

Getting Started with Snowflake - Zero to Snowflake

Overview

Welcome to Snowflake! This entry-level guide designed for database and data warehouse administrators and architects will help you navigate the Snowflake interface and introduce you to some of our core capabilities. [Sign up for a free 30-day trial of Snowflake](#) and follow along with this lab exercise. Once we cover the basics, you'll be ready to start processing your own data and diving into Snowflake's more advanced features like a pro.

Prerequisites:

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

What You'll Learn:

- How to create stages, databases, tables, views, and virtual warehouses.
- How to load structured and semi-structured data.
- How to perform analytical queries on data in Snowflake, including joins between tables.
- How to clone objects.
- How to undo user errors using Time Travel.
- How to create roles and users, and grant them privileges.
- How to securely and easily share data with other accounts.
- How to consume datasets in the Snowflake Data Marketplace.

Prepare Your Lab Environment

If you haven't already, register for a [Snowflake free 30-day trial](#). The rest of the sections in this lab assume you are using a new Snowflake account created by registering for a trial.

The Snowflake edition (Standard, Enterprise, Business Critical, etc.) and cloud provider (AWS, Azure, GCP), and Region (US East, EU, etc.) you use for this lab do not matter. However, we suggest you select the region that is physically closest to you and Enterprise, our most popular offering, as your Snowflake edition.

After registering, you will receive an email with an activation link and URL for accessing your Snowflake account.

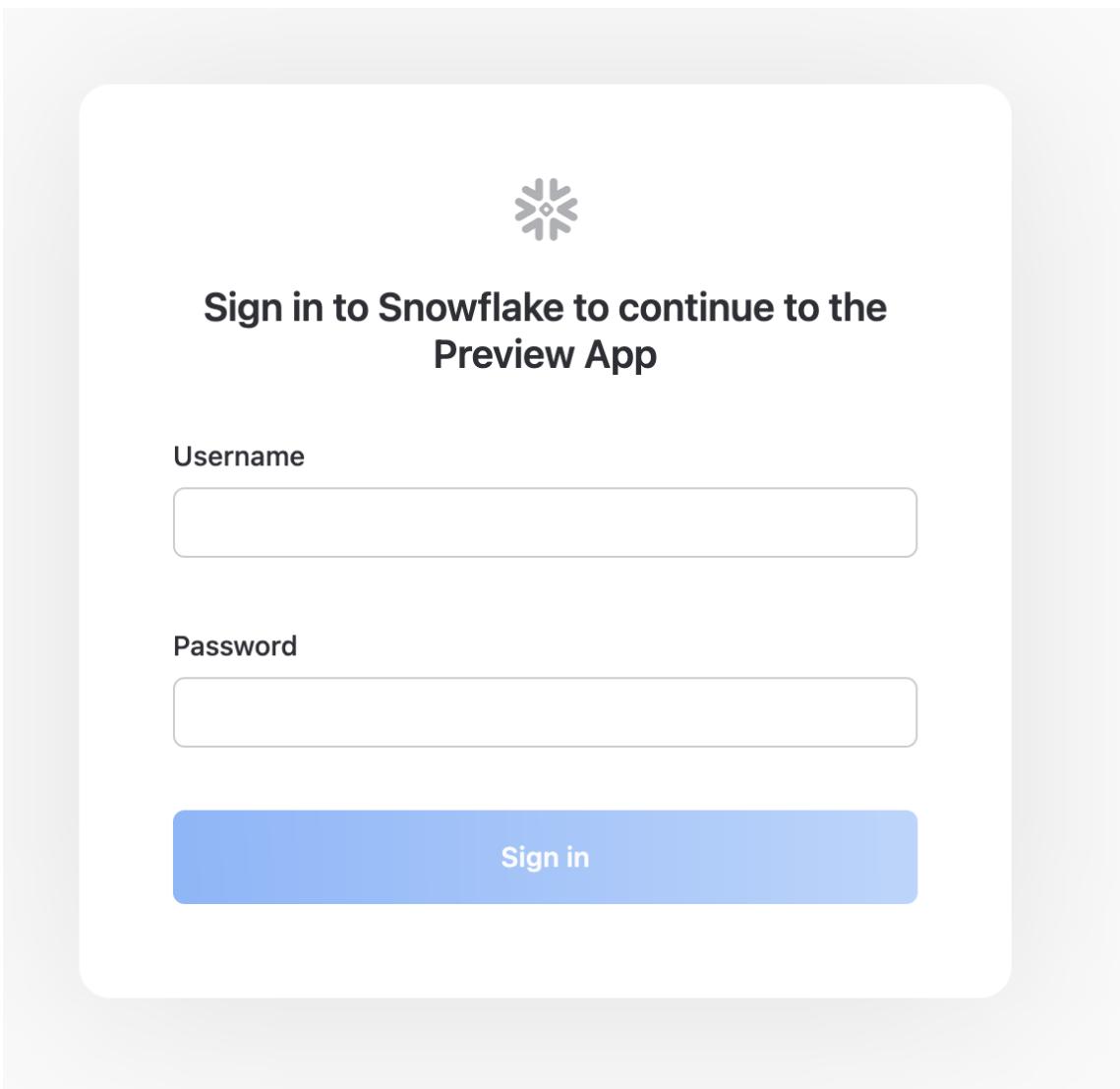
The Snowflake User Interface & Lab Story

Negative : **About the screenshots, sample code, and environment** Screenshots in this lab depict examples and results that may vary slightly from what you see when you complete the exercises.

Logging into the Snowflake User Interface (UI)

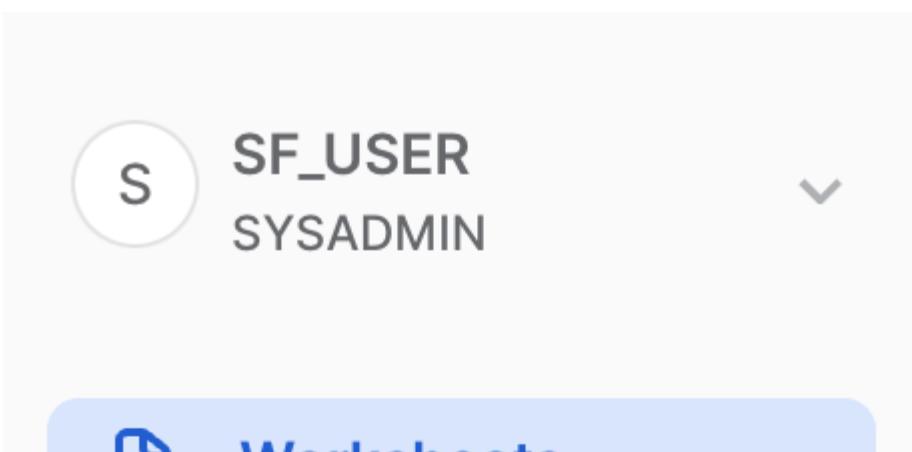
Open a browser window and enter the URL of your Snowflake 30-day trial environment that was sent with your registration email.

You should see the following login dialog. Enter the username and password that you specified during the registration:



Navigating the Snowflake UI

Let's get you acquainted with Snowflake! This section covers the basic components of the user interface. We will move from top to bottom on the left-hand side margin.





worksheets

Dashboards

Data

Marketplace

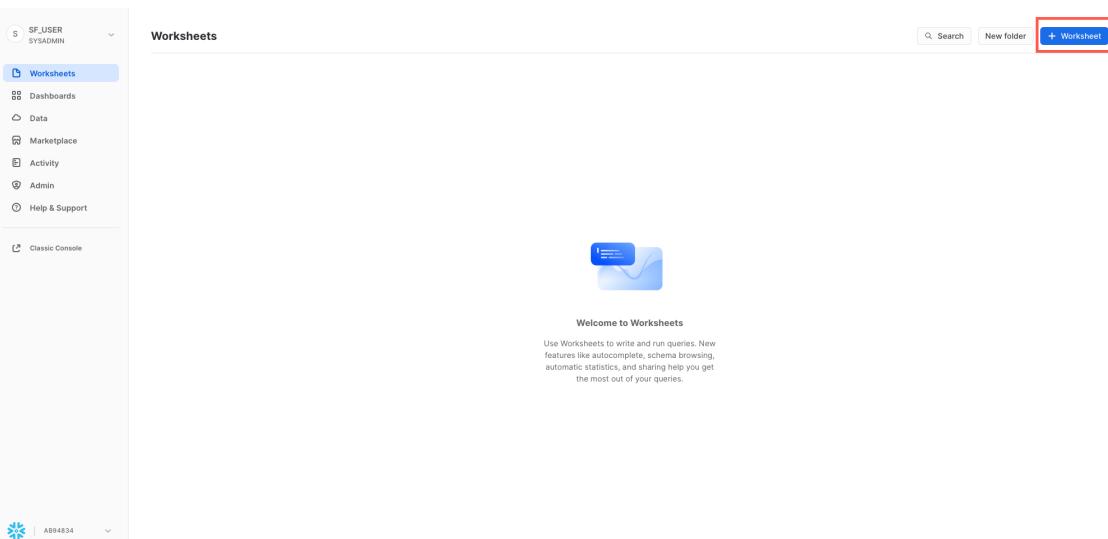
Activity

Admin

Help & Support



Classic Console



The **Worksheets** tab provides an interface for submitting SQL queries, performing DDL and DML operations, and viewing results as your queries or operations complete. A new worksheet is created by clicking **+ Worksheet** on the top right.

The top left corner contains the following:

- **Home** icon: Use this to get back to the main console/close the worksheet.
- **Worksheet_name** drop-down: The default name is the timestamp when the worksheet was created. Click the timestamp to edit the worksheet name. The drop-down also displays additional actions you can perform for the worksheet.
- **Manage filters** button: Custom filters are special keywords that resolve as a subquery or list of values.

The top right corner contains the following:

- **+ button**: This creates a new worksheet.

- **Context** box: This lets Snowflake know which role and warehouse to use during this session. It can be changed via the UI or SQL commands.
- **Share** button: Open the sharing menu to share to other users or copy the link to the worksheet.
- **Play/Run** button: Run the SQL statement where the cursor currently is or multiple selected statements.

The middle pane contains the following:

- Drop-down at the top for setting the database/schema/object context for the worksheet.
- General working area where you enter and execute queries and other SQL statements.

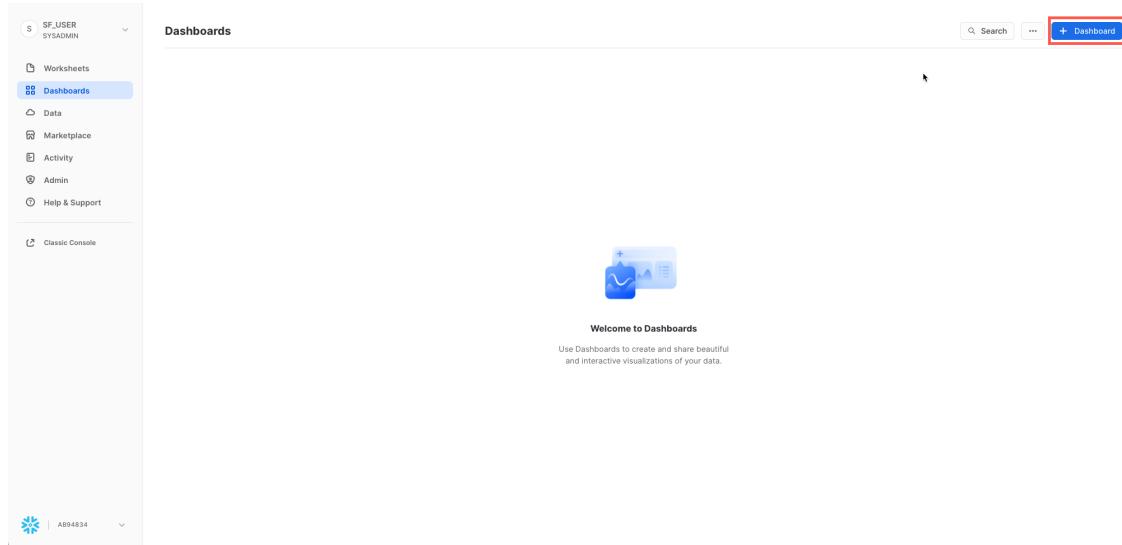
The middle-left panel contains the database objects browser which enables you to explore all databases, schemas, tables, and views accessible by the role currently in use for the worksheet.

The bottom pane displays the results of queries and other operations. Also includes 4 options (**Object**, **Query**, **Result**, **Chart**) that open/close their respective panels on the UI. **Chart** opens a visualization panel for the returned results. More on this later.

The various panes on this page can be resized by adjusting their sliders. If you need more room in the worksheet, collapse the database objects browser in the left panel. Many of the screenshots in this guide keep this panel closed.

Negative : **Worksheets vs the UI** Most of the exercises in this lab are executed using pre-written SQL within this worksheet to save time. These tasks can also be done via the UI, but would require navigating back-and-forth between multiple UI tabs.

Dashboards



The **Dashboards** tab allows you to create flexible displays of one or more charts (in the form of tiles, which can be rearranged). Tiles and widgets are produced by executing SQL queries that return results in a worksheet. Dashboards work at a variety of sizes with minimal configuration.

Databases

The screenshot shows the Snowflake UI with the left sidebar expanded. Under the 'Data' section, the 'Databases' tab is selected. The main area displays a table titled 'Databases' with one row:

NAME	SOURCE	OWNER	CREATED
SNOWFLAKE_SAMPLE_DATA	Share	ACCOUNTADMIN	1 hour ago

At the top right of the main area, there is a blue button labeled '+ Database'. The left sidebar also includes sections for 'Private Sharing', 'Provider Studio', 'Marketplace', 'Activity', 'Admin', and 'Help & Support'.

Under **Data**, the **Databases** tab shows information about the databases you have created or have permission to access. You can create, clone, drop, or transfer ownership of databases, as well as load data in the UI. Notice that a database already exists in your environment. However, we will not be using it in this lab.

Private Shared Data

The screenshot shows the Snowflake UI with the left sidebar expanded. Under the 'Data' section, the 'Private Sharing' tab is selected. The main area displays a table titled 'Shared With Me' with one row:

NAME	SOURCE	OWNER	CREATED
SNOWFLAKE_SAMPLE_DATA	Shared 1 hour ago		

At the top right of the main area, there is a blue button labeled 'Share Data'. The left sidebar also includes sections for 'Private Sharing', 'Provider Studio', 'Marketplace', 'Activity', 'Admin', and 'Help & Support'.

Also under **Data**, the **Private Shared Data** tab is where data sharing can be configured to easily and securely share Snowflake tables among separate Snowflake accounts or external users, without having to create a copy of the data. We will cover data sharing in Section 10.

Marketplace

The screenshot shows the Snowflake Marketplace interface. At the top, there's a search bar labeled 'Search Snowflake Marketplace' and navigation links for 'Categories', 'Business Needs', 'Providers', and 'My Requests'. Below this, a banner indicates 'Ready to Query', 'Free', 'Weather', 'Financial', '360-Degree Customer View', 'Demand Forecasting', 'Japan', and 'Last month'. The main content is divided into two sections: 'Most Popular' and 'Most Recent'. Each section contains four dataset cards. The 'Most Popular' section includes 'Starschema COVID-19 Epidemiological Data' (Personalized), 'Worldwide Address Data' (Personalized), 'Exchange Data International SAMPLE: EDI Foreign Exchange Rates' (Personalized), and 'Knomena Economy Data Atlas' (Personalized). The 'Most Recent' section includes 'AHEAD Chicago Divvy Bike Station Status' (Personalized), 'Alesco Data Alesco Email Database (sample)' (Personalized), 'Arcadia Arcadia Research Data' (Personalized), and 'Hyland Software, Inc. OnBase Log Bundle' (Personalized). A 'More >' link is visible at the bottom right of each section.

The **Marketplace** tab is where any Snowflake customer can browse and consume data sets made available by providers. There are two types of shared data: Public and Personalized. Public data is free data sets available for querying instantaneously. Personalized data requires reaching out to the provider of data for approval of sharing data.

Query History

The screenshot shows the 'Query History' page under the 'Activity' tab. The left sidebar shows the user is 'SF_USER SYSADMIN' with options for Worksheets, Dashboards, Data, Marketplace, Activity, Query History (which is selected and highlighted in blue), Copy History, Admin, Help & Support, and Classic Console. The right pane has a title 'Query History' and a message '0 Queries'. It includes a search bar and filter buttons for 'Status All', 'User SF_USER', 'Filters', and 'Columns'. Below the message, it says 'No Queries' and 'There are no queries matching your filters.'

Under **Activity** there are two tabs **Query History** and **Copy History**:

- **QuerHistory** is where previous queries are shown, along with filters that can be used to hone results (user, warehouse, status, query tag, etc.). View the details of all queries executed in the last 14 days from your Snowflake account. Click a query ID to drill into it for more information.
- **Copy History** shows the status of copy commands run to ingest data into Snowflake.

Warehouses

The screenshot shows the Snowflake Admin interface. On the left, there's a sidebar with a user icon and the text 'SF_USER SYSADMIN'. Below it are links for Worksheets, Dashboards, Data, Marketplace, Activity, Admin (which is selected), Usage, Warehouses (selected), Resource Monitors, Users & Roles, Security, Billing, Partner Connect, Help & Support, and Classic Console. At the bottom of the sidebar is a blue footer bar with a globe icon and the ID 'AB94834'. The main content area is titled 'Warehouses' and shows a table with one row. The table has columns: NAME, STATUS, SIZE, CLUSTERS, RUNNING, QUEUED, OWNER, and CREATED. The single row is for a warehouse named 'COMPUTE_WH' which is 'Suspended', 'X-Small', has '1-1' clusters, and was created '1 hour ago' by 'SYSADMIN'. There are also 'Search', 'Status All', 'Size All', and a clear button at the top right of the table.

Under **Admin**, the **Warehouses** tab is where you set up and manage compute resources known as virtual warehouses to load or query data in Snowflake. A warehouse called COMPUTE_WH already exists in your environment.

Resource Monitors

The screenshot shows the Snowflake Admin interface. The sidebar is identical to the previous one, with 'Admin' selected. The main content area is titled 'Resource Monitors' and shows a table with zero rows. The table has columns: NAME, STATUS, LEVEL, WAREHOUSES, and FREQUENCY. A message below the table says 'No Resource Monitors' and provides instructions: 'There are no resource monitors associated with this role. Switch roles to view resource monitors available to that role.' The footer bar at the bottom is identical to the one in the Warehouses screenshot.

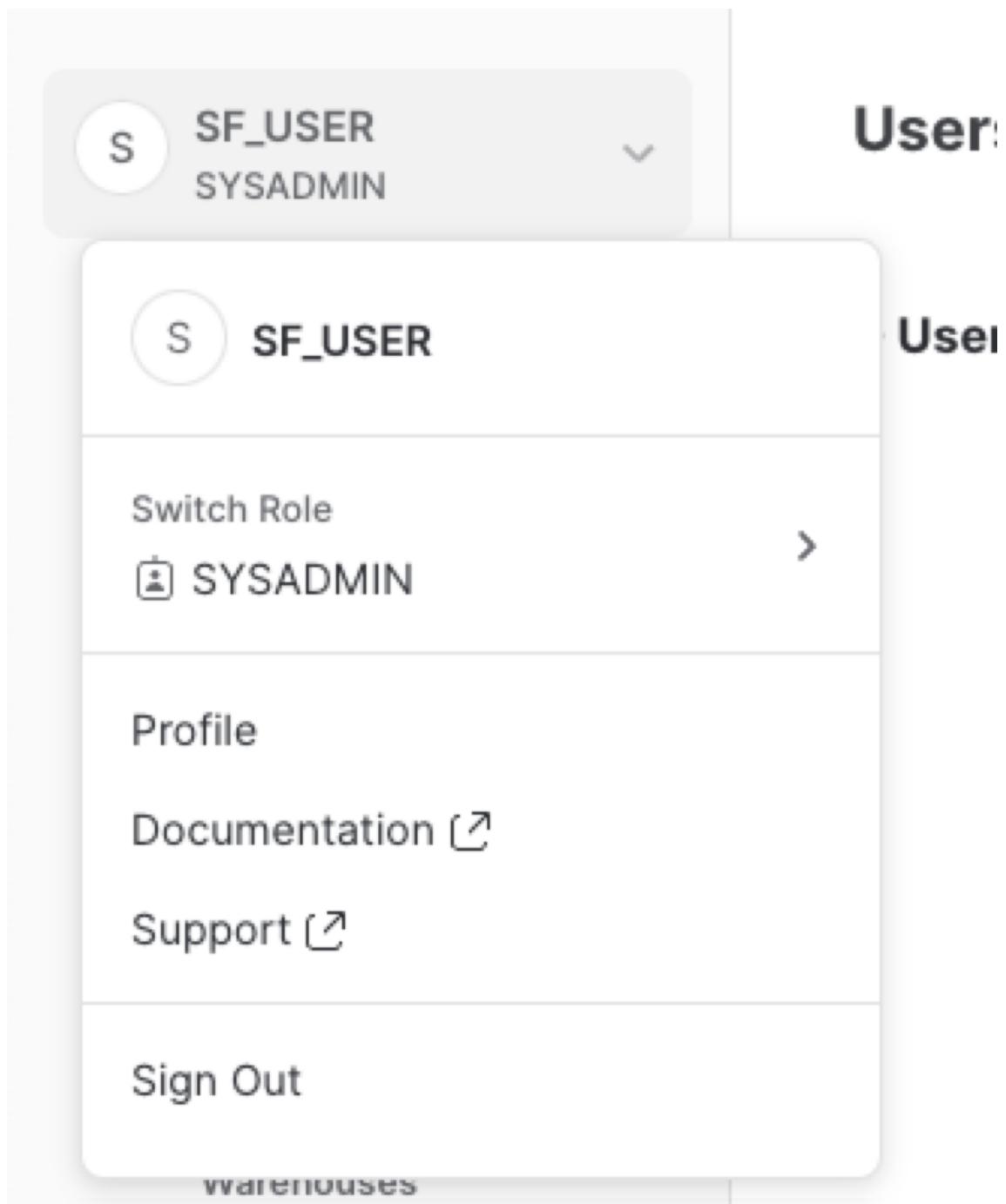
Under **Admin**, the **Resource Monitors** tab shows all the resource monitors that have been created to control the number of credits that virtual warehouses consume. For each resource monitor, it shows the credit quota, type of monitoring, schedule, and actions performed when the virtual warehouse reaches its credit limit.

Roles

Under **Admin**, the **Roles** sub-tab of the **Users and Roles** tab shows a list of the roles and their hierarchies. Roles can be created, reorganized, and granted to users in this tab. The roles can also be displayed in tabular/list format by selecting the **Table** sub-tab.

Users

Also under **Admin** tab, the **Users** sub-tab of the **Users and Roles** tab shows a list of users in the account, default roles, and owner of the users. For a new account, no records are shown because no additional roles have been created. Permissions granted through your current role determine the information shown for this tab. To see all the information available on the tab, switch your role to ACCOUNTADMIN.



Clicking on your username in the top right of the UI allows you to change your password, roles, and preferences. Snowflake has several system defined roles. You are currently in the default role of `SYSADMIN` and will stay in this role for the majority of the lab.

Negative : **SYSADMIN** The `SYSADMIN` (aka System Administrator) role has privileges to create warehouses, databases, and other objects in an account.

The Lab Story

This lab is based on the analytics team at Citi Bike, a real, citywide bike sharing system in New York City, USA. The team wants to run analytics on data from their internal transactional systems to better understand their riders and how to best serve them.

We will first load structured `.csv` data from rider transactions into Snowflake. Later we will work with open-source, semi-structured JSON weather data to determine if there is any correlation between the number of bike rides and the weather.

Preparing to Load Data

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

This section walks you through the steps to:

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

Negative : **Getting Data into Snowflake** There are many ways to get data into Snowflake from many locations including the COPY command, Snowpipe auto-ingestion, external connectors, or third-party ETL/ELT solutions.

For the purposes of this lab, we use the COPY command and AWS S3 storage to load data manually. In a real-world scenario, you would more likely use an automated process or ETL solution.

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, etc. On AWS S3, the data represents 61.5M rows, 377 objects, and 1.9GB 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
```

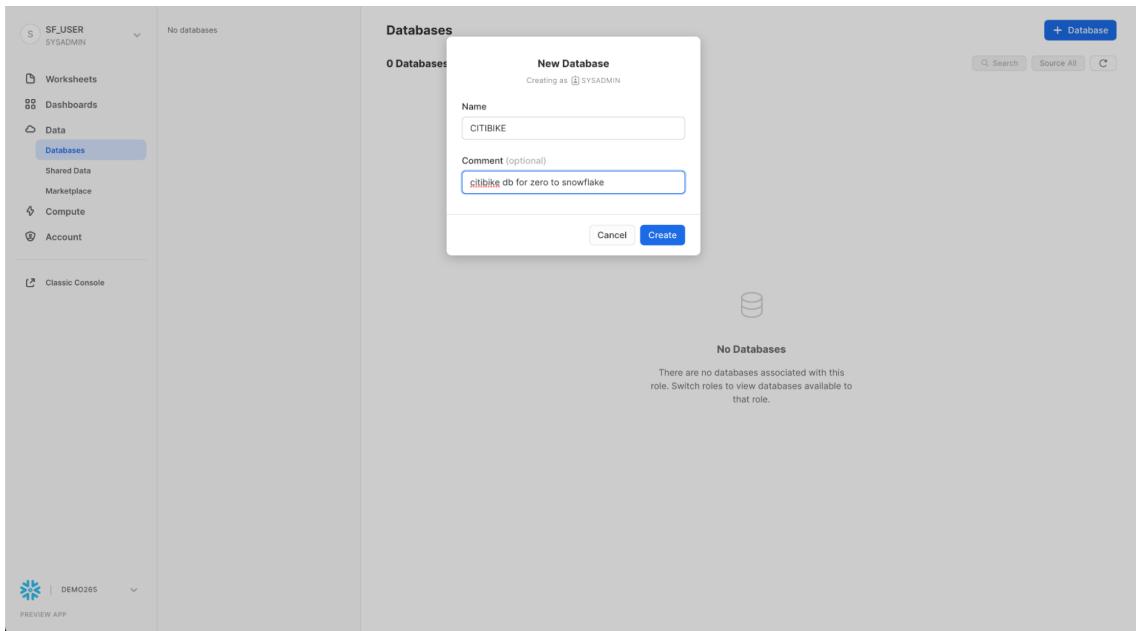
It is in comma-delimited format with a single header line and double quotes enclosing all string values, including the field headings in the header line. This will come into play later in this section as we configure the Snowflake table to store this data.

Create a Database and Table

First, let's create a database called `CITIBIKE` to use for loading the structured data.

Ensure you are using the sysadmin role by selecting **Switch Role > SYSADMIN**.

Navigate to the **Databases** tab. Click **Create**, name the database `CITIBIKE`, then click **CREATE**.



Now navigate to the **Worksheets** tab. You should see the worksheet we created in step 3.

We need to set the context appropriately within the worksheet. In the upper right corner of the worksheet, click the box next to the + to show the context menu. Here we control the elements you can see and run from each worksheet. We are using the UI here to set the context. Later in the lab, we will accomplish the same thing via SQL commands within the worksheet.

Select the following context settings:

Role: `SYSADMIN` Warehouse: `COMPUTE_WH`

2022-01-20 9:34am

Pinned (0)

No pinned objects

Search ...

CITIBIKE

Roles

- SYSADMIN
- ANALYST_CITIBIKE
- DBA_CITIBIKE
- DEV_CITIBIKE
- PUBLIC

Warehouses

- COMPUTE_WH

Query Details

- Query duration: 790ms
- Rows: 1
- COLUMN1: 100% blank

Next, in the drop-down for the database, select the following context settings:

Database: CITIBIKE Schema = PUBLIC

2022-02-04 6:01pm

Pinned (0)

No pinned objects

Search ...

CITIBIKE

SNOWFLAKE_SAMPLE_DATA

Databases

- CITIBIKE
- SNOWFLAKE_SAMPLE_DATA

Query Details

- Query duration: 56ms
- Rows: 1
- COLUMN1: 100% null

Negative : Data Definition Language (DDL) operations are free! All the DDL operations we have done so far do not require compute resources, so we can create all our objects for free.

To make working in the worksheet easier, let's rename it. In the top left corner, click the worksheet name, which is the timestamp when the worksheet was created, and change it to `CITIBIKE_ZERO_TO_SNOWFLAKE`.

Next we create a table called `TRIPS` to use for loading the comma-delimited data. Instead of using the UI, we use the worksheet to run the DDL that creates the table. Copy the following SQL text into your 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);

```

Negative : Many Options to Run Commands. SQL commands can be executed through the UI, via the **Worksheets** tab, using our SnowSQL command line tool, with a SQL editor of your choice via ODBC/JDBC, or through our other connectors (Python, Spark, etc.). As mentioned earlier, to save time, we are performing most of the operations in this lab via pre-written SQL executed in the worksheet as opposed to using the UI.

Run the query by placing your cursor anywhere in the SQL text and clicking the blue **Play/Run** button in the top right of the worksheet. Or use the keyboard shortcut [Ctrl]/[Cmd]+[Enter].

Verify that your TRIPS table has been created. At the bottom of the worksheet you should see a Results section displaying a "Table TRIPS successfully created" message.

The screenshot shows a Snowflake Worksheet interface. The top navigation bar includes a 'CITIBIKE_ZERO_TO_SNOWFLAKE' database dropdown, a '+' icon for creating new worksheets, a user dropdown for 'SYSADMIN - COMPUTE_WH', a 'Share' button, and a timestamp 'Updated 1 minute ago'. The main area is divided into two sections: 'Pinned (0)' on the left and 'CITIBIKE.PUBLIC +' on the right. In the 'CITIBIKE.PUBLIC +' section, a code editor contains the SQL command for creating the 'trips' table. Below the code editor is a results panel titled 'status' which displays the message 'Table TRIPS successfully created.' To the right of the results panel is a 'Query Details' sidebar showing a green progress bar for 'Query duration' at 277ms, 1 row processed, and a status of '100% filled'.

Navigate to the **Databases** tab by clicking the **HOME** icon in the upper left corner of the worksheet. Then click **Data** > **Databases**. In the list of databases, click **CITIBIKE** > **PUBLIC** > **TABLES** to see your newly created **TRIPS** table. If you don't see any databases on the left, expand your browser because they may be hidden.

The screenshot shows the Snowflake interface. On the left, the sidebar is visible with the user 'SF_USER' and the database 'CITIBIKE'. In the main area, under the 'Tables' section of the 'PUBLIC' schema, a new table named 'TRIPS' is listed. The table has one row and 0 bytes.

Click `TRIPS` and the **Columns** tab to see the table structure you just created.

The screenshot shows the detailed structure of the 'TRIPS' table. The table has 16 columns:

NAME	TYPE	NULLABLE	DEFAULT
BIKEID	NUMBER(38,0)	Yes	NULL
BIRTH_YEAR	NUMBER(38,0)	Yes	NULL
END_STATION_ID	NUMBER(38,0)	Yes	NULL
END_STATION_LATITUDE	FLOAT	Yes	NULL
END_STATION_LONGITUDE	FLOAT	Yes	NULL
END_STATION_NAME	VARCHAR(16777216)	Yes	NULL
GENDER	NUMBER(38,0)	Yes	NULL
MEMBERSHIP_TYPE	VARCHAR(16777216)	Yes	NULL
STARTTIME	TIMESTAMP_NTZ(9)	Yes	NULL
START_STATION_ID	NUMBER(38,0)	Yes	NULL
START_STATION_LATITUDE	FLOAT	Yes	NULL
START_STATION_LONGITUDE	FLOAT	Yes	NULL
START_STATION_NAME	VARCHAR(16777216)	Yes	NULL
STOPTIME	TIMESTAMP_NTZ(9)	Yes	NULL
TRIPDURATION	NUMBER(38,0)	Yes	NULL
USERTYPE	VARCHAR(16777216)	Yes	NULL

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.

Positive : For this lab we are using an AWS-East bucket. To prevent data egress/transfer costs in the future, you should select a staging location from the same cloud provider and region as your Snowflake account.

From the **Databases** tab, click the `CITIBIKE` database and `PUBLIC` schema. In the **Stages** tab, click the **Create** button, then **Stage > Amazon S3**.

The screenshot shows the Snowflake interface. On the left, the sidebar has 'SF_USER' selected under 'Databases'. The main area shows the 'CITIBIKE / PUBLIC' schema with a 'Tables' section containing 'TRIPS'. A context menu is open over the 'Stages' section, listing options like 'Table', 'View', 'Stage', 'Pipe', 'Stream', 'Task', 'Function', and 'Procedure'. Below this, another context menu is open for 'External Stage', listing 'Amazon S3', 'Microsoft Azure', and 'Google Cloud Platform'. At the bottom, a message says 'No stages' and 'This schema contains no stages.'

In the "Create Securable Object" dialog that opens, replace the following values in the SQL statement:

```
stage_name : citibike_trips

url : s3://snowflake-workshop-lab/citibike-trips-csv/
```

Note: Make sure to include the final forward slash (/) at the end of the URL or you will encounter errors later when loading data from the bucket. Also ensure you have removed 'credentials = (...) statejment which is not required. The create stage command should resemble that show above exactly.

Positive : The S3 bucket for this lab is public so you can leave the credentials options in the statement empty. In a real-world scenario, the bucket used for an external stage would likely require key information.

The screenshot shows the 'Create Securable Object' dialog. The left sidebar shows pinned objects and a search bar. The main area contains the following SQL code:

```
create stage citibike_trips
url = 's3://snowflake-workshop-lab/citibike-trips/';
--credentials = (aws_secret_key = '<key>' aws_key_id = '<id>');
```

At the top right, there are buttons for 'Open in Worksheets', 'SYSADMIN', 'No Warehouse selected', and 'Create Stage'. At the bottom, there are 'Objects' and 'Query' buttons.

Now let's take a look at the contents of the `citibike_trips` stage. Navigate to the **Worksheets** tab and execute the following SQL statement:

```
list @citibike_trips;
```

In the results in the bottom pane, you should see the list of files in the stage:

The screenshot shows the Snowflake UI with the following details:

- Top Bar:** CITIBIKE_ZERO_TO_SNOWFLAKE v, SYSADMIN - COMPUTE_WH, Share, Updated 7 seconds ago.
- Left Sidebar:** Pinned (0), No pinned objects, CITIBIKE PUBLIC *, CITIBIKE (0).
- Bottom Pane (Results):** A table listing 12 files from the `s3://snowflake-workshop-lab/citibike-trips-parquet` stage. The columns are name, size, md5, and last_modified. The table includes a header row and 12 data rows. To the right of the table are three cards: Query Details, Query Duration (3.6s), Rows (3.5K), and a histogram for size (123).

name	size	md5	last_modified
1 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/03/data_01a19496-c	219,733	5c4e4f21988b692a8cea379cdd094aca	Wed, 12 Jan 2022 13:10:38 GMT
2 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/04/data_01a19496-c	240,490	920744132d8da7cd8abbfd1f1465d8	Wed, 12 Jan 2022 13:10:37 GMT
3 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/05/data_01a19496-c	227,228	d982d1a2e3692abea51917f134e3925	Wed, 12 Jan 2022 13:10:36 GMT
4 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/06/data_01a19496-c	235,713	1ccc34af2eeb9c3eab0771f3ae193167	Wed, 12 Jan 2022 13:10:34 GMT
5 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/07/data_01a19496-c	224,639	82fe1f21949abeb6bd2c2e4579cfec3d3	Wed, 12 Jan 2022 13:10:39 GMT
6 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/08/data_01a19496-c	238,398	2d6cc208b20245127bc65beb1938	Wed, 12 Jan 2022 13:10:35 GMT
7 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/09/data_01a19496-c	227,747	4a6b6bcc6d3b3b379e23da4a466c1a8c	Wed, 12 Jan 2022 13:10:40 GMT
8 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/10/data_01a19496-c	249,432	fe216ba3f84d7dfb52919986194c6	Wed, 12 Jan 2022 13:10:33 GMT
9 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/11/data_01a19496-c	285,569	c057bbb681d8420e42d86a24777d8	Wed, 12 Jan 2022 13:10:40 GMT
10 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/12/data_01a19496-c	261,702	91cb6989f12e2c83ff70e75aa2a071c	Wed, 12 Jan 2022 13:10:32 GMT
11 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/13/data_01a19496-c	269,385	1b6d78c7f48d8995ac76e85278ef1d2a2	Wed, 12 Jan 2022 13:10:40 GMT
12 s3://snowflake-workshop-lab/citibike-trips-parquet/2013/06/14/data_01a19496-c	266,097	d85727a36ea0df3e7f5ebe1b725863be	Wed, 12 Jan 2022 13:10:38 GMT

Create a File Format

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

In the worksheet, run the following command to create the file format:

```
--create file format

create or replace file format csv type='csv'
compression = 'auto' field_delimiter = ',' record_delimiter = '\n'
skip_header = 0 field_optionally_enclosed_by = '\042' trim_space = false
error_on_column_count_mismatch = false escape = 'none' escape_unenclosed_field =
'\134'

date_format = 'auto' timestamp_format = 'auto' null_if = ('') comment = 'file format
for ingesting data for zero to snowflake';
```

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with 'CITIBIKE_ZERO_TO_SNOWFLAKE', 'SYSADMIN', 'COMPUTE_WH', 'Share', and a 'Updated 4 seconds ago' timestamp. Below the navigation is a sidebar with sections for 'Pinned', 'No pinned objects', 'Q Search', and 'CITIBIKE'. The main area contains a code editor with the following SQL script:

```

5   starttime timestamp,
6   stoptime timestamp,
7   start.station_id integer,
8   start.station_name string,
9   start.station.latitude float,
10  start.station.longitude float,
11  end.station_id integer,
12  end.station.name string,
13  end.station.latitude float,
14  end.station.longitude float,
15  bikeid integer,
16  membership_type string,
17  user_type string,
18  birth_year integer,
19  gender integer
20 );
21 --created external stage. List files.
22 list @citibike_trips;
23
24 --create file format
25 CREATE FILE FORMAT "CITIBIKE"."PUBLIC" CSV TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n' SKIP_HEADER = 0
FIELD_OPTIONALLY_ENCLOSED_BY = '\"' TRIM_SPACE = FALSE ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE_UNENCLOSED_FIELD = '\\"' DATE_FORMAT = 'AUTO'
TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('') COMMENT = 'creation of file format for zero to snowflake';

```

Below the code editor is a results table with one row:

	status
1	File format CSV successfully created.

On the right side, there's a 'Query Details' panel showing a duration of 123ms and 1 row processed, with a status of 100% filled.

Verify that the file format has been created with the correct settings by executing the following command:

```
--verify file format is created

show file formats in database citibike;
```

The file format created should be listed in the result:

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with 'CITIBIKE_ZERO_TO_SNOWFLAKE', 'SYSADMIN', 'COMPUTE_WH', 'Share', and a 'Updated 2 seconds ago' timestamp. Below the navigation is a sidebar with sections for 'Pinned', 'No pinned objects', 'Q Search', and 'CITIBIKE'. The main area contains a code editor with the following SQL script:

```

8   start.station.name string,
9   start.station.latitude float,
10  start.station.longitude float,
11  end.station_id integer,
12  end.station.name string,
13  end.station.latitude float,
14  end.station.longitude float,
15  bikeid integer,
16  membership_type string,
17  user_type string,
18  birth_year integer,
19  gender integer
20 );
21 --created external stage. List files.
22 list @citibike_trips;
23
24 --create file format
25 CREATE FILE FORMAT "CITIBIKE"."PUBLIC" CSV TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n' SKIP_HEADER = 0
FIELD_OPTIONALLY_ENCLOSED_BY = '\"' TRIM_SPACE = FALSE ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE_UNENCLOSED_FIELD = '\\"' DATE_FORMAT = 'AUTO'
TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('') COMMENT = 'creation of file format for zero to snowflake';
26
27 --verify file format is created
28 show file formats in database citibike;

```

Below the code editor is a results table with one row:

	created_on	name	database_name	schema_name	type	owner	comment	for
1	2022-01-20 11:12:27.666 -0800	CSV	CITIBIKE	PUBLIC	CSV	SYSADMIN	creation of file format for zero to snowflake	(T)

On the right side, there's a 'Query Details' panel showing a duration of 97ms and 1 row processed, with statuses for 'created_on', 'name', 'database_name', and 'schema_name' all at 100% filled.

Loading Data

In this section, we will use a virtual warehouse and the COPY command to initiate bulk loading of structured data into the Snowflake table we created in the last section.

Resize and Use a Warehouse for Data Loading

Compute resources are needed for loading data. Snowflake's compute nodes are called virtual warehouses and they can be dynamically sized up or out according to workload, whether you are loading data, running a query, or performing a DML operation. Each workload can have its own warehouse so there is no resource contention.

Navigate to the **Warehouses** tab (under **Admin**). This is where you can view all of your existing warehouses, as well as analyze their usage trends.

Note the **+ Warehouse** option in the upper right corner of the top. This is where you can quickly add a new warehouse. However, we want to use the existing warehouse COMPUTE_WH included in the 30-day trial environment.

Click the row of the `COMPUTE_WH` warehouse. Then click the ... (dot dot dot) in the upper right corner text above it to see the actions you can perform on the warehouse. We will use this warehouse to load the data from AWS S3.

The screenshot shows the Snowflake Admin interface. On the left, a sidebar navigation bar includes options like Worksheets, Dashboards, Data, Marketplace, Activity, Admin, Usage, Warehouses (which is selected), Resource Monitors, Users & Roles, Security, Billing, Contacts, Accounts, Partner Connect, Help & Support, and Classic Console. The main content area displays the details for the `COMPUTE_WH` warehouse. At the top, there's a breadcrumb trail: `COMPUTE_WH`, `Warehouse`, `SYSADMIN`, and `1 month ago`. Below this is a chart titled "Warehouse Activity" showing data load activity from Sep 6 to Sep 20. To the right of the chart is a context menu with options: Edit, Suspend, Drop, Transfer Ownership, and a "..." button. The "Edit" option is highlighted with a red box. Below the chart is a "Details" section with various configuration parameters. At the bottom of the main content area is a "Privileges" section.

Click **Edit** to walk through the options of this warehouse and learn some of Snowflake's unique functionality.

Positive : If this account isn't using Snowflake Enterprise Edition (or higher), you will not see the **Mode** or **Clusters** options shown in the screenshot below. The multi-cluster warehouses feature is not used in this lab, but we will discuss it as a key capability of Snowflake.

Edit Warehouse

COMPUTE_WH as SYSADMIN

Name	Size ?
COMPUTE_WH	X-Large 16 credits/hour
Comment (optional)	
<hr/>	
Multi-cluster Warehouse	
Scale compute resources as query needs change	
Mode	Maximized
Clusters	1
<hr/>	
Advanced Warehouse Options ▾	
Auto Resume	ON
Auto Suspend	ON
Suspend After (min)	10
<hr/>	
<button>Cancel</button>	<button>Save Warehouse</button>

- The **Size** drop-down is where the capacity of the warehouse is selected. For larger data loading operations or more compute-intensive queries, a larger warehouse is recommended. The sizes translate to the underlying compute resources provisioned from the cloud provider (AWS, Azure, or GCP) where your Snowflake account is hosted. It also determines the number of credits consumed by the warehouse for each full hour it runs. The larger the size, the more compute resources from the cloud provider are allocated to

the warehouse and the more credits it consumes. For example, the `4X-Large` setting consumes 128 credits for each full hour. This sizing can be changed up or down at any time with a simple click.

- If you are using Snowflake Enterprise Edition (or higher) and the **Multi-cluster Warehouse** option is enabled, you will see additional options. This is where you can set up a warehouse to use multiple clusters of compute resources, up to 10 clusters. For example, if a `4X-Large` multi-cluster warehouse is assigned a maximum cluster size of 10, it can scale out to 10 times the compute resources powering that warehouse...and it can do this in seconds! However, note that this will increase the number of credits consumed by the warehouse to 1280 if all 10 clusters run for a full hour (128 credits/hour x 10 clusters). Multi-cluster is ideal for concurrency scenarios, such as many business analysts simultaneously running different queries using the same warehouse. In this use case, the various queries are allocated across multiple clusters to ensure they run quickly.
- Under **Advanced Warehouse Options**, the options allow you to automatically suspend the warehouse when not in use so no credits are needlessly consumed. There is also an option to automatically resume a suspended warehouse so when a new workload is sent to it, it automatically starts back up. This functionality enables Snowflake's efficient "pay only for what you use" billing model which allows you to scale your resources when necessary and automatically scale down or turn off when not needed, nearly eliminating idle resources.

Negative : **Snowflake Compute vs Other Data Warehouses** Many of the virtual warehouse and compute capabilities we just covered, such as the ability to create, scale up, scale out, and auto-suspend/resume virtual warehouses are easy to use in Snowflake and can be done in seconds. For on-premise data warehouses, these capabilities are much more difficult, if not impossible, as they require significant physical hardware, over-provisioning of hardware for workload spikes, and significant configuration work, as well as additional challenges. Even other cloud-based 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 be careful about performing the following actions without good reason or you may burn through your \$400 of free credits more quickly than desired:

- Do not disable auto-suspend. If auto-suspend is disabled, your warehouses continues to run and consume credits even when not in use.
- Do not use a warehouse size that is excessive given the workload. The larger the warehouse, the more credits are consumed.

We are going to use this virtual warehouse to load the structured data in the CSV files (stored in the AWS S3 bucket) into Snowflake. However, we are first going to change the size of the warehouse to increase the compute resources it uses. After the load, note the time taken and then, in a later step in this section, we will re-do the same load operation with an even larger warehouse, observing its faster load time.

Change the **Size** of this data warehouse from `X-Small` to `Small`. then click the **Save Warehouse** button:

Edit Warehouse

COMPUTE_WH as SYSADMIN

Name: COMPUTE_WH

Size: Small 2 credits/hour

Comment (optional):

Multi-cluster Warehouse: Scale compute resources as query needs change

Mode: Maximized

Clusters: 1

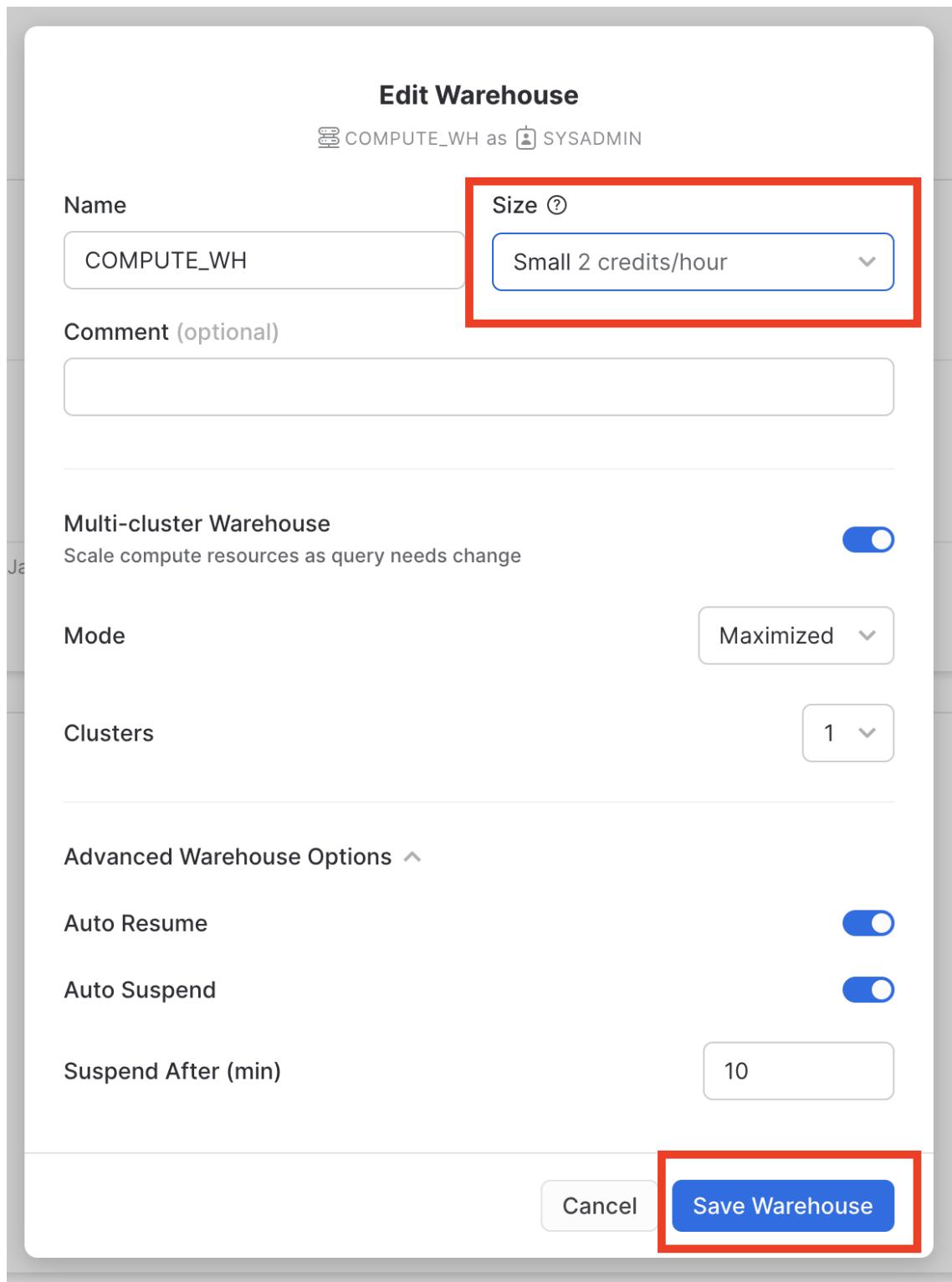
Advanced Warehouse Options ▾

Auto Resume:

Auto Suspend:

Suspend After (min): 10

Cancel **Save Warehouse**



Load the Data

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

Navigate back to the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet in the **Worksheets** tab. Make sure the worksheet context is correctly set:

Role: `SYSADMIN` Warehouse: `COMPUTE_WH` Database: `CITIBIKE` Schema = `PUBLIC`

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 PATTERN = '.*csv.*' ;
```

In the result pane, you should see the status of each file that was loaded. Once the load is done, in the **Query Details** pane on the bottom right, you can scroll through the various statuses, error statistics, and visualizations for the last statement executed:

file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error
1 s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_1_0.csv.gz	LOADED	112,123	112,123	1	0	
2 s3://snowflake-workshop-lab/citibike-trips/trips_2013_2_7_0.csv.gz	LOADED	93,143	93,143	1	0	
3 s3://snowflake-workshop-lab/citibike-trips/trips_2013_5_7_0.csv.gz	LOADED	111,042	111,042	1	0	
4 s3://snowflake-workshop-lab/citibike-trips/trips_2014_0_7_0.csv.gz	LOADED	112,334	112,334	1	0	
5 s3://snowflake-workshop-lab/citibike-trips/trips_2014_3_0_0.csv.gz	LOADED	126,789	126,789	1	0	
6 s3://snowflake-workshop-lab/citibike-trips/trips_2014_4_7_0.csv.gz	LOADED	109,835	109,835	1	0	
7 s3://snowflake-workshop-lab/citibike-trips/trips_2014_6_7_0.csv.gz	LOADED	106,608	106,608	1	0	
8 s3://snowflake-workshop-lab/citibike-trips/trips_2015_0_2_0.csv.gz	LOADED	125,082	125,082	1	0	
9 s3://snowflake-workshop-lab/citibike-trips/trips_2015_1_2_0.csv.gz	LOADED	154,940	154,940	1	0	

Next, navigate to the **Query History** tab by clicking the **Home** icon and then **Activity > Query History**. Select the query at the top of the list, which should be the COPY INTO statement that was last executed. Select the **Query**

Profile tab and note the steps taken by the query to execute, query details, most expensive nodes, and additional statistics.



Now let's reload the `TRIPS` table with a larger warehouse to see the impact the additional compute resources have on the loading time.

Go back to the worksheet and use the `TRUNCATE TABLE` command to clear the table of all data and metadata:

```
truncate table trips;
```

Verify that the table is empty by running the following command:

```
--verify table is clear
select * from trips limit 10;
```

The result should show "Query produced no results".

Change the warehouse size to `large` using the following `ALTER WAREHOUSE`:

```
--change warehouse size from small to large (4x)
alter warehouse compute_wh set warehouse_size='large';
```

Verify the change using the following `SHOW WAREHOUSES`:

```
--load data with large warehouse
show warehouses;
```

CITIBIKE_ZERO_TO_SNOWFLAKE

```

copy into trips from @citibike_trips file_format=csv;
--clear table and metadata on table trips for next test
truncate table trips;
--verify table is clear
select * from trips limit 10;
--change warehouse size from small to large (4x)
alter warehouse compute_wh set warehouse_size='large';
--verify large warehouse
show warehouses;

```

Query Details

	Value
Query duration	92ms
Rows	1

name state type size min_cluster_count max_cluster_count started_clusters running queued is_de

name	state	type	size	min_cluster_count	max_cluster_count	started_clusters	running	queued	is_de
1 COMPUTE_WH	STARTED	STANDARD	Large	1	1	1	0	0	N

The size can also be changed using the UI by clicking on the worksheet context box, then the **Configure** (3-line) icon on the right side of the context box, and changing `Small` to `Large` in the **Size** drop-down:

The screenshot shows the 'Roles' page in the Snowflake UI. The 'SYSADMIN' role is selected, indicated by a blue checkmark next to its name. Other roles listed include ANALYST_CITIBIKE, DBA_CITIBIKE, DEV_CITIBIKE, and PUBLIC.

The screenshot shows the 'Warehouses' page in the Snowflake UI. The 'COMPUTE_WH' warehouse is selected. A dropdown menu is open for the 'Size' setting, showing options: X-Small, Small, Medium, Large (selected), X-Large, 2X-Large, 3X-Large, 4X-Large, and 5X-Large.

Execute the same COPY INTO statement as before to load the same data again:

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

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with icons for Home, Activity, and Query History. Below it is a search bar and a sidebar with sections for Pinned objects, No pinned objects, and a Search bar. The main area contains a code editor with a query script. The script includes comments like `--VERIFY 'CITIBIKE' IS CLEAR;` and `--change warehouse size from small to large (4x);` followed by an `ALTER WAREHOUSE` command. A specific line, `copy into trips from @citibike_trips file_format=csv;`, is highlighted in blue. To the right of the code editor is a progress bar indicating the query is running. Below the code editor is a results table with columns: file, status, rows_parsed, rows_loaded, error_limit, errors_seen, first_error, and Query Details. The table lists 18 rows, each corresponding to a CSV file from S3. The status column shows all rows as LOADED. The Query Details section on the right provides a summary with a duration of 9.8s, 376 rows processed, and 100% filled. It also includes histograms for rows_parsed, rows_loaded, and error_limit.

file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error	Query Details
1 s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_0_0.csv.gz	LOADED	117,943	117,943	1	0		Query duration: 9.8s
2 s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_3_0.csv.gz	LOADED	89,057	89,057	1	0		Rows: 376
3 s3://snowflake-workshop-lab/citibike-trips/trips_2014_1_5_0.csv.gz	LOADED	135,516	135,516	1	0		100% filled
4 s3://snowflake-workshop-lab/citibike-trips/trips_2014_1_3_0.csv.gz	LOADED	158,108	158,108	1	0		status: LOADED
5 s3://snowflake-workshop-lab/citibike-trips/trips_2015_1_1_0.csv.gz	LOADED	168,725	168,725	1	0		rows_parsed: 123
6 s3://snowflake-workshop-lab/citibike-trips/trips_2015_4_1_0.csv.gz	LOADED	141,423	141,423	1	0		rows_loaded: 123
7 s3://snowflake-workshop-lab/citibike-trips/trips_2015_7_5_0.csv.gz	LOADED	156,016	156,016	1	0		error_limit: 123
8 s3://snowflake-workshop-lab/citibike-trips/trips_2016_6_5_0.csv.gz	LOADED	189,709	189,709	1	0		100% filled
9 s3://snowflake-workshop-lab/citibike-trips/trips_2017_6_3_0.csv.gz	LOADED	266,912	266,912	1	0		
10 s3://snowflake-workshop-lab/citibike-trips/trips_2018_6_0_0.csv.gz	LOADED	111,386	111,386	1	0		
11 s3://snowflake-workshop-lab/citibike-trips/trips_2018_7_1_0.csv.gz	LOADED	124,464	124,464	1	0		
12 s3://snowflake-workshop-lab/citibike-trips/trips_2013_4_4_0.csv.gz	LOADED	104,107	104,107	1	0		
13 s3://snowflake-workshop-lab/citibike-trips/trips_2016_7_4_0.csv.gz	LOADED	121,945	121,945	1	0		
14 s3://snowflake-workshop-lab/citibike-trips/trips_2013_4_0_0.csv.gz	LOADED	85,271	85,271	1	0		
15 s3://snowflake-workshop-lab/citibike-trips/trips_2014_2_3_0.csv.gz	LOADED	102,555	102,555	1	0		
16 s3://snowflake-workshop-lab/citibike-trips/trips_2015_3_2_0.csv.gz	LOADED	125,585	125,585	1	0		
17 s3://snowflake-workshop-lab/citibike-trips/trips_2014_4_5_0.csv.gz	LOADED	149,304	149,304	1	0		
18 s3://snowflake-workshop-lab/citibike-trips/trips_2018_2_2_0.csv.gz	LOADED	218,112	218,112	1	0		

Once the load is done, navigate back to the **Queries** page (**Home** icon > **Activity** > **Query History**). Compare the times of the two COPY INTO commands. The load using the `Large` warehouse was significantly faster.

Create a New Warehouse for Data Analytics

Going back to the lab story, let's assume the Citi Bike team wants to eliminate 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 various 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 use this warehouse to perform analytics in the next section.

Navigate to the **Admin** > **Warehouses** tab, click **+ Warehouse**, and name the new warehouse `` and set the size to `Large`.

If you are using Snowflake Enterprise Edition (or higher) and **Multi-cluster Warehouses** is enabled, you will see additional settings:

- Make sure **Max Clusters** is set to `1`.
- Leave all the other settings at their defaults.

The screenshot shows the Snowflake Web UI interface. On the left, there's a sidebar with navigation links like Worksheets, Dashboards, Data, Compute, History, Warehouses (which is selected), Resource Monitors, and Account. The main area is titled 'Warehouses' and shows a table with one row: '1 Warehouse'. The row contains 'COMPUTE_WH' under 'NAME' and 'Started' under 'STATUS'. A modal window titled 'New Warehouse' is open, prompting for a name ('ANALYTICS_WH'), size ('Large 8 credits/hour'), and a comment ('analytics warehouse for zero to snowflake'). It also includes settings for 'Multi-cluster Warehouse' (auto-scale is turned on), 'Mode' (Auto-scale), 'Min Clusters' (1), 'Max Clusters' (1), and 'Scaling Policy' (Standard). Under 'Advanced Warehouse Options', 'Auto Resume' and 'Auto Suspend' are turned on, with a 'Suspend After (min)' set to 10. At the bottom of the modal are 'Cancel' and 'Create Warehouse' buttons.

Click the **Create Warehouse** button to create the warehouse.

Working with Queries, the Results Cache, & Cloning

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

Negative : Real World Roles and Querying Within a real company, analytics users would likely have a different role than SYSADMIN. To keep the lab simple, we are going to stay with the SYSADMIN role for this section. Additionally, querying would typically be done with a business intelligence product like Tableau, Looker, PowerBI, etc. For more advanced analytics, data science tools like Datarobot, Dataiku, AWS Sagemaker or many others can query Snowflake. Any technology that leverages JDBC/ODBC, Spark, Python, or any of the other supported programmatic interfaces can run analytics on the data in Snowflake. To keep this lab simple, all queries are being executed via the Snowflake worksheet.

Execute Some Queries

Go to the **CITIBIKE_ZERO_TO_SNOWFLAKE** worksheet and change the warehouse to use the new warehouse you created in the last section. Your worksheet context should be the following:

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

The screenshot shows the Snowflake Worksheet interface. At the top, it displays the context: Role: `SYSADMIN`, Warehouse: `ANALYTICS_WH` (L), Database: `CITIBIKE`, Schema = `PUBLIC`. Below this, a code editor window shows the following SQL snippet:

```

1  CITIBIKE.PUBLIC =
2
3  create or replace table trips (
4    tripduration integer,
5    starttime timestamp,
6    stoptime timestamp,
7    start_station_id integer,
8    start_station_name string,
9    start_station_latitude float,
10   start_station_longitude float,
11   end_station_id integer,
12   end_station_name string
13
14
15
16
17

```

To the right of the code editor is a context menu with several options: 'Roles', 'Policy', 'Shares', 'Warehouse', 'Script', 'Copy', 'Share', and 'Edit'. The 'Warehouse' option is highlighted with a red box. The 'Warehouse' dropdown menu shows the available warehouses: `COMPUTE_WH` (selected) and `ANALYTICS_WH`. The 'Role' dropdown menu shows the available roles: `SYSADMIN` (selected), `ANALYST_CITIBIKE`, `DBA_CITIBIKE`, `DEV_CITIBIKE`, and `PUBLIC`. Both the 'Warehouse' and 'Role' dropdowns are highlighted with red boxes.

Run the following query to see a sample of the `trips` data:

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

The screenshot shows a Snowflake worksheet interface. At the top, there's a header with a home icon, a search bar, and tabs for 'Pinned' and 'Search'. Below the header, the code for loading data into the 'trips' table is displayed. A specific line of the code, `select * from trips limit 20;`, is highlighted with a blue selection bar. Below the code, the results of the query are shown in a table with columns: TRIPDURATION, STARTTIME, STOPTIME, START_STATION_ID, START_STATION_NAME, and START_STATION_LATIT. The table contains 14 rows of trip data. To the right of the table, there are several data visualization cards: 'Query Details' (duration 1.2s), 'Rows' (20), 'TRIPDURATION' histogram (123 trips), 'STARTTIME' histogram (100% filled), 'STOPTIME' histogram (100% filled), and 'START_STATION_ID' histogram (123). The bottom of the interface has tabs for 'Objects', 'Query', 'Results', and 'Chart', with the 'Results' tab currently selected.

	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	START_STATION_NAME	START_STATION_LATIT
1	258	2018-04-03 18:31:11.000	2018-04-03 18:35:30.000	3,684	North Moore St & Greenwich St	40.720195
2	659	2018-04-03 18:31:12.000	2018-04-03 18:42:12.000	127	Barrow St & Hudson St	40.731724
3	1,629	2018-04-03 18:31:13.000	2018-04-03 18:58:22.000	3,002	South End Ave & Liberty St	40.7111
4	286	2018-04-03 18:31:16.000	2018-04-03 18:36:05.000	465	Broadway & W 41 St	40.755131
5	1,097	2018-04-03 18:31:19.000	2018-04-03 18:49:36.000	305	E 58 St & 3 Ave	40.76095
6	346	2018-04-03 18:31:19.000	2018-04-03 18:37:06.000	526	E 33 St & 5 Ave	40.747651
7	313	2018-04-03 18:31:22.000	2018-04-03 18:36:35.000	519	Pershing Square North	40.7511
8	421	2018-04-03 18:31:24.000	2018-04-03 18:38:26.000	3,258	W 27 St & 10 Ave	40.750181
9	1,185	2018-04-03 18:31:25.000	2018-04-03 18:51:10.000	3,263	Cooper Square & Astor Pl	40.729514
10	272	2018-04-03 18:31:26.000	2018-04-03 18:35:59.000	3,140	1 Ave & E 79 St	40.771404
11	509	2018-04-03 18:31:27.000	2018-04-03 18:39:56.000	528	2 Ave & E 31 St	40.742901
12	454	2018-04-03 18:31:29.000	2018-04-03 18:39:04.000	248	Laight St & Hudson St	40.72185
13	2,268	2018-04-03 18:31:34.000	2018-04-03 19:09:20.000	334	W 20 St & 7 Ave	40.74238
14	358	2018-04-03 18:31:34.000	2018-04-03 18:37:32.000	418	Front St & Gold St	40.70

Now, let's look at some basic hourly statistics on Citi Bike usage. Run the query below in the worksheet. For each hour, it shows 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;
```

```

51  show warehouses;
52
53 --load data in with Large warehouse
54 copy into trips from @citibike_trips file_format=csv;
55
56 --preview trips data
57 select * from trips limit 20;
58
59 --hourly statistics on CitiBike usage
60 select date_trunc('hour', starttime) as "date",
61 count(*) as "num trips",
62 avg(tripduration)/60 as "avg duration (mins)",
63 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
64
65
66
67
68
69

```

	date	num trips	avg duration (mins)	avg distance (km)
1	2013-06-01 00:00:00	152	56.058442983333	2.127971476
2	2013-06-01 01:00:00	102	26.5251634	2.067906273
3	2013-06-01 02:00:00	67	36.1199005	2.31784827
4	2013-06-01 03:00:00	41	44.48536585	2.349126632
5	2013-06-01 04:00:00	16	23.278125	1.840026007
6	2013-06-01 05:00:00	13	34.584615383333	3.337844489
7	2013-06-01 06:00:00	40	22.3975	2.832927896
8	2013-06-01 07:00:00	93	66.397849466667	2.691774983
9	2013-06-01 08:00:00	177	18.530225983333	2.244399992
10	2013-06-01 09:00:00	280	40.2610119	2.292639556
11	2013-06-01 10:00:00	375	28.673466666667	2.22576826
12	2013-06-01 11:00:00	473	35.63520085	2.26058226
13	2013-06-01 12:00:00	604	33.8763521	2.177910979
14	2013-06-01 13:00:00	638	48.623093	2.270395192
15	2013-06-01 14:00:00	688	34.7400436	2.271981468

Query Details pane:

- Query duration: 1.3s
- Rows: 44.3K
- date: Histogram from 2013-06-01 to 2018-08-30
- num trips: Histogram from 1 to 8,810
- avg duration (mins): Histogram from 2.111111666667 to 5,898.822772866667
- avg distance (km): Histogram from 123 to 123

Use the Result Cache

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 **Query Details** pane on the right, note that the second query runs significantly faster because the results have been cached.

CITIBIKE_ZERO_TO_SNOWFLAKE

Pinned (0)
No pinned objects
Q Search CITIBIKE

```

61 count(*) as "num trips",
62 avg(tripduration)/60 as "avg duration (mins)",
63 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
64 from trips
65 group by 1 order by 1;
66
67 --run same query again to see performance improvement due to cache
68 select date_trunc('hour', starttime) as "date",
69 count(*) as "num trips",
70 avg(tripduration)/60 as "avg duration (mins)",
71 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
72 from trips
73 group by 1 order by 1;
74
75
76
77
78
79
80
81
82
83
84

```

Objects Query Results Chart

	date	num trips	...	avg duration (mins)	avg distance (km)
1	2013-06-01 00:00:00	152		56.058442983333	2.127971476
2	2013-06-01 01:00:00	102		26.5251634	2.067906273
3	2013-06-01 02:00:00	67		36.1199005	2.31794827
4	2013-06-01 03:00:00	41		44.48536585	2.349126632
5	2013-06-01 04:00:00	16		23.278125	1.840026007
6	2013-06-01 05:00:00	13		34.584615383333	3.337844489
7	2013-06-01 06:00:00	40		22.3975	2.832927896
8	2013-06-01 07:00:00	93		66.397849466667	2.691774983
9	2013-06-01 08:00:00	177		18.530225983333	2.244399992
10	2013-06-01 09:00:00	280		40.2610119	2.292639556
11	2013-06-01 10:00:00	375		28.673466666667	2.22576826
12	2013-06-01 11:00:00	473		35.63520085	2.26058226
13	2013-06-01 12:00:00	604		33.8763521	2.177910979
14	2013-06-01 13:00:00	638		48.623093	2.270395192
15	2013-06-01 14:00:00	688		34.7400436	2.271981468
...

Query Details

- Query duration: 55ms
- Rows: 44.3K
- Date: 2013-06-01 to 2018-08-30
- num trips: 123
- avg duration (mins): 123
- avg distance (km): 123

Execute Another Query

Next, let's run the following query to see which months are the busiest:

```

select
monthname(starttime) as "month",
count(*) as "num trips"
from trips
group by 1 order by 2 desc;

```

CITIBIKE_ZERO_TO_SNOWFLAKE

Pinned (0)
No pinned objects
Q Search CITIBIKE

```

61 select date_trunc('hour', starttime) as "date",
62 count(*) as "num trips",
63 avg(tripduration)/60 as "avg duration (mins)",
64 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
65 from trips
66 group by 1 order by 1;
67
68 --find busiest months
69 select
70 monthname(starttime) as "month",
71 count(*) as "num trips"
72 from trips
73 group by 1 order by 2 desc;
74
75
76
77
78
79
80
81
82
83
84

```

Objects Query Results Chart

month	num trips
Jun	7,800,765
Sep	6,804,899
Oct	6,550,164
Aug	6,516,652
May	6,388,309
Jul	6,013,644
Apr	4,959,244
Nov	4,719,798
Mar	3,405,178
Dec	3,349,319
Feb	2,817,291
Jan	2,541,096
...	...

Query Details

- Query duration: 1.2s
- Rows: 12
- month: Aa
- 100% filled
- num trips: 123

Clone a Table

Snowflake allows you to create clones, also known as "zero-copy clones" of tables, schemas, and databases in seconds. When a clone is created, Snowflake takes a snapshot of data present in the source object and makes it available to the cloned object. The cloned object is writable and independent of the clone source. Therefore, changes made to either the source object or the clone object are not included in the other.

A popular use case for zero-copy cloning is to clone a production environment for use by Development & Testing teams to test and experiment without adversely impacting the production environment and eliminating the need to set up and manage two separate environments.

Negative : Zero-Copy Cloning A massive benefit of zero-copy cloning is that the underlying data is not copied. Only the metadata and pointers to the underlying data change. Hence, clones are "zero-copy" and storage requirements are not doubled when the data is cloned. Most data warehouses cannot do this, but for Snowflake it is easy!

Run the following command in the worksheet to create a development (dev) table clone of the `trips` table:

```
create table trips_dev clone trips;
```

Click the three dots (...) in the left pane and select **Refresh**. Expand the object tree under the `CITIBIKE` database and verify that you see a new table named `trips_dev`. Your Development team now can do whatever they want with this table, including updating or deleting it, without impacting the `trips` table or any other object.

The screenshot shows the Snowflake UI interface. At the top, there's a navigation bar with a home icon and the database name "CITIBIKE_ZERO_TO_SNOWFLAKE". Below the navigation bar is a pinned objects section which is currently empty. A search bar and a three-dot menu are also present.

The main content area displays the database structure under the "CITIBIKE" schema. The "Tables" section is expanded, showing the "TRIPS" and "TRIPS_DEV" tables. The "TRIPS_DEV" table is selected and highlighted with a light gray background. A context menu is open over the "TRIPS_DEV" table, showing options like "clone" and "results".

On the right side, the table properties for "TRIPS_DEV" are displayed in a modal window. The properties listed are:

- Number of rows: 61.5M
- Size: 1.9GB
- Cluster Key: —
- Owner: SYSADMIN
- Created: 1 minute ago

Below the properties, there's a note: "Run same query again to see latest data".

```
57
58
59
60
61
62
63
64
65
66
67
68 select date_trunc('hour', starttime) as hour, count(*) as "num trips",
69   avg(tripduration)/60 as "avg trip duration"
70   avg(haversine(start_station_longitude, start_station_latitude,
71   end_station_longitude, end_station_latitude)) as "avg distance"
72 from trips
73 group by 1 order by 1;
74
75 --find busiest months
76 select monthname(starttime) as "month",
77   count(*) as "num trips"
78 from trips
79 group by 1 order by 2 desc;
```

Working with Semi-Structured Data, Views, & Joins

Positive : This section requires loading additional data and, therefore, provides a review of data loading while also introducing loading semi-structured data.

Going back to the lab's example, the Citi Bike analytics team wants to determine how weather impacts ride counts. To do this, in this section, we will:

- Load weather data in semi-structured JSON format held in a public S3 bucket.
 - Create a view and query the JSON data using SQL dot notation.
 - Run a query that joins the JSON data to the previously loaded `TRIPS` data.
 - Analyze the weather and ride count data to determine their relationship.

The JSON data consists of weather information provided by *MeteoStat* 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 consists of 75k rows, 36 objects, and 1.1MB compressed. If viewed in a text editor, the raw JSON in the GZ files looks like:

```
[{"temp":6.1,"dwpt":-3.3,"rhum":51.0,"prcp":null,"snow":null,"wdir":200.0,"wspd":27.7,"wpgt":null,"pres":1014.4,"tsun":null,"coco":null,"obsTime":"2017-01-01T00:00:00.000Z","stationID":"72502","name":"Newark Airport","country":"US","region":"NJ","wmo":"72502","icao":"KEWR","latitude":40.6833,"longitude":-74.0,"elevation":5.0,"timezone":"America/New_York","hourly_start":"1973-01-01T00:00:00.000Z","hourly_end":"2022-07-24T00:00:00.000Z","daily_start":"1893-01-01T00:00:00.000Z","daily_end":"2022-07-16T00:00:00.000Z","monthly_start":"1893-01T00:00:00.000Z","monthly_end":"2022-01-01T00:00:00.000Z","distance":7580.9151211035,"weatherCondition":null}, {"temp":6.1,"dwpt":-4.4,"rhum":47.0,"prcp":0.0,"snow":null,"wdir":210.0,"wspd":16.6,"wpgt":null,"pres":1013.9,"tsun":null,"coco":null,"obsTime":"2017-01-01T01:00:00.000Z","stationID":"72502","name":"Newark Airport","country":"US","region":"NJ","wmo":"72502","icao":"KEWR","latitude":40.6833,"longitude":-74.0,"elevation":5.0,"timezone":"America/New_York","hourly_start":"1973-01-01T00:00:00.000Z","hourly_end":"2022-07-24T00:00:00.000Z","daily_start":"1893-01-01T00:00:00.000Z","daily_end":"2022-01-01T00:00:00.000Z","monthly_start":"1893-01T00:00:00.000Z","monthly_end":"2022-01-01T00:00:00.000Z","distance":7580.9151211035,"weatherCondition":null}, {"temp":5.6,"dwpt":-4.3,"rhum":49.0,"prcp":0.0,"snow":null,"wdir":200.0,"wspd":18.4,"wpgt":null,"pres":1012.8,"tsun":null,"coco":null,"obsTime":"2017-01-01T03:00:00.000Z","stationID":"72502","name":"Newark Airport","country":"US","region":"NJ","wmo":"72502","icao":"KEWR","latitude":40.6833,"longitude":-74.0,"elevation":5.0,"timezone":"America/New_York","hourly_start":"1973-01-01T00:00:00.000Z","hourly_end":"2022-07-24T00:00:00.000Z","daily_start":"1893-01-01T00:00:00.000Z","daily_end":"2022-07-16T00:00:00.000Z","monthly_start":"1893-01T00:00:00.000Z","monthly_end":"2022-01-01T00:00:00.000Z","distance":7580.9151211035,"weatherCondition":null}, {"temp":6.1,"dwpt":-4.4,"rhum":47.0,"prcp":0.0,"snow":null,"wdir":230.0,"wspd":18.4,"wpgt":null,"pres":1012.8,"tsun":null,"coco":null,"obsTime":"2017-01-01T03:00:00.000Z","stationID":"72502","name":"Newark Airport","country":"US","region":"NJ","wmo":"72502","icao":"KEWR","latitude":40.6833,"longitude":-74.0,"elevation":5.0,"timezone":"America/New_York","hourly_start":"1973-01-01T00:00:00.000Z","hourly_end":"2022-07-24T00:00:00.000Z","daily_start":"1893-01-01T00:00:00.000Z","daily_end":"2022-07-16T00:00:00.000Z","monthly_start":"1893-01T00:00:00.000Z","monthly_end":"2022-01-01T00:00:00.000Z","distance":7580.9151211035,"weatherCondition":null} ]
```

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

Create a New Database and Table for the Data

First, in the worksheet, let's create a database named `WEATHER` to use for storing the semi-structured JSON data.

```
create database weather;
```

Execute the following USE commands to set the worksheet context appropriately:

```
use role sysadmin;

use warehouse compute_wh;

use database weather;

use schema public;
```

Positive : Executing Multiple Commands Remember that you need to execute each command individually. However, you can execute them in sequence together by selecting all of the commands and then clicking the **Play/Run** button (or using the keyboard shortcut).

Next, let's create a table named `JSON_WEATHER_DATA` to use for loading the JSON data. In the worksheet, execute the following CREATE TABLE command:

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

Note that Snowflake has a special column data type called `VARIANT` that allows storing the entire JSON object as a single row and eventually query the object directly.

Negative : Semi-Structured Data Magic The VARIANT data type allows Snowflake to ingest semi-structured data without having to predefine the schema.

In the results pane at the bottom of the worksheet, verify that your table, `JSON_WEATHER_DATA`, was created:

```

77
78
79      select
80        monthname(starttime) as "month",
81        count(*) as "num trips"
82      from trips
83      group by 1 order by 2 desc;
84
85      create table trips_dev clone trips;
86
87
88      create database weather;
89
90      use role sysadmin;
91
92      use warehouse compute_wh;
93
94      use database weather;
95
96      use schema public;
97
98
99      create table json_weather_data (v variant);
100
101
102
103
104
105

```

Objects Query Results Chart

status

1 Table JSON_WEATHER_DATA successfully created.

Query Details

Query duration 188ms

Rows 1

status

100% filled

Create Another External Stage

In the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet, use the following command to create a stage that points to the bucket where the semi-structured JSON data is stored on AWS S3:

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

Now let's take a look at the contents of the `nyc_weather` stage. Execute the following LIST command to display the list of files:

```
list @nyc_weather;
```

In the results pane, you should see a list of `.gz` files from S3:

```

WEATHER PUBLIC *
86  create table trips_dev clone trips;
87
88
89  create database weather;
90
91  use role sysadmin;
92
93  use warehouse compute_wh;
94
95  use database weather;
96
97  use schema public;
98
99  create table json_weather_data (v variant);
100
101
102  create stage nyc_weather
103  url = 's3://snowflake-workshop-lab/zero-weather-nyc';
104
105
106  list @nyc_weather;
107
108
109
110
111
112
113
114

```

name	size	md5	last_modified
1 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-10.json.gz	13,260	356d042ffac90934571e12fc7c348840	Mon, 25 Jul 2022 08:35
2 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-11.json.gz	12,930	dff30f516567061eb8d0fe9e953cd2f0	Mon, 25 Jul 2022 08:35
3 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-12.json.gz	13,574	4495224605a6a2bbebe5405bf690438	Mon, 25 Jul 2022 08:35
4 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-7.json.gz	11,760	35b5738b75b05620b58c636bf31ef7	Mon, 25 Jul 2022 08:35
5 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-8.json.gz	12,898	36849474cd97947462c4b69619b953ef	Mon, 25 Jul 2022 08:35
6 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-9.json.gz	12,656	50986ca04728278106982c5f354c10ed	Mon, 25 Jul 2022 08:35
7 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-1.json.gz	13,447	5d9664b36109e965ead983e17df827	Mon, 25 Jul 2022 08:35
8 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-10.json.gz	13,423	6eb5f864e4285bc93cb6431fc0d32d0	Mon, 25 Jul 2022 08:35
9 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-11.json.gz	13,055	4a24e38608b5dac9c8439d5767914090	Mon, 25 Jul 2022 08:35
10 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-12.json.gz	13,175	3ecb05ea4b17275e18d4ce398bc1a81	Mon, 25 Jul 2022 08:35
11 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-2.json.gz	12,432	70ba2fa598b81edd28700e02f0a887f	Mon, 25 Jul 2022 08:35
12 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-3.json.gz	14,065	cbb8cf0d224774e2e9017854ec67673ae	Mon, 25 Jul 2022 08:35
13 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-4.json.gz	13,276	7ca8af4ca3dfebc56d4691fd35c9f1	Mon, 25 Jul 2022 08:35
14 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-5.json.gz	13,461	976bf82ed554f9a9e33137cafc6600	Mon, 25 Jul 2022 08:35
15 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-6.json.gz	12,966	5cd814d9bf4e8e6ae9e9bbfbfb0c3f4	Mon, 25 Jul 2022 08:35
16 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-7.json.gz	13,187	834bc2452881ef07acf612d907355	Mon, 25 Jul 2022 08:35
17 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-8.json.gz	13,164	a5daad36c583664d55616704476c102e	Mon, 25 Jul 2022 08:35
18 e2://tennantdata/workshop-lab/zero-weather/nyc/hourlyData-2017-9.json.gz	12,360	10f31e40777a0077f6khakf358521ht	Mon, 25 Jul 2022 08:35

Query Details

- Query duration: 157ms
- Rows: 36
- name: 100% filled
- size: 123
- md5: 100% filled
- last_modified: 100% filled

Load and Verify the Semi-structured Data

In this section, we will use a warehouse to load the data from the S3 bucket into the `JSON_WEATHER_DATA` table we created earlier.

In the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet, execute the COPY command below to load the data.

Note that you can specify a `FILE FORMAT` object inline in the command. In the previous section where we loaded structured data in CSV format, we had to define a file format to support the CSV structure. Because the JSON data here is well-formed, we are able to simply specify the JSON type and use all the default settings:

```

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

```

Verify that each file has a status of `LOADED`:

CITIBIKE_ZERO_TO_SNOWFLAKE +

SYSADMIN COMPUTE_WH Share Updated 20 seconds ago

Worksheets Databases

Pinned (0)

No pinned objects

Q All Objects ...

CITIBIKE

> INFORMATION_SCHEMA

> PUBLIC

Tables

- TRIPS
- TRIPS_DEV

> Views

> Stages

> Pipes

> Streams

> Tasks

> Functions

> Procedures

SNOWFLAKE_SAMPLE_DATA

```

WEATHER.PUBLIC +
86   create table trips_dev clone trips;
87
88
89   create database weather;
90
91   use role sysadmin;
92
93   use warehouse compute_wh;
94
95   use database weather;
96
97   use schema public;
98
99
100  create table json_weather_data (v variant);
101
102  create stage nyc_weather
103    url = 's3://snowflake-workshop-lab/zero-weather-nyc';
104
105  list @nyc.weather;
106
107
108
109
110  copy into json_weather_data
111    from @nyc.weather
112      file_format = (type = json strip_outer_array = true);
113
114

```

Objects Query Results Chart

file	status	rows_parsed	rows_loaded	error_limit	errors
1 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2019-3.json.gz	LOADED	744	744	1	
2 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-10.json.gz	LOADED	744	744	1	
3 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2019-2.json.gz	LOADED	672	672	1	
4 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-6.json.gz	LOADED	720	720	1	
5 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-9.json.gz	LOADED	720	720	1	
6 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-11.json.gz	LOADED	720	720	1	
7 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-7.json.gz	LOADED	648	648	1	
8 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-5.json.gz	LOADED	744	744	1	
9 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-4.json.gz	LOADED	720	720	1	
10 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2016-8.json.gz	LOADED	744	744	1	
11 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-3.json.gz	LOADED	744	744	1	
12 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-11.json.gz	LOADED	720	720	1	
13 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-2.json.gz	LOADED	672	672	1	
14 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-4.json.gz	LOADED	720	720	1	
15 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-10.json.gz	LOADED	744	744	1	
16 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-2.json.gz	LOADED	672	672	1	
17 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2017-5.json.gz	LOADED	744	744	1	
18 s3://snowflake-workshop-lab/zero-weather-nyc/hourlyData-2018-7.json.gz	LOADED	744	744	1	

Query Details

Query duration 4.2s

Rows 36

file 100% filled

status AA

status LOADED 36

rows_parsed 123

rows_parsed 577 744

rows_loaded 123

rows_loaded 577 744

error_limit 123

error_limit 100% filled

Now, let's take a look at the data that was loaded:

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

Click any of the rows to display the formated JSON in the right panel:

```

    use warehouse compute_wh;
    use database weather;
    use schema public;

    create or replace table json_weather_data (v variant);

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

    select * from json_weather_data limit 10;

```

V
{ "coco": 3, "country": "US", "dwpt": 12.8, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 3, "country": "US", "dwpt": 12.8, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 3, "country": "US", "dwpt": 11.8, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 3, "country": "US", "dwpt": 11.7, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 3, "country": "US", "dwpt": 11.7, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 4, "country": "US", "dwpt": 11.1, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 3, "country": "US", "dwpt": 11.8, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 4, "country": "US", "dwpt": 11.8, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 4, "country": "US", "dwpt": 11.7, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }
{ "coco": 4, "country": "US", "dwpt": 11.7, "elevation": 4, "icao": "KJFK", "latitude": 40.6333, "longitude": -73.7667, "name": "John F. Kennedy Airport", "obsTime": 1510480000000 }

To close the display in the panel and display the query details again, click the **X** (Close) button that appears when you hover your mouse in the right corner of the panel.

Create a View and Query Semi-Structured Data

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

Negative : Views & Materialized Views A view allows the result of a query to be accessed as if it were a table. Views can help present data to end users in a cleaner manner, limit what end users can view in a source table, and write more modular SQL. Snowflake also supports materialized views in which the query results are stored as though the results are a table. This allows faster access, but requires storage space. Materialized views can be created and queried if you are using Snowflake Enterprise Edition (or higher).

Run the following command to create a columnar view of the semi-structured JSON weather data so it is easier for analysts to understand and query. The `72502` value for `station_id` corresponds to Newark Airport, the closest station that has weather conditions for the whole period.

```

// create a view that will put structure onto the semi-structured data
create or replace view json_weather_data_view as
select
    v:obsTime::timestamp as observation_time,
    v:station::string as station_id,
    v:name::string as city_name,
    v:country::string as country,
    v:latitude::float as city_lat,
    v:longitude::float as city_lon,

```

```

v:weatherCondition::string as weather_conditions,
v:coco::int as weather_conditions_code,
v:temp::float as temp,
v:prcp::float as rain,
v:tsun::float as tsun,
v:wdir::float as wind_dir,
v:wspd::float as wind_speed,
v:dwpt::float as dew_point,
v:rhum::float as relative_humidity,
v:pres::float as pressure
from
    json_weather_data
where
    station_id = '72502';

```

SQL dot notation `v:temp` is used in this command to pull out values at lower levels within the JSON object hierarchy. This allows us to treat each field as if it were a column in a relational table.

The new view should appear as `JSON_WEATHER_DATA` under `WEATHER > PUBLIC > Views` in the object browser on the left. You may need to expand or refresh the objects browser in order to see it.

WEATHER.PUBLIC > JSON_WEATHER_DATA_VIEW

Details **Definition**

Type: View
Number of columns: 16
Owner: SYSADMIN
Created: 19 minutes ago

	OBSERVATION_TIME	STATION_ID	CITY_NAME	COUNTRY	CITY_LAT	CITY_LON	WEATHER_CONDITIONS
1	2018-01-01 00:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
2	2018-01-01 01:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
3	2018-01-01 02:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
4	2018-01-01 03:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
5	2018-01-01 04:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
6	2018-01-01 05:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
7	2018-01-01 06:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
8	2018-01-01 07:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
9	2018-01-01 08:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
10	2018-01-01 09:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
11	2018-01-01 10:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
12	2018-01-01 11:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
13	2018-01-01 12:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
14	2018-01-01 13:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
15	2018-01-01 14:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
16	2018-01-01 15:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
17	2018-01-01 16:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear
18	2018-01-01 17:00:00.000	72502	Newark Airport	US	40.6833	-74	Clear

Verify the view with the following query:

```

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

```

```
limit 20;
```

Notice the results look just like a regular structured data source. Your result set may have different `observation_time` values:

```
WEATHER.PUBLIC *  
...  
126     v:country::string as country,  
127     v:latitude::float as city_lat,  
128     v:longitude::float as city_lon,  
129     v:weatherConditions::string as weather_conditions,  
130     v:weatherConditionsCode::string as weather_conditions_code,  
131     v:temp::float as temp,  
132     v:precip::float as rain,  
133     v:tsun::float as tsun,  
134     v:wdir::float as wind_dir,  
135     v:wspeed::float as wind_speed,  
136     v:dewpt::float as dew_point,  
137     v:rhum::float as relative_humidity,  
138     v:pres::float as pressure  
139     from json_weather_data  
140     where station_id = '72502';  
141  
142  
143  
144  
145  
146     select * from json_weather_data_view  
147     where date_trunc('month', observation_time) = '2018-01-01'  
148     limit 20;  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933
```

```

134     v:wind::float as wind_dir,
135     v:spd::float as wind_speed,
136     v:dept::float as dew_point,
137     v:rhum::float as relative_humidity,
138     v:pres::float as pressure
139
140     from json_weather_data
141     where
142         station_id = '72502';
143
144
145
146     select * from json_weather_data_view
147     where date_trunc('month',observation_time) = '2018-01-01'
148
149     limit 20;
150
151
152     select weather_conditions as conditions
153     ,count(*) as num_trips
154     from citibike.public.trips
155     left outer join json_weather_data_view
156     on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
157     where conditions is not null
158     group by 1 order by 2 desc;
159
160
161
162

```

CONDITIONS	NUM_TRIPS
1 Clear	27,651,743
2 Light Rain	1,577,552
3 Fair	717,754
4 Cloudy	676,990
5 Fog	606,701
6 Overcast	354,975
7 Rain	132,279
8 Light Snowfall	110,083
9 Thunderstorm	83,491
10 Heavy Rain	60,989
11 Heavy Thunderstorm	19,088
12 Sleet	12,398
13 Snowfall	5,123
14 Heavy Freezing Rain	1,068
15 Freezing Rain	1,003
16 Storm	653
17 Heavy Sleet	564
18 Heavy Snowfall	98

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

Using Time Travel

Snowflake's powerful Time Travel feature enables accessing historical data, as well as the objects storing the data, at any point within a period of time. The default window is 24 hours and, if you are using Snowflake Enterprise Edition, can be increased up to 90 days. Most data warehouses cannot offer this functionality, but - you guessed it - Snowflake makes it easy!

Some useful applications include:

- Restoring data-related objects such as tables, schemas, and databases that may have been deleted.
- Duplicating and backing up data from key points in the past.
- Analyzing data usage and manipulation over specified periods of time.

Drop and Undrop a Table

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

In the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet, run the following DROP command to remove the `JSON_WEATHER_DATA` table:

```
drop table json_weather_data;
```

Run a query on the table:

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

In the results pane at the bottom, you should see an error because the underlying table has been dropped:

The screenshot shows the Snowflake UI interface. On the left is the object browser with a tree view of schemas and tables. In the center is a query editor window containing the following code:

```
140 --vert user jvan json_weather_data_view
141 on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
142 where conditions is not null
143 group by 1 order by 2 desc;
144
145 ///////////////////////////////////////////////////////////////////
146
147 --drop table
148 drop table json_weather_data;
149
150 --verify table is dropped
151 select * from json_weather_data limit 10;
```

The last line of the query, `select * from json_weather_data limit 10;`, is highlighted in blue. Below the query editor, the results pane shows an error message:

Object 'JSON_WEATHER_DATA' does not exist or not authorized.

On the right side of the screen, there is a sidebar titled "Query Details" which shows the following information:

- Query duration: 32ms
- Rows: 0

Now, restore the table:

```
undrop table json_weather_data;
```

The json_weather_data table should be restored. Verify by running the following query:

```
--verify table is undropped

select * from json_weather_data_view limit 10;
```

```

152
153
154 ///////////////////////////////////////////////////////////////////
155 //MODULE 8//  

156 ///////////////////////////////////////////////////////////////////
157 --drop table json_weather_data;
158
159 --verify table is dropped
160 select * from json_weather_data limit 10;
161
162
163 --undrop the table
164 undrop table json_weather_data;
165
166 --verify table is undropped
167 select * from json_weather_data_view limit 10;
168
169
170

```

	OBSERVATION_TIME	CITY_ID	CITY_NAME	COUNTRY	CITY_LAT	CITY_LON	CLOUDS	TEMP_AVG	TEMP
1	2016-10-23 07:55:33.000	5,128,638	New York	US	43.000351	-75.499901	88	4.723	4
2	2016-10-20 05:51:43.000	5,128,638	New York	US	43.000351	-75.499901	20	8.623	8
3	2016-10-22 06:41:04.000	5,128,638	New York	US	43.000351	-75.499901	92	7.309	7
4	2016-09-13 11:25:35.000	5,128,638	New York	US	43.000351	-75.499901	1	12.12	
5	2016-09-08 23:03:53.000	5,128,638	New York	US	43.000351	-75.499901	40	25.3	2
6	2016-09-05 18:25:39.000	5,128,638	New York	US	43.000351	-75.499901	1	27.82	2
7	2016-09-15 03:04:10.000	5,128,638	New York	US	43.000351	-75.499901	1	13.22	1
8	2016-11-26 21:07:52.000	5,128,638	New York	US	43.000351	-75.499901	90	3.33	
9	2016-08-06 23:31:04.000	5,128,638	New York	US	43.000351	-75.499901	44	25.29	2
10	2016-08-10 01:24:18.000	5,128,638	New York	US	43.000351	-75.499901	0	24.449	24

Roll Back a Table

Let's roll back the `TRIPS` table in the `CITIBIKE` database to a previous state to fix an unintentional DML error that replaces all the station names in the table with the word "oops".

First, run the following SQL statements to switch your worksheet to the proper context:

```

use role sysadmin;

use warehouse compute_wh;

use database citibike;

use schema public;

```

Run the following command to replace all of the station names in the table with the word "oops":

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

Now, run a query that returns the top 20 stations by number of rides. Notice that the station names result contains only one row:

```

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

```

```

167  select * from json_weather_data_view limit 10;
168
169  set context;
170  use role sysadmin;
171  use warehouse compute_wh;
172  use database citibike;
173  use schema public;
174
175  --make a mistake by changing the column name
176  update trips set start_station_name='oops';
177
178  --query for top 20 stations by total number of rides
179  select start_station_name as "station",
180        count(*) as "rides"
181  from trips
182  group by 1
183  order by 2 desc
184
185  limit 20;

```

station	rides
oops	61,468,359

Query Details

- Query duration: 315ms
- Rows: 1
- station: 100% filled
- rides: 100% filled

Normally we would need to scramble and hope we have a backup lying around.

In Snowflake, we can simply run a command to find the query ID of the last UPDATE command and store it in a variable named `$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 desc limit 1);

```

Use Time Travel to recreate the table with the correct station names:

```

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

```

Run the previous query again to verify 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;

```

```

--find the query id that was the last update to the table
187 set query_id =
188 (select query_id from table(information_schema.query_history_by_session (result_limit>=5))
189 where query.text Like '%update%' order by start_time Limit 1);
190
191 --recreate the table with proper station names
192 create or replace table trips as
193 (select * from trips before (statement => $query_id));
194
195 --select from the restored table to verify
196 select
197 start_station_name as "station",
198 count(*) as "rides"
199 from trips
200 group by 1
201 order by 2 desc
202 limit 20;
203
204

```

station	rides
Pershing Square North	491,951
E 17 St & Broadway	481,065
W 21 St & 6 Ave	458,626
8 Ave & W 31 St	438,001
West St & Chambers St	432,518
Broadway & E 22 St	421,812
Lafayette St & E 8 St	397,724
Broadway & E 14 St	394,995
8 Ave & W 33 St	379,843
W 41 St & 8 Ave	359,838
Cleveland Pt & Spring St	358,485
W 20 St & 11 Ave	352,099
Caroline St & 6 Ave	348,158
University Pl & E 14 St	332,803
Greenwich Ave & 8 Ave	329,347

Query Details

- Query duration: 370ms
- Rows: 20

station

100% filled

rides

310,509 491,951

Working with Roles, Account Admin, & Account Usage

In this section, we will explore aspects of Snowflake's access control security model, such as creating a role and granting it specific permissions. We will also explore other usage of the ACCOUNTADMIN (Account Administrator) role, which was briefly introduced earlier in the lab.

Continuing with the lab 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.

Negative : Role-Based Access Control Snowflake offers very powerful and granular access control that dictates the objects and functionality a user can access, as well as the level of access they have.

Create a New Role and Add a User

In the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet, switch to the ACCOUNTADMIN role to create a new role. ACCOUNTADMIN encapsulates the SYSADMIN and SECURITYADMIN system-defined roles. It is the top-level role in the account and should be granted only to a limited number of users.

In the `CITIBIKE_ZERO_TO_SNOWFLAKE` worksheet, run the following command:

```
use role accountadmin;
```

Notice that, in the top right of the worksheet, the context has changed to ACCOUNTADMIN:

```

287   order by z desc
288   limit 20;
289
290  ///////////////////////////////////////////////////////////////////
291  //MODULE v1
292  ///////////////////////////////////////////////////////////////////
293  --assume account admin role
294  use role accountadmin;
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320

```

status

1 Statement executed successfully.

Query Details

- Query duration: 51ms
- Rows: 1

status

100% filled

Before a role can be used for access control, at least one user must be assigned to it. So let's create a new role named `JUNIOR_DBA` and assign it to your Snowflake user. To complete this task, you need to know your username, which is the name you used to log in to the UI.

Use the following commands to create the role and assign it to you. Before you run the GRANT ROLE command, replace `YOUR_USERNAME_GOES_HERE` with your username:

```

create role junior_dba;

grant role junior_dba to user YOUR_USERNAME_GOES_HERE;

```

Positive : If you try to perform this operation while in a role such as `SYSADMIN`, it would fail due to insufficient privileges. By default (and design), the `SYSADMIN` role cannot create new roles or users.

Change your worksheet context to the new `JUNIOR_DBA` role:

```
use role junior_dba;
```

In the top right of the worksheet, notice that the context has changed to reflect the `JUNIOR_DBA` role.

```

284
285
286 ///////////////////////////////////////////////////////////////////
287 //MODULE 9//
288 ///////////////////////////////////////////////////////////////////
289 --assume account admin role
290 use role accountadmin;
291
292 --create new role and grant role to current user
293 create role junior_dba;
294 grant role junior_dba to user SF_USER;
295
296 --execute junior_dba role
297 use role junior_dba;
298
299
300
301
302
303

```

The screenshot shows a Snowflake SQL editor interface. The top bar includes a database dropdown set to 'CITIBIKE_ZERO_TO_SNOWFLAKE', a role dropdown set to 'JUNIOR_DBA', and buttons for 'Share' and 'Run as...'. A status message at the top right says 'Updated 21 seconds ago' and 'Draft'. The main area has tabs for 'Objects', 'Query', and 'Results'. The 'Query' tab is active, displaying the provided SQL code. A blue bar highlights the line 'use role junior_dba;'. A tooltip 'JUNIOR_DBA required to view results' appears over the 'Results' tab. On the left, a sidebar shows 'No databases' and a note 'Objects unavailable. Select a different role.'

Also, the warehouse is not selected because the newly created role does not have usage privileges on any warehouse. Let's fix it by switching back to ADMIN role and grant usage privileges to `COMPUTE_WH` warehouse.

```

use role accountadmin;

grant usage on warehouse compute_wh to role junior_dba;
```

Switch back to the `JUNIOR_DBA` role. You should be able to use `COMPUTE_WH` now.

```

use role junior_dba;

use warehouse compute_wh;
```

Finally, you can notice that in the database object browser panel on the left, the `CITIBIKE` and `WEATHER` databases no longer appear. This is because the `JUNIOR_DBA` role does not have privileges to access them.

Switch back to the ACCOUNTADMIN role and grant the `JUNIOR_DBA` the USAGE privilege required 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;
```

Switch to the `JUNIOR_DBA` role:

```
use role junior_dba;
```

Notice that the `CITIBIKE` and `WEATHER` databases now appear in the database object browser panel on the left. If they don't appear, try clicking ... in the panel, then clicking **Refresh**.

CITIBIKE_ZERO_TO_SNOWFLAKE

Search

Objects unavailable. Select a different role.

CITIBIKE WEATHER

```
use role junior_dbadmin;
--change role to give permissions to junior_dbadmin role
use role accountadmin;
grant usage on database citibike to junior_dbadmin;
grant usage on database weather to junior_dbadmin;
switch back to junior_dbadmin
use role junior_dbadmin;
```

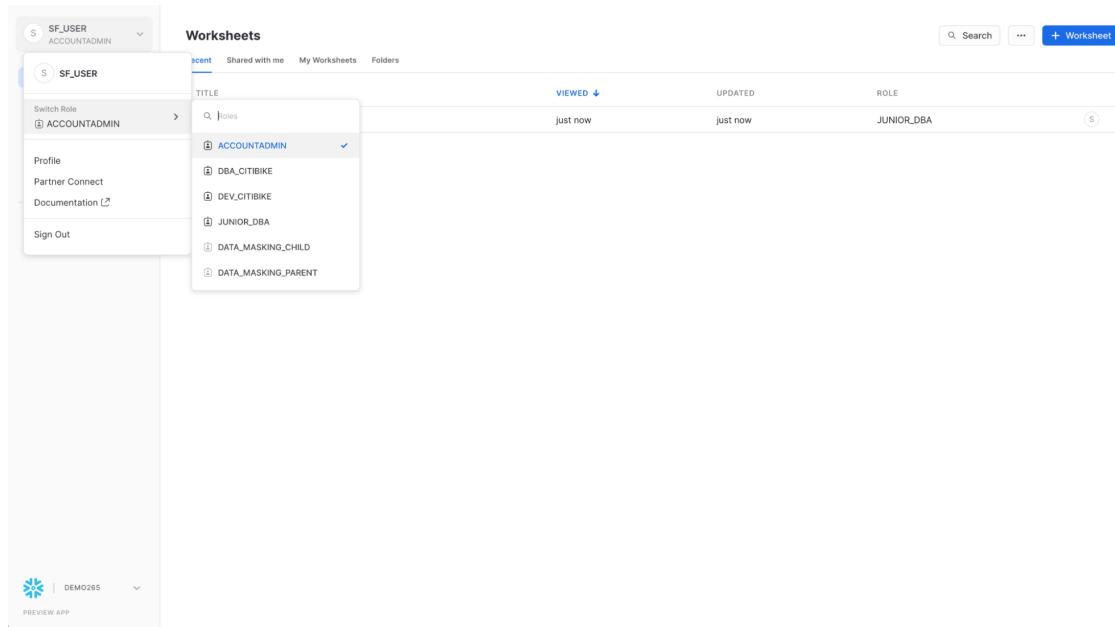
Objects Query Results Chart

JUNIOR_DBA required to view results

View the Account Administrator UI

Let's change our access control role back to `ACCOUNTADMIN` to see other areas of the UI accessible only to this role. However, to perform this task, use the UI instead of the worksheet.

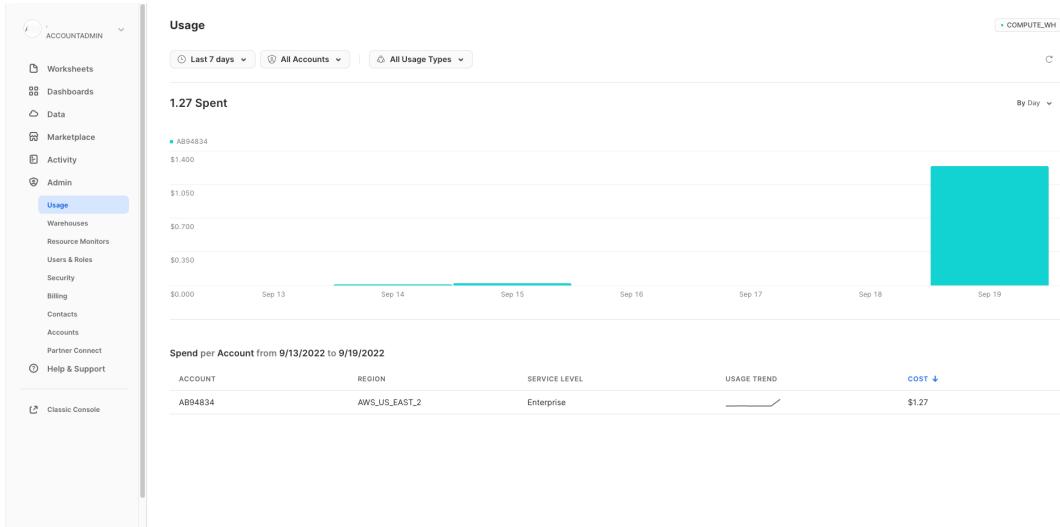
First, click the **Home** icon in the top left corner of the worksheet. Then, in the top left corner of the UI, click your name to display the user preferences menu. In the menu, go to **Switch Role** and select ACCOUNTADMIN.



Negative : Roles in User Preference vs Worksheet Why did we use the user preference menu to change the role instead of the worksheet? The UI session and each worksheet have their own separate roles. The UI session role controls the elements you can see and access in the UI, whereas the worksheet role controls only the objects and actions you can access within the role.

Notice that once you switch the UI session to the ACCOUNTADMIN role, new tabs are available under **Admin**.

Usage



The **Usage** tab shows the following, each with their own page:

- **Organization:** Credit usage across all the accounts in your organization.
- **Consumption:** Credits consumed by the virtual warehouses in the current account.
- **Storage:** Average amount of data stored in all databases, internal stages, and Snowflake Failsafe in the current account for the past month.
- **Transfers:** Average amount of data transferred out of the region (for the current account) into other regions for the past month.

The filters in the top right corner of each page can be used to break down the usage/consumption/etc. visualizations by different measures.

Security

The screenshot shows the Snowflake interface with the left sidebar expanded. The 'Security' tab is highlighted in blue. The main content area is titled 'Network Policies' and 'Sessions'. At the top right is a blue button labeled '+ Network Policy'. Below the title, there's a small icon of a document with a lock. A section titled 'No network policies' contains the text: 'Restrict or allow access to your Snowflake account based on IP address.' followed by a 'Learn More' link.

The **Security** tab contains network policies created for the Snowflake account. New network policies can be created by selecting "+ Network Policy" at the top right hand side of the page.

Billing

The screenshot shows the Snowflake interface with the left sidebar expanded. The 'Billing' tab is highlighted in blue. The main content area is titled 'Billing' and 'Payment Method'. It includes a note: 'Add a credit card to continue using Snowflake once your free trial ends.' and a blue button labeled '+ Credit Card'.

The **Billing** tab contains the payment method for the account:

- If you are a Snowflake contract customer, the tab shows the name associated with your contract information.
- If you are an on-demand Snowflake customer, the tab shows the credit card used to pay month-to-month, if one has been entered. If no credit card is on file, you can add one to continue using Snowflake when your trial ends.

For the next section, stay in the ACCOUNTADMIN role for the UI session.

Sharing Data Securely & the Data Marketplace

Snowflake enables data access between accounts through the secure data sharing features. Shares are created by data providers and imported by data consumers, either through their own Snowflake account or a provisioned Snowflake Reader account. The consumer can be an external entity or a different internal business unit that is required to have its own unique Snowflake account.

With secure data sharing:

- There is only one copy of the data that 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 to shares.
- Data sharing is simple and safe, especially compared to older data sharing methods, which were often manual and insecure, such as transferring large `.csv` files across the internet.

Snowflake uses secure data sharing to provide account usage data and sample data sets to all Snowflake accounts. In this capacity, Snowflake acts as the data provider of the data and all other accounts.

Secure data sharing also powers the Snowflake Data Marketplace, which is available to all Snowflake customers and allows you to discover and access third-party datasets from numerous data providers and SaaS vendors. Again, in this data sharing model, the data doesn't leave the provider's account and you can use the datasets without any transformation.

View Existing Shares

In the home page, navigate to **Data > Databases**. In the list of databases, look at the **SOURCE** column. You should see two databases with `Local` in the column. These are the two databases we created previously in the lab. The other database, `SNOWFLAKE`, shows `Share` in the column, indicating it's shared from a provider.

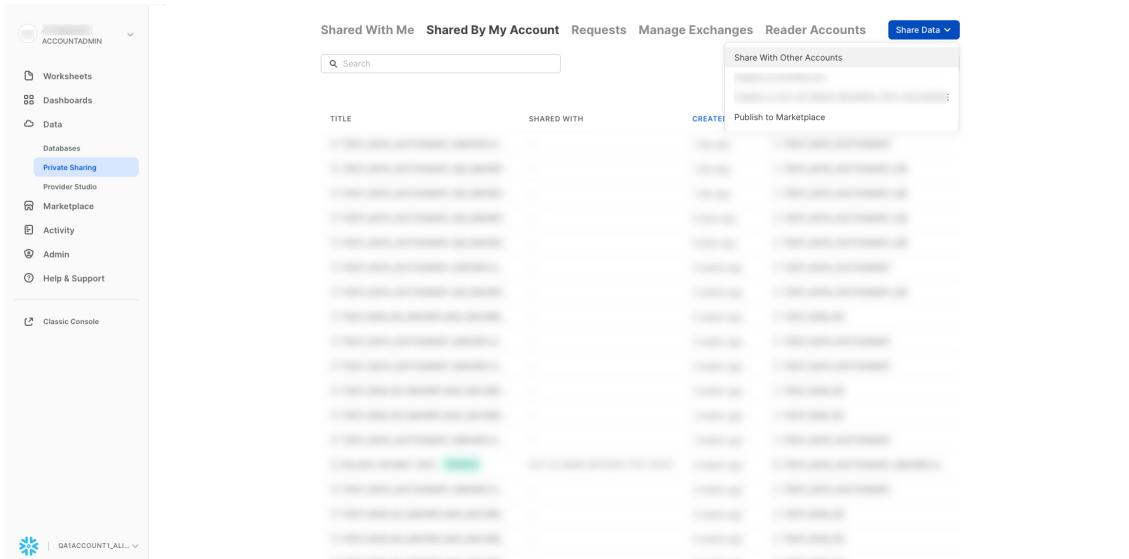
The screenshot shows the Snowflake Data Marketplace interface. On the left, there is a sidebar with navigation links: Worksheets, Dashboards, Databases (which is selected), Shared Data, Marketplace, Compute, and Account. Below the sidebar, it says "Classic Console". At the bottom, there is a preview app icon with "DEMO265" and "PREVIEW APP". The main area is titled "Databases" and shows a list of "3 Databases". The table has columns: NAME, SOURCE, OWNER, and CREATED. The data is as follows:

NAME	SOURCE	OWNER	CREATED
CITIBIKE	Local	SYSADMIN	13 hours ago
SNOWFLAKE	Share	—	9 months ago
WEATHER	Local	SYSADMIN	53 minutes ago

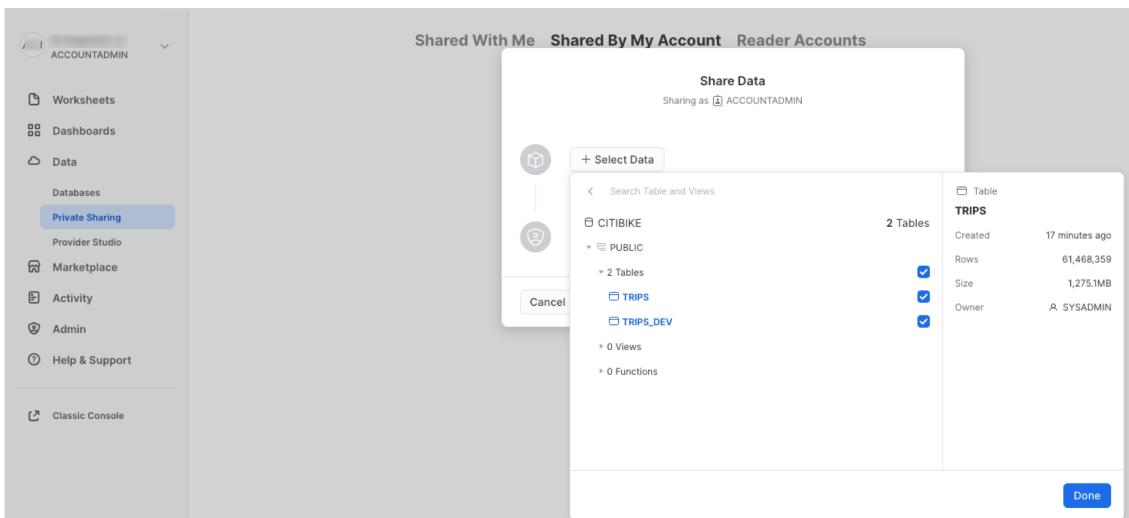
Create an Outbound Share

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 analyze the data in our `TRIPS` database on a near real-time basis. This partner also has their own Snowflake account in the same region as our account. So let's use secure data sharing to allow them to access this information.

Navigate to **Data > Shared Data**, then click **Shared by My Account** at the top of the tab. Click the **Share Data** button in the top right corner and select **Share with Other Accounts**:



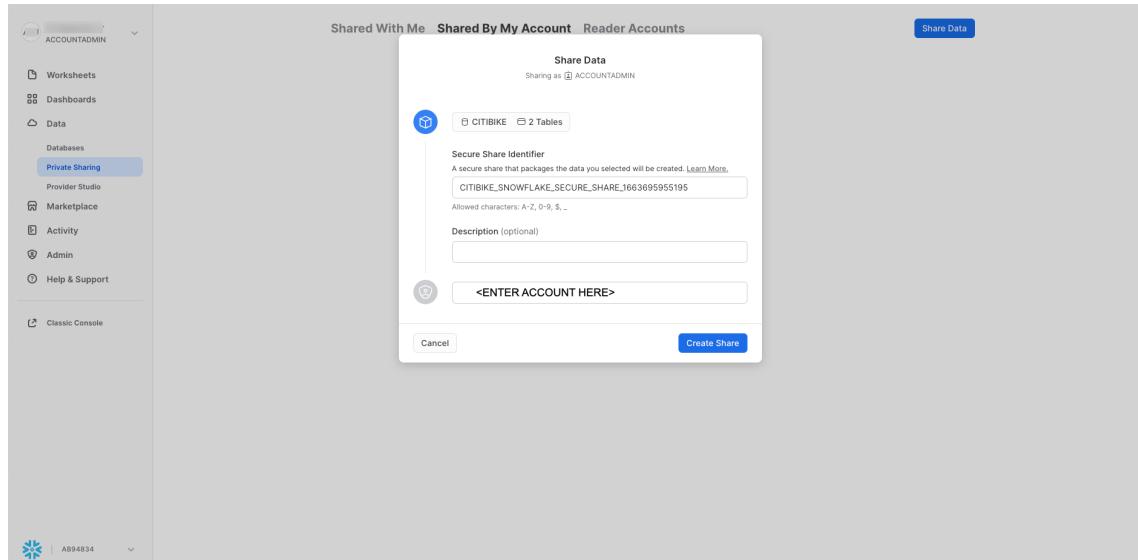
Click **+ Data** and navigate to the `CITIBIKE` database and `PUBLIC` schema. Select the 2 tables we created in the schema and click the **Done** button:



The default name of the share is a generic name with a random numeric value appended. Edit the default name to a more descriptive value that will help identify the share in the future (e.g. `ZERO_TO_SNOWFLAKE_SHARED_DATA`). You can also add a comment.

In a real-world scenario, the Citi Bike Account Administrator would next add one or more consumer accounts to the share, but we'll stop here for the purposes of this lab.

Click the **Create Share** button at the bottom of the dialog:



The dialog closes and the page shows the secure share you created:

A screenshot of the "ZERO_TO_SNOWFLAKE_DATA_SHARE" secure share details page. The page includes sections for "Shared With", "Description" (sharing data for zero to snowflake), and "Data" (listing two tables: TRIPS and TRIPS_DEV). The "Data" section has an "Edit" button.

You can add consumers, add/change the description, and edit the objects in the share at any time. In the page, click the < button next to the share name to return to the **Share with Other Accounts** page:

The screenshot shows the Snowflake Data Marketplace interface. On the left, there's a sidebar with navigation links: Worksheets, Dashboards, Data (with Shared Data selected), Databases, Marketplace, Compute, and Account. Below the sidebar, it says 'DEMO265' and 'PREVIEW APP'. At the top, there are tabs for 'Shared With Me', 'Shared By My Account' (which is selected), Requests, Manage Exchanges, Reader Accounts, and a 'Share Data' button. A search bar is present, along with filters for 'Shared with All' and 'All Types'. The main area displays a table with columns: TITLE, SHARED WITH, CREATED (sorted by time), and DATA. One entry is visible: 'ZERO_TO_SNOWFLAKE_SHARED_DATA' was shared with 'CITIBIKE' one minute ago.

We've demonstrated how it only takes seconds to give other accounts access to data in your Snowflake account in a secure manner with no copying or transferring of data required!

Snowflake provides several ways to securely share data without compromising confidentiality. In addition to tables, you can share secure views, secure UDFs (user-defined functions), and other secure objects. For more details about using these methods to share data while preventing access to sensitive information, see the [Snowflake documentation].

Snowflake Data Marketplace

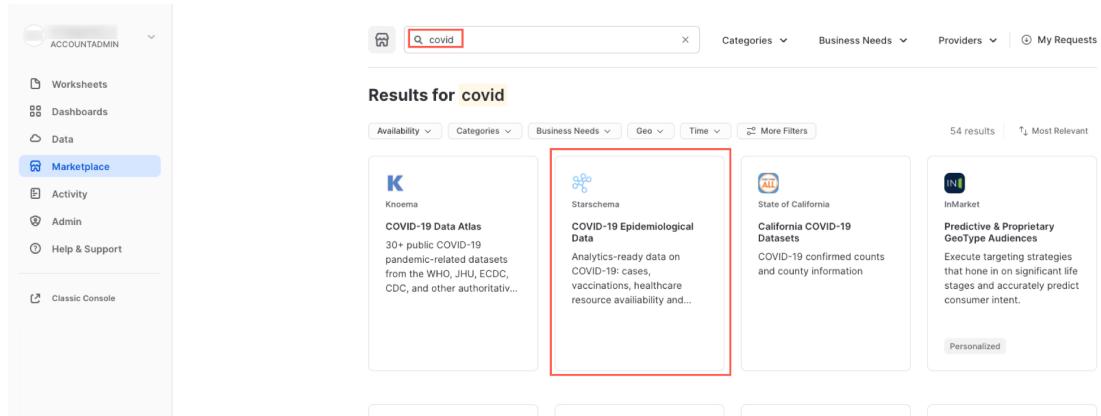
Make sure you're using the ACCOUNTADMIN role and, navigate to the **Marketplace**:

The screenshot shows the Snowflake Data Marketplace interface. On the left, there's a sidebar with navigation links: Worksheets, Dashboards, Data (with Marketplace selected), Activity, Admin, Help & Support, and a 'Classic Console' link. At the top, there's a search bar and filters for 'Categories', 'Business Needs', 'Providers', and 'My Requests'. Below the search bar, there's a promotional message: 'Together, we can build a sustainable future' with a call to join the Data Collaboration for the Climate initiative. There are four featured provider cards: OAG Emissions Data, GaiaLens Environmental Data..., Actionable Weather Forecasts (Demo.. AccuWeather, and OECD Greenhouse Gas Emissions Kroema. Below this, there's a 'Featured Providers' section with cards for Stripe, HubSpot, Braze, and iPinfo. At the bottom, there's a 'Most Popular' section with cards for Starschema COVID-19 Epidemiological Data, Starschema Worldwide Address Data, Knoema Economy Data Atlas, and EDI Exchange Data International SAMPLE: EDI Foreign Exchange Rates.

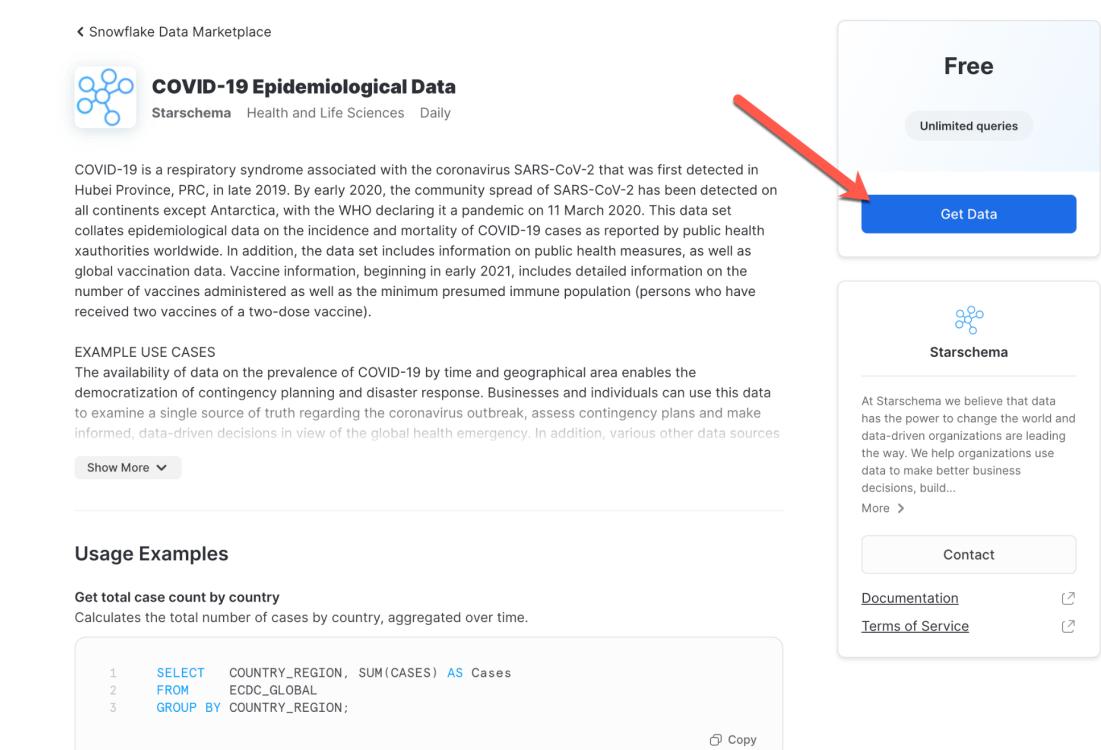
Find a listing

The search box at the top allows you to search for a listings. The drop-down lists to the right of the search box let you filter data listings by Provider, Business Needs, and Category.

Type **COVID** in the search box, scroll through the results, and select **COVID-19 Epidemiological Data** (provided by Starschema).



In the **COVID-19 Epidemiological Data** page, you can learn more about the dataset and see some usage example queries. When you're ready, click the **Get Data** button to make this information available within your Snowflake account:



COVID-19 Epidemiological Data
Starschema Health and Life Sciences Daily

COVID-19 is a respiratory syndrome associated with the coronavirus SARS-CoV-2 that was first detected in Hubei Province, PRC, in late 2019. By early 2020, the community spread of SARS-CoV-2 has been detected on all continents except Antarctica, with the WHO declaring it a pandemic on 11 March 2020. This data set collates epidemiological data on the incidence and mortality of COVID-19 cases as reported by public health authorities worldwide. In addition, the data set includes information on public health measures, as well as global vaccination data. Vaccine information, beginning in early 2021, includes detailed information on the number of vaccines administered as well as the minimum presumed immune population (persons who have received two vaccines of a two-dose vaccine).

EXAMPLE USE CASES
The availability of data on the prevalence of COVID-19 by time and geographical area enables the democratization of contingency planning and disaster response. Businesses and individuals can use this data to examine a single source of truth regarding the coronavirus outbreak, assess contingency plans and make informed, data-driven decisions in view of the global health emergency. In addition, various other data sources

Show More ▾

Usage Examples

Get total case count by country
Calculates the total number of cases by country, aggregated over time.

```
1  SELECT COUNTRY_REGION, SUM(CASES) AS Cases
2  FROM ECDC_GLOBAL
3  GROUP BY COUNTRY_REGION;
```

Copy

Free
Unlimited queries
Get Data

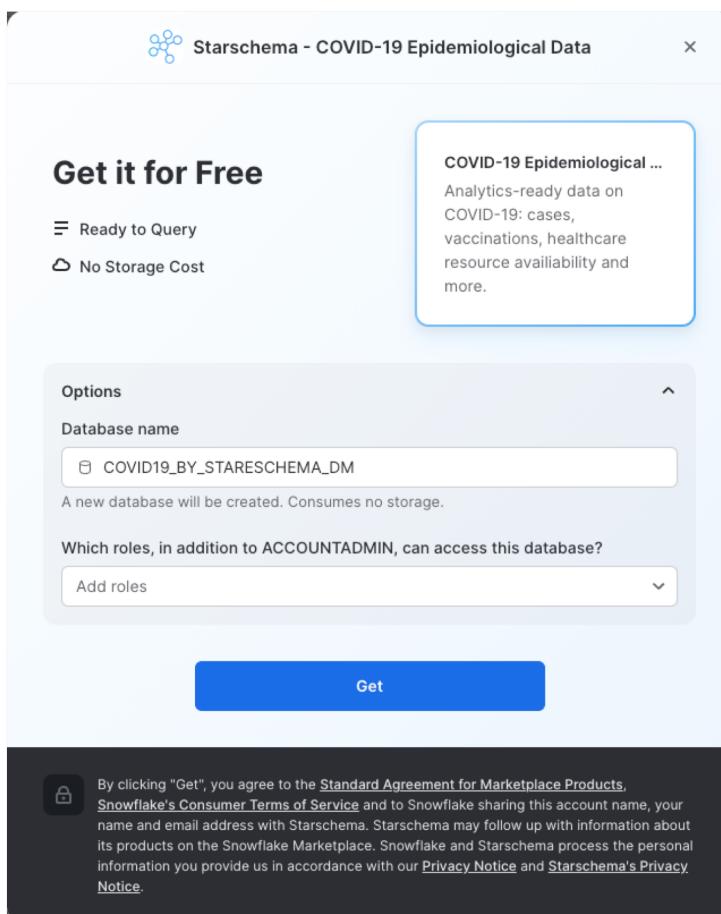
Starschema

At Starschema we believe that data has the power to change the world and data-driven organizations are leading the way. We help organizations use data to make better business decisions, build...

More ▾

Contact
Documentation Terms of Service

Review the information in the dialog and click **Get** again:



You can now click **Done** or choose to run the sample queries provided by Starschema:



Data is Ready to Query

COVID19_BY_STARESCHEMA_DM

Open ↗

Done

If you chose **Open**, a new worksheet opens in a new browser tab/window:

1. Select the query you want to run (or place your cursor in the query text).
2. Click the **Run/Play** button (or use the keyboard shortcut).
3. You can view the data results in the bottom pane.
4. When you are done running the sample queries, click the **Home** icon in the upper left corner.

Pinned (0)
No pinned objects

Databases

- APPLOG
- CITIBIKE
- CITIBIKE_RESET
- CITYBIKE2
- COVID19_BY_STARSCHEMA_DM
- ECONOMY_DATA_ATLAS
- JASON
- JASON_ANALYST_DB
- JASON_DATA
- JSON_DEMO
- KNOEMA_POVERTY_DATA_ATLAS
- MOVIELENS
- NETSPEND
- SAFEGRAPH_CENSUS_DATA2
- SALES
- SALES_AUSTRALIA
- SALES_COLOMBIA
- SALES_SWEDEN
- SECURE_VIEW_DEMO
- SNOWALERT
- SNOWCONVERT
- SNOWFLAKE
- SNOWFLAKE_SAMPLE_DATA
- STARBUCKS_PATTERNS_SAMPLE
- TIMF

```

1 // Get total case count by country
2 // Calculates the total number of cases by country, aggregated over time.
3 SELECT COUNTRY_REGION, SUM(CASES) AS Cases
4 FROM ECDC_GLOBAL
5 GROUP BY COUNTRY_REGION
6
7 // Change in mobility in over time
8 // Displays the change in visits to places like grocery stores and parks by
9 date, location and location type for a sub-region (Alexandria) of a state
10 (Virginia) of a country (United States).
11
12 SELECT DATE,
13 COUNTRY_REGION,
14 PROVINCE_STATE,
15 GROCERY_AND_PHARMACY_CHANGE_PERC,
16 PARKS_CHANGE_PERC,
17 RESIDENTIAL_CHANGE_PERC,
18 RETAIL_AND_RECREATION_CHANGE_PERC,
19 TRANSIT_STATIONS_CHANGE_PERC,
20 WORKPLACES_CHANGE_PERC
21
22 FROM GOOG_GLOBAL_MOBILITY_REPORT
23 WHERE COUNTRY_REGION = 'United States'
24 AND PROVINCE_STATE = 'Virginia'
25 AND SUB_REGION_2 = 'Alexandria';
26

```

COUNTRY_REGION	CASES
Afghanistan	49,273
Albania	48,530
Algeria	92,102
Andorra	7,338
Angola	16,188
Anguilla	10
Antigua and Barbuda	148
Argentina	1,498,160
Armenia	148,682
Aruba	5,049

Objects Query Results Chart

Query Details

Query duration 1.6s

Rows 214

COUNTRY_REGION Aa
100% filled

CASES 123
1 16,256,754

Next:

1. Click Data > Databases
2. Click the COVID19_BY_STARSCHEMA_DM database.
3. You can see details about the schemas, tables, and views that are available to query.

jason ACCOUNTADMIN

Worksheets

Dashboards

Data

Databases 1

Shared Data

Marketplace

Compute

Account

Organization

Classic Console

COVID19_BY_STARSCHEMA_DM 2

Database Database ACCOUNTADMIN 15 hours ago Snowflake Data Marketplace

Database Details Schemas 3

Source details

Provider Starschema

Title COVID-19 Epidemiological Data

Privileges

ACCOUNTADMIN (Current Role) OWNERSHIP

That's it! You have now successfully subscribed to the COVID-19 dataset from Starschema, which is updated daily with global COVID data. Notice we didn't have to create databases, tables, views, or an ETL process. We simply searched for and accessed shared data from the Snowflake Data Marketplace.

Resetting Your Snowflake Environment

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

First, ensure you are using the ACCOUNTADMIN role in the worksheet:

```
use role accountadmin;
```

Then, run the following SQL commands to drop all the objects we created in the lab:

```
drop share if exists zero_to_snowflake_shared_data;
-- If necessary, replace "zero_to_snowflake-shared_data" with the name you used for
the share

drop database if exists citibike;

drop database if exists weather;

drop warehouse if exists analytics_wh;

drop role if exists junior_dba;
```

Conclusion & Next Steps

Congratulations on completing this introductory lab exercise! You've mastered the Snowflake basics and are ready to apply these fundamentals to your own data. Be sure to reference this guide if you ever need a refresher.

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

What we've covered:

- How to create stages, databases, tables, views, and virtual warehouses.
- How to load structured and semi-structured data.
- How to perform analytical queries on data in Snowflake, including joins between tables.
- How to clone objects.
- How to undo user errors using Time Travel.
- How to create roles and users, and grant them privileges.
- How to securely and easily share data with other accounts.
- How to consume datasets in the Snowflake Data Marketplace.