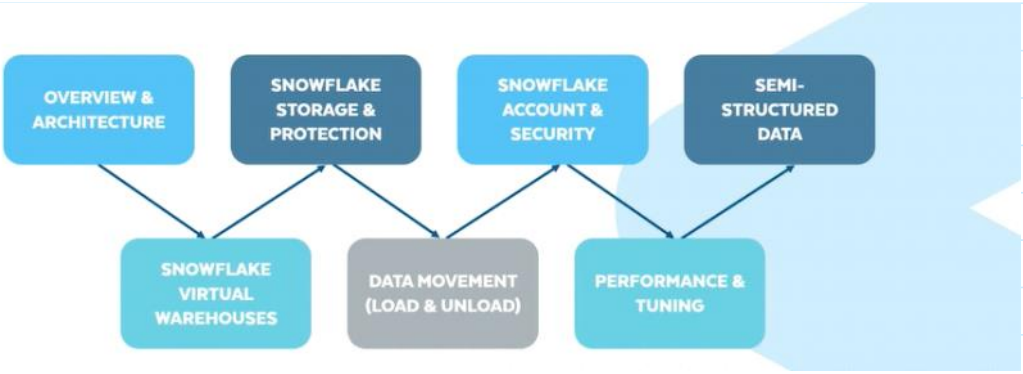
SnowPro Core Certification

13 October 2021 11:30

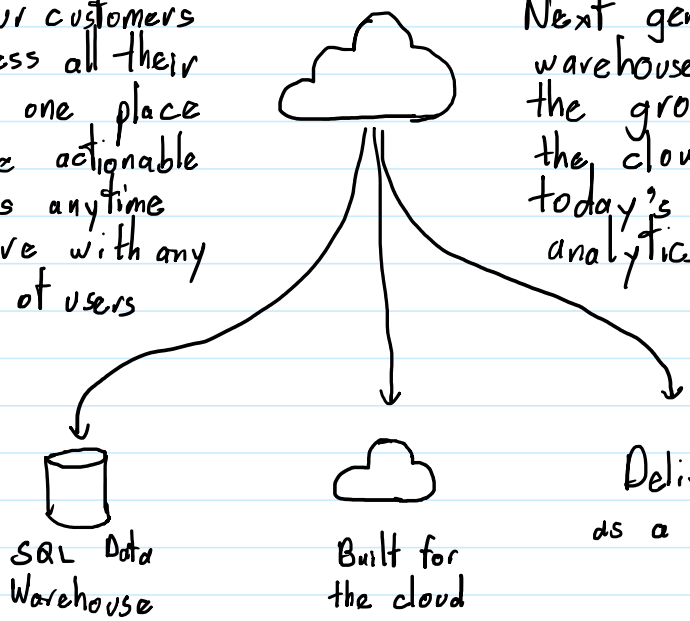
Overview & Architecture

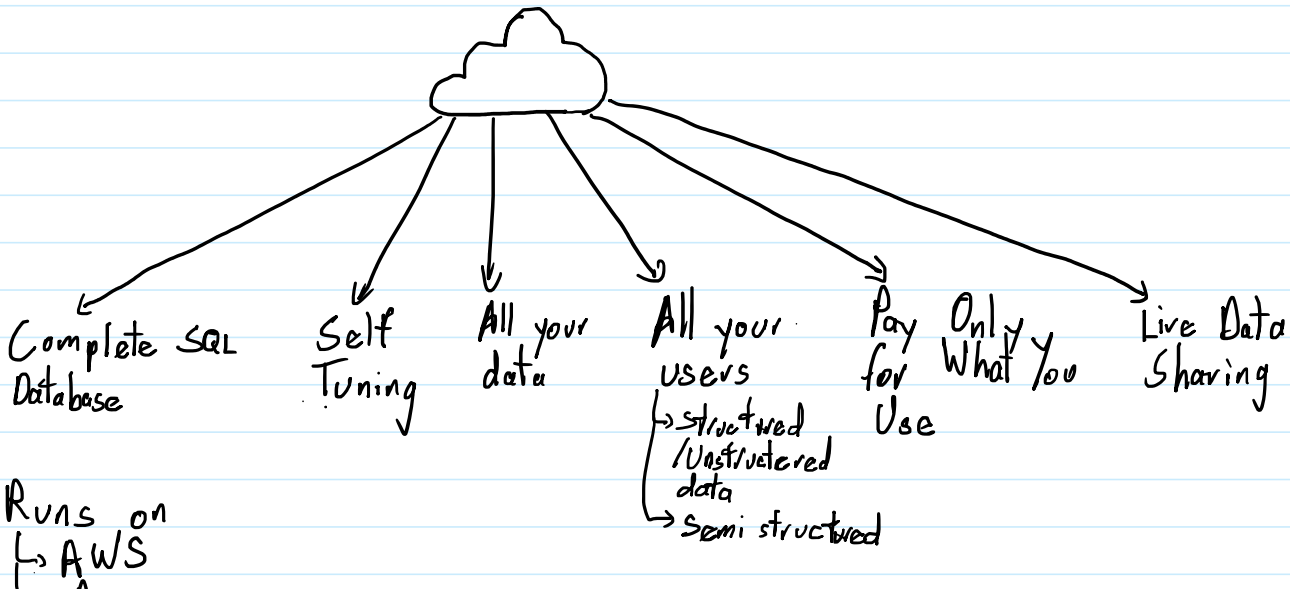


Cloud Data Platform

Vision
Allow our customers to access all their data in one place to make actionable decisions anytime anywhere with any number of users

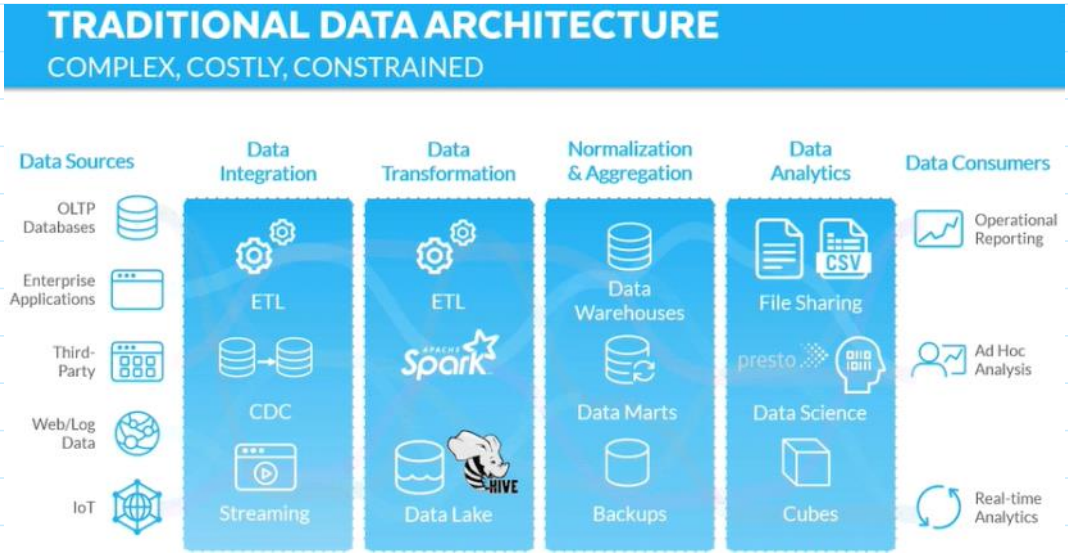
Our solution
Next generational data warehouse built from the ground-up for the cloud to address today's data & analytics challenges



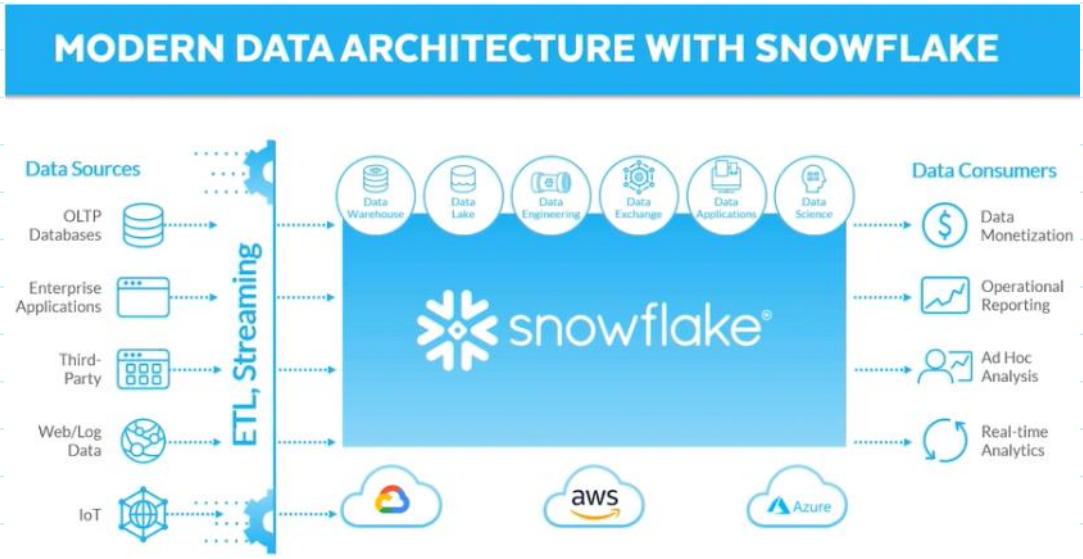


Runs on
↳ AWS
↳ Azure

↳ GCP

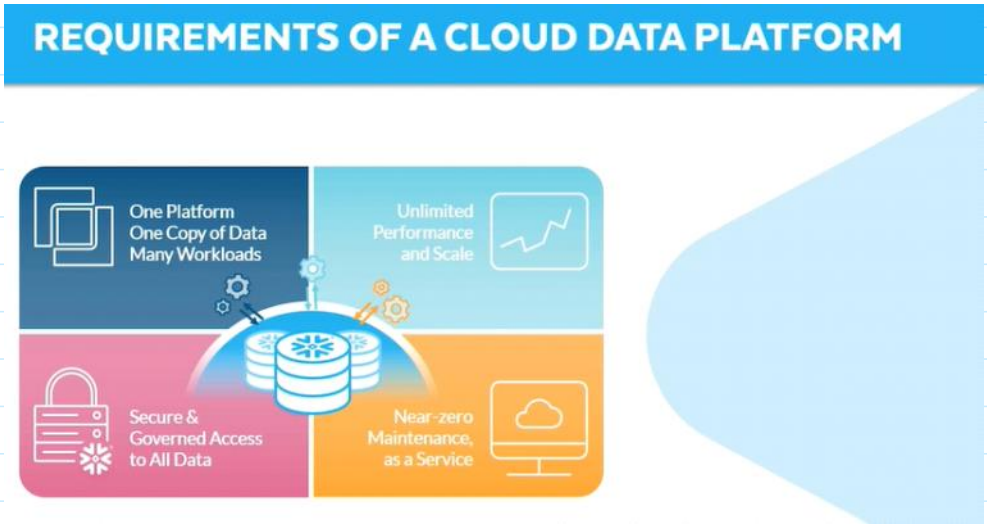


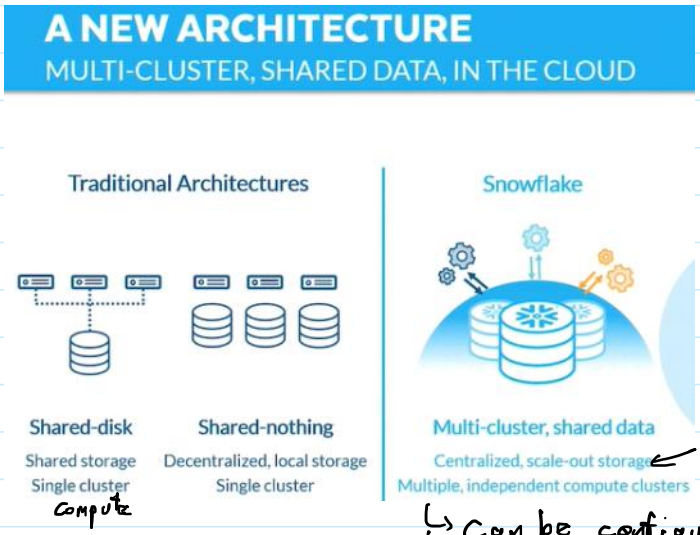
Cons:
↳ Complex
↳ Costly
↳ Constrained
↳ Difficult to maintain



Snowflake is designed as an analytical solution
↳ Cloud data solution
↳ Supports data warehousing
↳ Analytical queries
↳ OLAP system (Online Analytical Processing)

Not a siloed system





compute

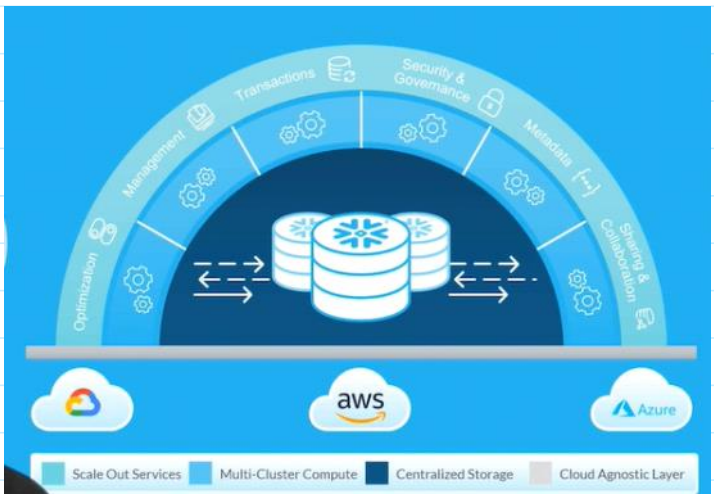
almost infinite

↳ Can be configured as per customer
↳ Changeable on the fly
↳ Infinite elasticity & scale

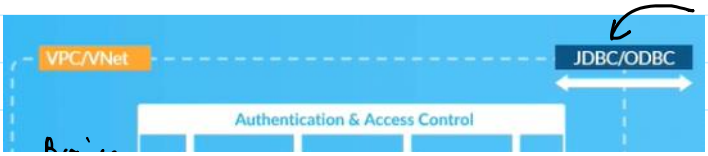
3 layers
↳ Services
↳ Multicluster Compute
↳ Centralized storage

⇓

Cloud Agnostic Layer

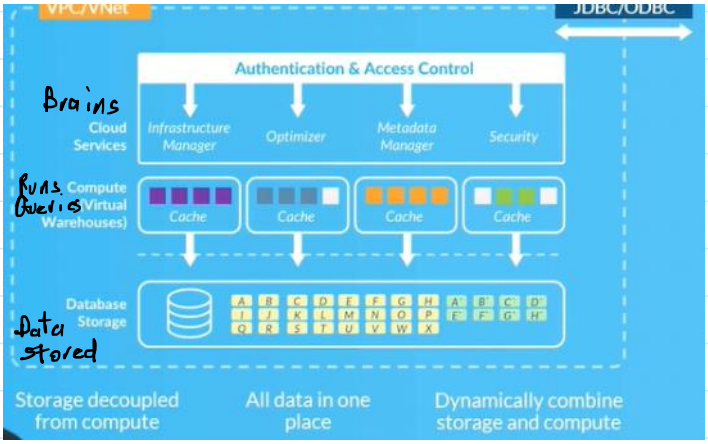


Architecture:-



Connectors (others like python, .net also supported)
(Read

https://www.microfocus.com/documentation/xdbc/win20/BKXDXDINTRXD1.5.html

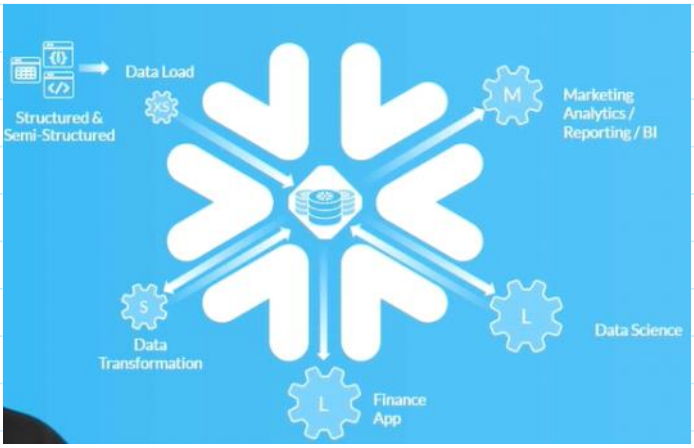


https://www.microfocus.com/documentation/xdbc/win20/BKXDXDINTRXD1.5.html

Independent Layers

When queries are running
→ read data from storage
→ run the query
→ return result

Functional Architecture



→ Compute layer is configurable by the customer (can be sized differently)

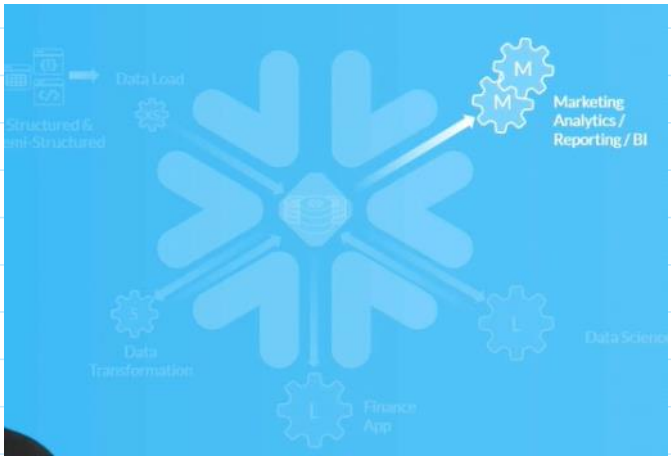
→ The larger the virtual warehouse (compute layer), the more compute power it has to run queries

Eg.
Data Science
↳ Very complex, compute intensive
↳ Large virtual warehouse may not be sufficient (if queries take long time)
↳ Resizing from L → XL can be done on the fly

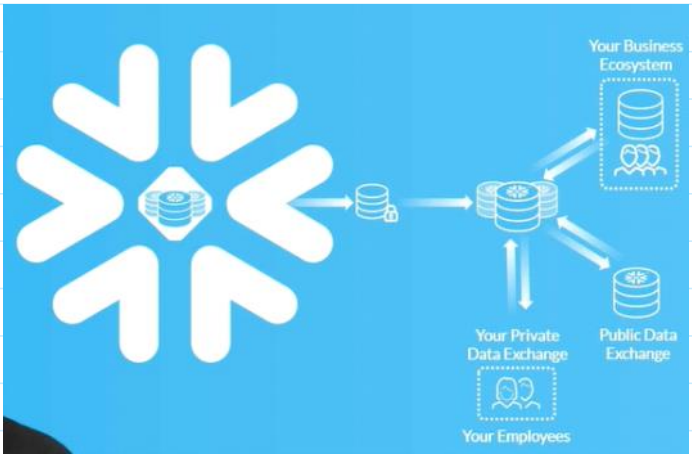
on the fly
- Resizing doesn't cause the query to stop / errors / crashes / downtimes
- Queries running on Large keep running on large
- New queries run on XL
- Once done, we can scale down to L / suspend

Marketing

Lots of queries
- Concurrency problem
  - Solve by multicluster warehouse
  - Multicluster compute
  - As workload comes on & queries start to queue, Snowflake can add additional clusters of compute to that one virtual warehouse (VW)



Data Sharing



Clone

Create a copy of an object/database/schema and not duplicate the storage

Can be used to quickly spin up test & development environments to run your testing & development cycles

1 Tb clone → 1 Tb
(no extra size!!)

Editions

Snowflake Structure

13 October 2021 23:25

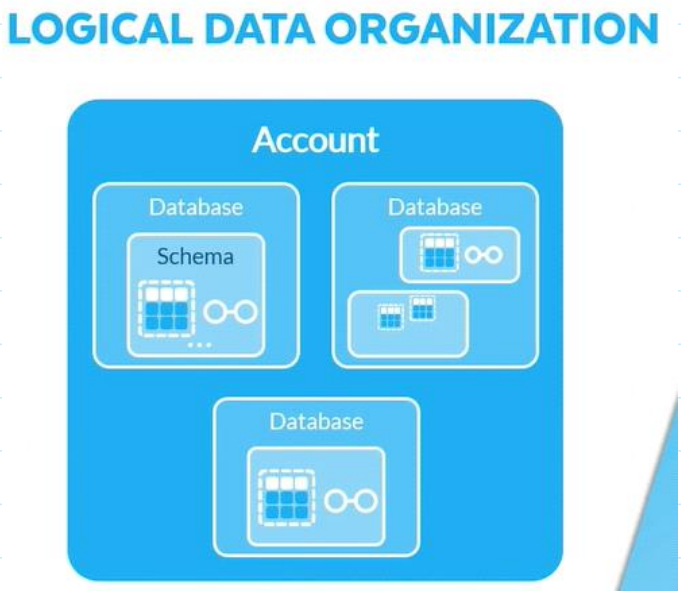
Snowflake Object Model

Logical Data Organization

→ Databases & schemas logically organise data within a Snowflake account

→ A database is a logical grouping of schemas
Each database belongs to a single account

→ A schema is a logical grouping of database objects such as tables & views



→ All Snowflake objects reside within logical containers with the top level container being the Snowflake Account

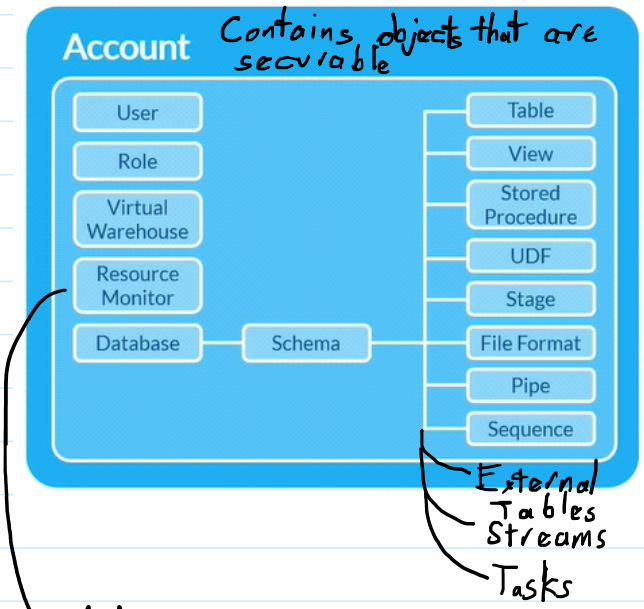
→ All objects are individually securable
- discretionary access control
- role based

→ Users perform operations on objects "privileges" that are granted to roles

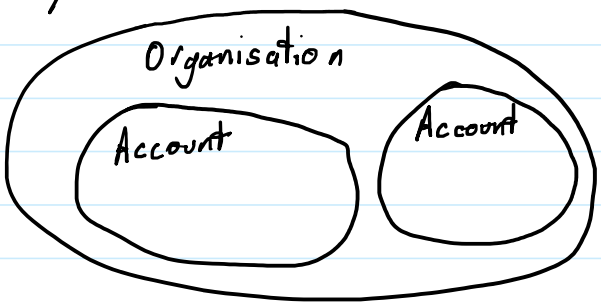
Sample privileges

→ Create a virtual warehouse
→ List tables in a schema
→ Insert data into a table
→ Select data from a table

Snowflake Objects



Sidenote :-
Snowflake plans to move to a system :-



Admin tool to monitor credit usage on virtual warehouses

Table Types

Permanent (Default Table type)
- Persist until dropped
- Designed for data that requires the highest level of data protection & recovery
- Default table type
- Time travel upto 90 days with Enterprise
- Fail Safe ✓

Temporary
- P . I R . I . I I . .

Temporary
- Persist & tied to a session (think single user)
- Used for transitory data (eg ETL/ELT)
- Time travel 0 or 1 days
- Fail Safe x

Transient (Applicable to Database, Schema & Table)
- Persist until dropped
- Multiple user
- Used for data that needs to persist but does not need the same level of data retention as a permanent table
- Time travel 0 or 1 days
- Fail Safe x

External
- Persist until removed
- Snowflake over an external data lake
- Data accessed via an external stage
- Read only
- Time Travel x
- Fail Safe x

Fail Safe → Safety net administration backup feature
- Allows admins to recover & store data for an additional 7 days beyond the time travel
- Only accessible through contacting the Snowflake

Travel
- Only accessible through contacting the Snowflake Tech Support
- Only Permanent Table supports this feature
- Costs rise since additional 7 days of storage

Time Travel
- Data Protection Feature
  - Retrieve Historical Data
- Query Historical Data

Views → Predefined SQL Query – a select statement against a table

Standard Views
- Default view types
- Named definition of a query – SELECT statement
- The role that creates it, owns it & it will execute as that owning role
- Underlying DDL is available to any role with access to the view

Secure View
- Definition & details only visible to authorised users
- Executes as owning role
- Snowflake query optimizer bypasses optimizations used for regular views } Cannot see underlying DDL

Materialized View

→ Behaves more like a table
→ Results of underlying query are stored
→ Auto-refreshed

→ Secure Materialized View is also supported
↳ More expensive

Two tables
Basic table
Generated Table

Standard View

```
CREATE VIEW reg_view COMMENT = 'test Regular View' AS
SELECT a, e
FROM generated_table;

SELECT *
FROM reg_view;

SHOW VIEWS;
```

→ when it was created, what its database, schema, owner, secure view? materialized view?

```
DESC VIEW reg_view;
SELECT get_ddl('view', 'reg_view')
```

← result returns the DDL (code) that was used to create the view

Secure View

```
CREATE SECURE VIEW sec_view COMMENT='Test Secure View' AS
SELECT a,d,f
FROM generated_table;

SHOW VIEWS;

DESC VIEW sec_view
SELECT get_ddl('view', 'sec_view')
```

← Can see only if using the role that owns it

Materialized View

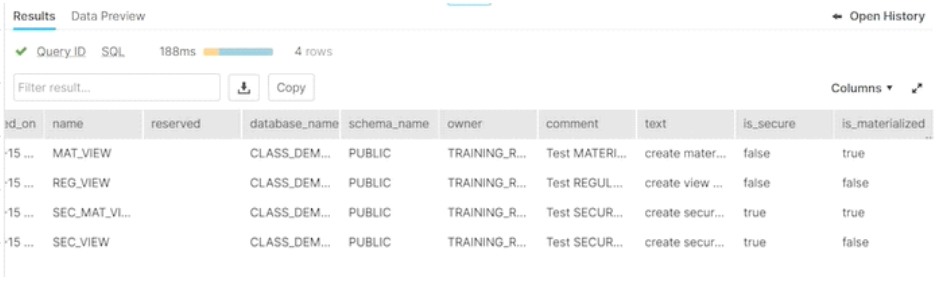
```
CREATE MATERIALIZED VIEW mat_view COMMENT = 'Test Materialized View' AS
SELECT B,C
FROM generated_table

SHOW VIEWS;

DESC VIEW mat_view
SELECT *
FROM mat_view
LIMIT 100
```

Secure Materialized View

```
CREATE SECURE MATERIALIZED VIEW sec_mat_view COMMENT = 'Test Secure Materialized View' AS
SELECT F,B,A
FROM generated_table

SHOW VIEWS;

DESC VIEW sec_mat_view

SELECT *
```

FROM sec_mat_view
LIMIT 100

Show Views



Results | Data Preview | Open History

Query ID | SQL | 188ms | 4 rows

Filter result... | Copy | Columns

ed_on	name	reserved	database_name	schema_name	owner	comment	text	is_secure	is_materialized
15 ...	MAT_VIEW		CLASS_DEM...	PUBLIC	TRAINING_R...	Test MATERI...	create mater...	false	true
15 ...	REG_VIEW		CLASS_DEM...	PUBLIC	TRAINING_R...	Test REGUL...	create view ...	false	false
15 ...	SEC_MAT_VI...		CLASS_DEM...	PUBLIC	TRAINING_R...	Test SECUR...	create secur...	true	true
15 ...	SEC_VIEW		CLASS_DEM...	PUBLIC	TRAINING_R...	Test SECUR...	create secur...	true	false

To drop views

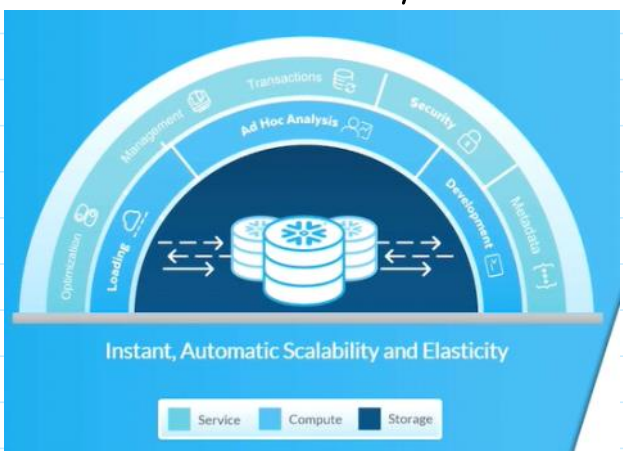
DROP MATERIALIZED VIEW mat_view
DROP VIEW reg_view
DROP VIEW sec_view
DROP MATERIALIZED VIEW sec_mat_view

Cloud Services Layer

14 October 2021 15:15

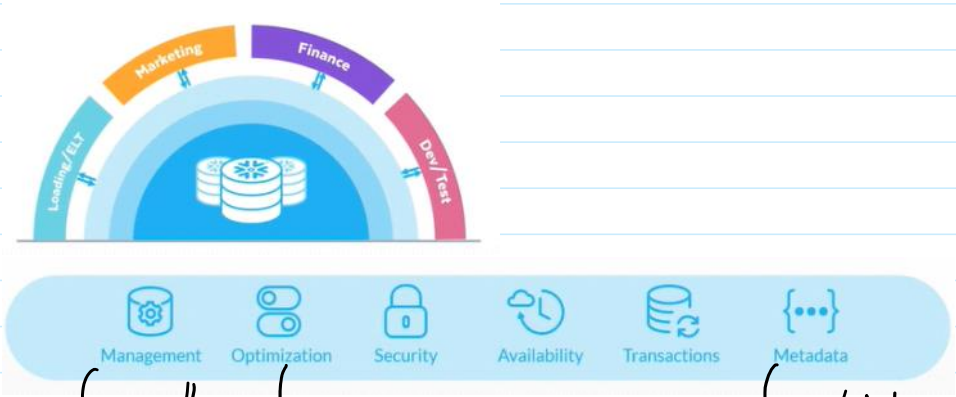
3 Architectural Layers :

→ Storage
→ Compute (Virtual Warehouse) Layer
→ Cloud Services Layer



Services Layer

→ Brains of the service
→ Coordinates activities across Snowflake
→ Runs on compute instances provisioned by Snowflake



Goverall operations → SQL → & statistics

- Management

→ Centralized management for all storage → tables, storage layer micropartioning, table versioning, micropartition versioning
→ Manages compute that works with the storage
→ Transparent, online updates & patches

- Optimizer Service

→ SQL Optimizer (Cost based optimization (CBO))
→ Automatic JOIN order optimization
  • No user input or training required
→ Automatic statistics gathering
→ Pruning using metadata about micro-partitions

- Security

→ Authentication
→ Access controls for users & roles
→ Access control for shares
→ Encryption & key management

- Metadata Management

→ Stores metadata as data is loaded into the system
→ Handles queries that can be processed completely from metadata
→ Used for time travel & cloning
→ Every aspect of Snowflake architecture leverages metadata

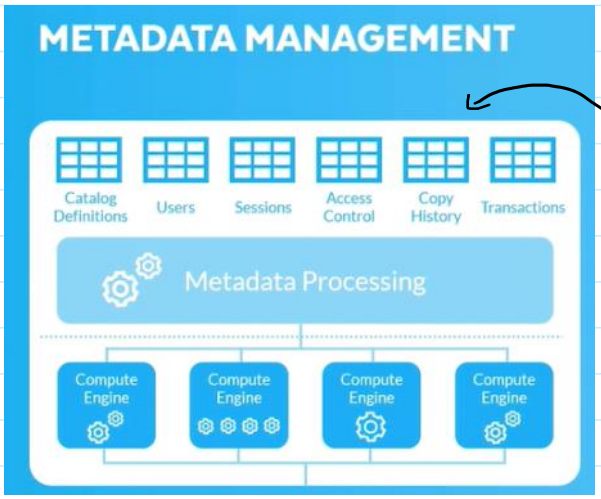


Table info such as :- min/max, null, no of records column

```
USE role training_role;

-- SHOWs are Cloud Services only. As you run all these statements, notice there's no GREEN DOT!
SHOW USERS;
SHOW DATABASES;

-- Session statements and context do just that.
USE DATABASE snowflake_sample_data;
USE SCHEMA TPCH_SF1;

-- Queries require compute (Virtual Warehouse)
SELECT *
FROM nation

-- Context functions also can be run on services only
SELECT current_account(), current_user()
```

```
-- Some queries that only use stats
SELECT COUNT(*), MIN(L_SHIPDATE), MAX(L_SHIPDATE)
FROM lineitem;

-- Can even create things
CREATE DATABASE instructor_services_demo;
CREATE SCHEMA instructor_services_demo.schema_demo;
USE instructor_services_demo.schema_demo;

-- Even created a table
CREATE TABLE demo_table (C1 INT, C2 VARCHAR);
SELECT COUNT(*)
FROM demo_table

-- And a view!
CREATE VIEW demo_view AS
SELECT *
FROM demo_table

DROP DATABASE instructor_services_demo;
```

Data Types

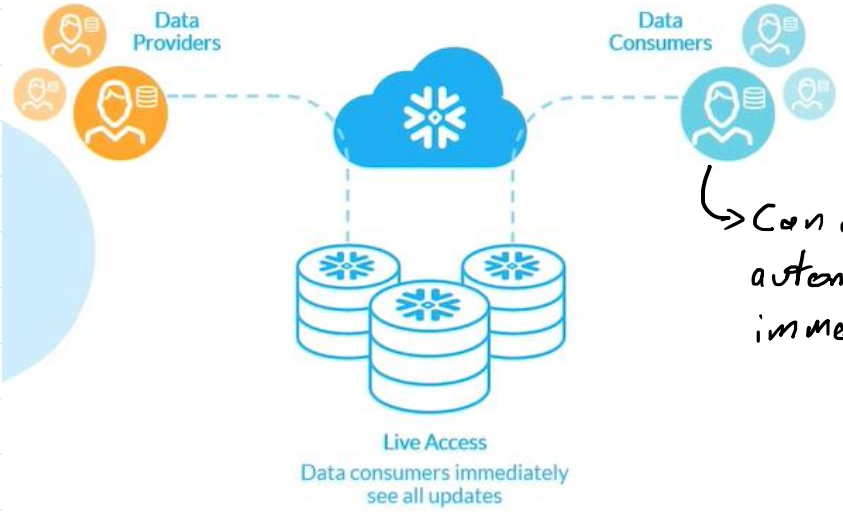
Strings: String & Binary Data Types — Snowflake Documentation
Numeric: Numeric Data Types — Snowflake Documentation

Data Sharing Overview

16 October 2021 09:27

→ Powerful & unique set of live read-only data sharing features, unparalleled in the industry

→



↳ Can see updates automatically & immediately

→ Data Providers as well as consumers are both Snowflake customers — both have their own Snowflake accounts

→ Data Providers —Data from provider account→ Data Consumer

↳ Can see data as a database
↳ schemas, tables, views

Direct Secure Data Sharing

→ No data movement, nothing is copied

↳ No need to export data to files & move data to files & ftp files around

↳ Query data live from Snowflake or

↳ Query data live from Snowflake or any external tool that supports connections from Snowflake (BI reporting & analytics tools)

→ Data Consumers immediately see all updates

→ Share with an unlimited number of consumers

Data Sharing Accounts

Data Providers

Share data with others

Accesses shared data with their own Snowflake account ↳ query, analyse, consume

Data Consumers

Query using compute from data provider's account

Reader Accounts

↳ Provider wants to share data with someone who is not a Snowflake customer

↳ Sets up a separate reader account & provider can administer the account that exists within their own Snowflake environment & they provide someone with access to it

Provider is not charged money for sharing data by Snowflake. Providers however can monetize sharing data

Consumers use a virtual warehouse on their Snowflake account & they would pay for those queries

Provider —(Data + Virtual Warehouse)→ Reader Account

↑ He will be charged for the queries the readers are executing

Control over the resources Reader Accounts are using

Can also monetize & charge Reader Accounts

What is a Share?

→ A securable object in Snowflake
→ Encapsulates all information required to share objects

→ Comprises :-
  → Privileges that grant access to the database & schema containing objects to share
  → Privileges that grant access to specific objects
  → The account(s) with which the objects are shared

Shares - Recap

→ Shares are read-only
→ Tables, secure views & secure UDFs can be shared
→ Access to a share can be revoked at any time
→ Consumers can create new tables from a share

Data Shares using two ways — Web interface, SQL commands

Consumers CANNOT :-

1) Reshare a share
2) Clone a share

Demo

How do I know if data is shared?

look for the inbound arrow for shared data or hover over it

Alternatively :-

Navigate to the Databases Tab

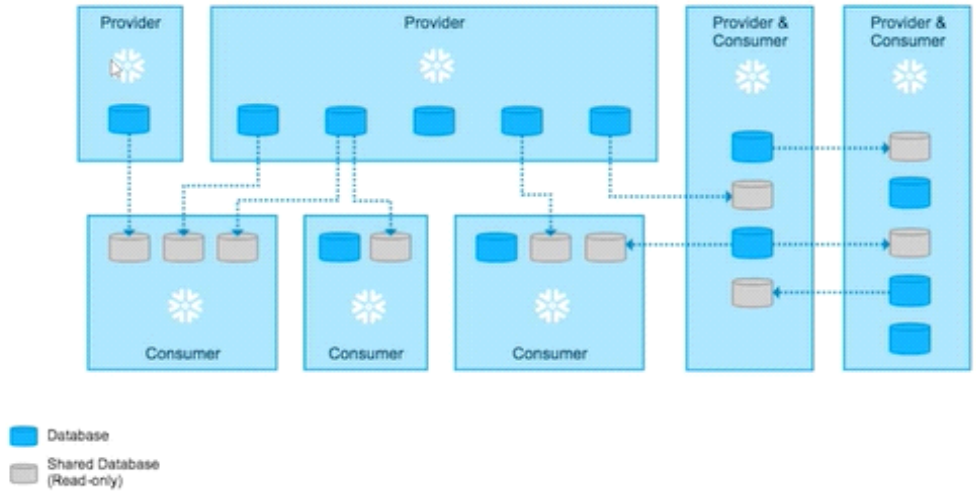
If 'Origin' column is populated against a database/schema, it is shared

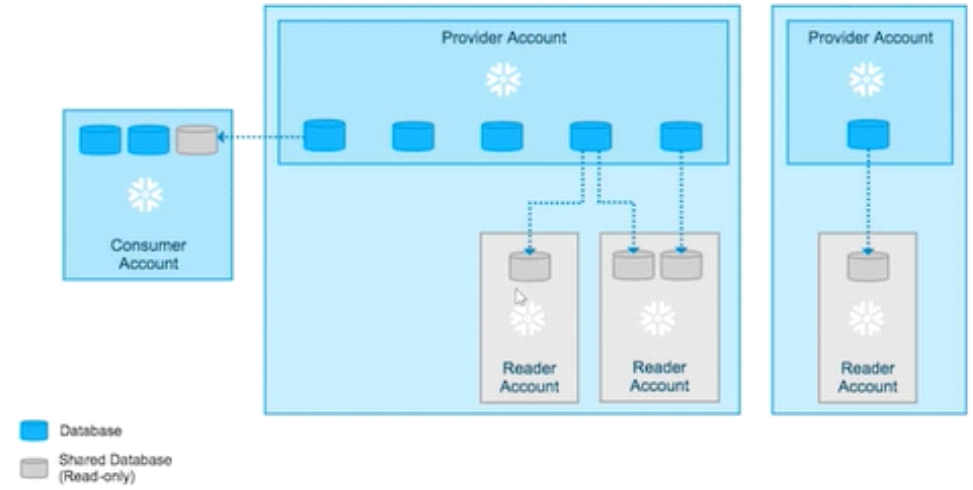
Share some data

Sign into the administrator account for creating a data share

↳ Either be admin or sign into a role with data sharing privileges

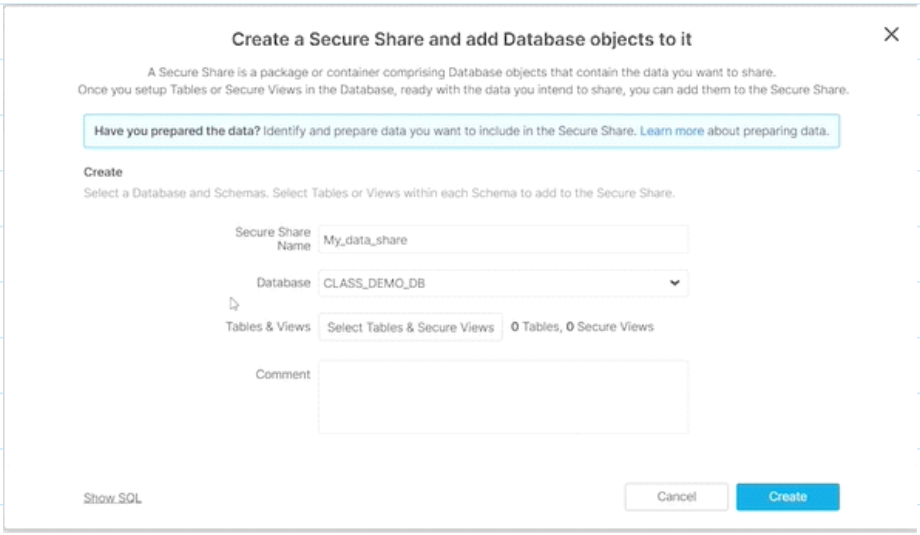
Note:-

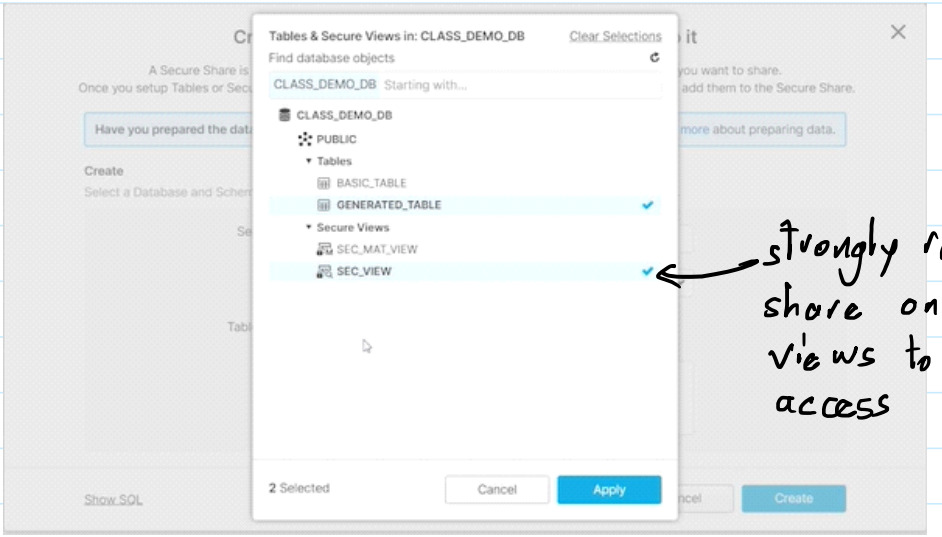




Sharing using Web UI Wizard

- Select Shares Tab
- Select Outbound Tab
- Select Create





strongly recommended to share only the secure views to prevent unwanted access

Can see SQL code as well

Click Create. You will be able to see a preview of data being shared before adding Consumers

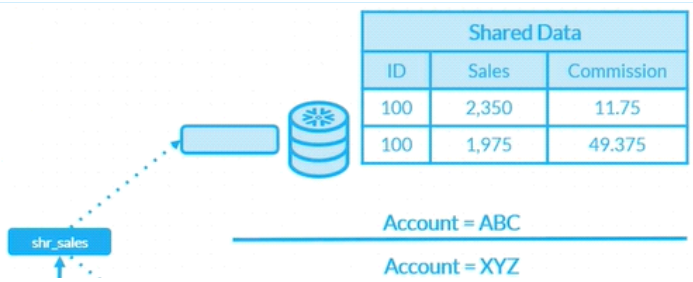
Sharing secure views with access controls

Data Provider

Data Consumers

Partner Data		
ID	Sales	Commission
100	2,350	11.75
100	1,975	49.375
200	3,459	8.6475
200	9,156	68.67

acct_map	
ID	acct_name
100	ABC
200	XYZ

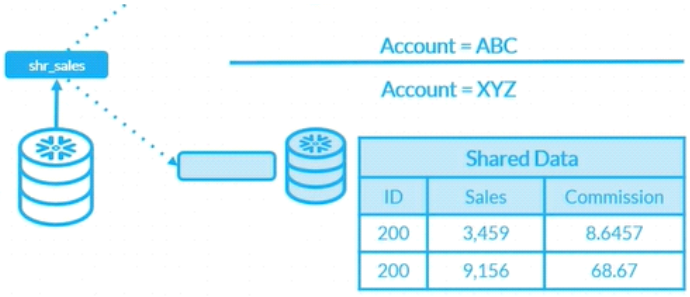



```
CREATE SECURE VIEW shared_data
```

200	9,156	68.67

```
CREATE SECURE VIEW shared_data
AS SELECT * FROM partner_data pd
  JOIN acct_map am ON pd.id = am.id
  AND am.acct_name = CURRENT_ACCOUNT();
```

```
CREATE SECURE VIEW shared_data
AS SELECT * FROM partner_data pd
  JOIN acct_map am ON pd.id = am.id
  AND am.acct_name = CURRENT_ACCOUNT();

GRANT SELECT ON shared_data TO SHARE shr_sales;

ALTER SHARE shr_sales ADD ACCOUNTS=ABC,XYZ;
```



Shared Data		
ID	Sales	Commission
200	3,459	8.6457
200	9,156	68.67

User Defined Functions in Snowflake

16 October 2021 10:48

→ Perform custom operations that are not available through the built in functions

→ SQL and Javascript supported

→ No DDL / DML supported

→ Can be unsecure or secure

→ Return a singular scalar value or if defined as a table function, a set of rows

SQL UDF Example

1. Create the table and data to use
2. Create the UDF

```
CREATE FUNCTION profit() RETURNS NUMERIC(11,2) AS
    $$
    SELECT SUM((retail_price - wholesale_price) * number_sold)
    FROM purchases
    $$;
```

3. CALL the UDF

```
SELECT profit();
```

number_sold	wholesale_price	retail_price
3	10.00	20.00
5	100.00	200.00

Profit ()
$530.00

```
create or replace function pi_udf()
    returns float
    as '3.141592654::FLOAT'
    ;

select pi_udf();

create or replace function simple_table_function()
    returns table(x integer, y integer)
    as
    $$
        select 1, 2
        union all
        select 3, 4
    $$;
select * from table(simple_table_function);
```

Javascript UDF

```
create function variant_nulls(v_variant)
        returns variant
        language javascript
        as '
        if (V === undefined){
        return "input SQL null";
        } else if (v === null){
          return "input variant null";
        } else if (v === 1){
        } else if (v ===2){
        } else if (v ===3){
        return {
                key1: undefined
                key2: null
                };
        }  else {
           return v;
        }
        ';
```

```
CREATE OR REPLACE FUNCTION variant_nulls(V VARIANT)
        RETURNS VARCHAR
        LANGUAGE JAVASCRIPT
        AS '
        if (V === undefined){
                return "input was SQL null";
        }
        else if (V === null){
                return "input was variant null";
        }
        else {
                return V;
        }
        ';

SELECT null,
        variant_nulls(cast(null as variant))
        variant_nulls(PARSE_JSON('null'))
        ;
```

```
CREATE OR REPLACE FUNCTION array_sort(a array)
        RETURNS array
        LANGUAGE JAVASCRIPT
        AS
        $$
                return A.sort();
        $$;

SELECT array_sort(PARSE_JSON('[2,4,5,3,1]'));
```

```
SHOW FUNCTIONS;
SHOW USER FUNCTIONS;

DESC FUNCTION pi_udf();
DESC FUNCTION simple_table_function()
```

Stored Procedures

16 October 2021 14:16

Only Javascript supported as of now

What can stored procedures do?

→ Allows procedural logic & error handling that straight SQL does not support

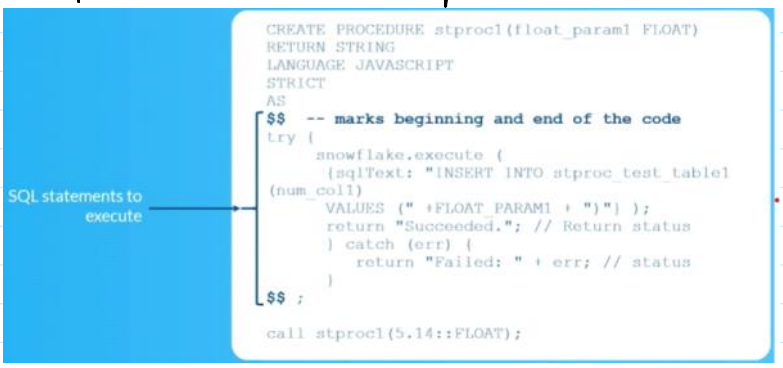
→ Implemented via Javascript

→ Javascript provides the control structures

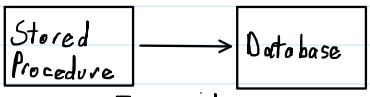
→ SQL is executed within Javascript by calling functions in an API

→ Argument names are case insensitive in the SQL portion of the stored procedure & case-sensitive in the Javascript portion

Components of a stored procedure



How it works



→ Snowflake provided Javascript APIs
  ↳ Javascript objects & methods
  ↳ Error handling & procedural logic

→ SQL and database access/operations
  ↳ Embed SQL codes in the Javascript
  ↳ The SQL codes execute database operations
  ↳ Migrate other databases stored procedure (SQL) codes by embedding SQL in Javascript

STORED PROCEDURE SQL
EXECUTED THROUGH API OBJECTS

Object Class	Description
Snowflake Object	Contains the methods in the stored procedure API. Accessible by default to JavaScript code (you do not need to create the object).
Statement Object	Provides the methods for executing a query statement, and accessing metadata (for example, information about columns and rows) about the statement
ResultSet Object	Contains the results returned by a query (zero or more rows, each with one or more columns). You iterate through a result set by repeating next() and taking some action
SfDate Object	Java does not have a native data type that corresponds to Snowflake SQL TIMESTAMP data types. Used when you want to retrieve a TIMESTAMP and stored it as a variable.

Stored Procedures vs UDFs

Stored Procedure	User-Defined Function
MAY return a value	MUST return a value
CANNOT return a set of rows	CAN return a set of rows (table)
Can access database objects and issue SQL statements	DDL and DML operations not permitted
Can run as function owner or caller Owner is default Specified at creation time	Runs as the function owner

Via stored procedures role that created it owns it so they can query database without explicit access

```
create or replace procedure sp.pi()
        returns float not null
        language javascript
        as $$
        return 3.1415926
        $$;

call sp.pi()

show procedures;

desc procedure sp.pi()

select get_ddl('procedure', 'sp.pi()')
```

```
create or replace procedure session_var_user()
  returns float
  language javascript
  EXECUTE AS CALLER
  as
  $$
  // Set the second session variable
  var stmt = snowflake.createStatement(
      {sqlText: "set $SESSION_VAR2 = 'I was set inside the StProc.'"}
      );
  var rs = stmt.execute();  // we ignore the result in this case
  // Run a query using the first session variable
  stmt = snowflake.createStatement(
      {sqlText: "select f from sv_table where f > $SESSION_VAR1"}
      );
  rs = stmt.execute();
  rs.next();
  var output = rs.getColumnValue(1);
  return output;
  $$
```

Snowflake Virtual Warehouses

19 November 2021 11:21

Compute Layer / Virtual Warehouses

→ A named wrapper around a cluster of servers with CPU, memory & disk

→ The larger the warehouse, the more servers in the cluster

→ Extra small has 1 server per cluster
↳ Each size doubles in size

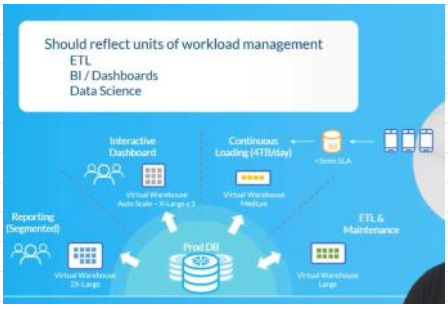
→ Jobs requiring compute run on virtual warehouses

→ While running a virtual warehouse consumes Snowflake credits
↳ You are charged for compute
- XS
- S
- M
- L
- XL

→ When a virtual warehouse is running, even if it is not executing a query, it consumes credits

Virtual Warehouses are the unit of workload segmentation

Workload Segmentation



Virtual Warehouse Types

Standard
→ Will only have a single compute cluster
→ Cannot scale out

Multi-cluster Warehouse
→ Can spawn additional compute clusters (scale out) to manage changes in user & concurrency needs
→ Enterprise Edition feature

Virtual Warehouse Sizing

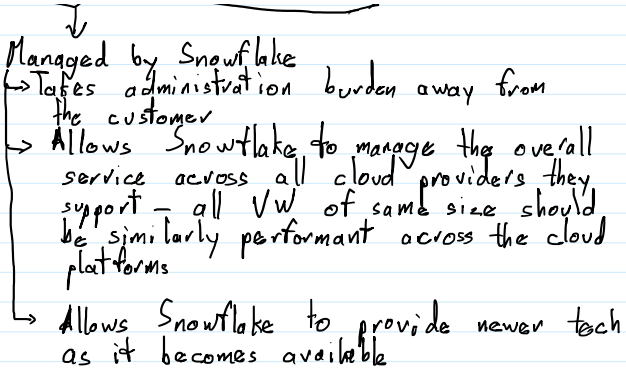
→ Warehouses are sized in 't-shirt' sizing
→ Size determines the number of servers that comprise each cluster in a warehouse
→ Each larger size is double the preceding, in both VMs in the cluster & in the Snowflake credits consumed

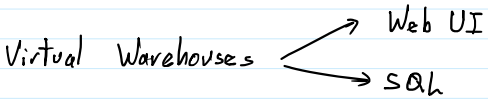
XS → S → M → L → XL → XXL → XXXL → XXXXL → (5XL & 6XL to be added)

Virtual Warehouses & their sizing do not expose the actual VMs or their specs
↳ or EC2 / Azure instances in use

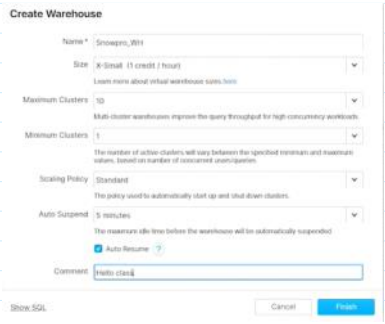
↓

Managed by Snowflake
↳ Takes administration burden away from





- Click on the Warehouses Tab
- Click on Create on top left corner
- Enter the Name without spaces
- Default size is XL (different for SQL command where default is XS)
- Choose the number of maximum clusters (improves the query throughput for high concurrency workloads)
- To set it to standard choose 1 for maximum clusters (upto 10 permitted)
- Scaling Policy (Standard or Economy).
  - Standard: If a query is queueing or snowflake thinks it will queue on a virtual warehouse, it will start up an additional warehouse
  - Economy: More conservative; wait until snowflake anticipates an additional 6 minutes of workload before firing up an additional cluster
- Auto-suspend policy can be set to the number of minutes to wait. Through the WebUI this number can be set to as low as 5 minutes. Through SQL, we can set it to as low as 1 minute. Also has a 'Never' option
- Auto-resume: When a query needs to run, go ahead and start the virtual warehouse and auto-resume.
- Can also add a comment



Select the Show SQL option to see underlying SQL command

```
CREATE WAREHOUSE Snowpro_WH WITH
WAREHOUSE_SIZE = 'XSMALL'
 WAREHOUSE_TYPE = 'STANDARD'
 AUTO_SUSPEND= 300
 AUTO_RESUME = TRUE
 MIN_CLUSTER_COUNT = 1
 MAX_CLUSTER_COUNT = 10
 SCALING_POLICY = 'STANDARD'
 COMMENT = 'Hello class';
```

Click on Finish to create the warehouse
The warehouse will auto-start on creation. We can prevent this by setting a parameter in SQL to False
To suspend a warehouse:

- Click on the virtual warehouse
- Click suspend on the top
- A prompt will appear that also gives an option to view suspend SQL code
  ALTER WAREHOUSE "Snowpro_WH" SUSPEND;

Can also use worksheets for creation of virtual warehouses as well as suspending them

```
 USE role TRAINING_ROLE;

-- a query cannot run without a virtual warehouse available to the role/user
SELECT *, random(10) FROM TRAINING_DB.TRAININGLAB.NATION;

-- Create a warehouse
CREATE OR REPLACE WAREHOUSE INSTRUCTOR_WH
        WAREHOUSE_SIZE = SMALL
        AUTO_RESUME = TRUE
        MIN_CLUSTER_COUNT = 1
        MAX_CLUSTER_COUNT = 10
        AUTO_SUSPEND = 60
        SCALING_POLICY = 'STANDARD'
        INITIALLY_SUSPENDED = TRUE
        COMMENT = 'A warehouse for demoing stuff';

-- Its a Snowflake object like any other
SHOW WAREHOUSES;

-- Resize to medium
ALTER WAREHOUSE INSTRUCTOR_WH
SET
        WAREHOUSE_SIZE = Medium
        AUTO_SUSPEND = 300;

-- Still not running
SHOW WAREHOUSES LIKE 'INSTRUCTOR%';

-- Run a query (starts up, notice green dot)
USE WAREHOUSE INSTRUCTOR_WH;
```

```
-- This will run now since virtual warehouse is available
SELECT *, random(10) FROM TRAINING_DB.TRAININGLAB.NATION;

-- Suspend your virtual warehouse
ALTER WAREHOUSE "INSTRUCTOR_WH" SUSPEND;

-- Auto resume
SELECT *, random(10) FROM TRAINING_DB.TRAININGLAB.NATION;

-- Drop
DROP WAREHOUSE "INSTRUCTOR_WH"
```

Compute Credits

→ Compute Credits are how you are billed for compute usage
  ↳ You may have a set number of credits
  ↳ You may be billed monthly for your credits

→ Compute Credits are charged based on the number of virtual warehouses you use, their size & how long you use them

→ Warehouse usage is charged per second with a one minute minimum

Virtual Warehouse Credits

Warehouse Size	Servers	Credits / Hour	Credits / Second
X-Small	1	1	0.0003
Small	2	2	0.0006
Medium	4	4	0.0011
Large	8	8	0.0022
X-Large	16	16	0.0044
2X-Large	32	32	0.0089
3X-Large	64	64	0.0178
4X-Large	128	128	0.0356

Cloud Services Compute Billing
- Some compute occurs in the cloud services layer
- Customers are charged for cloud computing that exceeds 10% compute costs for the account
- Use WAREHOUSE_METERING_HISTORY view to see how much cloud service compute your account is using

Resizing a Warehouse
- Can be completed at any time even when running
- Completed via ALTER WAREHOUSE statement or the UI
- Effects of resizing:
  - Suspended Warehouse: will start at new size upon next resume
  - Running Warehouse: immediate impact; running queries complete at current size, while queued queries run at new size

Scale up for performance

Elastic Processing Power (CPU, RAM, SSD)

→ Raw performance boost for complex queries or ingesting large data sets

→ More complex queries on larger datasets require larger warehouses

→ Not intended for handling concurrency issues (more users/ queries)

Warehouse sizing guidelines

→ Snowflake uses per-second billing
→ Use larger warehouses for more complex workloads & (auto) suspend when not in use
→ Keep queries of similar size & complexity on the same warehouse to simplify compute resource sizing

Scale out for Concurrency

General Functionality & Considerations

→ Single, Virtual, Warehouse with multiple

compute clusters

→ Delivers consistent SLA, automatically adding & removing clusters based on concurrent usage

→ Scale out during peak times & scale back during slow times

→ Queries load balanced across clusters in a Virtual Warehouse

→ Deployed across availability zones for high availability

Guidelines

MIN_CLUSTER_COUNT: 1-10 (default 1)
MAX_CLUSTER_COUNT: 1-10 (default 1) ≥ MIN_CLUSTER_COUNT

SCALING UP VS. OUT EXAMPLE

Credits Per Hour

Scale for Concurrency

Scale for Performance	1	2	3	4	5	6	7	8	9	10
4XL	128	256	384	512	640	768	896	1024	1152	1280
3XL	64	128	192	256	320	384	448	512	576	640
2XL	32	64	96	128	160	192	224	256	288	320
XL	16	32	48	64	80	96	112	128	144	160
L	8	16	24	32	40	48	56	64	72	80
M	4	8	12	16	20	24	28	32	36	40
S	2	4	6	8	10	12	14	16	18	20
XS	1	2	3	4	5	6	7	8	9	10

Complex queries / large datasets

Concurrent queries (users/queries)

Data Storage

19 November 2021 14:00

Data Storage Layer

→ Storage layer is a cloud storage layer
  ↳ Depends on which cloud provider you use - you use their storage but it is under the Snowflake service - managed by Snowflake & all metadata is stored in the cloud service layer. Snowflake owns that storage & manages it for their customers

Storage Layer

→ Made up of objects or files called micro-partitions

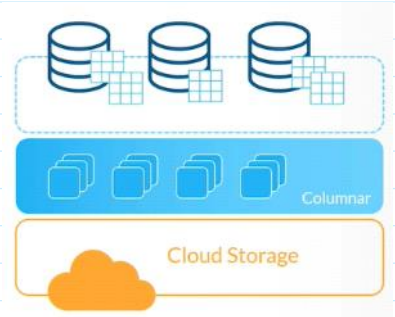
→ At a high level, micro-partitions stored in a hybrid columnar format

→ Automatic micropartitioning of table records into micro-partition objects & files

→ Natural data clustering & optimization

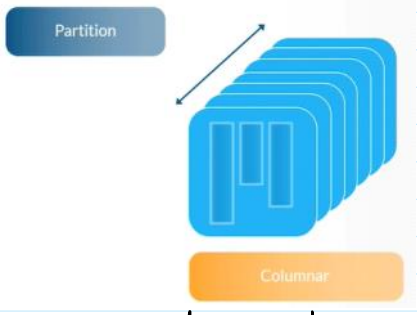
→ Native semi-structured data support (such as JSON, C, XML) & optimization

→ All Storage within Snowflake is billable (compressed)



Columnar Compression

→ Micro-partitions have columnar compression because they are in a hybrid columnar format



→ When we do a data load or data ingest, we automatically analyse & compress data into the table & into the micropartitions

→ Find the optimal compression scheme for each data type

→ Storing same data type enables efficient compression

→ Columns grow & shrink independently

→ Significant performance benefit by reduction I/O & storage

Micro-partitions

→ Contiguous units of storage that hold table data
  → 50-500 MB of uncompressed data
  → Generally max 16 MB (compressed)

→ Many micro-partitions per table

→ Immutable!! → To update/delete — create a new version of the micropartition

→ Services layer stores metadata about every micro-partition
  → MIN/MAX (Range of values in each column)
  → Number of distinct values

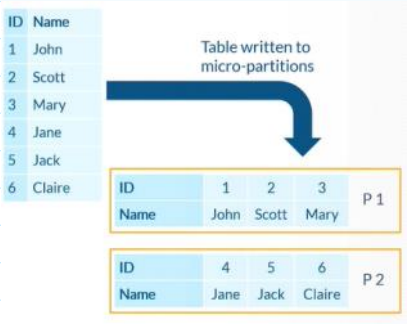
Micro-Partitioning

→ Physical data files that comprise Snowflake's logical tables

→ Automatically created contiguous units of storage, partitioned based on ingestion order

→ Attempts to preserve natural data co-location

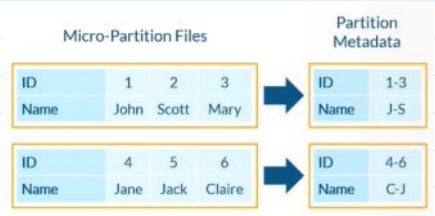
→ Immutable - updates create new micropartition versions



Metadata

→ Snowflake automatically collects & maintains metadata about tables & their underlying micro-partitions including:-

- Table level
  - Row count
  - Table size (in bytes) ↗ what table version?
  - File references (which micropartitions to use) & table versions
- Micro-partitions column level:
  - Range of values
  - Number of distinct values
  - MIN & MAX values
  - NULL count



-- Snowflake knows it has to scan only 1 micropartition and prunes away the excesses

```
SELECT *
FROM table
WHERE ID = 2
```

Data Storage Billing

→ Billed for actual storage use (compressed)
  Daily average terabytes per month

→ On-demand pricing
  Billed in arrears for storage used
  $40/Terabyte/month
  Minimum monthly charge of $25

→ Pre-purchased capacity
  Billed upfront with commitment to a certain capacity
  Price varies on amount & cloud platform
  Customer is notified at 70% of capacity

Data Protection in Snowflake

22 November 2021 10:32

Is your data safe?

→ Encrypted at rest & in motion
→ Replication across availability zones ↗ cloud provider's storage classes & mechanisms does this automatically
→ Time travel, Fail Safe & Replication Protection

Encryption

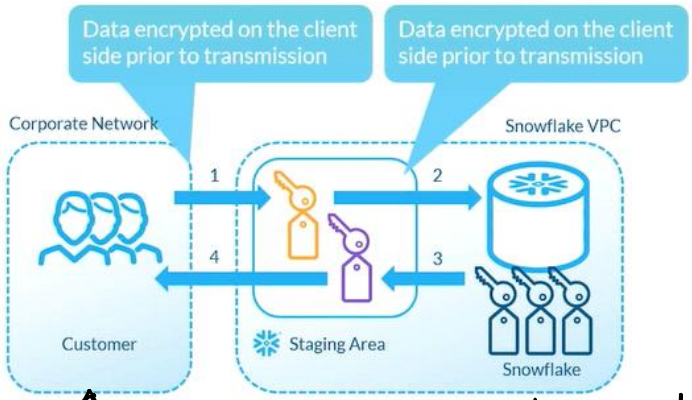
Encryption at rest & in motion
↳ Hierarchical key model
↳ Automatic key rotation → keys rotated every 30 days
↳ Periodic re-keying (Enterprise edition & above)

```
ALTER ACCOUNT SET
PERIODIC_DATA_REKEYING = true;
```

Tri-Secret Secure (BYOK)
↳ Composite master key: combine customer key with a Snowflake maintained key
Customer key has to stay available & if it isn't any system services will be revoked

Snowflake-Provided Staging Area

Automatic End 2 End Encryption



↑ compresses & encrypts file prior to transmission

→ Only the customer & the Snowflake service have access to the data

→ TLS 1.2 on HTTPS

→ File as well as communication is encryted using HTTPS & TLS1.2

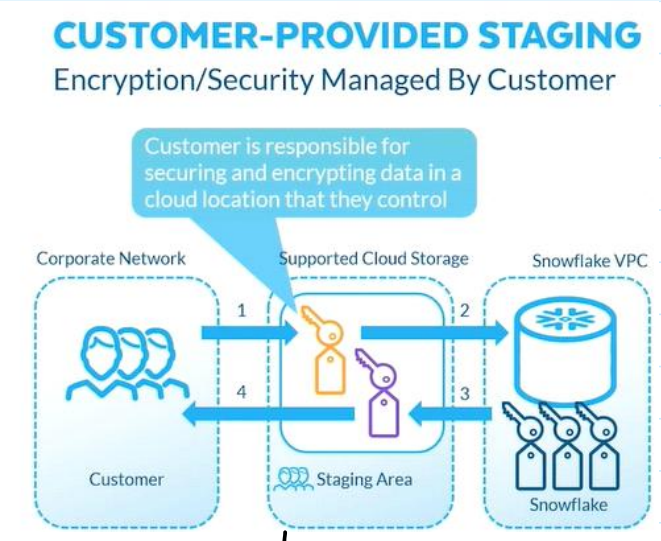
→ File gets encrypted in staging area with an encrytion key

→ Actual data load (2) is a copy into statement

→ Each micro-partition has its own encryption key

→ Each table has its own encryption key as well as an account master key

→ Customer may also choose to manage their own cloud storage (not using Snowflake's services)



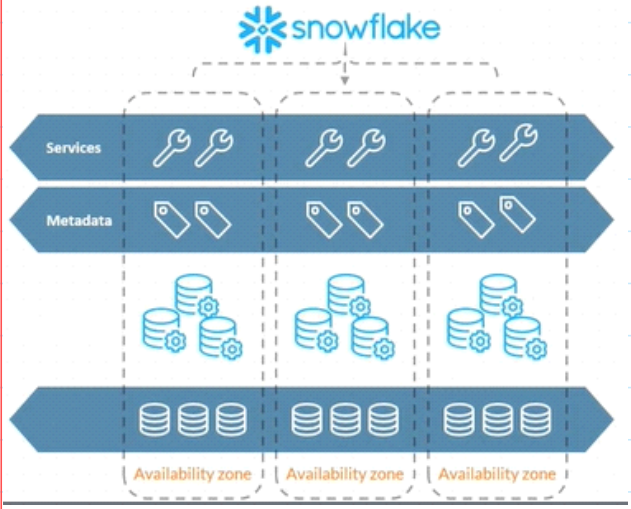
↳ AWS S3 buckets
↳ Azure blob storage
↳ GCP Cloud Storage

Built-In Continuous Availability

→ Transparently synchronises data across availability zones
↳ Geographic separation
↳ Separate power grids

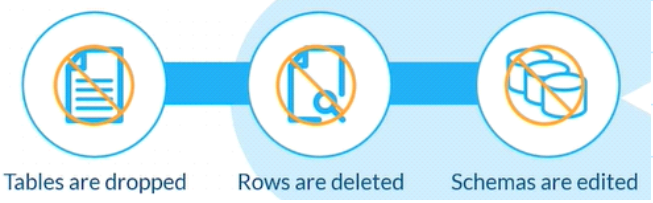
→ Fully online updates & patches

→ Time Travel & Fail Safe protection



Problem
Recovering From Mistakes?

→ User errors
→ System errors
→ Backup itself is a time-consuming task
→ Specialized skill & management overhead



Time Travel

→ Access historical data at any point within a defined retention period

→ UNDO common mistakes

→ Protect against accidental or intentional modification, removal or corruption
  ↳ Fix drops, deletes, edits

→ Backup / Restore from time or ID

→ Backup / Restore from time or ID

Standard Edition → 0 - 1 day
Enterprise Edition → 0 - 90 days } Configurable at account level, per database, per schema or on a per table basis
Business Critical
VPS

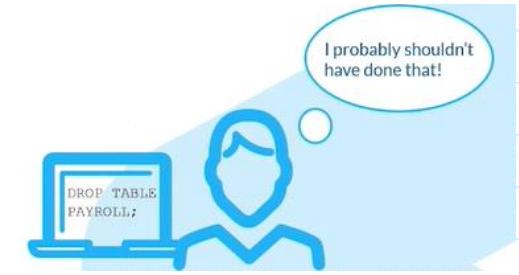
Allows :-
→ Undo mistakes by 'undropping' objects
→ Query historical databases within the defined period
→ Backup & restore functionalities

Time Travel SQL Extensions

→ Instantly bring back deleted tables, schemas, databases

→ Restore or duplicate data from key points in the past
↳ Point-in-time
↳ Prior to a specific query ID

→ Set retention times at the table, schema, database or account level



UNDROP <object>
CREATE <object> CLONE... AT|BEFORE
SELECT... FROM <table> AT|BEFORE
DATA_RETENTION_TIME_IN_DAYS

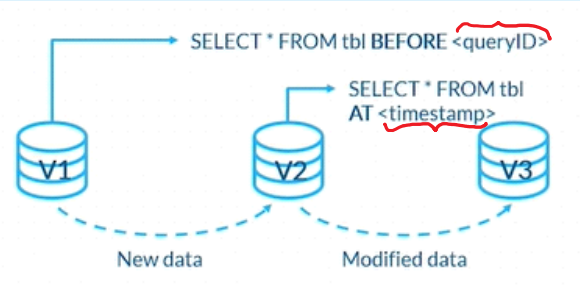
Time Travel

Configuration retention option:

DATA_RETENTION_TIME_IN_DAYS
Disable by setting to 0

DATA_RETENTION_TIME_IN_DAYS
Disable by setting retention to 0
Default is 1 day
Table churn and retention can increase storage costs

SQL extentions:
AT | BEFORE – querying clause
UNDROP - recovery




```
-- If you don't want time travel on a table, default is 1
ALTER TABLE tbl
SET DATA_RETENTION_TIME_IN_DAYS = 0;
```

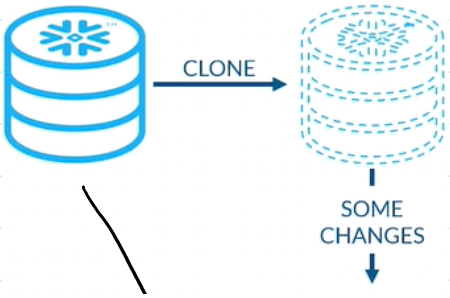
Zero Copy Cloning

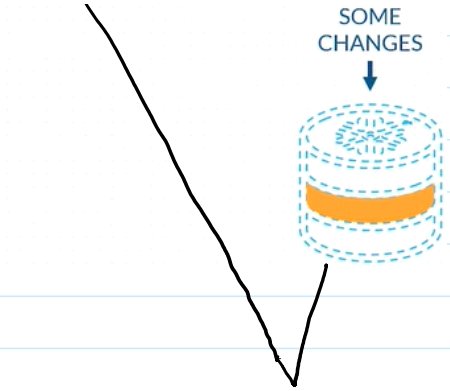
→ Quickly take a snapshot of any table, schema or database
→ When the clone is created:
  ↳ All micropartitions in both tables are fully shared
  ↳ Micropartition storage is owned by the oldest table, clone references them

→ No additional storage costs until changes are made to the original or clone

→ Often used to quickly spin up Dev or Test environments

→ Effective "backup" option as well





Become independent

Fail Safe Storage

→ Non configurable, 7 day retention for historical data after time travel expiration

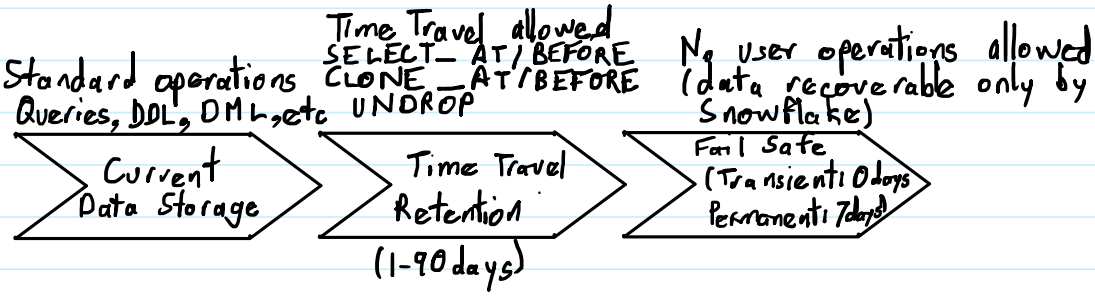
→ Accessible by Snowflake personnel

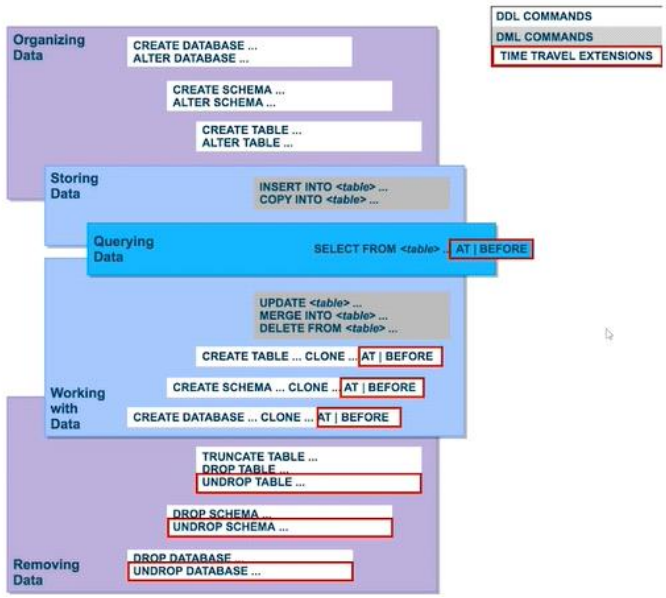
→ Admins can view fail-safe use in Snowflake WebUI under Account → Billing & Usage

→ Not supported for temporary or transient tables

Understanding & Using Time Travel

Continuous Data Protection lifecycle





AT or BEFORE in a query is specified in the FROM clause immediately after the table name.

Syntax:

```
SELECT ...
FROM ...
        {
                -- Timestamp or timestamp with offset or a specific query id
                AT ( {TIMESTAMP => <timestamp> | OFFSET => <time_difference>| STATEMENT => <id>} ) |
                -- Only supports query ids
                BEFORE (STATEMENT => id)
        }
[ ... ]
```

Demo

```
USE instructor_db_tt_demo;
SHOW parameters in database;

ALTER DATABASE SET data_retention_time_in_days = 7;
SHOW parameters in database;

-- No table,  no magic here
DESC TABLE t1;

-- Create a basic table
CREATE TABLE t1 (c1 string, c2 string);

-- Empty table
SELECT *
FROM t1;

-- Insert first row
INSERT INTO t1 values ("A", "B");

-- Insert second row and make a note of the query id
INSERT INTO t1 values ("C", "D"); -- <query id>

SHOW PARAMETERS in TABLE t1

-- Insert third row
INSERT INTO t1 values ("E", "F");

SELECT *
FROM t1;
```

```
SELECT *
FROM t1
AT(statement => '<query_id>')

-- Note that this is the version of the table right before the statement above
SELECT *
FROM t1
BEFORE(statement => 'query_id')

--alternatively --time offsets in seconds
SELECT *
FROM t1
AT(OFFSET => -60);

-- error if past Time Travel retention window

-- Mix both of them in the same query to find things that have changed

SELECT *
FROM t1
minus
SELECT *
FROM t1 AT(STATEMENT => '<query_id>')

-- Oops, we dropped the table
DROP TABLE t1;
SELECT * FROM t1;

SHOW TABLES history;

-- Lets get back our table
UNDROP TABLE t1

-- Read the table
SELECT *
FROM t1

CREATE TABLE T2
CLONE t1
 AT(statement => 'query_id')
```

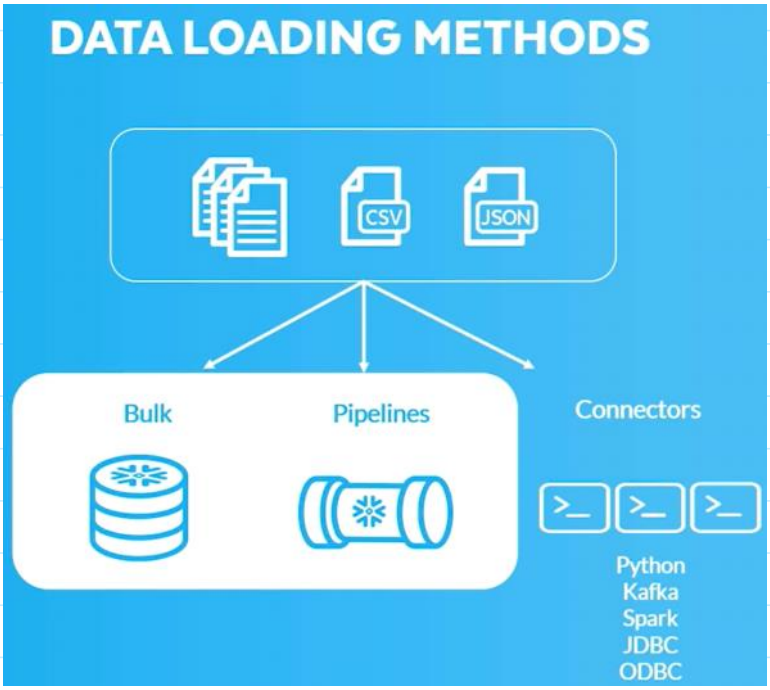
Infrastructure

22 November 2021 17:44

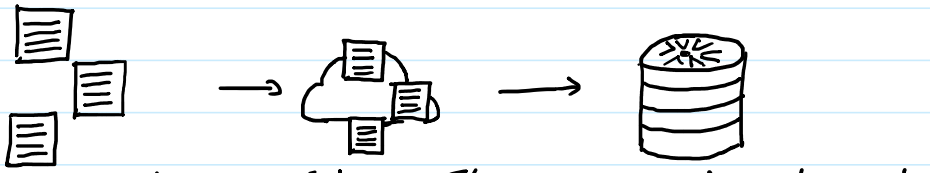
→ Cloud providers physical security
→ Cloud providers redundancy → backup, power redundancy
→ Limitless elasticity → scale up & down, scale out & back
→ Regional data centers - US, EU, AP

Data Movement Overview

29 November 2021 08:07



Data Loading Steps

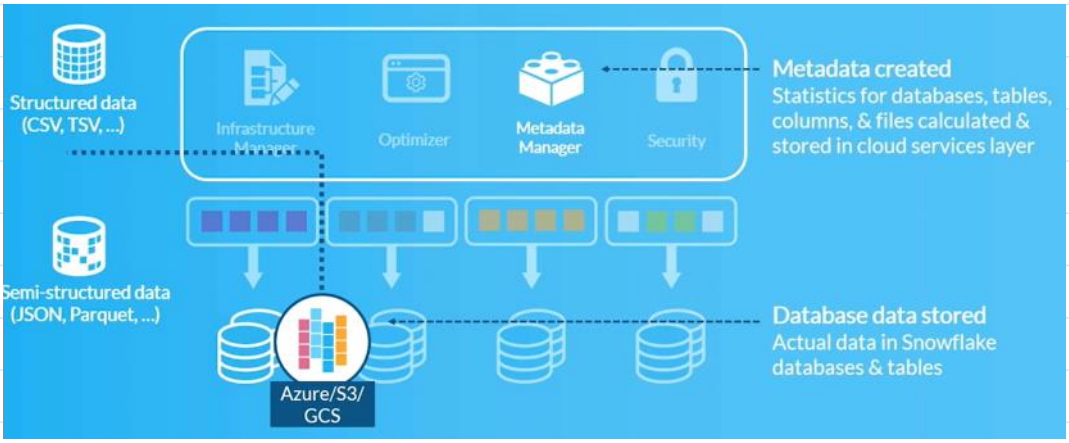


Output data to files

Stage Files to Cloud Storage

Load data into a table

Data Loading in Snowflake



Whether structured or semi-structured data

→ Create micropartitions (contains bunch of

rows from the table, metadata about them)
→ Compression & encryption as data goes into the table

Stage

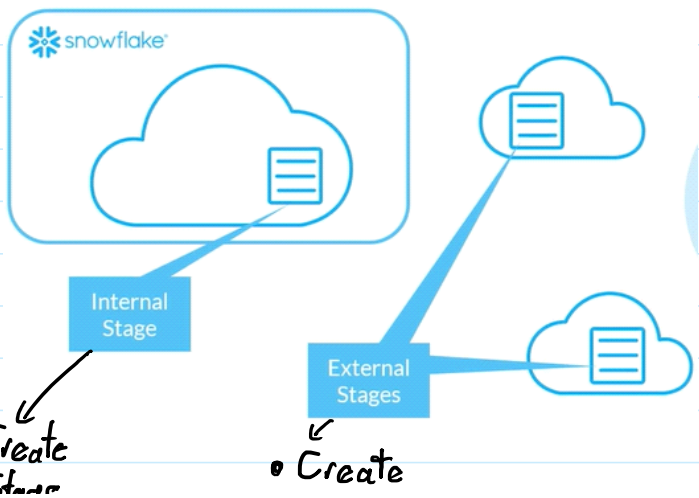
→ Cloud file repository / storage location

→ Simplifies & streamlines data movement

→ Can be internal or external

Internal → stored internally on Snowflake
External → stored in an external location – S3, Azure Blob, GCS

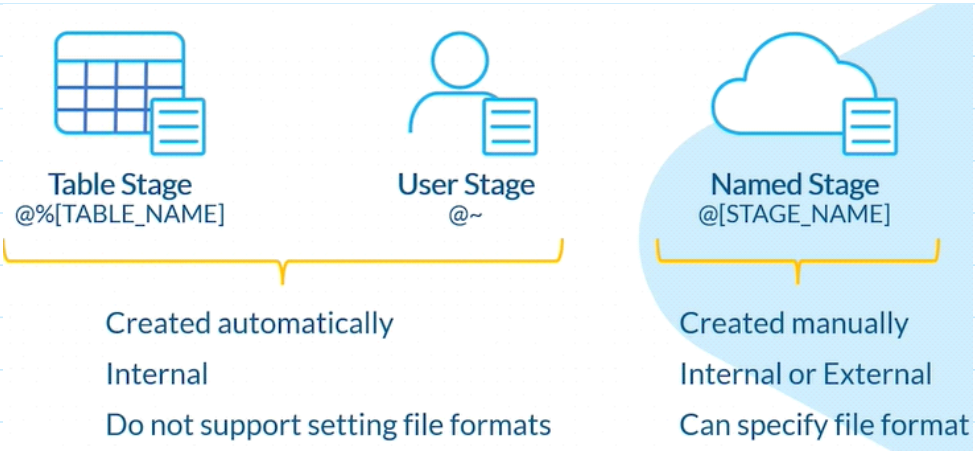
→ Used for all methods of data load/unload (Bulk, Pipe, Connectors)



- Create
- Stage
- Name

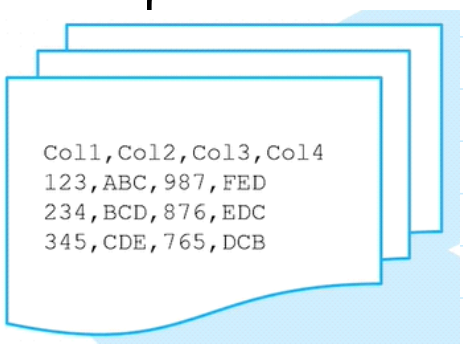
- Create
- Stage
- Name
- URL/URI of file location
- Credentials for access
- Decryption keys for decrypting data before loading into Snowflake

Types of Stages



File Format

→ Named object within a Schema
→ Instructs Snowflake how to parse files during load/unload
File types: CSV, JSON, Parquet, AVRO, ORC, XML
→ Specify file format:
In the COPY INTO command
As part of a named stage or pipe definition
As part of a table definition




```
CREATE FILE FORMAT MY_CSV_FORMAT
    TYPE = 'CSV'
    FIELD_DELIMITER = ','
    RECORD_DELIMITER = '\n'
    SKIP_HEADER = 1;
```

Pipe

→ Enables streaming/pipeline/micro-batch loading as opposed to bulk loading

→ A pre-defined object that specifies the COPY INTO statement syntax for a data load

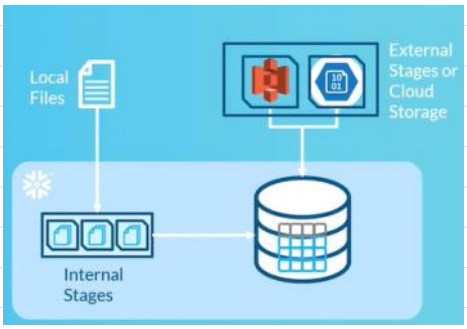
→ Instructs Snowflake how & where to load/copy/convert data into records in the forget table

→ All data types are supported, including semi-structured data types such as JSON and Avro

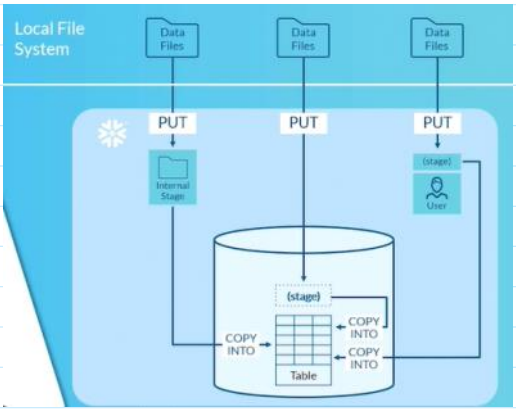
Bulk Loading

29 November 2021 09:49

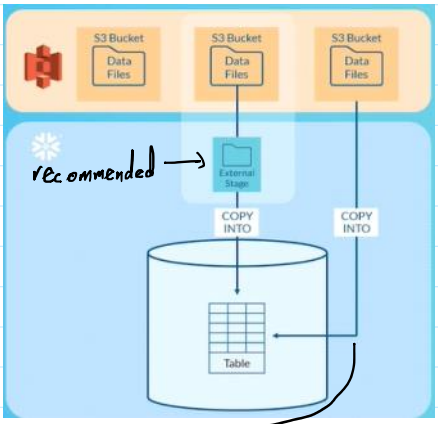
With bulk loading, we can ingest files from
→ local filesystem } Needs creation of a stage
→ cloud



Loading from local systems



loading from Cloud Storage



Same process flow for:
1) AWS
2) Azure
3) Google Cloud

↳ Not recommended !!
↳ Specify the location, credentials & the decryption keys everytime
↳ Much more secure to do this once & securely store this as part of the stage than to provide the information every time

Example: From Local Storage

1) Create internal stage

```
CREATE STAGE my_stage
    FILE_FORMAT = my_csv_format;
```

2) PUT data to stage

```
PUT FILE:///data/data.csv @my_stage
```

3) COPY data to table

```
COPY INTO my_table FROM @my_stage;
```

Note: Compression during PUT requires local resources. The local host needs sufficient memory & space in /tmp, or the PUT will fail

Example from Cloud Storage

1) Create external stage (pointer to cloud storage location)

```
CREATE STAGE my_s3_stage
url='s3://mybucket/encrypted_files/'
CREDENTIALS = (aws_key_id='*******'
aws_secret_key='*******')
ENCRYPTION=(master_key = '**********')
FILE_FORMAT = my_csv_format;
```

2) Load data using COPY

```
COPY INTO my_table
FROM @my_s3_stage
PATTERN='.*sales.*.csv'
```

Data Loading Demo

```
-- Set your context
USE_ROLE_TRAINING_ROLE;
USE WAREHOUSE_LOAD_WH;
USE DATABASE MY_DB;
USE SCHEMA PUBLIC;

-- Create the table to load
CREATE OR REPLACE TABLE REGION (
        R_REGIONKEY NUMBER(38,0) NOT NULL
        R_NAME VARCHAR(25) NOT NULL,
        R_COMMENT VARCHAR(152) NOT NULL,
        R_STATUS VARCHAR
);

-- Create the file format
CREATE OR REPLACE FILE FORMAT MY_CSV_FORMAT
        TYPE='CSV'
        FIELD_DELIMITER = "|";

DESCRIBE FILE FORMAT MY_CSV_FORMAT;

-- List the stage
LIST @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/pattern='.*region.tbl';

-- Copy the data into the table
COPY INTO REGION
        FROM @TRAINING_DB.TRAININGLAB.ED_STAGE/load/lab_files/
        FILES = ('region.tbl')
        FILE_FORMAT = ( FORMAT_NAME = MY_CSV_FORMAT );

-- Select and review the data in the REGION TABLE
SELECT
        R_REGIONKEY,
```

R_NAME,
FROM REGION;

Copy into Command Parameters

Optional items in blue

```
COPY INTO [<namespace>.]<table_name>
  FROM { <stage> | <external location> }

[ FILES = ( '<file_name>' [ , '<file_name>' ] [ , ... ] ) ]
[ PATTERN = '<regex_pattern>' ]
[ FILE_FORMAT = ( { FORMAT_NAME = '[<namespace>.]<format_name>' |
                    TYPE = { CSV | JSON | AVRO | ORC | PARQUET | XML }
                    [ <format_type_options> ] } ) ]
[ VALIDATION_MODE = RETURN_<n>_ROWS | RETURN_ROWS | RETURN_ALL_ERRORS ]
[ <copy_options> ]
```

The COPY command has several options for handling data and errors (defaults are shown in **bold**)

SIZE_LIMIT = <num_bytes> //stop if input data has size greater than size limit after loading upto size limit
PURGE = TRUE|**FALSE** //Delete from the stage any files that have been successfully loaded (you pay for storage in stage)
RETURN_FAILED_ONLY = TRUE|**FALSE** //return only failed rows
ENFORCE_LENGTH = **TRUE**|FALSE
TRUNCATECOLUMNS = TRUE|**FALSE** } If a column has a string of length greater than defined for the column use this to control what happens
FORCE = TRUE|**FALSE**
LOAD_UNCERTAIN_FILES = TRUE|**FALSE**
ON_ERROR = <error handling option>

→ Reload files that have already been loaded

→ If you leave files in stage for too long, metadata is kept for about 64 days after which Snowflake is no longer sure if it has loaded that file. If you want it to load even those uncertain files set this option to TRUE

ERROR HANDLING OPTIONS

USE ON_ERROR option to control how errors in the data load are handled

Supported Values	Notes
CONTINUE	Continue loading the file.
SKIP_FILE	Skip file if any errors are encountered in the file.
SKIP_FILE_<num> (e.g. SKIP_FILE_10)	Skip file when the number of errors in the file is equal to or exceeds the specified number.
SKIP_FILE_<num>% (e.g. SKIP_FILE_10%)	Skip file when the percentage of errors in the file exceeds the specified percentage.
ABORT_STATEMENT (default)	Abort the COPY statement if any error is encountered.

default option

Validate data loads

PRE-LOAD: COPY INTO - Validation MODE

Allows you to test a data load without executing it

```
[VALIDATION_MODE = RETURN_<n>_ROWS |
RETURN_ERRORS | RETURN_ALL_ERRORS]
```

POST- LOAD: Table Function - VALIDATE

Validates the files loaded in *a past execution* of the COPY INTO <table> command

Returns all the errors encountered during the load

```
VALIDATE( [<namespace>.]<table_name> ,
JOB_ID => { '<query_id>' | _last } )
```

Data Loading

29 November 2021 12:05

Data Loading Recommendations

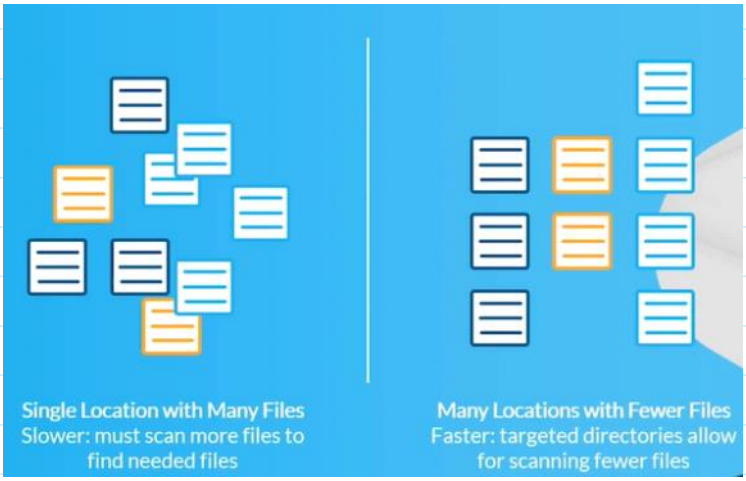
→ File size & number of files are crucial to data load performance
Split large files before loading into Snowflake → single server can process 8 files in parallel
Recommended: 10MB to 100 MB (Compressed)

→ Organise data in logical paths (eg subject area & create date
eg. /system/market/daily/2018/09/05/

same month files will be collocated in micropartition so will get better pruning

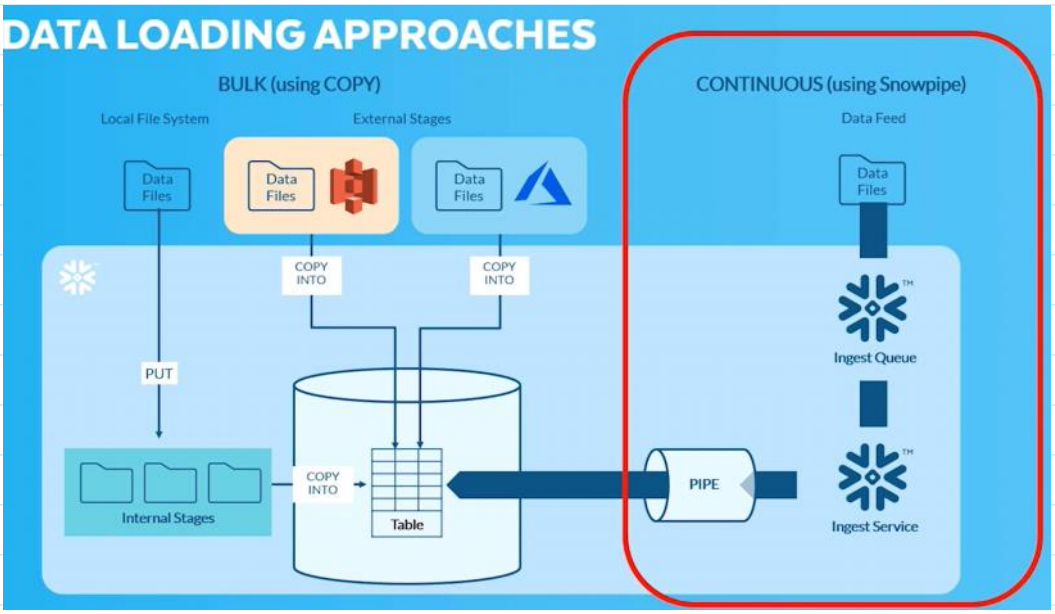
Warehouse Size	# servers	Concurrent files
XS	1	8
S	2	16
M	4	32
L	6	64
XL	16	128

File Locations



Continuous Data Loading

29 November 2021 12:59



COPY Command (BATCH)	Snowpipe (Continuous)
Migration from traditional data sources	Ingestion from modern data sources
Transaction boundary control BEGIN/START TRANSACTION/ COMMIT/ROLLBACK	Continuously generated data is available for analysis in seconds
Independently scale compute resources for different ingestion workloads	No scheduling (with auto-ingest)
	Serverless model needs no user-managed virtual warehouse

Batch ← Micro-Batch → Continuous

Pipe

Source application loads stage

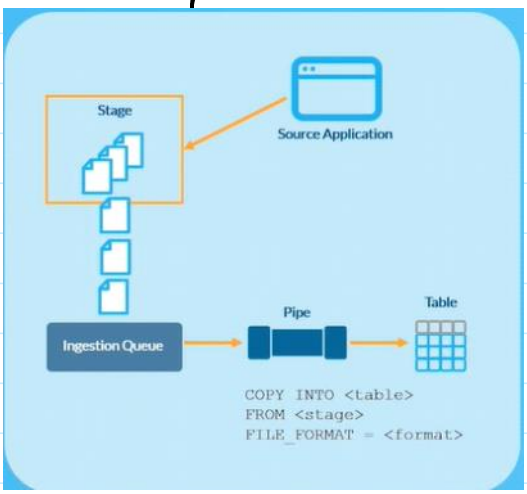
REST API Call to Snowpipe OR notification received from cloud

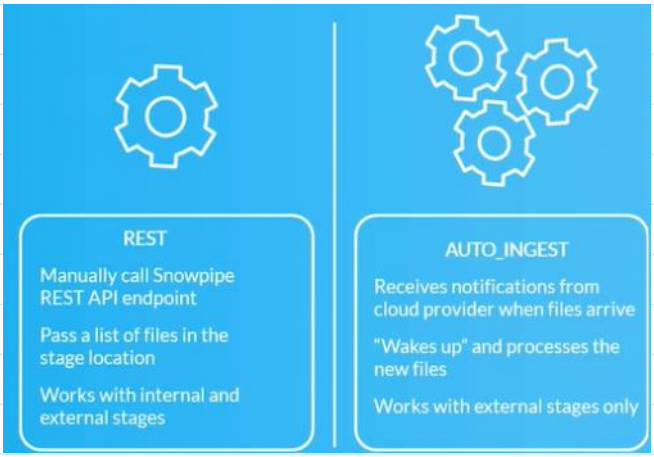
Files moved from stage to ingestion queue

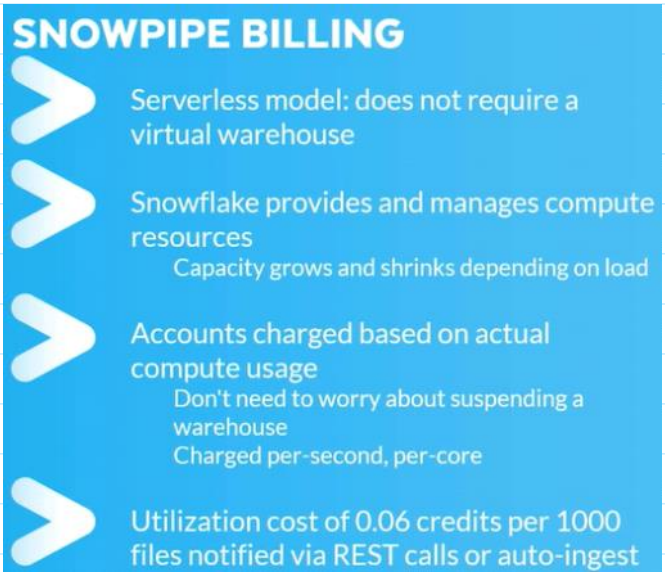
Pipe contains a COPY INTO statement
Source stage for data files
Target table

Load data into tables continuously from an ingestion queue

Can be paused/resumed, return status






SNOWPIPE BILLING
Serverless model: does not require a virtual warehouse
Snowflake provides and manages compute resources
Capacity grows and shrinks depending on load
Accounts charged based on actual compute usage
Don't need to worry about suspending a warehouse
Charged per-second, per-core
Utilization cost of 0.06 credits per 1000 files notified via REST calls or auto-ingest

Continuous Data Processing

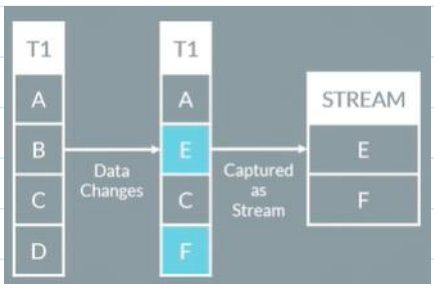
29 November 2021 13:55

Streams & Tasks

Streams

→ Change Data Capture

→ Identify & Act on changed table records



Tasks

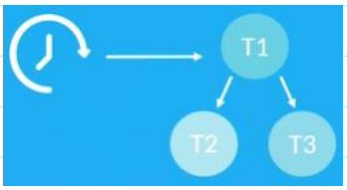
Scheduled SQL Execution

Can create a tree of tasks where one task follows another

Schedule interval set in minutes or CRON expression

May optionally be used with Streams

Example: Check a stream for data & process it



Working with Tasks

→ Executes a single DML statement or stored procedure call
→ Requires CREATE TASK on the schema
  Executes using the owning role
→ Can NOT be triggered manually
  Size warehouse via manual execution first
→ Use Cases:
  Keep aggregates up-to-date
  Generate periodic reports
  Copy data into or out of Snowflake
  Combining with streams to check if streams have data & acting on data

Working with Streams & Streams DDL

Working with Streams & Streams DDL

Working with Streams

→ Streams hold no data
→ Think of it like a bookmark
→ Standard & Append-Only
→ Stream Columns:
METADATA$ACTION
METADATA$ISUPDATE
METADATA$ROW_ID

Standard → Delta of changes

Streams DDL

CREATE
ALTER
DROP
DESCRIBE
SHOW
GRANT...TO ROLE
REVOKE...FROM ROLE
SHOW GRANTS

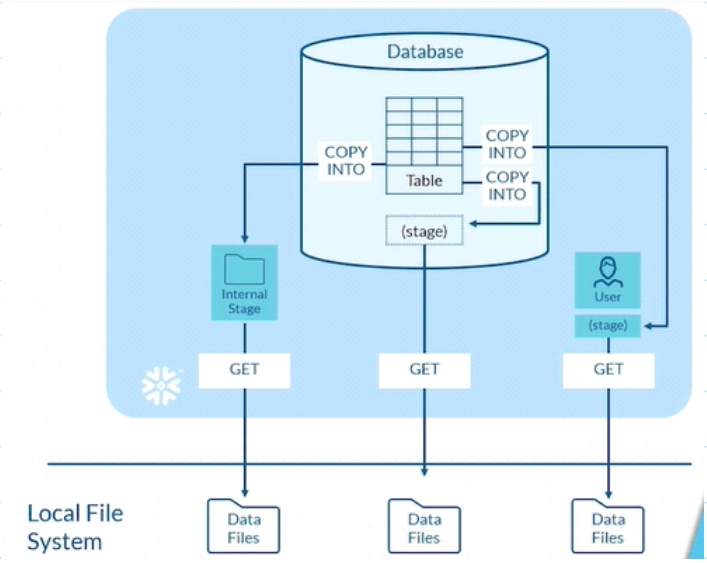
Demo

```
-- tasks
Create a task that inserts the current timestamp into a table every 5 minutes

CREATE OR REPLACE TASK mytask_minute
        WAREHOUSE = class_demo_wh
        SCHEDULE = '1 MINUTE'
AS
        INSERT INTO basic_table(ts) VALUES(CURRENT_TIMESTAMP);

SHOW TASKS;

SELECT *
FROM basic_table;

ALTER TASK mytask_minute RESUME;

-- Wait a couple of minutes
ALTER TASK mytask_minute suspend;



-- streams
CREATE STREAM mystream on table basic_table;

SHOW STREAMS;

SELECT *
FROM mystream;

SELECT system$stream_has_data('MYSTREAM');

--Create another stream (after the original data was inserted)
CREATE STREAM second_stream
ON table basic_table;

SHOW streams;

SELECT *
FROM second_stream;

SELECT system$stream_has_data('second_stream');



CREATE TASK mytask1
```

```
        WAREHOUSE = 'class_demo_wh'
        SCHEDULE = '5 minute'
WHEN
        system$stream_has_data('mystream')
AS
        INSERT INTO mytable(id, name) SELECT id, name FROM mystream WHERE METADATA$ACTION =
        "INSERT";

SELECT *
FROM basic_table;
```

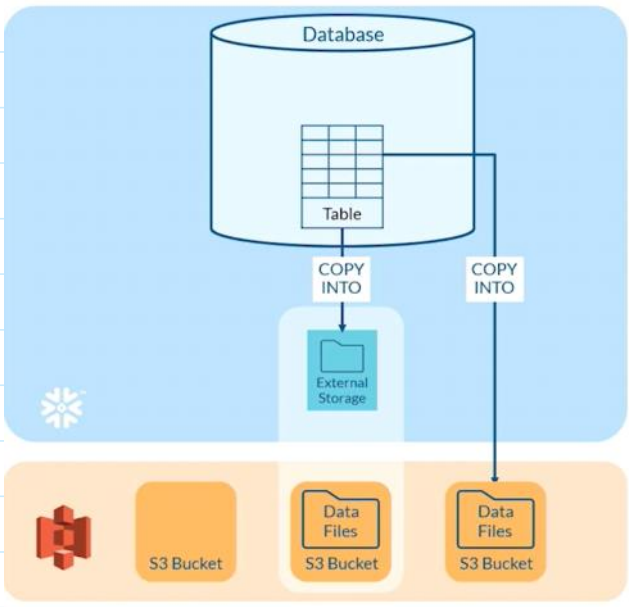
Data Unloading

30 November 2021 12:49



Unload to cloud

→ Use COPY INTO command
→ Recommend using external named stage
Can also unload directly by specifying the URL & any necessary credentials



Considerations

→ File formats

Flat (CSV, TSV, etc), JSON, Parquet

→ USE OBJECT_CONSTRUCT to create semi-structured format files from relational tables

→ File splitting
SINGLE or Multiple files option
MAX_FILE_SIZE can increase from default of 16 MB
↳ Limitations by cloud provider → Azure: 256MB, GCP, AWS: 5Gb

→ Use LIST & REMOVE command to manage files in stages

Empty strings vs Null Values

Handle NULLs & empty strings appropriately on unload

Enclose strings in double or single quotes:

```
FIELD_OPTIONALLY_ENCLOSED_BY = 'character' | NONE
```

Determine how empty fields are handled:

```
EMPTY_FIELD_AS_NULL = TRUE | FALSE
```

Convert SQL Null values:

```
NULL_IF = ('string1', [, 'string2'...])
```

Additional Functionality

1) Reorder Columns 2) Emit Columns 3) Casting
4) Substring

SELECT queries in COPY statements support the full syntax of Snowflake SQL queries

```
COPY INTO @my_stage FROM
(SELECT column1, column2
FROM my_table)
FILE_FORMAT = (FORMAT_NAME =
'my_format');
```

JOIN clauses enables downloading data from multiple tables

```
COPY INTO @my_stage
FROM (SELECT name, id1
      FROM my_table1
      JOIN my_table2 ON id1 = id2)
FILE_FORMAT =
FORMAT_NAME = 'my_format');
```

Account and Security

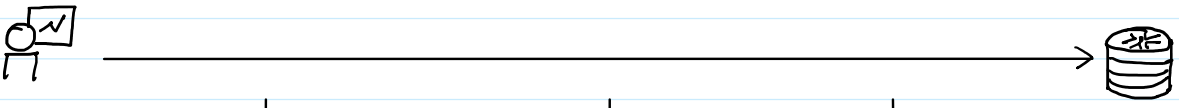
06 December 2021 10:29

Overview

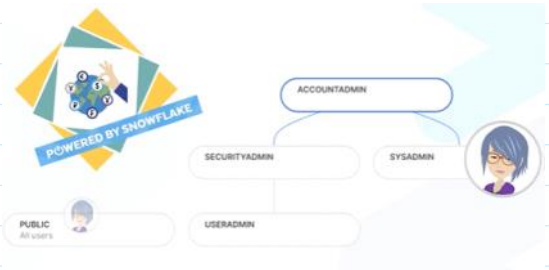
→ Access
→ Authentication
→ Authorization
→ Data Protection
→ Infrastructure ← Handled by cloud provider

Operational Controls

NIST 800-53
SOC2 Type 2
SOC1 Type II
ISO/IEC 27001:2013
HIPAA, PCI, FedRamp } Business Critical and Above



1	2	3.	4.
Access Restrict access to specified IP address or a range Optionally: Restrict via Secure Private Network	Authentication Authenticate users using a password, MFA or SSO	Authorization Restrict access to specific objects using roles & privileges	Data Protection Protect customer data using AES256 bit encryption & periodic rekeying



Access

06 December 2021 10:46

Can you get to the Snowflake Account?

⇌ IP allowlisting

TLS 1.2 for all communications

Support for cloud-provider private network solutions

Network Security

→ Network Policy support for IP allowlisting & blocklisting

→ User-Level Network Policy
  ↳ Assign Network Policy to individual User

→ Online certificate status protocol (OCSP) x.509 validation native to all Snowflake connectors

→ Support for cloud provider connectivity options like:
  ↳ PrivateLink (AWS & Azure) - no traffic over the public internet
  ↳ DirectConnect (AWS) - dedicated network connection on-premise to AWS

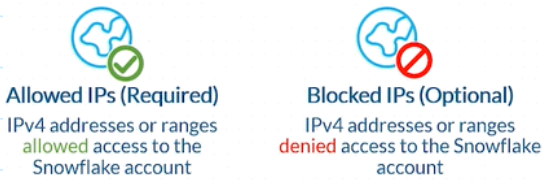
Account Level Network Policy

→ Restrict access to a Snowflake account to authorized network points (IPv4)

→ Allowlist or Blocklist

→ Use CIDR notation to specify a subnet range

→ One active network policy per account



→ Must be ACCOUNTADMIN or SECURITYADMIN

→ ALLOWED_IP_LIST is required,
BLOCKED_IP_LIST is optional

```
CREATE NETWORK POLICY <policy_name>
ALLOWED_IP_LIST = (<IP address or range>, ...)
BLOCKED_IP_LIST = (<IP address(es) in allowed range>)

ALTER ACCOUNT
SET NETWORK_POLICY = <policy_name>
```

User Level Network Policy

→ Restricts access by user based on allowed/blocked IP address list

→ User denied access if they attempt to log in from a restricted IP address

→ If a policy is assigned to a user who is already signed in, they are prevented from executing further queries

→ SQL only, not available in Web UI

```
ALTER USER <name>
  SET NETWORK_POLICY = <string>
```

Ensuring connectivity

Allowlist the following ports

- 443 → Required for general Snowflake traffic
- 80 → Required for OCSP cache server, which listens for all Snowflake client communication

Hostname Allowlisting:

All Snowflake clients require temporary access to cloud storage
Allowlist hostname patterns for the required hosts
Use SYSTEM$ALLOWLIST() to obtain required hostnames for your Snowflake account

Access flow steps

1. Enforce network policy set on the account 
2. Verify that the account is not locked/disabled 
3. Verify that the user is not locked/disabled 
4. Resolve the account + user from the given request 
5. Perform the basic authentication checks 

Authentication

06 December 2021 12:22

Who are you?

→ Username & password

→ Multi-factor authentication

→ Federated authentication (SAML 2.0)

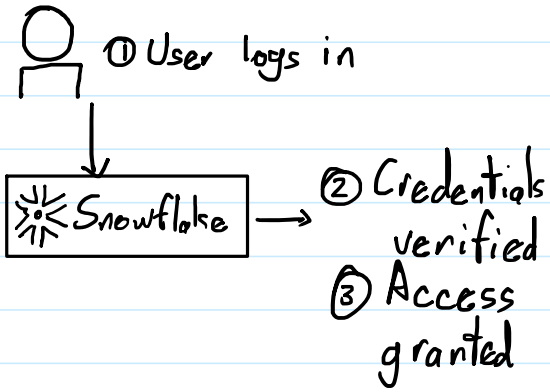
→ Other authentication methods (OAuth, Key-Pair, SCIM)

Username & Password (Default)

→ Verify the password given matches the one associated with the user

→ Default method

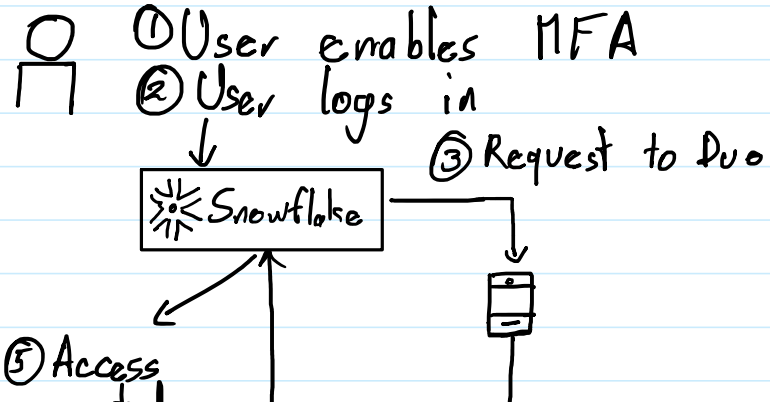
→ Requirements



8-256 characters
Atleast 1 uppercase letter
Atleast 1 lowercase letter
Atleast 1 number

Multi-factor Authentication

→ Requires an extra step after username/password

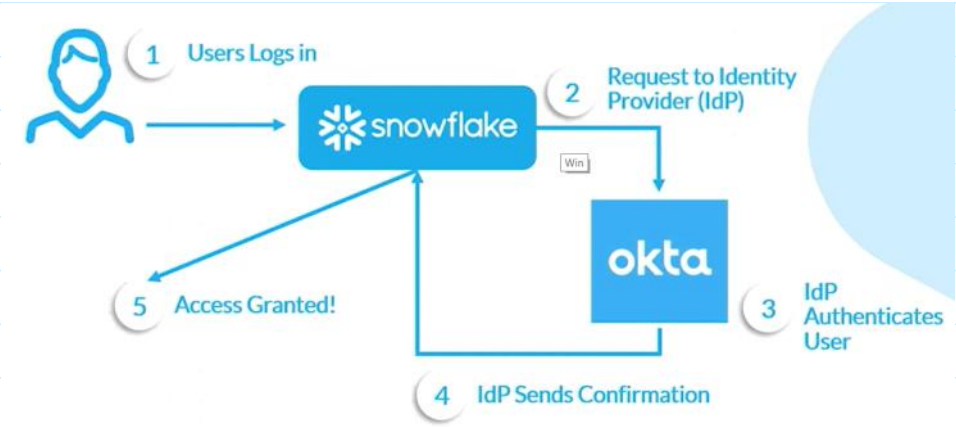


⑤ Access granted

④ Confirm on Smartphone or enter passcode

Federated Authentication (SSO)

→ Authenticate through an external identity provider such as Okta



OAuth: Authorisation on behalf of user from a third party service or application

KeyPair: JSON Web Token (JWTs) signed using a public/private key pair with RSA encryption

SCIM (System Cross-domain Identity Management): Automated Management of users & roles in cloud applications

Authorization

06 December 2021 13:21

What can you do?

→ Flexible & granular authorization

→ Secure views & UDFs to protect information access

→ Row level access control

Authorization Control

→ Combination of role-based & discretionary control
  ↳ Role based: Access privileges are assigned to roles
  ↳ Discretionary: Each object has an owner, which can grant others access to that object

→ Privileges that can be set depend on the securable object

→ Privileges can be broad or granular

Secure Views & UDFs

→ Prevent data from being indirectly exposed through programmatic methods

→ Only authorised users can see the view or UDF definition

→ Secure view & secure UDFs bypass some query optimizations to achieve greater security

Row-Level Authorization

Use CURRENT_ROLE or CURRENT_USER functions to provide row-level security

```
SELECT * FROM table_a
JOIN table_b ON
        table_a.role=table_b.auth
AND table_b.auth=CURRENT_ROLE;
```

Data Protection

06 December 2021 15:05

Is your data safe?

→ Data is encrypted at rest & in motion

→ Data is replicated across 3 availability zones in a region

→ Data protection features include Time Travel, Failsafe and Database Replication

Hierarchical Key Model

Root key

Account Master Keys (Every account has 1)
Organisation will have multiple accounts

Object Master Keys (Every object in Snowflake has 1)
↳ tables, file_format

File Keys (Actual keys protecting the micro-partitions

Key Life Cycle

Active : Used for encryption/decryption

Retired : Used for decryption only

Destroyed : No longer used

Key Rotation / Expiration

→ Keys automatically rotated every 30 days

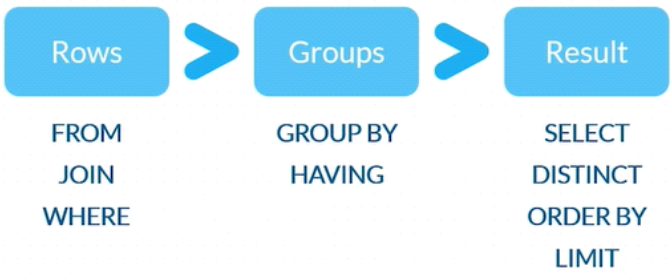
→ Option to annually re-key data (Enterprise edition & above)

→ Rotation / re-keying occurs with no customer impact

Performance and Tuning

06 December 2021 16:50

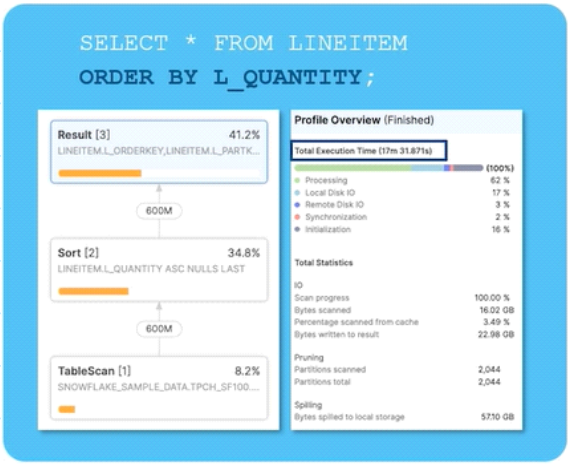
Typical Database Order of Execution

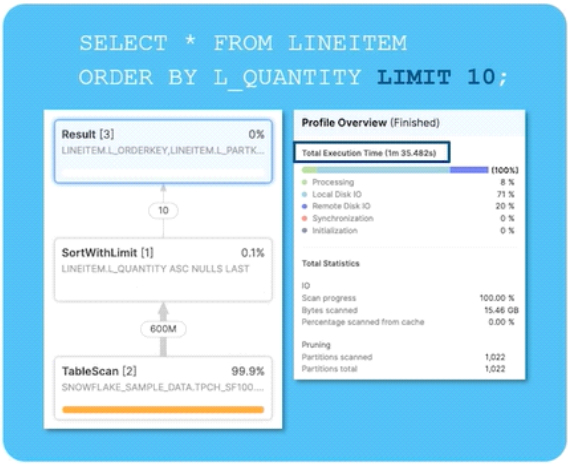


→ Row operations are performed before group operations

→ Check row operations first
↳ Check FROM & WHERE clauses
↓
Check GROUPBY & HAVING clauses

→ Use appropriate filters as soon as possible





Join on unique keys

- Best practices
→ Ensure keys are distinct
Understand relationships between your tables before joining
→ Avoid many-to-many join
→ Avoid unintentional cross join

Troubleshooting scenario

→ Joining on non-unique keys can explode your data output

→ Each row in table1 matches multiple rows in table 2

Snowflake built in optimizations

→ Snowflake provides patented micro-partition pruning to optimise query performance
  ↳ Static partition pruning based on columns in WHERE clause
  ↳ Dynamic partition pruning based on JOIN columns of a query

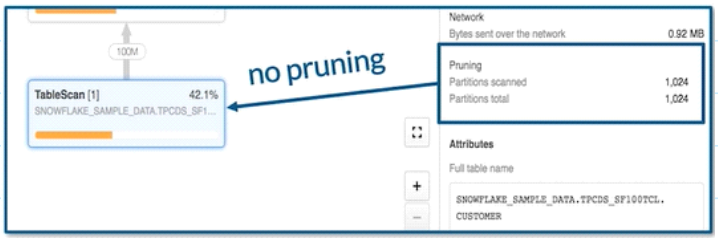
→ Best practices to assist SQL optimiser
  ↳ Apply appropriate filters as early as possible
  ↳ For naturally clustered tables apply predicate columns that have high correlation to ingestion order

```
SELECT <items>
FROM order
WHERE o_orderdate >= to_date('1993-10-01')
AND o_orderdate < dateadd(month, 3, to_date('1993-10-01'))
```

Not a good pruning query

Some WHERE predicates provide no opportunity to optimize pruning

```
SELECT c_customer_id, c_last_name
FROM tpcds.customer
WHERE UPPER(c_last_name) LIKE '%KROLL%'
```



Uncorrelated scalar subqueries

```
SELECT p.name, p.annual_wage, p.ctry FROM
pay AS p
WHERE p.annual_wage < (SELECT per_capita_gdp FROM intl_gdp
WHERE country = "Brazil")
```

Correlated scalar subqueries

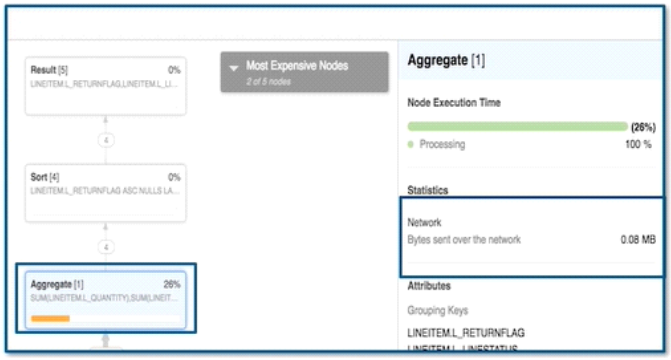
```
SELECT p.name, p.annual_wage, p.ctry FROM
pay AS p
WHERE p.annual_wage < (SELECT MAX(per_capita_gdp) FROM intl_gdp i
                       WHERE p.ctry = i.country)
```

Change for every row in the table

takes much longer !!!

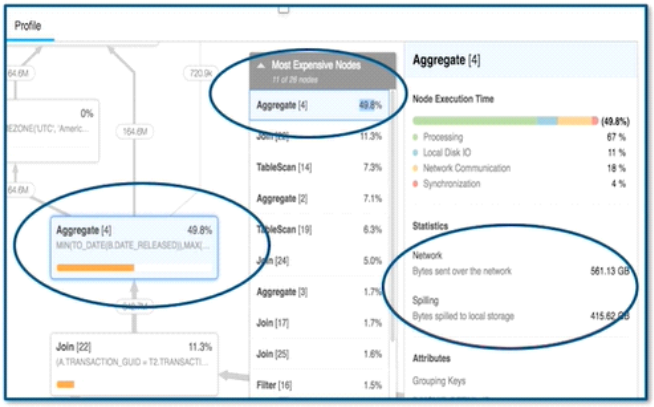
GROUP BY WITH FEW DISTINCT VALUES

```
SELECT
    l_returnflag,
    l_linestatus,
    SUM(l_quantity)
FROM lineitem
GROUP BY
    l_returnflag,
    l_linestatus;
```



GROUP BY WITH MANY DISTINCT VALUES

Memory-intensive
Spilling to disk
High network data
Sub-optimal performance



ORDER BY

Orders values in ascending or descending order on a specified column

Use ORDER BY in top level SELECT only

ORDER BY in subqueries does not impact the result, and slows performance

```
select
      o_orderpriority,
      count(*) as order_count
 from
      orders
 where
      o_orderdate >= to_date('1993-07-01')
   and o_orderdate < dateadd(month, 3, to_date('1993-07-01'))
   and exists (
       select
           *
       from
           lineitem
       where
           l_orderkey = o_orderkey
       and l_commitdate < l_receiptdate
       order by l_orderkey
       )
group by
      o_orderpriority
order by
      o_orderpriority;
```

Wasted Compute

Best Practice

Data Clustering

07 December 2021 10:00

Natural Data Clustering

→ As data is loaded into tables, micro-partitions are created based on ingestion order

→ Ingestion order may highly correlate with one or more columns

Tables with a sequential field
Tables with a date field

ID	NICKNAME	YOB
1	PRISCILLA	1962
2	DAWG	2001
3	JOKER	1997
4	PRINCESS	1998
5	SNEEZY	1983

DATE	ORDER_NUMBER	ITEM_TYPE
22-Jul-2019 23:29:07	2019_072111356	Office Supplies
22-Jul-2019 23:32:00	2019_072111357	Office Furniture
22-Jul-2019 23:44:56	2019_072111358	Services
22-Jul-2019 23:59:01	2019_072111359	Office Supplies
23-Jul-2019 00:01:07	2019_072111360	Printers

What determines Natural Clustering?

→ A single data load reads source data and writes it into some number of micro-partitions

→ The source data organisation determines what range of values are represented in each micro-partition

So if:-

- Source data contains rows from a specific day: that day's data will be in contiguous micro-partitions
- Source data contains rows for a month or year: those micro-partitions will have high/low values for dates within that month or year
- Source data is ordered alphabetically on a 'last name' column: contiguous micropartitions will be in alphabetical order

Tables & Clustering Keys

→ All tables have a 'natural' clustering based on ingestion order

→ Natural clustering can be overridden by designing clustering keys

↳ Clustering keys can be columns or expressions

→ After clustering, like data (by key) is co-loaded in the same micro-partitions

→ Snowflake keeps the clustering order updated in the background billed on a per-second basis

Note: Any number of keys can be defined
Recommended ≤ 3 (more is NOT better)
If multiple clustering keys:
↳ Lowest cardinality key 1st

What Tables should be clustered?

→ Clustering keys are not for every table!
↳ Automatic clustering consumes credits
↳ Reclustering also consumes credits

→ Good candidates for clustering keys
↳ Tables in the multi-terabyte range
↳ Query performance degrades noticeably over time

→ Keep in mind
↳ Frequently changing tables are more expensive to cluster
↳ Automatic clustering can be suspended or resumed

Clustering Command Samples

```
CREATE TABLE t1 (c1 date, c2 string, c3 number)
CLUSTER BY (c1, c2);

-- Can also have expressions in CLUSTER BY statements
CREATE TABLE t1 (c1 timestamp, c2 string, c3 number)
CLUSTER BY (TO_DATE(c1), SUBSTRING(c2, 0, 10));

ALTER TABLE t1
CLUSTER BY (c1, c3);

ALTER TABLE t2
CLUSTER BY (SUBSTRING(c2, 5, 10), TO_DATE(c1));
```

table name

```
SELECT
SYSTEM$CLUSTERING_INFORMATION('LINEITEM', '(L_ORDERKEY)');
```

Order of keys

Details

```
    keys.html for more information.",
4     "total_partition_count" : 1022,
5     "total_constant_partition_count" : 0,
6     "average_overlaps" : 1021.0,
7     "average_depth" : 1022.0,
8     "partition_depth_histogram" : {
9       "00000" : 0,
10      "00001" : 0,
11      "00002" : 0,
12      "00003" : 0,
13      "00004" : 0,
14      "00005" : 0,
15      "00006" : 0,
```

Copy

Not much pruning advantage

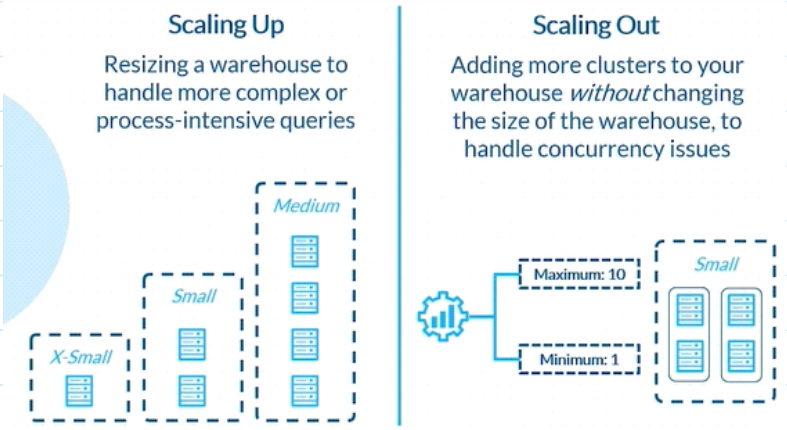
depth = # partitions

Not well clustered

Virtual Warehouse Scaling

07 December 2021 11:01

Scaling up vs. Scaling Out



Servers per cluster

→ As queries are submitted, required resources are calculated & reserved

→ If there are insufficient resources, that query is queued until other queries finish & release resources

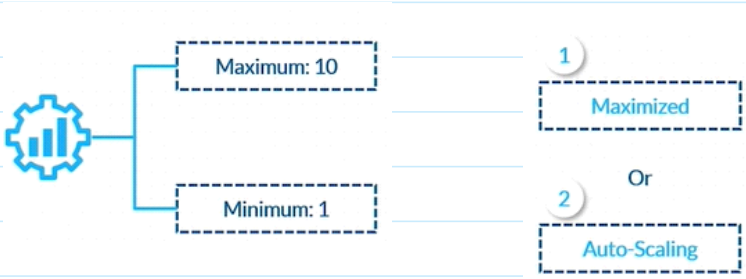
Warehouse Size	Servers	Clusters
X-Small	1	1
Small	2	1
Medium	4	1
Large	8	1
X-Large	16	1
2X-Large	32	1
3X-Large	64	1
4X-Large	128	1

Scaling Out: Multi-Cluster Warehouses

→ Multi-cluster warehouses are designed specifically for handling queueing & performance issues related to large numbers of concurrent users and/or queries

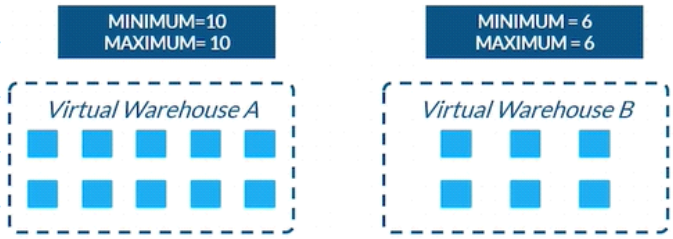
→ Multi-cluster warehouses help automate this process if

your number of users/queries tends to fluctuate



Multi-Cluster Warehouse : Maximized

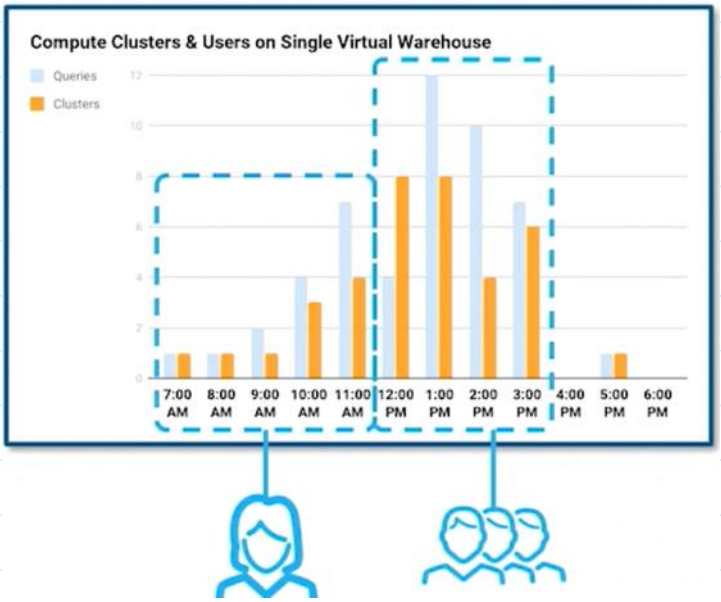
When the virtual warehouse is started, Snowflake starts all the clusters so that maximum resources are available when the warehouse is running



→ Recommended for very stable & predictable usage curve.

Multi-Cluster Warehouse : Auto Scaling

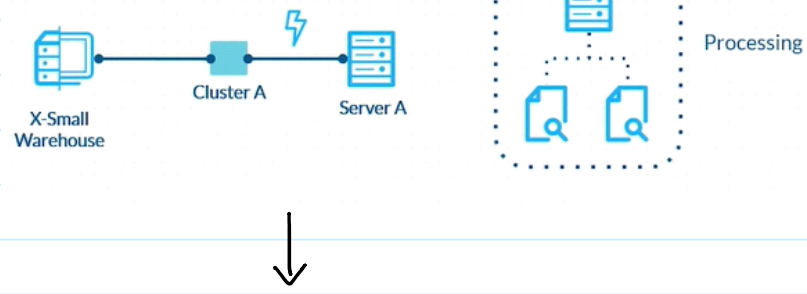
Snowflake will start & stop clusters as needed to dynamically manage the load on the warehouse

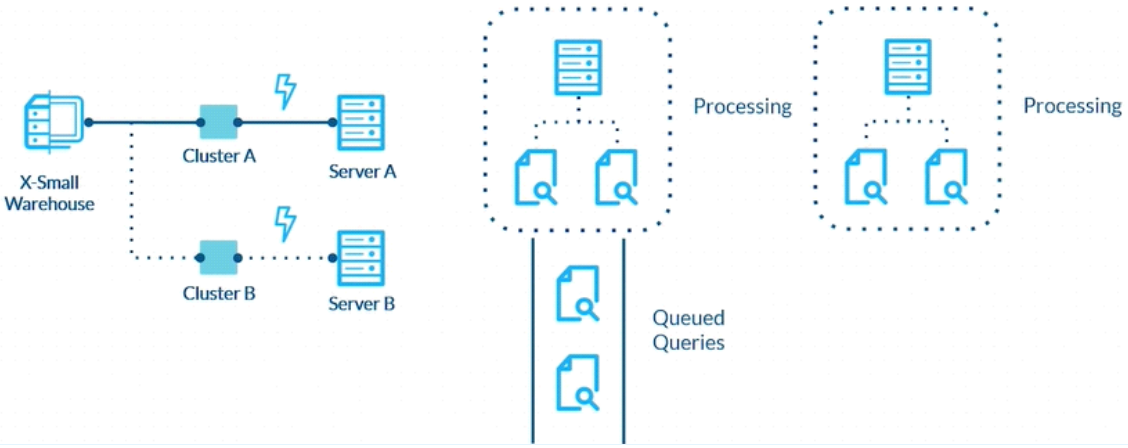


Auto-Scaling: Scaling Policy

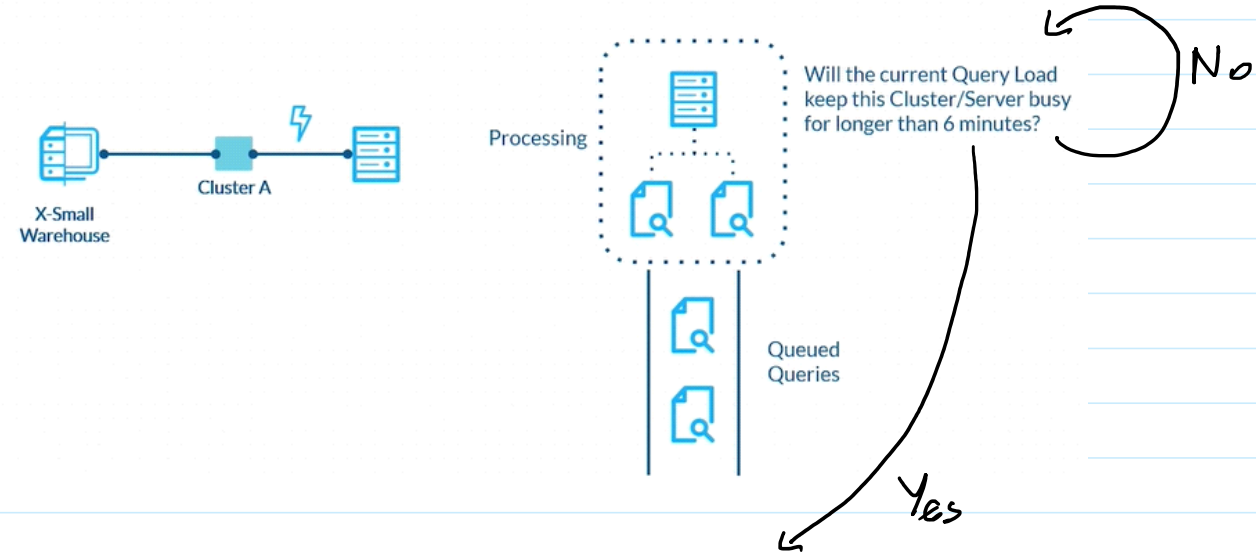
Policy	Description	Cluster Starts...	Cluster Shuts down...
Standard	Starts a new cluster as soon as queueing starts	Immediately when a query is queued or there is one more query than the currently-running clusters can execute	After 2-3 consecutive checks (at one-minute intervals) show that the load on the least-busy cluster could be redistributed to the others
Economy	Will keep queries queued up to six minutes (to see if the queue clears) before starting another cluster	Only if the system estimates there's enough query load to keep the cluster busy for at least six minutes	After 5-6 consecutive checks (at one-minute intervals), show that the load on the least-busy cluster could be redistributed to the others

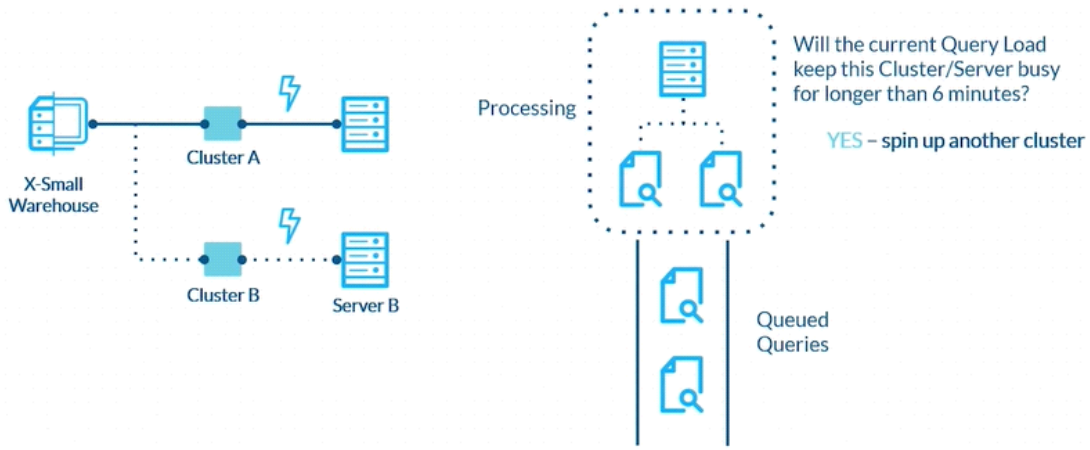
SCALING OUT POLICY: STANDARD





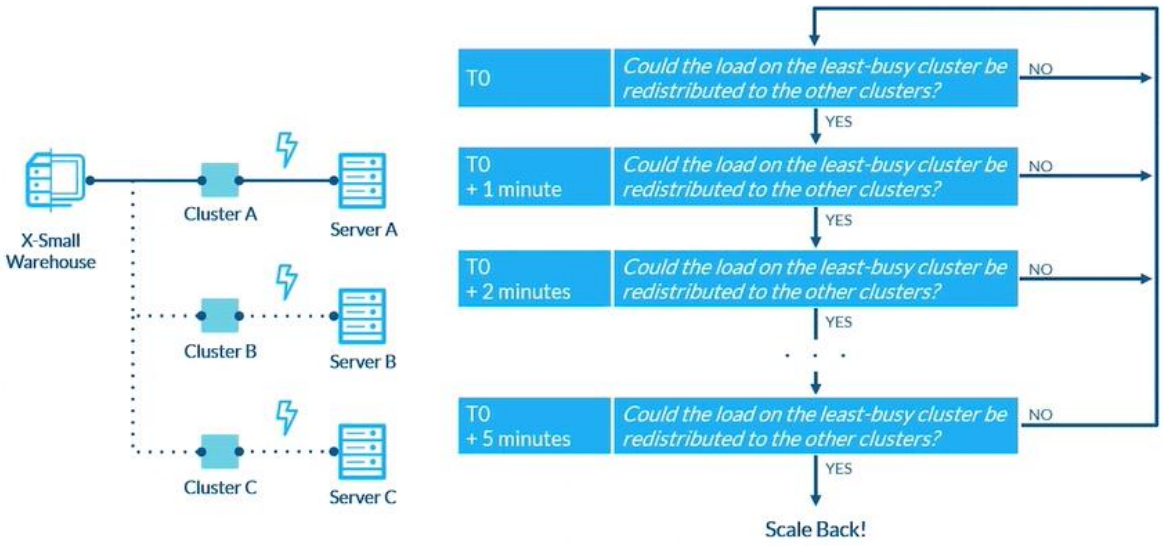
SCALING OUT POLICY: ECONOMY





Scaling Back Policy : Standard

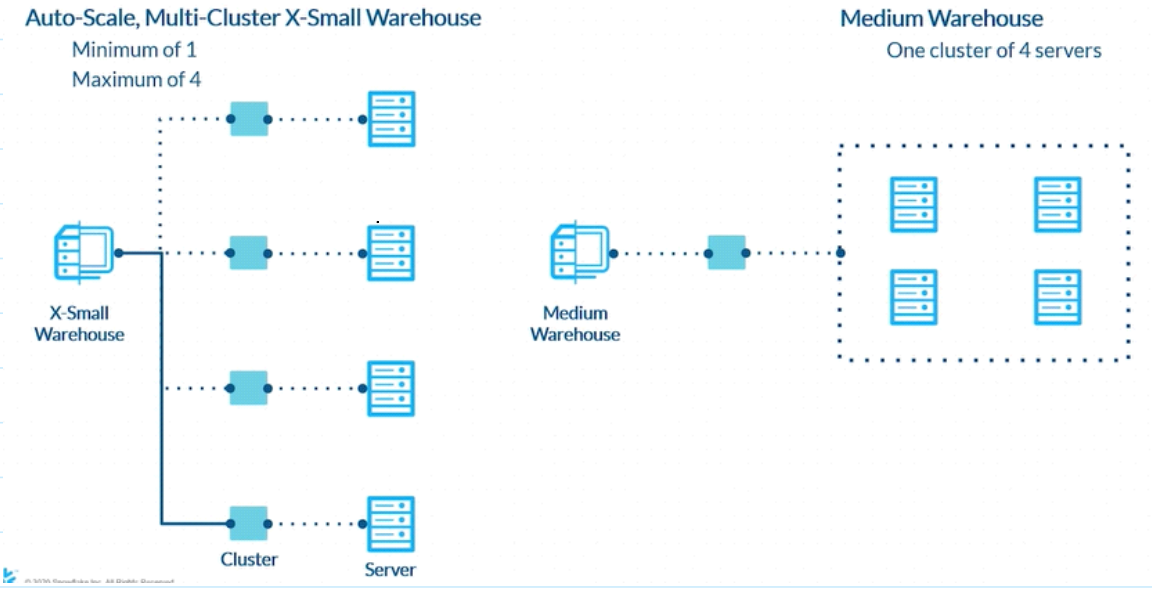
Scaling Back Policy : Economy



Standard Policy → Appropriate for a smooth usage curve

Economy → Little Erratic

ARE THESE WAREHOUSES EQUIVALENT?



NO!! Queries cannot cross clusters

Also unless it is of maximized type, charges will vary between the X-Small & Medium Warehouses as X-Small warehouse will scale back & forth

Altering Query Behavior

STATEMENT_QUEUED_TIMEOUT_IN_SECONDS

How long a queued query will wait before being cancelled by the system

Can be set for Account/Session/Role or Virtual Warehouse

Default: 0

STATEMENT_TIMEOUT_IN_SECONDS

How long a query can run before being cancelled by the system

Set for Account/Session/Role or Virtual Warehouse

Default: 2 days

Semi Structured Data Overview

07 December 2021 16:19

Native Support

→ Native Support for all your data
- ↳ Data Lakes with machine generated data
- ↳ Sensor IoT type data
- ↳ Semi-structured origin formats (JSON, AVRO, ORC, Parquet or XML)

→ First level data types
- ↳ Statistics are gathered and maintained for quicker query access and effective partition pruning for performance

→ Support for all SQL operations
- ↳ Native syntax for accessing data even in semi structured fields

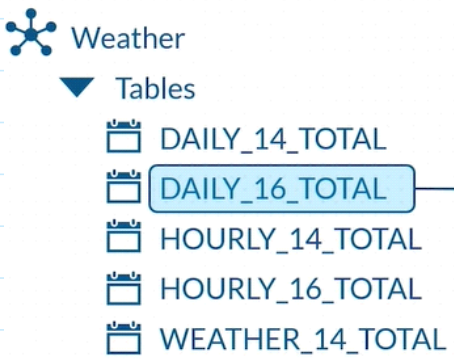
Semi-Structured Data types

→ VARIANT - holds values of standard SQL type arrays and objects
- ↳ Non-native values (eg dates & timestamps) are stored as strings in a variant column. Operations could be slower than when stored as the corresponding data type

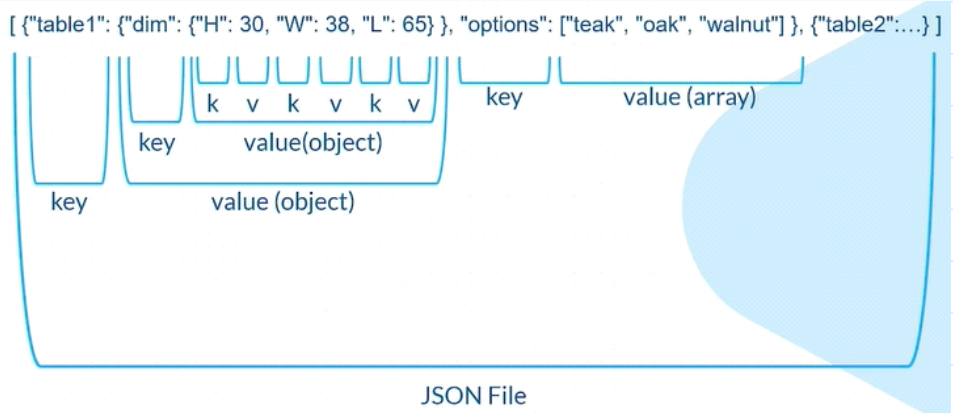
→ OBJECT - a collection of key-value pairs. The value is a VARIANT

→ ARRAY - Arrays of varying sizes. The value is a VARIANT

Sample weather data set in snowflake




```
{
  "city": {
    "coord": {
      "lat": 47.616669,
      "lon": 9.78333
    },
    "country": "DE",
    "id": 2906110,
    "name": "Hergensweiler"
  },
  "data": [
    {
      "clouds": 0,
      "deg": 132,
      "dt": 1473246000,
      "humidity": 78,
      "pressure": 959.79,
      "speed": 0.91,
      "temp": {
        "day": 295.03,
        "eve": 290.67,
        "max": 295.94,
```

Other semi-structured file formats can be loaded into a VARIANT column and queried, using similar commands and functions

Avro: Open-source framework originally developed for use with Apache Hadoop

ORC (Optimized Row Columnar): binary format used to store Hive data

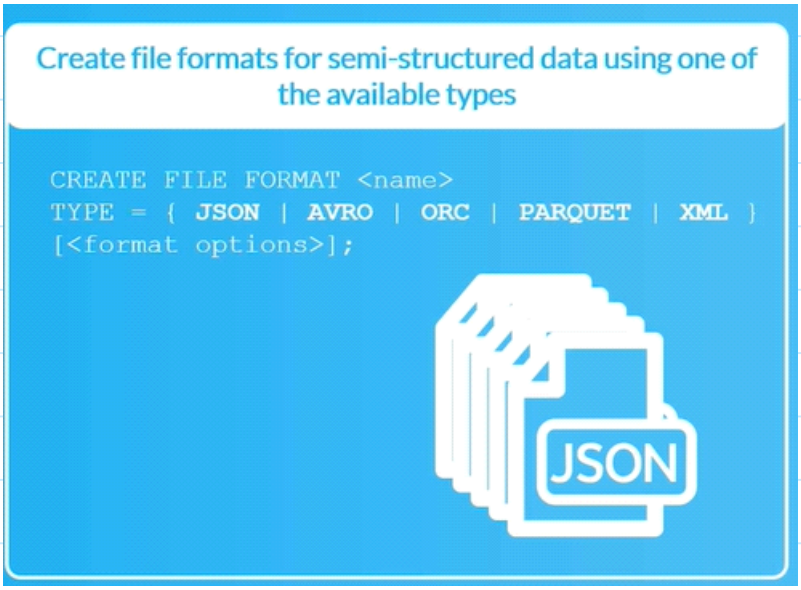
Parquet: binary format designed for projects in the Hadoop ecosystem

XML (Extensible Markup Language): consists primarily of tags <> and elements

Load and Unload Semi-Structured Data

07 December 2021 17:39

Load Semi-Structured Data



Option	Default	Details
COMPRESSION	Load: Auto Unload: GZIP	Supported algorithms: GZIP, BZ2, BROTLI, ZSTD, DEFLATE, RAW_DEFLATE, NONE. If BROTLI, cannot use AUTO.
FILE_EXTENSION	.JSON	Only used for unloading. Specifies file extension to use
ALLOW_DUPLICATE	FALSE	Only used for loading. If TRUE, allows duplicate object field names (only the last one will be preserved)
STRIP_OUTER_ARRAY	FALSE	Only used for loading. If TRUE, JSON parser will remove outer brackets [ ]
STRIP_NULL_VALUES	FALSE	Only used for loading. If TRUE, JSON parser will remove object fields or array elements containing NULL
DATE_FORMAT	AUTO	Used only for loading JSON data into separate columns. Defines the format of date string values in the data files.
TIME_FORMAT	AUTO	Used only for loading JSON data into separate columns. Defines the format on time string values in the data files.

Copy the file in with no parsing or transformation:

1. Create a table with a single VARIANT column
2. COPY INTO the table

```
COPY INTO <table>
FROM <file(s) in stage>
FILE_FORMAT = (FORMAT_NAME = <json format with options>);
```

Transform into columns and load:

1. Create a table with required column names and data types
2. COPY INTO the table with parsing and transformations

```
COPY INTO <table>
FROM(SELECT <keys> FROM <file(s) in stage>)
FILE_FORMAT = (FORMAT_NAME = <json format with options>);
```

Query Semi-Structured Data

07 December 2021 16:40

Convert JSON file to tabular format

```
SELECT
        value:temp:min::number(10,2) AS min,
        value:temp:max::number(10,2) AS max
FROM
        daily_14_total_dt,
        LATERAL FLATTEN(input=> dt.v:data)
WHERE
        dt.t = (SELECT MAX(t) FROM daily_14_total)
ORDER BY min
LIMIT 10;
```

Data Functions

→ Array creation & manipulation : eg : ARRAY_AGG

→ Object creation & manipulation : eg : OBJECT_AGG

→ JSON & XML Parsing : eg : PARSE_JSON

→ Conversion : eg : TO_ARRAY

→ Extraction : eg : FLATTEN

→ Type Checking : eg : IS_<object_type>

→ Casting : eg : AS_<object_type>

Access Values

Access a value in a VARIANT column

Colon - column:key | Brackets - column['key']

```
SELECT value:humidity AS pct_humidity
FROM my_json_table
```

```
SELECT value['humidity'] AS pct_humidity
FROM my_json_table
```

Casting Data From Variants

→ VARIANTS are often just strings, arrays or numbers

→ Without CASTing they remain VARIANT data types

→ Cast them to SQL data types using the :: operator

```
SELECT
        value:humidity::number(10,2) AS pct_humidity
FROM daily_14_total_dt,
        LATERAL FLATTEN(input=>dt.v:data)
LIMIT 3;
```

Flattening data

→ VARIANTs may contain nested elements (arrays & objects that contain the data

→ FLATTEN() extracts data from nested elements

→ Almost always used with a LATERAL join (to refer to columns from other tables, views or table function)

```
LATERAL FLATTEN
(input ⇒ expression[options])
```

→ input ⇒ The expression or column that will be unseated into rows
- the data must be of type VARIANT, OBJECT or ARRAY

Columns Returned by Flatten

SEQ	Unique sequence number associated with input record; not guaranteed to be gap-free or ordered any particular way
KEY	For maps or objects, this column contains the key to the exploded value
PATH	Path to the element within a data structure that needs to be flattened
INDEX	Index of the element, if it is an array; otherwise NULL
VALUE	Value of the element of the flattened array or object
THIS	The element being flattened (useful in recursive flattening)

```
SELECT * FROM
TABLE(FLATTEN(input=>PARSE_JSON('[1,2,3]')));
```

What position

SEQ	KEY	PATH	INDEX	VALUE	THIS
1	NULL	[0]	0	1	[1,2,3]

SEQ	KEY	PATH	INDEX	VALUE	THIS
1	NULL	[0]	0	1	[1,2,3]
1	NULL	[1]	1	2	[1,2,3]
1	NULL	[2]	2	3	[1,2,3]

No keys
Only array
of values

What is being
flattened

Demo

```
USE ROLE TRAINING_ROLE;
USE SCHEMA MY_DB.PUBLIC;
USE WAREHOUSE QUERY_WH;

SELECT *
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"
LIMIT 10;

SELECT v:time
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"
LIMIT 20;

SELECT v:time, v:city
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"
LIMIT 20;

SELECT v:city.name,
        v:city.country,
        v:city.id,
        v:city.coord
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"
LIMIT 20;

SELECT v:city.name::VARCHAR AS name,
        v:city.country::VARCHAR AS country,
        v:city.id::INTEGER AS id,
        v:city.coord.lon::NUMBER(10,2) AS longitude,
        v:city.coord.lat::NUMBER(10,2) AS latitude,
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"
LIMIT 20;

SELECT * FROM
TABLE(FLATTEN(input=>parse_json('{"a":1, "b":[77,88], "c": [1,2,3]}')));
```

Row	SEQ	KEY	PATH	INDEX	VALUE	THIS
1	1	a	a	NULL	1	{ "a": 1, "b": [77, 88], ...
2	1	b	b	NULL	[77, 88]	{ "a": 1, "b": [77, 88], ...
3	1	c	c	NULL	[1, 2, 3]	{ "a": 1, "b": [77, 88], ...

```
SELECT key, value
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"'
LATERAL FLATTEN(input=>daily_14_total.v:city) c
LIMIT 20;
```

Row	KEY	VALUE
1	city	{ "coord": { "lat": 27.716669, "lon": 80.900002 }, "country": "IN", "id": 12653...
2	data	[{ "clouds": 0, "deg": 297, "dt": 1479708000, "humidity": 84, "pressure": 10...
3	time	1479752447

```
SELECT v, c.key, c.index, c.value
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"'
LATERAL FLATTEN(input=>daily_14_total.v:city) c
LIMIT 20;
```

Row	V	KEY	INDEX	VALUE
1	{ "city": { "coord": { "lat": 27.716667...	coord	NULL	{ "lat": 27.716667, "lon": 85.316666 }
2	{ "city": { "coord": { "lat": 27.716667...	country	NULL	"NP"
3	{ "city": { "coord": { "lat": 27.716667...	id	NULL	1283240
4	{ "city": { "coord": { "lat": 27.716667...	name	NULL	"Kathmandu"
5	{ "city": { "coord": { "lat": 8.598333,...	coord	NULL	{ "lat": 8.598333, "lon": -71.144997 }
6	{ "city": { "coord": { "lat": 8.598333,...	country	NULL	"VE"

```
SELECT v, c.key, c.index, c.value
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"'
LATERAL FLATTEN(input=>daily_14_total.v:city, RECURSIVE=>TRUE) c
LIMIT 20;
```

Row	V	KEY	INDEX	VALUE
1	{ "city": { "coord": { "lat": 27.716669...	coord	NULL	{ "lat": 27.716669, "lon": 80.900002 }
2	{ "city": { "coord": { "lat": 27.716669...	lat	NULL	27.716669
3	{ "city": { "coord": { "lat": 27.716669...	lon	NULL	80.900002
4	{ "city": { "coord": { "lat": 27.716669...	country	NULL	"IN"
5	{ "city": { "coord": { "lat": 27.716669...	id	NULL	1265310
6	{ "city": { "coord": { "lat": 27.716669...	name	NULL	"Laharpur"

```
--Nested Flatten
SELECT v:city.country,
        v:city.name,
        d.value:dt,
        t.key,
        t.value
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"'
LATERAL FLATTEN(input=>daily_14_total.v:data) d,
LATERAL FLATTEN(input=>daily_14_total.v:temp) t,
LIMIT 100;
```

Row	V:CITY.COUNTRY	V:CITY.NAME	D.VALUE:DT	KEY	VALUE
1	"IN"	"Laharpur"	1479708000	day	287.45
2	"IN"	"Laharpur"	1479708000	eve	287.45
3	"IN"	"Laharpur"	1479708000	max	287.45
4	"IN"	"Laharpur"	1479708000	min	284.72
5	"IN"	"Laharpur"	1479708000	morn	287.45
6	"IN"	"Laharpur"	1479708000	night	284.72

```
CREATE TEMPORARY TABLE myjson(country, city, date, time_of_day, temp) AS
(
SELECT v:city.country::VARCHAR,
        v:city.name::VARCHAR,
        TO_DATE(d.value:dt::TIMESTAMP),
        t.key,
        t.value
FROM "SNOWFLAKE_SAMPLE_DATA"."WEATHER"."DAILY_14_TOTAL"'
LATERAL FLATTEN(input=>daily_14_total.v:data) d,
LATERAL FLATTEN(input=>daily_14_total.v:temp) t,
LIMIT 100;
)
```

```
SELECT *
FROM myjson;
```

Query History

20 December 2021 12:40

- Available from the ribbon item "History"
- Allows viewing of 14 days of query history
- Can filter by user, warehouse, session and many other attributes
- Can download previous result sets (within 24 hours)
- Can view other people's queries → SQL Queries + Metadata
- **Cannot** view other people's results

Queuing (Blue) → Compilation (Green) → Execution (Orange)

Bytes Scanned column

Remote (Green) Local (Blue)

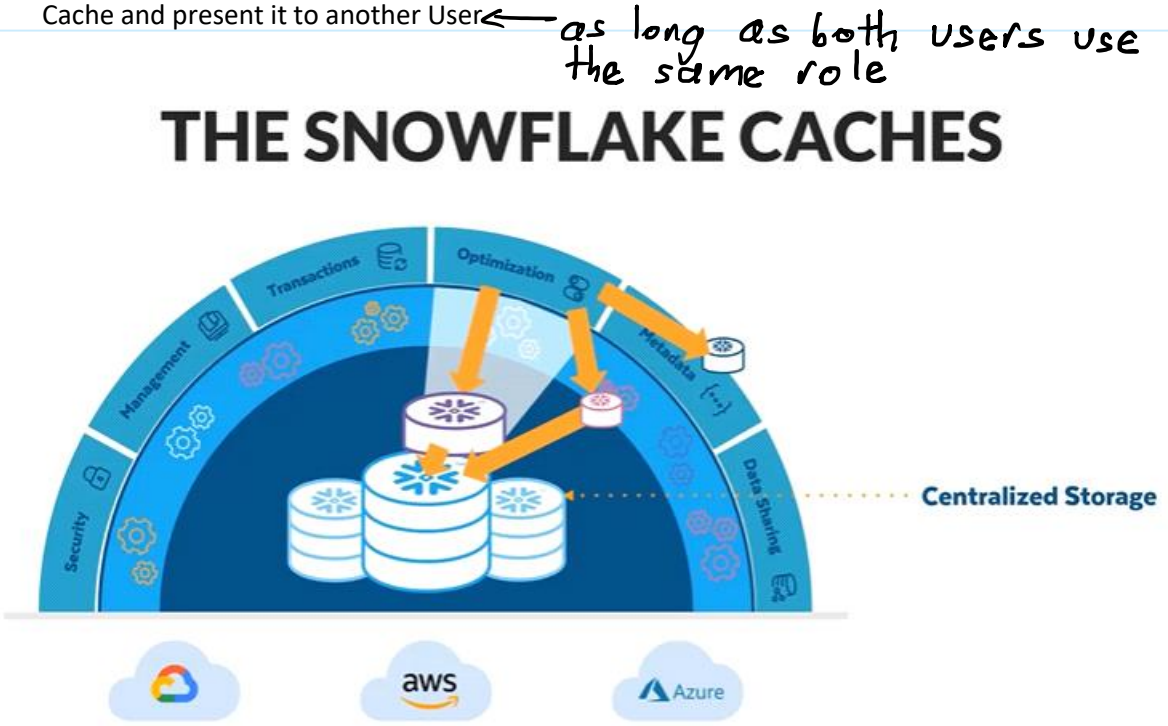
→ When same query is rerun, results of earlier run of query are cached so this time it just has to be fetched ← no compute used!!

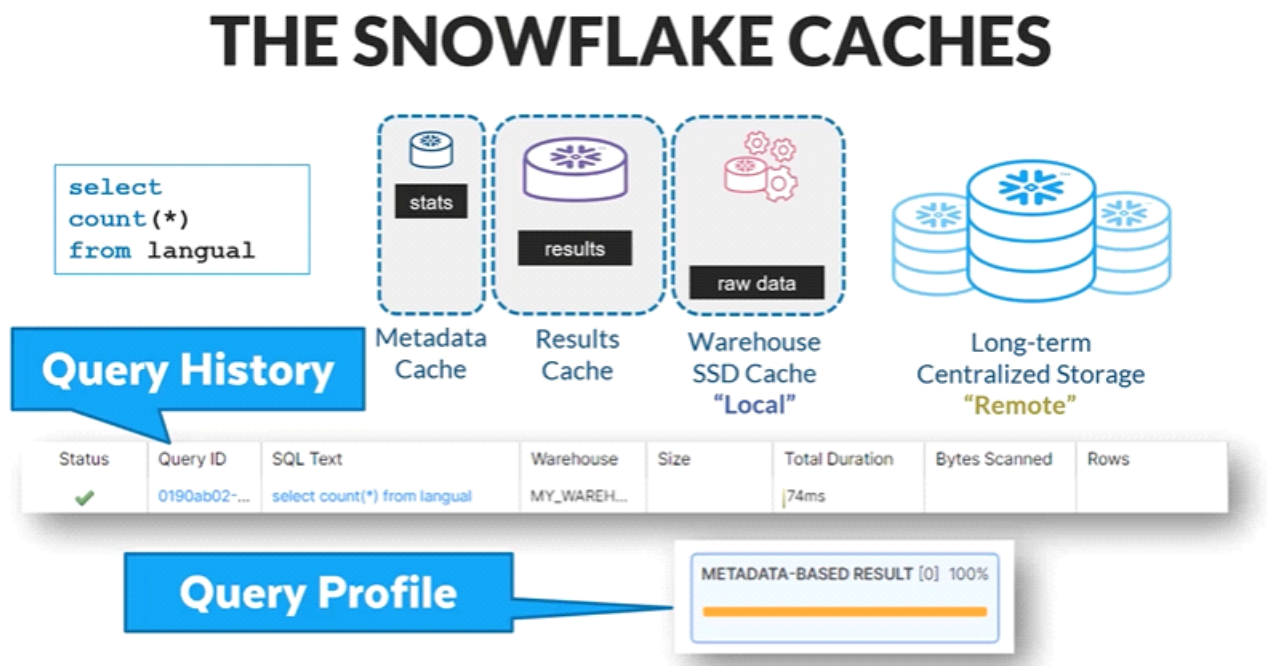
→ Can also download results from history as a csv file (for 24 hours)

Caching

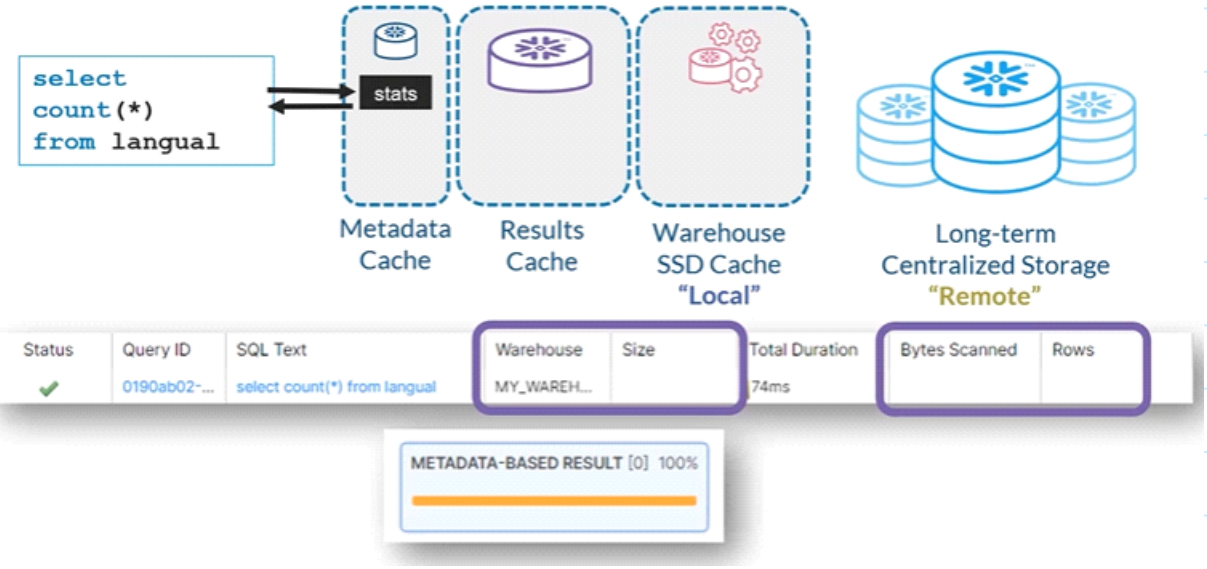
20 December 2021 13:06

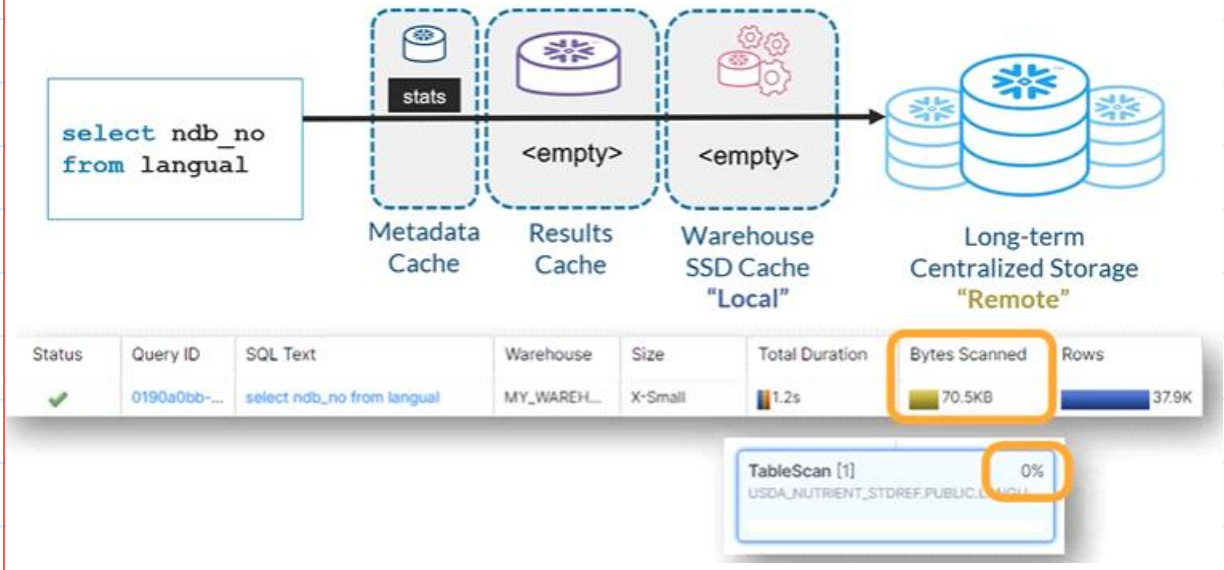
- Three types of Snowflake caches exist
- Caches can be leveraged to save money and speed results
- Metadata Cache - holds object information and statistics
- Results Cache - holds 24 hours of results
- Warehouse Cache - holds data 'local' as long as a warehouse is running
- The use of caches can be viewed in History area of WebUI
- Users **can't** see each other's results but the Results Cache **can** reuse one user's Results Cache and present it to another User ← as long as both users use the same role

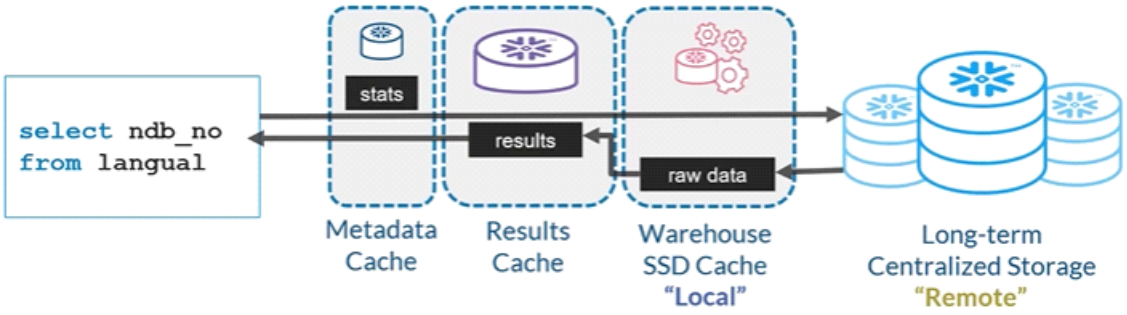




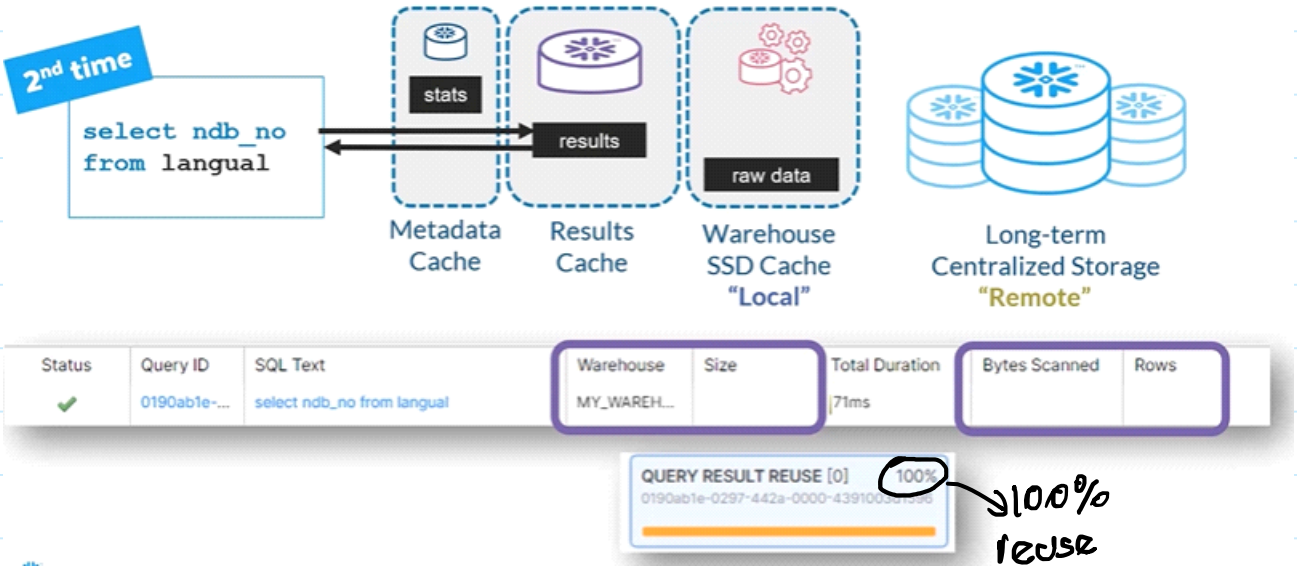
THE SNOWFLAKE CACHES

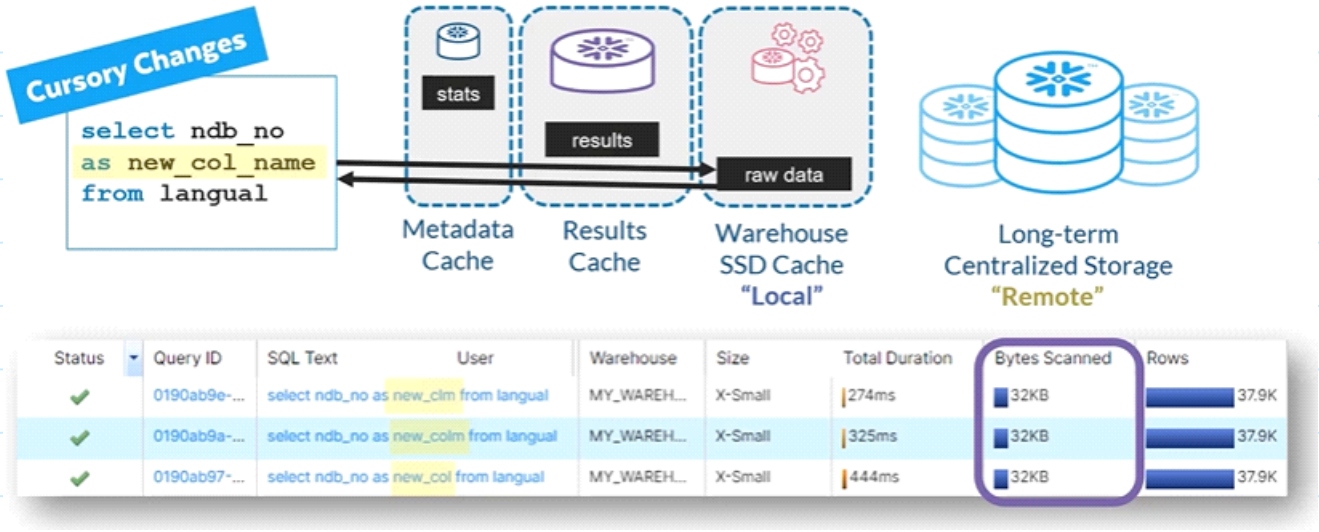


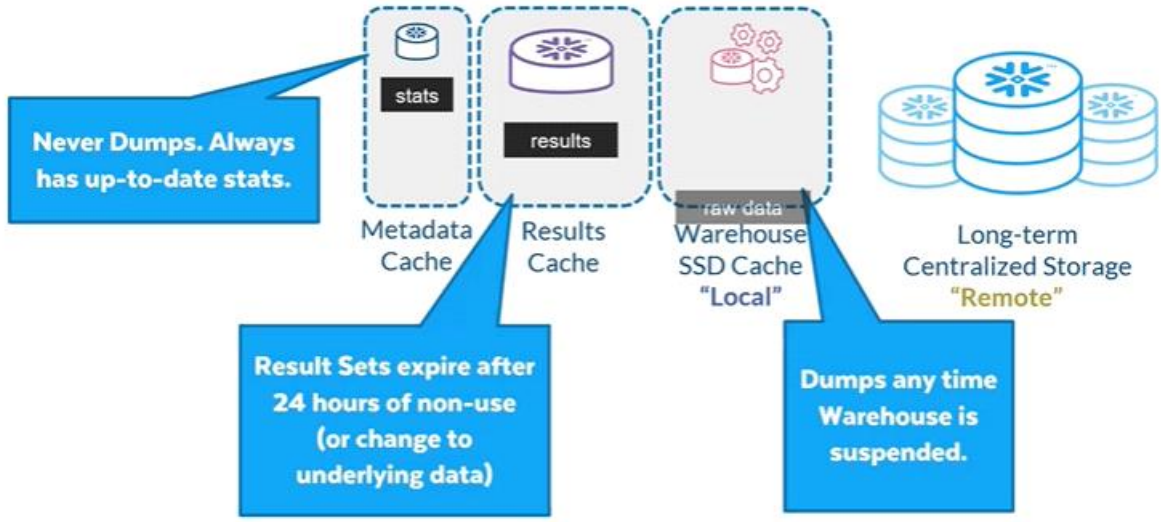


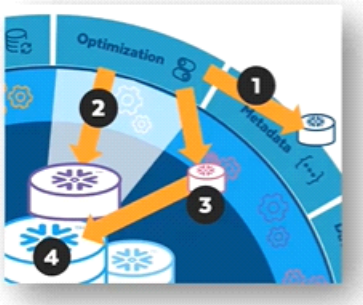


On running the query again,


2nd time
select ndb_no
from langual
stats
results
raw data
Metadata Cache
Results Cache
Warehouse SSD Cache "Local"
Long-term Centralized Storage "Remote"
Status
Query ID
SQL Text
Warehouse
Size
Total Duration
Bytes Scanned
Rows
0190ab1e-...
select ndb_no from langual
MY_WAREH...
71ms
QUERY RESULT REUSE [0]
100%
100% reuse


Cursory Changes
select ndb_no
as new_col_name
from langual
stats
results
raw data
Metadata Cache
Results Cache
Warehouse SSD Cache "Local"
Long-term Centralized Storage "Remote"
Status
Query ID
SQL Text
User
Warehouse
Size
Total Duration
Bytes Scanned
Rows
0190ab9e-...
select ndb_no as new_clm from langual
MY_WAREH...
X-Small
274ms
32KB
37.9K
0190ab9a-...
select ndb_no as new_colm from langual
MY_WAREH...
X-Small
325ms
32KB
37.9K
0190ab97-...
select ndb_no as new_col from langual
MY_WAREH...
X-Small
444ms
32KB
37.9K


Never Dumps. Always has up-to-date stats.
stats
results
raw data
Metadata Cache
Results Cache
Warehouse SSD Cache "Local"
Long-term Centralized Storage "Remote"
Result Sets expire after 24 hours of non-use (or change to underlying data)
Dumps any time Warehouse is suspended.



1 Metadata Cache
Object Definitions, Statistics

2 Results Cache
- Exact Results from Exact Queries
- Last 24 Hours
- Underlying data cannot have changed
- Functions like "CURRENT_TIME" cause expiration
- User can be different, but ROLE must be same

Do not need a warehouse

3 Warehouse Cache
- Sometimes called "Local" or "SSD" or "Data Cache"
- Contains "raw" data from table, not aggregated
- When warehouse is suspended, data is dropped
- Can use partial data and go deeper for remaining

4 Centralized Storage
- Long-term storage
- Databases, tables
- "Remote"

Context

20 December 2021 16:04

- More context parameters exist and can be explored
- Commands/Functions to set context
- Commands/Functions to query/return context
- Interesting context parameters:
  - Region Context
  - Client Context
  - Session Context
- View with History option, or Account Usage Share

Setting Context

```
USE DATABASE USDA_NUTRIENT_STDREF;
USE ROLE SYSADMIN;

SELECT CURRENT_DATABASE(),
       CURRENT_ROLE(),
       CURRENT_WAREHOUSE(),
       CURRENT_CLIENT(),  //Client ID refers to the current browser tab
       CURRENT_DATE(),
       CURRENT_REGION(),  //Depends on current cloud and its associated region.
       CURRENT_SESSION()  // Session depends on the worksheet that is being worked
       upon and also if you log out and log in it will change
       ;
```

gives name of current cloud provider as well

The query history tab has a lot more columns and also goes beyond 14 days of History provided by the history tab. We can access it via the shared SNOWFLAKE database in the ACCOUNT_USAGE schema and under the views folder we will find a QUERY_HISTORY secure view. Use this view to go beyond the History ribbon maintained history

Quiz Notes

20 December 2021 17:59

Snowpipe Designed Only To work with External stages. Not Internal Stages.
True

Cross Cloud Support Available in which of the following cloud
AWS

Data load speed in Snowflake is inversely proportional to the scale of the warehouse

True