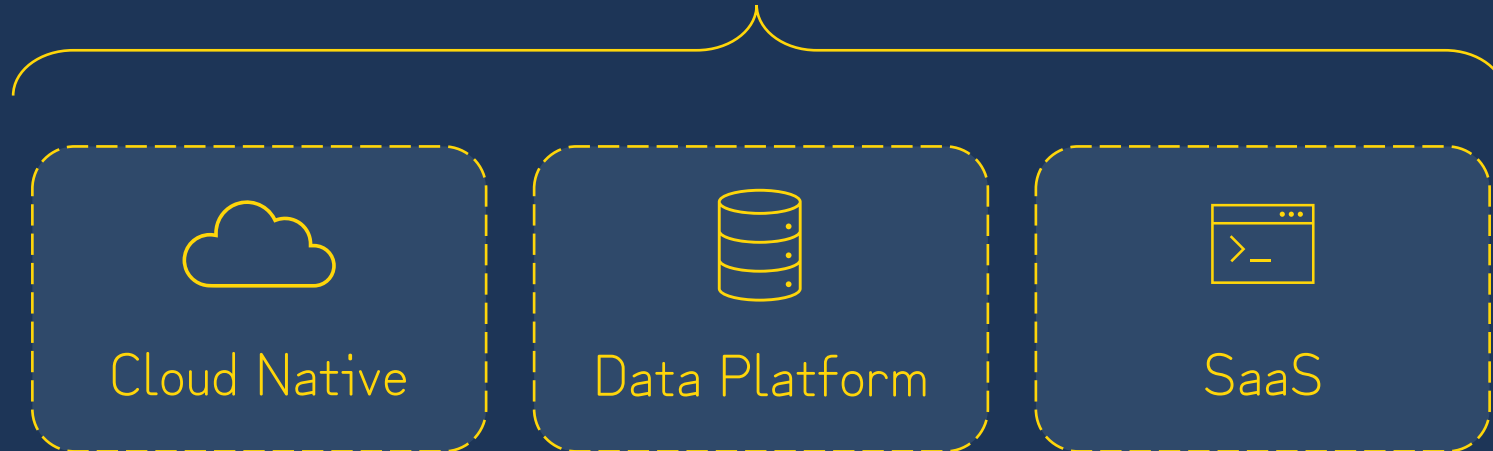


# What is Snowflake?

# What is Snowflake?



# Data Platform



## Data Warehouse

Structured & relational data

ANSI Standard SQL

ACID compliant transactions

Data stored in databases, schemas & tables



## Data Lake

Scalable storage and compute

Schema does not need to be defined upfront

Native processing of semi-structured data formats



## Data Engineering

COPY INTO & Snowpipe

Separate compute clusters

Tasks and Streams

All data encrypted at rest and in transit.



## Data Science

Remove data management roadblocks with centralised storage

Partner eco-system includes data science tooling:

- Amazon SageMaker
- DataRobot
- Dataiku



## Data Sharing

Secure Data Sharing

Data Marketplace

Data Exchange

BI with the Snowflake partner ecosystem tools



## Data Applications

Connectors and Drivers

UDFs and Stored Procedures

External UDFs

Preview features such as Snowpark

# Cloud Native



Snowflake's software is purpose built for the Cloud.

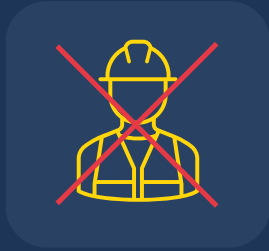


All Snowflake infrastructure runs on the Cloud in either AWS, GCP or Azure.



Snowflake makes use of Cloud's elasticity, scalability, high availability, cost-efficiency & durability.

# Software as a service (SaaS)



No management of  
hardware



Transparent updates  
and patches



Subscription payment  
model



Ease of access

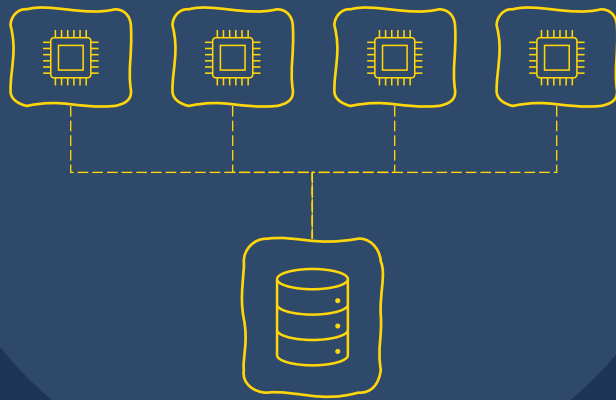


Automatic optimisation

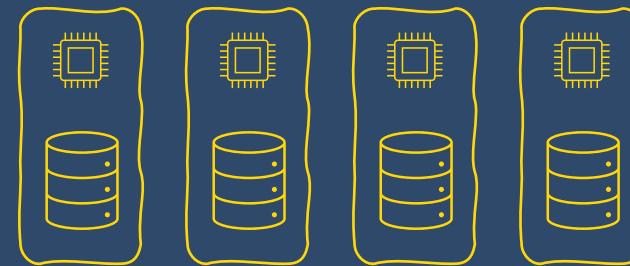
# Multi-cluster Shared Data Architecture

# Distributed Architectures

Shared-Disk

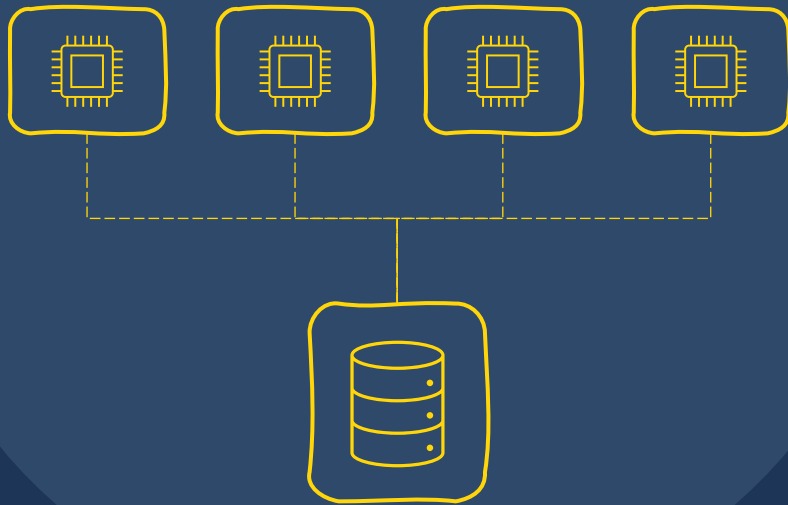


Shared Nothing



# Shared Disk Architecture

Shared-Disk



## Advantages

- Relatively simple to manage
- Single source of truth

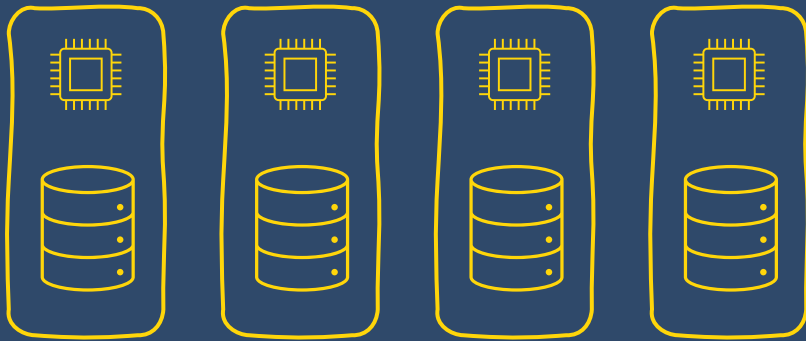
## Disadvantages

- Single point of failure
- Bandwidth and network latency
- Limited scalability



# Shared Nothing Architecture

## Shared Nothing



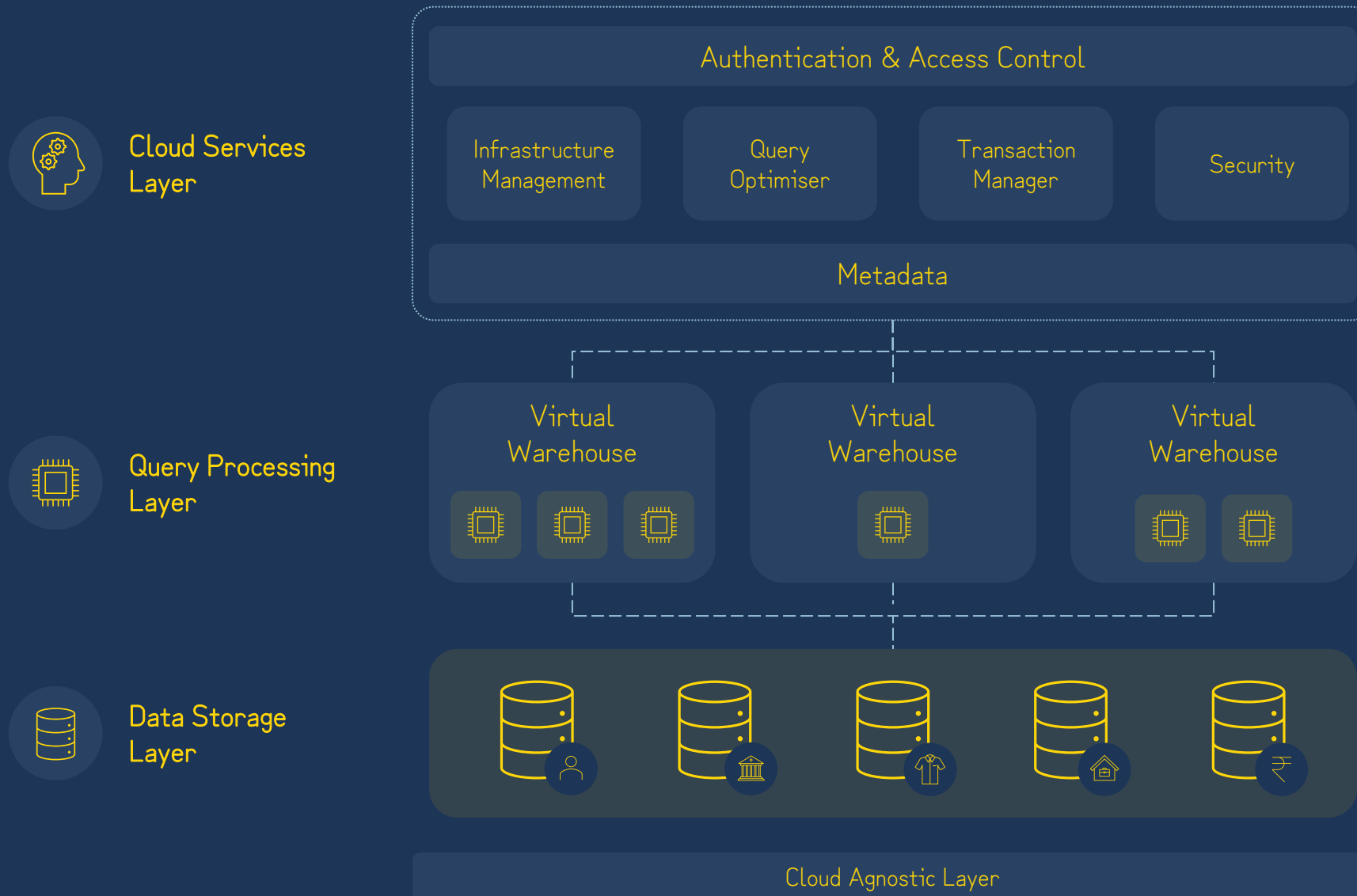
### Advantages

- Co-locating compute and storage avoids networking latency issues
- Generally cheaper to build & maintain
- Improved scaling over shared-disk architecture

### Disadvantages

- Scaling still limited
- Storage and compute tightly coupled
- Tendency to overprovision

# Multi-cluster Shared Data Architecture



- ✓ Decouple storage, compute and management services.
- ✓ Three infinitely scalable layers.
- ✓ Workload isolation with virtual warehouses.

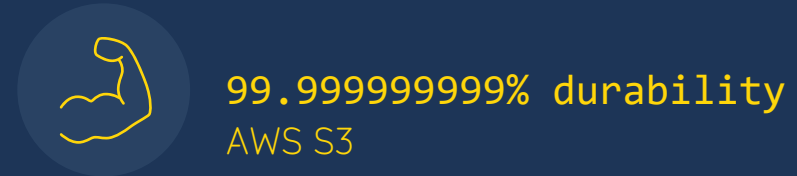
# Storage Layer

# Storage Layer

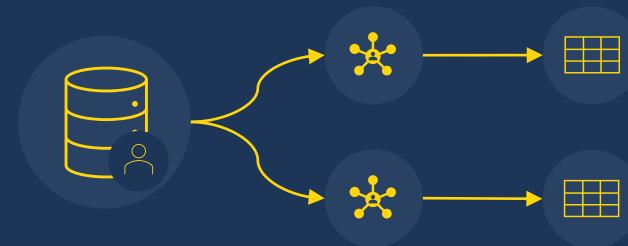
Persistent and infinitely scalable cloud storage residing in cloud providers blob storage service, such as AWS S3.



Snowflake users by proxy get the availability & durability guarantees of the cloud providers blob storage.



Data loaded into Snowflake is organized by databases, schemas and accessible primarily as tables.



Both structured and semi-structured data files can be loaded and stored in Snowflake.

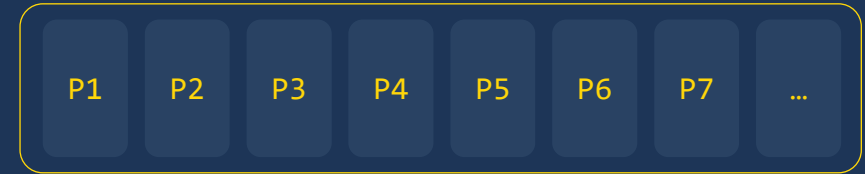


# Storage Layer

When data files are loaded or rows inserted into a table, Snowflake reorganizes the data into its proprietary compressed, columnar table file format.



The data that is loaded or inserted is also partitioned into what Snowflake call micro-partitions.



Storage is billed by how much is stored based on a flat rate per TB calculated monthly.



**\$42.00(TB/mo)**  
AWS Europe London

Data is not directly accessible in the underlying blob storage, only via SQL commands.



**SELECT \* FROM <table>;**

**TOM BAILEY COURSES**

[tombaileycourses.com](https://tombaileycourses.com)

# Query Processing Layer

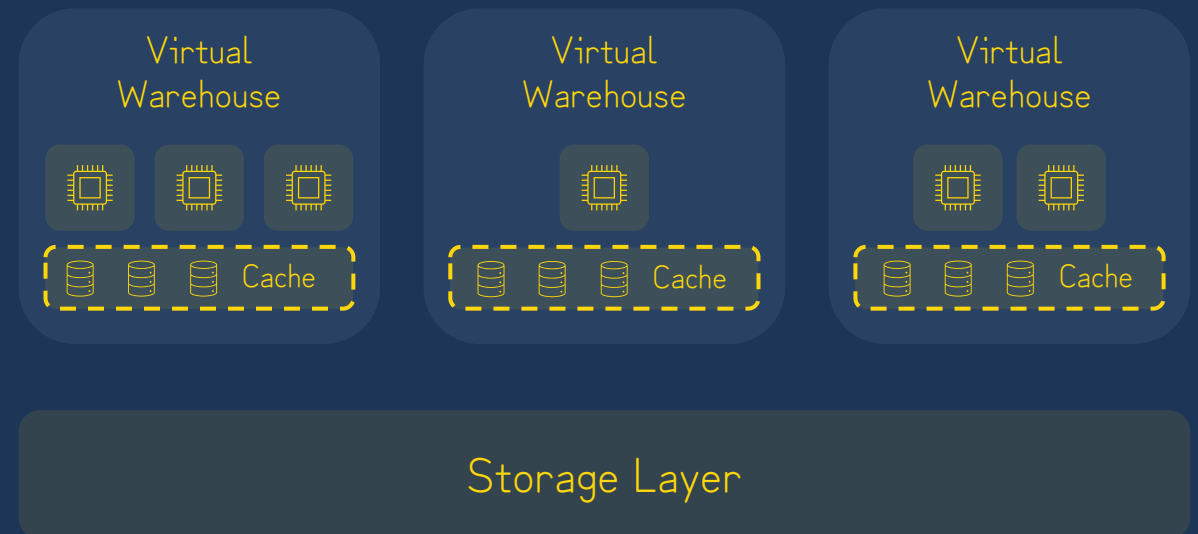
# Query Processing Layer

The query processing layer consists of “Virtual Warehouses” that execute the processing tasks required to return results for most SQL statements.

A **Virtual Warehouse** is a named abstraction for a cluster of a cloud-based compute instances that Snowflake manage.

```
CREATE WAREHOUSE MY_WH WAREHOUSE_SIZE=LARGE;
```

Underlying nodes of a Virtual Warehouse cooperate in a similar way to a shared-nothing compute clusters making use of local caching.



# Query Processing Layer

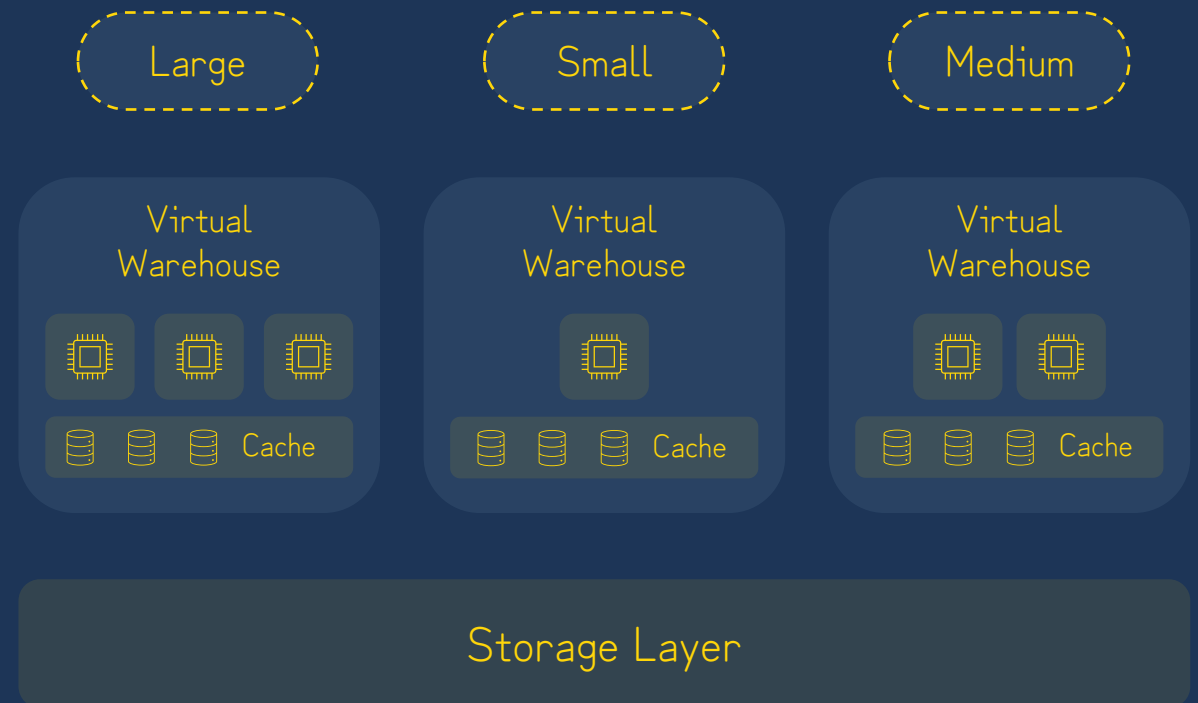
Virtual warehouses can be created or removed instantly.

Virtual warehouses can be paused or resumed.

Virtually unlimited number of virtual warehouses can be created each with it's own configuration.

Virtual warehouses come in multiple “t-shirt” sizes indicating their relative compute power.

All running virtual warehouses have consistent access to the same data in the storage layer.





# Services Layer

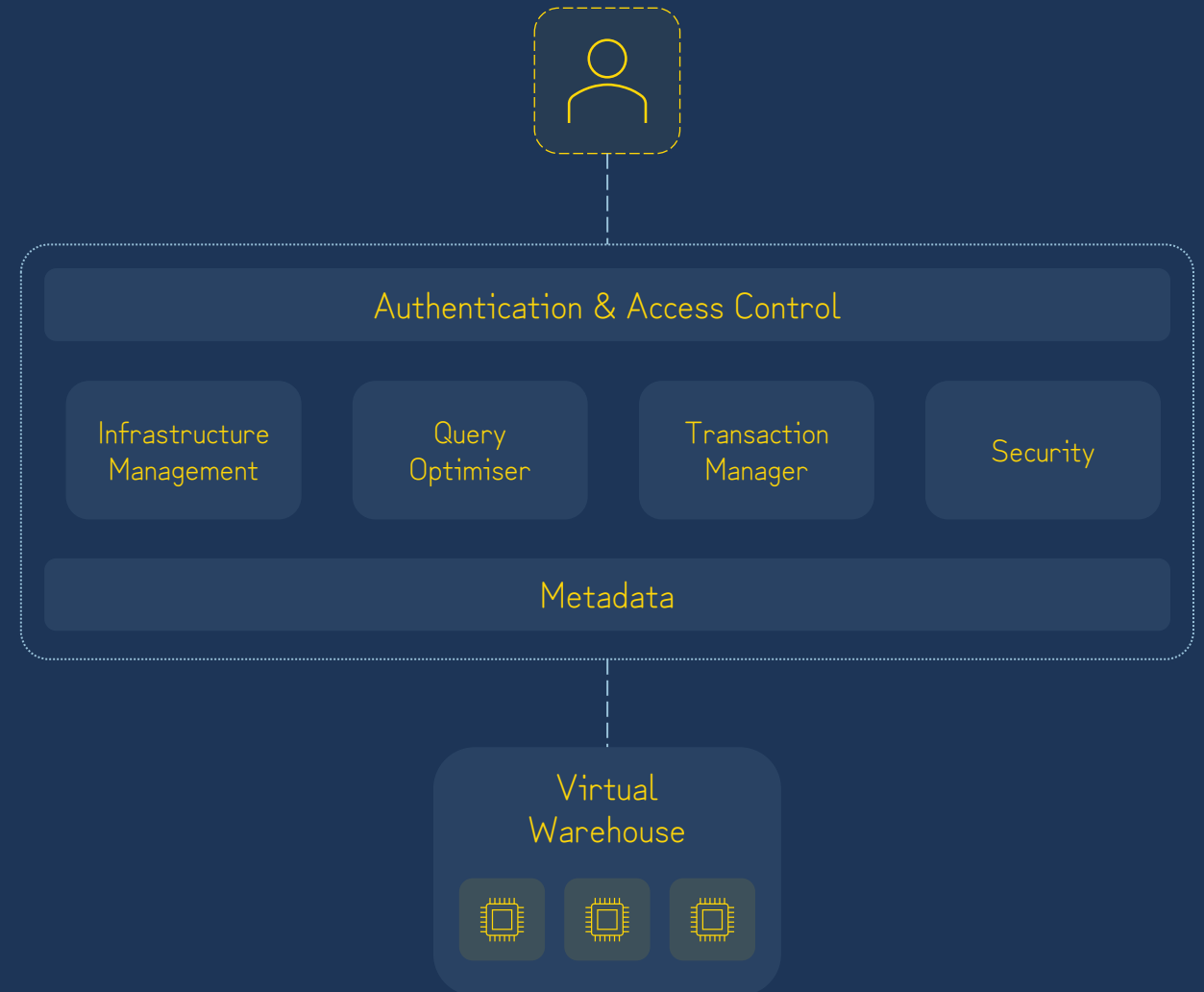
# Services Layer

The services layer is a collection of highly available and scalable services that coordinate activities such as authentication and query optimization across all Snowflake accounts.

Similar to the underlying virtual warehouse resources, the services layer also runs on cloud compute instances.

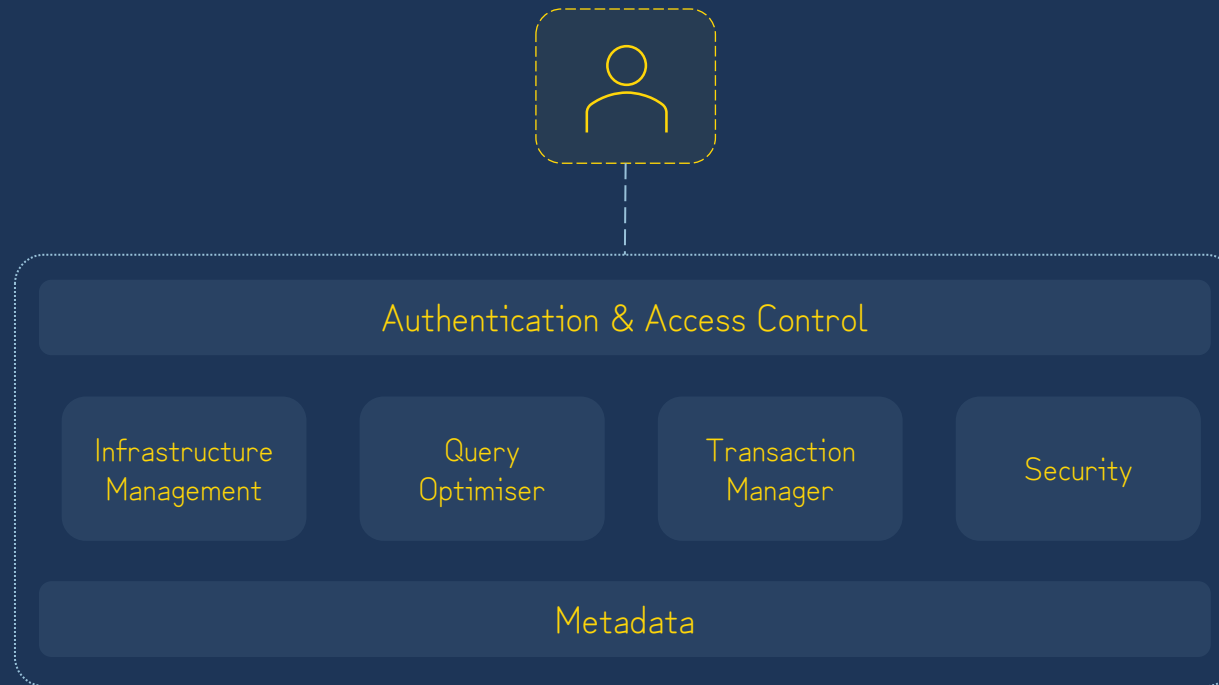
Services managed by this layer include:

- Authentication & Access Control
- Infrastructure Management
- Transaction Management
- Metadata Management
- Query parsing and optimisation
- Security



# Services Layer

The services layer is a collection of highly available and scalable services that coordinate activities such as authentication and query optimization across all Snowflake accounts.



# Editions & Key Features

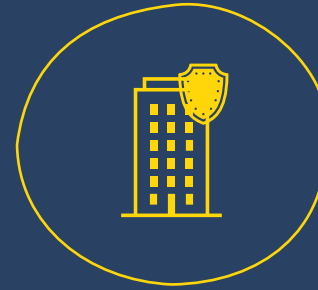
# Snowflake Editions & Key Features



Standard



Enterprise



Business Critical



Virtual Private  
Snowflake

SQL Support

Security, Governance, & Data Protection

Compute Resource Management

Interface & Tools

Releases

Data Import & Export

Data Replication & Failover

# SQL Support



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Standard SQL defined in SQL:1999



⇒ Advanced DML defined in SQL:2003



⇒ Standard data types: VARCHAR, NUMBER, TIMESTAMP etc



⇒ Semi-structured data types: VARIANT, OBJECT & ARRAY.



⇒ Multi-statement transactions



**TOM BAILEY COURSES**

[tombaileycourses.com](http://tombaileycourses.com)

# SQL Support



Feature



Standard



Enterprise



Business Critical



VPS

⇒ User Defined Functions (UDFs)



⇒ Automatic Clustering



⇒ Zero-copy Cloning



⇒ Search Optimization Service



⇒ Materialized Views



# Security, Governance & Data Protection



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Federated authentication and SSO



⇒ Multi-factor authentication (MFA)



⇒ Time Travel & Fail-safe



⇒ Encryption at-rest and in-transit



⇒ Network Policies



⇒ Access Control Framework



Continuous  
Data  
Protection

**TOM BAILEY COURSES**

[tombaileycourses.com](https://tombaileycourses.com)



# Security, Governance & Data Protection



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Column and row access policies



⇒ Tri-secret secure



⇒ Private Connectivity



⇒ Support for compliance regulations: PCI DSS,  
HIPAA, HITRUST CSF, IRAP, FedRAMP



⇒ Dedicated metastore and pool of compute  
resources



# Compute Resource Management



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Separate compute clusters (Virtual Warehouses)



⇒ Resource Monitors



⇒ Multi-cluster Virtual Warehouses



# Interface & Tools



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Classic UI & Snowsight UI



⇒ SnowSQL & SnowCD



⇒ Native programming interfaces



⇒ Third-party tools ecosystem



⇒ Snowflake Partner Connect



# Data Import & Export



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Bulk Loading



⇒ Bulk Unloading



⇒ Continuous Data Loading with Snowpipe



⇒ Snowflake Connector for Kafka



# Data Replication & Failover



Feature



Standard



Enterprise



Business Critical



VPS

⇒ Database replication

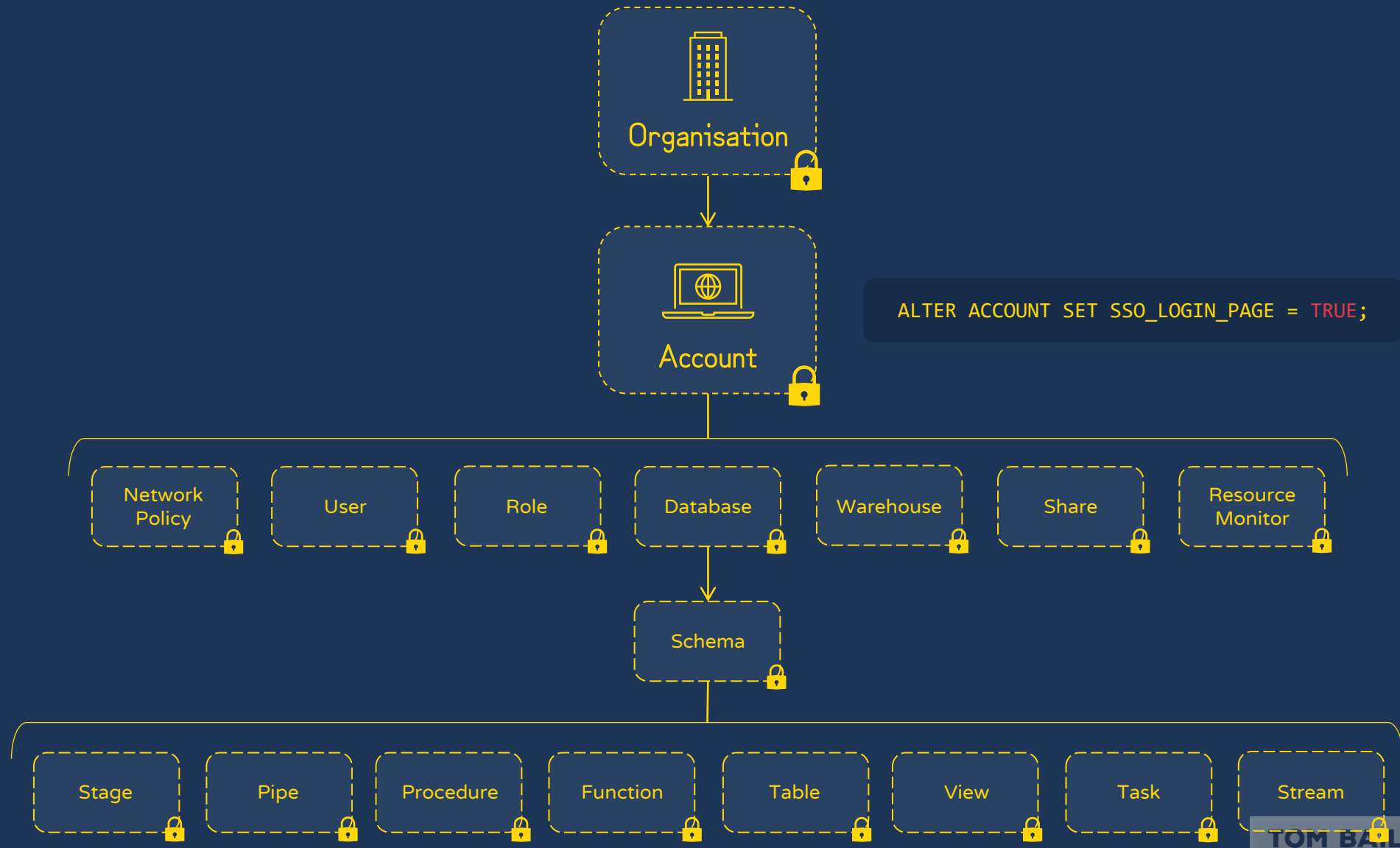


⇒ Database failover and failback



# Snowflake's Catalogue and Objects

# Snowflake Object Model



# Organisation, Account, Database & Schema.



# Organisation Overview



1

Manage one or more Snowflake accounts.

2

Setup and administer Snowflake features which make use of multiple accounts.

3

Monitoring usage across accounts.

# Organisation Setup



Contact Snowflake support



Provide organisation name and nominate an account



ORGADMIN role added to nominated account

**Organization** Last refreshed 6:02:41 PM

**Accounts**

+ Create New Account

Search Accounts 45 Accounts Columns ▾

Account Name	Edition	Snowflake Region	↓ Date Created	Account URL
TEST_AZURE	Standard	AZURE_SOUTHEASTASIA	6:02:36 PM	<a href="https://sfpmdemo-test_azure.snowflakecomputing.com">https://sfpmdemo-test_azure.snowflakecomputing.com</a>
TEST_GCP	Enterprise	GCP_EUROPE_WEST2	6:01:32 PM	<a href="https://sfpmdemo-test_gcp.snowflakecomputing.com">https://sfpmdemo-test_gcp.snowflakecomputing.com</a>
TEST_AWS	Enterprise	AWS_US_EAST_1	6:00:34 PM	<a href="https://sfpmdemo-test_aws.snowflakecomputing.com">https://sfpmdemo-test_aws.snowflakecomputing.com</a>
PROD_SECONDARY	Business Critical	AWS_US_WEST_2	5:58:44 PM	<a href="https://sfpmdemo-prod_secondary.snowflakecomputing.com">https://sfpmdemo-prod_secondary.snowflakecomputing.com</a>
PROD_PRIMARY	Business Critical	AWS_US_EAST_1	5:56:54 PM	<a href="https://sfpmdemo-prod_primary.snowflakecomputing.com">https://sfpmdemo-prod_primary.snowflakecomputing.com</a>
RAMEN	Standard	AZURE_EASTUS2	1/8/2021, 4:20:53 PM	<a href="https://sfpmdemo-ramen.snowflakecomputing.com">https://sfpmdemo-ramen.snowflakecomputing.com</a>

# ORGADMIN Role

ORGADMIN

Account Management

```
CREATE ACCOUNT MYACCOUNT1
  ADMIN_NAME = admin
  ADMIN_PASSWORD = 'Password123'
  FIRST_NAME = jane
  LAST_NAME = smith
  EMAIL = 'myemail@myorg.org'
  EDITION = enterprise
  REGION = aws_us_west_2;
```

```
SHOW ORGANIZATION ACCOUNTS;
```

```
SHOW REGIONS;
```

Enable cross-account features

```
SELECT
system$global_account_set_parameter(
  'UT677AA',
  'ENABLE_ACCOUNT_DATABASE_REPLICATION',
  'true');
```

Monitoring account usage

```
SELECT
ROUND(SUM(AVERAGE_BYTES) /POWER(1024,4),2)
FROM ORGANIZATION_USAGE.STORAGE_DAILY_HISTORY
WHERE USAGE_DATE = CURRENT_DATE();
```

# Account Overview

An account is the administrative name for a collection of storage, compute and cloud services deployed and managed entirely on a selected cloud platform.

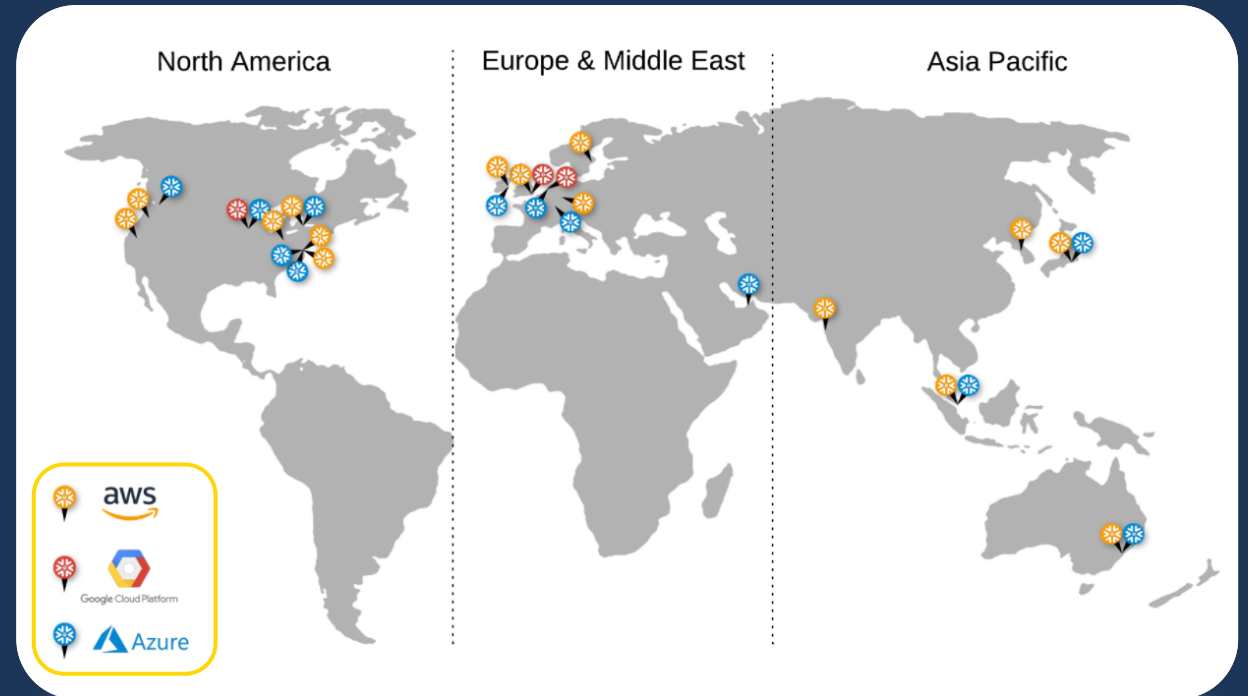
Each account is hosted on a **single cloud provider**:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure (Azure)

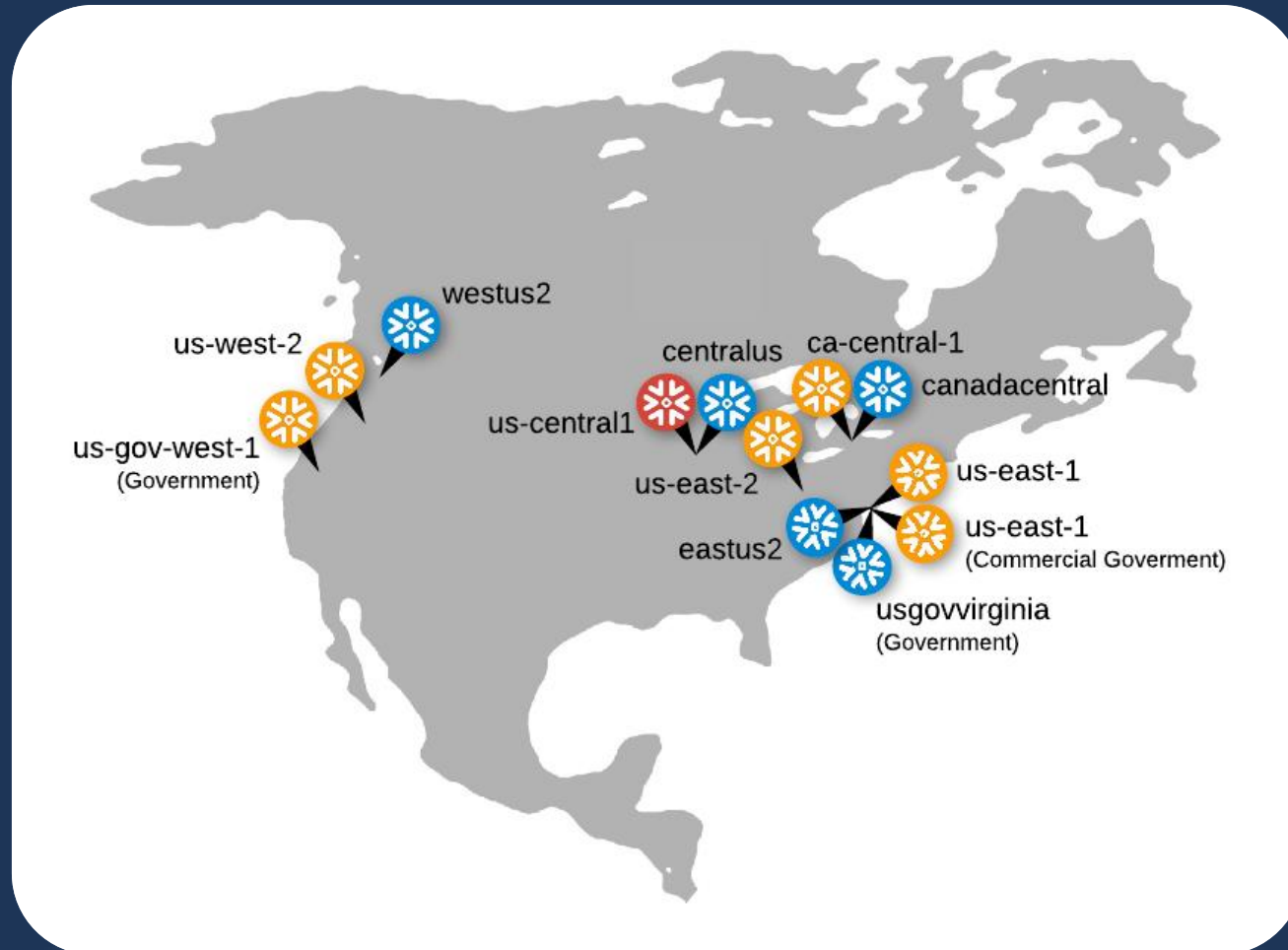
Each account is provisioned in a **single geographic region**.

Each account is created as a **single Snowflake edition**.

An account is created with the system-defined role **ACCOUNTADMIN**.



# Account Regions



aws.us-west-2

US West (Oregon)

aws.ca-central-1

Canada (Central)

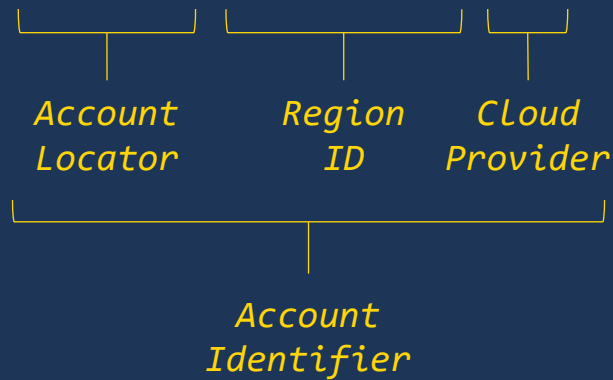
azure.westus2

West US 2 (Washington)

# Account URL

Using an Account Locator as an Identifier

xy12345.us-east-2.aws.snowflakecomputing.com

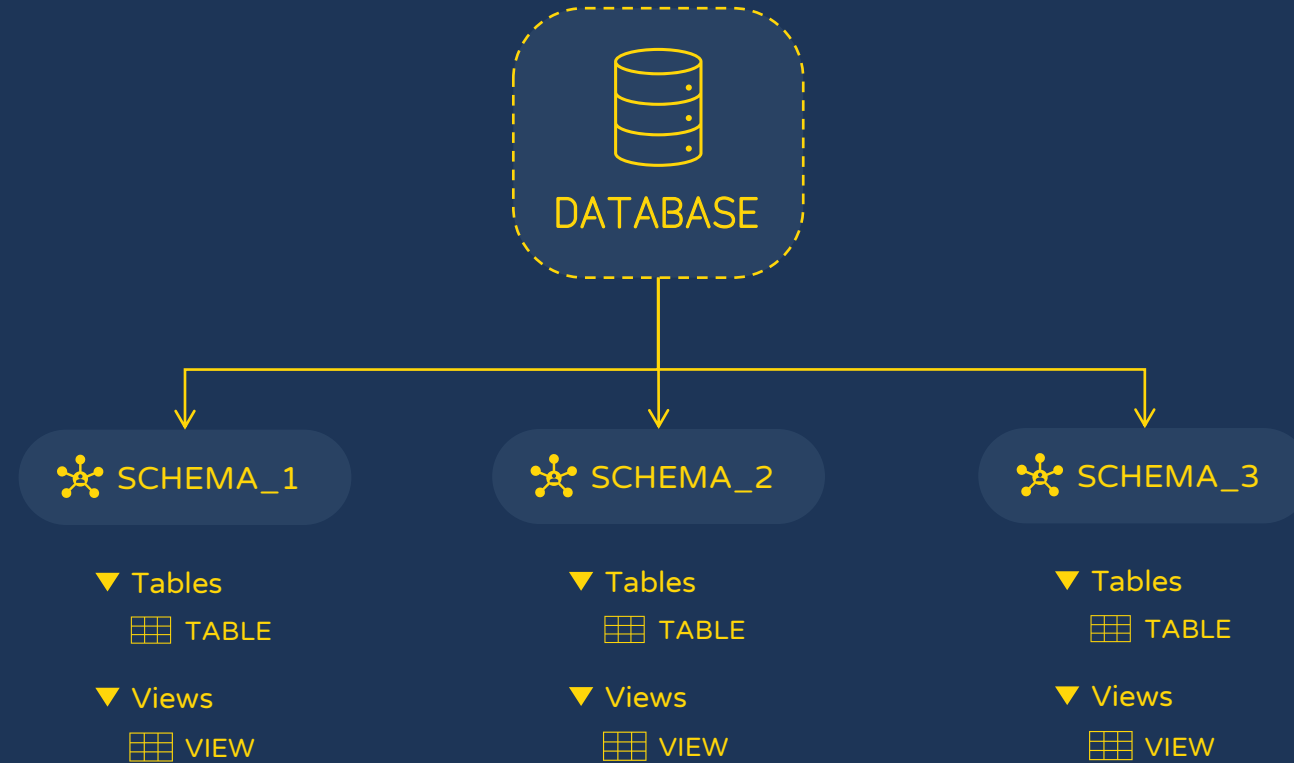


Using an Organization and Account Name as Identifier

acme-marketing-test-account.snowflakecomputing.com



# Database & Schemas



# Database & Schemas



## DATABASE

Databases must have a unique identifier in an account.

A database must start with an alphabetic character and cannot contain spaces or special characters unless enclosed in double quotes.

```
CREATE DATABASE MY_DATABASE;
```

```
CREATE DATABASE MY_DB_CLONE CLONE MYTESTDB;
```

```
CREATE DATABASE MYDB1  
  AS REPLICATION OF MYORG.ACCOUNT1.MYDB1  
  DATA_RETENTION_TIME_IN_DAYS = 10;
```

```
CREATE DATABASE SHARED_DB FROM SHARE UTT783.SHARE;
```



## SCHEMA

Schemas must have a unique identifier in a database.

A schema must start with an alphabetic character and cannot contain spaces or special characters unless enclosed in double quotes.

```
CREATE SCHEMA MY_SCHEMA;
```

```
CREATE SCHEMA MY_SCHEMA_CLONE CLONE MY_SCHEMA;
```

```
MY_DATABASE.MY_SCHEMA
```

*Namespace*



# Table and View Types

# Table Types



Permanent

Default table type.

Exists until explicitly dropped.



Temporary

Used for transitory data.

Persist for duration of a session.



Transient

Exists until explicitly dropped.

No fail-safe period.



External

Query data outside Snowflake.

Read-only table.

Time Travel

✓ 90 days

✓ 1 day

✓ 1 day



Fail-safe



# View Types



Standard

```
CREATE VIEW MY_VIEW AS  
SELECT COL1, COL2 FROM MY_TABLE;
```

Does not contribute to storage cost.

If source table is dropped, querying view returns error.

Used to restrict contents of a table.



Materialized

```
CREATE MATERIALIZED VIEW MY_VIEW AS  
SELECT COL1, COL2 FROM MY_TABLE;
```

Stores results of a query definition and periodically refreshes it.

Incurs cost as a serverless feature.

Used to boost performance of external tables.



Secure

```
CREATE SECURE VIEW MY_VIEW AS  
SELECT COL1, COL2 FROM MY_TABLE;
```

Both standard and materialized can be secure.

Underlying query definition only visible to authorized users.

Some query optimizations bypassed to improve security.

# UDFs and Stored Procedures

# User Defined Functions (UDFs)

User defined functions (UDFs) are schema-level objects that enable users to write their own functions in three different languages:

- SQL
- JavaScript
- Python
- Java

UDFs accept 0 or more parameters.

UDFs can return scalar or tabular results (UDTF).

UDFs can be called as part of a SQL statement.

UDFs can be overloaded.

```
CREATE FUNCTION AREA_OF_CIRCLE(radius FLOAT)
  RETURNS TABLE (area number)
AS
$$
  pi() * radius * radius
$$;
```

```
SELECT  AREA_OF_CIRCLE(col1) FROM MY_TABLE;
```

# JavaScript UDF

```
CREATE FUNCTION JS_FACTORIAL(d double)
  RETURNS DOUBLE
  LANGUAGE JAVASCRIPT
  AS
  $$
  if (D <= 0) {
    return 1
  } else {
    var result = 1;
    for (var i = 2; i <= D; i++) {
      result = result * i;
    }
    return result;
  }
  $$;
```

JavaScript is specified with the language parameter.

Enables use of high-level programming language features.

JavaScript UDFs can refer to themselves recursively.

Snowflake data types are mapped to JavaScript data types.

JavaScript

# Java UDF

```
CREATE FUNCTION DOUBLE(X INTEGER)
  RETURNS INTEGER
  LANGUAGE JAVA
  HANDLER='TestDoubleFunc.double'
  TARGET_PATH='@~/TestDoubleFunc.jar'
AS
$$
    class TestDoubleFunc {
        public static int double(int x) {
            return x * 2;
        }
    }
$$;
```

Java

Snowflake boots up a JVM to execute function written in Java.

Snowflake currently supports writing UDFs in Java versions 8.x, 9.x, 10.x, and 11.x.

Java UDFs can specify their definition as in-line code or a pre-compiled jar file.

Java UDFs cannot be designated as secure.

# External Functions

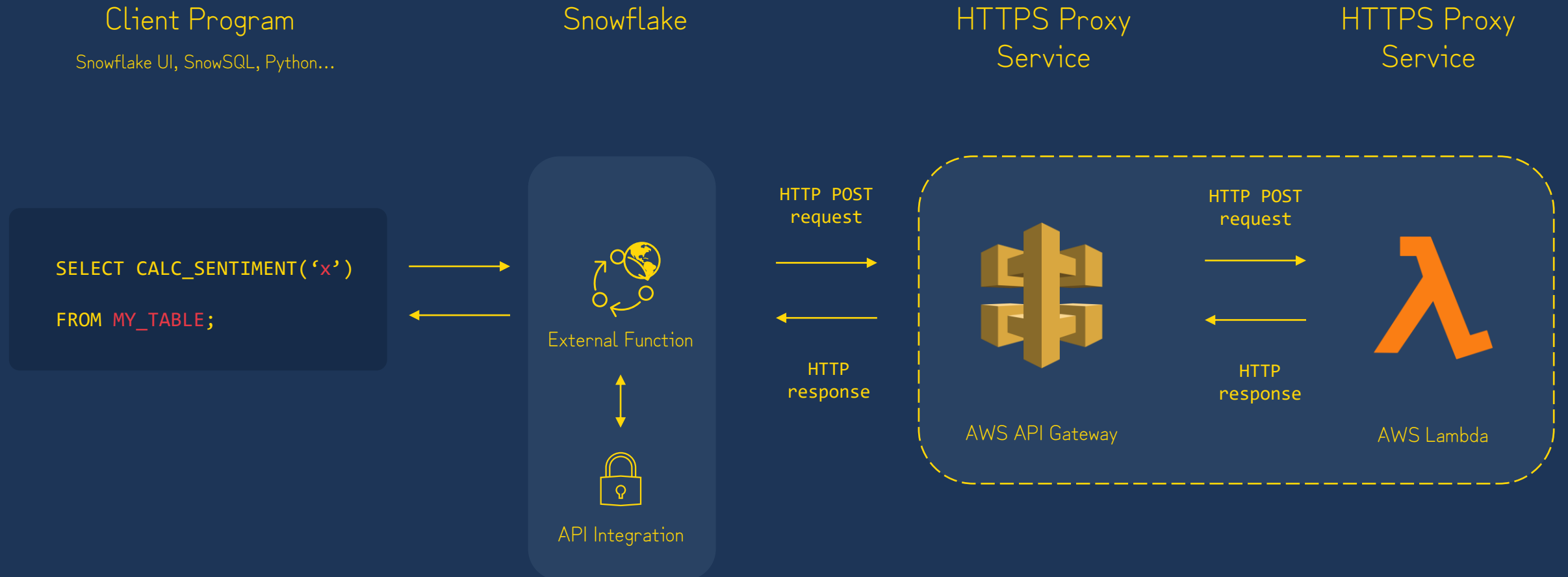
```
CREATE OR REPLACE EXTERNAL FUNCTION CALC_SENTIMENT(String_Col VARCHAR)
  RETURNS VARIANT
  API_INTEGRATION = AWS_API_INTEGRATION
  AS 'https://ttu.execute-api.eu-west-2.amazonaws.com/';
```

- ← Function Name and Parameters
- ← Return Type
- ← Integration Object
- ← URL Proxy Service

```
CREATE OR REPLACE API INTEGRATION AWS_API_INTEGRATION
  API_PROVIDER=AWS_API_GATEWAY
  API_AWS_ROLE_ARN='ARN:AWS:IAM::123456789012:ROLE/MY_CLOUD_ACCOUNT_ROLE'
  API_ALLOWED_PREFIXES=('HTTPS://XYZ.EXECUTE-API.US-WEST-2.AMAZONAWS.COM/PRODUCTION')
  ENABLED=TRUE;
```



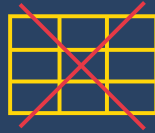
# External Function Call Lifecycle



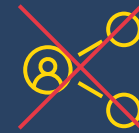
# External Function Limitations



Slower



Scalar only



Not sharable



Less secure



Egress charges

# Stored Procedures

In Relational Database Management Systems (RDBMS) **stored procedures** were **named collections of SQL statements** often containing procedural logic.



Database Admin (DBA)

```
CREATE PROCEDURE CLEAR_EMP_TABLES
DELETE FROM EMP01 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
AS
DELETE FROM EMP02 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
BEGIN
DELETE FROM EMP03 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
DELETE FROM EMP04 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
DELETE FROM EMP05 WHERE EMP_DATE < DATEADD(MONTH, -1, GET_DATE())
```

END



Data Engineer (DE)

```
EXECUTE CLEAR_EMP_TABLES;
```

 SQL examples on this slide from Microsoft SQL Server.

**TOM BAILEY COURSES**

[tombaileycourses.com](https://tombaileycourses.com)

# Snowflake Stored Procedures

①



JavaScript

②



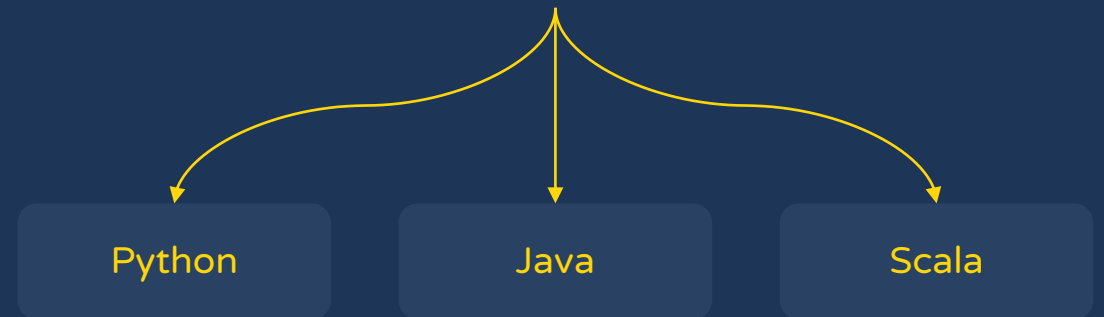
Snowflake Scripting



③



Snowpark



# Stored Procedure: JavaScript

Stored procedure identifier and input parameters. —————>

RETURNS option mandatory. —————>

JAVASCRIPT, SQL, PYTHON, JAVA & SCALA. —————>

Stored procedures can execute with the owner's rights or caller's rights. —————>

Stored procedures mix JavaScript and SQL in their definition using Snowflake's JavaScript API. —————>

```
CALL EXAMPLE_STORED_PROCEDURE('EMP01');
```

```
CREATE PROCEDURE EXAMPLE_STORED_PROCEDURE(PARAM1 STRING)

    RETURNS STRING

    LANGUAGE JAVASCRIPT

    EXECUTE AS OWNER

    AS

    $$

        var param1 = PARAM1;

        var sql_command = "SELECT * FROM " + param1;

        snowflake.execute({sqlText: sql_command});

        return "Succeeded.";

    $$;
```

# Stored Procedures & UDFs

Feature	UDF	Stored Procedure
Called as part of SQL statement	✓	✗
Ability to overload	✓	✓
0 or more input parameters	✓	✓
Use of JavaScript API	✗	✓
Return of value optional	✗	✓
Values returned usable in SQL	✓	✗
Call itself recursively	✗	✓

Functions calculate something and return a value to the user.

Stored procedures perform actions rather than return values.

# Sequences

# Sequences

```
CREATE SEQUENCE DEFAULT_SEQUENCE  
  
START = 1  
  
INCREMENT = 1;
```

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
1

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
2

```
SELECT DEFAULT_SEQUENCE.NEXTVAL;
```

NEXTVAL
3



Values generated by a sequence are globally unique.



# Sequences

```
CREATE SEQUENCE INCREMENT_SEQUENCE  
  
START = 0  
  
INCREMENT = 5;
```

```
SELECT INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL, INCREMENT_SEQUENCE.NEXTVAL;
```

NEXTVAL	NEXTVAL_1	NEXTVAL_2	NEXTVAL_3
30	40	40	50



Sequences cannot guarantee their values will be gap free.

# Sequences

① INSERT INTO TABLE.

```
CREATE SEQUENCE TRANSACTION_SEQ  
START = 1001  
INCREMENT = 1;
```

```
INSERT INTO TRANSACTION (ID)  
VALUES (TRANSACTION_SEQ.NEXTVAL)
```

```
SELECT ID FROM TRANSACTION;
```

NEXTVAL
1001

# Sequences

②

DEFAULT VALUE FOR  
A COLUMN TABLE.

```
CREATE TABLE TRANSACTIONS  
(ID INTEGER DEFAULT TRANSACTION_SEQ.NEXTVAL,  
AMOUNT DOUBLE);
```

```
INSERT INTO TRANSACTION (AMOUNT) VALUES (756.00);
```

```
SELECT ID FROM TRANSACTION;
```

ID	AMOUNT
1002	756.00

# Tasks & Streams

# Tasks & Streams



Tasks



Streams

# Tasks

A task is an object used to schedule the execution of a SQL command or a stored procedure.

## Task Workflow

① ACCOUNTNADMIN role or CREATE TASK privilege.

②

```
CREATE TASK T1
  WAREHOUSE = MYWH
  SCHEDULE = '30 MINUTE'
AS
  COPY INTO MY_TABLE
  FROM $MY_STAGE;
```

← Task Name

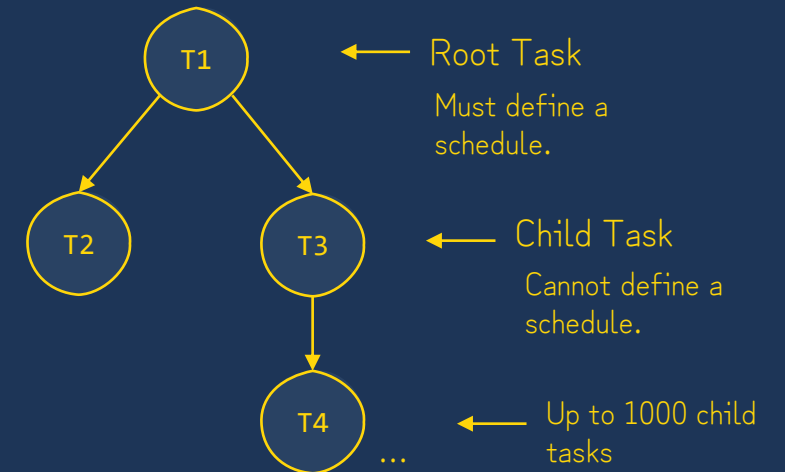
← Warehouse Definition

← Triggering Mechanism

← Query Definition

③ ALTER TASK MINUTE\_TASK RESUME; ← Start task

## Tree of Tasks



```
CREATE TASK T2
  WAREHOUSE = MYWH
  AFTER T1
AS
  COPY INTO MY_TABLE FROM $MY_STAGE;
```

← Child task triggering mechanism

# Streams

A stream is an object created to view & track DML changes to a source table – inserts, updates & deletes.

Create Stream

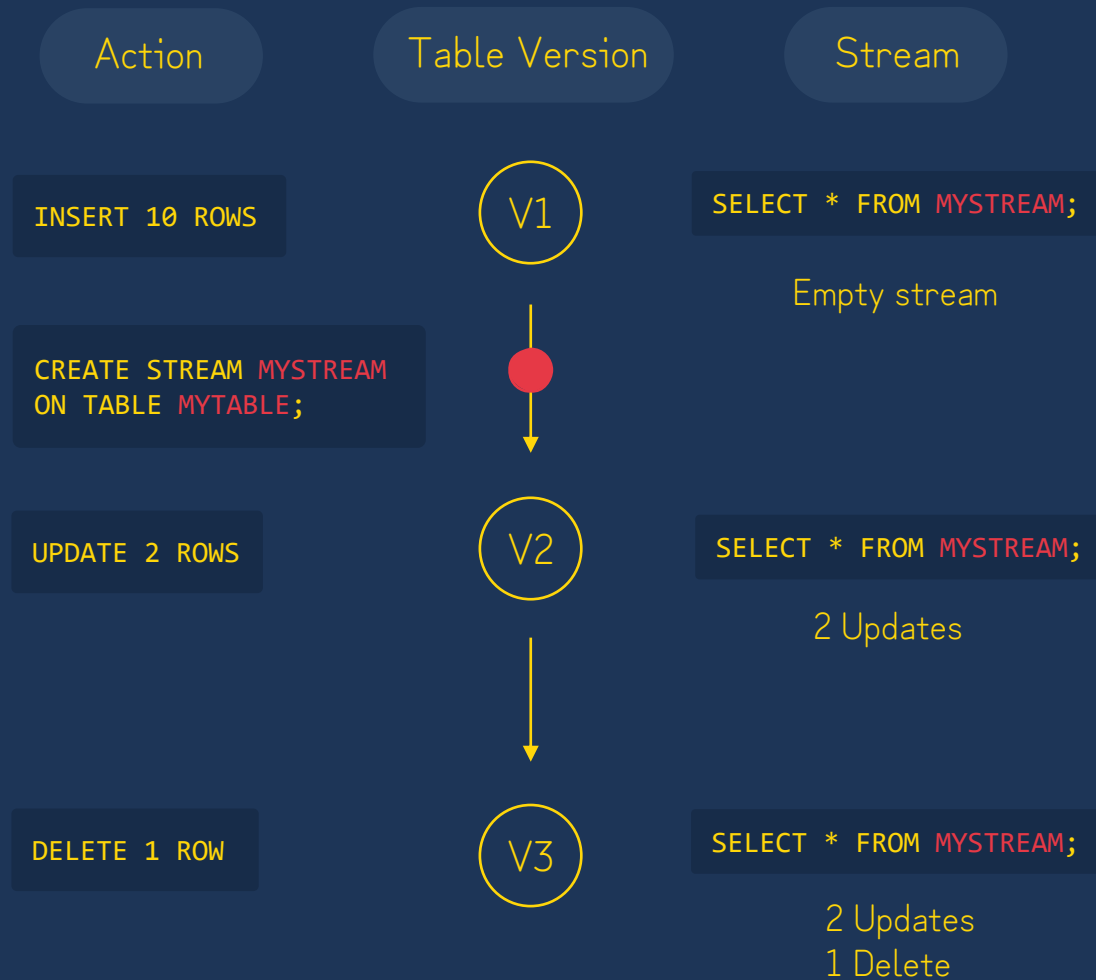
```
CREATE STREAM MY_STREAM ON TABLE MY_TABLE;
```

Query Stream

```
SELECT * FROM MY_STREAM;
```

EMP_ID	EMP_NAME	EMP_DOB	EMP_POSITION	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
AC1109	Ramesh Aravind	10/09/1964	Actor	INSERT	FALSE	cc576bf4fee43b88c4fd03

# Streams



## Progress stream offset

```
INSERT INTO MYTABLE2 SELECT * FROM MYSTREAM;
```

## Tasks & Streams

```
CREATE TASK MYTASK1
  WAREHOUSE = MYWH
  SCHEDULE = '5 MINUTE'
  WHEN
    SYSTEM$STREAM_HAS_DATA('MYSTREAM')
  AS
    INSERT INTO MYTABLE1(ID,NAME) SELECT ID, NAME
    FROM MYSTREAM WHERE METADATA$ACTION = 'INSERT';
```



# Billing

# Billing Overview



On-demand

Pay for usage as you go



Capacity

Pay for usage upfront

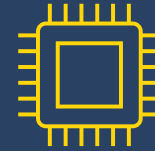
# Billing Overview



Virtual Warehouse  
Services



Cloud Services



Serverless Services



Storage

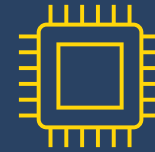


Data Transfer

# Billing Overview



Credits



Dollar Value



# Compute Billing Overview



**Credits** | Snowflake billing unit of measure for compute resource consumption.



## Virtual Warehouse Services

- Credit calculated based on size of virtual warehouse.
- Credit calculated on per second basis while a virtual warehouse is in 'started' state.
- Credit calculated with a minimum of 60 seconds.



## Cloud Services

- Credits calculated at a rate of 4.4 Credits per compute hour.
- Only cloud services that exceeds 10% of the daily usage of the compute resources are billed.
- This is called the Cloud Services Adjustment.



## Serverless Services

- Each serverless feature has it's own credit rate per compute-hour.
- Serverless features are composed of both compute services and cloud services.
- Cloud Services Adjustment does not apply to cloud services usage when used by serverless features.

# Data Storage & Transfer Billing Overview

\$ Dollar value | Storage and Data Transfer are billed in currency.



Data Storage



Data Transfer

- Data storage is calculated monthly based on the average number of on-disk bytes per day in the following locations:
  - Database Tables.
  - Internal Stages.
- Costs calculated based on a flat dollar value rate per terabyte (TB) based on:
  - Capacity or On-demand.
  - Cloud provider.
  - Region.
- Data transfer charges apply when moving data from one region to another or from one cloud platform to another.
- Unloading data from Snowflake using **COPY INTO <location>** command.
- Replicating data to a Snowflake account in a different region or cloud platform.
- External functions transferring data out of and into Snowflake.

# SnowCD

①

```
SELECT SYSTEM$WHITELIST();
```

Returns hostnames and port numbers.



```
1 [{"type":"SNOWFLAKE_DEPLOYMENT","host":"ht09440.eu-west-2.aws.snowflakecomputing.com","port":443},
2 {"type":"SNOWFLAKE_DEPLOYMENT_REGIONLESS","host":"jilikhy-ke89319.snowflakecomputing.com","port":443},
3 {"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com","port":443},
4 {"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com","port":443},
5 {"type":"STAGE","host":"sfc-uk-ds1-3-customer-stage.s3.amazonaws.com","port":443},
6 {"type":"SNOWSQL_REPO","host":"sfc-repo.snowflakecomputing.com","port":443},
7 {"type":"OUT_OF_BAND_TELEMETRY","host":"client-telemetry.snowflakecomputing.com","port":443},
8 {"type":"OCSP_CACHE","host":"ocsp.snowflakecomputing.com","port":80},
9 {"type":"DUO_SECURITY","host":"api-edddc45f.duosecurity.com","port":443},
10 {"type":"OCSP_RESPONDER","host":"ocsp.rootg2.amazontrust.com","port":80},
11 {"type":"OCSP_RESPONDER","host":"o.ss2.us","port":80},
12 {"type":"OCSP_RESPONDER","host":"ocsp.sca1b.amazontrust.com","port":80},
13 {"type":"OCSP_RESPONDER","host":"ocsp.rootca1.amazontrust.com","port":80}]
```

```
13 [{"type":"OCSP_RESPONDER","host":"ocsp.rootg2.amazontrust.com","port":80},
14 {"type":"OCSP_RESPONDER","host":"ocsp.sca1b.amazontrust.com","port":80},
15 {"type":"OCSP_RESPONDER","host":"o.ss2.us","port":80}]
```

# SnowCD

whitelist.json

```
1 [{"type": "SNOWFLAKE_DEPLOYMENT", "host": "ht09440.eu-west-2.aws.snowflakecomputing.com", "port": 443},
2 {"type": "SNOWFLAKE_DEPLOYMENT_REGIONLESS", "host": "jilikh-y-ke89319.snowflakecomputing.com", "port": 443},
3 {"type": "STAGE", "host": "sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com", "port": 443},
4 {"type": "STAGE", "host": "sfc-uk-ds1-3-customer-stage.s3.eu-west-2.amazonaws.com", "port": 443},
5 {"type": "STAGE", "host": "sfc-uk-ds1-3-customer-stage.s3.amazonaws.com", "port": 443},
6 {"type": "SNOWSQL_REPO", "host": "sfc-repo.snowflakecomputing.com", "port": 443},
7 {"type": "OUT_OF_BAND_TELEMETRY", "host": "client-telemetry.snowflakecomputing.com", "port": 443},
8 {"type": "OCSP_CACHE", "host": "ocsp.snowflakecomputing.com", "port": 80},
9 {"type": "DUO_SECURITY", "host": "api-edddc45f.duosecurity.com", "port": 443},
10 {"type": "OCSP_RESPONDER", "host": "ocsp.rootg2.amazontrust.com", "port": 80},
11 {"type": "OCSP_RESPONDER", "host": "o.ss2.us", "port": 80},
12 {"type": "OCSP_RESPONDER", "host": "ocsp.scalb.amazontrust.com", "port": 80},
13 {"type": "OCSP_RESPONDER", "host": "ocsp.rootcal.amazontrust.com", "port": 80}]
14 [{"type": "OCSP_RESPONDER", "host": "ocsp.snowflakecomputing.com", "port": 80}]
15 [{"type": "OCSP_RESPONDER", "host": "ocsp.snowflakecomputing.com", "port": 80}]
16 [{"type": "OCSP_RESPONDER", "host": "ocsp.snowflakecomputing.com", "port": 80}]
```



snowcd ~\whitelist.json



Performing 30 checks on 12 hosts  
All checks passed.



Check for 5 hosts failed, display as follow:

```
=====
Host: ocsp.snowflakecomputing.com
Port: 80
Type: OCSP_CACHE
Failed Check: HTTP checker
Error: Invalid http code received: 400 Bad Request
Suggestion: Check the connection to your http host or transparent Proxy
```



# Connectivity: Connectors, Drivers and Partnered Tools

# Connectors and Drivers



Python



Go



PHP



.NET



NodeJS



Spark



Kafka



JDBC



ODBC

ODBC

Connect third-party tools

TOM BAILEY COURSES

[tombaileycourses.com](https://tombaileycourses.com)

# Python Connector Example

```
pip install snowflake-connector-python==2.6.2
```

```
#!/usr/bin/env python
import snowflake.connector

# Gets the version
ctx = snowflake.connector.connect(
    user='<user_name>',
    password='<password>',
    account='<account_identifier>'
)
cs = ctx.cursor()
try:
    cs.execute("SELECT current_version()")
    one_row = cs.fetchone()
    print(one_row[0])
finally:
    cs.close()
ctx.close()
```

# Snowflake Partner Tools



Business Intelligence



Data Integration



Security & Governance



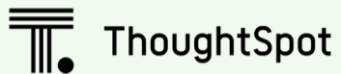
SQL Development &  
Management



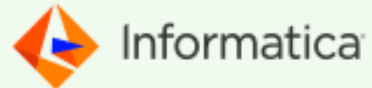
Machine Learning &  
Data Science

# Snowflake Partner Tools

## Business Intelligence



## Data Integration



## Security & Governance



 Snowflake Partner Connect is a feature to expedite connectivity with partnered tools.

TOM BAILEY COURSES

[tombaileycourses.com](https://tombaileycourses.com)

# Snowflake Partner Tools

## Machine Learning & Data Science



**DataRobot**



**Amazon  
SageMaker**



## SQL Development & Management



**SEEK WELL**



**Hackolade**



Agile Data  
**ENGINE**

# Snowflake Scripting

# Snowflake Scripting



**Snowflake Scripting** is an extension to Snowflake SQL that adds support for **procedural logic**.

It's used to write **stored procedures** and **procedural code** outside of a stored procedure.

```
DECLARE
```

```
    (variable declarations, cursor declarations, etc.)
```

```
BEGIN
```

```
    (Snowflake Scripting and SQL statements)
```

```
EXCEPTION
```

```
    (statements for handling exceptions)
```

```
END;
```



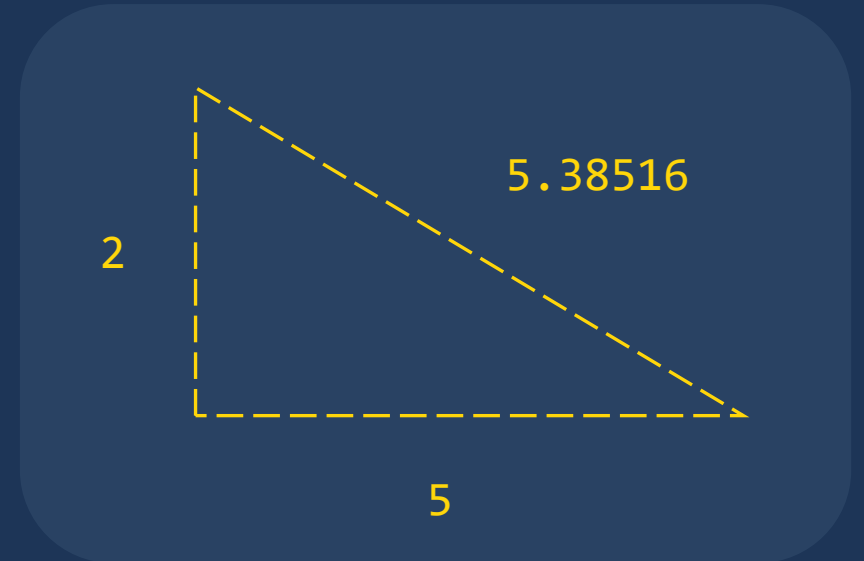
# Snowflake Scripting

```
declare
    leg_a number(38, 2);
    hypotenuse number(38,5);
begin
    leg_a := 2;
    let leg_b := 5;

    hypotenuse := sqrt(square(leg_a) + square(leg_b));
    return hypotenuse;
end;
```

anonymous block

5.38516



① Variables can only be used within the scope of the block.

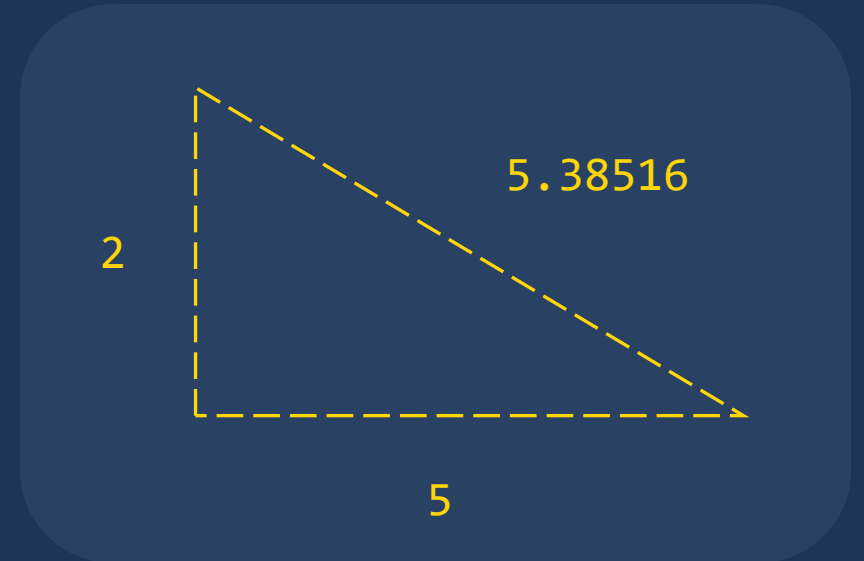
① Variables can also be declared and assigned in the BEGIN section using the LET keyword.

TOM BAILEY COURSES

tombaileycourses.com

# Snowflake Scripting

```
CREATE PROCEDURE pythagoras()  
RETURNS float  
LANGUAGE sql  
AS  
declare  
    leg_a number(38, 2);  
    hypotenuse number(38,5);  
begin  
    leg_a := 2;  
    let leg_b := 5;  
  
    hypotenuse := sqrt(square(leg_a) + square(leg_b));  
    return hypotenuse;  
end;
```



SnowSQL and the Classic Console do not correctly parse Snowflake Scripting blocks, they need to be wrapped in string constant delimiters like dollar signs.

# Branching Constructs

```
begin
  let count := 4;
  if (count % 2 = 0) then
    return 'even value';
  else
    return 'odd value';
  end if;
end;
```



DECLARE or EXCEPTION sections of a block are optional.

anonymous block
even value

# Looping Constructs

```
declare
    total integer default 0;
    max_num integer default 10;
begin
    for i in 1 to max_num do
        total := i + total;
    end for;
    return total;
end;
```

anonymous block

55

# Cursor

```
declare
    total_amount float;
    c1 cursor for select amount from transactions;
begin
    total_amount := 0.0;
    for record in c1 do
        total_amount := total_amount + record.amount;
    end for;
    return total_amount;
end;
```

anonymous block
-----------------

136.78
--------

# RESULTSET

## TABLE()

```
declare
    res resultset;
begin
    res := (select amount from transactions);
    return table(res);
end;
```

amount
101.01
24.78
10.99

# RESULTSET

## Cursor

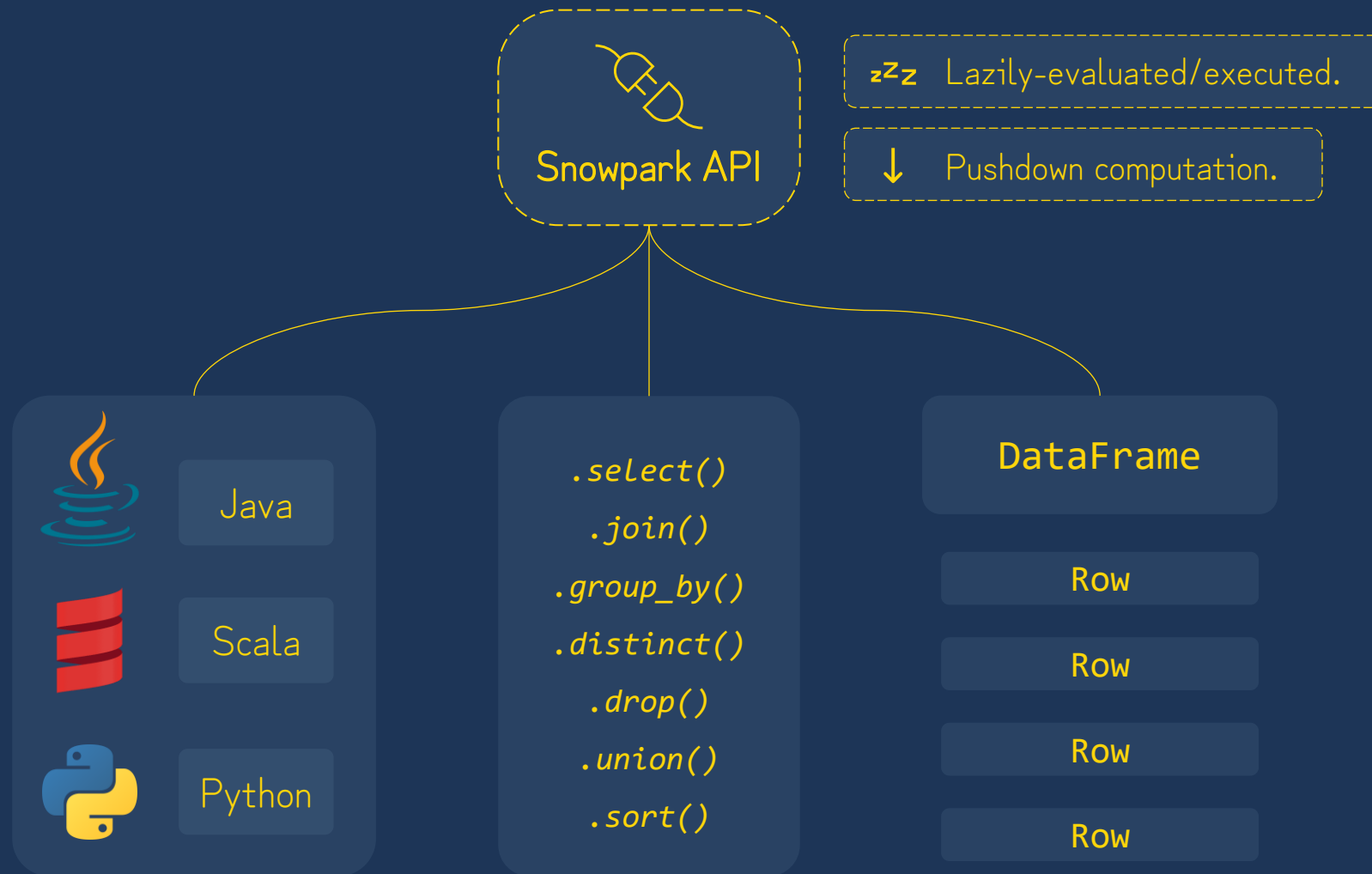
```
declare
    total_amount float;
    res resultset default (select amount from transactions);
    c1 cursor for res;
begin
    total_amount := 0.0;
    for record in c1 do
        total_amount := total_amount + record.amount;
    end for;
    return total_amount;
end;
```

anonymous block
136.78

# Snowpark



# Snowpark



# Snowpark API: Python

```
1 import os
2 from snowflake.snowpark import Session
3 from snowflake.snowpark.functions import col
```

```
5 connection_parameters = {
6     "account": os.environ["snowflake_account"],
7     "user": os.environ["snowflake_user"],
8     "password": os.environ["snowflake_password"],
9     "role": os.environ["snowflake_user_role"],
10    "warehouse": os.environ["snowflake_warehouse"],
11    "database": os.environ["snowflake_database"],
12    "schema": os.environ["snowflake_schema"]
13 }
```

# Snowpark API: Python

```
14 session = Session.builder.configs(connection_parameters).create()

15 transactions_df = session.table("transactions")

16 print(transactions_df.collect())
```

Console output:

```
[Row(ACCOUNT_ID=8764442, AMOUNT=12.99),
Row(ACCOUNT_ID=8764442, AMOUNT=50.0),
Row(ACCOUNT_ID=8764442, AMOUNT=1100.0),
Row(ACCOUNT_ID=8764443, AMOUNT=110.0),
Row(ACCOUNT_ID=8764443, AMOUNT=2766.0),
Row(ACCOUNT_ID=8764443, AMOUNT=1010.0),
Row(ACCOUNT_ID=8764443, AMOUNT=3022.23),
Row(ACCOUNT_ID=8764444, AMOUNT=6986.0),
Row(ACCOUNT_ID=8764444, AMOUNT=1500.0)]
```

# Snowpark API: Python

```
18 transactions_df_filtered = transactions_df.filter(col("amount") >= 1000.00)

19 transaction_counts_df = transactions_df_filtered.group_by("account_id").count()

20 flagged_transactions_df = transaction_counts_df.filter(col("count") >= 2).rename(col("count"), "flagged_count")

21 flagged_transactions_df.write.save_as_table("flagged_transactions", mode="append")

22 print(flagged_transactions_df.show())

23 session.close()
```

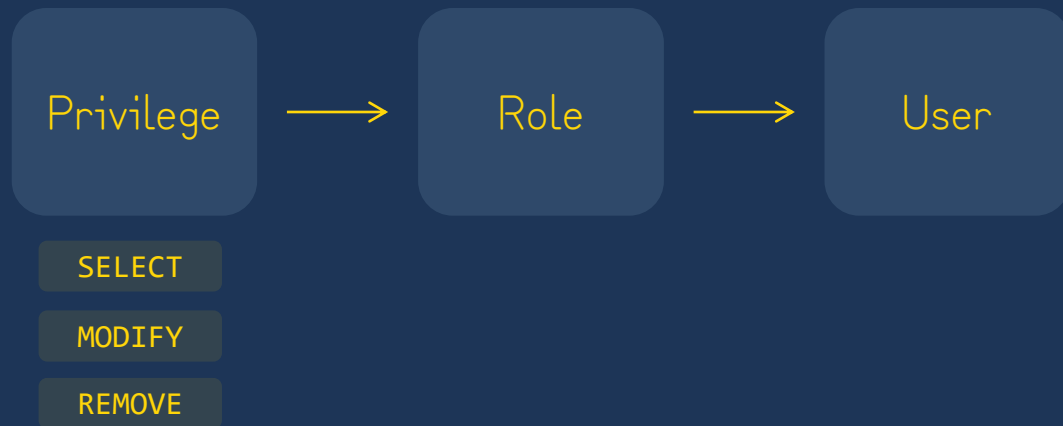
Console output:

"ACCOUNT_ID"	"FLAGGED_COUNT"
8764443	3
8764444	2

# Access Control Overview

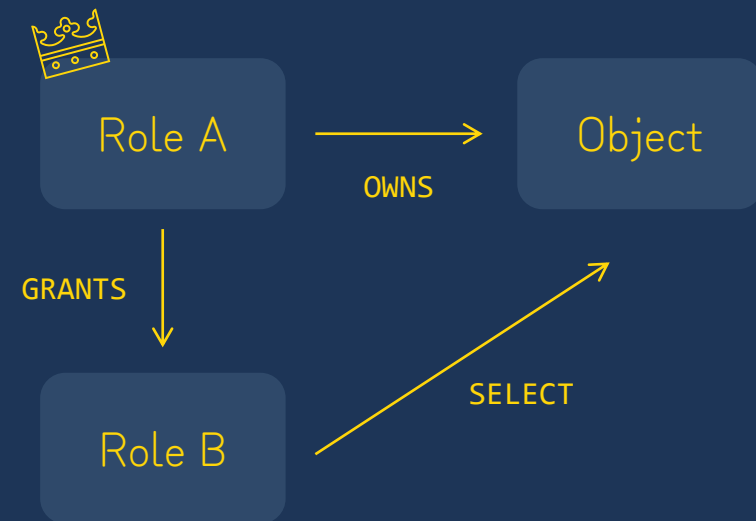
# Access Control Overview

## Role-based Access Control (RBAC)



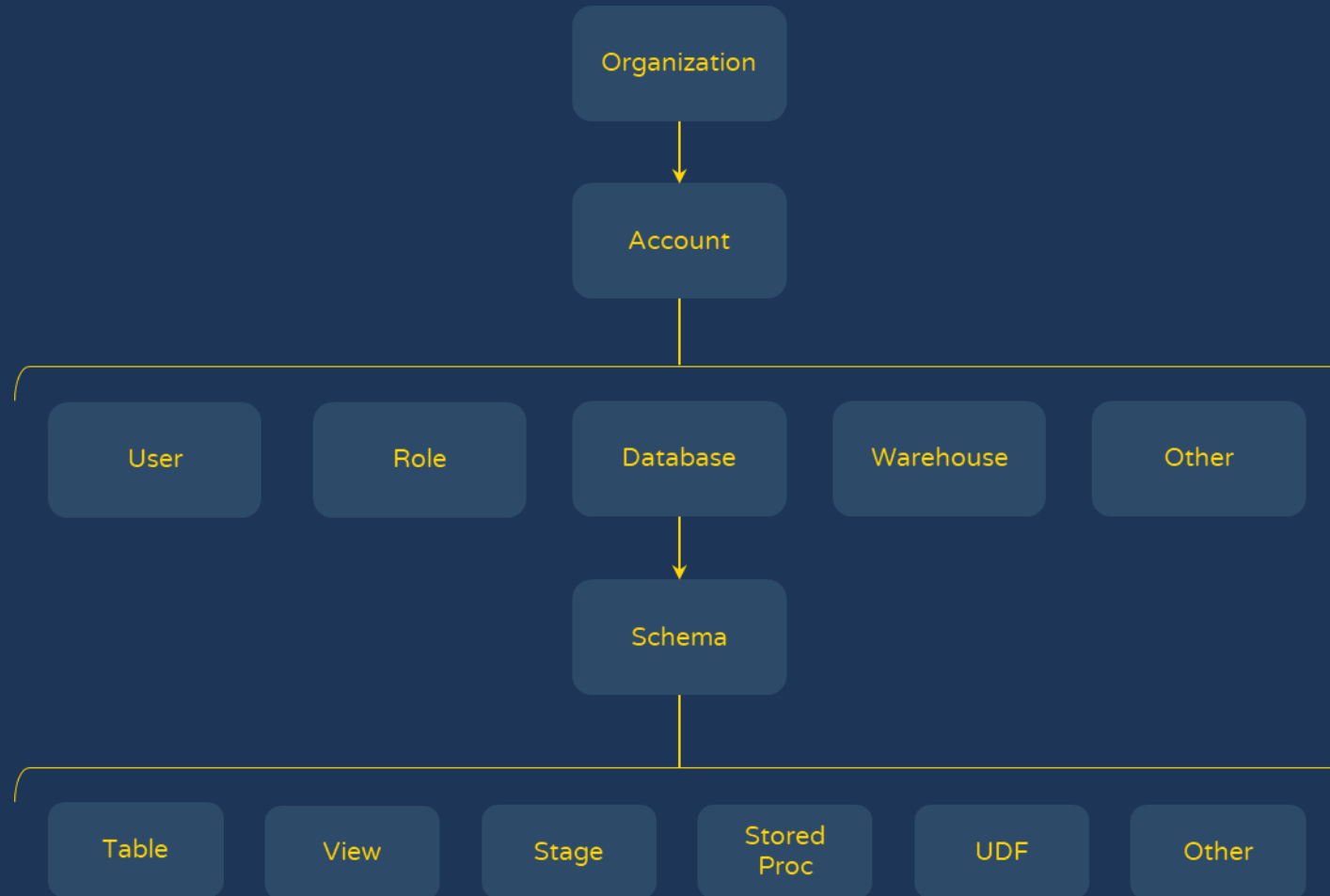
Role-based access control (RBAC) is an access control framework in which access privileges are assigned to roles and in turn assigned to users.

## Discretionary Access Control (DAC)



Snowflake combines RBAC with Discretionary Access Control (DAC) in which each object has an owner, who can in turn grant access to that object.

# Securable Objects



# Securable Objects

Every securable object is owned by a single role which can be found by executing a `SHOW <object>` command.

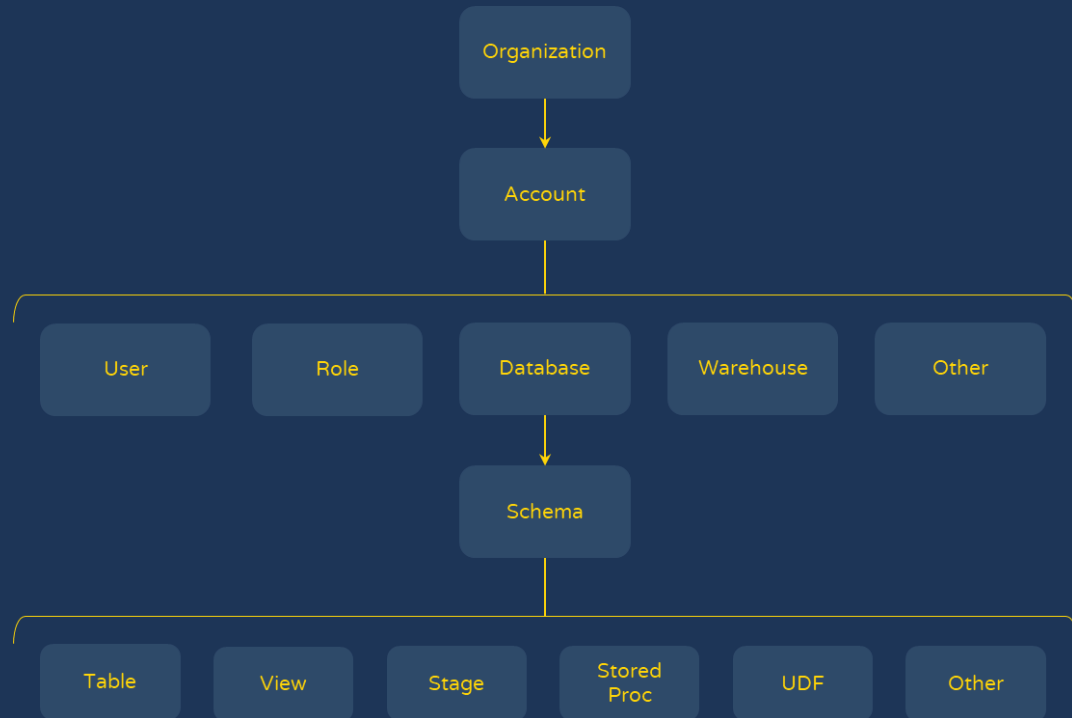
The owning role:

- Has all privileges on the object by default.
- Can grant or revoke privileges on the object to other roles.
- Transfer ownership to another role.
- Share control of an object if the owning role is shared.

Access to objects is also defined by privileges granted to roles:

- Ability to create a Warehouse.
- Ability to list tables contained in a schema
- Ability to add data to a table

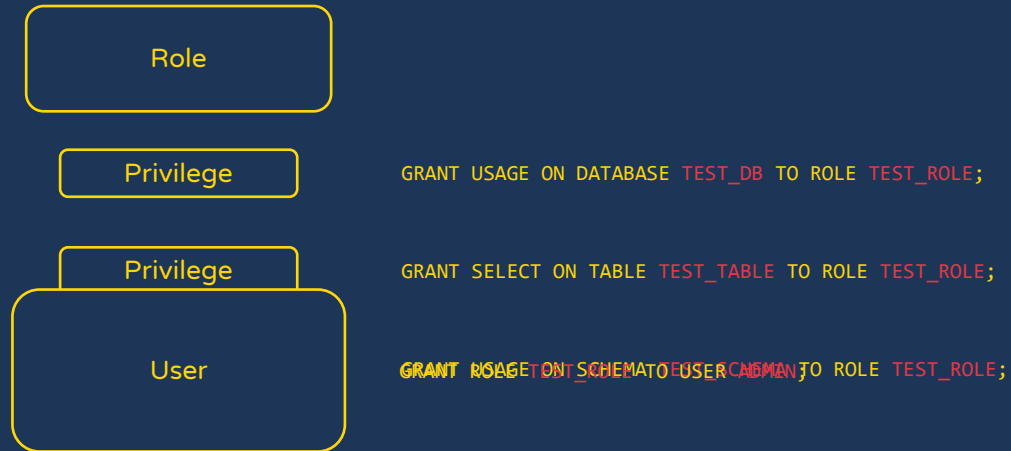
Unless allowed by a grant, access to a securable object will be denied.





# Roles

# Roles



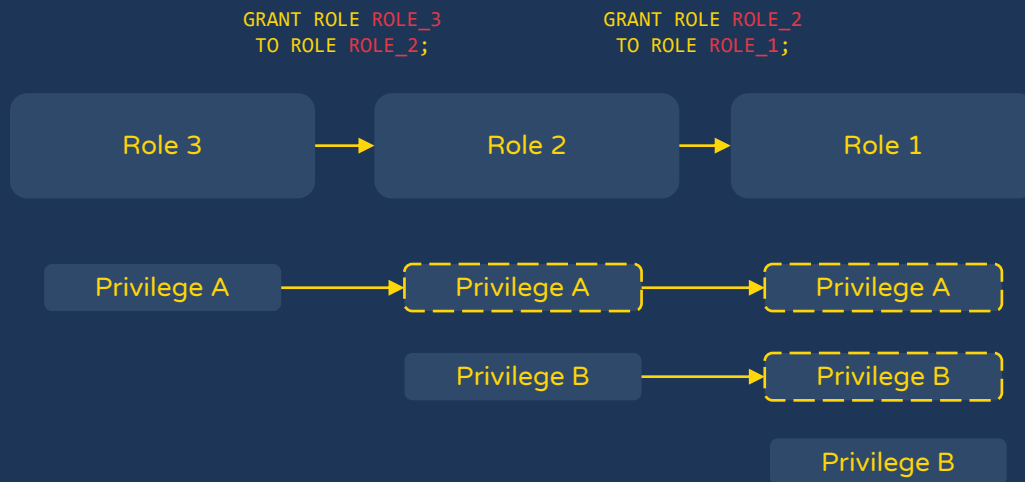
A role is an entity to which privileges on securable objects can be granted or revoked.

Roles are assigned to users to give them the authorization to perform actions.

A user can have multiple roles and switch between them within a Snowflake session.

Roles can be granted to other roles creating a role hierarchy.

Privileges of child roles are inherited by parent roles.



# System-defined Roles

ORGADMIN

ACCOUNTADMIN

SECURITYADMIN

SYSADMIN

USERADMIN

PUBLIC

# System-defined Roles



## ORGADMIN

- Manages operations at organization level.
- Can create account in an organization.
- Can view all accounts in an organization.
- Can view usage information across an organization.

## ACCOUNTADMIN

- Top-level and most powerful role for an account.
- Encapsulates SYSADMIN & SECURITYADMIN.
- Responsible for configuring account-level parameters.
- View and operate on all objects in an account.
- View and manage Snowflake billing and credit data.
- Stop any running SQL statements.

## SYSADMIN

- Can create warehouses, databases, schemas and other objects in an account.

# System-defined Roles



## SECURITYADMIN

- Manage grants globally via the MANAGE GRANTS privilege.
- Create, monitor and manage users and roles.

## USERADMIN

- User and Role management via CREATE USER and CREATE ROLE security privileges.
- Can create users and roles in an account.

## PUBLIC

- Automatically granted to every user and every role in an account.
- Can own securable objects, however objects owned by PUBLIC role are available to every other user and role in an account.

# Custom Roles



Custom roles allows you to create a role with custom and fine-grained security privileges defined.

Custom roles allow administrators working with the system-defined roles to exercise the security principle of least privilege.

Custom roles can be created by the **SECURITYADMIN** & **USERADMIN** roles as well as by any role to which the **CREATE ROLE** privilege has been granted.

It is recommended to create a hierarchy of custom roles with the top-most custom role assigned to the **SYSADMIN** role.

If custom roles are not assigned to the **SYSADMIN** role, system admins will not be able to manage the objects owned by the custom role.

# Privileges

# Privileges

A security privilege defines a level of access to an object.

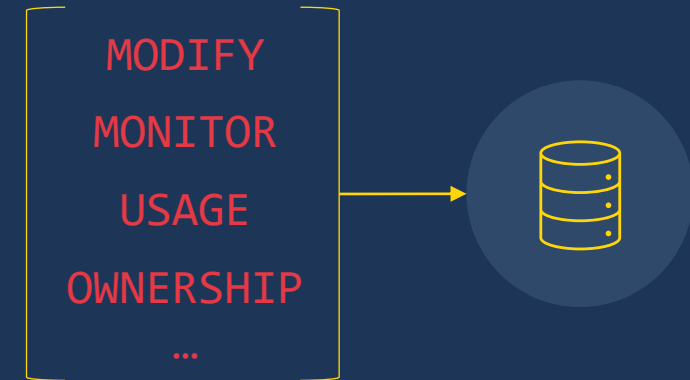
For each object there is a set of security privileges that can be granted on it.

There are 4 categories of security privileges:

- Global Privileges
- Privileges for account objects
- Privileges for schemas
- Privileges for schema objects

Privileges are managed using the GRANT and REVOKE commands.

Future grants allow privileges to be defined for objects not yet created.



```
GRANT USAGE ON DATABASE MY_DB TO ROLE MY_ROLE;
```

```
REVOKE USAGE ON DATABASE MY_DB TO ROLE MY_ROLE;
```

```
GRANT SELECT ON FUTURE TABLES IN SCHEMA MY_SCHEMA TO ROLE MY_ROLE;
```



# User Authentication

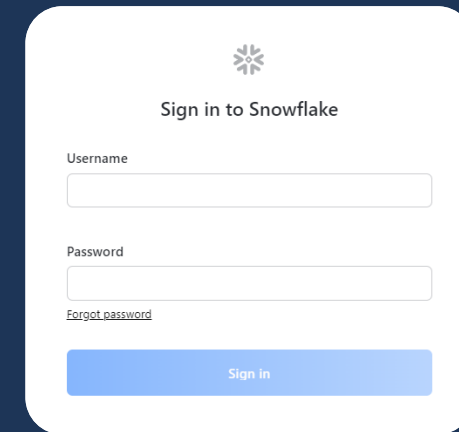
# User Authentication

User authentication is the process of authenticating with Snowflake via user provided username and password credentials.

User authentication is the default method of authentication.

Users with the USERADMIN role can create additional Snowflake users, which makes use of the CREATE USER privilege.

- › A password can be any case-sensitive string up to 256 characters.
- › Must be at least 8 characters long.
- › Must contain at least 1 digit.
- › Must contain at least 1 uppercase letter and 1 lowercase letter.

A screenshot of the Snowflake sign-in interface. It features the Snowflake logo at the top, followed by the text "Sign in to Snowflake". Below this are two input fields: "Username" and "Password". A link labeled "Forgot password" is positioned below the password field. At the bottom is a blue "Sign in" button.

```
CREATE USER USER1  
PASSWORD='ABC123'  
DEFAULT_ROLE = MYROLE  
MUST_CHANGE_PASSWORD = TRUE;
```

`'q@-*DaC2yjZoq3Re4JYX'`

# MFA

# Multi-factor Authentication (MFA)

MFA is an additional layer of security, requiring the user to prove their identity not only with a password but with an additional piece of information (or factor).



MFA in Snowflake is powered by a service called Duo Security.



MFA is enabled on a per-user basis & only via the UI.

## Multi-factor Authentication

Enroll in MFA, edit the phone number associated with your MFA account.

Status Not Enrolled [Enroll in MFA](#)

Phone -

Snowflake recommend that all users with the ACCOUNTADMIN role be required to use MFA.

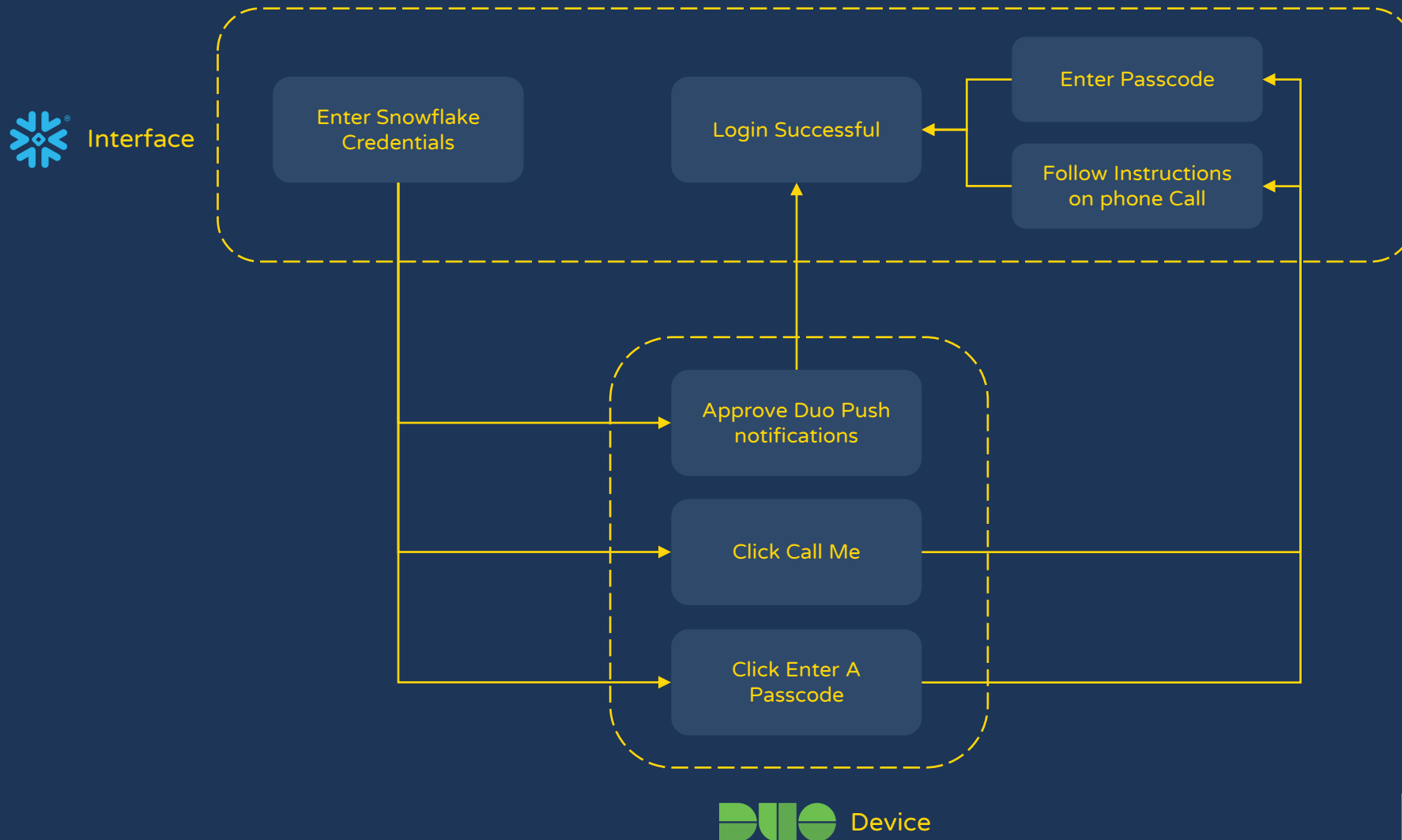


ACCOUNTADMIN

TOM BAILEY COURSES

tombaileycourses.com

# Multi-factor Authentication Flow



# MFA Properties

MINS\_TO\_BYPASS\_MFA

```
ALTER USER USER1 SET  
MINS_TO_BYPASS_MFA=10;
```

Specifies the number of minutes to temporarily disable MFA for the user so that they can log in.

DISABLE\_MFA

```
ALTER USER USER1 SET  
DISABLE_MFA=TRUE;
```

Disables MFA for the user, effectively cancelling their enrolment. To use MFA again, the user must re-enrol.

ALLOWS\_CLIENT\_MFA\_CACHING

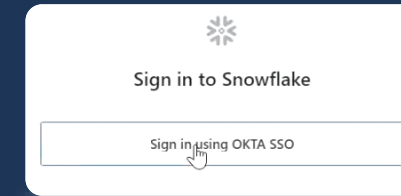
```
ALTER USER USER1 SET  
ALLOWS_CLIENT_MFA_CACHING=TRUE;
```

MFA token caching reduces the number of prompts that must be acknowledged while connecting and authenticating to Snowflake.

# Federated Authentication

# Federated Authentication (SSO)

Federated authentication enables users to connect to Snowflake using secure SSO (single sign-on).



Snowflake can delegate authentication responsibility to an SAML 2.0 compliant external identity provider (IdP) with native support for Okta and ADFS IdPs.



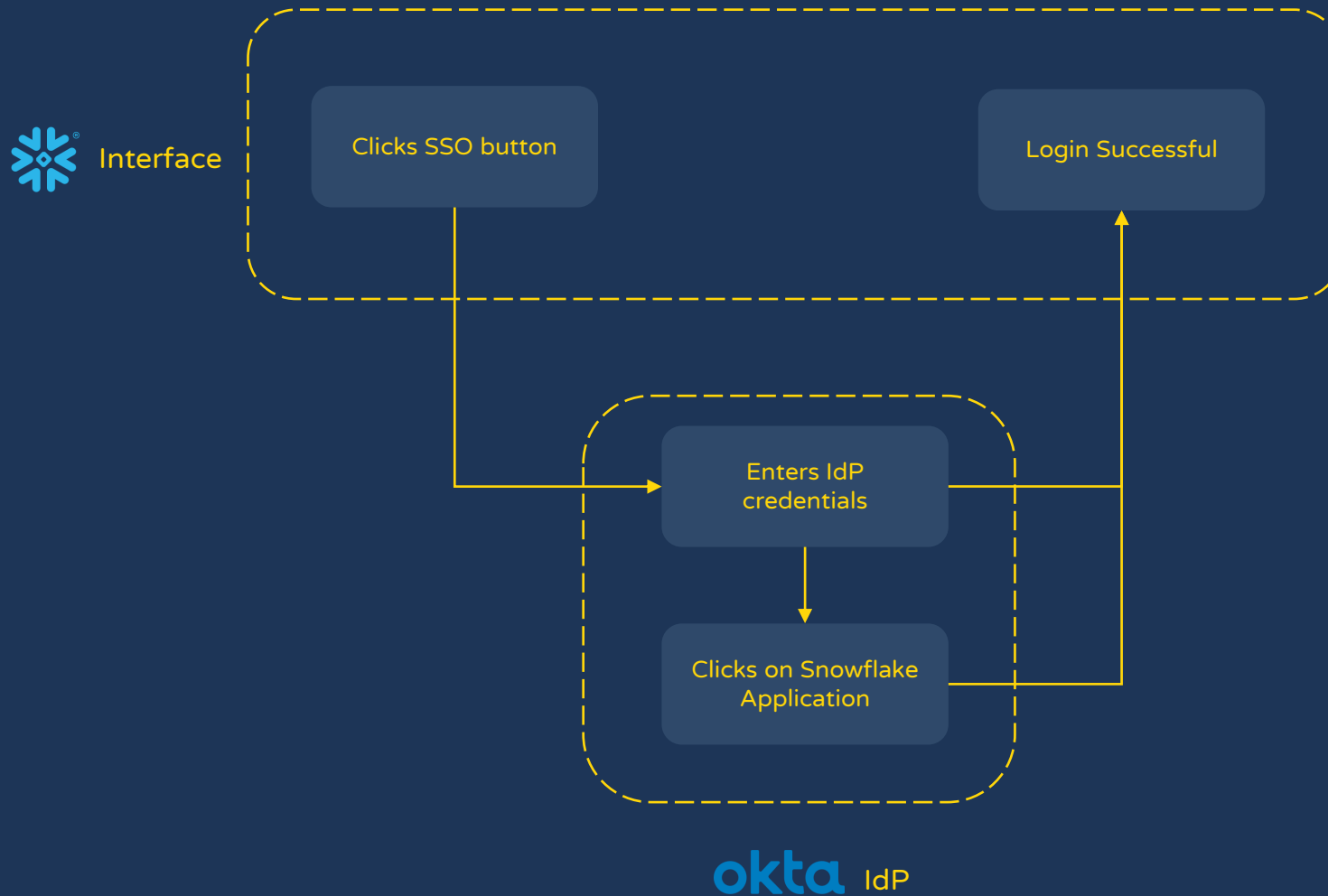
An IdP is an independent service responsible for creating and maintaining user credentials as well as authenticating users for SSO access to Snowflake.

In a federated environment Snowflake is referred to as a Service Provider (SP).





# Federated Authentication Login Flow



# Federated Authentication Properties

## SAML\_IDENTITY\_PROVIDER

```
ALTER ACCOUNT SET SAML_IDENTITY_PROVIDER =  
'{  
  "certificate": "XXXXXXXXXXXXXXXXXXXX",  
  "ssoUrl": "https://abccorp.testmachine.com/adfs/ls",  
  "type" : "ADFS",  
  "label" : "ADFSSingleSignOn"  
}';
```

How to specify an IdP during the Snowflake setup of Federated Authentication.

## SSO\_LOGIN\_PAGE

```
ALTER ACCOUNT SET  
SSO_LOGIN_PAGE = TRUE;
```

Enable button for Snowflake-initiated SSO for your identity provider (as specified in SAML\_IDENTITY\_PROVIDER) in the Snowflake main login page.

# Key Pair Authentication, OAuth & SCIM

# Key Pair Authentication



# OAuth & SCIM



OAuth

Snowflake supports the OAuth 2.0 protocol.

OAuth is an open-standard protocol that allows supported clients authorized access to Snowflake without sharing or storing user login credentials.

Snowflake offers two OAuth pathways: Snowflake OAuth and External OAuth.

SCIM

SCIM

System for Cross-domain Identity Management (SCIM) can be used to manage users and groups ( Snowflake roles) in cloud applications using RESTful APIs.



Snowflake (SP)

201



CREATE USER



ADFS (IdP)

# Network Policies

# Network Policies

MY\_NETWORK\_POLICY

us47171.eu-west-2.aws.snowflakecomputing.com

```
CREATE NETWORK POLICY MY_POLICY  
ALLOWED_IP_LIST=('192.168.1.0/24')  
BLOCKED_IP_LIST=('192.168.1.99');
```

Account

User

Network Policies provide the user with the ability to allow or deny access to their Snowflake account based on a single IP address or list of addresses.

Network Policies are composed of an allowed IP range and optionally a blocked IP range. Blocked IP ranges are applied first.

Network Policies currently support only IPv4 addresses.

Network policies use CIDR notation to express an IP subnet range.

Network Policies can be applied on the account level or to individual users.

If a user is associated to both an account-level and user-level network policy, the user-level policy takes precedence.

# Network Policies

```
CREATE NETWORK POLICY MYPOLICY  
ALLOWED_IP_LIST=( '192.168.1.0/24' )  
BLOCKED_IP_LIST=( '192.168.1.99' );
```

```
SHOW NETWORK POLICIES;
```

## ACCOUNT

Only one Network Policy can be associated with an account at any one time.

```
ALTER ACCOUNT SET NETWORK_POLICY = MYPOLICY;
```

SECURITYADMIN or ACCOUNTADMIN system roles can apply policies. Or custom role with the ATTACH POLICY global privilege.

```
SHOW PARAMETERS LIKE 'MYPOLICY' IN ACCOUNT;
```

## USER

Only one Network Policy can be associated with an user at any one time.

```
ALTER USER USER1 SET NETWORK_POLICY = MYPOLICY;
```

SECURITYADMIN or ACCOUNTADMIN system roles can apply policies. Or custom role with the ATTACH POLICY global privilege.

```
SHOW PARAMETERS LIKE 'MYPOLICY' IN USER USER1;
```



# Data Encryption

# Data Encryption

## Encryption At Rest



Table Data



Internal Stage  
Data



AES-256 strong encryption

Virtual Warehouse and Query Result Caches

## Encryption In Transit

ODBC

JDBC

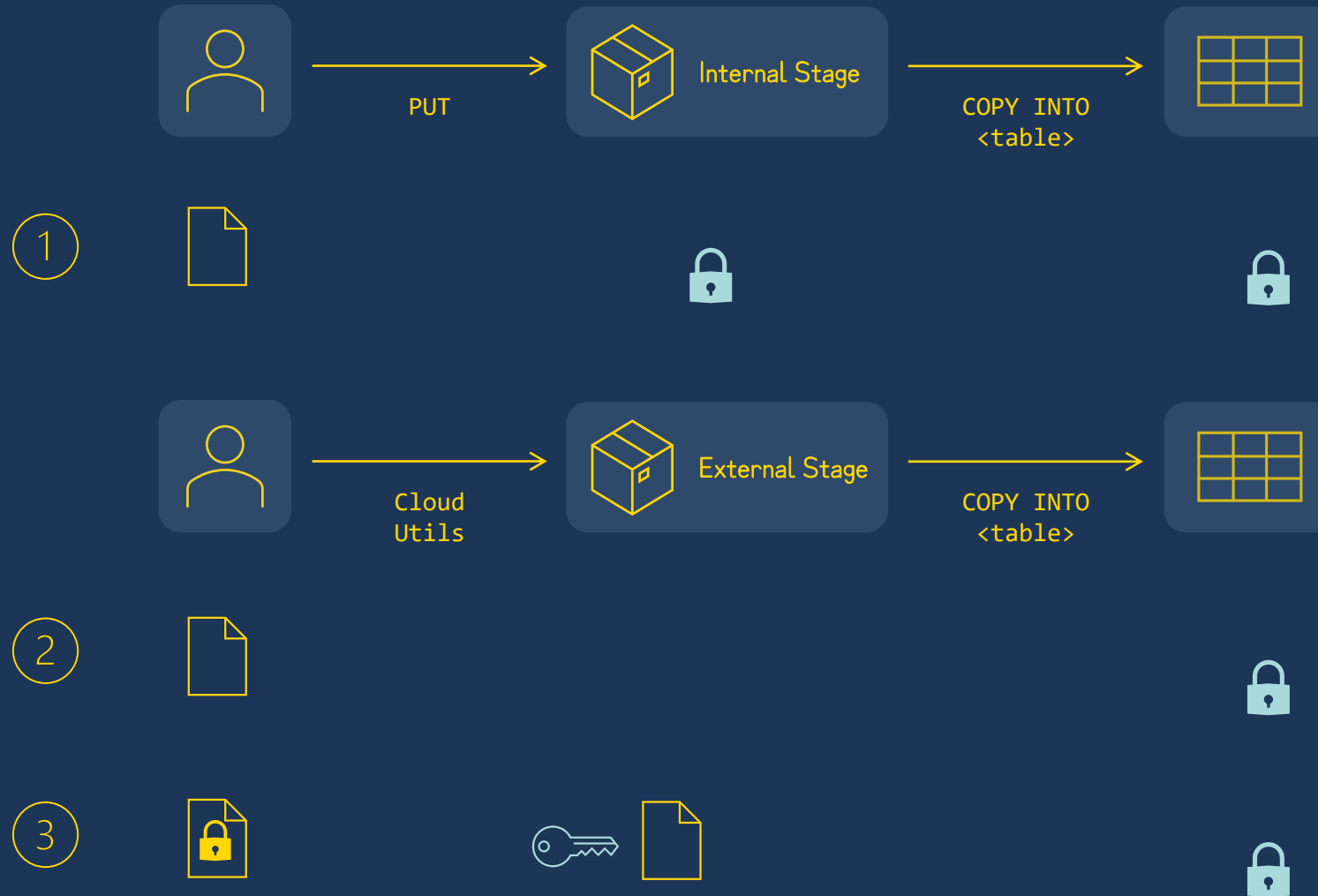
Web UI

SnowSQL

HTTPS  
TLS 1.2



# E2EE Encryption Flows



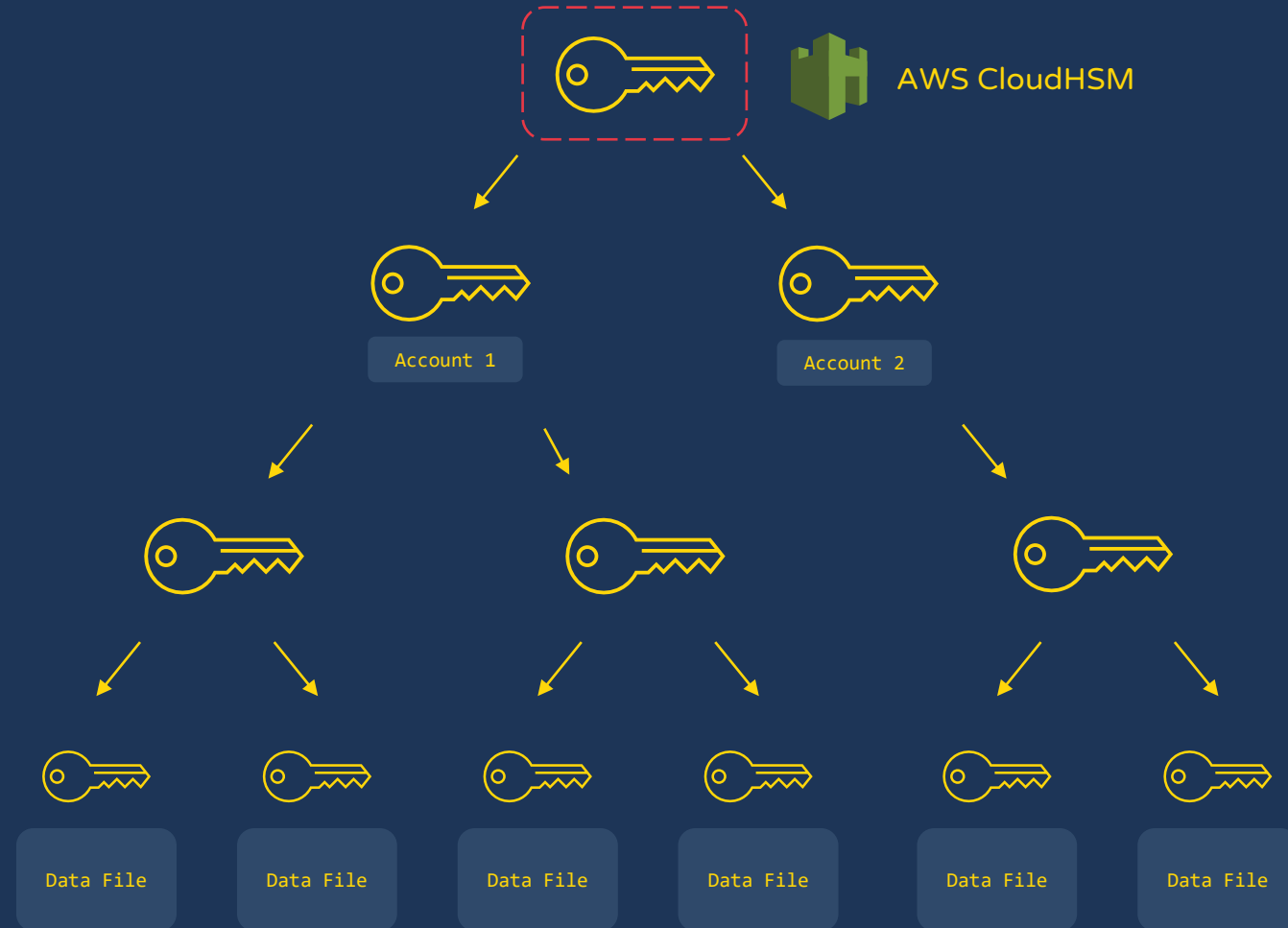
# Hierarchical Key Model

Root Key

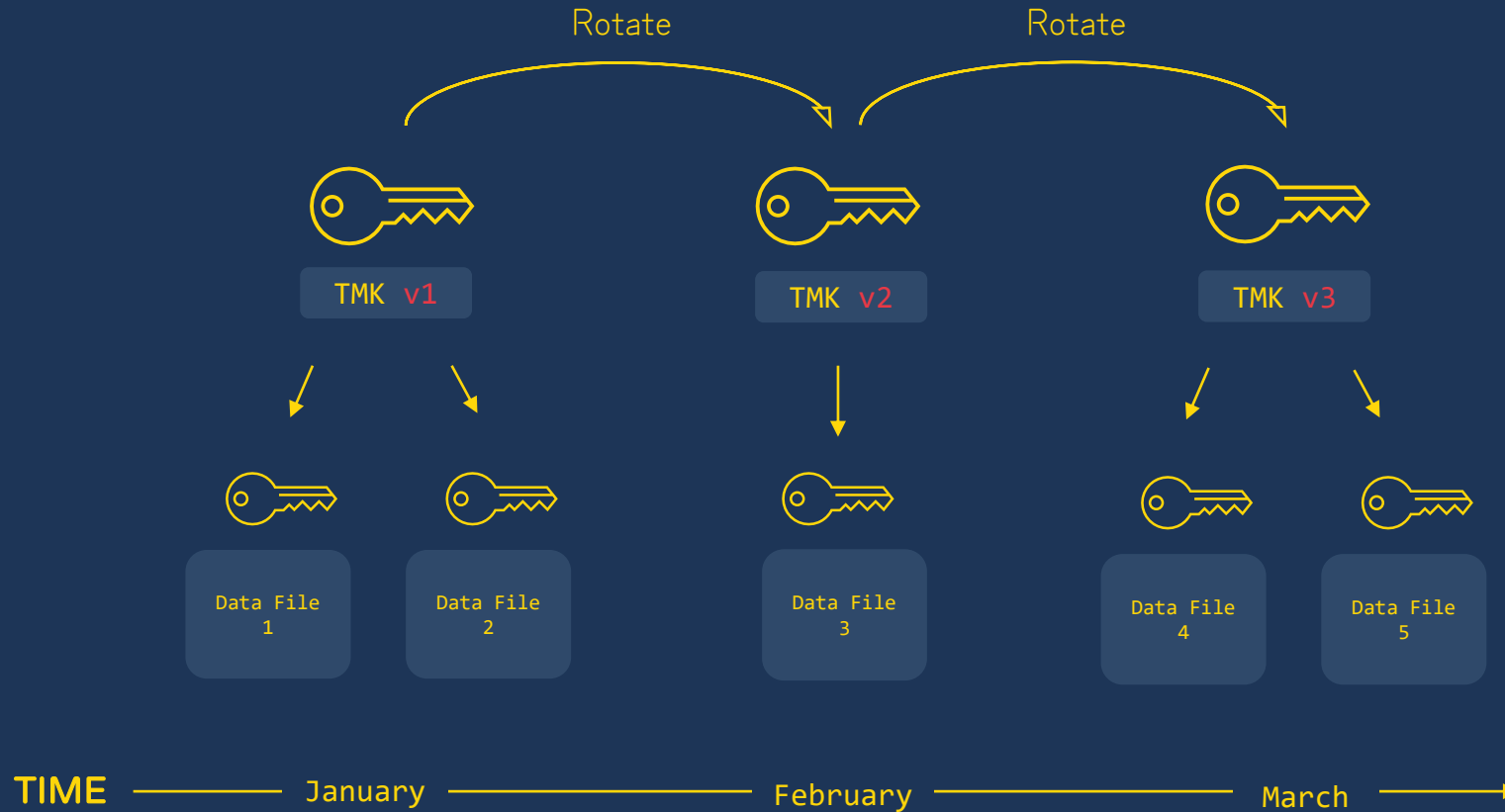
Account Master Keys

Table Master Keys

File Keys

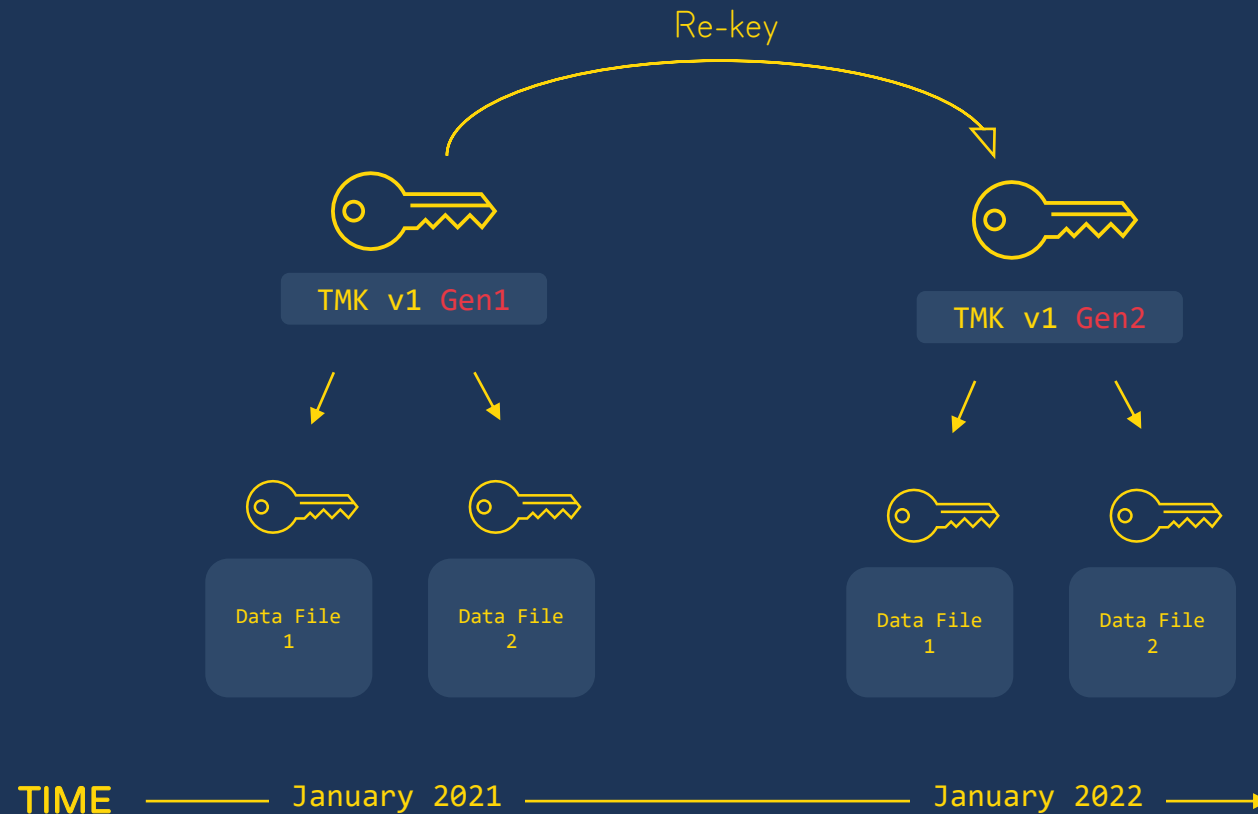


# Key Rotation



Key rotation is the practise of transparently replacing existing account and table encryption keys every **30 days** with a new key.

# Re-Keying



Once a retired key exceeds **1 year**, Snowflake automatically creates a new encryption key and re-encrypts all data previously protected by the retired key using the new key.

```
ALTER ACCOUNT SET PERIODIC_DATA_REKEYING = TRUE;
```

**TOM BAILEY COURSES**

[tombaileycourses.com](https://tombaileycourses.com)

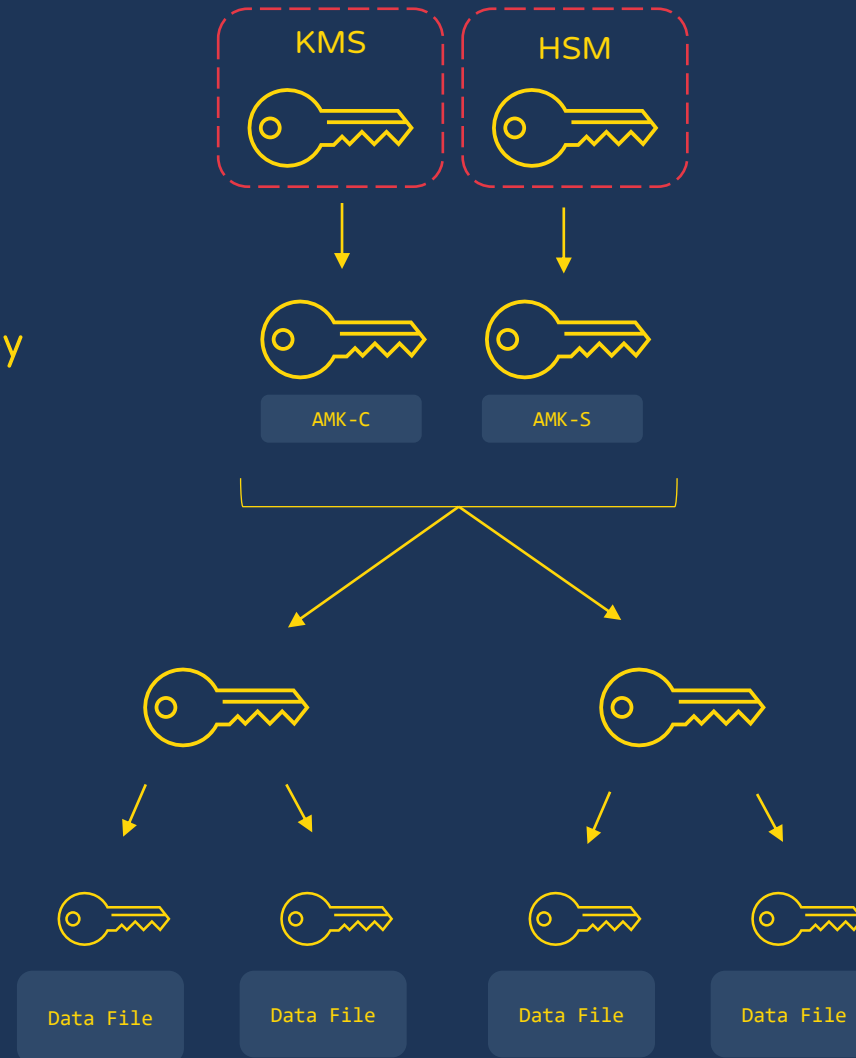
# Tri-secret Secure and Customer Managed Keys

Root Keys

Account Master Composite Key

Table Master Keys

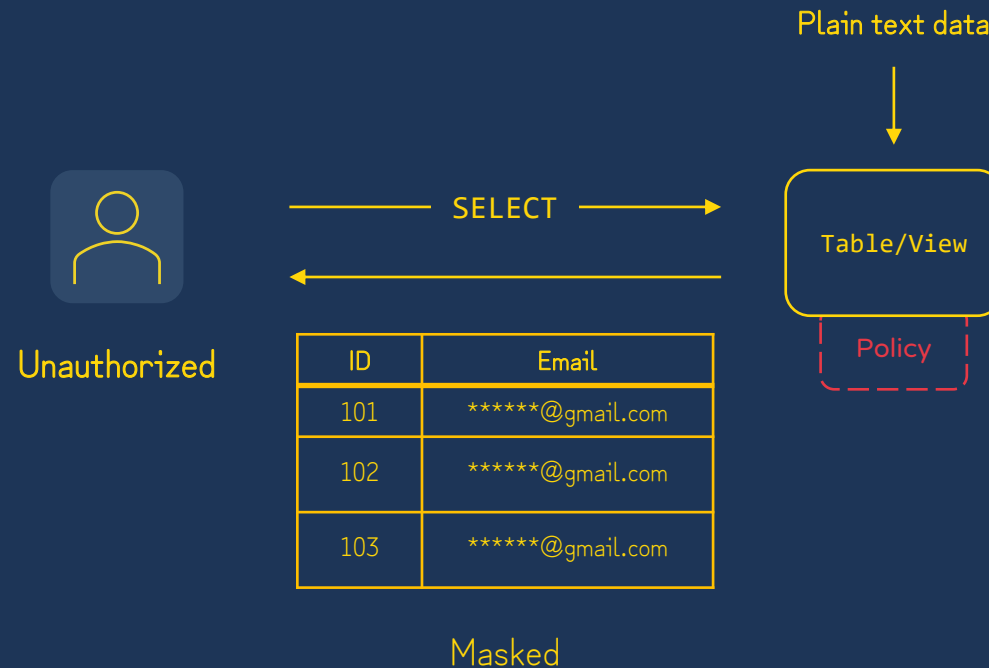
File Keys



# Column Level security

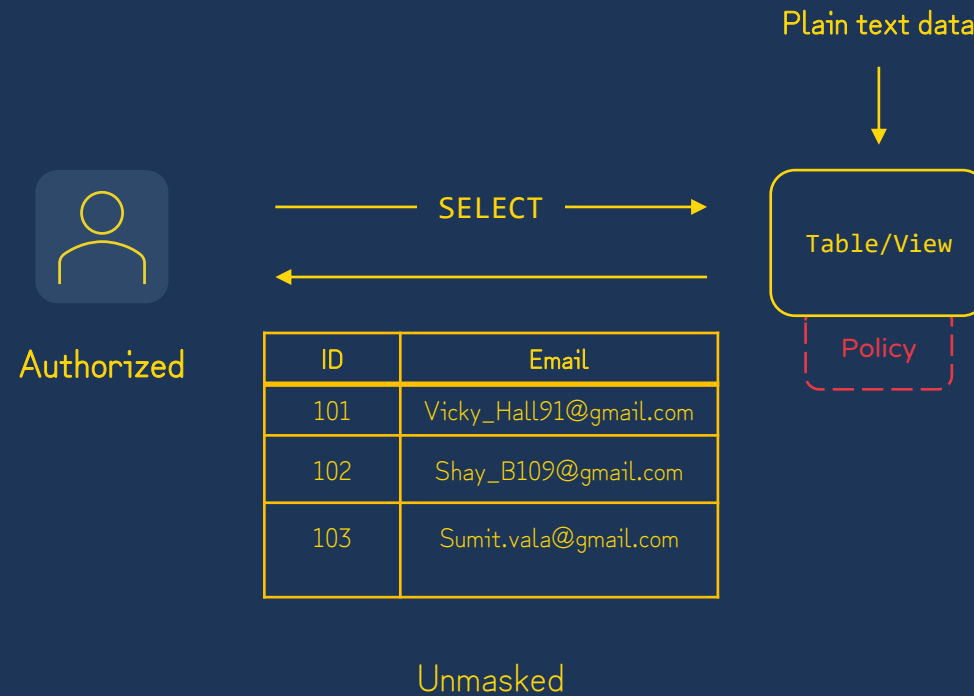


# Dynamic Data Masking



Sensitive data in plain text is loaded into Snowflake, and it is dynamically masked at the time of query for unauthorized users.

# Dynamic Data Masking



Sensitive data in plain text is loaded into Snowflake, and it is dynamically masked at the time of query for unauthorized users.

# Masking Policies

Unmasked	_____	WHEN CURRENT_ROLE() IN ( 'SUPPORT' ) THEN VAL
Partially Masked	_____	WHEN CURRENT_ROLE() IN ( 'ANALYST' ) THEN REGEXP_REPLACE(VAL, '.*\@', '*****@')
System Functions	_____	END;
User Defined Functions	_____	WHEN CURRENT_ROLE() IN ( 'HR' ) THEN SHA2(VAL)
Semi-structured	_____	WHEN CURRENT_ROLE() IN ( 'SALES' ) THEN MASK_UDF(VAL)
Fully Masked	_____	WHEN CURRENT_ROLE() IN ( 'FINANCE' ) THEN OBJECT_INSERT(VAL, 'USER_EMAIL', '*', TRUE)

```
CREATE MASKING POLICY EMAIL_MASK AS (VAL STRING) RETURNS STRING ->
```

```
CASE
```

```
ALTER TABLE IF EXISTS EMP_INFO MODIFY COLUMN USER_EMAIL SET MASKING POLICY EMAIL_MASK;
```

# Masking Policies

Unmasked	_____	WHEN CURRENT_ROLE() IN ( 'SUPPORT' ) THEN VAL
Partially Masked	_____	WHEN CURRENT_ROLE() IN ( 'ANALYST' ) THEN REGEXP_REPLACE(VAL, '.*\@', '*****@')
System Functions	_____	END;
User Defined Functions	_____	WHEN CURRENT_ROLE() IN ( 'HR' ) THEN SHA2(VAL)
Semi-structured	_____	WHEN CURRENT_ROLE() IN ( 'SALES' ) THEN MASK_UDF(VAL)
Fully Masked	_____	WHEN CURRENT_ROLE() IN ( 'FINANCE' ) THEN OBJECT_INSERT(VAL, 'USER_EMAIL', '*', TRUE)

```
CREATE MASKING POLICY EMAIL_MASK AS (VAL STRING) RETURNS STRING ->
```

```
CASE
```

```
ALTER TABLE IF EXISTS EMP_INFO MODIFY COLUMN USER_EMAIL SET MASKING POLICY EMAIL_MASK;
```

# Masking Policies

1

Data masking policies are schema-level objects, like tables & views.

2

Creating and applying data masking policies can be done independently of object owners.

3

Masking policies can be nested, existing in tables and views that reference those tables.

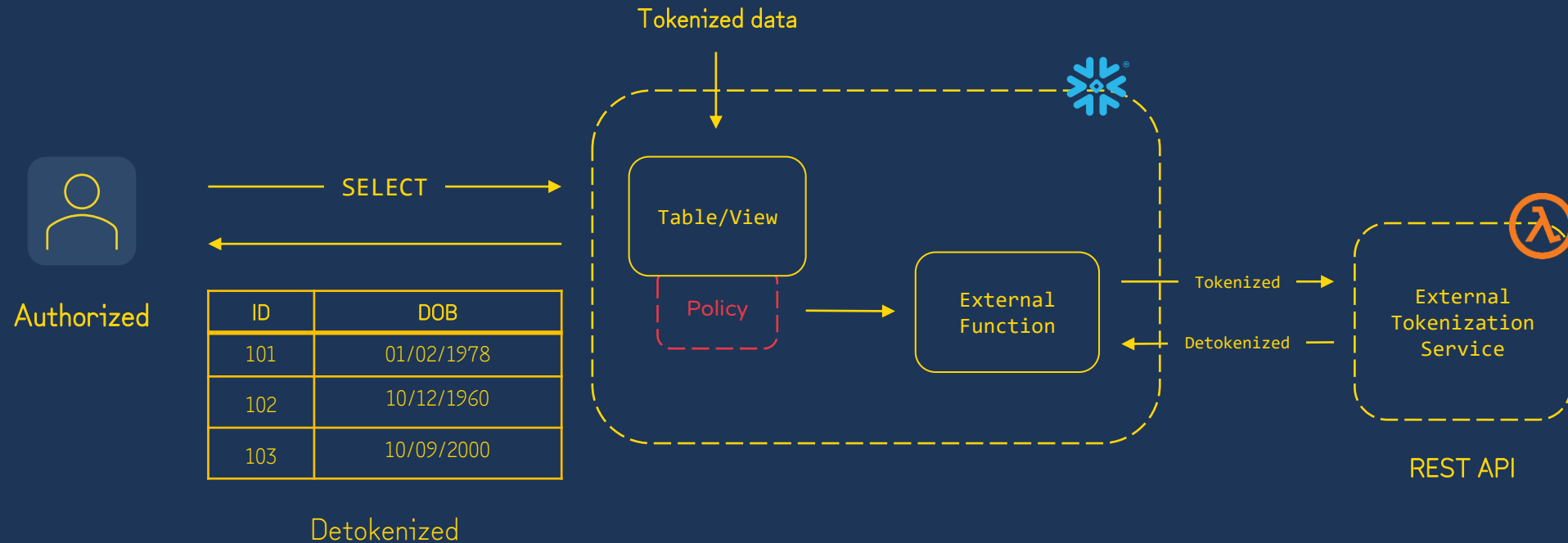
4

A masking policy is applied no matter where the column is referenced in a SQL statement.

5

A data masking policy can be applied either when the object is created or after the object is created.

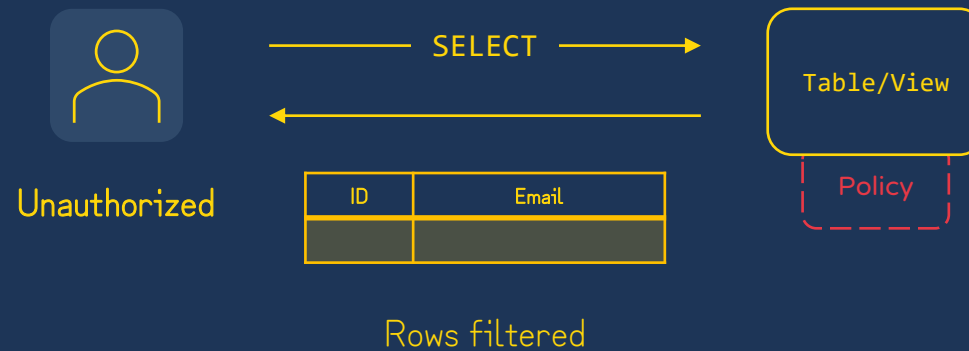
# External Tokenization



Tokenized data is loaded into Snowflake, which is detokenized at query run-time for authorized users via masking policies that call an external tokenization service using external functions.

# Row Level security

# Row Access Policies



Row access policies enable a security team to restrict which rows are return in a query.



# Row Access Policies



Row access policies enable a security team to restrict which rows are return in a query.

# Row Access Policies

```
CREATE OR REPLACE ROW ACCESS POLICY RAP_ID AS (ACC_ID VARCHAR) RETURNS BOOLEAN ->  
CASE  
    WHEN 'ADMIN' = CURRENT_ROLE() THEN TRUE  
    ELSE FALSE  
END;
```

```
ALTER TABLE ACCOUNTS ADD ROW ACCESS POLICY RAP_IT ON (ACC_ID);
```

## Similarities

Schema level object

Segregation of duties

Creation and applying workflow

Nesting policies



Adding a masking policy to a column fails if the column is referenced by a row access policy.



Row access policies are evaluated before data masking policies.

# Secure Views

# Secure Views

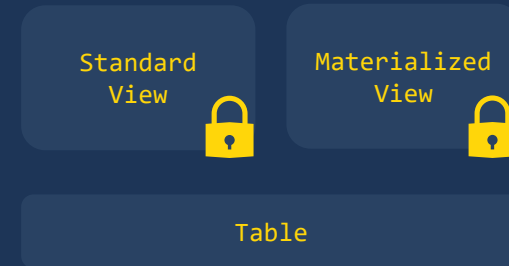
Secure views are a type of view designed to limit access to the underlying tables or internal structural details of a view.

Both standard and materialized views can be designated as secure.

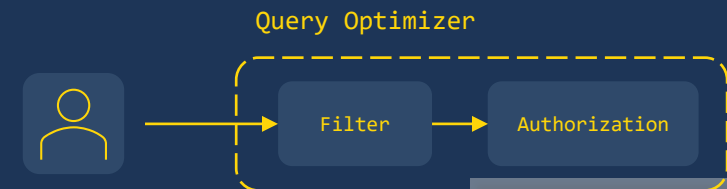
A secure view is created by adding the keyword `SECURE` in the view DDL.

The definition of a secure view is only available to the object owner.

Secure views bypass query optimizations which may inadvertently expose data in the underlying table.



```
CREATE OR REPLACE SECURE VIEW MY_SEC_VIEW AS  
SELECT COL1, COL2, COL3 FROM MY_TABLE;
```



# Account Usage and Information Schema

# Account Usage

Snowflake provide a shared read-only databased called SNOWFLAKE, imported using a Share object called ACCOUNT\_USAGE.

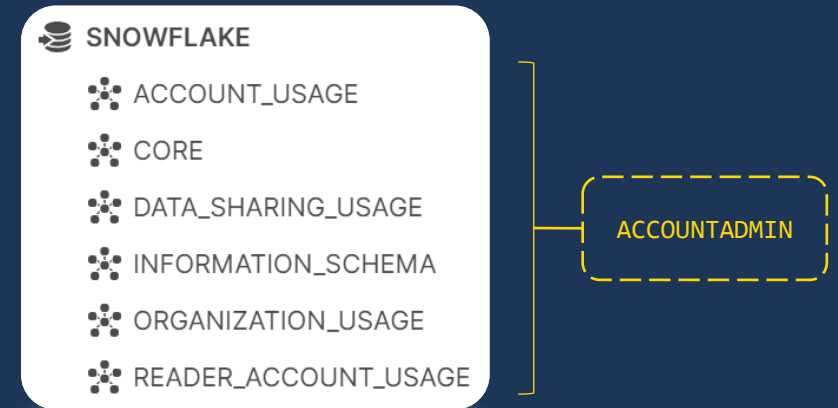
It is comprised of 6 schemas, which contain many views providing fine-grained usage metrics at the account and object level.

By default, only users with the ACCOUNTADMIN role can access the SNOWFLAKE database.

Account usage views record dropped objects, not just those that are currently active.

There is latency between an event and when that event is recorded in an account usage view.

Certain account usage views provide historical usage metrics. The retention period for these views is 1 year.



```
SELECT * FROM "SNOWFLAKE"."ACCOUNT_USAGE"."TABLES";
```

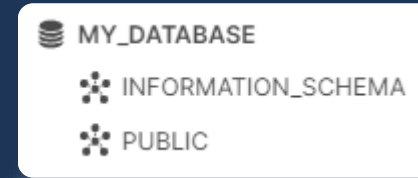
TABLE_ID	DELETED
4	2022-12-03 09:08:35.765 -0800

~ 2 Hours

365 Days

# Information Schema

Each database created in an account automatically includes a built-in, read-only schema named INFORMATION\_SCHEMA based on the SQL-92 ANSI Information Schema.



Each INFORMATION\_SCHEMA contains:

- Views displaying metadata for all objects contained in the database.
- Views displaying metadata for account-level objects (non-database objects such as roles, warehouses and databases).
- Table functions displaying metadata for historical and usage data across an account.

## Object Metadata Views

Tables  
Stages  
Pipes  
Functions  
...

## Account Metadata Views

Databases  
Load History  
Enabled Roles  
Applicable Roles  
...

## Table Functions

Task History  
Login History  
Copy History  
Tag References  
...

The output of a view or table function depends on the privileges granted to the user's current role.

# Account Usage vs. Information Schema

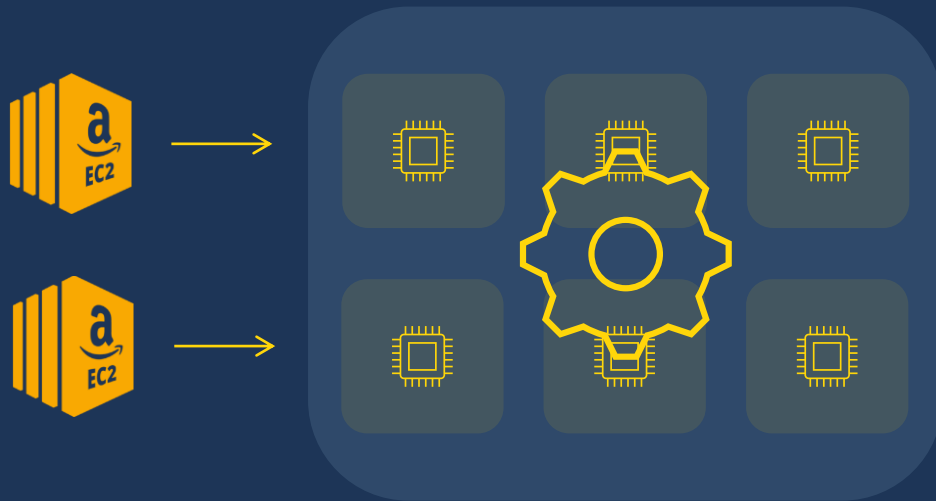
Property	Account Usage	Information Schema
Includes dropped objects	Yes	No
Latency of data	From 45 minutes to 3 hours (varies by view)	None
Retention of historical data	1 Year	From 7 days to 6 months (varies by view/table function)



# What is a Virtual Warehouse?

# Virtual Warehouse Overview

A Virtual Warehouse is a named abstraction for a Massively Parallel Processing (MPP) compute cluster.



Virtual Warehouses execute:

- DQL operations (SELECT)
- DML operations (UPDATE)
- Data loading operations (COPY INTO)

As a user you only interact with the named warehouse object not the underlying compute resources.

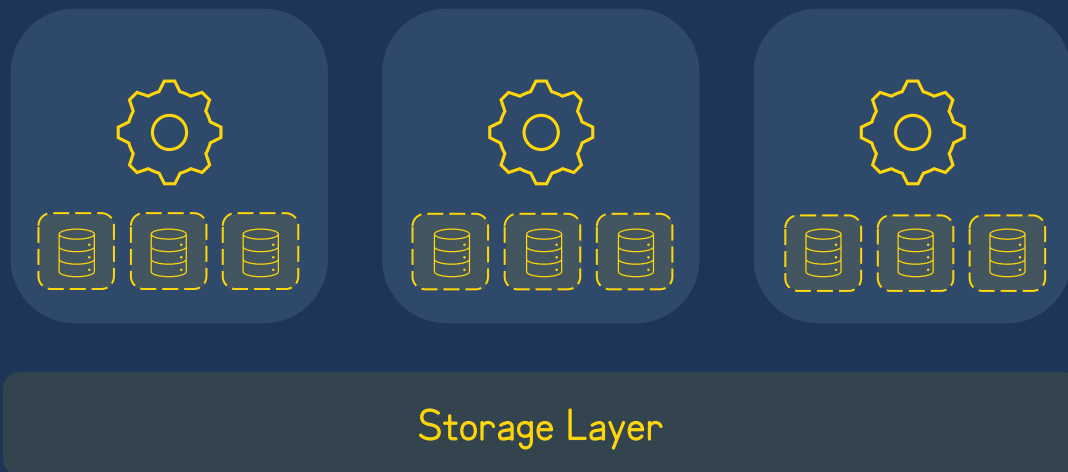
# Virtual Warehouse Overview

Spin up and shut-down a virtually unlimited number of warehouses without resource contention.

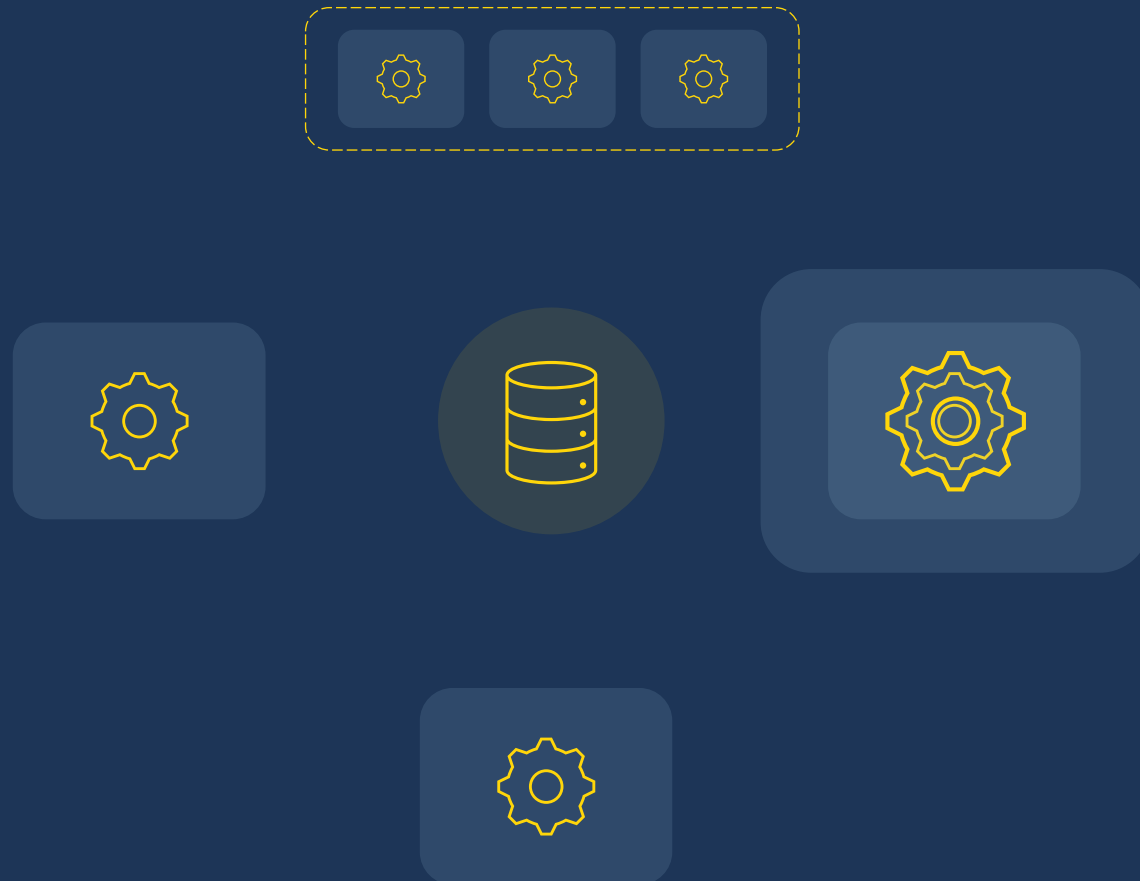
Virtual Warehouse configuration can be changed on-the-fly.

Virtual Warehouses contain local SSD storage used to store raw data retrieved from the storage layer.

Virtual Warehouses are created via the Snowflake UI or through SQL commands.



# Virtual Warehouse Overview



```
DROP WAREHOUSE MY_WAREHOUSE;
```

```
CREATE WAREHOUSE MY_MED_WH  
WAREHOUSE_SIZE='MEDIUM';
```

```
ALTER WAREHOUSE MY_WH SUSPEND;
```

```
ALTER WAREHOUSE MY_WH_2 SET  
WAREHOUSE_SIZE=MEDIUM;
```

```
CREATE WAREHOUSE MY_WH_3  
MIN_CLUSTER_COUNT=1  
MAX_CLUSTER_COUNT=3  
SCALING_POLICY=STANDARD;
```

# Sizing and Billing

# Virtual Warehouse Sizes



Virtual Warehouses can be created in 10 t-shirt sizes.

Underlying compute power approximately doubles with each size.

In general the larger the Virtual Warehouse the better the query performance.

Choosing a size is typically done by experimenting with a representative query of a workload.

Data loading does not typically require large Virtual Warehouses and sizing up does not guarantee increased data loading performance.

# Virtual Warehouse Billing

Virtual Warehouse Size										
X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large	5X-Large	6X-Large	
1	2	4	8	16	32	64	128	256	512	Credits / Hour
0.0003	0.0006	0.0011	0.0022	0.0044	0.0089	0.0178	0.0356	0.0711	0.1422	Credits / Second

# Credit Calculation



Running Time



Credits

0-60 Seconds

0.017

61 Seconds

0.017

2 Minutes

0.033

10 Minutes

0.167

1 Hour

1.000

The first 60 seconds after a virtual warehouse is provisioned and running are always charged.



# Credit Pricing

Table 2: On Demand Credit Pricing					
Cloud Provider	Region	Snowflake Service Edition			
		Standard	Enterprise	Business Critical	VPS
AWS	Europe (London)	\$2.70	\$4.00	\$5.40	\$8.10
AWS	AP Northeast 1 (Tokyo)	\$2.85	\$4.30	\$5.70	\$8.55
Azure	North Europe (Ireland)	\$2.60	\$3.90	\$5.20	\$7.80
GCP	Europe West 2 (London)	\$2.70	\$4.00	\$5.40	\$8.10

① Credit price is determined by **region** & **Snowflake edition**.

# Credit Pricing

Table 2: On Demand Credit Pricing					
Cloud Provider	Region	Snowflake Service Edition			
		Standard	Enterprise	Business Critical	VPS
AWS	Europe (London)	\$2.70	\$4.00	\$5.40	\$8.10
AWS	AP Northeast 1 (Tokyo)	\$2.85	\$4.30	\$5.70	\$8.55
Azure	North Europe (Ireland)	\$2.60	\$3.90	\$5.20	\$7.80
GCP	Europe West 2 (London)	\$2.70	\$4.00	\$5.40	\$8.10

## Virtual Warehouse

If a XS Virtual Warehouse is active for 1 hour on the Standard Edition of Snowflake deployed in AWS Europe (London) Region it will consume 1 Snowflake Credit costing \$2.70 (Nov 2021)

If a L Virtual Warehouse is active for 3 hours on the Enterprise Edition of Snowflake deployed in AWS AP Northeast 1 (Tokyo) Region it will consume 24 Snowflake Credit costing \$72.00 (Nov 2021)

# Virtual Warehouse State and Properties

# Virtual Warehouse State



STARTED



SUSPENDED



RESIZING

# Virtual Warehouse State

```
CREATE WAREHOUSE MY_MED_WH WITH  
WAREHOUSE_SIZE='MEDIUM';
```

By default when a Virtual Warehouse is created it is in the STARTED state.

```
ALTER WAREHOUSE MY_WH SUSPEND;
```

Suspending a Virtual Warehouse puts it in the SUSPENDED state, removing the compute nodes from a warehouse.

```
ALTER WAREHOUSE MY_WH RESUME;
```

Resuming a Virtual Warehouse puts it back into the STARTED state and can execute queries.

# Virtual Warehouse State Properties

## AUTO SUSPEND

```
CREATE WAREHOUSE MY_MED_WH  
AUTO_SUSPEND=300;
```

Specifies the number of seconds of inactivity after which a warehouse is automatically suspended.

## AUTO RESUME

```
CREATE WAREHOUSE MY_MED_WH  
AUTO_RESUME=TRUE;
```

Specifies whether to automatically resume a warehouse when a SQL statement is submitted to it.

## INITIALLY SUSPENDED

```
CREATE WAREHOUSE MY_MED_WH  
INITIALLY_SUSPENDED=TRUE;
```

Specifies whether the warehouse is created initially in the 'Suspended' state.

# Resource Monitors

# Resource Monitors

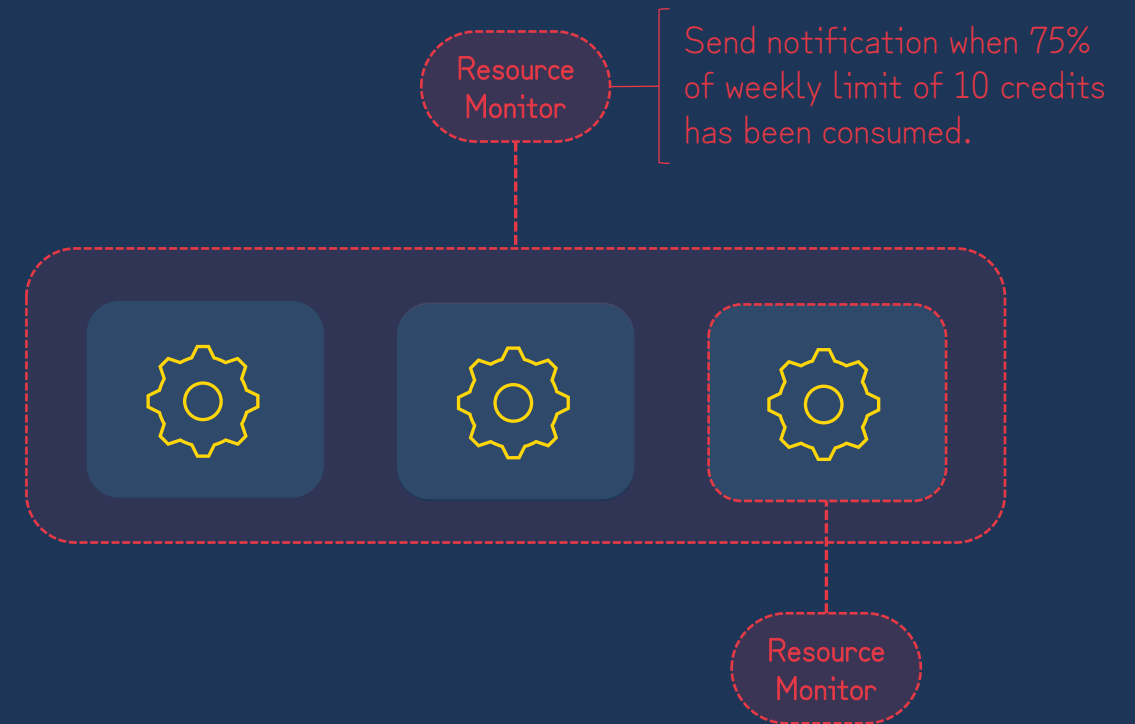
Resource Monitors are objects allowing users to set credit limits on user managed warehouses.

Resource Monitors can be set on either the account or individual warehouse level.

Limits can be set for a specified interval or data range.

When limits are reached an action can be triggered, such as notify user or suspend warehouse.

Resource Monitors can only be created by account administrators.





# Resource Monitors

```
CREATE RESOURCE MONITOR ANALYSIS_RM  
WITH CREDIT_QUOTA=100  
FREQUENCY=MONTHLY  
START_TIMESTAMP='2023-01-04 00:00 GMT'  
TRIGGERS ON 50 PERCENT DO NOTIFY  
ON 75 PERCENT DO NOTIFY  
ON 95 PERCENT DO SUSPEND  
ON 100 PERCENT DO SUSPEND_IMMEDIATE;
```

①

Number of credits allocated to the resource monitor per frequency interval.

②

DAILY, WEEKLY, MONTHLY, YEARLY or NEVER.

③

Start timestamp determines when a resource monitor will start once applied to a warehouse or account. The frequency is relative to the start timestamp.

④

Triggers determine the condition for a certain action to take place.

```
ALTER ACCOUNT SET RESOURCE_MONITOR = ANALYSIS_RM;
```

# Virtual Warehouse Concurrency & Query Complexity

# Scaling Up: Resizing Virtual Warehouses



X-Small

Scaling up a Virtual Warehouse is intended to improve query performance.

Virtual Warehouses can be manually resized via the Snowflake UI or SQL commands.

Resizing a running warehouse does not impact running queries. The additional compute resources are used for queued and new queries.

Large

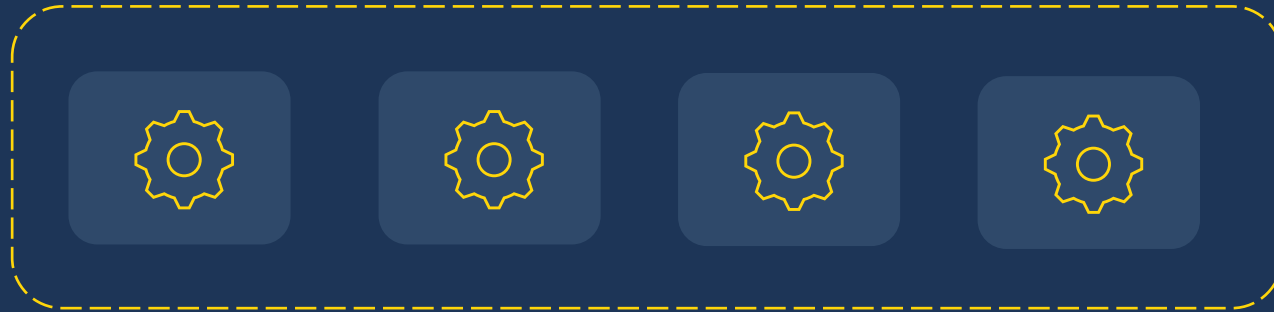
```
ALTER WAREHOUSE MY_WH  
SET WAREHOUSE_SIZE=LARGE;
```

Decreasing size of running warehouse removes compute resources from the warehouse and clears the warehouse cache.

# Scaling out: Multi-cluster Warehouses

MY\_MCW\_1

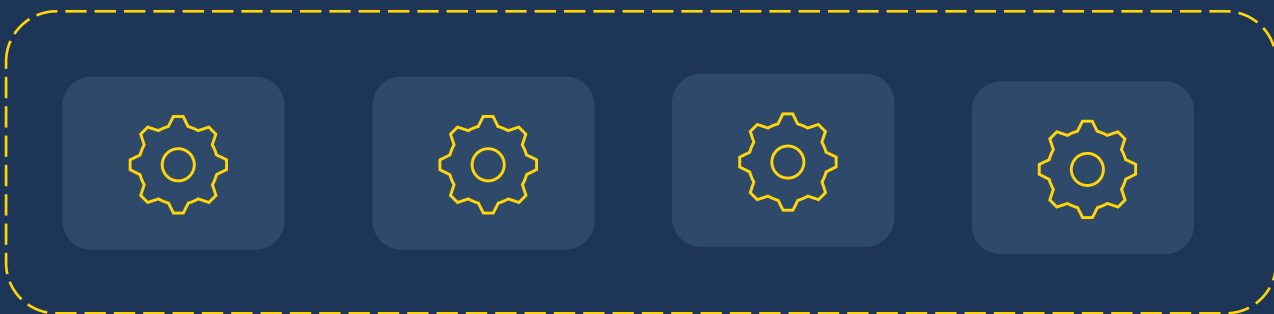
Maximized Mode



MIN\_CLUSTER\_COUNT=4 & MAX\_CLUSTER\_COUNT=4

MY\_MCW\_2

Auto-scale Mode



MIN\_CLUSTER\_COUNT=1 & MAX\_CLUSTER\_COUNT=4

A multi-cluster warehouse is a named group of virtual warehouses which can automatically scale in and out based on the number of concurrent users/queries.

**MIN\_CLUSTER\_COUNT** specifies the minimum number of warehouses for a multi-cluster warehouse.

**MAX\_CLUSTER\_COUNT** specifies the maximum number of warehouses for a multi-cluster warehouse.

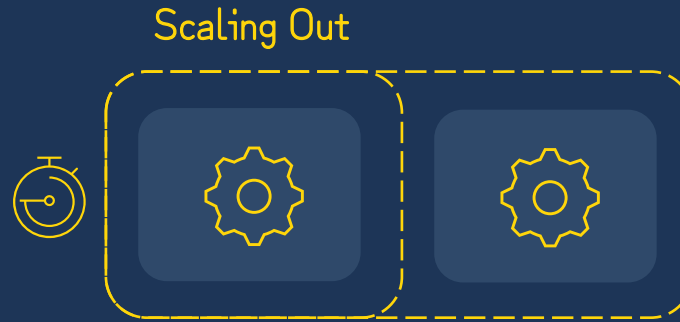
Setting these two values the same will put the multi-cluster warehouse in **MAXIMIZED** mode.

Setting these two values differently will put the multi-cluster warehouse in **AUTO-SCALE** mode.

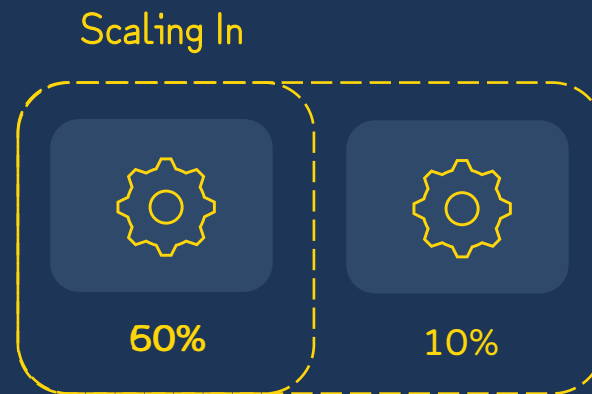
# Standard Scaling Policy

## Standard Scaling Policy

```
CREATE WAREHOUSE MY_MCW_1  
MIN_CLUSTER_COUNT=1  
MAX_CLUSTER_COUNT=4  
SCALING_POLICY=STANDARD;
```



When a query is queued a new warehouse will be added to the group immediately.



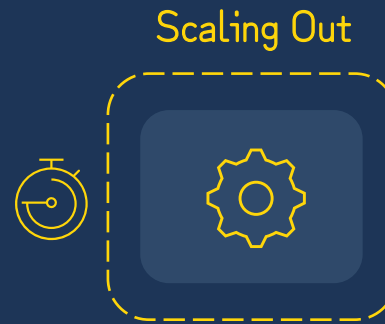
Every minute a background process will check if the load on the least busy warehouse can be redistributed to another warehouse.

If this condition is met after 2 consecutive minutes a warehouse will be marked for shutdown.

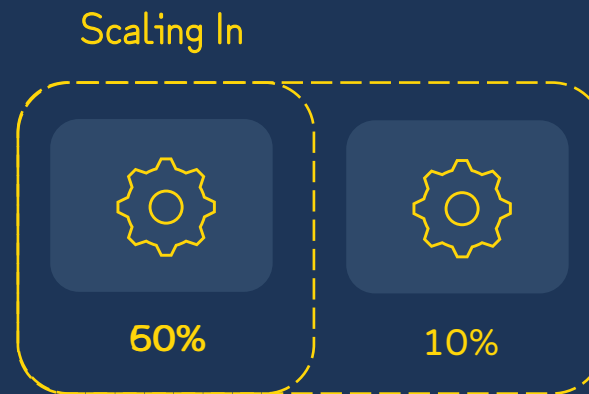
# Economy Scaling policy

## Economy Scaling Policy

```
CREATE WAREHOUSE MY_MCW_2  
MIN_CLUSTER_COUNT=1  
MAX_CLUSTER_COUNT=4  
SCALING_POLICY=ECONOMY;
```



When a query is queued the system will estimate if there's enough query load to keep a new warehouse busy for 6 minutes.



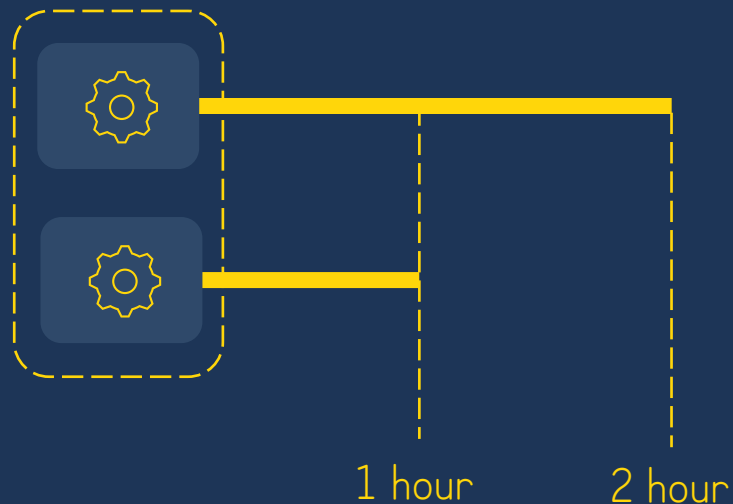
Every minute a background process will check if the load on the least busy warehouse can be redistributed to another warehouse.

If this condition is met after 6 consecutive minutes a warehouse will be marked for shutdown.

# Multi-cluster Warehousing Billing

```
CREATE WAREHOUSE MY_MCW  
MIN_CLUSTER_COUNT=1  
MAX_CLUSTER_COUNT=3  
WAREHOUSE_SIZE='MEDIUM';
```

3 x 4 Credits = 12 Credits / Hour



The total credit cost of a multi-cluster warehouse is the sum of all the individual running warehouses that make up that cluster.

The maximum number of credits a multi-cluster can consume is the number of warehouses multiplied by the hourly credit rate of the size of the warehouses.

Because multi-cluster warehouses scale in and out based on demand it's typical to get some fraction of the maximum credit consumption.

# Concurrency Behaviour Properties

## MAX CONCURRENCY LEVEL

```
CREATE WAREHOUSE MY_MED_WH  
MAX_CONCURRENCY_LEVEL=6;
```

Specifies the number of concurrent SQL statements that can be executed against a warehouse before either it is queued or additional compute power is provided.

## STATEMENT QUEUED TIMEOUT IN SECONDS

```
CREATE WAREHOUSE MY_MED_WH  
STATEMENT_QUEUED_TIMEOUT_IN_SECONDS=60;
```

Specifies the time, in seconds, a SQL statement can be queued on a warehouse before it is aborted.

## STATEMENT TIMEOUT IN SECONDS

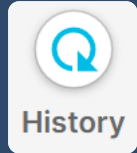
```
CREATE WAREHOUSE MY_MED_WH  
STATEMENT_TIMEOUT_IN_SECONDS=600;
```

It specifies the time, in seconds, after which any running SQL statement on a warehouse is aborted.



# Performance and Tuning Overview

# Query Performance Analysis Tools



History Tab



Query History

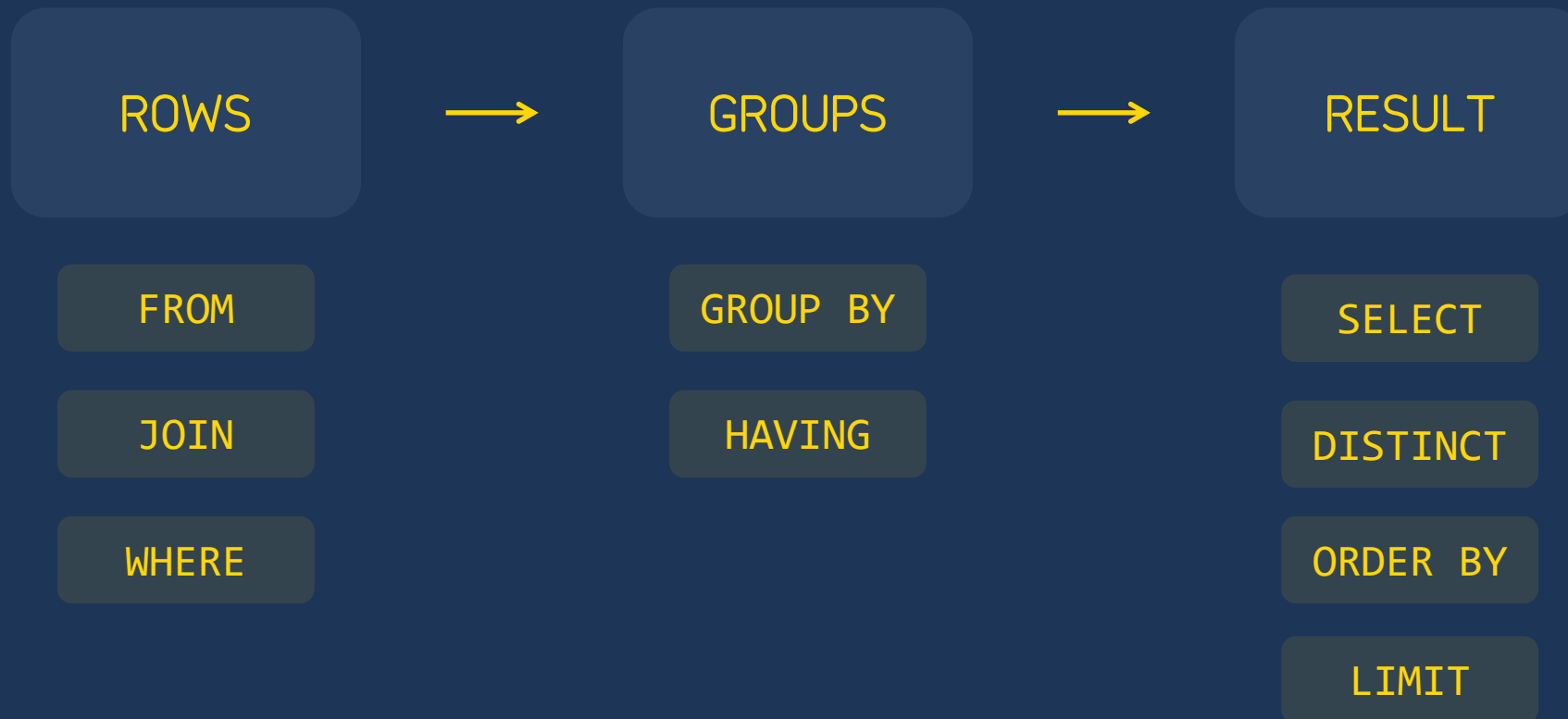
SQL

Query Profile

- ① History Tab displays query history for the last 14 days.
- ② Users can view other users queries but cannot view their query results.

# SQL Tuning

# Database Order Of Execution



# Join Explosion

Orders

ORDER_DATE	PRODUCT_NAME	CUSTOMER_NAME	ORDER_AMOUNT
01/12/2022	Apple MacBook Air	Arun	1
13/12/2021	Sony Playstation 5	Ana	1
01/01/2022	LG Blu-ray Player	Pawel	2
21/02/2020	Sony Playstation 5	Susan	1

Products

PRODUCT_NAME	PRODUCT_PRICE	ORDER_DATE
Apple MacBook Air	899.99	01/12/2022
LG Blu-ray Player	110.00	01/01/2022
Sony Playstation 5	449.00	12/11/2020
Sony Playstation 5	429.00	10/06/2021

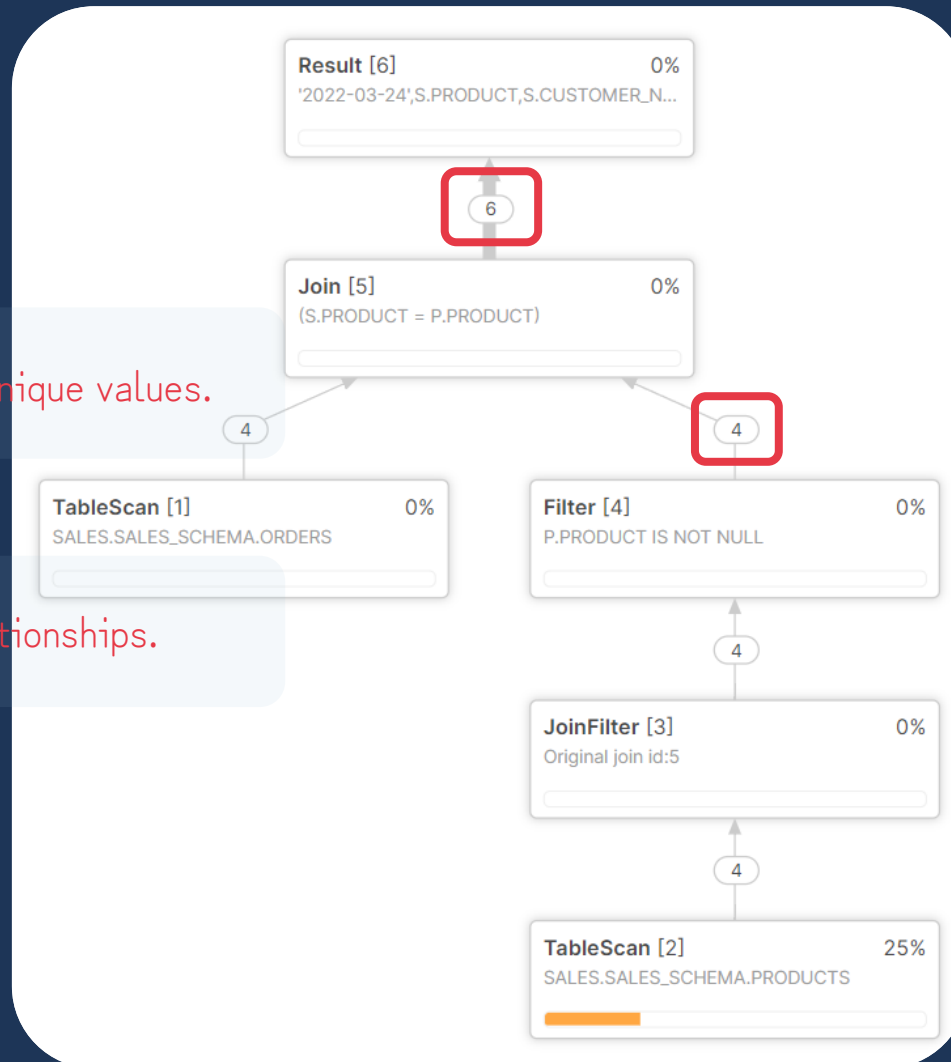
```
SELECT *, (O.ORDER_AMOUNT * P.PRODUCT_PRICE)
FROM ORDERS O
LEFT JOIN PRODUCTS P ON O.PRODUCT = P.PRODUCT;
```

ORDER_DATE	PRODUCT_NAME	CUSTOMER_NAME	ORDER_AMOUNT	ORDER_TOTAL
01/12/2022	Apple MacBook Air	Arun	1	899.99
13/12/2021	Sony Playstation 5	Ana	1	449.00
01/01/2022	LG Blu-ray Player	Pawel	2	220.00
21/02/2020	Sony Playstation 5	Susan	1	449.00
21/02/2020	Sony Playstation 5	Susan	1	429.00
13/12/2021	Sony Playstation 5	Ana	1	429.00

# Join Explosion

① Join on columns with unique values.

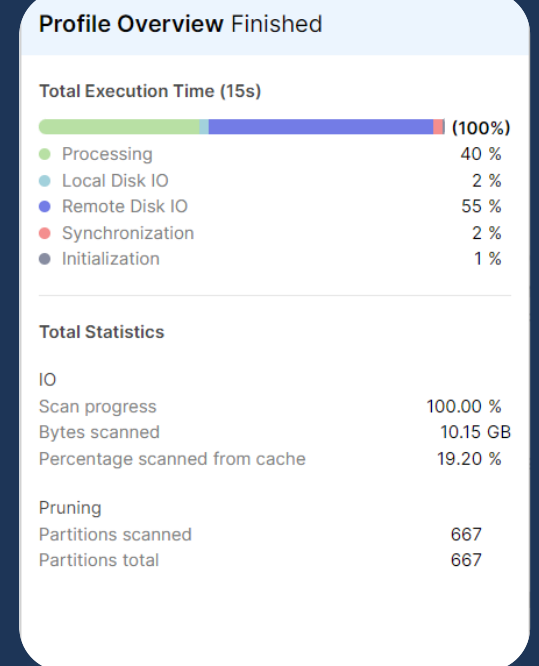
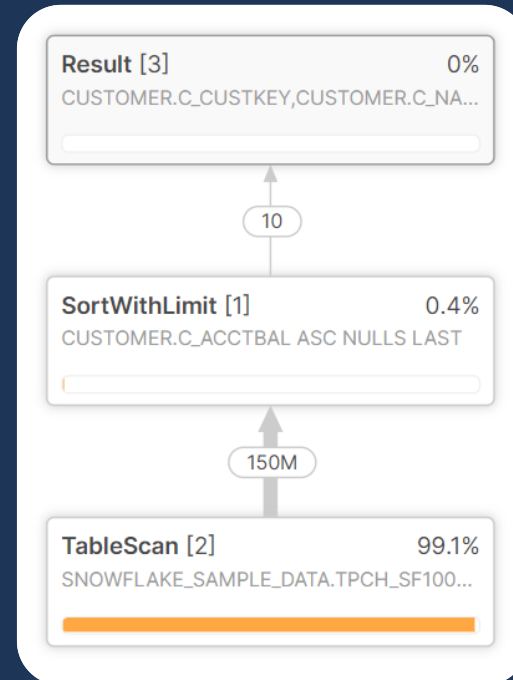
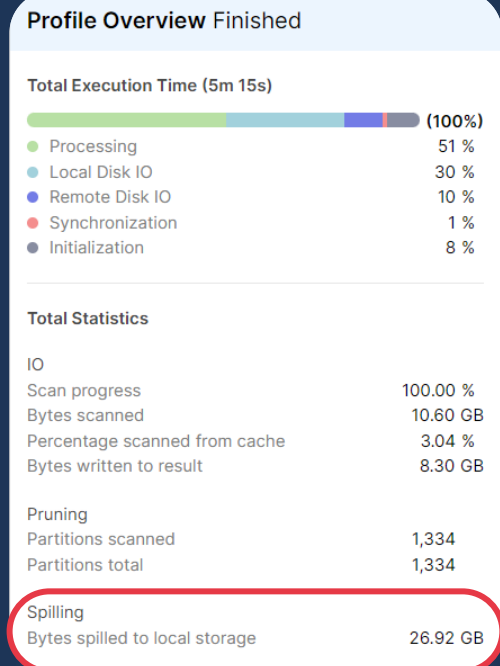
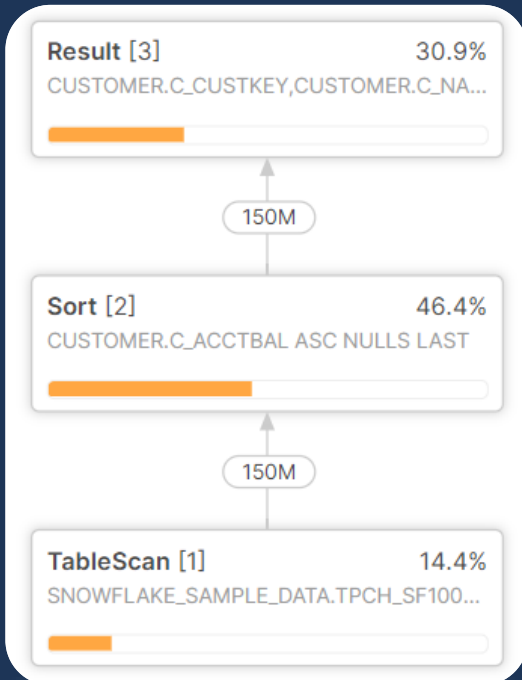
② Understand table relationships.



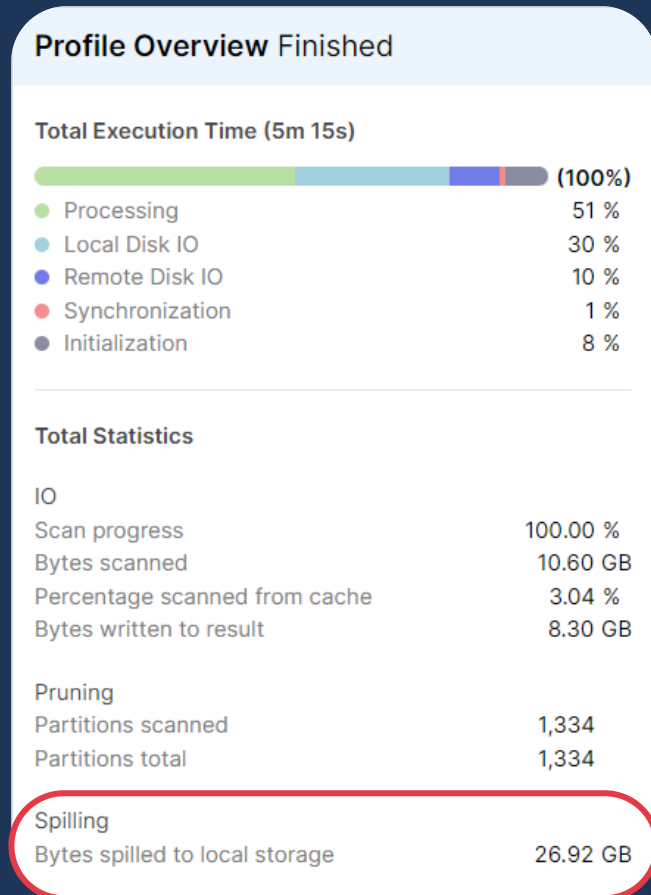
# Limit & Order By

```
SELECT * FROM CUSTOMER
ORDER BY C_ACCTBAL;
```

```
SELECT * FROM CUSTOMER
ORDER BY C_ACCTBAL
LIMIT 10;
```



# Spilling to Disk



## “Bytes spilled to local storage”

Volume of data spilled to virtual warehouse local disk.

## “Bytes spilled to remote storage”

Volume of data spilled to remote disk.

1 Process less data.

2 Virtual Warehouse size.



# Order By Position

```
SELECT C_NATIONKEY, R.R_NAME, TOTAL_BAL FROM
(
  SELECT
    C_NATIONKEY,
    COUNT(C_ACCTBAL) AS TOTAL_BAL
  FROM CUSTOMER
  GROUP BY C_NATIONKEY
  ORDER BY C_NATIONKEY
) C JOIN REGION R ON (C.C_NATIONKEY = R.R_REGIONKEY)
ORDER BY TOTAL_BAL;
```

Redundant



ORDER BY C\_NATIONKEY

Top-level



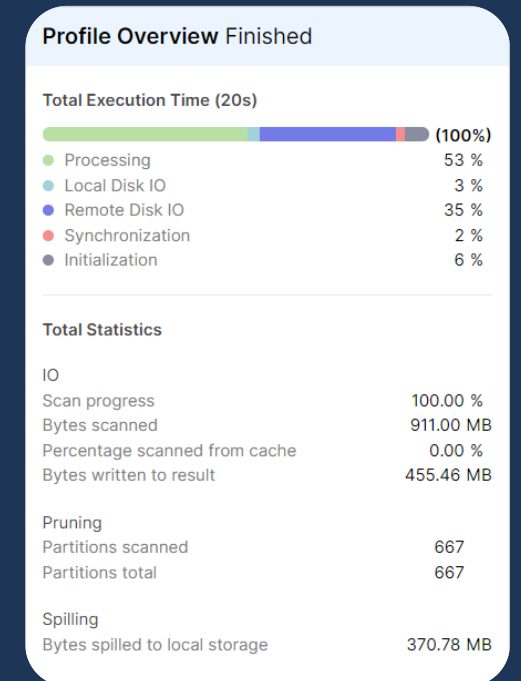
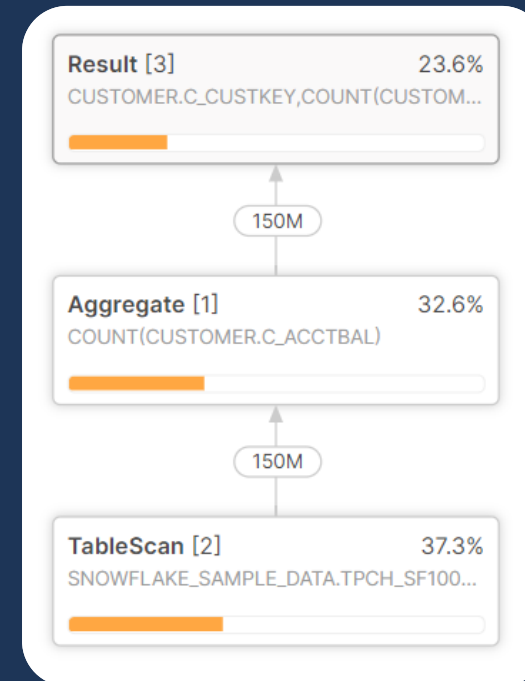
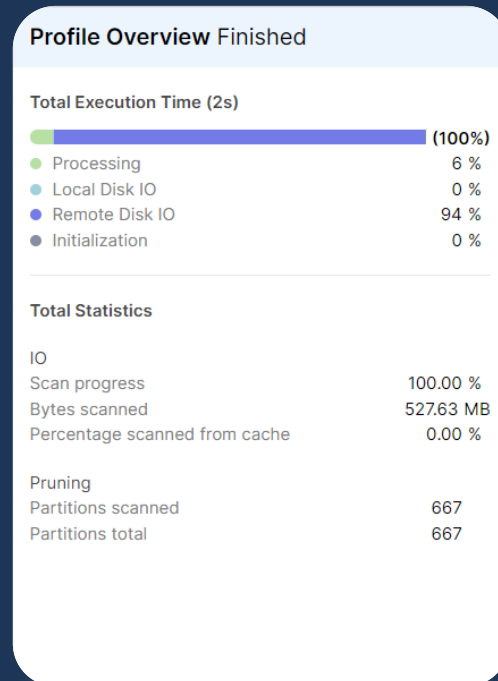
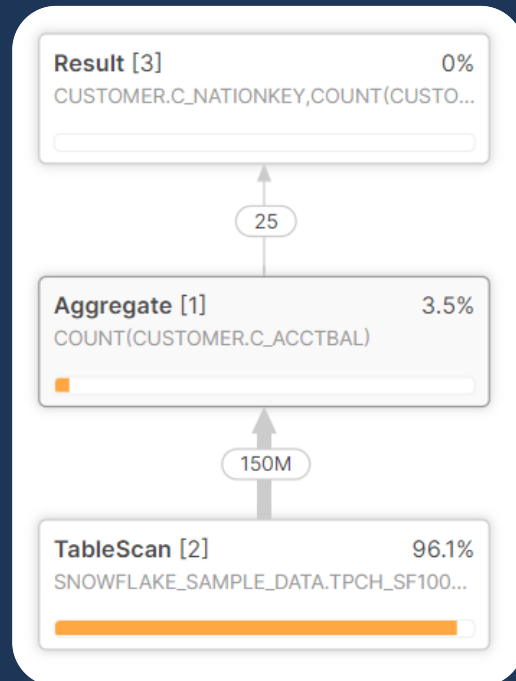
ORDER BY TOTAL\_BAL;

① ORDER BY in top-level select only.

# Group By

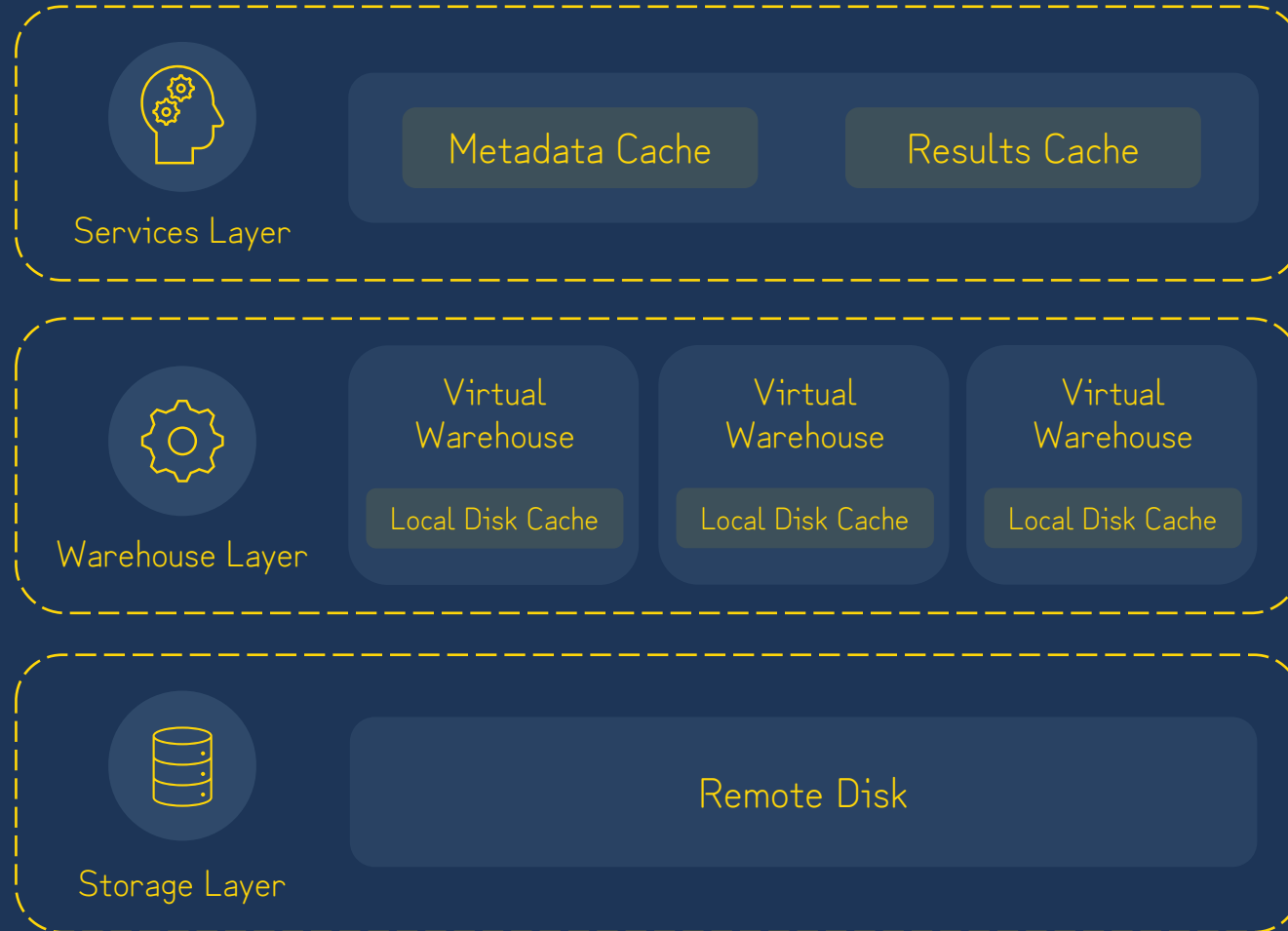
```
SELECT C_NATIONKEY, COUNT(C_ACCTBAL)
FROM CUSTOMER
GROUP BY C_NATIONKEY; -- Low Cardinality
```

```
SELECT C_CUSTKEY, COUNT(C_ACCTBAL)
FROM CUSTOMER
GROUP BY C_CUSTKEY; -- High Cardinality
```

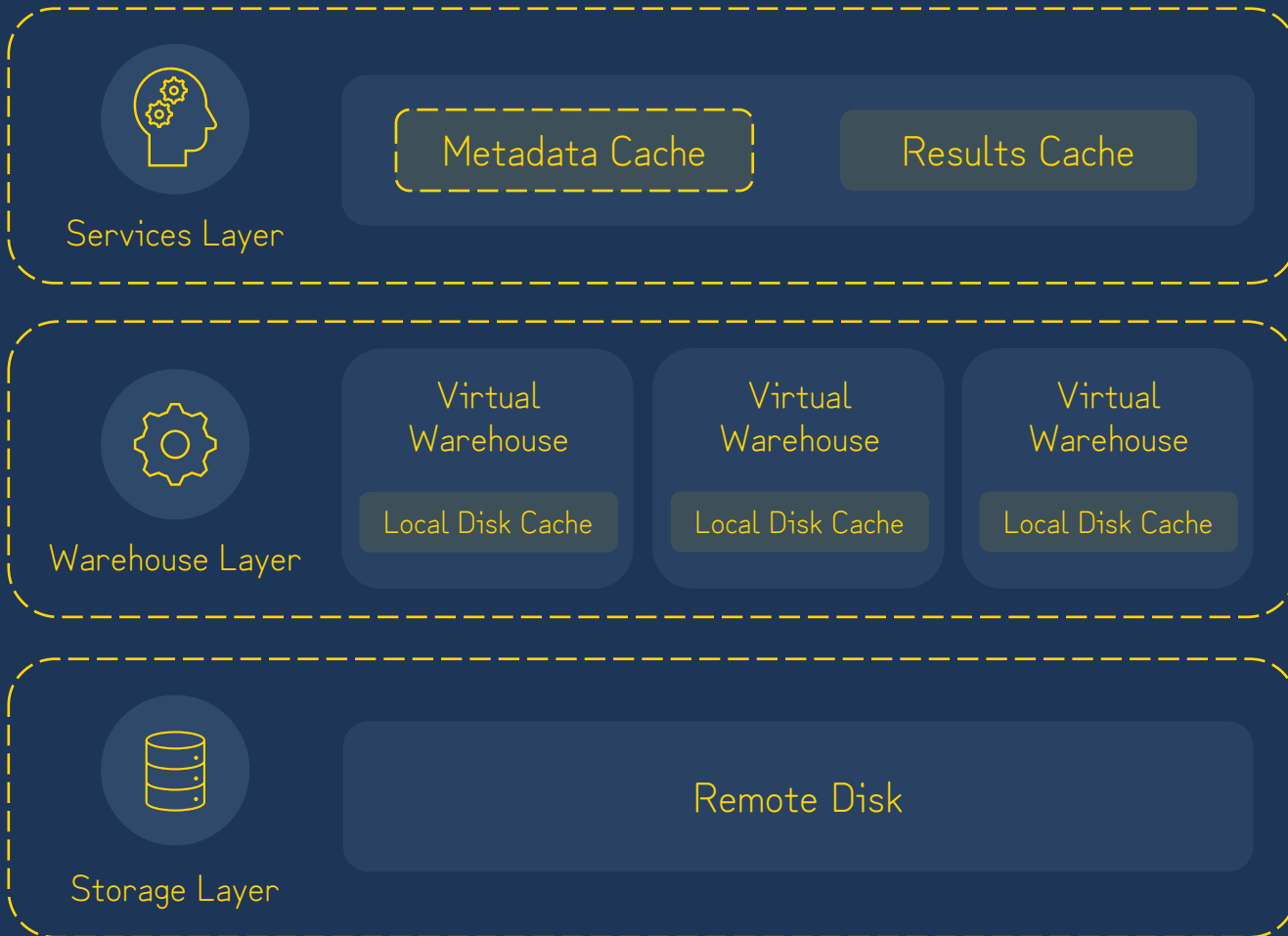


# Caching

# Caching



# Caching



## Metadata Cache

Snowflake has a high availability metadata store which maintains metadata object information and statistics.

Some queries can be completed purely using this metadata, not requiring a running virtual warehouse.

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

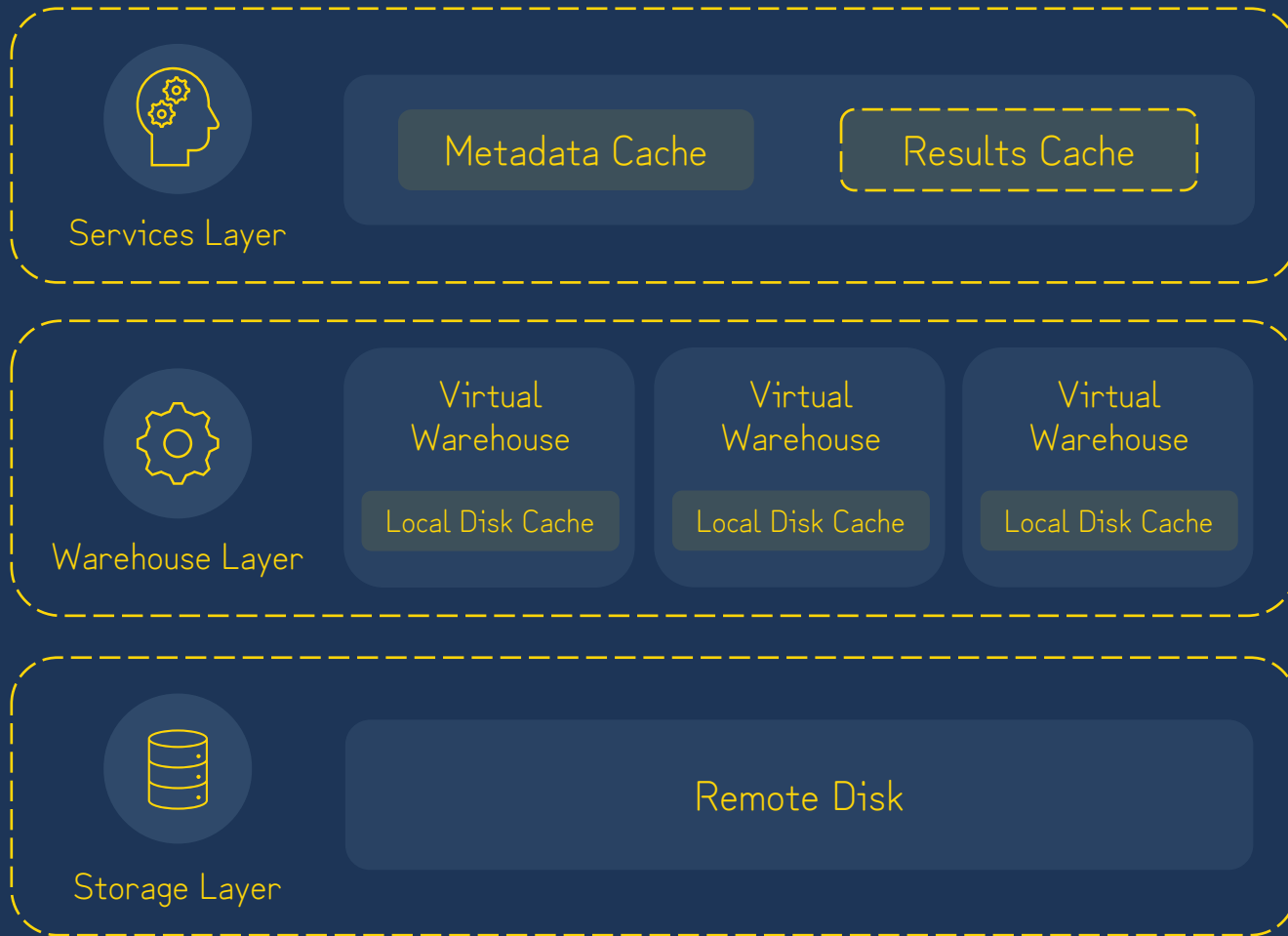
```
SELECT SYSTEM$WHITELIST();
```

```
SELECT CURRENT_DATABASE();
```

```
DESCRIBE TABLE MY_TABLE;
```

```
SHOW TABLES;
```

# Caching



## Result Cache

24hr

31 Days

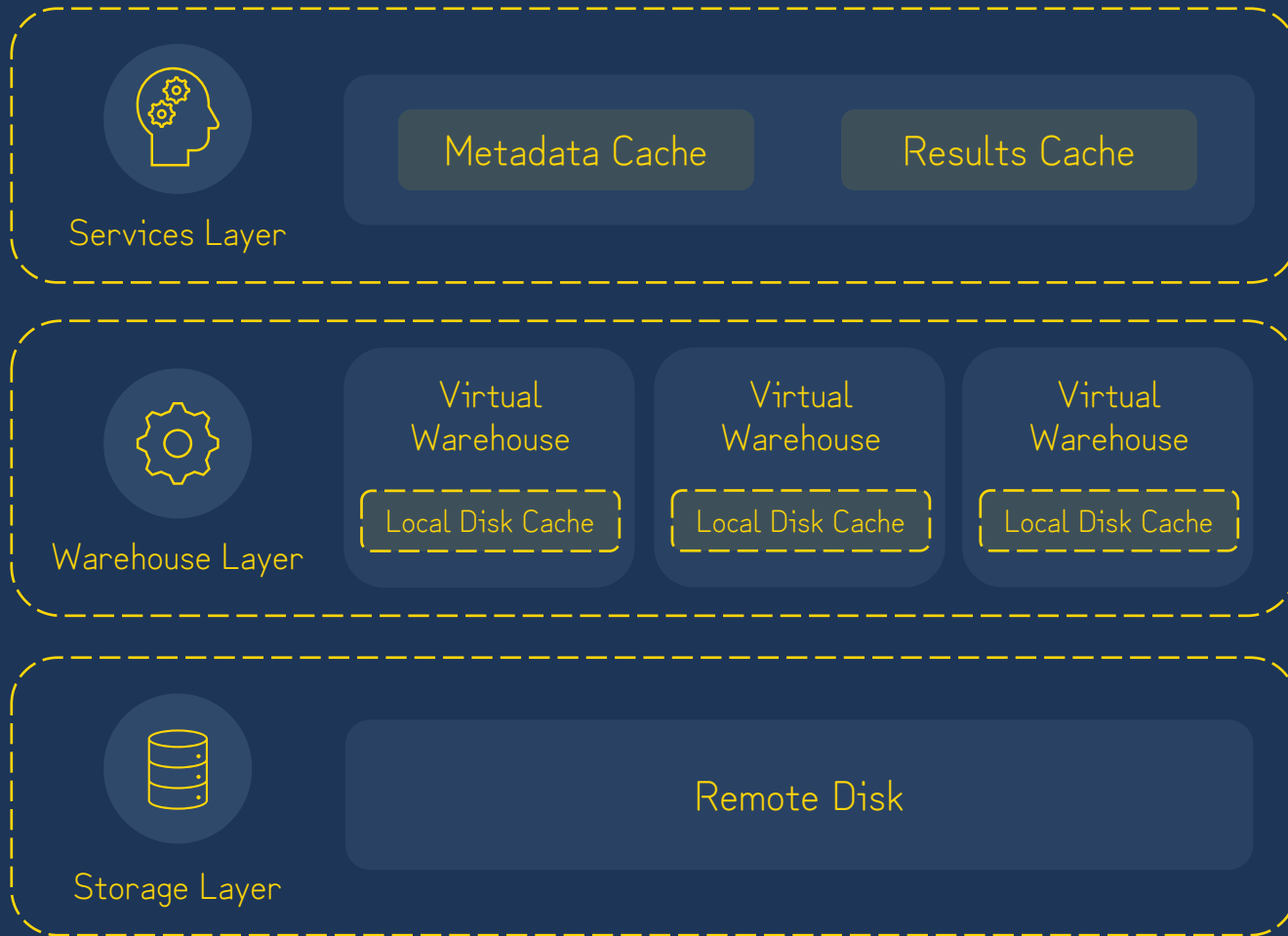
To reuse a result:

- New query exactly matches previous query.
- The underlying table data has not changed.
- The same role is used as the previous query.

If time context functions are used, such as **CURRENT\_TIME()**, the result cache will not be used.

Result reuse can be disabled using the session parameter **USE\_CACHED\_RESULT**.

# Caching



## Warehouse Cache

Virtual Warehouses have local SSD storage which maintains raw table data used for processing a query.

The larger the virtual warehouse the greater the local cache.

It is purged when the virtual warehouse is resized, suspended or dropped.

Can be used partially, retrieving the rest of the data required for a query from remote storage.

# Materialized Views



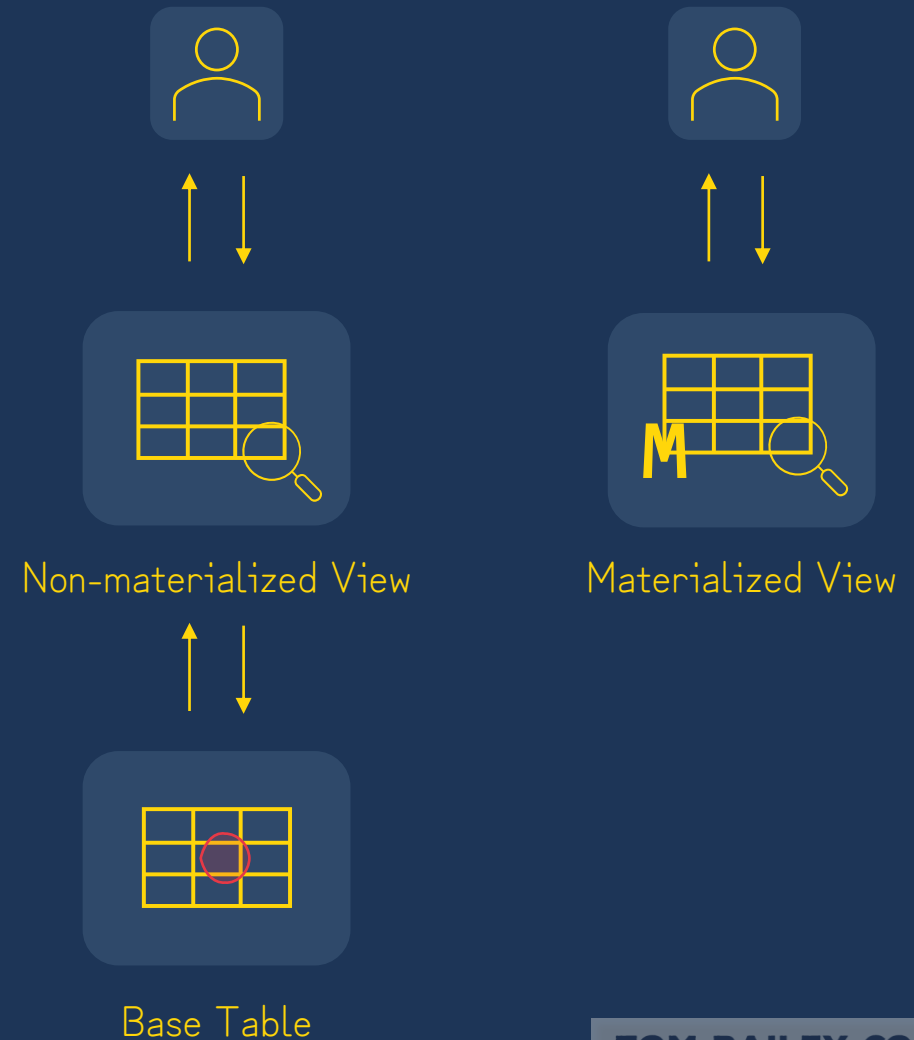
# Materialized Views

“A Materialized View is a pre-computed & persisted data set derived from a SELECT query.”

MVs are updated via a background process ensuring data is current and consistent with the base table.

MVs improve query performance by making complex queries that are commonly executed readily available.

MVs are an enterprise edition and above serverless feature.



# Materialized Views

MVs use compute resources to perform automatic background maintenance.

MVs use storage to store query results, adding to the monthly storage usage for an account.

MATERIALIZED\_VIEW\_REFRESH\_HISTORY

MVs can be created on top of External Tables to improve their query performance.

MVs are limited in the following ways:

Single  
Table

JOIN

UDF, HAVING, ORDER  
BY, LIMIT, WINDOW  
FUNCTIONS

```
CREATE OR REPLACE MATERIALIZED VIEW MV1 AS  
SELECT COL1, COL2 FROM T1;
```

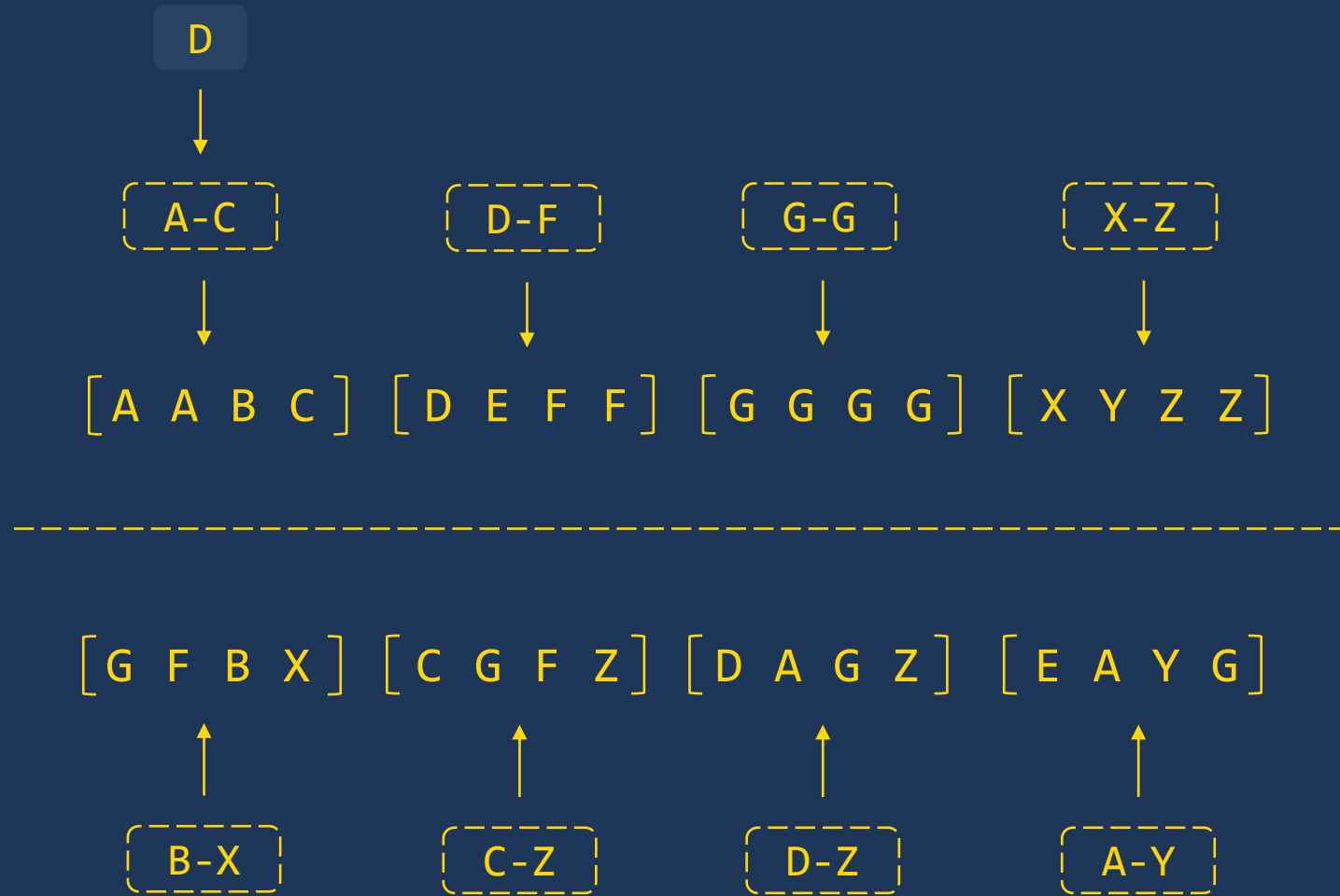
```
ALTER MATERIALIZED VIEW MV1 SUSPEND;
```

```
ALTER MATERIALIZED VIEW MV1 RESUME;
```

```
SHOW MATERIALIZED VIEWS LIKE 'MV1%';
```

# Clustering

# Natural Clustering



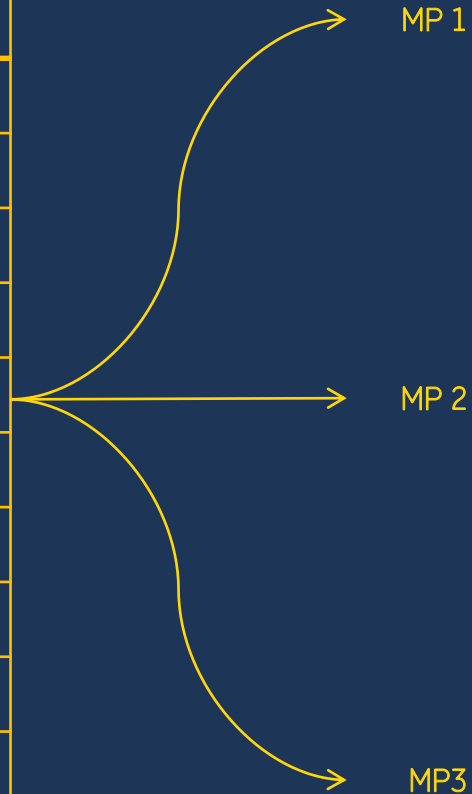
# Clustering

Products.csv

ORDER_ID	PRODUCT_ID	ORDER_DATE
1	PROD-YVN1VO	2022-06-16
2	PROD-Y5TTKB	2022-06-16
3	PROD-T9ISFR	2022-06-16
4	PROD-HK2USO	2022-06-16
5	PROD-YVN1VO	2022-06-17
6	PROD-BKMWB	2022-06-17
7	PROD-IPM6HU	2022-06-18
8	PROD-IPM6HU	2022-06-18
9	PROD-YVN1VO	2022-06-19
...	...	...

# Clustering

ORDER_ID	PRODUCT_ID	ORDER_DATE
1	PROD-YVN1VO	2022-06-16
2	PROD-Y5TTKB	2022-06-16
3	PROD-T9ISFR	2022-06-16
4	PROD-HK2USO	2022-06-16
5	PROD-YVN1VO	2022-06-17
6	PROD-BKMWB	2022-06-17
7	PROD-IPM6HU	2022-06-18
8	PROD-IPM6HU	2022-06-18
9	PROD-YVN1VO	2022-06-19
...	...	...



ORDER_ID	PRODUCT_ID	ORDER_DATE
1	PROD-YVN1VO	2022-06-16
2	PROD-Y5TTKB	2022-06-16
3	PROD-T9ISFR	2022-06-16

ORDER_ID	PRODUCT_ID	ORDER_DATE
4	PROD-HK2USO	2022-06-16
5	PROD-YVN1VO	2022-06-17
6	PROD-BKMWB	2022-06-17

ORDER_ID	PRODUCT_ID	ORDER_DATE
7	PROD-IPM6HU	2022-06-18
8	PROD-IPM6HU	2022-06-18
9	PROD-YVN1VO	2022-06-19

7

# Clustering Metadata

Snowflake maintains the following clustering metadata for micro-partitions in a table:

1

Total Number of Micro-partitions

2

Number of Overlapping Micro-partitions

3

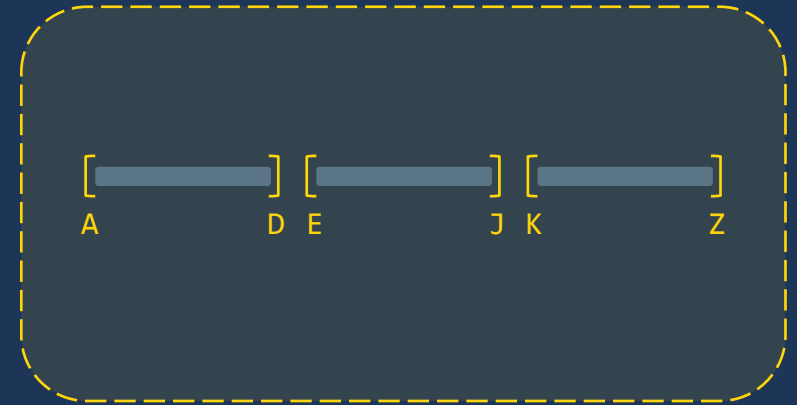
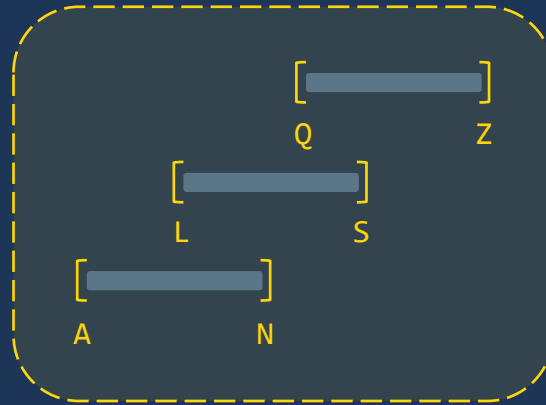
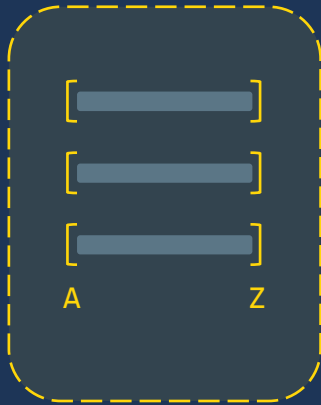
Depth of Overlapping Micro-partitions

## SYSTEM\$CLUSTERING\_INFORMATION

```
SELECT system$clustering_information('table','(col1,col3)');
```

```
"cluster_by_keys" : "(COL1, COL3)",
"total_partition_count" : 1156,
"total_constant_partition_count" : 0,
"average_overlaps" : 117.5484,
"average_depth" : 64.0701,
"partition_depth_histogram" : {
  "00000" : 0,
  "00001" : 0,
  "00002" : 3,
  "00003" : 3,
  "00004" : 4,
  "00005" : 6,
  "00006" : 3,
  "00007" : 5,
  "00008" : 10,
  "00009" : 5,
  "00010" : 7,
  "00011" : 6,
  "00012" : 8,
  "00013" : 8,
  "00014" : 9,
  "00015" : 8,
  "00016" : 6,
  "00032" : 98,
  "00064" : 269,
  "00128" : 698
}
```

# Clustering Depth



Overlapping  
Micro-partitions

3

3

0

Overlap Depth

3

2

1





# Automatic Clustering

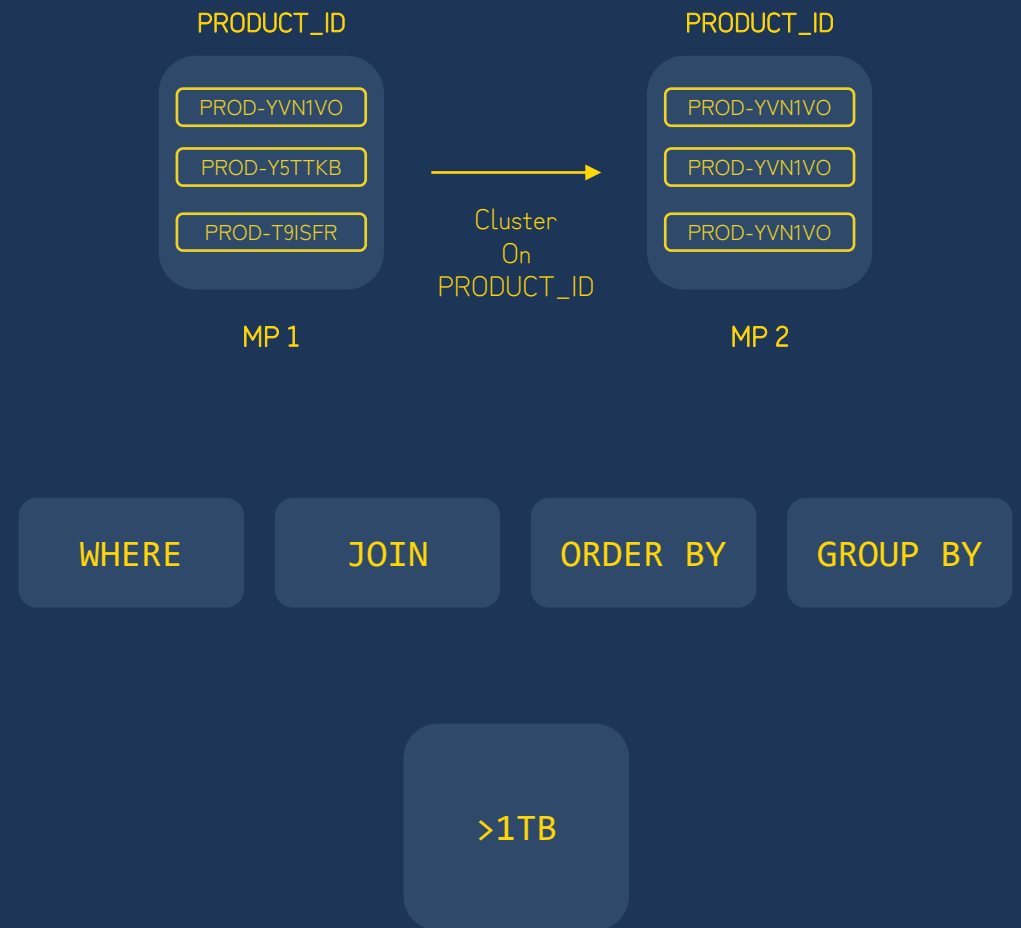
# Automatic Clustering

Snowflake supports specifying one or more table columns/expressions as a clustering key for a table.

Clustering aims to co-locate data of the clustering key in the same micro-partitions.

Clustering improves performance of queries that frequently filter or sort on the clustered keys.

Clustering should be reserved for large tables in the multi-terabyte range.



# Choosing a Clustering Key

Snowflake recommended a maximum of 3 or 4 columns (or expressions) per key.

Columns used in common queries which perform filtering and sorting operations.

Consider the cardinality of the clustering key:

HC	X	T	U	Z
	O	V	U	O
	O	F	F	Z
	P1			
	C	I	U	L
	B	I	U	L
	I	T	U	O
	P2			
	H	O	Q	D
	E	V	Q	F
	Z	F	T	W
	P3			
LC	F	F	F	F
	F	M	F	F
	F	F	M	M
	P1			
	F	M	F	F
	F	M	F	F
	F	M	F	F
	P2			
	F	F	M	M
	F	F	M	M
	F	F	M	M
	P3			

```
CREATE TABLE T1 (C1 date, C2 string, C3 number)
CLUSTER BY (C1, C2);
```

```
CREATE TABLE T1 (C1 date, C2 string, C3 number)
CLUSTER BY (MONTH(C1), SUBSTRING(C2,0,10));
```

```
ALTER TABLE T1 CLUSTER BY (C1, C3);
```

```
ALTER TABLE T2 CLUSTER BY (SUBSTRING(C2,5,10),
MONTH(C1));
```

# Reclustering and Clustering Cost

As DML operations are performed on a clustered table, the data in the table might become less clustered.

Reclustering is a background process which transparently reorganizes data in the micro-partitions by the clustering key.

Initial clustering and subsequent reclustering operations consume compute & storage credits.

Clustering is recommended for large tables which do not frequently change and are frequently queried.



COMPUTE

STORAGE

# Search Optimization

# Search Optimization Service

Search optimization service is a table level property aimed at improving the performance of selective point lookup queries.

①

```
SELECT NAME, ADDRESS FROM USERS  
WHERE USER_EMAIL = 'semper.google.edu';
```

②

```
SELECT NAME, ADDRESS FROM USERS  
WHERE USER_ID IN (4,5);
```

USER_ID	USER_NAME	USER_ADDRESS	USER_EMAIL
1	Duff Joisce	81 Mandrake Center	djoisce0@nasa.gov
2	Ira Downing	33214 Barnett Junction	idowning1@trellian.com
3	Alis Litel	9259 Russell Point	semper.google.edu
4	Cory Calderon	9266 New Castle Hill	ccalderon3@nydailynews.com
5	Pearl Denyuk	499 Thierer Hill	pdenyuk4@si.edu



The search optimization service is an enterprise edition feature.

# Search Optimization Service

A background process creates and maintains a search access path to enable search optimization.

```
ALTER TABLE MY_TABLE ADD SEARCH OPTIMIZATION;
```

```
ALTER TABLE MY_TABLE DROP SEARCH OPTIMIZATION;
```

```
SHOW TABLES LIKE '%MY_TABLE%';
```



The access path data structure requires space for each table on which search optimization is enabled. The larger the table, the larger the access path storage costs.

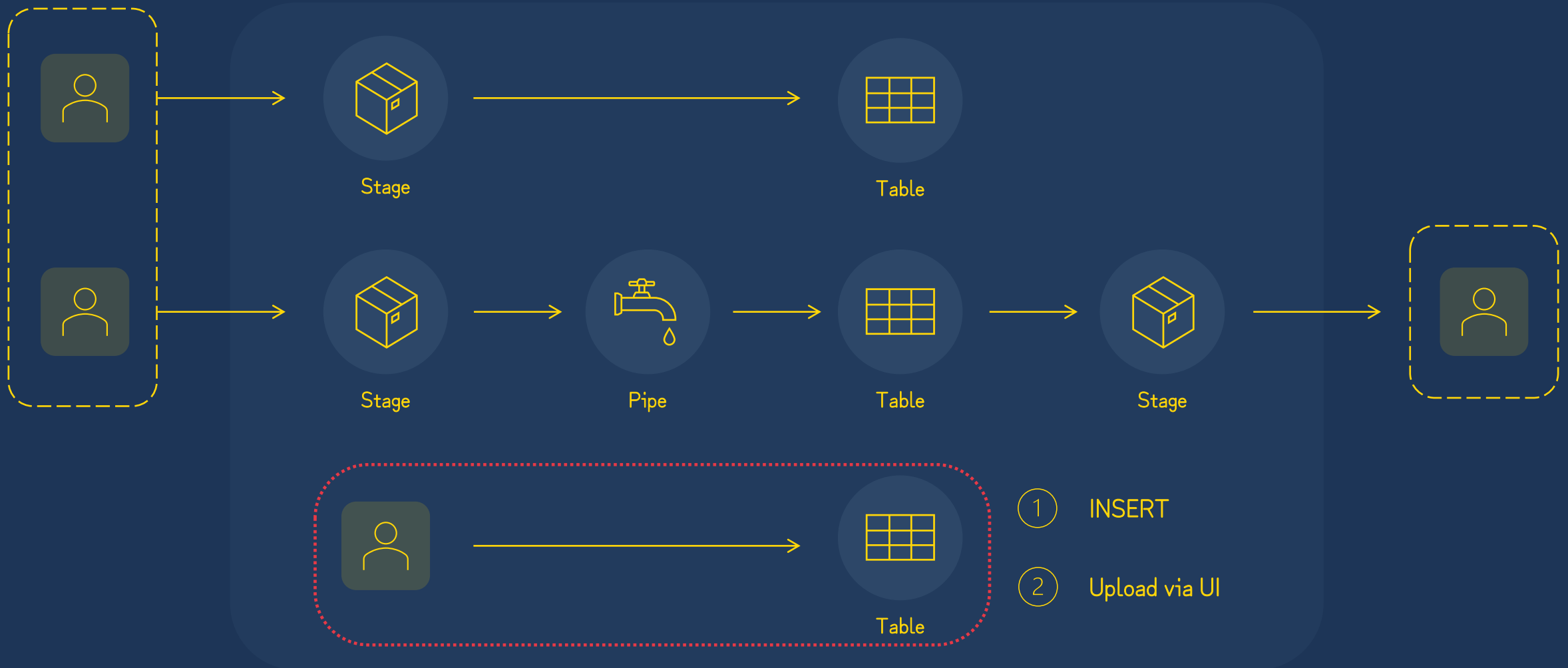


**10** Snowflake credits per Snowflake-managed compute hour  
**5** Snowflake credits per Cloud Services compute hour

# Data Loading Simple Methods



# Data Movement



# INSERT

```
INSERT INTO MY_TABLE SELECT '001', 'John Doughnut', '10/10/1976';
```

Insert a row into a table from the results of a select query.

```
INSERT INTO MY_TABLE (ID, NAME) SELECT '001', 'John Doughnut';
```

To load specific columns, individual columns can be specified

```
INSERT INTO MY_TABLE (ID, NAME, DOB) VALUES  
( '001', 'John Doughnut', '10/10/1976' ),  
( '002', 'Lisa Snowflake', '21/01/1934' ),  
( '003', 'Oggle Berry', '01/01/2001' );
```

The VALUES keyword can be used to insert multiple rows into a table.

```
INSERT INTO MY_TABLE SELECT * FROM MY_TABLE_2;
```

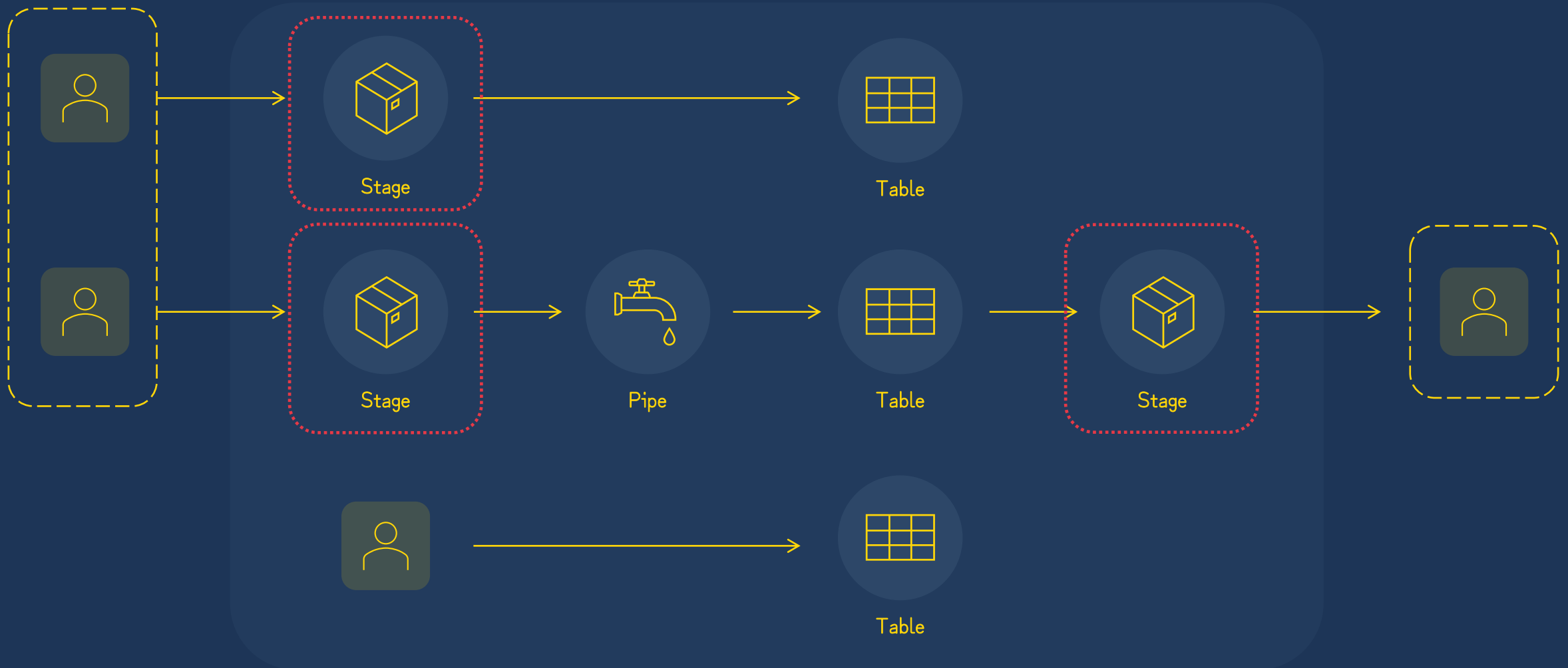
Another table can be used to insert rows into a table.

```
INSERT OVERWRITE INTO MY_TABLE SELECT * FROM MY_TABLE_2;
```

The keyword OVERWRITE will truncate a table before new values are inserted into it.

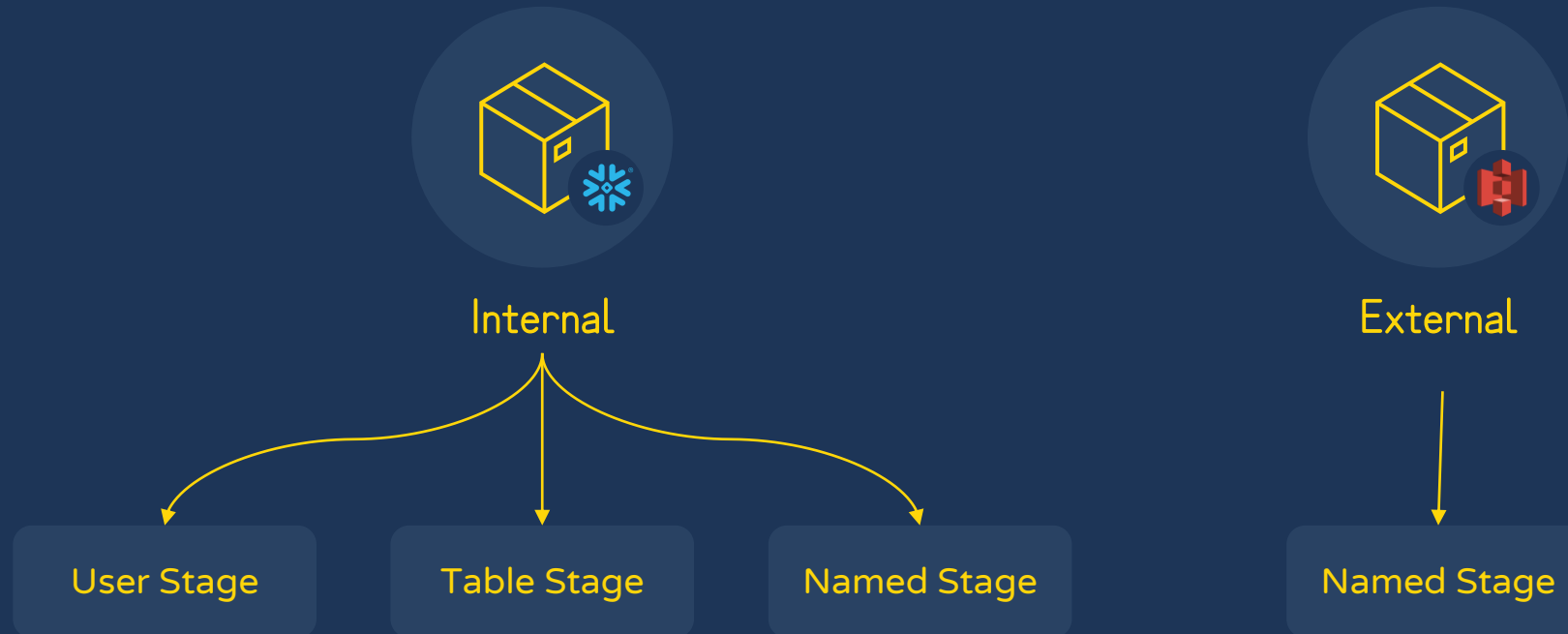
# Stages

# Stages



# Stages

Stages are temporary storage locations for data files used in the data loading and unloading process.



# Internal Stages

## User Stage

Automatically allocated when a user is created.

PUT

```
ls @~;
```

Cannot be altered or dropped.

Not appropriate if multiple users need access to stage.

## Table Stage

Automatically allocated when a table is created.

PUT

```
ls @%MY_TABLE;
```

Cannot be altered or dropped.

User must have ownership privileges on table.

## Named Stage

User created database object.

PUT

```
ls @MY_STAGE;
```

Securable object.

Supports copy transformations and applying file formats.

# External Stages and Storage Integrations

External stages reference data files stored in a location outside of Snowflake.

External Named  
Stage

User created database object.

```
CREATE STAGE MY_EXT_STAGE
URL='S3://MY_BUCKET/PATH/'
STORAGE_INTEGRATION=MY_INT;
AWS_SECRET_KEY=''
ENCRYPTION=(MASTER_KEY='')
```

Cloud Utilities

```
ls @MY_STAGE;
```

Storage location can be private or public.

```
CREATE STORAGE INTEGRATION MY_INT
TYPE=EXTERNAL_STAGE
STORAGE_PROVIDER=S3
STORAGE_AWS_ROLE_ARN='ARN:AWS:IAM::98765:ROLE/MY_ROLE'
ENABLED=TRUE
STORAGE_ALLOWED_LOCATIONS=('S3://MY_BUCKET/PATH/');
```

Copy options such as ON\_ERROR and PURGE can be set on stages.

A storage integration is a reusable and securable Snowflake object which can be applied across stages and is recommended to avoid having to explicitly set sensitive information for each stage definition.

# Stage Helper Commands

## LIST

```
LIST/ls @MY_STAGE;  
LIST/ls @~;  
LIST/ls @%MY_TABLE;
```

List the contents of a stage:

- Path of staged file
- Size of staged file
- MD5 Hash of staged file
- Last updated timestamp

Can optionally specify a path for specific folders or files.

Named and internal table stages can optionally include database and schema global pointer.

## SELECT

```
SELECT  
metadata$filename,  
metadata$file_row_number,  
$1,  
$2  
FROM @MY_STAGE  
(FILE_FORMAT => 'MY_FORMAT');
```

Query the contents of staged files directly using standard SQL for both internal and external stages.

Useful for inspected files prior to data loading/unloading.

Reference metadata columns such as filename and row numbers for a staged file.

## REMOVE

```
REMOVE/rm @MY_STAGE;  
REMOVE/rm @~;  
REMOVE/rm @%MY_TABLE;
```

Remove files from either an external or internal stage.

Can optionally specify a path for specific folders or files.

Named and internal table stages can optionally include database and schema global pointer.



# PUT

The PUT command uploads data files from a local directory on a client machine to any of the three types of internal stage.

PUT cannot be executed from within worksheets.

Duplicate files uploaded to a stage via PUT are ignored.

Uploaded files are automatically encrypted with a 128-bit key with optional support for a 256-bit key.

```
PUT FILE:///FOLDER/MY_DATA.CSV @MY_INT_STAGE;
```

```
PUT FILE:///FOLDER/MY_DATA.CSV @~;
```

```
PUT FILE:///FOLDER/MY_DATA.CSV @%MY_TABLE;
```

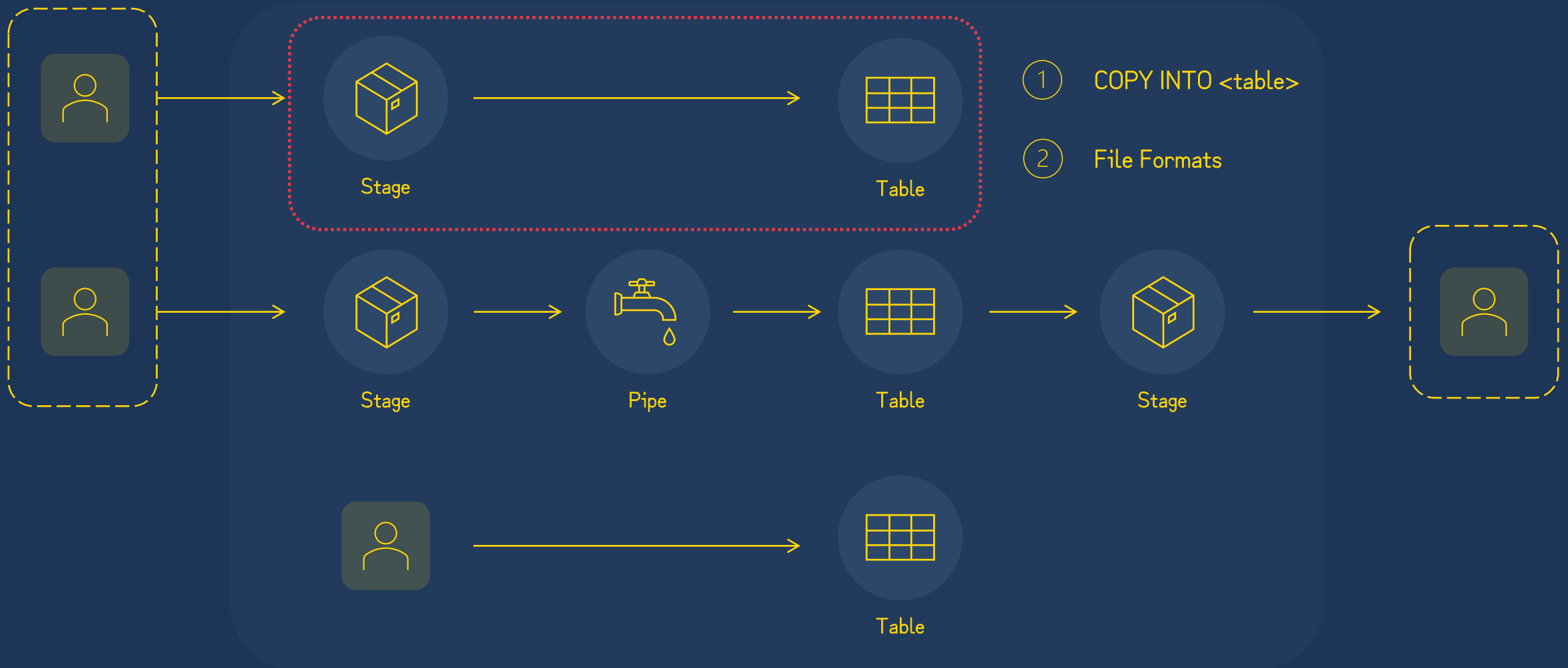
macOS / Linux

```
PUT FILE://c:\\FOLDER\\MY_DATA.CSV @MY_INT_STAGE;
```

Windows

# Bulk Loading with COPY INTO <table>

# Data Movement



# COPY INTO <table>

The COPY INTO <table> statement copies the contents of an internal or external stage or external location directly into a table.

The following file formats can be uploaded to Snowflake:

- Delimited files (CSV, TSC, etc)
- JSON
- Avro
- ORC
- Parquet
- XML

COPY INTO <table> requires a user created virtual warehouse to execute.

Load history is stored in the metadata of the target table for 64 days, which ensures files are not loaded twice.

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE;
```

CSV

JSON

Avro

ORC

Parquet

XML



64 Days

# COPY INTO <table>

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE;
```

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE/folder1;
```

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE/folder1/file1.csv;
```

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE  
FILE=('folder1/file1.csv', 'folder2/file2.csv');
```

```
COPY INTO MY_TABLE FROM @MY_INT_STAGE  
PATTERN=('people/.*[.]csv');
```

Copy all the contents of a stage into a table.

Copy contents of a stage from a specific folder/file path.

COPY INTO <table> has an option to provide a list of one or more files to copy.

COPY INTO <table> has an option to provide a regular expression to extract files to load.

# COPY INTO <table> Load Transformations

Snowflake allows users to perform simple transformations on data as it's loaded into a table.

Load transformations allows the user to perform:

- Column reordering.
- Column omission.
- Casting.
- Truncate test string that exceed target length.

Users can specify a set of fields to load from the staged data files using a standard SQL query.

```
COPY INTO MY_TABLE FROM (  
  SELECT  
    TO_DOUBLE(T.$1),  
    T.$2,  
    T.$3,  
    TO_TIMESTAMP(T.$4)  
  FROM @MY_INT_STAGE T);
```

# COPY External Stage/Location

```
COPY INTO MY_TABLE FROM @MY_EXTERNAL_STAGE;
```

Files can be loaded from external stages in the same way as internal stages.

```
COPY INTO MY_TABLE FROM S3://MY_BUCKET/  
STORAGE_INTEGRATION=MY_INTEGRATION  
ENCRYPTION=(MASTER_KEY='');
```

Data transfer billing charges may apply when loading data from files in a cloud storage service in a different region or cloud platform from your Snowflake account.

Files can be copied directly from a cloud storage service location.

Snowflake recommend encapsulating cloud storage service in an external stage.

# Copy Options

Copy Option	Definition	Default Value
ON_ERROR	Value that specifies the error handling for the load operation: <ul style="list-style-type: none"><li>• CONTINUE</li><li>• SKIP_FILE</li><li>• SKIP_FILE_&lt;num&gt;</li><li>• SKIP_FILE_&lt;num&gt;%</li><li>• ABORT_STATEMENT</li></ul>	'ABORT_STATEMENT'
SIZE_LIMIT	Number that specifies the maximum size of data loaded by a COPY statement.	null (no size limit)
PURGE	Boolean that specifies whether to remove the data files from the stage automatically after the data is loaded successfully.	FALSE
RETURN_FAILED_ONLY	Boolean that specifies whether to return only files that have failed to load in the statement result.	FALSE
MATCH_BY_COLUMN_NAME	String that specifies whether to load semi-structured data into columns in the target table that match corresponding columns represented in the data.	NONE
ENFORCE_LENGTH	Boolean that specifies whether to truncate text strings that exceed the target column length.	TRUE
TRUNCATECOLUMNS	Boolean that specifies whether to truncate text strings that exceed the target column length.	FALSE
FORCE	Boolean that specifies to load all files, regardless of whether they've been loaded previously and have not changed since they were loaded.	FALSE
LOAD_UNCERTAIN_FILES	Boolean that specifies to load files for which the load status is unknown. The COPY command skips these files by default.	FALSE



# COPY INTO <table> Output

Column Name	Data Type	Description
FILE	TEXT	Name of source file and relative path to the file.
STATUS	TEXT	Status: loaded, load failed or partially loaded.
ROWS_PARSED	NUMBER	Number of rows parsed from the source file.
ROWS_LOADED	NUMBER	Number of rows loaded from the source file.
ERROR_LIMIT	NUMBER	If the number of errors reaches this limit, then abort.
ERRORS_SEEN	NUMBER	Number of error rows in the source file.
FIRST_ERROR	TEXT	First error of the source file.
FIRST_ERROR_LINE	NUMBER	Line number of the first error.
FIRST_ERROR_CHARACTER	NUMBER	Position of the first error character.
FIRST_ERROR_COLUMN_NAME	TEXT	Column name of the first error.

Row	file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error	first_error_line	first_error_character	first_error_column_name
1	my_stage/pe...	LOADED	3	3	1	0	NULL	NULL	NULL	NULL

# COPY INTO <table> Validation

## VALIDATION\_MODE

Optional parameter allows you to perform a dry-run of load process to expose errors.

- RETURN\_N\_ROWS
- RETURN\_ERRORS
- RETURN\_ALL\_ERRORS

```
COPY INTO MY_TABLE  
FROM @MY_INT_STAGE;  
VALIDATION_MODE = 'RETURN_ERRORS';
```

## VALIDATE

Validate is a table function to view all errors encountered during a previous COPY INTO execution.

Validate accepts a job id of a previous query or the last load operation executed.

```
SELECT * FROM TABLE(VALIDATE(MY_TABLE,  
JOB_ID=>'5415FA1E-59C9-4DDA-B652-533DE02FDCF1'));
```

# File Formats

# File Formats

File format options can be set on a named stage or COPY INTO statement.

```
CREATE STAGE MY_STAGE  
FILE_FORMAT=(TYPE='CSV' SKIP_HEADER=1);
```

Explicitly declared file format options can all be rolled up into independent File Format Snowflake objects.

```
CREATE FILE FORMAT MY_CSV_FF  
TYPE='CSV'  
SKIP_HEADER=1;
```

File Formats can be applied to both named stages and COPY INTO statements. If set on both COPY INTO will take precedence.

```
CREATE OR REPLACE STAGE MY_STAGE  
FILE_FORMAT=MY_CSV_FF;
```

# File Formats

In the File Format object the file format you're expecting to load is set via the 'type' property with one of the following values: CSV, JSON, AVRO, ORC, PARQUET or XML.

Each 'type' has it's own set of properties related to parsing that specific file format.

```
CREATE FILE FORMAT MY_CSV_FF  
TYPE='CSV';
```

If a File Format object or options are not provided to either the stage or COPY statement, the default behaviour will be to try and interpret the contents of a stage as a CSV with UTF-8 encoding.

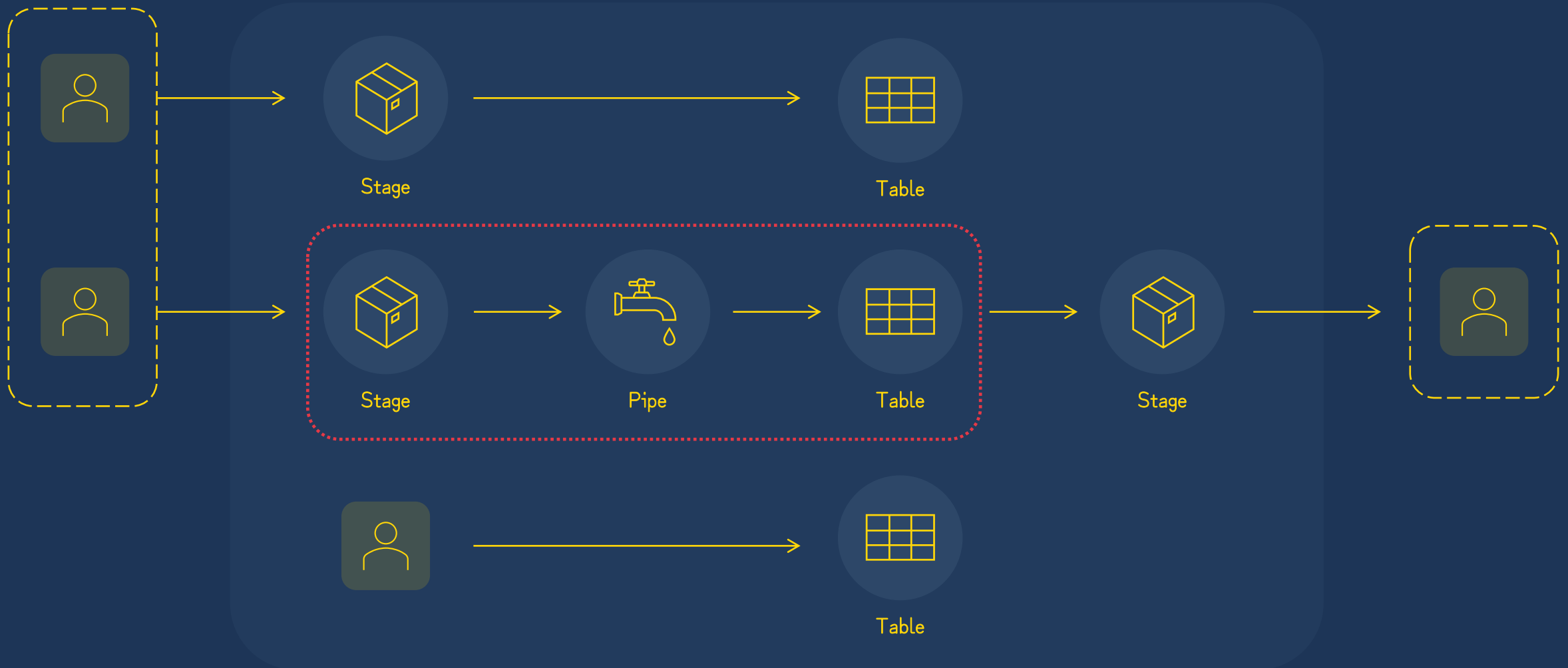
Row	property	property_value
1	TYPE	"CSV"
2	RECORD_DELIMITER	"\n"
3	FIELD_DELIMITER	","
4	FILE_EXTENSION	
5	SKIP_HEADER	0
6	DATE_FORMAT	"AUTO"
7	TIME_FORMAT	"AUTO"
8	TIMESTAMP_FORMAT	"AUTO"
9	BINARY_FORMAT	"HEX"
10	ESCAPE	"NONE"
11	ESCAPE_UNENCLOSED_FIELD	"\\"
12	TRIM_SPACE	false
13	FIELD_OPTIONALLY_ENCLOSED_BY	"NONE"
14	NULL_IF	["\\N"]
15	COMPRESSION	"AUTO"
16	ERROR_ON_COLUMN_COUNT_MISMATCH	true
17	VALIDATE_UTF8	true
18	SKIP_BLANK_LINES	false
19	REPLACE_INVALID_CHARACTERS	false
20	EMPTY_FIELD_AS_NULL	true
21	SKIP_BYTE_ORDER_MARK	true
22	ENCODING	"UTF8"

Number of lines at the start of the file to skip.

Specifies the current compression algorithm for the data file.

# Snowpipe and Loading Best Practises

# Snowpipe



# Snowpipe

```
CREATE PIPE MY_PIPE
```

```
AUTO_INGEST=TRUE
```

```
AS
```

```
COPY INTO MY_TABLE
```

```
FROM @MY_STAGE
```

```
FILE_FORMAT = (TYPE = 'CSV');
```

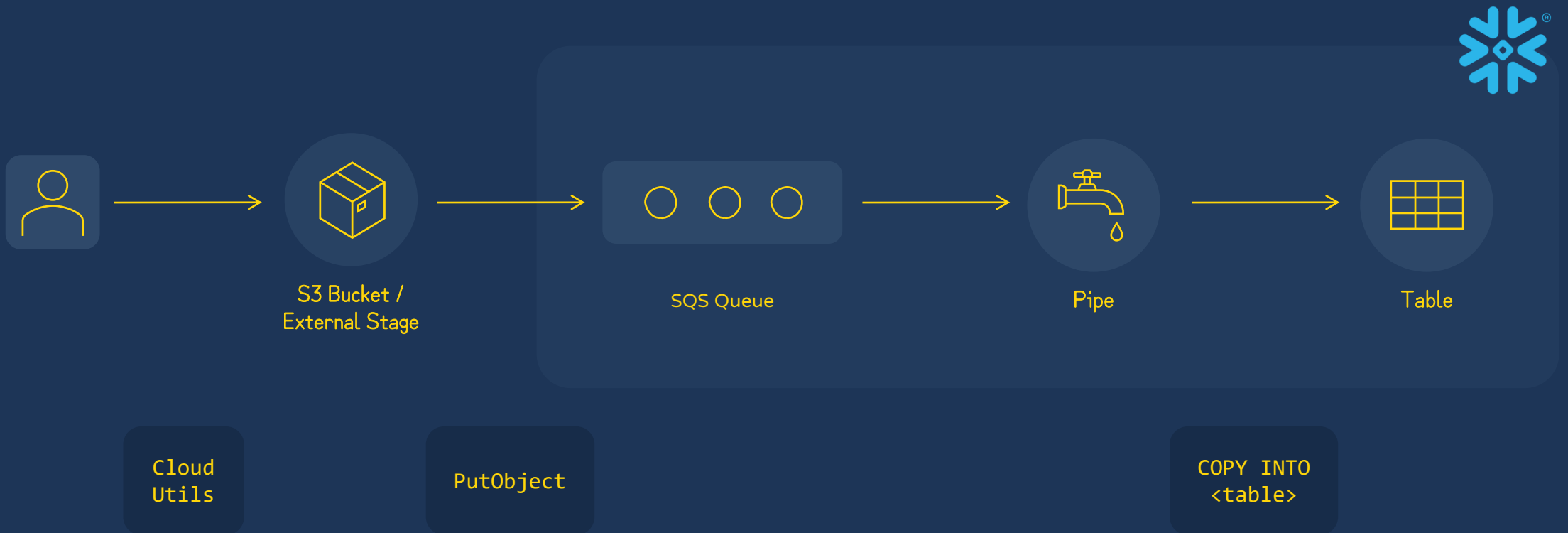
There are two methods for detecting when a new file has been uploaded to a stage:

- Automating Snowpipe using cloud messaging  
(external stages only)
- Calling Snowpipe REST endpoints  
(internal and external stages)

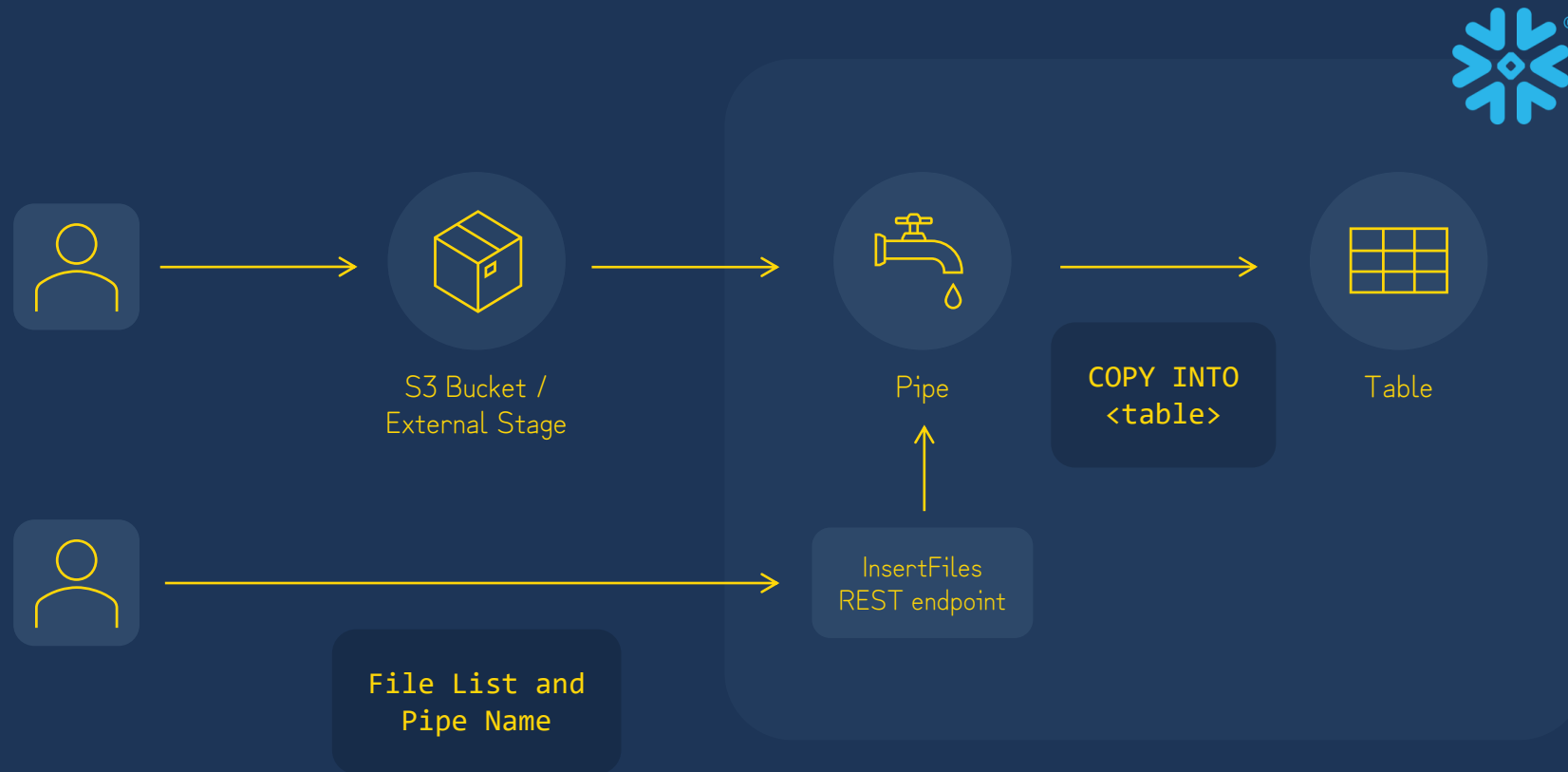
The Pipe object defines a COPY INTO <table> statement that will execute in response to a file being uploaded to a stage.



# Snowpipe: Cloud Messaging



# Snowpipe: REST Endpoint



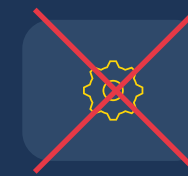
# Snowpipe

Snowpipe is designed to load new data typically within a minute after a file notification is sent.

Snowpipe is serverless feature, using Snowflake managed compute resources to load data files not a user managed Virtual Warehouse.

Snowpipe load history is stored in the metadata of the pipe for 14 days, used to prevent reloading the same files in a table.

When a pipe is paused, event messages received for the pipe enter a limited retention period. The period is 14 days by default.



14 Days

14 Days

# Bulk Loading vs. Snowpipe

Feature	Bulk Loading	Snowpipe
Authentication	Relies on the security options supported by the client for authenticating and initiating a user session.	When calling the REST endpoints: Requires key pair authentication with JSON Web Token (JWT). JWTs are signed using a public/private key pair with RSA encryption.
Load History	Stored in the metadata of the target table for <b>64 days</b> .	Stored in the metadata of the pipe for <b>14 days</b> .
Compute Resources	Requires a user-specified warehouse to execute COPY statements.	Uses Snowflake-supplied compute resources.
Billing	Billed for the amount of time each virtual warehouse is active.	<p>Snowflake tracks the resource consumption of loads for all pipes in an account, with per-second/per-core granularity, as Snowpipe actively queues and processes data files.</p> <p>In addition to resource consumption, an overhead is included in the utilization costs charged for Snowpipe: <b>0.06 credits per 1000 files</b> notified or listed via event notifications or REST API calls.</p>

# Data Loading Best Practises



100-250 MB Compressed

- 2022/07/10/05/
- 2022/06/01/11/

Organize Data By Path



Load



Query



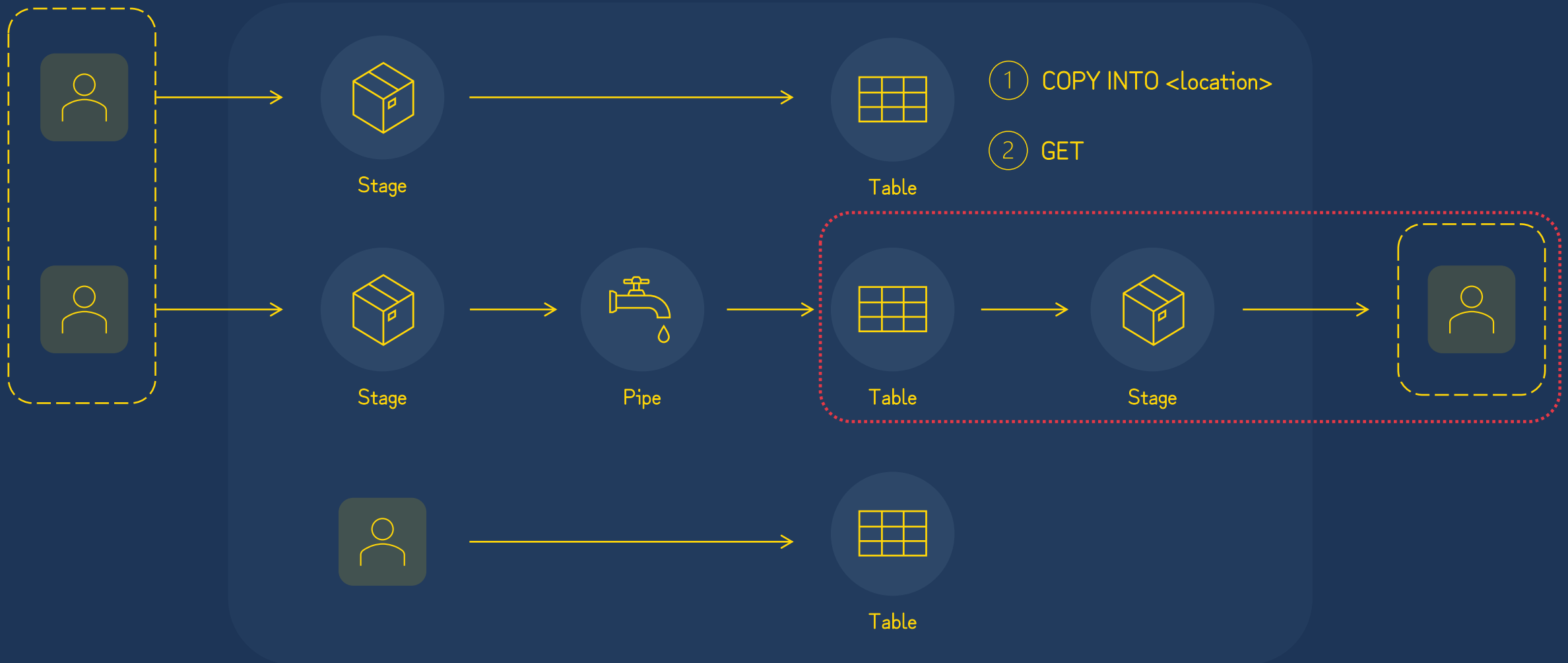
Pre-sort data

1  
Minute

Once per minute

# Data Unloading Overview

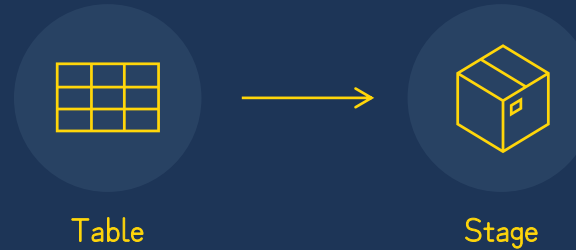
# Data Unloading



# Data Unloading

Table data can be unloaded to a stage via the `COPY INTO <location>` command.

```
COPY INTO @MY_STAGE  
FROM MY_TABLE;
```



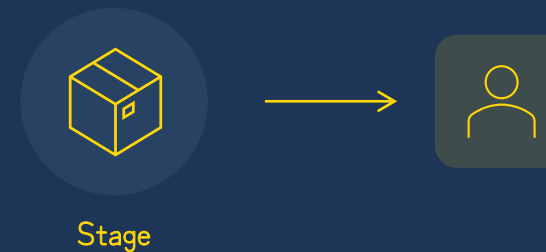
CSV

JSON

Parquet

The `GET` command is used to download a staged file to the local file system.

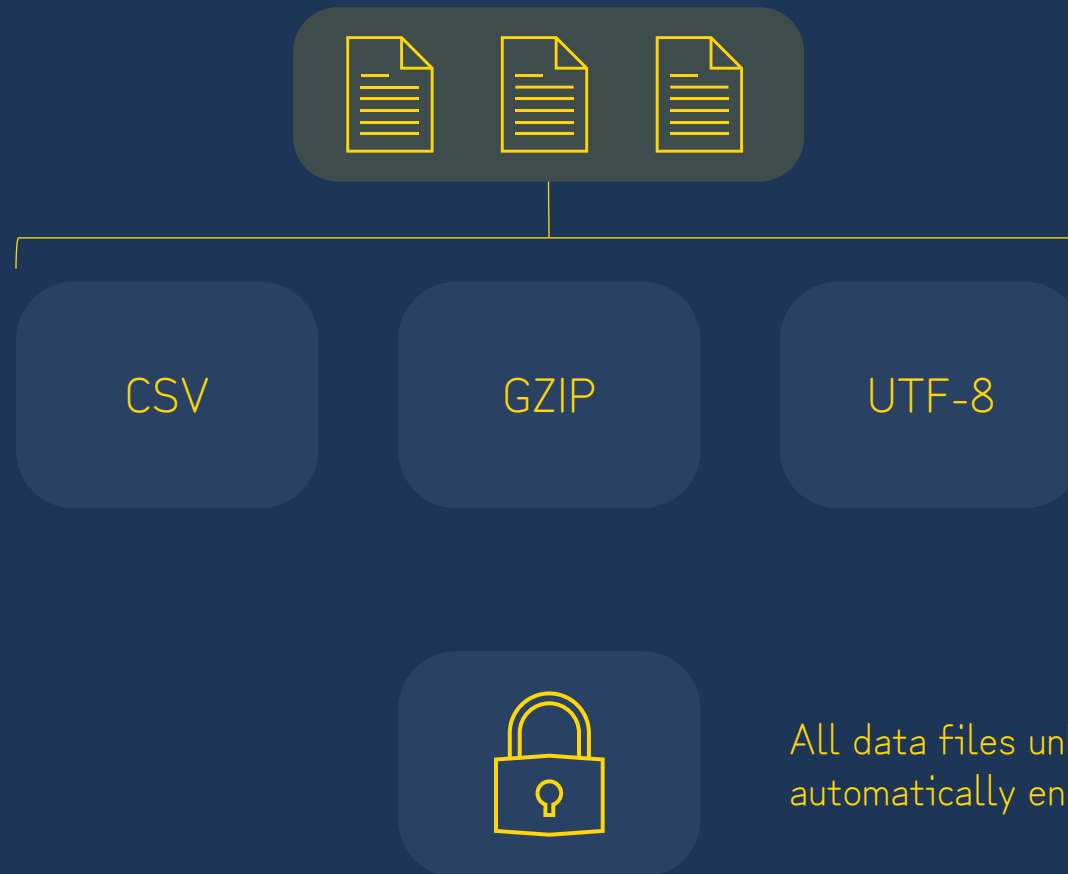
```
GET @MY_STAGE  
file:///folder/files/;
```





# Data Unloading

By default results unloaded to a stage using COPY INTO <location> command are split in to multiple files:



All data files unloaded to internal stages are automatically encrypted using 128-bit keys.

# COPY INTO <location> Examples

```
COPY INTO @MY_STAGE/RESULT/DATA_  
FROM (SELECT * FROM T1)  
FILE_FORMAT = MY_CSV_FILE_FORMAT;
```

Output files can be prefixed by specifying a string at the end of a stage path.

```
COPY INTO @%T1  
FROM T1  
PARTITION BY ('DATE=' || TO_VARCHAR(DT))  
FILE_FORMAT=MY_CSV_FILE_FORMAT;
```

COPY INTO <location> includes a PARTITION BY copy option to partition unloaded data into a directory structure.

```
COPY INTO 'S3://MYBUCKET/UNLOAD/'  
FROM T1  
STORAGE_INTEGRATION = MY_INT  
FILE_FORMAT=MY_CSV_FILE_FORMAT;
```

COPY INTO <location> can copy table records directly to external cloud provider's blob storage.

# COPY INTO <location> Copy Options

Copy Option	Definition	Default Value
OVERWRITE	Boolean that specifies whether the COPY command overwrites existing files with matching names, if any, in the location where files are stored.	'ABORT_STATEMENT'
SINGLE	Boolean that specifies whether to generate a single file or multiple files.	FALSE
MAX_FILE_SIZE	Number (> 0) that specifies the upper size limit (in bytes) of each file to be generated in parallel per thread.	FALSE
INCLUDE_QUERY_ID	Boolean that specifies whether to uniquely identify unloaded files by including a universally unique identifier (UUID) in the filenames of unloaded data files.	FALSE

# GET

GET is the reverse of PUT. It allows users to specify a source stage and a target local directory to download files to.

GET cannot be used for external stages.

GET cannot be executed from within worksheets.

Downloaded files are automatically decrypted.

**Parallel** optional parameter specifies the number of threads to use for downloading files. Increasing this number can improve parallelisation with downloading large files.

**Pattern** optional parameter specifies a regular expression pattern for filtering files to download.

```
GET @MY_STAGE FILE:///TMP/DATA/;
```

```
GET @MY_STAGE FILE:///TMP/DATA/  
PARALLEL=99;
```

```
GET @MY_STAGE FILE:///TMP/DATA/  
PATTERN='*\\. (csv)';
```

# Semi-structured Overview

# Semi-structured Data Types

## VARIANT

- VARIANT is universal semi-structured data type of Snowflake for loading data in semi-structured data formats.
- VARIANT are used to represent arbitrary data structures.
- Snowflake stores the VARIANT type internally in an efficient compressed columnar binary representation.
- Snowflake extracts as much of the data as possible to a columnar form, based on certain rules.
- VARIANT data type can hold up to 16MB compressed data per row.
- VARIANT column can contain SQL NULLs and VARIANT NULL which are stored as a string containing the word “null”.

# Semi-structured Data Overview

```
{
  "widget": {
    "debug": "on",
    "window": {
      "title": "Sample Konfabulator Widget",
      "name": "main_window",
      "width": 500,
      "height": 500
    },
    "image": {
      "src": "Images/Sun.png",
      "name": "sun1",
      "hOffset": [250, 300, 850],
      "alignment": "center"
    },
    "text": {
      "data": "Click Here",
      "size": 36,
      "style": "bold",
      "name": "text1",
      "hOffset": 250,
      "vOffset": 100,
      "alignment": "center",
      "onMouseUp": "sun1.opacity = 90;"
    }
  }
}
```

JSON

```
<widget>
  <debug>on</debug>
  <window title="Sample Konfabulator Widget">
    <name>main_window</name>
    <width>500</width>
    <height>500</height>
  </window>
  <image src="Images/Sun.png" name="sun1">
    <hOffset>250</hOffset>
    <hOffset>300</hOffset>
    <hOffset>850</hOffset>
    <alignment>center</alignment>
  </image>
  <text data="Click Here" size="36" style="bold">
    <name>text1</name>
    <hOffset>250</hOffset>
    <vOffset>100</vOffset>
    <alignment>center</alignment>
    <onMouseUp>
      sun1.opacity = 90;
    </onMouseUp>
  </text>
</widget>
```

XML

# Semi-structured Data Types

## ARRAY

Contains 0 or more elements of data. Each element is accessed by its position in the array.

```
CREATE TABLE MY_ARRAY_TABLE (  
  NAME VARCHAR,  
  HOBBIES ARRAY  
);
```

```
INSERT INTO MY_ARRAY_TABLE  
SELECT 'Alina Nowak', ARRAY_CONSTRUCT('Writing', 'Tennis', 'Baking');
```

Row	NAME	HOBBIES
1	Alina Nowak	[ "Writing", "Tennis", "Baking" ]



# Semi-structured Data Types

## OBJECT

Represent collections of key-value pairs.

```
CREATE TABLE MY_OBJECT_TABLE (  
  NAME VARCHAR,  
  ADDRESS OBJECT  
);
```

```
INSERT INTO MY_OBJECT_TABLE  
SELECT 'Alina Nowak', OBJECT_CONSTRUCT('postcode', 'TY5 7NN', 'first_line', '67 Southway Road');
```

↓ Row	NAME	ADDRESS
1	Alina Nowak	{ "first_line": "67 Southway Road", "postcode": "TY5 7NN" }

# Semi-structured Data Types

## VARIANT

Universal Semi-structured data type used to represent arbitrary data structures.

VARIANT data type can hold up to 16MB compressed data per row

```
CREATE TABLE MY_VARIANT_TABLE (  
  NAME VARIANT,  
  ADDRESS VARIANT,  
  HOBBIES VARIANT  
);
```

```
INSERT INTO MY_VARIANT_TABLE  
SELECT  
  'Alina Nowak'::VARIANT,  
  OBJECT_CONSTRUCT('postcode', 'TY5 7NN', 'first_line', '67 Southway Road'),  
  ARRAY_CONSTRUCT('Writing', 'Tennis', 'Baking');
```

Row	NAME	ADDRESS	HOBBIES
1	"Alina Nowak"	{ "first_line": "67 Southway Road", "postcode": "TY5 7NN" }	[ "Writing", "Tennis", "Baking" ]

# Semi-structured Data Formats

## JSON

Plain-text data-interchange format based on a subset of the JavaScript programming language.

Load

Unload

## AVRO

Binary row-based storage format originally developed for use with Apache Hadoop.

Load

## ORC

Highly efficient binary format used to store Hive data.

Load

## PARQUET

Binary format designed for projects in the Hadoop ecosystem.

Load

Unload

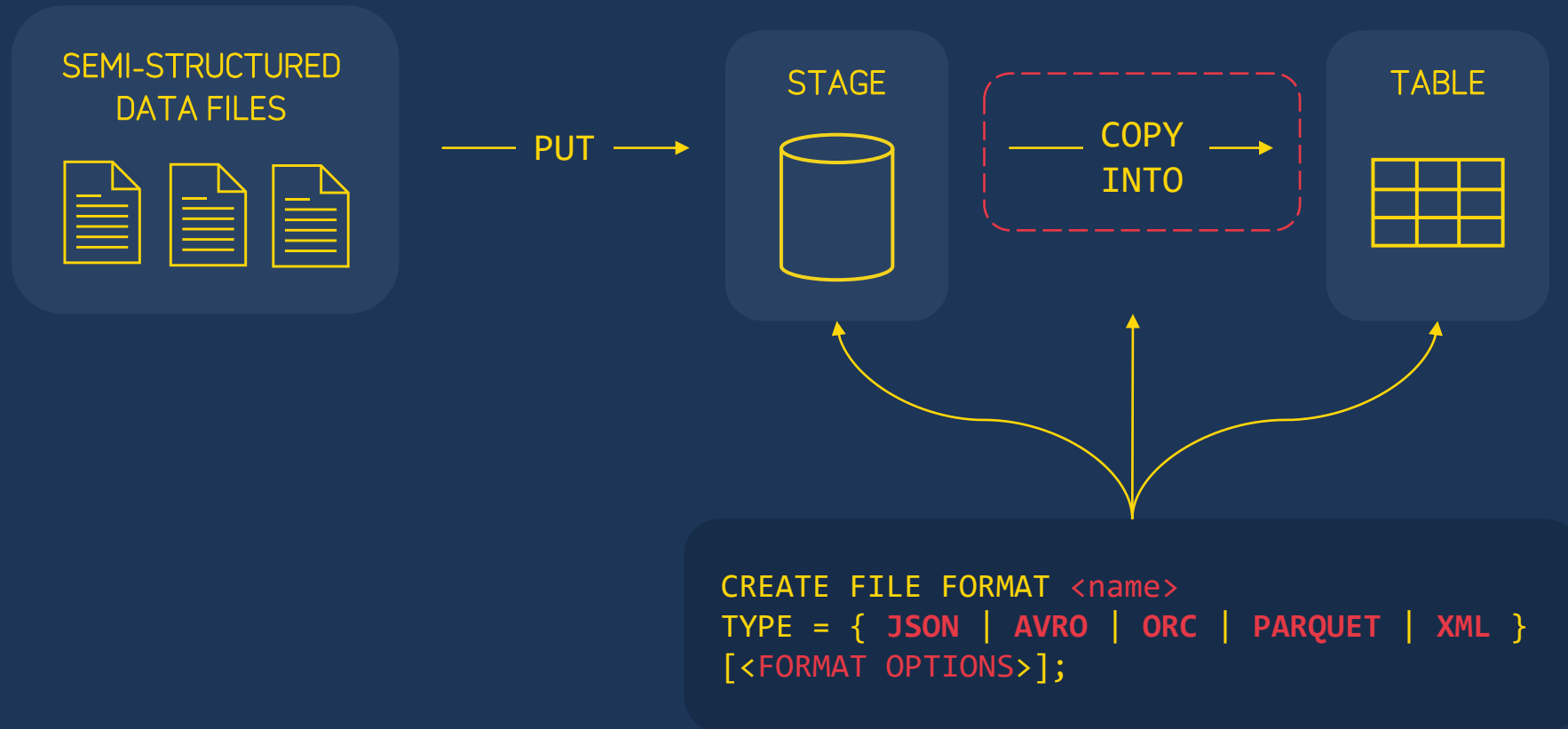
## XML

Consists primarily of tags <> and elements.

Load

# Loading and Unloading Semi-structured Data

# Loading Semi-structured Data



# JSON File Format options

```
DESC FILE FORMAT FF_JSON;
```

Option	Details
DATE_FORMAT	Used only for loading JSON data into separate columns. Defines the format of date string values in the data files.
TIME FORMAT	Used only for loading JSON data into separate columns. Defines the format on time string values in the data files.
COMPRESSION	Supported algorithms: GZIP, BZ2, BROTLI, ZSTD, DEFLATE, RAW_DEFLATE, NONE. If BROTLI, cannot use AUTO.
ALLOW DUPLICATE	Only used for loading. If TRUE, allows duplicate object field names (only the last one will be preserved)
STRIP OUTER ARRAY	Only used for loading. If TRUE, JSON parser will remove outer brackets []
STRIP NULL VALUES	Only used for loading. If TRUE, JSON parser will remove object fields or array elements containing NULL

property	property_value
TYPE	"JSON"
FILE_EXTENSION	
DATE_FORMAT	"AUTO"
TIME_FORMAT	"AUTO"
TIMESTAMP_FORMAT	"AUTO"
BINARY_FORMAT	"HEX"
TRIM_SPACE	false
NULL_IF	[]
COMPRESSION	"AUTO"
ENABLE_OCTAL	false
ALLOW_DUPLICATE	false
STRIP_OUTER_ARRAY	false
STRIP_NULL_VALUES	false
IGNORE_UTF8_ERRORS	false
REPLACE_INVALID_CHARACTERS	false
SKIP_BYTE_ORDER_MARK	true

# Semi-structured Data Loading Approaches

## ELT

①

```
CREATE TABLE MY_TABLE (  
  V VARIANT  
);
```

②

```
COPY INTO MY_TABLE  
FROM @MY_STAGE/FILE1.JSON  
FILE_FORMAT = FF_JSON;
```

## ETL

①

```
CREATE TABLE MY_TABLE (  
  NAME STRING,  
  AGE NUMBER,  
  DOB DATE  
);
```

②

```
COPY INTO MY_TABLE  
FROM ( SELECT  
  V:name,  
  V:age,  
  V:dob  
FROM @MY_STAGE/FILE1.JSON)  
FILE_FORMAT = FF_JSON;
```

## Automatic Schema Detection

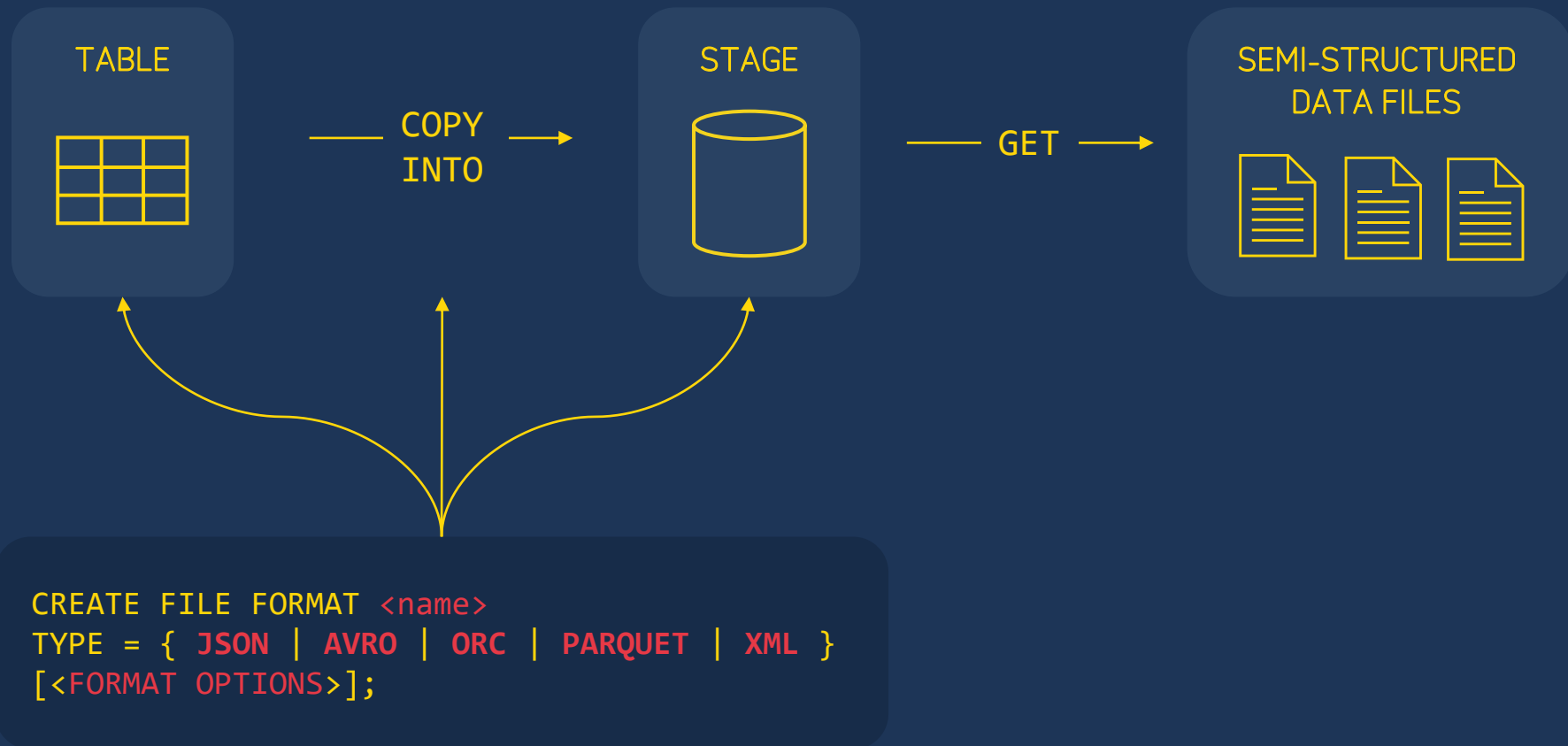
### INFER\_SCHEMA

```
CREATE TABLE MY_TABLE  
  USING TEMPLATE (  
    SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))  
    FROM TABLE(  
      INFER_SCHEMA(  
        LOCATION=>'@MY_STAGE',  
        FILE_FORMAT=>'FF_PARQUET'  
      )  
    ));
```

### MATCH\_BY\_COLUMN\_NAME

```
COPY INTO MY_TABLE  
FROM @MY_STAGE/FILE1.JSON  
FILE_FORMAT = (TYPE = 'JSON')  
MATCH_BY_COLUMN_NAME = CASE_SENSITIVE;
```

# Unloading Semi-structured Data





# Accessing Semi-structured Data

# Accessing Semi-structured Data

```
{  
  "employee": {  
    "name": "Aiko Tanaka",  
    "_id": "UNX789544",  
    "age": 42  
  },  
  "joined_on": "2019-01-01",  
  "skills": ["Java", "Kotlin", "Android"],  
  "is_manager": true,  
  "base_location": null,  
}
```

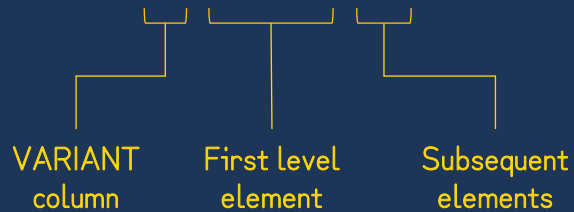
→  
COPY INTO

```
CREATE TABLE EMPLOYEE (  
  src VARIANT  
);
```

# Accessing Semi-structured Data

## Dot Notation

```
SELECT src:employee.name FROM EMPLOYEES;
```



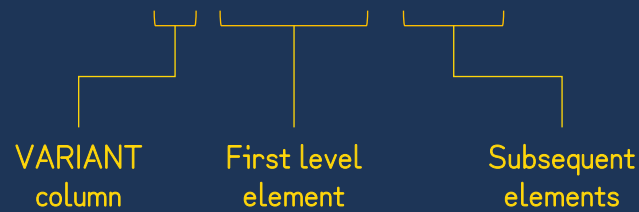
Row	SRC:EMPLOYEE.NAME
1	"Aiko Tanaka"

```
SELECT SRC:Employee.name FROM EMPLOYEES;
```

Column name is case insensitive but key names are case insensitive so the above query would result in an error.

## Bracket Notation

```
SELECT SRC['employee']['name'] FROM EMPLOYEES;
```



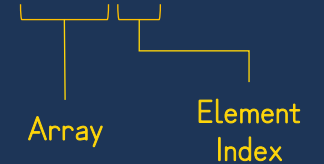
Row	SRC:EMPLOYEE.NAME
1	"Aiko Tanaka"

```
SELECT SRC['Employee']['name'] FROM EMPLOYEES;
```

Column name is case insensitive but key names are case insensitive so the above query would result in an error.

## Repeating Element

```
SELECT SRC:skills[0] FROM EMPLOYEES;
```



Row	SRC:SKILLS[0]
1	"Java"

```
SELECT GET(SRC, 'employee')  
FROM EMPLOYEE;
```

```
SELECT GET(SRC, 'skills')[0]  
FROM EMPLOYEE;
```

# Casting Semi-structured Data

```
SELECT src:employee.name, src:joined_on, src:employee.age, src:is_manager, src:base_location FROM EMPLOYEE;
```

Row	SRC:EMPLOYEE.NAME	SRC:JOINED_ON	SRC:EMPLOYEE.AGE	SRC:IS_MANAGER	SRC:BASE_LOCATION
1	"Aiko Tanaka"	"2019-01-01"	42	true	null

Double colon

```
SELECT src:employee.joined_on::DATE  
FROM EMPLOYEE;
```

Row	SRC:JOINED_ON::DATE
1	2019-01-01

TO\_<datatype>()

```
SELECT TO_DATE(src:employee.joined_on)  
FROM EMPLOYEE;
```

Row	TO_DATE(SRC:JOINED_ON)
1	2019-01-01

AS\_<datatype>()

```
SELECT AS_VARCHAR(src:employee.name)  
FROM EMPLOYEE;
```

Row	AS_VARCHAR(SRC:EMPLOYEE.NAME)
1	Aiko Tanaka

# Semi-structured Functions

# Semi-structured Functions

JSON and XML Parsing	Array/Object Creation and Manipulation	Extraction	Conversion/Casting	Type Predicates
CHECK_JSON	ARRAY_AGG	FLATTEN	AS_<object>	IS_<object>
CHECK_XML	ARRAY_APPEND	GET	AS_ARRAY	IS_ARRAY
JSON_EXTRACT_PATH_TEXT	ARRAY_CAT	GET_IGNORE_CASE	AS_BINARY	IS_BOOLEAN
PARSE_JSON	ARRAY_COMPACT	GET_PATH	AS_CHAR , AS_VARCHAR	IS_BINARY
PARSE_XML	ARRAY_CONSTRUCT	OBJECT_KEYS	AS_DATE	IS_CHAR ,
STRIP_NULL_VALUE	ARRAY_CONSTRUCT_COMPACT	XMLGET	AS_DECIMAL , AS_NUMBER	IS_VARCHAR
	ARRAY_CONTAINS		AS_DOUBLE , AS_REAL	IS_DATE ,
	ARRAY_INSERT		AS_INTEGER	IS_DATE_VALUE
	ARRAY_INTERSECTION		AS_OBJECT	IS_DECIMAL
	ARRAY_POSITION		AS_TIME	IS_DOUBLE ,
	ARRAY_PREPEND		AS_TIMESTAMP_*	IS_REAL
	ARRAY_SIZE		STRTOK_TO_ARRAY	IS_INTEGER
	ARRAY_SLICE		TO_ARRAY	IS_NULL_VALUE
	ARRAY_TO_STRING		TO_JSON	IS_OBJECT
	ARRAYS_OVERLAP		TO_OBJECT	IS_TIME
	OBJECT_AGG		TO_VARIANT	IS_TIMESTAMP_*
	OBJECT_CONSTRUCT		TO_XML	typeof
	OBJECT_CONSTRUCT_KEEP_NULL			
	OBJECT_DELETE			
	OBJECT_INSERT			
	OBJECT_PICK			

# FLATTEN Table Function

Flatten is a table function that accepts compound values (VARIANT, OBJECT & ARRAY) and produces a row for each item.

```
SELECT VALUE FROM TABLE(FLATTEN(INPUT => SELECT src:skills FROM EMPLOYEE));
```

Row	VALUE::VARCHAR
1	Java
2	Kotlin
3	Android

## Path

```
SELECT VALUE FROM TABLE(FLATTEN(  
  INPUT => PARSE_JSON('{ "A":1, "B":[77,88]}'),  
  PATH => 'B'));
```

Specify a path inside object to flatten.

## Recursive

```
SELECT VALUE FROM TABLE(FLATTEN(  
  INPUT => PARSE_JSON('{ "A":1, "B":[77,88]}'),  
  RECURSIVE => true));
```

Flattening is performed for all sub-elements recursively.

# FLATTEN Output

```
SELECT * FROM TABLE(FLATTEN(INPUT => (ARRAY_CONSTRUCT(1,45,34))));
```

SEQ	KEY	PATH	INDEX	VALUE	THIS
1	NULL	[0]	0	1	[ 1, 45, 34 ]
1	NULL	[1]	1	45	[ 1, 45, 34 ]
1	NULL	[2]	2	34	[ 1, 45, 34 ]

SEQ	KEY	PATH	INDEX	VALUE	THIS
A unique sequence number associated with the input record.	For maps or objects, this column contains the key to the exploded value.	The path to the element within a data structure which needs to be flattened.	The index of the element, if it is an array; otherwise NULL.	The value of the element of the flattened array/object.	The element being flattened (useful in recursive flattening).



# LATERAL FLATTEN

```
SELECT src:employee.name::varchar, src:employee._id::varchar, src:skills FROM EMPLOYEE;
```

Row	SRC:EMPLOYEE.NAME::VARCHAR	SRC:EMPLOYEE._ID::VARCHAR	SRC:SKILLS
1	Aiko Tanaka	UNX789544	[ "Java", "Kotlin", "Android" ]

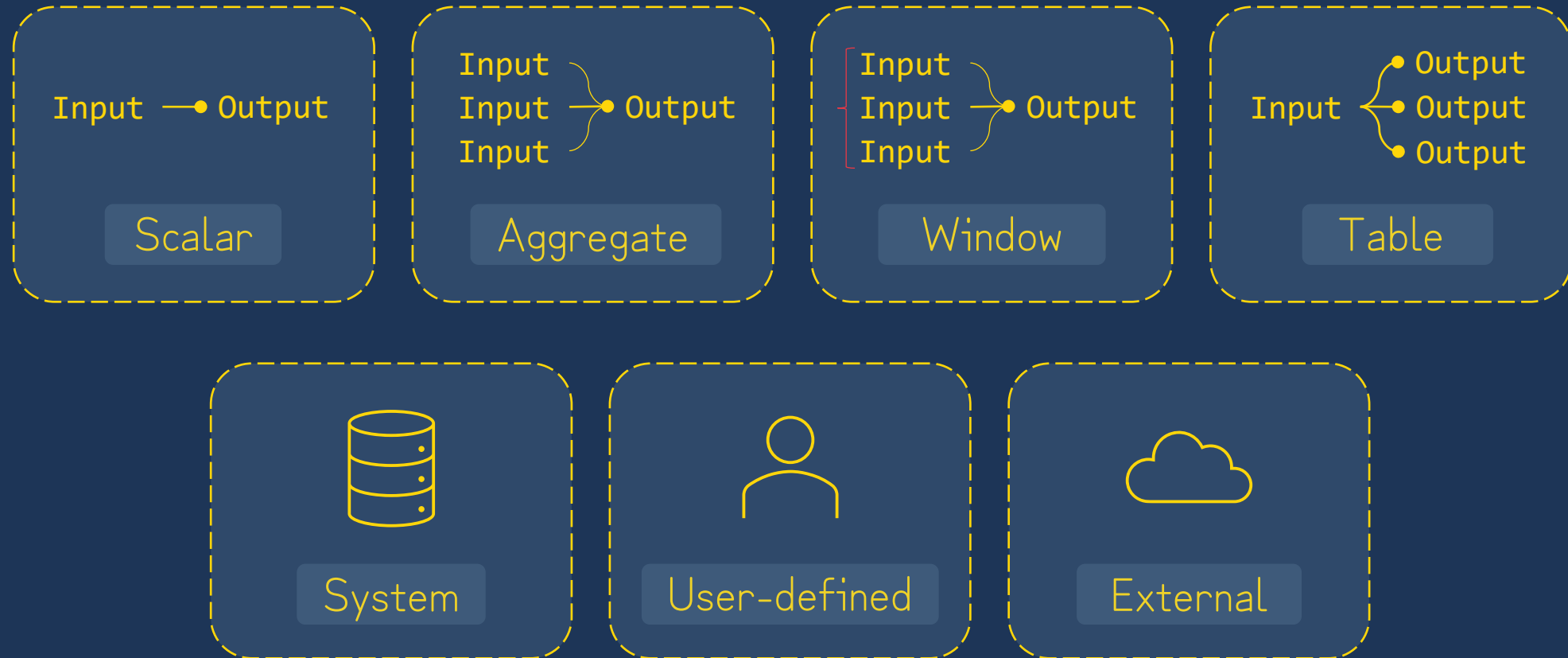
```
SELECT  
src:employee.name,  
src:employee._id,  
f.value  
FROM EMPLOYEE e,  
LATERAL FLATTEN(INPUT => e.src:skills) f;
```



```
Aiko Tanaka, UNX789544, [ "Java", "Kotlin", "Android" ]  
  
Aiko Tanaka, UNX789544,  
  
Aiko Tanaka, UNX789544,
```

# Summary of Snowflake Functions

# Supported Function Types



# Scalar Functions

Bitwise Expression

Conditional Expression

Context

Conversion

Data Generation

Date & Time

Encryption



A **scalar function** is a function that **returns one value per invocation**; these are mostly used for returning **one value per row**.

```
SELECT UUID_STRING();
```

Output:

```
|"UUID_STRING()"|  
|d29d4bfa-40cb-4159-9186-e10f5d59f031|
```

Semi-structured Data

String & Binary

Regular Expressions

Hash

Metadata

File

Geospatial

Numeric

# Aggregate Functions



**Aggregate functions** operate on values across rows to perform mathematical calculations such as sum, average & counting.

General

Bitwise

Boolean

Hash

Semi-structured

Linear Regression

Stats and Probability

Distinct Values

Cardinality Estimation

Similarity Estimation

Frequency Estimation

Percentile Estimation

```
INSERT INTO ACCOUNT VALUES ('001', 10.00), ('001', 23.78), ('002', 67.78);  
  
SELECT MAX(AMOUNT) FROM ACCOUNT;
```

Output:

"MAX(AMOUNT)"
67.78

# Window Functions



Window functions are a subset of aggregate functions, allowing us to **aggregate on a subset of rows** used as input to a function.

```
SELECT ACCOUNT_ID, AMOUNT, MAX(AMOUNT) OVER (PARTITION BY ACCOUNT_ID) FROM ACCOUNT;
```

Output:

"ACCOUNT_ID"	"AMOUNT"	"MAX(AMOUNT)"
001	10.00	23.78
001	23.78	23.78
002	67.78	67.78

# Table Functions



Table functions return a **set of rows** for each input row. The returned set can contain **zero, one, or more rows**. Each row can contain **one or more columns**.

Data Loading

Data Generation

Data Conversion

Object Modelling

Semi-structured

Query Results

Usage Information

```
SELECT RANDSTR(5, RANDOM()), RANDOM() FROM TABLE(Generator(rowcount => 3));
```

Output:

"RANDSTR(5, RANDOM())"	"RANDOM()"
My4FU	574440610751796211
YiPSS	1779357660907745898
cu2Hw	6562320827285185330

# System Functions

- 1 System functions provide a way to execute actions in the system.

```
SELECT system$cancel_query('01a65819-0000-2547-0000-94850008c1ee');
```

Output:

```
|“SYSTEM$CANCEL_QUERY('01A65819-0000-2547-0000-94850008C1EE’)”|  
|query [01a65819-0000-2547-0000-94850008c1ee] terminated.    |
```



# System Functions

- 2 System functions provide information about the system.

```
SELECT system$pipe_status('my_pipe');
```

Output:

```
| "SYSTEM$PIPE_STATUS('MYPIPE') " |  
| {"executionState": "RUNNING", "pendingFileCount": 0} |
```

# System Functions

3

System functions provide information about queries.

```
SELECT system$explain_plan_json('SELECT AMOUNT FROM ACCOUNT');
```

Output:

```
|“SYSTEM$EXPLAIN_PLAN_JSON('SELECT AMOUNT FROM ACCOUNT')” |  
|{  
|  "GlobalStats": {  
|    "partitionsTotal": 1,  
|    "partitionsAssigned": 1,  
|    "bytesAssigned": 1024  
|  }[...]  
|}
```

# Estimation Functions

# Estimation Functions

Cardinality Estimation

Estimate the  
number of  
distinct values.

Similarity Estimation

Estimate  
similarity of  
two or more  
sets.

Frequency Estimation

Estimate  
frequency  
values in a set.

Percentile Estimation

Estimate  
percentile of  
values in a set.

# Cardinality Estimation



Snowflake implemented something called the **HyperLogLog cardinality estimation algorithm**, which returns an **approximation of the distinct number of values** of a column.

HLL()

HLL\_ACCUMULATE()

HLL\_COMBINE()

HLL\_ESTIMATE()

HLL\_EXPORT()

HLL\_IMPORT()

```
SELECT APPROX_COUNT_DISTINCT(L_ORDERKEY) FROM LINEITEM;
```

Output: 1,491,111,415  
Execution Time: 44 Seconds

```
SELECT COUNT(DISTINCT L_ORDERKEY) FROM LINEITEM;
```

Output: 1,500,000,000  
Execution Time: 4 Minutes 20 Seconds

# Similarity Estimation



Snowflake have implemented a two-step process to **estimate similarity**, without the need to compute the intersection or union of two sets.

1

```
SELECT MINHASH(5, C_CUSTKEY) FROM CUSTOMER;
```

Output:

```
|"MINHASH(5, C_CUSTKEY)"|  
|{  
|  "state": [  
|    557181968304,  
|    67530801241,  
|    1909814111197,  
|    8406483771,  
|    34962958513  
|  ],  
|  "type": "minhash",  
|  "version": 1  
|}
```

# Similarity Estimation

1

Snowflake have implemented a two-step process to **estimate similarity**, without the need to compute the intersection or union of two sets.

2

```
SELECT APPROXIMATE_SIMILARITY(MH) FROM
(
  (SELECT MINHASH(5, C_CUSTKEY) MH FROM CUSTOMER)
  UNION
  (SELECT MINHASH(5, O_CUSTKEY) MH FROM ORDERS)
);
```

Output:

"APPROXIMATE_SIMILARITY(MH)"
0.8

# Frequency Estimation



Snowflake have implemented a family of functions using the **Space-Saving algorithm** to produce an **estimation of values and their frequencies**.

APPROX\_TOP\_K

APPROX\_TOP\_K\_ACCUMULATE

APPROX\_TOP\_K\_COMBINE

APPROX\_TOP\_K\_ESTIMATE

```
SELECT APPROX_TOP_K(P_SIZE, 3, 100000) FROM PART;
```

Output:

```
|"APPROX_TOP_K(P_SIZE, 3, 100000)" |
| [[13,401087],[38,401074],[35,401033]] |
```



# Frequency Estimation



Snowflake have implemented a family of functions using the **Space-Saving algorithm** to produce an estimation of values and their frequencies.

APPROX\_TOP\_K

```
SELECT P_SIZE, COUNT(P_SIZE) AS C FROM PART
GROUP BY P_SIZE
ORDER BY C DESC
LIMIT 3;
```

Output:

"P_SIZE"	"C"
13	401,087
38	401,074
35	401,033

# Percentile Estimation



Snowflake have implemented the **t-Digest algorithm** as an efficient way of **estimating approximate percentile values** in data sets.

APPROX\_PERCENTILE

APPROX\_PERCENTILE\_ACCUMULATE

APPROX\_PERCENTILE\_COMBINE

APPROX\_PERCENTILE\_ESTIMATE

```
INSERT OVERWRITE INTO TEST_SCORES VALUES (23),(67),(2),(3),(9),(19),(45),(81),(90),(11);  
SELECT APPROX_PERCENTILE(score, 0.8) FROM TEST_SCORES;
```

Output:

"APPROX_PERCENTILE(score,0.8)"
74

# Table Sampling

# Table Sampling



Table sampling is a convenient way to read a random subset of rows from a table.

## Fraction-based

```
SELECT * FROM LINEITEM TABLESAMPLE/SAMPLE [samplingMethod] (<probability>);
```

Row

$\frac{<p>}{100 * n}$

Block

```
SELECT * FROM LINEITEM TABLESAMPLE/BLOCK (50);
```

```
SELECT * FROM LINEITEM TABLESAMPLE/ROW (50);
```

```
SELECT * FROM LINEITEM SAMPLE (50) REPEATABLE/SEED (765);
```

0 to 2147483647C

# Table Sampling



Table sampling is a convenient way to read a random subset of rows from a table.

## Fixed-size

```
SELECT * FROM LINEITEM TABLESAMPLE/SAMPLE (<num> ROWS);
```

```
SELECT L_TAX, L_SHIPMODE FROM LINEITEM SAMPLE BERNOULLI/ROW (3 rows);
```

## Output:

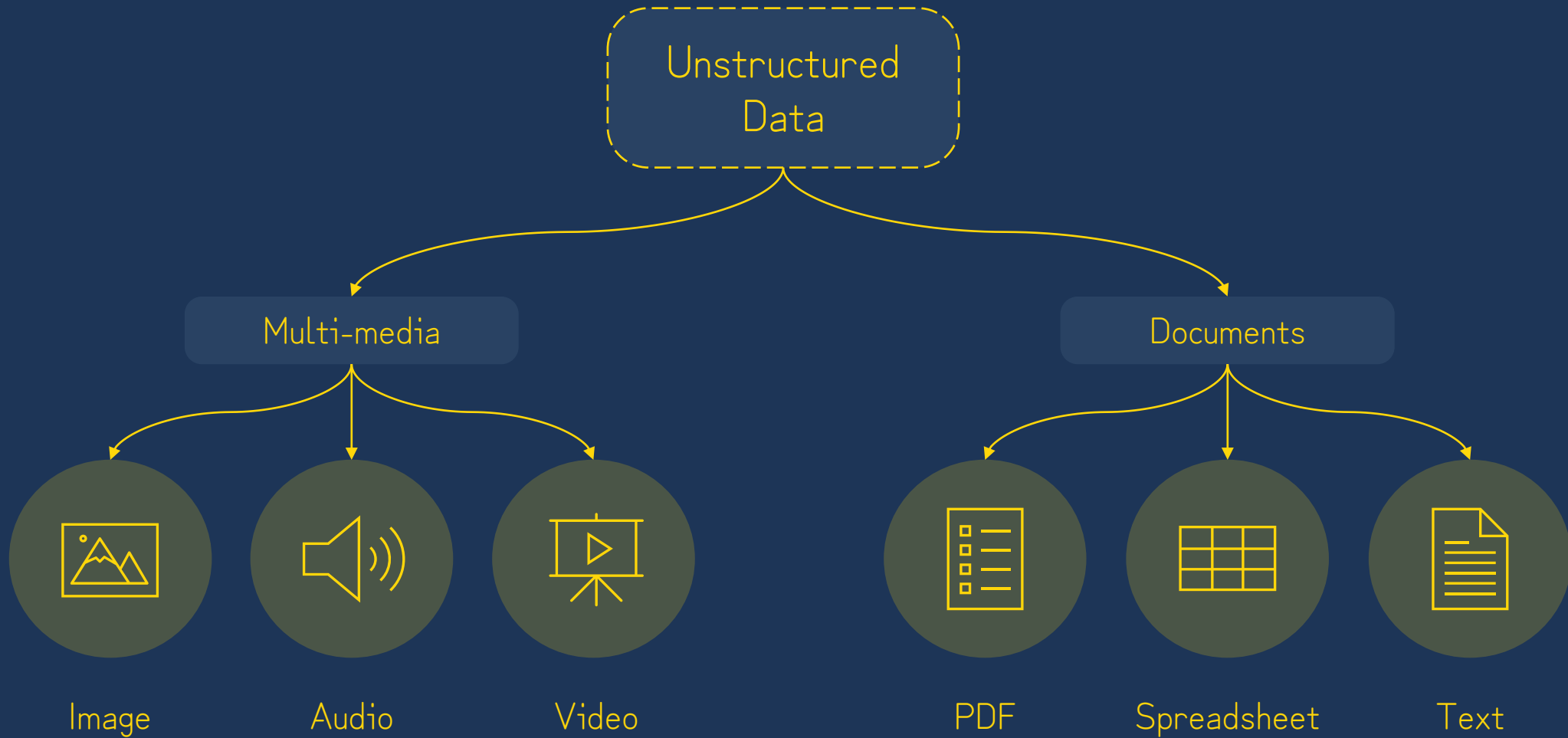
"L_TAX"	"L_SHIPMODE"
0.02	REG AIR
0.02	TRUCK
0.06	TRUCK



Fixed-size sampling does not support block sampling and use of seed. Adding these will result in an error.

# Unstructured Data File Functions

# Unstructured Data



# File Functions

BUILD\_SCOPED\_FILE\_URL

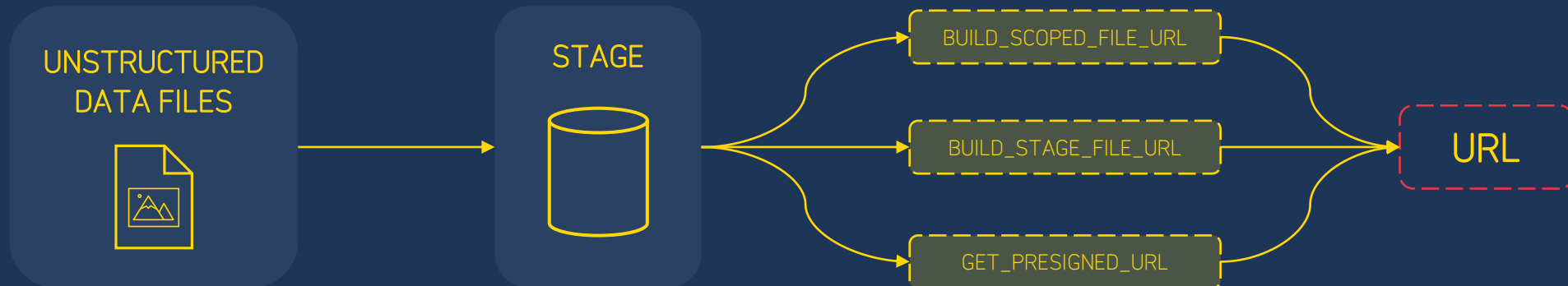
BUILD\_STAGE\_FILE\_URL

GET\_PREIGNED\_URL

GET\_STAGE\_LOCATION

GET\_RELATIVE\_PATH

GET\_ABSOLUTE\_PATH



```
PUT file://image.jpg @images_stage
```

```
SELECT GET_PREIGNED_URL(@images_stage, image.jpg)
```



# BUILD\_SCOPED\_FILE\_URL

```
build_scoped_file_url( @<stage_name> , '<relative_file_path>' )
```

 URL valid for 24 hours.

```
SELECT build_scoped_file_url(@images_stage, 'prod_z1c.jpg');
```

Output:

```
|“BUILD_SCOPED_FILE_URL(@images_stage, ‘prod_z1c.jpg’);”|
|https://go44755.eu-west-2.aws.snowflakecomputing.com/api/files/|
|01a691df-0000-277e-0000-9485000bc022/163298951696390/5fGgfDJX6kvA|
|qZx6tUJNjWDXEu%2f8%2b7a%2fqQ5HFpCKKMs81o1MC5NSLKPzC6p2hy670VChIC7o|
|Po2JwrY8%2fAQ13fVjwXtxs40Uf76eUDVH7G1Uz0f5ugveSR6qAQF60EV7y2F9e9cn|
|RWHBMncTyGuyCxd4gxtVSyXRQuQ7s2qBsh6%2bt0Yj4LNs0hjQFmD3EPgfGQ7P81gY|
|z2p%2fFyRcFX4V|
```



When this function is called in a query the role must have **USAGE** privileges on an external named stage and **READ** privileges on an internal named stage.

# BUILD\_SCOPED\_FILE\_URL



When this function is called in a UDF, Stored Procedure or View the calling role does not require privileges on the underlying stage.

```
CREATE VIEW PRODUCT_SCOPED_URL_VIEW AS  
SELECT build_scoped_file_url(@images_stage, 'prod_z1c.jpg') AS scoped_file_url;
```

```
SELECT * FROM PRODUCT_SCOPED_URL_VIEW;
```

Output:

SCOPED_FILE_URL
https://go44755.eu-west-2.aws.snowflakecomputing.com/api/files/ 01a691df-0000-277e-0000-9485000bc022/163298951696390/5fGgfDJX6kvA qZx6tUJNjWDXEu%2f8%2b7a%2fqQ5HFPCCKMs81o1MC5NSLKPzC6p2hy670VChIC7[...]

# BUILD\_STAGE\_FILE\_URL

```
build_stage_file_url( @<stage_name> , '<relative_file_path>' )
```



The file URL  
does not expire.

```
SELECT build_stage_file_url(@images_stage, 'prod_z1c.jpg');
```

Output:

```
| "BUILD_STAGE_FILE_URL(@images_stage, 'prod_z1c.jpg')"|  
|https://go44755.eu-west-2.aws.snowflakecomputing.com/api/files/DEMO_DB/DEMO_SCHEMA/IMAGES_STAGE/prod_z1c.jpg |
```



Calling this function whether it's part of a query, UDF, Stored Procedure or View requires privileges on the underlying stage, that is **USAGE** for external stages and **READ** for internal stages.

```
CREATE STAGE MY_STAGE  
ENCPTION = (TYPE = 'SNOWFLAKE_SSE');
```

```
ALTER STAGE MY_STAGE SET  
ENCPTION = (TYPE = 'SNOWFLAKE_SSE');
```

TOM BAILEY COURSES

tombaileycourses.com

# GET\_PREIGNED\_URL

```
get_presigned_url( @<stage_name> , '<relative_file_path>' , [ <expiration_time> ] )
```

```
SELECT get_presigned_url(@images_stage, 'prod_z1c.jpg', 600);
```

Output:

```
|GET_PREIGNED_URL(@images_stage, 'prod_z1c.jpg',600)|  
|-----|  
|https://sfc-uk-ds1-6-customer-stage.s3.eu-west-2.amazonaws.com/vml0-s-|  
|ukst8973/stages/42763db5-31fa-47ef-bdb1-729d81b645c2/tree.jpg?X-Amz-Algorithm=AWS4-|  
|HMAC-SHA256&X-Amz-Date=20220827T162316Z&X-Amz-SignedHeaders=host&X-Amz-Expires=599&X-|  
|Amz-Credential=AKIA4ANG2XQCHAHQKBKS%2F20220827%2Feu-west-2%2Fs3%2Faws4_request&X-Amz-|  
|Signature=0e8edf89acc24d9bd23b5387f8938671f54212be1c50c1bfe26c974ea151af55|
```




Calling this function whether it's part of a query, UDF, Stored Procedure or View requires privileges on the underlying stage, that is **USAGE** for external stages and **READ** for internal stages.

# GET\_PREIGNED\_URL

@DOCUMENTS\_STAGE

@images\_stage/document.pdf

@images\_stage/document\_metadata.json



```
{
  "relative_path": "document.pdf",
  "author": "Corado Fernandez",
  "published_on": "2022-01-23",
  "topics": [
    "nutrition",
    "health",
    "science"
  ]
}
```

# GET\_PREIGNED\_URL

```
CREATE TABLE document_metadata  
(  
    relative_path string,  
    author string,  
    published_on date,  
    topics array  
);
```

```
COPY INTO document_metadata  
FROM  
(SELECT  
    $1:relative_path,  
    $1:author,  
    $1:published_on::date,  
    $1:topics  
FROM  
@documents_stage/document_metadata.json)  
FILE_FORMAT = (type = json);
```

# GET\_PREIGNED\_URL

```
CREATE VIEW document_catalog AS
(
  SELECT
    author,
    published_on,
    get_presigned_url(@documents_stage, relative_path) as presigned_url,
    topics
  FROM
    documents_table
);
```

```
SELECT * FROM document_catalog;
```

Output:

AUTHOR	PUBLISHED_ON	PRESIGNED_URL	TOPICS
Corado Fernandez	2022-01-23	https://sfc-uk-ds1-6-customer-stage.[...]	["nutrition","health","science"]

# Directory Tables



# Directory Tables

Internal



Directory Table

External



Directory Table

```
CREATE STAGE INT_STAGE  
DIRECTORY = (ENABLE = TRUE)
```

```
ALTER STAGE EXT_STAGE SET  
DIRECTORY = (ENABLE = TRUE)
```

```
SELECT * FROM DIRECTORY(@INT_STAGE)
```

RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG	FILE_URL
document.pdf	250,838	55:42.0	ba247312[...]	f76b4327[...]	https://go44755.eu-west-2.aws.snowflake[...]

# Refreshing Directory Tables



Directory Tables must be refreshed to reflect the most up-to-date changes made to stage contents. This includes new files uploaded, removed files and changes to files in the path.

Internal



External



```
ALTER STAGE INT_STAGE REFRESH;
```

# Refreshing Directory Tables

**i** Directory Tables must be refreshed to reflect the most up-to-date changes made to stage contents. This includes **new files uploaded, removed files and changes to files in the path.**

External



- 1 Enable a directory table on an external stage.

```
CREATE STAGE INT_STAGE  
DIRECTORY = (ENABLE = TRUE)
```

- 2 Describe stage and retrieve ARN for Snowflake managed SQS Queue in field **directory\_notification\_channel**.

```
DESCRIBE STAGE EXT_STAGE;
```

- 3 Configure event notifications for a S3 bucket to notify Snowflake managed SQS queue associated with the stage when new or updated data is available to read into the directory table metadata.

# File Support REST API

# File Support REST API

**GET /api/files**

Scoped URL

File URL

**BUILD\_SCOPED\_FILE\_URL();**

**BUILD\_STAGE\_FILE\_URL();**

## Authorization

Only the user who generated the scoped URL can download the staged file.

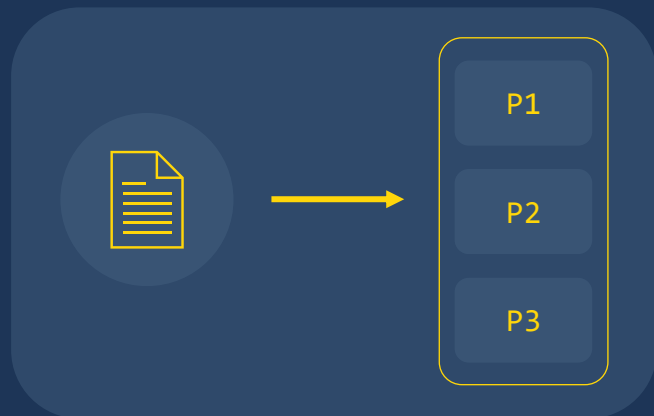
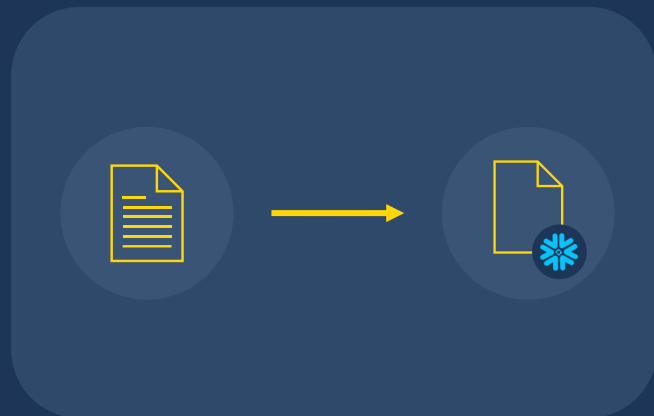
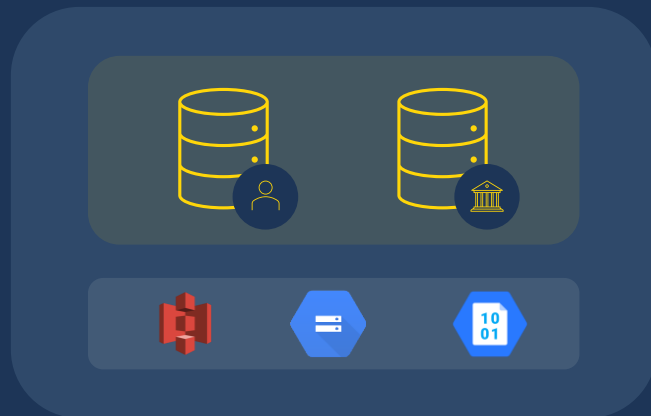
Any role that has privileges on the underlying stage can access the file.

# File Support REST API

```
1  import requests
2
3  response = requests.get("https://go44755.eu-west-2.aws.snowflakecomputing.com/api/files/DB/SCHEMA/STAGE/img.jpg",
4
5      headers={
6          "User-Agent": "reg-tests",
7          "Accept": "*/*",
8          "X-Snowflake-Authorization-Token-Type": "OAUTH",
9          "Authorization": """"Bearer {}""".format(token)
10     },
11     allow_redirects=True)
12
13 print(response.status_code)
14
15 print(response.content)
```

# Storage Layer Overview

# Storage Summary





# Micro-partitions

# Micro-partitions

order_id	item_id	order_date
001	2456ASTT	01/06/2022
002	098098SS	01/06/2022
003	TT778GH2	01/06/2022
004	JX098FJ32	02/06/2022
005	TF098SD32	02/06/2022
006	CC098FJ32	02/06/2022

```
COPY INTO MY_CSV_TABLE FROM  
@MY_STAGE/orders.csv
```

## Micro-partition 1

001	002	003
2456ASTT	098098SS	TT778GH2
01/06/2022	01/06/2022	01/06/2022

## Micro-partition 2

004	005	006
JX098FJ32	TF098SD32	CC098FJ32
02/06/2022	02/06/2022	02/06/2022

# Micro-partitions

Snowflake partitions along the natural ordering of the input data as it is inserted/loaded.

Micro-partitions are the physical file stored in blob storage and they range in size from **50-500mb** of uncompressed data.

Micro-partitions undergo a reorganisation process into the Snowflake columnar data format.

Micro-partitions are immutable, they are write once and read many.

## Micro-partition 1

001	002	003
2456ASTT	098098SS	TT778GH2
01/06/2022	01/06/2022	01/06/2022

## Micro-partition 2

004	005	006
JX098FJ32	TF098SD32	CC098FJ32
02/06/2022	02/06/2022	02/06/2022

# Micro-partition Metadata

Micro-partition 1

001	002	003
2456ASTT	098098SS	TT778GH2
01/06/2022	01/06/2022	01/06/2022

Partition Metadata

MIN:	1	MAX:	3
MIN:	098098SS	MAX:	TT778GH2
MIN:	01/06/2022	MAX:	01/06/2022

# Micro-partition Pruning

MY\_CSV\_TABLE

Micro-partition 1 001-100

Micro-partition 2 101-200

Micro-partition 3 201-300

Micro-partition 4 301-400

Micro-partition 5 401-500

Micro-partition 6 501-600

Micro-partition metadata allows Snowflake to optimize a query by first checking the min-max metadata of a column and discarding micro-partitions from the query plan that are not required.

```
SELECT ORDER_ID, ITEM_ID
FROM MY_CSV_TABLE
WHERE ORDER_ID > 360 AND ORDER_ID < 460;
```

The metadata is typically considerably smaller than the actual data, speeding up query time.

# Time Travel & Fail-Safe

# Data Lifecycle



# Time Travel



Time Travel enables users to restore objects such as tables, schemas and databases that have been removed.

```
UNDROP DATABASE MY_DATABASE;
```

Time Travel enables users to analyse historical data by querying it at points in the past.

```
SELECT * FROM MY_TABLE AT(TIMESTAMP =>  
TO_TIMESTAMP('2021-01-01'));
```

Time Travel enables users to create clones of objects from a point in the past.

```
CREATE TABLE MY_TABLE_CLONE CLONE MY_TABLE  
AT (TIMESTAMP => TO_TIMESTAMP('2021-01-01'));
```



# Time Travel Retention Period

The Time Travel retention period is configured with the parameter: `DATA_RETENTION_TIME_IN_DAYS`.

```
ALTER DATABASE MY_DB  
SET DATA_RETENTION_TIME_IN_DAYS=90;
```

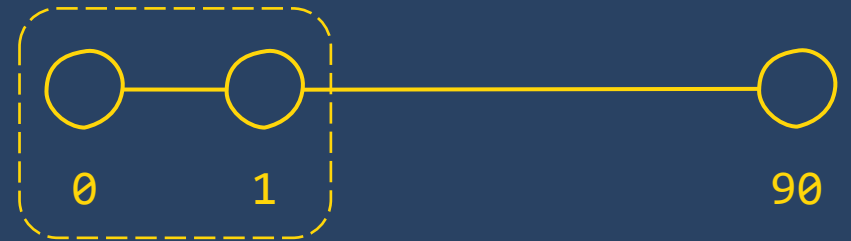
The default retention period on the account, database, schema and table level is **1 day**.

On the Standard edition of Snowflake the minimum value is 0 and maximum is 1 day and for Enterprise and higher the maximum is increase from 1 to 90.

Temporary and transient objects can have a max retention period of 1 day across all editions.

Enterprise Edition

Standard Edition



Temporary



Transient



TOM BAILEY COURSES

[tombaileycourses.com](https://tombaileycourses.com)

# Accessing Data In Time Travel

## AT

```
SELECT * FROM MY_TABLE  
AT(STATEMENT =>  
'01a00686-0000-0c47');
```

The AT keyword allows you to capture historical data inclusive of all changes made by a statement or transaction up until that point.

Three parameters are available to specify a point in the past:

- TIMESTAMP
- OFFSET
- STATEMENT

## BEFORE

```
SELECT * FROM MY_TABLE  
BEFORE(STATEMENT =>  
'01a00686-0000-0c47');
```

The BEFORE keyword allows you to select historical data from a table up to, but not including any changes made by a specified statement or transaction.

One parameter is available to specify a point in the past:

- STATEMENT

## UNDROP

```
UNDROP TABLE MY_TABLE;  
UNDROP SCHEMA MY_SCHEMA;  
UNDROP DATABASE MY_DATABASE;
```

The UNDROP keyword can be used to restore the most recent version of a dropped table, schema or database.

If an object of the same name already exists an error is returned.

To view dropped objects you can use:  
SHOW TABLES HISTORY;

# Fail-safe



Fail-safe is a non-configurable period of 7 days in which historical data can be recovered by contacting Snowflake support.



It could take several hours or days for Snowflake to complete recovery.

Fail-safe is only enabled for permanent objects, not temporary or transient.

# Cloning

# Cloning

```
CREATE TABLE MY_TABLE (  
  COL_1 NUMBER COMMENT 'COLUMN ONE',  
  COL_2 STRING COMMENT 'COLUMN TWO',  
);  
  
CREATE TABLE MY_TABLE_CLONE CLONE MY_TABLE;
```

```
CREATE STAGE MY_EXT_STAGE  
  URL='S3://RAW/FILES/'  
  CREDENTIALS=();  
  
CREATE STAGE MY_EXT_STAGE_CLONE CLONE  
  MY_EXT_STAGE;
```

```
CREATE FILE FORMAT MY_FF  
  TYPE=JSON;  
  
CREATE FILE FORMAT MY_FF_CLONE CLONE MY_FF;
```

Cloning is the process of creating a copy of an existing object within a Snowflake account.

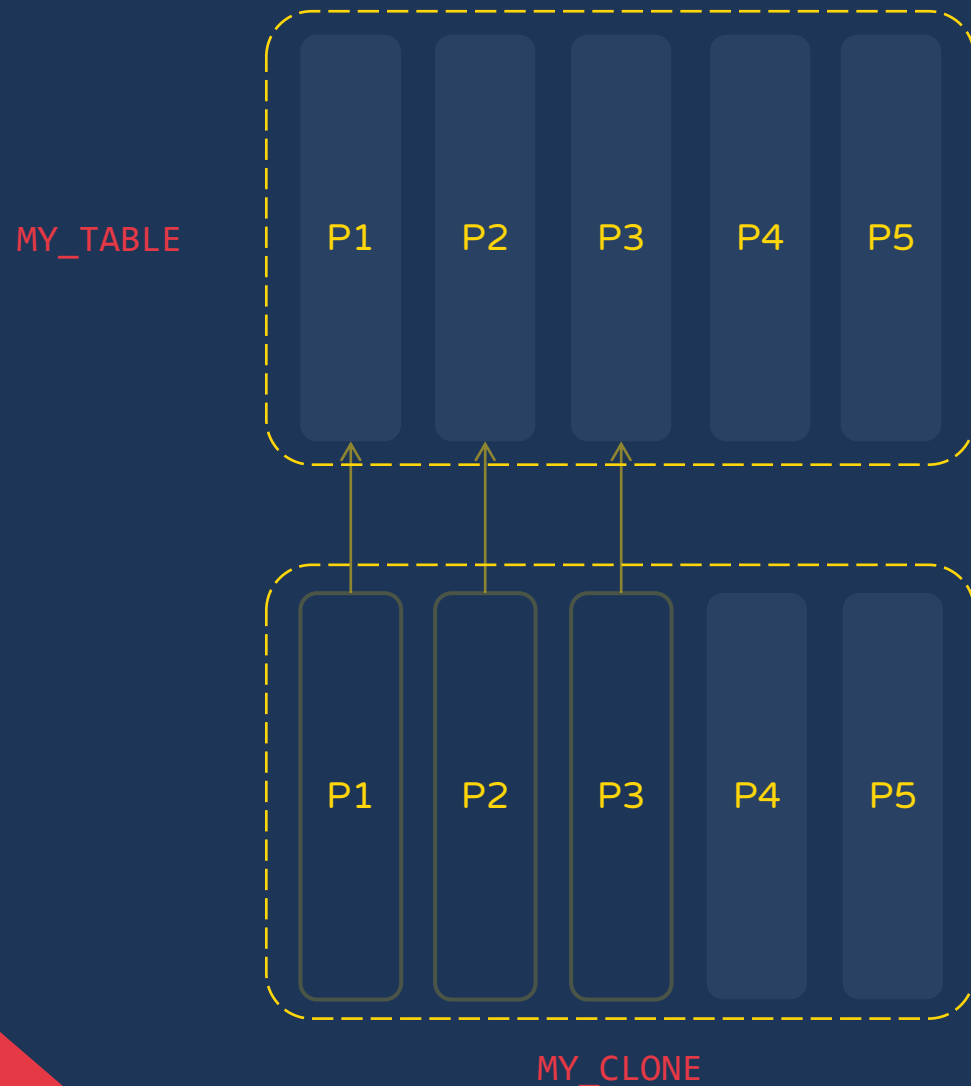
Users can clone:

- DATABASES
- SCHEMAS
- TABLES
- STREAMS
- STAGES
- FILE FORMATS
- SEQUENCES
- TASKS
- PIPES (reference external stage only)

Cloning is a metadata only operation, copying the properties, structure and configuration of it's source.

Cloning does not contribute to storage costs until data is modified or new data is added to the clone.

# Zero-copy Cloning



```
CREATE TABLE MY_CLONE CLONE MY_TABLE;
```

Changes made after the point of cloning then start to create additional micro-partitions.

Changes made to the source or the clone are not reflected between each other, they are completely independent.

Clones can be cloned with nearly no limits.

Because cloning is a meta-data only operation it's very quick, enabling interesting use-case, such as rapid integration testing.

# Cloning Rules

①

A cloned object does not retain the privileges of the source object, with the exception of tables.

②

Cloning is recursive for databases and schemas.

③

External tables and internal named stages are never cloned.

④

A cloned table does not contain the load history of the source table.

⑤

Temporary and transient tables can only be cloned as temporary or transient tables, not permanent tables.

# Cloning With Time Travel

```
CREATE TABLE MY_TABLE_CLONE CLONE MY_TABLE  
AT (TIMESTAMP => TO_TIMESTAMP('2022-01-01'));
```

Time Travel and Cloning features can be combined to create a clone of an existing database, schema non-temporary table and stream at a point within their retention period.

```
CREATE TABLE MY_TABLE_CLONE CLONE MY_TABLE  
AT (OFFSET => -60*30);
```

If the source object did not exist at the time specified in the AT | BEFORE parameter an error is thrown.



# Replication

# Replication

ORG\_1

account1.eu-west-2.aws

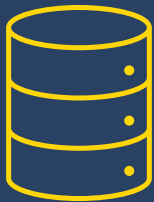


DB\_1



DB\_2

account2.eu-central-1.aws



DB\_1\_REPLICA

Replication is a feature in Snowflake which enables replicating databases between Snowflake accounts within an organization.

A database is selected to serve as the primary database from which secondary databases can be created in other accounts:

```
ALTER DATABASE DB_1  
ENABLE REPLICATION TO ACCOUNTS ORG1.account2;
```

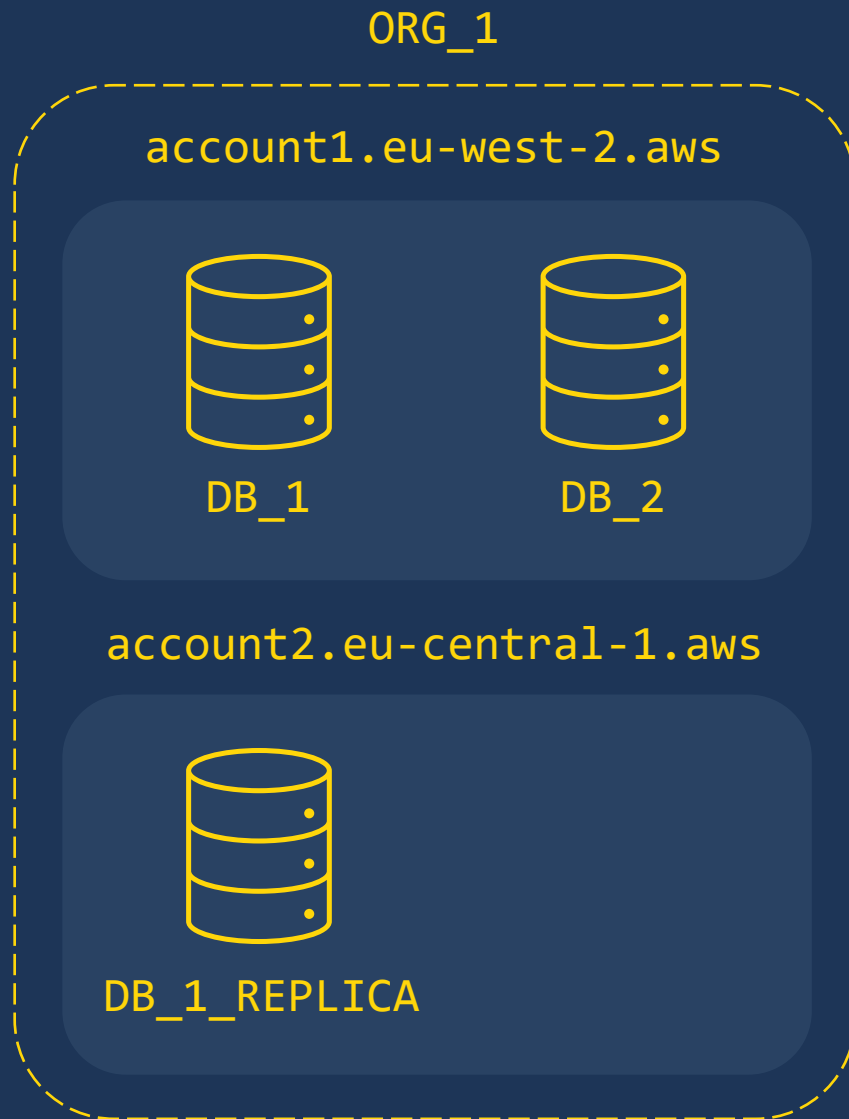
When a primary database is replicated a snapshot of it's database objects and data are transferred to the secondary database:

```
CREATE DATABASE DB_1_REPLICA  
AS REPLICA OF ORG_1.account1.DB_1;
```

The secondary database can be periodically refreshed:

```
ALTER DATABASE DB_1_REPLICA REFRESH;
```

# Replication



①

External Tables, Stages, Pipes, Streams and Tasks are not currently replicated.

②

Refreshing a secondary database can be automated by configuring a task object to run the refresh command on a schedule.

③

Only databases and some of their child objects can be replicated, not users, roles, warehouses, resource monitors or shares.

④

Privileges granted to database objects are not replicated to the secondary database.

⑤

Billing for database replication is based on data transfer and compute resources.

# Storage Billing

# Storage Billing

Data storage cost is calculated monthly based on the average number of on-disk bytes for all data stored each day in a Snowflake account.



The monthly costs for storing data in Snowflake is based on a flat rate per Terabyte.

\$42 per Terabyte per month  
for customers deployed in  
AWS – EU (London)

# Data Storage Pricing

Table 3: Storage Pricing		
Cloud Provider	Region	On Demand Storage Pricing (TB/mo)
AWS	Europe (London)	\$42.00
AWS	AP Mumbai	\$46.00
Azure	North Europe (Ireland)	\$40.00
GCP	Europe West 2 (London)	\$40.00
GCP	US Central 1 (Iowa)	\$35.00

① Storage cost is determined by **Cloud Provider, Region & Pricing Plan**.

# Data Storage Billing Scenarios

Table 3: Storage Pricing		
Cloud Provider	Region	On Demand Storage Pricing (TB/mo)
AWS	Europe (London)	\$42.00
AWS	AP Mumbai	\$46.00
Azure	North Europe (Ireland)	\$40.00
GCP	Europe West 2 (London)	\$40.00
GCP	US Central 1 (Iowa)	\$35.00

## Storage

If an average of 15TB is stored during a month on account deployed in AWS Europe (London) Region it will cost \$630.00 (Nov 2021)

If an average of 14TB is stored during a month on account deployed in AWS AP Mumbai Region it will cost \$644.00 (Nov 2021)

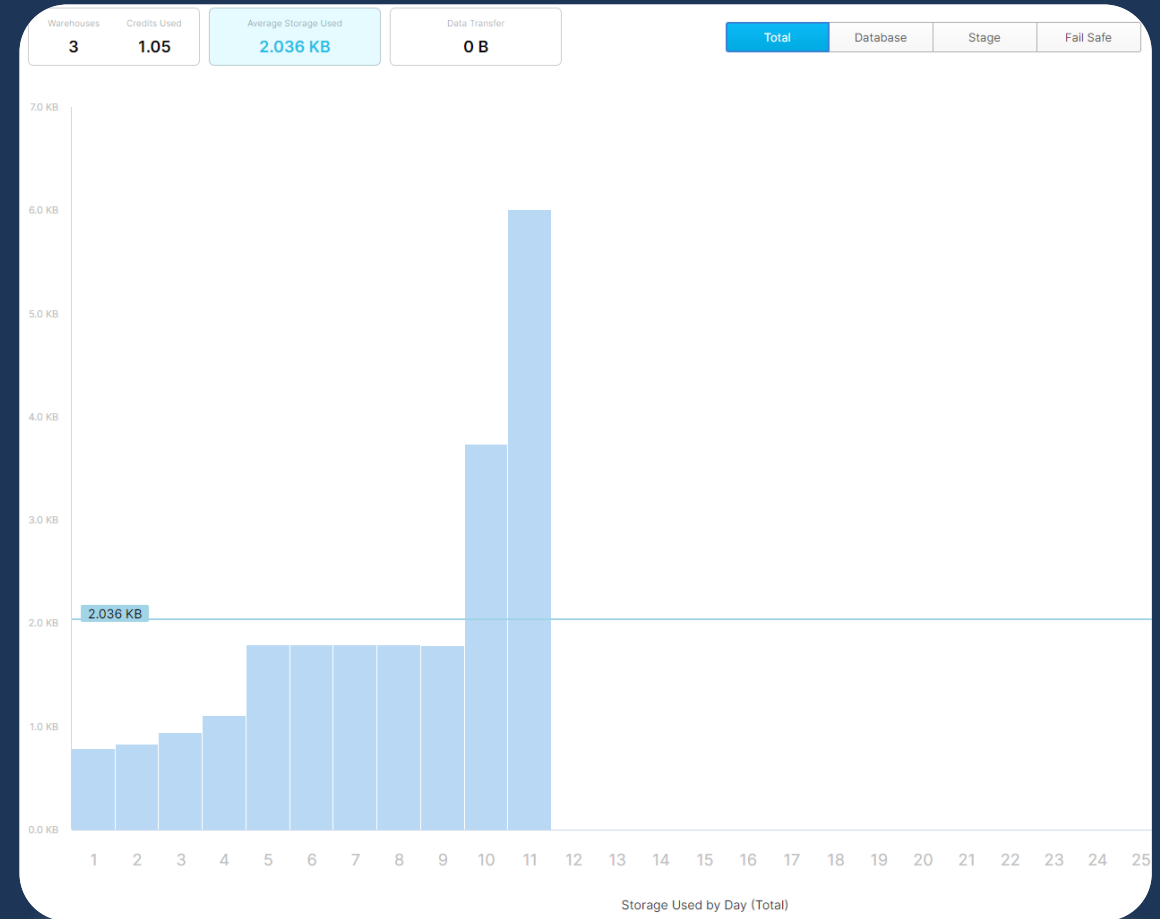
# Storage Monitoring

Data storage usage can be monitored from the Classic and Snowsight user interfaces.

Equivalent functionality can be achieved via the account usage views and information schema views and table functions.

Account Usage Views:

- DATABASE\_STORAGE\_USAGE
- STAGE\_STORAGE\_USAGE\_HISTORY
- TABLE\_STORAGE\_METRICS

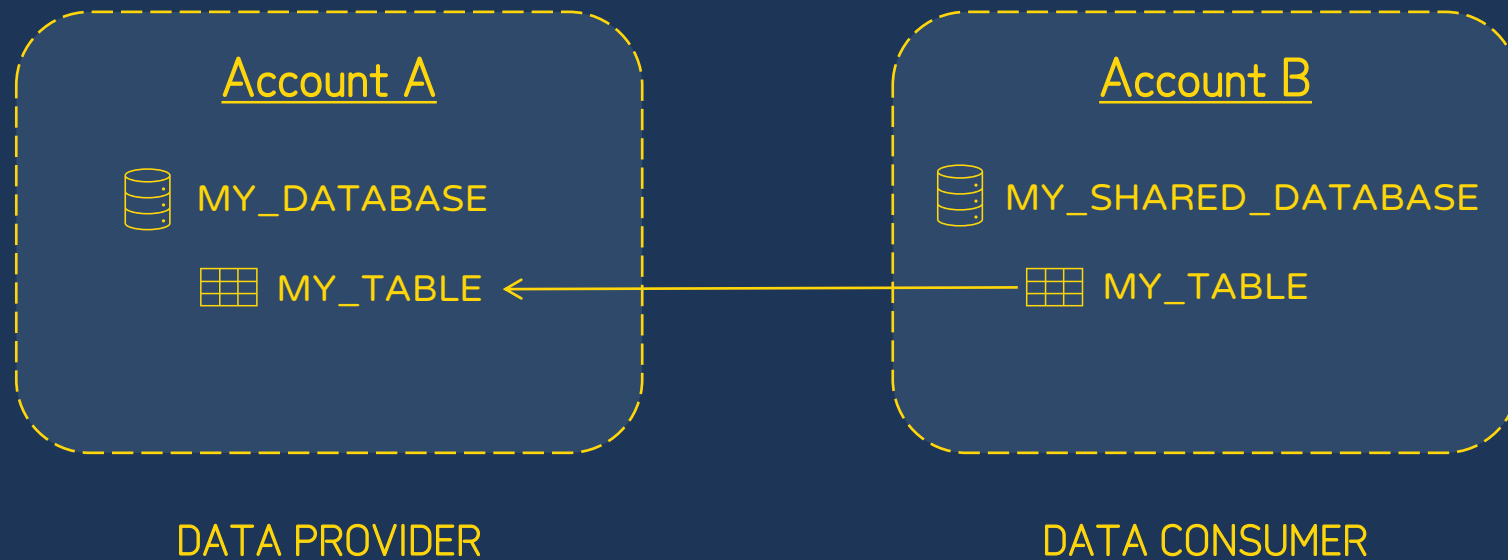




# Data Sharing

# Secure Data Sharing

Secure Data Sharing allows an account to provide read-only access to selected database objects to other Snowflake accounts without transferring data.



# Secure Data Sharing

Sharing is enabled with an account-level **SHARE** object. It is created by a data provider and consists of two key configurations:

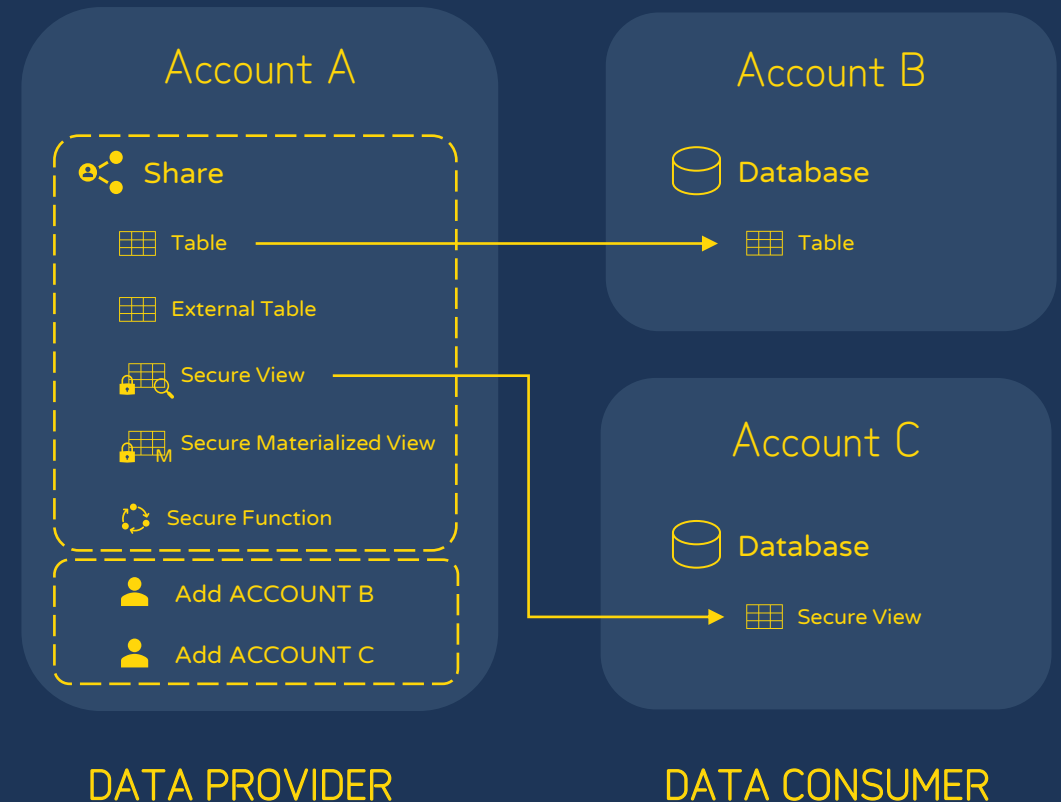
- Grants on database objects
- Consumer account definitions

An account can share the following database objects:

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

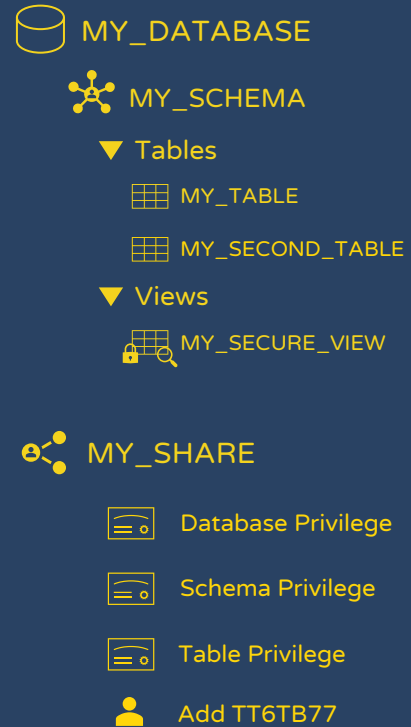
Data consumers create a database from a SHARE which contains the read-only database objects granted by the data provider.

Sharing is not available on the **VPS edition** of Snowflake.



# Secure Data Sharing Example

<https://GYU889T>



PROVIDER ACCOUNT

```
-- Create empty share (CREATE SHARE privilege required)
CREATE SHARE MY_SHARE;

-- Sharing objects
GRANT USAGE ON DATABASE MY_DATABASE TO SHARE MY_SHARE;
GRANT USAGE ON SCHEMA MY_DATABASE.MY_SCHEMA TO SHARE MY_SHARE;
GRANT USAGE ON TABLE MY_DATABASE.MY_SCHEMA.MY_TABLE TO SHARE MY_SHARE;

-- Add accounts
ALTER SHARE MY_SHARE ADD ACCOUNTS=TT67B77;
```

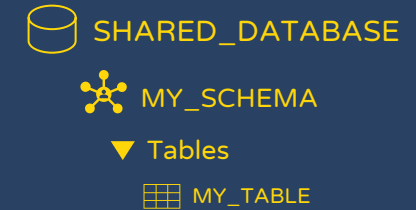
```
-- LIST AVAILABLE SHARES (IMPORT SHARE privilege required)
SHOW SHARES;

-- Create database from share (IMPORT SHARE privilege required)
CREATE DATABASE SHARED_DATABASE FROM SHARE GYU889T.MY_SHARE;

-- ACCOUNT ADMIN LISTS CONTENTS OF AVAILABLE SHARE
DESC SHARE GYU889T.MY_SHARE;

-- Query shared table
SELECT * FROM SHARED_DATABASE.MY_TABLE;
```

<https://TT67B77>



CONSUMER ACCOUNT

# Data Provider

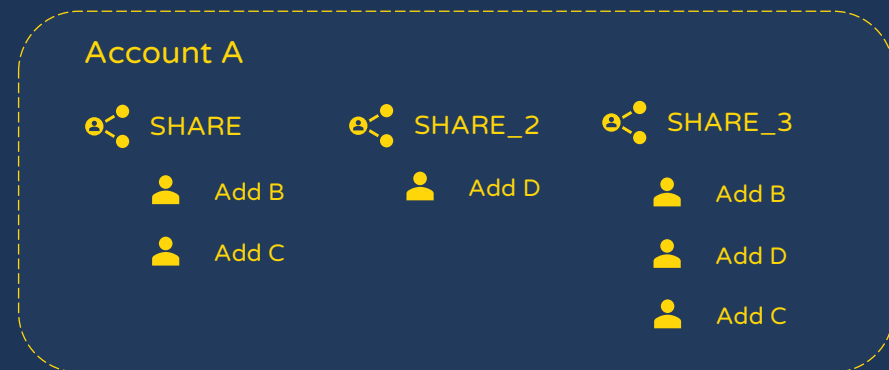
Database objects added to a share become immediately available to all consumers.



Only **one** database can be added per share.

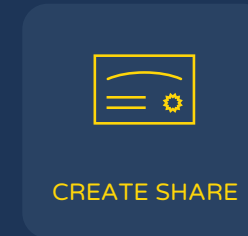
SQL compilation error: Database 'TEST\_DB' does not belong to the database that is being shared.

No hard limits on the number of shares you can create or the number of accounts you can add to a share.



# Data Provider

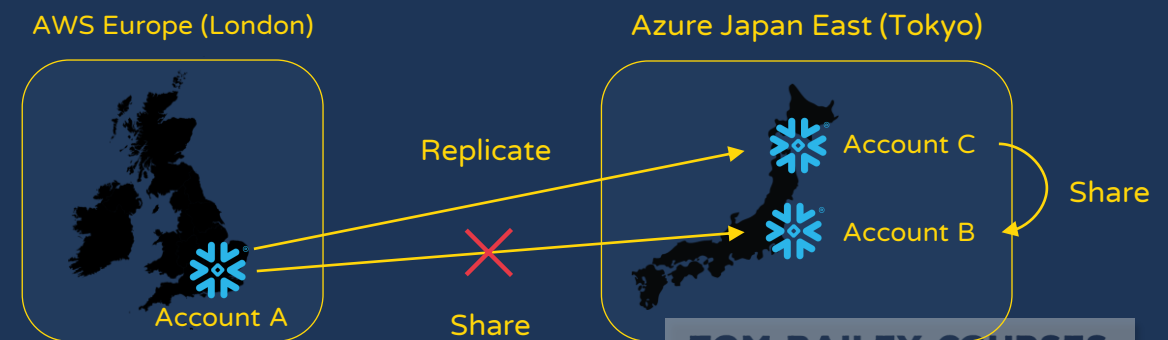
To execute a create share command a user must have a role with the **CREATE SHARE** privilege granted.



Access to a share or database objects in a share can be revoked at any time.



A share can only be granted to accounts in the same region and cloud provider as the data provider account.



# Data Consumer

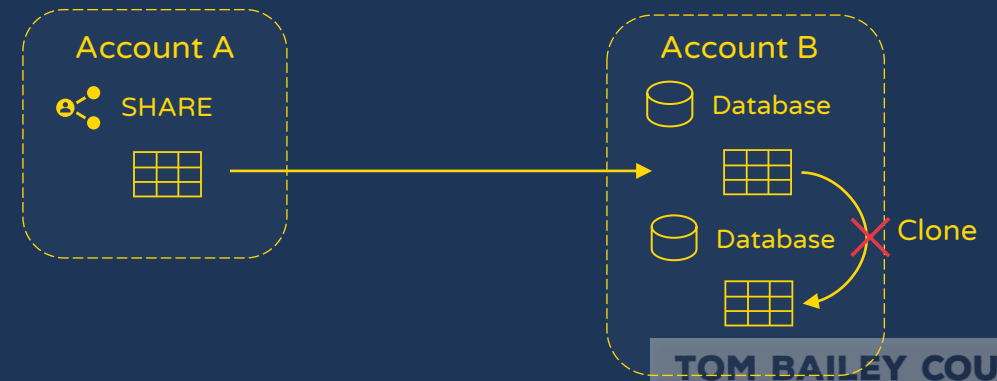
Only one database can be created per share.

Importing more than once is not supported. Database is already imported as 'SNOWFLAKE\_SAMPLE\_DATA'.

A data consumer cannot use the Time Travel feature on shared database objects.

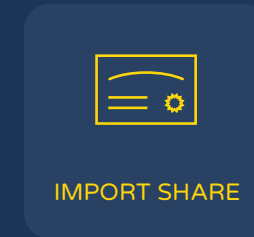


A data consumer cannot create a clone of a shared database or database objects.



# Data Consumer

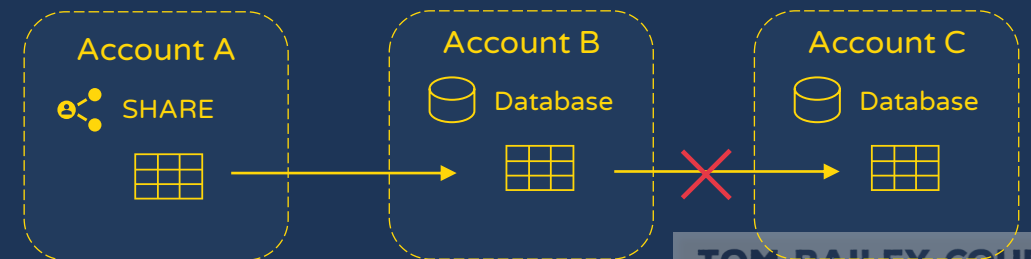
To create a database from a share a user must have a role with the **IMPORT SHARE** privilege granted.



Data consumers cannot create objects inside the shared database.



Data consumers cannot reshare shared database objects.





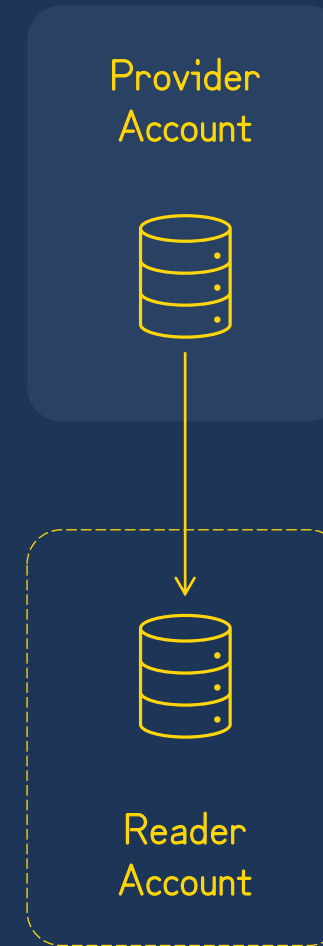
# Reader Account

A reader account provides the ability for non-Snowflake customers to gain access to a providers data.

Reader accounts can't insert data or copy data into an account.

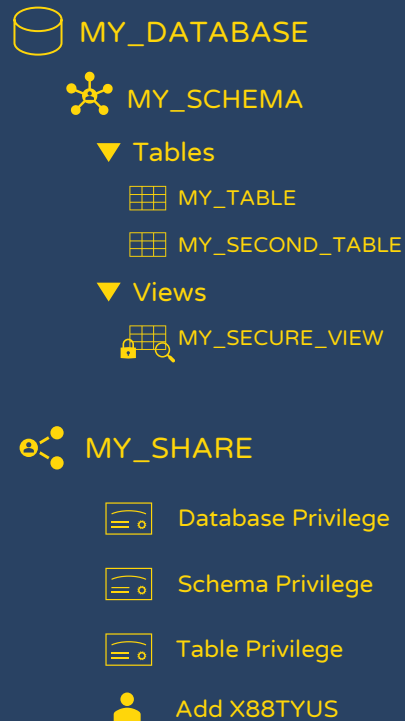
A reader account can only consume data from the provider account that created it.

Provider accounts assume responsibility for the reader account they create and are billed for their usage.



# Reader Account

<https://GYU889T>



PROVIDER ACCOUNT

```
-- CREATE MANAGED ACCOUNT privilege required

CREATE MANAGED ACCOUNT READER_ACCT1
  ADMIN_NAME=user1,
  ADMIN_PASSWORD='Sdfed43da!44',
  TYPE=READER;

-- Create empty share (CREATE SHARE privilege required)

CREATE SHARE MY_SHARE;

-- Sharing objects

GRANT USAGE ON DATABASE MY_DATABASE TO SHARE MY_SHARE;
GRANT USAGE ON SCHEMA MY_DATABASE.MY_SCHEMA TO SHARE MY_SHARE;
GRANT USAGE ON TABLE MY_DATABASE.MY_SCHEMA.MY_TABLE TO SHARE MY_SHARE;

-- Add accounts

ALTER SHARE MY_SHARE ADD ACCOUNTS=X88TYUS;
```

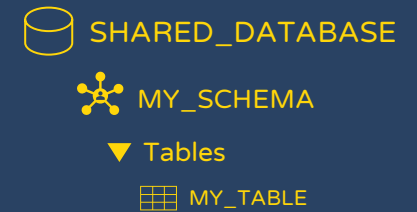
```
-- IMPORT SHARE privilege required

CREATE DATABASE SHARED_DATABASE FROM SHARE GYU889T.MY_SHARE;

-- Query shared table

SELECT * FROM SHARED_DATABASE.MY_TABLE;
```

<https://X88TYUS>



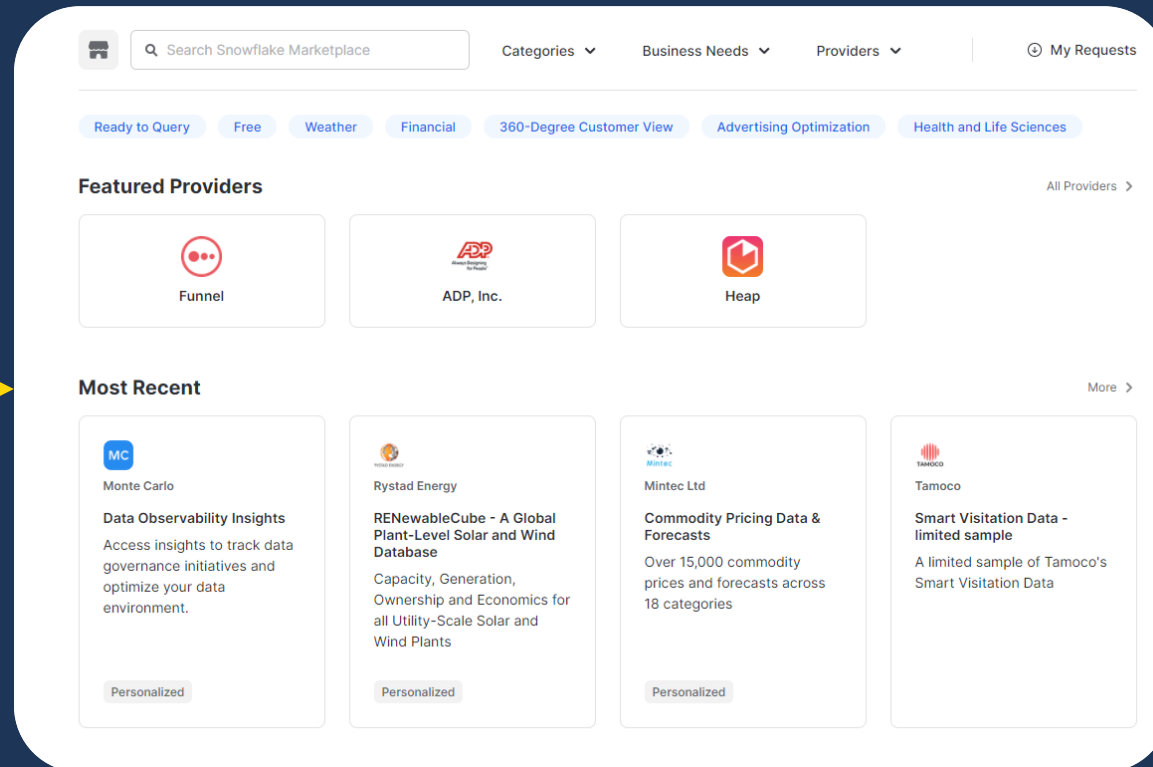
READER ACCOUNT

TOM BAILEY COURSES

tombaileycourses.com

# Data Marketplace

Portal for browsing publicly available third-party shares.



Data  
Provider

⇒ Via the Provider Studio  
Data Provides can list  
**standard** or **personalised**  
shares.

Data  
Consumer

⇒ Third-party dataset  
immediately available to  
query from account without  
transformation.

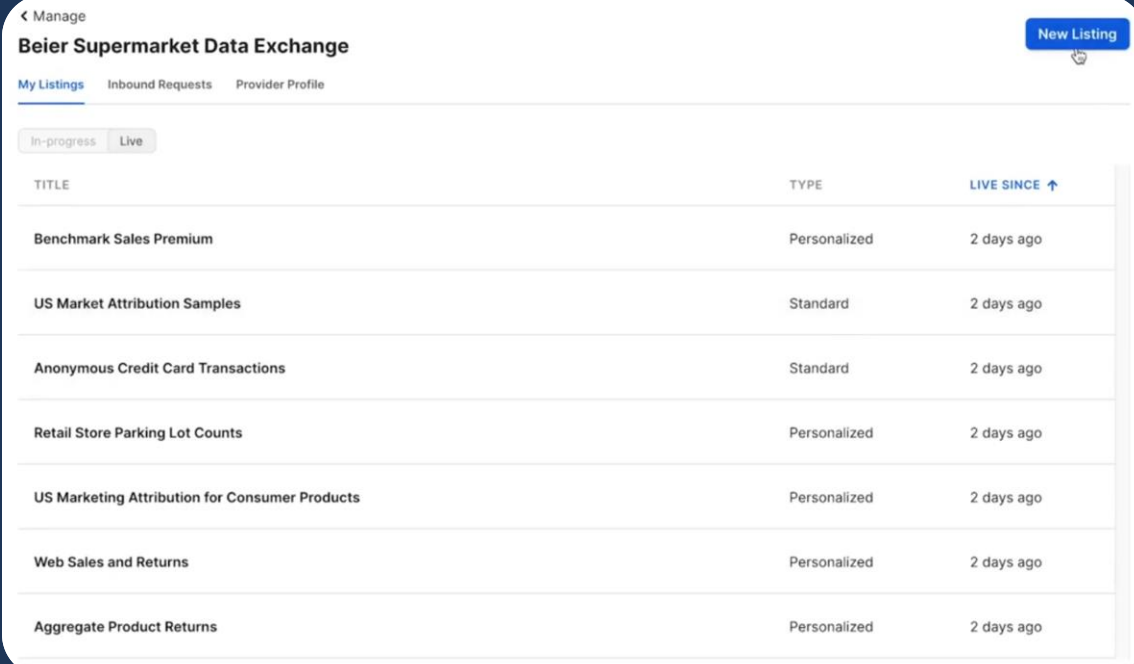
# Video 22: Data Exchange

# Data Exchange

A Data Exchange is a private version of the data marketplace for accounts to provide and consume data.

A Data Exchange for a Snowflake account is set up with Snowflake support by providing a name and description.

The Snowflake account hosting a Data Exchange is the Data Exchange Admin and can manage members, designate members as providers and consumers and manage listing requests.



The screenshot shows the 'Beier Supermarket Data Exchange' interface. At the top, there's a 'Manage' link and a 'New Listing' button. Below the title, there are tabs for 'My Listings', 'Inbound Requests', and 'Provider Profile'. Under 'My Listings', there are filters for 'In-progress' and 'Live'. The main table lists data exchange items with columns for Title, Type, and Live Since.

TITLE	TYPE	LIVE SINCE ↑
Benchmark Sales Premium	Personalized	2 days ago
US Market Attribution Samples	Standard	2 days ago
Anonymous Credit Card Transactions	Standard	2 days ago
Retail Store Parking Lot Counts	Personalized	2 days ago
US Marketing Attribution for Consumer Products	Personalized	2 days ago
Web Sales and Returns	Personalized	2 days ago
Aggregate Product Returns	Personalized	2 days ago