

Programming in Snowflake

Masterclass
2024 Hands-On!





Programming in Snowflake

- SQL & Data Analytics
- Snowflake Scripting
- Snowflake SQL REST API
- JSON Semi-Structured Data
- Stored Procedures
- User-Defined Functions
- User-Defined Table Functions
- SnowSQL & SnowCD
- Client Drivers (Python)
- Snowpark API & Data Frames
- Streamlit Web Applications
- Streamlit in Snowflake Apps
- Native App Framework
- Data Pipelines & Data Sharing
- Snowpipe and Snowpipe API
- Data Exchange & Marketplace



Best Ways to Benefit from this Course

- **Hands-On + Reviews (slides)**
- **Quizzes to Test Your Knowledge**
- **VSCode Project Setup + GitHub**
- **Captions + Playback Speed**
- **Q&A + Money Back Guarantee**
- **Review**

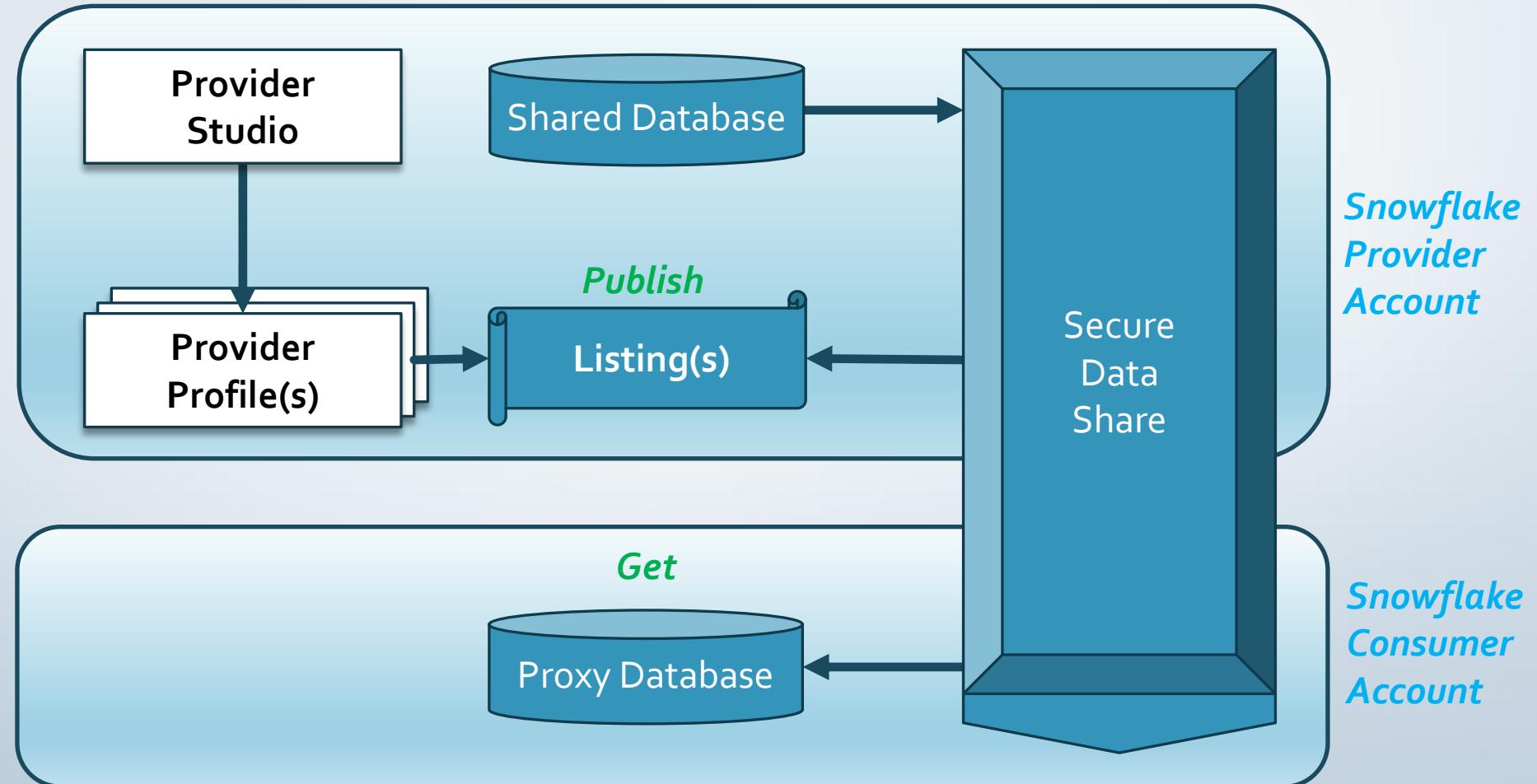


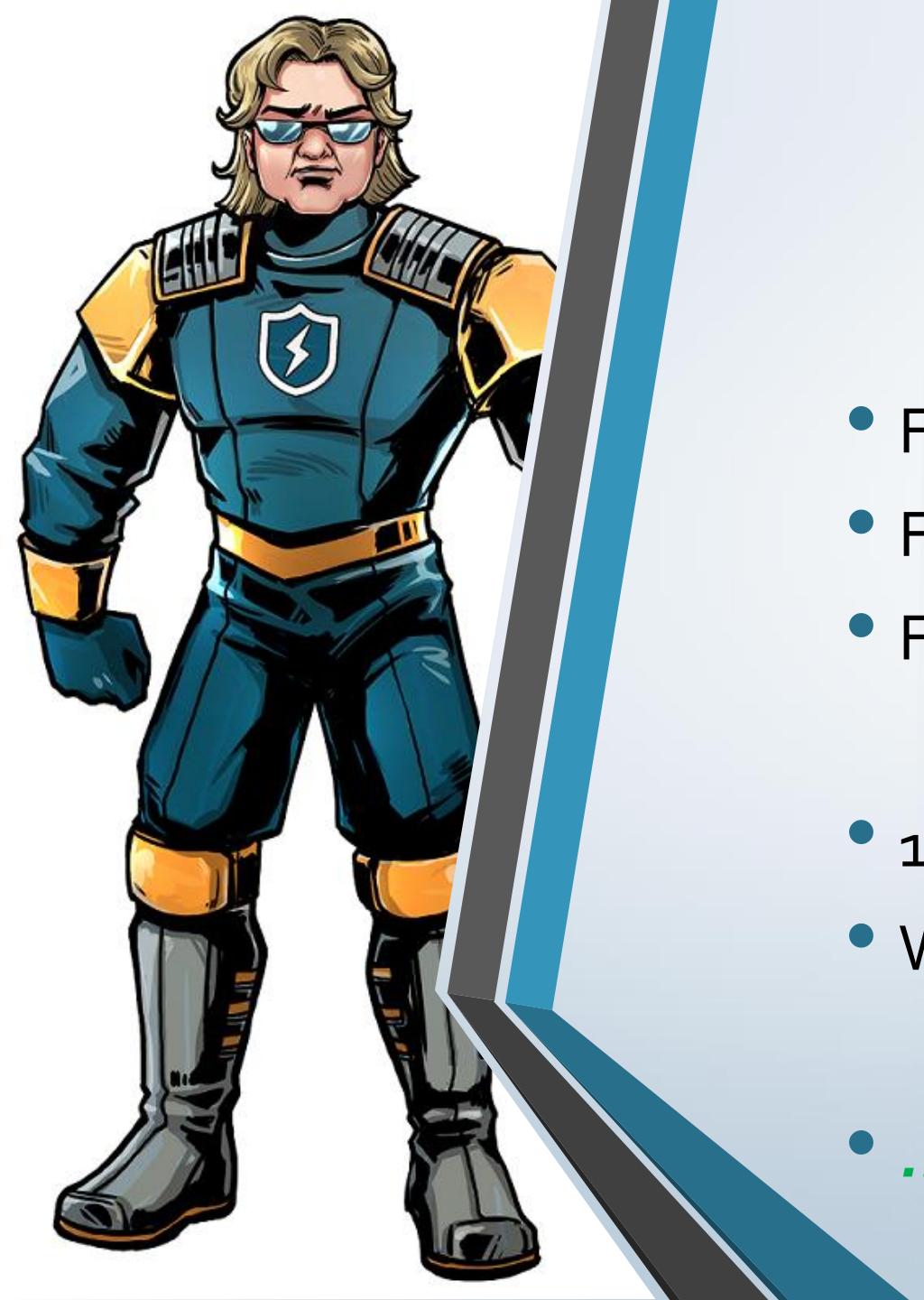
Course Structure

- ***Section Introduction***
 - Client Request
 - Review of Client Request
 - Section Summary

- ***Section Content***
 - Hands-On Programming Experiments
 - Review Checkpoint Slideshows
 - Check Your Knowledge Quiz

Architecture Diagram: Private Data Sharing





Credentials

- Former Snowflake Data Superhero
- Former SnowPro Certification SME
- Four Snowflake Certification Exams
- 100% Specialized in Snowflake
- World-Class Expert in Snowflake
- *...You Are in Good Hands!*



Real-Life Applications

Hierarchical Data Viewer

- with local CSV Python files
- as local/remote Streamlit web app
- as Python client to Snowflake
- as STREAMLIT in Snowflake (SiS) App
- as APPLICATION Native App for private Data Exchange

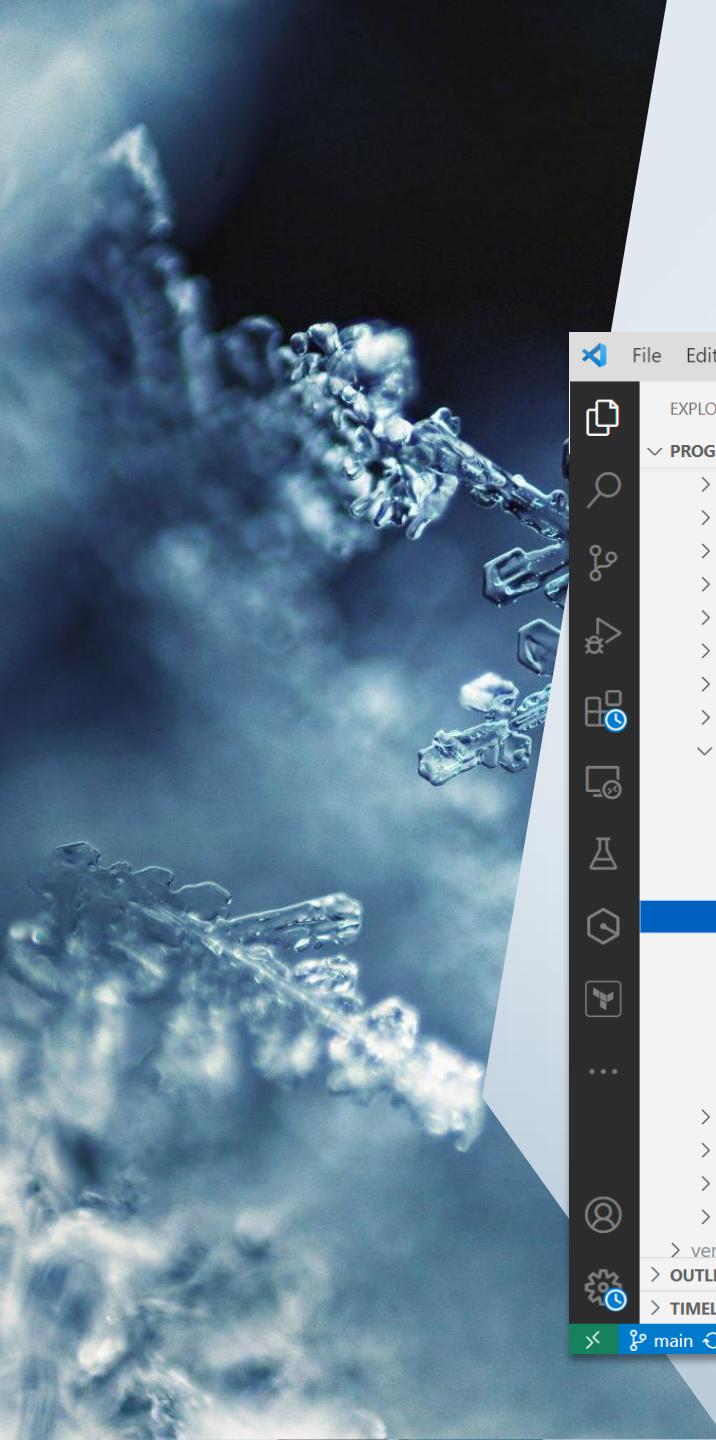
Hierarchical Metadata Viewer

- Data Lineage
- Object Dependencies
- Role Hierarchy
- STREAMLIT in Snowflake App

Enhanced Query Profile

- Query Analyzer
- STREAMLIT in Snowflake App

Real-Life Projects

A close-up photograph of several snowflakes against a dark background, serving as a decorative background for the slide.

A screenshot of a Microsoft Visual Studio Code (VS Code) interface. The title bar shows the search term "programming-in-snowflake". The left sidebar is the Explorer view, showing a project structure under "PROGRAMMING-IN-SNO...". The "app.py" file is selected in the Explorer, and its contents are displayed in the main editor area. The code is written in Python and uses Streamlit library to create a hierarchical data viewer application.

```
sections > 18-native-apps > app > app.py > ...  
1 import json  
2 import streamlit as st  
3 import modules.graphs as graphs  
4 import modules.formats as formats  
5 import modules.utils as utils  
6  
7 # setup page and create tabs  
8 st.set_page_config(layout="wide")  
9 st.title("Hierarchical Data Viewer")  
10 st.caption("Display your parent-child data pairs in a better manner.")  
11  
12 tabSource, tabHierarchy, tabFormat, tabGraph = st.tabs(  
13     ["Source", "Hierarchy", "Format", "Graph"])  
14  
15 # show source as data frame  
16 with tabSource:  
17     defTableName = "employees"  
18     tableName = st.text_input("Enter full name of a table or view:", value=defTableName)  
19     st.button("Refresh")  
20  
21     query = f"select * from {tableName}"  
22     df_orig = utils.getDataFrame(query)  
23     if df_orig is None: st.stop()  
24     st.dataframe(df_orig, use_container_width=True)  
25     cols = list(df_orig.columns)  
26  
27     child = st.sidebar.selectbox("Child Column Name", cols, index=0)  
28     parent = st.sidebar.selectbox("Parent Column Name", cols, index=1)  
29     df = df_orig[[child, parent]]  
30
```

At the bottom of the screen, there are several status icons and text: "main", "Sign in to Jira", "No active issue", "Bitbucket: Cristian Scutaru", "AWS", "CodeWhisperer", "Ln 11, Col 1", "Spaces: 4", "UTF-8", "CRLF", "Python 3.9.6 ('venv': venv)", and a small bell icon.

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His hands are visible, one holding the clipboard and the other holding a pen.

Client Request

We consider moving to Snowflake, would you have time for a quick demo?

We are particularly interested in its compute scalability, as we may need to run at some point a few heavy complex queries, on big data.

Why is Snowflake so interesting today, and what are some of the best practices we should be aware of?

Review of Client Request

We consider moving to Snowflake, would you have time for a quick demo?

We are particularly interested in its compute scalability, as we may need to run at some point a few heavy complex queries, on big data.

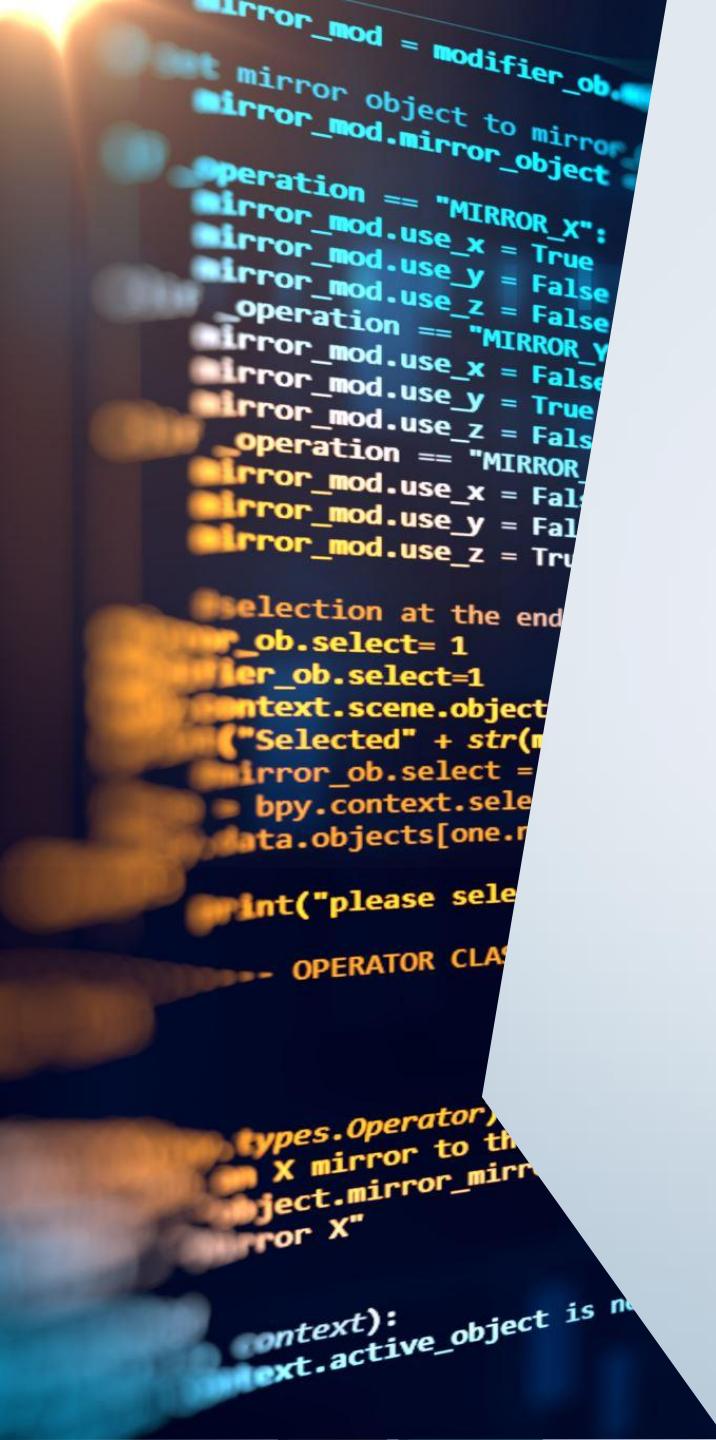
Why is Snowflake so interesting today, and what are some of the best practices we should be aware of?



Section Summary



- Sign-up for free trial account
- Snowsight (Snowflake's Web UI)
- Architecture and Virtual Warehouses
- Run Query with a Very Large Warehouse
- Resume a Large Multi-Cluster Warehouse
- Snowflake Editions & Pricing
- Best Practices for Compute and Storage



START YOUR 30-DAY FREE TRIAL

- Gain immediate access to the Data Cloud
- Enable your most critical data workloads
- Scale instantly, elastically, and near-infinitely across public clouds
- Snowflake is HIPAA, PCI DSS, SOC 1 and SOC 2 Type 2 compliant, and FedRAMP Authorized



Start your 30-day free Snowflake trial which includes \$400 worth of free usage

Cristian

Scutaru

your-email@company.com

XtractPro Software

Data Engineer

Canada

No, I do NOT want Snowflake to send me e-mails about products, services, and events that it thinks may interest me.

By clicking the button below you understand that Snowflake will process your personal information in accordance with its [Privacy Notice](#)

 **CONTINUE**

or sign in to an existing account

START YOUR 30-DAY FREE TRIAL

- Gain immediate access to the Data Cloud
- Enable your most critical data workloads
- Scale instantly, elastically, and near-infinitely across public clouds
- Snowflake is HIPAA, PCI DSS, SOC 1 and SOC 2 Type 2 compliant, and FedRAMP Authorized



Choose your Snowflake edition*

Standard

A strong balance between features, level of support, and cost.

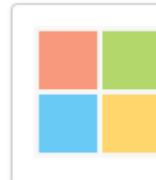
Enterprise

Standard plus 90-day time travel, multi-cluster warehouses, and materialized views.

Business Critical

Enterprise plus enhanced security, data protection, and database failover/fallback.

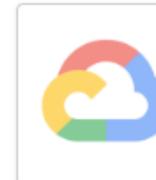
Choose your cloud provider*



Microsoft Azure



Amazon Web Services



Google Cloud Platform

US West (Oregon)

Check here to indicate that you have read and agree to the terms of the Snowflake Self Service On Demand Terms.

 **GET STARTED** 



Welcome to Snowflake!

Cristian Scutaru, please choose a username and password to get started

Username



Username can contain only letters and numbers.

Password



Your password must be 8 - 256 characters and contain at least 1 number(s), 0 special character(s), 1 uppercase and 1 lowercase letter(s).

Confirm password

Get started

Snowsight (Web UI)

CS Cristian Scutaru
ACCOUNTADMIN

Worksheets

Dashboards Streamlit Apps Data Marketplace Activity Admin Help & Support

Worksheets

Recent Shared with me My Worksheets Folders

TITLE	VIEWED ↓	UPDATED
Tutorial 1: ... Benc...	—	just now
Tutorial 2: ... Benc...	—	just now
Tutorial 3: ... Ben...	—	just now
Tutorial 4: ... Ben...	—	just now
Benchmarking Tu...	—	just now

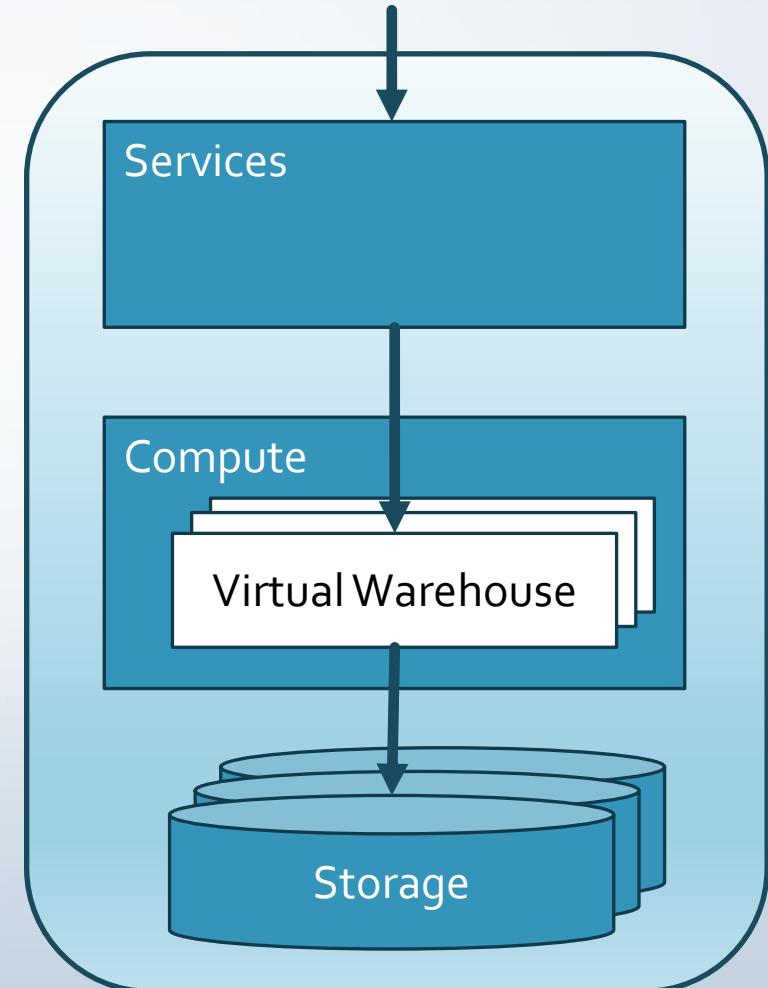
Classic Web UI (obsolete now)

The screenshot shows the Snowflake Classic Web UI interface. The top navigation bar includes icons for Databases, Shares, Marketplace, Warehouses, Worksheets, History, Account, Partner Connect, and Help. The 'Databases' icon is highlighted. Below the header, the title 'Databases' is displayed, along with a message 'Last refreshed 8:00:45 AM' and a refresh button. A toolbar provides options for 'Create...', 'Clone...', 'Drop...', and 'Transfer Ownership'. A search bar shows 'Search Databases' and indicates '44 databases'. The main content area is a table listing database details:

Database	Origin	Creation Time	Owner	Comment
STREAMLIT_QUERY_PROFILER		9/28/2023, 8:01 AM	ACCOUNTADMIN	
STREAMLIT_ERD_VIEWER		9/25/2023, 9:56 AM	ACCOUNTADMIN	
STREAMLIT_HIERARCHY_VIE...		9/24/2023, 1:15 PM	ACCOUNTADMIN	
Chinook		9/21/2023, 3:17 PM	ACCOUNTADMIN	
STREAMLIT_APPS		9/21/2023, 1:15 PM	ACCOUNTADMIN	
ERD_VIEWER		9/21/2023, 10:43 AM	ACCOUNTADMIN	
ERD_VIEWER_PACKAGE		9/21/2023, 10:41 AM	ACCOUNTADMIN	
HIERARCHY_VIEWER_PACKAGE		9/19/2023, 6:10 PM	ACCOUNTADMIN	
DECKTOOLS	SUNDECK.S2.DECK...	9/16/2023, 2:04 PM	ACCOUNTADMIN	

Snowflake Architecture (EPP)

- Cloud Data Warehouse only (no local version)
- Uses AWS, Azure or GCP infra
- All through SQL REST API
- Snowsight REST API for dashboards and worksheets
- Compute and Storage completely separated
- Snowflake Data Cloud



Snowflake

Services

- Authentication: basic, key pair, OAuth, SSO, MFA
- Infrastructure Management
- Metadata Management
- Query Parsing and Optimization
- Access Control: users, roles, privileges
- Serverless Tasks

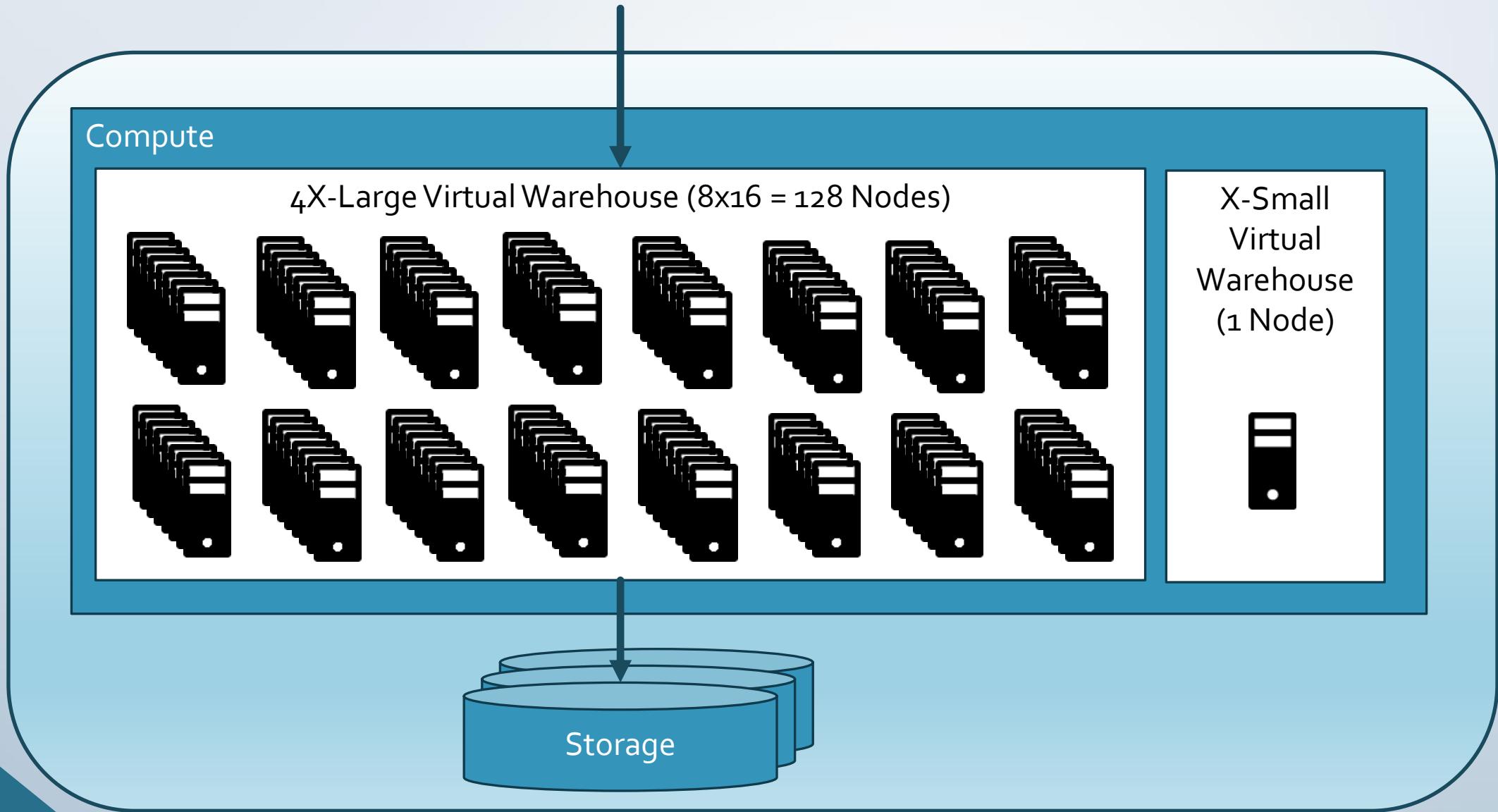
Compute

- Virtual Warehouses: ~your car engine
- Query Processing: single/multi-user, parallel processing
- Use bigger warehouse for a more complex query
- Use multi-cluster warehouses (Enterprise+ only) when multiple users
- Use query cache results when possible

Storage

- Back-End Data Storage: private to Snowflake
- Time Travel and Fail-safe storage
- Internal Stages: named, user, table stages
- Local and Remote Warehouse Storage
- Query Result Data Caches
- Data transfer in is free, but transfer out costs money!

Large Virtual Warehouse



Large Multi-Cluster Virtual Warehouse

Compute

3 Clusters x 3X-Large Virtual Warehouse (3 x 64 Nodes = 192 Nodes)



X-Small
Virtual
Warehouse
(1 Node)



Snowflake Editions & Pricing

STANDARD	ENTERPRISE	BUSINESS CRITICAL	VIRTUAL PRIVATE SNOWFLAKE (VPS)
 Complete SQL data warehouse Secure Data Sharing across regions / clouds Premier Support 24 x 365 1 day of time travel Always-on enterprise grade encryption in transit and at rest Customer-dedicated virtual warehouses Federated authentication Database replication External Functions Snowsight Create your own Data Exchange Data Marketplace access	 Standard + Multi-cluster warehouse Up to 90 days of time travel Annual rekeying of all encrypted data Materialized views Search Optimization Service Dynamic Data Masking External Data Tokenization	 Enterprise + HIPAA support PCI compliance Data encryption everywhere Tri-Secret Secure using customer-managed keys AWS PrivateLink support Database failover and failback for business continuity External Functions - AWS API Gateway Private Endpoints support	 Business Critical + Customer-dedicated virtual servers wherever the encryption key is in memory Customer-dedicated metadata store
\$2.00 cost per credit GET STARTED	\$3.00 cost per credit GET STARTED	\$4.00 cost per credit GET STARTED	CONTACT US
 ON-DEMAND STORAGE Pay for usage month to month.	\$40 per TB / per month LEARN MORE	 CAPACITY STORAGE Pay for usage up front. \$23 per TB / per month LEARN MORE	

Best Practices: Compute

- You are charged per VW up, not executed queries!
- Keep X-Small Warehouse
- Auto-Suspend after 1 minute (minimum!)
- Keep Standard Edition
- Economy mode
- Avoid querying too much the Account Usage schema

Best Practices: Storage

- Do not duplicate data, try zero-copy clone or data share
 - Do not store large amounts of data
 - No time travel or fail-safe unless necessary
-
- Use resource monitor to alert for over-spending
 - Use IP address
 - Switch from ACCOUNTADMIN

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His right hand holds a pen. The background behind him is a light grey.

Client Request

You convinced us, we'll move to Snowflake!

Here is a small CSV file with our employee test data, try to upload it into Snowflake.

We expect to expand in time to thousands of employees, each with thousands of related daily transactions, so some tables may reach in the order of TB.

Review of Client Request

You convinced us, we'll move to Snowflake!

Here is a small CSV file with our employee test data, try to upload it into Snowflake.

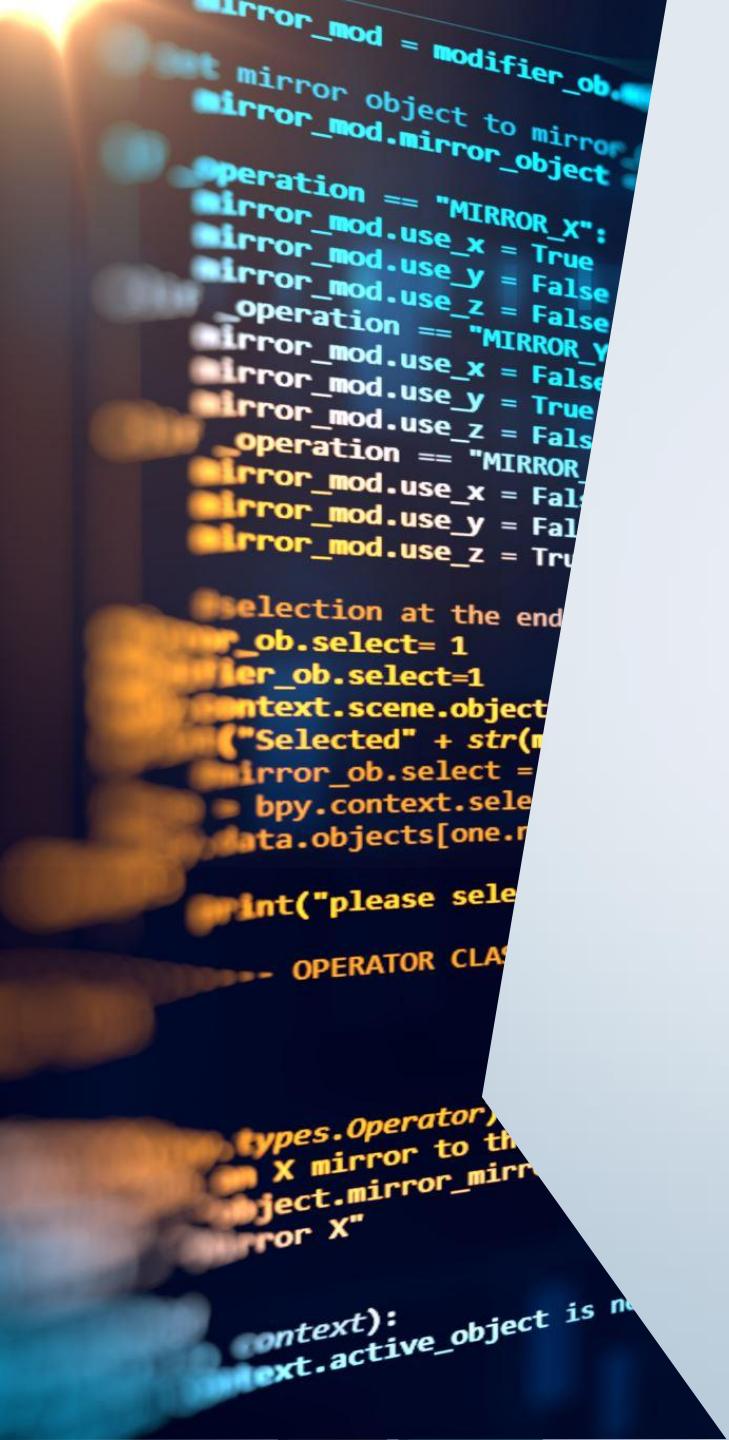
We expect to expand in time to thousands of employees, each with thousands of related daily transactions, so some tables may reach in the order of TB.



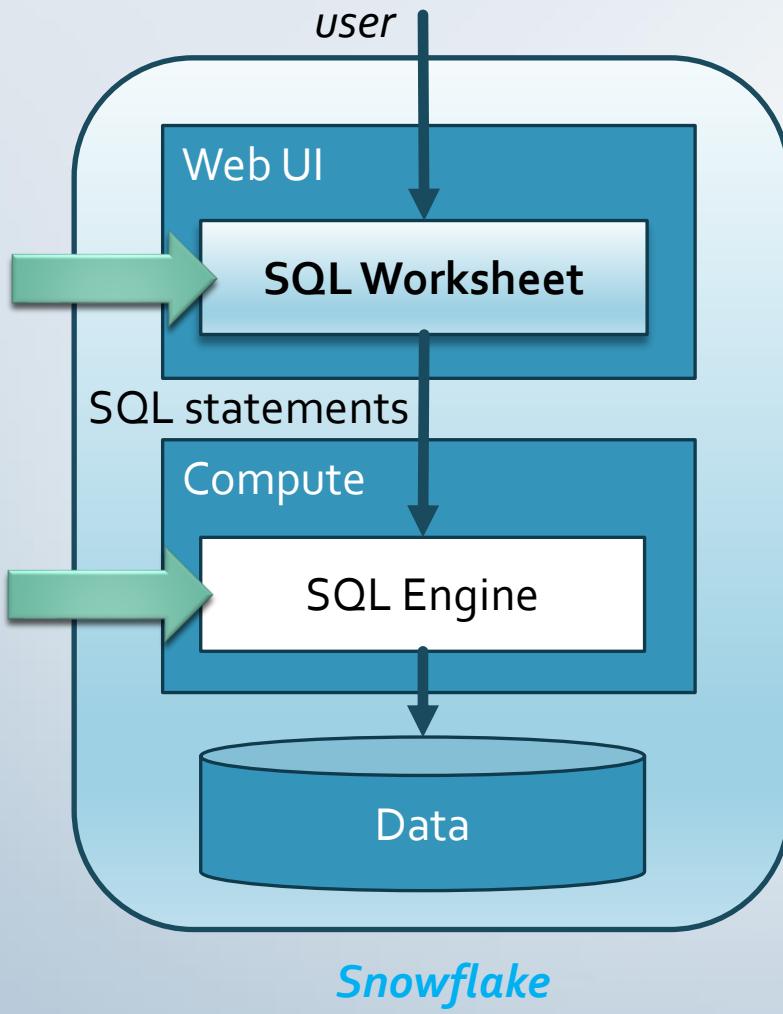
Section Summary



- Demo Project Setup
- SQL Worksheets
- Query Context and Identifiers
- Internal/External Stages
- Direct Access to Staged CSV File
- Schema Inference and DDL Script
- COPY File into Table



SQL Worksheets



Snowflake Worksheet interface showing a query execution:

Worksheet: Tutorial 1: Sample queries ...

Databases Worksheets

Search + ...

Benchmarking Tutorials

- Tutorial 1: Sample queries on TPC-H...
- Tutorial 2: Sample queries on TPC-D...
- Tutorial 3: TPC-DS 10TB Complete ...
- Tutorial 4: TPC-DS 100TB Complete ...

34 SELECT
35 l_returnflag,
36 l_linestatus,
37 sum(l_quantity) as sum_qty,
38 sum(l_extendedprice) as sum_base_price,
39 sum(l_extendedprice * (1-l_discount))

Results Chart

L_RETURNFLAG	L_LINESTATUS
A	F
N	F
N	O

Query Details

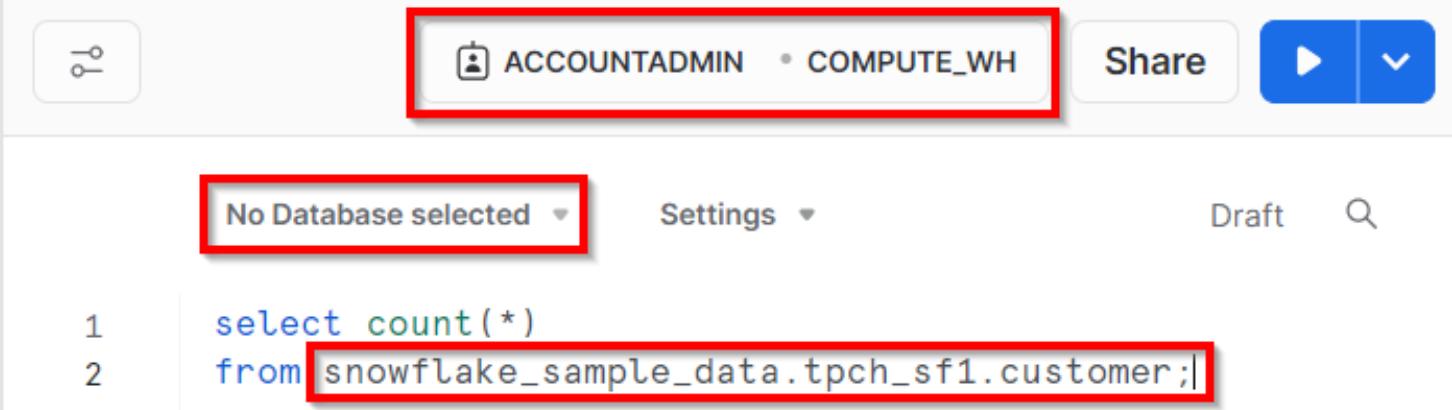
Query duration 2.0s

Rows 4

Query ID 01af9533-0001-cefd-0...

Query Context

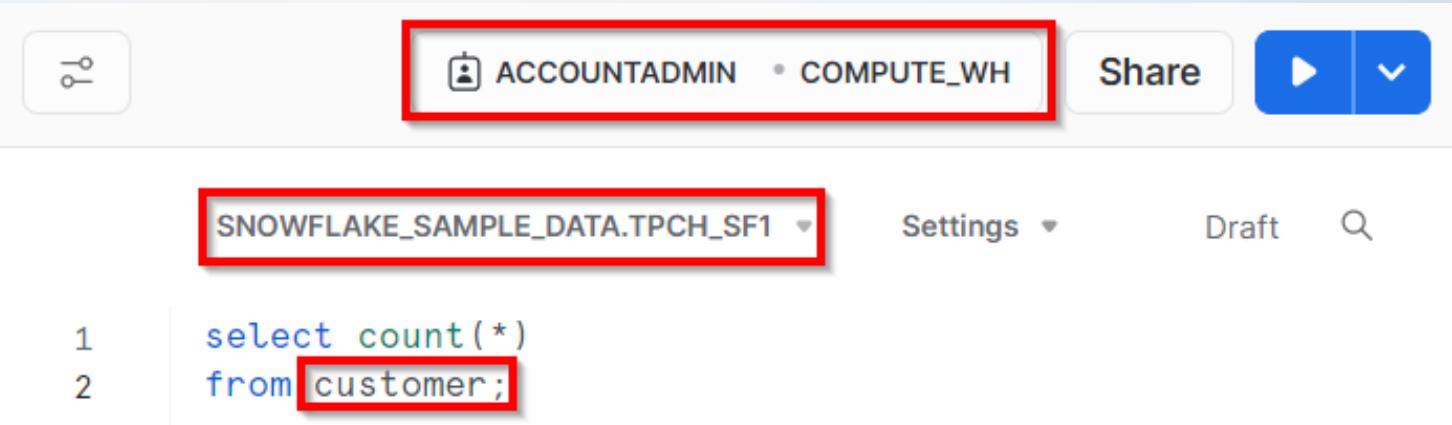
1. Role



A screenshot of a Snowflake query editor. At the top right, there is a user icon followed by the text "ACCOUNTADMIN · COMPUTE_WH". This entire box is highlighted with a red rectangle. To the right of this are buttons for "Share", "Run", and "Edit". Below this, there is a dropdown menu labeled "No Database selected" which is also highlighted with a red rectangle. To the right of the dropdown are "Settings", "Draft", and a search icon. The main area shows a query with two lines of code:

```
1 select count(*)  
2 from snowflake_sample_data.tpch_sf1.customer;
```

2. Warehouse



A screenshot of a Snowflake query editor. At the top right, there is a user icon followed by the text "ACCOUNTADMIN · COMPUTE_WH". This entire box is highlighted with a red rectangle. To the right of this are buttons for "Share", "Run", and "Edit". Below this, there is a dropdown menu labeled "SNOWFLAKE_SAMPLE_DATA.TPCH_SF1" which is highlighted with a red rectangle. To the right of the dropdown are "Settings", "Draft", and a search icon. The main area shows a query with two lines of code:

```
1 select count(*)  
2 from customer;
```

3. Database

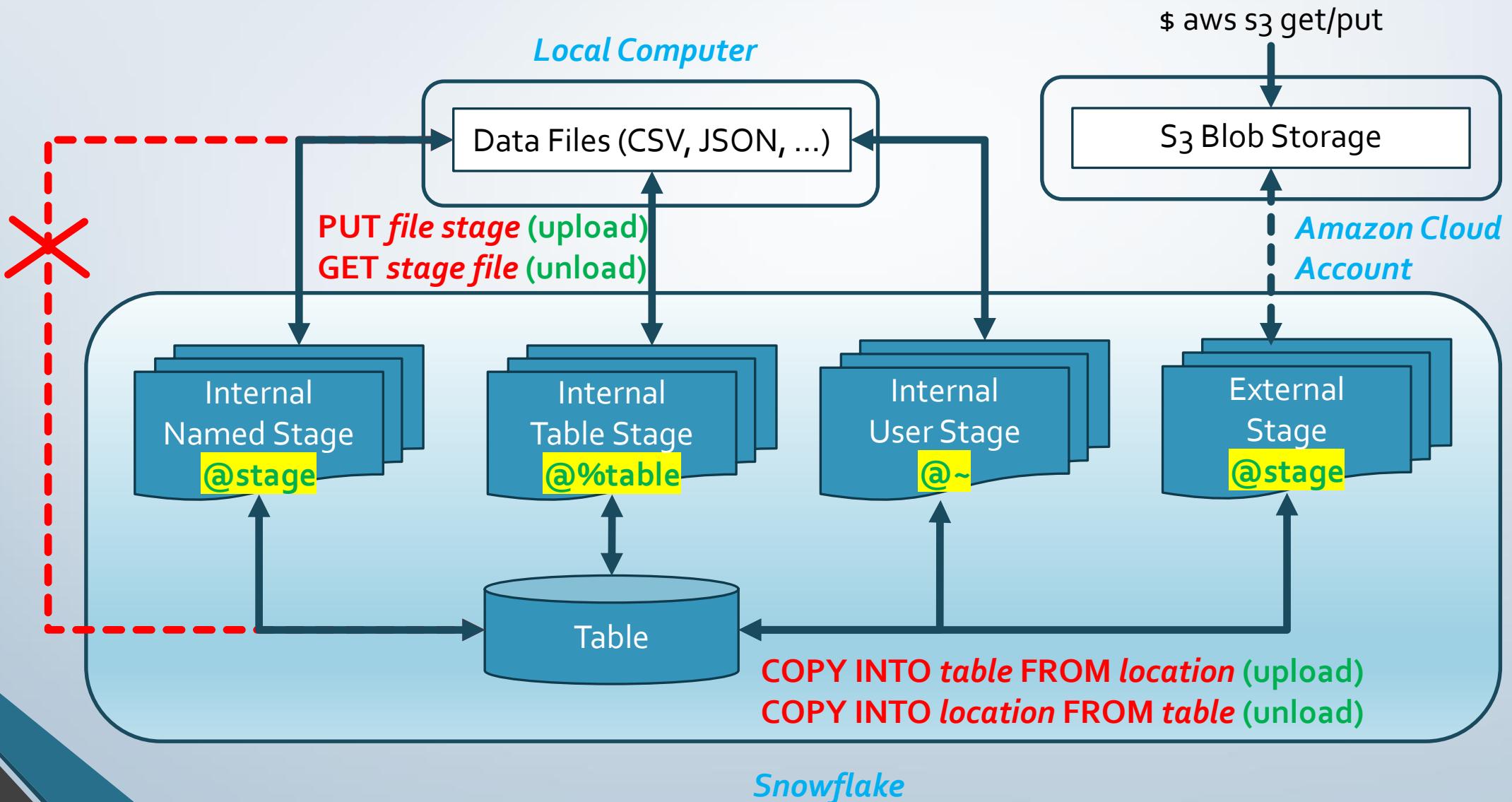
4. Schema

Query Context: Built-In Functions

- CURRENT_ROLE/WAREHOUSE/DATABASE/SCHEMA()
- CURRENT_USER/SESSION/STATEMENT/TRANSACTION()
- CURRENT_DATE/TIME/TIMESTAMP()
- CURRENT_ACCOUNT/CLIENT/VERSION/REGION/IP_ADDRESS()
- CURRENT_ACCOUNT/ORGANIZATION_NAME()
- CURRENT_AVAILABLE/SECONDARY_ROLES()

- IS_ROLE_IN_SESSION()
- LAST_TRANSACTION/QUERY_ID()

Stages: Uploading and Unloading Data



Stages: Examples

- `LIST @~;` ← list files from any stage
- `REMOVE ...` ← remove files from a stage

- `PUT file:///C:/data\data.csv @%my_table;` ← not from Snowsight!
- `COPY INTO my_table FROM @%my_table;`

- `CREATE TEMPORARY STAGE my_int_stage;`
- `PUT file:///C:/data\data.csv @my_int_stage;` ← not from Snowsight!
- `DESCRIBE STAGE my_int_stage;`

- `CREATE STAGE my_ext_stage URL='s3://load/files/';`
- `COPY INTO my_table FROM @my_ext_stage/data.csv;`
- `SHOW STAGES;`

Stages: External Stage in AWS S3

- Create a **S3 bucket**, in the same AWS region
 - Create a folder and upload some CSV files
- Create a **IAM policy**, to access this bucket
- Create a **IAM user**, w/ previous policy attached
 - Create access keys for the user → take note of both

```
CREATE STAGE mystage_s3
URL='s3://mybucket/spool/'
CREDENTIALS=
  AWS_KEY_ID='AKIAW6WN772VQZKDEOIY'
  AWS_SECRET_KEY='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') ;
LIST @mystage_s3;
```

name	size	md5	...	last_modified
s3://snowflake-demo-8888/spool/emp11.csv	232	51b76b37e922b51be6c140712c4cb38e		Mon, 6 Nov 2023 23:58:39 GMT
s3://snowflake-demo-8888/spool/emp12.csv	393	f75947c10fa974feb0ee86fedaf310ba		Mon, 6 Nov 2023 23:58:39 GMT

Schema Inference

- **INFER_SCHEMA**
 - **LOCATION** => '@mystage' ← internal/external named stage
 - **FILES** => 'emp.csv', ... ← 1+ uploaded files
 - **FILE_FORMAT** => 'myfmt' ← CREATE FILE FORMAT ... **PARSE_HEADER=TRUE**

```
-- show column definitions
SELECT *
FROM TABLE(INFER_SCHEMA(...))

-- generate create table DDL
SELECT GENERATE_COLUMN_DESCRIPTION(
    ARRAY_AGG(OBJECT_CONSTRUCT(*)), 'table') AS COLUMNS
FROM TABLE(INFER_SCHEMA(...));

-- create table directly from inferred schema
CREATE TABLE ... USING TEMPLATE(
    SELECT ARRAY_AGG(OBJECT_CONSTRUCT(*))
    FROM TABLE(INFER_SCHEMA(...))));
```

COPY Command

- **COPY INTO** table **FROM** stage ← upload data from staged file(s) into a table
 - **COPY INTO** stage **FROM** table ← unload data from a table into staged file(s)
-
- **LOCATION** => '@mystage' ← internal/external named stage
 - **FILES** => 'emp.csv', ... ← 1+ uploaded files
 - **FILE_FORMAT** => 'myfmt' ← CREATE FILE FORMAT ... PARSE_HEADER=TRUE
 - **PATTERN** => 'reg_exp'
 - **VALIDATION_MODE** => RETURN_n_ROWS | RETURN_ERRORS | RETURN_ALL_ERRORS

```
COPY INTO EMP FROM @mystage
  FILES = ('emp.csv')
  FILE_FORMAT = (FORMAT_NAME = mycsvformat)
  MATCH_BY_COLUMN_NAME = CASE_SENSITIVE
  FORCE = TRUE;
```

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His hands are visible, one holding the clipboard and the other holding a pen.

Client Request

What data file formats can we use with Snowflake?

We have another data file with our departments, but it is in JSON format. Please upload it in Snowflake, stored as tabular data.

Total salaries per department is something rarely updated, but frequently looked at.

Review of Client Request

What data file formats can we use with Snowflake?

We have another data file with our departments, but it is in JSON format. Please upload it in Snowflake, stored as tabular data.

Total salaries per department is something rarely updated, but frequently looked at.



Section Summary



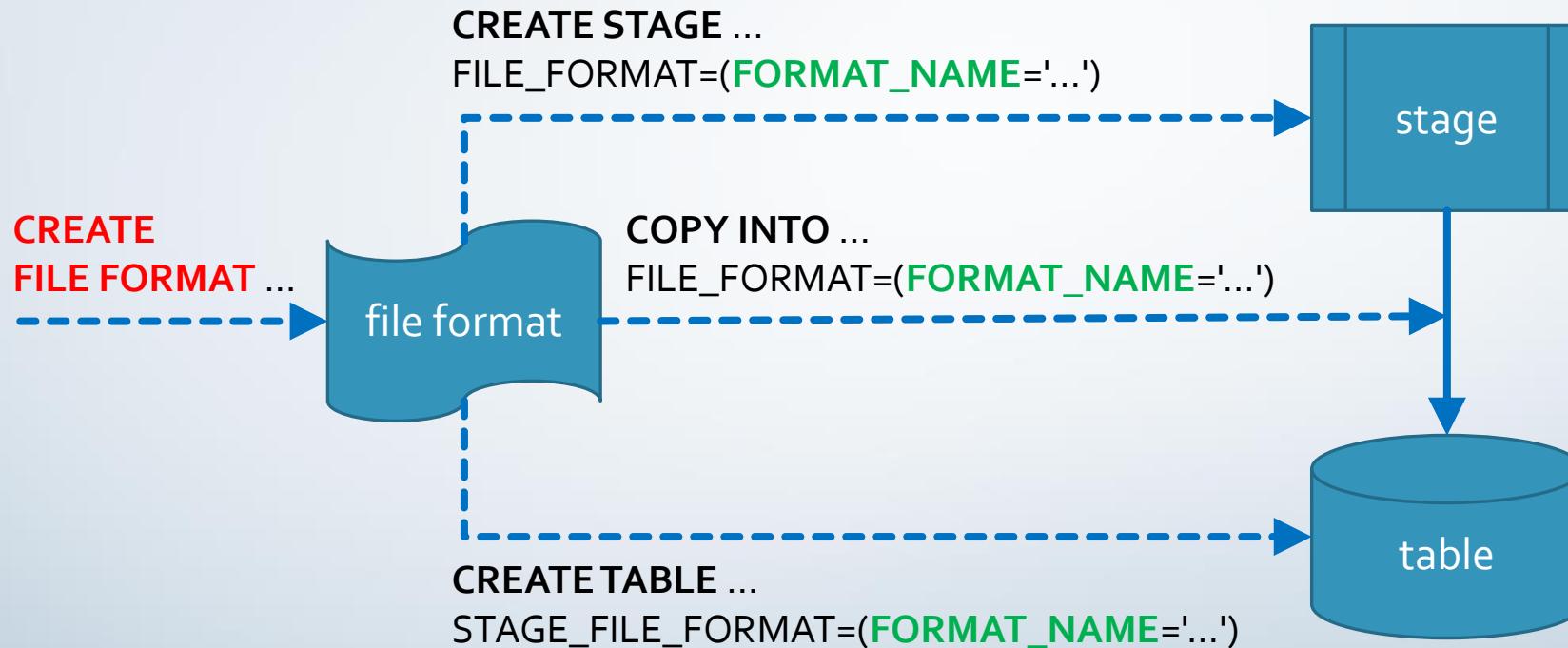
- File Formats: CSV, JSON, Parquet...
- Upload JSON Data into VARIANT
- JSON Objects and Arrays
- Flatten Arrays and Lateral Joins
- Primary and Foreign Key Constraints
- Temporary and Transient Tables
- Materialized Views



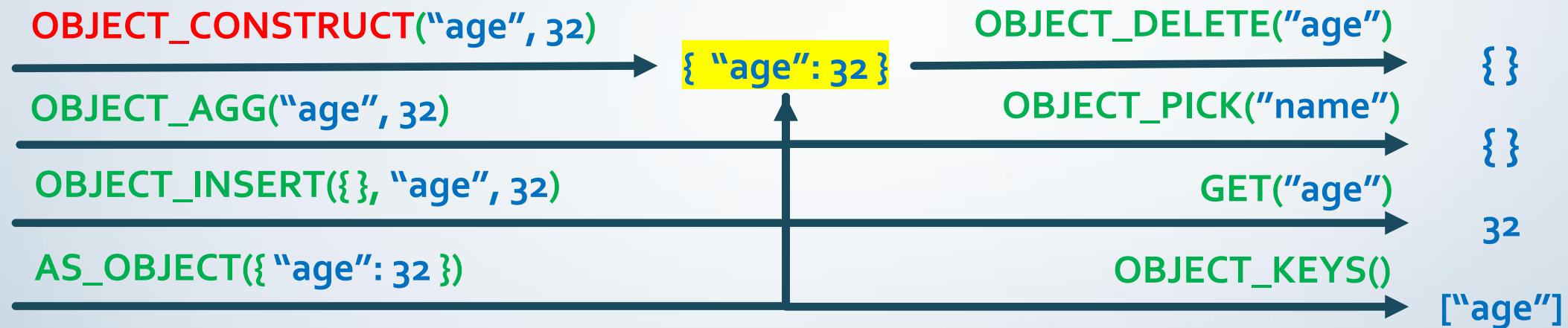
File Formats

- **CSV** (Comma-Separated Values) = tabular row-oriented (good for transactions). Includes any text-delimited (like tab-delimited values). First row may contain column names (but data types must be inferred).
- **JSON** (JavaScript Object Notation) = for hierarchical semi-structured data, including ND JSON ("Newline Delimited JSON") for both loading/unloading.
- **XML** (Extensible Markup Language) = for hierarchical semi-structured data, only for loading. More verbose than JSON.
- **PARQUET** = tabular format, but binary compressed and column-oriented (good for analytics). Used for Hadoop (Cloudera and Twitter collab).
- **ORC** (Optimized Row Columnar) = only for loading. Tabular format, but column-oriented (good for analytics). Used for Hadoop (Hortonworks and Facebook collab).
- **AVRO** = only for loading. Row-oriented (good for transactions), compressed binary format. Additional schema evolution metadata (in JSON), for RPC serialization. Used for Hadoop.

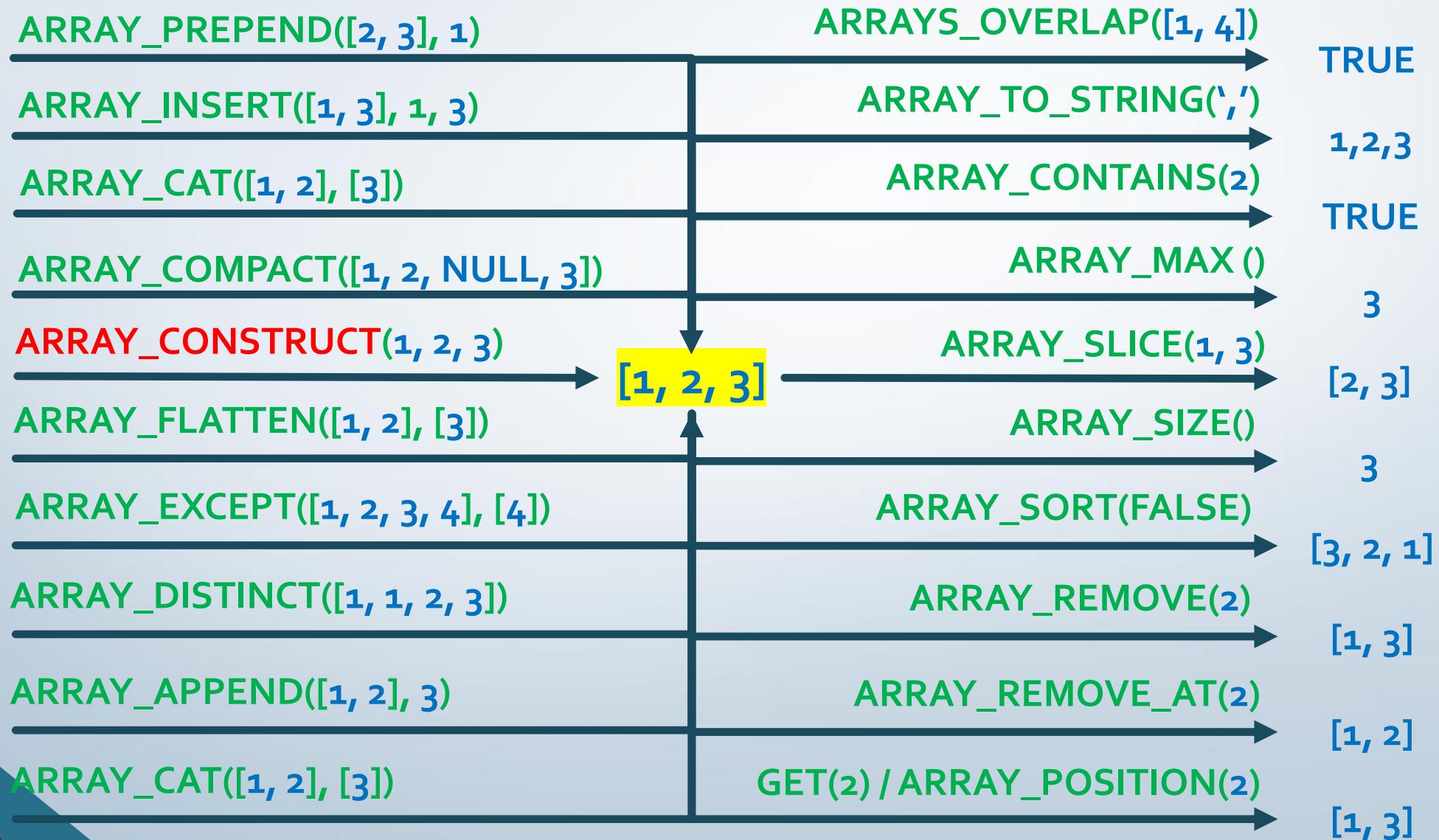
Named File Formats



JSON Objects Cheat Sheet



JSON Arrays Cheat Sheet



Flattening Arrays: JSON to VARIANT

```
create or replace table json(name string, v variant) as
  select 'John', parse_json($${
    "managers": [
      { "name": "Bill", "years": [2021, 2022] },
      { "name": "Linda", "years": [2020] }
    ]}$$)
union
  select 'Mary', parse_json($${
    "managers": [
      { "name": "Bill", "years": [2022, 2023] }
    ]}$$);

select *
  from json;
```

	NAME	V	...
1	John	{ "managers": [{ "name": "Bill", "years": [2021, 2022] }, { "name": "Linda", "years": [2020] }] }	
2	Mary	{ "managers": [{ "name": "Bill", "years": [2022, 2023] }] }	

Flattening Arrays: One Level

```
select j.name,
       m.value, m.value:name::string, m.value:years
  from json j,
       table(flatten(input => j.v, path => 'managers')) m;
```

```
select j.name,
       m.value, m.value:name::string, m.value:years
  from json j,
       lateral flatten(input => j.v, path => 'managers') m;
```

	NAME	VALUE	M.VALUE:NAME::STRING	M.VALUE:YEARS
1	John	{ "name": "Bill", "years": [2021, 2022] }	Bill	[2021, 2022]
2	John	{ "name": "Linda", "years": [2020] }	Linda	[2020]
3	Mary	{ "name": "Bill", "years": [2022, 2023] }	Bill	[2022, 2023]

Flattening Arrays: Two Levels

```
select name,
       m.value, m.value:name::string, m.value:years,
       y.value
  from json j,
       lateral flatten(input => j.v, outer => TRUE, path => 'managers') m,
       lateral flatten(input => m.value, path => 'years') y;
```

	NAME	VALUE	M.VALUE:NAME::STRING	M.VALUE:YEARS	...	VALUE
1	John	{ "name": "Bill", "years": [2021, 2022] }	Bill	[2021, 2022]		2021
2	John	{ "name": "Bill", "years": [2021, 2022] }	Bill	[2021, 2022]		2022
3	John	{ "name": "Linda", "years": [2020] }	Linda	[2020]		2020
4	Mary	{ "name": "Bill", "years": [2022, 2023] }	Bill	[2022, 2023]		2022
5	Mary	{ "name": "Bill", "years": [2022, 2023] }	Bill	[2022, 2023]		2023

Table Functions

- **FLATTEN(...)** ← when flattening JSON array data
- **LATERAL** ← when joining on UDF/UDTF input args
- **RETURNS TABLE**(col1 type, ...) ← in a UDTF definition
 - **SELECT * FROM TABLE(f(...))** ← when calling a UDTF
- **RESULT_SCAN(...)** ← when getting data from cache

Temporary Tables

- **CREATE TEMPORARY TABLE ...**
- **With no Fail-safe period** (but they may have time travel – can UNDROP!).
- Only exist within a session, not visible to other users or sessions.
- Auto-purged completely when the session ends, with no possibility to recover.
- Use for non-permanent, transitory data: ETL data, private session-specific data.
- Can create with the same name, and their name takes precedence over other tables!
- See also other temporary objects (like temp stages).

Transient Tables

- CREATE **TRANSIENT TABLE** ...
- **With no Fail-safe period** (but they may have time travel – can UNDROP!).
- Persist until explicitly dropped, but available to all users with appropriate privileges.
- Use for transitory data that needs to be maintained beyond a session.
- Use for data that does not need the same level of protection and recovery.

Materialized Views: Use Cases

- **CREATE MATERIALIZED VIEW ...**
- Query results have a lower number of rows/columns.
- Query results contain results of aggregated data.
- Query results come from semi-structured data analysis.
- Query on external table needs improved performance.
- Base table does not change frequently.
- Query results do not change often.
- Results are used often.
- Queries consume a lot of resources.

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His right hand holds a pen, and his left hand rests on the clipboard. The background behind him is a plain, light color.

Client Request

We don't have customer data today, so we will need to either extract a sample from an existing similar test table, or generate some synthetic data from scratch.

The customer ID must be also generated automatically.

Review of Client Request

We don't have customer data today, so we will need to either extract a sample from an existing similar test table, or generate some synthetic data from scratch.

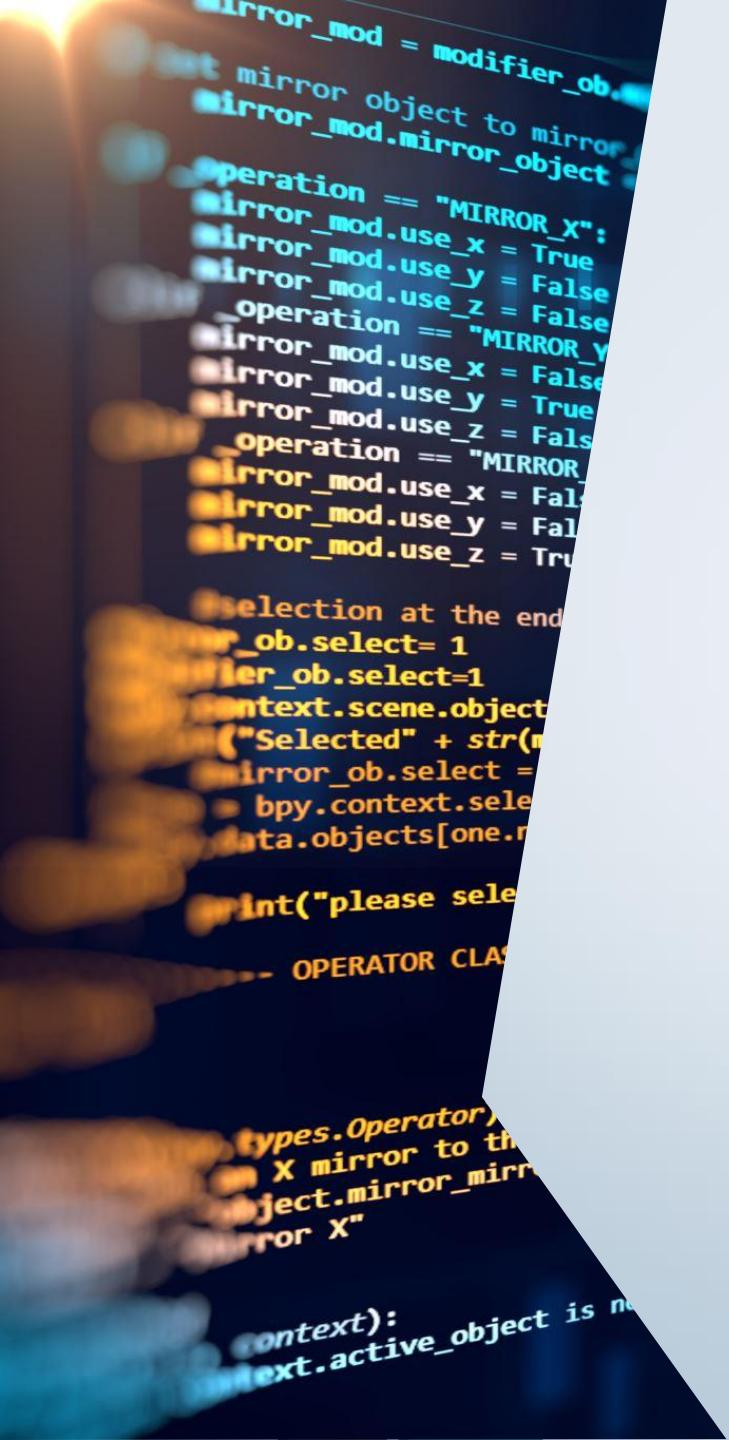
The customer ID must be also generated automatically.



Section Summary



- Snowflake Tutorials
- Snowflake Sample Databases
- Sample Data Extraction
- Synthetic Data Generation
- External Data Generation
- Sequences
- Identity Columns



Snowflake Tutorials

The screenshot shows the Snowflake Worksheets interface. On the left, a sidebar menu includes options like Worksheets (which is selected and highlighted in blue), Dashboards, Streamlit, Apps, Data, and Marketplace. The main area displays a list of worksheets under the heading "Benchmarking Tutorials". The list is sorted by "VIEWED" in descending order. The four visible worksheets are:

TITLE	VIEWED	UPDATED
Tutorial 4: TPC-DS 100TB Compl...	1 day ago	1 day ago
Tutorial 3: TPC-DS 10TB Complet...	1 day ago	1 day ago
Tutorial 2: Sample queries on TP...	1 day ago	1 day ago
Tutorial 1: Sample queries on TPC...	1 day ago	1 day ago

At the top right of the main area, there are "Search", "Share", and "+" buttons.

Snowflake Sample Databases: TPC-H

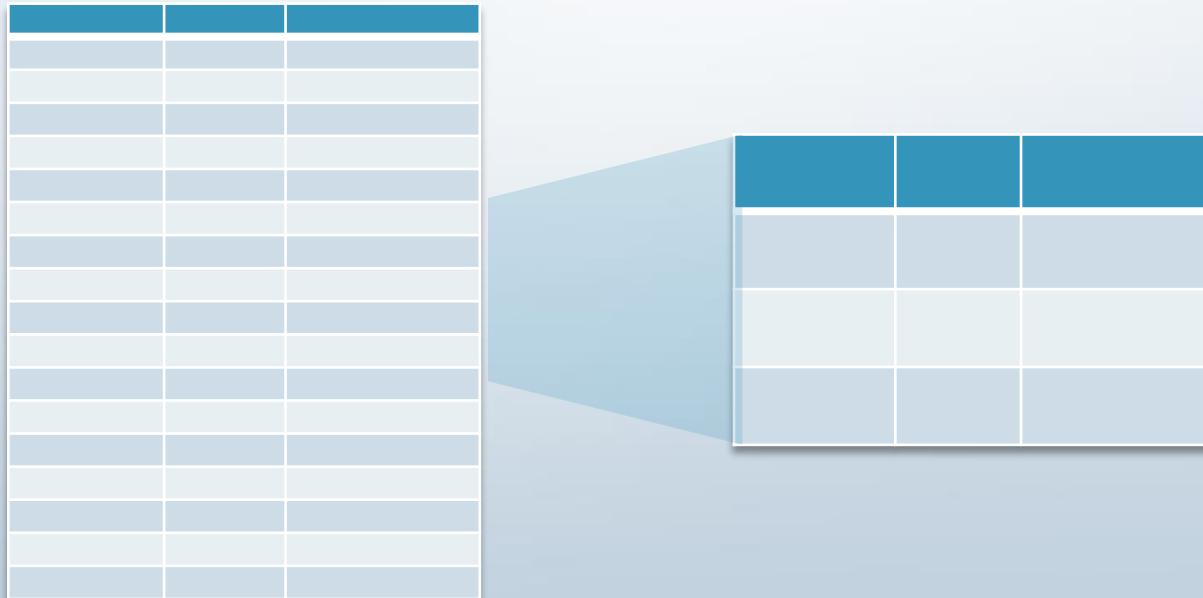
- TPCH_SF1 = ~M elements
 - *Tutorial 1: Sample queries on TPC-H data*
- TPCH_SF10 = ~10 x M elements
- TPCH_SF100 = ~100 x M elements
- TPCH_SF1000 = ~B elements

Snowflake Sample Databases: TPC-DS

- **TPCDS_SF10TCL** = 10 TB , 65M customers, 400K+ items
 - STORE_SALES table ~30B rows, fact tables 56+B rows
 - *Tutorial 2: Sample queries on TPC-DS data*
 - *Tutorial 3: TPC-DS 10TB Complete Query Test*
- **TPCDS_SF100TCL** = 100 TB, 100M customers, 500K+ items
 - STORE_SALES table ~300B rows, fact tables 560B rows
 - *Tutorial 4: TPC-DS 100TB Complete Query Test*

Sample Data Extraction

- FROM ... **SAMPLE (~TABLESAMPLE)** ← extracts subset of rows from a table
 - BERNOULLI (~ROW) | SYSTEM (~BLOCK) ← sampling method
 - <probability> | <n> ROWS ← sample size
 - REPEATABLE (~SEED) (<seed>) ← seed value (to make it deterministic)



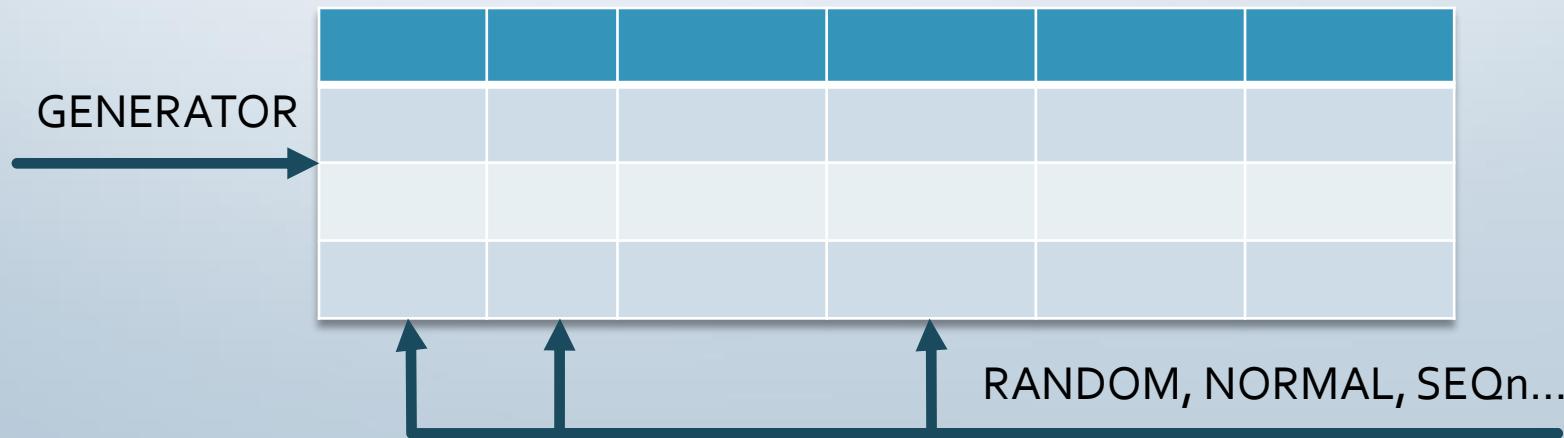
Sample Data Extraction Example

```
SELECT *
FROM SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL.CUSTOMER
SAMPLE (1000000 ROWS);
```

C_CUSTOMER_SK	C_CUSTOMER_ID	C_CURRENT_CDEMO_SK	C_CURRENT_HDEMO_SK	C_CURRENT_ADDR_SK	C_FIRST_SHIPTO_DATE_SK
12,543,226	AAAAAAAAKPEGPLAA	1,315,684	4,785	12,389,886	2,449,708
53,866,386	AAAAAAAACJPOFDDA	364,187	5,080	32,008,726	2,449,903
55,822,920	AAAAAAAIEKMDFDA	1,615,315	6,484	35,487,253	2,450,835
74,293,148	AAAAAAAAMJPJNGEA	32,097	2,892	36,934,840	2,449,313
47,168,904	AAAAAAAIINLPMCA	1,674,970	4,976	40,734,535	2,449,519
69,912,778	AAAAAAAAKMIMKCEA	1,687,963	3,767	45,816,818	2,450,115
16,511,430	AAAAAAAAGMBPLPAA	226,319	5,460	29,688,601	2,449,795
45,816,800	AAAAAAAAOLBLLCA	437,416	427	30,346,638	2,452,369

Synthetic Data Generation

- FROM TABLE(**GENERATOR**([rowcount], [timelimit])) ← rows (w/o columns)
- *Random* ← deterministic values
 - **RANDOM/RANDOMSTR** ← 64-bit integer/string with length
 - **UUID_STRING** ← UUID
- *Controlled Distribution* ← for unique ID values
 - **NORMAL/UNIFORM/ZIPF** ← number w/ specific distribution/integer
 - **SEQ1/SEQ2/SEQ4/SEQ8** ← sequence of integers



Data Generation Example

```
select
    randstr(uniform(10, 30, random(1)), uniform(1, 100000, random(1)))::varchar(30) as name,
    randstr(uniform(10, 30, random(2)), uniform(1, 10000, random(2)))::varchar(30) as city,
    randstr(10, uniform(1, 100000, random(3)))::varchar(10) as license_plate,
    randstr(uniform(10, 30, random(4)), uniform(1, 200000, random(4)))::varchar(30) as email
from table(generator(rowcount => 1000));
```

NAME	CITY	LICENSE_PLATE	...	EMAIL
Nqlz1q0R9B3C1770NMjrE0FKz	t5ySnovh41DWTs	gvNUTEICJj		rmlhEeUKqWSVWIDrE9Fv
dvcipiwdyUpY9	yAUhsuA9yosghO	2WsRzPBD4Z		JVHsrxUSzcEiO3cM8t9Pi
OMJ4G3dMbh	itHEOENN2G2K	pzQFwo470k		uC3dCuub0GF8N
BbbB99jp0P	illF8vrpDnXfIC2ADCLwlJpQk	b48F4MIHd9		J8gCKXavZlrkeoO
QWMdxYgtNYcQDVbcOAPuH	Hd8NZBuD1FMmmslfrvnmScHw6aq	bObgiO2WPn		KlxsgNrHRaE
vw8mHkaJNJixfp1srZ	JTjfrW0nxvQn66gG	BdHxzcZDr5		ek8coTvP3b4o5d12nanS

Faker Data Generation Example

Python Code

```
fake = Faker()
output = [
    "name": fake.name(),
    "address": fake.address(),
    "city": fake.city(),
    "state": fake.state(),
    "email": fake.email()
} for _ in range(1000)]
df = pd.DataFrame(output)
print(df)
```

	name	address	city	state	email
0	Charles Williams	41661 Mary Trail Suite 406\nLewisshire, VI 44371	Stacyport	Wyoming	stevensonryan@example.net
1	John Taylor	2455 Holmes Radial\nMorganville, UT 75583	South Jasonbury	New York	spencermary@example.net
2	Susan Madden	1857 Thomas Village\nWilliamsview, PW 45807	Christophertown	Delaware	luissawyer@example.net
3	Crystal Peterson	3418 Williams Springs Apt. 331\nEast Brittany,...	Turnerstad	Utah	woodardaaron@example.com
4	Gregory Hart	933 Smith Burg Suite 302\nNorth Lisaville, DC ...	North Donnastad	Maryland	kaufmancody@example.net
..
995	Lori Patton	139 Susan Causeway\nSouth Tammy, IL 89239	Dawnland	Nevada	gibbslindsay@example.org
996	Jose Harrison	3749 Rasmussen Island Suite 661\nPort Matthew,...	Jasmineton	West Virginia	fergusonterri@example.org
997	Melissa Stevens	2250 Hicks Plains\nSouth Ashleyside, NY 38746	Bradleyhaven	Florida	chandlermichael@example.net
998	Angela Pearson	20235 Brooke Dam\nEast Kimberly, MH 24215	Phillipsburgh	Montana	salinasbrandi@example.org
999	Mark Adams	9027 Ryan Corners Apt. 254\nNorth Juliaville, ...	Bennettberg	Utah	jeremy31@example.org

[1000 rows x 5 columns]

Unique Identifiers

- *Sequences*
- *Identity Columns*
- *UUIDs*

Sequences

- **CREATE SEQUENCE ...**
 - **START start INCREMENT incr** \leftarrow default (1, 1)
 - **ORDER | NOORDER** \leftarrow default ORDER (ASC)
-
- seq.*nextval* \leftarrow for next value (no seq.*currval*!)
 - TABLE(**GETNEXTVAL**(seq) \leftarrow ~seq.nextval

Identity Columns

- **CREATE TABLE ...**
- **AUTOINCREMENT | IDENTITY** ← auto-gen number (~seq number)
- **START start INCREMENT incr** ← alt. to (start, incr), def (1, 1)
- **ORDER | NOORDER** ← default ORDER (ASC)



Client Request

We need a better visual representation of the hierarchical employee-manager relationship. Please provide some solutions in plain SQL, if any.

Also, the name of the manager for an employee, and the names of employees directly supervised, will be some common functionalities used in queries.

Review of Client Request

We need a better visual representation of the hierarchical employee-manager relationship. Please provide some solutions in plain SQL, if any.

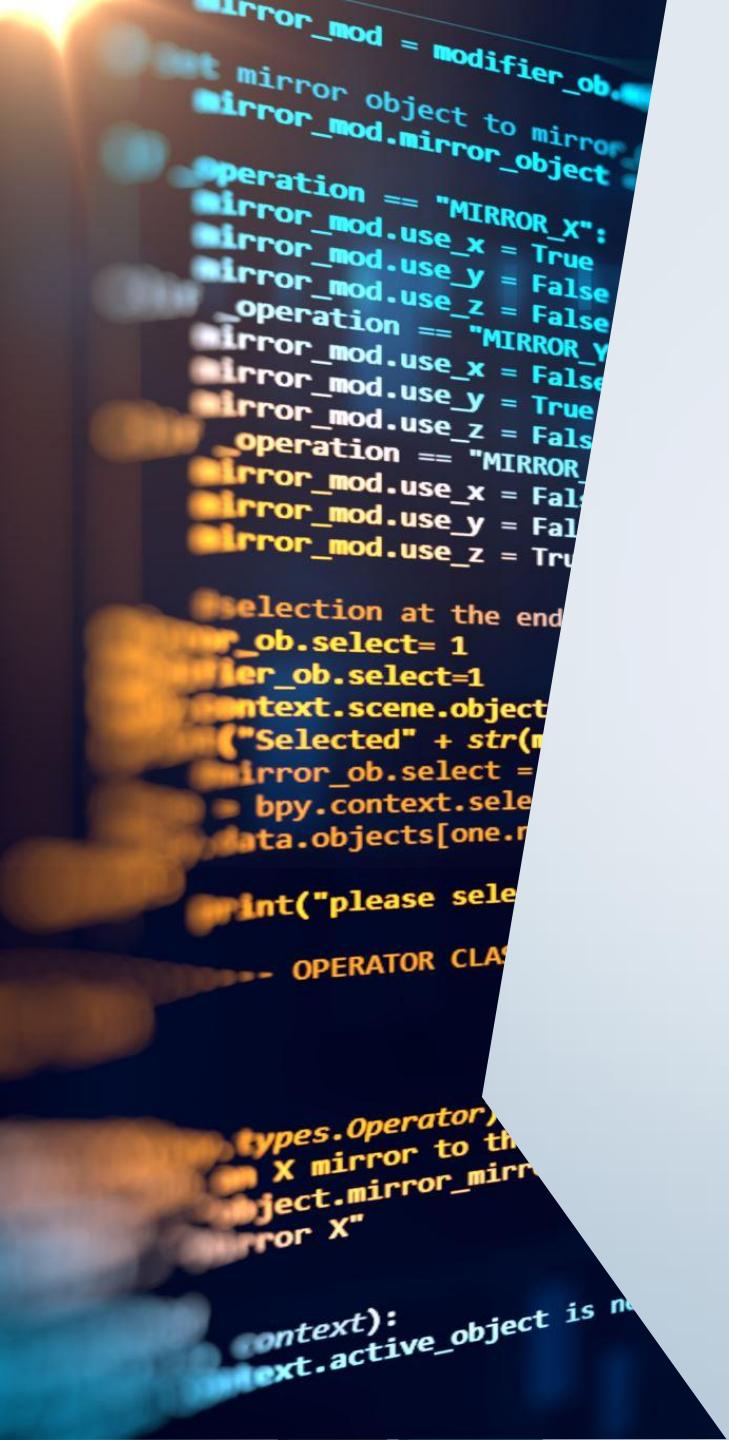
Also, the name of the manager for an employee, and the names of employees directly supervised, will be some common functionalities used in queries.



Section Summary



- Querying Hierarchical Data in SQL
- Views
- Stored Procedures
- JavaScript Stored Procedures API
- User-Defined Functions (UDFs)
- User-Defined Table Functions (UDTFs)



Hierarchical Data: SQL Queries

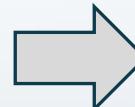
1. Multi-Level JOINs
2. CONNECT BY
3. Recursive CTEs
4. Recursive Views

child_parent view

EMPLOYEE	MANAGER
ADAMS	SCOTT
ALLEN	BLAKE
BLAKE	KING
CLARK	KING
FORD	JONES
JAMES	BLAKE
JONES	KING
KING	null
MARTIN	BLAKE
MILLER	CLARK
SCOTT	JONES
SMITH	FORD
TURNER	BLAKE
WARD	BLAKE

indented name + path

EMPLOYEE	MANAGERS
KING	→ KING
BLAKE	→ KING → BLAKE
ALLEN	→ KING → BLAKE → ALLEN
JAMES	→ KING → BLAKE → JAMES
MARTIN	→ KING → BLAKE → MARTIN
TURNER	→ KING → BLAKE → TURNER
WARD	→ KING → BLAKE → WARD
CLARK	→ KING → CLARK
MILLER	→ KING → CLARK → MILLER
JONES	→ KING → JONES
FORD	→ KING → JONES → FORD
SMITH	→ KING → JONES → FORD → SMITH
SCOTT	→ KING → JONES → SCOTT
ADAMS	→ KING → JONES → SCOTT → ADAMS



Hierarchical Data: (1) Multi-Level JOINS

```
select coalesce(m3.employee || '.', '')  
    || coalesce(m2.employee || '.', '')  
    || coalesce(m1.employee || '.', '')  
    || e.employee as path,  
    regexp_count(path, '\\\\.') as level,  
    repeat(' ', level) || e.employee as name  
from employee_manager e  
left join employee_manager m1 on e.manager = m1.employee  
left join employee_manager m2 on m1.manager = m2.employee  
left join employee_manager m3 on m2.manager = m3.employee  
order by path;
```

	PATH	LEVEL	NAME	...
1	KING	0	KING	
2	KING.BLAKE	1	BLAKE	
3	KING.BLAKE.ALLEN	2	ALLEN	
4	KING.BLAKE.JAMES	2	JAMES	
5	KING.BLAKE.MARTIN	2	MARTIN	
6	KING.BLAKE.TURNER	2	TURNER	
7	KING.BLAKE.WARD	2	WARD	
8	KING.CLARK	1	CLARK	

Hierarchical Data: (2) CONNECT BY

```
select repeat(' ', level-1) || employee as name,  
       ltrim(sys_connect_by_path(employee, '.'), '.') as path  
  from employee_manager  
 start with manager is null  
 connect by prior employee = manager  
 order by path;
```

NAME	PATH
KING	KING
BLAKE	KING.BLAKE
ALLEN	KING.BLAKE.ALLEN
JAMES	KING.BLAKE.JAMES
MARTIN	KING.BLAKE.MARTIN
TURNER	KING.BLAKE.TURNER
WARD	KING.BLAKE.WARD
CLARK	KING.CLARK
MILLER	KING.CLARK.MILLER
JONES	KING.JONES
FORD	KING.JONES.FORD

Hierarchical Data: (3) Recursive CTEs

```
with recursive cte (level, name, path, employee) as (
    select 1, employee, employee, employee
        from employee_manager
        where manager is null
    union all
    select m.level + 1,
        repeat(' ', level) || e.employee,
        path || '.' || e.employee,
        e.employee
        from employee_manager e join cte m on e.manager = m.employee)
select name, path
from cte
order by path;
```

Hierarchical Data: (4) Recursive Views

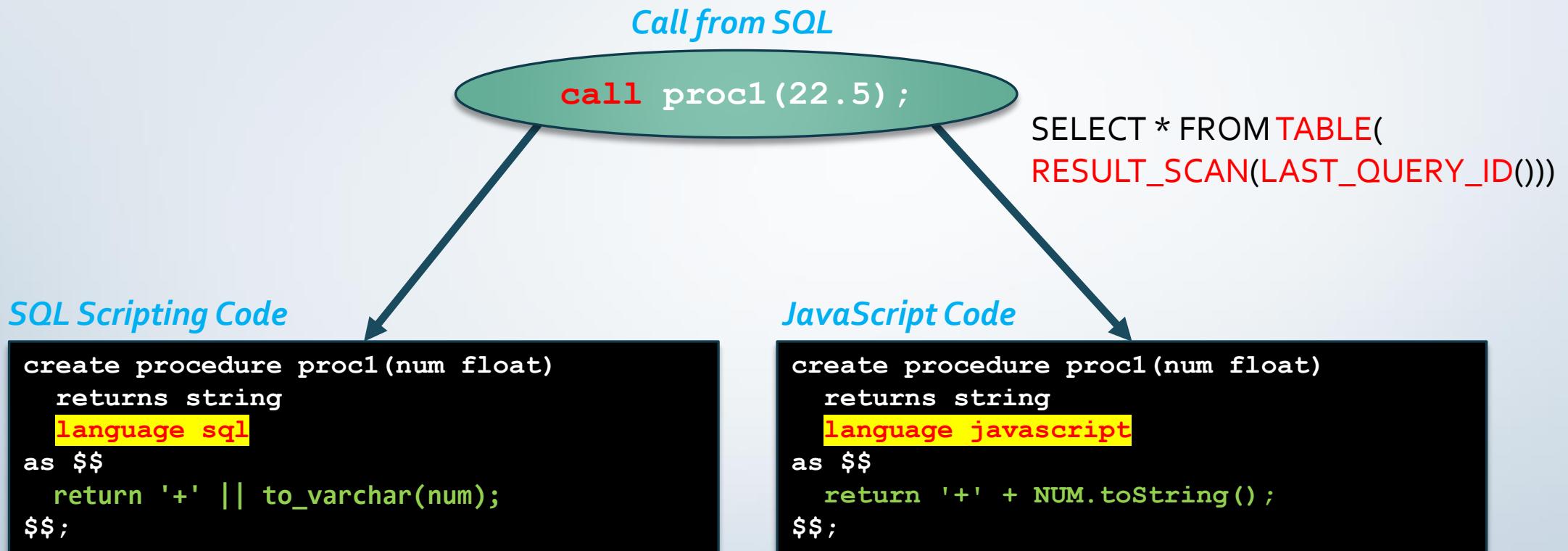
```
create recursive view employee_hierarchy (level, name, path, employee) as (
    select 1, employee, employee, employee
        from employee_manager
        where manager is null
    union all
    select m.level + 1,
        repeat(' ', level) || e.employee,
        path || '.' || e.employee,
        e.employee
        from employee_manager e join employee_hierarchy m on e.manager = m.employee);

select name, path
from employee_hierarchy
order by path;
```

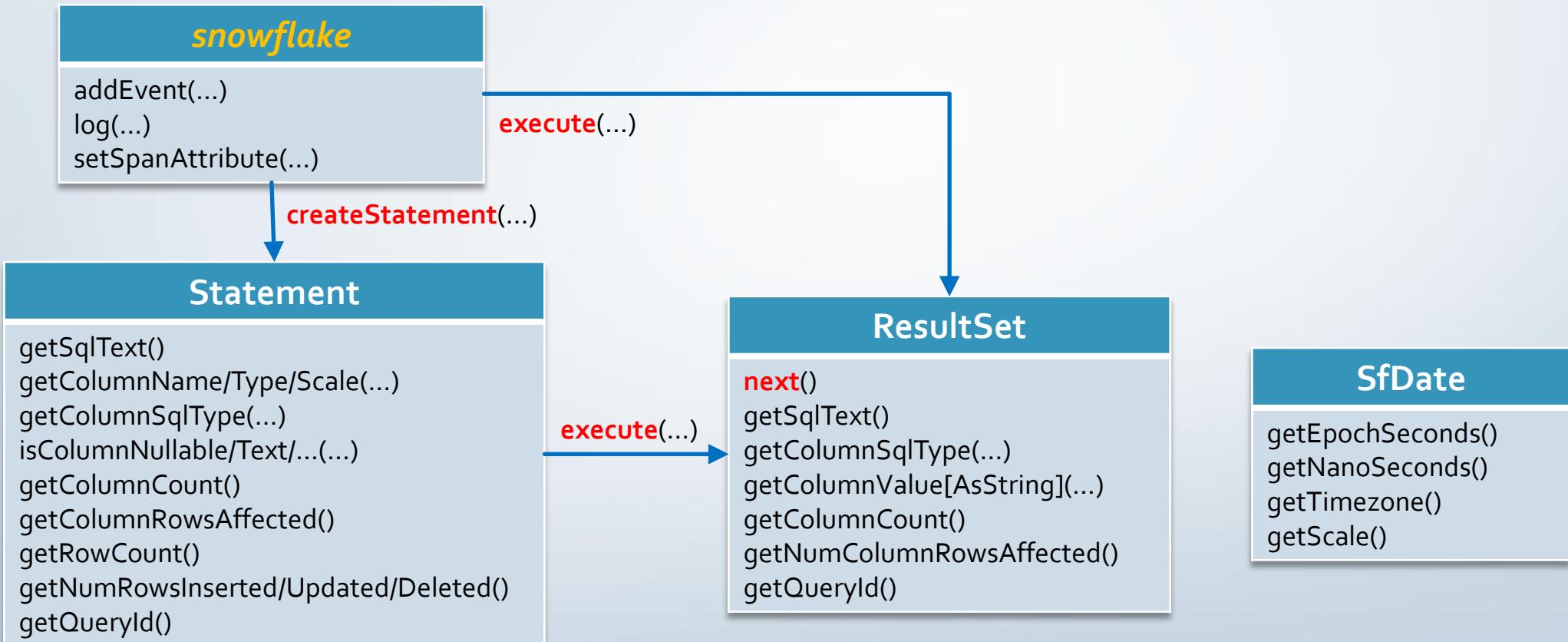
Stored Procedures and Functions

- CREATE PROCEDURE / CREATE FUNCTION
 - **RETURNS ... [NOT NULL]** ← data type (**TABLE** for UDTFs)
 - **EXECUTE AS OWNER/CALLER** ← access rights
 - **CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT**
- **LANGUAGE ...** ← **SQL/JavaScript** or **Python/Java/Scala** (w/ Snowpark)
- *for Python/Java/Scala*
 - **RUNTIME_VERSION = '...'** ← fixed
 - **HANDLER = '...'** ← internal class[+function]
 - **PACKAGES = ('...', ...)** ← imported packages
 - **IMPORTS = ('...', ...)** ← stage path and file names to read
 - **TARGET_PATH = '...'** ← stage path and JAR file name to write

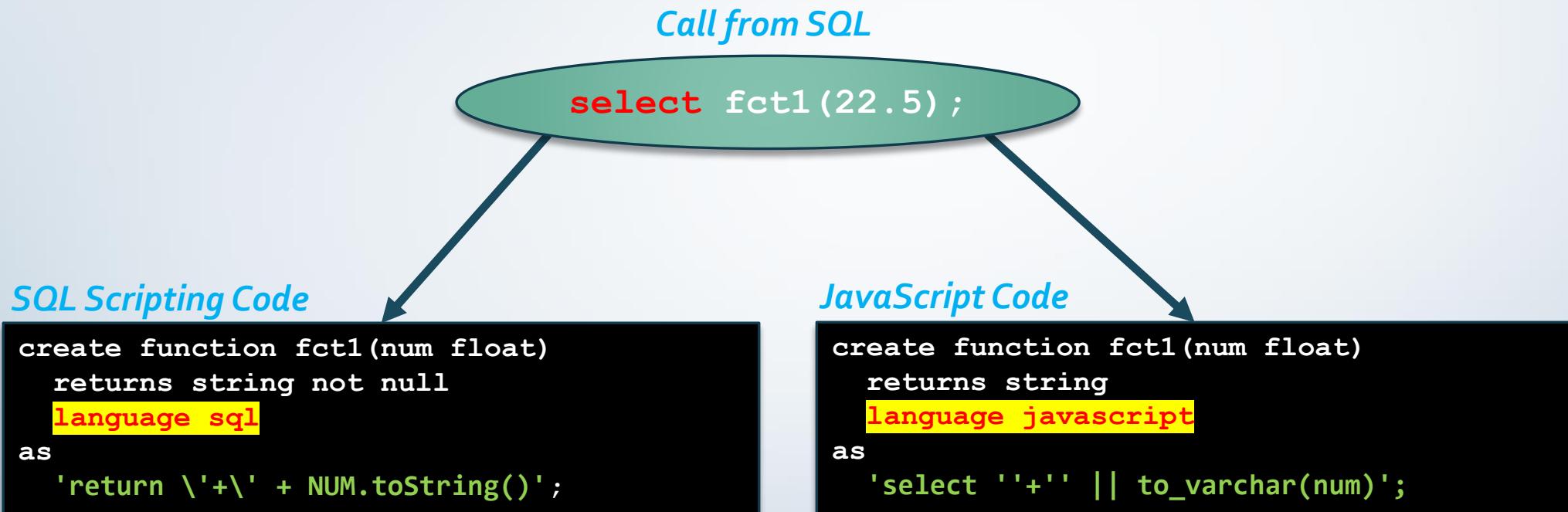
Stored Procedures



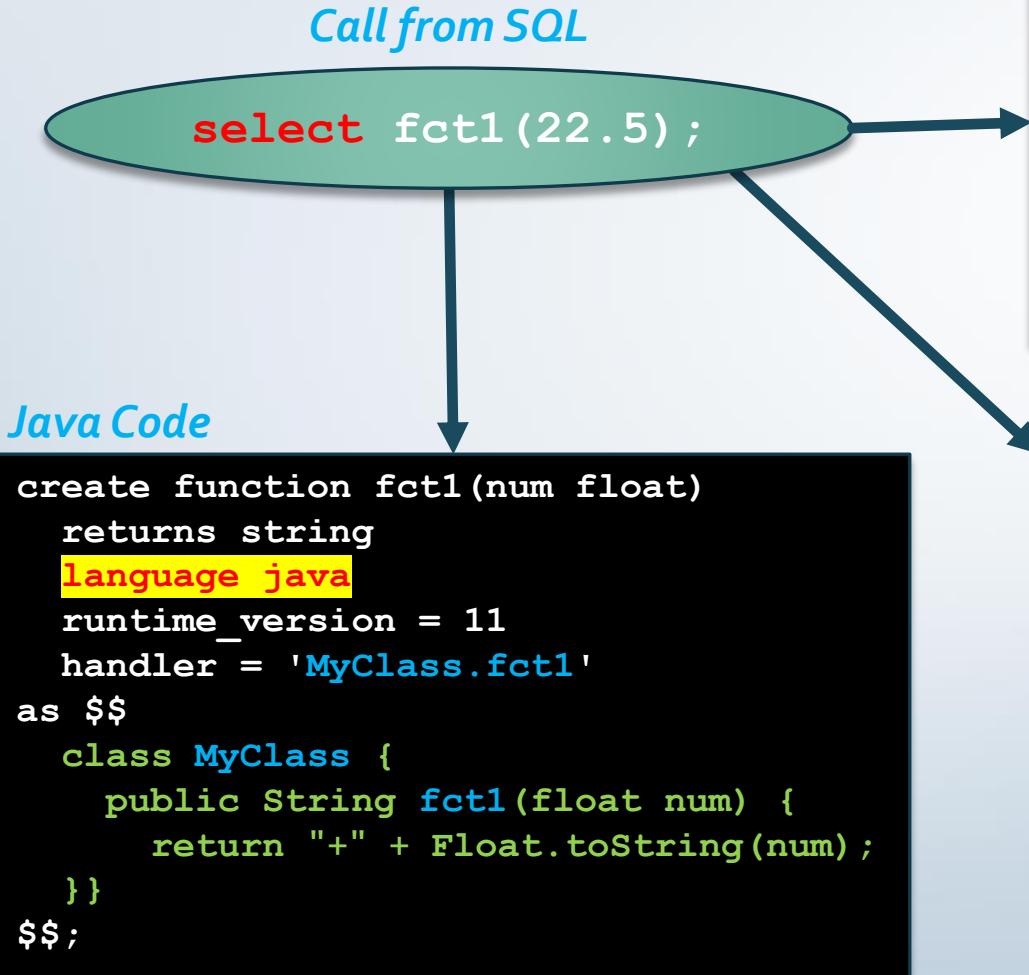
JavaScript Stored Procedures API



UDFs (User-Defined Functions)



UDFs (cont.)



Python Code

```
create function fct1(num float)
  returns string
language python
runtime_version = '3.8'
handler = 'proc1'
as $$
  def proc1(num: float):
    return '+' + str(num)
$$;
```

UDTFs (User-Defined Table Functions)

Call from SQL

```
select * from  
table(fcttl('abc'));
```

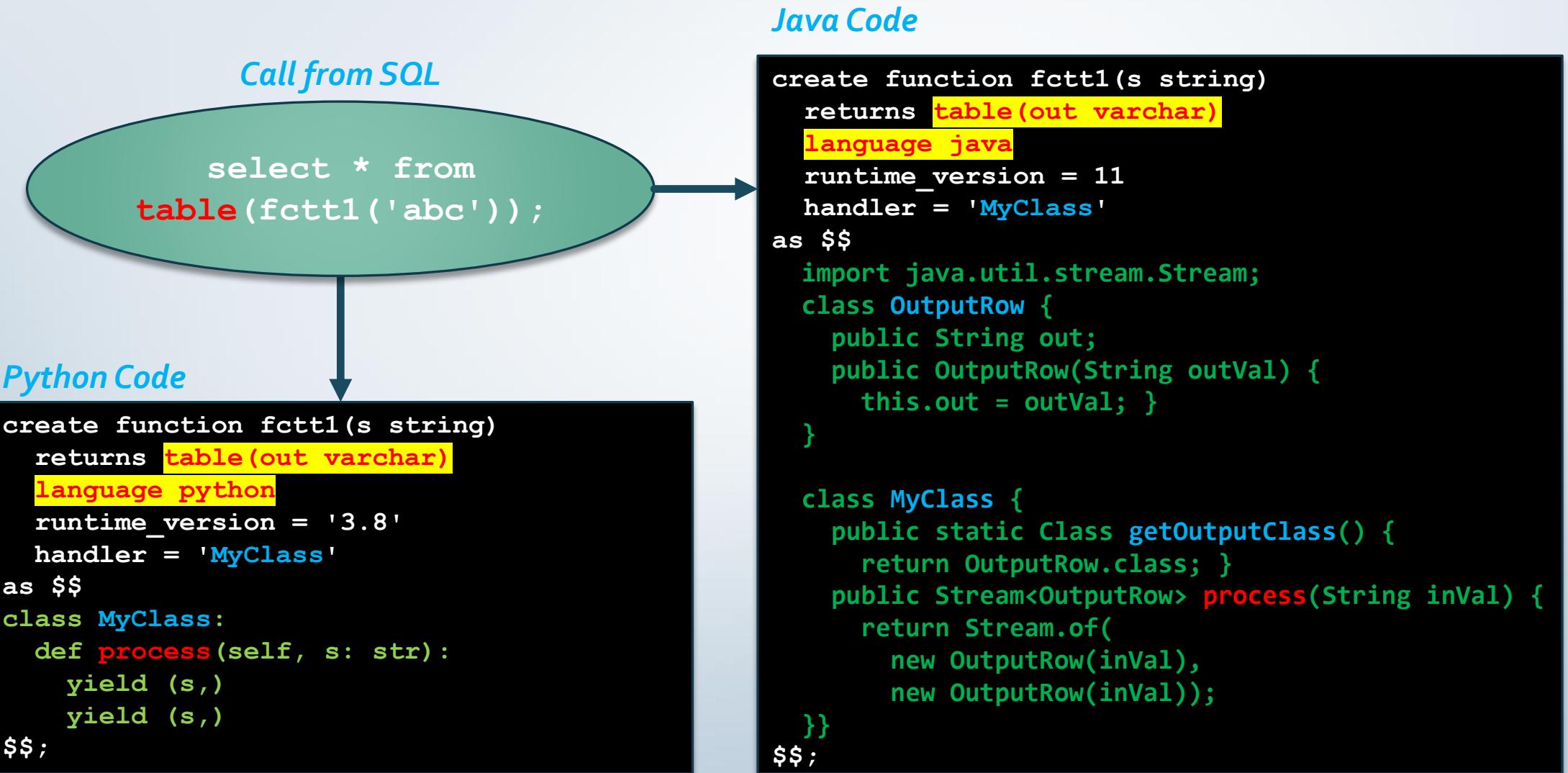
SQL Scripting Code

```
create function fcttl(s string)  
    returns table(out varchar)  
as  
begin  
    select s  
    union all  
    select s  
end;
```

JavaScript Code

```
create function fcttl(s string)  
    returns table(out varchar)  
language javascript  
strict  
as $$  
{  
    processRow: function f(row, rowWriter, context) {  
        rowWriter.writeRow({OUT: row.S});  
        rowWriter.writeRow({OUT: row.S});  
    }  
$$;
```

UDTFs (cont.)



A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His hands are visible, one holding a pen and the other resting on the clipboard. The background behind him is a plain, light color.

Client Request

Our DBA would like a quick intro to what's different in Snowflake's SQL. What are a few things we should be aware of when we run our first SQL queries?

Please make the hierarchical display more generic. We would like to show an indented name with a path for any table or view with child-parent as the first two columns.

Review of Client Request

Our DBA would like a quick intro to what's different in Snowflake's SQL. What are a few things we should be aware of when we run our first SQL queries?

Please make the hierarchical display more generic. We would like to show an indented name with a path for any table or view with child-parent as the first two columns.



Section Summary



- Object Identifiers
- DDL Statements
- Zero-Copy Cloning
- DML Statements
- Snowflake Scripting
- SQL vs SQL Scripting
- Cursor and ResultSet
- Transactions



Object Identifiers

- *identifiers*
 - NAME/Name/name → NAME
 - "Name" → Name
 - "This is a name" → This is a name
- *IDENTIFIER/TABLE functions*
 - IDENTIFIER/TABLE('MY_TABLE') ← MY_TABLE/My_Table/my_table
 - IDENTIFIER/TABLE("my_table") ← "my_table"
 - IDENTIFIER/TABLE(\$table_name) ← SET table_name = 'my_table';
- *database.schema.object* ← object name resolution

Column References and JSON Properties

- **SELECT \$1, \$2 ...**
 - from any table type (including external tables)
 - from staged files
- **v:myobj.prop1.prop2['name2'].array1[2]::string**
 - v: = table column name or alias
 - myobj = top *JSON object*
 - myobj.prop1 = myobj['prop1'], prop2.name2 = prop2['name2']
 - array1[2] = 3rd element in *JSON array*
 - ::string = cast conversion

Variables

- ***session variables*** = global variables
 - SET var = ..., UNSET var, \$var ← SHOW VARIABLES
 - SnowSQL variables = extensions, w/ var substitution
- ***local variables*** = in blocks (**Snowflake Scripting** / stored procs / functions)
 - var1 [type1] [DEFAULT expr1]
 - LET var1 := [type1] [DEFAULT / := expr1]
 - SELECT col1 INTO :var1
- ***bind variables*** = for parameterized queries, w/ runtime param values
 - SELECT (:1), (:2), TO_TIMESTAMP(?:), :2
- ***environment variables*** = for Bash (Linux/macOS) or PowerShell (Windows)
 - SET/EXPORT name=value → \$name or %name%

Structured Query Language (SQL)

- *Data Definition Language (DDL)*

- CREATE | ALTER | DROP
- COMMENT | USE
- SHOW | DESCRIBE

- *Data Manipulation Language (DML)*

- INSERT | UPDATE
- DELETE | TRUNCATE
- MERGE | EXPLAIN

- *Data Query Language (DQL)*

- SELECT | CALL

- *Transaction Control Language (TCL)*

- BEGIN TRANSACTION
- COMMIT | ROLLBACK
- DESCRIBE TRANSACTION
- SHOW TRANSACTIONS | LOCKS

- *Data Control Language (DCL)*

- USER | ROLE
- GRANT | REVOKE

DML Commands

- **INSERT [OVERWRITE]** ← with truncate
 - [ALL] **INTO <table>** [(<cols>)] ... **VALUES** (...), ...
 - **INTO <table>** [(<cols>)] ... **SELECT** ...
 - FIRST|ALL **WHEN** ... **THEN INTO** ... **ELSE INTO** ... ← multi-table insert
- **UPDATE <table> SET <col> = ..., ... [FROM ...] [WHERE ...]**
- **DELETE FROM <table> [USING ...] [WHERE ...]**
 - **TRUNCATE [TABLE] [IF EXISTS] <table>** ← w/ metadata delete

```
-- conditional multi-table insert
insert first
  when deptno <= 20
  then into dept_ctas
  else into dept_clone
select * from dept;
```

DDL Commands

- **CREATE [OR REPLACE] <type> [IF NOT EXISTS] <name> ...**
- **ALTER <type> <name> ...**
- **DROP <type> [IF EXISTS] <name> [CASCADE | RESTRICT]**
- **COMMENT [IF EXISTS] ON <type> <name> IS ...**
- **USE [<type>] <name>** ← context!
- **SHOW <types> [LIKE ...] [IN ...]**
- **DESC[RIBE] <type> <name>**

Create Tables

- **CREATE TABLE** <target>
 - (col1 type, ...) ← from scratch, empty
 - **LIKE** <source> ← empty copy
 - **AS SELECT** ... FROM <source> ← full copy **CTAS** (Create Table As Select)
 - **CLONE** <source> ← zero-copy clone (no data copied)

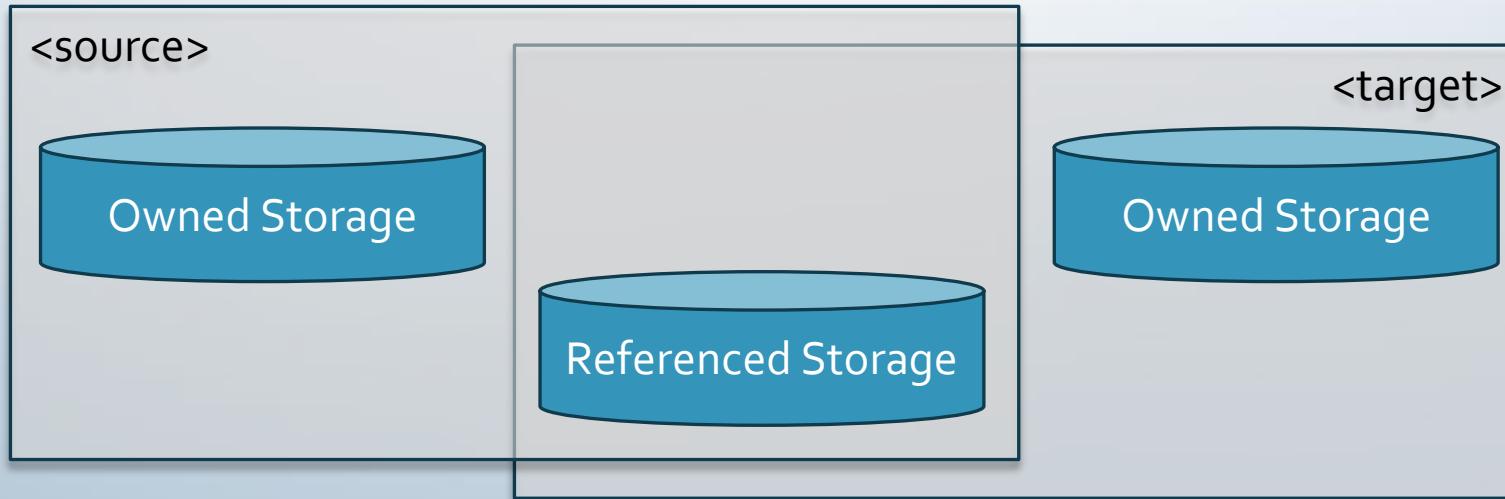
```
create table dept_like like dept;

create table dept_ctas as select * from dept;

create table dept_clone clone dept;
```

Zero-Copy Cloning

- **CREATE TABLE|SCHEMA|DATABASE <target> CLONE <source>**
- A clone *shares* all initial data with its source ← *referenced storage*
- Any further change is stored separately ← *owned storage*
- Can clone from a specific point back in time ← *time travel*



Snowflake Scripting

- SQL vs Scripting - ~**PL/SQL** (Oracle), **Transact-SQL** (Microsoft SQL Server)
- *Procedural language* (SQL is *declarative*), as *SQL extension*, since Feb 2022
- used only for Stored Procs (UDFs/UDTFs w/ SQL)
- Temporary bug in SnowSQL and Classic Console → requires **\$\$.. \$\$**

- **BEGIN .. END** *code blocks* (w/ optional **DECLARE** and **EXCEPTION**)
- Branching & Looping Statements
- **CURSOR** & **RESULTSET** Objects
- Variables & Return Value, Built-In Variables
- Exceptions

Snowflake Scripting: Code Block Template

<code block>

```
[declare]
    var1 float;
    my_exc exception (-202, 'Raised');

begin
    var1 := 3;
    let var2 := 4
    if (should_raise_exc) then
        raise my_exc;
    return var1;

[exception]
    when statement_error then
        return object_construct(
            'Error type', 'STATEMENT_ERROR',
            'SQLCODE', sqlcode,
            'SQLERRM', sqlerrm,
            'SQLSTATE', sqlstate);

end;
```

anonymous block

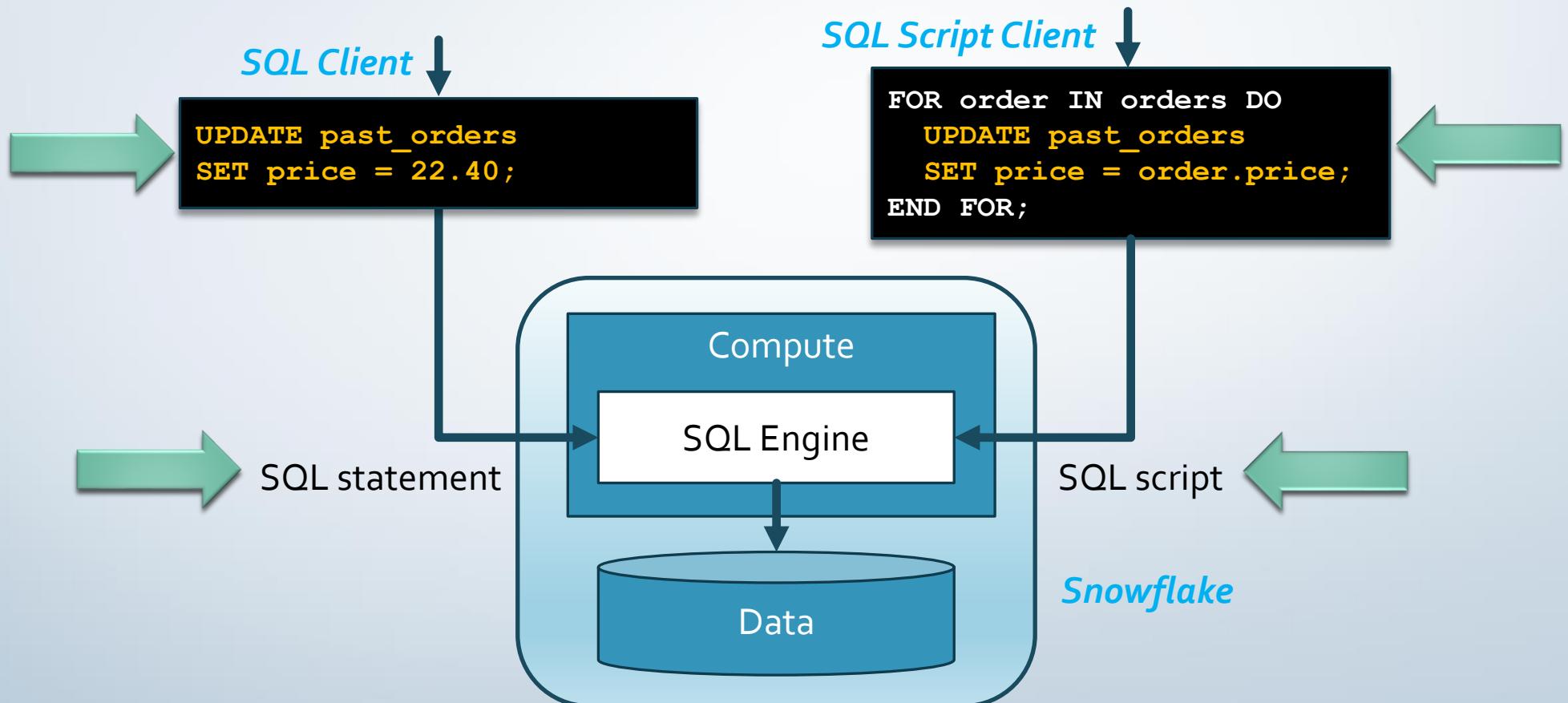
```
-- anonymous block
execute immediate
<code block>

<code block>
```

stored procedure

```
-- stored proc
create procedure proc()
    returns ...
as
    <code block>
```

Snowflake Scripting: Compared to SQL



Snowflake Scripting: Variables

- Optional top **DECLARE** block, for *SQL scalar/RESULTSET/CURSOR/EXCEPTION*
- Optional data type + **DEFAULT** value
- **LET** for vars not DECLARED, w/ optional data type + := initialization
- Can be used in **RETURN**
- Reference w/ : prefix only when in inline SQL

```
[declare]
    var1 FLOAT DEFAULT 2.3;
    res1 RESULTSET DEFAULT (SELECT ...);
    curl CURSOR FOR SELECT (?) FROM ...
    excl EXCEPTION (-202, 'Raised');

begin
    var1 := 3;
    LET var2 := var1 + 4;
    LET cur2 CURSOR FOR SELECT :var1, ...;
    LET res2 RESULTSET := (SELECT ...);
    RETURN var1;
end;
```

Snowflake Scripting: Built-In Variables

- *for last executed DML statement*
 - **SQLROWCOUNT** = rows affected by last statement, ~getNumRowsAffected()
 - **SQLFOUND** = true if last statement affected 1+ rows
 - **SQLNOTFOUND** = true if last statement affected 0 rows
 - **SQLID** = ID of the last executed query (of any kind)
- *exception classes* ← check in EXCEPTION block, with WHEN ...
 - **STATEMENT_ERROR** = execution error
 - **EXPRESSION_ERROR** = expression-related error
- *exception info* ← to use in EXCEPTION block
 - **SQLCODE** = exception_number
 - **SQLERRM** = exception_message
 - **SQLSTATE** = from ANSI SQL standard

Snowflake Scripting: Branching

- **IF ... THEN ... [ELSEIF ... THEN ... [...] ELSE ... END IF**
- **CASE cond WHEN val₁ THEN ... [...] ELSE ... END** ← simple
- **CASE WHEN cond₁ THEN ... [...] ELSE ... END** ← searched

```
-- IF-THEN-ELSE
IF (FLAG = 1) THEN
    RETURN 'one';
ELSEIF (FLAG = 2) THEN
    RETURN 'two';
ELSE
    RETURN 'Unexpected!';
END IF;
```

```
-- simple CASE
CASE (v)
    WHEN 'first' THEN
        RETURN 'one';
    WHEN 'second' THEN
        RETURN 'two';
    ELSE
        RETURN 'unexpected';
END;
```

```
-- searched CASE
CASE
    WHEN v = 'first' THEN
        RETURN 'one';
    WHEN v = 'second' THEN
        RETURN 'two';
    ELSE
        RETURN 'unexpected';
END;
```

Snowflake Scripting: Looping

- **FOR** row **IN** cursor **DO** ... **END FOR**
- **FOR** i **IN** start **TO** end **DO** ... **END FOR**
- **WHILE** cond **DO** ... **END WHILE**
- **REPEAT** ... **UNTIL** cond **END REPEAT**
- **LOOP** ... [**IF** cond **THEN** **BREAK/CONTINUE END IF**] **END LOOP**

```
-- cursor-based
FOR rec IN c1 DO
    i := i + 1;
END FOR;

-- counter-based
FOR i IN 1 TO 20 DO
    i := i + 1;
END FOR;
```

```
-- WHILE-DO + REPEAT-UNTIL
WHILE (i <= 8) DO
    i := i + 1;
END WHILE;

REPEAT
    i := i - 1;
UNTIL (i > 0)
END REPEAT;
```

```
-- LOOP + BREAK/CONTINUE
LOOP
    i := i + 1;
    IF (i > 5) THEN
        BREAK;
    ELSEIF (i < 20) THEN
        CONTINUE;
    END IF;
END LOOP;
```

Snowflake Scripting: CURSOR and RESULTSET

- *curl CURSOR FOR SELECT (?)*, ... ← w/ optional bind vars
- *res1 RESULTSET DEFAULT (SELECT :var ,...)* ← w/ optional var subst.

```
declare
    var1 DEFAULT 0;
    res1 RESULTSET DEFAULT
        (SELECT :var1, ...);
begin
    LET curl CURSOR FOR res1;
    FOR row1 IN curl DO
        ...
    END FOR;
end;
```

```
begin
    LET curl CURSOR FOR SELECT (?) ...;
    OPEN curl USING (:col1);
    RETURN TABLE(
        RESULTSET_FROM_CURSOR(curl));
end;
```

```
declare
    id1 INTEGER DEFAULT 0;
    curl CURSOR FOR SELECT ...;
begin
    OPEN curl;
    FETCH curl INTO id1;
    CLOSE curl;
    RETURN id1;
end;
```

```
begin
    LET var1 := 'FROM ...';
    LET stm1 := 'SELECT ...' || var1;
    LET res1 RESULTSET :=
        (EXECUTE IMMEDIATE :stm1);
    RETURN TABLE(res1);
end;
```

Snowflake Scripting: Exceptions

- `exc1 EXCEPTION (...)` ← custom exception object
- `RAISE exc1` ← go to EXCEPTION section, check WHEN exc1

```
DECLARE
    exc1 EXCEPTION (-2002, 'Raised');

BEGIN
    LET err := true;
    IF (err) THEN
        RAISE exc1;
    END IF;

EXCEPTION
    WHEN STATEMENT_ERROR THEN ...
    WHEN exc1 THEN ...
    WHEN OTHER THEN
        RETURN SQLCODE;
        RAISE;
END;
```

Transaction Control Language (TCL)

- BEGIN TRANSACTION
- COMMIT | ROLLBACK

- DESCRIBE TRANSACTION
- SHOW TRANSACTIONS

- SHOW LOCKS

Transactions

- **explicit** = BEGIN TRANSACTION ... COMMIT/ROLLBACK
 - **implicit** = single DDL stmt / DML stmt if AUTOCOMMIT on
 - **inner** = no commit/rollback if already rolled-back/committed
 - **nested** = no such thing (see before)
 - **scoped** = whole inside a stored proc and end in the same session
-
- **read committed** = the only isolation level supported for tables
 - **system\$abort_transaction** = abort a running transaction by ID

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His hands are visible, one holding the clipboard and the other holding a pen. The background behind him is a plain, light color.

Client Request

Our customers table is owned by an external department, but we need to keep an up-to-date copy into our database as well. Please make sure it is sync-ed frequently, and we can see all changes in almost real-time.

New employees can be added through CSV files dropped into an Amazon S3 bucket folder. We cannot update or delete exiting employees through this method.

Review of Client Request

Our customers table is owned by an external department, but we need to keep an up-to-date copy into our database as well. Please make sure it is sync-ed frequently, and we can see all changes in almost real-time.

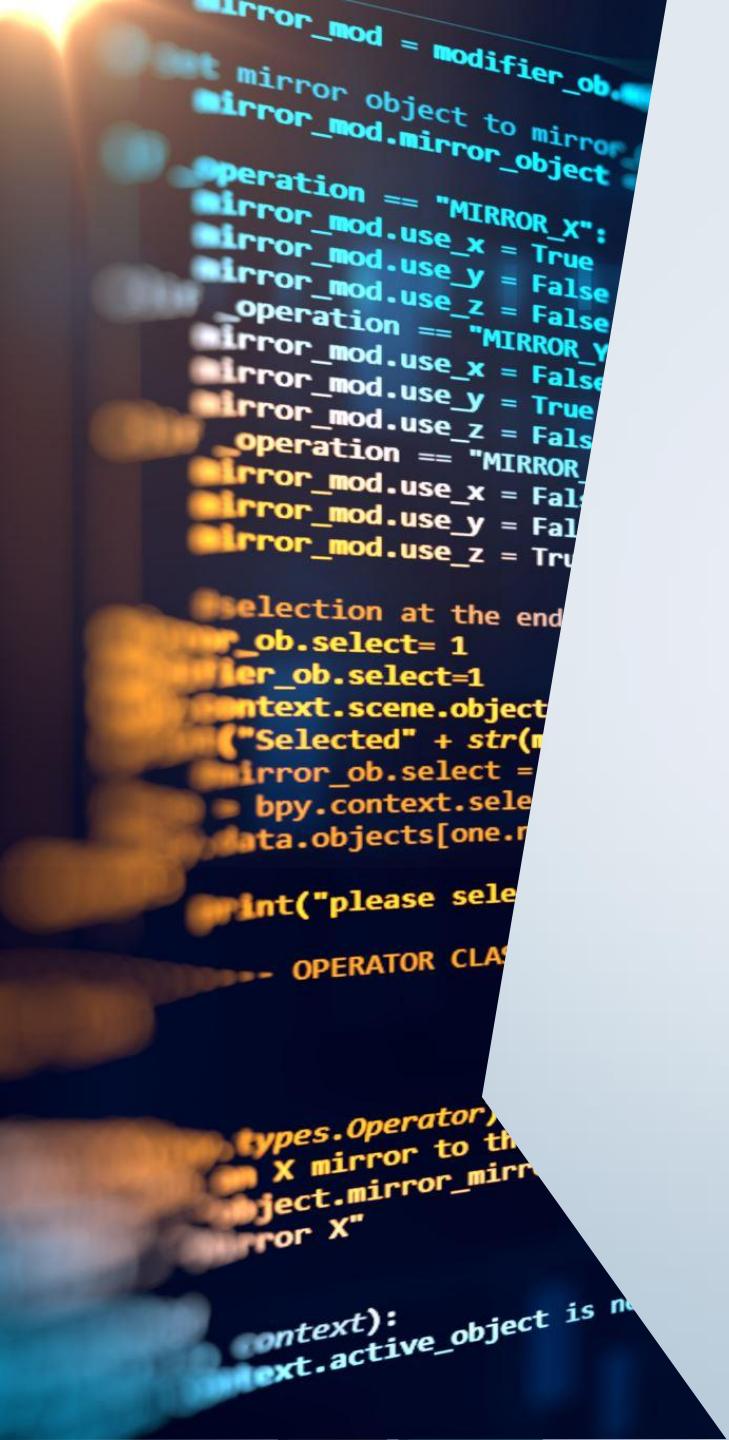
New employees can be added through CSV files dropped into an Amazon S3 bucket folder. We cannot update or delete exiting employees through this method.



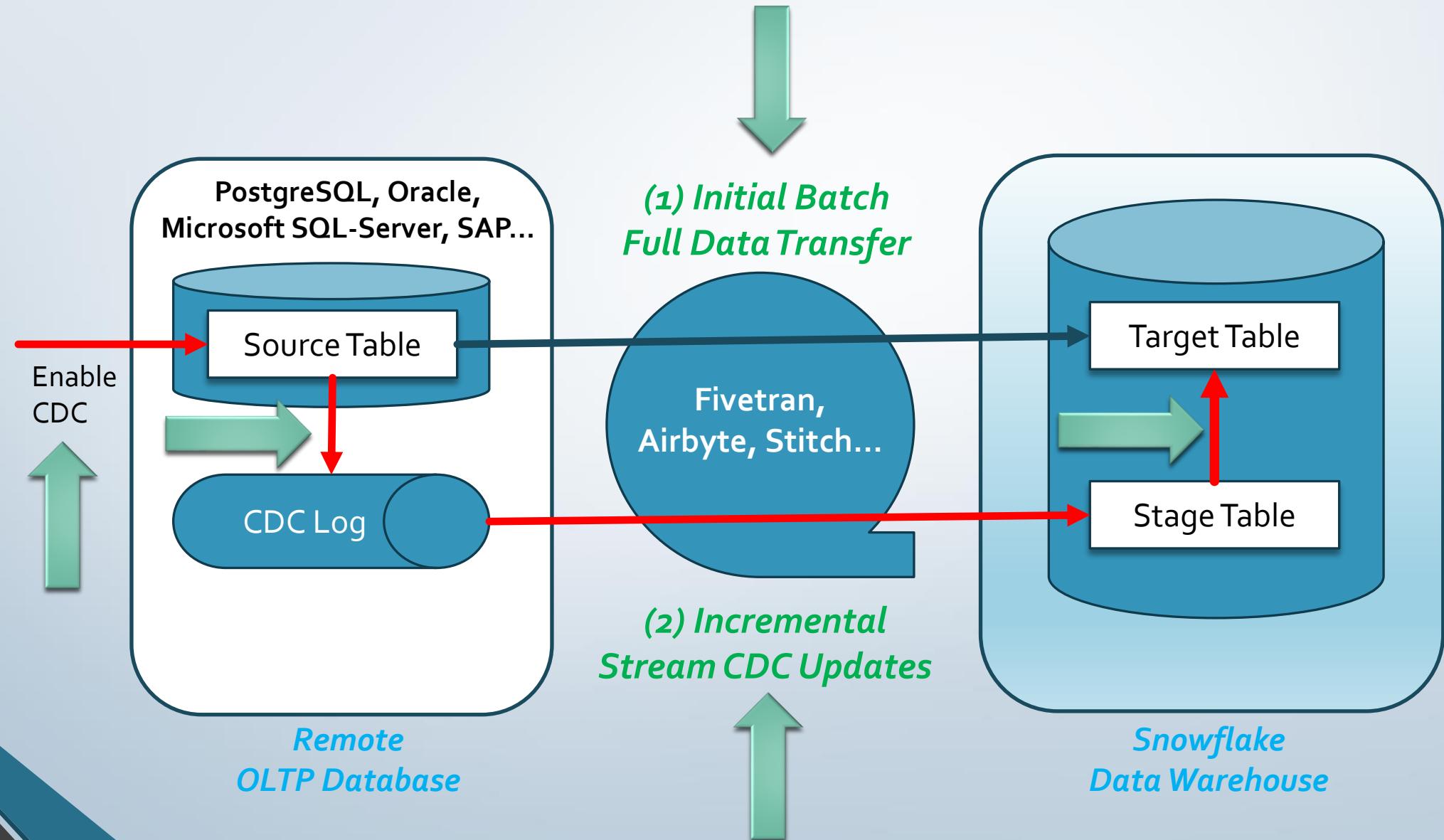
Section Summary



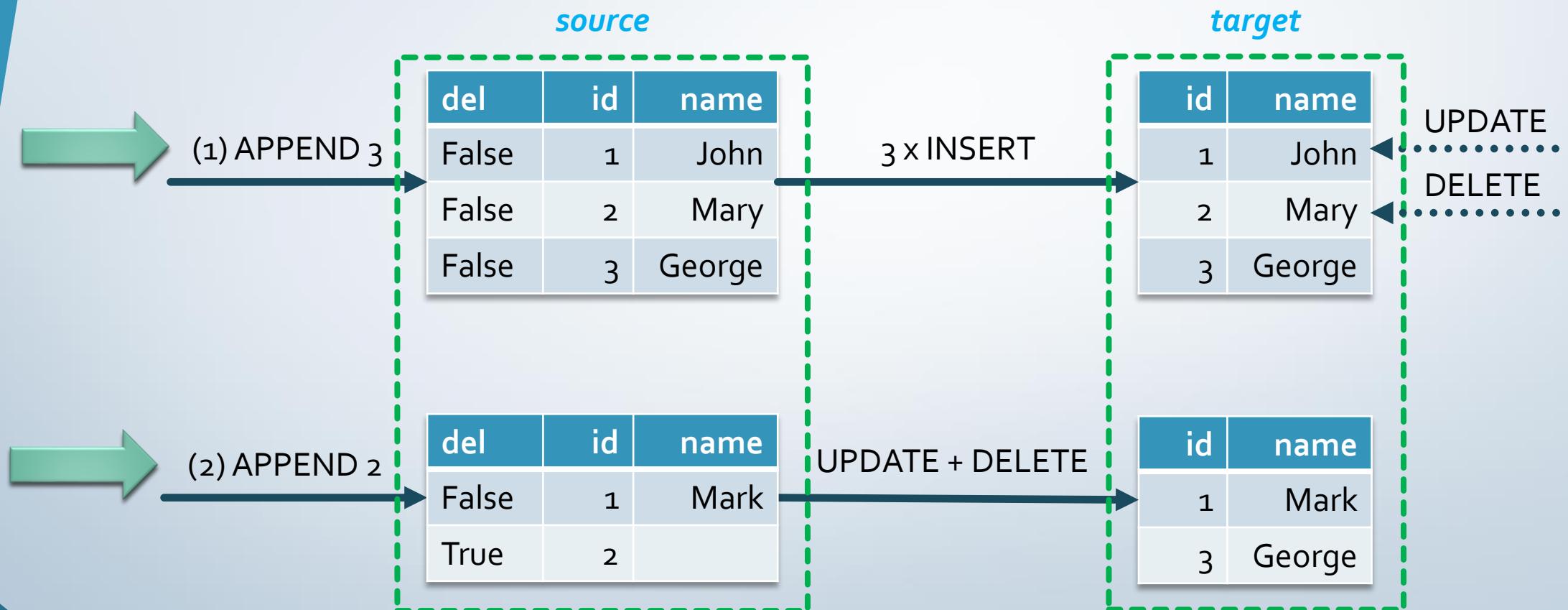
- Change Data Capture (CDC)
- MERGE Statement
- Change Tracking
- Streams and Tasks
- Dynamic Tables
- Snowpipe



Batch/Stream Data Transfer to Snowflake



Change Data Capture (CDC)



CDC Methods

- *CDC with Manual MERGE Statement*
 - `MERGE INTO ... USING ... WHEN ...`
- *CDC with Change Tracking*
 - `ALTER TABLE ... SET CHANGE_TRACKING = TRUE;`
 - `CHANGES`(information => DEFAULT/APPEND_ONLY)
- *CDC with Stream and Task*
 - `CREATE STREAM ... ON TABLE source`
 - `CREATE TASK ... AS ...`
 - `SYSTEM$HAS_STREAM_DATA(stream)`
 - `METADATA$ACTION, METADATA$ISUPDATE`
- *CDC with Dynamic Table*
 - `CREATE DYNAMIC TABLE ...`

Manual CDC with MERGE Statement

```
-- source (table) --> target (table)
CREATE TABLE source(del BOOLEAN, id INT, name STRING);
CREATE TABLE target(id INT, name STRING);

-- call after each batches of INSERT/DELETE/UPDATE on the source table
MERGE INTO target t USING source s ON t.id = s.id
    WHEN MATCHED AND del
        THEN DELETE
    WHEN MATCHED AND NOT del
        THEN UPDATE SET t.name = s.name
    WHEN NOT MATCHED AND NOT del
        THEN INSERT (id, name) VALUES (s.id, s.name);
```

CDC with Change Tracking

```
-- source (table) --> target (table)
CREATE TABLE source(id INT, name STRING);

-- enable change tracking and save that initial point in time
ALTER TABLE source SET CHANGE_TRACKING = TRUE;
SET ts1 = (SELECT CURRENT_TIMESTAMP());

-- perform INSERT/UPDATE/DELETE on the source table

-- see all INSERTs (since that point in time)
SELECT * FROM source
CHANGES (information => APPEND_ONLY) AT(timestamp => $ts1);

-- create target with all changes (since that point in time)
CREATE OR REPLACE TABLE target AS
    SELECT id, name FROM source
    CHANGES (information => DEFAULT) AT(timestamp => $ts1);
```

CDC with Stream and Task

```
CREATE TABLE source(id INT, name STRING);           ← source table
CREATE TABLE target(id INT, name STRING);           ← target table

CREATE STREAM stream1 ON TABLE source;             ← stream on source table
CREATE TASK task1
    WAREHOUSE = compute_wh
    SCHEDULE = '1 minute'
    WHEN SYSTEM$STREAM_HAS_DATA('stream1')          ← task for continuous CDC

AS
    MERGE INTO target t USING stream1 s ON t.id = s.id
    WHEN MATCHED AND metadata$action = 'DELETE'      ← DELETE
        AND metadata$isupdate = 'FALSE'
        THEN DELETE
    WHEN MATCHED AND metadata$action = 'INSERT'       ← DELETE + INSERT
        AND metadata$isupdate = 'TRUE'
        THEN UPDATE SET t.name = s.name
    WHEN NOT MATCHED AND metadata$action = 'INSERT'   ← INSERT
        THEN INSERT (id, name) VALUES (s.id, s.name);

ALTER TASK task1 RESUME/SUSPEND;                  ← suspended by default!
EXECUTE TASK task1;                            ← manual execution
```

CDC with Dynamic Table

```
CREATE TABLE source(id INT, name STRING);           ← source table

CREATE DYNAMIC TABLE target
  WAREHOUSE = compute_wh
  TARGET_LAG = '1 minute'
AS
  SELECT id, name FROM source;                    ← w/ inferred schema

-- perform INSERT/UPDATE/DELETE on the source table

ALTER DYNAMIC TABLE target SUSPEND;               ← stop internal task
```

Snowpipes

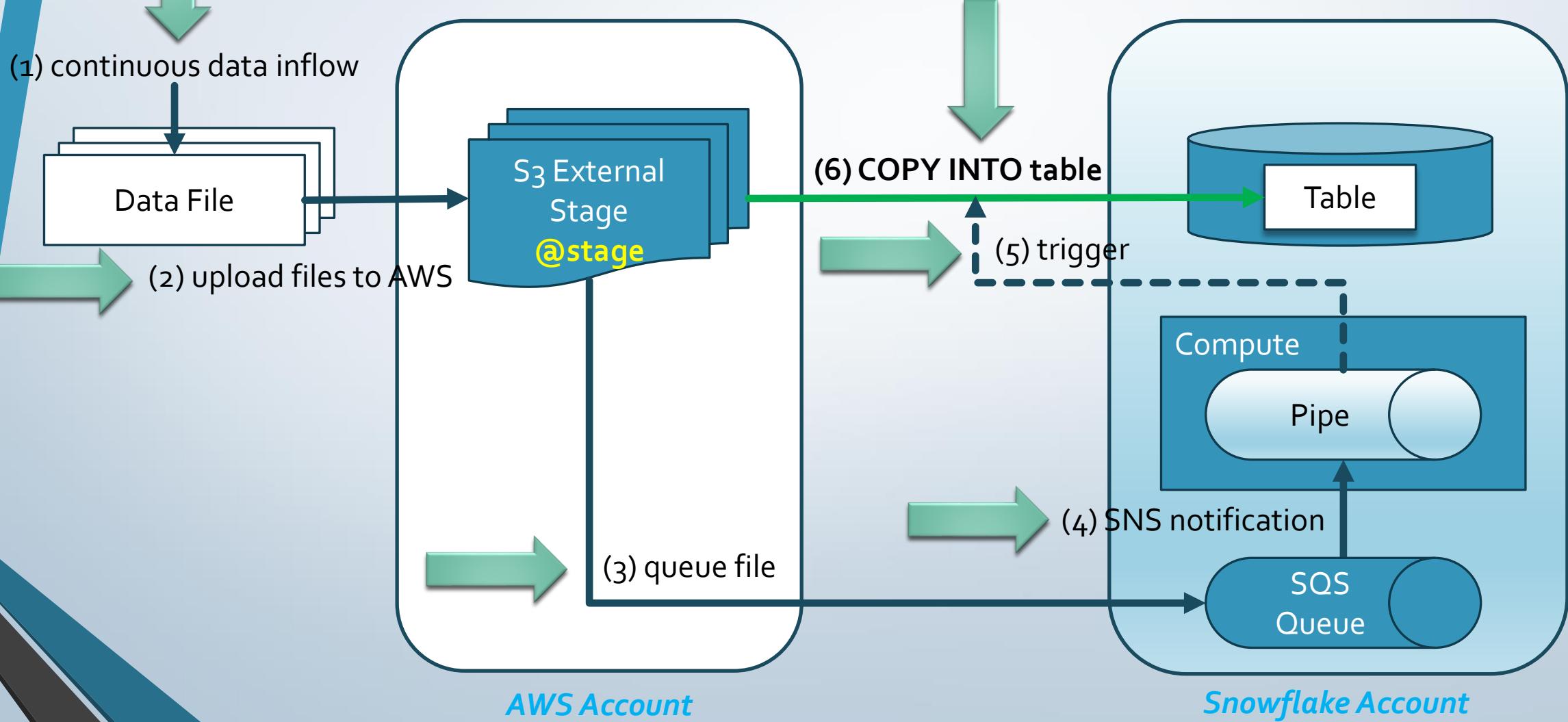
- **CREATE PIPE** *pipe AS COPY INTO table FROM @stage;*
- **AUTO_INGEST** on
 - automatic data loading
 - from external stages only (S3, Azure Storage, Google Storage)
- **AUTO_INGEST** off
 - no automatic data loading
 - from external/internal named/table stages (not user stages!)
 - only with **Snowpipe REST API** endpoint calls

Snowpipe on S3

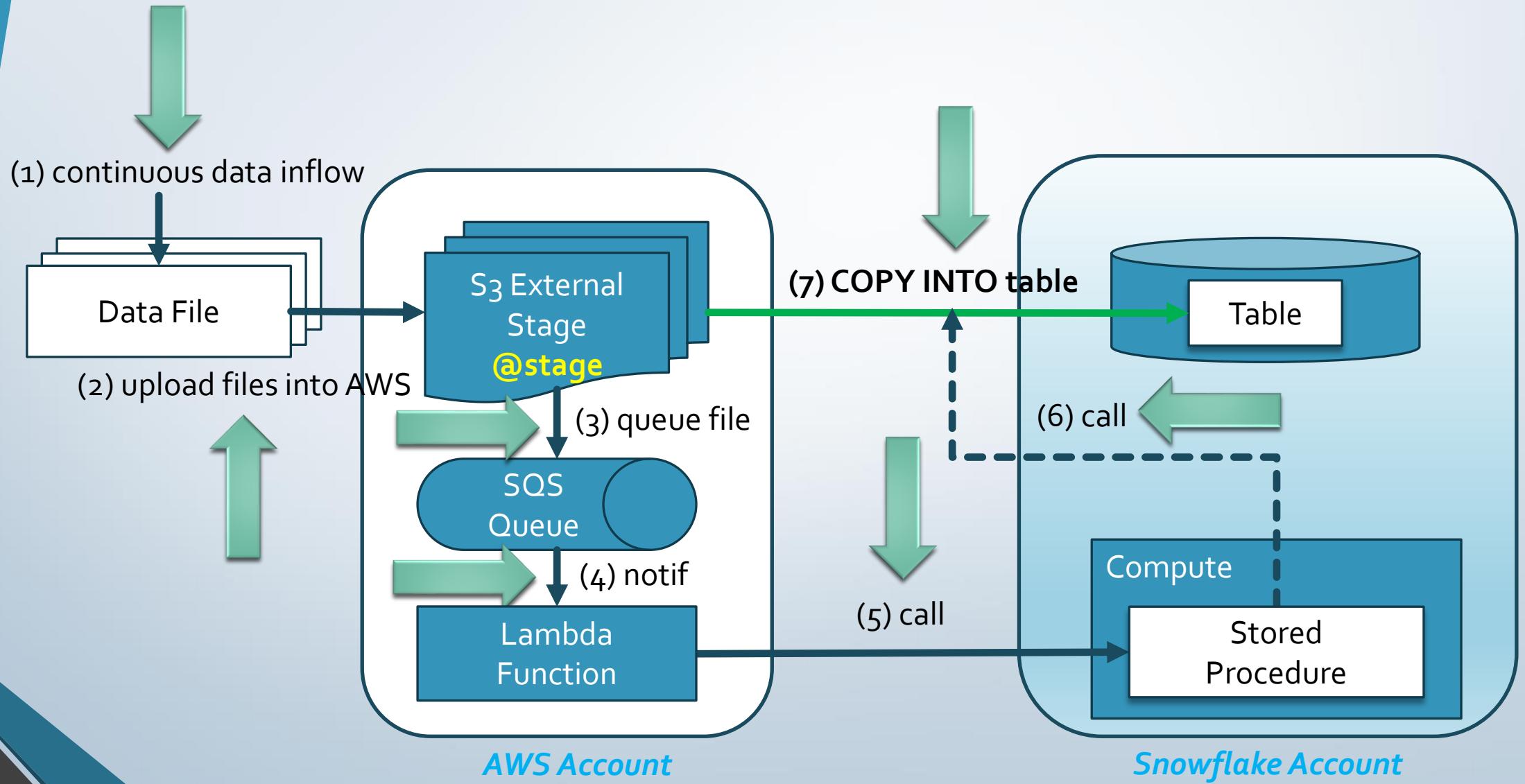
- Create a continuous loading pipe based on the external S3 stage created before
 - AUTO_INGEST = True
 - COPY INTO new table FROM external stage
- Add an event notification for the S3 bucket/folder
 - for "All object create events"
 - on SQS Queue w/ ARN copied from **show pipes** for crt pipe
- Upload some CSV files in the folder
 - check pipe status: **select system\$pipe_status('mypipe_s3');**

```
CREATE PIPE mypipe_s3
    AUTO_INGEST = TRUE
AS
    COPY INTO emp_s3 FROM @mystage_s3
        FILE_FORMAT = (TYPE = 'CSV')
        ON_ERROR = 'CONTINUE';
```

Snowpipe on S3



Snowpipe on S3: Alternative





Client Request

Hello, the VP of Sales here...

Are there some other storage data formats better suited for such an employee-manager hierarchical topology?

What other free or open-source data visualization options exist for this kind of relationship?

You already came up with recursive queries in SQL, but I am wondering if we cannot use graphs or charts instead...

Review of Client Request

Hello, the VP of Sales here...

Are there some other storage data formats better suited for such an employee-manager hierarchical topology?

What other free or open-source data visualization options exist for this kind of relationship?

You already came up with recursive queries in SQL, but I am wondering if we cannot use graphs or charts instead...



Section Summary



- Hierarchical Data Formats
- Graphs (with GraphViz)
- Charts (with Plotly)
- Trees (with D3)



Hierarchical Data Formats: JSON, XML, YAML

JSON

```
{  
    "name": "KING",  
    "children": [  
        {  
            "name": "BLAKE",  
            "children": [  
                { "name": "MARTIN" },  
                { "name": "JAMES" }  
            ]  
        },  
        {  
            "name": "JONES",  
            "children": [  
                { "name": "FORD" }  
            ]  
        }  
    ]  
}
```

XML

```
<?xml version="1.0"?>  
<object>  
    <name>KING</name>  
    <children>  
        <object>  
            <name>BLAKE</name>  
            <children>  
                <object>  
                    <name>MARTIN</name>  
                </object>  
                <object>  
                    <name>JAMES</name>  
                </object>  
            </children>  
        </object>  
        ...  
    </children>  
</object>
```

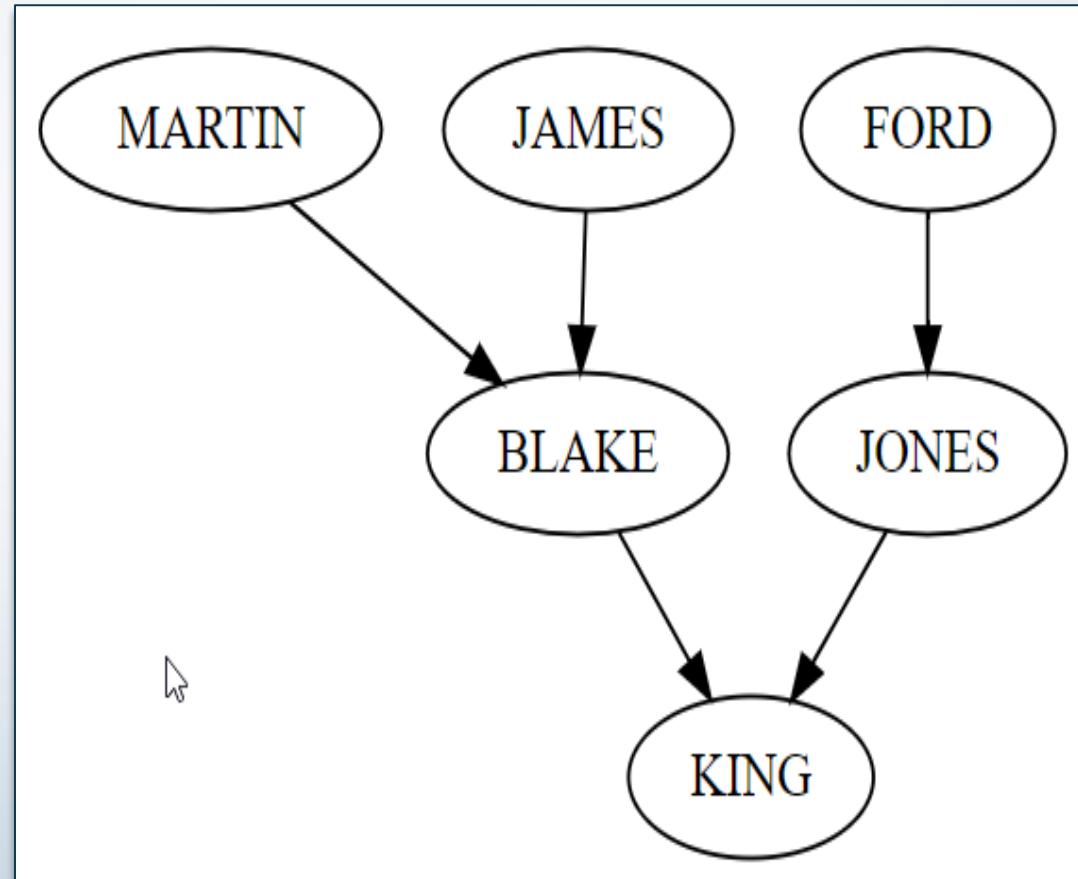
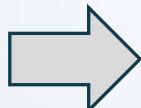
YAML

```
KING  
- BLAKE  
  - MARTIN  
  JAMES  
JONES  
- FORD
```

GraphViz DOT Notation: Edges

```
KING → BLAKE  
BLAKE → MARTIN  
BLAKE → JAMES  
KING → JONES  
JONES → FORD
```

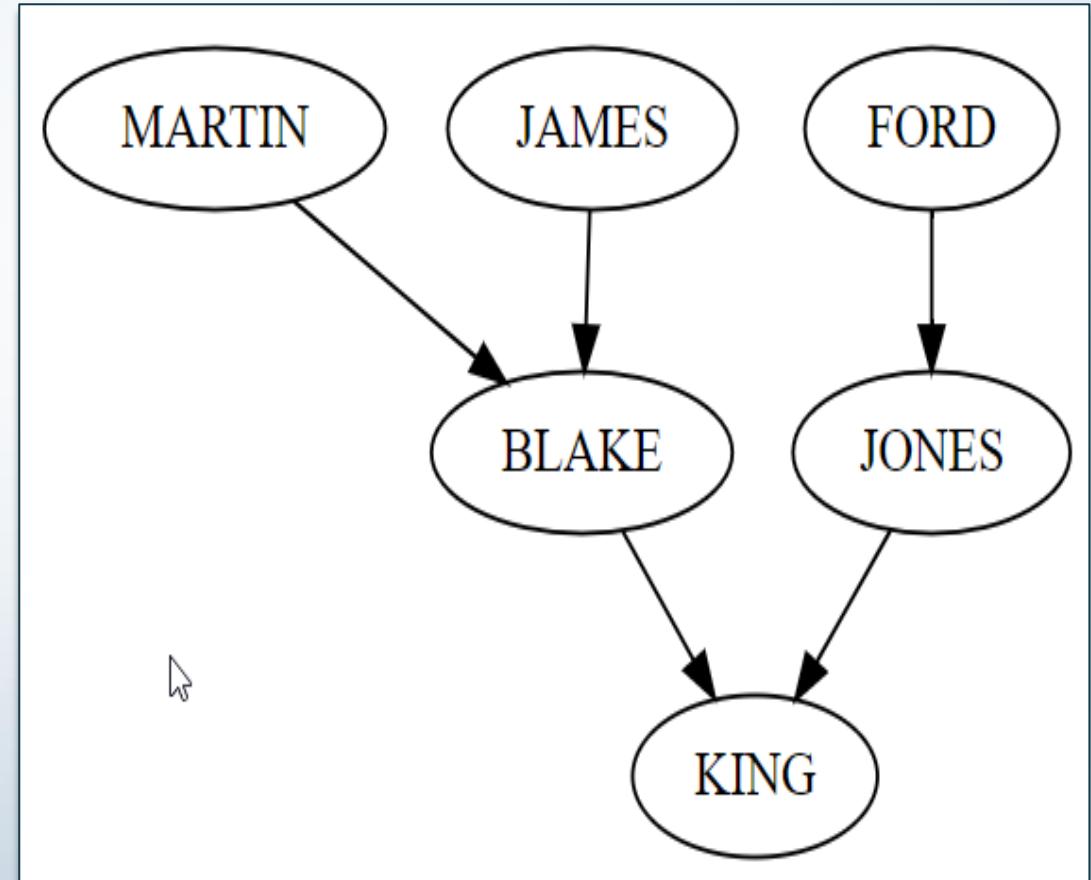
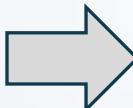
```
digraph {  
    BLAKE -> KING;  
    MARTIN -> BLAKE;  
    JAMES -> BLAKE;  
    JONES -> KING;  
    FORD -> JONES;  
}
```



GraphViz DOT Notation: Nodes & Edges

```
digraph {
    n7839 [label="KING"];
    n7698 [label="BLAKE"];
    n7566 [label="JONES"];
    n7902 [label="FORD"];
    n7654 [label="MARTIN"];
    n7900 [label="JAMES"];

    n7698 -> n7839;
    n7654 -> n7698;
    n7900 -> n7698;
    n7566 -> n7839;
    n7902 -> n7566;
}
```

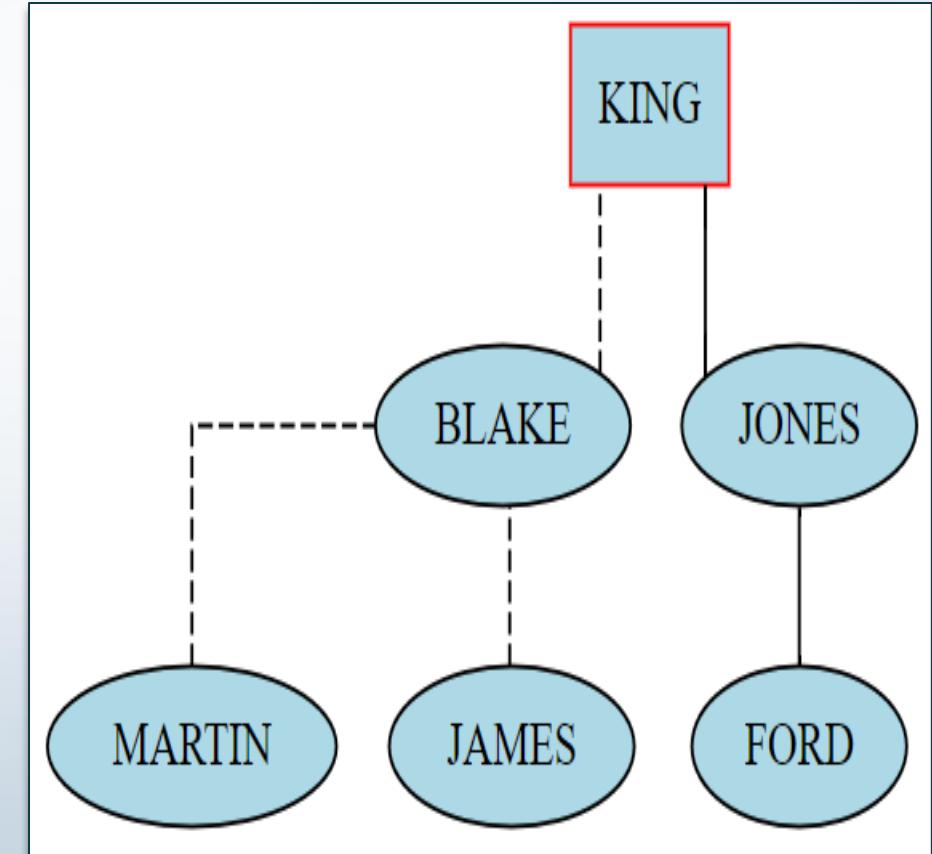
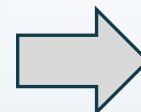


GraphViz DOT Notation: Styling

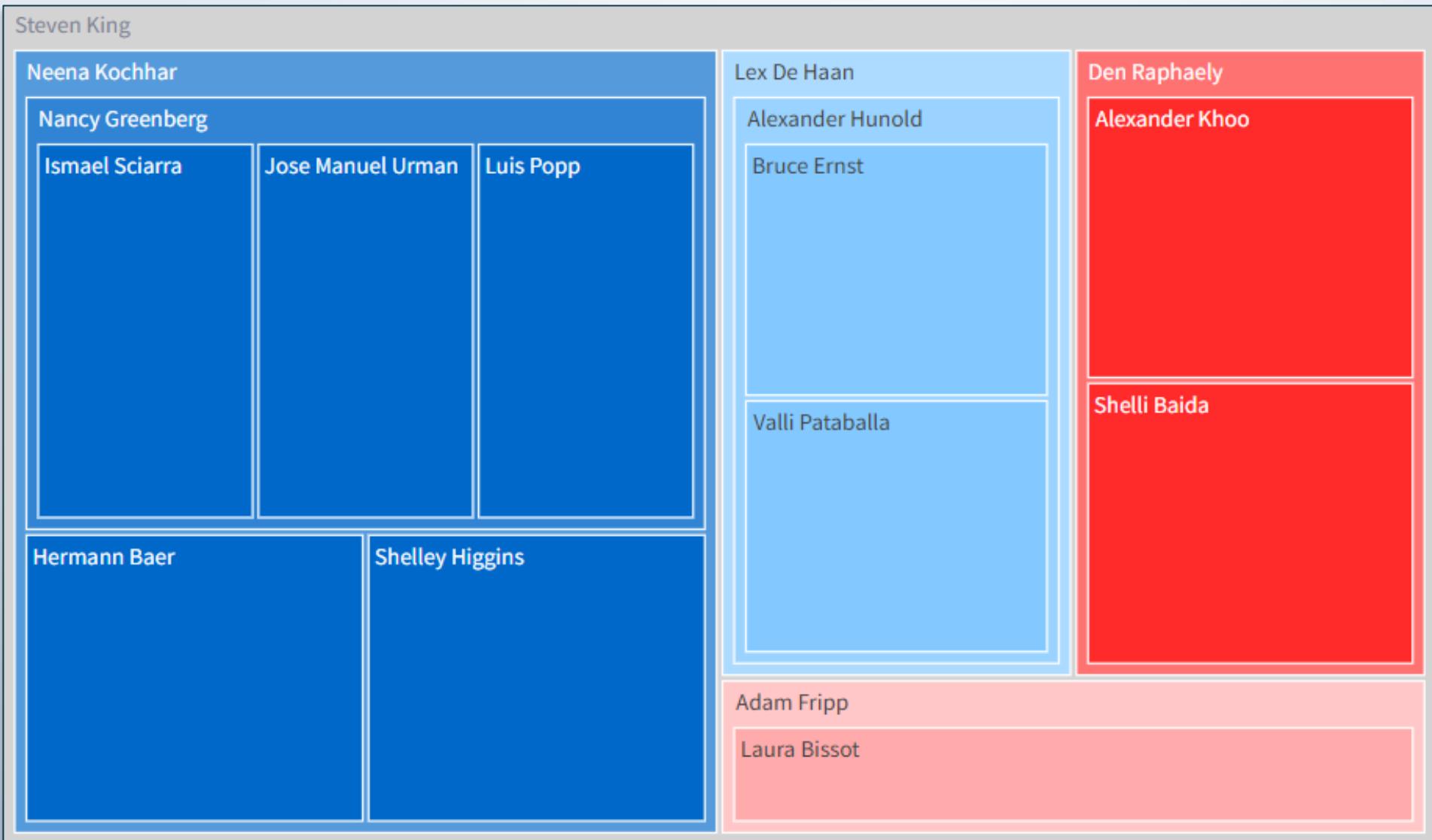
```
digraph {
    graph [rankdir="BT"
        bgcolor="#ffffff"
        splines="ortho"]
    node [style="filled"
        fillcolor="lightblue"]
    edge [arrowhead="None"]

    n7839 [label="KING"
        shape="rect" color="red"];
    n7698 [label="BLAKE"];
    n7566 [label="JONES"];
    n7902 [label="FORD"];
    n7654 [label="MARTIN"];
    n7900 [label="JAMES"];

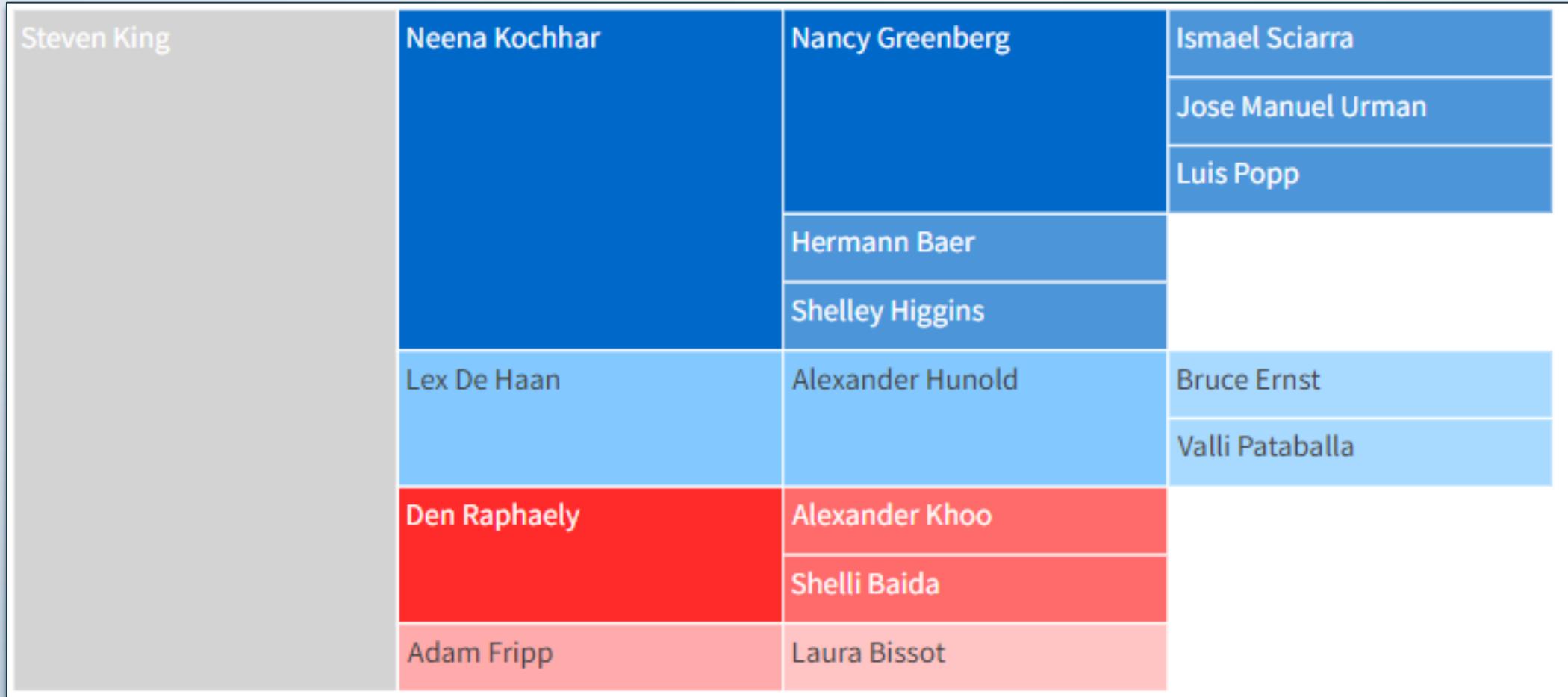
    n7698 -> n7839 [style="dashed"];
    n7654 -> n7698 [style="dashed"];
    n7900 -> n7698 [style="dashed"];
    n7566 -> n7839;
    n7902 -> n7566;
}
```



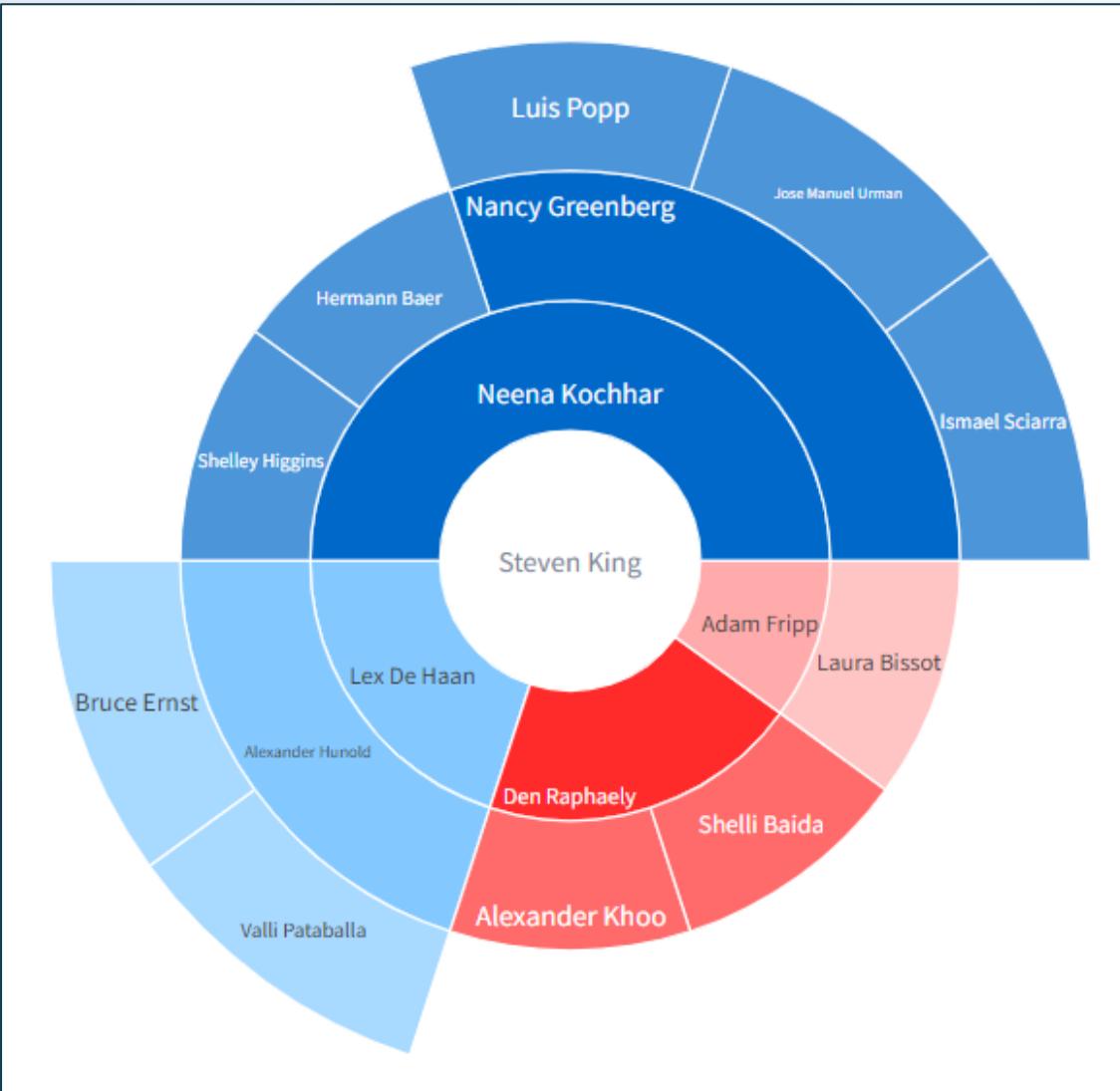
Plotly Charts: Treemap



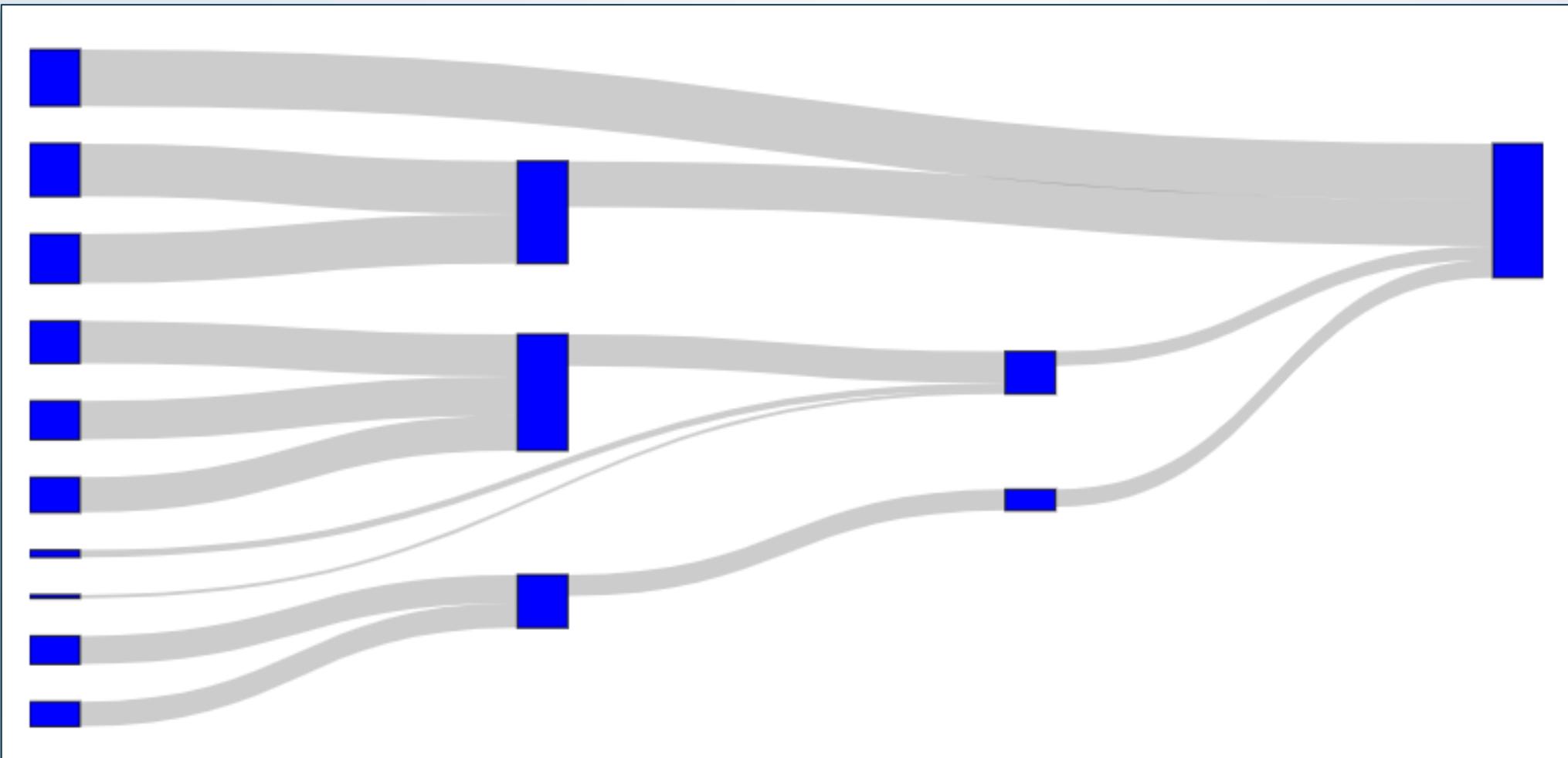
Plotly Charts: Icicle



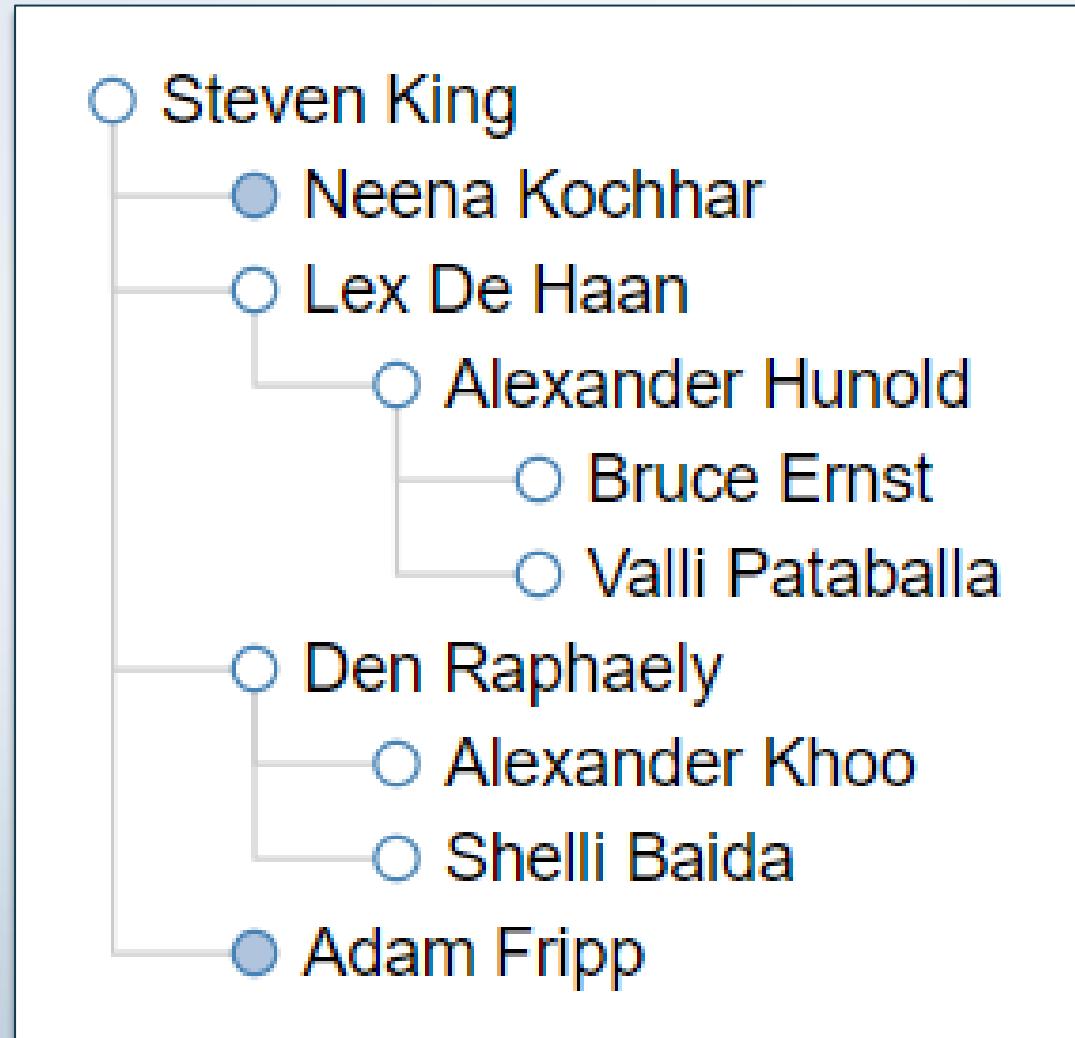
Plotly Charts: Sunburst



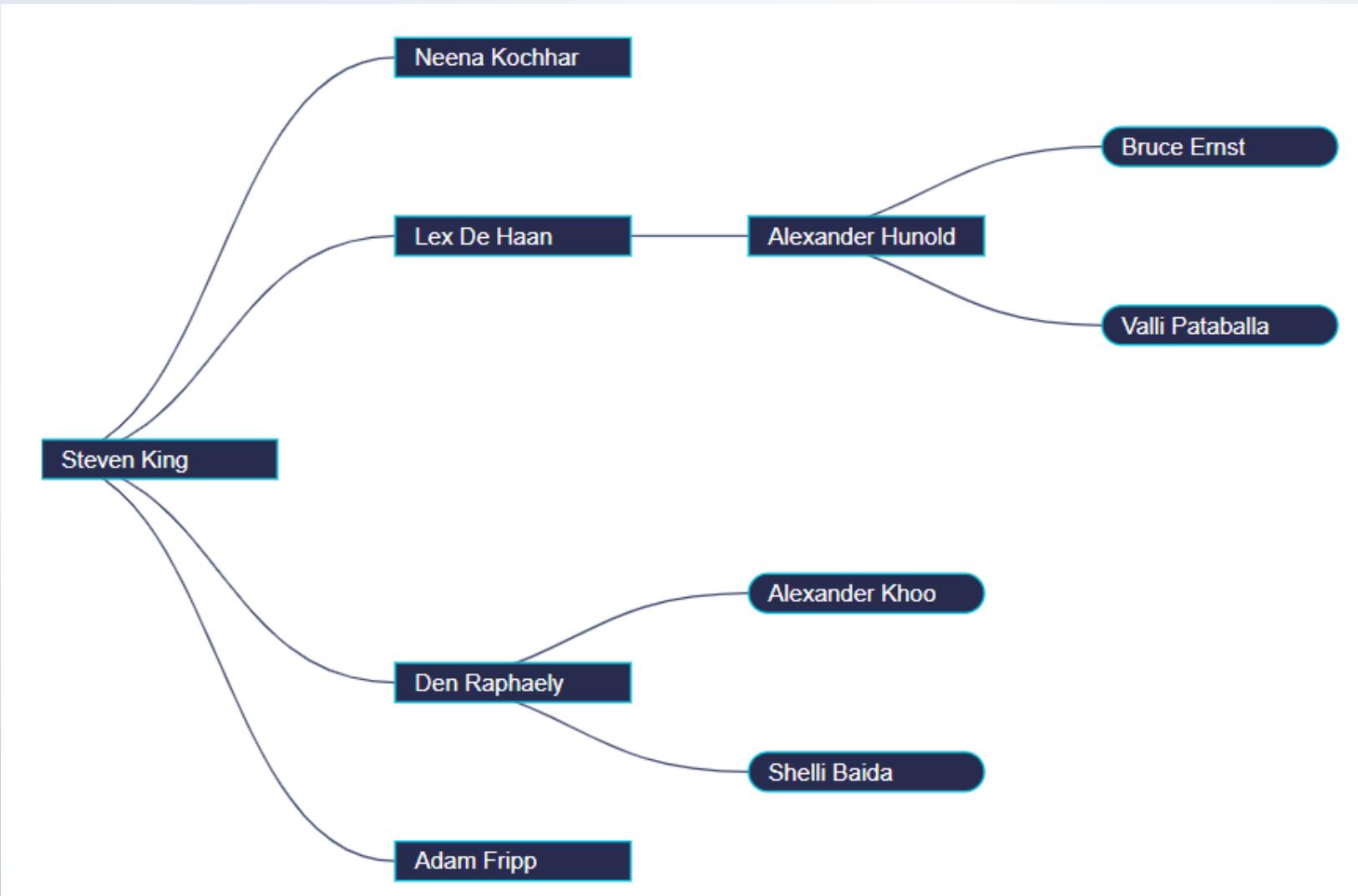
Plotly Charts: Sankey



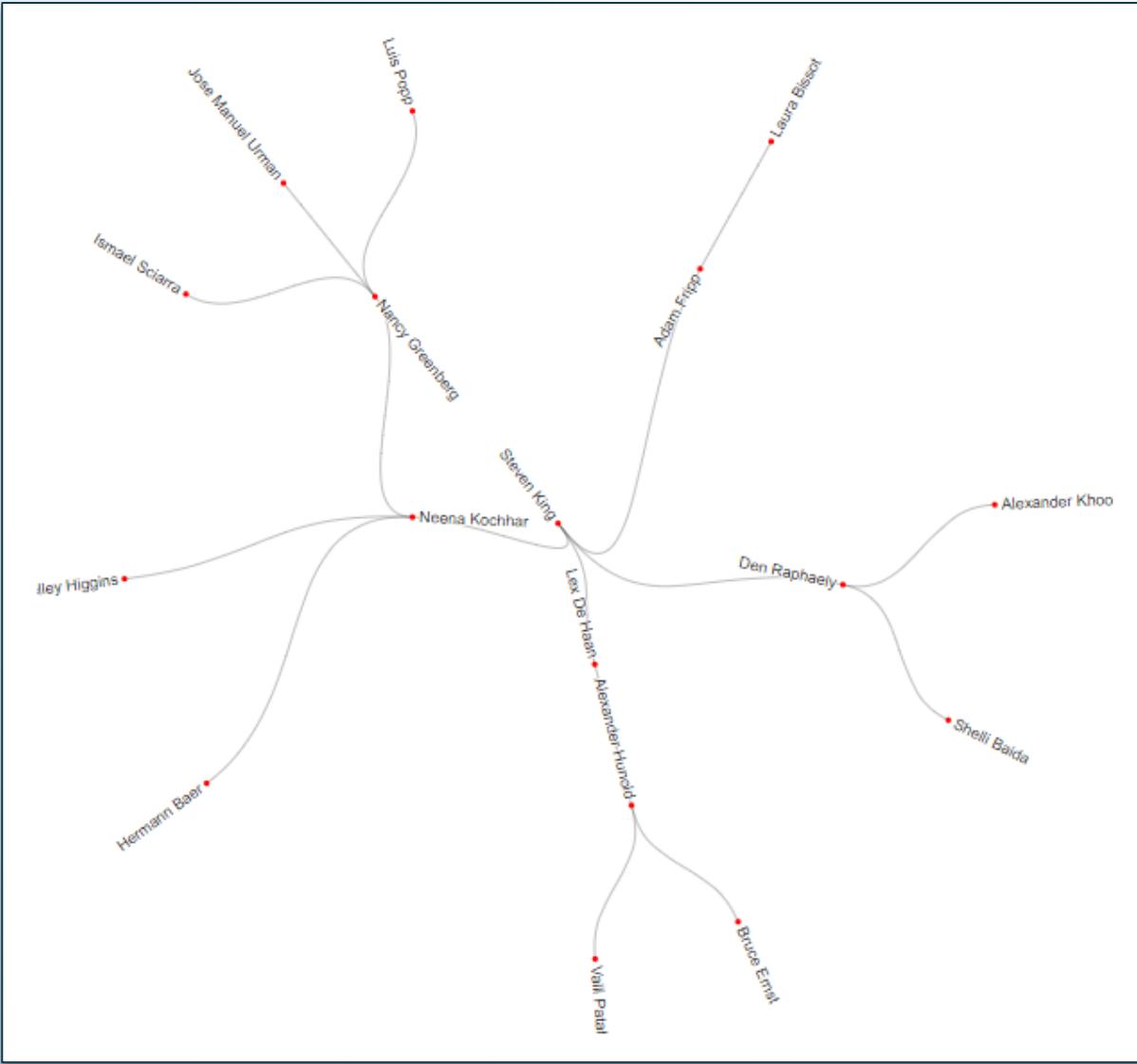
D3: Collapsible Tree



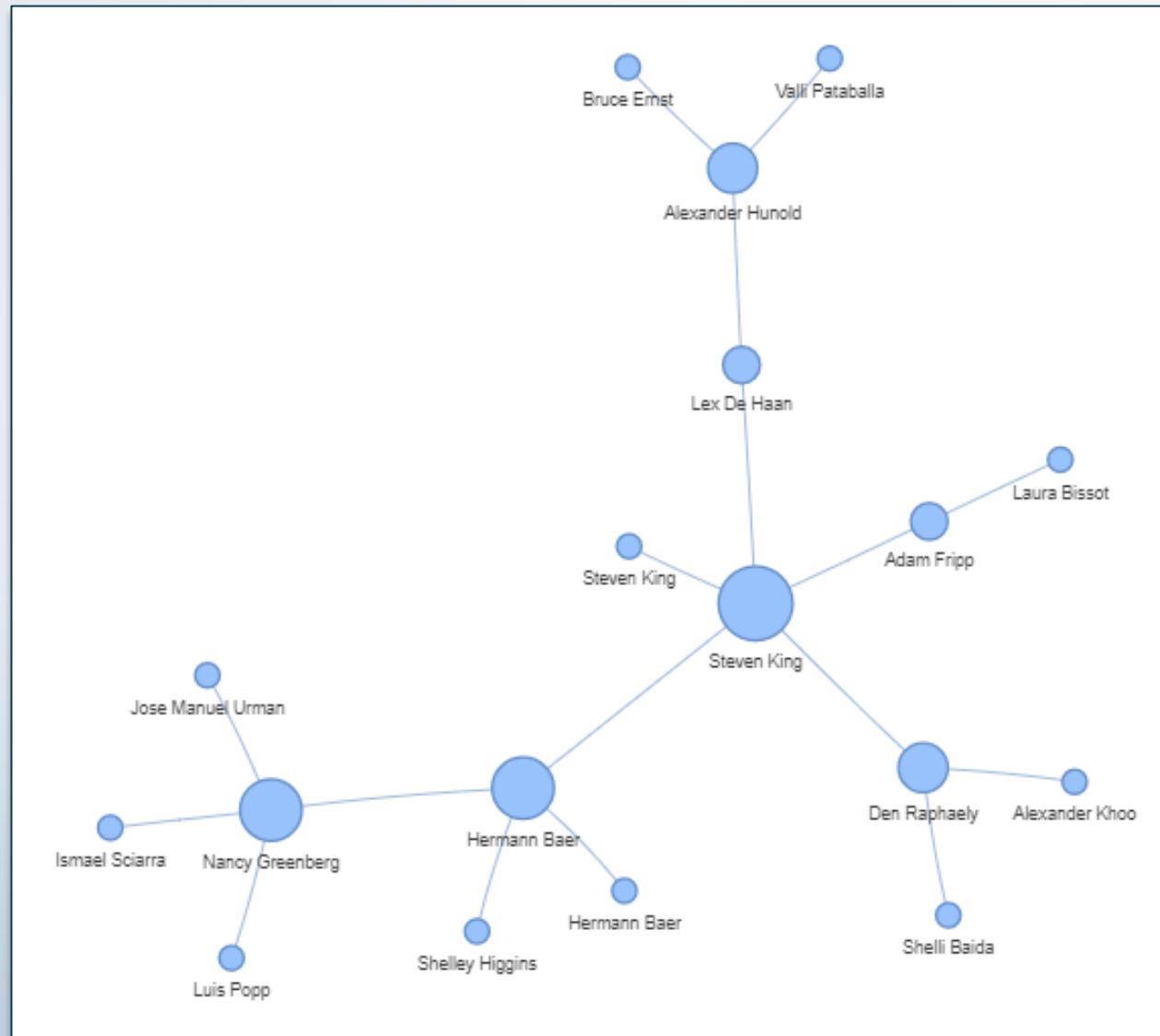
D3: Linear Dendrogram



D3: Radial Dendrogram



D3: Network Graph





Client Request

Can you make it customizable, to be able to upload any CSV file, and select the columns we want?

And find an easy solution to be able somehow to share it online. We here in Sales have limited technical knowledge and could not easily install and run your demo project as it is...

Review of Client Request

Can you make it customizable, to be able to upload any CSV file, and select the columns we want?

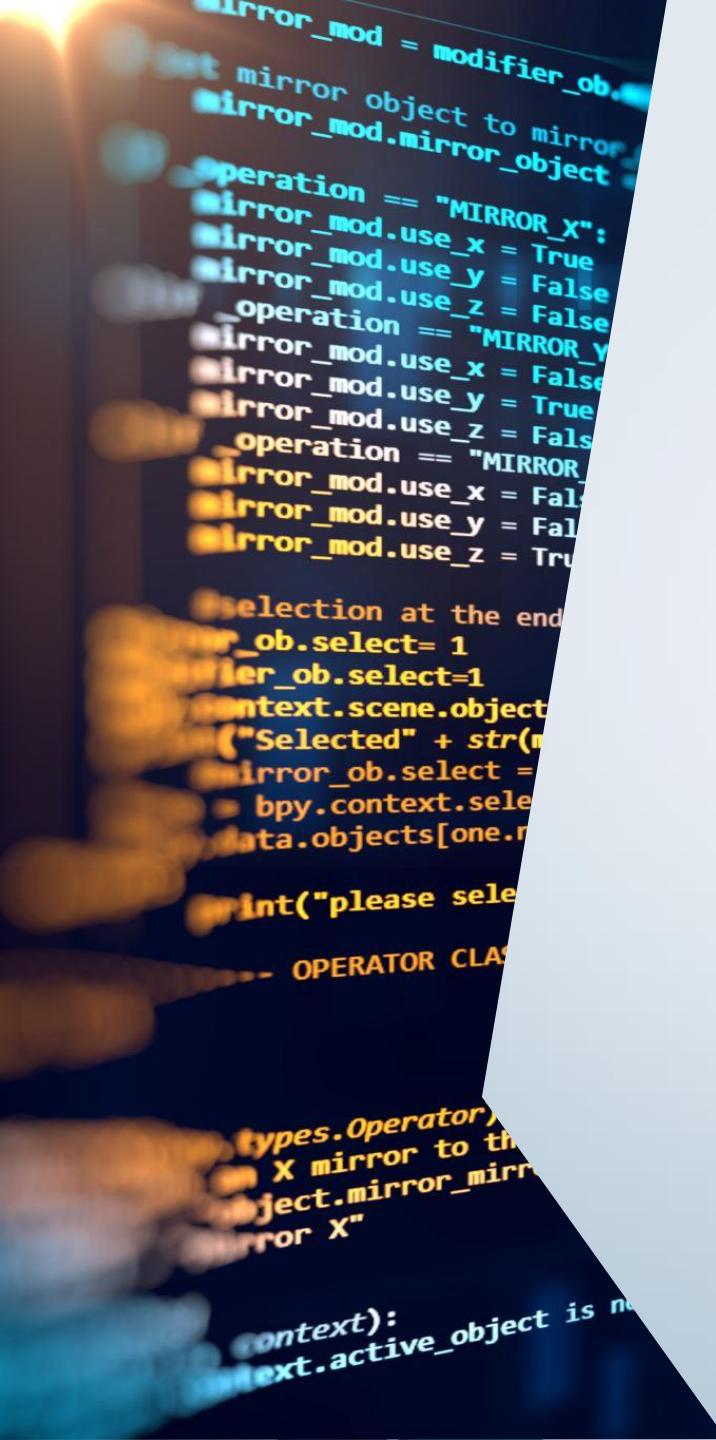
And find an easy solution to be able somehow to share it online. We here in Sales have limited technical knowledge and could not easily install and run your demo project as it is...



Section Summary



- Introduction to Streamlit
- Layout Components
- Interactive Widgets
- State and Callbacks
- Data Cache
- Multi-Page Applications
- Test as Local Web App
- Deploy and Share as Web App



Introduction to Streamlit

- **History**
 - Bought by Snowflake in 2022 for \$800M
 - Integrated with Snowpark: Streamlit in Snowflake + Native Apps
- **Features**
 - RAD framework for data science experiments (~VB, Access, Python at the app level)
 - Connect to all sorts of data sources (Snowflake etc)
 - Instant rendering as charts or HTML content, using many third-party components
- **Architecture**
 - Minimalistic layout components and simple input controls (single event per control!)
 - Full page rerun after any input control integration (tricky at the beginning!)
 - Cache between reruns with session state and control callbacks (added recently)
 - Data and resource/object cache (to avoid data reloads between page reruns)
- **Development**
 - Great for prototyping and proof-of-concept simple apps (not like heavy React apps!)
 - Support for single and multi-page applications
 - Test as local web app (not standalone!)
 - Share and deploy as remote web app to Streamlit Cloud (for 100% free!)

Layout Components

- st.sidebar ← collapsible left sidebar
 - st.tabs ← tab control
-
- st.columns ← side-by-side horizontal containers
 - st.expander ← collapsible container
 - st.container ← multi-element container
 - st.empty ← single-element placeholder

Layout Components

The diagram illustrates various Streamlit layout components:

- Container:** A code block shows how to use `st.container()` to group components. It includes examples of text output "First in container", "Second in container", and "Outside the container".
- Sidebar:** A code block shows `st.sidebar.selectbox("Select Box:", ["S", "M"])`. It displays a dropdown menu with options "S" and "M".
- Tabs:** A code block shows `tabs = st.tabs("Tab 1", "Tab 2", "Tab 3")` and `tabs[0].write("Text in first tab")`, `tabs[1].write("Text in second tab")`. It shows three tabs labeled "Tab 1", "Tab 2", and "Tab 3".
- Expander:** A code block shows `exps = st.expander("Collapsed", expanded=False)` and `exps.write("This is collapsed")`. It also shows an example with `with st.expander("Expanded"):` and `st.write("This is expanded")`.
- Columns:** A code block shows `cols = st.columns(3)` and `cols[0].write("Column 1")`, `cols[1].write("Column 2")`, `cols[2].write("Column 3")`. It displays three columns labeled "Column 1", "Column 2", and "Column 3".
- Empty:** A code block shows `with st.empty():`, `st.write("Replace this...")`, and `st.write("...by this one")`. It shows a placeholder area with the text "...by this one".

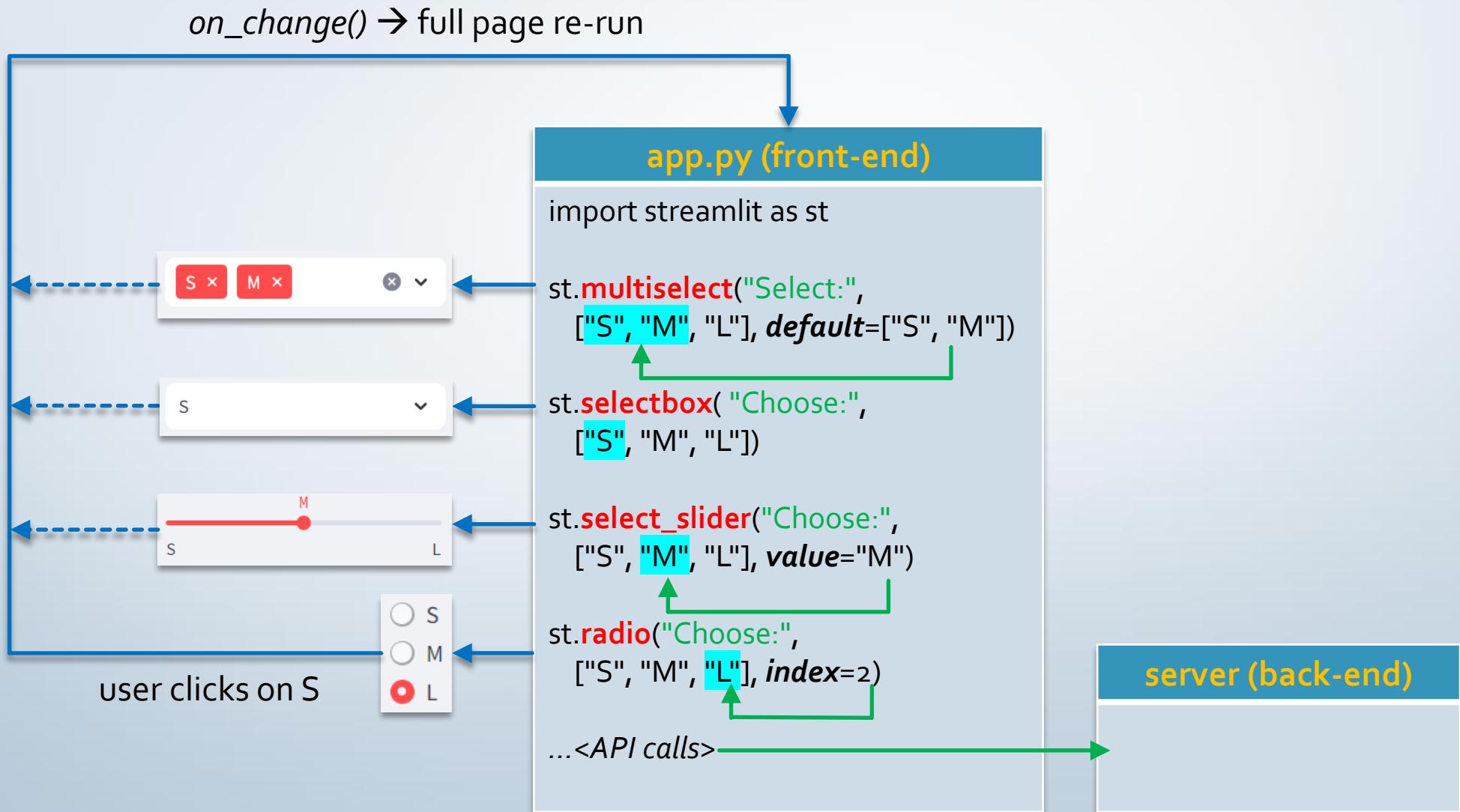
Display Text

- st.**write**('Most objects'), st.**write**(['st', 'is <', 3])
 - st.**text**('Fixed width text')
 - st.**title/header/subheader/caption**('My title')
-
- st.**code**('for i in range(8): foo()') ← source code (w/ optional line numbers)
 - st.**markdown**('_Markdown_') ← write markdown
 - st.**latex**(r''' $e^{i\pi} + 1 = 0$ ''') ← write formulas
-
- st.**divider** ← ~HR

Interactive Widgets

- st.**text/number/date/time_input**("First name") ← **on_change()** event
- st.**text_area**("Text to translate") ← **on_change()** event
- st.**selectbox**("Pick one", ["cats", "dogs"]) ← **on_change()** event
- st.**multiselect**("Buy", ["milk", "apples", "potatoes"]) ← **on_change()** event
- st.**slider**("Pick a number", 0, 100) ← **on_change()** event
- st.**select_slider**("Pick a size", ["S", "M", "L"]) ← **on_change()** event
- st.**color_picker**("Pick a color") ← **on_change()** event

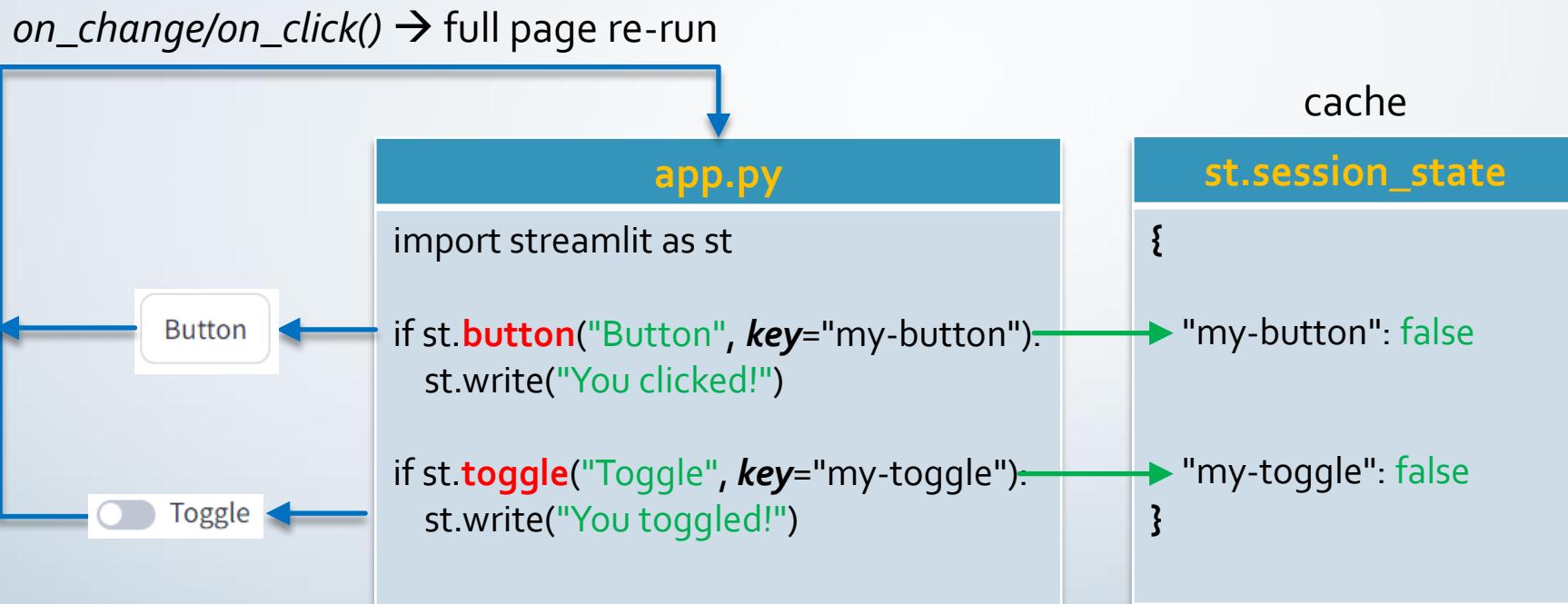
Interactive Widgets



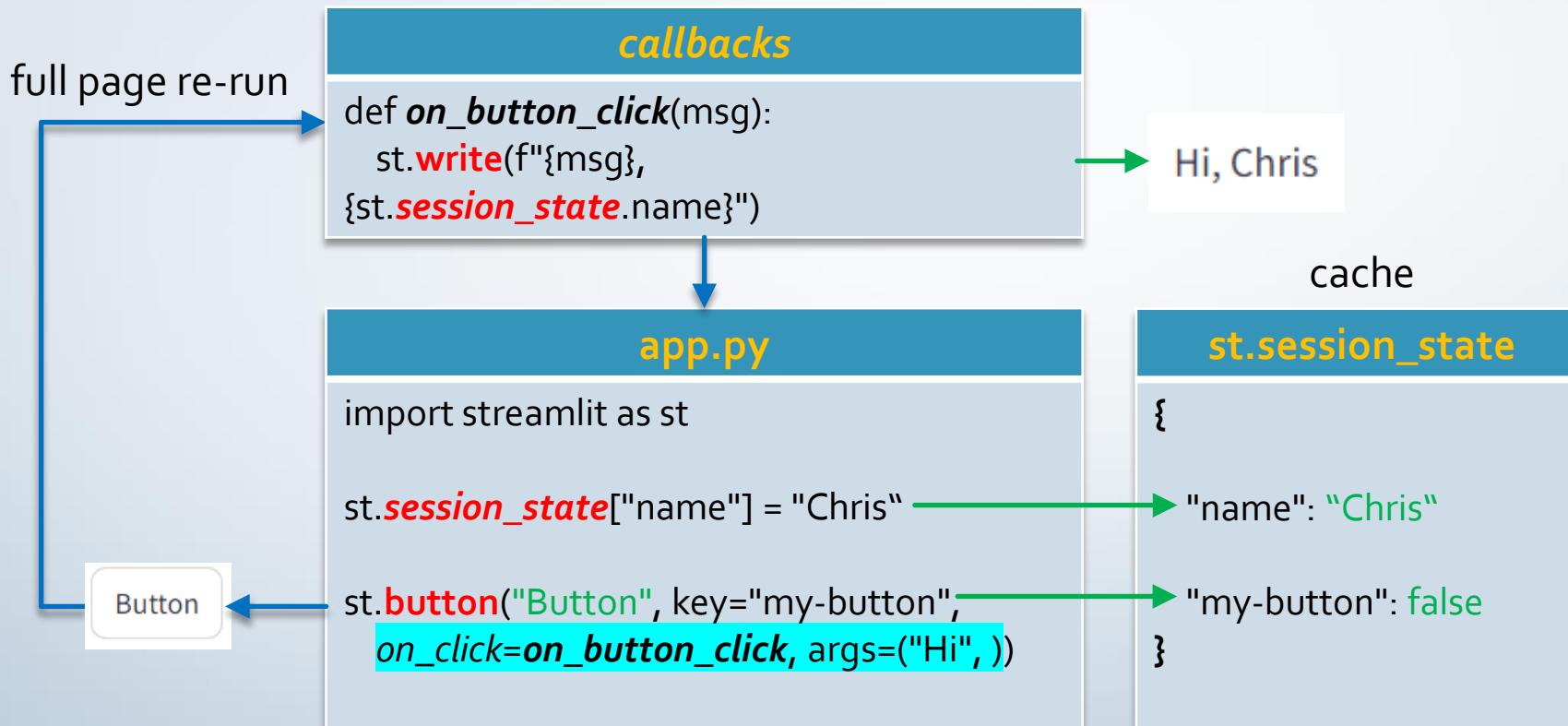
Buttons

- st.button("Click me") ← buttons on single line, `on_click()` event
 - st.toggle("Enable") ← `on_change()` event
 - st.checkbox("I agree") ← `on_change()` event
 - st.radio("Pick one", ["cats", "dogs"]) ← `on_change()` event
-
- st.file_uploader("Upload a CSV") ← `on_change()` event
 - st.download_button("Download file", data) ← `on_click()` event
 - st.link_button("Go to gallery", url) ← no events

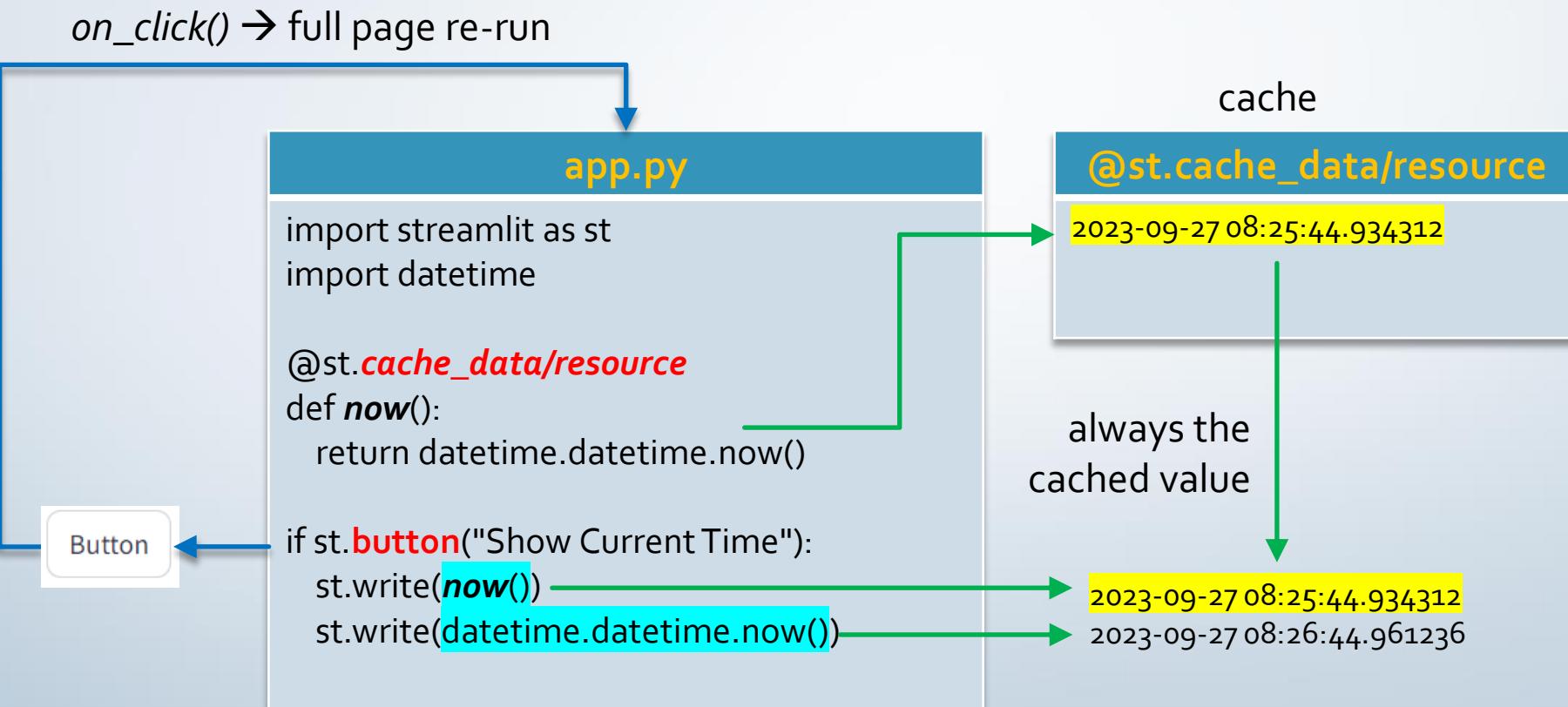
Buttons



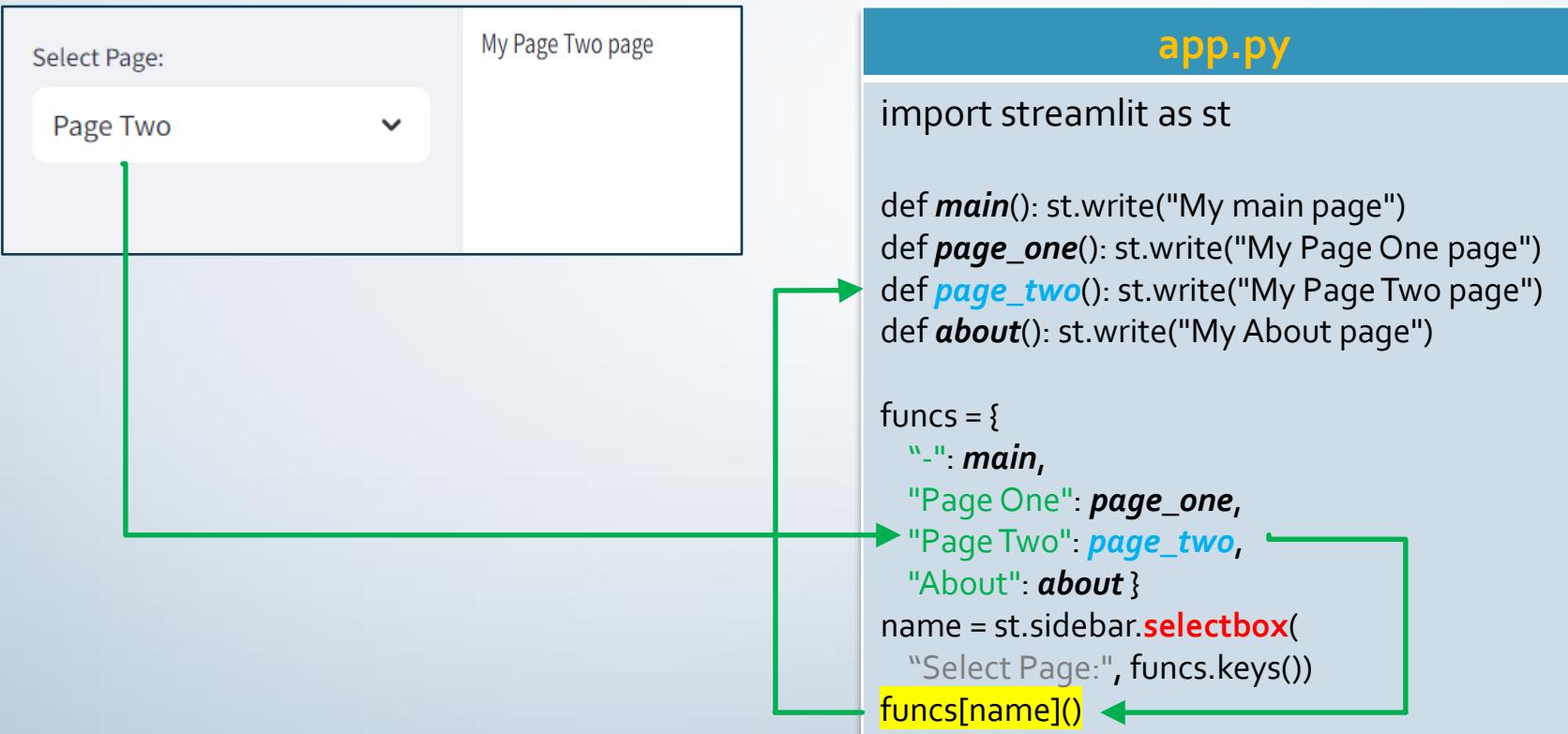
State and Callbacks



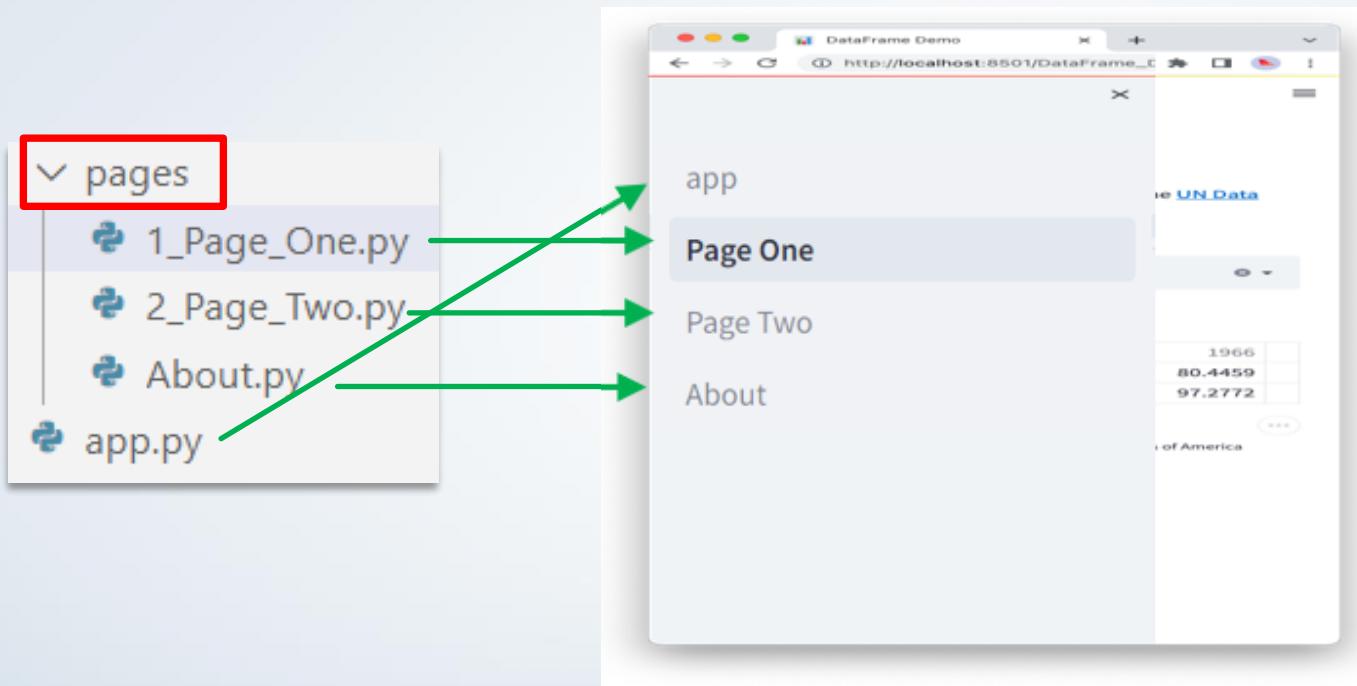
Data Cache



Multi-Page Applications (obsolete now)



Multi-Page Applications



1_Page_One.py

```
import streamlit as st

st.set_page_config(
    page_title="Plotting Demo",
    page_icon="📈")
```

Other Controls

- *display progress/status*
 - with `st.spinner(text='In progress')`: ...
 - `bar = st.progress(50)` ... `bar.progress(100)`
 - with `st.status('Authenticating...')` as `s`: ... `s.update(label='Response')`
 - `st.error/warning/info/success/toast('Error message')`
 - `st.exception(e)`
 - `st.balloons/snow()`
- *media*
 - `st.image` ← show image/list of images
 - `st.audio/video` ← show audio/video player
 - `st.camera_input("Take a picture")` ← `on_change()` event
- *chat*
 - with `st.chat_message("user")`: ... ← response to a chat message
 - `st.chat_input("Say something")` ← prompt chat widget, `on_submit()` event

Data Rendering

- st.**dataframe** ← w/ dataframes from Pandas, PyArrow, Snowpark, PySpark
- st.**table** ← show static table
- st.**data_editor** ← show widget, `on_change()` event
- st.**json** ← show pretty-printed JSON string

Charts

- st.area/bar/line/scatter_chart(df)
- st.map(df) ← geo map w/ scatterplot
- st.graphviz_chart(fig)
- st.altair_chart(chart)
- st.bokeh_chart(fig)
- st.plotly_chart(fig) ← interactive Plotly chart
- st.pydeck_chart(chart) ← free maps
- st.pyplot(fig) ← w/ matplotlib
- st.vega_lite_chart(df)
- st.column_config ← insert spark lines!
- st.metric ← show perf metric number in large

Deploy your Web App to Streamlit Cloud

- *publish your app into GitHub*
 - Streamlit will create access keys! → need authorization
 - your app will automatically refresh on each new GitHub push
 - can later add "Open in Streamlit" button in GitHub
- *deploying in Streamlit Cloud* → for free, if public!
 - sign-up with your Google email at share.streamlit.io
 - make sure your app can be shared publicly → see limits! use subdomain
 - replace any `app\myapp.py` to `app/myapp.py`, if from subfolder (\ → /)
 - prefix with `os.path.dirname(__file__)}{` any relative file names
 - make sure `requirements.txt` is updated → check black sidebar log
 - add any passwords or confidential data as Secrets (see *Advanced Options*)
 - make sure you'll run the same version of Python when deployed
 - can later add the link in Medium posts → expanded as gadget



Client Request

It looks great! And we're now approved to go ahead and implement the Hierarchical Data Viewer for our employee-manager data already loaded in Snowflake!

Your web app should now connect by default to our employee table from Snowflake instead. And show also hierarchical SQL query results with path, as you demonstrated not long ago. Make it generic as well.

Review of Client Request

It looks great! And we're now approved to go ahead and implement the Hierarchical Data Viewer for our employee-manager data already loaded in Snowflake!

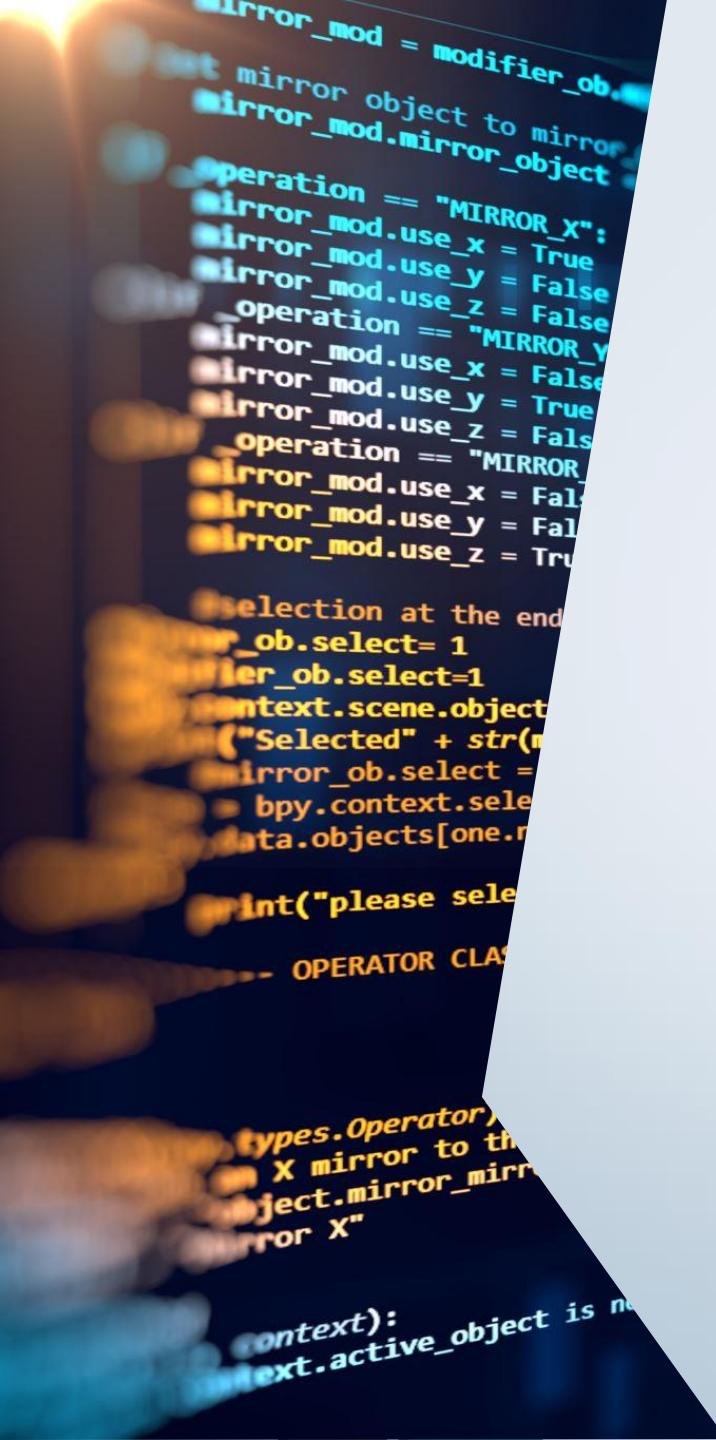
Your web app should now connect by default to our employee table from Snowflake instead. And show also hierarchical SQL query results with path, as you demonstrated not long ago. Make it generic as well.



Section Summary



- SnowCD (Connectivity Diagnostic)
- SnowSQL (Snowflake's CLI)
- Visual Studio Code Extension
- Client Drivers
- Client-Side vs Server-Side Cursors
- Snowflake Connector for Python
- Connecting to Snowflake
- Snowflake Connectors for .NET and NodeJS



SnowCD (Connectivity Diagnostic Tool)

```
select system$allowlist() -- former system$whitelist()  
-- copy JSON result into allowlist.json file
```

```
[  
  {  
    "host": "btb76003.snowflakecomputing.com",  
    "port": 443,  
    "type": "SNOWFLAKE_DEPLOYMENT"  
  },  
  {  
    "host": "ikotkai-ydb73888.snowflakecomputing.com",  
    "port": 443,  
    "type": "SNOWFLAKE_DEPLOYMENT_REGIONLESS"  
  },  
  {  
    "host": "sfc-prod3-ds1-20-customer-stage.s3.amazonaws.com",  
    "port": 443,  
    "type": "STAGE"  
  },  
]
```

```
C:\Users\crist\Downloads>snowcd allowlist.json  
Performing 42 checks for 16 hosts  
All checks passed
```

SnowCD: Private Links

```
select system$allowlist_privatelink()  
-- copy JSON result into allowlist_privatelink.json file
```

```
{  
    "host": "btb76003.us-west-2.privatelink.snowflakecomputing.com",  
    "port": 443,  
    "type": "SNOWFLAKE_DEPLOYMENT"  
},
```

```
C:\Users\crist\Downloads>snowcd allowlist_privatelink.json  
Performing 34 checks for 12 hosts  
lookup ikotkai-ydb73888.privatelink.snowflakecomputing.com: no such host  
lookup app.us-west-2.privatelink.snowflakecomputing.com: no such host  
lookup app-ikotkai-ydb73888.privatelink.snowflakecomputing.com: no such host  
lookup ocsp.btb76003.us-west-2.privatelink.snowflakecomputing.com: no such host  
lookup btb76003.us-west-2.privatelink.snowflakecomputing.com: no such host  
lookup ocsp.ikotkai-ydb73888.privatelink.snowflakecomputing.com: no such host  
  
Check for 6 hosts failed, display as follow:  
=====  
Host: btb76003.us-west-2.privatelink.snowflakecomputing.com  
Port: 443  
Type: SNOWFLAKE_DEPLOYMENT  
Failed Check: DNS Check  
Error: lookup btb76003.us-west-2.privatelink.snowflakecomputing.com: no such host  
Suggestion: Check your configuration on DNS server
```

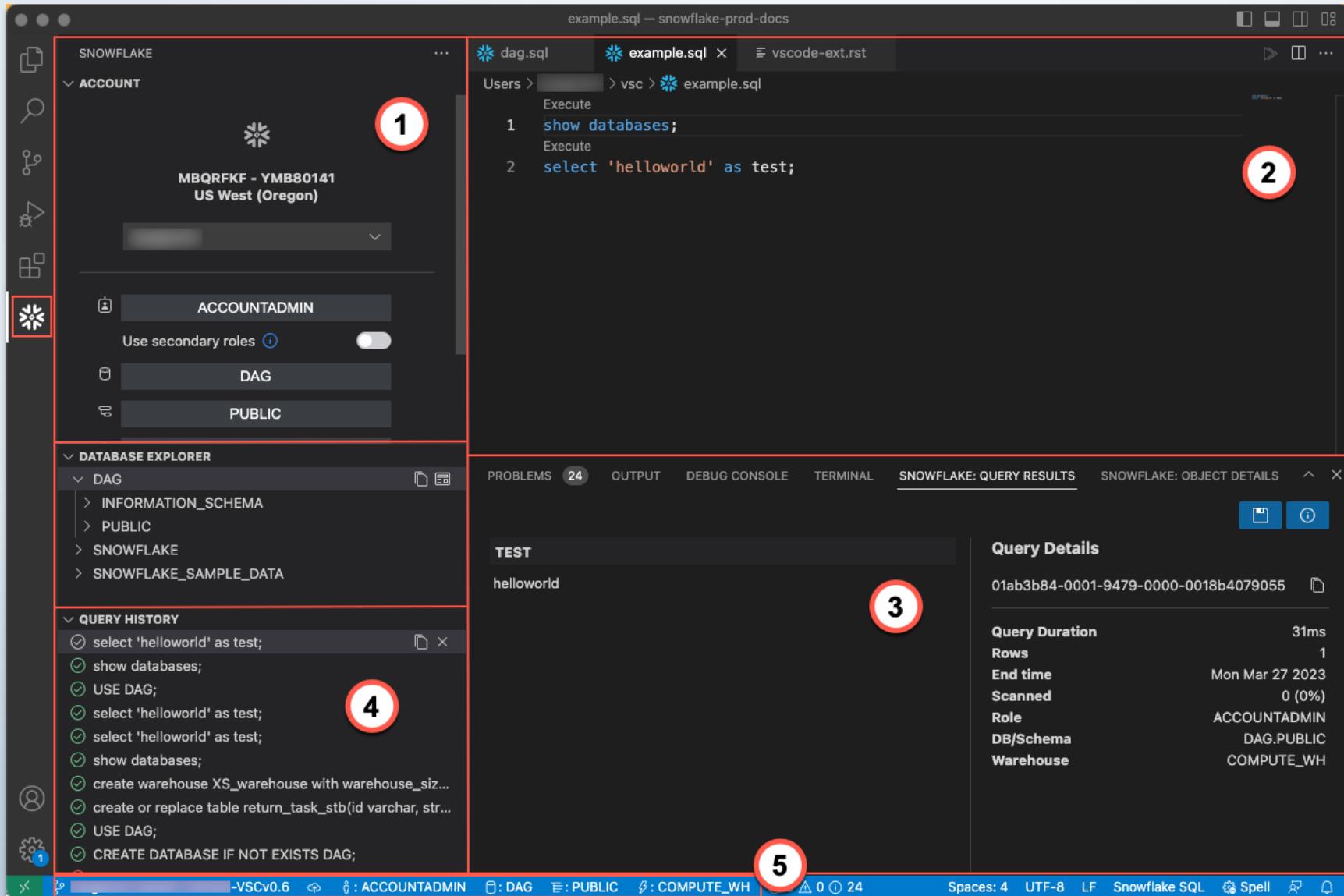
SnowSQL (CLI Client)

- **Overview**
 - Installs as command line tool for Linux/macOS/Windows
 - Developed with *Python Connector for Snowflake*
- **Features**
 - Connect to Snowflake account
 - Perform SQL (DDL/DML) operations
 - Run SQL deploy scripts (including **PUT/GET** for local files)
 - Automate SQL deployments from bash shell through batch scripts
 - Abort/pause running queries
- **Hidden Gems**
 - **~/.snowsql/config** file may be re-used to connect to Snowflake from client apps
 - Can use **\$SNOWSQL_PWD** env var to save local password
 - Customize scripts through **variable substitution** (for partner databases)
 - Generate **JWT tokens** for key pair authentication (for SQL REST API)

Snowflake Extension for VS Code

- Enable users to write and execute SQL statements directly in VS Code.
- Install from Code > Preferences > Extensions.
- Sign-in to your cloud Snowflake account.
- Can use SnowSQL configuration files.
- Execute commands or queries, from SQL files.
- Query History.
- Query Results pane.
- Database Explorer.
- Upload/download files from stages directly.

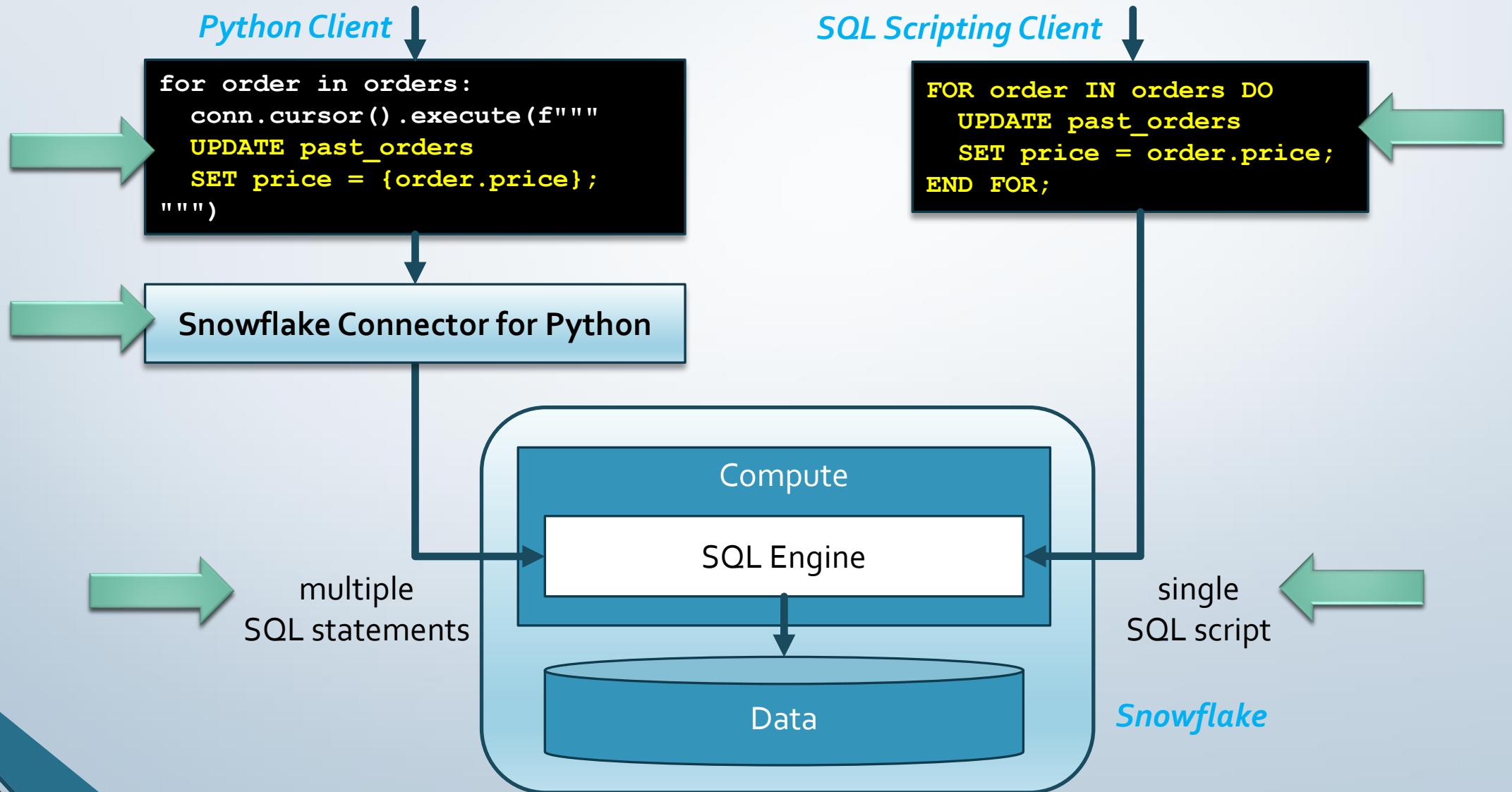
Snowflake Extension for VS Code



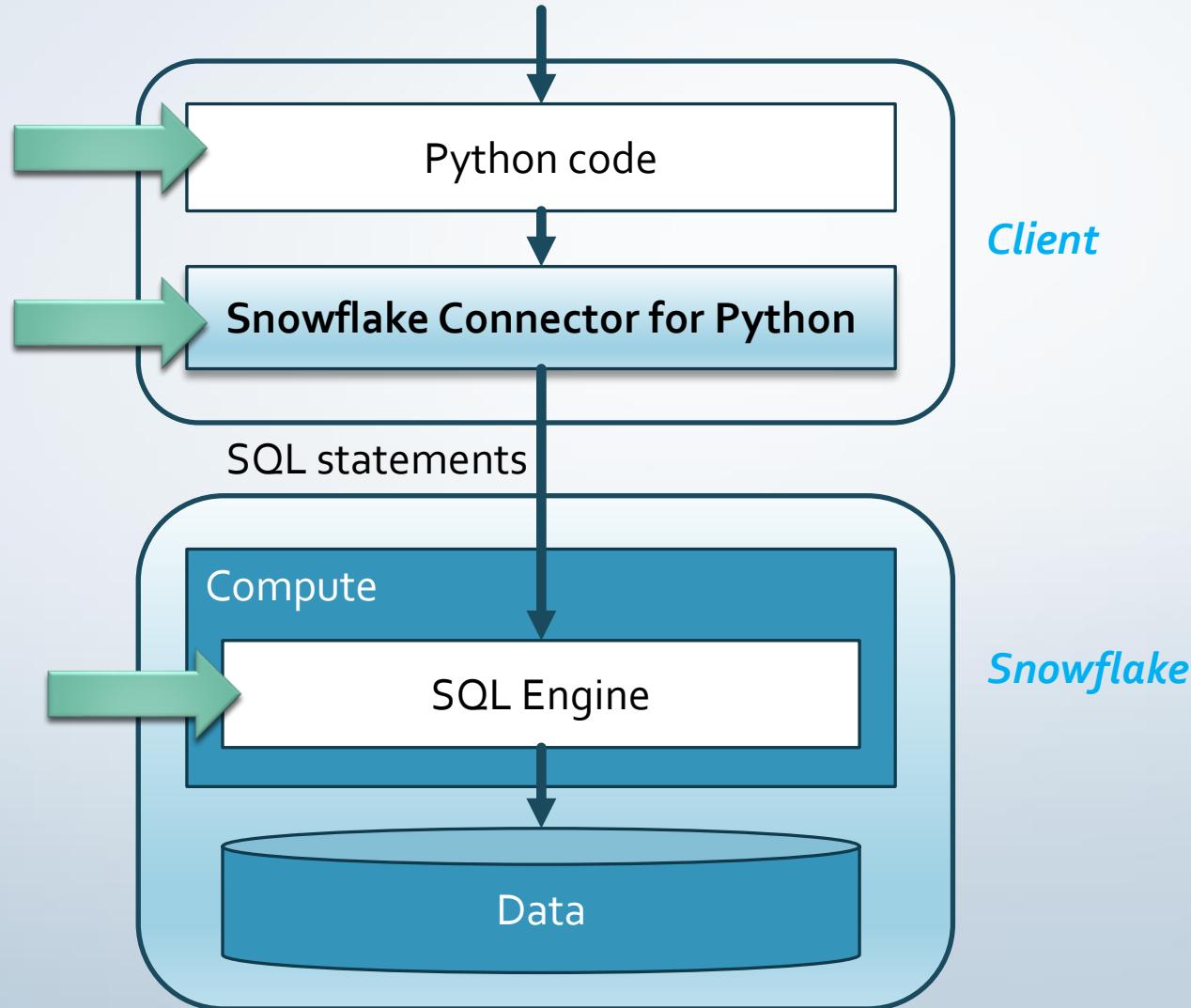
Client Drivers

- **Snowflake Connector for Python**
- **.NET Driver** – for C#
- **Node.js Driver** – for JavaScript
- **Go Snowflake Driver**
- **PHP PDO Driver for Snowflake**
- **JDBC Driver** ← for Java, Scala...
- **ODBC Driver** ← from Windows

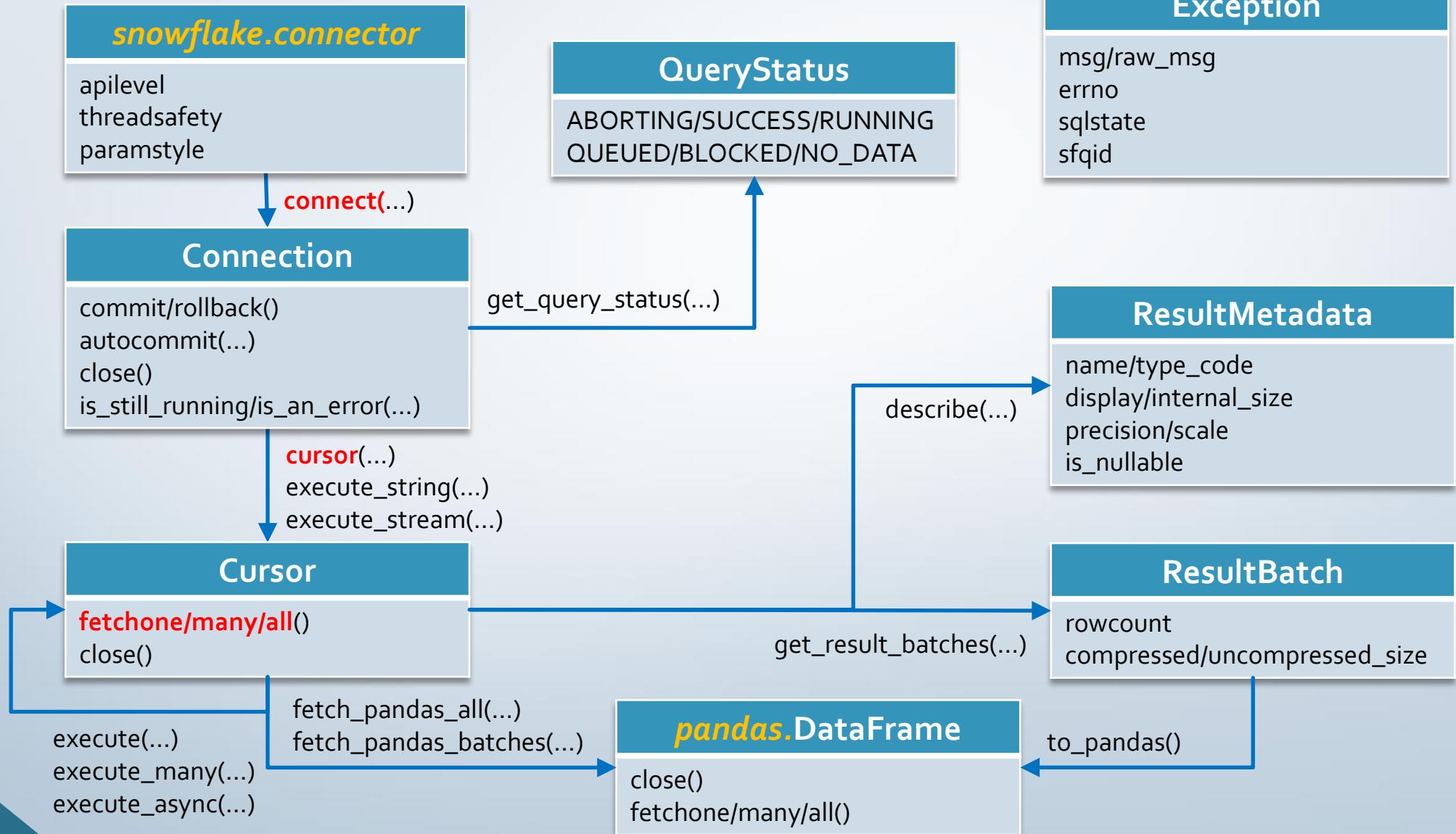
Client-Side vs Server-Side Cursors



Snowflake Connector for Python



Python Connector API



Python Connector: Common Pattern

Python Client

```
import snowflake.connector

# show all property=value pairs for current user
with snowflake.connector.connect(...) as conn:
    with conn.cursor() as cur:
        cur.execute("show parameters")
        for row in cur:
            print(f'{str(row[0])}={str(row[1])}'")
```

PROBLEMS OUTPUT TERMINAL PORTS CODEWHISPERER REFERENCE LOG

```
SNOWFLAKE_LOCK=false
SNOWFLAKE_SUPPORT=false
DAYS_TO_EXPIRY=null
MINS_TO_UNLOCK=null
DEFAULT_WAREHOUSE=COMPUTE_WH
DEFAULT_NAMESPACE=null
DEFAULT_ROLE=ACCOUNTADMIN
DEFAULT_SECONDARY_ROLES=null
EXT_AUTHN_DUO=false
EXT_AUTHN_UID=null
```

Connecting to Snowflake

Basic (Username/Password)

```
conn = snowflake.connector.connect(  
    account = ...,  
    user = ...,  
    role = ...,  
    database = ...,  
    schema = ...,  
    warehouse = ...,  
    password = os.getenv('SNOWSQL_PWD'))
```

SSO

```
conn = snowflake.connector.connect(  
    account = ...,  
    user = ...,  
    role = ...,  
    database = ...,  
    schema = ...,  
    warehouse = ...,  
    authenticator = "externalbrowser")
```

Key Pair

```
with open(f"{Path.home()}/.ssh/id_rsa", "rb") as key:  
    p_key= serialization.load_pem_private_key(  
        key.read(),  
        password = os.environ['SNOWSQL_PWD'].encode(),  
        backend = default_backend())  
  
    pkb = p_key.private_bytes(  
        encoding = serialization.Encoding.DER,  
        format = serialization.PrivateFormat.PKCS8,  
        encryption_algorithm = \  
            serialization.NoEncryption())  
  
conn = snowflake.connector.connect(  
    account = ...,  
    user = ...,  
    role = ...,  
    database = ...,  
    schema = ...,  
    warehouse = ...,  
    private_key = pkb)
```

Snowflake Connector for .NET

- Create new **Console App (.NET Framework)** C# project in free Visual Studio CE IDE
- Add **Snowflake.Data** NuGet package and ref w/ “**using Snowflake.Data.Client;**”

C# Client Code

```
var user = "cristiscu";
var pwd = Environment.GetEnvironmentVariable("SNOWSQL_PWD");
var connStr = $"account=BTB76003;user={user};password={pwd}";
using (var conn = new SnowflakeDbConnection(connStr)) {
    conn.Open();
    using (var cmd = conn.CreateCommand()) {
        cmd.CommandText = "show parameters";
        using (var reader = cmd.ExecuteReader())
            while (reader.Read())
                Console.WriteLine($"{reader[0]}={reader[1]}");
    }
    conn.Close();
}
```

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His right hand holds a pen, ready to write. The background behind him is a light grey.

Client Request

We have some data scientists familiar with Python, but not with SQL. Is it possible to write and deploy queries in Python only, the way we do it with Pandas DataFrame?

What about some other functionality, to write it in pure Python and transparently deploy it into Snowflake?

What if we need to load additional third-party components? Or access intermediate resources?

Review of Client Request

We have some data scientists familiar with Python, but not with SQL. Is it possible to write and deploy queries in Python only, the way we do it with Pandas DataFrame?

What about some other functionality, to write it in pure Python and transparently deploy it into Snowflake?

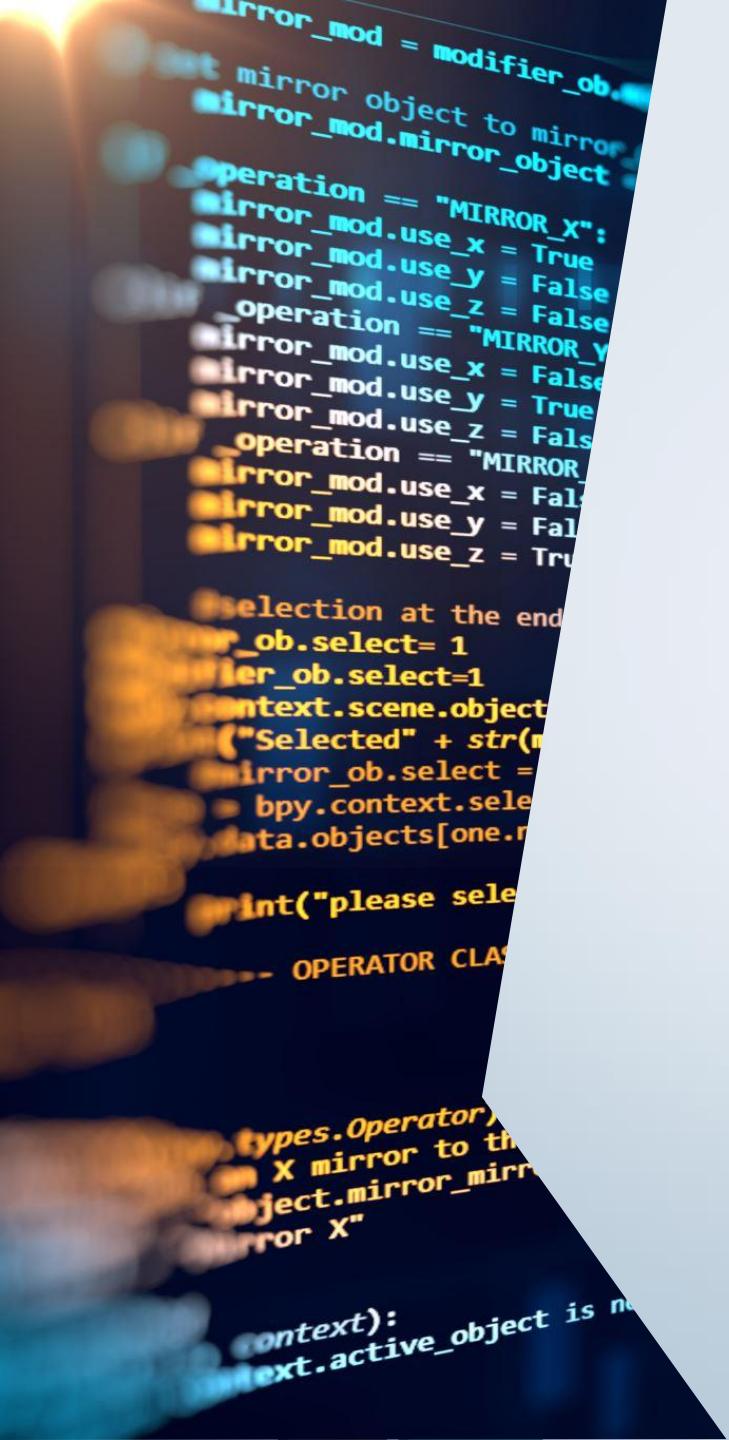
What if we need to load additional third-party components? Or access intermediate resources?



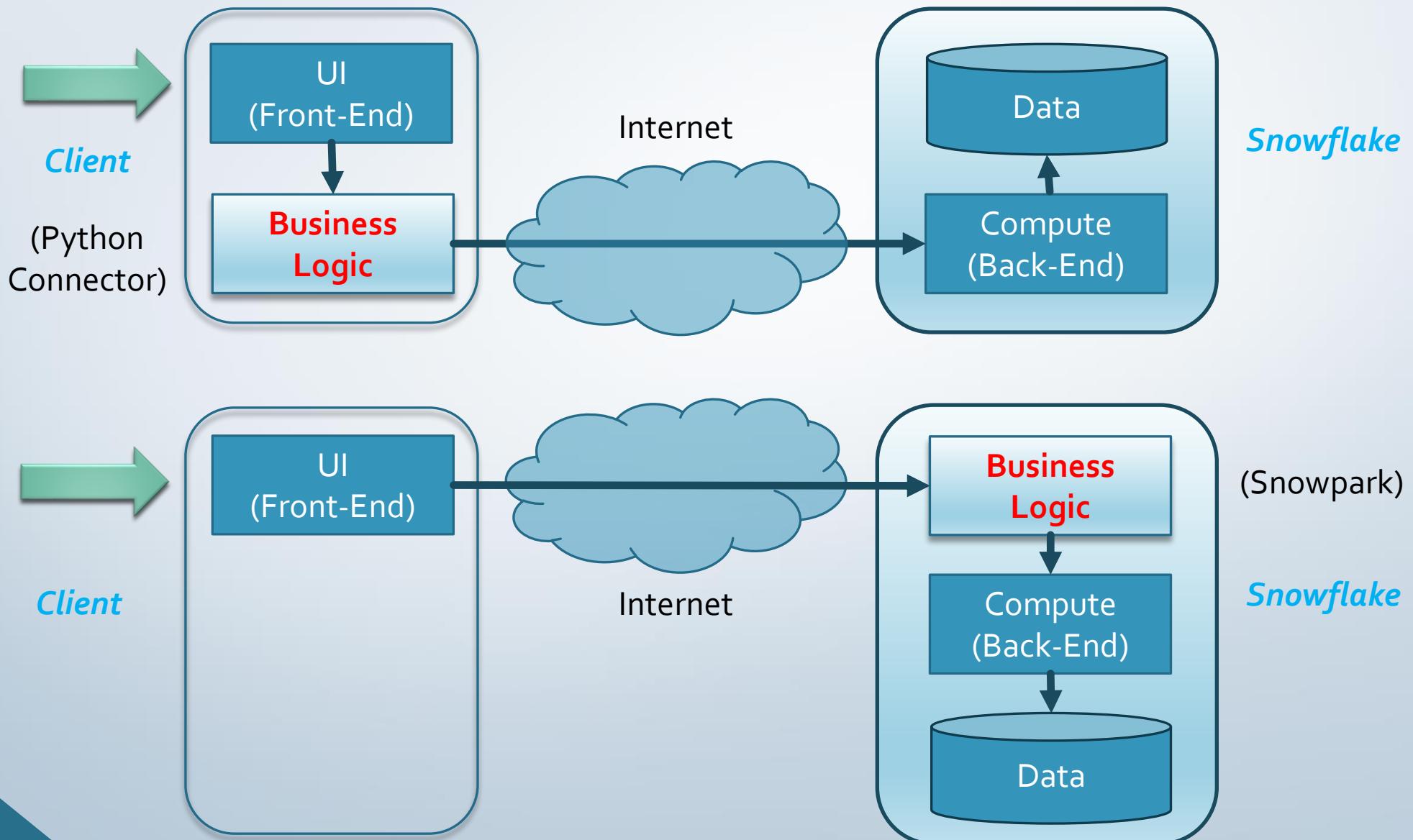
Section Summary



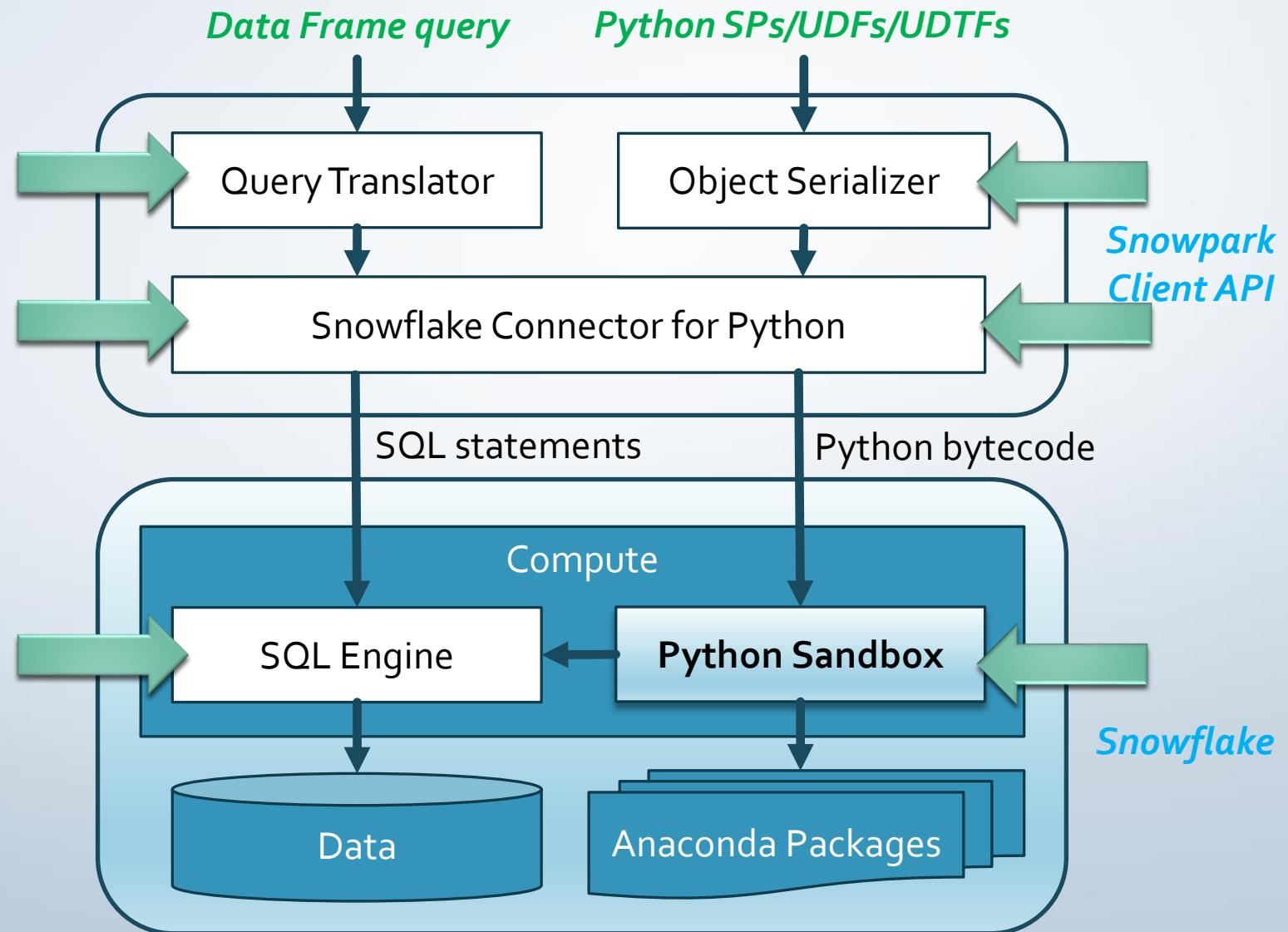
- Client vs Server-Side Programming
- Snowpark for Python Architecture
- Snowpark API: The Object Model
- Creating Queries with DataFrame
- SPs/UDFs/UDTFs in Python/Java/Scala
- Loading Additional Components
- Python Worksheets



Client-Side vs Server-Side Programming



Snowpark for Python



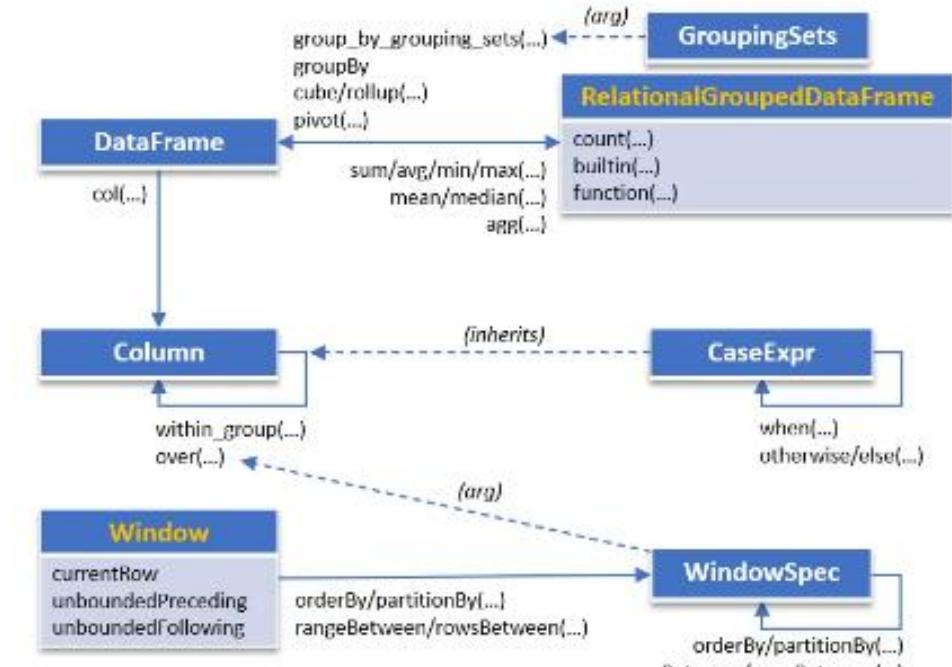
Snowpark API: The Object Model

- Personal Blog Post from Jul 2023
- Also on the Snowflake DSH Blog
- Reposted by 40+ Snowflake employees on LinkedIn
- Visual Class Diagrams
- Classes grouped by functionality

Snowpark API: The Object Model

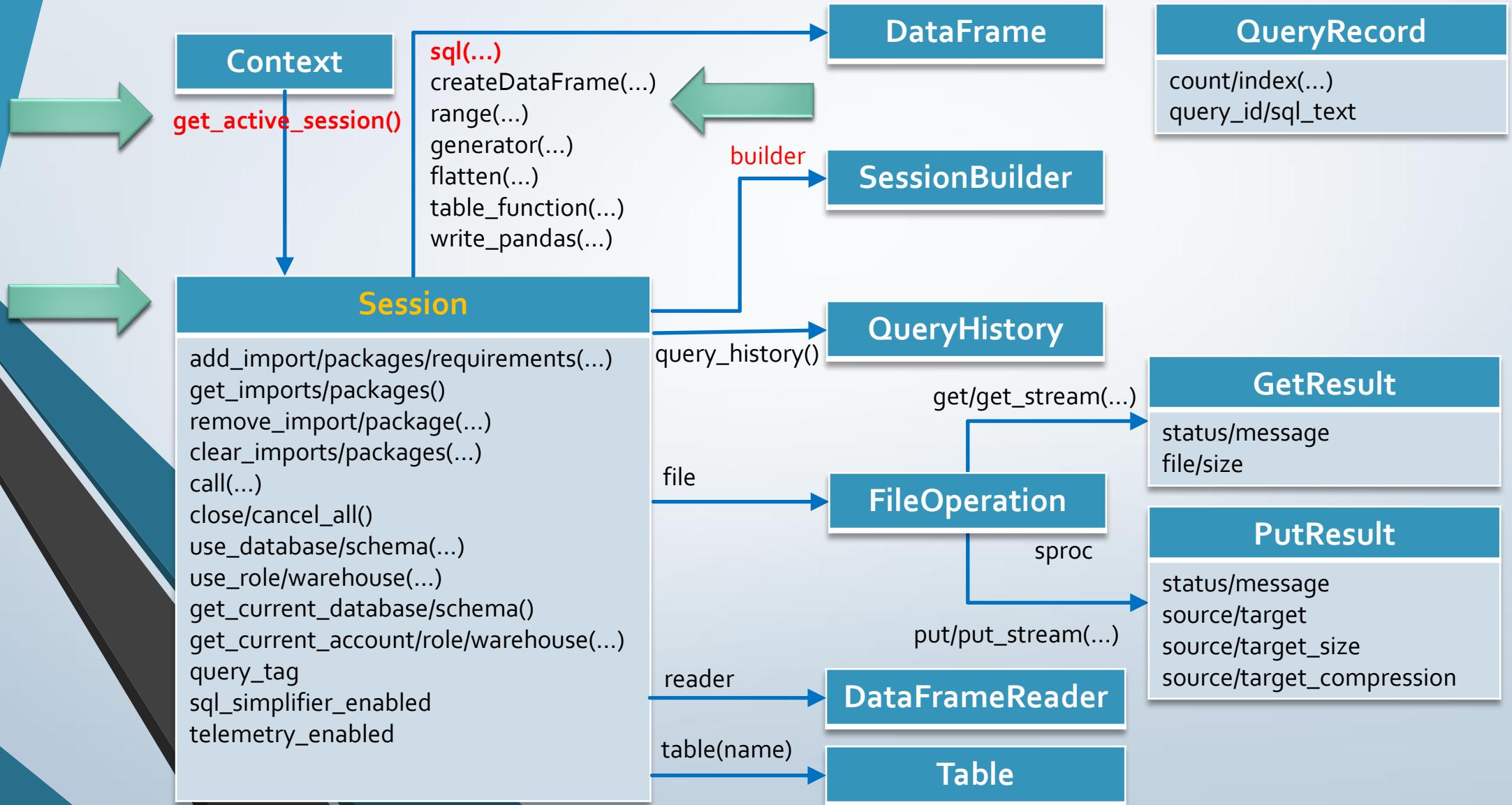
 Cristian Scutaru · Follow
Published In Snowflake · 5 min read · Jul 11

92 2

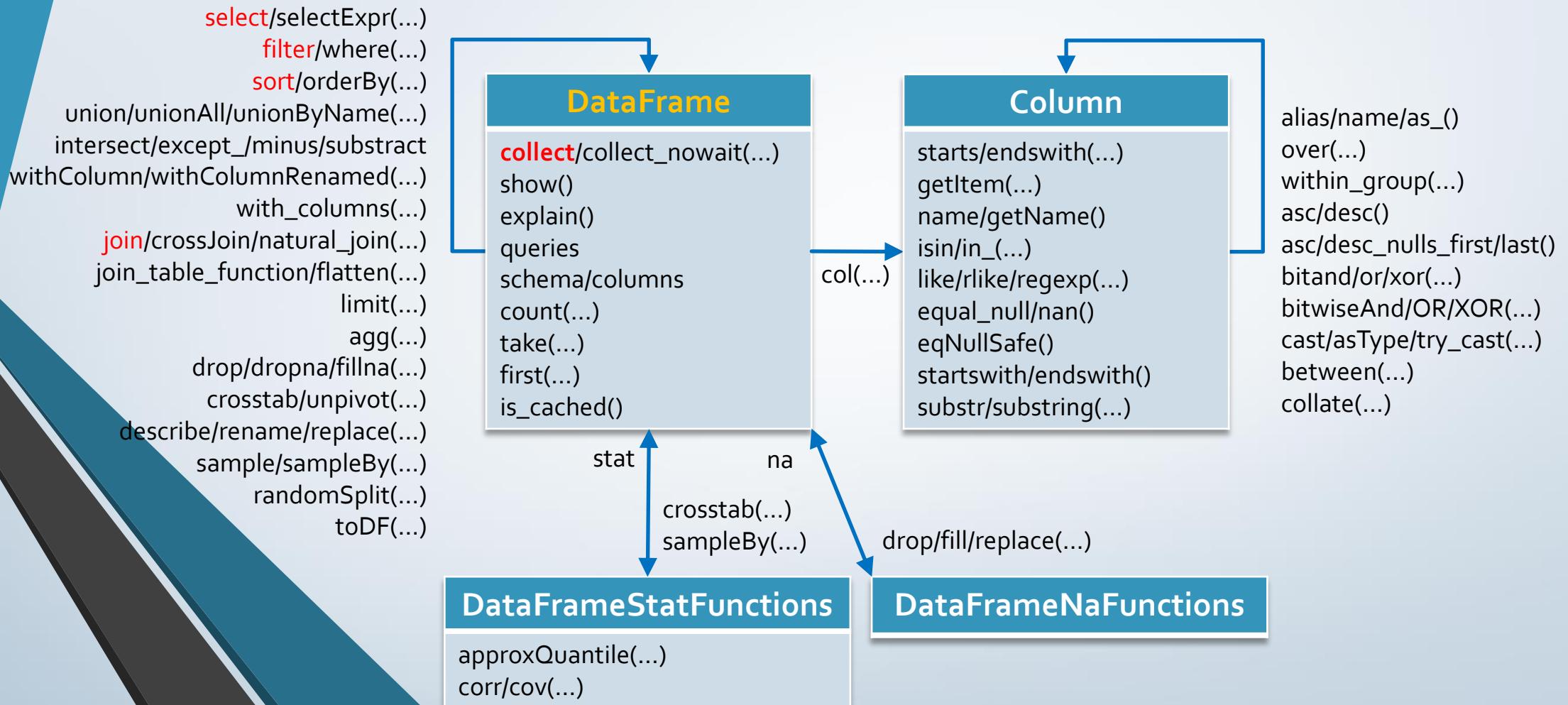


The [Snowpark API Reference for Python](#) looks great. But when you have dozens of classes with hundreds of methods and properties, some class diagrams for the whole object model will help as well.

Session Class

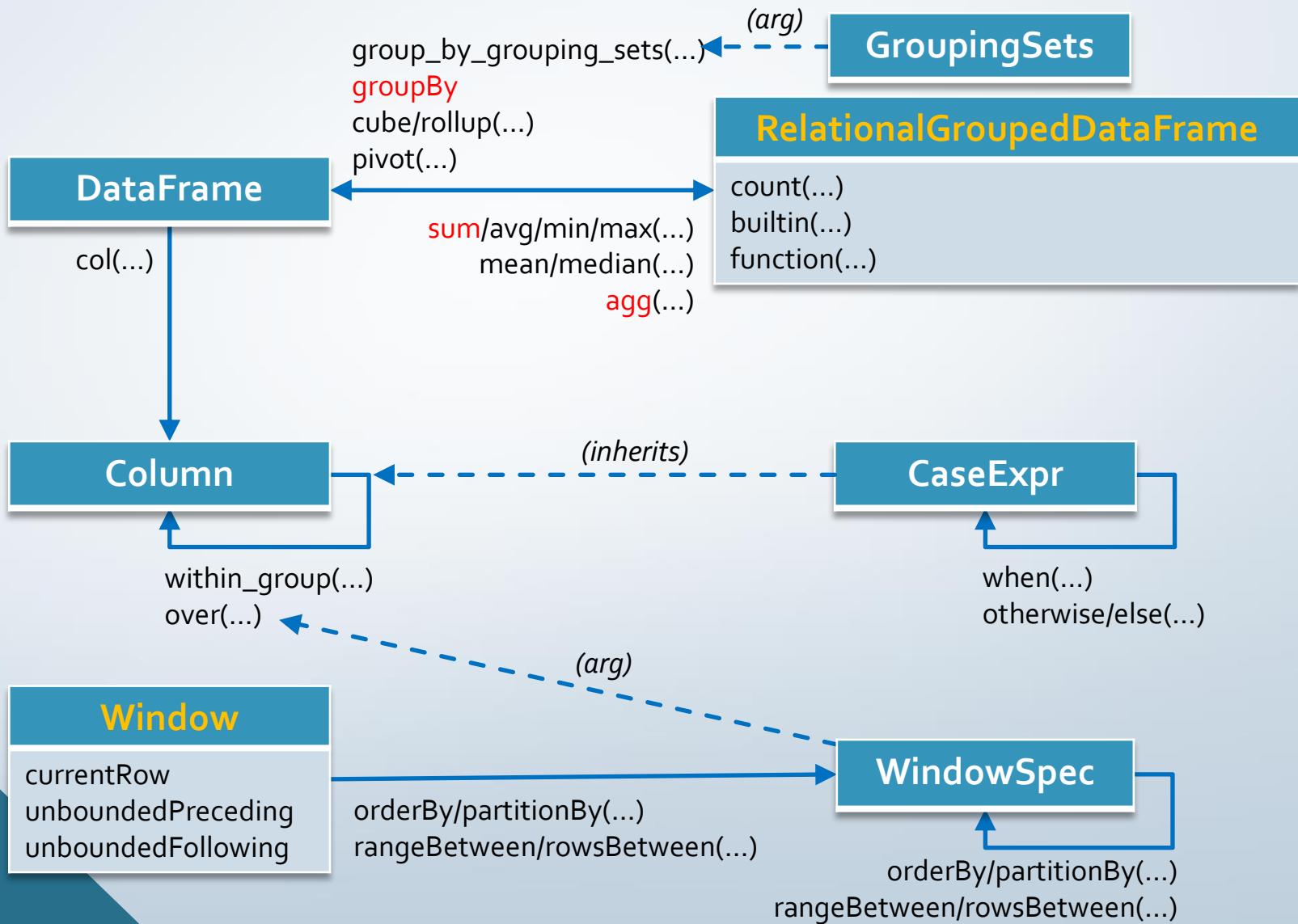


DataFrame Class

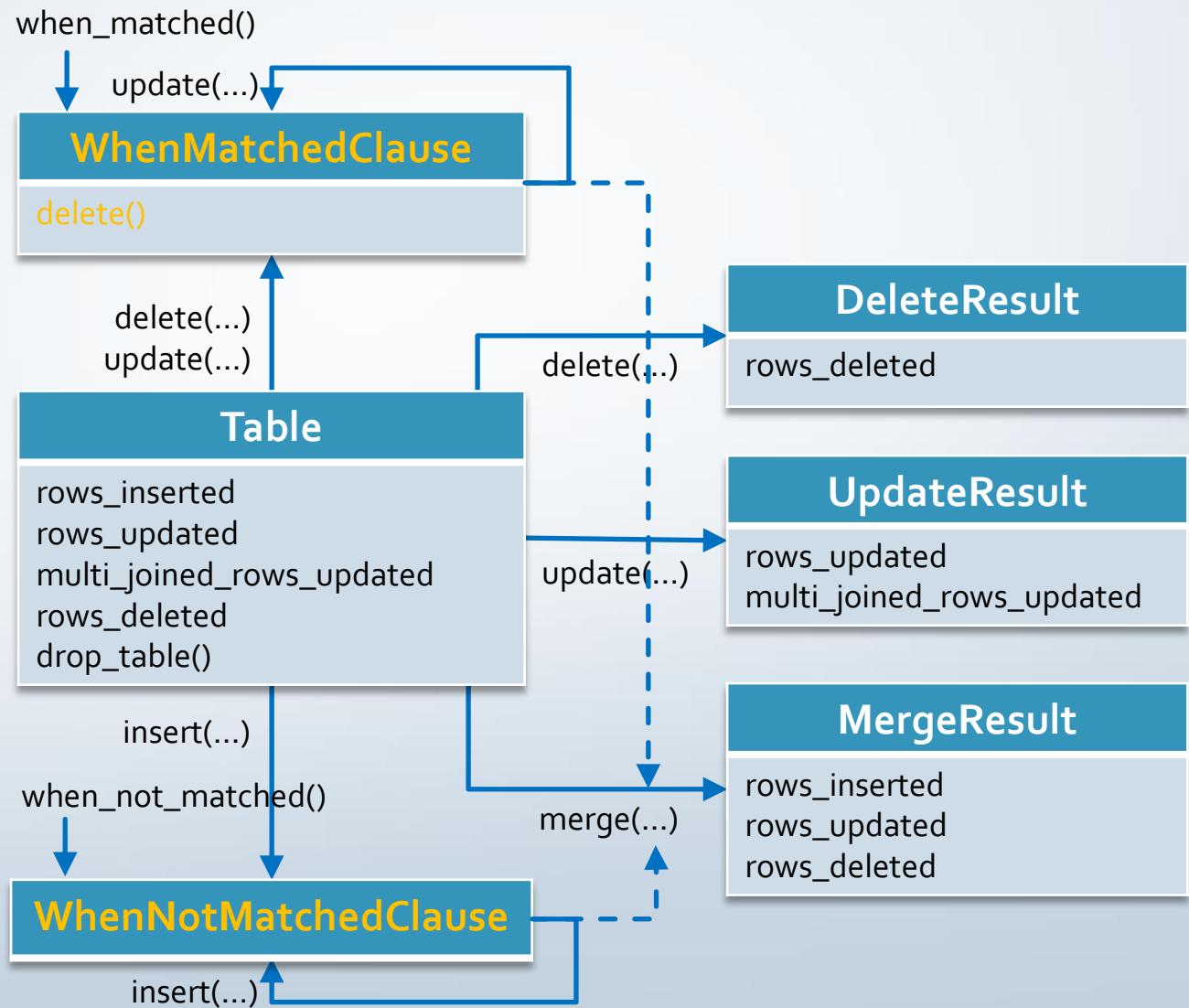


```
rows = get_active_session().sql("SELECT ...").collect()
```

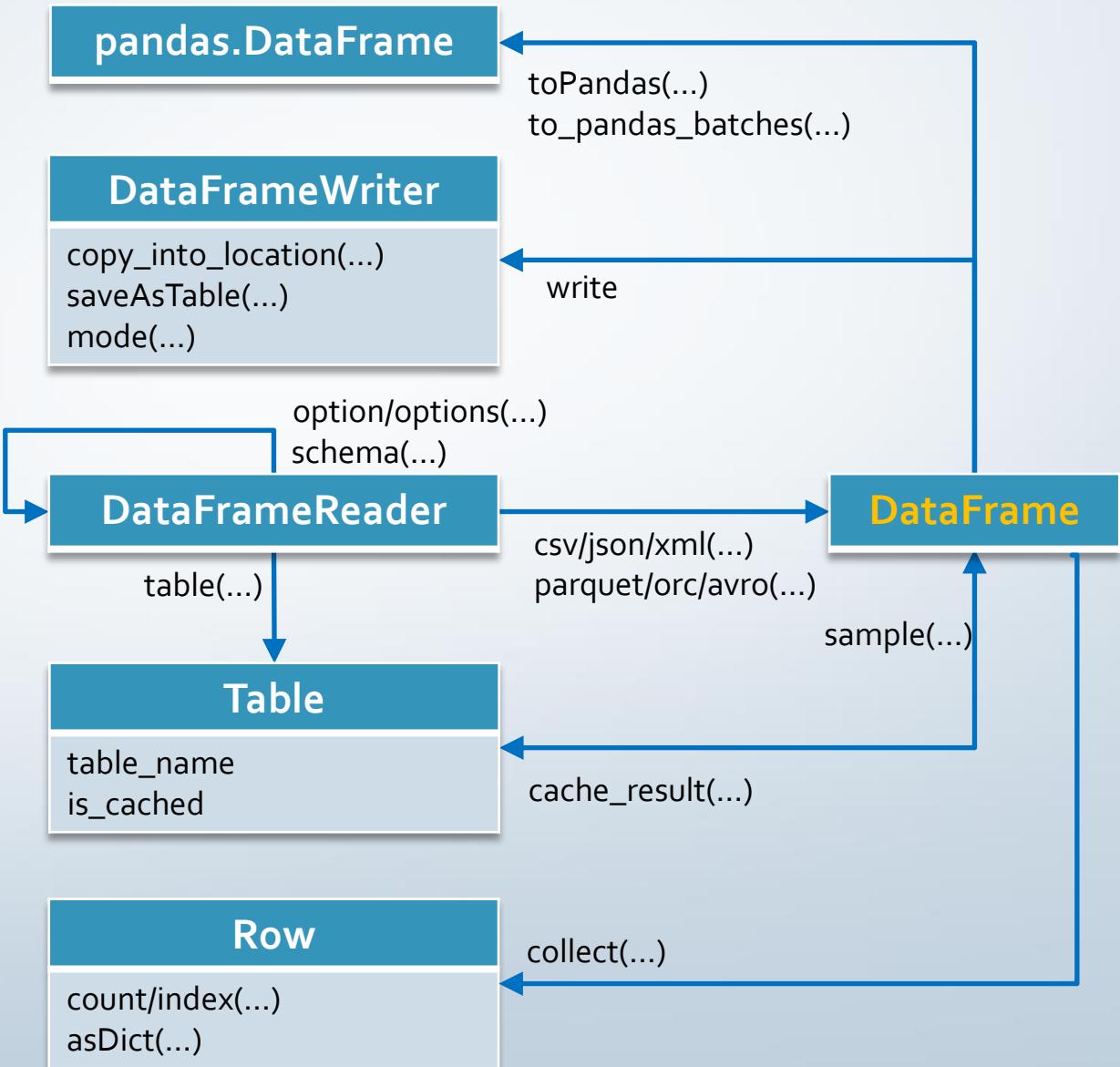
Grouping Functions



MERGE Statement



Input/Output



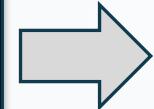
Create Query with DataFrame

Python Code using DataFrame

```
emps = (session.table("EMP")
         .select("DEPTNO", "SAL"))
depts = (session.table("DEPT")
         .select("DEPTNO", "DNAME"))
q = emps.join(depts,
               emps.deptno == depts.deptno)

q = q.filter(q.dname != 'RESEARCH')
q = (q.select("DNAME", "SAL")
      .group_by("DNAME")
      .agg({"SAL": "sum"})
      .sort("DNAME"))

q.show()
```

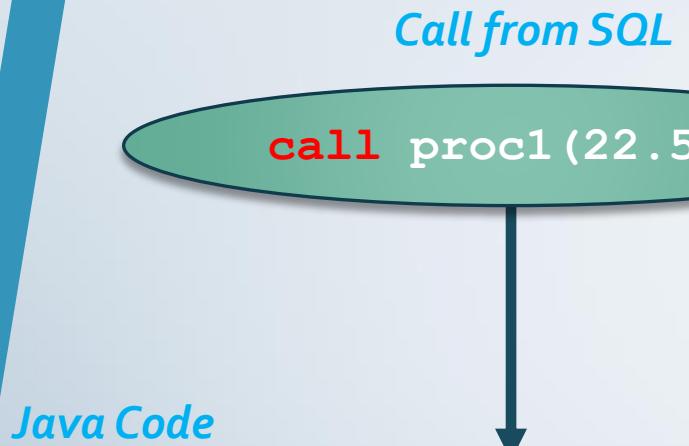


Generated SQL

```
select dname, sum(sal)
from emp join dept
on emp.deptno = dept.deptno
where dname <> 'RESEARCH'
group by dname
order by dname;
```

DNAME	SUM(SAL)
ACCOUNTING	8,750.5
SALES	9,400

Snowpark Stored Procedures



Python Code

```
create procedure proc1(num float)
returns string
language python
runtime_version = '3.8'
packages = ('snowflake-snowpark-python')
handler = 'proc1'
as $$

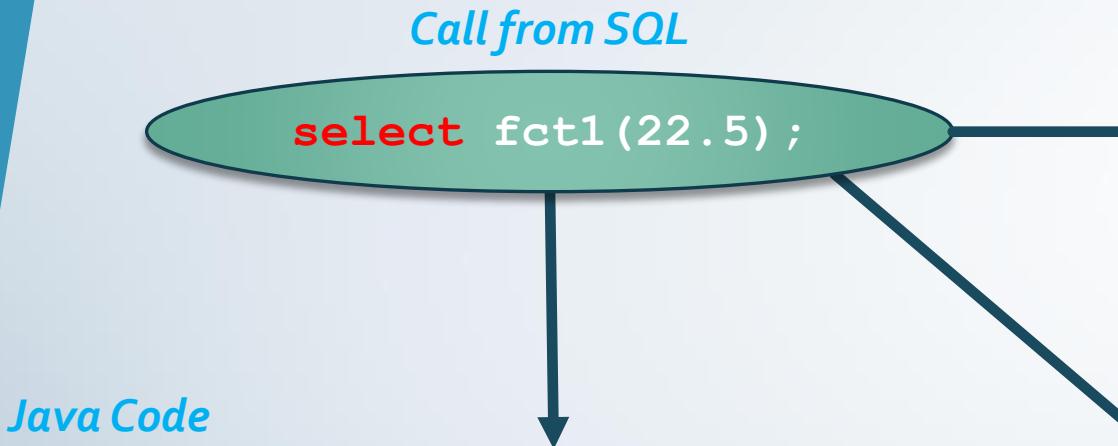
import snowflake.snowpark as snowpark
def proc1(sess: snowpark.Session, num: float):
    return '+' + str(num)
$$;
```

Scala Code

```
create procedure proc1(num float)
returns string
language scala
runtime_version = 2.12
packages = ('com.snowflake:snowpark:latest')
handler = 'MyClass.proc1'
as $$

import com.snowflake.snowpark.Session;
object MyClass {
    def proc1(sess: Session, num: Float): String = {
        return "+" + num.toString }
}
$$;
```

Snowpark UDFs (User-Defined Functions)



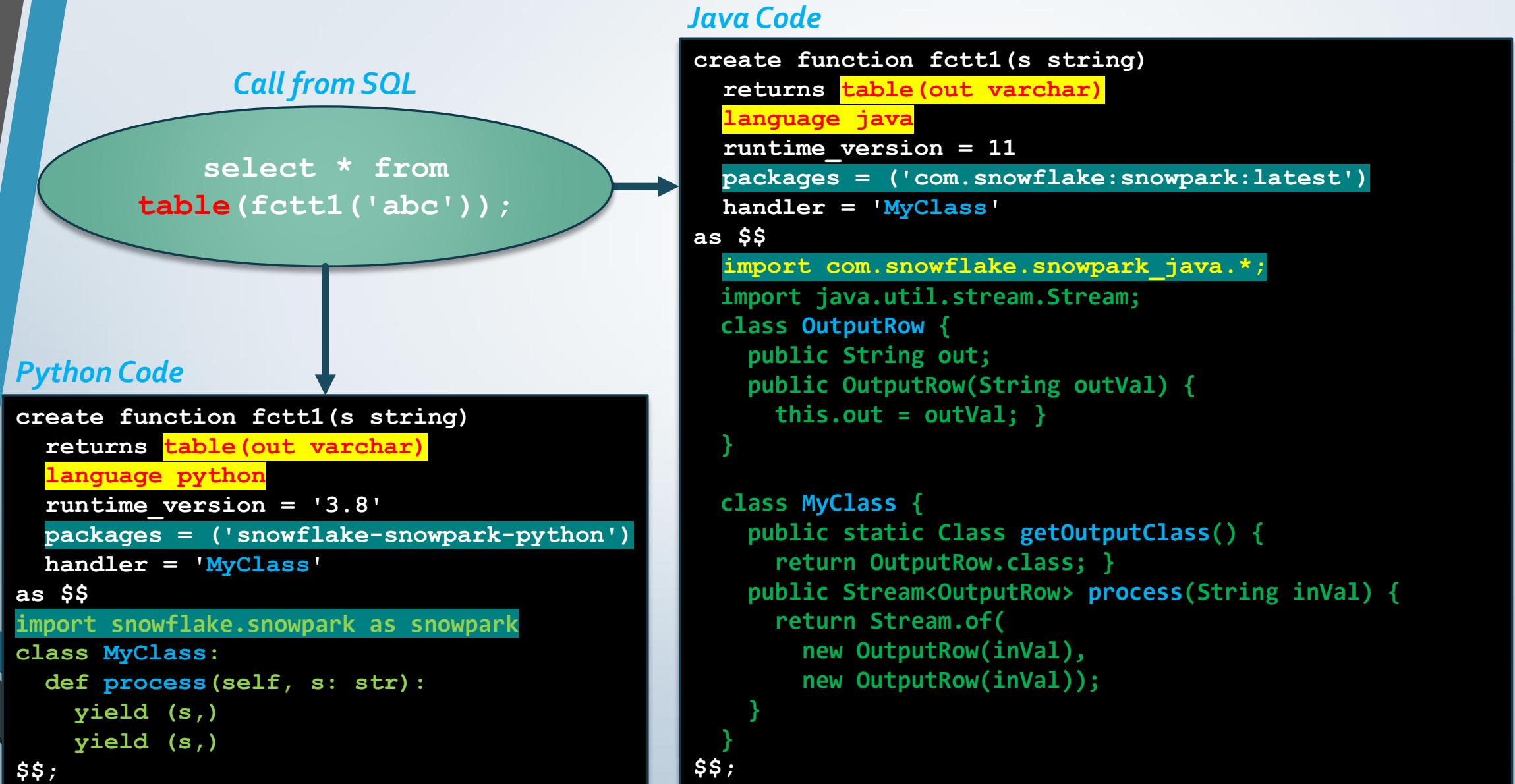
Python Code

```
create function fct1(num float)
    returns string
language python
runtime_version = '3.8'
packages = ('snowflake-snowpark-python')
handler = 'proc1'
as $$
import snowflake.snowpark as snowpark
def proc1(num: float):
    return '+' + str(num)
$$;
```

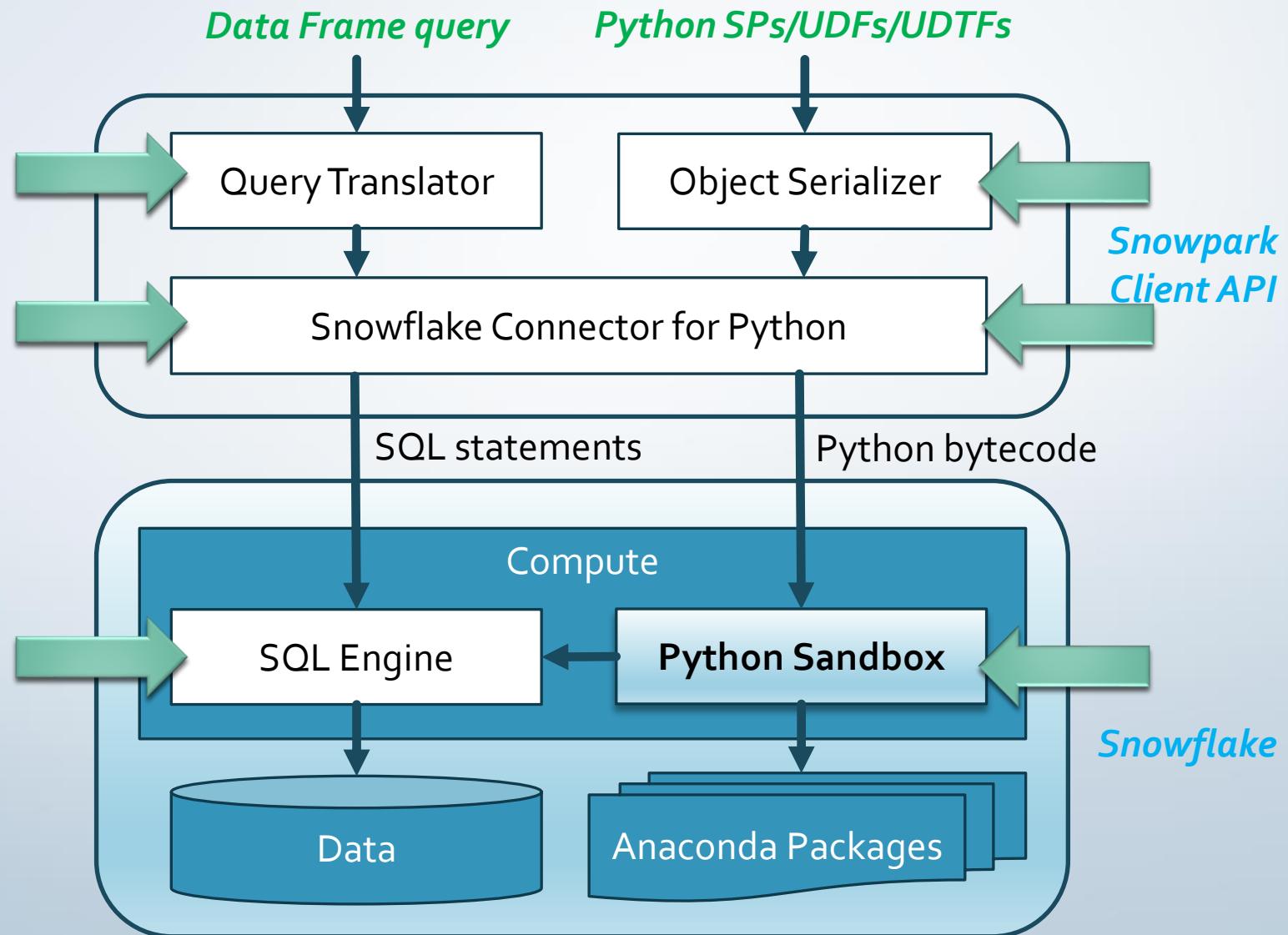
Scala Code

```
create function fct1(num float)
    returns string
language scala
runtime_version = 2.12
packages = ('com.snowflake:snowpark:latest')
handler = 'MyClass.fct1'
as $$
import com.snowflake.snowpark.Session;
object MyClass {
    def fct1(num: Float): String = {
        return "+" + num.toString
    }
}$$;
```

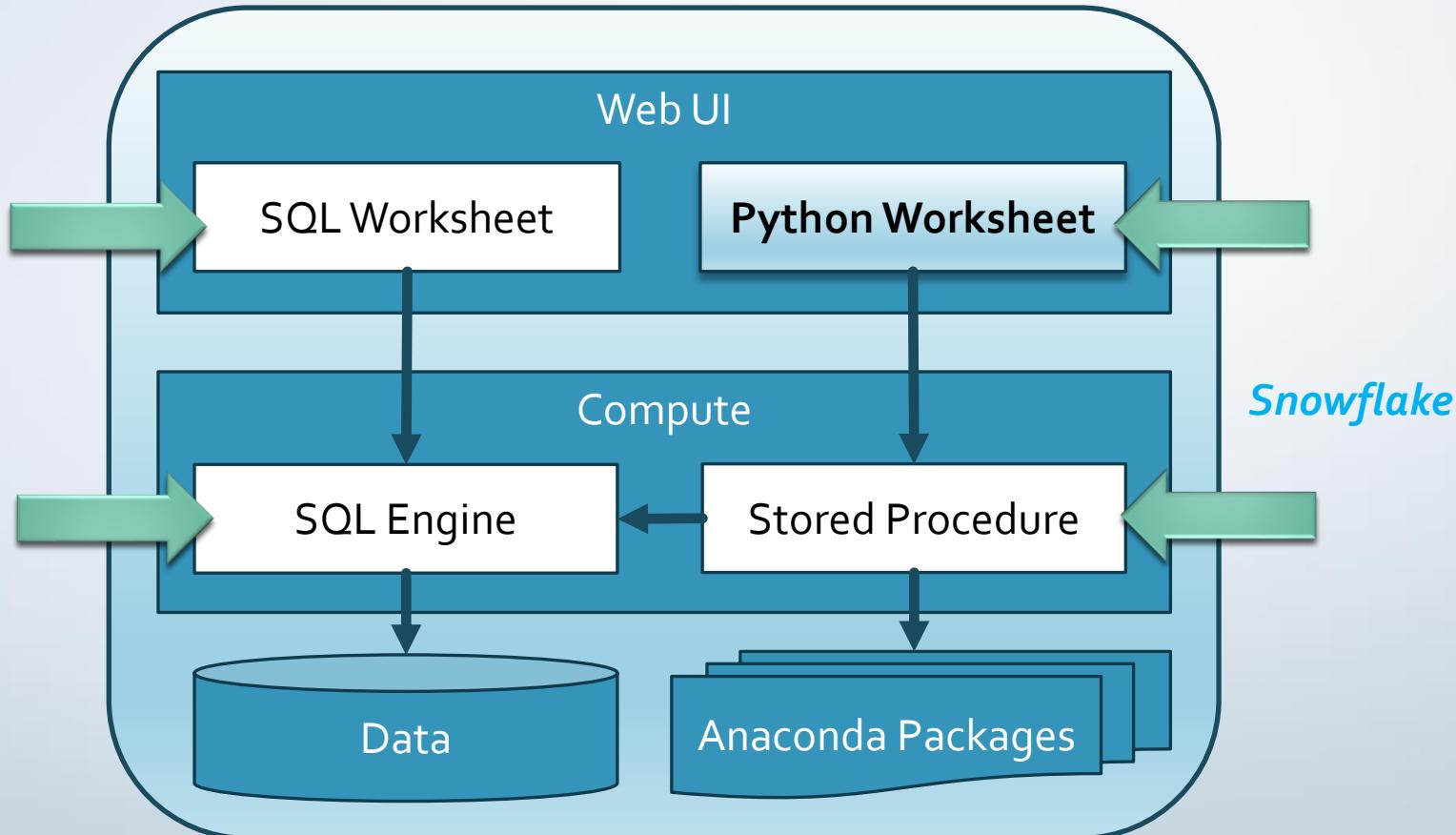
Snowpark UDTFs (User-Defined Table Functions)



Snowpark for Python



Python Worksheets



Functions



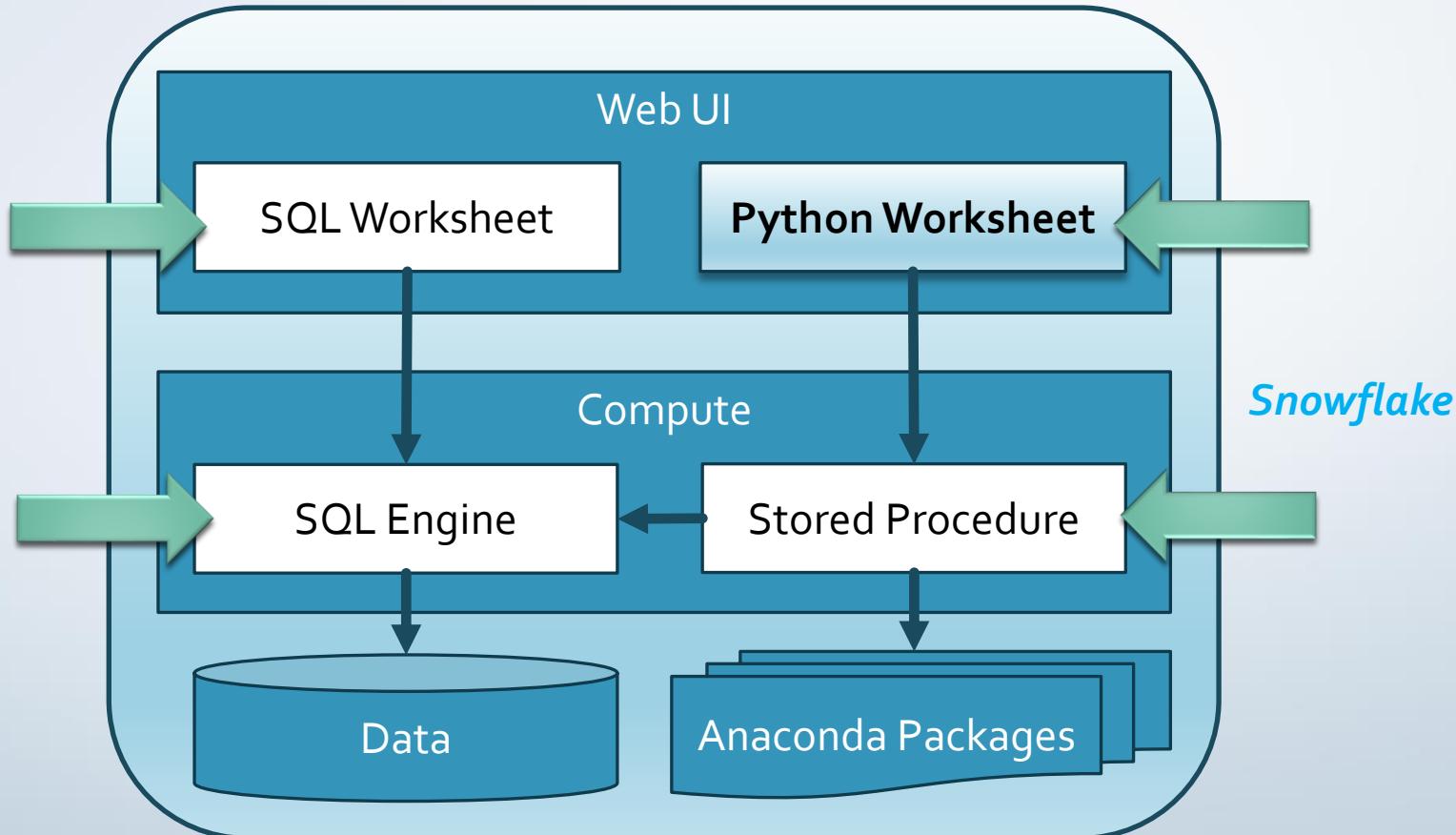
Functions in Snowpark Python

- *creating*
 - `sproc/udf/udtf(lambda: ..., [name="..."], ...)` ← *anonymous/named*
 - `sproc/udf/udtf.register(name="...", is_permanent=True, ...)` ← *registered*
 - `@sproc/@udf/@udtf(name="...", is_permanent=True, ...)` ← *registered*
- *UDTF handler class*
 - `__init__(self)` - optional
 - `process(self, ...)` - required, for each input row → tuples w/ tabular value
 - `end_partition(self)` - optional, to finalize processing of input partitions
- *calling*
 - `name(...)` / `fct = function("name")` ← by name/function pointer
 - `session.call/call_function/call_udf("name", ...)` ← SP/UDF
 - `session.table_function(...)` / `dataframe.join_table_function(...)` ← UDTF
 - `session.sql("call name(...)").collect()` ← SP

External Dependencies

- *imports* ← local/staged JAR/ZIP/Python/XML files/folders
 - **IMPORTS** = ('path') ← in CREATE PROCEDURE/FUNCTION
 - *session.add_import("path")* ← session level
- *packages* ← referenced int (Anaconda)/ext libraries (with "import")
 - **PACKAGES** = ('name', ...) ← in CREATE PROCEDURE/FUNCTION
 - *session.add_packages("name", ...)* ← session level
 - **@udtf(packages=["name", ...])** ← @proc/@udt/@udtf
 - *session.add_requirements("path/requirements.txt")* ← session level

Python Worksheets



Python Worksheets: Template

```
import snowflake.snowpark as snowpark    ← Snowpark required

def main(session: snowpark.Session):      ← default handler (can change)
    # your Python code goes here...
```

Python Stored Procedure wrapper



```
with <worksheet_name> as procedure ()    ← on-the-fly stored proc (wrapper)
    returns Table()                         ← from Settings: String/Variant/Table()
    language python                         ← always Python
    runtime_version=3.8                     ← always Python 3.8
    packages=('snowflake-snowpark-python')   ← from Packages
    handler='main'                          ← from Settings
    as '
        import snowflake.snowpark as snowpark
    def main(session: snowpark.Session):      ← default handler (can change)
        # your Python code goes here...
    '
call <worksheet_name>();                  ← execute on-the-fly stored proc
```

Python Worksheets: Custom Signature

Generated SQL Worksheet <Deploy> Open in Worksheets

```
create procedure <proc_name>(par1 int, par2 string, ...)
    returns Table()
    language python
    runtime_version = 3.8
    packages =('snowflake-snowpark-python==*')
    handler = 'main'
    as '
        import snowflake.snowpark as snowpark

        def main(session: snowpark.Session, par1, par2, ...):
            # your Python code goes here...
            '

call <proc_name>(100, "abc", ...);
```



Client Request

For security and performance reasons, can you deploy your whole Hierarchical Data Viewer in Snowflake, closer to data?

How can we log messages and trace events in Snowflake? We need to be alerted on the most serious errors.

Review of Client Request

For security and performance reasons, can you deploy your whole Hierarchical Data Viewer in Snowflake, closer to data?

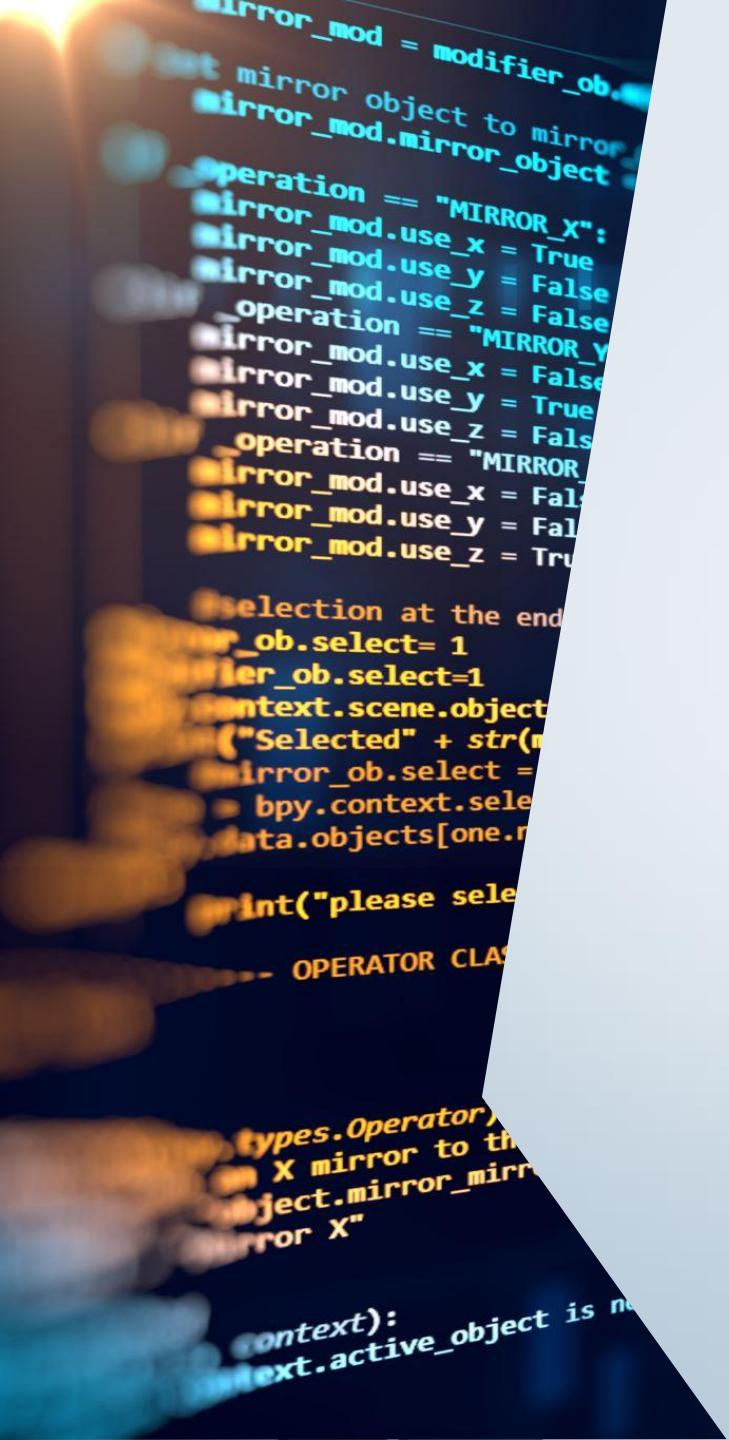
How can we log messages and trace events in Snowflake? We need to be alerted on the most serious errors.



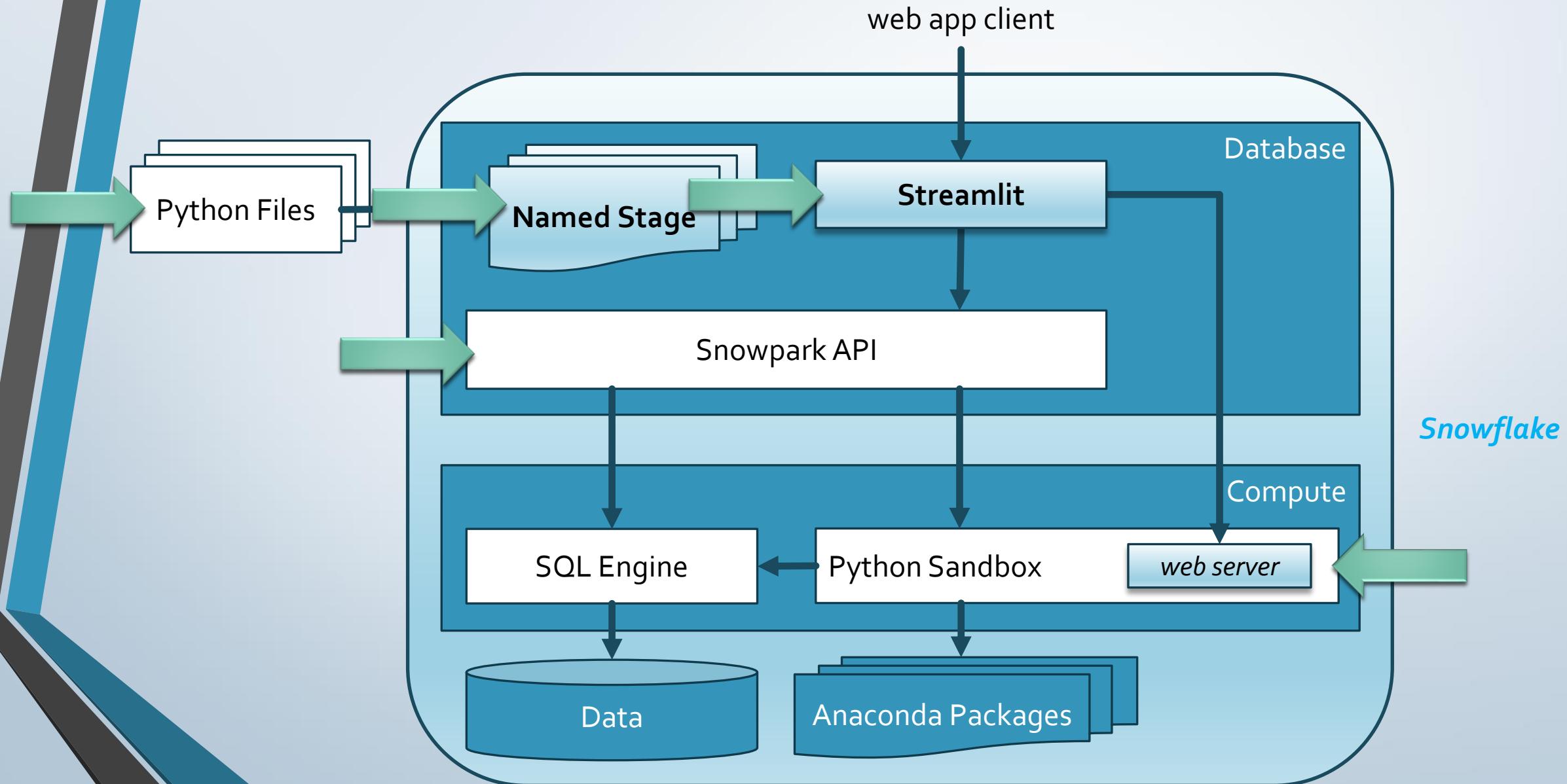
Section Summary



- Streamlit in Snowflake Architecture
- Test as Local Streamlit Web App
- Deploy as Streamlit in Snowflake App
- Event Tables
- Log Messages & Trace Events
- Alerts & Email Notifications



Streamlit in Snowflake



Streamlit in Snowflake

- **Create and test as a local Streamlit web app**
 - Create a local Streamlit app, with one or more Python files.
 - Connect locally to Snowflake through Snowpark.
 - Test your application as a local Streamlit web app.
- **Deploy as a Streamlit in Snowflake app**
 - Create a Snowflake `database` with a `named stage`.
 - Upload your Python and other app files into this stage.
 - Create a `STREAMLIT` object, mentioning the entry point file.
 - In Snowsight, start your new app in the new `Streamlit` tab.
 - Connect to Snowflake through `get_active_session()`
 - Continue editing, running, and testing the app in Snowsight.

Streamlit in Snowflake

< Streamlit Apps HIERARCHY_DATA_VIEWER

Share Run

Packages

```
1 import json, urllib.parse, os, configparser
2 import pandas as pd
3 import streamlit as st
4 import streamlit.components.v1 as components
5 import m2_hierarchical, m3_graphs, m4_charts, m5_animated
6 #import snowflake.connector
7 from snowflake.snowpark import Session
8 from snowflake.snowpark.context import get_active_session
9
10 # customize with your own Snowflake connection parameters
11 #@st.cache_resource(show_spinner="Connecting to Snowflake..")
12 def getSession():
13     try:
14         return get_active_session()
15     except:
16         parser = configparser.ConfigParser()
17         parser.read(os.path.join(os.path.expanduser('~'), 'connections.demo_conn'))
18         section = "connections.demo_conn"
19         pars = {
20             "account": parser.get(section, "accountname"),
21             "user": parser.get(section, "username"),
22             "password": os.environ['SNOWSQL_PWD']
23         }
24         return Session.builder.configs(pars).create()
25
26 #@st.cache_data(show_spinner="Running a Snowflake query...")
27 def getDataFrame(query):
28     try:
29         df = getSession().sql(query)
30         rows = df.collect()
```

Display your hierarchical data with charts and graphs.

Source Hierarchy Format Graph Chart

Select a chart type:

Treemap

KING

BLAKE

ALLEN MARTIN TURNER

JAMES WARD

JONES

FORD SCOTT

SMITH ADAMS

CLARK

MILLER

Event Tables

- **CREATE EVENT TABLE** *myevents* ← predefined cols, max 1MB/row
 - **ALTER ACCOUNT SET EVENT_TABLE** = *myevents* ← one per account
 - **ALTER ACCOUNT UNSET EVENT_TABLE**
 - **SHOW PARAMETERS LIKE 'event_table' IN ACCOUNT**
- *levels*
 - **ALTER ... SET LOG_LEVEL** = OFF/DEBUG/WARN/INFO/ERROR ← *log messages*
 - **ALTER ... SET TRACE_LEVEL** = OFF/ALWAYS/ON_EVENT ← *trace events*
- **SYSTEM\$LOG('level', message)**
- **SYSTEM\$LOG_TRACE/DEBUG/INFO/WARN/ERROR/FATAL(message)**

Event Table Columns

- **TIMESTAMP** - log time / event creation / end of a time span
- **START/OBSERVED_TIMESTAMP** - start of a time span
- **RECORD_TYPE** - **LOG / SPAN_EVENT / SPAN**
- **VALUE** - primary event value (as VARIANT)
- **TRACE** - trace_id/span_id, as tracing context
- **RESOURCE_ATTRIBUTES** - source db/schema/user/warehouse of event
- **SCOPE** - class names for logs (name...)
- **RECORD** - fixed values for each record type (severity_text...)
- **RECORD_ATTRIBUTES** - variable attributes for each record type

```
SELECT RECORD['severity_text'] AS SEVERITY, VALUE AS MESSAGE  
FROM myevents  
WHERE SCOPE['name'] = 'python_logger' AND RECORD_TYPE = 'LOG';
```

Event Tables: Log Messages & Trace Events

Python Code

```
# log messages
import logging

logger = logging.getLogger("mylog")
logger.info/debug/warning/error/log/... ("This is an INFO test.")
```

```
# trace events
from snowflake import telemetry

telemetry.add_event("FunctionEmptyEvent")
telemetry.add_event("FunctionEventWithAttributes", {"key1": "value1", ...})
```

Alerts

- **CREATE ALERT ...**
 - **WAREHOUSE** ← for compute resources
 - **SCHEDULE** ← cron expression, for periodical evaluation
 - **IF (EXISTS(*condition*))** ← SELECT/SHOW/CALL stmt to check condition
 - **THEN *action*** ← SQL CRUD/script/CALL, can also use system\$send_email(...)
- **ALTER ALERT ... SUSPEND/RESUME**
- INFORMATION_SCHEMA.**ALERT_HISTORY(...)** ← table function

```
CREATE ALERT myalert
  WAREHOUSE = mywarehouse
  SCHEDULE = '1 minute'
  IF(EXISTS(
    SELECT value FROM gauge WHERE value > 200
  ))
  THEN
    INSERT INTO gauge_history VALUES (current_timestamp());
```

Email Notifications

- **CREATE NOTIFICATION INTEGRATION ...**
 - **TYPE = EMAIL**
 - **ALLOWED_RECIPIENTS = (*email₁*, ...)** ← list of email addresses for TO
- SYSTEM\$**SEND_EMAIL**(*integration, to_list, subject, body, [mime_type]*)
- Snowflake Computing <no-reply@snowflake.net> ← from

```
CREATE NOTIFICATION INTEGRATION my_notif_int
  TYPE = EMAIL
  ENABLED = TRUE
  ALLOWED_RECIPIENTS = ('jon@example.com', 'mary@example.com');

-- send email
CALL SYSTEM$SEND_EMAIL(
  'my_notif_int',
  'jon@example.com, mary@example.com',
  'Email Alert: Task A has finished.',
  'Task A has successfully finished.\nEnd Time: 12:15:45');
```



Client Request

Our database could be accessed by local admins, editors and guests. Editors cannot make metadata changes, and guests can only see the data. Some other employees could be responsible with uploading data.

We also need a generic script to create specialized databases and prod/test envs for each of our partners.

Review of Client Request

Our database could be accessed by local admins, editors and guests. Editors cannot make metadata changes, and guests can only see the data. Some other employees could be responsible with uploading data.

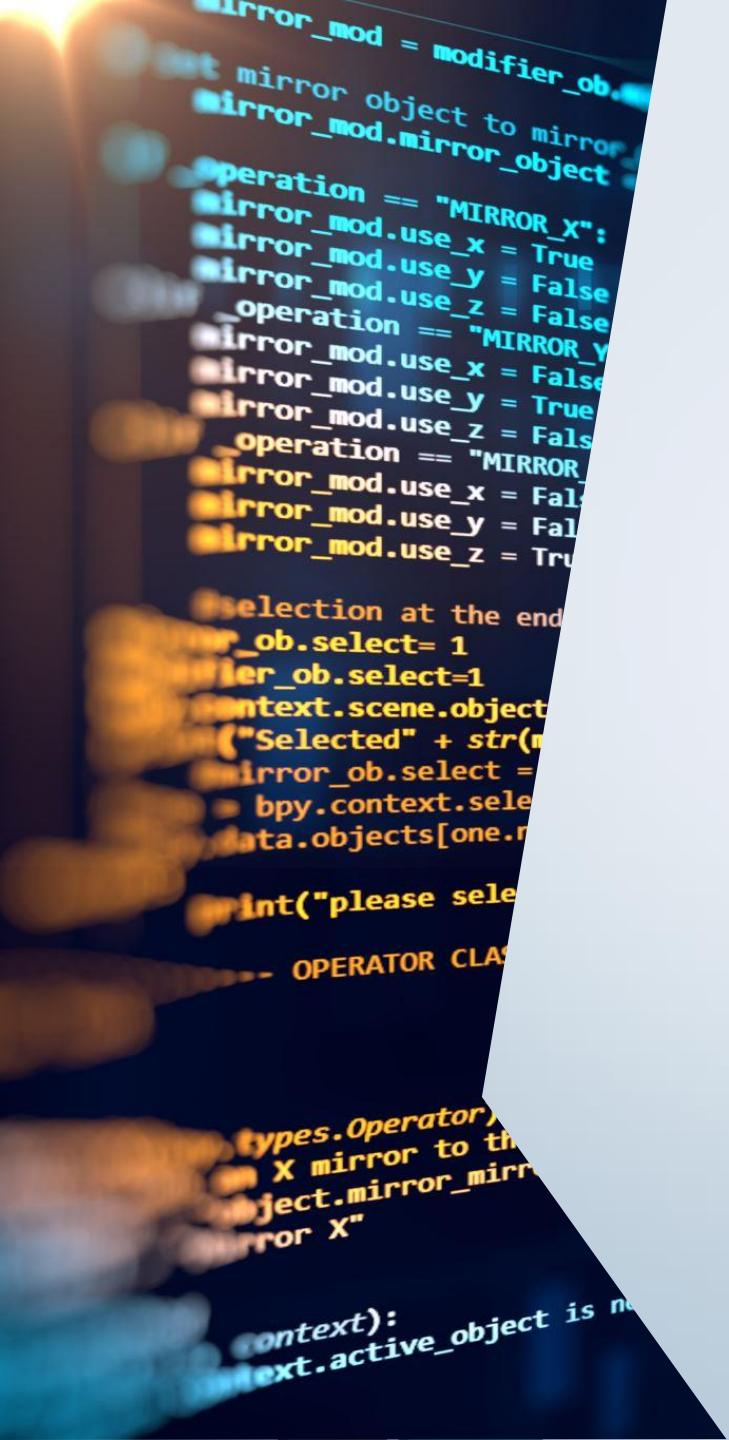
We also need a generic script to create specialized databases and prod/test envs for each of our partners.



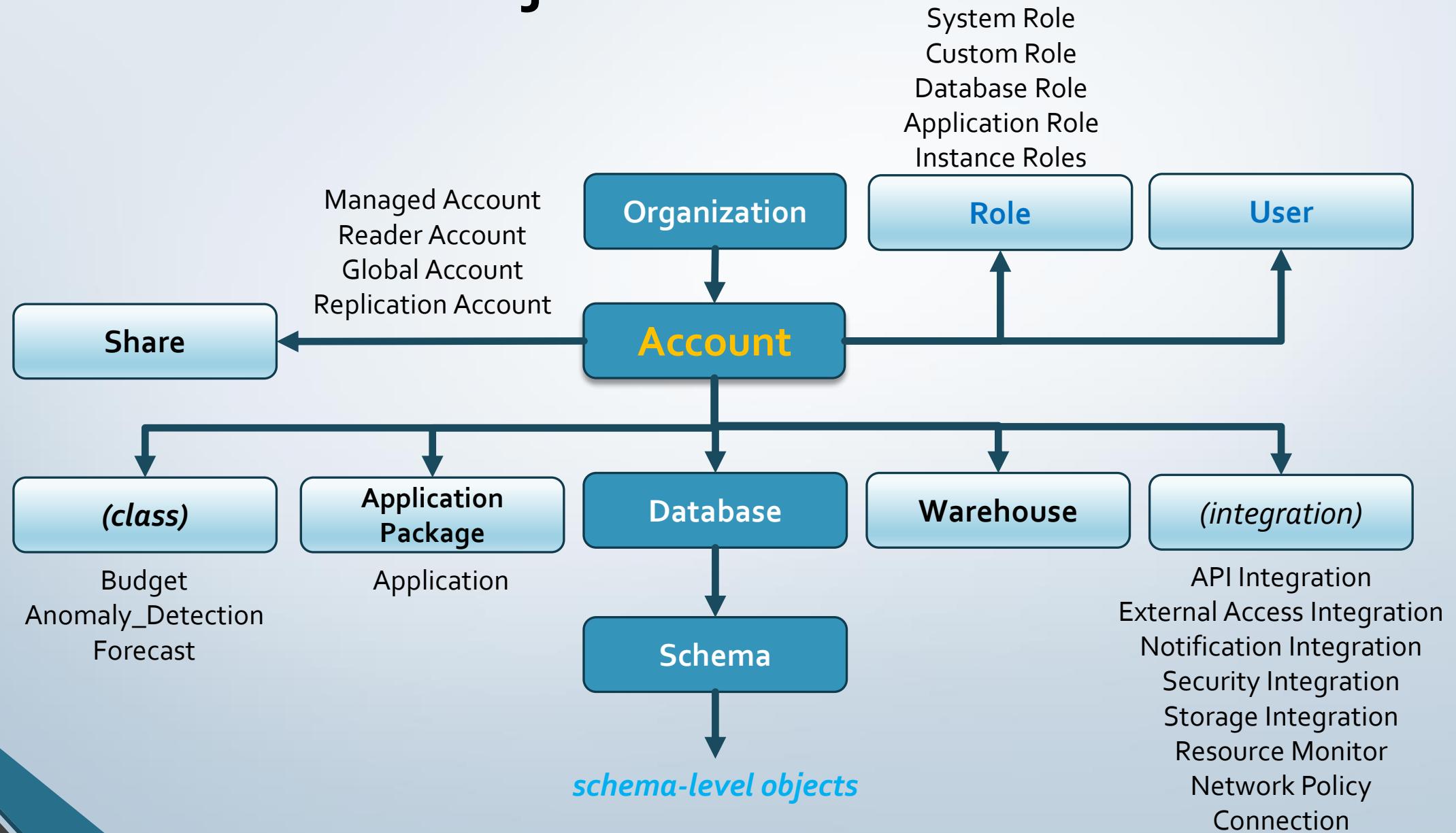
Section Summary



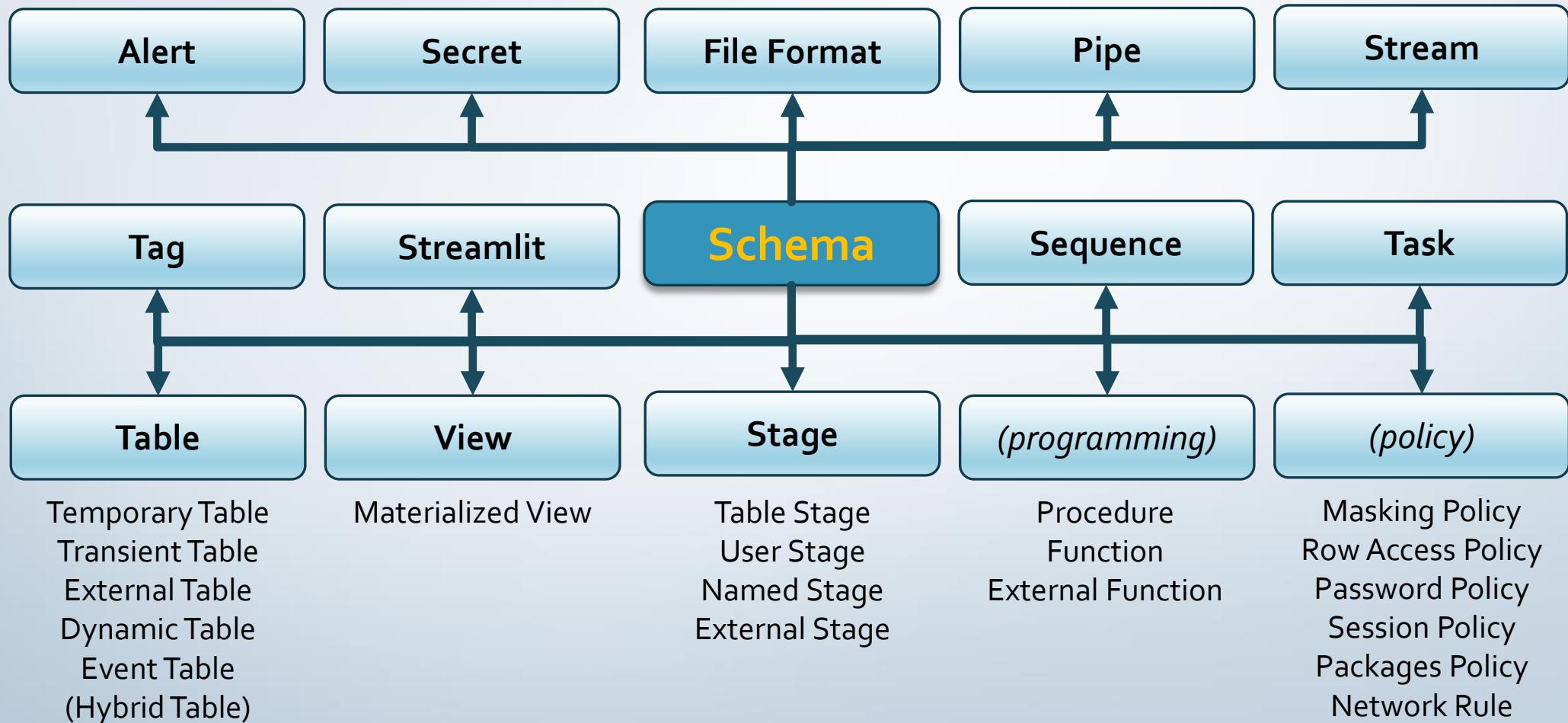
- Account and Schema-Level Objects
- Access Control Framework
- Data Control Language (DCL)
- The Roles Hierarchy
- System-Defined Roles
- Access Control Privileges
- Variables and Variable Substitution



Account-Level Objects

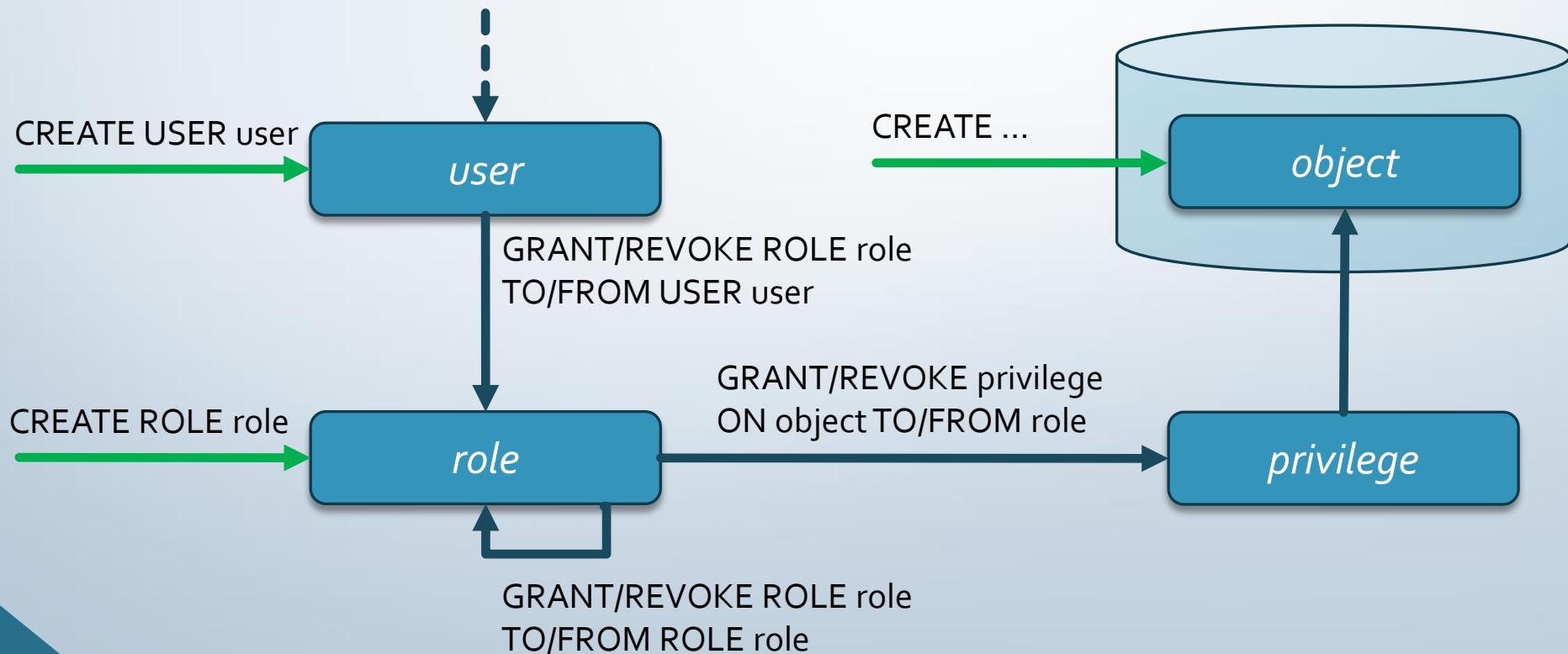


Schema-Level Objects



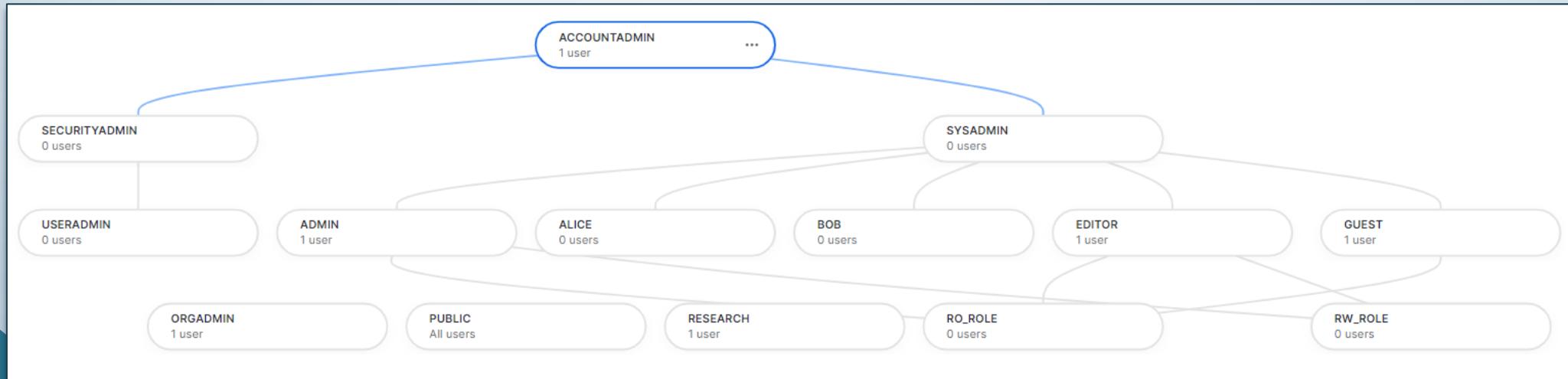
Access Control Framework

- **DAC (Discretionary Access Control)** - objects ← grant access ← owners
- **RBAC (Role-Based Access Control)** - objects ← privileges ← roles ← users

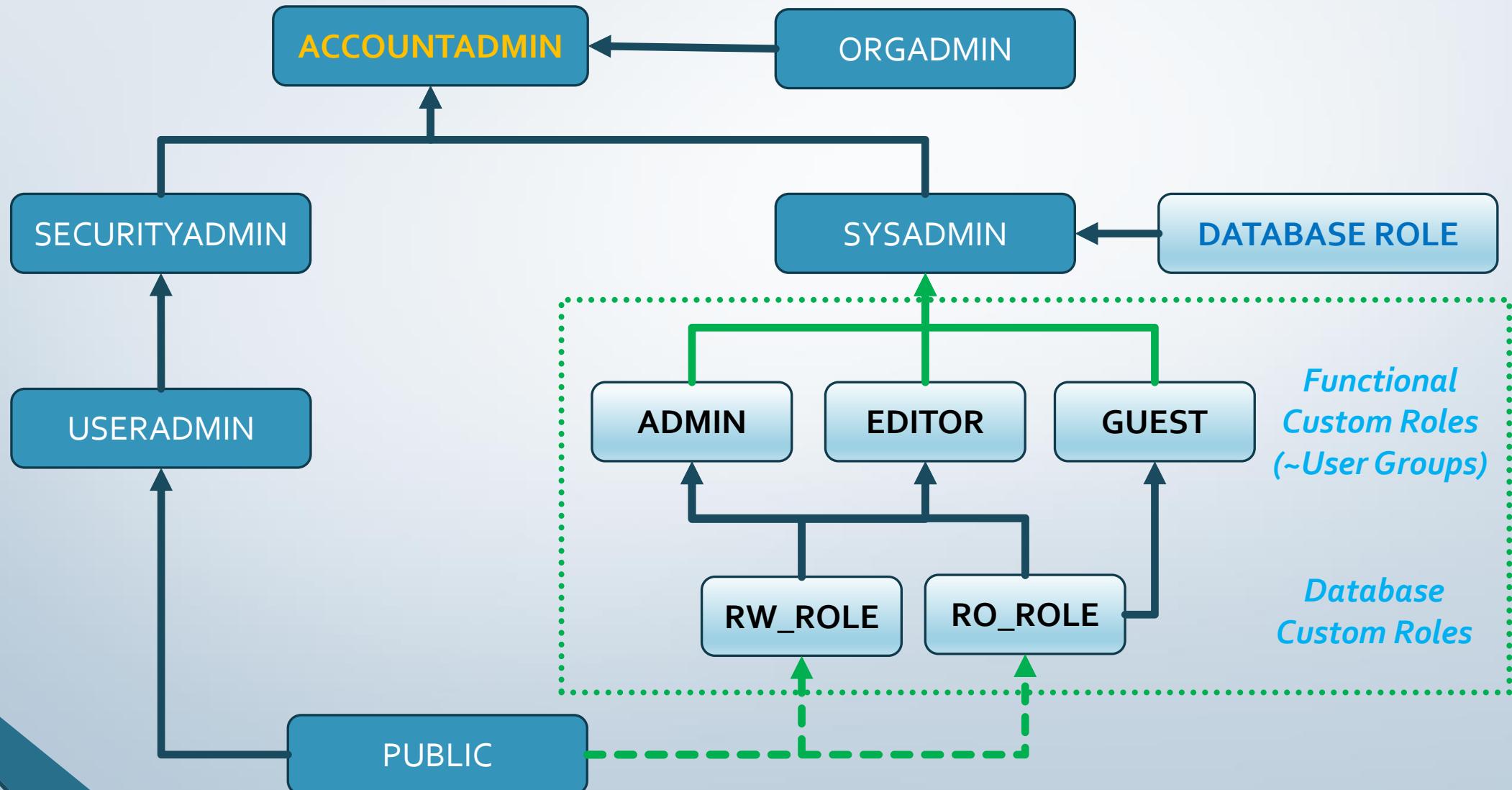


Data Control Language (DCL)

- **CREATE ROLE** role
 - **GRANT ROLE** role **TO ROLE** role ← create the role hierarchy
- **CREATE USER** user
 - **GRANT ROLE** role **TO USER** user ← assign roles to users
- **GRANT** privilege, ... **ON** obj_type obj_name **TO ROLE** role
 - **REVOKE** privilege, ... **ON** obj_type obj_name **FROM ROLE** role



The Role Hierarchy



System-Defined Roles

- **ACCOUNTADMIN** - top-level role
 - as SYSADMIN + SECURITYADMIN (+ ORGADMIN)
- **SECURITYADMIN** - to CREATE ROLE and GRANT ROLE to ROLE/USER
 - GRANT/REVOKE privileges, inherits USERADMIN
- **USERADMIN** - to CREATE USER
- **SYSADMIN** - to CREATE WAREHOUSE/DATABASE/...
 - GRANT privileges to objects, should inherit from any custom role
- **ORGADMIN** - to CREATE ACCOUNT in org
 - SHOW ORGANIZATION ACCOUNTS and SHOW REGIONS, view usage info in org
- **PUBLIC** - automatically granted to every user/role in your account.

Access Control Privileges

- **OWNERSHIP** - full control over an object to one single role
- **MANAGE GRANTS** - can grant/revoke privileges on any object (~OWNERSHIP)
- **IMPORTED PRIVILEGES** - may enable other roles to access a shared db
- **ALL [PRIVILEGES]** - all privileges, except OWNERSHIP
- **USAGE** - can USE/SHOW a db/schema/function/stage/warehouse etc
- **SELECT** - can query and display R/O table/view/stream data
- **INSERT/UPDATE/DELETE** - enable R/W CRUD operations on a table data
- **REFERENCES** - can display the structure of a table/view or set table constraints
- **EXECUTE** - can run task/alert
- **CREATE/MODIFT** - can create/alter different types of objects
- **APPLY** - can add/drop policies/tags
- **ON FUTURE** - on db/schema objects yet to be created

Create Roles and Users

```
-- create roles
USE ROLE SECURITYADMIN;
CREATE ROLE ADMIN;
CREATE ROLE EDITOR;
CREATE ROLE GUEST;
CREATE ROLE RO_ROLE;
CREATE ROLE RW_ROLE;
```

```
-- create users
USE ROLE USERADMIN;
CREATE USER MARK;
CREATE USER CLAUDE;
CREATE USER MARY;
```

```
-- create the role hierarchy
USE ROLE SECURITYADMIN;
GRANT ROLE ADMIN TO ROLE SYSADMIN;
GRANT ROLE EDITOR TO ROLE SYSADMIN;
GRANT ROLE GUEST TO ROLE SYSADMIN;
GRANT ROLE RO_ROLE TO ROLE ADMIN;
GRANT ROLE RW_ROLE TO ROLE ADMIN;
GRANT ROLE RO_ROLE TO ROLE EDITOR;
GRANT ROLE RW_ROLE TO ROLE EDITOR;
GRANT ROLE RO_ROLE TO ROLE GUEST;
```

```
-- assign roles to users
USE ROLE SECURITYADMIN;
GRANT ROLE ADMIN TO USER MARK;
GRANT ROLE EDITOR TO USER CLAUDE;
GRANT ROLE GUEST TO USER MARY;
```

Grant Database Privileges to Roles

```
-- create database (w/ PUBLIC schema)
CREATE DATABASE security;

-- R/O privileges (RO_ROLE)
GRANT OPERATE, USAGE ON WAREHOUSE compute_wh TO ROLE RO_ROLE;

GRANT USAGE ON DATABASE security TO ROLE RO_ROLE;
GRANT USAGE ON SCHEMA security.public TO ROLE RO_ROLE;
GRANT SELECT ON ALL TABLES IN SCHEMA security.public TO ROLE RO_ROLE;
GRANT SELECT ON FUTURE TABLES IN SCHEMA security.public TO ROLE RO_ROLE;

-- R/W privileges (RW_ROLE)
GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA security.public TO ROLE RW_ROLE;
GRANT INSERT, UPDATE, DELETE ON FUTURE TABLES IN SCHEMA security.public TO ROLE RW_ROLE;
GRANT CREATE TABLE, CREATE VIEW ON SCHEMA security.public TO ROLE RW_ROLE;

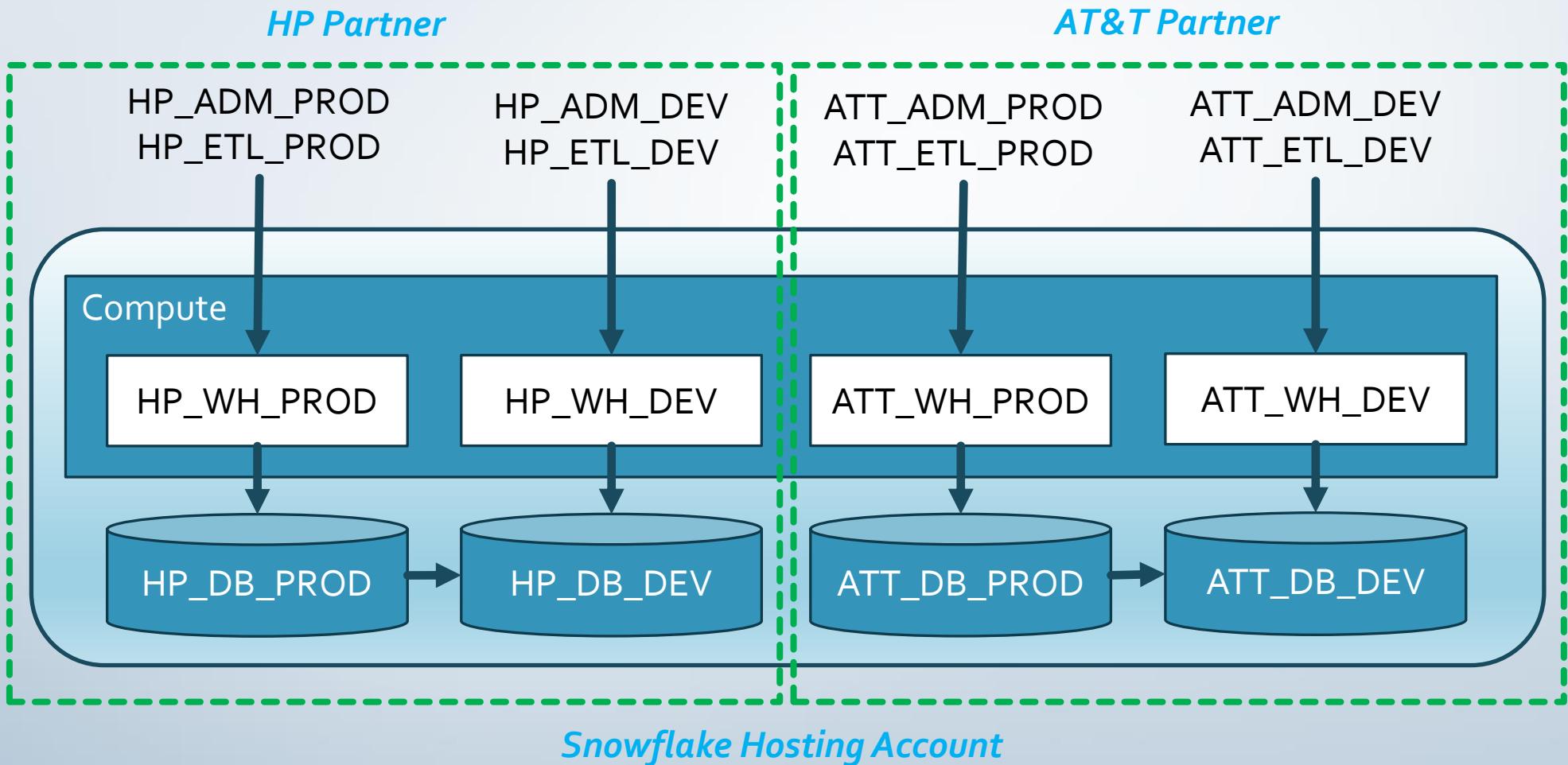
-- ADMIN privileges (ADMIN)
GRANT ALL ON DATABASE security TO ROLE ADMIN;
GRANT ALL ON FUTURE SCHEMAS IN DATABASE security TO ROLE ADMIN;
```

Inspect Database Object Privileges

```
select privilege_type, is_grantable, grantee  
from security.information_schema.object_privileges  
where object_catalog = 'SECURITY';
```

PRIVILEGE_TYPE	IS_GRANTABLE	GRANTEE
CREATE TABLE	NO	RW_ROLE
OWNERSHIP	YES	ACCOUNTADMIN
CREATE VIEW	NO	RW_ROLE
USAGE	NO	RO_ROLE

Multi-Tenant Architecture



Multi-Tenant Databases and Environments

Command-Line Call

```
> snowsql -c my_conn -f create.sql -D tenant=HP -D env=PROD
```

SQL Script w/ var substitution

```
-- have variable substitution ON
!SET VARIABLE_SUBSTITUTION=true;

-- create new roles for tenant Admin and tenant ETL data engineer
USE ROLE SECURITYADMIN;
CREATE OR REPLACE ROLE &{tenant}_ADM_&{env};
CREATE OR REPLACE ROLE &{tenant}_ETL_&{env};

-- grant privileges to new tenant Admin role
USE ROLE ACCOUNTADMIN;
GRANT CREATE DATABASE ON ACCOUNT TO ROLE &{tenant}_ADM_&{env};
GRANT CREATE WAREHOUSE ON ACCOUNT TO ROLE &{tenant}_ADM_&{env};

-- create tenant database, w/ new tenant Admin role
USE ROLE &{tenant}_ADM_&{env};
CREATE DATABASE &{tenant}_DB_&{env};
GRANT USAGE ON DATABASE &{tenant}_DB_&{env} TO ROLE &{tenant}_ETL_&{env};
...
```

A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His right hand holds a pen. The background behind him is a light grey.

Client Request

First, we'd like to run a Snowflake query from a PowerShell script, without any application or other code.

Second, when we drop employee data files into an internal named stage, we want those files automatically loaded into a table.

Review of Client Request

First, we'd like to run a Snowflake query from a PowerShell script, without any application or other code.

Second, when we drop employee data files into an internal named stage, we want those files automatically loaded into a table.



Section Summary



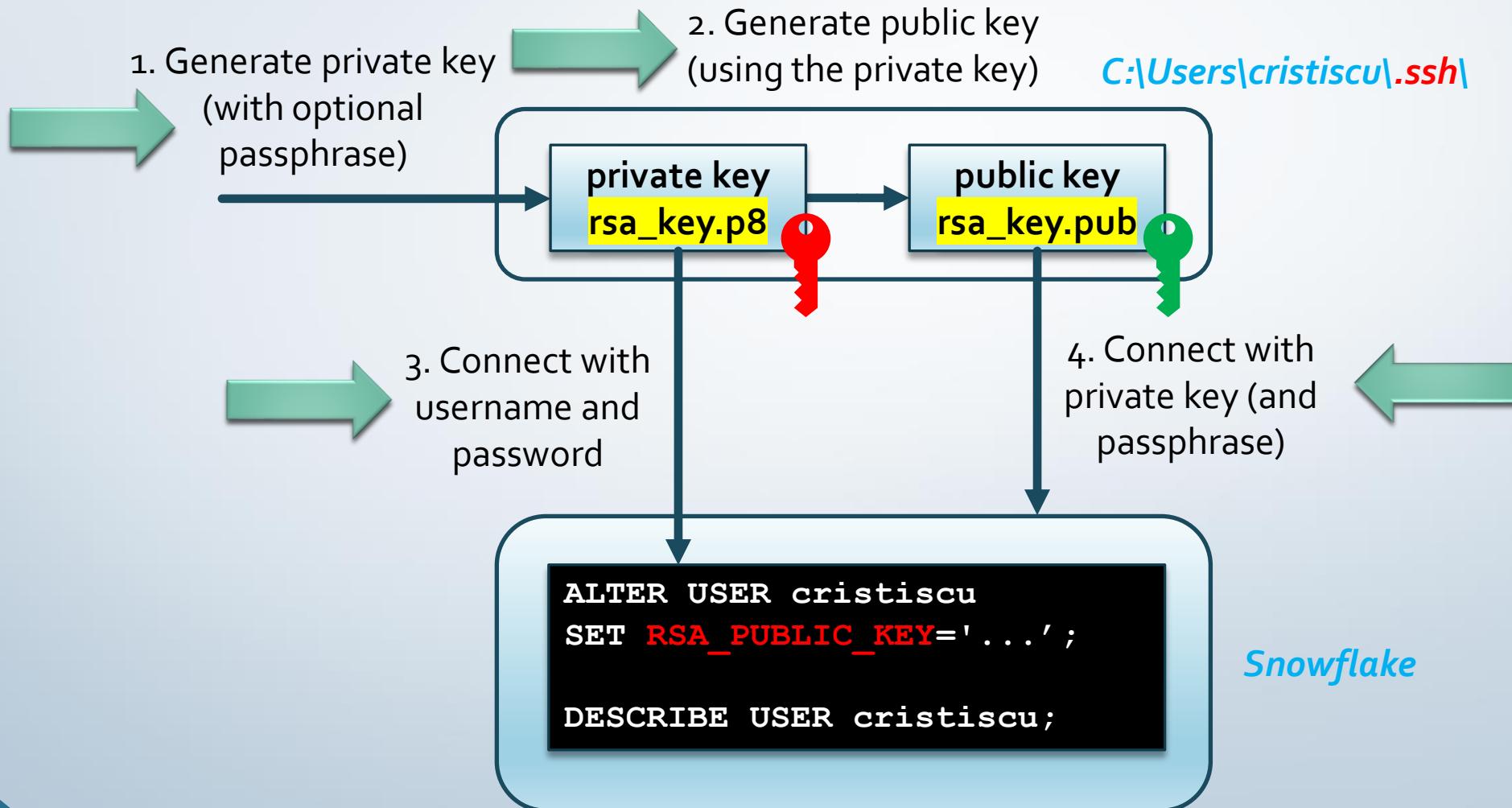
- Configure Key Pair Authentication
- Generate JWT with SnowSQL
- Snowpipe REST API
- Snowflake SQL REST API
- Cancel Running Query
- POST Command with curl



Key Pair Authentication

- Required for: Snowpipe API, SQL REST API
- Create and save a ***passphrase*** (in local env var) for an encrypted private key
- Generate and save (always local) a ***private key*** → **rsa_key.p8**
- Generate and save a ***public key*** (based on the private key) → **rsa_key.pub**
- Connect w/ basic authN and set **RSA_PUBLIC_KEY** for current user
- Reconnect w/ private key (and passphrase, if encrypted)
- Can later generate temporary ***JWT token*** w/ SnowSQL

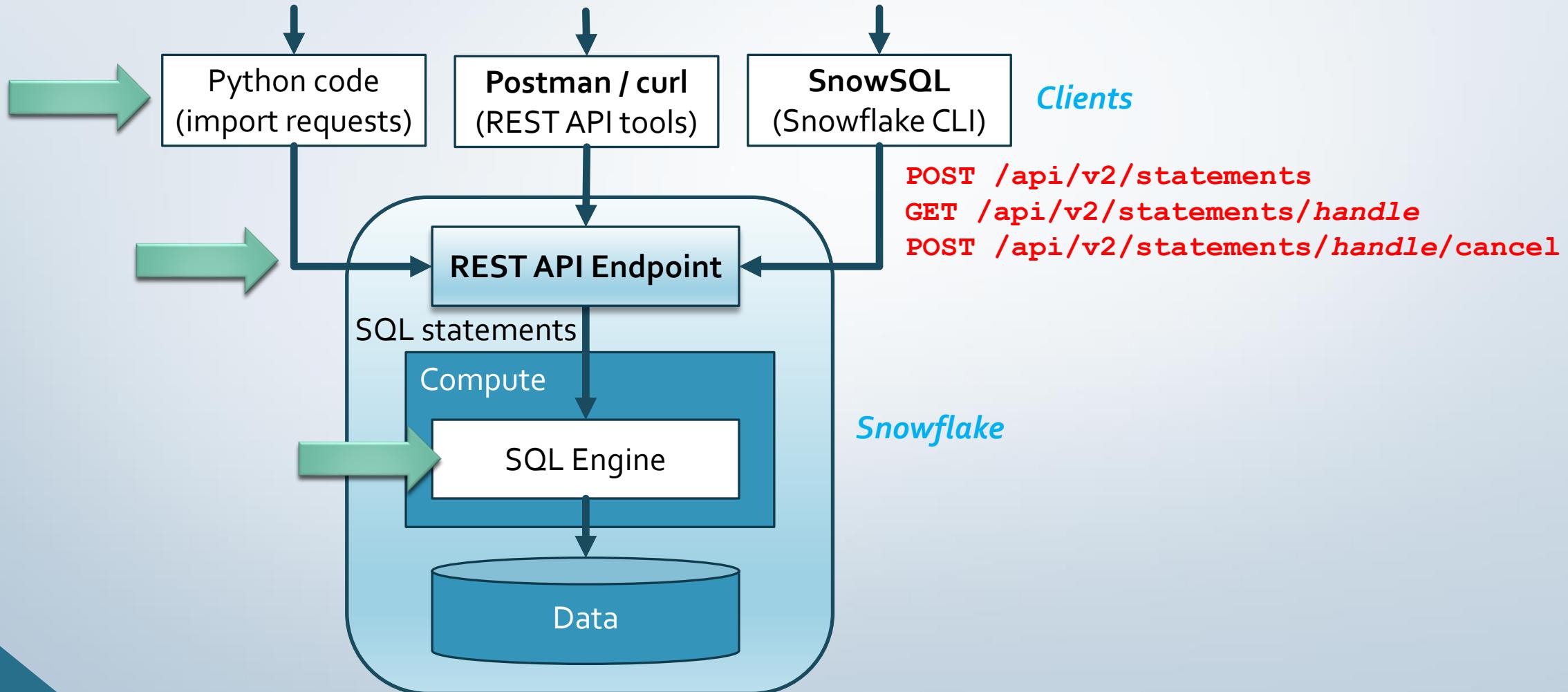
Key Pair Authentication: Configuration



SQL REST API

- <https://org.account.snowflakecomputing.com/api> ← API endpoint
- **POST /api/v2/statements** ← submit SQL statements for execution
- **GET /api/v2/statements/*handle*** ← check execution status of a statement
- **POST /api/v2/statements/*handle*/cancel** ← cancel statement execution
- Required authentication with either **OAuth** or **Key Pair**, with JWT token.
- Data is returned in partitions.
- Can fetch query results concurrently.
- No PUT or GET commands.

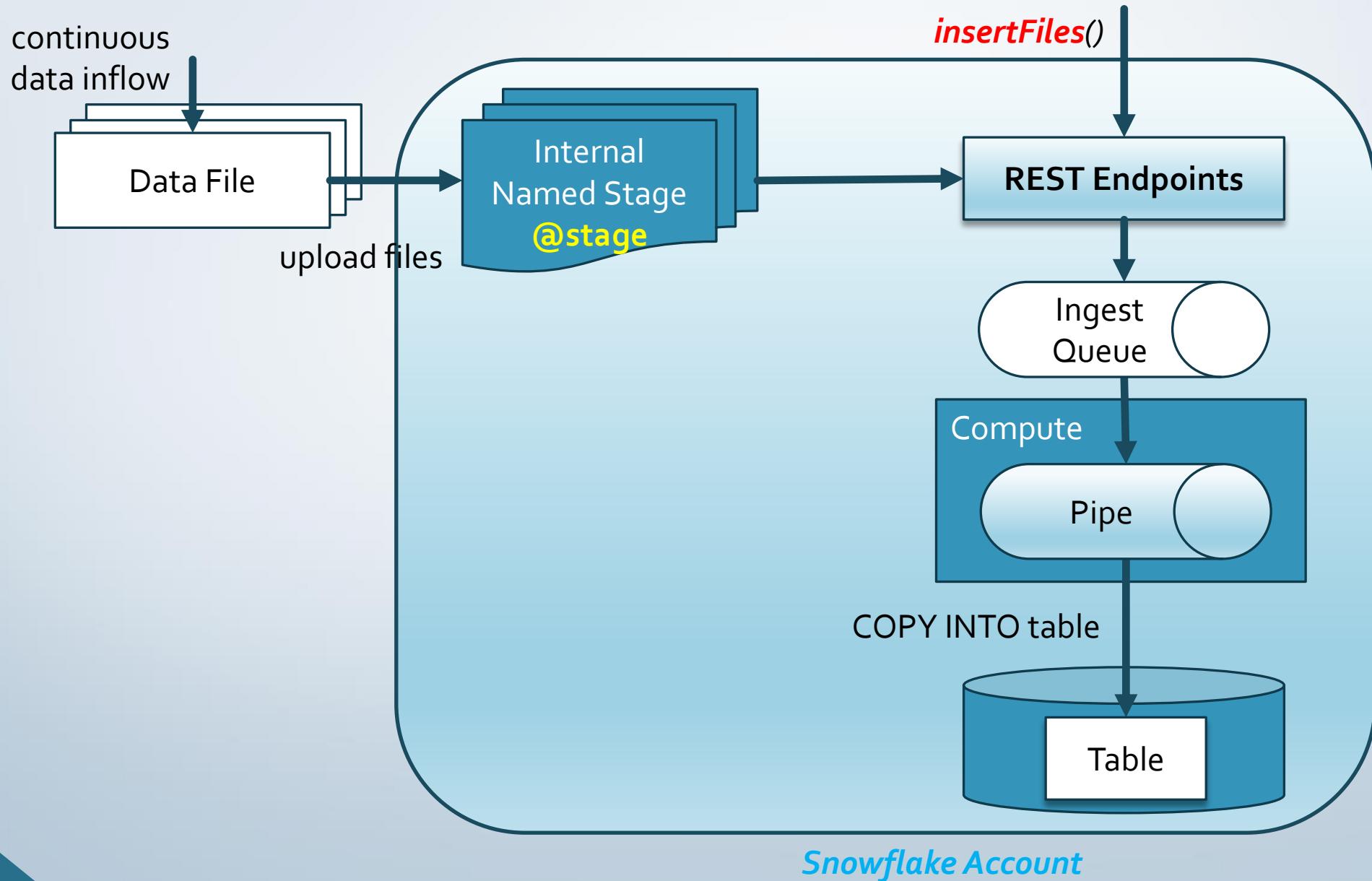
SQL REST API



Snowpipe REST API

- <https://acct.snowflakecomputing.com/v1/data/pipes/name> ← API endpoint
- **POST /insertFiles?requestId=id** ← triggers COPY TO cmd, to ingest list of files
- **GET /insertReport?requestId=id&beginMark=mark** ← report of previous ingestion
- **GET /loadHistoryScan?startTimeIncl=stime&endTimeExcl=etime&requestId=id** ← report of ingestion during period
- Required authentication with **Key Pair** and generated JWT token.

Snowpipe REST Endpoints



A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the slide. He is holding a white clipboard and looking down at it. His hands are visible, one holding the clipboard and the other holding a pen. The background behind him is a plain, light color.

Client Request

We need to auto-discover and mask the sensitive customer data. Employee salaries as well. And queries must be automatically tagged with user's department.

Employees of the RESEARCH department cannot see the actual year of the HIREDATE in the employees table, and should be restricted to their own records.

Review of Client Request

We need to auto-discover and mask the sensitive customer data. Employee salaries as well. And queries must be automatically tagged with user's department.

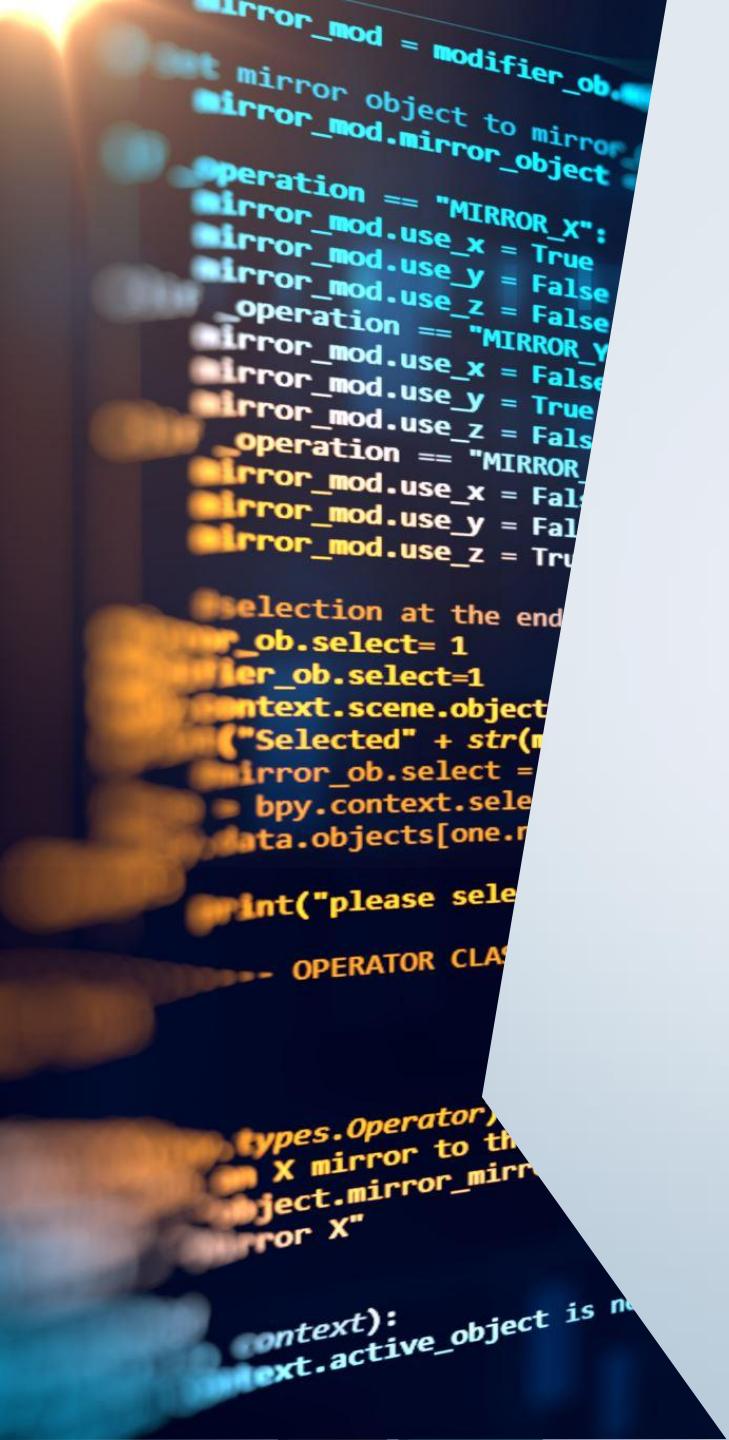
Employees of the RESEARCH department cannot see the actual year of the HIREDATE in the employees table, and should be restricted to their own records.



Section Summary



- Data Governance
- Object Tagging
- Query Tagging
- Data Classification
- Masking Policies
- Row Access Policies



Data Governance

- **Object Tagging**
 - Query Tagging
- **Data Classification**
 - System tags & categories
- **Column-Level Masking Policies**
 - Dynamic Data Masking
 - External Tokenization
 - Tag-Based Masking
- **Row Access Policies**

Object Tagging

- Monitors sensitive data for compliance, discovery, protection, resource usage.
- Schema-level object, inherited by any child object → **tag lineage**
- **CREATE TAG <tag> ALLOWED_VALUES <value1>, ...**
- **ALTER TAG <tag> DROP ALLOWED_VALUES <value1>, ...**
- **SHOW TAGS ...**
- **CREATE <object> WITH TAG (<tag> = <value>)**
- **ALTER <object> SET TAG <tag> = <value>**
- **ALTER <object> UNSET SET <tag>**
- **SYSTEM\$GET_TAG(<tag>, <obj>, <domain>)** → assigned object tag value
- **INFORMATION_SCHEMA.TAG_REFERENCES** table function
- **ACCOUNT_USAGE.TAG_REFERENCES** view

Query Tag

- **QUERY_TAG** ← *session*-level parameter to auto-tag queries
 - **QUERY_HISTORY** ← query w/ the tag
-
- can assign to group **users** as well
 - set at the **account** level for all allowed values
 - similar, but better than *end-comments* in a query text
 - analytic queries run by Looker/Tableau/Power BI can use this feature

Data Classification

- Applies system-defined tags to recognize and protect sensitive data
- SNOWFLAKE.CORE.PRIVACY_CATEGORY
 - **identifier** = name, email, IP address, url, bank account, driver license...
 - **quasi-identifier** = age, gender, country, date of birth, occupation, city...
 - **sensitive/insensitive** = salary
- SNOWFLAKE.CORE.SEMANTIC_CATEGORY
 - personal attributes: **name, age, email, bank account...**

```
show tags in schema SNOWFLAKE.CORE;  
select "name", "allowed_values" from table(result_scan(last_query_id()));
```

name	allowed_values
PRIVACY_CATEGORY	["IDENTIFIER","QUASI_IDENTIFIER","SENSITIVE","INSENSITIVE"]
SEMANTIC_CATEGORY	["ADMINISTRATIVE_AREA_1","ADMINISTRATIVE_AREA_2","AGE","BANK_ACCOUNT","C"]

Data Classification: Generate Fake Data

```
import snowflake.snowpark as snowpark
from snowflake.snowpark.types import StructType, StructField, StringType
from faker import Faker
def main(session: snowpark.Session):
    f = Faker()
    output = [[f.name(), f.address(), f.city(), f.state(), f.email()]]
    for _ in range(10000)]
    schema = StructType([
        StructField("NAME", StringType(), False),
        StructField("ADDRESS", StringType(), False),
        StructField("CITY", StringType(), False),
        StructField("STATE", StringType(), False),
        StructField("EMAIL", StringType(), False)])
    df = session.create_dataframe(output, schema)
    df.write.mode("overwrite").save_as_table("CUSTOMERS_FAKE")
    return df
```

NAME	ADDRESS	CITY	STATE	EMAIL
Angel Campbell	79055 Deanna Plaza Apt. 641 New Juliamouth,	Smithtown	North Carolina	russell86@example.com
Janet Anderson	USNS Wilkerson FPO AA 08823	Danielport	West Virginia	brittany53@example.net
Roger Cook	736 Smith Ferry Suite 402 Kimberlyfurt, SC 012	North Pennyfort	Connecticut	nicoleclarke@example.org
Brenda Perez	5207 Sanders Centers Apt. 172 North Cliffordfo	South Rebecca	Alabama	wbutler@example.com

Data Classification: Extract Semantic Categories

```
-- SELECT EXTRACT_SEMANTIC_CATEGORIES('CUSTOMERS_FAKE');

SELECT f.key::varchar as column_name,
       f.value:"recommendation":privacy_category::varchar as privacy_category,
       f.value:"recommendation":semantic_category::varchar as semantic_category,
       f.value:"recommendation":confidence::varchar as confidence,
       f.value:"recommendation":coverage::number(10,2) as coverage,
       f.value:"details":variant as details, f.value:"alternates":variant as alts
FROM TABLE(FLATTEN(EXTRACT_SEMANTIC_CATEGORIES('CUSTOMERS_FAKE')::VARIANT)) AS f;
```

COLUMN_NAME	PRIVACY_CATEGORY	SEMANTIC_CATEGORY	...	CONFIDENCE	COVERAGE
ADDRESS	null	null		null	null
CITY	null	null		null	null
EMAIL	IDENTIFIER	EMAIL		HIGH	1.00
NAME	IDENTIFIER	NAME		HIGH	1.00
STATE	QUASI_IDENTIFIER	ADMINISTRATIVE_AREA_1		HIGH	1.00

Data Classification: Apply Semantic Categories

```
CALL ASSOCIATE_SEMANTIC_CATEGORY_TAGS(  
    'EMPLOYEES.PUBLIC.CUSTOMERS_FAKE',  
    EXTRACT_SEMANTIC_CATEGORIES('EMPLOYEES.PUBLIC.CUSTOMERS_FAKE'))
```

Applied tag `semantic_category` to 3 columns.

Applied tag `privacy_category` to 3 columns.

```
select * from table(information_schema.tag_references(  
    'EMPLOYEES.PUBLIC.CUSTOMERS_FAKE.EMAIL', 'column'));
```

TAG_DATABASE	TAG_SCHEMA	...	TAG_NAME	TAG_VALUE	LEVEL	OBJECT_DATABASE
SNOWFLAKE	CORE		PRIVACY_CATEGORY	IDENTIFIER	COLUMN	EMPLOYEES
SNOWFLAKE	CORE	↓	SEMANTIC_CATEGORY	EMAIL	COLUMN	EMPLOYEES

Tagged Objects in Snowsight

The screenshot displays the Snowsight Governance interface. On the left, a sidebar menu under the 'Data' section includes options like Databases, Private Sharing, Provider Studio, and Governance, with Governance being the active tab. The main area is titled 'Governance' and shows two tabs: 'Dashboard' and 'Tagged Objects', with 'Tagged Objects' being the active tab. Below these tabs are several filter buttons: 'All All', 'Columns', 'Has Tags X', and 'All Policy'. A status message indicates '4 columns' and 'Information latency can be up to 2 hours'. The main content area is a table listing four tagged objects:

NAME ↑	DATABASE	SCHEMA	TABLE	TAGS ⓘ	MASKING...
EMAIL	EMPLOYEE...	PUBLIC	CUS...	S... E +1	—
NAME	EMPLOYEE...	PUBLIC	CUS...	P... I.. +1	—
SAL	EMPLOYEE...	PUBLIC	EMP	SECURITY_C... P.	—
STATE	EMPLOYEE...	PUBLIC	CUS...	S... A... +1	—

Masking & Access Policies

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
1000	SMITH	CLERK	12-17-1980	800	20
1010	KING	MANAGER	01-12-1981	2,975.8	20
1020	WARD	CLERK	12-03-1981	3,000	20
1030	ADAMS	CLERK	12-09-1981	1,100	20
1040	TURNER	ANALYST	04-02-1981	3,000	20
1050	MARTIN	ANALYST	01-12-1981	3,000	20
1060	JONES	MANAGER	04-02-1981	2,975.8	20
1070	SCOTT	ANALYST	12-09-1981	3,000	20
1080	FRANCIS	CLERK	12-17-1981	800	20
1090	BLAKE	MANAGER	01-12-1981	2,975.8	20
1100	CLARK	MANAGER	01-12-1981	2,975.8	20
1110	ADAMS	CLERK	12-03-1981	1,100	20
1120	WILLIAMS	ANALYST	12-09-1981	3,000	20
1130	JAMES	ANALYST	04-02-1981	3,000	20
1140	MILLER	CLERK	12-17-1981	800	20

*masking
(column-level)*

*access
(row-level)*

Masking Policies (Column-Level)

- **Dynamic Data Masking**
 - to mask stored data with **built-in function** ← *data visualization protection*
 - **(***)-****** ← *masked* (or NULL)
 - **(***)-4465** ← *partially-masked* (604) 555-4465
- **External Tokenization**
 - to store tokenized data w/ **external function** ← *data storage protection*
 - **(Gw6) fk2-cHSI** ← *obfuscated*
 - **dslgklnbsdfsdfxzc** ← *tokenized* (encoded)
- **Tag-Based Masking**
 - ~dynamic data masking, but with a 'PII' security tag value

Masking Policies (Column-Level)

```
create masking policy research_on_year  
as (hiredate date) returns date ->  
case when current_role() <> 'RESEARCH' then val  
else date_from_parts(2000, month(hiredate), day(hiredate)) end;  
  
alter table emp  
modify column hiredate  
set masking policy research_on_year;  
  
select * from emp;      ← with role RESEARCH
```

EMPNO	ENAME	JOB	...	MGR	HIREDATE	SAL	COMM
7,566	JONES	MANAGER		7,839	2000-04-02	2,975.8	null
7,788	SCOTT	ANALYST		7,566	2000-12-09	3,000	null
7,876	ADAMS	CLERK		7,788	2000-01-12	1,100	null
7,902	FORD	ANALYST		7,566	2000-12-03	3,000	null
7,369	SMITH	CLERK		7,902	2000-12-17	800	null

Tag-Based Column Masking

```
CREATE TAG security_class ALLOWED_VALUES 'PII', 'PCA', 'PHI';

create masking policy research_on_year_tag
    as (hiredate date) returns date ->
    case when SYSTEM$GET_TAG_ON_CURRENT_COLUMN(
        'EMPLOYEES.PUBLIC.SECURITY_CLASS') <> 'PII' then hiredate
    else date_from_parts(2000, month(hiredate), day(hiredate)) end;

ALTER TAG security_class
    SET MASKING POLICY research_on_year_tag;

ALTER TABLE emp
    ALTER COLUMN hiredate SET TAG security_class = 'PII';
```

Access Policies (Row-Level)

```
create row access policy research_on_emp  
as (deptno int) returns boolean ->  
deptno = 20 or current_role() <> 'RESEARCH';
```

```
alter table emp  
add row access policy research_on_emp  
on (deptno);
```

```
select * from emp;      ← with role RESEARCH
```

EMPNO	ENAME	JOB	...	MGR	HIREDATE	SAL	COMM	DEPTNO
7,566	JONES	MANAGER		7,839	2000-04-02	2,975.8	null	20
7,788	SCOTT	ANALYST		7,566	2000-12-09	3,000	null	20
7,876	ADAMS	CLERK		7,788	2000-01-12	1,100	null	20
7,902	FORD	ANALYST		7,566	2000-12-03	3,000	null	20
7,369	SMITH	CLERK		7,902	2000-12-17	800	null	20

A professional man with glasses, wearing a dark suit, light blue shirt, and patterned tie, is shown from the waist up. He is holding a white clipboard with both hands, looking down at the paper. The background behind him is plain white.

Client Request

Most departments will get their own Snowflake account, and we will need to share our data with them.

Some departments may not have their own account, but we still need to come up with a solution.

We also need to share some aggregate data with our partners. But they should not be able to guess individual row values, while we cannot access their data at all.

Review of Client Request

Most departments will get their own Snowflake account, and we will need to share our data with them.

Some departments may not have their own account, but we still need to come up with a solution.

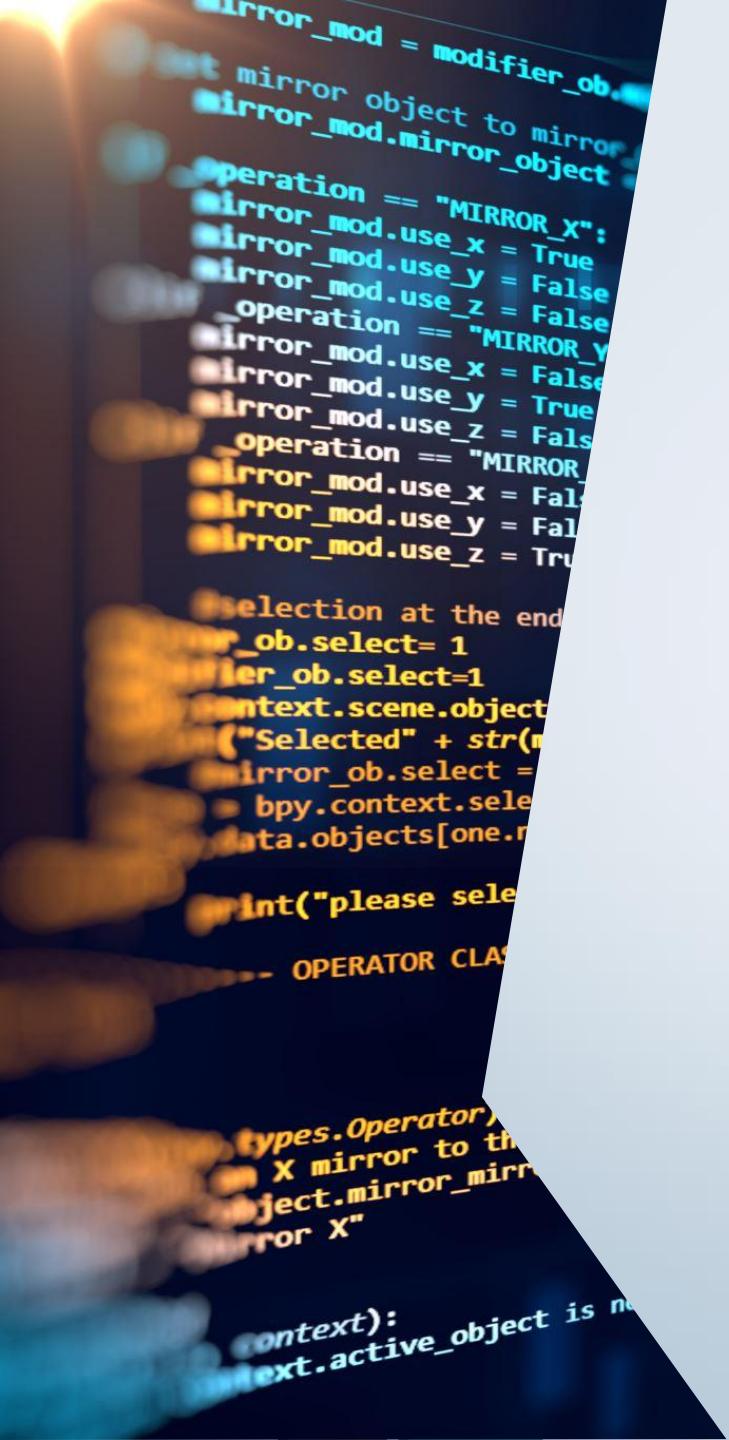
We also need to share some aggregate data with our partners. But they should not be able to guess individual row values, while we cannot access their data at all.



Section Summary



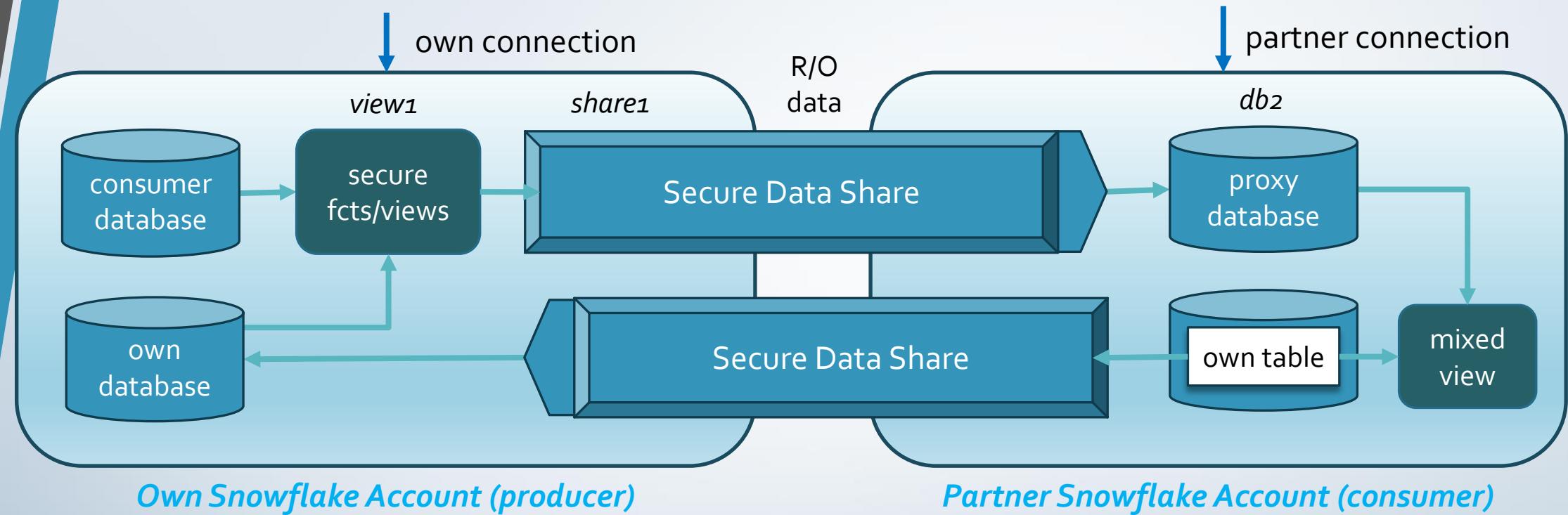
- Secure Data Sharing
- Secure Functions and Views
- Reader Accounts
- Private Share (Data Exchange)
- Public Share (Snowflake Marketplace)
- Data Clean Rooms



Secure Data Sharing

- **CREATE SHARE ...** ← by *producer*, always w/ R/O access!
 - **GRANT USAGE ON DATABASE/SCHEMA ... TO SHARE ...** ← required!
 - **GRANT SELECT ON VIEW ... TO SHARE ...** ← *secure views*
 - **GRANT USAGE ON FUNCTION ... TO SHARE ...** ← *secure functions*
 - **ALTER SHARE ... ADD ACCOUNTS = ..., ...** ← consumer & reader accounts
- **CREATE DATABASE ... FROM SHARE ...** ← by *consumer*, as proxy db
 - **GRANT IMPORTED PRIVILEGES ... ON DATABASE ... TO ROLE ...**

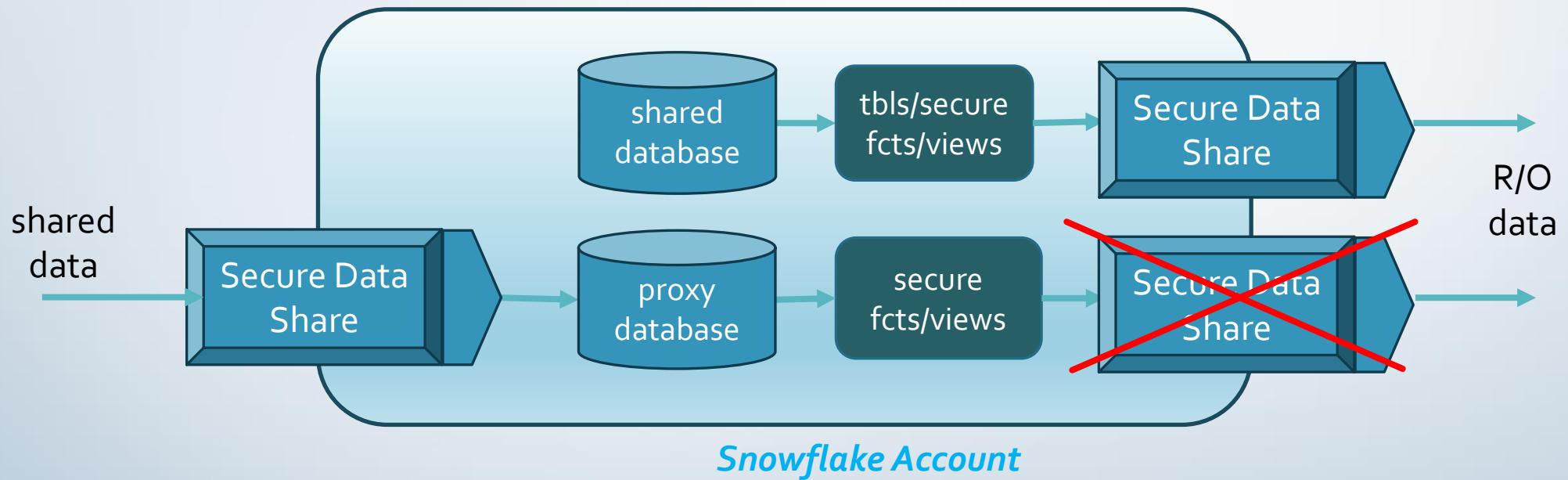
Inbound/Outbound Data Shares



```
create share share1;
grant usage on database db1 to share share1;
grant usage on schema db1.sch1 to share share1;
grant select on view view1 to share share1;
alter share share1 add accounts = consumer;
```

```
create database db2
from share producer.share1;
select * from db2.view1;
grant imported privileges
on database db2 to role role2;
```

Cannot Share Already Shared Data!

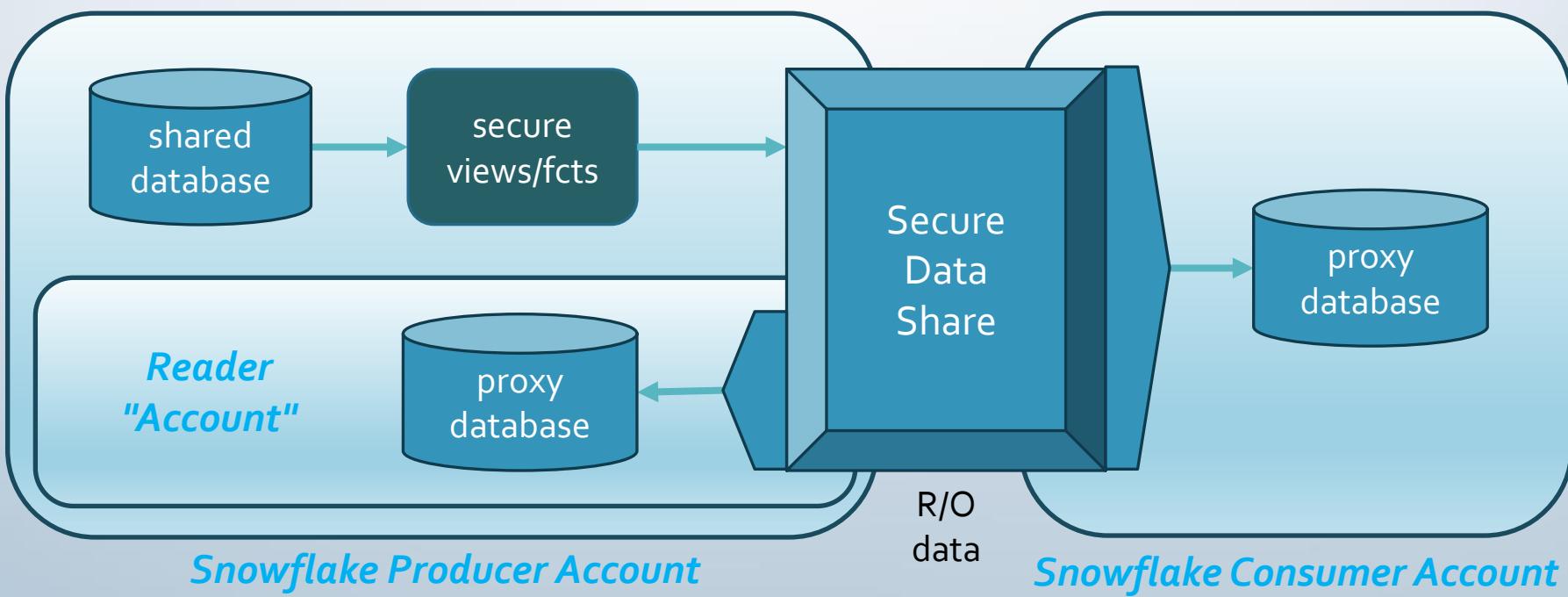


Secure Functions and Views

- *Secure UDFs/Store Procedures*
 - **CREATE SECURE FUNCTION** ..., **IS_SECURE** field
 - Users cannot see code definitions (body, header info, imports...)
 - No internal optimizations (may be slower), avoid push-down
 - No exposed amount of data scanned, in queries
- *Secure Views/Materialized Views*
 - **CREATE SECURE VIEW** ..., **IS_SECURE** field
 - Users cannot see view definitions (base tables...)
 - No user access to underlying data (function calls...)
 - No internal optimizations (may be slower)
 - No exposed amount of data scanned, in queries

Reader Accounts

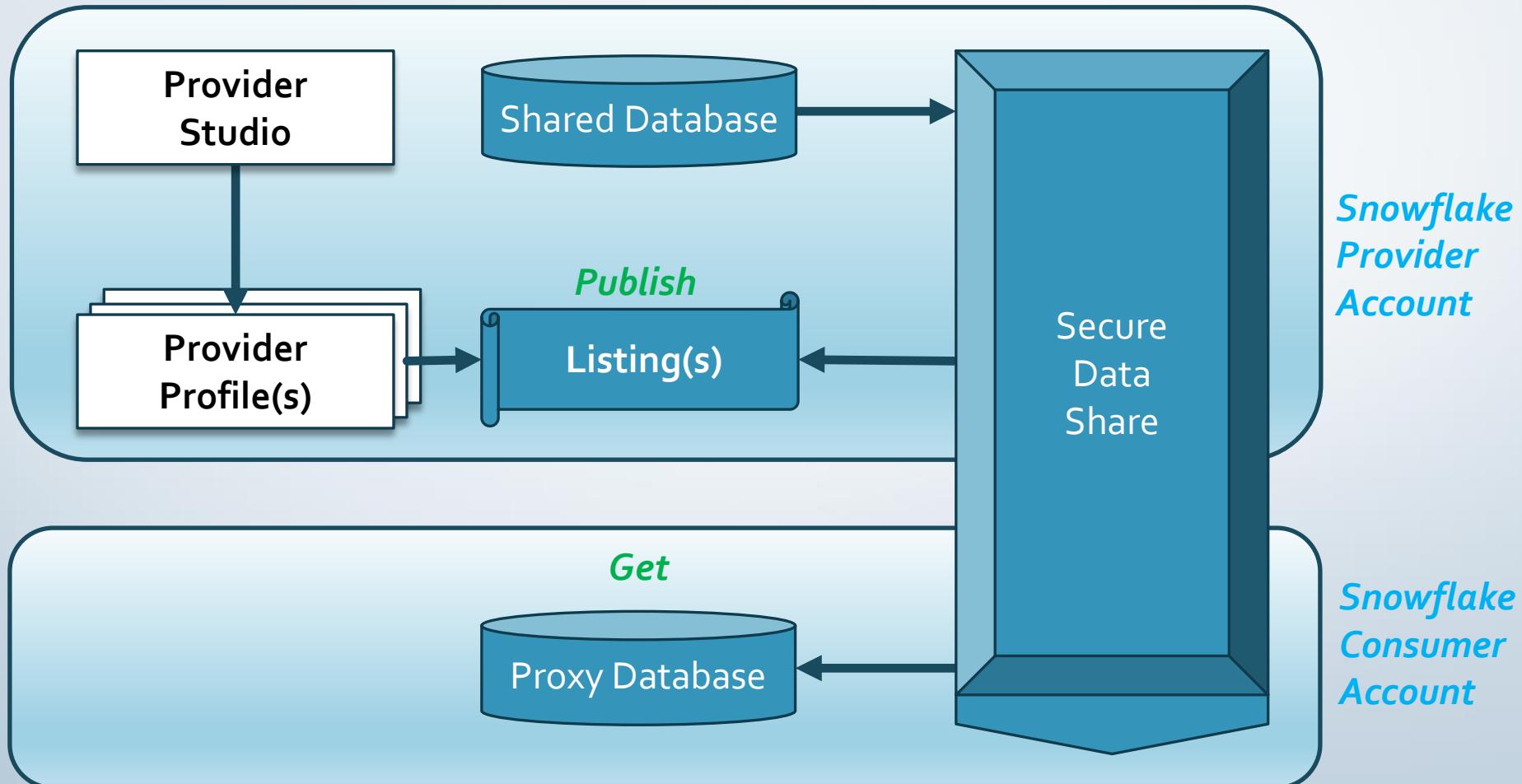
- **CREATE MANAGED ACCOUNT** name **TYPE = READER** → locator + URL (same region and edition)
 - **ADMIN_NAME/PASSWORD** ← username/password
 - or *Data > Private Sharing > Reader Accounts*
- *features*
 - cannot modify existing data or create database objects
 - cannot upload new data or unload data through storage integrations
 - can create users/roles/warehouses/rmons + shared databases (on the inbound shares ← cannot see anything else)



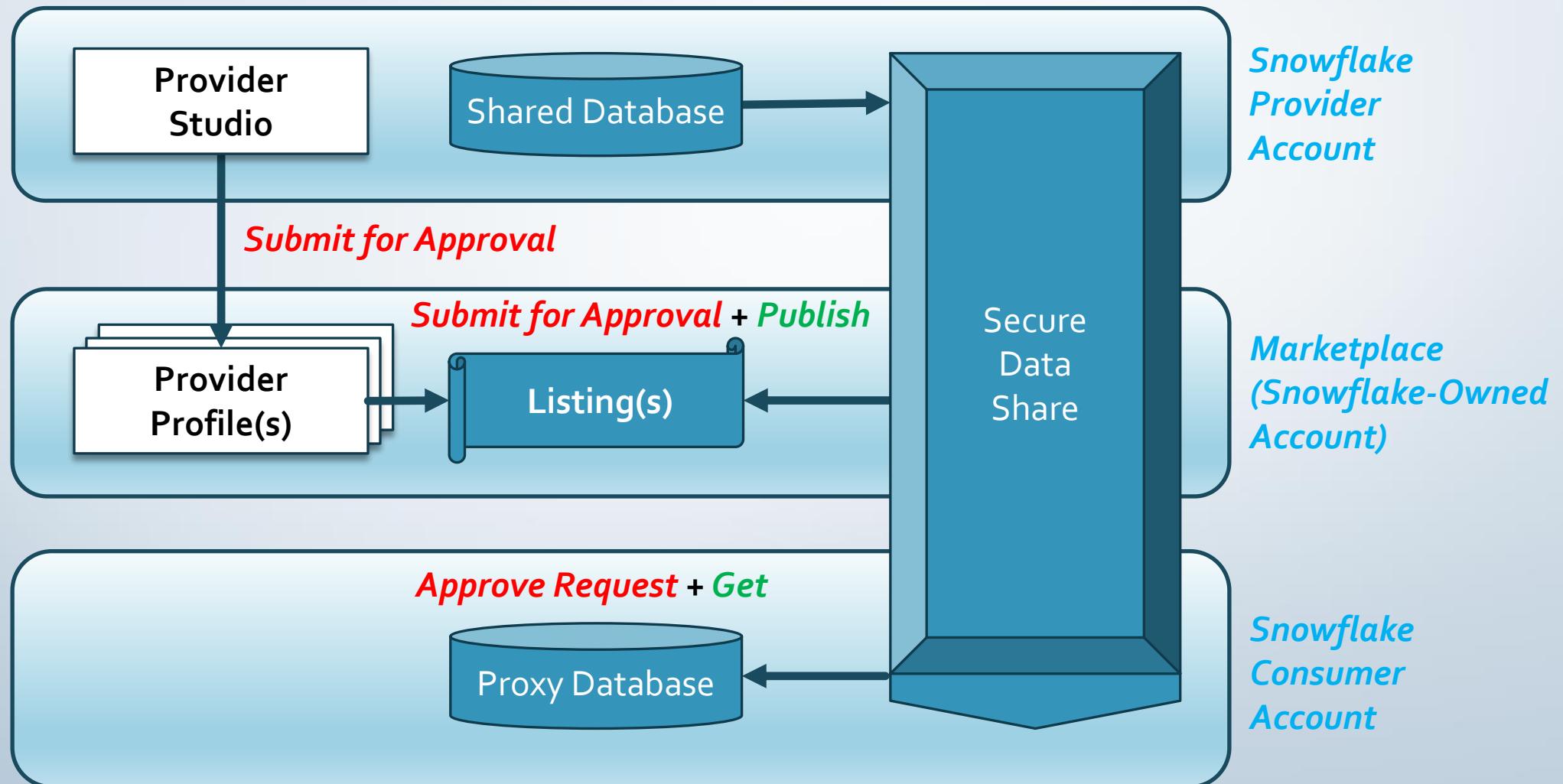
Private/Public Shares: Listings

- ***private share*** = Data Exchange
 - w/ other specific consumers (separate/reader accounts)
 - no need for approvals
 - can share data through secure views/functions, or native apps
- ***public share*** = Snowflake Marketplace
 - public, for everybody
 - needs approval for Provider Profile + each published Listing
 - may offer free shares w/ ***Get*** or ***Request*** (wait for approval)
 - free or could monetize

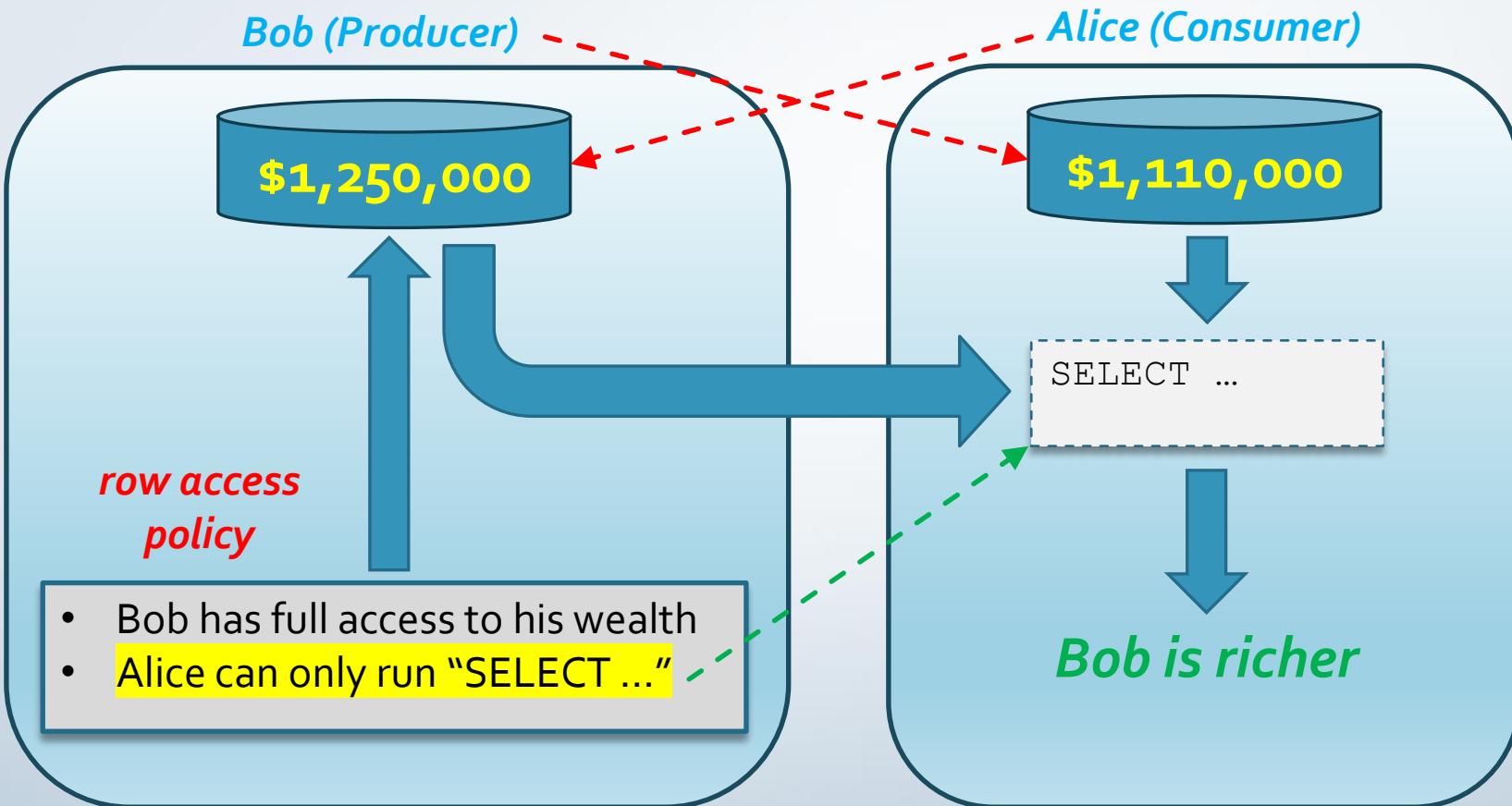
Private Share: Data Exchange



Public Share: Snowflake Marketplace



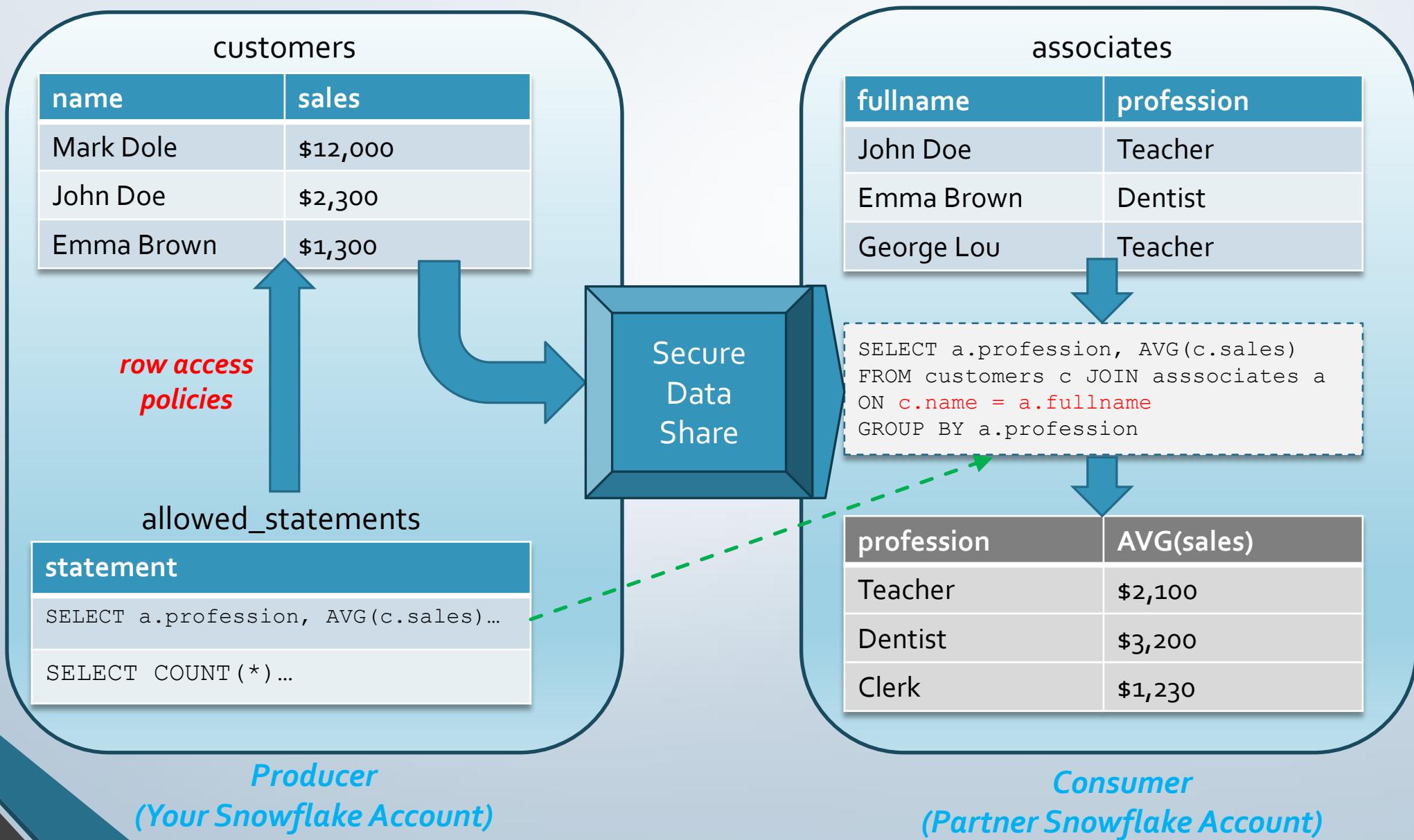
Data Clean Room: Yao's Millionaire Problem



Data Clean Room: Design Steps

- The producer creates and attaches a **row access policy** on its table.
- The policy allows only the producer to get **full access** to its data.
- The policy may allow a consumer role to run some **allowed statements**.
- The consumer must run the **exact statements** allowed by the producer.
- Any other statement run by the consumer will return **no data**.
- The producer will have **no access** to any consumer data, at any time.

Data Clean Room: with Secure Data Share



A professional man in a dark suit, light blue shirt, and patterned tie is shown from the waist up, looking down at a clipboard he is holding. He is wearing glasses and has a pen in his pocket. The background behind him is white.

Client Request

Could you extend your Hierarchical Data Viewer to render Snowflake metadata as well?

We're interested in better data visualizations in these particular areas:

- * Entity-relationship diagrams
- * Users and the role hierarchy
- * Task dependencies
- * Data lineage
- * Database object dependencies

Review of Client Request

Could you extend your Hierarchical Data Viewer to render Snowflake metadata as well?

We're interested in better data visualizations in these particular areas:

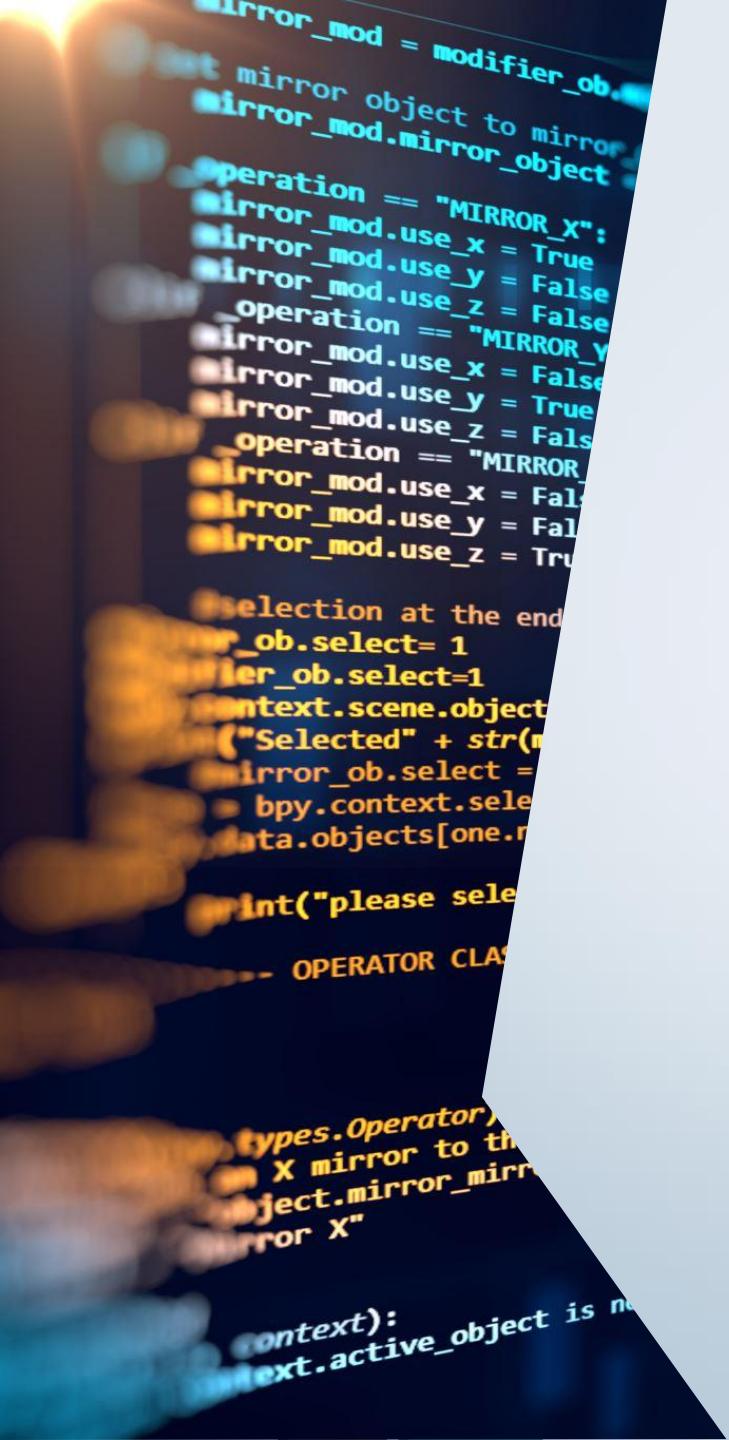
- * Entity-relationship diagrams
- * Users and the role hierarchy
- * Task dependencies
- * Data lineage
- * Database object dependencies



Section Summary



- Information Schema vs Account Usage
- Table Constraints
- Entity-Relationship Diagrams
- Users and Roles
- Task Workflows & Task Runs
- Data Lineage
- Object Dependencies



Information Schema vs Account Usage

- <db>.INFORMATION_SCHEMA
 - local to a database
 - no dropped objects
 - 7 days..6 months retention time
 - instant data access
- read-only views & table functions
- extensive metadata info
- SNOWFLAKE.ACOUNT_USAGE
 - global, in the Snowflake app
 - dropped objects included
 - 1 year retention time
 - 45 min..3h data latency (2h mostly)
- mirror Info schema views/functions
- many historical views
- metering history (cost info)
- READER_ACCOUNT_USAGE
- ORGANIZATION_USAGE

Information Schema Views

- **Inventory**
 - Tables, Columns, Views, Event_Tables, External_Tables
 - Databases, Stages, Sequences, Pipes, File_Formats
 - Replication_Databases, Replication_Groups
- **Programming**
 - Packages, Functions & Procedures
 - Class_Instances, Class_Instance_Functions, Class_Instance_Procedures
- **Constraints**
 - Table_Constraints, Referential_Constraints
- **Roles & Privileges**
 - Enabled_Roles, Applicable_Roles
 - Applied_Privileges, Usage_Privileges, Object_Privileges, Table_Privileges
- **Metrics**
 - Table_Storage_Metrics, Load_History

Account Usage Views (Historical Only)

- **Query_History**, **Access_History**, Login_History
- Alert_History, **Task_History**, Serverless_Task_History
- **Copy_History**, **Load_History**, Pipe_Usage_History
- Data_Transfer_History
- Metering_History, Metering_Daily_History
- Warehouse_Load_History, Warehouse_Metering_History
- Storage_Usage
- **Object_Dependencies**
- Tag_References
- Sessions

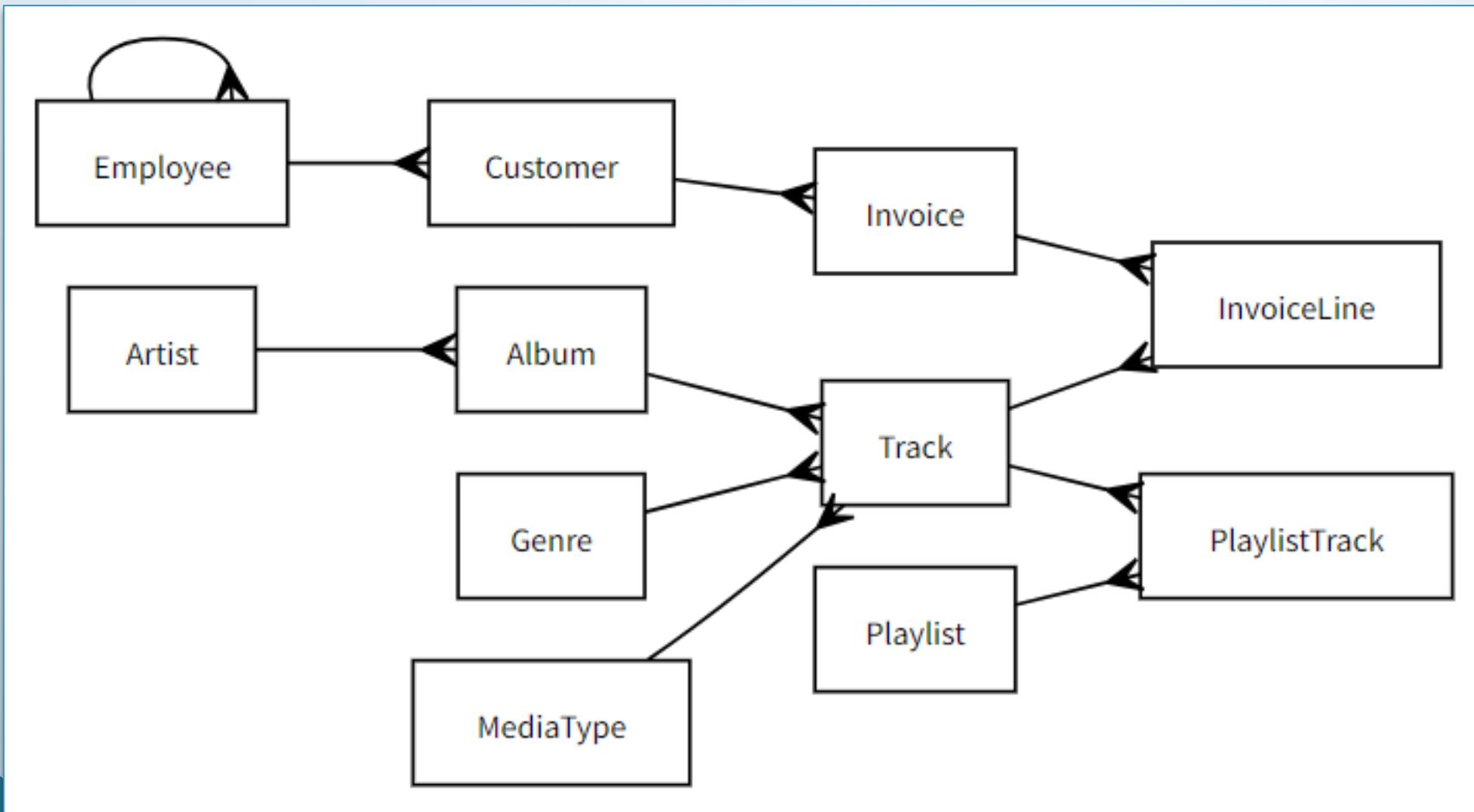
Table Constraints

- **NOT NULL** = the only one always enforced.
- **PRIMARY KEY** (PK) = for referential integrity, as unique table row identifier, never enforced.
- **FOREIGN KEY** (FK) = for referential integrity, as propagation of a PK, never enforced.
- **UNIQUE** = for unique combination of column values, other than the PK, never enforced.
- **ENFORCED/DEFERRABLE/INITIALLY** = never enforced.
- **MATCH/UPDATE/DELETE** = for FK only, never enforced.
- **CLUSTERING KEYS** = optional, similar to PKs, but used for better micro-partitioning, not referential integrity.

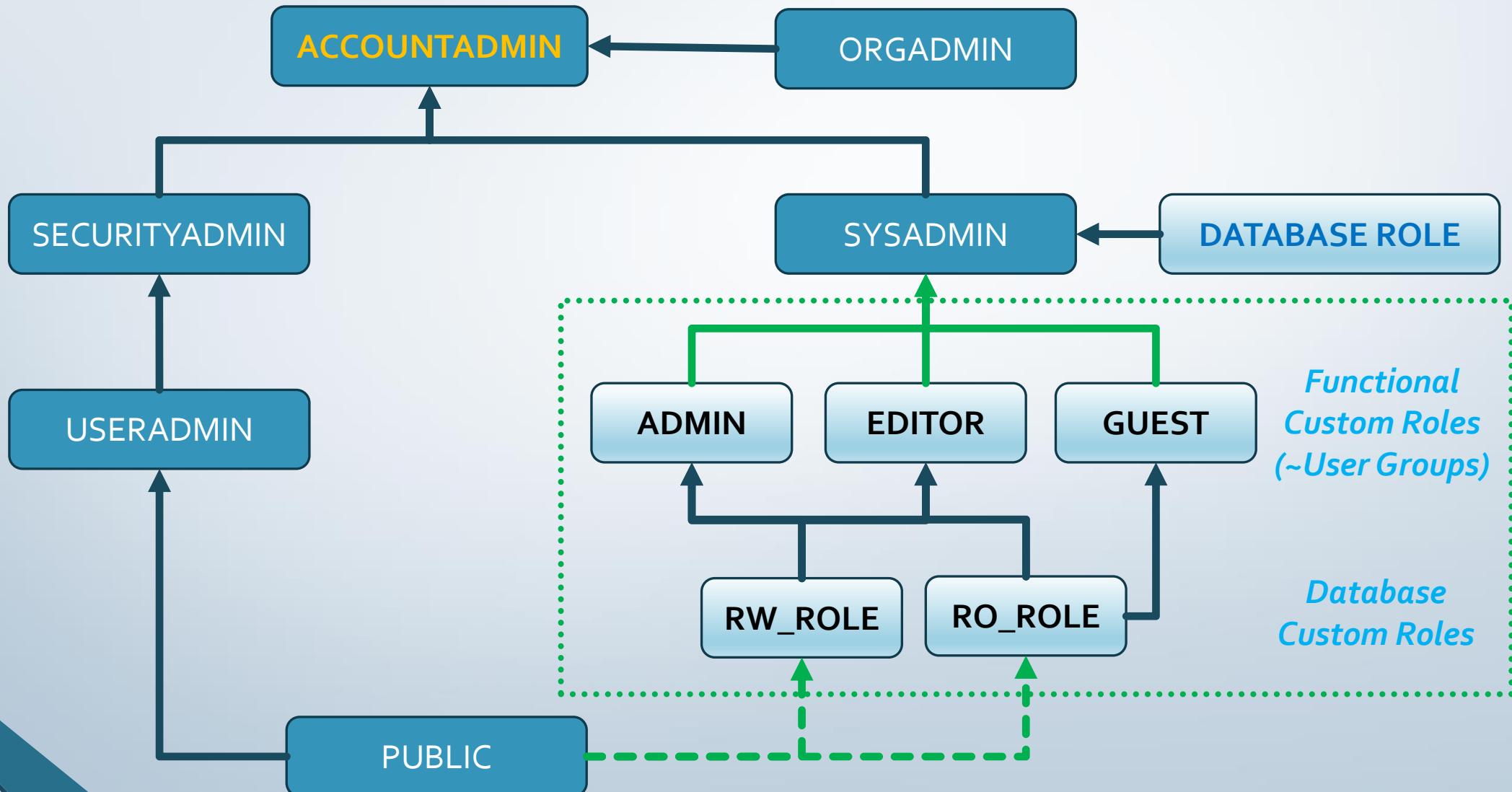
ER (Entity-Relationship) Diagrams

- **SHOW DATABASES** — database names
- **SHOW SCHEMAS IN DATABASE <db>** — database schemas
- **SHOW TABLES IN SCHEMA <db>.<sch>** — schema tables
- **SHOW COLUMNS IN SCHEMA <db>.<sch>** — table columns
- **SHOW UNIQUE KEYS IN SCHEMA <db>.<sch>** — UNIQUE constraints
- **SHOW PRIMARY KEYS IN SCHEMA <db>.<sch>** — PK constraints
- **SHOW IMPORTED KEYS IN SCHEMA <db>.<sch>** — FK constraints

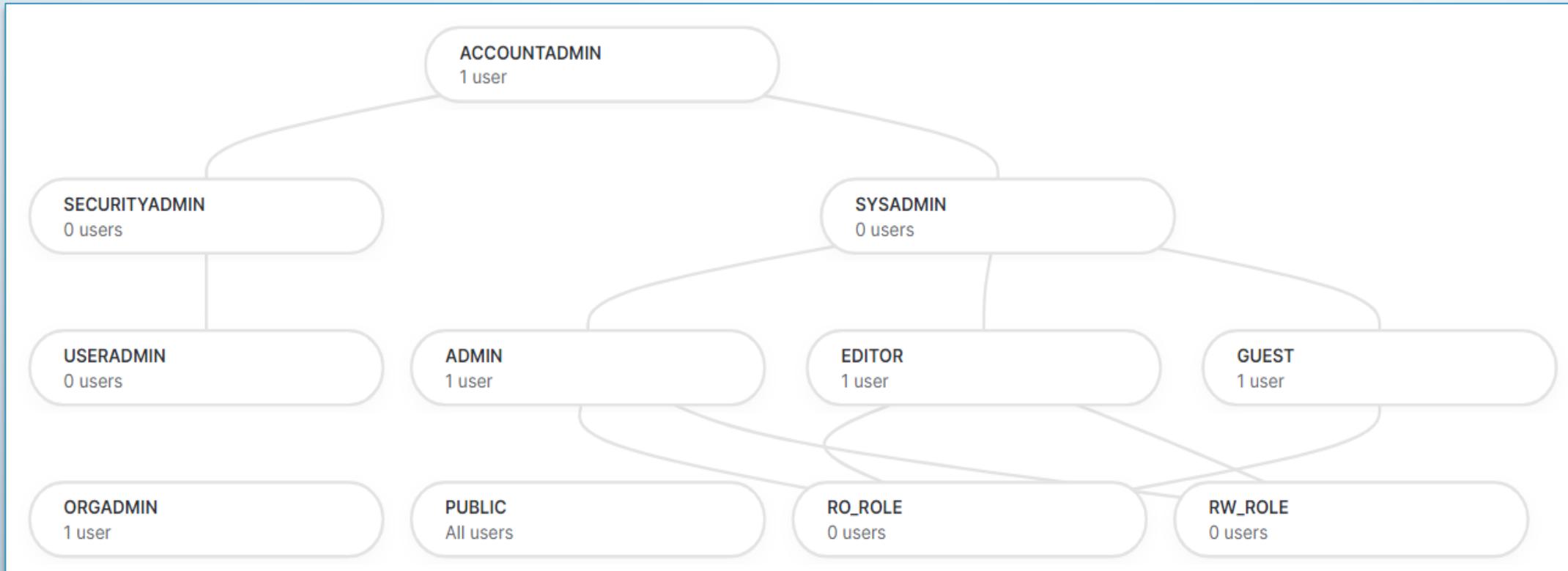
ER (Entity-Relationship) Diagram



Security: The Role Hierarchy



Security: Roles in Snowsight



Security: Parsing Users and Roles

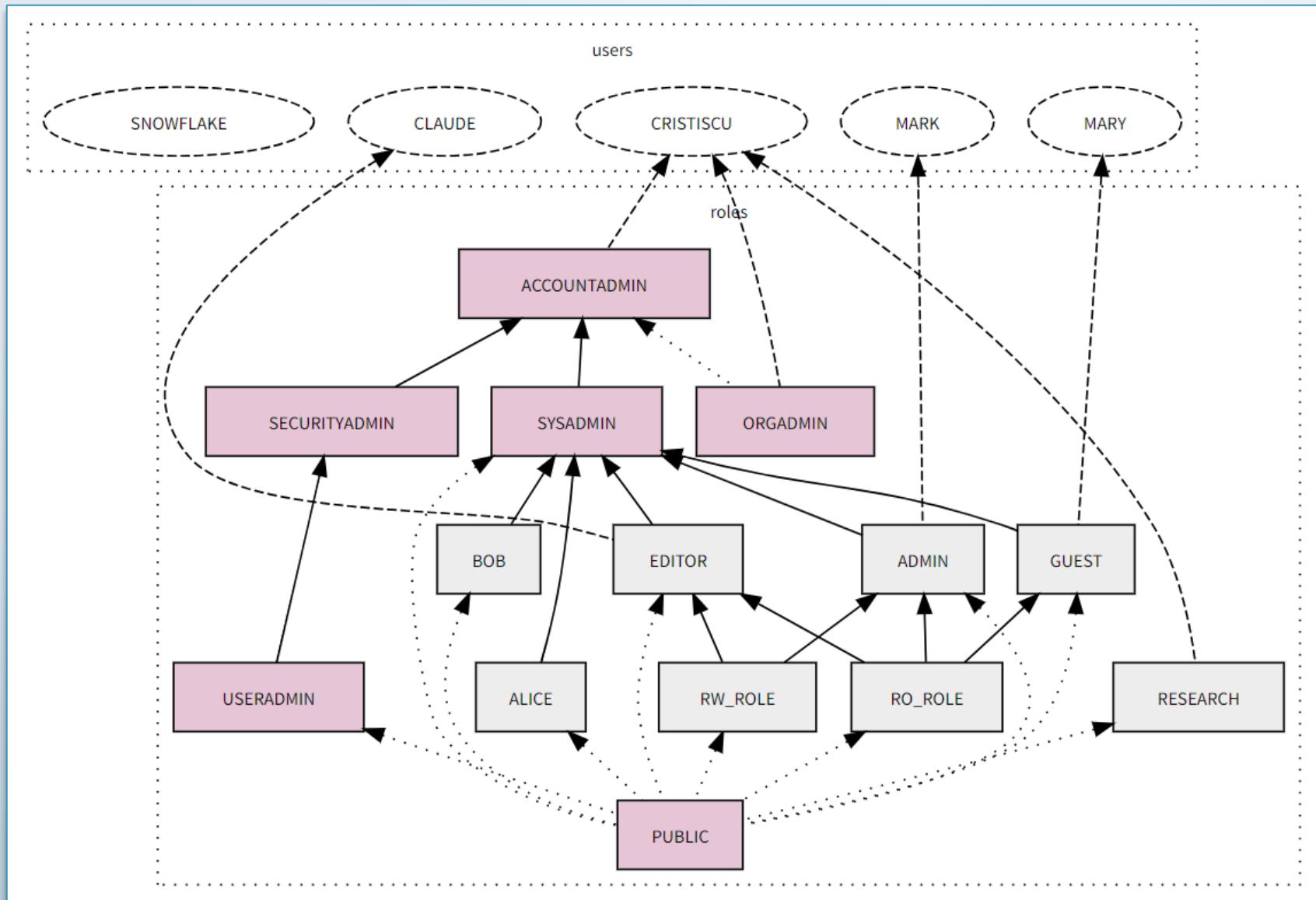
```
# users
users = {}
rows = runQuery("show users")
for row in rows:
    users[str(row["name"])] = []
```

```
# user roles
for user in users:
    rows = runQuery(f'show grants to user "{user}"')
    for row in rows:
        users[user].append(str(row["role"]))
```

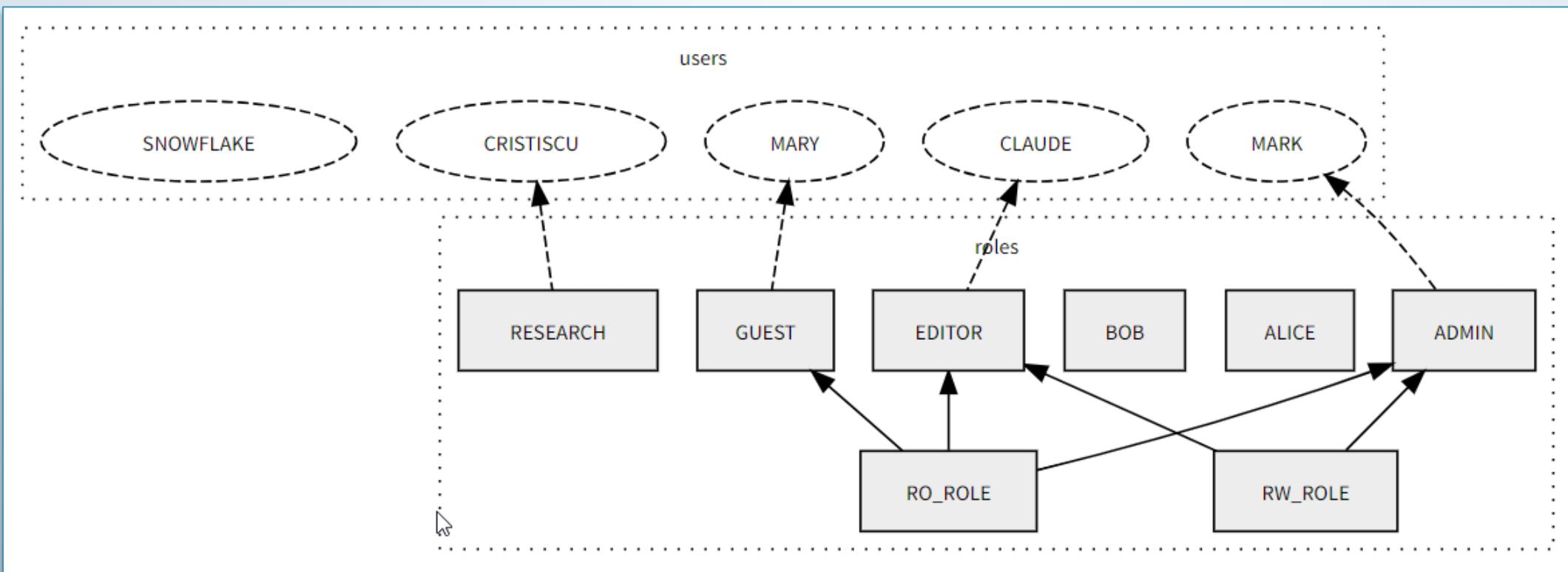
```
# roles
roles = {}
rows = runQuery("show roles")
for row in rows:
    roles[str(row["name"])] = []
```

```
# role hierarchy
for role in roles:
    rows = runQuery(f'show grants to role "{role}"')
    for row in rows:
        if (str(row["privilege"]) == "USAGE"
            and str(row["granted_on"]) == "ROLE"):
            roles[role].append(str(row["name"]))
```

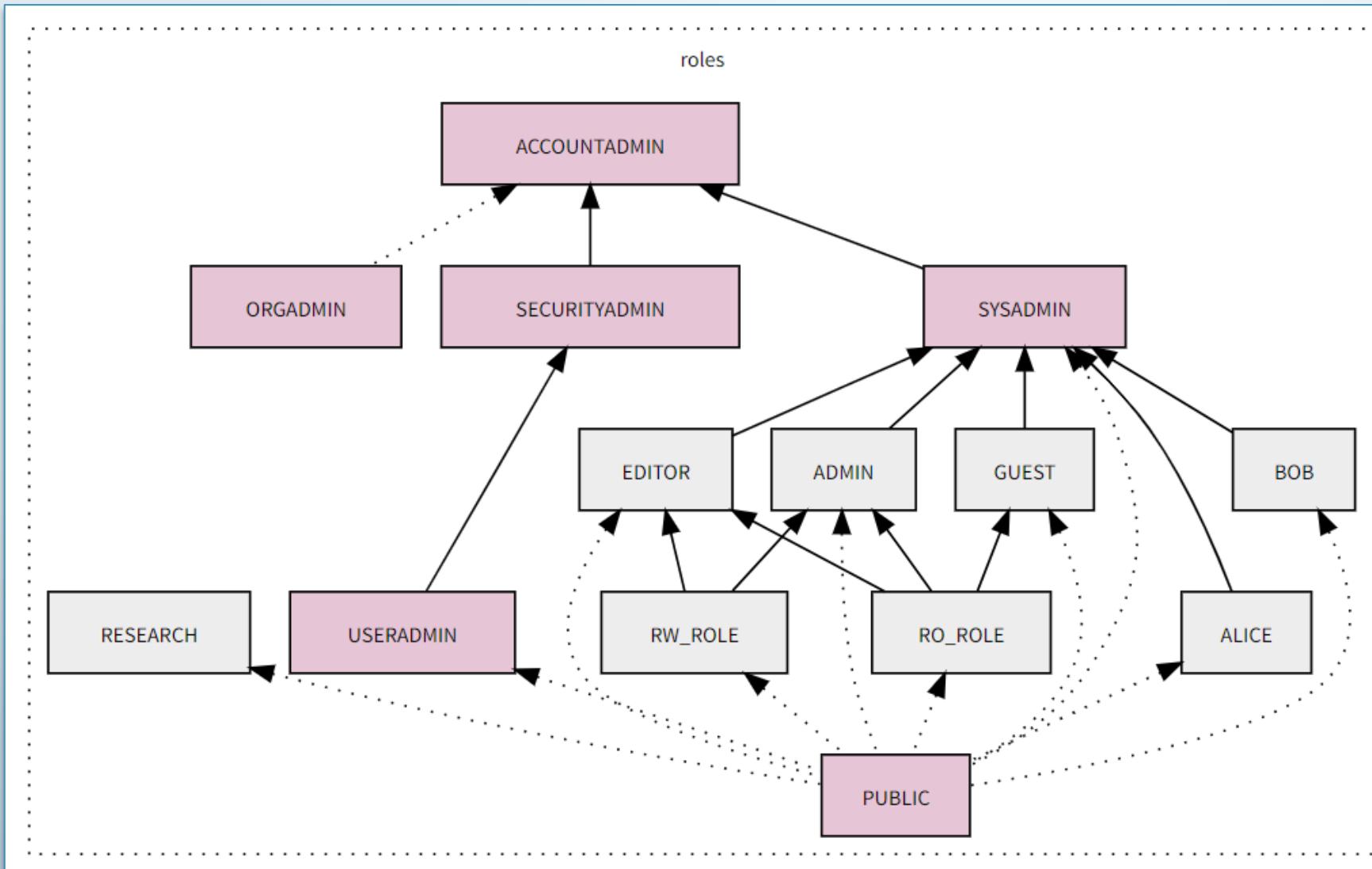
Security: All Users and Roles



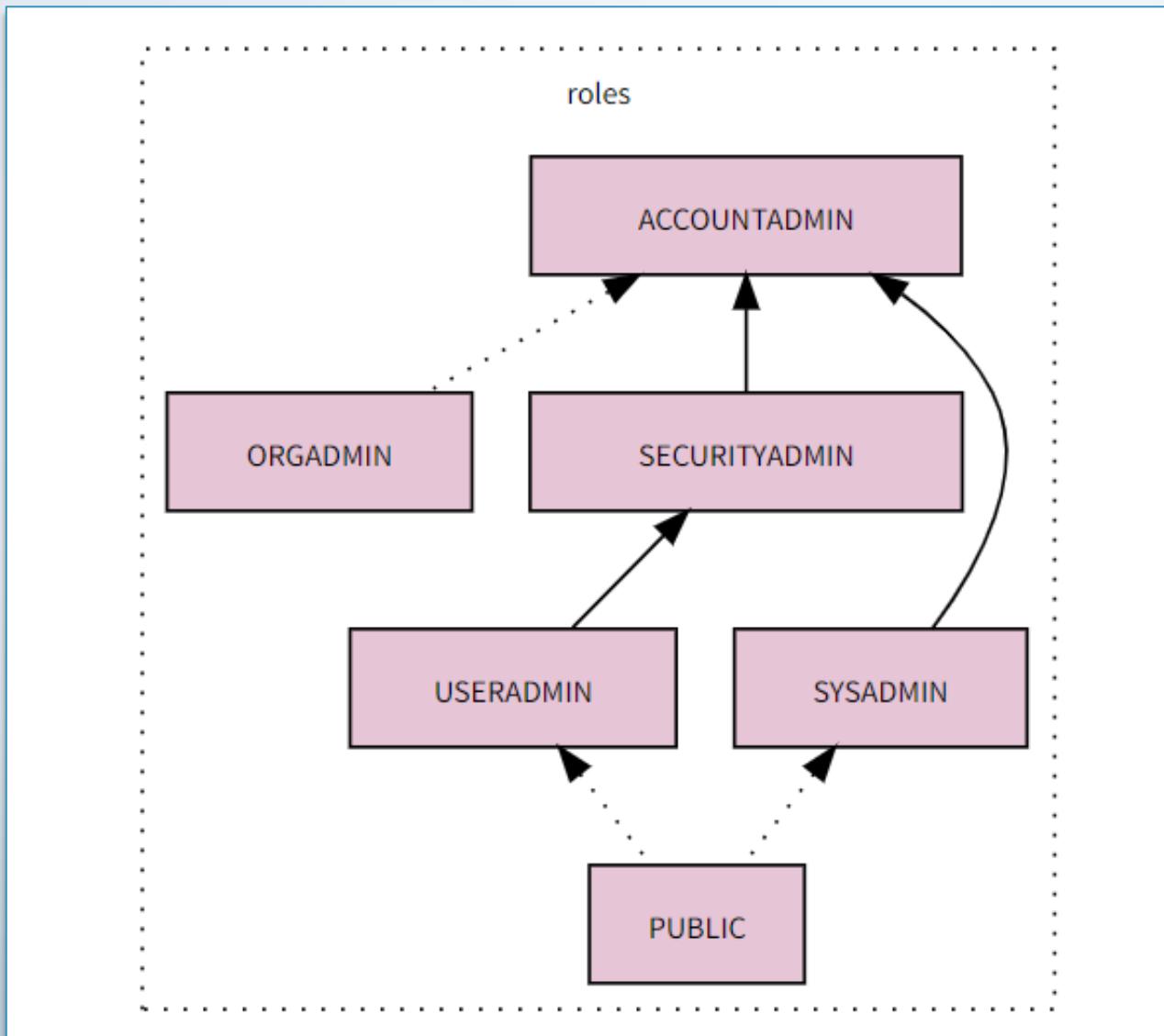
Security: All Users and Custom Roles



Security: All Custom and System-Defined Roles



Security: All System-Defined Roles



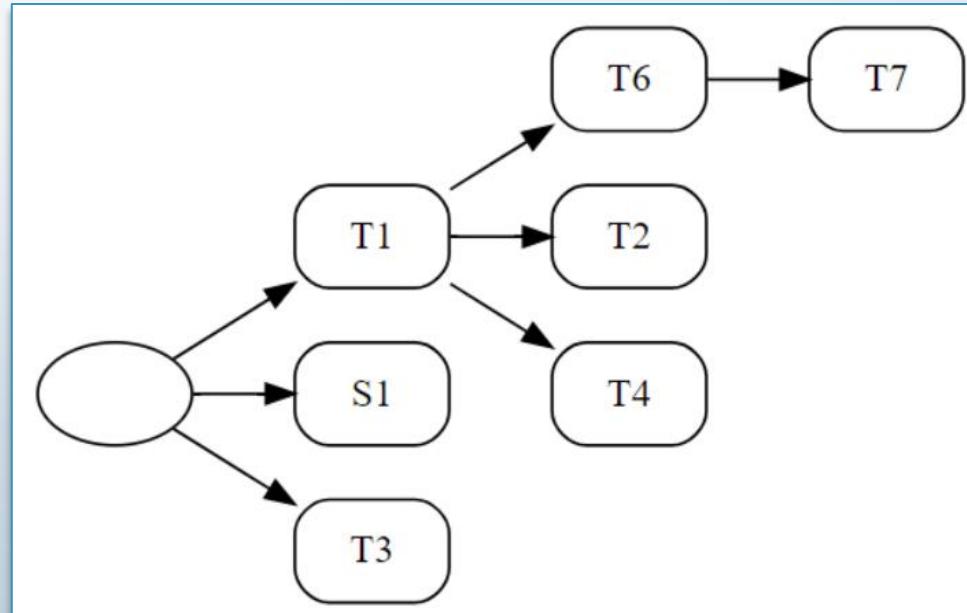
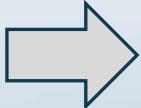
Task Workflows (DAGs)

- **CREATE TASK** name ... AS ...
 - **AFTER** ..., ... ← parent task(s)
 - **CONFIG** ← key-value pairs accessed by all tasks in a DAG
 - **ALLOW_OVERLAPPING_EXECUTION** ← cocurrent DAGs
- **SHOW TASKS** ... IN SCHEMA ...
- SYSTEM\$**TASK_DEPENDENTS_ENABLE**(name)
 - enable all children before DAG run
- INFORMATION_SCHEMA.**TASK_HISTORY**(task_name=>name))
 - show all task runs, with errors and status: COMPLETED/FAILED/SCHEDULED
 - sort DESC by RUN_ID to see most recent runs

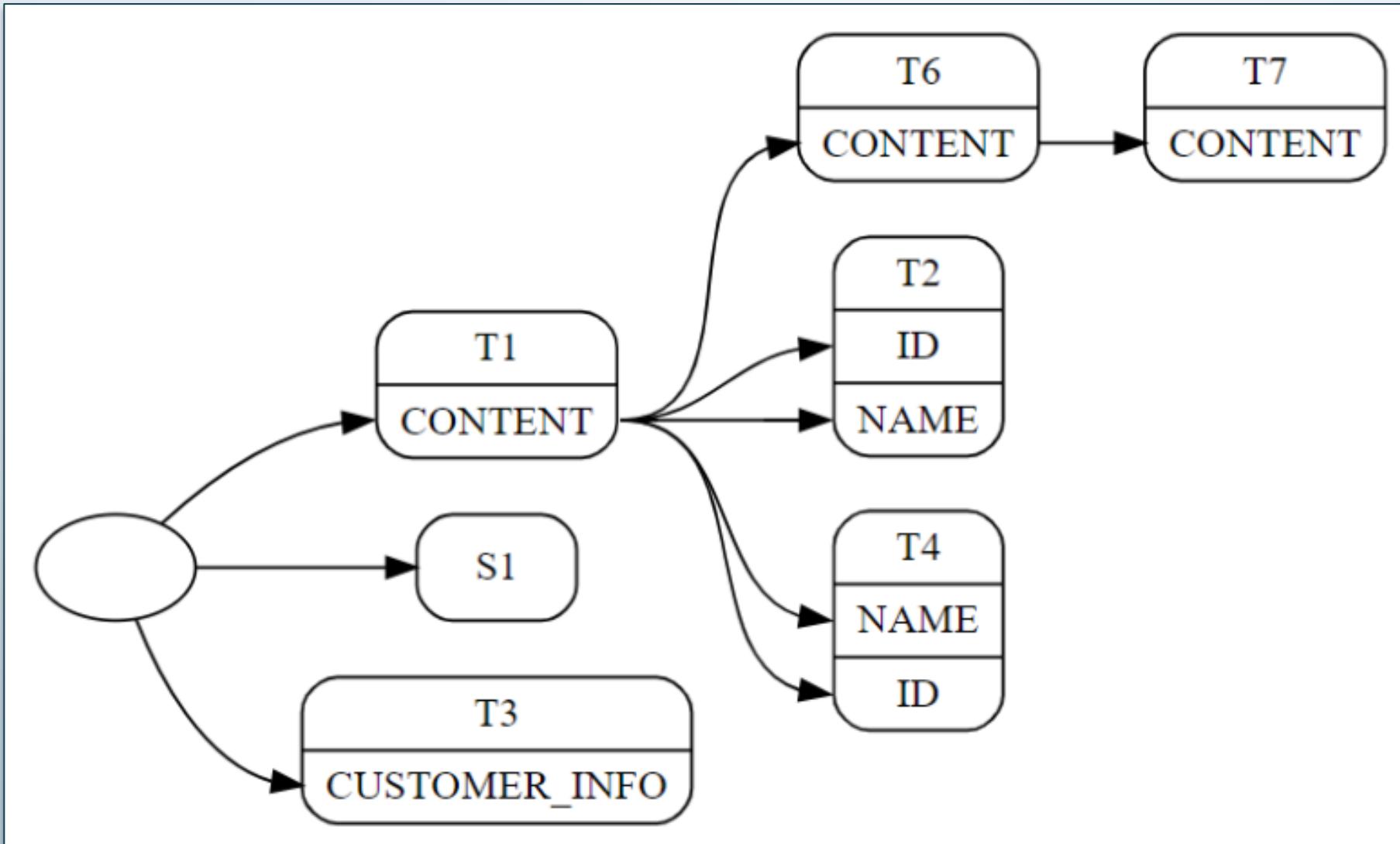
Data Lineage: Table-Level Views

```
select distinct
    substr(directSources.value:objectName, len($SCH)+2) as source,
    substr(object_modified.value:objectName, len($SCH)+2) as target
from snowflake.account_usage.access_history ah,
    lateral flatten(input => objects_modified) object_modified,
    lateral flatten(input => object_modified.value:"columns", outer => true) cols,
    lateral flatten(input => cols.value:directSources, outer => true) directSources
where directSources.value:objectName like $SCH || '%'
    or object_modified.value:objectName like $SCH || '%'
```

SOURCE	TARGET
T1	T2
T6	T7
null	T3
T1	T4
null	T1
T1	T6
null	S1



Data Lineage: Column-Level Graph View



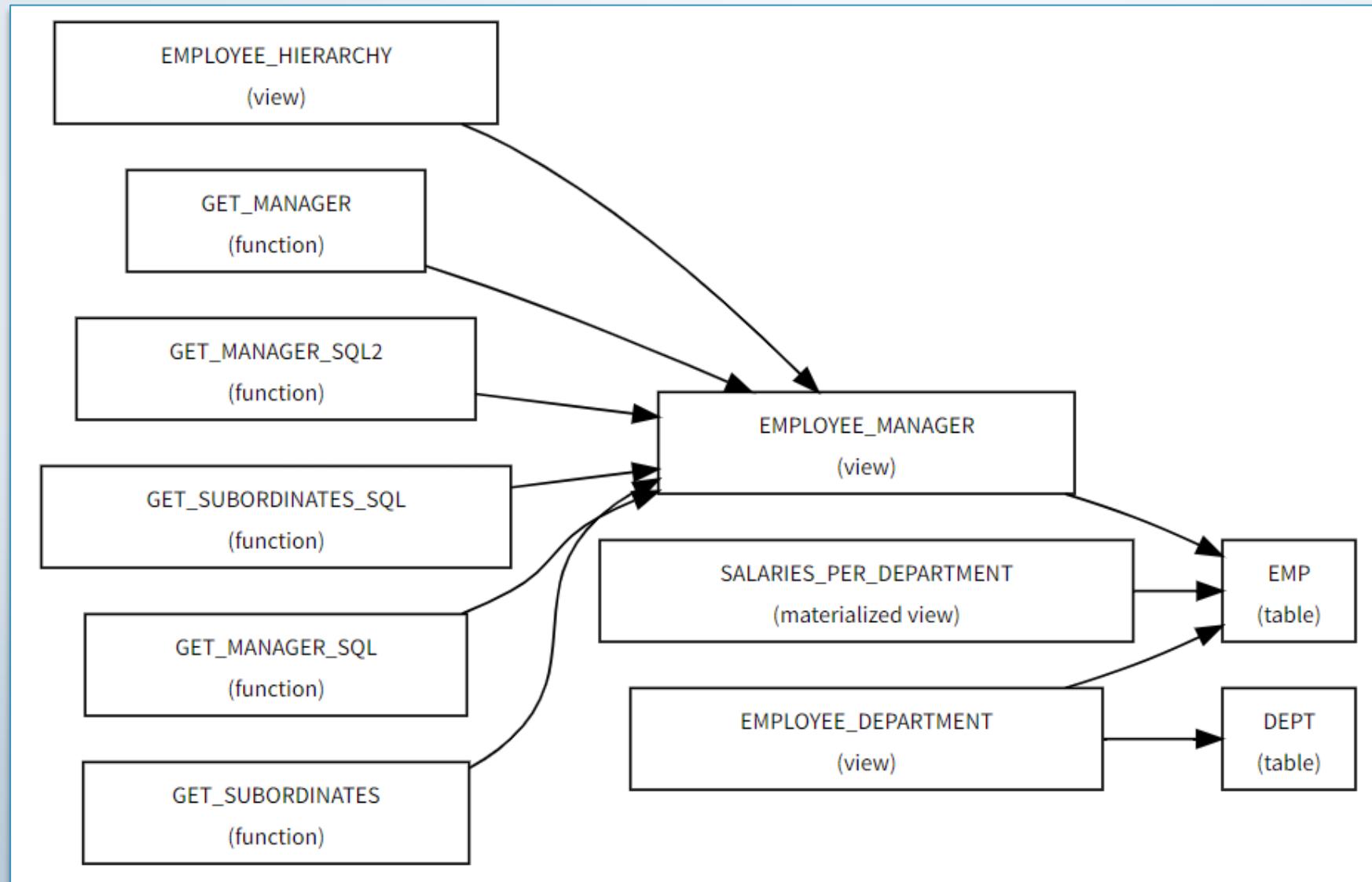
OBJECT_DEPENDENCIES View

- **REFERENCING → REFERENCED** object (pairs)
 - object name / id / domain (type)
 - database + schema
- **DEPENDENCY_TYPE**
 - **BY_NAME** = view/UDF... → view/UDF...
 - **BY_ID** = ext stage → storage integration, stream → table/view
 - **BY_NAME_AND_ID** = materialized view → table

Object Dependencies: Tabular View

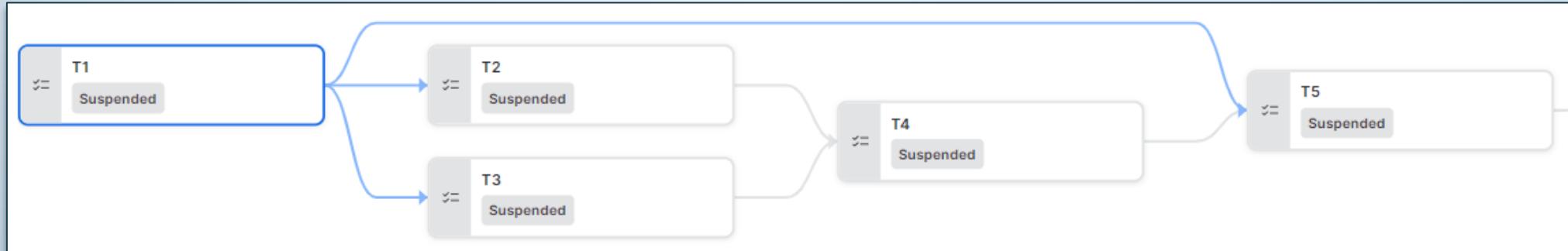
REFING_NAME	REFING_TYPE	...	REFED_NAME	REFED_TYPE
EMPLOYEE_DEPARTMENT	view		EMP	table
EMPLOYEE_MANAGER	view		EMP	table
EMPLOYEE_DEPARTMENT	view		DEPT	table
EMPLOYEE_HIERARCHY	view		EMPLOYEE_MANAGER	view
SALARIES_PER_DEPARTMENT	materialized view		EMP	table
GET_MANAGER	function		EMPLOYEE_MANAGER	view
GET_MANAGER_SQL2	function		EMPLOYEE_MANAGER	view
GET_SUBORDINATES_SQL	function		EMPLOYEE_MANAGER	view
GET_MANAGER_SQL	function		EMPLOYEE_MANAGER	view
GET_SUBORDINATES	function		EMPLOYEE_MANAGER	view

Object Dependencies: Graph View



Task Dependencies: Initial DAG Topology

```
create task t1 as select SYSTEM$WAIT(1);
create task t2 after t1 as select SYSTEM$WAIT(2);
create task t3 after t1 as select SYSTEM$WAIT(1);
create task t4 after t2, t3 as select SYSTEM$WAIT(1);
create task t5 after t1, t4 as select SYSTEM$WAIT(1);
create task t6 after t5 as select SYSTEM$WAIT(1);
create task t7 after t6 as select SYSTEM$WAIT(1);
create task t8 after t6 as select SYSTEM$WAIT(2);
```

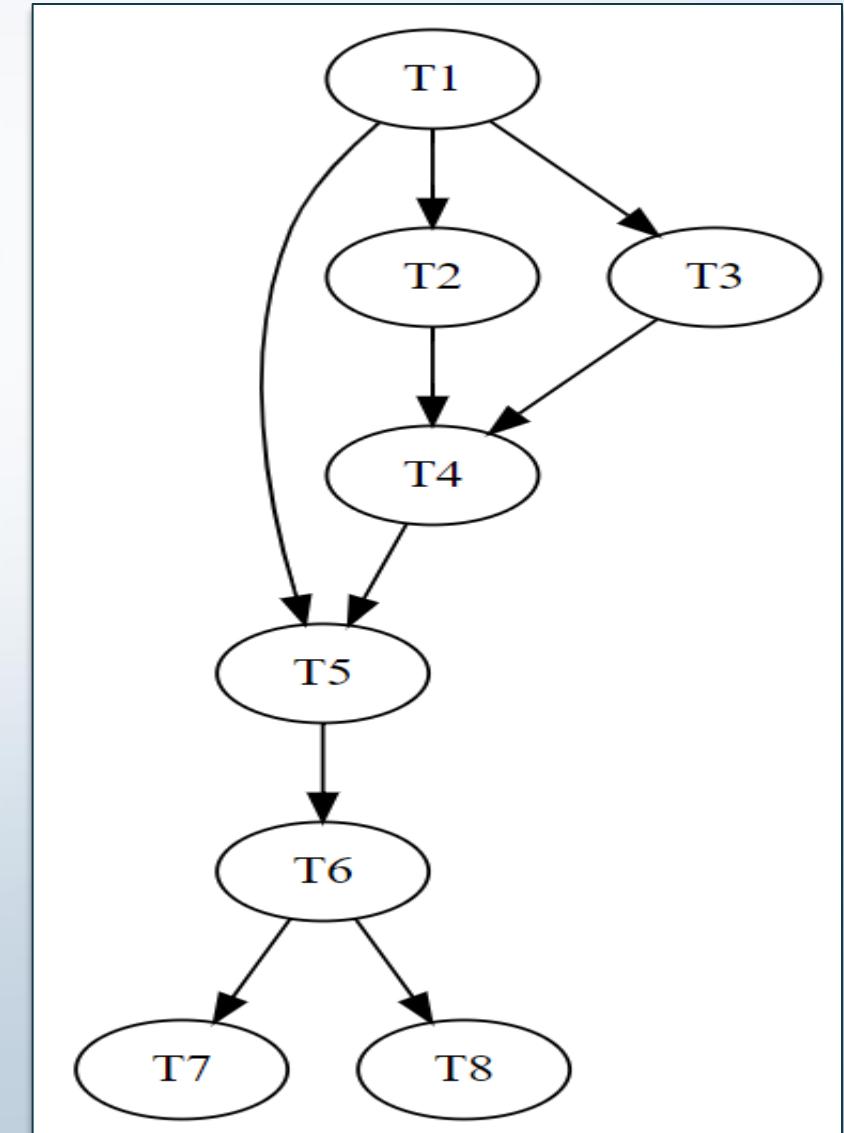


Task Dependencies : Task-Parent Pairs

```
show tasks in schema tasks.public;
select t."name" task,
       split_part(p.value::string, '.', -1) parent
  from table(result_scan(last_query_id())) t,
lateral flatten(input => t."predecessors",
outer => true) p;
```

TASK	PARENT
T1	null
T2	T1
T3	T1
T4	T2
T4	T3
T5	T1
T5	T4
T6	T5
T7	T6
T8	T6

```
digraph {
    rankdir="BT";
    edge [dir="back"];
    T2 -> T1;
    T3 -> T1;
    T4 -> T2;
    T4 -> T3;
    T5 -> T1;
    T5 -> T4;
    T6 -> T5;
    T7 -> T6;
    T8 -> T6;
}
```



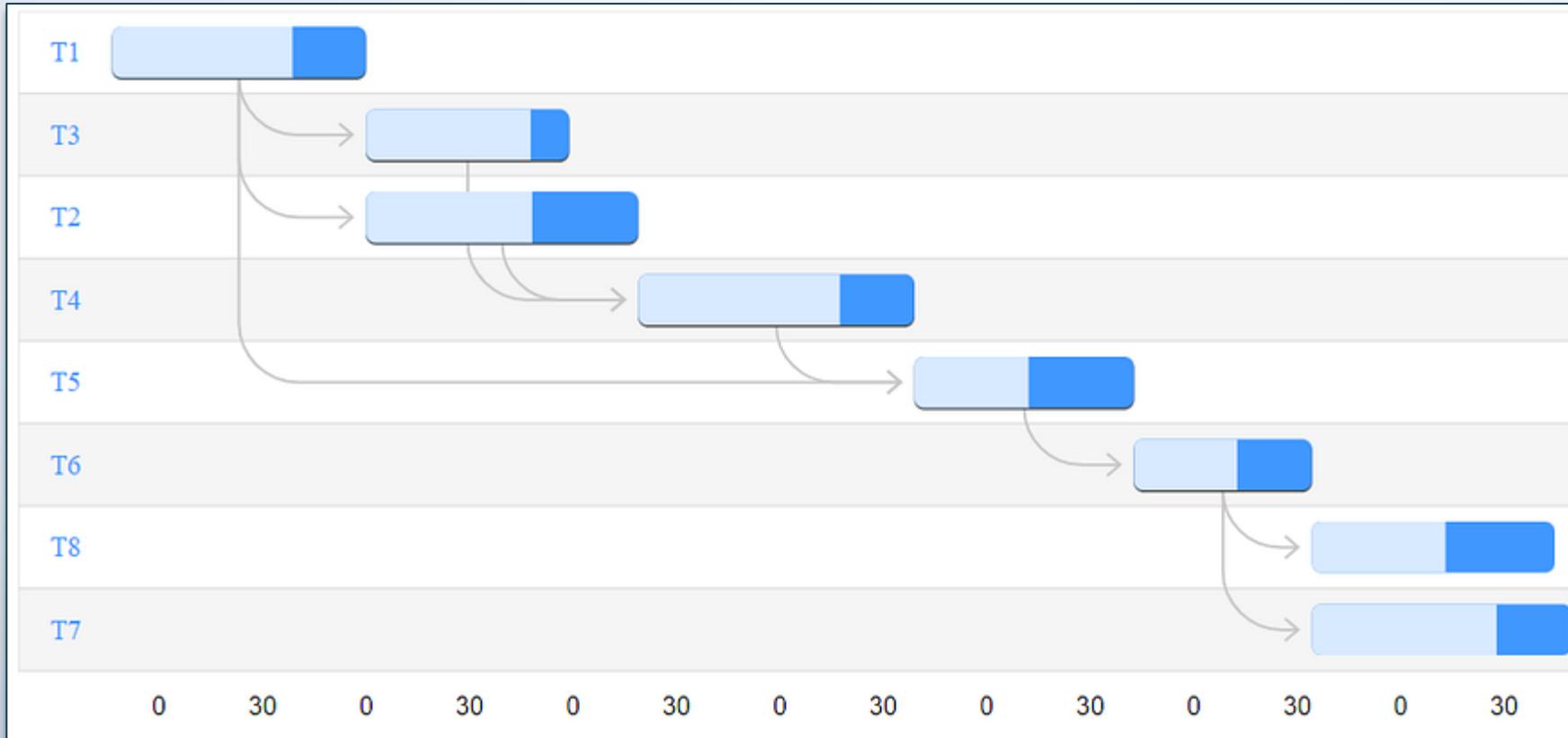
Task Workflows: Examine DAG Task Runs

```
select SYSTEM$TASK_DEPENDENTS_ENABLE('tasks.public.t1');
execute task t1;

select name, state, scheduled_time, query_start_time, completed_time
  from table(information_schema.task_history())
 where run_id = (select top 1 run_id
                  from table(information_schema.task_history(task_name => 'T1'))
                  order by query_start_time desc)
    order by query_start_time;
```

NAME	STATE	SCHEDULED_TIME	QUERY_START_TIME	COMPLETED_TIME
T1	SUCCEEDED	2023-10-30 11:24:33.697 -0700	2023-10-30 11:24:42.964 -0700	2023-10-30 11:24:44.397 -0700
T2	SUCCEEDED	2023-10-30 11:24:44.397 -0700	2023-10-30 11:24:45.713 -0700	2023-10-30 11:24:48.318 -0700
T3	SUCCEEDED	2023-10-30 11:24:44.397 -0700	2023-10-30 11:24:50.843 -0700	2023-10-30 11:24:52.362 -0700
T4	SUCCEEDED	2023-10-30 11:24:52.362 -0700	2023-10-30 11:25:11.291 -0700	2023-10-30 11:25:13.497 -0700
T5	SUCCEEDED	2023-10-30 11:25:13.497 -0700	2023-10-30 11:25:54.346 -0700	2023-10-30 11:25:55.717 -0700
T6	SUCCEEDED	2023-10-30 11:25:55.717 -0700	2023-10-30 11:26:11.000 -0700	2023-10-30 11:26:12.717 -0700
T7	SUCCEEDED	2023-10-30 11:26:12.717 -0700	2023-10-30 11:26:22.808 -0700	2023-10-30 11:26:25.540 -0700
T8	SUCCEEDED	2023-10-30 11:26:12.717 -0700	2023-10-30 11:26:23.537 -0700	2023-10-30 11:26:26.304 -0700

Task Workflows: Gantt Chart with DAG Task Run



A professional man in a dark suit, light blue shirt, and patterned tie is standing on the left side of the image. He is holding a white clipboard with a pen in his right hand and looking down at the paper. The background behind him is a plain, light color.

Client Request

We would like to expose your Hierarchical Data Viewer application to our internal partners. We could publish the app in private, and our partners would subscribe only if they want to.

The app should be able to process any child-parent data pairs from their Snowflake accounts, similar to our employee-manager hierarchy.

Review of Client Request

We would like to expose your Hierarchical Data Viewer application to our internal partners. We could publish the app in private, and our partners would subscribe only if they want to.

The app should be able to process any child-parent data pairs from their Snowflake accounts, similar to our employee-manager hierarchy.



Section Summary



- Snowflake Native App Framework
- Application Package
- Application
- Load and Test Apps in Snowflake
- Publish and Consume Native Apps



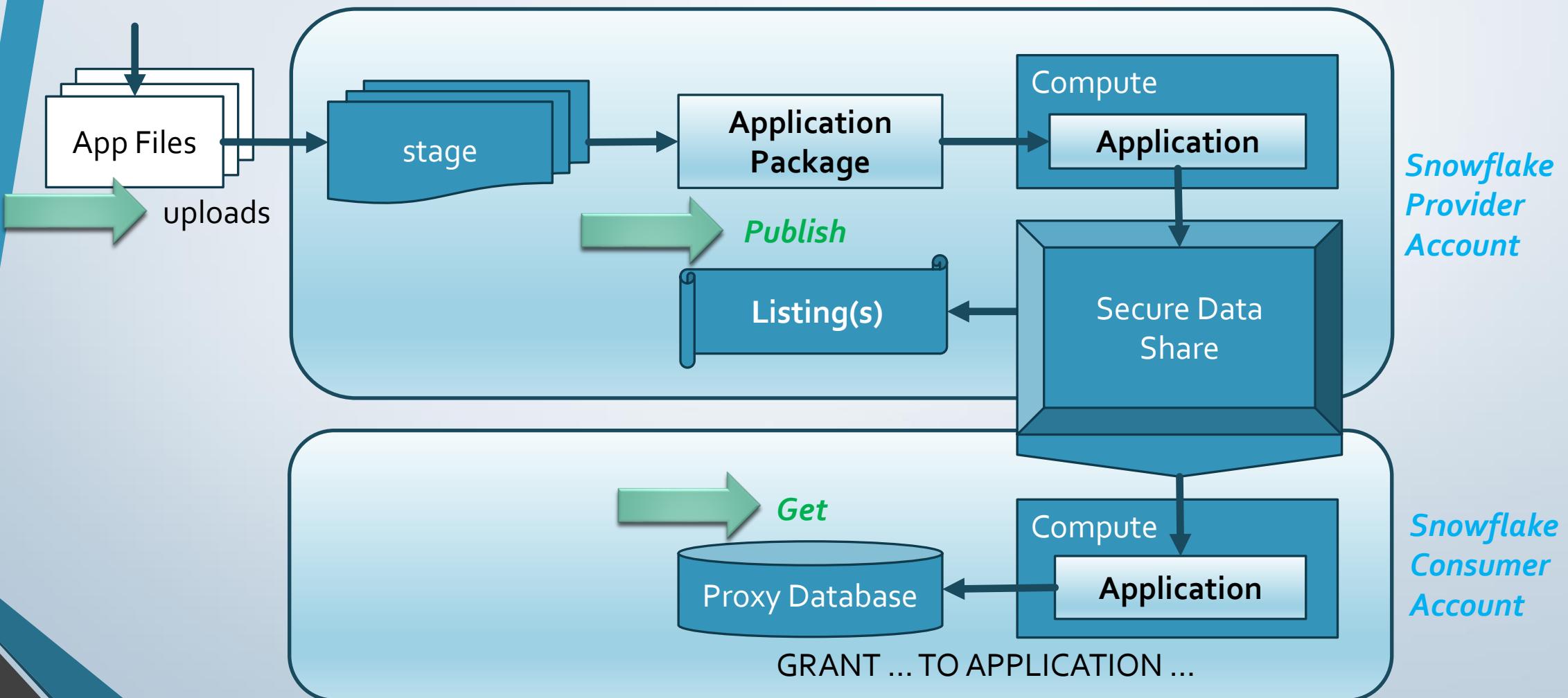
Native App: Prepare and Upload

- *Create and test first as a local Streamlit web app*
 - Create a `script.sql` file, to prepare data on the consumer's side.
 - Create a `readme.md` file, for the first info page of the app.
 - Create a `manifest.yml` file, pointing to the two previous files.
- *In Snowflake*
 - Upload all your app files into a `named stage`.
 - Create an **APPLICATION PACKAGE** with the files uploaded in the stage.
 - Create an **APPLICATION** for this package.
 - Create a **STREAMLIT** object for the code.

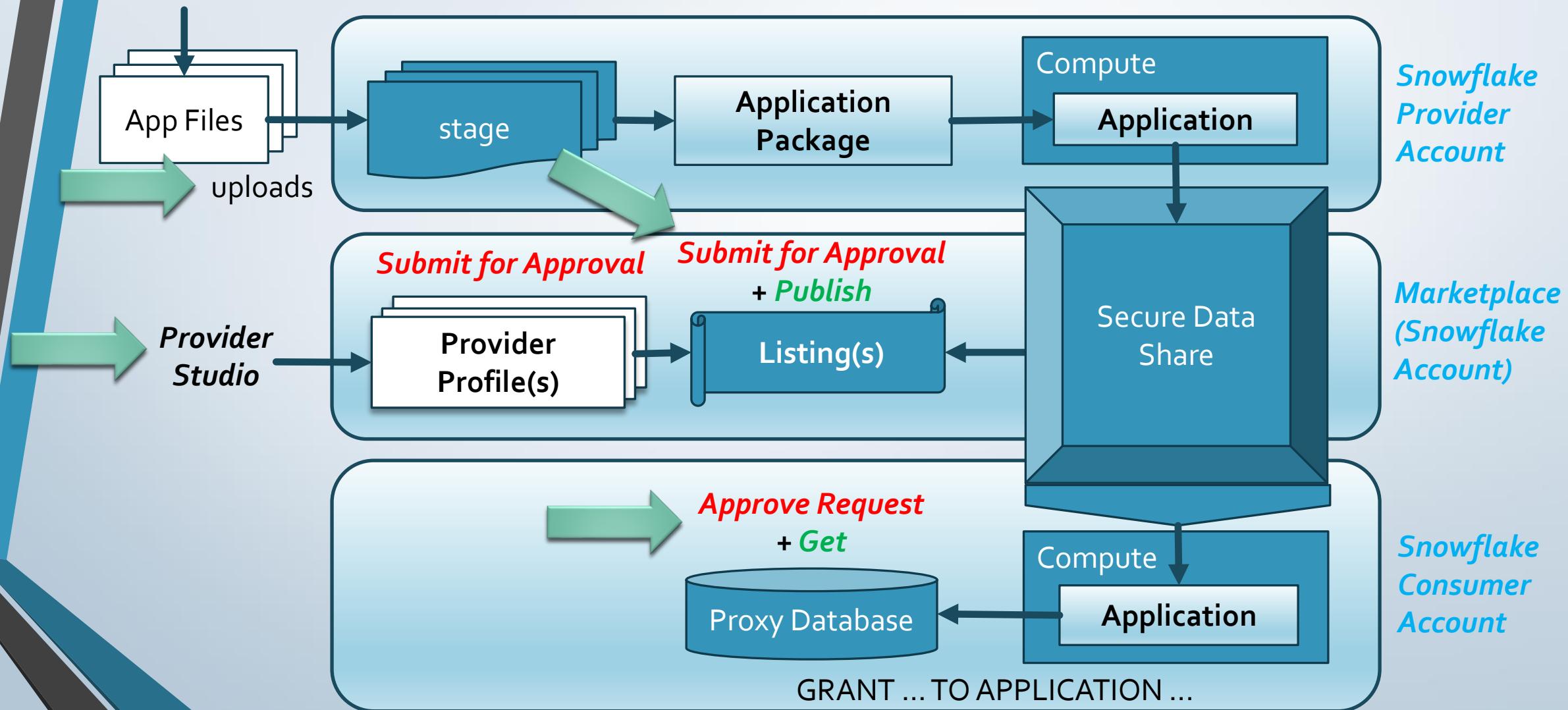
Native App: Test and Deploy

- *In Snowsight*
 - Start your new app in the new *Apps* tab.
 - Connect to Snowflake through *get_active_session()*
 - Continue editing, running, testing the app in Snowsight, as a producer.
- *In the Marketplace/Data Exchange* → public/private share
 - Create [and get approved by Snowflake] a **provider profile**.
 - Publish your app [and get approved in the Marketplace] as a Native App.

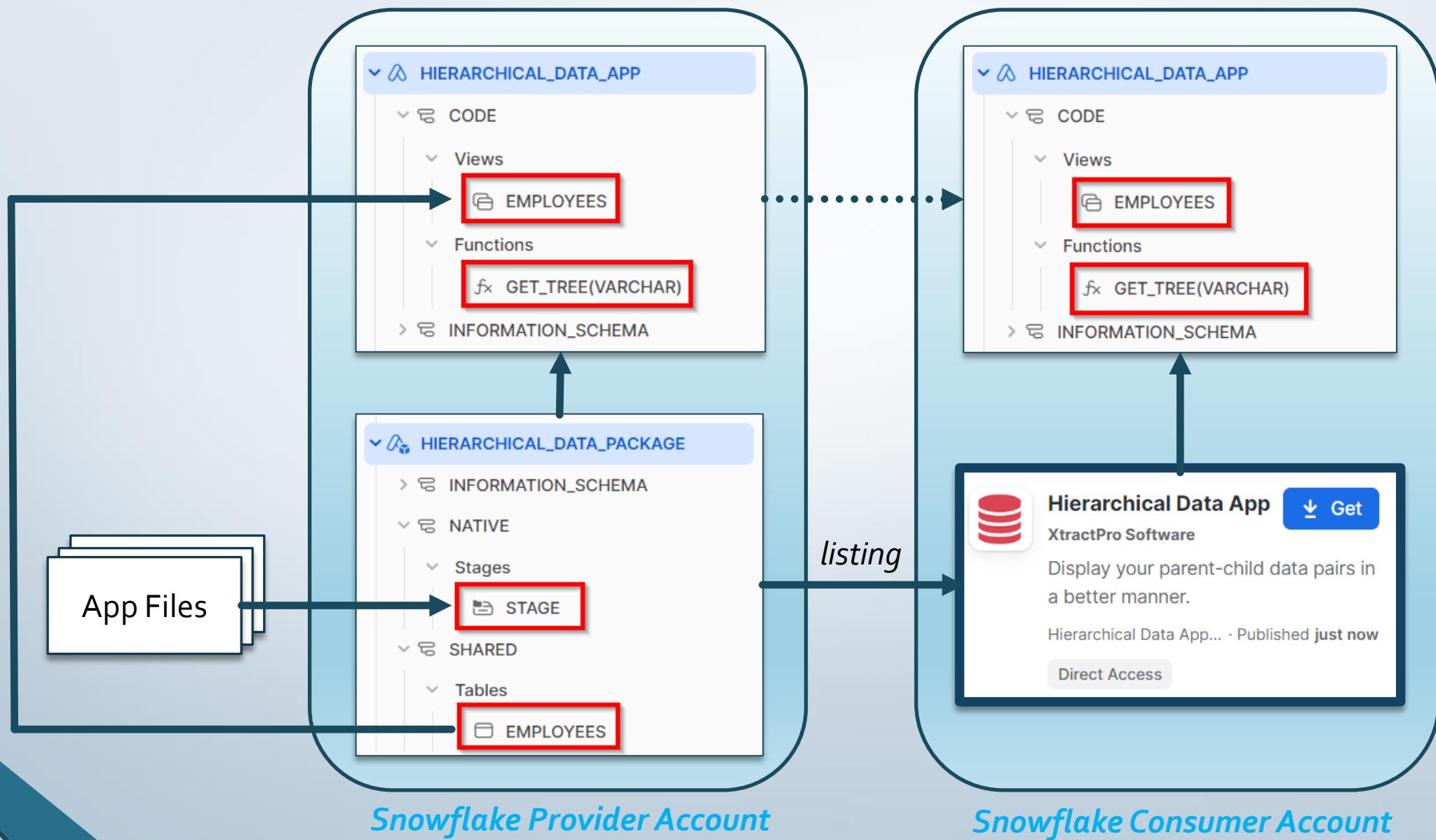
Native App: Private Share (Data Exchange)



Native App: Public Share (Snowflake Marketplace)



Native App: Provider-Consumer





Client Request

Hi there. I'm the Lead Data Analyst....

People wonder how much we consume with Snowflake, and on which queries.

The Usage charts are good, but we need more insights.

Review of Client Request

Hi there. I'm the Lead Data Analyst....

People wonder how much we consume with Snowflake, and on which queries.

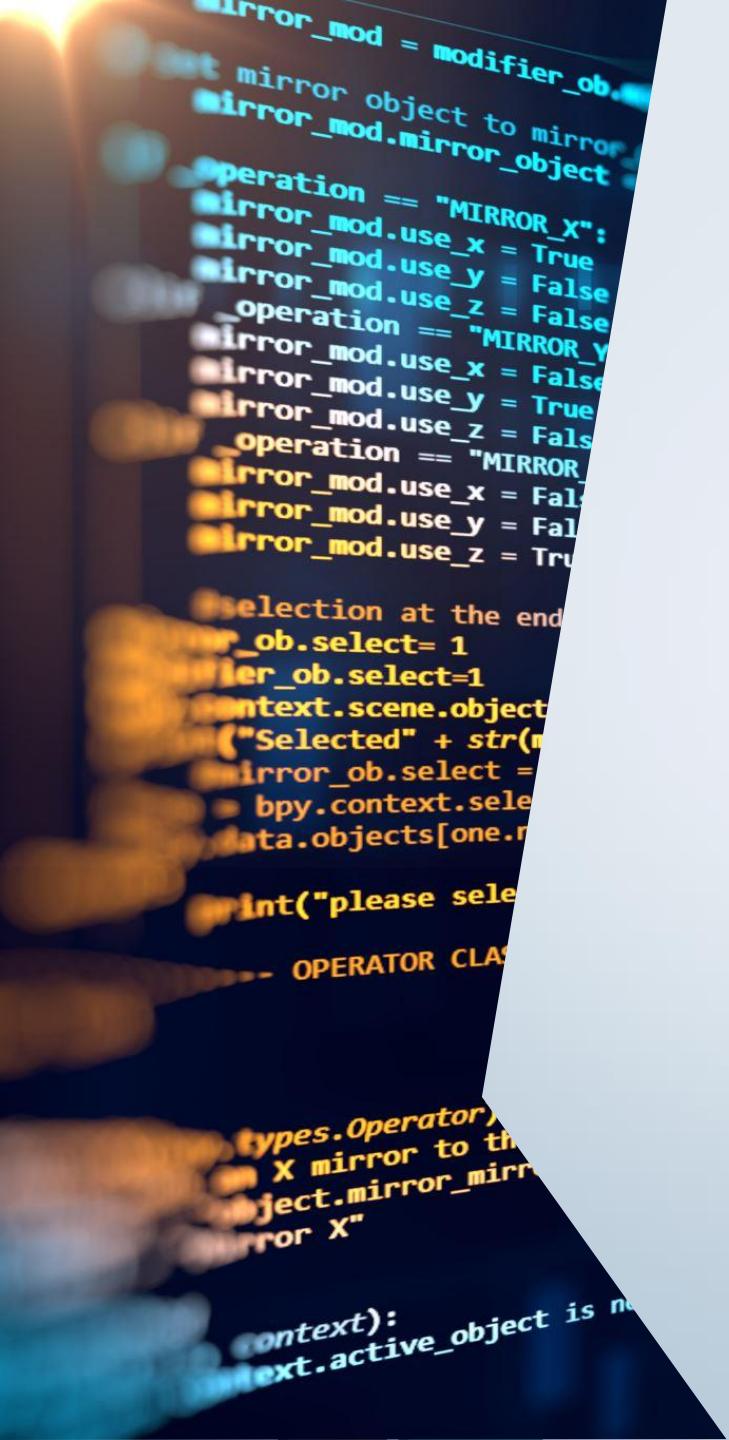
The Usage charts are good, but we need more insights.



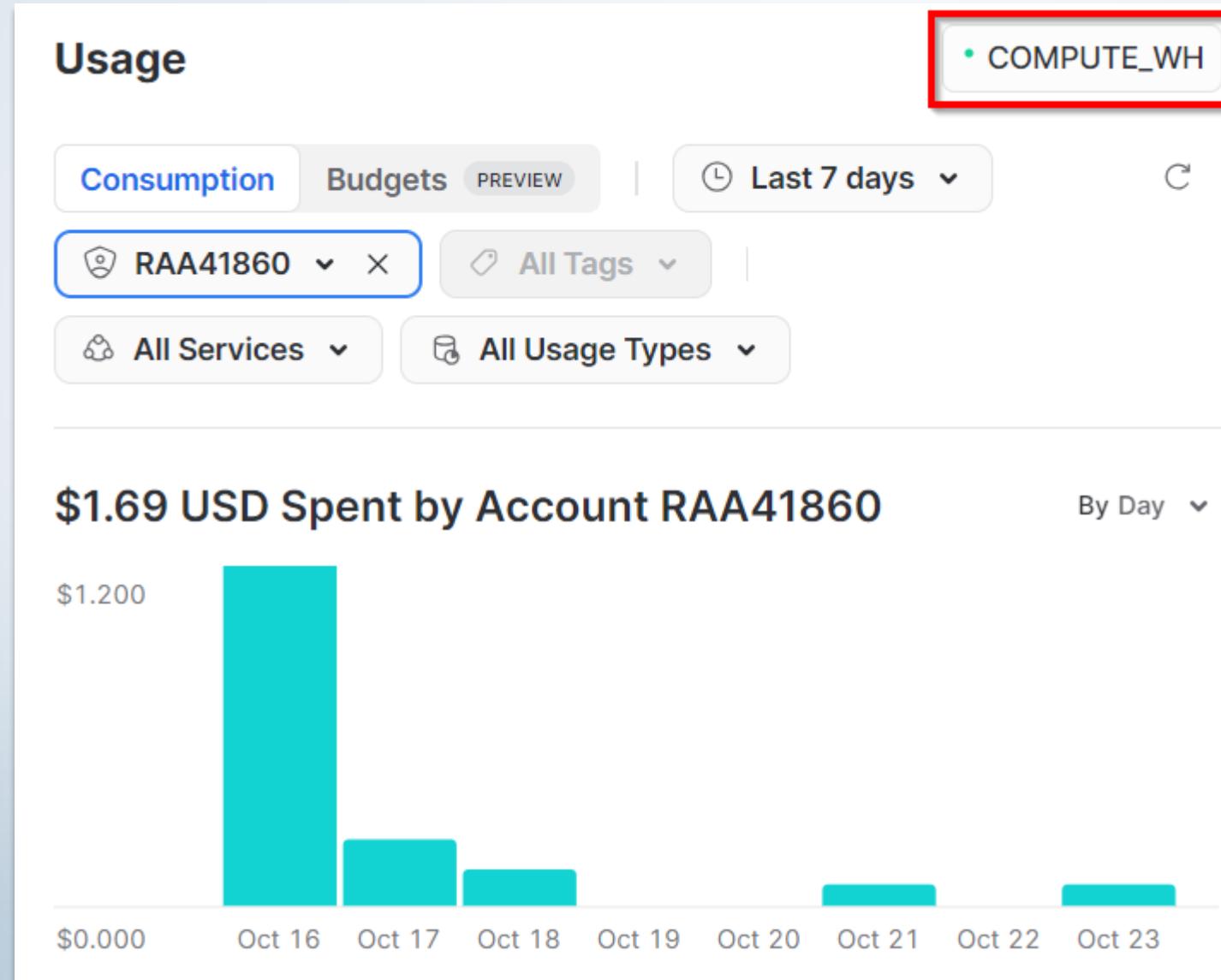
Section Summary



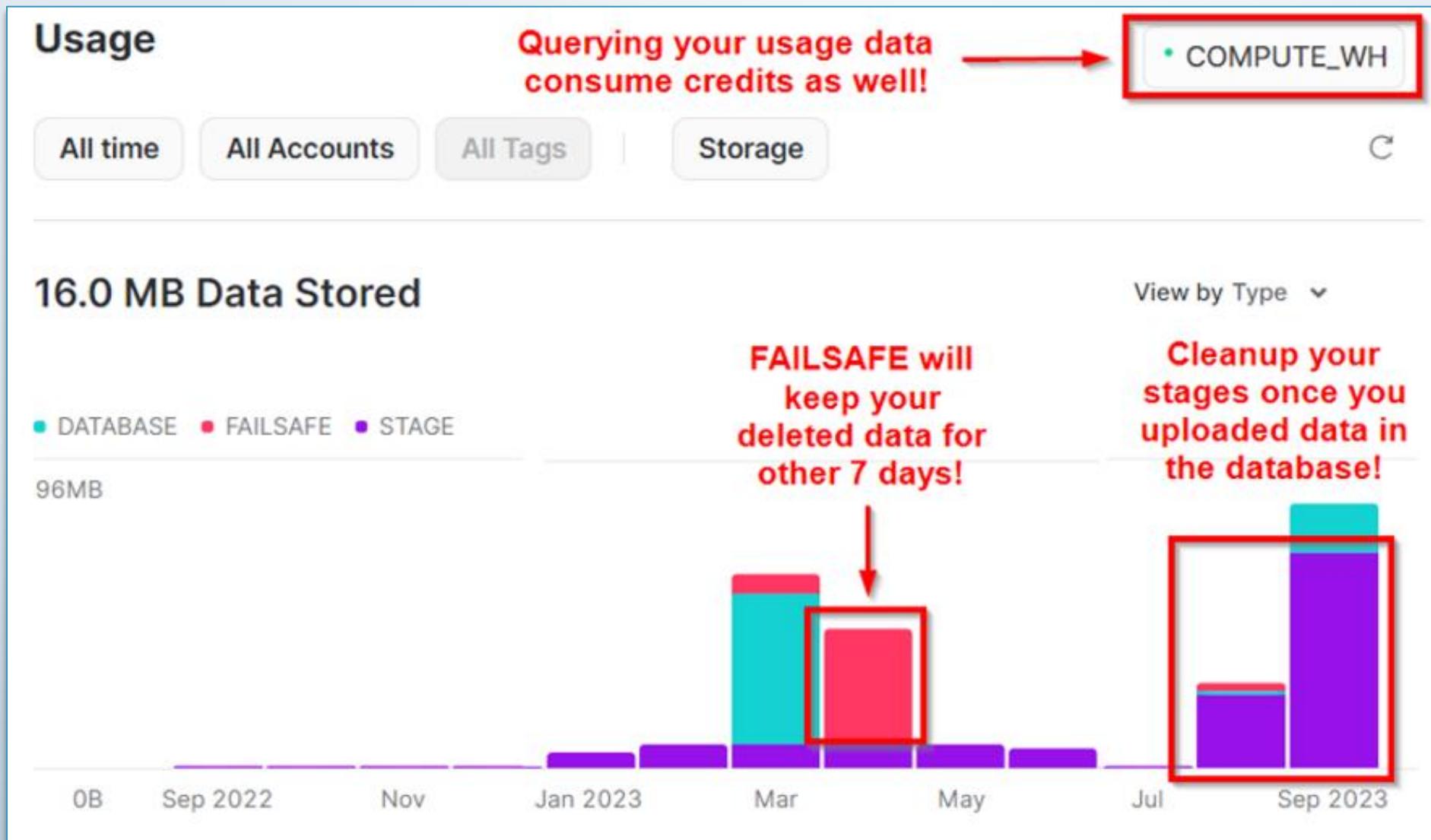
- Admin Usage Charts
- Hidden Cost Traps
- Metering History & Daily History
- Warehouse Metering History
- Query & Load History
- Storage Usage



Admin Usage Charts

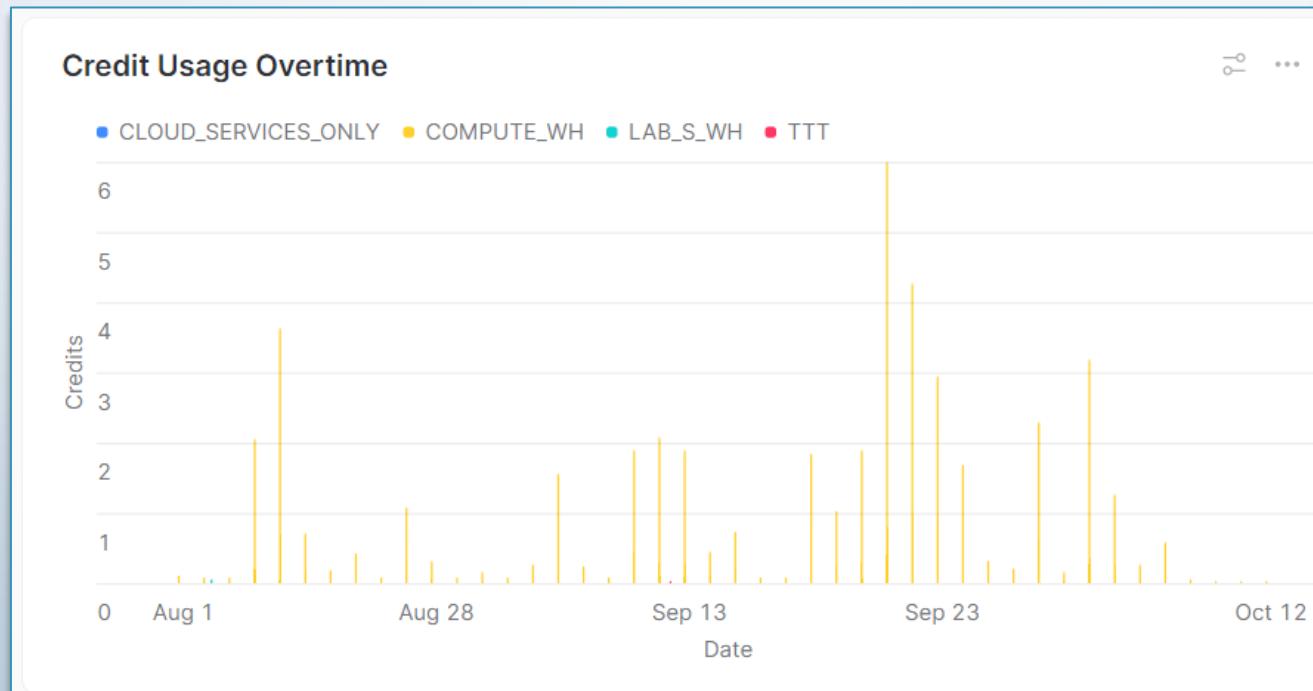


Hidden Cost Traps



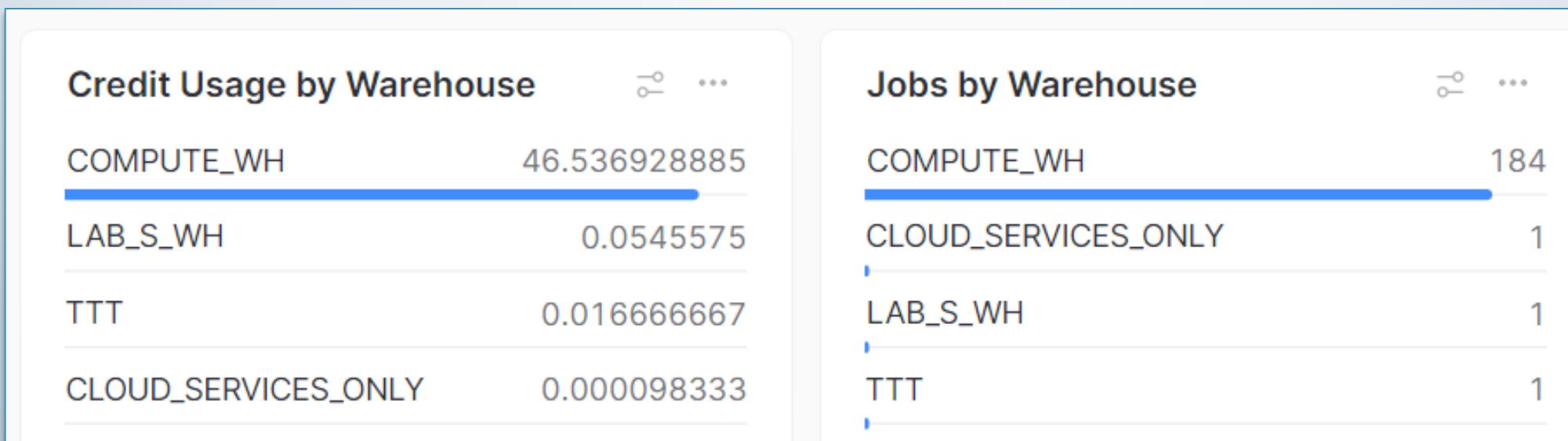
Admin Dashboard: METERING Views

- **METERING_HISTORY**
 - Credits Used [between ...]
- **METERING_DAILY_HISTORY**
 - Credits Billed by Month
 - Credit Breakdown by Day with Cloud Services Adjustment



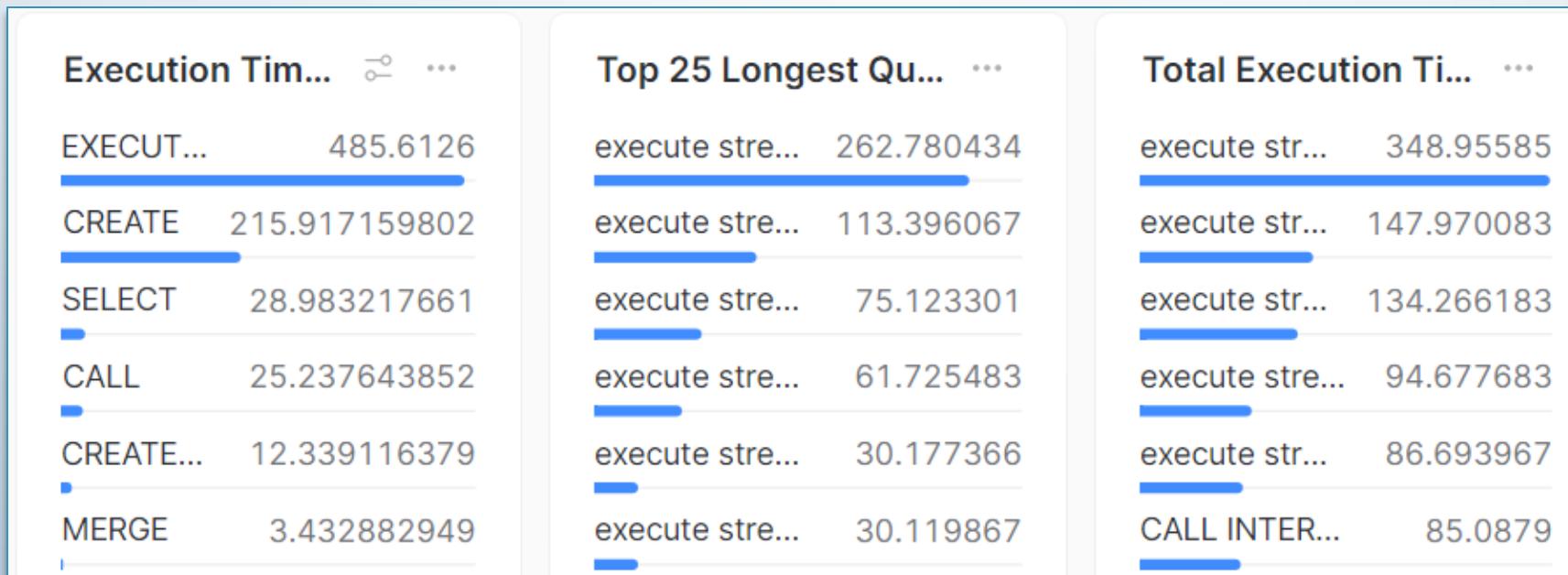
Admin Dashboard: WAREHOUSE_METERING_HISTORY

- Credit Usage by Warehouse [between ...]
- Jobs by Warehouse [between ...]
- Credit Usage by Warehouse over Time [between ...]
- Warehouse Usage Greater than 7 Day Average [between ...]
- Compute and Cloud Services by Warehouse [between ...]



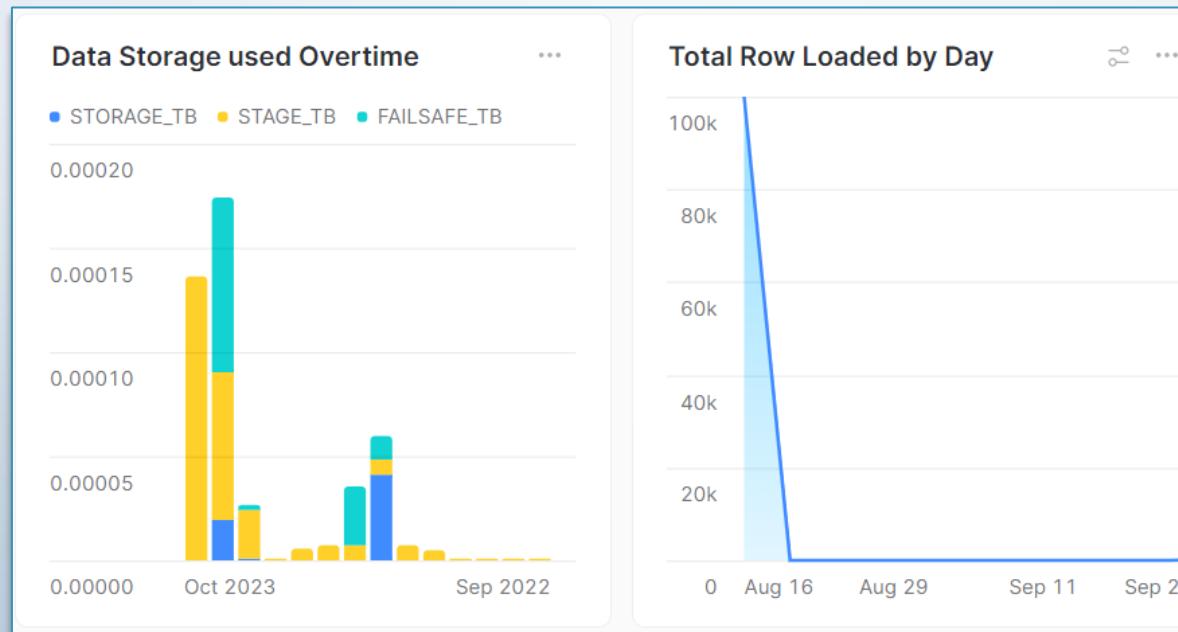
Admin Dashboard: QUERY_HISTORY

- Total Number of Executed Jobs [since ...]
- Execution Time by Query Type
- Top 25 Longest Queries
- Total Execution Time by Repeated Queries
- Average Query Execution Time (By User)
- Cloud Service Credit Utilization by Query Type (Top 10)



Admin Dashboard: Miscellaneous

- **LOAD_HISTORY**
 - Total Row Loaded by Day [between ...]
- **LOGIN_HISTORY**
 - Logins by User or Client [between ...]
- **STORAGE_USAGE**
 - Average Current Storage (for Data, Stages and Fail-Safe) [since ...]
 - Data Storage used Overtime





Client Request

A few of our employees would need a crush course in intermediate SQL.

In particular, could you refresh our memory on:

- Conversion of inline subqueries to CTEs
- GROUP BY queries with grouping sets
- Pivot and unpivot queries

We also need to look at data back in time at any moment. What choices do we have?

Review of Client Request

A few of our employees would need a crush course in intermediate SQL.

In particular, could you refresh our memory on:

- Conversion of inline subqueries to CTEs
- GROUP BY queries with grouping sets
- Pivot and unpivot queries

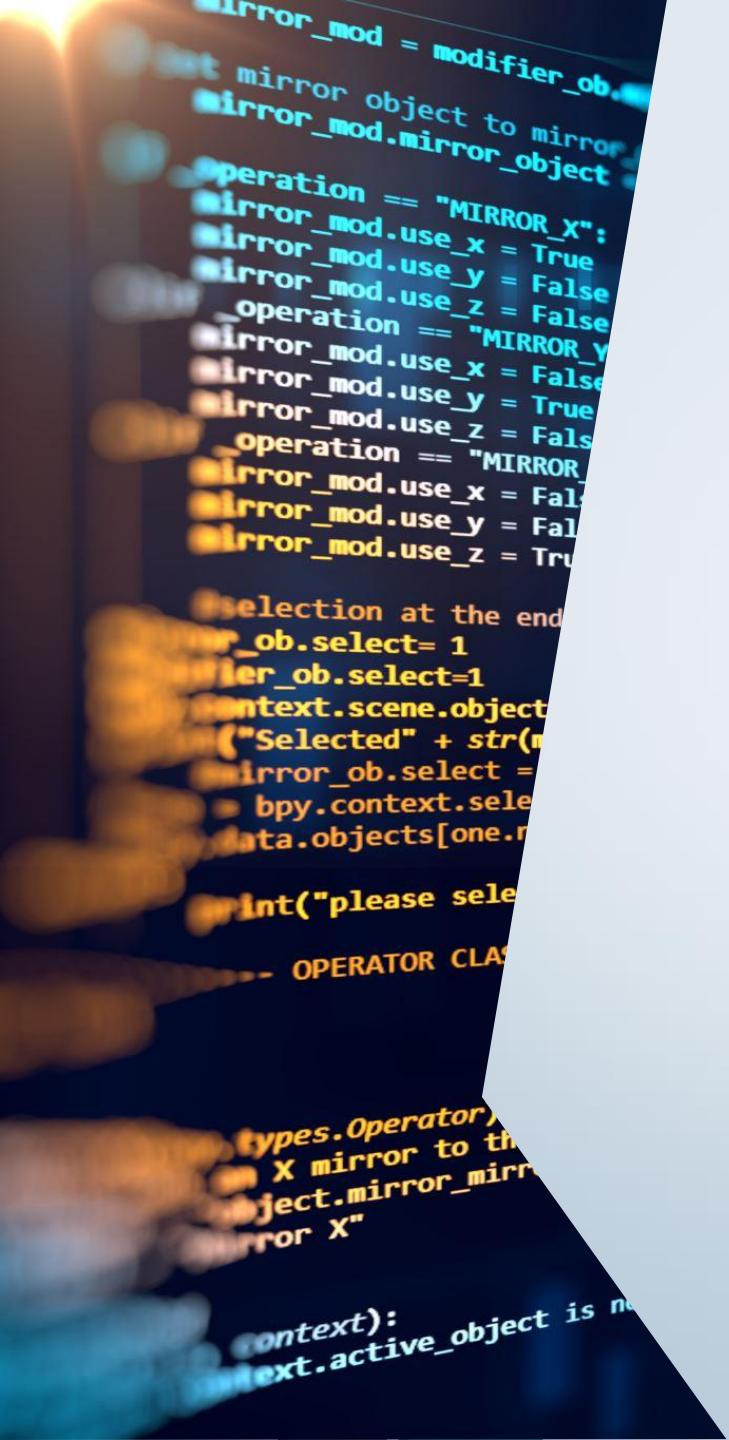
We also need to look at data back in time at any moment. What choices do we have?



Section Summary



- Data Analytics
- SELECT Statement
- Subqueries vs CTEs
- Group Queries
- Pivot/Unpivot Queries
- Time Travel and Fail-safe



SELECT Statement

- **SELECT ...** ← projection
- **DISTINCT ...** ← dedup
- **FROM ...** ← sources (joins)
- **PIVOT | UNPIVOT ...** ← dicing
- **GROUP BY [CUBE/ROLLUP/] ...** ← grouping
- **WHERE | HAVING | QUALIFY ...** ← filters
- **ORDER BY ...** ← sorting
- **TOP | LIMIT | OFFSET | FETCH ...** ← slicing

Subqueries vs CTEs

Subqueries

```
select ee.deptno,
       sum(ee.sal) as sum_sal,
       (select max(sal)
        from emp
        where deptno = ee.deptno) as max_sal
  from emp ee
 where ee.empno in
       (select empno
        from emp e
        join dept d on e.deptno = d.deptno
        where d.dname <> 'RESEARCH')
 group by ee.deptno
 order by ee.deptno;
```

CTEs

```
with q1 as
      (select empno
       from emp e
       join dept d on e.deptno = d.deptno
       where d.dname <> 'RESEARCH'),
q2 as
      (select deptno, max(sal) max_sal
       from emp
       group by deptno)
select ee.deptno,
       sum(ee.sal) as sum_sal,
       max(q2.max_sal) as max_sal
  from emp ee
   join q2 on q2.deptno = ee.deptno
   join q1 on q1.empno = ee.empno
 group by ee.deptno
 order by ee.deptno;
```

GROUP BY with WHERE and HAVING

```
select deptno,
       to_char(year(hiredate)) as year,
       sum(sal) sals
  from emp
 where year > '1980'
 group by deptno, year      -- all
 having sum(sal) > 5000
 order by deptno, year;
```

DEPTNO	YEAR	SALS
10	1981	7,450.5
20	1981	5,975.8
30	1981	9,400

GROUP BY with QUALIFY

```
select deptno,  
       row_number() over (order by deptno) as rn,  
       to_char(year(hiredate)) as year,  
       sum(sal) sals  
  from emp  
 where year > '1980'  
 group by deptno, year  
 having sum(sal) > 5000  
 qualify rn > 1  
 order by deptno, year;
```

DEPTNO	RN	YEAR	...	SALS
20	2	1981		5,975.8
30	3	1981		9,400

GROUP BY with GROUPING SETS

```
select deptno,
       to_char(year(hiredate)) as year,
       grouping(deptno) deptno_g,
       grouping(year) year_g,
       grouping(deptno, year) deptno_year_g,
       sum(sal) sals
  from emp where year > '1980'
 group by grouping sets (deptno, year)
 having sum(sal) > 5000
 order by deptno, year;
```

DEPTNO	YEAR	DEPTNO_G	YEAR_G	DEPTNO_YEAR_G	SALS
10	null	0	1	1	8,750.5
20	null	0	1	1	10,075.8
30	null	0	1	1	9,400
null	1981	1	0	2	22,826.3

GROUP BY with ROLLUP

```
select deptno,
       to_char(year(hiredate)) as year,
       grouping(deptno) deptno_g,
       grouping(year) year_g,
       grouping(deptno, year) deptno_year_g,
       sum(sal) sals
  from emp where year > '1980'
 group by rollup (deptno, year)
 having sum(sal) > 5000
 order by deptno, year;
```

DEPTNO	YEAR	DEPTNO_G	YEAR_G	DEPTNO_YEAR_G	...	SALS
10	1981	0	0	0	7,450.5	
10	null	0	1	1	8,750.5	
20	1981	0	0	0	5,975.8	
20	null	0	1	1	10,075.8	
30	1981	0	0	0	9,400	
30	null	0	1	1	9,400	
null	null	1	1	3	28,226.3	

GROUP BY with CUBE

```
select deptno,
       to_char(year(hiredate)) as year,
       grouping(deptno) deptno_g,
       grouping(year) year_g,
       grouping(deptno, year) deptno_year_g,
       sum(sal) sals
  from emp where year > '1980'
 group by cube (deptno, year)
 having sum(sal) > 5000
 order by deptno, year;
```

DEPTNO	YEAR	DEPTNO_G	YEAR_G	DEPTNO_YEAR_G	SALS
10	1981	0	0	0	7,450.5
10	null	0	1	1	8,750.5
20	1981	0	0	0	5,975.8
20	null	0	1	1	10,075.8
30	1981	0	0	0	9,400
30	null	0	1	1	9,400
null	1981	1	0	2	22,826.3
null	null	1	1	3	28,226.3

PIVOT Query

GROUP BY Query

```
select dname,  
       to_char(year(hiredate)) as year,  
       sum(sal) as sals  
  from emp  
 join dept on emp.deptno = dept.deptno  
 where year >= '1982'  
 group by dname, year  
 order by dname, year;
```

DNAME	YEAR	SALS
ACCOUNTING	1982	1,300
RESEARCH	1982	3,000
RESEARCH	1983	1,100

PIVOT Query

```
with q as  
(select dname,  
       year(hiredate) as year,  
       sum(sal) as sals  
  from emp  
 join dept on emp.deptno = dept.deptno  
 where year >= 1982  
 group by dname, year  
 order by dname, year)  
  
select * from q  
pivot (sum(sals)  
      for year in (1982, 1983)) as p;
```

DNAME	1982	...	1983
ACCOUNTING	1,300		null
RESEARCH	3,000		1,100

UNPIVOT Query

PIVOT Query

```
with q as
  (select dname,
    year(hiredate) as year,
    sum(sal) as sals
  from emp
  join dept on emp.deptno = dept.deptno
  where year >= 1982
  group by dname, year
  order by dname, year)

select * from q
pivot (sum(sals)
      for year in (1982, 1983)) as p;
```

DNAME	1982	...	1983
ACCOUNTING	1,300		null
RESEARCH	3,000		1,100

UNPIVOT Query (*→ back to GROUP BY*)

```
with q as
  (select ...),

p as
  (select * from q
   pivot (sum(sals)
         for year in (1982, 1983)) as p)

select * from p
unpivot (sals
         for year in ("1982", "1983"));
```

DNAME	YEAR	SALS
ACCOUNTING	1982	1,300
RESEARCH	1982	3,000
RESEARCH	1983	1,100

Time Travel and Fail-safe

- *Time Travel*
 - = for DATABASE|SCHEMA|TABLE
 - **DATA_RETENTION_TIME_IN_DAYS** ← in CREATE ... / ALTER ... SET ...
 - set to zero to disable
 - 1 for transient/temporary tables, or permanent tables in Standard Edition
 - 1..90 days for permanent tables in Enterprise Edition
- *Fail-safe*
 - = additional days to restore tables (no SQL, call Snowflake support!)
 - 7 days for permanent tables ← regardless of the edition + cannot disable!
 - 0 days for transient tables

Time Travel

- *Looking Back in Time*
 - `SELECT ... FROM ... AT(TIMESTAMP => <timestamp>) ...`
 - `SELECT ... FROM ... AT(OFFSET => <time_diff>) ...`
 - `SELECT ... FROM ... AT(STREAM => '<name>') ...`
 - `SELECT ... FROM ... AT|BEFORE(STatement => <id>) ...`
- *In Zero-Copy Cloning*
 - `CREATE DATABASE | SCHEMA | TABLE <t> CLONE <s> AT|BEFORE(...)`
- *Restoring Dropped Objects*
 - `DROP DATABASE | SCHEMA | TABLE ...`
 - `SHOW DATABASES | SCHEMAS | TABLES HISTORY [...]` ← dropped
 - `UNDROP DATABASE | SCHEMA | TABLE ...` ← restore dropped obj



Client Request

Our employees know basic SQL, but advanced analytics is not their area of expertise.

Could you quickly walk us through the main categories of window functions and a few statistical SQL extensions, with practical demos on our small live database?

Review of Client Request

Our employees know basic SQL, but advanced analytics is not their area of expertise.

Could you quickly walk us through the main categories of window functions and a few statistical SQL extensions, with practical demos on our small live database?



Section Summary



- Window Functions
- Ranking Functions
- Offset Functions
- Statistical Functions
- Regression Functions



Window Functions: OVER Clause

- PARTITION BY
- ORDER BY
- ROWS/RANGE ← window framing (# rows / values)

Window Frame

```
select ename, hiredate, sal,  
       round(avg(sal) over (order by hiredate  
                           rows between 1 preceding and 1 following), 2) as avg  
from emp  
order by hiredate;
```

ENAME	HIREDATE	...	SAL	AVG
SMITH	1980-12-17		800	1,200
ALLEN	1981-02-20		1,600	1,216.67
WARD	1981-02-22		1,250	1,941.93
JONES	1981-04-02		2,975.8	2,358.6
BLAKE	1981-05-01		2,850	2,758.77
CLARK	1981-06-09		2,450.5	2,266.83
TURNER	1981-09-08		1,500	1,733.5
MARTIN	1981-09-28		1,250	2,583.33
KING	1981-11-17		5,000	3,083.33

Rank Functions

- **ROW_NUMBER()** $\leftarrow 1, 2, 3, 4, 5, 6, 7 \dots$
- **RANK()** $\leftarrow 1, 1, 3, 4, 4, 4, 7 \dots$
- **DENSE_RANK()** $\leftarrow 1, 1, 2, 3, 3, 3, 4, \dots$
- **PERCENT_RANK()** $\leftarrow 0\%, 0\%, 23\%, 48\%, \dots$
- **NTILE(n)** $\leftarrow 1, 1, 1, 2, 2, 2, 3, 3 \dots$
- **CUME_DIST()**

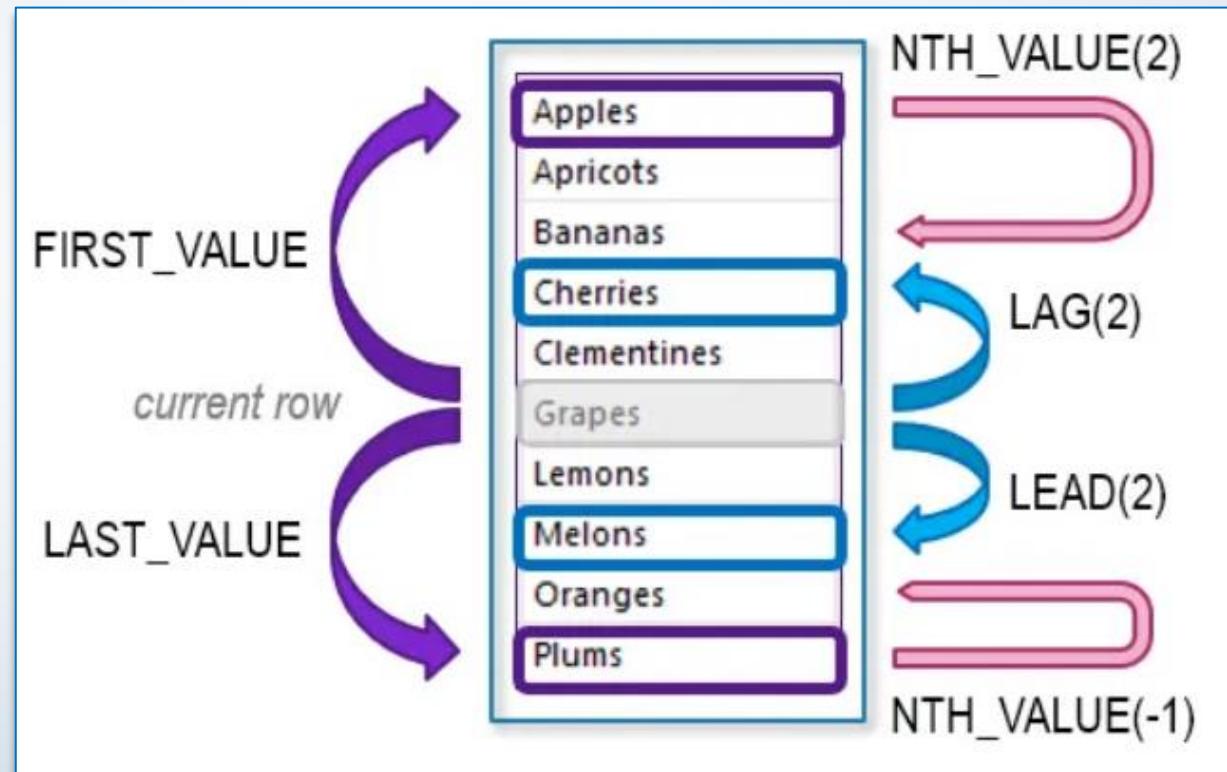
Rank Functions

```
select deptno, ename,
       row_number() over (order by deptno) row_number,
       rank() over (order by deptno) rank,
       dense_rank() over (order by deptno) dense_rank,
       round(percent_rank() over (order by deptno) * 100) || '%' percent_rank
from emp
order by deptno;
```

DEPTNO	ENAME	ROW_NUMBER	RANK	...	DENSE_RANK	PERCENT_RANK
10	KING	1	1		1	0%
10	CLARK	2	1		1	0%
10	MILLER	3	1		1	0%
20	JONES	4	4		2	23%
20	SCOTT	5	4		2	23%
20	ADAMS	6	4		2	23%
20	FORD	7	4		2	23%
20	SMITH	8	4		2	23%
30	BLAKE	9	9		3	62%
30	MARTIN	10	9		3	62%

Offset Functions

- **LEAD(expr, offset=1)**
- **LAG(expr, offset=1)**
- **FIRST_VALUE(expr)**
- **LAST_VALUE(expr)**
- **NTH_VALUE(expr,offset)**
- **RATIO_TO_REPORT(expr)**



Offset Functions

```
select ename,  
       lead(sal, 1) over (order by ename) lead,  
       sal,  
       lag(sal, 1) over (order by ename) lag,  
       first_value(sal) over (order by ename) first,  
       last_value(sal) over (order by ename) last,  
       nth_value(sal, 1) over (order by ename) nth  
  from emp  
order by ename;
```

ENAME	LEAD	SAL	LAG	FIRST	LAST	NTH
ADAMS	1,600	1,100	null	1,100	1,250	1,100
ALLEN	2,850	1,600	1,100	1,100	1,250	1,100
BLAKE	2,450.5	2,850	1,600	1,100	1,250	1,100
CLARK	3,000	2,450.5	2,850	1,100	1,250	1,100
FORD	950	3,000	2,450.5	1,100	1,250	1,100
JAMES	2,975.8	950	3,000	1,100	1,250	1,100

Statistical Functions

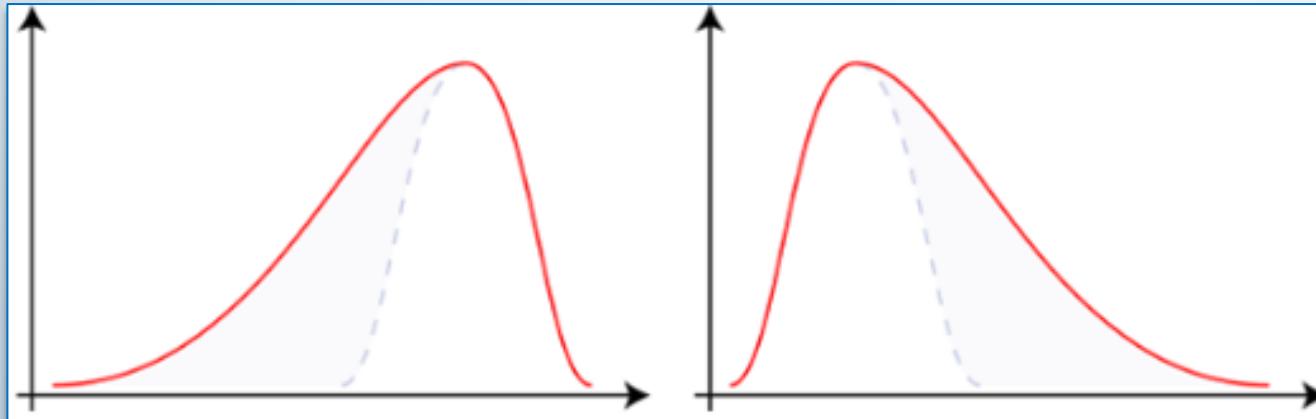
- **VAR_POP/SAMP**
- **VARIANCE**
- **STDDEV_POP/SAMP**
- **STDDEV**
- **COVAR_POP/SAMP**
- **CORR**

- **SKEW(expr)**
- **KURTOSIS(expr)**

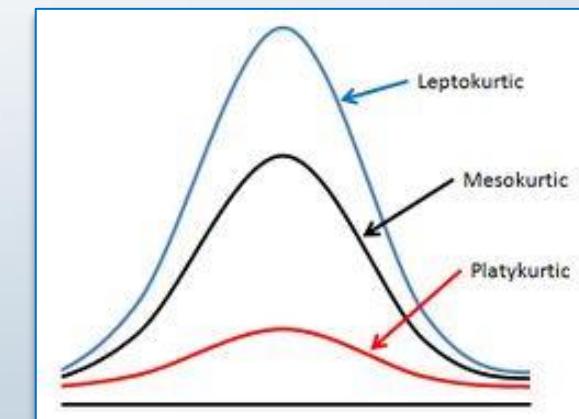
Skew & Kurtosis

```
select count(*), avg(sal), median(sal), skew(sal), kurtosis(sal) from emp;
```

COUNT(*)	AVG(SAL)	MEDIAN(SAL)	...	SKEW(SAL)	KURTOSIS(SAL)
14	2,073.307142857	1,550		1.174366425	1.318086599



Skew (negative/positive)

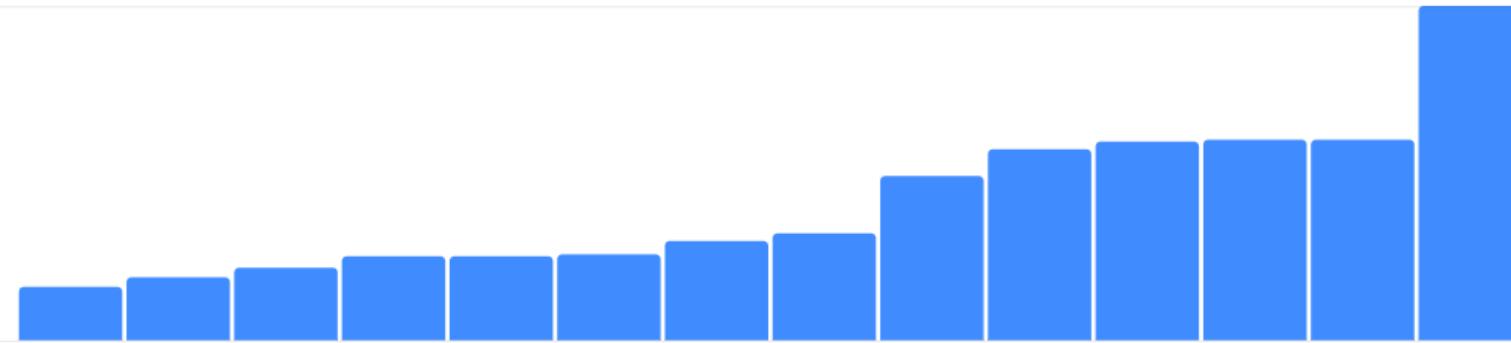


Kurtosis

Distributions

```
select row_number() over(order by sal) rn, sal  
from emp  
order by sal;
```

5,000



```
select width_bucket(SAL, 800, 5000, 10) as sals  
from emp  
order by sal;
```

20

0

11

6

5

4

2

1



Regression Functions

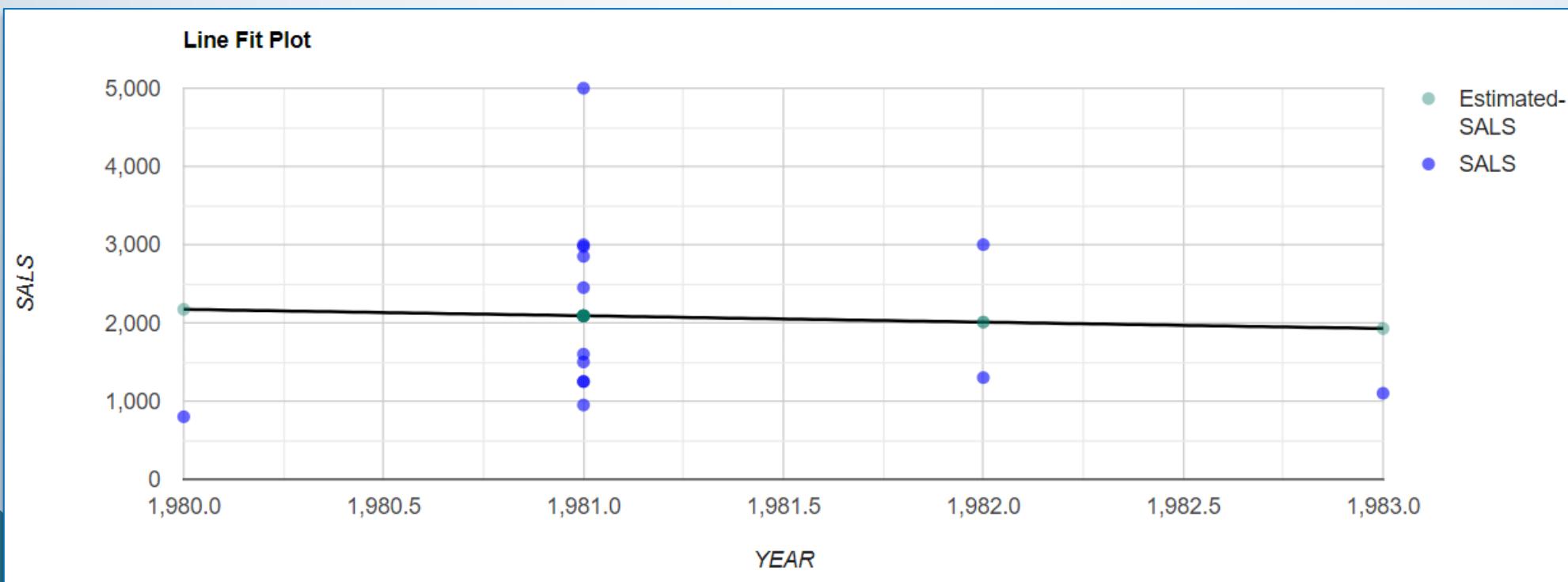
- **REGR_SLOPE**
- **REGR_INTERCEPT**
- **REGR_R2**

- **REGR_COUNT**
- **REGR_SXX/SYY/SXY**
- **REGR_AVGX/AVGY**

Linear Regression ($y = x * \text{SLOPE} + \text{INTERCEPT}$)

```
select REGR_SLOPE(sals, year), REGR_INTERCEPT(sals, year), REGR_R2(sals, year)  
from (select year(hiredate) as year, sal as sals from emp order by year);
```

REGR_SLOPE(SALS, YEAR)	...	REGR_INTERCEPT(SALS, YEAR)	REGR_R2(SALS, YEAR)
-81.785393259		164,107.696630673	0.002338955478





Client Request

In practice, some queries will always be slower.

Could you give us some pointers where to look at to eventually improve the query performance?

How can we estimate if the storage is appropriate for a query? If not, will clustering truly improve the performance on some specific queries?

Review of Client Request

In practice, some queries will always be slower.

Could you give us some pointers where to look at to eventually improve the query performance?

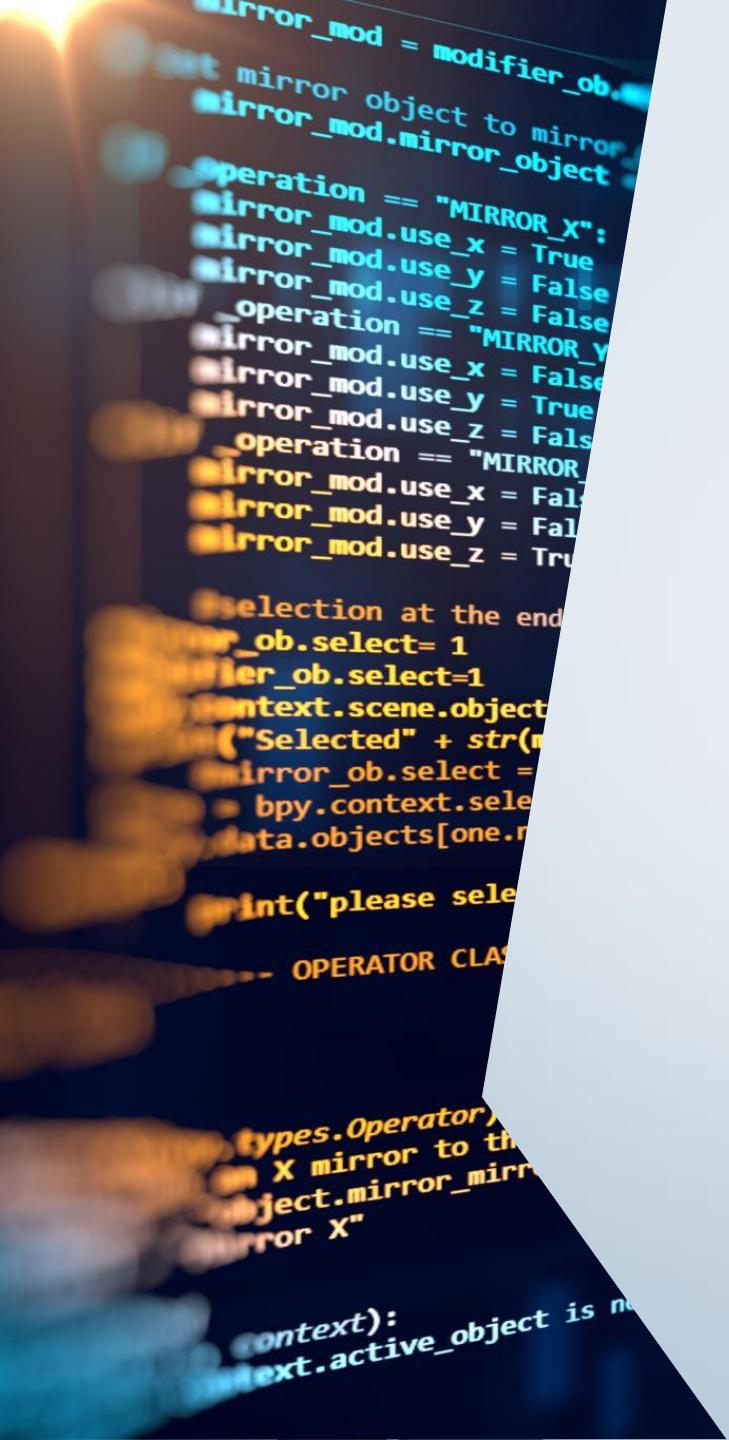
How can we estimate if the storage is appropriate for a query? If not, will clustering truly improve the performance on some specific queries?



Section Summary



- Query Performance Optimization
- Query History & Load History
- Query Result Caching
- Query Execution Plan (EXPLAIN)
- Query Profile
- Clustering Keys
- Enhanced Query Profile & Analysis



Query Performance Optimization Methods

- **Check Query History and Load History** ← in Web UI / Admin Dashboard
 - Check Execution Times ← use Query Hashes
 - Top Longest Queries Chart
 - Long Running Repeated Queries Chart
- **Check Query Profile & Query Plan (EXECUTE Statement)**
 - Avoid Spilling from RAM to Local (SSD) or Remote (S3) Storage
 - Look for “Exploding” Joins
 - Understand Caching + Use Materialize Views
 - Need Clustering? Understand Micro-Partitioning
 - Inefficient Pruning ← check Partition Scanned/Total in TableScan
- **Optimize Warehouse Performance**
 - Increase Warehouse Size
 - Query Acceleration
 - Check Queued Queries

Query History

- ***QUERY_HISTORY***
 - **INFORMATION_SCHEMA** - table function, for the past 7 days, limited, by session/user/wh
 - **ACCOUNT_USAGE** - view, for the past year, with latency
- ***query types***
 - **most frequently executed** queries → count(*), check task schedules
 - **most time-consuming queries**, overall (in total) → is it ok to exec so frequently?
 - **slowest queries**, with longest execution time (in average) → may need optimization
 - **heaviest queries**, with most scanned data (in average)
- ***information***
 - which KPI am I looking at? → order by + row_number() sort
 - is my query in this list? on which place? → find by query text + use row_number()
- ***filters***
 - only successfully executed, by period (last mo, year) + limit to max 10,000
 - by query type (SELECT, CALL...), warehouse size, query tag... → optional

Enhanced Query Analysis

Query **Analysis** Graph Metadata Explain

The query was found by ID [in](#) the ACCOUNT_USAGE schema.

The query has been run successfully [in 542 ms](#): it compiled [in 379 ms](#) + it ex
The query run between [2023-11-04 07:10:29.331000-07:00](#) and [2023-11-04 07:10:](#)

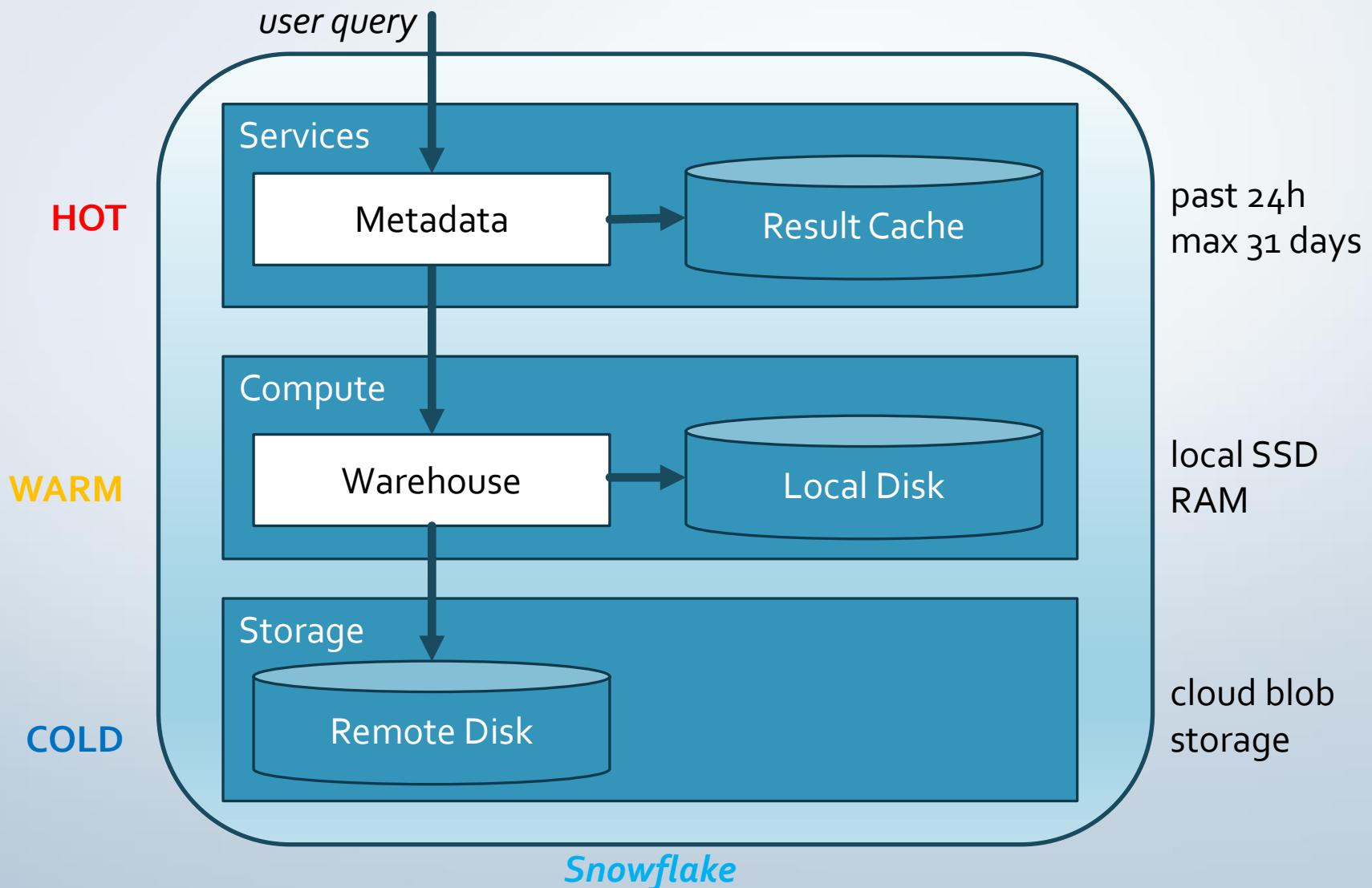
It has been executed [1](#) times in the last month, for a total of [0.542000](#) secc
It [is NOT](#) among the top [10](#) longest queries executed [in](#) the last month.
It [is NOT](#) among the top [100](#) longest queries executed [in](#) the last month.
It [is NOT](#) among the top [10](#) queries with most data scanned executed [in](#) the la
It [is NOT](#) among the top [100](#) queries with most data scanned executed [in](#) the l

The query was executed by the CRISTISCU user, using the ACCOUNTADMIN role.
The query was executed within the EMPLOYEES.PUBLIC database and schema conte
The query used the X-Small COMPUTE_WH standard warehouse, with [1](#) nodes, [100%](#)

The query produced [2](#) rows.

Nothing spilled to local storage, which [is](#) good.
This often means that the warehouse node(s) had enough memory to process it

Query Result Caching



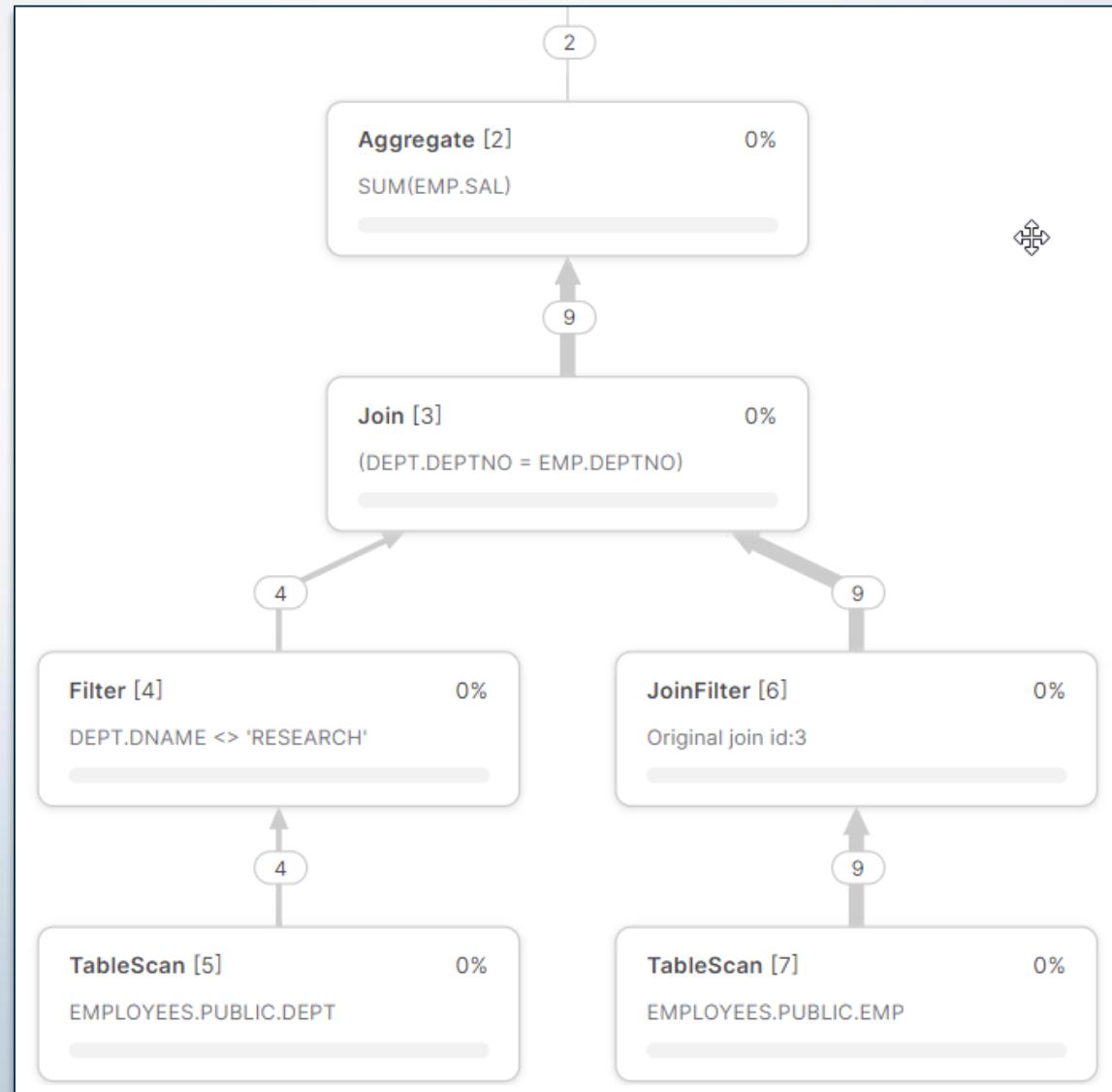
Query Result Caching

- **HOT**
 - result cache on
 - cache hit → `SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()))`
- **WARM**
 - result cache off or no cache hit
 - warehouse up and result still in the warehouse (in RAM or local disks)
 - get query result from warehouse cache (`BYTES_SCANNED > 0`)
- **COLD**
 - no warehouse cache (warehouse suspended, not up)
 - result cache off (`ALTER SESSION SET USE_CACHED_RESULT = false`) or no cache hit
 - must run query

Query Profile

```
select dname, sum(sal)
  from employees.public.emp e
    join employees.public.dept d
      on e.deptno = d.deptno
 where dname <> 'RESEARCH'
 group by dname
 order by dname;
```

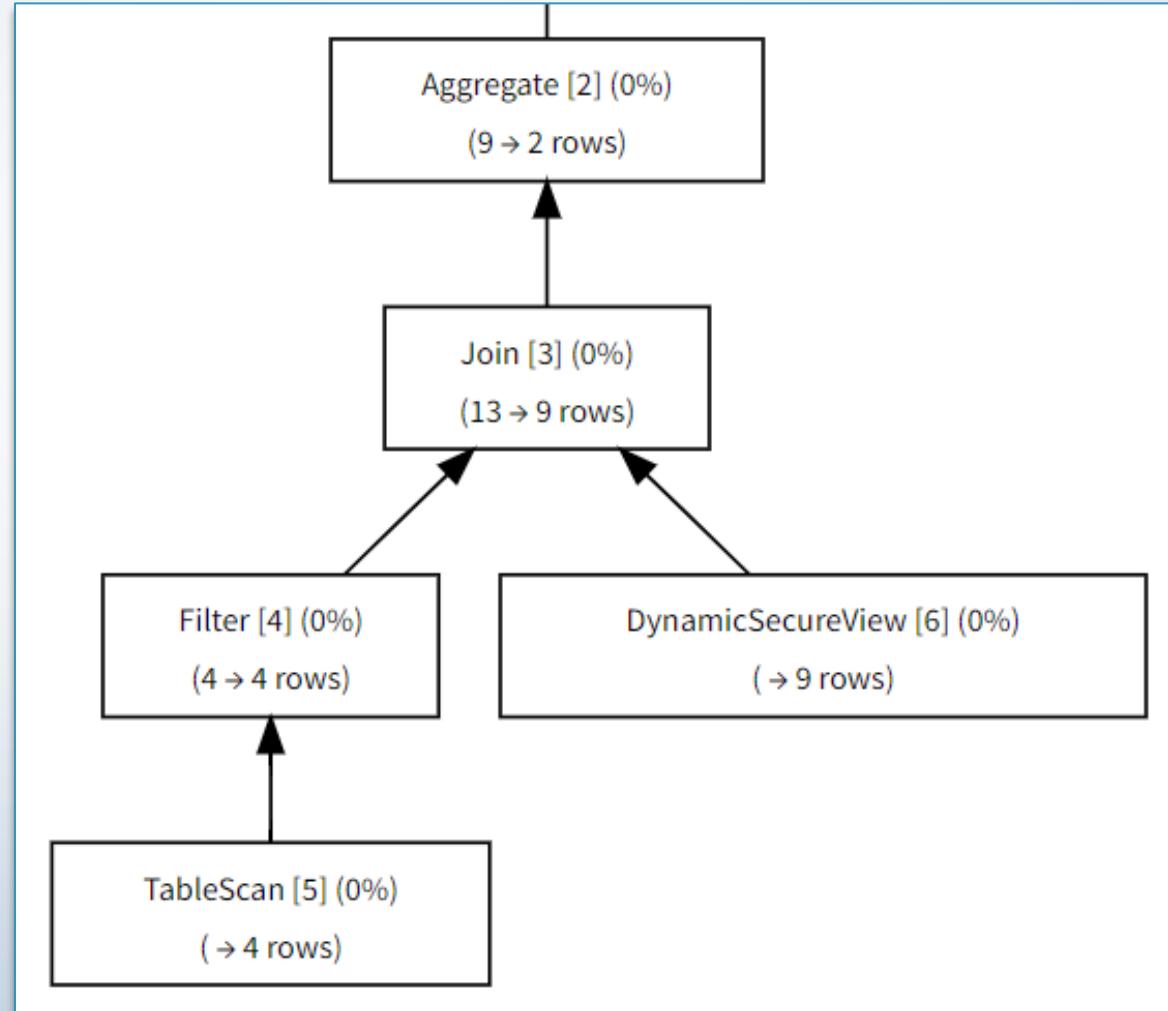
DNAME	...	SUM(SAL)
ACCOUNTING		8,750.5
SALES		9,400



Enhanced Query Profile

```
select dname, sum(sal)
  from employees.public.emp e
    join employees.public.dept d
      on e.deptno = d.deptno
 where dname <> 'RESEARCH'
 group by dname
 order by dname;
```

DNAME	...	SUM(SAL)
ACCOUNTING		8,750.5
SALES		9,400



Enhanced Query Profile

```
select * from table(get_query_operator_stats('01b01af2-0001-db84-003d-b087000abc02'))
```

	OPERATOR_TYPE	OPERATOR_STATISTICS	EXECUTION_TIME_BREAKDOWN	OPERATOR_ATTRIBUTES
0	Result	{ "input_rows": 2 }	{ "overall_percentage": 0.000000000000000e+00 }	{ "expressions": ["D.DNAME", "SUM(E.SAL)"] }
1	Sort	{ "input_rows": 2, "output_rows": 2 }	{ "overall_percentage": 0.000000000000000e+00 }	{ "sort_keys": ["D.DNAME ASC NULLS LAST"] }
2	Aggregate	{ "input_rows": 9, "output_rows": 2 }	{ "overall_percentage": 0.000000000000000e+00 }	{ "functions": ["SUM(E.SAL)"], "grouping": true }
3	Join	{ "input_rows": 13, "output_rows": 9 }	{ "overall_percentage": 0.000000000000000e+00 }	{ "equality_join_condition": "(D.DEPTNO = E.DEPTNO)" }
4	Filter	{ "input_rows": 4, "output_rows": 4 }	{ "overall_percentage": 0.000000000000000e+00 }	{ "filter_condition": "D.DNAME <> 'RESEARCH'" }
5	TableScan	{ "io": { "bytes_scanned": 1536, "percentage": 0.000000000000000e+00 } }	{ "overall_percentage": 0.000000000000000e+00 }	{ "columns": ["DEPTNO", "DNAME"], "table": "D" }
6	DynamicSecureView	{ "output_rows": 9 }	{ "overall_percentage": 0.000000000000000e+00 }	{}

“Exploding” Joins Problem



Query Execution Plan: EXPLAIN

- EXPLAIN [USING TABULAR] <query>
 - ~EXPLAIN_JSON function (JSON → tabular output)
- EXPLAIN USING JSON <query>
 - ~SYSTEM\$EXPLAIN_PLAN_JSON function (JSON output)
- EXPLAIN USING TEXT <query>
 - ~SYSTEM\$EXPLAIN_JSON_TO_TEXT function (JSON → TEXT output)

Operations:

```
1:0      ->Result D.DNAME, SUM(E.SAL)
1:1          ->Sort D.DNAME ASC NULLS LAST
1:2              ->Aggregate aggExprs: [SUM(E.SAL)], groupKeys: [D.DNAME]
1:3                  ->InnerJoin joinKey: (D.DEPTNO = E.DEPTNO)
1:4                      ->Filter D.DNAME <> 'RESEARCH'
1:5                          ->TableScan EMPLOYEES.PUBLIC.DEPT as D DE
1:6                              ->DynamicSecureView "EMP (+ RowAccessPolicy)"
```

Query Execution Plan

```
explain
  select dname, sum(sal)
    from employees.public.emp e
      join employees.public.dept d
        on e.deptno = d.deptno
   where dname <> 'RESEARCH'
 group by dname
 order by dname;
```

operation	expressions	partitionsTotal	partitionsAssigned	bytesAssigned
GlobalStats		2	2	5120
Result	DEPT.DNAME, SUM(EMP.SAL)			
Sort	DEPT.DNAME ASC NULLS LAST			
Aggregate	aggExprs: [SUM(EMP.SAL)], groupKeys: [DEPT.DNAME]			
InnerJoin	joinKey: (DEPT.DEPTNO = EMP.DEPTNO)			
Filter	DEPT.DNAME <> 'RESEARCH'			
TableScan	EMPLOYEES.PUBLIC.DEPT (DEPTNO, DNAME)	1	1	1536
JoinFilter	joinKey: (DEPT.DEPTNO = EMP.DEPTNO)			
TableScan	EMPLOYEES.PUBLIC.EMP (SAL, DEPTNO)	1	1	3584

Clustering

- *Clustering Keys*
 - `SYSTEM$CLUSTERING_DEPTH(table, (col1, ...))`
 - `SYSTEM$CLUSTERING_INFORMATION(table, (col1, ...))`
- *micro-partitions* = table data storage segments, with min/max values on specific (groups of) column values, that can drastically improve the query search.
- *pruning* = possibility of scanning fewer micro-partitions for a query.

Clustering: With Full Clustering Keys

SQL Code → JSON Result

```
select system$clustering_information(  
    'snowflake_sample_data.tpcds_sf100tcl.store_sales');      -- ~300B rows
```

```
{  
    "cluster_by_keys" : "LINEAR(ss_sold_date_sk, ss_item_sk)",  
    "total_partition_count" : 721507,  
    "total_constant_partition_count" : 9,                      ← so-so (higher is better)  
    "average_overlaps" : 3.4849,                                ← bad (many overlaps)  
    "average_depth" : 2.7497,                                    ← system$clustering_depth  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 3,  
        "00002" : 180604,                                     ← so-so (most on depth 2-3)  
        "00003" : 540900,  
        "00004" : 0,  
        ...  
        "00016" : 0                                         ← good (none so deep)  
    },  
    "clustering_errors" : [ ]  
}
```

Clustering: With Partial Clustering Keys

SQL Code → JSON Result

```
select system$clustering_information(  
    'snowflake_sample_data.tpcds_sf100tcl.store_sales',      -- ~300B rows  
    '(ss_sold_date_sk)');
```

```
{  
    "cluster_by_keys" : "LINEAR(ss_sold_date_sk)",  
    "total_partition_count" : 721507,  
    "total_constant_partition_count" : 719687,           ← good! (high)  
    "average_overlaps" : 0.0132,                          ← good! (low)  
    "average_depth" : 1.0076,                            ← system$clustering_depth  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 719203,                                ← good! (most w/ depth 1)  
        "00002" : 366,  
        "00003" : 1197,  
        "00004" : 624,  
        ...  
        "00032" : 22                                     ← so-so (22 partitions on depth 32)  
    },  
    "clustering_errors" : [ ]  
}
```

Clustering: Worst Case Scenario

SQL Code → JSON Result

```
select system$clustering_information(  
    'snowflake_sample_data.tpcds_sf100tcl.store_sales',      -- ~300B rows  
    '(ss_store_sk)');
```

```
{  
    "cluster_by_keys" : "LINEAR(ss_store_sk)",  
    "total_partition_count" : 721507,  
    "total_constant_partition_count" : 0,                      ← very bad (no constant partition)  
    "average_overlaps" : 721506.0,                            ← very bad (almost all overlap)  
    "average_depth" : 721507.0,                                ← system$clustering_depth  
    "partition_depth_histogram" : {  
        "00000" : 0,  
        "00001" : 0,  
        "00002" : 0,  
        "00003" : 0,  
        "00004" : 0,  
        ...  
        "1048576" : 721507                                  ← very bad (all w/ full scan)  
    },  
    "clustering_errors" : [ ]  
}
```

Programming in Snowflake

Masterclass
2024 Hands-On!