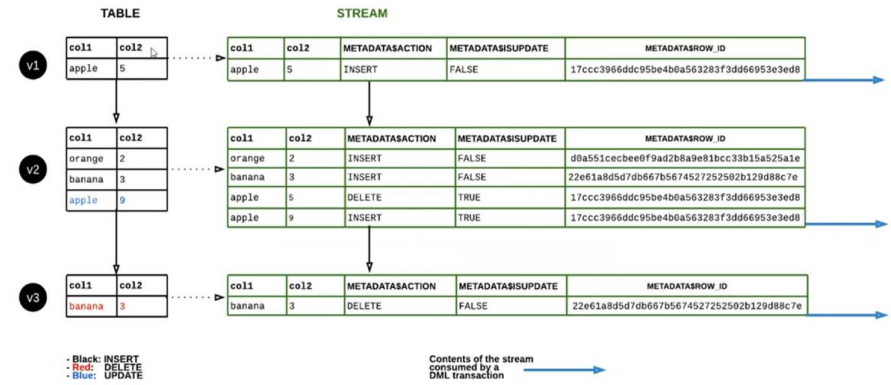
1. METADATA$ACTION: Indicates the DML operation (INSERT, DELETE) recorded.

2. METADATA$ISUPDATE: Indicates whether the operation was part of an UPDATE statement. Updates to rows in the source object are represented as a pair of DELETE and INSERT records in the stream with a metadata column METADATA$ISUPDATE values set to TRUE.

3. METADATA$ROW_ID: Specifies the unique and immutable ID for the row, which can be used to track changes to specific rows over time.

How stream works or data flows in stream:

How a Stream works? or Data flow in a Stream



Types of streams:

1. Standard Streams: A Standard stream tracks all DML changes to the source object, including inserts, updates, and deletes (including table truncates).

```
CREATE OR REPLACE STREAM my_stream ON TABLE my_table;
```

2. Append-only Streams: An Append-only stream tracks row inserts only. Update and delete operations (including table truncates) are not recorded.

```
CREATE OR REPLACE STREAM my_stream ON TABLE my_table
APPEND_ONLY = TRUE;
```

3. Insert-only Streams: Supported for **External tables** only. An insert-only stream tracks row inserts only. They do not record delete operations.

```
CREATE OR REPLACE STREAM my_stream ON EXTERNAL TABLE my_table
INSERT_ONLY = TRUE;
```

Consuming data from streams.

Every update is tracked by one delete and one insert. The stream will track an update record with one delete and insert record.

For tracking updates and deletes we will have to write merge queries.

```
// Consume stream object and merge into final table
MERGE INTO INTG_TBLS.EMPL E
USING MYSTREAMS.STREAM_EMPL S
    ON E.EMPID = S.EMPID
WHEN MATCHED
    AND S.METADATA$ACTION ='INSERT'
    AND S.METADATA$ISUPDATE ='TRUE'   -- indicates the record has been updated
THEN UPDATE
    SET E.EMPNAME = S.EMPNAME,
        E.SALARY = S.SALARY,
        E.AGE = S.AGE,
        E.DEPT = S.DEPT,
        E.LOCATION = S.LOCATION,
```

If we create a snow pipe, task and stream there will be continuous ingestion of data and it's a continuous data pipeline.

Snowflake Alerts and Email Notifications:

When to use snowflake alerts:

If you need to send a notification or perform an action when data in Snowflake meets certain conditions, you can set up a Snowflake Alert.

For example

- The warehouse credit usage increases by a specified percentage of your current quota.
- The resource consumption for your pipelines, tasks, materialized views, etc. increases beyond a specified amount.
- Your data is not as expected to meet business rule that you have set up.
- When a query is taking more time than expected.
- When a table is modified or truncated.

To do this, we can set up a Snowflake alert.

A Snowflake alert is a schema-level object that specifies:

- A condition that triggers the alert.
- The action to perform when the condition is met
- When and how often the condition should be evaluated.

For example, if you want to send an email notification when the credit consumption exceeds a certain limit for a warehouse. And you want to check for this every 30 minutes. You can create an alert with the following properties:

Condition: The credit consumption for a warehouse (the sum of the credits_used column in the WAREHOUSE_METERING_HISTORY view in the ACCOUNT_USAGE) schema exceeds a specified limit.

Action: Email to the administrator.

Frequency/schedule: Check for this condition every 30 minutes.

Privileges needed to create Alert:

In order to create an alert, the user or role must have following privileges:

- The EXECUTE ALERT privilege on the account. Only ACCOUNTADMIN can grant this privilege.

```
GRANT EXECUTE ALERT ON ACCOUNT TO ROLE my_alert_role;
```

- The USAGE and CREATE ALERT privileges on the schema in which you want to create the alert. The Owner of the schema can grant this privilege.

```
GRANT CREATE ALERT ON SCHEMA my_schema TO ROLE my_alert_role;
GRANT USAGE ON SCHEMA my_schema TO ROLE my_alert_role;
```

- The USAGE privilege on the database containing the schema. The Owner of the database can grant this privilege.

```
GRANT USAGE ON DATABASE my_database TO ROLE my_alert_role;
```

- The USAGE privilege on the warehouse used to execute the alert. The Owner of the warehouse can grant this privilege.

```
GRANT USAGE ON DATABASE my_warehouse TO ROLE my_alert_role;
```

Creating and Dropping Alert:

To Create an Alert:

```
CREATE [ OR REPLACE ] ALERT <alert_name>
   WAREHOUSE = <warehouse_name>
   SCHEDULE = '{ <num> MINUTE | USING CRON <expr> <time_zone> }'
   COMMENT = '<string_literal>'
IF( EXISTS( <condition> ))
THEN  <action> ;
```

Note: We have to resume the alert after creating them to start the alerts.

```
ALTER ALERT hrdata.alert_high_salary RESUME;
```

To Drop an Alert:

```
DROP ALERT alert_name;
```

Viewing and executing Alerts:

To see all Alerts:

```
SHOW ALERTS;
```

To see the details of an Alert:

```
DESC ALERT alert_name;
```

Executing Alerts:

In general, the Alerts will be executed at the specified time intervals.

But if we want to execute manually (suppose first time testing purpose or when needed) we can execute using below command.

```
EXECUTE ALERT alert_name;
```

Altering Alerts:

- To suspend alerts
  ```
  ALTER ALERT my_alert SUSPEND;
  ```
- To Resume alerts
  ```
  ALTER ALERT my_alert RESUME;
  ```
- To modify the Alert condition
  ```
  ALTER ALERT my_alert MODIFY CONDITION
  EXISTS (SELECT gauge_value FROM gauge WHERE gauge_value>300);
  ```
- To modify Alert action
  ```
  ALTER ALERT my_alert MODIFY ACTION CALL my_procedure();
  ```
- To modify warehouse name
  ```
  ALTER ALERT my_alert SET WAREHOUSE = my_other_warehouse;
  ```
- To modify schedule time
  ```
  ALTER ALERT my_alert SET SCHEDULE = '2 minutes';
  ```

Alerts Hands on:

```
-- Start the Alert
ALTER ALERT hrdata.alert_high_salary RESUME;

-- Now check the data
SELECT * FROM hrdata.high_salaried_emp;


-- Now insert 2 more records
INSERT INTO hrdata.employees VALUES
( 501,  'Narayan',  'Patil',  'NPATIL',  '420.271.4567',  TO_DATE('22-SEP-1989', 'dd-MON-yyyy'),  'ADM_PRES',  37500,  NULL,  NULL,
90),
( 502,  'Paresh',  'Raval',  'PRAVAL',  '420.271.4567',  TO_DATE('09-NOV-1988', 'dd-MON-yyyy'),  'ADM_PRES',  29000,  NULL,  NULL,
90);

-- Check data after a minute or Execute immediately
EXECUTE ALERT hrdata.alert_high_salary;

SELECT * FROM hrdata.high_salaried_emp;

-- Suspend the Alert, otherwise it will consume warehouse for every minute.
ALTER ALERT hrdata.alert_high_salary SUSPEND;

-- See all Alerts
SHOW ALERTS;

-- Drop Alert
DROP ALERT hrdata.alert_high_salary;
```

Email Notifications:

- Email notifications are used to send email messages or email alerts to users.
- We have to create an email type notification integration object.
- SYSTEM$SEND_EMAIL() is the built-in stored procedure that we can use to send emails.
- Email address must be verified to receive emails from Snowflake.
  - If Email is not configured, you can do it by going to Profile section on Snowflake
- If you specify an email address that hasn't been verified, the Stored procedure will fail and emails will not go.
- A single account can define a maximum of ten email integrations.
- We can send emails inside a stored procedure as well.
- Emails will come from 'no-reply@snowflake.net'
- There is no extra payment for sending emails or for the Snowflake Alerts themselves. Users only pay for the compute time required to run the validation and action queries on the specified virtual warehouse.

Create Notification Integration:

```
CREATE NOTIFICATION INTEGRATION my_email_int
    TYPE=EMAIL
    ENABLED=TRUE
    ALLOWED_RECIPIENTS=('first.last@abc.com','first2.last2@abc.com');
```

If we don't specify Allowed Recipients, e-mail will go to all the users in the account.

Types of possible Alerts:

1. Warehouse usage alert
2. Long running query alert
3. Failed user logins
4. Storage/Credit threshold alerts
5. Warehouse resize/drop alerts
6. Table drop/truncate alerts
7. Snowpipe failure notifications
8. Tasks failure notifications
9. Stored procedure failure notifications
10. Objects(Tables/Schema/Database/Procedures/Streams/Views) modification(ALTER) alerts
11. Business requirements violation alerts

Example Alerts with Email Notification:

Create an Alert to notify users on long running queries(for example queries taking more than 30 min to run).

```
CREATE OR REPLACE ALERT LONG_RUNNING_QUERY_ALERT
WAREHOUSE = COMPUTE_WH
SCHEDULE = '15 MINUTE'
IF (EXISTS
(           SELECT QUERY_ID, QUERY_TEXT, EXECUTION_STATUS, START_TIME, END_TIME, TOTAL_ELAPSED_TIME    FROM
            TABLE(SNOWFLAKE.INFORMATION_SCHEMA.QUERY_HISTORY())
            WHERE EXECUTION_STATUS ILIKE 'RUNNING'
            AND START_TIME <= CURRENT_TIMESTAMP() - INTERVAL '30 MINUTES'
))
THEN CALL SYSTEM$SEND_EMAIL
(           'email_intg',
            'abc.def@xyz.com',
            'Alert: Long Running Queries',
            'There are one or more queries running for more than 30 minutes, Please check query history'
);
```

Testing the email notification:

```
-- Test email
CALL SYSTEM$SEND_EMAIL
('email_alerts', 'jana.snowflake3@gmail.com', 'Hi', 'This is a test email');

-- Create another integration object with non-verified email-address
CREATE NOTIFICATION INTEGRATION email_alerts2
  TYPE = EMAIL
  ENABLED = TRUE;
  ALLOWED_RECIPIENTS = ('jana.snowflake2@gmail.com', 'jana.snowflake3@gmail.com');

-- Test email with newly created intg object
CALL SYSTEM$SEND_EMAIL
('email_alerts2', 'jana.snowflake3@gmail.com, jana.snowflake2@gmail.com', 'Hi', 'This is a test email');
```

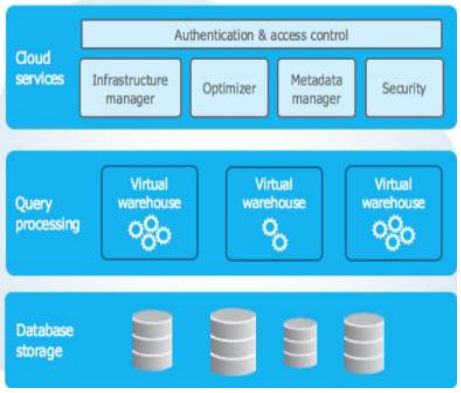
Snowflake Interview questions and Scenario based

1. What is snowflake and what kind of DB it is?

Snowflake is a cloud based DWH available as SaaS, Snowflake enables data storage and data analytics solutions. It doesn't have their own infra structure and currently it can be setup on Azure, GCP and AWS.
Snowflake is pure SQL DB. It organizes the data into multiple micro partitions that are internally optimized and compressed. It uses a columnar format to store.

2. Explain the architecture of Snowflake?

- The cloud services layer is a collection of services that coordinate activities across snowflake.
- Actual processing unit of snowflake, it processes the queries using VWH's.
- Data is stored in columnar format in Micro-partitions.



3. What are the advantages of snowflake over traditional databases or what are the new features available in snowflake?

Lot of new features and advantages:
Pay as you go, No infra structure maintenance, Easy data loading, Time travel and failsafe, Zero copy cloning, Easy data sharing, Tasks and Streams.

4. What are the stages in snowflake and query to create a stage?

Snowflake stages are the locations where data files are stored. There are 2 types of stages in snowflake.
External Stages- If the data that needs to be loaded into snowflake is stored in other cloud regions like AWS S3 or Azure or GCP then we can use external stages.

```
CREATE OR REPLACE EXT_STG_AZURE
    STORAGE_INTEGRATION = azsf_jana_apr22
    URL = 'azure://optumstagejana22.blob.core.windows.net/datalakejana'
    FILE_FORMAT = FILE_FORMAT_AZURE;
```

Internal stages – Stores the data files internally. We can copy files to internal stages by using PUT command from snowsql.

5. Syntax to copy file data into snowflake table:

```
COPY INTO JANA_DB.STAGE_TBLS.STG_EMPL_DTLS
    FROM @EXT_STG_AZURE
    file_format= (type = csv field_delimiter=',' skip_header=1);
    FILES=('empl_dtls1.txt', 'empl_dtls2.txt');
    (or) PATTERN='.*empl_dtls.*';
```

6. Can you use where clause in the copy command?

No we can't use where clause in copy command. We can do some transformations while loading the data by using COPY.

- Select only required fields.
- Can use functions like substring, cast etc.
- Can use case statement

```
COPY INTO JANA_DB.STAGE_TBLS.STG_EMPL_DTLS
    FROM (select
            s.$1,
            s.$3,
            substring(s.$4,1,5)
            CASE WHEN CAST(s.$5 as int) < 0 THEN 'ABC' ELSE 'XYZ' END
          FROM @EXT_STG_AZURE s)
    file_format= (type = csv field_delimiter=',' skip_header=1);
    FILES=('empl_dtls1.txt');
```

7. How can you load a json file to Snowflake or how can you process and load semi structured data?

We can store this semi structured data into a table by using a data type called Variant. Then we can read this data from variant, we can process it into rows and columns and load into another table.

```
CREATE OR REPLACE TABLE JANA_DB.STAGE_TBLS.PETS_DATA_JSON_RAW
(raw_file variant);
```

8. Some performance tuning techniques in snowflake.

-Use cluster keys effectively – don't define on small tables, define on filter columns, define on join keys, define on function based columns.
-Make use of results cache for faster retrieval of data.
-Use materialized views wisely – On more frequently accessed tables, On tables with less frequent data changes.
-and other common sql tuning techniques like- select only required columns, replace or with union, Union all is always better if we are sure there are no duplicates, Try to avoid inequality with 'OR' condition, avoid unnecessary joins, avoid using 'distinct'.

9. How can you handle If the data coming from file is exceeds the length of a field in the table.

Ans: We can handle this by using (truncatecolumns=true) in copy command. If we don't specify this property, copy command will fail. By default, this property is set to FALSE.

```
COPY INTO JANA_DB.STAGE_TBLS.STG_EMPL_DTLS
    FROM @EXT_STG_AZURE
    file_format= (type = csv field_delimiter=',' skip_header=1);
    FILES=('empl_dtls1.txt')
    TRUNCATECOLUMNS = TRUE;
```

10. How is the cost calculated in Snowflake?

2 types of cost – storage cost & compute cost

11. What is clustering in Snowflake?

Clustering is basically grouping a bunch of values togetheso that it improves your query performance. We define cluster keys on big tables, below are the best practices to define cluster keys.
Don't define on small tables.
Define on filter columns.
Define on join keys.
Define on function-based columns.

We can define cluster keys at the time of creating the tables. Also we can add or modify the cluster keys by using alter statement.

12. How many cluster keys is advised on single table?

Snowflake recommends a max of 3 or 4 columns for clustering keys on tables. Adding more than 3-4 columns tends to increase costs more than benefits.

13. Write a query to retrieve data that was deleted from a table?

```
SELECT * FROM TABLE_NAME
    before (timestamp => '2022-07-03 00:00:00.000'::timestamp);

SELECT * FROM TABLE_NAME
    at (offset => -60*60*24*2);

SELECT * FROM TABLE_NAME
    before (statement => '019b9ee5-0500-8473-0043-4d8300073062');
```

14. What are all the objects we can restore after delete or drop?

We can restore the deleted data from any table based on the time travel retention period defined on the table. Based on the edition of snowflake the retention period can be 1 to 90 days. We can un-drop the tables, schemas and databases that were dropped by mistake or wantedly.

15. Write a query to create a table with previous version of another table.
(or) Write a query for Clone with Time Travel.

Ans:

TEST_DB.PUBLIC

```
1  CREATE OR REPLACE TABLE DATABASE.SCHEMA.TABLE_PREV_VER
2      CLONE DATABASE.SCHEMA.TABLE_NAME at (OFFSET => -60*1.5);
```

16. What are all the objects that can be cloned in Snowflake?

Ans: Below objects can be cloned in Snowflake.

Data Containment Objects
- Databases
- Schemas
- Tables
- Streams

Data Configuration and Transformation Objects
- Stages
- File Formats
- Sequences
- Tasks

17. What are Secure Views in Snowflake?

Ans:

If we define the views with secure keyword then unauthorized users can't see the definition of views using GET_DDL, SHOW VIEWS, DESC commands.

Normal views allows anyone to see the view definition.

TEST_DB.PUBLIC

```
1  CREATE OR REPLACE SECURE VIEW SALES_DATA
2  AS
3      SELECT ID, AMOUNT, TRAN_DATE FROM SALES WHERE STATE = 'TX';
```

18. What are all the objects that can be shared in Snowflake?

Ans: The following Snowflake database objects can be shared.

- Tables.
- External tables.
- Secure views.
- Secure materialized views.
- Secure UDFs.

19. What is a materialized view in Snowflake?

Ans: A materialized view is a pre-computed data set derived from a query specification and stored for later use.

Because the data is pre-computed, querying a materialized view is faster than executing a query against the base table of the view.

TEST_DB.PUBLIC

```
CREATE OR REPLACE MATERIALIZED VIEW ORDERS_MV
AS
SELECT
YEAR(O_ORDERDATE) AS YEAR,
MAX(O_COMMENT) AS MAX_COMMENT,
MIN(O_COMMENT) AS MIN_COMMENT,
MAX(O_CLERK) AS MAX_CLERK,
MIN(O_CLERK) AS MIN_CLERK
FROM ORDERS.TPCH_SF100.ORDERS
GROUP BY YEAR(O_ORDERDATE);
```

20. How to refresh the data in materialized views?

Ans: No need to manually refresh material views. After you create a materialized view, a background process automatically maintains the data in the materialized view.

To see the last time that a materialized view was refreshed, check the REFRESHED_ON and BEHIND_BY columns in the output of the command SHOW MATERIALIZED VIEWS.

To see the refresh history of any Materialized View

```
MATERIALIZED_VIEW_REFRESH_HISTORY(
    [ DATE_RANGE_START => <constant_expr> ]
    [ , DATE_RANGE_END => <constant_expr> ]
    [ , MATERIALIZED_VIEW_NAME => '<string>' ] )
```

Refreshed_on & Behind_columns in the output of the above show command.

22. What is the retention period in Business critical edition and how can you increase or reduce it?

Ans: The default retention period in Business critical edition is 90 days.

We can't increase beyond 90 days but we can change it by using below Alter command.

```
ALTER TABLE TABLENAME SET DATA_RETENTION_TIME_IN_DAYS=30;
```

There is a need to choose the time travel retention period less than the 90 days as it would impact costs if the data is not required to be kept till 90 days, in that case it can be reduced.

23. Snowflake is an ETL or ELT?

Ans: Snowflake supports both ETL and ELT.

We can transform and load the data at the same time we can load the data to snowflake and transform it.

There is no indexing concept in snowflake. Instead we can define cluster keys on large tables for better performance.

25. How to grant select on all future tables in a schema and database level?

Ans:

1. Schema level:

```
use role accountadmin;
grant usage on database dbname to role role_name;
grant usage on schema dbname.schemaname to role role_name;
grant select on future tables in schema dbname.schemaname to role role_name;
```

2. Database Level:

```
use role accountadmin;
grant usage on database dbname to role role_name;
grant usage on future schemas in database dbname to role role_name;
grant select on future tables in database dbname to role role_name;
```

Snowflake Interview questions Part-2

26. What is snowpipe and write syntax for creating snowpipe.

Ans: Snowpipe is Snowflake's continuous data ingestion service. Snowpipe loads data within minutes after files are added to a stage and submitted for ingestion.

→ Snowpipe is serverless compute model.

→ Snowpipe provides a "pipeline" for loading fresh data in micro-batches as soon as it is available.

```
CREATE OR REPLACE PIPE PIPE_NAME
AUTO_INGEST = TRUE
AS
COPY INTO DBNAME.SCHEMANAME.TABLENAME  FROM @EXTERNAL_STAGE_NAME;

DESC PIPE PIPE_NAME;
```

27. What are the roles available in Snowflake?

Ans: Roles are the entities to which privileges on snowflake objects can be granted and revoked.

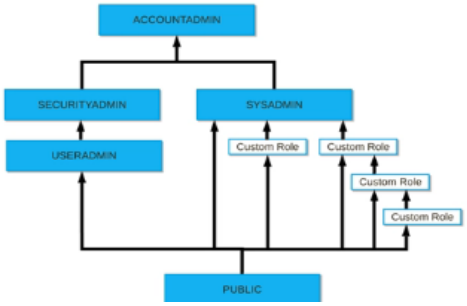
→ Roles are assigned to users to allow them to perform actions required for business functions in their organization.
→ A user can be assigned multiple roles.

Two types of roles in Snowflake.

1. System defined roles
2. Custom roles

27. What are the roles available in Snowflake? -- Continued

Ans:



28. What are the editions of Snowflake and which one you are using in your project?

Ans: There are 4 editions of Snowflake.

1. Standard
2. Enterprise
3. Business Critical
4. Virtual Private

→ These editions differ in terms of multi clusters, time travel, security and many other features.
→ Cost depends on Snowflake edition we choose
  Standard - $2.7/Credit
  Enterprise - $4/Credit
  Business Critical - $5.4/Credit

Most of the organizations use either Enterprise or Business Critical

29. What is the virtual warehouse size you are using in your project? and how many clusters?

Ans:
→ We have to choose size based on the data size you are dealing with and the queries complexity. Size can be anything from XS to 6XL

→ Number of clusters depends on the number of concurrent jobs you are running. You can start with single cluster and you can scale out when your jobs increases.

→ Also VW size and number of clusters depends on the Environment, Dev, Test, Prod etc. based on the data and jobs you are dealing within that environment.

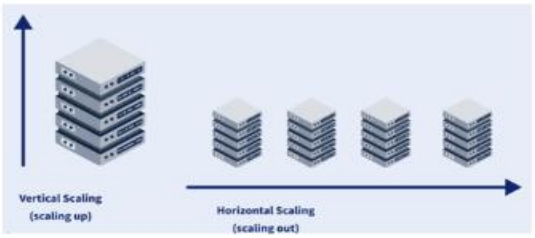
You can say we are using 'M' size with 4 clusters

30. What is Vertical scaling and Horizontal scaling?

Ans:

Vertical Scaling (Scale up) : Increasing the Virtual Warehouse Size to reduce the processing time and make you queries running faster.

Horizontal Scaling (Scale out): Increasing the number of clusters to avoid queries going into queues, you need to scale out when your customer base grows or when your parallel queries/jobs increases.



31. What are the diff table types in Snowflake?

Ans: Snowflake supports 3 types of tables.

1. Permanent Tables: For permanent tables, time travel is 0-90 days of retention period and 7 days fail safe period.

2. Transient Tables: Similar to permanent table but use where data protection and data recovery is not required, 0-1 day retention period and does not support fail safe.

3. Temporary Tables: Only active in that session and gets dropped once we close the session, 0-1 day retention period and does not support fail safe. Can be used in stored procedure for intermediate data storage.

Note: If you create any Database/Schema as Transient then all the tables under that Database/Schema are Transient by default.

32. What is the use of transient tables and temporary tables?

Ans:

Transient tables are specifically designed for transitory data that needs to be maintained beyond each session, but does not need the same level of data protection and recovery provided by permanent tables. You can create the stage tables and intermediate work tables as Transient.

Temporary tables are designed for storing non-permanent and transitory data, you can use them in stored procedures for intermediate data processing, these will get dropped when the execution of stored proc completed.

33. What is the retention period of Permanent, Transient and Temp tables?

Ans:

Permanent tables: 0-90 days retention period and 7 days fail safe period after retention period is completed. We can adjust retention period.

alter table tablename set data_retention_time_in_days=30;

Transient tables: 0-1 day retention period and does not support fail safe.

Transient tables: 0-1 day retention period and does not support fail safe.

Default retention period for all types of tables is 1.

34. Can you create a Temp table with the same name as a Permanent table?

Ans: Yes we can create, and all the queries will be fetching the data from Temporary table in that session.

35. How can you convert a Temp/Transient table into Permanent table?

Ans:

After creation, transient tables cannot be converted to any other table type.

After creation, temporary tables cannot be converted to any other table type.

36. What are different caches available in Snowflake?

Ans: Two types of caches available in Snowflake.

- **Result Cache:** It holds the results of every query executed in the past 24 hours. These are available across virtual warehouses, so query results returned to one user is available to any other user on the system who executes the same query, provided the underlying data has not changed.
- **Local Disk Cache:** It is used to cache data used by SQL queries. Whenever data is needed for a given query it's retrieved from the *Remote Disk* storage, and cached in SSD and memory.

37. What are streams in Snowflake? Write a query to create a Stream.

Ans: A Stream object records the delta of change data capture (CDC) information for a table such as a staging table including inserts and other data manipulation language (DML) changes like Update and Delete.

CREATE OR REPLACE STREAM STREAM_NAME ON TABLE TABLE_NAME;

This stream will record all the changes occurring to the table over time.

38. How the Stream tracks the changes occurring a table?

Ans: Every stream contains 2 fields, based on the values of these 2 fields we can identify the record is a Insert record or Update record or Delete record.

METADATA$ACTION – Insert/Delete
METADATA$ISUPDATE – True/False

METADATA$ACTION=INSERT and METADATA$ISUPDATE=FALSE → Insert Record
METADATA$ACTION=DELETE and METADATA$ISUPDATE= FALSE → Delete Record
METADATA$ACTION= INSERT and METADATA$ISUPDATE= TRUE → Update Record

Note: A Streams records Update operation as a set of Delete(delete old record) and Insert(insert updated record).

Given an example merge query to process all changes Insert/Delete/Update in the description of this video.

39. What are the types of streams available in Snowflake?

Ans: Three types of stream.

Standard Streams: Supported for streams on tables, directory tables(external stages), or views. A standard stream tracks all DML changes to the source object, including inserts, updates, and deletes.

Append-Only Streams: Supported for streams on tables, directory tables, or views. An append-only stream tracks row inserts only. Update and delete operations are not recorded.

Insert-Only Streams: Supported for streams on external tables only. An insert-only stream tracks row inserts only; they do not record delete operations.

40. What is a Task and write syntax to create a Task?

Ans: A Task object is used schedule the execution of a SQL statement, including statements that call stored procedures.

→ We have to provide the Virtual Warehouse to execute the sql statement.
→ We can also use Cron scheduling in Tasks.

```
CREATE OR REPLACE TASK TASK_NAME
   WAREHOUSE = COMPUTE_WH
   SCHEDULE = '1 MINUTE'
   AS
   'SQL Statement';

CREATE OR REPLACE TASK TASK_NAME
   WAREHOUSE = COMPUTE_WH
   SCHEDULE = 'USING CRON 0 7 * * * UTC'
   AS
  'SQL Statement';
```

41. How can you implement column level security in Snowflake?

Ans: We can do this by using Dynamic data masking feature available in Snowflake. Snowflake supports using Dynamic Data Masking on columns of tables and views. We can partially mask the data or fully mask the data from un-authorized users.

First we have to create a masking policy and then we can apply this policy on the columns we wanted to implement security.

```
-- Creating masking policy
create or replace masking policy policy_name
   as (val varchar) returns varchar ->
         case
         when current_role() in ('SYSADMIN', 'ACCOUNTADMIN') then val
         else '######'
         end;

-- Applying policy on a column
ALTER TABLE IF EXISTS table_name MODIFY COLUMN column_name
SET MASKING POLICY policy_name;
```

42. Write syntax to create a stored proc and an UDF.

Ans:

```
create or replace procedure proc_name()
      returns datatype
      language sql /* can write in sql, java, jave script, scala */
      AS
      $$
            sql statements;
      $$;


create function area_of_circle(radius float)
  returns float   /* can write in sql, java, jave script */
  as
  $$
     pi() * radius * radius
  $$
  ;
```

43. Best practices you followed in Snowflake?

Ans:

1. Choose appropriate table type
2. Define cluster keys on large table only and choose proper cluster keys
3. Reduce default retention period
4. Enable auto suspend and auto resume
5. Choose appropriate warehouse size, use scale-up and scale-out effectively
6. Understand storage and compute costs

49. What is the difference between Coalesce and Decode?

Ans:

COALESCE() : The COALESCE() function examines the first expression, if the first expression is not null, it returns that expression Otherwise, it does a COALESCE of the remaining expressions. In simple words COALESCE() function returns the first not-null expression in the list.

Syntax – COALESCE (expr_1, expr_2, ... expr_n)

DECODE(): The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.

Syntax –

DECODE(colexpression, search1, result1 [, search2, result2.....][, default])

50. What is diff between Primary Key, Unique Key and Surrogate Key?

Ans:

Unique key contains unique values in that field, it does not allow duplicates but allows nulls.

Primary key helps identifying the record uniquely in the table, basically it is Unique+Not Null, means it won't allow duplicates and nulls.

Surragte key is a key with no business meaning and it is just a number (mostly we define Surragate Id as Numeric) assigned to each row in the table.

Note 1: A table can contain any number of unique keys but contains only one primary key.

Note 2: A primary key can be combination of multiple columns that define a unique record in the table, but surrogate key is a single column.

51. How to convert a timestamp to date in Snowflake?

Ans:

To extract Date from Timestamp:

```
select to_date('2022-05-22 00:00:00'); -- 2022-05-22
```

To extract Year, Month, Day from Timestamp:

```
select year(to_date('2022-05-22 00:00:00')); -- 2022
select month(to_date('2022-05-22 00:00:00')); -- 05
select day(to_date('2022-05-22 00:00:00')); -- 22
```

52. How to fetch only numeric characters from a string in Snowflake?

Ans:

```
SELECT  TRIM(REGEXP_REPLACE(string, '[^[:digit:]]', '')) AS Numeric_value
FROM
( SELECT ' Area code for employee ID 12345 is 6789.' AS string ) a;
```

53. Tell me some performance tuning techniques of SQL queries.

Ans:

1. Define proper Indexes
2. Define proper partitioning keys. (Cluster keys in Snowflake).
3. Select only required fields, don't put SELECT * blindly.
4. Replace OR with UNION if possible.
5. Use UNION instead of UNION ALL if you are sure there are no duplicates.
6. Use CTEs instead of subqueries if you want to use same result set at multiple places in the query.
7. Use Inner Join instead of putting Cross Join + Where clause.
8. Collecting missing stats on table will help in Teradata.
9. Use materialized views for faster data retrieval and continuous data availability.
10. Look at query execution plan or Query profile to understand where is the bottleneck.

Some other frequently asked sql questions.

1. What is the diff between NVL and NVL2.
2. What are the DML commands in sql?
3. What is diff between varchar and nvarchar?
4. What is an index in database?
5. What is the diff between case and decode?
6. What are the types of slowly changing dimensions?
7. How a data warehouse is different from a database?
8. What is the diff between rank() and dense_rank()?
9. Write a query to find nth highest salary.
10. Write a query to fetch all the employee details with even number emp_id.

Snowflake Interview questions scenario based

Q1: how to give sequence number to newly added column in a table in snowflake?

we can do this by using sequences in snowflake. execute below code to understand.

Example:

```
create table abc(col1 varchar);
insert into abc values ('a'),('b'),('c'),('d'),('e'),('f'),('g'),('h');

alter table abc add column col2 number;
create sequence seq1;
update abc set col2=seq1.nextval;

select * from abc;
```

==========

Q2: While apply masking to a date field can I show only year and rest of the date can be masked?? If yes how can we achieve that??

Yes, we can do that.

If the column is DATE, then we need to return a valid DATE. So maybe you can truncate the dates up to the years.

```
CREATE OR REPLACE MASKING POLICY date_mask_ldm AS (val date)
RETURNS date -> CASE WHEN CURRENT_ROLE() in ('ADMIN')
  THEN val
ELSE DATE_TRUNC( 'year', val ) END;
```

So if the date is '2022-11-30', it will be shown as '2022-01-01' because of the masking policy.

```
select date_trunc('year', current_date);
select current_timestamp;
select date_trunc('year', current_timestamp);
select date_trunc('month', current_timestamp);
select date_trunc('day', current_timestamp);
```

==========

Q3: One of your query is taking long time to run, how can you handle in that situation?

There can be lot of reasons for this, we can resolve with below steps.

1. First check that, query is in running state or in waiting state, if it is waiting state, check how many clusters of virtual warehouses you are using and how many concurrent queries are running. If it is happening every day then we have to scale out(increase no.of clusters) to avoid queries going into waiting queue.
2. If query is in running state, then check how much data you are handling and your virtual warehouse size, if required scaleup your virtual warehouse to next size.
3. Still if query is running for long time, go to query profile where it shows the flow diagram of steps and check which step is taking long time and then we can apply some performance tuning techniques.

Note: if we running same queries again and again enable cache, it will improve the performance a lot.

==========

Q4: How Pivot works in Snowflake?

Pivot operator converts the rows data of the table into the column data.
This operator supports the built-in aggregate functions AVG, MIN, MAX, COUNT, SUM.
Example:

Input:
(empid, amount, month)

```
(1, 10000, 'JAN'),
(1, 400, 'JAN'),
(2, 4500, 'JAN'),
(2, 35000, 'JAN'),
(1, 5000, 'FEB'),
(1, 3000, 'FEB'),
(2, 200, 'FEB'),
(2, 90500, 'FEB'),
(1, 6000, 'MAR'),
(1, 5000, 'MAR'),
(2, 2500, 'MAR'),
(2, 9500, 'MAR').
```

Output:

```
+-------+-------+-------+-------+-------+
| EMPID | 'JAN' | 'FEB' | 'MAR' |
|-------+-------+-------+-------+-------+
|     1 | 10400 |  8000 | 11000 |
|     2 | 39500 | 90700 | 12000 |
+-------+-------+-------+-------+-------+
```

```
select * from monthly_sales
  pivot (sum(amount) for month in ('JAN', 'FEB', 'MAR'))  as p
order by empid;
```

Q5: How Unpivot works in Snowflake?

Unpivot operator converts the column based data into rows, this is opposite to Pivot.

Example:

Input:

```
(empid, dept, jan, feb, mar);
(1, 'electronics', 100, 200, 300),
(2, 'clothes', 100, 300, 150),
(3, 'cars', 200, 400, 100);
```

```
select * from monthly_sales
 unpivot (sales for month in (jan, feb, mar))
order by empid;
```

Output:

```
+-------+-------------+-------+-------++-------
| EMPID | DEPT        | MONTH | SALES |
|-------+-------------+-------+-------+-------
|     1 | electronics | JAN   |   100 |
|     1 | electronics | FEB   |   200 |
|     1 | electronics | MAR   |   300 |
|     2 | clothes     | JAN   |   100 |
|     2 | clothes     | FEB   |   300 |
|     2 | clothes     | MAR   |   150 |
|     3 | cars        | JAN   |   200 |
|     3 | cars        | FEB   |   400 |
|     3 | cars        | MAR   |   100 |
```

==========

Q6: How to get dropped table data, if we recreate a table with same name as dropped table?

There is table with name EMP with 1000 records

- Day1: dropped EMP table
- Day2: created a table with name EMP and inserted 5000 records
- Day3: I need data from EMP table that I had dropped on Day1

How to get that Day1 data?

Ans:

```
ALTER TABLE RENAME EMP to EMP_Day2;
UNDROP TABLE EMP;
ALTER TABLE RENAME EMP to EMP_Day1;
ALTER TABLE RENAME EMP_ Day2 to EMP;
```

Now EMP_Day1 table contains the data from dropped table.

When we can schedule queries by using Tasks in snowflake, why we go for third party scheduling tools?

Answer:

By using Tasks we can schedule the tasks and monitor them from TASK_HISTORY table which is difficult. But third party scheduling tools offer UI based monitoring and it is very easy to control the job flow like holding jobs, releasing dependencies, cancelling or killing jobs etc.

Control-M

Airflow

Ansible

TWS

Active Batch

JAMS Scheduler

Q9: How can I convert my Teradata DDL to Snowflake DDL?

We can't convert 100s of tables DDL from a traditional database to Snowflake manually, but we can use below ways.

1. Use roboquery tool which can convert DDL from any database to any other database including Snowflake, in free version we can convert only 5 per day.
   https://roboquery.com/app/
2. You can develop a python script to convert all DDL at a time.
3. You can write a Procedure in snowflake to convert the DDLs to Snowflake.

Q10: What is the difference btn Full load and Incremental Delta Load? And how to choose?

In full load, the complete data set will be loaded every time where we delete/truncate the target table data and load the new dataset.

In Incremental loads we will fetch only the data that was inserted/updated after previous load and load that to target table by using UPSERT operations(update existing record and insert new record). We can pull incremental data from source with help of LAST_UPDATE_TIMESTAMP and perform UPSERT by using key fields.

How to choose Full load or Incremental load?

1. Based on the data volume and load frequency, we can't perform full loads if we are dealing with large data sets and we have to load every hour or every day.
2. If there are no keys to identify incremental data from Source, we can go for full load.
3. If there are no join keys to perform UPSERTs, we can go for full load.

Q11: My Snowflake account was hosted on Azure, can I load the data that is present in AWS S3 or Google Cloud Storage?

Yes we can read the data from any of the 3 cloud storage providers(Google, Azure, AWS).

As Snowflake doesn't have their own cloud they are dependent(can host) on other cloud storage providers, it doesn't mean that if you host your account on Azure you can only extract data from Azure cloud, you can extract the data from any of these 3 clouds.

Q12: Write a Query to get below output from given input.

Input:

EMP_ID	FULL_NAME	LANG
101	Virat Kohli	English
101	Virat Kohli	Hindi
101	Virat Kohli	Marathi
102	Mahesh Babu	English
102	Mahesh Babu	Telugu
102	Mahesh Babu	Tamil
102	Mahesh Babu	Hindi
103	Janardhan	Telugu
103	Janardhan	English

Output:

FULL_NAME	LANG_SPEAK
Virat Kohli	English,Hindi,Marathi
Mahesh Babu	English,Telugu,Tamil,Hindi
Janardhan	Telugu,English

Query:

```
select FULL_NAME, LISTAGG(LANG, ',') as LANG_SPEAK
from EMP_LANG
group by FULL_NAME;
```

The above can be achieved using listagg in snowflake.

We can implement the SCD ‘s in snowflake using Streams & Tasks.

Q15: In which scenarios you have written stored procedures and UDFs(User Defined Functions)?

We can write stored procedures

1. When there is a need to execute multiple SQL statements in some order.
2. When there is a need to apply same logic multiple times or at multiple places with different parameters.
3. To automate certain things.

We can write UDFs

1. When certain functionality is not available through built-in or system defined functions, and we use that functionality multiple times.

   Ex: we can write a function to calculate Tax_Amount for all employees.

Note: In snowflake we can write procedures and UDFs in multiple languages like SQL, JavaScript, Python, Java and Scala.

Stored procedures for automating data loads

How to write an automated stored procedure?

How to create stored proc for automating data loads from any external clouds?

How to schedule same procedure at different timings to load different tables?

How to Load data into multiple tables?

1. With single stage
2. With single copy command
3. Any type of files
4. From any cloud