HANDS-ON LAB GUIDE FOR IN PERSON ZERO-TO-SNOWFLAKE

To be used with the Snowflake free 30-day trial at:
https://trial.snowflake.com

- Register for the trial 24+ hours prior to lab in case of activation delays
- Works for any Snowflake edition or cloud provider
- Approximate duration: 90 minutes. Approximately 7 credits used.

Table of Contents

Lab Overview

This entry-level lab introduces you to the user interface and basic capabilities of Snowflake, and is designed specifically for use with the Snowflake, free 30-day trial at https://trial.snowflake.com. When done with the lab you should be ready to load your own data into Snowflake and learn its more advanced capabilities.

Target Audience

Database and Data Warehouse Administrators and Architects

What you'll learn

The exercises in this lab will walk you through the steps to:

- Create stages, databases, tables, views, and warehouses
- Load structured and semi-structured data
- Query data including joins between tables
- Clone objects
- Undo user errors
- Create roles and users, and grant them privileges
- Securely and easily share data with other accounts

Prerequisites

- Use of the Snowflake free 30-day trial environment
- Basic knowledge of SQL, and database concepts and objects
- Familiarity with CSV comma-delimited files and JSON semi-structured data

Module 1: Prepare Your Lab Environment

1.1 Steps to Prepare Your Lab Environment

1.1.1 If not yet done, register for a Snowflake free 30-day trial at https://trial.snowflake.com

- The Snowflake edition (Standard, Enterprise, e.g.), cloud provider (AWS, Azure, e.g.), and Region (US East, EU, e.g.) do *not* matter for this lab. But we suggest you select the region which is physically closest to you. And select the Enterprise edition so you can leverage some advanced capabilities that are not available in lower Editions.

- After registering, you will receive an email with an activation link and your Snowflake account URL. Bookmark this URL for easy, future access. After activation, you will create a user name and password. Write down these credentials.

1.1.2 Resize your browser windows so you can view this lab guide PDF and your web browser side-by-side to more easily follow the lab instructions. If possible, even better is to use a secondary display dedicated to the lab guide.

1.1.3 Click on https://s3.amazonaws.com/snowflake-workshop-lab/lab_scripts_InpersonZTS.sql and download the “lab_scripts_InpersonZTS.sql” file to your local machine. This file contains pre-written SQL commands and we will use this file later in the lab.

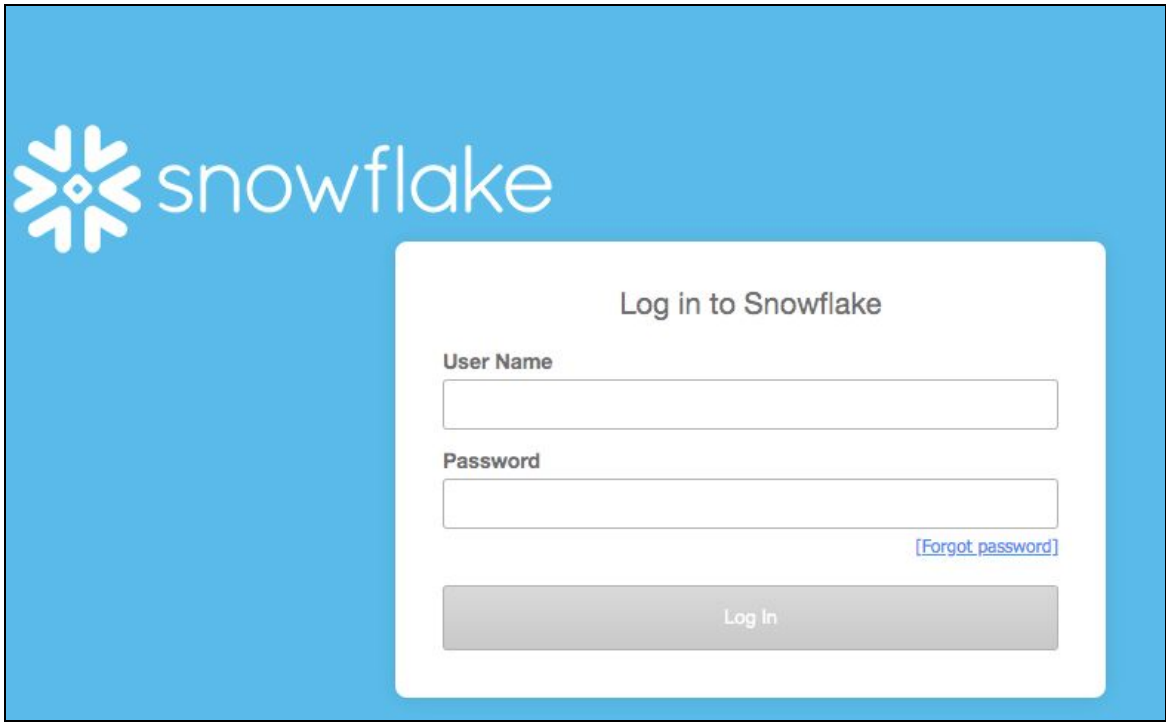
Module 2: The Snowflake User Interface & Lab "Story"

About the screen captures, sample code, and environment
Screen captures in this lab depict examples and results that may slightly vary from what you may see when you complete the exercises.

2.1 Logging Into the Snowflake User Interface (UI)

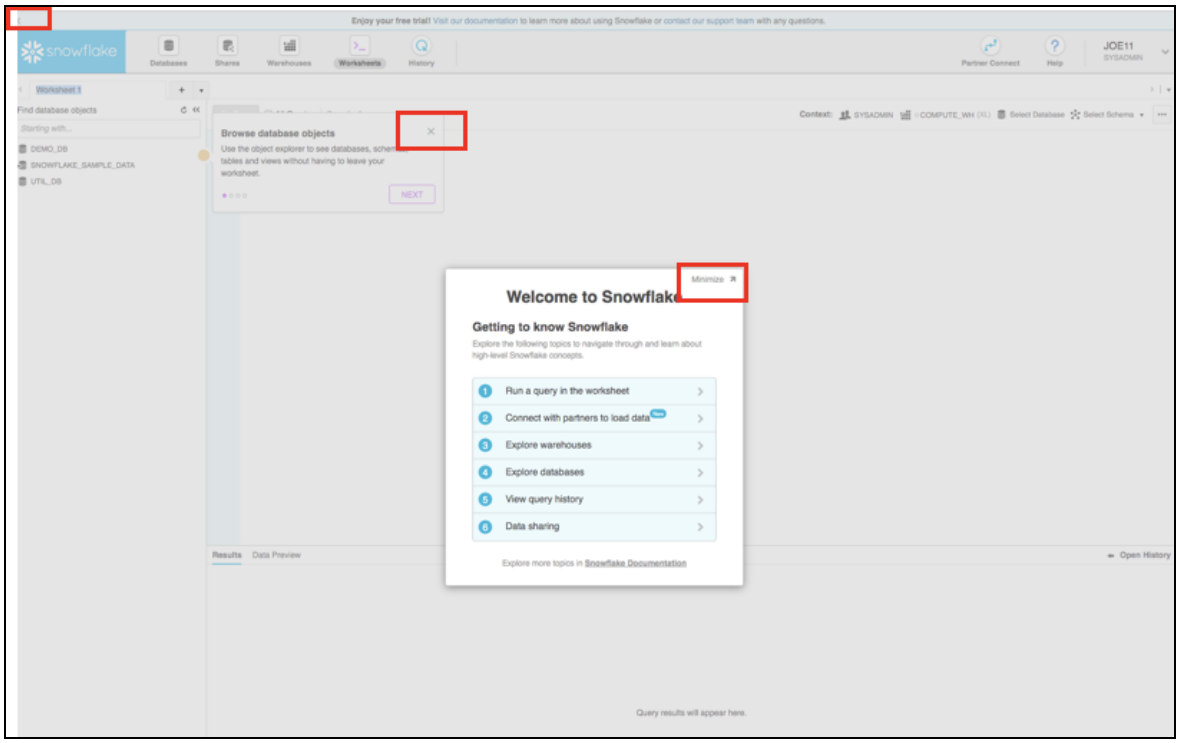
2.1.1 Open a browser window and enter the URL of your the Snowflake 30-day trial environment.

2.1.2 You should see the login screen below. Enter your unique credentials to log in.



2.2 Close any Welcome Boxes and Tutorials

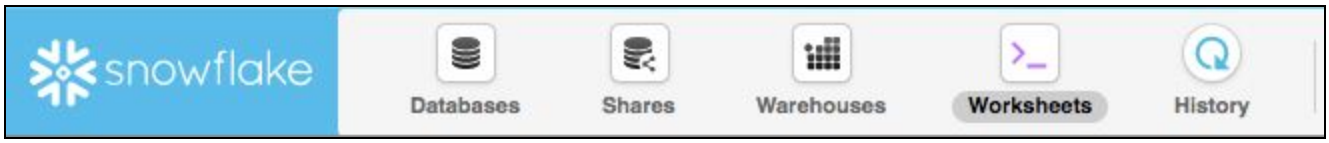
2.2.1 You may see "welcome" and "helper" boxes in the UI when you log in for the first time. Also a "Enjoy your free trial…" ribbon at the top of the UI. Minimize and close them by clicking on the items in the red boxes on screenshot below.



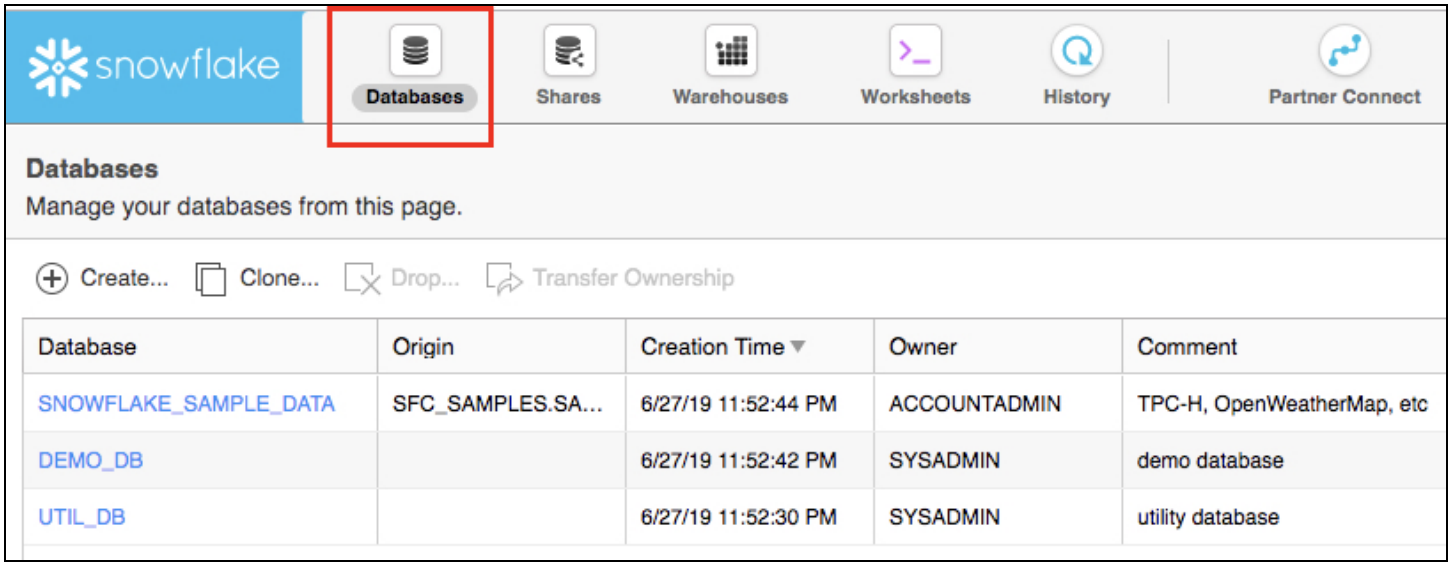
2.3 Navigating the Snowflake UI

First let's get you acquainted with Snowflake! This section covers the basic components of the user interface to help you orient yourself. We will move left to right in the top of the UI.

2.3.1 The top menu allows you to switch between the different areas of Snowflake:

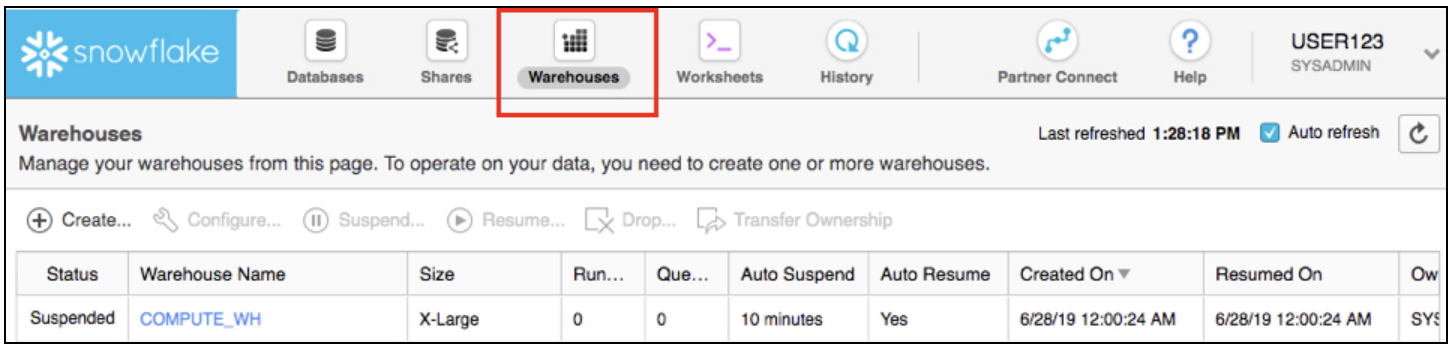


2.3.2 The **Databases** tab shows information about the databases you have created or have privileges to access. You can create, clone, drop, or transfer ownership of databases as well as load data (limited) in the UI. Notice several databases already exist in your environment. However, we will not be using these in this lab.



2.3.3 The **Shares** tab is where data sharing can be configured to easily and securely share Snowflake table(s) among separate Snowflake accounts or external users, without having to create a second copy of the table data. At the end of this lab is a module on data sharing.

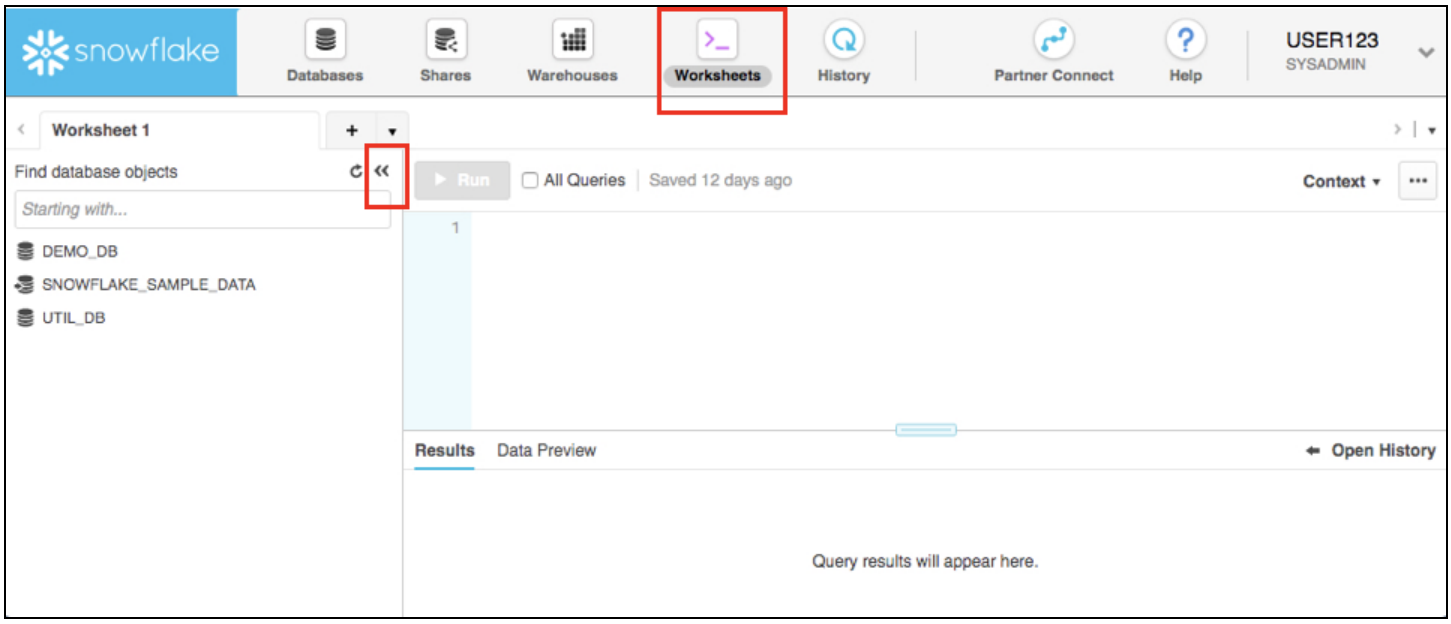
2.3.4 The **Warehouses** tab is where you set up and manage compute resources (virtual warehouses) to load or query data in Snowflake. Note a warehouse called "COMPUTE_WH (XL)" already exists in your environment.



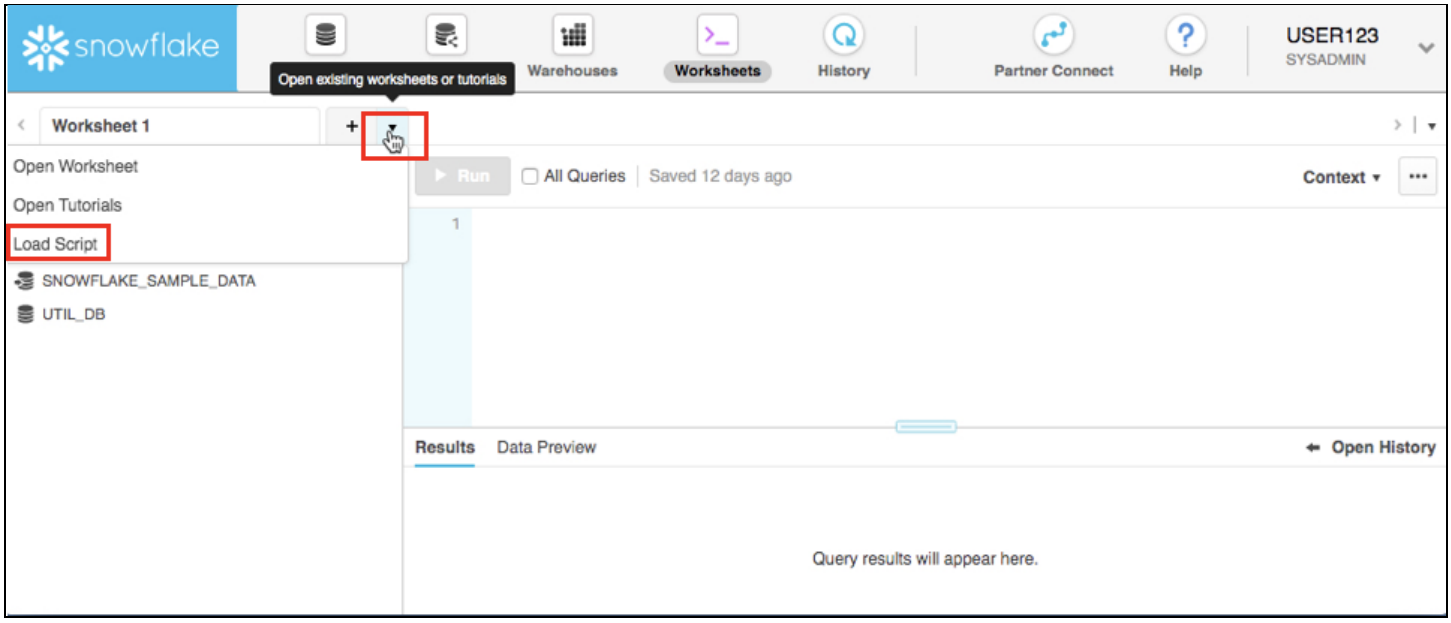
2.3.5 The **Worksheets** tab provides an interface for submitting SQL queries, performing DDL and DML operations and viewing results as your queries/operations complete. The default “Worksheet 1” appears.

In the left pane is the database objects browser which enables users to explore all databases, schemas, tables, and views accessible by the role selected for a worksheet. The bottom pane shows results of queries and operations.

The various windows on this page can be resized by moving the small sliders on them. And if during the lab you need more room to work in the worksheet, collapse the database objects browser in the left pane. Many of the screenshots in this guide will have this database objects browser closed.



2.3.6 At the top left of the default “Worksheet 1,” just to the right of the worksheet tab, click on the small, downward facing arrow, select “Load Script”, then browse to the “lab_scripts.sql” file you downloaded in the prior module and select “Open”. All of the SQL commands you need to run for the remainder of this lab will now appear on the new worksheet. Do not run any of the SQL commands yet. We will come back to them later in the lab and execute them one at a time.

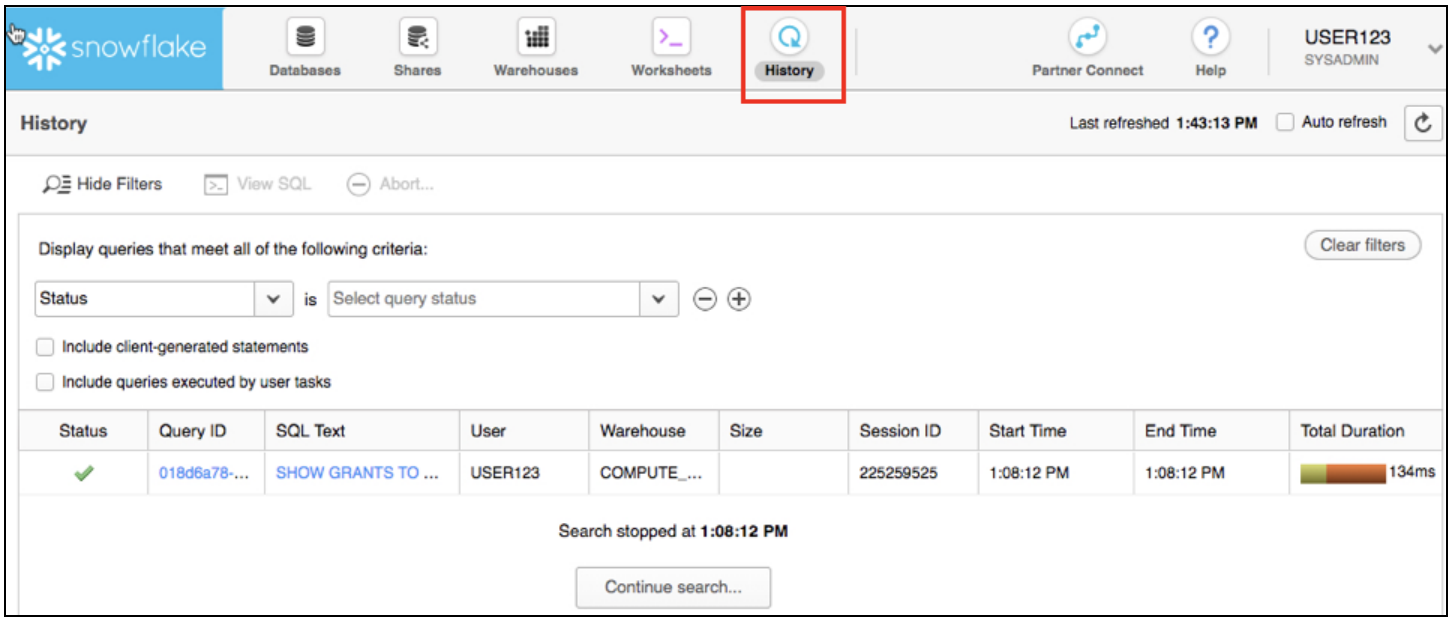


Warning - Do Not Copy/Paste SQL From This PDF to a Worksheet
Copy-pasting the SQL code from this PDF into a Snowflake worksheet will result in formatting errors and the SQL will not run correctly. Make sure to use the “Load Script” method just covered.

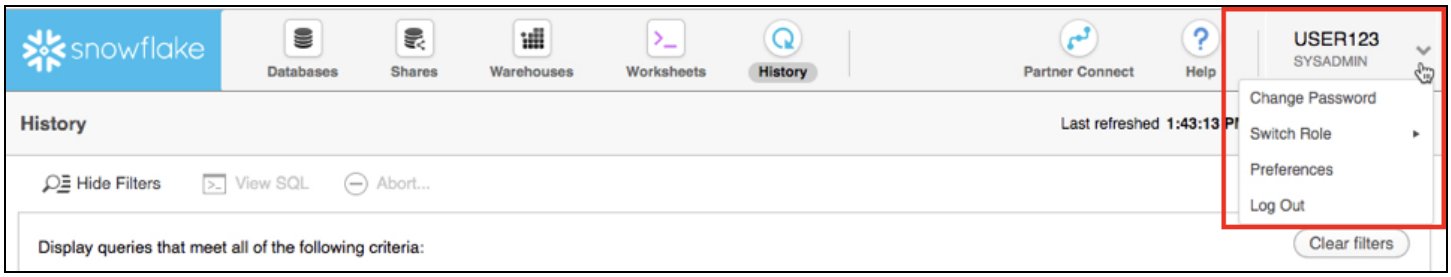
On older or locked-down browsers, this “load script” step may not work as the browser will prevent you from opening the .sql file. If this is the case, open the .sql file with a text editor and then copy/paste all the text from the .sql file to the “Worksheet 1”

Worksheets vs the UI
Much of the configurations in this lab will be executed via this pre-written SQL in the Worksheet in order to save time. These configurations could also be done via the UI in a less technical manner but would take more time.

2.3.7 The **History** tab allows you to view the details of all queries executed in the last 14 days in the Snowflake account (click on a Query ID to drill into the query for more detail).



2.3.8 If you click on the top right of the UI where your user name appears, you will see that here you can do things like change your password, roles, or preferences. Snowflake has several system defined roles. You are currently in the default role of SYSADMIN and we will stay in this role until towards the end of this lab.



SYSADMIN
For most this lab you will remain in the SYSADMIN (aka System Administrator) role which has privileges to create warehouses and databases and other objects in an account.

In a real-world environment, you would use different roles for the tasks in this lab, and assign the roles to your users. More on access control in Snowflake is in towards the end of this lab and also at https://docs.snowflake.net/manuals/user-guide/security-access-control.html

2.4 The Lab story

2.4.1 This Snowflake lab will be done as part of a theoretical real-world "story" to help you better understand why we are performing the steps in this lab and in the order they appear.

The “story” of this lab is based on the analytics team at Citi Bike, a real, citywide bike share system in New York City, USA. This team wants to be able to run analytics on data to better understand their riders and how to serve them best.

We will first load structured .csv data from rider transactions into Snowflake. This comes from Citi Bike internal transactional systems. Then later we will load open-source, semi-structured JSON weather data into Snowflake to see if there is any correlation between the number of bike rides and weather.

Module 3: Preparing to Load Data

Let's start by preparing to load the structured data on Citi Bike rider transactions into Snowflake.

This module will walk you through the steps to:

- Create a database and table
- Create an external stage
- Create a file format for the data

Getting Data into Snowflake
There are many ways to get data into Snowflake from many locations including the COPY command, Snowpipe auto-ingestion, an external connector, or a third-party ETL/ELT product. More information on getting data into Snowflake, see https://docs.snowflake.net/manuals/user-guide-data-load.html

We are using the COPY command and S3 storage for this module in a manual process so you can see and learn from the steps involved. In the real-world, a customer would likely use an automated process or ETL product to make the data loading process fully automated and much easier.

The data we will be using is bike share data provided by Citi Bike NYC. The data has been exported and pre-staged for you in an Amazon AWS S3 bucket in the US-EAST region. The data consists of information about trip times, locations, user type, gender, age of riders, etc. On AWS S3, the data represents 61.5M rows, 377 objects, and 1.9GB total size compressed.

Below is a snippet from one of the Citi Bike CSV data files:

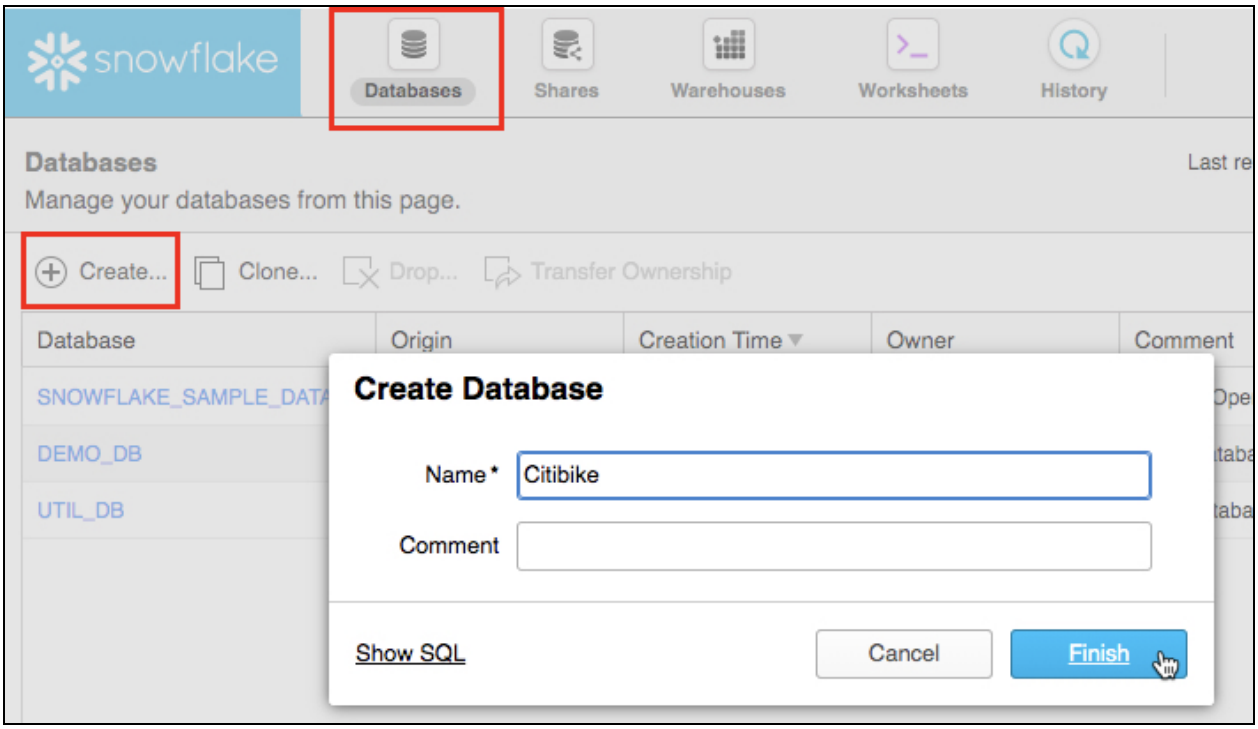
```
"tripduration","starttime","stoptime","start station id","start station name","start
station latitude","start station longitude","end station id","end station name","end
station latitude","end station
longitude","bikeid","name_localizedValue0","usertype","birth year","gender"
196,"2018-01-01 00:01:51","2018-01-01 00:05:07",315,"South St & Gouverneur Ln",
40.70355377,-74.00670227,259,"South St & Whitehall St",
40.70122128,-74.01234218,18534,"Annual Membership","Subscriber",1997,1
207,"2018-01-01 00:02:44","2018-01-01 00:06:11",3224,"W 13 St & Hudson St",
40.73997354103409,-74.00513872504234,470,"W 20 St & 8 Ave",
40.74345335,-74.00004031,19651,"Annual Membership","Subscriber",1978,1
613,"2018-01-01 00:03:15","2018-01-01 00:13:28",386,"Centre St & Worth St",
40.71494807,-74.00234482,2008,"Little West St & 1 Pl",
40.70569254,-74.01677685,21678,"Annual Membership","Subscriber",1982,1
```

It is in comma-delimited format with double quote enclosing and a single header line. This will come into play later in this module as we configure the Snowflake table which will store this data.

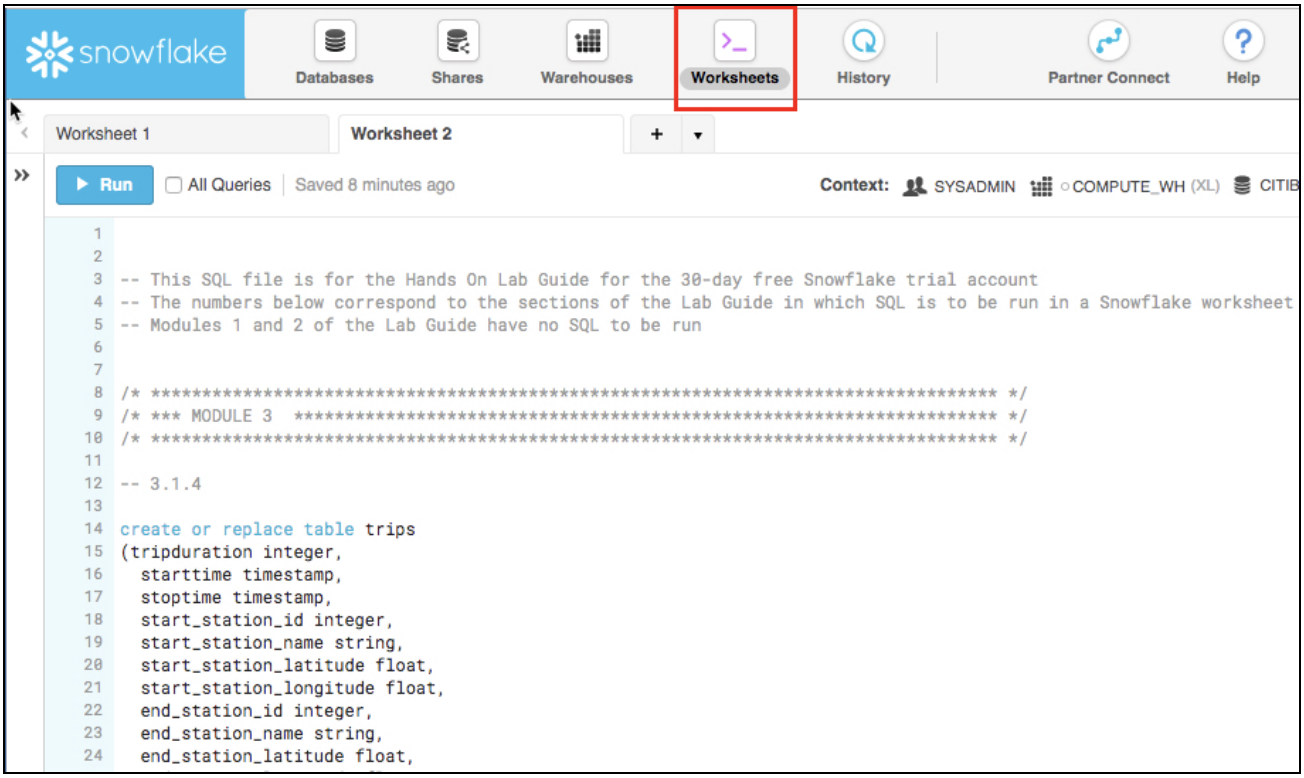
3.1 Create a Database and Table

3.1.1 First, let's create a database called CITIBIKE that will be used for loading the structured data.

At the top of the UI select the "Databases" tab. Then click on "Create" and name the database "CITIBIKE" and click "Finish".



3.1.2 At the top of the Snowflake UI, click the Worksheets tab. You should see the worksheet with all the SQL we loaded in a prior step.



3.1.3 First, we need to set the context appropriately within the Worksheet. In the top right, click on the drop-down arrow next to the “Context” section to show the worksheet context menu. Here we control what elements the user can see and run from each worksheet. We are using the UI here to set the context. Later in the lab we will set the worksheet context via SQL commands in the worksheet.

As needed use the downward arrows to select and show the showing:

Role: SYSADMIN
Warehouse: COMPUTE_WH (XL)
Database: CITIBIKE
Schema = PUBLIC

DDL operations are free!
Note that all the DDL operations we have done so far do NOT require compute resources, so we can create all our objects for free.

3.1.4 Let’s now create a table called TRIPS that will be used for loading the comma-delimited data. We will be using the Worksheets tab in the Snowflake UI to run the DDL (data definition language) to create the table. Based on a prior step, the SQL text below should be showing on the worksheet.

```
create or replace table trips
(tripduration integer,
  starttime timestamp,
  stoptime timestamp,
  start_station_id integer,
  start_station_name string,
  start_station_latitude float,
```

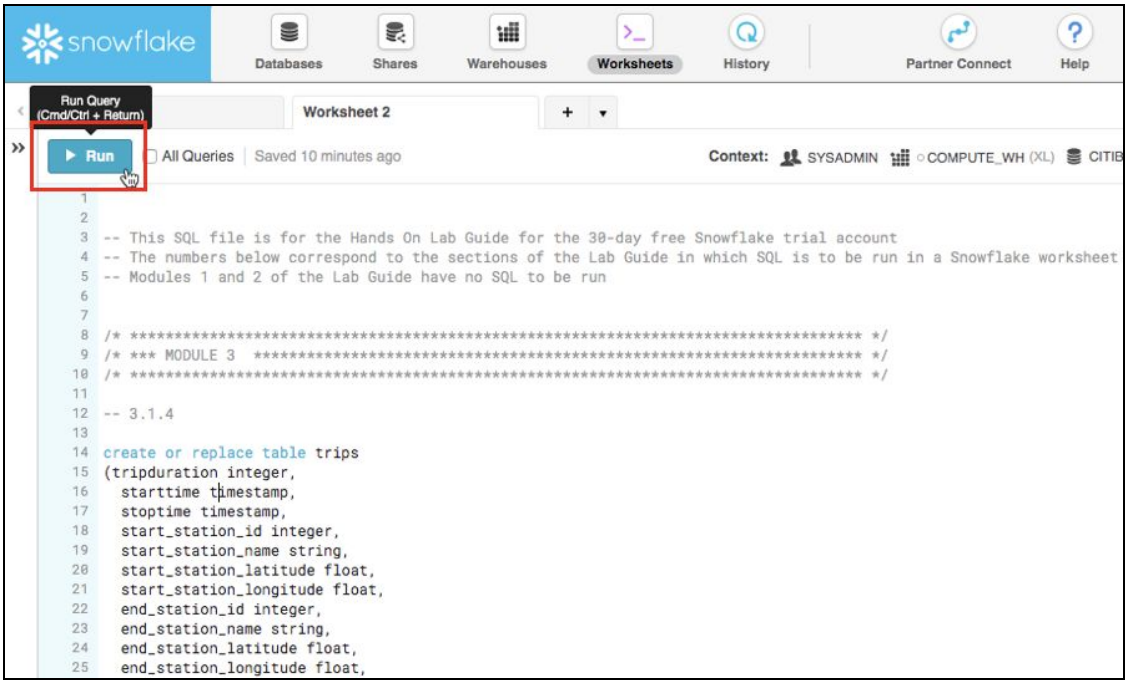
```
start_station_longitude float,
end_station_id integer,
end_station_name string,
end_station_latitude float,
end_station_longitude float,
bikeid integer,
membership_type string,
usertype string,
birth_year integer,
gender integer);
```

Many Options to Run Commands.
SQL commands can be executed through the UI (limited), via the Worksheets tab, using our SnowSQL command line tool, a SQL editor of your choice via ODBC/JDBC, or through our Python or Spark connectors.

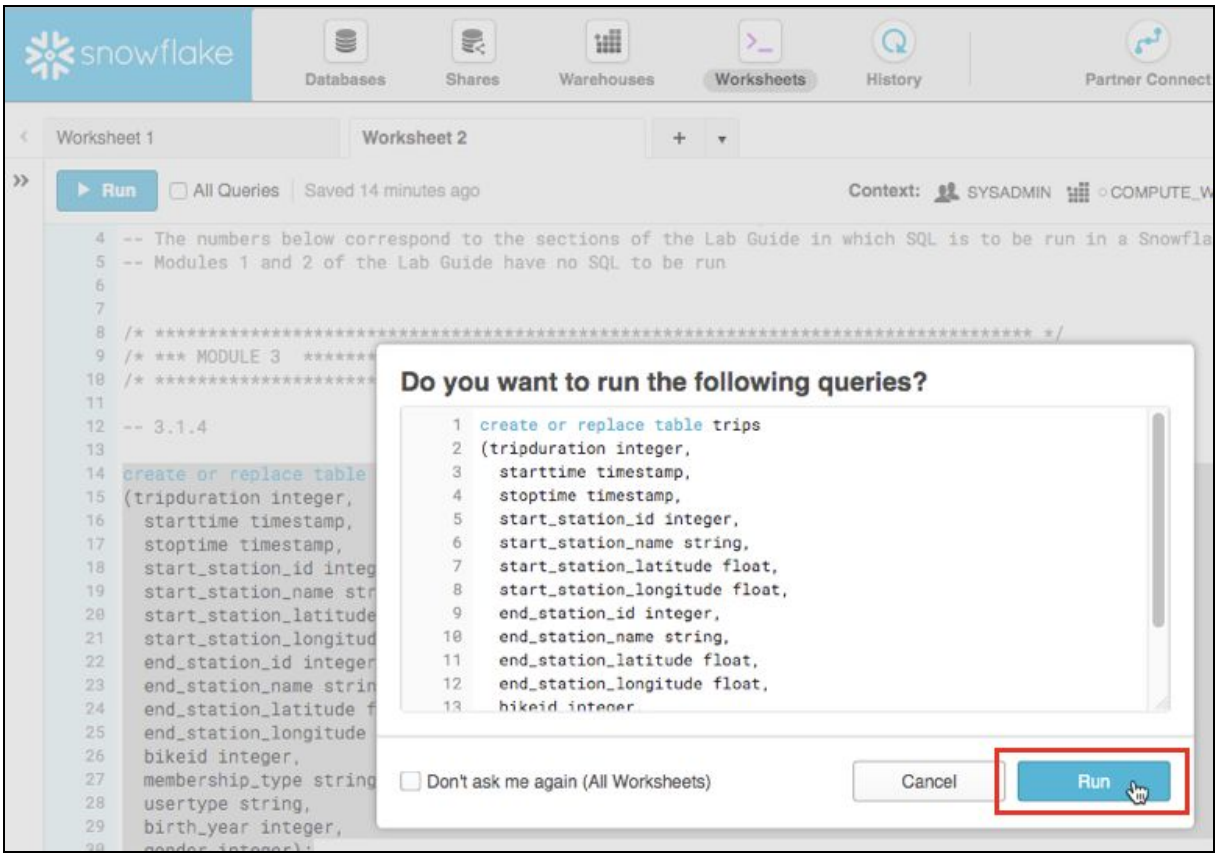
As mentioned earlier, in this lab we will run some operations via pre-written SQL in the worksheet (as opposed to using the UI) to save time.

3.1.5 Run the query by placing your cursor anywhere in the command and clicking the blue "Run" button at the top of the page or by hitting Ctrl/Cmd+Enter on your keyboard.

Warning
In this lab, never check the "All Queries" box at the top of the worksheet. We want to run SQL queries one at a time in a specific order; not all at once.



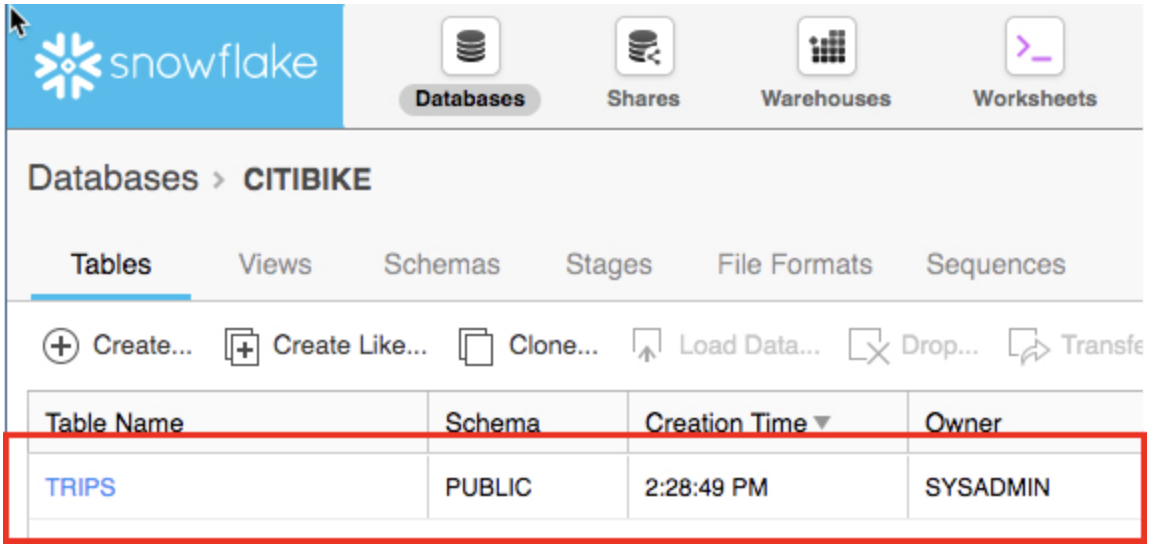
3.1.6 *If* you highlighted the entire SQL text of the command (did not just place your cursor in the command) and ran it, a confirmation box should appear asking “Do you want to run the following queries?”. Click the blue “Run” button in the box. In the future you can keep clicking this “Run” button on this confirmation box or check the “Don’t ask me again (All Worksheets)” option in this box.



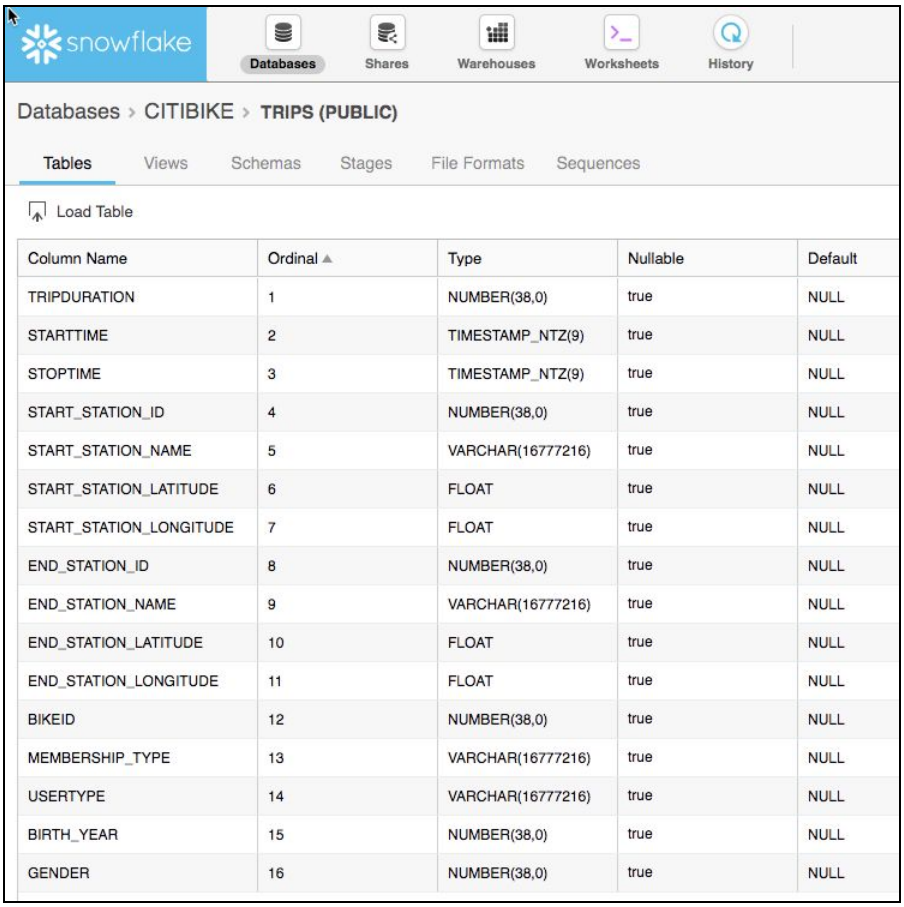
3.1.7 Verify that your table TRIPS has been created. At the bottom of the worksheet you should see a “Results” section which says “Table TRIPS successfully created”

3.1.8 At the top of the page, go to the Databases tab and then click on the “CITIBIKE” database link. You should see your newly created TRIPS table.

IMPORTANT: If you do not see the databases, expand your browser as they may be hidden.



3.1.9 Click on the “TRIPS” hyperlink to see the table structure you just configured for it.



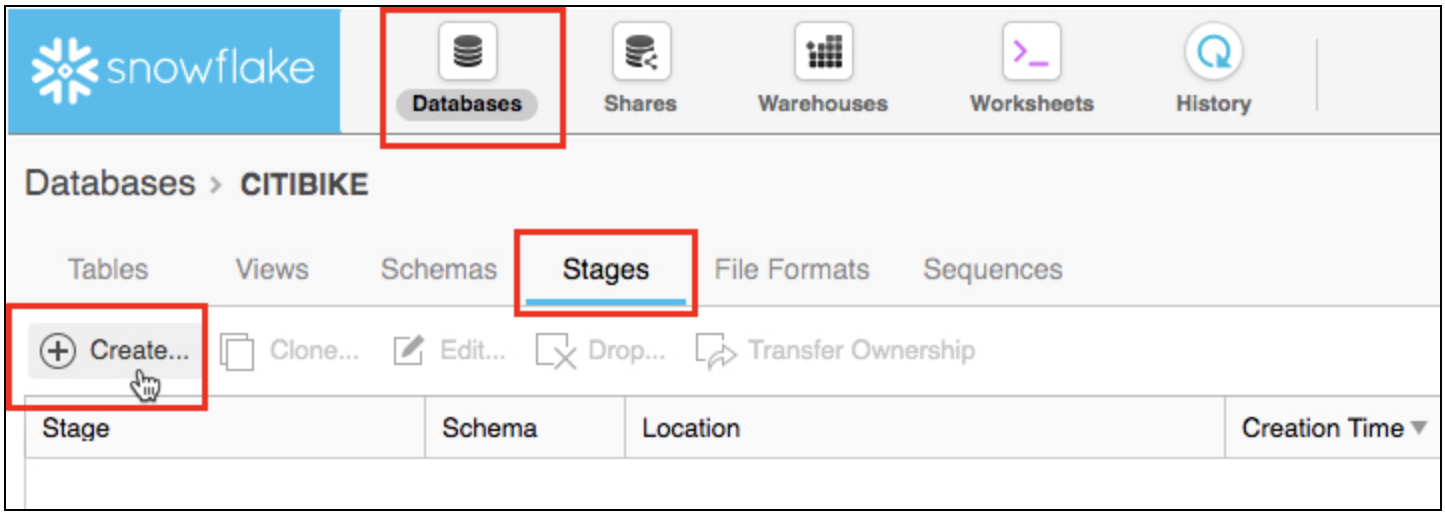
Column Name	Ordinal ▲	Type	Nullable	Default
TRIPDURATION	1	NUMBER(38,0)	true	NULL
STARTTIME	2	TIMESTAMP_NTZ(9)	true	NULL
STOPTIME	3	TIMESTAMP_NTZ(9)	true	NULL
START_STATION_ID	4	NUMBER(38,0)	true	NULL
START_STATION_NAME	5	VARCHAR(16777216)	true	NULL
START_STATION_LATITUDE	6	FLOAT	true	NULL
START_STATION_LONGITUDE	7	FLOAT	true	NULL
END_STATION_ID	8	NUMBER(38,0)	true	NULL
END_STATION_NAME	9	VARCHAR(16777216)	true	NULL
END_STATION_LATITUDE	10	FLOAT	true	NULL
END_STATION_LONGITUDE	11	FLOAT	true	NULL
BIKEID	12	NUMBER(38,0)	true	NULL
MEMBERSHIP_TYPE	13	VARCHAR(16777216)	true	NULL
USERTYPE	14	VARCHAR(16777216)	true	NULL
BIRTH_YEAR	15	NUMBER(38,0)	true	NULL
GENDER	16	NUMBER(38,0)	true	NULL

3.2 Create an External Stage

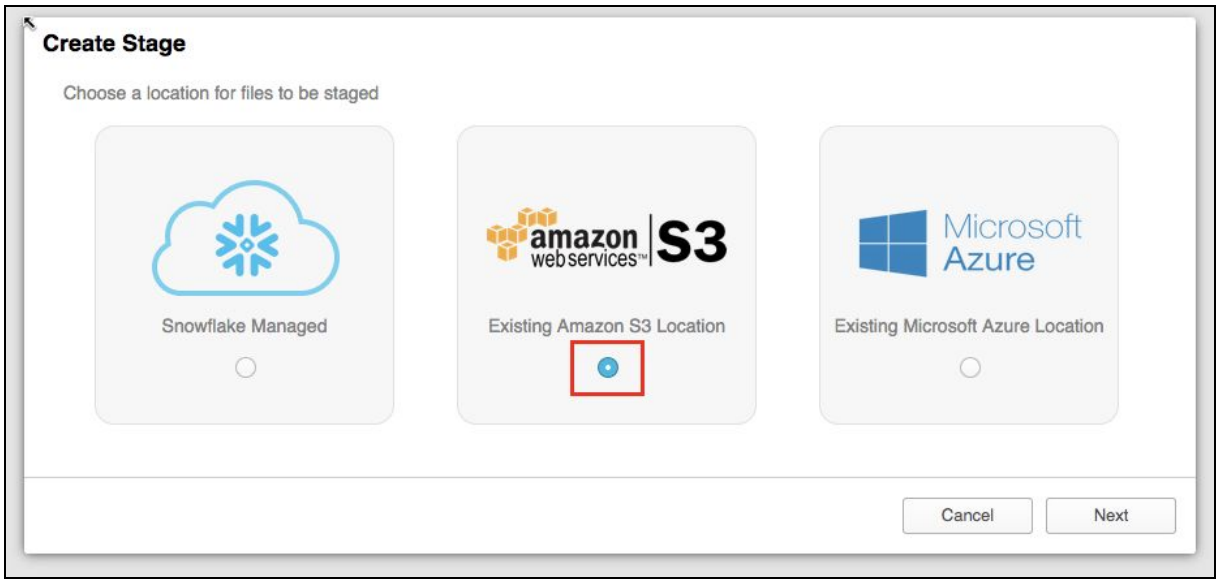
We are working with structured, comma-delimited data that has already been staged in a public, external S3 bucket. Before we can use this data, we first need to create a Stage that specifies the location of our external bucket.

NOTE - For this lab we are using an AWS-East bucket. In the real-world, to prevent data egress/transfer costs, you would want to select a staging location from the same cloud provider and region that your Snowflake environment is in.

3.2.1 From the Databases tab, click on the “CITIBIKE” database, then click on “Stages” and click “Create…”



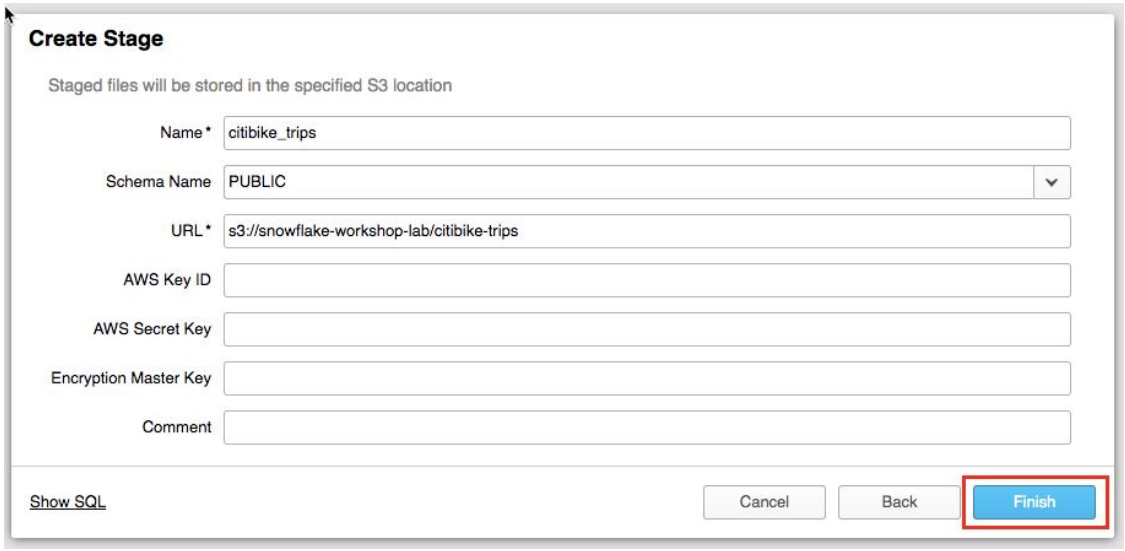
3.2.2 Select the option for “Existing Amazon S3 Location” and click “Next”:



3.2.3 On the “Create Stage” box that appears, enter/select the following settings, then click “Finish”.

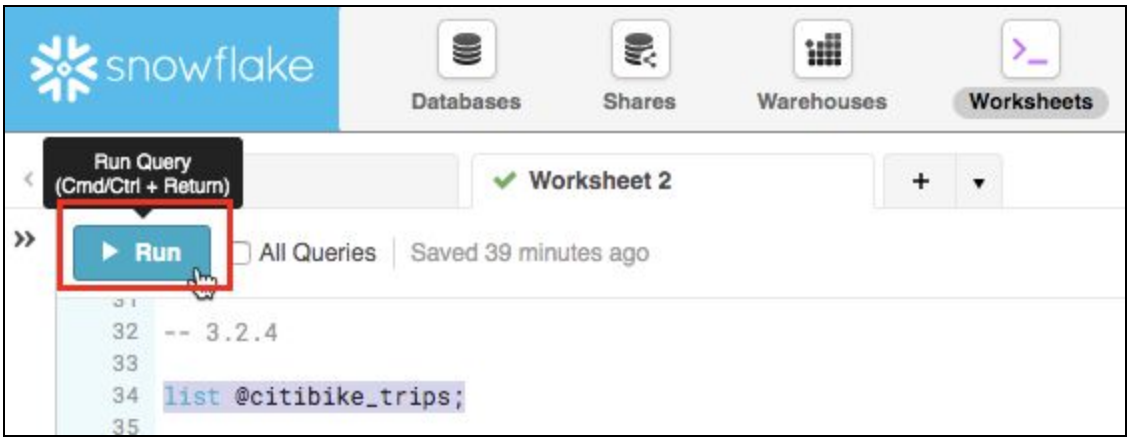
Name	citibike_trips
Schema Name	PUBLIC
URL	s3://snowflake-workshop-lab/citibike-trips

NOTE - The S3 bucket for this lab is public so you can leave the key fields empty. In the “real world” this bucket would likely require key information.



3.2.4 Now let's take a look at the contents of the citibike_trips stage. At the top of the page, click on the Worksheet tab. Then execute the following statement:

```
list @citibike_trips;
```



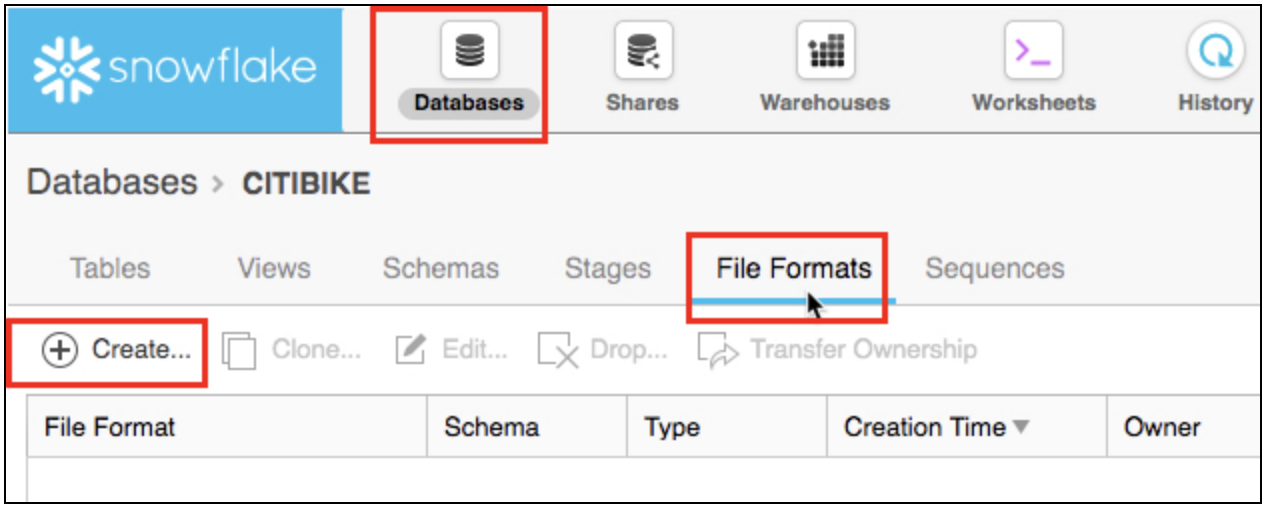
You should see the output in the Results window in the bottom pane:

Row	name	size	md5	last_modified
1	s3://snowflake-workshop-lab/citibike-tri…	3072073	cfc69e04228a94d1337ab383a3af7472	Wed, 12 Jun 2019 16:30:58 GMT
2	s3://snowflake-workshop-lab/citibike-tri…	2877852	92a1c064a3c632f338b57d5c6531e97d	Wed, 12 Jun 2019 16:30:58 GMT
3	s3://snowflake-workshop-lab/citibike-tri…	3174598	39faac098802c1b29f2d4d99f31378be	Wed, 12 Jun 2019 16:30:58 GMT
4	s3://snowflake-workshop-lab/citibike-tri…	3031012	cd0dca1dcfa309c0bb4bd40d1265c3a9	Wed, 12 Jun 2019 16:30:58 GMT
5	s3://snowflake-workshop-lab/citibike-tri…	3005838	fb24c0cc5fb6ee54d2aa4d50265792c7	Wed, 12 Jun 2019 16:30:58 GMT
6	s3://snowflake-workshop-lab/citibike-tri…	3099881	441efe806352c57a50c4f31afcccb2e3	Wed, 12 Jun 2019 16:30:58 GMT
7	s3://snowflake-workshop-lab/citibike-tri…	3060952	372765a1ca713eeb1d56f79e0956e48f	Wed, 12 Jun 2019 16:30:59 GMT

3.3 Create a File Format

Before we can load the data into Snowflake, we have to create a File Format that matches the data structure.

3.3.1 From the Databases tab, click on the CITIBIKE database hyperlink. Then click on “File Formats”. Then click “Create”.

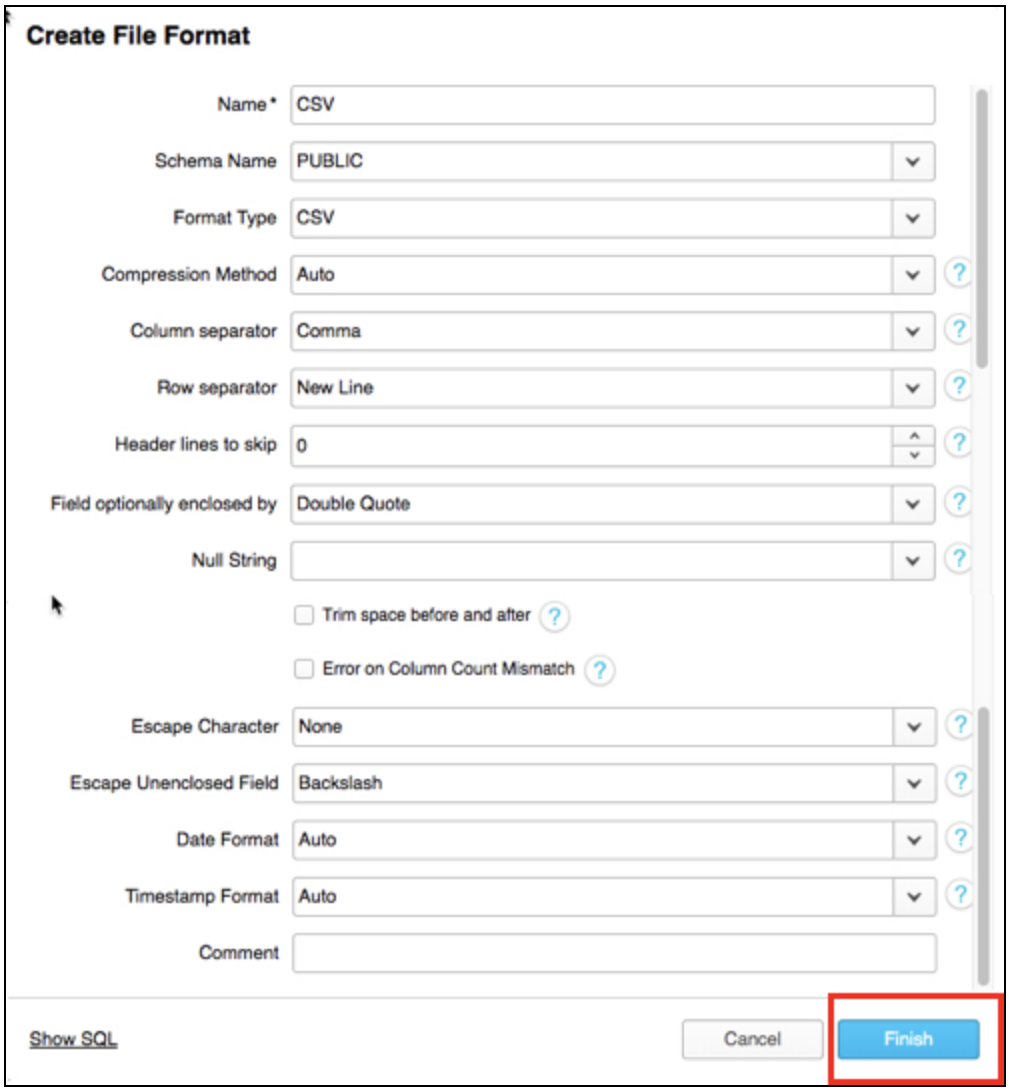


3.3.2 On the resulting page, we then create a file format. In the box that appears, leave all the default settings as-is but make the changes below:

Name: **CSV**
Field optionally enclosed by: **Double Quote**
Null string: <Delete the existing text in this field so it is empty>
[ ] Error on Column Count Mismatch: <uncheck this box>

IMPORTANT: If you do not see the “Error on Column Count Mismatch” box, scroll down in the dialogue box

When you are done, the box should look like:



Then click on the “Finish” button to create the file format.

Module 4: Loading Data

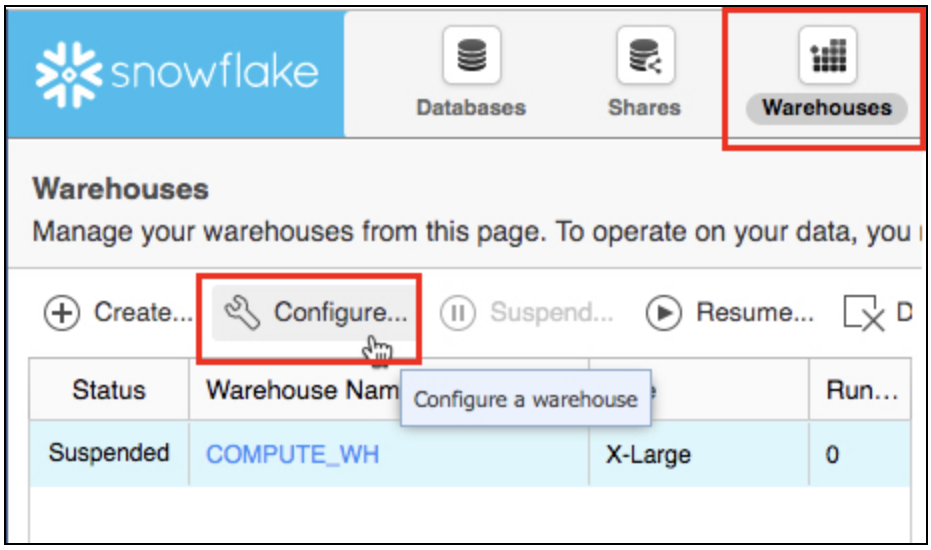
For this module, we will use a data warehouse and the COPY command to initiate bulk loading of the structured data into the Snowflake table we just created.

4.1 Resize and Use a Warehouse for Data Loading

For loading data, compute power is needed. Snowflake's compute nodes are called Warehouses and they can be dynamically sized up or out according to workload, whether the workload be loading data, running a query, or performing a DML operation. And each workload can have its own data warehouse so there is no resource contention.

4.1.1 Navigate to the Warehouses tab. Note the "Create…" option at the top is where you can quickly create a new warehouse. However, we want to use the existing warehouse "COMPUTE_WH" which comes with the 30-day trial environment.

Click on the row of this "COMPUTE_WH" warehouse (not the blue hyperlink that says "COMPUTE_WH") so the entire row is highlighted. Then click on the "Configure..." text above it to see the configuration detail of the "COMPUTE_WH". We will use this warehouse to load in the data from AWS S3.

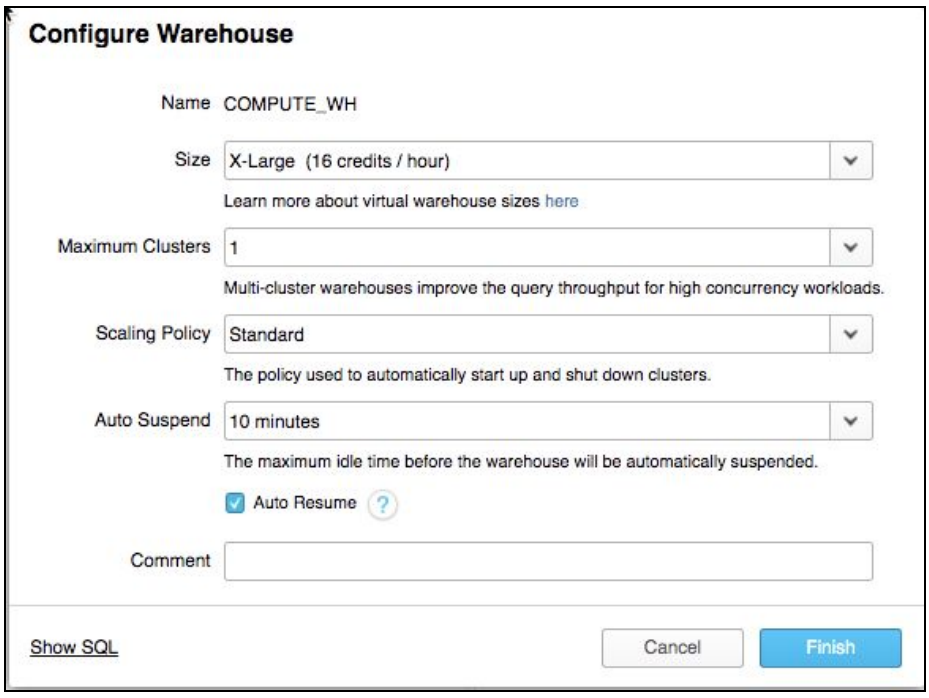


4.1.2 Let's walk through the settings of this warehouse as there is a lot of functionality here, much of which is unique to Snowflake versus other data warehouses.

NOTE - If you do not have a Snowflake Edition of Enterprise or greater, you will *NOT* see the "Maximum Clusters" or "Scaling Policy" configurations from the screenshot below. Multi-clustering is not utilized in this lab, but we will still discuss it as it is a key capability of Snowflake.

- The "Size" drop-down is where the size of the warehouse is selected. For larger data loading operations or more compute-intensive queries, a larger warehouse will be needed. The t-shirt sizes translate to underlying compute nodes, either AWS EC2 or Azure Virtual Machines. The larger the t-shirt size, the more compute resources from the cloud provider are allocated to that warehouse. As an example, the 4-XL option allocates 128 nodes. Also, this sizing can be changed up or down on the fly with a simple click.
- If you have Snowflake Edition or greater you will see the Maximum Clusters section. This is where you can set up a single warehouse to be multi-cluster up to 10 clusters. As an example, if the 4-XL warehouse we just mentioned was assigned a maximum cluster size of 10, it could scale up to be 1280 (128 * 10) AWS EC2 or Azure VM machines modes powering that warehouse...and it can do this in seconds! Multi-cluster is ideal for concurrency scenarios, such as many business analysts simultaneously running different queries using the same warehouse. In this scenario, the various queries can be allocated across the multiple clusters to ensure they run fast.

- The final sections allow you to automatically suspend the warehouse so it suspends (stops) itself when not in use and no credits are consumed. There is also an option to automatically resume (start) a suspended warehouse so when a new workload is assigned to it, it will automatically start back up. This functionality enables Snowflake's fair "pay as you use" compute pricing model which enables customers to minimize their data warehouse costs.



Snowflake Compute vs Other Warehouses
Many of the warehouse/compute capabilities we just covered, like being able to create, scale up and out, and auto-suspend/resume warehouses are things that are simple in Snowflake and can be done in seconds. Yet for on-premise data warehouses these capabilities are very difficult (or impossible) to do as they require significant physical hardware, over-provisioning of hardware for workload spikes, significant configuration work, and more challenges. Even other cloud data warehouses cannot scale up and out like Snowflake without significantly more configuration work and time.

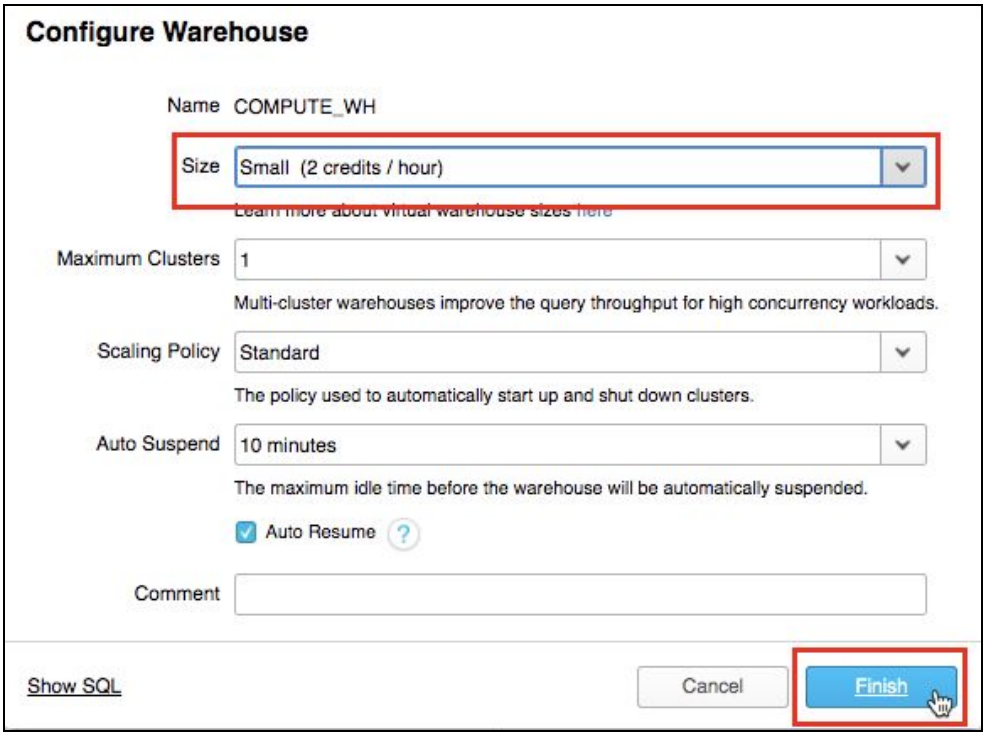
Warning - Watch Your Spend!
During or after this lab you should *NOT* do the following without good reason or you may burn through your $400 of free credits more quickly than desired:

- Disable auto-suspend. If auto-suspend is disabled, your warehouses will continue to run and consume credits even when not being utilized.
- Use a warehouse size that is excessive given the workload. The larger the warehouse, the more credits are consumed.

4.1.3 We are going to use this data warehouse to load the structured data into Snowflake. However, we are first going to decrease the size of the warehouse to reduce the compute power it contains. Then in later steps we will note the time this load takes, then

re-do the same load operation with a larger warehouse, and observe how much faster the load is with a larger warehouse.

Change the Size of this data warehouse from X-Large to Small. Then click the “Finish” button.

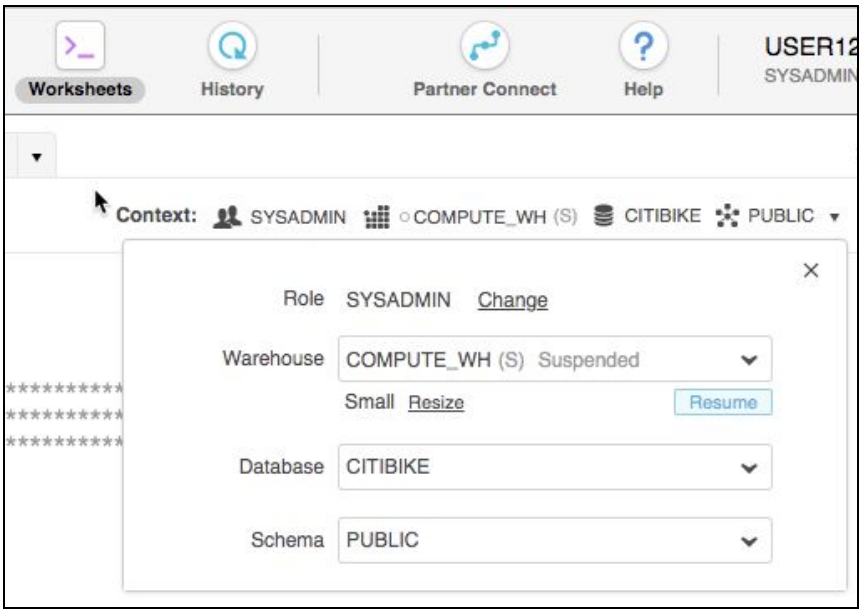


4.2 Load the Data

Now we can run a COPY command to load the data into the TRIPS table we created earlier.

4.2.1 Via the top of the UI, navigate back to the Worksheets tab and click on it. Make sure the context is correct by setting these in the top right of the worksheet:

Role: SYSADMIN
Warehouse: COMPUTE_WH (S)
Database: CITIBIKE
Schema = PUBLIC



4.2.2 Execute the following statements in the worksheet to load the staged data into the table. This may take up to 30 seconds.

```
copy into trips from @citibike_trips
file_format=CSV;
```

In the Results window, you should see the status of the load:

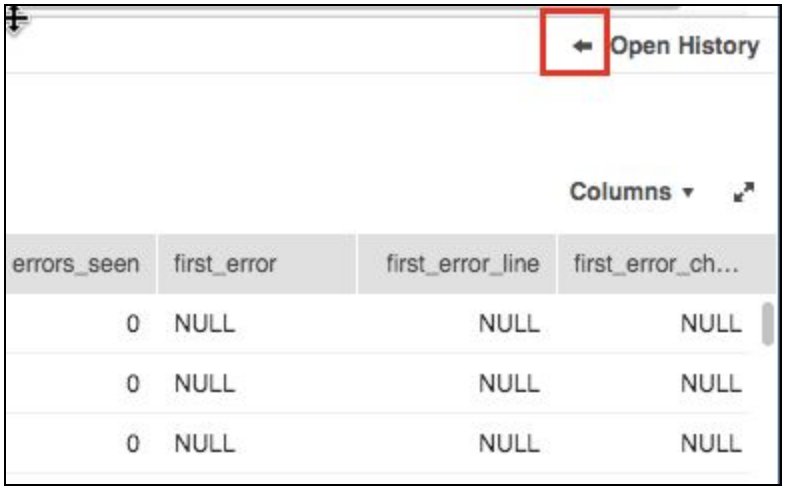
Results Data Preview — Open History

Query ID SQL 31.34s 376 rows

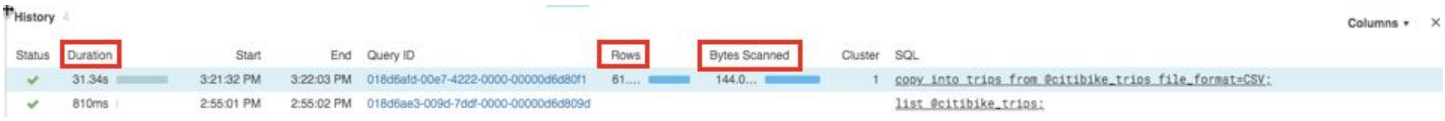
Filter result... Copy Columns

Row	file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error	first_error_line	first_error_ch...
1	s3://snowflak...	LOADED	116731	116731	1	0	NULL	NULL	NULL
2	s3://snowflak...	LOADED	85049	85049	1	0	NULL	NULL	NULL
3	s3://snowflak...	LOADED	81988	81988	1	0	NULL	NULL	NULL
4	s3://snowflak...	LOADED	108605	108605	1	0	NULL	NULL	NULL
5	s3://snowflak...	LOADED	109178	109178	1	0	NULL	NULL	NULL
6	s3://snowflak...	LOADED	99747	99747	1	0	NULL	NULL	NULL
7	s3://snowflak...	LOADED	144097	144097	1	0	NULL	NULL	NULL

4.2.3 Once the load is done, at the bottom right of the worksheet click on the small arrow next to the “Open History” text to show the history of Snowflake operations performed in that specific worksheet.



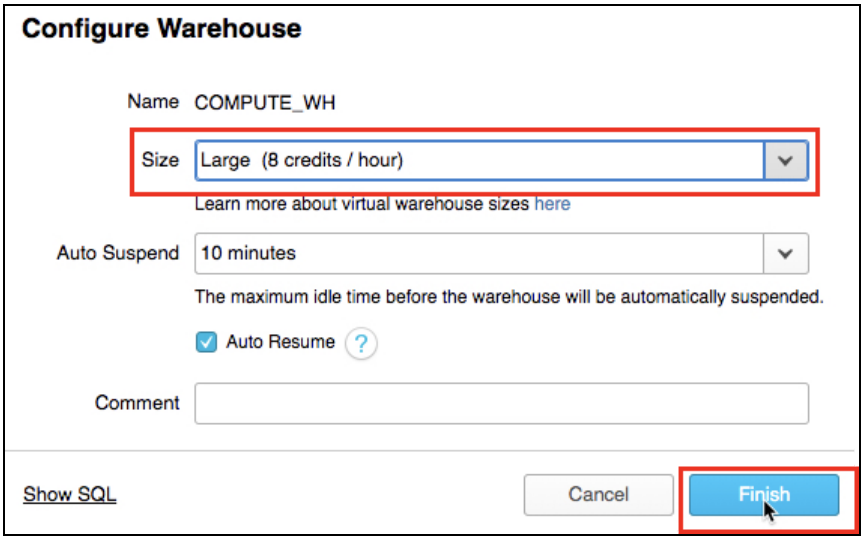
4.2.4 In the History window see the “copy into trips from @citibike_trips file_format=CSV;” SQL query you just ran and note the duration, bytes scanned and rows. Use the slider on the left side of the pane to expand it if needed to see all the detail in it.



4.2.5 Go back to the worksheet to clear the table of all data and metadata, by using the TRUNCATE TABLE command.

```
truncate table trips;
```

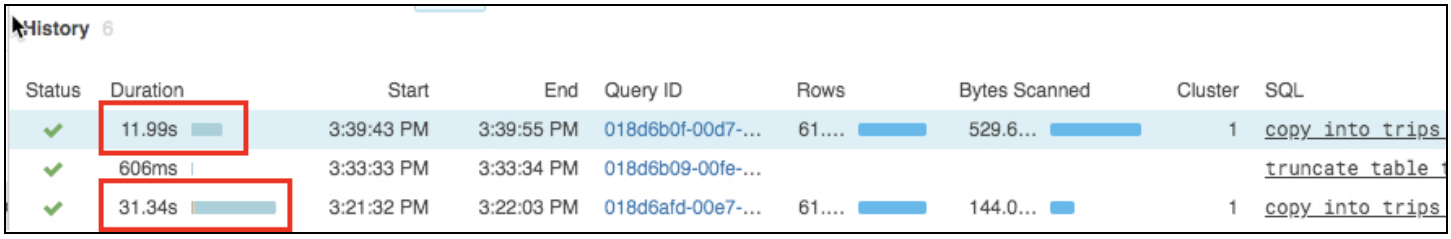
4.2.6 At the top of the UI go to the Warehouses tab, then click on “Configure..” Resize the warehouse to size Large and click “Finish”. This warehouse is four times larger than the Small size.



4.2.7 Go back to the Worksheets tab at the top of the UI and click on it. Execute the following statements in the worksheet to load the same data again.

```
copy into trips from @citibike_trips
file_format=CSV;
```

4.2.8 Once the load is done, at the bottom of the worksheet in the History window compare the times between the two loads. The one with the Large warehouse was significantly faster.

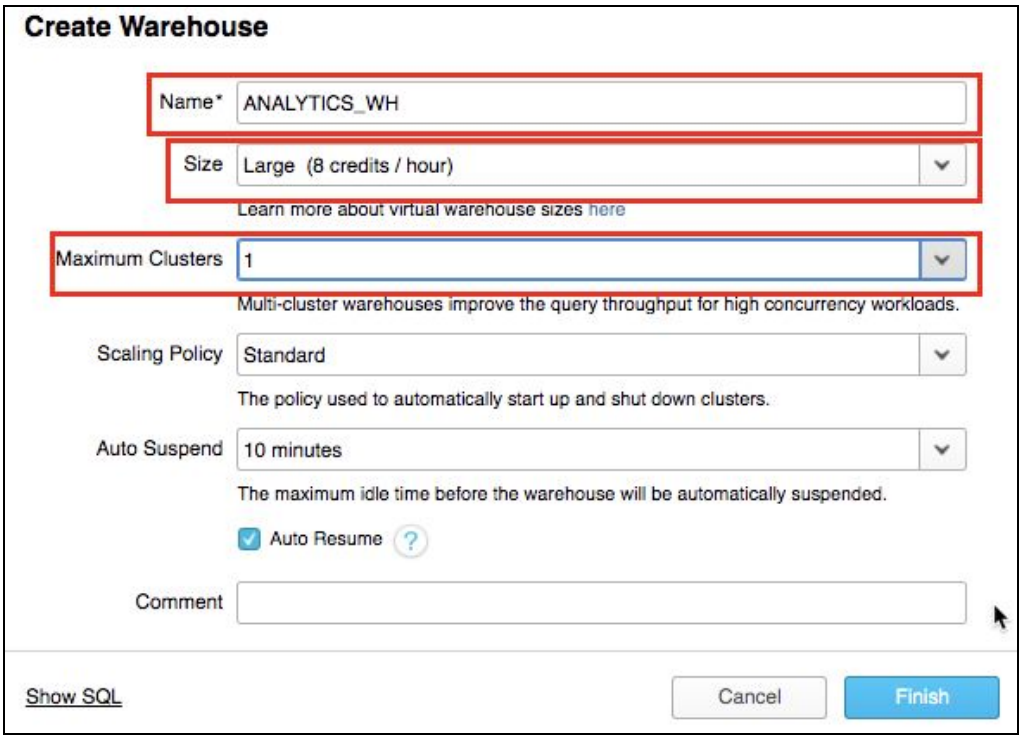


4.3 Create a New Warehouse for Data Analytics

Going back to the lab story, let’s assume the Citi Bike team wants to ensure no resource contention between their data loading/ETL workloads and the analytical end users using BI tools to query Snowflake. As mentioned earlier, Snowflake can easily do this by assigning different, appropriately-sized warehouses to different workloads. Since Citi Bike already has a warehouse for data loading, let’s create a new warehouse for the end users running analytics. We will then use this warehouse to perform analytics in the next module.

4.3.1 At the top of the UI click on the Warehouses tab then “Create…”. Name it “ANALYTICS_WH” with size “Large”. If you have Snowflake edition Enterprise or greater, you will see a setting for “Maximum Clusters”. Set this to “1”.

Leave the other settings in their default settings. It should look like:



Then click on the “Finish” button to create the warehouse.

Module 5: Analytical Queries, Results Cache, Cloning

In the previous exercises, we loaded data into two tables using Snowflake's bulk loader (COPY command) and the warehouse COMPUTE_WH. Now we are going to pretend we are analytics users at Citi Bike who need to query data in those tables using the worksheet and the second warehouse ANALYTICS_WH.

"Real World" Roles and Querying
In the "real world" the analytics users would likely have a different role then SYSADMIN; to keep the lab simple we are going to stay with the SYSADMIN role for this module.

Also, in the "real-world" querying would typically be done with a business intelligence product like Tableau, Looker, PowerBI, etc. Or for more advanced analytics, data science products like Spark or R can query Snowflake. Basically any technology that leverages JDBC/ODBC can run analytics on the data in Snowflake. But to keep this lab simple, all queries are being done via the Snowflake worksheet.

5.1 Execute SELECT Statements and Result Cache

5.1.1 Go the Worksheets tab. Within the worksheet, make sure you set your context appropriately:

Role: SYSADMIN
Warehouse: ANALYTICS_WH (L)
Database: CITIBIKE
Schema = PUBLIC

5.1.2 Run the query below to see a sample of the trips data

```
select * from trips limit 20;
```

Results Data Preview

Query ID SQL 4.05s 20 rows

Filter result... Copy

Row	TRIPDURATION	STARTTIME	STOPTIME	START_STATION...	START_STATION_NAME	START_STATION_LATITUDE
1	518	2016-09-18 ...	2016-09-18 ...	3108	Nassau Ave & Russell St	40.72557
14	1898	2016-09-18 ...	2016-09-18 ...	3345	Madison Ave & E 99 St	40.789485416
13	1231	2016-09-18 ...	2016-09-18 ...	3168	Central Park West & W 8...	40.78472675
8	510	2016-09-18 ...	2016-09-18 ...	3160	Central Park West & W 7...	40.77896784
17	847	2016-09-18 ...	2016-09-18 ...	3423	West Drive & Prospect P...	40.661063372

5.1.3 First let's look at some basic hourly statistics on Citi Bike usage. Run the query below in the worksheet. It will show for each hour the number of trips, average trip duration, and average trip distance.

```
select date_trunc('hour', starttime) as "date",
count(*) as "num trips",
avg(tripduration)/60 as "avg duration (mins)",
avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude,
                        end_station_longitude)) as "avg distance (km)"
from trips
group by 1 order by 1;
```

Results Data Preview Open History

Query ID SQL 984ms 44,295 rows

Filter result... Copy Columns

Row	date	num trips	avg duration (mins)	avg distance (km)
1	2013-06-01 00:00:00.000	152	56.058442983333	2.127971476
2	2013-06-01 01:00:00.000	102	26.525163400000	2.067906273
3	2013-06-01 02:00:00.000	67	36.119900500000	2.31784827
4	2013-06-01 03:00:00.000	41	44.485365850000	2.349126632
5	2013-06-01 04:00:00.000	16	23.278125000000	1.840026007

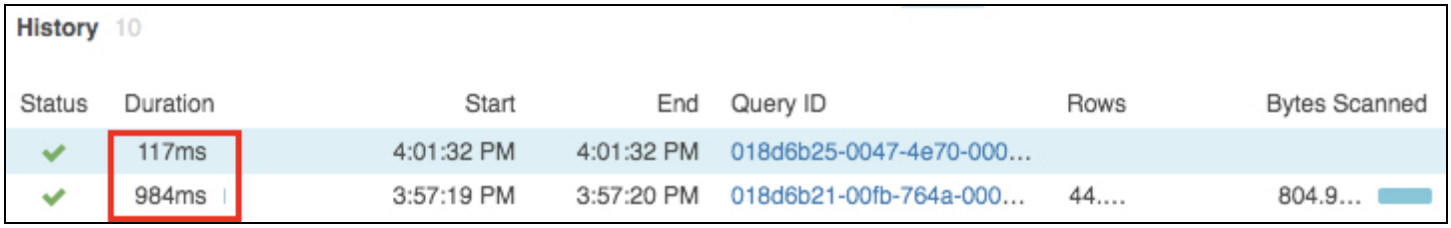
5.1.4 Snowflake has a result cache that holds the results of every query executed in the past 24 hours. These are available across warehouses, so query results returned to one user are available to any other user on the system who executes the same query, provided

the underlying data has not changed. Not only do these repeated queries return extremely fast, but they also use no compute credits.

Let's see the result cache in action by running the exact same query again.

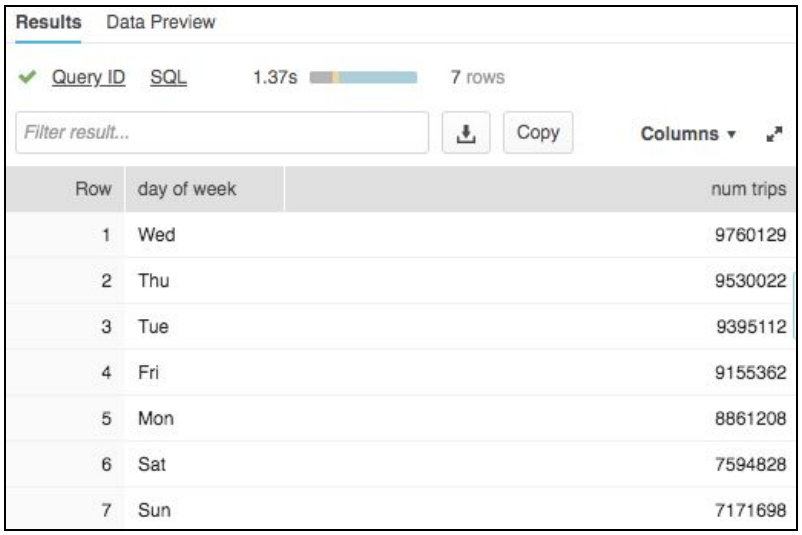
```
select date_trunc('hour', starttime) as "date",
count(*) as "num trips",
avg(tripduration)/60 as "avg duration (mins)",
avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude,
                        end_station_longitude)) as "avg distance (km)"
from trips
group by 1 order by 1;
```

In the History window note that the query runs significantly faster now because the results have been cached.



5.1.5 Next, let's run this query to see which days of the week are the busiest:

```
select
    dayname(starttime) as "day of week",
    count(*) as "num trips"
from trips
group by 1 order by 2 desc;
```



5.2 Clone a Table

Snowflake allows you to create clones, also known as “zero-copy clones” of tables, schemas, and databases in seconds. A snapshot of data present in the source object is taken when the clone is created, and is made available to the cloned object. The cloned object is writable, and is independent of the clone source. That is, changes made to either the source object or the clone object are not part of the other.

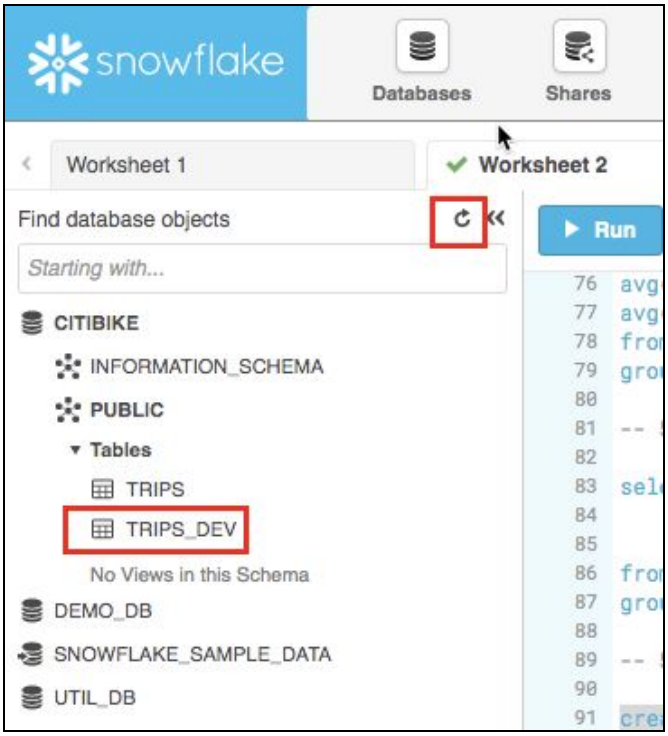
A popular use case for zero-copy cloning is to clone a production environment for use by Development & Testing to do testing and experimentation on without (1) adversely impacting the production environment and (2) eliminating the need to set up and manage two separate environments for production and Development & Testing.

Zero-Copy Cloning FTW!
A massive benefit is that the underlying data is not copied; just the metadata/pointers to the underlying data change. Hence “zero-copy” and storage requirements are not doubled when data is cloned. Most data warehouses cannot do this; for Snowflake it is easy!

5.2.1 Run the following command in the worksheet to create a development (dev) table

```
create table trips_dev clone trips
```

5.2.2 If closed, expand the database objects browser on the left of the worksheet. Click the small Refresh button in the left-hand panel and expand the object tree under the Citibike database. Check that you can see a new table under the CITIBIKE database named TRIPS_DEV. The development team now can do whatever they want with this table, including even deleting it, without having any impact on the TRIPS table or any other object.



Module 6: Working With Semi-Structured Data, Views, JOIN

NOTE - The first steps here are similar to prior Modules 3 (Preparing to Load Data) and 4 (Loading Data) but we will do most of it via SQL in the worksheet, as opposed to via the UI, to save time.

Going back to the lab "story", the Citi Bike analytics team wants to see how weather impacts ride counts. To do this, in this module we will:

- Load weather data in JSON format held in a public S3 bucket
- Create a View and query the semi-structured data using SQL dot notation
- Run a query that joins the JSON data to the TRIPS data from a prior module of this guide
- See how weather impacts trip counts

The JSON data consists of weather information provided by OpenWeatherMap detailing the historical conditions of New York City from 2016-07-05 to 2019-06-25. It is also staged on AWS S3 where the data represents 57.9k rows, 61 objects, and 2.5MB total size compressed.

The raw JSON in GZ files and in a text editor looks like:


```
{"city":{"coord":{"lat":43.000351,"lon":-75.499901},"country":"US","findname":"NEW YORK","id":5128638,"name":"New York","zoom":
1},"clouds":{"all":90},"main":{"humidity":93,"pressure":1008,"temp":293.47,"temp_max":295.37,"temp_min":292.04},"time":
1561467737,"weather":[{"description":"moderate rain","icon":"10d","id":501,"main":"Rain"}],"wind":{"deg":170,"speed":4.1}}
{"city":{"coord":{"lat":40.714272,"lon":-74.005966},"country":"US","findname":"NEW YORK","id":5128581,"name":"New York","zoom":
1},"clouds":{"all":90},"main":{"humidity":94,"pressure":1010,"temp":295.16,"temp_max":296.15,"temp_min":294.15},"time":
1561467737,"weather":[{"description":"light rain","icon":"10d","id":500,"main":"Rain"},{"description":"mist","icon":"50d","id":
701,"main":"Mist"}],"wind":{"deg":0,"speed":2.1}}
{"city":{"coord":{"lat":43.000351,"lon":-75.499901},"country":"US","findname":"NEW YORK","id":5128638,"name":"New York","zoom":
1},"clouds":{"all":90},"main":{"humidity":94,"pressure":1008,"temp":294.58,"temp_max":297.04,"temp_min":292.04},"time":
1561471336,"weather":[{"description":"overcast clouds","icon":"04d","id":804,"main":"Clouds"}],"wind":{"deg":270,"speed":3.1}}
{"city":{"coord":{"lat":40.714272,"lon":-74.005966},"country":"US","findname":"NEW YORK","id":5128581,"name":"New York","zoom":
1},"clouds":{"all":90},"main":{"humidity":100,"pressure":1010,"temp":295.37,"temp_max":296.48,"temp_min":294.26},"time":
1561471336,"weather":[{"description":"mist","icon":"50d","id":701,"main":"Mist"}],"wind":{"deg":170.797,"speed":0.4}}
```


SEMI-STRUCTURED DATA
Snowflake can easily load and query semi-structured data, such as JSON, Parquet, or Avro, without transformation. This is important because an increasing amount of business-relevant data being generated today is semi-structured, and many traditional data warehouses cannot easily load and query this sort of data. With Snowflake it is easy!

6.1 Create a Database and Table

6.1.1 First, via the Worksheet, let's create a database called WEATHER that will be used for storing the unstructured data.

```
create database weather;
```

6.1.2 Set the context appropriately within the Worksheet.

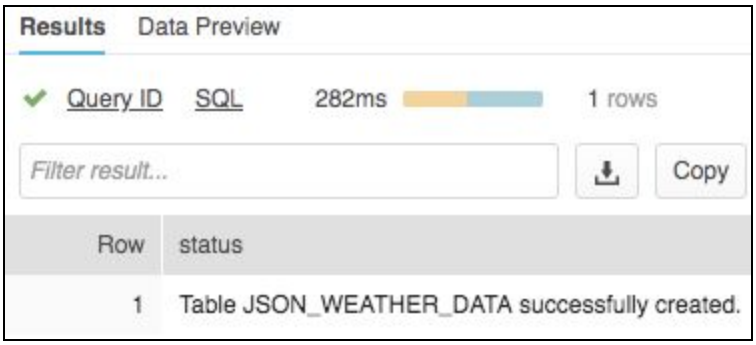
```
use role sysadmin;
use warehouse compute_wh;
use database weather;
use schema public;
```

6.1.3 Via the worksheet, let's now create a table called JSON_WEATHER_DATA that will be used for loading the JSON data. In the worksheet, run the SQL text below. Snowflake has a special column type called VARIANT which will allow us to store the entire JSON object and eventually query it directly.

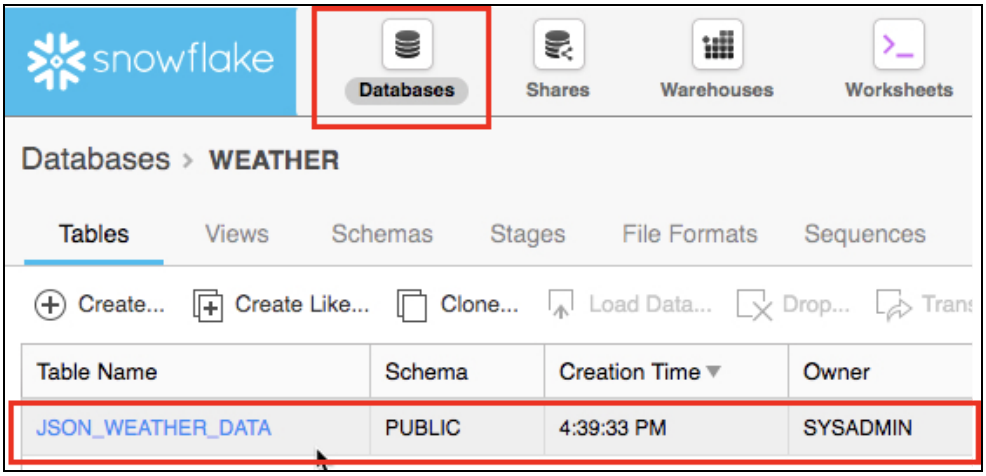
```
create table json_weather_data (v variant);
```

Semi-Structured Data Magic
Snowflake's VARIANT data type allows Snowflake to ingest semi-structured data without having to pre-define the schema.

6.1.4 Verify that your table JSON_WEATHER_DATA has been created. At the bottom of the worksheet you should see a "Results" section which says "Table JSON_WEATHER_DATA successfully created"



6.1.5 At the top of the page, go to the Databases tab and then click on the “WEATHER” database link. You should see your newly created JSON_WEATHER_DATA table.



6.2 Create an External Stage

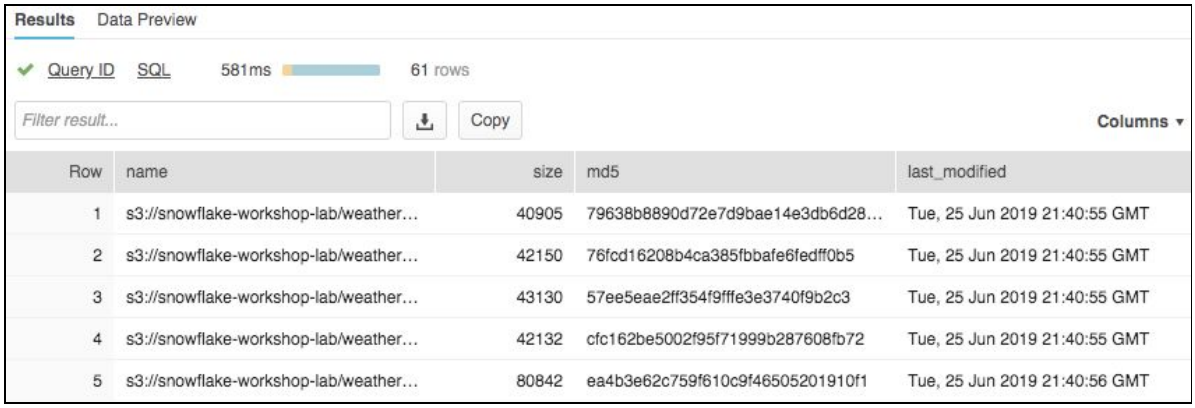
6.2.1 Via the Worksheet create a stage from where the unstructured data is stored on AWS S3.

```
create stage nyc_weather
url = 's3://snowflake-workshop-lab/weather-nyc';
```

6.2.2 Now let’s take a look at the contents of the nyc_weather stage. At the top of the page, click on the Worksheets tab. In the worksheet, then execute the following statement with a LIST command to see the list of files:

```
list @nyc_weather;
```

You should see the output in the Results window in the bottom pane showing many gz files from S3:



Row	name	size	md5	last_modified
1	s3://snowflake-workshop-lab/weather…	40905	79638b8890d72e7d9bae14e3db6d28…	Tue, 25 Jun 2019 21:40:55 GMT
2	s3://snowflake-workshop-lab/weather…	42150	76fcd16208b4ca385fbbafe6fedff0b5	Tue, 25 Jun 2019 21:40:55 GMT
3	s3://snowflake-workshop-lab/weather…	43130	57ee5eae2ff354f9fffe3e3740f9b2c3	Tue, 25 Jun 2019 21:40:55 GMT
4	s3://snowflake-workshop-lab/weather…	42132	cfc162be5002f95f71999b287608fb72	Tue, 25 Jun 2019 21:40:55 GMT
5	s3://snowflake-workshop-lab/weather…	80842	ea4b3e62c759f610c9f46505201910f1	Tue, 25 Jun 2019 21:40:56 GMT

6.3 Loading and Verifying the Unstructured Data

For this section, we will use a warehouse to load the data from the S3 bucket into the Snowflake table we just created.

6.3.1 Via the worksheet, run a COPY command to load the data into the JSON_WEATHER_DATA table we created earlier.

Note how in the SQL here we can specify a FILE FORMAT object inline. In the prior module where we loaded structured data, we had to define a file format in detail. But because the JSON data here is well-formatted, we use default settings and simply specify the JSON type.

```
copy into json_weather_data
from @nyc_weather
file_format = (type=json);
```

6.3.2 Take a look at the data that has been loaded.

```
select * from json_weather_data limit 10;
```

6.3.3 Click on one of the values. Notice how the data is stored in raw JSON format. Click “Done” when finished.

6.4 Create a View and Query Semi-Structured Data

Let's look at how Snowflake allows us to create a view and also query the JSON data directly using SQL.

Views & Materialized Views

A View allows the result of a query to be accessed as if it were a table. Views can help you: present data to end users in a cleaner manner (like in this lab we will present "ugly" JSON in a columnar format), limit what end users can view in a source table for privacy/security reasons, or write more modular SQL.

There are also Materialized Views in which SQL results are stored, almost as though the results were a table. This allows faster access, but requires storage space. Materialized Views require Snowflake Enterprise Edition or higher.

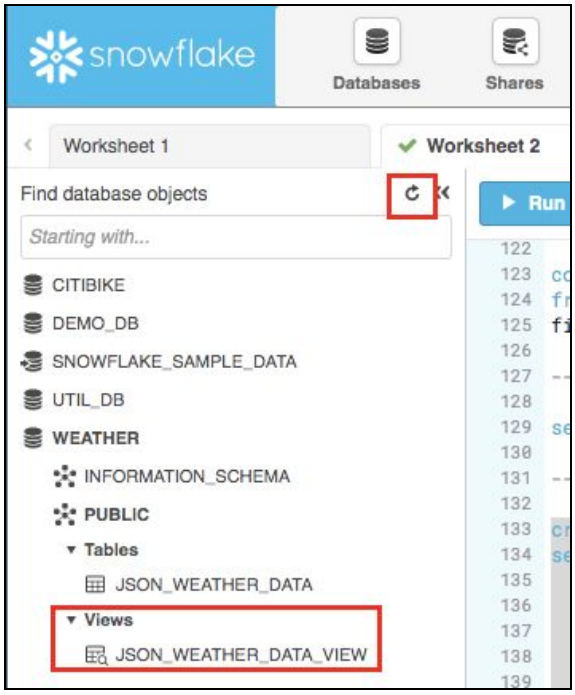
6.4.1 From the Worksheet tab, go into the worksheet and run the following command. It will create a view of the unstructured JSON weather data in a columnar view so it is easier for analysts to understand and query. FYI - the city ID for New York City is 5128638.

```
create view json_weather_data_view as
select
  v:time::timestamp as observation_time,
  v:city.id::int as city_id,
  v:city.name::string as city_name,
  v:city.country::string as country,
  v:city.coord.lat::float as city_lat,
  v:city.coord.lon::float as city_lon,
  v:clouds.all::int as clouds,
  (v:main.temp::float)-273.15 as temp_avg,
  (v:main.temp_min::float)-273.15 as temp_min,
```

```
  (v:main.temp_max::float)-273.15 as temp_max,
  v:weather[0].main::string as weather,
  v:weather[0].description::string as weather_desc,
  v:weather[0].icon::string as weather_icon,
  v:wind.deg::float as wind_dir,
  v:wind.speed::float as wind_speed
from json_weather_data
where city_id = 5128638;
```

6.4.2 The prior step showed how you can use SQL dot notation (**v.city.coord.lat**) to pull out values at lower levels in the JSON hierarchy. This allows us to treat each field as if it were a column in a relational table.

6.4.3 Verify the view at the top left of the UI where the new view should appear just under the table json_weather_data. You may need to expand and and/or refresh the database objects browser in order to see it.



6.4.4 Via the worksheet, verify the view with the following query. Notice the results look just like a regular structured data source (NOTE - your result set may have different observation_time values)

```
select * from json_weather_data_view
where date_trunc('month',observation_time) = '2018-01-01'
limit 20;
```

Results Data Preview

Query ID SQL 352ms 20 rows

Filter result... Copy

Row	OBSERVATION_TIME	CITY_ID	CITY_NAME	COUNTRY	CITY_LAT	CITY_LON	CLOUDS
1	2018-01-03 01:02:38.000	5128638	New York	US	43.000351	-75.499901	90
2	2018-01-03 02:05:20.000	5128638	New York	US	43.000351	-75.499901	90
3	2018-01-15 13:04:58.000	5128638	New York	US	43.000351	-75.499901	1
4	2018-01-15 14:03:24.000	5128638	New York	US	43.000351	-75.499901	1
5	2018-01-14 04:02:46.000	5128638	New York	US	43.000351	-75.499901	1
6	2018-01-14 05:04:41.000	5128638	New York	US	43.000351	-75.499901	1

6.5 Use a Join Operation to Correlate Against Data Sets

We will now join the JSON weather data to our CITIBIKE.PUBLIC.TRIPS data to determine the answer to our original question of how weather impacts the number of rides.

6.5.1 Run the command below to join WEATHER to TRIPS and count the number of trips associated with certain weather conditions .

Note - Since we are still in a worksheet use the WEATHER database as default, we will fully qualify our reference to the TRIPS table by providing its database and schema name.

```
select weather as conditions
    ,count(*) as num_trips
 from citibike.public.trips
 left outer join json_weather_data_view
    on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
 where conditions is not null
 group by 1 order by 2 desc;
```

Results | Data Preview

Query ID | SQL | 435ms | 10 rows

Filter result... | Copy | Columns

Row	CONDITIONS	NUM_TRIPS
1	Clear	9249531
2	Clouds	8934964
3	Rain	3206720
4	Snow	1380418
5	Mist	798487
6	Fog	362496
7	Thunderstorm	230539
8	Drizzle	98779
9	Haze	40724
10	Smoke	2871

6.5.2 The Citi Bike initial goal was to see if there was any correlation between the number of bike rides and weather by analyzing both ridership and weather data. Per the table above we have a clear answer. As one would imagine, the number of trips is significantly higher when the weather is good!

For the rest of this lab, we will look at other capabilities of Snowflake.

Module 7: Using Time Travel

Snowflake's Time Travel capability enables accessing historical data at any point within a pre-configurable period of time. The default period of time is 24 hours and with Snowflake Enterprise Edition it can be up to 90 days. Most data warehouses cannot offer this functionality; with Snowflake it is easy!

Some useful applications of this include:

- Restoring data-related objects (tables, schemas, and databases) that may have been accidentally or intentionally deleted
- Duplicating and backing up data from key points in the past
- Analysing data usage/manipulation over specified periods of time

7.1 Drop and Undrop a Table

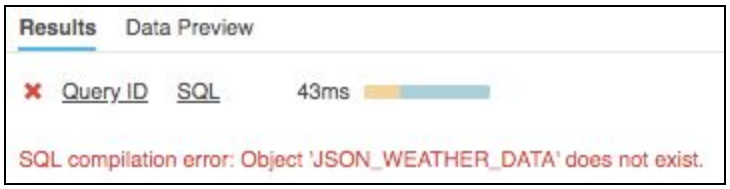
First let's see how we can restore data objects that have been accidentally or intentionally deleted.

7.1.1 From the worksheet, run the following command which will drop (remove) the json_weather_data table:

```
drop table json_weather_data;
```

7.1.2 Now run a SELECT statement on the json_weather_data table. In the "Results" pane you should see an error because the underlying table has been dropped.

```
select * from json_weather_data limit 10;
```



7.1.3 Now restore the table:

```
undrop table json_weather_data;
```

7.1.4 The json_weather_data table should be restored.

7.2 Roll Back a Table

Now let's look rolling back a table to a previous state to fix an unintentional DML error that replaces all the station names in the Citibike database TRIPS table with the word "oops."

7.2.1 First make sure the worksheet is in the proper context:

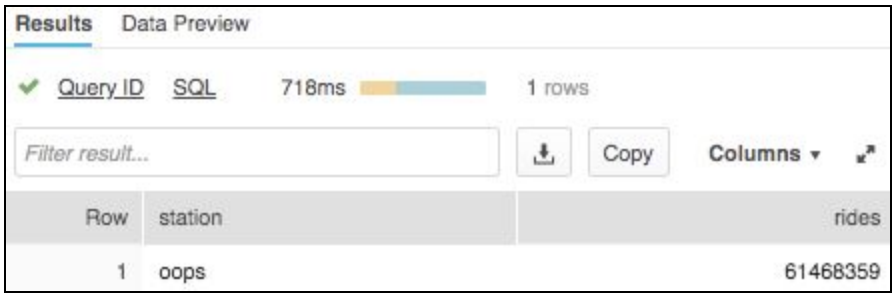
```
use role sysadmin;
use warehouse compute_wh;
use database citibike;
use schema public;
```

7.2.2 Then run the following command that replaces all the station names in the table with the word "oops".

```
update trips set start_station_name = 'oops';
```

7.2.3 Now run a query that returns the top 20 stations by # of rides - notice how we've screwed up the station names so we only get one row:

```
select
start_station_name as "station",
count(*) as "rides"
from trips
group by 1
order by 2 desc
limit 20;
```



7.2.4 Normally, we would need to scramble and hope we have a backup lying around. But in Snowflake, we can simply run commands to find the query ID of the last UPDATE command & store it in a variable called $QUERY_ID…

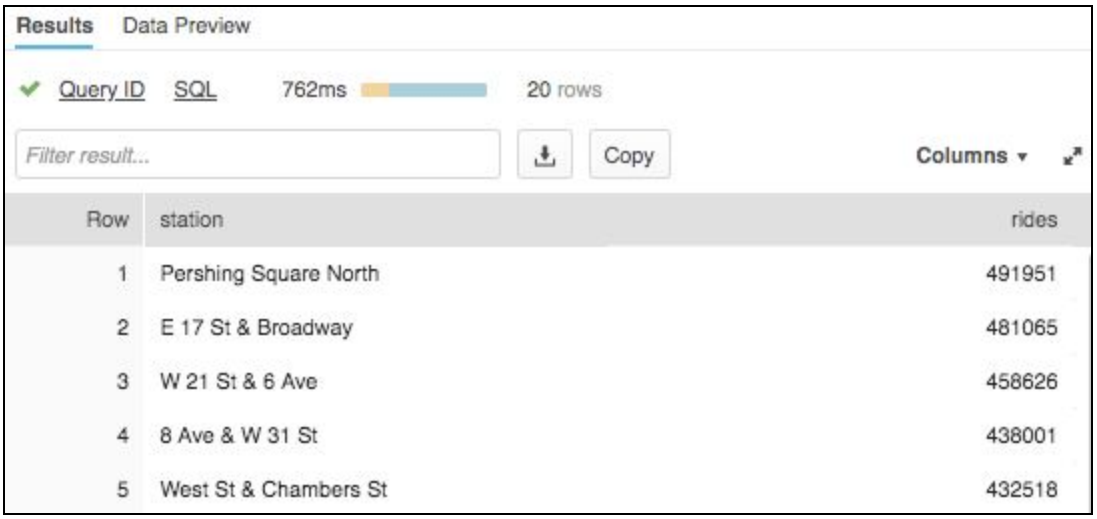
```
set query_id =
(select query_id from
table(information_schema.query_history_by_session (result_limit=>5))
where query_text like 'update%' order by start_time limit 1);
```

7.2.5 Then re-create the table as of before the update:

```
create or replace table trips as
(select * from trips before (statement => $query_id));
```

7.2.6 Run the SELECT statement again to check that the station names have been restored:

```
select
start_station_name as "station",
count(*) as "rides"
from trips
group by 1
order by 2 desc
limit 20;
```



Row	station	rides
1	Pershing Square North	491951
2	E 17 St & Broadway	481065
3	W 21 St & 6 Ave	458626
4	8 Ave & W 31 St	438001
5	West St & Chambers St	432518

Module 8: Roles Based Access Controls and Account Admin

In this module we will show some aspects of Snowflake roles based access control (RBAC), including creating a new role and granting it specific permissions. We will also cover the ACCOUNTADMIN (aka Account Administrator) role.

To continue with the Citi Bike story, let's assume a junior DBA has joined Citi Bike and we want to create a new role for them with less privileges than the system-defined, default role of SYSADMIN. Let's now do that.

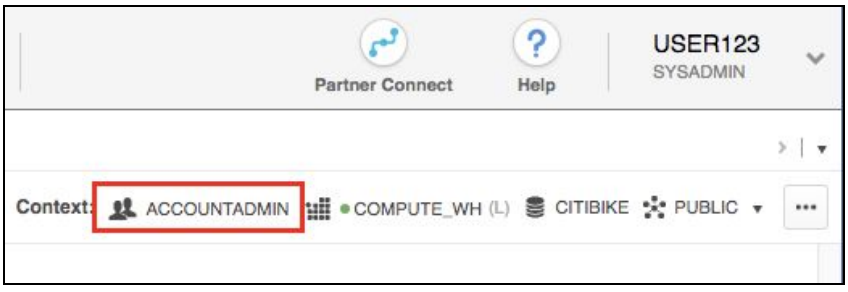
Roles-Based Access Control (RBAC)
Snowflake offers very powerful and granular RBAC which can control what objects and capabilities a role or user can access, and what level of access they have. For more detail, see the documentation at https://docs.snowflake.net/manuals/user-guide/security-access-control.html

8.1 Create New Role and Add User to it

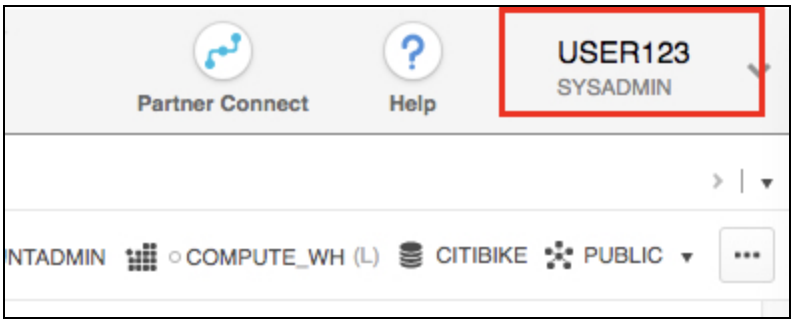
8.1.1 In the worksheet let's switch to the ACCOUNTADMIN role to create a new role. This role encapsulates the SYSADMIN and SECURITYADMIN system-defined roles. It is the top-level role in the system and should be granted only to a limited/controlled number of users in your account. In the worksheet, run:

```
use role accountadmin;
```

When done, notice at the top right of the worksheet, the worksheet context has changed so now the role is ACCOUNTADMIN



8.1.2 In order for any role to function, we need at least one user assigned to it. So let's create a new role called "junior_dba" and assign your user name to it. This is the user name you created when you first opened your 30-day free trial Snowflake account. This name also appears at the top right of the UI. In the screenshot below it is "USER123". Of course yours will be different. Make a note of your user name.



8.1.3 Let's now create the role and add a user to it with your unique user name:

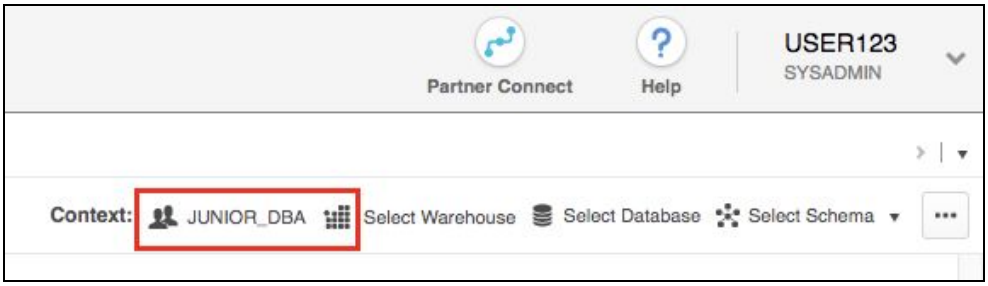
```
create role junior_dba;
grant role junior_dba to user YOUR_USER_NAME_GOES HERE;
```

NOTE - if you tried to perform this operation while in a role like SYSADMIN, it would fail due to insufficient privileges as the SYSADMIN role by default cannot create new roles or users.

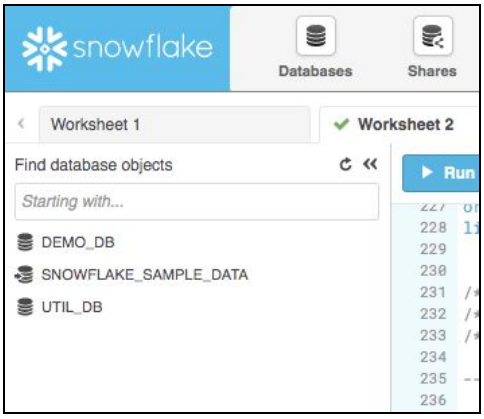
8.1.4 Change your worksheet context to the new junior_dba role

```
use role junior_dba;
```

At the top right of the worksheet, note that the context has changed to reflect the junior_dba role



8.1.5 On the left side of the UI in the database object browser pane, notice that both the Citibike and Weather databases do not appear. This is because the junior_dba role does not have access to view them.

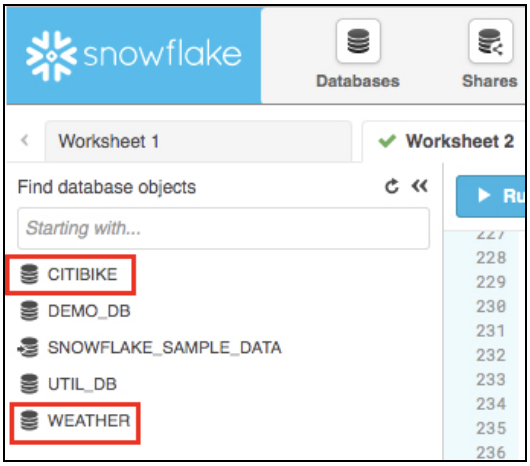


8.1.6 Let's switch back to the ACCOUNTADMIN role and grant the junior_dba the ability to view and use the CITIBIKE and WEATHER databases

```
use role accountadmin;
grant usage on database citibike to role junior_dba;
grant usage on database weather to role junior_dba;
```

8.1.7 Switch to the junior_dba role and at the left in the database object browser, note the Citibike and Weather databases now appear. Click the refresh icon if they do not appear.

```
use role junior_dba;
```

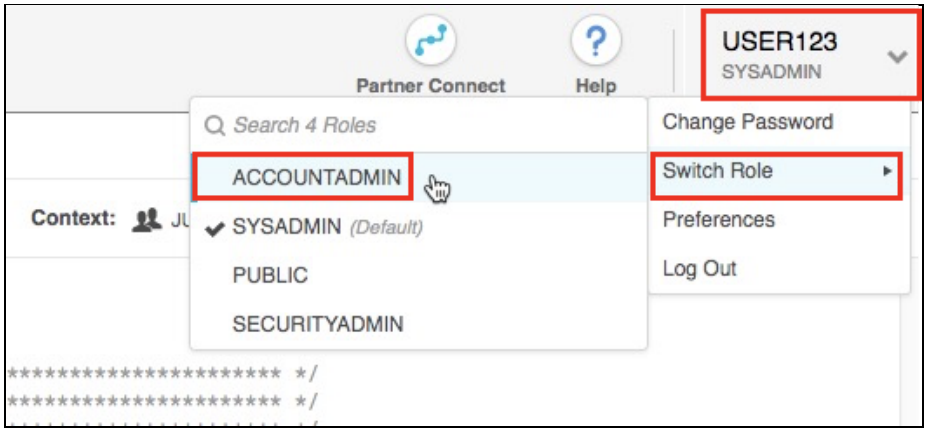


8.2 Account Administrator View

Let's change our security role for the session to ACCOUNTADMIN to see other parts of the UI only this role can see.

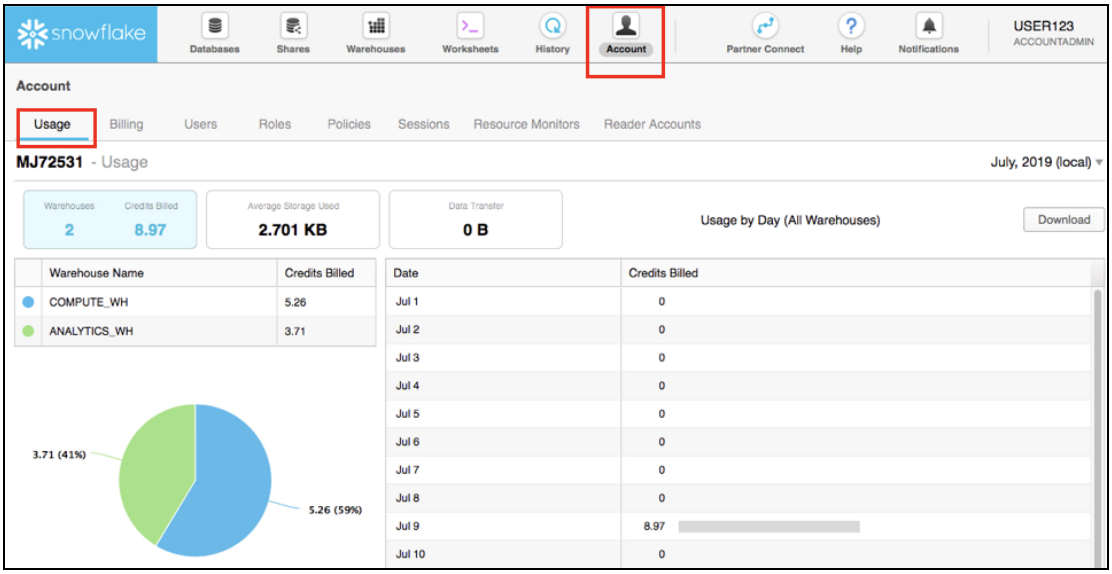
Roles in User Preference vs Worksheet
We just changed the security role for the session in the user preference menu at the top right of the UI. This changes what we can see in the UI. This is different then the worksheet context menu where we assign a role that is applied to the commands run on that specific worksheet. Also, session security role can simultaneously be different from the role used in a worksheet.

8.2.1 In the top right corner of the UI, click on your user name to show the User Preferences menu. Then go to Switch Role, then select the ACCOUNTADMIN role.



8.2.2 Notice at the very top of the UI you will now see a sixth tab called “Account” that you can only view in the ACCOUNTADMIN role.

Click on this Account tab. Then towards the top of this page click on “Usage” which by default already appearsI. Here you see detail on credits, storage, and daily usage.



8.2.3 To the right of “Usage” is “Billing” where you can add a credit card if you want to go beyond your free $400 worth of credits for this free trial. Further to the right is information on Users, Roles, and Resource Monitors. The latter set limits on your account's credit consumption so you can appropriately monitor and manage credit consumption.

NOTE - Stay in the ACCOUNTADMIN role for the next module.

Module 9: Data Sharing

Snowflake enables account-to-account sharing of data through *shares*, which are created by data providers and "imported" by data consumers, either through their own Snowflake account or a provisioned Snowflake Reader account. The consumer could be an external entity/partner, or a different internal business unit which is required to have its own, unique Snowflake account.

With Data Sharing –

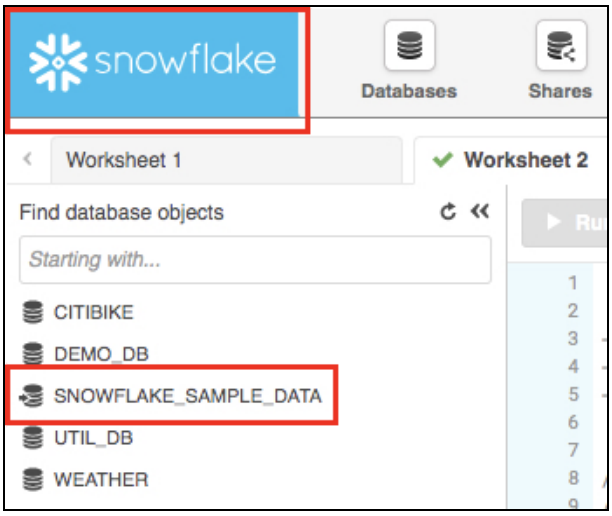
- There is only one copy of data, which lives in the data provider's account
- Shared data is always live, real-time and immediately available to consumers
- Providers can establish revocable, fine-grained access grants to shares
- Data sharing is simple and secure, especially compared to the "old" way of sharing data which was often manual and involved transferring large .csv across the Internet in a manner that might be insecure

  Note - Data Sharing currently only supported between accounts in the same Snowflake Provider and Region

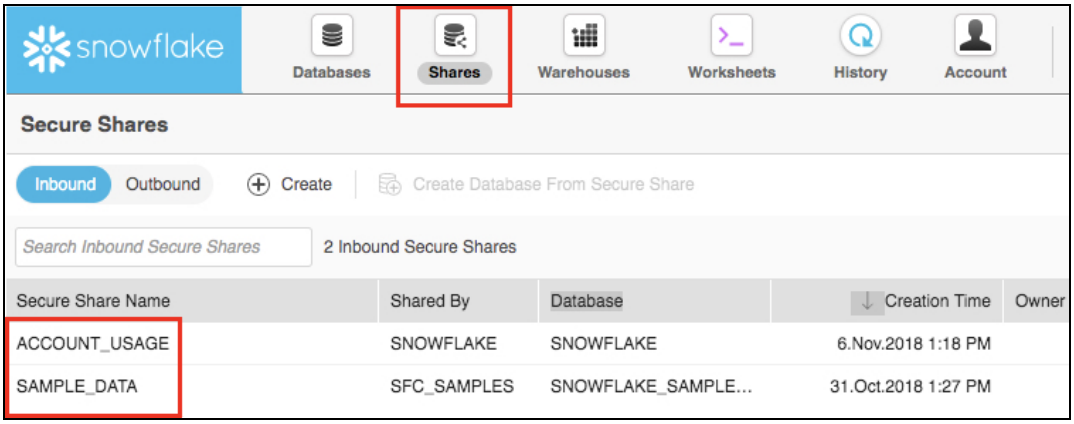
One example of data sharing is that Snowflake uses secure data sharing to share account usage data and sample data sets with all Snowflake accounts. In this capacity, Snowflake acts as the provider of the data and all other accounts act as the consumers. In your Snowflake environment you can easily see this and we walk through this in the next section.

9.1 See Existing Shares

9.1.1 Click on the blue Snowflake logo at the very top left of the UI. On the left side of the UI in the database object browser, notice the database “SNOWFLAKE_SAMPLE_DATA”, The small arrow on the database icon indicates this is a share.



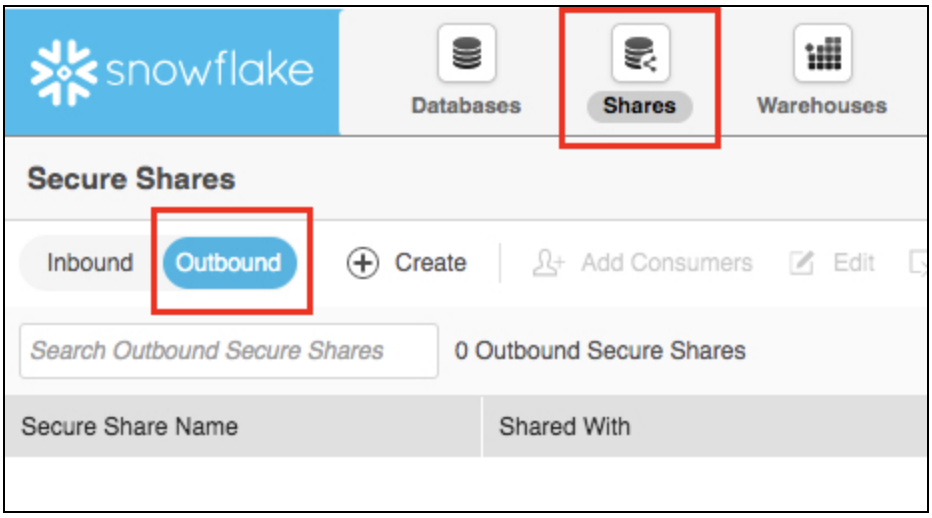
9.1.2 At the top right of the UI verify you are in the ACCOUNTADMIN role. Then at the top of the UI click on the Shares tab. Notice on this page you are looking at your Inbound Secure Shares and there are two shares shared by Snowflake with your account. One contains your account usage and the other has sample data you can use. This is data sharing in action - your Snowflake account is a consumer of data shared/provided by Snowflake!



9.2 Create an Outbound Share

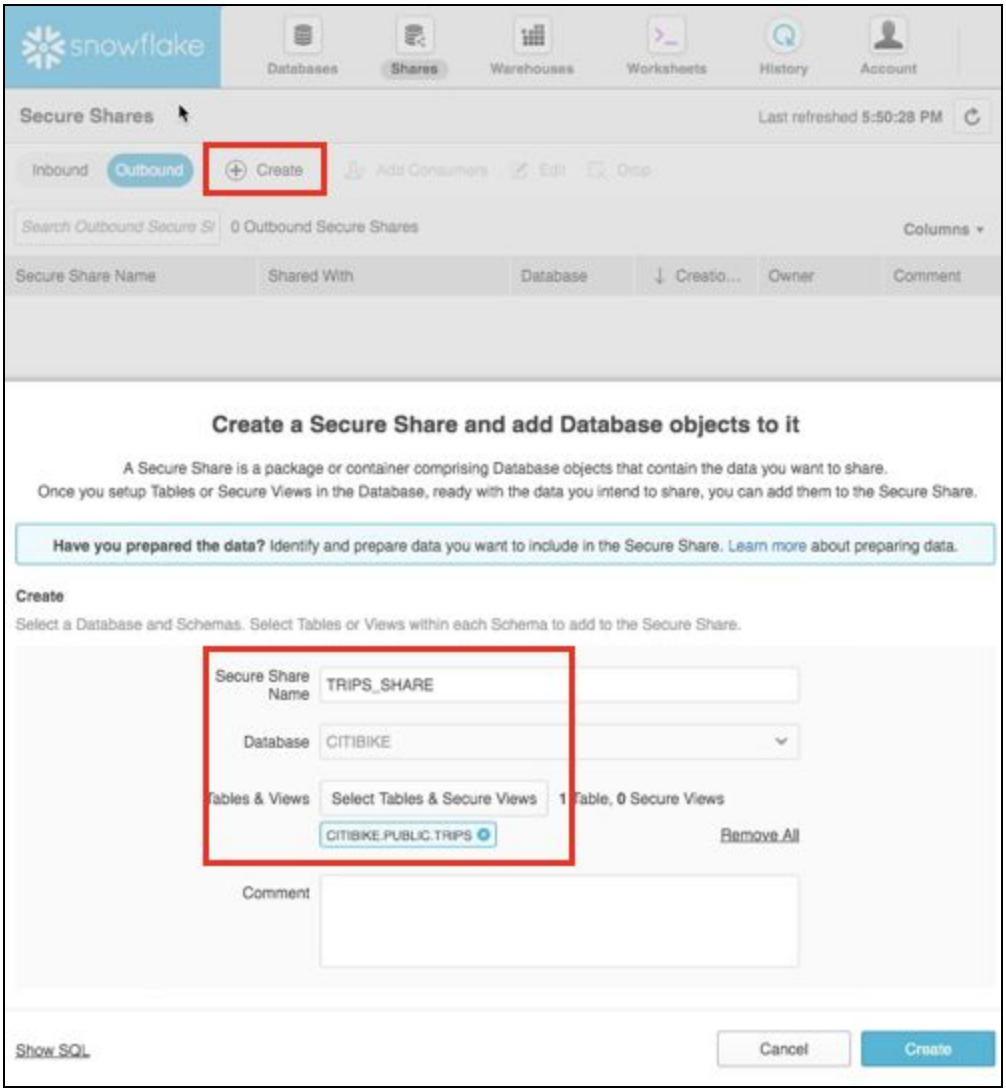
9.2.1 Let's go back to the Citi Bike story and assume we are the Account Administrator for Snowflake at Citi Bike. We have a trusted partner who wants to do perform data science on the data in our TRIPS database on a near real-time basis to further analyze it. This partner also has their own Snowflake account in our region. So let's use Snowflake Data Sharing to share this data with them so they can analyze it.

At the top of the UI click on the Shares tab. Then, further down on the page click on the "Outbound" button.



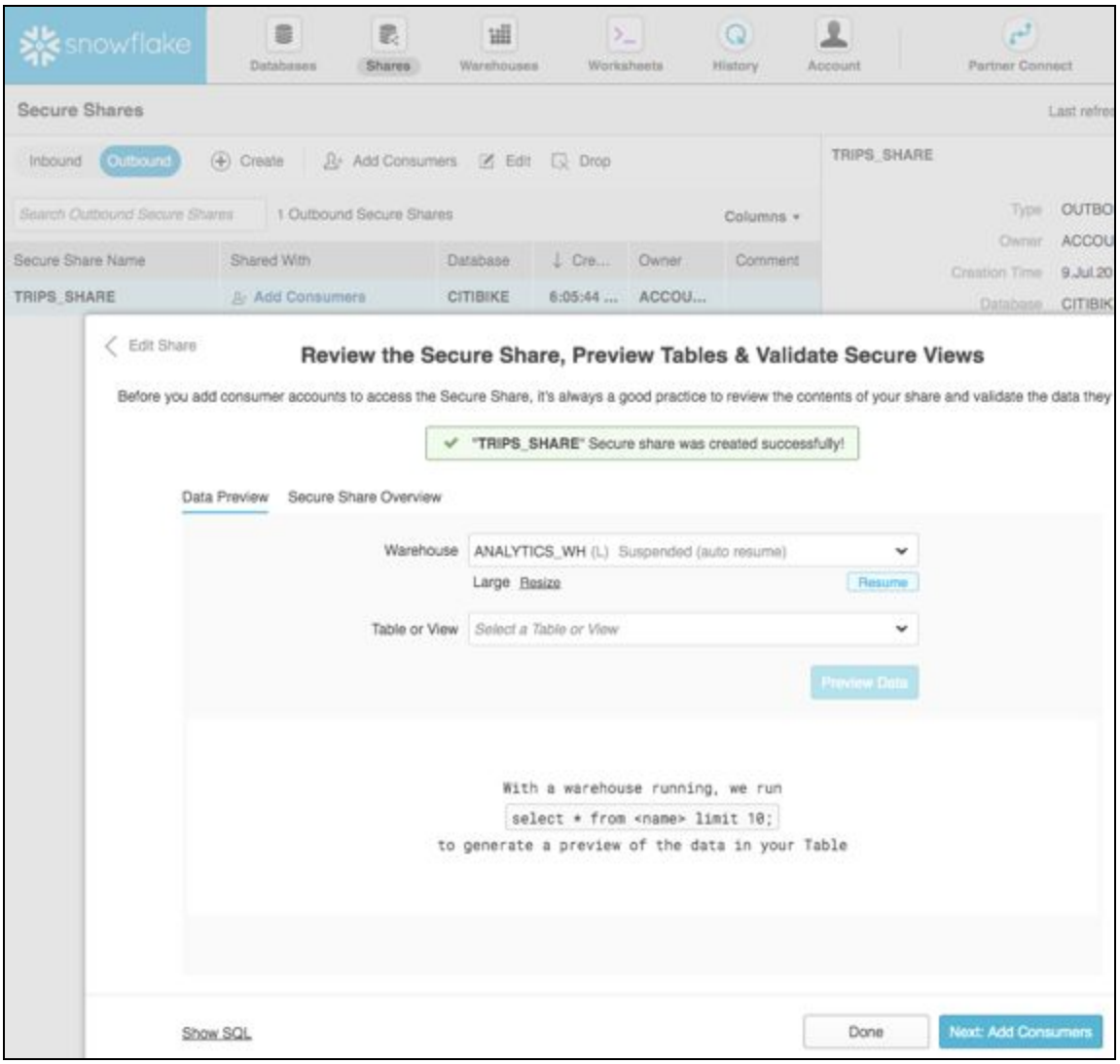
9.2.2 Click on the "Create" button and in the fields that appear, fill them out as shown below.

- For "Secure Share Name" enter "TRIPS_SHARE"
- For "Database" you will use the drop-down to select "CITIBIKE"
- For "Tables & Views" you will use the database object browser to browse to CITIBIKE > PUBLIC > TRIPS.
- Click on the blue "Apply" button



9.2.3 Click on the blue “Create” button at the bottom of the box.

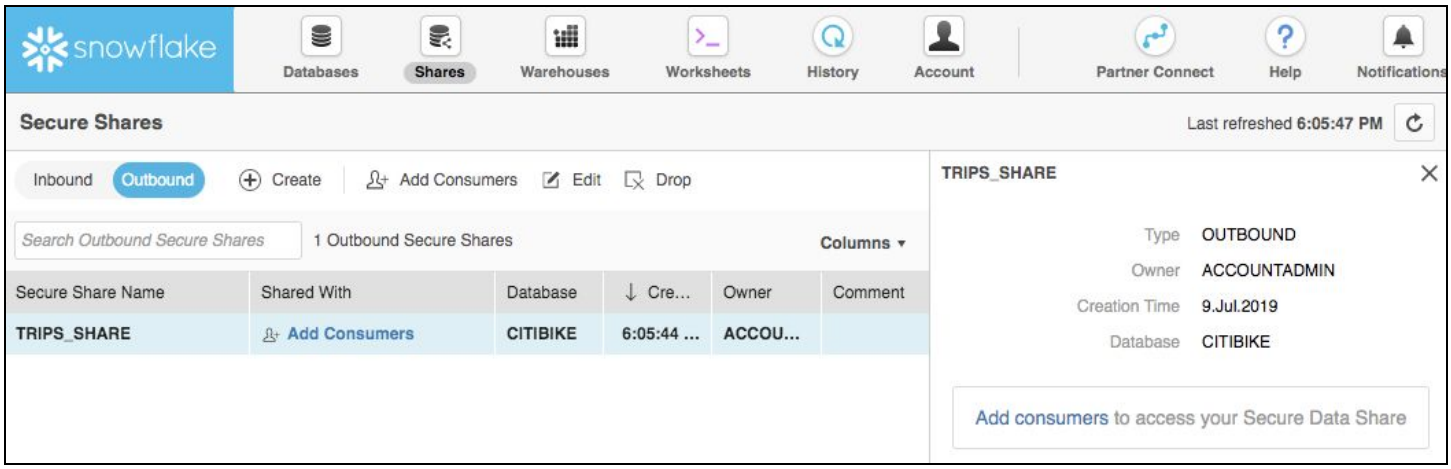
Note the window indicates the Secure share was created successfully.



In the real-world, the Citi Bike Account Administrator would click on the “Next: Add Consumers” blue button to add information on their partner’s Snowflake account name and type. But since in the lab we are just using our own account, we will stop here.

9.2.4 Click on the “Done” button at the bottom of the box.

Note this page now shows the “TRIPS_SHARE” secure share. It only took seconds to give other accounts access to data in Snowflake in a secure manner with no copies of the data having to be made!



Lastly, note that Snowflake provides several ways to securely share data without compromising confidentiality. You can share not only tables and views, but also Secure Views, Secure UDFs (User Defined Functions), and Secure Joins. For more details on how to use these methods for sharing data while preventing access to sensitive information, see the Snowflake documentation.

Congratulations, you are now done with this lab! Let's wrap things up in the next, and final, section.

Summary & Next Steps

This tutorial was designed as a hands-on introduction to Snowflake to simultaneously teach you how to use it, while showcasing some of its key capabilities and differentiators. We covered how to navigate the UI, create databases and warehouses, load & query structured and semi-structured data, perform zero-copy cloning, undo user errors, RBAC, and data sharing.

We encourage you to continue with your free trial by loading in your own sample or production data and by using some of the more advanced capabilities of Snowflake not covered in this lab. There are several ways Snowflake can help you with this:

- At the very top of the UI click on the “Partner Connect” icon to get access to trial/free ETL and BI tools to help you get more data into Snowflake and then analyze it
- Read the “Definitive Guide to Maximizing Your Free Trial” document at: https://www.snowflake.com/test-driving-snowflake-the-definitive-guide-to-maximizing-your-free-trial/
- Attend a Snowflake virtual or in-person event to learn more about our capabilities and how customers use us https://www.snowflake.com/about/events/
- Contact Sales to learn more https://www.snowflake.com/free-trial-contact-sales/

Resetting Your Snowflake Environment

Lastly, if you would like to reset your environment by deleting all the objects created as part of this lab, run the SQL below in a worksheet.

Run this SQL to set the worksheet context:

```
use role accountadmin;
use warehouse compute_wh;
use database weather;
use schema public;
```

Then run this SQL to drop all the objects we created in the lab:

```
drop share if exists trips_share;
drop database if exists citibike;
drop database if exists weather;
drop warehouse if exists analytics_wh;
drop role if exists junior_dba;
```

TIME TRAVEL

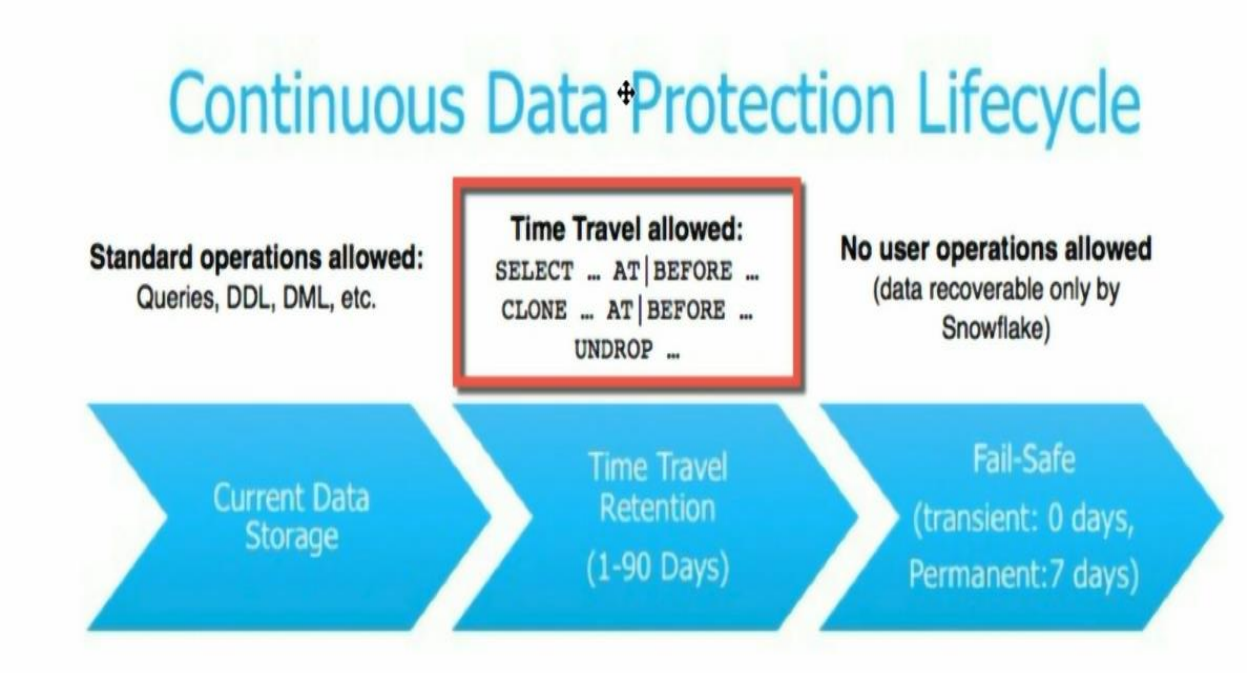
Time Travel:

- It is the feature which allows a user to access the data back in history with a defined period - **RETENTION PERIOD ~ 1 - 90 days, Default 1**
- Time travel helps to protect data from loss of data due to (intentional or unintentional) update or delete operations.
- Protects from data errors resulting during various operations (writing, processing, batch execution,production installs) due to which unwanted changes can also go in the database.

- Features:
    - Can go in the past and retrieve/select data via
        - Exact timestamp in the past
        - Using offset e,g in last few hours
        - Using query ID. example - get the data before a query execution.
        - Select data using time travel **AT | BEFORE**
        - Clone data using time travel **AT | BEFORE**

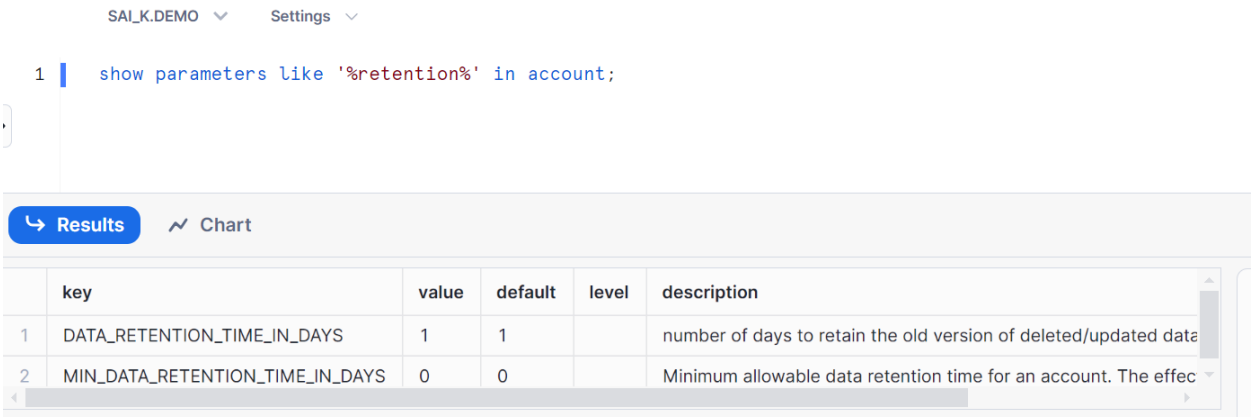
    - Restores tables, schemas and databases.
        - UNDROP

How Time Travel works?



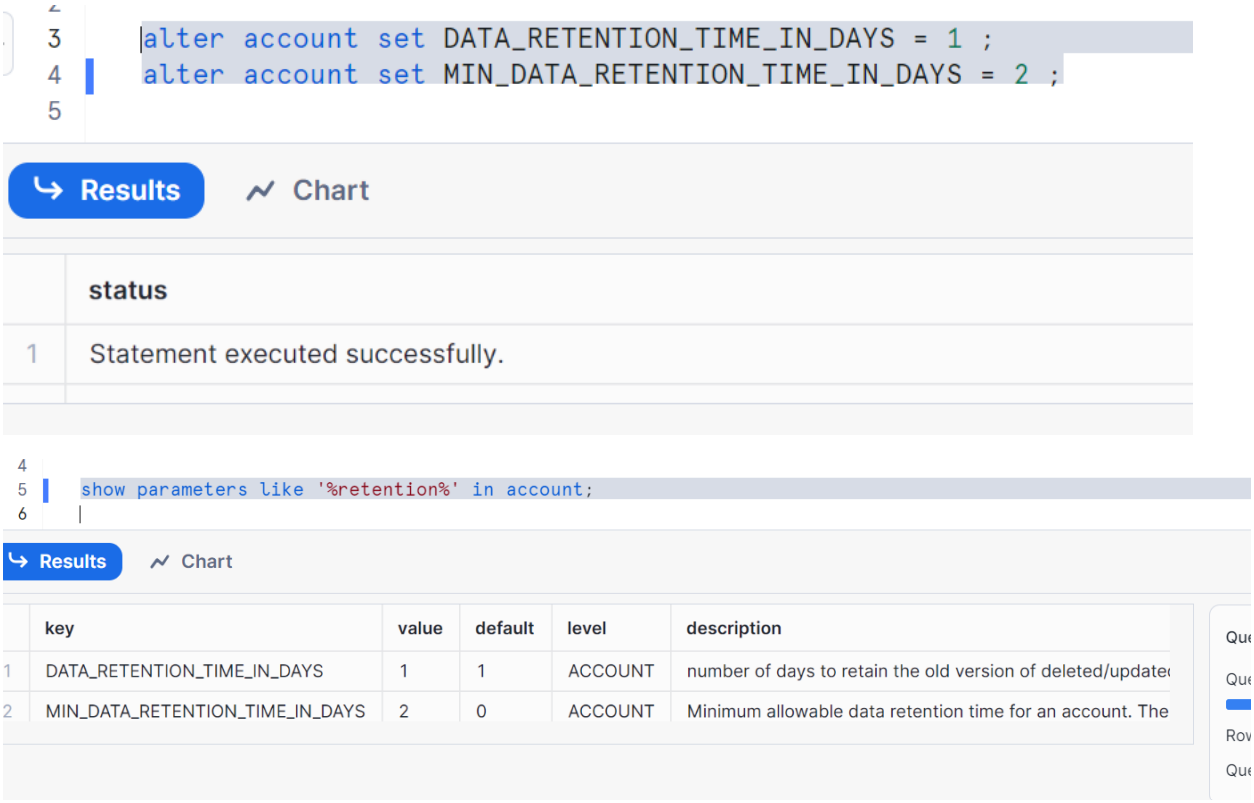
Let’s understand in practical step by step

Please read the description in account level as shown below to understand better



Default is 1 and 0, Value can be 1 to 90 days

We can alter as well



Let’s check in DB level

```
show parameters like '%retention%' in database;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	1	1	ACCOUNT	number of days to retain the old version of deleted/updated dat

Only DATA_RETENTION_TIME_IN_DAYS is in account level implies

MIN_ DATA_RETENTION_TIME_IN_DAYS can’t be altered in DB, let’s check

```
alter database set DATA_RETENTION_TIME_IN_DAYS = 5 ;
```

Results | Chart

	status
1	Statement executed successfully.

```
alter database set MIN_DATA_RETENTION_TIME_IN_DAYS = 1 ;
```

Results | Chart

SQL access control error:

Insufficient privileges to operate on database 'SAI_K'

```
show parameters like '%retention%' in database;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	5	1	DATABASE	number of days to retain the old version of deleted/updated dat

Let’s see for schemas

```
13  show parameters like '%retention%' in schema;
14
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	5	1	DATABASE	number of days to retain the old version of deleted/updated dat

Here, in schema level also applies the same as that of DB

```
15  alter schema set DATA_RETENTION_TIME_IN_DAYS = 6 ;
16
```

Results | Chart

	status
1	Statement executed successfully.

```
17  alter schema set MIN_DATA_RETENTION_TIME_IN_DAYS = 1 ;
```

Results | Chart

SQL access control error:

Insufficient privileges to operate on schema 'DEMO'

Effective retention time =

Max (data_retention_time_in_days, min_data_retention_time_in_days)

Let's retrieve data using 'at' & 'before':

Firstly, let's create a table

```
create or replace table income_band as select * from SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL.INCOME_BAND ;
```

Results | Chart

	status
1	Table INCOME_BAND successfully created.

```
select * from income_band;
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000
7	7	60001	70000

Query Details
Query duration 130ms
Rows 20
Query ID 01b49acc-0000-a6a6-...

IB_INCOME_BAND_SK

Create another table named client

```
create or replace table client
(  id NUMBER(38,0), first_name VARCHAR(16), last_name VARCHAR(50),
   sex VARCHAR(1), ethinicity VARCHAR(30), ssn VARCHAR(15),
   street_address VARCHAR(90),status VARCHAR(10)
);
```

Results | Chart

	status
1	Table CLIENT successfully created.

```
insert into client values (111111, 'James',     'Schwartz', 'M', 'American','342-76-9087','5676 Washington Street','ACTIVE') ;
insert into client values (222222, 'Jessica',   'Escobar',  'F', 'Hispanic','456-93-5629','3234 WateringCan Drive','INACTIVE') ;
insert into client values (333333, 'Ben',       'Hardy',    'M', 'American','876-98-3245','6578 Historic Circle','INACTIVE') ;
insert into client values (444444, 'Anjali',     'Singh',   'F', 'Indian American','435-87-6532','8978 Autumn Day Drive','ACTIVE');
insert into client values (555555, 'Dean',      'Tracy',    'M', 'African','767-34-7656','2343 India Street','ACTIVE') ;
```

Results | Chart

	number of rows inserted
1	1

Query Details
Query duration 294ms

```
select * from client ;
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	ACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	ACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	ACTIVE

Query Details
Query duration 106ms
Rows 5
Query ID 01b49ad1-0000-a6a6-...

ID

```
show parameters like '%time%' ;
```

Our time zone is America/los_angeles

Results | Chart

key	value	default	level	description
TIMESTAMP_OUTPUT_FORMAT	YYYY-MM-DD HH24:MI:SS.	YYYY-MM-DD		Default display format for all tin
TIMESTAMP_TYPE_MAPPING	TIMESTAMP_NTZ	TIMESTAMP_N		If TIMESTAMP type is used, wh
TIMESTAMP_TZ_OUTPUT_FORMAT				Display format for TIMESTAMP.
TIMEZONE	America/Los_Angeles	America/Los_A		time zone
TIME_INPUT_FORMAT	AUTO	AUTO		input format for time
TIME_OUTPUT_FORMAT	HH24:MI:SS	HH24:MI:SS		display format for time

Query De
Query du
Rows
Query ID
key
100% fill

Before performing any accidental operations, first copy your current timestamp aside

```
select current_timestamp() ;
```

Results | Chart

	CURRENT_TIMESTAMP()
1	2024-05-27 01:57:47.217 -0700

Let’s perform accidental delete i.e. wanted to delete > 121000 but mistakenly we deleted ib_lower_bound>101000 accidentally

```
delete from income_band where ib_lower_bound > 101000;
```

Results | Chart

	number of rows deleted
1	9

Copy the query id of above in your notepad

```
delete from income_band where ib_lower_bound > 101000;
```

Results | Chart

	number of rows deleted
1	9

Query Details
Query duration
Rows
Query ID
Copy Query ID
View Query Profile
number of rows deleted
100% filled

Another accidental operation i.e. accidental update, should have been only for Id=222222 i.e. we forgot to set for Id 222222 in table client

```
update client set status = 'INACTIVE' ;
```

Results | Chart

	number of rows updated	number of multi-joined rows updated
1	5	0

Query Details
Query duration 279ms
Rows 1
Query ID 01b49adb-0000-a6a6-...

Copy the Query ID of above

Now, see the data changes in table i.e. current status

```
select * from income_band ;
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000

Query Details
Query duration 146ms
Rows 11
Query ID 01b49ade-0000-a6ed-...
IB_INCOME_BAND_SK

```
select * from client ;
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	INACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	INACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	INACTIVE

Query Details
Query duration 105ms
Rows 5
Query ID 01b49ade-0000-a6a6-...

ID

Now, we can query using time travel BEFORE | AT

Paste your timestamp

```
select * from client before (timestamp => '2024-05-27 02:10:04.503 -0700'::timestamp_ltz);
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	ACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	ACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	ACTIVE

Query Details
Query duration 57ms
Rows 5
Query ID 01b49ae8-0000-a6ed-...

ID

Shows 5 rows with correct status as that of without accidental operations

```
select * from income_band before (timestamp => '2024-05-27 02:10:04.503 -0700'::timestamp_ltz);
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000
7	7	60001	70000

Query Details
Query duration 47ms
Rows 20
Query ID 01b49aec-0000-a6ed-...

IB_INCOME_BAND_SK

Similarly check for the client, you should get all 20 rows

Now, **using BEFORE via query ID**

Copy your query id as shown

```
select * from income_band before (statement => '01b49ae6-0000-a6ed-0000-0007317a9219');
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000
7	7	60001	70000

Query Details
Query duration 82ms
Rows 20
Query ID 01b49af0-0000-a6a6-...

IB_INCOME_BAND_SK

We got our 20 rows

```
select * from client before (statement => '01b49ae6-0000-a6a6-0000-0007317aa049');
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	ACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	ACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	ACTIVE

Query Details
Query duration 76ms
Rows 5
Query ID 01b49af2-0000-a6ed-0...

ID

We got our 5 rows

Another method using offset

Get the seconds first i.e. difference in seconds of current time and time when we had not accidental operations

```
select datediff(second, '2024-06-03 10:04:54.086 -0700'::timestamp_ltz, current_timestamp()) as seconds;
```

Results | Chart

	SECONDS
1	979

Query De
Query du
Rows

```
61
62  select * from client at(offset => -979);
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	ACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	ACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	ACTIVE

Query Details

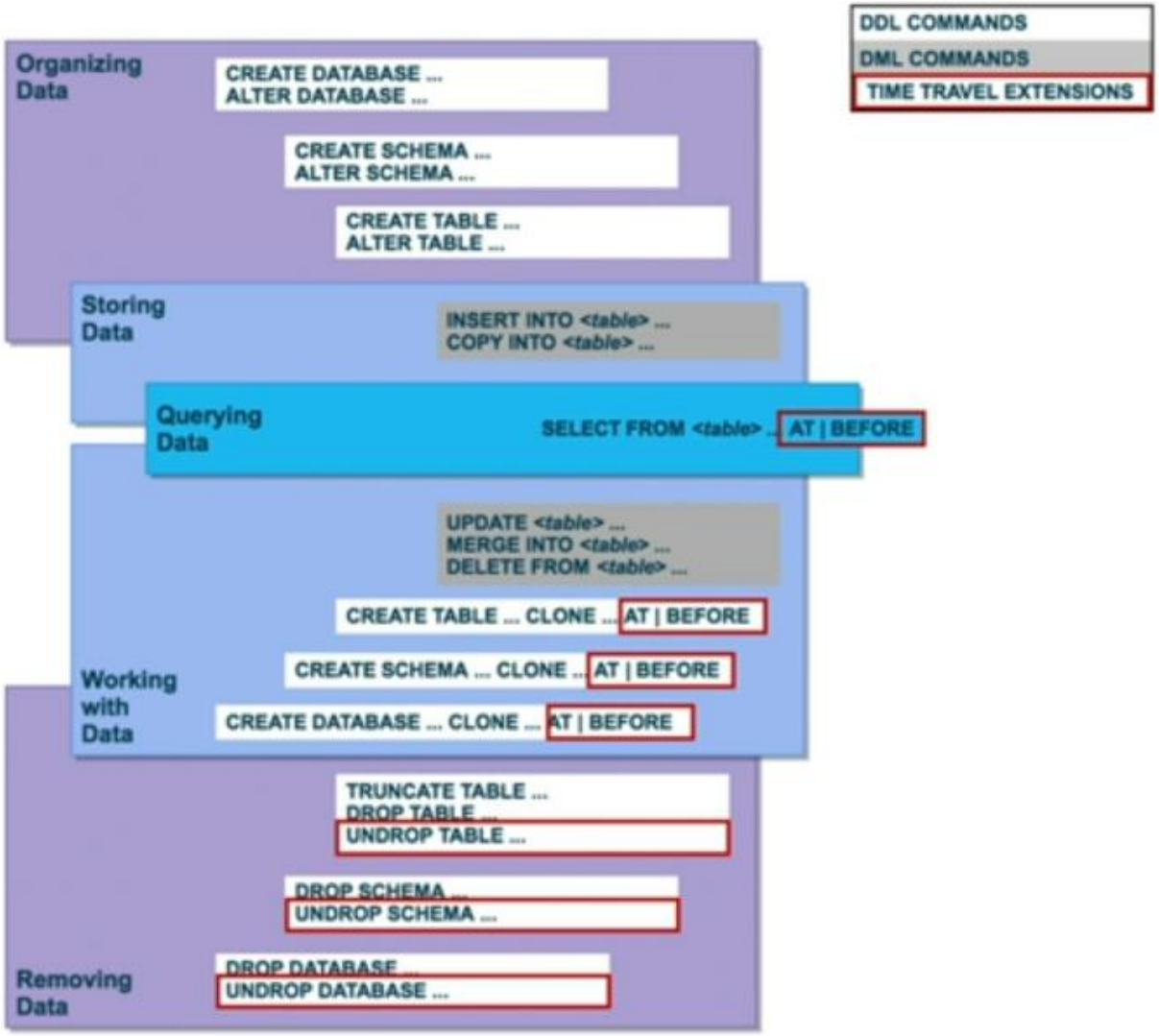
Query duration

Rows

Query ID 01b4c43

ID

Snowflake data lifecycle with time travel



Now let’s do accidental operation on client table i.e. first accidental update (query ID 1) and then accidental delete (query ID 2)

Now, Paste your query id 1 on client table so that you have to get the data after the update operation by checking the real table

```
select * from client before (statement => '01b4c43d-0000-a73b-0007-317a00010252');
```

Results | Chart

ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	ACTIVE
222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	ACTIVE
555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	ACTIVE

Query Details
Query duration
Rows
Query ID 01b4c43e-0000-a7
ID

```
select * from client ;
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	INACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	INACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	INACTIVE

Query Details
Query duration
Rows
Query ID 01b4c43e-0
ID

We had done delete and update implies second query Id pasted will give me the data before update operation but

Do the same for income_band

Data restoration (restoring your data)

1. Direct method
2. Indirect method

Direct data restoration:

Let’s do practical

I had done two accidental operations on client table as shown, I copied my query Id’s

```
update client set status = 'INACTIVE' ; --01b4c422-0000-a7ea-0000-0007317af0c1
delete from client;--01b4c448-0000-a73b-0007-317a00010286
```

Now, I want to get the data before my delete operation i.e. I had done update operation. So, my table need to have all status as inactive

Here is the syntax

Create or replace table client as select * from client before (statement => 'paste your query ID of delete operation here');

Now check the table

```
70  select * from client ;
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	STATUS
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	INACTIVE
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	INACTIVE
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	INACTIVE
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	INACTIVE
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	INACTIVE

Query Deta
Query dura
Rows
Query ID
ID

Now, I need to get the data before update operation i.e. my original data. Let's perform the same

```
68  create or replace table client as select * from client before (statement => '01b4c422-0000-a7ea-0000-0007317af0c1');
69
```

Results | Chart

Time travel data is not available for table CLIENT. The requested time is either beyond the allowed time travel period or before the object creation time.

Query Details
Query duration
Rows
Query ID 01b4c44

That means we cannot go back to first query ID because we have used create or replace table client for second ID as main table client is actually got recreated and it has lost time travel history.

Hence, direct method of creating or replacing a table has this particular limitation

If you are using it, you have to be 100% sure what particular time frame/query ID you are using and once you do that, you will not be able to use again for another period.

Hence to avoid this, we use indirect method

Indirect data restoration:

Let’s do in practical

```
create or replace table income_band as select * from SNOWFLAKE_SAMPLE_DATA.TPCDS_SF100TCL.INCOME_BAND ;
```

Results | Chart

	status
1	Table INCOME_BAND successfully created.

First accidental delete i.e. wanted to delete > 121000

```
delete from income_band where ib_lower_bound > 101000;
```

Results | Chart

	number of rows deleted
1	9

Query Details
Query duration 453ms
Rows 1
Query ID 01b4c45d-0000-a73b-...

number of rows deleted

Second accidental update i.e. shift the band by 100, accidentally shifted by 1000.

```
update income_band set ib_lower_bound = ib_lower_bound + 1000, ib_upper_bound = ib_upper_bound + 1000 ;
```

Results | Chart

	number of rows updated	number of multi-joined rows updated
1	11	0

Query Details
Query duration 330ms
Rows 1
Query ID 01b4c460-0000-a7ea-...

Creating a temporary table and pasted second query ID as shown

```
create or replace table income_band_temp as  select * from income_band before (statement => '01b4c460-0000-a7ea-0000-0007317af26d');
```

Results | Chart

	status
1	Table INCOME_BAND_TEMP successfully created.

Query Details
Query duration

Let’s check data

```
98 select * from income_band_temp ;
99
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000

I got my records which are not increased by 1000

Now delete the main table and restore from temp table

```
100 delete from  income_band;
101
102
103
104
105
```

Results | Chart

	number of rows deleted
1	11

```
102 insert into income_band select * from income_band_temp;
103
104
105
```

Results | Chart

	number of rows inserted
1	11

```
104 select * from income_band ;
105
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000

Query Details

Query duration 95ms

Rows 11

Query ID 01b4c468-0000-a73b-...

IB_INCOME_BAND_SK

We went back one step, now we realize that we need to go back further

Again we can build the temp table using first query, let's see

```
106  create or replace table income_band_temp as  select * from income_band before (statement => '01b4c45d-0000-a73b-0007-
     317a000102d6');
107
108
```

Results | Chart

	status
1	Table INCOME_BAND_TEMP successfully created.

Query Details
Query duration 576ms
Rows 1
Query ID 01b4c46c-0000-a7ea-...

Delete the main table

```
107
108  delete from  income_band ;
```

Results | Chart

	number of rows deleted
1	11

Query Details
Query duration 164ms
Rows 1
Query ID 01b4c46d-0000-a73b-...

Restore from temp table

```
110  insert into income_band select * from income_band_temp;
111
112
113
```

Results | Chart

	number of rows inserted
1	20

Query Details
Query duration 362ms
Rows 1

Check the data

```
113  select * from income_band ;
114
```

Results | Chart

	IB_INCOME_BAND_SK	IB_LOWER_BOUND	IB_UPPER_BOUND
1	1	0	10000
2	2	10001	20000
3	3	20001	30000
4	4	30001	40000
5	5	40001	50000
6	6	50001	60000

Query Details
Query duration 95ms
Rows 20
Query ID 01b4c46f-0000-a7ea-0...

IB_INCOME_BAND_SK

- The idea is you should not drop the table to have its data retention.
- The moment a table is dropped it loses time travel and is considered as a new object,
- Create or replace actually drops the object.

UNDROP:

To retain our tables, schemas, database which we dropped

Let's see in practical

```
115  show tables ;
116
117
```

Results | Chart

	created_on	name	database_name	schema_name	kind	comment	cluster_by
1	2024-06-03 10:47:15.478 -0700	CLIENT	SAI_K	DEMO	TABLE		
2	2024-06-03 11:04:27.261 -0700	INCOME_BAND	SAI_K	DEMO	TABLE		
3	2024-06-03 11:20:14.087 -0700	INCOME_BAND_TEMP	SAI_K	DEMO	TABLE		
4	2024-05-25 01:17:31.153 -0700	ITEMS	SAI_K	DEMO	TABLE		
5	2024-05-23 23:26:06.831 -0700	PROMOTIONS	SAI_K	DEMO	TABLE		
6	2024-05-23 23:26:42.247 -0700	PROMOTIONS_EXT	SAI_K	DEMO	TABLE		

Query Details
Query duration
Rows
Query ID 01b4
created_on

```
117  select * from promotions ;
118
```

Results | Chart

	P_PROMO_SK	P_PROMO_ID	P_START_DATE_SK	P_END_DATE_SK	P_ITEM_SK	P_COST	P_RESPONSE_TARG
1	2002	AAAAAAAACNHAAAAA	2450775	2450810	454459	1000.00	
2	2003	AAAAAAAADNHAAAAA	2450454	2450480	164948	1000.00	
3	2004	AAAAAAAAENHAAAAA	2450723	2450741	319846	1000.00	
4	2005	AAAAAAAAFNHAAAAA	2450615	2450649	185626	1000.00	
5	2006	AAAAAAAAGNHAAAAA	2450838	2450898	28927	1000.00	
6	2007	AAAAAAAAHNHAAAAA	2450845	2450904	396698	1000.00	

Query Details
Query duration 539ms
Rows 499
Query ID 01b4c483-0000-a73b-...
P_PROMO_SK

Let's drop it

```
119  drop table promotions ;
120
```

Results | Chart

	status
1	PROMOTIONS successfully dropped.

Please follow the syntax

```
show tables history ;
```

Results | Chart

	created_on	name	database_name	schema_name	kind	comment	cluster_by
1	2024-06-03 10:47:15.478 -0700	CLIENT	SAI_K	DEMO	TABLE		
2	2024-06-03 11:04:27.261 -0700	INCOME_BAND	SAI_K	DEMO	TABLE		
3	2024-06-03 11:20:14.087 -0700	INCOME_BAND_TEMP	SAI_K	DEMO	TABLE		
4	2024-05-25 01:17:31.153 -0700	ITEMS	SAI_K	DEMO	TABLE		
5	2024-05-23 23:26:42.247 -0700	PROMOTIONS_EXT	SAI_K	DEMO	TABLE		
6	2024-05-23 23:26:06.831 -0700	PROMOTIONS	SAI_K	DEMO	TABLE		

Query Details
Query duration 57ms
Rows 12
Query ID 01b4c484-0000-a73b-...

created_on

Visible, Hence can be undropped

```
undrop table promotions ;
```

Results | Chart

	status
1	Table PROMOTIONS successfully restored.

```
select * from promotions ;
```

Results | Chart

	P_PROMO_SK	P_PROMO_ID	P_START_DATE_SK	P_END_DATE_SK	P_ITEM_SK	P_COST	P_RESPONSE_TARG
1	2002	AAAAAAAACNHAAAAA	2450775	2450810	454459	1000.00	
2	2003	AAAAAAAADNHAAAAA	2450454	2450480	164948	1000.00	
3	2004	AAAAAAAAENHAAAAA	2450723	2450741	319846	1000.00	
4	2005	AAAAAAAAFNHAAAAA	2450615	2450649	185626	1000.00	
5	2006	AAAAAAAAGNHAAAAA	2450838	2450898	28927	1000.00	
6	2007	AAAAAAAAHNHAAAAA	2450845	2450904	396698	1000.00	

Query Details
Query duration 30ms
Rows 499
Query ID 01b4c486-0000-a7ea-...

P_PROMO_SK

For schemas

```
create or replace schema cln_demo clone demo ;
```

Results | Chart

	status
1	Schema CLN_DEMO successfully created.

```
131 show schemas ;
132
```

Results | Chart

	created_on	name	is_default	is_current	database_name	owner	cc
1	2024-06-03 11:51:14.990 -0700	CLN_DEMO	N	N	SAI_K	ACCOUNTADMIN	
2	2024-05-23 23:16:28.036 -0700	DEMO	N	Y	SAI_K	ACCOUNTADMIN	
3	2024-06-03 11:52:06.060 -0700	INFORMATION_SCHEMA	N	N	SAI_K		Vi
4	2024-05-23 23:16:12.831 -0700	PUBLIC	N	N	SAI_K	ACCOUNTADMIN	

Query Details
Query duration
Rows
Query ID
created_on

```
130
131 drop schema cln_demo ;
132
133
```

Results | Chart

	status
1	CLN_DEMO successfully dropped.

```
133 show schemas history ;
134
```

Results | Chart

	created_on	name	is_default	is_current	database_name	owner	cc
1	2024-06-03 11:51:14.990 -0700	CLN_DEMO	N	N	SAI_K	ACCOUNTADMIN	
2	2024-05-23 23:16:28.036 -0700	DEMO	N	Y	SAI_K	ACCOUNTADMIN	
3	2024-06-03 11:52:37.864 -0700	INFORMATION_SCHEMA	N	N	SAI_K		Vi
4	2024-05-23 23:16:12.831 -0700	PUBLIC	N	N	SAI_K	ACCOUNTADMIN	

Query Details
Query duration 37ms
Rows 4
Query ID 01b4c48c-0000-a73b-...
created_on

```
137
138 undrop schema CLN_DEMO ;
139
```

Results | Chart

	status
1	Schema CLN_DEMO successfully restored.

```
134 show schemas ;
135
```

Results | Chart

	created_on	name	is_default	is_current	database_name	owner	cc
1	2024-06-03 11:51:14.990 -0700	CLN_DEMO	N	N	SAI_K	ACCOUNTADMIN	
2	2024-05-23 23:16:28.036 -0700	DEMO	N	Y	SAI_K	ACCOUNTADMIN	
3	2024-06-03 11:55:13.349 -0700	INFORMATION_SCHEMA	N	N	SAI_K		Vi
4	2024-05-23 23:16:12.831 -0700	PUBLIC	N	N	SAI_K	ACCOUNTADMIN	

Similarly for databases as well

Retention time:

This is the main component of time travel. The time travel is achieved by using the value of this parameter

Account Level
Database Level
Schema Level
Object level

Default is 1 day and automatically enabled.
0 means time travel is disabled.

DATA_RETENTION_TIME_IN_DAYS
MIN_DATA_RETENTION_TIME_IN_DAYS

effective:
max(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS)

Standard	Enterprise	Business Critical	Virtual private
1 (Default), Can be set to 0 at account and object level Range 0 - 1	90 Range 0 - 90	90 Range 0 - 90	90 Range 0 - 90

Let’s do hands on

Creating new database and schema

```
140
141   create database training_rt ;
142   create schema demo_rt ;
```

Results | Chart

	status
1	Schema DEMO_RT successfully created.

```
144 show parameters like '%retention%' in account;
145
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	1	1	ACCOUNT	number of days to retain the old version of deleted/update
2	MIN_DATA_RETENTION_TIME_IN_DAYS	2	0	ACCOUNT	Minimum allowable data retention time for an account. The

```
145
146 show parameters like '%retention%' in database;
147
```

Results | Chart

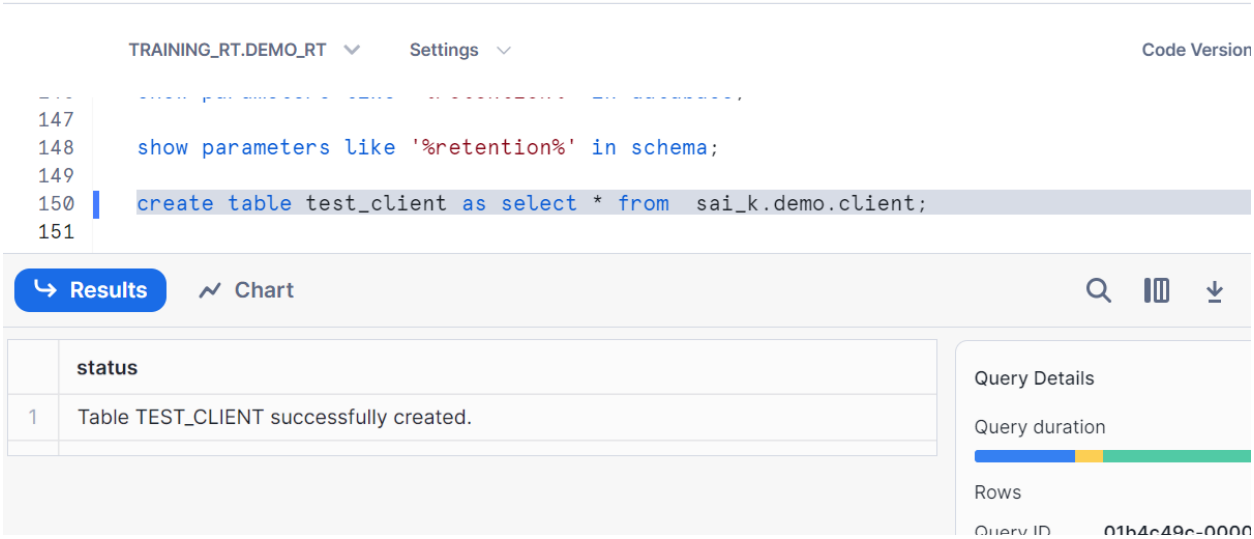
	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	1	1	ACCOUNT	number of days to retain the old version of deleted/updated dat

```
148 show parameters like '%retention%' in schema;
149
```

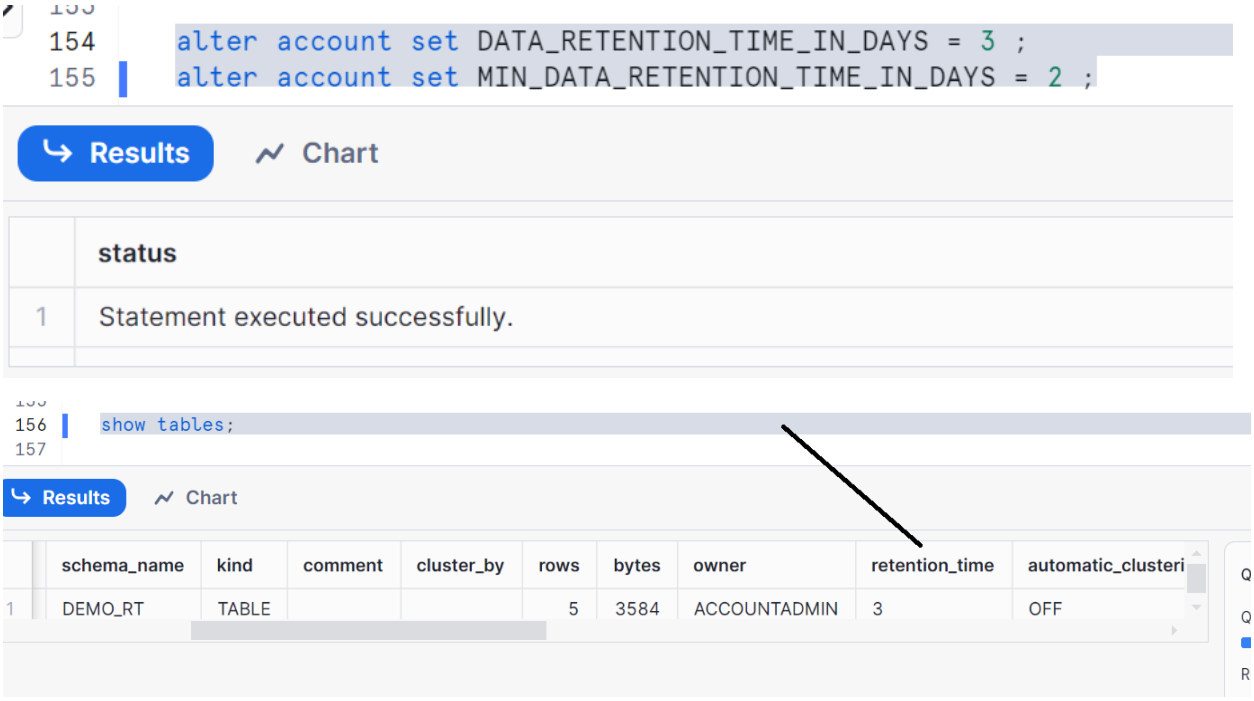
Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	1	1	ACCOUNT	number of days to retain the old version of deleted/updated dat

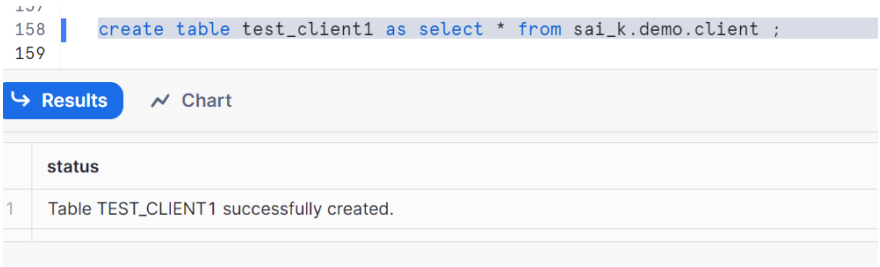
Let’s create a table in training_rt DB and demo_rt schema



Let’s alter



Creating new table and check



```
160  show tables ;
```

Results | Chart

	kind	comment	cluster_by	rows	bytes	owner	retention_time	au
1	TABLE			5	3584	ACCOUNTADMIN	3	OF
2	TABLE			5	3584	ACCOUNTADMIN	3	OF

Retention time is 3, taken from account

Now, change retention at DB level

```
161
162  alter database set DATA_RETENTION_TIME_IN_DAYS = 5 ;
163
```

Results | Chart

	status
1	Statement executed successfully.

```
164  show parameters like '%retention%' in account;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	3	1	ACCOUNT	number of days to retain the old version of deleted/update
2	MIN_DATA_RETENTION_TIME_IN_DAYS	2	0	ACCOUNT	Minimum allowable data retention time for an account. The

```
165
166  show parameters like '%retention%' in database;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	5	1	DATABASE	number of days to retain the old version of deleted/updated dat

Query Detail
Query durati
Rows

Overrides account, since modified at DB and flows to schema as there is no override at schema

```
show parameters like '%retention%' in schema;
```

Results | Chart

key	value	default	level	description
DATA_RETENTION_TIME_IN_DAYS	5	1	SCHEMA	number of days to retain the old version of deleted/updated data

Gets from database

```
create table test_client2 as select * from sai_k.demo.client ;
```

Results | Chart

status
Table TEST_CLIENT2 successfully created.

```
show tables;
```

Results | Chart

kind	comment	cluster_by	rows	bytes	owner	retention_time	automatic_clustering	change_tracki
TABLE			5	3584	ACCOUNTADMIN	5	OFF	OFF
TABLE			5	3584	ACCOUNTADMIN	5	OFF	OFF
TABLE			5	3584	ACCOUNTADMIN	5	OFF	OFF

5 for new table as well, 5 for all (prev)

Now let’s set at schema level

```
alter schema set DATA_RETENTION_TIME_IN_DAYS = 6 ;
```

Results | Chart

status
Statement executed successfully.

```
176 show parameters like '%retention%' in account;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	3	1	ACCOUNT	number of days to retain the old version of deleted/update
2	MIN_DATA_RETENTION_TIME_IN_DAYS	2	0	ACCOUNT	Minimum allowable data retention time for an account. The

Value is 3

```
177
178 show parameters like '%retention%' in database;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	5	1	DATABASE	number of days to retain the old version of deleted/updated dat

Value is 5

```
180 show parameters like '%retention%' in schema;
```

Results | Chart

	key	value	default	level	description
1	DATA_RETENTION_TIME_IN_DAYS	6	1	SCHEMA	number of days to retain the old version of deleted/updated data

Value is 6, schema override will flow to tables

```
181
182 show tables;
```

Results | Chart

	kind	comment	cluster_by	rows	bytes	owner	retention_time	automatic_clustering	change_tracki
1	TABLE			5	3584	ACCOUNTADMIN	6	OFF	OFF
2	TABLE			5	3584	ACCOUNTADMIN	6	OFF	OFF
3	TABLE			5	3584	ACCOUNTADMIN	6	OFF	OFF

Retention time is 6 for new and existing

Setting at table level

```
184  alter table test_client1 set DATA_RETENTION_TIME_IN_DAYS = 7 ;
185
```

Results | Chart

	status
1	Statement executed successfully.

```
186  alter table test_client2 set DATA_RETENTION_TIME_IN_DAYS = 8 ;
187
```

Results | Chart

	status
1	Statement executed successfully.

```
188  show tables;
```

Results | Chart

	kind	comment	cluster_by	rows	bytes	owner	retention_time	automatic_clustering	change_track
1	TABLE			5	3584	ACCOUNTADMIN	6	OFF	OFF
2	TABLE			5	3584	ACCOUNTADMIN	7	OFF	OFF
3	TABLE			5	3584	ACCOUNTADMIN	8	OFF	OFF

Setting up at create table level

```
190  create or replace table test_client3 (id number, name varchar(30)) data_retention_time_in_days=9;
191
```

Results | Chart

	status
1	Table TEST_CLIENT3 successfully created.

```
192 show tables;
```

Results | Chart

	e	schema_name	kind	comment	cluster_by	rows	bytes	owner	retention_time	automatic_cluste
1		DEMO_RT	TABLE			5	3584	ACCOUNTADMIN	6	OFF
2		DEMO_RT	TABLE			5	3584	ACCOUNTADMIN	7	OFF
3		DEMO_RT	TABLE			5	3584	ACCOUNTADMIN	8	OFF
4		DEMO_RT	TABLE			0	0	ACCOUNTADMIN	9	OFF

Increasing retention - it increases the time period for which the data is available in time travel => 5 -> 10, does not apply to data already in fail safe

Decreasing retention - it reduces the time period for which the data is available in time travel => 6 -> 4. Some data move to fail-safe

-- check the storage cost and different buckets

-- UI only shows Stage, Database and Fail Safe

-- Since time travel stores the data to be available it incurs the cost.

-- We can check the cost –bucket for time travel via UI as well -- as using SQL queries

-- check the storage cost and different buckets

Below query shows time travel bytes

```
194 select * from snowflake.account_usage.table_storage_metrics ;
```

Results | Chart

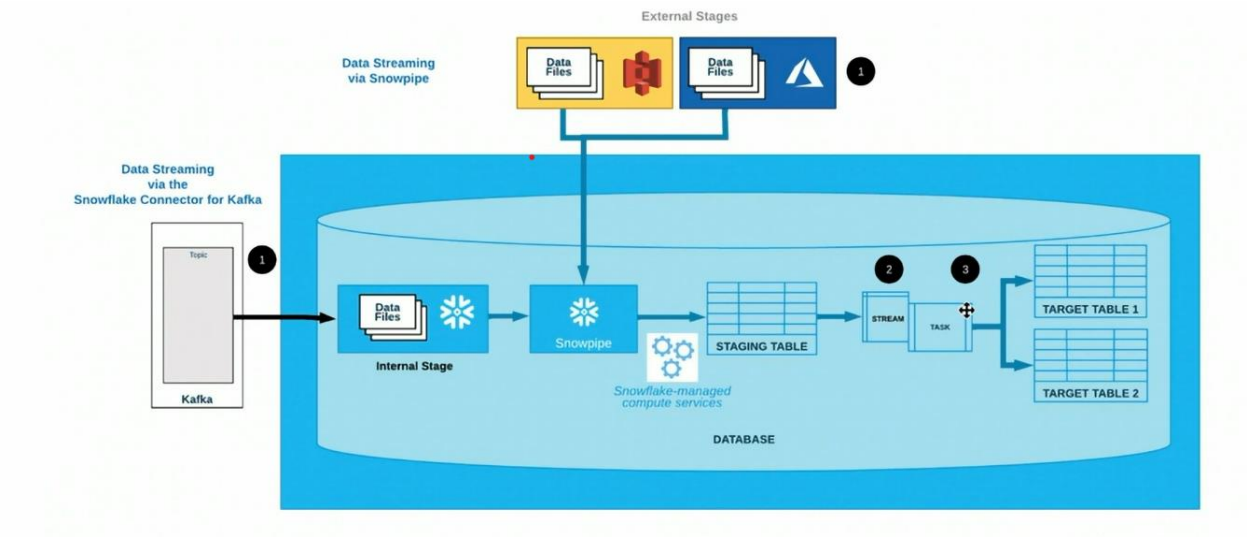
	ID	TABLE_NAME	TABLE_SCHEMA_ID	TABLE_SCHEMA	TABLE_CATALOG_ID	TABLE_CATALOG	CLONE_GR
1	5128	CLIENT	4	CLN_DEMO	4	SAI_K	
2	5129	INCOME_BAND	4	CLN_DEMO	4	SAI_K	
3	6146	CLIENT	3	DEMO	4	SAI_K	
4	5133	PROMOTIONS	4	CLN_DEMO	4	SAI_K	
5	5122	INCOME_BAND	3	DEMO	4	SAI_K	
6	5131	ITEMS	4	CLN_DEMO	4	SAI_K	

Note: fail safe will be explained in upcoming post

Thanks

STREAMS

Here is the representation of continuous data pipeline, streams come in the position of 2 as shown below



Let's perform what streams actually does by creating staging and target tables as per the above image

Let's create a staging table

```
create or replace table raw_applicant_staging
(  id number,
   first_name varchar,
   last_name varchar,
   sex varchar,
   ethinicity varchar,
   ssn varchar,
   street_address varchar,
   education_level varchar,
   years_of_experience number,
   job_id number
);
```

Creating job_details table

```
create or replace table job_details (
    job_id number,
    name varchar,
    city varchar,
    state varchar,
    education_level varchar
    ) ;
```

Results | Chart

	status
1	Table JOB_DETAILS successfully created.

```
insert into job_details values (10, 'Painter', 'Raleigh', 'NC','High School') ;
insert into job_details values (20, 'Software Engineer', 'Raleigh', 'NC','Masters') ;
insert into job_details values (30, 'Data Architect', 'Raleigh', 'NC','Undergrad') ;
insert into job_details values (40, 'Vice President', 'Raleigh', 'NC','Masters') ;
insert into job_details values (50, 'Associate', 'Raleigh', 'NC','Masters') ;
```

Results | Chart

	number of rows inserted
1	1

Query Details
Query duration 278ms
Rows 1

```
select * from job_details ;
```

Results | Chart

	JOB_ID	NAME	CITY	STATE	EDUCATION_LEVEL
1	10	Painter	Raleigh	NC	High School
2	20	Software Engineer	Raleigh	NC	Masters
3	30	Data Architect	Raleigh	NC	Undergrad
4	40	Vice President	Raleigh	NC	Masters
5	50	Associate	Raleigh	NC	Masters

Query Details
Query duration 107ms
Rows 5
Query ID 01b4ee7a-0000-a908-...

JOB_ID

Creating a target table

```
create or replace table candidates (
    id number,
    first_name varchar,
    last_name varchar,
    sex varchar,
    ethinicity varchar,
    ssn varchar,
    street_address varchar,
    candidate_education_level varchar,
    years_of_experience number,
    job_id number,
    job_name varchar,
    job_city varchar,
    job_state varchar,
    required_education_level varchar,
    status varchar,
    comments varchar
    ) ;
```

Inserting one record in our target table

```
56 insert into candidates
57 values (100000, 'James',     'Schwartz', 'M', 'American','342-76-9087','5676 Washington Street','High School', 5,10, 'Painter',
   'Raleigh', 'NC','High School','SHORTLISTED','The education level of candidate matched with job') ;
58
59
```

Results | Chart

	number of rows inserted
1	1

Query Details
Query duration 283ms
Rows 1

```
59 select * from candidates ;
60
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	CANDIDATE_EDUCAT
1	100000	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School

Query Details
Query duration 128ms
Rows 1

Now, let's create a stream as shown which is connected to staging table

```
61 create or replace stream strm_applicant1 on table raw_applicant_staging ;
62
```

Results | Chart

	status
1	Stream STRM_APPLICANT1 successfully created.

```
63 desc stream strm_applicant1 ;
64
```

Results | Chart

It is default stream, we can set to other as well which we can see later

stream will be staled over after 14 days

ype	base_tables	type	stale	mode	stale_after	invalid_reason
1	SAI_K.PUBLIC.RAW_APPLICANT_STAGING	DELTA	false	DEFAULT	2024-06-24 22:50:51.528 -0700	N/A

Query Details
Query duration 53ms
Rows 1

For now, just understand stale means 'not new'

```
66 show parameters like '%extension%' in account;
```

Results | Chart

	key	value	default	level	description
1	MAX_DATA_EXTENSION_TIME_IN_DAYS	14	14		Maximum number of days to extend data retention beyond the r

Query Detail
Query durati
Rows

Let’s insert some data in raw-applicant_staging table

```
68  insert into raw_applicant_staging values (111111, 'James',     'Schwartz', 'M', 'American','342-76-9087','5676 Washington
    Street','High School', 5,10) ;
69  insert into raw_applicant_staging values (222222, 'Jessica',   'Escobar',  'F', 'Hispanic','456-93-5629','3234 WateringCan
    Drive','Undergrad', 4, 10) ;
70  insert into raw_applicant_staging values (333333, 'Ben',       'Hardy',    'M', 'American','876-98-3245','6578 Historic
    Circle','Masters', 6, 30) ;
71  insert into raw_applicant_staging values (444444, 'Anjali',     'Singh',   'F', 'Indian American','435-87-6532','8978 Autumn Day
    Drive','Masters', 8,20) ;
72  insert into raw_applicant_staging values (555555, 'Dean',      'Tracy',    'M', 'African','767-34-7656','2343 India
    Street','Undergrad', 2,50) ;
73

74  select * from raw_applicant_staging ;
75
76
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LE\
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	Undergrad
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	Masters
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	Masters
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	Undergrad

Query Details
Query duration 93ms
Rows 5
Query ID 01b4ee8b-0000-a914-...

ID

Check stream and Observe extra columns

```
76  select * from strm_applicant1;
```

Results | Chart

as we had inserted in raw_applicant_staging

Is Update is false as we didn't update

	S_OF_EXPERIENCE	JOB_ID	METADATA$ACTION	METADATA$ISUPDATE	METADATA$ROW_ID
1	2	50	INSERT	FALSE	ab296cb8edb4504e02d7c6eb1854352e5ee1169
2	5	10	INSERT	FALSE	e610197f5a5edf1ea42744f812c71b6ba36d8d95
3	8	20	INSERT	FALSE	f9e72af73862bcb2929111644b89500869b7737
4	4	10	INSERT	FALSE	55e6b0deea7f2ddfe85286c63759d503a9498d3f
5	6	30	INSERT	FALSE	4c14a2143eb3619240472e179eb6c6bc8fe03c86

Update the data in staging

```
80  update raw_applicant_staging set job_id = 10 where id = 555555 ;
81
```

Results | Chart

	number of rows updated	number of multi-joined rows updated
1	1	0

Query Details
Query duration 773ms
Rows 1

Check stream

```
80  update raw_applicant_staging set job_id = 10 where id = 555555 ;
81
82  select * from strm_applicant1;
```

Results | Chart

	SSN	STREET_ADDRESS	EDUCATION_LEVEL	YEARS_OF_EXPERIENCE	JOB_ID	METADATA$AC
1	342-76-9087	5676 Washington Street	High School	5	10	INSERT
2	456-93-5629	3234 WateringCan Drive	Undergrad	4	10	INSERT
3	876-98-3245	6578 Historic Circle	Masters	6	30	INSERT
4	435-87-6532	8978 Autumn Day Drive	Masters	8	20	INSERT
5	767-34-7656	2343 India Street	Undergrad	2	10	INSERT

So stream is immediately reflecting what changes are done in staging, this is called net effect or default stream. It doesn't show you intermediate states, it shows net effect of multiple insert, updates and deletes

Let's delete and check

```
84  delete from raw_applicant_staging where id = 555555 ;
```

Results | Chart

	number of rows deleted
1	1

Query Details
Query duration 836ms
Rows 1

```
85  select * from strm_applicant1;
86
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LEVEL
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	Undergrad
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	Masters
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	Masters

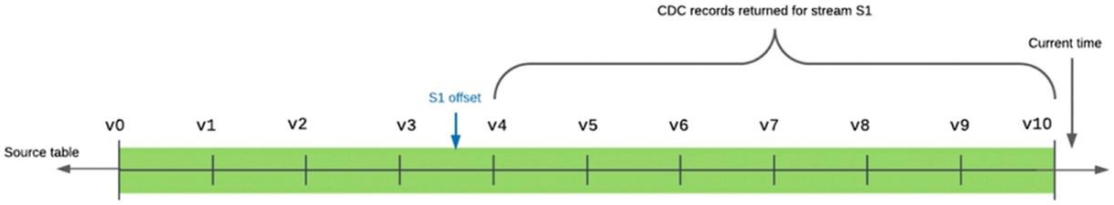
Query Details
Query duration 482ms
Rows 4
Query ID 01b51602-0000-a9e6-...

So, even if I delete it, it will not show you any history that it has 5 rows/records initially

So stream is immediately reflecting what changes are done in staging

Streams:

1. It is a database object. -> create or replace stream
2. Does not contain any data. Stores **Offset or a bookmark** when created.



3. Offset moves only if the stream is used in DML (truncate also) and DML is committed.
    a. Controlled by AUTOCOMMIT OR
    b. Explicit commit.
    c. To advance offset without DML
        i. Either recreate stream
        ii. OR insert the data to a temp table from stream using the false condition.

4. Stream additional columns
    a. METADATA$ACTION
    b. METADATA$ISUPDATE
    c. METADATA$ROW_ID

Let's see consumption of streams, how offset actually moves & how can stream is used to load the data into the target

Let's delete the data from staging and see consumption of stream

```
87  delete from raw_applicant_staging ;
88
```

Results | Chart

	number of rows deleted
1	4

Query Details
Query duration 477ms
Rows 1

```
89  select * from raw_applicant_staging ;
90
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LEVEL

Query produced no results

Query Details
Query duration 65ms
Rows 0
Query ID 01b5161d-0000-a9e6-0...

No records in stream

```
91  select * from strm_applicant1;
92
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LEVEL

Query produced no results

Query Details
Query duration 65ms
Rows 0
Query ID 01b5161e-0000-a9e6-0...

Let's insert some records

```
93  insert into raw_applicant_staging values (111111, 'James',    'Schwartz', 'M', 'American','342-76-9087','5676 Washington
    Street','High School', 5,10) ;
94  insert into raw_applicant_staging values (222222, 'Jessica',  'Escobar',  'F', 'Hispanic','456-93-5629','3234 WateringCan
    Drive','Undergrad', 4, 10) ;
95  insert into raw_applicant_staging values (333333, 'Ben',      'Hardy',    'M', 'American','876-98-3245','6578 Historic
    Circle','Masters', 6, 30) ;
96  insert into raw_applicant_staging values (444444, 'Anjali',    'Singh',   'F', 'Indian American','435-87-6532','8978 Autumn Day
    Drive','Masters', 8,20) ;
97  insert into raw_applicant_staging values (555555, 'Dean',     'Tracy',    'M', 'African','767-34-7656','2343 India
    Street','Undergrad', 2,50) ;
98
```

Let's check the stream

```
99  select * from strm_applicant1;
100
101
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LEVEL
1	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School
2	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	Undergrad
3	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	Masters
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	Masters
5	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	Undergrad

Query Details
Query duration 203ms
Rows 5
Query ID 01b51626-0000-a9e6-...

ID

Our target table has one record as shown

```
101  select * from candidates ;
102
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	CANDIDATE_EDUCATIO
1	100000	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School

Query Details
Query duration 275ms
Rows 1
Query ID 01b51628-0000-a9ff-0...

Let’s check our job_details table as well

```
select * from job_details ;
```

Results | Chart

	JOB_ID	NAME	CITY	STATE	EDUCATION_LEVEL
1	10	Painter	Raleigh	NC	High School
2	20	Software Engineer	Raleigh	NC	Masters
3	30	Data Architect	Raleigh	NC	Undergrad
4	40	Vice President	Raleigh	NC	Masters
5	50	Associate	Raleigh	NC	Masters

Query Details
Query duration 271ms
Rows 5
Query ID 01b5162b-0000-a9e6-...

JOB_ID

Now, let us merge the data into target table

```
merge into candidates tgt
using
(select
    id ,
    first_name ,
    last_name ,
    sex ,
    ethinicity ,
    ssn ,
    street_address ,
    str.education_level as ed_level,
    years_of_experience ,
    jdtl.job_id ,
    name ,
    city ,
    state ,
    jdtl.education_level as jreq_level ,
    case when str.education_level = jdtl.education_level then 'SHORTLISTED ' else 'STAGED' end as status,
    case when str.education_level = jdtl.education_level then 'The education level of candidate matched with job'
                                                          else 'Application received from candidate'
    end  as comments,
    str.metadata$action,
    str.metadata$isupdate
   from strm_applicant1 str inner join job_details jdtl on (str.job_id = jdtl.job_id)
) src
on tgt.id = src.id
-- insert clause
when not matched and src.metadata$action = 'INSERT' and metadata$isupdate = 'FALSE'
then insert
values (
    id ,
    first_name ,
    last_name ,
    sex ,
    ethinicity ,
    ssn ,
    street_address,
    ed_level,
    years_of_experience ,
    job_id ,
    name ,
    city ,
    state ,
    jreq_level,
    status,
    comments
)
;
```

Results | Chart

	number of rows inserted
1	5

Query Details
Query duration 731ms
Rows 1
Query ID 01b51630-0000-a9ff-0...

Hence our target table got all the 5 records, initially it is 1 record so total 6 records

```
154 select * from candidates ;
155
```

Results | Chart

	ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	CANDIDATE_EDUC
1	100000	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School
2	111111	James	Schwartz	M	American	342-76-9087	5676 Washington Street	High School
3	222222	Jessica	Escobar	F	Hispanic	456-93-5629	3234 WateringCan Drive	Undergrad
4	444444	Anjali	Singh	F	Indian American	435-87-6532	8978 Autumn Day Drive	Masters
5	333333	Ben	Hardy	M	American	876-98-3245	6578 Historic Circle	Masters
6	555555	Dean	Tracy	M	African	767-34-7656	2343 India Street	Undergrad

Query Details

Query duration 24ms

Rows 6

Query ID 01b51632-0000-a9ff-0...

ID

Check if stream is consumed, offset is moved

```
156 select * from strm_applicant1 ;
157
```

Results | Chart

ID	FIRST_NAME	LAST_NAME	SEX	ETHINICITY	SSN	STREET_ADDRESS	EDUCATION_LEVEI

Query produced no results

Query Details

Query duration 109ms

Rows 0

Query ID 01b51634-0000-a9e6-...