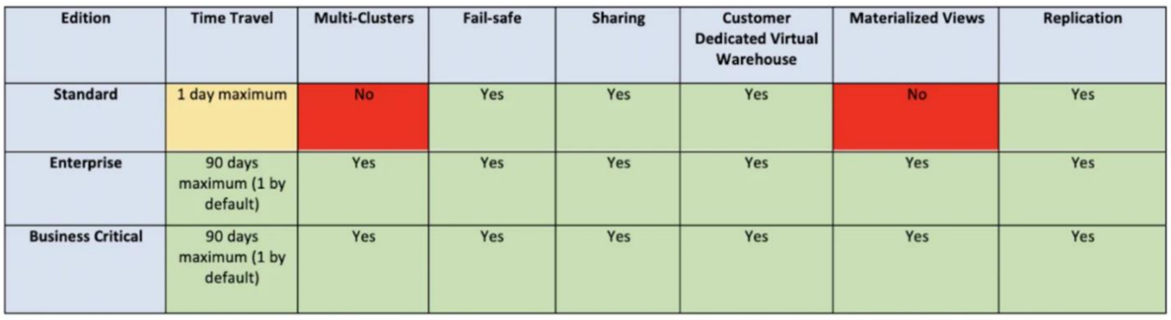
# Snowflake introduction

## SNOWFLAKE AS SAAS

- Snowflake is a Data Solution provided as Software-as-a-Service (SaaS)
- Snowflake enables data storage, processing, and analytic solutions that are faster, easier to use, and far more flexible than traditional offerings.
- It combines a new SQL query engine with an innovative architecture natively designed for the cloud.
- Snowflake releases new features weekly.
- Thanks to this, Snowflake is optimal for:
    - Data Warehouse
    - Data Lake
    - Data Exchange
    - Data Apps
    - Data Science
    - Data Engineering

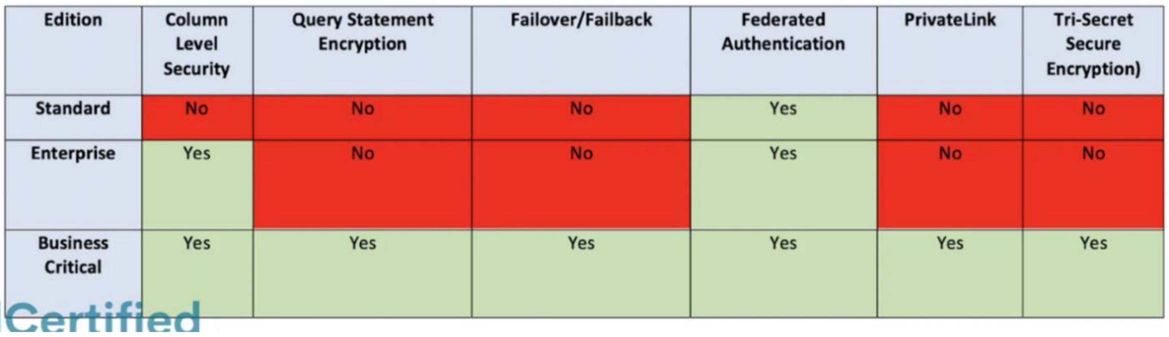
## SNOWFLAKE EDITIONS

- Standard → Introductory level offering, providing full, unlimited access to all of Snowflake's standard features.
- Enterprise → All the features and services of Standard Edition, with additional features designed specifically for the needs of large-scale enterprises and organizations.
- Business Critical → Formerly known as Enterprise for Sensitive Data (ESD), it offers even higher levels of data protection to support the needs of organizations with extremely sensitive data.
- Virtual Private Snowflake → Includes all the features and services of Business Critical Edition but in a completely different Snowflake environment, isolated from all other Snowflake accounts.



| Edition | Time Travel | Multi-Clusters | Fail-safe | Sharing | Customer Dedicated Virtual Warehouse | Materialized Views | Replication |
|---|---|---|---|---|---|---|---|
| Standard | 1 day maximum | No | Yes | Yes | Yes | No | Yes |
| Enterprise | 90 days maximum (1 by default) | Yes | Yes | Yes | Yes | Yes | Yes |
| Business Critical | 90 days maximum (1 by default) | Yes | Yes | Yes | Yes | Yes | Yes |

SnowFlake Features by Edition (I).



| Edition | Column Level Security | Query Statement Encryption | Failover/Failback | Federated Authentication | PrivateLink | Tri-Secret Secure Encryption) |
|---|---|---|---|---|---|---|
| Standard | No | No | No | Yes | No | No |
| Enterprise | Yes | No | No | Yes | No | No |
| Business Critical | Yes | Yes | Yes | Yes | Yes | Yes |

## CLOUD PROVIDERS

- Amazon Web Services.
- Azure.
- Google Cloud Platform.

## CONNECTING TO SNOWFLAKE

- Web Interface or Console
- SnowSQL → The CLI client. This is important to remember.
- ODBC
- JDBC
- SDK (Node, Python, Kafka, Spark, Go…)

Both for the ODBC and JDBC clients, we need to install the drivers from Snowflake.

# Second Chapter: Snowflake Architecture

## SHARED-DISK & SHARED-NOTHING

- Similar to shared-disk architectures → Snowflake uses a central data repository for persisted data accessible from all compute nodes in the platform.
- Similar to shared-nothing architectures → Snowflake processes queries using virtual warehouses where each node in the cluster stores a portion of the entire data set locally.

## SNOWFLAKE LAYERS

1) Centralized Storage → When data is loaded into Snowflake, Snowflake reorganizes that data into its internal optimized, compressed, columnar format in this layer. Snowflake manages all aspects of how this data is stored.

2) Compute → Query execution is performed using virtual warehouses in the compute/processing layer. Each virtual warehouse is an MPP compute cluster composed of multiple compute nodes allocated by Snowflake from a cloud provider.

3) Cloud Services → Collection of services that coordinate activities across Snowflake. It includes:

- Authentication.
- Infrastructure management.
- Metadata management.
- Query parsing and optimization.
- Access control.

4) Cloud Agnostic Layer → We could also talk about the fourth layer, also known as the Cloud Agnostic Layer. It is used only the first time when we choose a provider.

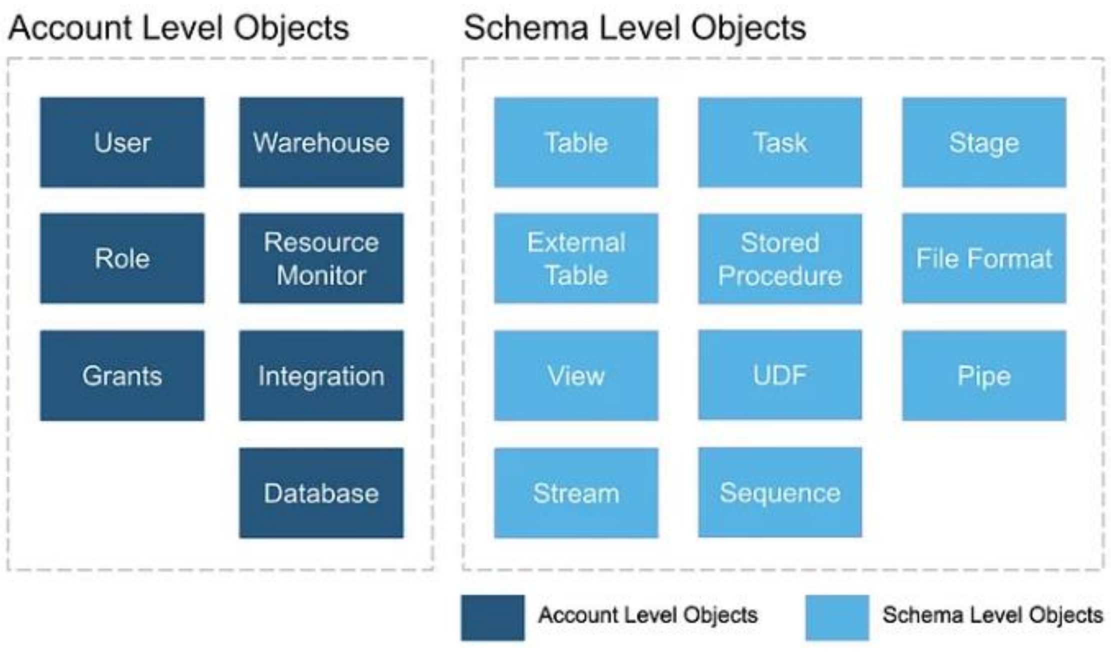
## SNOWFLAKE OBJECTS

- Account → The account's name must be unique within an organization, regardless of which Snowflake Region the account is in. An account could be, for example, "tn10000.eu-west-1.snowflakecomputing.com", although you can ask for a vanity address to convert the identifier into your company's name, for example, "fullcertified.eu-west-1.snowflakecomputing.com".
- Warehouse → Warehouses are the compute part of the Snowflake engine. They are a set of virtual machines provided at the runtime to help execute a given query.

- Database → Logical collection of Schemas. Like any other Database that we already know. It's a logical container for storing other objects, such as Tables and Views, organized in user-defined Schemas.
- Schema → Logical collection of objects that provides a good facility for managing objects and their respective accesses. One schema belongs to a single Database. When a database is created, two schemas are created with it: the Public schema, which stores by default any object created without specifying any schema, and the Information_Schema, which stores the metadata information.

Schemas contain the following objects:

- Tables → Database objects that contain all the data in a database.
- Views → Virtual table whose contents are defined by a query. We can perform a query and save the result in a view.
- Stages → Stage is the location of data files in cloud storage.
- File Formats → Pre-defined format structure that describes a set of staged data to access or load into Snowflake tables for CSV, JSON, AVRO, ORC, PARQUET, and XML input types.
- Sequences → You can generate unique numbers with sequences. They are like counters.
- Pipes → Special and unique type of object in Snowflake which enables the automatic loading of data from files as soon as they are available in a Stage.
- Stored Procedures & User-Defined Functions (UDF) → They let you extend the system to perform operations in SQL and JavaScript.



Snowflake objects are divided by Account or Schema Level.

# Third Chapter: Snowflake Pricing

## COSTS IN SNOWFLAKE

- Snowflake pricing and cost are based on the actual usage.
- we will be charged individually for the computing and storage.
- Storage cost is measured using the average amount of storage used per month, after compression, for all customer data stored in Snowflake.
- Customers pay for virtual warehouses using Snowflake credits.
- The cost of these credits depends on the Snowflake edition that you are using, and the region of the Snowflake account.

Number of credits that a data warehouse consumes is determined by the following:

- The warehouse size, as we will see in the following image.
- The number of clusters (for multi-cluster warehouses).
- The time each server in each cluster runs → They are billed by seconds with a one-minute minimum. For example, if our warehouse runs for 20 seconds, it will be billed as 1 minute, whereas if it runs for 1.20 minutes, it will be billed for this exact time.

## CAPACITY OPTIONS

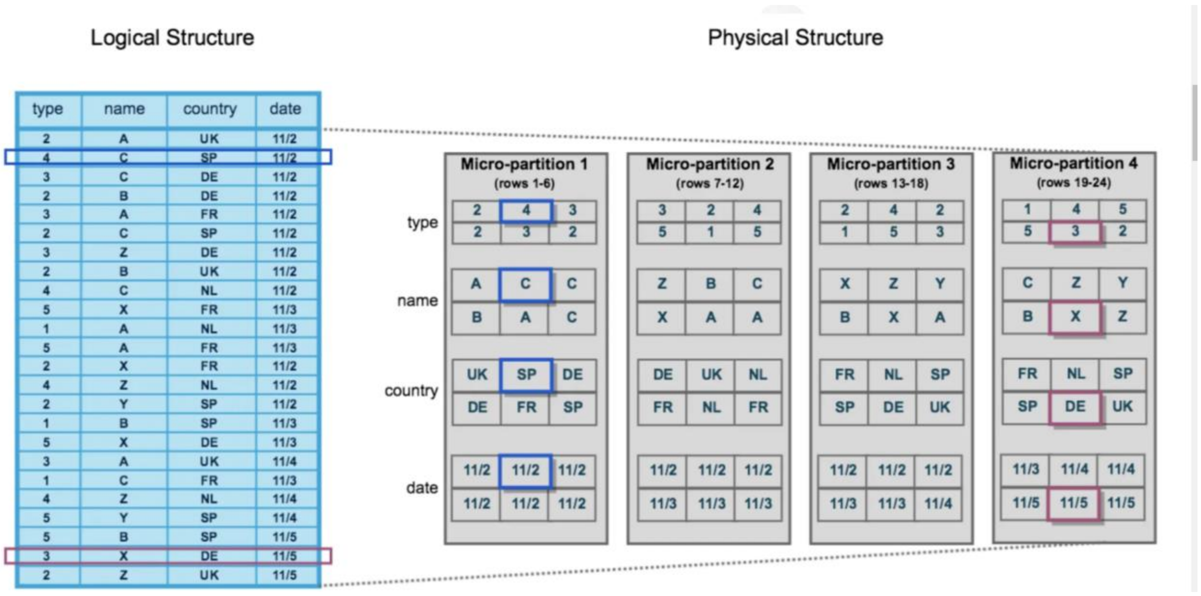
There are two ways to buy the Snowflake Service: On-Demand or pre-paid Capacity.

- On-Demand → Customers are charged a fixed rate for the consumed services and billed in monthly arrears.
- Pre-paid → Pre-purchase Capacity, which involves a commitment to Snowflake. The Capacity purchased is then consumed monthly, providing lower prices and a long-term price guarantee, among other advantages.

# Fourth Chapter: Micro-partitions

## SNOWFLAKE MICRO-PARTITIONS

- All data in Snowflake tables are automatically divided into micro-partitions, contiguous units of storage between 50 and 500MB of uncompressed data, organized in a columnar way.



Logical Structure
Physical Structure
Micro-partition 1 (rows 1-6)
Micro-partition 2 (rows 7-12)
Micro-partition 3 (rows 13-18)
Micro-partition 4 (rows 19-24)
type
name
country
date

| type | name | country | date |
|---|---|---|---|
| 2 | A | UK | 11/2 |
| 4 | C | SP | 11/2 |
| 3 | C | DE | 11/2 |
| 2 | B | DE | 11/2 |
| 3 | A | FR | 11/2 |
| 2 | C | SP | 11/2 |
| 3 | Z | DE | 11/2 |
| 2 | B | UK | 11/2 |
| 4 | C | NL | 11/2 |
| 5 | X | FR | 11/3 |
| 1 | A | NL | 11/3 |
| 5 | A | FR | 11/3 |
| 2 | X | FR | 11/2 |
| 4 | Z | NL | 11/2 |
| 2 | Y | SP | 11/2 |
| 1 | B | SP | 11/3 |
| 5 | X | DE | 11/3 |
| 3 | A | UK | 11/4 |
| 1 | C | FR | 11/3 |
| 4 | Z | NL | 11/4 |
| 5 | Y | SP | 11/4 |
| 5 | B | SP | 11/5 |
| 3 | X | DE | 11/5 |
| 2 | Z | UK | 11/5 |

- Micro partitions are immutable, meaning they cannot be changed once created. If a row is updated, the micro-partition holding the row is copied into a new micro-partition, where the updated row will be inserted. The older micro-partition is then marked for deletion.

## SNOWFLAKE PRUNING PROCESS

- Snowflake uses micro-partitions for the query pruning process, which consists of analyzing the smallest number of micro-partitions to solve a query. This technique retrieves all the necessary data to give a solution without looking at all the micro-partitions, saving a lot of time to return the result.
- When we query the table, Snowflake will know which micro-partition to access just by looking at the metadata. This is the pruning process mentioned above.
- Snowflake also has column pruning, only reading the columns that we need
- Snowflake stores metadata about all rows stored in a micro-partition, including:
    - The range of values for each of the columns in the micro-partition.
    - The number of distinct values.
    - Additional properties are used for both optimization and efficient query processing.

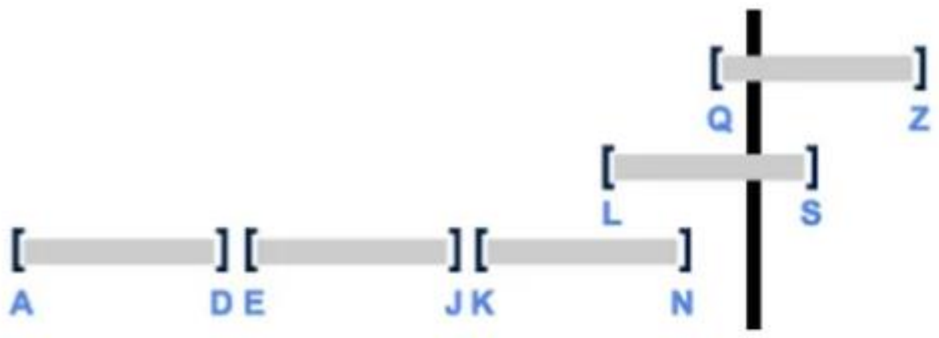
# Fifth Chapter: Clustering

## DATA CLUSTERING

- Typically, data stored in tables is sorted along natural dimensions, for example, by date.
- In Snowflake, clustering metadata is collected and recorded for each micro-partition.
- Clustering metadata that is collected for the micro-partitions in a table:
  - The number of micro-partitions that comprise the table.
  - The number of micro-partitions containing values that overlap with each other.
  - The depth of the overlapping micro-partitions.

## CLUSTERING DEPTH

- The clustering depth measures the average depth of the overlapping micro-partitions for specified columns in a table (1 or greater). The smaller the cluster depth is, the better clustered the table is.



Clustering Depth Example.

The first partition contains data from A to D.
The second partition contains data from E to J.
The third partition contains data from K to N.
The fourth partition contains data from L to S.
The fifth partition contains data from Q to Z.

There are three overlapping micro-partitions.
The clustering depth is 2. Why? Because if you look for data, it will check in two micro-partitions, as we saw before.

- You can use the following commands to get the Cluster Depth:
  - SYSTEM$CLUSTERING_DEPTH
  - SYSTEM$CLUSTERING_INFORMATION

## CLUSTER KEYS

- Clustering keys are a subset of columns or expressions on a table designated to co-locate the data in the same micro-partitions.
- Cluster Keys are placed on columns usually used in the WHERE / JOINS / ORDER BY… commands.

## RECLUSTERING

- From time to time, as DML operations (INSERT, UPDATE, DELETE, MERGE, COPY) are performed on a clustered table, the data in the table might become less clustered.
- To solve that, Snowflake provides periodic & automatic re-clustering to maintain optimal clustering.
- Re-clustering operation consumes both credits and storage
- Clustering is generally more cost-effective for tables that are queried often and do not change frequently.

# Sixth Chapter: Tables & Views

## Permanent tables

- The tables that we can create on Snowflake by default.
- Seven days FAIL-SAFE.
- You can create a temporary, transient, or permanent table as a clone from permanent tables.

## Transient tables

- Transient tables persist until explicitly dropped and are available to all users with the appropriate privileges
- Similar to permanent tables, with the key difference being that they do not have Fail-safe, and they have a shorter Time Travel period
- 0–1 day of Time Travel. You cannot set 90 days.
- No Fail-Safe
- You can create a temporary or transient table as a clone, but you cannot create a permanent table as a clone.

## Temporary tables

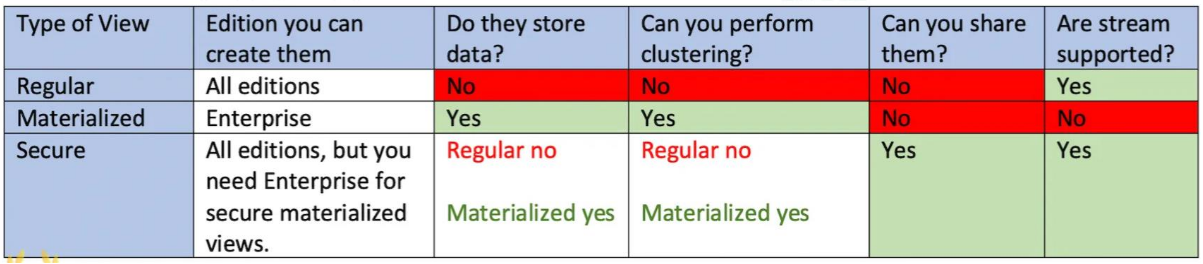
- They store non-permanent data and only exist within the session they were created.
- They are not visible to other users or sessions
- 0–1 day of Time Travel. You cannot set 90 days. A temporary table is purged once the session ends, so the actual retention period is for 24 hours or the remainder of the session.
- No Fail-Safe.
- You can create a temporary or transient table as a clone, but not a permanent table as a clone.

## External tables

- The data is stored in an external stage
- They store file-level metadata about the data files, enabling querying data stored in files in an external stage as if it were inside a database
- You can only perform read operations on these tables
- No Time Travel.
- No Fail-Safe.
- No Cloning.
- External tables are helpful when you have objects in the Cloud Provider that cannot be copied elsewhere by regulations

## VIEWS

- A view allows us to access the result of a query as a table.
- We have two types of views, Non-Materialized and Materialized views, which can be Secure or Not Secure.



| Type of View | Edition you can create them | Do they store data? | Can you perform clustering? | Can you share them? | Are stream supported? |
|---|---|---|---|---|---|
| Regular | All editions | No | No | No | Yes |
| Materialized | Enterprise | Yes | Yes | No | No |
| Secure | All editions, but you need Enterprise for secure materialized views. | Regular no<br><br>Materialized yes | Regular no<br><br>Materialized yes | Yes | Yes |

## Materialized View

- In many ways, it behaves more like a table. The results are stored to allow faster access but require storage space and active maintenance, which incur additional costs.
- Changes to a base table are not automatically propagated to materialized views based on that table
- You can also do clustering in Materialized views.
- This view incurs both storage and computing costs.

## Secure View

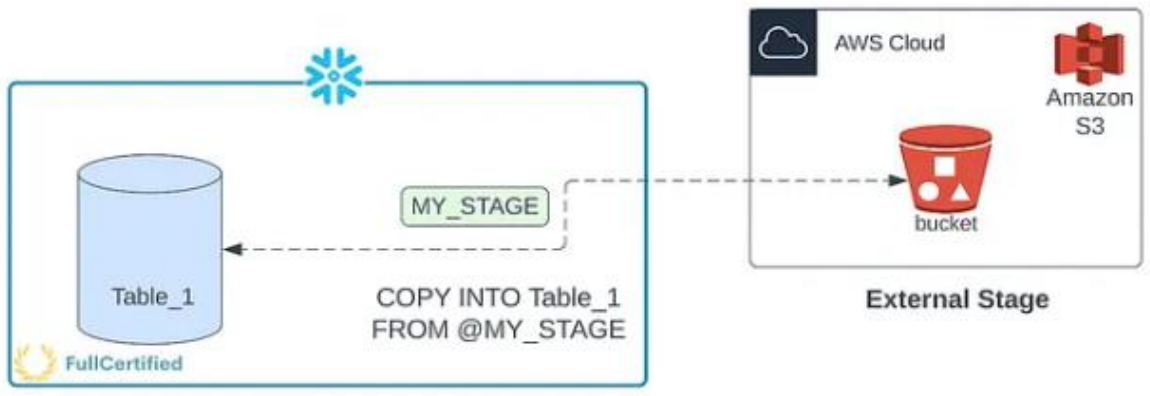
- View definition and details are only visible to authorized users with secure views.
- You cannot see the syntax of how the view was created from the table.
- Both Regular and Materialized views can also be at the same time Secure Views.

# Seventh Chapter: Stages & Storage Integration

- Stages in Snowflake specify where data files are stored (staged)
- It is where the files are before moving them to Snowflake tables
- There are two types of stages, internal and external stages

## EXTERNAL STAGES

- Reference data files stored outside Snowflake
- The following Cloud Storage Services are supported:
  - Amazon S3 Buckets for AWS.
  - Google Cloud Storage Buckets for Google Cloud.
  - Microsoft Azure Containers for Azure.

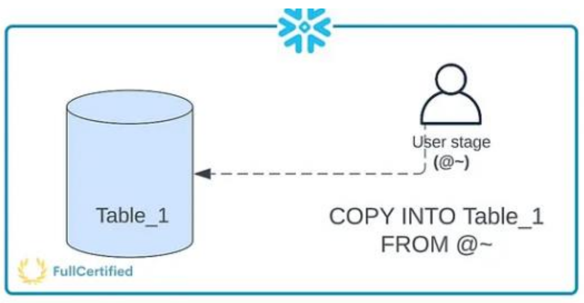


External Stage in Snowflake.

## INTERNAL STAGES

Stores data files internally within Snowflake. There are the following internal stages:

- User (Represented with "@~") → Each user has a Snowflake personal stage allocated to them by default for storing files, and no one can access them except the user it belongs to.
  - you can only access them using the CLI (SnowSQL).
  - data that we copy to the table remains in Snowflake; we don't get it from a Cloud Provider.



- It's important to realize that we use two commands, one to move the data from local to Snowflake and another one to copy from the internal user stage to the Snowflake table

```
PUT file://c:/myData/myCSV.csv @~;

COPY INTO MYTABLE FROM @~staged file_format=(format_name = 'my_csv_format');
```

- Table (Represented with "@%")
  - convenient option if your files must be accessible to multiple users and they only need to be in a single table
  - You cannot see them in the Snowflake interface; only access them using the CLI
  - In the following example, you are copying a file from the user's local directory to the table stage of a table called "myTable":

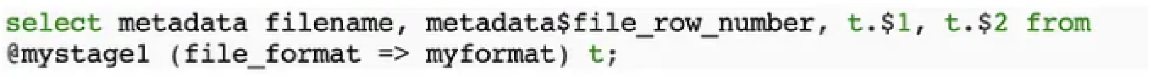
    ```
    PUT file://c:/myData/myCSV.csv @%myTable
    ```

- Named Internal Stage
  - Cloud Storage Location managed by Snowflake
  - Named stages are database objects, both external and internal ones
  - Named Internal stages can be either permanent or temporary.

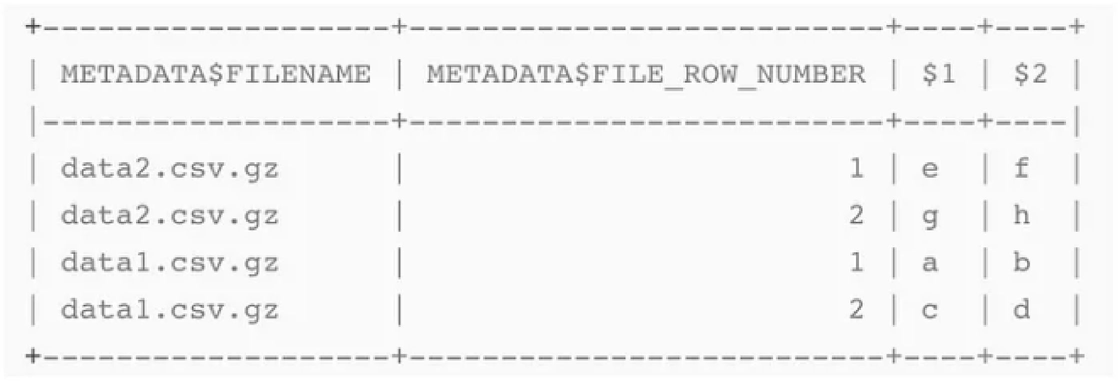
## STAGE METADATA

Snowflake automatically generates metadata for files in internal or external stages. This metadata is "stored" in the following virtual columns:

- METADATA$FILENAME → Name of the staged data file the current row belongs to. It includes the path to the data file in the stage.
- METADATA$FILE_ROW_NUMBER → Row number for each record in the container staged data file.



```
select metadata filename, metadata$file_row_number, t.$1, t.$2 from
@mystage1 (file_format => myformat) t;
```

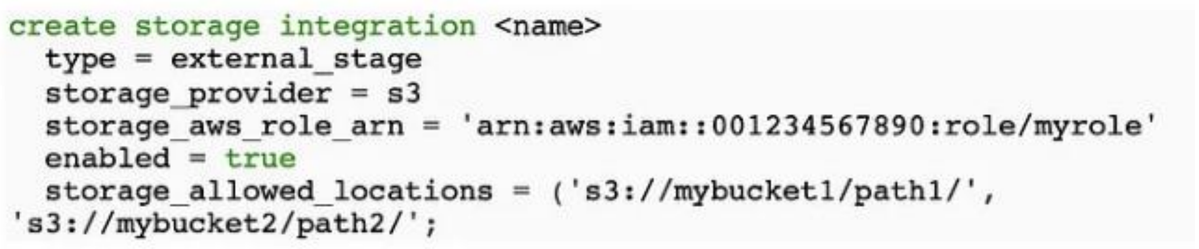


```
+-------------------+-------------------------+----+----+
| METADATA$FILENAME | METADATA$FILE_ROW_NUMBER | $1 | $2 |
|-------------------+-------------------------+----+----|
| data2.csv.gz      |                        1 | e  | f  |
| data2.csv.gz      |                        2 | g  | h  |
| data1.csv.gz      |                        1 | a  | b  |
| data1.csv.gz      |                        2 | c  | d  |
+-------------------+-------------------------+----+----+
```

Example of how to select the metadata from files in a Stage in Snowflake.

## STORAGE INTEGRATION

- Object that stores a generated identity and access management (IAM) entity for your external cloud storage
- This option will enable users to avoid supplying credentials when creating stages or when loading or unloading data.
- Storage integrations are objects at the Account Level



```
create storage integration <name>
  type = external_stage
  storage_provider = s3
  storage_aws_role_arn = 'arn:aws:iam::001234567890:role/myrole'
  enabled = true
  storage_allowed_locations = ('s3://mybucket1/path1/',
's3://mybucket2/path2/';
```

# Eighth Chapter: Data Loading

## DATA LOADING

- When we load the data into Snowflake, we usually perform the following steps:

Source system → Snowflake Stage → Snowflake Table

  - Source system - it can be, for example, your application, or not even that, it can be whatever machine that generates data, like sensors
  - Stage - for example, AWS S3

- There are two different ways to load into Snowflake Table, Bulk Load or Continuous Load

## BULK LOAD

- Bulk load is the process of loading batches of data from files already available at any stage into Snowflake tables
- It supports data transformation while loading, using column reordering, column omission, casting
- COPY INTO - Using this command, you can load data from staged files to an existing table
  - It works for any stage (internal, external, table, and user stages)
  - Some factors affect the loading time, like the physical location of the stage, GZIP compression efficiency or the number and types of columns.
- LOAD → COPY data from the stages into a table. You load data into Snowflake tables.
- UNLOAD → COPY data from the table to stages. You unload data from the tables to a different location (internal stage, external stage, or external location).
- 64 days of metadata. The information about the loaded files is stored in Snowflake metadata. You cannot COPY the same file again in the next 64 days unless you specify it ("FORCE=True" command).
- You cannot Load/Unload files from your Local Drive
- Some transformations like Flatten, Join, Group by, Filters or Aggregations are not supported.
- Using the Snowflake UI, you can only Load 50MB files. You can copy bigger files using SnowSQL.

Loading some of the files might produce errors. For example, you are copying .csv files, and the data is incorrect inside the file. There are several options that you can specify in this case:

- ABORT_STATEMENT → Abort the load operation if there are errors in a data file. If you don't specify any parameter in the ON_ERROR option, this will be the VALUE BY DEFAULT.
- CONTINUE → Continue loading the file.
- SKIP_FILE → Skip file if there are errors in the files.

- SKIP_FILE_num → Skip the file when the number of error rows in the file equals or exceeds the specified number.
- SKIP_FILE_num% → Skip the file when the percentage of error rows in the file exceeds the specified percentage.
- pattern = <pattern> → Load files from a stage into the table, using pattern matching.
- FORCE = TRUE → Once the files are copied into a table, they cannot be copied again in the next 64 days because of the files' metadata. If this option is true, it loads all files, regardless of whether they've been loaded previously and have not changed since they were loaded.
- PURGE = TRUE → It specifies whether to automatically remove the data files from the stage after loading the data successfully. If the purge operation fails for any reason, no error is returned. An excellent way to check whether there was an error or not would be to list the files from the stage with the "LIST stage" command.
- MAX_FILE_SIZE → You can specify the maximum size for each file when unloading the data with this option.

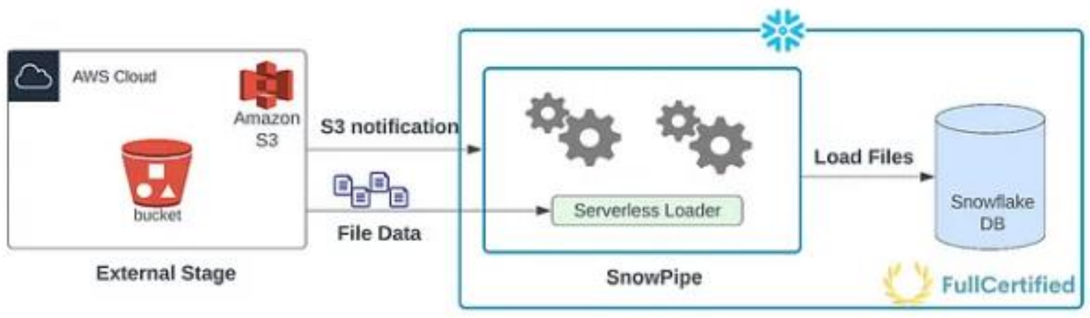
## CONTINUOUS LOAD

Load small volumes of data (micro-batches) and incrementally make them available for analysis. There are different ways to do that:

- Snowpipe → The easiest and most popular way to do it. We'll see it in this chapter.
- Snowflake Connector for Kafka → Reads data from Apache Kafka topics and loads the data into a Snowflake table.
- Third-Party Data Integration Tools → You can do it with other supported integration tools.

## SNOWPIPE

- You use it when you have a small volume of frequent data, and you load it continuously (micro-batches).
- Snowpipe is serverless, which means that it doesn't use Virtual Warehouses.
- Snowpipe does not guarantee that files will be loaded in the same order as staged



The real question is, how can Snowpipe detect new files in the stage? There are two different ways to do it:

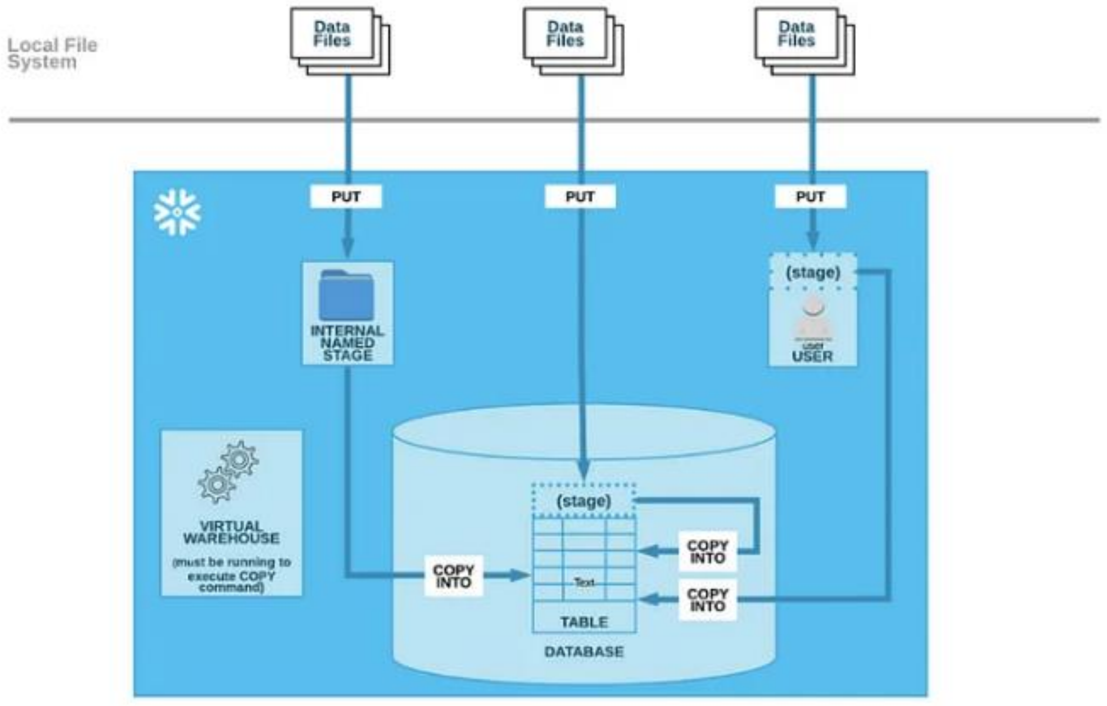
- Automating Snowpipe using cloud messaging → You can do a trigger when there is an event in the stage (for example, when there is a new document) using Amazon SQS (Simple Queue Service) notifications for an S3 bucket.

- Calling Snowpipe REST endpoints → Your client application calls a public REST endpoint with the name of a pipe object and a list of data filenames.

# Ninth Chapter: PUT and GET commands

## PUT COMMAND

- We will use the PUT command to UPLOAD files from a local directory/folder on a client machine into INTERNAL STAGES
- It does NOT work with external stages.
- It doesn't copy the files in the tables; it copies them into the internal stages
- If you want to move the data into tables, you need to use the COPY INTO command
- You cannot use the PUT command from the Snowflake Web UI

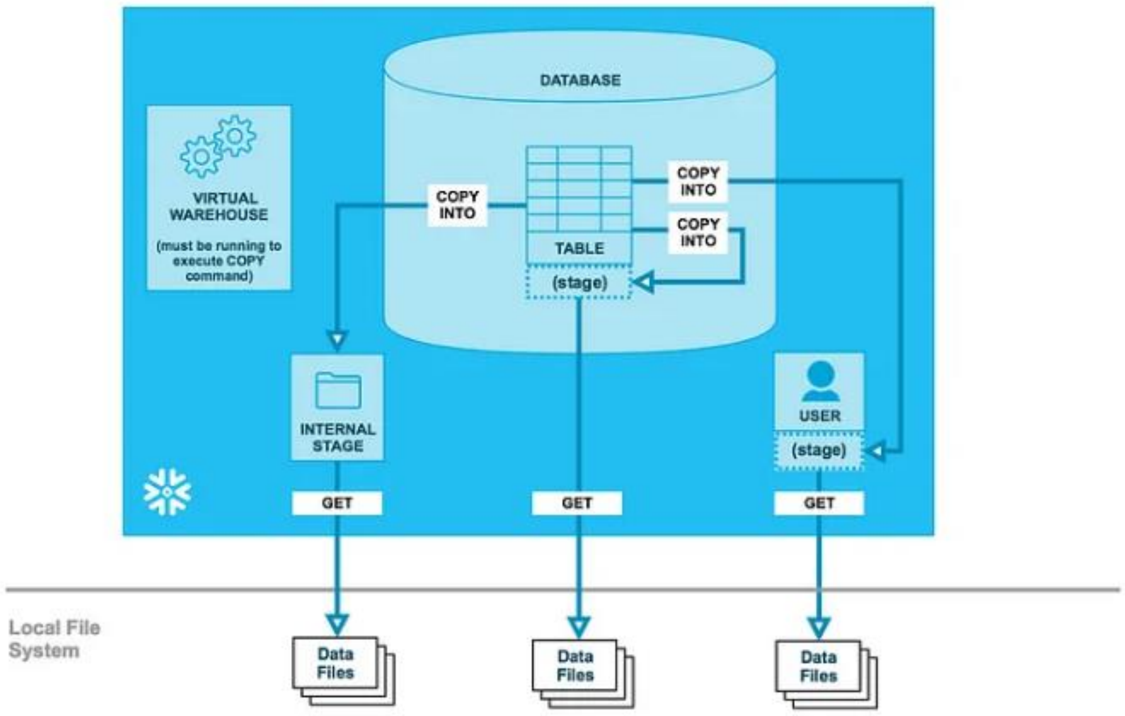


PUT Command diagram (via docs.snowflake.com).

PUT file:///tmp/data/mydata.csv @my_int_stage;

## GET COMMAND

- We will use the GET command to DOWNLOAD files from a Snowflake internal stage into a directory/folder on a client machine.
- It doesn't download the data from the table; it downloads the data from the stage, so first of all, you need to unload the data from the table using the COPY INTO command to the internal stage.
- As with the PUT command, you cannot use it with the Snowflake Web UI; you need to use SnowSQL.

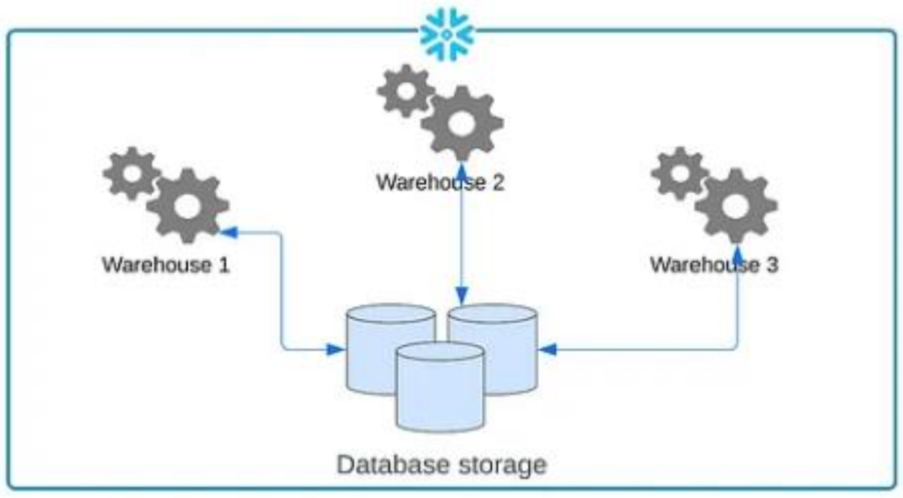


```
GET @my_int_stage file:///tmp/data/;
```

# Tenth Chapter: Data Warehouses

## WAREHOUSES IN SNOWFLAKE

- A Data Warehouse is a cluster of computing resources in Snowflake. It provides the required resources, such as CPU, memory, and temporary storage, to perform queries and DML operations
- While a warehouse is running, it consumes Snowflake credits.



Virtual Warehouses in Snowflake

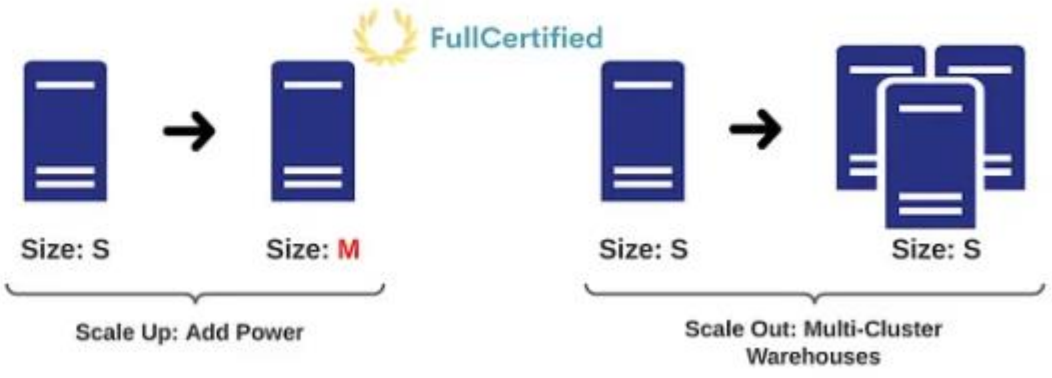
- Snowflake utilizes per-second billing (with a 60-second minimum each time the warehouse starts)
- A Warehouse is defined by its size and other properties:
  - Size - size of a warehouse can impact the amount of time required to execute queries, query will be executed faster in a Large warehouse than in a Small warehouse
  - Multi-Cluster Warehouses - with multi-cluster warehouses, you can scale compute resources to manage query concurrency during, for example, peak hours, You need (at least) the Snowflake Enterprise Edition (typical exam question) to activate this option

To create multi-cluster warehouses, you need to specify the following properties:

- Maximum clusters (1–10)
- Minimum clusters (≤ than the maximum)

Scale Up vs Scale-Out

Multi-cluster warehouses are best utilized for scaling resources to improve concurrency for users/queries, also known as scale OUT/IN. If we wanted to improve the performance of the queries, we should resize the warehouse, also known as scale UP/DOWN the Data Warehouse.



Scale Up vs. Scale Out a Snowflake Warehouse.

### Multi-warehouse modes

- Maximized → We will enable this mode by specifying the SAME value (larger than 1) for the maximum and the minimum number of clusters. Snowflake will start all the warehouses so that the maximum resources are available while they run.
- Auto-Scale → We will enable this mode by specifying DIFFERENT values for the maximum and the minimum number of clusters. Snowflake will start and stop warehouses as needed. To control how the warehouse scales, we can define Scaling Policies.

### Scaling policy

- Standard policy → It prioritizes starting additional warehouses over conserving credits.
- Economy policy → A more restrictive policy that prioritizes conserving credits over starting additional warehouses.

### Auto Suspend & Auto Resume

- Snowflake can suspend the warehouse if it's inactive for a specific period of time. By default, the auto-suspend option is enabled. For example, if we don't use the warehouse for 10 minutes, it will automatically be suspended.

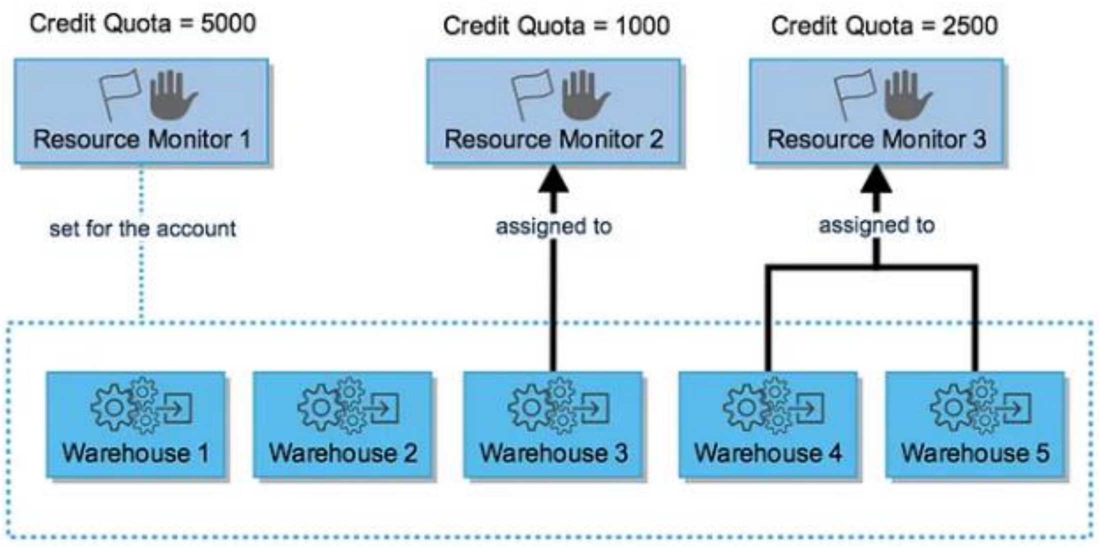
- Auto Resume allows Snowflake to automatically resume the warehouse when any statement requires the use of the warehouse, like any query or DML command. By default, it's also enabled.

# Eleventh Chapter: Resource Monitors

- We use Resource monitors to help control costs and avoid unexpected credit usage caused by running data warehouses.
- You can impose limits on the number of credits that warehouses consume.
- Resource Monitors are ONLY created by AccountAdmins, although roles with the MONITOR & MODIFY privileges can view and modify the resource monitor.

## Resource Monitors Assignment

- You can set a single monitor at the account level to control credit usage for all warehouses in your account.
- You can assign a monitor to one or more warehouses, controlling the credit usage for each warehouse.
- You can assign a warehouse to only a single resource monitor below the account level — this is a typical exam question.



## Resource Monitors Parameters

We need to specify several parameters when creating a Resource Monitor.

1) Credit Quota → Specifies the number of Snowflake credits allocated to the monitor for the specified frequency interval. A credit quota of 1000 would mean that the warehouse can consume until 1000 credits. It works by interval, and it will be restarted. For example, we can set the credit quota for an entire month, and the following month the credit quota would reset to 0.

2) Monitor Level→ It specifies whether the resource monitor is used to monitor the credit usage for your entire Account or individual WareHouses.

3) Schedule → When the monitor is going to start monitoring and when the credits reset back to 0. The credits are reset to 0 at the beginning of each calendar month.

4) Actions → They are triggers. Each action specifies a threshold and the action to perform when the threshold is reached within the specified interval. To receive notifications, each account administrator must explicitly enable notifications through the web interface, specifying their email if they want to receive them by email. Resource monitors support the following actions:

- Notify (send notification) → Perform no action but send an alert notification (email/web UI).
- Notify & Suspend (suspend warehouse) → Send a notification and suspend all assigned warehouses after all statements being executed by the warehouse (s) have been completed.
- Notify & Suspend Immediately (kill query) → Send a notification and suspend all assigned warehouses immediately.

# Twelfth Chapter: Cache and Query Performance

One of the most important concepts is the cache to improve the speed of our queries in Snowflake and optimize costs.

## METADATA CACHE

- Metadata caching is maintained in Global Service Layer and contains Objects Information & Statistics
- This metadata information lasts for 64 days
- This cache will help us perform operations like MIN, MAX, COUNT
- Snowflake will NOT use the warehouses in these cases, so we don't spend computing credits

## QUERY RESULT CACHE

- This cache stores the results of our queries for 24 hours
- As long as we perform the same query and the data hasn't changed in the Storage layer, it will return the same result without using the warehouse
- You cannot see the results from other people in the History tab, but Snowflake stores the result so that if different people (with the same role) perform the same query, they will also use this cache
- You can disable the Query Result cache with the following command:

```
ALTER SESSION SET USE_CACHED_RESULT = FALSE
```

## WAREHOUSE CACHE

- while the data warehouse runs, the table fetched in the query will remain. When the warehouse is suspended, the information will be lost.

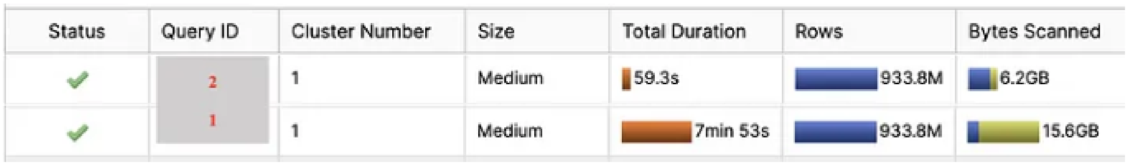
- **1st query**, we fetch all the columns from *MyTable:*

```
SELECT *
FROM ANALYTICS.PUBLIC.MYTABLE
ORDER BY DATE DESC
```

- **2nd query**, we fetch the columns “Date” and “Score” from *MyTable*:

```
SELECT DATE, SCORE
FROM ANALYTICS.PUBLIC.MYTABLE
ORDER BY DATE DESC
```

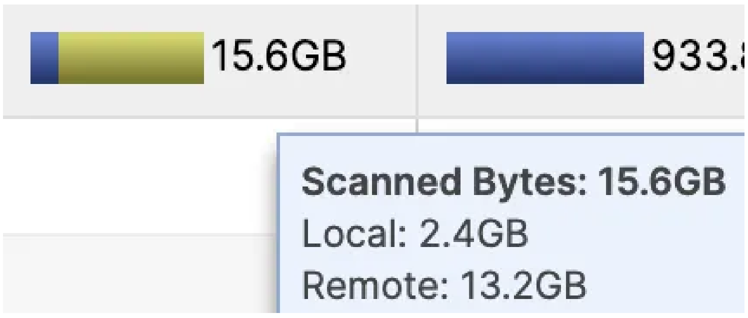
Why isn’t the second query re-used from the Query Result Cache? Because it’s NOT the same query, we are selecting fewer columns. But let’s take a look at the History tab:



| Status | Query ID | Cluster Number | Size | Total Duration | Rows | Bytes Scanned |
|---|---|---|---|---|---|---|
| ✓ | 2 | 1 | Medium | 59.3s | 933.8M | 6.2GB |
| ✓ | 1 | 1 | Medium | 7min 53s | 933.8M | 15.6GB |

Warehouse cache example

- **Bytes Scanned** → We can also see that the number of Bytes Scanned was 15.6GB, and there are two colors. The blue one means the data is re-used from the warehouse, whereas the yellow one is fetched from the Storage layer. Almost everything was taken from the Storage layer in the first query, whereas in the second case, almost everything was re-used from the warehouse. The bytes scanned size is also lower because fewer columns were fetched.



15.6GB

933.

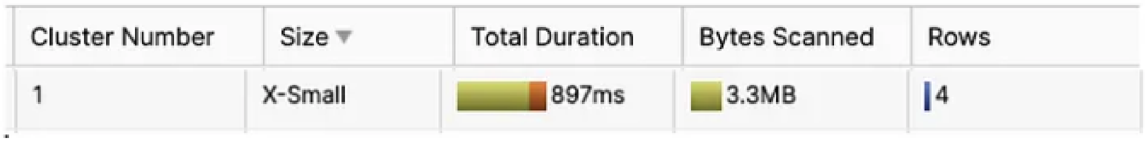
Scanned Bytes: 15.6GB
Local: 2.4GB
Remote: 13.2GB

Different Bytes Scanned values

## COMPLETE EXAMPLE WITH ALL THE SNOWFLAKE CACHE TYPES

- First query: There is no information in the cache, so everything comes from the Long Term Storage (Storage Layer). We can see that with the Bytes Scanned row, as it’s completely yellow.

```
SELECT *
FROM ANALYTICS.PUBLIC.MYTABLE
```

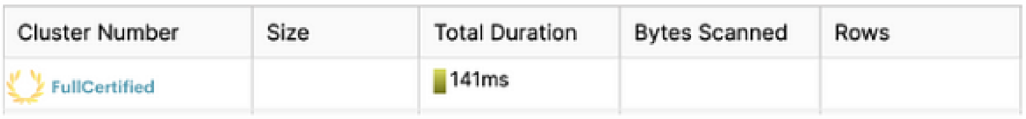


| Cluster Number | Size ▼ | Total Duration | Bytes Scanned | Rows |
|---|---|---|---|---|
| 1 | X-Small | 897ms | 3.3MB | 4 |

Example not using Snowflake cache

- Second query: We perform the same query. As less than 24 hours have been spent, this information is in the Query Result Cache, so we can see that we didn’t scan bytes, and the duration is eight times lower than before.

```
SELECT *
FROM ANALYTICS.PUBLIC.MYTABLE
```

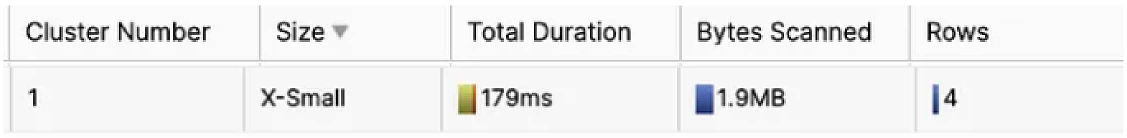


| Cluster Number | Size | Total Duration | Bytes Scanned | Rows |
|---|---|---|---|---|
| FullCertified |  | 141ms |  |  |

Example of Query Result Cache in Snowflake

- Third query: We access the same table but do not run the same query. We need the column "MADEBY". We will not be able to fetch the information from the Query Result Cache, so we will fetch it from the Warehouse Cache. For that reason, the Bytes Scanned parameter is not empty like before.

```
SELECT MADEBY
FROM ANALYTICS.PUBLIC.MYTABLE
```

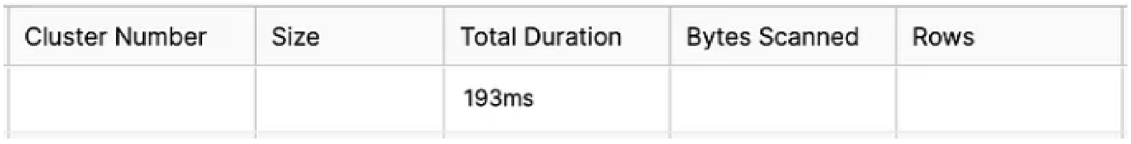


| Cluster Number | Size | Total Duration | Bytes Scanned | Rows |
|---|---|---|---|---|
| 1 | X-Small | 179ms | 1.9MB | 4 |

Warehouse Cache example in Snowflake

- Fourth query: We want to know the number of rows in "MYTABLE". This information is stored in the Metadata Caché, so we won't use any warehouse; having a tiny duration, and no Bytes Scanned.

```sql
SELECT COUNT(*)
FROM ANALYTICS.PUBLIC.MYTABLE
```



| Cluster Number | Size | Total Duration | Bytes Scanned | Rows |
|---|---|---|---|---|
|  |  | 193ms |  |  |

Metadata Cache Example

## HOW TO IMPROVE SNOWFLAKE PERFORMANCE

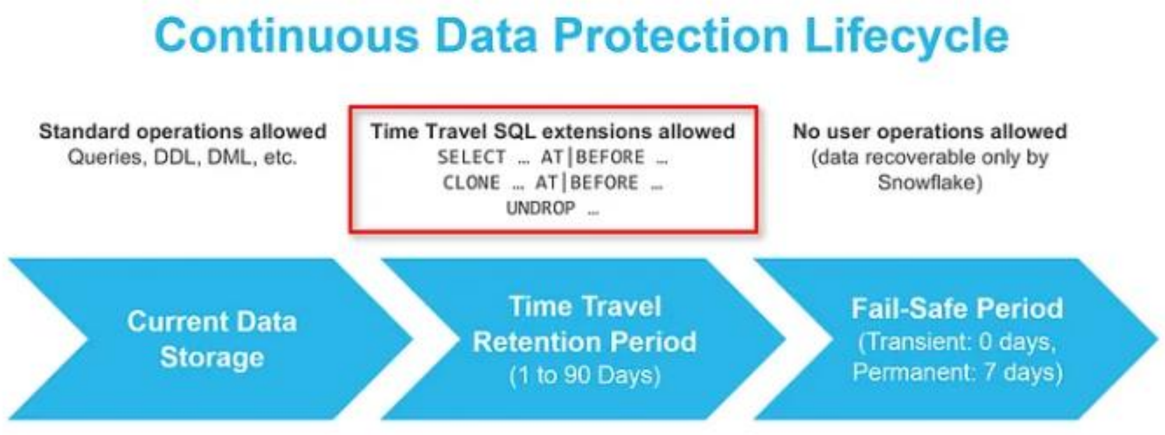
- Use dedicated Virtual Warehouses → It's good to have a Virtual Warehouse for each type of task. For example, a Virtual Warehouse for BI tasks, another for Data Science… The results can be re-used easily as the users will perform similar queries.
- Scale UP/DOWN for workloads that are known → If we know that Mondays at 10 a.m., the number of queries increases by x2, or we need to do a report that requires a lot of Snowflake power, we should scale up.
- Multi-Warehouses for unknown workloads → Sometimes, the number of users increases without knowing that. For that reason, we can set up multi-warehouses.
- Try to maximize the use of the cache.
- Cluster keys → Use them in big tables to improve their performance, especially in columns that you use to filter (WHERE, JOINS…).

# Thirteenth Chapter: Time Travel, Fail-Safe & Zero-Copy Cloning

## INTRODUCTION TO SNOWFLAKE STORAGE FEATURES

Have you ever seen the meme (I hope it hasn't happened to you) where you start working at 9 a.m. on a Monday, and you delete a table in a production environment? Well, this will no longer be a problem, thanks to Snowflake!

In the following diagram, we can see different ways to recover data in Snowflake, first by ourselves using the Time Travel functionality, then by Snowflake support using the Fail-Safe retention period.


Continuous Data Protection Lifecycle
Standard operations allowed
Queries, DDL, DML, etc.
Time Travel SQL extensions allowed
SELECT … AT|BEFORE …
CLONE … AT|BEFORE …
UNDROP …
No user operations allowed
(data recoverable only by Snowflake)
Current Data Storage
Time Travel Retention Period
(1 to 90 Days)
Fail-Safe Period
(Transient: 0 days, Permanent: 7 days)

Recovering data using Snowflake (via docs.snowflake.com).

## TIME TRAVEL

- Time travel enables accessing historical data (i.e., data that has been changed or deleted) at any point within a defined period.
- If we drop a table, we can restore it with time travel. You can use it with Databases, Schemas & Tables.
- No tasks are required to enable Time Travel.
- By default, it's enabled with a 1-day retention period. However, we can increase it to 90 days if we have (at least) the Snowflake Enterprise Edition.
- It requires additional storage, which will be reflected in your monthly storage charges.

Use cases of Time Travel:

- Restoring data-related objects that we might have accidentally or intentionally deleted.
- Duplicating and backing up data from key points in the past.
- Query data from the past.

To restore objects, we use the command "UNDROP".

```
DROP TABLE mytable; -- We accidentally drop a table

UNDROP TABLE mytable; -- We restore it using Time Travel
```

## QUERYING OVER HISTORICAL DATA

Imagine that a table has changed a lot during the last week, and we want to query how it was seven days ago

- **By timestamp**

```
SELECT *
FROM my_table
AT(timestamp => 'Mon, 01 May 2021 08:00:00 -0700'::timestamp_tz);
```

- **By offset** → In this example, we select the historical data from a table as of 15 minutes ago:

```
SELECT *
FROM my_table
AT(offset => -60*15);
```

- **By query statement ID** → We can see the Query Statement ID in the history of queries:

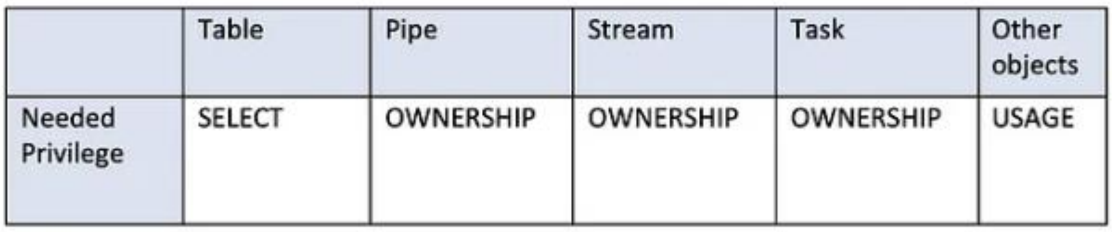
```
SELECT *
FROM my_table
BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

## FAIL-SAFE

- Fail-safe ensures historical data is protected in the event of a system failure or other catastrophic event, e.g., a hardware failure or security breach.
- It provides a (NON-CONFIGURABLE) 7-day period during which Snowflake support may recover historical data.
- You cannot recover this data alone; you must ask Snowflake support.
- This period starts immediately after the Time Travel retention period ends, and it also requires additional storage as Time Travel
- Only permanent tables have a Fail-Safe period

## ZERO-COPY CLONING

- Using Zero-Copy cloning, you can create a snapshot of any table, schema, or Database.
- The cloned object is independent and can be modified without modifying the original.
- Very good for development jobs.
- Zero-Copy cloning does NOT duplicate data; it duplicates the metadata of the micro-partitions.
- For this reason, Zero-Copy cloning is FREE, as it doesn't consume storage.
- Privileges are not cloned.
- Named Internal Stages are not cloned, but external ones are cloned. Table Stages are also cloned.
- Pipes that reference the internal stages are not cloned. If they reference an external stage, they are cloned.
- Data History is not cloned.



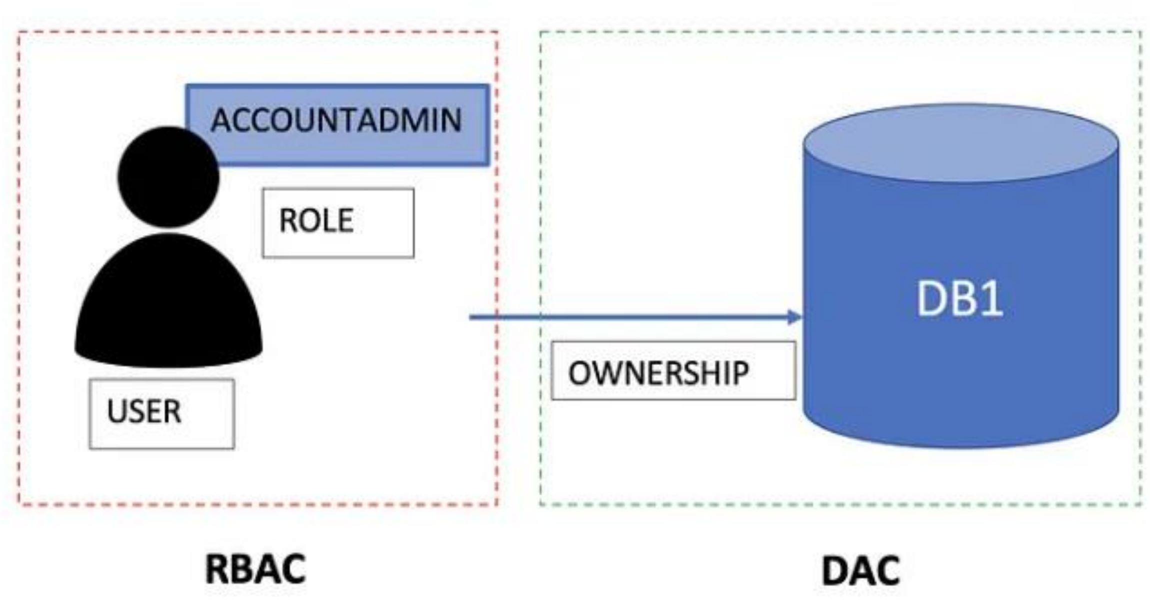
| | Table | Pipe | Stream | Task | Other objects |
|---|---|---|---|---|---|
| Needed Privilege | SELECT | OWNERSHIP | OWNERSHIP | OWNERSHIP | USAGE |

Required Privileges for Zero-Copy Clone.

# Fourteenth Chapter: Access Management. Roles in Snowflake

## ACCESS MANAGEMENT KEY CONCEPTS

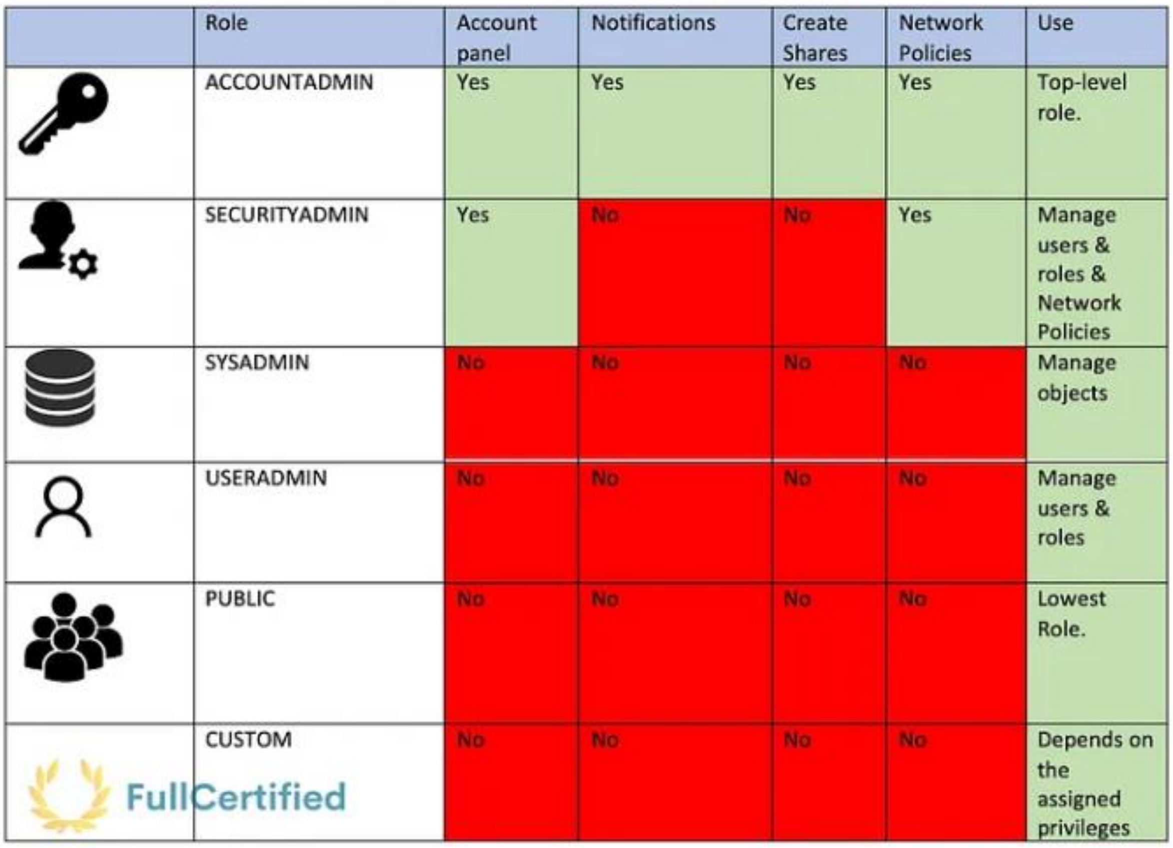
- Access control privileges determine who can access and perform operations on specific objects in Snowflake.
- Discretionary Access Control (DAC) → Each object has an owner who can, in turn, grant access to that object.
- Role-Based Access Control (RBAC) → Access privileges are assigned to roles, which are, in turn, given to users.



- User → A person or a program. For example, Gonzalo, as a user.
- Role → An entity to which we can grant privileges. We can grant roles to users. Roles are account-level objects and can be granted to other roles, creating a hierarchy. The privileges associated with a role are inherited by any roles above that role in the hierarchy. For example, Gonzalo will be associated with the AccountAdmin role.
- Securable object → An entity to which we can grant access. Unless allowed by a privilege, access will be denied. For example, a table, database, or schema are securable objects.
- Privilege → A defined level of access to an object. For example, the user Gonzalo, as AccountAdmin, will have the SELECT privilege on database “DB1”.
- It’s important to remember that privileges are assigned to Roles, which will be assigned to users. USER <- ROLE <- PRIVILEGE.

## DEFAULT ROLES

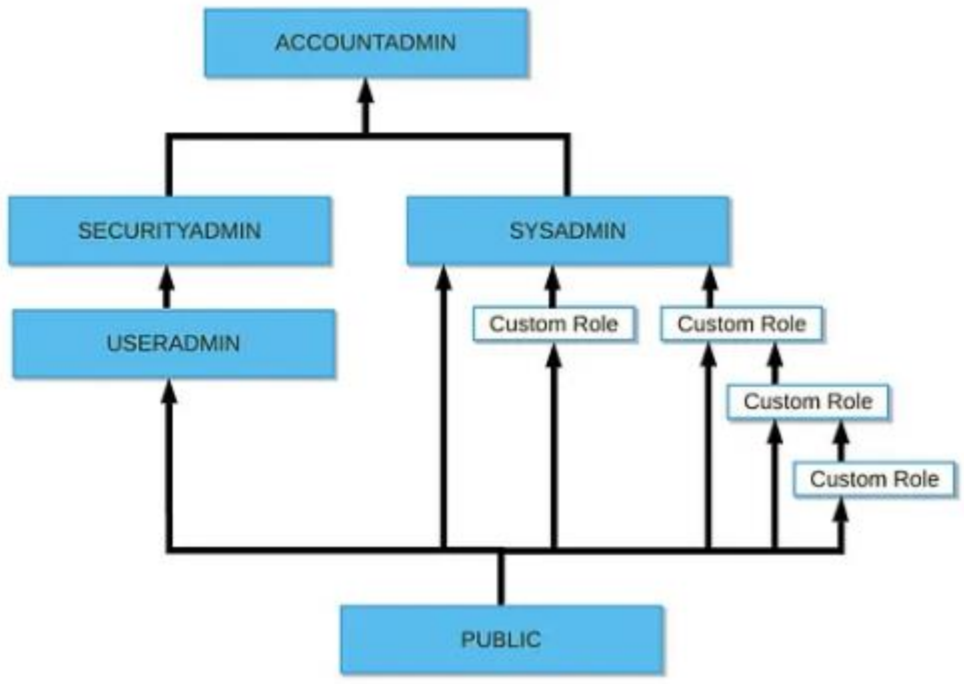
- Snowflake gives five different default roles plus the possibility to create custom ones

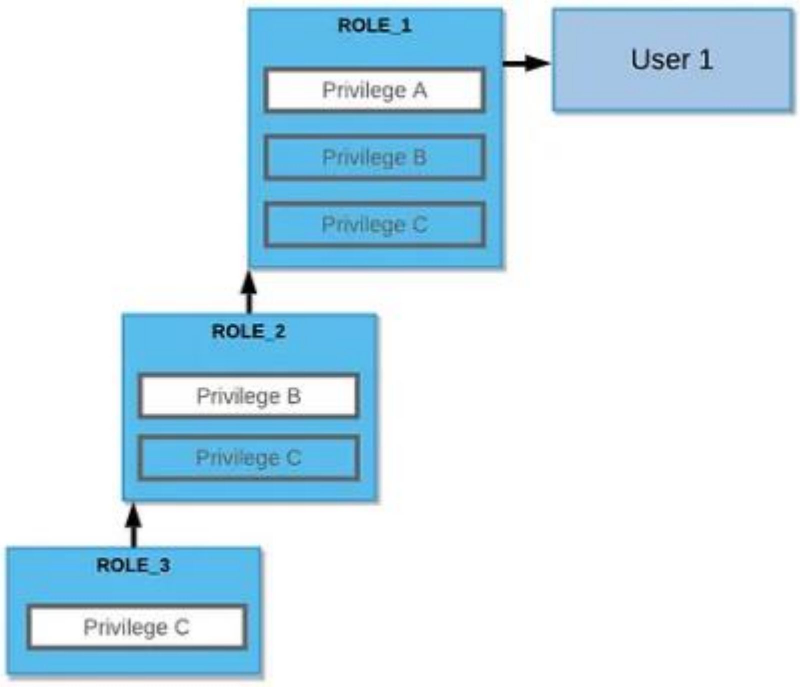


| | Role | Account panel | Notifications | Create Shares | Network Policies | Use |
|---|---|---|---|---|---|---|
| | ACCOUNTADMIN | Yes | Yes | Yes | Yes | Top-level role. |
| | SECURITYADMIN | Yes | No | No | Yes | Manage users & roles & Network Policies |
| | SYSADMIN | No | No | No | No | Manage objects |
| | USERADMIN | No | No | No | No | Manage users & roles |
| | PUBLIC | No | No | No | No | Lowest Role. |
| | CUSTOM | No | No | No | No | Depends on the assigned privileges |

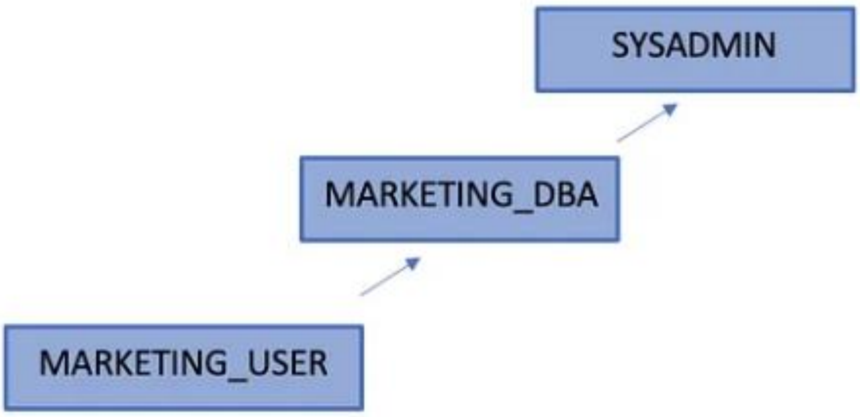
- ACCOUNTADMIN → It encapsulates the SYSADMIN and SECURITYADMIN system-defined roles. It's the top-level role, and you shouldn't give it to many users. AccountAdmin users can access the Account and Notification sections in the Snowflake UI. They can also CREATE, ALTER, or DROP Network Policies and Shares (we will see this concept soon).
- SECURITYADMIN → It can manage users and roles. SecurityAdmins can access the Account tab in the interface, but they cannot see the Usage & Billing part. They cannot either create Reader sharing accounts or see the Notifications section. Users with this role can also CREATE, ALTER, or DROP Network Policies.
- SYSADMIN → User with the SysAdmin role can create warehouses and databases (and other objects) in an account.
- USERADMIN → This role is dedicated to user and role management only. It's like an admin role with fewer privileges than the SECURITYADMIN one.
- PUBLIC → This role is automatically granted to every user and every role in your account. It can own its objects, but they'll be available for everybody as it is the lowest role in the Snowflake hierarchy.
- CUSTOM ROLES → They can be created by the USERADMIN (or a higher role) and by any role to which the CREATE ROLE privilege has been granted. You can assign the privileges that you want. We will see an example later.

## ROLES ENCAPSULATION





Roles encapsulation in Snowflake (via docs.snowflake.com)



Expected hierarchy to create

```
CREATE ROLE MARKETING_USER;

GRANT ROLE MARKETING_USER TO ROLE MARKETING_DBA;
GRANT ROLE MARKETING_DBA TO ROLE SYSADMIN;

CREATE USER plazagonzalo PASSWORD = 'Gonzalo123' DEFAULT_ROLE = MARKETING_USER
    MUST_CHANGE_PASSWORD = TRUE;

GRANT ROLE MARKETING_USER TO USER plazagonzalo;
```

# Fifteenth Chapter: Data Sharing

## INTRODUCTION TO SECURE DATA SHARING

- Secure Data Sharing enables sharing selected objects in a database in your account with other Snowflake accounts.
- The account that receives the data cannot modify it, as shared data is always read-only.

These are the Snowflake objects that we can share:

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

## SHARES

- Shares are named Snowflake objects that encapsulate all information required to share a database.
- Each share consists of the following:
    - The privileges that grant access to the database(s) and the schema containing the objects to share.
    - The privileges grant access to the specific objects we want to share.
    - The consumer accounts with which the database and its objects are shared.

Imagine that we want to share a table “myTable” contained in the “myDb” database. We should create a share and give the needed privileges to the share to access this table:

```
CREATE SHARE myShare;

GRANT USAGE ON DATABASE myDb TO SHARE myShare;

GRANT USAGE ON SCHEMA myDb.public TO SHARE myShare;

GRANT SELECT ON TABLE myDb.public.myTable TO SHARE myShare;
```

PRODUCERS & CONSUMERS

- **Show all the shares that we have in the system:**

```
SHOW SHARES
```

- **See all the privileges that a share has:**

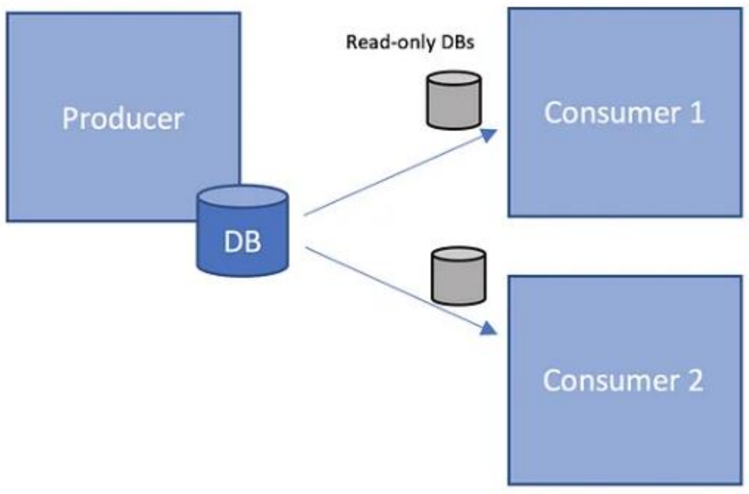
```
SHOW GRANTS TO SHARE myShare
```

- **See the accounts (consumers) that are using the share:**

```
SHOW GRANTS OF SHARE myShare
```

## PRODUCERS & CONSUMERS

- In Snowflake, the account that shares a Share will be the producer, while the account that receives it will be the consumer (already mentioned before).



Producers and consumers in Snowflake

- Producers (or providers) are called to the Snowflake account that creates shares and makes them available to other Snowflake accounts to consume.
- The producers will always pay for the storage of the data they share.
- Consumers don’t pay for storage, as the producer account already pays for it
- Shared data is instantaneous for consumers as no actual data is copied or transferred between accounts
- There are several features that a consumer cannot do with a Shared Object, for example, create a clone or Time-Travel in tables.
- Once the consumers receive the share, they have to create a database from this share, and at this point, all the shared objects will be accessible to users in the consumer account.
- The consumer and producer accounts must be in the same region to share data.

We have two types of Consumers:

- Full account → When you share something with an existing Snowflake account. The consumer account pays for the queries they make
- Reader Account → Imagine that you want to share data with someone that doesn’t have a Snowflake account. The producer account pays all the compute credits that their warehouses use.

## INBOUND & OUTBOUND SHARES

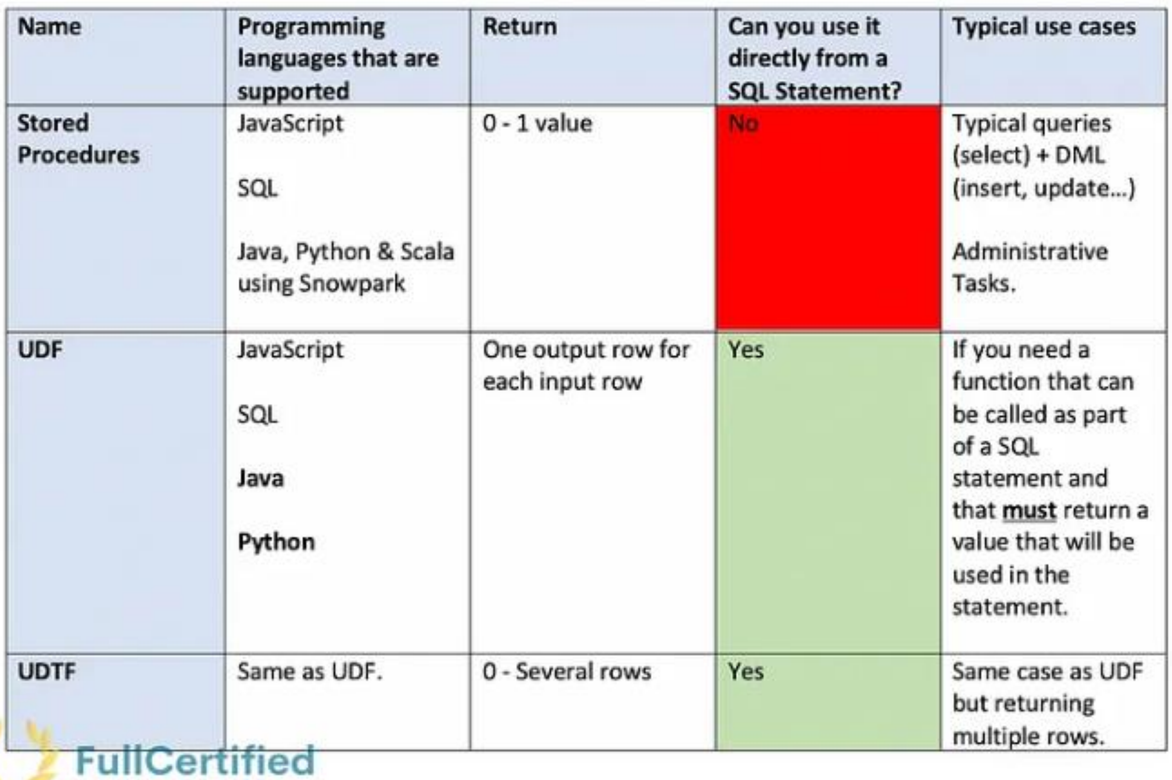
We will have two types of shares:

- Outbound shares → Shares that you (as a producer) have created to share with other accounts (consumer accounts).
- Inbound shares → Shares that other accounts (as producers) have created and shared with you (as a consumer). These are the available shares for your account to consume.

# Sixteenth Chapter: Stored Procedures & User-Defined Functions for the SnowPro Core Certification

## INTRODUCTION

- You can use JavaScript, SQL, Java, and Python to extend Snowflake functionality.



| Name | Programming languages that are supported | Return | Can you use it directly from a SQL Statement? | Typical use cases |
|---|---|---|---|---|
| **Stored Procedures** | JavaScript<br><br>SQL<br><br>Java, Python & Scala using Snowpark | 0 - 1 value | No | Typical queries (select) + DML (insert, update...)<br><br>Administrative Tasks. |
| **UDF** | JavaScript<br><br>SQL<br><br>**Java**<br><br>**Python** | One output row for each input row | Yes | If you need a function that can be called as part of a SQL statement and that **must** return a value that will be used in the statement. |
| **UDTF** | Same as UDF. | 0 - Several rows | Yes | Same case as UDF but returning multiple rows. |

## STORE PROCEDURES

- You can include programming constructs such as branching and looping
- Using the Snowpark library, you can also write them in Python, Java or Scala
- They return either a SINGLE Value or nothing
- The returned values CANNOT be used directly in a SQL statement.

## USER-DEFINED FUNCTIONS (UDFs)

- User-defined functions (UDFs) let you extend the system to perform operations that are not available through Snowflake's built-in, system-defined functions

The difference with Store procedures is that:

- It returns one output row for each input row. The returned row consists of a single column/value.
- It must return something.
- The returned values CAN be used directly in statement SQL

```
---- Function definition ----
create or replace function add5 (n number)
  returns number
  as 'n + 5';

---- Calling the function ----
SELECT add5(1)

---- Result ----
+----------+
| ADD5(1) |
|---------|
|       6 |
+----------+
```

## USER-DEFINED TABLE FUNCTIONS (UDTFs)

- UDTFs can return multiple rows for each input row; that’s the only difference with UDFs.

```
---- Function definition ----
create function t()
    returns table(msg varchar)
    as
    $$
        select 'Hello'
    $$;

---- Calling the function ----
select msg
    from table(t())
    order by msg;

---- Result ----
+--------+
| MSG   |
|-------|
| Hello |
| World |
+--------+
```

# Seventeenth Chapter: Tasks & Transactions in Snowflake

## INTRODUCTION TO TASKS

- Snowflake tasks are schedulable scripts that are run inside your Snowflake environment
- No event source can trigger a task; instead, a task runs on a schedule
- A task can execute a single SQL statement, including a call to a Stored Procedure
- Snowflake ensures only one instance of a task with a schedule is executed at a given time
- If a task is still running when the next scheduled execution time occurs, that scheduled time is skipped
- Tasks also have a maximum duration of 60 minutes by default. If they haven't finished by then, they will be automatically terminated.
- When you first create a task, it will be suspended by default. You can activate it with the following command:

```
ALTER TASK mytask RESUME;
```

## TREE OF TASKS

- Users can define a simple tree-like structure of tasks that starts with a root task and is linked together by task dependencies
- The Root task will be the only one to which the scheduler is set. The children's tasks only run after the parent's task finishes
- Each task can have a maximum of 100 children tasks.
- A tree of tasks can have a maximum of 1000 tasks, including the root one.
- We can create child tasks with the following commands:

```
CREATE TASK <newTask> AFTER <rootTask>;

ALTER TASK <newTask> ADD AFTER <rootTask>;
```

What would happen if we eliminated the predecessor tasks. In this case, there are two options:

- Child task becomes a standalone task.
- Child task becomes a root task.

And what if the owner role of a task is deleted? Task Ownership is reassigned to the role that dropped this role.

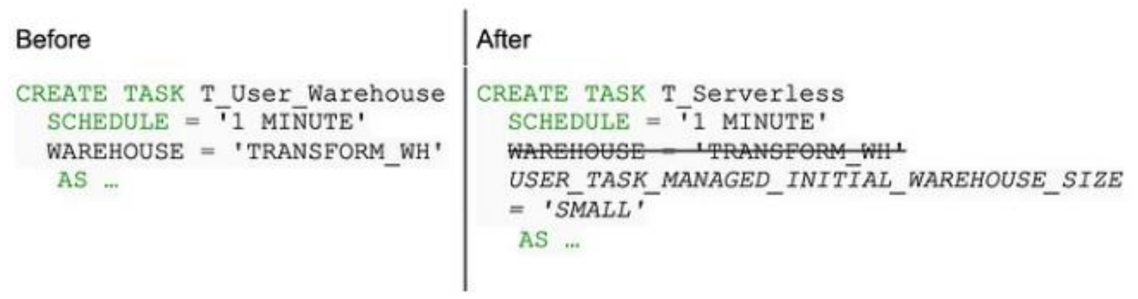
You can use this Snowflake function to query the history of task usage within a specified date range and find it in the information schema. You need one of these privileges to see the task history:

- AccountAdmin role.
- You are the owner of a task.
- You have the global MONITOR_EXECUTION privilege.

```
SELECT *
FROM table(information_schema.task_history())
ORDER BY scheduled_time;
```

## SERVERLESS TASKS

Imagine that you have a task that runs every 5 minutes in a warehouse with auto-suspend mode enabled after 10 minutes. This would mean that the warehouse won't EVER suspend! The serverless compute model for tasks enables you to rely on compute resources managed by Snowflake instead of user-managed virtual warehouses.


Before

```
CREATE TASK T_User_Warehouse
  SCHEDULE = '1 MINUTE'
  WAREHOUSE = 'TRANSFORM_WH'
   AS ...
```

After

```
CREATE TASK T_Serverless
  SCHEDULE = '1 MINUTE'
  ~~WAREHOUSE = 'TRANSFORM_WH'~~
  USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE
  = 'SMALL'
   AS ...
```

How to create Snowflake serverless tasks.

## TRANSACTIONS

- A transaction is a sequence of SQL statements that are committed or rolled back as a unit.
- Snowflake takes 4 hours to abort it if we do not abort it with the SYSTEM$ABORT_TRANSACTION function if the session is disconnected for whatever reason and the transaction remains in a detached state.
- Each transaction has independent scope.
- Snowflake does not support Nested Transactions, although it supports Nested Procedure Calls.

# Eighteenth Chapter of the Snowflake SnowPro Core Course: Streams

## INTRODUCTION TO STREAMS

- Streams are Snowflake objects that record data manipulation language (DML) changes made to tables*, including INSERTS, UPDATES, and DELETES, as well as metadata about each change.
- Streams don't contain table data; they only store offsets
- We have three different types of Streams:
  - Standard → Tracks all DML changes to the source table, including inserts, updates, and deletes. Supported on tables, directory tables, and views.
  - Append Only → Tracks row inserts only. Supported on tables, directory tables, and views.
  - Insert Only → Tracks row inserts only. The difference with the previous one is that this one is only supported on EXTERNAL TABLES.

Apart from that, each stream contains the following columns:

- METADATA$ACTION → Indicates the DML operation (INSERT, DELETE) recorded.
- METADATA$ISUPDATE → Indicates whether the operation was part of an UPDATE statement.
- METADATA$ROW_ID → Unique and immutable ID for the row.

First example: Create a stream from a table, add a new row and update the new row.

1) Create a Stream from a table → The stream will be empty; it doesn't have any row.

```
+--------+---------+-----------+---+
| ROW_ID |  ACTION |  ISUPDATE |   |
+--------+---------+-----------+---+
|        |         |           |   |
+--------+---------+-----------+---+
```

2) Add a row in the table → As a new DML operation (INSERT) is done, Snowflake will add a new entry in the stream. In this case, the action is "INSERT".

```
+--------+---------+-----------+
| ROW_ID |  ACTION |  ISUPDATE |
+--------+---------+-----------+
| #1234  |  INSERT |  FALSE    |
+--------+---------+-----------+
```

3) Update the row in the table → The stream will remain like in the previous case. Why is ISUPDATE false if the row has been updated? We need to see how the stream was at the beginning. When we first created it, the row didn't exist, so from the point of view of the stream, it's just a new row, not an updated one, since it didn't exist at the beginning.

```
+---------+---------+-----------+
| ROW_ID |  ACTION |  ISUPDATE |
+---------+---------+-----------+
| #1234   |  INSERT |  FALSE     |
+---------+---------+-----------+
```

Second example: Add a new row in the table, create a stream from a table, and then modify the stream.

1) Add a row in the table → No stream has been created.

2) Create the stream

```
+---------+---------+-----------+--+
| ROW_ID |  ACTION |  ISUPDATE |  |
+---------+---------+-----------+--+
|         |         |           |  |
+---------+---------+-----------+--+
```

3) Update the row in the table → In this case, the row already existed. If we modify it, it's not a new row; it's just a modification from the table before creating the stream, so ISUPDATE will be true in this case.

```
+---------+---------+-----------+
| ROW_ID |  ACTION |  ISUPDATE |
+---------+---------+-----------+
| #1234   |  INSERT |  TRUE      |
+---------+---------+-----------+
```

Third example: Create a stream from a table, add a new row and delete the same row.

1) Create a Stream from a table → The stream will be empty; it doesn't have any row.

```
+---------+---------+-----------+--+
| ROW_ID |  ACTION |  ISUPDATE |  |
+---------+---------+-----------+--+
|         |         |           |  |
+---------+---------+-----------+--+
```

2) Add a row in the table → As a new DML operation is done, Snowflake will add a new entry in the stream. In this case, the action is "INSERT".

```
+--------+---------+-----------+
| ROW_ID |  ACTION |  ISUPDATE |
+--------+---------+-----------+
| #1234  |  INSERT |  FALSE    |
+--------+---------+-----------+
```

3) Delete the row in the table→ The stream won't contain any information. Comparing this to the state of the table before creating the stream, it will be the same. You just add a new row that you delete, so it will be exactly the same table state. So the stream will be empty. This is important to understand.

```
+--------+---------+-----------+--+
| ROW_ID |  ACTION |  ISUPDATE |  |
+--------+---------+-----------+--+
|        |         |           |  |
+--------+---------+-----------+--+
```

Fourth example: Add a new row in the table, create the stream and delete the same row.

1) Add a row to the table→ The stream has not been created yet.

2) Create a stream from the table.

```
+--------+---------+-----------+--+
| ROW_ID |  ACTION |  ISUPDATE |  |
+--------+---------+-----------+--+
|        |         |           |  |
+--------+---------+-----------+--+
```

3) Delete the row in the table → In this case, there will be a difference between the moment you created the stream and this moment because the table will contain one less row. That's why the stream will contain this information.

```
+--------+---------+-----------+
| ROW_ID |  ACTION |  ISUPDATE |
+--------+---------+-----------+
| #1234  |  DELETE |  FALSE    |
+--------+---------+-----------+
```

All the examples show that the stream only contains the last DML action on a row.

# Chapter 19: Other Snowflake Objects. File Formats & Sequences

Snowflake supports both Structured and Semi-Structured Data, so just as an example, you can store JSON files in tables. Let's see the differences between them:

## Structured Data

CSV → You can Load and Unload files in CSV format. It's the fastest file format to load data.

## Semi-structured Data

Semi-structured data is saved as Variant type in Snowflake tables, with a maximum limit size of 16MB, and it can be queried using JSON notation.

JSON → It's used for both loading & unloading data.

Parquet → Binary format used for both loading & unloading data.

XML → You can only load data in Snowflake using the XML format.

Avro → Binary format used to load data.

ORC → Binary format used to load data.

You can use FLATTEN to convert semi-structured data to a relational representation. It takes a Variant, Object, or Array column and produces a lateral view.

## SEQUENCES

We use sequences to generate unique numbers across sessions and statements, including concurrent statements. You can use them to generate values for a primary key or any column that requires a unique value. They have an initial value and an interval.

You can access sequences in queries as expressions. The function "nextval", will generate a set of distinct values.

```
INSERT INTO PEOPLE (ID, NAME) VALUES
    (PEOPLE_SEQ.nextval, "Gonzalo"),
    (PEOPLE_SEQ.nextval, "Nacho"),
    (PEOPLE_SEQ.nextval, "Megan"),
    (PEOPLE_SEQ.nextval, "Angel")
```

We can also use the "Default" statement when creating the tables, increasing the sequence automatically.

```
CREATE OR REPLACE TABLE PEOPLE
(
  ID NUMBER DEFAULT PEOPLE_SEQ.nextval,
  NAME VARCHAR(50)
)

-------------

INSERT INTO PEOPLE (NAME) VALUES
  ("Gonzalo"),
  ("Nacho"),
  ("Megan"),
  ("Angel")
```

# Chapter 20: Snowflake Ecosystem, Compliance, Data Marketplace & Security

## SNOWFLAKE ECOSYSTEM

Partner Connect → Lets you easily create trial accounts with selected Snowflake business partners and integrate them with Snowflake to try various 3rd-party tools and services. There are two types of partners:

- Technology partners → They integrate their solutions with Snowflake to get data quickly into Snowflake. They offer Software, driver, interfaces…
- Solution partners → Trusted and validated experts and services. They are like consulting partners.

These partners are classified into six different categories, as shown in the previous image:

- Data Integration
- ML & Data Science
- Security & Governance
- Business Intelligence
- SQL Editors
- Programming Interfaces

## SNOWFLAKE COMPLIANCE

This is the list of reports available on Snowflake. You can get more information about them at the following link. I've pointed out the most important ones, although I don't think you will be asked in-depth questions about them in the exam. Just remember the names.

- HITRUST / HIPAA
- ISO/IEC 27001
- FedRAMP Moderate
- PCI-DSS
- SOC 2 Type II
- SOC 1 Type II
- GxP

## SNOWFLAKE DATA MARKETPLACE

In the data marketplace, you can buy or sell data provided by providers who share their datasets.

There are three types of listings available in the Snowflake Data Marketplace:

- Free Listing → Also known as Standard Listing. It's the best for providing generic, aggregated, or non-customer-specific data.
- Personalized Listing → Premium data. You can request specific datasets from providers.
- Paid Listing → As a provider, you can charge consumers to access or use your listing.

There is also another type of listing, the Private Listing. However, it's not available in the Data Marketplace. With them, you can use listings to share data and other information directly with another Snowflake account.

As we mentioned, we can also upload our datasets to the Snowflake Marketplace. To do that, our dataset must meet these conditions:

- Fresh data
- Non-static data
- Real data
- Compliant data
- Legally Distributable data

## COLUMN LEVEL SECURITY

Snowflake provides the following column-level security features for users with at least Enterprise Edition:

- Dynamic Data Masking → Column-level Security feature that uses masking policies to mask data at query time. You can hide some columns for users with a lower role.
- External Tokenization → You can tokenize sensitive data before loading it into Snowflake. You can also detokenize it using masking policies. It's useful for data like passwords or any sensitive data.