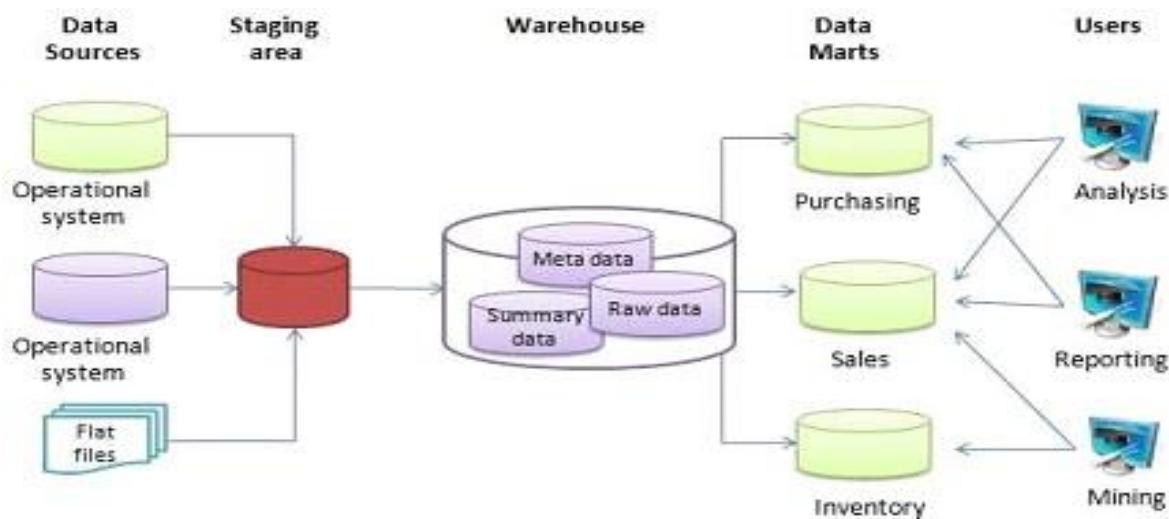


DATA WAREHOUSE

A data warehouse is a type of data management system that is designed to enable and support business intelligence (BI) activities, especially analytics. Data warehouses are solely intended to perform queries and analysis and often contain large amounts of historical data.



A data warehouse is a large and centralized repository of data that is used for storing, managing, and analyzing data from different sources within an organization. The data in a data warehouse is typically collected from various sources such as operational systems, transactional databases, and external sources, and then transformed and loaded into the data warehouse.

The primary purpose of a data warehouse is to support business intelligence (BI) activities, such as data analysis, reporting, and decision-making. By centralizing data from different sources, a data warehouse allows organizations to analyze and understand their business operations more effectively, identify patterns and trends, and make informed decisions.

Data warehouses are designed to handle large amounts of data and complex queries. They typically use specialized technologies such as data modeling, extraction, transformation, and loading (ETL) tools, and OLAP (Online Analytical Processing) systems to process and analyze data.

Data warehouses can be built on-premises or in the cloud, depending on the organization's needs and resources. Cloud-based data warehouses offer several advantages over traditional on-premises solutions, including scalability, cost-effectiveness, and ease of use.

Problems with Traditional Data Warehouses :

- Scalability
- Meant for OLTP Processing (Row oriented & Normalized)
- No Distributed Processing
- No Concurrency
- Single Point of failure

Cloud Data Warehouses :

- On-Demand Scalability
- OLAP (Columnar databases)
- Concurrency and distributed work loads
- No Single point of failure
- Zero Overhead and maintenance

I would call Athena a better managed version of Hive because they both have their own limitations, especially when it comes to handling the file system, manipulating the data on your file system on which the tables are mounted. DML, DDL operations, so on and so forth. But I wouldn't go as far as calling them a complete cloud data warehouse because they come with a lot of pain points. Building a data warehouse by our data lake solution on cloud can essentially be split into two verticals around. The first one is your data warehouse management, it also takes understanding of how the data is set up in your data warehouse in the form of tables, schemas and databases.

Traditional Solutions for Data Warehouse :

- SQL Server
- PostgreSQL
- MySQL

Cloud Data Warehouses :

- BigQuery (Google Cloud)
- Redshift (Amazon Cloud)
- Snowflake

Other Options :

- AWS Athena
- Hadoop Hive

You have to know how to optimize the skill queries, which means you've got to be strong in writing as well. You have to ensure the compute resources are being utilized in the most optimal manner so as to optimize your cloud cost. You have to know user and access management who has access to what sort of data.

You also have to make sure of the readiness of the data so as to enable better boarding and analytics. Finally, you have your data governance and security.

So the first one to build requires one to have a good understanding of the data warehouse solution which is being used. The second vertical is where your cloud provider outside of your data warehouse comes into play, like Google or Amazon Cloud.

1 - Data-warehouse management (Strong SQL & Data Modelling Skills):

- Data Modelling
- Optimizing Queries
- Billing/Cost management
- User / Access Management
- Enabling BI/Analytics for decision making
- Data Governance and Security

2 - ETL/Data Movement (Strong Scripting/Coding skills along with understanding of cloud components)

- Writing Pipelines using scala/java/python
- Data orchestration tools (Airflow,Step Functions)
- Deployment and integration of analytics/ML Models
- Know-how of best solutions for deploying scalable pipelines
- Logging , alerting and notifications

This all mainly deals with ETL movement of data, data, pipeline, orchestration and deployment for which you need someone with strong scripting skills along with the knowhow of different cloud components. You got to know how to orchestrate your pipelines using an orchestration like, let's say, air flow or state functions.

You have to know when to use spark as opposed to just writing simple python or scalar jobs and how and where to deploy them. Like you have AWG glue, EMR Sage Maker and forecasting. Finally you have your logging stack stacked with alerting and notifications, which requires knowing the respective tools as well.

So in general, when you build a pipeline on cloud, you have to make sure of two aspects. One, they are optimized in terms of performance. Do they optimized in terms of cost as well? Because on cloud there are so many tools which can possibly do the same thing. But for one to be able to decide which the best solution is, you have to have a basic understanding first.

What is a Data Warehouse?

Businesses today are full of data. The amount of data produced daily is truly amazing. With Data Explosion, it becomes increasingly difficult to capture, process, and store large or complex data sets. Therefore, it becomes necessary for organizations to have a Central Archive where all data is stored securely and can be re-analysed to make informed decisions. This is where Data Warehouses comes into play.

The Data Warehouse, also known as the “One Source of Truth”, is a Central Database that supports the activities of Data Analytics and Business Intelligence (BI). Data Warehouses stores large amounts of data from multiple sources in one place and is intended for questioning and analysis to improve their business. Its analytical power allows organizations to obtain important business information from their data in order to improve decision-making.

What is the Snowflake Data Warehouse?

Snowflake is a cloud-based Data Warehouse solution provided as SaaS (Software-as-a-Service) with full ANSI SQL support. It also has a unique structure that allows users to simply create tables and start query data with very little management or DBA tasks required. Find out about Snowflake prices here.

Features of Snowflake Data Warehouse

Let's talk about some of the great features of Snowflake data warehouse:

1. **Data Protection and Protection:** Snowflake data repository provides advanced authentication by providing Multi-Factor Authentication (MFA), federal authentication and Single Login (SSO) and Oauth. Communication between client and server is secured by TLS.
2. **Standard and Extended SQL Support:** Snowflake data repository supports multiple DDL and SQL DML commands. It also supports advanced DML, transactions, lateral views, saved processes, etc.
3. **Connectivity:** Snowflake Database supports a comprehensive set of client and driver connectors such as Python connector, Spark connector, Node.js driver, .NET driver, etc.
4. **Data Sharing:** You can securely share your data with other Snowflake accounts.

Learn more about the features of Snowflake data warehouse here. Let's learn more about Snowflake buildings.

1. **The partner connect program** - Unlike Big Query and Redshift, Snowflake does not have other native tools for specialised services like machine learning and business intelligence that are built in. Instead, Snowflake works with a wide range of industry-leading technology partners and programmatic interfaces to build connectors and drivers for a bigger analytics ecosystem. This way customers can leverage Snowflake as a core data storage engine while piping their data to and from various third-party integrations for other specialised tasks. Partner connect is an extension to the ecosystem that lets you easily connect your Snowflake account with trial accounts from select third-party integrations to try and choose whichever works best for you.

2. Optimised table structures under the hood - Query performance and table optimisation is not something you have to worry about with Snowflake. It's taken care of with micro-partitions and data clustering. So you don't need to think about indexes, figuring out partitions or shard data. It's all done for you as data is loaded into tables.

Micro-partitions

Snowflake automatically divides tables by grouping rows into individual micro-partitions of 50–500 MB of data. Micro-partitions create more uniformly-sized partitions that limit data skew and make it easy to prune large tables at a extremely granular level. In other words, queries can use micro-partition metadata to determine which partitions are relevant for a query so that only those partitions are scanned. Snowflake goes one step further by limiting scanning of the partition to only the columns filtered in a query.

Clustered tables

When data in a table is not ordered queries aren't as per formant. Data stored in tables can be ordered on some subset of columns that can be used to co-locate data — this subset is called the clustering key. In Snowflake, clustering metadata is collected for each micro-partition created during data load. The metadata is then leveraged to avoid unnecessary scanning of micro-partitions. For very large tables, clustering keys can be explicitly created if queries are running slower than expected.

3. Sharing data between accounts - Secure data sharing is an innovative feature from Snowflake that allows you to share objects (like tables) from a database in your account with another Snowflake account without actually creating copies of the data. Thus, the shared data doesn't take up additional storage and does not contribute to storage costs for the data consumer. Since data sharing is accomplished through Snowflake's metadata store, the setup is quick and data consumers can access the data instantaneously.

Through this architecture, Snowflake enables creating a network of data providers and data consumers that allows for many use cases. One of them is Snowflake data marketplace — a marketplace that connects providers who want to share free or paid data with consumers. Consumers can have shared data available directly in their accounts to query and join with other data sources as they like. Data exchange is another use case by Snowflake that allows users to collaborate on data with invited members. This makes use cases like sharing data between business customers, suppliers and partners very easy.

For data consumers who don't have Snowflake accounts, Snowflake enables providers to create reader accounts that are a cost-effective way of allowing consumers to access shared data without becoming a Snowflake customer.

4. Zero-copy clones - Cloning data is really painful in traditional data warehousing services because if you want to clone an existing database you have to deploy a whole, new separate environment and load data into it. Having to do this frequently for testing changes, doing ad hoc analysis or creating dev/test environments isn't feasible because you're paying for additional storage. Snowflake's zero-copy allows you to almost instantaneously clone any database or table without creating a new copy. It does this by tracking changes to the clone on its metadata store while in the back-end still referencing to the same data files. The benefit of zero-copy cloning is that you can create multiple independent clones of the same data without any additional costs. What's even cooler is that you can use Snowflake's time travelling feature to make clones of data from a past point in time.

5. Recovering objects using undrop - The UNDROP command is a great feature for recovering from mistakes like dropping the wrong table. Usually when this happens you have to spend a lot of time recovering the backup and restoring your data. With the UNDROP command Snowflake allows you to recover objects instantaneously as long as you are still within the recovery window.

6. Support for semi-structured data - One of the biggest step towards big data is Snowflake's ability to combine structured and semi-structured data without having to use complex technologies like Hadoop or Hive. Data can come in multiple forms from sources like machine-generated data, sensors and mobile devices. Snowflake supports ingestion of semi-structured data in various formats like JSON, Avro, ORC, Parquet and XML with the VARIANT data type which imposes a 16MB size limit. Snowflake also optimises the data by extracting as much in columnar form and storing the rest as a single column. Flattening nested structures in the semi-structured data is also easy using data functions that can parse, extract, cast and manipulate the data.

7. Continuous data pipelines - Continuous data pipelines automate many of the manual steps involved in loading data into Snowflake tables and then transforming the data for further analysis. Snowflake provides a set of features that enable continuous data ingestion, change data tracking and setting up recurring tasks to build workflows for continuous data pipelines.

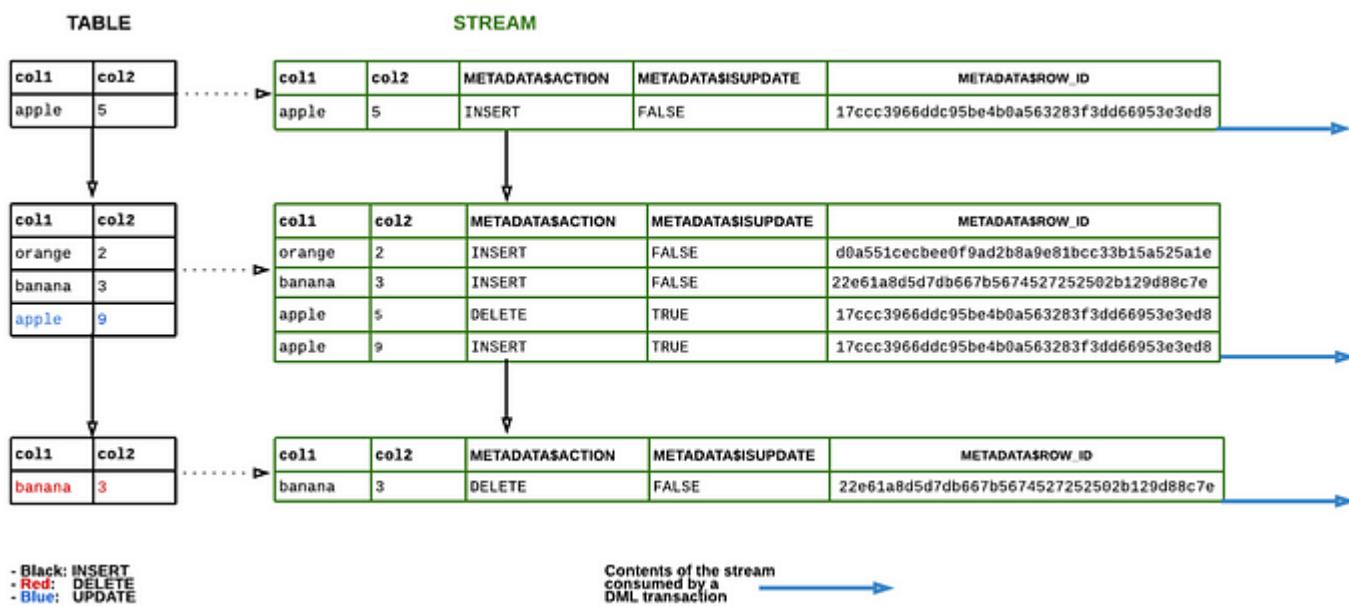
Snowpipe for continuous data ingestion

Snowpipe is Snowflake's continuous data ingestion service. Snowpipe provides a pipeline for loading fresh data in micro-batches as soon as it's available in an external stage like AWS S3, making it available to you within minutes rather than having to manually load larger batches using COPY statements.

Snowpipe works by leveraging event notification in external stages to tell it that new files are available for ingestion. Those files are then copied into a queue from which they are loaded into Snowflake. Snowpipe can also be calling through its REST endpoint. This is advantageous for applications that can call Snowpipe with a list of data filenames that need to be ingested.

Change data capture using Streams

When copying new data from staging tables to other tables its useful to know which data has changed so that only changed data can be copied. Snowflake table streams are a useful feature for doing just that — capturing metadata about DML changes made to a table and the state of a row before and after the change, so that actions can be taken using the changed data. This is also referred to as change data capture.



Schedules SQL statements using Tasks

Tasks are a feature to allow scheduled execution of SQL statements that trigger on an interval that is defined when the task is created. Tasks can either execute single SQL statements or call stored procedures. They are really useful for periodic work like generating report tables or as part of data pipelines that capture recently changed table rows from table streams and transform them for insertion into other tables.

8. Caching results - The Snowflake architecture includes caching at various levels to help speed up your queries and minimise costs. For example, when a query is executed Snowflake holds the results of the query for 24 hours. So if the same query is executed again, by the same user or another user

within the account, the results are already available to be returned, provided that the underlying data has not changed. This is especially beneficial for analysis work where you don't have to rerun complex queries to access previous data or when you want to compare the results of complex query before and after a change.

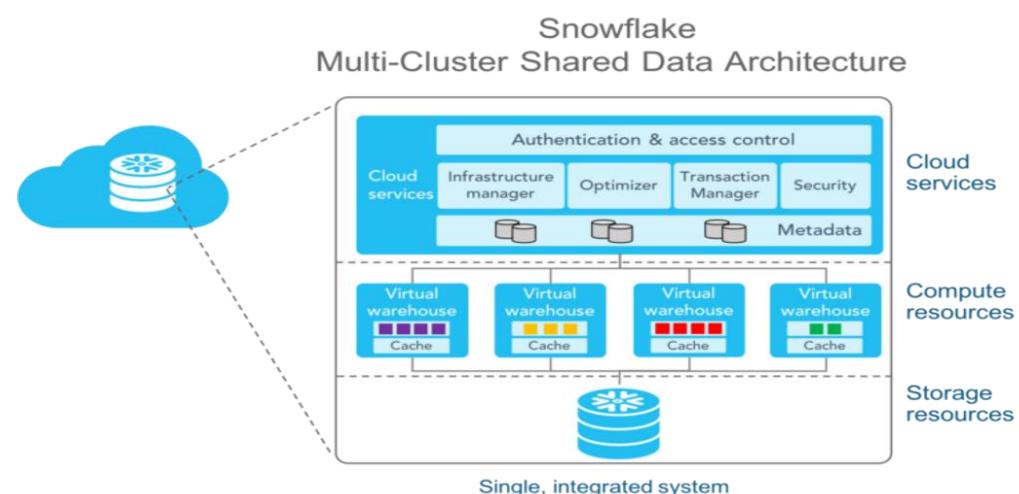
9. External tables - External tables allow you to query data in files stored in data lakes like AWS S3 as if it were inside a Snowflake database. You don't have to know the schema of the data files or format of records beforehand, external tables use schema on read and cast all regular or semi-structured data as the VARIANT data type. This feature gives you the advantage of performing queries on data files in any format supported by Snowflake to see what the data looks like before actually ingesting it into Snowflake.

Types of Data Warehouse Architecture

There Are 3 Ways To Improve Data Warehouse:

- **Single-Tier Architecture:** This type of architecture aims to extract data in order to reduce the amount of data stored.
- **Two-tiered Architecture:** This type of architecture aims to separate the actual Data Sources from the Database. This enables Data Warehouse to expand and support multiple end users.
- **Three-tiered Architecture:** This type of architecture has 3 phases in it. The section below contains Data Warehouse Server Databases, an intermediate section of Online Analytical Processing (OLAP) Server used to provide a vague view of Websites, and finally, the advanced section Advanced Client Framework that includes tools and APIs used to extract data.

Components of the Database Development



The 4 components of the Data Warehouse are as follows.

1. **Database Warehouse Database** - The website forms an integral part of the Database. Database stores and provides access to corporate data. Amazon Redshift and Azure SQL come under cloud-based Database services.
2. **Extraction, Transform, and Load (ETL) Tools** - All activities associated with the extraction, conversion, and uploading (ETL) of data in a warehouse fall under this category. Traditional ETL tools are used to extract data from multiple sources, convert it to readable format, and finally upload to Data Warehouse.
3. **Metadata** - Metadata provides the framework and definitions of data, allowing for the creation, storage, management, and use of data.
4. **Database Access Tools** - Accessibility Tools allow users to access usable and business-friendly information from Data Warehouse. These Warehouse tools include Data Reporting Tools, Data Inquiry Tools, Application Development Tools, Data Mining Tools, and OLAP Tools.

Snowflake - Architecture

- SAAS Offering on Cloud
- No maintenance & administrative overhead costs
- Snowflake separates its architecture into the below 3 layers :
 - a. Cloud Services
 - b. Compute Services
 - c. Database Storage
- Snowflake architecture is a hybrid of “Shared Disk Database” and “Shared Nothing Database”

There was no networking, basically no effort at all. You don't have to worry about provisioning servers, database and schemas. You don't have to worry about building them together. You don't even have to worry about optimizing your data.

Of course, you need to be aware of certain concepts when it comes to the optimization. But for the most part, nothing handles it extremely well.

It has two layers.

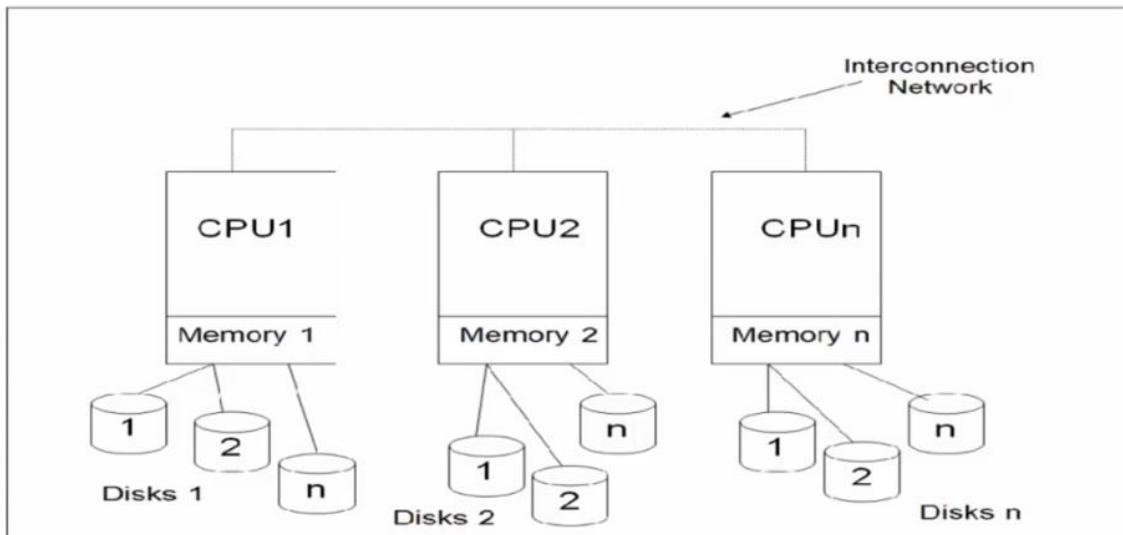
First, one is a cloud services layer.

And then you had your compute services layer.

Finally you had your database storage.

Snowflake by itself is built on an architecture which is a combination or a hybrid of shared this database and shared nothing database.

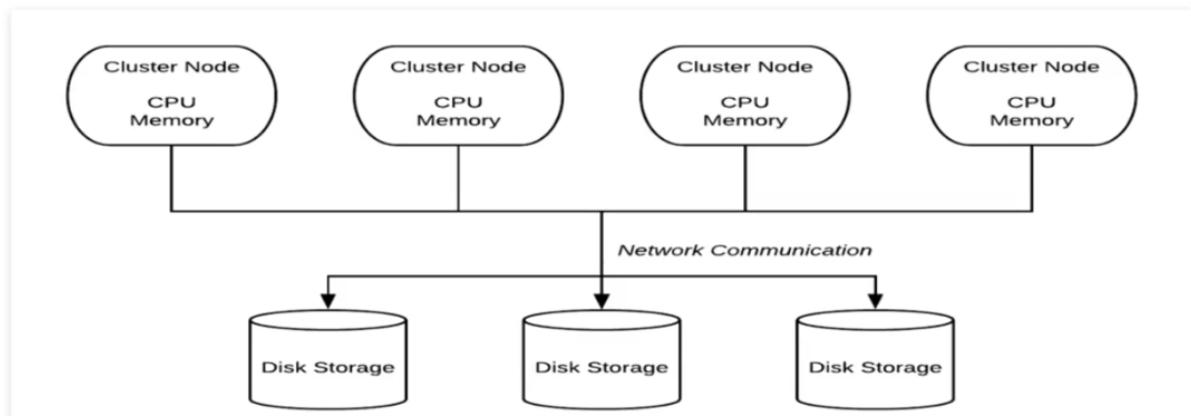
Shared Nothing Architecture



You have your different servers within a cluster and each of those servers has its own copy of data. So the data isn't shared, nor is the memory shared for obvious reasons, because they're different servers within a cluster.

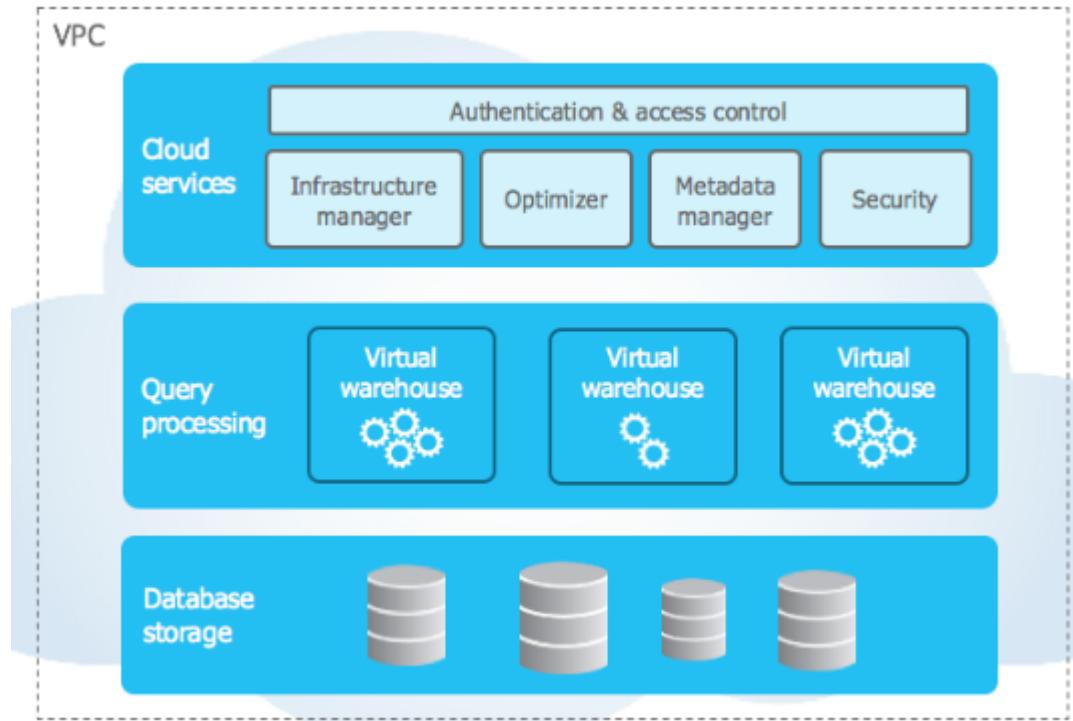
When it comes to essentials. The question is sent to your data. Majority of the applications that you hear of now are built on shared nothing architecture as opposed to share based architecture which look something like this, the database or the disk is shared across multiple servers.

Shared Disk Architecture



When it comes to MapReduce, also this architecture, the data is sent to your code as opposed to destiny, where the code is sent to your data.

When it comes to the different layers that we talked about, you have three different layers. The cloud services, which comprises of authentication and access, control, body optimizer, infrastructure, manager, metadata and security layers.



Now, this is the blend of your system, the cloud services layer, and you don't have to worry too much about what's what each of these components need.

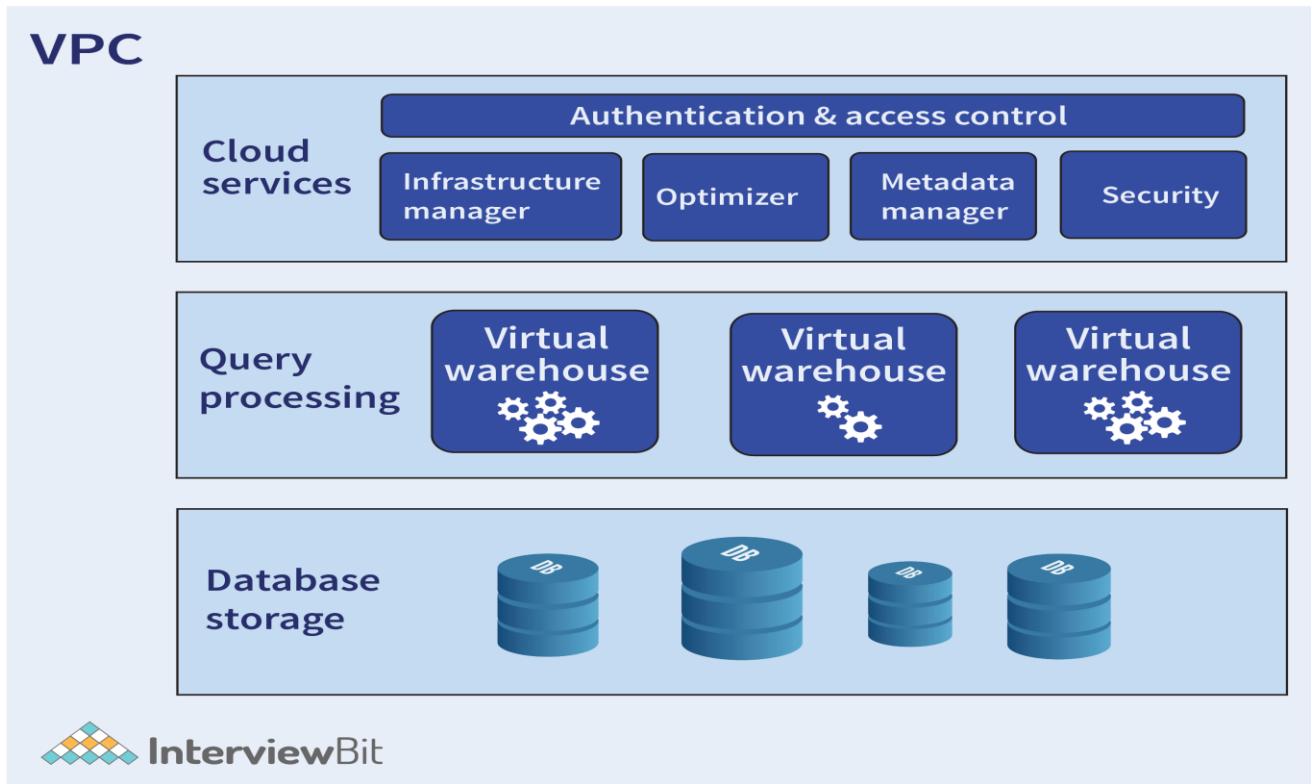
Next up, you have your computer processing layer. If you are a data person, Iraqi engineer, scientist, analyst, you will be spending majority of your time on this layer.

so was come in a range of T-shirts sizes. So what's a warehouse that you can spin up is extra small and it goes all the way up until double excellence. And each of these types have different pricing.

Snowflake Architecture

Snowflake architecture contains a combination of standard shared disk and unallocated formats to provide the best for both. Let's go through these buildings and see how Snowflake integrates them into a new mixed-type construction.

The Snowflake architecture is a hybrid of shared-disk (A common disk or storage device is shared by all computing nodes) and shared-nothing (Each computing node has a private memory and storage space) database architecture in order to combine the best of both. Snowflake utilizes a central data repository for persistent data, which is available to all compute nodes similar to a shared-disk architecture. But, equally, as with shared-nothing architectures, Snowflake uses massively parallel computing (MPP) clusters for query processing, in which each node stores part of the whole data set locally.



The Snowflake architecture is divided into three key layers as shown below:

- **Database Storage Layer:** Once data has been loaded into Snowflake, this layer reorganizes that data into a specific format like columnar, compressed, and optimized format. The optimized data is stored in cloud storage.
- **Query Processing Layer:** In the processing layer, queries are executed using virtual warehouses. Virtual warehouses are independent MPP (Massively Parallel Processing) compute clusters comprised of multiple compute nodes that Snowflake allocates from cloud providers. Due to the fact that virtual warehouses do not share their compute resources with each other, their performance is independent of each other.
- **Cloud Services Layer:** It provides services to administer and manage a Snowflake data cloud, such as access control, authentication, metadata management, infrastructure management, query parsing, optimization, and many more.

Shared-Disk Architecture Overview

Used on a standard website, shared disk architecture has a single storage layer accessible to all cluster nodes. Many cluster nodes with CPU and Memory without disk storage themselves connect to the central storage layer for data processing and processing.

Shared-Nothing Architecture Overview

In contrast to the Shared-Disk architecture, Shared-Nothing architecture distributed cluster nodes and disk storage, its CPU, and Memory. The advantage here is that data can be categorized and stored across all cluster nodes as each cluster node has its own disk storage.

Snowflake Architecture – Hybrid Model

The snowflake supports high-level formation as shown in the diagram below. Snowflake has 3 different layers:

1. Storage Layout
2. Computer Layer
3. Cloud Services Background

1. **Storage Layout** - Snowflake organizes data into many smaller compartments that are internalized and compressed. Uses column format to save. Data is stored in cloud storage and acts as a shared disk model thus providing ease of data management. This ensures that users do not have to worry about data distribution across all multiple nodes in the unassigned model.

Calculation notes connect to the storage layer to download query processing data. Since the storage layer is independent, we only pay for the monthly storage amount used. As Snowflake is offered on the Cloud, storage is expandable and charged as per the monthly TB use.

2. **Computer Layer** - Snowflake uses the “Virtual Warehouse” (described below) to answer questions. Snowflake splits the query processing layer into disk storage. It uses queries in this layer using data from the storage layer.

Virtual Warehouses MPP compiles include multiple nodes with CPU and Memory provided in the cloud by Snowflake. Multiple Virtual Warehouses can be created on Snowflake for a variety of needs depending on the workload. Each visible warehouse can operate on a single layer of storage. Typically, the visible Warehouse has its own independent computer collection and does not interact with other warehouses.

Benefits of Virtual Warehouse

Some of the benefits of a visual warehouse are listed below:

1. Virtual Warehouses can be started or suspended at any time and can be measured at any time without the impact of effective queries.

2. They can also be set to auto-stop or restart automatically so that storage is stopped after a certain period of inactivity and when the query is sent it is restarted.
3. It can also be set to default on a smaller and larger collection size, hence e.g. we can set a minimum of 1 and a maximum of 3 so that depending on the load of Snowflake can provide between 1 to 3 bulk storage containers.

3. Cloud Services Layer - All functions such as authentication, security, uploaded metadata data management and query integration linking Snowflake across this layer occur in this layer.

Examples of services hosted in this layer:

1. If the application for entry is filed it must go through this section,
2. The query sent to Snowflake will be sent to the developer in this layer and forwarded to the Compute Layer for processing the query.
3. Metadata needed to improve query or filter data is stored in this layer.

These three layers measure independently and charge Snowflake for storage and a warehouse that looks separately. The service layer is managed within the given computer nodes, which is why it can be charged.

The advantage of this Snowflake structure is that it can measure any one layer without the other. For example you can measure the storage layer by extension and you will be charged for storage separately. More virtual repositories can be provided and rated when additional resources are needed to quickly process queries and improve efficiency. Learn more about Snowflake buildings from here.

Connects to Snowflake

Now that you are familiar with Snowflake buildings, now is the time to discuss how to connect to Snowflake. Let's take a look at some of the best foreign company tools and technologies that create an extended ecosystem to connect to Snowflake.

- **Snowflake Ecosystem** – This list will take you to Snowflake partners, customers, third-party tools, and emerging technologies in the Snowflake ecosystem.

Partners of foreign companies and technology are certified to provide indigenous connections to Snowflake.

Data integration or ETL tools are known for providing traditional connectivity to Snowflake.

Business intelligence (BI) tools make it easier to analyse, retrieve, and report on business data to help organizations make informed business decisions.

Mechanical & Data Science incorporates a wide range of vendors, tools, and technologies that extend Snowflake functionality to provide advanced mathematical modelling and forecasting skills.

Protection and Management tools ensure that your data is stored and kept secure. Snowflake also provides native SQL development and Data query areas.

Snowflake supports application development using many popular programming languages and development platforms.

- **Snowflake Partner Connect** – This list will take you to Snowflake Partners who offer free Snowflake Connect tests.
- **Standard Configuration (All Clients)** – This is a set of standard configuration commands that apply to all Clients provided by Snowflake (CLI, connectors, and drivers).
- **SnowSQL (CLI Client)** – SnowSQL is a next-generation command line utility to connect to Snowflake. Allows you to create SQL queries and perform all DDL and DML tasks.
- **Connectors and Drivers** – Snowflake provides drivers and connectors for Python, JDBC, Spark, ODBC, and other clients to improve the app. You can go through all of them listed below to start learning and applying them.

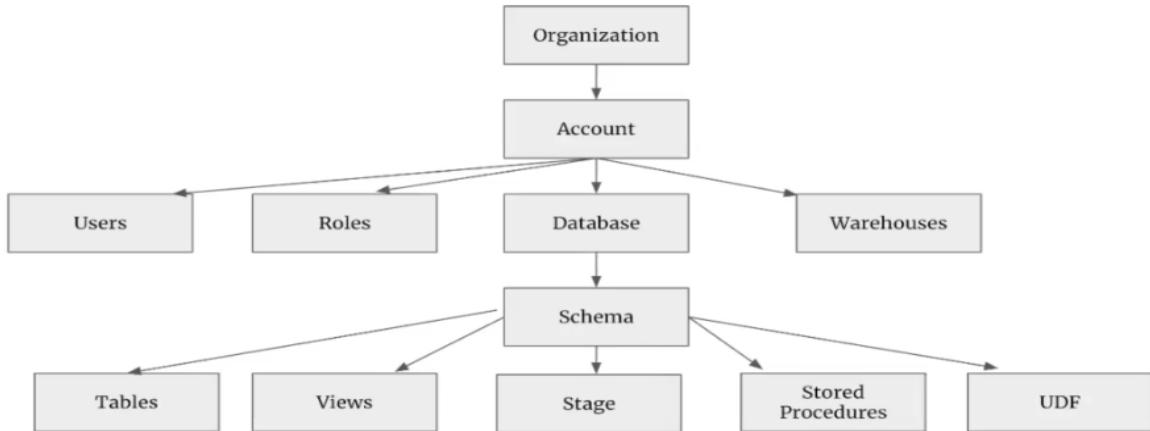
[Python Snowflake connector](#) [Spark Snowflake connector](#) [Kafka Snowflake connector](#) [Driver of Node.js](#)

[Go Snowflake Driver](#)

[.NET Driver](#) [JDBC driver](#) [ODBC driver](#)

[PHP PDO Snowflake Driver](#)

Snowflake – Object Hierarchy



The screenshot shows the Microsoft Edge browser displaying the Snowflake Worksheet interface at <https://hn54032.eu-west-1.snowflakecomputing.com/console#/internal/worksheet>. The top navigation bar includes File, Edit, View, History, Favorites, Tools, Profiles, Tab, Window, Help, and a user account icon for 'SIDDY794 ACCOUNTADMIN'. The main area shows a 'New Worksheet' tab selected under 'Worksheets'. A sidebar on the left lists 'Databases' (SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA) and 'Shares'. The central workspace displays a query results table with one row: '1 ECOMMERCE_DB successfully dropped.' The bottom right corner shows account details: Organization (JCBPYFJ), Edition (Standard), Cloud (Amazon Web Services), and Region (EU (Ireland)).

Everything inside of Snowflake is an object because it's a very generic term. So as you can see on my screen at the topmost, you have your organization and each organization can have one or more accounts under them. What we signed up for was an account.

If you want to understand that if the relationship between account and organization, it's many two one. And let's say you work for Amazon and hypothetically, let's say Amazon has two different entities. It has Amazon e-commerce and then it has Amazon. It's the music right now if Amazon.

The screenshot shows the Microsoft Edge browser displaying the Snowflake Organization page at <https://hn54032.eu-west-1.snowflakecomputing.com/console#/organization>. The top navigation bar includes File, Edit, View, History, Favorites, Tools, Profiles, Tab, Window, Help, and a user account icon for 'SIDDY794 ORGADMIN'. The main area shows the 'Organization' tab selected. Below it, the 'Accounts' section is shown with a single account listed: 'W63665' (Edition: Standard, Snowflake Region: AWS_EU_WEST_1, Date Created: 1/1/2022, Account URL: <https://jcbpyfj-ew63665.snowflakecomputing.com>, Account Locator: HN54032, Account Locator URL: <https://hn54032.eu-west-1.snowflakecomputing.com>). A message at the top states 'Last refreshed 12:31:19 PM'.

Each of these schemas will have the exact same set of tables inside of them is that they used for different purposes. One is for production, a live one is the staging, and the other one is for testing.

Snowflake - Virtual warehouses

- Warehouse is a cluster of computing resources
- A warehouse is a combination of resources like CPU, memory, and temporary storage required to do multiple operations
- The Operations include :
 - Executing sql queries
 - Data loading/unloading
- Warehouses come in sizes ranging from XS to 5XL
- Warehouse can be of 2 types :
 - Single Cluster Warehouse
 - Multi Cluster Warehouse

The first one is the maximum cluster. It's the maximum number of warehouses that Snowflake will spin up during an operation.

The next one is the minimum number of cluster, which is the defined number warehouses that snowflake will spin up while creating a warehouse. So let's say you want to create a thing.

If you have the warehouse, in this case, your minimum cluster is going to be equal to one. And the max clusters also want to just one node, right? But if you want to create a multiple of the warehouse, your mini cluster should be greater than or equal to one, and the max cluster has to be greater than one. So that's when it can scale out and become a multi cluster warehouse.

When it comes to a Multi-Class warehouse, there are two modes in which it can be executed. The first one is the maximise mode and the second one is the auto scale mode.

So maximize more theoretically means you're telling snowflake max out at the time of creation of a warehouse.

Snowflake - Virtual warehouse Configuration

- **Max Cluster** is the maximum number of warehouses that can be spun up during an operation
- **Min Cluster** is the default number of warehouses that Snowflake will spin up while creating a warehouse
- **Single Cluster Warehouse** : Min Cluster = 1 , Max Cluster = 1
- **Multi Cluster Warehouse** : Min Cluster >= 1 , Max Cluster > 1

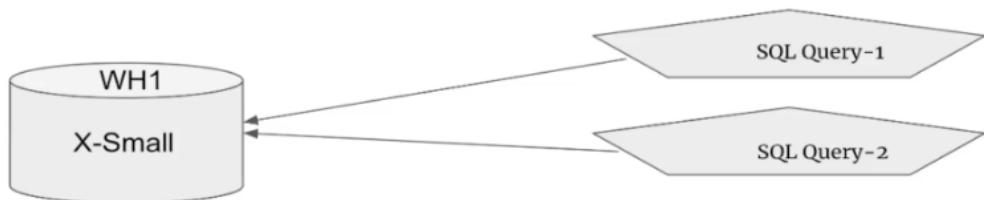
Inside Snowflake, the virtual warehouse is a cluster of compute resources. It provides resources — including memory, temporary storage and CPU — to perform tasks such as DML operation and SQL execution.

Snowflake - Multi Cluster warehouse modes

- Multi-cluster warehouse can be executed in two modes :
 - Maximised : Min Clusters = 2 ,Max Cluster = 2
 - AutoScale : Min Clusters = 1 ,Max Cluster = 2
- Scaling Policy :
 - Standard (Default) : Prevents queuing of operations by starting additional warehouses
 - Economy : Conserves credits by favoring keeping running warehouses fully-loaded rather than starting additional warehouses

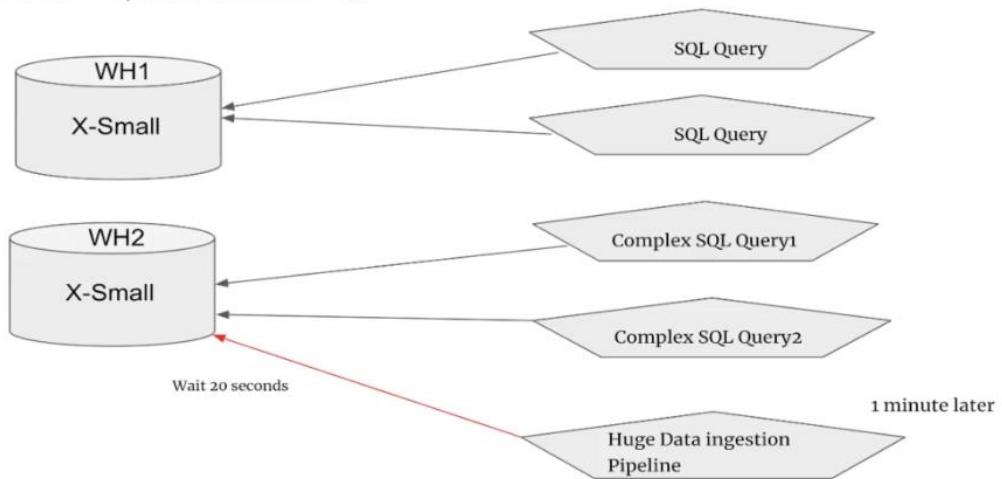
Multi Cluster Warehouse | Scaling Policy - Standard

Min Cluster = 1, Max Cluster = 3



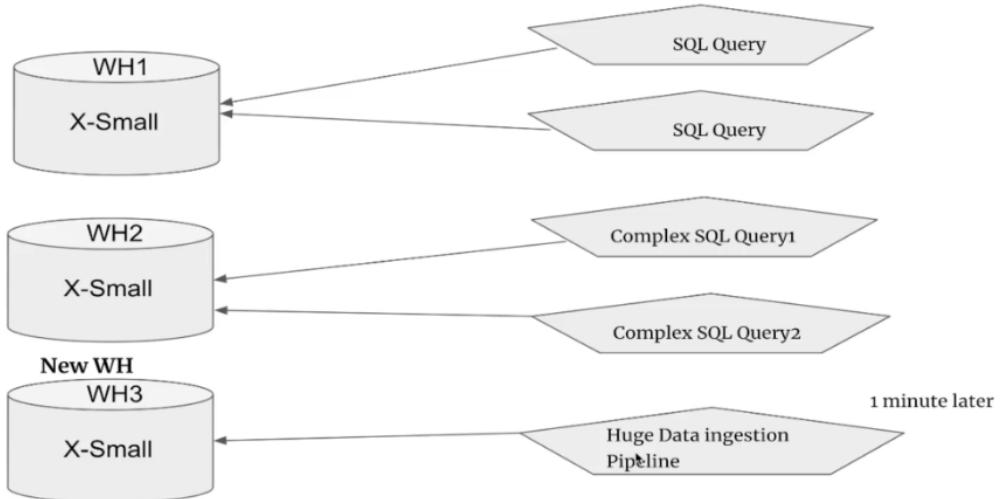
Scaling Policy - Standard

Min Cluster = 1, Max Cluster = 3



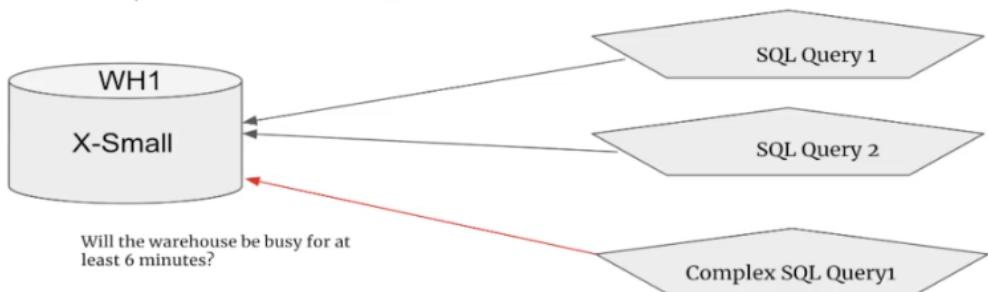
Scaling Policy - Standard

Min Cluster = 1, Max Cluster = 3



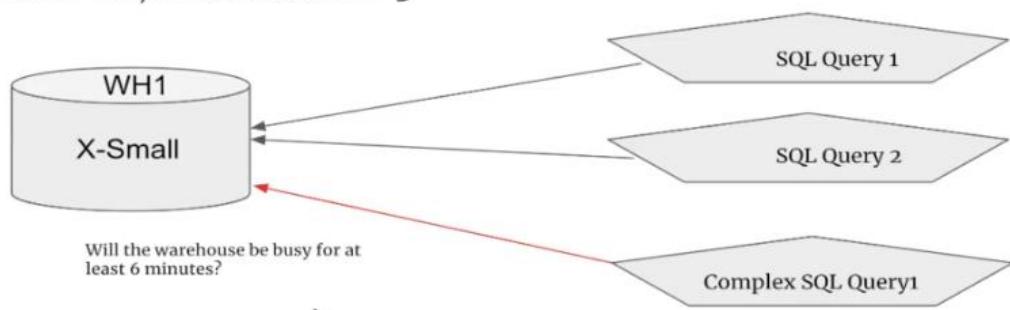
Scaling Policy - Economy

Min Cluster = 1, Max Cluster = 3



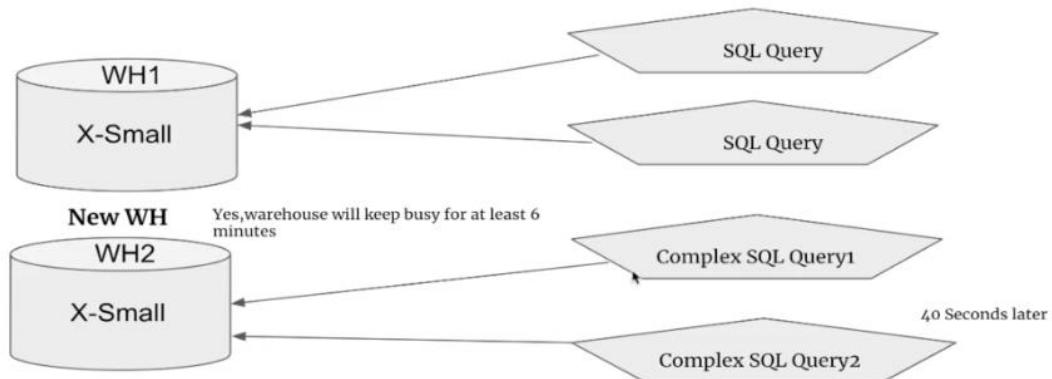
Scaling Policy - Economy

Min Cluster = 1, Max Cluster = 3



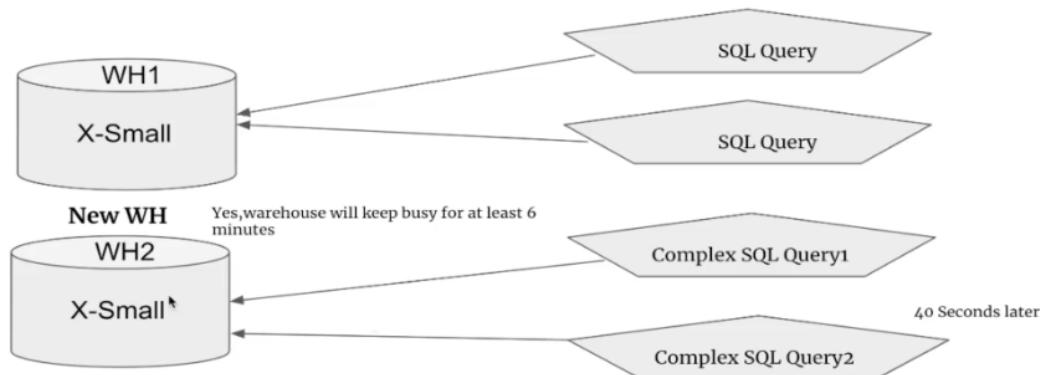
Scaling Policy - Economy

Min Cluster = 1, Max Cluster = 3



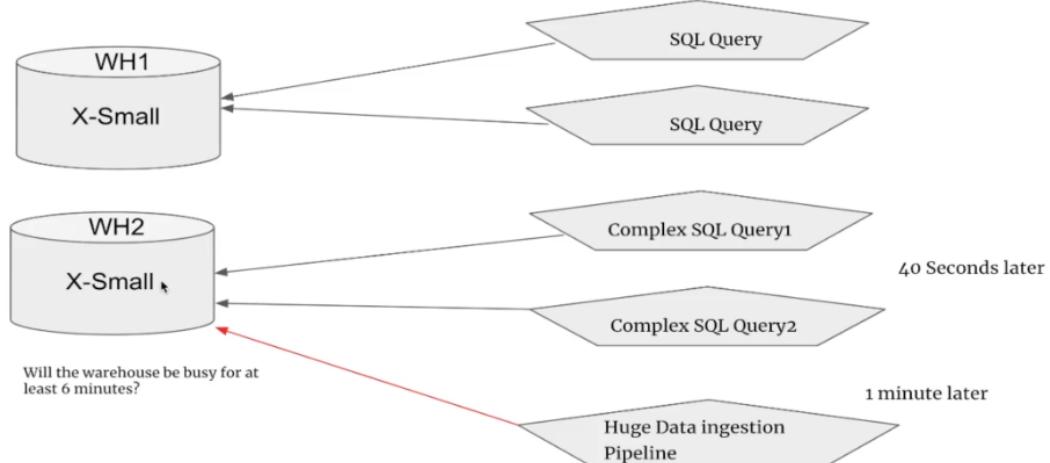
Scaling Policy - Economy

Min Cluster = 1, Max Cluster = 3



Scaling Policy - Economy

Min Cluster = 1, Max Cluster = 3



Snowflake Warehouse T-Shirts are available in a variety of sizes, as shown below:

Size specifies the number of servers that comprise each cluster in a warehouse. Snowflake supports the following warehouse sizes:

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

An increase in T-Shirt size (XS-4XL) corresponds to a proportional increase in CPU, Memory, and Temporary Storage. You don't have control over individual sizes, but you can change your warehouse size by selecting one of the T-Shirt sizes. Because Snowflake storage and computing are loosely coupled, you can start and stop the warehouse at any time.

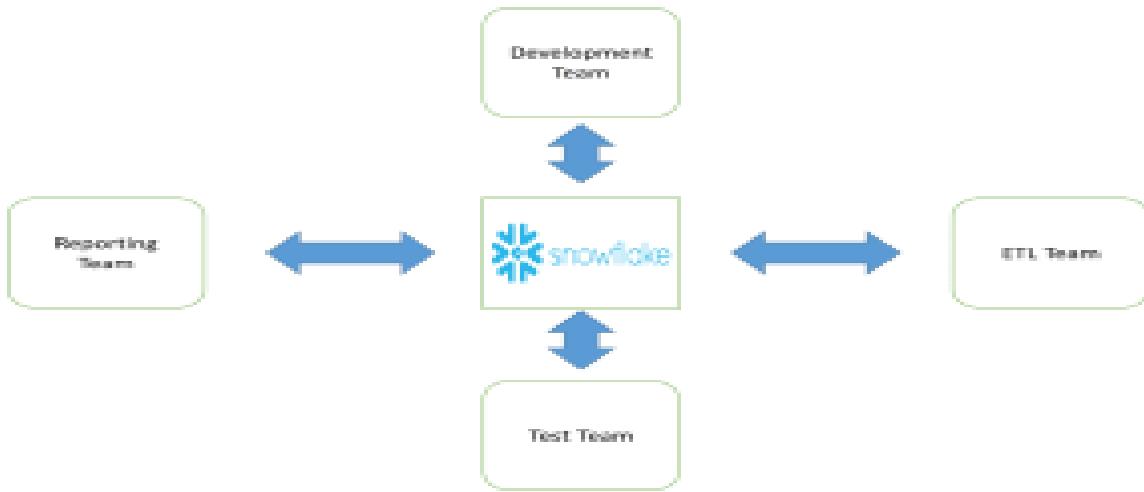
You can specify the size, multi-cluster attribute (Enterprise & above), and scaling policy at the time of creation. See the list below.

Create Warehouse

Name *	DUMMY
Size	X-Large (16 credits / hour)
Learn more about virtual warehouse sizes here	
Maximum Clusters	2
Multi-cluster warehouses improve the query throughput for high concurrency workloads.	
Minimum Clusters	1
The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.	
Scaling Policy	Standard
The policy used to automatically start up and shut down clusters.	
Auto Suspend	10 minutes
The maximum idle time before the warehouse will be automatically suspended.	
<input checked="" type="checkbox"/> Auto Resume ?	
Comment	

Because there are a variety of workloads in any organization, you can't choose just one or two to cover the entire workload. Also, in Snowflake, creating multiple warehouses is free; you can create as many as you want and only pay for what you use. For instance, we have the following groups in any high-level IT project:

- Development Team
- ETL Team
- Test Team
- Reporting Team



Each group of people is assigned a different amount of work. Let's say the ETL team is constantly busy loading data, while the reporting team is only working on query reports. While the development team is busy writing code, they may try a variety of activities.

Similarly, a test team will run the entire flow from beginning to end or module by module to ensure the program's sanity. As a result, each group of people has a different workload, necessitating the use of different warehouses.

So the best option is to set up a separate warehouse for each team with some initial sizing and monitor whether you need to adjust the size as they require.

What is Snowflake's Scaling Policy?

You can choose between two types of scaling policies in Snowflake:

- Standard(default)
- Economy

Snowflake supports the following scaling policies:

Policy	Description	Cluster Starts...	Cluster Shuts Down...
Standard (default)	Prevents/minimizes queuing by favoring starting additional clusters over conserving credits.	The first cluster starts immediately when either a query is queued or the system detects that there's one more query than the currently-running clusters can execute. Each successive cluster waits to start 20 seconds after the prior one has started. For example, if your warehouse is configured with 10 max clusters, it can take a full 200+ seconds to start all 10 clusters.	After 2 to 3 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed to the other clusters without spinning up the cluster again.
Economy	Conserves credits by favoring keeping running clusters fully-loaded rather than starting additional clusters, which may result in queries being queued and taking longer to complete.	Only if the system estimates there's enough query load to keep the cluster busy for at least 6 minutes.	After 5 to 6 consecutive successful checks (performed at 1 minute intervals), which determine whether the load on the least-loaded cluster could be redistributed to the other clusters without spinning up the cluster again.

What are the 2 modes to Define Warehouse in Snowflake?

Warehouses can be provisioned in two different ways in Snowflake. It can be set to either maximized or Auto-Scale mode.

- [Auto-Scale Mode](#)
- [Maximized Mode](#)

Auto-Scale Mode

You select auto-scale mode when you enter different values for Minimum and Maximum clusters, as shown below. Snowflake uses this mode to dynamically manage the warehouse load by starting and stopping clusters as needed. The warehouse scales out/in as the number of concurrent users or query load increases or decreases.

Configure Warehouse

Name COMPUTE_WH

Size X-Small (1 credit / hour) [Learn more about virtual warehouse sizes here](#)

Maximum Clusters Multi-cluster warehouses improve the query throughput for high concurrency workloads.

Minimum Clusters The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.

Scaling Policy Standard The policy used to automatically start up and shut down clusters.

Auto Suspend 10 minutes The maximum idle time before the warehouse will be automatically suspended.

Auto Resume [?](#)

Comment

[Show SQL](#) [Cancel](#) [Finish](#)

Maximized Mode

When you set the Minimum and Maximum clusters to the same value (Should be >1), you're choosing maximized mode. When the warehouse is started in this mode, Snowflake starts all of the clusters, ensuring that the warehouse has the most resources available. This mode is appropriate when you know your workload will have multiple concurrent users and will require a provisioned server to support them.

Configure Warehouse

Name COMPUTE_WH

Size X-Small (1 credit / hour) [here](#)

Maximum Clusters 5
Multi-cluster warehouses improve the query throughput for high concurrency workloads.

Minimum Clusters 5
The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.

Scaling Policy Standard
The policy used to automatically start up and shut down clusters.

Auto Suspend 10 minutes
The maximum idle time before the warehouse will be automatically suspended.
 Auto Resume [?](#)

Comment

[Show SQL](#) [Cancel](#) [Finish](#)

How to Create a Snowflake Virtual Warehouse?

Creating a Snowflake Virtual Warehouse is a fairly simple process. Follow the steps given below to do so:

- The SQL script given below demonstrates how to create a **SMALL** Snowflake Virtual Warehouse that will automatically suspend after **10 minutes** and resume immediately once queries are executed.

```
create warehouse PROD_REPORTING with
  warehouse_size = SMALL
  auto_suspend = 600
  auto_resume = true
  initially_suspended = true
  comment = 'Sample Reporting Warehouse';
```

- Once you have successfully created a Virtual Warehouse, use the following command to select a specific Virtual Warehouse:

```
use warehouse SAMPLE_REPORTING;
```

- Any SQL Statements/Queries executed after this point will be executed on the specified Virtual Warehouse. Using this method, different teams can be assigned dedicated hardware, and each user can be assigned a default warehouse.

How to Manage the Cost of Your Snowflake Virtual Warehouse?

T-Shirt Size	Database Servers	Credits per Hour
X-Small	1	1
Small	2	2
Medium	4	4
Large	8	8
X-Large	16	16
2X-Large	32	32
3X-Large	64	64
4X-Large	128	128

Depending on the size of your data and the number of users tasked with the Data Warehousing and Data Management operations, Snowflake offers a set of computing clusters categorized by their sizes. While running, a **Snowflake Virtual Warehouse** consumes **Snowflake Credits**. The number of credits consumed is determined by the size of the warehouse and the duration of the Virtual Warehouse. Managing Credit Consumption to ensure that credits are used efficiently helps in managing the cost of a Snowflake Virtual Warehouse. There are some very simple things you can do to manage the cost of your Snowflake Virtual Warehouse. They are as follows:

1) Set up Resource Monitors

Snowflake Virtual Warehouse's Compute Costs can be managed by triggering specific actions, such as sending notifications or suspending one or more warehouses. This can be done with the help of **Resource Monitors**. Hence, it is advisable to create Resource Monitors if you are an account administrator.

2) Query the Snowflake Database

The Account Administrator also has access to the Snowflake database, which is a **read-only shared database** provided by Snowflake. This database, among other things, can be queried to retrieve

information about the Warehouses consuming the most credits. Once you've successfully determined the warehouses consuming the maximum credits, you can delve into further details using the **History Tab**.

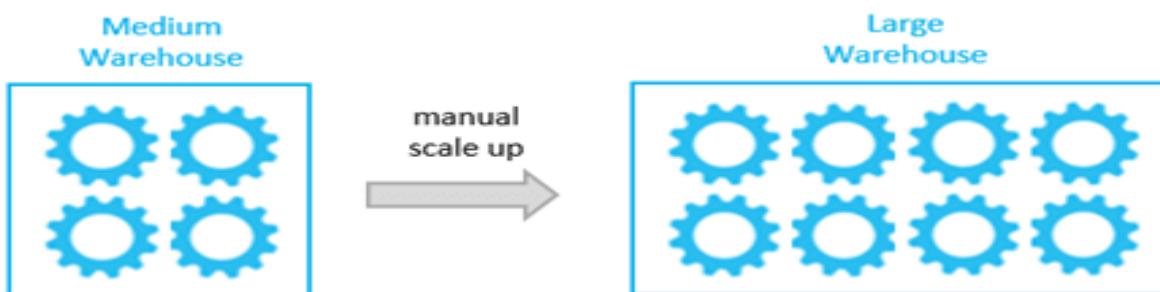
3) Make Good Warehouse Auto-Suspension Choices

When you create a new Warehouse, the **Auto-Suspend** value is set to **600 seconds** by default. This means that the Warehouse will automatically suspend after 10 minutes of inactivity. If your workload runs infrequently, you may want to reduce your default value to one minute (60 seconds). It is important to note that you **should not set** the Auto-Suspend value to NULL unless your query workloads necessitate a continuously running Warehouse, otherwise your consumption charges will be much higher than necessary. Since the Warehouse cache is lost every time a Warehouse is suspended, it may be cheaper and faster not to have the Warehouse Auto-Suspend if jobs are continuously executed.

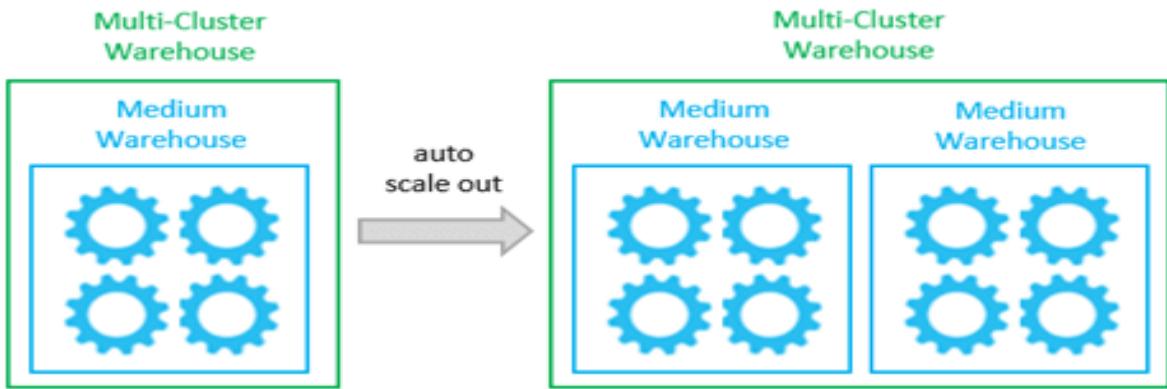
What are the Misconceptions about Virtual Warehouses in Snowflake?

The following is a list of common honest mistakes that most of us make during the learning process:

- **Virtual Warehouses can be Dynamically Resized:** Even if you have running queries and other queries queued for them, you can only manually resize virtual warehouses at any time. Snowflake only allows you to auto-scale horizontally, not vertically (up or down) (in or out). In the Enterprise Edition, this last auto-scale mode is supported by a multi-cluster environment, in which each cluster is a virtual warehouse of the same size, processing queries in parallel, and can be started and stopped.

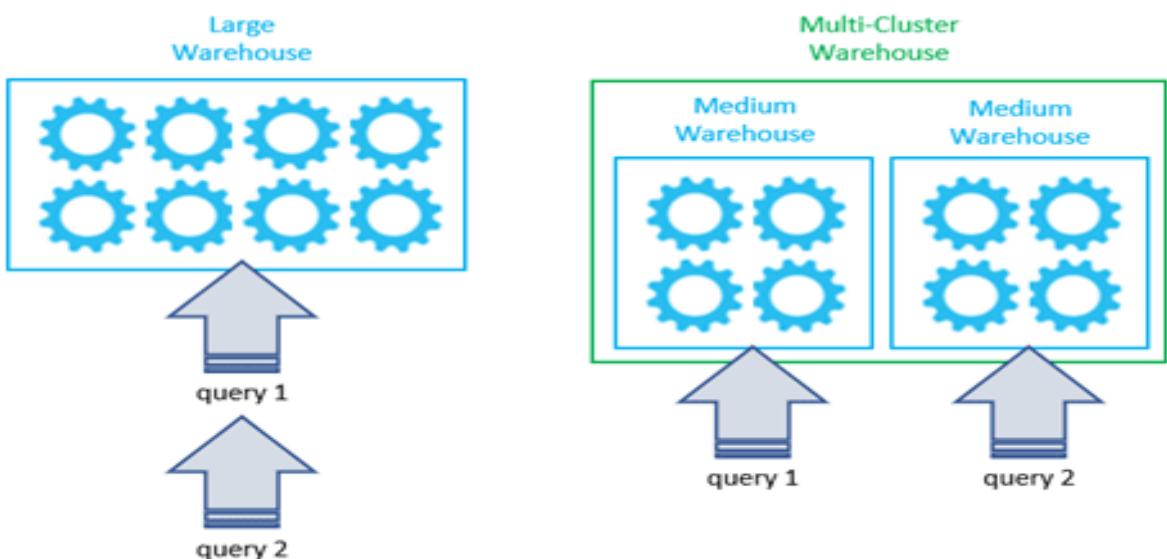


A size-4 (Medium) warehouse has been resized to a size-8 warehouse in the previous image (Large). Warehouses can be resized in powers of two (i.e. 1, 2, 4, 8, 16, 32,... 256, 512) from 1 (X-Small) to 512 (6X-Large). The warehouse can be self-contained or part of a larger cluster. When a multi-cluster warehouse is resized, the size of each individual warehouse is increased or decreased by the same amount.



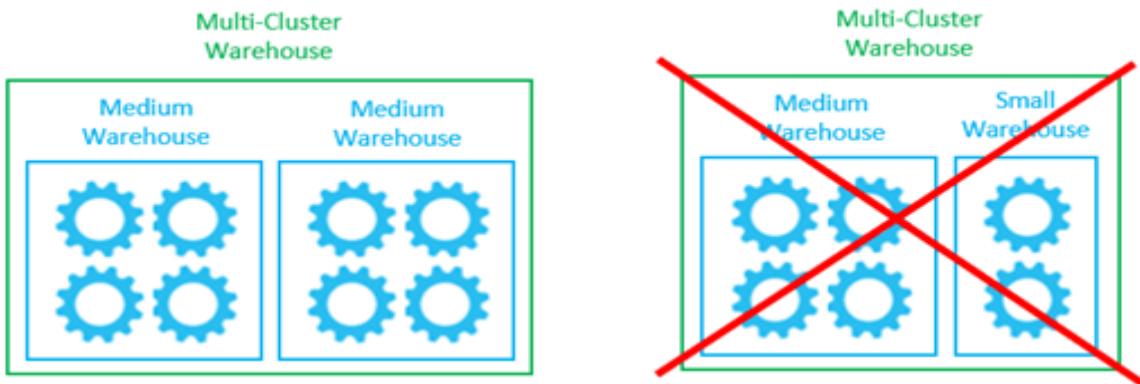
An auto-scale of a multi-cluster warehouse can be seen in the previous image. Individual warehouses, each of the same size, are duplicated here. A size-4 (Medium) warehouse has been replicated in this case for a cluster of two warehouses of the same size.

- **Resizing a Virtual Warehouse will Improve Concurrency:** The key distinction between resizing a warehouse and resizing the number of clusters in a multi-cluster warehouse is that each has a distinct impact on performance or concurrency:
 - Warehouse resizing boosts efficiency. Query size scales linearly with warehouse size for the most part, especially for larger, more complex queries. This feature allows for manual vertical scaling (up or down).
 - Concurrency is improved with multi-cluster warehouses. They're made to deal with queuing and performance issues caused by a large number of concurrent users and/or queries. Automatic horizontal scalability is enabled by this feature (in or out).



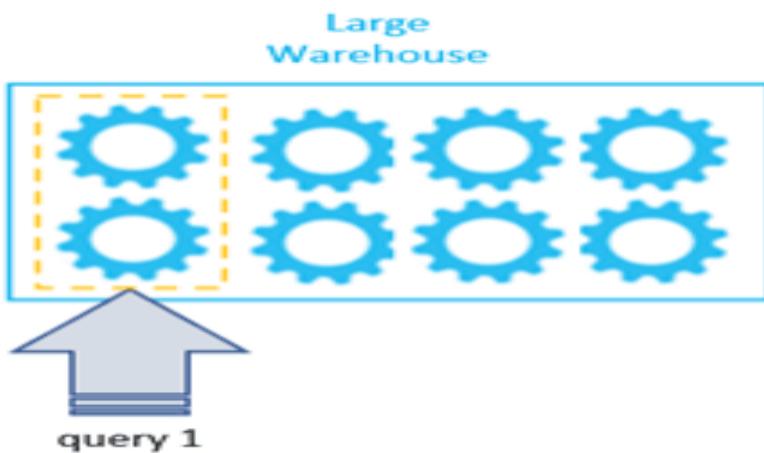
Any individual query will most likely be processed faster by the size-8 (Large) warehouse on the left than by any size-4 (Medium) warehouse on the right. If queries 1 and 2 are submitted at the same time, query 2 will most likely be queued and will have to wait for query 1 to be processed first. However, query 1 and query 2 can be executed in parallel by each individual warehouse in the multi-cluster warehouse.

- **You may have Warehouses of Different Sizes in a Multiple-Cluster:** When you create a virtual warehouse with multiple clusters in Enterprise Edition, each cluster must have the same number of nodes, which is referred to as a “size.” It’s not possible to start parallel clusters with different numbers of nodes for the same multi-cluster virtual warehouse.



On the left, there are two size-4 warehouses in a multi-cluster warehouse (Medium). The one on the right has warehouses of various sizes, which is impossible to accommodate.

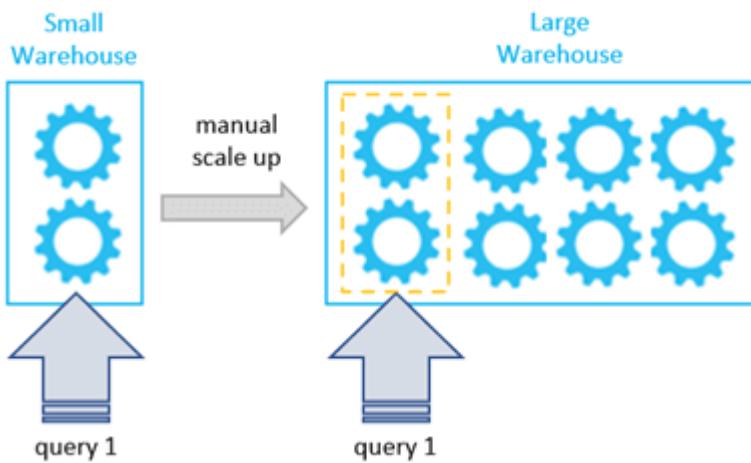
- **A Larger Warehouse will Run your Query Faster:** Possibly not. One or two nodes may be enough for simple queries. There are a slew of other variables to consider as well. In a high-volume multi-user environment, a multi-cluster virtual warehouse, rather than a larger size, can prevent queries from being queued. Concurrency issues, not complexity, may make your query take longer to process.



Only two nodes may be required for the previous query. The remaining nodes simply waste credits and aren't used. If the majority of the queries are similar, a size-2 (Small) warehouse could be more cost-effective and efficient.

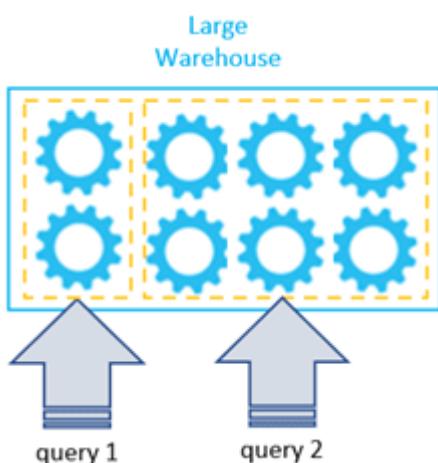
- **An Increase in Size will Improve the Performance of a Currently Slow Running Query without Restart:** Manually resizing virtual warehouses is possible at any time, but it has no effect on existing queries. Any currently running queries are unaffected by the additional

resources. They become available for use by any queued or newly submitted queries once they have been fully provisioned.



You can manually resize the virtual warehouse from size-2 (Small) to size-8 while query 1 is still running (Large). This change, however, has no effect on the execution of query 1, which is still served solely by the first two nodes. If the query could benefit from more nodes, it must be paused and restarted.

- **A Query is Always Queued if the Current Warehouse is Busy:** The number of queries that can be processed concurrently in single-cluster virtual warehouses is limited by the size and complexity of each query. The warehouse calculates and reserves the compute resources required to process each query as queries are submitted. Only if there are insufficient remaining resources in the warehouse to process a query will it be queued, awaiting resources to become available as other running queries complete.

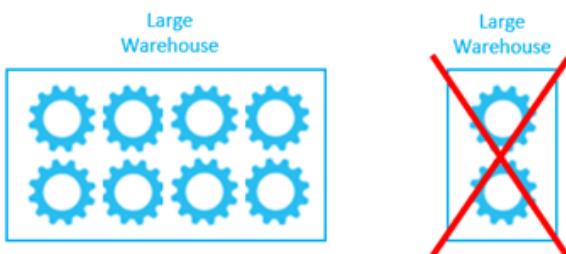


A virtual warehouse of size 8 (Large) may determine that query 1 only requires two nodes. If some of the remaining nodes are idle, another incoming query may use them, and so on. If both query 1 and query 2 have enough resources, a single warehouse can run both queries in parallel.

Snowflake recommends multi-cluster warehouses to enable fully automated concurrency scaling. Multi-cluster warehouses provide essentially the same benefits as adding additional warehouses and

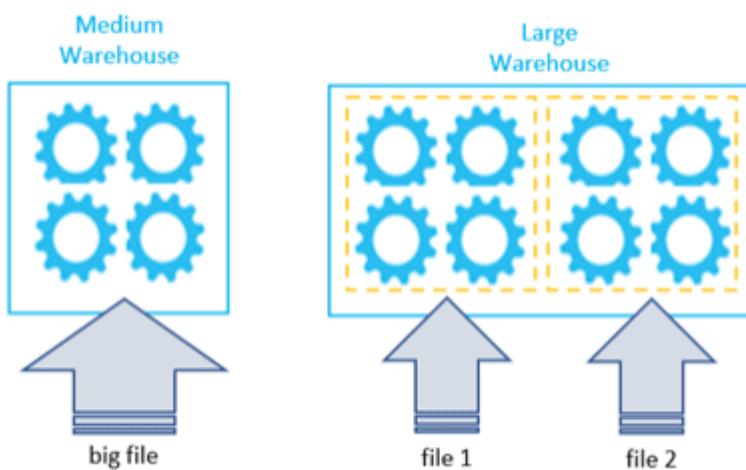
redirecting queries, but without the need for manual intervention. Queries that would otherwise be queued could be automatically redirected to another cluster in the same warehouse and executed.

- **A Large Warehouse can Provision fewer Nodes for very Simple Queries:** In a single-cluster virtual warehouse, all nodes are started and provisioned simultaneously. Only in multi-cluster virtual warehouses can different clusters of nodes be started and stopped automatically and individually. This means that you waste all other running nodes and pay more for nothing for simple queries that may only require one or two nodes to process. A size-1 (X-Small) or size-2 (Small) virtual warehouse might suffice if you're the only user on a Snowflake instance and only run simple queries one at a time.



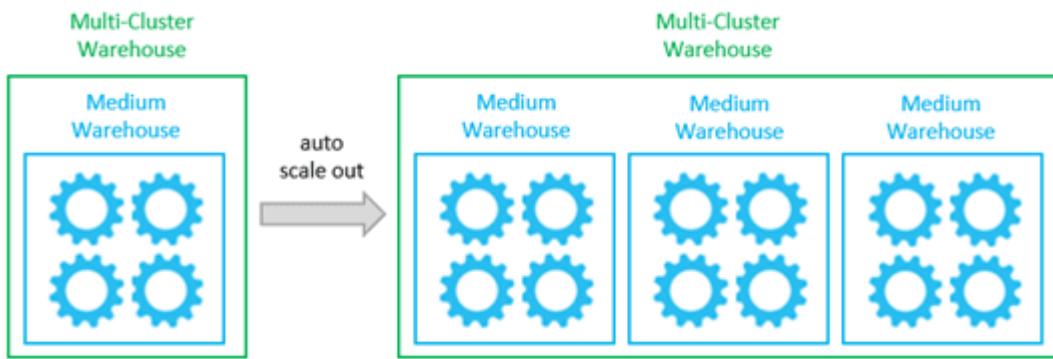
A warehouse of size 8 (Large) will always start with and provision 8 nodes.

- **Increase the Size of a Virtual Warehouse if you have one large File to Ingest:** Increased warehouse-size does not always result in improved data loading performance. The number of files to be loaded and the size of each file has a greater impact on data loading performance than the warehouse's size. A smaller warehouse is usually sufficient unless you're bulk loading hundreds or thousands of files at the same time. Using a larger warehouse will use more credits and may not improve performance.



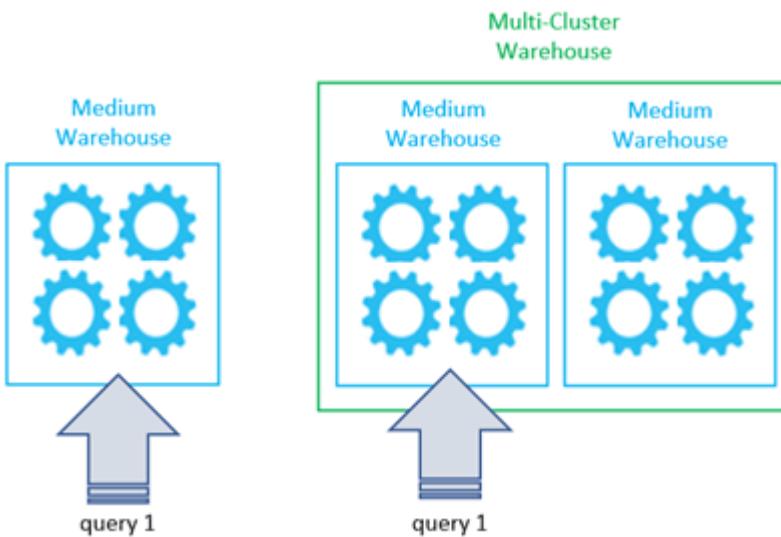
If you ingest a large number of files at once, the data ingestion with the size-8 (Large) warehouse may be faster. Otherwise, because of the volume of data, rather than data processing, a single ingestion of a large file may take longer. Per request, each COPY INTO command can take up to four nodes. With a size-8 (Large) warehouse, using the big file might be a waste of compute credits.

- **Multi-cluster warehouses can auto-scale between a minimum and the maximum number of compute nodes:** The minimum and maximum configuration numbers (accepted values between 1 and 10) refer to the number of single-cluster warehouses, not the number of compute nodes. Each virtual warehouse cluster must have the same number of compute nodes or resources, referred to as “size.” A warehouse can only be scaled up or down manually by resizing it. A multi-cluster warehouse’s number of warehouses, on the other hand, can be automatically scaled up or down.



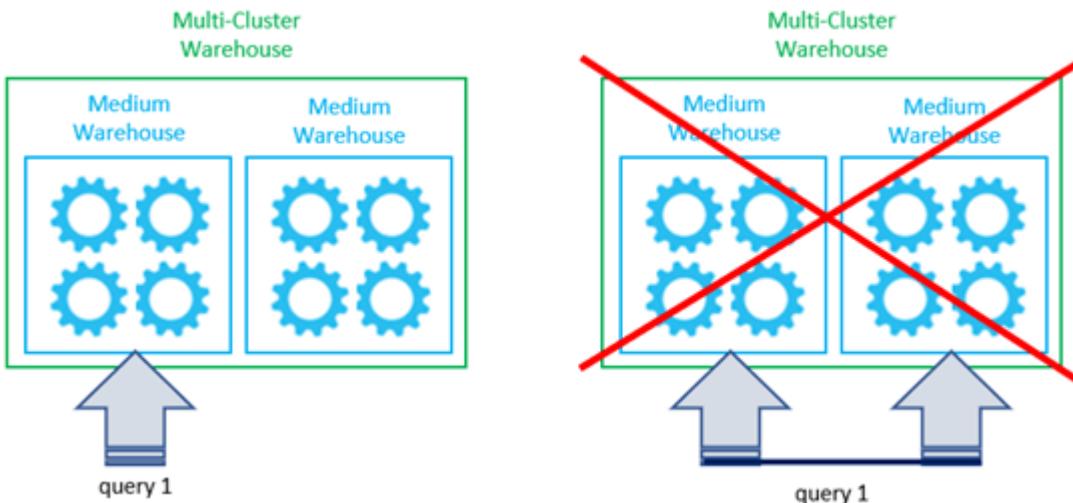
The previous multi-cluster warehouse was expanded from one to three warehouses, each of the same size 4 (Medium). In auto-scale mode, the minimum is set to 1 and the maximum is set to 3 or 10 (maximum allowed value).

- **A Multi-Cluster Warehouse will help Process a Complex Query Faster:** Multiple queries can be processed in parallel using virtual warehouses with multiple clusters. Only the size of the query, not the number of clusters, matters when processing a single query by a single warehouse. A large virtual warehouse could be beneficial for a complex query. It makes no difference in this case if there are more clusters.



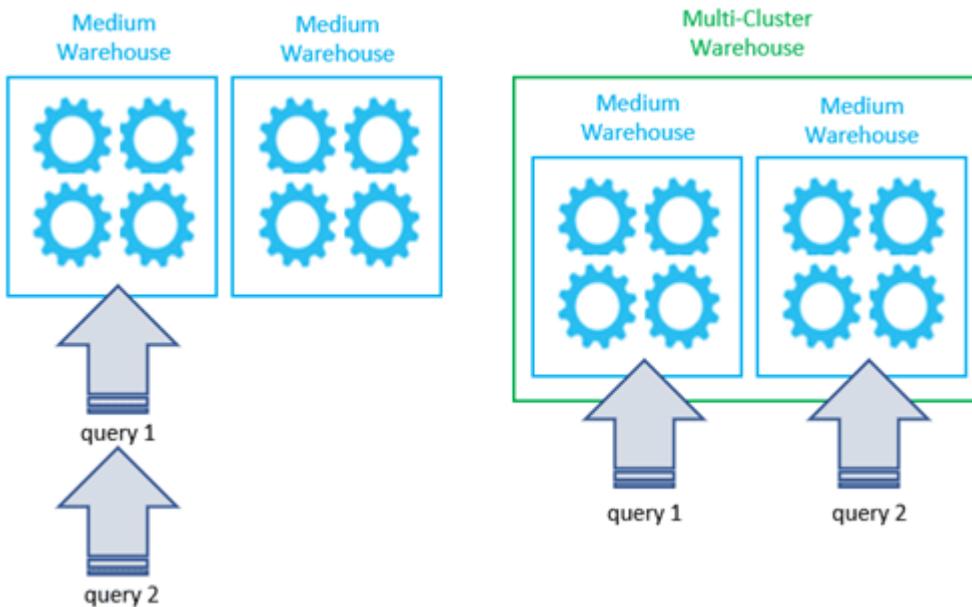
It makes no difference whether you have one size-4 (Medium) warehouse or a multi-cluster warehouse with two warehouses of the same size when running query 1. Each of these individual warehouses, each with four compute nodes, will still be assigned to the query.

- **A Query can be Processed by more than one Warehouse at the same time:** Only one virtual warehouse can process a query. A USE WAREHOUSE statement is always issued for the current session in various ways, and this will be the virtual warehouse that is running your current query. If you have a multi-cluster warehouse, your query may be run by a different cluster in this virtual warehouse. When the other clusters are busy executing other queries, queued queries can be redirected to another running cluster. Remember that a cluster is made up of N compute nodes that can be provisioned and used to run queries, with N being the “size” of the cluster. A virtual warehouse is made up of clusters like this. The virtual warehouse is defined as a collection of nodes for every single cluster. Our warehouse, on the other hand, is comprised of a single cluster of N nodes.



Even though it may require more resources for optimal processing, the previous query 1 can only be run by one warehouse. Additional nodes from a different warehouse cannot be allocated to the same query.

- **Incoming Queries are Automatically Redirected to any Available Virtual Warehouse:** Each query is executed by a single warehouse, which is either set as the current context — with USE WAREHOUSE — or set as the default for the current user. The process is always manual; there is no automatic redirection to another warehouse. Different clusters can be started in parallel for queries in the queue only in the Enterprise Edition, which has a multi-cluster virtual warehouse.



The difference between the two size-4 (Medium) warehouses on the left and the similar warehouses on the right is that the ones on the right are part of a multi-cluster warehouse, which is currently in use but can eventually scale out to accommodate more processing power. If the current context for queries 1 and 2 is the same separate warehouse, the second query will most likely be queued and executed after the first.

What are Auto-resumption and Auto-suspension?

A warehouse can be programmed to resume or suspend operations based on activity:

- Auto-suspension is turned on by default. If the warehouse is inactive for an extended period of time, Snowflake suspends it automatically.
- Auto-resume is activated by default. When any statement that requires a warehouse is submitted, Snowflake automatically resumes the warehouse, and the warehouse becomes the session's current warehouse.
- Only the entire warehouse is affected by auto-suspend and auto-resume, not individual clusters.

Snowflake - Credits

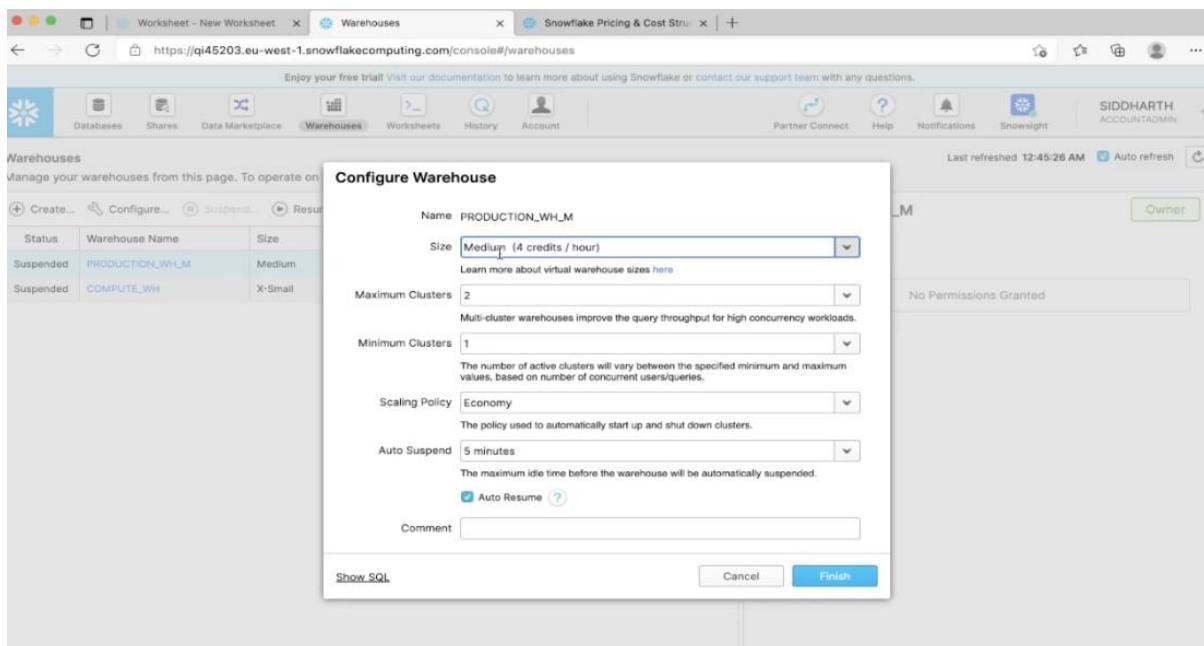
Snowflake Credits : Its a unit of measure on snowflake and its used to pay for the consumption of resources in Snowflake .

Let's say you have spent 100 Credits in the month of January , it means the amount that you or your company would have to pay is : $(100 * \text{Cost per Credit})$ in dollar .

Cost Per Credit - : <https://www.snowflake.com/pricing/>

Snowflake - Billing Components

- Storage Costs :** On-demand (~\$40 per TB/month storage) and pre-purchased storage (~\$23 per TB storage/month)
- Compute Costs :** Virtual Warehouses (XS to 5XL Large) , per-minute pricing
- Cloud Services Cost :** Authentication , infrastructure management, metadata management , query optimization & access control
- Serverless Features Cost :** Snowpipe,Database replication,Materialized View maintenance , automatic clustering



Snowflake - Warehouse/Compute Cost

- Cluster Configuration => Medium Sized Warehouse : 4 Credits/hour => 4 Credits / 60 minutes
 - Single Cluster Warehouse
 - 1 minute = 0.066 Credits
 - 30 minutes = $30 * 0.066 = 1.98$ Credits
 - Dollar Amount => $1.98 * \$3.9 = \7.72
 - Multi Cluster Warehouse (min=1,max=2)
 - 1 minute = $0.066 * 2 = 0.132$ Credits
 - 30 minutes = $30 * 0.132 = 3.96$ Credits
 - Dollar Amount => $3.96 * \$3.9 = \15.44

Snowflake - Cloud Services Cost

Date	Compute Credits Consumed	Cloud Services - Credits Consumed	Adjustments	Total Credits Billed
1st Jan	100	20	-10	110
2nd Jan	120	10	-10	120
3rd Jan	80	5	-5	80
Total	300	35	-25	310

The screenshot shows the Snowflake Worksheet interface with the following details:

Worksheet - New Worksheet

URL: <https://tn37104.eu-central-1.snowflakecomputing.com/console#/internal/worksheet>

Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

SIDDHARTH ACCOUNTADMIN

New Worksheet

Ind database objects

Starting with...

1. use role accountadmin;
2.
3. select * from snowflake.account_usage.warehouse_metering_history limit 10;
4.
5. select * from snowflake.account_usage.query_history limit 10;
6.

Results Data Preview

Filter result... Copy Open History

Row	START_TIME	END_TIME	WAREHOUSE_ID	WAREHOUSE_NAME	CREDITS_USED	USED_COMPUT	CREDITS_USED_CLOUD
1	2022-03-15 11:00:00....	2022-03-15 12:00:00....	1	COMPUTE_WH	0.369175555	0.368333333	0.000842222
2	2022-03-15 23:00:00....	2022-03-16 00:00:00....	3	PROD_XL	0.000258611	0.000000000	0.000258611
3	2022-03-15 01:00:00....	2022-03-15 02:00:00....	3	PROD_XL	2.44041944	2.440000000	0.00041944
4	2022-03-16 00:00:00....	2022-03-16 01:00:00....	3	PROD_XL	1.858610556	1.857777778	0.000832778
5	2022-03-17 09:00:00....	2022-03-17 10:00:00....	1	COMPUTE_WH	0.186038112	0.185555556	0.000480556
6	2022-03-17 10:00:00....	2022-03-17 11:00:00....	0	CLOUD_SERVICES_O...	0.000147222	0.000000000	0.000147222
7	2022-03-15 01:00:00....	2022-03-15 02:00:00....	2	PROD_XL	0.266681945	0.266666667	0.000015278
8	2022-03-15 01:00:00....	2022-03-15 02:00:00....	0	CLOUD_SERVICES_O...	0.000115278	0.000000000	0.000115278

Worksheet - New Worksheet

Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

Databases Shares Data Marketplace Warehouses Worksheets History Account

Partner Connect Help Notifications Snowsight SIDDHARTH ACCOUNTADMIN

New Worksheet New Worksheet New Worksheet New Worksheet New Worksheet

Views Masking

All Queries | Saved 27 seconds ago

```

1: use role accountadmin;
2:
3: select * from snowflake.account_usage.warehouse_metering_history limit 10;
4:
5: select * from snowflake.account_usage.query_history limit 10;
6:
7:
8: select
9:   round(sum(credits_used*3.9),2) as billed_amount,
10:  warehouse_name,
11:  timestampdiff('minute',start_time,end_time) as warehouse_run_time_minutes,
12:  date(start_time) as execution_date,
13:  hour(start_time) as execution_hour
14:  from snowflake.account_usage.warehouse_metering_history
15:  group by 2,3,4,5;
16:

```

Results Data Preview

Query ID SQL 330ms 22 rows

Row	BILLED_AMOUNT	WAREHOUSE_NAME	WAREHOUSE_RUN_TIME_MINUTES	EXECUTION_DATE	EXECUTION_HOUR
1	0.00	CLOUD_SERVICES_ONLY	60	2022-03-15	1
2	5.64	PROD_XL	60	2022-03-15	9
3	10.56	PROD_XL	60	2022-03-15	1
4	7.25	PROD_XL	60	2022-03-16	0
5	0.00	CLOUD_SERVICES_ONLY	60	2022-03-17	9
6	10.59	PROD_XL	60	2022-03-15	11
7	0.00	COMPUTE_WH	60	2022-03-16	0

Resource Monitors

Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

Databases Shares Data Marketplace Warehouses Worksheets History Account

Partner Connect Help Notifications Snowsight SIDDHARTH ACCOUNTADMIN

Create Resource Monitor

Name: daily.consumption

Credit Quota: 50

Monitor Level: ACCOUNT

Schedule: Start monitoring immediately and reset daily until . Customize

Actions and Notifications

Specify what action to perform when quota is reached.

Stay up to date: Get notified when warehouses reach your action thresholds by turning on notifications in your account preferences.

Suspend and notify when 80 % of credits is used.

Suspend immediately and notify when 95 % of credits is used.

Notify when 70 % of credits is used.

Add more notification thresholds

Show SQL Cancel Create

Snowflake - Tables

Permanent Tables : Default table types when creating tables

Temporary Tables : Type of tables used for non-permanent or transitory data . This type of tables only exist or are available within the session and are deleted once the session has finished .

Transient Tables : Similar to Permanent tables. This type of tables exist until they are explicitly dropped . Used to persist transitory data that is needed beyond a session .

- **NOTE :** Temporary and Transient tables do not have Time-travel and Fail-safe features enabled .

the different types of tables that we have in Snowflake.

The first one is if woman on tables. This is your default table type. You do not have to explicitly use any keyword while creating your tables. If you do not do that, it becomes your permanent table that we will only create a tables and all of this table that we created a permanent tables.

So there's nothing specific about it. You do not have to use anything. Next up is your temporary tables. Now, temporary tables, as the name suggests, is used to this temporary data. Unless you're doing some ad hoc analysis in and, you know, you will not need the results of this analysis for later. So what do you do? You create a temporary table out of your desk with all you ingest the results of the stool inside a temporary table. Now, if you do not want others to see what you're doing for some reason, let's say. That's also one of the reasons why you create a temporary table because. Your empty tables do not exist outside of the session. The second you log out and try to log back in, you'll notice the identity tables have disappeared.

They are not similar to template tables because changing the data in your transit tables exists outside of your station. They will be on your session as well. So if you want to delete or drop down to tables, you will have to do it explicitly.

Two types of Snowflake, which is time, travel and failsafe.

Now we have a completely, entirely different section that talks about data protection, security and governance, where we'll be covering these two features in detail. But for now, if you just want to understand the definitions but if Is a seven day period during which historical data may be recoverable by Snowflake.

Now, let's hold on to that thought. Time travel is the future using which you can go back in time and recover the data that existed for that during that time period.

Snowflake - Data Recovery & Protection

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

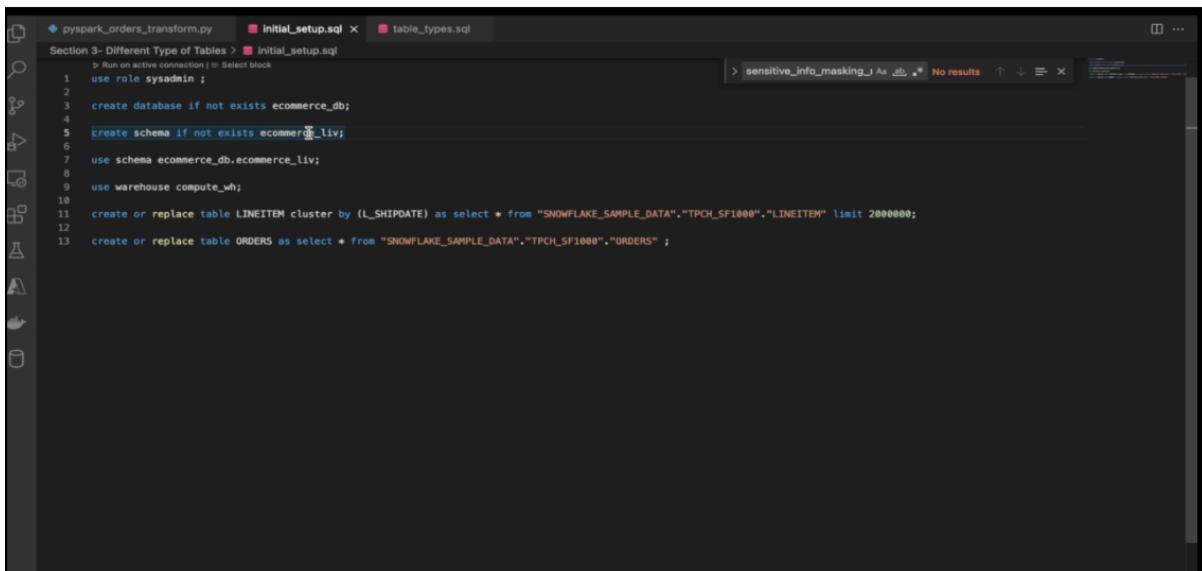
Time Travel : Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It serves as a powerful tool for performing the following tasks :

- Restoring Objects (Db/Schema/Tables)
- Backing up data from a specific time in the past

Let's say you're working on a Extremely important table that has transactions. Okay, now you accidentally drop this table So what do you do? Let's say you want the exact same table as it existed yesterday so you can time travel by one day and recover the data. This is the time to have a feature with our enterprise edition of Snowflake that we signed up for. You can travel back by 90 days at the most and recover the data. So if you want the snapshot of the database or the table that existed 89 days ago or 90 days ago, you can still do that if you signed up for the enterprise addition.

Now let's go back to feel safe. Now, failsafe is the duration or the amount of number of days. During which you can perform the time travel action. Mostly it's a seven day period. So let's say you drop the table or the database today. Now, for the next seven days, you can perform your time action.

Now the difference between permanent and transient peoples is that transient. Let's say for three days the data will be there, that you will still see the table for three days. But let's say on the fourth day, you drop your diving table. There's absolutely no way you can recover this data. So time travel and field speeches do not apply to timetables. So I hope theoretically the definitions are clear.



The screenshot shows a Snowflake interface with a code editor and a results pane. The code editor contains a SQL script named 'initial_setup.sql' with the following content:

```
use role sysadmin;
create database if not exists ecommerce_db;
create schema if not exists ecommerce_liv;
use schema ecommerce_db.ecommerce_liv;
use warehouse compute_wh;
create or replace table LINEITEM cluster by (L_SHIPDATE) as select * from "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1000"."LINEITEM" limit 2000000;
create or replace table ORDERS as select * from "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1000"."ORDERS" ;
```

The results pane shows a query titled 'sensitive_info_masking_j' with the message 'No results'.

Table Types

Snowflake categorizes tables into different types based on its uses and nature. There are four types of tables –

Permanent Table

- Permanent tables are created in the **database**.
- These tables persist until **deleted** or **dropped** from database.
- These tables are designed to store the data that requires highest level of data protection and recovery.
- These are default table type.
- Time travel is possible in these tables up to 90 days, i.e., that someone can get the data up to 90 days back.
- It is Fail-safe and data can be recovered if lost due to fail.

Temporary Table

- Temporary tables, as the name indicates, exist for a shorter duration.
- These tables persist for a session.
- If a user wants a temporary table for his subsequent queries and analytics, then once a session is completed, it automatically drops the temporary table.
- It is mostly used for transitory data like ETL/ELT
- Time travel is possible in temporary tables but only 0 to 1 day.
- It is not fail-safe, which means data cannot be recovered automatically.

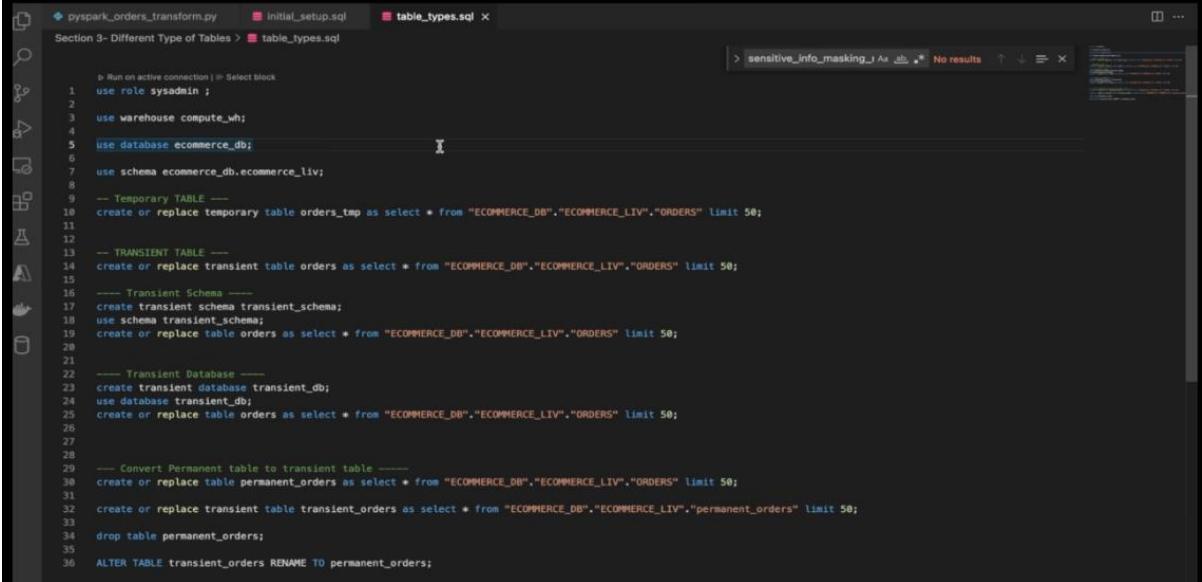
Transient Table

- These tables persist until the users drop or delete them.
- Multiple users can access a transient table.
- It is used where "data persistence" is required but doesn't need "data retention" for a longer period. For example, the details of guest visitors of a website, the details of users who visited a website as well as registered on it, so after registration, storing the details in two different tables might not be required.
- Time travel is possible in transient tables but only for 0 to 1 day.
- It is also not fail-safe.

External Table

- These tables persist until removed.
- Here, the word **removed** is used, as external tables are like outside of snowflake and they can't be dropped or deleted. It should be removed.

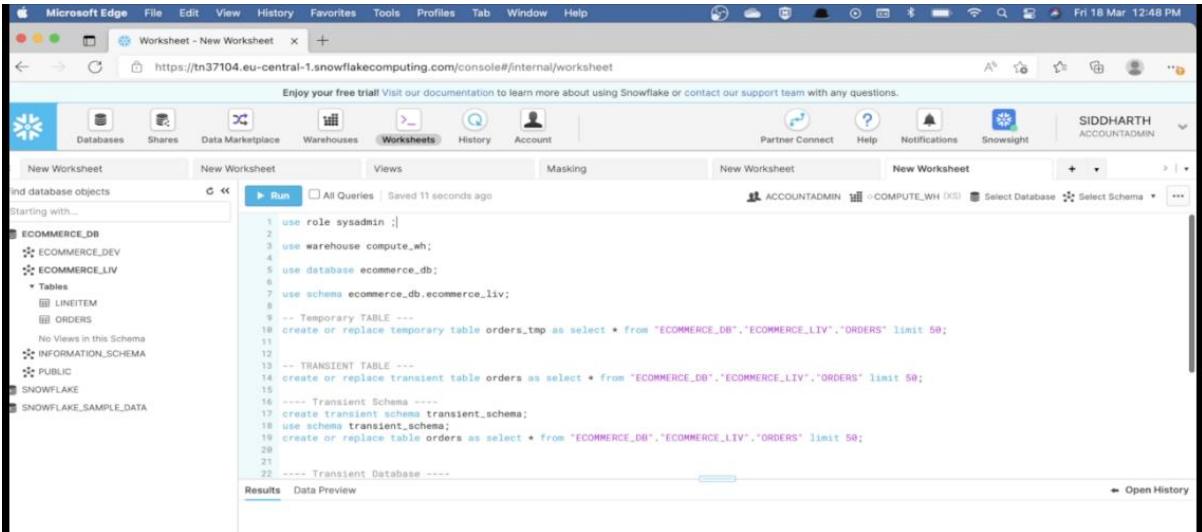
- It can be visualized as Snowflake over an external data lake, i.e., the main source of data lake is pointed to Snowflake to utilize the data as per user's need.
- Data cannot be directly accessed. It can be accessed in Snowflake via an external stage.
- External tables are only meant for reading.
- Time travel is not possible for external tables.
- It is not fail-safe inside Snowflake environment.



```

1  -- Run on active connection | == Select block
2  use role sysadmin ;
3  use warehouse compute_wh;
4
5  use database ecommerce_db;
6
7  use schema ecommerce_db.ecommerce_liv;
8
9  -- Temporary TABLE --
10 create or replace temporary table orders_tmp as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
11
12
13 -- TRANSIENT TABLE --
14 create or replace transient table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
15
16 ---- Transient Schema --
17 create transient schema transient_schema;
18 use schema transient_schema;
19 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
20
21
22 ---- Transient Database --
23 create transient database transient_db;
24 use database transient_db;
25 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
26
27
28 ---- Convert Permanent table to transient table --
29 create or replace table permanent_orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
30
31 create or replace transient table transient_orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."permanent_orders" limit 50;
32
33 drop table permanent_orders;
34
35
36 ALTER TABLE transient_orders RENAME TO permanent_orders;

```



The screenshot shows the Snowflake web interface with a browser window titled 'Worksheet - New Worksheet'. The URL is <https://tn37104.eu-central-1.snowflakecomputing.com/console#/internal/worksheet>. The page displays a 'New Worksheet' tab with the following content:

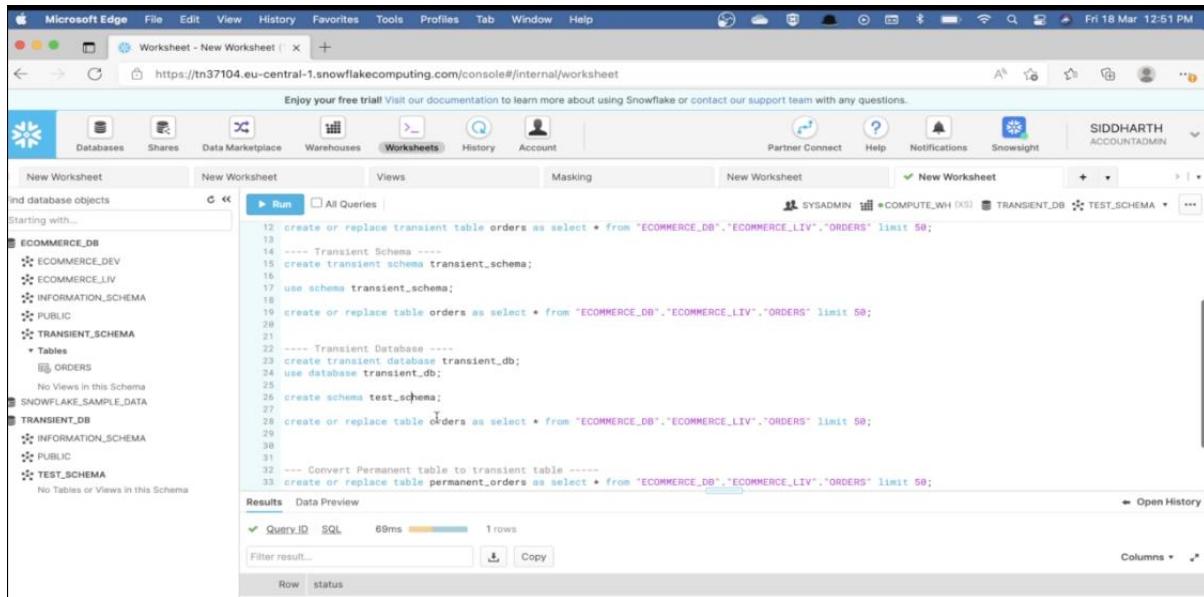
```

1  use role sysadmin ;
2  use warehouse compute_wh;
3
4  use database ecommerce_db;
5
6  use schema ecommerce_db.ecommerce_liv;
7
8  -- Temporary TABLE --
9  create or replace temporary table orders_tmp as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
10
11
12 -- TRANSIENT TABLE --
13 create or replace transient table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
14
15 ---- Transient Schema --
16 create transient schema transient_schema;
17 use schema transient_schema;
18 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
19
20
21 ---- Transient Database --
22 create transient database transient_db;
23 use database transient_db;
24 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
25
26
27 ---- Convert Permanent table to transient table --
28 create or replace table permanent_orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
29
30 create or replace transient table transient_orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."permanent_orders" limit 50;
31
32 drop table permanent_orders;
33
34
35
36 ALTER TABLE transient_orders RENAME TO permanent_orders;

```

But let's go ahead and create this. The table has been created. Now if you log out. And you login as another user assuming you have multiple users. You can go ahead and actually test this if you want. You will notice that you will not be able to see this temporary table from the other user account.

And like I mentioned, this is because your temporary table is only bound to your specific session, and it's meant for transitory data.



The screenshot shows the Snowflake Worksheet interface in Microsoft Edge. The URL is https://tn37104.eu-central-1.snowflakecomputing.com/console#/internal/worksheet. The worksheet contains the following SQL code:

```
12 create or replace transient table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
13
14 ---- Transient Schema -----
15 create transient schema transient_schema;
16
17 use schema transient_schema;
18
19 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
20
21
22 ---- Transient Database -----
23 create transient database transient_db;
24 use database transient_db;
25
26 create schema test_schema;
27
28 create or replace table orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
29
30
31
32 ---- Convert Permanent table to transient table -----
33 create or replace table permanent_orders as select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" limit 50;
```

The results pane shows a single row with the following details:

Query_ID	SQL	69ms	1 Rows

Buttons in the results pane include 'Filter result...', 'Copy', and 'Columns'.

If you create a transient database, every object inside of this database is transient by default. If you create a timeline schema, every object inside of the schema is going to be transient by default. But your database can be a permanent database, and you can have transient schemas and permanent schemas inside of it, but it doesn't go backwards.

Now the most important question. Is Why would you ever want to do this? Creating a tea, converting a table from one type to another type is actually very simple, technically. But the only reason as to why would you want to convert a permanent table to a transient table is to avoid the costs that are associated with failsafe and recovery. Now, as I mentioned in the previous video that permanent tables have time travel and failsafe recovery automatically enabled. But for transient and temporary tables, they are not. And the only difference between transient and permanent table is your time, travel and failsafe recovery.

So if you want to avoid the cost and let's say you get a permanent table and then you realize that you actually using it for ad hoc analysis or aggregation purposes, but you don't want to keep this table for too long. And you also want to avoid the cost of time, travel and failsafe. That's when you convert a permanent table to a transient table.

There are 3 types of tables in Snowflake, listed below:

1. Permanent Table
2. Temporary Table
3. Transient Table

Permanent Table - It is the default table in Snowflake, and it's possible to enable space consumption and a fail-safe timeframe.

Temporary Table - The temporary table in Snowflake is visible for the current session and exists during the session in which they are created. These tables are only available for that duration of the session and it does not support some standard functionalities.

The syntax to create a Temporary table is given below:

```
create temporary table [TABLE_NAME]([COLUMN_NAME] DATA  
TYPE,... );
```

Transient Table - Transient tables are available to all the users with the necessary privileges until they are revoked explicitly. These are similar to permanent tables except they don't have a fail-safe period. The fail-safe provides a 7 day period in which it allows users to retrieve the data. Also, these tables cannot be converted to another table type once created.

The syntax to create a Transient table is given below:

```
create Transient table emp (name VARCHAR(100), eid int, CONSTRAINT pk_emp PRIMARY Key (eid));
```

Output - status - Table EMP successfully created.

Snowflake – Views

1. A view allows the result of a query to be accessed as if it were a table
2. Views serve a variety of purposes
 - a. Combining
 - b. Segregating
 - c. protecting data
3. Views make your code more :
 - a. Clearer
 - b. More modular
 - c. Easy to decompose into smaller chunks
4. Views are of 2 types :
 - a. Non-materialized (Views)
 - b. Materialized

View Types

There are three main categorized views in Snowflake –

Standard View

- It is the default view type.
- Select queries for tables to view data.
- User can execute queries based on role and permissions.
- Underlying DDL is available to any role who has access to these view.

Snowflake – Secure Views

1. Views should be defined as secure when they are specifically designated for data privacy
2. With secure views, the view definition and details are only visible to authorized users
3. Both non-materialized and materialized views can be defined as **secure**.
4. A **Secure View** has improved data privacy and data sharing
5. Both Views and Materialized View can be clustered for better performance

Secure View

- Secure View means it can be accessed only by authorized users.
- Authorized users can view the definition and details.
- Authorized users with proper role can access these tables and execute the queries.
- In secure view, Snowflake query optimizer bypasses optimizations used for regular view.

Snowflake - Materialized Views

1. A materialized view is a pre-computed data set derived from a query specification (the SELECT in the view definition) and stored for later use.
2. Because the data is pre-computed, querying a materialized view is faster than executing a query against the base table of the view
3. Materialized views can speed up expensive aggregation, projection, and selection operations, especially those that run frequently and that run on large data sets.

Materialized View

- Materialized view is more like a table.
- These views store the result from the main source using filter conditions. For example, a company has records of all employees who are active, inactive, or deceased from starting of the company. Now, if a user needs the details of active employees only, then the main table can be queried and stored as materialized view for further analytics.
- Materialized view is auto-refreshed, i.e., whenever the main table gets additional/new employee records, it refreshes the materialized view as well.
- Snowflake supports secure materialized view as well.
- Materialized views are maintained automatically, and it can consume significant compute resources.
- Total costs for materialized views are based on "data storage + compute + server less services."
- Compute charges per materialized view are calculated based on the volume of data changes.

The screenshot shows a Snowflake SQL editor interface with several tabs at the top: 'initial_setup.sql', 'table_types.sql', 'views-materialized_views.sql' (which is the active tab), and 'create-role-to-test-security.sql'. The main area contains the following SQL code:

```
1  use role sysadmin ;
2
3  -- Change the warehouse name if need be -----
4  use warehouse prod_xl;
5  use database ecommerce_db;
6  use schema ecommerce_db.ecommerce_liv;
7
8  -- View for Orders with "Urgent" Priority --
9  create or replace view urgent_priority_orders as
10 select * from "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS" where o_orderpriority='1-URGENT'
11
12
13 -- Create a materialized view --
14 create or replace materialized view vw_aggregated_orders as
15 select
16   count(1) as total_orders,
17   o_ORDERSTATUS as order_status ,
18   o_ORDERDATE as order_date
19 from
20   "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS"
21 where o_orderpriority='1-URGENT'
22 group by 2,3;
23
24
25 -- Create a secure materialized view --
26 create or replace materialized view secure_vw_aggregated_orders as
27 select
28   count(1) as total_orders,
29   o_ORDERSTATUS as order_status ,
30   o_ORDERDATE as order_date
31 from
32   "ECOMMERCE_DB"."ECOMMERCE_LIV"."ORDERS"
33 where o_orderpriority='1-URGENT'
34 group by 2,3;
35
36
37 ----- Add a clustering key to the view
38 ALTER MATERIALIZED VIEW vw_aggregated_orders CLUSTER BY (order_date);
39 ALTER MATERIALIZED VIEW secure_vw_aggregated_orders CLUSTER BY (order_date);
40
41
```

The screenshot shows a SQL editor interface with a sidebar containing various icons. The main area displays a script titled "create-role-to-test-security.sql". The script content is as follows:

```
use role accountadmin;
use schema "ECOMMERCE_DB"."ECOMMERCE_LIV";
create or replace role view_role;
grant usage on warehouse compute_wh to role view_role;
grant usage on database ECOMMERCE_DB to role view_role;
grant usage on schema ECOMMERCE_LIV to role view_role;
grant select on "ECOMMERCE_DB"."ECOMMERCE_LIV"."SECURE_VW_AGGRAGATED_ORDERS" to role view_role;
grant select on "ECOMMERCE_DB"."ECOMMERCE_LIV"."URGENT_PRIORITY_ORDERS" to role view_role;
grant select on "ECOMMERCE_DB"."ECOMMERCE_LIV"."VW_AGGRAGATED_ORDERS" to role view_role;
grant role view_role to user siddharth;
-- Test the GET_DDL using the new role created above -----
use role view_role;
use schema "ECOMMERCE_DB"."ECOMMERCE_LIV";
select * from SECURE_VW_AGGRAGATED_ORDERS limit 20;
select get_ddl('view','URGENT_PRIORITY_ORDERS');
select get_ddl('view','SECURE_VW_AGGRAGATED_ORDERS');
```

This screenshot shows the same SQL editor interface, but it includes a results pane on the right side. The results pane displays the output of the "get_ddl" command from the previous screenshot, showing the DDL for the "URGENT_PRIORITY_ORDERS" view.

Some of the internal optimizations for views require access to the underlying data in the base tables for the view. This access might allow data that is hidden from users of the view to be exposed through user code, such as user-defined functions, or other programmatic methods. Secure views do not utilize these optimizations, ensuring that users have no access to the underlying data.

In addition, by default, the query expression used to create a standard view, also known as the view definition or text, is visible to users in various commands and interfaces.

For security or privacy reasons, you might not wish to expose the underlying tables or internal structural details for a view. With secure views, the view definition and details are only visible to authorized users (i.e. users who are granted the role that owns the view).

When Should I Use a Secure View?

Views should be defined as secure when they are specifically designated for data privacy (i.e. to limit access to sensitive data that should not be exposed to all users of the underlying table(s)).

Secure views should **not** be used for views that are defined for query convenience, such as views created for simplifying querying data for which users do not need to understand the underlying data representation. This is because the Snowflake query optimizer, when evaluating secure views, bypasses certain optimizations used for regular views. This might result in some impact on query performance for secure views.

Lets check with help of an example : We already have employee table :

The screenshot shows the Snowflake SQL interface. In the top-left, there's a 'Run' button and a status bar indicating 'All Queries | Saved 0 seconds ago'. On the right, it shows the user 'DEV_POWERUSER' and database 'DWEDHD (M)'. Below the status bar, a query is run:

```
1 select * from employee;
```

The results section shows the following data:

Row	ID	NAME	SALARY
1	101	rahul	1000
2	102	karan	1000
3	104	ank	1000
4	101	rahul	1000
5	102	karan	1000
6	104	ank	1000

At the bottom, there's a taskbar with various icons and system information like '70°F Mostly cloudy', 'ENG', and '9:23 AM'.

— Now , we will create a secure view where a USER who has login to snowflake account will be able to access only his information from employee table

The screenshot shows the Snowflake SQL interface. In the top-left, there's a 'Run' button and a status bar indicating 'All Queries | Saved 35 seconds ago'. On the right, it shows the user 'DEV_POWERUSER' and database 'DWEDHD (M)'. Below the status bar, a query is run to create a secure view:

```
6
7 create secure view vw_emp as
8 select * from employee where upper(name)=left(current_user(),5);
9
10
11
```

The results section shows the following message:

Row	status
1	View VW_EMP successfully created.

— Select the data from Secure view

As we have login with USER RAHUL.TANDON . This user will be able to access the records with name=RAHUL

The screenshot shows a database query interface. At the top, there is a toolbar with a 'Run' button, an 'All Queries' dropdown, and a 'Saved 0 seconds ago' message. To the right of the toolbar are user icons and names: DEV_POWERUSER, DWEDHD (M), EDHD, and DATARAW. Below the toolbar is a code editor window containing the following SQL:

```
13  
14  
15  
16 select * from vw_emp;  
17  
18
```

Below the code editor is a results table. The table has three tabs: 'Results' (which is selected), 'Data Preview', and 'Open History'. The 'Results' tab shows the following data:

Row	ID	NAME	SALARY
1	101	rahul	1000
2	101	rahul	1000

At the bottom of the results table are buttons for 'Filter result...', 'Copy', and 'Columns'.

— Lets try to use filter in VIEW to access other employee data :

The user will not be able to access any other employee's information

The screenshot shows a database query interface. At the top, there is a toolbar with a 'Run' button, an 'All Queries' dropdown, and a 'Saved 0 seconds ago' message. To the right of the toolbar are user icons and names: DEV_POWERUSER, DWEDHD (M), EDHD, and DATARAW. Below the toolbar is a code editor window containing the following SQL:

```
15  
16 select * from vw_emp where name <> 'rahul';  
17  
18
```

Below the code editor is a results table. The table has three tabs: 'Results' (which is selected), 'Data Preview', and 'Open History'. The 'Results' tab shows the following data:

Row	ID	NAME	SALARY
1	101	rahul	1000
2	101	rahul	1000

At the bottom of the results table are buttons for 'Filter result...', 'Copy', and 'Columns'.

Snowflake - Advantages of Materialized Views

1. Materialized views can improve the performance of queries that use the same subquery results repeatedly
2. Materialized views are automatically and transparently maintained by Snowflake.
3. Data accessed through materialized views is always current, regardless of the amount of DML that has been performed on the base table

I believe most of us already know this materialized views improve your quality performance that use the same subquery results repeatedly because they possess the data on the disk, they are automatically and transparently maintained by Snowflake. So you do not have to worry about ensuring the data is updated at all times.

Data access to marginalised views is always current regardless of the amount of DML operations that have been performed on the base table.

So those are the advantages of using a materialized view. Now the next big question that comes up often is when should you go for medicalized views and eventually go for normal views? So I put together three points each that will help you decide if you ever have this confusion of choosing between the two. So you create a materialized view.

Snowflake - When to create Materialized Views?

1. Create a **materialized** view if :
 - a. The query results from the view don't change often
 - b. The results of the view are used often
 - c. The query consumes a lot of resources
2. Create a **non-materialized** (regular) view if :
 - a. The results change often
 - b. The results of the view are not used often
 - c. The underlying query is not resource intensive

If the query results from the view don't change often, and these query results are also used often, and if your query consumes a lot of resources as well, so if all of these three points check, you go for the materialized view. You create a non-material life or a simple view if the results are changing often and the results of this view are not used often. Finally, if your quarry is not resource intensive, now, if these three points check, you go for the non-material view.

That cannot be an overlap between any two points between view and materialise view.

Snowflake – Secure Views

1. Create a **secure view** if :
 - a. The results are designated for data privacy
 - b. There is a need to limit the access to sensitive data in the underlying table
2. Do Not Create a **secure view** for query convenience or code-reusability
3. Snowflake query optimizer bypasses certain optimizations when it comes to **secure views**
4. Secure views are not as performant as other views

Now when it comes to security, it's also important as to know when you should go for the security so you create a secure view. If your results are designated for data privacy and there's a need to limit the access to sensitive data in the underlying tables, keep in mind you do not create a secure view for quality, convenience or usability because the Snowflake Query Optimizer will bypass certain optimization techniques when it comes to secured views. And you will realize if you create a secure view, it is not as performant as the other views. So just don't create a secure view because it sounds secure. You create it only if. It's meant for data privacy purposes.

Traditional databases allows partitioning of large tables where the data in table divided into segments, called partitions, that make it easier to manage and query your data. This is referred to as static partitioning which requires you to include a partitioning clause in the CREATE TABLE statement to create a partitioned table.

The Snowflake Data Platform, as opposed to a traditional data warehouse, uses a unique and distinctive method of partitioning called **Micro-partitioning** that offers all the benefits of static partitioning without the known drawbacks and offers extra substantial advantages.

2. What are Micro-partitions?

All data in Snowflake is stored in database tables, logically structured as collections of columns and rows. Although this is how the front-end functions, Snowflake does not actually handle and store data in this manner. Instead, Snowflake stores all table data automatically divided into encrypted compressed files which are referred to as **Micro-partitions**.

Micro-partitions are **immutable** files i.e. cannot be modified once created and stores data in columnar format. Every new data arrival creates a new micro-partition. Each micro-partition contains between 50 MB and 500 MB of uncompressed data.

Micro-partitioning is automatically performed on all Snowflake tables. The order in which the data is put or loaded is used to transparently partition the tables.

3. Query Pruning

Snowflake also stores metadata about all rows stored in a micro-partition. The metadata information includes

- The range of values for each of the columns in the micro-partition.
- The number of distinct values.
- Additional properties used for both optimization and efficient query processing.

This metadata is an essential component of the Snowflake architecture because it enables queries to decide whether or not to query the data included in a micro-partition. In this manner, only the micro-partitions containing the relevant data are queried when a query is executed, saving time and resources. This process is known as **Query Pruning**, as the data is pruned before the query itself is executed.

4. How does the organization of data into micro-partitions impact query performance?

The performance of pruning greatly depends on how data is divided into micro-partitions. Let us understand with an example provided by the Snowflake's own documentation.

Table: t1

Logical Structure				Physical Structure			
type	name	country	date	type	name	country	date
2	A	UK	11/2	type	Micro-partition 1 (rows 1-6)		
4	C	SP	11/2		2	4	3
3	C	DE	11/2		2	3	2
2	B	DE	11/2		Micro-partition 2 (rows 7-12)		
3	A	FR	11/2		3	2	4
2	C	SP	11/2		5	1	5
3	Z	DE	11/2	name	Micro-partition 3 (rows 13-18)		
2	B	UK	11/2		2	4	2
4	C	NL	11/2		1	5	3
5	X	FR	11/3		Micro-partition 4 (rows 19-24)		
1	A	NL	11/3		X	Z	Y
5	A	FR	11/3		B	X	A
2	X	FR	11/2	country	Micro-partition 1 (rows 1-6)		
4	Z	NL	11/2		UK	SP	DE
2	Y	SP	11/2		DE	FR	SP
1	B	SP	11/3		FR	NL	FR
5	X	DE	11/3		11/2	11/2	11/2
3	A	UK	11/4		11/2	11/3	11/3
1	C	FR	11/3	date	11/2	11/2	11/4
4	Z	NL	11/4		11/3	11/3	11/4
5	Y	SP	11/4		11/3	11/3	11/5
5	B	SP	11/5		11/5	11/5	11/5
3	X	DE	11/5		11/5	11/5	11/5
2	Z	UK	11/5		11/5	11/5	11/5

Logical vs Physical Structure of a Snowflake table

This image may initially be difficult to understand. Let's analyse it step by step, beginning with the left side. This is the way a table might show up in the Snowflake user interface. This is referred to as the **Logical Structure**. There are 24 rows of data and 4 columns – type, name, country, and date.

An illustration of how this information would be kept in backend of Snowflake's architecture can be seen on the right. This is referred to as the **Physical Structure**. Here, there are four distinct micro-partitions that each hold six rows of data. Data of rows 1 through 6 are contained in the first micro-partition and so on. Since the data is stored in columnar format, Snowflake can retrieve certain columns of data without dissecting individual rows.

Row 2 and row 23 are two particular rows of data that have been highlighted to see how they are stored in logical vs physical structure.

Now consider that you wanted to retrieve records that belong to date '11/2' from the above example.

```
SELECT type, country FROM MY_TABLE
```

```
WHERE Date = '11/2';
```

The data of the records that belong to date '11/2' is present in micro-partitions 1, 2 and 3. Similarly for other dates, the data is spread across micro-partitions is as depicted below.

Date	Micro-Partitions
11/2	1,2,3
11/3	2,3,4
11/4	3,4
11/5	4

When you execute this query in Snowflake, only those micro-partitions are scanned which satisfies the **filter** condition. Once the micro-partitions to be scanned are identified, a portion of the micro-partitions is scanned that contain the data for the columns in the **select** query so that an entire partition is not scanned.

As the query performance is directly linked to the amount of the data that it scans, organizing data in micro-partitions yields better overall performance which could be done using the concept called Data Clustering.

5. What is Data Clustering?

Clustering is a technique used to organize data storage to better accommodate anticipated queries. The key objective is to increase query performance while lowering the amount of system resources needed to run queries.

Snowflake supports explicitly selecting the columns on which a table is clustered, giving you additional control over the clustering process. These columns are known as **Clustering keys**, and they allow Snowflake to maintain the clustering in accordance with the chosen columns while also allowing you to recluster on demand.

Although clustering can substantially improve the performance and reduce the cost of some queries, the compute resources used to perform clustering consume credits. So, you should cluster only when the table contains multiple terabytes (TB) of data with a large number of micro-partitions and queries will benefit substantially from the clustering.

Let us take the earlier example we discussed and see how clustering the table on date column would have distributed data into micro-partitions using below image.



Physical Structure of a Snowflake table after Clustering on *date* field

The below table shows distribution of rows with distinct date values in various micro-partitions. This clearly shows clustering the table on date columns reduced the number of micro-partitions to be scanned than earlier scenario.

Date	Micro-Partitions
11/2	1,2
11/3	3
11/4	4
11/5	4

This example is just a simple conceptual representation of the data clustering that Snowflake utilizes in micro-partitions. In real world clustering table with such a small data set would not have any noticeable impact. Extrapolated to a very large table (i.e. consisting of millions of micro-partitions or more), clustering can be very effective.

6. How to create Clustered Tables in Snowflake?

In the above example, we clustered our dataset based on the *date* field. As we have clustered based on this field, this field is referred to as the clustering key.

A clustering key can be defined when a table is created by appending a **CLUSTER BY** clause to **CREATE TABLE** as shown below

```
CREATE TABLE MY_TABLE (
    type NUMBER
    , name VARCHAR(50)
    , country VARCHAR(50)
    , date DATE
)
CLUSTER BY (date);
```

In the above example we have created a table with clustered key. However you can also cluster a table by specifying a **CLUSTER KEY** at a later point in time using **ALTER TABLE** as shown below.

```
ALTER TABLE MY_TABLE
CLUSTER BY (date);
```

Although we have just used one field as a clustering key in our example, this is not the only choice. A clustering key can be defined using many fields if desired as shown below.

```
ALTER TABLE MY_TABLE
CLUSTER BY (date, type);
```

Another advantage is that Snowflake also supports expressions on fields in a cluster key. The below example shows the table is clustered using the year of the *date* field and first two letters of *country* field.

```
ALTER TABLE MY_TABLE
CLUSTER BY (YEAR(date), SUBSTRING(country,1,2));
```

7. Reclustering

A clustered table's data may become less clustered as DML operations (INSERT, UPDATE, DELETE, MERGE, COPY) are carried out on it. The table must be periodically or regularly reclustered in order to maintain optimal clustering.

Changing the clustering key for a table using ALTER TABLE does not affect existing records in the table until the table has been reclustered by Snowflake.

During reclustering, Snowflake uses the clustering key for a clustered table to reorganize the column data, so that related records are relocated to the same micro-partition. Each time data is reclustered, the rows are physically grouped based on the clustering key for the table, which results in Snowflake generating new micro-partitions for the table.

Reclustering in Snowflake is automatic and no maintenance is needed. However, the rule does not apply to tables created by cloning from a source table that has clustering keys.

This reclustering results in storage costs because the original micro-partitions are marked as deleted, but retained in the system to enable [Time Travel](#) and [Fail-safe](#).

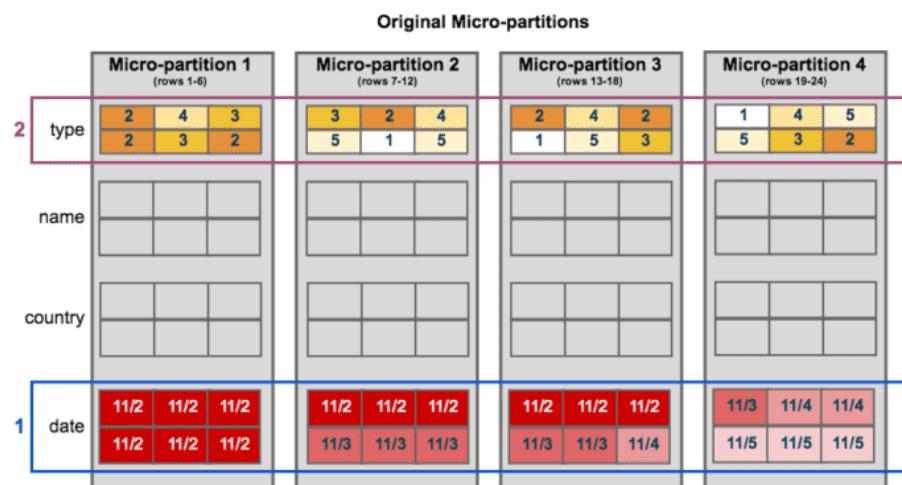
The below image shows how reclustering the data on *date* and *type* fields creates new micro-partitions based on Clustering key defined.

Table: t1

Logical Structure

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

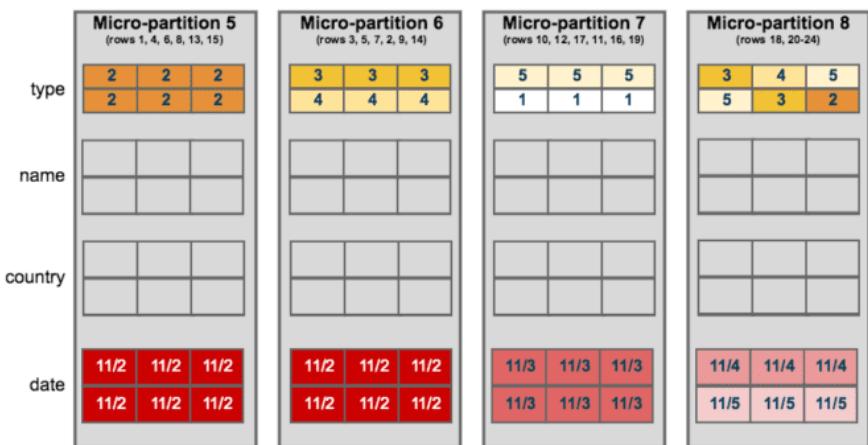
Physical Structure



```
ALTER TABLE t1  
CLUSTER BY (date, type);
```

```
SELECT name, country FROM t1  
WHERE type = 2  
AND date = '11/2';
```

New Micro-partitions (After Reclustering)



8. Automatic Clustering

Automatic Clustering is the Snowflake service that seamlessly and continually manages all reclustering, as needed, of clustered tables. Note that Automatic Clustering consumes Snowflake credits, but does not require you to provide a virtual warehouse. Instead, Snowflake internally manages and achieves efficient resource utilization for reclustering the tables.

To suspend Automatic Clustering for a table, use the **ALTER TABLE** command with a **SUSPEND RECLUSTER** clause. For example:

ALTER TABLE my_table

SUSPEND RECLUSTER ;

To resume Automatic Clustering for a clustered table, use the **ALTER TABLE** command with a **RESUME RECLUSTER** clause. For example:

```
ALTER TABLE my_table
```

```
RESUME RECLUSTER ;
```

Verify if Automatic Clustering is enable for a table using **SHOW TABLES** command.

```
SHOW TABLES LIKE 'my_table';
```

9. Clustering Information

Snowflake has provided a system function named **SYSTEM\$CLUSTERING INFORMATION** to help in determining how well-clustered a table is. This function accepts a table name and a list of columns as inputs and returns a summary of how well the table is clustered based on the list of columns.

If no list of columns is provided, the function will instead return the clustering information for the table based on its current clustering key. If no current clustering key is defined on the table, the function will error.

The syntax to execute the function is as shown below

```
SELECT SYSTEM$CLUSTERING_INFORMATION('my_table', '(date, type)');
```

The function returns a JSON object containing the following name/value pairs:

9.1. cluster_by_keys

- Columns in table used to return clustering information. The columns which you pass as arguments to the function.

9.2. total_partition_count

- Total number of micro-partitions that comprise the table.

9.3. total_constant_partition_count

- Total number of micro-partitions for which the value of the specified columns have reached a constant state i.e. the micro-partitions will not benefit significantly from reclustering. As this number increases, we can expect query pruning to improve and queries to execute more efficiently.

9.4. average_overlaps

- The term “**Overlap**” indicates the number of partitions which share the same specified column value.
- For each micro-partition in the table, this gives the average number of overlapping micro-partitions. A high number indicates the table is not well-clustered.

9.5. average_depth

- The term “**Depth**” indicates the number of partitions in which same column value exists when an overlap occurs.
- For each micro-partition in the table, this gives the average overlap depth. A high number indicates the table is not well-clustered.
- This value is also returned by **SYSTEM\$CLUSTERING_DEPTH**.

9.6. partition_depth_histogram

- A histogram depicting the distribution of overlap depth for each micro-partition in the table. The histogram contains buckets with widths:
 - 0 to 16 with increments of 1.
 - For buckets larger than 16, increments of twice the width of the previous bucket (e.g. 32, 64, 128, ...).

I have included below an example provided by snowflake showing output of **system\$clustering_information**

```
{  
  "cluster_by_keys" : "(COL1, COL3)",  
  "total_partition_count" : 1156,  
  "total_constant_partition_count" : 0,  
  "average_overlaps" : 117.5484,  
  "average_depth" : 64.0701,  
  "partition_depth_histogram" : {  
    "00000" : 0,  
    "00001" : 0,  
    "00002" : 3,  
    "00003" : 3,  
    "00004" : 4,  
    "00005" : 6,  
    "00006" : 3,  
    "00007" : 5,  
    "00008" : 10,
```

```

    "00009" : 5,
    "00010" : 7,
    "00011" : 6,
    "00012" : 8,
    "00013" : 8,
    "00014" : 9,
    "00015" : 8,
    "00016" : 6,
    "00032" : 98,
    "00064" : 269,
    "00128" : 698
}

}

```

This might be a lot of information to take in at the moment. But based on data do you think the table is well clustered?

To answer the question, every metric indicates that the table is not well clustered. The metric values which are expected to be high are low and vice versa.

Snowflake: Micro Partitions, Clustering Keys & dbt



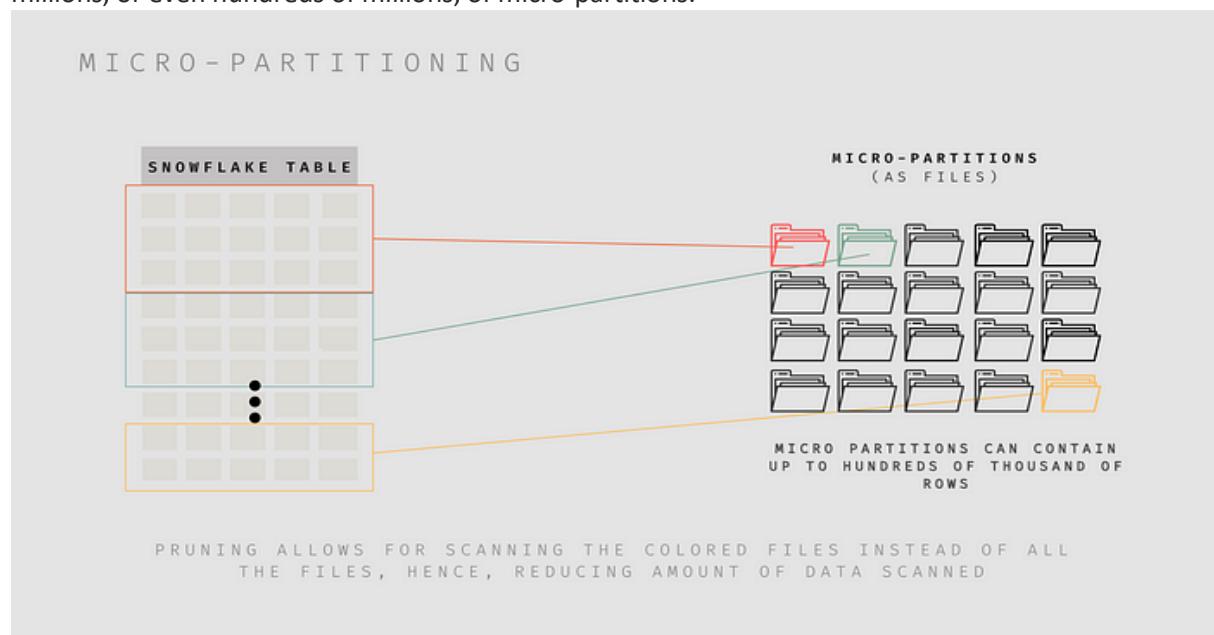
This is a bit of deep dive on the Snowflake's table structures. We will be looking at the following:

1. Important Definitions such as Micro-Partitioning, Pruning and Clustering keys
2. Strategies and notes when implementing clustering keys
3. Using Clustering Keys in dbt

Introduction & Pretext:

Snowflake uses a unique form of partitioning that is different than traditional data warehouses called '**Micro-Partitioning**', but what does that mean? Well, here is the official definition of micro partitioning taken from the [Snowflake Docs](#):

All data in Snowflake tables is automatically divided into micro-partitions, which are contiguous units of storage. Each micro-partition contains between 50 MB and 500 MB of uncompressed data (note that the actual size in Snowflake is smaller because data is always stored compressed). Groups of rows in tables are mapped into individual micro-partitions, organised in a columnar fashion. This size and structure allows for extremely granular pruning of very large tables, which can be comprised of millions, or even hundreds of millions, of micro-partitions.

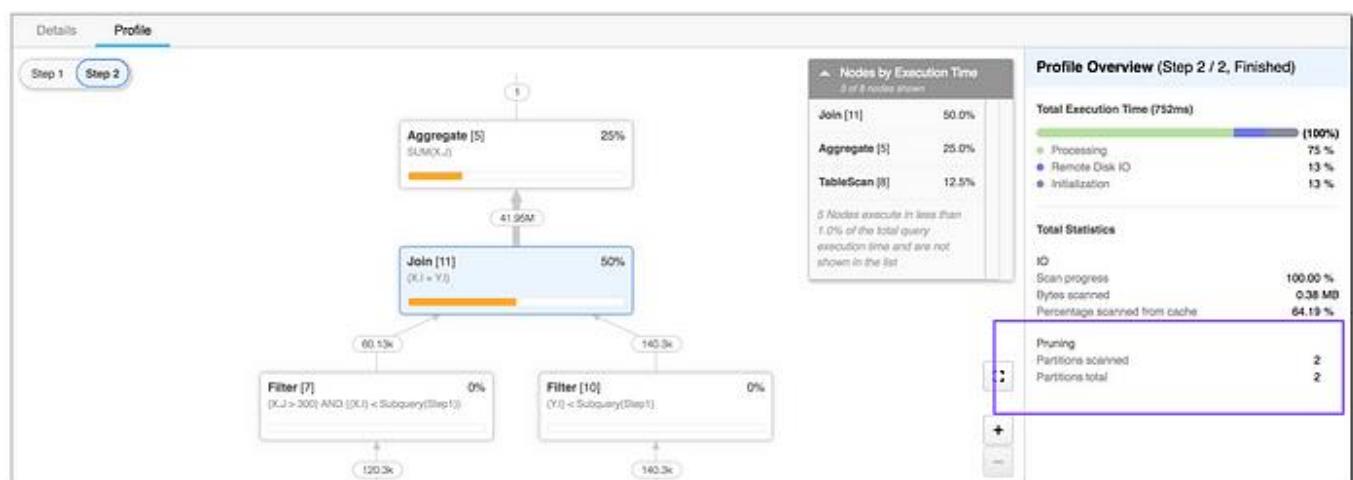


Visualising Micro-Partitions & Pruning

So in simple terms, *micro-partitions* are like immutable files (units of storage in Snowflake), each file is mapped to a group of rows. When you have micro-partitions, you allow for pruning. Pruning is a technique in snowflake, that allows queries to scan less micro partitions. Pruning helps reduce the amount of data scanned, hence optimising the query performance on a table. In addition, Snowflake stores metadata on all rows stored within a micro-partition, such as min and max values within a partition, count of total and distinct values, number of NULL entries. These are used to maximise query performance.

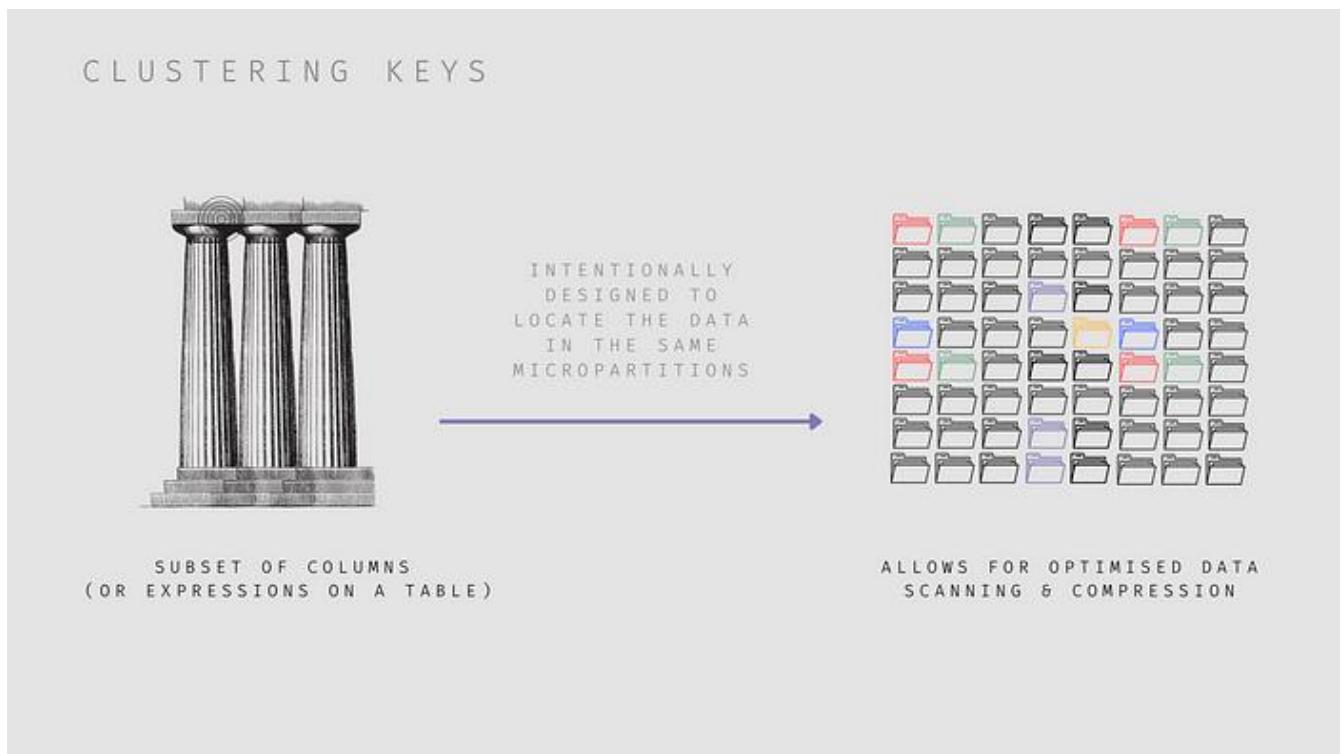
What if a certain table's query **performance degrade overtime**? what if we want to further **optimise the performance and costs**? Well, we can be a bit more *intentional* in setting up what the make up of micro partitions are by defining **Clustering Keys** that will help us achieve those optimisation and cost reduction goals.

Hint: You can always look at the *Snowflake Query Profile* to identify inefficient pruning and identify potential bottlenecks in your query. The efficiency of pruning can be derived by observing *Partitions scanned* and *Partitions total* statistics in the **TableScan** operators shown below. We always want a smaller *partitions scanned* number in relation to *partitions total*. If that is not the case, you can consider your pruning inefficient. Example of the interface from Snowflake docs:



Query Profile: Pruning on bottom right

Clustering keys in Snowflake



What are Clustering Keys?

Clustering keys behave like indexes in columnar databases, which allows to avoid costly scans by providing filtered scans using pruning techniques and better compression. Check out this visualisation taken from Snowflake docs, for a more visual representation:

Logical Structure					Physical Structure				
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						

A simplistic representation of data clustering : “The table consists of 24 rows stored across 4 micro-partitions, with the rows divided equally between each micro-partition. Within each micro-partition, the data is sorted and stored by column”

Adding clustering keys could really help optimise for a degraded table performance and long query times. Some reasons for these issues:

extensively large/growing underlying table, non ideal ordering of data at the time the data was inserted ,extensive DML caused the table’s **Natural Clustering** (e.g. think of the data insertion timestamp in the ETL process being used as a clustering key) to degrade over time.

Optimising for costs can also come from reducing table scans as mentioned before, or sometimes if you are using out of the shelf solutions such as [Automatic Clustering](#) , costs can become unpredictable and you would want to optimise for that too. However, it is very important to note that clustering keys are not intended for all tables, there are certain criteria that tables should adhere to.

Note: The Initial/Automatic clustering will result in snowflake credits usage. Re-clustering will result in consuming credits too so here are things to note:

THINGS TO NOTE :

- ⚠ CLUSTERING KEYS ARE NOT INTENDED FOR ALL TABLES
- ✓ SIZE OF THE TABLE & QUERY PERFORMANCE SHOULD BE THE MAIN INDICATOR OF USING CLUSTERING KEYS, PARTICULARLY IF DML IS PERFORMED FREQUENTLY
- ✓ THE COLUMNS DEFINED AS CLUSTERING KEYS ARE SUFFICIENT FOR FILTERING SUBSETS OF MICRO PARTITIONS
- ✓ SOMETIMES TESTING A REPRESENTATIVE SETS OF QUERIES ON A TABLE ESTABLISHES PERFORMANCE BASELINES THAT CAN HELP WHEN CHOOSING A CLUSTERING KEY

Things to Note around Clustering Keys

Some Strategies to Choosing your Clustering Keys

Snowflake strongly recommends to test a representative set of queries on the table to get some performance baselines **BEFORE** choosing clustering keys. You can then use some [Snowflake system clustering function](#) & [clustering depth](#) & [its function](#) for analysis.

A Single Clustering key can contain one or more columns/Expressions

- Snowflake Recommends 3–4 Columns Max per key
- Selecting the right columns will significantly Improve Performance
- Adding More than 3–4 keys tends to increase **Cost** providing less benefits

How to choose your clustering keys:



CHOOSING CLUSTERING KEYS

CLUSTER COLUMNS THAT ARE MOST ACTIVELY USED IN YOUR SELECT STATEMENTS FILTERS
(THINK CATEGORIES, TYPES, DATE ..ETC,
THEN CONSIDER COLUMNS USED FREQUENTLY IN JOINS)

CARDINALITY OF A KEY MATTERS (NUMBER OF DISTINCT VALUE) :
MUST HAVE:

- LARGE ENOUGH VALUES FOR EFFECTIVE PRUNING
- SMALL ENOUGH DISTINCT VALUES TO GROUP ROWS IN THE SAME MICRO PARTITION

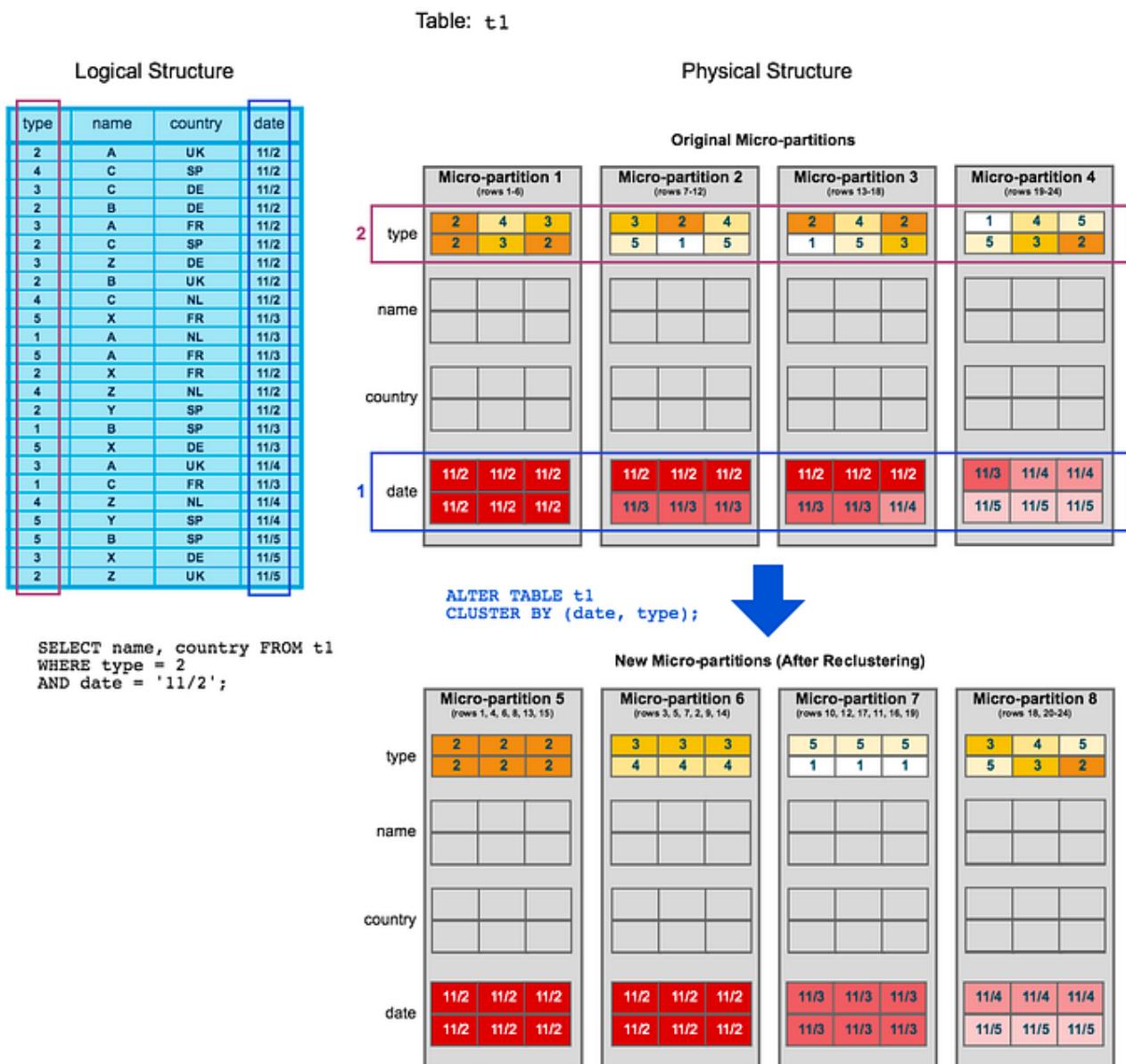
TOO LARGE (E.G UUID) & TOO LITTLE (E.G ACTIVE VS INACTIVE STATUS) CARDINALITY IS A NO GO

Choosing Clustering Keys

Note: When choosing a `VARCHAR` type column as a clustering key , Snowflake will take into account the first 5 bytes. So you could consider using an expression such as `substr()` to be more intentional.

In addition, pay attention to timestamps and cardinality. Consider casting timestamps to dates.

Snowflake has a really good visualisation on what happens when you ‘Re-cluster’, taken from their [documentation here](#):



Re-clustering visualisation of Micro-partitions. Notice the new micro partitions and how they are organised

To start, table t1 is naturally clustered by date across micro-partitions 1-4. The query (in the diagram) requires scanning micro-partitions 1, 2, and 3. date and type are defined as the clustering key. When the table is reclustered, new micro-partitions (5-8) are created. After reclustering, the same query only scans micro-partition 5.

My First Success Story

I had a table, relatively wide, with about billion rows and was subjected to heavy use of aggregation on Looker's semantic layer. Looker's semantic layer acts as a *view* on top of the snowflake *table*. The most used 'Look' (a single visualisation) that reports based on the said table, usually takes 6 to 7 minutes to load. As you can imagine, since this visualisation is heavily used by numerous people, we need to prioritise optimising that.

I decided to add clustering keys to my dbt model based on the most used *where* filters and what is used in the joins.

The result: *Loading times decreased from 6–7 minutes to 30–40 seconds.*

Ways to Implement Clustering Keys

A clustering key can be defined in at the time of table creation , using the **CREATE TABLE** command or later on using the **ALTER** command.

```
/* clustering by a column */
alter table your_table_name cluster by (clustering_column_name);

/* clustering by an expression */
alter table your_table_name cluster by (left(clustering_column_name,2));
```

Clustering Key with dbt

You can also use **dbt model configuration** to define the clustering keys. That is what I usually use and it can look something like this:

```
 {{config(  
     materialized='table',  
     unique_key = 'unique_id',  
     cluster_by=['category','city','created_at']  
)  
}}  
  
WITH first_cte AS (  
  
    SELECT  
  
        user_id,  
        transaction_id,  
        unique_id,  
        category, -- 60 distinct values  
        city, -- 200 distinct values  
        created_at, -- date (daily) column  
        revenue  
  
    FROM {{ ref('raw_customer_txs') }}  
    WHERE 1=1  
  
,  
...  
,
```

Notice in the code snippet above the **cluster_by** field in the model configuration. this is how you can define your chosen clustering keys.

Another Important thing to note, is **the order** of columns in the **cluster_by configuration**, we start with the columns that have the *lowest* cardinality (unique values) which is '**category' (60 values)**, followed by **city (200 distinct values)** and so on. In this example, these clustering keys are

used frequently in joins across different data models, and are heavily used in filtering by business stakeholders using BI tools.

Snowflake - Partitions

- **Partitioning** is a process of dividing a huge dataset/table into smaller chunks so that each partition of data can be manipulated independently
- In snowflake the data inside the table **automatically** gets divided into smaller chunks of data in the range of 50MB to 500MB each called micro-partitions
- These micro-partitions do not need to be defined explicitly & can overlap in the range of values
- The metadata for each of these partitions is stored in the cloud services layer
 - The metadata contains :
 - Range of values for each of the columns in micro-partitions
 - Number of unique values
 - Additional properties required for efficient query processing and data scanning

Snowflake - Clustering

- Clustering is a process to optimize data retrieval
- Clustering is performed on micro-partitions to ensure similar data reside in the same micro-partition which can be fetched in a single query
- Clustering Key is a column or a group of columns that are designated to locate the data in the same micro-partition
- Clustering keys play a key role in ensuring the data is sorted/ordered inside the micro-partitions which especially helps while querying large tables
- Pruning is a process where snowflake avoids scanning the unnecessary micro-partitions using the clustering keys in the “where clause” thus resulting in better performance

NOTE :

- Snowflake automatically splits the table into clustered micro-partitions using the natural dimensions such as date columns
- However , these clustered micro-partitions might not be necessarily ideal in a long run.

The screenshot shows the Microsoft Edge browser displaying the Snowflake Worksheet interface at <https://q45203.eu-west-1.snowflakecomputing.com/console#/internal/worksheet>. The user is Siddharth AccountAdmin. The left sidebar shows databases like ECOMMERCE_DB, SNOWFLAKE, and SNOWFLAKE_SAMPLE_DATA. The main area displays the results of a query on the PARTSUPP table, which has columns Row, PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, and PS_COMMENT. The results show 12 rows of data.

Row	PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
1	115504341	5504342	7576	960.18	lites, quickly final courts snooze ...
2	88004317	5504342	6881	416.02	s use against the instructions. q...
3	63004335	5504342	962	818.56	ctornis are alongside of the furio...
4	128004305	5504342	2356	361.93	ctions. furiously silent instructio...
5	150504311	5504342	2510	718.87	requests are, regular, even theo...
6	58004326	5504342	8258	281.41	s haggle special ideas. special, fl...
7	25504341	5504342	5676	105.53	silent instructions are furiously, ...
8	180504305	5504342	701	584.85	nst the ironic theodolites. blithel...
9	98004314	5504342	452	5.13	iously pending pinto beans. fluffi...
10	125504341	5504342	6789	208.87	e accounts are slyly above the ir...
11	95504341	5504342	4737	940.81	beans haggle blithely-- carefully...
12	195504341	5504342	7555	3.23	ending accounts. fluffy even re...

Snowflake - Partitions

Table Name : PARTSUPP

Clustering Key

PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
51578321	4078327	2356	23.56	Some text
196736070	2201164	3223	22.72	Some text
116736079	4078327	5543	23.56	Some text
179236090	4078327	1121	17.00	Some text
24236087	7993204	1433	13.88	Some text
49236091	7993204	1222	20.45	Some text
19236090	2201164	3211	24.2	Some text
139877911	2201164	2133	16.89	Some text

Micro-partition-1

PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
51578321	4078327	2356	23.56	Some text
196736079	4078327	5543	23.56	Some text
179236090	4078327	1121	17.00	Some text

Micro-partition-2

PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
196736070	2201164	3223	22.72	Some text
19236090	2201164	3211	24.2	Some text
139877911	2201164	2133	16.89	Some text

Micro-partition-3

PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
24236087	7993204	1433	13.88	Some text
49236091	7993204	1222	20.45	Some text

Snowflake - Partitions

Table Name : PARTSUPP

Micro-partition-1

PS_PARTKEY	51578321	116736079	179236090
PS_SUPPKEY	4078327	4078327	4078327
PS_AVAILQTY	2356	5543	1121
PS_SUPPLYCOST	23.56	21.56	17.00
PS_COMMENT	Some text	Some text	Some text

Micro-partition-2

PS_PARTKEY	51578321	116736079	179236090
PS_SUPPKEY	2201164	2201164	2201164
PS_AVAILQTY	3223	3211	1569
PS_SUPPLYCOST	23.56	21.56	17.00
PS_COMMENT	Some text	Some text	Some text

Micro-partition-3

PS_PARTKEY	51578321	116736079
PS_SUPPKEY	7993204	7993204
PS_AVAILQTY	2356	5543
PS_SUPPLYCOST	23.56	21.56
PS_COMMENT	Some text	Some text

SQL : select ps_partkey,ps_supplycost from partsupp where ps_suppkey='4078327'

Scanned micro-partition = micro-partition-1

Pruned micro-partitions = micro-partition-2 & 3

05:28

The screenshot shows the Microsoft Edge browser displaying the Snowflake Worksheet interface at the URL <https://qi45203.eu-west-1.snowflakecomputing.com/console#/internal/worksheet>. The interface includes a top navigation bar with Microsoft Edge icons, a title bar, and a status bar indicating 'Wed 2 Feb 10:18 PM'. Below the title bar is a toolbar with 'Databases', 'Shares', 'Data Marketplace', 'Warehouses', 'Worksheets' (which is selected), 'History', and 'Account'. To the right of the toolbar are 'Partner Connect', 'Help', 'Notifications', 'Snowsight', and the user account 'SIDDHARTH ACCOUNTADMIN'. The main workspace shows a 'Clustering Info' section with a dropdown menu and a 'Find database objects' search bar. A code editor window displays the following SQL query:

```

2
3 use warehouse etl_XL;
4
5 select system$clustering_information('LINEITEM');
6
7 show tables like 'SLINE%';
8
9
10 select * from LINEITEM where l_shipdate in ('1998-12-01','1998-09-20') limit 10000;
11
12 ALTER SESSION SET USE_CACHED_RESULT = FALSE;
13
14 select system$clustering_information('PARTSUPP','ps_suppkey'); I

```

Below the code editor is a results table titled 'Table: SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.PARTSUPP'. The table has columns: Row, PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, and PS_COMMENT. The data preview shows the following rows:

Row	PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY	PS_SUPPLYCOST	PS_COMMENT
1	193198397	698455	2811	345.88	special requests against the styl...
2	145698426	698455	6424	104.61	iously express deposits haggle f...
3	143198412	698455	2837	12.77	g requests across the enticing dl...
4	155698424	698455	7802	162.21	y unusual accounts. frets wake ...
5	130698454	698455	7208	634.43	r accounts detect slyly above th...
6	23198449	698455	64	126.40	after the final instructions. furio...
7	83198430	698455	309	267.04	nto beans solve even packages. ...

Snowflake - Benefits of Micro-partitions and clustering

- Query Pruning - Avoiding unnecessary scanning of micro-partitions
- Micro-partitions being small in size enables efficient DML operations
- Columns are stored independently and also compressed within micro-partitions thus enabling efficient scanning of individual columns .
- Micro-partitions are created automatically and maintained by snowflake
- Clustering is handled automatically unless automatic clustering is turned off

The screenshot shows the Snowflake Worksheet interface in Microsoft Edge. The URL is <https://qi45203.eu-west-1.snowflakecomputing.com/console#/internal/worksheet>. The user is Siddharth, ACCOUNTADMIN.

The left sidebar shows the database structure:

- ECOMMERCE_DB
 - Tables
 - CUSTOMERS
 - LINEITEM
 - NATION
 - ORDERS
 - PART
 - PARTSUPP
 - REGION
 - SUPPLIER
- INFORMATION_SCHEMA
- PUBLIC
- SNOWFLAKE
- SNOWFLAKE_SAMPLE_DATA
 - INFORMATION_SCHEMA
 - TPCDS_SF100TCL
 - TPCH_SF1
 - TPCH_SF10
 - TPCH_SF100
 - TPCH_SF1000
 - Tables
 - CUSTOMER

The main area shows a query being run:

```
use schema "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1000";
select system$clustering.information('CUSTOMER');
select * from customer limit 10;
use schema "ECOMMERCE_DB"."ECOMMERCE_LIV";
show tables like '%LINE%';
alter table LINEITEM suspend recluster;
```

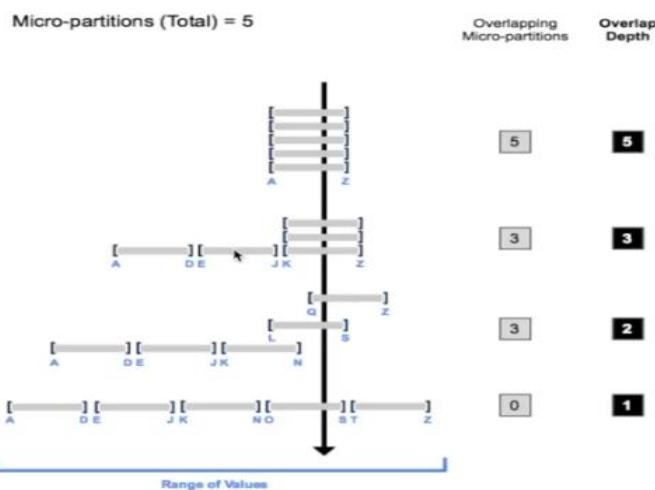
The results show one row returned in 43ms:

Query_ID	SQL	43ms	1 rows

Details of the table:

Name	Schema Name	Kind	Comment	Cluster_By	Rows	Bytes	Owner	Retention_Time	Automatic_Cluster	Change_Tracking	Search_Optimized	Se
C...	ECOMMERCE...	TABLE		LINEAR1_S...	5999989709	1571078379...	ACCOUNTA...	1	ON	OFF	OFF	

Clustering Depth and Overlap



DATA CLUSTERING

Typically, data stored in tables is sorted along natural dimensions, for example, by *date*. This process is called clustering, and data that is not sorted/clustered may hurt queries performance, particularly on huge tables, as Snowflake will have to analyze more micro-partitions to give a query result. Let's look at the following example, where the micro-partitions are ordered by date. In this case, if we had to query for the date 11/2, Snowflake wouldn't scan the last two micro-partitions, improving the performance of the query.

	Micro-partition 5 (rows 1, 4, 6, 8, 13, 15)			Micro-partition 6 (rows 3, 5, 7, 2, 9, 14)			Micro-partition 7 (rows 10, 12, 17, 11, 16, 19)			Micro-partition 8 (rows 18, 20-24)		
type	2	2	2	3	3	3	5	5	5	3	4	5
name				4	4	4	1	1	1	5	3	2
country												
date	11/2	11/2	11/2	11/2	11/2	11/2	11/3	11/3	11/3	11/4	11/4	11/4
				11/2	11/2	11/2	11/3	11/3	11/3	11/5	11/5	11/5

In Snowflake, clustering metadata is collected and recorded for each micro-partition. **Snowflake then leverages this clustering information to avoid unnecessary scanning of micro-partitions during querying**, significantly accelerating the performance of queries that reference these columns.

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.

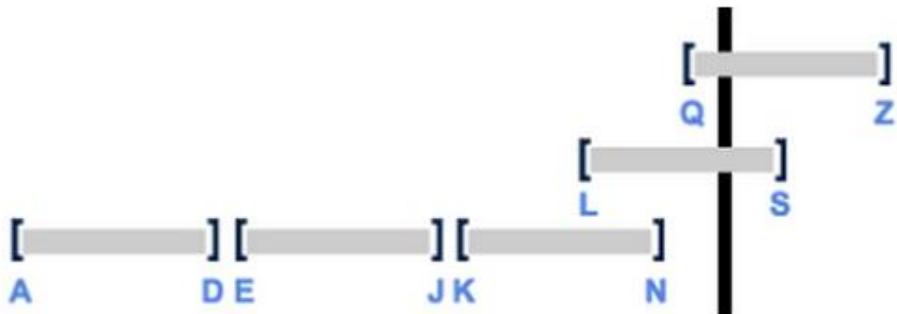
- The depth of the overlapping micro-partitions.

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.** A table without partitions would have a cluster depth of 0, although this is not possible as a table will contain at least one micro-partition (but this can appear as an exam question). You can use the following commands to get the Cluster Depth:

- **SYSTEM\$CLUSTERING_DEPTH**
- **SYSTEM\$CLUSTERING_INFORMATION**

Let's take a look at an example to understand the cluster keys. Let's imagine that we have the following table that is partitioned as follows:

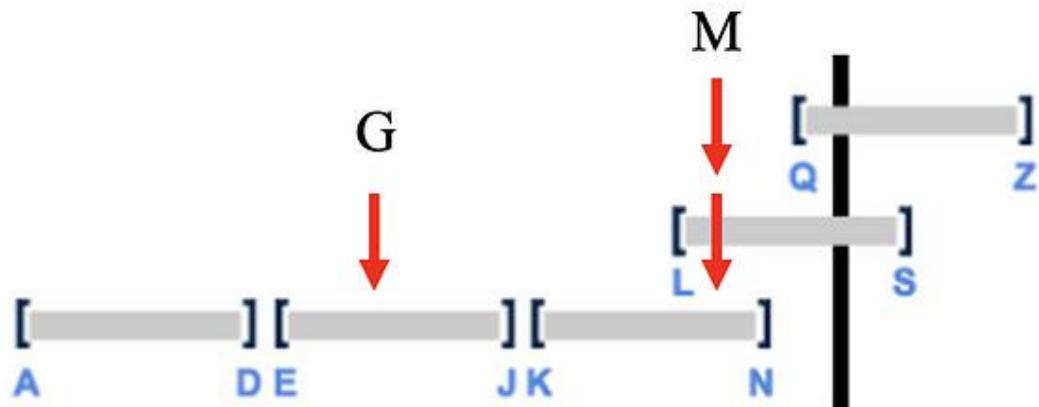


1. The first partition contains data from A to D.
2. The second partition contains data from E to J.
3. The third partition contains data from K to N.

4. The fourth partition contains data from L to S.

5. The fifth partition contains data from Q to Z.

It would be straightforward to search for a value that starts with the letter G. Snowflake would only have to go to the second partition and return the values, as there are no overlapping micro-partitions. But what would happen if we searched for a value that starts with the letter M? In this case, Snowflake is accessing the third and fourth micro-partitions.



Clustering Depth looking for the G and the M letters.

Thus, we could say from this example:

- 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.

As we can see, **a higher clustering depth would indicate that Snowflake checks a lot of micro-partitions, making the query slower.**

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. This is useful for **huge tables** where the ordering was not ideal or extensive insert operations have caused the table's natural clustering to degrade. Cluster Keys are **placed on columns usually used in the WHERE / JOINS / ORDER BY...** commands. As we said before, they can also be a subset of expressions on a table. Imagine you filter the table by month; you can use TO_MONTH to create a cluster key.

Some general indicators that can help determine whether to define a clustering key for a table include:

- Queries on the table are running slower than expected or have noticeably degraded over time.
- The clustering depth for the table is large.

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. Let's imagine we have [the table of the previous example](#) and we want to add another row with the "11/2" date, but the micro-partitions 5 & 6 are full of data. It will go to the micro-partition 9 for example. Or what if we modify a value from the micro-partition 5 from "11/2" to "11/4"? It will make the table less clustered. If we wanted to look for the "11/2" value, we would have to access three micro-partitions instead of just two like before:

	Micro-partition 5 (rows 1, 4, 6, 8, 13, 15)			Micro-partition 6 (rows 3, 5, 7, 2, 9, 14)			Micro-partition 7 (rows 10, 12, 17, 11, 16, 19)			Micro-partition 8 (rows 18, 20-24)			Micro-partition 9 (rows 18, 20-24)		
type	2	2	2	3	3	3	5	5	5	3	4	5	3		
name				4	4	4	1	1	1	5	3	2			
country															
date	11/2	11/2	11/2	11/2	11/2	11/2	11/3	11/3	11/3	11/4	11/4	11/4	11/2		
	11/2	11/2	11/2	11/2	11/2	11/2	11/3	11/3	11/3	11/5	11/5	11/5			

How micro-partitions can be less clustered.

To solve that, **Snowflake provides periodic & automatic re-clustering to maintain optimal clustering**.

This re-clustering operation consumes both credits and storage, and for this reason, the more frequently a table changes, the more expensive it will be to keep it clustered. Therefore, **clustering is generally more cost-effective for tables that are queried often and do not change frequently.**

In the following example, we can see how re-clustering works. Date and Type are defined as the clustering key. **When the table is re-clustered, new micro-partitions (5–8) are created, and the old ones will be marked as deleted.** We had to access three partitions if we needed to do a query with a filter with *Type = 2 and Date = 11/2*. After re-clustering, we would only need to access one micro-partition.

Original Micro-partitions					New Micro-partitions (After Reclustering)				
	Micro-partition 1 (rows 1-4)	Micro-partition 2 (rows 5-8)	Micro-partition 3 (rows 9-16)	Micro-partition 4 (rows 17-24)		Micro-partition 5 (rows 1, 4, 6, 8, 15, 16)	Micro-partition 6 (rows 2, 5, 7, 12, 13, 14)	Micro-partition 7 (rows 10, 11, 12, 17, 18, 19)	Micro-partition 8 (rows 18, 20-24)
2 type	2 4 3 2 3 2	3 2 4 5 1 5	2 4 2 1 6 3	1 4 5 5 3 2	type	2 2 2 2 2 2	3 3 3 4 4 4	5 5 5 1 1 1	3 4 5 5 3 2
name					name				
country					country				
1 date	11/2 11/2 11/2 11/3 11/2 11/3	11/2 11/2 11/2 11/3 11/3 11/4	11/2 11/2 11/2 11/3 11/3 11/4	11/3 11/4 11/4 11/5 11/5 11/5	date	11/2 11/2 11/2 11/2 11/2 11/2	11/2 11/2 11/2 11/2 11/2 11/2	11/3 11/3 11/3 11/3 11/3 11/3	11/4 11/4 11/4 11/5 11/5 11/5

Snowflake - Clustering depth and Overlap

- A table with no micro-partitions(empty table) has a depth of Zero (0)
- Average depth :
 - Average overlap depth of each micro-partition in the table.
 - Smaller the average depth , the better clustered the table is . The value is always 1 or greater .
 - Eg :
 - **average_depth=65** --> Not well clustered
 - **average_depth=1.2** --> Well clustered
- Average overlaps :
 - Average number of overlapping micro-partitions for each micro-partition in the table.
 - A high number indicates the table is not well clustered .
 - Eg :
 - **average_overlap = 120** --> Not well clustered
 - **average_overlap = 4.5** --> Well clustered
- **Partition_depth_histogram** : A histogram depicting the distribution of overlap depth for each micro-partition in the table.

The screenshot shows the Snowflake Worksheet interface. On the left, the sidebar displays the database structure under 'ECOMMERCE_DB'. A query is being run in the main area:

```
use schema 'SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1000';
select system$clustering_information('LINEITEM');
```

The results pane shows the output of the query, which includes clustering details and a histogram of partition depths. A 'Details' modal is open over the results, showing the histogram data:

Bin Range	Count
"00000" : 0,	1367
"00001" : 0,	0
"00002" : 2308,	0
"00003" : 2206,	0
"00004" : 58,	0
"00005" : 0,	0
"00006" : 0,	0

At the bottom right of the modal, there is a 'Done' button.

Snowflake stores data in columnar format in varying length micro-partitions and automatically applies partition elimination upon every column. Using this method can lead to incredible query performance. This article explains how adding a Snowflake cluster key (which is neither an index nor the same as a Redshift Cluster Key) can further improve query performance by physically clustering data together.

Cluster keys are one of the few ways to maximize query performance on Snowflake and this article explains both how Snowflake data is stored, and how to monitor query performance.

How does Snowflake physically store data?

The diagram below illustrates how nearly every other relational database (including Oracle) physically stores data in rows, a solution designed initially for Online Transaction Processing (OLTP) systems back in the 1970s.

Row Optimized

```
select *  
from tab  
where id = 3;
```



Row Format				
ID	Sales Channel	City	Value	Month
1	Web	Moscow	400	Jan
2	Web	Paris	250	Jan
3	Direct	Helsinki	300	Jan
4	Direct	Tokyo	150	Jan
5	Web	Moscow	900	Jan
6	Web	Tokyo	100	Jan

Row Optimised Data Storage

The above method optimizes the speed of single-row queries typically found in OLTP transaction-based systems, and each fetch returns an entire row at a time.

While the above solution works well for single-row queries, analytic platforms typically fetch thousands or even millions of rows at a time and often return just a few selected columns at a time. This makes them a very poor solution - considered a [legacy technology](#) by some.

The diagram below illustrates how Snowflake stores data in *columnar* format.

Column Optimized

```
select channel  
, sum(value)  
from tab  
group by channel;
```



Columnar Format						
ID:	1	2	3	4	5	6
Channel:	Web	Web	Direct	Direct	Web	Web
City:	Moscow	Paris	Helsinki	Tokyo	Moscow	Tokyo
Value:	400	250	300	150	900	100
Month:	Jan	Jan	Jan	Jan	Jan	Jan

Snowflake - Column Optimised Data Storage

Using this method, a query that returns just two items from a table with 100 columns, has an immediate 98% performance advantage over the row-based solution. This technique was proven to produce incredible query performance by Professor Mike Stonebraker when he published a paper on the [C-Store Column Oriented database](#).

In addition to quite remarkable query performance gains, storing data in columnar format means there is extensive scope to compress the data, and Snowflake automatically uses a different compression algorithm against each column, often achieving a compression ratio of over ten times compared to data stored in Amazon S3.

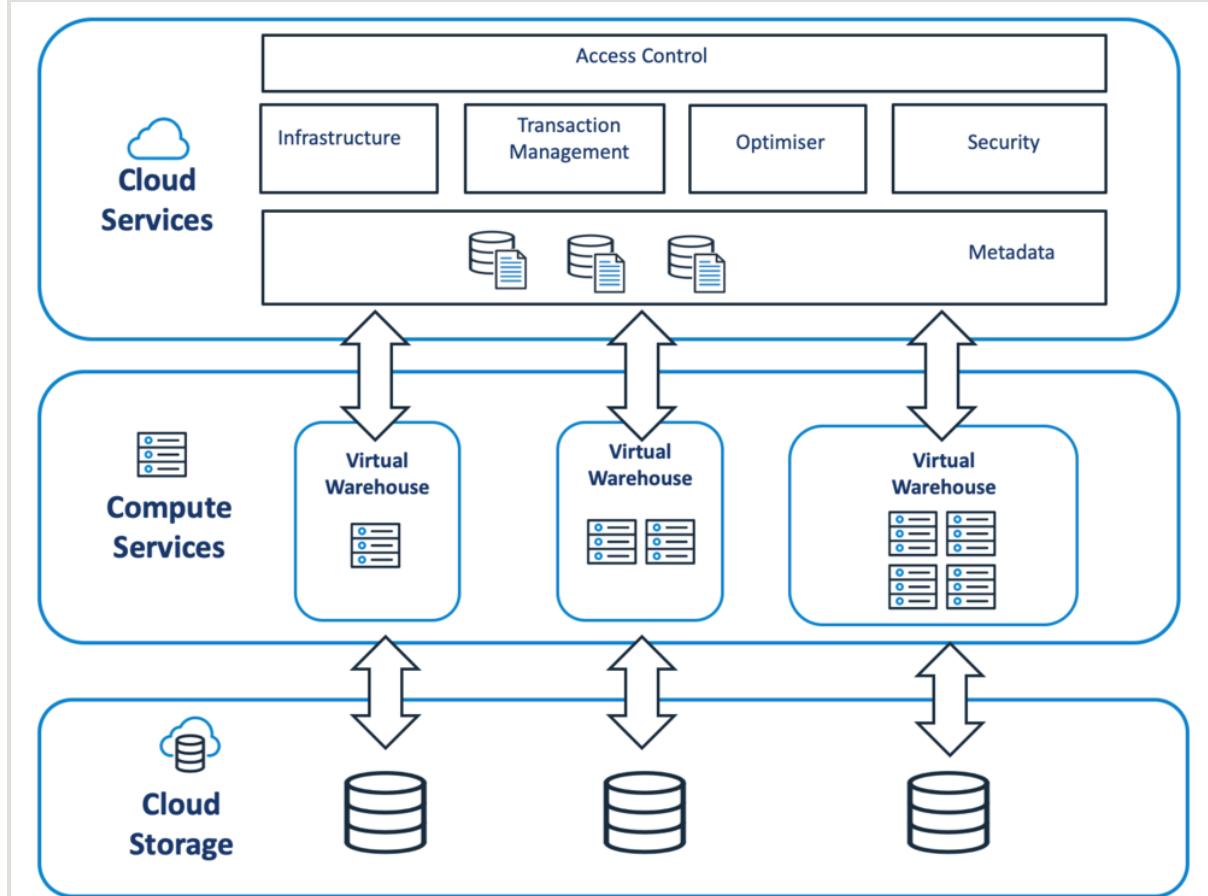
Snowflake Micro-partitions

In addition to storing data in columnar format, Snowflake also stores multiple rows together in [*micro-partitions*](#), which are variable-length data blocks of around 16 Mb size. Like many Snowflake features, you cannot control the micro-partition size, and the diagram below illustrates a typical layout, with each micro-partition holding a large number of rows in columnar format.

Micro-Partition 1				Micro-Partition 2			
ID: 1 2 3 4 5 6				7 8 9 10 11 12 13 14			
Sales Channel: Web Direct Web Web Web Direct Web Web				Web Shop Shop Direct Web Shop Web Web Direct Web			
City: Moscow Paris Helsinki Tokyo Moscow Tokyo				Berlin Rio Tokyo Rio Moscow Berlin Rio Tokyo			
Value: \$400 \$250 \$300 \$150 \$900 \$100				\$150 \$600 \$250 \$450 \$350 \$200 \$150 \$500			
Month: Jan Jan Jan Jan Jan Jan				Feb Feb Feb Feb Feb Feb Feb Feb			

Snowflake - Micro-partitions

At this point, it's worth considering the overall [Snowflake architecture](#), illustrated in the diagram below, which shows, unlike a traditional database generally hosted on a single server, Snowflake uses three distinct layers. Cloud services, Query Services, and Database Storage.



Snowflake System Architecture

The above diagram illustrates how a Virtual Warehouse executes end-user queries, which in turn are coordinated by the Cloud Services layer. This scale-out database architecture acts as *the brains* of the operation and automatically captures metadata about data as it's loaded, including query statistics used to tune query performance automatically. This metadata includes for every micro-partition:

- **Maximum & Minimum Values:** Which holds the numeric, date, or alphabetic maximum and minimum value of every column in the table. This is recorded for every micro-partition.
- **Other Statistics:** Which includes the count of total values, distinct values, and NULL entries, again used to maximize query performance.

Using the above techniques, Snowflake can execute several queries within milliseconds without actually needing any access to the underlying data, or even needing a virtual warehouse.

For example, the query below returns instantly without needing a virtual warehouse at all:

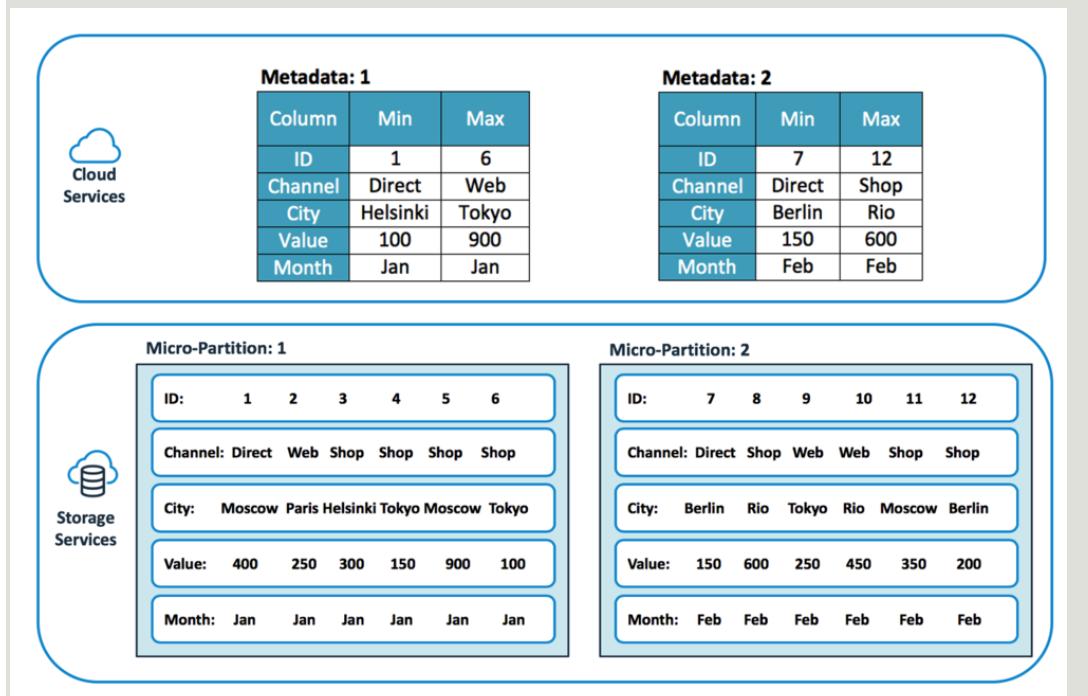
```
select count(*)
      , max(l_discount)
    from
snowflake.sample_data.tpch_sf1000.lineitem;

COUNT(*)          MAX(L_DISCOUNT)
-----           -----
59,999,989,703        0.10

Elapsed: 479 ms
```

Automatic Partition Elimination

The diagram below illustrates how Snowflake uses the maximum and minimum values to provide automatic partition elimination against every column on the table - even without using Snowflake clustering. This method has considerable advantages over traditional physical partitioning, including modern formats like Parquet, as it means partition elimination is automatically applied to every column on the entire table.



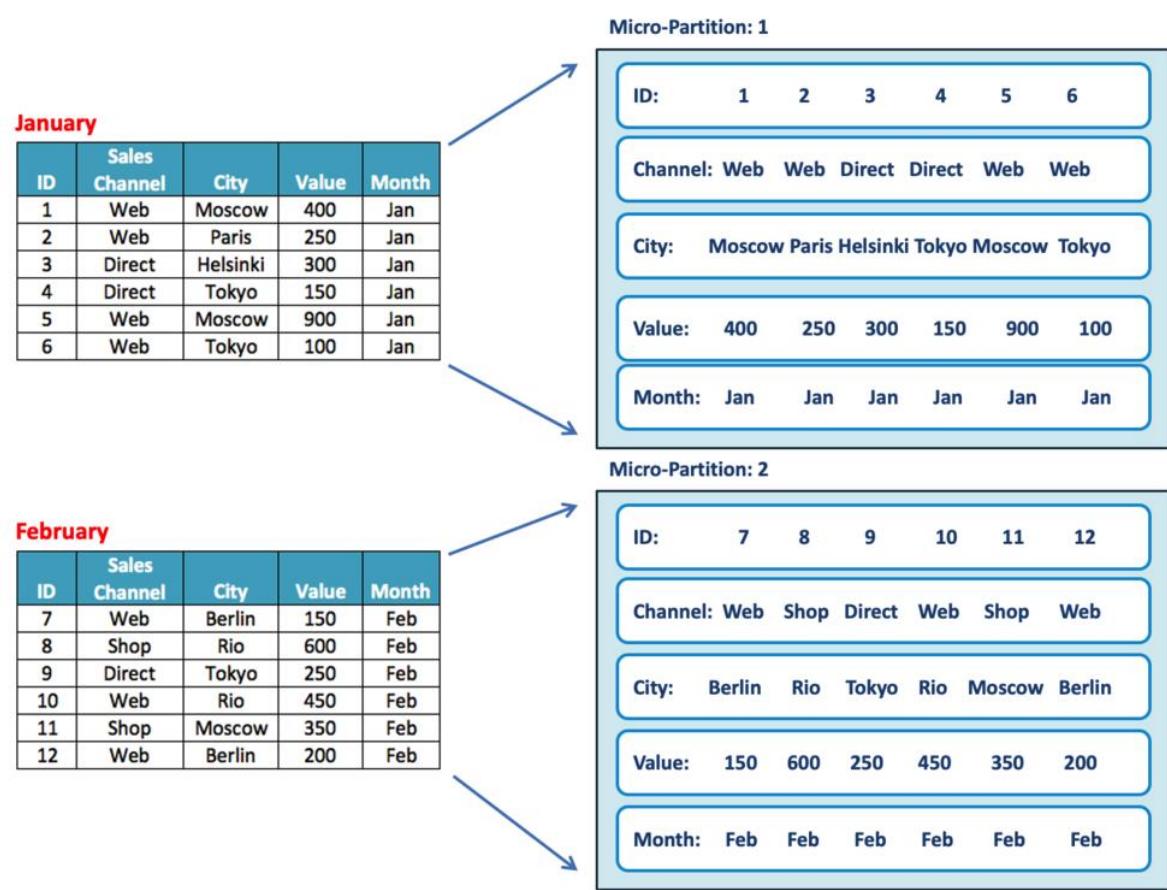
For example, despite the fact the above table may hold millions of rows in thousands of micro-partitions, the following query returns only one partition-based purely on the minimum and maximum values:

```
select *
from sales_table
where month = 'JAN'
and value = 601
and channel = 'Web'
and city = 'Tokyo';
```

Automatic partition elimination can lead to substantial performance gains for queries against massive tables, and unlike traditional solutions, there are no indexes to slow data ingestion performance.

Natural Data Clustering

The combination of holding data in micro-partitions and automatically capturing statistics during data ingestion has another interesting side effect – that time-based data is often loaded in *natural clustering sequence*. The diagram below illustrates this scenario whereby data is regularly ingested in Snowflake.



If *fact table* entries are loaded daily, it's reasonable to find the data is naturally clustered, with the data for each month in a separate micro-partition and few overlapping entries. Using this technique, queries which include the DATE as a predicate automatically gain performance benefits by eliminating unnecessary reads, without any further action required. You can monitor the existing clustering level using Snowflake [system functions](#).

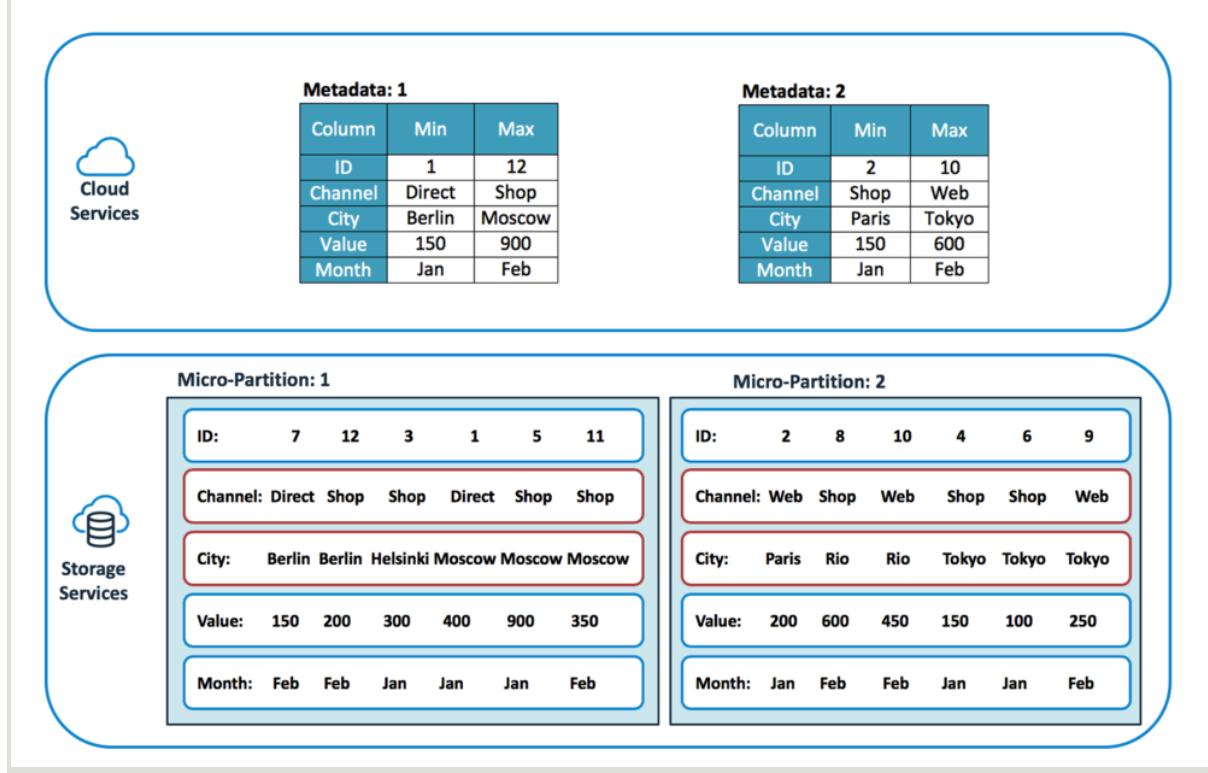
However, be aware, this behaviour is not guaranteed, and (for example) loading the data in a random sequence or using multiple parallel load processes may reduce the effect of natural clustering.

Maximizing Query Performance using Cluster Keys

One way to guarantee physical data clustering and maximize query performance, especially on very large (terabyte-sized) tables, is to define a cluster key. The SQL statement below illustrates the command to cluster the data by CITY and then CHANNEL:

```
alter table sales
cluster by (city, channel);
```

The above solution would be useful if most queries are limited by these two columns and indicate to Snowflake that the data should be physically sorted by these two. The diagram below illustrates the effect of this clustering, with the data sorted by the alphabetic sequence of CITY.

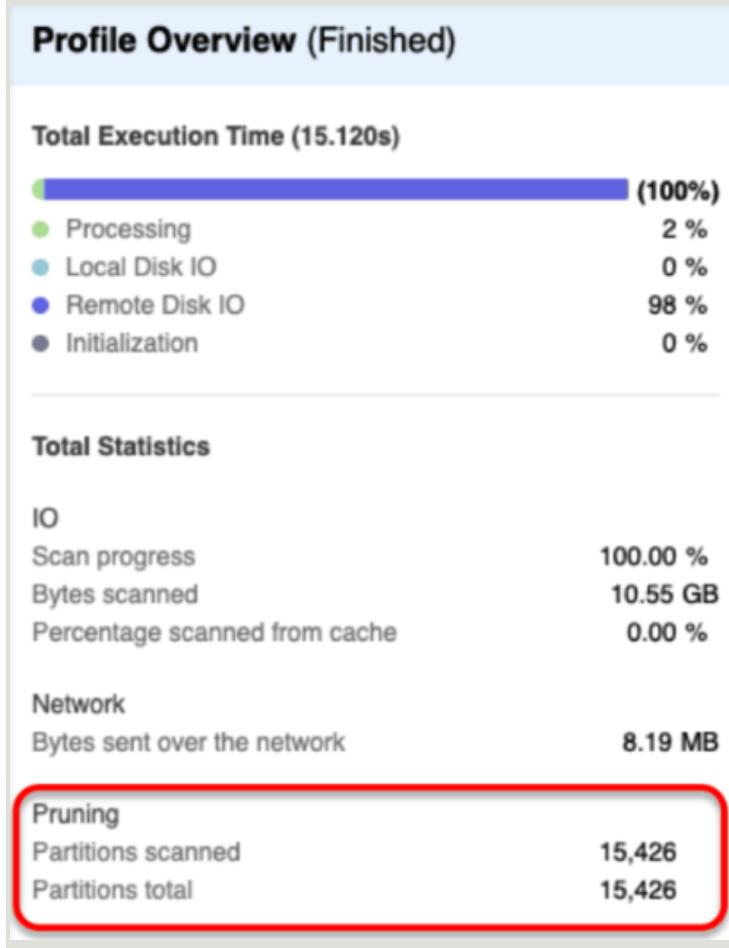


Using this method, a query to filter by CITY would reduce the number of micro-partitions scanned, and therefore maximize query performance.

Monitoring Partition Elimination - Best Practice

Snowflake includes a massive amount of metadata, including the history of every query executed with full statistics to help diagnose problems. One of the most useful information sources is the *Query Profile*, which is available on the web user interface.

The screenshot below shows an example query profile of a query run against 250m rows, which took over 15 seconds to complete.



The following points are worth noting:

- **Execution Time:** Is as little as 15 seconds, which, although fast, could be improved.
- **Percentage Processing:** At just 2% is relatively low.
- **Remote Disk IO:** At 98% is very high, indicating the query spent most of the time fetching data from disk.
- **Pruning:** The query returned the entire table consisting of over 15,000 micro-partitions.

The screenshot below shows the results of the same query against a table partitioned by DATE, which appears as a predicate in the WHERE clause.

Profile Overview (Finished)

Total Execution Time (1.327s)



Total Statistics

IO

Scan progress	0.18 %
Bytes scanned	8.99 MB
Percentage scanned from cache	0.00 %

Network

Bytes sent over the network	7.38 MB
-----------------------------	---------

Pruning

Partitions scanned	28
Partitions total	15,726

Unlike the previous query, which took 15 seconds, this query ran eleven times faster against the same data. The points to note are:

- **Execution Time:** Has been reduced from over 15 seconds to 1.3 seconds.
- **Remote Disk IO:** Has been reduced from 98% to just 8% of the total execution time.
- **Bytes Scanned:** Was reduced from 10.5Gb to under nine megabytes, a significant improvement.
- **Pruning:** The query fetched just 28 partitions, which eliminated 99% of the data.

Avoid using functions against WHERE clause predicates

Be cautious using functions in the WHERE clause. For example, the following queries produce exactly the same result, but the second query has much greater partition elimination.

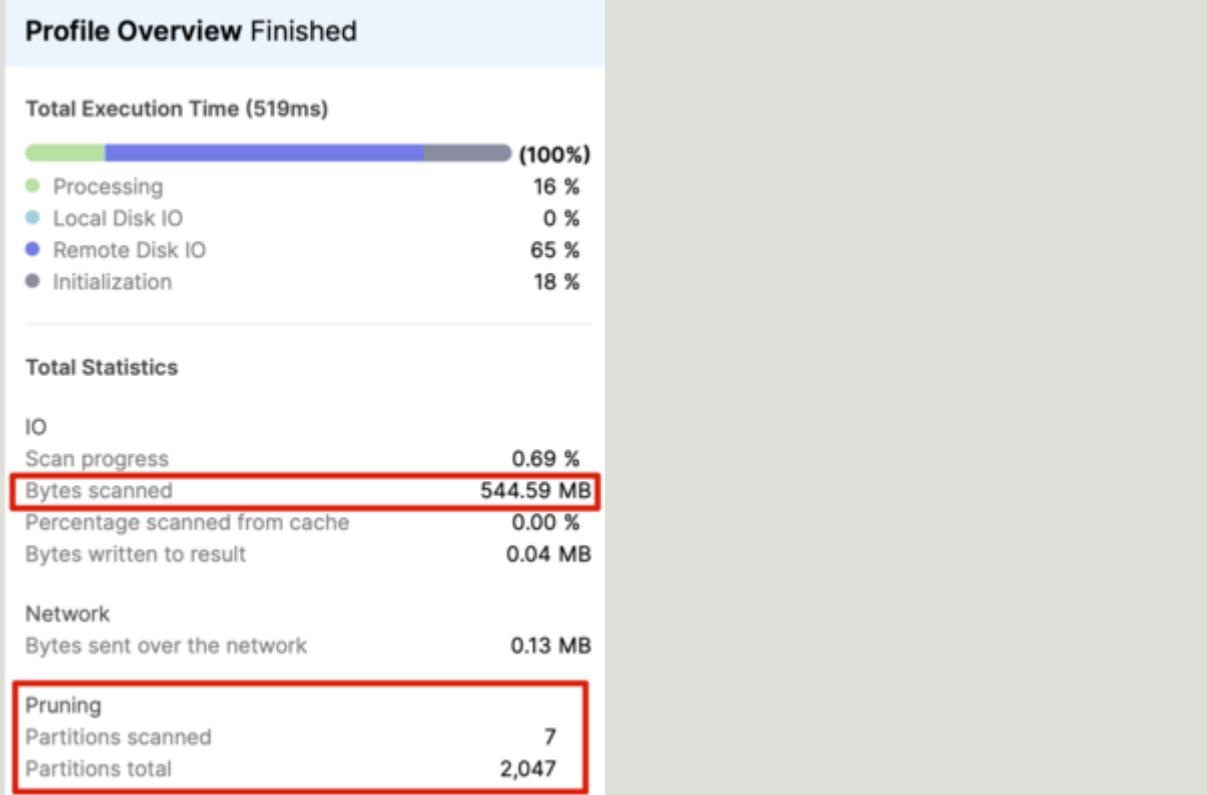
```
select *
from monitor_query_history
where execution_status = 'SUCCESS'
AND warehouse_size IS NOT NULL
AND to_char(start_time,'YYYYMMDD') >= '20210620'
AND to_char(start_time,'YYYYMMDD') <= '20210720'
and bytes_scanned > 0
limit 100;
```

The screen shot below shows the partition elimination on this query.



Now compare the above to the following SQL which returns exactly the same result, but much faster.

```
select *
from monitor_query_history
where execution_status = 'SUCCESS'
AND warehouse_size IS NOT NULL
AND start_time >= '2021-06-20'
AND start_time <= '2021-07-20'
and bytes_scanned > 0
limit 100;
```



As you can see, the second query which avoided the function against the START_TIME column scanned 20 times fewer micro-partitions. This is because Snowflake cannot use the column in the partition elimination. However, you can declare a clustering key using a function and provided the WHERE clause uses the same function this works fine.

Snowflake Clustering Best Practice

It is however, not sensible to simply place clustering on every table in the database. You should use the following best practice guidelines:

- **If it aint broke:** Clustering happens in background and is charged for the sorting. Balance the cost of clustering against the benefits in partition elimination. If you don't have a query performance issue, or the clustering is not helping, then don't use it.
- **Deploy on Large Tables:** As Snowflake stores data in 16Mb micro-partitions (chunks), there's no point clustering small tables. Snowflake recommend clustering tables over a terabyte in size. In reality, consider anything above 500Mb, but base your decision upon the need to improve partition elimination.
- **Choose your key wisely:** Clustering physically sorts the data, which means you only get one key (with possible sub-keys). Choose a cluster key that appears frequently in query WHERE clauses, as this will have the greatest impact upon partition elimination and therefore query performance improvements.
- **Composite Keys:** Are fine, for example you could cluster by DATE and REGION, but experience shows adding additional keys increase the cost of clustering, but have a reduced improvement in partition elimination.

- **Cluster relatively static tables:** Clustering will automatically sort data (if needed) to improve partition elimination. However frequent random updates and deletes will potentially degrade the effect and need re-sorting. Keep in mind, it's a balance of cost and benefit. Don't cluster frequently changing tables or the cost of clustering will out-weigh the benefits in query speed.
- **Be cautious using functions against predicates:** By all means, if you define a cluster key using a function (for example to convert a timestamp to a date) then you should use the same function in your where clause. However, avoid using functions against columns if possible as these tend to be ignored for the purposes of partition elimination.

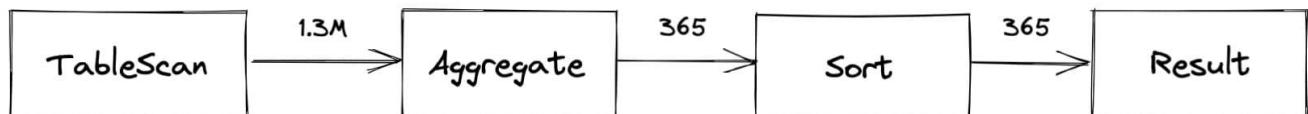
What is a Snowflake query plan?

Before we can talk about the Query Profile, it's important to understand what a "query plan" is. For every SQL query in Snowflake, there is a corresponding query plan produced by the query optimizer. This plan contains the set of instructions or "steps" required to process any SQL statement. It's like a data recipe. Because Snowflake automatically figures out the optimal way to execute a query, a query plan can look different than the logical order of the associated SQL statement.

In Snowflake, the query plan is a **DAG** that consists of **operators** connected by **links**. Operators process a set of rows. Example operations include scanning a table, filtering rows, joining data, aggregating, etc. Links pass data between operators. To ground this in an example, consider the following query:

```
select  
  
    date_trunc('day', event_timestamp) as date,  
  
    count(*) as num_events  
  
from events  
  
group by 1  
  
order by 1
```

The corresponding query plan would look something like this:



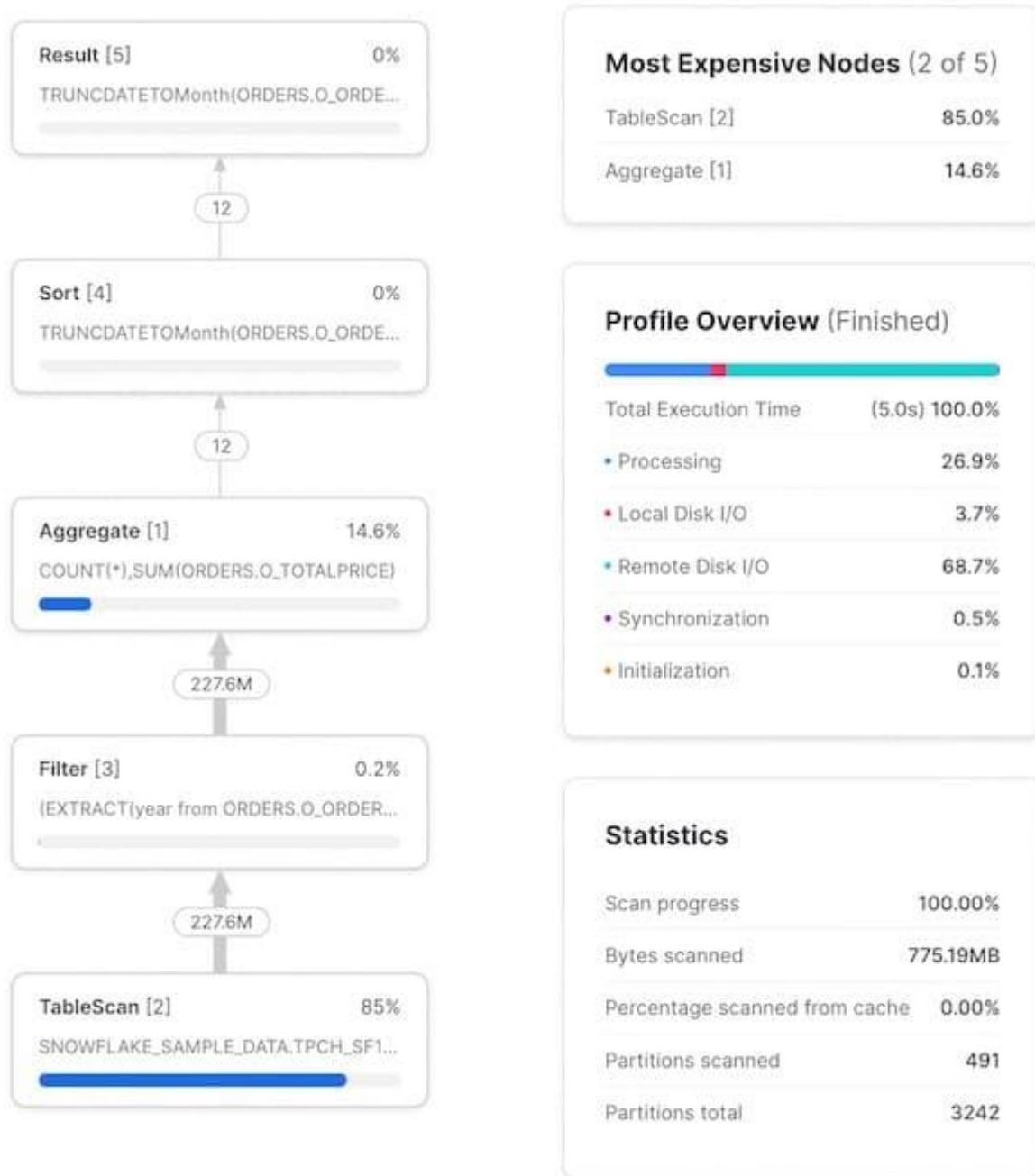
In this plan there are 4 "operators" and 3 "links":

1. **TableScan**: reads the records from the **events** table in remote storage. It passes 1.3 million records¹ through a link to the next operator.
2. **Aggregate**: performs our group by date and count operations and passes 365 records through a link to the next operator.
3. **Sort**: orders the data by date and passes the same 365 records to the final operator.
4. **Result**: returns the query results.

The Query Profile will often refer to operators as "operator nodes" or "nodes" for short. It's also common to refer to them as "stages".

What is the Snowflake Query Profile?

Query Profile is a feature in the Snowflake UI that gives you detailed insights into the execution of a query. It contains a visual representation of the query plan, with all nodes and links represented. Execution details and statistics are provided for each node as well as the overall query.



When should I use it?

The query profile should be used whenever more diagnostic information about a query is needed. A common example is to understand why a query is performing a certain way. The Query Profile can help reveal stages of the query that take significantly longer to process than others. Similarly, you can use the Query Profile to figure out why a query is still running and where it is getting stuck.

Another useful application of the Query Profile is to figure out why a query didn't return the desired result. By carefully studying the links between nodes, you can potentially identify parts of your query that are resulting in dropped rows or duplicates, which may explain your unexpected results.

How do you view a Snowflake Query Profile?

After running a query in the Snowsight query editor, the results pane will include a link to the Query Profile:

No Database selected ▾

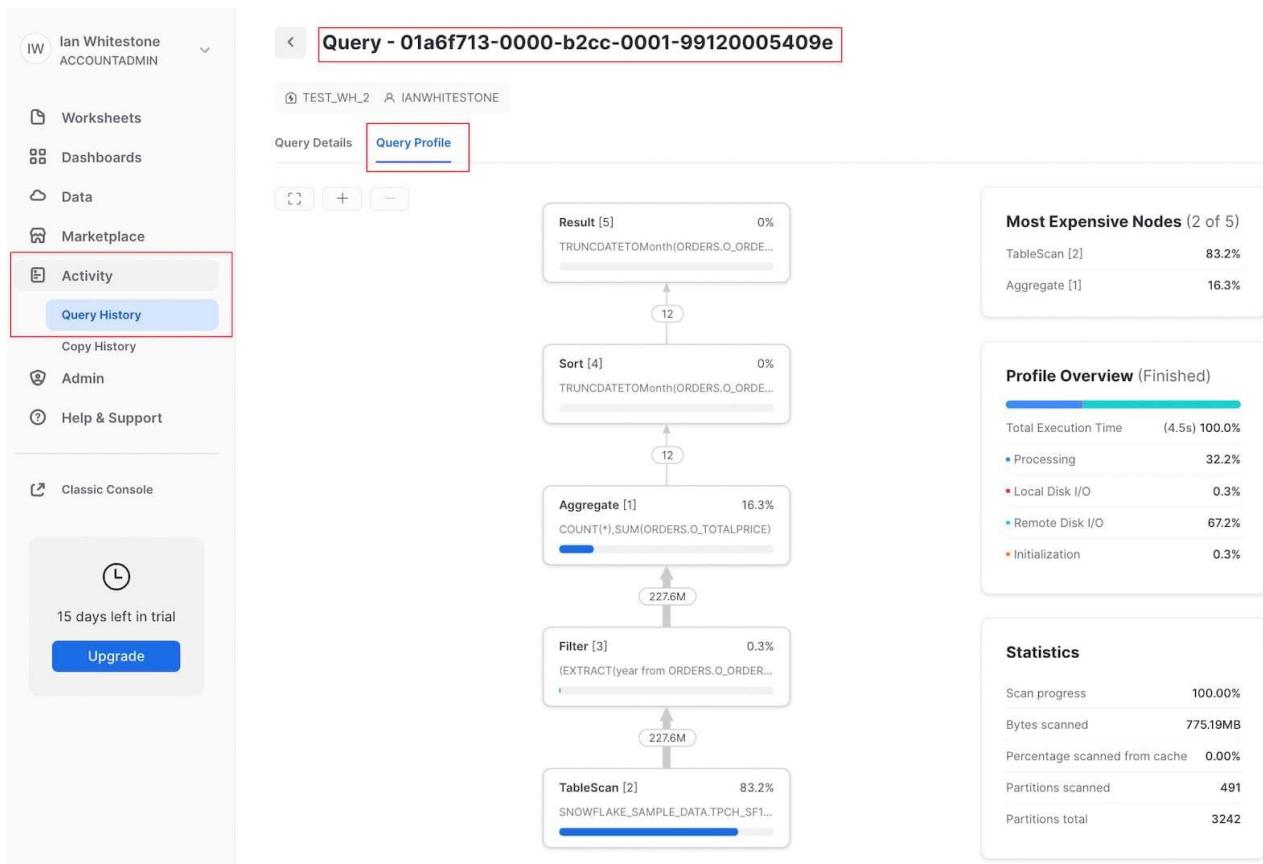
```
1 select
2     date_trunc('month', o_orderdate) as order_month,
3     count(*) as num_orders,
4     sum(o_totalprice) as total_order_value
5 from snowflake_sample_data.tpch_sf1000.orders
6 where
7     year(o_orderdate)=1997
8 group by order_month
9 order by order_month
```

The screenshot shows the Snowsight interface. At the top, there are tabs: Objects, Editor, Results (which is selected), and Chart. Below the tabs is a search bar and a download icon. The main area displays a table with three rows of data. The table has columns: ORDER_MONTH, NUM_ORDERS, and TOTAL_ORDER_VALUE. The data is as follows:

	ORDER_MONTH	NUM_ORDERS	TOTAL_ORDER_VALUE
1	1997-01-01	19,329,974	2,920,965,699,955.59
2	1997-02-01	17,456,308	2,638,055,850,515.02
3	1997-03-01	19,326,642	2,920,705,939,980.2

To the right of the table is a sidebar with the following sections: Query Details, Query duration (with a progress bar), Rows, Copy Query ID, and View Query Profile. A red arrow points to the 'View Query Profile' button, which is highlighted with a red box.

Alternatively, you can navigate to the "Query History" page under the "Activity" tab. For any query run in the **last 14 days**, you can click on it and see the Query Profile.



If you already have the **query_id** handy, you can take advantage of Snowflake's structure URLs by filling out this URL template:

How do you read a Snowflake Query Profile?

Basic Query

We'll start with a simple query anyone can run against the Snowflake sample dataset:

```
select
```

```
date_trunc('month', o_orderdate) as order_month,
```

```
count(*) as num_orders,
```

```
sum(o_totalprice) as total_order_value
```

```
from snowflake_sample_data.tpch_sf1000.orders
```

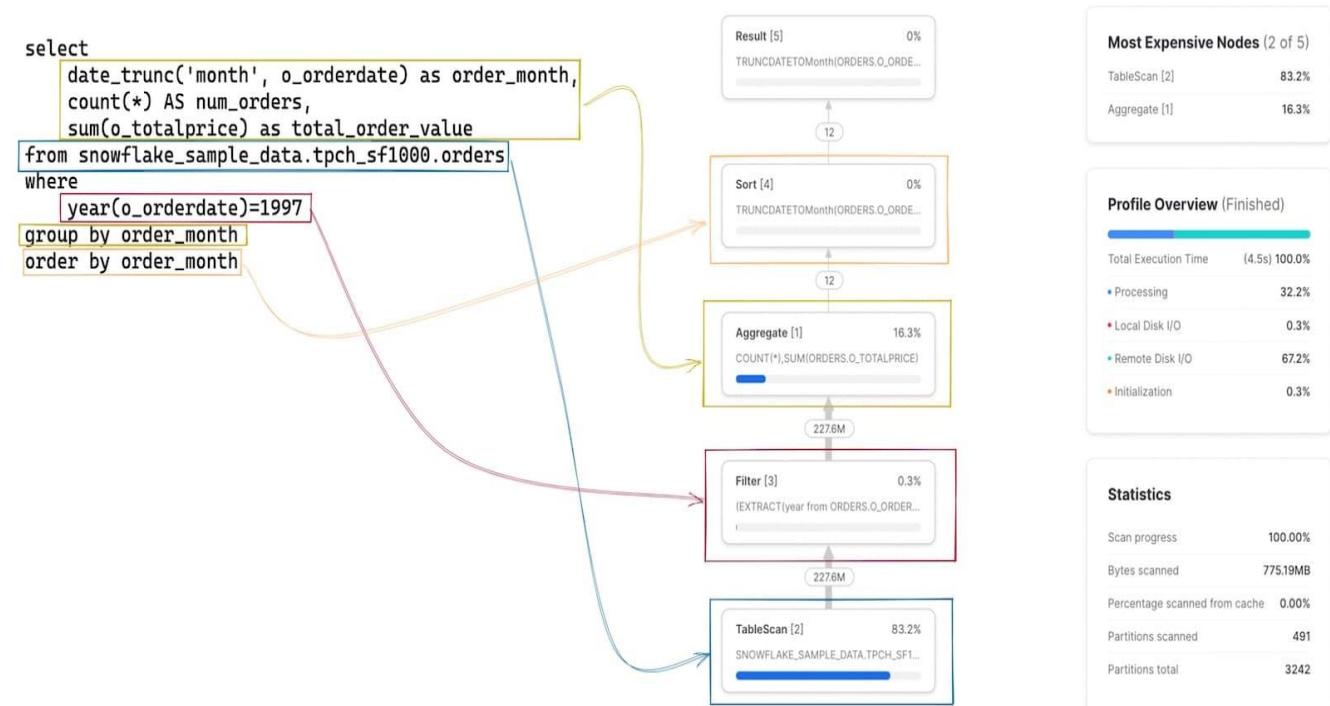
```
where
```

```
year(o_orderdate)=1997
```

```
group by order_month
```

```
order by order_month
```

As a first step, it's important to build up a mental model of how each stage/operator in the Query Profile matches up to the query you wrote. This is tricky to do at first, but very quick once you get the hang of it. Clicking on each node will reveal more details about the operator, like the table it is scanning or the aggregations it is performing, which can help identify the relevant SQL. The SQL relevant to each operator is highlighted in the image below:

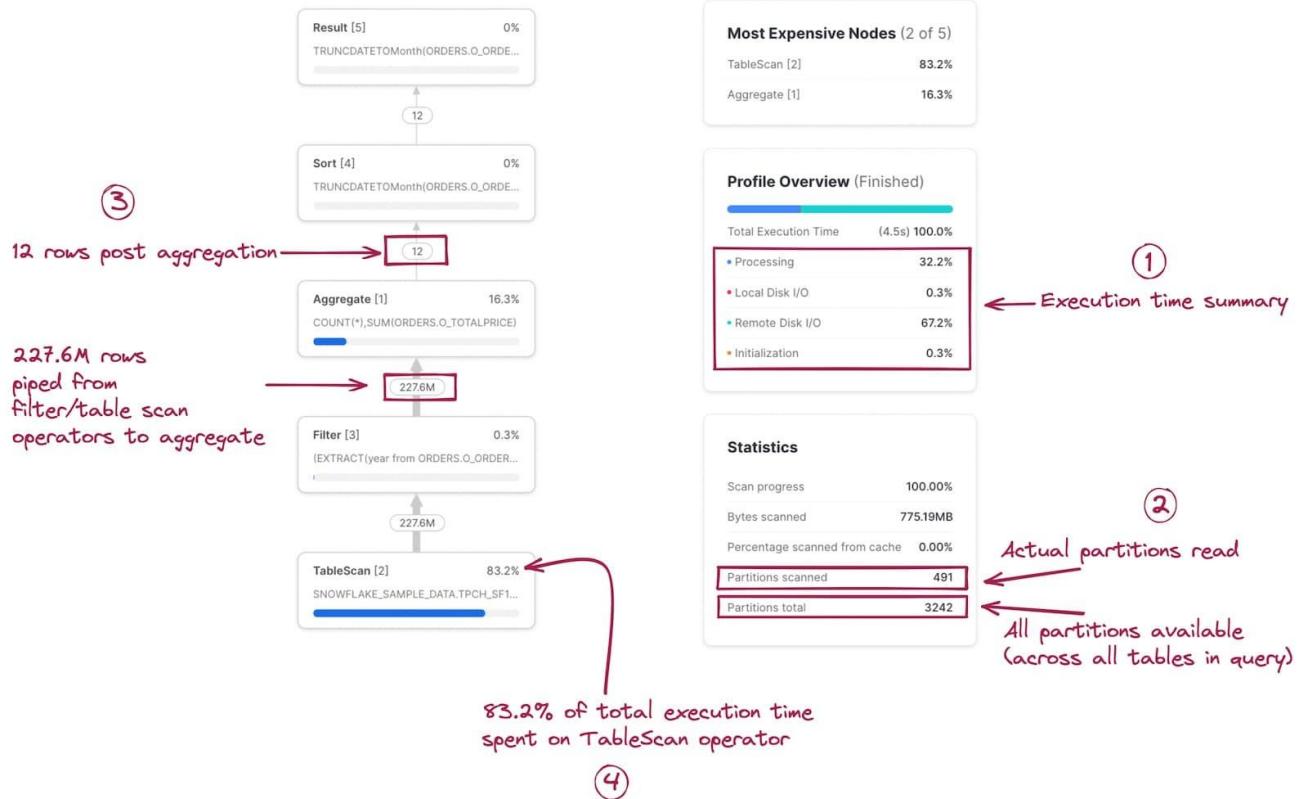


The Query Profile also contains some useful statistics. To highlight a few:

1. An execution time summary. This shows what % of the total query execution time was spent on different buckets. The 4 options listed here include:
 1. Processing: time spent on query processing operations like joins, aggregations, filters, sorts, etc.
 2. Local Disk I/O: time spent reading/writing data from/to local SSD storage. This would include things like spilling to disk, or reading cached data from local SSD.
 3. Remote Disk I/O: time spent reading/writing data from/it remote storage (i.e. S3 or Azure Blob storage). This would include things like spilling to remote disk, or reading your datasets.
 4. Initialization: this is an overhead cost to start your query on the warehouse. In our experience, it is always extremely small and relatively constant.
2. Query statistics. Information like the number of partitions scanned out of all possible partitions can be found here. Note that this is across all tables in the query. Fewer partitions scanned means

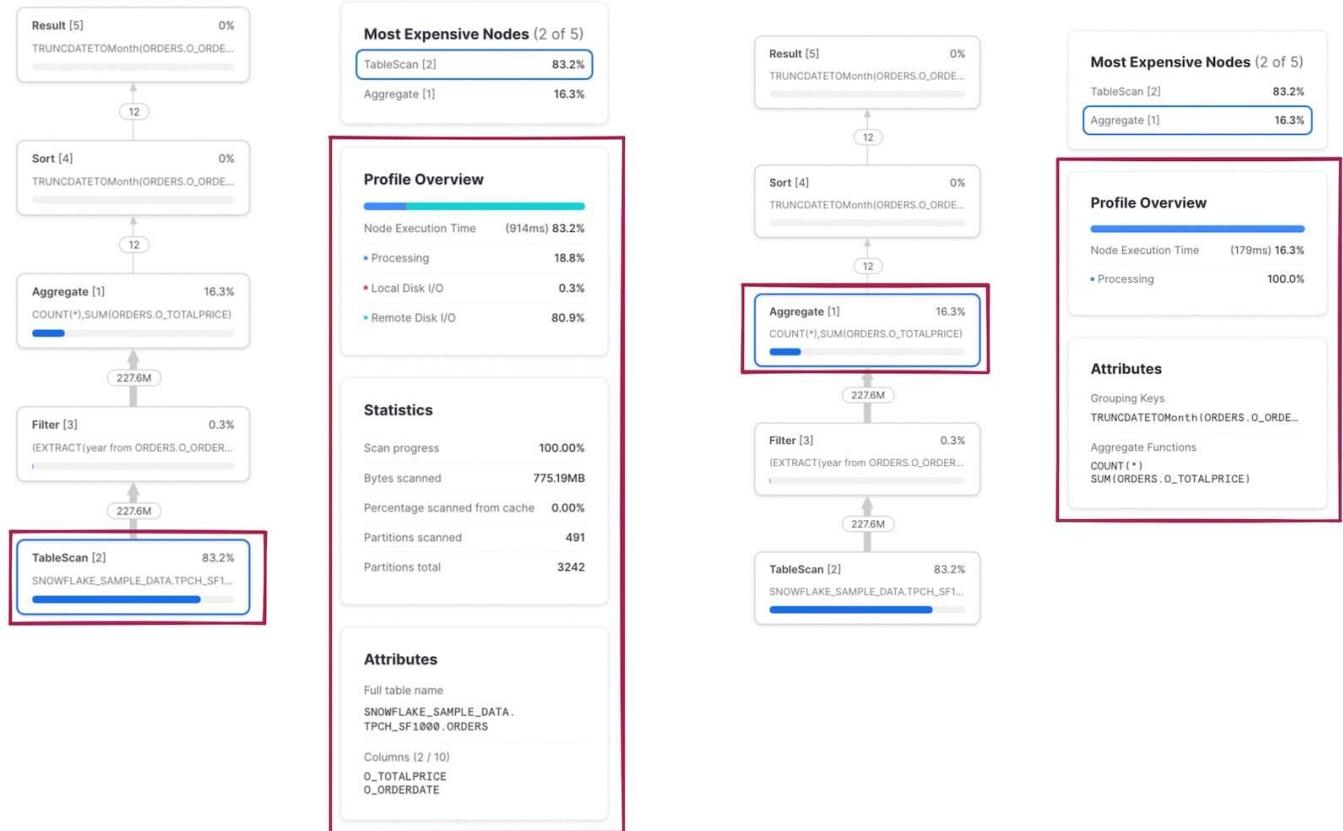
the [query is pruning well](#). If your warehouse doesn't have enough memory to process your query and is spilling to disk, this information will be reflected here.

3. Number of records shared between each node. This information is very helpful to understand the volume of data being processed, and how each node is reducing (or expanding) that number.
4. Percentage of total execution time spent on each node. Shown on the top right of each node, it indicates the percentage of total execution time spent on that operator. In this example, 83.2% of the total execution time was spent on the **TableScan** operator. This information is used to populate the "Most Expensive Nodes" list at the top right of the Query Profile, which simply sorts the nodes by the percentage of total execution time.



You may notice that the number of rows in/out of the **Filter** node are the same, implying that the `year(o_orderdate)=1997` SQL code did not accomplish anything. The filter is eliminating records though, as this table contains 1.5 billion records. This is an unfortunate pitfall of the Query Profile; it does not show the exact number of records being removed by a particular filter.

As mentioned earlier, you can click on each node to reveal additional execution details and statistics. On the left, you can see the results of clicking on the **TableScan** operator. On the right, the results of the **Aggregate** operator are shown.



Multi-step Query

When we modify the filter in the query above to contain a subquery, we end up with a multi-step query.

select

```
date_trunc('month', o_orderdate) as order_month,
```

```
count(*) as num_orders,
```

```
sum(o_totalprice) as total_order_value
```

```
from snowflake_sample_data.tpch_sf1000.orders
```

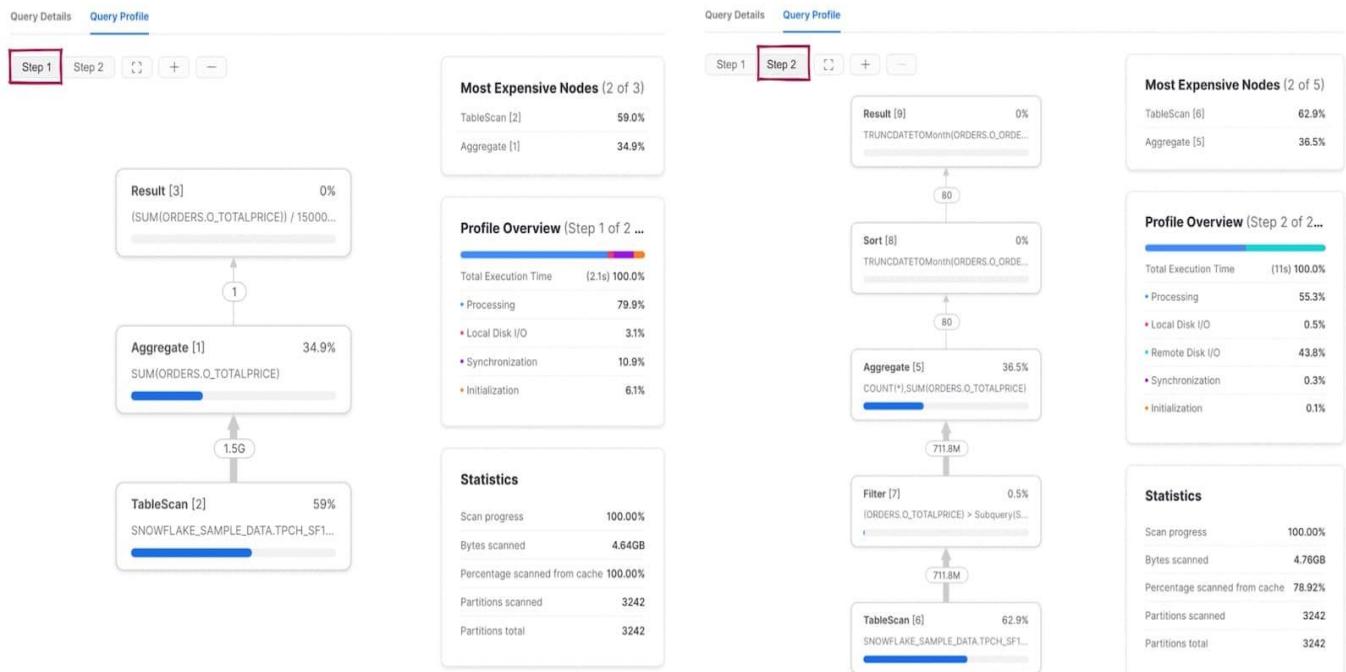
where

```
o_totalprice > (select avg(o_totalprice) from snowflake_sample_data.tpch_sf1000.orders)
```

group by order_month

```
order by order_month
```

Unlike above, the query plan now contains two steps. First, Snowflake executes the subquery and calculates the average `o_totalprice`. The result is stored, and used in the second step of the query which has the same 5 operators as our query above.



Complex Query

Here is a slightly more complex query² with multiple CTEs, one of which is referenced in two other places.

```
with
```

```
daily_shipments AS (
```

```
select
```

```
    l_shipdate,
```

```
    sum(l_quantity) AS num_items
```

```
from snowflake_sample_data.tpch_sf1000.lineitem
```

```
where
```

```
l_shipdate >= DATE'1998-01-01'

and l_shipdate <= DATE'1998-08-02'

group by 1

<),

daily_summary as (

select

o_orderdate,

count(*) AS num_orders,

any_value(num_items) AS num_items

from snowflake_sample_data.tpch_sf1000.orders

inner join daily_shipments

on orders.o_orderdate=daily_shipments.l_shipdate

group by 1

<),

summary_stats as (

select

min(num_items) as min_num_items,

max(num_items) as max_num_items

from daily_shipments

<>)
```

```
select
```

```
    daily_summary.*,
```

```
    summary_stats.*
```

```
from daily_summary
```

```
cross join summary_stats
```

There are a few things worth noting in this example. First, the `daily_shipments` CTE is computed just once. Any downstream SQL that references this CTE calls the `WithReference` operator to access the results of the CTE rather than re-calculating them each time.



Most Expensive Nodes (4 of ...)

TableScan [7]	83.5%
Aggregate [4]	6.9%
TableScan [9]	6.4%
Aggregate [6]	1.3%

Profile Overview (Finished)

Total Execution Time	(4.1s) 100.0%
• Processing	18.9%
• Local Disk I/O	0.5%
• Remote Disk I/O	77.6%
• Synchronization	2.7%
• Initialization	0.3%

Statistics

Scan progress	39.03%
Bytes scanned	533.41MB
Percentage scanned from cache	0.00%
Partitions scanned	1075
Partitions total	13578

The scanned/total partitions metric is now a combination of both tables being read in the query. If we click into the `TableScan` node for the `snowflake_sample_data.tpch_sf1000.orders` table, we can

see that the table is being pruned well, with only 154 out of 3242 partitions being scanned. How this pruning happening when there was no explicit `where` filter written in the SQL? This is the `JoinFilter` operator in action. Snowflake automatically applies this neat optimization where it determines the range of dates from the `daily_shipments` CTE during the query execution, and then applies those as a filter to the `orders` table since the query uses an `inner join`!



A full mapping of the SQL code to the relevant operator nodes can be found in the notes below [3](#).

What are things to look for in the Snowflake Query Profile?

The most common use case of the Query Profile is to understand why a particular query is not performing well. Now that we've covered the foundations, here are some indicators you can look for in the Query Profile as potential culprits of poor query performance:

1. High spillage to remote disk. As soon as there is any data spillage, it means your warehouse does not have enough memory to process the data and must temporarily store it elsewhere. Spilling data to remote disk is extremely slow, and will significantly degrade your query performance.
2. Large number of partitions scanned. Similarly to spilling data to remote disk, reading data from remote disk is also very slow. A large number of partitions scanned means your query has to do a lot of work reading remote data.
3. Exploding joins. If you see the number of rows coming out of a join increase, this may indicate that you have incorrectly specified your join key. Exploding joins generally take longer to process, and lead to other issues like spilling to disk.

4. Cartesian joins. Cartesian joins are a cross-join, which produce a result set which is the number of rows in the first table multiplied by the number of rows in the second table. Cartesian joins can be introduced unintentionally when using a non equi-join like a range join. Due to the volume of data produced, they are both slow and often lead to out-of-memory issues.
5. Downstream operators blocked by a single CTE. As discussed above, Snowflake computes each CTE once. If an operator relies on that CTE, it must wait until it is finished processing. In certain cases, it can be more beneficial to repeat the CTE as a subquery to allow for parallel processing.
6. Early, unnecessary sorting. It's common for users to add an unneeded sort early on in their query. Sorts are expensive, and should be avoided unless absolutely necessary.
7. Repeated computation of the same view. Each time a view is referenced in a query, it must be computed. If the view contains expensive joins, aggregations, or filters, it can sometimes be more efficient to materialize the view first.
8. A very large Query Profile with lots of nodes. Some queries just have too much going on and can be vastly improved by simplifying them. Breaking apart a query into multiple, simpler queries is an effective technique.

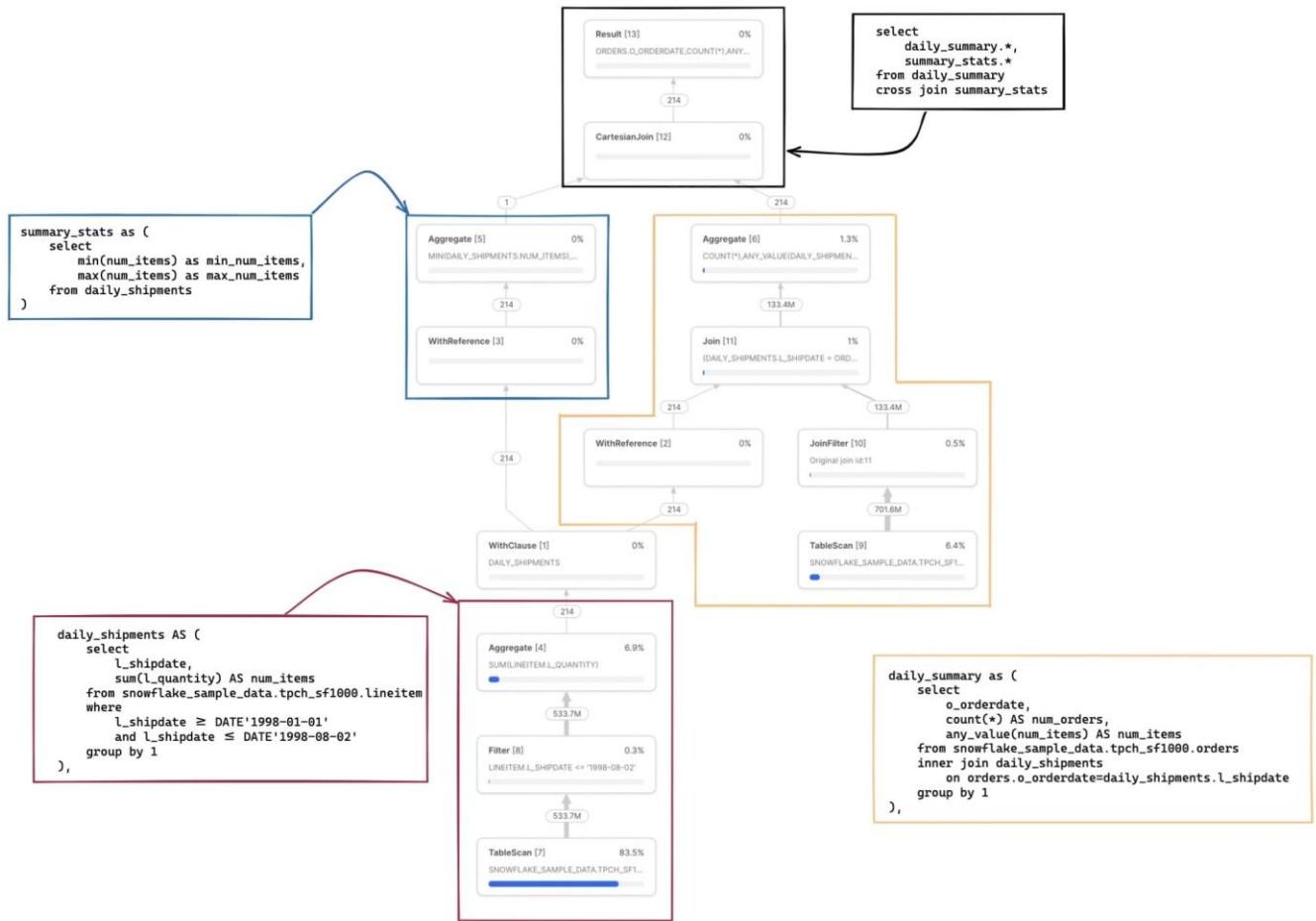
In future posts, we'll dive into each of these signals in more detail and share strategies to resolve them.

Notes

¹ Not all 1.3 million records are sent at once. Snowflake has a vectorized execution engine. Data is processed in a pipelined fashion, with batches of a few thousand rows in columnar format at a time. This is what allows an XSMALL warehouse with 16GB of RAM to process datasets much larger than 16GB. [←](#)

² I wouldn't pay too much attention to what this query is calculating or the way it's written. It was created solely for the purpose of yielding an interesting example query profile. [←](#)

³ For readers interested in improving their ability to read Snowflake Query Profiles, you can use the example query from above to see how each CTE maps to the different sections of the Query Profile.



Screenshot of the Snowflake Worksheet interface showing a query history and results.

Query History:

```

1. use role accountadmin;
2. select * from table(snowflake.information_schema.query_history()) order by start_time;
3. select * from table(snowflake.information_schema.query_history_by_user()) order by start_time;
4. select * from table(information_schema.query_history_by_warehouse()) order by start_time;
5.
6.
7.
8.
9.
10. WITH access_history AS (
    SELECT base_objects_accessed AS "QUERY_ID"
    ,SEPARATOR AS "DATABASE_NAME"
    ,split(base_value:objectName, ',')[]@[]::string "SCHEMA_NAME"
    ,split(base_value:objectName, ',')[]@[]::string "TABLE_NAME"
    ,split(base_value:objectName, ',')[]@[]::string "COLUMN_NAME"
    FROM snowflake.account_usage.access_history
    ,lateral flatten (base_objects_accessed) base
    )
    10._select_regex_replace(replace(replace(query_text,char(10),'_'),char(13,'_'),'_')) query_text

```

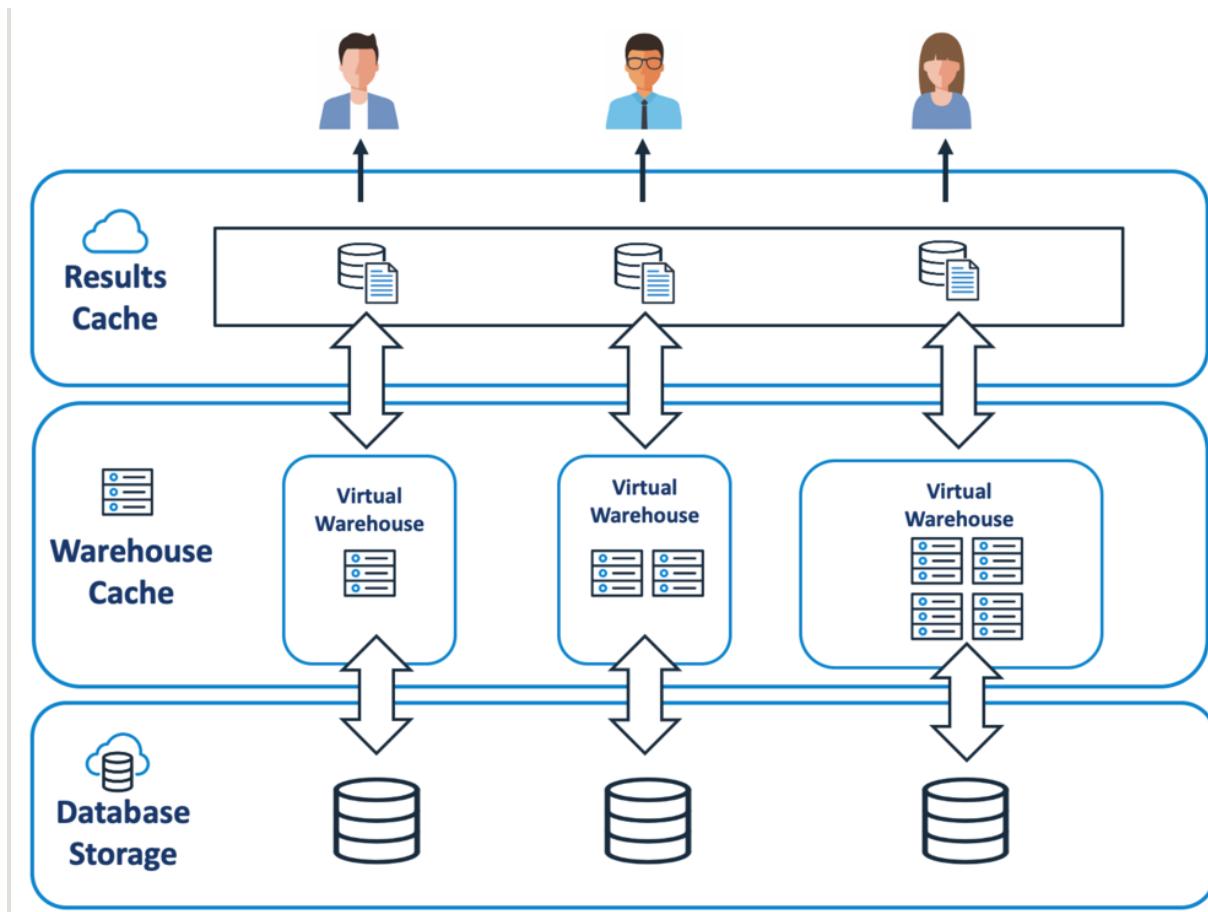
Results:

Row	QUERY_ID	QUERY_TEXT	DATABASE_NAME	SCHEMA_NAME	QUERY_TYPE	SESSION_ID	USER_NAME	ROLE_NAME	WAREHOUSE_N	WAREHOUSE_SI	WAREHOUSE_ST
1	01a22083-3...	select 1	NULL	NULL	SELECT	4479692805	SNOWFLAKE	ACCOUNTA...	NULL	NULL	STANDAI
2	01a22083-3...	CREATE OR ...	NULL	NULL	CREATE	4479692805	SNOWFLAKE	ACCOUNTA...	NULL	NULL	STANDAI
3	01a22083-3...	GRANT IMP...	SNOWFLAK...	NULL	GRANT	4479692805	SNOWFLAKE	ACCOUNTA...	NULL	NULL	STANDAI
4	01a22083-3...	GRANT ROL...	SNOWFLAK...	NULL	GRANT	4479692805	SNOWFLAKE	ACCOUNTA...	NULL	NULL	STANDAI
5	01a2209f-3...	CREATE WA...	NULL	NULL	CREATE	4479696917	SIDDHARTH	ACCOUNTA...	COMPUTE...	NULL	STANDAI
6	01a2209f-3...	use role sys...	NULL	NULL	USE	4479696921	SIDDHARTH	ACCOUNTA...	COMPUTE...	NULL	STANDAI

Snowflake Cache Layers

The diagram below illustrates the levels at which data and results are cached for subsequent use. These are:-

1. **Result Cache:** Which 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.
2. **Local Disk Cache:** Which 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.
3. **Remote Disk:** Which holds the long term storage. This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.99999999% durability. Even in the event of an entire data centre failure.



Snowflake Cache Layers

How to you clear a Snowflake cache?

Snowflake caches data in the Virtual Warehouse and in the Results Cache and these are controlled as separately.

Clear the Snowflake Virtual Warehouse Cache

Unlike many other databases, you cannot directly control the virtual warehouse cache. However, you can determine its size, as (for example), an X-Small virtual warehouse (which has one database server) is 128 times smaller than an X4-Large. Each increase in virtual warehouse size effectively doubles the cache size, and this can be an effective way of improving snowflake query performance, especially for very large volume queries.

You can also clear the virtual warehouse cache by suspending the warehouse and the SQL statement below shows the command.

```
alter warehouse prod_reporting_vwh suspend;
```

Be aware however, if you immediately re-start the virtual warehouse, Snowflake will try to recover the same database servers, although this is not guaranteed.

Because suspending the virtual warehouse clears the cache, it is good practice to set an automatic suspend to around ten minutes for warehouses used for online queries, although warehouses used for batch processing can be suspended much sooner.

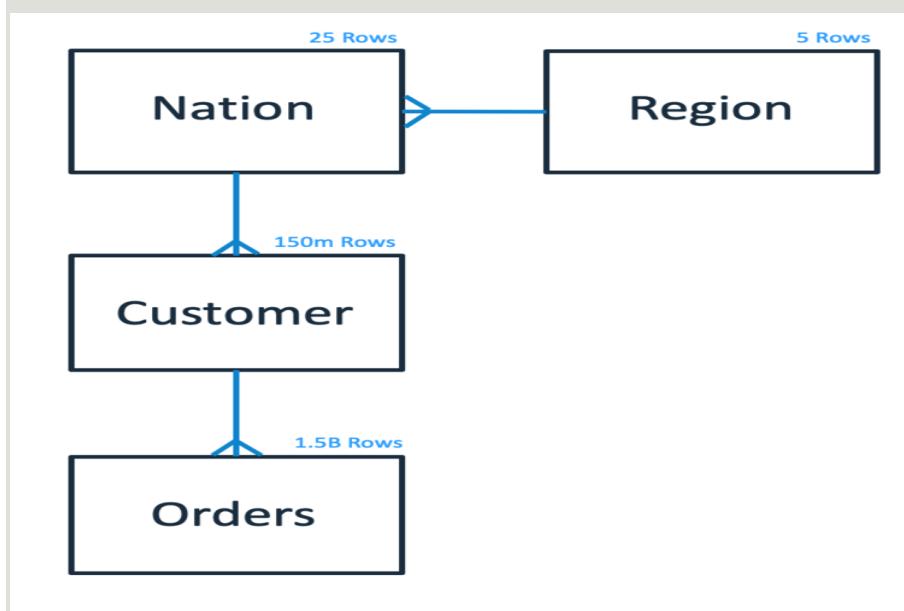
Disable the Snowflake Database Results Cache

While it is not possible to clear or disable the virtual warehouse cache, the option exists to [disable the results cache](#), although this only makes sense when benchmarking query performance. Use the following SQL statement:

```
alter session set use_cached_result = false;
```

Benchmark the Snowflake Cache

Every Snowflake database is delivered with a pre-built and populated set of Transaction Processing Council (TPC) benchmark tables. To test the result of caching, I set up a series of test queries against a small sub-set of the data, which is illustrated below.



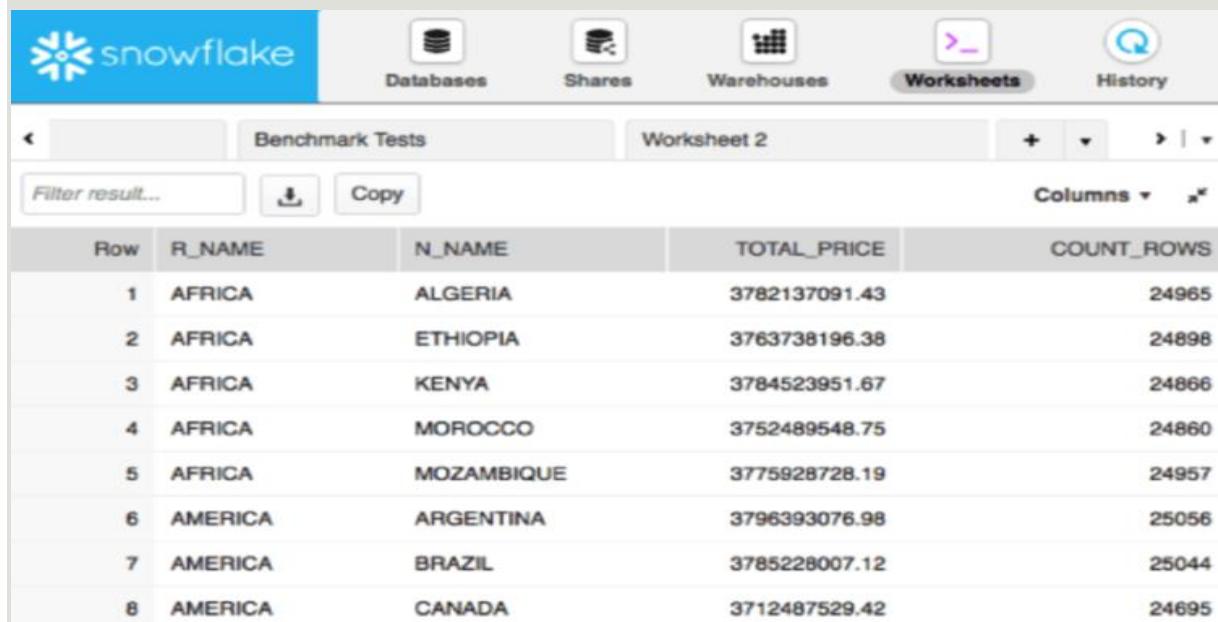
Transaction Processing Council - Benchmark Table Design

All the queries were executed on a MEDIUM sized cluster (4 nodes), and joined the tables. The tables were queried exactly as is, without any performance tuning.

The following query was executed multiple times, and the elapsed time and query plan were recorded each time.

```
select r_name
      , n_name
      , sum(o_totalprice)    as
total_price
      , count(*)           as
count_rows
  from orders o
  join customer c
    on c_custkey = o_custkey
  join region r
    on r_regionkey = n_regionkey
 where o_orderdate = '01-APR-199'
 group by r_name
      , n_name
  order by r_name
      , n_name
```

The screen shot below illustrates the results of the query which summarise the data by Region and Country. In total the SQL queried, summarised and counted over 1.5 Billion rows. The screenshot shows the first eight lines returned.



The screenshot shows the Snowflake interface with the 'Worksheets' tab selected. The results of the query are displayed in a table with the following data:

Row	R_NAME	N_NAME	TOTAL_PRICE	COUNT_ROWS
1	AFRICA	ALGERIA	3782137091.43	24965
2	AFRICA	ETHIOPIA	3763738196.38	24898
3	AFRICA	KENYA	3784523951.67	24866
4	AFRICA	MOROCCO	3752489548.75	24860
5	AFRICA	MOZAMBIQUE	3775928728.19	24957
6	AMERICA	ARGENTINA	3796393076.98	25056
7	AMERICA	BRAZIL	3785228007.12	25044
8	AMERICA	CANADA	3712487529.42	24695

Benchmark Test Sequence

The test sequence was as follows:-

1. **Run from cold:** Which meant starting a new virtual warehouse (with no local disk caching), and executing the query.
2. **Run from warm:** Which meant disabling the result caching, and repeating the query. This makes use of the local disk caching, but not the result cache.
3. **Run from hot:** Which again repeated the query, but with the result caching switched on.

Each query ran against 60Gb of data, although as Snowflake returns only the columns queried, and was able to automatically compress the data, the actual data transfers were around 12Gb. As Snowflake is a columnar data warehouse, it automatically returns the columns needed rather than the entire row to further help maximise query performance.

Query Performance with no Cache

This query returned in around 20 seconds, and demonstrates it scanned around 12Gb of compressed data, with 0% from the local disk cache. This means it had no benefit from disk caching.

Profile Overview (Finished)

Total Execution Time (20.447s)



Total Statistics

IO

Scan progress	100.00 %
Bytes scanned	12.48 GB
Percentage scanned from cache	0.00 %

Network

Bytes sent over the network	27.50 MB
-----------------------------	----------

Pruning

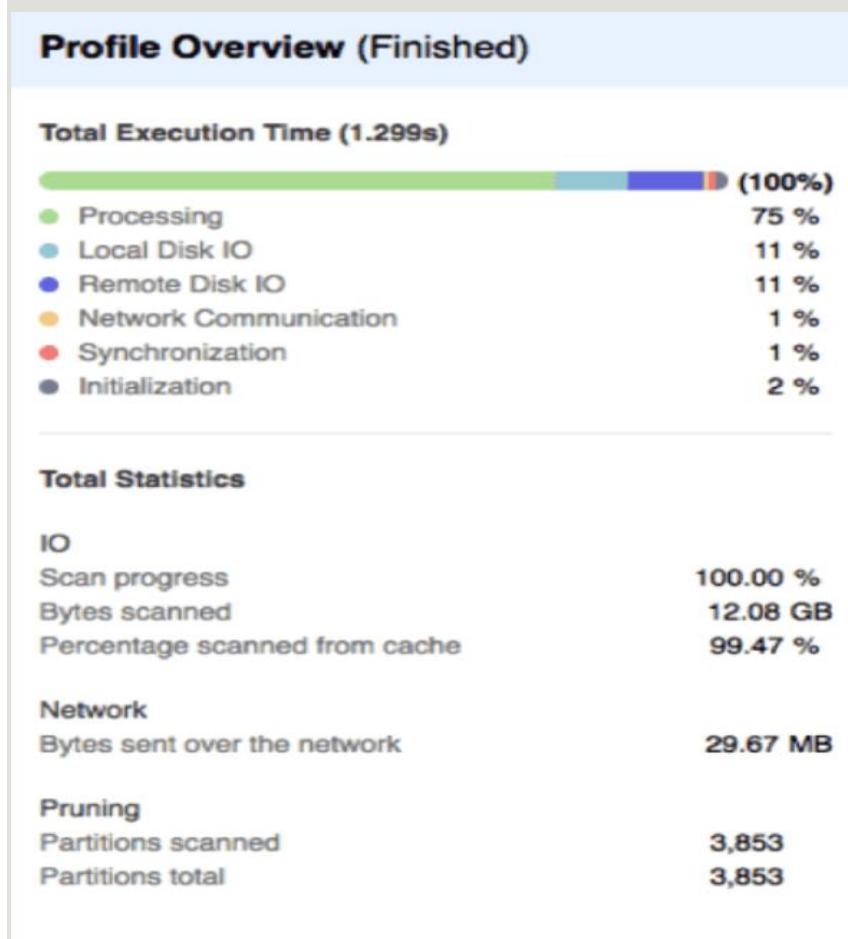
Partitions scanned	3,853
Partitions total	3,853

The bar chart above demonstrates around 50% of the time was spent on local or remote disk I/O, and only 2% on actually processing the data. Clearly any design changes we can do to reduce the disk I/O will help this query.

The results also demonstrate the queries were unable to perform any *partition pruning* which might improve query performance. We'll cover the effect of partition pruning and clustering in the next article.

Query using Virtual Warehouse Database Cache

This query was executed immediately after, but with the result cache disabled, and it completed in 1.2 seconds – around 16 times faster. In this case, the *Local Disk* cache (which is actually SSD on Amazon Web Services) was used to return results, and disk I/O is no longer a concern.



In the above case, the disk I/O has been reduced to around 11% of the total elapsed time, and 99% of the data came from the (local disk) cache. While querying 1.5 billion rows, this is clearly an excellent result.

Query using the Results Cache

This query returned results in milliseconds, and involved re-executing the query, but with this time, the result cache enabled. Normally, this is the default situation, but it was disabled purely for testing purposes.

Profile Overview (Finished)

Total Execution Time (2ms)



The above profile indicates the entire query was served directly from the result cache (taking around 2 milliseconds). Although not immediately obvious, many dashboard applications involve repeatedly refreshing a series of screens and dashboards by re-executing the SQL. In these cases, the results are returned in milliseconds.

Although more information is available in the [Snowflake Documentation](#), a series of tests demonstrated the result cache will be reused unless the underlying data (or SQL query) has changed. As a series of additional tests demonstrated inserts, updates and deletes which don't affect the underlying data are ignored, and the result cache is used, provided data in the micro-partitions remains unchanged.

Finally, results are normally retained for 24 hours, although the clock is reset every time the query is re-executed, up to a limit of 30 days, after which results query the remote disk.

Snowflake Performance Summary

The sequence of tests was designed purely to illustrate the effect of data caching on Snowflake. The tests included:-

- **Raw Data:** Including over 1.5 billion rows of TPC generated data, a total of over 60Gb of raw data
- **Initial Query:** Took 20 seconds to complete, and ran entirely from the remote disk. Quite impressive.
- **Second Query:** Was 16 times faster at 1.2 seconds and used the *Local Disk (SSD)* cache.
- **Result Set Query:** Returned results in 130 milliseconds from the result cache (intentionally disabled on the prior query).

To put the above results in context, I repeatedly ran the same query on Oracle 11g production database server for a tier one investment bank and it took over 22 minutes to complete.

Finally, unlike Oracle where additional care and effort must be made to ensure correct partitioning, indexing, stats gathering and data compression, Snowflake caching is entirely automatic, and available by default. Absolutely no effort was made to tune either the queries or the underlying design, although there are a small number of options available, which I'll discuss in the next article. Sign up below for further details.

Snowflake Cache Best Practice

Clearly data caching data makes a massive difference to Snowflake query performance, but what can you do to ensure maximum efficiency when you cannot adjust the cache?

Here's a few best practice tips:-

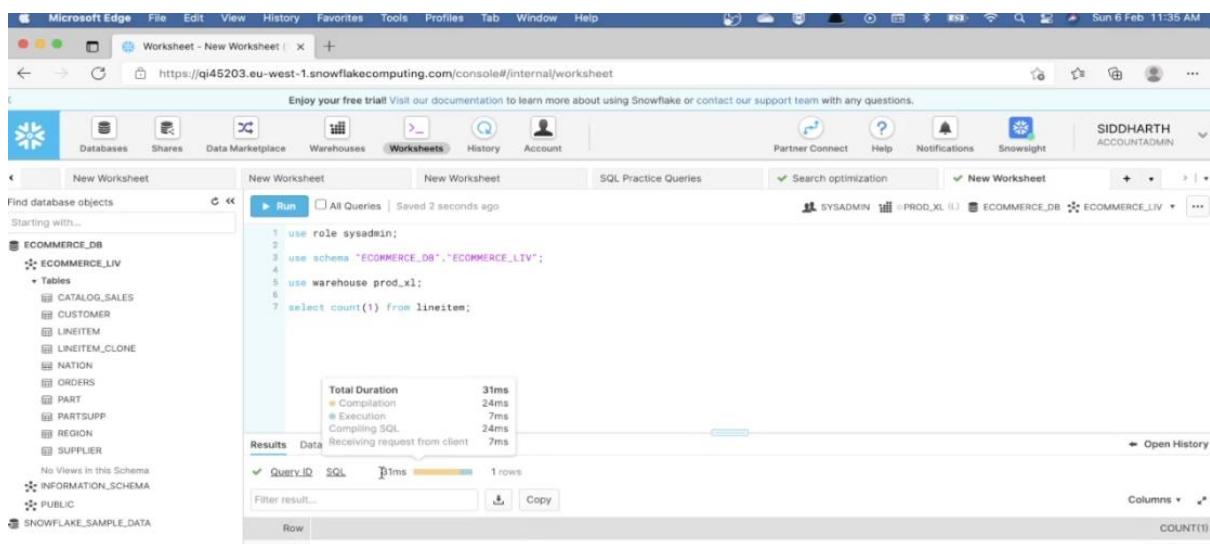
- **Auto-Suspend:** By default, Snowflake will auto-suspend a virtual warehouse (the compute resources with the SSD cache after 10 minutes of idle time. Best practice? **Leave this alone!** Keep in mind, you should be trying to balance the cost of providing compute resources with fast query performance. To illustrate the point, consider these two extremes:
 1. **If you auto-suspend after 60 seconds:** When the warehouse is re-started, it will (most likely) start with a clean cache, and will take a few queries to hold the relevant cached data in memory. (Note: Snowflake will try to restore the same cluster, with the cache intact, but this is not guaranteed).
 2. **If you never suspend:** Your cache will always be *warm*, but you will pay for compute resources, even if nobody is running any queries. However, provided you set up a script to shut down the server when not being used, then maybe (just maybe), it may make sense.
- **Auto-Suspend Best Practice?** Is remarkably simple, and falls into one of two possible options:
 1. **Online Warehouses:** Where the virtual warehouse is used by online query users, leave the auto-suspend at 10 minutes. This means if there's a short break in queries, the cache remains warm, and subsequent queries use the query cache.
 2. **Batch Processing Warehouses:** For warehouses entirely deployed to execute batch processes, suspend the warehouse after 60 seconds. The performance of an individual query is not quite so important as the overall throughput, and it's therefore unlikely a batch warehouse would rely on the query cache.

Scaling the Virtual Warehouse Cache

- **Scale up for large data volumes:** If you have a sequence of large queries to perform against massive (multi-terabyte) size data volumes, you can improve workload performance by scaling up. Simple execute a SQL statement to increase the virtual warehouse size, and new queries will start on the larger (faster) cluster. While this will start with a clean (empty) cache, you should normally find performance doubles at each size, and this extra performance boost will more than out-weigh the cost of refreshing the cache. However, be aware, if you scale up (or down) the data cache is cleared.
- **Scale down - but not too soon:** Once your large task has completed, you could reduce costs by scaling down or even suspending the virtual warehouse. Be aware again however, the cache will start again clean on the smaller cluster. By all means tune the warehouse size dynamically, but don't keep adjusting it, or you'll lose the benefit.

Snowflake - Query processing & Caching

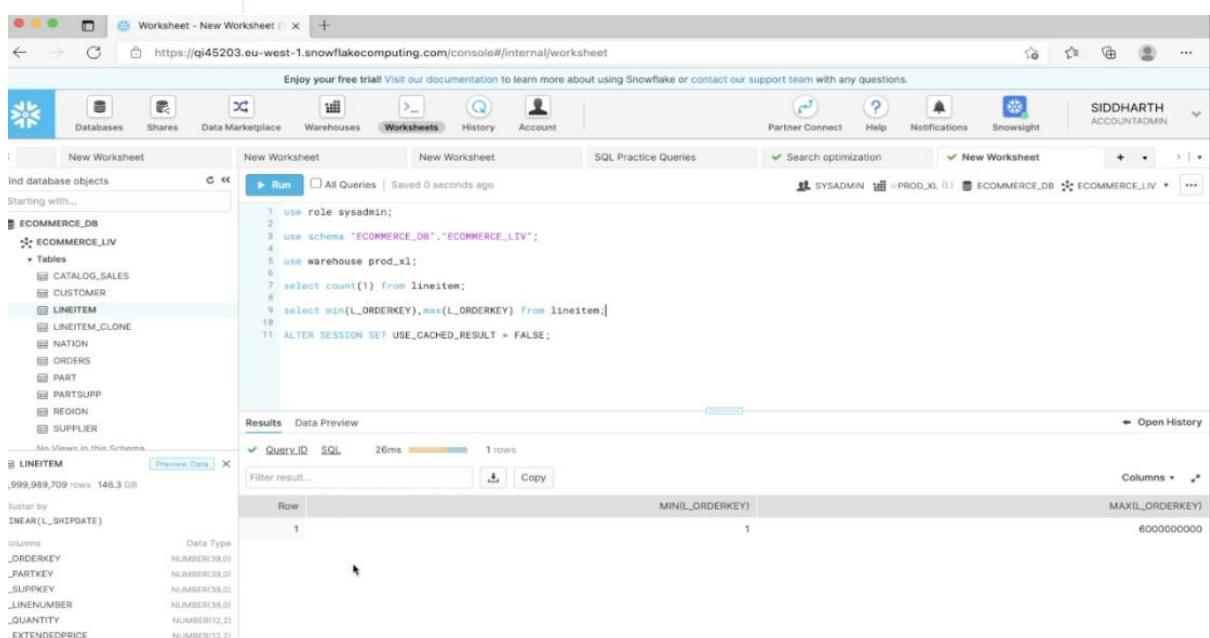
- Snowflake caches three types of data :
 - Query Results
 - The data from tables in the warehouse storage (SSD / local cache)
 - Metadata (Cloud Services Layer)
- Metadata Cache is always persisted and updated , never dropped .
- Query Results cache expires after 24 hours of non-usage or if the underlying data changes .
- Warehouse cache is dropped as soon as the warehouse is suspended .



The screenshot shows the Snowflake Worksheet interface. A query is run against the ECOMMERCE_DB schema to count the number of rows in the LINEITEM table. The results show a total duration of 31ms, with breakdowns for compilation, execution, and compilation SQL. The output displays a single row with COUNT(1) as 5999989709.

```
use role sysadmin;
use schema "ECOMMERCE_DB"."ECOMMERCE_LIV";
use warehouse prod_xl;
select count(1) from lineitem;
```

Row	COUNT(1)
1	5999989709



The screenshot shows the Snowflake Worksheet interface again. A query is run to find the minimum and maximum ORDERKEY values from the LINEITEM table. The results show a total duration of 26ms, with breakdowns for compilation, execution, and compilation SQL. The output displays a single row with MIN(L_ORDERKEY) as 1 and MAX(L_ORDERKEY) as 6000000000.

```
use role sysadmin;
use schema "ECOMMERCE_DB"."ECOMMERCE_LIV";
use warehouse prod_xl;
select count(1) from lineitem;
select min(L_ORDERKEY), max(L_ORDERKEY) from lineitem;
ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

Row	MIN(L_ORDERKEY)	MAX(L_ORDERKEY)
1	1	6000000000

Microsoft Edge File Edit View History Favorites Tools Profiles Tab Window Help Sun 6 Feb 11:37 AM

Worksheet - New Worksheet x + https://qi45203.eu-west-1.snowflakecomputing.com/console#/internal/worksheet Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

Databases Shares Data Marketplace Warehouses Worksheets History Account Partner Connect Help Notifications Snowsight SIDDHARTH ACCOUNTADMIN

New Worksheet New Worksheet SQL Practice Queries Search optimization New Worksheet + | v

Find database objects Starting with... Run All Queries Saved 51 seconds ago

ECOMMERCE_DB

- ECOMMERCE_LIV
 - Tables
 - CATALOG_SALES
 - CUSTOMER
 - LINEITEM**
 - LINEITEM_CLONE
 - NATION
 - ORDERS
 - PART
 - PARTSUPP
 - REGION
 - SUPPLIER

Mn. Views in this Schema: LINEITEM Preview Data x 5,999,989,709 rows. 146.3 GB

Cluster by LINEAR(L_SHIPDATE)

Columns

L_ORDERKEY	NUMBER(38,0)
L_PARTKEY	NUMBER(38,0)
L_SUPPKEY	NUMBER(38,0)

```

14 set current_dt='1998-08-01';
15
16 select
17     sum(L.quantity) as sum_qty,
18     count(*) as order_count,
19     date(L.shipdate) as shipped_date,
20     L.shipped_mode as shipped_mode
21
22 from
23 LINEITEM
24 where
25     shipped_date >= date($current_dt) - 7
26 group by
27     shipped_date,
28     shipped_mode
29 order by
30     shipped_date;

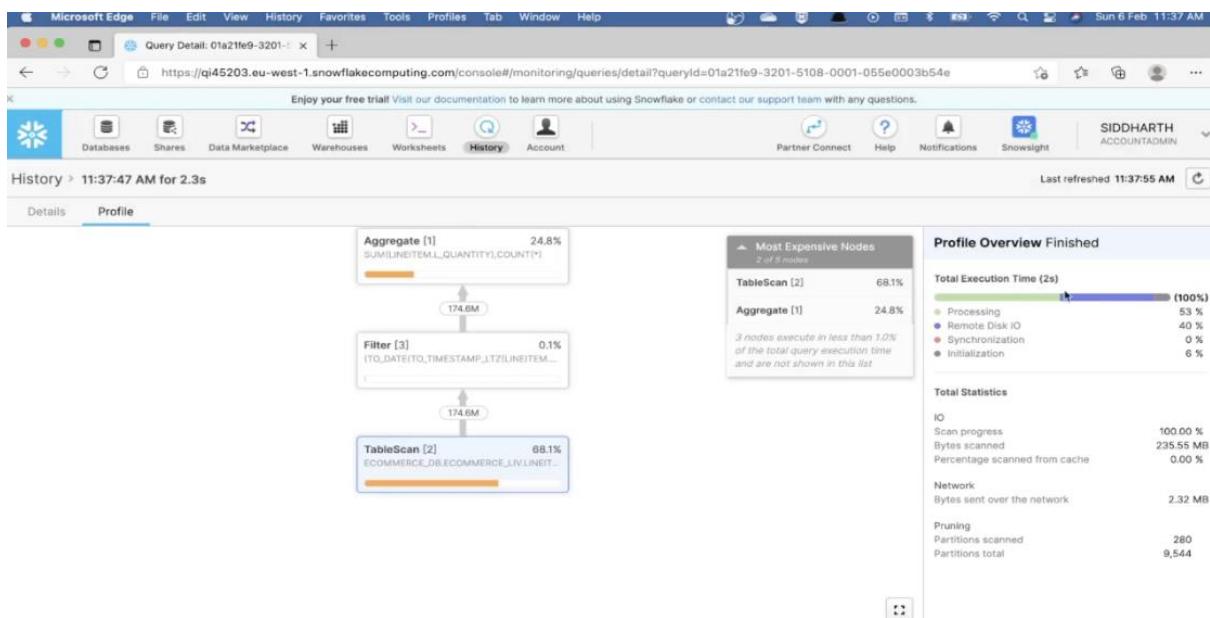
```

Results Data Preview Open History

Query ID: 2.32s 910 rows

Row	SUM_QTY	ORDER_COUNT	SHIPPED_DATE	SHIPPED_MODE
1	9083081.00	356496	1998-07-25	FOB
2	911290.00	357331	1998-07-25	RAIL
3	9082579.00	355845	1998-07-25	REG AIR
4	9097086.00	356789	1998-07-25	SHIP

Columns ▾



Snowflake - Search Optimization

- Data Workloads contain 2 types of queries :
 - Analytical Queries
 - Eg : Aggregation SQL
 - select sum(column),count(distinct column) , date_column from table group by date_column *
 - Point Lookup Queries
 - Eg : Select a few rows and all columns by using any random column in the where clause
 - Select * from table where column = "abc"

Now when it comes to your workloads and your data warehouse, the cloud in a warehouse, on premise data warehouse, you can essentially split them into two types. The first one is analytical workloads or analytical queries. Now these queries, essentially aggregation, Times Square is used for reporting or data science purposes. You select, you sum up a column, you take a cloud of another column, or you apply some aggregate functions and you select a bunch of other columns and you grew by them.

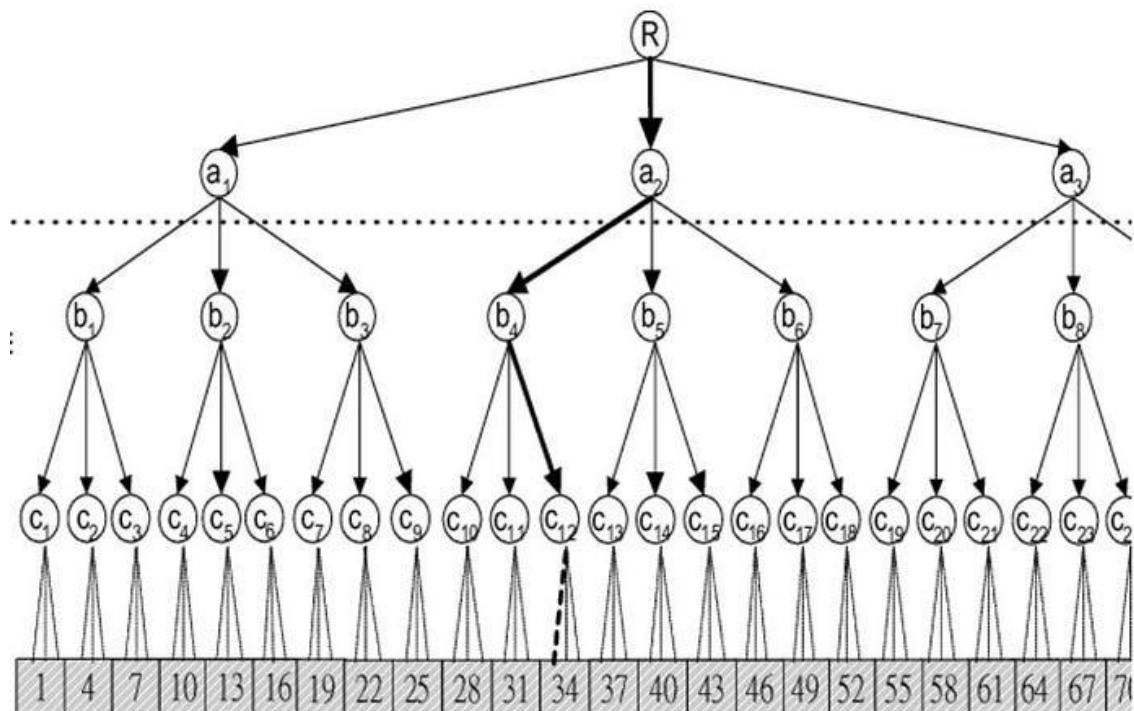
Now, this is a typical analytical workload. The next step is your point lookup queries, and point lookup queries are the quarries that literally select all of your columns from the table and expect a few rows in return using a word predicate clause. Now, this word condition might not necessarily be one of your clustering keys. Let's say you have a table line item, something you've seen in the past, and you have a downstream obligation that constantly queries this table. And does not use the clustering key and it's expecting all of the columns neatly so. These kind of queries that select all of the columns and expect a few rows are called wine lookup queries. So a table like select start from table, a query like select starting table where column name is equal to some value is a point lookup query.

Snowflake - Search Optimization

- Tables can benefit from Search optimization if :
 - Tables are not clustered
 - Table is frequently queried on columns which are not used in the clustering keys
 - Table has a lot of point-lookup queries
- Search Optimization Feature impacts costs for :
 - Storage resources
 - Compute resources
- Cost depends on :
 - Number of tables using this feature
 - The amount of data in the table(s)

What is Search Optimization Service?

Search Optimization Service is an optimized data structure, called the **search access path** which is created by Snowflake as background maintenance service which scans all the table micro partition and records the metadata. This metadata helps snowflake to build the best search access path for table data. When a user fires a query, snowflake optimizer looks into the shortest and best access path for that data to give the quickest response if SOS is enabled for that table.



Pictorial Presentation of search access path

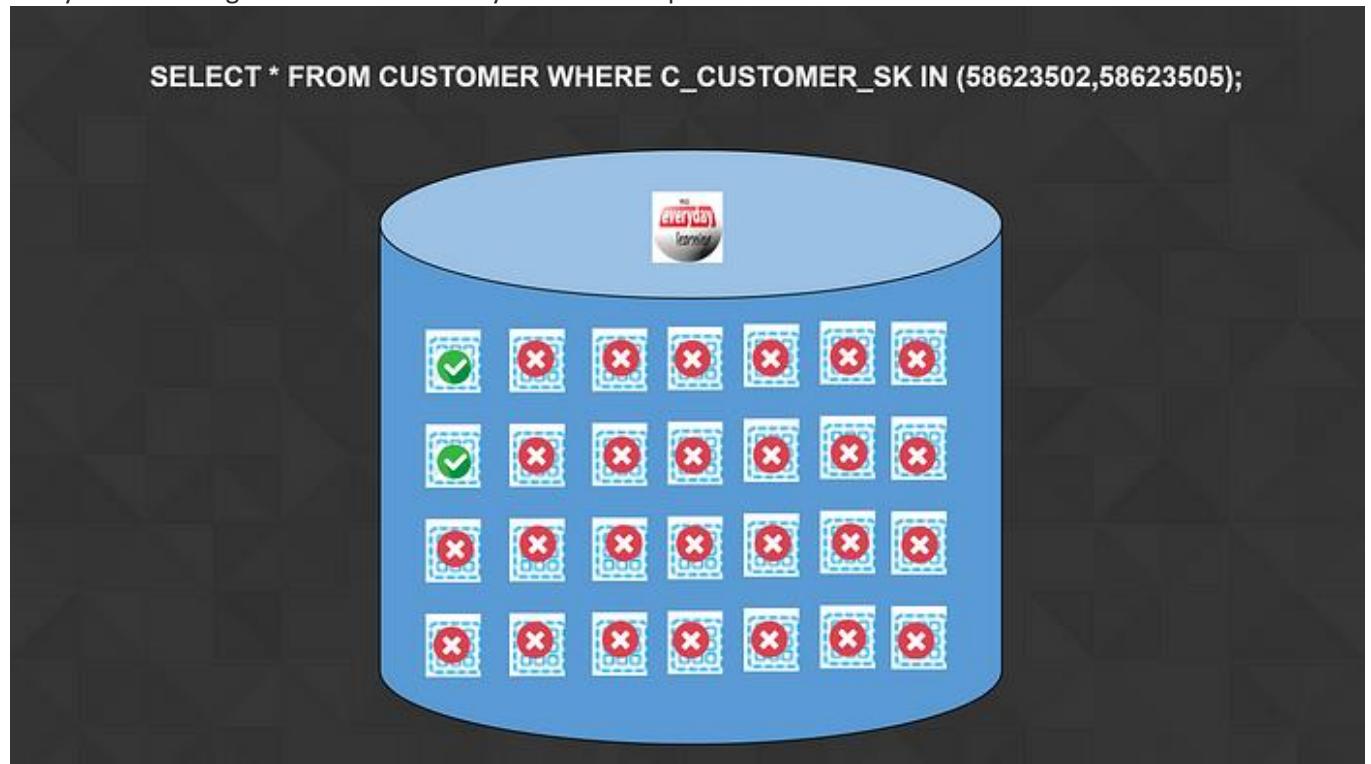
The search optimization service can significantly improve the performance of certain types of lookup and analytical queries that use an extensive set of predicates for filtering.

Search Optimization Service incur both storage and compute cost.

How does Search Optimization Service work at high level?

Search Optimization Service build the search access path which helps eliminate the micro-partition which doesn't have requested data. This helps prune micro-partition which doesn't contain matching data which helps to scan very less amount of micro-partition.

In the below diagram you can see that the query is only looking for value(i.e. 58623502,58623505) from customer table. Since search optimization service is enabled for this table which has billions of records this make optimizer use for SOS service as best option to query the data. The pictorial image below is to make you understand that as per SOS search access path only 2 micro-partition out of many has matching data hence it will only scan 2 micro-partition.



Concept Courtesy Snowflake.

I know the thought came to your mind than what is the difference between SOS & table clustering, both are doing the same thing i.e. partition pruning. We will cover this part later in this blog.

How to identify the best candidate table for Search Optimization Service?

Below could be taken as approach for SOS candidate selection:

1. **For Clustered Table** -Table is frequently queries on non-cluster key column
2. **For Non-Clustered Table** -Table is frequently queries on any column.

How to identify the best candidate queries for Search Optimization Service?

Below could be taken as approach for SOS query candidate selection:

1. A query that typically runs for a few seconds or longer.
2. The query returns a few rows with highly selective filters.
3. A query in which at least one of the columns accessed through the query filter operation has at least 1- 2 lakh distinct values.

— Use APPROX_COUNT_DISTINCT to get the approximate number of distinct values:

```
SELECT APPROX_COUNT_DISTINCT(C_CUSTOMER_SK) FROM CUSTOMER;
```

How can we enable Search Optimization Service for a table ?

Equality or IN Predicates(General Available):

By default, if you don't provide any column name while enabling Search Optimization Service for a table than it will enable **equality** SOS for all column of table which supports any lookup queries using =, IN clause.

Consider below example where I just add Search Optimization Service for Customer table.

— How to enable SOS for table?

ALTER TABLE CUSTOMER ADD SEARCH OPTIMIZATION;

— How to check the list of column for which SOS is enabled ?

DESCRIBE SEARCH OPTIMIZATION ON CUSTOMER;

Row	expression_id	method	target	target_data_type	active
1	1	EQUALITY	C_CUSTOMER_SK	NUMBER(38,0)	true
2	2	EQUALITY	C_CUSTOMER_ID	VARCHAR(16)	true
3	3	EQUALITY	C_CURRENT_CDENO_SK	NUMBER(38,0)	true
4	4	EQUALITY	C_CURRENT_HDEMO_SK	NUMBER(38,0)	true
5	5	EQUALITY	C_CURRENT_ADDR_SK	NUMBER(38,0)	true
6	6	EQUALITY	C_FIRST_SHIPTO_DATE_SK	NUMBER(38,0)	true
7	7	EQUALITY	C_FIRST_SALES_DATE_SK	NUMBER(38,0)	true
8	8	EQUALITY	C_SALUTATION	VARCHAR(10)	true
9	9	EQUALITY	C_FIRST_NAME	VARCHAR(20)	true
10	10	EQUALITY	C_LAST_NAME	VARCHAR(30)	true
11	11	EQUALITY	C_PREFERRED_CUST_FLAG	VARCHAR(1)	true
12	12	EQUALITY	C_BIRTH_DAY	NUMBER(38,0)	true
13	13	EQUALITY	C_BIRTH_MONTH	NUMBER(38,0)	true
14	14	EQUALITY	C_BIRTH_YEAR	NUMBER(38,0)	true
15	15	EQUALITY	C_BIRTH_COUNTRY	VARCHAR(20)	true
16	16	EQUALITY	C_LOGIN	VARCHAR(13)	true
17	17	EQUALITY	C_EMAIL_ADDRESS	VARCHAR(50)	true
18	18	EQUALITY	C_LAST REVIEW_DATE	VARCHAR(10)	true

Substrings and Regular Expressions(Preview feature):

If we want to improve the performance of queries with predicates that search for **substrings or use regular expressions** than we can enable Substring SOS for a column in a table.

The search optimization service can improve performance when searching for substrings that are **5 or more characters long**. Search with lesser character than 5 will not trigger SOS even if you have SOS enable for that column.

This includes predicates that use:

- LIKE
- CONTAINS
- RLIKE
- LIKE ANY
- ENDSWITH
- REGEXP
- LIKE ALL
- STARTSWITH
- REGEXP_LIKE
- ILIKE
- SPLIT_PART (in equality predicates)
- ILIKE ANY

— How to enable Substring SOS for a all column in a table?

```
ALTER TABLE CUSTOMER_SUBSTRING ADD SEARCH OPTIMIZATION ON SUBSTRING(*) ;
```

```
DESCRIBE SEARCH OPTIMIZATION ON CUSTOMER_SUBSTRING;
```

expression_id	method	target	target_data_type	active
1	SUBSTRING	C_CUSTOMER_ID	VARCHAR(16)	true
2	SUBSTRING	C_SALUTATION	VARCHAR(10)	true
3	SUBSTRING	C_FIRST_NAME	VARCHAR(20)	true
4	SUBSTRING	C_LAST_NAME	VARCHAR(30)	true
5	SUBSTRING	C_PREFERRED_CUST_FLAG	VARCHAR(1)	true
6	SUBSTRING	C_BIRTH_COUNTRY	VARCHAR(20)	true
7	SUBSTRING	C_LOGIN	VARCHAR(13)	true
8	SUBSTRING	C_EMAIL_ADDRESS	VARCHAR(50)	true
9	SUBSTRING	C_LAST REVIEW_DATE	VARCHAR(10)	true

— How to enable Substring SOS for a particular column?

```
ALTER TABLE CUSTOMER ADD SEARCH OPTIMIZATION ON SUBSTRING(C_CUSTOMER_ID) ;
```

expression_id	method	target	target_data_type	active
19	SUBSTRING	C_CUSTOMER_ID	VARCHAR(16)	true

Geospatial Functions(Preview feature):

If we want to improve the performance of queries with predicates that use geospatial functions with GEOGRAPHY objects than we can enable GEO SOS for a column in a table. Look to Snowflake documentation for [supported predicates with geospatial functions](#).

— -Create Geography data type table

```
CREATE OR REPLACE TABLE MAP (ID NUMBER, G1 GEOGRAPHY);
```

— Insert some Geo data

```
INSERT INTO MAP VALUES  
(1, 'POINT(-122.35 37.55)'),  
(2, 'LINESTRING(-124.20 42.00, -120.01 41.99)'),  
(3, 'POLYGON((0 0, 2 0, 2 2, 0 2, 0 0))');
```

— How to enable GEO SOS for a particular column?

```
ALTER TABLE MAP ADD SEARCH OPTIMIZATION ON GEO(G1) ;
```

— How to check the list of column for which SOS is enabled ?

DESCRIBE SEARCH OPTIMIZATION ON MAP;

expression_id	method	target	target_data_type	active
1	GEO	G1	GEOGRAPHY	true

— How to enable GEO SOS for a all column in a table?

ALTER TABLE MAP ADD SEARCH OPTIMIZATION ON GEO(*) ;

VARIANT Columns(Preview feature):

If we want to improve the performance of queries with predicates of **point lookup queries on semi-structured data in Snowflake tables**(data in VARIANT, OBJECT, and ARRAY columns) than we can enable SOS for a matching datatype column in a table. For current limitation look into [snowflake documentation](#).

See below example where we had enabled the SOS on src variant data type column of CAR_SALES table.

name	type	kind	null?	default	primary key	unique key	check	expression	comment	policy name
SRC	VARIANT	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL

— How to enable EQUALITY SOS for a variant, array, object data type column in a table?

ALTER TABLE CAR_SALES ADD SEARCH OPTIMIZATION ON EQUALITY(SRC);

— How to check the list of column for which SOS is enabled ?

```
DESCRIBE SEARCH OPTIMIZATION ON CAR_SALES;
```

expression_id	method	target	target_data_type	active
1	EQUALITY	SRC	VARIANT	true

— How to enable EQUALITY SOS for all variant, array, object data type column in a table?

```
ALTER TABLE CAR_SALES ADD SEARCH OPTIMIZATION ON EQUALITY(*);
```

How to verify if Search Optimization Service is enabled and active for a table?

You can use SHOW TABLES command and look at "search_optimization" column. If Search Optimization Service is enabled it will have value "ON".

After enabling SOS it takes some time to create a search access path for that table via background maintenance service. It also depends on the size of the table. This progress can be monitored via "search_optimization_progress" column which says up to what percentage of work has been completed. Once it is 100% than your search access path is ready to use. Since this is background process, so it is transparent to user and don't interrupt any processing on table. You might not get the power of SOS or query might be a bit slow, but you will always see up-to-date data.

The search optimization service creates a search access path data structure that requires space for each table on which search optimization is enabled. Typically, the size is approximately 1/4 of the original table's size but sometime can be the same as the table size where all columns have data types that use the search access path, and all data values in each column are unique, the required storage can be as much as the original table's size. You can see the SOS size under "search_optimization_bytes" column

```
SHOW TABLES LIKE 'CUSTOMER';
```

```
SELECT
```

```
"name","rows","bytes","automatic_clustering","search_optimization","search_optimization_progres  
s","search_optimization_bytes" FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

name	rows	bytes	automatic_clustering	search_optimization	search_optimization_progress	search_optimization_bytes
CUSTOMER	10000000	3598690816	OFF	ON	100	1604073984

What are queries not supported by the Search Optimization Service?



How to drop the Search Optimization Service?

To drop search optimization for substrings on the column C_CUSTOMER_ID, execute the following statement:

```
ALTER TABLE CUSTOMER DROP SEARCH OPTIMIZATION ON SUBSTRING(C_CUSTOMER_ID) ;
```

To drop search optimization for all methods on the column C_CURRENT_CDEMO_SK, execute the following statement

```
ALTER TABLE CUSTOMER DROP SEARCH OPTIMIZATION ON C_CURRENT_CDEMO_SK;
```

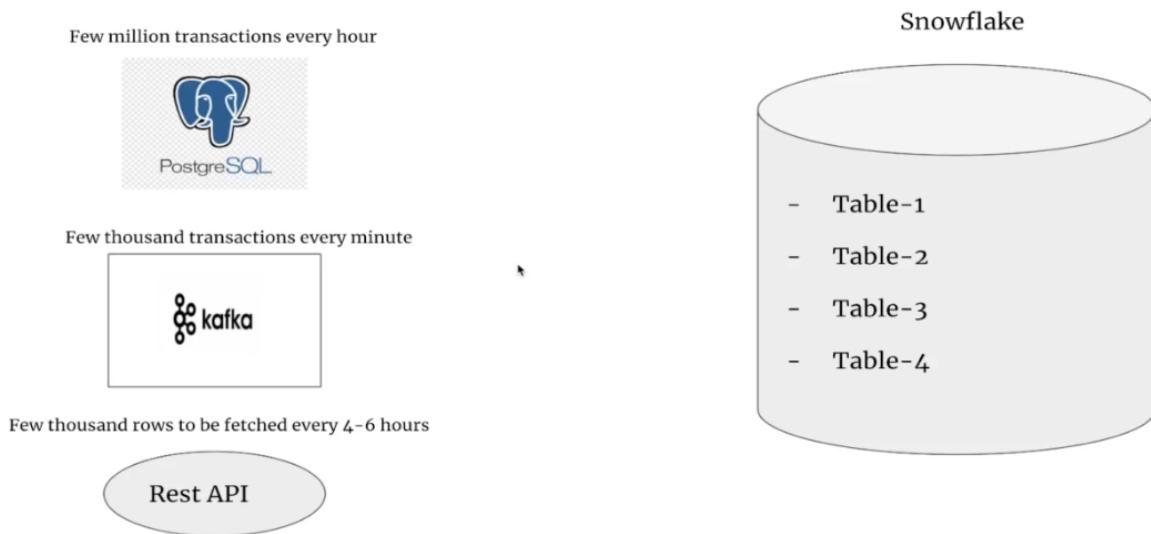
To removing Search Optimization service from the table.

```
ALTER TABLE IF EXISTS CUSTOMER DROP SEARCH OPTIMIZATION;
```

I will cover the remaining part of SOS service in my next blog. Stay Tuned...!

Hope this blog helps you to get insight into the **Search Optimization Service** feature. Feel free to ask a question in the comment section if you have any doubts regarding this. Give a clap if you like the blog. Stay connected to see many more such cool stuff. Thanks for your support.

Snowflake - Data Ingestion/Loading



Snowflake - Data loading options

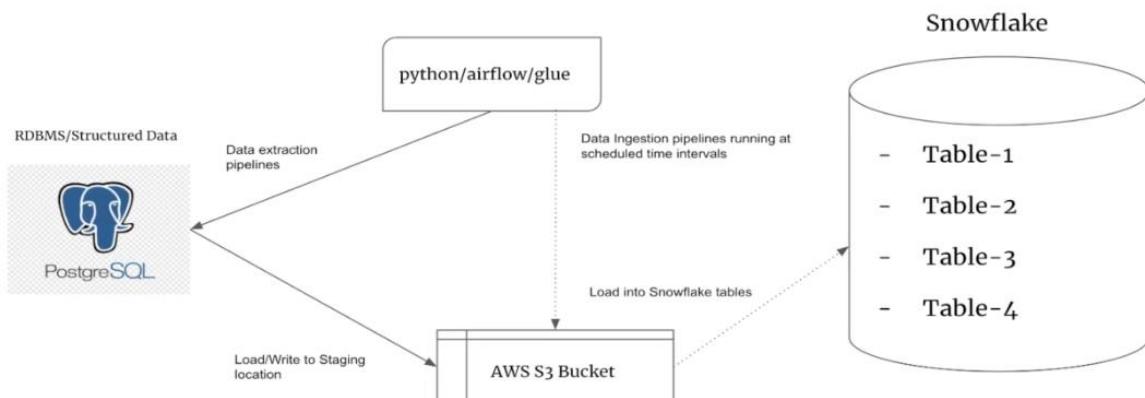
Batch Bulk Data Ingestion :

- Write/load the data into your staging location (S3 , GCS Buckets)
- Ingest the data into Snowflake in batches at frequent time intervals using :
 - Snowflake Copy commands scheduled using Snowflake tasks
 - Trigger Copy commands using python/Glue/Airflow running at specified time intervals

Real-time Data Ingestion :

- Write/load the data into your staging location (S3 , GCS Buckets) and ingest the data in near-real time using :
 - Snowpipe (Continuous data ingestion)
 - Airflow S3 sensors/triggers
- Kafka-snowflake Connector for real-time data ingestion

Snowflake - Batch Data Ingestion



you guys a high level overview of the different scenarios that we have when it comes to ingesting data. So in a typical set up, you have your transaction database or DBM as to be possible sequel, my sequel or LIB, you name it, which has a few million transactions that you need to ingest, let's say, every hour into Snowflake. And then you would have your streaming data, right, like Kafka Kinesis or any streaming application that gets the streaming data and needs this data needs to be ingested into Snowflake in real time. And finally, you have your rest APIs that you need to fetch or a ball every few us to extract the data out of it and write it in Snowflake.

Now this is as far as the source database is go or the source data goes. When it comes to ingesting data in the snowflake, you essentially split them into two types. The first one is batch, but then ingestion, which is your batch ingestion. Pipelines. Bad jobs. Typically you extract the data from your source and you write them into a staging location. The staging location is essentially a storage bucket, external storage bucket, since they're going to be dealing with AWB or working with the AWB.

In this case, it's going to be. It could also be your Google Cloud storage bucket. So as your storage. And once the data is written into your seeding storage bucket. You have your coffee commands running. Let's say your schedule, bobby garments using snowflake masks or you have your glue python or airflow pipelines that read the data every x hours, every opportunity to us read this data and write it in a snowflake. Now that's a typical batch data ingestion pipeline. When it comes to real time data ingestion, you know that there are two ways of going about it. You write the data and your staging location, let's say S3 bucket. And as soon as the data is written into your storage location, you can have a snowball logic that gets triggered as soon as the data is written and a lot of magically write the data into your snowflake table. Or you can have. S3 census. You know, using airflow or you can have some sort of a trigger for it, abuse, glue or any job that you have written that leaves the data and writes it in a snowflake as soon as the data arrives.

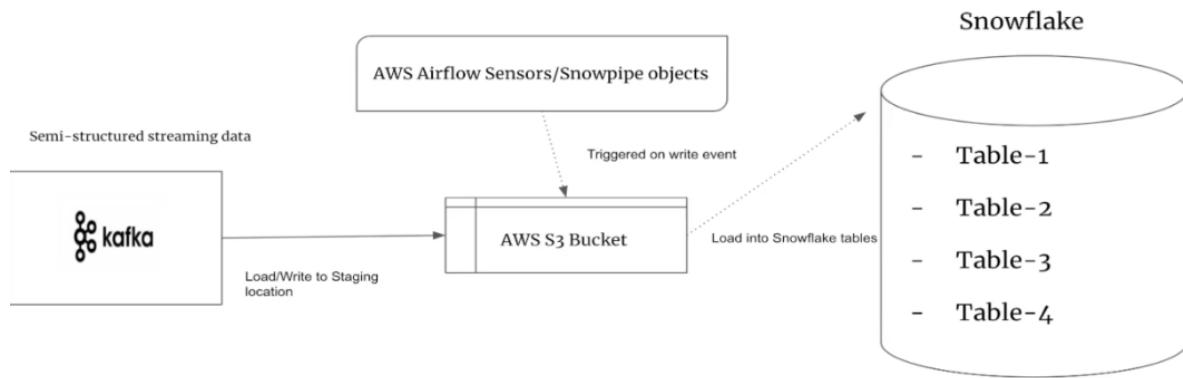
And finally, the real time data ingestion, especially between Kafka and Snowflake, is can also be done using a connector. So all you have to do is set up a connector for Kafka in Snowflake and it'll automatically handle your data, which is coming in real time. So this is these are the two ways and we'll be looking at each one of this in detail in the following lectures.

Structured IBM as data transactional leader in both sequel. Not typically how you do is you will have your data extraction by blanks using by the lambda glue airflow Then extracted the data and writes it into your bucket. And then once the data arrives into your bucket, you would probably have your glue jobs or airflow jobs which are running every X.

I was like I mentioned that schedule, Diamond Dallas that read this data and write it in Snowflake. When it comes to their land ingestion, there are two ways of going about it, essentially.

One, the streaming data from Kafka is written in the city bucket or staging location. As soon as the data arrives, you can have your snow five objects or air flow sensors that write the data in near real time in Snowflake. Finally You have your Kafka snowflake connector that handles everything for you.

Snowflake - Real-Time Data Ingestion



Snowflake - Real-Time Data Ingestion



Snowflake is now capable of near real-time data ingestion, data integration, and data queries at an incredible scale. This article explains how Snowflake uses Kafka to deliver real-time data capture, with results available on Tableau dashboards within minutes.

It summarises the challenges faced, the components needed and why the traditional approach (the Lambda Architecture) is no longer a sensible strategy to deliver real-time data queries at scale.

Real-Time Data Requirements

The requirements include the ability to capture, transform and analyze data at a potentially massive velocity in near real-time. This involves capturing data from millions of electronic sensors and transforming and storing the results for real-time analysis on dashboards. The solution must minimize latency – the delay between a real-world event and its impact upon dashboards, to a few minutes.



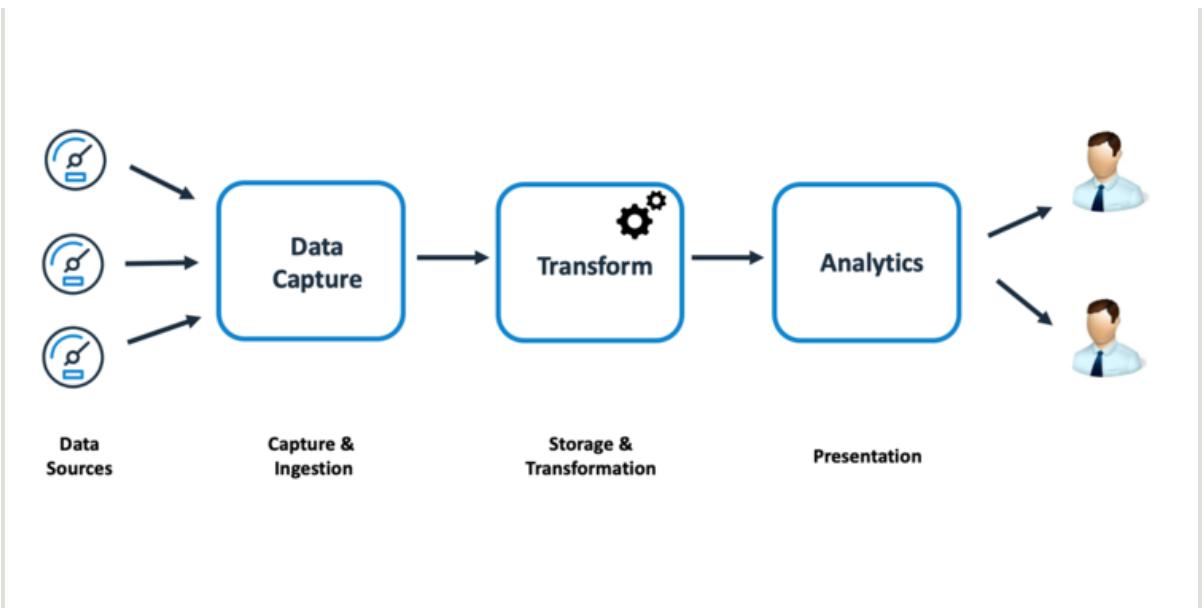
Typical applications include: -

- **Monitoring Machine Sensors:** Using embedded sensors in industrial machines or vehicles. For example, Progressive Insurance uses real-time speed data to help analyze customer behavior and deliver appropriate discounts. Similar technology is used by logistics giant FedEx which uses SenseAware to provide near real-time parcel tracking.
- **Fraud Detection:** To assess the risk of credit card fraud before authorizing or declining the transaction. This can be based upon a simple report of a lost or stolen card, or more likely, an analysis of aggregate spending behavior, aligned with machine learning techniques.
- **Customer Sentiment Analysis:** Used by many retail operations, this involves the capture of and analysis of social media feeds including Twitter and Facebook. In addition to the velocity challenge, the data is provided in JSON format where the structure is likely to change over time.

What's the Problem?

The primary challenge for systems architects is the potentially massive throughput required which could exceed a million transactions per second. NoSQL databases can handle the data velocity but have the disadvantages associated with a lack of SQL access, no transaction support, and eventual consistency. Finally, they don't support flexible join operations, and analytic query options are limited or non-existent. This means you can quickly retrieve a key-value pair for an event, but analyzing the data is a severe challenge.

However it doesn't stop there.



Real-Time Data Pipeline

Real-time Components

The diagram above illustrates the main architectural components needed to solve this problem. This includes:

Data Capture

- **High-Velocity Data Capture:** The ability to capture high-velocity message streams from multiple data sources in the range of hundreds of megabytes per second.
- **Message Queuing:** We can expect short term spikes in data volume which implies a message handling solution to avoid the need to scale up the entire solution for the worst possible case.
- **Guaranteed message delivery:** Which implies a scale-out, fault-tolerant, highly available solution that gracefully handles individual node failure, and guarantees message delivery.
- **Architectural Separation:** To decouple the source systems from the messaging, transformation and data storage components. Ideally, the solution should allow independent scaling of each component in the stack.

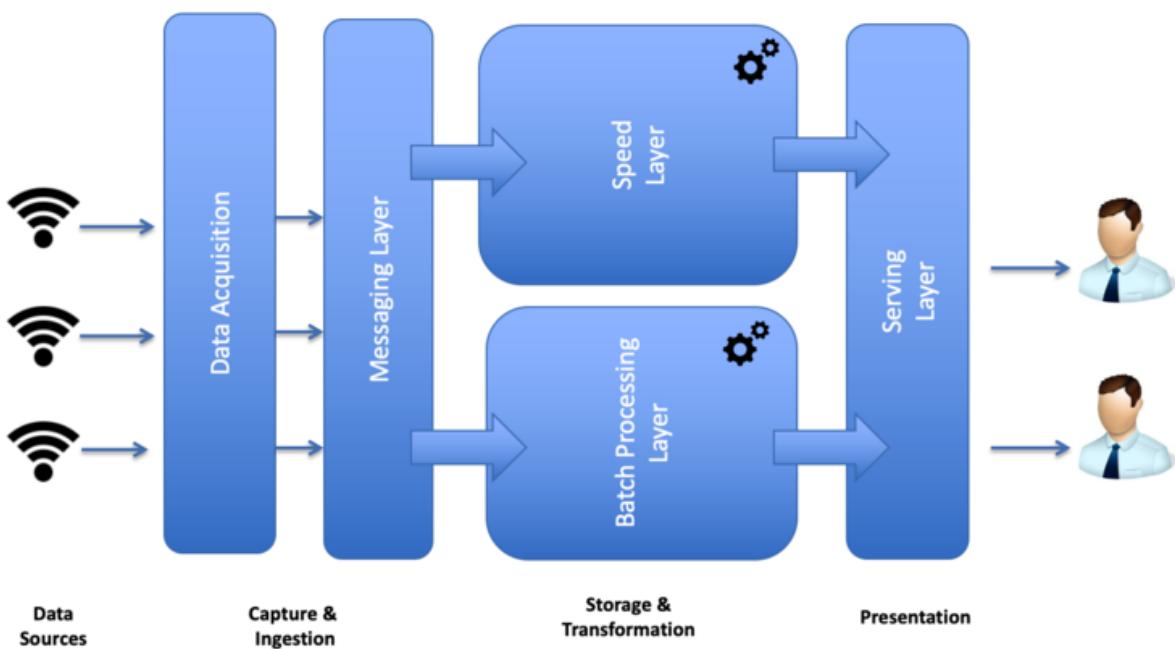
Transformation

- **Data integration:** The transformation process will almost certainly need to combine real-time transaction streams with existing reference data from databases and other data sources. The solution must, therefore, provide excellent data source connectivity and seamless integration with other sources.
- **Guaranteed once only processing:** The transformation process needs to be resilient to failure and re-start operations, effectively guaranteeing every message will be processed once only.

- **Massively Scalable:** While the data capture component will help smooth out massive spikes in velocity, the solution must transparently scale to deal with both regular and unexpected workloads.

Storage and Analytics

- **Unlimited Data Storage:** The data storage solution must be capable of accepting, processing and storing millions of transactions, ideally in a single repository. This implies an almost unlimited data storage capacity, combining both Data Lake and analytics capability on a single platform.
- **Dashboard Connectivity:** The solution must provide support for open connectivity standards including JDBC and ODBC to support Business Intelligence and dashboards. There's little value in capturing the data if it cannot be analyzed.



Classic Lambda Architecture

The Traditional Solution

The diagram above illustrates a common architecture referred to as the Lambda Architecture which includes a Speed Layer to process data in real-time with a Batch Layer to produce an accurate historical record. In essence, this splits the problem into two distinct components, and the results are combined at query time in the Serving Layer to deliver results.

While the Lambda Architecture has many advantages including decoupling and separation of responsibility, it also has the following disadvantages:

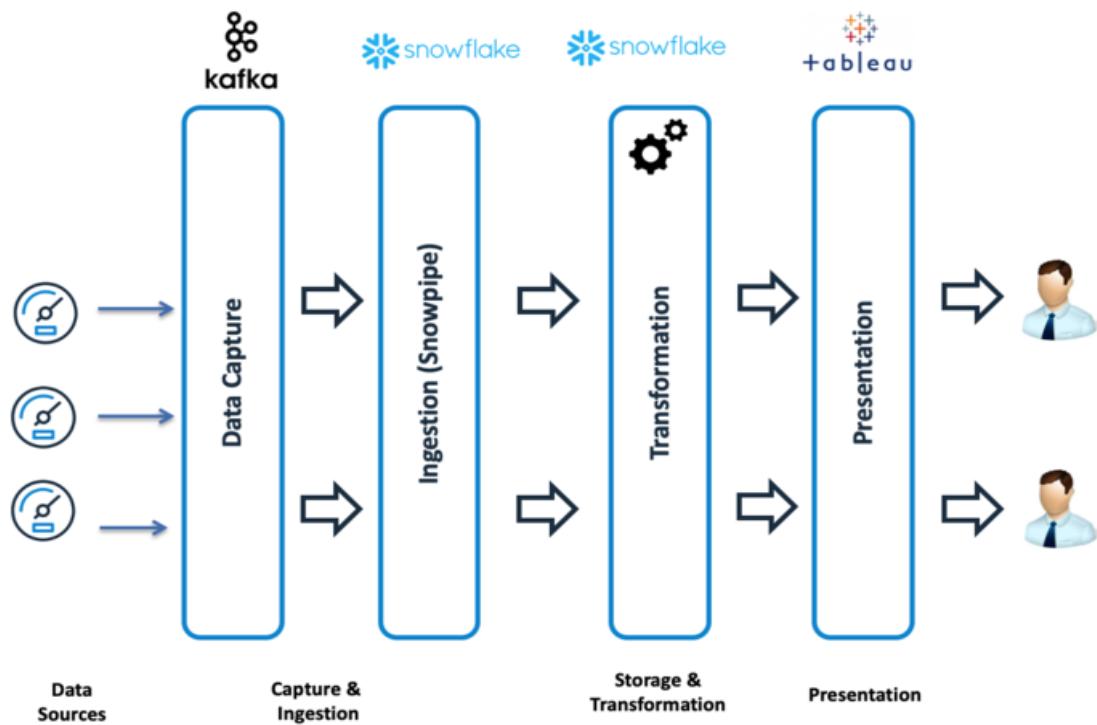
- **Logic Duplication:** Much of the logic to transform the data is duplicated in both the Speed and Batch layers. This adds to the system complexity and creates challenges for maintenance as code needs to be maintained in two places – often using two

completely different technologies. As Jay Kreps who invented the Lambda Architecture while at LinkedIn testifies, keeping code written in two different systems was really hard.

- **Batch Processing Effort:** The batch processing layer assumes all input data is re-processed every time. This has the advantage of guaranteeing accuracy as code changes are applied to the data every time but place a huge batch processing burden on the system.
- **Serving Layer Complexity:** As data is independently processed by the Batch and Speed layers, the Serving Layer must execute queries against two data sources and combine real-time and historical results into a single query. This adds additional complexity to the solution and may rule out direct access from some dashboard tools or need additional development effort.
- **NoSQL Data Storage:** While batch processing typically uses Hadoop/HDFS for data storage, the Speed Layer needs fast random access to data, and typically uses a NoSQL database, for example, HBase. This comes with huge disadvantages including no industry standard SQL interface, a lack of join operations, and no support for ad-hoc analytic queries.

When the only transformation tool available was Map-Reduce with NoSQL for data storage, the Lambda Architecture was a sensible solution, and it has been successfully deployed at scale at Twitter and LinkedIn. However, there are more advanced (and simple) alternatives available.

The Snowflake Solution



Snowflake Real-Time Data Query Architecture

The diagram above illustrates an alternative simple solution with a single real-time data flow from source to dashboard. The critical component that makes this possible is the Snowflake data warehouse which now includes a native [Kafka connector](#) in addition to [Streams and Tasks](#) to seamlessly capture, transform and analyze data in near real-time.

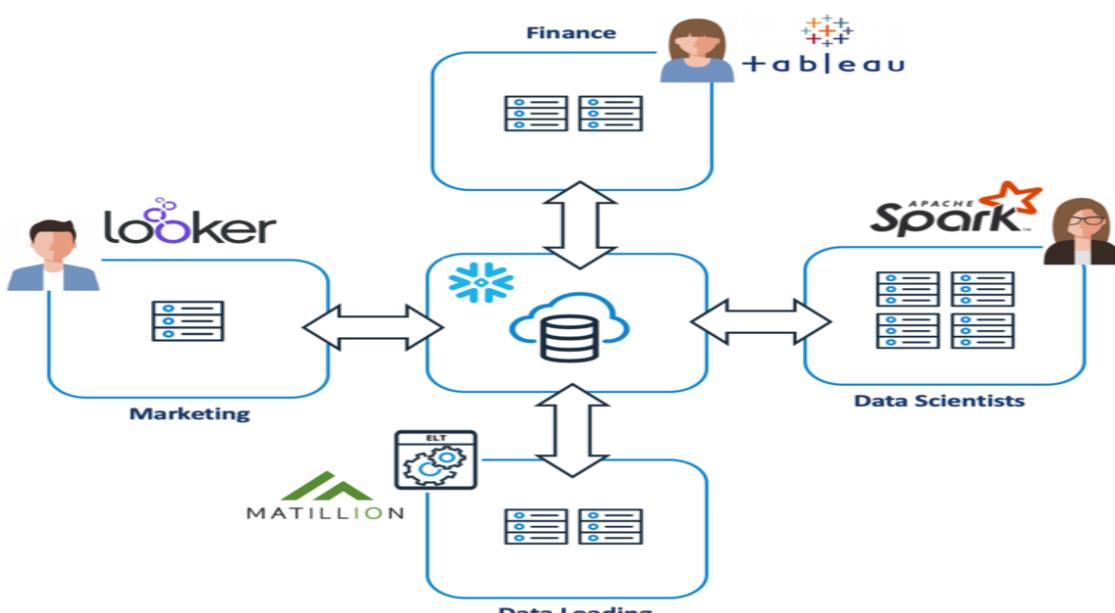
The components in the above solution are:

- **Apache Kafka:** For fault-tolerant message queuing and broadcast system.
- **Snowflake Streams & Tasks:** To receive the data, perform change data capture and transform and store data ready for analysis and presentation.
- **Snowflake Multi-cluster Architecture:** To seamlessly handle thousands of online concurrent users to analyze results.
- **Tableau:** For analytic presentation and dashboards.

The advantages of this architecture include:

- **Absolute Simplicity:** As a pipeline to capture, implement change data capture, and storage can be completed with just a handful of SQL statements.
- **SQL Transformations:** With all data transformation logic in the Snowflake transformation component (using industry-standard SQL), there's no code duplication or multiple technologies to cause maintenance issues.
- **Real-Time Accuracy:** As the database solution provides full relational support and ACID compliance, there is no issue around eventual consistency from NoSQL solutions.

The diagram below illustrates one of the unique benefits of Snowflake which delivers unlimited cloud-based compute resources as Virtual Warehouses over a shared storage system. This means, it's no longer necessary to provide a separate speed and batch processing layer, as queries can be continually streamed into the warehouse using Snowpipe, while being transformed on one virtual warehouse, and results analyzed on yet another.



Snowflake Virtual Warehouse Architecture

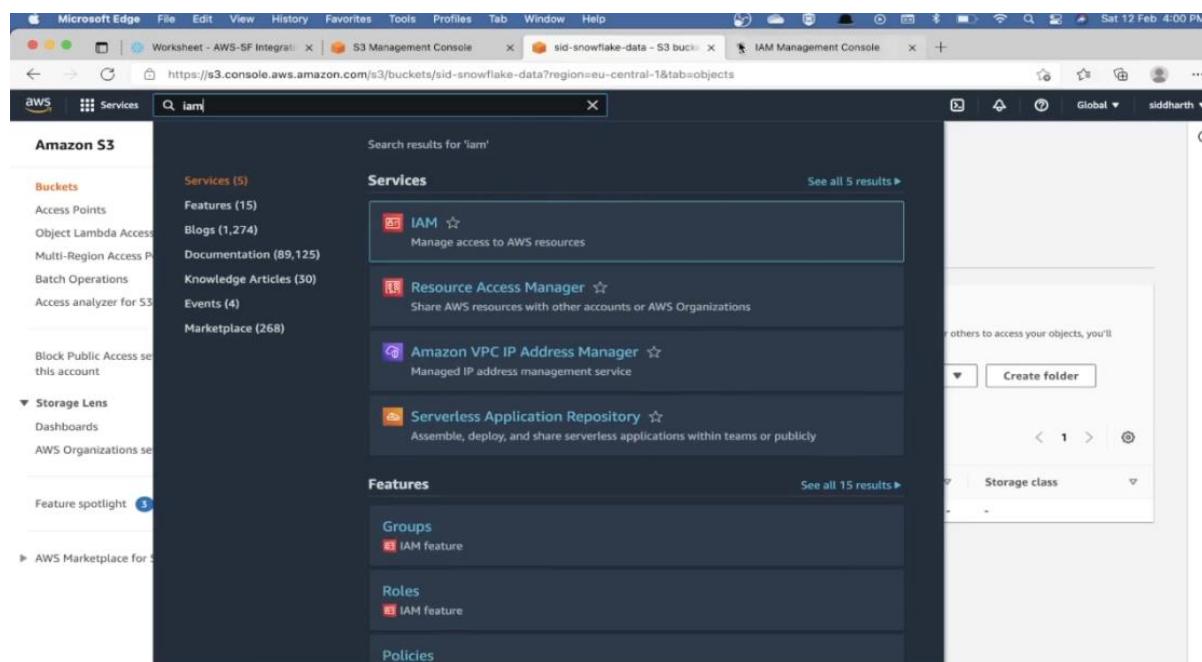
Conclusion

Snowflake is uniquely positioned to provide a single platform for all your data warehouse storage, processing, and analysis needs. This includes:

- **Near Real-Time Streaming:** Using Snowpipe and the native Kafka connector to capture and ingest streaming data without contention.
- **Semi-structured Processing:** Which uses a simple SQL interface to provide a real-time schema-on-read view over JSON, AVRO, Parquet and XML. Unlike other databases that store semi-structured data as simple text, Snowflake parses the data and seamlessly stores it in a columnar data structure for fast retrieval.
- **A Data Lake:** Which can combine both semi-structured JSON and structured CSV formats. This helps abolish the separate data silos of Data Lake and Warehouse.
- **A Data Warehouse:** With full ACID compliance, and an industry-standard SQL interface over an unlimited data store.
- **Integration to Transformation and Analytic tools:** Including native connectors for Spark, Talend, Informatica, Looker, PowerBI and Tableau.

Snowflake - AWS Connection

- Step-1 : Create an IAM Role for Snowflake to access data in S3 Buckets
- Step-2 : Create S3 Bucket in AWS and upload Sample Files into the bucket
- Step-3 : Create an Integration Object in Snowflake for authentication (using Accountadmin Role)
- Step-4 : Create a File Format Object (Using Sysadmin/Custom Role)
- Step-5: Create a stage object referencing the location from which the data needs to be ingested (Using Sysadmin/Custom Role)
- Step-5 : Load the data into Snowflake Tables (Using Sysadmin/Custom Role)



Screenshot of the AWS IAM Management Console showing the 'Create role' wizard Step 2: Set permissions boundary.

Options:

- SAML 2.0 federation
- Custom trust policy

An AWS account

Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

This account (077056421570)

Another AWS account

External ID

00000

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

Require MFA

Requires that the assuming entity use multi-factor authentication.

Cancel Next

Screenshot of the AWS IAM Management Console showing the 'Create role' wizard Step 2: Add permissions.

Add permissions

Permissions policies (726)

Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter

Clear filters

Policy name	Type	Description
AmazonDMSRedshiftS3Role	AWS m...	Provides access to manage S3 settings for Redshift endpoints for...
AmazonS3FullAccess	AWS m...	Provides full access to all buckets via the AWS Management Con...
QuickSightAccessForS3StorageManagem...	AWS m...	Policy used by QuickSight team to access customer data produce...
AmazonS3ReadOnlyAccess	AWS m...	Provides read only access to all buckets via the AWS Managemen...
AmazonS3OutpostsFullAccess	AWS m...	Provides full access to Amazon S3 on Outposts via the AWS Man...
AmazonS3ObjectLambdaExecutionRoleP...	AWS m...	Provides AWS Lambda functions permissions to interact with Am...
AmazonS3OutpostsReadOnlyAccess	AWS m...	Provides read only access to Amazon S3 on Outposts via the AW...

Set permissions boundary - optional

Set a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.

Sat 12 Feb 4:01 PM

Worksheet - AWS-SF Integral | S3 Management Console | sid-snowflake-data - S3 buck | IAM Management Console | +

https://console.aws.amazon.com/iamv2/home#/roles/create?externalId=00000&isThirdParty=true&step=review&trustedEntityType=AWS_ACCOUNT

aws Services Search for services, features, blogs, docs, and more [Option+S]

Step 1 Select trusted entity

Step 2 Add permissions

Step 3 Name, review, and create

Name, review, and create

Role details

Role name Enter a meaningful name to identify this role.
snowflake-aws-role Maximum 128 characters. Use alphanumeric and '+-, @-' characters.

Description Add a short explanation for this policy.
Access for snowflake Maximum 1000 characters. Use alphanumeric and '+-, @-' characters.

Step 1: Select trusted entities

Edit

```
1- [
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Effect": "Allow",
6-       "Action": "sts:AssumeRole",
7-       "Principal": {
8-         "AWS": "077056421570"
9-       },
10-      "Condition": {
11-        "StringEquals": {
```

Sat 12 Feb 4:04 PM

Worksheet - AWS-SF Integral | S3 Management Console | sid-snowflake-data - S3 buck | IAM Management Console | sid-snowflake-data - S3 buck | +

https://ua52590.eu-central-1.snowflakecomputing.com/console#/internal/worksheet

Enjoy your free trial! Visit our documentation to learn more about using Snowflake or contact our support team with any questions.

New Worksheet Search Optimization Tasks AWS-SF Integration Load CSV from S3

Databases Shares Data Marketplace Warehouses Worksheets History Account Partner Connect Help Notifications Snowsight SIDDHARTH ACCOUNTADMIN

Find database objects Starting with... Run All Queries Saved 5 minutes ago

ECOMMERCE_DB

- ECOMMERCE_LIV
 - Tables CUSTOMER LINEITEM NATION ORDERS PART PARTSUPP REGION SUPPLIER
- No Views in this Schema

INFORMATION_SCHEMA

KAFKA_LIV_SCHEMA PUBLIC SNOWFLAKE SNOWFLAKE_SAMPLE_DATA

```
1 use role accountadmin;
2 
3 
4 CREATE OR REPLACE STORAGE INTEGRATION aws_sf_data
5 TYPE = EXTERNAL_STAGE
6 STORAGE_PROVIDER = S3
7 ENABLED = TRUE
8 STORAGE_AWS_ROLE_ARN =
9 STORAGE_ALLOWED_LOCATIONS = ('s3://sid-snowflake-data/');
10 
11 desc INTEGRATION aws_sf_data;
12 
13 use role sysadmin;
14 
15 create schema ecommerce_dev;
```

Results Data Preview Open History

Query_ID SQL 379ms 1 rows

Filter result... Copy

Row	status
1	Statement executed successfully.

Columns

Screenshot of the AWS IAM Management Console showing the creation of a new role named "snowflake-aws-role".

Role Details:

- ARN:** arn:aws:iam::077056421570:role/snowflake-aws-role
- Creation date:** February 12, 2022, 16:01 (UTC+04:00)
- Last activity:** None
- Maximum session duration:** 1 hour

Permissions: A single policy named "AmazonS3FullAccess" is attached.

Policy name	Type	Description
AmazonS3FullAccess	AWS managed	Provides full access to all buckets via the AWS Management Console.

Screenshot of the Snowflake Worksheet interface showing the configuration of an AWS integration.

SQL Query:

```

use role accountadmin;
CREATE or replace STORAGE INTEGRATION aws_sf_data
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = S3
ENABLED = TRUE
STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::077056421570:role/snowflake-aws-role'
    
```

Results - Data Preview:

Row	property	property_type	property_value	property_default
1	ENABLED	Boolean	true	false
2	STORAGE_PROVIDER	String	S3	
3	STORAGE_ALLOWED_LOCATIONS	List	s3://sid-snowflake-data/	
4	STORAGE_BLOCKED_LOCATIONS	List		
5	STORAGE_AWS_IAM_USER_ARN	String	arn:aws:iam::206954943453:user/2f40-s-eusb6631	
6	STORAGE_AWS_ROLE_ARN	String	arn:aws:iam::077056421570:role/snowflake-aws-role	
7	STORAGE_AWS_EXTERNAL_ID	String	UA52590_SFRole=3_pmlqqTjE0TuUr8O6jBEMV054=	
8	COMMENT	String		

Sat 12 Feb 4:06 PM

https://console.aws.amazon.com/iamv2/home#/roles/details/snowflake-aws-role?section=trust_relationships

Identity and Access Management (IAM)

Last activity: None

Maximum session duration: 1 hour

Permissions | Trust relationships | Tags | Access Advisor | Revoke sessions

Trusted entities

Entities that can assume this role under specified conditions.

Edit trust policy

```
1 + [
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Principal": {
7                 "AWS": "arn:aws:iam::077056421570:root"
8             },
9             "Action": "sts:AssumeRole",
10            "Condition": {
11                "StringEquals": {
12                    "sts:ExternalId": "00000"
13                }
14            }
15        }
16    ]
17 ]
```

Sat 12 Feb 4:07 PM

https://console.aws.amazon.com/iamv2/home#/roles/details/snowflake-aws-role/edit-trust-policy

IAM > Roles > snowflake-aws-role > Edit trust policy

Edit trust policy

```
1 + [
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Principal": {
7                 "AWS": "arn:aws:iam::206954943453:user/20-s-eusb6631"
8             },
9             "Action": "sts:AssumeRole",
10            "Condition": {
11                "StringEquals": {
12                    "sts:ExternalId": "UAS290_SFRole=3_pmlqqtTjE0TuUr806j/j8EMV054="
13                }
14            }
15        }
16    ]
17 ]
```

Add new statement

[Snowflake](#) Data Cloud is an advanced data platform provided as Software-as-a-Service (SaaS). It includes many data storage, processing, and analytic solutions that are faster, easier to use, and more flexible than traditional offerings.



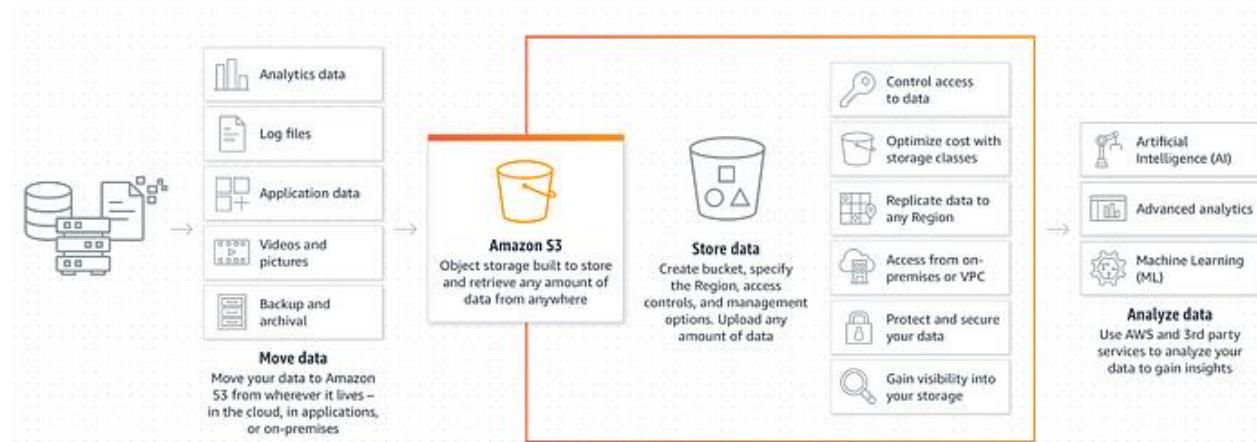
As shown in the previous image, this platform has a big offer with different ways and tools to do an environment collaborative between business data areas such as Engineering, Cybersecurity, Data Science & ML, and, Backend/Applications.

To create and maintain the environment, using data as input to generate outcomes such as insights, predictions, monetization reports, and data products allows the generation of a continuous flow between areas and processes.

Amazon Simple Storage Service (S3 Bucket)

[Amazon Simple Storage Service](#) (Amazon S3) is an object storage service for storing and protecting many amounts of data for various use cases. An S3 bucket can store data in any format (records, images, videos, and complex data) in different sizes.

This storage is an excellent control to access the data and optimize costs while adding various features such as region replication, VPC, protection, etc.



In addition to the information mentioned as an introduction to the exercise, objectives, and conceptual terms to be applied and learned, the reader needs the following steps and pieces of knowledge to understand and apply this task:

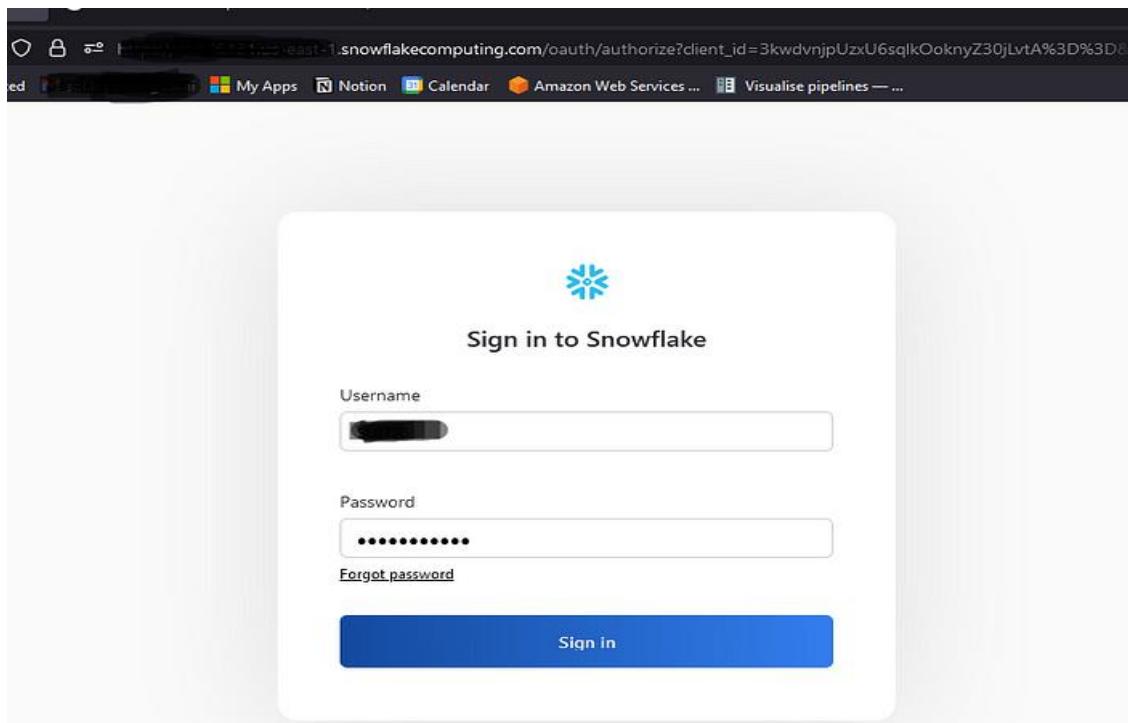
Pre-requirements

1. The exercise needs a **Snowflake Account** to work on the sample data. If the reader has not an account, Snowflake offers a way to [create a trial account](#).
2. The reader needs initial knowledge about these concepts in Snowflake or any other database/data warehouse: [ROLES](#), [WAREHOUSE](#), [STAGE](#), [COPY INTO](#), and [FILE FORMAT](#).
3. The Snowflake account needs permission to execute SYSADMIN and ACCOUNTADMIN roles.

After understanding and complying with these prerequisites, you can start developing the exercise.

1- Access to the Snowflake console

In this step, we enter the Snowflake console we have at our disposal, or the one built as indicated in **Step 1 of the Prerequisites**. This is intended to give users access to the main console so that the exercise can begin.



Snowflake Console.

Then, we can open a worksheet:

Open a new worksheet.

And we can find the possibility to create a new one in any part into a folder or not:

The screenshot shows a database interface with a sidebar on the left containing 'Worksheets' and 'Databases' tabs. A search bar and a '+' button are at the top. A context menu is open over a folder named 'Working with CSV/JSON files'. The menu options are 'New Worksheet' and 'New Folder', both highlighted with a black rectangle. To the right, there is a list of rows with numbers 100 to 114 and labels CIBA.F, SELE, ALTE, and TRUM.

Create a new worksheet or folder.

And as a result, we have our worksheet ready to continue with the code part:

The screenshot shows the same database interface after creating a new worksheet. The 'Working with CSV files' folder is now highlighted with a blue selection bar. The right panel displays a table with columns for row numbers (1 to 5) and statements (USE R and CREAT).

1	USE R
2	
3	
4	
5	CREAT

The new worksheet to save our statements.

2- Creation of *CIBA* database

In this step, it is necessary to create a database to save our information in the following steps. For this, the process begins with a role equal to **SYSADMIN** for database creation (*this role is designated to create the component into the database without admin permission*):

```
USE ROLE SYSADMIN;

CREATE OR REPLACE DATABASE CIBA COMMENT = 'Test database for CIBA company';
```

After the execution, the database is created and ready to use for the **SYSADMIN** role

The screenshot shows a Snowflake workspace interface. On the left, there's a sidebar with 'Worksheets' and 'Databases'. Under 'Databases', there's a list with 'CIBA' highlighted and a red box around it. Other items in the list include 'INFORMATION_SCHEMA', 'PUBLIC', and 'SNOWFLAKE'. The main area shows a query editor with the following code:

```
1 USE ROLE SYSADMIN;
2
3 CREATE OR REPLACE DATABASE CIBA COMMENT = 'Test database for CIBA company';
```

Below the code, there's a results pane showing:

status
1 Database CIBA successfully created.

On the right side of the interface, there are tabs for 'Objects', 'Editor', 'Results', and 'Chart'. The 'Results' tab is selected. At the top right, there are user and role indicators ('SYSADMIN - COMPUTE_WH'), a 'Share' button, and a timestamp ('Updated 29 seconds ago').

Database created with SYSADMIN role

3- Adjustments in the *COMPUTE_WH* warehouse configuration

In this part, the assignment of an intermediate configuration for the data warehouse is necessary to check the performance of the queries with a configuration that generates little cost.

For that, the **ACCOUNTADMIN** role grants permission to **SYSADMIN** to work on the next steps with the warehouse configuration.

Also, the warehouse should be sized MEDIUM with the largest cluster number of 2 to confirm the data loading/downloading performance.

```
USE ROLE ACCOUNTADMIN;

GRANT USAGE ON WAREHOUSE COMPUTE_WH TO ROLE SYSADMIN;

CREATE OR REPLACE WAREHOUSE COMPUTE_WH
WITH WAREHOUSE_SIZE = 'MEDIUM'
WAREHOUSE_TYPE = 'STANDARD'
AUTO_SUSPEND = 60
AUTO_RESUME = TRUE
MIN_CLUSTER_COUNT = 1
MAX_CLUSTER_COUNT = 2
SCALING_POLICY = 'STANDARD';
```

After the execution, all changes were done in our warehouse, called COMPUTE_WH

4- Creation of a *trips* table

In this step, and with the previous overview about the context of the data in the files, we can find more details about the data type needed to fill this table as the trip duration with their start and final time; the general information about the initial and final station of the trip; the general information about the bike used, the user and details about the membership.

A DDL was created to work in this exercise for the *trips* table.

Note: The use of a general role, a warehouse, a database, and a schema is necessary to generate the table

```
USE ROLE SYSADMIN;
USE WAREHOUSE COMPUTE_WH;
USE DATABASE CIBA;
USE SCHEMA PUBLIC;

CREATE OR REPLACE TABLE TRIPS
(TRIPDURATION INTEGER,
STARTTIME TIMESTAMP,
```

```

STOPTIME TIMESTAMP,
START_STATION_ID INTEGER,
START_STATION_NAME STRING,
START_STATION_LATITUDE FLOAT,
START_STATION_LONGITUDE FLOAT,
END_STATION_ID INTEGER,
END_STATION_NAME STRING,
END_STATION_LATITUDE FLOAT,
END_STATION_LONGITUDE FLOAT,
BIKEID INTEGER,
MEMBERSHIP_TYPE STRING,
USERTYPE STRING,
BIRTH_YEAR INTEGER,
GENDER INTEGER);

```

After the execution, the table called trips was created into CIBA.PUBLIC database/schema

The screenshot shows a data warehouse interface with the following details:

- Header:** Working with CSV/JSON files - Working with CSV files - CIBA.PUBLIC
- Role:** SYSADMIN + COMPUTE_WH
- Timestamp:** Updated 34 seconds ago
- Navigation:** Worksheets (Pinned: 0) and Databases (CIBA.PUBLIC)
- Databases:** CIBA.PUBLIC (selected)
- Code Editor:** Contains the SQL code for creating the TRIPS table.
- Table View:** Shows the TRIPS table with columns: TRIPDURATION, STARTTIME, STOPTIME, and START_STATION_ID.
- Status Bar:** status: Table TRIPS successfully created.
- Query Details:** Query duration: 141ms, Rows: 1

TRIPS table created into CIBA as database and PUBLIC as schema.

5- Creation of *CIBA_TRIPS* stage

With the database and table created in the above steps, this step is fundamental to the stage creation to continue the loading process using the S3 bucket designated. The sentences to do this goal are the following:

```
CREATE STAGE "CIBA"."PUBLIC".CIBA_TRIPS
URL = 's3://snowflake-workshop-lab/citibike-trips-csv'
COMMENT = 'EXTERNAL STAGE FOR CIBA LOADING PROCESS';
```

In detail, the bucket used in this process and maintained by Snowflake during a lecture held earlier during a lab. This bucket has many types of data samples to work with them.

After the execution, our stage was created in our database.

The screenshot shows the Snowflake UI interface. At the top, there is a code editor window with the following SQL code:

```
42
43 CREATE STAGE "CIBA"."PUBLIC".CIBA_TRIPS |
44   URL = 's3://snowflake-workshop-lab/citibike-trips-csv'
45   COMMENT = 'EXTERNAL STAGE FOR CIBA LOADING PROCESS';
46
```

Below the code editor are three tabs: **Objects**, **Editor** (which is selected), and **Results**. The **Results** tab displays a table with one row:

	status
1	Stage area CIBA_TRIPS successfully created.

CIBA_TRIPS stage created.

Then, only to confirm the correct connection, it is a good practice the execution of a list function to see all files in the bucket:

```
LIST @CIBA_TRIPS;
```

```

46
47 LIST @CIBA_TRIPS;
48
49 SELECT FLOOR(SUM($2)/POWER(1024, 3),1) TOTAL_COMPRESSED_STORAGE_GB,
50   FLOOR(AVG($2)/POWER(1024, 2),1) AVG_FILE_SIZE_MB,
```

Objects Editor Results Chart

	name	size	md5
1	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_0.csv.gz	3,072,073	cfc69e04228a94d13
2	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_1.csv.gz	2,877,852	92a1c064a3c632f33
3	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_2.csv.gz	3,174,598	39faac098802c1b29
4	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_3.csv.gz	3,031,012	cd0dca1dcfa309c0bt
5	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_4.csv.gz	3,005,838	fb24c0cc5fb6ee54d
6	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_5.csv.gz	3,099,881	441efe806352c57a5l
7	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_6.csv.gz	3,060,952	372765a1ca713eeb1c
8	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_0_7.csv.gz	3,302,808	0298668f685d5e6f4

Query Details

Query duration 700ms

Rows 376

name Aa
100% filled

size 123

List of files in the stage created.

A good idea, too, to test the amount and sizes in all files generated in the last execution using the following query:

```

SELECT FLOOR(SUM($2)/POWER(1024, 3),1) TOTAL_COMPRESSED_STORAGE_GB,
       FLOOR(AVG($2)/POWER(1024, 2),1) AVG_FILE_SIZE_MB,
       COUNT(*) AS NUM_FILES
  FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

After this, we can detect the total storage consumed by the files in GB, an average between the file sizes and the total amount of files

```

9   SELECT FLOOR(SUM($2)/POWER(1024, 3),1) TOTAL_COMPRESSED_STORAGE_GB,
10  FLOOR(AVG($2)/POWER(1024, 2),1) AVG_FILE_SIZE_MB,
11  COUNT(*) AS NUM_FILES
12  FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

Objects Editor Results Chart

TOTAL_COMPRESSED_STORAGE_GB	AVG_FILE_SIZE_MB	...	NUM_FILES
1.8	5.1		376

Query Details

Query duration 757ms

Rows 1

TOTAL_COMPRESSED_STORAGE_GB 123
100% filled

AVG_FILE_SIZE_MB 123

Details in sizes and amount of files in the STAGE created.

6- Creation of CSV_FORMAT file format

The CSV reading process is primordial to create a file format object in Snowflake because this element is a priority to read any complex data type from any kind of file to save all in Snowflake.

But, we will apply as a good practice the creation of a new database to manage these types of components since thinking in the future, from a database it is easier and more controlling to share the access between users/database on these components. Thus, the lines to execute are the following ones:

```
CREATE OR REPLACE SCHEMA FILE_FORMAT;  
USE SCHEMA FILE_FORMAT;
```

And finally, the file format to read a simple CSV file is the following:

```
CREATE FILE FORMAT "CIBA"."FILE_FORMAT".CSV_FORMAT  
TYPE = 'CSV' COMPRESSION = 'AUTO'  
FIELD_DELIMITER = ',',  
RECORD_DELIMITER = '\n'  
SKIP_HEADER = 0  
FIELD_OPTIONALLY_ENCLOSED_BY = '\042'  
TRIM_SPACE = FALSE  
ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE  
ESCAPE = 'NONE'  
ESCAPE_UNENCLOSED_FIELD = '\134'  
DATE_FORMAT = 'AUTO'  
TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('')  
COMMENT = 'FILE FORMAT TO LOAD CSV - CIBA';
```

After this, our CSV_FORMAT was created in our FILE_FORMAT database:

The screenshot shows the Snowflake interface. On the left, under 'Databases', the 'CIBA' database is selected. Inside 'CIBA', the 'FILE_FORMAT' object is highlighted with a red box. The code editor on the right contains the following SQL script:

```
CREATE OR REPLACE FILE FORMAT CIBA.FILE_FORMAT
  FIELD_DELIMITER = ','
  RECORD_DELIMITER = '\n'
  SKIP_HEADER = 0
  FIELD_OPTIONALLY_ENCLOSED_BY = '\042'
  TRIM_SPACE = FALSE
  ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE
  ESCAPE = 'NONE'
  ESCAPE_UNENCLOSED_FIELD = '\134'
  DATE_FORMAT = 'AUTO'
  TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('')
  COMMENT = 'FILE FORMAT TO LOAD CSV - CIBA';
```

Below the code, there are tabs for 'Objects', 'Editor', 'Results', and 'Chart'. The 'Results' tab is selected, showing a single row:

	status
1	File format CSV_FORMAT successfully created.

CSV_FORMAT file format created in FILE_FORMAT schema.

This previous work in this exercise shows us the best way to generate the *STAGE* component and use it to connect with an S3 bucket containing many CSV files.

Also, as a recommendation, it is a good practice to create a new schema to handle FILE FORMAT and other structures generated in Snowflake other than tables and data.

So, we are ready in the practice part to learn about creating different kinds of elements in Snowflake and validating the files extracted from an S3 bucket.

Now, as the final step, we can start to check the best way to find a correct configuration with our warehouse, considering all the necessary economic aspects, and always fulfilling the business needs for the availability and agility of the data.

What needs to be evaluated next?

Let's imagine that, in a production case, the company needs to load many files from a table in Snowflake and optimize resources and costs during these tasks.

And according to the introduction about the steps to complete this exercise, we need to start evaluating with different configurations our response times with this data flow and upload it to the database, so, ***is this exercise a good approach to help us in a final decision on this part? Yes, it is.***

The best way to test this is to simulate a general load with different warehouse sizes during the load, validate if the configuration is the most appropriate for the need, and try several options until you find the best time/cost needed by the business and the rest of the process that consumes the final table and data.

Let's continue with the exercise

Now, with this new focus on this, the use of PUBLICschema is necessary to list all elements called through the stage created, and then, the following script is the owner to show us a file from the STAGE:

```
USE SCHEMA PUBLIC;

SELECT $1, $2, $3, $4, $5
FROM @CIBA_TRIPS/trips_2018_7_2_0.csv.gz
(FILE_FORMAT => "CIBA"."FILE_FORMAT".CSV_FORMAT )
LIMIT 100;
```

After the execution, we can show the different columns and information in the file called trips_2018_7_2_0.csv.gz

```

73
74   SELECT $1, $2, $3, $4, $5
75   FROM @CIBA_TRIPS/trips_2018_7_2_0.csv.gz
76   (FILE_FORMAT => "CIBA"."FILE_FORMAT".CSV_FORMAT )
77   LIMIT 100;
78

```

	\$1	\$2	\$3	\$4	\$5
1	2069	2018-06-19 17:29:33.717	2018-06-19 18:04:03.390	466	W 25 St & 6 Ave
2	1025	2018-06-19 17:29:33.902	2018-06-19 17:46:39.669	2008	Little West St & 1 Pl
3	1592	2018-06-19 17:29:34.034	2018-06-19 17:56:06.053	468	Broadway & W 56 St
4	1121	2018-06-19 17:29:34.561	2018-06-19 17:48:15.641	382	University Pl & E 14
5	1729	2018-06-19 17:29:35.975	2018-06-19 17:58:25.549	479	9 Ave & W 45 St
6	568	2018-06-19 17:29:36.833	2018-06-19 17:39:05.091	3435	Grand St & Elizabeth

Content in a file obtained from a STAGE.

Then, after this validation, we can execute a COPY INTO to save the information located on the S3 bucket through the STAGE to save the records in the *trips* table.

```

COPY INTO TRIPS
FROM @CIBA_TRIPS/trips
FILE_FORMAT="CIBA"."FILE_FORMAT".CSV_FORMAT;

```

After this, we can find all the files in the STAGE in the *trips* table.

```

80   COPY INTO TRIPS
81   FROM @CIBA_TRIPS/trips
82   FILE_FORMAT="CIBA"."FILE_FORMAT".CSV_FORMAT;
83

```

	file	status	rows_parsed
1	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_4_4_0.csv.gz	LOADED	104,107
2	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_6_3_0.csv.gz	LOADED	78,595
3	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2014_4_4_0.csv.gz	LOADED	162,174
4	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_3_1_0.csv.gz	LOADED	124,930
5	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_4_2_0.csv.gz	LOADED	141,532
6	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2016_3_6_0.csv.gz	LOADED	242,232
7	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2017_0_1_0.csv.gz	LOADED	260,914
8	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2017_3_2_0.csv.gz	LOADED	232,046

Files loaded in TRIPS table.

Then, we can confirm the total amount of the records in the tables with the following statement:

```
SELECT COUNT(*) FROM TRIPS;
```

84
85 | SELECT COUNT(*) FROM TRIPS;
86
87

Objects Editor Results Chart

	COUNT(*)
1	61,468,359

Query Details ...
Query duration 118ms
Rows 1
COUNT(*) 123
100% filled



Query details related to the COUNT statement.

We can see the time that Snowflake took to process these files and save them in a table with a COPY INTO (13 seconds) and during a COUNT(*) (0.118 seconds) to know the total amount of records in this table.

Is a good configuration for loading these files on cost? No, because Snowflake has a warehouse with a MEDIUMsize, and the loading data in this table must be a quick load.

So, we can test with the following configuration:

```
TRUNCATE TABLE TRIPS;  
  
ALTER WAREHOUSE "COMPUTE_WH"  
SET WAREHOUSE_SIZE = 'LARGE'  
AUTO_SUSPEND = 60  
AUTO_RESUME = TRUE;  
  
SELECT COUNT(*) FROM TRIPS;
```

In this case, the LARGE size in our data warehouse handles more memory to create more threads during the load. Also, an AUTO_SUSPEND equal to 60 handles the resource costs for this warehouse (*After 60 min without activity, the warehouse should be off*).

```

99
100    COPY INTO TRIPS
101      FROM @CIBA_TRIPS/trips
102      FILE_FORMAT="CIBA"."FILE_FORMAT".CSV_FORMAT;
103

```

Objects Editor Results Chart

	file	status	rows_parsed
1	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_6_0.csv.gz	LOADED	115,615
2	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2015_2_1_0.csv.gz	LOADED	214,714
3	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2016_0_5_0.csv.gz	LOADED	202,832
4	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2017_2_7_0.csv.gz	LOADED	270,066
5	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2018_2_1_0.csv.gz	LOADED	116,790
6	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2013_4_1_0.csv.gz	LOADED	81,506
7	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2014_5_3_0.csv.gz	LOADED	159,926
8	s3://snowflake-workshop-lab/citibike-trips-csv/trips_2016_3_4_0.csv.gz	LOADED	238,247

Query Details

- Query duration: 10s
- Rows: 376
- file: 100% filled
- status: LOADED

Content in a file obtained from a STAGE with this new warehouse configuration.

```

104
105    SELECT COUNT(*) FROM TRIPS;
106
107

```

Objects Editor Results Chart

	COUNT(*)
1	61,468,359

Query Details

- Query duration: 75ms
- Rows: 1
- COUNT(*) = 123
- 100% filled

Query details related to the COUNT statement with this new warehouse configuration.

In this case, our results are based on **10 seconds** during the loading process using COPY INTO a table as a target (*3 seconds less than the first attempt*) and **0.0075 seconds** to take the total amount of this table (*the best time compared to the first attempt.*)

Snowflake and JSON files

Snowflake is a data warehouse on AWS. The Snowflake **COPY** command lets you copy JSON, XML, CSV, Avro, Parquet, and XML format data files.

But to say that Snowflake supports JSON files is a little misleading—it does not parse these data files, as we showed in an [example with Amazon Redshift](#). Instead, Snowflake copies the entirety of the data into one Snowflake column of type **variant**. Then you run JSON SQL queries against that.

We will load two data files, which you can download from here:

- [Customers](#)
- [Orders](#)

Note: This is in **NDJSON** format. That means the entire file is not a valid JSON file. Instead it is composed of individual JSON records.

Create database and warehouse

We are running our Snowflake cluster on Amazon AWS. (It is not listed as a service on the Amazon AWS Console. Instead you sign up for it on the Snowflake site, then it launches an instance on Amazon, Microsoft, or Google clouds.)

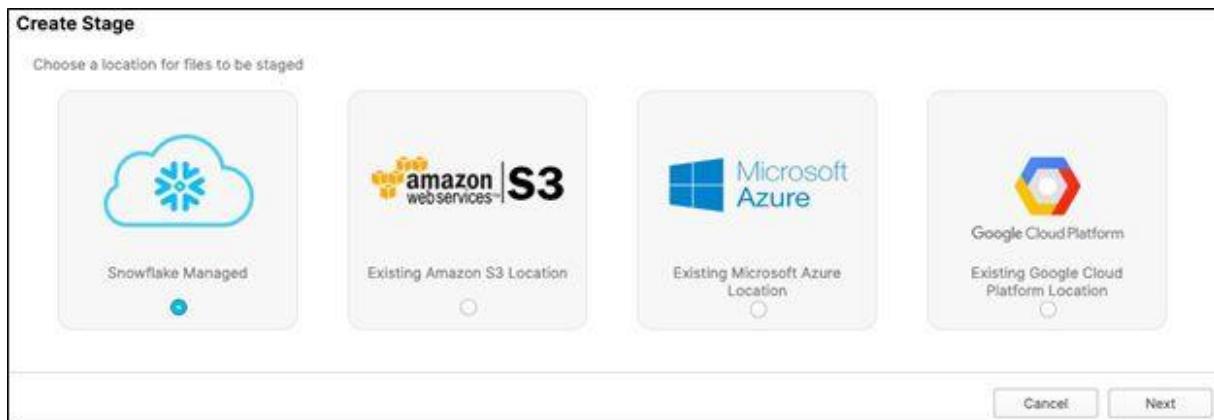
You create a warehouse like this. Here is where you pick the machine site and number of servers, thus picking the computing power and cost.

Create Warehouse

Name *	walkerwarehouse
Size	Small (2 credits / hour)
Learn more about virtual warehouse sizes here	
Maximum Clusters	1
Multi-cluster warehouses improve the query throughput for high concurrency workloads.	
Scaling Policy	Standard
The policy used to automatically start up and shut down clusters.	
Auto Suspend	10 minutes
The maximum idle time before the warehouse will be automatically suspended.	
<input checked="" type="checkbox"/> Auto Resume ?	
Comment	
Show SQL	Cancel Finish

Create external stage

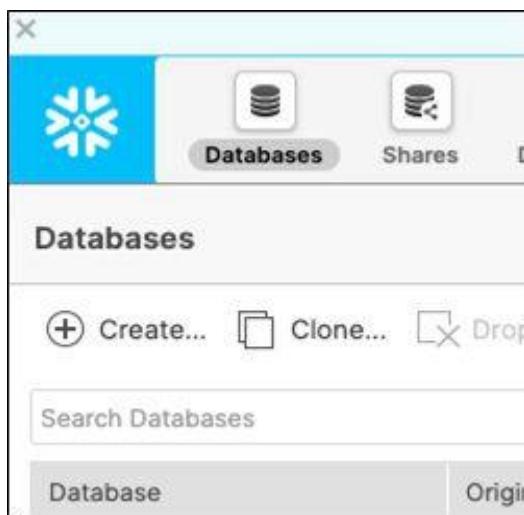
You can copy data directly from Amazon S3, but Snowflake recommends that you use their **external stage** area. They give no reason for this. But, doing so means you can store your credentials and thus simplify the copy syntax plus use wildcard patterns to select files when you copy them.



You give it a name and point it to an S3 bucket.

Create a database

As with other databases, a **database** is a collection of tables. So, it's just a name in this simple example. Create it from the Snowflake console like this:



Create tables

Next, open the worksheet editor and paste in this SQL to create the **customers** and **orders** tables. Note that both have one column of type **variant**.

```
use database inventory;
create table customers (customer variant);
create table orders (orders variant);
```

Upload JSON data to S3

Copy the data to S3 using the Amazon S3 console or AWS CLI command line:

```
aws s3 cp customers.json s3://(bucket name)
aws s3 cp orders.json s3://(bucket name)
```

Bulk load the JSON data into Snowflake

Copy the customers and orders data into Snowflake like this. Since that S3 bucket contains both files we give the name using **PATTERN**. That can be any regular expression.

```
copy into customers
from @GLUEBMCWALKERROWE
FILE_FORMAT=(TYPE= 'JSON')
PATTERN= 'customers.json';
```

One annoying feature is that you have to select the **All Queries** checkbox in order to enable the **run** button:



```
▶ Run (1)  All Queries | Saved 1 second ago

1 copy into customers
2   from @GLUEBMCWALKERROWE
3   FILE_FORMAT=(TYPE= 'JSON')
4   PATTERN= 'customers.json';
5
6
7
```

The worksheet looks like this when you execute the SQL.

```
1
2 copy into orders
3 from @GLUEBMCWALKERROWE
4 FILE_FORMAT=(TYPE='JSON')
5 PATTERN='orders.json';
6
7
8
9
```

Results Data Preview

✓ Query_ID SQL 640ms 1 rows

Filter result... Copy

Row	file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error	first_e
1	s3://gluebmc...	LOADED	2	2	1	0	NULL	

Then click on the file and see the data that it loaded.

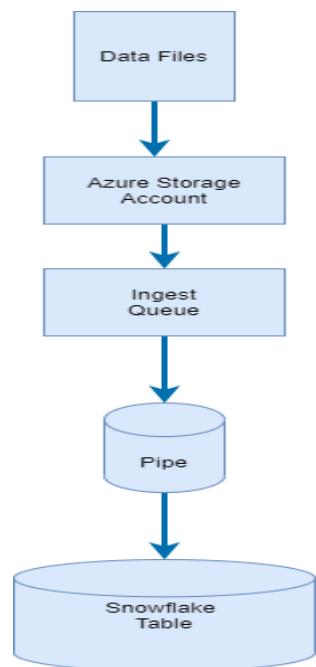
Details

```
1 {
2   "customername": "sgyomykutraerhwxuwp1",
3   "customernumber": "d5d5b72c-edd7-11ea-ab7a-0ec120e133fc",
4   "datecreated": "2020-09-03",
5   "email": "zybd@hcvc.com",
6   "locale": "en-US.utf-8",
7   "phonenumber": 7295614,
8   "postalcode": "fjqw"
9 }
```

Continuous Data Ingestion from Azure Blob Storage to Snowflake using Snowpipe

In this blog, you will learn to load data from Azure storage account to snowflake automatically using Snowpipe.

Process Flow



Process Flow Diagram

1. **Prerequisite** - Assume, we have a table in our Snowflake's Database, and we want to copy the data from Azure Blob Storage into our tables as soon as new files are uploaded into the Blob Storage.



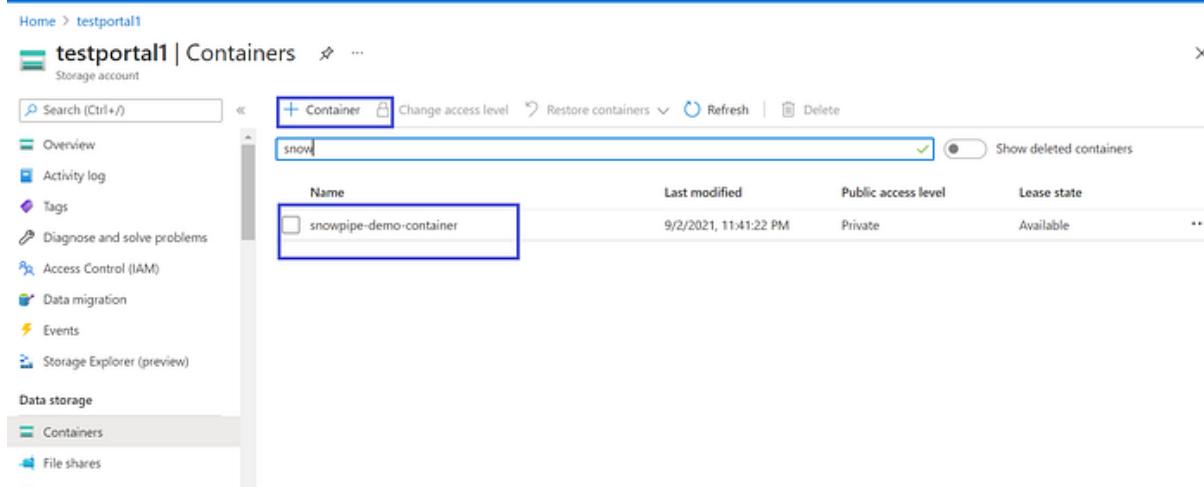
```
▶ Run | All Queries | Saved 0 seconds ago  
1  
2  
3 CREATE OR REPLACE TABLE SNOWPIPE_DEMO_DB.PUBLIC.STUDENTS.JSON(STUDENT_ID INT,NAME VARCHAR(200));
```

Create a table

2. Facilitate Azure Services and Snowflake to build a data pipeline to auto ingest files from Azure Blob Storage into Snowflake's table.

Create a storage account under the resource group. Example: "testportal1"

2.1 Create a container (Azure) - Example: "snowpipe-demo-container"



The screenshot shows the Azure Storage Explorer interface for a storage account named "testportal1". The left sidebar has a "Containers" section selected. The main area displays a table of containers with one entry:

Name	Last modified	Public access level	Lease state
snowpipe-demo-container	9/2/2021, 11:41:22 PM	Private	Available

Create a container

2.2 Create a queue (Azure) - Example: "snowpipe-demo-queue"

The screenshot shows the 'Queues' blade in the Azure portal for a storage account named 'testportal1'. The left sidebar has 'Queues' selected. The main area shows a table with one row for 'pipe-demo-queue'. A blue box highlights the '+ Queue' button in the top right and the queue name 'pipe-demo-queue' in the table.

Queue	Url
pipe-demo-queue	https://

Create a queue

2.3 Create an Event Grid Subscription (Azure) - Use the below step to create an Event Grid Subscription for the Container and set the endpoint to the Storage Queue.

Step 1: Navigate to Storage Account and click on Events, then click +Event Subscription.

The screenshot shows the 'Events' blade in the Azure portal for the same storage account 'testportal1'. The left sidebar has 'Events' selected. The main area shows a table with one row for 'Event Subscriptions'. A blue box highlights the '+ Event Subscription' button in the top right. A callout bubble on the right says 'Build reactive, is built into infrastructure'.

Event Subscriptions
Get Started

Create Event Subscription

Step 2: In the EVENT SUBSCRIPTION DETAILS Section, Give a name and choose **Event Grid Schema** for Event Schema.

The screenshot shows the 'Create Event Subscription' page in the Azure portal. At the top, there's a breadcrumb navigation: Home > testportal1 >. The main title is 'Create Event Subscription' with a 'Event Grid' icon. Below the title, there are four tabs: 'Basic' (selected), 'Filters', 'Additional Features', and 'Delivery Properties'. A note below the tabs says 'Pick a topic resource for which events should be pushed to your destination.' with a 'Learn more' link. Under 'Topic Type', 'Storage account' is selected. Under 'Source Resource', 'testportal1' is listed. Under 'System Topic Name', 'SNOWPIPE-DEMO-TOPIC' is entered and highlighted with a green checkmark. The 'EVENT TYPES' section has a note 'Pick which event types get pushed to your destination.' with a 'Learn more' link. A dropdown menu titled 'Filter to Event Types' shows 'Blob Created' selected. The 'ENDPOINT DETAILS' section has a note 'Pick an event handler to receive your events.' with a 'Learn more' link. Under 'Endpoint Type', 'Storage Queues (change)' is selected. At the bottom left is a blue 'Create' button.

Fill details in Event Subscriptions

Step 3: In Topic Details Section, Give a name for the Topic

Step 4: In the Event Types Section, Choose only **Blob Created** and uncheck the others.

Step 5: In Endpoint Details Section, Choose **Storage Queues** as endpoint type.

Step 6: Now click on Select an endpoint, then Select your subscription, Storage Account the created Storage Queue, finally, click on Confirm Selection.

Home > testportal1 > Create Event Subscription >

Select Storage Account

Event Grid

Subscription

BI3-Scalene

Select Storage Account

Selected Account

testportal1

Storage Queues

pipe-demo-queue

Confirm Selection

Select storage account

Step 7: Click on Create, now the Topic and the Event Subscription creation are completed.

2.4 Create a Notification Integration in Snowflake (Snowflake)

For this we need, the Azure Tenant Id and Storage Queue URL

Step 1: To get the Storage Queue URL, Navigate to Storage Account, then click on Queues, Copy the URL of the desired queue.

Queue	Url
<input type="checkbox"/> pipe-demo-queue	https:// - queue

Copy Storage Queue URL

Step 2: To get the Tenant Id, Navigate to Azure Active Directory Service, In the Overview pane, copy the Tenant Id.

The screenshot shows the Azure Active Directory Overview page. On the left, there's a sidebar with 'Overview', 'Preview features', 'Diagnose and solve problems', and sections for 'Manage' like 'Users', 'Groups', 'External Identities', etc. The main area has tabs for 'Overview', 'Monitoring', and 'Tutorials'. A search bar says 'Search your tenant'. Under 'Basic information', it shows:

Name	BI3Technologies.com	Users	48
Tenant ID	1-d49a-45bd-8ad1-886951	Groups	20
Primary domain	thirubi3technologies.onmicrosoft.com	Applications	24
License	Azure AD Free	Devices	30

Below that is a 'My feed' section.

Copy Tenant ID

Step 3: Now Navigate to Snowflake and change the role to ACCOUNTADMIN.

Step 4: Use the below syntax for creating Notification Integration.

```
// Create Notification Integration
CREATE NOTIFICATION INTEGRATION
INTEGRATION_NAME=ENABLED=TRUE TYPE=QUEUE NOTIFICATION_PROVIDER=AZURE_STORAGE_QUEUE
AZURE_STORAGE_QUEUE_PRIMARY_URI='STORAGE_QUEUE_URL' AZURE_TENANT_ID='AD_TENANT_ID';
```

Replace the **STORAGE_QUEUE_URL** with the actual Storage Queue URL and **AD_TENANT_ID** with the actual Tenant Id.

```
// Create Notification Integration
CREATE NOTIFICATION INTEGRATION
SNOWPIPE_DEMO2_NOTFINT ENABLED=TRUE TYPE=QUEUE NOTIFICATION_PROVIDER=AZURE_STORAGE_
QUEUE AZURE_STORAGE_QUEUE_PRIMARY_URI='https://***.core.windows.net/***/queue'
AZURE_TENANT_ID='***-d49a-45bd-8ad1-886951**';
```

Step 5: Now snowflake knows that this integration is for Azure, where will it get the event notifications, but we need Azure also to authenticate our Snowflake Account one time.

Execute the below command to achieve this.

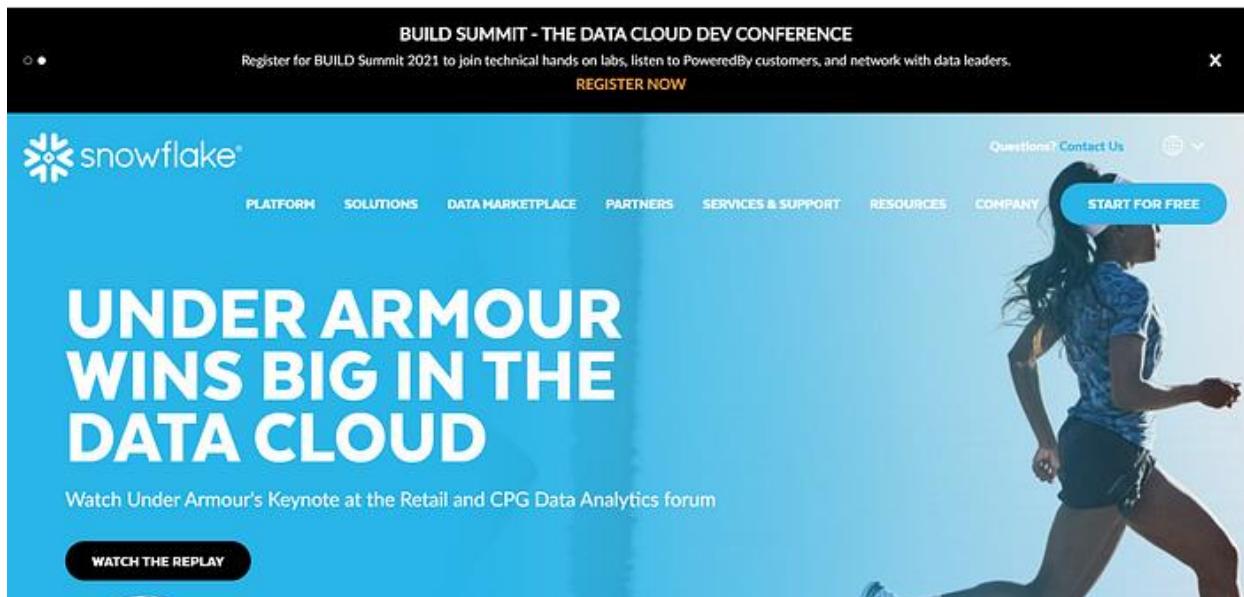
```
DESC NOTIFICATION INTEGRATION SNOWPIPE_DEMO2_NOTFINT;
```

It gives an **AZURE_CONSENT_URL** which will be used to establish a protocol between Azure Account and Snowflake Account.

Step 6: Copy the URL, and paste it into the browser, where you have already logged into Azure Portal.

Step 7: Click on Accept.

Once the establishment is successful, you will be redirected to the Snowflake page like below.



Azure Portal

2.5 Confirm Snowflake is authorized

Navigate to Azure Portal and redirect to the Active Directory service page, and click on Enterprise Applications.

The screenshot shows the 'Enterprise applications | All applications' page in Azure Active Directory. The left sidebar includes sections for Overview, Diagnose and solve problems, Manage (with 'All applications' selected), and Security. The main area has filters for Application type (Enterprise Applications), Applications status (Any), and Application visibility (Any). A search bar at the top right contains the query 'snowflakePACInt0181'. Below the search bar, a table lists one application: 'SnowflakePACInt0181' with the URL 'https://snowflake.net'. The table columns are Name, Homepage URL, Object ID, and Application ID.

Enterprise Application

You can see now Snowflake is added as an Enterprise Application.

2.6 Grant Storage access to the Snowflake Application (Azure)

Step 1: Copy the name of the Snowflake Enterprise Application.

Name: SnowflakePACInt0181

Step 2: Navigate to Azure Storage Account and choose Access Control IAM.

Step 3: Click on Add and Add Role Assignment.

The screenshot shows the 'Access Control (IAM)' blade for a storage account. On the left, there's a list of roles: 'test' (selected), 'testportal1', 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'Access Control (IAM)' (highlighted with a blue box), 'Data migration', 'Events', 'Storage Explorer (preview)', 'Data storage', 'Containers', 'File shares', 'Queues', and 'Tables'. In the center, there's a 'My access' section with a 'View my access' button. To the right, there are three main sections: 'Grant access to this resource' (with a 'Grant access to resources by assigning a role.' link), 'Add role assignment (Preview)' (with a 'Use the classic experience' link and a 'Learn more' link), and 'View access to this resource' (with a 'View' button and a 'Learn more' link). At the bottom right, there's a 'View deny assignments' link.

Role Assignment

Step 4: Choose Storage Queue Data Contribute for Role.

Step 5: Choose our created Snowflake Enterprise Application and click on Save.

Add role assignment

Selected role: Storage Blob Data Contributor

Assign access to: User, group, or service principal Managed identity

Members: + Select members

Name	Object ID	Type
SnowflakePACInt0181	-4315-9aaf-48236ad250...	App

Description: Optional

Review + assign Previous Next

Add role assignment

Now the Snowflake application is added to the Storage Queue Data Contributor role.

2.7 Create Stages in Snowflake for two different file formats (Snowflake) - For this, we need two pieces of information. One is Blob Service URL and Shared Access Signature Token which we need to generate.

Step 1: To get the Blob Service Endpoint, Go to Storage Account, Click on Endpoints and copy the Blob Service Endpoint.

Storage accounts

testportal1 | Endpoints

Security + networking

Provisioning state: Succeeded

Created: 2/4/2020, 4:47:15 PM

Last Failover: Never

Storage account resource ID:

Blob service

Resource ID: https://core.windows.net/

Primary endpoint: Blob service

Secondary endpoint: Blob service

File service

Resource ID:

Copy Blob Service Endpoint

Step 2: To get the Shared Access Signature Token, Click on Shared Access Signature

Step 3: In the selection pane, select only the required permissions.

Step 4: Now click on Generate SAS and Connection String and Copy the SAS Token

The screenshot shows the Azure Storage Accounts interface. On the left, there's a list of storage accounts: 'test' and 'testportal1'. The 'testportal1' account is selected. On the right, a detailed view of the 'testportal1 | Shared access signature' settings is shown. Under 'Allowed services', 'Blob', 'File', 'Queue', and 'Table' are checked. Under 'Allowed resource types', 'Container' and 'Object' are checked. Under 'Allowed permissions', 'Read', 'Write', 'List', 'Add', 'Create', and 'Filter' are checked. Other options like 'Delete', 'Update', and 'Process' are unchecked. At the bottom, under 'Blob versioning permissions', 'Enables deletion of versions' is checked. Under 'Allowed blob index permissions', 'Read/Write' and 'Filter' are checked.

Generate SAS

This screenshot shows the 'Generate SAS and connection string' section of the Azure Storage interface. It displays three fields: 'Connection string', 'SAS token', and 'Blob service SAS URL'. The 'Connection string' field contains 'blobEndpoint=https://testportal1.blob.core.windows.net/';. The 'SAS token' field contains '?sv=2021-06-23T18:32Z&st=2021-06-23T18:32Z&se=2021-06-23T18:32Z&sr=c&sig=...'. The 'Blob service SAS URL' field contains 'https://testportal1.blob.core.windows.net/'.

Generate SAS and Connection String and Copy the SAS Token

Step 5: Navigate to Snowflake and create a stage for JSON file using the below commands.

```
// Create Stage for a JSON File
CREATE OR REPLACE STAGE "SNOWPIPE_DEMO_DB"."PUBLIC".SNOWPIPE_STUDENT_STG_JSONURL =
'https://***.core.windows.net/snowpipe-demo-container/**/*' CREDENTIALS = (AZURE_SAS_TOKEN =
'?sv=***&se=2021-06-23T18:32Z&st=2021-06-23T18:32Z&sr=c&sig=...') FILE_FORMAT = (TYPE = 'JSON');Note: Replace the URL and AZURE_SAS_TOKEN with the actual values.To see the available staging files, execute the below commands.LS @SNOWPIPE_STUDENT_STG_JSON;
```

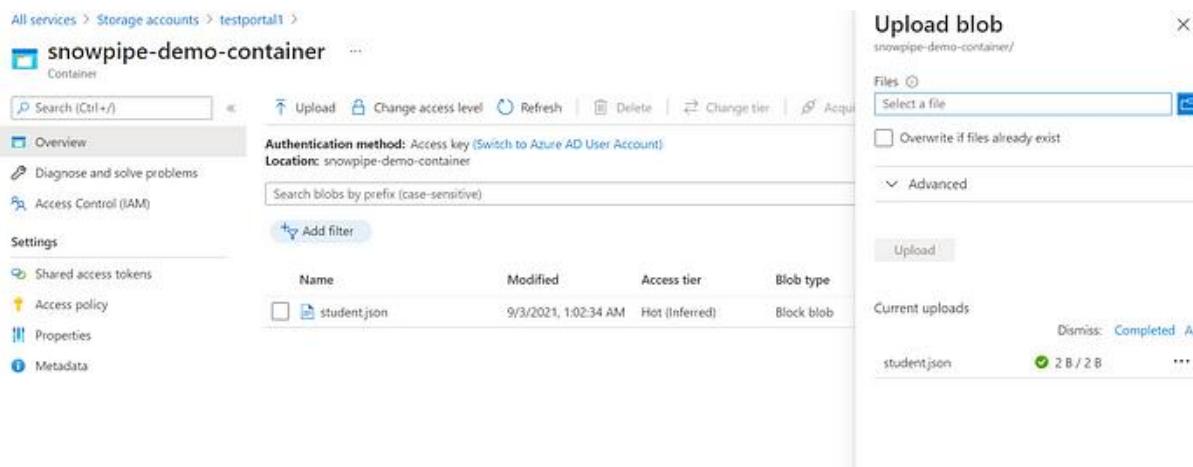
2.8 Creating SnowPipe (Snowflake)

Create the SnowPipe for different stage files using the below commands.

```
// Create pipe for Stage JSON File
CREATE OR REPLACE PIPE
SNOWPIPE_DEMO_DB.PUBLIC.DEMO_SNOWPIPE_STUDENTS_JSON
AUTO_INGEST=TRUE
INTEGRATION=SNOWPIPE_DEMO2_NOTFINTASCOPY INTO
SNOWPIPE_DEMO_DB.PUBLIC.STUDENTS_JSON(STUDENT_ID,NAME)
FROM (SELECT $1:STUDENT_ID,
$1:NAME FROM @SNOWPIPE_DEMO_DB.PUBLIC.SNOWPIPE_STUDENT_STG_JSON );
```

2.9 The Results

As soon as we created the pipe, Let's try uploading the file, as of now, the records count is 0 in a table.



The screenshot shows the Azure Storage Explorer interface. On the left, there is a navigation tree with 'All services > Storage accounts > testportal1 > snowpipe-demo-container'. The main area shows a table with one row:

Name	Modified	Access tier	Blob type
student.json	9/3/2021, 1:02:34 AM	Hot (Inferred)	Block blob

To the right, a modal window titled 'Upload blob' is open, showing a file selection input field with 'student.json' selected, and an 'Upload' button.

Upload file in blob

Use the below command to make sure our file is available in staging.

```
LS @SNOWPIPE_STUDENT_STG_JSON;
```

If the tables are not loaded with values try refreshing the pipes using the below commands.

```
ALTER PIPE DEMO_SNOWPIPE_STUDENTS_JSON REFRESH;
```

Our file was added to the stage, now let's check target table is loaded with new data by executing the below command.

```
SELECT * FROM SNOWPIPE_DEMO_DB.PUBLIC.STUDENTS_JSON;
```

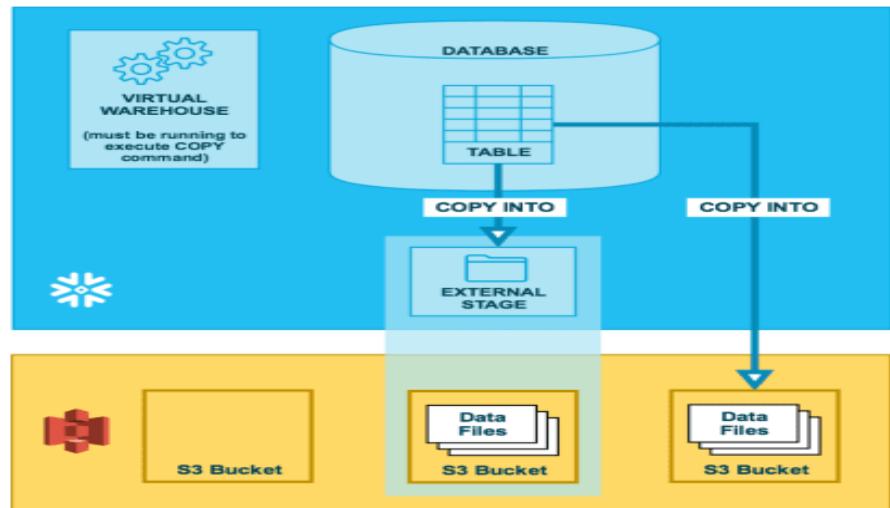
Row	STUDENT_ID	NAME
1	1	GOWTHAM
2	2	HARISH
3	3	JANANI
4	4	JAYASHREE
5	5	JAGAN
6	6	RASIIKA
7	7	VIZI

Target table

Yes, our table is loaded with new values, we have now successfully set up a data pipeline for continuous data ingestion from Azure Blob Storage to Snowflake.

Finally, We have accomplished loading the data from Azure blob storage into snowflake using Snowpipe. It uses the Microsoft Azure Queue and Event Subscriptions and a Snowflake container to load the file into Blob which acts as Source to the Snowpipe and target table to load the data.

Steps for Snowflake Unload to S3



Now that you have understood about Snowflake and Amazon S3. In this section, you will learn the process of Snowflake Unload to S3. The following steps to easily perform Snowflake Unload to S3 are listed below:

- [Step 1: Allowing the Virtual Private Cloud IDs](#)
- [Step 2: Configuring an Amazon S3 Bucket](#)
- [Step 3: Unloading Data into an External Stage](#)

Step 1: Allowing the Virtual Private Cloud IDs

The foremost step for Snowflake Unload to S3 is to explicitly grant Snowflake access to your AWS S3 storage account. For this, the location or region for both Amazon S3 located in your AWS account and Snowflake should be the same.

- First, log in to your Snowflake account.
- Then, set the **ACCOUNTADMIN** as an active role for the user session by executing the following code in the Snowflake console.

```
use role accountadmin;
```

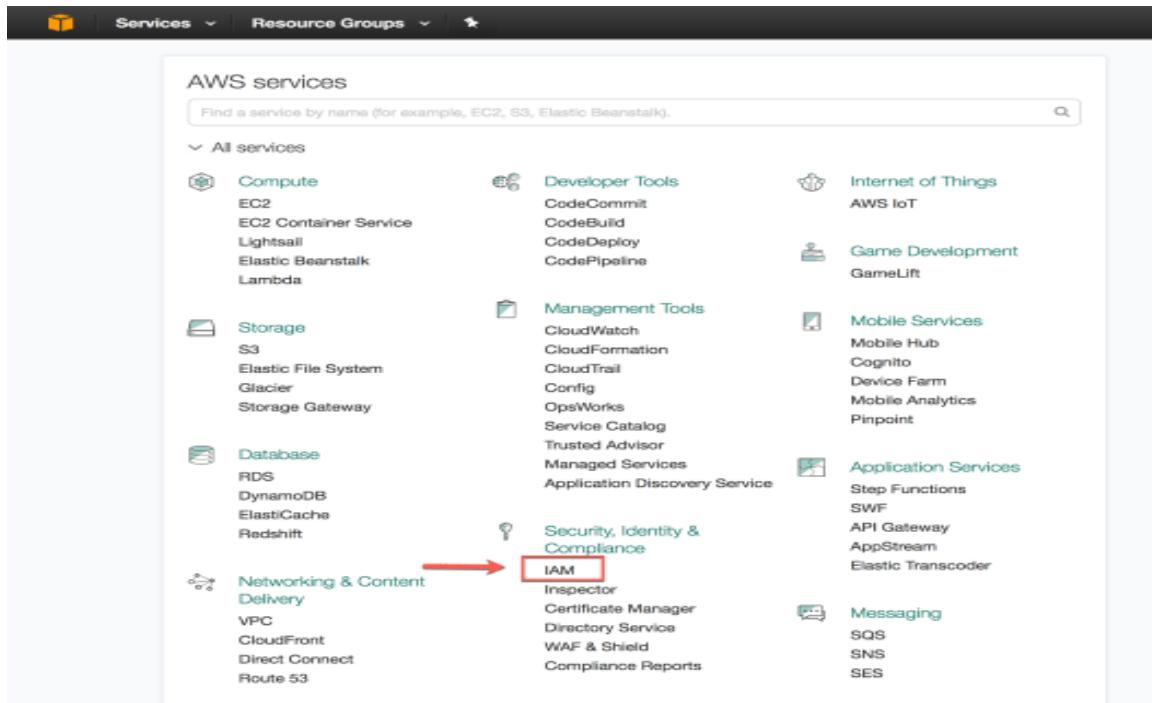
- Now, retrieve the IDs of the AWS Virtual Network (VNet) by querying the SYSTEM\$GET_SNOWFLAKE_PLATFORM_INFO function in the console. You can follow the following code given below.

```
select system$get_snowflake_platform_info();
```

- After getting the query result from the above replace the VPC IDs.
- Allow the VPC IDs by creating an Amazon S3 policy for a specific VPC.
- Now, let's provide an AWS Identity and Access Management (IAM) role to Snowflake so that you can access the Amazon S3 Bucket.

Step 2: Configuring an Amazon S3 Bucket

- You must allow Snowflake to access your Amazon S3 Bucket and folder to create new files. The following permissions are required for Snowflake Unload to S3:
 - s3:DeleteObject
 - s3:PutObject
- First, you have to create an IAM policy by logging in to your AWS Management Console and from the dashboard navigating to the **Identity & Access Management (IAM)** options under **Security, Identity and Compliance** section, as shown in the image below.



- Now, choose the “**Account Settings**” option from the left side navigation bar.
- Here, just expand the “**Security Token Service Regions**” list corresponding to the region your AWS account is located.
- Then, “**Activate**” the status if “**Inactive**”.
- Choose the “**Policies**” option from the left-hand navigation bar.
- Then, click on the “**Create Policy**” option, as shown in the image below.
-

The screenshot shows the IAM Policies page. On the left, there's a navigation menu with 'Policies' selected. The main area displays a table of existing policies, with one row highlighted. The columns in the table are 'Policy Name', 'Attached Entities', 'Creation Time', and 'Edited Time'. The table shows 242 results.

Policy Name	Attached Entities	Creation Time	Edited Time
AmazonS3FullAccess	2	2015-02-06 13:40 EST	2015-02-06 13:40 EST
AmazonS3ReadOnlyAccess	2	2015-02-06 13:40 EST	2015-02-06 13:40 EST
AdministratorAccess	1	2015-02-06 13:39 EST	2015-02-06 13:39 EST
IAMUserChangePassword	5	2015-11-14 19:25 EST	2016-11-15 18:18 EST
SelfPasswordChange	5	2016-12-27 10:19 EST	2016-12-27 10:19 EST
AmazonAPIGatewayAdministrator	3	2015-07-09 13:34 EST	2015-07-09 13:34 EST
AmazonAPIGatewayInvokeFullAccess	3	2015-07-09 13:36 EST	2015-07-09 13:36 EST
AmazonAPIGatewayPushToCloudWatchLogs	3	2015-11-11 18:41 EST	2015-11-11 18:41 EST
AmazonAppStreamFullAccess	3	2015-02-06 13:40 EST	2015-02-06 13:40 EST
AmazonAppStreamReadOnlyAccess	3	2015-02-06 13:40 EST	2016-12-07 16:00 EST
AmazonAppStreamServiceAccess	3	2016-11-18 23:17 EST	2016-11-18 23:17 EST

- Click on the “**JSON**” tab and add a policy document to allow Snowflake Unload to S3.
- A sample policy document in JSON format is given below.

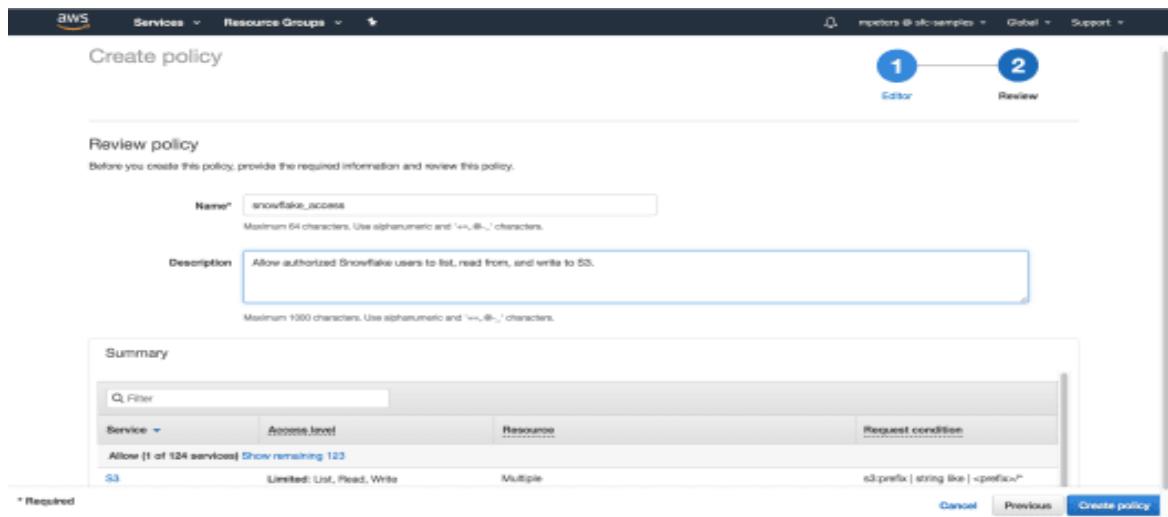
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::snowflake-data-warehouse/*"
      ]
    }
  ]
}
```

```

    "s3:PutObject",
    "s3:GetObject",
    "s3:GetObjectVersion",
    "s3:DeleteObject",
    "s3:DeleteObjectVersion"
],
"Resource": "arn:aws:s3:::<bucket>/<prefix>/*"
},
{
"Effect": "Allow",
"Action": [
    "s3>ListBucket",
    "s3:GetBucketLocation"
],
"Resource": "arn:aws:s3:::<bucket>",
"Condition": {
    "StringLike": {
        "s3:prefix": [
            "<prefix>/*"
        ]
    }
}
]
}

```

- In the above code replace the <bucket> and <prefix> with the bucket name and folder path prefix.
- Then, click on the “Review policy” button and enter the policy name eg.- snowflake_access.
- Click on the “Create policy” button, as shown in the image below.



- Now, it's time to create the IAM Role in AWS for Snowflake unload to S3.
- Choose the “Roles” from the left navigation bar and click on the “Create role” button, as shown in the image below.

Services ▾ Resource Groups ▾

Create role

Select type of trusted entity

AWS service EC2, Lambda and others

Another AWS account Belonging to you or 3rd party

Web Identity OpenID or any OpenID provider

SAML 2.0 federation Your corporate directory

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

Specify accounts that can use this role

Account ID* 123456789000

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

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

External ID
0000

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

* Required

Canceled Next: Permissions

- Here, select the “**Another AWS account**” as the trusted entity type.
- In the Account ID field, provide your AWS account ID.
- Then, select the “**Require external ID**” option. For now, enter the dummy ID as “**0000**”. You will need to modify it later to grant Snowflake Unload to S3.
- Click on the “**Next**” button, locate the policy that you created and again click on the “**Next**” button, as shown in the image below.

Services ▾ Resource Groups ▾

Create role

Review

Provide the required information below and review this role before you create it.

Role name* mysnowflakerole

Role description My Snowflake role

Trusted entities The account 000000000000

Policies snowflake_access

* Required

Canceled Previous Create role

- Create the role by providing a valid name and description. You will have an IAM policy for the bucket, IAM role, and attached policy.
- After this, you have to record the “**Role ARN**” value located on the role summary page, as shown in the image below.

- Now, create a Snowflake Unload to S3 integration that references this role.
- Next, you need to create a Cloud Storage integration in Snowflake using the “**CREATE STORAGE INTEGRATION**” command.
- Open up the console and enter the following code given below.

```
CREATE STORAGE INTEGRATION <integration_name>
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = S3
ENABLED = TRUE
STORAGE_AWS_ROLE_ARN = '<iام_role>'
[ STORAGE_AWS_OBJECT_ACL = 'bucket-owner-full-control' ]
STORAGE_ALLOWED_LOCATIONS = ('s3://<bucket>/<path>/', 's3://<bucket>/<path>/')
[ STORAGE_BLOCKED_LOCATIONS = ('s3://<bucket>/<path>/', 's3://<bucket>/<path>/' ) ]
```

- Everything is ready for Snowflake Unload to S3, now you have to retrieve the AWS IAM user for your Snowflake account.
- For this, execute the following command given below to retrieve the ARN.

```
DESC INTEGRATION <integration_name>;
```

- Here, provide the “**integration_name**” that you created.
- For example, the sample command is given below with the output.

```
desc integration s3_int;

+-----+
+=====+
-----+
| property      | property_type | property_value |
property_default |
+-----+
+=====+
-----+
| ENABLED       | Boolean     | true           | false        |
| STORAGE_ALLOWED_LOCATIONS | List       | s3://mybucket1/mypath1/,s3://mybucket2/mypath2/ | []          |
| STORAGE_BLOCKED_LOCATIONS | List       | s3://mybucket1/mypath1/sensitivedata/,s3://mybucket2/mypath2/sensitivedata/ | []          |
```

```

| STORAGE_AWS_IAM_USER_ARN | String      | arn:aws:iam::123456789001:user/abc1-b-self1234
|                         |
| STORAGE_AWS_ROLE_ARN   | String      | arn:aws:iam::001234567890:role/myrole
|                         |
| STORAGE_AWS_EXTERNAL_ID | String      |
MYACCOUNT_SFCRole=2_a123456/s0aBCDefGHIJklmNoPq=
+-----+
+=====+
-----+

```

- Record the “**STORAGE_AWS_IAM_USER_ARN**” and “**STORAGE_AWS_EXTERNAL_ID**” values from the output.
- Next, click on the “**Trust relationships**” from the “**Roles**” option located at the side navigation bar.
- Then, click on the “**Edit trust relationship**” button.
- Now, you can modify the policy document with the “**DESC STORAGE INTEGRATION**” output values you just recorded.
- The sample policy document for Snowflake Unload to S3 is given below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "<snowflake_user_arn>"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "<snowflake_external_id>"
        }
      }
    }
  ]
}
```

- In the above document, replace the recorded values.
- Then, click on the “**Update Trust Policy**” button.
- Now, you have to create an external stage for Snowflake Unload to S3 that references the storage integration you created earlier.
- You have to create the stage using the “**CREATE STAGE**” command as given below.

```
grant create stage on schema public to role myrole;

grant usage on integration s3_int to role myrole;
```

- Now, set the “mydb.public” as the current Database and schema for the user session by creating the named stage “**my_s3_stage**”. You can follow the sample code given below.

```
use schema mydb.public;

create stage my_s3_stage
storage_integration = s3_int
url = 's3://bucket1/path1'
file_format = my_csv_format;
```

Step 3: Unloading Data into an External Stage

- Create an external named stage using the web interface by clicking on the “**Databases**” option then, “**<db_name>**” and selecting then Stages.
- You can also do the same thing using the SQL command given below.

CREATE STAGE

- The following sample code will create an external stage for Snowflake Unload to S3 given below.

```
create or replace stage my_ext_unload_stage url='s3://unload/files/'
storage_integration = s3_int
file_format = my_csv_unload_format;
```

- In the above code, the name of the external stage is “**my_ext_unload_stage**” with bucket name as “**unload**” and folder “**path**” as files.
- Here, the data to unload as the file format object is “**my_csv_unload_format**”.
- Now, use the COPY INTO command for Snowflake Unload to S3 from a table using the external stage.
- The code for Snowflake Unload to S3 using the stage is given below.

```
copy into @my_ext_unload_stage/d1 from mytable;
```

- In the above code, all the rows in “**mytable**” table are Snowflake Unload to S3 bucket. A “**d1**” filename prefix is applied to the files using the code given below.
- Then, retrieve the objects from the S3 bucket using the S3 console from the output.

That's it! You have successfully performed Snowflake Unload to S3.

Conclusion

In this article, you learnt about the Snowflake, Amazon S3, and simple steps to Snowflake Unload to S3. The process of Snowflake Unload to S3 consists of accessing required permission from the AWS console and Snowflake account. Then, configuring the staging area along with the data that needs to be unloaded from Snowflake to S3. Snowflake Data Unloading to S3 helps companies cut the cost of storing data that is not needed in Data Warehouse or create a storage area with Snowflake.

What is a Snowflake Task?

A **Snowflake Task** allows scheduled execution of SQL statements including calling a stored procedure or Procedural logic using Snowflake Scripting.

To create a task you need to be defining the following optional parameters using CREATE TASK along with the SQL code.

- The compute resources using which the SQL code executes using a **WAREHOUSE** parameter.
- The schedule details when the code needs to be executed using a **SCHEDULE** parameter.

Few key points to be noted before we get into building Snowflake Tasks

- Only one SQL statement is allowed to be executed through a task. If you need to execute multiple statements, build a procedure.
- Once task is created it will be in suspended state. You need to be manually resume the task using ALTER TASK.
- The Schedule parameter takes only minutes. It does not support second or hour.
- The minimum value of a schedule parameter is 1 minute and the maximum value that can be assigned to schedule parameter is 8 days i.e 11520 minutes.

3. Building a Snowflake Task

Tasks require compute resources to execute SQL code. Either of the following compute models can be chosen for individual tasks:

- User-managed (i.e. Virtual warehouse)
- Snowflake-managed (i.e. Serverless compute model)

3.1. Building User-managed Snowflake Task

You can manage the compute resources for individual tasks by specifying an existing virtual warehouse when creating the task. Make sure you choose a right sized warehouse for the SQL actions defined in task.

Below is an example which creates a user-managed task that inserts data into employees table every 5 minutes using **COMPUTE_WH** warehouse.

```
CREATE TASK mytask
```

```
WAREHOUSE = COMPUTE_WH
```

```
SCHEDULE = '5 MINUTE'
```

```
AS
```

```
INSERT INTO employees VALUES( EMPLOYEE_SEQUENCE.NEXTVAL,'F_NAME','L_NAME','101')
```

```
;
```

If WAREHOUSE parameter is not defined, by default the task is created with Snowflake-managed compute resources, also referred as Serverless compute model.

3.2. Building Serverless Snowflake Task

The Serverless compute model for tasks enables you to rely on compute resources managed by Snowflake instead of user-managed virtual warehouses.

The compute resources are automatically scaled up or down by Snowflake as required for each workload in serverless model.

Below is the same example which creates a task using serverless compute model.

```
CREATE TASK mytask_serverless
  USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE = 'XSMALL'
  SCHEDULE = '5 MINUTE'
  AS
    INSERT INTO employees VALUES( EMPLOYEE_SEQUENCE.NEXTVAL,'F_NAME','L_NAME','101')
;
```

To specify the initial warehouse size for the task, set the **USER_TASK_MANAGED_INITIAL_WAREHOUSE_SIZE** parameter. Though the tasks starts with a XSMALL warehouse, if it sees a needs for more capacity, it will go for a bigger warehouse.

4. Scheduling a Snowflake Task

Snowflake Tasks are not event based, instead a task runs on a schedule. The Snowflake task engine has a **CRON** and **NONCRON** variant scheduling mechanisms. You must be familiar with CRON variant's syntax if you are a Linux user.

A *schedule must be defined for a task or the root task in a task tree; otherwise, the task only runs if manually executed using [EXECUTE TASK](#).*

4.1. Scheduling a Snowflake Task in NON-CRON notation

The above examples which we saw while building User-managed and Serverless tasks in sections [3.1](#) and [3.2](#) are scheduled in NON-CRON notation.

```
1 CREATE TASK mytask
2   WAREHOUSE = COMPUTE_WH
3   SCHEDULE = '5 MINUTE'
4 AS
5   INSERT INTO employees VALUES( EMPLOYEE_SEQUENCE.NEXTVAL, 'F_NAME', 'L_NAME', '101')
6 ;
```

Create task in NON-CRON notation

The disadvantage with NON-CRON notation is that you can only schedule a job which runs at a specific interval. You will not be able to schedule a job which triggers at a specified time.

For example if you scheduled a job to run every 10 minutes in NON-CRON mode and the job started at 12:07, the next run will be at 12:17.

4.2. Scheduling a Snowflake Task in CRON notation

To schedule a snowflake task in CRON notation, you must use the keyword USING CRON followed by 5 asterisks and the time zone.

Each asterisk denotes a specific time value as shown below.

MIN	HOUR	DAY	MON	WEEKDAY	DESCRIPTION
*	*	*	*	*	Every minute
0	2	*	*	SUN	Every Sunday at 2 PM
0	5,17	*	5	*	Twice daily, at 5AM and 5PM in May month

Below is an example of Snowflake task in CRON notation which runs every Sunday at 10 AM UTC.

```
CREATE OR REPLACE TASK my_crontask
```

```
WAREHOUSE = COMPUTE_WH
```

```
SCHEDULE = 'USING CRON * 10 * * SUN UTC'
```

```
AS
```

```
INSERT INTO employees VALUES( EMPLOYEE_SEQUENCE.NEXTVAL,'F_NAME','L_NAME','101')
```

```
;
```

5. Turning the Snowflake Tasks on and off

Once task is created, its status can be verified in Snowflake using SHOW TASKS.

SHOW TASKS;

8 SHOW TASKS;						
Results		Data Preview				
✓ Query ID SQL		57ms	1 rows			
Filter result...						
name	database_name	schema_name	owner	warehouse	schedule	state
MYTASK	ANALYTICS	HR	ACCOUNTADMIN	COMPUTE_WH	5 MINUTE	suspended

The initial status of the task will be “suspened” when created. To turn on a Snowflake task, issue below alter task command.

ALTER TASK mytask RESUME;

10 ALTER TASK mytask RESUME; 11 SHOW TASKS;						
Results		Data Preview				
✓ Query ID SQL		44ms	1 rows			
Filter result...						
name	database_name	schema_name	owner	warehouse	schedule	state
MYTASK	ANALYTICS	HR	ACCOUNTADMIN	COMPUTE_WH	5 MINUTE	started

To turn off a Snowflake task, issue below alter task command.

ALTER TASK mytask SUSPEND;

6. How to verify the task history of a Snowflake Task?

The status of every task run can be verified using the below query which provides the entire task history.

--CHECK TASK HISTORY

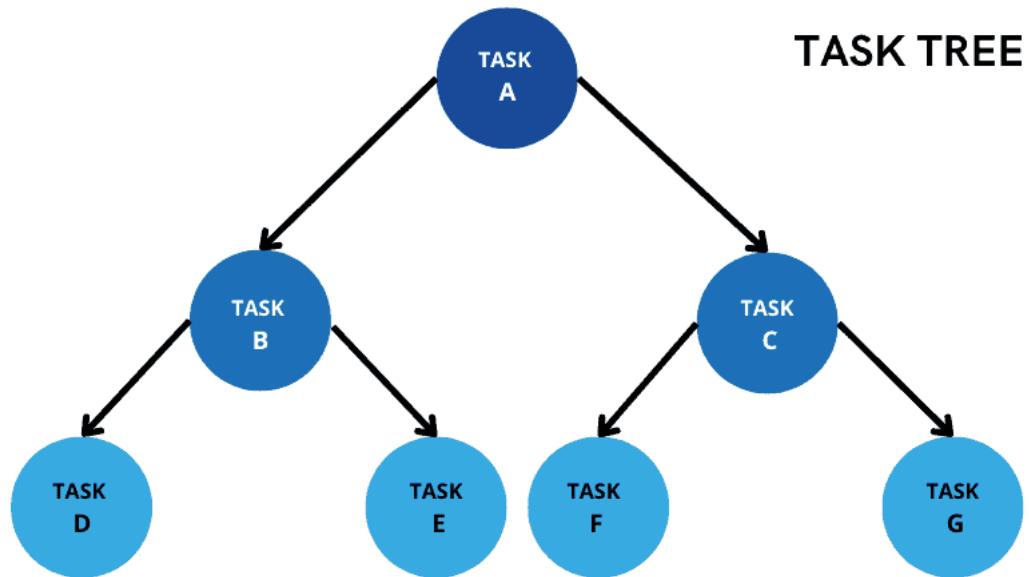
```
SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY()) WHERE NAME = 'MYTASK';
```

15 SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY()) WHERE NAME = 'MYTASK';							
Results		Data Preview					
✓ Query_ID		SQL		482ms	90 rows	Columns ▾	
Filter result...		Download	Copy				
Row	QUERY_ID			NAME	DATABASE_NAME	SCHEMA_NAME	QUERY_TEXT
1	NULL	MYTASK	ANALYTICS	HR	INSERT INT...	NULL	SCHEDULED
2	01a50ef6-0...	MYTASK	ANALYTICS	HR	INSERT INT...	NULL	SUCCEEDED
3	01a50ef5-0...	MYTASK	ANALYTICS	HR	INSERT INT...	NULL	SUCCEEDED
4	01a50ef4-0...	MYTASK	ANALYTICS	HR	INSERT INT...	NULL	SUCCEEDED
5	01a50ef3-0...	MYTASK	ANALYTICS	HR	INSERT INT...	NULL	SUCCEEDED
Check Task History							

If there is a failure you can find the error_code and error_message associated with the failure.

7. What is Snowflake Task Tree?

Consider a scenario where you wanted to trigger another task immediately after an initial task completes and so on. This is supported in Snowflake by creating a B-Tree-like task structure which is referred as Task Tree.



You can have only 1 root task and all child tasks are linked to the root task based on task dependency (i.e. before or after). Root tasks will have scheduler defined, and all child tasks follow sequential execution as per their dependency defined. The child tasks runs only after all specified predecessor task have successfully completed their own run.

The following example shows a task **mytask** set as a dependency for the task **mytask_2** to trigger. The task **mytask_2** which is a child task has no schedule of its own.

```
CREATE OR REPLACE TASK mytask_2
```

```
WAREHOUSE = COMPUTE_WH
```

```
AFTER mytask
```

```
AS
```

```
INSERT INTO mytable(ts) VALUES(CURRENT_TIMESTAMP)
```

```
;
```

Make sure the root task is suspended before assigning a child task; otherwise the child task creation fails.

We can verify the task dependence by using the following query

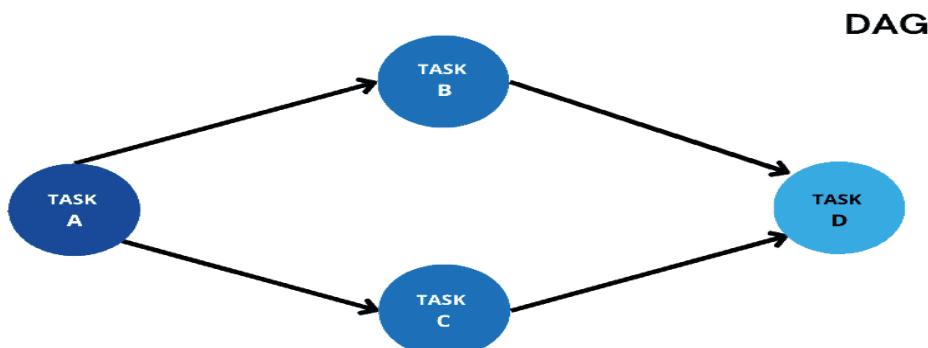
```
--CHECK DEPENDANT TASKS
```

```
SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_DEPENDENTS(TASK_NAME => 'mytask',  
RECURSIVE => FALSE));
```

23	SELECT * FROM					
24	TABLE(INFORMATION_SCHEMA.TASK_DEPENDENTS(TASK_NAME => 'mytask', RECURSIVE => FALSE));					
Results Data Preview						
Filter result... Columns ▾						
Row	NAME	OWNER	WAREHOUSE	SCHEDULE	PREDECESSOR	STATE
1	MYTASK	ACCOUNTADMIN	COMPUTE_WH	1 MINUTE	NULL	started
2	MYTASK_2	ACCOUNTADMIN	COMPUTE_WH	NULL	ANALYTICS.HR.MYTASK	suspended

Check Dependent Tasks

Consider a scenario where taskA is a root task and taskB and taskC are its child tasks. You wanted to define a taskD and specify taskB and taskC as its dependencies. This kind of task dependency set up is currently not supported in Snowflake where a task is dependent on multiple tasks.



Directed Acyclic Graph (DAG) of tasks is currently available under public preview as of June 2022 where it is possible to set these kind of dependencies.

8. Building a Snowflake Task that tracks INSERT operations from a Stream

Tasks can be combined with table streams for continuous ELT workflows to process recently changed table rows.

The below task tracks data for INSERT operations from a stream and inserts changes into a table every 5 minutes. The task polls the stream using the SYSTEM\$STREAM_HAS_DATA function to determine whether change data exists and, if the result is FALSE, skips the current run.

```
CREATE OR REPLACE TASK my_streamtask
  WAREHOUSE = COMPUTE_WH
  SCHEDULE = '5 minute'
  WHEN
    SYSTEM$STREAM_HAS_DATA('my_stream')
  AS
    INSERT INTO EMPLOYEE(id,name,salary) SELECT id,name,salary FROM my_stream WHERE
    metadata$action = 'INSERT' AND metadata$isupdate = 'FALSE';
```

9. Building a Snowflake Task that calls a Stored Procedure

The following query creates a task named mytask_sp that calls a stored procedure MY_STORED_PROC every hour.

```
CREATE TASK mytask_sp
  WAREHOUSE = MYWH
  SCHEDULE = '60 MINUTE'
  AS
    CALL MY_STORED_PROC()
;
;
```

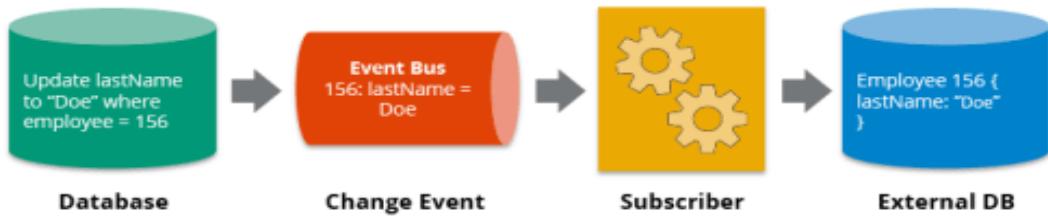
10. Manually executing a Snowflake Task

The EXECUTE TASK command manually triggers a single run of a scheduled task independent of the schedule defined for the task. This SQL command is useful for testing new or modified tasks before you enable them to execute SQL code in production.

```
EXECUTE TASK mytask;
```

In summary tasks are very handy in Snowflake, they can be combined with [Streams](#), [Snowpipe](#) and other techniques to make them extremely powerful.

What is Change Data Capture (CDC)?



Change Data Capture (CDC) is an ideal solution to capture the near real-time movement of data in Databases. CDC refers to the collection of software design patterns used to detect and track data changes in a Database. It triggers the event connected with data, causing a specific action to be executed for any Change Data Capture. All companies require access to real-time [Data Streams](#) to perform efficient Data Analytics and CDC provides near-real-time movement of data by processing data as soon as new database events occur.

Events are captured and streamed in real-time using CDC and it helps to achieve reliable, low-latency, and scalable data replication in high-velocity data environments. It eliminates the process of bulk data loading by implementing incremental data loading. This way, Data Warehouses or Databases remain active in order to execute specific actions as soon as a Change Data Capture event occurs. Furthermore, companies can send fresh data updates to BI (Business Intelligence) tools and team members in near real-time with CDC, keeping them up to date.

What is Snowflake Streams?

In today's data-driven economy, data in your systems change quite frequently, and it would be a complex task to load the data fully into Snowflake every time. It will cost you both time and money. This is where Snowflake Change Data Capture (CDC) comes in. You can implement CDC in Snowflake effectively by just using a few commands, and this is made possible with the concept of Streams in Snowflake.

A Snowflake Stream object basically tracks all DML changes made to rows in a source table and stores the metadata of each change. This metadata between two transactional points of time in a table is used later in order to retrieve the changed data.

[Snowflake Streams](#) capture an initial snapshot of all the rows present in the source table as the current version of the table with respect to an initial point in time. Streams then enable Change Data Capture every time you insert, update, or delete data in your source table. The Streams will have additional columns whenever any DML change is committed. So, by capturing the CDC Events you can easily merge just the changes from source to target using the [MERGE](#) statement.

How to Setup Snowflake Change Data Capture with Streams?

To get started with Snowflake Change Data Capture, log in to Snowflake Web-Based UI or SnowSQL. Then, follow the below-mentioned steps.

Step 1: Run the following command to create a [CDC_Stream](#).

```
create or replace database CDC_STREAM;
```

Use CDC_STREAM;

Step 2: You'll need a source and a destination table to operate upon. Run the following command to create a source table named "**employees_source**".

```
create or replace table employees_source (
id int,
first_name varchar(200),
last_name varchar(200) )
```

Step 3: Run the following command to create a destination table named "**employees_destination**".

```
create or replace table employees_destination ( id int, first_name varchar(200), last_name
varchar(200) );
```

Step 4: Run the following command to create a stream on top of the source table. This stream will track any data changes made in the **EMPLOYEES** table.

```
create or replace stream employee_stream on table employees_source;
```

Note: As long as there is no data management command to consume it, any changes to the source table with respect to data would be considered as **INSERT** and not **UPDATE**.

For the purpose of this demonstration, let's add a couple of records to the **employees_source** table. Run the following command to do so.

```
insert into employees_source values (1,'Ron','Higgins');
insert into employees_source values (2,'Joseph','Geller');
insert into employees_source values (3,'Charlie','Bricks');
insert into employees_source values (4,'Mathew','Smith');
insert into employees_source values (5,'Daisy','Whelsh');
insert into employees_source values (6,'Scott','Hall');
insert into employees_source values (7,'Jose','Martinez');
insert into employees_source values (8,'Bryan','Son');
insert into employees_source values (9,'Nisha','Grant');
insert into employees_source values (10,'Hamilton','Gell');
insert into employees_source values (11,'Larry','Armstrong');
insert into employees_source values (12,'Emilia','Rodriguez');
insert into employees_source values (13,'Kate','Becket');
```

Step 5: Let's view the change log in the stream before proceeding.

```
select * from employee_stream;
```

As you can see, here are the records and metadata fields. There are 3 additional META columns (**METADATA\$ROW_ID**, **METADATA\$ISUPDATE**, and **METADATA\$ACTION**) introduced and they make it very easy to detect if a row has been updated, deleted, or inserted. And, since it's an initial load everything is inserted.

Now, there are no records in the destination table as they're still in the stream.

Step 6: Let's move the records to the destination using the **MERGE** statement.

```
Use CDC_STREAM;
MERGE into employees_destination as T
using (select *
from employee_stream) AS S
ON T.id = s.id
when matched AND S.metadata$action = 'INSERT' AND S.metadata$isupdate
THEN
update set T.first_name = S.first_name, T.last_name = S.last_name
When matched
And S.metadata$action = 'DELETE' THEN DELETE
when not matched
And S.metadata$action = 'INSERT' THEN
INSERT (id,
first_name,
last_name)
VALUES (S.id,
S.first_name,
S.last_name);
```

Now, the data has moved to the destination and there will be nothing in the stream. After the **MERGE** command has consumed, the stream object will become empty.

The cycle continues, and the stream will continue to record all the changes that happen in the **employees_source** table.

Step 7: Let's view the destination table now. Run the following command to do so.

```
select * from employees_destination;
```

As you can see, the destination table is updated with the records.

Step 8: Now, it's time to demonstrate the Snowflake Change Data Capture. For that purpose, go ahead and make a change in the source table. Let's update a record and observe the stream.

```
update employees_source
set last_name="Harper"
where id=1;
```

This basically sets the **last_name** as “**Harper**” for **id=1**.

- While observing the stream, you'll find 2 records, **INSERT** and **DELETE**. This is because the original **last_name** (i.e Higgins) was deleted for **id=1** and the new **last_name** (i.e Harper) was updated.
- Use the **MERGE** command to update the destination table as shown below.

```
Use CDC_STREAM;
MERGE into employees_destination as T
using (select *
```

```
from employee_stream
Where Not (metadata$action = 'DELETE' AND metadata$isupdate = TRUE)) AS S
ON T.id = s.id
when matched AND S.metadata$action = 'INSERT' AND S.metadata$isupdate
THEN
update set T.first_name = S.first_name, T.last_name = S.last_name
When matched
And S.metadata$action = 'DELETE' THEN DELETE
when not matched
And S.metadata$action = 'INSERT' THEN
INSERT (id,
first_name,
last_name)
VALUES (S.id,
S.first_name,
S.last_name);
```

Step 9: Run the following command to take a look at the destination table.

```
select * from employees_destination;
```

Here, you will be clearly able to observe the changes made to the source table updated in the destination table. That's it, that's how easy Snowflake Change Data Capture is.

For a better understanding, you can try out Snowflake Change Data Capture by deleting or updating other records in the source table. And, don't forget to use the **MERGE** command for the changes to reflect in the destination table.

Conclusion

Snowflake Change Data Capture has totally replaced the old ways of implementing CDC. Although Snowflake is already an industry-leading Cloud-Based Data Platform known for its speed and flexible warehousing options, Snowflake CDCs make it much worth it to use.

Snowflake Change Data Capture proves to be very helpful in cases where millions of records get transacted on a daily basis but you only want to update the modified ones. Doing a full load will eat up your resources and time, so just leverage the Snowflake Change Data Capture and use the **MERGE** command to update the destination.

However, hand-coding the CDC infrastructure can prove to be a tedious task without dedicated engineering resources.

What is Change Data Capture(CDC)?

A stream object records **data manipulation language (DML)** changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data. This process is referred to as **change data capture (CDC)**. An individual table stream tracks the changes made to rows in a source table. A **table stream** (also referred to as simply a “stream”) makes a “**change table**” available of what changed, at the row level, between two transactional points of time in a table.

What is Snowflake Streams?

A Stream adds changed data capture to Snowflake so you’re gonna have your source table and imagine the source table will typically live in your staging environment and when you put the stream on it to enable change data capture every time you insert data insert, update or delete data in your source table the stream just captures the changes so imagine you have you know many hundreds of millions of rows or terabytes of data in a large table and then you’re doing your daily loads you don’t have to reload your entire target table so by capturing just the change data capture we can use the merge statement to merge just the changes from source to target.

Why to use CDC?

1. Can load real-time data from transactional databases easily.
2. It doesn’t affect/harm source data/system.
3. It changes expectations for speed and flexibility of a data warehouse.
4. Old batch ETL uploads achieve the objective of moving the data to a target that has high-latency approaches that cannot support the continuous data pipelines and real-time operational decision-making.

How to use CDC(Capture Data Change)?

First of all either login to Snowflake WebBased UI or SnowSQL. Then follow below steps:

Create database CDC_Stream

```
create or replace database CDC_STREAM;  
Use CDC_STREAM;
```

Create a table to be the source

```
create or replace table members_source (  
id int,  
first_name varchar(255),  
last_name varchar(255) )
```

Results [Data Preview](#)

✓ [Query ID](#) [SQL](#) 117ms 1 rows

Row	status
1	Table MEMBERS_SOURCE successfully created.

Create a table to be the destination

```
create or replace table members_destination ( id int, first_name varchar(255), last_name varchar(255) );
```

Create a stream to track changes to date in the MEMBERS table

```
create or replace stream member_stream on table members_source;
```

Anytime we make changes to source table which is the members source it's going to be populated in the stream until some data management command comes and consumes it

Enter a couple records into source table

Add some record

```
insert into members_source values (1,'Wayne','Bell');  
insert into members_source values (2,'Anthony','Allen');
```

```

insert into members_source values (3,'Eric','Henderson');
insert into members_source values (4,'Jimmy','Smith');
insert into members_source values (5,'Diana','Wheeler');
insert into members_source values (6,'Karen','Hall');
insert into members_source values (7,'Philip','Rodriguez');
insert into members_source values (8,'Ashley','Bryant');
insert into members_source values (9,'Norma','Grant');
insert into members_source values (10,'Helen','Lewis');
insert into members_source values (11,'Larry','McCoy');
insert into members_source values (12,'Emily','Wood');
insert into members_source values (13,'Patrick','Alvarez');

```

Here's our source data

```
select * from members_source;
```

Results Data Preview Open History

✓ Query ID: SQL 123ms 13 rows

Row	ID	FIRST_NAME	LAST_NAME
1	1	Wayne	Bell
2	2	Anthony	Allen
3	3	Eric	Henderson
4	4	Jimmy	Smith

As you see destination it should be empty

```
select * from members_destination;
```

View the change log in the stream

```
select * from member_stream;
```

Results Data Preview Open History

✓ Query ID: SQL 138ms 13 rows

Row	ID	FIRST_NAME	LAST_NAME	METADATASACTION	METADATASISUPDATE	METADATASROWID
1	2	Anthony	Allen	INSERT	FALSE	43c22aeb4eea86e301bd519264c9015bae0b91d6
2	12	Emily	Wood	INSERT	FALSE	24c15cf4a10bc858d3fe2ad4408821cf6a58dd53
3	3	Eric	Henderson	INSERT	FALSE	78852fe96f3e4bc1b4cb89da45393c8b9d8a3772
4	6	Karen	Hall	INSERT	FALSE	607db31594aae07f543b7765e4c0451bd4215f1f
5	5	Diana	Wheeler	INSERT	FALSE	eee3f1714ee60b6ef0f31781e3e4d368b3d46c9a
6	10	Helen	Lewis	INSERT	FALSE	2323e0d23daa40b9c89fdbdb9a01d2378cf5daba

Here are the records and we had a couple of metadata fields some data action will say insert or delete, in this case we have inserts field only.

So, right now there are no records in our destination but they're sitting here in the stream and we're going to move it over with the merge statement

we'll have the merge State, we're gonna merge into destination everything that came from source table but from the stream.

```
Use CDC_STREAM;MERGE into members_destination as T
using (select *
from member_stream) AS S
ON T.id = s.id
when matched AND S.metadata$action = 'INSERT' AND S.metadata$isupdate
THEN
update set T.first_name = S.first_name, T.last_name = S.last_name
when not matched
And S.metadata$action = 'DELETE' THEN DELETE
when not matched
And S.metadata$action = 'INSERT' THEN
INSERT (id,
first_name,
last_name)
VALUES (S.id,
S.first_name,
S.last_name);
```

The screenshot shows a database query results interface. At the top, there is a code editor window containing the provided SQL script. Below the code editor is a toolbar with buttons for 'Results' (which is selected), 'Data Preview', 'Open History', and other options. The main area displays the execution results. It includes a summary bar at the top with 'Query ID', 'SQL', '600ms', and '1 rows'. Below this is a search bar labeled 'Filter result...'. The results table has four columns: 'Row', 'number of rows inserted', 'number of rows updated', and 'number of rows deleted'. A single row is shown with values: Row 1, number of rows inserted 13, number of rows updated 0, and number of rows deleted 0. There are also 'Copy' and 'Columns' buttons.

Row	number of rows inserted	number of rows updated	number of rows deleted
1	13	0	0

let's execute the destination table and now the data came by just like we wanted it to and then if we go and look at our stream there's not going to be anything in it because the merge consumed the stream so the change they capture the stream will continue to record all the change that happens in your source table.

View the results

```
select * from members_destination;
```

Results Data Preview

✓ Query ID SQL 128ms 13 rows

Filter result...

Row	ID	FIRST_NAME	LAST_NAME
1	4	Jimmy	Smith
2	9	Norma	Grant
3	2	Anthony	Allen
4	3	Eric	Henderson
5	1	Wayne	Bell
6	13	Patrick	Alvarez

We're gonna update the first record from Bell we're gonna say Wright or ID equals to 1

```
update members_source
```

```
set last_name='Wright'
```

```
where id=1;
```

Results Data Preview Open History

✓ Query ID SQL 263ms 1 rows

Filter result... Columns ▾

Row	number of rows updated	number of multi-joined rows updated
1	1	0

When we look at the stream it has two records insert and delete and you can see the insert has the

Wright value that's the new value we want the other column metadata is update

Results Data Preview

✓ Query ID SQL 213ms 2 rows

Filter result...

Row	ID	FIRST_NAME	LAST_NAME	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
1	1	Wayne	Wright	INSERT	TRUE	a13ec58d6198d8f486ac47dd2ea982c3fb425bc8
2	1	Wayne	Bell	DELETE	TRUE	a13ec58d6198d8f486ac47dd2ea982c3fb425bc8

Let's execute our merge statement as shown below

```
Use CDC_STREAM;MERGE into members_destination as T
using (select *
from member_stream
Where Not (metadata$action = 'DELETE' AND metadata$isupdate = TRUE)) AS S
ON T.id = s.id
when matched AND S.metadata$action = 'INSERT' AND S.metadata$isupdate
THEN
update set T.first_name = S.first_name, T.last_name = S.last_name
When matched
And S.metadata$action = 'DELETE' THEN DELETE
when not matched
And S.metadata$action = 'INSERT' THEN
INSERT (id,
first_name,
last_name)
VALUES (S.id,
S.first_name,
S.last_name);
```

Run merge command and let's take a look at destination table

View the results

```
select * from members_destination;
```

Let delete the some data from the source table i.e we are going to delete the **Anthony**
delete from members_source where id = 2;

Results		Data Preview
✓ Query ID		SQL
		221ms
1 rows		
<input type="text" value="Filter result..."/>		 Copy
Row	number of rows deleted..	
1	1	

View the change log in the stream

```
select * from member_stream;
```

Results Data Preview Open

✓ Query ID SQL 239ms 1 rows

Row	ID	FIRST_NAME	LAST_NAME	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
1	2	Anthony	Allen	DELETE	FALSE	43c22aeb4ee86e301bd519264c9015bae0b91d6

Let's execute our merge statement as shown below

```
Use CDC_STREAM;MERGE into members_destination as T
using (select *
from member_stream
Where Not (metadata$action = 'DELETE' AND metadata$isupdate = TRUE)) AS S
ON T.id = s.id
when matched AND S.metadata$action = 'INSERT' AND S.metadata$isupdate
THEN
update set T.first_name = S.first_name, T.last_name = S.last_name
When matched
And S.metadata$action = 'DELETE' THEN DELETE
when not matched
And S.metadata$action = 'INSERT' THEN
INSERT (id,
first_name,
last_name)
VALUES (S.id,
S.first_name,
S.last_name);
```

Run merge command and let's take a look at destination table

View the results

```
select * from members_destination;
```

Conclusion

Change Data Capture in Snowflake using Streams, changes the whole scenario. It just totally replaced the old ways implementing CDC. Snowflake is quite popular for its speed and flexible warehousing options. But CDCs make it much worth it to use Snowflake as compared to others.

User-defined functions (UDFs) let you extend the system to perform operations that are not available through the built-in, system-defined functions provided by Snowflake. Snowflake currently supports the following languages for writing UDFs:

SQL: A SQL UDF evaluates an arbitrary SQL expression and returns either scalar or tabular results.

JavaScript: A JavaScript UDF lets you use the JavaScript programming language to manipulate data and return either scalar or tabular results.

Java: A Java UDF lets you use the Java programming language to manipulate data and return scalar results.

	UDF's	
	Scalar	Tabular
Definition	Returns one output row for each input row. The returned row consists of a single column/value	Returns zero, one, or multiple rows for each input row and return clause contains the TABLE keyword and specifies the names and data types of the columns in the table results
Which language supports ?	SQL, JavaScript, Java	SQL, JavaScript

Introduction to SQL UDFs:

A SQL UDF evaluates an arbitrary SQL expression and returns the result(s) of the expression. The function definition can be a SQL expression that returns either a scalar (i.e. single) value or, if defined as a table function, a set of rows.

We start with SQL syntax,

```
create or replace function function_name(argument1 type, argument2 type,...)
  returns type
as
$$
  function_definition [SQL statements]
$$
;
```

Example 1: (Scalar)

Write an UDF to add ' {@@' sign beginning and end of First name and Last name.

Input:

First Name: Tata

Last Name: Harrier

Expected Output:

@@Tata Harrier@@

Script:

```
CREATE OR REPLACE FUNCTION String_Concat(Firstname varchar(100),Lastname varchar(100))
  returns VARCHAR
as
$$
  SELECT ' {@@'||Firstname||' |||Lastname||'@@' FROM DUAL
$$
;
```

```
1 CREATE OR REPLACE FUNCTION String_Concat(Firstname varchar(100),Lastname varchar(100))
2   returns VARCHAR
3   as
4   $$|
5     SELECT '@@'||Firstname||' '||Lastname||'@@' FROM DUAL
6   $$|
7 ;
```

Results Data Preview

✓ Query ID SQL 148ms 1 rows

Filter result...



Copy

Row	status
1	Function STRING_CONCAT successfully created.

Function for String Concatenation

Output:

```
SELECT String_Concat('TATA','HARRIER') as FULLNAME;
```

Results Data Preview

✓ Query ID SQL 27ms 1 rows

Filter result...



Copy

Row	FULLNAME
1	@@TATA HARRIER@@

Example 2: (Scalar)

How to find biggest of two numbers ?

Input:

100,500

Expected Output:

500

Script:

```
CREATE OR REPLACE FUNCTION Biggest_of_Two_Numbers (Num1 float, Num2 float)
```

```
returns float
```

```
as
```

```
$$
```

```
SELECT CASE WHEN Num1 > Num2 THEN Num1 ELSE Num2 END
```

```
$$
```

```
;
```

```
12 CREATE OR REPLACE FUNCTION Biggest_of_Two_Numbers (Num1 float, Num2 float)
13     returns float
14     as
15     $$|
16         SELECT CASE WHEN Num1 > Num2 THEN Num1 ELSE Num2 END
17     $$|
18     ;
```

Results Data Preview

✓ Query_ID SQL 44ms 1 rows

Filter result...



Copy

Row	status ↓
1	Function BIGGEST_OF_TWO_NUMBERS successfully created.

Output:

```
SELECT Biggest_of_Two_Numbers (100,500) as Biggest_Number;
```

The screenshot shows a database query interface. At the top, there are two code snippets: one for line 20 and one for line 21. Line 21 contains the SQL query: `SELECT Biggest_of_Two_Numbers (100,500) as Biggest_Number;`. Below the code, there are two tabs: "Results" (which is selected) and "Data Preview". Under the "Results" tab, there is a green checkmark icon, a "Query ID" link, an "SQL" link, a performance metric "308ms" with a progress bar, and "1 rows". Below this, there is a search bar labeled "Filter result...", a download icon, and a "Copy" button. The main area displays a table with one row. The table has two columns: "Row" and "BIGGEST_NUMBER". The first row contains the value "1" in the "Row" column and "500" in the "BIGGEST_NUMBER" column.

Row	BIGGEST_NUMBER
1	500

Note: When using a query expression in a SQL UDF, do not include a semicolon within the UDF body to terminate the query expression.

You can include only one query expression. The expression can include UNION [ALL].

Tabular SQL UDFs (UDTFs)

Snowflake supports SQL UDFs that return a set of rows, consisting of 0, 1, or multiple rows, each of which has 1 or more columns. Such UDFs are called *tabular UDFs*, *table UDFs*, or, most frequently, *UDTFs* (user-defined table functions).

Syntax:

```
CREATE OR REPLACE FUNCTION <name> ( [ <arguments> ] )
RETURNS TABLE ( <output_col_name><output_col_type> [, <output_col_name><output_col_type> ... ] )
AS '<sql_expression>'
```

Example 1: (Tabular)

Fetch TRIPDURATION, BIKEID, BIRTH_YEAR from **Trips** table by passing parameters START_STATION_ID, END_STATION_ID AND TRIPDURATION.

Below tables considered for this example,

Table Name: Trips	
Column_Name	Datatype
TRIPDURATION	NUMBER(38,0)
STARTTIME	TIMESTAMP_NTZ(9)
STOPTIME	TIMESTAMP_NTZ(9)
START_STATION_ID	NUMBER(38,0)
START_STATION_NAME	VARCHAR(16777216)
START_STATION_LATITUDE	FLOAT
START_STATION_LONGITUDE	FLOAT
END_STATION_ID	NUMBER(38,0)
END_STATION_NAME	VARCHAR(16777216)
END_STATION_LATITUDE	FLOAT
END_STATION_LONGITUDE	FLOAT
BIKEID	NUMBER(38,0)
MEMBERSHIP_TYPE	VARCHAR(16777216)
USERTYPE	VARCHAR(16777216)
BIRTH_YEAR	NUMBER(38,0)
GENDER	NUMBER(38,0)

Script:

```
CREATE OR REPLACE FUNCTION GetTripsDetail (START_STATION_IN number, END_STATION_IN
number, TRIPDURATION_IN number)
returns table (TRIPDURATION number, BIKEID number, BIRTH_YEAR number)
as
$$
SELECT TRIPDURATION,BIKEID,BIRTH_YEAR FROM TRIPS
WHERE
TRIPS.START_STATION_ID = START_STATION_IN AND
```

```

TRIPS.END_STATION_ID = END_STATION_IN AND
TRIPS.TRIPDURATION < TRIPDURATION_IN
$$
;

```

```

32 CREATE OR REPLACE FUNCTION GetTripsDetail (START_STATION_IN number, END_STATION_IN number, TRIPDURATION_IN number)
33 returns table (TRIPDURATION number, BIKEID number, BIRTH_YEAR number)
34 as
35 $$|
36 SELECT TRIPDURATION,BIKEID,BIRTH_YEAR FROM TRIPS
37 WHERE
38 TRIPS.START_STATION_ID = START_STATION_IN AND
39 TRIPS.END_STATION_ID = END_STATION_IN AND
40 TRIPS.TRIPDURATION < TRIPDURATION_IN
41 $$|
42 ;

```

Results Data Preview

✓ Query ID SQL 50ms 1 rows

Filter result... Copy

Row	status
1	Function GETTRIPSDetail successfully created.

Output:

```

SELECT TRIPDURATION,BIKEID,BIRTH_YEAR
from table(GetTripsDetail (537,435,200));

```

```

44 SELECT TRIPDURATION,BIKEID,BIRTH_YEAR
45 from table(GetTripsDetail (537,435,200));

```

Results Data Preview

✓ Query ID SQL 388ms 8 rows

Filter result... Copy Columns ▾

Row	TRIPDURATION	BIKEID	BIRTH_YEAR
1	182	26043	1997
2	186	16708	1985
3	197	30837	1977
4	192	17130	1997
5	181	17087	1981

In next article we will see more about JavaScript UDFs, Java UDFs & Secure UDFs

Snowflake - UDF

- User defined functions let you extend operations that are not available through system defined functions in Snowflake .
- UDF's are reusable code which can be written in :
 - Sql
 - Javascript
 - Java
 - Python (External Functions)
- SQL UDF's can execute re-usable sql queries whereas Javascript UDF's give you more flexibility when it comes to branching or looping statements .
- UDFs are of 2 types :
 - Scalar - A scalar function returns one output row .
 - Tabular - Also called a table function, returns zero, one, or multiple rows for each input row

Snowflake - Query Filter Execution

- Load First , filter later :
 - Read all rows from the table into memory
 - Scan the rows in memory, filtering out any rows that do not match the filter
 - Select the columns that remain in the memory (select clause)
- Early Filtering (Pushdown)
 - Do not load/read the records that do not match the filter conditions from the WHERE clause
 - Scan the remaining rows in memory and select the columns from the select clauses

Pushdown Benefits :

- Pushdown improves performance by filtering out unneeded rows as early as possible during query processing.
- Pushdown can also reduce memory consumption. However, pushdown can allow confidential data to be exposed indirectly.

Snowflake Pushdown

Pushdown is an alternative computation method for running a DQ job, where all of the job's processing is submitted to a SQL data warehouse, such as Snowflake. Snowflake Pushdown jobs generate SQL queries to offload the compute to the data source, reducing the amount of data transfer and Spark computation of the DQ Job.

By running a Snowflake Pushdown job, you can:

- Reduce latency.
- Eliminate dependencies on Spark compute to run Collibra Data Quality, and increase processing speeds.
- Eliminate the egress costs for running DQ Jobs against large data sets.
- Auto-scale based on your processing requirements.

- Successfully run the [Pushdown setup script](#).
- [Enable Pushdown](#) from the Collibra DQ UI.

Pushdown vs. Pull Up

Collibra DQ Pull Up is a DQ Job without pushdown, where all of the processing is executed inside the Apache Spark compute engine. Source data is stored inside a database, where Spark reads it out, and the parameters you set when you select a scope, define a range, and add build layers, are partitioned and sorted. The results of the profile job are then recorded in the DQ Metastore. Depending on the size of your data set and the number of DQ checks performed, this process can greatly slow run times because Spark has its own compute resources, such as memory and CPUs. Pull up has limited support for profiling but you can't run it without setting up Spark.

With Snowflake Pushdown, the Collibra DQ Agent, which creates the Apache Spark DQ Job, is no longer needed. No agent is required to submit a Snowflake Pushdown job because all of the processing is sent directly to Snowflake. Therefore, Agent ID is always set to 0 for Snowflake Pushdown jobs.

With Snowflake, you can also scale your compute needs based on the specific requirements of your DQ Job. This is because Snowflake's architecture features auto-scaling, which allows you to automatically scale up, or *burst*, to 64 or 128 nodes when you require greater processing needs. Snowflake also automatically scales down when your DQ Job does not require robust processing. With auto-scaling, the processing of your data is enhanced, improving runtime performance and removing the egress costs of reading large amounts of data.

Snowflake – UDF

- SQL UDFs should be defined as secure when they are specifically designated for data privacy
- Secure UDFs should not be used for UDFs that are created for query re-usability or querying convenience
- Snowflakes query optimizer bypasses the optimizations(pushdowns) when it comes to Secure UDFs
- Secure UDFs have a relatively poor query performance compared to Regular UDFs.

Many organizations would like to securely link and join their data with data generated by their partners, customers, and industry peers, but they have concerns about protecting personally identifiable information (PII), protected health information (PHI), and other forms of fine-grained data.

Snowflake recently launched a feature called [Secure User-Defined Functions](#) (Secure UDFs). Secure UDFs allow Snowflake users to link, join, and analyze fine-grained data with data from other Snowflake users while preventing the other parties from viewing or exporting the raw data.

This form of secure linking and joining moves towards the vision of enabling a global data economy, where the world's data assets are leveraged together with artificial intelligence (AI) and business intelligence (BI) technologies to improve corporate and industry performance, increase transparency, and address common problems.

SECURE VIEWS AND THEIR LIMITATIONS

Today, most [data sharing](#) in Snowflake uses secure views. Secure views are a great way for a data owner to grant other Snowflake users secure access to select subsets of their data. Secure views are effective for enforcing cell-level security in multi-tenant situations. This includes software-as-a-service (SaaS) providers granting access to each of their customers, while allowing each customer to see only their specific rows of data from each table. However, there is nothing preventing another user from running a `SELECT *` query against the secure view and then exporting all the data that's visible to them.

In many situations, allowing a data consumer to see and export the raw data is completely acceptable. However, in other situations, such as when monetizing data, the most valuable analyses are often run against low-level and raw data, and allowing a data consumer to export the raw data is not desirable. Furthermore, when PII and PHI are involved, privacy policies and government regulations often do not permit providing data access to other parties.

THE POWER OF SECURE UDFS

Secure UDFs are small pieces of SQL or JavaScript code that securely operate against raw data, but provide only a constrained set of outputs in response to specific inputs. For example, imagine a retailer that wants to allow its suppliers to see which items from other suppliers are commonly sold together with theirs. This is known as *market basket analysis*.

Using the TCP-DS sample data set that's available to all users from the **Shares** tab within Snowflake, we can run the following SQL commands to create a test data set and perform a market basket analysis:

```
create database if not exists UDF_DEMO;

create schema if not exists UDF_DEMO.PUBLIC;

create or replace table udf_demo.public.sales as

(select * from snowflake_sample_data.TPCDS_SF10TCL.store_sales

sample block (1));
```

```

select 6139 as input_item, ss_item_sk as basket_item,
       count(distinct ss_ticket_number) baskets
  from udf_demo.public.sales
 where ss_ticket_number in (select ss_ticket_number from udf_demo.public.sales where ss_item_sk = 6139)
 group by ss_item_sk
 order by 3 desc, 2;

```

INPUT_ITEM BASKET_ITEM BASKETS

<i>INPUT_ITEM</i>	<i>BASKET_ITEM</i>	<i>BASKETS</i>
6139	6139	1048
6139	7115	405
6139	7114	189
6139	9257	128
6139	9256	102
6139	7116	95
6139	9258	40
6139	23492	37
6139	40008	10

This example returns the items that sold together with item #6139. This example outputs only aggregated data, which is the number of times various other products are sold together, in the same transaction, with item #6139. This SQL statement needs to operate across all of the raw data to find the right subset of transactions. To enable this type of analysis while preventing the user who is performing the analysis from seeing the raw data, we wrap this SQL statement in a secure UDF and add an input parameter to specify the item number we are selecting for market basket analysis, as follows:

create or replace secure function

```
UDF_DEMO.PUBLIC.get_market_basket(input_item_sk number(38))

returns table (input_item NUMBER(38,0), basket_item_sk NUMBER(38,0),

num_baskets NUMBER(38,0))

as

'select input_item_sk, ss_item_sk basket_item, count(distinct

ss_ticket_number) baskets

from udf_demo.public.sales

where ss_ticket_number in (select ss_ticket_number from udf_demo.public.sales where ss_item_sk

= input_item_sk)

group by ss_item_sk

order by 3 desc, 2';

select * from table(UDF_DEMO.PUBLIC.get_market_basket(6139));

<returns same results as previous example>
```

We can then call this function and specify any item number as an input, and we will get the same results we received when running the SQL statement directly. Now, we can grant a specified user access to this function while preventing the user from accessing the underlying transactional data.

HOW TO SHARE SECURE UDFS

To share a secure UDF, we can then grant usage rights on the secure UDF to a Snowflake share. This gives other specified Snowflake accounts the ability to run the secure UDF, but does not grant any access rights to the data in the underlying tables.

```
use database UDF_DEMO;

create share if not exists UDF_DEMO_SHARE;

grant usage on database UDF_DEMO to share UDF_DEMO_SHARE;

grant usage on schema UDF_DEMO.PUBLIC to share UDF_DEMO_SHARE;
```

```
grant usage on function UDF_DEMO.PUBLIC.get_market_basket(number) to share  
UDF_DEMO_SHARE;
```

```
alter share UDF_DEMO_SHARE add accounts=<consumer account id>;
```

If we then log in to the other Snowflake account, we can run the secure UDF from the share using the second account's virtual warehouse. However, from the second account, we cannot select any data from the underlying tables, determine anything about the names or structures of the underlying tables, or see the code behind the secure UDF.

```
use role accountadmin;
```

```
create database UDF_TEST from share <provider_account>.UDF_DEMO_SHARE;
```

```
grant imported privileges on database UDF_TEST to role PUBLIC;
```

```
use database UDF_TEST;
```

```
select * from table(UDF_TEST.PUBLIC.get_market_basket(6139));
```

INPUT_ITEM_BASKET_ITEM_SK NUM_BASKETS

6139 6139 1048

6139 7115 405

6139 7114 189

6139 9257 128

6139 9256 102

6139 7116 95

6139 9258 40

6139 23492 37

6139 40008 10

```
describe share jtest2.UDF_DEMO_SHARE;

kind    name

DATABASE UDF_TEST

SCHEMA   UDF_TEST.PUBLIC

FUNCTION UDF_TEST.PUBLIC."GET_MARKET_BASKET(...)"
```

The secure UDF is essentially using the data access rights of its creator, but allowing itself to be run by another Snowflake account that has access rights to run it. With Snowflake Data Sharing, the compute processing for secure UDFs runs in the context of, and is paid for by, the data consumer using the consumer's virtual warehouse, against the function provider's single encrypted copy of the underlying data.

This ability to share a secure UDF enables a myriad of secure data sharing and data monetization use cases, including the ability to share raw and aggregated data and powerful analytical functions, while also protecting the secure UDF's code. It also prevents other parties from directly viewing or exporting the underlying encrypted data.

SHARED JAVASCRIPT SECURE UDFS

The previous example is a SQL secure UDF. JavaScript secure UDFs, which are also shareable, operate in a slightly different manner. A shared JavaScript secure UDF is incapable of running any queries, and it cannot perform any other form of external I/O. However, it can perform standard JavaScript operations (without any external libraries), coded by the function's provider, against data passed in by the function's consumer. A SQL secure UDF, on the other hand, can run queries, but only against the provider's data.

JavaScript secure UDFs are useful for data cleansing, address matching, and other data manipulation operations. In the next installment of this multi-part blog post, we'll include an example of a JavaScript secure UDF for matching up data across two accounts.

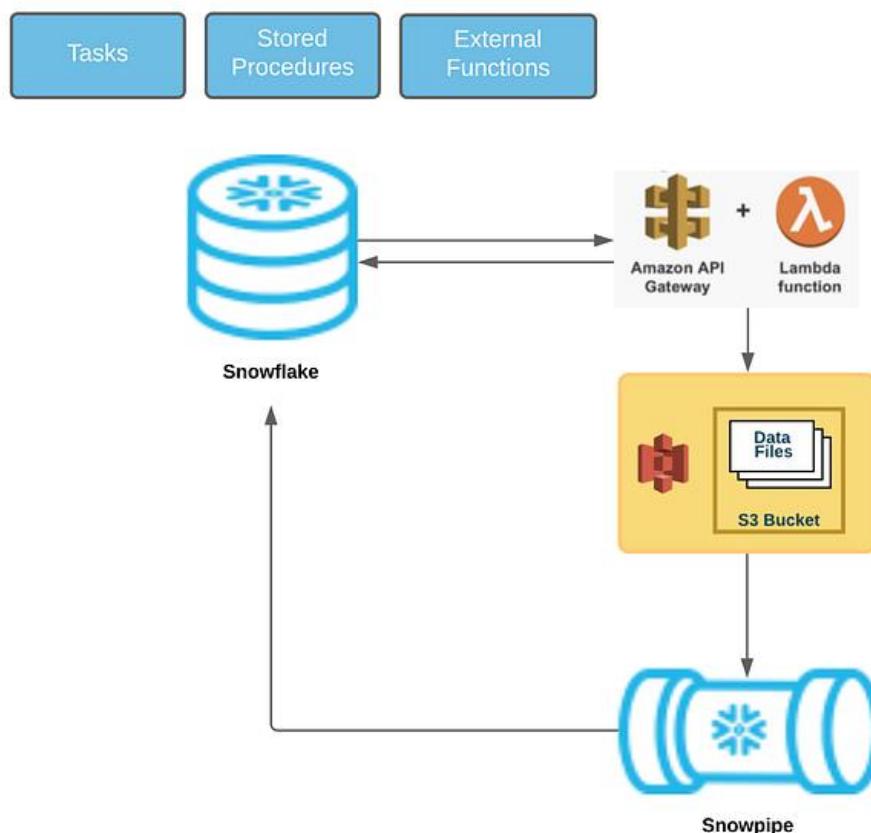
As a Data Engineer on Snowflake's IT team, it's my team's job to build and monitor the ingest pipelines from various external sources and write back the enriched data from Snowflake to source systems. One Snowflake feature that is particularly useful for our work is external functions.

The external functions feature in Snowflake enables the ability to call code that is executed outside Snowflake (remote service). An external function is like a regular UDF, but it calls cloud serverless compute services such as AWS Lambda. It can pass parameters to the remote service and accepts the data returned by remote service, which can be consumed in Snowflake.

External functions open new ways to implement various workflows for *Ingestion*, *Reverse ETL*, and *Alerting* in Snowflake.

Data Ingestion

Below is the high-level diagram for API Data Ingestion Architecture using external functions.



We use Tasks, Stored Procedures, and External Functions to implement External API Data ingestions.

Let's walk through how we use each of these components:

1. **Tasks** to schedule the job, which triggers a Stored Procedure call if there is a pagination/ control flow logic. If not, the task will directly call the external function in a SQL Query.

2. **Stored Procedure** to implement pagination/ control flow logic and call external function, consume the JSON response from external function, flatten it and ingest into target tables.
3. **External function** takes API parameters like Endpoint URL, Post Data, Query Parameters and passes them to the remote service, which makes the actual API call to retrieve the data and returns the response to Snowflake.
4. In some cases, API response is not directly consumable in Snow SQL, for example, CSV files. The remote service can upload the files to S3, and Snowpipe will take it from there.

Slack Ingestion Use case

Using external functions, the Snowflake IT team ingests Slack user data. This data allows us to send personal notifications to employees.

By using external functions to ingest this data we minimize the number of moving pieces and don't have to manage any infrastructure. Additionally, we don't have any additional costs from or dependency on external orchestrators for building the ingestion pipelines.

Let's take a look at how we do this.

External Function

Create an external function with required parameters in the headers section:

```
CREATE OR REPLACE SECURE EXTERNAL FUNCTION slack_get_api_call(url string) RETURNS VARIANT
RETURNS NULL ON NULL INPUT
VOLATILE
MAX_BATCH_ROWS=1
API_INTEGRATION=sfc_datasci_api_gatewayHEADERS=
(
'api-url'='{0}'
'api-method'='get'
'secrets'= 'Slack_API_Key'
)as 'https://*****.execute-api.us-west-2.amazonaws.com/it-data-ingest-lambda-stage/';
```

Retrieve API Response In SQL

Calling the external function in the SQL query to visualize the JSON response:

```
select "VALUE" output_json from (
select slack_get_api_call('https://slack.com/api/users.list?token={{Slack_API_Key}}')[0]:members val) s,
lateral flatten(input => val)
```

Stored Procedure

We need to paginate through the API response to fetch the complete dataset from the API. We use stored procedures to build the pagination logic and flatten the data from raw JSON to ingest into the target table:

```
CREATE OR REPLACE PROCEDURE load_slack_users()
RETURNS VARIANT
LANGUAGE JAVASCRIPT
AS
$$var next_url = 'https://slack.com/api/users.list?token={{Slack_API_Key}}';
var insert_snippet;var del_cmd = "truncate table slack_users";
var del_stmt = snowflake.createStatement(
{
sqlText: del_cmd
},
);
del_stmt.execute();while (next_url!=null)
{
var cmd = "select return[0]:members members, return[0]:response_metadata:next_cursor from(select
slack_get_api_call(?) return) s";var stmt = snowflake.createStatement({sqlText: cmd,binds:[next_url]});var
result1 = stmt.execute();result1.next();var next_cursor = result1.getColumnValue(2);
insert_snippet = result1.getColumnValue(1);
var snippet = JSON.stringify(insert_snippet);var stmt_insert = snowflake.createStatement({
sqlText: `insert into slack_users
select $1:id::string slack_id, $1:name::string slack_user_name,$1:profile:display_name::string
display_name, $1:profile:email::string email, $1:deleted::boolean deleted_user, $1:is_admin::boolean
is_admin, $1:is_app_user::boolean is_app_user, $1:is_bot::boolean is_bot, $1:is_owner::boolean
is_owner, $1:is_primary_owner::boolean is_primary_owner, $1:is_restricted::boolean is_restricted,
$1:is_ultra_restricted::boolean is_ultra_restricted, $1:profile:display_name_normalized::string
display_name_normalized, $1:profile:first_name::string first_name, $1:profile:last_name::string
last_name, $1:profile:real_name::string real_name, $1:profile:real_name_normalized::string
real_name_normalized`});stmt_insert.execute();
next_url = next_cursor;
}
}$$;
```

```
real_name_normalized, $1:profile:phone::string phone, $1:profile:team::string team, $1:profile:title::string title, $1:team_id::string team_id, $1:tz::string timezone,$1:enterprise_user:teams::variant enterprise_user_teamsfrom (select VALUE output_json from (select parse_json(?) val) s, lateral flatten(input => val )) S`,binds:[snippet]});stmt_insert.execute();next_url = (next_cursor == " ? null: 'https://slack.com/api/users.list?token={{Slack_API_Key}}&cursor='+next_cursor)}  
$$;
```

Task

Schedule a task to refresh the slack_users table:

```
create or replace task slack_user_refresh  
WAREHOUSE = 'SNOWHOUSE'  
SCHEDULE = 'USING CRON 0 5 * * * America/Los_Angeles'  
ascall load_slack_users();
```

In the above example, the scheduled task runs every day at 5 AM and reloads the Slack_users table with the latest data from the Slack API.

Alerting

Our team also leverages external functions for alerting. For instance, we create incidents for ServiceNow when there is a failure in our data pipeline.

Alerting with external functions is very similar to the above example. Let's walk through what creating a ServiceNow alert looks like.

External Function

```
CREATE OR REPLACE SECURE EXTERNAL FUNCTION it_servicenow_send(url string,post_data string,  
auth_headers string, secrets string)  
RETURNS VARIANT  
VOLATILE  
MAX_BATCH_ROWS=1  
API_INTEGRATION=sfc_datasci_api_gatewayHEADERS=(  
'api-url'='{0}'  
'api-method'='POST'  
'headers'='{2}'  
'api-post-data' = '{1}'
```

```

'secrets' = '{3}'
'api-post-data-string' ='true'
)
as 'https://*****.execute-api.us-west-2.amazonaws.com/it-data-ingest-lambda-stage/';

```

Below is an example call to the external function creating a problem task in ServiceNow:

```

INSERT INTO it_snowdesk_problem_alert_logSELECT CURRENT_DATE() alert_date,
it_servicenow_send('https://****.snowflake.com/api/now/table/u_snowflake_problem',To_varchar(Obj
ct_construct('u_category', u_category,'u_subcategory',u_subcategory,
'u_problem_statement', u_problem_statement,
'u_impact', u_impact,'u_urgency', u_urgency,
'u_assignment_group', u_assignment_group,
'u_configuration_item', cmdb_ci,
'u_description', u_description, 'u_alert_type',
'SNOWDESK.01','u_contact_type','Integration'
)),Concat({"Authorization": "Basic {{'svc_snowdesk_sec' , '}}", "Content-Type":"application/json"}) from
snowdesk_problem_alert_source

```

Reverse ETL with External Functions

Our team uses Snowflake as the hub for all relevant data. Within Snowflake we have a unified view of all our application data, allowing our team to more effectively manage user access and support our employees. After we've enriched data within Snowflake, we often need to write data back to SaaS applications, ranging from Okta to ServiceNow. This process is called *reverse ETL*.

We use external functions to write back the data directly from Snowflake to relevant applications.

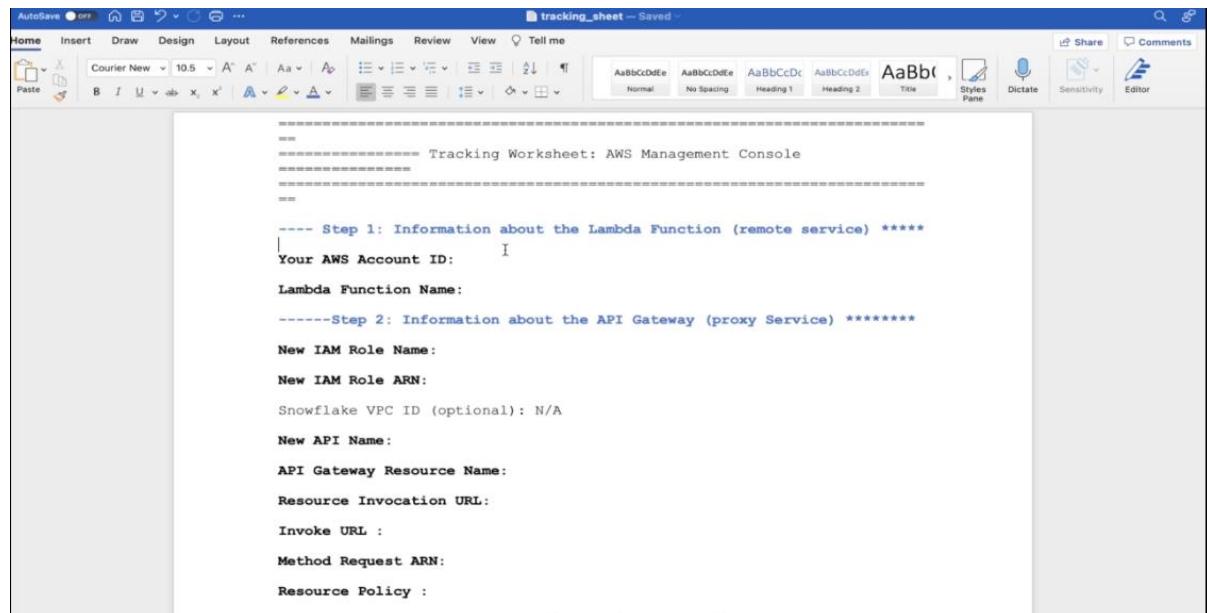
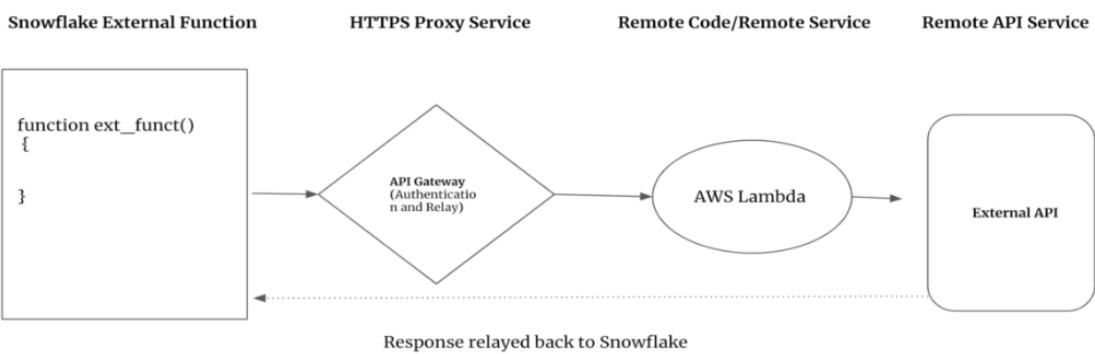
Enriching & Aligning the Application Layer through Snowflake



Snowflake - External Functions

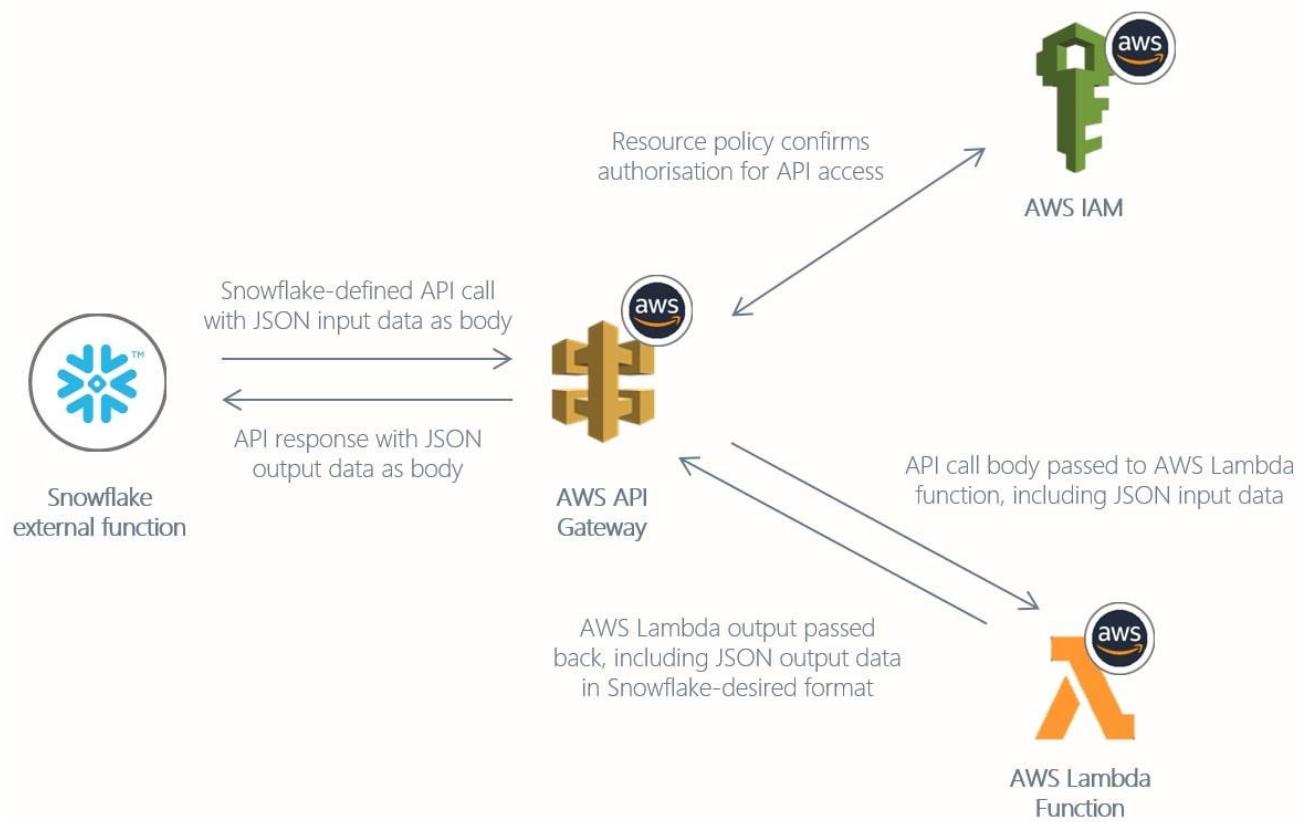
- An external function calls code that is executed outside Snowflake
- The remotely executed code is known as a remote service
- Information sent to a remote service is usually relayed through a proxy service
- An external function is a type of UDF. Unlike other UDFs, an external function does not contain its own code; instead, the external function calls code that is stored and executed outside Snowflake

Snowflake - External Function



Integration Between AWS Lambda and Snowflake

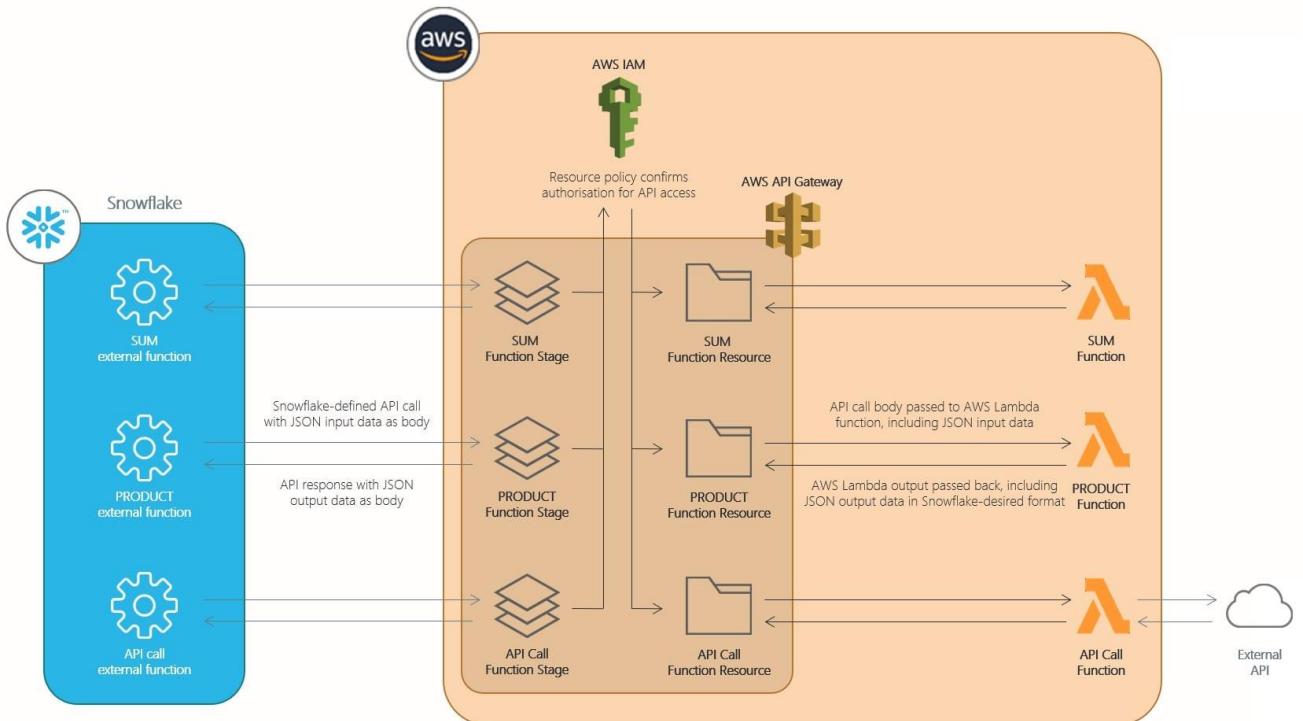
Before we can begin on our more complex flood warning measurements example, or even our two simpler examples, let us first consider the general data flow for an external function being called from Snowflake:



As we can see in this diagram, there are four key components to a Snowflake external function leveraging AWS Lambda:

1. A Snowflake external function object
2. An AWS API gateway which facilitates the integration by exposing the Lambda function to Snowflake (API stands for Application Programming Interface)
3. Authorised AWS IAM roles – There are actually two here: one account role to access the API gateway from Snowflake and one service role to execute the Lambda function. (IAM stands for Identity and Access Management)
4. An AWS Lambda function

This is a high-level diagram for a single external function setup. Our goal today is to set up three external functions, which makes our diagram more complex:



To set up our examples as simply as possible, we are going to complete the following steps:

1. Create an AWS IAM service role to execute all of the Lambda functions
2. Create AWS Lambda functions for our examples
3. Create an AWS API gateway to expose the Lambda functions to Snowflake
4. Create an AWS IAM role to access the API gateway and configure it to access both the API gateway and a Snowflake API integration object
5. Create and test some external function objects in Snowflake

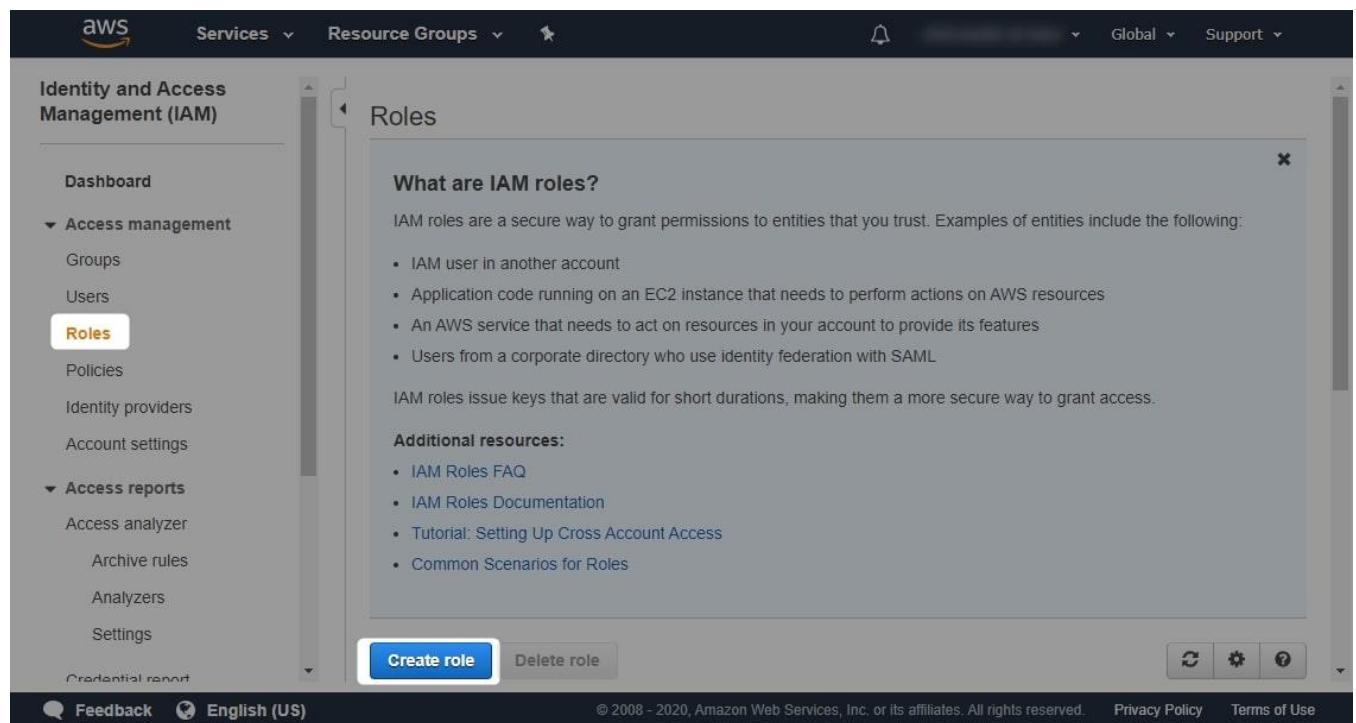
I personally believe the setup is simplest when following this order, as it results in the least amount of back-and-forth whilst allowing us to set up all of our functions under a single service role, account role and API gateway. If you are more experienced with AWS, you are more than welcome to play around with this setup to make it fit your own requirements.

Step-by-Step Guide to Setting up Our Two Simple Example Functions

We will begin by setting up and configuring our two simple example functions: **SUM** and **PRODUCT**. Once we have these set up and working, adding new functions becomes much simpler and allows us to focus on the complexity of the function rather than the complexity of the underlying architecture.

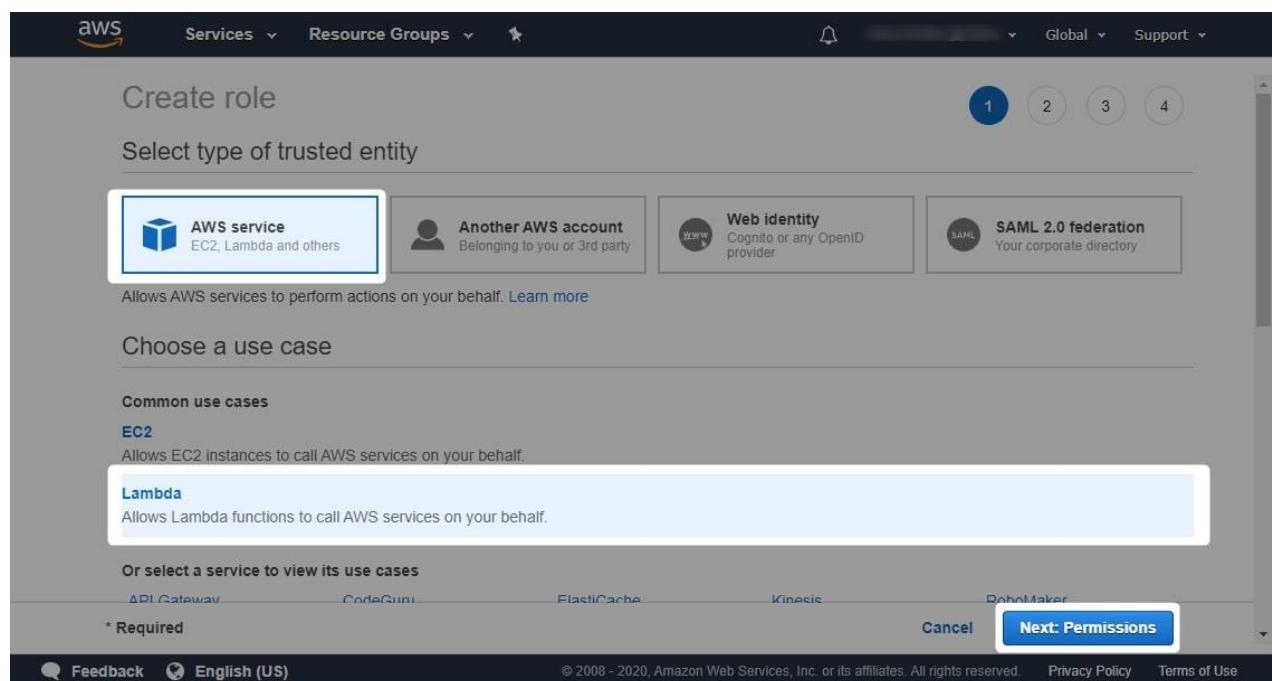
1. Configuring Your AWS IAM Service Role to Execute Each Lambda Function

Navigate to the IAM area in AWS, and select Roles then Create Role:



The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, there's a navigation sidebar with options like Dashboard, Access management, Roles (which is selected and highlighted in orange), Policies, Identity providers, Account settings, and more. The main content area is titled 'Roles' and contains a section titled 'What are IAM roles?'. It explains that IAM roles are a secure way to grant permissions to entities that you trust. Below this, there's a list of additional resources related to IAM roles. At the bottom of the main content area, there are two buttons: 'Create role' (highlighted with a blue box) and 'Delete role'. The footer of the page includes links for Feedback, English (US), and various AWS services.

In this section, navigate to AWS service, select Lambda and Next: Permissions:



The screenshot shows the 'Create role' wizard, Step 1: 'Select type of trusted entity'. It lists four options: 'AWS service' (selected and highlighted with a blue border), 'Another AWS account', 'Web identity', and 'SAML 2.0 federation'. Below each option is a brief description. The 'AWS service' option is described as allowing AWS services to perform actions on your behalf. The 'Lambda' option is described as allowing Lambda functions to call AWS services on your behalf. At the bottom of the screen, there are sections for 'Common use cases' (with 'EC2' and 'Lambda' listed) and 'Or select a service to view its use cases' (with options like API Gateway, CodeGuru, ElastiCache, Kinesis, and RoboMaker). A note at the bottom says '* Required'. On the right side, there are buttons for 'Cancel' and 'Next: Permissions' (highlighted with a blue box).

The following through menus are not required for this example, so click **Next** a few times until you reach the **Review** area. Once here, enter an appropriate name and description for your service role then click **Create Role**:

The screenshot shows the 'Review' step of the AWS IAM 'Create New Role' wizard. The 'Role name*' field contains 'snowflake-lambda-service-role'. The 'Role description' field contains the text: 'Allows Lambda functions to call AWS services on your behalf. Used for Lambda functions which act as external functions to Snowflake.' Below the description is a note: 'Maximum 1000 characters. Use alphanumeric and '+,-,@,_' characters.' Under 'Trusted entities', it lists 'AWS service: lambda.amazonaws.com'. Under 'Policies', it says 'Policies not attached'. Under 'Permissions boundary', it says 'Permissions boundary is not set'. A note at the bottom left says 'No tags were added.' At the bottom right are buttons for 'Cancel', 'Previous', and a prominent blue 'Create role' button.

Take note of the name of your service role. In fact, it is good practice to keep a file tracking all the various IDs, ARNs and other elements used throughout this process, as I will start doing now:

Lambda Service Role: snowflake-lambda-service-role

2. Configuring Your AWS Lambda Functions

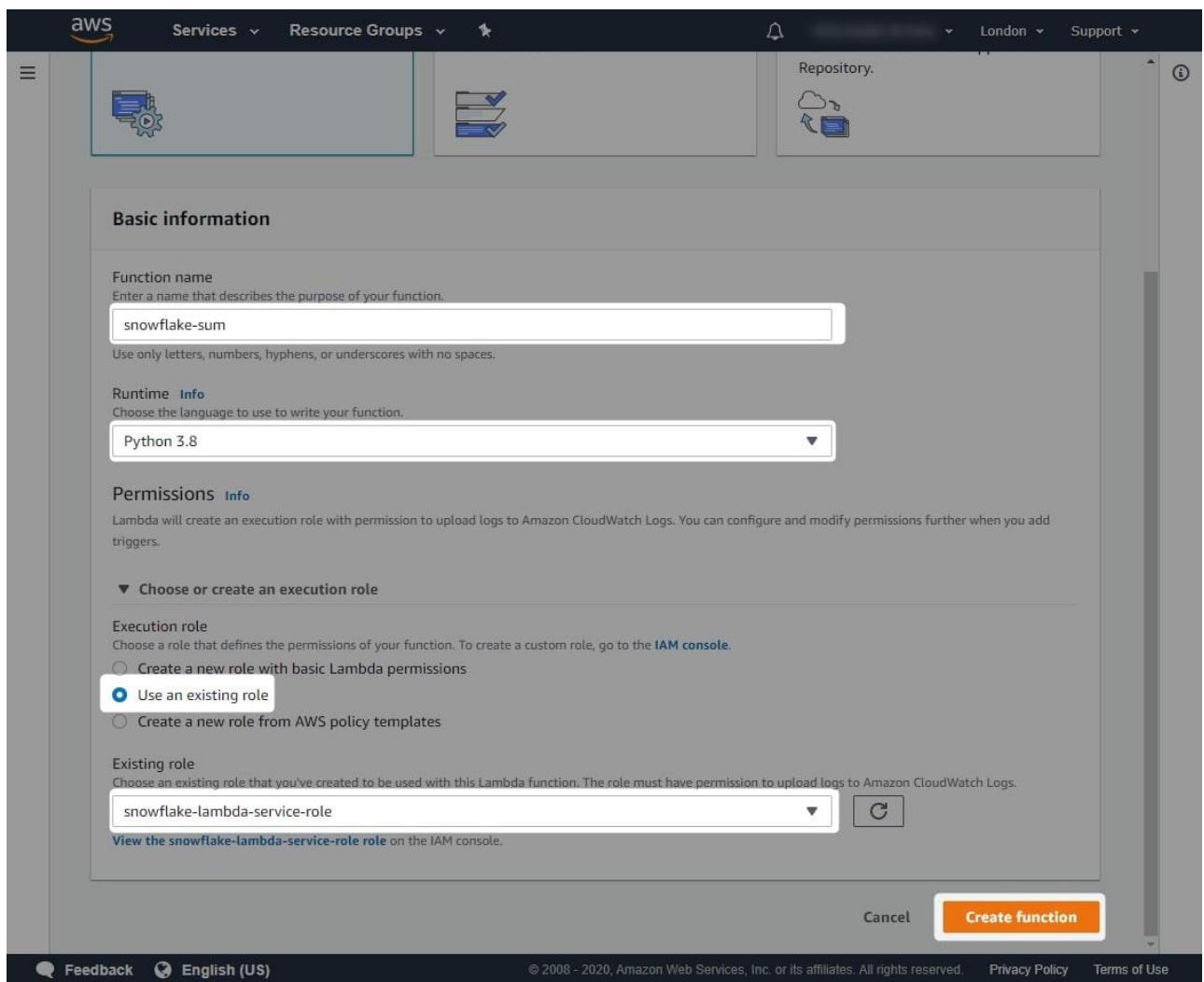
We will start by creating our simple SUM function. Navigate to the Lambda area in AWS and select Create Function:

The screenshot shows the AWS Lambda 'Functions' page. On the left, there's a sidebar with 'AWS Lambda' selected. The main area shows a table titled 'Functions (10)' with columns for 'Function name', 'Description', 'Runtime', 'Code size', and 'Last modified'. A search bar at the top says 'Filter by tags and attributes or search by keyword'. At the top right, there are 'Actions' and a large orange 'Create function' button. The bottom right of the table has navigation icons for pages 1-2 and a refresh symbol.

Leave the setting on Author from Scratch and populate the following fields:

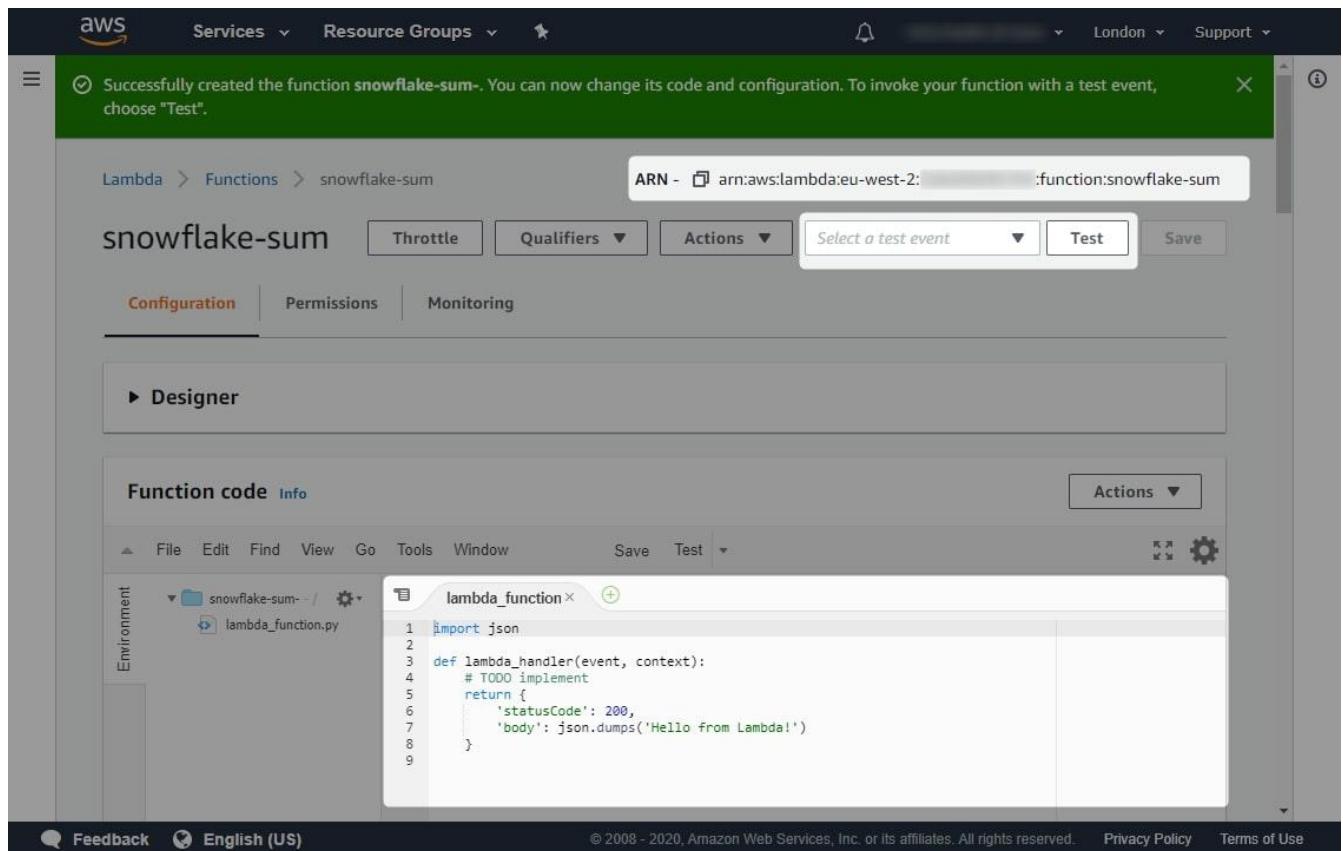
- **Function Name:** Enter a descriptive name for the function. For our example, we use snowflake-sum.
- **Runtime:** Choose the language which the function is written in. For our example, we use Python 3.8.
- **Execution Role:** Choose Use an Existing Role and pick the service role you just created from the drop-down list.

Once these settings are entered, select Create Function:



This opens up a fresh Python Lambda function for editing. Before we continue, let's take a quick look at what we can see. I have highlighted three areas:

- The ARN, or Amazon Resource Name, of the function. This is a unique way of referencing the function and should be added to our list of important variables.
- The test event section, where we can define test data for our function
- The main editor window where the function itself is defined



2a. How Snowflake External Functions Transfer Data

Before we can define our Lambda function itself, we need to understand how the data is transferred between Snowflake and our function. This determines how the function should expect to retrieve any input variables. This is best understood by using an example, so consider the following query and resulting table from Snowflake:

The screenshot shows the Snowflake SQL interface. The query window contains the following SQL code:

```
131
132 SELECT
133   x
134 , y
135 FROM external_functions.lambda.numeric_example
136 ;
137
```

The results pane shows the output of the query:

Row	X	Y
1	5	11
2	17	-4
3	16	11
4	-1	2
5	-2	-2

As we can see here, any **SELECT** query in Snowflake outputs not only the requested columns but also a row number. Our query selects two columns, X and Y, and outputs both of these along with a row number. Our full output is five rows, each comprised of three columns.

Before we continue, if you are not familiar with JSON data structures, please recall our [Introduction to Semi-Structured JSON Data Formats](#).

Using arrays, we can right this tabular output a different way. Instead of outputting it as a table, we could break the rows into members of an array where each row is also its own array:

```
[  
  [0, 5, 11]  
  , [1, 17, -4]  
  , [2, 16, 11]  
  , [3, -1, 2]  
  , [4, -2, -2]  
]
```

There are a few things to understand here. We can see that the second two numbers in each row match our X and Y values in our table, so this makes sense. However, the first digit in each row is a bit more confusing. This is actually the row number. We can see that it increments by 1 with each row, beginning at 0 and ending at 4. This is in line with our five-row output; however, we start counting at 0 instead of at 1. This is a common practice in programming and data, but if you are seeing it for the first time, take a minute to absorb what is happening to ensure you understand this structure.

This JSON structure forms the basis of how Snowflake will transfer data within an API request to an API gateway. However, we are not done yet. Snowflake will wrap this JSON structure into a larger JSON requests object (which we will not go into), and it achieves this by making two changes.

First, the information is inserted into a "data" key as follows:

```
{  
  "data" : [  
    [0, 5, 11]  
    , [1, 17, -4]  
    , [2, 16, 11]
```

```
, [3, -1, 2]  
, [4, -2, -2]  
]  
}
```

Second, this entire structure is converted into a string and inserted under another key called **body**:

```
{  
  "body" : "{\\\"data\\\" : [ [0, 5, 11], [1, 17, -4], [2, 16, 11], [3, -1, 2], [4, -2, -2] ]}"  
}
```

This is the same structure as we began with, but it has been:

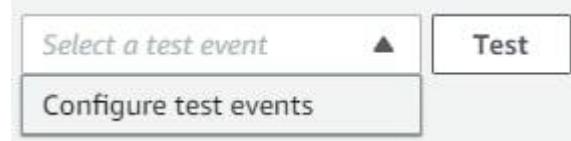
1. Collapsed into a single row
2. Stored under a "data" key
3. Converted to a string
4. Stored under a "body" key

This is the final format with which the data will be transferred from Snowflake to the API gateway.

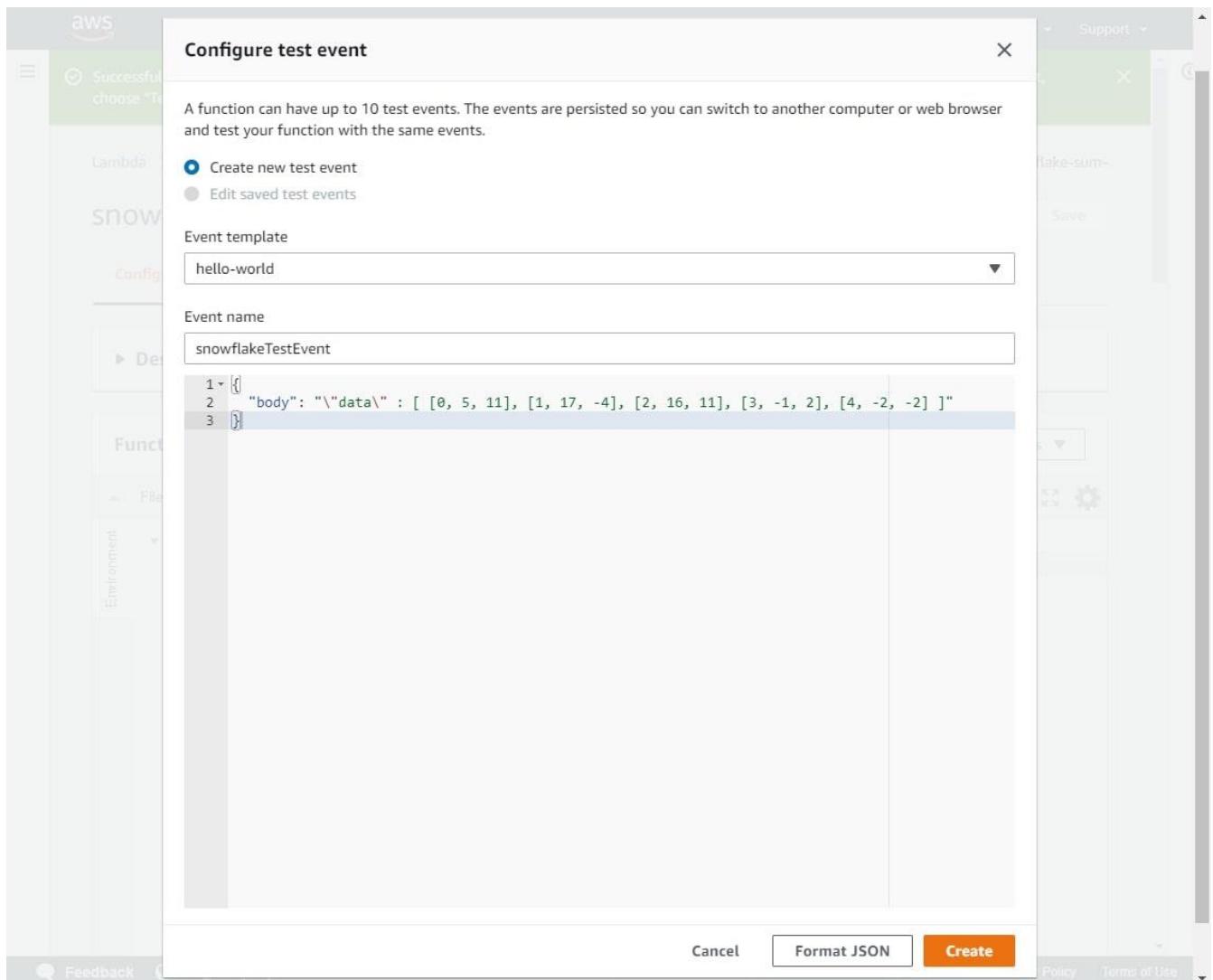
Important Note: Keep in mind that this structure is used by Snowflake both when *sending* and *receiving* data. Any external function you build in AWS Lambda must return data in this same format and *must return the same number of rows as the incoming request*.

2b. Configuring a Test Event in AWS Lambda

Now that we understand how Snowflake will transmit the data to our function, we can set up a test event in AWS Lambda to mimic a Snowflake request. Begin by selecting the Select a test event dropdown and selecting Configure test events:



Enter a name for your event (in our example, we use snowflakeTestEvent), and enter the following sample data, which is the same set of five rows described above. Once done, select **Create**:



Notice that the dropdown in the top-right corner has already updated to select your test event.

2c. Building an AWS Lambda Function for Snowflake-Compatible Inputs and Outputs

The purpose of this post is not to run through Python itself or Lambda functions, so I will not go into significant detail here. The key things to understand about a function designed to work with Snowflake is that it must perform the following feats:

1. Read the string "body" value of the incoming event and convert it to a JSON object
2. Read the "data" value from this JSON object to gain an array of data rows
3. Iterate through each of these rows, performing the desired functionality on the input variables. During this iteration, build a new "data" JSON object which will be returned by the function. Ensure this new "data" object still contains row numbers and outputs the same number of rows as it receives in the input.
4. Convert the "data" JSON object to a string
5. Output this string in the "body" key of the response

AWS Lambda Function to Perform a SUM in a Snowflake-Friendly Way

Here is the Python code which creates a Snowflake-friendly **SUM** function. You should be able to copy and paste this directly into the editor area in your own Lambda function:

```
```python
```

```
Import necessary modules for the function

import json

import functools

The function itself, in which event is the full request sent to the API

def lambda_handler(event, context):

 # Declare return variables

 statusCode = 200

 # dataArray stores the resultset of the function which will be

 # returned by the function. This begins as an empty array and is added

 # to in the rows loop below

 dataArray = []

 # json_compatible_string_to_return is the json body returned by the

 # function, stored as a string. This empty value is replaced by the real

 # result when ready

 json_compatible_string_to_return = ""

 # try/except is used for error handling

 try:

 # Retrieve the body of the request as a JSON object

 body = json.loads(event['body'])

 # Retrieve the 'data' key of the request body

 # When the request comes from Snowflake, we expect data to be a

 # an array of each row in the Snowflake query resultset.

 # Each row is its own array, which begins with the row number,

 # for example [0, 2, 3] would be the 0th row, in which the

 # variables passed to the function are 2 and 3. In this case,
```

```
the function sum output would be 5.

rows = body['data']

Loop through each row

for row in rows:

 # Retrieve the row number from the start of the row array

 rowNumber = row[0]

 # Retrieve the array of numbers to sum

 numbersToSum = row[1:]

 # try/except is used for error handling

 try:

 # Calculate the rowSum

 rowSum = functools.reduce(lambda a,b : a+b, numbersToSum)

 except:

 rowSum = "Error"

 # Create a new array entry

 newArrayEntry = [rowNumber, rowSum]

 # Add the newArrayEntry to the main dataArray list

 dataArray.append(newArrayEntry)

 # Put dataArray into a dictionary, then convert it to a string

 dataArrayToReturn = {'data' : dataArray}

 json_compatible_string_to_return = json.dumps(dataArrayToReturn)

except Exception as err:

 # Statuscode = 400 signifies an error

 statusCode = 400

 # Function will return the error

 json_compatible_string_to_return = json.dumps({"data":str(err)})

return {

 'statusCode': statusCode

 , 'headers': { 'Content-Type': 'application/json' }
```

```

 , 'body' : json_compatible_string_to_return
 }
...

```

Once you have inserted this code, you should be able to test it straightaway. We have already configured a test event called **snowflakeTestData**, so go ahead and click “**Test**”. The following screenshot should match your test result:

The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, London dropdown, and Support dropdown. The main title is "snowflake-sum". Below the title, there are buttons for Throttle, Qualifiers, Actions, and a dropdown set to "snowflakeTestData". A prominent "Test" button is highlighted. To the right of the "Test" button are "Save" and a help icon. The main workspace is titled "Function code" with an "Info" link. It contains a code editor window showing a Python script named "lambda\_function.py". The script retrieves a JSON object from the request body, extracts the "data" key, loops through each row, and sums the values. The "Execution Result" tab shows a successful execution with a status of "Succeeded", maximum memory used of 51 MB, and a duration of 0.84 ms. The response body is a JSON object with a single key "body" containing the array [0, 16, 1, 13, 2, 27, 3, 1, 4, -4]. Below the execution results, the "Environment variables" section is visible, showing a table with one entry: "Key" (empty) and "Value" (empty). At the bottom of the page, there are links for Feedback, English (US), Copyright notice (© 2008–2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

Specifically, the response we expect from our function is:

```
{
 "statusCode": 200,
 "headers": {
 ...
 }
}
```

```
"Content-Type": "application/json"

},
"body": "{\"data\": [[0, 16], [1, 13], [2, 27], [3, 1], [4, -4]]}"
}
```

Now that we have tested our function successfully, we must remember to add our function's name and ARN to our list of important variables:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

AWS Lambda Function to Perform a PRODUCT in a Snowflake-Friendly Way

Follow the above steps to create another AWS Lambda Python function, with the same sample data and owned by the same IAM service role. The code for the product function is:

```
Import necessary modules for the function

import json

import functools

The function itself, in which event is the full request sent to the API

def lambda_handler(event, context):

 # Declare return variables

 statusCode = 200

 # dataArray stores the resultset of the function which will be

 # returned by the function. This begins as an empty array and is added

 # to in the rows loop below

 dataArray = []

 # json_compatible_string_to_return is the json body returned by the

 # function, stored as a string. This empty value is replaced by the real

 # result when ready

 json_compatible_string_to_return = ""

 # try/except is used for error handling
```

```
try:

Retrieve the body of the request as a JSON object

body = json.loads(event['body'])

Retrieve the 'data' key of the request body

When the request comes from Snowflake, we expect data to be a

an array of each row in the Snowflake query resultset.

Each row is its own array, which begins with the row number,

for example [0, 2, 3] would be the 0th row, in which the

variables passed to the function are 2 and 3. In this case,

the function product output would be 6.

rows = body['data']

Loop through each row

for row in rows:

Retrieve the row number from the start of the row array

rowNumber = row[0]

Retrieve the array of numbers to multiply

numbersToMultiply = row[1:]

try/except is used for error handling

try:

Calculate the rowProduct

rowProduct = functools.reduce(lambda a,b : a*b, numbersToMultiply)

except:

rowProduct = "Error"

Create a new array entry

newArrayEntry = [rowNumber, rowProduct]

Add the newArrayEntry to the main dataArray list

dataArray.append(newArrayEntry)

Put dataArray into a dictionary, then convert it to a string

dataArrayToReturn = {'data' : dataArray}
```

```

json_compatible_string_to_return = json.dumps(dataArrayToReturn)

except Exception as err:

Statuscode = 400 signifies an error

statusCode = 400

Function will return the error

json_compatible_string_to_return = json.dumps({"data":str(err)})

return {

'statusCode': statusCode

, 'headers': { 'Content-Type': 'application/json' }

, 'body' : json_compatible_string_to_return

}

```

Once tested, the following result should be returned:

```
{
"statusCode": 200,
"headers": {
"Content-Type": "application/json"
},
"body": "{\"data\": [[0, 55], [1, -68], [2, 176], [3, -2], [4, 4]]}"
}
```

Now that we have tested our function successfully, we must remember to add our function's name and ARN to our list of important variables:

<b>Lambda Service Role:</b>	<b>snowflake-lambda-service-role</b>
<b>SUM Function:</b>	<b>snowflake-sum</b>
<b>SUM Function ARN:</b>	<b>arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum</b>
<b>PRODUCT Function:</b>	<b>snowflake-product</b>
<b>PRODUCT Function ARN:</b>	<b>arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product</b>

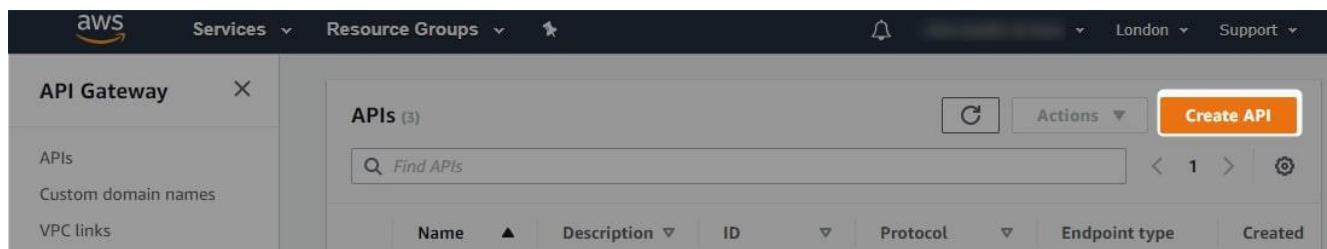
### 3. Configure an AWS API Gateway to Expose the Lambda Functions to Snowflake

There are several stages to creating an API gateway to expose lambda functions:

1. Create the API gateway itself
2. Create a resource and method for each function, which determines the possible functionality within the API
3. Deploy the API to a stage, which exposes the resources and methods to authorised roles (by default, none are authorised)

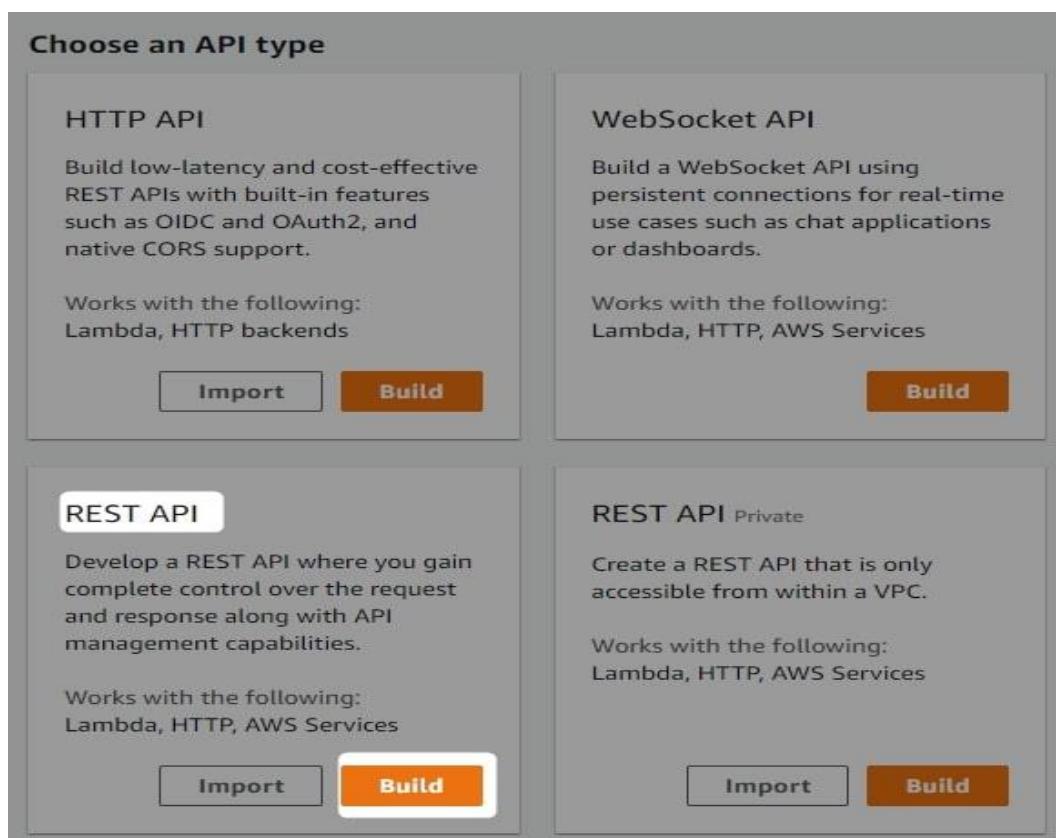
#### 3a. Create an AWS API Gateway

Navigate to the API Gateway area in AWS and select Create API:



The screenshot shows the AWS API Gateway console. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, London region, and Support dropdown. The main left sidebar has tabs for 'API Gateway' (selected), APIs, Custom domain names, and VPC links. The main content area is titled 'APIs (3)' with a search bar labeled 'Find APIs'. A table header includes columns for Name, Description, ID, Protocol, Endpoint type, and Created. At the top right of the main area is a 'Create API' button.

Locate the API type called **REST API** and select **Build**:



The screenshot shows a modal dialog titled 'Choose an API type'. It contains four options: 'HTTP API', 'WebSocket API', 'REST API', and 'REST API Private'. Each option has a description, a 'Works with the following:' section, and 'Import' and 'Build' buttons. The 'REST API' option is highlighted with a white border.

API Type	Works with the following:	Action Buttons
HTTP API	Lambda, HTTP backends	Import, Build
WebSocket API	Lambda, HTTP, AWS Services	Build
REST API	Lambda, HTTP, AWS Services	Import, Build
REST API Private	Lambda, HTTP, AWS Services	Import, Build

There are a few options on this next screen, though none of them need to be changed. We are creating a new REST API and will stick with a Regional endpoint type. For advanced readers who wish to experiment with other endpoint types, please note that the current version of External Functions in Snowflake only supports regional endpoint types.

**Give your API a name and description, then select Create API:**

The screenshot shows the AWS Management Console interface for creating a new API. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, London location, and Support dropdown. The main navigation bar shows 'Amazon API Gateway' selected under 'APIs > Create'. On the left, there's a sidebar with 'APIs', 'Custom Domain Names', and 'VPC Links'. The main content area starts with 'Choose the protocol' and 'REST' selected. Below that is 'Create new API' with 'New API' selected. The 'Settings' section contains fields for 'API name\*' (set to 'snowflake-external-functions-api'), 'Description' (set to 'API for Snowflake external Lambda functions.'), and 'Endpoint Type' (set to 'Regional'). At the bottom right is a large blue 'Create API' button. The footer includes links for 'Feedback', 'English (US)', copyright notice (© 2008–2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.), 'Privacy Policy', and 'Terms of Use'.

### 3b. Create a Resource and Method for Each Function

Once you have created your API gateway, open it up to see the following screen, which is empty as we have a fresh API gateway. Our next task is to create resources and methods for each function. By this stage, I imagine readers unfamiliar with APIs may start feeling a bit overwhelmed. This is completely understandable, and as long as you follow the steps we take, you shouldn't have a problem. You do not need to understand how an API works to set one up, and the following setup has been tested successfully. However, if anything is not working, you may wish to find other material online to boost your knowledge on API gateways.

In short and simple terms, an API resource is like a folder that stores API methods. An API method is a function that can be called when something accesses the API. We start by creating a resource for our SUM function. Select Actions at the top and select Create Resource:

The screenshot shows the AWS Amazon API Gateway interface. In the left sidebar, under the 'Resources' section, there is a single resource entry labeled '/'. A context menu is open over this entry, with 'Actions' selected. The 'Create Resource' option is highlighted. The main pane displays the message 'No methods defined for the resource.'

Populate the resource name. You should find that the resource path updates itself automatically, mimicking the resource name but replacing any white space with hyphens. Do not change any other settings, and select **Create Resource**:

The screenshot shows the 'New Child Resource' dialog. The 'Resource Name\*' field is set to 'Snowflake Sum'. The 'Resource Path\*' field is set to '/snowflake-sum'. Below the fields, there is explanatory text about path parameters. At the bottom right, there are 'Cancel' and 'Create Resource' buttons, with 'Create Resource' being the active button.

Once you have created the resource, select it so that it is highlighted. You can then create a method for this resource by selecting **Actions**, then **Create Method**:

The screenshot shows the AWS API Gateway console. In the left sidebar, under the 'APIs' section, there is a link labeled 'API: snowflake-exte...'. Below this, under the 'Resources' heading, several options are listed: Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy, Documentation, and Settings. The 'Actions' dropdown menu is open over a resource path '/snowflake-sum'. The menu is divided into two sections: 'RESOURCE ACTIONS' and 'API ACTIONS'. Under 'RESOURCE ACTIONS', the options are Create Method, Create Resource, Enable CORS, Edit Resource Documentation, and Delete Resource. Under 'API ACTIONS', the options are Deploy API, Import API, Edit API Documentation, and Delete API. A message at the top right of the main content area says 'No methods defined for the resource.'

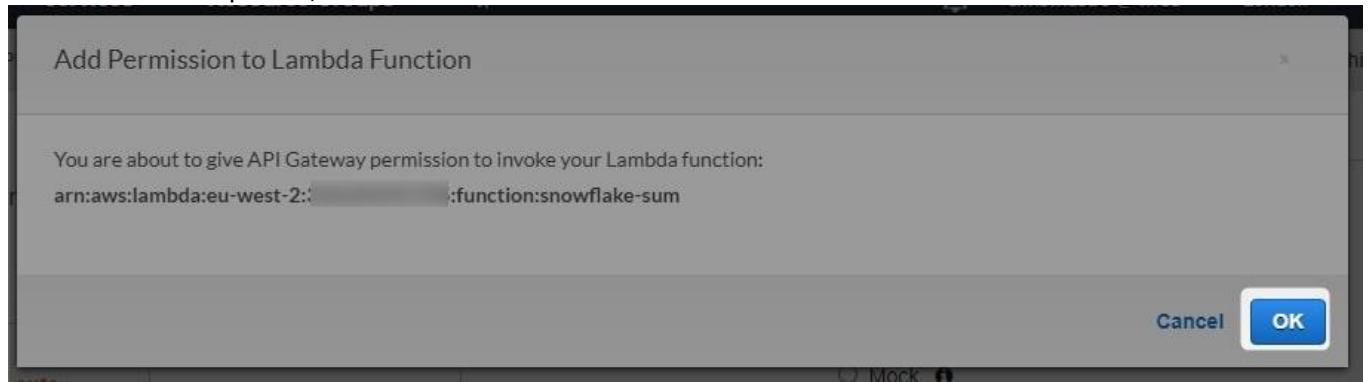
A small drop-down window will appear underneath your resource. Use this dropdown to select a **POST** resource then click the small tick icon:



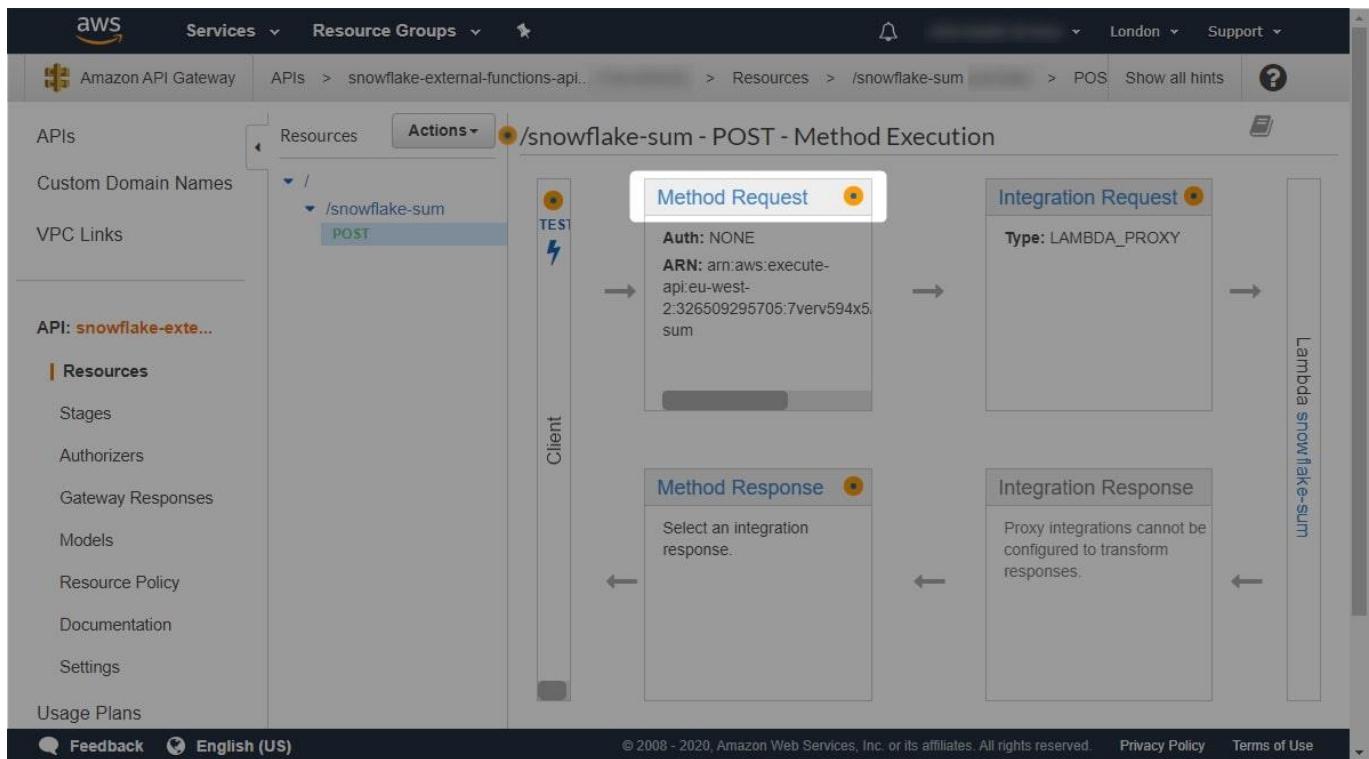
You will then see a screen requesting various details for the new method. As our method is intended to call a Lambda function, select **Lambda Function** as the **Integration Type**, and select **Use Lambda Proxy integration** to allow the API to call the Lambda function automatically. Choose a region for your function—ideally the same region as your Snowflake environment or your Lambda functions just to keep things simple. Then select **Save**:

The screenshot shows the AWS API Gateway configuration page. The top navigation bar includes 'Services' (dropdown), 'Resource Groups' (dropdown), 'London' (dropdown), and 'Support' (dropdown). The main navigation bar shows 'APIs > snowflake-external-functions-api... > Resources > /snowflake-sum > POS > Show all hints'. On the left sidebar, under 'API: snowflake-exte...', the 'Resources' tab is selected. The main content area shows a tree structure with a POST method under '/snowflake-sum'. A callout box highlights the 'Integration type' section, which is set to 'Lambda Function' (radio button selected). Other options include 'HTTP', 'Mock', 'AWS Service', and 'VPC Link'. Below this, there's a 'Use Lambda Proxy integration' checkbox (checked) and a dropdown for 'Lambda Region'. A 'Lambda Function' input field contains 'snowflake-sum'. A 'Use Default Timeout' checkbox (checked) is also present. At the bottom right is a blue 'Save' button.

You will be greeted by a warning message that you are about to add permission to a Lambda function. This is expected, so click **OK**:

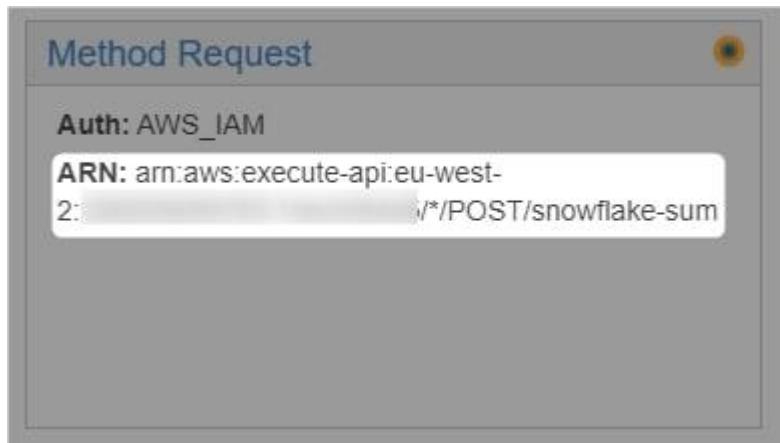


After a few moments, the settings will save and your view will refresh. You will now see a diagram displaying how traffic will be received and returned by your method. The key thing that requires attention here is that the method request authorisation is currently **NONE**. We need to fix that, so select the **Method Request** header:



Click the small pencil icon next to the Authorisation setting to edit. Change the selection to AWS\_IAM, and click the small tick icon to save your new setting:

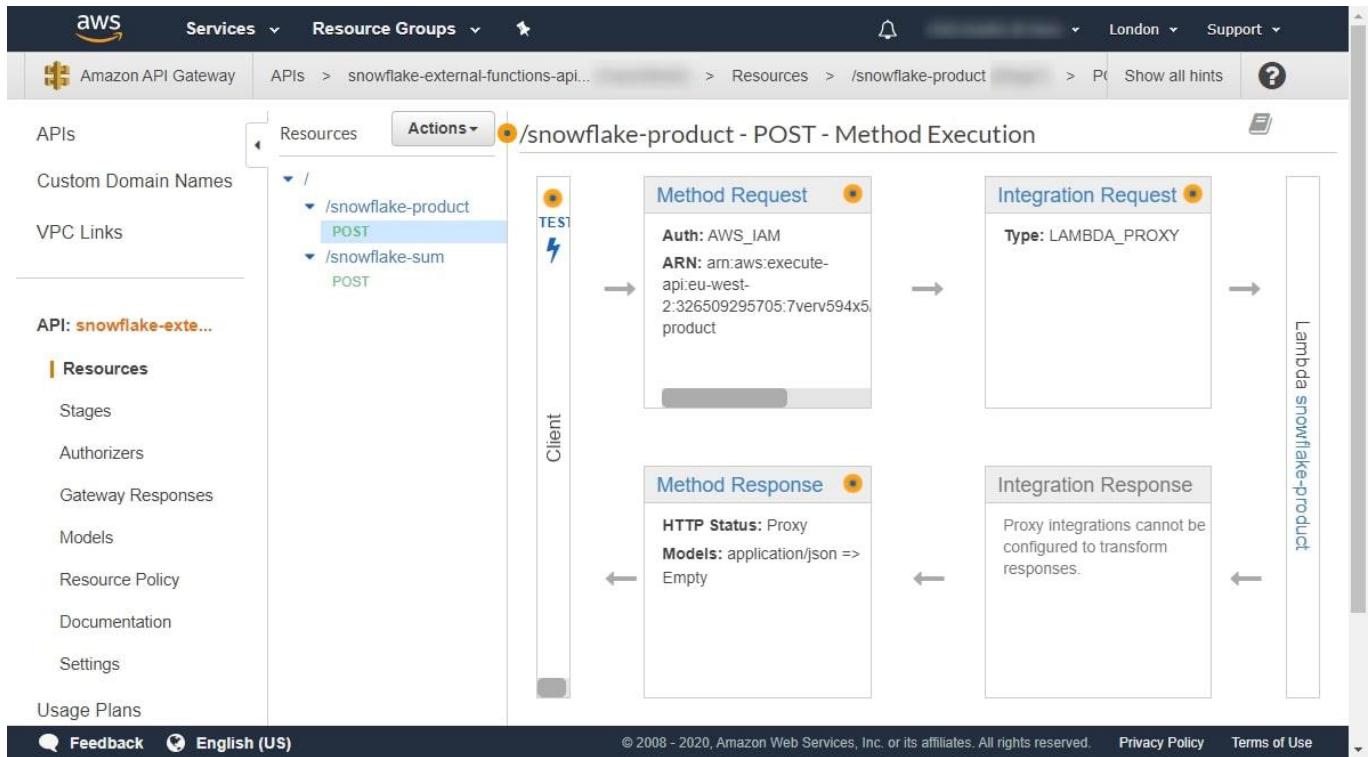
Once this change is made, return to the method overview. You will now see that the Method Request section is populated with both the **Auth** method and the method **ARN**:



Add this method ARN to the list of important variables:

**Lambda Service Role:** snowflake-lambda-service-role  
**SUM Function:** snowflake-sum  
**SUM Function ARN:** arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum  
**PRODUCT Function:** snowflake-product  
**PRODUCT Function ARN:** arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product  
**SUM Method ARN:** arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*:POST/snowflake-sum

Repeat the steps above to create a resource and method for our PRODUCT function. Be sure to select the root / when creating your resource so that it is created next to the **snowflake-sum** resource and not beneath it. When you are finished, you should see a screen like this:



Remember to add the method ARN to the list of important variables:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

### 3c. Deploy the API to a Stage

We are now ready to deploy our API to a stage, which exposes it to the outside world. Don't worry—it will only expose the API to authorised roles, and right now we have not authorised any, so this is still fully secure. Still within the Resources area, select Actions and then Deploy API:

The screenshot shows the AWS API Gateway console. In the top navigation bar, 'Services' is selected, followed by 'Resource Groups'. The main area displays an API named 'snowflake-external-functions-api...' with a single resource path '/snowflake'. A context menu is open over this path, showing options under 'Actions':

- RESOURCE ACTIONS
  - Create Method
  - Create Resource
  - Enable CORS
  - Edit Resource Documentation
- API ACTIONS
  - Deploy API
  - Import API
  - Edit API Documentation
  - Delete API

The 'Deploy API' option is highlighted. The status message 'No methods defined for the resource.' is visible on the right.

In the menu that appears, select **[New Stage]** from the drop-down menu and populate the name and description fields. For our example, we will use the name **snowflake-external-function-stage**. Once these details are entered, select **Deploy**:

The 'Deploy API' dialog box is shown. It contains the following fields:

Deployment stage	[New Stage] <input type="button" value="▼"/>
Stage name*	snowflake-external-function-sta
Stage description	Stage for external functions for Snowflake
Deployment description	First deployment

At the bottom right are two buttons: 'Cancel' and 'Deploy'.

Once the stage has deployed, you will see a screen similar to the below. At the top of this screen is an invoke URL for your stage. This is the URL through which the API is accessed by authorised roles, and it is critical to setting up our API integration:

The screenshot shows the AWS Amazon API Gateway Stage Editor. On the left, the navigation pane shows an API named "snowflake-exte...". Under the "Stages" section, "snowflake-external-function-stage" is selected. The main panel displays the "snowflake-external-function-stage Stage Editor". A prominent box highlights the "Invoke URL: https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage". Below this, there are tabs for Settings, Logs/Tracing, Stage Variables, SDK Generation, and Export. Under Settings, there are sections for Cache Settings (with an unchecked "Enable API cache" checkbox) and Default Method Throttling (with an checked "Enable throttling" checkbox set to 10000 requests per second). The bottom of the page includes standard AWS footer links for Feedback, English (US), and various legal notices.

Add this stage Invoke URL to the list of important variables that we are using:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>**

Selecting each method in turn allows us to see the Invoke URL for that method as well:

The screenshot shows the AWS API Gateway console. On the left, the navigation pane includes 'Services' (selected), 'Resource Groups', 'APIs' (selected), 'Custom Domain Names', 'VPC Links', and a section for the current API named 'snowflake-external-functions-api...'. Under 'Stages', 'snowflake-external-function-stage' is selected, and it contains two methods: '/snowflake-product' (POST) and '/snowflake-sum' (POST). The '/snowflake-product' method is currently selected. On the right, the main panel displays the configuration for the POST method to '/snowflake-product'. It shows the 'Invoke URL' as <https://execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>. Below the URL, there's a note: 'Use this page to override the snowflake-external-function-stage settings for the POST to /snowflake-product method.' There are two radio buttons under 'Settings': ' Inherit from stage' and ' Override for this method'. A 'Save Changes' button is located at the bottom right.

Add these to the list of important variables, too:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>>**

**SUM Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-sum>>**

**PRODUCT Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>>**

Our API gateway is now configured, apart from integration with Snowflake. The next step is to create a AWS IAM role for Snowflake and configure the various access and integrations.

#### **4. Configuring an AWS IAM Role Which Will Facilitate the Integration Between Snowflake and the AWS API Gateway**

So far, we have created two Lambda functions, a service role to execute them and an API gateway to expose them. We are nearer the end of our journey and will now set up the authorisation and integration to allow Snowflake to access these objects.

There are six steps to configuring an AWS IAM role to integrate an API with Snowflake:

1. Retrieve your AWS Account ID
2. Configure an AWS IAM role using this account ID
3. Retrieve the role ARN
4. Authorise this role ARN in the resource policy for the API gateway
5. In Snowflake, create a Snowflake API Integration Object which leverages this role to access the API gateway
6. In AWS, create a trust relationship between the AWS IAM role and the Snowflake API Integration object

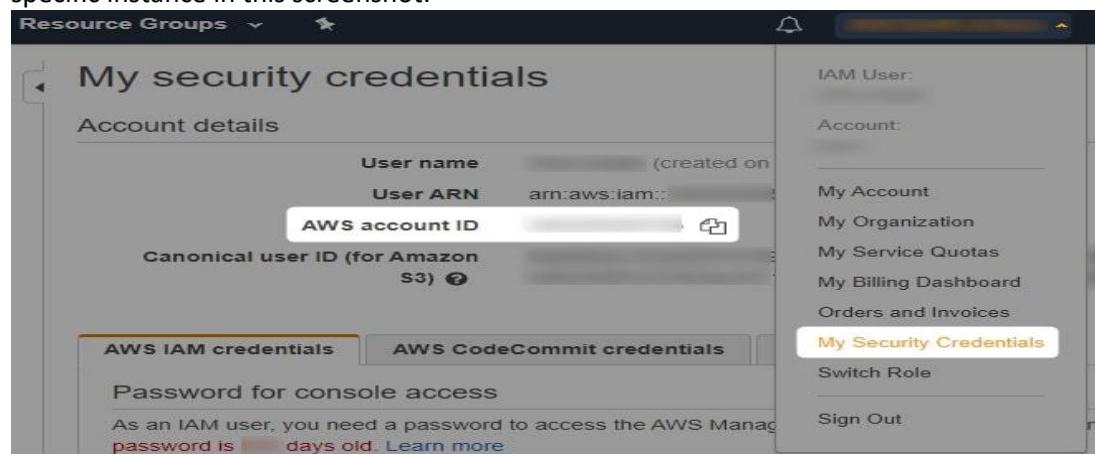
##### **4a. Retrieve Your AWS Account ID**

Before we can create this role, we first must locate our AWS account ID. There are several ways to identify this ID, with the simplest being to extract it from the ARNs of your lambda functions. A Lambda function ARN is formed as follows:

```
arn:aws:lambda:<region>:<account ID>:function:<function name>
```

Looking back at our function ARNs on the list of important variables, we can see that our account ID is 012345678910. Note that 012345678910 is only an example value.

If you are taking these steps in a different order, usually the account ID is available in the My Security Credentials area, as in the screenshot below. Note that I have blurred out details on the specific instance in this screenshot:



If this does not work, you can follow the [AWS account identifiers documentation](#) to find several ways of tracking down your account ID. Take note of your account ID by adding it to your list of important variables:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>>**

**SUM Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-sum>>**

**PRODUCT Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>>**

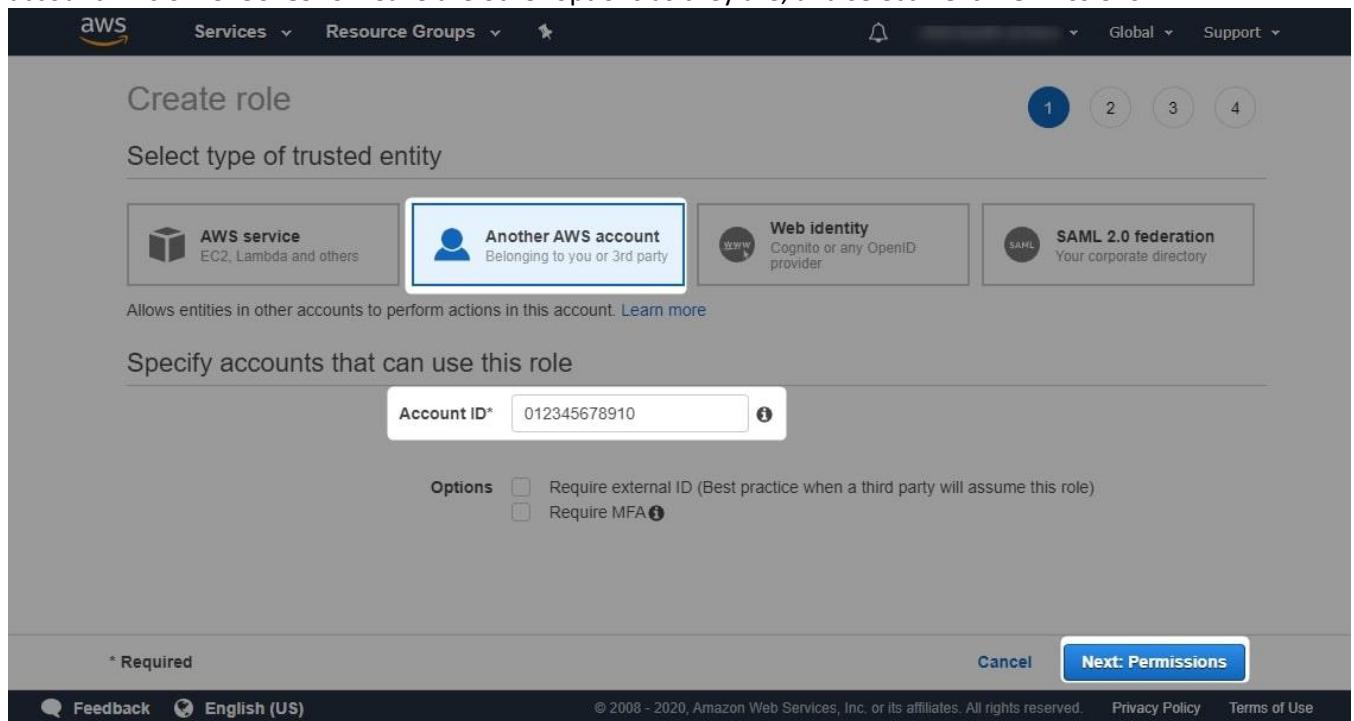
**Account ID: 012345678910**

#### 4b. Configure an AWS IAM Role

Navigate to the IAM area in AWS and select Roles then Create Role:

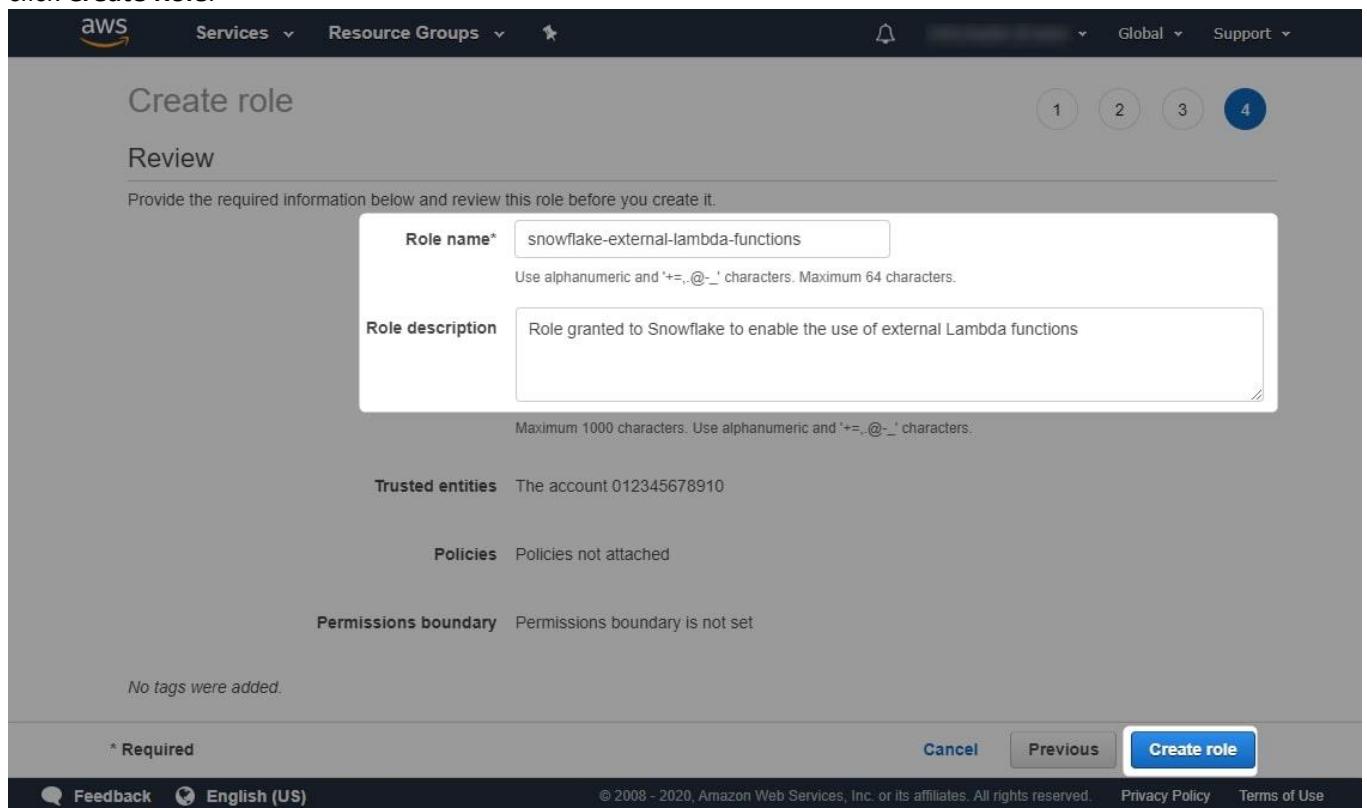
The screenshot shows the AWS IAM service interface. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, Global dropdown, and Support dropdown. The main menu on the left is under 'Identity and Access Management (IAM)'. It has sections for 'Access management' (Groups, Users, Roles - highlighted in yellow), 'Access reports' (Access analyzer, Archive rules, Analyzers, Settings), and 'Credential report'. The 'Roles' section is currently selected. The main content area is titled 'Roles' and contains a 'What are IAM roles?' section with a list of entities that can be granted permissions (IAM user in another account, Application code running on an EC2 instance, An AWS service, Users from a corporate directory). Below this is a note about IAM roles issuing short-lived keys for secure access. There is also an 'Additional resources:' section with links to IAM Roles FAQ, Documentation, and Tutorials. At the bottom of the content area are 'Create role' and 'Delete role' buttons, along with other standard UI controls. The footer of the page includes 'Feedback' (with a speech bubble icon), language selection ('English (US)'), copyright information ('© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.'), and links to 'Privacy Policy' and 'Terms of Use'.

In this section, navigate to **Another AWS Account** and enter your account ID. For our example, our account ID is **012345678910**. Leave the other options as they are, and select **Next: Permissions**:



The screenshot shows the 'Create role' wizard on the AWS IAM service. The first step, 'Select type of trusted entity', is displayed. It offers four options: 'AWS service' (EC2, Lambda and others), 'Another AWS account' (Belonging to you or 3rd party, which is selected and highlighted in blue), 'Web identity' (Cognito or any OpenID provider), and 'SAML 2.0 federation' (Your corporate directory). Below the options, a note states: 'Allows entities in other accounts to perform actions in this account. Learn more'. The 'Account ID\*' field contains '012345678910'. Under 'Options', there are two checkboxes: 'Require external ID (Best practice when a third party will assume this role)' and 'Require MFA'. At the bottom, there are buttons for 'Cancel', 'Next: Permissions', 'Feedback', 'English (US)', and links to 'Privacy Policy' and 'Terms of Use'.

The following through menus are not required for this example, so click **Next** a few times until you reach the review area. Once here, enter an appropriate name and description for your role, then click **Create Role**:



The screenshot shows the 'Create role' wizard on the AWS IAM service, currently at the 'Review' step. The role has been named 'snowflake-external-lambda-functions'. The 'Role description' is set to 'Role granted to Snowflake to enable the use of external Lambda functions'. Other details shown include 'Trusted entities' (The account 012345678910), 'Policies' (Policies not attached), and 'Permissions boundary' (Permissions boundary is not set). A note at the bottom states 'No tags were added.' At the bottom, there are buttons for 'Cancel', 'Previous', 'Create role', 'Feedback', 'English (US)', and links to 'Privacy Policy' and 'Terms of Use'.

Take note of your new role name by adding it to your list of important variables:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>>**

**SUM Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-sum>>**

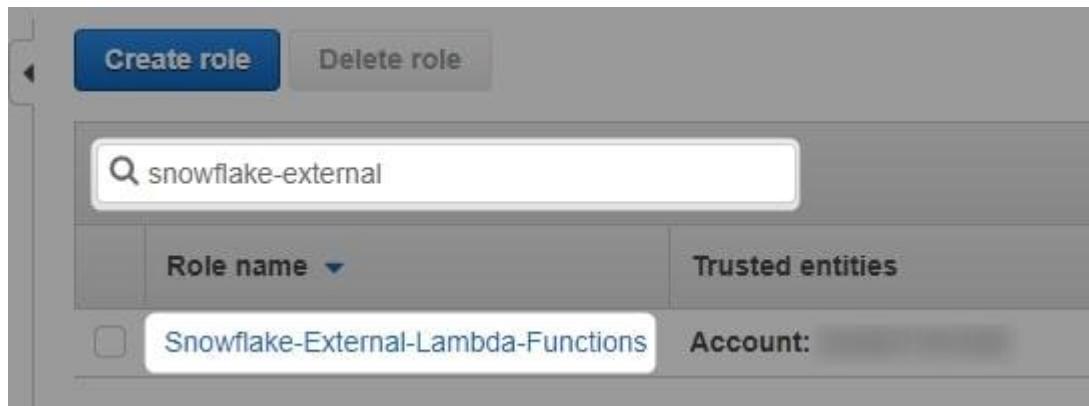
**PRODUCT Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>>**

**Account ID: 012345678910**

**IAM Account Role: snowflake-external-lambda-functions**

#### 4c. Retrieve the Role ARN

Once you have created your role, we need to open it and find its ARN. To do this, return to the main IAM area in AWS, search for your role and select it from the list:



Once you have opened the role, you will see the following screen. At the top is your role ARN:

The screenshot shows the AWS IAM Roles Summary page. The left sidebar has 'Identity and Access Management (IAM)' selected under 'Roles'. The main area shows a role named 'Snowflake-External-Lambda-Functions'. The 'Role ARN' field is highlighted with a yellow box, showing the value 'arn:aws:iam::XXXXXXXXXX:role/Snowflake-External-Lambda-Functions'. Below it, the 'Role description' is 'Role granted to Snowflake to enable the user of external lambda functions' with an 'Edit' link. Other fields include 'Instance Profile ARNs' (empty), 'Path' ('/'), 'Creation time' ('2020-08-04 15:18 UTC+0100 (Today)'), 'Last activity' ('1 hour ago'), 'Maximum session duration' ('Edit'), and a note to 'Give this link to users who can switch roles in the console'. At the bottom, tabs for 'Permissions', 'Trust relationships', 'Tags', 'Access Advisor', and 'Revoke sessions' are visible, with 'Permissions' being the active tab. A 'Permissions policies' section is shown below.

Take note of your new role ARN by adding it to your list of important variables:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:012345678910:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>>**

**SUM Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-sum>>**

**PRODUCT Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>>**

**Account ID: 012345678910**

**IAM Account Role: snowflake-external-lambda-functions**

**IAM Account Role ARN: arn:aws:iam::012345678910:role/snowflake-external-lambda-functions**

#### 4d. Authorise This Role in the Resource Policy for the API Gateway

Return to the API Gateway area in AWS, and open the API gateway that we created earlier. Navigate to the Resource Policy section on the left-hand side to see the following empty resource policy area:

The screenshot shows the AWS API Gateway interface. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, London region, Support dropdown, and a question mark icon. The main navigation menu on the left lists APIs, Custom Domain Names, VPC Links, API: snowflake-exte..., Resources, Stages, Authorizers, Gateway Responses, Models, Resource Policy (which is selected and highlighted in orange), Documentation, Dashboard, Settings, Usage Plans, API Keys, Client Certificates, and Settings. The central content area is titled "Resource Policy" and contains the following text: "Configure access control to this private API using a Resource Policy. Access can be controlled by IAM condition elements, including conditions on AWS account, Source VPC, VPC Endpoints (Private API), and/or IP range. If the Principal in the policy is set to \*, other authorization types can be used alongside the resource policy. If the Principal is set to AWS, then authorization will fail for all resources not secured with AWS\_IAM auth, including unsecured resources." Below this text is a large, empty text input field with the number "1" at its top left corner. At the bottom of the page, there are three buttons labeled "AWS Account Whitelist", "IP Range Blacklist", and "Source VPC Whitelist", followed by a blue "Save" button. The footer includes links for Feedback, English (US), Privacy Policy, and Terms of Use.

Resource policies are used to control authorisation within an object in AWS. Our goal is to enter a resource policy which grants our IAM Account role the capability to invoke the API methods for our two Lambda functions. We start with this empty template, which you will notice is in JSON format:

```
{
 "Version": "2012-10-17",
```

```
"Statement":
[
{
"Effect": "Allow",
"Principal":
{
"AWS": "arn:aws:sts::<account ID>:assumed-role/<IAM Account Role>/snowflake"
},
"Action": "execute-api:Invoke",
"Resource": "<Method ARN>"
}
]
}
```

The above template allows us to grant an IAM account role the capability to invoke a single method.

Don't worry about changing the version. Using this as a template, we can create the following resource policy which grants our account role access to both of our methods:

```
{
"Version": "2012-10-17",
"Statement":
[
{
"Effect": "Allow",
"Principal":
{
"AWS": "arn:aws:sts::<account ID>:assumed-role/<IAM Account Role>/snowflake"
},
"Action": "execute-api:Invoke",
"Resource": "<SUM Method ARN>"
}]
```

```
},
{
 "Effect": "Allow",
 "Principal":
 {
 "AWS": "arn:aws:sts::<account ID>:assumed-role/<IAM Account Role>/snowflake"
 },
 "Action": "execute-api:Invoke",
 "Resource": "<PRODUCT Method ARN>"
}
]
```

If we populate this resource policy template with our example values, we have the following:

```
{
 "Version": "2012-10-17",
 "Statement":
 [
 {
 "Effect": "Allow",
 "Principal":
 {
 "AWS": "arn:aws:sts::012345678910:assumed-role/snowflake-external-lambda-functions/snowflake"
 },
 "Action": "execute-api:Invoke",
 "Resource": "arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/*/POST/snowflake-sum"
 }
]
}
```

```

"Effect": "Allow",
"Principal":
{
 "AWS": "arn:aws:sts::012345678910:assumed-role/snowflake-external-lambda-
functions/snowflake"
},
"Action": "execute-api:Invoke",
"Resource": "arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/*/POST/snowflake-
product"
}
]
}

```

Note that there is a shortcut we could use to add multiple resources under the same list member by leveraging a \* notation. However, I prefer avoiding this to ensure functions are not made available accidentally during development and testing. If you wish to do so, replace the resource name with a \* as in the following example:

```

{
 "Version": "2012-10-17",
 "Statement":
 [
 {
 "Effect": "Allow",
 "Principal":
 {
 "AWS": "arn:aws:sts::012345678910:assumed-role/snowflake-external-lambda-
functions/snowflake"
 },
 "Action": "execute-api:Invoke",
 "Resource": "arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/*/*"
 }
]
}
```

```
]
}
```

Whichever your preference, insert this into the resource policy and hit **Save**:

The screenshot shows the AWS Amazon API Gateway Resource Policy configuration page. The left sidebar lists various API management options like APIs, Custom Domain Names, VPC Links, Resources, Stages, Authorizers, Gateway Responses, Models, and Usage Plans. The 'Resource Policy' option is selected and highlighted in orange. The main content area is titled 'Resource Policy' and contains a detailed JSON-based policy document. The policy document is as follows:

```
1 {
2 "Version": "2012-10-17",
3 "Statement":
4 [
5 {
6 "Effect": "Allow",
7 "Principal":
8 [
9 {
10 "AWS": "arn:aws:sts::012345678910:assumed-role/snowflake-external-lambda-functions/snowflake"
11 },
12 "Action": "execute-api:Invoke",
13 "Resource": "arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/*/POST/snowflake-sum"
14 },
15 {
16 "Effect": "Allow",
17 "Principal":
18 [
19 {
20 "AWS": "arn:aws:sts::012345678910:assumed-role/snowflake-external-lambda-functions/snowflake"
21 },
22 "Action": "execute-api:Invoke",
23 "Resource": "arn:aws:execute-api:eu-west-2:012345678910:xxxxxxxxxx/*/POST/snowflake-product"
24]
25 }
26]
27 }
```

Below the policy editor, there are three buttons: 'AWS Account Whitelist', 'IP Range Blacklist', and 'Source VPC Whitelist'. A blue 'Save' button is located at the bottom right. The bottom navigation bar includes links for Feedback, English (US), Copyright notice (© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

This concludes the setup for the API gateway. Now to integrate Snowflake!

#### 4e. Create a Snowflake API Integration Object

Since this is a post about setting something up for Snowflake, it's surprising how much goes on in AWS and how much progress we have made without even touching Snowflake. We are now at a stage, though, where we finally get to use Snowflake a little bit, so go ahead and log into your Snowflake environment.

Using a role that has account admin privileges, we can set up an API integration object with the following code:

```
CREATE OR REPLACE api integration <integration name>

api_provider = aws_api_gateway

api_aws_role_arn = '<IAM Account Role ARN>'

enabled = true

api_allowed_prefixes = ('<API Stage Invoke URL>')

;
```

For our example, we will call the integration **aws\_lambda**, so our script would be:

```
CREATE OR REPLACE api integration aws_lambda

api_provider = aws_api_gateway

api_aws_role_arn = 'arn:aws:iam::012345678910:role/snowflake-external-lambda-functions'

enabled = true

api_allowed_prefixes = ('<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>')

;
```

Once this completes, execute the following code to see the specific details of the integration:

```
DESCRIBE integration aws_lambda;
```

Snowflake will output the following table of values:

The screenshot shows the Snowflake interface with the query results for the DESCRIBE integration command. The results are displayed in a table format.

Row	property	property_type	property_value	property_default
1	ENABLED	Boolean	true	false
2	API_AWS_IAM_USER_ARN	String	[REDACTED]	
3	API_AWS_ROLE_ARN	String	[REDACTED]	
4	API_AWS_EXTERNAL_ID	String	[REDACTED]	
5	API_ALLOWED_PREFIXES	List	[REDACTED]	
6	API_BLOCKED_PREFIXES	List	[REDACTED]	

We are particularly interested in the **API\_AWS\_IAM\_USER\_ARN** and the **API\_AWS\_EXTERNAL\_ID** fields, as these are the key indicators for how Snowflake is attempting to access AWS. Add the integration name, along with these two values to the list of important variables, being sure not to drop the = sign at the end of the external ID:

**Lambda Service Role: snowflake-lambda-service-role**

**SUM Function: snowflake-sum**

**SUM Function ARN: arn:aws:lambda:eu-west-2:**012345678910**:function:snowflake-sum**

**PRODUCT Function: snowflake-product**

**PRODUCT Function ARN: arn:aws:lambda:eu-west-2:**012345678910**:function:snowflake-product**

**SUM Method ARN: arn:aws:execute-api:eu-west-2:**012345678910**:xxxxxxxxxx/\*/POST/snowflake-sum**

**PRODUCT Method ARN: arn:aws:execute-api:eu-west-2:**012345678910**:xxxxxxxxxx/\*/POST/snowflake-product**

**API Stage Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage>>**

**SUM Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-sum>>**

**PRODUCT Invoke URL: <<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-stage/snowflake-product>>**

**Account ID: **012345678910****

**IAM Account Role: snowflake-external-lambda-functions**

**IAM Account Role ARN: arn:aws:iam::**012345678910**:role/snowflake-external-lambda-functions**

**API Integration Name: aws\_lambda**

**api\_aws\_iam\_user\_arn: arn:aws:iam::**109876543210**:user/xxxx-x-xxxxxxx**

**api\_aws\_external\_id: DEMO\_SFCRole=0\_x0xxxxxxxxxxxxxx+xx0xx0xx=**

#### *4f. Create a Trust Relationship*

Return to the AWS IAM area, and open your IAM Account Role. Navigate to the Trust relationships tab and select Edit trust relationship:

The screenshot shows the AWS Identity and Access Management (IAM) service interface. On the left, the navigation pane is open, showing various sections like Dashboard, Access management, Roles (which is selected), Policies, Identity providers, Account settings, and more. The main content area is titled 'Summary' for the role 'snowflake-external-lambda-functions'. Key details shown include:

- Role ARN:** arn:aws:iam:::role/snowflake-external-lambda-functions
- Role description:** Role granted to Snowflake to enable the use of external Lambda functions | Edit
- Instance Profile ARNs:** [empty]
- Path:** /
- Creation time:** [redacted]
- Last activity:** Not accessed in the tracking period
- Maximum session duration:** 1 hour | Edit

Below these details is a link: "Give this link to users who can switch roles in the console".

The bottom section contains tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The Trust relationships tab is currently selected. It displays the following information:

- A message: "You can view the trusted entities that can assume the role and the access conditions for the role. Show policy document"
- A blue button: "Edit trust relationship"
- Trusted entities:** The following trusted entities can assume this role.
  - The account
- Conditions:** The following conditions define how and when trusted entities can assume the role.
  - No conditions are listed.

At the bottom of the page, there are links for Feedback, English (US), Copyright notice (© 2008 - 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

Here you will find another JSON-structured policy document. By default, it appears as follows:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::<account ID>:root"
 },
 "Action": "sts:AssumeRole",
 "Condition": {}
 }
]
}
```

To allow Snowflake to leverage this role, modify the policy as follows:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "<api_aws_iam_user_arn>"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "sts:ExternalId": "<api_aws_external_id>"
 }
 }
 }
]
}
```

For our example, this would appear as follows:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "AWS": "arn:aws:iam::109876543210:user/xxxx-x-xxxxxxxx"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "sts:ExternalId": "<api_aws_external_id>"
 }
 }
 }
]
}
```

```
"Condition": {
 "StringEquals": {
 "sts:ExternalId": "DEMO_SFCRole=0_x0xxxxxxxxxxxxxx+xx0xx0xx="
 }
}
}
}
]
}
```

Enter this into the policy document area and select **Update Trust Relationship**:

The screenshot shows the AWS IAM console with the 'Edit Trust Relationship' page open. The top navigation bar includes the AWS logo, Services dropdown, Resource Groups, a notification bell, Global dropdown, and Support dropdown. On the left, there's a sidebar with 'Edit Trust Relationship'. The main content area has a title 'Edit Trust Relationship' and a sub-instruction 'You can customize trust relationships by editing the following access control policy document.' Below this is a 'Policy Document' section containing the JSON policy code shown in the code block above. At the bottom right of the main area are 'Cancel' and 'Update Trust Policy' buttons.

```
1 {
2 "Version": "2012-10-17",
3 "Statement": [
4 {
5 "Effect": "Allow",
6 "Principal": {
7 "AWS": "arn:aws:iam::109876543210:user/xxxx-x-xxxxxxx"
8 },
9 "Action": "sts:AssumeRole",
10 "Condition": {
11 "StringEquals": {
12 "sts:ExternalId": "DEMO_SFCRole=0_x0xxxxxxxxxxxxxx+xx0xx0xx="
13 }
14 }
15 }
16]
17 }
```

We have now finished the AWS portion of this guide and can focus on the final Snowflake steps.

## 5. Creating External Function Objects in Snowflake

The final steps are to build and test some external functions in Snowflake. Let's return to Snowflake and begin by granting our new API integration to the SYSADMIN role. Of course, you can grant this to another role if you would like:

```
USE role securityadmin;
GRANT usage on integration aws_lambda to role sysadmin;
```

Using the SYSADMIN role, we can now create a database and schema to store our external functions:

```
USE role sysadmin;
CREATE OR REPLACE database external_functions;
CREATE OR REPLACE schema external_functions.lambda;
```

This gives us the basic framework to start creating our external functions. The template to create an external function is as follows:

```
CREATE [OR REPLACE] external function <database>.<schema>.<function name>(variables)
returns variant
api_integration = <api integration object>
as '<method invoke url>'
```

Following this template, we can create both of our functions:

```
CREATE OR REPLACE external function external_functions.lambda.sum(x number, y number)
returns variant
api_integration = aws_lambda
as '<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-
stage/snowflake-sum>'

CREATE OR REPLACE external function external_functions.lambda.product(x number, y number)
returns variant
api_integration = aws_lambda
as '<https://xxxxxxxxxx.execute-api.eu-west-2.amazonaws.com/snowflake-external-function-
stage/snowflake-product>'
```

Now all we need to do is test our functions to make sure they work. The simplest way to test these is to execute them directly with manual inputs. First, we manually test the **SUM** function:

```
88 USE warehouse demo_warehouse;
89
90 SELECT external_functions.lambda.sum(20, 5);
91
```

results Data Preview

Query\_ID SQL 1.43s 1 rows

Filter result...

Row	EXTERNAL_FUNCTIONS.LAMBDA.SUM(20, 5)
1	25

And then we manually test the **PRODUCT** function:

```
01
88 USE warehouse demo_warehouse;
89
90 SELECT external_functions.lambda.product(20, 5);
91
```

results Data Preview

Query\_ID SQL 416ms 1 rows

Filter result...

Row	EXTERNAL_FUNCTIONS.LAMBDA.PRODUCT(20, 5)
1	100

Both are working! Just for fun, let's wrap up with a more comprehensive test. Use the following script to create a table of 500 rows, with randomly generated **x** and **y** values between -10 and 20:

```
CREATE OR REPLACE table external_functions.lambda.numeric_example(
```

```
 x number
```

```
 ,y number
```

```
)
```

```
AS
```

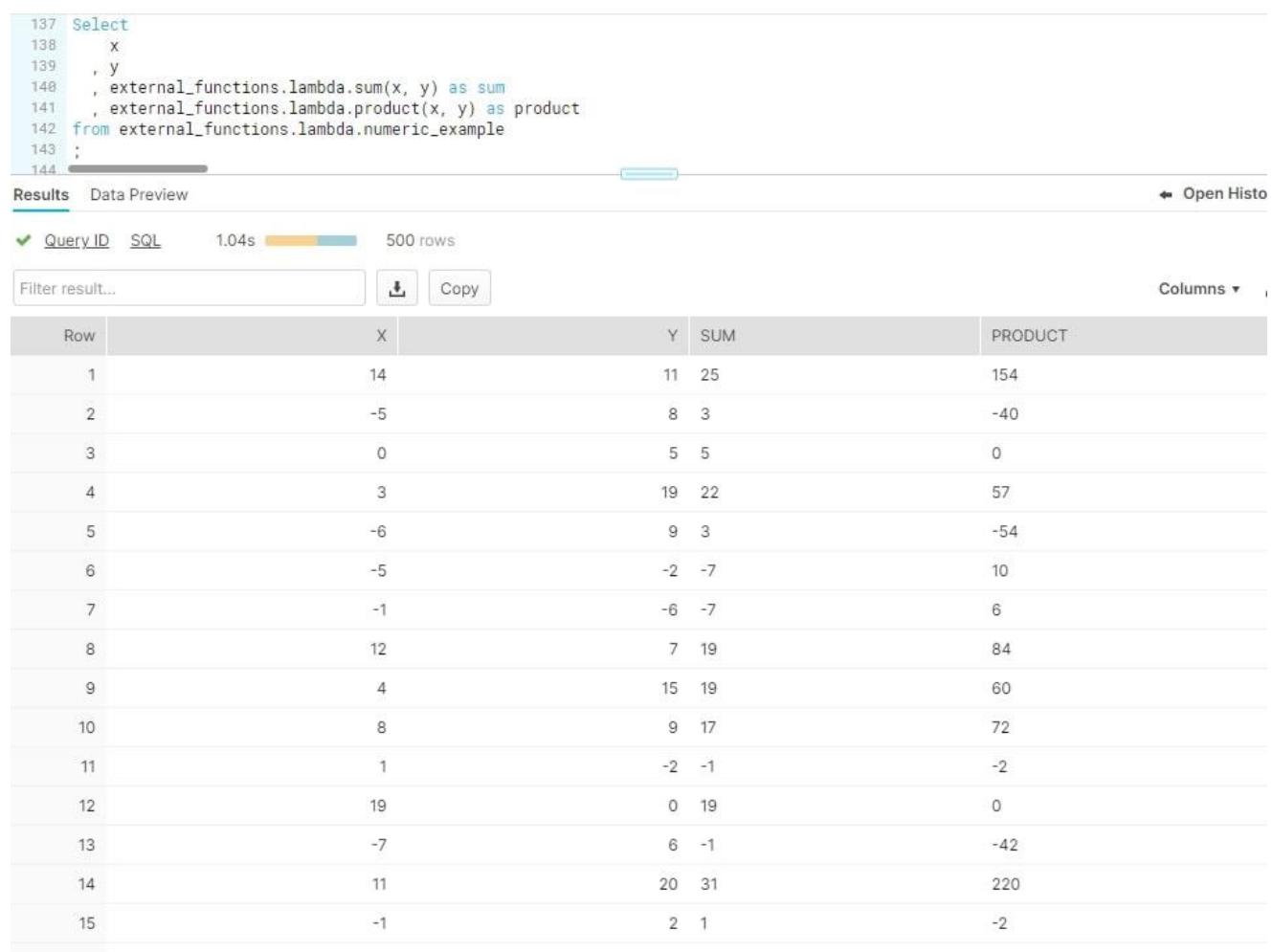
```
select
```

```

uniform(-10, 20, random()) as x
, uniform(-10, 20, random()) as y
from table(generator(rowcount => 500))
;

```

The following screenshot demonstrates this test in action:



The screenshot shows a Snowflake query results page. At the top, the SQL code is displayed:

```

137 Select
138 x
139 , y
140 , external_functions.lambda.sum(x, y) as sum
141 , external_functions.lambda.product(x, y) as product
142 from external_functions.lambda.numeric_example
143 ;
144

```

Below the code, the results are shown in a table format:

Row	X	Y	SUM	PRODUCT
1	14	11	25	154
2	-5	8	3	-40
3	0	5	5	0
4	3	19	22	57
5	-6	9	3	-54
6	-5	-2	-7	10
7	-1	-6	-7	6
8	12	7	19	84
9	4	15	19	60
10	8	9	17	72
11	1	-2	-1	-2
12	19	0	19	0
13	-7	6	-1	-42
14	11	20	31	220
15	-1	2	1	-2

And there we have it: two external functions in Snowflake that leverage AWS Lambda functions and work as intended. It took a while to set up, but now that the framework is there, it is much easier to add new functions.

The Spark streaming is a distributed processing engine that will take care of running input streams incrementally and continuously and updating the final result as streaming data continues to arrive.

It consumes data-streams from Kafka topic and preprocesses the data and writes the data into the Snowflake table.

Snowflake is an analytic data warehouse provided as Software-as-a-Service (SaaS), which runs on the cloud. Snowflake provides a data warehouse that is faster, easier to use, and far more flexible than traditional data warehouse offerings. Snowflake's data warehouse is not built on an existing database or "big data" software platform such as Hadoop.

**Jar versions:**

- Spark: spark-sql-kafka-0-10\_2.11.2.4.0.jar
- Spark-Snowflake-connector: spark-snowflake\_2.11-2.7.1-spark\_2.4.jar
- Snowflake-JDBC: snowflake-jdbc-3.12.8.jar

**Requirement:**

To design data warehouse solutions for live events that are getting populated from the messaging queue in real-time. It should support Upsert operation and manage large storage.

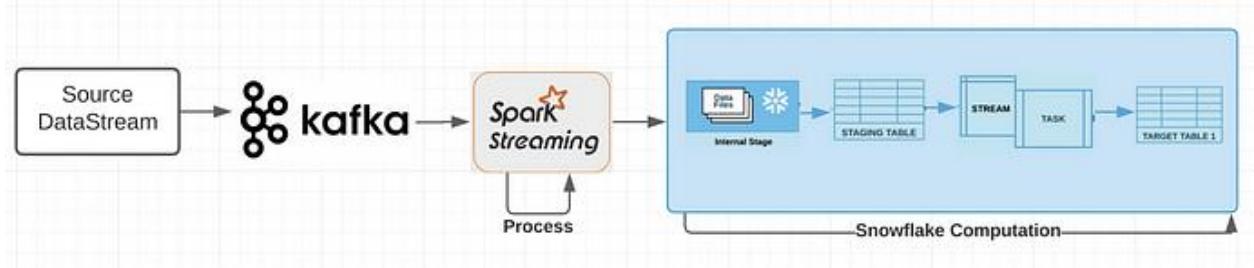
Make the data readily available for consumption once it is ingested into Snowflake via the Spark Connector.

**Design:**

Below steps are taken care of while designing the solution :

1. Source DataStream will send a series of events to the Kafka topic.

2. In order to capture a series of events/change data capture, create snowflake streams that capture the delta changes in the underlying table.
3. Spark Streaming application will consume the input stream from the Kafka topic and preprocess the data and this parsed data is being pushed to the snowflake staging table.
4. Create snowflake Tasks that will execute the underlying query and load data from the stream to the target table (upsert ops).



Key terminology based on the diagram

- **DataStream**: Input data stream which produces a series of CDC/events into Kafka.
- **Spark-streaming**: Preprocessing distributed engine.
- **Staging Table**: Realtime data appended to the Snowflake stage table.
- **Target Table**: This is the final table which will be used by the user to query the records.
- **Stream**: Snowflake object type that provides change data capture (CDC) capabilities to track the delta of changes in a table, including inserts and data manipulation language (DML) changes
- **Task**: A snowflake task that can schedule SQL/stored-procedure.

Implementation:

**Step 1:** Source DataStream will send a series of events to the Kafka topic.

For POC purposes we will cook some sample data and will send it to the Kafka topic. This step act as a production series event generator.

For this random data generator, I have used Spark Structured APIs, which will send data to the Kafka topic.

```
import spark.implicits._import org.apache.spark.sql.functions._var df = Seq[(Int, String, Int)]((1, "Bob", 200),(2, "Alex", 200),(3, "Raama", 100)).toDF("id", "name", "fee")df = df.withColumn("event_time", lit(System.currentTimeMillis()))df.toJSON.selectExpr("CAST(value AS STRING)").write.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("topic", "input_stream").save()Sample Kafka event series data looks as below
>{"id":1,"name":"Bob","fee":200,"event_time":1599993002991}
>{"id":2,"name":"Alex","fee":200,"event_time":1599993002991}
>{"id":3,"name":"Raama","fee":100,"event_time":1599993002991}
```

**Step2: Create a snowflake stage table and stream to capture CDC data.**

Create a Snowflake stage table and append-only stream on the stage table.

```
use UAT.PUBLIC;create or replace table members_stage (id number(8) not null,name varchar(255)
default null,fee number(3) null,event_time number(15));create or replace stream members_stream on
table members_stage append_only=true;create or replace table members_prod (id number(8) not
null,name varchar(255) default null,fee number(8) null);
```

Snowflake Streams: Provides a set of changes made to the underlying table since last time. It captures the snapshot of the table for the change data records. It is being consumed in DML statements in ETL load.

### Step3 Spark-streaming for process and send to the snowflake staging table.

Here we are using Spark-structured streaming APIs for processing/parsing and parsed data is sent to the snowflake stage table through the Spark-Snowflake internal stage connector.

The transfer of data between the two systems is facilitated through a Snowflake internal stage that the connector automatically creates and manages:

- Upon connecting to Snowflake and initializing a session in Snowflake, the connector creates the internal stage.
- Throughout the duration of the Snowflake session, the connector uses the stage to store data while transferring it to its destination.
- At the end of the Snowflake session, the connector drops the stage, thereby removing all the temporary data in the stage.
- 

```
import net.snowflake.spark.snowflake.SnowflakeConnectorUtilsimport
org.apache.spark.sql.types.{DataType, IntegerType, LongType, StringType, StructType}import
org.apache.spark.sql.streaming.Triggerimport spark.implicits._import org.apache.spark.sql.functions._val
inputStream = spark.readStream.format("kafka").option("kafka.bootstrap.servers",
"localhost:9092").option("subscribe", "input_stream").option("startingOffsets",
"earliest").load().selectExpr("CAST(value AS STRING)")val jSchema = new StructType().add("id",
IntegerType).add("name", StringType).add("fee", IntegerType).add("event_time", LongType)val stream =
inputStream.select(from_json(col("value"), jSchema).as("col1")).select(col("col1.*"))val
SNOWFLAKE_SOURCE_NAME =
"net.snowflake.spark.snowflake"SnowflakeConnectorUtils.enablePushdownSession(spark)val sfOptions =
new scala.collection.mutable.HashMap[String, String]()sfOptions += ("sfURL" ->
"https://*****.privatelink.snowflakecomputing.com/", "sfUser" -> "*****", "sfPassword" ->
"*****", "sfDatabase" -> "UAT", "sfSchema" -> "PUBLIC", "sfWarehouse" ->
"UAT_XSMALL_WH")stream.writeStream.trigger(Trigger.ProcessingTime("30 seconds")).foreachBatch((ds,
dt) =>
{ds.toDF().show(false)ds.toDF().write.format(SNOWFLAKE_SOURCE_NAME).options(sfOptions).option("d
btable", "members_stage").mode(SaveMode.Append).save()}).start().awaitTermination()
```

27 select * from members_stage;				
28 select * from members_stream;				
29 select * from members_prod;				
30				
Results Data Preview				
✓ Query_ID SQL 4.17s 3 rows				
Filter result... <input type="button" value="Copy"/> Columns ▾				
Row	ID	NAME	FEE	EVENT_TIME
1	1	Bob	200	1599993002991
2	3	Raama	100	1599993002991
3	2	Alex	200	1599993002991

### Sample data on stage table

27 select * from members_stage;							
28 select * from members_stream;							
29 select * from members_prod;							
30							
Results Data Preview							
✓ Query_ID SQL 773ms 3 rows							
Filter result... <input type="button" value="Copy"/> Columns ▾							
Row	ID	NAME	FEE	EVENT_TIME	METADATA\$ACTION	METADATA\$UPDATE	METADATA\$ROW_ID
1	1	Bob	200	1599993002991	INSERT	FALSE	87b2629ab54d28652a083...
2	3	Raama	100	1599993002991	INSERT	FALSE	2a617515668bb424a676bb...
3	2	Alex	200	1599993002991	INSERT	FALSE	31ce775e03998550a4070b...

### Sample data on Stream which captures CDC data

**Step4: Create snowflake Tasks that will execute the underlying query and load data from the stream to the target table (upsert ops).**

Once data is captured in the staging table and the CDC data is available in the stream, the next step is to merge these changes to target table.

This step can be scheduled at regular intervals by scheduling it on Task.

```
CREATE TASK UAT.PUBLIC.members_task1
WAREHOUSE = UAT_XSMALL_WHSCHEDULE = '1
minute'
ERROR_ON_NONDETERMINISTIC_MERGE=FALSEWHEN SYSTEM$STREAM_HAS_DATA('members_
stream')AS
merge into members_prod A using (select * from (select id, row_number() over (partition by id
order by event_time desc) rm, FIRST_VALUE(name ignore nulls) over (partition by id order by event_time
desc) as name, FIRST_VALUE(fee ignore nulls) over (partition by id order by event_time desc) as fee
from members_stream
where metadata$action = 'INSERT') temp
where rm = 1) Bon (A.id = B.id)
when matched
then update set A.id = COALESCE(B.id, A.id), A.fee = COALESCE(B.fee, A.fee), A.name =
coalesce(B.name, A.name)
when not matched then insert values (B.id, B.name, B.fee);
alter task UAT.PUBLIC.members_task1 RESUME;
-- By default task will be suspended mode
```

Here members\_task1 is a snowflake task that runs at an interval of every 1 minute. Which executes the underlying SQL query.

Basically we are merging CDC data in stream **members\_stream** with target table **members\_prod**.

We can also mention SYSTEM\$STREAM\_HAS\_DATA('members\_stream'), which means the task is going to execute only when there is CDC data inside the stream, leveraging snowflake computation only when there is data populated.

Results Data Preview			
Query ID	SQL	1.36s	3 rows
<code>77 select * from members_stage; 78 select * from members_stream; 79 select * from members_prod;</code>			
Row	ID	NAME	FEE
1	2	Alex	200
2	1	Bob	200
3	3	Rama	100

### Sample data in Target table post completion of the task

### Step 5: Sometimes later if the input data stream sends updated events with new records along.

{"id":1,"fee":1000,"event\_time":1599993240133}

{"id":2,"name":"Jhon","fee":200,"event\_time":1599993240133}

{"id":4,"name":"Sita","fee":500,"event\_time":1599993240133}

Results Data Preview					
Query ID	SQL	6.03s	6 rows	Columns	↓ EVENT_TIME
<code>7 select * from members_stage; 8 select * from members_stream; 9 select * from members_prod;</code>					
Row	ID	NAME	FEE		
1	4	Sita	500		1599993240133
2	1	NULL	1000		1599993240133
6	2	Jhon	200		1599993240133
3	1	Bob	200		1599993002991
4	3	Rama	100		1599993002991
5	2	Alex	200		1599993002991

### Sample data in stage: All the CDC is appended in the staging table

77	select * from members_stage;						
78	select * from members_stream;						
79	select * from members_prod;						
80							
<b>Data Preview</b>							
Query ID	SQL						
2.62s	3 rows						
Iter result...							
Row	ID	NAME	FEE	EVENT_TIME	METADATA\$ACTION	METADATA\$UPDATE	METADATA\$ROW_ID
1	4	Sita	500	1599993240133	INSERT	FALSE	23d44ad50f5f4cd474da12...
2	1	NULL	1000	1599993240133	INSERT	FALSE	8e3d98d00bd1d0979eee7a...
3	2	Jhon	200	1599993240133	INSERT	FALSE	7bac5496e336fea618f273c...

**Sample CDC only changed data captured in-stream.**

17	select * from members_stage;		
18	select * from members_stream;		
19	select * from members_prod;		
20			
<b>Data Preview</b>			
Query ID	SQL		
154ms	4 rows		
Iter result...			
Row	ID	NAME	FEE
1	2	Jhon	200
2	1	Bob	1000
3	3	Rama	100
4	4	Sita	500

**The sample target table is upserted with CDC data and null column data is handled with retaining old record, with one record inserted and two records updated to existing data in the target table**

#### Note:

- The primary key is ID in Target Table.
- In the merge query, I have used row number and first\_value function in order to handle multiple events arrived with the same PK value, it has to perform rollup on that particular interval of the task before it is merged to the target table.
- To check the Task history and time took to complete the underlying table is as below.

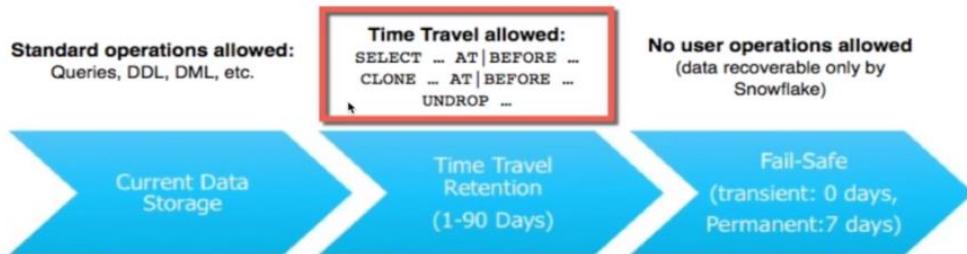
```
USE UAT.PUBLIC; select min (ts),max(ts) from (select
datediff('seconds',query_start_time,completed_time) ts,* from table(information_schema.task_history(
scheduled_time_range_start=>dateadd('hour',-2,current_timestamp()),task_name=>'members_task1')))
tt order by 1;
```

- You can also create one more Task scheduled at a daily interval, which deletes data from the staging table with data less than the current day.

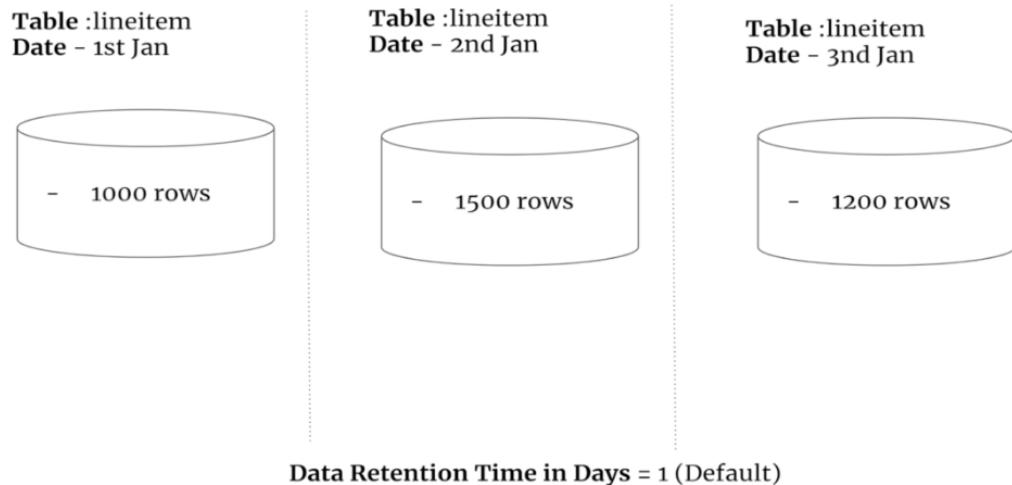
- You can also use Spark-JDBC with a merge query to do upsert operation with the target table, but it operates at an individual record level and it is not optimized by Snowflake query execution. As merging queries are always expensive.

## Snowpipe - Data Protection lifecycle

### Continuous Data Protection Lifecycle

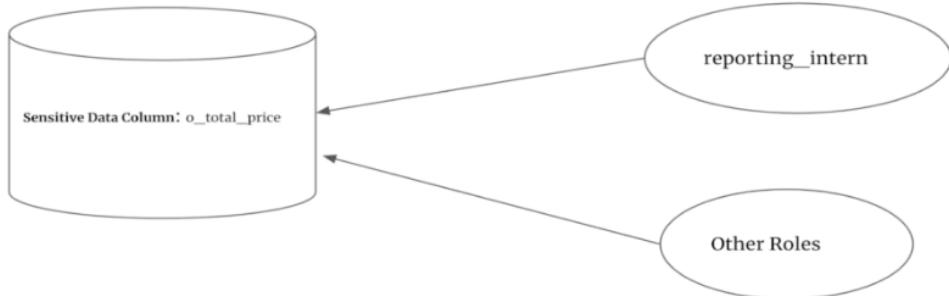


### Snowflake - Time Travel



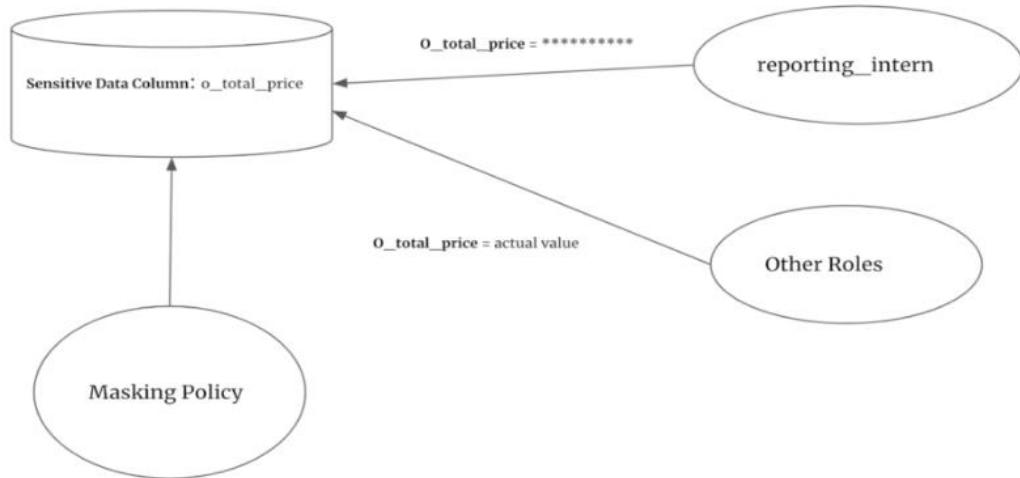
### Snowflake - Column Level Data Masking

#### Table : Orders



## Snowflake - Column Level Data Masking

Table : Orders

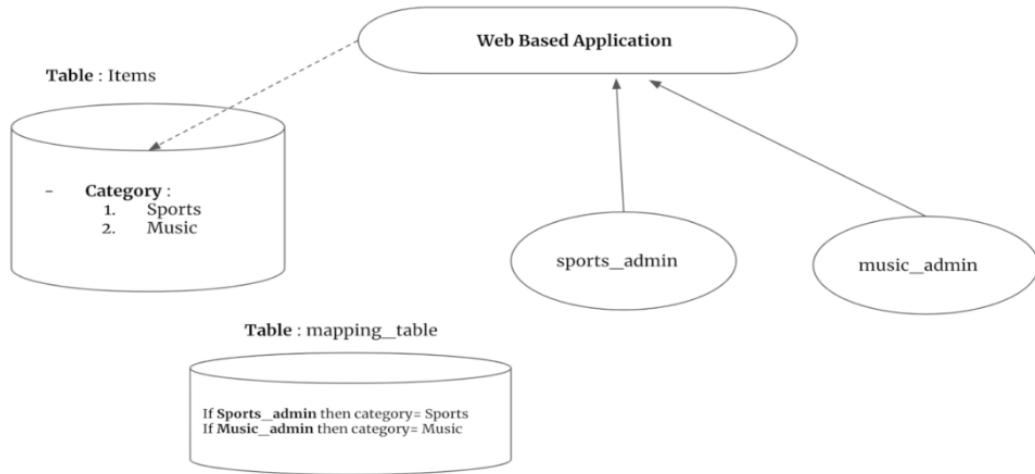


## Snowflake - Row Level Access Policy

Table : Items



## Snowflake - Row Level Access Policy



- Data governance is an organization's management of its data availability, usability, consistency, and [data integrity](#) and [data security](#). It defines who can take what action, upon what data, in what situations, using what methods and includes the processes, roles, policies, standards, and metrics for ensuring effective [data management](#) throughout the lifecycle of the data and for its use by the entire organization. Effective data governance empowers users to develop business insights from high-quality, secure, and trustworthy data.
- Every organization should have some form of data governance to prevent sensitive information from getting into the wrong hands. And if your company is large or in an industry subject to regulation such as healthcare or banking, data governance is critical.
- The right practices, processes, and tools for handling and using your data ensure data security, availability, and quality — so everyone within the organization can use the data to fix business problems and discover opportunities. Additional benefits:
  - **Regulatory Compliance**

You may be considering your governance strategy as part of your need to comply with regulatory policies such as the European Union's General Data Protection Regulation (GDPR) or the United States' Health Insurance Portability and Accountability Act (HIPAA). These and other regulations necessitate that you trace your data from source to retirement, identify who has access, and know how and where it is used. Effective data governance ensures that data won't fall into the wrong hands or be improperly removed.

- **Data Security**

Data breaches, including the theft of information and inappropriate access to data, are an all-too-common occurrence. As with regulatory compliance, data security depends on traceability — knowing where the data comes from, where it is located, who has access to it,

how it is used, and how to delete it upon request. Effective data governance prevents data leaks and misuse, thereby protecting your organization's reputation and revenue.

- **Improved Data Quality**

You can't discover valuable data-driven insights from data of low quality.

Effective [governance](#) increases trust in data. With effective data governance, you can identify when data is corrupt or inaccurate, when it's too stale and irrelevant, or when it's analyzed out of context. You can also identify siloed data and set rules and processes that integrate it among lines of business.

- Effective governance is achieved with a well-crafted strategy as part of an overall [data governance framework](#). Here are a few essential best practices for a successful data governance strategy.

- **1. Start small**

In the beginning, take advantage of existing opportunities to improve data management rather than attempt to tackle all [data governance](#) obstacles at once. Successful data governance projects include change management within the organization, so take small steps to test processes and communications and help everyone become more comfortable with changes.

- **2. Identify your data domains and define controls**

Corral all your data sets and access points to find your data control points. This includes mapping out a detailed plan to define automated workflow processes, approval thresholds, reviews, issue resolution, and more.

- **3. Measure progress**

Maintaining data governance best practices is an ongoing process. Include plans to regularly define, report, and measure against your data management goals. Metrics will naturally vary depending on your data size, scope, and sources, as well as how your data is disseminated both inside the organization — and potentially outside. A few metrics to consider include rate of adoption, number of data issues and events, and the program's overall cost, from data rectification to issue resolution.

- **4. Create a recurring, repeatable process**

Developing a data governance strategy is not a one-and-done project. As your data volumes grow, new data streams and access points will emerge. Devise a policy for periodic reviews of your data governance structure.

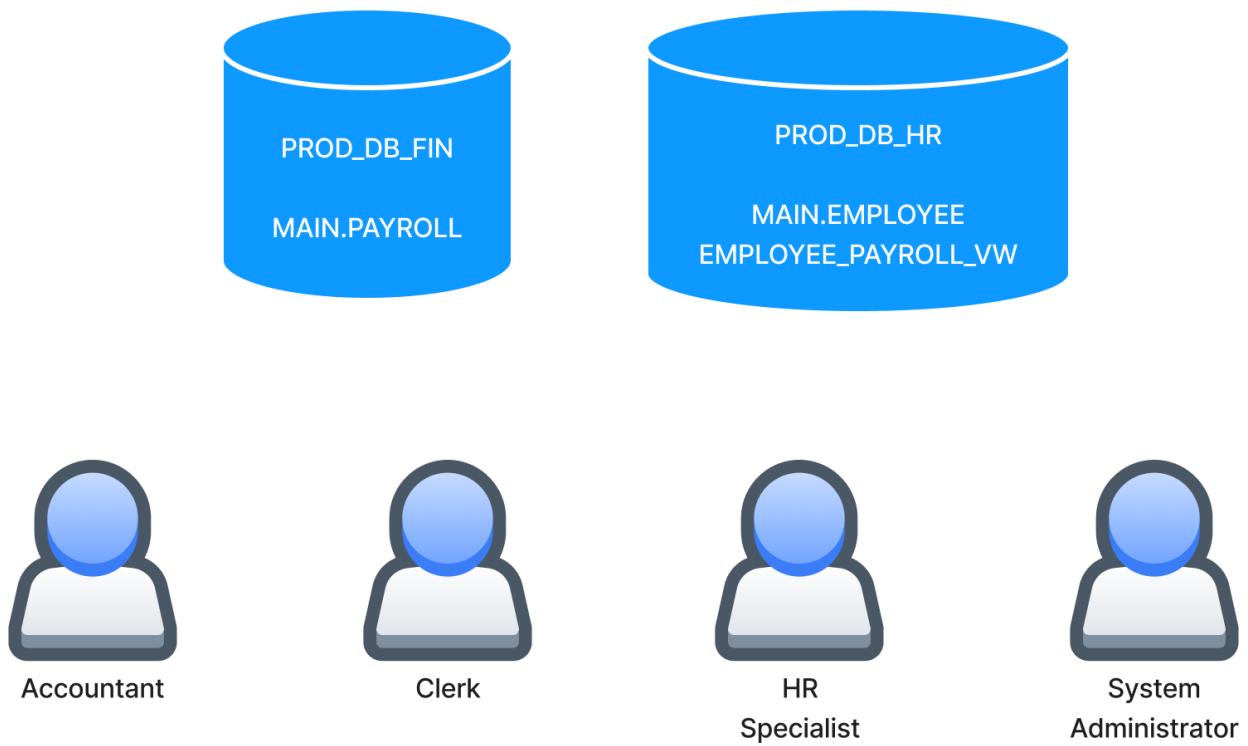
*Data governance has become an essential agenda item for many organizations. The reasons are [varied](#), but two of the most compelling ones are: (1) as the number of data assets grows, it becomes harder to properly control access to them; and (2) new and more stringent privacy regulations are increasing the calls for compliance. Snowflake provides a strong foundation that ensures high levels of governance on*

*your data assets. In this post, we will showcase the main features with some examples. We assume you have some essential experience working with Snowflake and its [access control](#) features.*

### Governance requirements

The organizational setup in this post is inspired by an [example](#) in the Snowflake documentation but adapted to showcase data governance features. Consider an organization that keeps payroll and employee data in two separate databases: PROD\_DB\_FIN and PROD\_DB\_HR, respectively.

In this example, the data needs to be accessed by three business users and managed by a system administrator. The following diagram shows this setup:



The organization needs to govern its data assets according to the following requirements:

- The accountant needs read/write access on PROD\_DB\_FIN to update payroll data but read-only access on PROD\_DB\_HR to read employee data. However, employees' dates of birth in PROD\_DB\_HR should be kept private.
- The clerk helps the accountant do some administrative tasks, so she also needs access to both databases in read-only mode. Additionally, the clerk shouldn't be able to read personal information such as date of birth in PROD\_DB\_HR or social insurance number (SIN) and salary in PROD\_DB\_FIN. The organization has employees in Canada and USA, but the clerk should have access only to the employees in Canada. The accountant takes care of the employees in the USA.
- The HR specialist needs read/write access on PROD\_DB\_HR to update employee data but read-only access on PROD\_DB\_FIN to read payroll data. Unlike the other roles, the HR specialist needs full access to personal information to perform her functions.
- The system administrator should manage both databases and implement the necessary controls to enforce the governance requirements. Only the system administrator should grant access to the databases.

## Database objects

The code below shows the definition of the tables in PROD\_DB\_HR and PROD\_DB\_FIN, as well as the data that is initially loaded:

```
--
-- Permanent tables

-- SIN numbers encoded as base64 strings

CREATE OR REPLACE TABLE PROD_DB_HR.MAIN.EMPLOYEE (
 EMPLOYEE_ID INTEGER,
 FIRST_NAME STRING,
 LAST_NAME STRING,
 BIRTH_DATE DATE,
 TITLE STRING,
 DEPARTMENT STRING,
 COUNTRY STRING,
 SOCIAL_INSURANCE_NUMBER STRING
);

INSERT INTO PROD_DB_HR.MAIN.EMPLOYEE
 VALUES
 (1, 'Jason', 'Chapman', '1980-05-31', 'Director', 'Sales', 'Canada', 'OTk5LTk5OS05OTE='),
 (2, 'Jennifer', 'Villanueva', '1996-03-14', 'Graphic Designer', 'Marketing', 'Canada', 'OTk5LTk5OS05OTI='),
 (3, 'Wayne', 'Cooper', '2000-10-25', 'Software Developer', 'Product Development', 'USA',
 'OTk5LTk5OS05OTM='),
 (4, 'Anna', 'Thompson', '1971-06-28', 'Accountant', 'Finance', 'USA', 'OTk5LTk5OS05OTQ=');

CREATE OR REPLACE TABLE PROD_DB_FIN.MAIN.PAYROLL (
 EMPLOYEE_ID INTEGER,
 EFFECTIVE_DATE DATE,
 SALARY_AMOUNT NUMERIC(10, 2)
);

INSERT INTO PROD_DB_FIN.MAIN.PAYROLL
```

## VALUES

```
(1, '2005-01-01', 85000),
(2, '2010-05-12', 72000),
(3, '2020-03-22', 76000),
(4, '2018-08-05', 69000);
```

We will be querying the data in the databases using the EMPLOYEE\_PAYROLL\_VW view below, which simply joins the data from both tables:

```
--
```

```
-- Secure views
```

```
--
```

```
USE PROD_DB_HR.MAIN;
```

```
CREATE OR REPLACE SECURE VIEW EMPLOYEE_PAYROLL_VW
```

```
AS
```

```
WITH LAST_RAISE AS (
```

```
SELECT
```

```
EMPLOYEE_ID,
```

```
MAX(EFFECTIVE_DATE) AS EFFECTIVE_DATE
```

```
FROM PROD_DB_FIN.MAIN.PAYROLL
```

```
GROUP BY
```

```
1
```

```
)
```

```
SELECT
```

```
E.EMPLOYEE_ID,
```

```
E.FIRST_NAME,
```

```
E.LAST_NAME,
```

```
E.FIRST_NAME || ' ' || E.LAST_NAME AS FULL_NAME,
```

```
E.BIRTH_DATE,
```

```
E.TITLE,
```

```
E.DEPARTMENT,
```

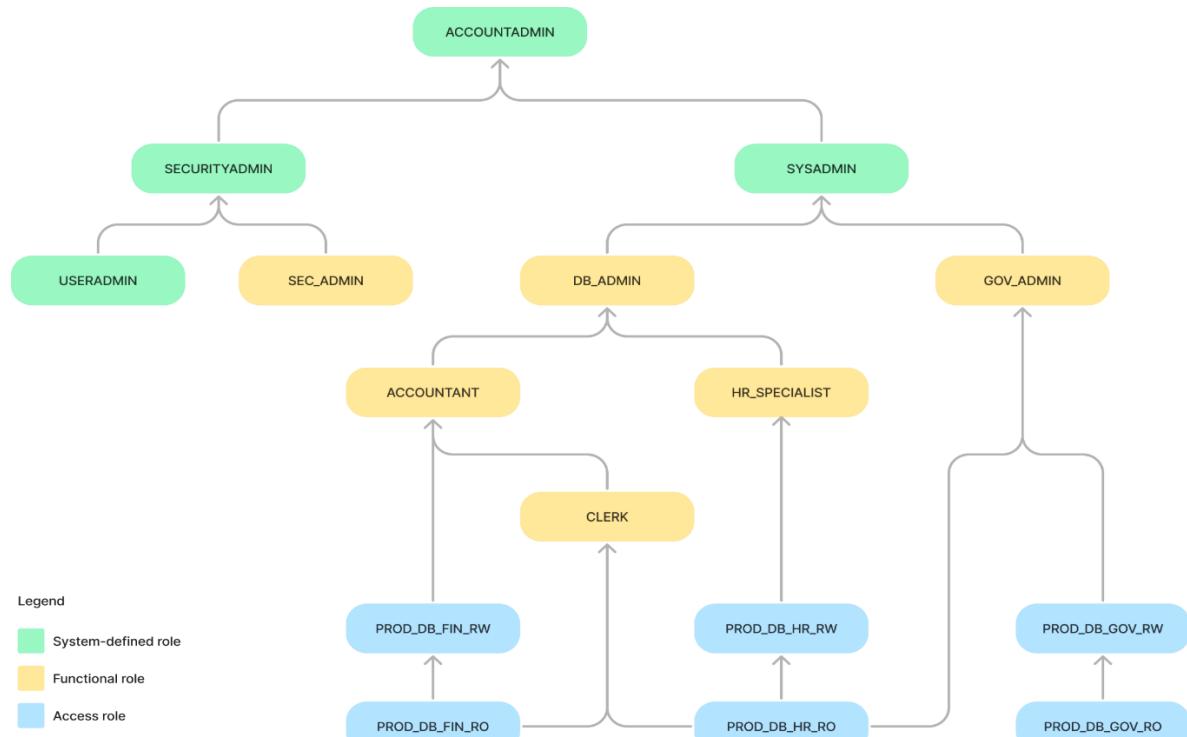
```

E.COUNTRY,
E.SOCIAL_INSURANCE_NUMBER,
PP.SALARY_AMOUNT,
SI.EFFECTIVE_DATE AS LAST_RAISE_DATE
FROM PROD_DB_HR.MAIN.EMPLOYEE E
LEFT JOIN LAST_RAISE SI
ON E.EMPLOYEE_ID = SI.EMPLOYEE_ID
LEFT JOIN PROD_DB_FIN.MAIN.PAYROLL PP
ON E.EMPLOYEE_ID = PP.EMPLOYEE_ID
AND PP.EFFECTIVE_DATE = SI.EFFECTIVE_DATE;

```

### Access control

The [role hierarchy](#) below implements the requirements outlined in the section above. At the top, you can find [system-defined roles](#) (green), which are created by default and come with privileges related to account management. It is not recommended to modify these roles to add entity-specific privileges. Instead, the recommended approach is to create “access roles” (blue) with the correct set of privileges on specific objects. Then, you grant these access roles to “functional roles” (yellow) based on requirements. Finally, you grant functional roles to users so that they can do their work on the databases.



## Admin roles

A user can be assigned more than one functional role, inheriting, in this way, the combined privileges of the assigned roles. For instance, in the role hierarchy, we defined the following administrative roles:

- DB\_ADMIN: This role includes privileges to create database objects and virtual warehouses in the account.
- GOV\_ADMIN: This role includes privileges to create tags, masking policies and row access policies in the account.
- SEC\_ADMIN: This role includes privileges to grant/revoke user access roles.

As stated in the governance requirements section, this separation of roles allows database administration assignment, governance and security functions to three separate users if needed or to a single system administrator. Make sure to ultimately assign any custom role hierarchy to SYSADMIN. Otherwise, the account administrator cannot [modify objects](#) created by a custom role.

The code below shows the definition of the admin roles:

```
--
-- Custom admin roles

--
USE ROLE SECURITYADMIN;
CREATE OR REPLACE ROLE DB_ADMIN;
CREATE OR REPLACE ROLE GOV_ADMIN;
CREATE OR REPLACE ROLE SEC_ADMIN;
GRANT ROLE DB_ADMIN, GOV_ADMIN TO ROLE SYSADMIN;
GRANT ROLE SEC_ADMIN TO ROLE SECURITYADMIN;

--
-- Grant privileges

--
-- DB_ADMIN

USE ROLE ACCOUNTADMIN;
GRANT CREATE DATABASE ON ACCOUNT TO ROLE DB_ADMIN;
GRANT CREATE WAREHOUSE ON ACCOUNT TO ROLE DB_ADMIN;
-- GOV_ADMIN

USE ROLE ACCOUNTADMIN;
GRANT APPLY MASKING POLICY ON ACCOUNT TO ROLE GOV_ADMIN;
```

```
GRANT APPLY ROW ACCESS POLICY ON ACCOUNT TO ROLE GOV_ADMIN;

GRANT APPLY TAG ON ACCOUNT TO ROLE GOV_ADMIN;

-- SEC_ADMIN

USE ROLE SECURITYADMIN;

GRANT CREATE USER ON ACCOUNT TO ROLE SEC_ADMIN;

GRANT CREATE ROLE ON ACCOUNT TO ROLE SEC_ADMIN;

GRANT MANAGE GRANTS ON ACCOUNT TO ROLE SEC_ADMIN;
```

### Access roles

These roles should be granted [privileges](#) on a specific database or account objects at a granular level. In the role hierarchy, we defined read-only (RO) and read/write (RW) access roles at the database level only. However, in a real-life scenario, you will probably need to define roles at the schema or “group of tables” level.

The code below shows the definition of the access roles for the PROD\_DB\_FIN database:

```
USE ROLE SEC_ADMIN;

-- PROD_DB_FIN

-- Grant read-only permissions on database PROD_DB_FIN to PROD_DB_FIN_RO

CREATE OR REPLACE ROLE PROD_DB_FIN_RO;

GRANT USAGE ON DATABASE PROD_DB_FIN TO ROLE PROD_DB_FIN_RO;

GRANT USAGE ON ALL SCHEMAS IN DATABASE PROD_DB_FIN TO ROLE PROD_DB_FIN_RO;

GRANT SELECT ON ALL TABLES IN DATABASE PROD_DB_FIN TO ROLE PROD_DB_FIN_RO;

GRANT SELECT ON FUTURE TABLES IN DATABASE PROD_DB_FIN TO ROLE PROD_DB_FIN_RO;

-- Grant read-write permissions on database PROD_DB_FIN to PROD_DB_FIN_RW

CREATE OR REPLACE ROLE PROD_DB_FIN_RW;

GRANT INSERT,UPDATE,DELETE ON ALL TABLES IN DATABASE PROD_DB_FIN TO ROLE
PROD_DB_FIN_RW;

GRANT INSERT,UPDATE,DELETE ON FUTURE TABLES IN DATABASE PROD_DB_FIN TO ROLE
PROD_DB_FIN_RW;

-- Create role hierarchy

GRANT ROLE PROD_DB_FIN_RO TO ROLE PROD_DB_FIN_RW;
```

## Functional roles

As we will soon see, many data governance checks involve verifying the current user's role. Therefore, it is essential to design a functional role hierarchy that aligns with business functions. In our role hierarchy, we defined a role for each business user: ACCOUNTANT, CLERK and HR\_SPECIALIST. Moreover, notice how the ACCOUNTANT role inherits the CLERK role, allowing it to read the same databases as the CLERK. In addition, it can update the PROD\_DB\_FIN database because it inherits the PROD\_DB\_FIN\_RW access role.

The code below shows the definition of the functional roles:

```
--
-- Functional roles

--

USE ROLE SEC_ADMIN;

-- ACCOUNTANT

CREATE OR REPLACE ROLE ACCOUNTANT;

GRANT USAGE ON WAREHOUSE COMPUTE_WH TO ROLE ACCOUNTANT;

-- Grant access roles

GRANT ROLE PROD_DB_FIN_RW TO ROLE ACCOUNTANT;

-- Create role hierarchy

GRANT ROLE ACCOUNTANT TO ROLE DB_ADMIN;

-- CLERK

CREATE OR REPLACE ROLE CLERK;

GRANT USAGE ON WAREHOUSE COMPUTE_WH TO ROLE CLERK;

-- Grant access roles

GRANT ROLE PROD_DB_HR_RO TO ROLE CLERK;

GRANT ROLE PROD_DB_FIN_RO TO ROLE CLERK;

-- Create role hierarchy

GRANT ROLE CLERK TO ROLE ACCOUNTANT;

-- HR_SPECIALIST

CREATE OR REPLACE ROLE HR_SPECIALIST;

GRANT USAGE ON WAREHOUSE COMPUTE_WH TO ROLE HR_SPECIALIST;

-- Grant access roles
```

```
GRANT ROLE PROD_DB_HR_RW TO ROLE HR_SPECIALIST;
```

-- Create role hierarchy

```
GRANT ROLE HR_SPECIALIST TO ROLE DB_ADMIN;
```

-- GOV\_ADMIN

-- Grant access roles

```
GRANT ROLE PROD_DB_GOV_RW TO ROLE GOV_ADMIN;
```

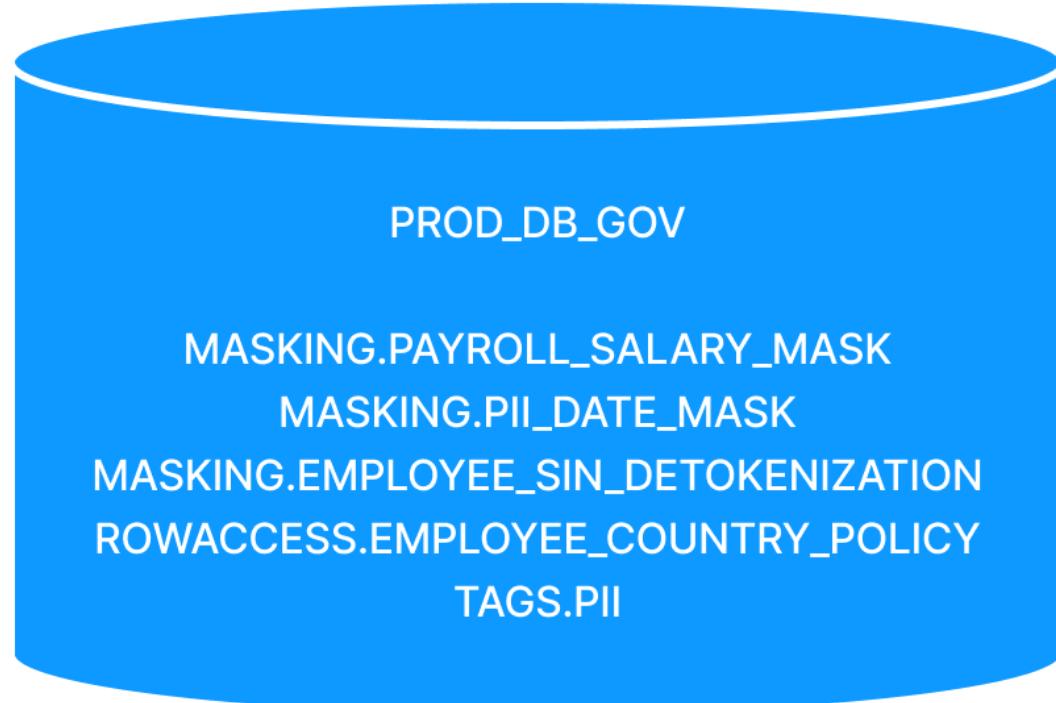
```
GRANT ROLE PROD_DB_HR_RO TO ROLE GOV_ADMIN;
```

### Data governance policies

The role hierarchy in the previous section defines what can be done on different objects and by whom. However, it doesn't restrict which records within a table a user can see or which values should be masked within a column. That's where the data governance policies in this section come into play.

All data governance policies and tags are stored in the PROD\_DB\_GOV database under three schemas: MASKING, ROWACCESS and TAGS. Putting all the policies and tags in a single database allows us to centralize them and better restrict access to them. Please note that only the GOV\_ADMIN role has read/write permissions on it.

The following diagram shows the PROD\_DB\_GOV database and its contents:



## Column-level security

We implemented two types of column-level security features to satisfy the salary and SIN columns requirements. We defined a [dynamic data masking](#) policy for the salary column that hides the value depending on the current user's role. The condition is straightforward: If the current user's role cannot see the value, then return a NULL. Snowflake supports masking policies on tables and views. You can find additional information about column data masking in our company blog in this [post](#).

The code below shows the definition of the dynamic data masking policy:

```
USE ROLE GOV_ADMIN;
USE WAREHOUSE COMPUTE_WH;
```

```
-- Dynamic data masking
```

```
CREATE OR REPLACE MASKING POLICY PROD_DB_GOV.MASKING.PAYROLL_SALARY_MASK AS (VAL
NUMBER(10, 2)) RETURNS NUMBER(10, 2) -> CASE
WHEN CURRENT_ROLE() IN ('ACCOUNTANT', 'HR_SPECIALIST') THEN VAL -- Only ACCOUNTANT and
HR_SPECIALIST can see the actual value
ELSE NULL
END;
ALTER TABLE PROD_DB_FIN.MAIN.PAYROLL
MODIFY COLUMN SALARY_AMOUNT SET MASKING POLICY
PROD_DB_GOV.MASKING.PAYROLL_SALARY_MASK;
```

We defined a masking policy for the SIN column that simulates decoding a value masked through [external tokenization](#). In this approach, sensitive data is “tokenized” outside Snowflake, and the value needs to be “detokenized” at query runtime. Therefore, you usually need to call a third-party API through an [external function](#). In this post, however, we just tokenized the SIN values using a base64 encoding function to keep things simple.

The code below shows the definition of the external detokenization masking policy:

```
USE ROLE GOV_ADMIN;
USE WAREHOUSE COMPUTE_WH;
```

```
-- External tokenization
```

```
CREATE OR REPLACE MASKING POLICY PROD_DB_GOV.MASKING.EMPLOYEE_SIN_DETOKENIZATION AS
(VAL VARCHAR) RETURNS VARCHAR -> CASE
```

```
WHEN CURRENT_ROLE() IN ('ACCOUNTANT', 'HR_SPECIALIST') THEN base64_decode_string(VAL) -- Using
internal function
```

```
-- WHEN CURRENT_ROLE() IN ('ACCOUNTANT', 'HR_SPECIALIST') THEN external_function_decode(VAL) --
Using external function (AWS Lambda)
```

```
ELSE VAL
```

```
END;
```

```
ALTER TABLE PROD_DB_HR.MAIN.EMPLOYEE
```

```
MODIFY COLUMN SOCIAL_INSURANCE_NUMBER SET MASKING POLICY
PROD_DB_GOV.MASKING.EMPLOYEE_SIN_DETOKENIZATION;
```

## Row-level security

Column-level security allows you to mask values on specific columns. On the other hand, row-level security will enable you to filter out records in a table based on a particular condition. You can apply both types of security features to your tables and views. In this post, we defined a [row access policy](#) restricting access to only Canadian employees for the CLERK role. If the condition to filter out records is complex, you can use a [mapping table](#) to map roles to allowed values.

The code below shows the definition of the row access policy:

```
USE ROLE GOV_ADMIN;
```

```
USE WAREHOUSE COMPUTE_WH;
```

```
-- Row access policies
```

```
CREATE OR REPLACE ROW ACCESS POLICY PROD_DB_GOV.ROWACCESS.EMPLOYEE_COUNTRY_POLICY
AS (COUNTRY VARCHAR) RETURNS BOOLEAN -> CASE
```

```
WHEN CURRENT_ROLE() IN ('CLERK') AND COUNTRY NOT IN ('Canada') THEN FALSE -- Hide employees
outside Canada for CLERK
```

```
ELSE TRUE
```

```
END;
```

```
ALTER TABLE PROD_DB_HR.MAIN.EMPLOYEE
ADD ROW ACCESS POLICY PROD_DB_GOV.ROWACCESS.EMPLOYEE_COUNTRY_POLICY ON (COUNTRY);
```

## Tag-based masking policies

Snowflake supports [object tagging](#) to help track sensitive data or resource usage on multiple types of database objects. You can find an overview of Snowflake's object tagging in this [post](#) on our company blog.

Snowflake provides a convenient way to apply masking policies to a group of related columns using tags. First, you apply masking policies to a tag and then attach the tag to the associated columns. Snowflake will check the guidelines for the tag and apply one that matches the column type. For instance, in this post, we defined a PII tag to track Personal Identifiable Information (PII) columns in our tables. Next, we defined a PII\_DATE\_MASK masking policy for values of type DATE and applied it to the tag.

Finally, we attached the PII tag to the BIRTH\_DATE column. As a result, the values on the BIRTH\_DATE column are masked based on the conditions in the masking policy. The same tag can be applied to other DATE columns for the same result. Similarly, you could mask an "address" column by applying the same technique but using a masking policy for the type STRING.

The code below shows the definition of the tag and the tag-based policy:

```
USE ROLE GOV_ADMIN;
USE WAREHOUSE COMPUTE_WH;
-- Tag-based masking
CREATE OR REPLACE TAG PROD_DB_GOV.TAGS.PII;
-- Apply the PII tag to the BIRTH_DATE column
ALTER TABLE PROD_DB_HR.MAIN.EMPLOYEE
ALTER COLUMN BIRTH_DATE SET TAG PROD_DB_GOV.TAGS.PII='true';
-- Define the masking policy for the DATE data type
CREATE OR REPLACE MASKING POLICY PROD_DB_GOV.MASKING.PII_DATE_MASK AS (VAL DATE)
RETURNS DATE -> CASE
WHEN CURRENT_ROLE() IN ('HR_SPECIALIST') THEN VAL
ELSE '1900-01-01'::DATE
END;
-- Apply the masking policy to the tag
```

```
ALTER TAG PROD_DB_GOV.TAGS.PII
```

```
SET MASKING POLICY PROD_DB_GOV.MASKING.PII_DATE_MASK;
```

## Data classification

Snowflake provides [data classification](#) functions to help you identify and classify PII data in your tables. The EXTRACT\_SEMANTIC\_CATEGORIES function scans a table's contents, giving you a result with the possible categories and their corresponding probabilities. The GOV\_ADMIN role can run this function as part of his audit process to help him identify columns with PII information. The function's result is on the EMPLOYEE table in the screenshot below. Note how the function returns nothing for the BIRTH\_DATE and SOCIAL\_INSURANCE\_NUMBER columns. That is due to the masking policies that we have put in place.

```
PROD_DB_GOV.PUBLIC *

1 USE ROLE GOV_ADMIN;
2 USE WAREHOUSE COMPUTE_WH;
3
4 SELECT
5 f.key::varchar AS column_name,
6 f.value:"privacy_category":varchar AS privacy_category,
7 f.value:"semantic_category":varchar AS semantic_category,
8 f.value:"extra_info":"probability":number(10,2) AS probability,
9 f.value:"extra_info":"alternates":variant AS alternates
10 FROM TABLE(FLATTEN(EXTRACT_SEMANTIC_CATEGORIES('PROD_DB_HR.MAIN.EMPLOYEE')::VARIANT)) AS f;
11
```

COLUMN_NAME	PRIVACY_CATEGORY	SEMANTIC_CATEGORY	...	PROBABILITY	ALTERNATES
1 BIRTH_DATE	null	null		null	[]
2 COUNTRY	QUASI_IDENTIFIER	COUNTRY		1	[]
3 DEPARTMENT	null	null		null	[ { "privacy_category":"QUASI_IDENTIFIER", "probability":"0.19", "semantic_category":"OCCUPATION" } ]
4 EMPLOYEE_ID	null	null		null	[]
5 FIRST_NAME	IDENTIFIER	NAME		1	[]
6 LAST_NAME	IDENTIFIER	NAME		1	[]
7 SOCIAL_INSURANCE_NUMBER	null	null		null	[]
8 TITLE	QUASI_IDENTIFIER	OCCUPATION		1	[]

## Policy verification

Snowflake provides the [POLICY\\_CONTEXT](#) function to easily verify the query results returned from tables and views protected with data governance policies. We will use it to easily switch roles and verify the returned results.

Let's query the EMPLOYEE\_PAYROLL\_VW view with the ACCOUNTANT role:

```

1 USE ROLE GOV_ADMIN;
2 USE WAREHOUSE COMPUTE_WH;
3
4 EXECUTE USING POLICY_CONTEXT(CURRENT_ROLE => 'ACCOUNTANT') AS SELECT * FROM PROD_DB_HR.MAIN.EMPLOYEE_PAYROLL_VW;

```

✓ Birth date is masked ✓ SIN I

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	FULL_NAME	BIRTH_DATE	TITLE	DEPARTMENT	COUNTRY	SOCIAL_ID
1	1	Jason	Chapman	Jason Chapman	1900-01-01	Director	Sales	Canada	999-999-
2	2	Jennifer	Villanueva	Jennifer Villanueva	1900-01-01	Graphic Designer	Marketing	Canada	999-999-
3	3	Wayne	Cooper	Wayne Cooper	1900-01-01	Software Developer	Product Development	USA	999-999-
4	4	Anna	Thompson	Anna Thompson	1900-01-01	Accountant	Finance	USA	999-999-

And now, let's query the view with the HR\_SPECIALIST role:

```

1 USE ROLE GOV_ADMIN;
2 USE WAREHOUSE COMPUTE_WH;
3
4 EXECUTE USING POLICY_CONTEXT(CURRENT_ROLE => 'HR_SPECIALIST') AS SELECT * FROM PROD_DB_HR.MAIN.EMPLOYEE_PAYROLL_VW;

```

✓ Birth date is visible ✓ SIN I

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	FULL_NAME	BIRTH_DATE	TITLE	DEPARTMENT
1	1	Jason	Chapman	Jason Chapman	1980-05-31	Director	Sales
2	2	Jennifer	Villanueva	Jennifer Villanueva	1996-03-14	Graphic Designer	Marketing
3	3	Wayne	Cooper	Wayne Cooper	2000-10-25	Software Developer	Product Development
4	4	Anna	Thompson	Anna Thompson	1971-06-28	Accountant	Finance

Finally, let's query the view with the CLERK role:

```

1 USE ROLE GOV_ADMIN;
2 USE WAREHOUSE COMPUTE_WH;
3
4 EXECUTE USING POLICY_CONTEXT(CURRENT_ROLE => 'CLERK') AS SELECT * FROM PROD_DB_HR.MAIN.EMPLOYEE_PAYROLL_VW;

```

✓ Birth date is masked ✓ SIN I

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	FULL_NAME	BIRTH_DATE	TITLE	DEPARTMENT
1	1	Jason	Chapman	Jason Chapman	1900-01-01	Director	Sales
2	2	Jennifer	Villanueva	Jennifer Villanueva	1900-01-01	Graphic Designer	Marketing

## INTERVIEW

### 1. What are the essential features of Snowflake?

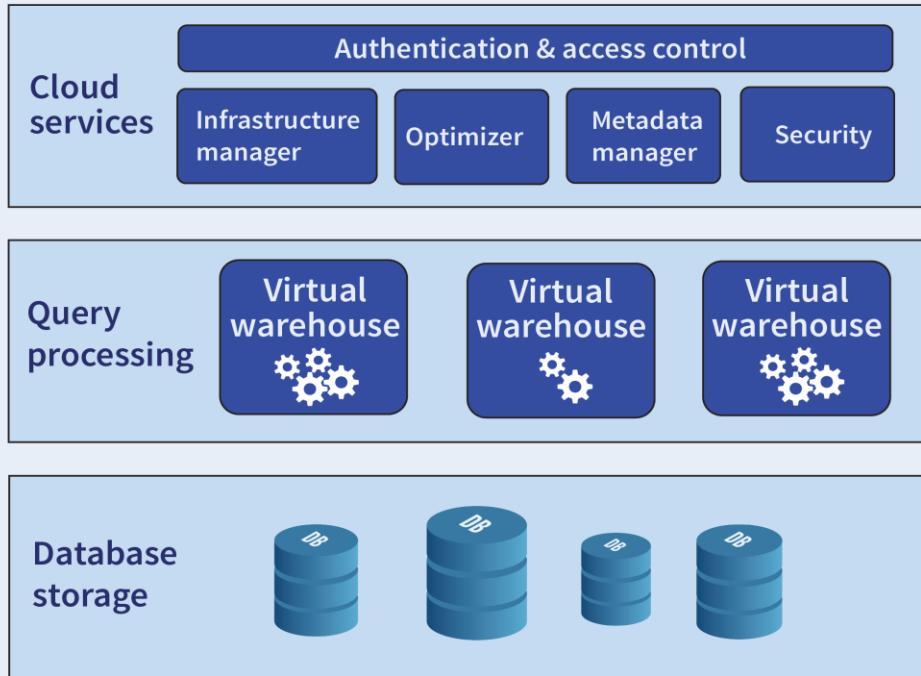
Snowflake has the following key features:

- With Snowflake, you can interact with the data cloud through a web interface. Users can navigate the web GUI to control their accounts, monitor resources, and monitor resources and system usage queries data, etc.
- Users can connect to Snowflake's data cloud using a wide range of client connectors and drivers. Among these connectors are Python Connector (an interface for writing Python applications to connect to Snowflake), Spark connector, NodeJS driver, .NET driver, JDBC driver for Java development, ODBC driver for C or C++ programming, etc.
- The core architecture of Snowflake enables it to operate on the public cloud, where it uses virtualized computing instances and efficient storage buckets for processing huge amounts of big data cost-effectively and scalable.
- Snowflake integrates with a number of big data tools, including business intelligence, machine learning, data integration, security, and governance tools.
- With advanced features such as simplicity, increased performance, high concurrency, and profitability, Snowflake is incomparable to other traditional data warehouse solutions.
- Snowflake supports the storage of both structured and semi-structured data (such as JSON, Avro, ORC, Parquet, and XML data).
- Snowflake automates cloud data management, security, governance, availability, and data resilience, resulting in reduced costs, no downtime, and better operational efficiency.
- With it, users can rapidly query data from a database, without having an impact on the underlying dataset. This allows them to receive data closer to real-time.
- Most DDL (Data Definition Language) and DML (Data Manipulation Language) commands in SQL are supported by the Snowflake data warehouse. Additionally, advanced DML, lateral views, transactions, stored procedures, etc., are also supported.

### 2. Explain Snowflake Architecture.

The Snowflake architecture is a hybrid of shared-disk (A common disk or storage device is shared by all computing nodes) and shared-nothing (Each computing node has a private memory and storage space) database architecture in order to combine the best of both. Snowflake utilizes a central data repository for persistent data, which is available to all compute nodes similar to a shared-disk architecture. But, equally, as with shared-nothing architectures, Snowflake uses massively parallel computing (MPP) clusters for query processing, in which each node stores part of the whole data set locally.

# VPC



The Snowflake architecture is divided into three key layers as shown below:

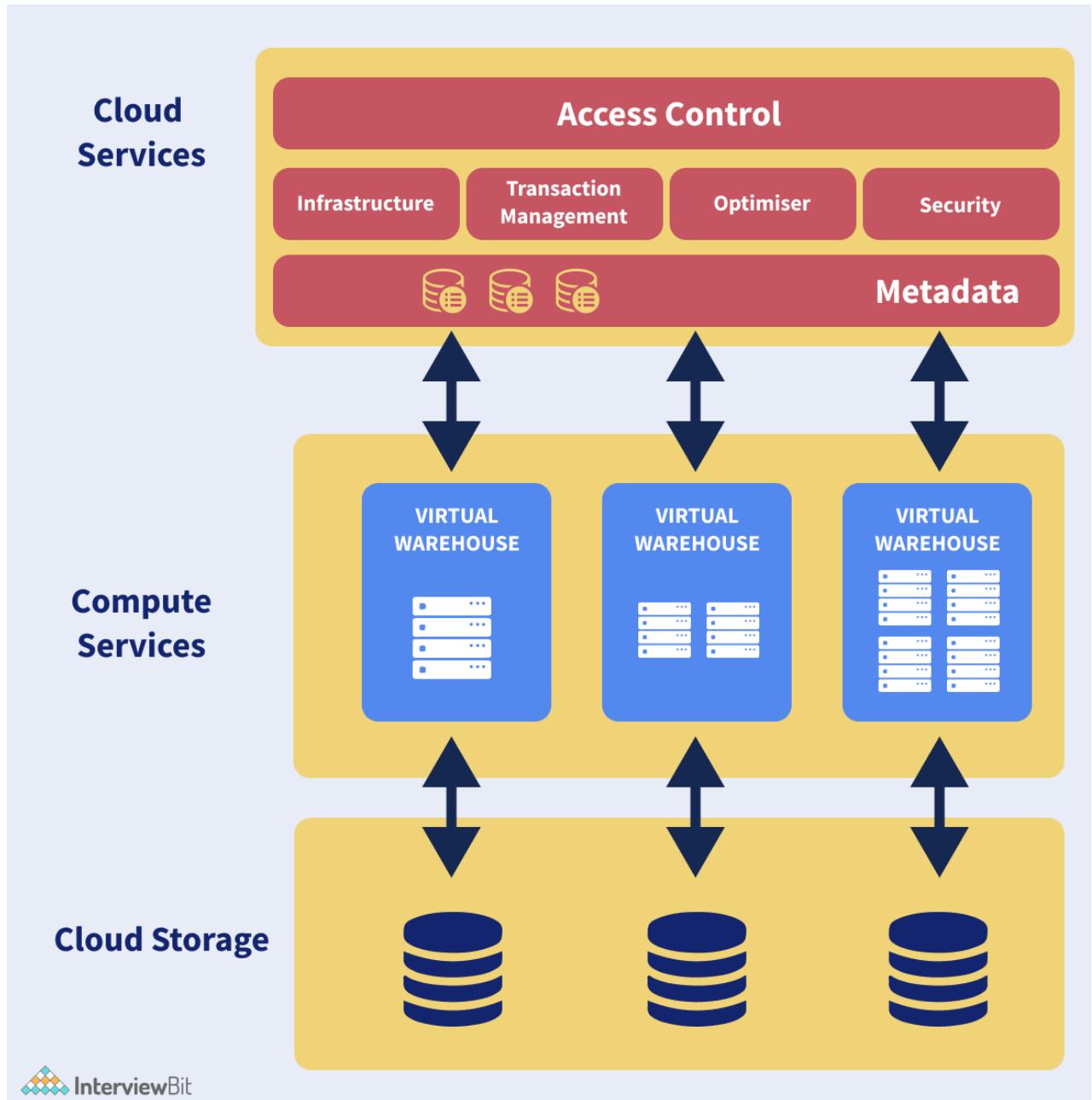
- **Database Storage Layer:** Once data has been loaded into Snowflake, this layer reorganizes that data into a specific format like columnar, compressed, and optimized format. The optimized data is stored in cloud storage.
- **Query Processing Layer:** In the processing layer, queries are executed using virtual warehouses. Virtual warehouses are independent MPP (Massively Parallel Processing) compute clusters comprised of multiple compute nodes that Snowflake allocates from cloud providers. Due to the fact that virtual warehouses do not share their compute resources with each other, their performance is independent of each other.
- **Cloud Services Layer:** It provides services to administer and manage a Snowflake data cloud, such as access control, authentication, metadata management, infrastructure management, query parsing, optimization, and many more.

### 3. What do you mean by virtual warehouse?

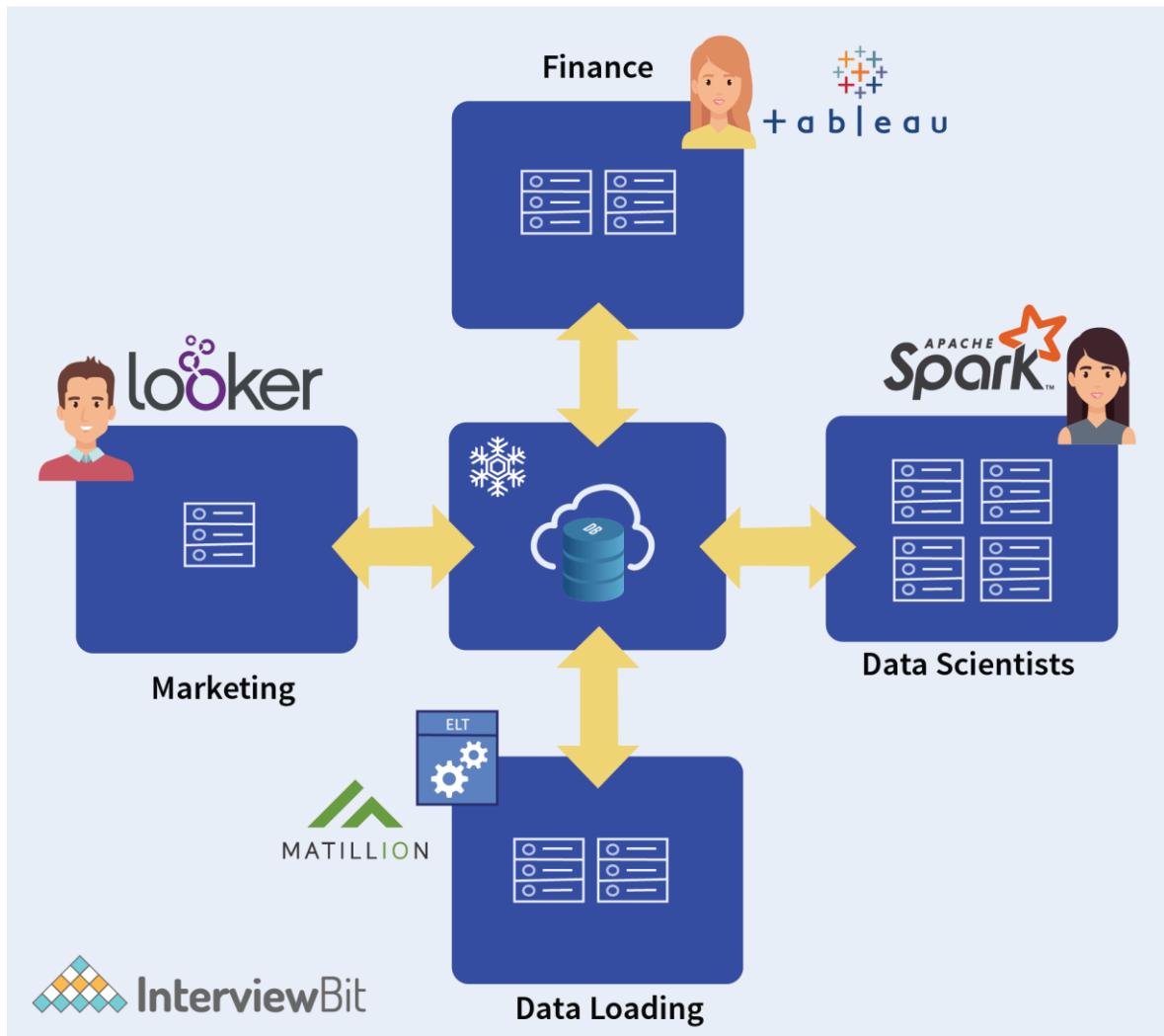
A virtual warehouse is basically a collection of computing resources (like CPU, memory, Solid state drive, etc.) that customers can access to run queries, load data, and perform other DML (Data Manipulation Language) and SQL (Structured Query Language) operations.

For example, it provides memory, temporary storage, and CPU resources that can be used for DML operations and SQL execution. You can use this independent compute cluster at any time and then turn it off when not needed. You are charged (paid) for each virtual warehouse you run, their size,

and how long they run. Virtual warehouses do not share their compute resources with each other, and therefore, their performance is independent of each other.



As shown in the following diagram, different groups of users can be assigned separate and dedicated virtual warehouses. Therefore, ETL processes can continuously load and execute complex transformation procedures on separate warehouses, ensuring no impact on [data scientists](#) or finance reports.



#### 4. Can you tell me how to access the Snowflake Cloud data warehouse?

Snowflake's data warehouse can be accessed using the following ways:

- ODBC Drivers (a driver for connecting to Snowflake).
- JDBC Drivers (a driver enabling a Java application to interact with a database).
- Python Libraries (for creating Python applications that connect to Snowflake and perform standard operations).
- Web User Interface (can be used for almost any task you can accomplish with SQL and the command line, such as: Creating and managing users and other account-level objects).
- SnowSQL Command-line Client (Python-based command-line interface to connect to Snowflake from Windows, Linux, and MacOS).

## 5. What is the difference between Snowflake and Redshift?

Cloud-based data warehouses are becoming increasingly popular, with Redshift and Snowflake being two of the biggest players. These are large data analytics databases capable of analyzing and reading vast amounts of data.

Snowflake vs Redshift -

Snowflake	Redshift
Despite similar on-demand pricing, Snowflake and Redshift package their features differently. In Snowflake's pricing structure, compute usage is separate from storage usage	In Redshift, both computer usage and storage usage are combined.
Snowflake is more robust than Redshift when it comes to JSON storage. In essence, Snowflake makes it possible to store and query JSON with built-in, native functions.	On the other hand, when JSON is loaded into Redshift, it is split into strings, making it more difficult to query and work with.
Snowflake offers security and compliance features specifically tailored to its editions so that your data is protected to the maximum level as per your data strategy.	The Redshift platform offers a wide range of encryption solutions
Data vacuuming and compression can be automated on Snowflake. It offers the best advantage as it automates much of the process, saving time and effort.	Data vacuuming and compression cannot be automated on Redshift, so the system requires more hands-on maintenance.

## 6. Explain stages in Snowflake.

Stages are locations in Snowflake where data is stored, and staging is the process of uploading data into a stage. Data that needs to be loaded or stored within Snowflake is stored either elsewhere in the other cloud regions like in AWS (Amazon Web Service) S3, GCP (Google Cloud Platform), or Azure, or is stored internally within Snowflake. When data is stored in another cloud region, this is known as an external stage; when it is stored inside a snowflake, it is known as an internal stage. Internal stages can be further categorized as follows:

- **User stages:** Each of these stages pertains to a specific user, so they'll be assigned to every user by default for storing files.
- **Table stages:** Each of these stages pertains to a specific database table, so they'll be assigned to every table by default.
- **Internal named stages:** Compared to the user or table stages, these stages offer a greater degree of flexibility. As these are some of the Snowflake objects, all operations that can be performed on objects can also be performed on internally named stages. These stages must be created manually and we can specify file formats when creating these stages.

## **7. Explain Snowpipe.**

In simple terms, Snowpipe is a continuous data ingestion service provided by Snowflake that loads files within minutes as soon as they are added to a stage and submitted for ingestion. Therefore, you can load data from files in micro-batches (organizing data into small groups/matches), allowing users to access the data within minutes (very less response time), rather than manually running COPY statements on a schedule to load large batches. By loading the data into micro-batches, Snowpipe makes it easier to analyze it. Snowpipe uses a combination of filenames and file checksums to ensure that only new data is processed.

### **Advantages of Snowpipe -**

- By eliminating roadblocks, Snowpipe facilitates real-time analytics.
- It is cost-effective.
- It is simple to use.
- There is no management required.
- It provides flexibility, resilience, and so on.

## **8. What do you mean by Snowflake Computing?**

The term snowflake computing refers to Snowflake's ability to provide instant, secure, and governed access to all data networks, along with its core architecture that enables multiple types of data workloads and offers a unified platform for modern data applications. In contrast to other data warehouses, Snowflake does not use a database or "big data" software platform such as Hadoop. Snowflake, however, combines an entirely new SQL query engine with a natively cloud-based architecture.

## **9. Which cloud platforms does Snowflake currently support?**

Snowflake currently supports the following cloud platforms:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure).

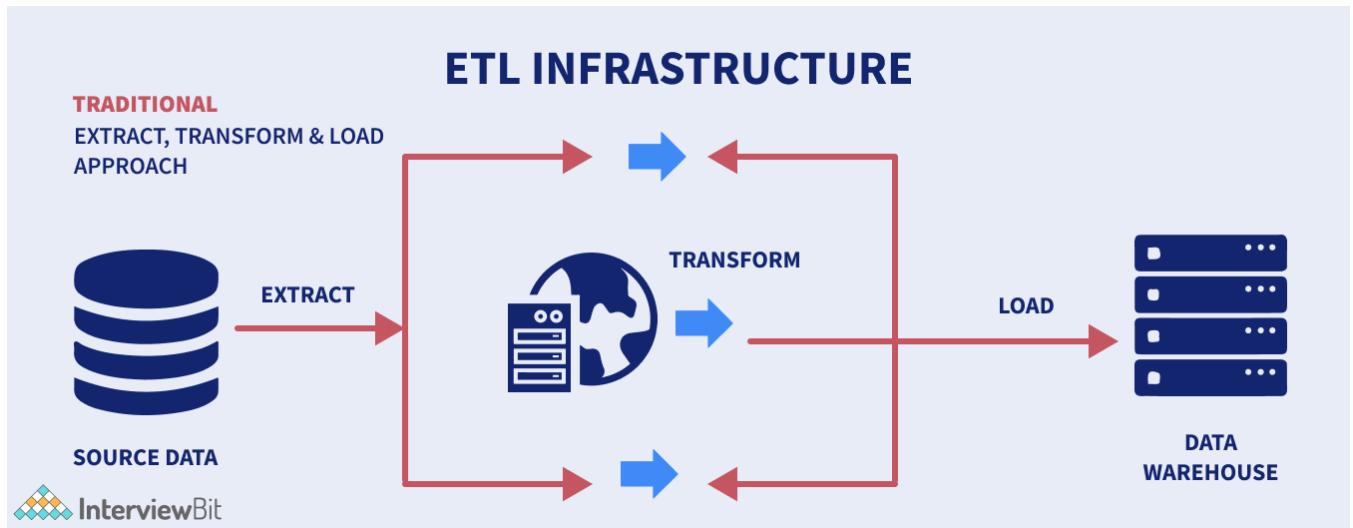
## **10. In Snowflake, how are data and information secured?**

Every organization considers data security to be one of its top priorities. The Snowflake platform adheres to the best security standards in the industry to encrypt and safeguard customer data. The platform provides the best key management features at no additional charge. To protect client data, Snowflake employs the following security measures:

- Snowflake automatically encrypts the data that it contains using a managed key.
- Snowflake is based on Transport Layer Security (TLS) to ensure data security between customers and servers.
- Depending on your cloud region, you can select a geographic location to store your data.

## 11. Is Snowflake an ETL (Extract, Transform, and Load) tool?

Yes, Snowflake is an **ETL** (Extract, Transform, and Load) tool, since it is performed in three steps, including:



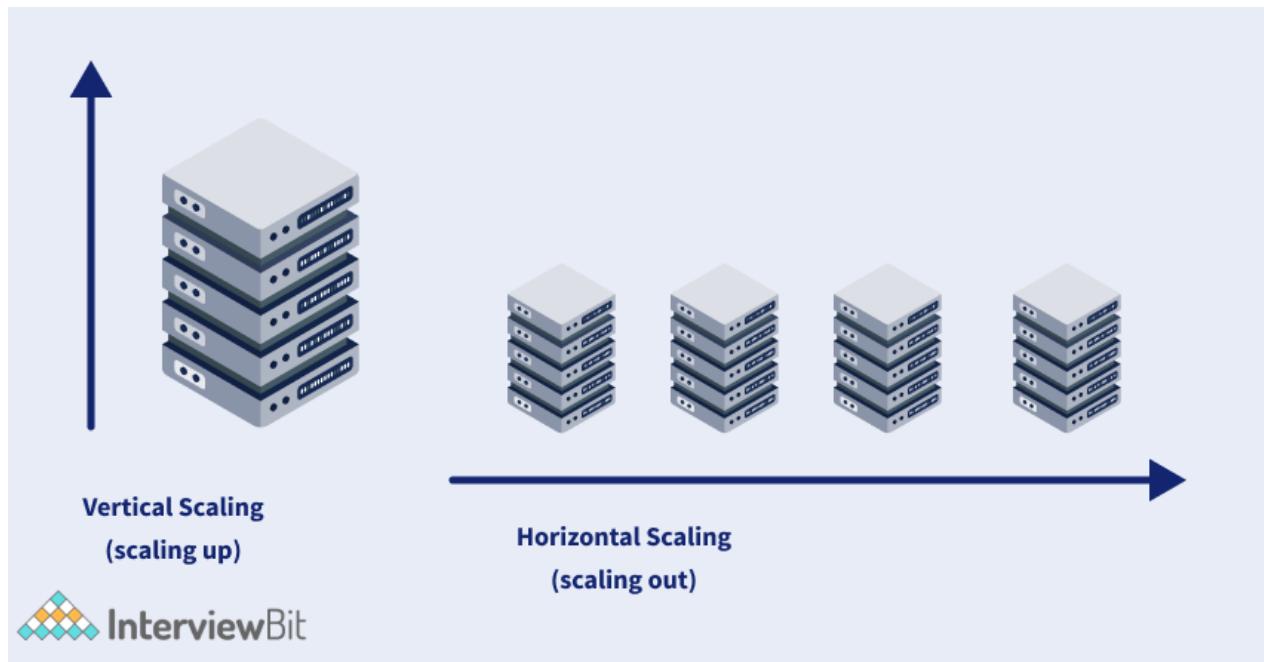
- The data is extracted from the source and saved in data files in a variety of formats including JSON, CSV, XML, and more.
- Loads data into a stage, either internal (Snowflake managed location) or external (Microsoft Azure, Amazon S3 bucket, Google Cloud).
- The COPY INTO command is used to copy data into the Snowflake database.

## 12. Which ETL tools are compatible with Snowflake?

Snowflake is compatible with the following ETL tools:

- Matillion
- Blendo
- Hevo Data
- StreamSets
- Etleap
- Apache Airflow, etc.

### 13. What do you mean by Horizontal and Vertical Scaling?



- **Horizontal Scaling:** Horizontal scaling increases concurrency by scaling horizontally. As your customer base grows, you can use auto-scaling to increase the number of virtual warehouses, enabling you to respond instantly to additional queries.
- **Vertical Scaling:** Vertical scaling involves increasing the processing power (e.g. CPU, RAM) of an existing machine. It generally involves scaling that can reduce processing time. Consider choosing a larger virtual warehouse-size if you want to optimize your workload and make it run faster.

### 14. Is snowflake OLTP (Online Transactional Processing) or OLAP (Online Analytical Processing)?

Snowflake is developed as an OLAP (Online Analytical Processing) database system, not as an OLTP (Online Transaction Processing) database system. In OLTP (Online Transaction Processing), data is collected, stored, and processed from real-time transactions, but in OLAP (Online Analytical Processing), complex queries are used to evaluate aggregated historical data from OLTP systems. Snowflake is not designed to handle much updating and inserting of small amounts of data like a transactional database would. Snowflake, for instance, cannot handle referential integrity because, even though it supports integrity and other constraints, they are not enforced except the NOT NULL constraint that is always enforced. Other constraints than NOT NULL are created as disabled constraints. However, depending on the use, we may also use it for online transaction processing (OLTP).

### 15. Snowflake is what kind of database?

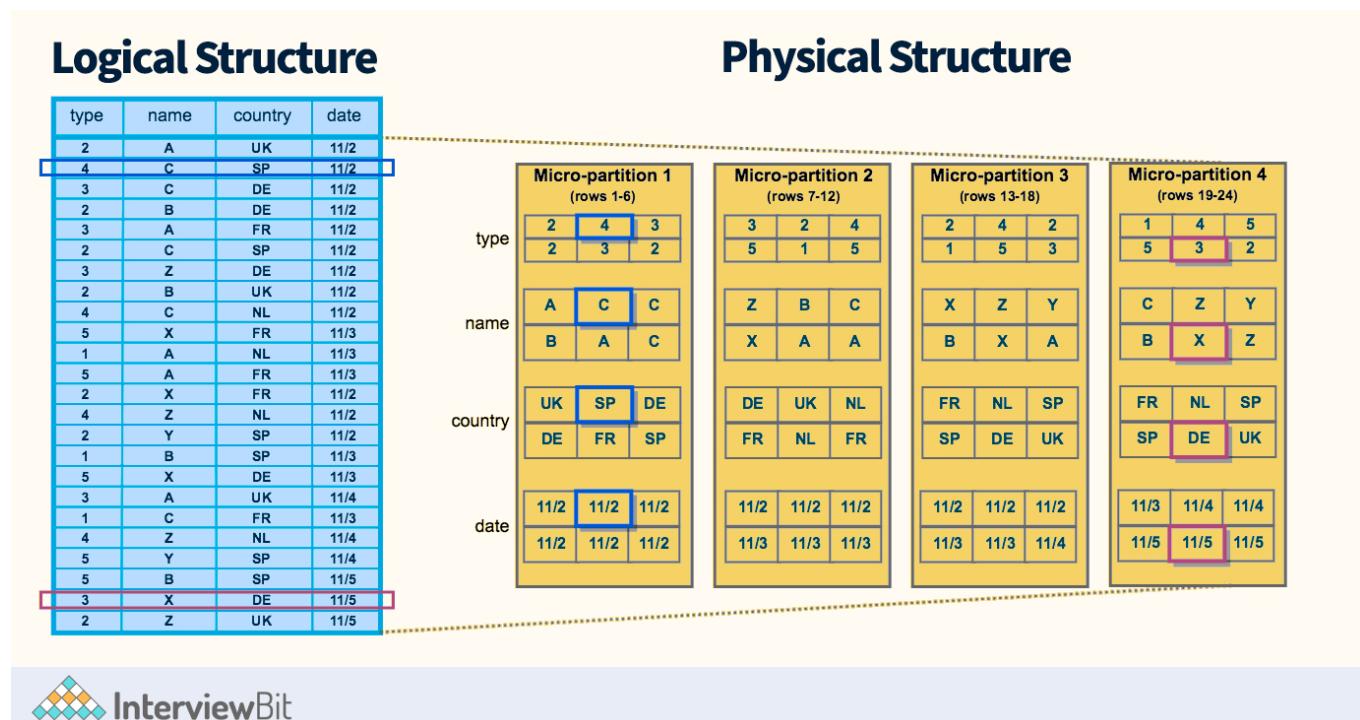
All Snowflake's features are built on top of SQL (Structured Query Language) databases. The data in this relational database system is stored in columns and it is compatible with other tools, including Excel and Tableau. As a SQL database, Snowflake contains a query tool, supports multi-statement transactions, provides role-based security, etc.

## 16. Explain in short about Snowflake Clustering.

In Snowflake, clustering is a type of data partitioning, where unique cluster keys are specified for each table. Cluster keys are subsets of a table's columns that are used to co-locate data within the table. These keys are appropriate for comprehensive tables. The process of managing clustered data in a table is known as re-clustering.

## 17. How is data stored in Snowflake? Explain Columnar Database.

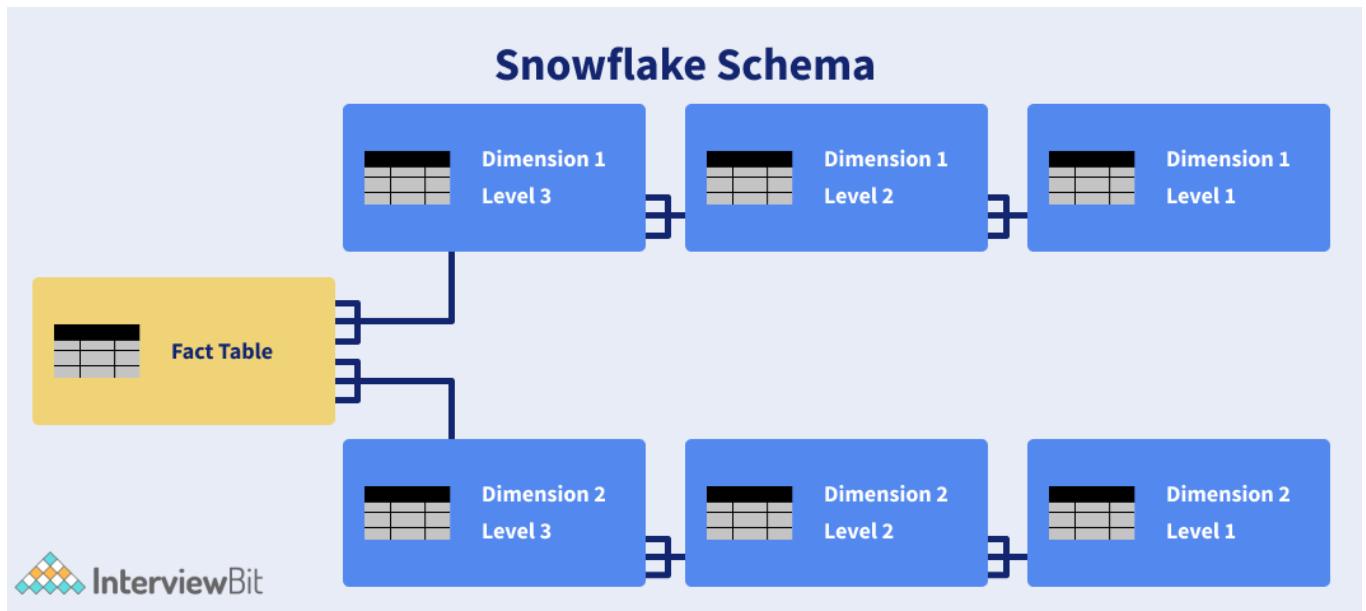
After data is loaded into Snowflake, it automatically reorganizes the data into a compressed, optimized, columnar format (micro-partitions). The optimized data is then stored in the cloud storage. Snowflake manages all aspects of storing these data, including file structure, size, statistics, compression, metadata, etc. Snowflake data objects aren't visible to customers or users. Users can only access data by performing SQL queries on Snowflake. Snowflake uses a columnar format to optimize and store data within the storage layer. With it, data is stored in columns instead of rows, allowing for an analytical querying method and improving database performance. With Columnar databases, business intelligence will be easier and more accurate. Compared to row-level operations, column-level operations are faster and use fewer resources than row-level operations.



Above is an example of a table with 24 rows divided into four micro-partitions, arranged and sorted by column. As the data is divided into micro-partitions, Snowflake can first remove those micro-partitions not relevant to the query, followed by pruning the remaining micro-partitions by column. The result is fewer records traversed, resulting in significantly faster response times.

## 18. Explain Schema in Snowflake.

The Snowflake Schema describes how data is organized in Snowflake. Schemas are basically a logical grouping of database objects (such as tables, views, etc.). Snowflake schemas consist of one fact table linked to many dimension tables, which link to other dimension tables via many-to-one relationships. A fact table (stores quantitative data for analysis) is surrounded by its associated dimensions, which are related to other dimensions, forming a snowflake pattern. Measurements and facts of a business process are contained in a Fact Table, which is a key to a Dimension Table, while attributes of measurements are stored in a Dimension Table. Snowflake offers a complete set of DDL (Data Definition Language) commands for creating and maintaining databases and schemas.

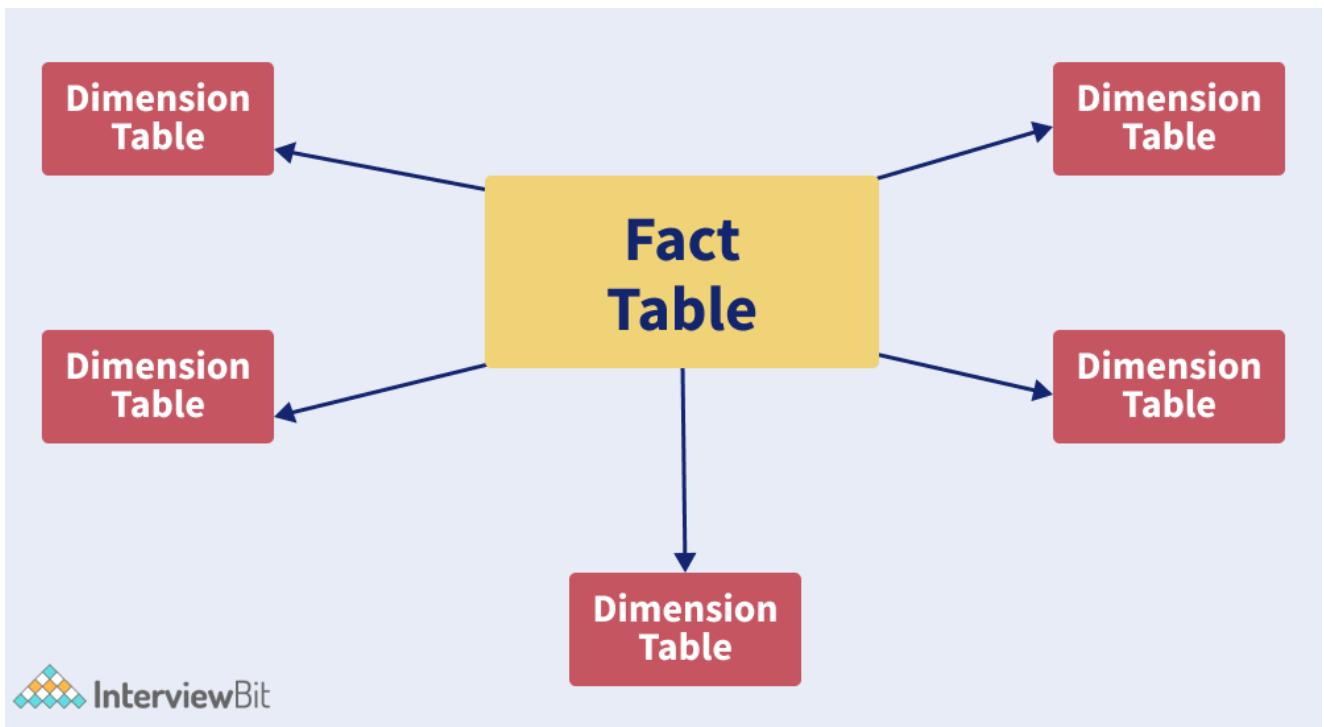


As shown in the above diagram, the snowflake schema has one fact table and two-dimension tables, each with three levels. Snowflake schemas can have an unlimited number of dimensions, and each dimension can have an infinite number of levels.

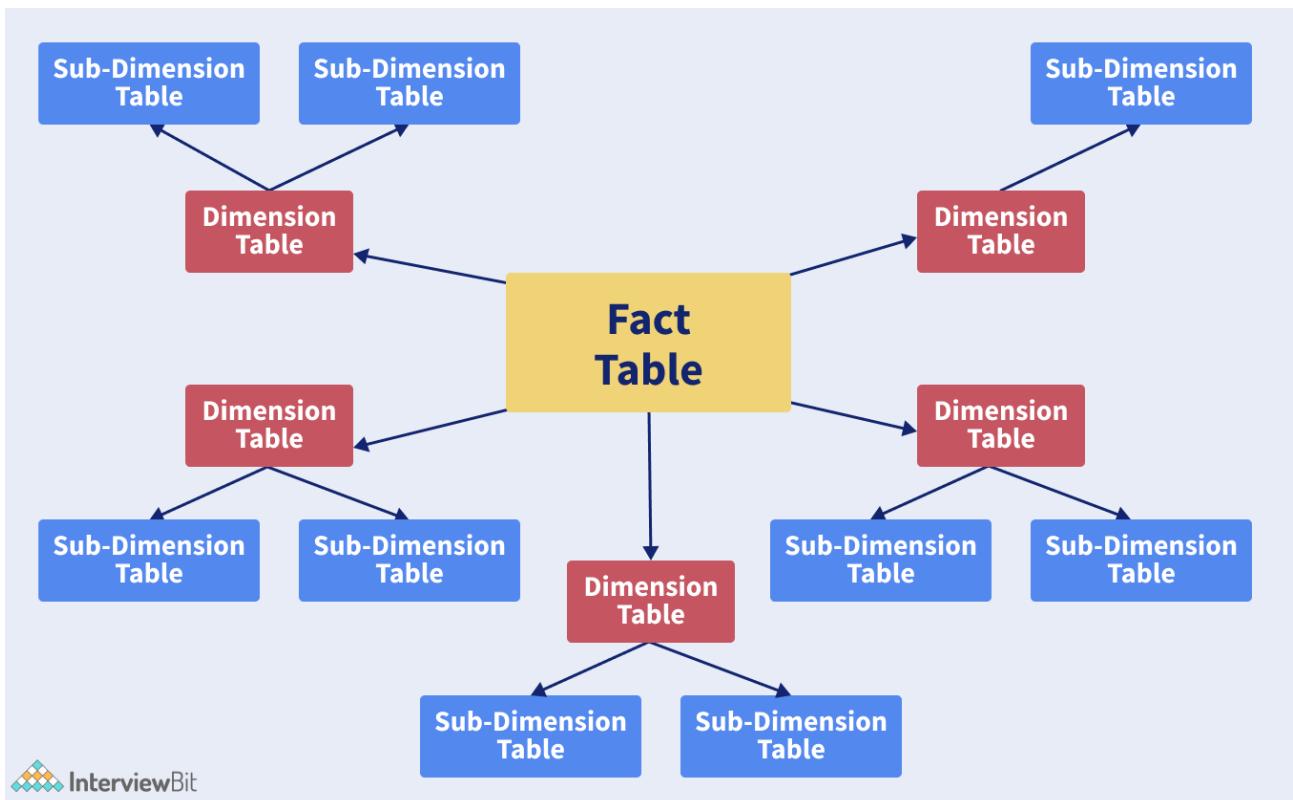
## 19. State difference between Star Schema and Snowflake Schema.

Schemas like Star and Snowflake serve as a logical description of the entire database, or how data is organized in a database.

- **Star Schema:** A star schema typically consists of one fact table and several associated dimension tables. The star schema is so named because the structure has the appearance of a star. Dimensions of the star schema have been denormalized. When the same values are repeated within a table, it is considered denormalization.



- **Snowflake Schema:** In a snowflake schema, the center has a fact table, which is associated with a number of dimension tables. Those dimension tables are in turn associated with other dimension tables. Snowflake schemas provide fully normalized data structures. Separate dimensional tables are used for the various levels of hierarchy (city > country > region).



## **20. Explain what is Snowflake Time travel and Data Retention Period.**

Time travel is a Snowflake feature that gives you access to historical data present in the Snowflake data warehouse. For example, suppose you accidentally delete a table named Employee. Using time travel, it is possible to go back five minutes in time to retrieve the data you lost. Data that has been altered or deleted can be accessed via Snowflake Time Travel at any point within a defined period. It is capable of performing the following tasks within a specific/defined period of time:

- Analyzing data manipulations and usage over a specified period of time.
- Restoring data-related objects (tables, schemas, and databases) that are accidentally lost (dropped)/
- Backup and duplication of data (clones) at or before specific points in the past.

As soon as the defined/specific period of time (data retention period) expires, the data moves into Snowflake Fail-safe and these actions/tasks cannot be performed.

## **21. What is Data Retention Period in Snowflake?**

The data retention period is a critical component of Snowflake Time Travel. When data in a table is modified, such as when data is deleted or objects containing data are removed, Snowflake preserves the state of that data before it was updated. Data retention specifies how many days historical data will be preserved, enabling Time Travel operations (SELECT, CREATE, CLONE, UNDROP, etc.) to be performed on it.

All Snowflake accounts have a default retention period of 1 day (24 hours). By default, the data retention period for standard objectives is 1 day, while for enterprise editions and higher accounts, it is 0 to 90 days.

## **22. Explain what is fail-safe.**

Snowflake offers a default 7-day period during which historical data can be retrieved as a fail-safe feature. Following the expiration of the Time Travel data retention period, the fail-safe default period begins. Data recovery through fail-safe is performed under best-effort conditions, and only after all other recovery options have been exhausted. Snowflake may use it to recover data that has been lost or damaged due to extreme operational failures. It may take several hours to several days for Fail-safe to complete data recovery.

## **23. Can you explain how Snowflake differs from AWS (Amazon Web Service)?**

Cloud-based data warehouse platforms like Snowflake and Amazon Redshift provide excellent performance, scalability, and business intelligence tools. In terms of core functionality, both platforms provide similar capabilities, such as relational management, security, scalability, cost efficiency, etc. There are, however, several differences between them, such as pricing, user experience and deployment options.

- There is no maintenance required with Snowflake as it is a complete SaaS (Software as a Service) offering. In contrast, AWS Redshift clusters require manual maintenance.
- The Snowflake security model uses always-on encryption to enforce strict security checks, while Redshift uses a flexible, customizable approach.
- Storage and computation in Snowflake are completely independent, meaning the storage costs are approximately the same as those in S3. In contrast, AWS bypasses this problem with a Red Shift spectrum and lets you query data that is directly available in S3. Despite this, it is not flawless like Snowflake.

#### **24. Could AWS glue connect to Snowflake?**

Yes, you can connect the Snowflake to AWS glue. AWS glue fits seamlessly into Snowflake as a data warehouse service and presents a comprehensive managed environment. Combining these two solutions makes data ingestion and transformation easier and more flexible.

#### **25. Explain how data compression works in Snowflake and write its advantages.**

An important aspect of data compression is the encoding, restructuring, or other modifications necessary to minimize its size. As soon as we input data into Snowflake, it is systematically compacted (compressed). Compressing and storing the data in Snowflake is achieved through modern data compression algorithms. What makes snowflake so great is that it charges customers by the size of their data after compression, not by the exact data.

Snowflake Compression has the following advantages:

- Compression lowers storage costs compared with original cloud storage.
- On-disk caches do not incur storage costs.
- In general, data sharing and cloning involve no storage expenses.

#### **26. Explain Snowflake caching and write its type.**

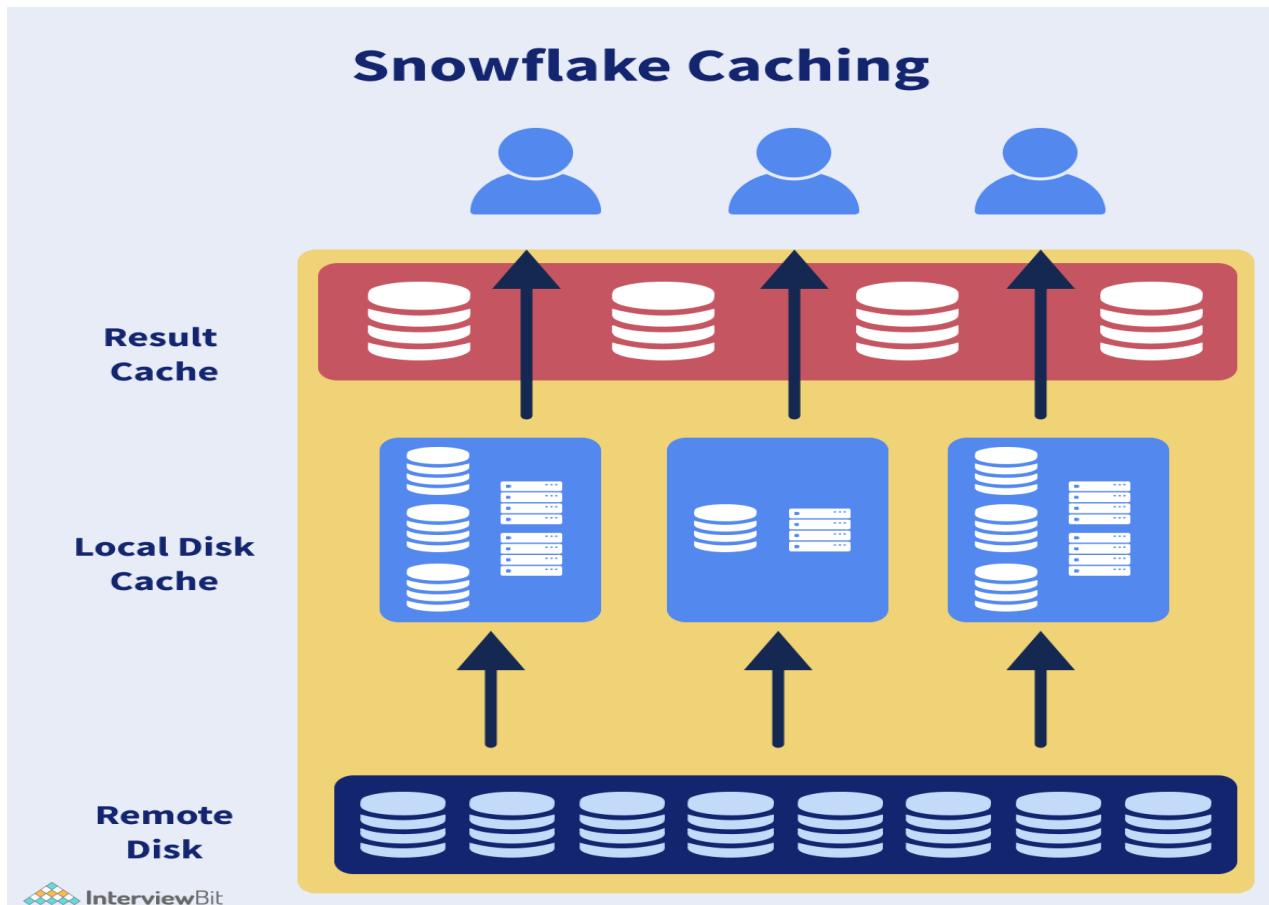
Consider an example where a query takes 15 minutes to run or execute. Now, if you were to repeat the same query with the same frequently used data, later on, you would be doing the same work and wasting resources.

Alternatively, Snowflake caches (stores) the results of each query you run, so whenever a new query is submitted, it checks if a matching query already exists, and if it did, it uses the cached results rather than running the new query again. Due to Snowflake's ability to fetch the results directly from the cache, query times are greatly reduced.

##### **Types of Caching in Snowflake**

- Query Results Caching: It stores the results of all queries executed in the past 24 hours.
- Local Disk Caching: It is used to store data used or required for performing SQL queries. It is often referred to as
- Remote Disk Cache: It holds results for long-term use.

The following diagram visualizes the levels at which Snowflake caches data and results for subsequent use.



## 27. What are different snowflake editions?

Snowflake offers multiple editions to meet your organization's specific needs. In every subsequent edition, either new features are introduced or a higher level of service is provided. It's easy to switch editions as your organization's needs change.

The following are some of the Snowflake Editions:

STANDARD	ENTERPRISE	BUSINESS CRITICAL	VIRTUAL PRIVATE SNOWFLAKE (VPS)
			
Complete SQL data warehouse Secure Data Sharing across regions/clouds Business hour support M-F 1 day of time travel Always-on enterprise grade encryption in transit and at rest Customer-dedicated virtual warehouses Federated authentication Database replication	<b>Premier +</b> Multi-cluster warehouse Up to 90 days of time travel Annual rekey of all encrypted data Materialized views AWS PrivateLink available for extra fee	<b>Enterprise +</b> HIPAA support PCI compliance Data encryption everywhere Tri-Secret Secure using customer-managed keys AWS PrivateLink support Enhanced security policy Database failover and fallback for business continuity	<b>Business Critical +</b> Customer-dedicated virtual servers wherever the encryption key is in memory Customer-dedicated metadata store Additional operational visibility
<b>\$2.50</b> cost per credit	<b>\$3.70</b> cost per credit	<b>\$5.00</b> cost per credit	



InterviewBit

- **Standard Edition:** This is Snowflake's entry-level offering, which allows full access to all of Snowflake's standard features. This edition strikes a good balance between features, level of support, and price.
- **Enterprise Edition:** With Enterprise Edition, you get all the features and services of Standard Edition, but with some extra features designed to meet the needs of large-scale enterprises.
- **Business-critical Edition:** Previously known as Enterprise for Sensitive Data (ESD), Business Critical Edition offers even more advanced levels of data security for organizations that deal with highly sensitive data. This edition includes all the features and services of the Enterprise Edition, plus enhanced security and data protection.
- **Virtual Private Snowflake (VPS):** With this edition, organizations with strict security requirements, like financial institutions and companies that collect, analyze, and share highly sensitive data, can get the highest level of security.

## 28. What do you mean by zero-copy cloning in Snowflake?

Zero-copy cloning is one of the great features of Snowflake. It basically allows you to duplicate the source object without making a physical copy of it or adding additional storage costs to it. A snapshot of the data in a source object is taken when a clone (cloned object) is created, and it is made available to the cloned object. Cloned objects are independent of the source object and are therefore writable, and any changes made to either object are not reflected in the other. The keyword CLONE allows you to copy tables, schemas, databases without actually copying any data.

### Zero copy cloning syntax in Snowflake

- In order to clone an entire production database for development purposes:

```
CREATE DATABASE Dev CLONE Prod;
```

- In order to clone a schema

```
CREATE SCHEMA Dev.DataSchema1 CLONE Prod.DataSchema1;
```

- In order to clone a single table:

```
CREATE TABLE C CLONE Dev.public.C;
```

## 29. Explain what do you mean by data shares in Snowflake?

Data sharing via Snowflake allows organizations to share data quickly and securely between Snowflake accounts. Database objects that are shared between snowflake accounts are only readable and can't be changed or modified. The three types of sharing are as follows:

- Data sharing between management units
- Data sharing between functional units
- Sharing data between geographically dispersed areas.

## 30. What is the best way to remove a string that is an anagram of an earlier string from an array?

For instance, an array of strings arr is given. The task is to remove all strings that are anagrams of an earlier string, then print the remaining array in sorted order.

- Examples:

*Input:* arr[] = { "Scaler", "Lacers", "Accdemey", "Academy" }, N = 4

*Output:* ["Scaler", "Academy"]

*Explanation:* "Listen" and "Silent" are anagrams, so we remove "Silent". Similarly, "Scaler" and "Lacers" are anagrams, so we remove "Lacers".

- Code Implementation

```
import java.util.*;
class InterviewBit{
 // Function to remove the anagram String
 static void removeAnagrams(String arr[], int N)
 {
 // vector to store the final result
 Vector ans = new Vector();
 // A data structure to keep track of previously encountered Strings
 HashSet found = new HashSet ();
 for (int i = 0; i < N; i++) {
 String word = arr[i];
 // Sort the characters of the current String
 word = sort(word);
 // Check if the current String is not present in the hashmap
 // Then, push it into the resultant vector and insert it into the hashmap
 }
 }
}
```

```

 if (!found.contains(word)) {
 ans.add(arr[i]);
 found.add(word);
 }
 }
 // Sort the resultant vector of Strings
 Collections.sort(ans);
 // Print the required array
 for (int i = 0; i < ans.size(); ++i) {
 System.out.print(ans.get(i)+ " ");
 }
}
static String sort(String inputString)
{
 // convert input string to char array
 char tempArray[] = inputString.toCharArray();
 // sort tempArray
 Arrays.sort(tempArray);
 // return new sorted string
 return new String(tempArray);
}
// Driver code
public static void main(String[] args)
{
 String arr[]={ "Scaler", "Lacers", "Accdemy", "Academy" };
 int N = 4;
 removeAnagrams(arr, N);
}
}

```

- Output:

Scaler Academy

### 31. What do we need to do to create temporary tables?

In the CREATE TABLE DDL, specify the TEMPORARY keyword (or the TEMP abbreviation) to create a temporary table. The following syntax must be used to create temporary tables:

Create temporary table mytable (id number, creation\_date date);

## **1. What are the different types of Stages?**

*Stages are commonly referred to as the storage platform used to store the files. In Snowflake, there are two types of stages:*

*1. Internal stage — Resides in the Snowflake storage*

*2. External stage — Resides in any of the cloud object storage (AWS S3, Azure Blob, GCP bucket )*

*Data can be retrieved from the stage or transferred to the stage using the COPY INTO command.*

*For BULK loading you can use COPY INTO and for continuous data loading you need to use*

*SNOWPIPE, an autonomous service provided by Snowflake.*

*To load data from the local file system into snowflake you can use the PUT command.*

## **2. What is Unique about Snowflake Cloud Data Warehouse?**

*Snowflake has introduced many unique features that are not been used in any of the other data warehouses currently in the market.*

### ***1. Totally cloud-agnostic ( SAAS )-***

*Snowflake relies on 3 cloud service providers (AWS, Azure, GCP) for its underlying infrastructure.*

*It provides true SAAS functionality where the user does not require to download or install any*

*kind of software to use snowflake or need to worry about any kind of hardware.*

### ***2. Decoupled storage and compute-***

*By decoupled it means storage and computes are work separately and work collaboratively with the interface provided by the cloud provider. This helps in decreasing usage costs where the user pays only what he is using.*

### ***3. Zero copy cloning-***

*This feature is used to take a snapshot of the table at the current instance to take a backup of the table. The snapshot taken will not consume any physical space in the data storage unless any changes have been done on the clone object. This will occupy the same columnar partition used by the source table. Once changes are done on the cloned object they will be stored in the different micro partitions.*

### ***4. Secure data sharing-***

*This feature provides secure sharing of the data with different snowflake accounts or users outside of the snowflake account. By secure, it means you can assign authorized users to access any particular table in order to keep the table secured from the rest of the snowflake users. The shared objects are always in Read-only mode. You can create a Reader account to share data with the user who is not using Snowflake.*

## **5. Supports semi-structured data-**

*Snowflake supports file formats such as JSON, AVRO, ORC, PARQUET, and XML. The variant data type is used to load semi-structured data into snowflake. Once loaded it can be separated into multiple columns as a table.*

*The variant has a limit of 16MB for an individual row. Flatten function is used to split the nested attributes into separate columns.*

## **6. Scalability-**

*As Snowflake is built upon cloud infrastructure, it uses cloud services for storage and computing.*

*The warehouse is a VM that is used to carry out the computation required to execute any query. This enables users the ability to scale up resources when they need large amounts of data to be loaded faster and scale back down when the process is finished without any interruption to service.*

## **7. Time-travel and Failsafe-**

*Time-travel is to retrieve snowflake objects which are removed/dropped from snowflake. You can read/retrieve data that is deleted within a permissible time frame using time travel.*



### **CDP lifecycle**

*Using Time Travel, you can perform the following actions within a defined period of time:*

1. Query data in the past that has since been updated or deleted.
2. Create clones of entire tables, schemas, and databases at or before specific points in the past.
3. Restore tables, schemas, and databases that have been dropped.

### **3. What are the different ways to access the Snowflake Cloud Data warehouse?**

*Snowflake provides WebUI to access snowflake as well as SnowSQL to execute SQL queries and perform all DDL and DML operations including data loading and unloading. It also provides native connectors for Python, Spark, Go, Nodejs, JDBC, and ODBC.*

### **4. What are the data security features in Snowflake?**

*Snowflake provides below security features:*

1. *Data encryption*
2. *Object-level access*
3. *RBAC*
4. *Secure data sharing*
5. [Masking policies](#) for sensitive data

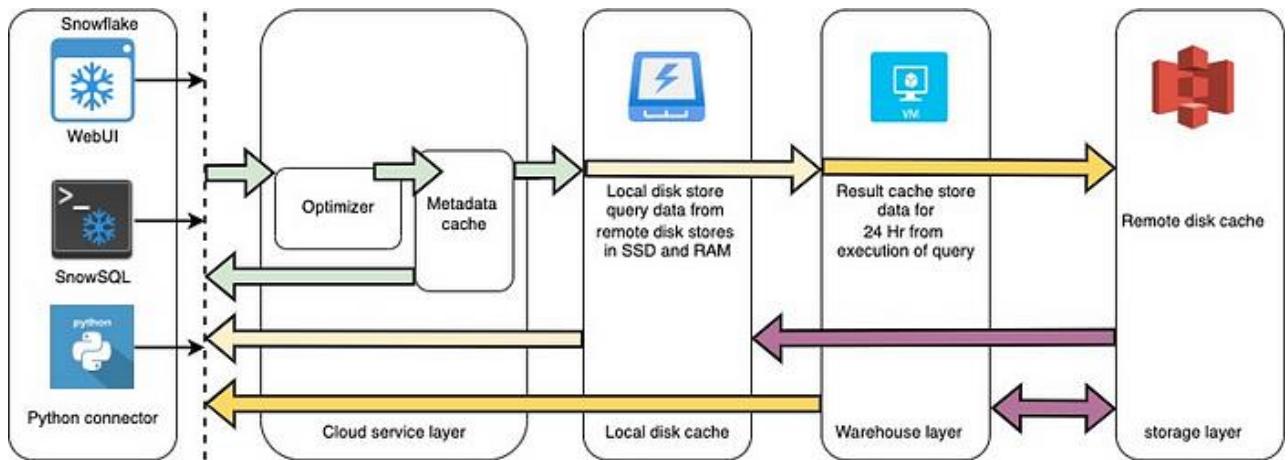
### **5. What are the benefits of Snowflake Compression?**

*Snowflake stores files in storage as compressed by default as gzip format which helps to reduce the storage space occupied by that file also improves the data loading and unloading performance. It also detects compressed file formats such as gzip,bzip2,deflate,raw\_deflate.*

### **6. What is Snowflake Caching? What are the different types of caching in Snowflake?**

*It comprises three types of caching :*

1. ***Result cache***- This holds the results of every query executed in the past 24 hours.
2. ***Local disk cache***- This 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.
3. ***Remote cache***- Which holds the long-term storage. This level is responsible for data resilience, which in the case of Amazon Web Services, means 99.99999999% durability. Even in the event of an entire data center failure.



Snowflake cache

## 7. Is there a cost associated with Time Travel in Snowflake?

*Yes, Time travel is the feature provided by snowflake to retrieve data that is removed from Snowflake databases.*

*using time travel you can do :*

1. *Query data in the past that has since been updated or deleted.*
2. *Create clones of entire tables, schemas, and databases at or before specific points in the past.*
3. *Restore tables, schemas, and databases that have been dropped.*

*Once the Time travel period is over, data is moved to the Fail-safe zone.*

*For the snowflake standard edition, the default Time travel period is 1.*

*For the snowflake Enterprise edition,*

*for transient and temp DB, schema, tables, the default time travel period is 1.*

*for permanent DB, schema, tables, and views, the default time travel can ranges from 1 to 90 days.*

## 8. What is fail-safe in Snowflake

*When the time-travel period elapses, removed data moves to Fail-safe zone of 7 days for Ent. edition snowflake and above. Once data went to Failsafe, we need to contact Snowflake in order to restore the data. It may take from 24 hrs to days to get the data. The charges will occur from where the state of the data is changed on basis of 24 Hr.*

## 9. What is the difference between Time-Travel vs Fail-Safe in Snowflake

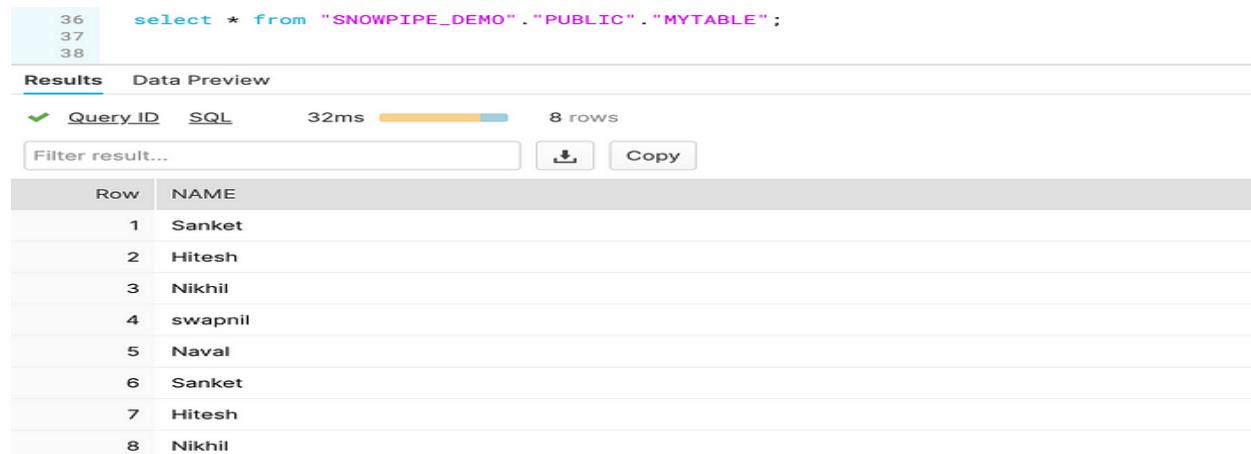
*Time travel has a time period ranging from 0 to 90 days for permanent DB, schema, and tables where Fail safe time is of 7 days only.*

*Once the table/schema is dropped from the SF account it will get into Time travel according to the time travel duration of that object (0–90) days.*

*Once TT is elapsed, objects move into the Fail-safe zone.*

Snowflake provides us with 3 methods of time travel –

- Using Timestamp** — We can do time travel to any point of time before or after the specified timestamp.
- Using Offset** — We can do time travel to any previous point in time.
- Using Query ID** — We can do time travel to any point of time before or after the specified Query ID.



The screenshot shows a Snowflake query results page. At the top, there are three numbered lines: 36, 37, and 38. Below them is the SQL query: `select * from "SNOWPIPE_DEMO"."PUBLIC"."MYTABLE";`. The results tab is selected, showing a table with two columns: Row and NAME. The data is as follows:

Row	NAME
1	Sanket
2	Hitesh
3	Nikhil
4	swapnil
5	Naval
6	Sanket
7	Hitesh
8	Nikhil

Below the table, there are buttons for 'Filter result...', 'Download', and 'Copy'.

Now lets drop the table :

```
44 drop table "SNOWPIPE_DEMO"."PUBLIC"."MYTABLE";
45
46
```

Results Data Preview

✓ [Query ID](#) [SQL](#) 58ms 1 rows

Filter result... [Download](#) [Copy](#)

Row	status
1	MYTABLE successfully dropped.

Try reading the table again:

```
35
36 select * from "SNOWPIPE_DEMO"."PUBLIC"."MYTABLE";
37
38
```

Results Data Preview

✗ [Query ID](#) [SQL](#) 25ms

SQL compilation error: Object 'SNOWPIPE\_DEMO.PUBLIC.MYTABLE' does not exist or not authorized.

Now we can recover the dropped table using UNDROP :

```

52 undrop table "SNOWPIPE_DEMO"."PUBLIC"."MYTABLE";
53
54

```

**Results** Data Preview

✓ [Query\\_ID](#) [SQL](#) 47ms 1 rows

Filter result... [Download](#) [Copy](#)

Row	status
1	Table MYTABLE successfully restored.

and then we can read the data from the table again :

```

35
36 select * from "SNOWPIPE_DEMO"."PUBLIC"."MYTABLE";
37
38

```

**Results** Data Preview

✓ [Query\\_ID](#) [SQL](#) 36ms 8 rows

Filter result... [Download](#) [Copy](#)

Row	NAME
1	Sanket
2	Hitesh
3	Nikhil
4	swapnil
5	Naval
6	Sanket
7	Hitesh
8	Nikhil

## 10. How does zero-copy cloning work and what are its advantage

Zero copy cloning is just like a creating clone of the snowflake object.

You can create clones of SF objects such as DB, schema, table, stream, stage, file formats, sequence, and task.

when you create a clone, Snowflake will point the metadata of the source object to cloned object depicting cloning until you make any changes to cloned object.

1. Main advantage of this is it creates a copy of the object in less time.
2. It does not consume any extra space if no updates happen on the cloned object.
3. fast way to take backup of any object.

Syntax:

```
create table orders_clone clone orders;
```

## **11. What are Data Shares in Snowflake?**

Data sharing is the feature provided by snowflake to share data across snowflake accounts and people outside of the snowflake accounts. You can share data according to the customized datasets shared. For people outside snowflake, you need to create a reader account with access to only read the data.

***Below are the objects that can be shared:***

*Tables*

*External tables*

*Secure views*

*Secure materialized views*

*Secure UDFs*

There are two types of users:

1. Data provider: The provider creates a share of a database in their account and grants access to specific objects in the database. The provider can also share data from multiple databases, as long as these databases belong to the same account.
2. Data consumer: On the consumer side, a **read-only** database is created from the share. Access to this database is configurable using the same, standard role-based access control that Snowflake provides for all objects in the system.

## 12. What is Horizontal scaling vs Vertical scaling in Snowflake.

Snowflake Enterprise and the above versions support a multi-cluster warehouse where you can create a multi-cluster environment to handle scalability. The warehouse can be scaled horizontally or vertically.

The multicluster warehouse can be configured in two ways :

1. **Maximized** mode: Where min amount and max amount of clusters are the same but (1 < cluster size ≤10)

**Create Warehouse**

Name \*

Size  ▼

Learn more about virtual warehouse sizes [here](#)

Maximum Clusters  ▼

Multi-cluster warehouses improve the query throughput for high concurrency workloads.

Minimum Clusters  ▼

The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.

Scaling Policy  ▼

The policy used to automatically start up and shut down clusters.

Auto Suspend  ▼

The maximum idle time before the warehouse will be automatically suspended.

Auto Resume ?

Comment

[Show SQL](#) Cancel Finish

**Maximized mode**

2. Auto-Scale mode: Where min amount and max amount of clusters are different ( min = 2 and Max=10)

**Create Warehouse**

Name \*

Size  ▼

Learn more about virtual warehouse sizes [here](#)

Maximum Clusters  ▼

Multi-cluster warehouses improve the query throughput for high concurrency workloads.

Minimum Clusters  ▼

The number of active clusters will vary between the specified minimum and maximum values, based on number of concurrent users/queries.

Scaling Policy  ▼

The policy used to automatically start up and shut down clusters.

Auto Suspend  ▼

The maximum idle time before the warehouse will be automatically suspended.

Auto Resume ?

Comment

Show SQL Cancel Finish

#### Auto-Scale mode

You can manually change your warehouse according to your query structure and complexity. Below are the scaling methods available in snowflake.

#### Vertical scaling :

scaling up: Increasing the size of the warehouse (small to medium)

scaling down: decreasing the size of the warehouse (medium to small)

**Horizontal scaling:**

Scaling in:

Removing unwanted clusters from warehouse limit. (4 → 2)

scaling out:

Adding more clusters to the existing list of warehouses. (2 → 4)

**12. Where is metadata stored in Snowflake?**

*Once the table is created in Snowflake, it generates metadata about the table containing a count of the rows, the date-time stamp on which it gets created, and aggregate functions such as **sum**, **min**, and a **max** of numerical columns.*

*Metadata is stored in S3 where snowflake manages the data storage.*

*that's why while querying the metadata, there is no need of running a warehouse.*

**13. Briefly explain the different data security features that are available in Snowflake**

Multiple data security options are available in snowflake such as :

1. Secure view
2. Reader account
3. Shared data
4. RBAC

**14. What are the responsibilities of a storage layer in Snowflake?**

The storage layer is nothing but the cloud storage service where data resides.

It has responsibilities such as :

1. Data protection
2. Data durability
3. Data Encryption
4. Archival of Data

## **15. Is Snowflake an MPP database**

Yes. By MPP it means Massively Parallel processing. Snowflake is built on the cloud so it inherits the characteristics of the cloud such as scalability. It can handle parallel running queries by adding necessary compute resources.

Snowflake supports shared-nothing architecture where the compute env is shared between the users.

When the query load increases, it automatically creates multiple clusters on nodes capable of handling the complex query logic and execution.

## **16. Explain the different table Types available in Snowflake:**

It supports three types of tables :

### **1. Permanent :**

Permanent tables are the default type of tables getting created in snowflake. It occupies the storage in cloud storage. The data stored in a permanent table gets partitioned into [micro-partitions](#) for better data retrieval. This type of table has better security features such as Time travel. The default time travel period for the permanent table is 90 days.

**2. Temporary:** Unlike permanent tables, temporary tables do not occupy the storage. All the data stays temporarily in the memory. It holds the data only for that particular session.

**3. Transients:** Transient tables are similar to temporary with respect to the time travel period but the only difference is transient tables need to be dropped manually. They will not get dropped until explicitly dropped.

## **17. Explain the differences and similarities between Transient and Temporary tables**

Table type	Characteristics	Time travel period	Fail-safe	Persistence
Permanent	Stored physically in storage layer	0-90 days	7 days	until dropped
Temporary	hold data per session	1 day	0 days	Until session closed
Transient	table present until explicitly dropped	1 day	0 days	until dropped

**18. Which Snowflake edition should you use if you want to enable time travel for up to 90 days :**

The Standard edition supports the time travel period of up to 1 day. For time travel of more than 1 day for the permanent table, we need to get a Snowflake edition higher than standard. All snowflake editions support only one day of time travel by default.

**19. What are Micro-partitions :**

Snowflake has its unique way of storing the data in cloud storage. Snowflake is a columnar data warehouse as it stores data in columnar format. By columnar, it means instead of storing data row-wise it splits the table into columnar chunks called Micro-partitions. Why micro because it only limits each partition to be 50 to 500 MB.

Snowflake doesn't support indexing instead it manages the metadata of each micro-partition to retrieve data faster. A relational database when queried uses indexes to traverse all the rows to find requested data. The overhead of reading all the unused data causes the data retrieval time consuming and compute-heavy. Contrary to relational DB, snowflake uses the metadata of MP and checks which chunk or MP contains the data requested by the user. Metadata content the offset and the number of rows consist in that particular micro partition. Using the metadata, snowflake manages all micro-partitions for data storage and retrieval.

Check the snowflake doc on [micro-partitions](#).

**20. By default, clustering keys are created for every table, how can you disable this option**

When new data continuously arrived and loaded into micro-partitions some columns (for example, event\_date) have constant values in all partitions (naturally clustered), while other columns (for example, city) may have the same values appearing over and over in all partitions.

Snowflake allows you to define **clustering keys**, one or more columns that are used to co-locate the data in the table in the same micro-partitions.

To suspend Automatic Clustering for a table, use the [ALTER TABLE](#) command with a SUSPEND

RECLUSTER clause. For example:

```
alter table t1 suspend recluster;
```

To resume Automatic Clustering for a clustered table, use the [ALTER TABLE](#) command with a RESUME

RECLUSTER clause. For example:

```
alter table t1 resume recluster;
```

## 21. What is the default type of table created in the Snowflake.

In addition to permanent tables, which is the default table type when creating tables, Snowflake supports defining tables as either temporary or transient. These types of tables are especially useful for storing data that does not need to be maintained for extended periods of time (i.e. transitory data).

## 22. How many servers are present in X-Large Warehouse

**Create Warehouse**

Name \*

Size

X-Small (1 credit / hour)

Small (2 credits / hour)

Medium (4 credits / hour)

Large (8 credits / hour)

X-Large (16 credits / hour)

2X-Large (32 credits / hour)

3X-Large (64 credits / hour)

4X-Large (128 credits / hour)

5X-Large (256 credits / hour)

Maximum Clusters

Minimum Clusters

Scaling Policy

Auto Suspend

The maximum idle time before the warehouse will be automatically suspended.

Auto Resume

Comment

Show SQL

**23. As Snowflake should use one of the cloud providers (like AWS or Azure) as part of its architecture, why can't the AWS database Amazon Redshift can be used instead of the Snowflake warehouse.**

**24. What view types can be created in Snowflake but not in traditional databases:**

Likewise tables in snowflake there are different types of views that can be created in snowflake i.e normal Views, Secure Views, and Materialized Views.

Normal views are similar to the views found in RDBMS where the output data depends on the query it will run on a table or multiple tables. The query needs to be refreshed in order to reflect the updated data.

Secure Views prevent users from possibly being exposed to data from rows of tables that are filtered by the view. With secure Views, the view definition and details are only visible to authorized users (i.e. users who are granted the role that owns the View).

A materialized view is a pre-computed dataset derived from a query specification which is nothing but a SELECT query in its definition. The output is stored for later use.

Since the underlying data of the given query is pre-computed, querying a materialized view is faster than executing the original query. This performance difference can be significant when a query is run frequently or it is too complex.

**25. Is Snowflake a Data Lake**

A data lake is normally used for dumping all kinds of data coming from various data sources where it can contain text data, chats, files, images, or videos. The data will be unfiltered, unorganized, and difficult to analyze.

We cannot use this data to carry any information out of it.

On a similar basis, the snowflake is supporting structured and semi-structured data with scalable cloud storage providing data lake features along with analytical usage of the data.

By choosing snowflake you get the best of both data lake and data warehouse.

**26. What are the key benefits you have noticed after migrating to Snowflake from a traditional on-premise database.**

1. Cloud agnostic.
2. Decoupled storage and compute.
3. Highly scalable.
4. Query performance.
5. supports structured and semi-structured data.
6. Native connectors such as python , scala , R, and JDBC/ODBC.
7. Secure data sharing.
8. Materialized views.

**27. When you execute a query, how does Snowflake retrieves the data as compared to the traditional databases.**

1. When end user execute any query, it first goes to cloud service layer where it get optimized and restructured for better performance. The query will be tuned in terms of getting data from the underlying data storage. Also the query gets compiled by query compiler in same layer.
2. after compilation, it goes to metadata cache to check if the cache has stored any data related to that query

**28. Explain the difference between External Stages and Internal Name Stages:**

Stages denotes where you want to stage (hold) the data in snowflake.

There are two types of stages exists in snowflake :

1. Internal stage :

In this stage , snowflake provide place to hold the data within itself. Data never leave snowflake VPC in this kind of stage.

its also gets divided into sub categories as :

1. User : Each user get automatically allocated stage for data loading
2. Table : Each table get automatically allocated stage for data loading
3. Named : Named stages can be created manually for data loading.

2. External stage:

In opposite to Internal stage, external stages points to locations outsides on Snowflake. i.e. Cloud storage buckets ( S3, GCS, Azure blob )

*You must specify an internal stage in the [PUT](#) command when uploading files to Snowflake.*

*You must specify the same stage in the [COPY INTO <table>](#) command when loading data into a table from the staged files.*

**29. Explain the difference between User and Table Stages.****User stages:**

Each user has a Snowflake stage allocated to them by default for storing files. This stage is a convenient option if your files will only be accessed by a single user, but need to be copied into multiple tables.

User stages have the following characteristics and limitations:

- User stages are referenced using @~; e.g. use LIST @~ to list the files in a user stage.
- Unlike named stages, user stages cannot be altered or dropped.
- User stages do not support setting file format options. Instead, you must specify file format and copy options as part of the [COPY INTO <table>](#) command.

This option is not appropriate if:

- Multiple users require access to the files.
- The current user does not have INSERT privileges on the tables the data will be loaded into.

#### **Table stage:**

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

Table stages have the following characteristics and limitations:

- Table stages have the same name as the table; e.g. a table named mytable has a stage referenced as @%mytable.
- Unlike named stages, table stages cannot be altered or dropped.
- Table stages do not support transforming data while loading it (i.e. using a query as the source for the COPY command).

Note that a table stage is not a separate database object; rather, it is an implicit stage tied to the table itself. A table stage has no grantable privileges of its own. To stage files to a table stage, list the files, query them on the stage, or drop them, you must be the table owner (have the role with the OWNERSHIP privilege on the table).

### **30. What are the constraints which are enforced in Snowflake?**

Normally there no constraints are enforced in snowflake except for NOT NULL constraints, which are always enforced.

Usually, in traditional databases, there are many constraints being used to validate or restrict the incorrect data from being stored such as primary key, not null, Unique, etc.

Snowflake provides the following constraint functionality:

- Unique, primary, and foreign keys, and NOT NULL columns.
- Named constraints.
- Single-column and multi-column constraints.
- Creation of constraints inline and out-of-line.
- Support for creation, modification and deletion of constraints.