O'REILLY®

Snowflake architecture

Snowflake's three layers of architecture

Tomas Sobotik

Learning objectives

By the end of this live, hands-on 2-part series, you'll understand:

- Snowflake architecture
- Virtual Warehouse management
- Data loading features & best practices
- Security concepts
- Data protection features
- Basic Data engineering & data governance features
- Scaling possibilities and caching in Snowflake
- Data sharing
- Billing
- Basics of performance tuning

Tomas Sobotik

https://www.linkedin.com/in/tomas-sobotik/

- Based in Czech Republic
- 2 kids
- 15+ years in data industry
- Data engineer and architect
- Big Snowflake fan and enthusiast
- Snowflake Data Superhero
- Snowflake Subject Matter Expert

ARCHITECTURE

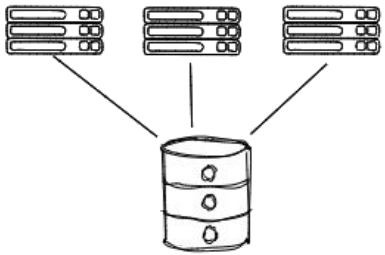
Snowflake in a nutshell

- Founded in 2012 and publicly launched in 2014
- Software as a Service (SaaS)
  - No license to buy, no HW to maintain
- Only public cloud (AWS, Azure, GCP)
- Separated storage and compute
  - Possibility to scale each part independently
- Per second billing (pay as you go)
- Originally with DWH in the cloud workload -> platformization

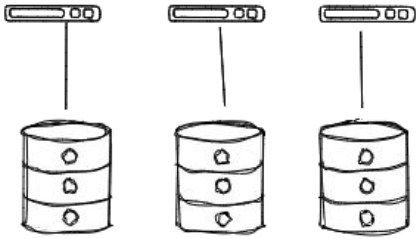
Next level of DWH architecture

Storage and compute separated

Traditional architectures



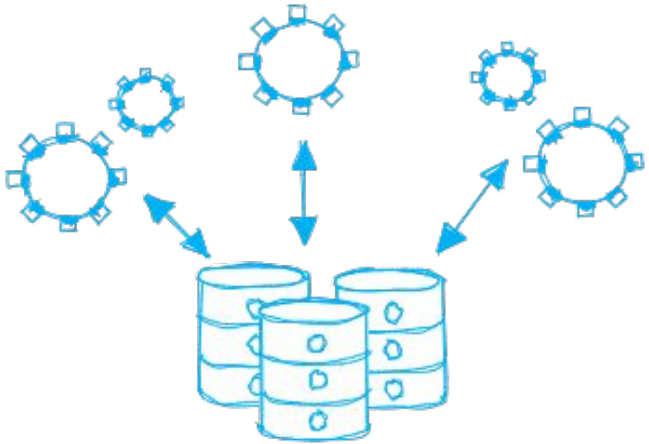
"Shared-disk"
Shared storage
Single Server



"Shared-nothing"
Decentralized, local storage
Single cluster
MPP APPLIANCE

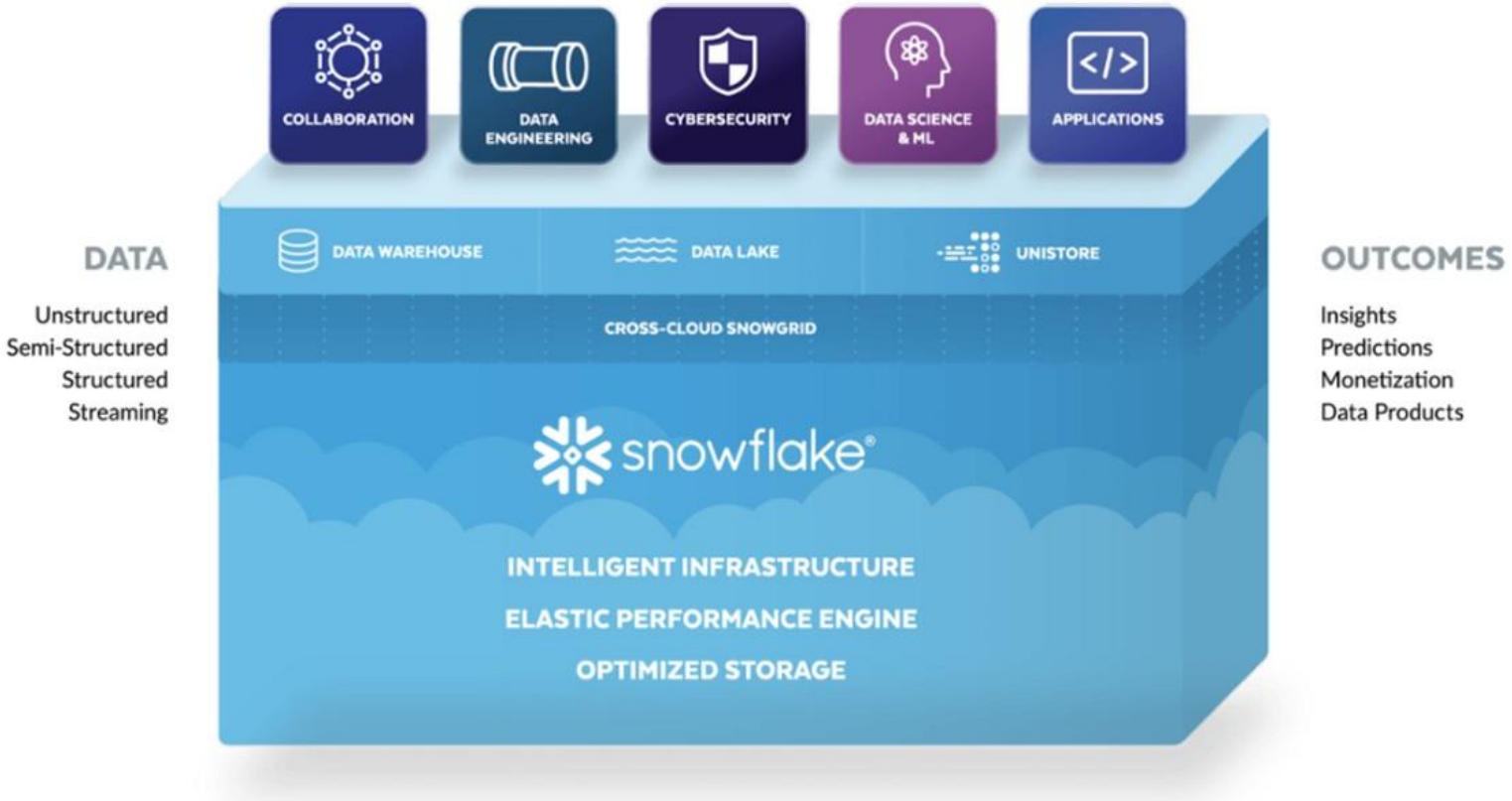
Teradata, Netezza, Vertica
Redshift

Data Cloud

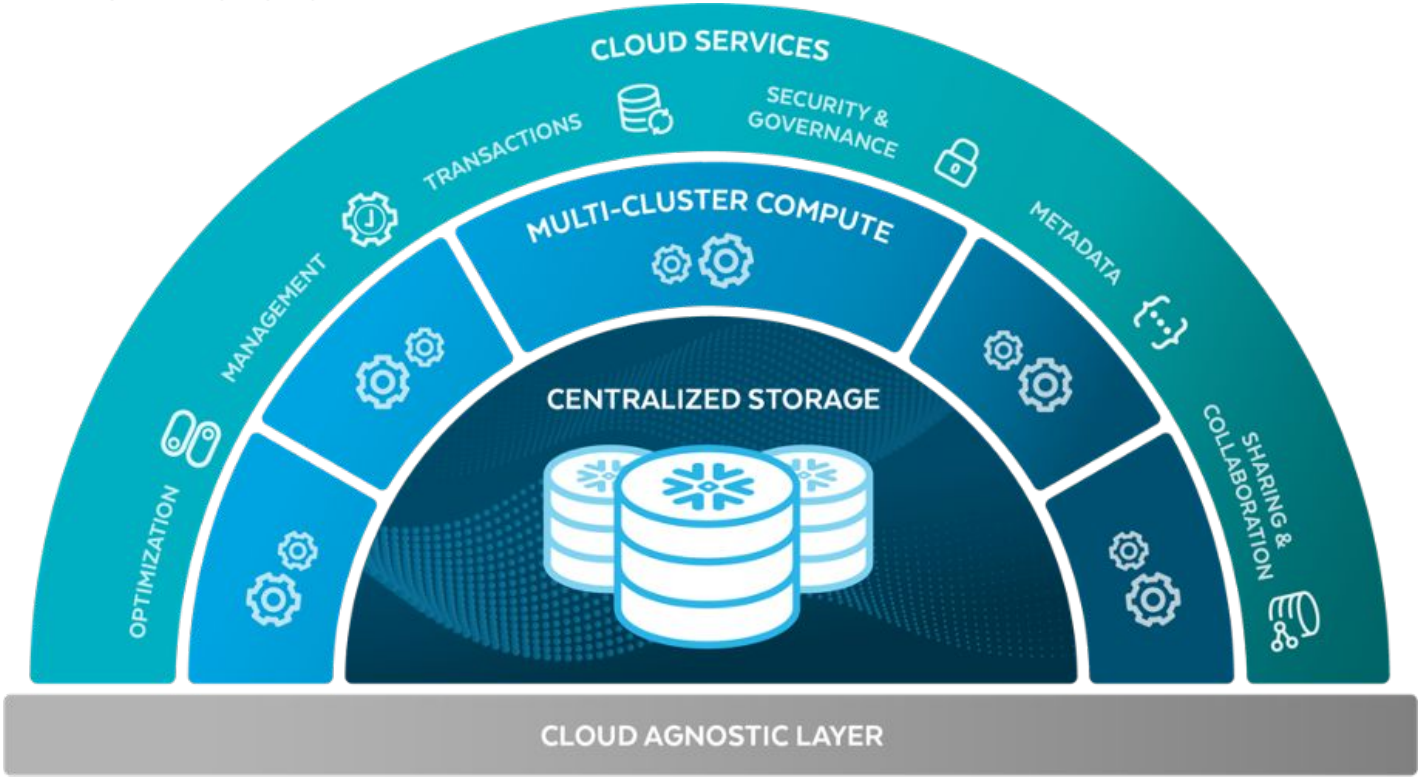


"Multi-cluster, shared data"
Centralized, elastic storage
Multiple MPP, independent compute clusters
UNLIMITED Storage and Compute

Global view


COLLABORATION
DATA ENGINEERING
CYBERSECURITY
DATA SCIENCE & ML
APPLICATIONS
DATA
Unstructured
Semi-Structured
Structured
Streaming
DATA WAREHOUSE
DATA LAKE
UNISTORE
CROSS-CLOUD SNOWGRID
snowflake
INTELLIGENT INFRASTRUCTURE
ELASTIC PERFORMANCE ENGINE
OPTIMIZED STORAGE
OUTCOMES
Insights
Predictions
Monetization
Data Products

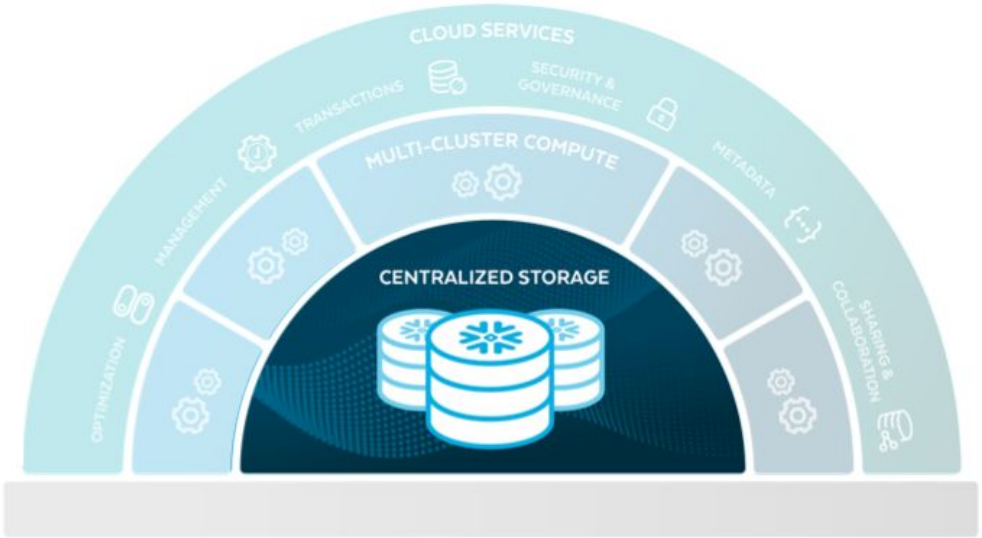
Under the hood



Google Cloud aws Azure

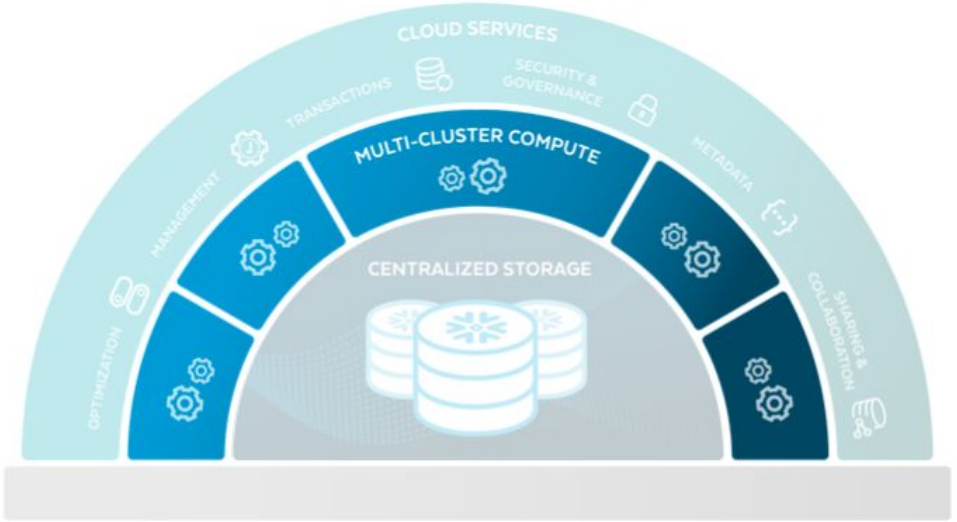
Data Storage

- Centralized storage
- Up to 5x compression
- Native support for semi structured data
- Data are organized in micro partitions
- Zero-copy cloning
- Time travel
- Storage is small part of the overall bill
- All storage within Snowflake is billable



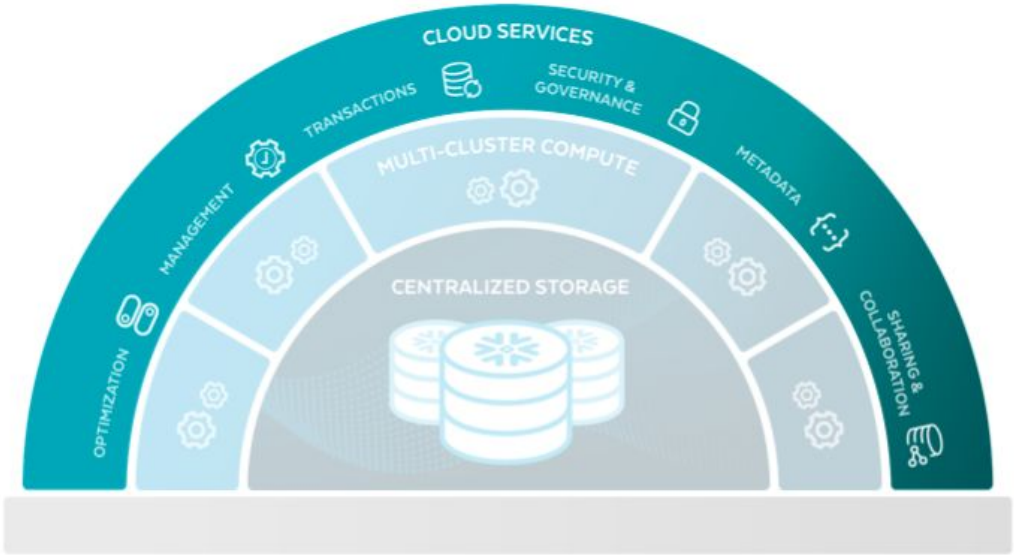
Compute

- On-demand virtual warehouses
  - Elastic (turn on/off instantly)
  - Scaling up - bigger
    - T shirt sizes
  - Scaling out - more
    - multi cluster warehouses
- Suspend
- Per second billing
- Most spendings is for virtual warehouses run time



Cloud services

- Compute that does not run on virtual warehouse
  - Show commands
  - Query result cache
- Serverless features
  - Snowpipe
  - Auto-clustering
  - Maintenance of MV
  - Cross - cloud replication



Snowflake editions

Standard

- Complete SQL Data Warehouse
- Secure Data Sharing across regions / clouds
- Premier support 24/7
- 1 day of time travel
- Always-on encryption in transit and at rest
- Customer dedicated virtual warehouses

Enterprise

Standard +

- Multi-Cluster warehouse
- Up to 90 days of time travel
- Annual rekey of all encrypted data
- Materialized views

Business Critical

Enterprise +

- HIPAA support
- PCI compliance
- Data encryption everywhere
- Tri-Secret Secure using customer-managed keys (AWS)
- Enhanced security policy

Virtual Private Snowflake (VPS)

Business Critical +

- Customer dedicated virtual servers wherever the encryption key is in memory
- Customer dedicated metadata store
- Additional operational visibility

Snowflake Supported Regions

US West (Oregon)
US East (Ohio)
US East (N. Virginia)
Canada Central (Montreal)
US Gov West 1
US East (Commercial Gov – N Virginia)
EU (Ireland)
Europe (London)
EU (Frankfurt)
EU (Stockholm)
Asia Pacific (Tokyo)
Asia Pacific (Osaka)
Asia Pacific (Seoul)
AWS Pacific (Mumbai)
Asia Pacific (Singapore)
Asia Pacific (Sydney)
South American (Sao Paulo)

West US 2 (Washington)
Central US (Iowa)
South Central US (Texas)
East US 2 (Virginia)
Canada Central (Toronto)
US Gov (Virginia)
North Europe (Ireland)
West Europe (Netherlands)
Switzerland North (Zurich)
UAE North (Dubai)
Japan East (Tokyo)
Central India (Pune)
Southeast Asia (Singapore)
Australia East (New South Wales)

US Central 1 (Iowa)
US East 4 (N. Virginia)
Europe West 2 (London)
Europe West 4 (Netherlands)

Summary

- Storage, compute, cloud services – 3 layers of architecture
- Independent layers
  - resize can be done on the fly
- Snowflake AI Data Cloud
  - Cloud platform supporting various use cases and workloads

Knowledge check

1. How many layers has Snowflake architecture?
   a) 2
   b) 3
   c) 4

2. Where can you run Snowflake? - Mark all correct options
   a) Public cloud
   b) Private cloud
   c) Hybrid Cloud
   d) On prem

3. How do you pay for Snowflake? - Mark all correct options
   a) Need to buy a license
   b) Usage based pricing model
   c) Need to buy servers

4. What is Snowflake Storage architecture
   a) Centralized storage
   b) Decentralized Storage
   c) Shared storage

5. Does Snowflake store data compressed?
   a) True
   b) False

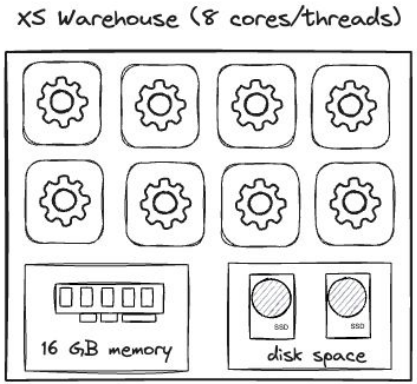
6. How many running virtual warehouse can you have?
   a) 3
   b) 5
   c) Almost unlimited

7. Can you change the size of virtual warehouse?
   a) True
   b) False

Virtual Warehouses

Virtual Warehouses

- A named wrapper around a cluster of servers with CPU, memory, and disk
  - A compute cluster = logical collection of VMs (1+)
- The larger the warehouse, the more servers in the cluster
- Each size up doubles compute resources
- Jobs requiring compute run on virtual warehouse
- While running, a virtual warehouse consumes Snowflake credits



Warehouse sizes

Warehouse Size	Servers	Credits / Hours	Credits / Second
X-Small	1	1	0.003
Small	2	2	0.006
Medium	4	4	0.0011
Large	8	8	0.0022
X-Large	16	16	0.0044
2X-Large	32	32	0.0089
3X-Large	64	64	0.0178
4X-Large	128	128	0.0356
5X-Large*	256	256	0.0711
6x-Large*	512	512	0.1422

* Preview feature in some regions

Benefits

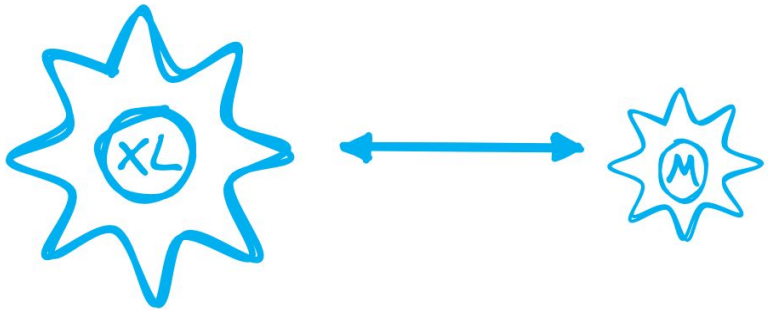
- **Instant provisioning**
  - with few clicks/commands you can enable required amount of computing power based on the size or type of your workload
- **On-demand performance**
  - instantly scale up or down the resource to get required performance
- **On-demand concurrency**
  - multi-cluster warehouse to support your concurrent queries
- **Per-second pricing**
  - predictable cost
- **Faster performance**
  - use large warehouse for shorter periods of time and keep the same cost
- **Reduced cost**
  - idle warehouses could be suspended, pay only for running times

Warehouses operations

- Start / Stop
  - could be automated via warehouse parameters
- AUTO_SUSPEND
  - warehouse will be suspended after specified period of time
  - enabled by default
- AUTO_RESUME
  - warehouse is automatically resumed when any statement that requires a warehouse is submitted
  - current session needs to have specified warehouse
  - enabled by default
- Resizing
  - could be done anytime, even when WH is running and queries are being executed
  - ALTER WAREHOUSE Command or via UI
  - Effects
    - Suspended warehouse will start at new size when it will be resumed
    - Running warehouse: immediate impact
      - running queries - finish with current size
      - new queries run at new size

Resizing

- Can be done any time, even when warehouse is running
- No impact on running processes
- Can be done either via UI or with `ALTER WAREHOUSE` statement
- How it works
  - Suspended warehouse will start at new size when resumed again
  - Running warehouse will finish running queries at actual size, new ones run at new size



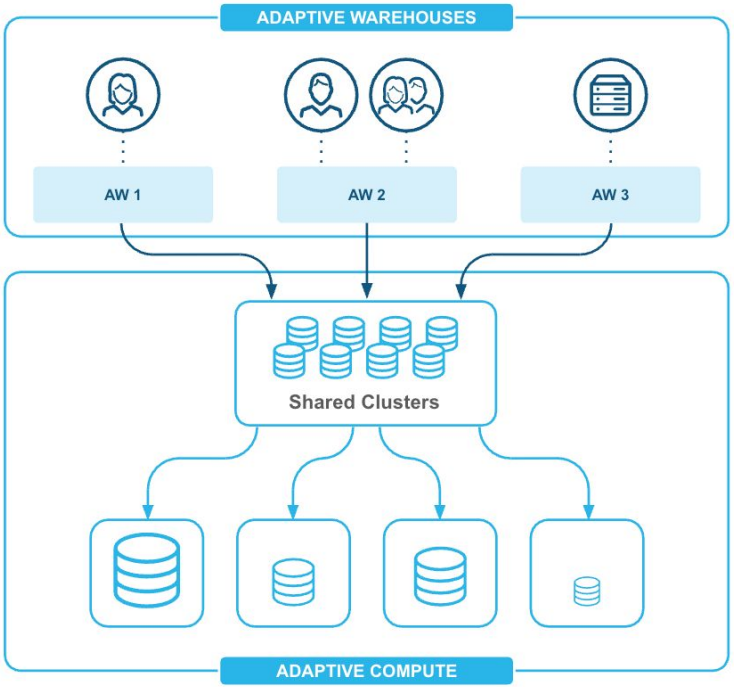
Generation 2 warehouses

- Updated version of the current standard virtual warehouse
- Newest hardware focused on better performance for analytics and data engineering workloads
- Most of the queries should finish faster
- Limited availability to some some regions only for now
- You can update gen1 to gen2 and other way around
- They are about 30% more expensive than gen1 (1.35 credits for XS size)
- How to create gen2 warehouse

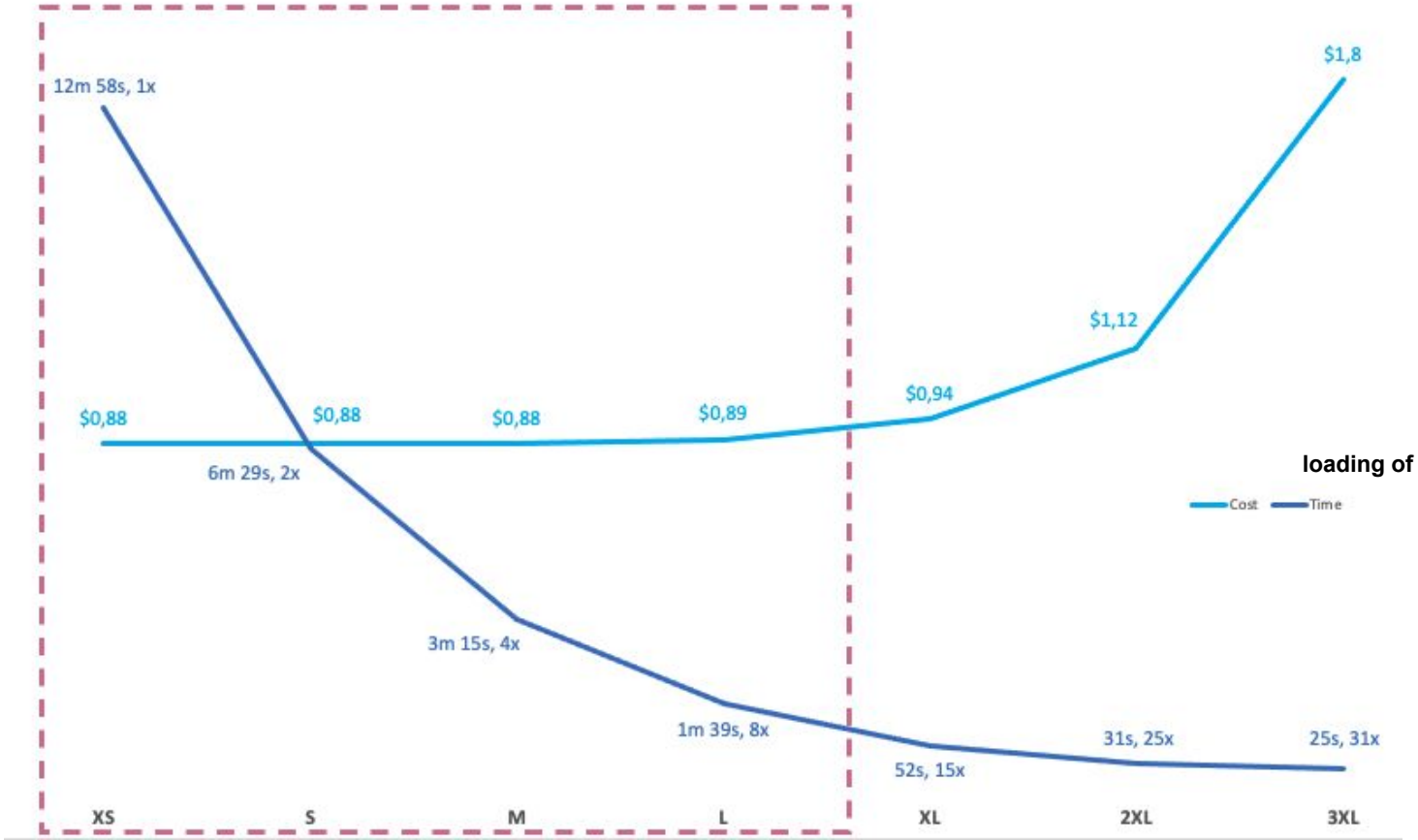
```
CREATE OR REPLACE WAREHOUSE next_gen_wh
RESOURCE_CONSTRAINT = STANDARD_GEN_2
WAREHOUSE_SIZE = SMALL;
```

New Adaptive Compute – in preview

- Another simplification
- New warehouse type – Adaptive Warehouse
- Minimal configuration
  - No cluster size selection
  - Automatic suspend/resume
- No need to think about `COMPUTE_WH`
  - Size
  - Concurrency settings
  - Multi-cluster settings
- Better price/performance ratio
- Intelligent query routing
- Always running pool of clusters



Bigger does not mean more expensive



Exercise

- Create your first virtual warehouse called `COMPUTE_WH`, with following parameters:
  - Extra small size
  - Enabled auto resume
  - Enabled auto suspend
  - Suspend after 1 minute
  - It will be suspended after creation

- Use SQL for virtual warehouse creation
- Use `SYSADMIN` role for object creation
- Let's create `COMPUTE_WH` as gen2 version as well

Billing

Cost drivers

- SaaS solution - no license or HW to buy
- Usage based pricing
- Billing is done per layer - storage, compute, cloud services
- Base invoiceable unit is called Snowflake Credits
  - Used to pay for the consumption of resources
  - Consumed only when some resource is used

Credit price components

Snowflake Edition

Standard, Enterprise,
Business Critical and
Virtual Private Snowflake

Cloud Provider

aws

Google Cloud

Azure

Snowflake is cloud agnostic,
available on all major clouds

Region

Global presence with
cross region, cross cloud
replication
or data collaboration

Storage costs

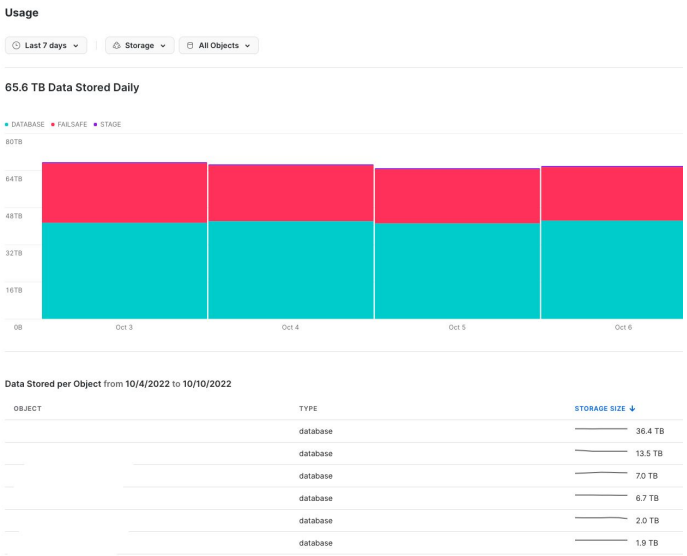
- Based on average amount of terabyte storage used per month
- Calculation is based on compressed and ingested data into Snowflake
- Ingested data are always compressed, encrypted and columnrised
  - 5x - 10x compression based on data types
- Calculated as daily average amount
- Storage costs includes
  - Data itself
  - Time Travel Data

Where to find Data Usage

Snowsight

Admin -> Usage

ACCOUNTADMIN role is needed



SQL

Table functions (Information Schema)

`DATABASE_STORAGE_USAGE_HISTORY`

`STAGE_STORAGE_USAGE_HISTORY`

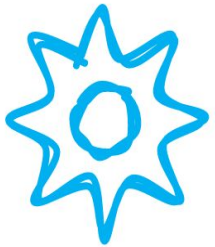
Views (Account Usage)

`DATABASE_STORAGE_USAGE_HISTORY`

`STAGE_STORAGE_USAGE_HISTORY`

Compute costs

- given by credits consumption
- warehouse size (number of nodes) corresponds to Snowflake credits for full hour of run time
  - XS Warehouse: 1 credit for 1 hour
- charged by second after first minute
- when suspended = no charge (auto suspend!)
- Usually 95% of all costs



	XS	S	M	L	XL	2XL	3XL	4XL	5XL*	6XL*
credits/hour	1	2	4	8	16	32	64	128	256	512

Where to find Warehouse Credit Usage

Snowsight

Admin -> Usage
ACCOUNTADMIN role is needed

SQL

Table functions (Information Schema)
`WAREHOUSE_MEERING_HISTORY`

Views (Account Usage)
`WAREHOUSE_MEERING_HISTORY`

Cloud service layer costs

- authentication, metadata management, SQL API, Access Control, Query parsing and optimization
- Snowflake credits are also used for paying the usage of cloud service layer
- Free up to 10% of the daily usage of compute resources

Date	Credits Used	Cloud Services Credits Used	Credit Adjustment for Cloud Services	Credits Billed totally
Feb 21	100	20	-10	110
Feb 22	120	10	-10	120

How to find out cloud service layer usage source

- `ACCOUNT_USAGE` schema in `SNOWFLAKE` database
- different perspective
  - query point of view
  - warehouse point of view
- typical use cases
  - find queries which consume the most cloud services credits
  - informative character

Keep in mind

- some particular features can have significant impact on cost. Needs to be used wisely
    - materialized views
    - auto clustering
    - replication
- always keep cost in mind - it is Pay as you go service
- cost optimization might be important part in later phases of the projects
    - use warehouse parameters

Knowledge check

1. How is compute usage billed in Snowflake?
A) A fixed monthly fee, regardless of usage
B) Billed based on actual compute usage
C) Compute resources are free of charge
D) Billed based on the amount of data stored

2. True or false: Cloud service layer services are always available for free
   A. True
   B. False

3. Select all Snowflake cost driver
   A. Number of Snowflake users
   B. Cloud provider
   C. Used region
   D. Number of virtual warehouses
   E. Number of databases
   F. Snowflake edition

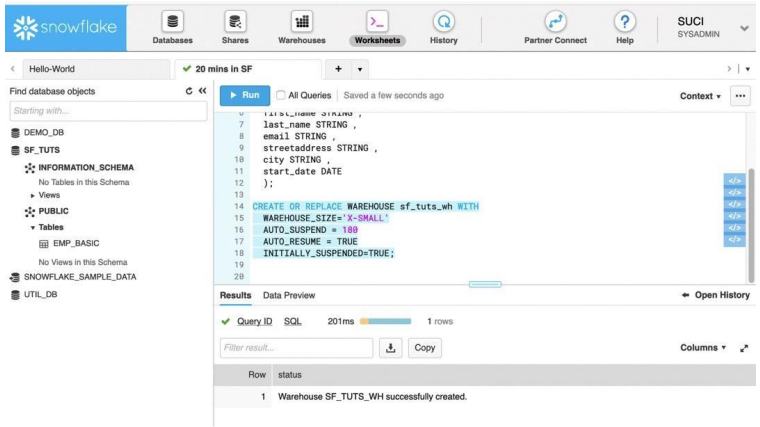
Snowflake tools overview

Snowflake UI tools

Classic Console

- original Snowflake UI interface
- no new development but still supported
- still preferred by some users
- no auto complete for a code
- Not available to new accounts

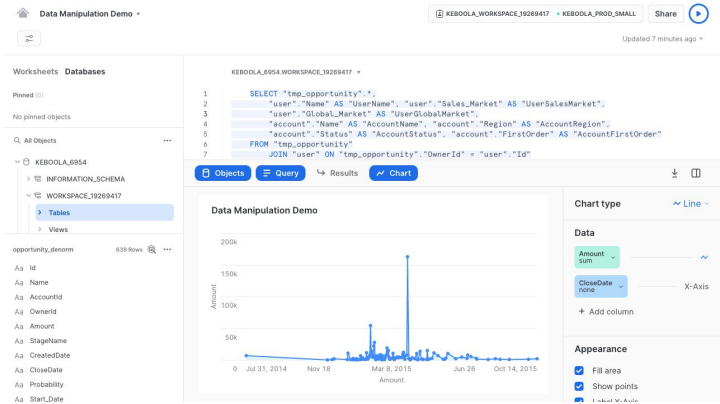
  (starting May 30 2023)



Snowflake UI tools

Snowsight

- new Snowflake web interface
- actively developed and improved over time
- modern look and feel
- code auto complete & object names suggestion
- integrated interface for reaching the support
- integrated collaborative features
- integrated query visualization



SnowSQL

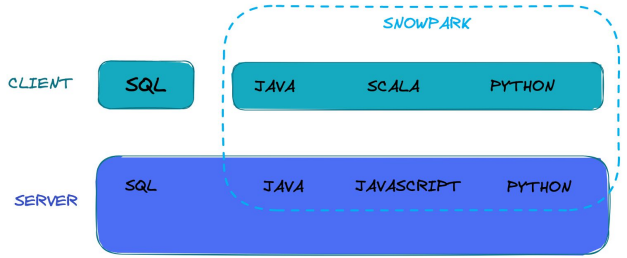
- Command line interface (CLI) to interact with your Snowflake account from your terminal
- Almost everything that can be done via UI can be done in CLI
    - Execute SQL queries
    - Manage snowflake objects
    - Loading data
- There are operations which could be done only in CLI (managing files in internal stages)
- Support for all major OS (Linux, Mac, Windows)

SnowCLI – Snowflake Developer CLI

- Open source tool originally, official product
- It allows you locally run and debug Snowflake apps
- Automation of many workflows (apps, Snowpark code, CI/CD pipelines)
- Manage containers, snowpark code, streamlit and native apps, git, etc.
- Define packages using `requirements.txt`, with dependencies automatically added via integration with Anaconda at deploy time.
- Deployment artifacts are automatically managed and uploaded to Snowflake stages
- SnowCLI vs SnowSQL ?

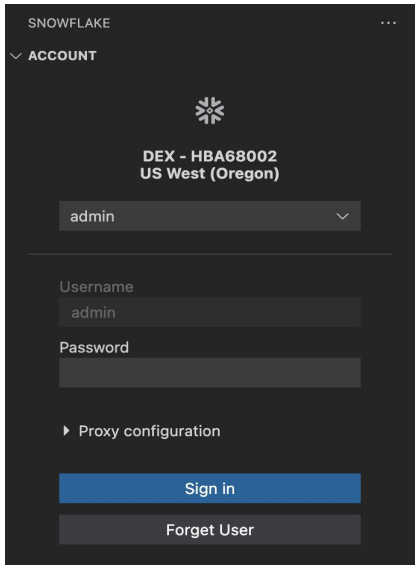
Snowpark

- New development framework for Snowflake
- Developers can code in their familiar way with language of their choice
- Create ML model, data pipeline, data apps in single platform
  - Faster
  - More secure
- Data frame style programming
- Code runs in Snowflake using the elastic platform
- Supports Scala, Java & Python



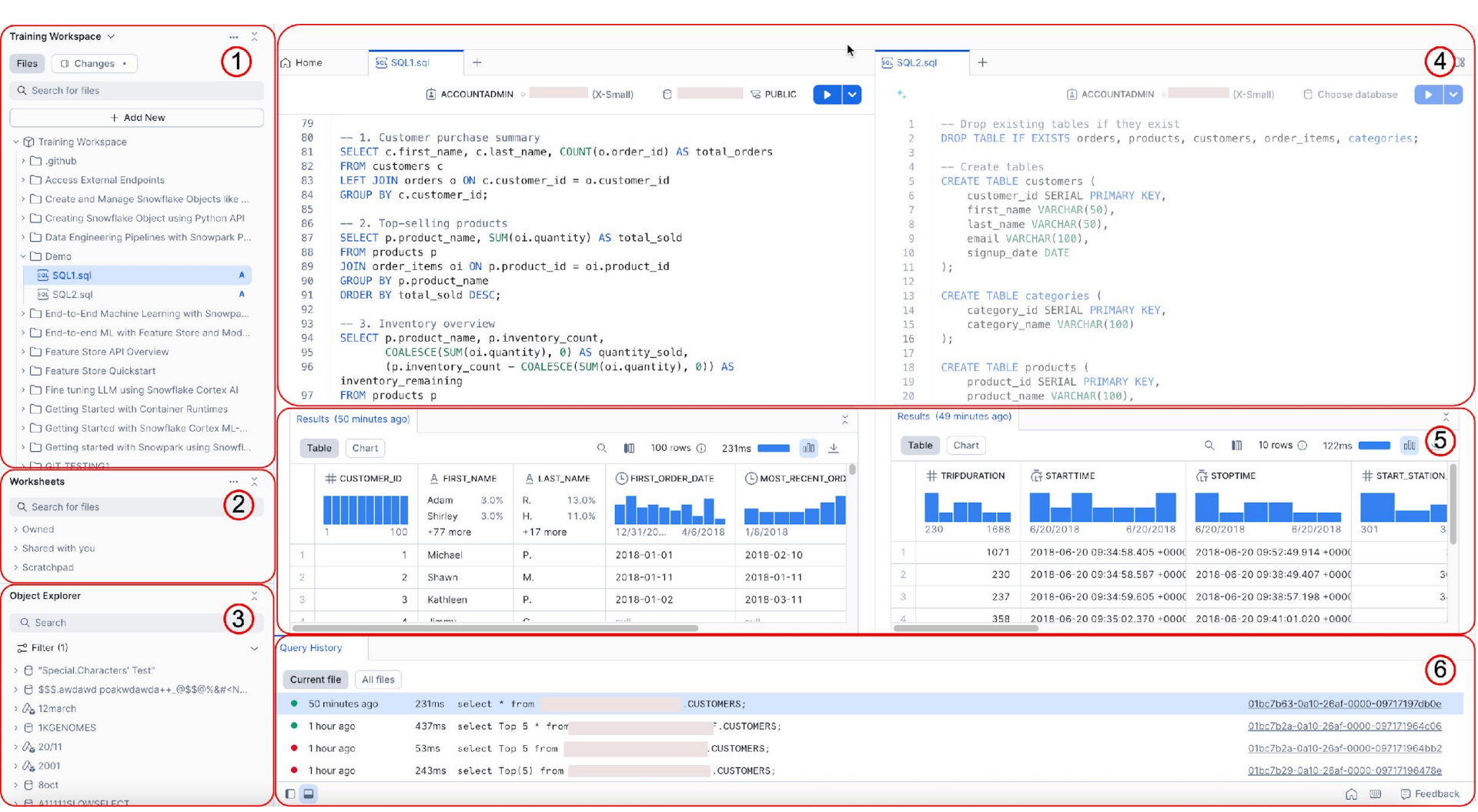
VS Code extension

- Official Snowflake offering
- Intellisense support
- DB explorer
- Query results and history
- Connect to multiple accounts and easily switch
- Easily integrable with other VS Code extensions (e.g. GIT)



Workspaces

- New cloud based IDE
- Built-into Snowsight
- Run DBT projects
- Managed your code
- Integrate with GIT
  - Store your worksheets in GIT!
  - Integrate whole workspace with GIT and sync it
- Project -> workspaces
- It can run two queries simultaneously from the same SQL file.


Training Workspace
1
Files
Changes
Search for files
Add New
Training Workspace
.github
Access External Endpoints
Create and Manage Snowflake Objects like ...
Creating Snowflake Object using Python API
Data Engineering Pipelines with Snowpark P...
Demo
SQL1.sql
SQL2.sql
End-to-End Machine Learning with Snowpa...
End-to-end ML with Feature Store and Mod...
Feature Store API Overview
Feature Store Quickstart
Fine tuning LLM using Snowflake Cortex AI
Getting Started with Container Runtimes
Getting Started with Snowflake Cortex ML-...
Getting started with Snowpark using Snowfl...
Worksheets
2
Search for files
Owned
Shared with you
Scratchpad
Object Explorer
3
Search
Filter (1)
"Special.Characters' Test"
$$S.awdawd poakwdawda++_@$$@%&#<N...
12march
1KGENOMES
20/11
2001
8oct
Home
SQL1.sql
ACCOUNTADMIN
(X-Small)
PUBLIC
79
80 -- 1. Customer purchase summary
81 SELECT c.first_name, c.last_name, COUNT(o.order_id) AS total_orders
82 FROM customers c
83 LEFT JOIN orders o ON c.customer_id = o.customer_id
84 GROUP BY c.customer_id;
85
86 -- 2. Top-selling products
87 SELECT p.product_name, SUM(oi.quantity) AS total_sold
88 FROM products p
89 JOIN order_items oi ON p.product_id = oi.product_id
90 GROUP BY p.product_name
91 ORDER BY total_sold DESC;
92
93 -- 3. Inventory overview
94 SELECT p.product_name, p.inventory_count,
95 COALESCE(SUM(oi.quantity), 0) AS quantity_sold,
96 (p.inventory_count - COALESCE(SUM(oi.quantity), 0)) AS
inventory_remaining
97 FROM products p
SQL2.sql
4
ACCOUNTADMIN
(X-Small)
Choose database
1 -- Drop existing tables if they exist
2 DROP TABLE IF EXISTS orders, products, customers, order_items, categories;
3
4 -- Create tables
5 CREATE TABLE customers (
6 customer_id SERIAL PRIMARY KEY,
7 first_name VARCHAR(50),
8 last_name VARCHAR(50),
9 email VARCHAR(100),
10 signup_date DATE
11);
12
13 CREATE TABLE categories (
14 category_id SERIAL PRIMARY KEY,
15 category_name VARCHAR(100)
16);
17
18 CREATE TABLE products (
19 product_id SERIAL PRIMARY KEY,
20 product_name VARCHAR(100),
Results (50 minutes ago)
Table
Chart
100 rows
231ms
CUSTOMER_ID
FIRST_NAME
Adam 3.0%
Shirley 3.0%
+77 more
LAST_NAME
R. 13.0%
H. 11.0%
+17 more
FIRST_ORDER_DATE
12/31/20... 4/6/2018
MOST_RECENT_ORD
1/8/2018
1 1 Michael P. 2018-01-01 2018-02-10
2 2 Shawn M. 2018-01-11 2018-01-11
3 3 Kathleen P. 2018-01-02 2018-03-11
Results (49 minutes ago)
5
Table
Chart
10 rows
122ms
TRIPDURATION
230 1688
STARTTIME
6/20/2018 6/20/2018
STOPTIME
6/20/2018 8/20/2018
START_STATION_
301
1 1071 2018-06-20 09:34:58.405 +0000 2018-06-20 09:52:49.914 +0000
2 230 2018-06-20 09:34:58.567 +0000 2018-06-20 09:38:49.407 +0000
3 237 2018-06-20 09:34:59.605 +0000 2018-06-20 09:38:57.198 +0000
4 358 2018-06-20 09:35:02.370 +0000 2018-06-20 09:41:01.020 +0000
Query History
6
Current file
All files
50 minutes ago 231ms select * from .CUSTOMERS; 01bc7b63-0a10-26af-0000-09717197db0e
1 hour ago 437ms select Top 5 * from .CUSTOMERS; 01bc7b2a-0a10-26af-0000-097171964c06
1 hour ago 53ms select Top 5 from .CUSTOMERS; 01bc7b2a-0a10-26af-0000-097171964bb2
1 hour ago 243ms select Top(5) from .CUSTOMERS; 01bc7b29-0a10-26af-0000-09717196478e
Feedback

Snowsight Demo

Exercise – DB objects creation

1. Use role `SYSADMIN`
2. Create a database called `CITIBIKE` with **SQL command**
3. Create schema called `WORK`, this time use **Snowsight UI**
4. Create table `TRIPS` in `WORK` schema. Please use **SQL** or **UI** (do not forget to setup the context for your worksheet)

Here you can find columns and data types for the `TRIPS` table:

```
 trip_id number,
starttime timestamp_ntz,
stoptime timestamp_ntz,
duration number,
start_station_id string,
end_station_id string,
trip_order string,
bike_type string,
bike_id string,
user_name string,
user_birth_date string,
gender string,
ride_type string,
membership_type string,
verification_method string
```

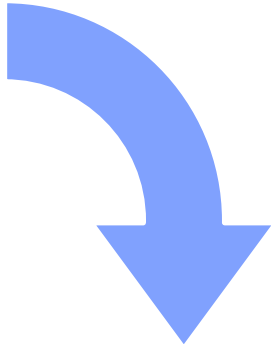
Micropartitions

Micropartitions

Storage unit holding the data

- Physical files holding the logical tables data
- Many micro partitions per table
- Max 16 MB compressed data …50 - 500 MB of uncompressed data
- Immutable
  - Updates create a new micropartition
- Snowflake keeps metadata about micro partitions in service layer
  - Helps with query optimization and performance
- Tables divided horizontally in smaller units

ID	NAME
1	Alice
2	Bob
3	Chuck
4	Jane
5	John
6	Mary



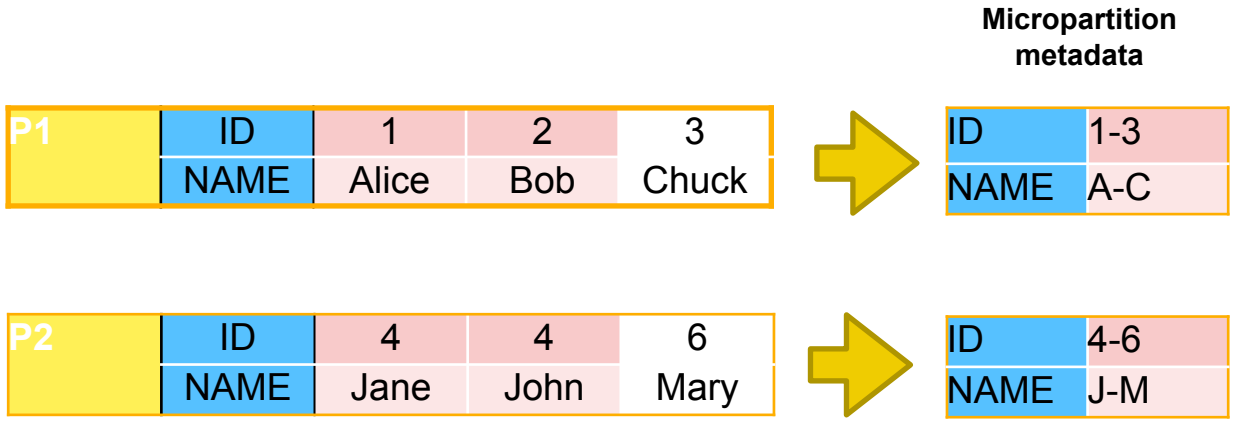
P1	ID	1	2	3
	NAME	Alice	Bob	Chuck

P2	ID	4	5	6
	NAME	Jane	John	Mary

Micropartitions metadata

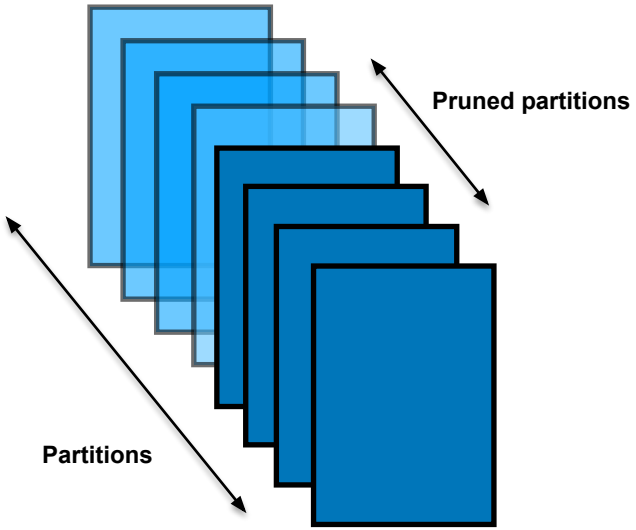
Automatically collected and maintained

- Table level
  - Row count
  - Table size in bytes
  - Referenced files
- Micropartition column level
  - Range of value
  - Number of distinct values
  - Min and MAX values
  - NULL count



Micropartition advantages

- Metadata used by optimizer for partition pruning
  - Horizontal pruning first - by partition
  - Vertical pruning next - by column
- Unit of management for DML - because of immutability
- Thanks to stored metadata some queries are pure metadata operations
  - No need for running virtual warehouse!
- Data Clustering
  - Effective data loading have impact on query performance



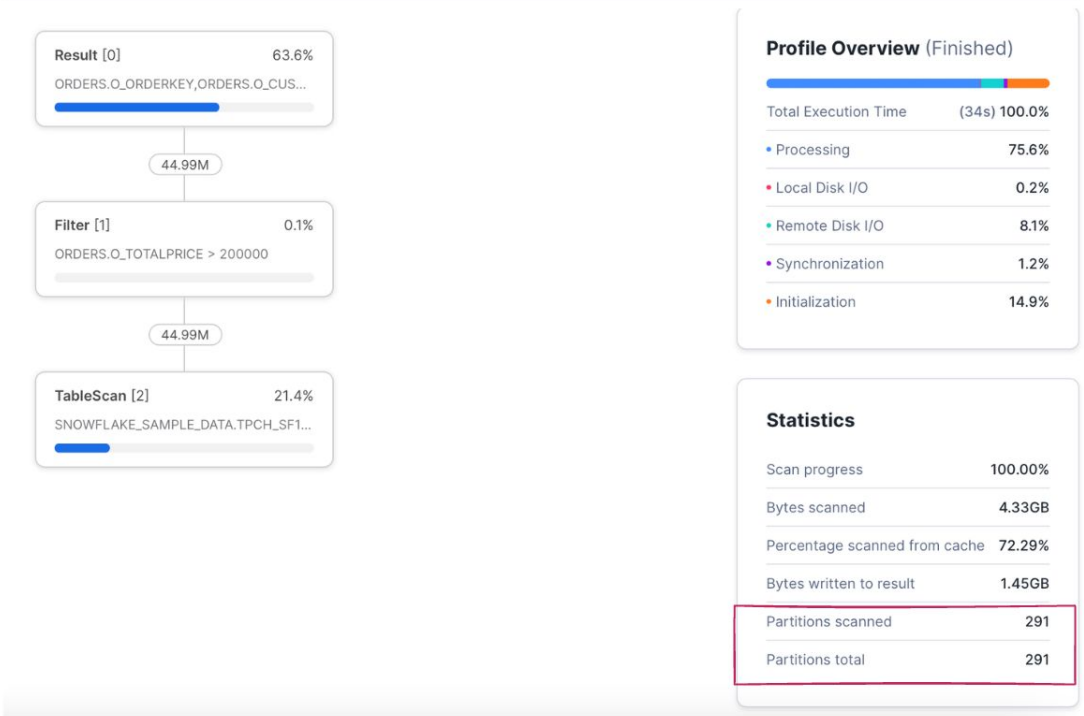
Pruning

- Elimination of not needed micro partitions
- Reading of MP = very costly operation (reading data over network)
- Full table scans are expensive for large tables (many micro partitions)
- Try to avoid scanning a full tables:
  - Limit number of columns - scan only accessed ones
  - Using column metadata to avoid scanning data that is filtered out

- Pruning depends on
    - Applied filters in the query
    - Clustering of data (natural, clustering keys)

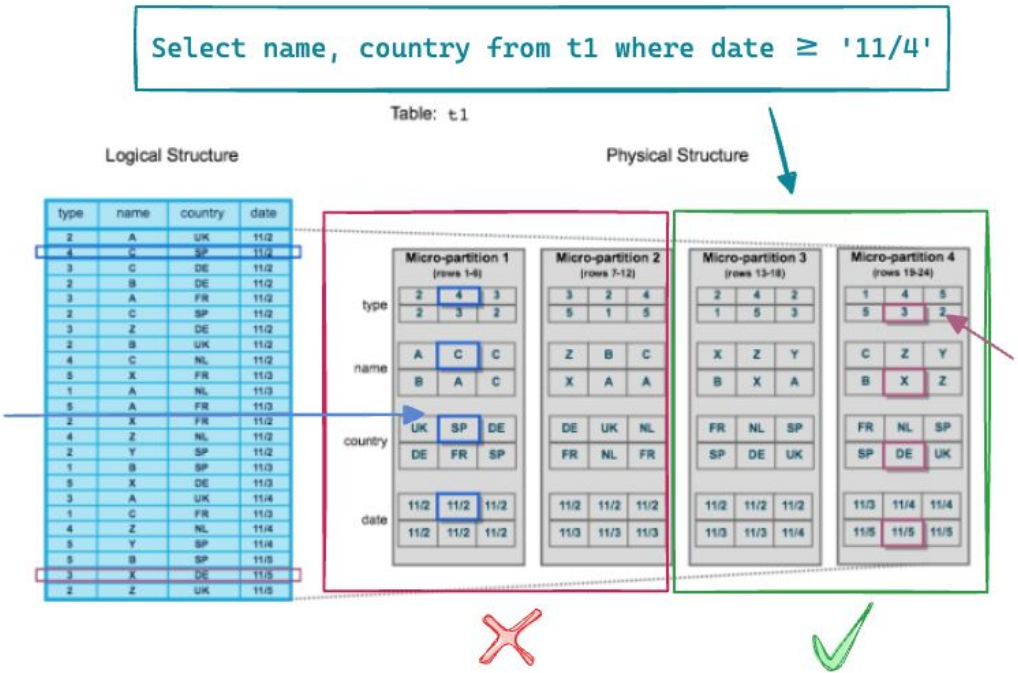
Identifying query pruning performance

```
select * from orders where o_totalprice > 200000;
```



- Bad pruning
- Full table scan
- Table is not clustered on that column
- Add your data knowledge

Data stored by column – hybrid columnar



Data stored by column

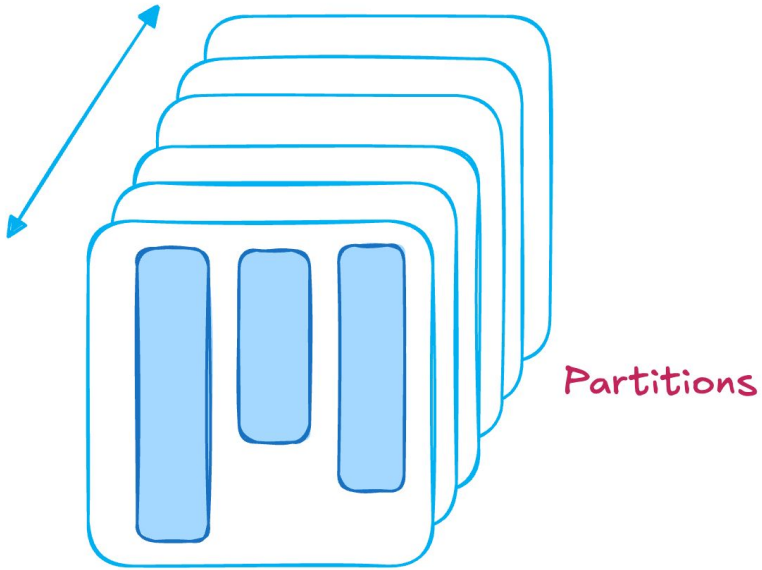
- values of each column grouped together and compressed

Hybrid columnar

- Grouping by rows and columns.
- Each micropartition always contains the whole row

Columnar compression

- Ingestion automatically analyzes and compress data when loading it into a table
- Trying to find the best compression for each data type
- Efficient compression is achieved thanks to storing a same data type
- Columns can grow independently
- Impact on performance thanks to reduce I/O operations and storage



Knowledge check

1. When can be micro partitions updated?

a) Only if there is less than 16 MB of data
b) Never
c) Whenever it is needed
d) When it’s enabled by admin

2. What metadata snowflake keeps about columns in micropartitions. Select them all.
A) Creation date
b) Range of values
c) Number of distinct values
d) Virtual warehouse name
e) AVG value
e) Null Count

Authentication methods

Authentication methods

User & Password

Key Pair

Single Sign On

Multi Factor Authentication

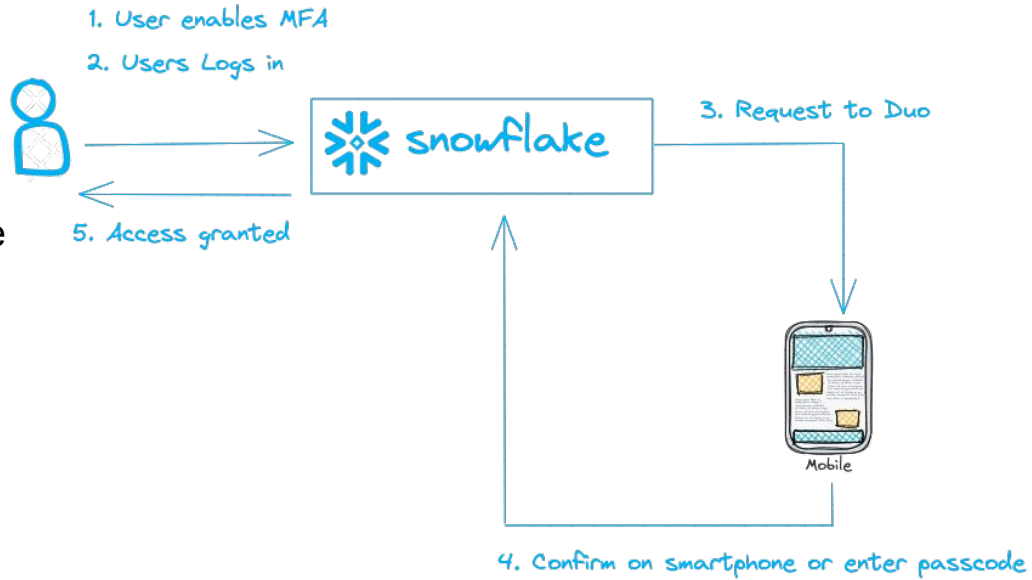
** API authentication based on tokens or secrets*

User & Password

- Basic authentication method
- Password policies to define rules
  - Length and type of characters
  - Rotation period
- Behavioral change
  - Will be blocked by November 2025 – need to use MFA

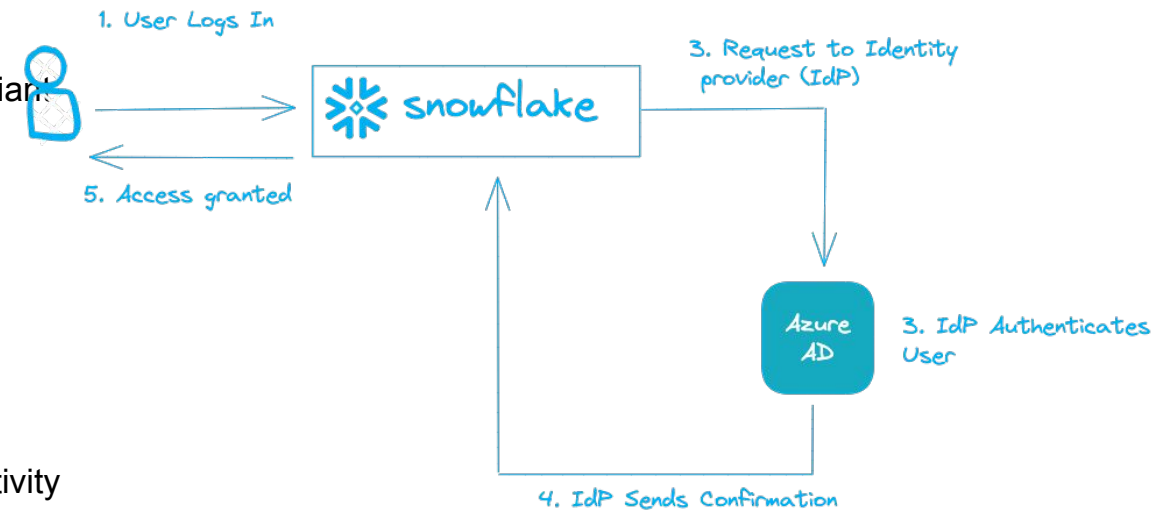
Multi factor authentication

- One extra step after username/password
- Multiple MFA methods
  - Duo service, Microsoft, Google authenticator
  - passkey
- Users must enroll themselves (User Profile)
  - Unenroll needs to be done by account admin
    - ALTER user parameters
      - MINS_TO_BYPASS_MFA
      - DISABLE_MFA



Federated Authentication (SSO)

- Native support
  - OKTA
  - Microsoft ADFS
- Support for most SAML 2.0 compliant vendors as IdP
  - MS Azure Active Directory
  - Google G Suite
  - OneLogin
- Supported workflows
  - Login
  - Logout
  - System timeout due to inactivity
- Disable local login = set empty password



Authentication policies

- Provides control how user authenticates by specifying different rules:
  - Enforce MFA
  - Which authentication methods requires MFA
  - Allowed different authentication methods
- Configure on account or user level

Type property for users

- Improves security for system users by enforcing different rules
- Define type of the user
  - `PERSON, SERVICE, LEGACY_SERVICE`
- `SERVICE` user type
  - Cannot use user & password
  - Cannot log in using SAML SSO
  - Cannot enroll in MFA
  - Cannot have various properties like name, password, etc.

Network Policies

- Account or User level
- Created by security admin or higher
- IP allowlisting or blocklisting
- User denied access if they attempt to log in from restricted IP address
- Single IP or IP range
  - 192.168.1.0/24
    - All IP address in the range of 192.168.1.0 to 192.168.1.255
- 1. CREATE network policy (UI or SQL)
- 2. Assign it to the user
  - `ALTER USER <name> SET NETWORK_POLICY = <my_policy>`

Exercise

- Let’s practice configuration of different authentication methods
  - We will generate key-pair and assign the public key to our user
  - We enroll MFA
  - We define authentication policy to define MFA methods

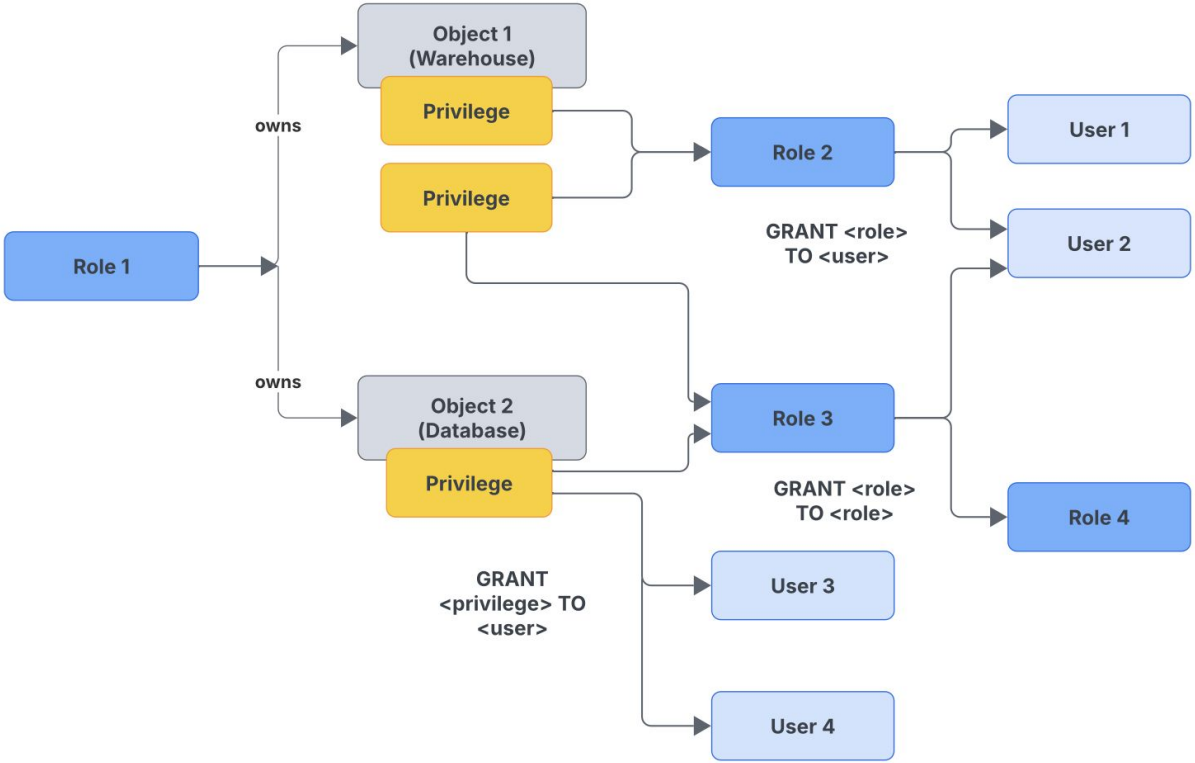
Access control framework

Authorization

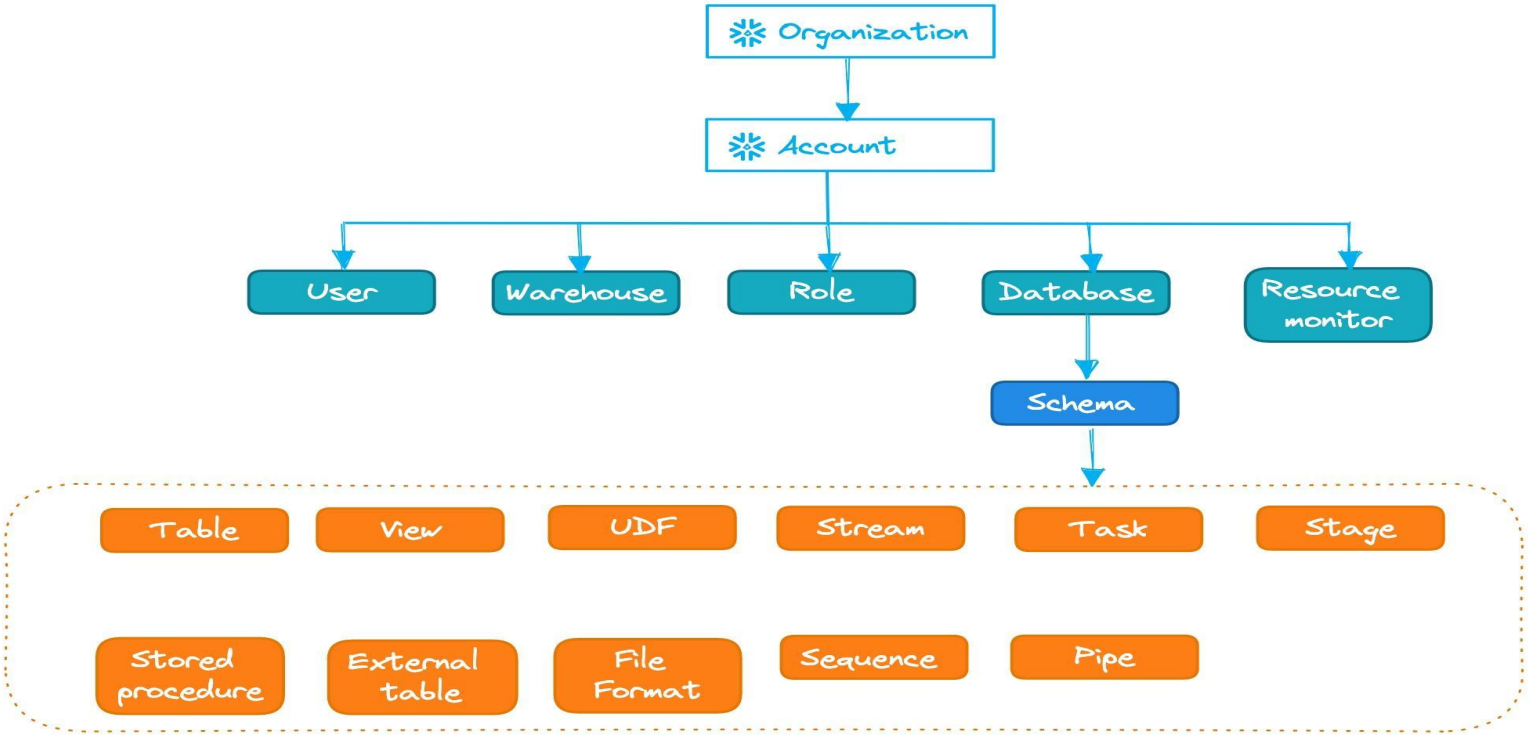
- Independent of authentication
- Who can access what objects
- Combines following modes
  - Role-based Access Control (RBAC)
  - Discretionary Access Control (DAC)
  - User-based Access Control (UBAC)

- Key terms of the concept
  - Securable object
  - Role
  - Privilege
  - User

Authorization schema



Securable objects



- Each object has a single owner (role)

Roles

- Entity to which privileges on objects are granted
- Assigned to users
  - User can have multiple roles
  - Active roles and secondary roles
- Role hierarchies
- Custom & system defined roles
- System roles can’t be dropped
- Account vs database roles

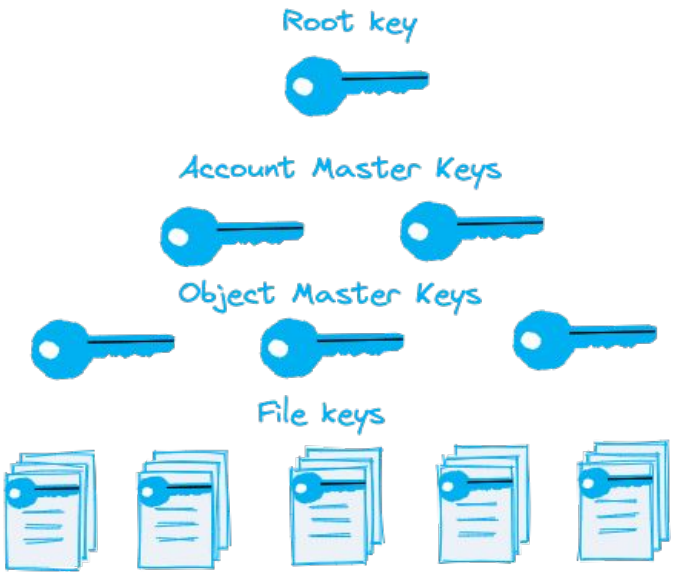
Privileges

- What action can be performed on the object
- Assigned to roles
- Managed by `GRANT` and `REVOKE` commands
- Object owner can perform GRANT operations
  - Or role with `MANAGE GRANTS` global privilege
- Privileges are inherited between roles

```
GRANT SELECT ON TABLE
my_table
```

Data protection

- All data are always stored across 3 Availability zones (disaster recovery)
- All data are always stored encrypted
- Keys are managed automatically
- Snowflake uses AES256 bit encryption
- Separated key tree per customer
- Hierarchical key model
  - The root key
  - Account Master Keys
  - Table Master Keys
  - File keys



Key Life cycle

- Rotation is done automatically every 30 days
- Rekeying has no customer impact
- Option to annually re-key data (Enterprise feature)
- Option to have customer-managed key together with Snowflake-maintained key: Tri-Secret Secure
- (Business Critical feature)

Knowledge check

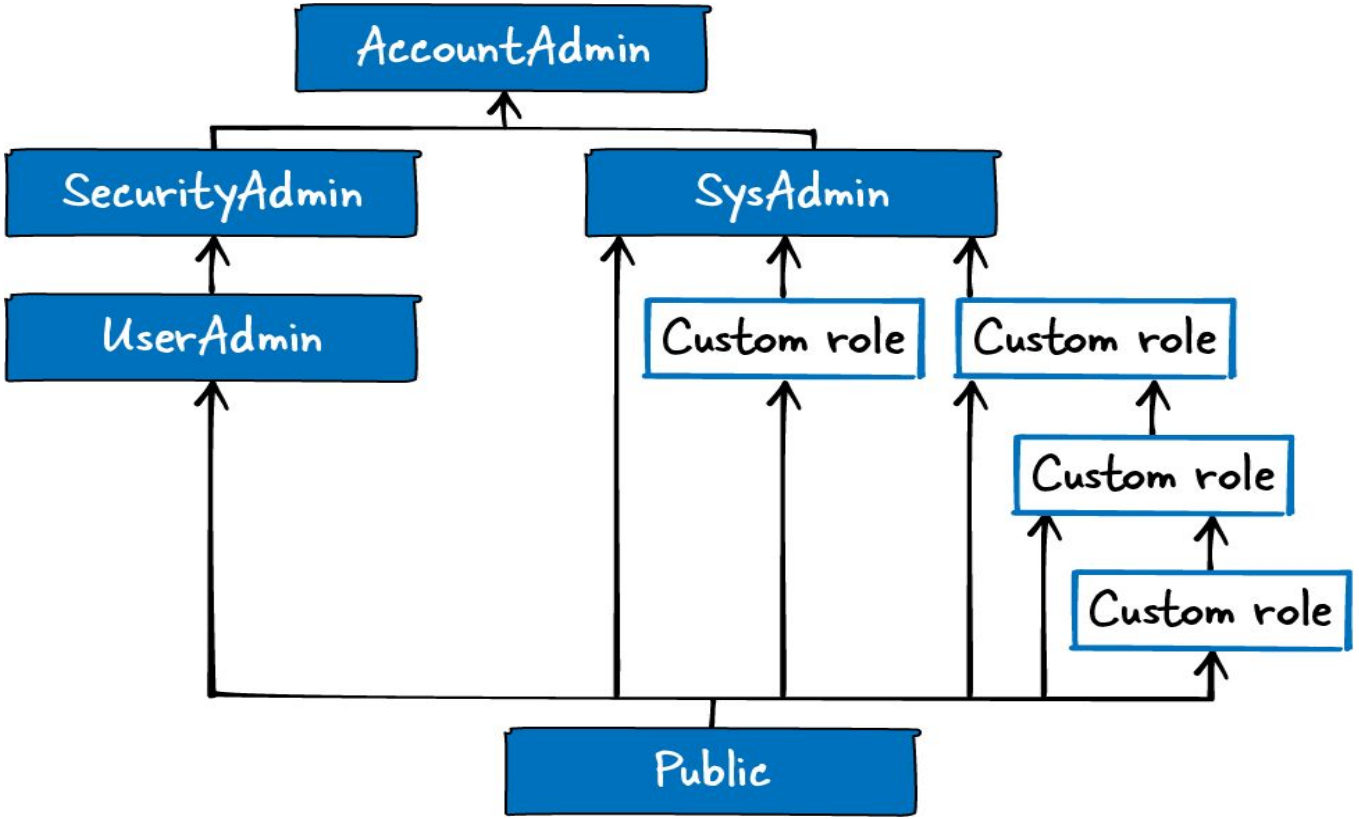
1. What is not the system role in Snowflake
A] SECURITY ADMIN
B] SYS ADMIN
C] WAREHOUSE ADMIN
D] USER ADMIN

3. How often Snowflake rotate encryption keys?
a] every 7 days
b] every 14 days
c] every 30 days
d] every 365 days

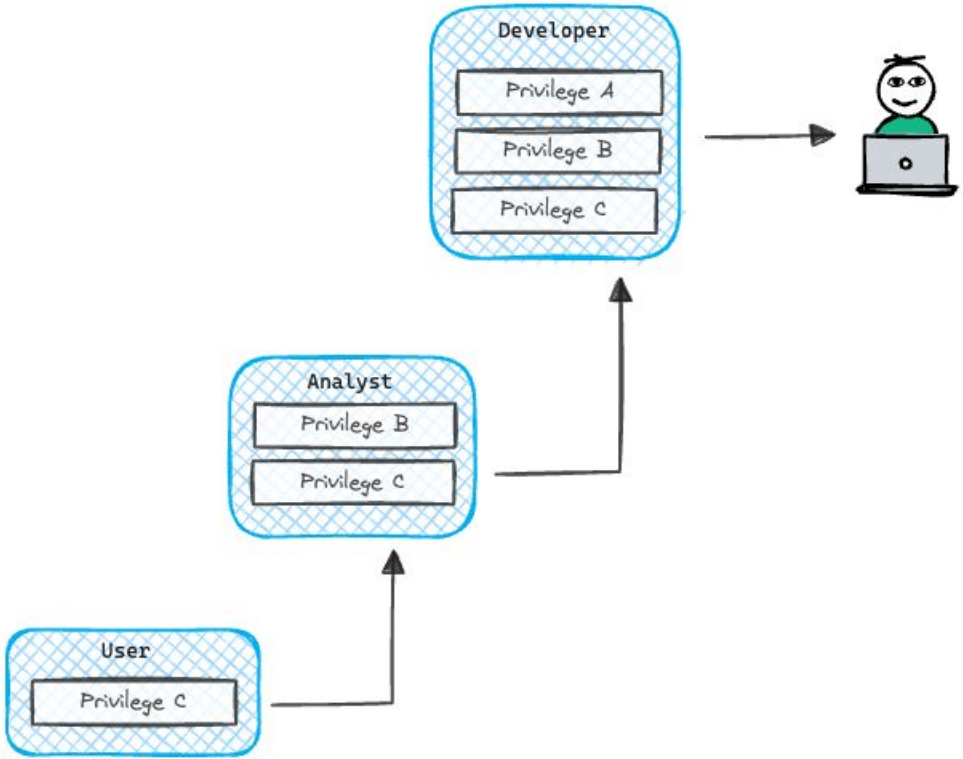
2. Which role can’t be activated in the session?
a] database role
b] account role
c] secondary role
d] primary role

System roles and hierarchies

System defined roles



Roles hierarchy



Database roles

- Account roles = perform actions on any object in your account
- Database roles – limit actions to single database
- Can’t be activated in the session
  - Grant them to account role but not other way around

- Use cases
  - Management simplification
  - Data sharing

DB roles - management simplification

- Pass the access management to database owners
  - No need to use SECURITYADMIN role (or `MANAGE GRANTS` privilege)
  - DB owners can manage DB roles and grant privileges

- Database roles exist within DB
  - Can only manage objects inside that DB
  - Any privilege can be granted
  - DB OWNERSHIP must be granted to account level role

- Extend role hierarchies with DB roles
  - Grant the highest DB role to account role

Managed access schemas

- Privilege management is done by schema owners not object owner
- Schema owners and roles with `MANAGE GRANTS` can grant privileges
- Privilege management centralization

Authorization Best practices

- Limit the usage & assignment of the `ACCOUTADMIN` role
- In case you need ACCOUNTADMIN features grant them to other roles! (e. g. `IMPORTED PRIVILEGES`, `EXECUTE TASK`, etc.)
- Use MFA for human users
- Always grant custom roles to `SYSADMIN` to keep building role hierarchy
- Align object access with business functions
- Simplify grant management with future grants

.

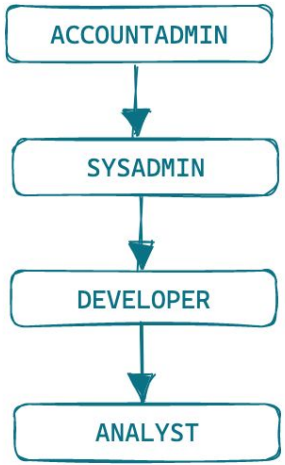
Authorization Best practices

- Limit number of users managing the grants
- Define future grants on schema level to simplify the object management
- Follow the least privilege principle
- Keep it under version control
- Create separate roles for managing features (tasks, shares, etc.)

.

Exercise

- Let's create couple of custom roles for our project. We will try to follow the best practises, meaning that:
  - `SECURITYADMIN` will be owner of those roles
  - `SYSADMIN` should have access to those roles
- Create a role called `ANALYST` with following privileges:
  - Read access to table `TRIPS` (SELECT privileges)
  - Ability to use warehouse `COMPUTE_WH`



- Create a role called `DEVELOPER` with following privileges:
  - Access to `ANALYST` Role
  - Full access to DB `CITIBIKE`
- Use role `ANALYST` and try to create a new table in public schema
- Use role `DEVELOPER` and try to create a new table in public schema
- **Note:** If you need to grant access to tables you have to grant USAGE privilege on SCHEMA and DB level.

Data Governance Capabilities

Accounts vs Organization

- Accounts can be grouped under organization
- Central view of all accounts in organization
- Self-service account creation & management
- Data replication and failover
- Separated metadata for organizations
  - `ORGANIZATION_USAGE` schema

Secure Objects

- Views, MVs, UDFs
- Protect underlying data from unintentional exposing
- Standard view DDL is visible to user in various commands or interfaces -> not for secure views
  - Because of security or privacy reasons you might not wish to expose it
- Strongly recommend use secure views for sharing
- Use it only in case of security concern, not like a replacement of ordinary object

Snowflake Metadata

Usage metrics

ACCOUNT_USAGE

- Part of Snowflake database
- Views
- Up to year of data retention
- Include drop objects

Snowflake Metadata

Usage metrics

INFORMATION_SCHEMA

- Part of each database
- Table functions
- Data retention between 7 days to 6 months
- No dropped objects

ACCESS_HISTORY

Key view from ACCOUNT_USAGE schema for data governance

- Get an overview about reads and writes in your Snowflake account
- Satisfy regulatory compliance
- Understand your usage patterns and optimise storage cost and performance
  - Discover unused tables/columns
  - Discover the most frequently used tables for performance optimisations
  - Check out the speed of adoption for new objects

ACCESS_HISTORY – Read Operations

Satisfy regulatory compliance requirements

- Complete overview about accessed tables, views and columns from each query
- view base tables included (indirectly accessed)
- data stored as JSON documents in ACCESS_HISTORY view
  - data could be easily queried for auditing or log purposes

Storage cost and performance optimization

- Discover unused tables and columns

ACCESS_HISTORY – Write Example

Loading data from external stage

```
copy into table1(col1, col2)
from (select t.$1, t.$2 from @mystage1/data1.csv.gz);
```

The DIRECT_OBJECTS_ACCESSED and BASE_OBJECTS_ACCESSED column says that external named stage was accessed

The OBJECTS_MODIFIED column says that data was written to two columns of the table

ACCESS_HISTORY – write example

Loading data from external stage

```
copy into table1(col1, col2)
from (select t.$1, t.$2 from @mystage1/data1.csv.gz);
```


```
{
  "objectDomain": STAGE
  "objectName": "mystage1",
  "objectId": 1,
  "stageKind": "External Named"
}
```

```
{
  "columns": [
     {
       "columnName": "col1",
       "columnId": 1
     },
     {
       "columnName": "col2",
       "columnId": 2
     }
  ],
  "objectId": 1,
  "objectName": "TEST_DB.TEST_SCHEMA.TABLE1",
  "objectDomain": TABLE
}
```


DIRECT_OBJECT_ACCESSED

BASE_OBJECT_ACCESSED

Knowledge check

1. Where you can find metadata about dropped objects in FINANCE database?
a] ACCOUNT_USAGE Schema in SNOWFLAKE database
b] metadata about dropped objects are not available
c] INFORMATION_SCHEMA in FINANCE database
d] ACCESS_HISTORY schema in FINANCE database

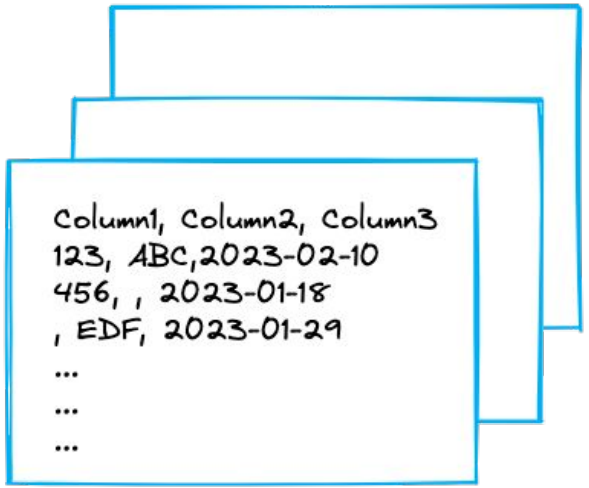
2. How is called system view which provides overview about read and write operations?
A] DIRECT_OBJECT_ACCESS
B] ACCESS_HISTORY
C] QUERY_HISTORY
D] ACCESS_OVERVIEW

Which metadata provides longer retention?
A] ACCOUNT_USAGE views
B] INFORMATION_SCHEMA views and table functions
C] CLOUD_SERVICE_METADATA

Data loading objects

FILE FORMAT

File structure description



CSV

- Record and field delimiters
- Escape characters
- NULL handling and other parameters

JSON

- Strip null values
- Date and time format
- Trim spaces and other parameters

Can be defined on different levels

- STAGE object
- COPY command
- Named object

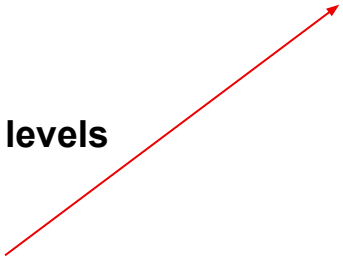
FILE FORMAT

File structure description

```
CREATE OR REPLACE FILE FORMAT
my_csv
   TYPE = csv
   FIELD_DELIMITER = ‘,’
   SKIP_HEADER = 1
   NULL_IF = ''
```

Can be defined on different levels

- STAGE object
- COPY command
- Named object



Use name objects!

- Reusability
- Segregation of duties
- Strong governance model
- Speed up the development process
- Easy to change it – just one place

STAGE

File location description

Stage types

- Table Stage:
  @%[Table_Name]
- User stage:
  @~
- Named stage: created manually with SQL - internal or external
- You can’t set file format for user and table stages

Internal or External

- Files could be loaded from local machine or existing landing zone in the cloud (S3, Blob storage, Cloud Storage)
- PUT command for loading local files into internal stage

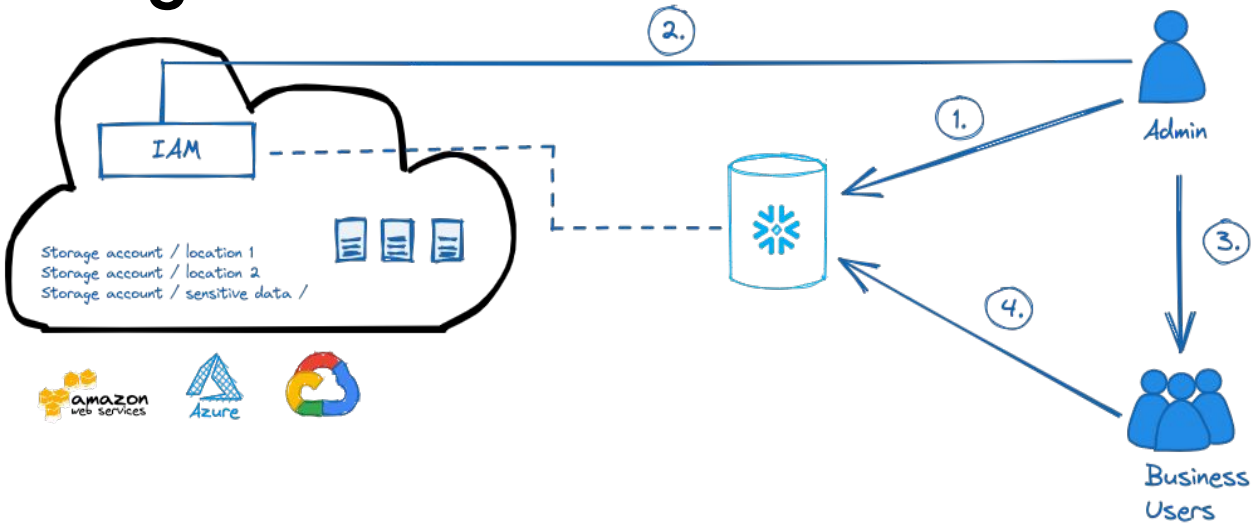
Stage security parameters

```
create stage my_ext_stage1
  url='s3://load/files/'
  credentials=(aws_key_id='1a2b3c' aws_secret_key='4x5y6z');
```

- Access keys are part of Stage definition
  - Disadvantages:
    - Security information needs to be shared with developers
    - Key rotation process
    - Needs to be defined for each stage
    - Security information is stored in Snowflake

Credential-less stages

Storage integration



1. Admin creates storage integration
2. Admin create trust policy between cloud provider and Snowflake
3. Admin grant storage integration usage to dev roles
4. Developers can create external stage with this storage integration

Storage integration benefits

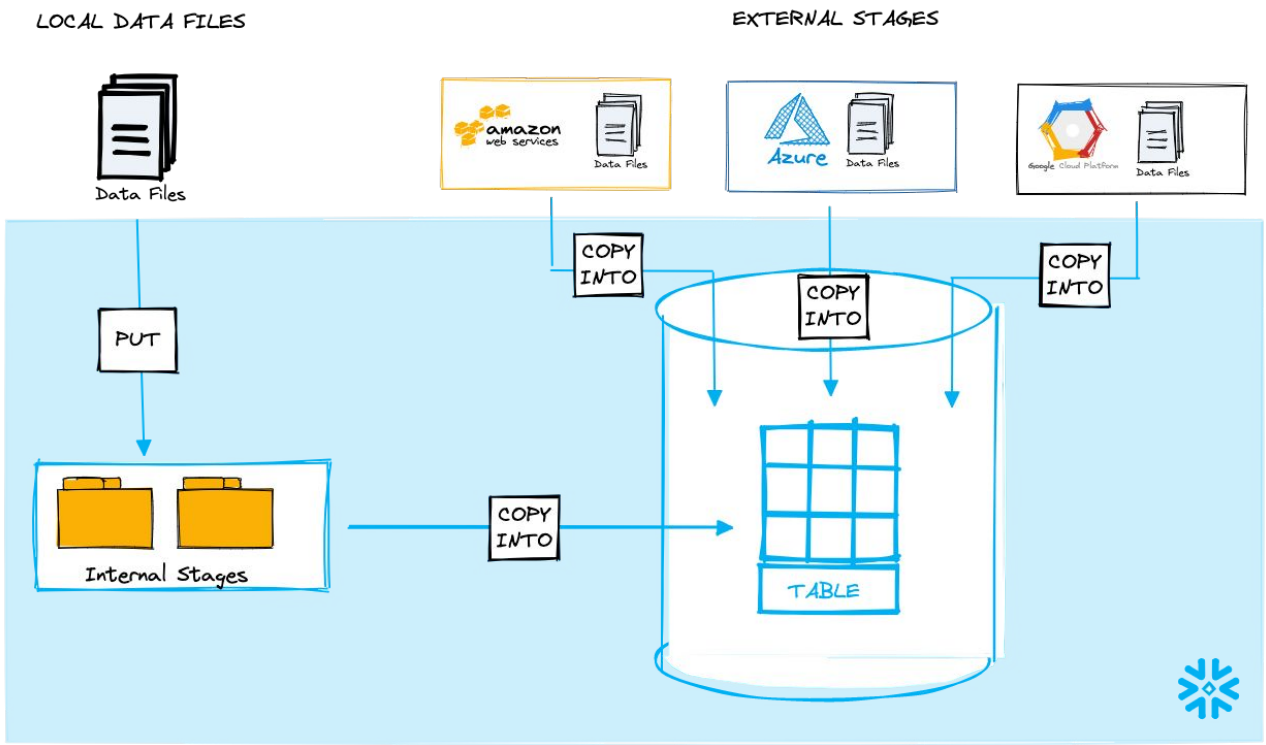
- Credentials are neither shared nor stored in Snowflake
- Reusability
- No need for recurring keys rotation
- Developers do not need to have the credentials in order to access files in external storages from Snowflake
- Segregation of duties

Storage integration creation

```
CREATE STORAGE INTEGRATION s3_int
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = 'S3'
ENABLED = TRUE
STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::001234567890:role/myrole'
STORAGE_ALLOWED_LOCATIONS = ('s3://mybucket1/mypath1/', 's3://mybucket2/mypath2/')
STORAGE_BLOCKED_LOCATIONS = ('s3://mybucket1/mypath1/sensitivedata/');
```

```
CREATE STAGE my_s3_stage
STORAGE_INTEGRATION = s3_int
URL = 's3://bucket1/path1/'
FILE_FORMAT = my_csv_format;
```

Loading data into Snowflake overview



COPY command – basics

- Migration from traditional data sources
- Batch processing
- Independent scaling of compute resources based on your needs and different workloads
- For both data loading and unloading
- Using Stage object as data source/target
- Need to define user managed warehouses for processing
- Basics transformations could be part of command
- A lot of copy options
- Support for plenty of formats

Basic syntax

```
COPY INTO
<table_name>
FROM
<internal_stage> |
    <external_stage>
FILE_FORMAT = ( … )
```

COPY output

Column name	Description
FILE	Name of source file and relative path to the file
STATUS	Loaded, failed …
ROWS_PARSED	# parsed rows from source file
ROWS_LOADED	# loaded rows from source file
ERROR_LIMIT	If the number of errors reaches this limit, then abort
FIRST_ERROR	First error in source file
FIRST_ERROR_LINE	Line number of the first error
FIRST_ERROR_CHARACTER	Position of the first error character
FISRT_ERROR_COLUMN_NAME	Column name of the first error

Transformations in COPY command

- Column ordering, omission and cast
- Done through the **SELECT** statement with limitations. Not supported features
  - `WHERE, ORDER BY, LIMIT, FETCH, TOP, JOIN, GROUP BY`

- What can be done
  - Using `CURRENT_TIMESTAMP()`
  - Using `SEQUENCE, AUTOINCREMENT` or IDENTITY columns
  - FLATTEN semi-structured data into individual columns

Transformations in COPY command

```
copy into home_sales(city, zip, sale_date, price)
from (select t.$1, t.$2, t.$6, t.$7 from @mystage/sales.csv.gz t)
FILE_FORMAT = (FORMAT_NAME = mycsvformat);
```

Exercise

Let’s create objects which we need for loading data into Snowflake – file format and stage to try to load some data into Snowflake with COPY command.

Best practices for data loading

Files preparation

Organize files by path

- Create logical paths per required dimension
  - Time
  - Location
  - Department

```
US/east_cost/finance/monthly/2022/0
```

2

File sizes

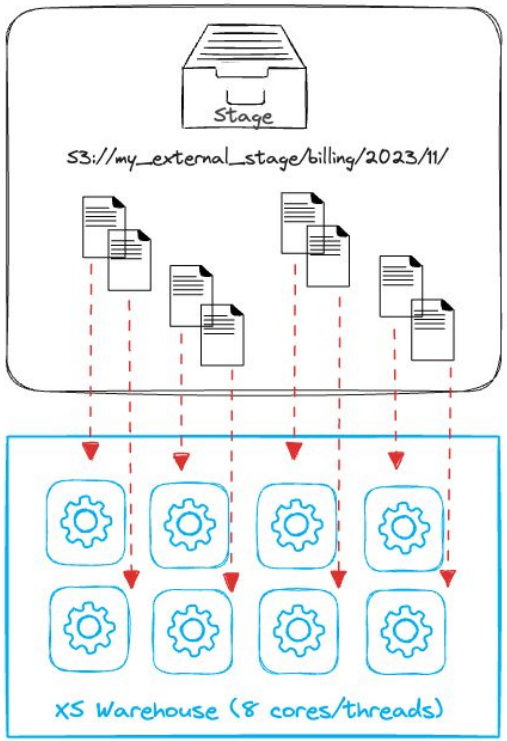
- Split large files before loading
- Batch / combine small files before loading
- Recommended size is 100 to 250 MB (compressed)

Parallel processing

Warehouse size	Number of Threads
XS	8
S	16
M	32
L	64
XL	128

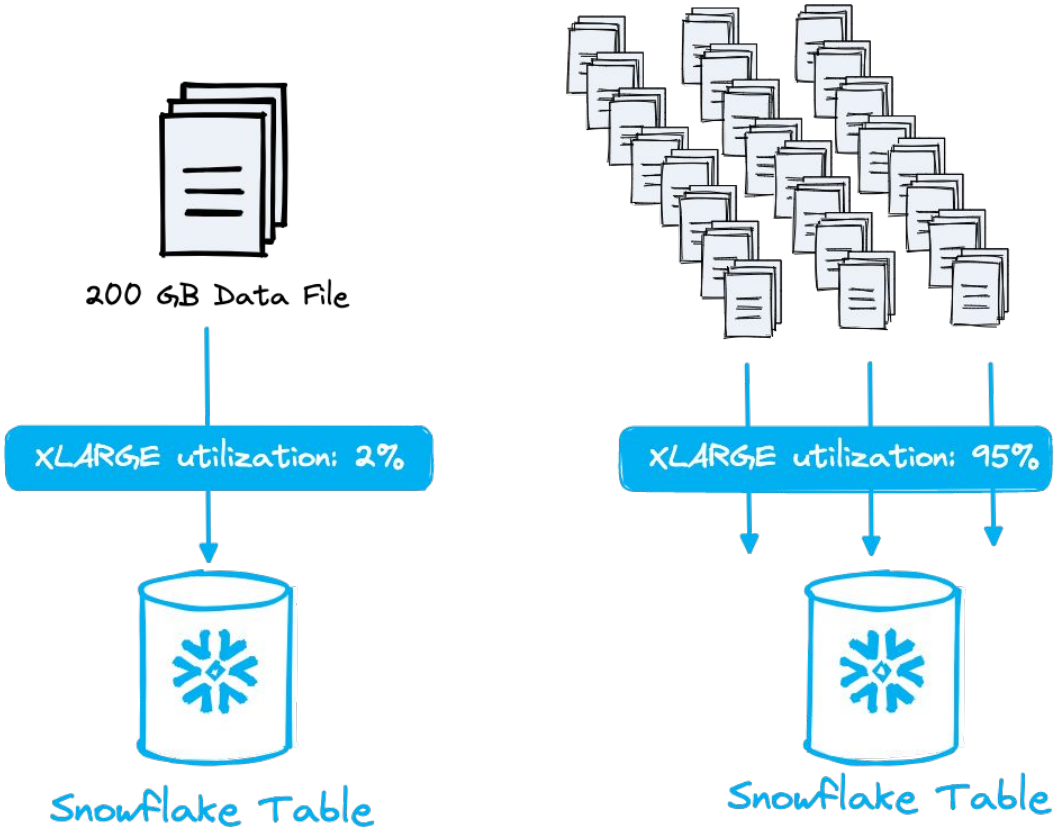
Number of files that can be loaded in parallel

Warehouse utilization



Warehouse size	Number of Cores/threads	Number of processed files in parallel
XS	8	8
S	16	16
M	32	32
L	64	64

Serial vs Parallel processing



File locations

All files in single location

Processing is slower as engine needs to scan more files to find the right ones

Organized files in multiple locations

Faster. Multiple directories with same file types allow for scanning fewer files

Takeaways

- You need to keep the most warehouse threads busy (warehouse utilization)
- Dedicated warehouse for loading operations
- Files needs to be properly sized and compressed
- Files needs to be organized
- Use the smallest warehouse possible
  - Little queuing is better than unutilized cores
  - Increasing the size until meet your SLA
  - Remember you are always billed for first 60 seconds

Ingesting 1 GB of data in 100 MB chunks (10 files):

Warehouse size	Number of cores	Utilization ratio	Run time	Billed time	Cost/day	Cost/year
Medium	32	~30%	10s	60s	$0.2	$73
Extra small	8	~100%	50s	60s	$0.05	$18.25

Knowledge check

1. How is called a Snowflake feature for credential-less authentication for external stages?
a] secure integration
b] secure keys
c] storage integration
d] credential integration

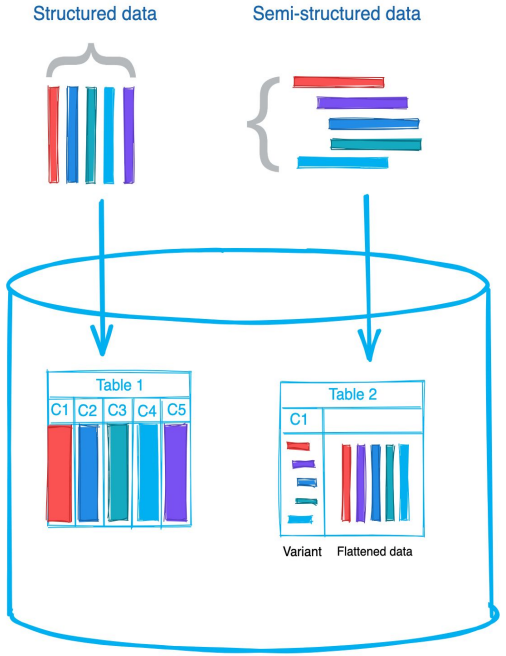
2. What is recommend file size (compressed) for data loading?
1.a] up to 16 MB
2.b] between 16 - 100 MB
3.c] between 100 - 250 MB [correct]
4.d] between 250 - 1000 MB

3. Mark all ways how you can define File Format
a] as part of user STAGE object definiton
b] as part of landing table definition
c] as named object
d] as part of COPY command
e] as part of table STAGE object definiton

Working with semi structured data

Overview

- Native support for JSON, AVRO, Parquet, XML, ORC
- Snowflake has a VARIANT data type for storing semi structured data
- Stores original document as is
- Traditional DBs requires complex transforms to get data into column structure
- SQL syntax is simple dot notation
- When ingesting SSD, data could be columnized and metadata collected
- Native support for all SQL operations



Semi structured data challenges

Only for imports/exports

Parquet, Avro, ORC

Popular formats for Data Lakes

Not readable by humans

Need to know the schema before ingestion

Table creation and COPY command automation could be automated nowadays

Semi structured data challenges

Import/export or storing

Easily readable by humans

Popular formats for APIs

No defined structure

Could contains many nested levels with objects, arrays or key – value pairs

Store it as is or flatten?

Parquet ingestion challenges

- You need to define schema when you read from Parquet file
  - You need to create a landing table with file schema
  - You need to use the schema in COPY command
  - You need to Cast columns in COPY command - defining target data type
- All parquet data are stored in Single column
- We use SELECT statement as part of COPY command

A lot of manual effort is needed

Syntax example:

```
COPY INTO <table_name>
    FROM ( SELECT
        $1:column1::<target_data_type>
        $1:column2::<target_data_type>
        $1:column3::<target_data_type>
        FROM
<my_stage>.<my_file.parquet>
);
```

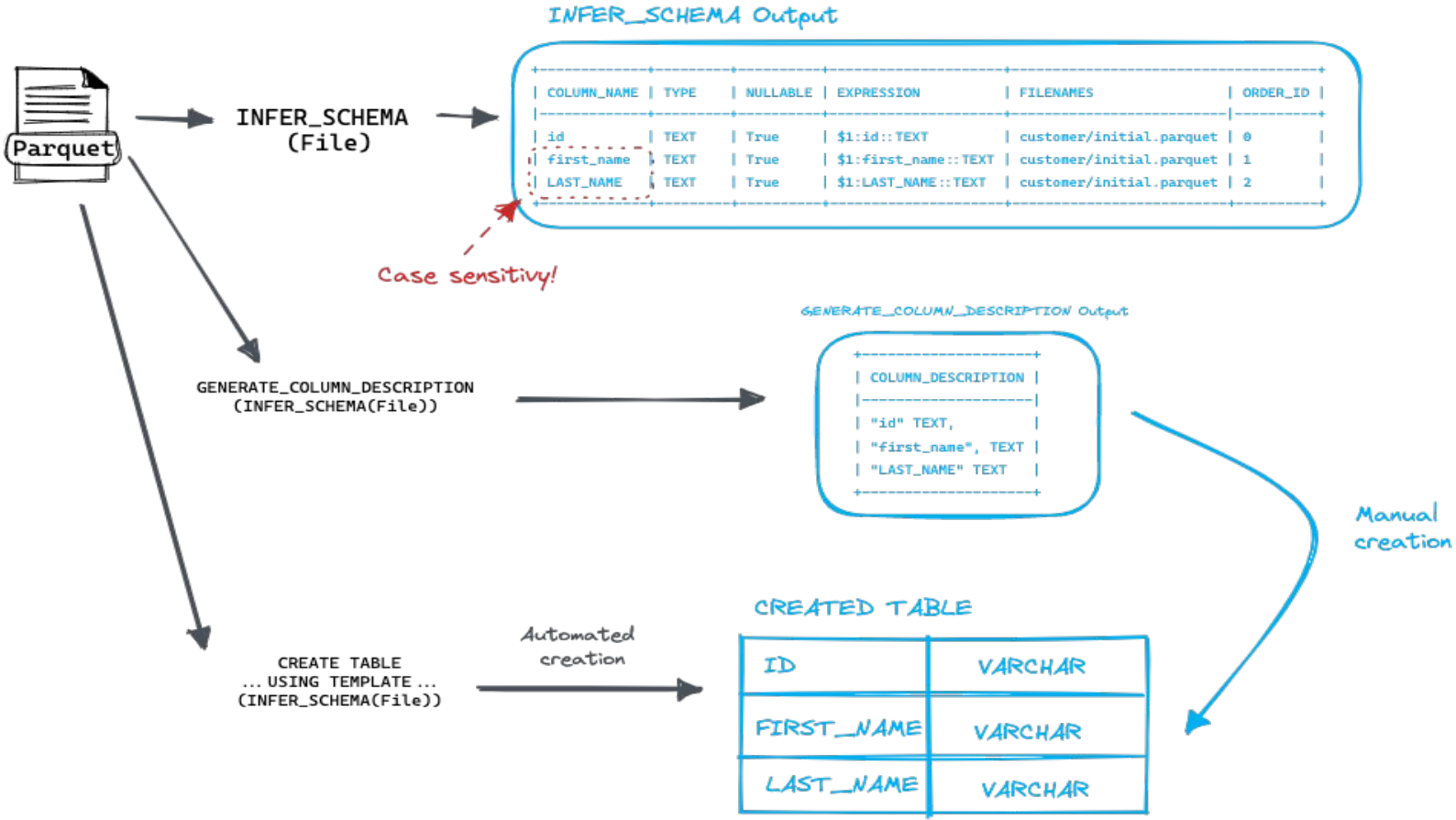
Binary format ingestion automation

- Built-in functions help you with schema detection, COPY command automation and landing table creation

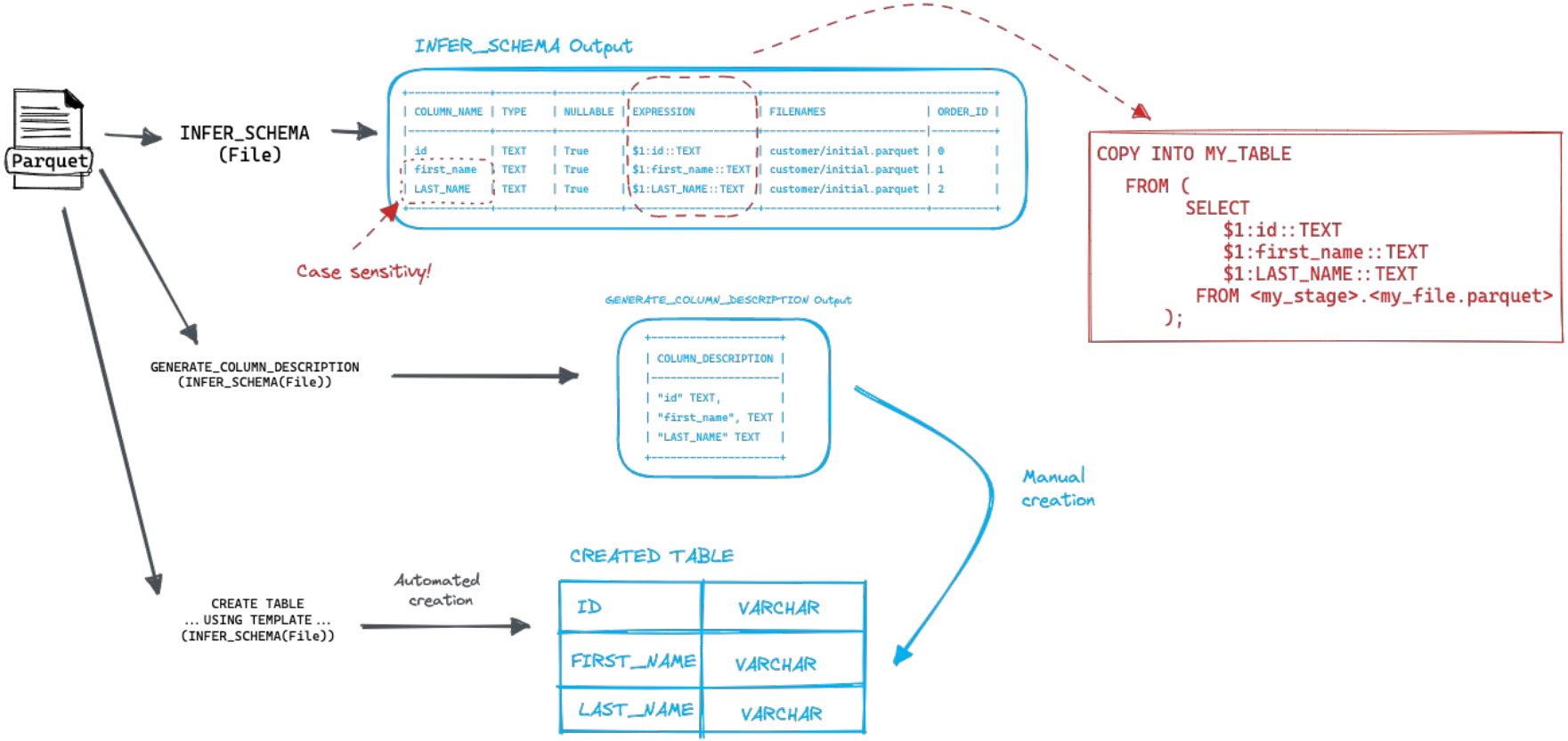
- Faster and more reliable development by eliminating manual tasks which are very error prone

- It works for Parquet, Avro, and ORC

Ingestion Automation schema



Ingestion Automation schema



Exercise

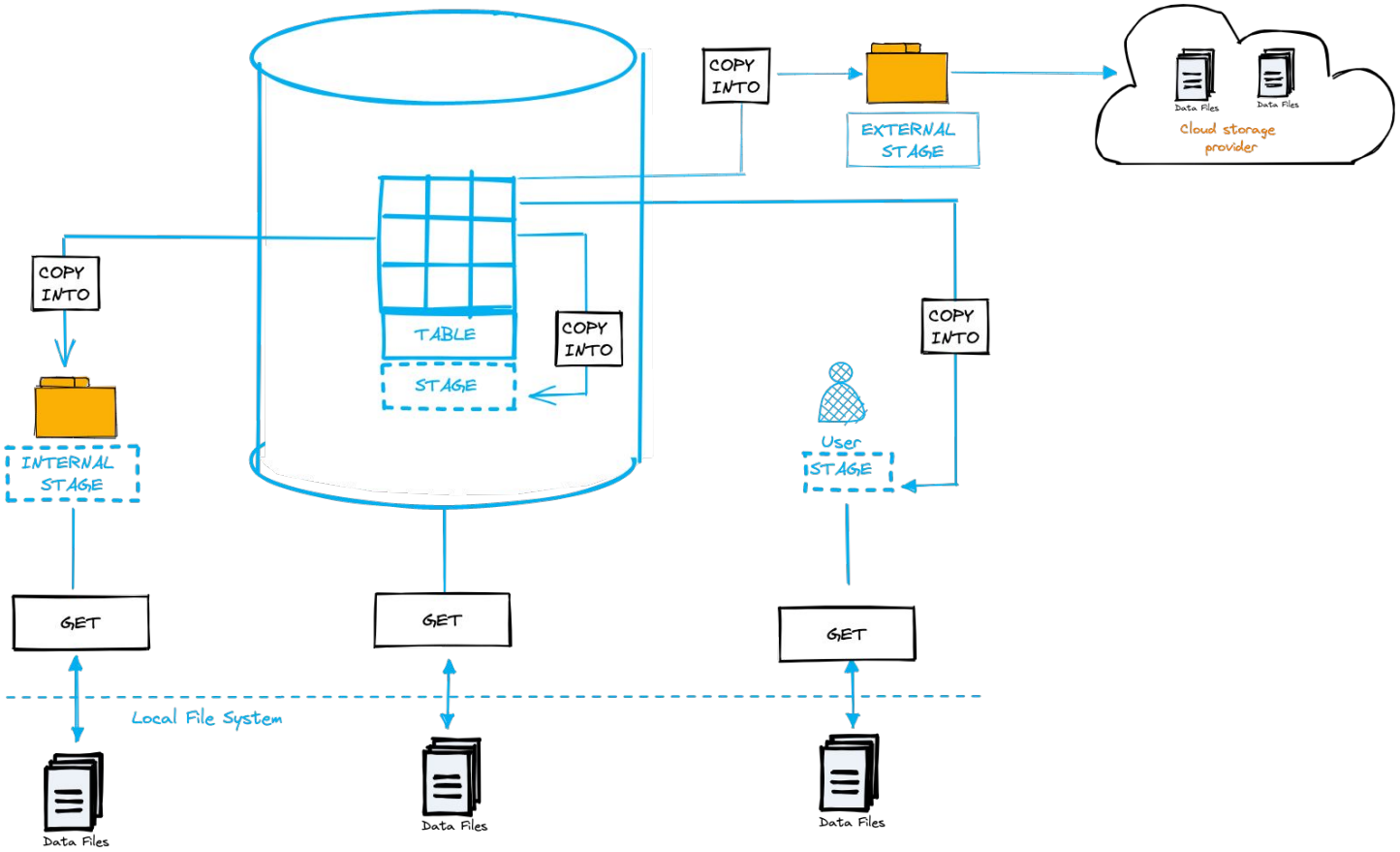
Let’s try to practice the parquet ingestion and how to use the functions for schema detection and ingestion automation.

DATA UNLOADING

Data Unloading

- Sharing Data with partners, co workers (different teams)
- Exporting data out
  - Internal stages
  - External stages
- Utilizing same objects like data import
  - Stages + File Format + storage integration
  - COPY command
- Copy command has different restriction than copy command for import
- Data unload without export -> sharing via various APIs

Unloading schema



Unload syntax

```
COPY INTO @my_stage
    FROM my_data
    FILE_FORMAT = ( FORMAT_NAME = 'my_format')
```

- Can use any SQL command including joins – no limitations
- COPY options related to unloading
  - OVERWRITE
  - SINGLE
  - INCLUDE_QUERY_ID
  - MAX_FILE_SIZE (5 GB, default 16 MB)
- Supports same formats like import (csv, Parquet, JSON, etc.)
- Handling nulls -> part of file format definitions

File paths and names

- Snowflake appends a suffix to the file name. It includes:
  - The number of virtual machine in the virtual warehouse
  - The unload thread
  - A sequential file number
  - Example: **data_0_0_0.csv**

- You can set the file path for the files. Add it after the stage name
  - `COPY INTO @mystage/Unloading/TableX`

- To set the file name for the files, add the name after the folder name
  - `COPY INTO @mystage/Unloading/TableX/exported_data`

Unloading JSON data

- Data needs to be shared via API

1. Unload data into external stage first
2. Share data via API with consumers

- It must be done from VARIANT data type or you need to convert the data back to JSON

- Constructing the JSON structure -> `OBJECT_CONSTRUCT()` + `ARRAY_AGG()` functions

Constructing JSON

- `OBJECT_CONSTRUCT()`
  - Construct an object from arguments
  - Returns object
  - Arguments = key – value pairs
  - Nested calls are supported
  - It supports expressions and queries to add

```
SELECT OBJECT_CONSTRUCT('a',1,'b','BBBB')
```

Constructing JSON

- `ARRAY_AGG()`
  - Input values are pivoted into an array
  - Returns array type value
  - Arguments – values to be put into the array and values determining the partition into which to group the values
  - Supports DISTINCT

```
SELECT ARRAY_AGG(O_ORDERKEY) WITHIN GROUP (ORDER BY O_ORDERKEY ASC) FROM orders
```

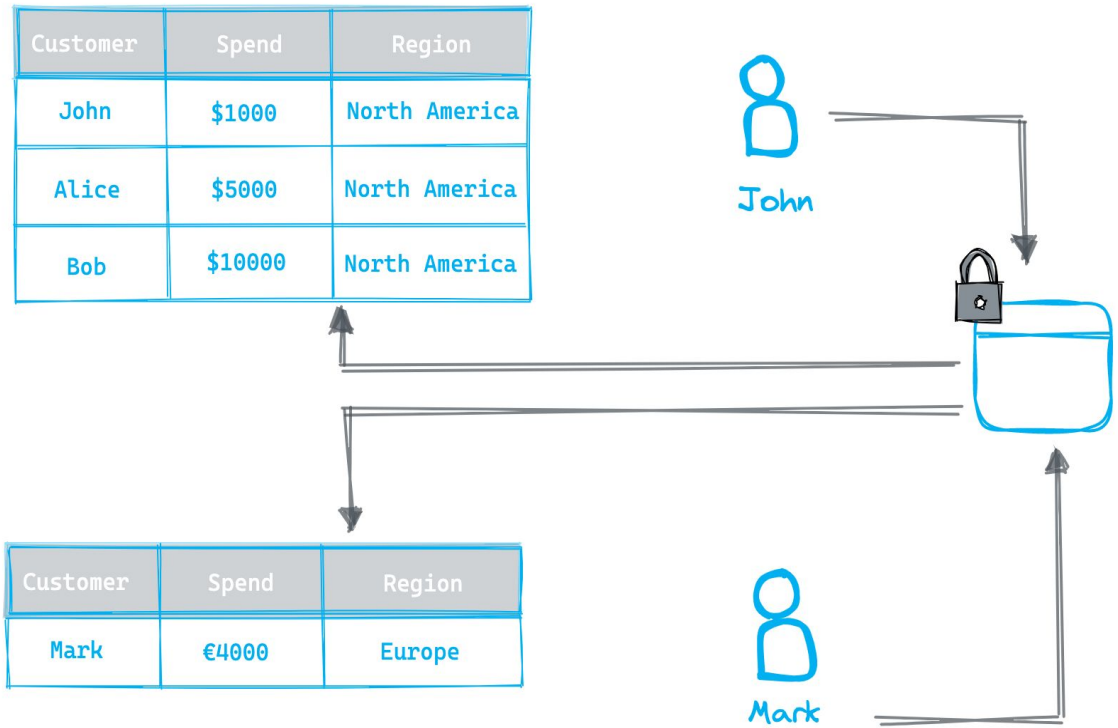
Exercise

Let’s practise JSON creation by using functions like ARRAY_AGG() and OBJECT_CONSTRUCT()

Row and Column-Level Security Features

Row Access Policies

Row-level security



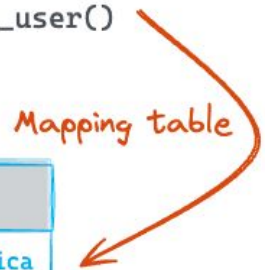
- Filter unauthorized rows at query time
- Authorization can be based on:
  - User roles
  - Mapping tables
- One policy can be applied to many tables
- Central management

Row Access Policies details

- Applied on query run time

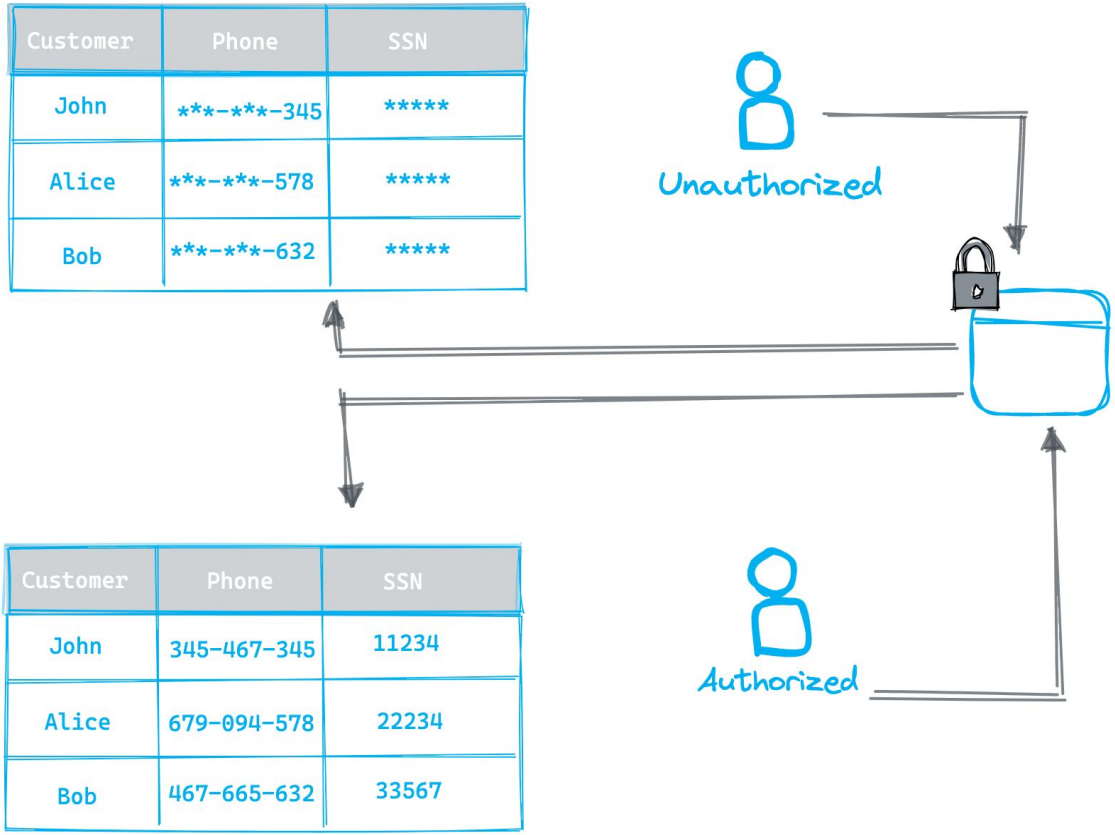
- All rows and all accesses are protected (including table/view owner)

```
create or replace row access policy sales as (region varchar) returns boolean ->
  'sales_executive_role' = current_role()
      or exists (
            select 1 from sales_regions
              where sales_manager = current_user()
                and region = region
          )
;
```



Sales_manager	Region
John	North America
Alice	North America
Mark	Europe

Dynamic Data Masking



- Column level security
- Dynamically mask data at query run time
- Centralized management
- Apply one policy to many columns

Dynamic Data Masking Details

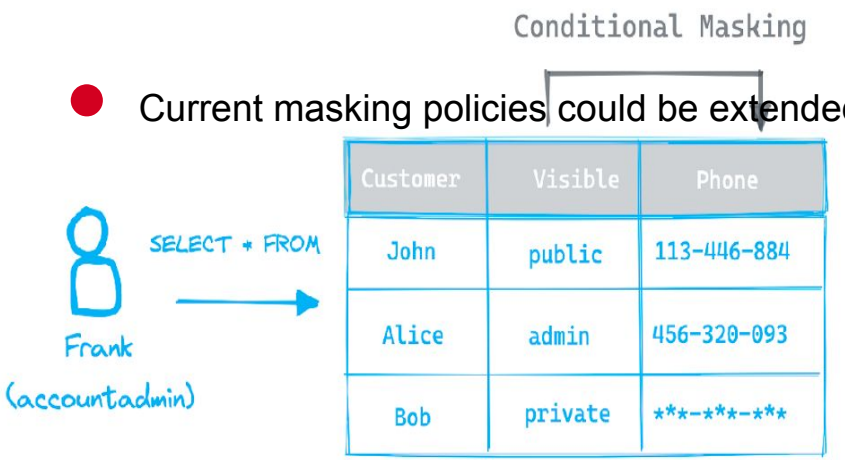
- Can be assigned at creation of table/view or later
- Policy management should be done by security/privacy officers - create a special role for policy

```
create or replace masking policy email_mask as (val string) returns string ->
  case
    when current_role() in ('ANALYST') then val
    else '*********'
  end;

case
  when current_role() in ('ANALYST') then val
  when current_role() in ('SUPPORT') then regexp_replace(val,'.+\@','*****@')
  else '********'
end;
```

Conditional Masking

- Mask data based on value from different column
- No change to stored data
- Unmask them only for authorized users
- Current masking policies could be extended

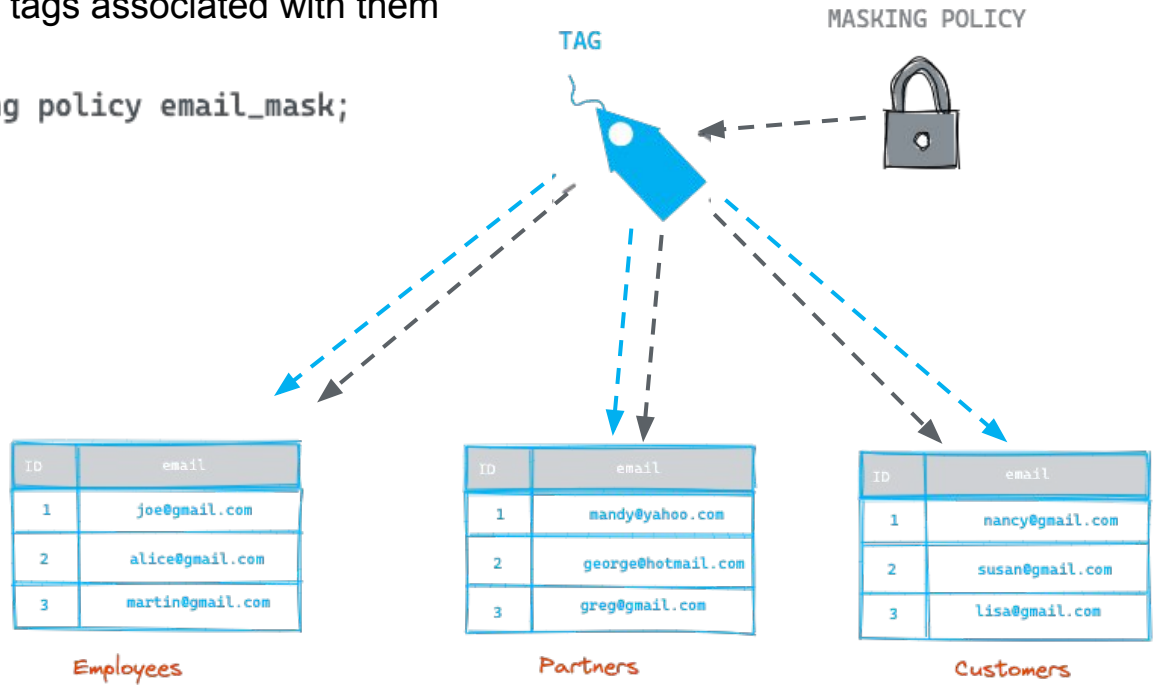



```
CREATE MASKING POLICY conditional_mask_phone
AS
(val STRING, visible STRING) RETURNS STRING ->
CASE
   WHEN current_role() = 'ACCOUNTADMIN' and visible='admin' THEN val
   WHEN visible='public' THEN val
   ELSE '******'
END;
```

Tag-Based Masking

Mask columns automatically based on tags associated with them

```
alter tag personal_email set masking policy email_mask;
```



Limitations:

- One masking policy per data type per tag
- Neither tag nor masking policy can be dropped when masking policy is assigned

Main Benefits

Easy to Use

Write the policy once and use it multiple times, across DBs and schemas

Change Management

You can change just policy definitions without touching the tables

SoD

Segregation of duties and different management approaches

Exercise

- Let’s create dynamic data masking policy to try how data masking works
- We do it in two ways
  - Assign masking policy to columns directly
  - Assign masking policy to tag and then assign tag to columns