



# SNOWFLAKE

## Advanced



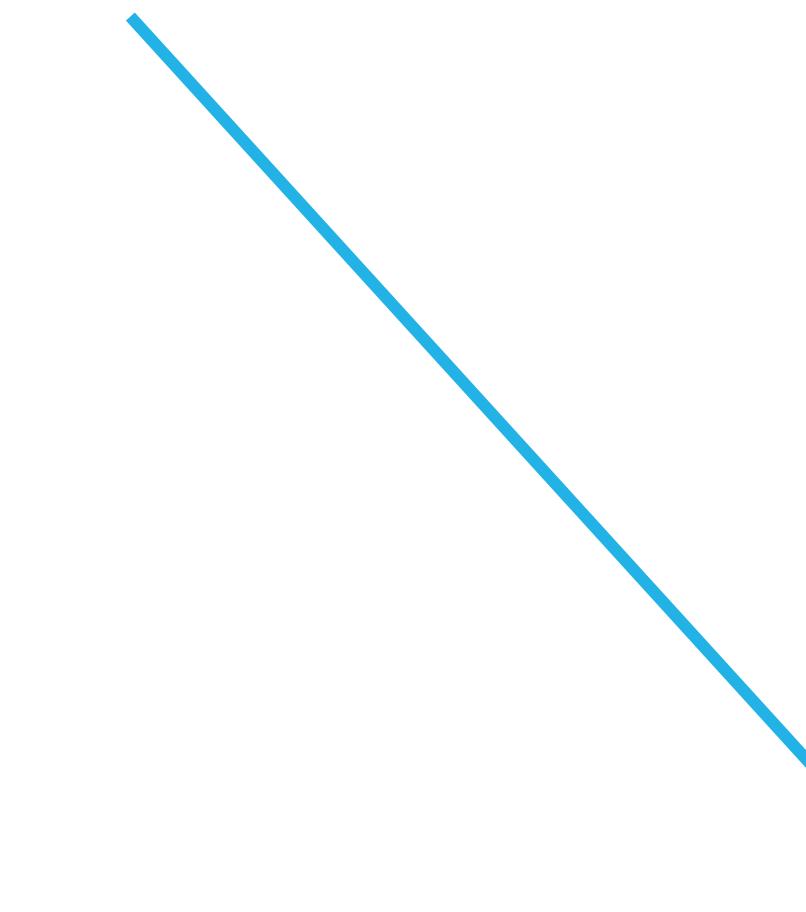
# INTRODUCTIONS

- Name
- Company & Role
- Experience with Snowflake
  - What is your experience with Snowflake?
  - How is your company using Snowflake?
- Tell us something about yourself



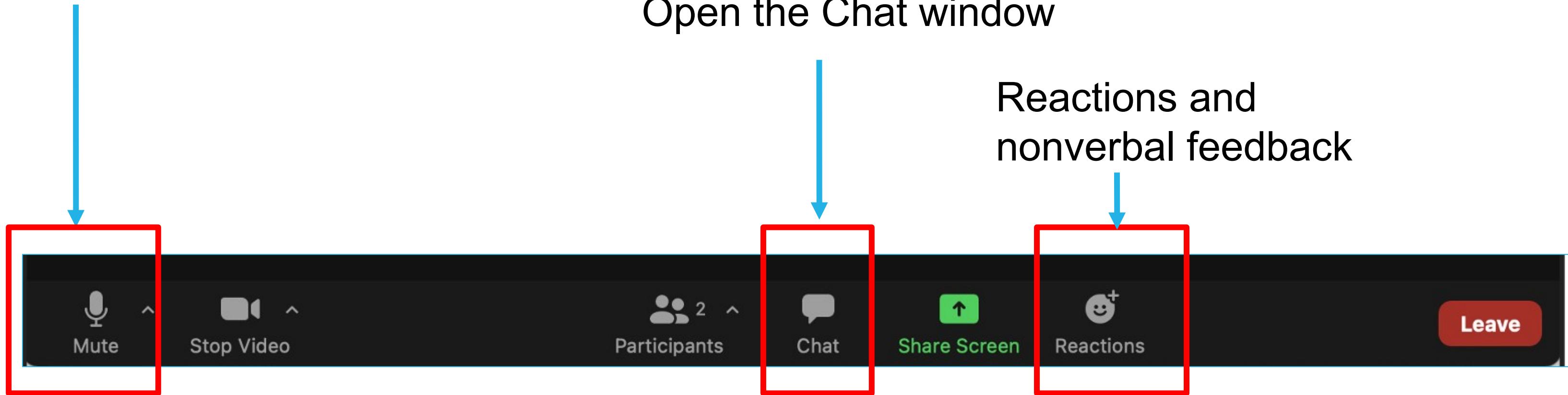
# REMOTE ATTENDEE CONTROLS

- Hover over your Zoom interface to display the control panel at the bottom

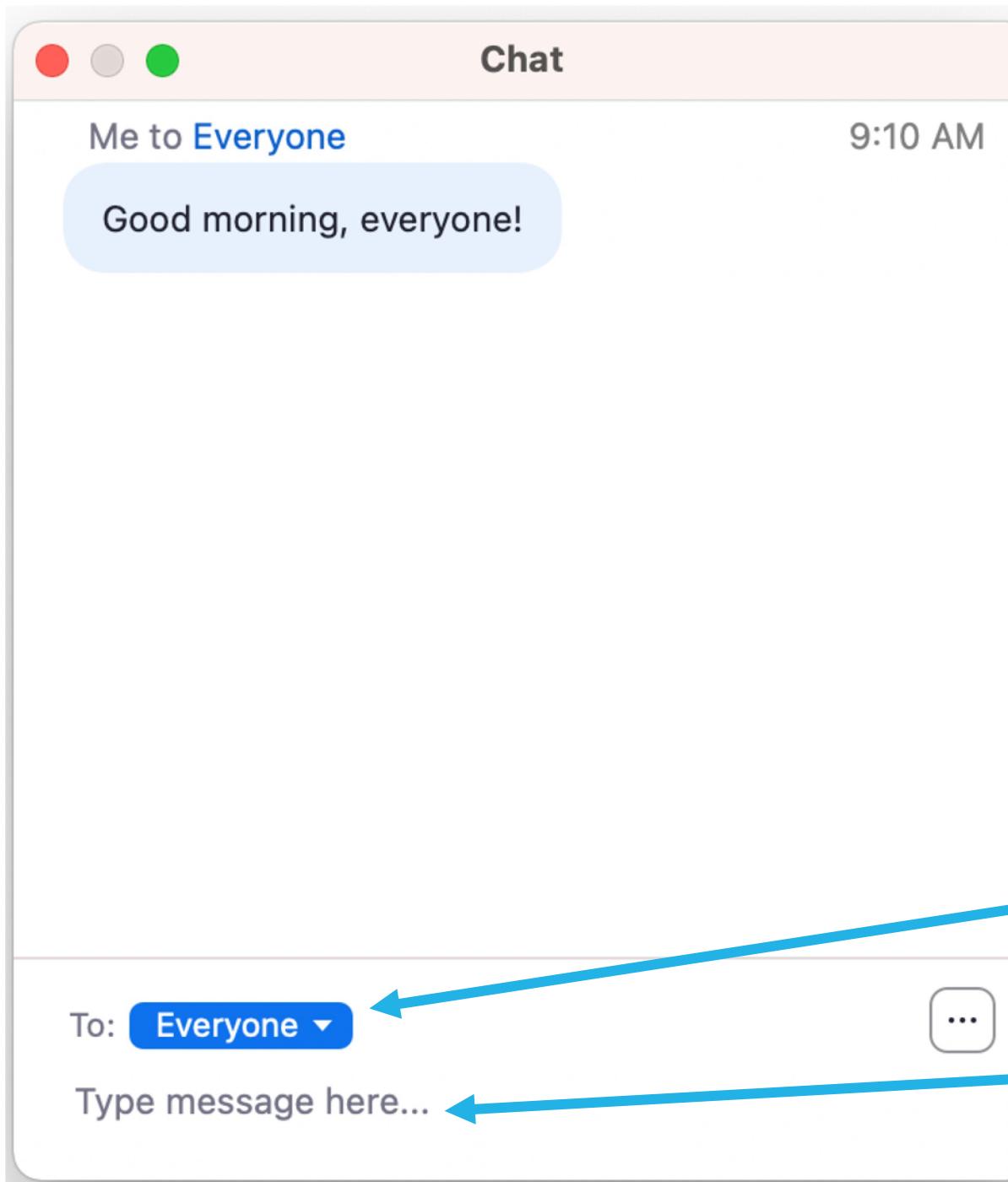


# CONTROL PANEL

Mute/Unmute



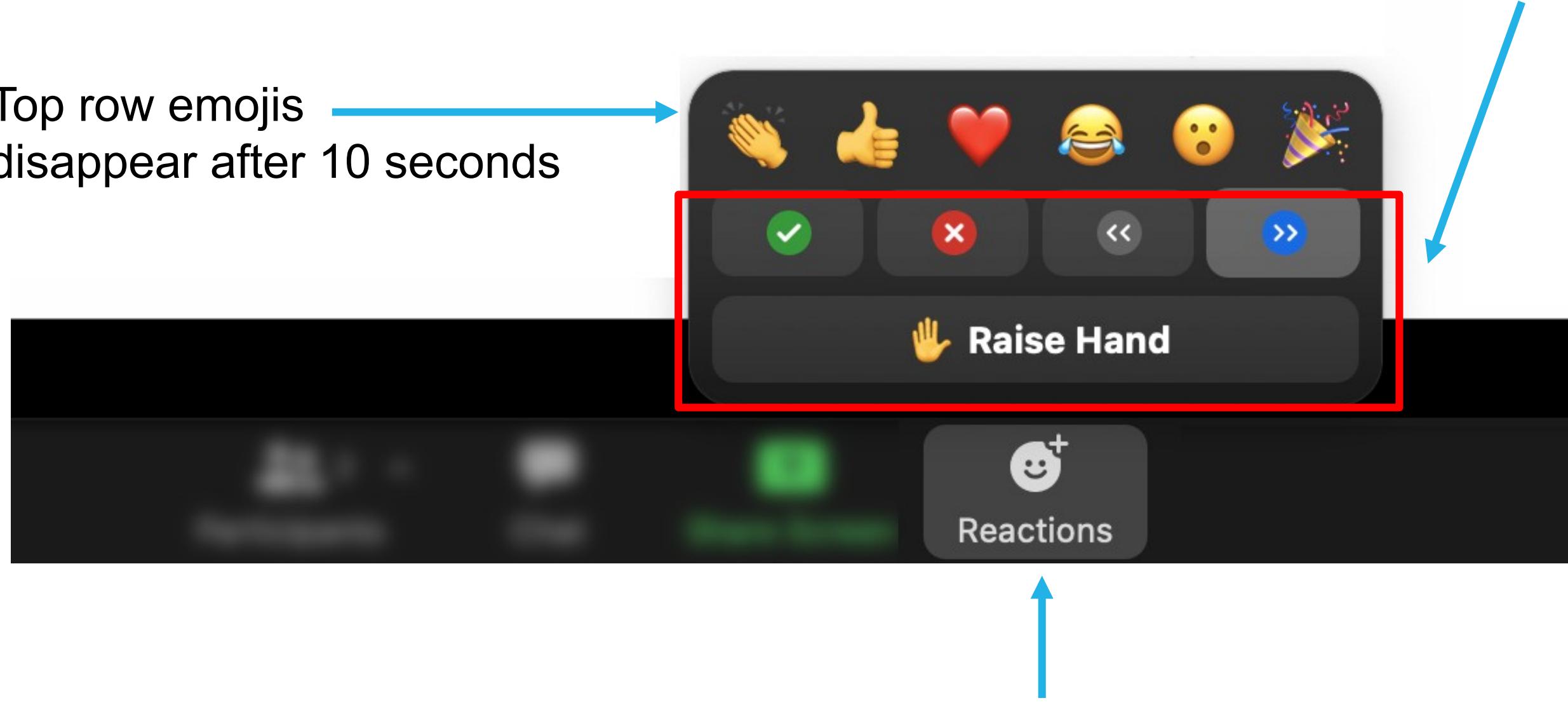
# CHAT INTERFACE



1. Use drop-down to choose recipient
2. Click to type your message and press Enter

# REACTIONS AND NONVERBAL FEEDBACK

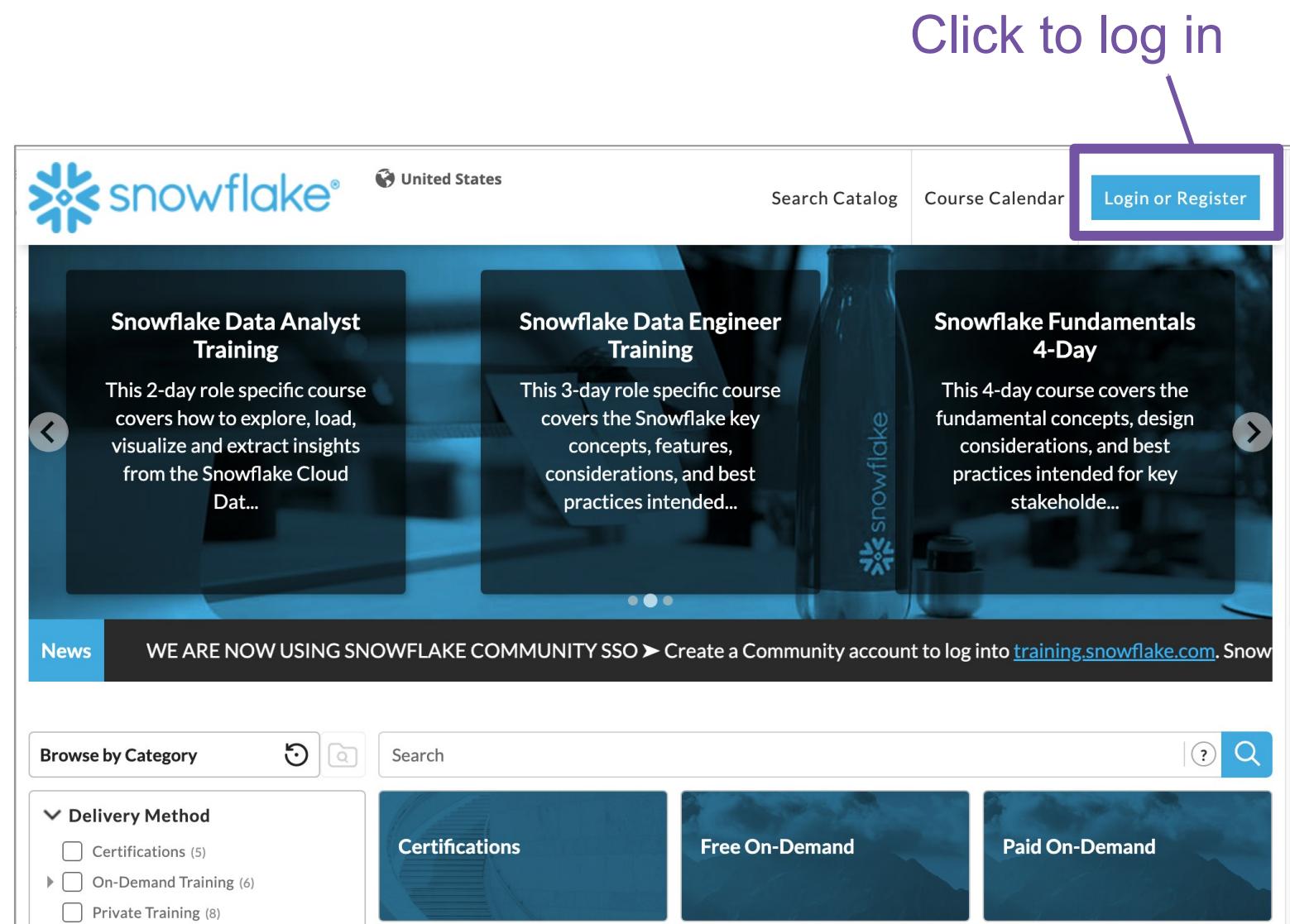
Top row emojis  
disappear after 10 seconds



Use bottom two rows to raise your hand,  
click Yes/No, etc. (click again to cancel)

# LOG IN TO YOUR COURSE DASHBOARD

Go to <https://training.snowflake.com>  
and click the "Login or Register" Button



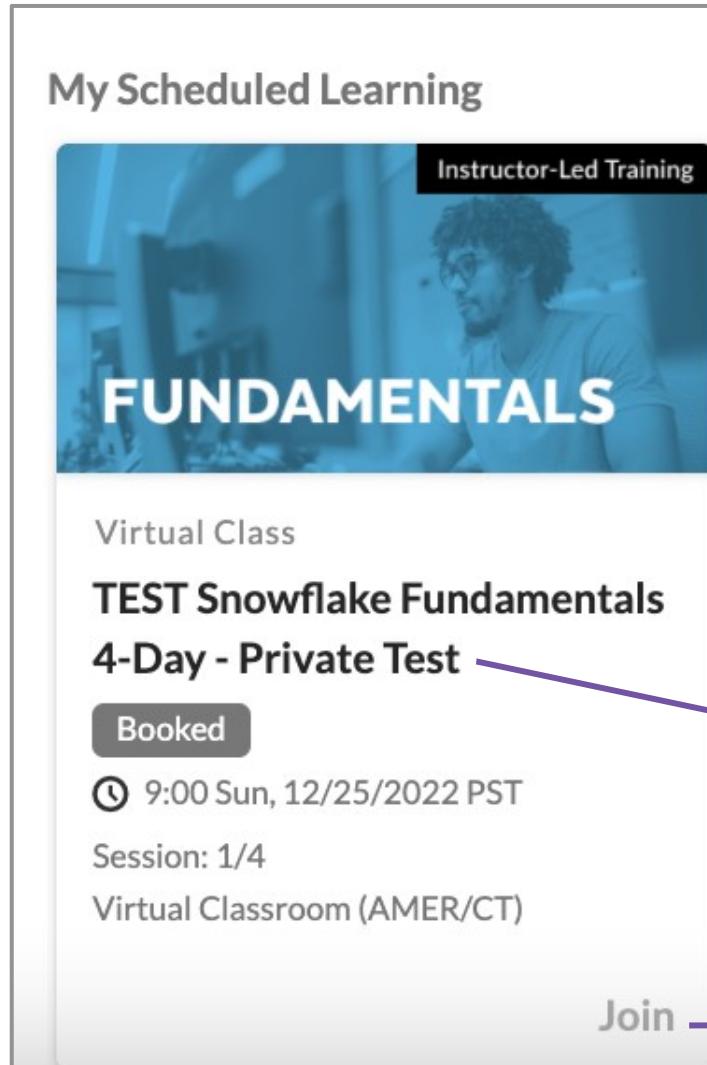
Log in with your  
Snowflake Community credentials

Type email address here

The screenshot shows the 'Login or Register' page. It starts with a message: 'Please enter your email address below and click 'Next' to continue.' Below this is a form with a text input field labeled 'Enter Your Email Address' and two buttons: 'Next' (blue) and 'Cancel' (white). At the bottom of the page, there's a link 'Don't have an account? Create Account'.

Don't have a Snowflake Community login?  
Click "Create Account."

# CLICK TO JOIN SESSION



Your course name  
will be **different** from  
what is shown here

**Join here** each  
day of the course  
(link active about  
30 minutes ahead)

# NAVIGATE TO YOUR COURSE PAGE

The screenshot shows a course tile for "TEST Snowflake Fundamentals 4-Day - Private Test". The tile includes the course name, a "Booked" status indicator, the start date and time (9:00 Sun, 12/25/2022 PST), the session count (Session: 1/4), and the virtual classroom location (Virtual Classroom (AMER/CT)). A purple arrow points from the text "Click tile to view details" to the course name on the tile.

My Scheduled Learning

Instructor-Led Training

FUNDAMENTALS

Virtual Class

TEST Snowflake Fundamentals  
4-Day - Private Test

Booked

9:00 Sun, 12/25/2022 PST

Session: 1/4

Virtual Classroom (AMER/CT)

Join

Click tile to view details

Your course name  
will be **different** from  
what is shown here

Join here each  
day of the course  
(link active about  
30 minutes ahead)

The screenshot shows the course page for "Snowflake Fundamentals 4-Day - Private". It displays the course title, a brief description, and three session details. The first session is listed as "Booked". A purple arrow points from the text "Join here each day of the course (link active about 30 minutes ahead)" to the "Join" button for Session 1. The page also includes sections for "Details", "To-do", and "Reference Materials".

Home | Catalog | My Learning | Reporting | eCommerce

Snowflake Fundamentals 4-Day - Private

This 4-day course covers the fundamental concepts, design considerations, and best practices intended for key stakeholders who will be working on the Snowflake Cloud Data Platform. The course will consist of lecture, demos, and labs on a wide range of foundational topics.

Unenroll To Do

Sessions

Session 1 Join  
Sun, 12/25, 12:00 - 20:00 (GMT-5) EST  
Virtual Classroom (AMER/CT)

Session 2 Join  
Mon, 12/26, 12:00 - 20:00 (GMT-5) EST  
Virtual Classroom (AMER/CT)

Session 3 Join  
Tue, 12/27, 12:00 - 20:00 (GMT-5) EST  
Virtual Classroom (AMER/CT) Pazmany, Nicole

ILT Post Class Survey Play  
Type: Post-Work Online  
Date Available: 12/28/2022 20:00 EDT - Completion Required Status: Not Attempted

My Status  
Status Booked Date Enrolled 03/30/2022

Unenroll

Reference Materials  
Fundamentals Slides (15.2 MB)  
Fundamentals Lab Files (2.2 MB)  
Fundamentals Workbook (2.3 MB)

# ACCESS THE COURSE MATERIALS

The screenshot shows a course titled "Snowflake Fundamentals 4-Day - Private" listed as a "Live Virtual Class". The course description states: "This 4-day course covers the fundamental concepts, design considerations, and best practices intended for key stakeholders who will be working on the Snowflake Cloud Data Platform. The course will consist of lecture, demos, and labs on a wide range of foundational topics." Below the title are two buttons: "Unenroll" (blue) and "To Do" (green). The "Sessions" section lists three sessions:

- Session 1**: Sunday, 12/25, 12:00 - 20:00 (GMT-5) EST, Virtual Classroom (AMER/CT). Join button.
- Session 2**: Monday, 12/26, 12:00 - 20:00 (GMT-5) EST, Virtual Classroom (AMER/CT). Join button.
- Session 3**: Tuesday, 12/27, 12:00 - 20:00 (GMT-5) EST, Virtual Classroom (AMER/CT). Join button. Instructor listed: Pazmany, Nicole.

Below the sessions are tabs for "Details" (selected) and "To-do". Under "Details", there is a section for "ILT Post Class Survey" with status "Not Attempted".

To the right, a detailed view of Session 1 is shown:

- Instructor-Led Training**
- FUNDAMENTALS**
- Live Virtual Class**
- Identifier**: EDU-FUND-PVT-4D
- Training Funds**: 28000
- Start Date**: 12/25/2022 12:00 EST
- End Date**: 12/28/2022 20:00 EST
- Latest Cancellation**: 11/25/2022 03:00 EST
- Availability**: Places Available
- Delivery Language**: English (en-us)
- Tag**: Fundamentals
- My Status**: Status Booked, Date Enrolled: 03/30/2022
- Unenroll** button

A purple box highlights the "Reference Materials" section at the bottom of the detailed view, which includes links to "Fundamentals Slides (15.2 MB)", "Fundamentals Lab Files (2.2 MB)", and "Fundamentals Workbook (2.3 MB)".

Your course title may be different from what is shown

Use links to access materials  
(file names will vary from what is shown here)



# COURSE AGENDA

- Overview
- Architecture Highlights
- Connecting to Snowflake
- Data Loading and Troubleshooting
- Working with Semi-Structured Data
- Unloading Data
- Working with Unstructured Data
- Working with Data Lakes
- Data Governance in Snowflake
- Advanced Features
- Streams (Change Data Capture)
- Database Replication
- Data Sharing
- Data Clustering
- Performance Optimization



# OVERVIEW



# SNOWFLAKE – THE CLOUD DATA PLATFORM



# PLATFORM REQUIREMENTS

## FAST FOR ANY WORKLOAD



Run any number or type of job across all users and data volumes quickly and reliably.

## IT JUST WORKS



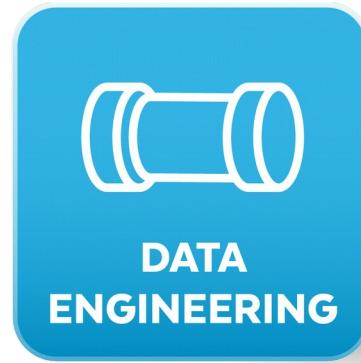
Replace manual with automated to operate at scale, optimize costs, and minimize downtime.

## CONNECTED TO WHAT MATTERS

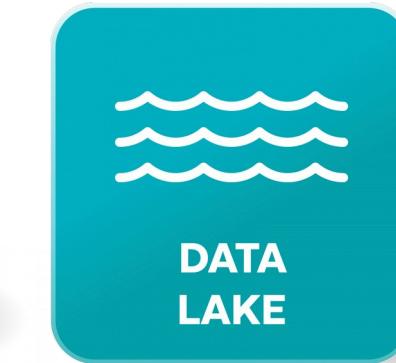


Extend access and collaboration across teams, workloads, clouds, and data, seamlessly and securely.

# POWERING MANY WORKLOADS



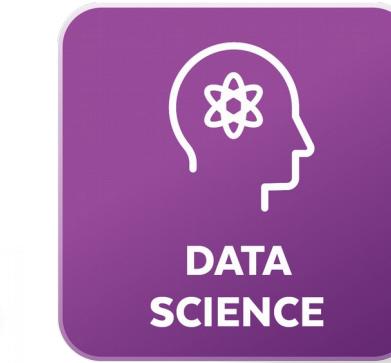
DATA  
ENGINEERING



DATA  
LAKE



DATA  
WAREHOUSE



DATA  
SCIENCE



DATA  
APPLICATIONS



DATA  
SHARING

Fast, reliable pipelines in the language of your choice

Governed, cost-effective data lake for any scale and any data

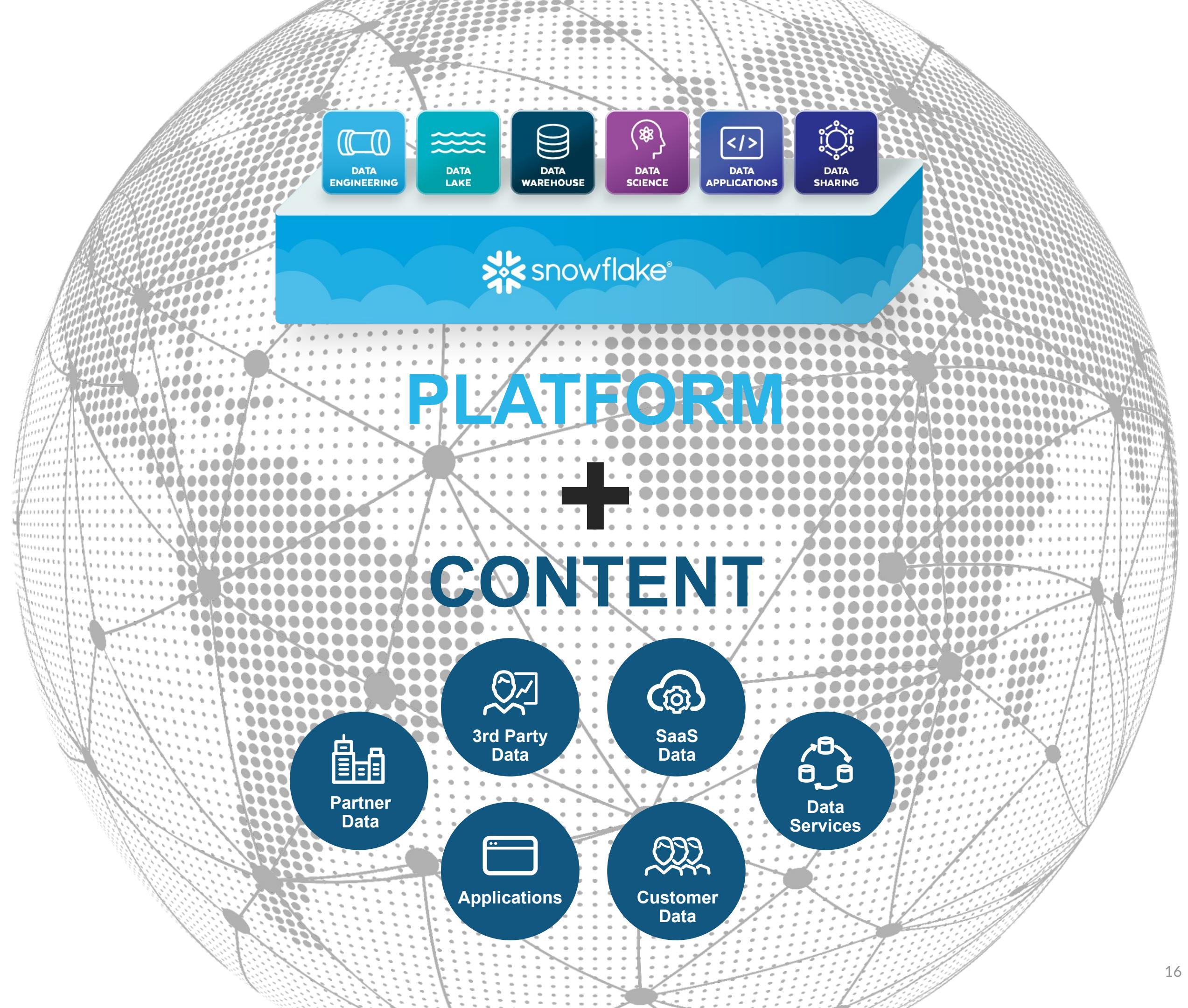
Analytics at scale on all your data with zero administration

Simple data preparation for modeling with your framework of choice

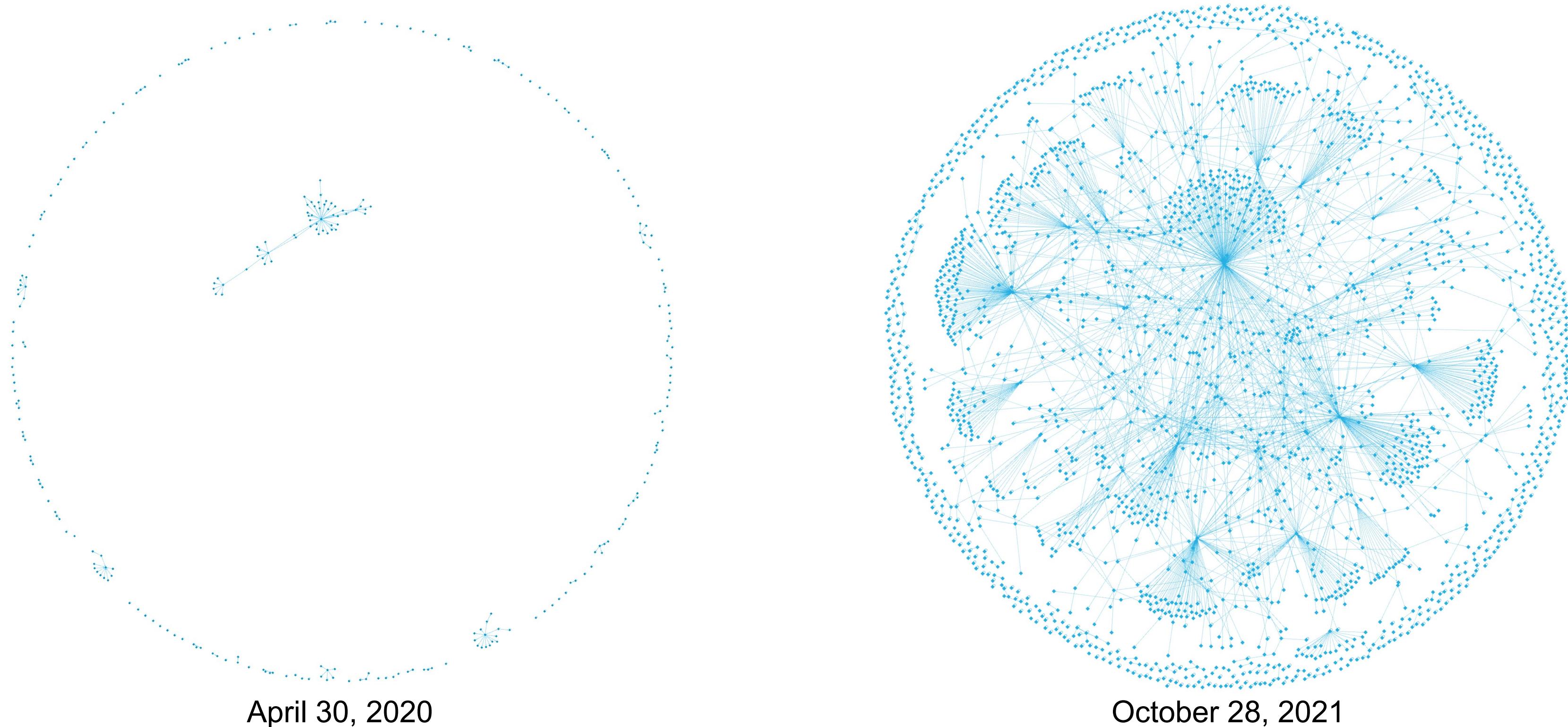
Build data-intensive applications without operational burden

Share and collaborate on live data across your business ecosystem

# ELEMENTS OF THE DATA CLOUD



# DATA CLOUD GROWTH



Visualizations based on actual Data Cloud sharing activity as of April 30, 2020 and Oct 28, 2021, as applicable (Represents stable edges between two Snowflake accounts)



© 2022 Snowflake Computing Inc. All Rights Reserved

# ARCHITECTURE HIGHLIGHTS



# MODULE AGENDA

- Three Architectural Layers
- Snowflake Object Hierarchy
- Caching



# THREE ARCHITECTURAL LAYERS

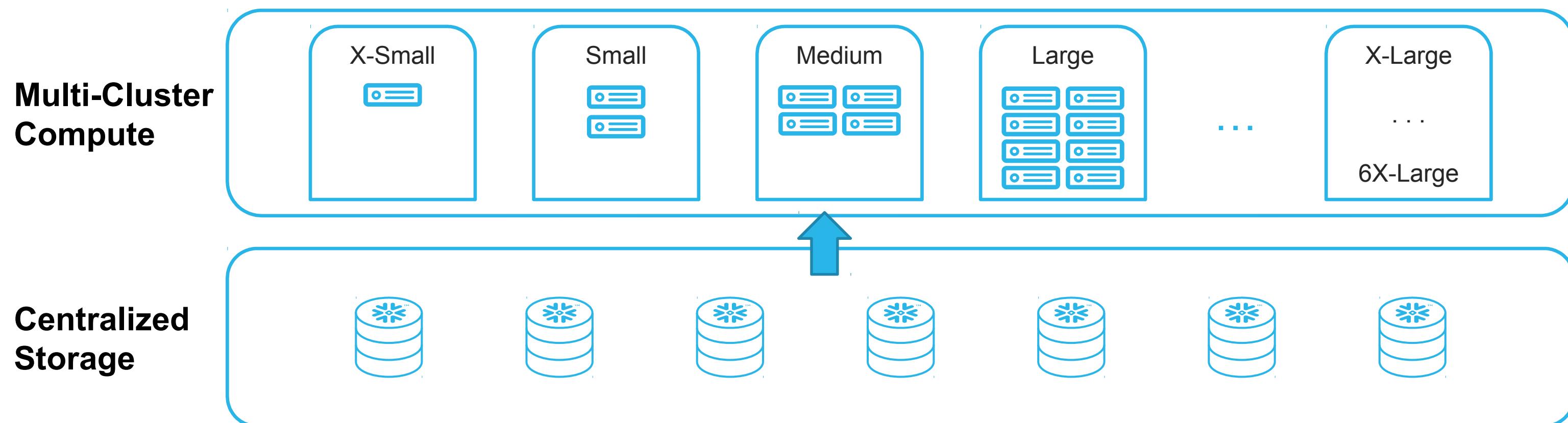


# STORAGE LAYER

**Centralized  
Storage**

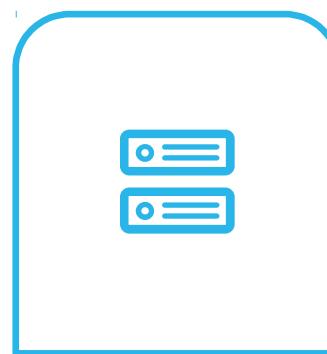


# COMPUTE LAYER

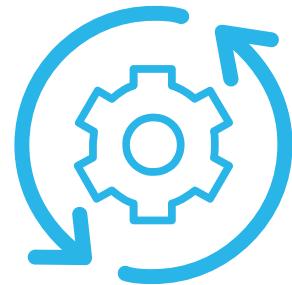


# TYPES OF COMPUTE

Virtual Warehouses



Serverless Features



Cloud Services Compute



- **Virtual Warehouses**

- Customer allocated for query execution
- Charged per second (one-minute minimum)

- **Serverless Features**

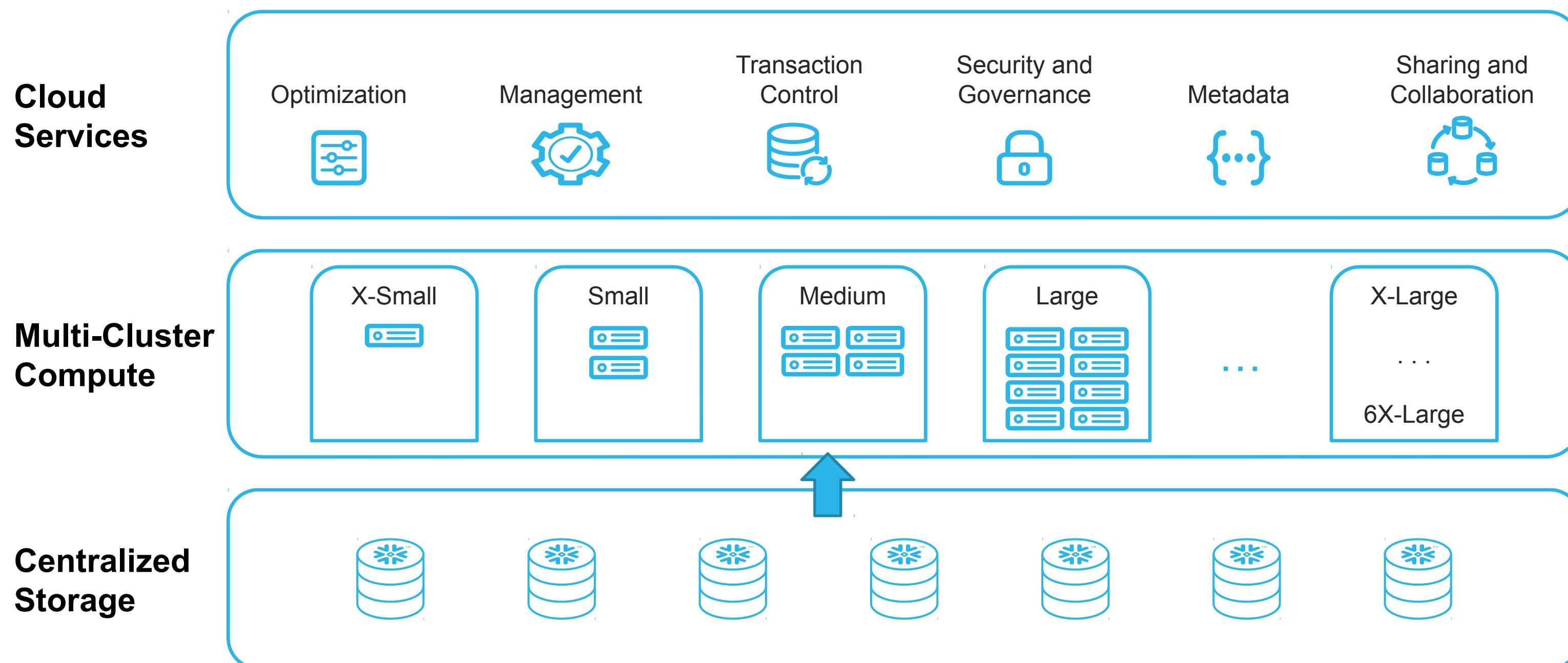
- Provisioned by Snowflake
- For automated features (snowpipe, materialized view refresh, etc.)
- Charged per second per core

- **Cloud Services**

- Cache lookups, query optimization, etc.
- Charged per second per core
- Only charged when > 10% of daily compute



# CLOUD SERVICES LAYER



Google Cloud



© 2022 Snowflake Computing Inc. All Rights Reserved

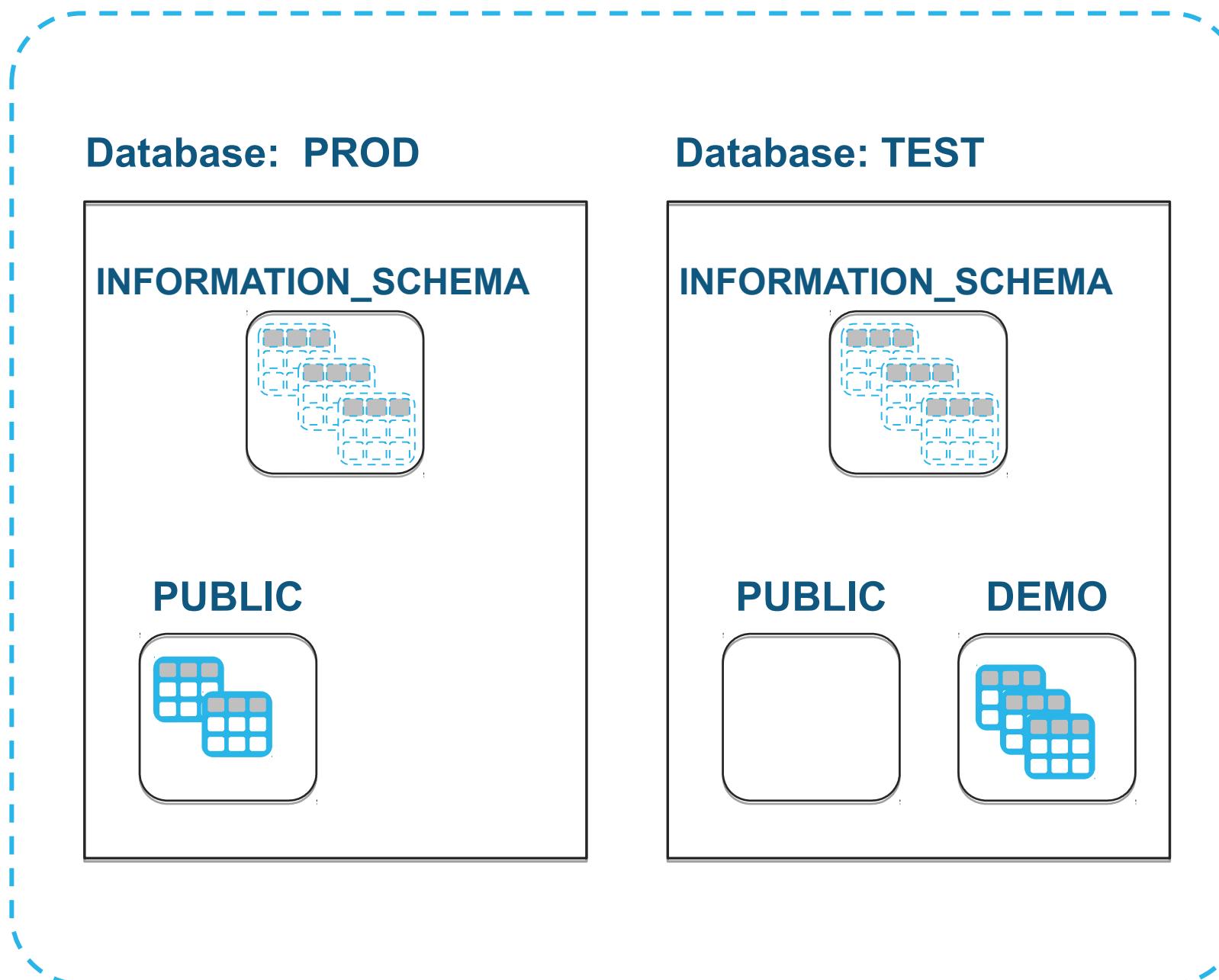


# **SNOWFLAKE OBJECT HIERARCHY**



# OBJECT HIERARCHY

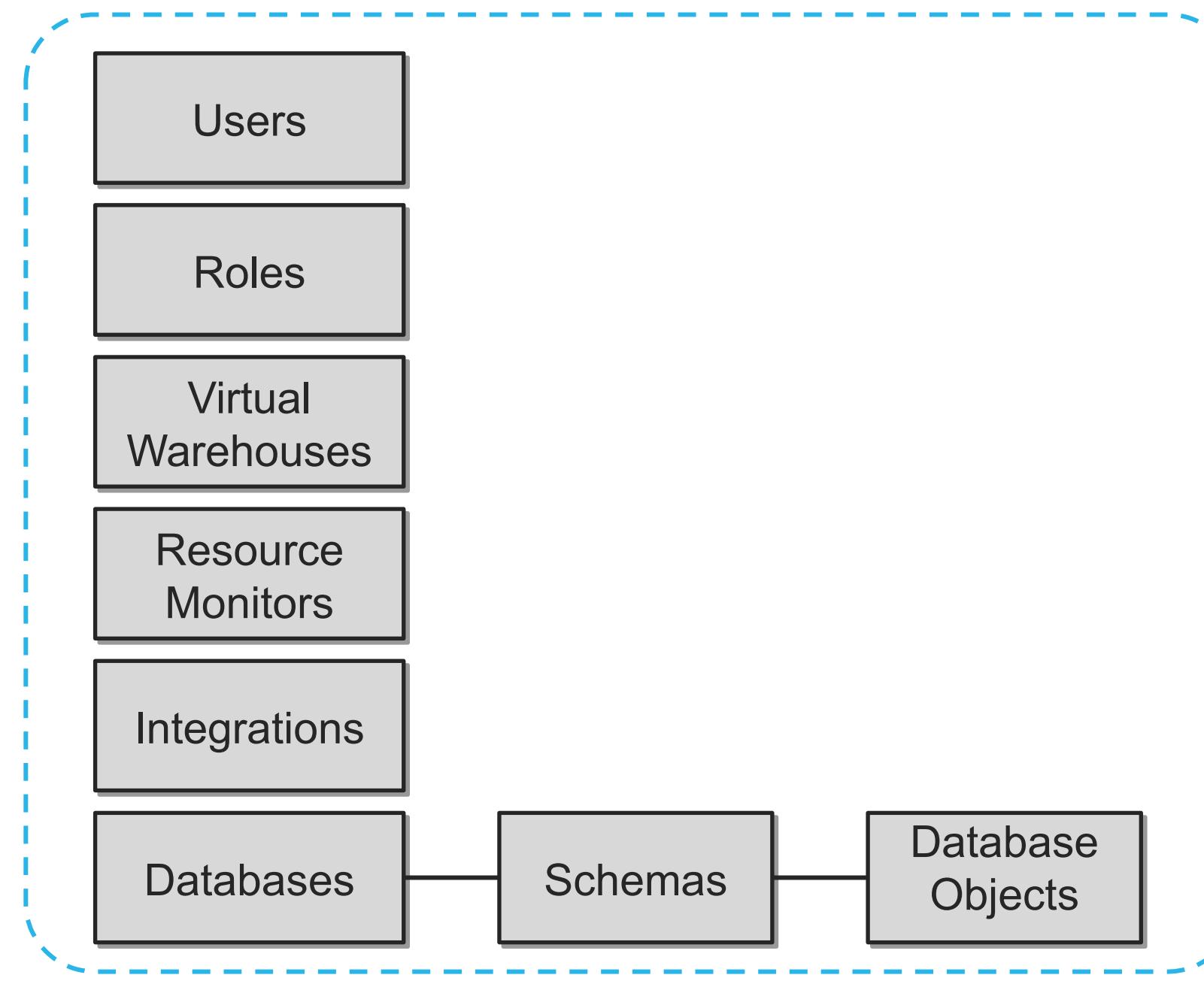
## Snowflake Account



- Account: Users log in to an account
- Database: A logical container to group schemas
- Schema: A logical container to group database objects (tables, views, etc.)
  - PUBLIC and INFORMATION\_SCHEMA created automatically
- Create additional custom databases and schemas as needed

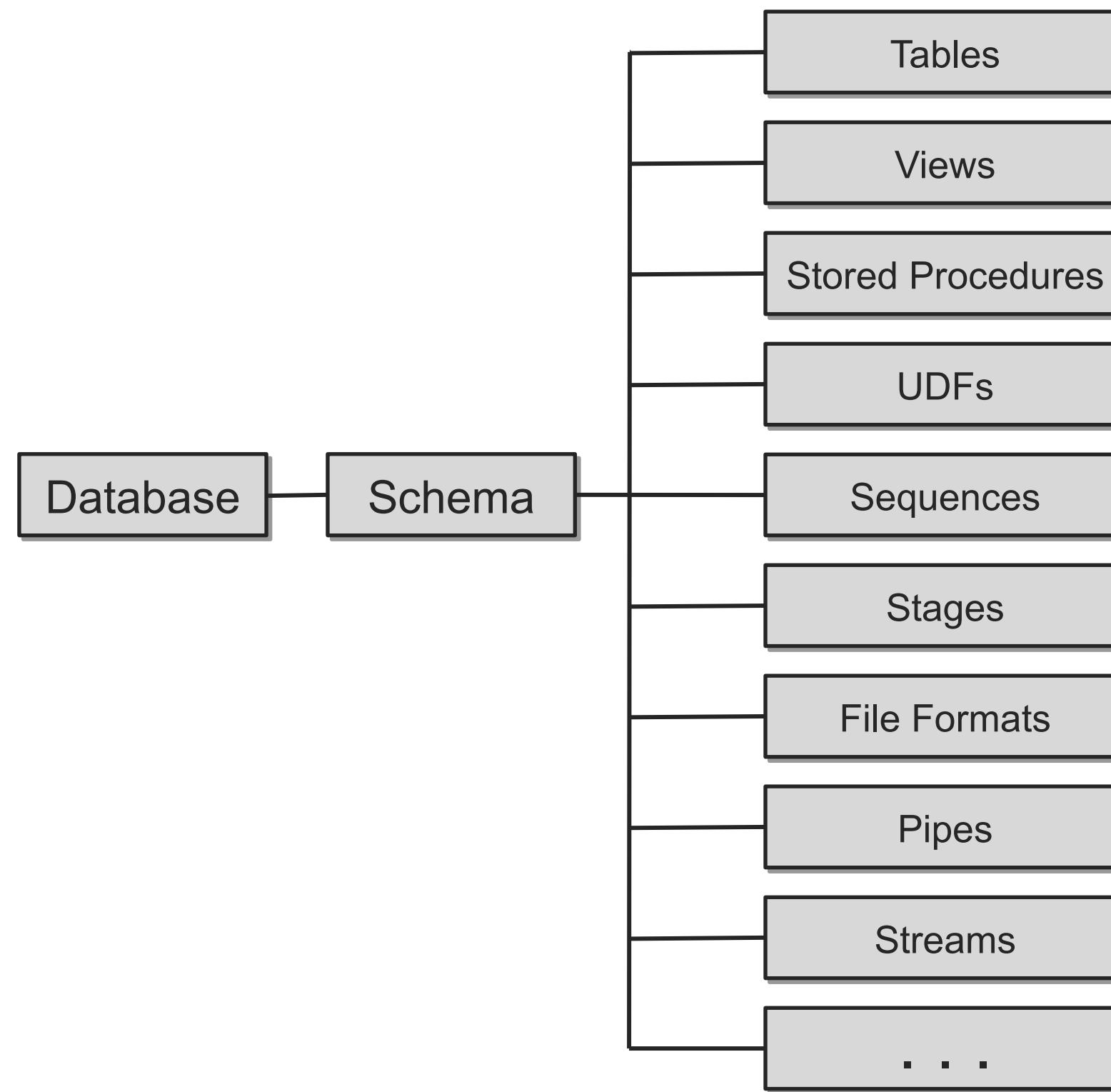


# ACCOUNT HIERARCHY



- Account-level object names: unique
- Recommendation: create a standard naming schema
  - By data type
  - By business unit
  - Other meaningful criteria for your account
- Example:
  - PROD\_DB
    - RAW\_DATA
    - CONFORMED\_DATA
  - TEST\_DB
    - RAW\_DATA
    - CONFORMED\_DATA

# SCHEMA-LEVEL OBJECTS



- A schema can contain unlimited objects
- Objects live within a specific schema
- Object names can be duplicated if the full path (*database.schema.object*) is unique

# CACHING



# SNOWFLAKE MULTI-LEVEL CACHE

- Three distinct caches help optimize query performance



# METADATA CACHE

- Three distinct caches help optimize query performance

Cloud  
Services

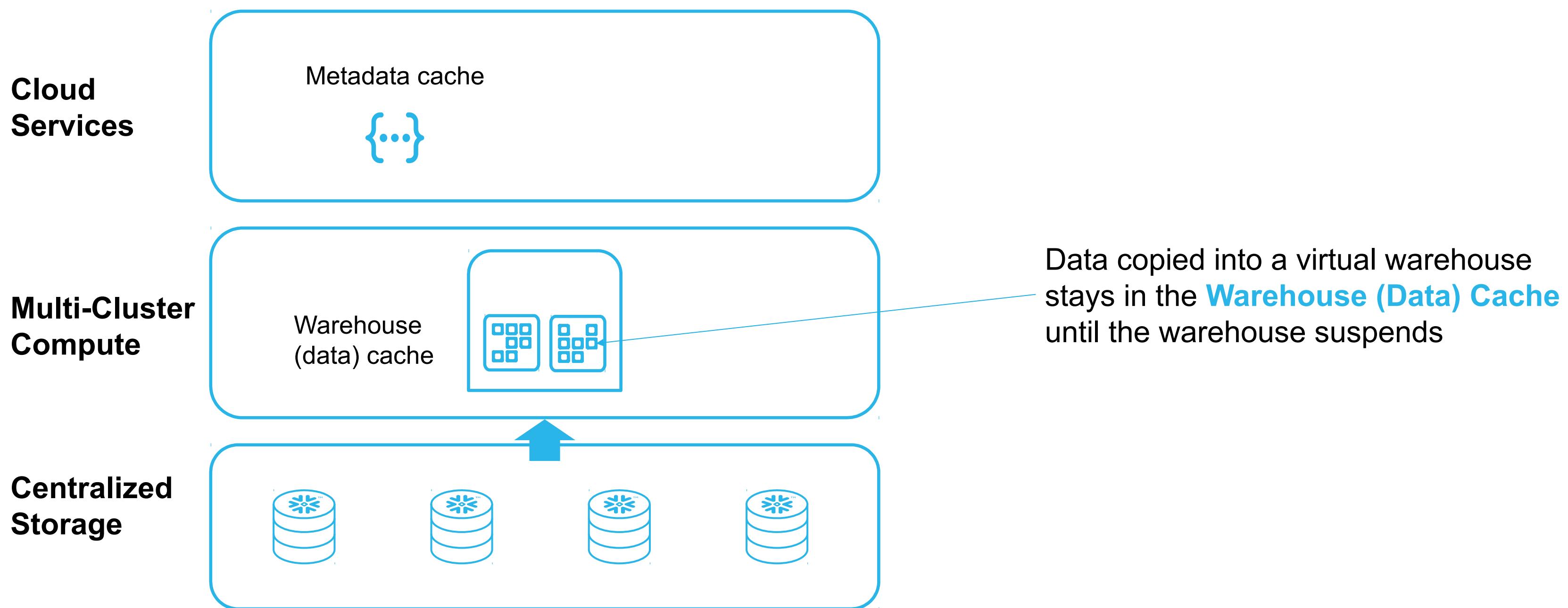
Metadata cache  


The **Metadata Cache** is populated as data is loaded into the system, or as changes to the data are committed



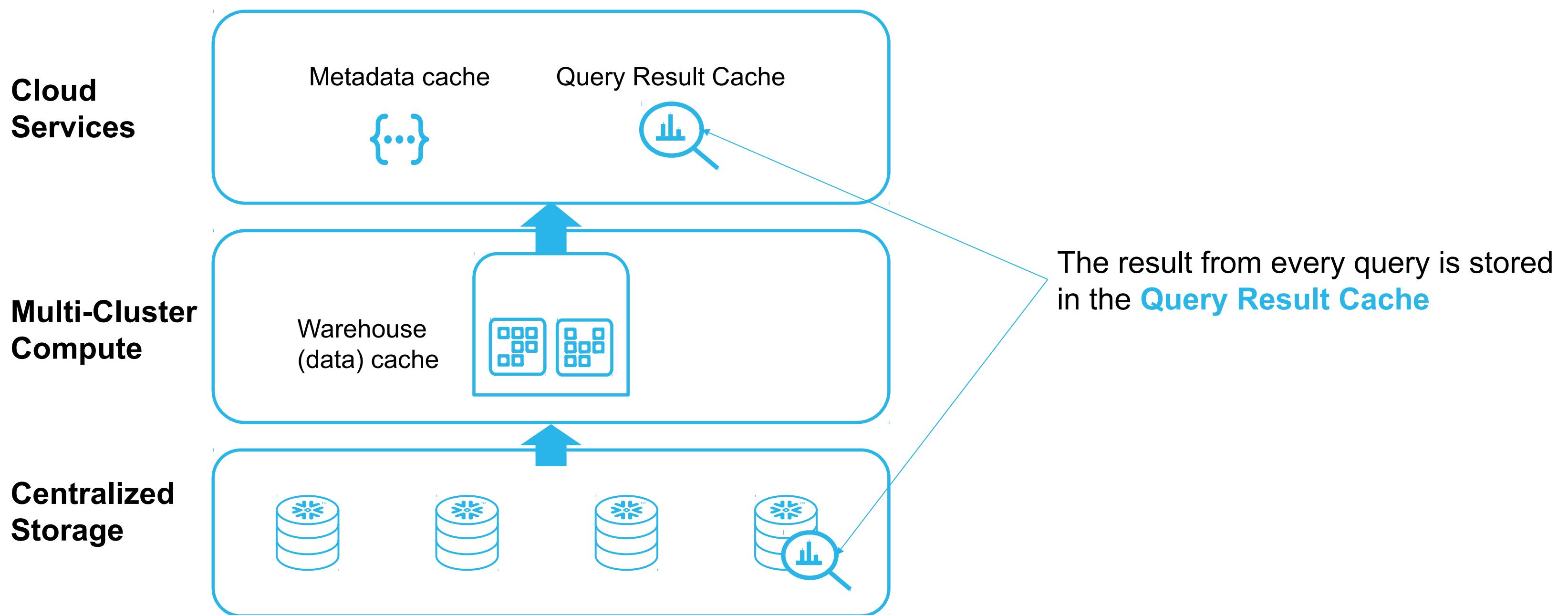
# WAREHOUSE (DATA) CACHE

- Three distinct caches help optimize query performance



# QUERY RESULT CACHE

- Three distinct caches help optimize query performance



# LAB EXERCISE: 1

## Course Lab Setup and Snowsight Review

40 minutes

In this lab, you will connect to the Snowflake User Interface (Snowsight), explore the menu options, and create some objects that will be used in later labs. You will also practice some of the skills you learned in Fundamentals, as a refresher.

### Learning objectives:

- Navigate the user interface
- Create objects
- Explore the object hierarchy
- Refresh prior learning



# CONNECTING TO SNOWFLAKE



# MODULE AGENDA

- What is an Integration?
- Security Integrations
- Key-Pair Authentication



# WHAT IS AN INTEGRATION?



# WHAT IS AN INTEGRATION?

- An object that simplifies and secures repeated operations that require communication outside of snowflake
  - Sits outside of Snowflake
- Credential-less, yet secure, access to Snowflake from clients
- Types of integrations:
  - Security integrations
  - Storage integration
  - Notification integration
  - API integration

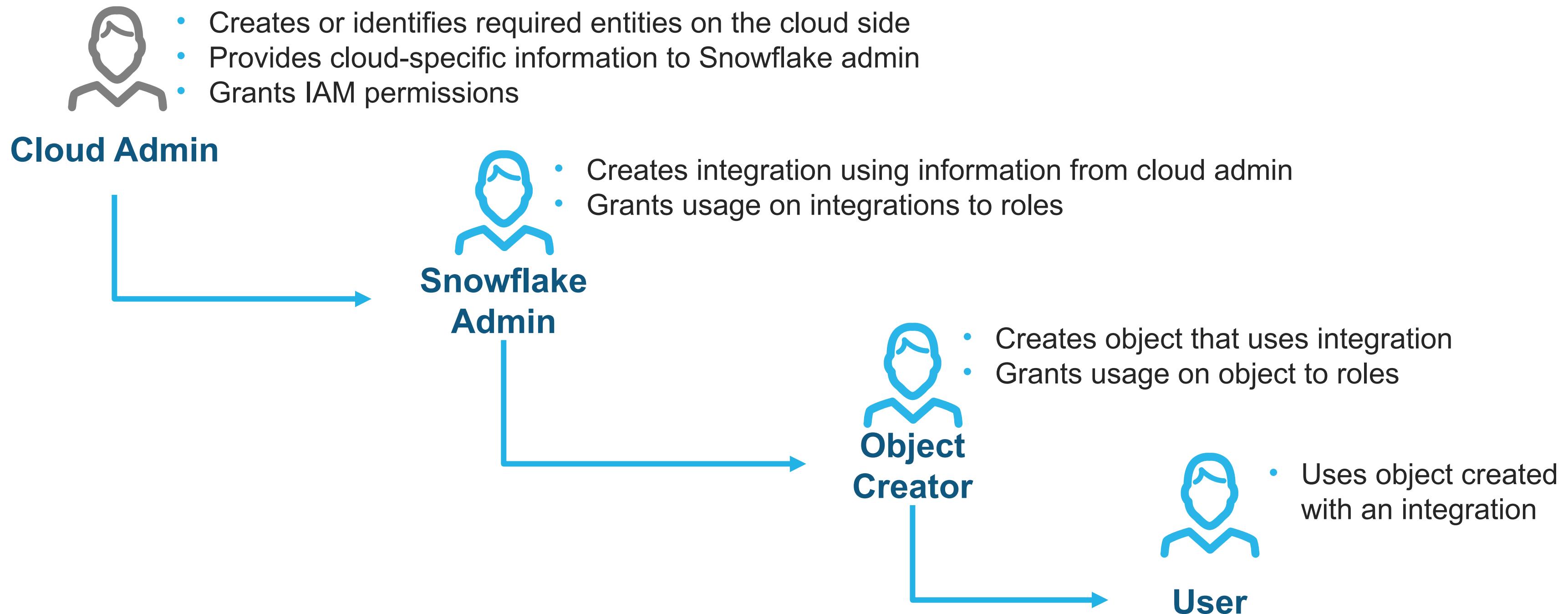


# WHY USE INTEGRATIONS?



- Increase security for client communication
- Limit the passing of credentials
- Limit the number of times a user needs to log in
- Simplify connection to clients

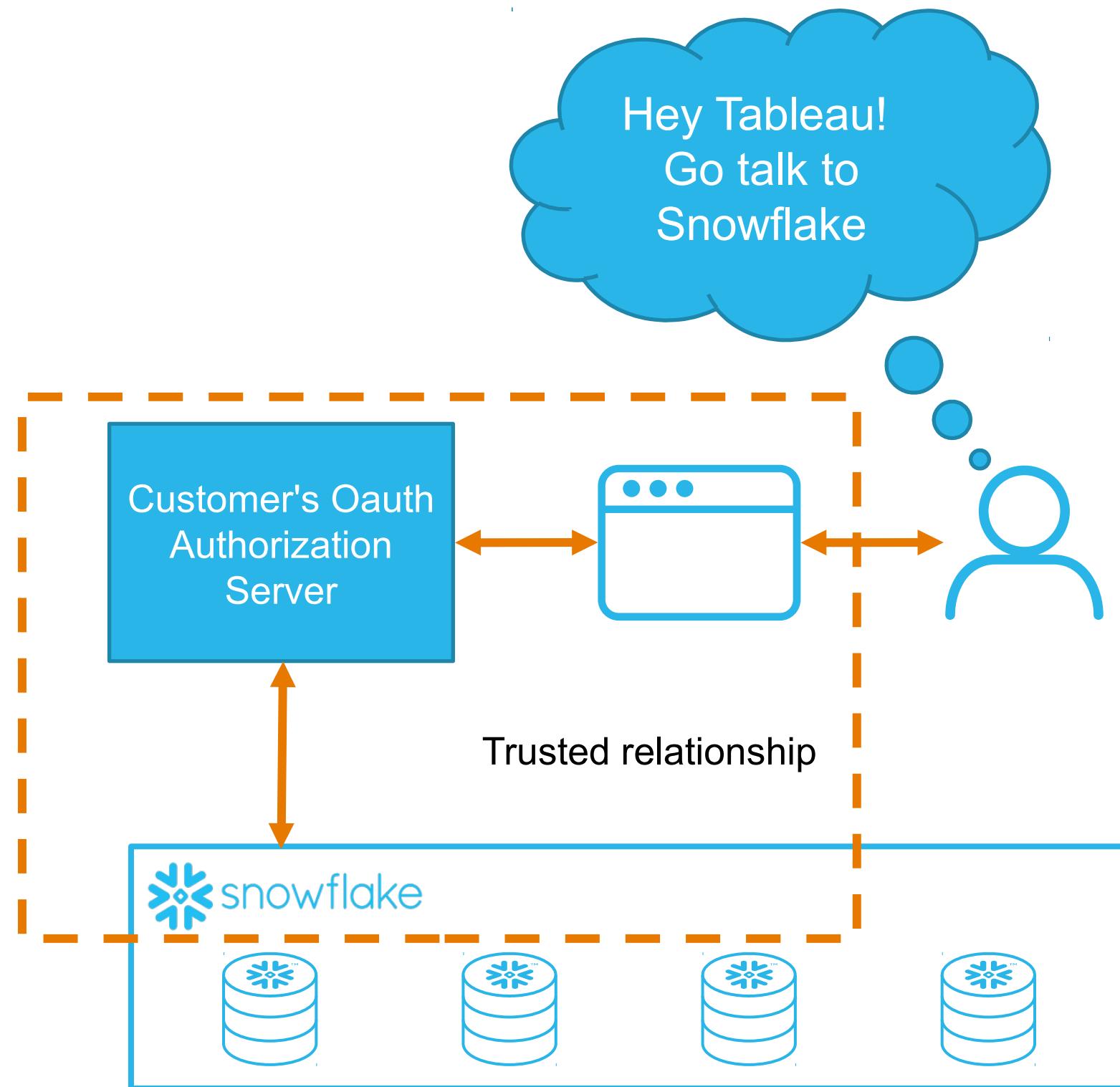
# INTEGRATION WORKFLOW



# SECURITY INTEGRATIONS



# SECURITY INTEGRATIONS



- Provides an interface between Snowflake and third-party services
- Enables Oauth clients to:
  - Redirect to an authorization page
  - Generate (or refresh) access tokens
- Establishes trusted relationship between clients and Snowflake
- Use if you will frequently access Snowflake from client applications

# SECURITY INTEGRATION TYPES

- EXTERNAL\_OAUTH: between client and the customer's authorization server
  - Most common for client connections
- SAML2: between Snowflake and an SSO identity provider (Okta, AD)
- SCIM: between Snowflake and a SCIM client (Okta, AD, custom)
- OAUTH: uses Snowflake as the authorization server
  - Not commonly used



# EXTERNAL OAUTH SYNTAX

- Required parameters:

```
CREATE SECURITY INTEGRATION <name>
```

```
TYPE = EXTERNAL_OAUTH
```

```
EXTERNAL_OAUTH_TYPE = OKTA | AZURE | PING_FEDERATE | CUSTOM
```

```
EXTERNAL_OAUTH_ISSUER = <URL to authorization server>
```

```
EXTERNAL_OAUTH_TOKEN_USER_MAPPING CLAIM = '<string literal>'
```

```
EXTERNAL_OAUTH_SNOWFLAKE_USER_MAPPING_ATTRIBUTE = '<attribute>'
```



# EXTERNAL OAUTH EXAMPLE: OKTA

```
CREATE SECURITY INTEGRATION oauth_okta_integration
TYPE = EXTERNAL_OAUTH
ENABLED = TRUE
EXTERNAL_OAUTH_TYPE = okta
EXTERNAL_OAUTH_ISSUER = <URL to Okta authorization server>
EXTERNAL_OAUTH_TOKEN_USER_MAPPING_CLAIM = 'sub'
EXTERNAL_OAUTH_SNOWFLAKE_USER_MAPPING_ATTRIBUTE = 'login_name'
```



# EXTERNAL OAUTH EXAMPLE: AZURE AD

```
CREATE SECURITY INTEGRATION oauth_azure_integration
TYPE = EXTERNAL_OAUTH
ENABLED = TRUE
EXTERNAL_OAUTH_TYPE = azure
EXTERNAL_OAUTH_ISSUER = '<azure_issuer>'
EXTERNAL_OAUTH_JWS_KEYS_URL = '<azure_jws_key_endpoint>'
EXTERNAL_OAUTH_TOKEN_USER_MAPPING_CLAIM = 'upn'
EXTERNAL_OAUTH_SNOWFLAKE_USER_MAPPING_ATTRIBUTE = 'email_address'
```



# SOME OPTIONAL PARAMETERS

Parameter	Use
EXTERNAL_OAUTH_BLOCK_ROLES_LIST	Specifies a list of roles that the client <b>cannot</b> set as the primary role
EXTERNAL_OAUTH_ALLOWED_ROLES_LIST	Specifies a list of roles that the client <b>can</b> set as the primary role
EXTERNAL_OAUT_ANY_ROLE_MODE	<ul style="list-style-type: none"><li>• DISABLE - OAuth client or user <b>cannot</b> switch roles</li><li>• ENABLED - OAuth client or user <b>can</b> switch roles</li><li>• ENABLE_FOR_PRIVILEGE - OAuth client or user <b>can</b> switch roles only for a client or user with the USE_ANY_ROLE privilege</li></ul>



# KEY-PAIR AUTHENTICATION



# LAB EXERCISE

## Spin up the Jupyter Server

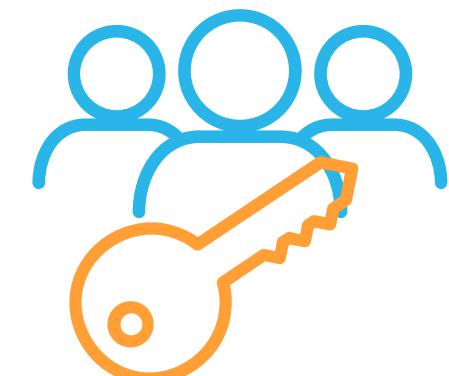
5 minutes

PRE-

- Start and log in to the Jupyter server
- Will be used for the lab at the end of this section



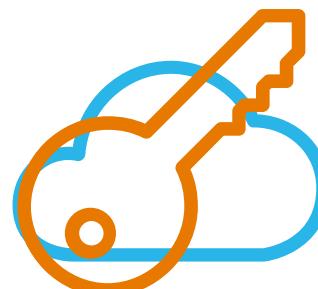
# PRIMARY AUTHENTICATION METHODS



- Username and Password
  - For people



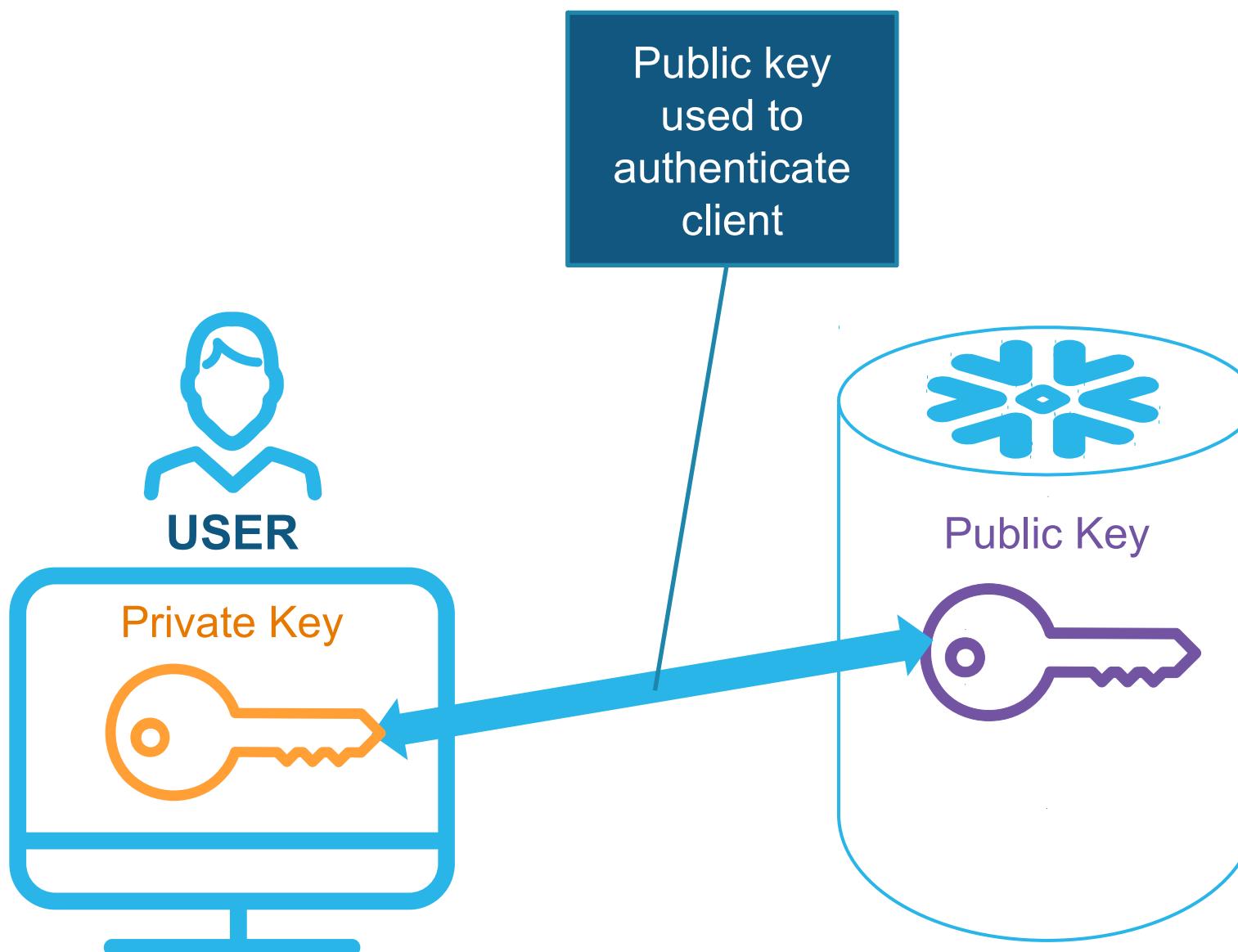
- Multi-factor Authentication
  - For security-minded people



- SSO / Federated Authentication
  - Commonly used by larger companies

# KEY PAIR AUTHENTICATION

## OVERVIEW



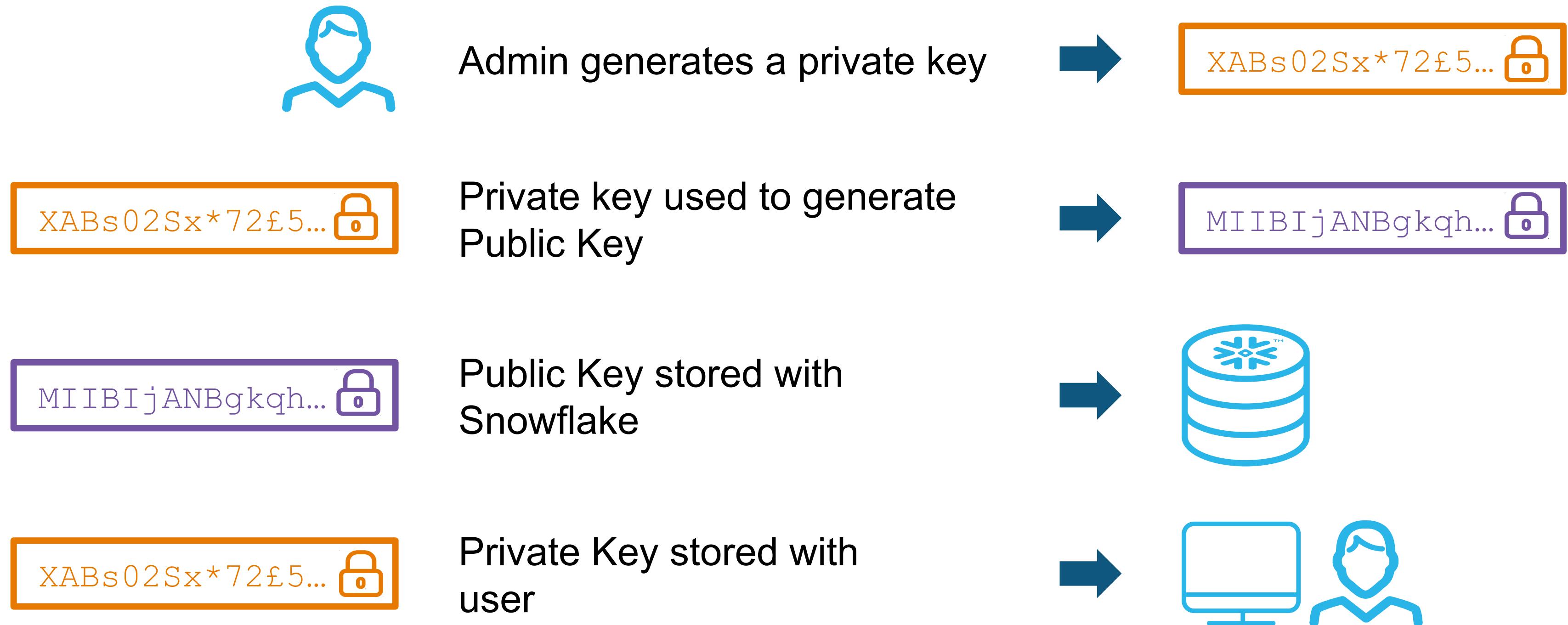
- Used for client connections
- Consists of a pair of keys
  - Public key stored on the application a user wants to access
  - Private key stored on user's system
- Secure connection, easy for user

# SUPPORTED SNOWFLAKE CLIENTS

Client	Key Pair Authentication	Key Pair Rotation	Unencrypted Private Keys
SnowSQL (CLI Client)	✓	✓	
Snowflake Connector for Python	✓	✓	✓
Snowflake Connector for Spark	✓	✓	✓
Snowflake Connector for Kafka	✓	✓	✓
Go driver	✓	✓	
JDBC Driver	✓	✓	✓
ODBC Driver	✓	✓	✓
Node.js Driver	✓	✓	✓
.NET Driver	✓	✓	✓



# GENERATE AN RSA KEY-PAIR: WORKFLOW



# GENERATE AN RSA KEY-PAIR: EXAMPLE

1. Generate a private key:

```
$ openssl genrsa 2048 | openssl pkcs8 -topk8 -inform PEM -out rsa_key.p8
```

2. Generate a public key:

```
$ openssl rsa -in rsa_key.p8 -pubout rsa_key.pub
```

3. Copy keys to a local directory for storage (record the paths to the files)

4. Assign the public key to a Snowflake user (SECURITYADMIN or higher):

```
ALTER USER jsmith SET RSA_PUBLIC_KEY='<public key>'
```



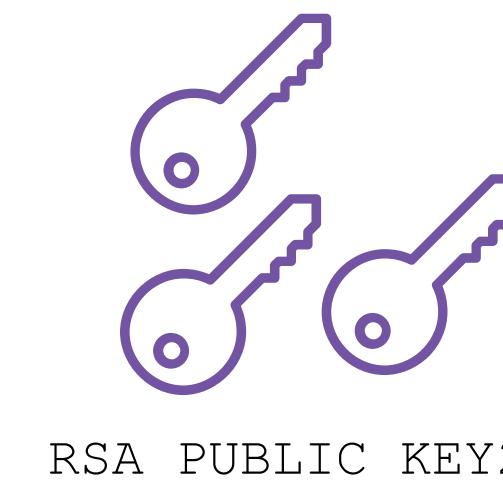
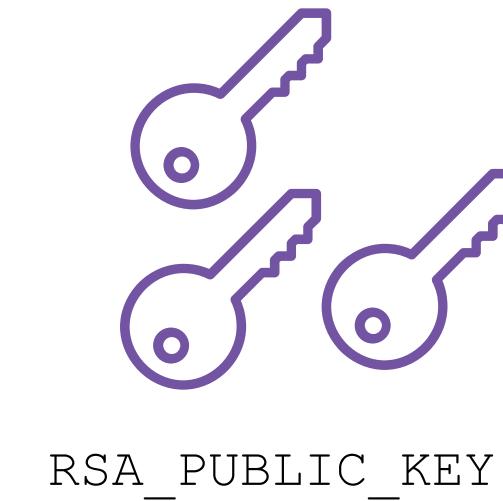
# CONFIGURE THE CLIENT

- Client must be set up to use key-pair authentication
- Example configuration for SnowSQL:
  1. In config file, set PRIVATE\_KEY\_PATH = <path>/rsa\_key.p8
  2. In config file, set the SNOWSQL\_PRIVATE\_KEY\_PASSPHRASE environment variable:  
**Linux/Mac:** export SNOWSQL\_PRIVATE\_KEY\_PASSPHRASE = <passphrase>  
**Windows:** set SNOWSQL\_PRIVATE\_KEY\_PASSPHRASE = '<passphrase>'
  3. Include the private-key-path when connecting to Snowflake:  

```
$ snowsql -a <account> -u <user> --private-key-path <path>/rsa_key.p8
```



# KEY ROTATION



- Rotate public key on a regular basis
  - Protect against key theft
  - Comply with security best practices
- Set up an internal rotation schedule
- Snowflake supports multiple active keys
  - Uninterrupted rotation

# KEY ROTATION WORKFLOW

1. Generate a new private/public key set

2. Assign the new public key to the user

```
ALTER USER jsmith SET RSA_PUBLIC_KEY2='<public key>'
```

3. Update client connections and specify new private key

4. Remove the old public key from the user

```
ALTER USER jsmith UNSET RSA_PUBLIC_KEY
```



# INTEGRATIONS AND KEY PAIRS

## 1. Create an integration

```
CREATE SECURITY INTEGRATION kp_int
  TYPE = OAUTH
  ENABLED = true
  OAUTH_CLIENT = custom
  OAUTH_CLIENT_TYPE = 'confidential'
  OAUTH_REDIRECT_URI = 'https://localhost.com'
  OAUTH_CLIENT_RSA_PUBLIC_KEY = '<public key>'
  OAUTH_CLIENT_RSA_PUBLIC_KEY_2 = '<public key>'
```

## 2. Configure calls to the OAuth endpoints from the Snowflake authorization server

- See documentation for details

## 3. Update public key as needed:

```
ALTER SECURITY INTEGRATION kp_int SET oauth_client_rsa_public_key = '<public key>'
```



# LAB EXERCISE: 2

## Key Pair Authentication

10 minutes

- Generate Public and Private Keys
- Assign Public Key to your User
- Log in to SnowSQL with Key Pair Authentication



# DATA LOADING AND TROUBLESHOOTING



# MODULE AGENDA

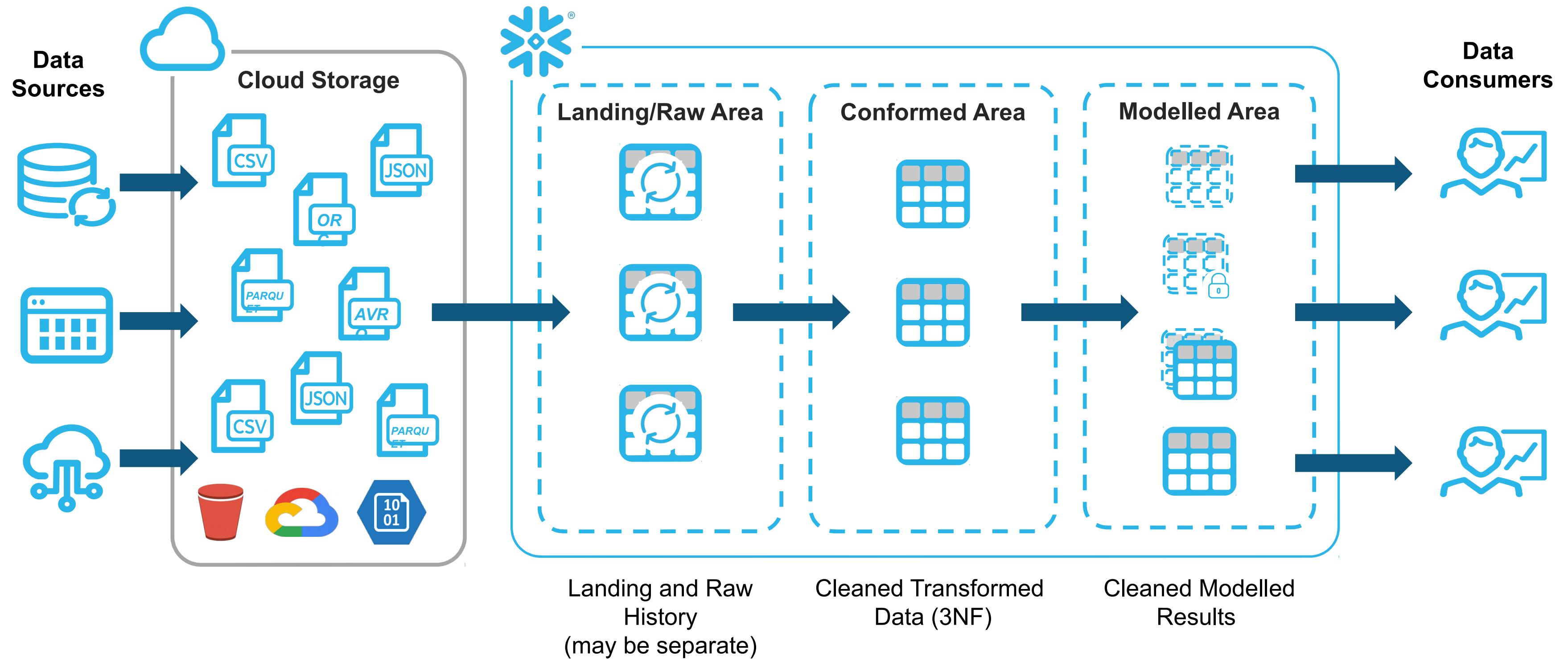
- Data Ingestion Options
- Storage Integrations
- COPY Validation and Error Handling
- Fixing Load Failures
- Continuous Loading with Snowpipe
- Notification Integrations



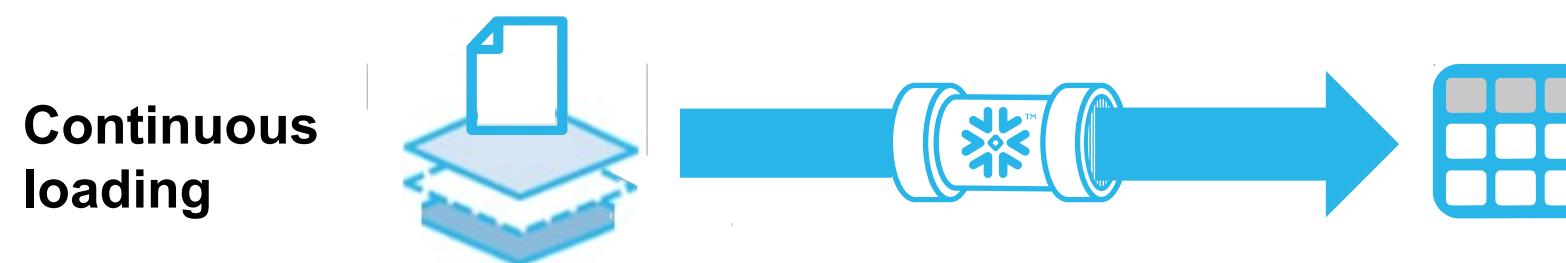
# DATA INGESTION OPTIONS



# LOGICAL DATA ARCHITECTURE



# COMMON INGESTION OPTIONS



- Snowflake COPY command
  - Bulk loading and data migrations
  - Scheduled/orchestrated loads
- Snowpipe
  - Incremental and near real time loading
  - Unscheduled/unpredictable data arrival
- 3rd Party Tools

# BATCH VS. CONTINUOUS

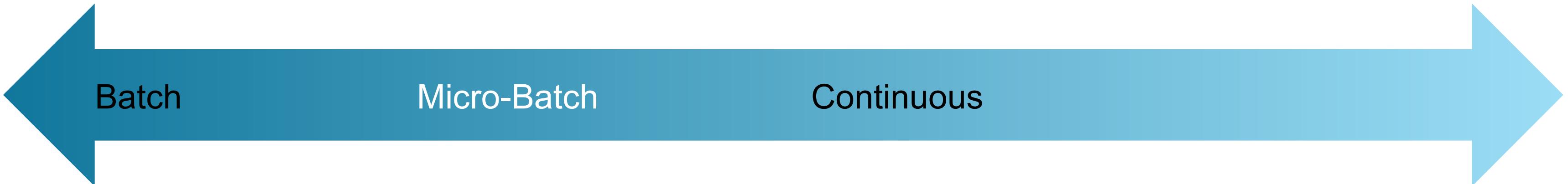
## DIFFERENT REQUIREMENTS

### Batch Loading

- Bulk data delivery
- Scheduled/Predictable
- Latency hours or days

### Continuous Loading

- Incremental data loads
- Erratic/Unpredictable arrival
- Latency in seconds to minutes

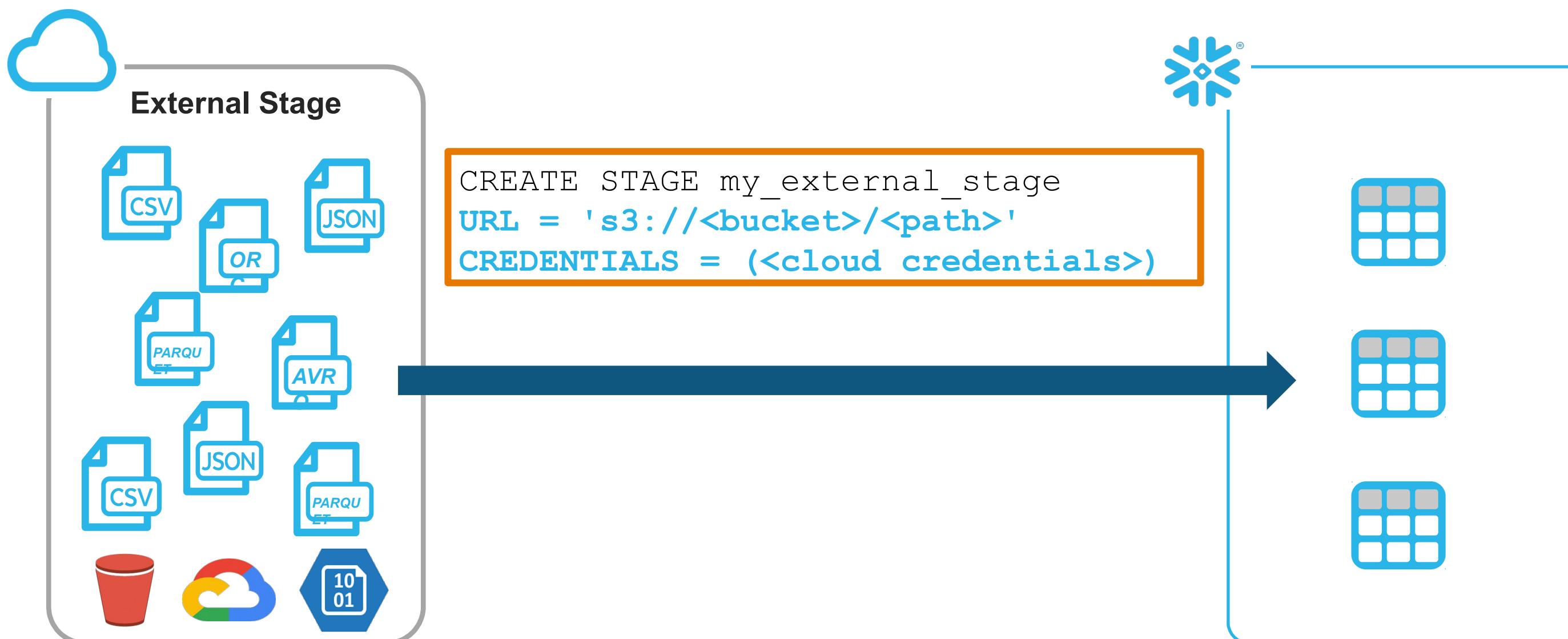


# STORAGE INTEGRATIONS



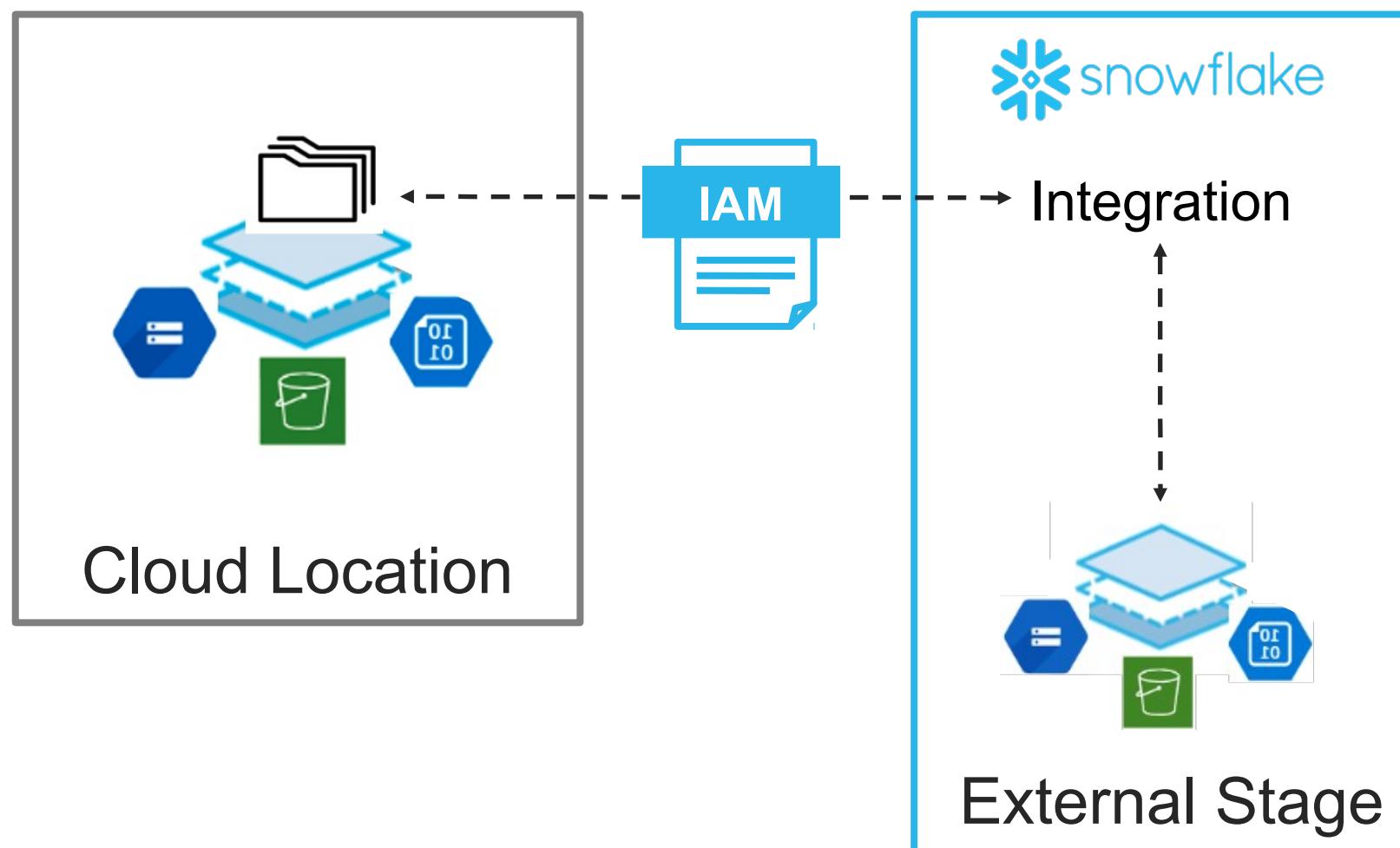
# WHAT IS AN EXTERNAL STAGE?

- A cloud storage location (outside Snowflake) where files are staged before loading
- You can create and access a stage by passing credentials



# WHAT IS A STORAGE INTEGRATION?

TRUSTED CLOUD STORAGE ACES



- More secure method to create an external stage
- Generates and stores an Identity and Access Management (IAM) entity
- Single storage integration can support multiple external stages

# CREATE STORAGE INTEGRATION

- Create the storage integration and give it a name

**CREATE STORAGE INTEGRATION <name>**

TYPE = EXTERNAL\_STAGE

<cloud provider parameters>

ENABLED = TRUE

STORAGE\_ALLOWED\_LOCATIONS = (<list of locations>)

STORAGE\_BLOCKED\_LOCATIONS = (<list of locations>);



# CREATE STORAGE INTEGRATION

- Creates an interface between Snowflake and an external cloud storage location

```
CREATE STORAGE INTEGRATION <name>
```

```
TYPE = EXTERNAL_STAGE
```

```
<cloud provider parameters>
```

```
ENABLED = TRUE
```

```
STORAGE_ALLOWED_LOCATIONS = (<list of locations>)
```

```
STORAGE_BLOCKED_LOCATIONS = (<list of locations>);
```



# CREATE STORAGE INTEGRATION

- Cloud provider-specific parameters required to allow access

```
CREATE STORAGE INTEGRATION <name>
  TYPE = EXTERNAL_STAGE
  <cloud provider parameters>
  ENABLED = TRUE
  STORAGE_ALLOWED_LOCATIONS = (<list of locations>)
  STORAGE_BLOCKED_LOCATIONS = (<list of locations>);
```



# CREATE STORAGE INTEGRATION

- Allows authorized users to create new stages that reference this integration

```
CREATE STORAGE INTEGRATION <name>
  TYPE = EXTERNAL STAGE
  <cloud provider parameters>
ENABLED = TRUE
  STORAGE_ALLOWED_LOCATIONS = (<list of locations>)
  STORAGE_BLOCKED_LOCATIONS = (<list of locations>);
```



# CREATE STORAGE INTEGRATION

- Comma-separated list of cloud storage locations this integration can (or cannot) access

```
CREATE STORAGE INTEGRATION <name>
  TYPE = EXTERNAL STAGE
  <cloud provider parameters>
  ENABLED = TRUE
  STORAGE_ALLOWED_LOCATIONS = (<list of locations>)
  STORAGE_BLOCKED_LOCATIONS = (<list of locations>);
```



# STORAGE INTEGRATION: AWS

```
CREATE STORAGE INTEGRATION s3_integration
    TYPE = EXTERNAL_STAGE
    ENABLED = TRUE
    STORAGE_PROVIDER = S3
    STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::001234567890:role/iamrole'
    STORAGE_ALLOWED_LOCATIONS =
        ('s3://bucket1/path1/', 's3://bucket2/path2/');
```



# STORAGE INTEGRATION: AZURE

```
CREATE STORAGE INTEGRATION azure_integration
    TYPE = EXTERNAL_STAGE
    ENABLED = TRUE
    STORAGE_PROVIDER = azure
    AZURE_TENANT_ID = '<tenant_id>'
    STORAGE_ALLOWED_LOCATIONS =
        ('azure://myaccount.blob.core.windows.net/mycontainer.path1/' ,
         'azure://myaccount.blob.core.windows.net/mycontainer.path2/');
```



# STORAGE INTEGRATION: GCS

```
CREATE STORAGE INTEGRATION s3_integration
    TYPE = EXTERNAL_STAGE
    ENABLED = TRUE
    STORAGE_PROVIDER = gcs
    STORAGE_ALLOWED_LOCATIONS = ('*')
    STORAGE_BLOCKED_LOCATIONS = ('gcs://bucket3/path3/');
```



# ENFORCE STORAGE INTEGRATION USAGE

## ACCOUNT-LEVEL PARAMETERS

- Can be set only at the account level, by a role with appropriate privileges

Parameter	Notes
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_CREATION	Allow only storage integration-based stages to be created
REQUIRE_STORAGE_INTEGRATION_FOR_STAGE_OPERATION	Allow only storage integration-based stages to be used

- Can be restricted at the user level
  - Most block access at the account level, and enable access for individual users

Parameter	Notes
PREVENT_UNLOAD_TO_INLINE_URL	Restrict data unload to inline URLs (can also be set at the user level)



# COPY VALIDATION AND ERROR HANDLING



# COPY FAILURES



Load failures are typically due to:

- File and table column order mismatch
  - CSV format
- Column data type or size mismatch
  - 'longmont' > VARCHAR (4)
  - 'fer sure' > BOOLEAN
- Incorrect file format specified
- Corrupt data files

# REVIEW BEFORE LOADING

## STEP 1: QUERY FILE CONTENT

- Query a data file directly from cloud storage
  - Explore content
  - Confirm data types

The screenshot shows the Snowflake interface with a query results page. At the top, there is a code editor with the following SQL query:

```
5 | SELECT TOP 100 $1, $2, $3, $4, $5, $6 FROM @class_stage;
```

Below the code editor, there are four tabs: "Objects", "Query" (which is selected), "Results" (which is active), and "Chart".

The results table has the following data:

	\$1	\$2	\$3	\$4	\$5	\$6
1	10332	Lori	Bridges	22	714.04	null
2	18972	Otto	Cox	4	9092.12	null
3	87683	Sierra	Morales	20	8612.53	null

# PREPARE TO LOAD

## STEP 2: CREATE TABLE

```
CREATE TRANSIENT TABLE members (
    id      INT,
    fname   VARCHAR(10),
    lname   VARCHAR(10),
    nation  INT,
    bal     NUMBER(10,2));
```

- Create table based on exploratory query
- Use a temporary or transient table
  - Avoid 7 days of failsafe while testing



# OPTIONAL: VALIDATE BEFORE LOADING

## STEP 3: LOAD USING VALIDATION\_MODE

```
7   COPY INTO members
8     FROM @my_stage
9     VALIDATION_MODE = RETURN_100_ROWS
10
```

Objects    Query    Results    Chart

	ERROR	...	FILE	LINE
1	User character length limit (10) exceeded by string 'Maximiliano'		jja93497/data_0_0_0.csv.gz	21
2	User character length limit (10) exceeded by string 'Maximiliano'		jja93497/data_0_0_0.csv.gz	23
3	User character length limit (10) exceeded by string 'Labindalawa'		jja93497/data_0_0_0.csv.gz	100

### VALIDATION\_MODE:

- Does not load any data
- Does not support transformations



# ERROR HANDLING OPTIONS DURING LOAD

Mode	Action	Notes
ABORT_STATEMENT	Any error causes the entire load to fail. Any data that was loaded will be rolled back	Default method; all or nothing
CONTINUE	Loads all valid records, invalid records are skipped.	May result in partially-loaded files
SKIP_FILE	File is skipped if any error is encountered. Good records loaded <b>from that file</b> are rolled back.	File buffered while loading – entire file is either loaded, nor not loaded (slower than ABORT_STATEMENT or CONTINUE)
SKIP_FILE_<num>	File is skipped if <num> errors are encountered. If fewer than <num> errors, loaded records remain.	May result in partially-loaded files
SKIP_FILE_<pct>%	File is skipped if <pct>% of the records processed so far have errors. If fewer than <num>% errors, loaded records remain.	May result in partially-loaded files



# FIXING LOAD FAILURES



# DIAGNOSE FAILURES AFTER LOAD

1. Load with an error mode that will continue loading even with errors

```
COPY INTO members  
FROM @my_stage  
ON_ERROR = CONTINUE;
```

file	status	rows_parsed	rows_loaded	error_limit	errors_seen
s3stage/Pla	PARTIALLY_LOADED	86	83	86	3

first_error	first_error_line	first_error_character	first_error_column_name
Numeric value 'Num_Revie	1	48	"MEMBERS" ["NUM_REVIE



# DIAGNOSE FAILURES AFTER LOAD

## 2. Use the VALIDATE table function to see all the errors

```
SELECT * FROM TABLE (VALIDATE (members, JOB_ID => '_last'));
```

Shortcut to find last COPY INTO for given table in current session – can also specify Query ID

	ERROR	FILE	LINE	REJECTED_RECORD
1	User character length limit (10) exce	jja93497/data_0_0_0.csv.g:	21	Hummelo Betony,Betony,Early Sur
2	User character length limit (10) exce	jja93497/data_0_0_0.csv.g:	23	Name,Type,Start,End,Light,Color1
3	User character length limit (10) exce	jja93497/data_0_0_0.csv.g:	100	Name,Type,Start,End,Light,Color1



# DIAGNOSE FAILURES AFTER LOAD

## 3. Capture rejected records to fix

Option: Create table of rejected records, update, insert to target table

```
CREATE TABLE records_to_fix (rec VARCHAR)
AS (SELECT rejected_record FROM
    TABLE (VALIDATE (members, JOB_ID=> '_last')));
```

Option: Download rejected records, fix, re-load to target table

```
COPY INTO @my_stage
AS (SELECT rejected_record FROM
    TABLE (VALIDATE (members, JOB_ID=> '_last')));
```



# WORKFLOW SUMMARY

1. Run COPY INTO
2. Capture the query ID
  - Can set variable immediately after load – `SET qid = LAST_QUERY_ID();`
3. Run the VALIDATE table function using the table name and query ID
  - Can use '\_last' as shortcut to query ID of last COPY INTO on that table
4. Capture the rejected records in a file or table
5. Fix the errors
6. Insert or COPY the corrected records into the target table



# LAB EXERCISE: 3

## Loading, Transforming and Validating Data

45 minutes

- Loading Structured Data
- Loading Data and File Sizes
- Loading Semi-Structured Data
- Loading Fixed Format Data
- Detect File Format Problems with VALIDATION\_MODE
- Detect Load Failures with ON\_ERROR set to CONTINUE
- Diagnose Load Errors

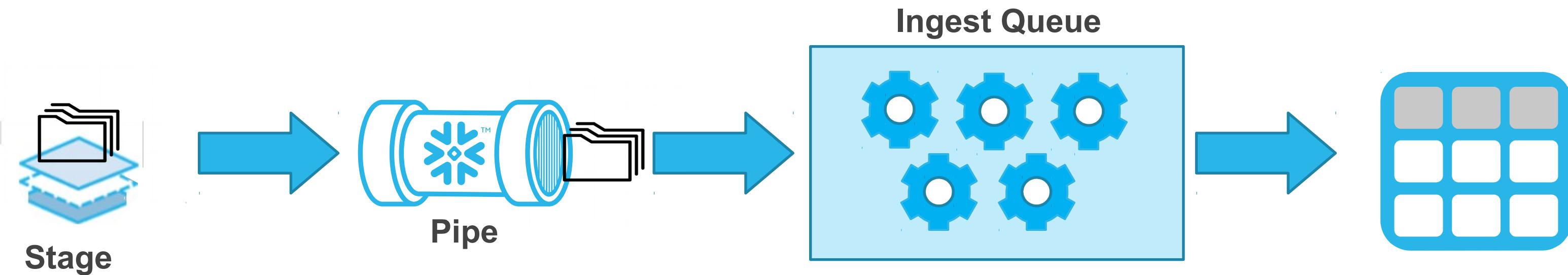


# CONTINUOUS LOADING WITH SNOWPIPE



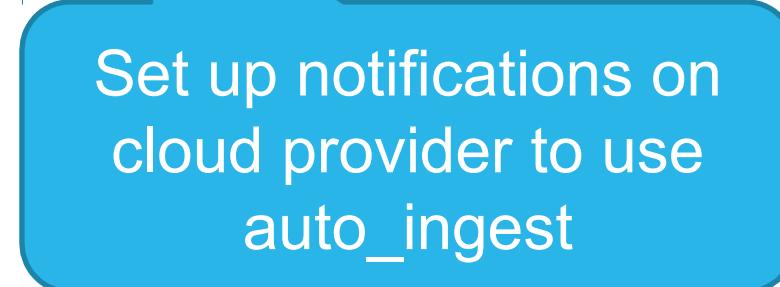
# SNOWPIPE

- For continuous data ingestion (typically < 60 seconds for 100 – 250 MB file)
- Uses Snowflake serverless compute resources (no virtual warehouse needed)



# WHAT IS A PIPE?

```
CREATE PIPE sales_pipe AS  
COPY INTO sales  
FROM @sales_stage  
FILE_FORMAT = csv_format  
AUTO_INGEST = true;
```



Set up notifications on  
cloud provider to use  
auto\_ingest

- Named object that holds the definition of a COPY INTO command
- Enables automatic data loading on arrival
- Latency ~1 minute
- Triggered using:
  - Auto Ingest – cloud notification
  - REST API – API passes file name(s)
  - Manually – ALTER PIPE REFRESH
  - Not intended for regular use!

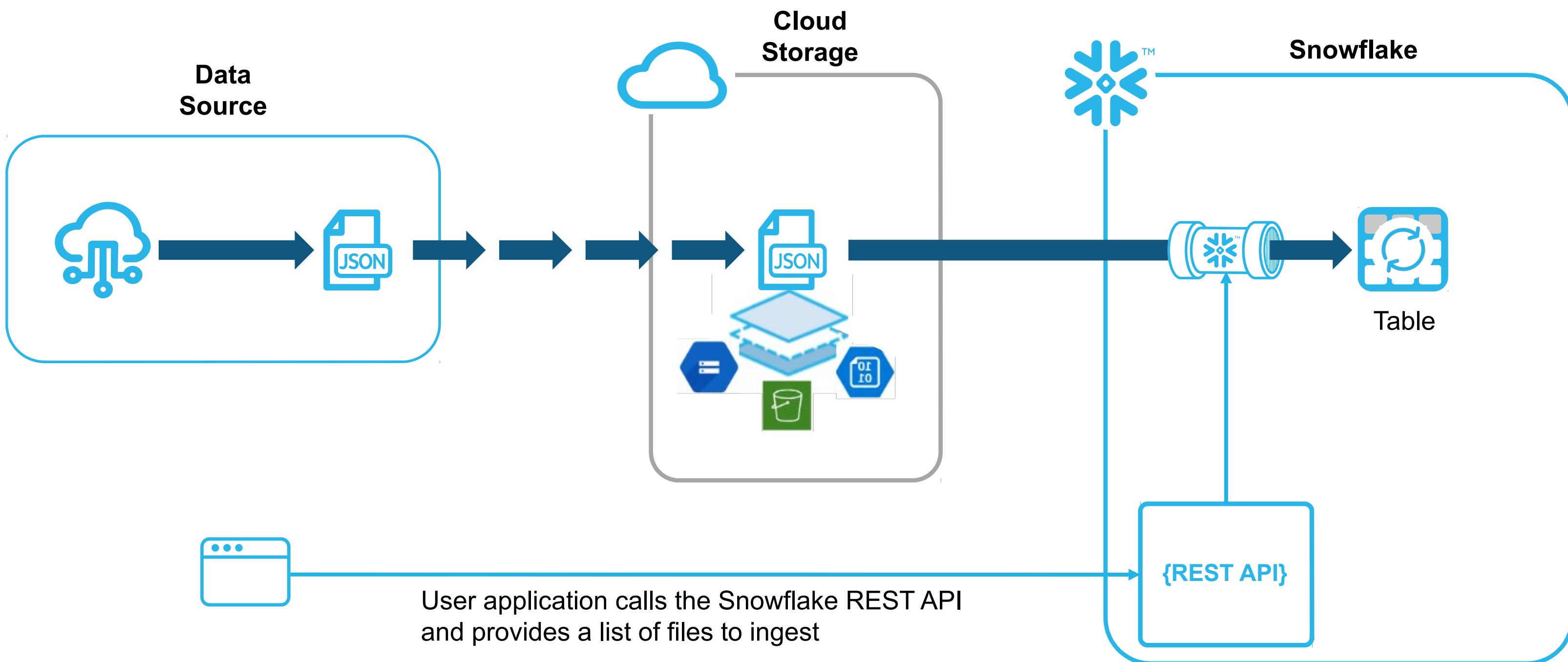


# COPY INTO VS SNOWPIPE

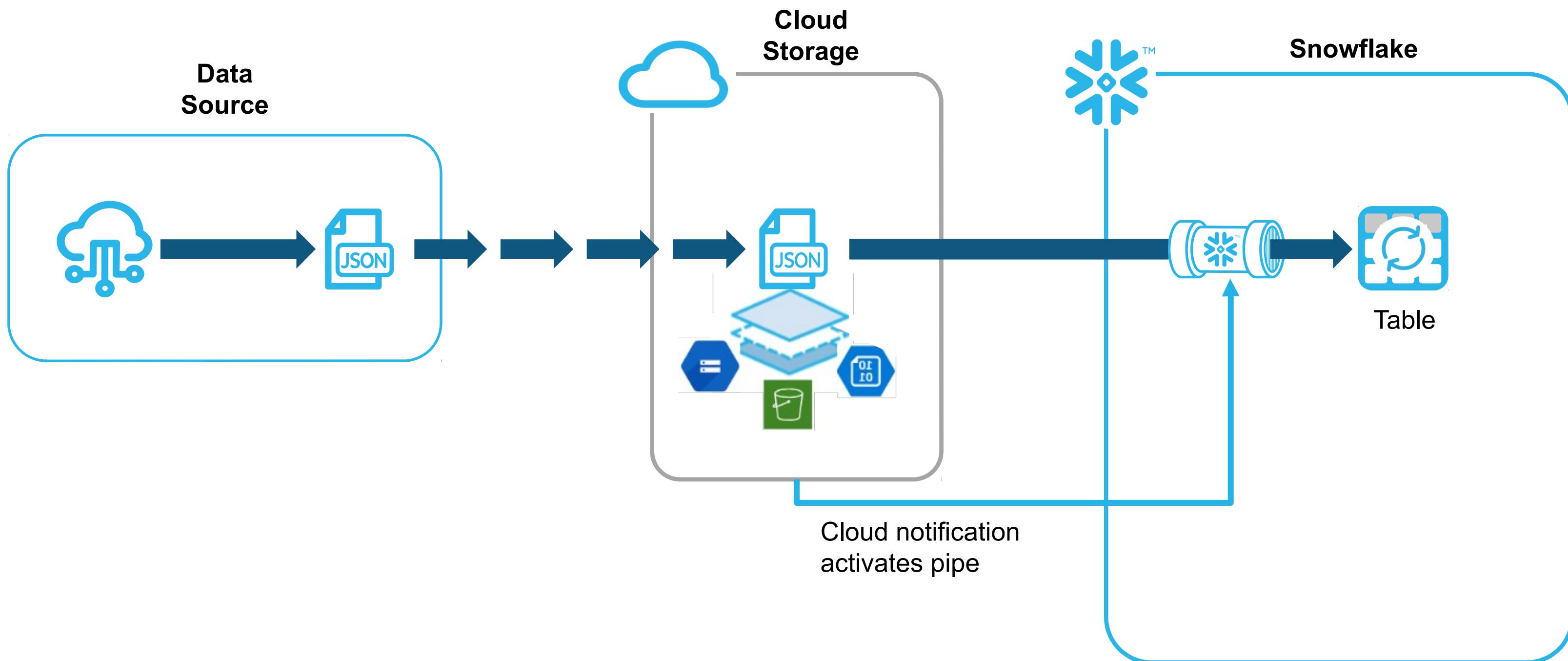
Option	COPY INTO	SNOWPIPE
Default error handling method	ABORT _STATEMENT	SKIP _FILE
PURGE option of COPY INTO	Supported	Not supported
Authentication	Relies on client security options	With REST API, requires key pair authentication
Load History	Stored in the metadata of the target table for 64 days	Stored in the metadata of the pipe for 14 days
Transactions	Loads performed in a single transaction	Loads split into single or multiple transactions based on the number and size of rows in each data file.
Compute resources	Uses a virtual warehouse	Uses "serverless" compute



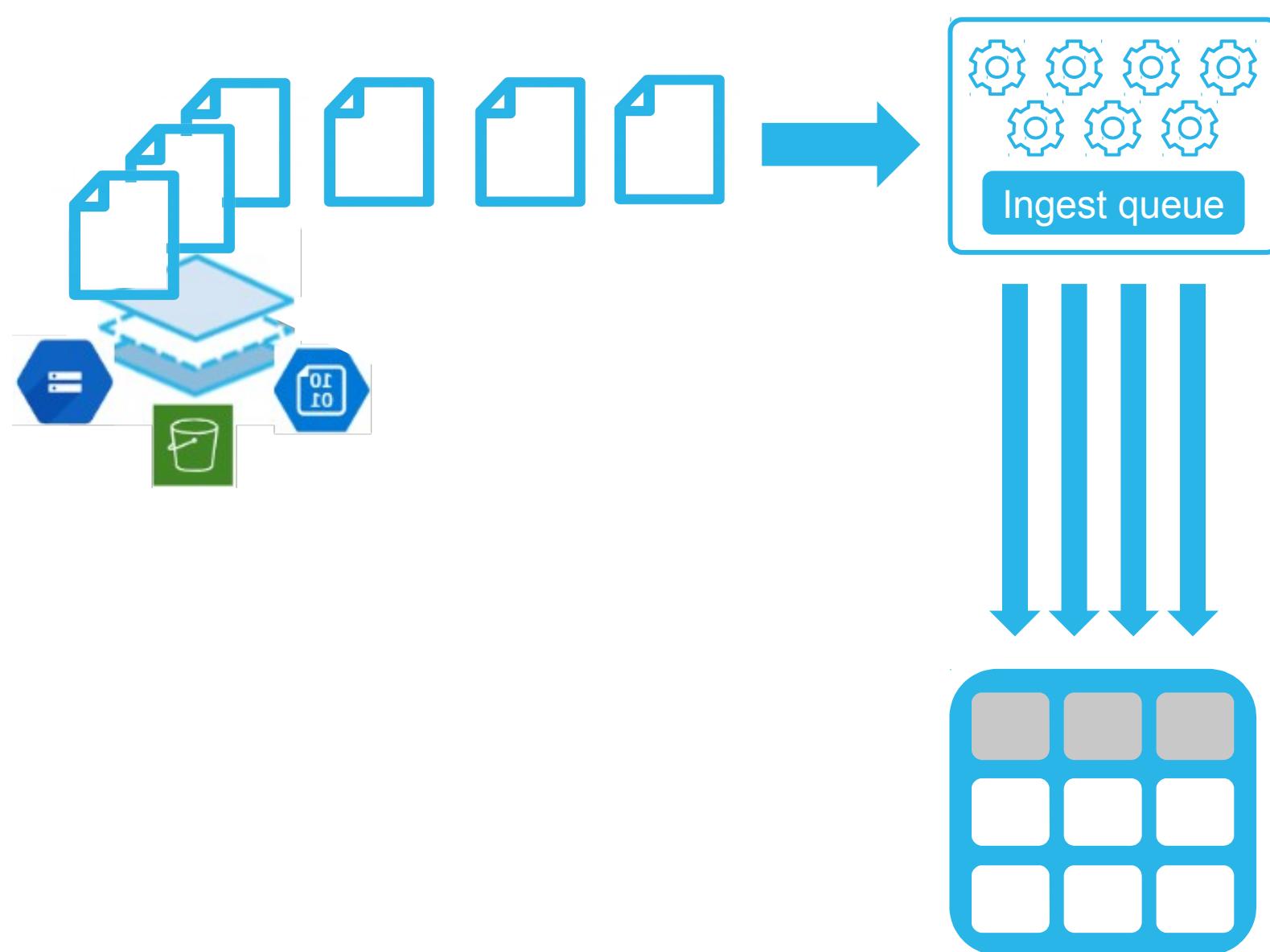
# INGEST USING REST API



# AUTO INGEST

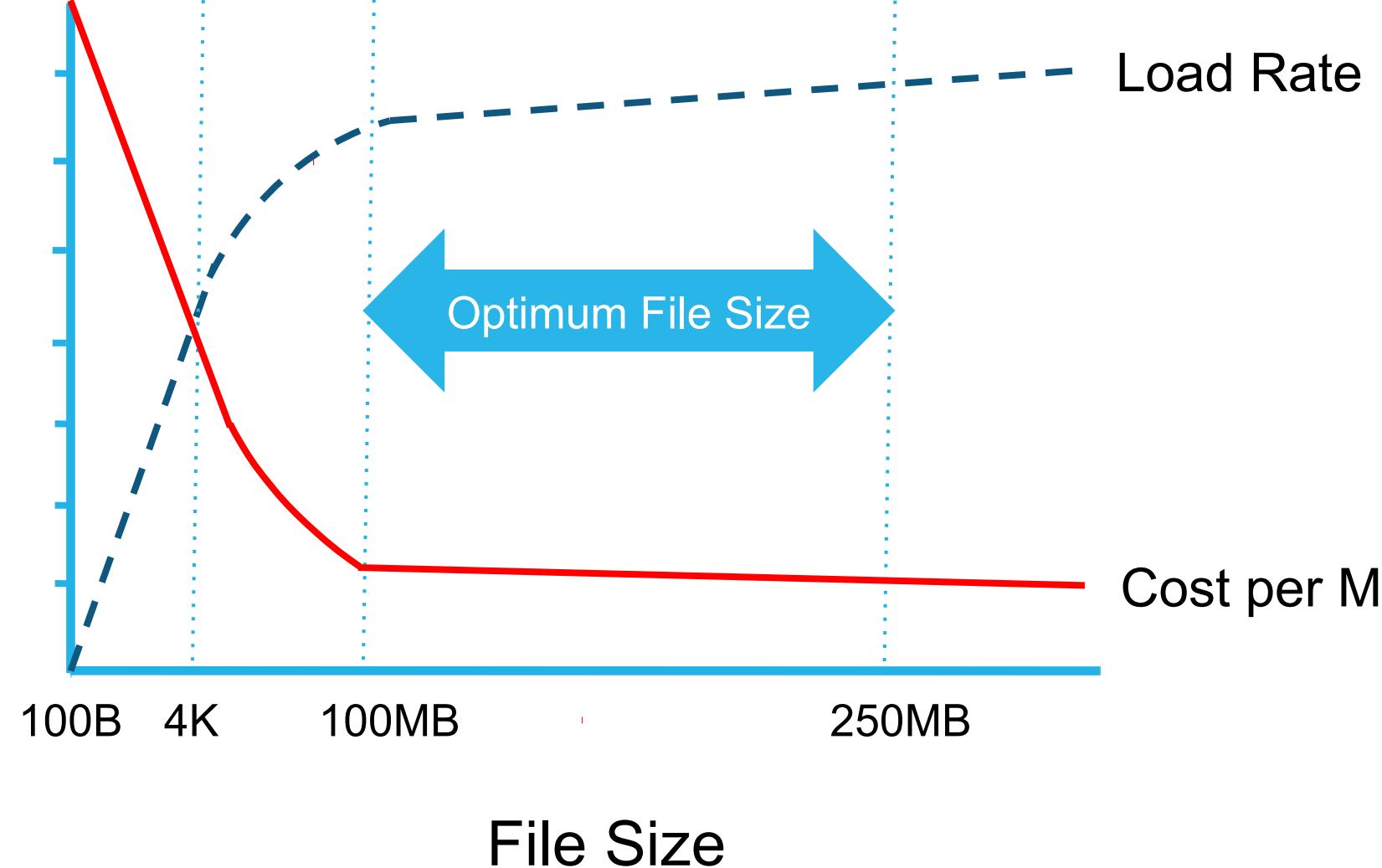


# FILE LOAD ORDER



- File names are moved into ingest queue
- Names of files that appear in the stage later are appended to the queue
- Multiple threads pull from the queue
- Older files are generally loaded first, but that is not guaranteed

# THROUGHPUT WITH TINY FILES



## Challenge

- Each file incurs some overhead
- Millions of tiny files = slow load rate and high compute cost
- Sweet spot is 100-250MB per file

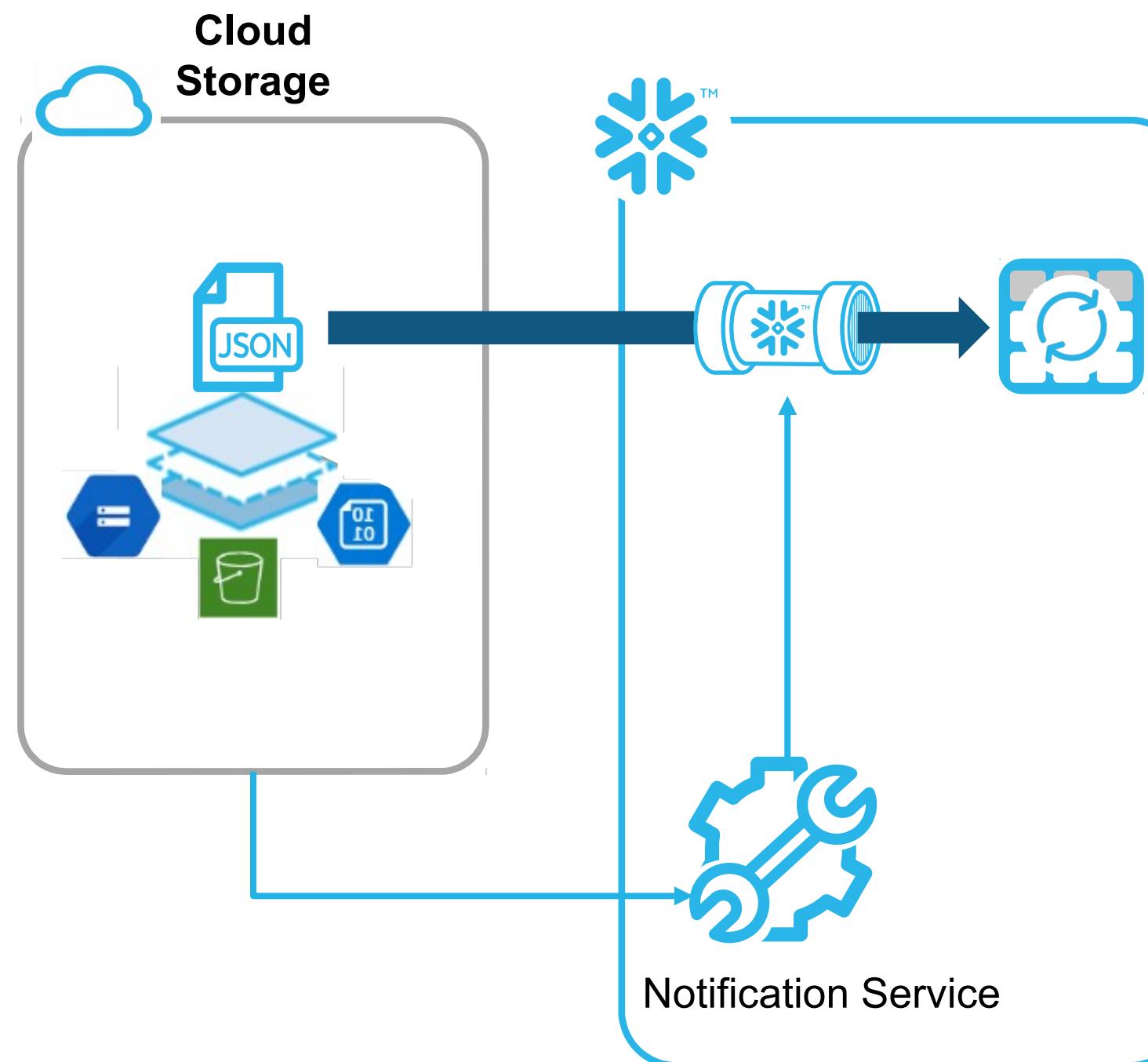
## Recommendation

- Stage files ~ once/minute
- With streaming loads (thousands/hour), trigger loads every 60 seconds or 100 MB
- Even increasing file size from 100 bytes to 4K will show a cost/speed improvement

# NOTIFICATION INTEGRATIONS



# NOTIFICATION INTEGRATION



- Interface between Snowflake and cloud message queues
- Use for:
  - Snowpipe data loading
  - Updating external tables
  - Updating directory tables
  - Snowpipe error notifications (AWS)

# SAMPLE ON AWS

```
CREATE NOTIFICATION INTEGRATION <name>
ENABLED = TRUE
TYPE = QUEUE
```

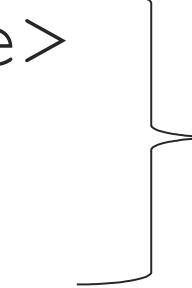
Same for all providers

```
NOTIFICATION_PROVIDER = AWS_SNS
AWS_SNS_TOPIC_ARN = '<topic_arn>'
AWS_SNS_ROLE_ARN - '<iam_role_arn>';
```



# SAMPLE ON GOOGLE

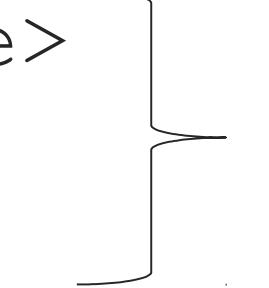
```
CREATE NOTIFICATION INTEGRATION <name>
ENABLED = TRUE
TYPE = QUEUE
NOTIFICATION_PROVIDER = GCP_PUBSUB
GCP_PUBSUB_SUBSCRIPTION_NAME = '<subscription_id>';


Same for all providers
```



# SAMPLE ON AZURE

```
CREATE NOTIFICATION INTEGRATION <name>
ENABLED = TRUE
TYPE = QUEUE
NOTIFICATION_PROVIDER = AZURE_STORAGE_QUEUE
AZURE_STORAGE_QUEUE_PRIMARY_URI = '<queue_URL>'
AZURE_TENANT_ID = '<tenant_id>';


    Same for all providers
```



# LAB EXERCISE: 4

## Using Snowflake Pipes

15 minutes

In this lab you will create a basic pipe, put data into the stage the pipe is connected to, and then manually refresh the pipe and make sure the file ingested.

### Learning Objectives:

- Create a pipe
- Verify ingestion



# WORKING WITH SEMI-STRUCTURED DATA



# MODULE AGENDA

- Structured vs. Semi-Structured Data
- Loading and Transforming Semi-Structured Data



# STRUCTURED VS SEMI-STRUCTURED



# STRUCTURED DATA

- Structured data is stored in pre-defined, fixed-format columns

The screenshot shows a database interface with the following components:

- Code Area:** Displays two lines of SQL code:

```
5   SELECT firstname, lastname, points_balance, city, state, zip
6   FROM members LIMIT 10;
```
- Tool Buttons:** A row of buttons with icons and labels: Objects (blue), Query (blue, selected), Results (white), and Chart (light gray).
- Results Table:** A grid showing the query results with the following data:

	FIRSTNAME	LASTNAME	POINTS_BALANCE	CITY	STATE	ZIP
1	Jed	Choules	2,819,999	Evansville	Indiana	47747
2	Sunny	Mammatt	3,230,571	San Jose	California	95138
3	Orelee	Aaronsohn	7,475,443	Knoxville	Tennessee	37914
4	Dew	Grewer	5,235,957	Humble	Texas	77346
5	Jessey	Cotherill	806.553	Cincinnati	Ohio	45238



# SEMI-STRUCTURED DATA

- Semi-structured data is self-describing data stored in a flexible schema
  - Example: JSON stored as key-value pairs

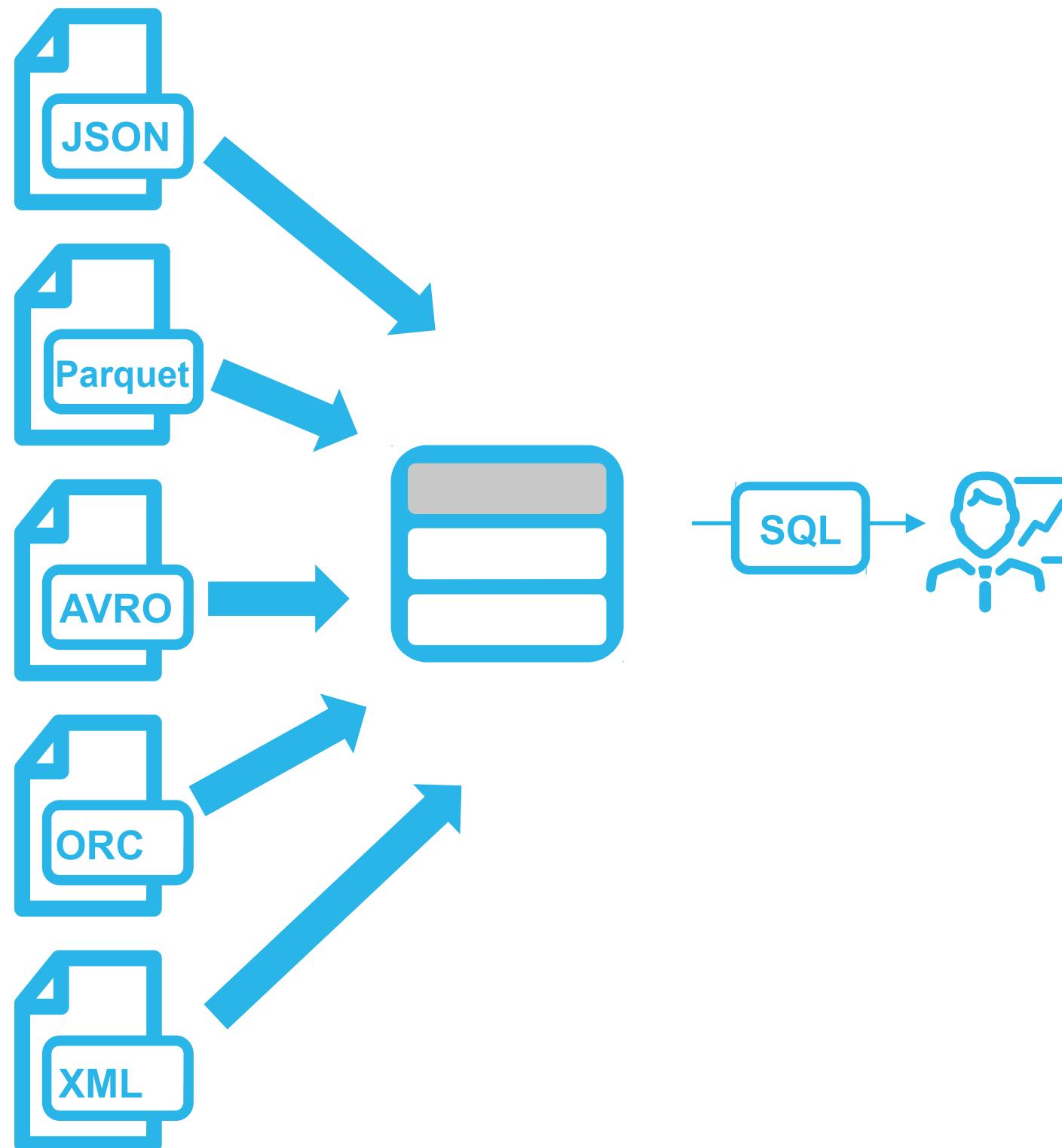
```
6 | SELECT v as VARIANT_RECORD
7 | FROM daily_14_total LIMIT 10;
```

Objects    Query    Results    Chart

	VARIANT_RECORD
1	{ "city": { "coord": { "lat": 11.16667, "lon": 77.616669 }, "country": "IN", "id": 1274422, "name": "Chennimalai" }, "data": [ { "category": "A", "value": 100 } ] }
2	{ "city": { "coord": { "lat": 9.33333, "lon": 76.633331 }, "country": "IN", "id": 1274428, "name": "Chengannur" }, "data": [ { "category": "B", "value": 150 } ] }
3	{ "city": { "coord": { "lat": 12.3, "lon": 78.800003 }, "country": "IN", "id": 1274429, "name": "Chengam" }, "data": [ { "category": "C", "value": 200 } ] }
4	{ "city": { "coord": { "lat": 12.7, "lon": 79.98333 }, "country": "IN", "id": 1274430, "name": "Chengalpattu" }, "data": [ { "category": "D", "value": 250 } ] }
5	{ "city": { "coord": { "lat": 10.53333, "lon": 76.050003 }, "country": "IN", "id": 1274468, "name": "Chavakkad" }, "data": [ { "category": "E", "value": 300 } ] }



# VARIANT DATA TYPE



## VARIANT – Data Type

- JSON, AVRO, Parquet, ORC or XML
- Automatic partition elimination on SELECT
  - Some performance impact
- DATE and TIMESTAMP stored as text

## Data Structures

- OBJECT – collection of key-value pairs
- ARRAY – collection of values

# JSON DATA SAMPLE

Key	Value
"observations": [	
{	
"air": {	
"dew-point": 15.7,	
"dew-point-quality-code": "1",	
<span style="border: 1px solid orange; padding: 2px;">"temp": 18.1,</span>	
"temp-quality-code": "1"	
},	
"atmospheric": {	
"pressure": 99999,	
"pressure-quality-code": "9"	
},	
"dt": "1984-04-11T00:00:00",	
"sky": {	
"ceiling": 9000,	
"ceiling-quality-code": "1"	
},	
"visibility": {	

- Click on a semi-structured record ("row") to see its structure



# JSON DATA SAMPLE

The diagram illustrates a JSON data sample with two labels: 'Key' and 'Value'. The 'Key' label is positioned on the left side of the JSON structure, and the 'Value' label is positioned on the right side. A blue line connects the 'Key' label to the key 'air' in the JSON data. Another blue line connects the 'Value' label to the object value under 'air', which is enclosed in curly braces. The JSON data itself is a semi-structured record with various keys and values.

```
{  
    "observations": [  
        {  
            "air": {  
                "dew-point": 15.7,  
                "dew-point-quality-code": "1",  
                "temp": 18.1,  
                "temp-quality-code": "1"  
            },  
            "atmospheric": {  
                "pressure": 99999,  
                "pressure-quality-code": "9"  
            },  
            "dt": "1984-04-11T00:00:00",  
            "sky": {  
                "ceiling": 9000,  
                "ceiling-quality-code": "1"  
            },  
            "visibility": 10  
        }  
    ]  
}
```

- Click on a semi-structured record ("row") to see its structure
- Values that are objects are enclosed in curly braces



# JSON DATA SAMPLE

The diagram shows a JSON object with a legend. A blue box labeled "Key" points to the key "observations". A blue box labeled "Value" points to the value "[", which is the start of an array. The JSON data is as follows:

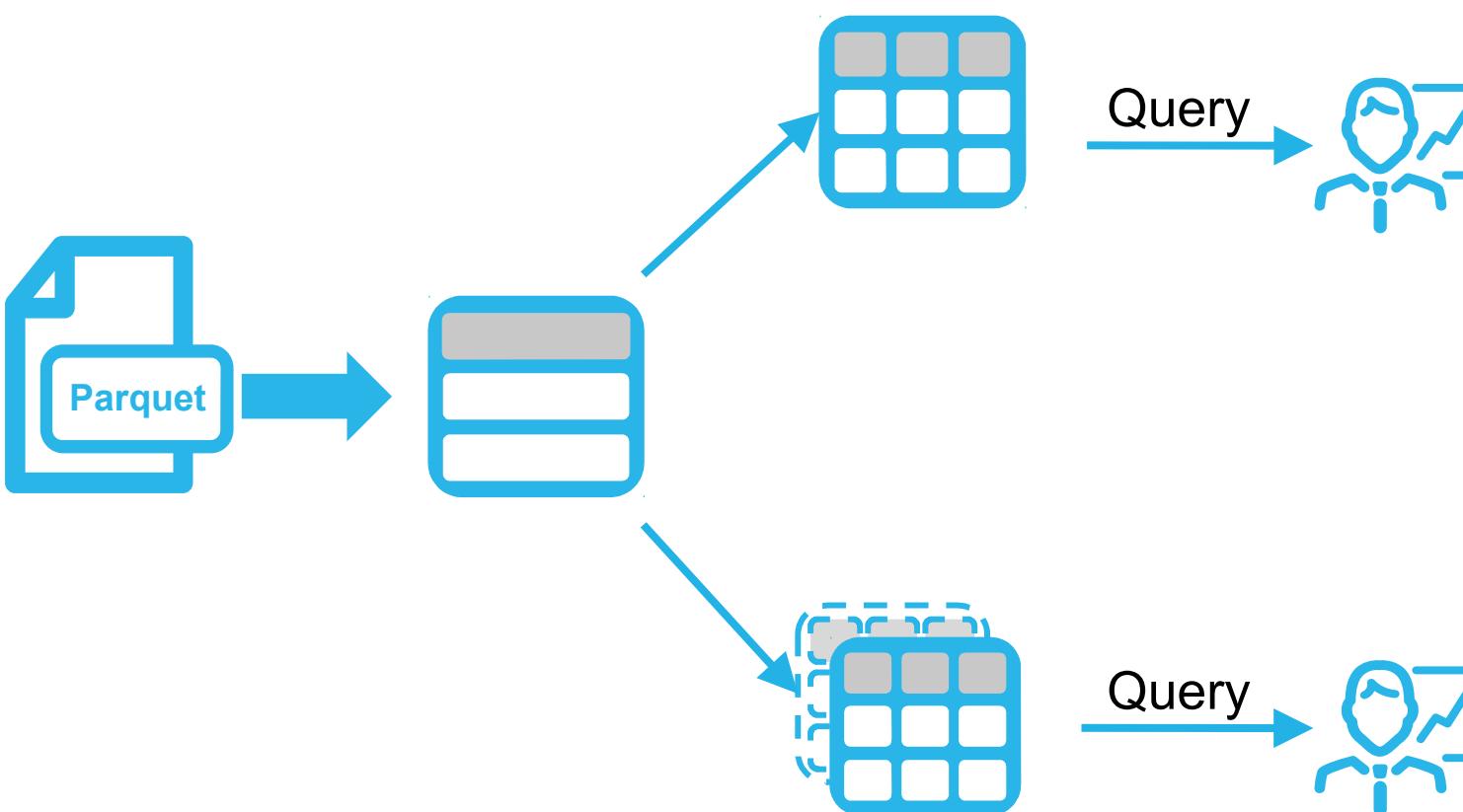
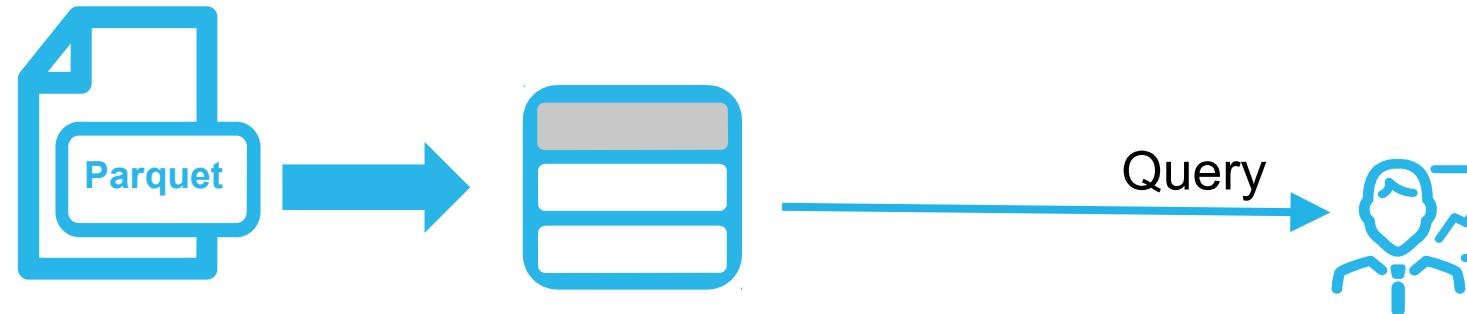
```
{  
    "observations": [  
        {"air": {  
            "dew-point": 15.7,  
            "dew-point-quality-code": "1",  
            "temp": 18.1,  
            "temp-quality-code": "1"  
        },  
        {"atmospheric": {  
            "pressure": 99999,  
            "pressure-quality-code": "9"  
        },  
        {"dt": "1984-04-11T00:00:00",  
        "sky": {  
            "ceiling": 9000,  
            "ceiling-quality-code": "1"  
        },  
        "visibility": 10  
    ]  
}
```

- Click on a semi-structured record ("row") to see its structure
- Values that are objects are enclosed in curly braces
- Values that are arrays are enclosed in square brackets
  - End of value array not shown here...
- Objects and arrays can be nested inside each other
  - Ultimately, broken down into nothing but keys and values

# **LOADING AND TRANSFORMING SEMI-STRUCTURED DATA**

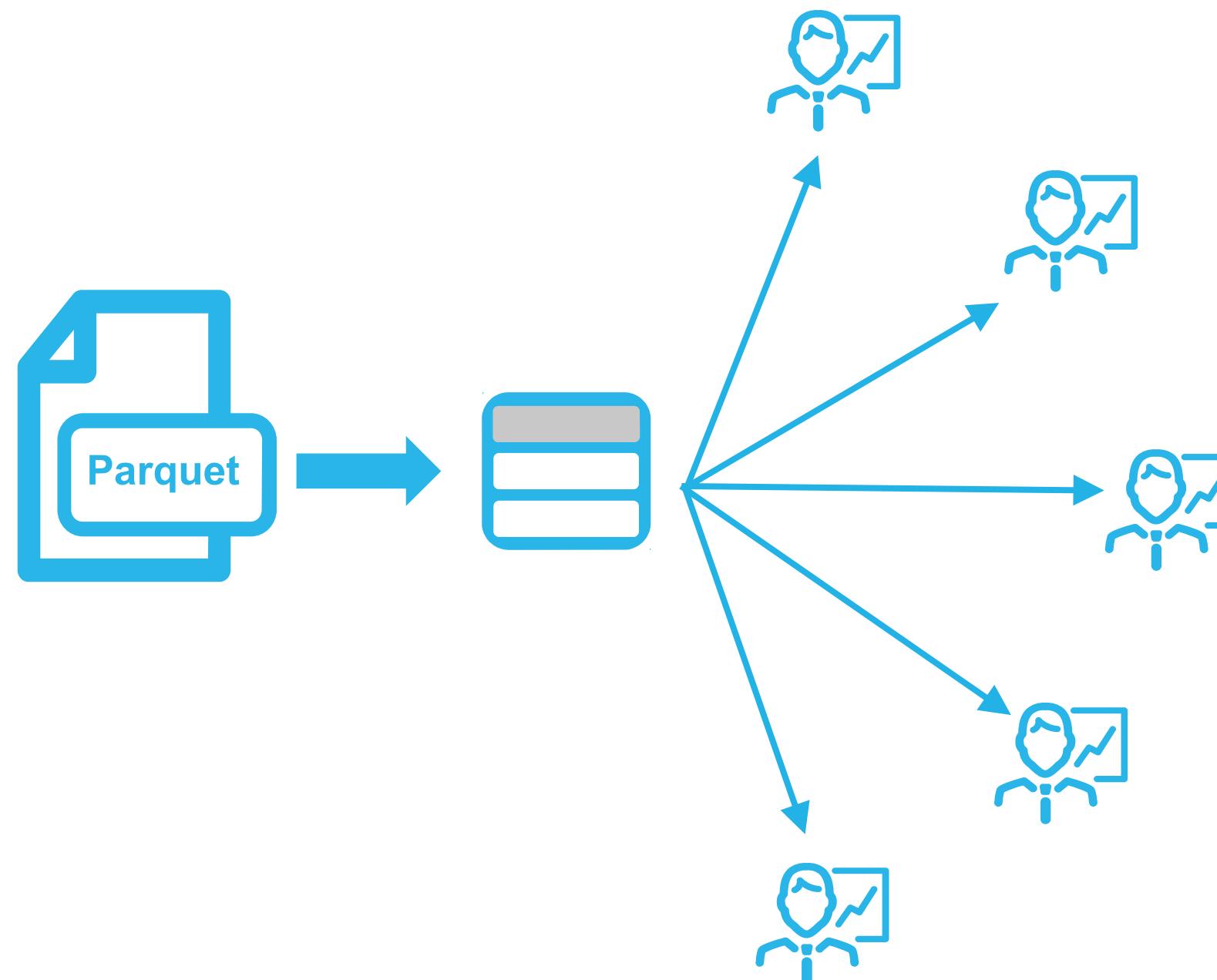


# WAYS TO QUERY SEMI-STRUCTURED DATA



- Query the variant data type directly
- Transform semi-structured data into structured data (tables or materialized views) and query that

# QUERY VARIANT DIRECTLY



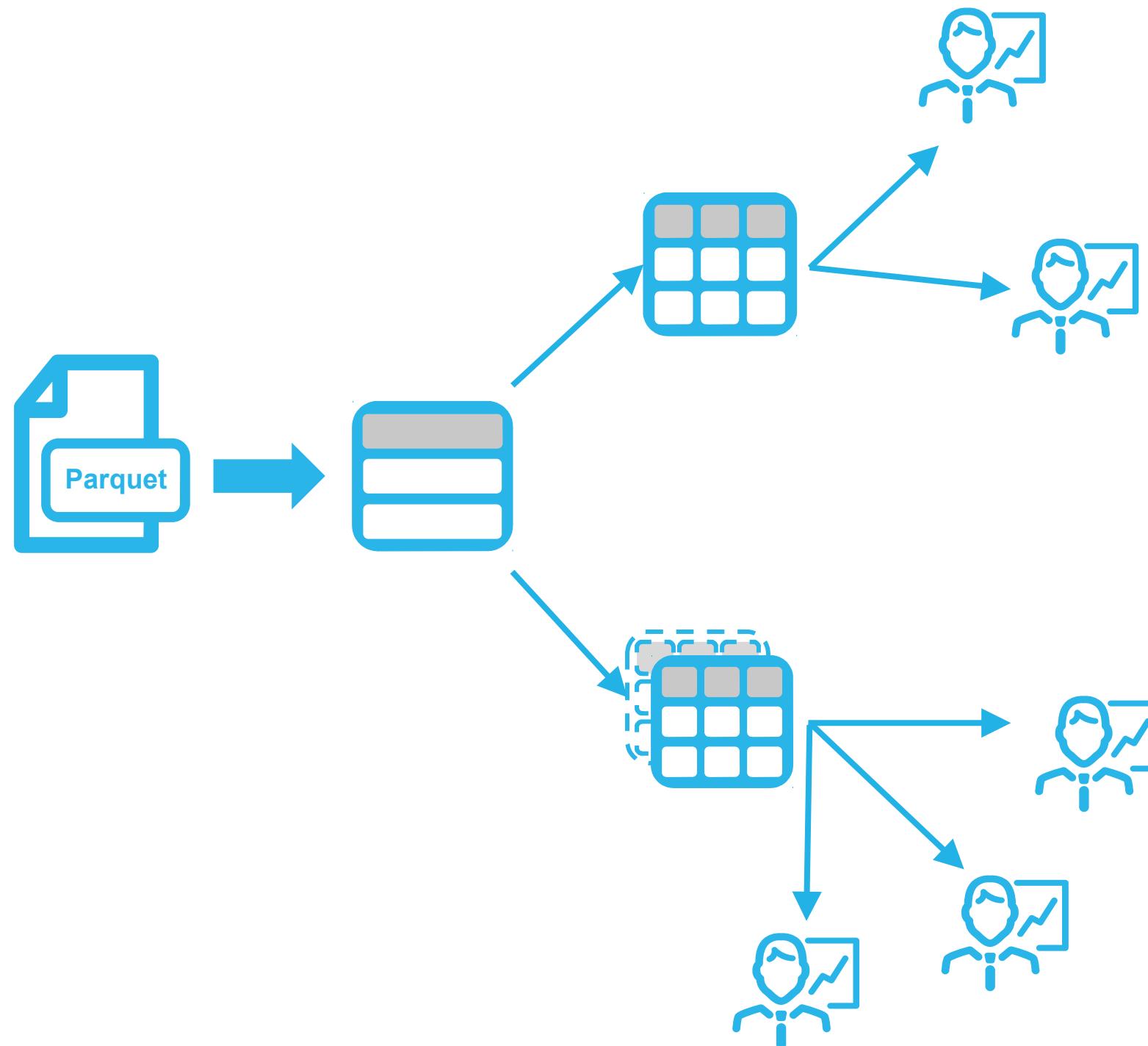
## Pros:

- Flexibility – no need to pre-determine what someone will want to query

## Cons:

- Duplication of effort – many people doing the same transformations against data
- Requires knowledge of how to query semi-structured data
- Not useful for BI tools

# CONVERT UP FRONT



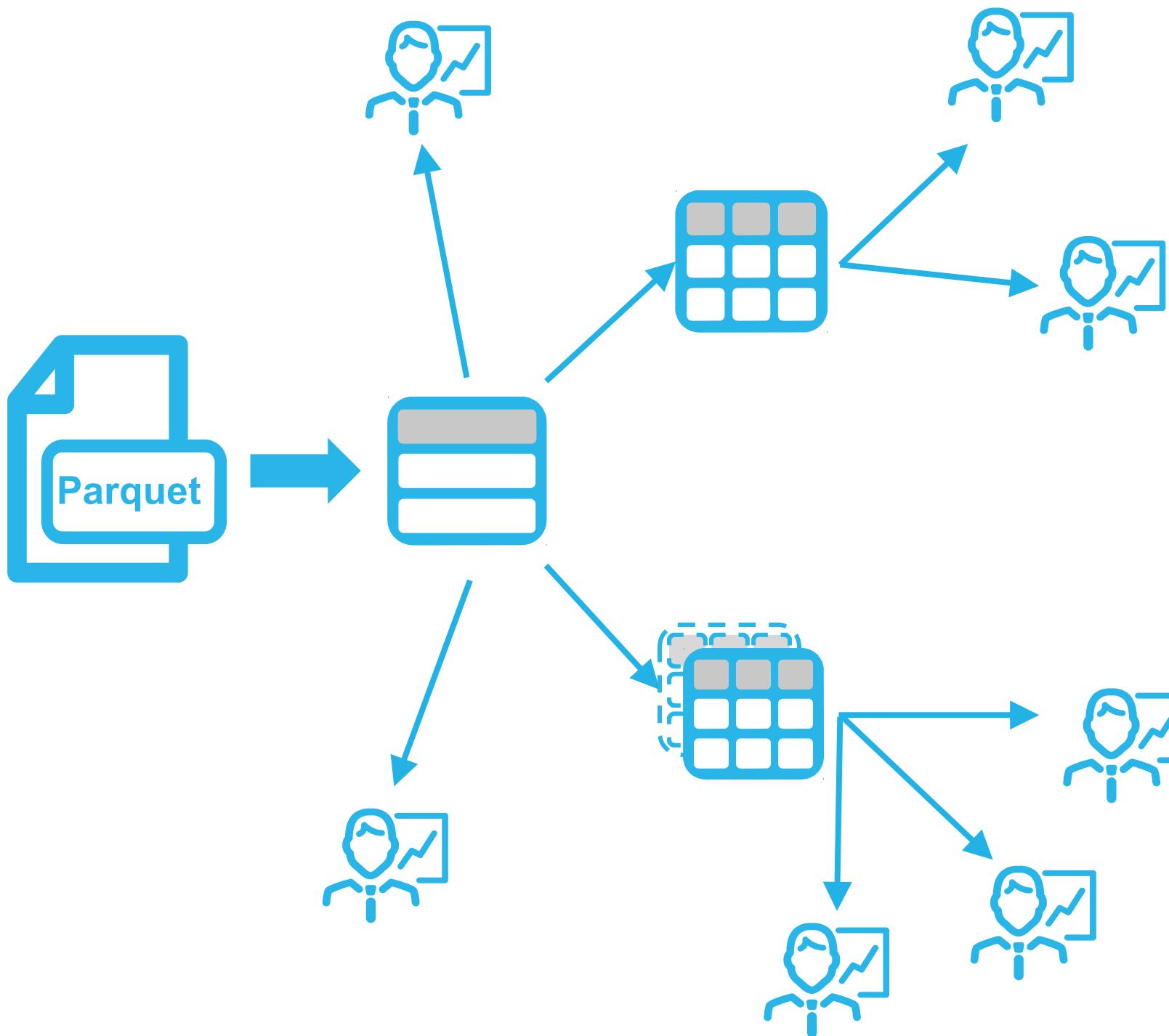
## Pros:

- Simple to query with basic SQL
- Faster queries
  - Parsing and aggregations performed up front
- Easily used by BI tools
- Performance benefits of micro-partitioning

## Cons:

- May require multiple tables and/or materialized views
- Need to know up front what users query
- Need to update if query needs change
- Lose access to raw data

# BEST OF BOTH WORLDS



- Everyone gets what they want
- Structured data for BI tools
- Access to raw data for exploration

# MATCH\_BY\_COLUMN\_NAME

```
{  
    "INPUT_DATE" : "2020-03-12",  
    "NATION" : "ALGERIA",  
    "POPULATION" : 438500000,  
    "REGION" : "AFRICA"  
}  
  
{  
    "INPUT_DATE" : "2020-01-17",  
    "NATION" : "COLUMBIA",  
    "POPULATION" : 50880000,  
    "REGION" : "SOUTH AMERICA"  
}  
  
{  
    "INPUT_DATE" : "2020-12-29",  
    "NATION" : "UNITED STATES",  
    "POPULATION" : 3295000000,  
    "REGION" : "NORTH AMERICA"  
}
```

- Use to load semi-structured data into columns based on outer-level keys
  - INPUT\_DATE
  - NATION
  - POPULATION
  - REGION
- Particularly useful for relatively flat files
- JSON, Avro, ORC, and Parquet
- Case-sensitive, or case-insensitive



# MATCH\_BY\_COLUMN\_NAME

```
CREATE TABLE nation (
    input_date DATE,
    nation VARCHAR,
    population NUMBER(10,0),
    region VARCHAR);

COPY INTO nation FROM @my_stage
MATCH_BY_COLUMN_NAME=CASE_INSENSITIVE
FILE_FORMAT = (TYPE = JSON);

SELECT * FROM nation;
```



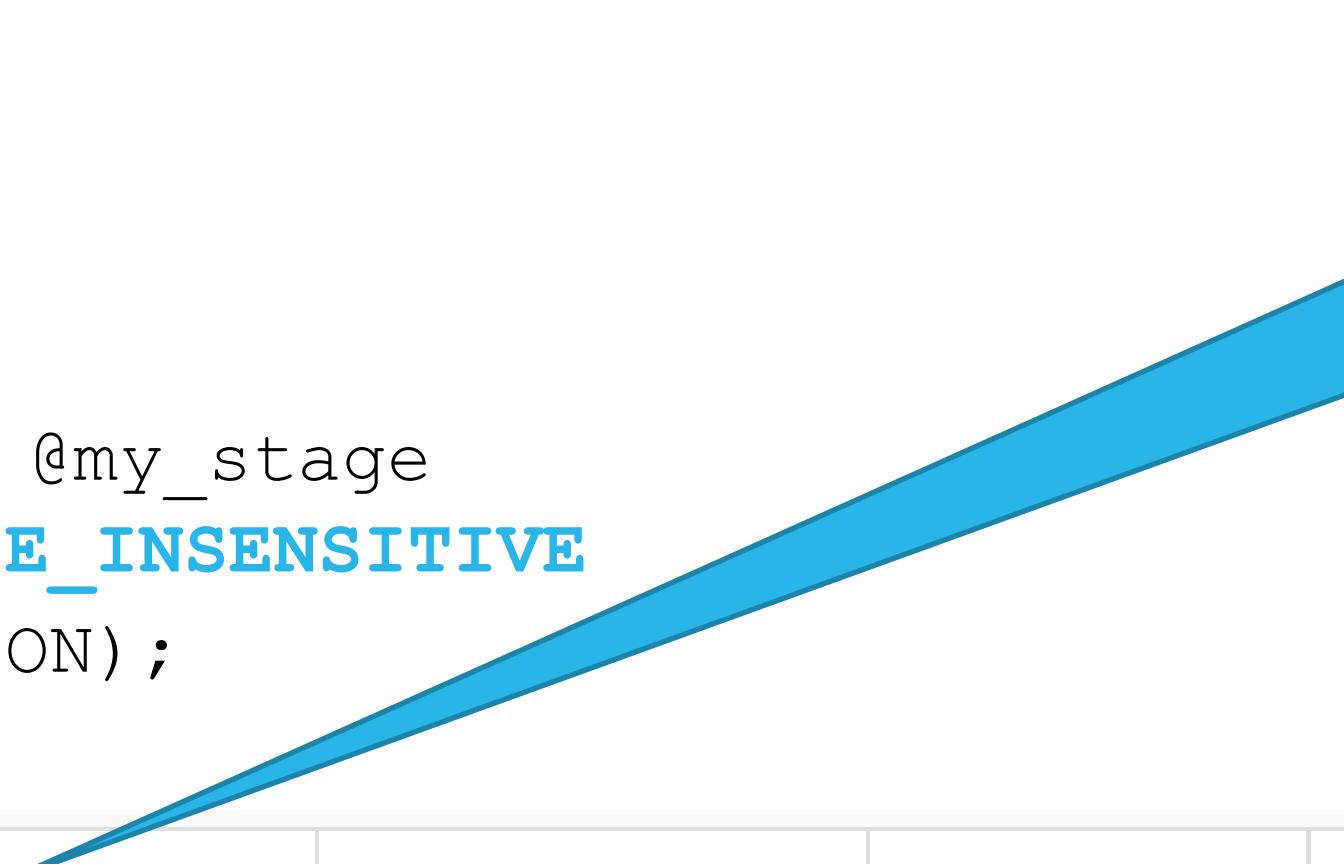
	INPUT_DATE	NATION	...	POPULATION	REGION
1	2020-03-12	ALGERIA		438,500,000	AFRICA
2	2020-01-17	COLUMBIA		50,880,000	SOUTH AMERICA
3	2020-12-29	UNITED STATES		3,295,000,000	NORTH AMERICA
4	2020-12-29	CANADA		38,010,000	NORTH AMERICA
5	2020-12-18	ZIMBABWE		14,860,000	AFRICA
6	2020-03-16	JAPAN		1,258,000,000	APAC
7	2021-06-26	IRELAND		5,032,827	EUR
8	2020-06-27	FRANCE		6,739,000,000	EUR
9	2021-05-17	CZECHIA		10,700,000	NORTH AMERICA
10	2020-09-28	UKRAINE		44,130,000	EUR
11	2020-02-14	GREECE		10,720,000	EUR
12	2020-11-23	SOUTH AFRICA		59,310,000	AFRICA
13	2020-10-31	AUSTRALIA		25,690,000	APAC

# MATCH\_BY\_COLUMN\_NAME: NESTED FILES

Nested values loaded into a VARIANT column

1. `CREATE TABLE countries (`  
    `coord     VARIANT,`  
    `country   VARCHAR,`  
    `id       NUMBER(10, 0),`  
    `name      VARCHAR);`
2. `COPY INTO countries FROM @my_stage`  
`MATCH_BY_COLUMN_NAME=CASE_INSENSITIVE`  
`FILE_FORMAT = (TYPE = JSON);`

Keys with a  
nested value load  
into a VARIANT  
column



	COORD	...	COUNTRY	ID	NAME
1	{"lat":22.183331,"lon":87.98333}		IN	1264368	Mahishadal



# SCHEMA INFERENCE

## Challenge

- Parquet, Avro and ORC data files
- Need to extract column names and data types
- Need to create tables, views
- Could simply load to VARIANT – but how to identify data structure?

## Snowflake Solutions

`INFER_SCHEMA()`

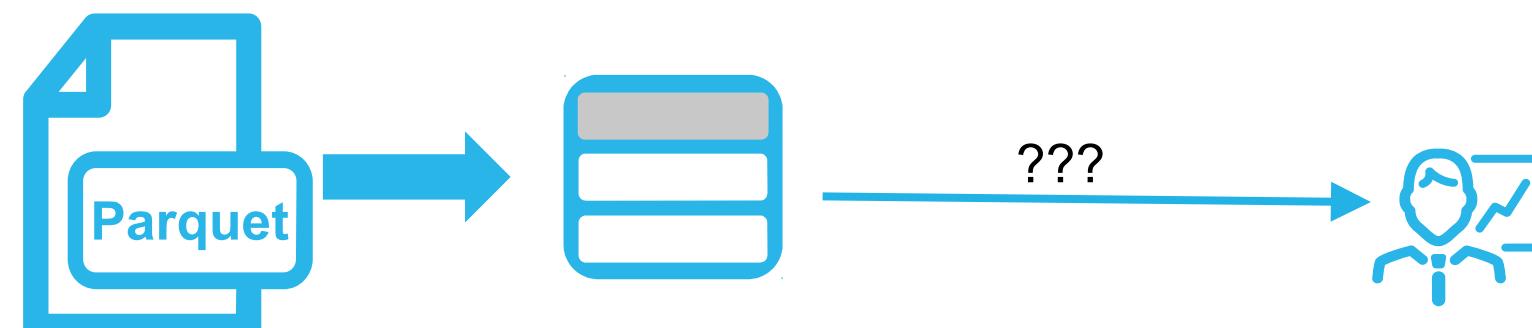
- Examines data file and infers structure

`CREATE TABLE USING TEMPLATE`

- Create table from inference

`GENERATE_COLUMN_DESCRIPTION()`

- Extracts column attributes



# SCHEMA INFERENCE



```
{  
    "input_date" : "2022-02-17",  
    "nation" : "ALGERIA",  
    "population" : 1000000000,  
    "region" : "AFRICA"  
}  
  
{  
    "input_date" : "2022-02-17",  
    "nation" : "ARGENTINA",  
    "population" : 1000000000,  
    "region" : "AMERICA"  
}  
  
{  
    "input_date" : "2022-02-17",  
    "nation" : "BRAZIL",  
    "population" : 1000000000,  
    "region" : "AMERICA"  
}
```

## 1. Stage the data file



# SCHEMA INFERENCE



```
SELECT *
FROM TABLE(INFER_SCHEMA(
    LOCATION      => '@parquet_stage'
    , FILE_FORMAT => 'my_parquet'));
```

	COLUMN_NAME	TYPE	NULLABLE	EXPRESSION
1	_COL_6	TEXT	TRUE	\$1:_COL_6::TEXT
2	_COL_21	TEXT	TRUE	\$1:_COL_21::TEXT
3	_COL_25	TEXT	TRUE	\$1:_COL_25::TEXT

1. Stage the data file

2. Infer the schema

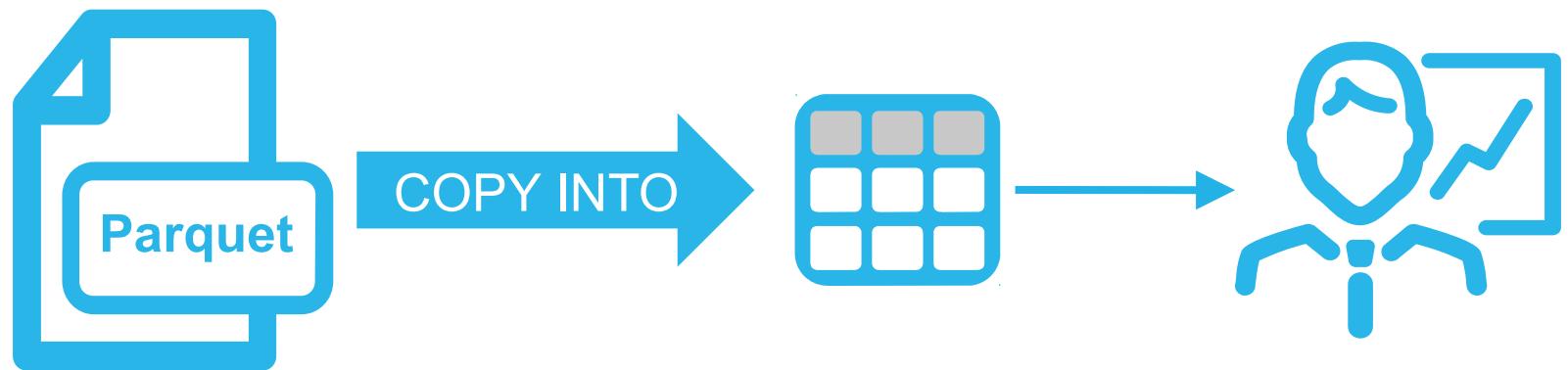
# SCHEMA INFERENCE



```
CREATE TABLE regions USING TEMPLATE (
SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))
FROM TABLE(INFER_SCHEMA(
    LOCATION => '@parquet_stage' ,
    FILE_FORMAT =>'my_parquet')));
```

1. Stage the data file
2. Infer the schema
3. Create table from template
  - a. Use INFER\_SCHEMA on file
  - b. Create a semi-structured object using ARRAY\_AGG and OBJECT\_CONSTRUCT
  - c. Create the table with USING TEMPLATE, from the output of the other functions

# SCHEMA INFERENCE



```
COPY INTO regions  
FROM @parquet_stage  
MATCH_BY_COLUMN_NAME = CASE_INSENSITIVE  
FILE_FORMAT = (TYPE = parquet);
```

1. Stage the data file
2. Infer the schema
3. Create table from template
4. Copy data into table, and query



# GENERATE\_COLUMN\_DESCRIPTION

```
SELECT GENERATE_COLUMN_DESCRIPTION(<expression>, '<type>')
```

Output of the  
INFER\_SCHEMA function,  
formatted as an array

Type of object to create from column list  
TABLE, EXTERNAL\_TABLE, or VIEW



# GENERATE\_COLUMN\_DESCRIPTION

```
{  
    "input_date" : "2022-02-17",  
    "nation" : "ALGERIA",  
    "population" : 1000000000,  
    "region" : "AFRICA"  
}  
  
{  
    "input_date" : "2022-02-17",  
    "nation" : "ARGENTINA",  
    "population" : 1000000000,  
    "region" : "AMERICA"  
}  
  
{  
    "input_date" : "2022-02-17",  
    "nation" : "BRAZIL",  
    "population" : 1000000000,  
    "region" : "AMERICA"  
}
```



```
SELECT GENERATE_COLUMN_DESCRIPTION  
    (ARRAY_AGG(OBJECT_CONSTRUCT(*)), 'table')  
AS COLUMNS  
FROM TABLE(INFER_SCHEMA(  
    location => '@parquet_stage'  
, file_format => 'my_parquet'));
```



```
"input_date" DATE NOT NULL,  
"nation" TEXT NOT NULL,  
"population" NUMBER(10, 0) NOT NULL,  
"region" TEXT NOT NULL
```



# GENERATE\_COLUMN\_DESCRIPTION

```
SELECT GENERATE_COLUMN_DESCRIPTION
  (ARRAY_AGG(OBJECT_CONSTRUCT(*)), 'external_table')
AS COLUMNS
FROM TABLE(INFER_SCHEMA(
    location => '@parquet_stage'
  , file_format => 'my_parquet'));
```



```
"input_date" DATE AS ($1:input_date::DATE)
"nation" TEXT AS ($1:nation::TEXT),
"population" NUMBER(10, 0) AS ($1:population::NUMBER(10, 0),
"region" TEXT AS ($1:region::TEXT)
```



# GENERATE\_COLUMN\_DESCRIPTION

```
SELECT GENERATE_COLUMN_DESCRIPTION
  (ARRAY_AGG(OBJECT_CONSTRUCT(*)), 'view')
AS COLUMNS
FROM TABLE(INFER_SCHEMA(
    location => '@parquet_stage'
  , file_format => 'my_parquet'));
```



```
"input_date" ,
"nation" ,
"population" ,
"region"
```



# UNLOADING DATA



# MODULE AGENDA

- Workflow
- Unload Options
- Unloading Semi-Structured Data



# WORKFLOW



# UNLOADING DATA

- The same process as loading, but in reverse

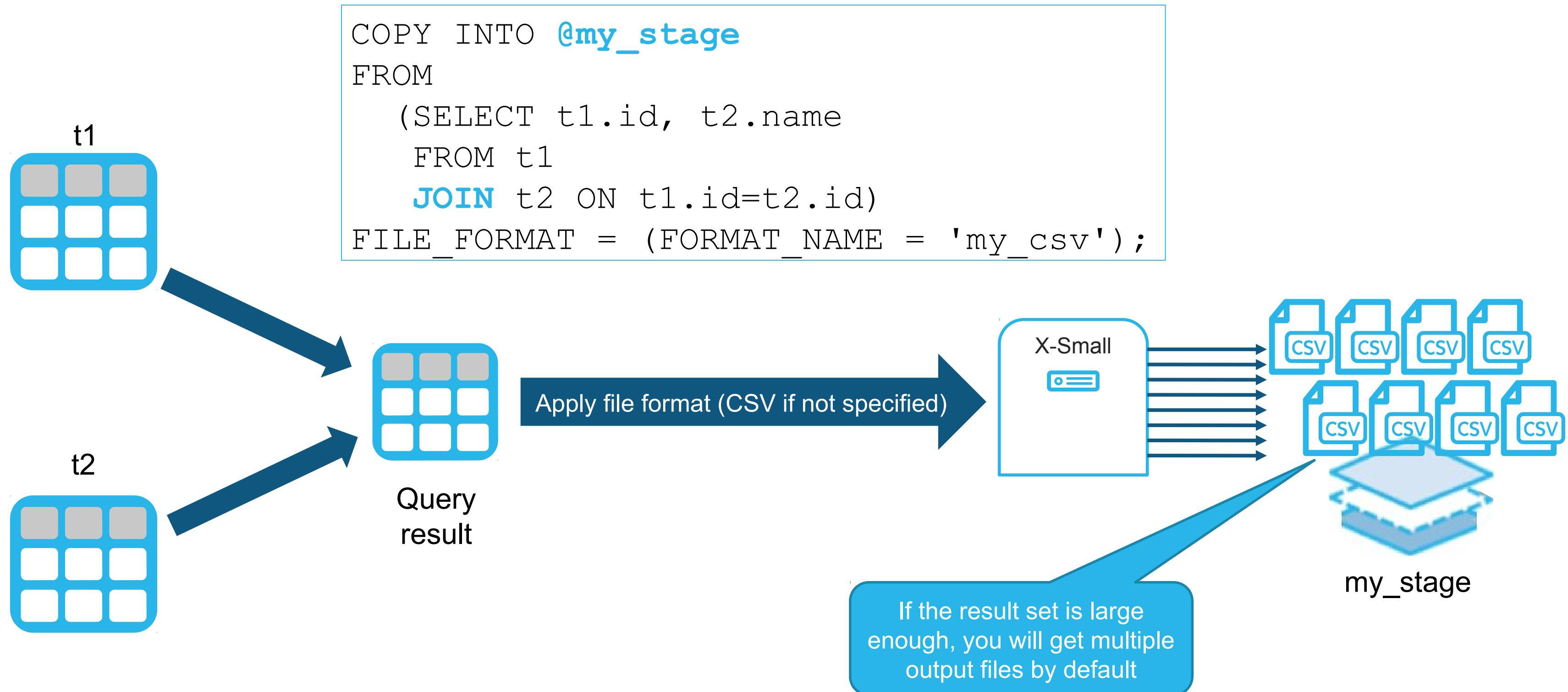
```
COPY INTO <stage> FROM <table>;
```

- More flexibility in transformations

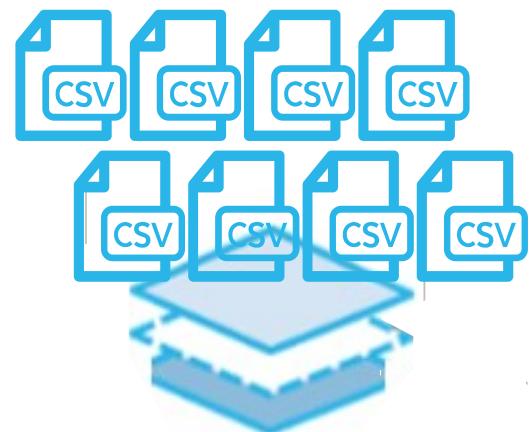
```
COPY INTO <stage> FROM (<any SELECT statement>);
```



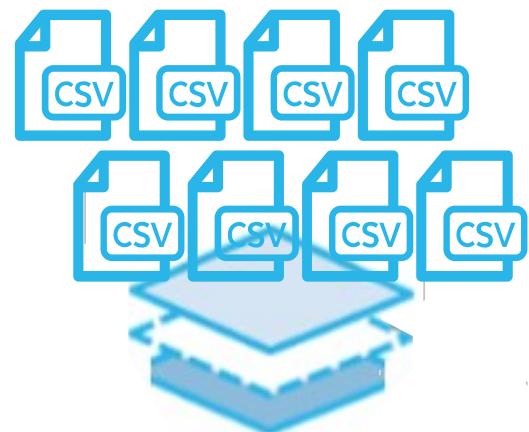
# UNLOAD WORKFLOW



# UNLOAD WORKFLOW



Use GET from a command-line client  
to retrieve files from an internal stage



Use 3<sup>rd</sup> party or cloud provider tools to  
retrieve files from an external stage

# UNLOAD OPTIONS



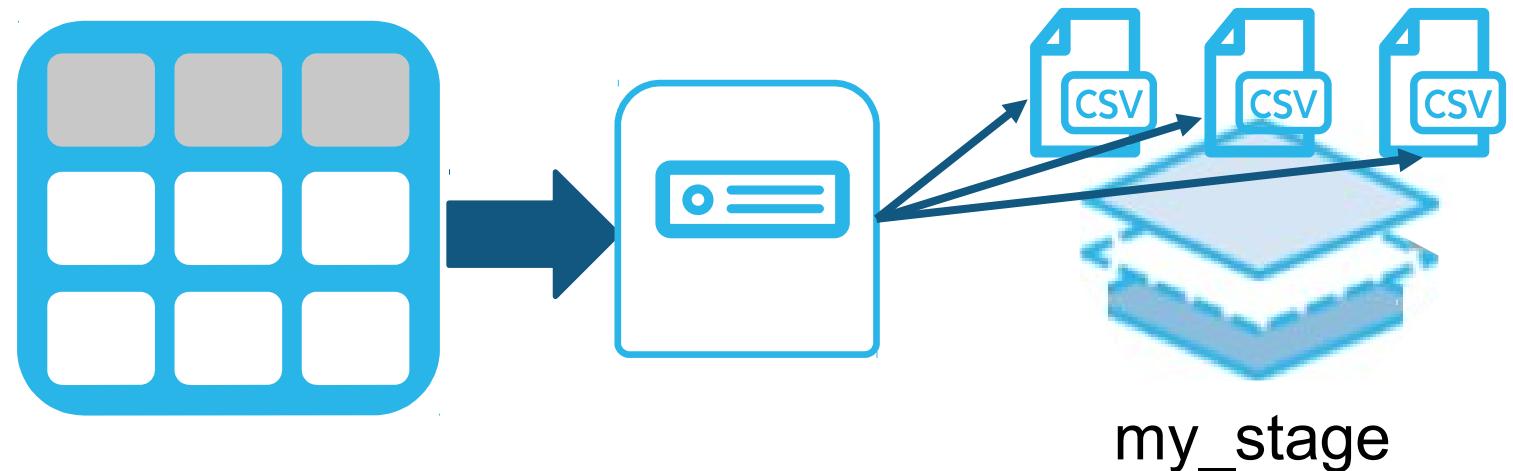
# FILE FORMATS



- CSV
  - Default if no format is specified
  - Can use custom format to change things like the delimiter used
- JSON
  - Unload from VARIANT data type
  - Use `object_construct()` to unload data that is not of type VARIANT
- Parquet
  - Unloads to a single column by default
  - Use `SELECT` to unload columns

# FILE SIZE

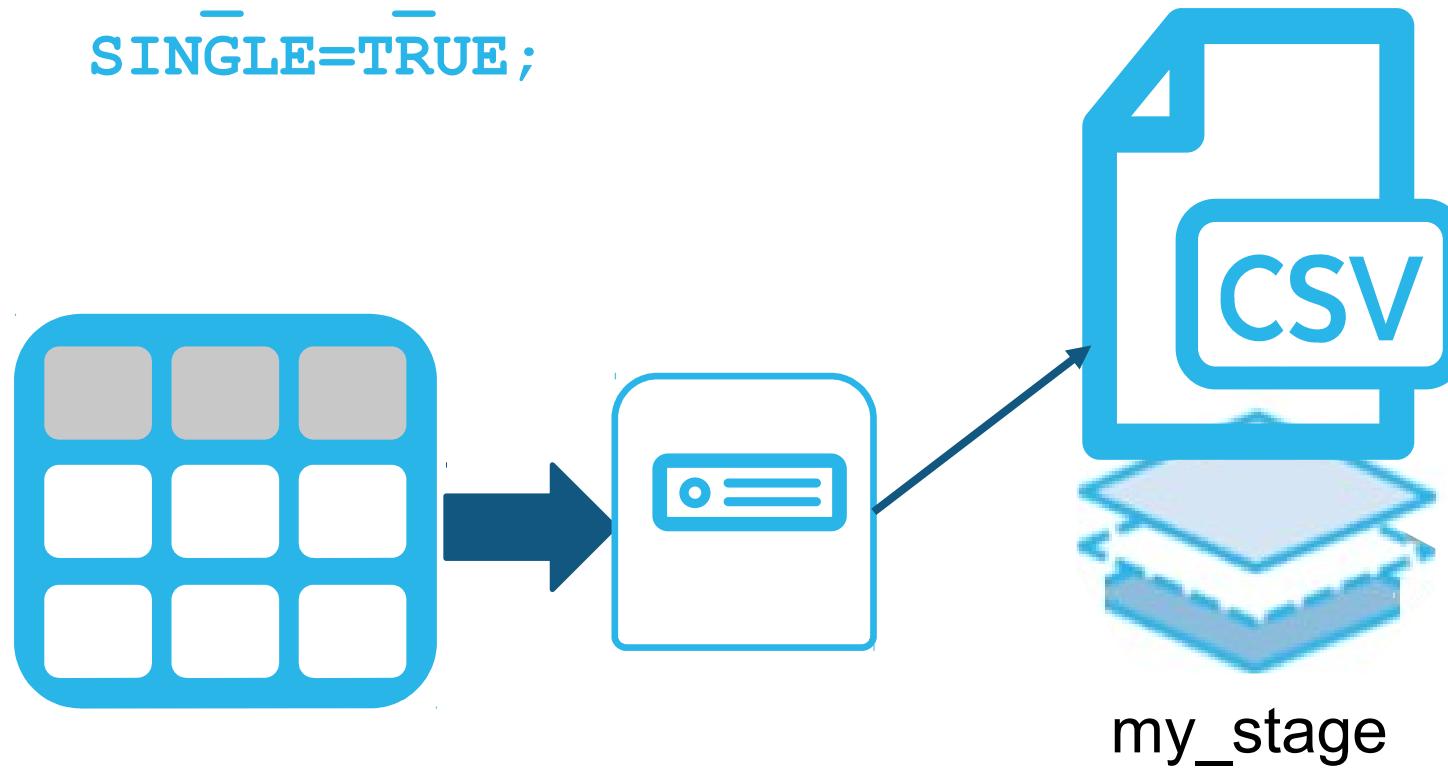
```
COPY INTO @my_stage  
FROM orders  
MAX_FILE_SIZE=30000000;
```



- Default max output file size is ~16MB (compressed)
- Change with `MAX_FILE_SIZE`
- Largest valid value is 5GB
- Actual size and number of files will vary

# UNLOAD AS A SINGLE FILE

```
COPY INTO @my_stage  
FROM orders  
MAX_FILE_SIZE=5000000000  
SINGLE=TRUE;
```

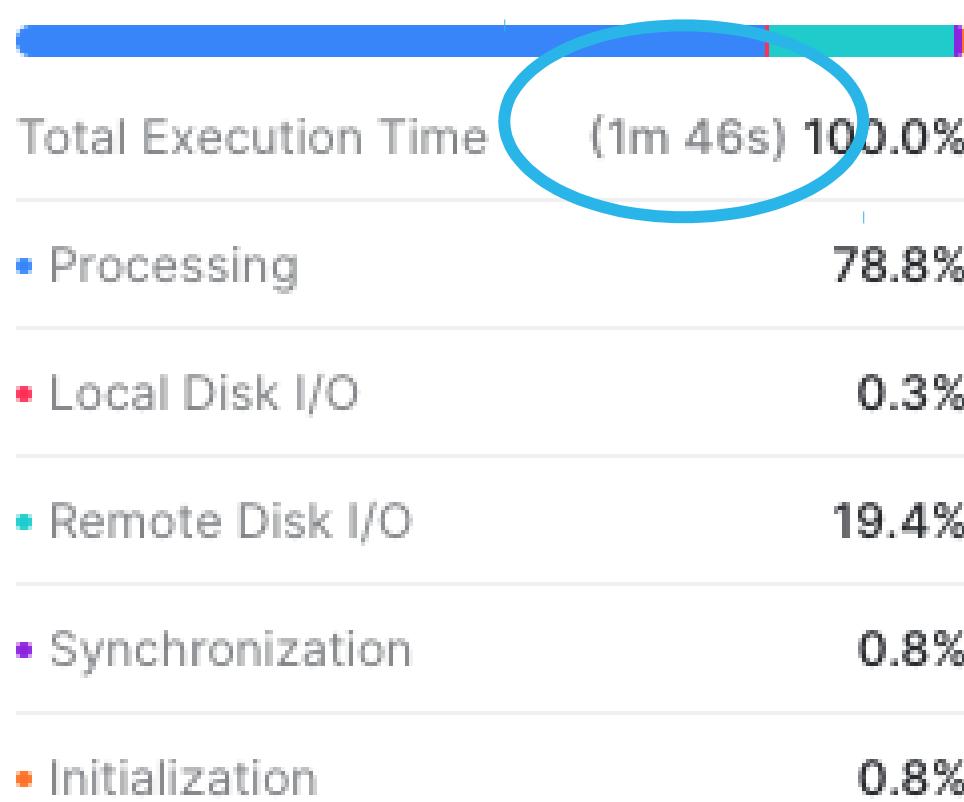


- Use SINGLE with MAX\_FILE\_SIZE to unload the data into a single file
- Disadvantages:
  - No parallelization
  - Significant impact on unload time (always use XS virtual warehouse)
- Advantages:
  - Convenience for downstream developers
  - CDN (Content Delivery Network) feed
  - No need to assemble content, especially if output file will be small

# UNLOAD AS A SINGLE FILE

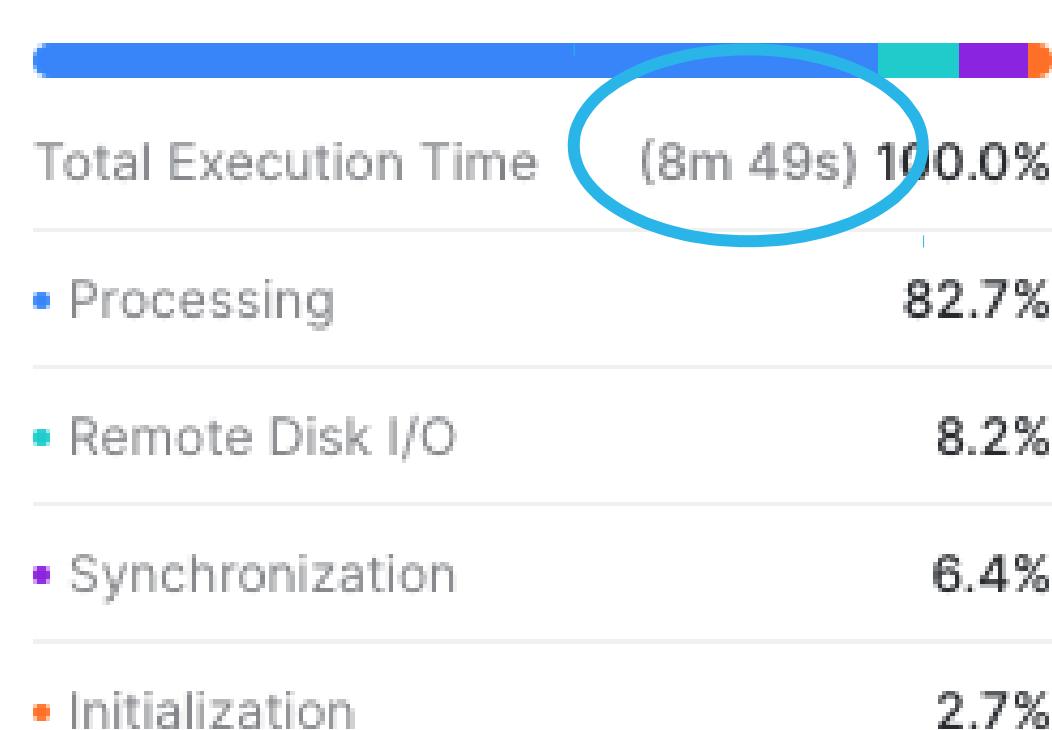
```
COPY INTO @orders_stage  
FROM orders;
```

## Profile Overview (Finished)



```
COPY INTO @orders_stage  
FROM orders  
SINGLE=TRUE  
MAX_FILE_SIZE=5268709120;
```

## Profile Overview (Finished)



# FILE PATHS AND NAMES

- Default file names: 

name
orders_stage/data_0_0_0.csv.gz
orders_stage/data_0_0_1.csv.gz

  - data\_<x>\_<y>\_<z>
- Extract to **path in stage**: 

name
orders_stage/my_orders/data_0_0_0.csv.gz
orders_stage/my_orders/data_0_0_1.csv.gz

  - COPY INTO @orders\_stage/my\_orders/  
FROM orders;
- To override file name **prefix**: 

name
orders_stage/us_orders_0_0_0.csv.gz
orders_stage/us_orders_0_0_1.csv.gz

  - COPY INTO @orders\_stage/us\_orders  
FROM sales\_table;



# PARTITION OUTPUT FILES

- Create output files with meaningful names using one or more column values

```
USE SNOWBEARAIR_DB.MODELED;
```

```
COPY INTO @%members
FROM members
PARTITION BY (state);
```

```
LIST @%members;
```

name	...	size
Alabama/data_01a3fce5-0401-67fb-005b-e40300089fc6_03_2_0.csv.gz		1,808
Alaska/data_01a3fce5-0401-67fb-005b-e40300089fc6_03_3_0.csv.gz		464
Arizona/data_01a3fce5-0401-67fb-005b-e40300089fc6_03_3_0.csv.gz		1,776



# PARTITION OUTPUT FILES

- Combine text and column values

```
COPY INTO @my_stage
FROM customer_address
PARTITION BY ('State=' || CA_STATE || '/County=' || CA_COUNTY);
```

```
LIST @drl_db.public.my_stage;
```

name	size	modified
my_stage/State=AK/County=Aleutians East Borough/data_01a3fd03-0401-66e3-005	370,240	40
my_stage/State=AK/County=Aleutians West Census Area/data_01a3fd03-0401-66e3-	374,336	a7
my_stage/State=AK/County=Anchorage Borough/data_01a3fd03-0401-66e3-005e-e	371,488	af:



# LAB EXERCISE: 5

## Unload Structured Data

40 minutes

### Tasks:

- Unload a pipe-delimited file
- Unload part of a table
- JOIN and unload



# **UNLOADING SEMI-STRUCTURED DATA**



# UNLOAD PARQUET

```
SELECT $1
FROM cities;

{
  "continent": "Europe",
  "country": {
    "city": [
      "Paris",
      "Nice",
      "Marseilles",
      "Cannes"
    ],
    "name": "France"
  }
}
```

- Unloads to a single column by default
- Specify output file format
- Use SELECT to unload into multiple columns



# UNLOAD TO PARQUET

- Source table stores parquet data in a single VARIANT column

```
COPY INTO @my_stage/output/  
FROM cities  
FILE_FORMAT = (TYPE = 'parquet');
```



# UNLOAD TO PARQUET

- Source table stores parquet data in a single VARIANT column

```
COPY INTO @my_stage/output/
FROM cities
FILE_FORMAT = (TYPE = 'parquet');
```

```
SELECT $1 FROM @my_stage/output/
(FILE_FORMAT => 'my_parquet');
```

```
$1
```

```
{ "_COL_0": {"continent": "Europe", "country": {"city": ["Paris", "Nice", "Marseilles", "Cannes"], "name": "France"} } }
{ "_COL_0": {"continent": "Europe", "country": {"city": ["Athens", "Piraeus", "Hania", "Heraklion", "Rethymnon", "Fira"]}, "name": "Greece" } }
{ "_COL_0": {"continent": "North America", "country": {"city": ["Toronto", "Vancouver", "St. John's", "Saint John", "Montreal"]}, "name": "Canada" } }
```



# UNLOAD STRUCTURED DATA TO PARQUET

```
COPY INTO @my_stage/output/  
FROM  
  (SELECT * FROM members)  
HEADER = true  
FILE_FORMAT = (TYPE = 'parquet');
```

Table with standard  
SQL data types

Output  
File Type



# UNLOAD STRUCTURED DATA TO PARQUET

```
COPY INTO @my_stage/output/
FROM
  (SELECT * FROM members)
HEADER = true
FILE_FORMAT = (TYPE = 'parquet');
```

```
SELECT $1 FROM @my_stage/output/
(FILE_FORMAT => 'my_parquet');
```

\$1
{ "AGE": 21, "CITY": "Evansville", "EMAIL": "jchoules0@meetup.com", "FIRSTNAME": "Jed", "GENDER": "M", "LASTNAME": "Choules", "MEMBERID": 1000000000000000000 }
{ "AGE": 25, "CITY": "San Jose", "EMAIL": "smammatt1@narod.ru", "FIRSTNAME": "Sunny", "GENDER": "M", "LASTNAME": "Mammatt", "MEMBERID": 1000000000000000001 }
{ "AGE": 37, "CITY": "Knoxville", "EMAIL": "oaaronsohn2@ibm.com", "FIRSTNAME": "Orelee", "GENDER": "F", "LASTNAME": "Aaronsohn", "MEMBERID": 1000000000000000002 }
{ "AGE": 39, "CITY": "Humble", "EMAIL": "dgrewer3@wsj.com", "FIRSTNAME": "Dew", "GENDER": "M", "LASTNAME": "Grewer", "MEMBERID": 1000000000000000003 }



# UNLOAD STRUCTURED DATA TO PARQUET

```
COPY INTO @my_stage/output/
FROM
  (SELECT * FROM members)
HEADER = true
FILE_FORMAT = (TYPE = 'parquet');
```

```
SELECT $1 FROM @my_stage/output/
(FILE_FORMAT => 'my_parquet');
```

\$1
{ "AGE": 21, "CITY": "Evansville", "EMAIL": "jchoules0@meetup.com", "FIRSTNAME": "Jed", "GENDER": "M", "LASTNAME": "Choules",
{ "AGE": 25, "CITY": "San Jose", "EMAIL": "smammatt1@narod.ru", "FIRSTNAME": "Sunny", "GENDER": "M", "LASTNAME": "Mammatt",
{ "AGE": 37, "CITY": "Knoxville", "EMAIL": "oaaronsohn2@ibm.com", "FIRSTNAME": "Orelee", "GENDER": "F", "LASTNAME": "Aaronsohn",
{ "AGE": 39, "CITY": "Humble", "EMAIL": "dgrewer3@wsj.com", "FIRSTNAME": "Dew", "GENDER": "M", "LASTNAME": "Grewer", "MEME": "



# UNLOAD STRUCTURED DATA TO PARQUET

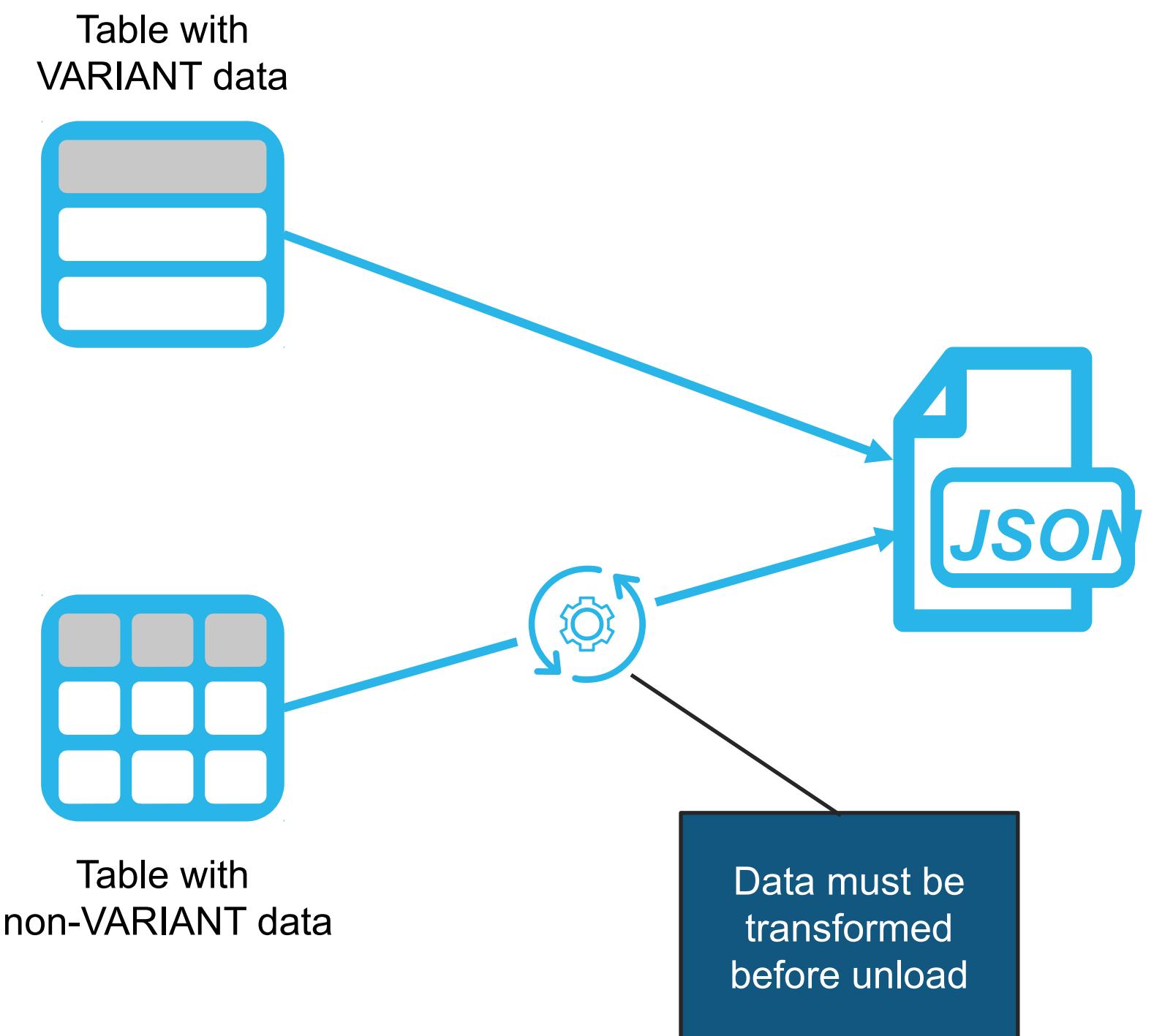
```
COPY INTO @my_stage/output/
FROM
  (SELECT * FROM members)
HEADER = false
FILE_FORMAT = (TYPE = 'parquet');
```

```
SELECT $1 FROM @my_stage/output/
(FILE_FORMAT => 'my_parquet');
```

\$1
{ "_COL_0": "bc1fc0f9-ccb5-46b9-ac8a-e70a44c5b253", "_COL_1": 2819999, "_COL_10": "Evansville", "_COL_11": "Indiana", "_COL_12": "7" }
{ "_COL_0": "10f54a6d-a010-4de7-b29e-d0aad565b1c3", "_COL_1": 3230571, "_COL_10": "San Jose", "_COL_11": "California", "_COL_12": "7" }
{ "_COL_0": "4cc95016-d078-4d4d-bf19-f8b50b8efabd", "_COL_1": 7475443, "_COL_10": "Knoxville", "_COL_11": "Tennessee", "_COL_12": "7" }
{ "_COL_0": "ae1168e1-840f-477d-a03f-8e4f491c0af0", "_COL_1": 5235957, "_COL_10": "Humble", "_COL_11": "Texas", "_COL_12": "7" }



# UNLOAD TO JSON



- Outputs to Newline-Delimited JSON (NDJSON) standard format
- Unload directly from VARIANT to JSON
- Unload from non-VARIANT columns by converting to JSON in SELECT
  - Use functions such as OBJECT\_CONSTRUCT

# UNLOAD TO JSON

- Source file has two columns: v (VARIANT) and t (TIMESTAMP)

```
USE SNOWFLAKE_SAMPLE_DATA.WEATHER.DAILY_14_TOTAL;
```

```
COPY INTO @drl_db.public.my_stage
FROM (SELECT v FROM daily_14_total LIMIT 10000)
FILE_FORMAT = (TYPE = 'json')
MAX_FILE_SIZE = 1000000000;
```



# UNLOAD TO JSON

- Source file has two columns: v (VARIANT) and t (TIMESTAMP)

```
USE SNOWFLAKE_SAMPLE_DATA.WEATHER.DAILY_14_TOTAL;
```

```
COPY INTO @drl_db.public.my_stage
FROM (SELECT v FROM daily_14_total LIMIT 10000)
FILE_FORMAT = (TYPE = 'json')
MAX_FILE_SIZE = 1000000000;
```

[LIST @drl\\_db.public.my\\_stage](#)

<b>name</b>	<b>size</b>	<b>md5</b>	<b>last_modified</b>
my_stage/data_0_0_0.json.gz	6,351,760	f285b1c88e3333f56215d71ee08bad3c	Mon, 2 May 2022 03:48:18 GMT



# UNLOAD STRUCTURED DATA TO JSON

- Source table is a structured table with many columns

```
USE SNOWFLAKE_SAMPLE_DATA.TPCH_SF10;
```

```
COPY INTO @drl_db.public.my_stage
FROM (SELECT OBJECT_CONSTRUCT(*) FROM lineitem)
FILE_FORMAT = (TYPE = 'json');
```



# UNLOAD STRUCTURED DATA TO JSON

- Source table is a structured table with many columns

```
USE SNOWFLAKE_SAMPLE_DATA.TPCH_SF10;
```

```
COPY INTO @drl_db.public.my_stage
FROM (SELECT OBJECT_CONSTRUCT(*) FROM lineitem)
FILE_FORMAT = (TYPE = 'json');
```

```
SELECT * FROM @drl_db.public.my_stage/data_0_0_0.json.gz
(FILE_FORMAT => 'my_json');
```

```
$1
```

```
{ "L_COMMENT": "ickly regular packages. blithely ironic ", "L_COMMITDATE": "1994-09-09", "L_DISCOUNT": 0.07, "L_EXTENDEDPRICE": 1012.43, "L_FREIGHT": 8.5, "L_ORDERKEY": 100000, "L_ORDERSTATUS": "F", "L_QUANTITY": 30, "L_SHIPMODE": "R", "L_SHINUM": 1, "L_SHIPPER": "QUICK", "L_SUPPLIER": "Sparta", "L_SUPPKEY": 1000000000000000000}, { "L_COMMENT": " the furiously final requests. quick", "L_COMMITDATE": "1994-10-22", "L_DISCOUNT": 0.02, "L_EXTENDEDPRICE": 1012.43, "L_FREIGHT": 8.5, "L_ORDERKEY": 100001, "L_ORDERSTATUS": "F", "L_QUANTITY": 30, "L_SHIPMODE": "R", "L_SHINUM": 1, "L_SHIPPER": "QUICK", "L_SUPPLIER": "Sparta", "L_SUPPKEY": 1000000000000000001}, { "L_COMMENT": " along the furiously ", "L_COMMITDATE": "1994-09-02", "L_DISCOUNT": 0, "L_EXTENDEDPRICE": 33934.64, "L_FREIGHT": 10.0, "L_ORDERKEY": 100002, "L_ORDERSTATUS": "F", "L_QUANTITY": 30, "L_SHIPMODE": "R", "L_SHINUM": 1, "L_SHIPPER": "QUICK", "L_SUPPLIER": "Sparta", "L_SUPPKEY": 1000000000000000002}
```



# UNLOAD STRUCTURED DATA TO JSON

```
{  
    "L_COMMENT": "are slyly. regular, bold packages",  
    "L_COMMITDATE": "1995-03-11",  
    "L_DISCOUNT": 0.09,  
    "L_EXTENDEDPRICE": 2738.62,  
    "L_LINENUMBER": 2,  
    "L_LINESSTATUS": "F",  
    "L_ORDERKEY": 5232544,  
    "L_PARTKEY": 341328,  
    "L_QUANTITY": 2,  
    "L_RECEIPTDATE": "1995-02-13",  
    "L_RETURNFLAG": "A",  
    "L_SHIPDATE": "1995-02-10",  
    "L_SHIPINSTRUCT": "DELIVER IN PERSON",  
    "L_SHIPMODE": "AIR",  
    "L_SUPPKEY": 16338,  
    "L_TAX": 0.03  
}
```



# LAB EXERCISE: 6

## Unload Semi-Structured Data

30 minutes

### Tasks:

- Unload Semi-Structured JSON Data
- Unload Semi-Structured JSON Data to a Dynamic Path
- Unload Structured Data to a JSON File
- Unload Semi-Structured Parquet Data



# WORKING WITH UNSTRUCTURED DATA



# MODULE AGENDA

- Overview
- Concepts
- Workflow



# OVERVIEW



# WHAT IS UNSTRUCTURED DATA?

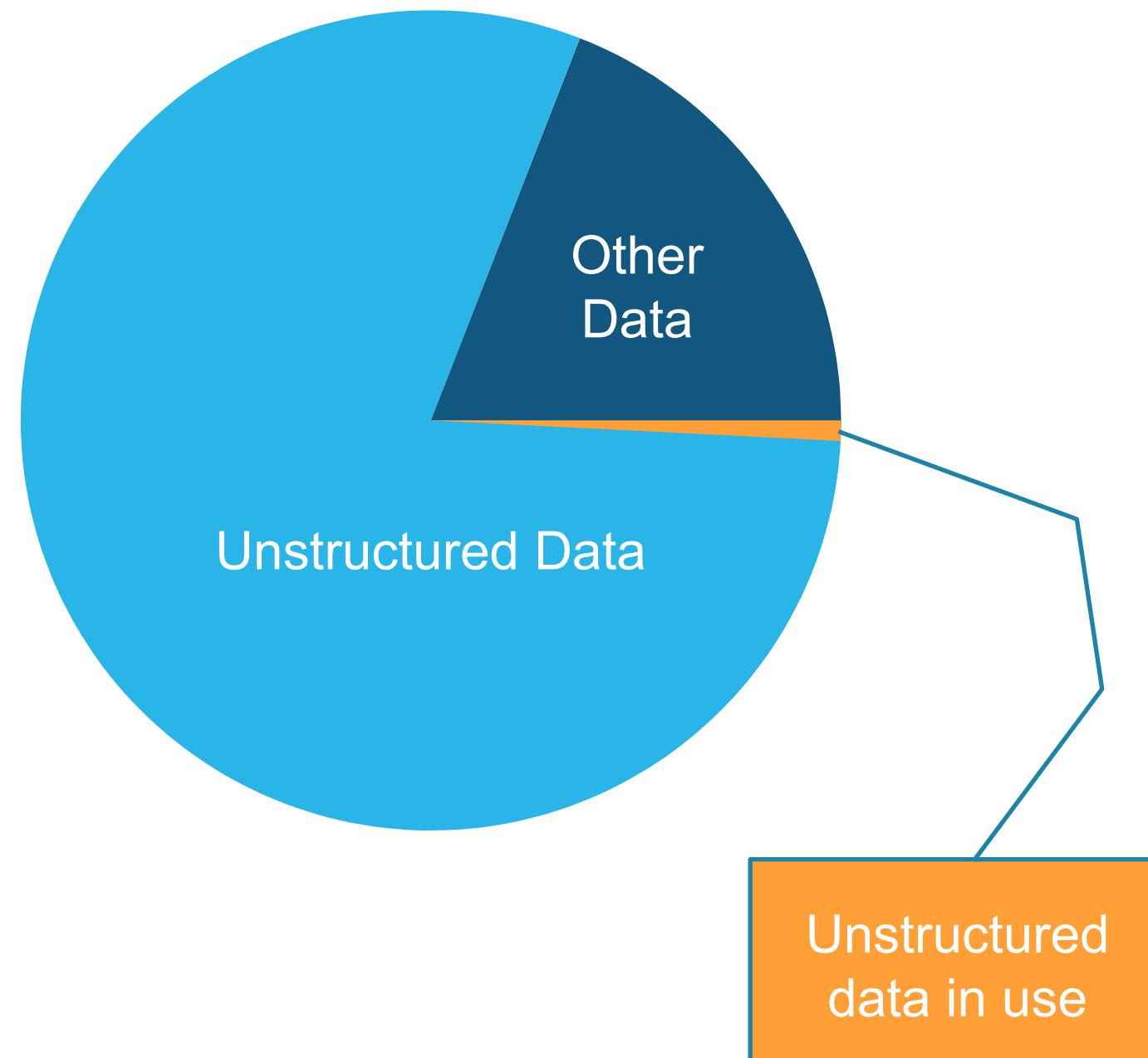
- Files without a pre-determined structure or schema

- Examples:

- Social media conversations
- Images
- Video
- Audio
- Documents
- And more...



# WHY IS IT IMPORTANT?



According to IDC projections\* reported by Analytics Insight –

- By the year 2025, 80% of the world's data will be unstructured
- Currently, about 0.5% of unstructured data is analyzed and used

\* Worldwide Global Datasphere and Global StorageSphere Structured and Unstructured Data Forecast, 2021-2025: July 2021

# SAMPLE USE CASES

## UNSTRUCTURED DATA



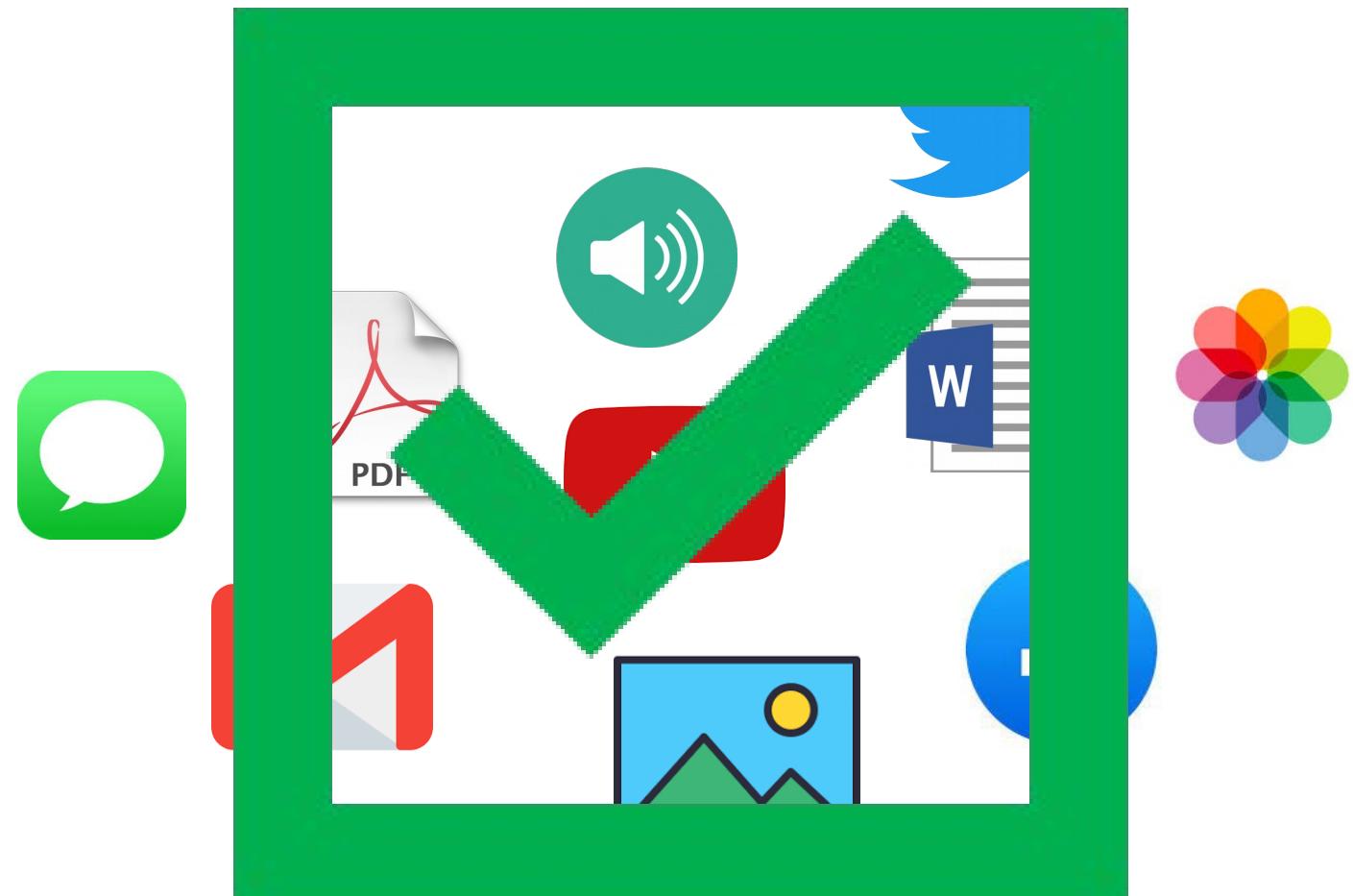
- Medical image (DICOM) analysis for research and diagnosis
- Sentiment analysis from social media or call center recordings
- Store screenshots of documents and extract text using OCR
- Text analysis for chatbot support
- Surveillance data or photos for facial recognition

# CHALLENGES



- Complex
- Difficult to analyze
- Many different file formats
- Difficult to join with other data sets
- Difficult to manage and secure

# UNSTRUCTURED DATA ON SNOWFLAKE



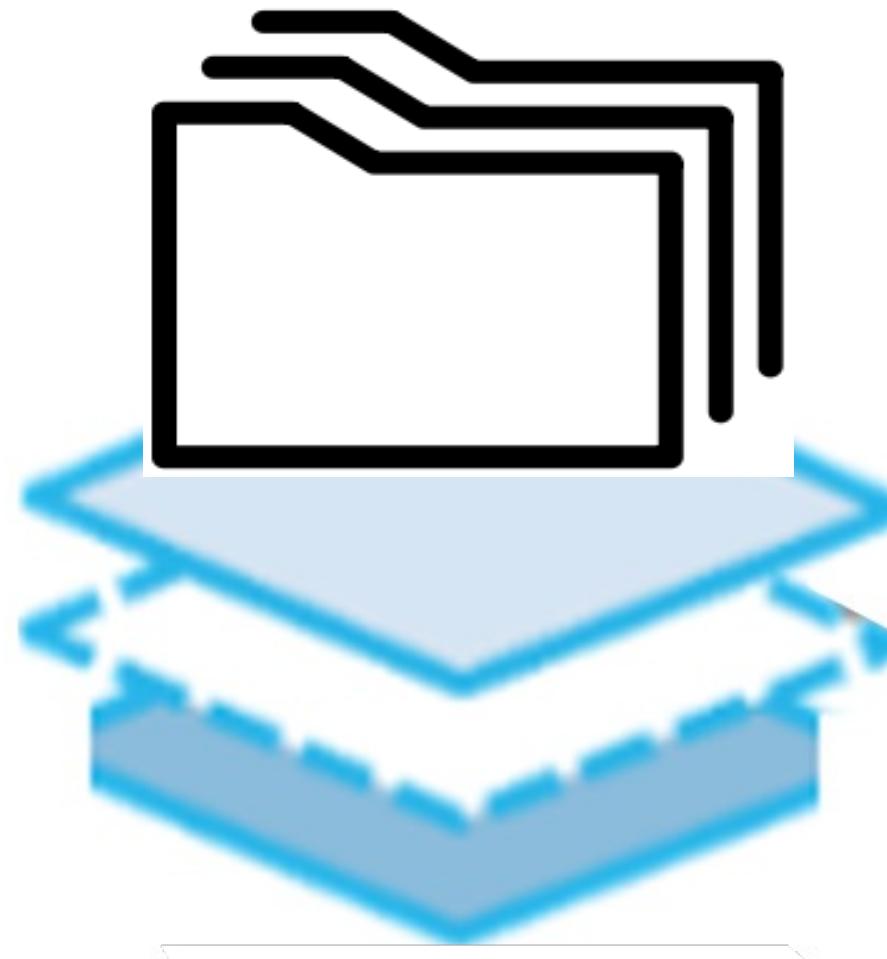
- Load to internal or external stages
- Catalog with directory tables
- Secure with Snowflake-native tools
- Access with REST APIs and URLs
- Process with Java and external functions
- Securely share

# CONCEPTS



# STAGES

- Unstructured data files are stored in internal or external named stages
- Stage options:
  - Enable directory tables (recommended)
  - Access URLs
  - Specify encryption type (server- or client-side)



# DIRECTORY TABLES

- Built-in table that stores a catalog of the files in a stage
- Must be specifically enabled: CREATE STAGE <name> **DIRECTORY= (ENABLE=true)** ;
- View contents: SELECT \* FROM  **DIRECTORY (@<stage\_name>)**  ;

RELATIVE_PATH	SIZE	LAST_MODIFIED	MD5	ETAG	FILE_URL
Image1.png.gz	5904	2022-01-25 14:24:30...	a6812c26616dc3b...	a6812c26616dc...	<a href="https://sh32710.us-...">https://sh32710.us-...</a>
Image2.png.gz	4000	2022-01-25 14:24:31...	4650fe363e554e8...	4650fe363e554...	<a href="https://sh32710.us-...">https://sh32710.us-...</a>
Image3.png.gz	4352	2022-01-25 14:24:31...	b79de6e91ec7394...	b79de6e91ec73...	<a href="https://sh32710.us-...">https://sh32710.us-...</a>
Image4.png.gz	2112	2022-01-25 14:24:31...	e1ffe716036b98c3...	e1ffe716036b98...	<a href="https://sh32710.us-...">https://sh32710.us-...</a>

# REFRESH DIRECTORY TABLES

- Option 1: manual refresh (internal or external stages)
  - Refresh periodically with `ALTER STAGE <name> REFRESH;`
- Option 2: automatic refresh (external stages only)
  - Set up notifications through the cloud provider
  - Create stage with directory table and auto-refresh

```
CREATE STAGE <name>
  DIRECTORY = (ENABLE=true AUTO_REFRESH=true);
```



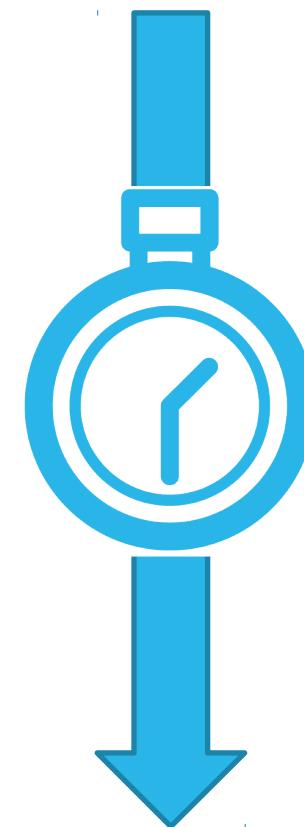
# ACCESS URLs

- URLs that provide access to the unstructured files on the stage
- Three types of access URLs:
  - Stage file URL
  - Scoped file URL
  - Pre-signed URL



# STAGE FILE URL

```
SELECT build_stage_file_url  
      (@<stage name>, '<path to  
file>');
```

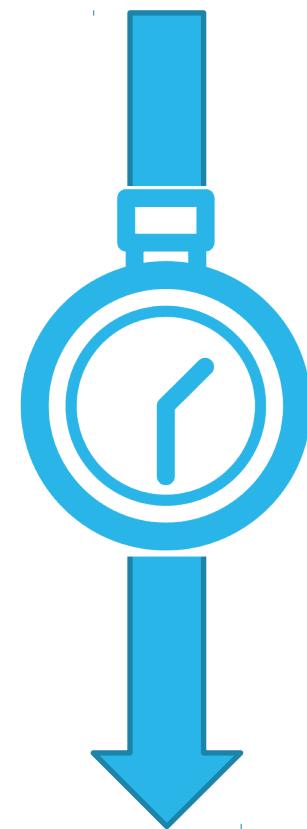


URL is permanent

- Permanent URL consists of database, schema, stage, and path to the file
- Access through Snowsight result set or via REST API
- Ideal for custom applications that need to access unstructured data files
- Providers who share data cannot share file URLs with consumers

# SCOPED FILE URL

```
SELECT build_scoped_file_url  
      (@<stage name>, '<path to  
file>');
```

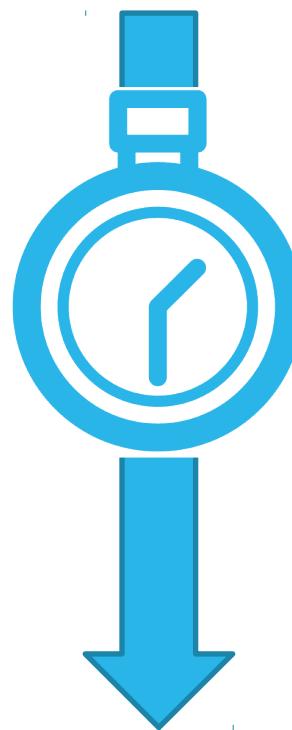


URL available for 24 hours

- Encoded URL permits temporary access (currently 24 hours) to files, without granting privileges to the stage
- Access through Snowsight result set or via REST API
- Provide access via a view that retrieves the scoped URL(s)
- Providers who share data can share scoped URLs in secure views

# PRE-SIGNED URL

```
SELECT get_presigned_URL  
(@<stage name>, '<path to file>'  
[, <time to expire>]);
```



URL available for set time  
(default is one hour)

- Temporary, pre-signed (open) URL that provides file access via a web browser
- Set user-defined expiration time
- Anyone with link can directly access or download the files
- Ideal for BI applications or reporting tools that need to display file contents
- Providers who share data can share pre-signed URLs in secure views

# SUMMARY OF ACCESS URLs

	<b>Stage File URL</b>	<b>Scoped File URL</b>	<b>Pre-Signed URL</b>
How long is it good for?	As long as the file exists	24 hours	User-defined expire time (default 1 hour)
How is it accessed?	Snowsight or REST API	Snowsight or REST API	Provided link
Who can access it?	Role with USAGE (external stage) or READ (internal stage) privileges	The user who generated it	Anyone with the link
Can it be used in data sharing?	No	Yes	Yes
Does it require read access to the stage?	Yes	No	No



# ENCRYPTION FOR INTERNAL STAGES

```
CREATE STAGE my_videos  
    directory = (enable=true)  
    encryption = (TYPE='SNOWFLAKE_SSE');
```



- SNOWFLAKE\_FULL
  - Client-side encryption
  - Files encrypted by client before PUT
- SNOWFLAKE\_SSE
  - Server-side encryption
  - Files encrypted when they arrive in stage
- Must use SSE for unstructured data files

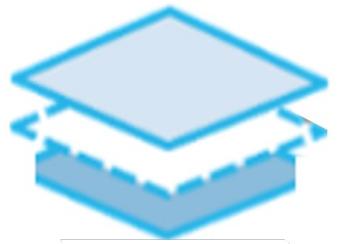


# WORKFLOW



# 1. CREATE A STAGE

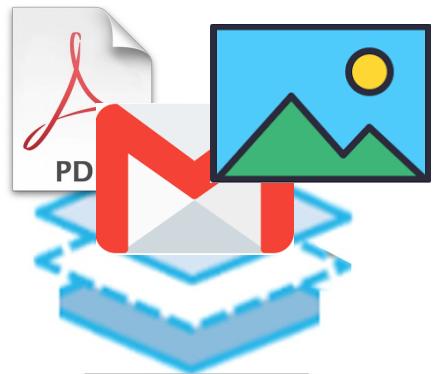
`CREATE STAGE . . .`



- Create an internal or external stage to hold the data files
  - For external stages, recommend using a storage integration
- What type of stage?
  - Use an external stage if you already have a large repository of files in the cloud
  - Use an internal stage if you want Snowflake to manage the files, encryption, etc.
- If using an internal stage, must set encryption to **SNOWFLAKE\_SSE**
- Recommended: enable directory tables

## 2. LOAD FILES TO STAGE

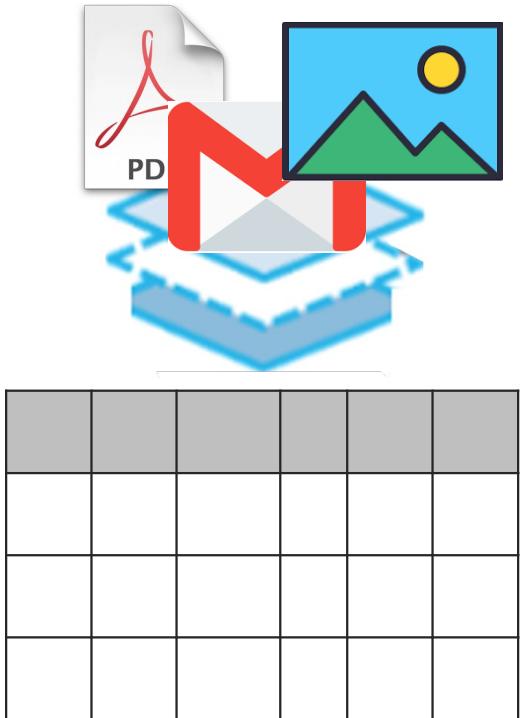
CREATE STAGE . . .



- For external stage, use applications or cloud provider tools
- For internal stage, use PUT command to move files from local system

# 3. REFRESH DIRECTORY TABLE

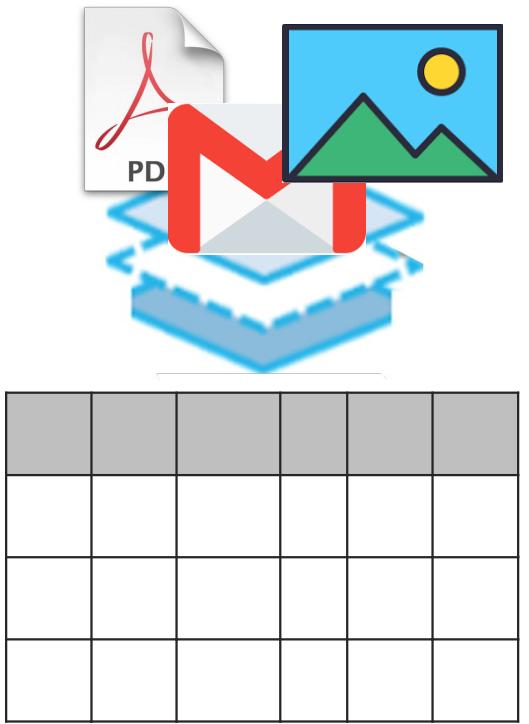
CREATE STAGE . . .



- By default, directory tables are refreshed when the stage is created
- External stages
  - Auto-refresh (AWS or Azure) with notification integration
  - Optionally refresh on schedule/as needed
- Internal stages
  - Refresh on a schedule or as needed

# 4. GRANT STAGE PRIVILEGES

`CREATE STAGE . . .`

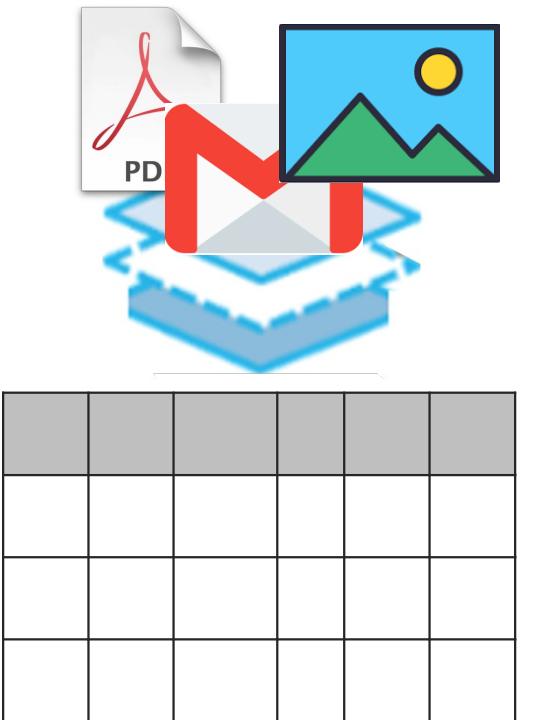


`GRANT READ ON STAGE . . .`

- Grant required privileges on the stage or integration
  - USAGE, READ, and/or WRITE

# 5. GENERATE ACCESS URLs

CREATE STAGE . . .



GRANT READ ON STAGE . . .

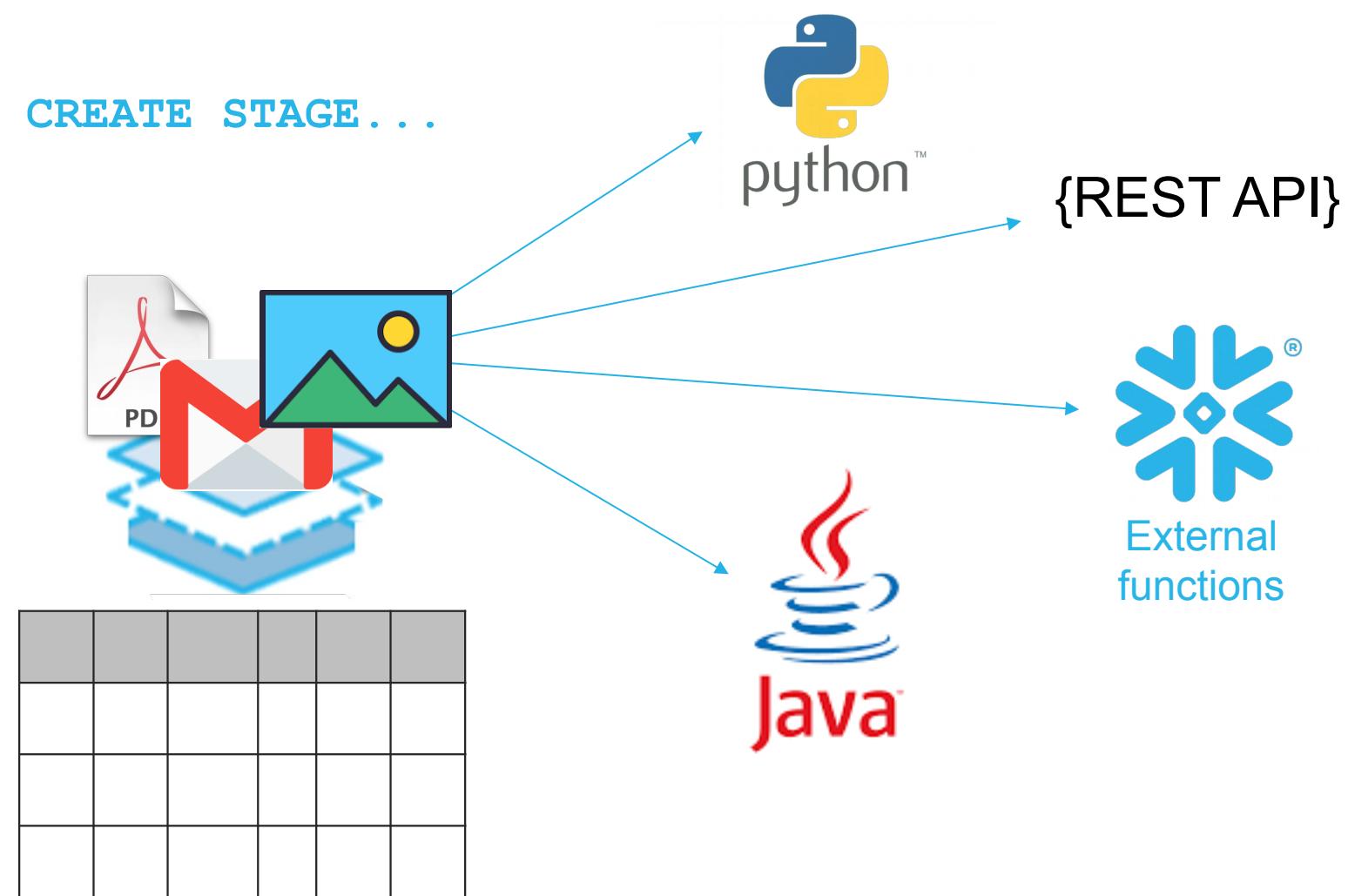
SELECT build\_stage\_file\_url . . .

SELECT build\_scoped\_file\_url . . .

SELECT get\_presigned\_URL

- Stage File URL
  - Permanent
  - Requires READ access to stage
- Scoped File URL
  - Good for 24 hours
  - Does not require READ access to stage
- Pre-signed URL
  - User-defined expiry time (default 1 hour)
  - Does not require READ access to stage

# 6. ACCESS/PROCESS DATA



GRANT READ ON STAGE...

```
SELECT build_stage_file_url...
SELECT build_scoped_file_url...
SELECT get_presigned_URL...
```

- Unstructured data can be accessed for processing or analysis by several means

# BEST PRACTICES

- Use a storage integration for an external stage
  - Increased security
- Create scoped file URLs for enhanced governance
  - Limit the time the scoped file URL is valid
- Refresh directory tables
  - With external stage, use auto-refresh
  - With internal stage, schedule refreshes with a task
- Use secure views to share unstructured data



# LAB EXERCISE: 7

## Use Unstructured Data

20 minutes

Snowbear Air has decided to offer members a set of free luggage tags with their picture on them, if they will send in a picture that the airline can store as part of their members data.

In this lab you will create a test members table and create a stage that will be used to hold the picture data. Once the picture data has been uploaded, you will create view of the members table that includes a link to the picture.

### Learning Objectives:

- Set up a development environment
- Access unstructured data



# WORKING WITH DATA LAKES



# MODULE AGENDA

- Data Lake Deployments
- Querying External Data Lakes
- External Tables
- Partitioning External Tables

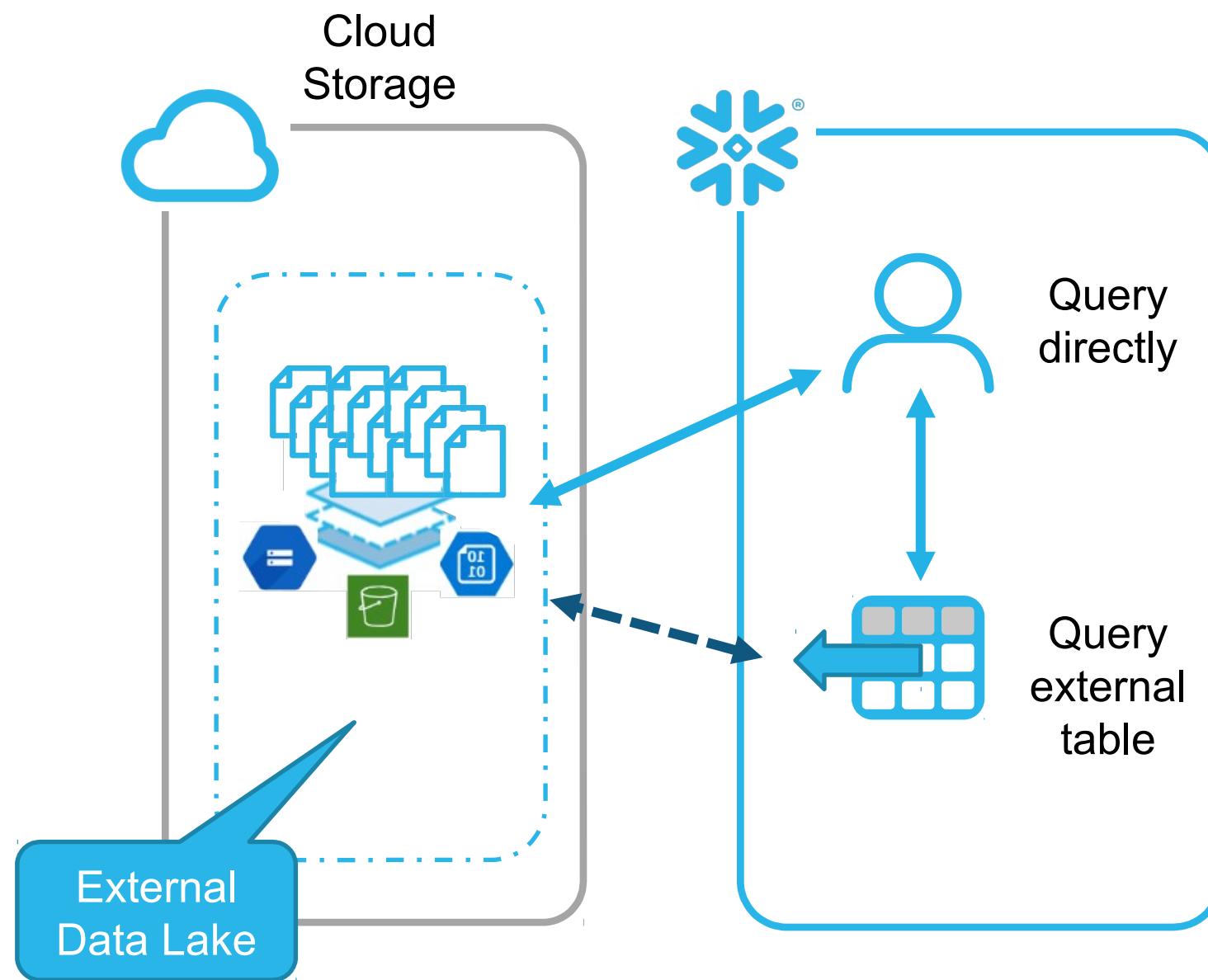


# DATA LAKE DEPLOYMENTS

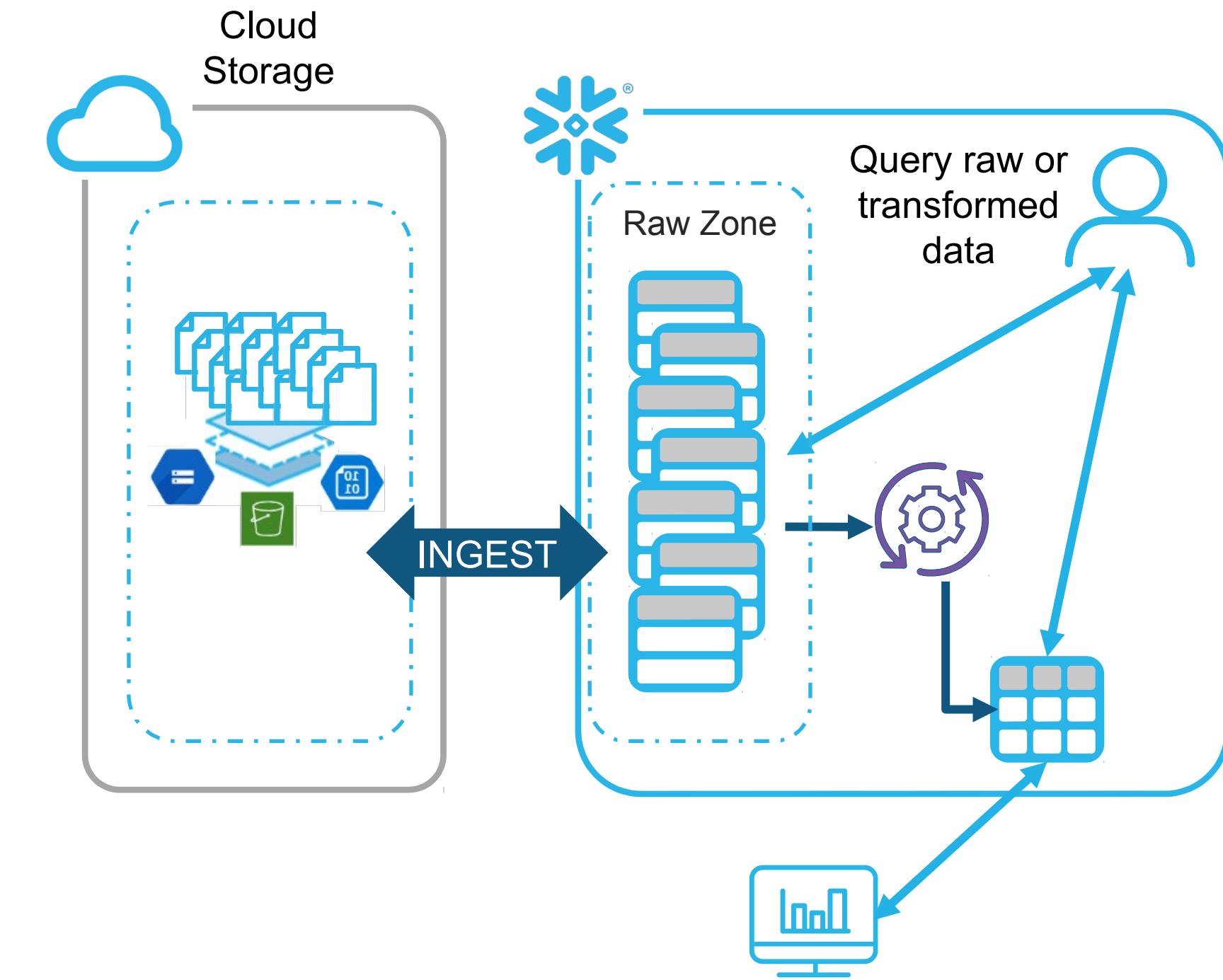


# DATA LAKE OPTIONS

## External Data Lake



## Data Lake in Snowflake



# DATA LAKE : EXTERNAL OR SNOWFLAKE

Single Source of Truth	Data Format	Comments
<b>External Data Lake</b>	Typically Semi-Structured Files	<ul style="list-style-type: none"><li>• Best option for customers with an existing data lake</li><li>• External data read with Snowflake virtual warehouse</li></ul>
<b>Snowflake Data Lake</b>	VARIANT or Structured Tables	<ul style="list-style-type: none"><li>• Benefits from data compression and query performance</li><li>• Single data governance and technology platform for all data</li></ul>

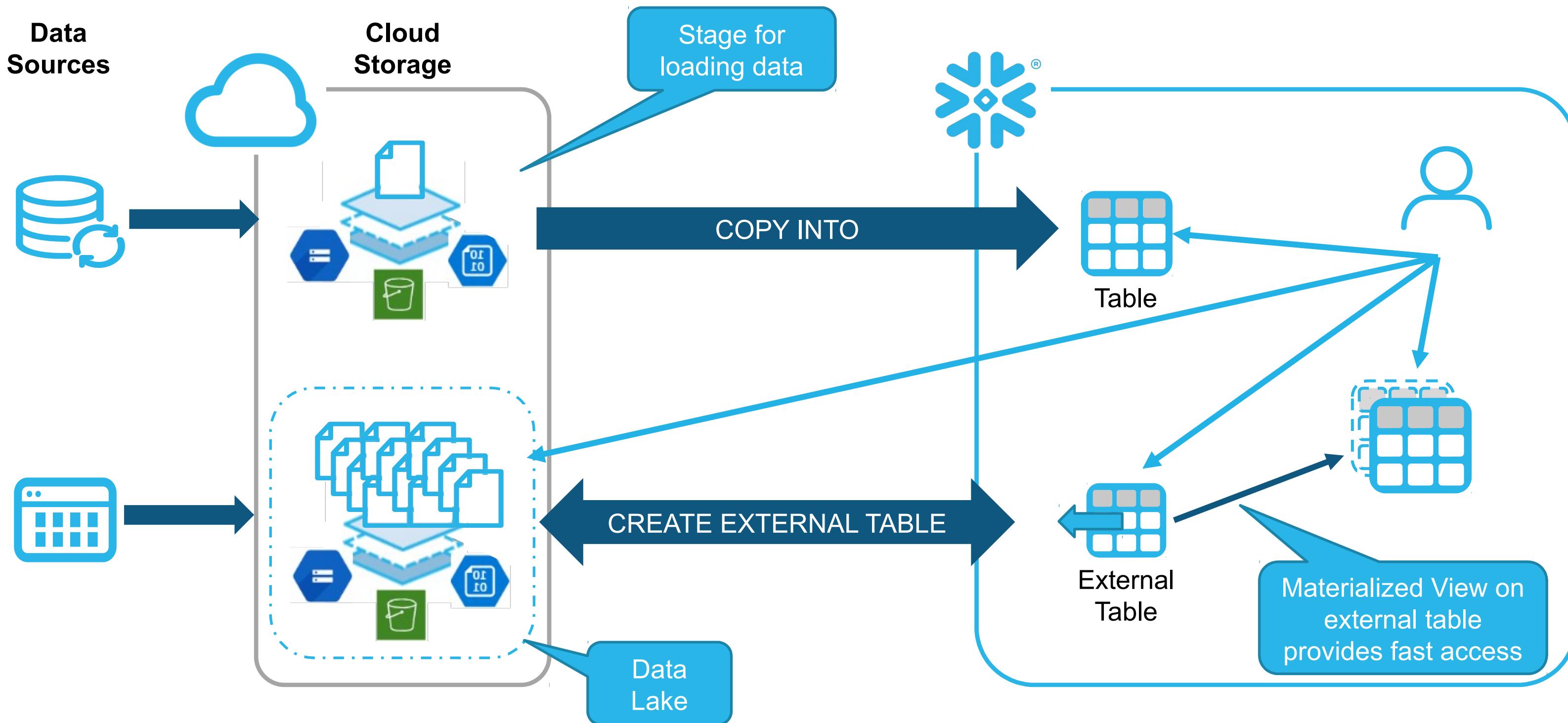


# DATA LAKE : EXTERNAL OR SNOWFLAKE

Single Source of Truth	Data Format	Comments
<b>External Data Lake</b>	Typically Semi-Structured Files	<ul style="list-style-type: none"><li>• Best option for customers with an existing data lake</li><li>• External data read with Snowflake virtual warehouse</li></ul>
<b>Snowflake Data Lake</b>	VARIANT or Structured Tables	<ul style="list-style-type: none"><li>• Single data governance and technology platform for all data</li><li>• Benefits from data compression and query performance</li></ul>
Alternative	Comments	
<b>Snowflake as Transformation Engine</b>	<ul style="list-style-type: none"><li>• Raw data files loaded and transformed in Snowflake and extracted to hydrate cloud data lake</li><li>• Benefits of single transformation technology and cloud-based data lake</li></ul>	



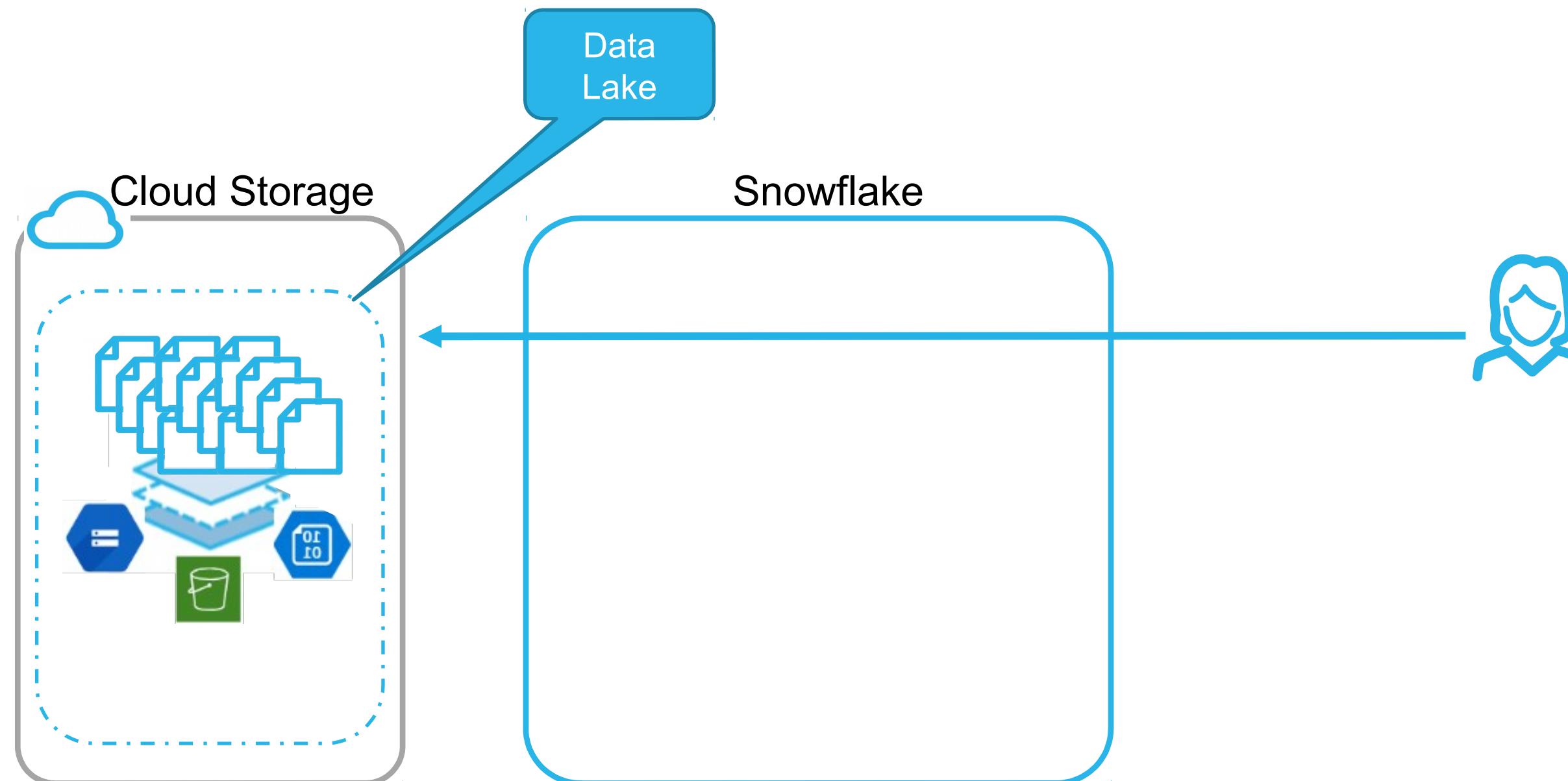
# INTEGRATE CLOUD-BASED DATA LAKE



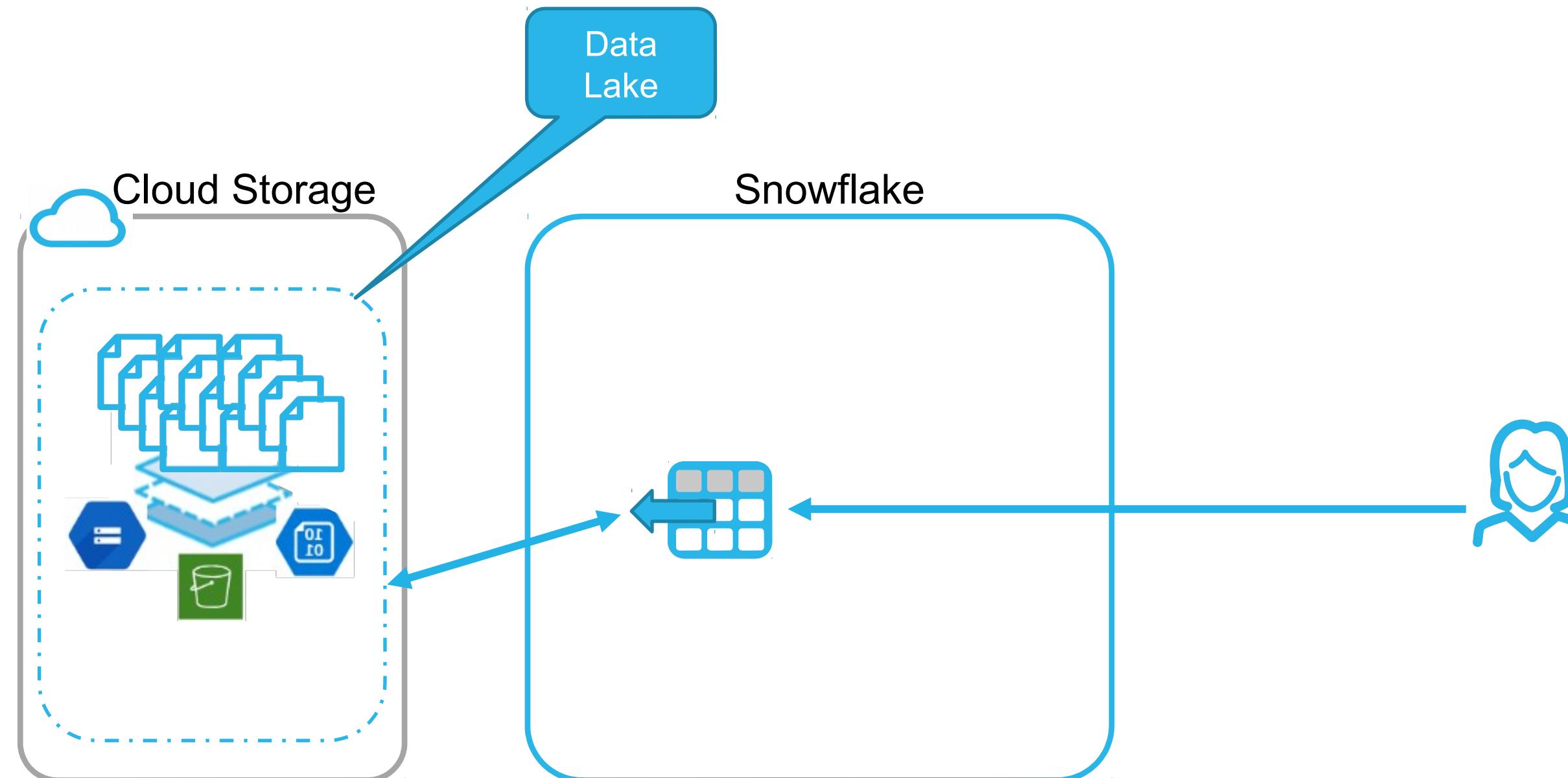
# QUERYING EXTERNAL DATA LAKES



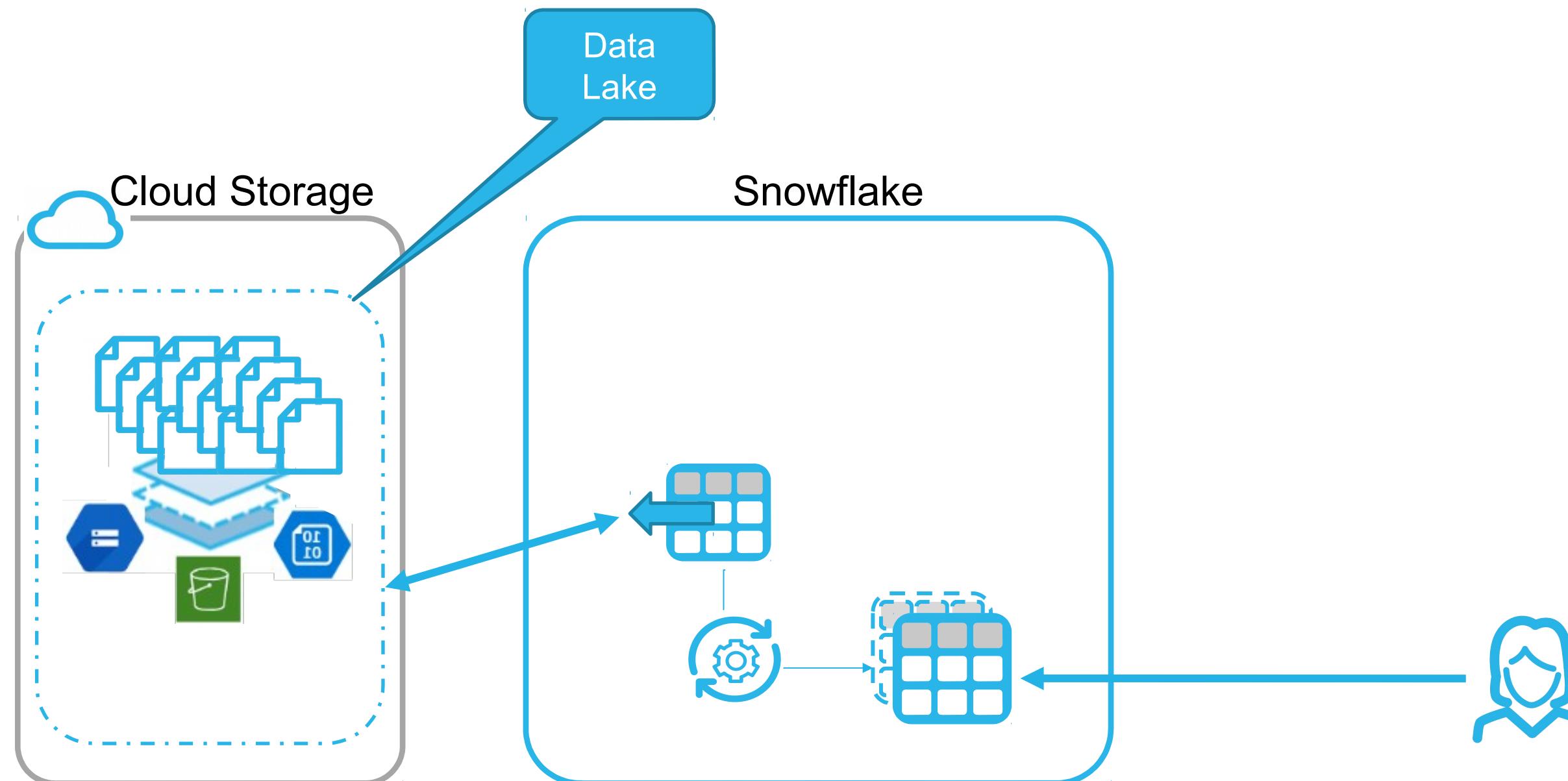
# OPTION 1: QUERY EXTERNAL DATA DIRECTLY



# OPTION 2: CREATE EXTERNAL TABLE

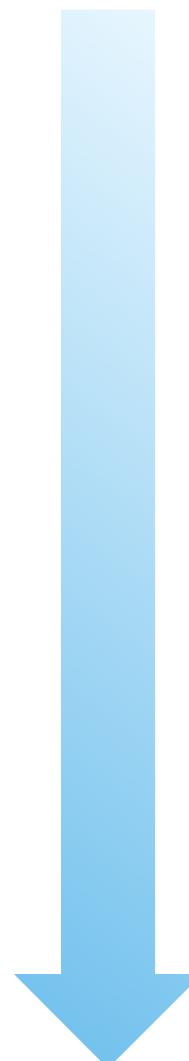


# OPTION 3: ADD MATERIALIZED VIEW



# DATA LAKE ACCESS SUMMARY

SLOWEST



Query	Format	Partition Pruning	Schema
<b>Directly</b>	Semi-Structured Files	None	Query time
<b>External Tables</b>	Semi-Structured Files	Coarse-grained, based on file path	Either query time or View over External Table
<b>External Tables + MViews</b>	MView or Semi-Structured Files	Fine, based on micro-partitions	Either query time or View over External Table

FASTEST



# CREATING EXTERNAL TABLES



# CREATE EXTERNAL TABLE

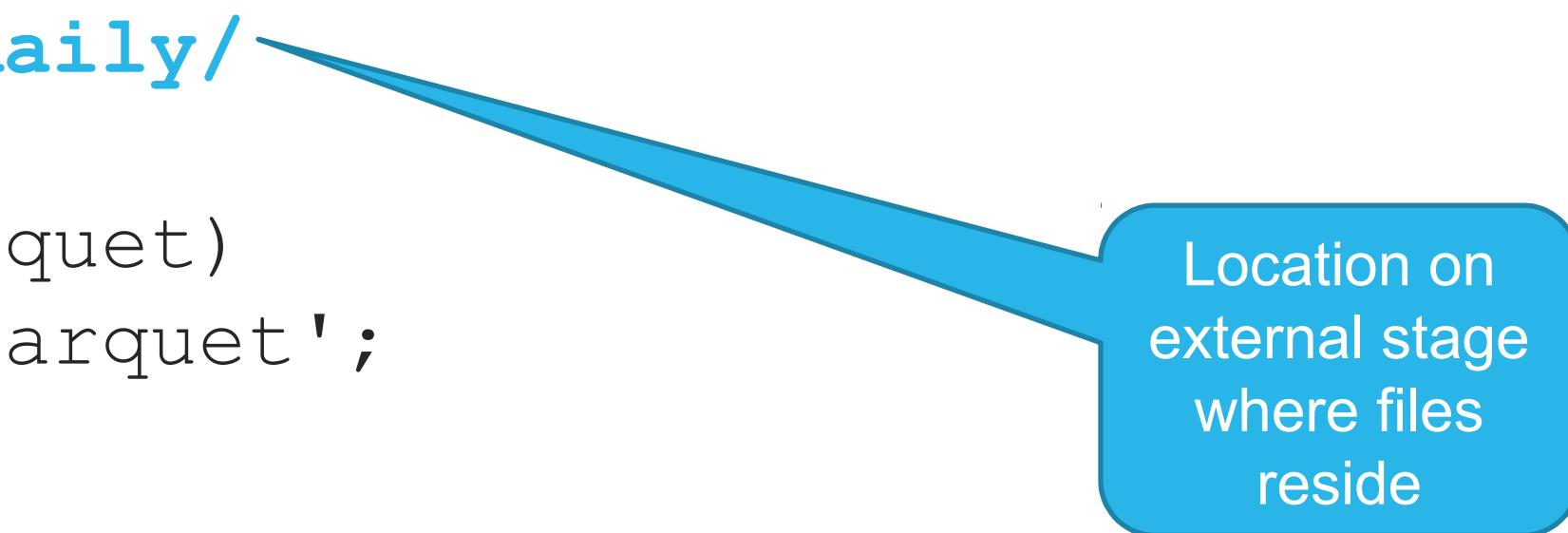
```
CREATE EXTERNAL TABLE twitter_feed_ext WITH  
LOCATION = @twitter_stg/daily/  
AUTO_REFRESH = true  
FILE_FORMAT = (TYPE = parquet)  
PATTERN = '.*twitter.*[.]parquet';
```

Table  
name



# CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE twitter_feed_ext WITH  
  LOCATION = @twitter_stg/daily/  
  AUTO_REFRESH = true  
  FILE_FORMAT = (TYPE = parquet)  
  PATTERN = '.*twitter.*[.]parquet';
```



Location on  
external stage  
where files  
reside



# CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE twitter_feed_ext WITH  
LOCATION = @twitter_stg/daily/  
AUTO_REFRESH = true  
FILE_FORMAT = (TYPE = parquet)  
PATTERN = '.*twitter.*[.]parquet';
```

Set up notifications on cloud provider; Snowflake notified when something changes



# CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE twitter_feed_ext WITH  
  LOCATION = @twitter_stg/daily/  
  AUTO_REFRESH = true  
FILE_FORMAT = (TYPE = parquet)  
  PATTERN = '.*twitter.*[.]parquet';
```

File format



# CREATE EXTERNAL TABLE

```
CREATE EXTERNAL TABLE twitter_feed_ext WITH  
LOCATION = @twitter_stg/daily/  
AUTO_REFRESH = true  
FILE_FORMAT = (TYPE = parquet)  
PATTERN = '.*twitter.*[.]parquet';
```

Regular expression  
defining files to  
include



# QUERY EXTERNAL TABLE

- External tables behave like read-only Snowflake tables, and can be queried the same way

```
SELECT *
FROM twitter_feed_ext;
```

- Query performance may be poor
  - Without partitioning or a pattern to limit files, a query will scan EVERY file on the stage
- Queries to external tables will use Query Result Cache or Data Cache if possible
  - If auto-refresh is disabled, and metadata is not refreshed manually, outdated results may be returned from cache



# EXTERNAL TABLE STRUCTURE

VALUE	METADATA\$FILENAME
<row information>	<file name>

- Every external table contains three columns:
  - VALUE – a VARIANT column that represents a single row in the external file
  - METADATA\$FILENAME
  - METADATA\$FILE\_ROW\_NUMBER
- You can create additional virtual columns based on the VALUE column or the METADATA\$FILENAME column



# PARTITIONING EXTERNAL TABLES



# EXTERNAL TABLE PARTITIONS

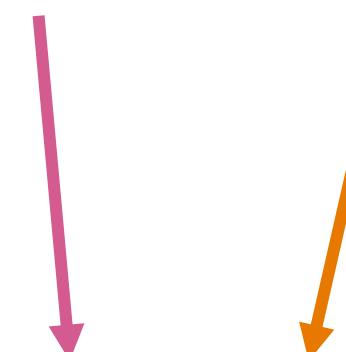
- Based on the file path
  - **Important** to have standard structure/naming convention for external table files
- Without partitions, Snowflake must query every file on the data lake
  - Slow and costly



# PARTITION AN EXTERNAL TABLE

```
CREATE EXTERNAL TABLE twitter_feed_ext WITH
    country_code  VARCHAR AS TO_DATE(SPLIT_PART(metadata$filename,'/',4)),
    year          NUMBER AS TO_DATE(SPLIT_PART(metadata$filename,'/',5))
LOCATION = @twitter_stg/daily/
AUTO_REFRESH = true
FILE_FORMAT = (TYPE = csv)
PATTERN = '.*twitter.*[.]csv'
PARTITION BY (country_code, year);
```

File names:



```
s3://twitter/us/2022/04/14/tweet_000.csv
s3://twitter/uk/2022/04/14/tweet_000.csv
s3://twitter/fr/2022/04/14/tweet_000.csv
```

5<sup>th</sup> field of file  
name, with '/' as  
the field delimiter



# MATERIALIZED VIEW ON EXTERNAL TABLE

- Create a materialized view for frequently accessed data
- Very fast access for queries against the materialized view, while less frequent queries can still run against the external table.
- Create multiple materialized views to support several high-use queries

```
CREATE MATERIALIZED VIEW twitter_feed_uk AS
```

```
SELECT * FROM twitter_feed_ext  
WHERE country_code = 'UK'  
AND year = 2022;
```



# MATERIALIZED VIEW ON EXTERNAL TABLE

- Example of multiple materialized views
- Automatic query rewrite will use the most appropriate source

```
CREATE MATERIALIZED VIEW twitter_feed_uk AS  
SELECT * FROM twitter_feed_ext  
WHERE country_code = 'UK'  
AND year = 2022;
```

```
CREATE MATERIALIZED VIEW twitter_feed_fr AS  
SELECT * FROM twitter_feed_ext  
WHERE country_code = 'FR';
```



# REFRESH EXTERNAL TABLES

```
CREATE EXTERNAL TABLE...  
REFRESH_ON_CREATE = TRUE;
```

```
CREATE EXTERNAL TABLE...  
AUTO_REFRESH = TRUE;
```

```
ALTER EXTERNAL TABLE...  
REFRESH
```



- Refresh on create
  - Default is set to TRUE
- Refresh automatically (recommended)
  - Set up notifications on cloud provider
  - Notification sent when anything changes
  - External table is refreshed
- Refresh manually
  - As needed, or on a schedule
  - Uses tasks to schedule

# LAB EXERCISE: 8

## Work with External Tables

35 minutes

- Unloading Data to Cloud Storage as a Data Lake
- Unload Date to Cloud Storage Using Different Virtual Warehouses
- Executing Queries Against External Tables
- Working with External Tables
- Create a Partitioned External Table



# DATA GOVERNANCE IN SNOWFLAKE



# MODULE AGENDA

- Data Governance Overview
- Dynamic Data Masking
- External Tokenization
- Row Access Policies
- Classification and Object Tagging
- Audit Access History



# DATA GOVERNANCE OVERVIEW



# WHAT IS DATA GOVERNANCE?



- Defines who within an organization has control over sensitive data
  - Creates policies to ensure that sensitive data within your organization is protected
  - Ensures data adheres to Regulatory Compliance
    - GDPR, FedRAMP, PCI, and HIPAA

# SNOWFLAKE DATA GOVERNANCE CAPABILITIES

KNOW, PROTECT, AND AUDIT YOUR DATA



**Know your data**

Classification

Object Tagging



**Protect your data**

Dynamic Data  
Masking

External  
Tokenization

Row Access Policies



**Audit your data**

Account Usage

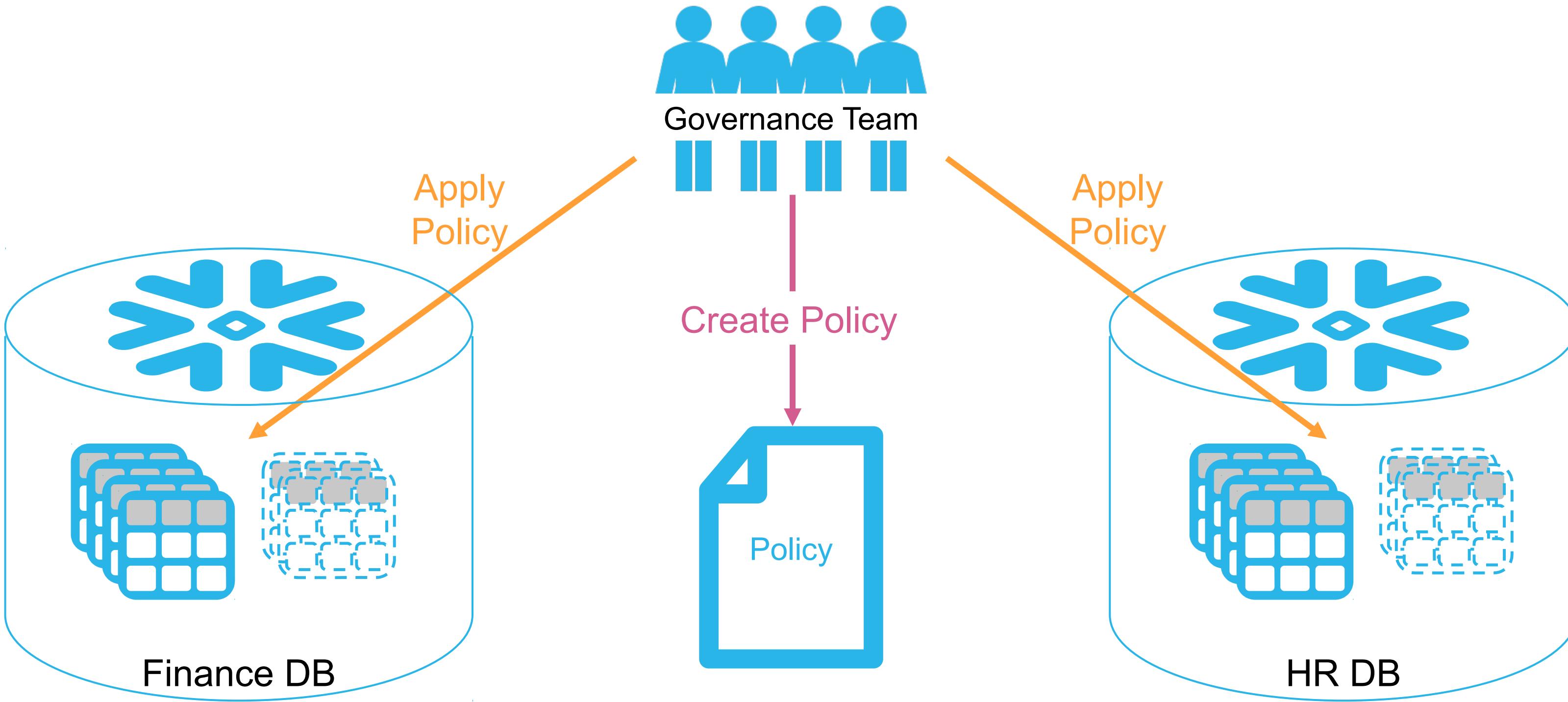
Access History

# POLICY-DRIVEN APPROACH

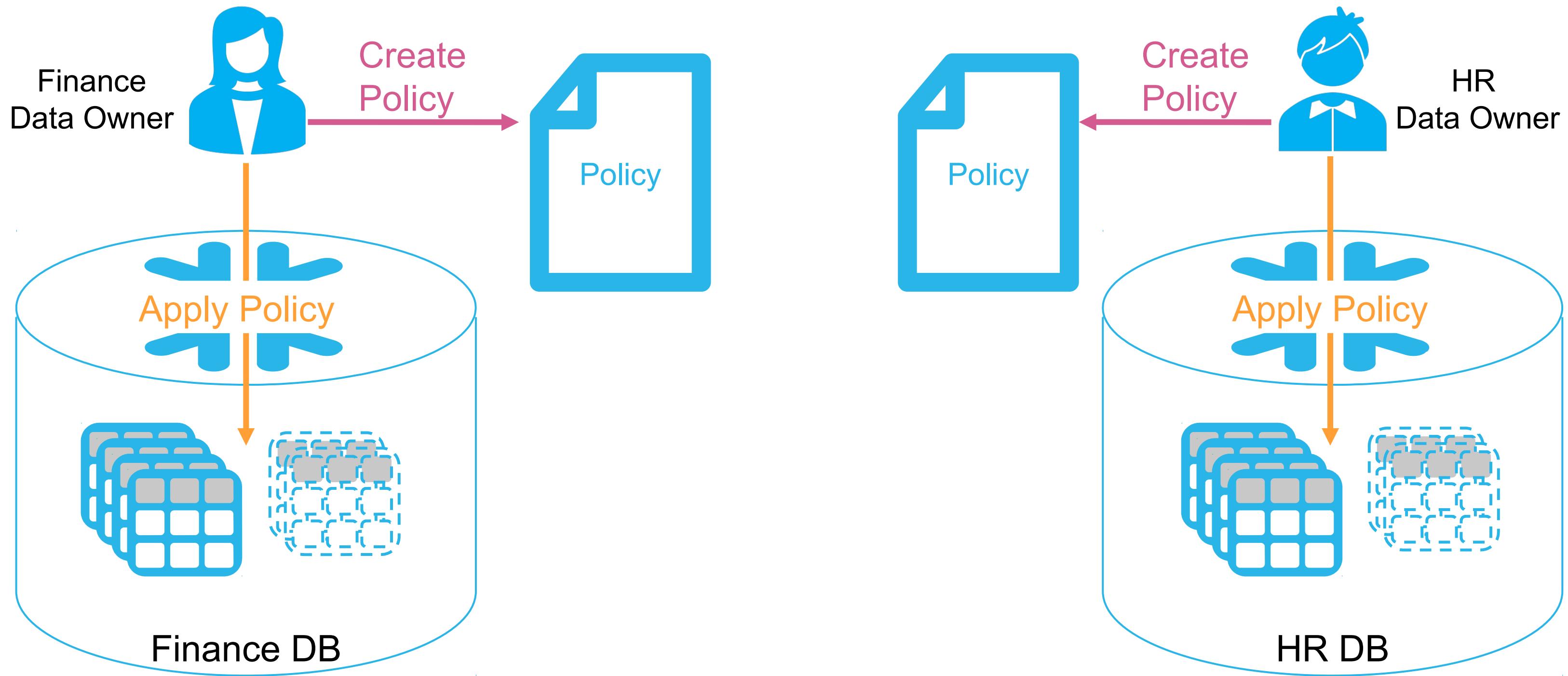
- A policy-driven approach facilitates separation of duties
  - A policy defines who can see what part of the data
  - A single policy can be set on multiple tables and views
- Snowflake provides industry-leading features to enable creation, deployment, and management of your data governance policy:
  - Object Tagging
  - Data masking and tokenization
  - Row access policies
- Determine your policy management approach
  - Centralized, Decentralized, or Hybrid



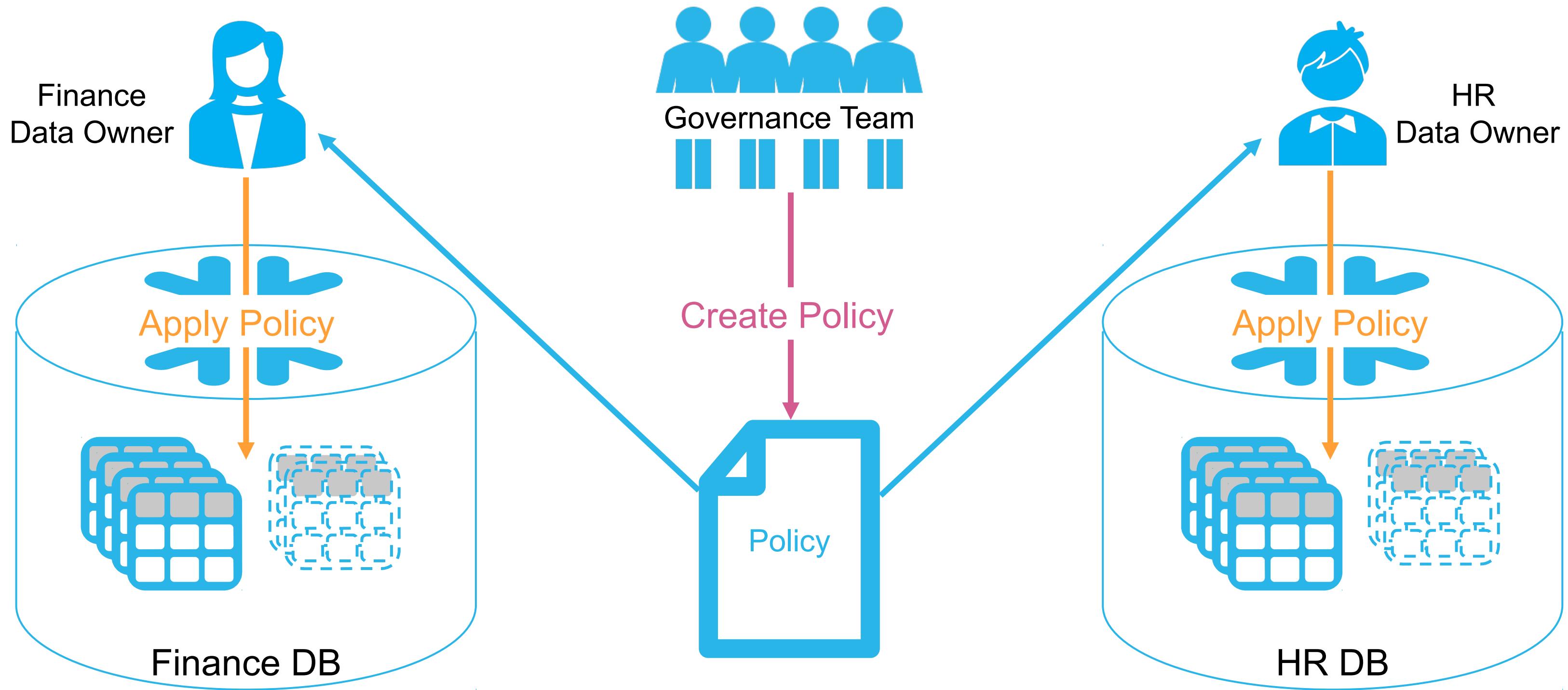
# CENTRALIZED POLICY MANAGEMENT



# DECENTRALIZED POLICY MANAGEMENT



# HYBRID POLICY MANAGEMENT



# DYNAMIC DATA MASKING



# DYNAMIC DATA MASKING

## COLUMN-LEVEL SECURITY

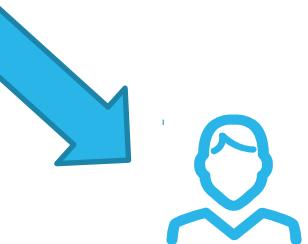
Name	Gender	Age	Zip Code	Phone
John Smith	Male	39	79007	123-555-1234
Jane Doe	Female	50	77001	333-555-1236
Mary Taylor	non-binary	27	77020	222-333-1111



Masking Policy



Unauthorized



Partially  
Authorized

Name	Phone
*****	***-***-5534
*****	***-***-3564
*****	***-***-9787

Name	Phone
*****	408-123-5534
*****	510-335-3564
*****	214-553-9787

- Protects access to PII and other sensitive data at the column level
  - Role-based access control (RBAC) limits access only at the object level
- Can exclude ACCOUNTADMIN and data owner from access
  - RBAC cannot
- Granular control builds upon RBAC
- Dynamically applied at run-time

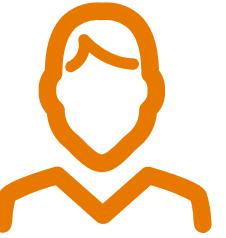
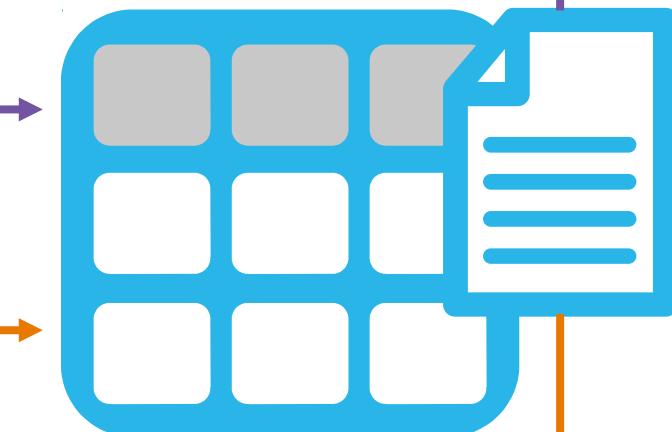
# HOW DYNAMIC DATA MASKING WORKS



Bob

ID	PHONE	SSN
101	***-***-5534	*****
102	***-***-3564	*****
103	***-***-9787	*****

`SELECT ID, PHONE, SSN FROM CUSTOMER;`



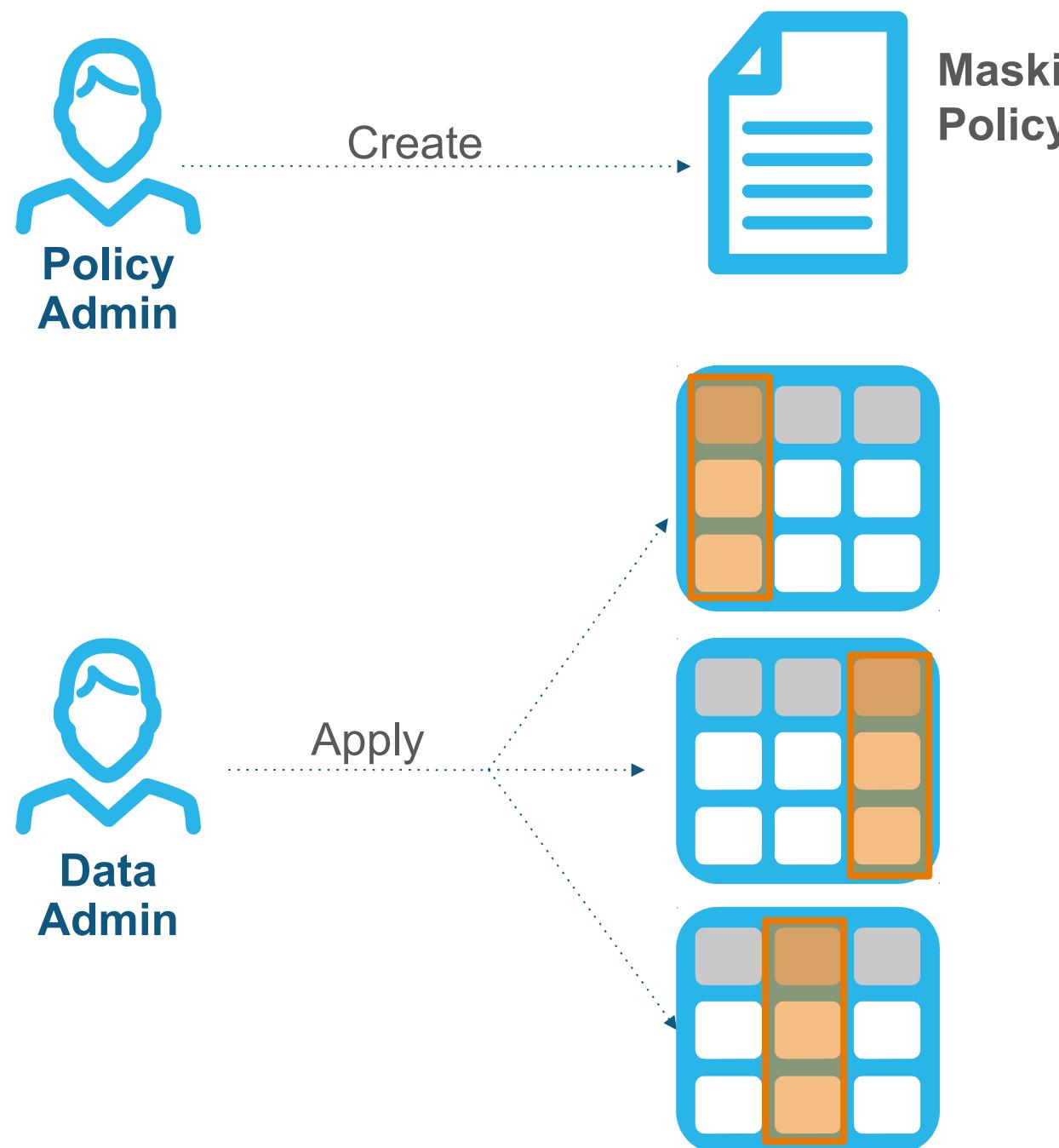
Alice

ID	PHONE	SSN
101	408-123-5534	*****6903
102	510-335-3564	*****1008
103	214-553-9787	*****1870

`SELECT ID, PHONE, SSN FROM CUSTOMER;`

# MASKING POLICIES

## SEPARATION OF DUTIES



- Create a masking policy that defines the rules for masking
- Apply to one or more tables, views, or external table columns
- Clones, data shares, and replication will propagate the policy

# POLICY EXAMPLE

## MASK NAME FROM UNAUTHORIZED USERS

- Mask entries unless PII\_AUTHORISED role is in user's hierarchy

Raw value in the  
masked column

```
CREATE MASKING POLICY pii_name_mask AS
(val STRING) RETURNS STRING ->
CASE
    WHEN IS_ROLE_IN_SESSION('PII_AUTHORISED') THEN val
    ELSE '*** Redacted ***'
END;
```



# POLICY EXAMPLE

## MASK NAME FROM UNAUTHORIZED USERS

- Mask entries unless PII\_AUTHORISED role is in user's hierarchy

```
CREATE MASKING POLICY pii_name_mask AS  
(val STRING) RETURNS STRING ->  
CASE  
    WHEN IS_ROLE_IN_SESSION('PII_AUTHORISED') THEN val  
    ELSE '*** Redacted ***'  
END;
```

When the role  
PII\_AUTHORISED is  
in the user's hierarchy,  
actual value is returned



# POLICY EXAMPLE

## MASK NAME FROM UNAUTHORIZED USERS

- Mask entries unless PII\_AUTHORISED role is in user's hierarchy

```
CREATE MASKING POLICY pii_name_mask AS  
(val STRING) RETURNS STRING ->  
    CASE  
        WHEN IS_ROLE_IN_SESSION('PII_AUTHORISED') THEN val  
        ELSE '*** Redacted ***'  
    END;
```



Else, masked  
value is returned

# POLICY EXAMPLE

## VARY MASK BASED ON ROLE

Raw value in the  
masked column

```
CREATE MASKING POLICY pii_name_mask AS
(val STRING) RETURNS STRING ->
CASE
    WHEN CURRENT_ROLE() IN ('SUPPORT')
        THEN REGEX_REPLACE(val, '^.{6}', '*****')
    WHEN CURRENT_ROLE() IN ('SYSADMIN') THEN mask_name_udf(val)
    ELSE '*** Redacted ***'
END;
```



# POLICY EXAMPLE

## VARY MASK BASED ON ROLE

```
CREATE MASKING POLICY pii_name_mask AS  
(val STRING) RETURNS STRING ->  
CASE  
    WHEN CURRENT_ROLE() = 'SUPPORT'  
        THEN REGEX_REPLACE(val, '^.{6}', '*****')  
    WHEN CURRENT_ROLE() = 'SYSADMIN' THEN mask_name_udf(val)  
    ELSE '*** Redacted ***'  
END;
```

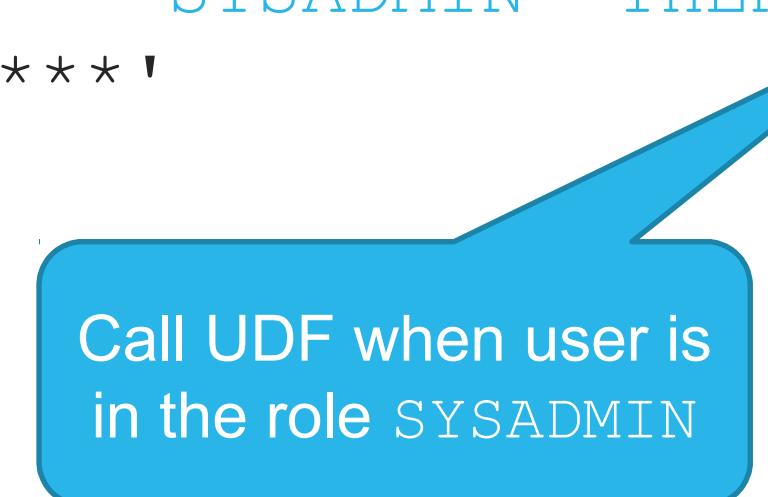
Apply regular expression when user is in the role SUPPORT



# POLICY EXAMPLE

## VARY MASK BASED ON ROLE

```
CREATE MASKING POLICY pii_name_mask AS
(val STRING) RETURNS STRING ->
CASE
    WHEN CURRENT_ROLE() = 'SUPPORT'
        THEN REGEX_REPLACE(val, '^.{6}', '*****')
    WHEN CURRENT_ROLE() = 'SYSADMIN' THEN mask_name_udf(val)
    ELSE '*** Redacted ***'
END;
```



Call UDF when user is  
in the role SYSADMIN

# POLICY EXAMPLE

## VARY MASK BASED ON ROLE

```
CREATE MASKING POLICY pii_name_mask AS
(val STRING) RETURNS STRING ->
CASE
    WHEN CURRENT_ROLE() IN ('SUPPORT')
        THEN REGEX_REPLACE(val, '^.{6}', '*****')
    WHEN CURRENT_ROLE() IN ('SYSADMIN') THEN mask_name_udf(val)
    ELSE '*** Redacted ***'
END;
```



Else, see fully  
masked value

# CONDITIONAL MASKING

USER_NAME	PRIVATE	EMAIL
J_SIMMONS	YES	*** MASKED***
C_BELLINGER	NO	cbellinger@d5.com
A_ZELLER	YES	*** MASKED***
B_WHITE	YES	*** MASKED***
J_PERKINS	NO	jperk321@zorn.com

- Mask data in one column based upon the value in another column
- Some column values are masked and others remain unmasked

```
CREATE MASKING POLICY mask_private_email AS
(email STRING, private STRING) RETURNS STRING ->
CASE
    WHEN CURRENT_ROLE() = 'SECURE_ROLE' THEN email
    WHEN private = 'NO' THEN email
    ELSE '*** MASKED ***'
END;
```



# ONE POLICY, MANY COLUMNS

- Apply one policy to multiple objects, or to multiple columns in the same object

```
ALTER TABLE customer
    MODIFY column cust_email
    SET MASKING POLICY email_mask;
```

```
ALTER TABLE customer
    MODIFY column secondary_email
    SET MASKING POLICY email_mask;
```

```
ALTER TABLE supplier
    MODIFY COLUMN supplier_email
    SET MASKING POLICY email_mask;
```



# SHOW MASKING POLICIES

- Restricted to:
  - Masking policy OWNER
  - Role with APPLY privilege on masking policy
- Examples:
  - SHOW MASKING POLICIES;
  - SHOW MASKING POLICIES LIKE '%email%';
  - SHOW MASKING POLICIES IN ACCOUNT;
  - SHOW MASKING POLICIES LIKE '%email%' IN DATABASE taxpayer\_db;



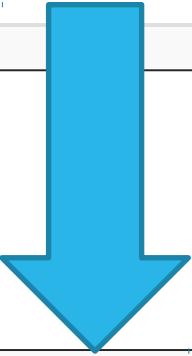
# DESCRIBE MASKING POLICY

14 | DESCRIBE MASKING POLICY email\_mask;

Objects    Query    Results    Chart

	name	signature	return_type	body
1	EMAIL_MASK	(VAL VARCHAR)	VARCHAR(16777216)	CASE WHEN CURRENT_ROLE() = 'SYSADMIN' THEN val WHEN

Click in body column to show definition



body	Aa body
CASE WHEN CURRENT_ROLE() = 'SYSAD'	CASE WHEN CURRENT_ROLE() = 'SYSADMIN' THEN val WHEN CURRENT_ROLE() = 'TRAINING_ROLE' THEN REGEXP_REPLACE(val,'.+@','*****@') ELSE '*** UNAUTHORIZED ***' END



# LAB EXERCISE: 9

## Use Dynamic Data Masking

30 minutes

Data scientists at Snowbear Air want to use membership data to develop an analytic application. Some of the tables related to membership contain sensitive data or personally identifiable information (PII).

To address this requirement, you will create a development environment and then build dynamic data masking policies that protect sensitive data in the `MEMBERS` table from unauthorized access. You will then view and test the policies.

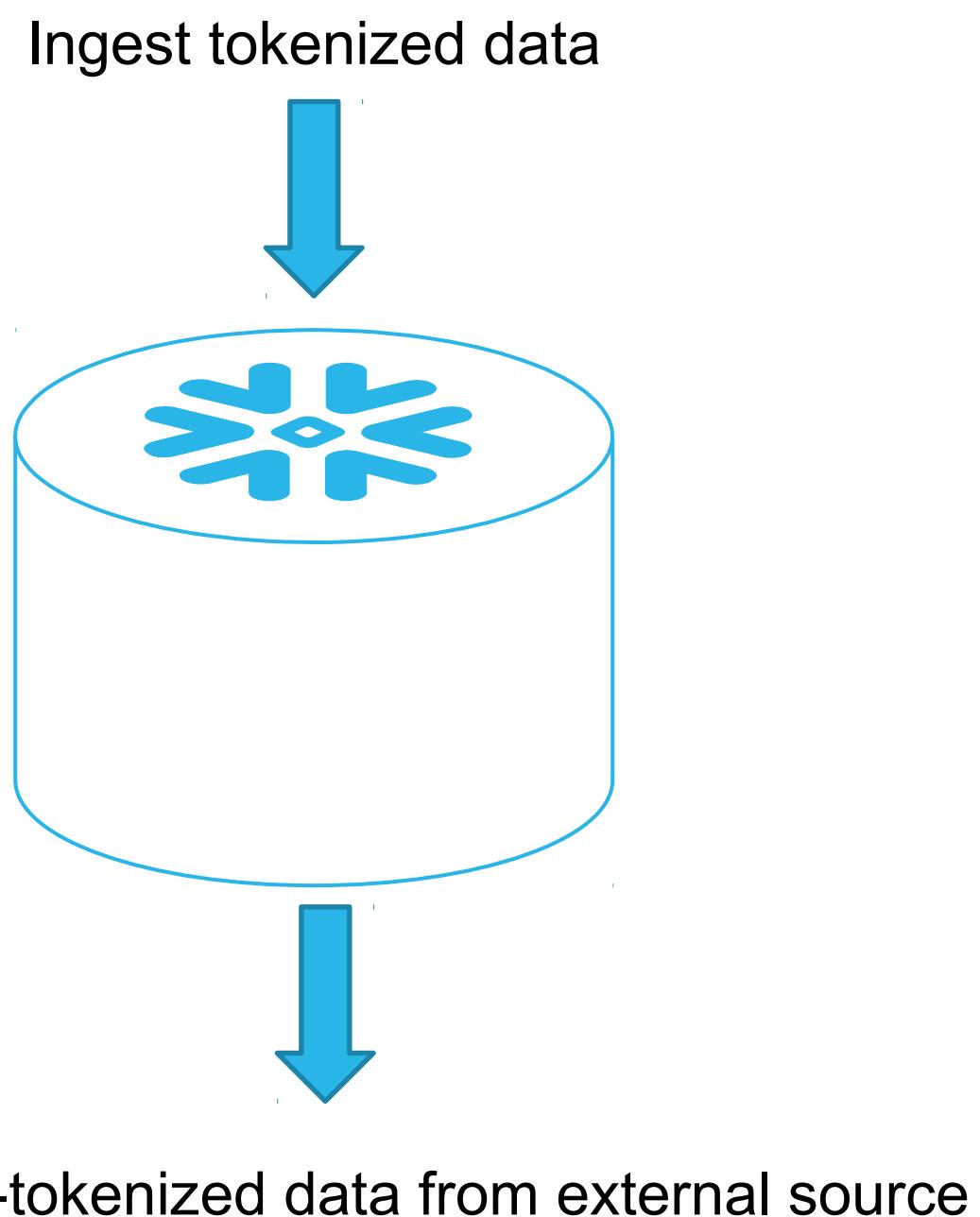


# EXTERNAL TOKENIZATION



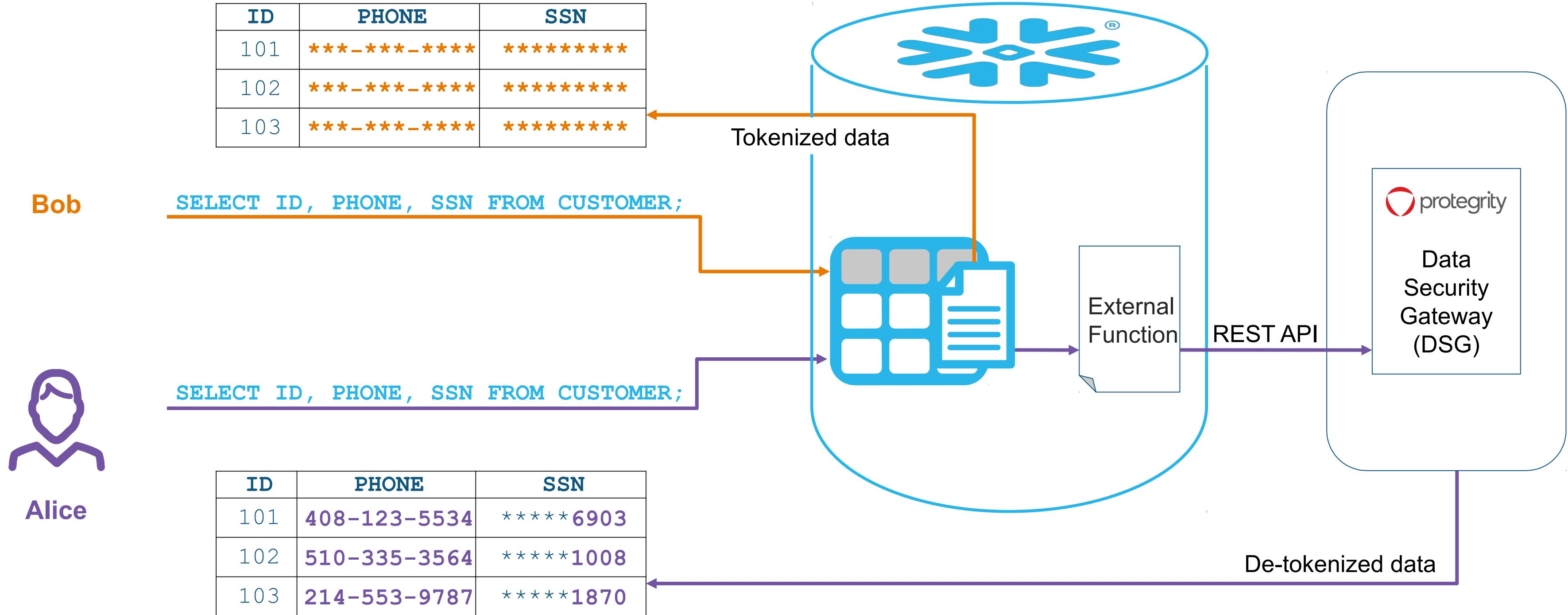
# EXTERNAL TOKENIZATION

## TOKENIZE DATA STORED IN SNOWFLAKE



- Actual data is never stored in Snowflake
- Policies determine who can view the de-tokenized data
- When needed, external function is used to retrieve de-tokenized data from provider

# HOW EXTERNAL TOKENIZATION WORKS



# DATA MASKING VS TOKENIZATION

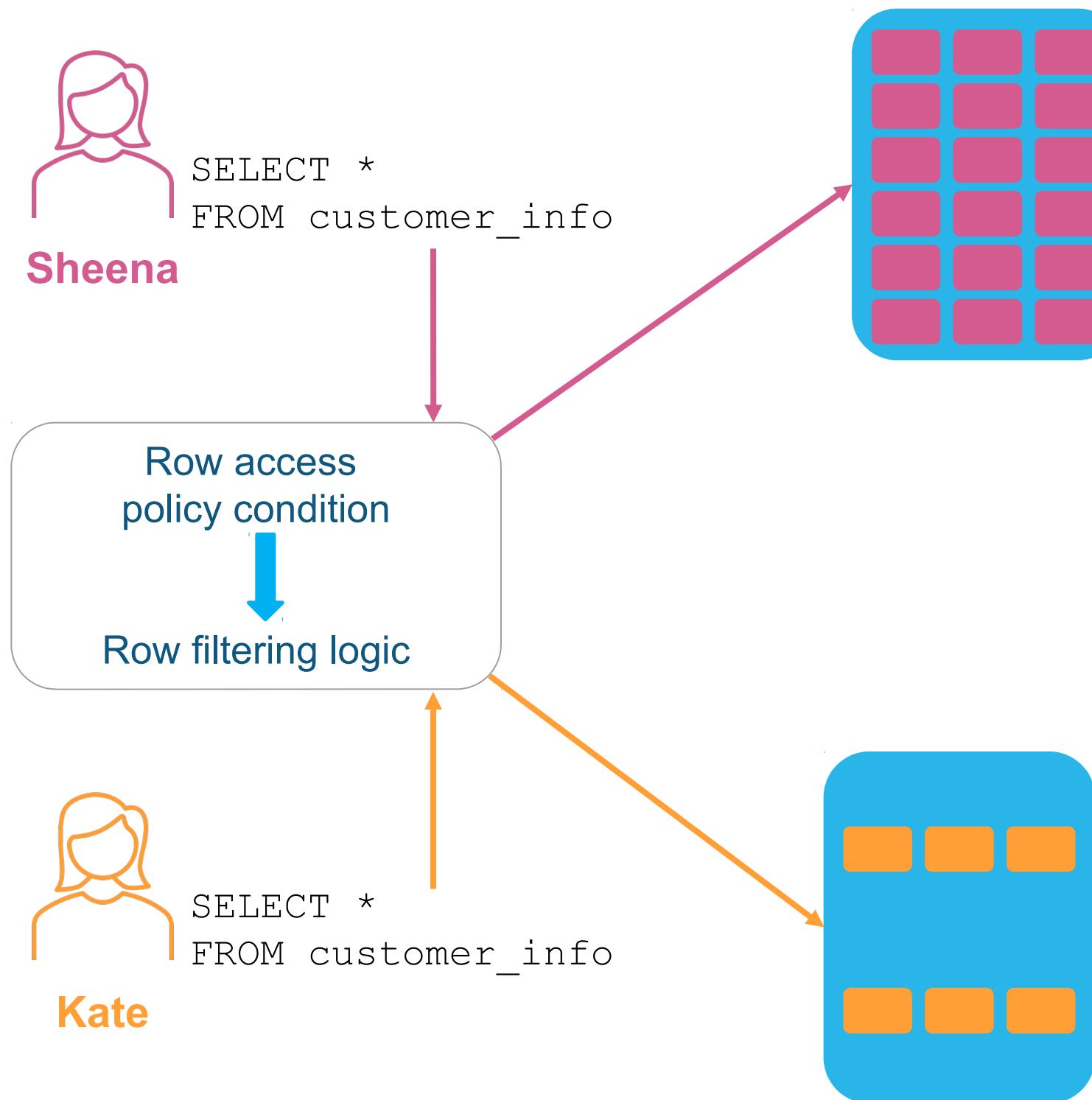
	DYNAMIC DATA MASKING	EXTERNAL TOKENIZATION
Data storage	Stored in clear	Stored as tokenized by partner solution
Delivery	Built-in	Through partner integrations (such as Protegrity) and external functions
Query time	User sees masked vs unmasked data based on <b>Snowflake masking policy</b>	User sees tokenized vs de-tokenized data based on <b>partner solution's access policy</b>



# ROW ACCESS POLICIES



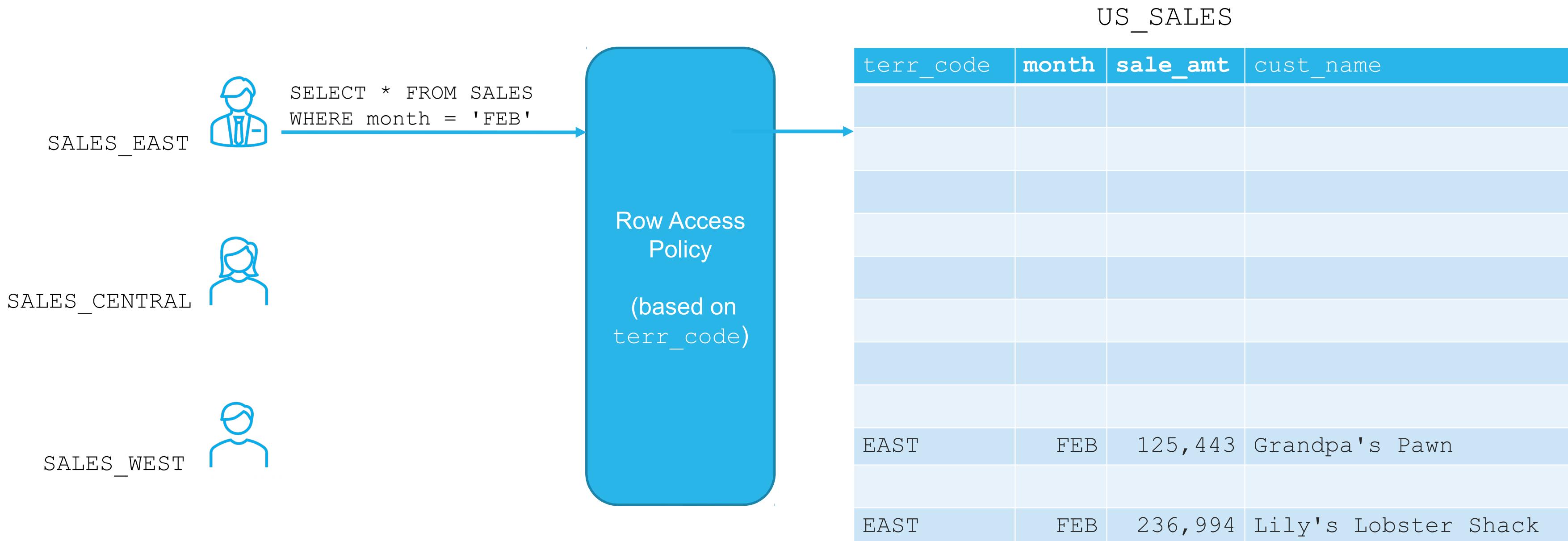
# WHAT IS A ROW ACCESS POLICY?



- A schema-level object that determines which rows are returned based on certain conditions
- Row access policies can be applied to:
  - Tables
    - Permanent
    - Transient
    - Temporary
    - External
  - Views
    - Standard
    - Secure
    - Materialized

# SAMPLE USE CASE

- One large table of sales, you want each rep to access only the sales in their territory



# ROW ACCESS POLICY SUPPORT

- You can apply row access policies to:
  - Tables
  - Views
  - Materialized views
  - External tables



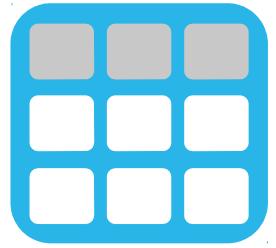
# SECURE VIEWS VS. ROW ACCESS POLICIES

Secure Views	Row Access Policies
Can create multiple secure views per source	Only one row access policy per source
Not policy-based; limited granularity	Granular, complex custom policies
Incur administration overhead	Concurrently enable multiple conditions on each table or view
View owner has full access to data	Access to data can be denied to objects owners and system administrators



# ROW ACCESS POLICIES

## WORKFLOW

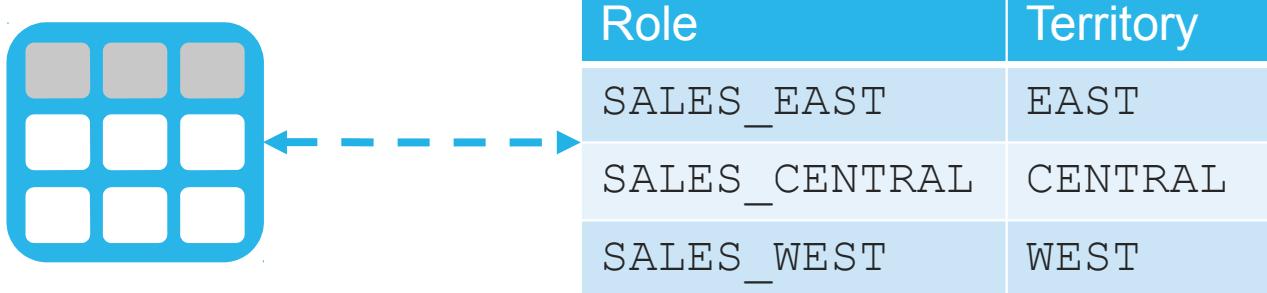


1. Identify or create the table or view that needs the policy

# ROW ACCESS POLICIES

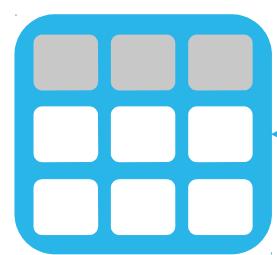
## WORKFLOW

1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required



# ROW ACCESS POLICIES

## WORKFLOW



Role	Territory
SALES_EAST	EAST
SALES_CENTRAL	CENTRAL
SALES_WEST	WEST

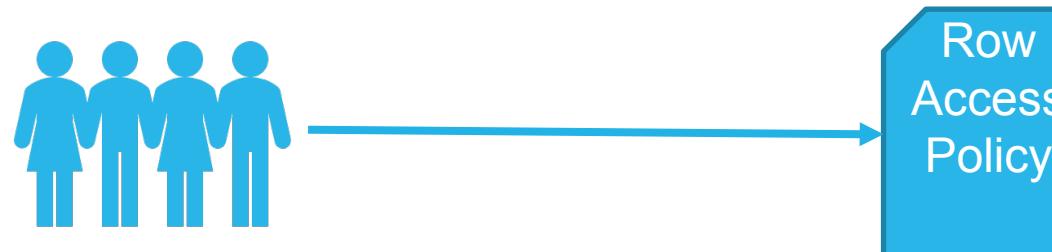
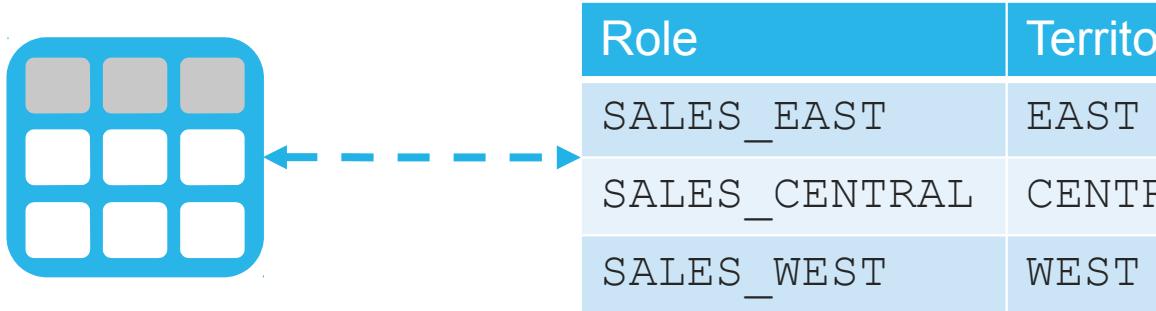


```
GRANT CREATE ROW ACCESS POLICY...
GRANT APPLY ROW ACCESS POLICY...
```

1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required
3. Grant privileges to policy administrator

# ROW ACCESS POLICIES

## WORKFLOW



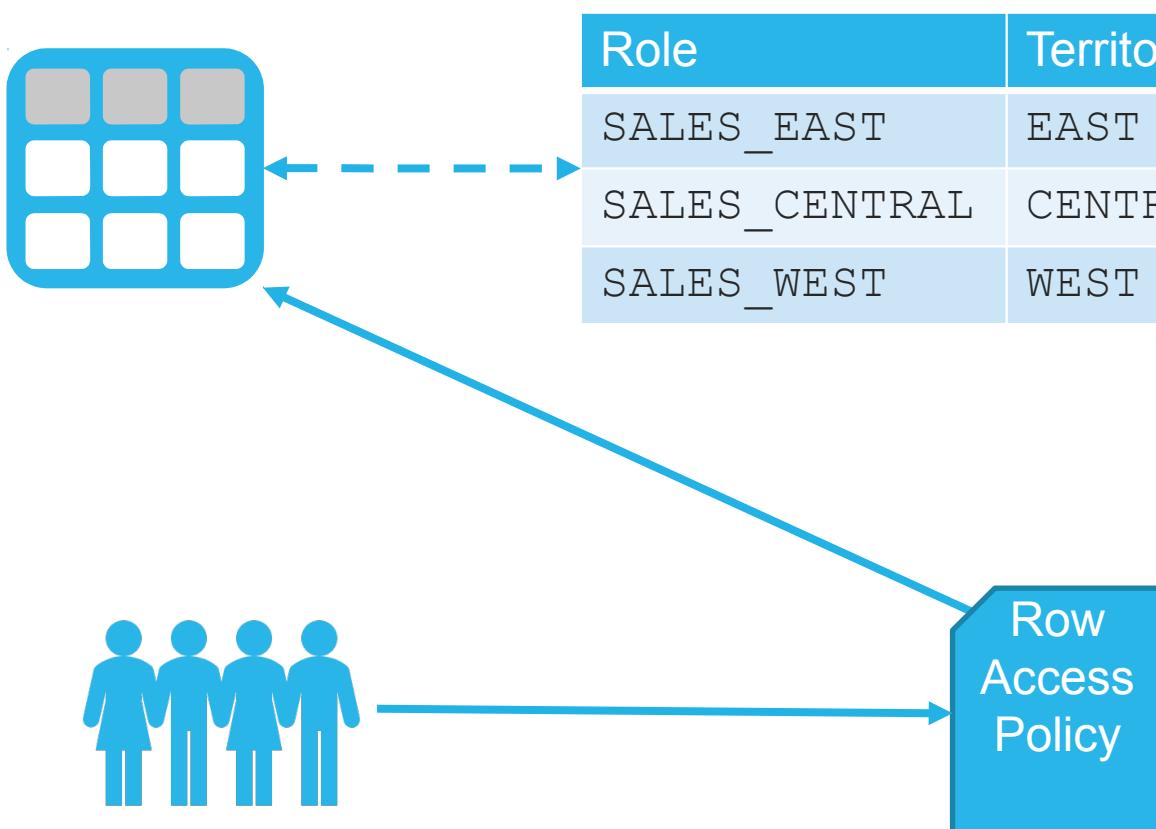
```
GRANT CREATE ROW ACCESS POLICY...
GRANT APPLY ROW ACCESS POLICY...
```

1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required
3. Grant privileges to policy administrator
4. Create row access policy



# ROW ACCESS POLICIES

## WORKFLOW

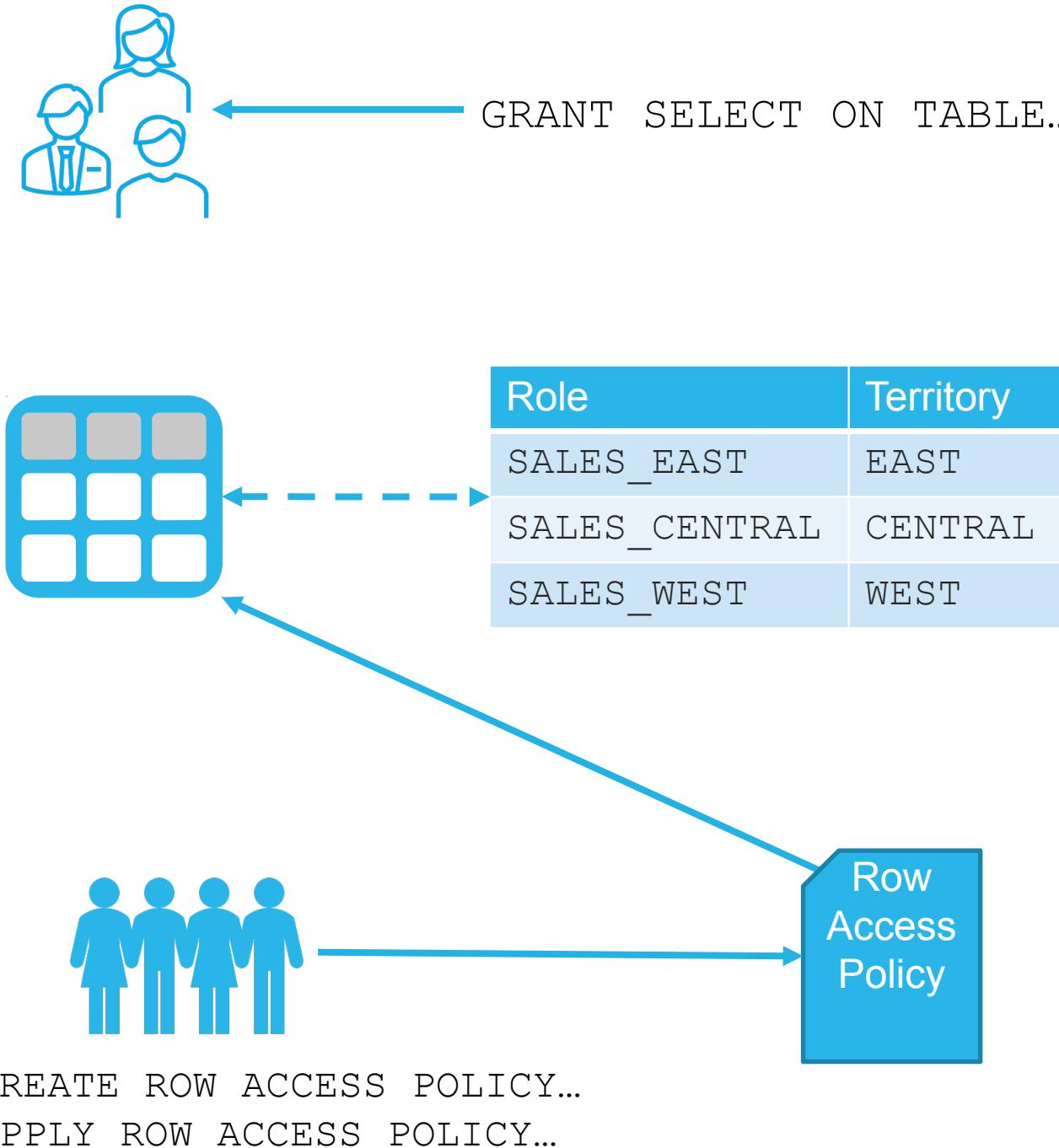


```
GRANT CREATE ROW ACCESS POLICY...
GRANT APPLY ROW ACCESS POLICY...
```

1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required
3. Grant privileges to policy administrator
4. Create row access policy
5. Apply row access policy

# ROW ACCESS POLICIES

## WORKFLOW

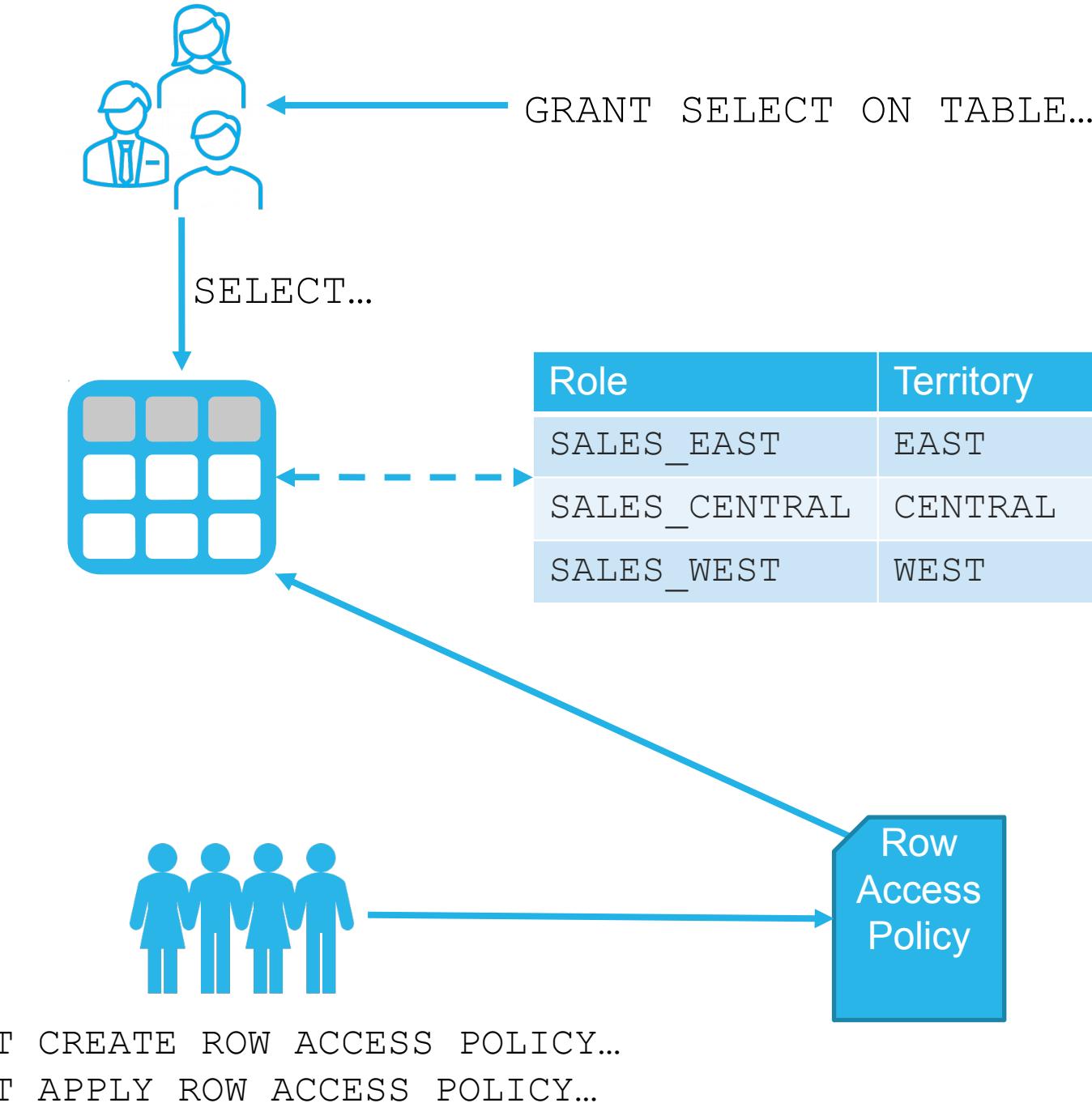


1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required
3. Grant privileges to policy administrator
4. Create row access policy
5. Apply row access policy
6. Grant privileges on table/view to users



# ROW ACCESS POLICIES

## WORKFLOW



1. Identify or create the table or view that needs the policy
2. Set up a mapping table, if required
3. Grant privileges to policy administrator
4. Create row access policy
5. Apply row access policy
6. Grant privileges on table/view to users
7. Test to verify the policy works

# CREATE A ROW ACCESS POLICY

- Row access policies can be created by ACCOUNTADMIN, or the schema owner
- Basic syntax:

```
CREATE ROW ACCESS POLICY <name> AS  
(<column> <type>) RETURNS BOOLEAN ->  
<expression>;
```

Name of the policy

One or more argument names and their type

Expression that evaluates as TRUE or FALSE



# ROW ACCESS POLICY

## SIMPLE FILTERING

ZIP	DETAILS
80504	...
00265	...
10024	...
30305	...

```
CREATE ROW ACCESS POLICY rap_class AS  
(zip INTEGER) RETURNS BOOLEAN ->  
    CURRENT_ROLE() = 'SECURE_ROLE';
```

- A table contains information with classified records that only someone using the role SECURE\_ROLE can view
- All or nothing – SECURE\_ROLE can view all rows, no other role can see any rows
- You must pass in an argument to the row access policy – even if the policy doesn't use it



# ROW ACCESS POLICY

## CONDITIONAL FILTERING

DETAILS	TYPE
...	CLASSIFIED
...	PUBLIC
...	PUBLIC
...	CLASSIFIED

```
CREATE ROW ACCESS POLICY rap_class AS
(type VARCHAR) RETURNS BOOLEAN ->
    CURRENT_ROLE = 'SECURE_ROLE' OR
    type = 'PUBLIC';
```

- A table contains information with both classified and public records
  - Users in the role SECURE\_ROLE role can see all records
  - Other roles can see only public records
- Query a column in the table to determine whether a row should be returned



# ROW ACCESS POLICY

## FILTER BASED ON ANOTHER TABLE

DETAILS	TYPE
...	CLASSIFIED

USER_NAME	LEVEL
HEDGEHOG	0
PONY	2
EAGLE	3

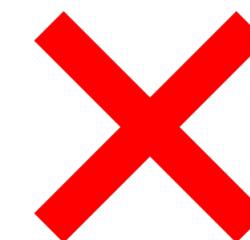
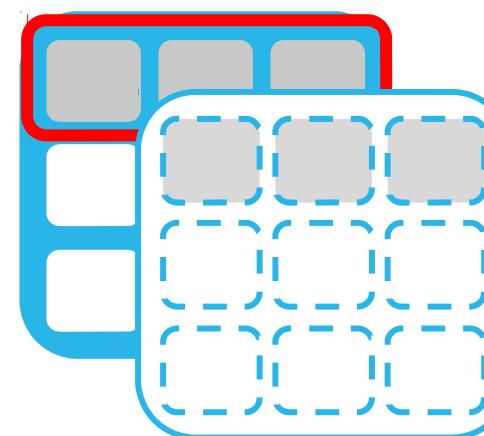
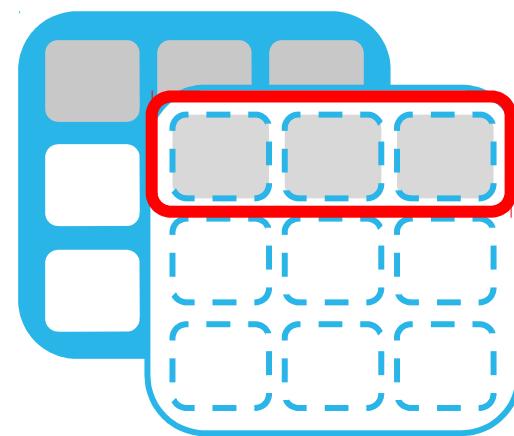
```
CREATE ROW ACCESS POLICY rap_class AS
(type VARCHAR) RETURNS BOOLEAN ->
CURRENT_ROLE() = 'SECURE_ROLE' AND
EXISTS (SELECT 1 FROM clearance_levels
WHERE user_name = CURRENT_USER()
AND level > 2);
```

- A table contains information with both classified and public records
  - Users in the role SECURE\_ROLE will be checked for their security level
  - Users with a security clearance level above 2 can see all records
  - No other roles can see records
- Query a column in a different (mapping) table to determine whether a row should be returned



# MATERIALIZED VIEW SUPPORT

- You can put a row access policy on a materialized view
- You **cannot** put/have a row access policy on a base table used to create a materialized view



# EXTERNAL TABLE SUPPORT

- You can apply a row access policy to the `VALUE` column of an external table
- A row access policy cannot be added to a virtual column
- You cannot use an external table as a mapping table for a row access policy



# ROW ACCESS POLICIES: BEST PRACTICES

- Limit the number of policy arguments
  - Snowflake must scan every column the policy is bound to, even if it's not referenced in queries
- Simplify the policy expression
  - Simple expressions or CASE statements perform better than using mapping tables
- For very large tables, cluster by the attributes used for policy filtering



# LAB EXERCISE: 10

## Use Row Access Policies

10 minutes

- Create objects
- Work with Row Access Policies



# CLASSIFICATION AND OBJECT TAGGING



# WHAT IS CLASSIFICATION?

NAME	GENDER	AGE	SSN	SALARY

**Identifier** – information that can uniquely and directly identify an individual

**Quasi-Identifier** – information that can indirectly identify someone when combined with other information

**Sensitive** – information that will not identify an individual, but they may not want disclosed for privacy reasons

- Analyzes and automatically categorizes columns of tables or views
- Samples all supported columns and classifies the data based on column names and values
- Assigns pre-defined privacy categories
  - Identifier
  - Quasi-identifier
  - Sensitive



# WHAT IS TAGGING?

A tag is stored as a **key–value** pair

NAME	GENDER	AGE	SSN	SALARY

Diagram illustrating the mapping of columns to tags:

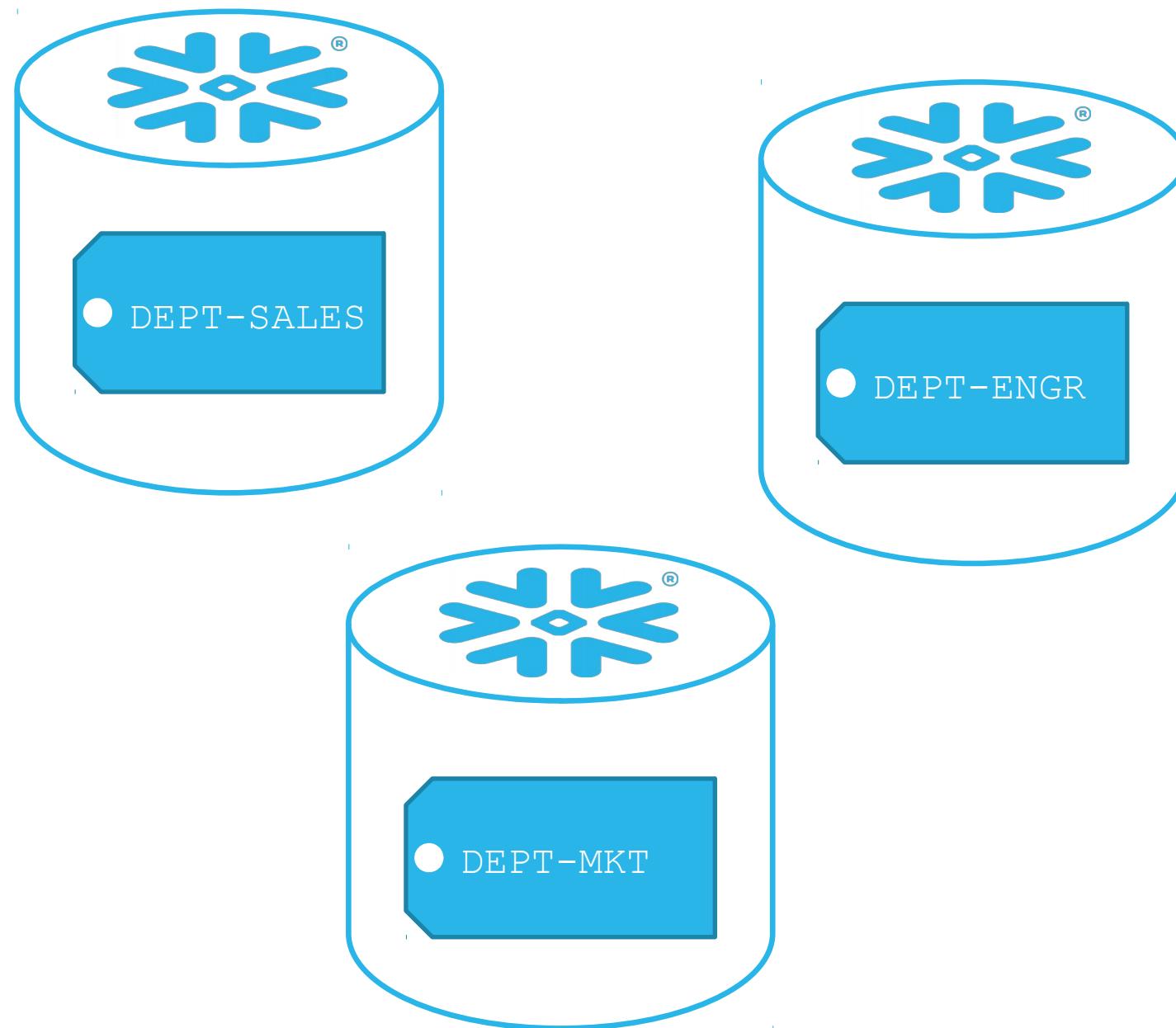
- NAME → QID-Name
- GENDER → QID-Gender
- AGE → QID-Age
- SSN → Identifier-SSN
- SALARY → Sensitive-Salary

- Tagging is labeling certain objects (such as views, tables, or columns) with attributes you want to track
  - PII or Sensitive Data
  - Cost centers
- Tags can automatically be applied to classified columns, or created and applied manually
- Can set 20 unique tags on a single table
  - Multiple tags per column supported



# WORKFLOW

Sample use case: resource tracking



1. Define a role with the ability to create and apply tags
2. Create a tagging plan
3. EXTRACT\_SEMANTIC\_CATEGORIES to analyze a table or view
4. Create tags
5. Assign tags to existing objects or columns using ALTER TAG
6. Track tags using table functions and views

# TO LEARN MORE

- At [docs.snowflake.com](https://docs.snowflake.com), search for:
  - Data Classification
  - Object Tagging
  - Managing Governance
- Object classification and tagging are covered in more detail in Snowflake's one-day Data Governance course



# AUDIT ACCESS HISTORY



# ACCESS HISTORY VIEW

- Used to see who accesses what data, and when
- Found in SNOWFLAKE.ACOUNT\_USAGE.ACCESS\_HISTORY
  - Information retained for one year
  - Only read operations are currently included
- Each ACCESS\_HISTORY view row contains a single audit record per query
  - Describes columns the query accessed, directly or indirectly
  - Provides insights on frequently accessed tables and columns



# USE CASES

- Discover unused data to determine whether to archive or delete the data
- Audit data access to comply with regulatory requirements and data governance initiatives
- Audit accesses to tagged columns of PII data
- Identify columns tagged as PII but without any row access or data masking policies applied



# ACCESS HISTORY DATA

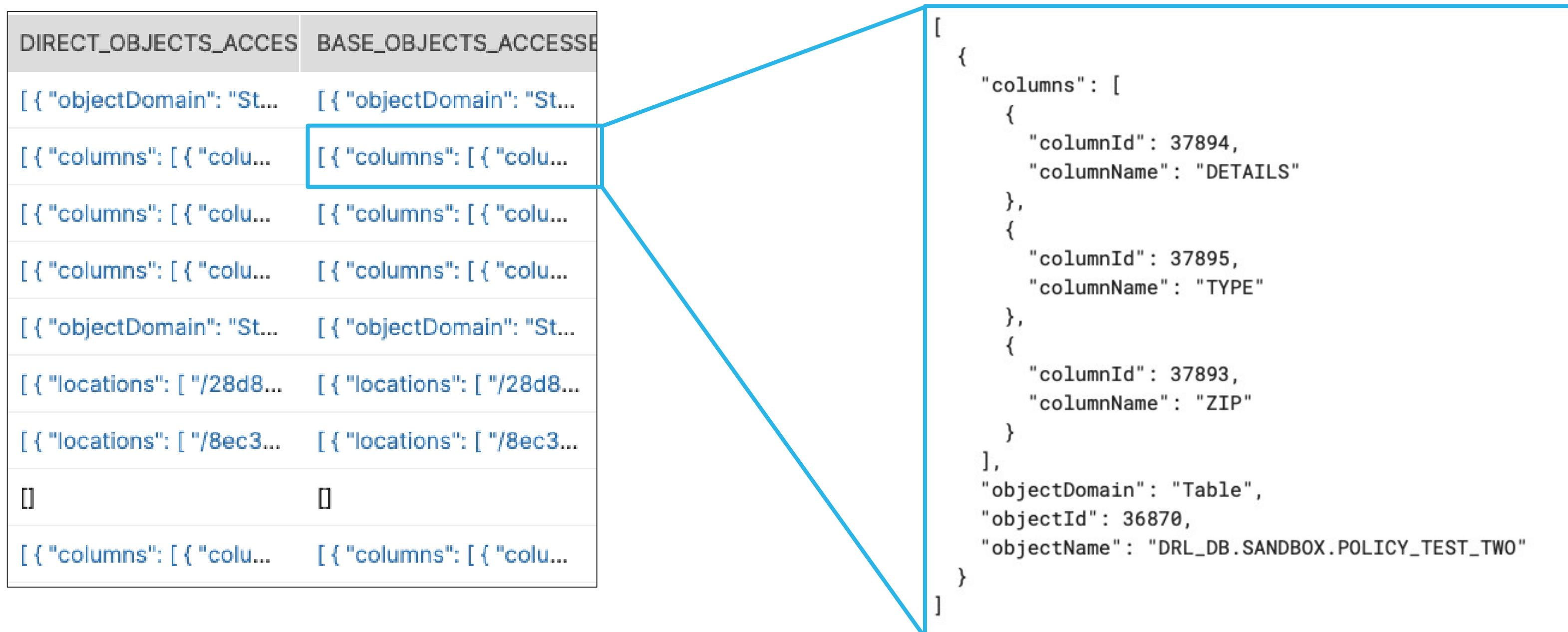
```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY LIMIT 10;
```

QUERY_ID	QUERY_START_TIME	USER_NAME	DIRECT_OBJECTS_ACCE	BASE_OBJECTS_ACCESE	OBJECTS_MODIFIED
01a24f29-0601-2fbf-0...	2022-02-14 09:13:55....	SNOWACE	[ { "objectDomain": "St...	[ { "objectDomain": "St...	[ { "columns": [ { "colu...
01a2edd6-0601-481a-...	2022-03-14 15:14:15.9...	HEDGEHOG	[ { "columns": [ { "colu...	[ { "columns": [ { "colu...	[]
01a2eddb-0601-481a-...	2022-03-14 15:19:44....	HEDGEHOG	[ { "columns": [ { "colu...	[ { "columns": [ { "colu...	[]
01a29f55-0601-3bce-...	2022-02-28 15:17:22....	PENGUIN	[ { "columns": [ { "colu...	[ { "columns": [ { "colu...	[]
01a29d57-0601-3c94...	2022-02-28 06:47:30....	PONY	[ { "objectDomain": "St...	[ { "objectDomain": "St...	[ { "locations": [ "/" ] }]
01a29d66-0601-3be2...	2022-02-28 07:02:04....	WORKSHEETS_APP_U...	[ { "locations": [ "/28d8..."	[ { "locations": [ "/28d8..."	[ { "objectDomain": "St...
01a29d67-0601-3c7f-...	2022-02-28 07:03:40....	WORKSHEETS_APP_U...	[ { "locations": [ "/8ec3..."	[ { "locations": [ "/8ec3..."	[ { "objectDomain": "St...
01a29e57-0601-3c94...	2022-02-28 11:03:44.1...	HEDGEHOG	[]	[]	[ { "columns": [ { "colu..."
01a2a514-0601-3de6-...	2022-03-01 15:48:44....	PONY	[ { "columns": [ { "colu..."	[ { "columns": [ { "colu..."	[]
01a29d66-0601-3c69...	2022-02-28 07:02:02....	WORKSHEETS_APP_U...	[ { "locations": [ "/532f..."	[ { "locations": [ "/532f..."	[ { "objectDomain": "St..."



# ACCESS HISTORY DATA

```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY LIMIT 10;
```



# ACCESS HISTORY DATA

```
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.ACCESS_HISTORY LIMIT 10;
```

BASE_OBJECTS_ACCESESSED	OBJECTS_MODIFIED
[{"objectDomain": "St...	[{"columns": [{"colu...
[{"columns": [{"colu...	[]
[{"columns": [{"colu...	[]
[{"columns": [{"colu...	[]
[{"objectDomain": "St...	[{"locations": ["/"]}]
[{"locations": ["/28d8...]	[{"objectDomain": "St...
[{"locations": ["/8ec3...]	[{"objectDomain": "St...
[]	[{"columns": [{"colu...

A blue line connects the 'OBJECTS\_MODIFIED' column of the first table to the JSON object in the second table, indicating a relationship between the two.

```
[{"columns": [{"columnId": 6189, "columnName": "CUST_NAME"}, {"columnId": 6187, "columnName": "MONTH"}, {"columnId": 6188, "columnName": "SALE_AMT"}, {"columnId": 6186, "columnName": "TERR_CODE"}], "objectDomain": "Table", "objectId": 6151, "objectName": "DRL_DB.RAP.SALES"}
```



# EXAMPLE: FLATTEN ACCESS HISTORY DATA

```
SELECT query_id                                AS QID,
       split_part(f.value:objectName::string,'.',1)   AS database_name,
       split_part(f.value:objectName::string,'.',2)   AS schema_name,
       split_part(f.value:objectName::string,'.',3)   AS object_name,
       upper(f.value:objectDomain)::string           AS object_type,
       f3.value:columnName::string                  AS column_name,
       user_name
FROM snowflake.account_usage.access_history,
     LATERAL FLATTEN(base_objects_accessed) f,
     LATERAL FLATTEN(f.value) f2,
     LATERAL FLATTEN(f2.value) f3;
```



# EXAMPLE: FLATTEN ACCESS HISTORY DATA

```
SELECT query_id AS QID,  
       split_part(f.value:objectName::string,'.',1) AS database_name,  
       split_part(f.value:objectName::string,'.',2) AS schema_name,  
       split_part(f.value:objectName::string,'.',3) AS object_name,
```

QID	DB_NAME	SCHEMA_NAME	OBJECT_NAME	OBJECT_TYPE	COLUMN_NAME	USER_NAME
01a2990c-0...	ZEBRA_TAX...	TAXSCHEMA	TAXPAYER	TABLE	STATE	ZEBRA
01a2990c-0...	ZEBRA_TAX...	TAXSCHEMA	TAXPAYER	TABLE	LASTNAME	ZEBRA
01a2edd6-0...	DRL_DB	SANDBOX	POLICY_TEST_TWO	TABLE	DETAILS	HEDGEHOG
01a2edd6-0...	DRL_DB	SANDBOX	POLICY_TEST_TWO	TABLE	TYPE	HEDGEHOG
01a2edd6-0...	DRL_DB	SANDBOX	POLICY_TEST_TWO	TABLE	ZIP	HEDGEHOG
01a2eddb-0...	DRL_DB	SANDBOX	POLICY_TEST_TWO	TABLE	DETAILS	HEDGEHOG



# ADVANCED FEATURES



# MODULE AGENDA

- External Functions
- API Integrations
- SQL API
- Geography Data Type

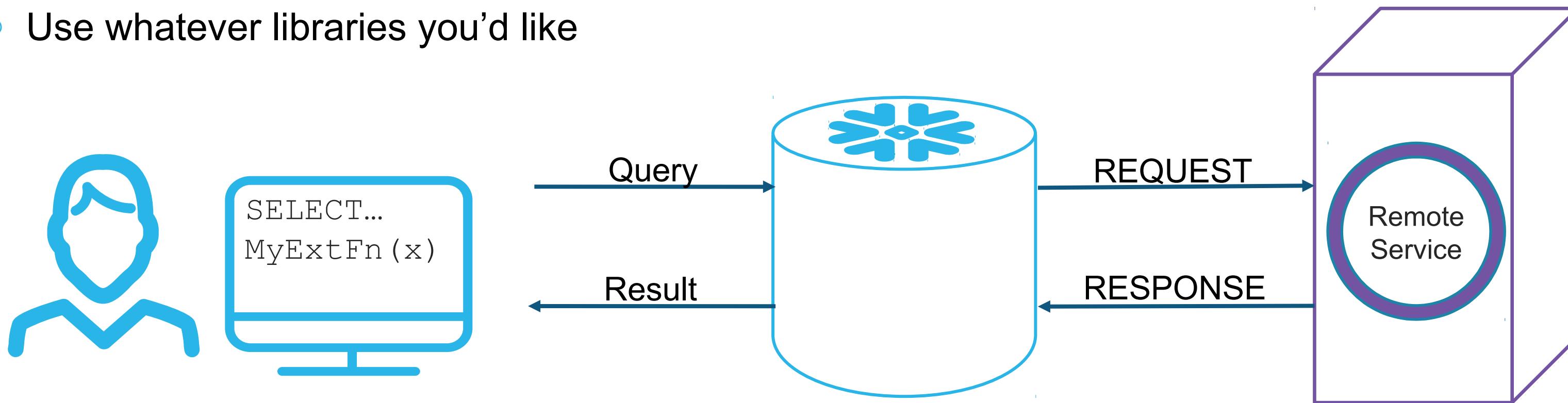


# EXTERNAL FUNCTIONS



# WHAT IS AN EXTERNAL FUNCTION?

- Calls code (remote service) that executes outside of Snowflake
- Host arbitrary code to implement a scalar function:
  - Call custom code or third-party services
  - Host however you'd like (EC2, Lambda, Kubernetes, etc.)
  - Write in whatever language you'd like (Python, C#, Go, etc.)
  - Use whatever libraries you'd like



# REMOTE SERVICE

Geocoding



ML Scoring  
and Inference



Complex  
business logic



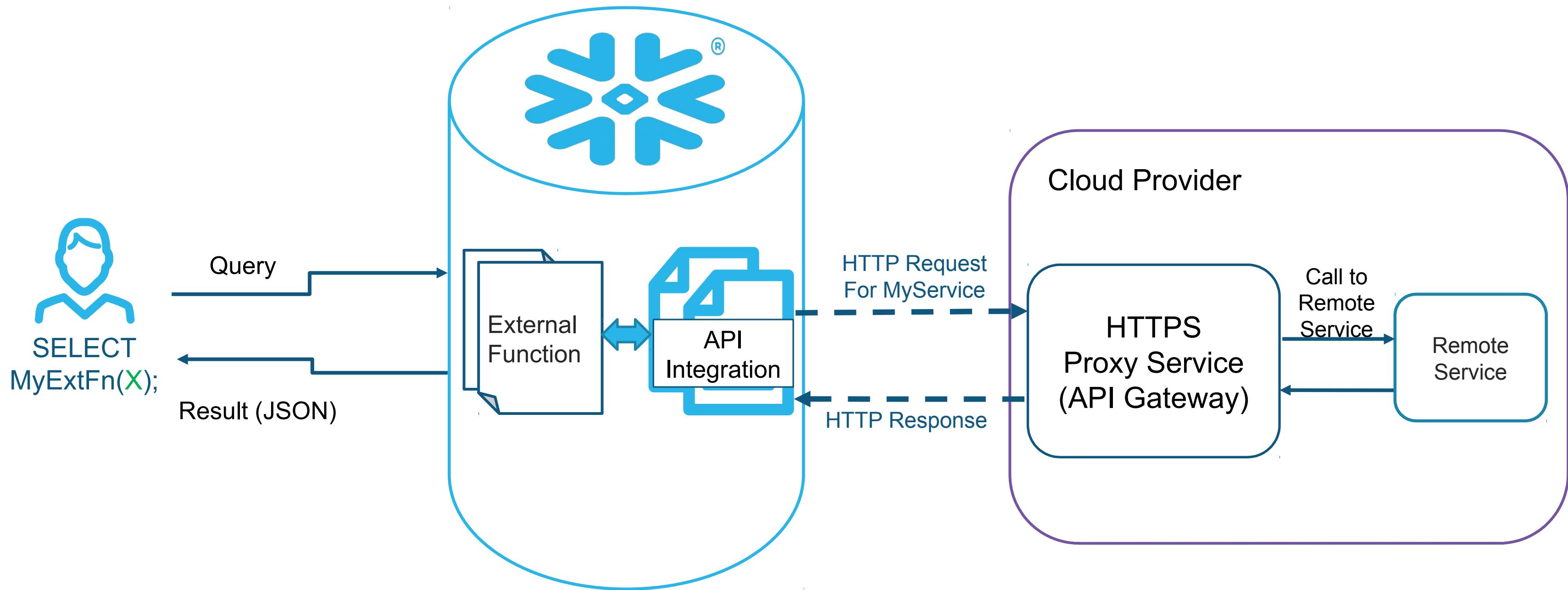
Access to  
live data



To be called by the Snowflake external function feature, the remote service must:

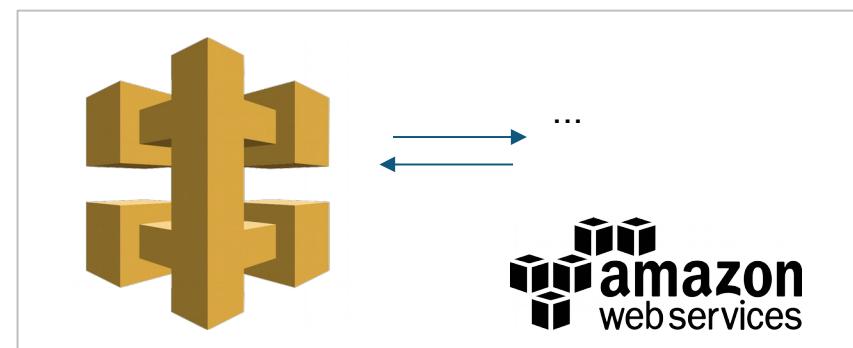
- Expose an HTTPS endpoint
- Accept JSON inputs and return JSON outputs
- Be callable from a proxy service
- Be a scalar function
  - Return a single value for each input “row”

# HOW EXTERNAL FUNCTIONS WORK

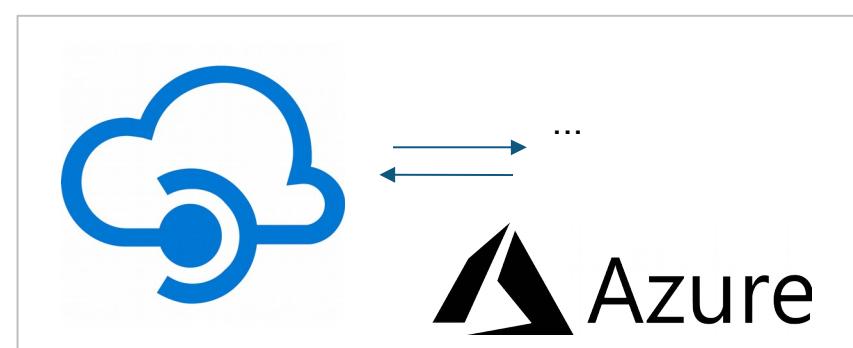


# CREATE API INTEGRATION

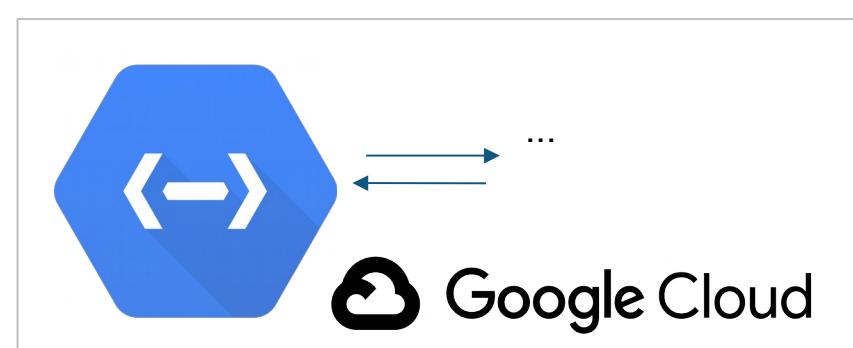
## HTTPS PROXY SERVICE



**Amazon API Gateway**



**Azure API Management Service**



**Google Cloud API Gateway**

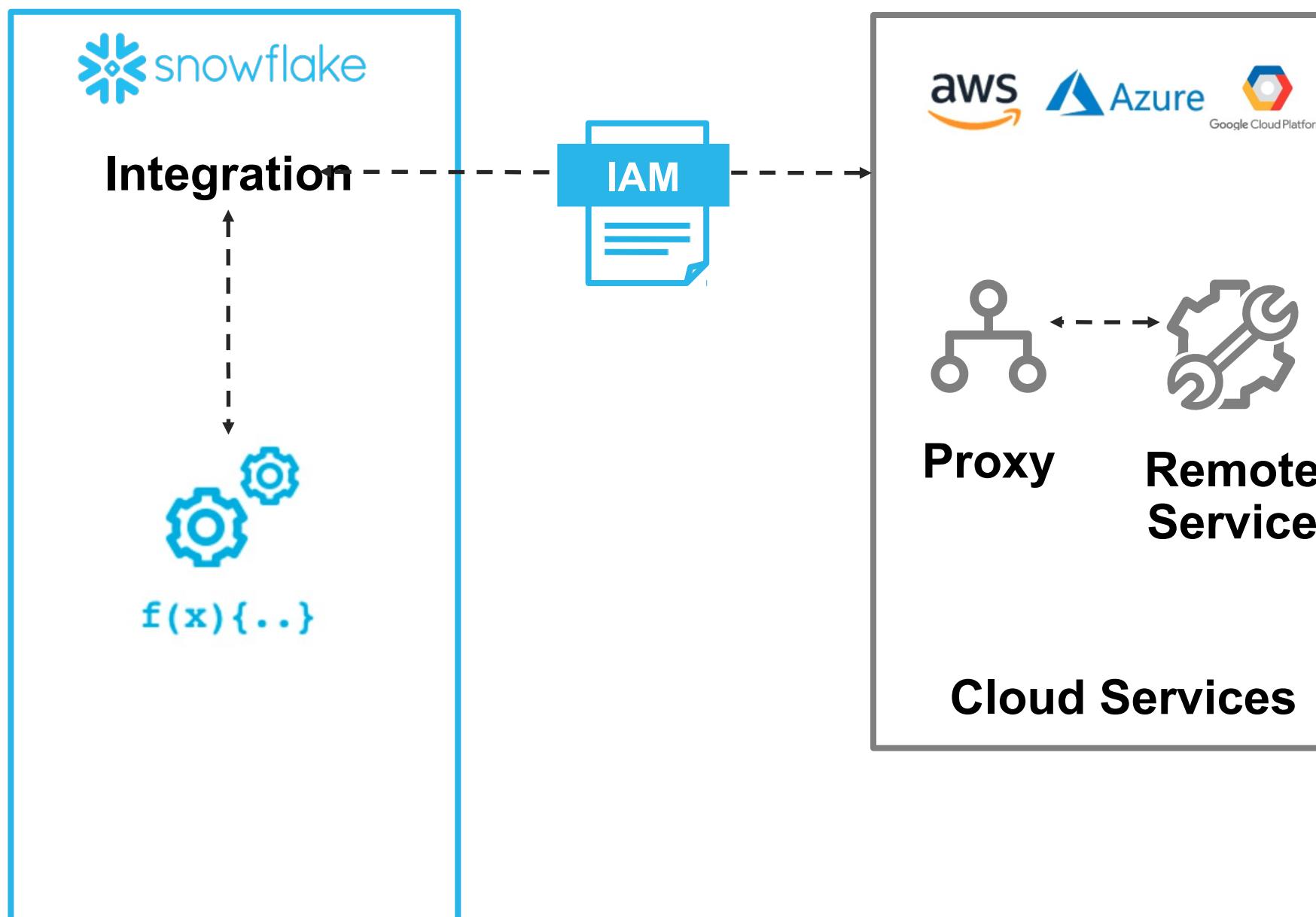
- API Integration object stores information about an HTTPS proxy service:
  - Cloud platform provider
  - Type of proxy service
  - Identifier and access credentials
  - Specifies allowed (and optionally blocked) endpoints and resources
- The syntax is different for each cloud platform

# API INTEGRATIONS



# API INTEGRATION

## TRUSTED REMOTE SERVICE ACCESS



- Allows users to create external functions and work with services that are outside of Snowflake
- Stores information about an outside service, including:
  - Cloud provider
  - Type of proxy service
  - Identifier/access credentials for a cloud platform role with sufficient privileges to use the proxy service

# CREATE AN API INTEGRATION (AWS)

```
CREATE API INTEGRATION aws_integration
```

```
API_PROVIDER = aws_api_gateway
```

```
API_AWS_ROLE_ARN = 'arn:aws:iam::123456789012:role/my_role'
```

```
API_ALLOWED_PREFIXES =
```

```
    ('https://xyz.execute-api.us-west-2.amazonaws.com/production')
```

```
ENABLED = TRUE;
```

- See the documentation for a complete list of options for all cloud providers



# API INTEGRATION USAGE NOTES

- An API Integration object is tied to a specific cloud platform
  - You can have multiple integration objects, to support different providers
- You can use the same API integration to authenticate to multiple HTTPS proxy services in that account
- Multiple external functions can use the same API integration object
- Use `API_BLOCKED_PREFIXES` option to list endpoints and resources in the HTTPS proxy services that are NOT allowed to be called from Snowflake
  - Values outside `API_ALLOWED_PREFIXES` do not need to be explicitly blocked



# LAB EXERCISE: 11

## Using External Functions

10 minutes

- Investigate the API Integration
- CREATE an External Function
- Call the External Function to execute the remote service code
- Create another External Function and call it
- Dropping an External Function



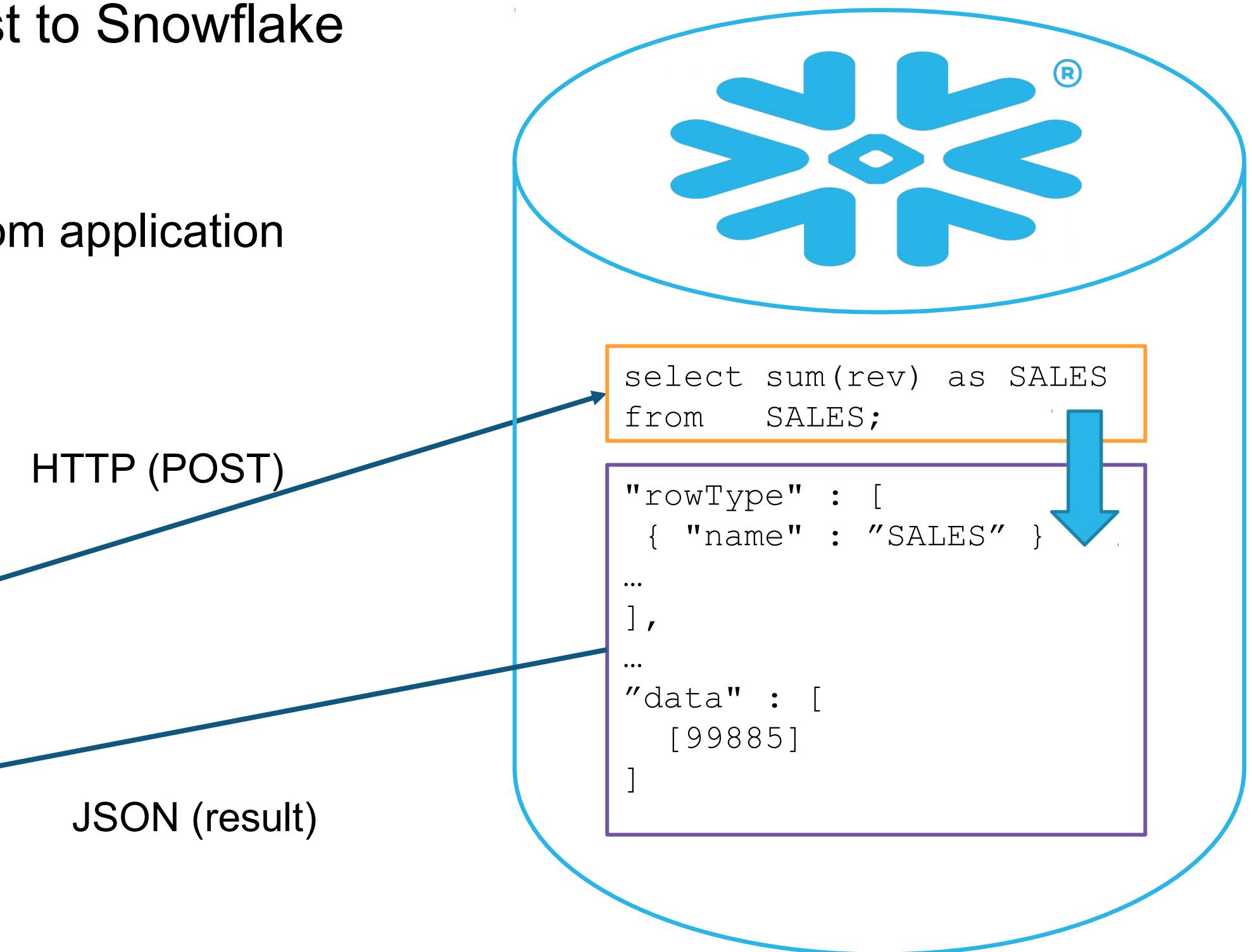
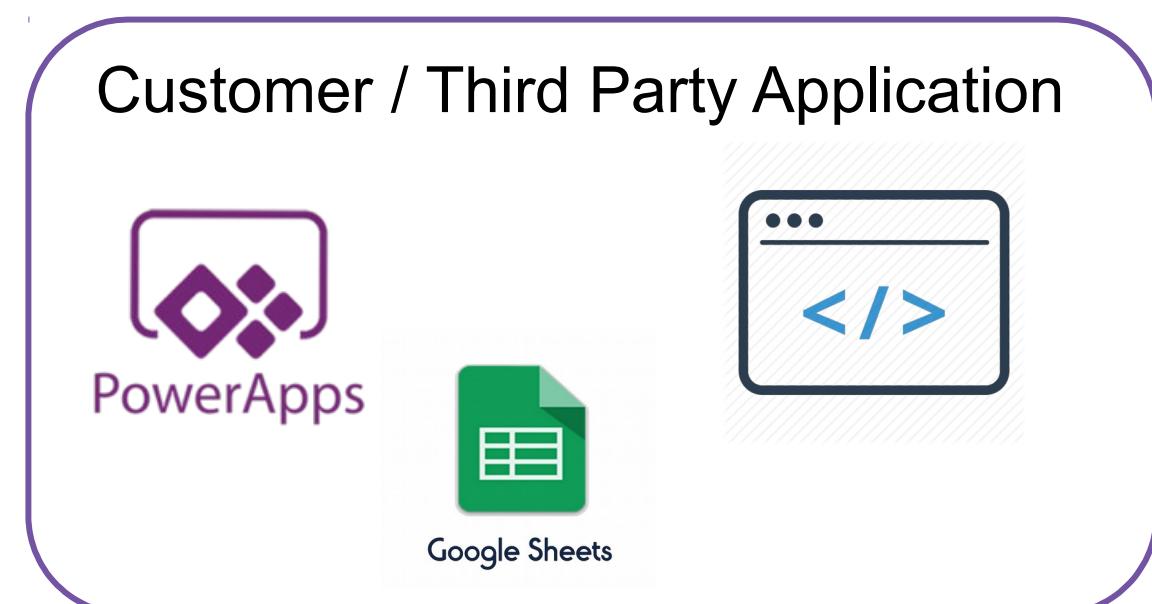
# SQL API



# WHAT IS SQL API?

A **Client Application** issues REST request to Snowflake

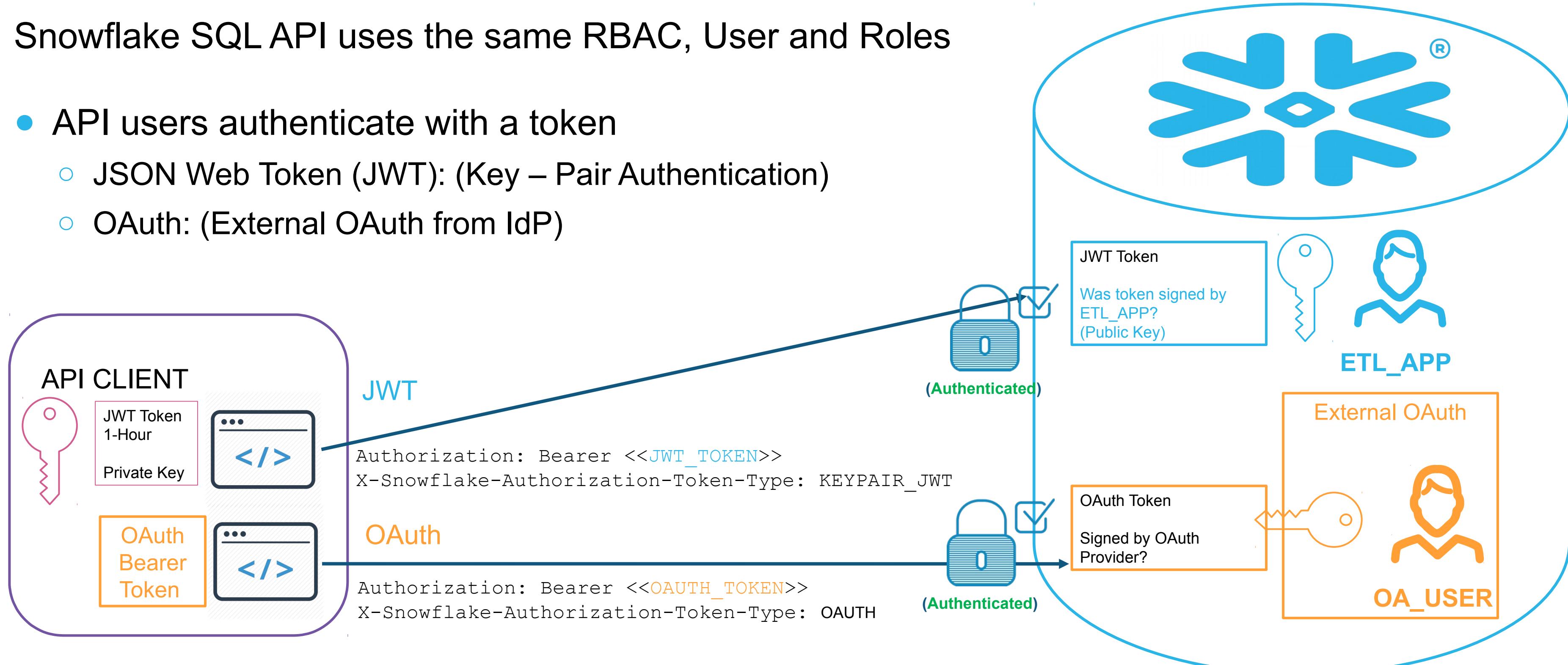
- Lightweight, secure, endpoints to
  - Query and retrieve data (in JSON) to custom application
  - Call stored procedures
  - Perform DML (COPY/INSERT)
  - Execute administrative tasks



# SQL API AUTHENTICATION

Snowflake SQL API uses the same RBAC, User and Roles

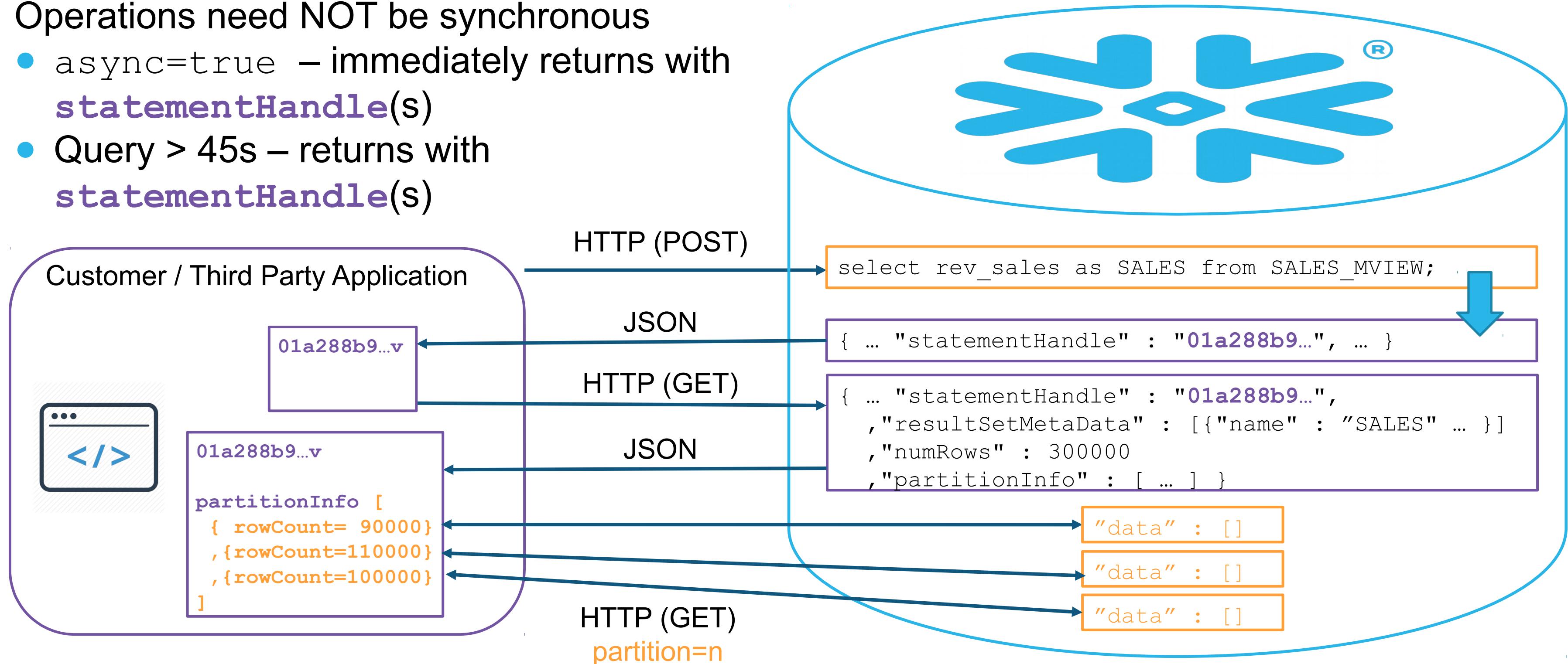
- API users authenticate with a token
  - JSON Web Token (JWT): (Key – Pair Authentication)
  - OAuth: (External OAuth from IdP)



# ASYNCHRONOUS CAPABILITIES

Operations need NOT be synchronous

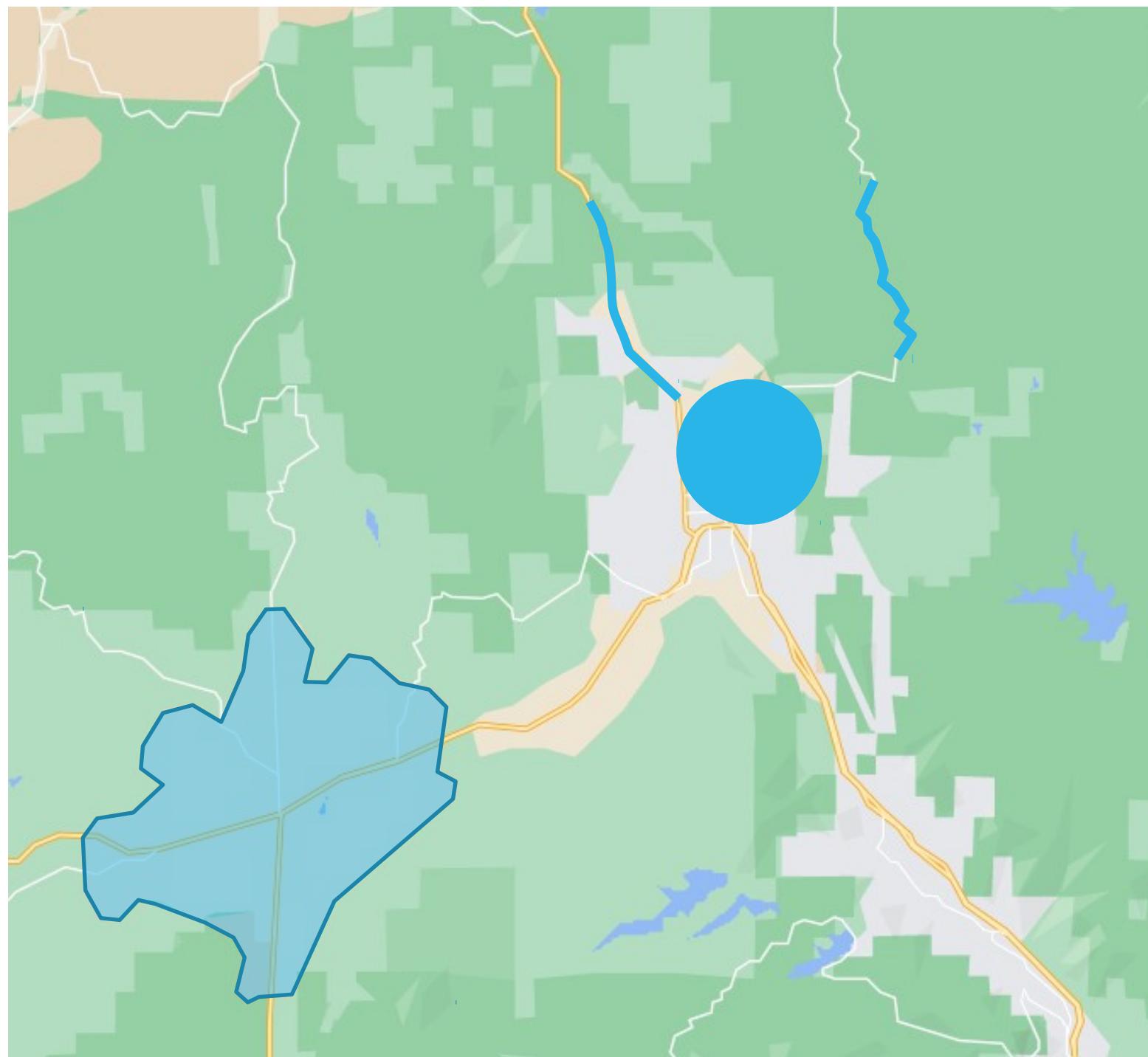
- `async=true` – immediately returns with **statementHandle(s)**
- Query > 45s – returns with **statementHandle(s)**



# USING GEOGRAPHY DATA



# GEOSPATIAL ELEMENTS



GEOGRAPHY data is used to represent geospatial features, including:

- Points – indicate a specific place
  - Points have no area
- Lines – indicate things like streams and roads
- Polygons – outline an area, such as the city limits, or a lake

# WHAT IS AN SRID?



The earth is not a perfect sphere, and its surface is not flat

- SRS – Spatial Reference System
  - A method for calculating the distance between two points on earth
- SRID – Spatial Reference ID
  - Indication of which SRS is being used with geospatial data
- Snowflake uses SRID 4326
  - Coordinates expressed as (lon, lat)
  - Same system used by GPS
  - Lines interpreted as geodesic arcs

# INPUT AND OUTPUT FORMATS

Format	Notes	Example
Well Known Text (WKT)	Human-readable	POINT (-105.06 38.99)
Extended WKT (EWKT)	WKT with SRID	SRID=4326; POINT (-105.06 38.99)
Well Known Binary (WKB)	Hexadecimal string	0101000000A4703D0AD7435AC01F85EB51B87E4340
Extended WKB (EWKB)	WKB with SRID	0101000020E6100000A4703D0AD7435AC01F85EB51B87E4340
GeoJSON (default)	JSON Format	{ "coordinates": [ -105.06, 38.99 ], "type": "Point" }



# SUPPORTED GEOSPATIAL OBJECTS

## (E)WKT / (E)WKB / GeoJSON

- Point
- MultiPoint
- LineString
- MultiLineString
- Polygon
- MultiPolygon
- GeometryCollection

## GeoJSON Only

- Feature
- FeatureCollection



# GEOSPATIAL FUNCTIONS

- Operate on values of type GEOGRAPHY, or convert GEOGRAPHY values to/from other data types
- Examples:
  - TO\_GEOGRAPHY
  - TRY\_TO\_GEOGRAPHY
  - ST\_ASWKB or ST\_ASBINARY
  - ST\_MAKEPOINT
- Refer to the documentation for a complete list



# USAGE NOTES

- By default, Snowflake will generate an error if an input value in EWKB or EWKT format includes an SRID other than 4326
- Standard GeoJSON line segments are generally straight lines; Snowflake uses spherical semantics
- Altitude is currently not supported
- Geospatial data stored in `GEOGRAPHY` columns may perform significantly better than the same data stored in `VARCHAR`, `VARIANT`, or `NUMBER` columns
- JavaScript UDFs allow the `GEOGRAPHY` type as an argument and a return value



# LAB EXERCISE: 12

## Using Geospatial Geography

15 minutes

- Acquire geospatial Data from Snowflake Marketplace
- Understand Geospatial Formats
- Visualize the data
- Example Business school scenario using some geospatial functions



# STREAMS (CHANGE DATA CAPTURE)



# MODULE AGENDA

- Traditional Micro-Batch Pipeline
- Alternative: Continuous Data Pipeline
- Stream Details
- How Streams Work
- Common Use Cases

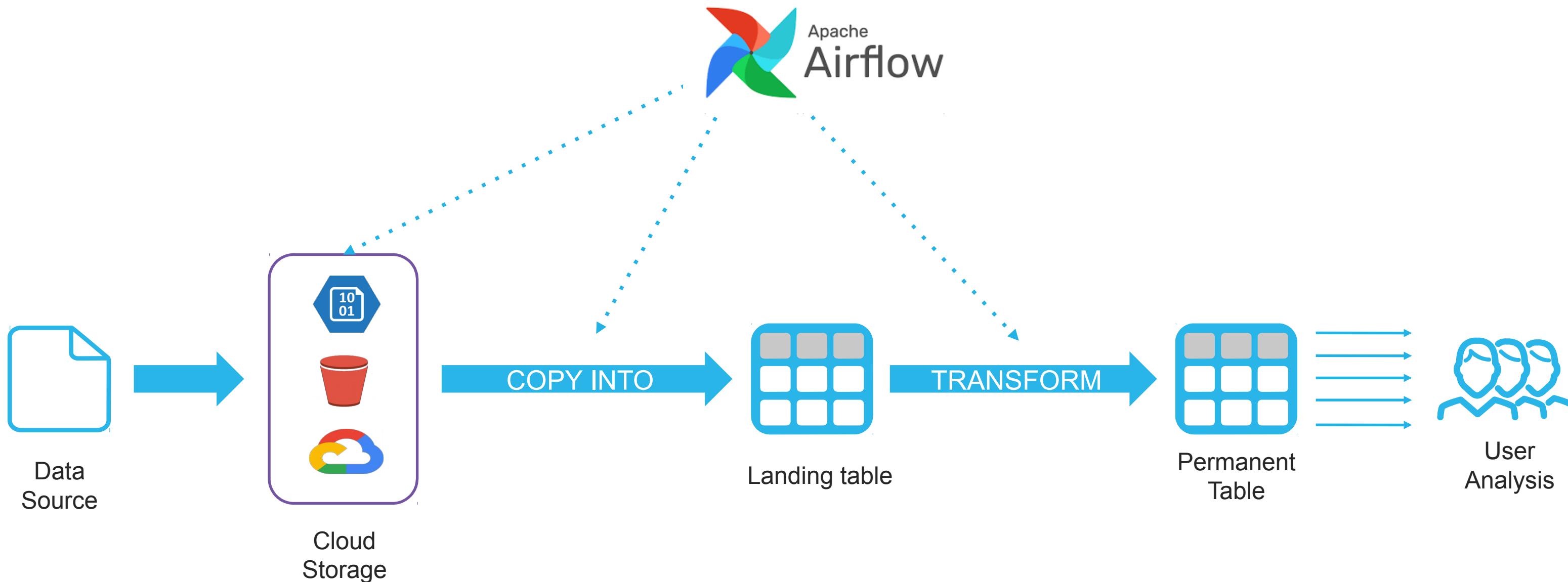


# TRADITIONAL MICRO-BATCH PIPELINE



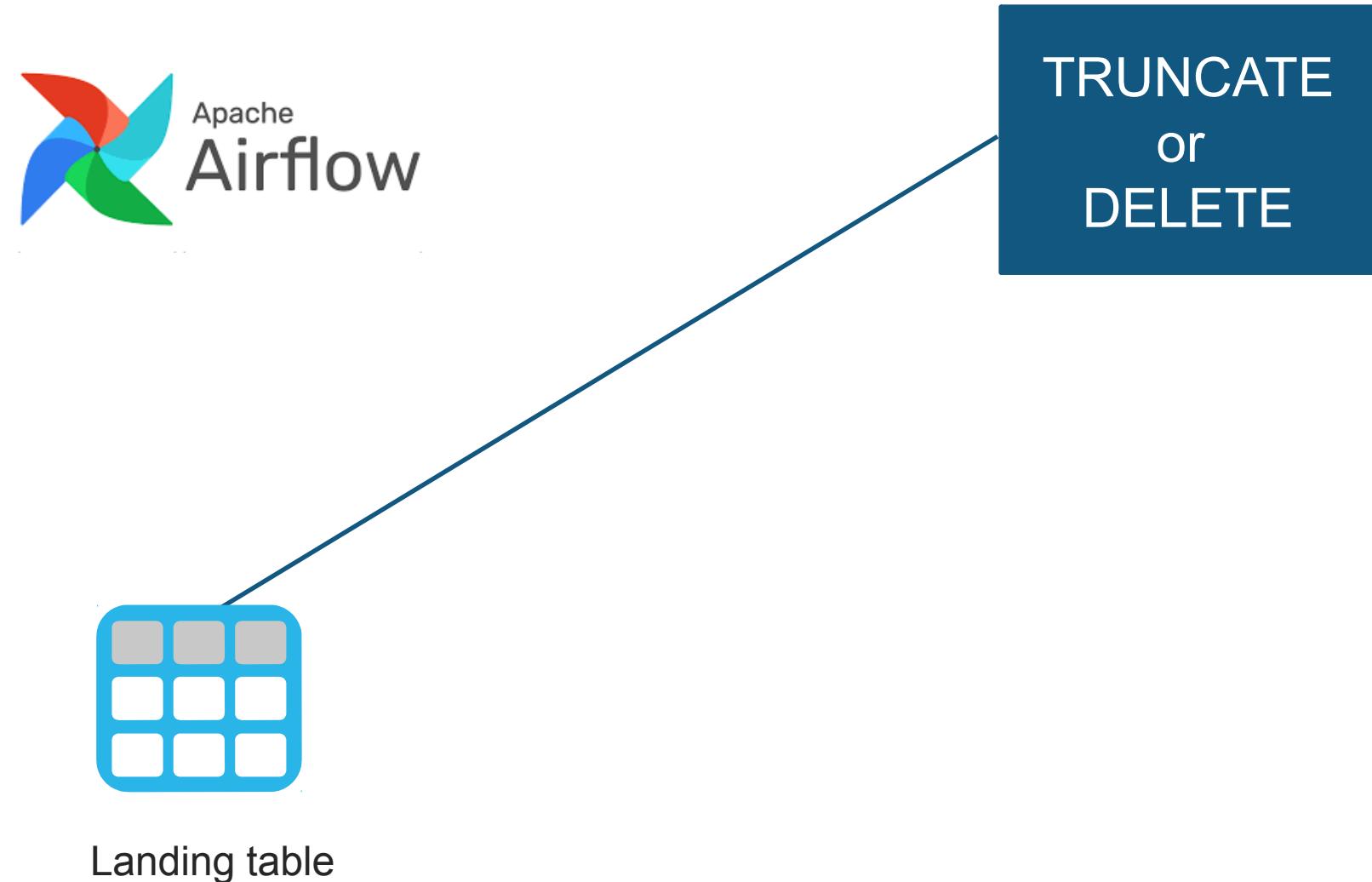
# TYPICAL MICRO-BATCH PIPELINE

- Application like Apache Airflow may be used to trigger events



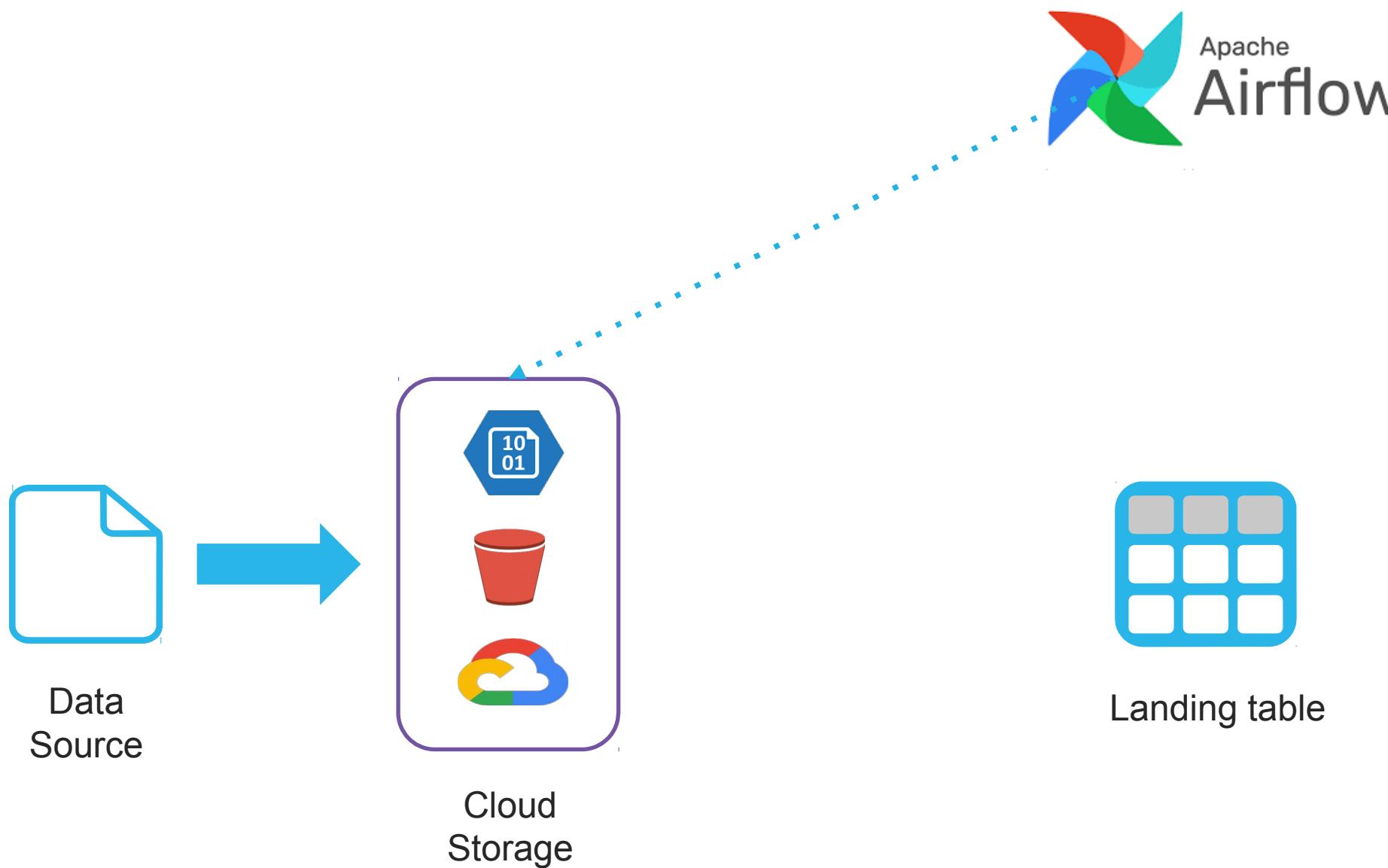
# TYPICAL MICRO-BATCH PIPELINE

1. Prepare landing table for run of pipeline



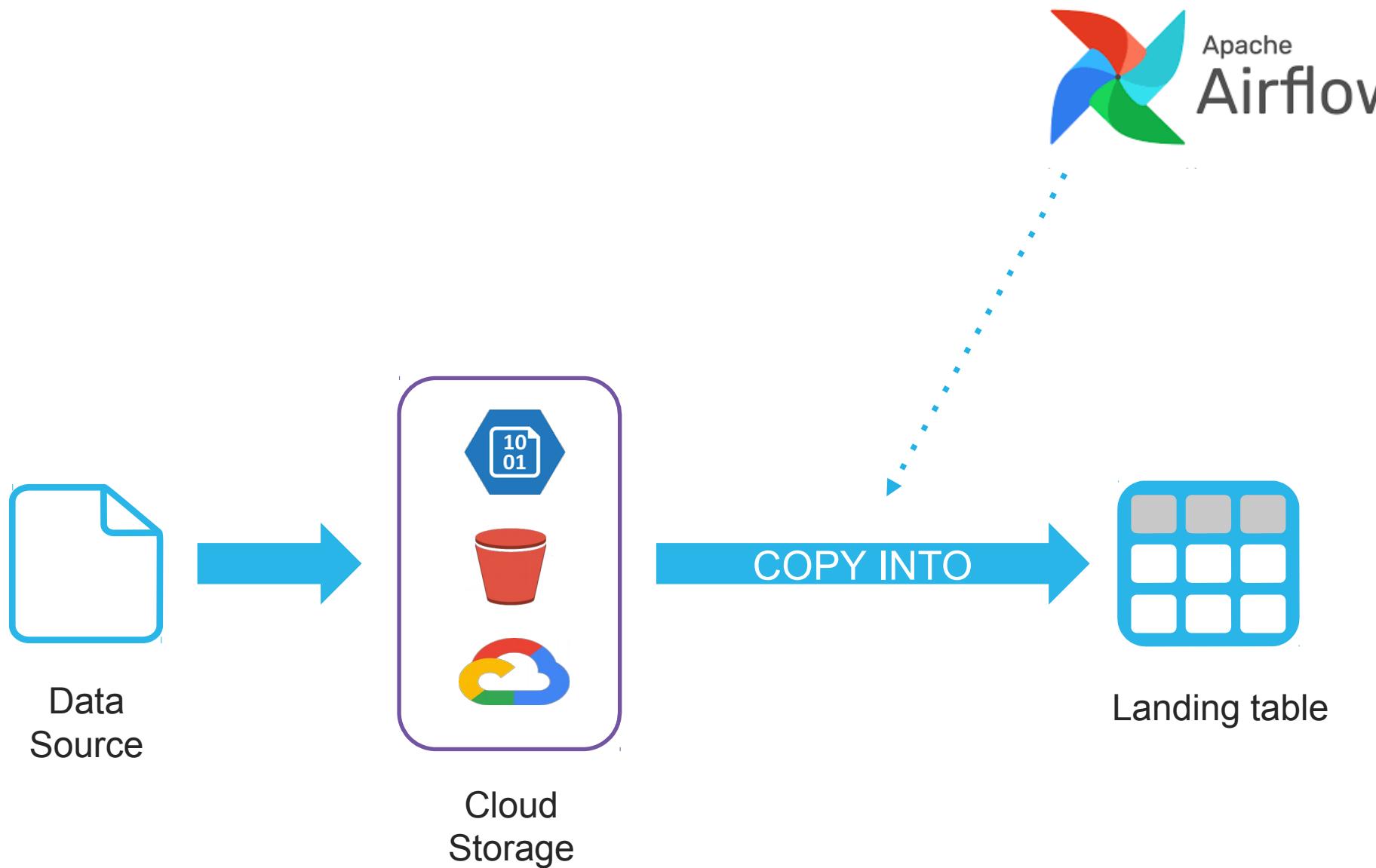
# TYPICAL MICRO-BATCH PIPELINE

## 2. Load data into cloud storage



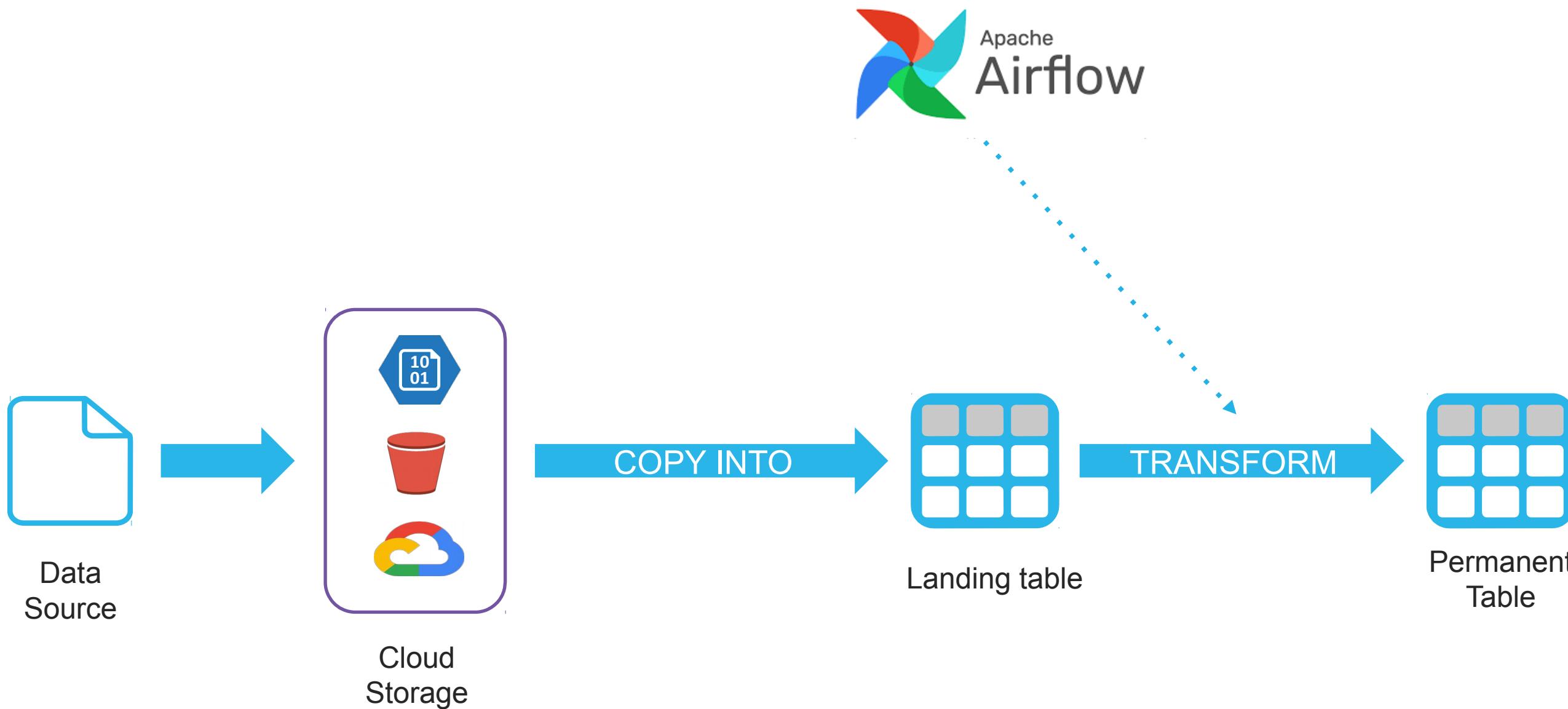
# TYPICAL MICRO-BATCH PIPELINE

## 3. Copy data into landing table



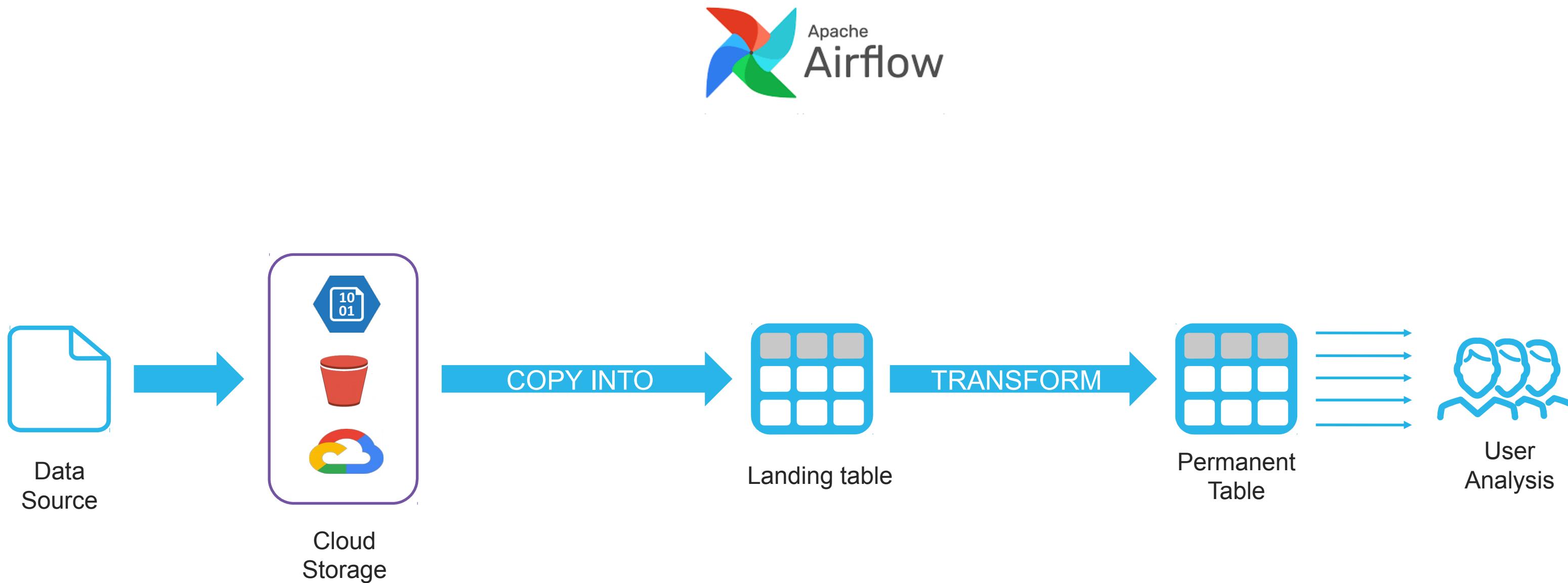
# TYPICAL MICRO-BATCH PIPELINE

4. Transform data into one or more permanent tables



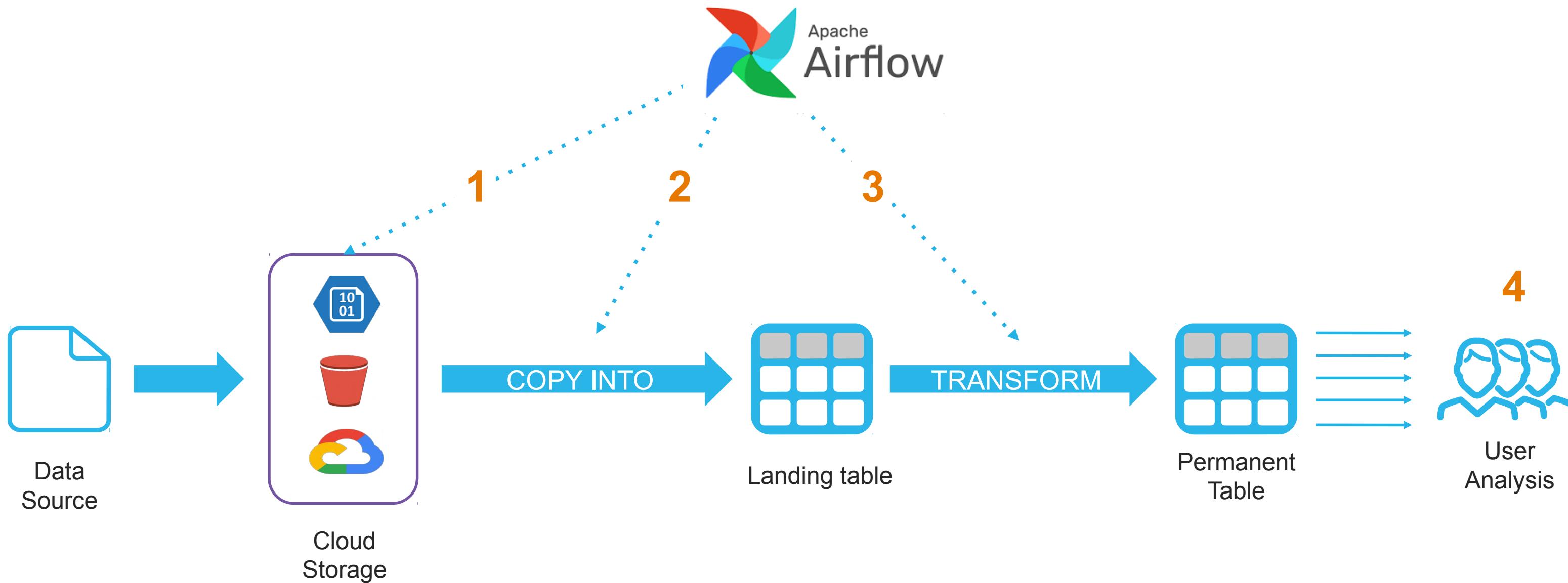
# TYPICAL MICRO-BATCH PIPELINE

- Data ready for end-users



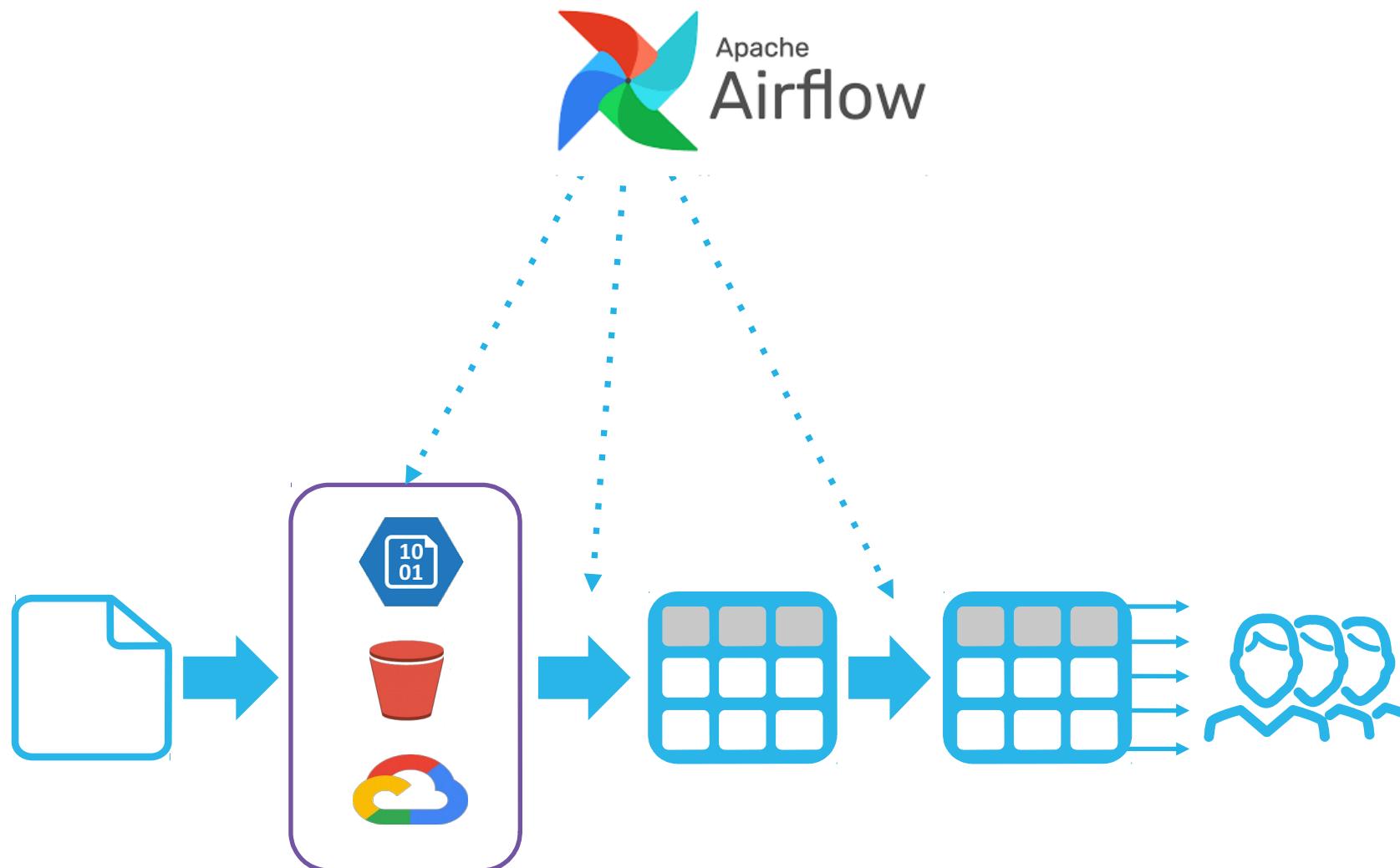
# TYPICAL MICRO-BATCH PIPELINE

- Not ideal for continuous data feeds requiring near real-time results



# MICRO-BATCH PIPELINE

## SUMMARY



### Advantages

- Good for complex ELT dependencies

### Drawbacks

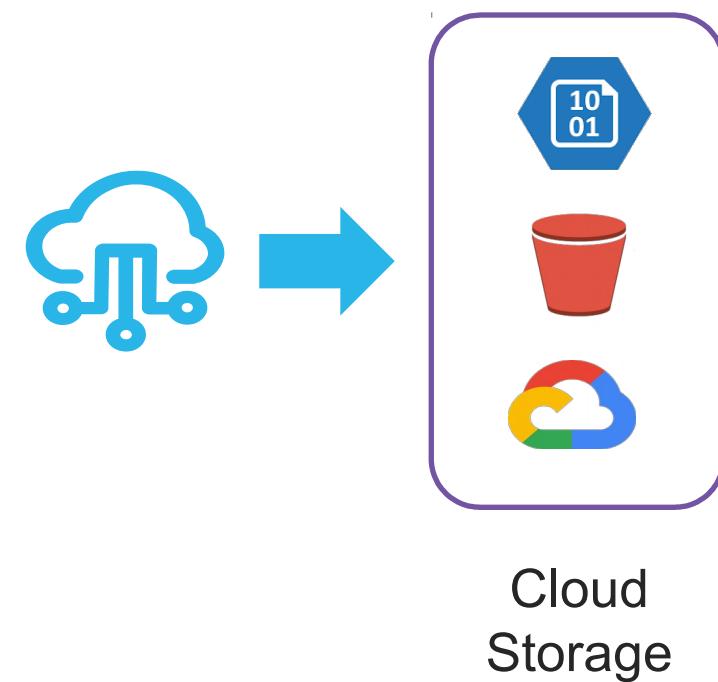
- Must complete before repeating
- Needs orchestration
  - Schedule execution
  - Coordinate each step
- Virtual Warehouses need sizing
  - Initial and ongoing sizing

# ALTERNATIVE: CONTINUOUS DATA PIPELINE



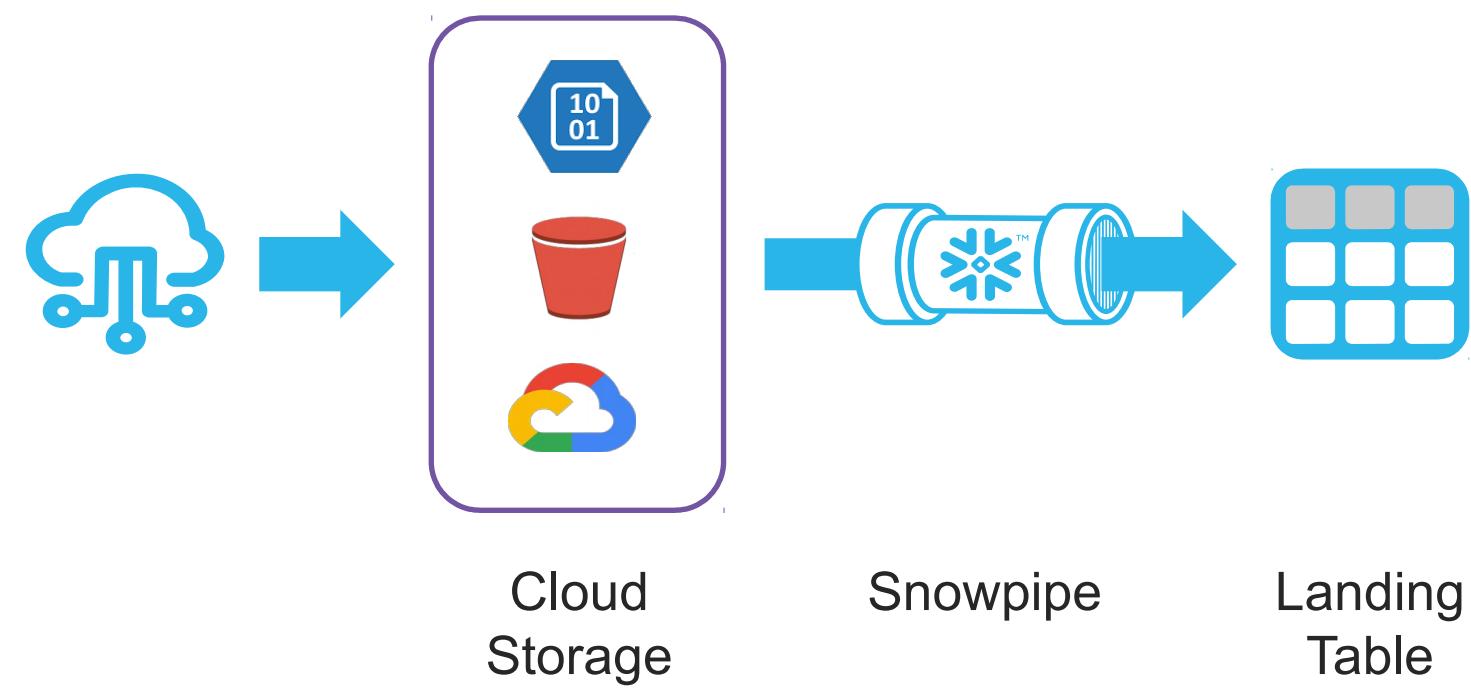
# AUTOMATED CONTINUOUS DATA PIPELINE

1. Data collected as files that are loaded to an external stage



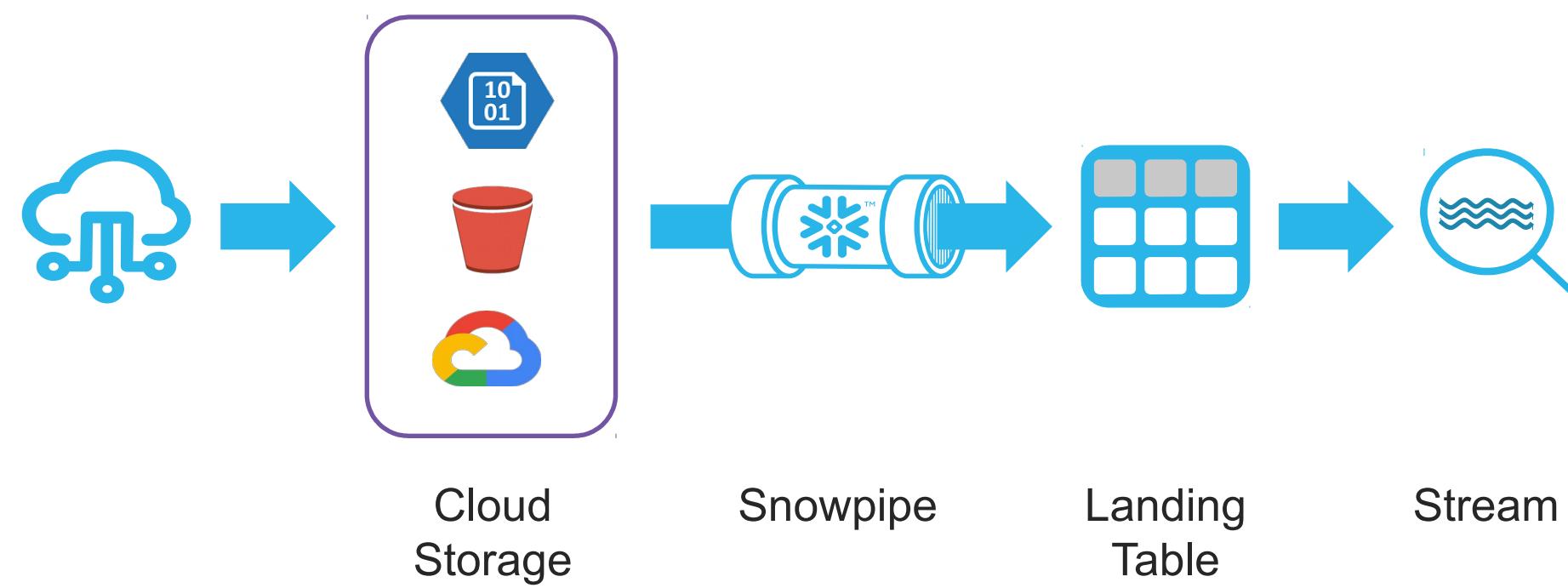
# AUTOMATED CONTINUOUS DATA PIPELINE

2. Snowpipe, receiving automated notification, loads new file to landing table



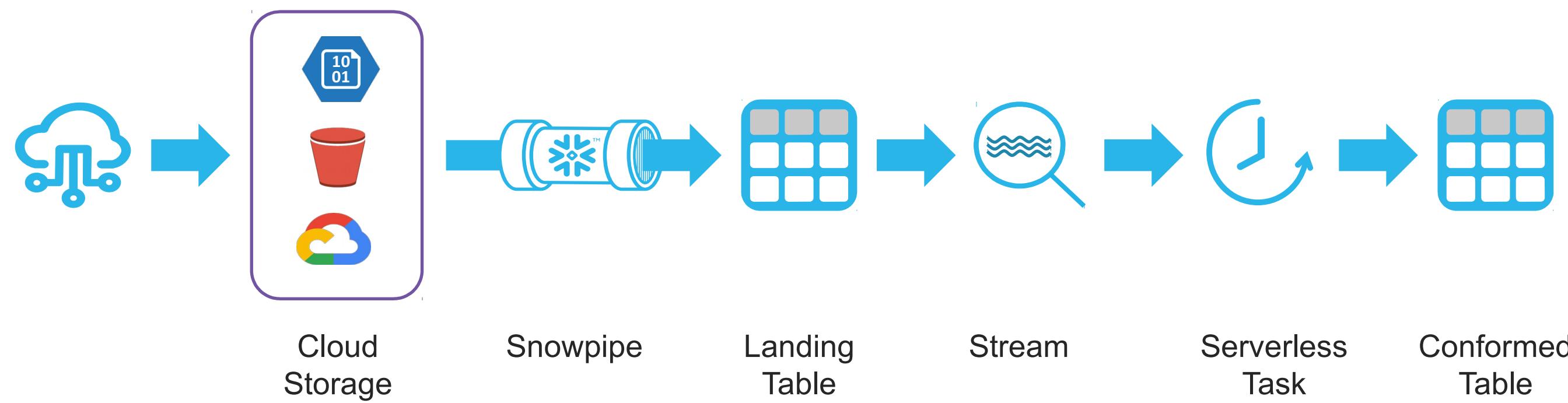
# AUTOMATED CONTINUOUS DATA PIPELINE

3. A Snowflake stream detects and tracks changes to the table



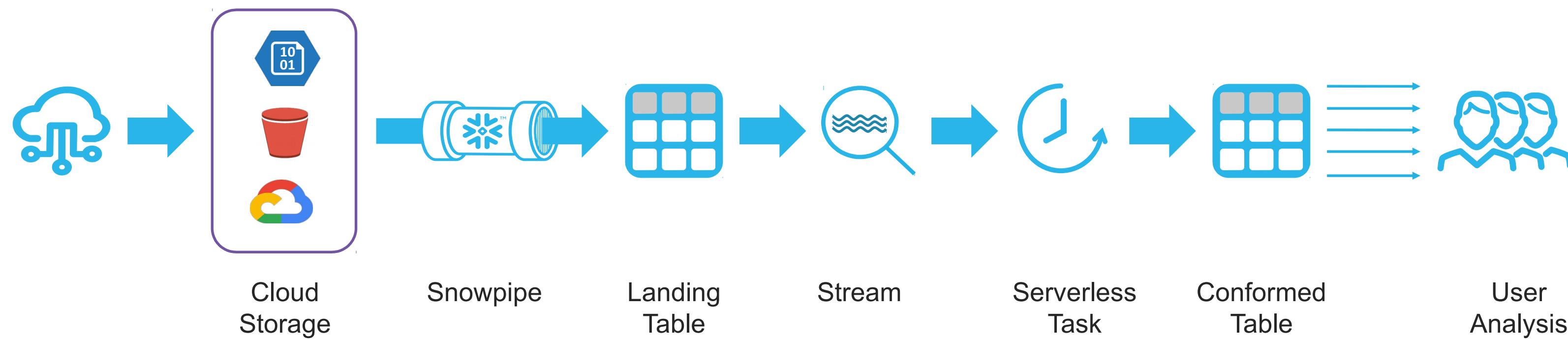
# AUTOMATED CONTINUOUS DATA PIPELINE

4. A serverless task periodically checks for new data to be transformed



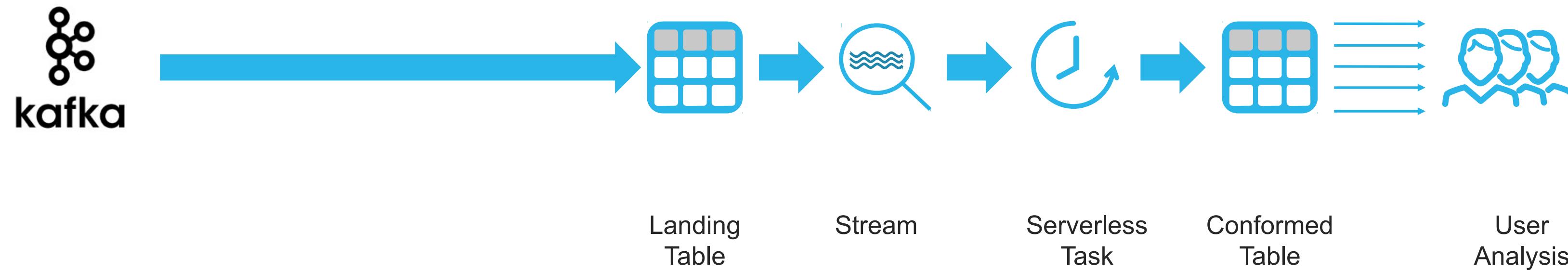
# AUTOMATED CONTINUOUS DATA PIPELINE

## 5. Results analyzed



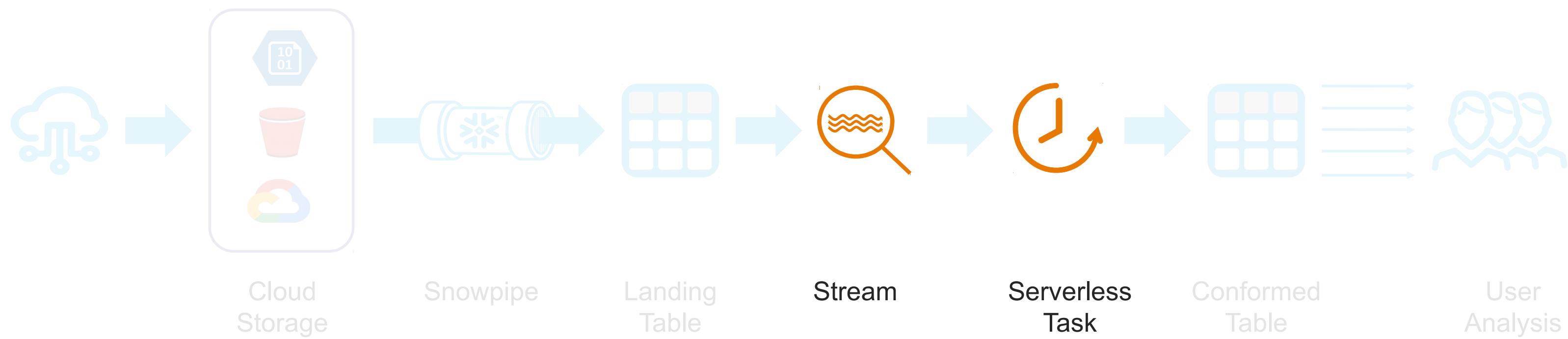
# AUTOMATED CONTINUOUS DATA PIPELINE

- Ingesting with the Kafka connector reduces complexity

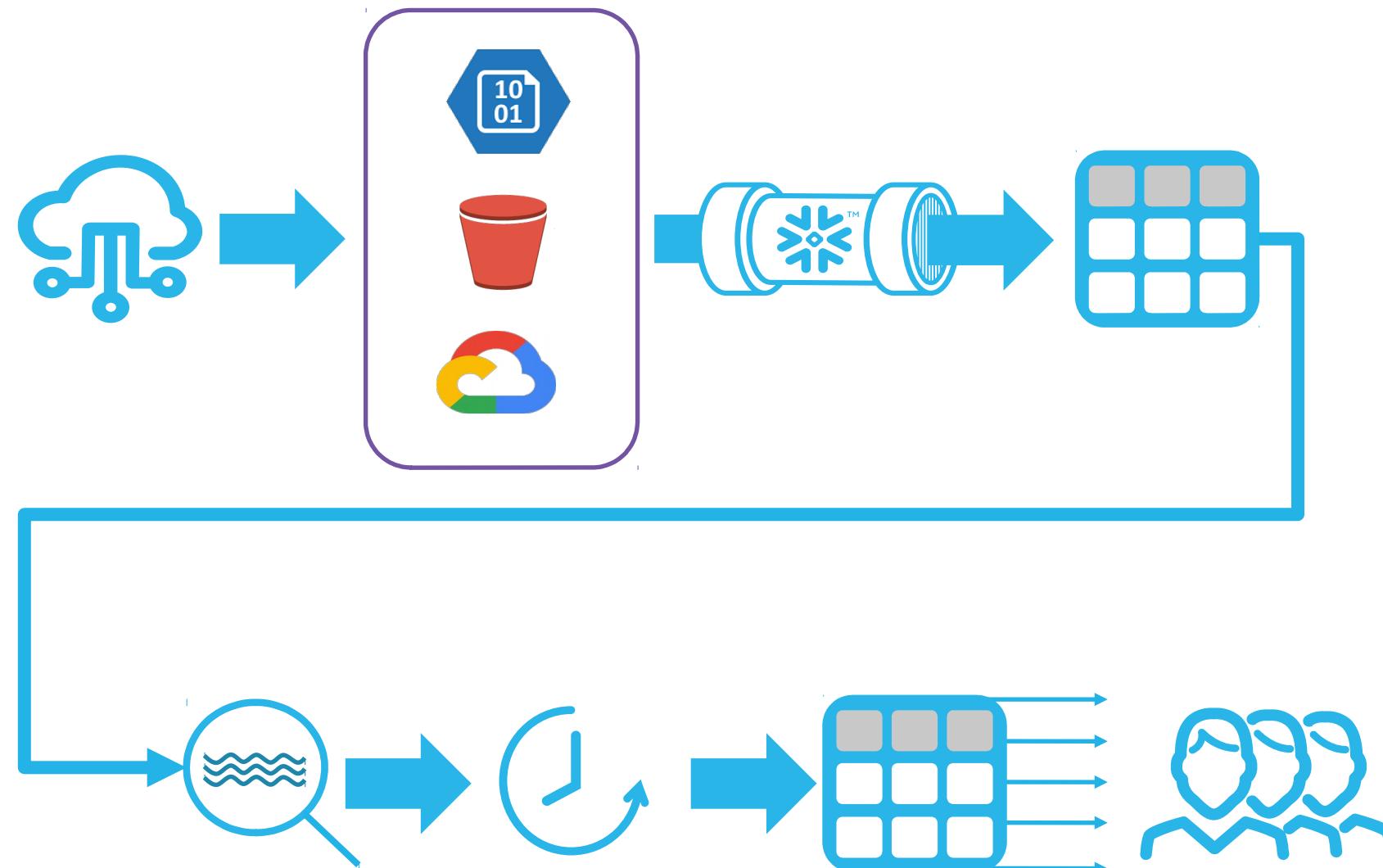


# STREAMS AND TASKS

- Stream: tracks changes to a table (automated CDC)
- Task: runs scheduled operations



# CONTINUOUS DATA PIPELINE SUMMARY



## Advantages

- Good for continuous/near real-time feeds
- Entirely automated pipeline
- Automated Change Data Capture (CDC)
- Auto-scaling (serverless) compute

## Drawbacks

- Not ideal for complex data dependencies
  - Huge number of data sources
  - Strict dependencies between tables

# STREAM DETAILS



# STREAM

## CHANGE DATA CAPTURE (CDC)

```
CREATE STREAM sales_str  
ON TABLE sales;
```



- Produces audit trail of changes
  - Guaranteed once-only delivery
- Create on a table or view
- Read only
- Types
  - Delta (default)
  - Append-only (tracks inserts only)

# CREATE AND QUERY A STREAM

## Table

```
CREATE TABLE sales_tbl(  
    product  VARCHAR  
, quantity NUMBER);
```

## Stream

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```



# CREATE AND QUERY A STREAM

## Table

```
CREATE TABLE sales_tbl (  
    product  VARCHAR  
, quantity NUMBER);
```

```
INSERT INTO sales_tbl VALUES  
    ('APPLE',      10),  
    ('ORANGE',     50),  
    ('BANANA',     0);
```

## Stream

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```



# CREATE AND QUERY A STREAM

## Table

```
CREATE TABLE sales_tbl (  
    product  VARCHAR  
, quantity NUMBER);
```

```
INSERT INTO sales_tbl VALUES  
    ('APPLE',      10),  
    ('ORANGE',     50),  
    ('BANANA',      0);
```

**SELECT \* FROM sales\_tbl;**

PROD	QTY
APPLE	10
ORANGE	50
BANANA	0

## Stream

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```



# CREATE AND QUERY A STREAM

## Table

```
CREATE TABLE sales_tbl (  
    product  VARCHAR  
, quantity NUMBER);
```

```
INSERT INTO sales_tbl VALUES  
( 'APPLE' ,      10) ,  
( 'ORANGE' ,     50) ,  
( 'BANANA' ,      0) ;
```

```
SELECT * FROM sales_tbl;
```

PROD	QTY
APPLE	10
ORANGE	50
BANANA	0

## Stream

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```

```
SELECT * FROM sales_str;
```

PROD	QTY	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
APPLE	10	INSERT	FALSE	ed0c3b7baa0f969d1f
ORANGE	50	INSERT	FALSE	46ee715f49b071f26e
BANANA	0	INSERT	FALSE	3d324888847a57a1c



# CONTENTS OF A STREAM

PROD	...	QTY	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
APPLE		10	INSERT	FALSE	ed0c3b7baa0f969d1f
ORANGE		50	INSERT	FALSE	46ee715f49b071f26e
BANANA		0	INSERT	FALSE	3d324888847a57a1c

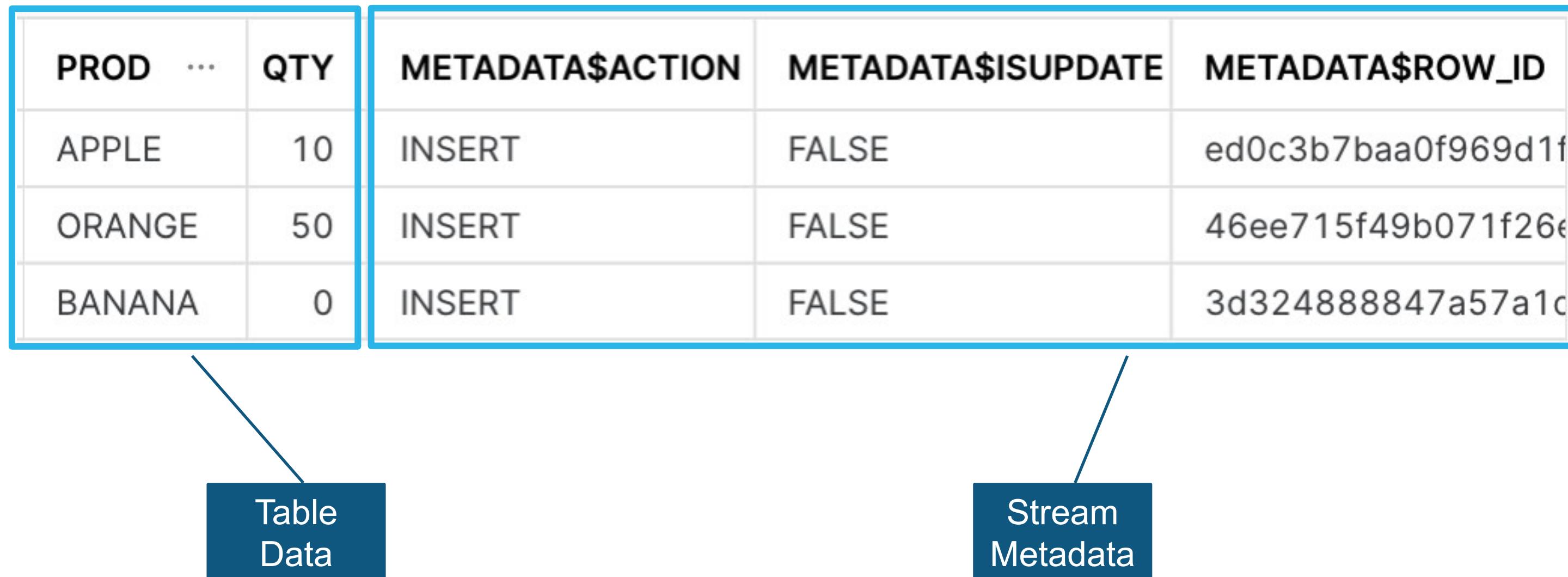
The diagram illustrates the structure of a stream. It is composed of two distinct parts: 'Table Data' and 'Stream Metadata'. The 'Table Data' is represented by a standard table with columns: PROD, ..., QTY. The 'Stream Metadata' is represented by a table with columns: METADATA\$ACTION, METADATA\$ISUPDATE, and METADATA\$ROW\_ID. Both tables are enclosed within a single large blue-bordered container.

Table Data

Stream Metadata

# CHANGES REFLECTED IN A STREAM

PROD	QTY	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID	
APPLE	10	DELETE	FALSE	641944ce81a4573ad	<span>DELETE</span>
ORANGE	50	DELETE	TRUE	a64a89140ec306f37.	<span>UPDATE</span>
ORANGE	30	INSERT	TRUE	a64a89140ec306f37.	
PEACH	35	INSERT	FALSE	6bdd088a987d1ae6d	<span>INSERT</span>



# QUERY A STREAM

- Find inserts: 

```
SELECT * FROM sales_str  
WHERE metadata$action = 'INSERT'  
AND metadata$isupdate = 'FALSE';
```
- Find deletes: 

```
SELECT * FROM sales_str  
WHERE metadata$actions = 'DELETE'  
AND metadata$isupdate = 'FALSE';
```
- Find updates: 

```
SELECT * FROM sales_str  
WHERE metadata$action = 'INSERT'  
AND metadata$isupdate = 'TRUE';
```



# HOW STREAMS WORK



# CREATE TABLE

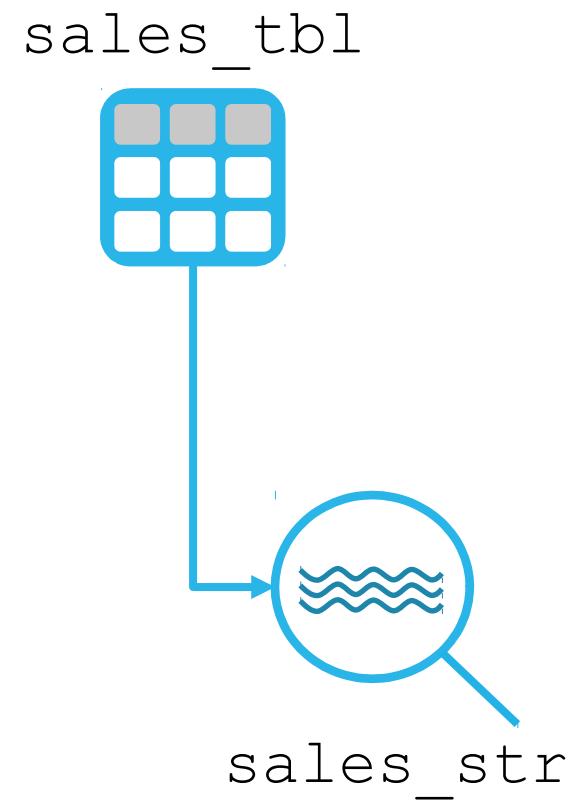
```
CREATE TABLE sales_tbl (
    product    VARCHAR
,   quantity   NUMBER);
```

sales\_tbl



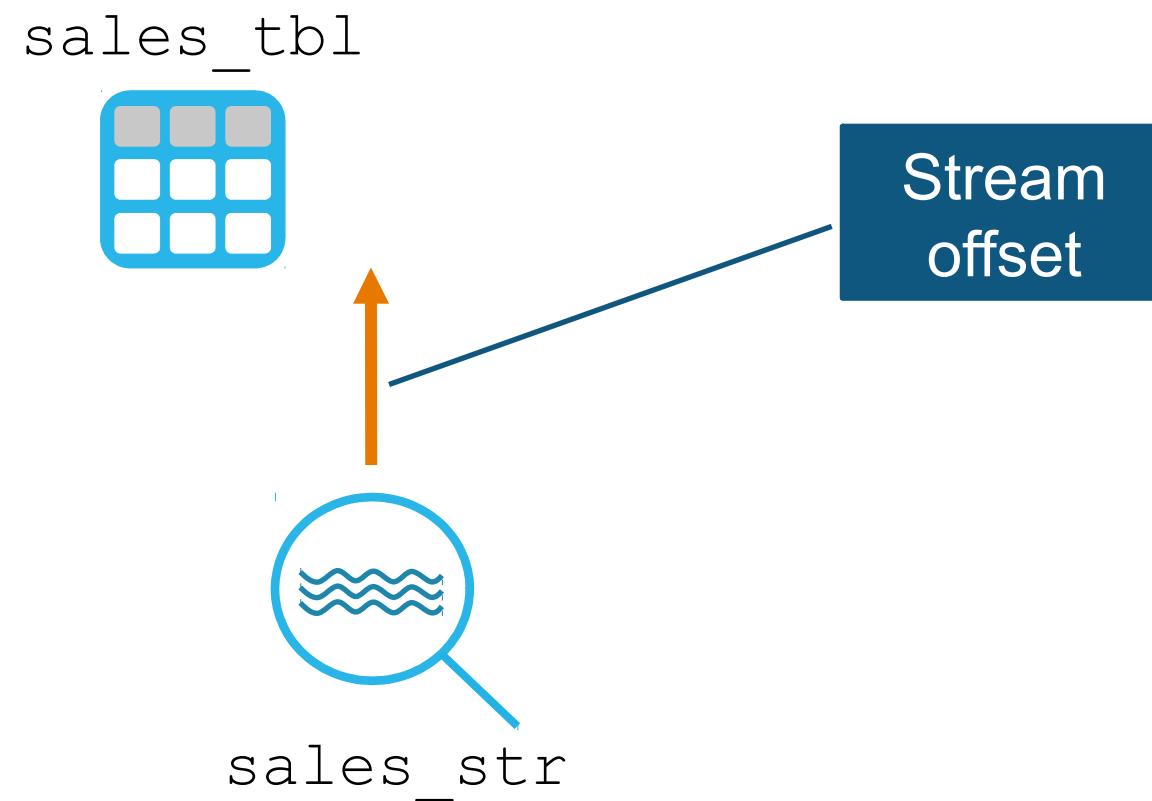
# CREATE STREAM ON TABLE

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```



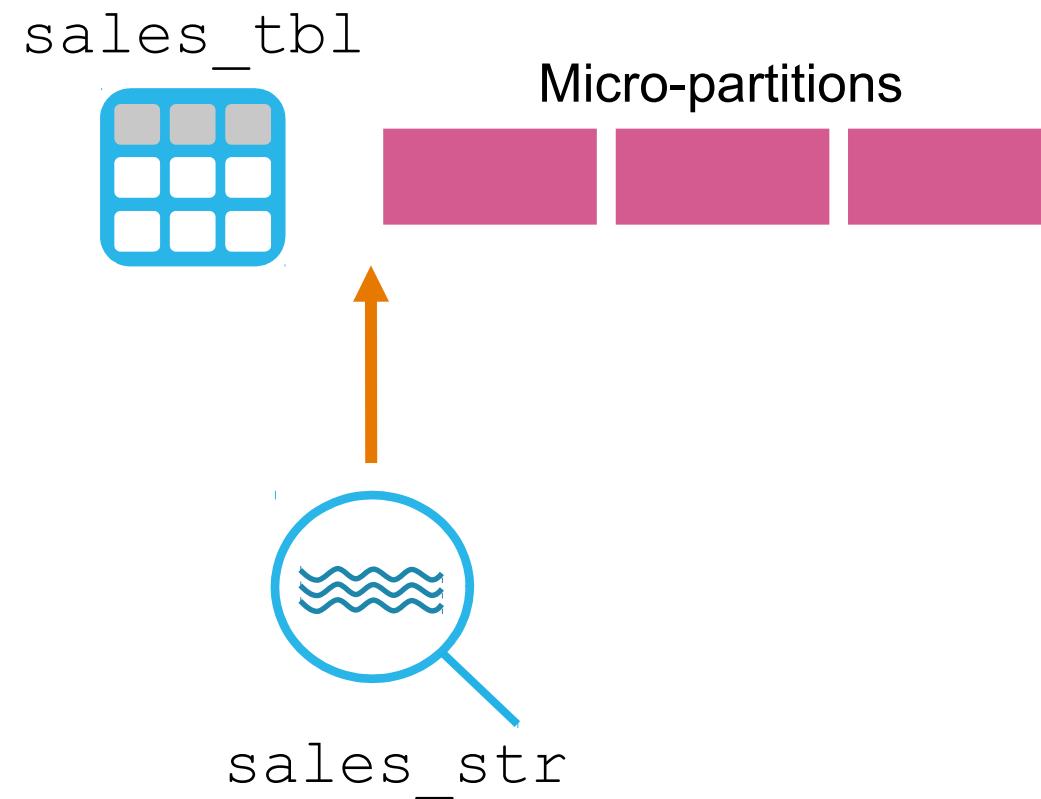
# STREAM OFFSET

```
CREATE STREAM sales_str  
ON TABLE sales_tbl;
```



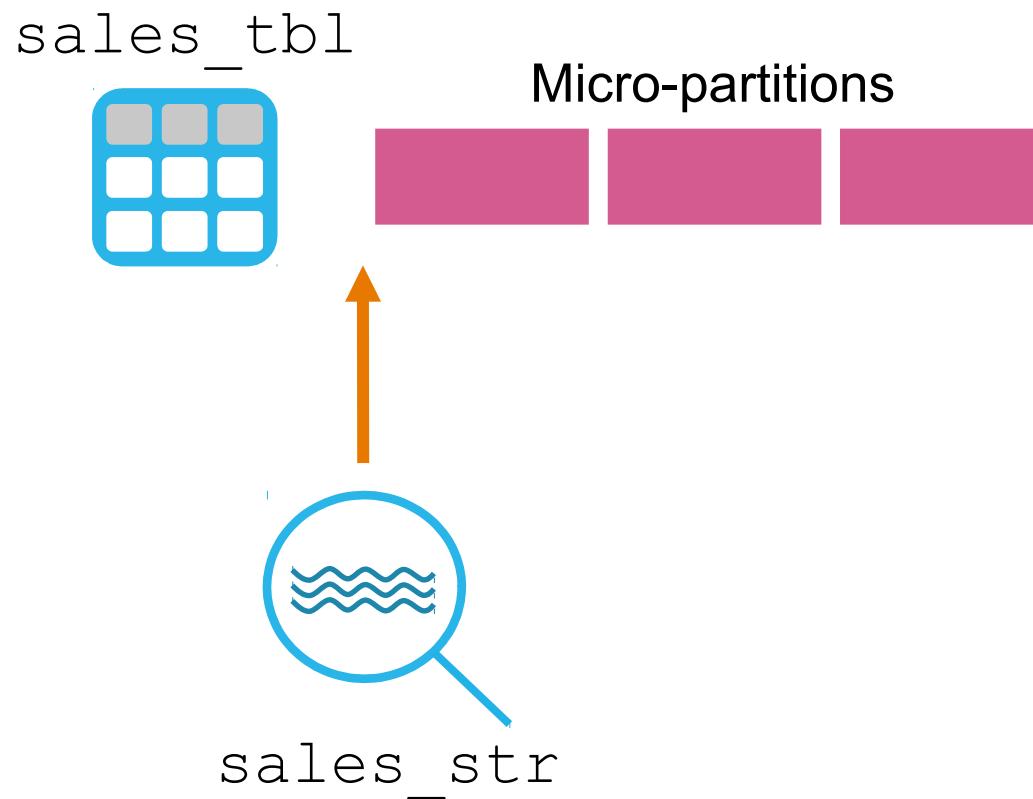
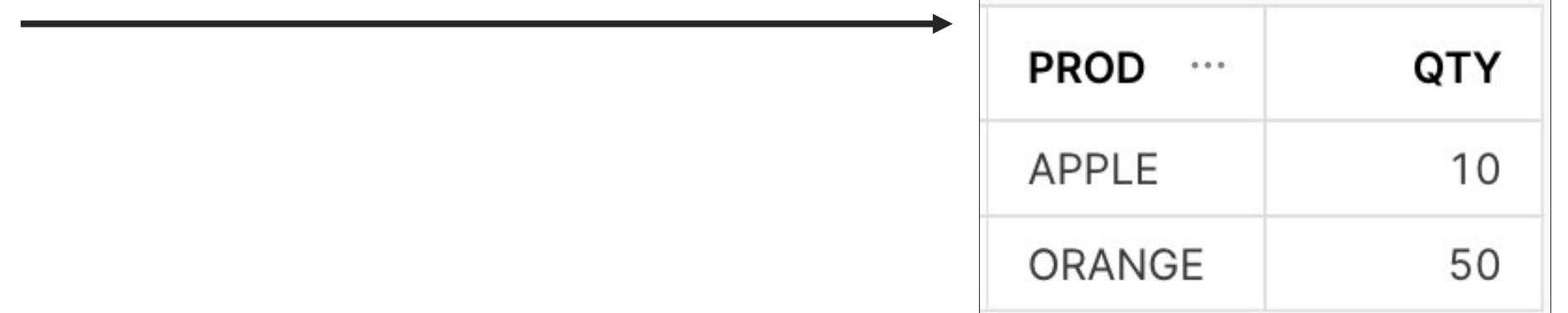
# UPDATE TABLE

```
INSERT INTO sales_tbl VALUES  
('APPLE', 10),  
('ORANGE', 50);
```



# QUERY TABLE

```
SELECT *  
FROM sales_tbl;
```

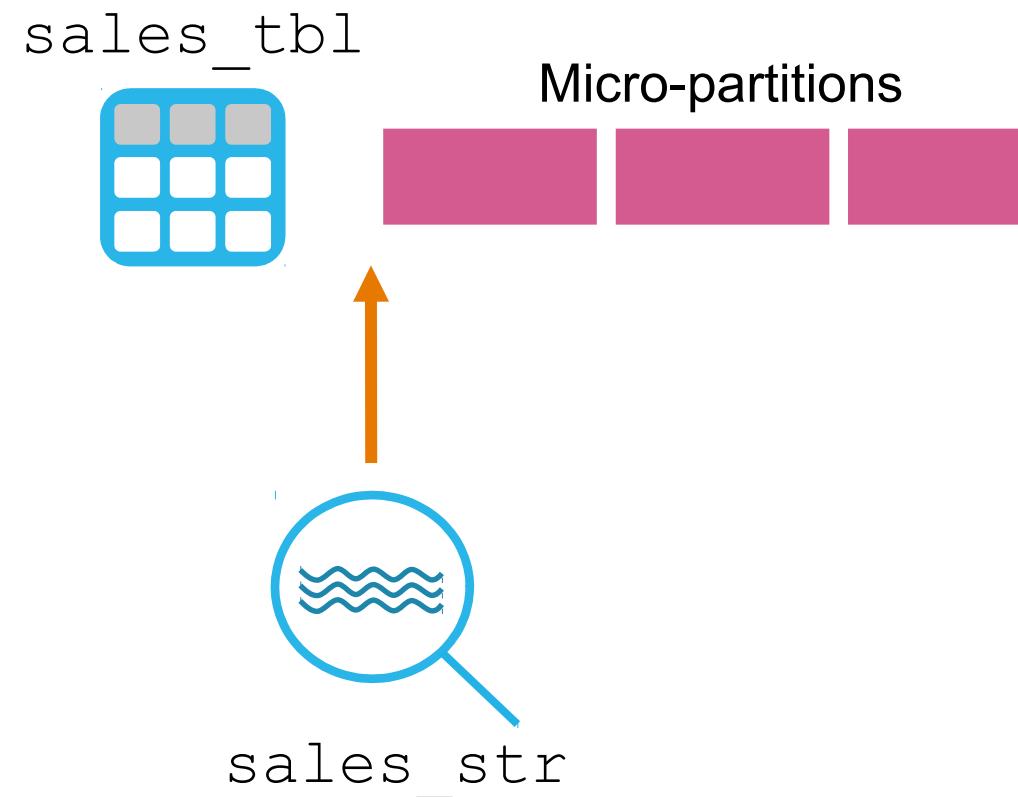


# QUERY STREAM

```
SELECT *  
FROM sales_str;
```



PROD	QTY	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
APPLE	10	INSERT	FALSE	dc237edaa36bbecdfc
ORANGE	50	INSERT	FALSE	f12126099a7c938cc

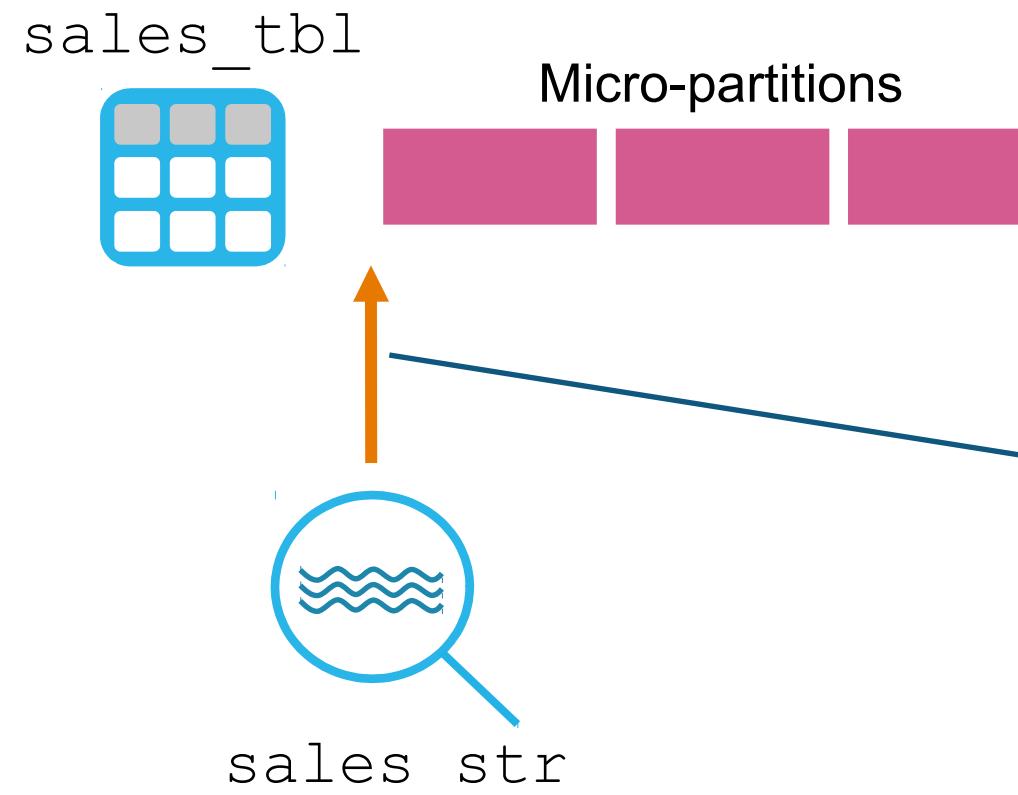


# QUERY STREAM

```
SELECT *  
FROM sales_str;
```



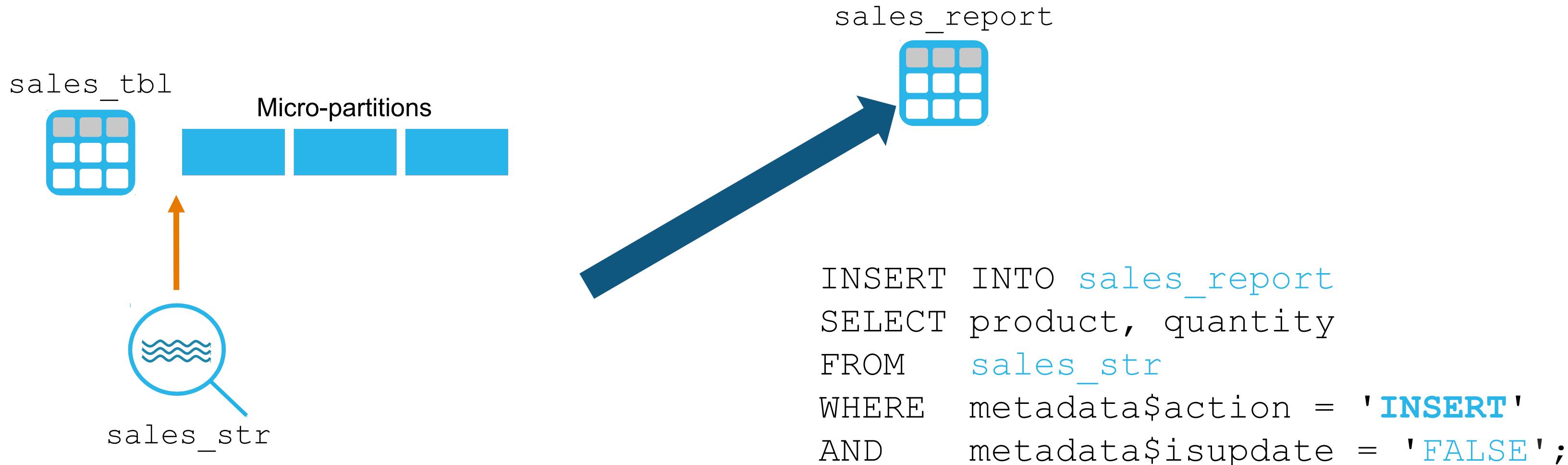
PROD	QTY	METADATA\$ACTION	METADATA\$ISUPDATE	METADATA\$ROW_ID
APPLE	10	INSERT	FALSE	dc237edaa36bbecdfc
ORANGE	50	INSERT	FALSE	f12126099a7c938cc



- Query as many times as you want
- Stream offset does not change

# CONSUME A STREAM

- Use stream's information for DML against another object
- Consuming stream moves offset



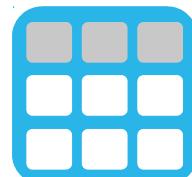
# TABLE HAS NOT CHANGED

```
SELECT *  
FROM sales_tbl;
```



PROD	...	QTY
APPLE		10
ORANGE		50

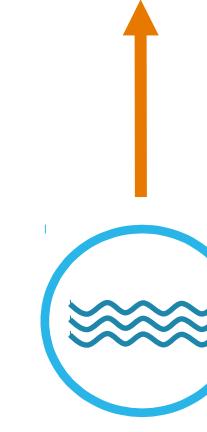
sales\_tbl



Micro-partitions

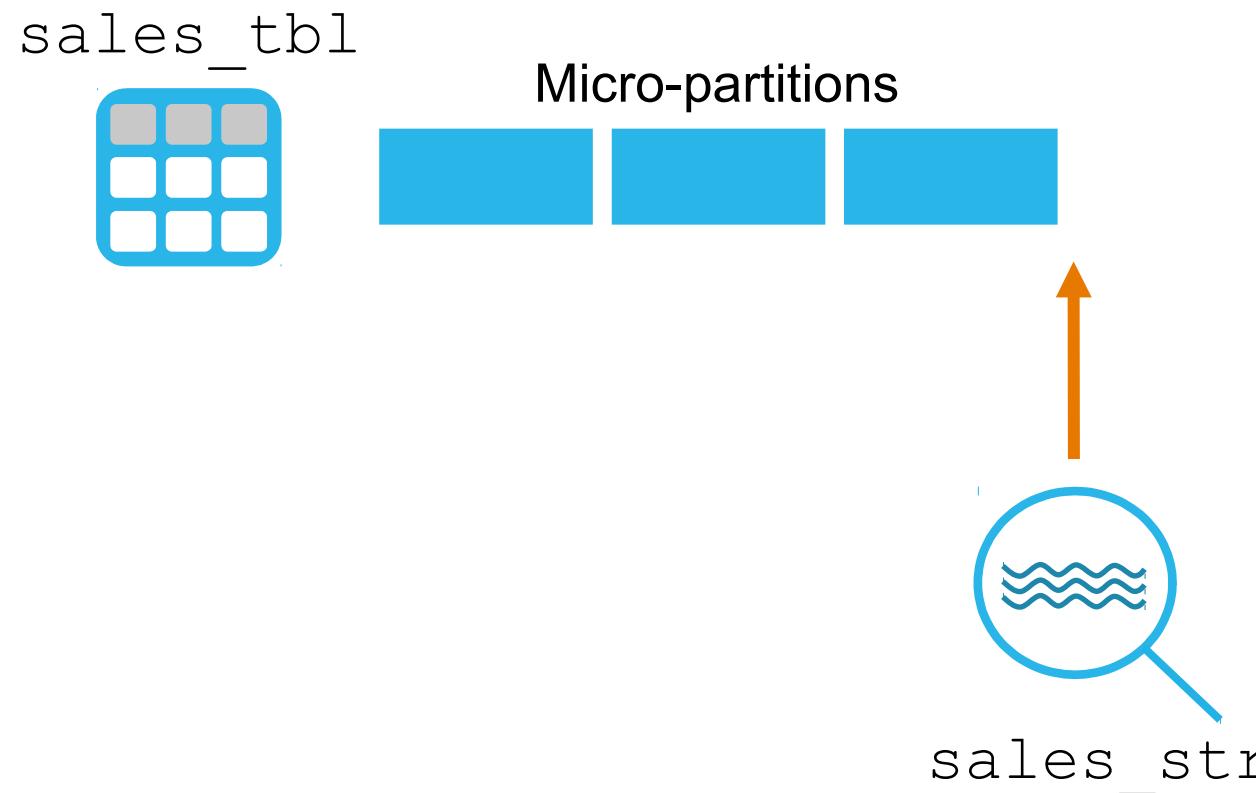
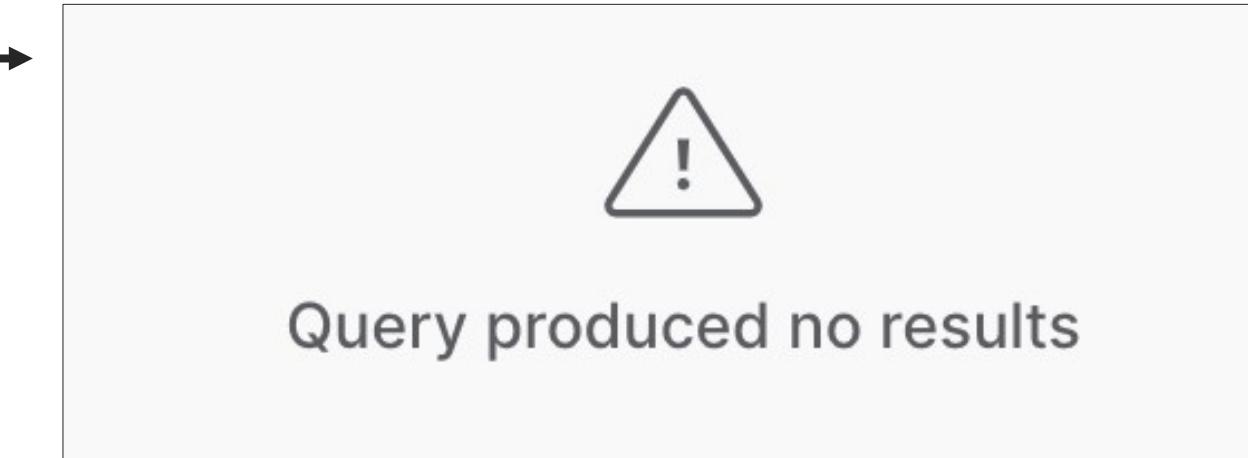


sales\_str



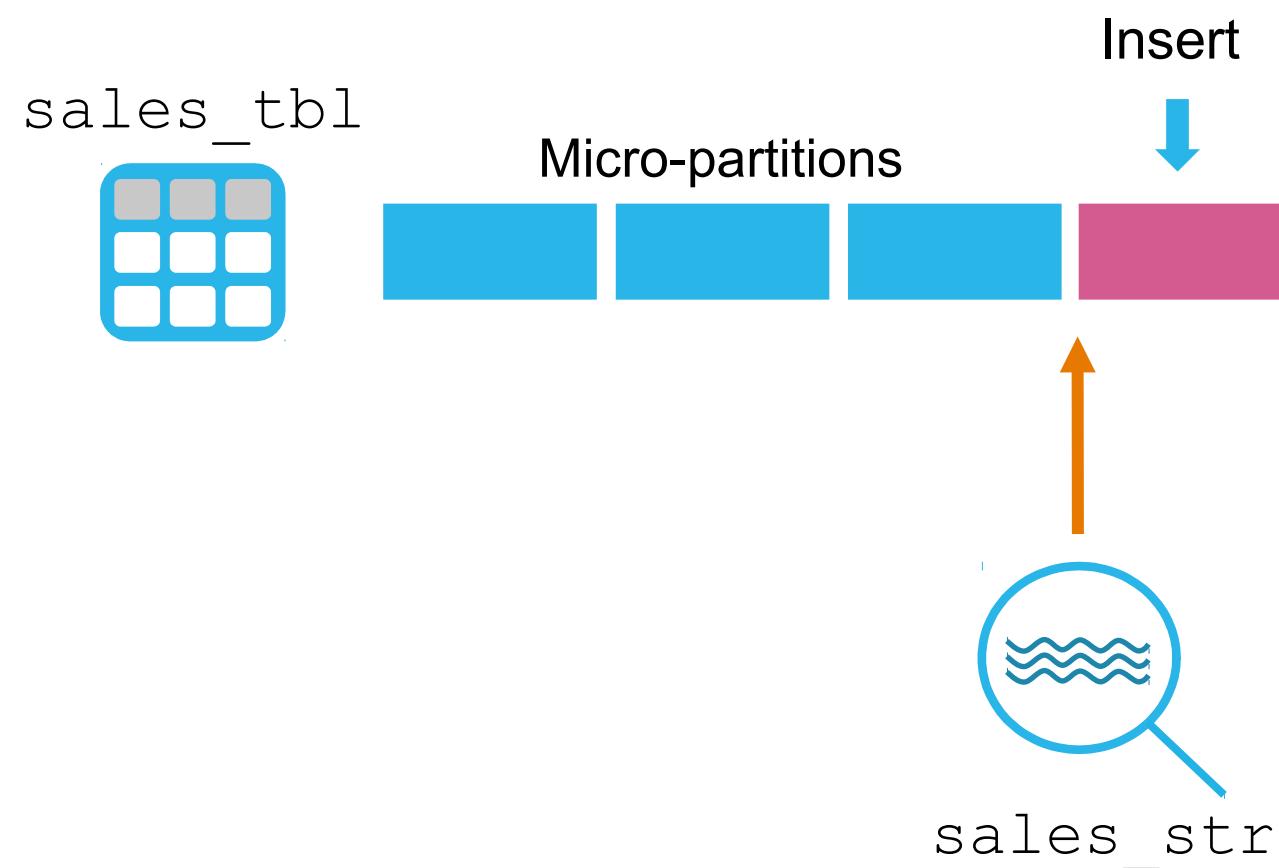
# STREAM IS NOW EMPTY

```
SELECT *\nFROM sales_str;
```



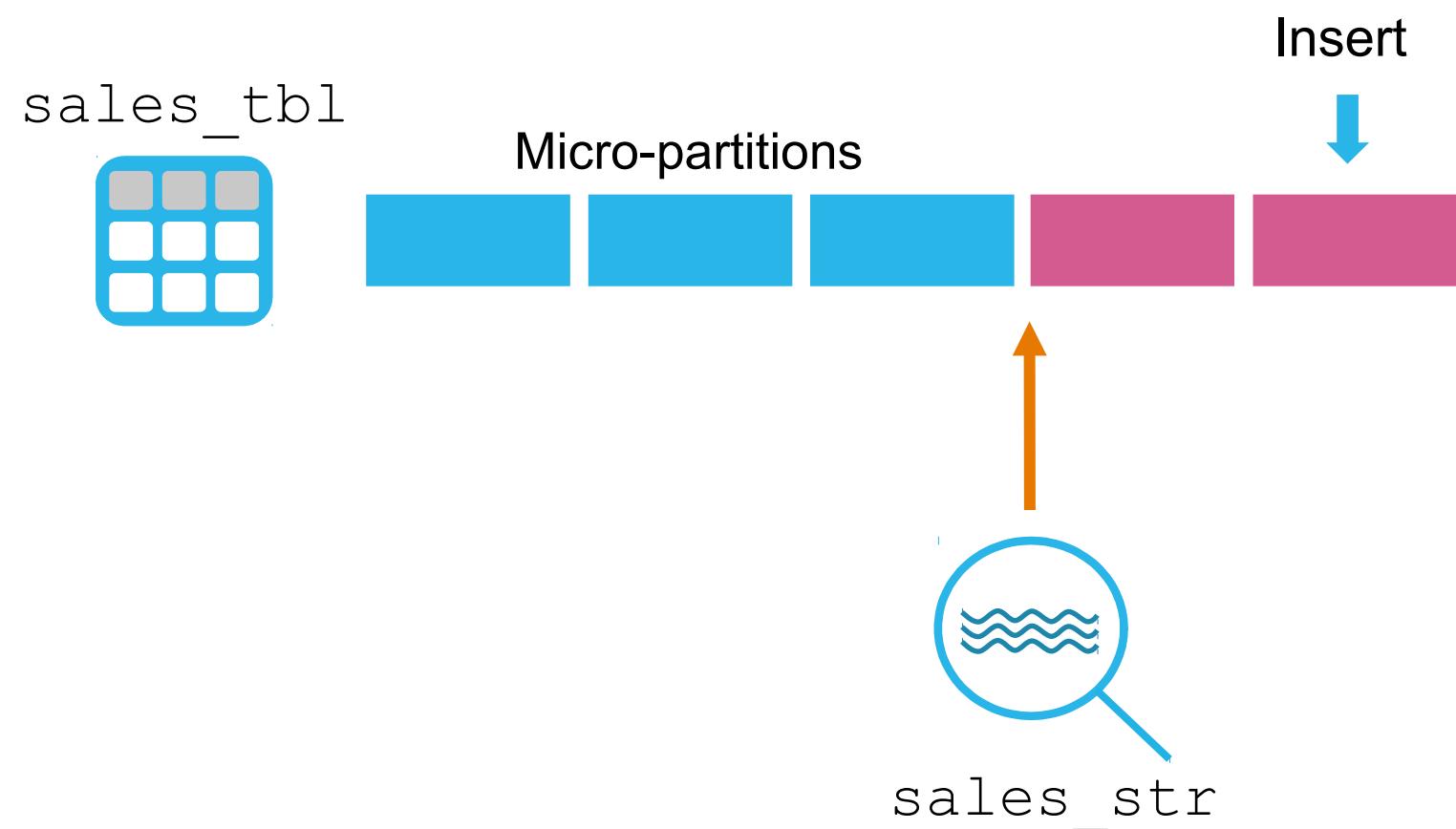
# ADD OR UPDATE TABLE DATA

```
INSERT INTO sales_tbl VALUES  
('BANANA', 10);
```



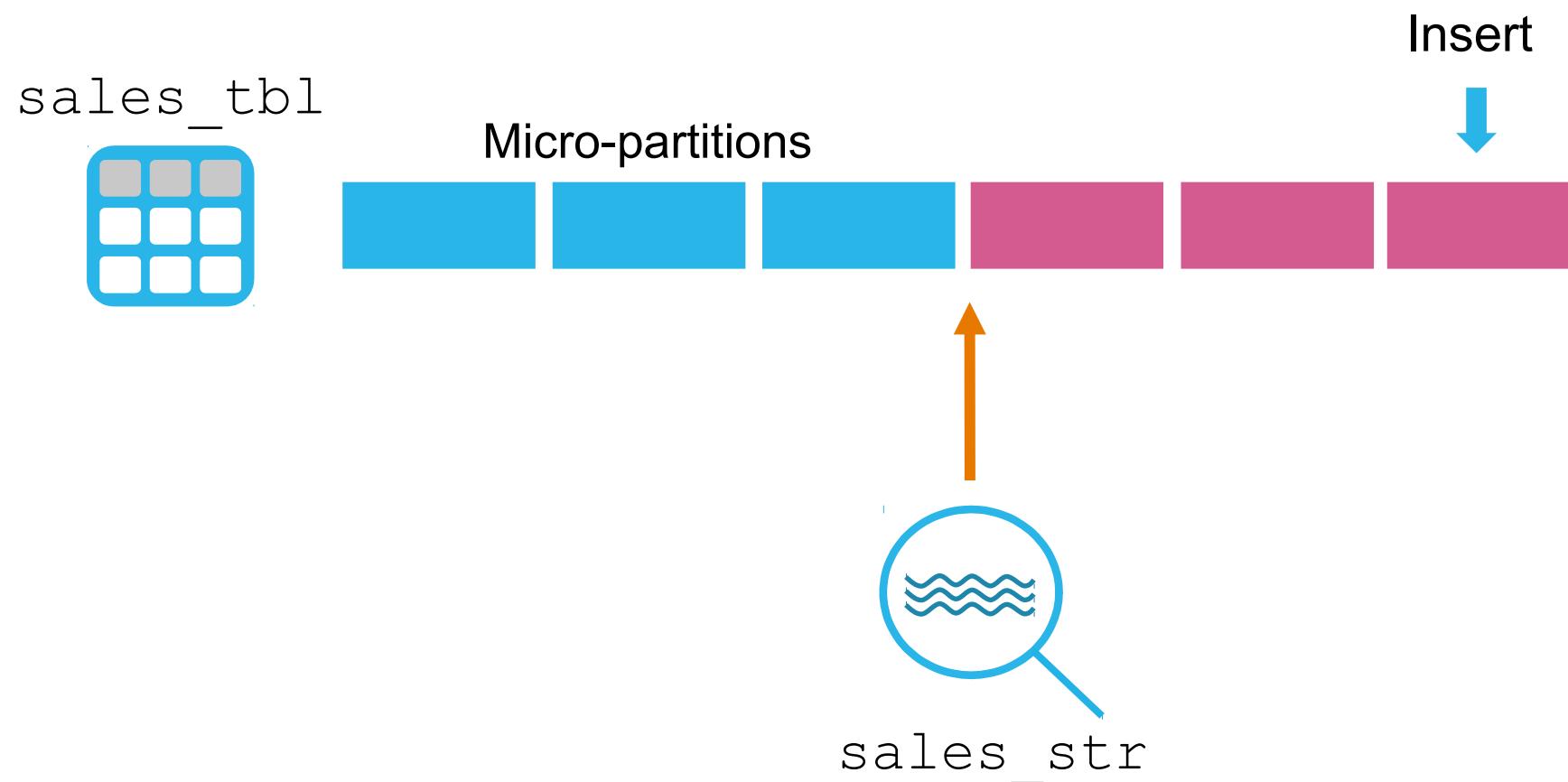
# ADD OR UPDATE TABLE DATA

```
INSERT INTO sales_tbl VALUES  
('PEAR', 43);
```



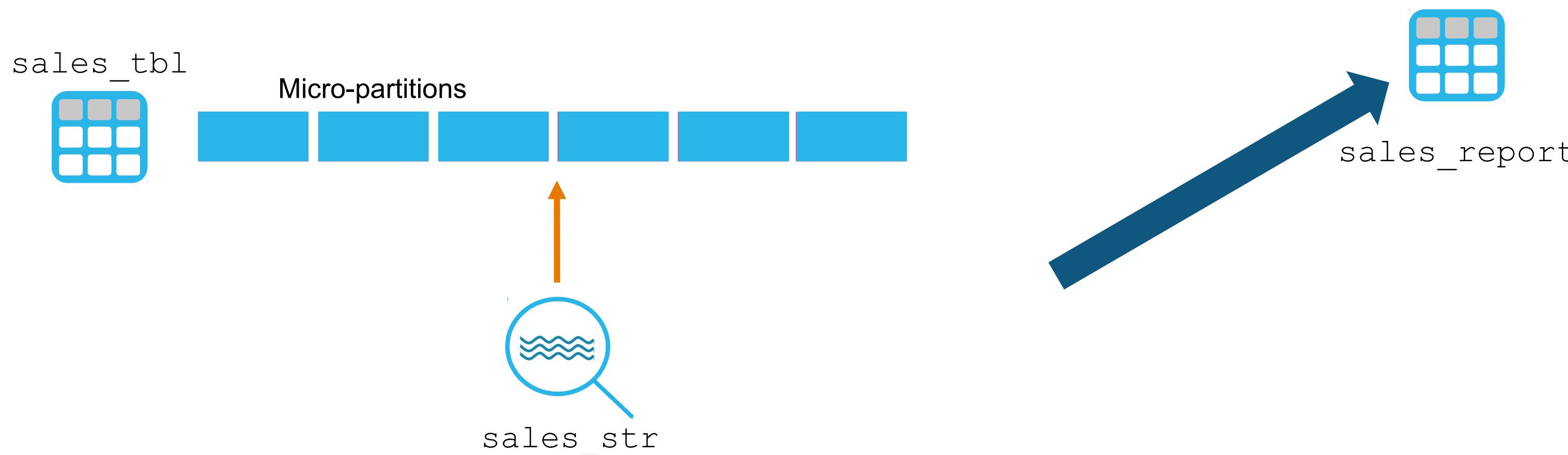
# ADD OR UPDATE TABLE DATA

```
INSERT INTO sales_tbl VALUES  
(...);
```



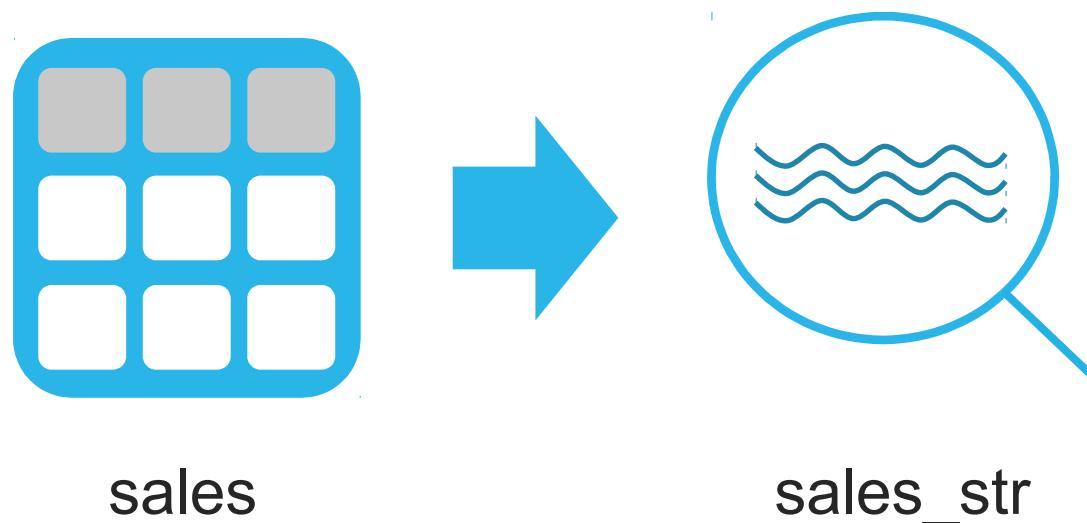
# CONSUME THE STREAM

```
INSERT INTO sales_report  
SELECT product, quantity  
FROM sales_str  
WHERE metadata$action = 'INSERT'  
AND metadata$isupdate = 'FALSE';
```



# APPEND-ONLY STREAMS

```
CREATE STREAM sales_str  
ON TABLE sales  
APPEND_ONLY = 'TRUE';
```



- Can track inserts only
  - With APPEND\_ONLY clause
- Query performance is better against append-only streams
  - Use delta only when necessary

# SQL CHANGES CLAUSE

## STREAM ALTERNATIVE

```
ALTER TABLE sales_tbl  
  SET CHANGE_TRACKING = TRUE;
```

<changes to table>

```
SELECT *  
FROM sales_tbl  
CHANGES (INFORMATION => DEFAULT)  
AT (TIMESTAMP => <timestamp>);
```

```
SELECT *  
FROM sales_tbl  
CHANGES (INFORMATION => APPEND_ONLY)  
AT (TIMESTAMP => <timestamp>);
```

- Track changes without creating a stream
  - Adds same metadata to table
- DML does not consume changes
- Once-only delivery not guaranteed
  - Can look back in time
- Query with CHANGES clause
  - All transactions, or appends only

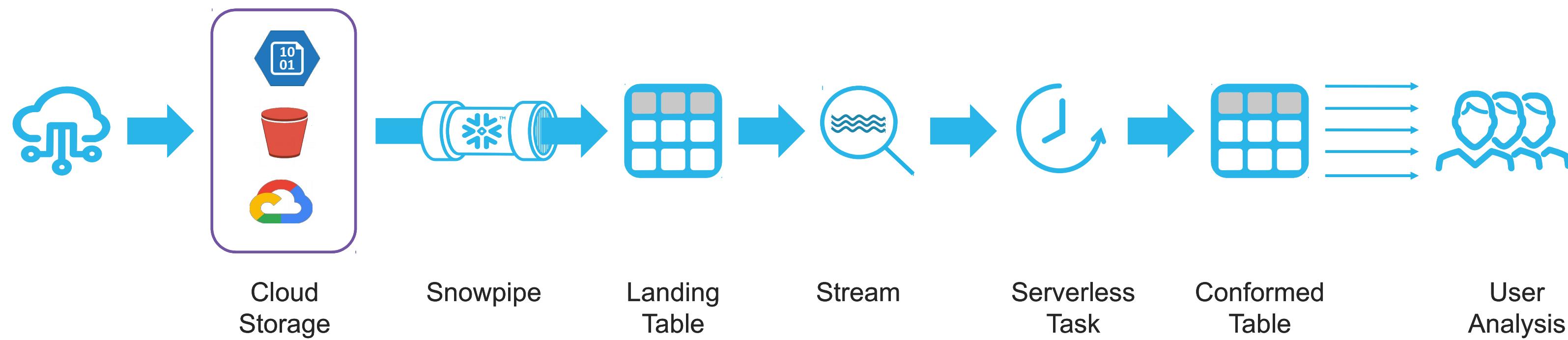


# COMMON USE CASES



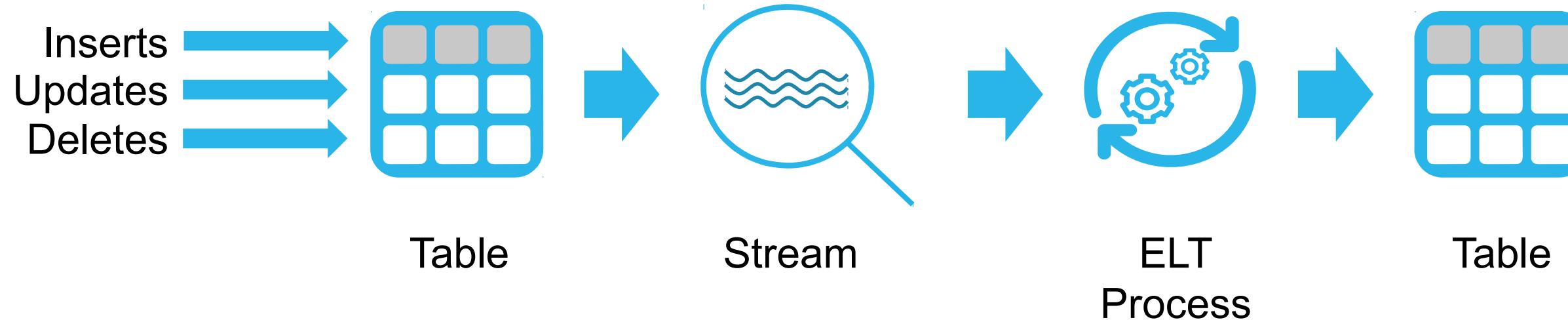
# AUTOMATE DATA PIPELINES

- Use Snowpipe, streams, and tasks to automate data pipelines



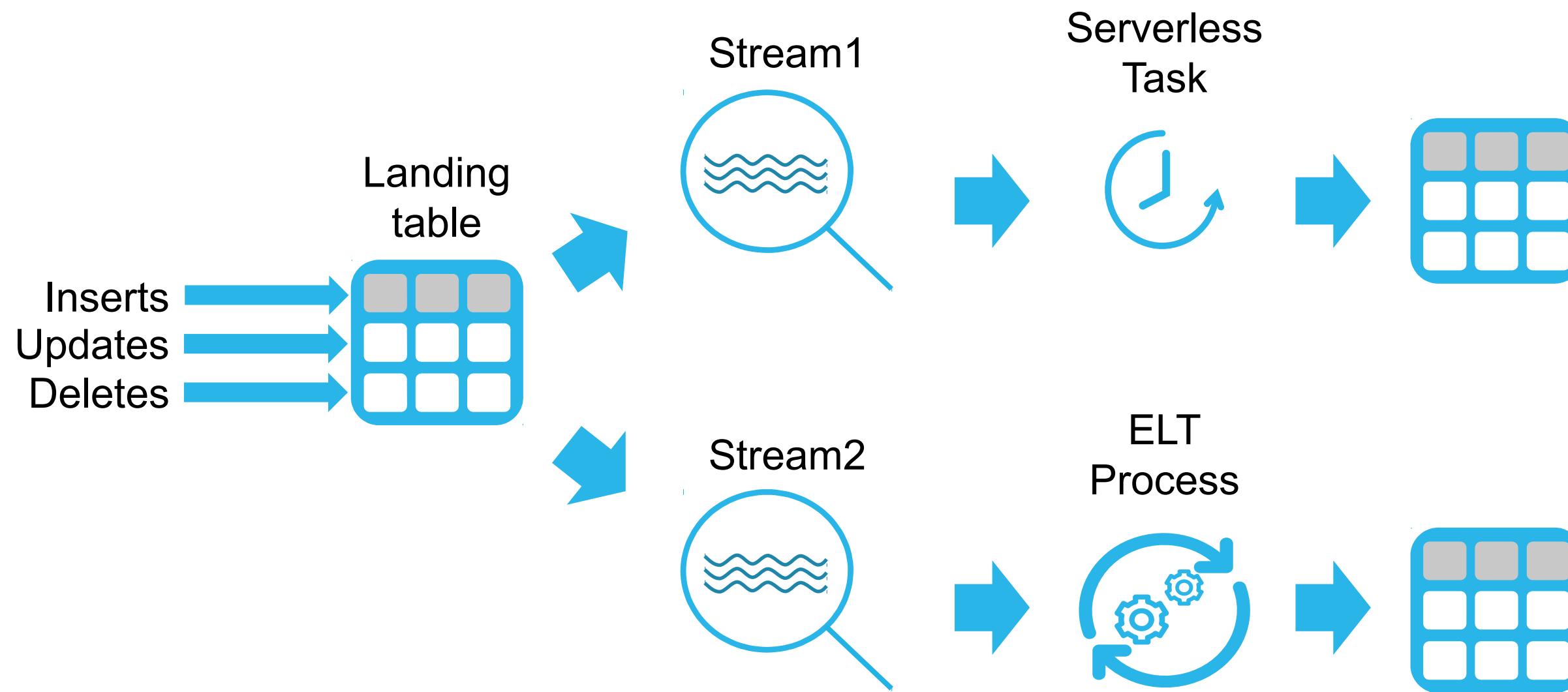
# SIMPLIFIED CHANGE DATA CAPTURE

- Streams can be used without tasks
- Automatically identify CDC on a table
- An ELT process executes to transform data

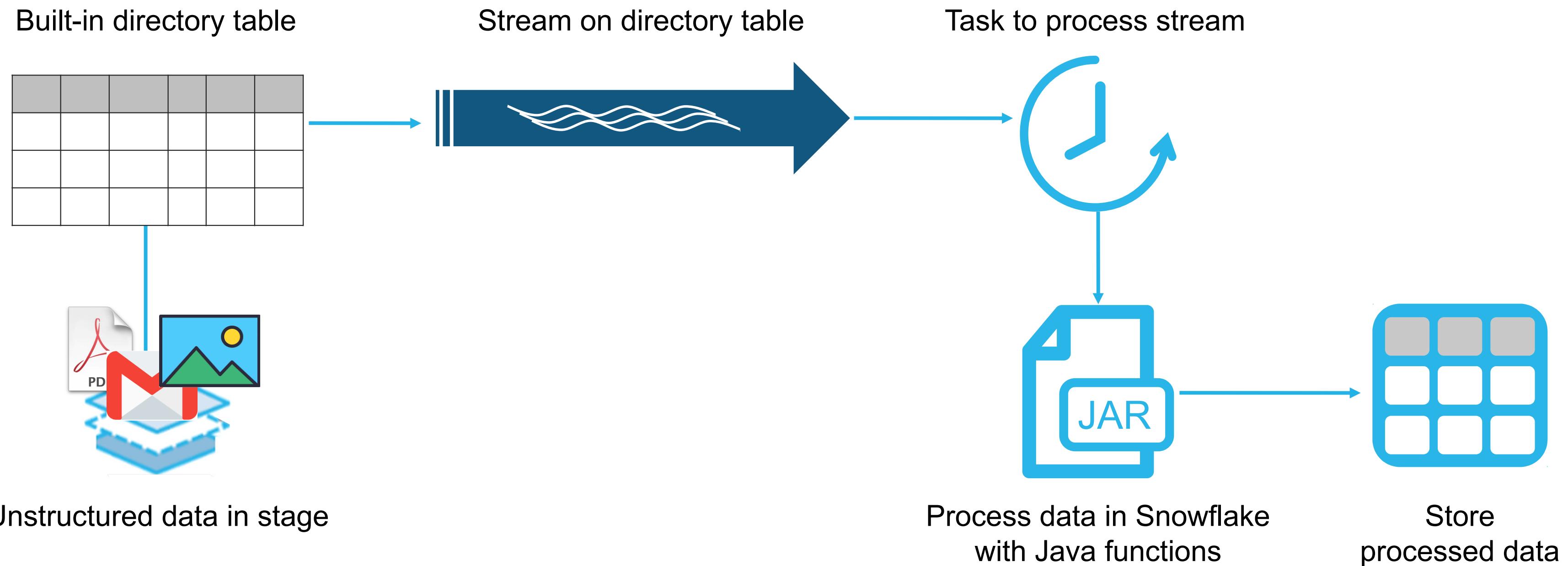


# OUTPUT TO MULTIPLE TABLES

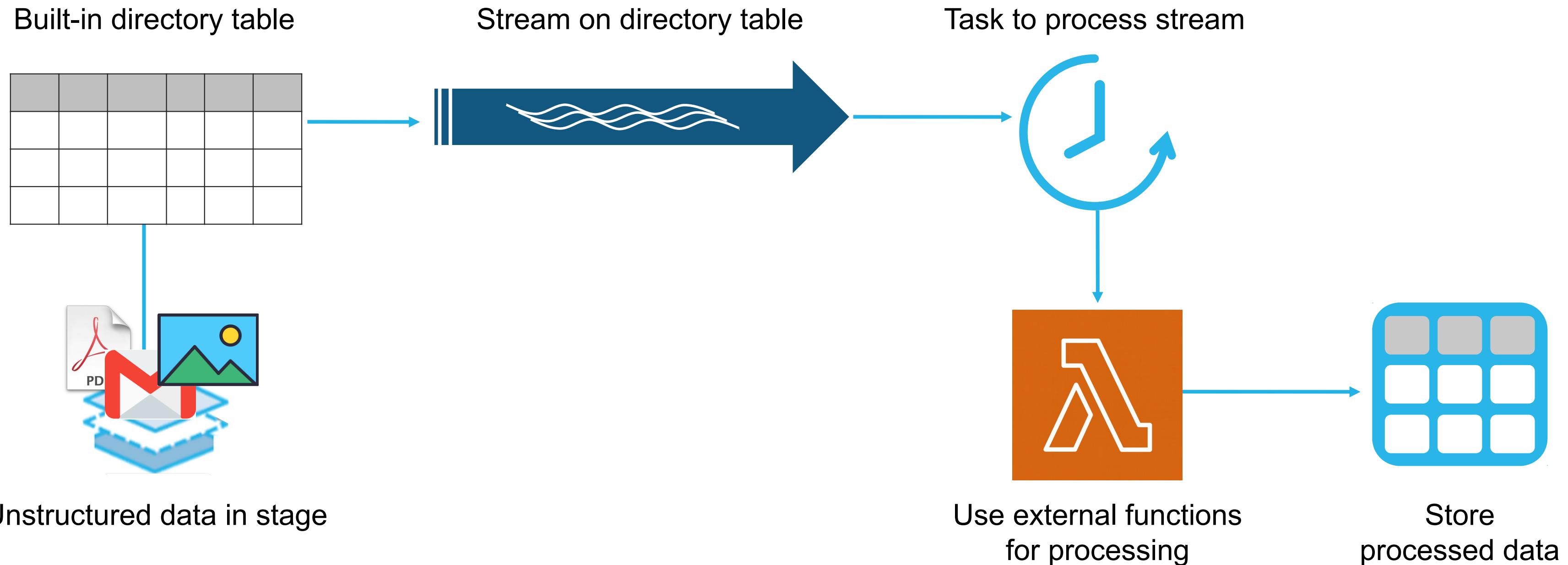
- Multiple streams on a single table
- Offset for each stream updates independently



# END-TO-END PROCESSING PIPELINE

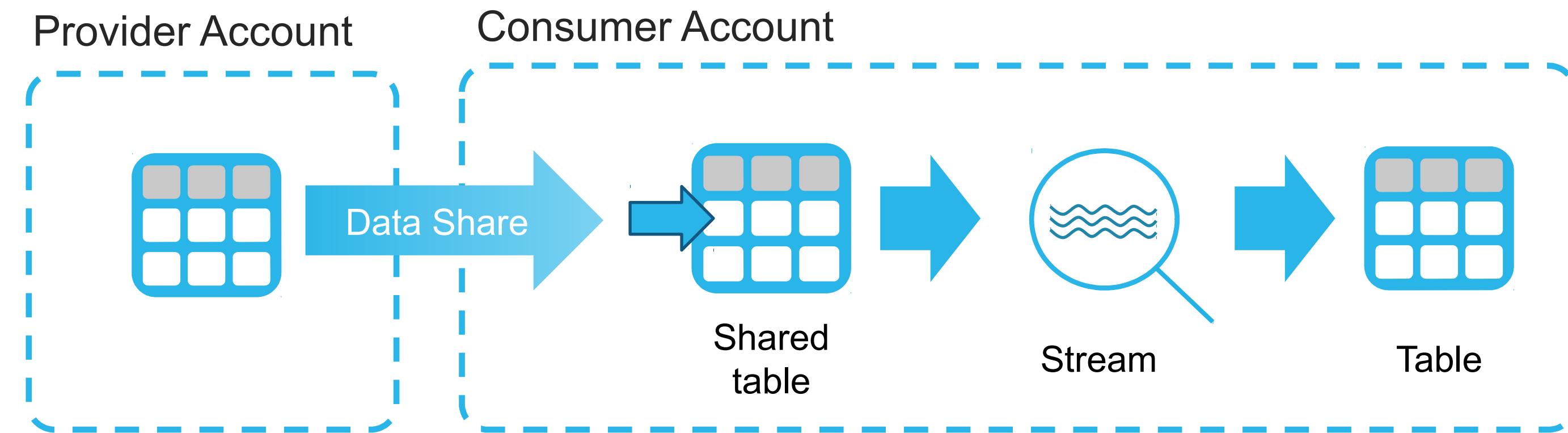


# END-TO-END PROCESSING PIPELINE



# SHARING STREAMS

- Provider creates stream on shared table, or enables change tracking
- Consumer can track changes on shared table



# LAB EXERCISE: 13

## Track Table Changes with Streams

25 minutes

- Creating Basic Table Streams
- Querying Table Streams
- Exploring Delta and Append Streams
- Pairing Streams and Tasks



# DATABASE REPLICATION



# MODULE AGENDA

- Overview
- Replication Workflow
- Failover Workflow

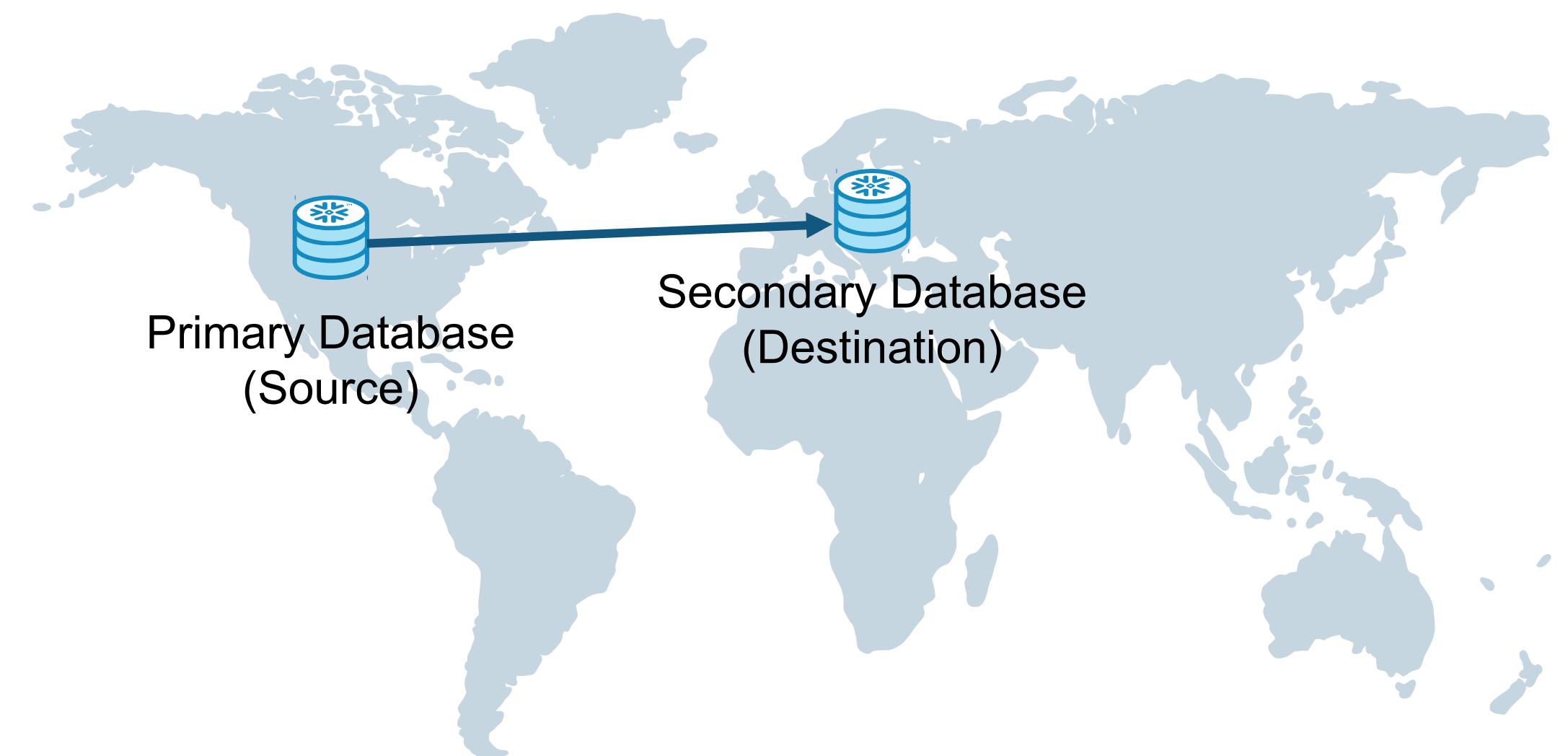


# OVERVIEW



# WHAT IS REPLICATION?

- Keeps databases synchronized between accounts (same organization)
- Replicates entire database
- Secondary - read-only

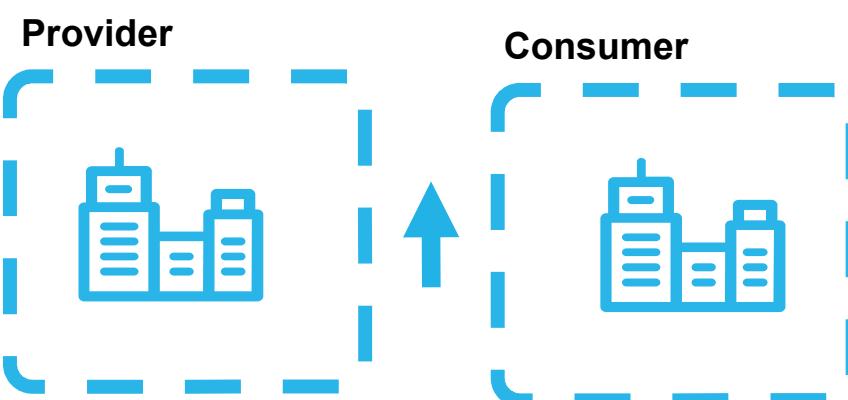


# USE CASES

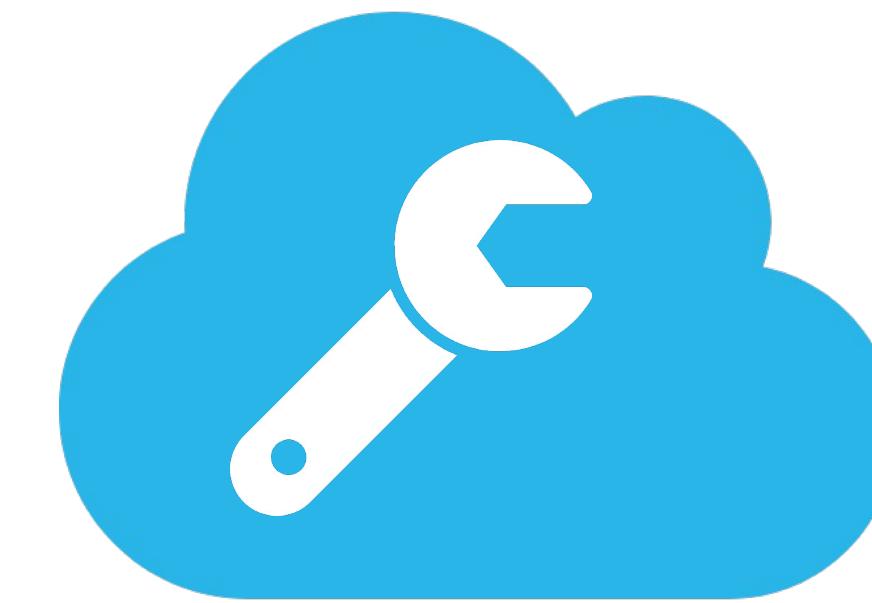
**Cross-Region/Cloud  
Disaster Recovery**



**Cross-Region/Cloud  
Data Sharing**

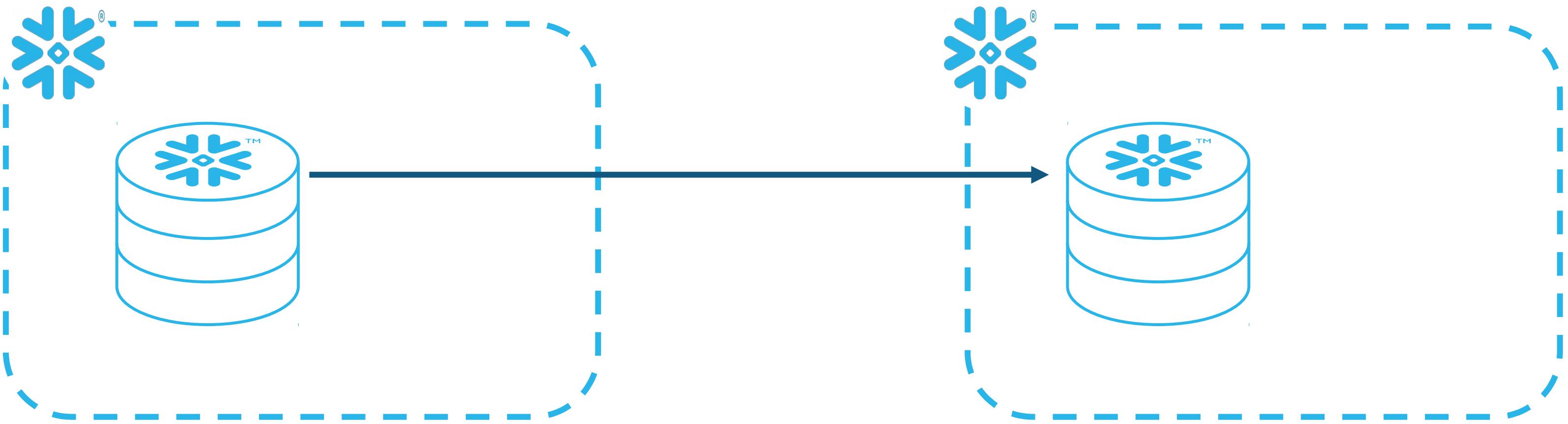


**Account  
Migration**



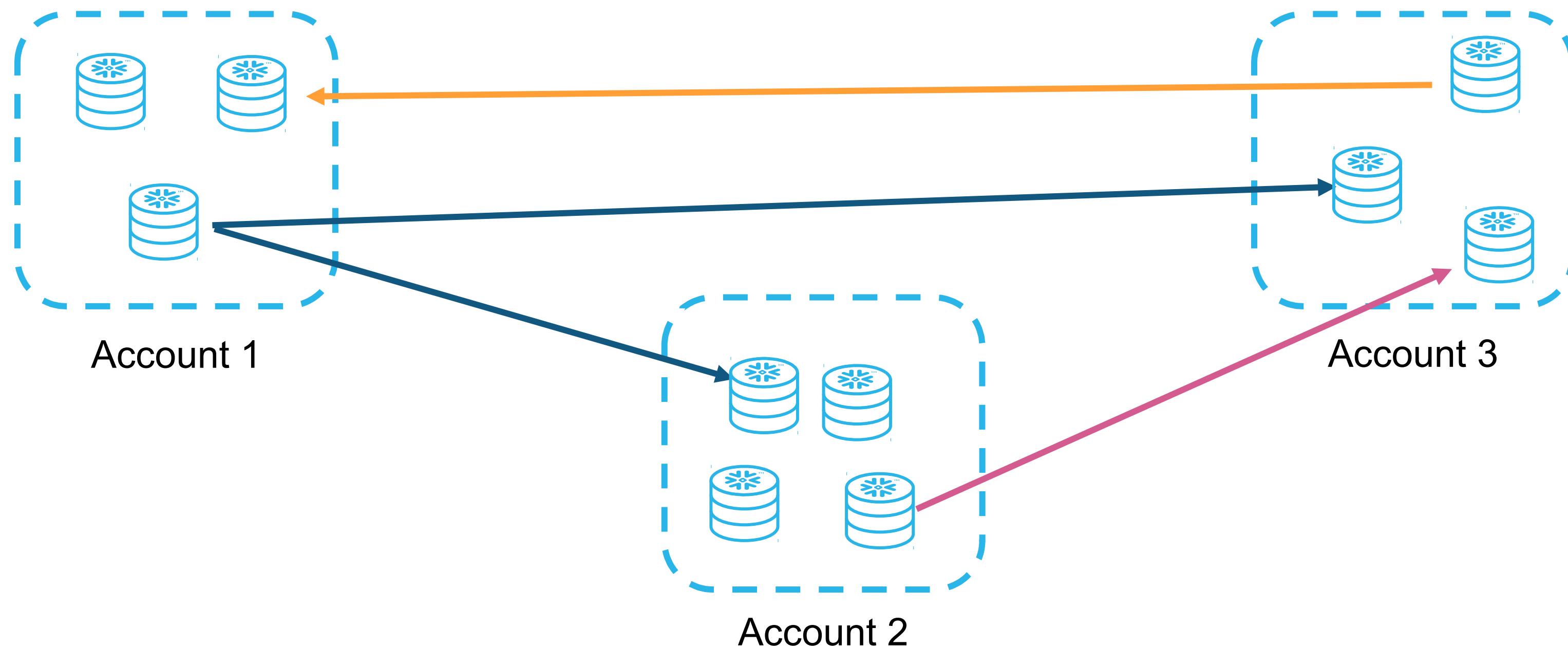
# DATABASE REPLICATION

- Replication can be as simple as one account to another



# DATABASE REPLICATION

- Replication can involve many accounts, replicating in multiple directions



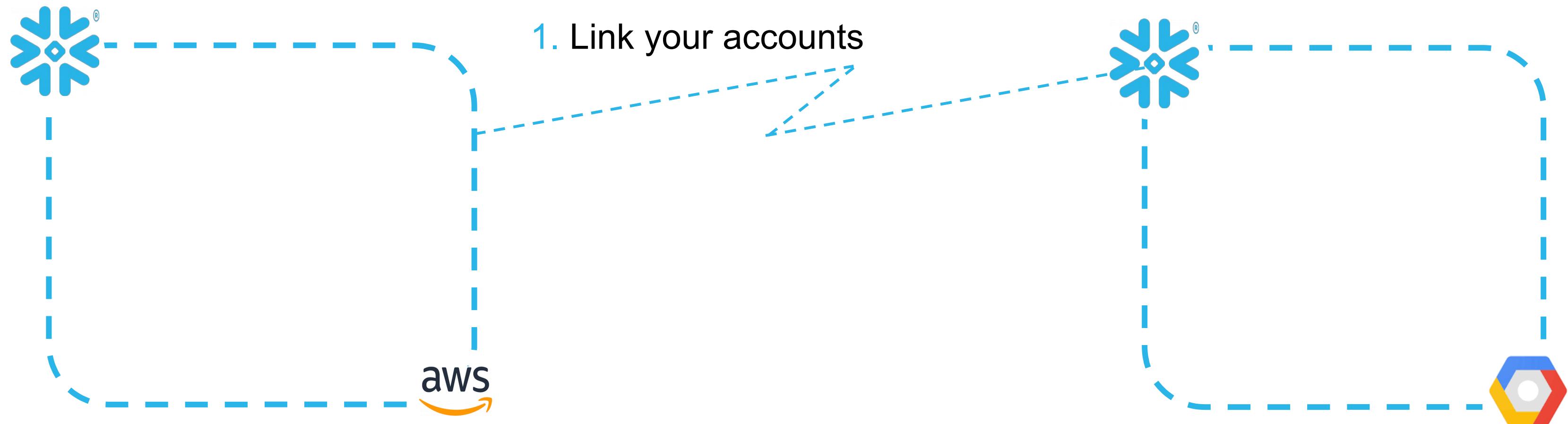
# REPLICATION WORKFLOW



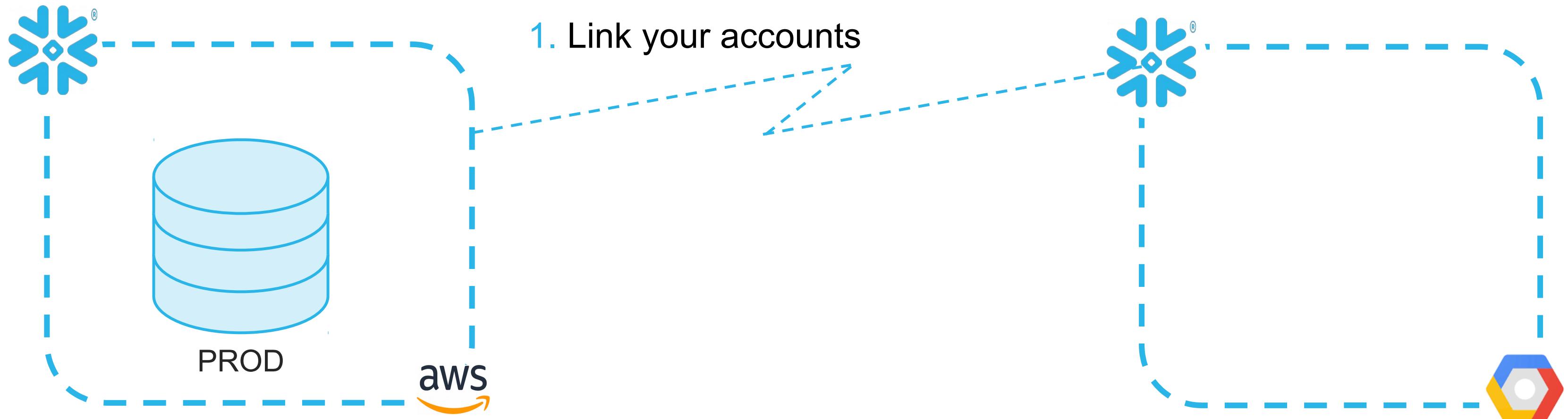
# SET UP DATABASE REPLICATION



# SET UP DATABASE REPLICATION



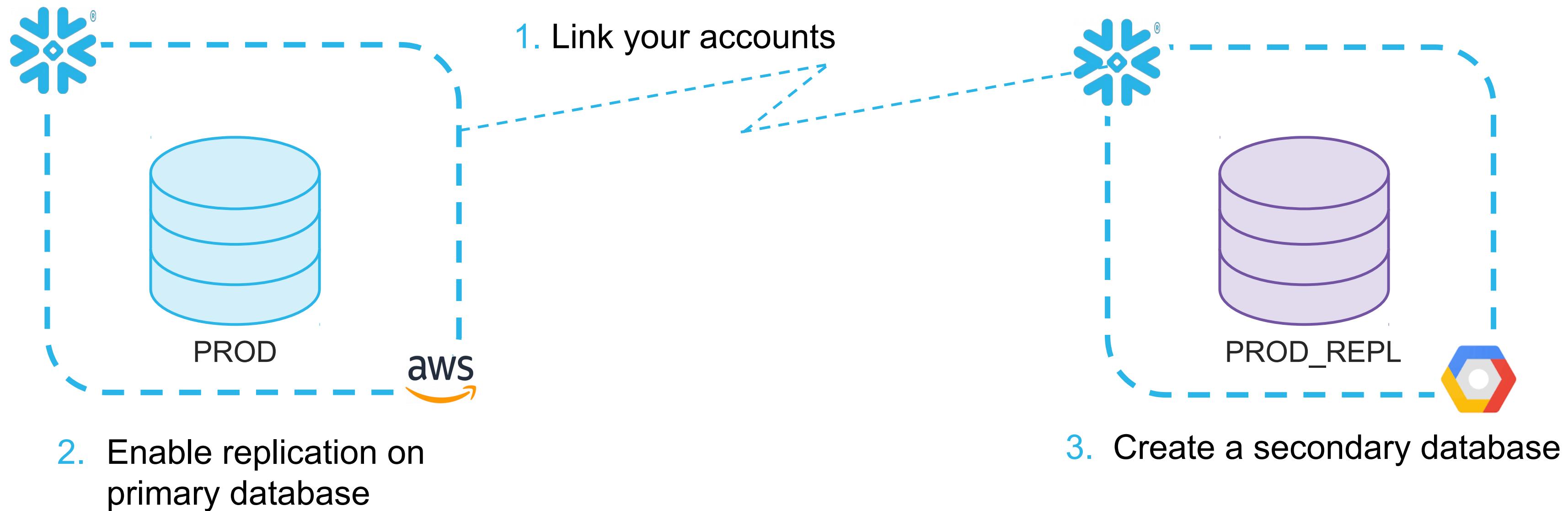
# SET UP DATABASE REPLICATION



1. Link your accounts
2. Enable replication on primary database

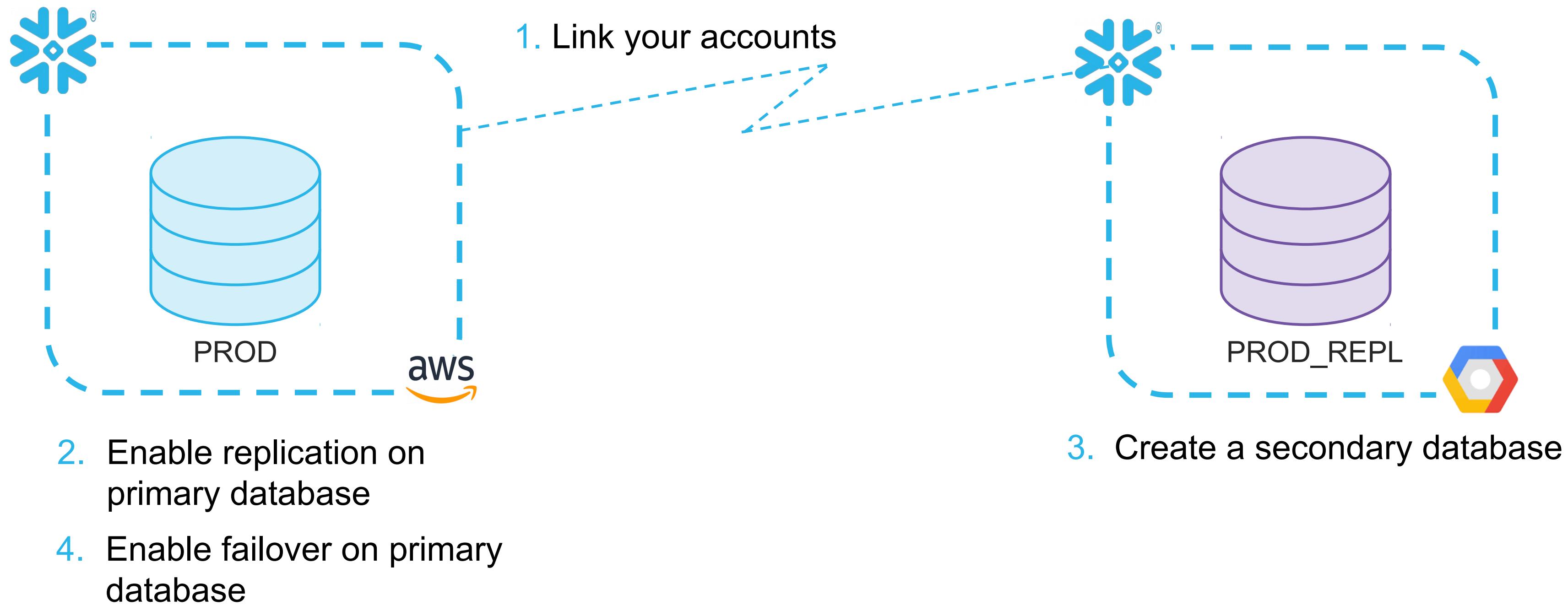
```
ALTER DATABASE prod ENABLE REPLICATION TO  
ACCOUNTS gcp_us_central1.companya2;
```

# SET UP DATABASE REPLICATION



```
CREATE DATABASE prod_repl AS REPLICA OF  
aws_us_west2.companya1.prod;
```

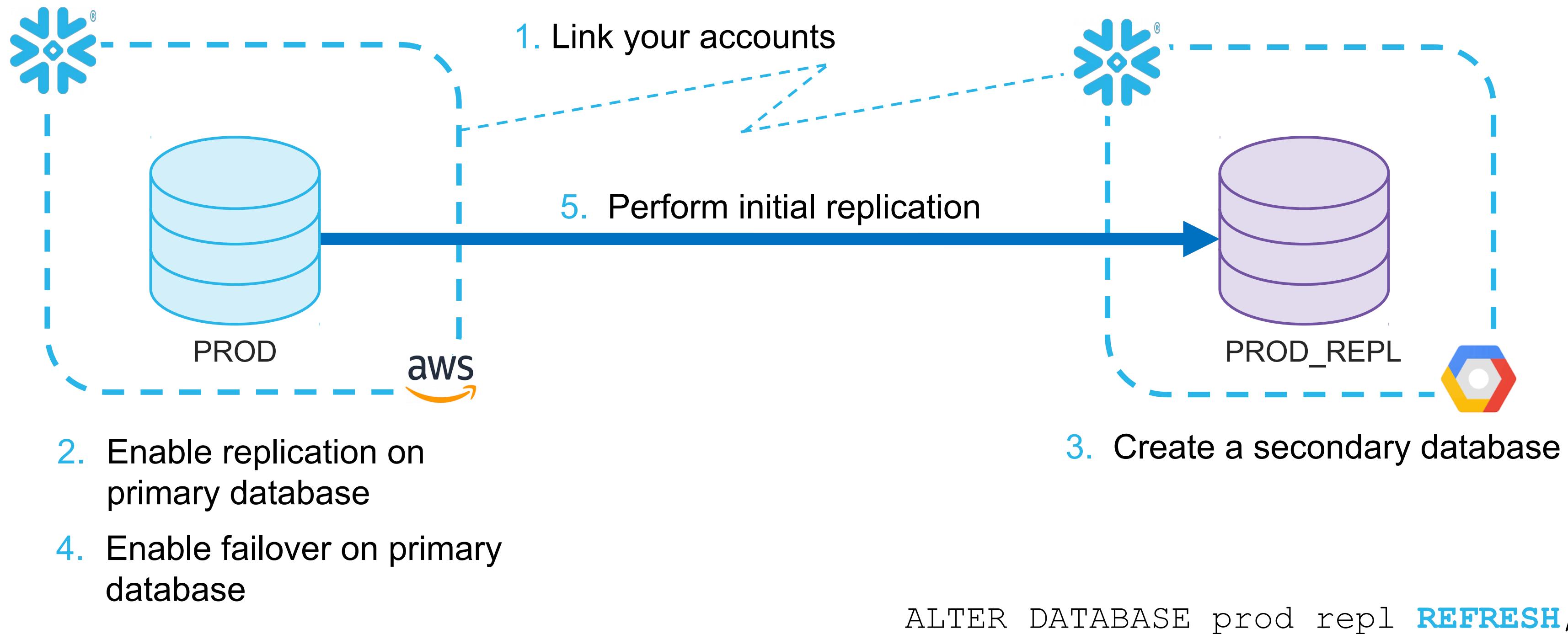
# SET UP DATABASE REPLICATION



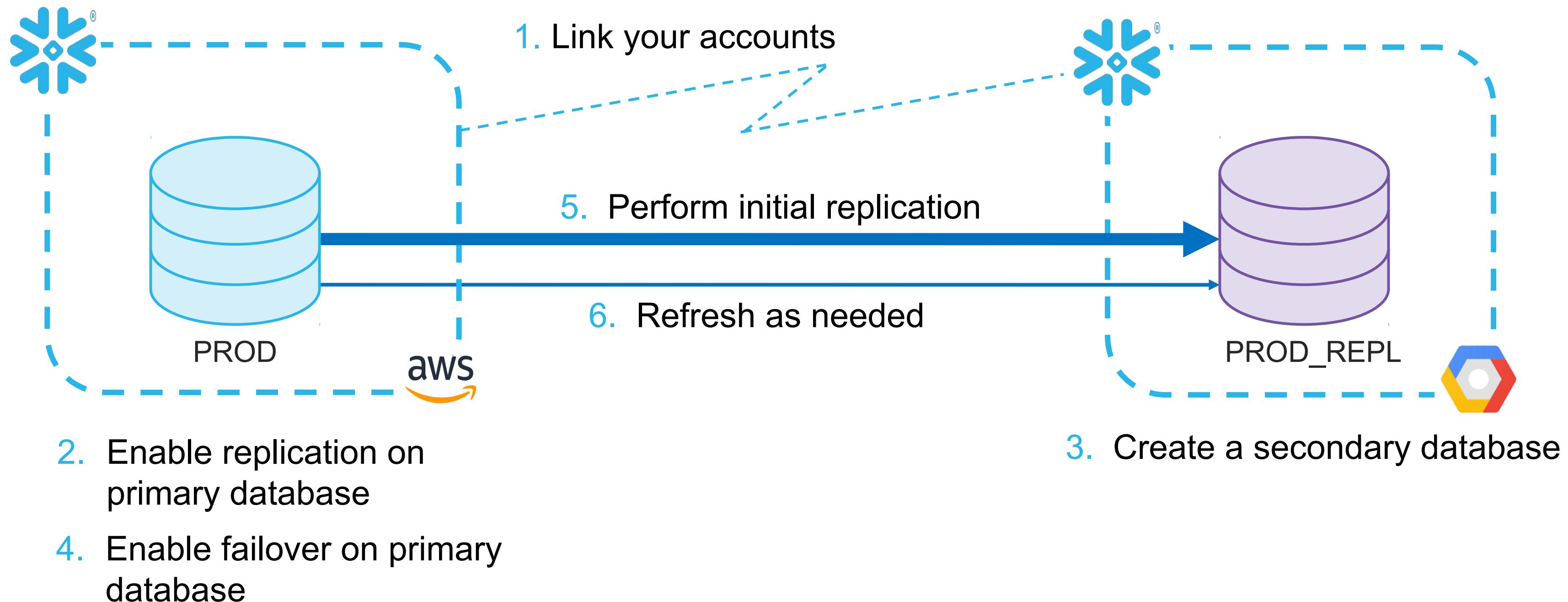
```
ALTER DATABASE prod
```

```
ENABLE FAILOVER TO ACCOUNT gcp_us_central1.companya2;
```

# SET UP DATABASE REPLICATION



# SET UP DATABASE REPLICATION



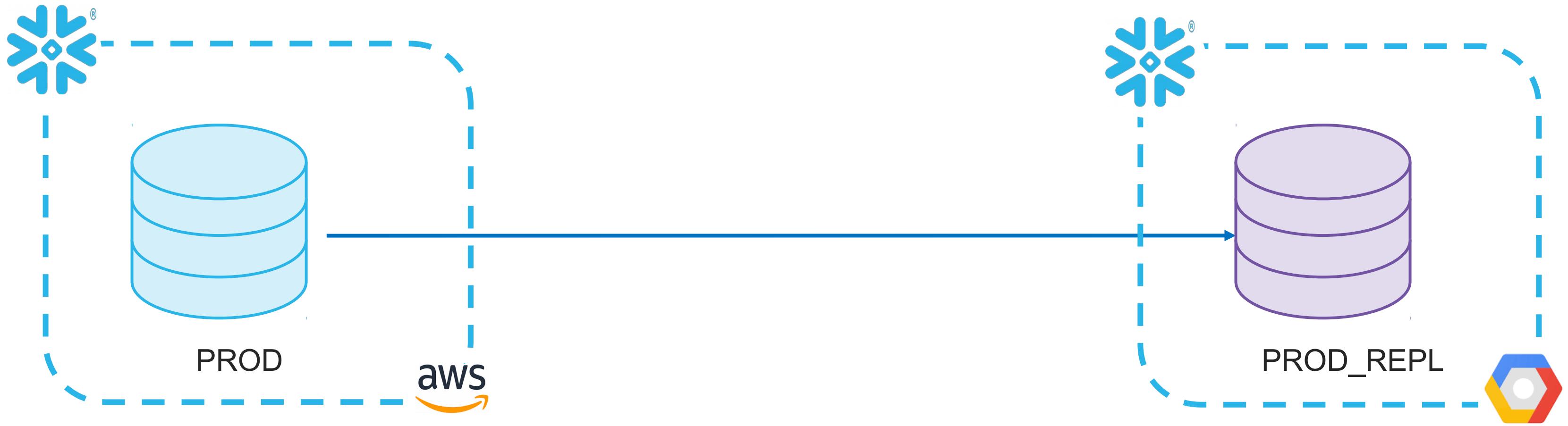
# REFRESH THE SECONDARY DATABASE

- Refresh manually, or on a schedule using tasks

```
CREATE OR REPLACE TASK refresh_db
SCHEDULE = '30 minute'
AS
ALTER DATABASE prod_repl REFRESH;
```



# VERIFY REPLICATION RELATIONSHIP



**SHOW REPLICATION DATABASES;**

SNOWFLAKE_REGION	ACCOUNT_NAME	NAME	IS_PRIMARY	PRIMARY
AWS_US_WEST_2	COMPANYA1	PROD	Y	ASW_US_WEST_2.COMPANYA1.PROD
GCP_US_CENTRAL1	COMPANYA2	PROD REPL	N	ASW_US_WEST_2.COMPANYA1.PROD

# HOW OFTEN TO REFRESH?

- Refresh rate depends on your RPO (Recovery Point Objective)

T<sub>1</sub> Secondary database is refreshed

T<sub>2</sub> Changes are happening on the primary database

T<sub>3</sub> Outage on primary database



Any changes made to the primary database during this time will NOT be reflected in the secondary database

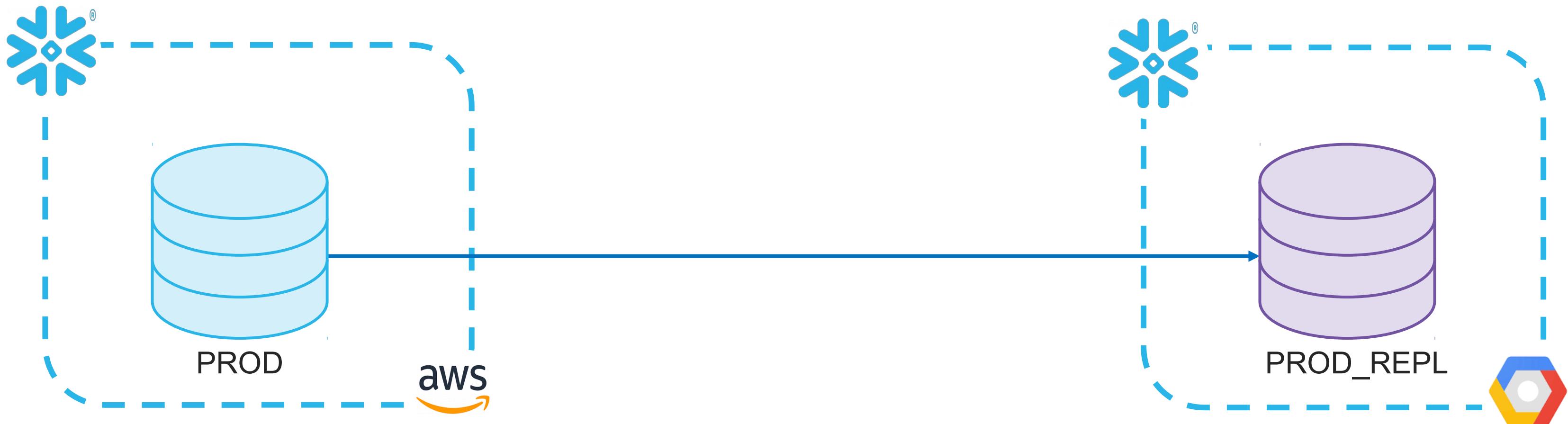


# FAILOVER WORKFLOW



# FAILOVER WORKFLOW

- Task is applying incremental changes on a regular basis



# FAILOVER WORKFLOW

- Outage on the primary database or account



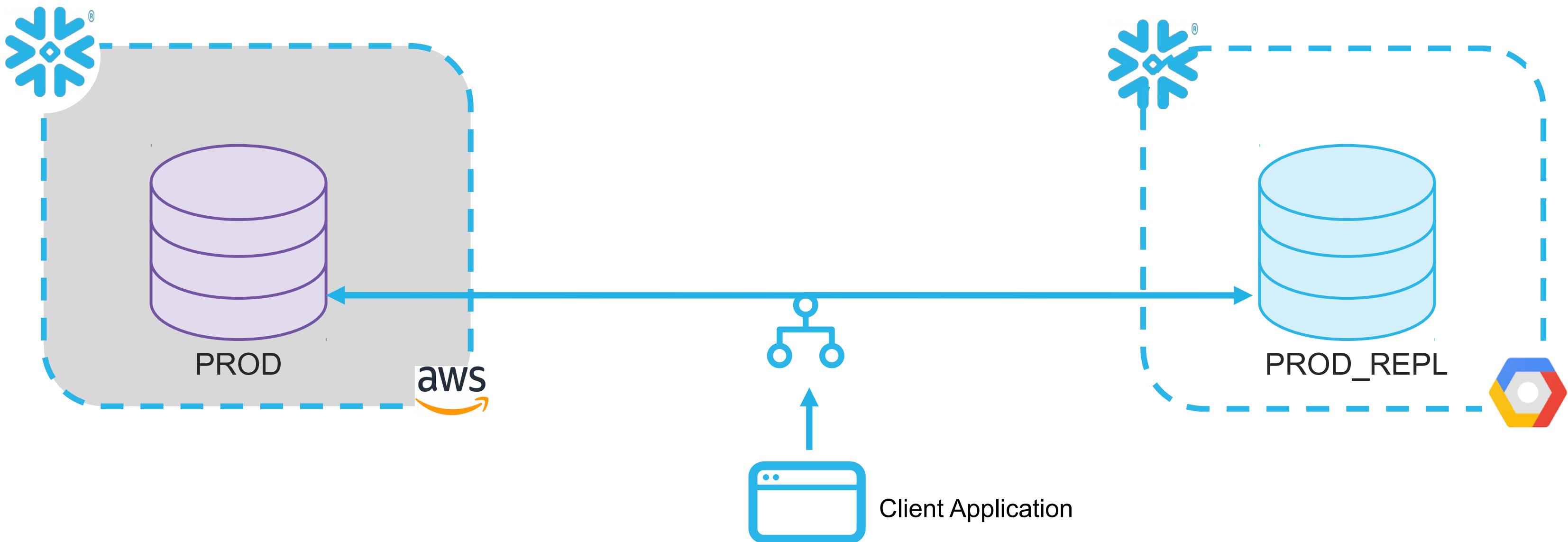
# FAILOVER WORKFLOW

- Clients connected directly to the PROD database will lose connectivity



# OPTIONAL: SET UP CLIENT REDIRECT

- Use client redirect, to automatically direct the connection to whichever database is primary



# REPLICATION CONSIDERATIONS

- **Database objects only** – users, roles, and warehouses not currently replicated
- **Clones physically copied** – Increased storage on secondary
- **Time Travel** – Potentially different results on primary and secondary
- **Egress Charges** – For cross-region or cross-cloud transfers
- **External tables not supported** – Replication will fail if they exist on primary
- **Failover / Fallback** – Risk of missing transactions during fail-back



# LAB EXERCISE: 14

## Database Replication and Disaster Recovery (OPTIONAL)

40 minutes

- Setup Primary a Database to Another Account
- Enable Replication and Failover of Primary Database
- Create Replica of Primary Database
- Monitor Replication Manually
- Schedule Replication Automatically (OPTIONAL)
- Changing Replication Direction

**Note:** Instructor will grant students ACCOUNTADMIN role for this lab.



# DATA SHARING



# MODULE AGENDA

- Data Access Options
- Direct Data Sharing
- Direct Data Sharing Workflow



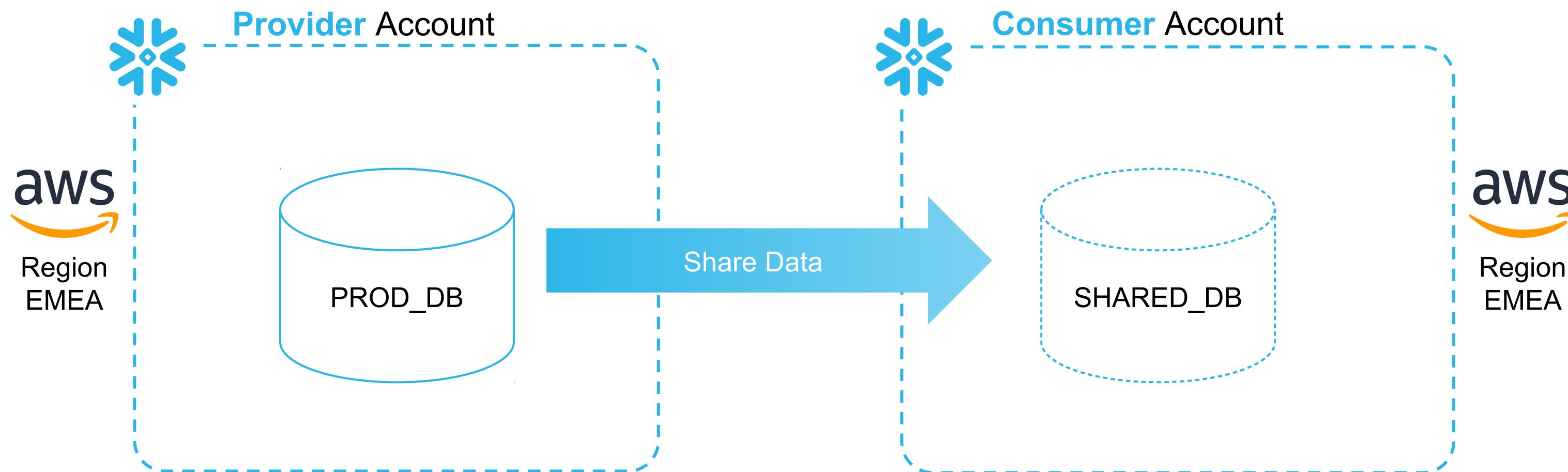
# DATA ACCESS OPTIONS



# DATA ACCESS OPTIONS

## DIRECT DATA SHARING

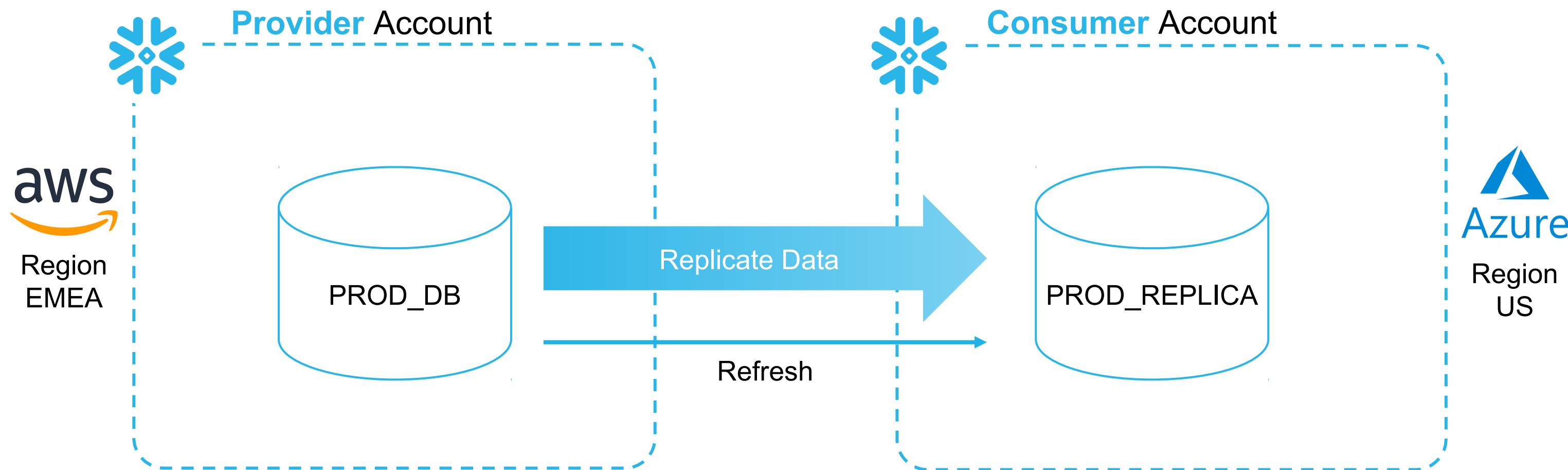
- Securely and directly share data
- Tables (including external tables), secure views, and secure UDFs



# DATA ACCESS OPTIONS

## REPLICATION

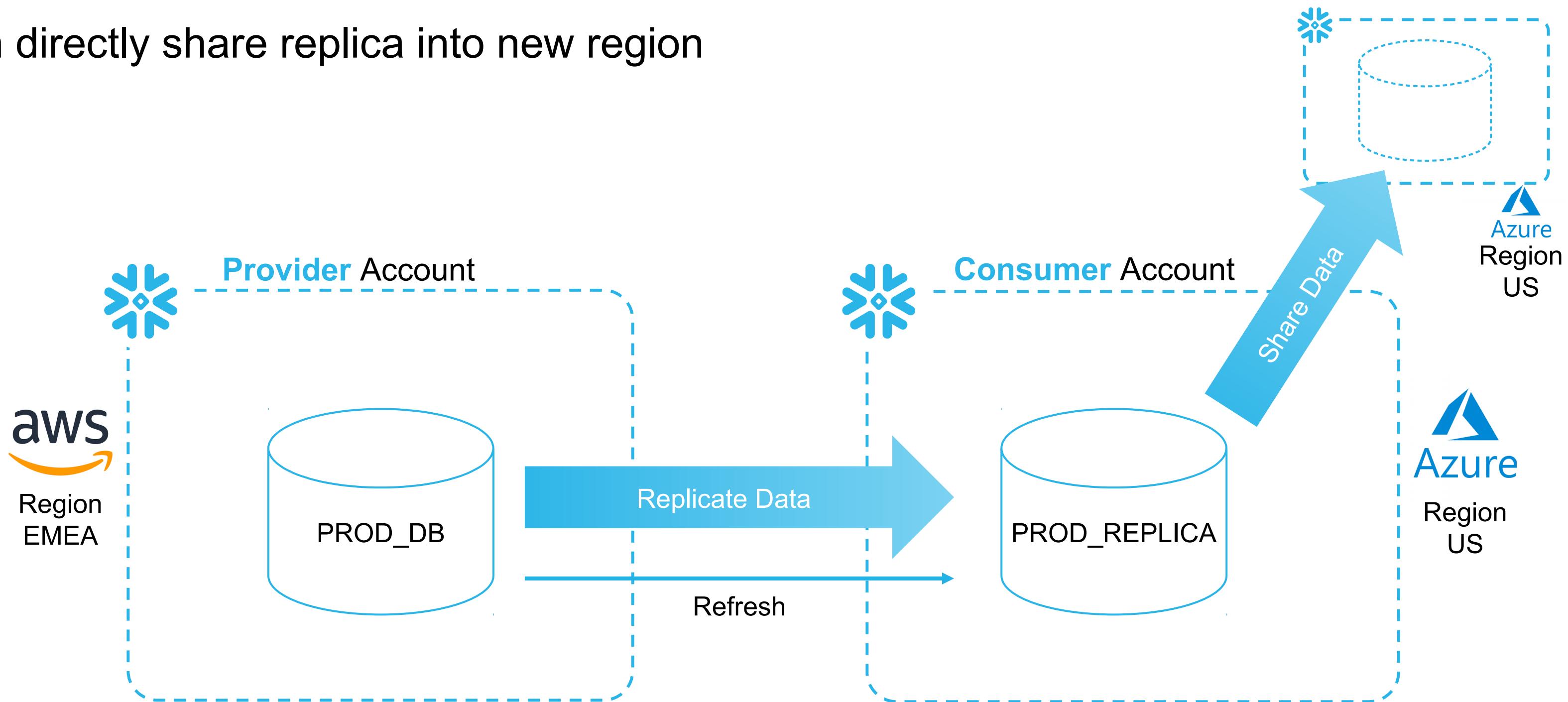
- Replicate data from one cloud/region to another
- Read-only access to replica



# DATA ACCESS OPTIONS

## CROSS-REGION OR CLOUD SHARING

- Can directly share replica into new region



# DATA ACCESS OPTIONS

Option	Boundary	Access	Clone or Share?	Data Transfer	Use Cases
<b>Zero Copy Clone</b>	Within a single account	READ WRITE ALTER	YES	No data physically copied	Spin up dev or test environment Backup or archival
<b>Direct Data Share</b>	Across accounts in same region/cloud	READ	NO	No data physically copied	Sharing data with partners Sharing with other accounts in your organization
<b>Database Replication</b>	Across account in different region/cloud	READ	YES	Data physically replicated (incrementally)	Disaster recovery Migration across clouds Sharing to another cloud/region
<b>Cross-region or cloud sharing</b>	Replicate across accounts in different regions; share across accounts in same region	READ	NO	No data physically copied from replica to consumer	Sharing to another cloud/region



# DIRECT DATA SHARING



# TRADITIONAL WAYS OF SHARING DATA

## Traditional Methods



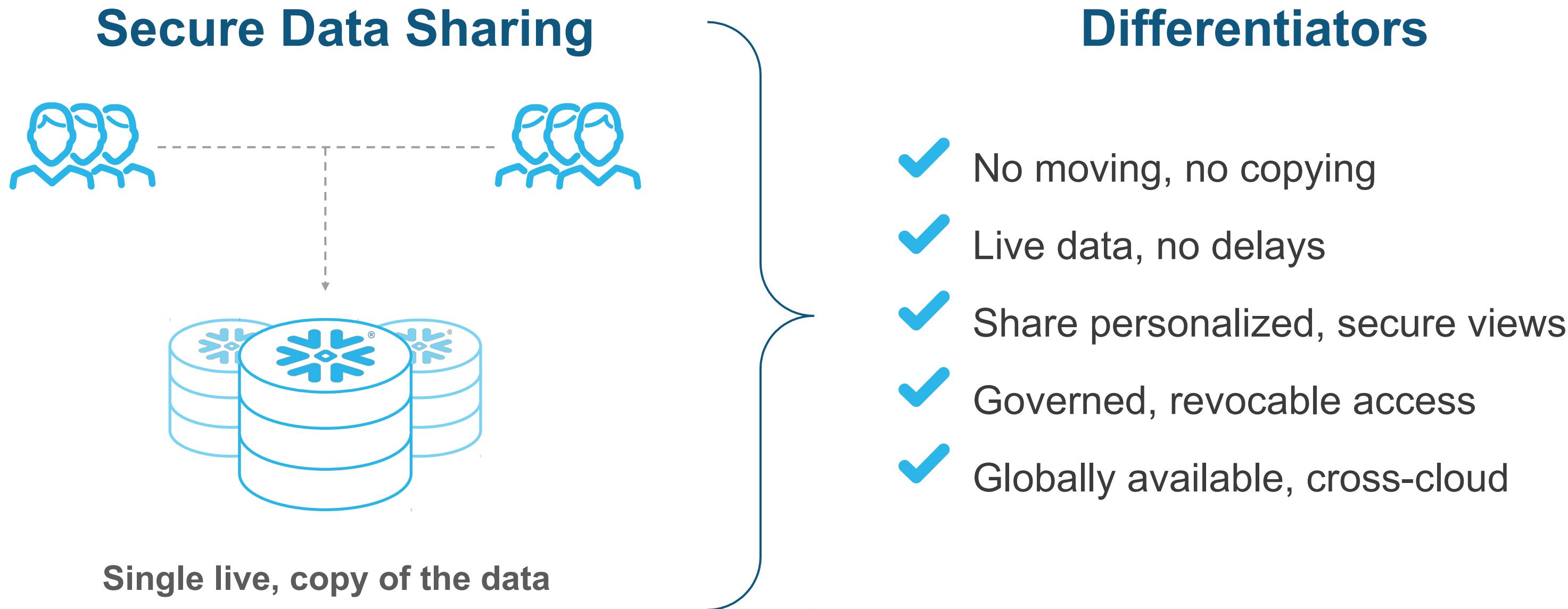
- Copying files in FTP/ cloud buckets
- Building, Maintaining & Calling APIs
- ETL pipelines
- Data marts

## Gaps

- ✖ Copy and move data
- ✖ Costly to maintain
- ✖ Data is delayed
- ✖ Error-prone
- ✖ Unsecure, once data is moved

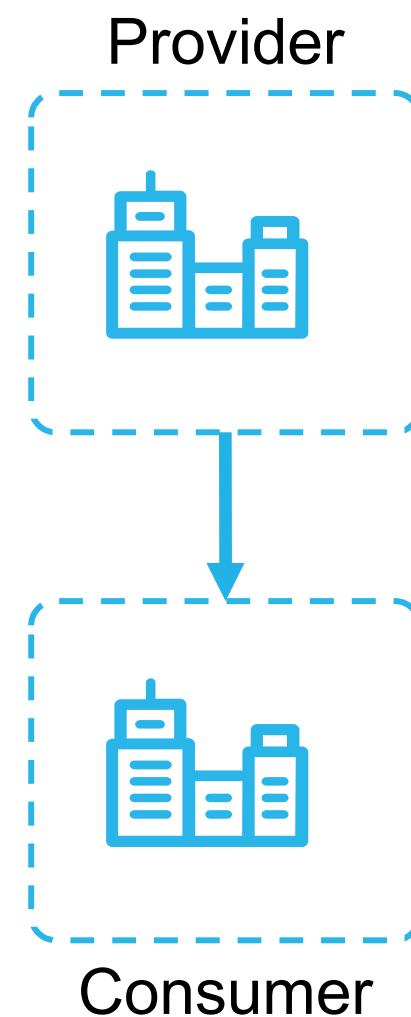


# SECURE DIRECT DATA SHARING

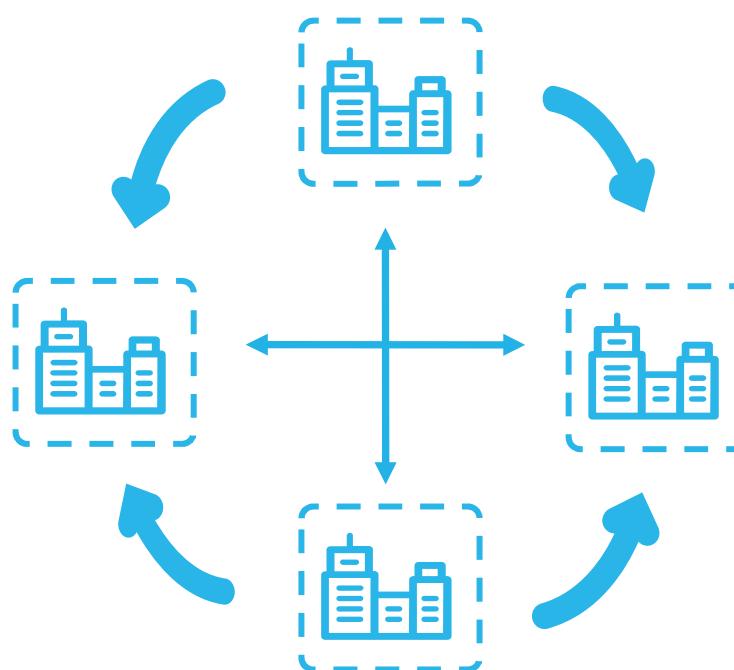


# DIRECT SHARING OPTIONS

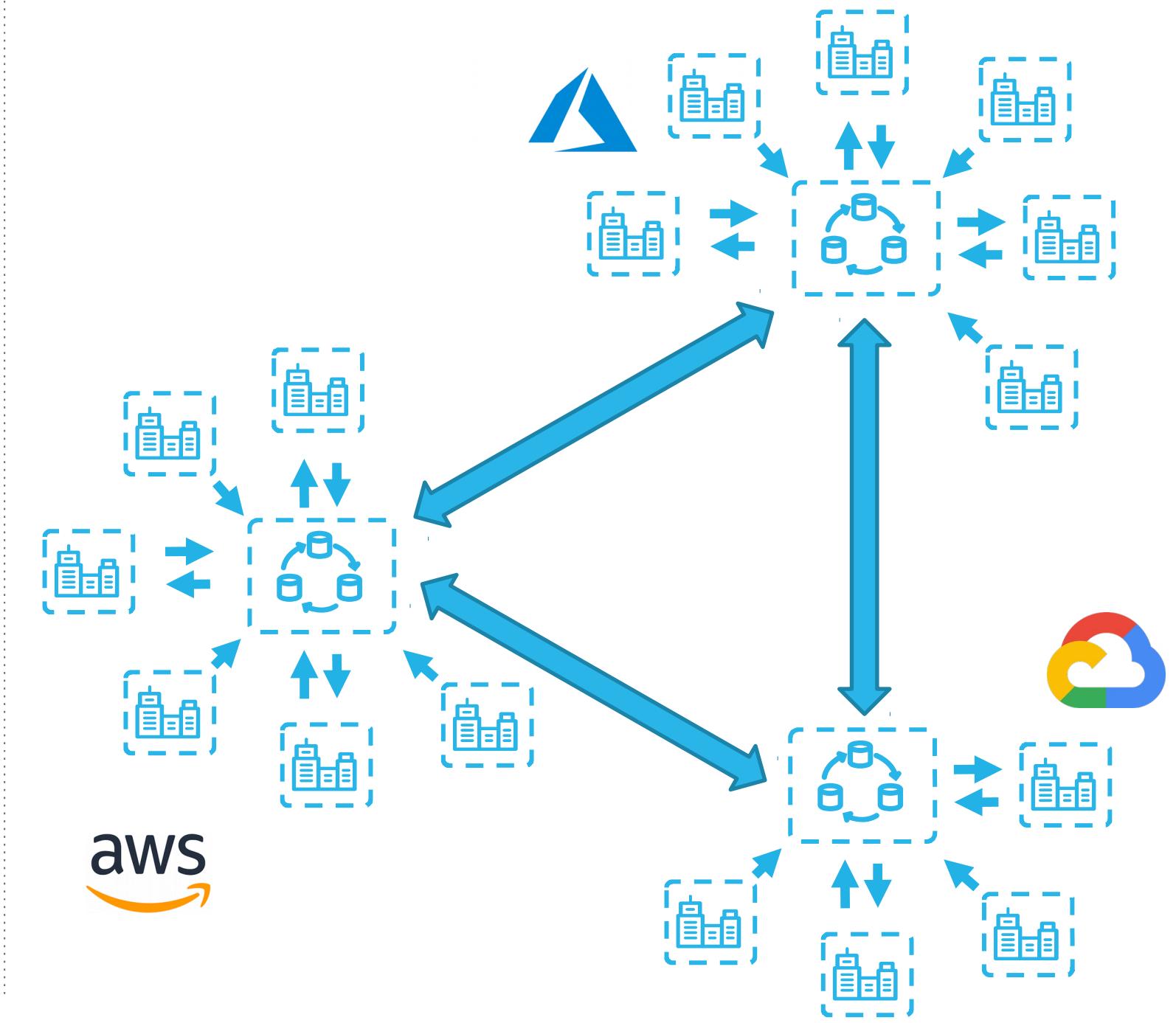
## Direct Data Share



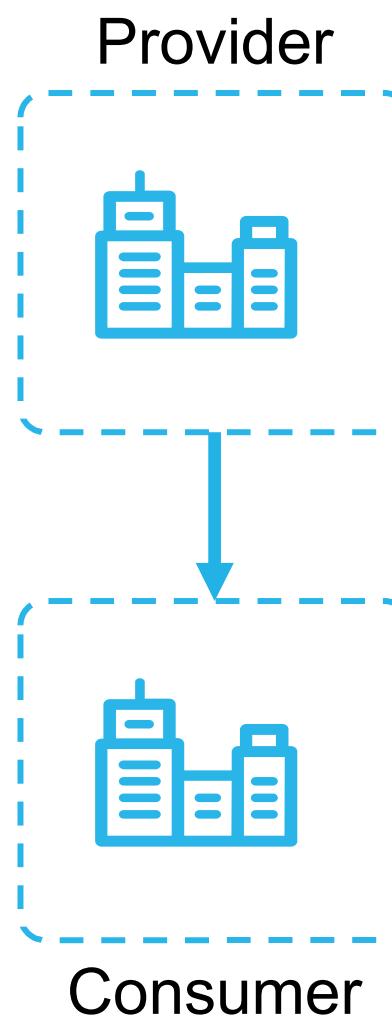
## Snowflake Data Exchange



## Snowflake Data Marketplace



# WHAT IS A DATA SHARE?



- Named Snowflake object that contains no data
- Contains sharing information
  - Privileges granting access to objects
  - Full Snowflake data governance control
  - Names of account to share with
- Fully revokable by provider at any time

# DIRECT DATA SHARING WORKFLOW

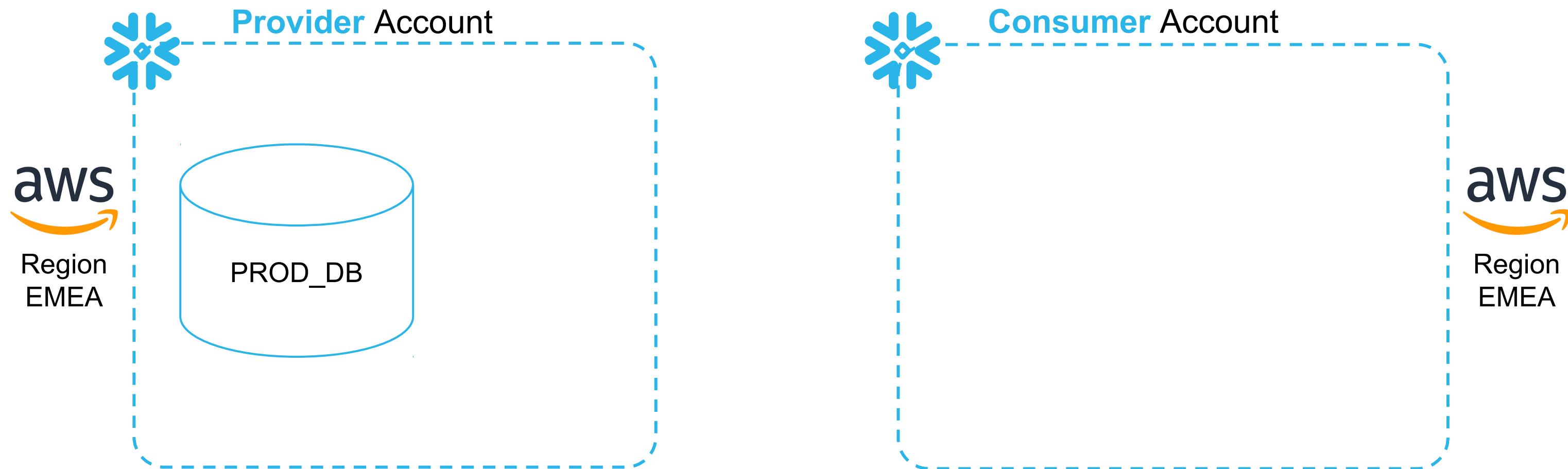
\$\$\$



© 2022 Snowflake Computing Inc. All Rights Reserved

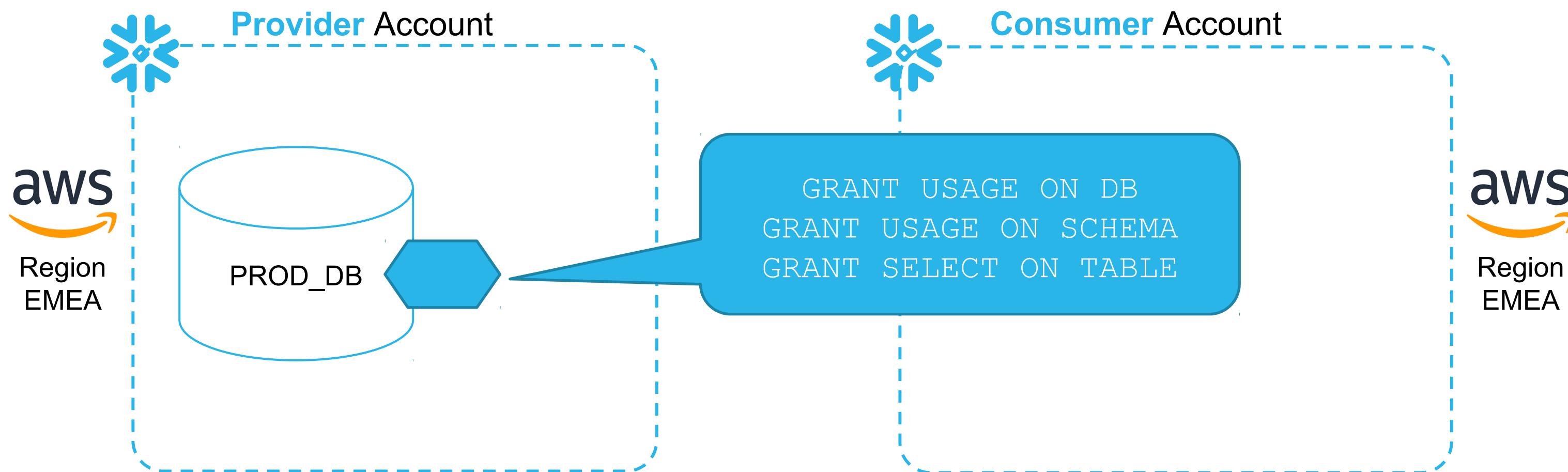
# DIRECT DATA SHARING

1. Provider and consumer in same region; consumer needs access



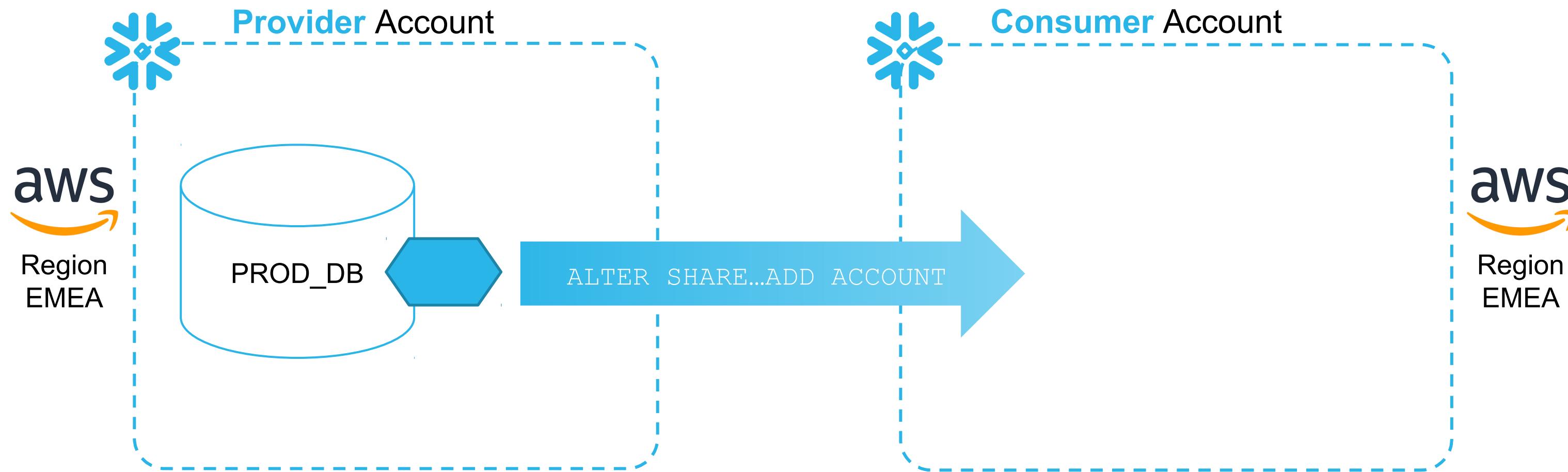
# DIRECT DATA SHARING

2. Provider creates data share object and grants privileges to it



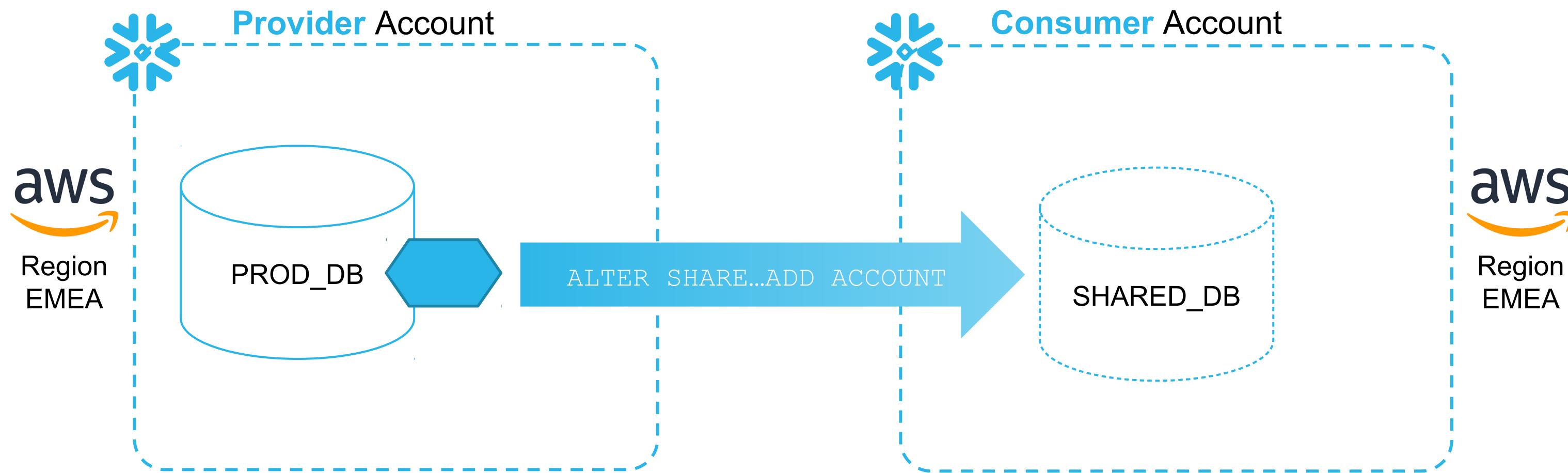
# DIRECT DATA SHARING

- Provider grants access to consumer; share is instantly available to the consumer



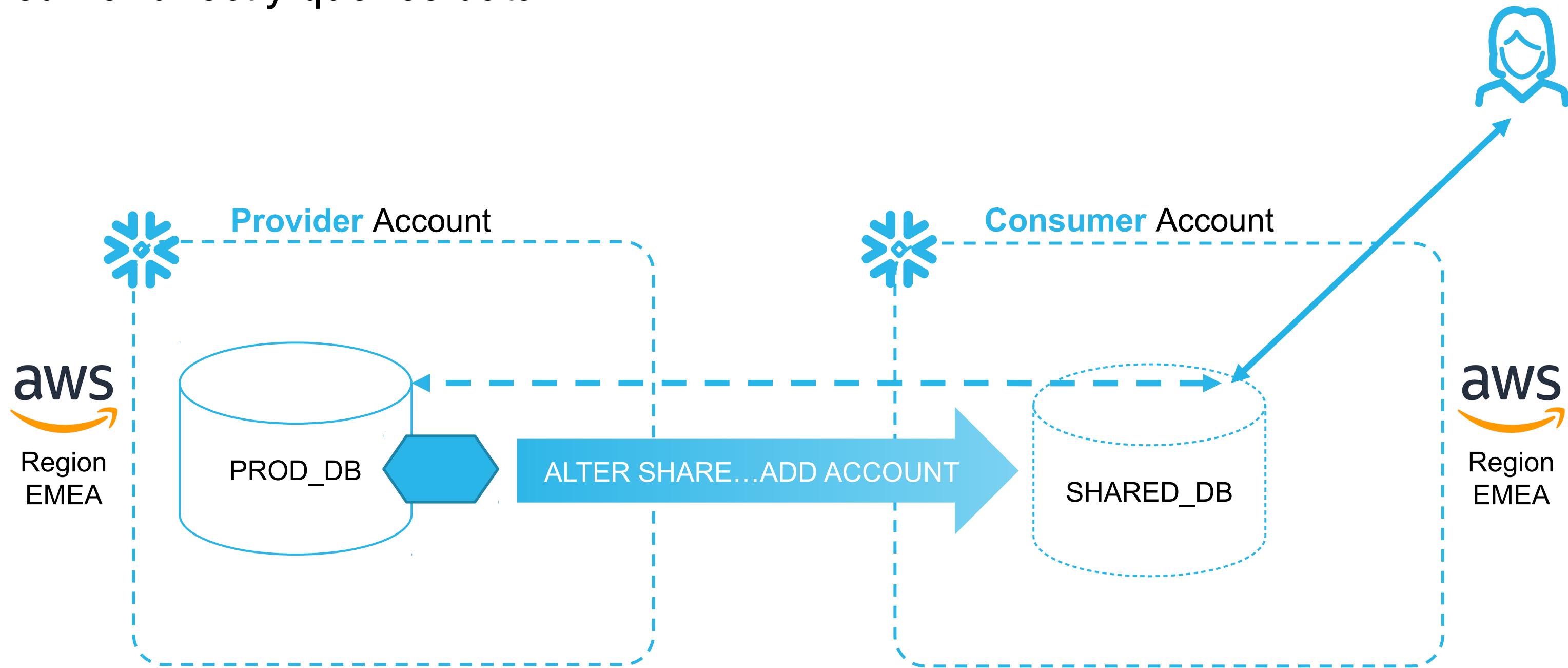
# DIRECT DATA SHARING

4. Consumer creates a dummy database to "hold" the share and grants imported privileges



# DIRECT DATA SHARING

## 5. Consumer directly queries data



# READER ACCOUNTS



Data  
Consumers



Reader  
Accounts

- Create a Snowflake account for a partner you want to share data with, who does not already have an account of their own
- Provisioned in minutes
- Provider is responsible for compute and storage costs incurred by the reader account

# LAB EXERCISE: 15

## Data Sharing

30 minutes

- Set up
- Basic Data Sharing
- Create Database on Consumer Account from Data Provider
- Use a Share as a Data Consumer
- Remove Objects
- Explore Secure User Defined Functions for Protecting Shared Data



# DATA CLUSTERING



# MODULE AGENDA

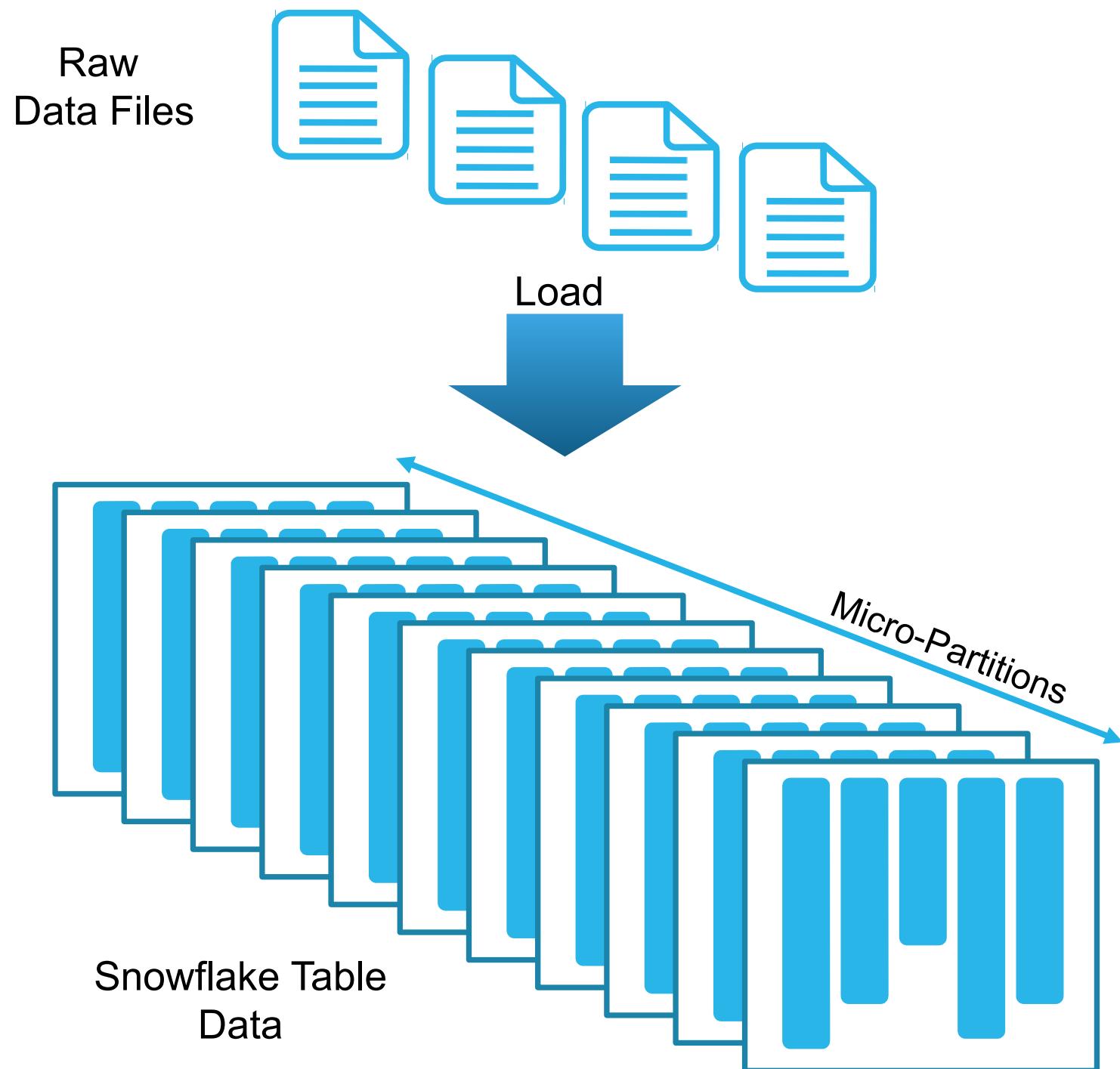
- What is Data Clustering?
- Micro-Partition Pruning (Elimination)
- Evaluating Clustering
- Implement and Test Cluster Keys



# WHAT IS DATA CLUSTERING?



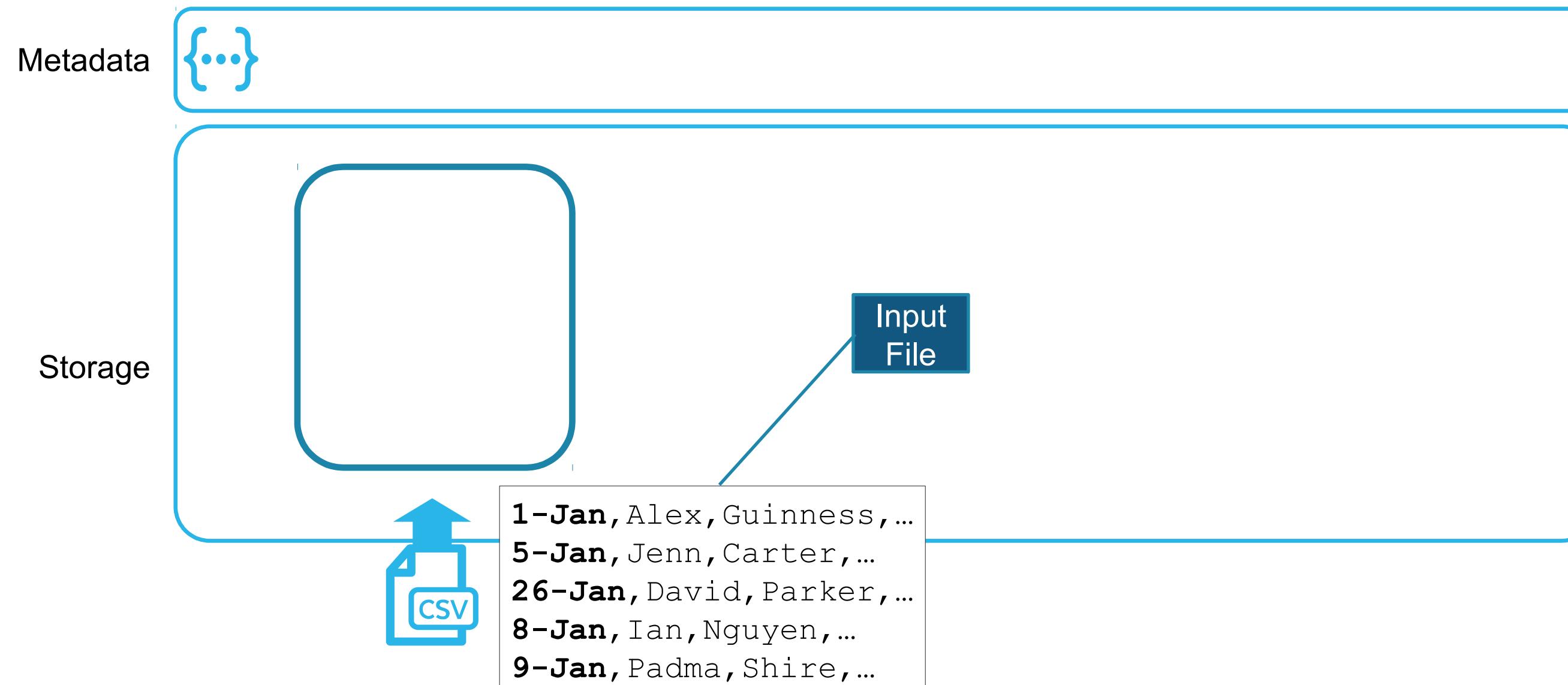
# MICRO-PARTITIONS



- Contiguous units of Snowflake storage:
  - 50 - 500 MB compressed to ~16MB
- Data stored in arrival sequence
- Immutable – never updated
- Statistics automatically captured:
  - Count of rows, distinct values, nulls
  - Minimum/maximum of every column

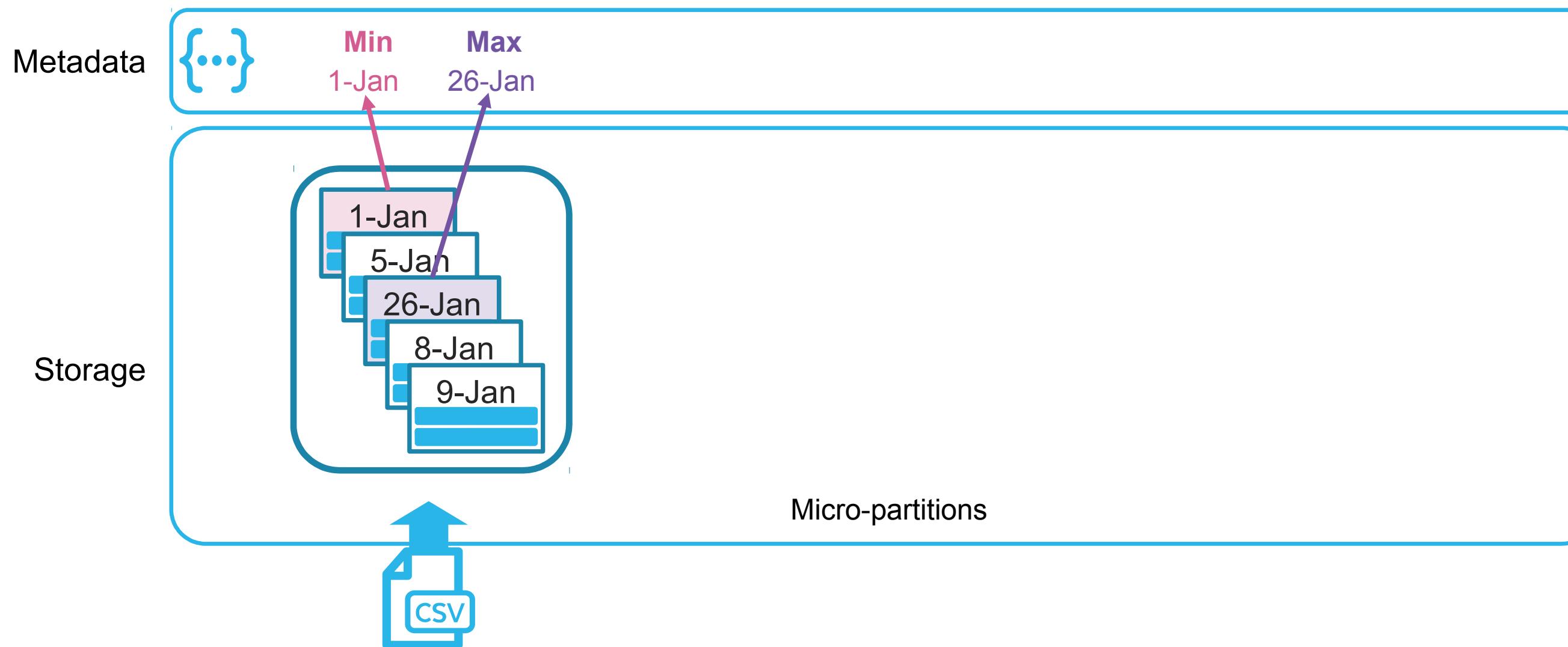
# NATURAL DATA ORDERING ("CLUSTERING")

- Data loaded each month – stored together by default



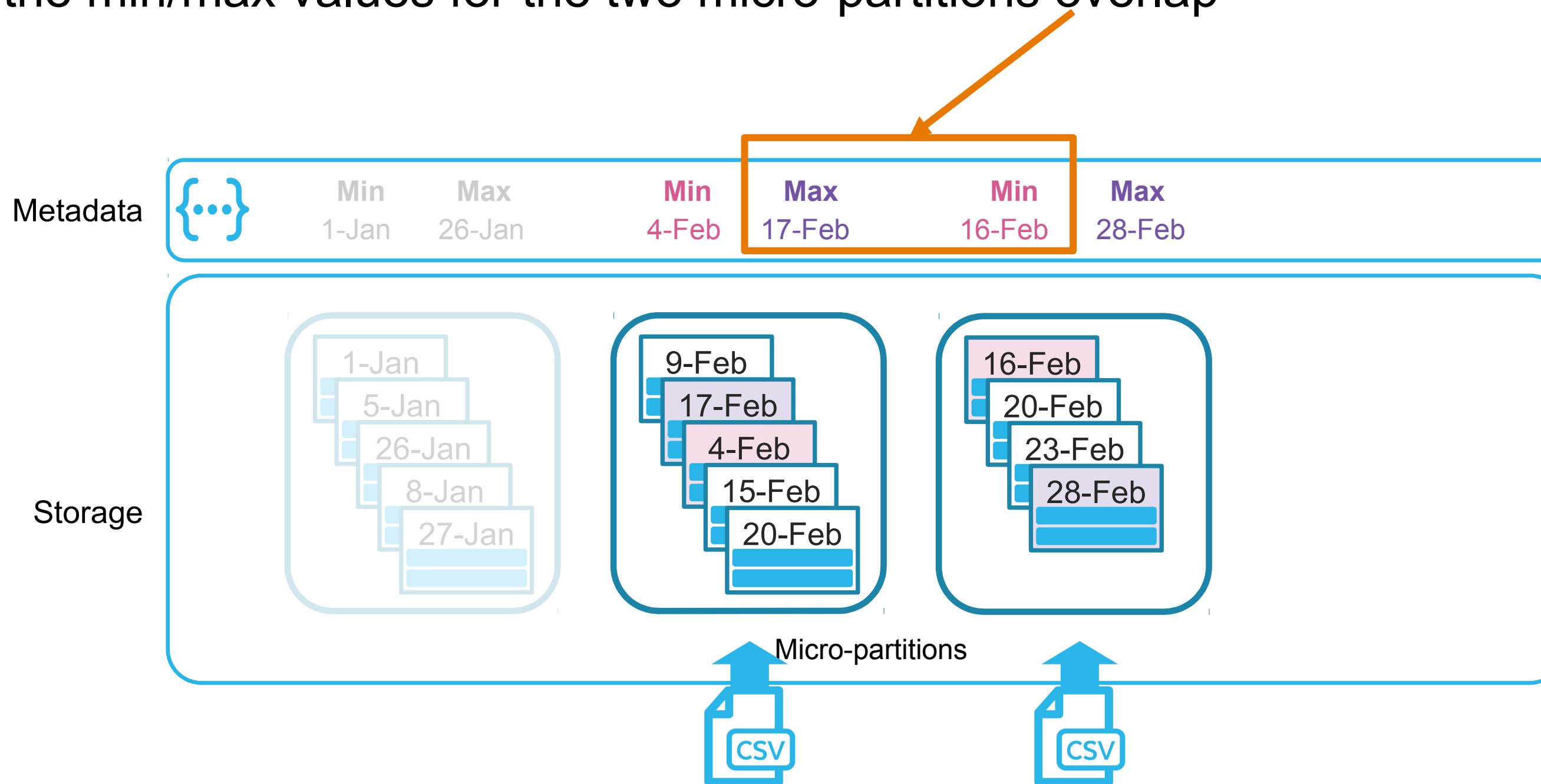
# NATURAL DATA ORDERING ("CLUSTERING")

- Data loaded each month – stored together by default
- Min and max of every column stored in metadata



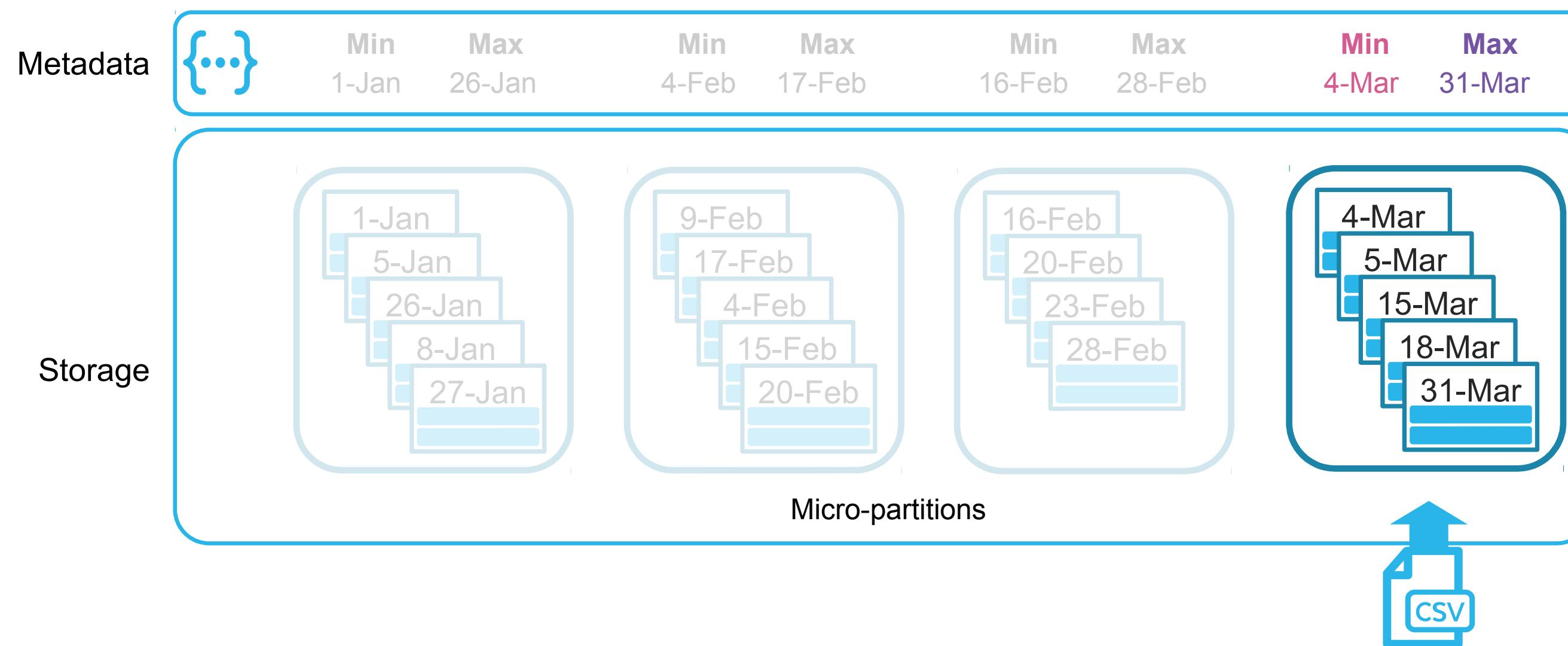
# FEBRUARY DATA LOADED

- February data appended, fills two micro-partitions
- Notice the min/max values for the two micro-partitions overlap



# MARCH DATA LOADED

- Finally, March data is appended
- Again, by default – stored together



# **MICRO-PARTITION PRUNING (ELIMINATION)**



# MICRO-PARTITION PRUNING

## Statistics

Scan progress	100.00%
Bytes scanned	435.95MB
Percentage scanned from cache	0.00%
Bytes written to result	510.12MB
Bytes sent over the network	0.02MB
Partitions scanned	28
Partitions total	86546

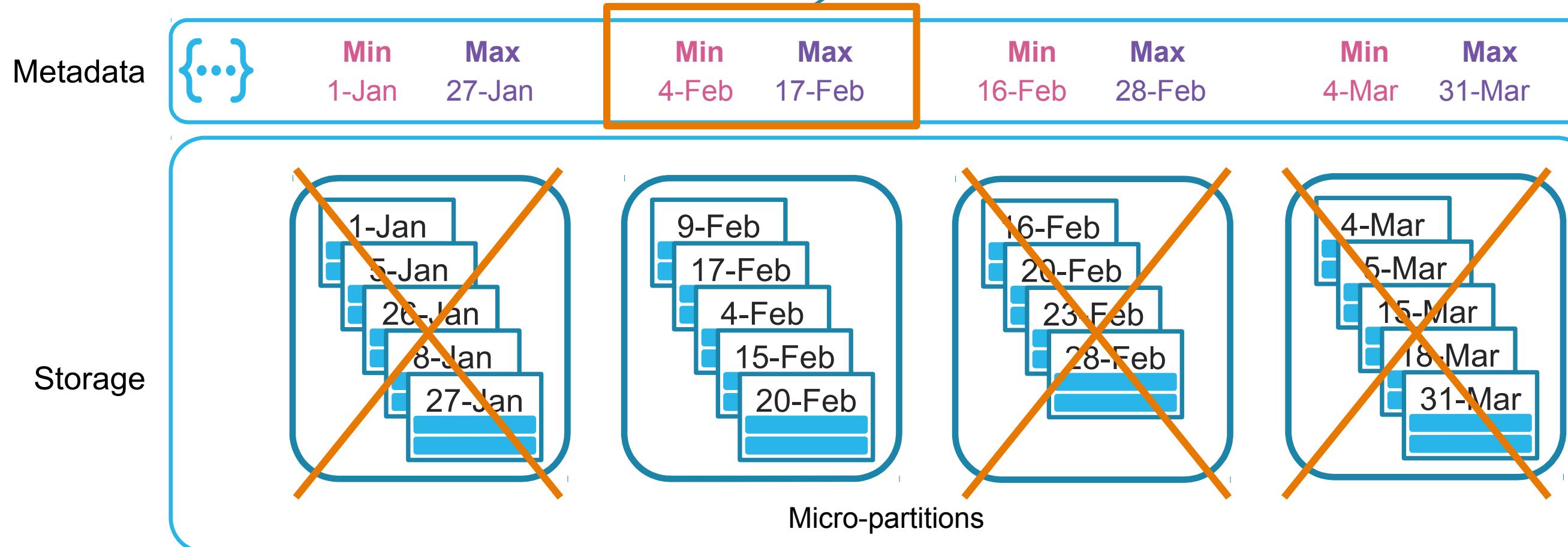
- Queries are pruned based on the WHERE clause
  - Uses minimum/maximum metadata to eliminate unnecessary micro-partitions
- Reduce I/O
- Maximize query performance
  - Reduce table scan phase



# WELL CLUSTERED EXAMPLE

```
SELECT *\nFROM sales\nWHERE sales_date = '5-Feb';
```

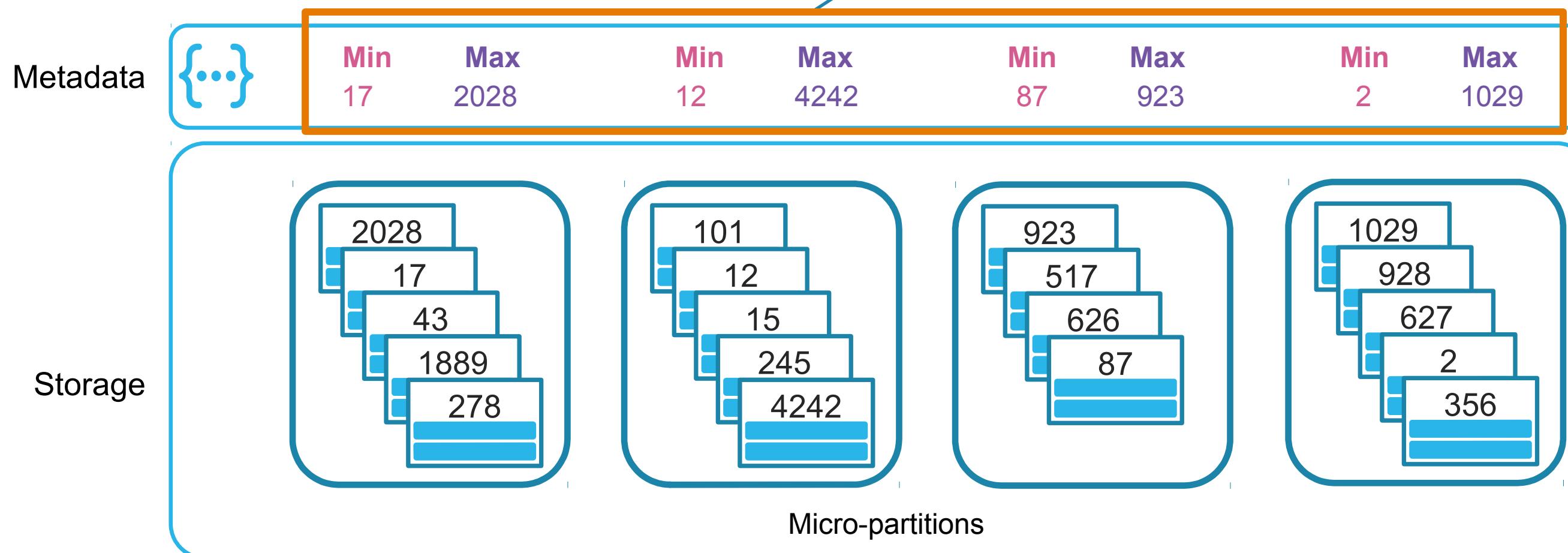
**Good outcome:**  
Only data from one  
micro-partition will  
be included



# POORLY CLUSTERED EXAMPLE

```
SELECT *\nFROM sales\nWHERE customer_id = 101;
```

**Poor outcome:**  
All micro-partitions  
must be included



# PRUNING RESULT

```
SELECT *  
FROM store_sales  
WHERE ss_sold_date_sk = 2451234;
```

```
SELECT *  
FROM store_sales  
WHERE ss_customer_sk = 101444;
```

## Statistics

Scan progress 100.00%

Bytes scanned 436.23MB

Percentage scanned from cache 0.00%

Bytes written to result 510.49MB

Bytes sent over the network 0.01MB

Partitions scanned 28

Partitions total 86546

## Statistics

Scan progress 100.00%

Bytes scanned 175.39GB

Percentage scanned from cache 0.00%

Bytes written to result 0.03MB

Bytes sent over the network 0.87MB

Partitions scanned 86546

Partitions total 86546

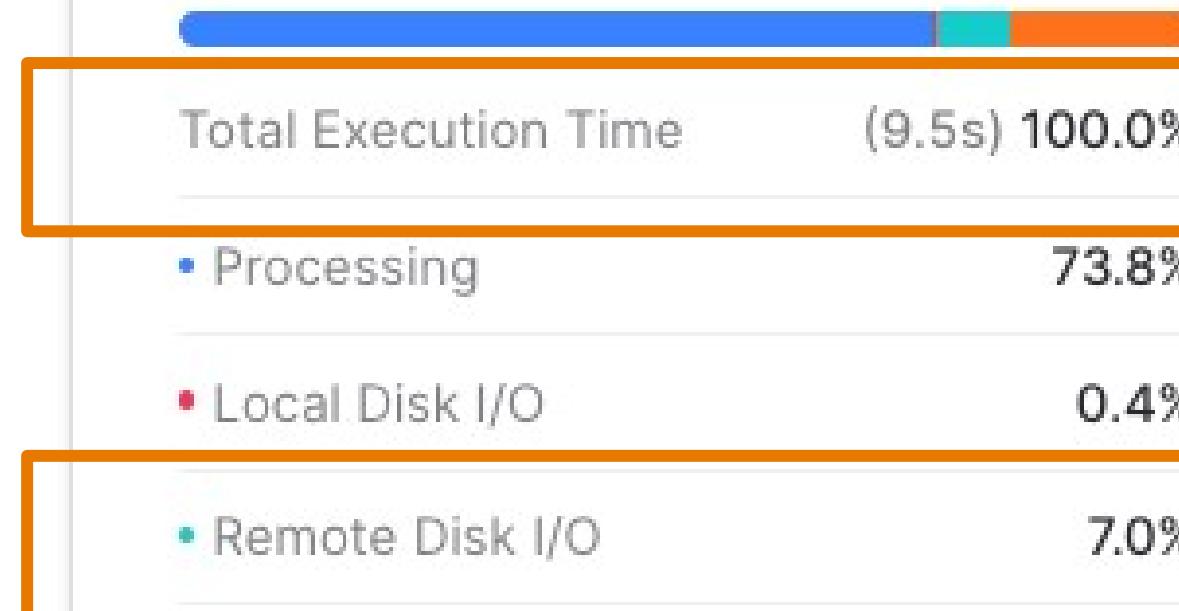


# PRUNING BENEFITS

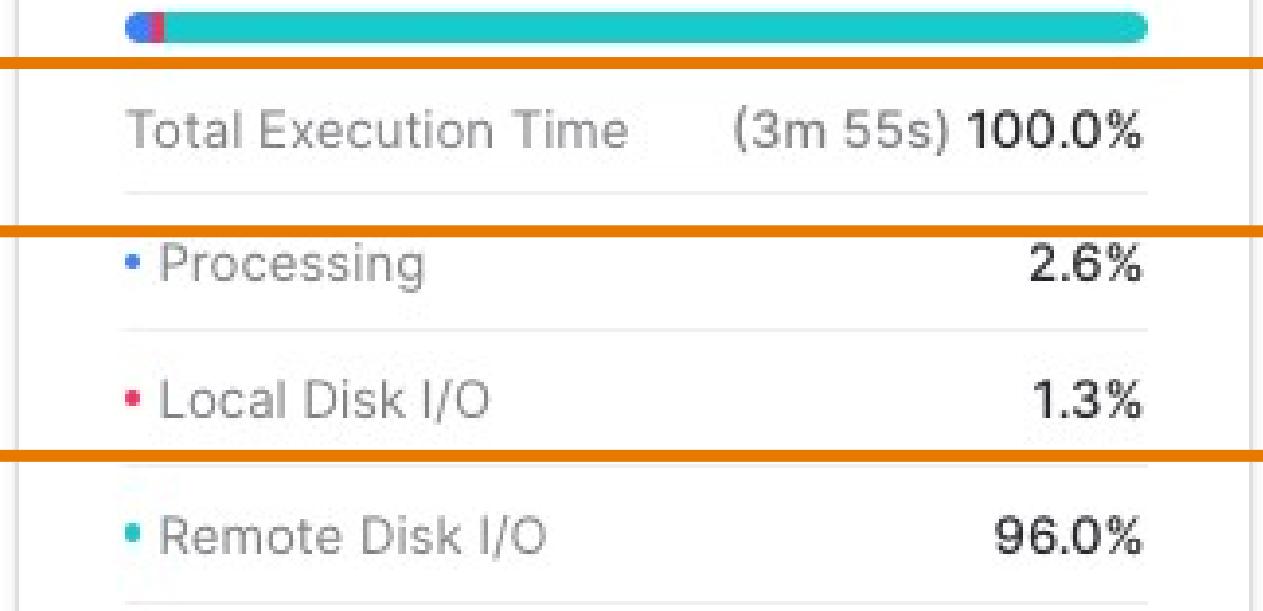
```
SELECT *  
FROM store_sales  
WHERE ss_sold_date_sk = 2451234;
```

```
SELECT *  
FROM store_sales  
WHERE ss_customer_sk = 101444;
```

## Profile Overview (Finished)



## Profile Overview (Finished)



# CHANGING NATURAL CLUSTERING

You can change the natural clustering of data you have loaded

- Method 1: Re-order the data in your input files
  - Good solution if it is simple to adjust your pipeline
- Method 2: Sort data after loading
  - Computationally expensive
  - Not recommended unless data is relatively static
- Method 3: Apply a clustering key
  - Only recommended in specific circumstances (stay tuned to learn when!)



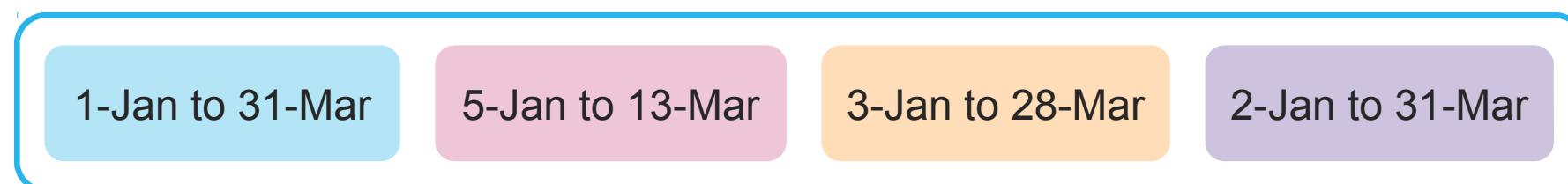
# EVALUATING CLUSTERING



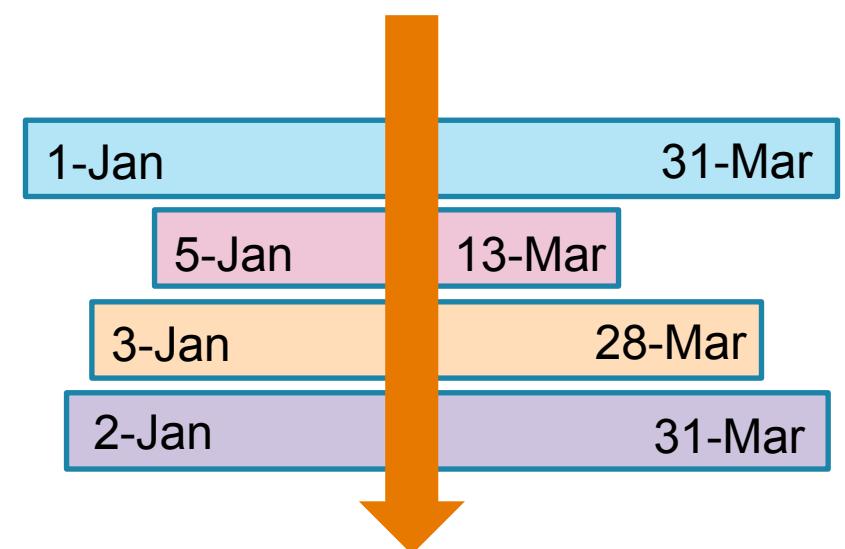
# AVERAGE CLUSTERING DEPTH

# MANY OVERLAPPING VALUES

## Metadata



## Average Depth = 3.9



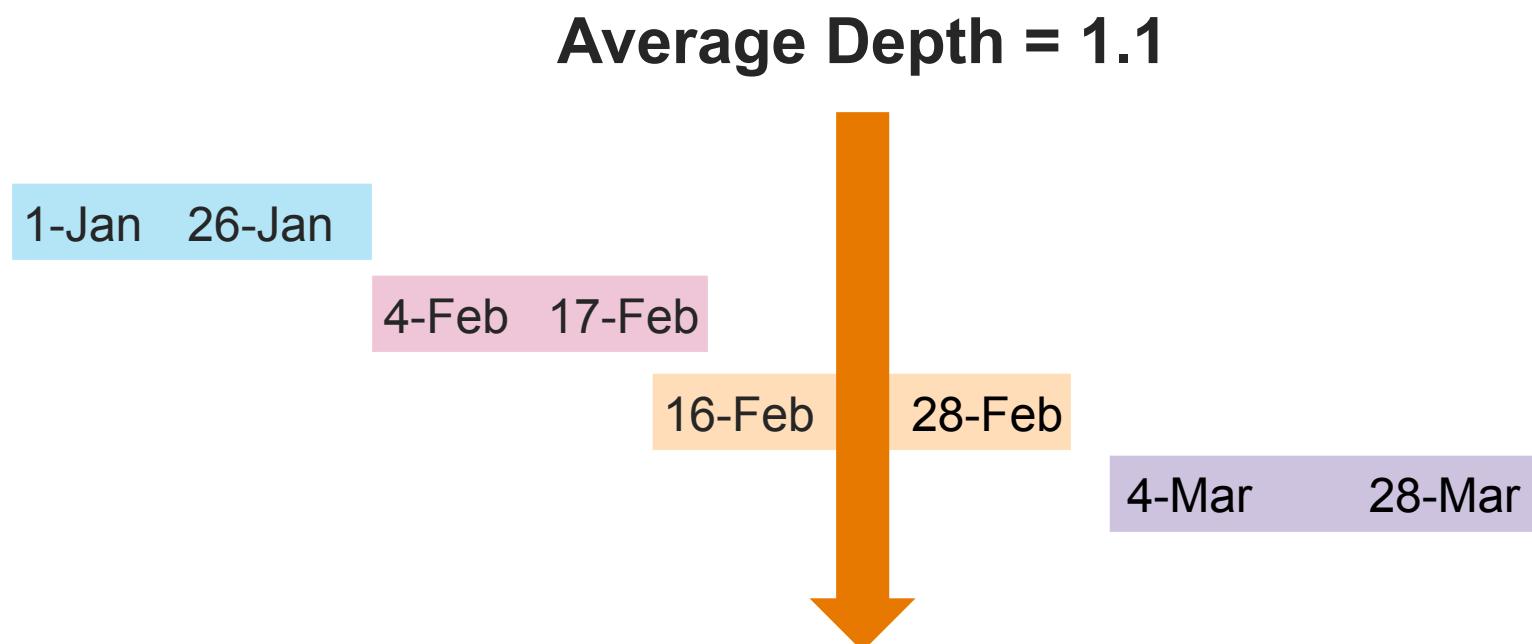
- A measure of how much overlap there is in the values between micro-partitions
  - High average clustering depth means:
    - Overlapping values in many micro-partitions
    - Less effect pruning
    - Little or no performance improvement based on WHERE clause

# AVERAGE CLUSTERING DEPTH

## FEW OVERLAPPING VALUES

Metadata

1-Jan to 26-Jan    14-Feb to 17-Feb    16-Feb to 28-Feb    4-Mar to 28-Mar

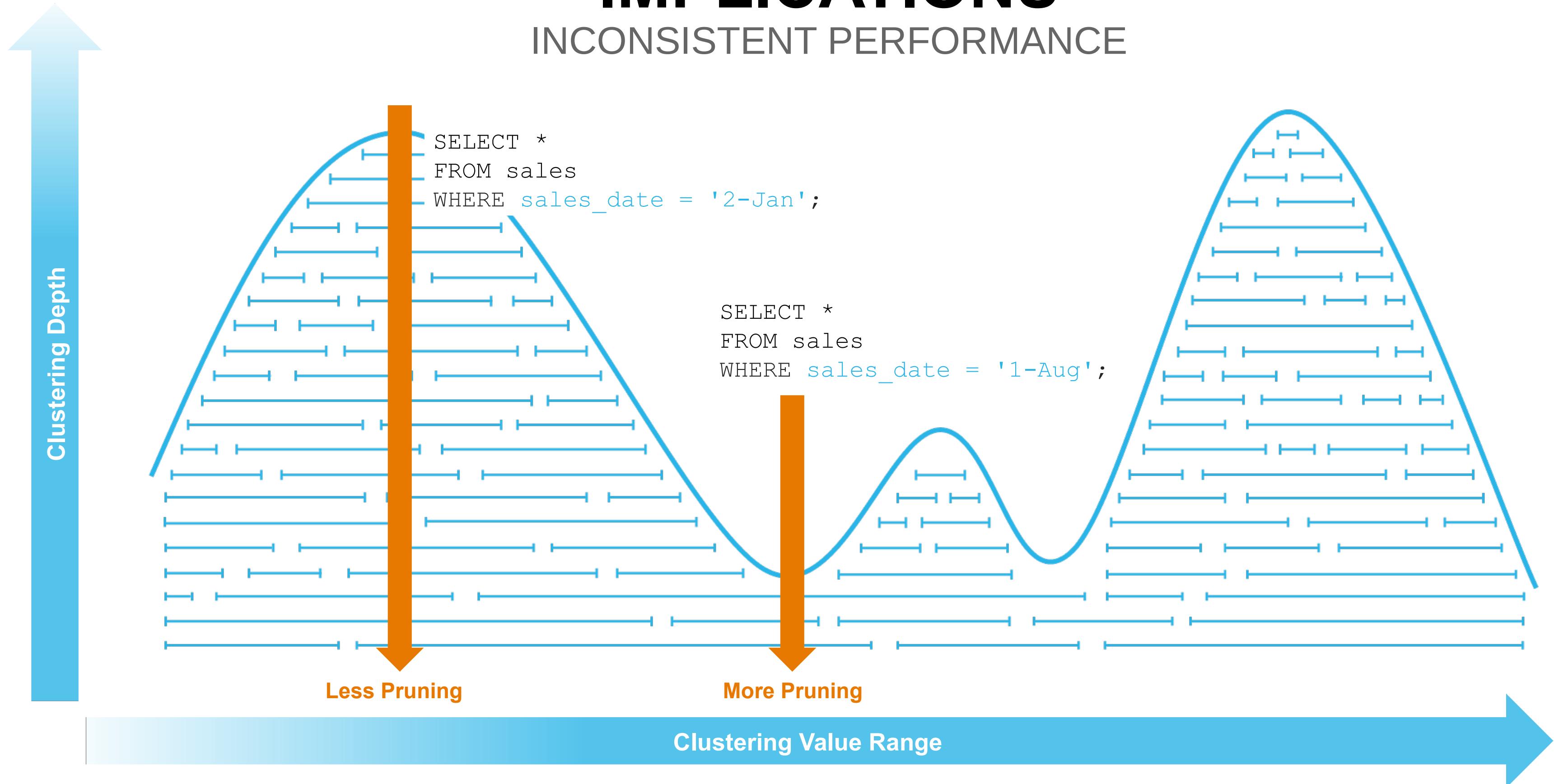


- Low average depth means:
  - Few micro-partitions with overlapping values
  - Better pruning
  - Improved performance



# IMPLICATIONS

## INCONSISTENT PERFORMANCE



# SYSTEM\$CLUSTERING\_INFORMATION

- Function used to evaluate clustering
- Returns output in JSON format

```
SELECT SYSTEM$CLUSTERING_INFORMATION('store_sales', '(ss_sold_date_sk)');
```

System  
function

Table name

One or more  
column names



# READING CLUSTERING INFORMATION

```
SELECT SYSTEM$CLUSTERING_INFORMATION('orders', '(o_orderdate)');

{
  "cluster_by_keys" : "LINEAR(o_orderdate)",
  "total_partition_count" : 3242,
  "total_constant_partition_count" : 1409,
  "average_overlaps" : 1.4553,
  "average_depth" : 1.8436,
  "partition_depth_histogram" : {
    "00000" : 0,
    "00001" : 1364,
    "00002" : 1362,
    "00003" : 272,
    "00004" : 151,
    "00005" : 89,
    "00006" : 4,
    "00007" : 0,
    ...
    "00016" : 0
  }
}
```



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Column(s) being evaluated



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Total number of  
micro-partitions that  
make up the table



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Total "stable" micro-partitions:  
unlikely to benefit from clustering



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Average number of other micro-partitions that a given micro-partition will overlap with



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Average clustering depth for all the micro-partitions in the table



# READING CLUSTERING INFORMATION

```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Histogram showing:  
"<depth>" : <number of micro-partitions at that depth>



# KEY CLUSTERING METRICS

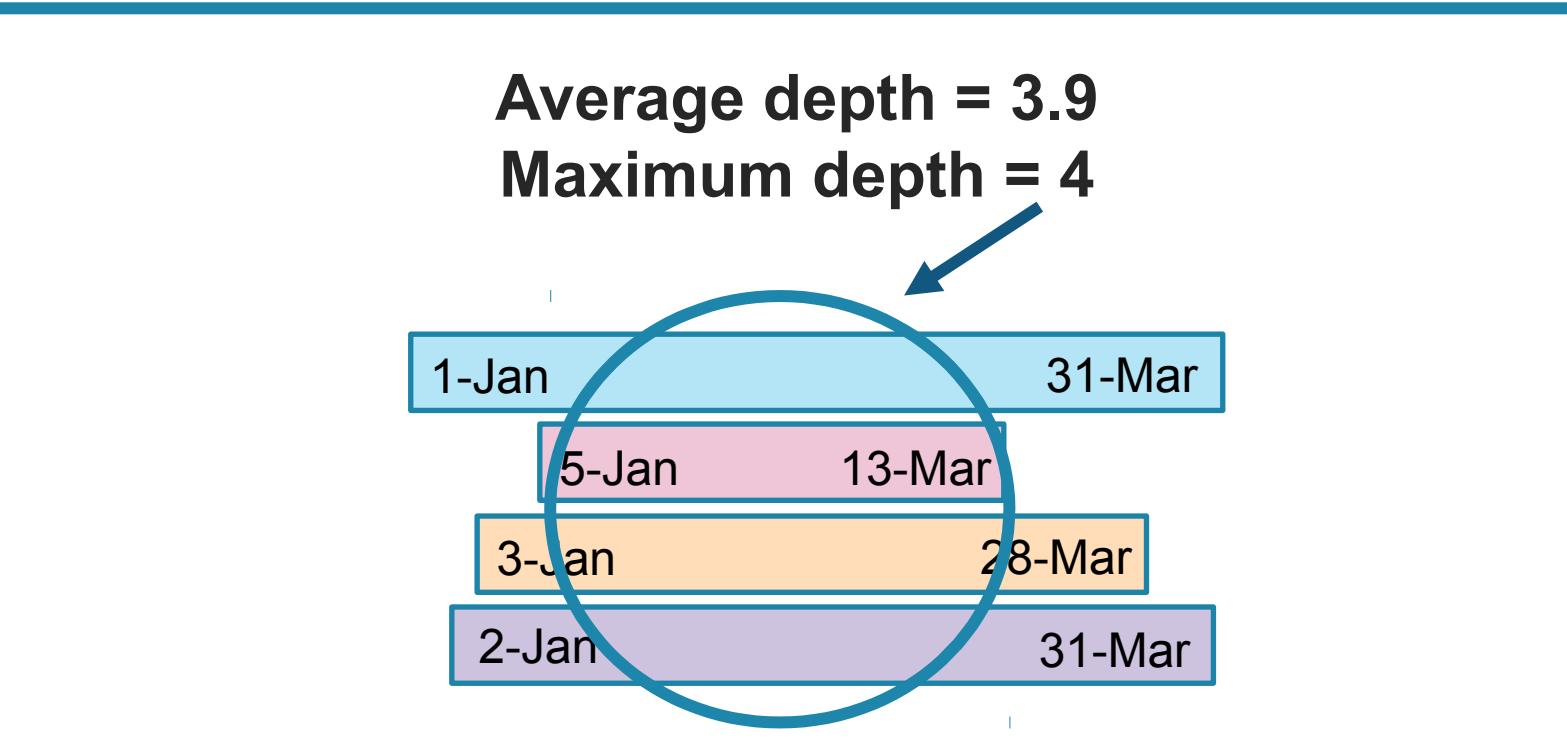
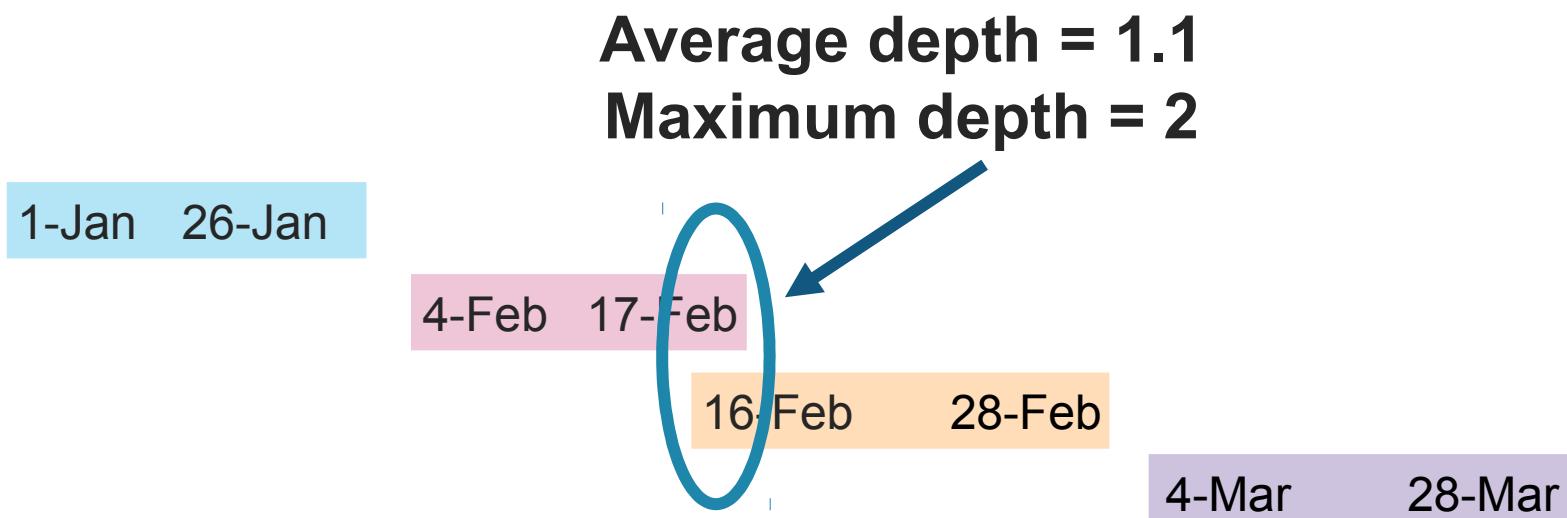
```
{  
    "cluster_by_keys" : "LINEAR(o_orderdate)",  
    "total_partition_count" : 3242,  
    "total_constant_partition_count" : 1409,  
    "average_overlaps" : 1.4553,  
    "average_depth" : 1.8436,  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 1364,  
        "00002" : 1362,  
        "00003" : 272,  
        "00004" : 151,  
        "00005" : 89,  
        "00006" : 4,  
        "00007" : 0,  
        ...  
        "00016" : 0  
    }  
}
```

Average depth

Maximum depth (6)



# KEY CLUSTERING METRICS



## Average Clustering Depth

- The average number of micro-partitions read for a specific value
- Lower numbers indicate better pruning
- Values <10 are generally good

## Maximum Clustering Depth

- The maximum number of micro-partitions read for a specific value
- Values < 10 are generally good



# CLUSTERING PROFILE

## Well clustered

- Average depth = 1.8
- Maximum depth = 5

```
{  
  ...  
  "average_depth" : 1.8436,  
  "partition_depth_histogram" : {  
    "00000" : 0,  
    "00001" : 1364,  
    "00002" : 1362,  
    "00003" : 272,  
    "00004" : 151,  
    "00005" : 89,  
  }
```

## Poorly clustered

- Average depth = 3,242
- Maximum depth = 3,242

```
{  
  ...  
  "average_depth" : 3242.0,  
  "partition_depth_histogram" : {  
    "00000" : 0,  
    "00001" : 0,  
    ...  
    "00016" : 0,  
    "04096" : 3242  
  }
```



# IMPLEMENT AND TEST CLUSTER KEYS



# CLUSTER KEY

```
ALTER TABLE orders  
CLUSTER BY (customer_id);
```

```
CREATE TABLE orders (  
    o_orderkey    NUMBER  
,   o_orderdate   TIMESTAMP  
,   o_totalprice  NUMBER)  
CLUSTER BY (<expr 1>, <expr 2>);
```

- Specify which columns (or expressions) should be well clustered for the table
  - They can't ALL be well clustered
- Background task consumes credits
  - Minutes to hours to complete
- Physically re-orders the data
- Goal is predictable performance
  - Clustering will not perfectly order data



# METHODOLOGY



## Driving Goal:

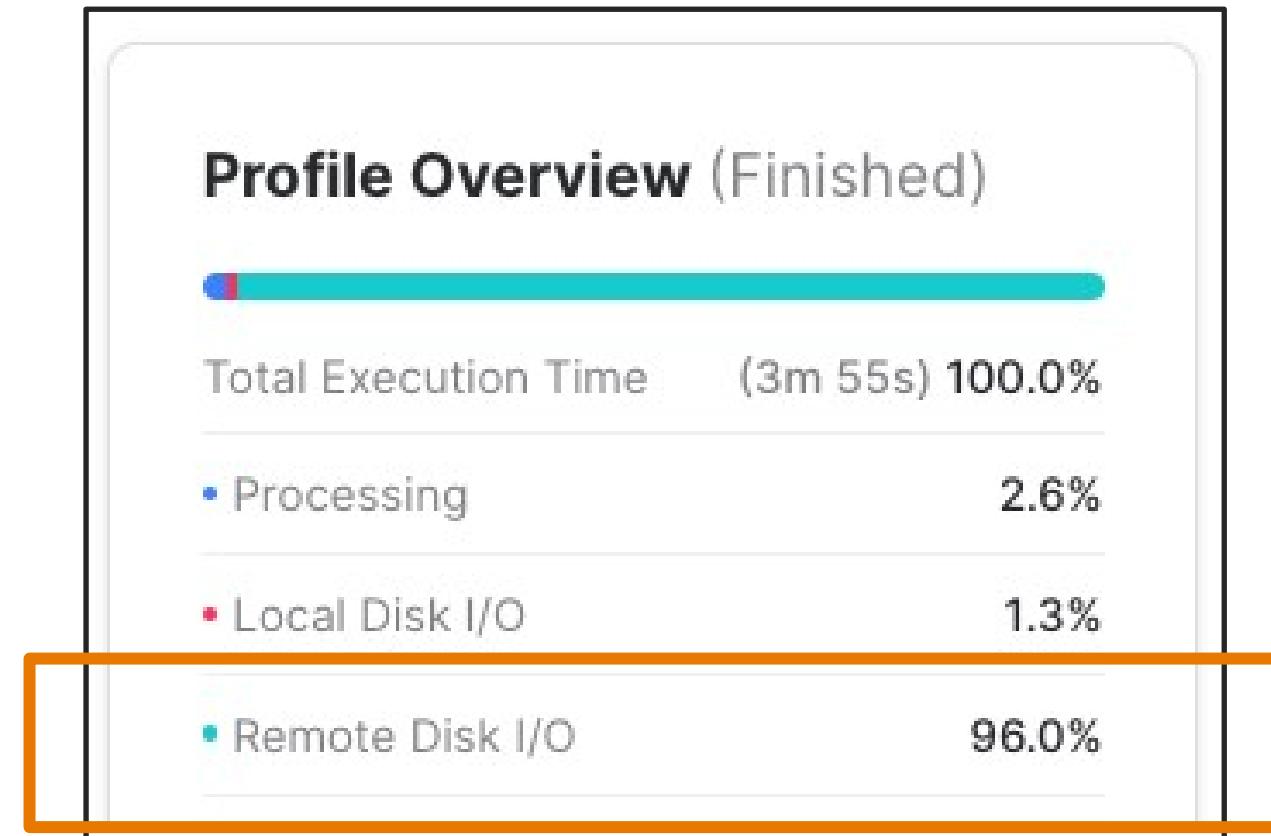
- Balance cost and benefit

## Methodology:

1. Identify use case
2. Benchmark pre-clustering performance
3. Deploy clustering
4. Evaluate post-clustering performance

# 1. IDENTIFY USE CASE

Statistics	
Partitions scanned	86546
Partitions total	86546



- Large (fact) table (>1TB)
  - Ideally, not updated frequently
- High percentage of partitions scanned for columns frequently used in WHERE
- Unacceptable performance
- Significant time spent in remote I/O
  - Table scan



## 2. BENCHMARK EXISTING PERFORMANCE

Task	Notes
Benchmark existing performance with queries that are representative of actual workload	<p>Use the views and table functions in ACCOUNT_USAGE and INFORMATION_SCHEMA</p> <p>QUERY_HISTORY can be especially helpful</p>
Produce a baseline to compare to post-clustering query performance	For example: 90% of the queries run in under 10 minutes
Agree on success criteria	For example: 90% of the queries run in under 5 minutes



# 3. IDENTIFY AND DEPLOY CLUSTER KEY

Best Practice	Reason
Avoid unique keys	Excellent query performance, but high cost to keep the table clustered
Do not cluster by a TIMESTAMP column - instead, cluster by DATE_TRUNC('DAY', TS_COLUMN)	Updates or deletes on a TIMESTAMP column may be prohibitively expensive to re-cluster
Avoid low-cardinality keys (for example, gender)	Will not provide enough pruning to produce a significant performance gain
Prioritize columns frequently used in WHERE clause	Pruning will positively impact the greatest number of queries
Limit cluster key to <= 3 columns or expressions	Diminishing ROI for each additional column
With multiple columns, specify the lowest cardinality column first	Results in larger chunks of well-clustered data



# CLUSTERING AND CARDINALITY EXAMPLE

STATUS	STORE_NO	CUST_NUM	TOTAL
APPROVED	17	01236	2245.87
APPROVED	17	11835	123.95
APPROVED	22	44390	225.87
APPROVED	22	33210	1.87
APPROVED	22	00236	2245.98
APPROVED	25	44039	1328.99
APPROVED	26	93012	145.33
APPROVED	26	01236	358.33
COMPLETE	17	44210	24.97
COMPLETE	21	55439	3.98
COMPLETE	22	22409	987.54
COMPLETE	22	42239	493.22
COMPLETE	22	64452	87.33
COMPLETE	25	99325	23.87
COMPLETE	26	01266	449.20
COMPLETE	26	32009	53.28
COMPLETE	27	52393	182.45
COMPLETE	27	43992	3356.87
COMPLETE	27	88423	192.45
COMPLETE	30	62345	312.87

Clustered by  
(STATUS, STORE\_NO)

Clustered by  
(STORE\_NO, STATUS)

STATUS	STORE_NO	CUST_NUM	TOTAL
APPROVED	17	01236	2245.87
APPROVED	17	11835	123.95
COMPLETE	17	44210	24.97
COMPLETE	21	55439	3.98
APPROVED	22	44390	225.87
APPROVED	22	33210	1.87
APPROVED	22	00236	2245.98
COMPLETE	22	22409	987.54
COMPLETE	22	42239	493.22
COMPLETE	22	64452	87.33
APPROVED	25	44039	1328.99
COMPLETE	25	99325	23.87
APPROVED	26	93012	145.33
APPROVED	26	01236	358.33
COMPLETE	26	01266	449.20
COMPLETE	26	32009	53.28
COMPLETE	27	52393	182.45
COMPLETE	27	43992	3356.87
COMPLETE	27	88423	192.45
COMPLETE	30	62345	312.87



# 4. EVALUATE THE RESULT

- Re-run your initial benchmark test
  - Does reduced compute from fast queries offset the cost of clustering?
  - Initial clustering operation should be the most expensive
- Evaluate pre- and post-clustering storage costs
  - More additional storage on frequently-updated tables
  - Temporarily reduce `DATA_RETENTION_TIME_IN_DAYS` for initial clustering operation
- Evaluate ongoing re-clustering costs
  - Consider suspending and resume clustering
- Did you meet your agreed-upon success criteria?



# PERFORMANCE OPTIMIZATION



# MODULE AGENDA

- Search Optimization
- Materialized Views
- Materialized View Use Cases
- When to Use Materialized Views



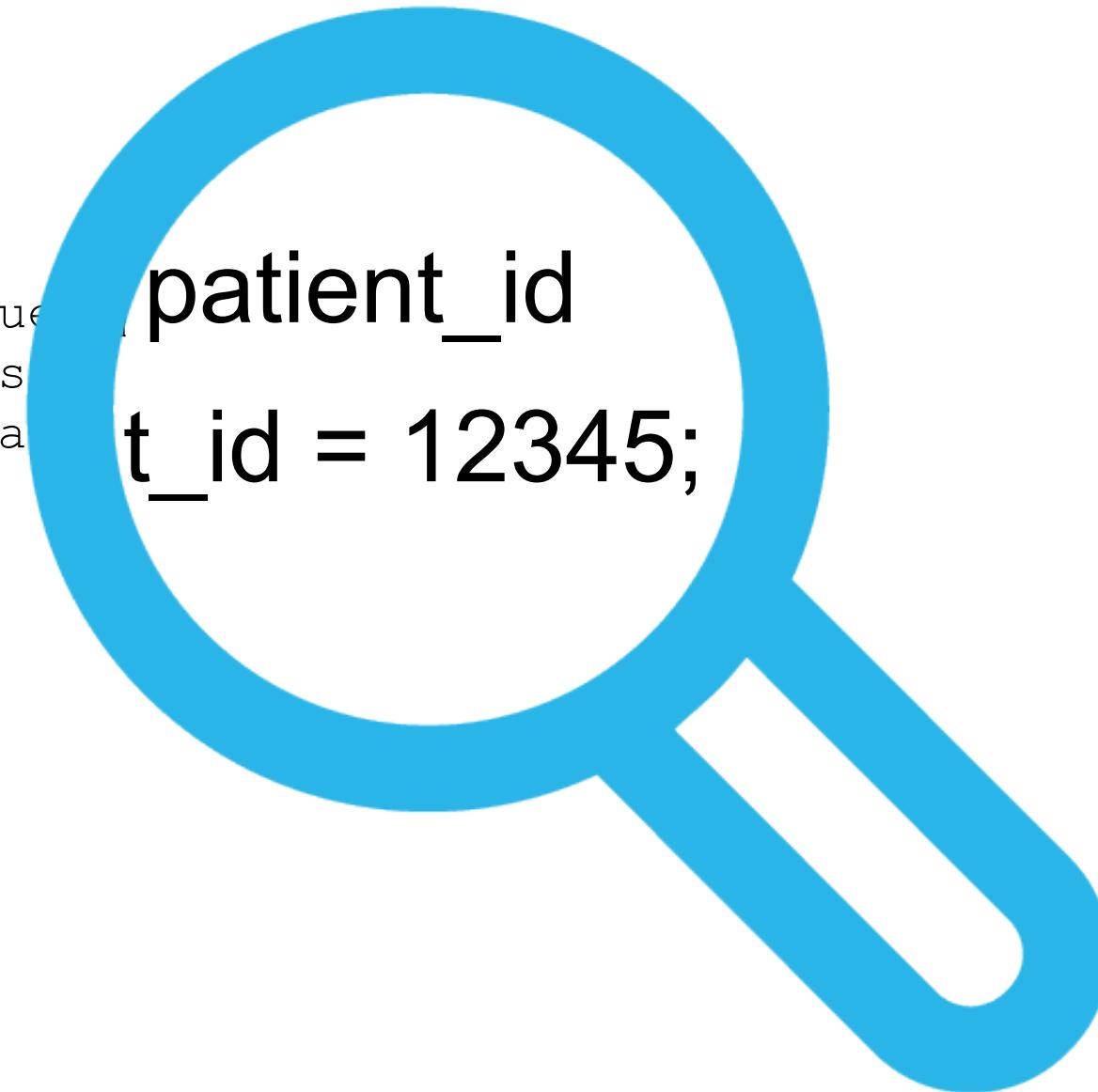
# SEARCH OPTIMIZATION



# OVERVIEW

## WHEN TO USE SEARCH OPTIMIZATION

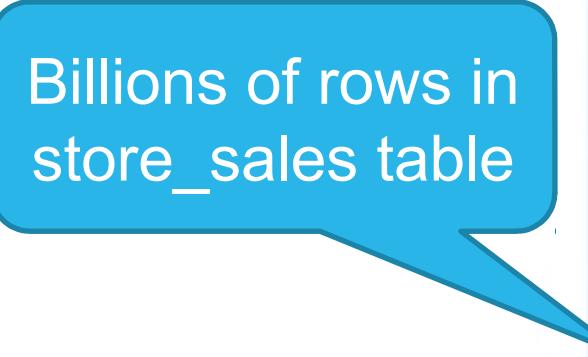
```
SELECT bal_due  
FROM patients  
WHERE pa
```



- Equality searches
  - Also known as point query or fast lookup query
  - Returns only a small % of rows
- Tune tables with frequent selective queries using Search Optimization

# SELECTIVE QUERY EXAMPLE

- Looking for a single value (or small number) in a large table
- Put a Search Optimization on the target table to increase performance

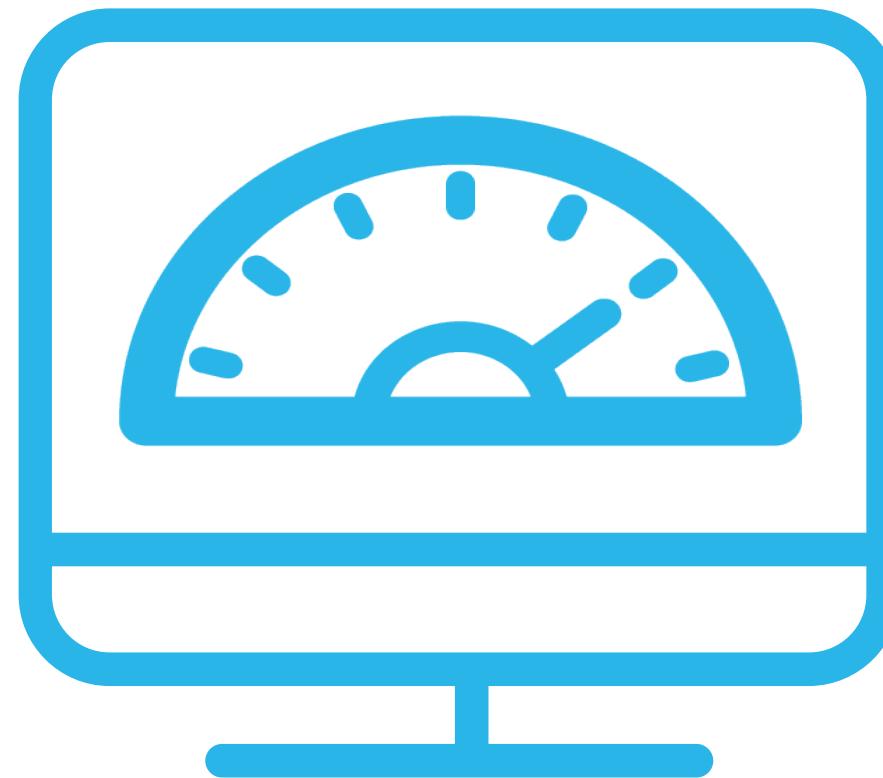


```
-- Find the sales ticket, items sold and sale date  
-- for one customer  
  
SELECT ss_ticket_number, ss_sold_date_sk, ss_item_sk  
FROM store_sales  
WHERE  
ss_customer_sk = 3229284;
```



Only a small number of rows meet the criteria for result

# CREATE A SEARCH PATH



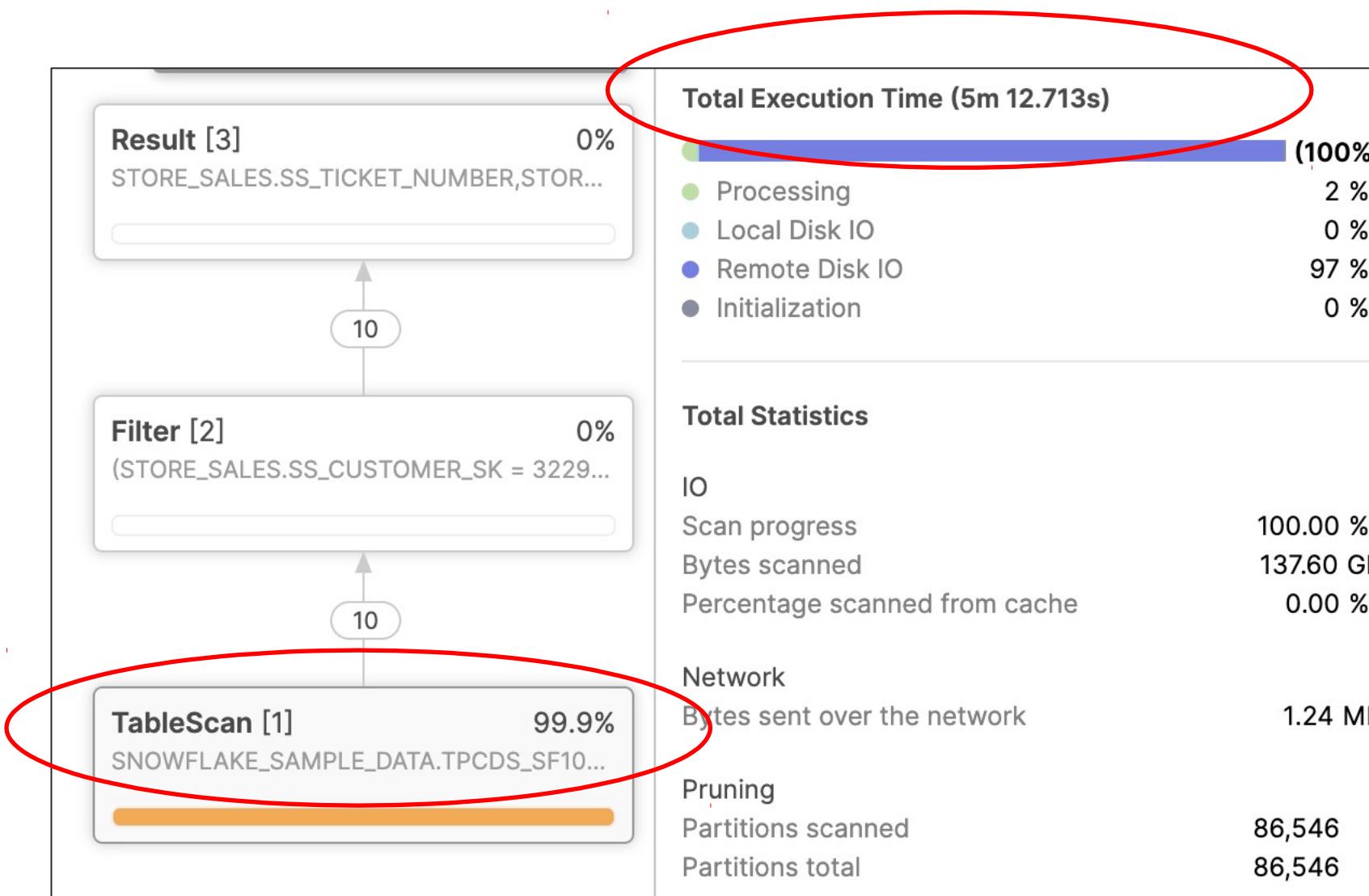
- Add table to the Search Optimization service:

```
ALTER TABLE store_sales  
ADD SEARCH OPTIMIZATION;
```

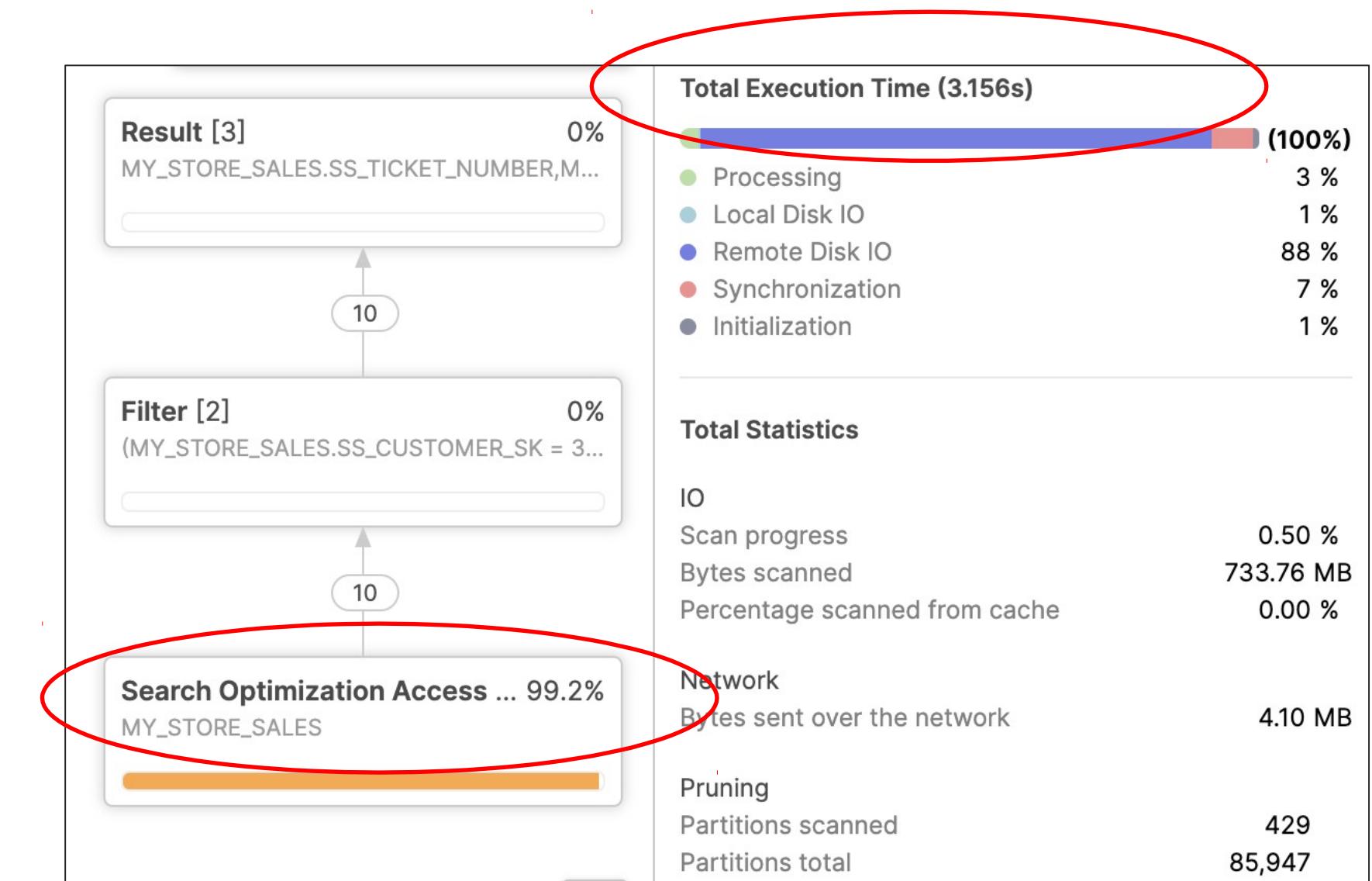
- Search service builds the search access path to optimize point queries on table
  - Will take some time to build the optimization

# PERFORMANCE BENEFIT

## Without search optimization

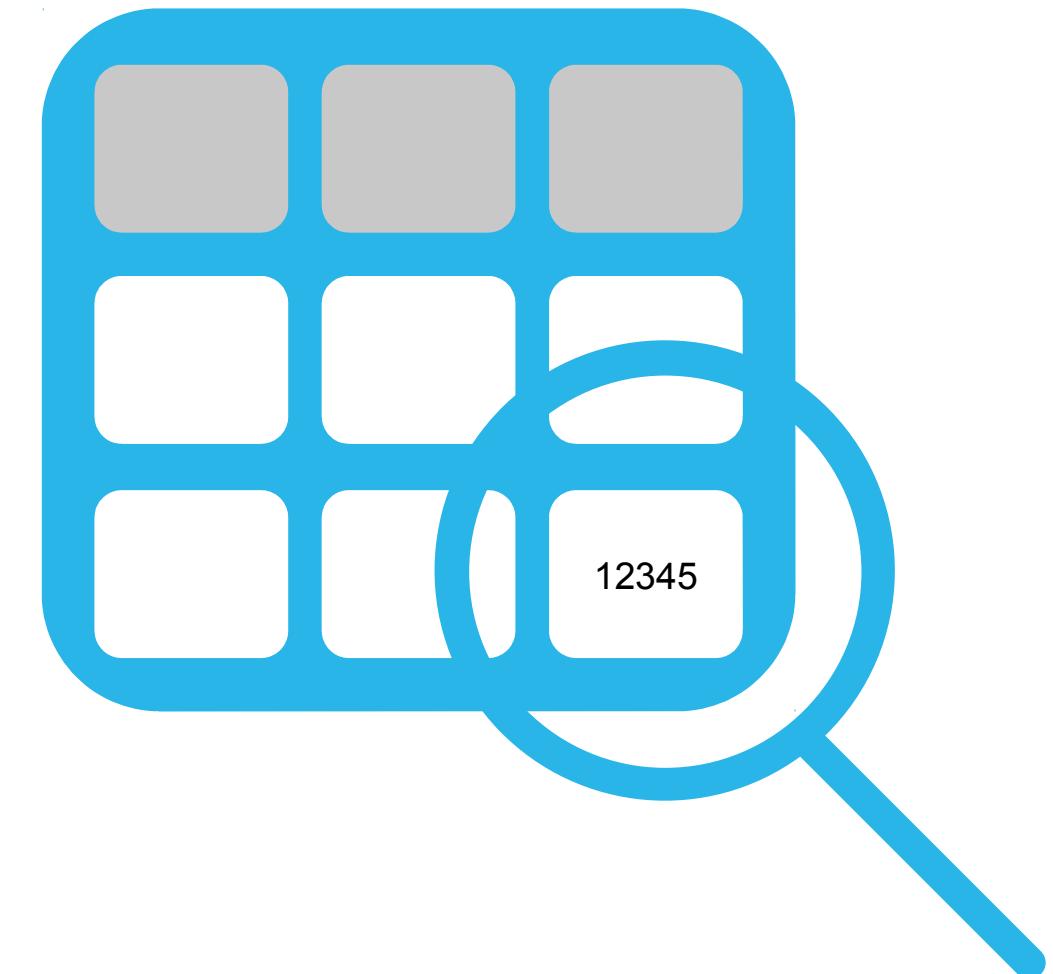


## With search optimization



# WHAT IS A SEARCH OPTIMIZATION?

- A data structure that stores the optimized search access path
  - One per table
- Contains **all** columns of a permanent table
- Maintained in a separate structure (storage)
- Maintained as changes are made to the base table



# COST CONSIDERATIONS

- The larger the number of high cardinality columns, the higher the cost
- Tables with frequent mutations (from DML operations) will also result in higher cost
- Estimate search optimization costs with:

```
SYSTEM$ESTIMATE_SEARCH_OPTIMIZATION_COSTS ('<table name>')
```

- Estimate requires 7 days of history



# MONITORING SEARCH OPTIMIZATION SERVICE

- SEARCH\_OPTIMIZATION\_HISTORY
  - ACCOUNT\_USAGE view in SNOWFLAKE database
  - INFORMATION\_SCHEMA table function
- SHOW TABLES
  - SEARCH\_OPTIMIZATION column exposes additional storage use for storage optimization

Column Name	Data Type
START_TIME	TIMESTAMP_LTZ
END_TIME	TIMESTAMP_LTZ
CREDITS_USED	TEXT
TABLE_ID	NUMBER
TABLE_NAME	TEXT
SCHEMA_ID	NUMBER
SCHEMA_NAME	TEXT
DATABASE_ID	NUMBER
DATABASE_NAME	TEXT



# LAB EXERCISE: 16

## Query Optimization

15 minutes

- Explore Query Performance
- Explore GROUP BY and ORDER BY Operation Performance
- Querying with LIMIT Clause
- JOIN Optimizations in Snowflake

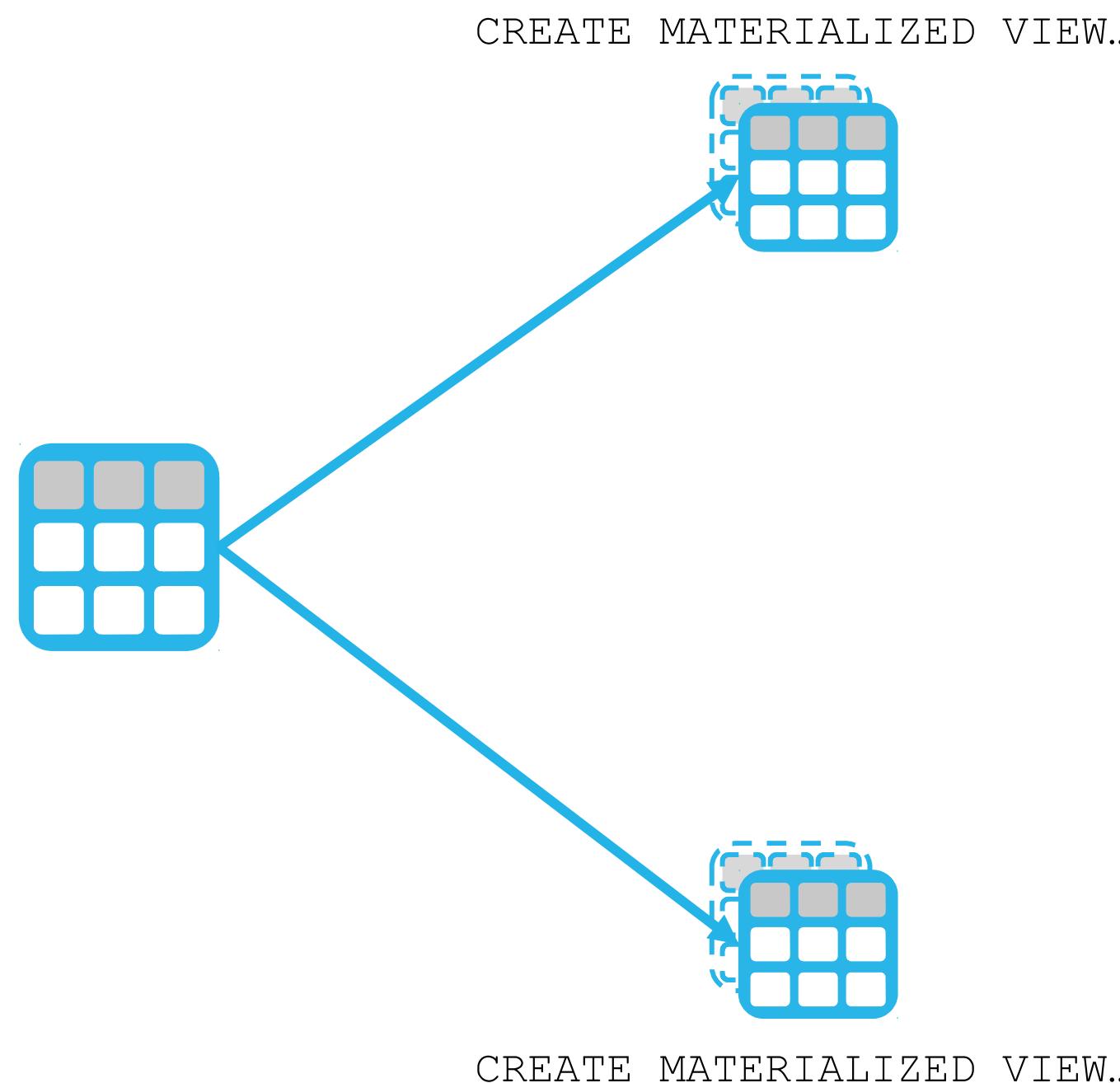


# MATERIALIZED VIEWS



# MATERIALIZED VIEW

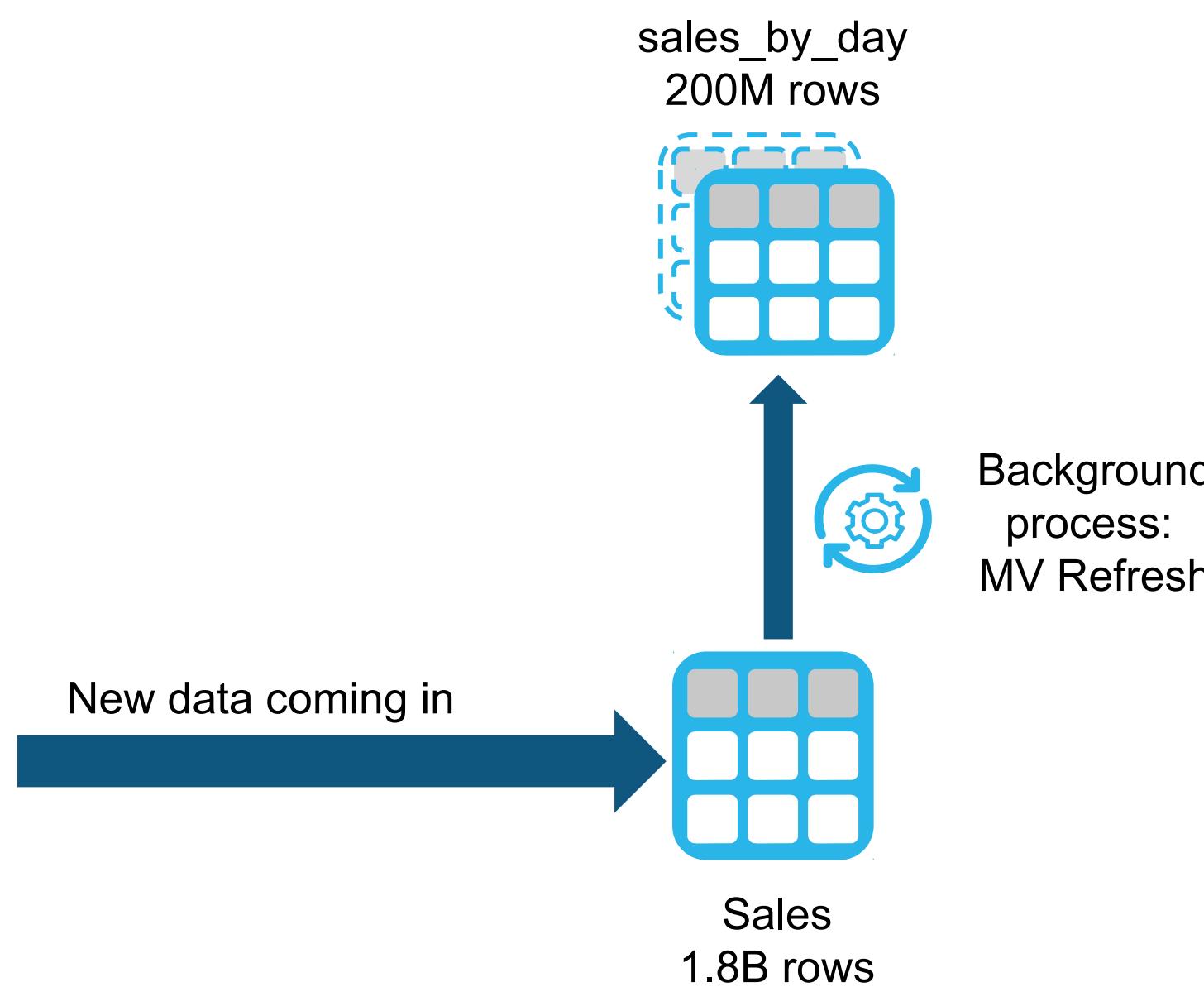
## MATERIALIZED QUERY RESULT



- *Materialized SQL query results*
- Purpose: maximize query performance
- Use cases:
  - Summary tables
  - Repeated queries - complex calculations
  - Cache recent entries from External Tables
  - Fast semi-structured data analysis
  - Optimized projections for different use-cases

# AUTOMATIC REFRESH

## FOR DATA CONSISTENCY

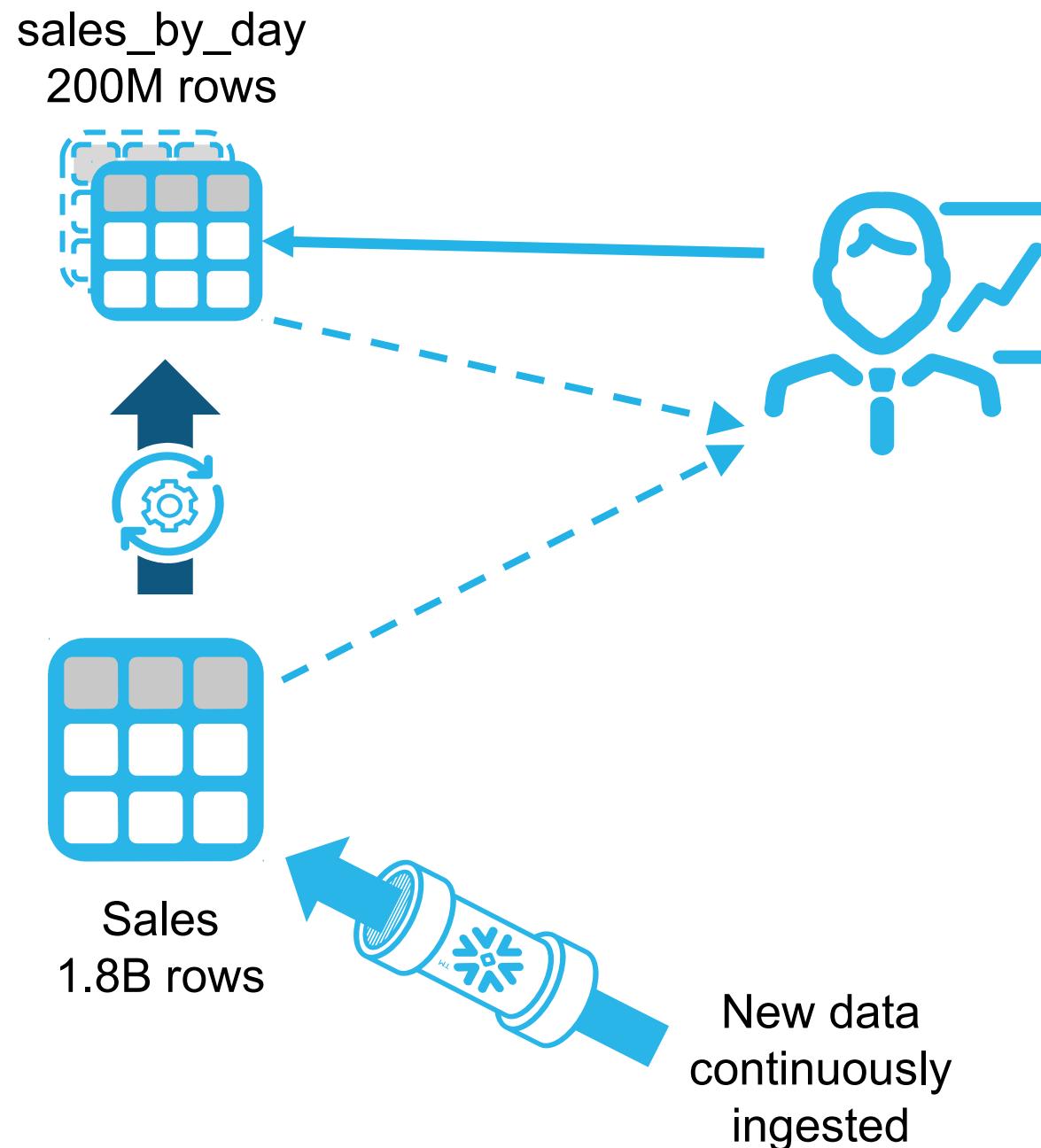


- Materialized views automatically refreshed
  - Background process – per second billing
- Track credit cost: MATERIALIZED VIEW REFRESH HISTORY
- Also, adds to storage costs



# DATA CONSISTENCY

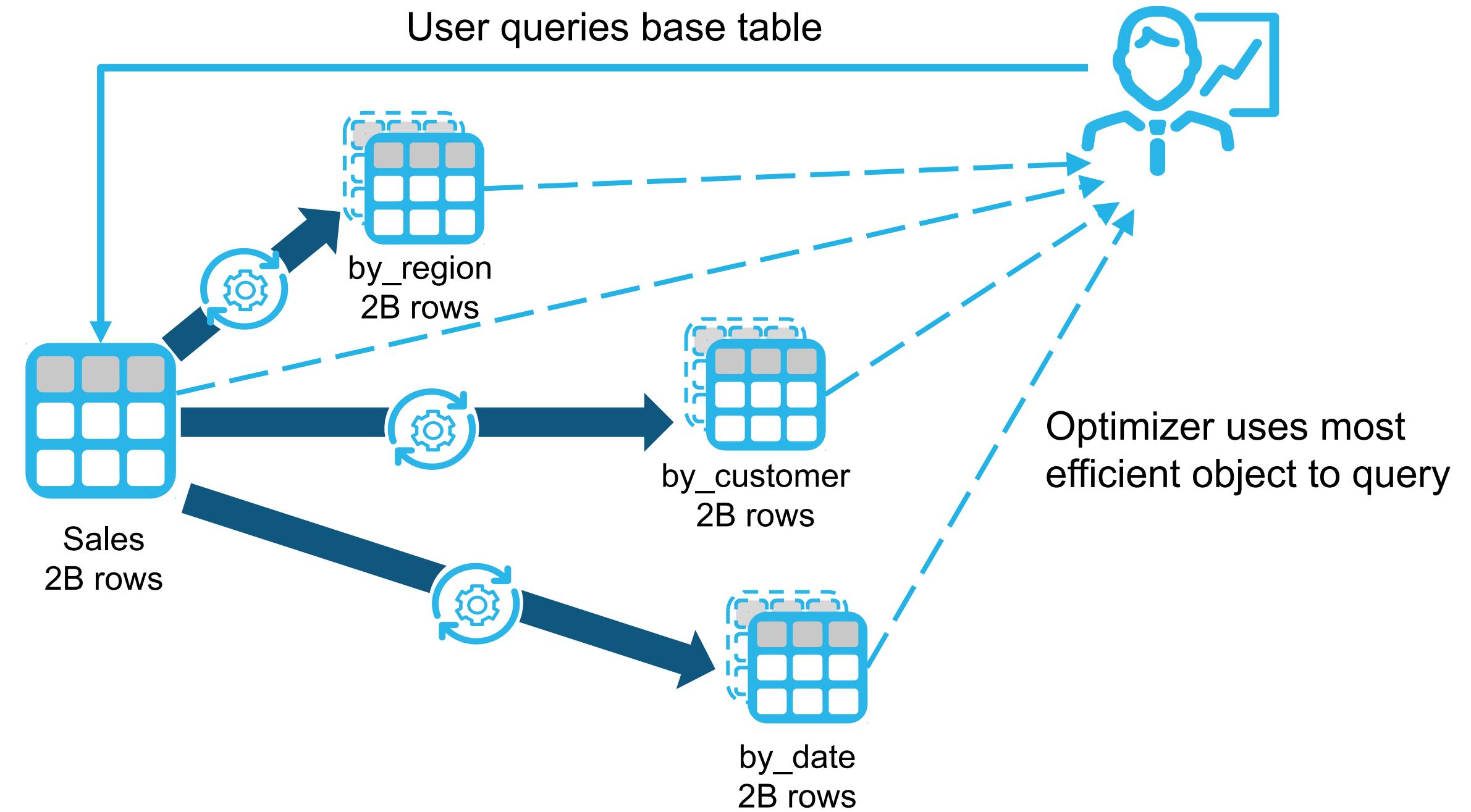
## ENSURING ACCURATE RESULTS



- Queries against view always consistent with base table
- If query comes in before materialized view is fully updated, results may combine materialized view and base table data
  - Example: querying  $\text{SUM}(\text{sales})$  for month, current day results are still coming in

# AUTOMATIC QUERY REWRITE

- Cost-based optimizer finds fastest access to data
- Rewrites query to use materialized view if that will be fastest

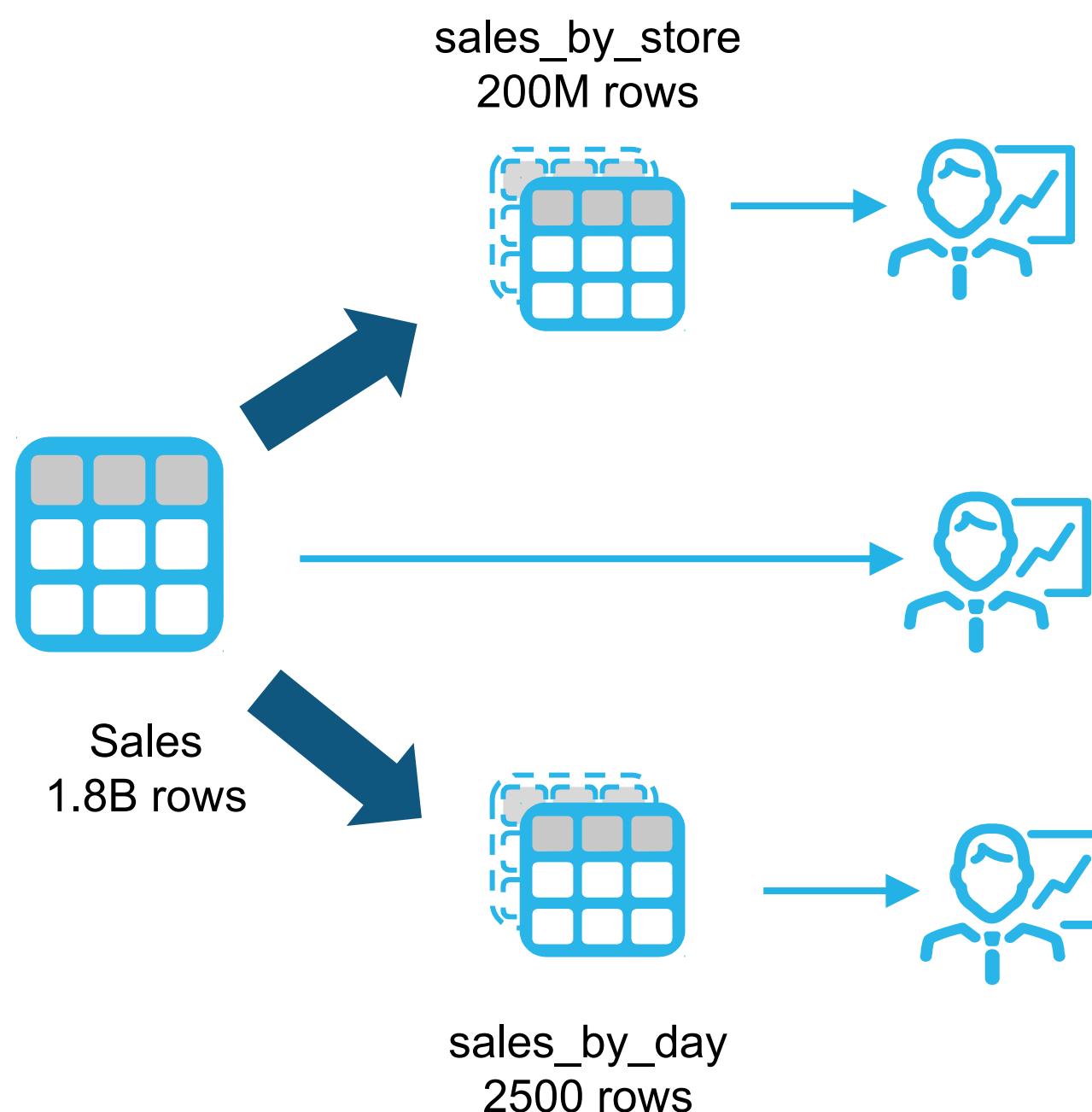


# MATERIALIZED VIEW USE CASES



# SUMMARY TABLES

## AGGREGATE RESULTS



### Challenge:

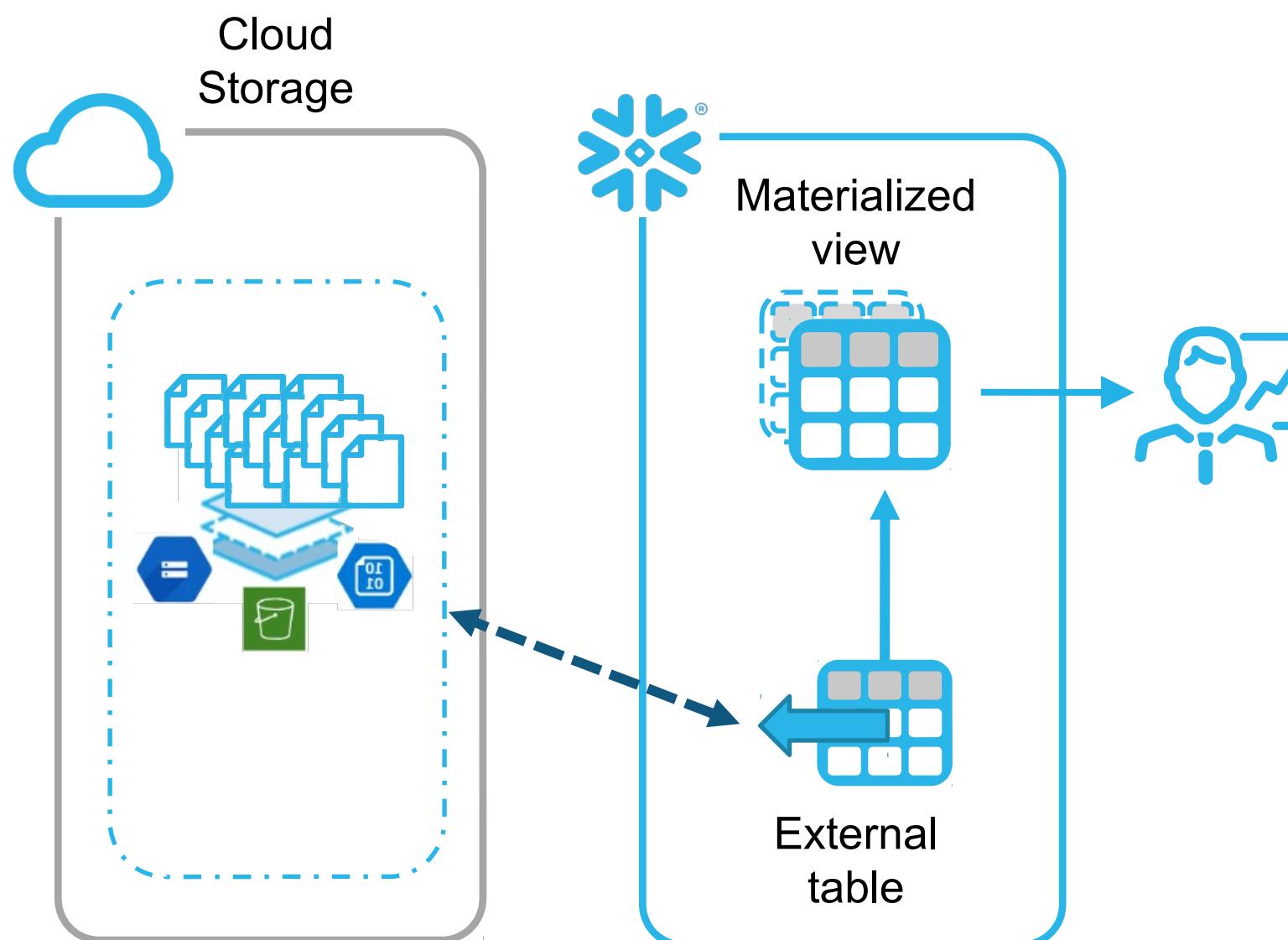
- Query performance against billions of rows
- Repeated queries
- Complex run-time calculations

### Solution:

- Multiple aggregates to reduce data volumes
- Calculate materialized view once, query repeatedly

# EXTERNAL TABLES

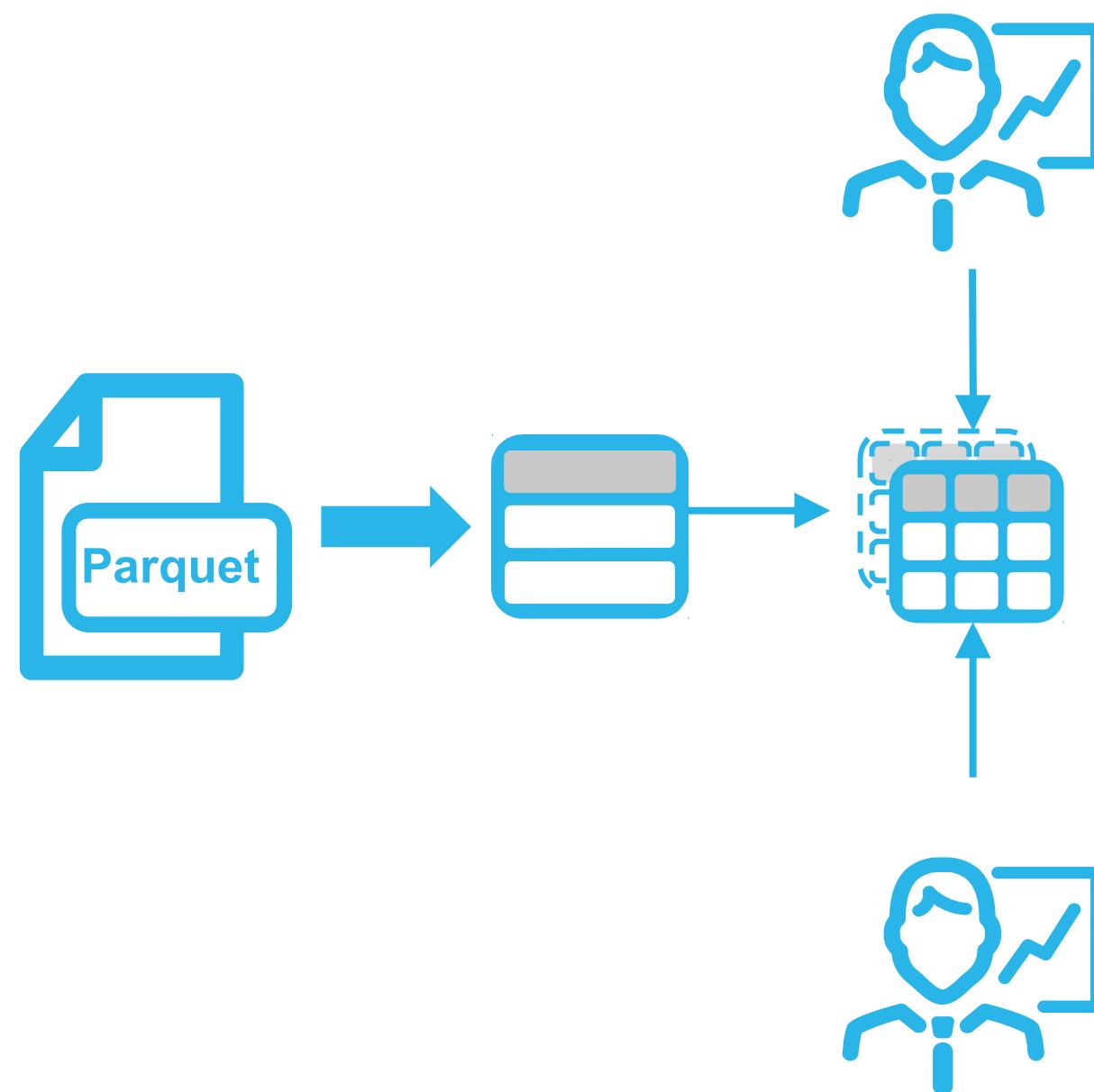
## CACHE RECENT EXTERNAL TABLE DATA



- Challenge:
  - Query performance against external tables
  - 80% of queries over recent data
- Solution:
  - Materialized view over external table
  - MV holds data for last 12 months
  - 80% of queries – fast performance
  - 20% of queries – against external table
  - Materialized view refreshed when data changes on external table

# SEMI-STRUCTURED DATA

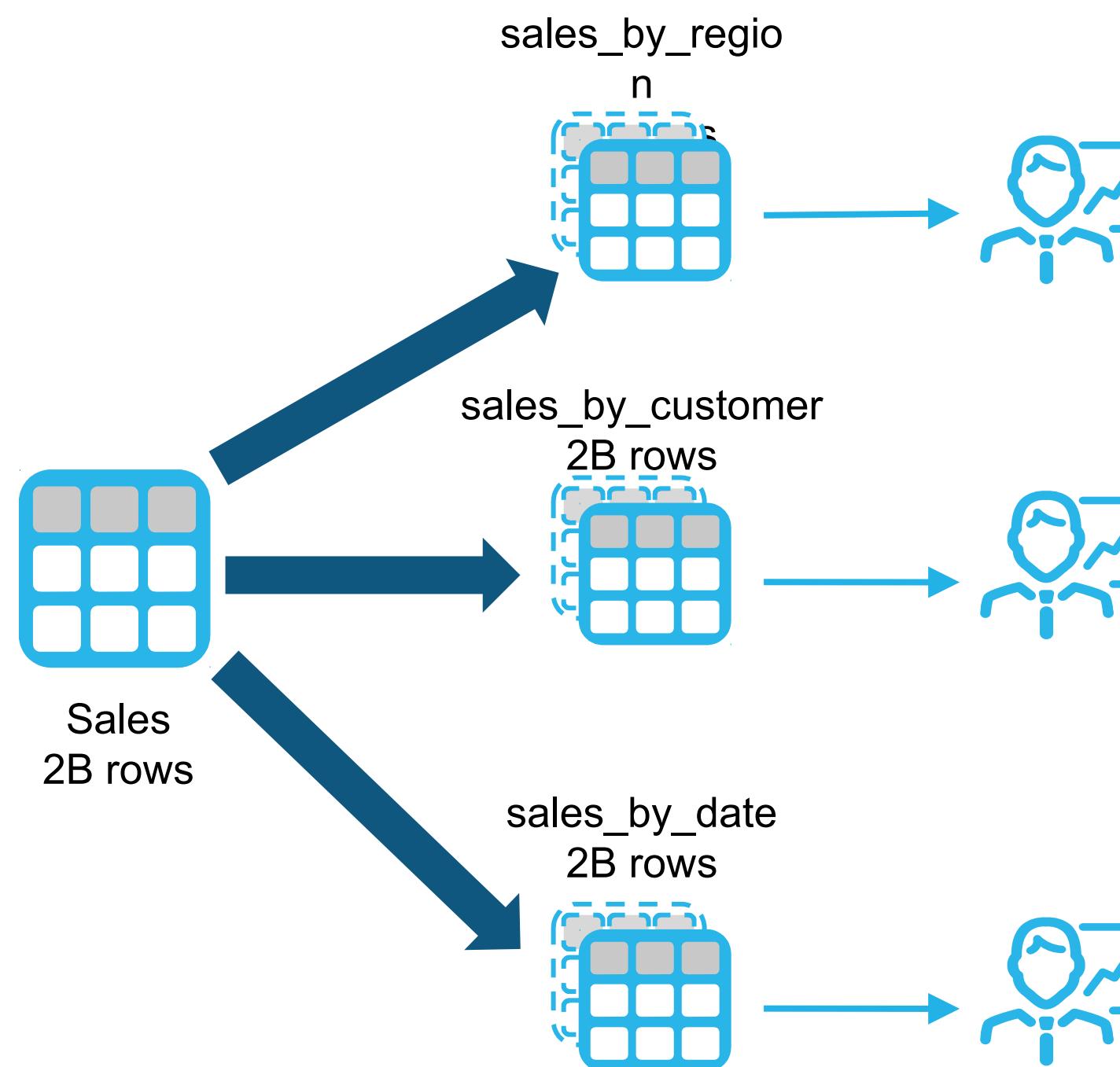
FLATTEN DATA FOR PERFORMANCE



- Challenge
  - Semi-structured data needs to be flattened
  - Need to centralize conversion logic
  - Slow filtering on DATE or TIMESTAMP
- Solution: Materialized view
  - Centralize logic
  - Pre-flatten to optimized format
  - Improve micro-partition pruning on DATE and TIMESTAMP columns

# OPTIMIZED PROJECTIONS

## OPTIMIZING CLUSTERING



- Challenge
  - How to optimize queries for different access paths
- Solution: multiple materialized views
  - Load data to central table
  - Create a materialized view for each use case/access path
  - Cluster materialized views for each specific use case

# LAB EXERCISE: 17

## Materialized View Use Cases

20 minutes

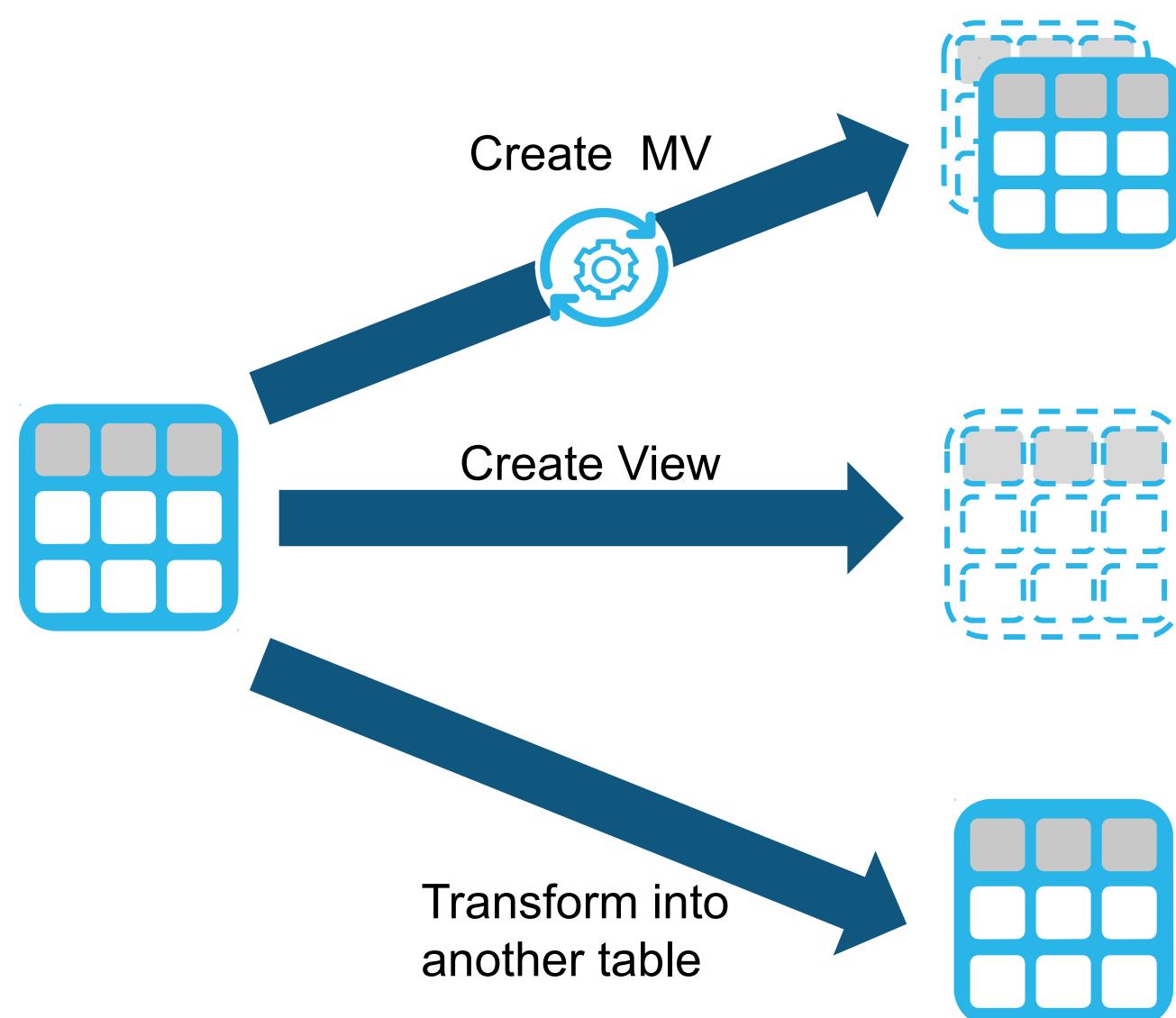
- Cluster a Table Using a Timestamp Column Type
- Cluster a Table to Improve Performance
- Automatic Transparent Rewrite on Materialized Views
- Materialized Views on External Tables



# WHEN TO USE MATERIALIZED VIEWS



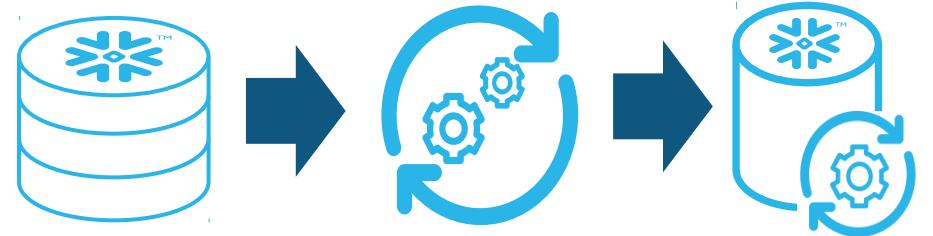
# MULTIPLE SOLUTIONS TO THE SAME PROBLEM



1. Create a materialized view
2. Create a standard view
3. Scheduled transformation process

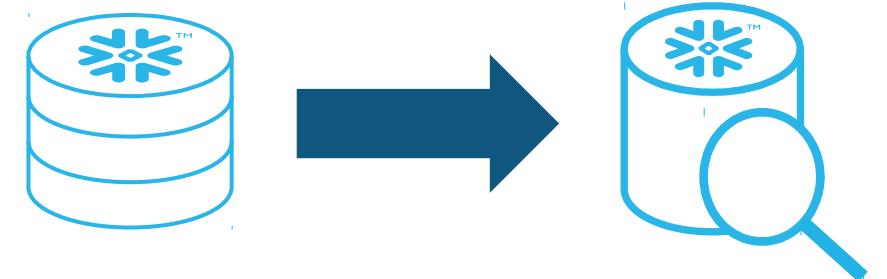
# MATERIALIZED VIEW

- **Query Performance:** Optimized (fast) for Use-Case
- **Storage:** The MView stores results to maximize query performance
- **Consistent:** Queries against MView & Table always consistent
- **Data Transformed:** Once only – by MView
- **No Warehouse:** As transformation in background process
- **Simple Transformation:** Only one SQL statement. Joins not supported
- **Query Rewrite:** Automatic by optimizer



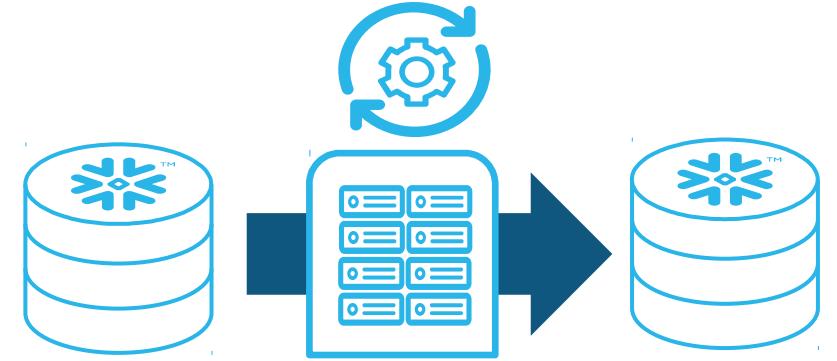
# VIEW

- **Query Performance:** No improvement. View directly over data
- **Storage:** No additional storage needed
- **Consistent:** Query executed at run-time
- **Data Transformed:** Multiple times – every time user queries against the view
- **No additional Warehouse:** Uses the end-user warehouse
- **Simple Transformation:** Only one SQL statement. But joins are supported
- **Query Rewrite:** None



# TRANSFORM PROCESS

- **Query Performance:** Optimized for Use-Case
- **Not Consistent:** Updates to target on scheduled basis
- **Data Transformed:** Once only – by ELT process
- **Virtual Warehouse:** Used to transform the data
- **Complex Transformation:** Multi-step procedures an option
- **Query Rewrite:** None





# BEST PRACTICES

- Balance:
  - Cost: MView maintenance and storage
  - Benefit: Query performance and potentially smaller query VWH
- Best For: Write Once – (resource intensive), Read Often (faster queries)
- Optimize cost:
  - Avoid MViews on rapidly changing tables
  - Batch Inserts, Updates, Deletes on base table
  - Cluster the MView. Not base table



# COMPARISON

Feature	Best for
<b>Cluster Key</b>	<ul style="list-style-type: none"><li>• Multi-terabyte tables</li><li>• Frequent queries on predefined columns (potential cluster key)</li><li>• Queries returning a <b>range</b> of keys (rather than unique)</li><li>• Queries IO bound – performing table scans</li></ul>
<b>Materialized View</b>	<ul style="list-style-type: none"><li>• Query aggregates or significantly reduced data set</li><li>• High frequency fast queries against semi-structured data</li><li>• Varying access paths on same table – clustered MViews</li></ul>
<b>Search Optimization</b>	<ul style="list-style-type: none"><li>• Equality lookups or small ranges</li><li>• Against <b>any</b> column – no consistent lookup key</li></ul>



# LAB EXERCISE

## Fill Out Course Survey

10 minutes

- Available:
  - In course materials at <https://training.snowflake.com>
  - Via email (sent this afternoon)
  - Direct link: <https://www.surveymonkey.com/r/QZJ6LRS>



# LAB EXERCISE: 18

## Performance Analysis Toolkit and Tuning Metrics

20 minutes

- Spilling to local storage and remote storage
- Explore a Large GROUP BY
- Explore WHERE clause without clustering
- Explore WHERE clause with clustering
- Rogue Query Syndrome from Join Explosion
- Tune timeout parameter to manage rogue queries
- Using Query tags to identify and track queries for monitoring





THANK YOU

