WELCOME TO SNOWFLAKE TRAINING

TRAINER INTRO

Name: Rajesh Hegde

Contact: rajeshh@useready.com

Prior Experience:

Goal: To make sure you walk out with better understanding about Snowflake

Hobby/Passion: Photography/Long Drive

**** Note: These sessions will be recorded for training & quality improvement purpose*

WELCOME TO DAY 1

SESSION 1

- Key Concepts & Architecture
- Snowflake Editions & Regions
- Overview of Key Features
- Overview of the Data Lifecycle
- Continuous Data Protection

KEY CONCEPTS AND ARCHITECTURE

- Snowflake is an analytic data warehouse provided as Software-as-a-Service (SaaS).

- Snowflake provides a data warehouse that is faster, easier to use, and far more flexible than traditional data warehouse offerings.

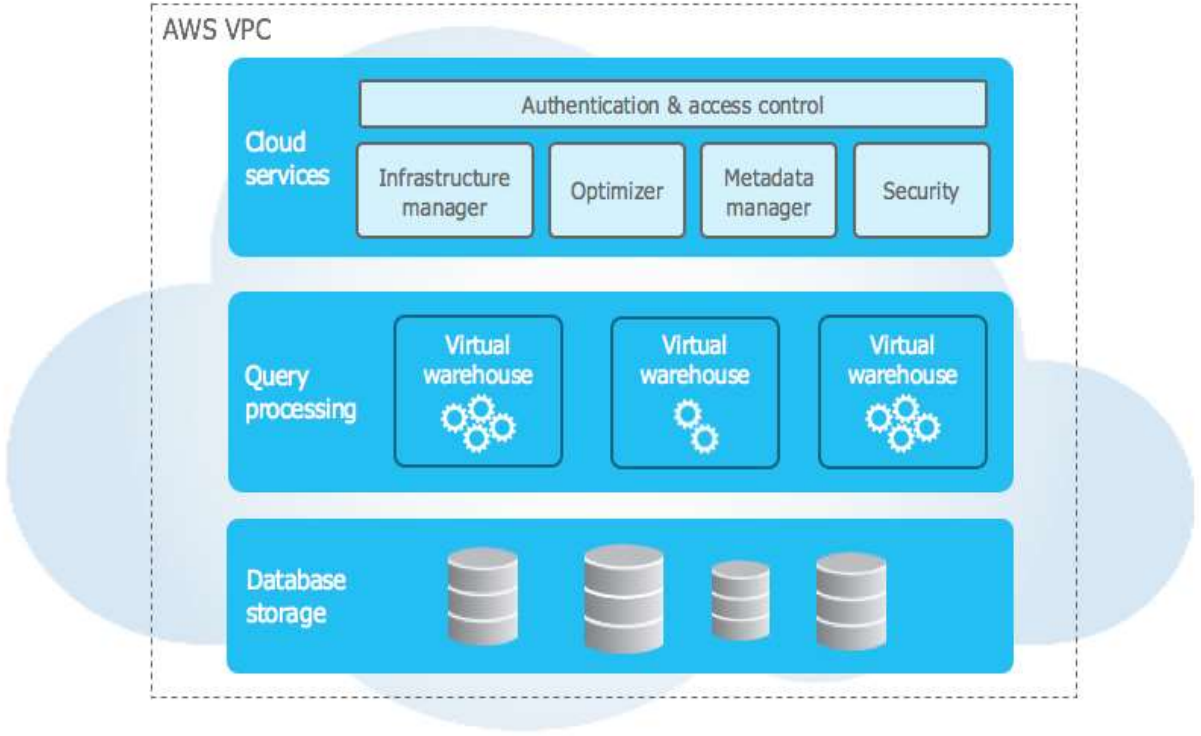
In this Topic:
Data Warehouse as a Cloud Service
Snowflake Architecture
-Database Storage
-Query Processing
-Cloud Services
Connecting to Snowflake

DATA WAREHOUSE AS A CLOUD SERVICE

- Snowflake runs completely on cloud infrastructure. All components of Snowflake's service (other than an optional command line client), run in a public cloud infrastructure.
- At this time, Snowflake runs exclusively in the Amazon Web Services (AWS) cloud infrastructure.
- Snowflake uses virtual compute instances provided by AWS EC2 (Elastic Compute Cloud) for its compute needs and AWS S3 (Simple Storage Service) storage for persistent storage of data.

amazon
web services

SNOWFLAKE ARCHITECTURE

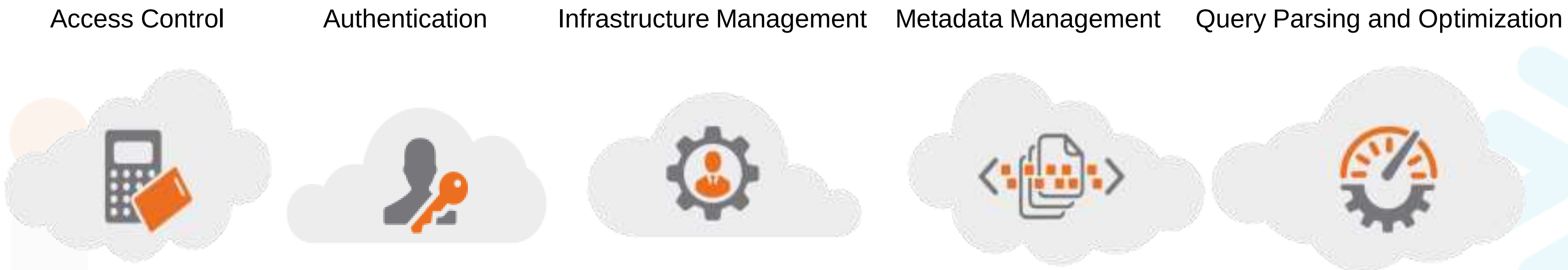


SNOWFLAKE ARCHITECTURE

- Hybrid of traditional shared-disk database architectures and shared-nothing database architectures
- Central data repository for persisted data that is accessible from all compute nodes in the data warehouse
- Processes queries using MPP (massively parallel processing) compute clusters where each node in the cluster stores a portion of the entire data set locally
- Snowflake’s unique architecture consists of three key layers.

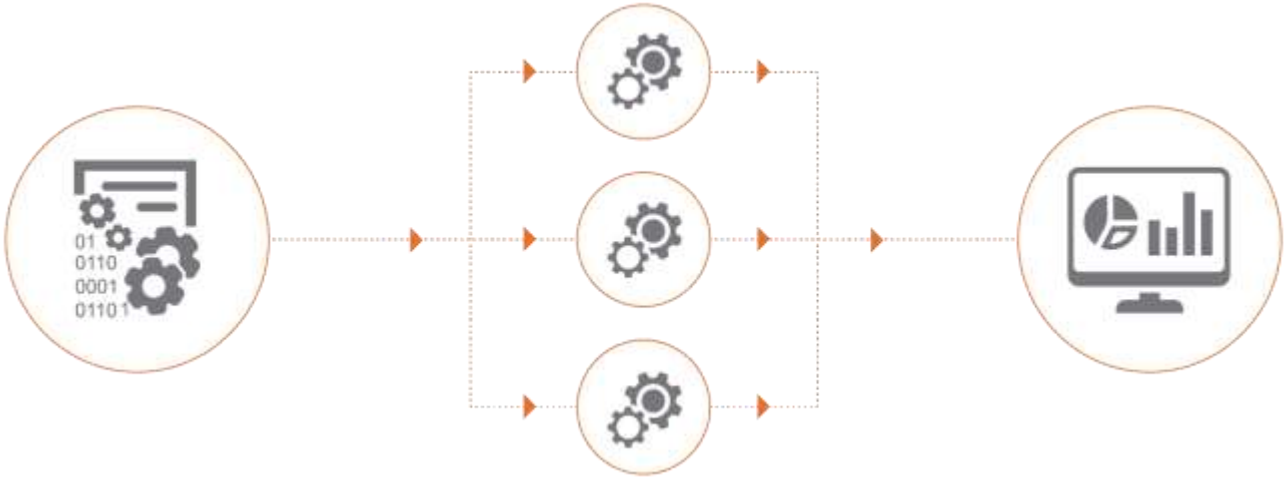
CLOUD SERVICES

- Cloud Services - The cloud services layer is a collection of services that coordinate activities across Snowflake. These services tie together all of the different components of Snowflake in order to process user requests, from login to query dispatch. The cloud services layer also runs on compute instances provisioned by Snowflake from Amazon EC2.



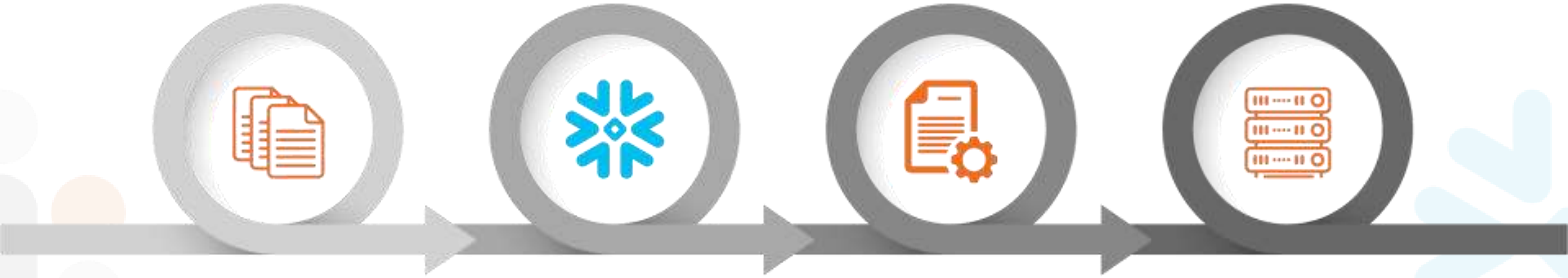
QUERY PROCESSING

- Query Processing - Query execution is performed in the processing layer. Snowflake processes queries using “virtual warehouses”. Each virtual warehouse is an MPP compute cluster composed of multiple compute nodes allocated by Snowflake from Amazon EC2.



DATABASE STORAGE

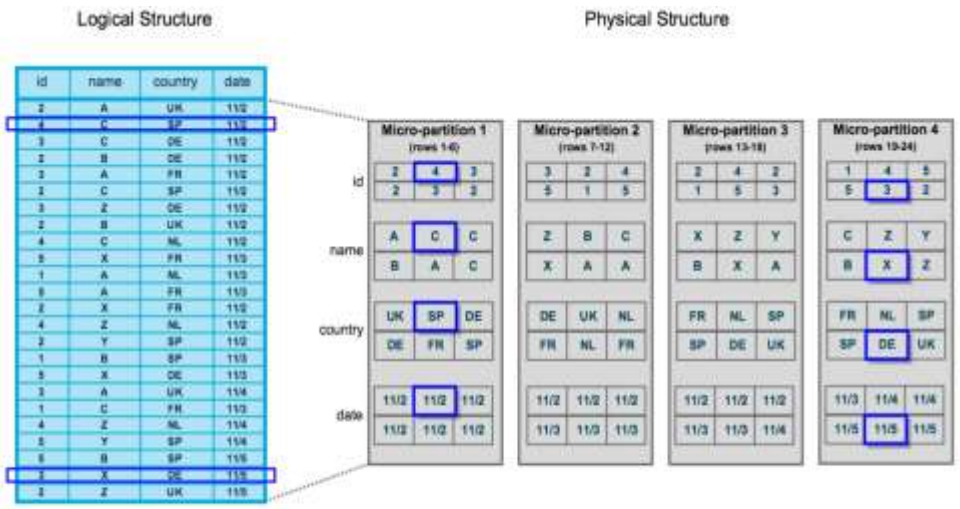
- Database Storage - When data is loaded into Snowflake, Snowflake reorganizes that data into its internal optimized, compressed, columnar format. Snowflake stores this optimized data using Amazon Web Service's S3 (Simple Storage Service) cloud storage.



DATABASE STORAGE

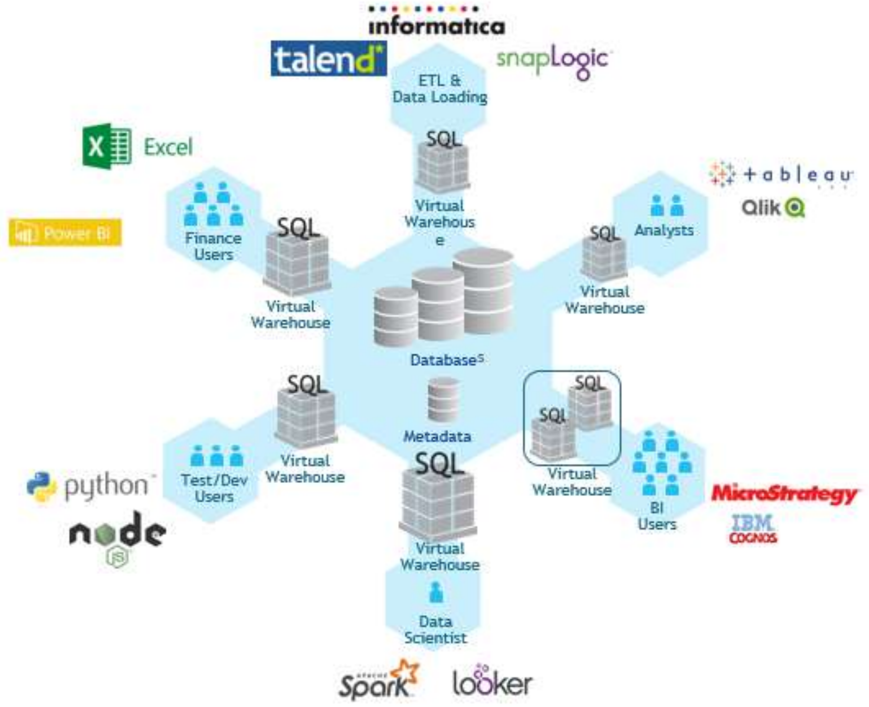
- **Micro-partitions**: All data in Snowflake tables is automatically divided into micro-partitions

- Contiguous units of storage
- 50 MB and 500 MB of uncompressed data
- enables extremely efficient DML and fine-grained pruning for faster queries
- Cluster Key is used to co-locate data within same micro-partitions
- User can create custom cluster key


Logical Structure
Physical Structure
id
name
country
date
Micro-partition 1 (rows 1-6)
Micro-partition 2 (rows 7-12)
Micro-partition 3 (rows 13-18)
Micro-partition 4 (rows 19-24)

https://docs.snowflake.net/manuals/user-guide/tables-micro-partitions.html

SNOWFLAKE OVERVIEW



CONNECTING TO SNOWFLAKE

Snowflake supports multiple ways of connecting to the service:

- A web-based user interface from which all aspects of managing and using Snowflake can be accessed.
- Command line clients (e.g. SnowSQL) which can also access all aspects of managing and using Snowflake.
- ODBC and JDBC drivers that can be used by other applications (e.g. Tableau) to connect to Snowflake.
- Native connectors (e.g. Python) that can be used to develop applications for connecting to Snowflake.
- Third-party connectors that can be used to connect applications such as ETL tools (e.g. Informatica) and BI tools to Snowflake.

SNOWFLAKE EDITIONS AND REGIONS

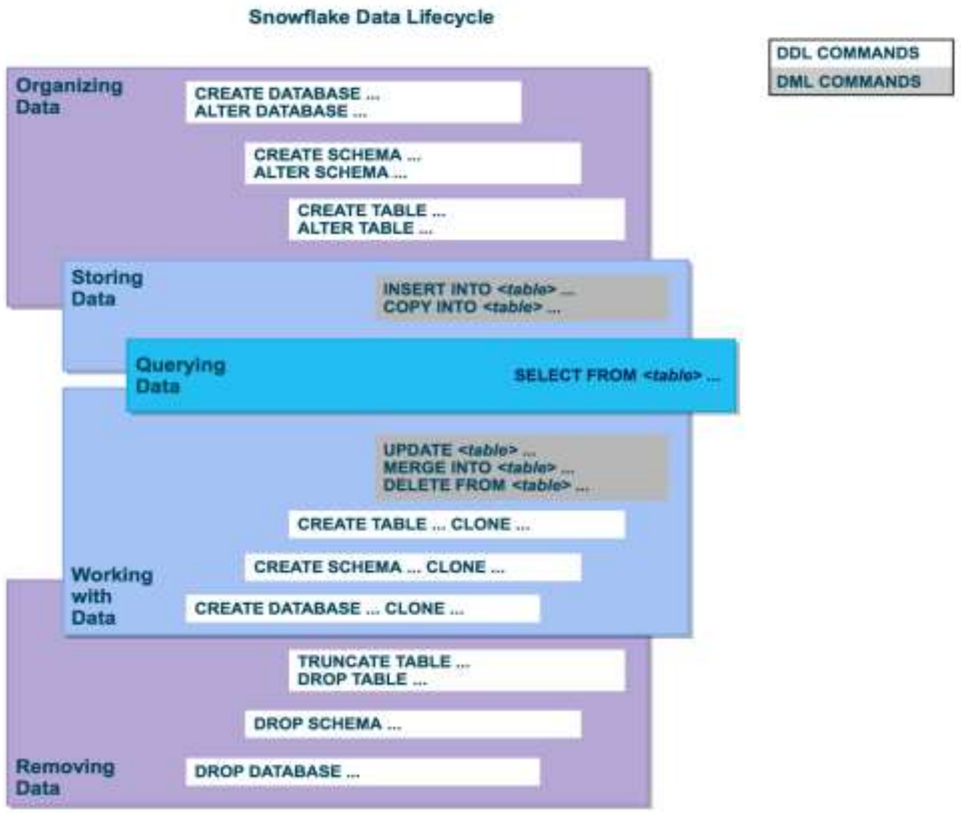
- The Snowflake Edition and Region that your organization choose determine the unit costs for the credits you consume and the data storage you use. Another factor that impacts unit costs is whether you have an *On Demand* or *Capacity* account:
- On Demand: Usage-based pricing with no long-term licensing requirements.
- Capacity: Discounted pricing based on an up-front Capacity commitment.
- For pricing details, go to the pricing page on the Snowflake website.

OVERVIEW OF KEY FEATURES

- Security and Data Protection
- SQL Support
- Tools and Interfaces
- Connectivity
- Data Import and Export

DATA LIFECYCLE

SNOWFLAKE DATA LIFECYCLE



CONTINUOUS DATA PROTECTION

Continuous Data Protection encompasses a comprehensive set of features that help protect data stored in Snowflake against human error, malicious acts, and software or hardware failure. The features include:

- Network policies for granting or restricting users access to the site based on their IP address (i.e. IP whitelisting).
- User verification required for account access, including support for:
  - Multi-factor authentication (standard for all accounts; enabled per user by Snowflake on request).
  - Federated authentication (for Snowflake Enterprise Edition).
- Access to all objects in the system through security roles.
- Automatic encryption of data:
  - 256-bit AES encryption of data at rest and in transit.
  - 128-bit or 256-bit AES encryption of all files staged for bulk loading/unloading data.
- Maintenance of historical data (i.e. data that has been changed or deleted):
  - Querying and restoring historical data using Snowflake Time Travel.
  - Disaster recovery of historical data (by Snowflake) through Snowflake Fail-safe.

SUMMARY OF SUPPORTED SQL

- Snowflake supports most SQL defined in ANSI SQL-99, as well as parts of the SQL-2003 analytic extensions.
- https://docs.snowflake.net/manuals/user-guide/intro-summary-sql.html

SUMMARY OF DATA LOADING FEATURES

Feature	Supported Type/Component	Notes
Sources	Local files	First upload into a Snowflake internal staging location or external location (i.e. S3 bucket), then load into table.
	Files in S3	Can be any user-supplied bucket in Amazon S3.
File Formats	Delimited (CSV, TSV, etc.)	Any single-character delimiter is supported; default is comma (i.e. CSV).
	JSON	
	Avro	Includes support for Snappy compression in Avro files.
	Parquet	
	XML	Supported as a preview feature. For more information, see Preview Features.
File Compression	gzip	During upload, uncompressed files are automatically compressed using gzip, unless compression is explicitly set to NONE.
Compressed Files	gzip, bzip2, deflate, raw deflate	Explicitly set the compression type to one of these values or use AUTO (default) to automatically detect the type.
Encoding	UTF-8	

SESSION 2

DRIVERS AND CONNECTORS

- Overview of the Ecosystem
- Snowflake CLI Client (SnowSQL)
- Snowflake Connector for Python
- Snowflake Connector for Spark
- Snowflake Node.js Driver
- Snowflake JDBC Driver
- Snowflake ODBC Driver

OVERVIEW OF THE ECOSYSTEM

- Snowflake works with a broad array of industry-leading tools and technologies.
- From ETL to business intelligence and advanced analytics, our native solutions, partnerships, and other integrations enable you to leverage Snowflake to quickly and easily deliver insights into your data.

In this Topic:

JDBC or ODBC Drivers
Data Integration and Management
Business Intelligence
Advanced Analytics
Programmatic Interfaces
SQL Editing and Querying Tools

JDBC OR ODBC DRIVERS

- If you need to connect to Snowflake using a tool or technology that is not listed here, we suggest first attempting to connect through our JDBC or ODBC drivers.
- These drivers provide general, multi-purpose connection functionality for most tools and technologies that support JDBC or ODBC.
- https://docs.snowflake.net/manuals/user-guide/ecosystem.html

SNOWFLAKE CLI CLIENT (SNOWSQL)

- SnowSQL is the next-generation command line client for connecting to Snowflake to execute SQL queries and perform all DDL and DML operations, including loading data into and unloading data out of database tables.

- SnowSQL (snowsql executable) can be run as an interactive shell or in batch mode through stdin or using the -f option.

- https://docs.snowflake.net/manuals/user-guide/snowsql.html

SNOWFLAKE CONNECTOR FOR PYTHON

- The Snowflake Connector for Python provides an interface for developing Python applications that can connect to Snowflake and perform all standard operations. It provides a programming alternative to developing applications in Java or C/C++ using the Snowflake JDBC or ODBC drivers.
- The connector is a native, pure Python package that has no dependencies on JDBC or ODBC. It can be installed using pip on Linux, Mac OSX, and Windows platforms where either Python 2.7.9 (or higher) or Python 3.4.3 (or higher) is installed.
- The connector supports developing applications using the Python Database API v2 specification (PEP-249), including using the following standard API objects:
- Connection objects for connecting to Snowflake.
- Cursor objects for executing DDL/DML statements and queries.
- SnowSQL, the command line client provided by Snowflake, is an example of an application developed using the connector.

- https://docs.snowflake.net/manuals/user-guide/python-connector.html

SNOWFLAKE CONNECTOR FOR SPARK

- The Snowflake Connector for Spark, provided as a Spark package (spark-snowflake), brings Snowflake into the Spark ecosystem, enabling Snowflake to be used as a Spark data source, similar to other data sources (PostgreSQL, HDFS, S3, etc.).

- https://docs.snowflake.net/manuals/user-guide/spark-connector.html

SNOWFLAKE NODE.JS DRIVER

- Written in pure JavaScript, the Snowflake Node.js driver provides a native asynchronous Node.js interface to Snowflake. The driver supports Node.js v4.0.0 and higher. For more information about Node.js, see nodejs.org.

The typical workflow for using the driver is:

- Establish a connection with Snowflake.
- Execute statements, e.g. queries and DDL/DML commands.
- Consume the results.
- Terminate the connection.

- https://docs.snowflake.net/manuals/user-guide/nodejs-driver.html

SNOWFLAKE JDBC DRIVER

- Snowflake provides a JDBC type 4 driver that supports core JDBC functionality. The JDBC driver must be installed in a 64-bit environment and requires Java 1.7 (or higher).

- The driver can be used with most client tools/applications that support JDBC for connecting to a database server. sfsql, the now-deprecated command line client provided by Snowflake, is an example of a JDBC-based application.

- https://docs.snowflake.net/manuals/user-guide/jdbc.html

SNOWFLAKE ODBC DRIVER

- Snowflake provides an ODBC driver for connecting to Snowflake using ODBC-based client applications.
- Snowflake provides specific versions of the driver for the following platforms:

Microsoft Windows (32-bit and 64-bit)

Mac OS X, v10.9 or higher

Linux (64-bit), developed and tested with:

- CentOS 6 or higher
- Ubuntu 14

- https://docs.snowflake.net/manuals/user-guide/odbc.html

PRACTICE SESSION/DEMO

- Download snowSQL client & ODBC driver
- Install these
- Check if PATH variable is updated with snowSQL path
- Login to SnowSQL with your credentials

WAREHOUSE ADMINISTRATION CAPABILITIES

PRE-REQ - **Install Of Drivers And Snowsql**

- Overview of Using the Web Interface
- Resource Monitors

USING THE WEB INTERFACE

- Creating and modifying all Snowflake objects, including virtual warehouses, databases, and all database objects
- Creating and managing users and other account-level objects, if you have the necessary administrator roles
- Loading data into tables
- Submitting and monitoring queries
- Admin Control

DOCUMENTATION, SUPPORT, DOWNLOADS AND HELP

- □ **View the Snowflake Documentation**
- □ **Visit the Support Portal**
- □ **Download**
- □ **Help Panel**

MANAGING RESOURCE MONITOR

- A virtual warehouse consumes Snowflake credits for each hour (or portion of an hour) that it runs. The number of credits consumed depends on the size of the warehouse and how long it runs.
- To help control costs and avoid unexpected credit usage related to using warehouses, Snowflake provides resource monitors.
- Resource Monitors can be used by account administrators to impose limits on the number of Snowflake credits that warehouses can consume within each monthly billing period.
- Resource monitors provide fine-grained control, allowing limits to be applied selectively to specific roles.

RESOURCE MONITOR PARAMETERS

- Credit quota:

The number of Snowflake credits allocated to the monitor for each monthly billing period.

- Credit usage:

The number of Snowflake credits consumed during the current monthly billing period.

- Triggers:

A trigger specifies a credit quota threshold (as a percentage) and an action to perform when the threshold is reached. If credit usage reaches the threshold within the monthly billing period, the trigger fires and the action is performed. The following actions are supported:

- Suspend a warehouse once the statements being executed by the warehouse have completed.

- Suspend a warehouse immediately, which aborts any statements being executed by the warehouse at the moment.

RESOURCE MONITOR DDL

Snowflake provides the following DDL commands for creating, managing, and using resource monitors:

- CREATE RESOURCE MONITOR
- ALTER RESOURCE MONITOR
- SHOW RESOURCE MONITORS
- DROP RESOURCE MONITOR

CREATE

- Syntax

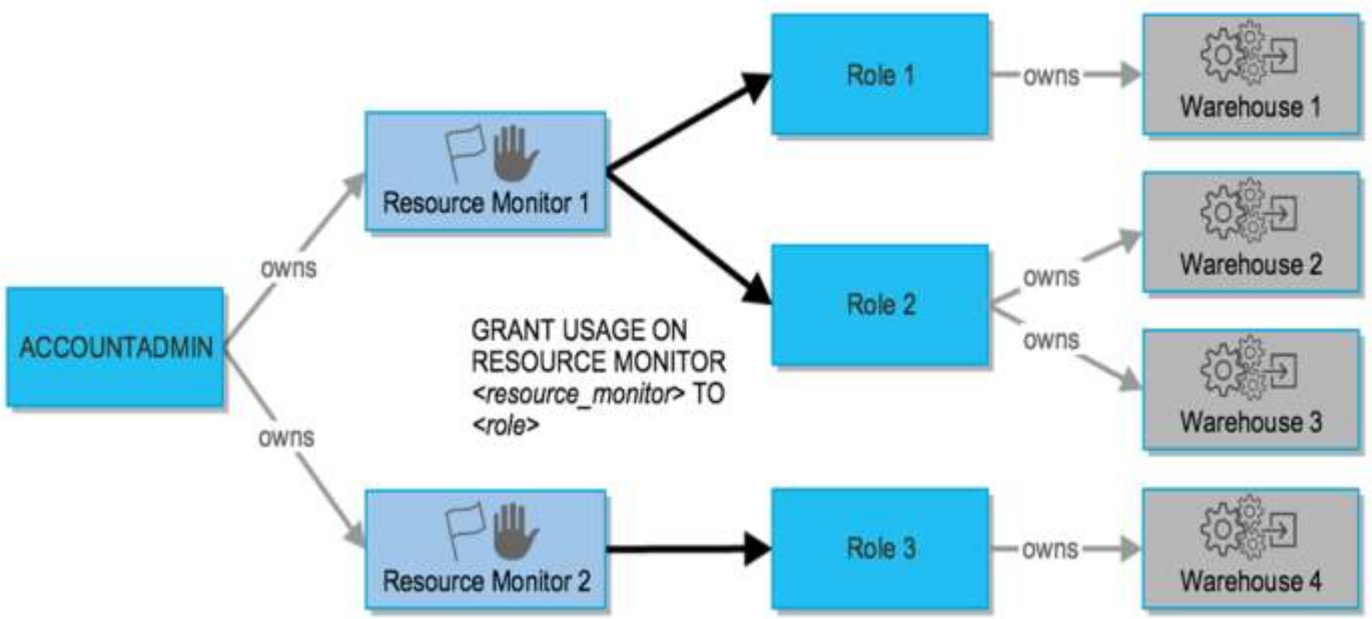
```
CREATE [ OR REPLACE ] RESOURCE MONITOR <identifier> WITH
    [ CREDIT_QUOTA = <number> ]
    [ TRIGGERS <trigger1_definition> <trigger2_definition> ... ]
```

- Where:

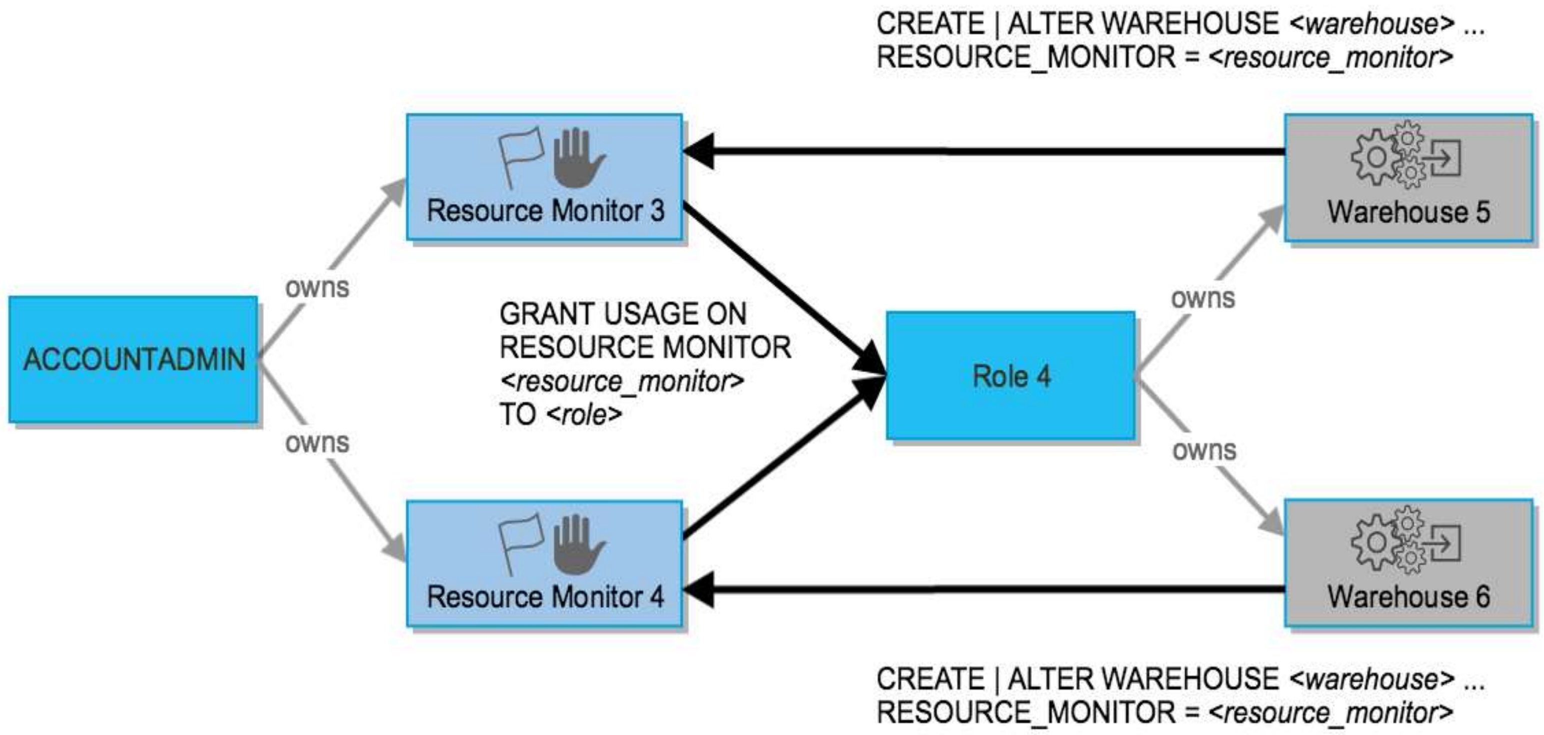
```
-- <trigger_definition> is:

   ON <threshold> PERCENT DO <action>
```

GRANTING RESOURCE MONITOR USAGE TO ROLES (REQUIRED)



EXPLICITLY ASSIGNING A RESOURCE MONITOR TO A WAREHOUSE (OPTIONAL)


CREATE | ALTER WAREHOUSE <*warehouse*> ...
RESOURCE_MONITOR = <*resource_monitor*>
Resource Monitor 3
Warehouse 5
owns
ACCOUNTADMIN
GRANT USAGE ON
RESOURCE MONITOR
<*resource_monitor*>
TO <*role*>
Role 4
owns
owns
owns
Resource Monitor 4
Warehouse 6
CREATE | ALTER WAREHOUSE <*warehouse*> ...
RESOURCE_MONITOR = <*resource_monitor*>

SESSION 6

DATA DEFINTION AND MANIPULATION

- PRE-REQ **Creating Of Warehouse And DB , Applying Roles & Security**
- DDL Commands
- DML Commands

DATA DEFINITION LANGUAGE COMMANDS

- Data Definition Language (DDL) commands are used to create, manipulate, and modify objects in Snowflake, such as users, virtual warehouses, databases, schemas, tables, views, columns, and functions.

- The following commands serve as the base for all DDL commands:

- ALTER
- COMMENT
- CREATE
- DESC
- DROP
- SHOW
- USE

DATA DEFINITION LANGUAGE COMMANDS

- **ALTER**

Modifies the metadata of an object (account/user or database) or the parameters for a session.

Syntax: ALTER <object_type> <identifier> <command> ...

- **Account/User Objects:**

ALTER ACCOUNT
ALTER NETWORK POLICY
ALTER USER
ALTER ROLE
ALTER WAREHOUSE
ALTER DATABASE

DATA DEFINITION LANGUAGE COMMANDS

- **COMMENT**

Add a comment or overwrite an existing comment for an existing object.

Syntax: COMMENT [IF EXISTS] ON <object_type> <identifier> IS '<string_literal>';

COMMENT [IF EXISTS] ON COLUMN <table_identifier>.<column_identifier> IS '<string_literal>';

Comments can be added to all objects, e.g. :

- users
- roles
- warehouses
- databases
- schemas
- tables
- views

DATA DEFINITION LANGUAGE COMMANDS

- **CREATE**

Creates a new object of the specified type.

Syntax: CREATE [OR REPLACE] <object_type> [IF NOT EXISTS] <identifier>
[<object_properties>]
[<object_params>]
[COMMENT = '<string_literal>']

- **Account/User/Database Objects:**

CREATE USER
CREATE ROLE
CREATE WAREHOUSE
CREATE DATABASE
CREATE SCHEMA
CREATE TABLE
CREATE VIEW

DATA DEFINITION LANGUAGE COMMANDS

- **DESC**

Describes the details for the specified object.

Syntax: DESC <target_type> <identifier**>**

- **Account/User/Database:**

DESC USER
DESC WAREHOUSE
DESC DATABASE
DESC NETWORK POLICY
DESC SCHEMA
DESC TABLE
DESC VIEW

DATA DEFINITION LANGUAGE COMMANDS

- **DROP**

Removes the specified object from the system.

Syntax: DROP <target_type> [IF EXISTS] <identifier> [CASCADE **|** *RESTRICT]*

- **Account/User/Database:**

DROP USER
DROP ROLE
DROP WAREHOUSE
DROP DATABASE
DROP SCHEMA
DROP TABLE
DROP VIEW
DROP FUNCTION

DATA DEFINITION LANGUAGE COMMANDS

- **SHOW**

Lists all the existing objects for the specified object type.

Syntax: SHOW <object_types> [LIKE '<pattern>'] [IN <parent_object_name>]

SHOW <object_types> [LIKE '<pattern>'] [IN <parent_object_type> [<parent_object_name>]]

- **Account/User/Database:**

SHOW PARAMETERS
SHOW FUNCTIONS
SHOW USERS
SHOW TRANSACTIONS
SHOW SCHEMAS
SHOW OBJECTS
SHOW TABLES

DATA DEFINITION LANGUAGE COMMANDS

- **USE**

- Specifies the database, schema, warehouse, or role to use for the current user session:

- Specifying a warehouse for a session enables executing DML statements and queries in the session.
- If a database or schema are not specified for a session, any objects referenced in SQL statements
  or querys executed in the session must be qualified with the database and schema, also known as
  a namespace, for the object.

Syntax: USE ROLE <role_identifier>

USE [DATABASE] <db_identifier>

USE [SCHEMA] [<db_identifier>.]<schema_identifier>

DATA MANIPULATION LANGUAGE COMMANDS

DML commands for inserting, deleting, updating, and merging data in Snowflake tables:

- INSERT
- INSERT (Multi-table)
- MERGE
- UPDATE
- DELETE

DATA MANIPULATION LANGUAGE COMMANDS

INSERT:

Insert data into a table.

Syntax:

INSERT INTO <target_table> (<column1>,<column3>,<column3>) VALUES (<value1>,<value2>,<value3>)

INSERT INTO <target_table> VALUES (<value1>, <value2>, ...)

DATA MANIPULATION LANGUAGE COMMANDS

INSERT (Multi-table):

Inserts one or more rows with specified column values into multiple tables. Supports both conditional and unconditional inserts.

Syntax:

```
INSERT ALL
  intoClause+
<subquery>

INSERT { FIRST | ALL }
  { WHEN <condition> THEN intoClause+ }+
  [ ELSE intoClause ]
<subquery>
```

DATA MANIPULATION LANGUAGE COMMANDS

MERGE:

Inserts, updates, and deletes data in a table based on data in a second table or a subquery. This can be useful if the second table is a change log that contains new rows to be added, modified rows to be updated, and marked rows to be deleted.

```
Syntax: MERGE INTO <target_table> USING <source> ON <join_expr> matchedClause |
notMatchedClause ...

-- Where:
  matchedClause ::=
        WHEN MATCHED [ AND <case_predicate> ] THEN
        DELETE | UPDATE SET <column> = <update_expr> ...

  notMatchedClause ::=
        WHEN NOT MATCHED [ AND <case_predicate> ] THEN
        INSERT [ ( <column_list> ) ] VALUES ( <insert_expr> ...)
```

DATA MANIPULATION LANGUAGE COMMANDS

UPDATE:

Updates specified rows in the target table with new values.

Syntax:

```
UPDATE <target_table>
    SET <column1> = <value>, <column2> = <value>, ...
    [FROM <additional_tables>]
    [WHERE <condition_query>]
```

DATA MANIPULATION LANGUAGE COMMANDS

DELETE:

Remove data from a table.

Syntax:

```
DELETE FROM <target_table>
  [USING <additional_tables>]
  [WHERE <condition_query>];
```

SESSION 5

DATA LOADING AND UNLOADING

- Databases & Tables
- Data Loading
- Data Unloading
- Queries
- Binary Data
- Dates & Timestamps
- Semi-structured Data
- **Snowflake Time Travel & Fail-safe**
- Sample Data Sets

DATA LOADING

- Snowflake supports bulk import (i.e. loading) of data from one or more files into tables in Snowflake databases

- Supported file formats for data loading:

- Any flat, delimited plain text format (comma-separated values, tab-separated values, etc.)
- Semi-structured data in JSON, Avro, Parquet, or XML format (XML is currently supported as a preview feature)
- Supported character encoding format for data files: UTF-8

STAGING IN SNOWFLAKE

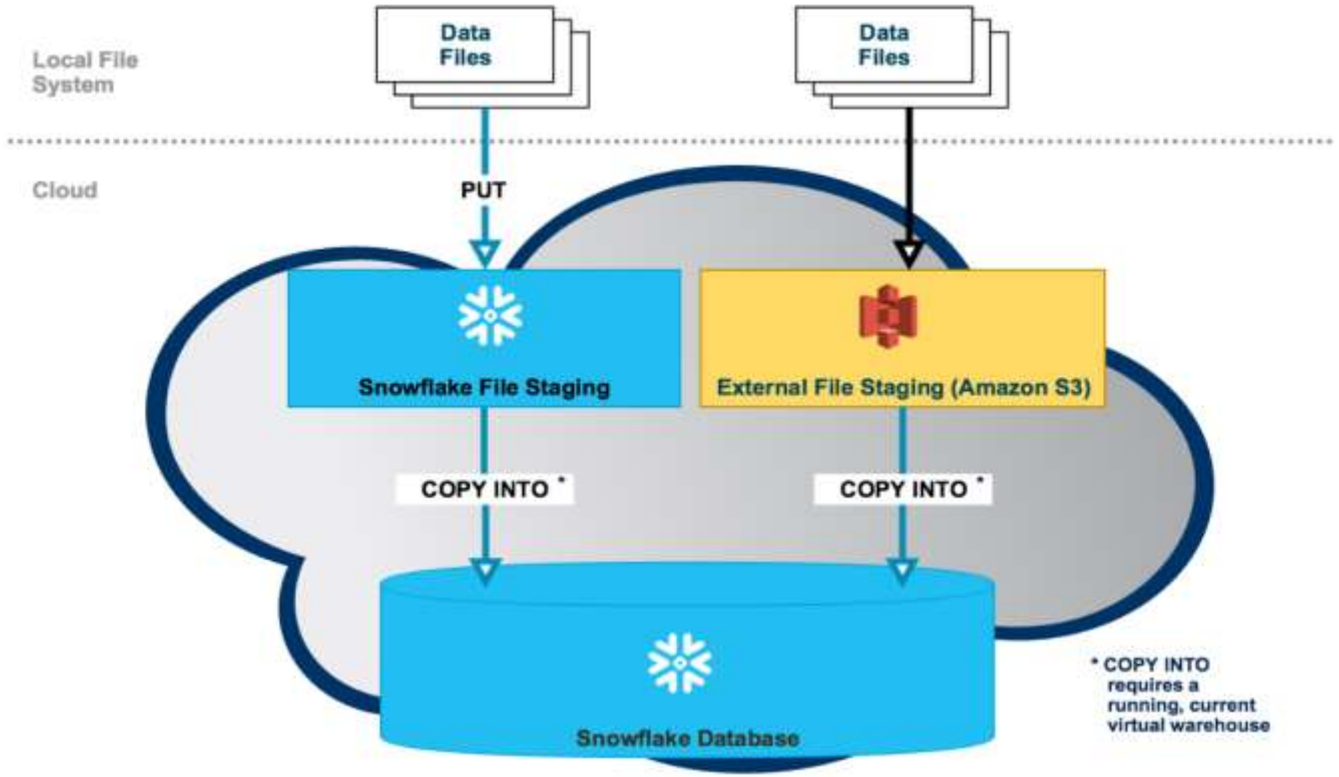
Internal Locations – use PUT while uploading data file to staging

- User Staging location (inside VWH)– accessed using “~”
- Table Staging location (inside VWH) – accessed using “%”
- Internal Stage (inside database)

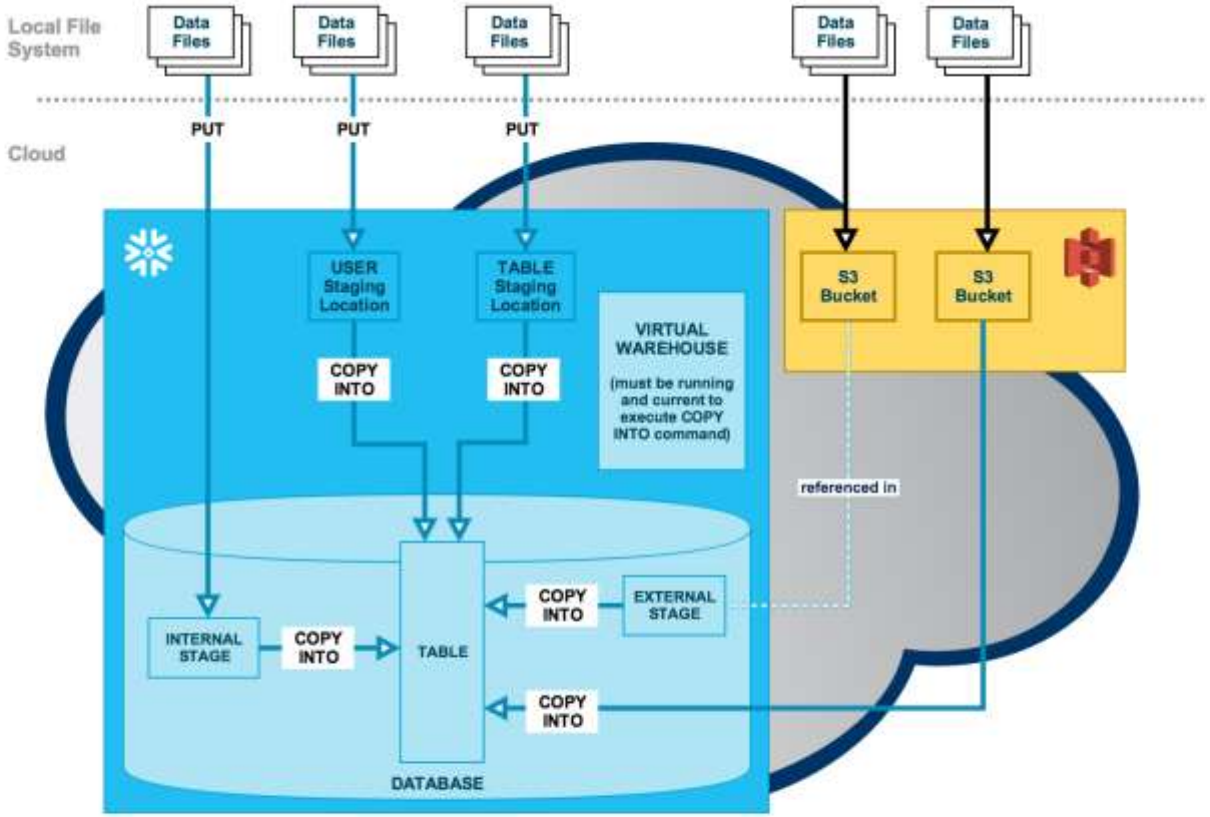
External Locations

- S3 Buckets

DATA LOADING



DATA LOADING

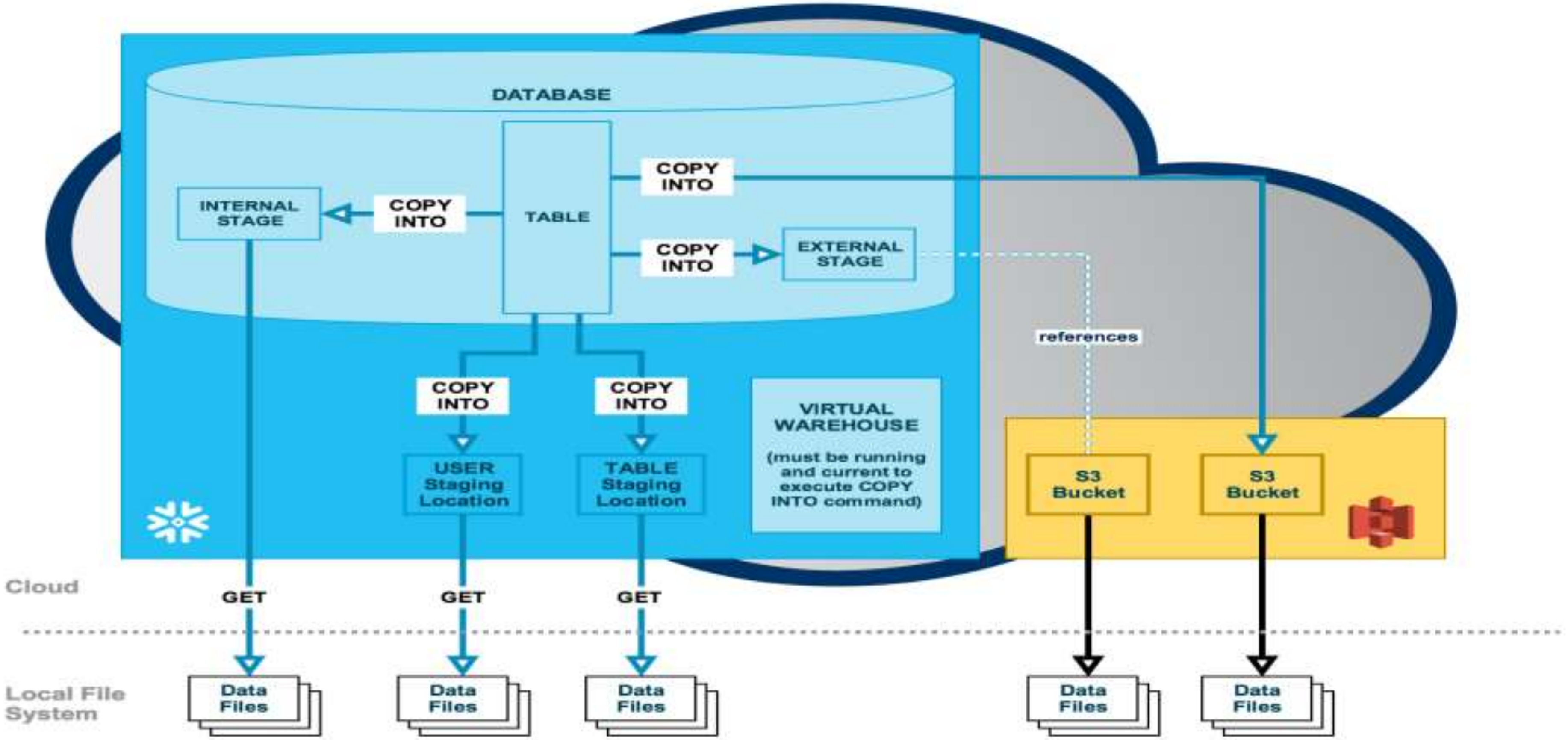


DATA UNLOADING

- Similar to data loading, Snowflake supports bulk export (i.e. unloading) of data from database tables into flat, delimited text files.

- *Required Information and Metadata for Data Unloading*
- *Unloading into Snowflake Locations*
- *Unloading into External Locations*

DATA UNLOADING PROCESS



DATA UNLOADING USING QUERIES

- Snowflake supports specifying a SELECT statement instead of a table in the COPY INTO location command.

- SELECT statements in COPY INTO location support the full syntax and semantics of Snowflake SQL queries, including JOIN clauses, which enables downloading data from multiple tables.

QUERIES

- Snowflake supports standard SQL, including a subset of ANSI SQL-99 and the ANSI SQL-2003 analytic extensions

In this Topic:
Querying Semi-structured Data
Querying Metadata Columns
Using Sequences in Queries

QUERYING METADATA COLUMNS

- For each data file staged to an internal stage or staging location, Snowflake can generate additional columns which contain metadata about the data being copied.

- Currently, the following metadata columns can be queried or copied into tables:
- **METADATA$FILENAME -** Name of the staged data file the current row belongs to
- **METADATA$FILE_ROW_NUMBER -** Row number for each record in the container staged data file

In this Topic:
Querying Metadata Columns in Staged Files
Loading Metadata Columns into a Table

QUERYING METADATA COLUMNS IN STAGED FILES

- You can query metadata columns just as you would other columns.

- Example:

```
-- Create an internal stage
create or replace stage mystage;

-- Stage a data file
put file:///tmp/store.csv @mystage;

-- Query the filename and row number metadata columns and the regular data columns in the staged
file:
select metadata$filename, metadata$file_row_number, t.$1, t.$2, t.$3, t.$4 from
@mystage/store.csv.gz t;
```

- NOTE: NULL in t.$4

LOADING METADATA COLUMNS INTO A TABLE

- The COPY INTO table command can copy metadata from staged data files into a target table.

- Example

```
create or replace table rajtable (
Filename string, row_id integer, order_id string, order_date date, ship_date date);
```

If header is removed from file

```
copy into rajtable(Filename, row_id, order_id, order_date, ship_date)
from (select metadata$filename, metadata$file_row_number, t.$1, t.$2, t.$3
from @mystage/store.csv.gz t);
```

If header is included use a File Format

```
copy into rajtable(Filename, row_id, order_id, order_date, ship_date)
from (select metadata$filename, metadata$file_row_number, t.$1, t.$2, t.$3
from @mystage/store.csv.gz t) FILE_FORMAT = ( FORMAT_NAME =‘CSV_WITH_HEADER’ TYPE = ‘gzip’);

select * from rajtable;
```

BINARY DATA

- Snowflake’s BINARY data type allows you to store arbitrary binary data.

In this Topic:
Binary Input and Output
Using Binary Data

BINARY INPUT AND OUTPUT

- Supported Binary Formats
  - Hex
  - Base64
  - UTF-8

- Session Parameters for Binary Values

- **BINARY_INPUT_FORMAT** - Used for loading data into Snowflake and for performing conversion to binary in the one-argument version of TO_CHAR , TO_VARCHAR

- **BINARY_OUTPUT_FORMAT -** Used for displaying binary result sets, unloading data from Snowflake, and performing conversion to string in the one-argument version of TO_BINARY.

USING BINARY DATA

Converting Between Hex and Base64:

The BINARY data type can be used as an intermediate step when converting between hex and base64 strings.

- Convert from hex to base64 using TO_CHAR , TO_VARCHAR:

```
select c1, to_char(to_binary(c1, 'hex'), 'base64') from hex_strings;
```

- Convert from base64 to hex:

```
select c1, to_char(to_binary(c1, 'base64'), 'hex') from base64_strings;
```

USING BINARY DATA

Converting Between Text and UTF-8 Bytes:

Strings in Snowflake are composed of Unicode characters, while binary values are composed of bytes. By converting a string to a binary value with the UTF-8 format, we can directly manipulate the bytes that make up the Unicode characters.

- Convert single-character strings to their UTF-8 representation in bytes using TO_BINARY:

```
select c1, to_binary(c1, 'utf-8') from characters;
```

- Convert a UTF-8 byte sequence to a string using TO_CHAR , TO_VARCHAR:

```
select to_char(X'41424320E29D84', 'utf-8');
```

DATES & TIMESTAMPS

- Snowflake’s date, timestamp, and time zone support enables you to calculate and store consistent information about the time of events and transactions.

- Session Parameters for Dates, Times, and Timestamps:

Input Formats:

- DATE_INPUT_FORMAT
- TIME_INPUT_FORMAT
- TIMESTAMP_INPUT_FORMAT

Output Formats:

- DATE_OUTPUT_FORMAT
- TIME_OUTPUT_FORMAT
- TIMESTAMP_OUTPUT_FORMAT

DATES & TIMESTAMPS

- **DATE_INPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Input format for dates or AUTO. AUTO specifies that Snowflake attempts to automatically detect the format of dates stored in the system during the session. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	AUTO

DATES & TIMESTAMPS

- **TIME_INPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Input format for times or AUTO. AUTO specifies that Snowflake attempts to automatically detect the format of times stored in the system during the session. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	AUTO

DATES & TIMESTAMPS

- **TIMESTAMP_INPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Input format for timestamps or AUTO. AUTO specifies that Snowflake attempts to automatically detect the format of timestamps stored in the system during the session. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	AUTO

DATES & TIMESTAMPS

- **DATE_OUTPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Display format for date. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	YYYY-MM-DD

DATES & TIMESTAMPS

- **TIME_OUTPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Display format for TIME. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	HH24:MI:SS

DATES & TIMESTAMPS

- **TIMESTAMP_OUTPUT_FORMAT**

Type:	Session — Can be set for Account ▸ User ▸ Session
Data Type:	String
Description:	Display format for TIMESTAMP. For more information about date and time input and output formats, see Date and Time Input / Output.
Default Value:	YYYY-MM-DD HH24:MI:SS.FF3 TZHTZM

SEMI-STRUCTURED DATA

- To support these new types of data, semi-structured data formats, such as JSON, Avro, Parquet, and XML, with their support for flexible schemas, have become popular standards for transporting and storing data.

Snowflake provides native support for semi-structured data, including:

- Flexible-schema data types for loading semi-structured data without transformation

- Automatic conversion of data to optimized internal storage format

- Database optimization for fast and efficient SQL querying

SNOWFLAKE TIME TRAVEL & FAIL-SAFE

Snowflake provides powerful CDP features for ensuring the maintenance and availability of your historical data (i.e. data that has been changed or deleted):

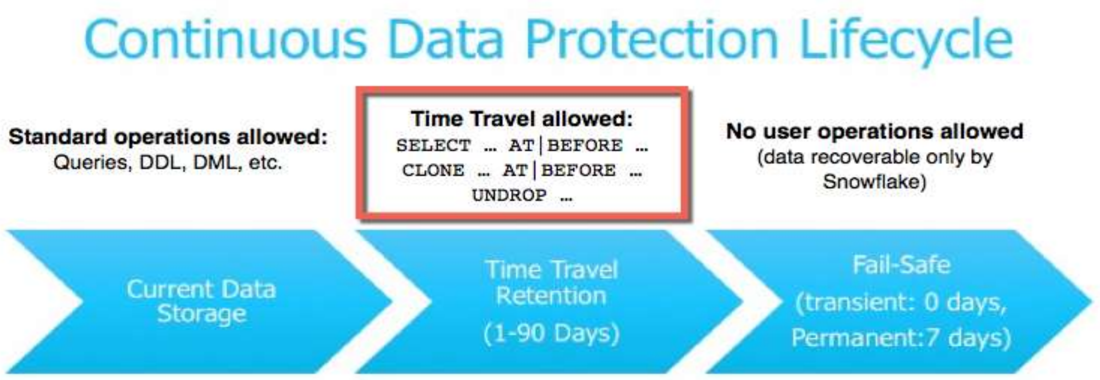
- Querying, cloning, and restoring historical data in tables, schemas, and databases for up to 90 days through Snowflake Time Travel.
- Disaster recovery of historical data (by Snowflake) through Snowflake Fail-safe.

Next Topics:
Understanding & Using Time Travel
Understanding & Viewing Fail-safe
Storage Costs for Time Travel and Fail-safe

UNDERSTANDING & USING TIME TRAVEL

- Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It provides powerful tools for restoring data that may have been accidentally or intentionally deleted

- What is Time Travel?



UNDERSTANDING & USING TIME TRAVEL

Data Retention Period:

The standard retention period is 1 day (24 hours) and is automatically enabled for all Snowflake accounts:

- For Snowflake Standard Edition, the retention period can be changed to 0 at the account and object levels (for databases, schemas, and tables).
- For Snowflake Enterprise Edition, the retention period can be changed as follows:
- For temporary or transient objects, 0 or 1 day.
- For permanent objects, any value from 0 up to 90 days.

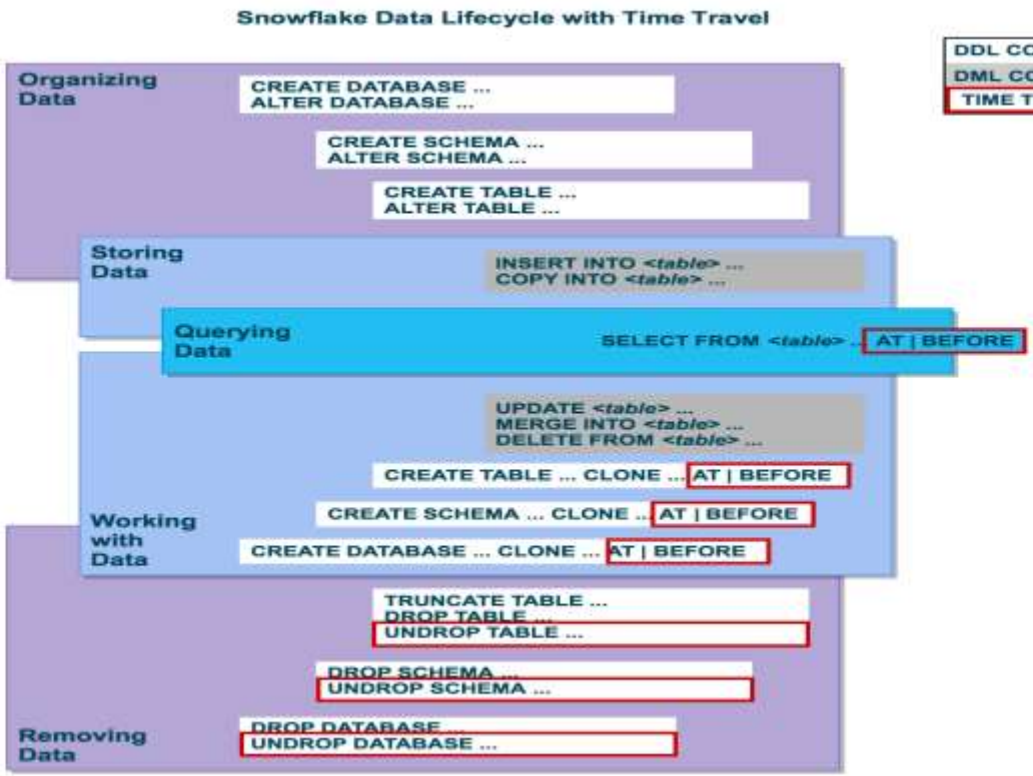
UNDERSTANDING & USING TIME TRAVEL

Time Travel SQL Extensions:

To support Time Travel actions, the following SQL extensions are provided:

- **AT | BEFORE** clause which can be specified in SELECT statements and **CREATE...CLONE** commands (immediately after the table name). The clause uses one of the following parameters to pinpoint the exact historical data you wish to access:
  - TIMESTAMP
  - OFFSET (time difference in seconds from the present time)
  - STATEMENT (identifier for statement, e.g. query ID)
- **UNDROP** command for tables, schemas, and databases.

UNDERSTANDING & USING TIME TRAVEL



☐	DDL COMMANDS
▨	DML COMMANDS
🟥	TIME TRAVEL EXTENSIONS

Dropping and Restoring Objects:

To drop a table, schema, or database, use the following commands:

- DROP TABLE
- DROP SCHEMA
- DROP DATABASE

A dropped object that has not been purged from the system can be restored using the following commands:

- UNDROP TABLE
- UNDROP SCHEMA
- UNDROP DATABASE

CONTENT

DATA TYPES

- Snowflake supports most basic SQL data types. Data types are automatically coerced when necessary and appropriate.

- Different Data Types:

- **Numeric Data Types**
- **Text and Binary Data Types**
- **Logical Data Types**
- **Date and Time Data Types**
- **Semi-Structured Data Types**
- **Unsupported Data Types**

NUMERIC DATA TYPES

- **NUMBER**

Fixed-point numbers of up to 38 digits, with precision and scale specified. By default, precision is 38
and scale is 0, i.e. NUMBER(38, 0).

- **FLOAT, FLOAT4 , FLOAT8**

Floating point numbers. Snowflake uses double-precision (64 bit) IEEE 754 floating point numbers.

EXAMPLE OF ALL NUMERIC DATA TYPES

```
create or replace table test_number(num number,
num10 number(10,1),
dec decimal(20,2),
numeric numeric(30,3),
 int int,
integer integer,
d double, f
float,
dp double precision,
r real );
```

TEXT AND BINARY DATA TYPES

- **VARCHAR**

Text data type. The maximum length is 16MB (uncompressed). The maximum number of Unicode characters that can be stored in a VARCHAR column depends on whether the characters are single-byte or multi-byte:

Single-byte:	16,777,216
Multi-byte:	Between 8,388,609 (2 bytes per character) and 4,194,304 (4 bytes per character)

- **BINARY**

BINARY data type. The maximum length is 8MB. Unlike VARCHAR, the BINARY data type has no notion of Unicode characters, so the length is always measured in terms of bytes.

LOGICAL DATA TYPES

- **BOOLEAN**

BOOLEAN can have TRUE or FALSE values. BOOLEAN can also have an “unknown” value, which is represented by NULL. Boolean columns can be used in expressions (e.g. SELECT list), as well as predicates (e.g. WHERE clause).

BOOLEAN Example:

```
create or replace table test_Boolean
( b boolean,
 n number,
 s string);

insert into test_boolean values (true, 1, 'yes'), (false, 0, 'no'), (null, null, null);

select * from test_boolean;
```

DATE AND TIME DATA TYPES

- **DATE**

Snowflake supports a single DATE data type for storing dates (without time). DATE accepts dates in the most common forms such as YYYY-MM-DD or DD-MON-YYYY etc. All accepted timestamps are valid inputs for dates as well

- **TIME**

Snowflake supports a single TIME data type for storing times in the form of HH:MI:SS. TIME supports an optional precision parameter for fractional seconds, e.g. TIME(3)

- **TIMESTAMP , TIMESTAMP_***

Snowflake supports three variations of the TIMESTAMP data type, as well as a special TIMESTAMP alias. All TIMESTAMP variations, including the alias, supports an optional precision parameter for fractional seconds, e.g. TIMESTAMP(3).

QUERYING SEMI-STRUCTURED DATA

- Snowflake supports SQL queries that access semi-structured data using special operators and functions
- the examples in this topic refer to a table named **customers** that contains a single VARIANT column named **src:**

```
select src:name from customers;

     +-----------------+
     | SRC:NAME        |
     |-----------------|
     | "Patrick Jones" |
     | "Anna Glass"    |
     +-----------------+
```

SEMI-STRUCTURED DATA TYPES

The following data types are used to represent arbitrary data structures which can be used to import and operate on semi-structured data (JSON, Avro, etc.)

- **VARIANT**

A tagged universal type, which can store values of any other type, including OBJECT and ARRAY, up to a maximum size of 16MB.

- **OBJECT**

Used to represent collections of key-value pairs, where the key is a non-empty string, and the value is a value of VARIANT type. Snowflake does not currently support explicitly-typed objects.

- **ARRAY**

Used to represent dense or sparse arrays of arbitrary size, where index is a non-negative integer (up to 2^31-1), and values have VARIANT type.

UNSUPPORTED DATA TYPES

- **LOB (Large Object)**

Snowflake does not support LOB data types (e.g. BLOB, CLOB).

SQL FORMAT MODELS

- SQL format models (i.e. literals containing format strings) are used to specify conversion of numeric values to and from text strings. Format models can be used as arguments in the following functions:
- **TO_CHAR , TO_VARCHAR**
- **TO_DECIMAL , TO_NUMBER , TO_NUMERIC**

In this Topic:

FORMAT ELEMENTS AND LITERALS

A format model consists of a sequence of format elements and literals:

- A format element is a case-insensitive sequence of digits or letters, and, in some cases, signs (see tables below). Format elements can be directly concatenated to each other.

- Literals can be quoted (within double quotes) sequences of arbitrary characters, (a double quote is represented as two adjacent double quotes), or sequences of spaces or symbols from the following table:

FORMAT ELEMENTS AND LITERALS

Symbol/Character	Notes
. (period)	In numeric models, treated as a format element when following 0, 9, or X; otherwise preserved as-is.
, (comma)	In numeric models, treated as a format element when following 0, 9, or X; otherwise preserved as-is.
; (semi-colon)	Always preserved as-is.
: (colon)	Always preserved as-is.
- (minus sign)	Always preserved as-is.
= (equal sign)	Always preserved as-is.
/ (forward slash)	Always preserved as-is.
((left parenthesis)	Always preserved as-is.
) (right parenthesis)	Always preserved as-is.
blank space	Always preserved as-is.
tab	Always preserved as-is.

https://docs.snowflake.net/manuals/sql-reference/sql-format-models.html

FORMAT MODIFIERS
GENERIC SPACE HANDLING

- The following format elements control printing and matching input text, and are common to all format models:

Element	Description
_ (underscore)	Nothing printed; optional space on input.
FM	Fill mode modifier; toggles fill mode for subsequent elements.
FX	Exact match modifier; toggles exact match mode for subsequent elements.

NUMERIC FORMAT MODELS

- Numeric format models can be either fixed-position (with explicit placement of digits where 0, 9 or x elements are placed) or text-minimal (TM, TME and TM9 elements) ; these elements cannot be intermingled within the same model.

The following table lists all the supported elements for numeric format models:

- The **Usage** column indicates how each element can be used:
- Fixed: Element can be used only for fixed-position numbers.
- Minimal: Element can be used only for text-minimal models.

CONSTRAINTS

- Constraints define integrity and consistency rules for data stored in tables

- Snowflake provides the following constraint functionality:

- Unique, primary, and foreign keys, and NOT NULL columns.
- Named constraints.
- Single-column and multi-column constraints.
- Creation of constraints inline and out-of-line.
- Support for creation, modification and deletion of constraints.

SUPPORTED CONSTRAINT TYPES

Snowflake supports the following constraint types from the ANSI SQL standard:

- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- NOT NULL

QUERY OPERATORS

Query operators allow queries to be combined using set operators.

- **INTERSECT**

Returns rows from one query's result set which also appear in another query's result set, with duplicate elimination.

Syntax: SELECT ... FROM ... INTERSECT SELECT ... FROM

- **MINUS / EXCEPT**

Removes rows from one query's result set which appear in another query's result set, with duplicate elimination. The MINUS and EXCEPT keywords have the same meaning.

Syntax: SELECT ... FROM ...MINUS   SELECT ... FROM
SELECT ... FROM ...EXCEPT SELECT ... FROM

TRANSACTIONS

- A transaction is a set of SQL statements, both reads and writes, that are processed as a unit.
- All the statements in the transaction are either applied (i.e. committed) or undone (i.e. rolled back).

In this Topic:

SCOPE OF A SNOWFLAKE TRANSACTION

- A transaction is associated with a single session. Multiple sessions cannot share the same transaction.
- A transaction can be started explicitly by executing a BEGIN statement. After a transaction has been started, it must be closed by executing either a COMMIT or ROLLBACK statement.
- If a session with an open transaction is closed, then the open transaction is rolled back.
- Every Snowflake transaction is assigned a unique start time (includes milliseconds), which serves as the ID for the transaction.

SCOPE OF A SNOWFLAKE TRANSACTION

- **Autocommit**

A DML statement executed without explicitly starting a transaction is automatically committed on success or rolled back on failure at the end of the statement.

- **Statement Rollback**

If a DML statement executed in an explicitly-started transaction fails, the changes made by the DML will be rolled back. However, the transaction will be kept open, until the transaction is committed or rolled back.

RESOURCE LOCKING

- Transactional operations acquire locks on resources while they are being modified. This blocks other statements from modifying the resource concurrently until the lock holder completes the changes.

There are two types of locks:

- **Partition:** DML operations acquire partition locks within a table only for micro-partitions they might modify. This schema allows INSERT statements to add micro-partitions to the same table without locking any existing partitions, i.e. inserts can execute without being blocked by other DMLs.

- **Table:** A transaction only acquires a table lock during the commit phase, a length of time proportional to the amount of data inserted or modified in the transaction but overall very short. During the commit phase, INSERT and DML operations can continue. A table lock only blocks a separate transaction from performing a concurrent commit.

ABORTING TRANSACTIONS

- If a transaction is running in a session and the session disconnects abruptly, preventing the transaction from committing or rolling back, the transaction will be left in a detached state, including any locks that the transaction is holding on resources. If this happens, you will likely want/need to abort the transaction.

- To abort a running transaction, the user who started the transaction or an account administrator can call the system function, SYSTEM$ABORT_TRANSACTION.

TABLE FUNCTIONS

- Table functions return a set of rows instead of a single scalar value.

- **General Syntax**

- *SELECT * FROM <input_table> T, LATERAL <table_function>(T.<col1>, ..., <argN>) F;*

In this Topic:
General Syntax
List of Table Functions

LIST OF TABLE FUNCTIONS

Sub-category	Function	Notes
Data Loading	•VALIDATE	For more information, see Data Loading.
Data Generation	•GENERATOR	
Semi-structured Queries	•FLATTEN	For more information, see Querying Semi-structured Data.
Query Results	•RESULT_SCAN	
Query History (Information Schema)	•QUERY_HISTORY , QUERY_HISTORY_BY_*	For more information, see Information Schema.
Warehouse & Storage Usage History (Information Schema)	•DATABASE_STORAGE_USAGE_HISTORY •WAREHOUSE_LOAD_HISTORY •WAREHOUSE_METERING_HISTORY •STAGE_STORAGE_USAGE_HISTORY	For more information, see Information Schema.

USER-DEFINED FUNCTIONS

User-defined functions (UDFs) let you extend the system to perform operations that are not available via the system-defined functions provided by Snowflake.

Snowflake currently supports two types of UDFs:

- *SQL UDFs* evaluate and return an arbitrary SQL expression.
- *JavaScript UDFs* allow you to manipulate data using the JavaScript programming language and runtime.

TYPES OF USER-DEFINED FUNCTIONS

- **SQL**

A SQL UDF evaluates and returns an arbitrary SQL expression. The function definition must be a SQL expression that returns a single value. The expression may be a query expression, though for non-table functions, a query expression must be guaranteed to return at most one row, containing a single column.

- **JavaScript**

JavaScript user-defined functions (UDFs) allow you to manipulate data using the JavaScript programming language and runtime. Functions are created similarly to SQL UDFs.

SESSION 5

- Operators
- All Functions
- Bitwise Expression Functions
- Conditional Expression Functions
- Context Functions
- Numeric Functions
- String & Binary Functions
- Regular Expression (String) Functions
- Date & Time Functions
- Semi-structured Data Functions
- Conversion Functions
- Aggregate Functions
- Window/Analytic Functions
- Miscellaneous Functions

OPERATORS

- Operators for performing arithmetic, comparison, and logical operations on expressions.

In this Topic:
Arithmetic Operators
Comparison Operators
Logical Operators

ARITHMETIC

Operator	Description
a + b	Add two numeric expressions (number or real).
+ a	Return *a*, which can cause implicit conversion of *a* to a numeric value.
a - b	Subtract a numeric expression from another.
- a	Negate the numeric input expression.
a * b	Multiply two numeric expressions.
a / b	Divide two numeric expressions.
a % b	Compute the modulo of numeric expressions *a* per *b*.

COMPARISON

Operator	Description
a = b	*a* is equal to *b*.
a != b	*a* is not equal to *b*.
a <> b	*a* is less than or greater than *b*.
a > b	*a* is greater than *b*.
a >= b	*a* is greater than or equal to *b*.
a < b	*a* is less than *b*.
a <= b	*a* is less than or equal to *b*.

LOGICAL

Operator	Description
a AND b	Matches both inputs.
a OR b	Matches either input.
NOT a	Does not match the input.

ALL FUNCTIONS

- This topic provides a list of all system-defined Snowflake functions, including table functions, in alphabetical order.

- Find the entire list here https://docs.snowflake.net/manuals/sql-reference/functions-all.html

BITWISE EXPRESSION FUNCTIONS

- BITAND
- BITAND_AGG
- BITNOT
- BITOR
- BITOR_AGG
- BITSHIFTLEFT
- BITSHIFTRIGHT
- BITXOR
- BITXOR_AGG

EXAMPLE

- select BITAND(3, 4) from table1
- select BITOR(3, 4) from table1

CONDITIONAL EXPRESSION FUNCTIONS

- BETWEEN
- CASE
- COALESCE
- DECODE
- EQUAL_NULL
- GREATEST
- IFF
- IFNULL
- IN / NOT IN
- IS NULL / IS NOT NULL
- IS_NULL_VALUE
- LEAST
- NULLIF
- NVL
- NVL2
- REGR_VALX
- REGR_VALY
- ZEROIFNULL

EXAMPLE

- select column1, case
        when column1=1 then 'one'
        when column1=2 then 'two'
        else 'other'
    end as result from (values(1),(2),(3)) v;
- select val, iff(val::int = val, 'integer', 'non-integer')
from (select column1 as val
      from values(1.0), (1.1), (-3.1415), (-5.000), (null));
- select a, b, nullif(a,b) from i;

CONTEXT FUNCTIONS

General Context

- CURRENT_CLIENT
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_VERSION
- LOCALTIME
- LOCALTIMESTAMP

CONTEXT FUNCTIONS

Session Context

- CURRENT_SESSION
- CURRENT_STATEMENT
- CURRENT_TRANSACTION
- CURRENT_USER
- LAST_QUERY_ID
- LAST_TRANSACTION

CONTEXT FUNCTIONS

Session Object Context

- CURRENT_DATABASE
- CURRENT_ROLE
- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- CURRENT_WAREHOUSE

EXAMPLE

- Display the current warehouse, database, and schema for the session:

```
select current_warehouse(), current_database(), current_schema();
```

- Display the current date, time, and timestamp (note that parentheses are not required to call these functions):

```
select current_date, current_time, current_timestamp;
```

NUMERIC FUNCTIONS

Rounding and Truncation

- ABS
- CEIL
- FLOOR
- MOD
- ROUND
- SIGN
- TRUNC(for date DATE_TRUNC) , TRUNCATE

NUMERIC FUNCTIONS

Exponent and Root

- CBRT
- EXP
- POW, POWER
- SQRT
- SQUARE

Logarithmic

- LN
- LOG

NUMERIC FUNCTIONS

Trigonometric

- ACOS
- ACOSH
- ASIN
- ASINH
- ATAN
- ATAN2
- ATANH
- COS
- COSH
- COT
- DEGREES
- HAVERSINE
- PI
- RADIANS
- SIN
- SINH
- TAN
- TANH

EXAMPLE

- select n, scale, round(n, scale) from tab2;
- select column1, square(column1) from (values (0), (1), (-2), (3.15), (null)) v;
- select sin(0), sin(pi()/3), sin(radians(90));

STRING AND BINARY FUNCTIONS

General

- ASCII
- BIT_LENGTH
- CHARINDEX
- CHR , CHAR
- CONCAT , ||
- CONTAINS
- EDITDISTANCE
- ENDSWITH
- ILIKE
- INITCAP
- INSERT
- LEFT
- LENGTH
- LIKE
- LOWER
- LPAD
- LTRIM
- OCTET_LENGTH
- PARSE_URL
- POSITION
- REPEAT
- REPLACE
- REVERSE
- RIGHT
- RPAD
- RTRIM
- RTRIMMED_LENGTH
- SPACE
- SPLIT
- SPLIT_PART
- STARTSWITH
- SUBSTR , SUBSTRING
- TRANSLATE
- TRIM
- UPPER
- UUID_STRING

STRING AND BINARY FUNCTIONS

Encode/Decode

- BASE64_DECODE_BINARY
- BASE64_DECODE_STRING
- BASE64_ENCODE
- HEX_DECODE_BINARY
- HEX_DECODE_STRING
- HEX_ENCODE
- TRY_BASE64_DECODE_BINARY
- TRY_BASE64_DECODE_STRING
- TRY_HEX_DECODE_BINARY
- TRY_HEX_DECODE_STRING

STRING AND BINARY FUNCTIONS

Hash/Cryptographic

- MD5 , MD5_HEX
- MD5_BINARY
- MD5_NUMBER
- SHA1 , SHA1_HEX
- SHA1_BINARY
- SHA2 , SHA2_HEX
- SHA2_BINARY

EXAMPLE

- select v, lower(v) from lu;
- select v, upper(v) from lu;
- select '123456', pos, len, substr('123456', pos, len) from o;
- select
  md5_binary('Snowflake') as md5_binary,
  base64_encode(md5_binary('Snowflake'),0,'$') as base64_encode,

base64_decode_binary(base64_encode(md5_binary('Snowflake'),0,'$'),'
$') as base64_decode_binary;

- select sha2_binary('Snowflake', 384);

REGULAR EXPRESSION (STRING) FUNCTIONS

- REGEXP
- REGEXP_COUNT
- REGEXP_INSTR
- REGEXP_LIKE
- REGEXP_REPLACE
- REGEXP_SUBSTR
- RLIKE

EXAMPLE

- select * from rlike_ex where name regexp 'J.h.* [dD]+o.*';
- select body, regexp_count(body, '\\b\\S*o\\S*\\b') from message;
- select regexp_substr('Customers - (NY)','\\([[:alnum:]\-]+\\)') as customers;

DATE AND TIME FUNCTIONS

Construction

- DATE_FROM_PARTS
- TIME_FROM_PARTS
- TIMESTAMP_FROM_PARTS

Extraction

- DATE_PART
- DAYNAME
- EXTRACT
- HOUR / MINUTE / SECOND
- LAST_DAY
- MONTHNAME
- YEAR / QUARTER / MONTH / WEEK / DAY / ...

DATE AND TIME FUNCTIONS

Addition/Subtraction

- ADD_MONTHS
- DATEADD
- DATEDIFF
- TIMEADD
- TIMEDIFF
- TIMESTAMPADD
- TIMESTAMPDIFF

Truncation

- DATE_TRUNC
- TRUNC

Conversion

- TO_DATE
- TO_TIME
- TO_TIMESTAMP / TO_TIMESTAMP_

Time Zone

- CONVERT_TIMEZONE

EXAMPLE

- select dayname(to_date('2015-05-01')) as day;
- select datediff(year, '2010-04-09 14:39:20'::timestamp, '2013-05-08 23:39:20'::timestamp) as diff_years;
- select to_time('12:34:56') from (values(1)) v;

SEMI-STRUCTURED DATA FUNCTIONS

JSON and XML Parsing

- CHECK_JSON
- CHECK_XML
- PARSE_JSON
- PARSE_XML
- STRIP_NULL_VALUE

Array Creation/Manipulation

- ARRAY_AGG
- ARRAY_APPEND
- ARRAY_CAT
- ARRAY_COMPACT
- ARRAY_CONSTRUCT
- ARRAY_CONSTRUCT_COMPACT
- ARRAY_INSERT
- ARRAY_PREPEND
- ARRAY_SIZE
- ARRAY_SLICE
- ARRAY_TO_STRING

SEMI-STRUCTURED DATA FUNCTIONS

Object Creation/Manipulation

- OBJECT_AGG
- OBJECT_CONSTRUCT
- OBJECT_DELETE
- OBJECT_INSERT

Extraction

- FLATTEN
- GET
- GET_PATH , :
- XMLGET

SEMI-STRUCTURED DATA FUNCTIONS

Conversion/Casting

- AS_type
- TO_ARRAY
- TO_JSON
- TO_OBJECT
- TO_VARIANT
- TO_XML

Type Predicates

- IS_type
- TYPEOF

EXAMPLE

- select *, check_xml(xml_str) from my_table where check_xml(xml_str) is not null;
- select array_size(v) from colors;
- select n, as_real(v), typeof(v) from vartab;
- select * from vartab where is_decimal(v);

CONVERSION FUNCTIONS

Any Data Type

- CAST , ::
- TRY_CAST

Text/Character/Binary Data Types

- TO_CHAR , TO_VARCHAR
- TO_BINARY
- TRY_TO_BINARY

Numeric Data Types

- TO_DECIMAL , TO_NUMBER , TO_NUMERIC
- TO_DOUBLE
- TRY_TO_DECIMAL, TRY_TO_NUMBER, TRY_TO_NUMERIC
- TRY_TO_DOUBLE

Boolean Data Type

- TO_BOOLEAN
- TRY_TO_BOOLEAN

Date and Time Data Types

- TO_DATE
- TO_TIME
- TO_TIMESTAMP / TO_TIMESTAMP_*
- TRY_TO_DATE
- TRY_TO_TIME
- TRY_TO_TIMESTAMP / TRY_TO_TIMESTAMP_*

Semi-structured Data Types

TO_ARRAY

- TO_OBJECT
- TO_VARIANT

EXAMPLE

- select to_date('2013-05-17') from (values(1)) v;
- select to_varchar('2013-04-05 01:02:03'::timestamp, 'mm/dd/yyyy, hh24:mi hours');

AGGREGATE FUNCTIONS

General Aggregation

- ANY_VALUE
- AVG
- CORR
- COUNT
- COVAR_POP
- COVAR_SAMP
- GROUPING
- GROUPING_ID
- HASH_AGG
- LISTAGG
- MEDIAN
- MIN / MAX
- PERCENTILE_CONT
- PERCENTILE_DISC
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE_POP
- VARIANCE , VARIANCE_SAMP

AGGREGATE FUNCTIONS

Bitwise Aggregation

- BITAND_AGG
- BITOR_AGG
- BITXOR_AGG

Linear Regression

- REGR_AVGX
- REGR_AVGY
- REGR_COUNT
- REGR_INTERCEPT
- REGR_R2
- REGR_SLOPE
- REGR_SXX
- REGR_SXY
- REGR_SYY

Semi-structured Data Aggregation

- ARRAY_AGG
- OBJECT_AGG

Cardinality Estimation (Using HyperLogLog)

- APPROX_COUNT_DISTINCT
- HLL
- HLL_ACCUMULATE
- HLL_COMBINE
- HLL_ESTIMATE
- HLL_EXPORT
- HLL_IMPORT

Similarity Estimation (Using MinHash)

- APPROXIMATE_JACCARD_INDEX
- APPROXIMATE_SIMILARITY
- MINHASH
- MINHASH_COMBINE

EXAMPLE

- select k, min(d), max(d) from minmax_example group by k;
- select array_agg(distinct o_orderstatus) from orders where o_totalprice > 450000;
- select hll(o_orderdate), hll_estimate(hll_import(hll_export(hll_accumulate(o_orderdate)))) from orders;
- select approximate_similarity(mh) from

```
 ((select minhash(100, c5) mh from orders where c2 <= 50000)
      union
   (select minhash(100, c5) mh from orders where c2 > 50000));
```

MISCELLANEOUS FUNCTIONS

System Functions

- SYSTEM$ABORT_SESSION
- SYSTEM$ABORT_TRANSACTION
- SYSTEM$CANCEL_ALL_QUERIES
- SYSTEM$CANCEL_QUERY
- SYSTEM$CLUSTERING_DEPTH
- SYSTEM$CLUSTERING_INFORMATION
- SYSTEM$CLUSTERING_RATIO
- SYSTEM$TYPEOF
- SYSTEM$WAIT

WINDOW FUNCTIONS

CUME_DIST	Finds the cumulative distribution of a value with regard to other values within the same window partition.
DENSE_RANK	Finds the rank of a value within a group of values, without gaps.
FIRST_VALUE	Returns the first value within an ordered group of values.
LAG	Returns the value of *expr* at negative row offset *offset*, i.e. the value of *expr* at *offset* rows before with respect to the current input row.
LAST_VALUE	
LEAD	
NTH_VALUE	Returns the nth value (up to 1000) within an ordered group of values.
NTILE	Divides an ordered data set equally into the number of buckets specified by constant_value. Buckets are sequentially numbered 1 through constant_value.
PERCENT_RANK	Returns the relative rank of a value within a group of values.
RANK	Returns the rank of a value within a group of values.
ROW_NUMBER	Returns a unique row number for each row within a window partition starting with 1 and increasing sequentially.
WIDTH_BUCKET	Constructs equi-width histograms, in which the histogram range is divided into intervals of identical size.

MISCELLANEOUS FUNCTIONS

Utility Functions

- GET_DDL
- HASH

Data Generation Functions

- RANDOM
- SEQ1 / SEQ2 / SEQ4 / SEQ8
- UUID_STRING

Random Distribution Functions

NORMAL

RANDSTR

UNIFORM

ZIPF

EXAMPLE

- select system$abort_session(1065153868222);
- select get_ddl('view', 'tpch_customer');
- select get_ddl('function', 'multiply(number, number)');
- select random(4711) from table(generator(rowcount => 5));
- select normal(0, 1, 1234) from table(generator(rowcount => 5));

WELCOME TO DAY 2

DAY 2 AGENDA

- Snowflake Administration
- End To End Usage Of Snowflake
- Feedback Form
- JSON Data Upload And Query – completed on Day 2
- ETL Access
- Tableau Desktop/Server connectivity
- Q & A

In this Topic:

- ☐ *Snowflake Password Policy*
- ☐ *Creating Users*
- ☐ *Resetting a User Password (Administrators)*
- ☐ *Disabling / Enabling a User*
- ☐ *Unlocking a User*
- ☐ *Altering Session Parameters for a User*
- ☐ *Modifying Other User Properties*
- ☐ *Viewing Users*
- ☐ *Dropping a User*

PASSWORD POLICY

Snowflake applies the following policy to passwords:

- Must be at least 8 characters long.
- Must contain at least 1 digit.
- Must contain at least 1 uppercase letter and 1 lowercase letter.

USER MANAGEMENT

- Snowflake enables administrators to create and manage user credentials and default values using SQL. In addition, the Snowflake web interface provides a convenient wizard for creating individual users, as well as the ability to complete common activities such as resetting a user’s password.

- Only users with the SECURITYADMIN role (i.e. security administrators) or higher can create, alter, or drop users.

CREATING USERS

- Using Web Interface

Click on Account > Users> Create Button

In the User Name field, enter a unique identifier for the user and in the Password and Confirm Password fields, enter the login password for the user.

Leave the Force Password Change checkbox checked to force the user to change their password on their next login; otherwise, clear the checkbox.

Click the Next button. The Advanced screen opens.

Optionally enter the Login Name, Display Name, and personal information for the user.

Click the Next button. The Preferences screen opens.

Optionally enter defaults for the user: Virtual warehouse, Namespace in the form of <db_name> or <db_name>.<schema_name>, Role

Click the Finish button. Snowflake displays a success message.

CREATING USERS

- Using SQL
- Syntax

```
CREATE [ OR REPLACE ] USER [ IF NOT EXISTS ] <name>
  [ <properties> ... ]
  [ <session_params> ... ]
  [ COMMENT = '<string_literal>' ]
```

- Example

```
create user user1 password='abc123' must_change_password = true;
```

RESETTING A USER PASSWORD (ADMINISTRATORS)

- Using the Web Interface

Click on Account > Users.

Click on a user row to select it, then click the Reset Password button. The Reset Password dialog opens.

Enter the new login password for the user, and confirm the password.

Leave the Force Password Change checkbox checked to force the user to change their password on their next login; otherwise, clear the checkbox.

Click the Finish button.

RESETTING A USER PASSWORD (ADMINISTRATORS)

- Using SQL
- Syntax

```
ALTER USER [ IF EXISTS ] <name> RENAME TO <new_name>
ALTER USER [ IF EXISTS ] <name> ABORT ALL QUERIES
ALTER USER [ IF EXISTS ] <name> SET { [ <property> [ <property> ... ] ]
                                       [ <session_param> [ <session_param> ... ]
                                      }
ALTER USER [ IF EXISTS ] <name> UNSET { [ <property_name> [, <property_name> ... ] ] ,
                                       [ <session_param_name> [, <session_param_name> ... ] ]
                                       }
```

- Example

```
alter user user1 set password = 'welcome1' must_change_password = true;
```

DISABLING/ENABLING A USER

- Using the Web Interface

Click on Account > Users.

Click on a user row to select it, then click the Disable User button. A confirmation dialog opens.

Click Yes to disable the user.

DISABLING/ENABLING A USER

- Using SQL
- Example
- Disable a user:

```
alter user jane.smith set disabled = true;
```

- Enable a user:

```
alter user jane.smith set disabled = false;
```

UNLOCKING A USER

- If a user login fails after five consecutive attempts, the user is locked out of their account for a period of time (currently 15 minutes). Once the period of time elapses, the system automatically clears the lock and the user can attempt to log in again.
- Using SQL
- Example

alter user jane.smith set mins_to_unlock= 0;

ALTERING SESSION PARAMETERS FOR A USER

- Using SQL
- To show the session parameters for a user, use the following SQL syntax:

```
SHOW PARAMETERS [ LIKE '<pattern>' ] FOR USER <name>
```

- To alter the session parameters for a user, use the following syntax:

```
ALTER USER <name> SET <session_param> = <value>
```

- For example, allow a user to remain connected to Snowflake indefinitely without timing out:

```
alter user `jane.smith` set client_session_keep_alive = true;
```

- To reset a session parameter for a user to the default value, use the following syntax:

```
ALTER USER <name> UNSET <session_param>
```

MODIFYING OTHER USER PROPERTIES

- Using SQL
- Example

```
alter user jane.smith set last_name = 'Jones';
```

VIEWING USERS

- Using SQL
- Syntax

```
DESC USER <name>
SHOW USERS [LIKE '<pattern>']
```

- Example

```
desc user janeksmith;
```

DROPPING A USER

- Using Web Interface

Click on **Account** > **Users**.

Click on a user row to select it, then click the **Drop** button. A confirmation dialog opens.

Click **Yes** to drop the user.

DROPPING A USER

- Using SQL
- Syntax

```
DROP USER [ IF EXISTS ] <name>
```

- Example

```
drop user user1;
```

SYSTEM USAGE AND BILLING

- The Snowflake cloud architecture separates data warehousing into two distinct functions:

  virtual warehouse processing and data storage.

- The costs associated with using Snowflake are based on your usage of the service for each of these functions.

In this Topic:

Virtual Warehouse Credit
Data Storage Usage
Managing Resource Monitors
Common Use Cases for Enforcing Credit Quotas

VIRTUAL WAREHOUSE CREDIT

- Warehouses come in different sizes: X-Small, Small, Medium, Large, X-Large, 2X-Large, and 4X-Large. The number of credits consumed by a warehouse per second while it is running is based directly on its size, but for the first time when it is launched credit for 60Sec is charged:

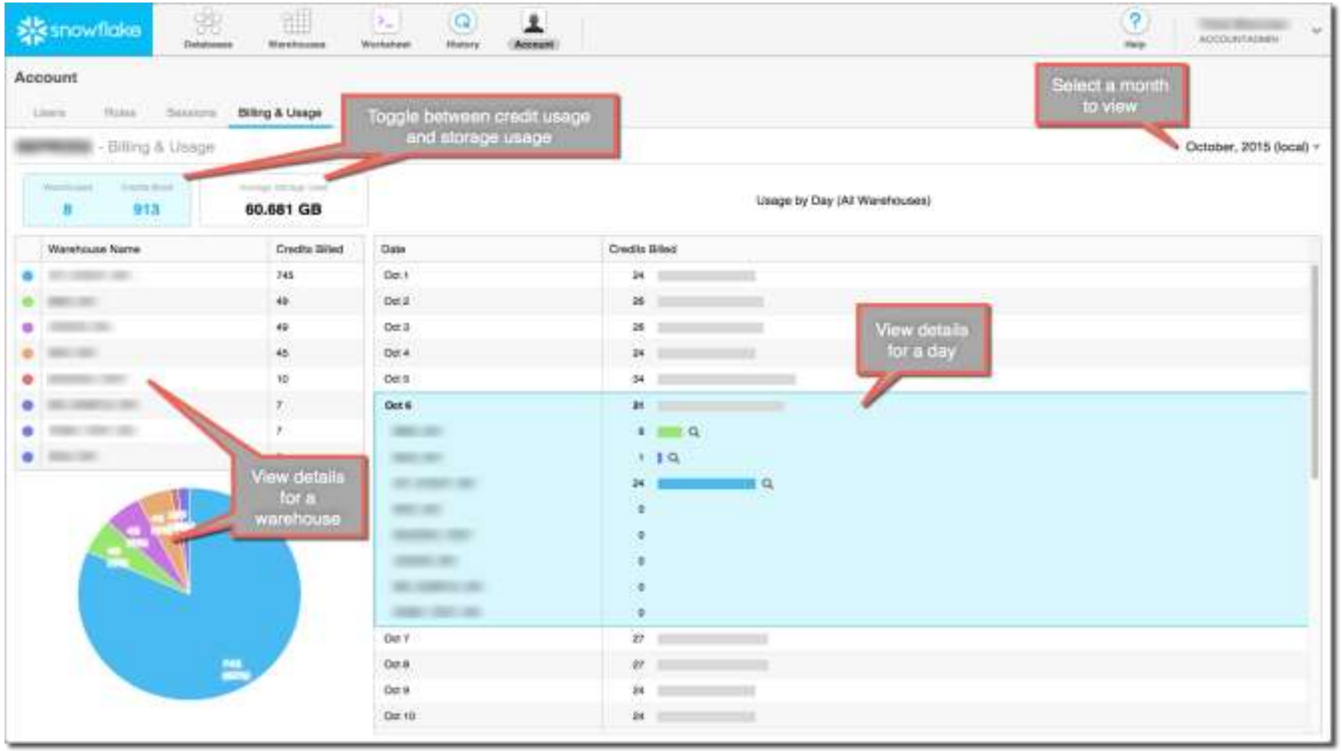
X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large
1	2	4	8	16	32	64	128

- Users with the ACCOUNTADMIN role can view warehouse usage and data storage for your account:

Web Interface: Click on Account > Billing & Usage

SQL: WAREHOUSE_METERING_HISTORY (in the Information Schema)

VIEWING CREDIT USAGE FOR WAREHOUSES



[redacted] - Billing & Usage

Warehouses	Credits Billed	Average Storage Used
8	913	60.681 GB

Warehouse Name	Credits Billed
● [redacted]	745
● [redacted]	49
● [redacted]	49
● [redacted]	45
● [redacted]	10
● [redacted]	7
● [redacted]	7
● [redacted]	[illegible]

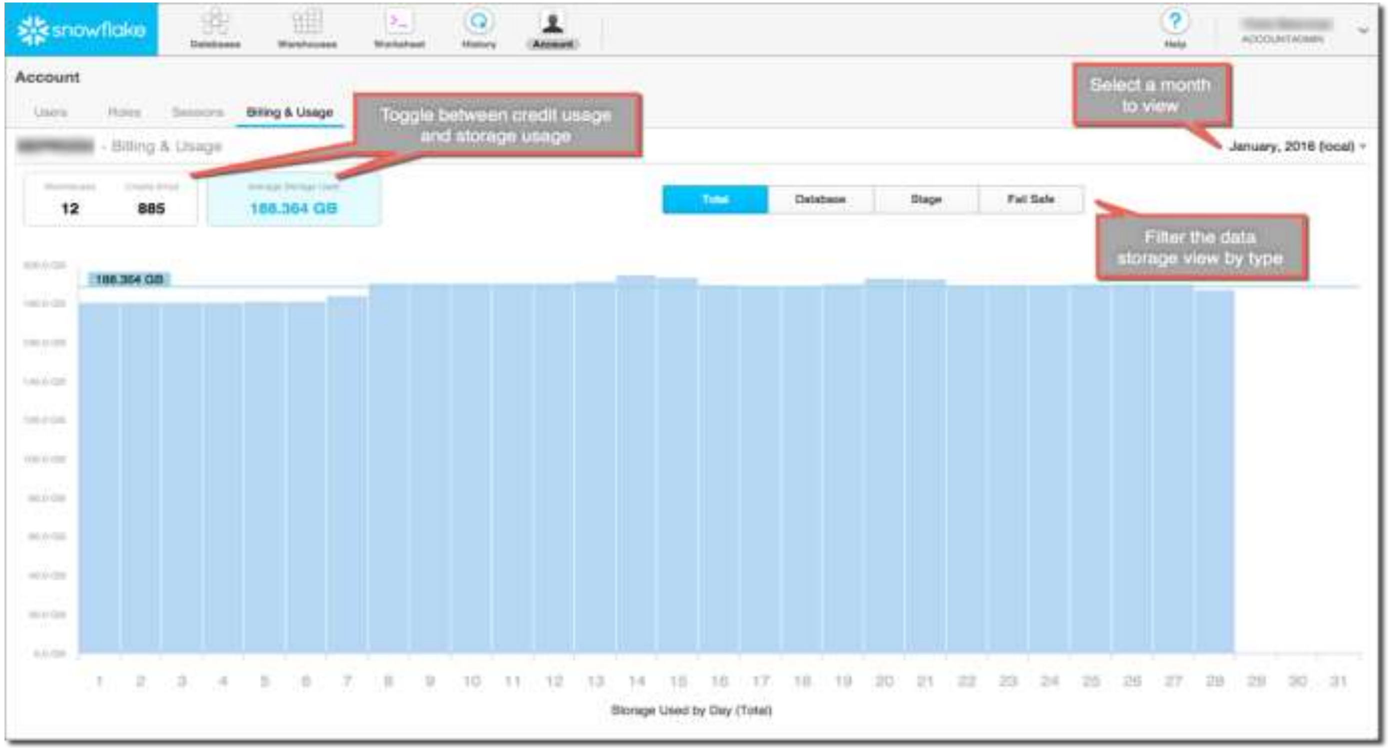
Usage by Day (All Warehouses)

Date	Credits Billed
Oct 1	24
Oct 2	26
Oct 3	26
Oct 4	24
Oct 5	34
Oct 6	31
[redacted]	6
[redacted]	1
[redacted]	24
[redacted]	0
[redacted]	0
[redacted]	0
[redacted]	0
[redacted]	0
Oct 7	27
Oct 8	27
Oct 9	24
Oct 10	24

DATA STORAGE USAGE

- The monthly costs for storing data in Snowflake is based on a flat rate per terabyte (TB). The amount charged per TB depends on your type of account (Capacity or On Demand) and region (US or EU). For data storage pricing, see the pricing page on the Snowflake website.
- Users with the ACCOUNTADMIN role can view warehouse credit usage for your account:
- Web Interface: Click on Account > Billing & Usage
- SQL: Information Schema functions: - DATABASE_STORAGE_USAGE_HISTORY - STAGE_STORAGE_USAGE_HISTORY
- Users with the appropriate access privileges can use either the web interface or SQL to view the size (in bytes) of individual tables in a schema/database:
- Web Interface: Click on Databases > db_name > Tables
- SQL: SHOW TABLES
- In addition, users with the ACCOUNTADMIN role can use SQL to view table size information:
- SQL: TABLE_STORAGE_METRICS View (in the Information Schema)

VIEWING DATA STORAGE USAGE



ENFORCING CREDIT QUOTAS

- Defining a Global Quota for Your Account

A global credit quota for an account can be implemented by ensuring that all warehouses are associated with a single resource monitor. To accomplish this, grant the USAGE privilege for the resource monitor to the predefined PUBLIC role.

- Creating a Quota for a Group of Users

To limit the credit usage for a group of users without restricting other users, you can use multiple roles.

ACCOUNT MANAGEMENT

Snowflake provides three types of parameters that can be set for your account:

- Account parameters that affect your entire account.
- Session parameters that default to users and their sessions.
- Object parameters that default to objects (warehouses, databases, schemas, and tables).

In this Topic:

Parameters

SECURITY

- Snowflake provides industry-leading features that ensure the highest levels of security for your account, users, and all the data you store in Snowflake:
- Network/site access

Network policies for controlling site access, i.e. IP whitelisting

- Account/user authentication

Multi-factor authentication (MFA)

Federated authentication and SSO

- Object security

Hybrid model for controlling access to any object in the account (users, warehouses, databases, tables, etc.):

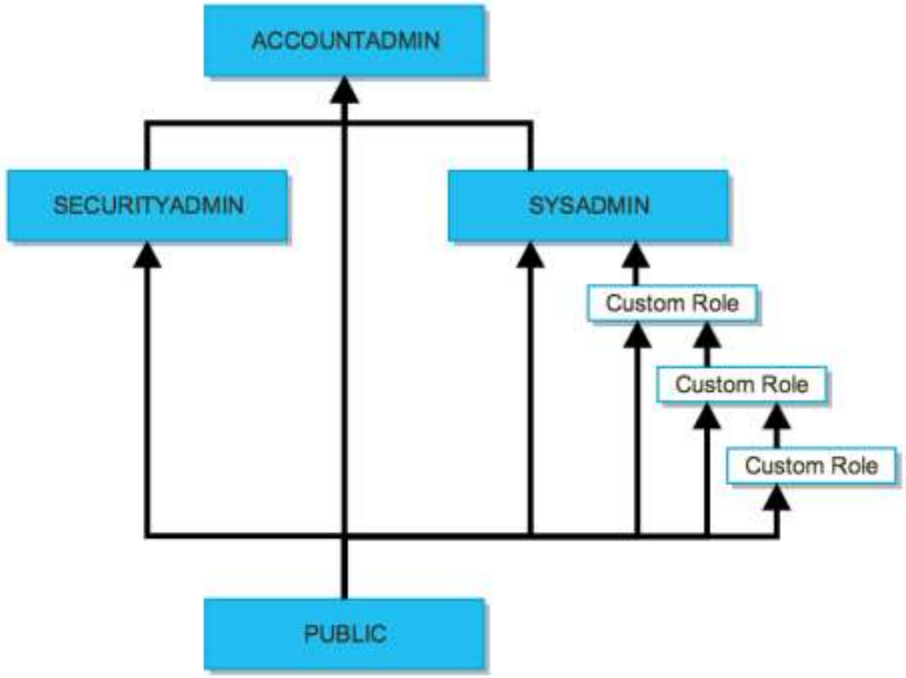
-DAC (discretionary access control)

-RBAC (role-based access control)

- Data security

Automatic data encryption using AES 256 strong encryption

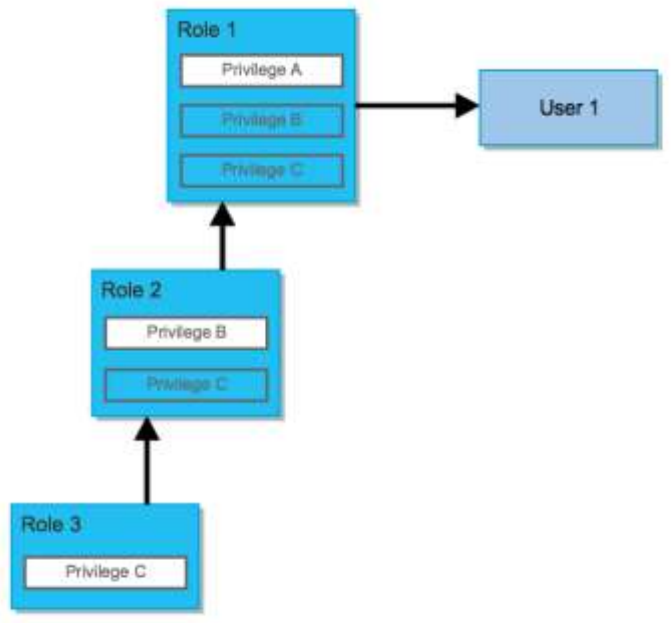
ROLES


ACCOUNTADMIN
SECURITYADMIN
SYSADMIN
Custom Role
Custom Role
Custom Role
PUBLIC

INHERITANCE OF PERMISSIONS

For a more specific example of role hierarchy and privilege inheritance, consider the following scenario:

- Role 3 has been granted to Role 2.
- Role 2 has been granted to Role 1.
- Role 1 has been granted to User 1.



- In this scenario:

- Role 2 inherits Privilege C
- Role 1 inherits Privileges B and C
- User 1 has all three privileges

CHANGE IN ROLE WITHIN A SESSION

During the course of a session, the user can use the USE ROLE command to change the current role. If the role is granted other roles, the user's session has privileges equal to the sum of the privileges of the current role and all the privileges of the roles granted to the current role within the role hierarchy.

When a user attempts to execute an action on an object, Snowflake compares the privileges available in the user's session against the privileges required on the object for that action. If the session has the required privileges on the object, the action is allowed.

INFORMATION SCHEMA

The Snowflake Information Schema consists of a set of system-defined views and functions that provide metadata information about the objects in your account:

The views describe database objects, as well as non-database objects, defined for your account, including:

- Views from the Information Schema SQL standard that are relevant to Snowflake (tables, columns, views, etc).
- Views for the non-standard database objects that Snowflake supports (e.g. stages and file formats).
- Views for other, non-database objects (e.g. roles).

The functions provide historical information, including:

- Query history.
- Data storage usage for databases and stages.
- Warehouse usage.

INFORMATION SCHEMA VIEWS

The views in INFORMATION_SCHEMA display metadata about all objects defined in the database, as well as Snowflake objects that are common across all databases, e.g. roles.
Some of the important views INFORMATION_SCHEMA contains are:

- APPLICABLE_ROLES View
- COLUMNS View
- DATABASES View
- FUNCTIONS View
- OBJECT_PRIVILEGES View
- REFERENTIAL_CONSTRAINTS View
- SCHEMATA View
- TABLE_PRIVILEGES View
- TABLES View
- VIEWS View

INFORMATION SCHEMA

The Snowflake Information Schema consists of a set of system-defined views and functions that provide metadata information about the objects in your account:

The views describe database objects, as well as non-database objects, defined for your account, including:

- Views from the Information Schema SQL standard that are relevant to Snowflake (tables, columns, views, etc).
- Views for the non-standard database objects that Snowflake supports (e.g. stages and file formats).
- Views for other, non-database objects (e.g. roles).

The functions provide historical information, including:

- Query history.
- Data storage usage for databases and stages.
- Warehouse usage.

OPEN ITEMS

- Given a batch of 15 SQL files what is the best method to insert a PAUSE between each?
- Can Snowflake support EXCLUDE list while cloning a database or schema? Some tables may not be needed in the clone.
- Snowflake document states that SEQ.CURRVAL is not supported. What is the workaround that should be suggested for entrenched Oracle users who prefer to maintain the current value across multiple SELECT statements?

END TO END USAGE OF SNOWFLAKE

PRE-REQ **Hands-On Loading Data and Unloading**

- □ Download data shared by trainer
- □ Create Multiple Warehouses For Upload And Analytics
- □ Create DB & table in Snowflake
- □ Load data to Table
- □ Connect to Tableau and Analyze

DOWNLOAD DATA

- The data is in csv format and you can download it in the link mentioned earlier.
- The name of the file is Crimes.csv
- The data is about the crimes that have happened across the Chicago, USA from 2016-2017.

LOGIN TO SNOWFLAKE

- Go to cmd and install SNOWSQL

snowsql;

- Login to your account

snowsql –a useready –u rajeshh;

- Load file to staging

put file://C:\SnowflakeStuffs\OtherFiles\Crimes.csv @raj_stage;

CONFIGURE SNOWFLAKE

- Go to https://useready.snowflakecomputing.com/console/login#/ and login using username and password
- Go to Snowflake Worksheet

Use Warehouse TRAINING_TEST_WAREHOUSE

Use Database TRNG_ANLYTICS_DB

Use ETL schema

CREATE TABLE IN SNOWFLAKE

- Create a table CRIME_SF

create or replace table CRIME_SF

(ID int not null, Case_Number varchar(50), Date Timestamp, Block varchar(255), IUCR varchar(50), Primary_Type varchar(255), Description varchar(255), Location_Description varchar(255),

Arrest varchar(50), Domestic varchar(50), Beat int, District int, Ward int, Community_Area int, FBI_Code varchar(50), X_Coordinate int, Y_Coordinate int, Year int, Updated_On timestamp, Latitude float, Longitude float, Location varchar(150));

LOAD DATA INTO SNOWFLAKE

```
CREATE FILE FORMAT "TRNG_ANLYTICS_DB"."ANALYTICS".csv TYPE = 'CSV'
COMPRESSION = 'AUTO' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n'
SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '\042' TRIM_SPACE =
TRUE ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE ESCAPE = 'NONE'
ESCAPE_UNENCLOSED_FIELD = '\134' DATE_FORMAT = 'AUTO'
TIMESTAMP_FORMAT = 'AUTO' NULL_IF = (‘’);
```

```
copy into CRIME_SF
  from @raj_stage/Crimes.csv  FILE_FORMAT = (FORMAT_NAME =
“csv”, compression=“gzip”);
```

LOAD DATA INTO SNOWFLAKE

```
copy into @raj_stage/Crimes_unload.csv
  from CRIME_SF
FILE_FORMAT = (FORMAT_NAME = “csv”, compression=“gzip”);

Get @raj_stage/Crimes_unload.csv file:///tmp;

list @raj_stage PATTERN='.*1.csv';

Remove @raj_stage pattern='.*.csv';
```

JSON DATA UPLOAD AND QUERY

- Create warehouse, database, schema, file format
- Create table to store raw JSON
- Load data to Snowflake
- Load data to Snowflake tables
- Sample query

CREATE TABLE TO STORE RAW JSON

```
CREATE TABLE people_data.people_json_src
 (
    src_json variant
 );
```

LOAD DATA TO SNOWFLAKE TABLES

```
create TABLE people_data.people
as
SELECT src_json:"_id"::varchar as id
    ,src_json:guid::varchar as guid ,src_json:isActive::varchar as isActive
    ,regexp_replace(src_json:balance,'\\$|\,',''):decimal(15,2) as balance ,src_json:picture::varchar as picture
    ,src_json:age::number as age ,src_json:eyeColor::varchar as eyeColor
    ,src_json:name.first::varchar as first_name ,src_json:name.last::varchar as last_name
    ,src_json:company::varchar as company ,src_json:email::varchar as email
    ,src_json:phone::varchar as phone ,src_json:address::varchar as address
    ,src_json:about::varchar as about ,src_json:registered::varchar as registered
    ,src_json:latitude::decimal(15,6) as latitude ,src_json:longitude::decimal(15,6) as longitude
    ,src_json:greeting::varchar as greeting ,src_json:favoriteFruit::varchar as favoriteFruit
FROM people_data.people_json_src
WHERE check_json(src_json) is null;
```

LOAD DATA TO SNOWFLAKE TABLES

```
CREATE TABLE people_data.tags
as
SELECT
        src_json:_id::varchar as id
        ,tags.value::varchar as tag
FROM
        people_data.people_json_src p
        ,TABLE(FLATTEN(p.src_json:tags)) tags;
```

LOAD DATA TO SNOWFLAKE TABLES

```
CREATE TABLE people_data.friends
as
SELECT
        src_json:_id::varchar as id
        ,friends.value:id::number as friend_id
        ,friends.value:name::varchar as friend_name
FROM
        people_data.people_json_src p
        ,TABLE(FLATTEN(p.src_json:friends)) friends;
```

SAMPLE QUERY

```
select
        *
from
        people_data.people
where
        id in (select id from people_data.friends where
friend_name='Parks Ray');
```

ETL ACCESS

- Use of ODBC –System DSN
- Sample workflow
- Parameters

PARAMETERS

- View

SHOW PARAMETERS [LIKE '<pattern>'] IN ACCOUNT

- Alter

ALTER ACCOUNT SET <param> = <value>

- Reset

ALTER ACCOUNT UNSET <param>

PARAMETERS

- Snowflake provides parameters that let you control the behaviour of your account, individual user sessions, and objects.

- All the parameters have default values, which can be set and then overridden at different levels depending on the parameter type (Account, Session, or Object).

In this Topic:

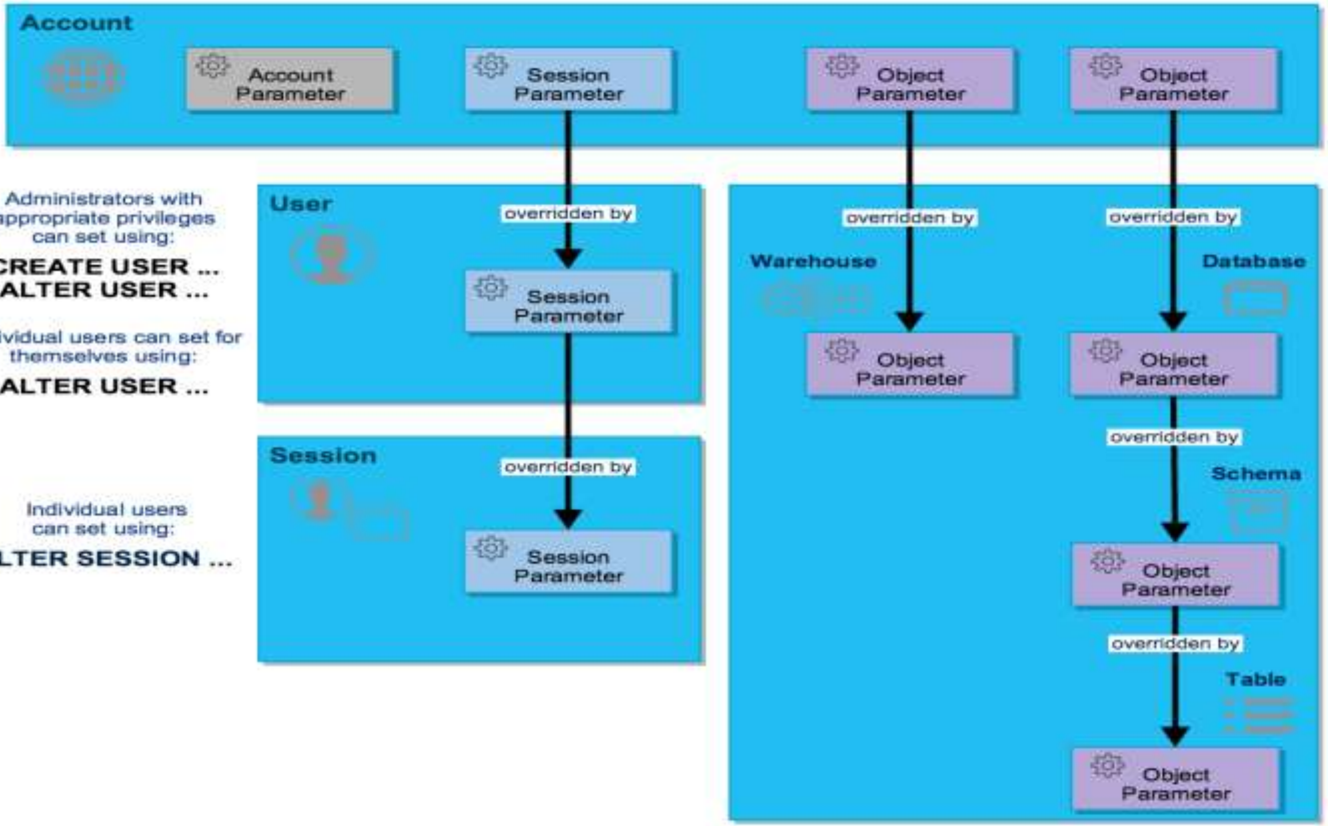
Parameter Types and Hierarchy

Account Parameters

Session Parameters

Object Parameters

PARAMETER TYPES AND HIERARCHY



Account administrators can set using:

ALTER ACCOUNT…

Administrators with appropriate privileges can set using:

CREATE USER …
ALTER USER …

Individual users can set for themselves using:

ALTER USER …

Individual users can set using:

ALTER SESSION …

Users with appropriate privileges can set using:

CREATE *object* …
ALTER *object* …

ACCOUNT PARAMETERS

- Account parameters can be set only at the account level by users with the ACCOUNTADMIN role (i.e. account administrators). Account parameters are set using the ALTER ACCOUNT command.

- Snowflake provides the following account parameters:

- CLIENT_ENCRYPTION_KEY_SIZE
- NETWORK_POLICY
- PERIODIC_DATA_REKEYING
- SAML_IDENTITY_PROVIDER
- SSO_LOGIN_PAGE

ACCOUNT PARAMETERS

- **CLIENT_ENCRYPTION_KEY_SIZE:**

Type:	Account — Can be set only for Account
Data Type:	Integer
Description:	Specifies the AES encryption key size, in bits (128 or 256), used to encrypt/decrypt files stored in internal staging locations (for loading/unloading data). Note that, if set to 256 (i.e. strong encryption), additional JCE policy files must be installed on each client machine from which data is loaded/unloaded. For more information about installing the required files, see Java Requirements for the JDBC Driver.
Default Value:	128

ACCOUNT PARAMETERS

- **NETWORK_POLICY:**

Type:	Account — Can be set only for Account
Data Type:	String
Description:	Supports restricting access to your account based on user IP address. References a network policy created using CREATE NETWORK POLICY. Note that NETWORK_POLICY is the only account parameter that can be set by both account administrators and security administrators. For more information about network policies, see Managing Network Policies.
Default Value:	None

ACCOUNT PARAMETERS

- **PERIODIC_DATA_REKEYING:**

Type:	Account — Can be set only for Account
Data Type:	Boolean
Description:	Enables/disables re-encryption of table data with new keys on a yearly basis to provide additional data protection: •TRUE specifies that data is rekeyed after one year has passed since the data was last encrypted. Rekeying occurs in the background so no down-time is experienced and the affected data/table is always available. •FALSE specifies that data is not rekeyed. You can enable and disable rekeying at any time. Enabling/disabling rekeying does not result in gaps in your encrypted data: •If rekeying is enabled for a period of time and then disabled, all data already tagged for rekeying is rekeyed, but no further data is rekeyed until you renable it again. •If rekeying is re-enabled, Snowflake automatically rekeys all data that has keys which meet the criteria (i.e. key is older than one year).
Default Value:	FALSE

ACCOUNT PARAMETERS

- **SAML_IDENTITY_PROVIDER:**

Type:	Account — Can be set only for Account
Data Type:	JSON
Description:	Enables federated authentication. The parameter parameter accepts a JSON object, enclosed in single quotes, with the following fields: `{ "certificate": "", "ssoUrl": "", "type" : "", "label" : "" }` Where: **certificate:**Specifies the certificate (generated by the IdP) that verifies communication between the IdP and Snowflake. **ssoUrl:**Specifies the URL endpoint (provided by the IdP) where Snowflake sends the SAML requests. **type:**Specifies the type of IdP used for federated authentication ("OKTA" , "ADFS" , "Custom"). **label:**Specifies the button text for the IdP in the Snowflake login page. The default label is Single Sign On. If you change the default label, the label you specify can only contain alphanumeric characters (i.e. special characters and blank spaces are not currently supported). Note that, if the "type" field is "Okta", a value for the label field does not need to be specified because Snowflake displays the Okta logo in the button.
Default Value:	None

ACCOUNT PARAMETERS

- **SSO_LOGIN_PAGE**

Type:	Account — Can be set only for Account
Data Type:	Boolean
Description:	Disables preview mode for testing SSO (after enabling federated authentication) before rolling it out to users: •If TRUE, preview mode is disabled and users will see the button for Snowflake-initiated SSO for your identity provider (as specified in SAML_IDENTITY_PROVIDER) in the Snowflake main login page. •If FALSE, preview mode is enabled and SSO can be tested using the following URL: •https://<account_name>.snowflakecomputing.com/console/login?fedpreview=true For more information, see Configuring and Using Federated Authentication in Snowflake.
Default Value:	FALSE

SESSION PARAMETERS

Most parameters are session parameters, which can be set at the following levels:
Account ▸ User ▸ Session

Account
Account administrators can use the ALTER ACCOUNT command to set session parameters for the account. The values set for the account default to individual users and their sessions.

User
Administrators with the appropriate privileges can use the ALTER USER command to override session parameters for individual users. The values set for a user default to any sessions started by the user. In addition, users can override sessions parameters for themselves using ALTER USER.

Session
Users can use the ALTER SESSION to override session parameters within their sessions.

OBJECT PARAMETERS

Object parameters can be set at the following levels:
Account ▸ Object

Account
Account administrators can use the ALTER ACCOUNT command to set object parameters for the account. The values set for the account default to the objects created in the account.

Object
Users with the appropriate privileges can use the corresponding CREATE or ALTER commands to override object parameters for an individual object.

Currently, Snowflake provides the following object parameters:

- DATA_RETENTION_TIME_IN_DAYS (for databases, schemas, and tables)
- MAX_CONCURRENCY_LEVEL (for warehouses)
- STATEMENT_QUEUED_TIMEOUT_IN_SECONDS (for warehouses); session parameter also
- STATEMENT_TIMEOUT_IN_SECONDS (for warehouses); session parameter also

OBJECT PARAMETERS

- **DATA_RETENTION_TIME_IN_DAYS :**

Type:	Object (for databases, schemas, and tables) Can be set for Account ▸ Database ▸ Schema ▸ Table
Data Type:	Number
Description:	Number of days for which Snowflake retains historical data for performing Time Travel actions (SELECT, CLONE, UNDROP) on the object: •For Standard Edition accounts, this parameter can be set to either 1 or 0. •For Enterprise Edition accounts, this parameter can be changed to any value from 0 up to 90 days. The value specified defaults to databases, schemas, and permanent tables, and can be overridden for each. A value of 0 for a database, schema, or table effectively disables Time Travel for the specified object. For more information, see Understanding & Using Time Travel.
Default Value:	1

OBJECT PARAMETERS

- **MAX_CONCURRENCY_LEVEL**

Type:	Object (for warehouses) — Can be set for Account ▸ Warehouse
Data Type:	Number
Description:	Specifies the maximum number of SQL statements (queries, DDL, DML, etc.) a warehouse cluster can execute concurrently. When the max level is reached: •For a single-cluster warehouse or a multi-cluster warehouse (in Maximized mode), additional statements are queued until resources are available. •For a multi-cluster warehouse (in Auto-scale mode), additional clusters are started. •With large, complex statements, fewer statements than the max level may execute concurrently. As each statement is submitted to a warehouse, Snowflake ensures that resources are not over-allocated; if there aren't enough resources available to execute a statement, it is automatically queued.
Default Value:	8

OBJECT PARAMETERS

STATEMENT_QUEUED_TIMEOUT_IN_SECONDS:

Type:	Session ***and*** Object (for warehouses) Can be set for for Account ▸ User ▸ Session; can also be set for individual warehouses
Data Type:	Number
Description:	Maximum amount of time, in seconds, a SQL statement (query, DDL, DML, etc.) can be queued on a warehouse before it is canceled by the system. A value of 0 specifies that no timeout is enforced. This parameter can be used in conjunction with the MAX_CONCURRENCY_LEVEL parameter to ensure a warehouse is never backlogged. The parameter can be set within the session hierarchy. It can also be set for a warehouse to control the queue timeout for all SQL statements processed by the warehouse. When the parameter is set for both a warehouse and a session, the lowest non-zero value is enforced.
Default Value:	0 (i.e. no timeout)

OBJECT PARAMETERS

- **STATEMENT_TIMEOUT_IN_SECONDS:**

Type:	Session ***and*** Object (for warehouses) Can be set for for Account ▸ User ▸ Session; can also be set for individual warehouses
Data Type:	Number
Description:	Time, in seconds, after which a running SQL statement (query, DDL, DML, etc.) is cancelled by the system. A value of 0 specifies that no timeout is enforced. The parameter can be set within the session hierarchy. It can also be set for a warehouse to control the runtime for all SQL statements processed by the warehouse. When the parameter is set for both a warehouse and a session, the lowest non-zero value is enforced. For example, a warehouse has a timeout of 1000 seconds and the current user session has a timeout of 500 seconds. Any statement that executes for longer than 500 seconds in the session is cancelled.
Default Value:	0 (i.e. no timeout)

TRAINING FEEDBACK

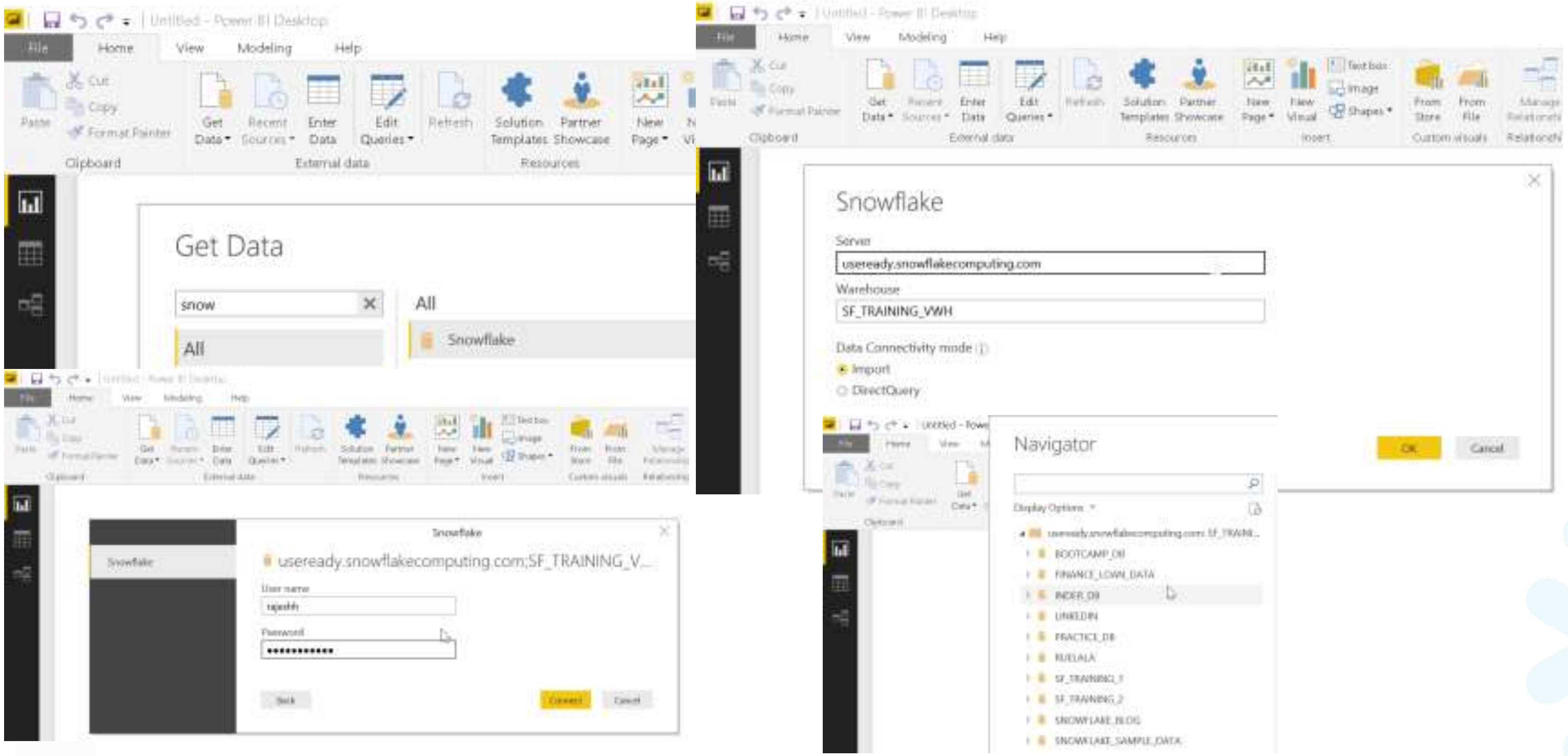
- Feedback URL:
- https://docs.google.com/forms/d/e/1FAIpQLSeJLPDihApnTep3XTvtmGNg4Bvcg4iCpq2Eja8aPp3rcV4YQw/viewform

- Required to improve training quality

BUSINESS QUESTIONS

- What are the total no of cases?
- What are the total no of crimes?
- Percentage of arrested and not arrested
- Where have most crimes happened?
- What type of crime has been committed the most?
- At what time of the day do most of the crimes happen?

CONNECTING FROM POWER BI



Get Data

snow

All

All

Snowflake

Snowflake

Server
useready.snowflakecomputing.com

Warehouse
SF_TRAINING_VWH

Data Connectivity mode
[✓ Import]
DirectQuery

Snowflake

useready.snowflakecomputing.com;SF_TRAINING_V...

User name
rajeshh

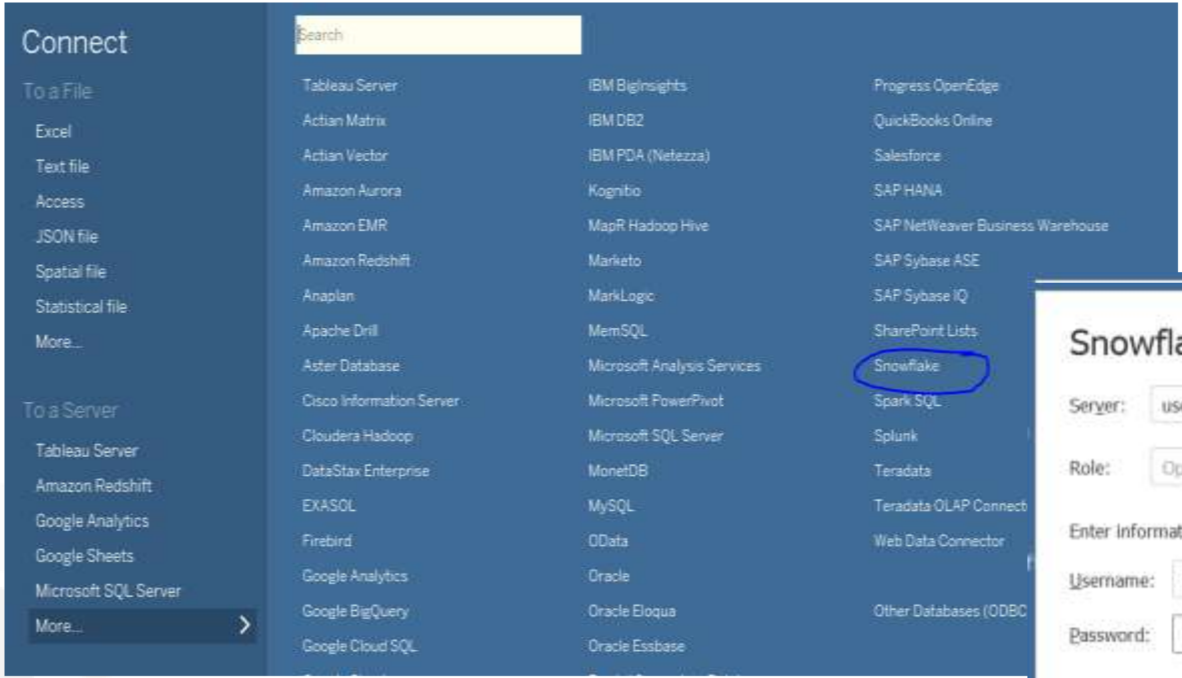
Password
••••••••••••

Navigator

Display Options

- useready.snowflakecomputing.com: SF_TRAINI...
  - BOOTCAMP_DB
  - FINANCE_LOAN_DATA
  - INDER_DB
  - LINKEDIN
  - PRACTICE_DB
  - RUELALA
  - SF_TRAINING_1
  - SF_TRAINING_2
  - SNOWFLAKE_BLOG
  - SNOWFLAKE_SAMPLE_DATA

CONNECTING FROM TABLEAU



Connect

To a File

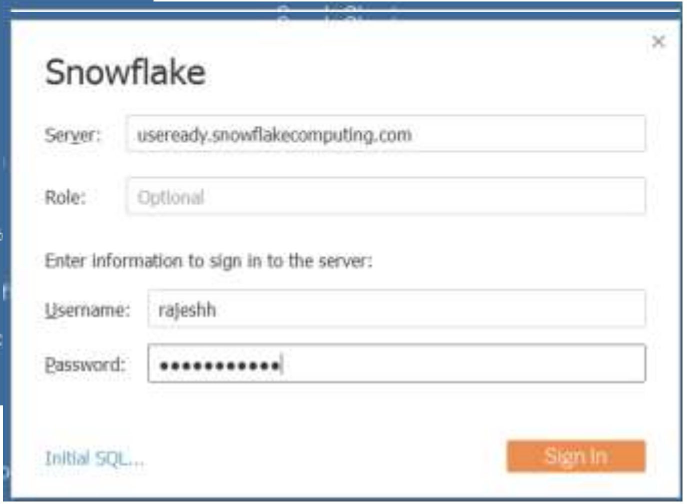
- Excel
- Text file
- Access
- JSON file
- Spatial file
- Statistical file
- More...

To a Server

- Tableau Server
- Amazon Redshift
- Google Analytics
- Google Sheets
- Microsoft SQL Server
- More...

Search

Tableau Server	IBM BigInsights	Progress OpenEdge
Actian Matrix	IBM DB2	QuickBooks Online
Actian Vector	IBM PDA (Netezza)	Salesforce
Amazon Aurora	Kognitio	SAP HANA
Amazon EMR	MapR Hadoop Hive	SAP NetWeaver Business Warehouse
Amazon Redshift	Marketo	SAP Sybase ASE
Anaplan	MarkLogic	SAP Sybase IQ
Apache Drill	MemSQL	SharePoint Lists
Aster Database	Microsoft Analysis Services	Snowflake
Cisco Information Server	Microsoft PowerPivot	Spark SQL
Cloudera Hadoop	Microsoft SQL Server	Splunk
DataStax Enterprise	MonetDB	Teradata
EXASOL	MySQL	Teradata OLAP Connect
Firebird	OData	Web Data Connector
Google Analytics	Oracle	
Google BigQuery	Oracle Eloqua	Other Databases (ODBC
Google Cloud SQL	Oracle Essbase	



Snowflake

Server: useready.snowflakecomputing.com

Role: Optional

Enter information to sign in to the server:

Username: rajeshh

Password: ●●●●●●●●●●●

Initial SQL...

Sign In

USEReady

www.useready.com
contact@useready.com

+1-212-370-0000

805 Third Avenue
11th floor, New York, NY 10022

snowflake

www.snowflake.net

+1-844-766-9355

101 South Ellsworth Avenue,
San Mateo, CA, United States