snowflake
PRODUCT
PRICING
YOUR CLOUD DATA PLATFORM
Secure and easy access to any data with infinite scalability
snowflake

Snowflake Fundamentals 3-Day Workbook

21I16

snowflake EDUCATION SERVICES

workbook

Contents

1 Take a Quick Test Drive

1.1 Create Objects in the UI

In this task you will create some objects that will be used for labs throughout the course. You may not yet fully understand the concepts, but you will learn more about these objects as the course progresses. For now, you are just creating some objects that will be required in later exercises.

1.1.1 Log Into Your Snowflake Account

Log in to your Snowflake account using the information provided by your instructor. Remember, this is the web UI that was set up for you by the instructor. Use the username (an animal name assigned by the instructor) and the default password to access this account. A sample URL would look something like this:

https://xy12345.snowflakecomputing.com/console#/internal/worksheet

1.1.2 Locate the top ribbon, which has several icons across the top (**Databases**, **Shares**, etc.).

These icons are used to activate different areas of the UI. Identify each and attempt to learn their function. They are fairly user-friendly in what they do.

1.1.3 Now locate your user name and current role in the right most part of the top ribbon.

The down arrow located directly to the right of your user name and role can be used to view or modify your preferences, change your password, switch roles or log out.

1.1.4 Verify that you have TRAINING_ROLE selected.

If you have a different role selected, click the down arrow to the right of your user name, select Switch Role, and then select TRAINING_ROLE.

1.1.5 Navigate to **[Warehouses]** in the top ribbon.

1.1.6 Click **[Create]** above the list of warehouses**.** The **Create Warehouse** dialog box appears.

1.1.7 Fill in the fields with the information shown below to create a warehouse you will use to run queries:

- Name: **[login]**_WH
- Size: **Small**
- Set Auto-Suspend to **5 Minutes**
- Then click **Finish**.

In a production environment, you will likely be using several warehouses. For example, one for each group, or one for each type of function (queries, loads, etc.). For this course, you will use just this one warehouse and change its size as needed for various functions.

1.2 Create a Database:

1.2.1 Navigate to **[Databases]**.

1.2.2 Click **[Create]** to create a database.

1.2.3 Name the database **[login]**_DB.

1.3 Create a table:

1.3.1 In the left-side list, click the link for the database you just created.

1.3.2 Click **[Create]** and create a table named **[login]**_TBL.

1.3.3 Place it in the **Public** schema.

1.3.4 Click **[Add]** above the empty table to add a column.

1.3.5 Create four columns with the attributes shown below, then click **Finish**.

NOTE: The table below should be displayed. You must include the numbers in parentheses for the types that have them (NUMBER, STRING, and VARCHAR), or you will get an error:

```
NAME          TYPE            NOT NULL     DEFAULT
------------  --------------  -----------  -------------
ID            NUMBER(38,0)    Unchecked    Leave Blank
NAME          STRING (10)     Unchecked    Leave Blank
COUNTRY       VARCHAR (20)    Unchecked    Leave Blank
ORDER_DATE    DATE            Unchecked    Leave Blank
```

1.4 Create Objects Using SQL

1.4.1 Navigate to **[Worksheets]**.

1.4.2 Click in the tab labeled **New Worksheet**, and rename your worksheet *Test Drive*.

1.4.3 Set the context and drop the database you created in the first task:

```
USE WAREHOUSE [login]_WH;
USE DATABASE [login]_DB;
USE SCHEMA PUBLIC;
```

```
DROP DATABASE [login]_DB;
```

NOTE: This will drop any tables in the database, as well. You do not need a warehouse to create objects.

1.4.4 Drop your warehouse, and then recreate your database (**[login]**_DB) using SQL:

```
DROP WAREHOUSE [login]_WH;
CREATE DATABASE [login]_DB;
```

1.4.5 Re-create your table in the PUBLIC schema of your database:

```
CREATE OR REPLACE TABLE [login]_DB.PUBLIC.[login]_TBL
    (id NUMBER(38,0), name STRING(10),
    country VARCHAR(20), order_date DATE);
```

1.4.6 Re-create your warehouse, leaving it initially suspended so it is not using credits until you need it:

```
CREATE WAREHOUSE [login]_WH
    WAREHOUSE_SIZE=XSmall
    INITIALLY_SUSPENDED=True
    AUTO_SUSPEND=300;
```

1.4.7 Use the following commands to set defaults for your role, database, schema, and warehouse. This will be referred to as your “standard context” throughout the rest of this workbook. Once you set these defaults, any worksheet you open will automatically set your context to these values.

```
USE ROLE SECURITYADMIN;
ALTER USER [login]
    SET
    DEFAULT_ROLE=TRAINING_ROLE
    DEFAULT_NAMESPACE=[login]_DB.PUBLIC
    DEFAULT_WAREHOUSE=[login]_WH;
```

1.4.8 Log out of the web UI, and back in. This forces your new user settings to be used.

1.4.9 Open a new worksheet, and verify that your standard context is automatically set when you open a new worksheet. If the context isn't set, select the training_role and [login]_wh in the context section.

1.5 Run Queries on Sample Data

1.5.1 Click the three dots to the right of the context area and select the option **Turn on Code Highlight**.

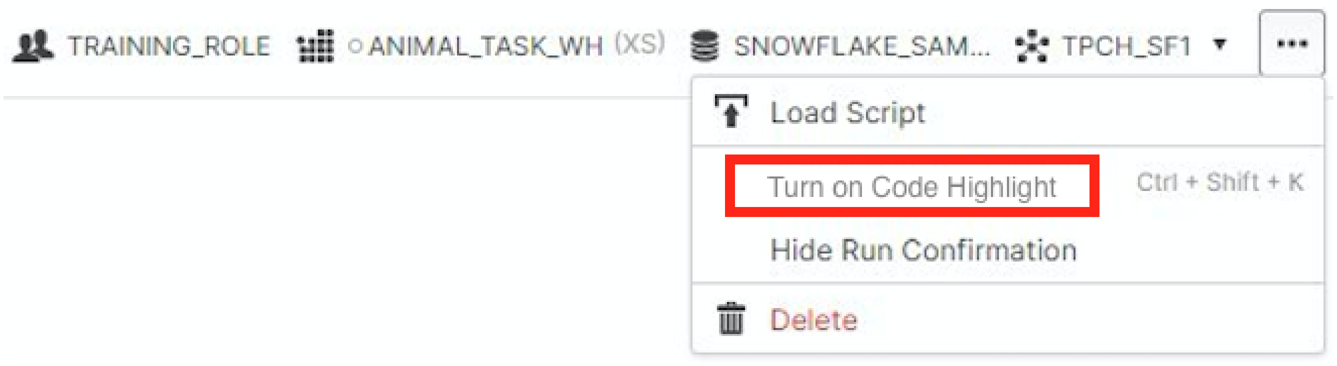


Figure 1: Enable Code Highlight

1.5.2 In the left-side navigation pane, navigate to **SNOWFLAKE_SAMPLE_DATA**, then **TPCH_SF1**.

1.5.3 Right-click the schema name (TPCH_SF1) and select **Set as Context**.

1.5.4 Verify that the new database and schema are now set in your context.

1.5.5 Click **TPCH_SF1** to expand the schema, and then click the **ORDERS** table. A pane describing the orders table appears at the bottom of the navigation pane.

1.5.6 Click **Preview Data** to preview the data in the **ORDERS** table.

Above the results is a slider with **Data** and **Details**. **Data** should be selected by default.

1.5.7 Select **Details** to view the detailed information on the column definitions.

1.5.8 In your worksheet, run the following commands to explore the data:

```
USE ROLE TRAINING_ROLE;

SHOW TABLES;

SELECT COUNT(*) FROM orders;
SELECT * FROM supplier LIMIT 10;

SELECT MAX(o_totalprice) FROM orders;

SELECT o_orderpriority, SUM(o_totalprice)
FROM orders
GROUP BY o_orderpriority
ORDER BY SUM(o_totalprice);

SELECT o_orderpriority, SUM(o_totalprice)
FROM orders
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

1.5.9 Using the syntax in the above commands as a guide, write a query that will return the ps_partkey, ps_suppkey, and ps_availqty columns from the PARTSUPP table. Order the output by part key.

```
SELECT ps_partkey, ps_suppkey, ps_availqty
FROM partsupp
ORDER BY ps_partkey;
```

1.5.10 Notice that there are multiple rows for each ps_partkey, with different values for ps_suppkey and ps_availqty. This means that several suppliers stock each part key, and have different quantities available.

1.5.11 Rewrite the query to return just the part key, and the total available quantity from all suppliers combined. GROUP and ORDER the output by the part key.

```
SELECT ps_partkey, SUM(ps_availqty)
FROM partsupp
GROUP BY ps_partkey
ORDER BY ps_partkey;
```

How many total rows were returned by your query? **200,000**

How many total rows are in the PARTSUPP table? **800,000**

1.5.12 Write a query that will return the lowest- and highest-priced items (based on the extended price) from the LINEITEM table.

What are the lowest and hightest prices returned? (Answer: **104949.50** and **901.00**)

```
SELECT MIN(l_extendedprice), MAX(l_extendedprice)
FROM lineitem;
```

1.5.13 Suspend the warehouse

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

2 Work with Storage and Compute

2.1 Review the TRAINING_DB Database

2.1.1 Navigate to **[Databases]**, then locate and select **TRAINING_DB**.

NOTE: If you do not see a list of databases, you are probably still drilled down into a table from the previous lab. Look for the breadcrumb trail under the Snowflake logo and click **Databases** to return to the top level of this area. You will then see the list, including **TRAINING_DB**.

2.1.2 Select **Schemas** from the tabs above the list of tables. Review the list of schemas that are defined for this database.

2.1.3 Return to the **Tables** tab.

2.1.4 Review the table list.

2.1.5 Click the **LINEITEM** table located in the **TPCH_SF1** schema.

2.1.6 Review the information about the table's structure.

2.1.7 Using the breadcrumb trail above the table list, click **TRAINING_DB** to go back up one level.

2.1.8 Click the **LINEITEM** table that is located in the **TPCH_SF10** schema.

2.1.9 Review the information about the table's structure.

2.1.10 Compare the columns in this **LINEITEM** table in **TPCH_SF1** with the columns in the **LINEITEM** table in **TPCH_SF10**.

You will find that the columns are identical.

Question: What is the difference between these tables?

Answer: The number of rows in each table. Tables in the TPCH_SF10 schema have 10 times as many rows as tables in the TPCH_SF1 schema.

2.1.11 Use the breadcrumb trail to go back to **TRAINING_DB**.

2.1.12 Sort the results by the **Table Name** column and find all the tables named **LINEITEM**.

2.1.13 Compare their sizes to see the difference between the tables in the different schemas.

2.1.14 Navigate through the **Views**, **Schemas**, and **File Formats** tabs.

2.1.15 Notice that all views and tables reside within a schema.

Snowflake Accounts are composed of one or more Databases which are composed of one or more Schemas. In turn, Schemas comprise one or more data objects like Tables, Views, Stages, File Formats, and Sequences.

2.2 Create and Organize Objects

2.2.1 Use the breadcrumb trail to return to the **[Databases]** main page.

2.2.2 Locate and click **[login]**_DB in the database list.

2.2.3 In the **Tables** tab, click **[Create]**.

2.2.4 Take note of how the Create Table wizard requires that a Schema be selected.

NOTE: The **LINEITEM** tables you looked at in the previous task are all different tables, in different schemas - they just all happen to have the same name. A table name must be unique **within a schema** but can be duplicated in other schemas.

2.2.5 Click **[Cancel]** to exit the table creation wizard.

2.2.6 Return to **[Databases]**. Toggle between the **Views**, **Stages**, **File Formats**, and **Sequences** tabs and click **[Create]** for each one to see how to create objects of these types.

2.2.7 Cancel without creating anything for each of these object types.

Snowflake enforces a logical hierarchy:

- An account can contain many databases / a database belongs to one account
- A database can contain many schemas / a schema belongs to one database
- A schema can contain many tables / a table belongs to one schema

2.2.8 Open a new worksheet, and name it *Create Objects*.

2.2.9 If you haven't created the class database or warehouse, do it now

```
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
```

2.2.10 Verify that it automatically sets the context to your standard context.

2.2.11 Create a schema in your database, and name it **[login]**_SCHEMA.

2.2.12 Specify the full database.schema path when you create it, then verify that it is set as the default schema in your worksheet.

HINT: Review the syntax for creating a database; the syntax for a schema is similar.

```
CREATE SCHEMA [login]_DB.[login]_SCHEMA;
```

2.2.13 Create a MEMBERS table.

2.2.14 Give it columns to hold a numeric customer ID, a first name (up to 20 characters), a last name (up to 30 characters), the date they joined, and their award program level (bronze, silver, or gold):

```
CREATE TABLE members (id INT, first_name VARCHAR(20),
last_name VARCHAR(30), member_since DATE, level VARCHAR(6));
```

2.2.15 Query the table to make sure the columns are all there.

```
SELECT * FROM members;
```

2.3 Review Storage Usage

2.3.1 Select **[Account]**. This section displays information on storage, data transfer, and how many credits have been billed.

2.3.2 Click the **Average Storage Used** box in the display area. This shows average total storage over time.

NOTE: Since class just started, there will be very little data in this area, and in the other areas inspected in this task - this is just to show you what information can be found here.

Just above the display area on the right-hand side is a month indicator.

2.3.3 Use the pull-down menu to change months and review the change in total storage over time.

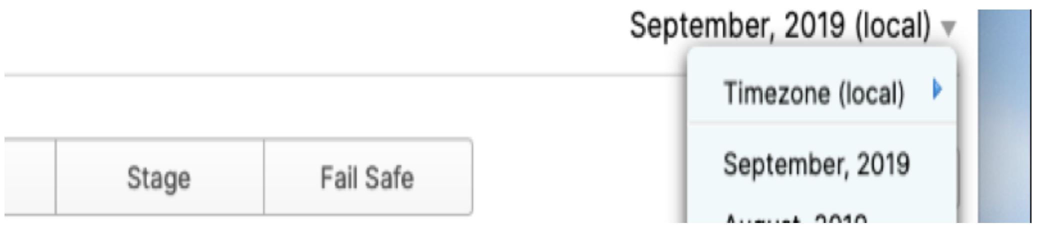


Figure 2: Month Pull Down Menu

NOTE: The scale of the y-axis in the graph will change from month to month based on the highest storage amount for that month. To compare two months, make sure you are paying attention to the y-axis. Two months that are visually similar may be very different as far as actual storage used.

2.3.4 Toggle between the **Total**, **Database**, **Stage**, and **Fail Safe** categories and review the change in storage for each throughout the month. You will learn more about Stages and Fail Safe storage later in the course.

2.3.5 Select **[Worksheets]** and return to the worksheet you have been using.

INFORMATION_SCHEMA is a schema that is automatically created for all databases. It contains database-specific information. You will learn more about this later. Use this to run a query to return average daily storage usage for the past 10 days, per database, for all databases in your account:

```
SELECT * FROM TABLE (INFORMATION_SCHEMA.DATABASE_STORAGE_USAGE_HISTORY
    (DATEADD('days', -10, CURRENT_DATE()), CURRENT_DATE()));
```

2.4 Run Commands with No Virtual Warehouse

2.4.1 Suspend your warehouse:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

2.4.2 Change the worksheet context to the following:

- Database: SNOWFLAKE_SAMPLE_DATA
- Schema: TPCH_SF1

2.4.3 You can do this either with the context menu, or with the SQL command below:

```
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCH_SF1;
```

Throughout the labs when asked to set your context, you can choose between the SQL method and the context menu method.

2.4.4 Run the following command to disable the query result cache. You will learn more about this cache in the next module.

```
ALTER SESSION SET USE_CACHED_RESULT=FALSE;
```

2.4.5 Execute the following query which selects MIN and MAX values for the **L_ORDERKEY** column and a row count for the **LINEITEM** table:

```
SELECT MIN(l_orderkey), MAX(l_orderkey), COUNT(*)
FROM lineitem;
```

2.4.6 Notice that the results return almost immediately.

2.4.7 Click **Query ID** in the result frame, then click the Query ID.

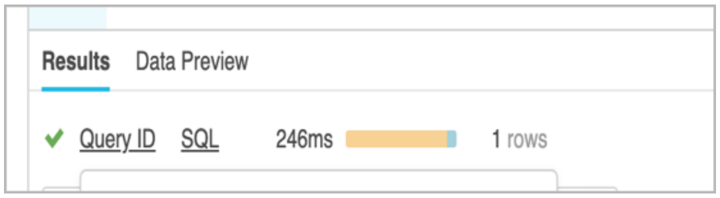


Figure 3: Results Query Id

2.4.8 This will bring up the query detail pages under Query History.

2.4.9 From within the history, select the **[Profile]** tab above the summary area. Note that the query profile states **METADATA-BASED RESULT**. This means the results were pulled from the Snowflake metadata store, and the query completed with no virtual warehouse.

2.5 Work with Virtual Warehouses

2.5.1 Return to your worksheet.

2.5.2 Using SQL, create a warehouse named **[login]**_WH2, with the following characteristics.

2.5.3 Take note of the various parameters you can set for warehouses:

- Name: **[login]**_WH2
- Size: XSmall

- Initially Suspended: True
- Auto Resume: True
- Auto Suspend: 300
- Minimum Clusters: 1
- Maximum Clusters: 1

2.5.4 Next, run the following query:

```
CREATE WAREHOUSE [login]_WH2
    WAREHOUSE_SIZE = XSMALL
    AUTO_RESUME = TRUE
    AUTO_SUSPEND = 300
    MIN_CLUSTER_COUNT = 1
    MAX_CLUSTER_COUNT = 1
    INITIALLY_SUSPENDED = TRUE
    COMMENT = 'Another warehouse for completing labs';
```

NOTE: This query sets INITIALLY_SUSPENDED to TRUE. When you created a warehouse with the UI the warehouse automatically started. With this option the warehouse will not start until it is used for a query. Note also that when you create a warehouse using SQL it is automatically set in your context.

2.5.5 Execute the command below to list the configured warehouses.

2.5.6 You will see several more warehouses than you have created. Why do you suppose that is the case?

```
SHOW WAREHOUSES;
```

Answer: All students are working in the TRAINING_ROLE role, and all users that are in the same role have access to all the same objects. Therefore, you will see all the warehouses that your fellow students create, as well as your own.

2.5.7 Alter **[login]**_WH2 to change the following parameters:

- Size: Small
- Auto Suspend: 600
- Maximum Clusters: 3

NOTE: the **AUTO_SUSPEND** factor is specified in seconds (600/60 = 10 minutes).

2.5.8 Next, execute the following command:

```
ALTER WAREHOUSE [login]_WH2
    SET
    WAREHOUSE_SIZE = Small
    AUTO_SUSPEND = 600
    MAX_CLUSTER_COUNT = 3;
```

2.5.9 Navigate to **[Warehouses]**.

2.5.10 Locate your Warehouse and confirm the parameters are as follows:

- Status: Suspended
- Name: **[login]**_WH2
- Size: Small
- Cluster: min: 1, max: 3
- Auto Suspend: 10 minutes
- Auto Resume: Yes
- Owner: TRAINING_ROLE

2.5.11 Return to your worksheet and run the following query:

```
SHOW WAREHOUSES LIKE '[login]%';
```

2.5.12 Run the following command to disable the query result cache. You will learn more about this cache in the next module.

```
ALTER SESSION SET USE_CACHED_RESULT=FALSE;
```

2.5.13 Execute the following query:

```
SELECT *
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.NATION
LIMIT 10;
```

2.5.14 Return to **[Warehouses]**.

2.5.15 Locate **[login]**_WH2 and verify the Status is now “Started”.

Because your warehouse was set to auto resume, it started automatically when it was needed to complete a query.

2.5.16 Return to **[Worksheets]**.

2.5.17 Suspend **[login]**_WH2:

```
ALTER WAREHOUSE [login]_WH2 SUSPEND;
```

2.5.18 Show your warehouses and confirm that the warehouse is suspended:

```
SHOW WAREHOUSES LIKE '[login]%';
```

2.5.19 Alter **[login]**_WH2 and set the WAREHOUSE_SIZE to XSmall:

```
ALTER WAREHOUSE [login]_WH2 SET WAREHOUSE_SIZE=XSmall;
```

2.5.20 Suspend and then drop **[login]**_WH2:

```
ALTER WAREHOUSE [login]_WH2 SUSPEND;
```

Note that you should have gotten the following error because the warehouse has already been suspended: "Invalid state. Warehouse '**[login]**_WH2' cannot be suspended."

Proceed with the statement below to drop the warehouse:

```
DROP WAREHOUSE [login]_WH2;
```

2.5.21 Show warehouses to verify that **[login]**_WH is still listed, but **[login]**_WH2 is not.

```
SHOW WAREHOUSES LIKE '[login]%';
```

2.5.22 Suspend and resize the warehouse

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

3 Explore Snowsight

This lab should take approximately 45 minutes.

In this lab you will practice logging in to Snowsight, navigating the Snowsight interface, and using its features to gain insights into data.

3.1 Connecting to Snowsight

In this step you're going to practice connecting to Snowsight and importing worksheets from the classic console.

3.1.1 Open the WebUI and click the Preview App button:

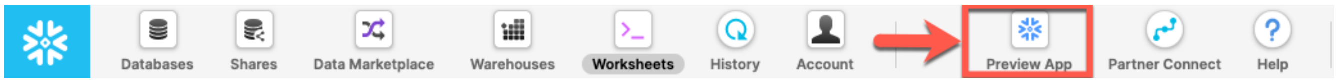


Figure 4: Preview App Button

3.1.2 When the new tab opens, click the Sign in to continue button

3.1.3 Type in the username and password used in class.

3.1.4 You should now see the Worksheets pane. Click the [...] button in the upper right hand corner.

3.1.5 A drop down menu should appear. Choose the Import Worksheets option.

3.1.6 You will see an Import Worksheets dialog box. Click the Import button in the dialog box.

3.2 Working with worksheets.

Now let's create a new worksheet.

3.2.1 Click the Worksheets link to go back to the worksheets page.

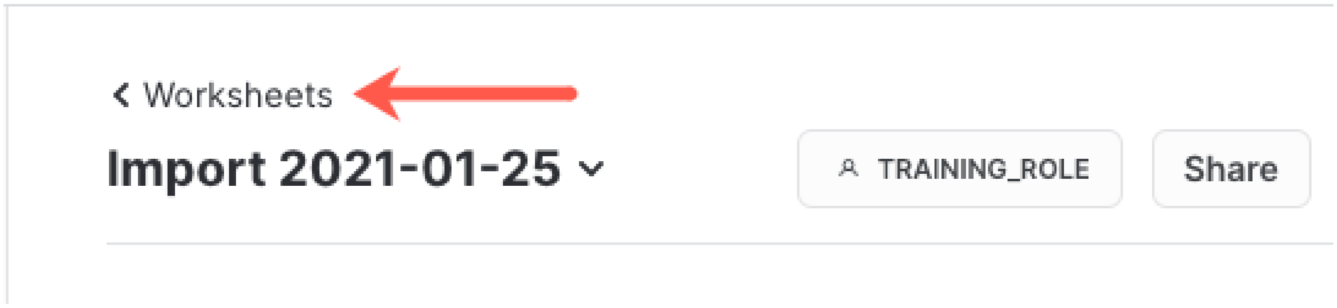


Figure 5: The New Worksheet Button

3.2.2 Click the + Worksheet in the top right corner.

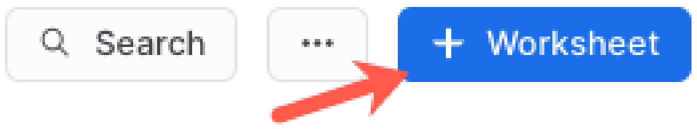


Figure 6: The New Worksheet Button

You will be taken to a window with a new worksheet. Next to the home button in the upper left, there is a drop down with the title of the new worksheet in a date-time format. Worksheets are given a title in date-time format by default when created. Now let's change the worksheet name.

3.2.3 Click on the worksheet name drop down arrow and click on the worksheet name.

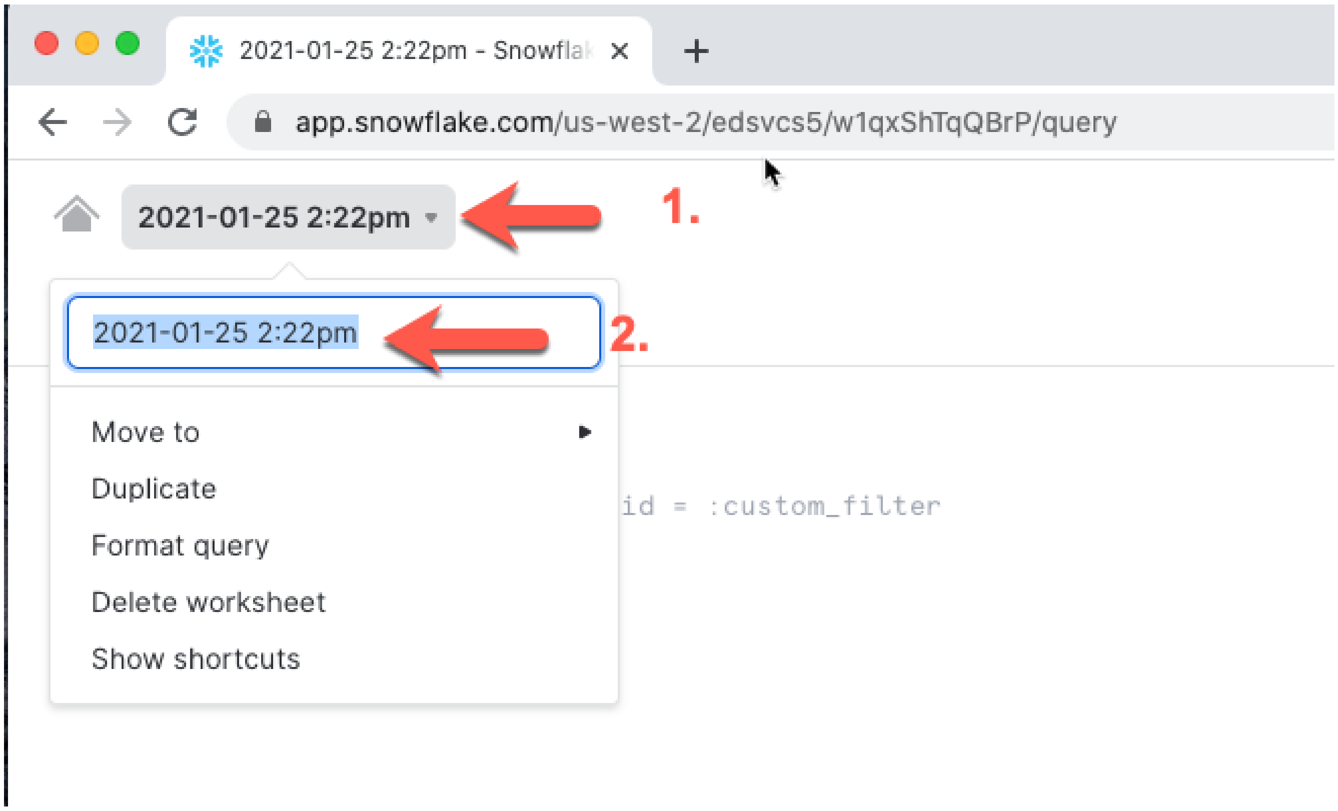


Figure 7: Renaming the Worksheet

3.2.4 Rename the worksheet to "Member List" and hit Enter.

3.3 Setting the context

Now let's select the context. The context consists of a role, warehouse, database and schema. These four objects must be explicitly specified in order for a query to execute.

Your role is already selected, so let's select a warehouse, database and schema so we can run a query.

3.3.1 Select your animal name warehouse in the upper right-hand corner:

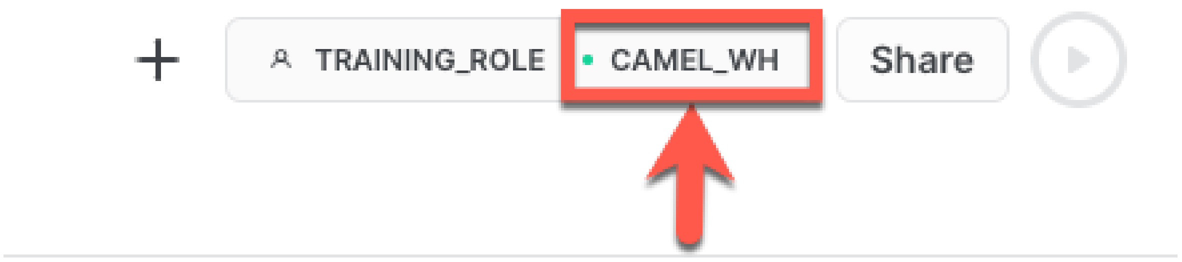
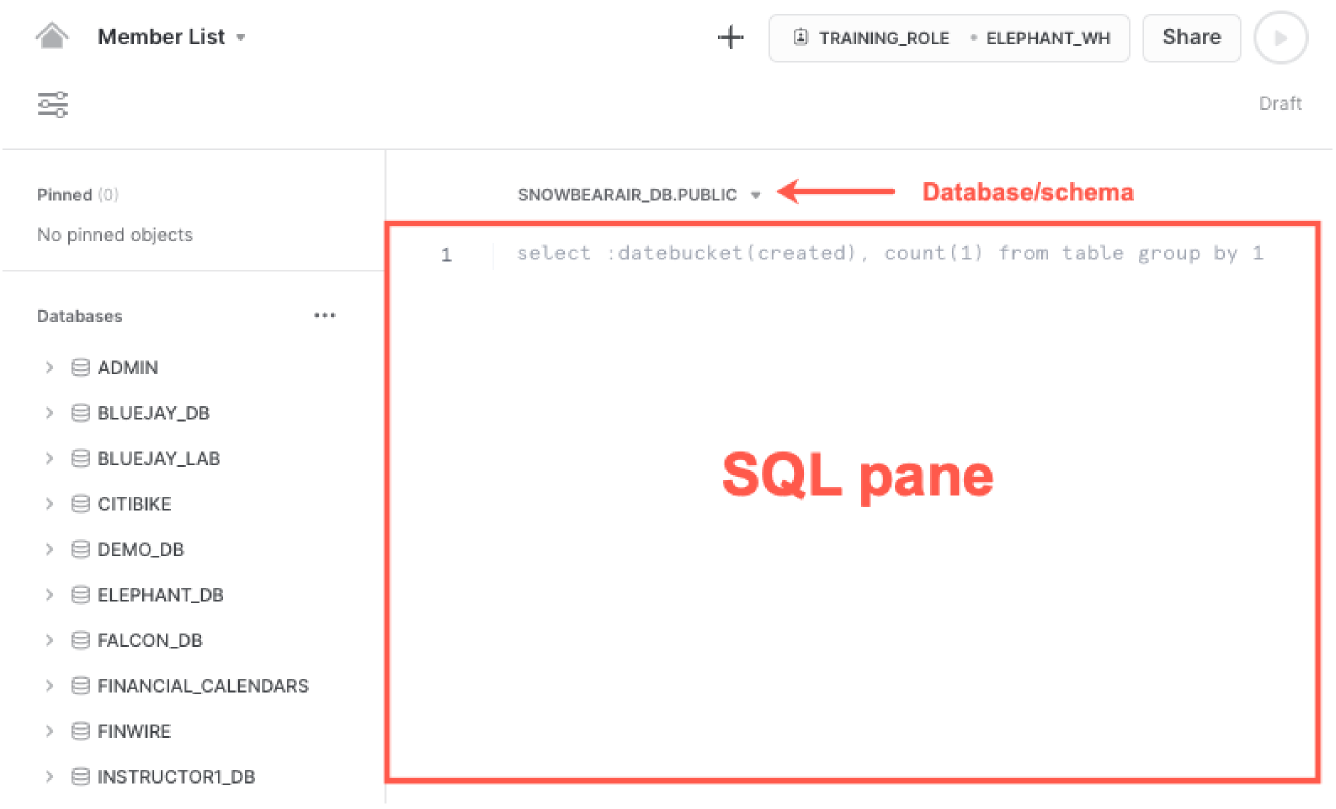


Figure 8: Selecting a Warehouse

3.3.2 Locate the SQL pane

Note the SQL pane in the image below. Just above the SQL pane is the selector for the database and schema.



```
select :datebucket(created), count(1) from table group by 1
```

Figure 9: The SQL Pane

3.3.3 In the upper left-hand corner just above the SQL pane, change the database and schema by selecting the SnowBearAir_DB database and the MODELED schema as shown below:

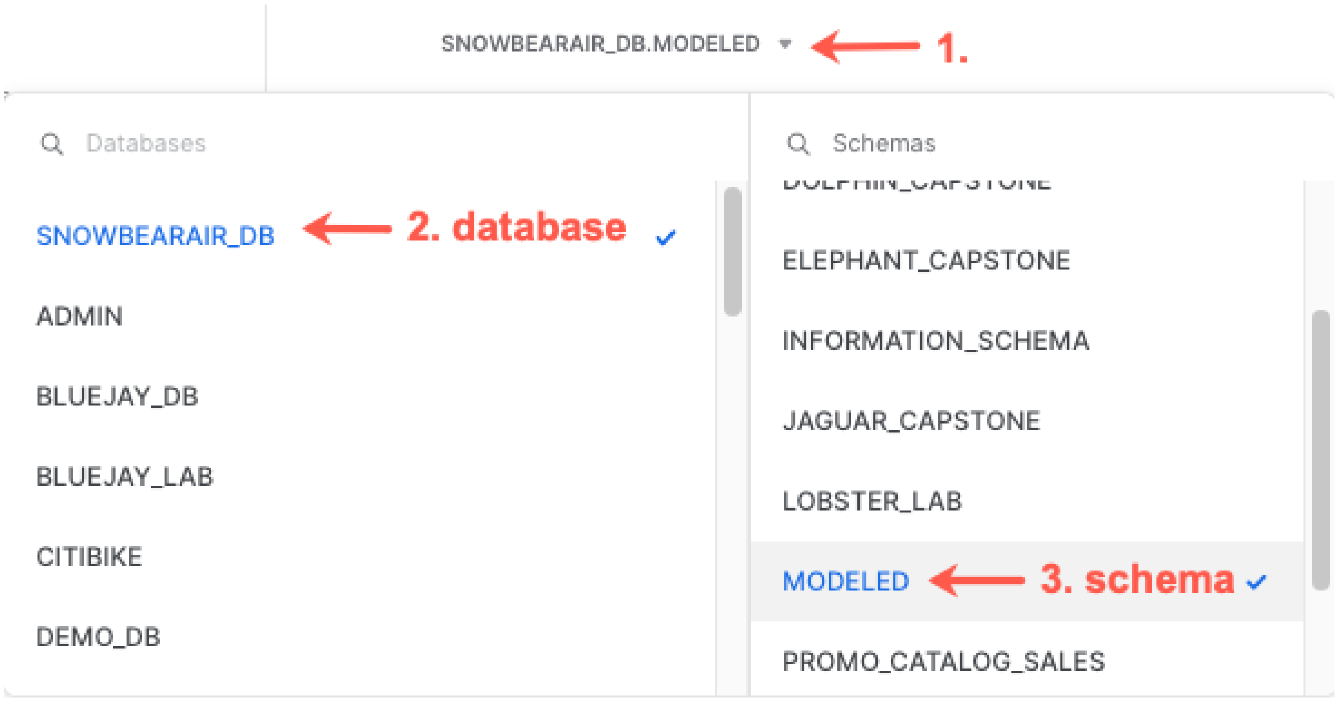


Figure 10: Selecting a Database and Schema

3.3.4 Preview the data

Now let's preview the data before we run a query. Navigate to the table in the left-hand navigation bar as shown below by following steps 1, 2 and 3.

Notice that in the fourth step window has appeared that shows the fields in the table as well as an Aa, 123, or clock. Aa represents a text field, 123 a numeric field and the clock a date/time field.

Now click the preview button as shown in step 4.

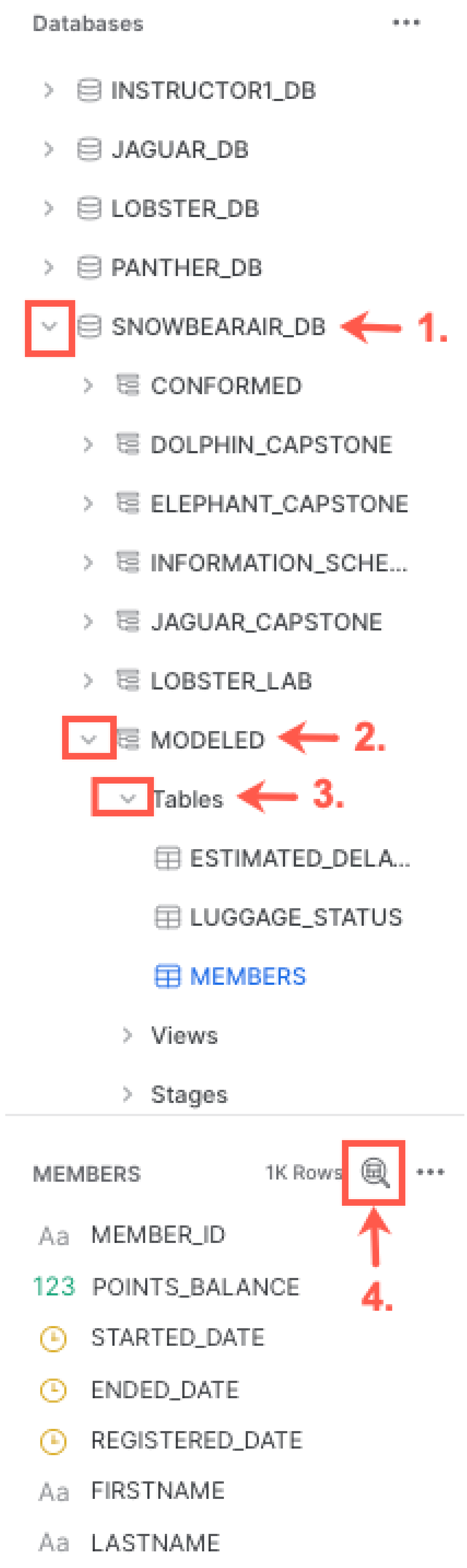


Figure 11: Selecting a Database and Schema

You should now see a pop up window with a preview of the data in the table.

3.3.5 Scroll using the horizontal and vertical scroll bars in order to examine all of the available fields.

3.3.6 Click outside the preview window to close the preview window.

3.4 Write and run a query

3.4.1 Click back to the query screen.

3.4.2 SLOWLY TYPE the following query into the query screen. Do not cut and paste. As hints pop up, press the Tab key to auto-complete:

```
SELECT * FROM MODELED.MEMBERS M;
```

3.4.3 Run the query by pressing the run button in the upper right-hand portion of the screen:

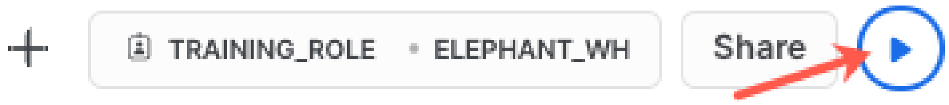
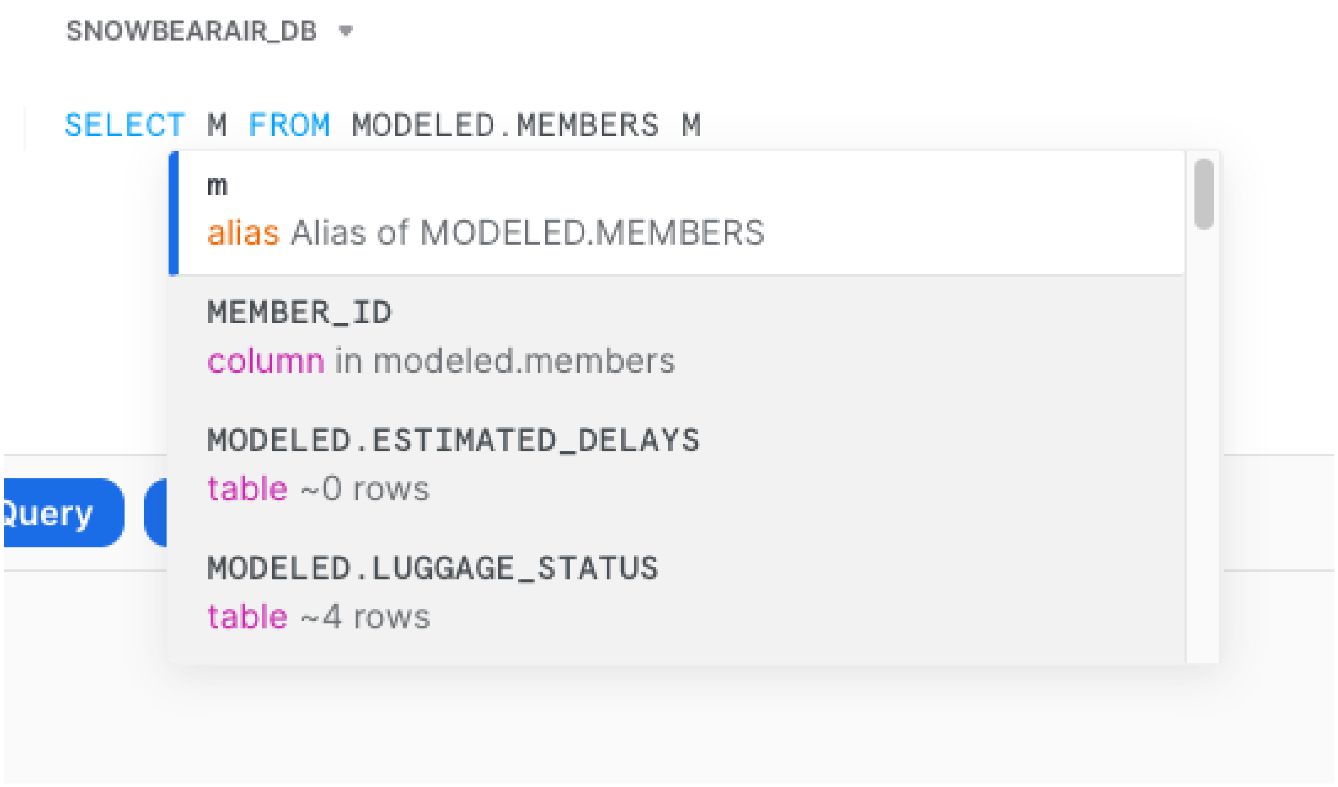


Figure 12: Running a query

You can see the data in the results pane below. Now let's change the columns selected using the auto-complete feature

3.4.4 Put the cursor DIRECTLY after the * and type an 'M' followed by a '.':

You should see a pop-up dialog box with a list of columns from the member table as shown below:



```
SELECT M FROM MODELED.MEMBERS M
```

m
alias Alias of MODELED.MEMBERS

MEMBER_ID
column in modeled.members

MODELED.ESTIMATED_DELAYS
table ~0 rows

MODELED.LUGGAGE_STATUS
table ~4 rows

Figure 13: Column List

3.4.5 Select the member_id column from the pop-up dialog box.

3.4.6 Now let's add the first name and last name columns. Type a comma after the member_id column and then 'M.f' to auto-generate a suggestion.

You should see the first name column in the list. Press the tabkey to add it to the query. Repeat these steps for the last name column. The query should look like this:

```
SELECT M.member_id, M.firstname, M.lastname FROM modeled.members M;
```

3.4.7 Now let's make the query easier to read. Click on the drop down arrow next to the worksheet name and select Format query.

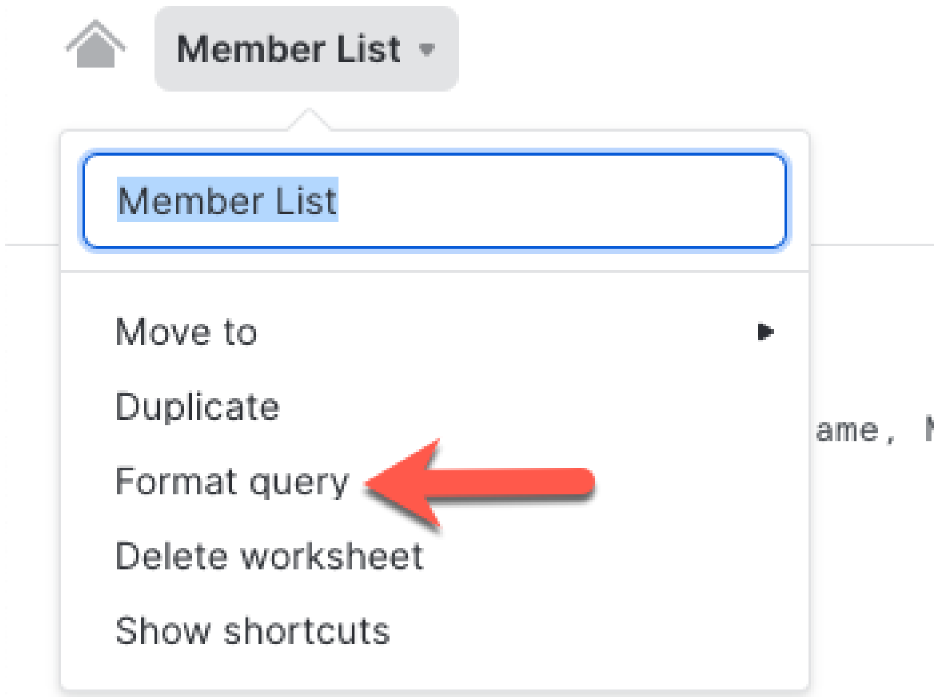
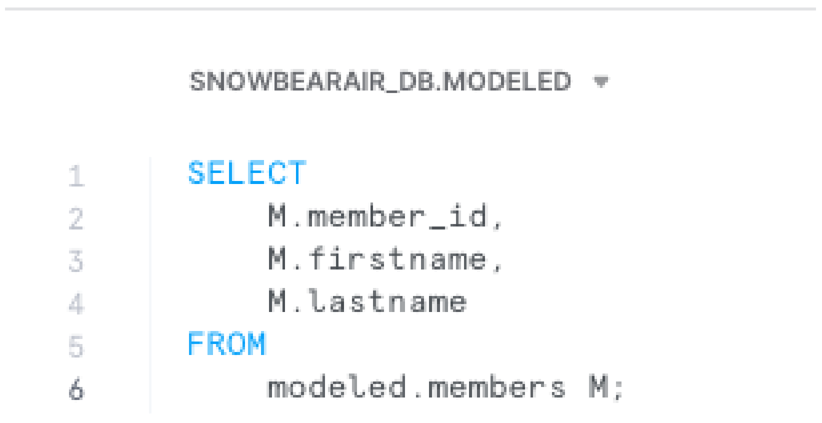


Figure 14: Selecting Format Query

The query should now appear as it does below.



```
SELECT
    M.member_id,
    M.firstname,
    M.lastname
FROM
    modeled.members M;
```

Figure 15: Formatted Query

3.4.8 Now run the query.

You should now see the results of the query in the bottom pane.

Just above the query results pane, there are four buttons that can be used to control what is visible in the query screen: Objects, Query, Results and Chart. When selected, each shows or hides the different parts of the query screen.

3.4.9 Click each button and observe the results.

Note that Objects hides the left-hand navigation bar, Query hides the SQL pane, Results hides the results pane, and Chart displays a default chart that you can modify to your liking.

3.5 Working with worksheet history

Snowsight offers the ability to preview and re-run previously edited versions of worksheets. Let's learn how to work with that feature now.

3.5.1 Modify the query to include the city and state columns and run the query again.

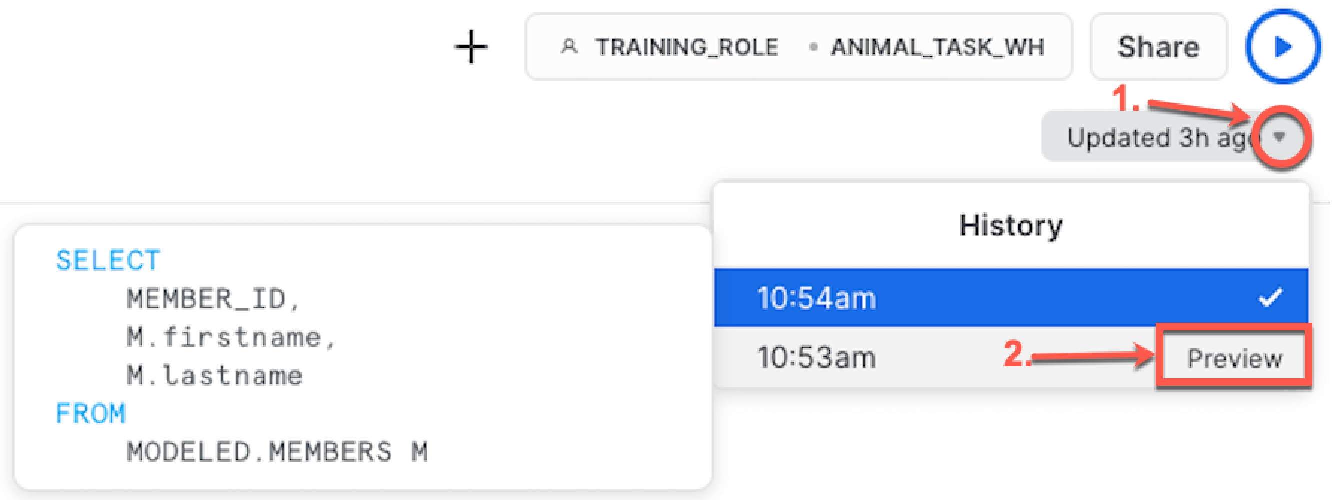
3.5.2 Modify the query again to include the points_balance column and run the query again.

3.5.3 Now that we have a worksheet history to examime, click on the drop down arrow below the blue run arrow.

Note that there are several time slots now.

3.5.4 For any of the time slot entries, float your arrow to the right of the entry to see the “preview” link.

3.5.5 Next, float your arrow over the link to see a preview of the SQL language for the query represented by that time slot.



```
SELECT
    MEMBER_ID,
    M.firstname,
    M.lastname
FROM
    MODELED.MEMBERS M
```

Figure 16: Previewing a Query Previously Run

Select a previous entry to see the query results again. FInally, click back to the current query.

3.6 Change the sort order of a column.

Each column offers an ellipsis next to the column name for sorting. Let’s try sorting now.

3.6.1 Click in the top of the State column to select the column.

3.6.2 Click the ellipsis (…).

A box should appear with down and up arrows. Click these to change the sort order of the column.

Now click in the top of the points_balance. Change the order of the column and try changing the “Show Thousands” separators.

3.7 Create a second Worksheet with another query

Now let’s learn how to create new worksheets.

3.7.1 Click on the Home button in the upper left hand corner.

You should be back at the home screen and a list of worksheets should be displayed.

3.7.2 From the home screen, click the blue “+ Worksheet” button in the upper right hand corner.

A new empty worksheet should be displayed.

3.7.3 Rename the worksheet to “Points for Members” and add in the following query:

```
SELECT m.age,
count(*) AS member_cnt,
sum(points_balance) as points_sum,
(points_sum / member_cnt)::integer as member_avg
FROM MODELED.MEMBERS M
     GROUP BY 1;
```

3.7.4 Click the drop down arrow next to the worksheet name and select Format query.

3.7.5 Click the run button. (Arrow in the top right corner of the screen)

3.8 Using the Automatic Contextual Statistics section

Now we’ll practice applying statistical filters to the query we just ran. These filters can be used to quickly gain insights into the result set.

In a nutshell, the way filters work is that you select a column and will be presented with an interactive graph to filter the data. Let’s walk through this step by step.

3.8.1 Locate the filters pane to the right of the result pane and click the member_avg column graph.

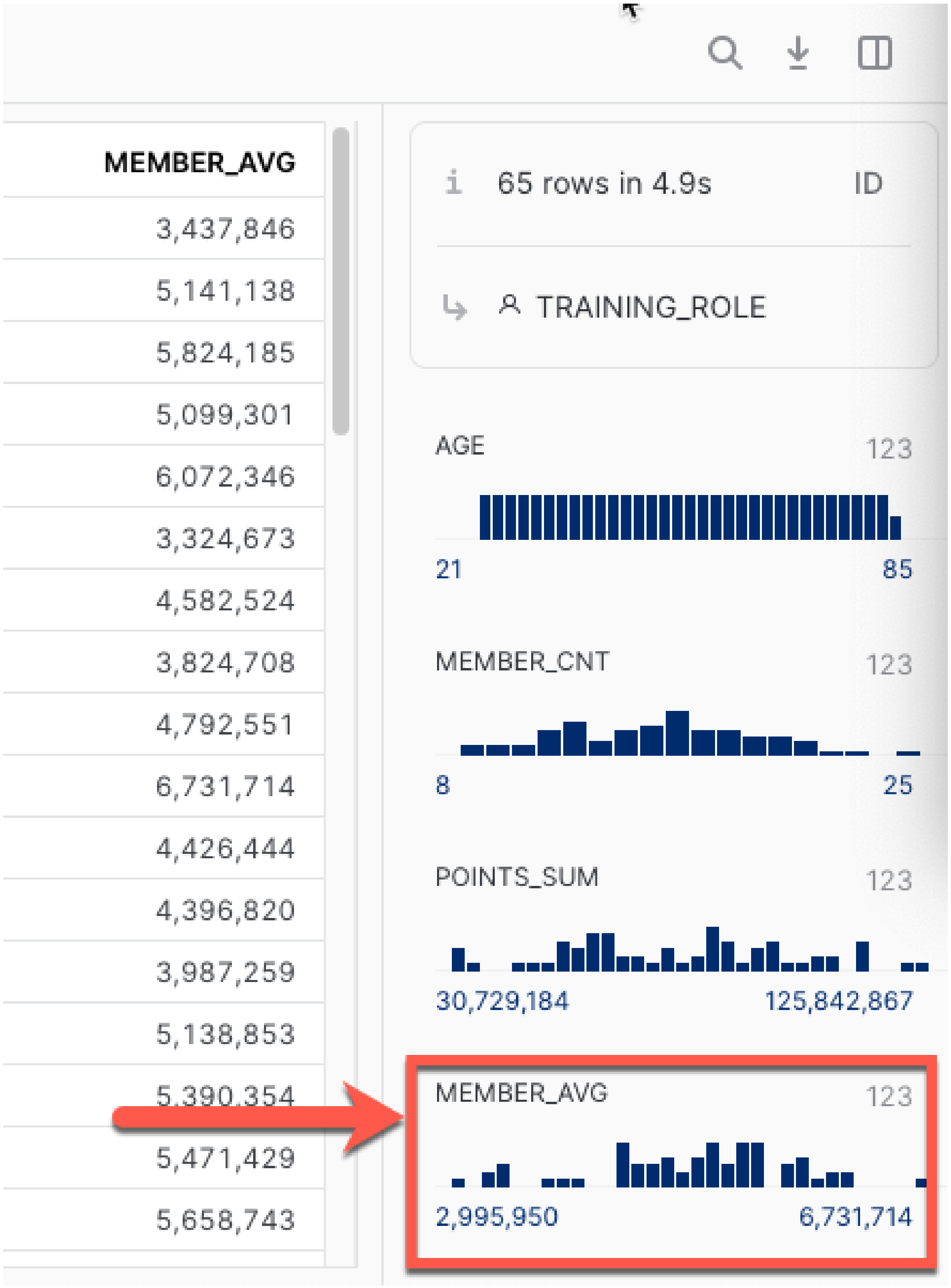
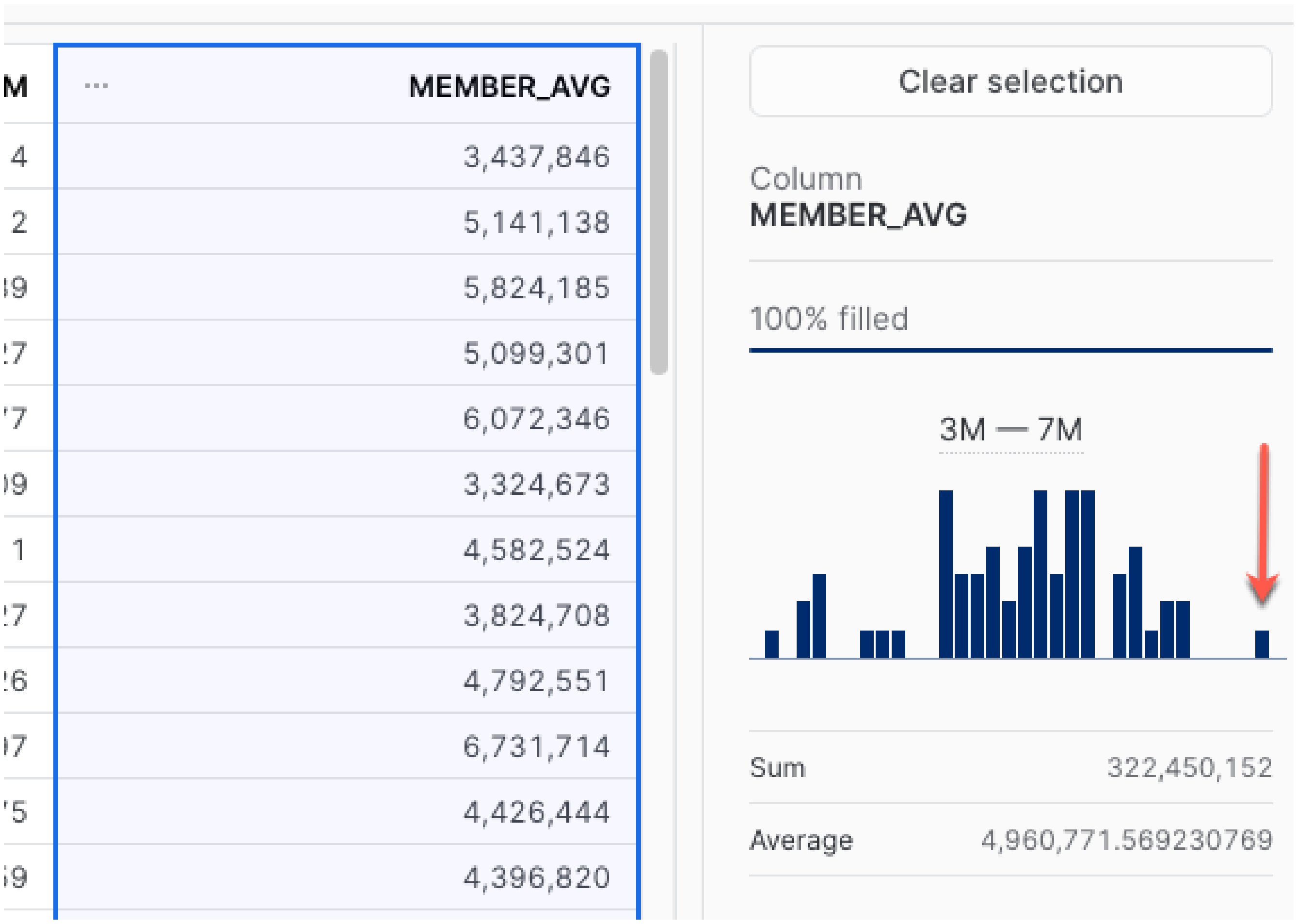


Figure 17: Member Average filter

All available columns except for the member average column should have been hidden in the filter pane. You should now see an interactive bar graph labeled “Column MEMBER_AVG” in the filter pane. Click the last bar in the graph to further filter the data:

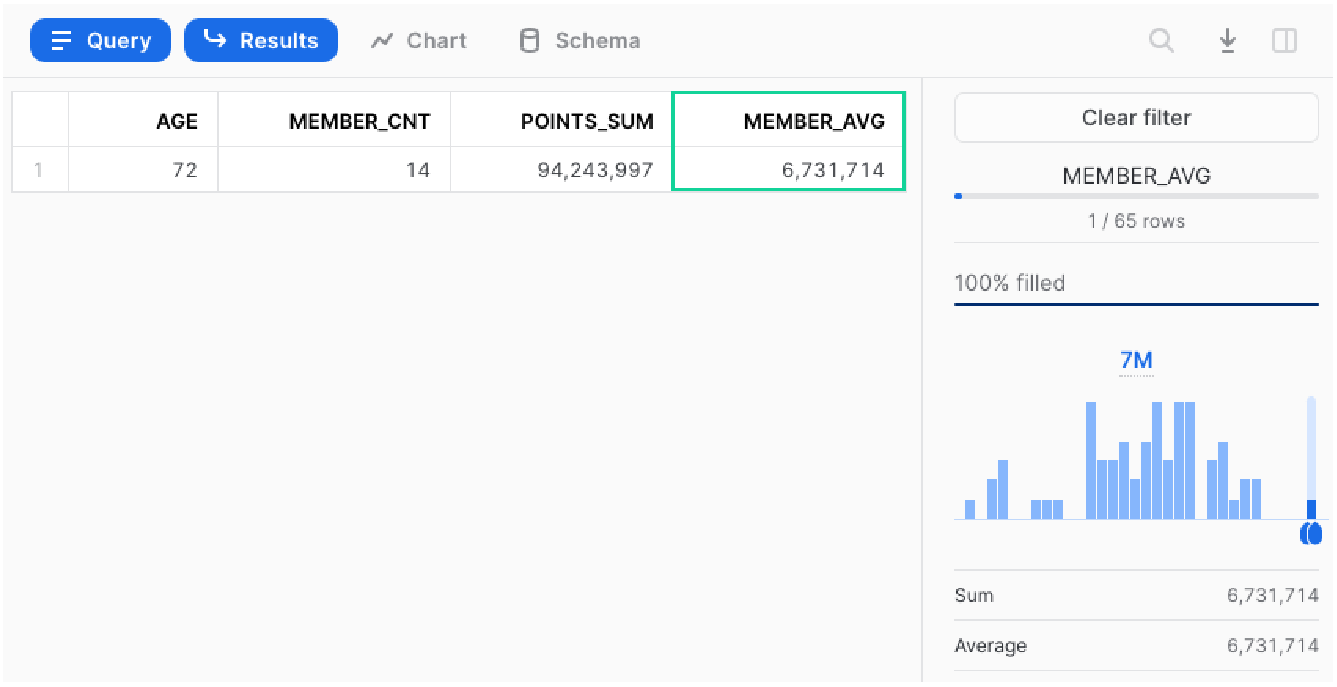


M	MEMBER_AVG
4	3,437,846
2	5,141,138
9	5,824,185
7	5,099,301
7	6,072,346
9	3,324,673
1	4,582,524
7	3,824,708
6	4,792,551
7	6,731,714
5	4,426,444
9	4,396,820

Sum	322,450,152
Average	4,960,771.569230769

Figure 18: Filtering by the last bar in the Member Average graph

The data should now be filtered as shown below:



	AGE	MEMBER_CNT	POINTS_SUM	MEMBER_AVG
1	72	14	94,243,997	6,731,714

Sum	6,731,714
Average	6,731,714

Figure 19: Member Average when filtered

3.8.2 Now click the Clear filter button in the filter pane.

All four columns should be shown now, and the Member Avg column should be highlighted.

3.8.3 Click on each column header and note how the column you clicked is highlighted and a new column filter graph is shown.

3.8.4 Practice filtering on each column as you did above.

For example, select the MEMBER_CNT filter. How many ages are in the highest member count? HINT: Click the longest bar in the filter.

3.9 Creating a folder and adding worksheets into the folder

Now we'll practice organizing worksheets into folders. The ability to organize worksheets in folders was designed to make it easier for you to group worksheets into logical topics and thus be able to locate what you need quickly.

The way this works is that you can create folders within the root of the Worksheets pane and add worksheets to those folders. Let's practice that now.

3.9.1 Click the home button.

3.9.2 Click the ellipsis (…) next to the +Worksheet button and select New Folder from the menu.

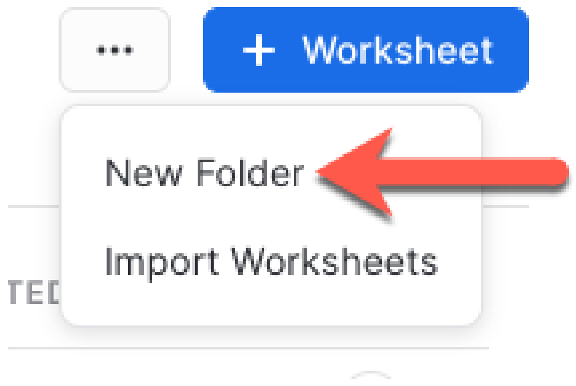


Figure 20: New Folder button

3.9.3 Name the folder Members Queries and create the folder.

Note the "Members Queries" folder name at upper left hand portion of the worksheets pane. You are now in that folder. Now we're going to add a worksheet to this folder.

3.9.4 Click "Worksheets" above "Member Queries" to return to the root of the Worksheets pane.

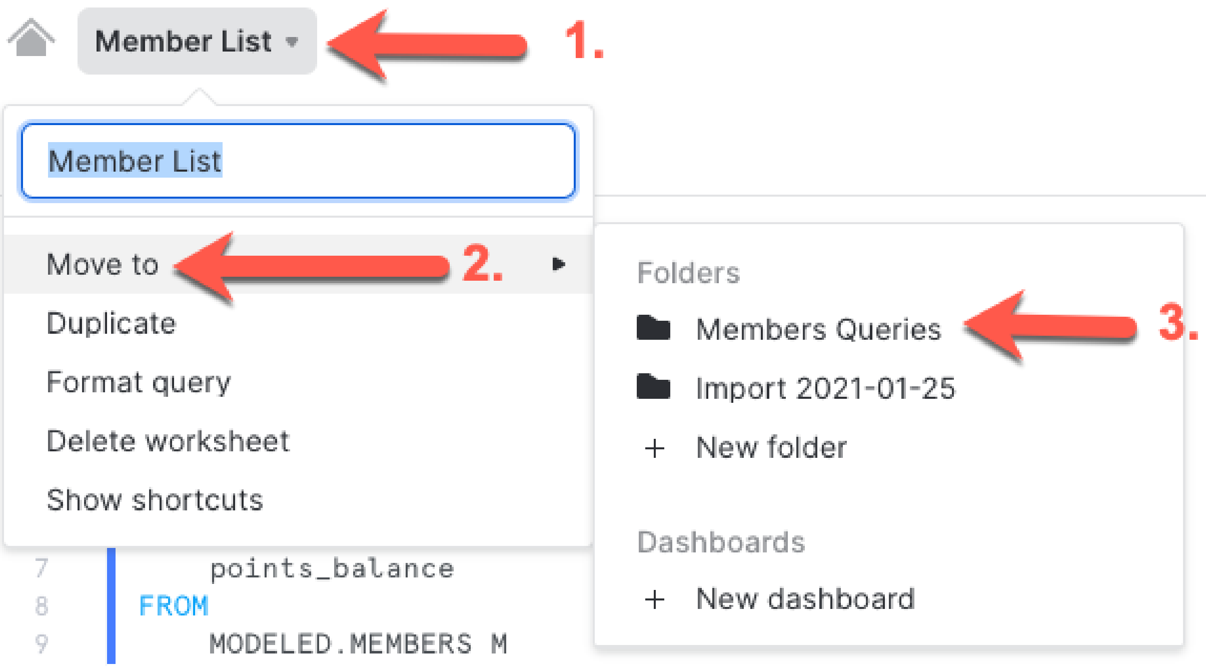
You are now at the root of the Worksheets pane. Note that there are four "tabs" beneath the Worksheets title: Recent, Shared with me, My Worksheets and Folders. By using My Worksheets and Folders, you can navigate to your worksheets, select one and add it to a folder, then navigate to the folder to see the worksheet.

3.9.5 Click the "My Worksheets" tab.

3.9.6 Select the "Members List" worksheet.

3.9.7 You should now see the contents of the Members List worksheet, to include the SQL code and perhaps the results.

3.9.8 Click the drop down arrow next to the worksheet name and select Move to and the Members Queries folder.

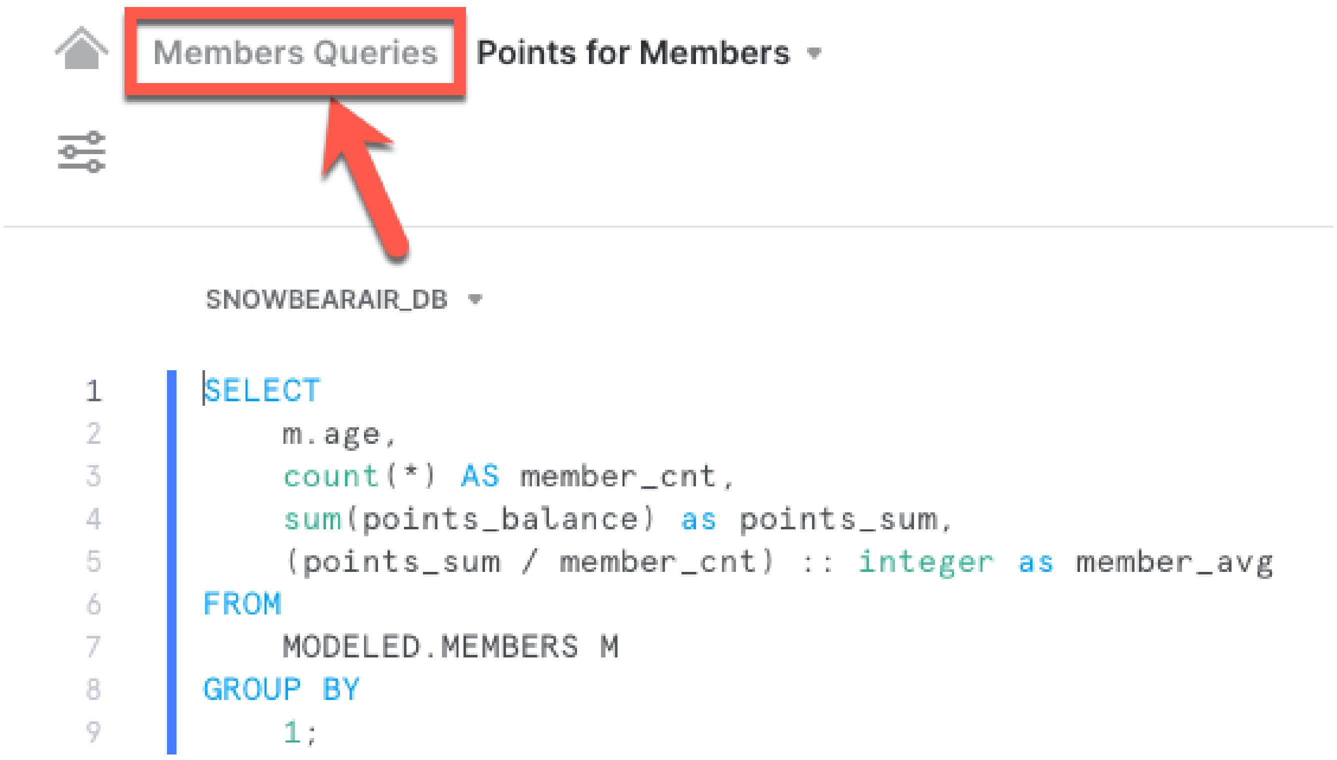


```
    points_balance
FROM
    MODELED.MEMBERS M
```

Figure 21: Move To Members Queries

3.9.9 Repeat the steps to move the Points for Members query into the Members Queries folder.

3.9.10 Click the Members Queries folder link in the bread crumb trail to check if the two queries are in the folder.



```
SELECT
    m.age,
    count(*) AS member_cnt,
    sum(points_balance) as points_sum,
    (points_sum / member_cnt) :: integer as member_avg
FROM
    MODELED.MEMBERS M
GROUP BY
    1;
```

Figure 22: Checking the Members Queries folder

3.10 Sharing a worksheet

Snowflake has created the ability to share worksheets with other users. The idea is that data analysts, data engineers, or even admins could write SQL code to accomplish very specific goals and tasks, and then share that code with colleagues. Let's learn how to do that now.

3.10.1 Return to the home screen.

3.10.2 Create a new worksheet.

3.10.3 Using the skills you've learned, write a simple query. For example, select member_id and points_balance from the members table and limit the results to 10 rows.

3.10.4 Name the worksheet Shared WS.

3.10.5 Click on the Share button.

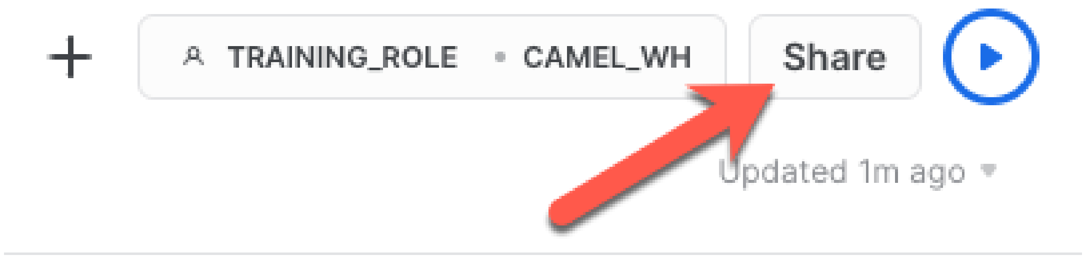
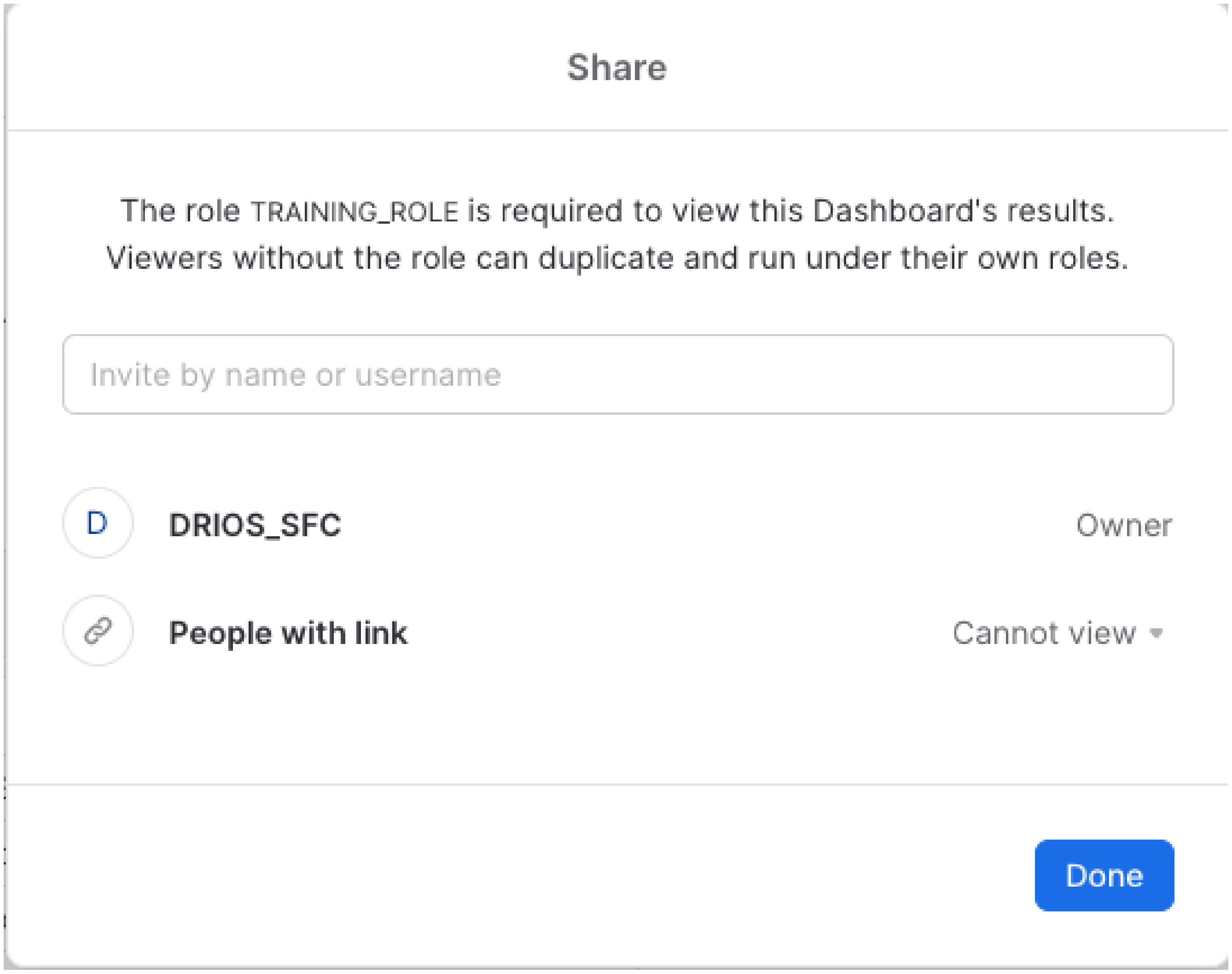


Figure 23: Share button

You should see a dialog box like the one below:



The role TRAINING_ROLE is required to view this Dashboard's results. Viewers without the role can duplicate and run under their own roles.

Figure 24: Share dialog box

3.10.6 Select another user in the class to share by typing their user name in the text box. HINT: If you don't know someone else in the class, go back to the classic web UI and view the WebUI -> History tab to find another user.

If you need help, ask your instructor.

3.11 Using a filter

Now let's practice adding filters to a query and passing values into the filter parameters.

3.11.1 Create a new worksheet named Query Filter.

Create the following query in the worksheet:

```
SELECT m.firstname, m.lastname, m.points_balance, m.started_date FROM
    modeled.members m
     WHERE m.started_date = :daterange;
```

Once you put the SQL code into the query pane, the “Date Range” button should appear above with the text “Last Day”.

3.11.2 Click the Date Range button.

You should see the following dialog box:

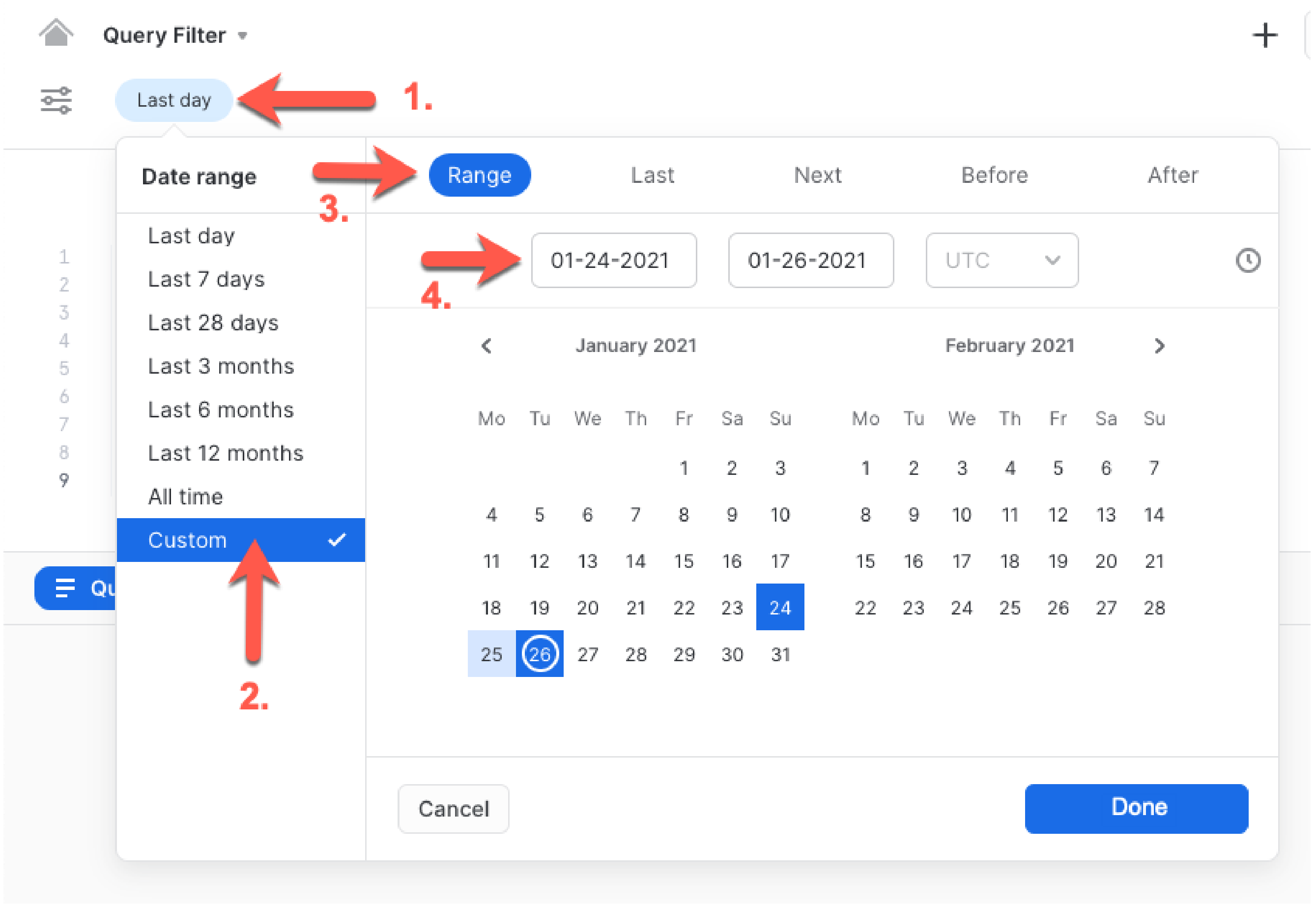


Figure 25: Query filter dialog box

3.11.3 Navigate as shown to choose a custom date range and set the range from 01-20-2019 to 10-06-2020 and click the Apply button to run the query.

You should see results in the result pane. Practice using the column filters to the right of the results pane to gain further insight into the data.

3.12 Working with charts using a GROUP BY Query.

Now let's practice using the charting function.

3.12.1 Return home and select the Points for Members worksheet.

3.12.2 Select the Chart button.

Note that you have a line chart and that the Chart dialog box has appeared to the right.

You should see Line for Chart type. Click on the drop down arrow next to Line. You should see options for Line, Bar, Scatter, Heatgrid and Scorecard.

Now let's modify the Data section of the Chart dialog box.

3.12.3 Set the X-Axis to Age.

3.12.4 Set the Data selection to Member_avg.

3.12.5 Change the look of the chart by clicking on each option in the Appearance section of the Chart dialog box.

3.12.6 Switch to a Bar chart.

Notice the X-Axis stays as Age but the value for the bars changes to Age sum.

3.12.7 Switch the data selection back to Member_avg.

Notice that now you have bars for each age.

3.12.8 Float your arrow over each bar to see the value.

Change the different Aggregation options.

3.13 Working with charts with a query without a GROUP BY.

This is similar to the previous exercise except your SQL statement won't have a GROUP BY clause.

3.13.1 Create a new worksheet named Members Name List.

3.13.2 Add the following query into the worksheet and run the query:

```
SELECT
    (m.firstname || ' ' || m.lastname) as Name,
    m.age,
    m.state,
    m.city,
    m.started_date,
    m.points_balance
FROM
    modeled.members m;
```

3.13.3 Select Chart, then change the Chart type to Bar.

3.13.4 Change the Data from Age to Points_Balance and set the Aggregation to Sum.

3.13.5 Select labels for both axes.

3.13.6 Hover the cursor over various bars in the bar chart and observe that the details for that data are shown.

3.13.7 Try the follow changes to observe the results:

Change the X-Axis to Age.

Change the Orientation .

Change the “Order bars” by selection.

Change the Orientation back.

3.14 Working with a date field as the X-Axis

3.14.1 Change the X-Axis to Started_Date.

Notice the X-Axis label doesn’t change and has to be changed manually.

3.15 Downloading a chart

3.15.1 Using the month chart from the last section, Click the Download Chart button in the top right corner. This is an arrow with a line under it.

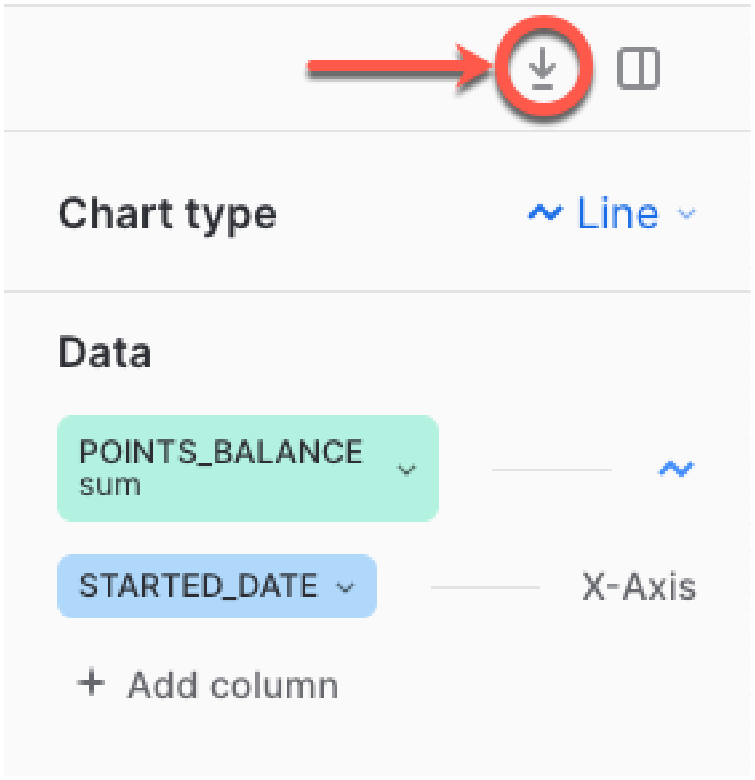


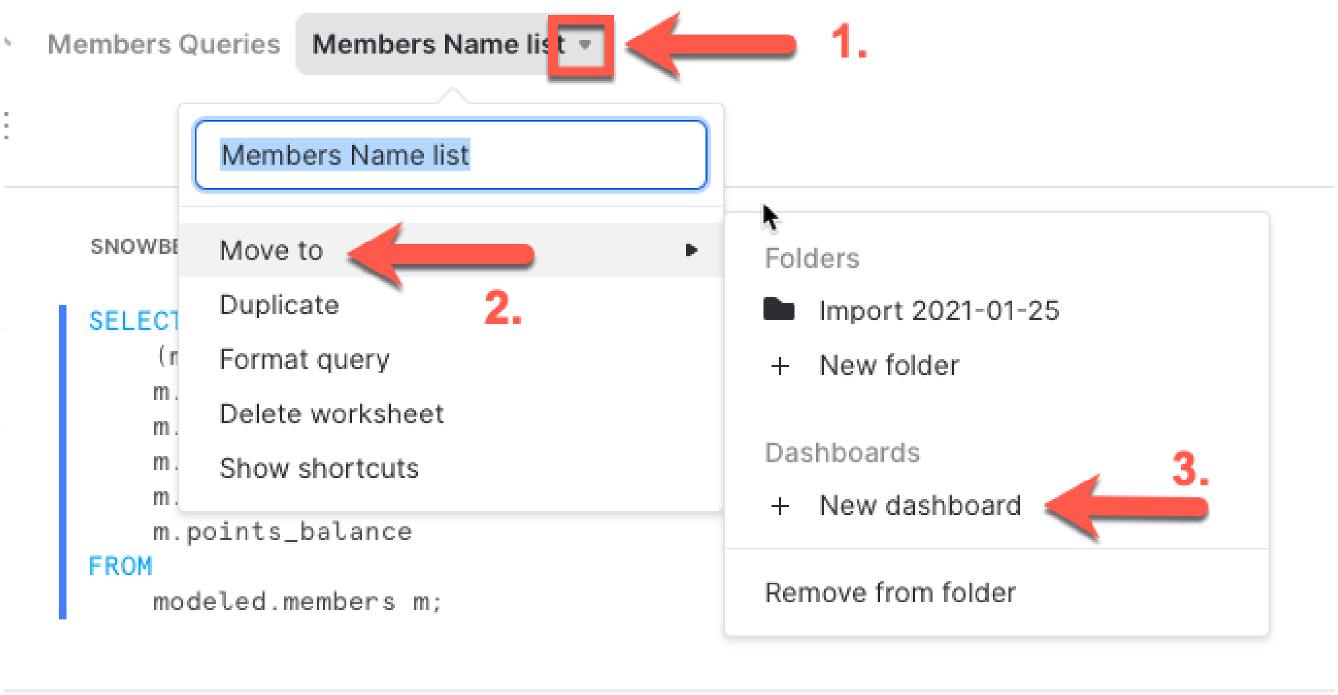
Figure 26: Downloading a Chart

3.15.2 Click on the file at the bottom of the browser to open and view it.

NOTE: This downloads a .png file. This file can be opened in an image viewer.

3.16 Creating a Dashboard

3.16.1 Using the worksheet with the chart from the last section create a new dashboard as shown in the steps below:



```
SELECT
    (r
    m.
    m.
    m.
    m.
    m.points_balance
FROM
    modeled.members m;
```

Figure 27: Creating a New Dashboard

3.16.2 Name the dashboard Member Charts and create the dashboard.

You should now see a screen you are already familiar with. This is where the chart can be edited.

3.16.3 Locate the Return to Member Charts Link at the top of the page and click it.

You should now see the dashboard itself in read only mode.

3.16.4 In the dashboard, select the ellipse (…) in the top right corner and select View Chart to enter edit mode.

3.16.5 Under data, click on the drop down arrow next to the X-Axis field STARTED_DATE. Under Bucketing, select "quarter" to display quarters. Change the X-Axis label to read Quarters instead of STARTED_DATE.

3.16.6 Now let's practice making another chart. Click "Return to Member Charts" at the top, click the home button, click the "Worksheets" tab, then select the Points for Members worksheet.

3.16.7 Select the Chart and create a bar chart with Age as the X-Axis and Member_avg as the bars. Make sure the labels are correct.

3.16.8 Click on the Worksheet name drop down arrow and select Move to -> Member Charts in the dashboards.

3.16.9 Click Return to Member Charts at the top of the screen.

3.16.10 Move the two charts by selecting one and dragging it where you would like it to be.

3.17 Working with the Data section

The data section of the home page's left hand navigation bar allows you to view what data is available to you. Let's familiarize ourselves with that section.

3.17.1 Return to the home page.

3.17.2 In the left hand navigation bar, select the Data->Databases option from the menu.

3.17.3 Select your class database and a schema.

3.17.4 Click on Tables.

3.17.5 Select a table from the list and look at the Details and the columns.

Notice the options in the schema to view all the database objects including functions and stored procedures.

3.17.6 Return to a worksheet and suspend and resize the warehouse:

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

4 Explore Snowflake Caching

Expect this lab to take approximately *45 minutes*.

4.1 Metadata Caching

4.1.1 Open a new worksheet and name it *Caching*.

4.1.2 If you haven't created the class database or warehouse, do it now

```
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
```

4.1.3 Set the context as follows:

- ROLE: TRAINING_ROLE
- WAREHOUSE: [login]_WH
- DATABASE: SNOWFLAKE_SAMPLE_DATA
- SCHEMA: TPCH_SF100

4.1.4 Alternatively, you can set the context using SQL:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCH_SF100;
```

4.1.5 Suspend your warehouse. Be aware you will get an error if your warehouse is already suspended; you can ignore it:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

4.1.6 Run the following query:

```
SELECT MIN(l_orderkey), MAX(l_orderkey), COUNT(*) FROM lineitem;
```

4.1.7 Click the Query ID at the top of the result pane, then click the link to open the profile.

4.1.8 Click the profile tab. It should show that 100% of the result came from metadata cache:

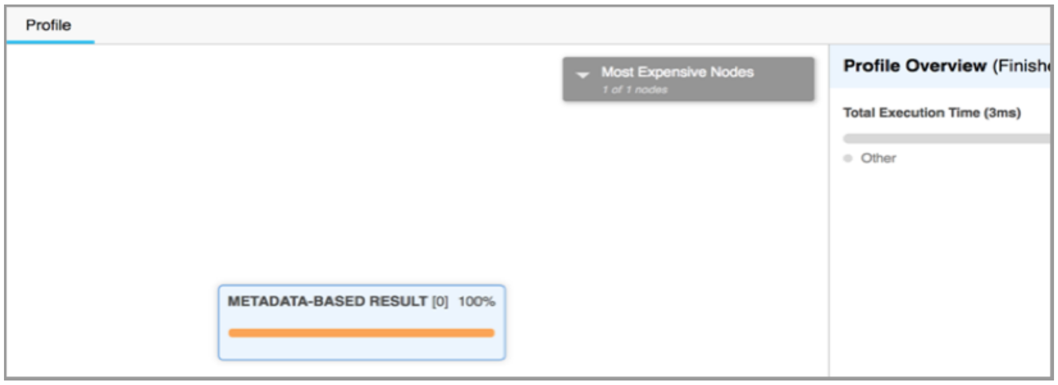


Figure 28: Query Profile - Metadata Cache Result

If your results did not come from the metadata cache, where did they come from? Can you explain why?

4.2 Data Caching in the Compute Cluster

4.2.1 Disable USE_CACHED_RESULT so you are only using metadata cache and the data cache (not the query result cache):

```
ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

4.2.2 Run Query 1 of the TPCH benchmark. This will automatically resume your warehouse:

```
SELECT l_returnflag, l_linestatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (l_discount)) AS sum_disc_price, SUM(l_extendedprice *
        (l_discount) * (1+l_tax))
   AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) as count_order
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

4.2.3 Review the Query Profile and view the metric "Percentage Scanned from Cache."

What do you see? Is it what you expected?

Since the query is being run for the first time on a newly resumed warehouse, the cache will be cold and all data will be read from disk.

4.2.4 Run the following query, with a slightly different WHERE clause:

```
SELECT l_returnflag, l_linestatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (l_discount)) AS sum_disc_price, SUM(l_extendedprice *
       (l_discount) * (1+l_tax))
    AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) as count_order
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
and l_extendedprice <= 20000
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

4.2.5 Review the Query Profile.

See what has happened to the caching metric. It should have increased, since this query has a similar pattern to the previous query so it could reuse some data from the data cache.

4.2.6 Run the following query which is similar, but JOINs the data to another table:

```
SELECT l_orderkey,
SUM(l_extendedprice*(l_discount)) AS revenue,
o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE C_mktsegment = 'BUILDING'
  AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < to_date('1995-03-15')
  AND l_shipdate > to_date('1995-03-15')
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY 2 DESC, o_orderdate
LIMIT 10;
```

4.2.7 Review the Query Profile. Do you see what you expect to see?

There will still be some Percentage Scanned from Cache, but it will not be as high as in the previous step. This is because you joined it to another table, so more of the data was not already loaded into cache.

4.3 Explore Query Result Caching

4.3.1 Suspend your warehouse, to clear the data cache:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

4.3.2 Set USE_CACHED_RESULT back to TRUE:

```
ALTER SESSION SET USE_CACHED_RESULT=TRUE;
```

4.3.3 Set the Worksheet contexts as follows:

- ROLE: TRAINING_ROLE
- WAREHOUSE: [login]_WH
- SCHEMA: SNOWFLAKE_SAMPLE_DATA.TPCH_SF100

4.3.4 Alternatively, you can set the context using SQL:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE SNOWFLAKE_SAMPLE_DATA.TPCH_SF100;
```

4.3.5 Run Query 1 of the TPCH benchmark:

```
SELECT l_returnflag, l_linestatus,
    SUM(l_quantity) AS [login]_sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (l_discount) * (1+l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

4.3.6 Check the query profile. How much cache was used?

4.3.7 Suspend your warehouse:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

4.3.8 Rerun Query 1 from the previous step. What do you think will happen?

4.3.9 Bring up the Query Profile and check it. The query completed without using a warehouse, because the results were accessible in the query result cache.

4.3.10 Grant privileges to your warehouse to the role SYSADMIN, and then change to that role:

```
GRANT USAGE ON WAREHOUSE [login]_WH TO ROLE SYSADMIN;
USE ROLE SYSADMIN;
```

4.3.11 Re-run the query. What do you think will happen?

```
SELECT l_returnflag, l_linestatus,
    SUM(l_quantity) AS [login]_sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (l_discount) * (1+l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

4.3.12 Check the query profile. Did the new role use any cache? Why or why not?

Your session can use the query result cache, even if the query was originally performed by a different role–as long as your role has the same SELECT privileges on the tables involved in the query.

4.3.13 Open a new worksheet, to start a new session.

4.3.14 Set your context to TRAINING_ROLE, SNOWFLAKE_SAMPLE_DATA, TPCH_SF100, and [login]_WH:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCH_SF100;
```

4.3.15 Run the query again. What do you think will happen?

4.3.16 Check the query profile. Was the query cached used? Why or why not?

Answer: The query cache was used, because it was run by someone in the same role. So the query cache can be used across sessions as long as it is in the same role.

4.3.17 Change the line `AS [login]_sum_qty` to `AS [login]_sum_qty_new` and run the query again. What do you think will happen?

```
SELECT l_returnflag, l_linestatus,
    SUM(l_quantity) AS [login]_sum_qty_new,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (l_discount) * (1+l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

4.3.18 Check the query profile and the percentage of cache used.

The query result cache was not used, because the query was not identical. However, most of the result was able to use the data cache, stored on the virtual warehouse.

4.3.19 Suspend and resize the warehouse

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

5 Review the Query Profile

Expect this lab to take approximately *15 minutes*.

5.1 Run the first explain plan

5.1.1 Navigate to **[Worksheets]** and create a new worksheet named *Query Profile*.

5.1.2 Set the worksheet context as follows:

- ROLE: TRAINING_ROLE
- WAREHOUSE: **[login]**_WH
- WAREHOUSE SIZE: Extra Small
- DATABASE: SNOWFLAKE_SAMPLE_DATA
- SCHEMA: TPCDS_SF10TCL

5.1.3 Alternatively, execute the following SQL:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = xsmall;
USE SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL;
```

5.1.4 Disable the query result cache:

```
ALTER SESSION SET USE_CACHED_RESULT=FALSE;
```

5.1.5 Run an explain plan on the sample query with a LIMIT:

```
EXPLAIN
SELECT c_customer_sk,
        c_customer_id,
        c_last_name,
        (ca_street_number || ' ' || ca_street_name),
        ca_city,  ca_state
    FROM customer, customer_address
    WHERE c_customer_id = ca_address_id
    AND c_customer_sk between 100000 and 600000
    ORDER BY ca_city, ca_state
    LIMIT 10;
```

5.1.6 Examine the explain plan

- What does the explain plan show for the order the tables will be accessed?
- How much partitions are read on each table and why is the query not performing a full table scan on the customer table?
- OPTIONAL: Click the download button and create a CSV file of the explain plan. This will allow you to compare this plan to the next one.

5.2 Run the same query without the explain plan

5.2.1 Run the SQL below:

```
SELECT c_customer_sk,
        c_customer_id,
        c_last_name,
        (ca_street_number || ' ' || ca_street_name),
        ca_city,  ca_state
    FROM customer, customer_address
    WHERE c_customer_id = ca_address_id
    AND c_customer_sk between 100000 and 600000
    ORDER BY ca_city, ca_state
    LIMIT 10;
```

5.3 Perform a Review of the Query Profile

5.3.1 In the worksheet in the Results section click Query ID.

5.3.2 Once the ID shows, click on it; the detail page for the query is displayed:

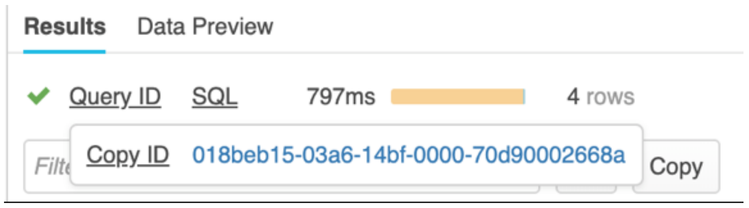


Figure 29: Query Id Details

5.3.3 Click the profile tab and review the Query Profile:

- Did the explain plan show the correct number of partitions read?
- Notice the step number for the SortWithLimit operation.
- How did the filter on the customer table affect the number of partitions?
- Click the tableScan step for the custom table. Notice the pruning on the partitions.

5.3.4 Navigate back to the worksheet.

5.4 Run an explain plan on the query without the limit

5.4.1 Run the SQL below:

```
EXPLAIN
SELECT c_customer_sk,
        c_customer_id,
        c_last_name,
        (ca_street_number || ' ' || ca_street_name),
        ca_city,  ca_state
    FROM customer, customer_address
    WHERE c_customer_id = ca_address_id
    AND c_customer_sk between 100000 and 600000
    ORDER BY ca_city, ca_state;
```

5.4.2 Compare this plan to the one created above.

- How did removing the limit change the number of partitions being accessed?

5.5 Run the same query without the limit or the explain plan

5.5.1 Run the SQL below:

```
SELECT c_customer_sk,
       c_customer_id,
       c_last_name,
       (ca_street_number || ' ' || ca_street_name),
       ca_city,  ca_state
    FROM customer, customer_address
    WHERE c_customer_id = ca_address_id
    AND c_customer_sk between 100000 and 600000
    ORDER BY ca_city, ca_state;
```

5.5.2 Open the query profile for this.

- Notice this query had some spillage to the local storage.
- Click through the steps. Which steps had the spillage?
- What is the step number for the sort operation?
- Notice the sort is now performed after the data is read, not during the scanning of the table.
- Did we get the same pruning with the customer table?

5.5.3 Navigate to **[History]**. You should see the query you just ran at the top of the table.

5.5.4 Click on another query ID from the list - this is another way to get the query profile page.

5.5.5 View the profile for the query you selected.

5.5.6 Run some additional queries on your own or select other queries from the **[History]** table, and view the query results.

5.5.7 If you haven't already, make sure you resize the warehouse to XSMALL and then suspend it:

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

6 Loading and Unloading Structured Data

Expect this lab to take approximately *30 minutes*.

6.1 Create Tables and File Formats

This exercise will load the `region.tbl` file into a REGION table in your Database. The `region.tbl` file which is already in the cloud is pipe ('|') delimited. It has no header and contains the following five rows:

```
0|AFRICA|lar deposits. blithely final packages cajole. regular waters are final
   requests. regular accounts are according to |
1|AMERICA|hs use ironic, even requests. S|
2|ASIA|ges. thinly even pinto beans ca|
3|EUROPE|ly final courts cajole furiously final excuse|
4|MIDDLE EAST|uickly special accounts cajole carefully blithely close requests.
   carefully final asymptotes haggle furiousl|
```

NOTE: There is a delimiter at the end of every line, which by default is interpreted as an additional column by the COPY INTO statement.

6.1.1 Navigate to **[Worksheets]** and create a new worksheet named *Load Structured Data.* Use the following SQL to set the context:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
USE WAREHOUSE [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
USE [login]_DB.PUBLIC;
```

6.1.2 Execute all of the CREATE TABLE statements:

```
CREATE OR REPLACE TABLE REGION (
       R_REGIONKEY NUMBER(38,0)  NOT NULL,
       R_NAME      VARCHAR(25)   NOT NULL,
       R_COMMENT   VARCHAR(152)
);
```

6.1.3 Create a file format called `MYPIPEFORMAT`, that will read the pipe-delimited `region.tbl` file:

```
CREATE OR REPLACE FILE FORMAT MYPIPEFORMAT
  TYPE = CSV
  COMPRESSION = NONE
  FIELD_DELIMITER = '|'
  FILE_EXTENSION = 'tbl'
  ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE;
```

6.1.4 Create a file format called `MYGZIPPIPEFORMAT` that will read the compressed version of the region.tbl file. It should be identical to the `MYPIPEFORMAT`, except you will set `COMPRESSION = GZIP`.

```
CREATE OR REPLACE FILE FORMAT MYGZIPPIPEFORMAT
 TYPE = CSV
 COMPRESSION = GZIP
 FIELD_DELIMITER = '|'
 FILE_EXTENSION = 'tbl'
 ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE;
```

6.2 Load the *region.tbl* File

The files for this task have been pre-loaded into a location on AWS. The external stage that points to that location has been created for you. The stage is in the TRAININGLAB schema of the TRAINING_DB database. In this task you will review the files in the stage, and load them using the file formats you created.

6.2.1 Review the properties of the stage:

```
DESCRIBE STAGE TRAINING_DB.TRAININGLAB.ED_STAGE;
```

NOTE: The file format defined in the stage is not quite right for this data. In particular, the field delimiter is set to a comma. You have two choices - you could either modify they file format definition in the stage itself, or you could specify a different file format with the COPY INTO command. You will use your MYPIPEFORMAT file format.

6.2.2 Confirm the file is in the external stage with the list command:

```
LIST @training_db.traininglab.ed_stage/load/lab_files/ pattern='.*region.*';
```

6.2.3 Load the data from the external stage to the REGION table, using the file format you created in the previous task:

```
COPY INTO REGION
FROM @training_db.traininglab.ed_stage/load/lab_files/
FILES = ('region.tbl')
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT);
```

6.2.4 Select and review the data in the REGION table, either by executing the following command in your worksheet or by using **Preview Data** in the sidebar:

```
SELECT * FROM REGION;
```

6.3 Load a GZip Compressed File

This exercise will reload the REGION Table from a gzip compressed file that is in the external stage. You will use your MYGZIPPIPEFORMAT file format.

For these next steps, you will use a smaller warehouse.

6.3.1 Empty the REGION Table in the PUBLIC schema of **[login]**_DB:

```
TRUNCATE TABLE region;
```

6.3.2 Confirm that the `region.tbl.gz` file is in the external stage:

```
LIST @training_db.traininglab.ed_stage/load/lab_files/ pattern='.*region.*';
```

6.3.3 Reload the REGION table from the region.tbl.gz file. Review the syntax of the COPY INTO command used in the previous task. Specify the file to COPY as 'region.tbl.gz'.

```
COPY INTO region
FROM @training_db.traininglab.ed_stage/load/lab_files/
FILES = ('region.tbl.gz')
FILE_FORMAT = ( FORMAT_NAME = MYGZIPPIPEFORMAT);
```

6.3.4 Query the table to view the data:

```
SELECT * FROM region;
```

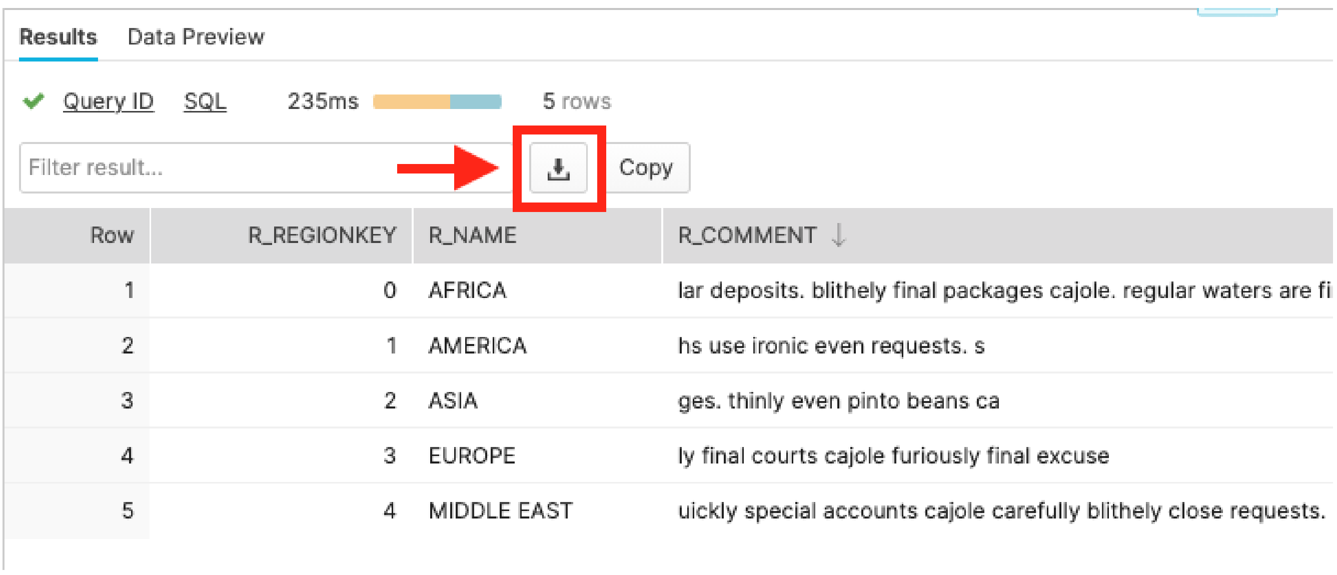
6.4 Load data using the Load Data Wizard

In this portion of the lab you'll learn how to load data from a file on your desktop using the Load Data Wizard. You are going to generate the file to load in CSV format, create your own file format, and load the data using the Wizard.

6.4.1 First, run the following query to generate the file you're going to load. Note that it uses the REPLACE function to remove any stray commas in the R_COMMENT field. Since we're going to ultimately load the file using a CSV file format, this will ensure that all rows will load without any inadvertent errors.

```
SELECT R_REGIONKEY, R_NAME, REPLACE(R_COMMENT, ',') AS R_COMMENT FROM region;
```

6.4.2 You should see five rows in the results. Click the download button to download the file:



Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fi
2	1	AMERICA	hs use ironic even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests.

Figure 30: Download Button

6.4.3 You will see a dialog box offering you either TSV or CSV format. Select CSV and click Export.

6.4.4 Your file should have saved to the location you selected. Examine the file using the text editor of your choice. The file should have a header row and five rows of data.

6.4.5 Now let's create a pipe format using the user interface. In the Web UI, click the Databases button in the ribbon at the top, select your database, then click the File Formats tab.

6.4.6 You should see a list of existing file formats, including the ones you created. Click Create.

6.4.7 The Create File Format dialog box should appear. Type MYCSVFORMAT in the Name text box. As you can see, the other options have been automatically selected for you: CSV as Format Type, Auto as Compression Method, and Comma as Column Separator. Go ahead and click Finish.

6.4.8 Your new file format should now be listed. Now click the Tables tab and select the row for table REGION.

6.4.9 Click the Load Table option just above the table list.

6.4.10 You should now see the Load Data dialog box. You will have to select the warehouse, source files, file format and load options. Select **[login]**_WH and click Next.

6.4.11 You should see an option to load a file from your computer. Click the Select Files button, navigate to and select the file you downloaded, then click Next.

6.4.12 You should now see an option to select a file format. Select MYCSVFORMAT and click Next.

6.4.13 You should now see a list of Load Options. These give you different options for how errors are handled while parsing the file. We want to load the data despite any errors, so select “Continue loading valid data from the file” and click Load.

6.4.14 You should see a dialog box saying files are being encrypted and staged. Then you should see the load results showing that 6 rows were parsed and 5 rows were loaded. Evaluate the column First Error.

6.4.15 The error should read “Numeric value ‘R_REGIONKEY’ is not recognized”. This means that the column headers in the file weren’t loaded. That is okay because we obviously didn’t want to load them anyway.

6.4.16 Now go back to your worksheet and run the following SQL statement:

```
SELECT * FROM region;
```

6.4.17 You should have ten rows in the REGION table now.

6.5 Unload a Pipe-Delimited File to a Table Stage

6.5.1 Open a new worksheet and set the context as follows:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
USE WAREHOUSE [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
USE [login]_DB.PUBLIC;
```

6.5.2 Create a fresh version of the `REGION` table with 5 records to unload:

```
create or replace table REGION as
select * from SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.REGION;
```

6.5.3 Unload the data to the REGION table stage. Remember that a table stage is automatically created for each table. Use the slides, workbook, or Snowflake documentation for questions on the syntax. You will use MYPIPEFORMAT for the unload:

```
COPY INTO @%region
FROM region
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT);
```

6.5.4 List the stage and verify that the data is there:

```
LIST @%region;
```

6.5.5 OPTIONAL: Use GET to download the file to your local system. Open it with an editor and see what it contains.

NOTE: The GET command is not supported in the GUI; use the SnowSQL CLI:

```
GET @%region file:///<path to dir> ; -- this is for MAC
GET @%region file://c:<path to dir>; -- this is for windows
```

6.5.6 Remove the file from the REGION table's stage:

```
REMOVE @%region;
```

6.6 Use a SQL statement containing a JOIN to Unload a Table into an internal stage

6.6.1 Do a SELECT with a JOIN on the `REGION` and `NATION` tables. You can JOIN on any column you wish. Review the output from your JOIN.

```
SELECT *
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."REGION" r
JOIN "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."NATION" n ON r.r_regionkey =
   n.n_regionkey;
```

6.6.2 Create a named stage (you can call it whatever you want):

```
CREATE OR REPLACE STAGE mystage;
```

6.6.3 Unload the JOINed data into the stage you created:

```
COPY INTO @mystage FROM
(SELECT * FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."REGION" r JOIN
   "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."NATION" n
ON r.r_regionkey = n.n_regionkey);
```

6.6.4 Verify the file is in the stage:

```
LIST @mystage;
```

6.6.5 OPTIONAL: Use GET to download the file to your local system and review it (requires SnowSQL)

```
GET @mystage file:///<path> -- for Mac
GET @mystage file://c:<path -- For Windows
```

6.6.6 Remove the file from the stage:

```
REMOVE @mystage;
```

6.6.7 Remove the stage:

```
DROP STAGE mystage;
```

6.6.8 Suspend and resize the warehouse

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

7 Access Control and User Management

Expect this lab to take approximately *40 minutes*.

Lab Purpose: Students will work with the Snowflake security model and learn how to create roles, grant privileges, build, and implement basic security models.

7.1 Determine Privileges (GRANTs)

7.1.1 Navigate to **[Worksheets]** and create a new worksheet named *Managing Security*.

7.1.2 If you haven't created the class database or warehouse, do it now

```
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
```

7.1.3 Run these commands to see what has been granted to you as a user, and to your roles:

```
SHOW GRANTS TO USER [login];
SHOW GRANTS TO ROLE TRAINING_ROLE;
SHOW GRANTS TO ROLE SYSADMIN;
SHOW GRANTS TO ROLE SECURITYADMIN;
```

NOTE: The TRAINING_ROLE has some specific privileges granted - not all roles in the system would be able to see these results.

7.2 Work with Role Permissions

7.2.1 Change your role to SECURITYADMIN:

```
USE ROLE SECURITYADMIN;
```

7.2.2 Create two new custom roles, called `[login]_CLASSIFIED` and `[login]_GENERAL` :

```
CREATE ROLE [login]_CLASSIFIED;
CREATE ROLE [login]_GENERAL;
```

7.2.3 GRANT both roles to SYSADMIN, and to your user:

```
GRANT ROLE [login]_CLASSIFIED, [login]_GENERAL TO ROLE SYSADMIN;
GRANT ROLE [login]_CLASSIFIED, [login]_GENERAL TO USER [login];
```

7.2.4 Change to the role SYSADMIN, so you can assign permissions to the roles you created:

```
USE ROLE SYSADMIN;
```

7.2.5 Create a warehouse named `[login]_SHARED_WH` :

```
CREATE WAREHOUSE [login]_SHARED_WH;
```

7.2.6 Grant both new roles privileges to use the shared warehouse:

```
GRANT USAGE ON WAREHOUSE [login]_SHARED_WH
  TO ROLE [login]_CLASSIFIED;
GRANT USAGE ON WAREHOUSE [login]_SHARED_WH
  TO ROLE [login]_GENERAL;
```

7.2.7 Create a database called `[login]_CLASSIFIED_DB` :

```
CREATE DATABASE [login]_CLASSIFIED_DB;
```

7.2.8 Grant the role `[login]_CLASSIFIED` all necessary privileges to create tables on any schema in `[login]_CLASSIFIED_DB` :

```
GRANT USAGE ON DATABASE [login]_CLASSIFIED_DB
TO ROLE [login]_CLASSIFIED;
GRANT USAGE ON ALL SCHEMAS IN DATABASE [login]_CLASSIFIED_DB
TO ROLE [login]_CLASSIFIED;
GRANT CREATE TABLE ON ALL SCHEMAS IN DATABASE [login]_CLASSIFIED_DB
TO ROLE [login]_CLASSIFIED;
```

7.2.9 Use the role `[login]_CLASSIFIED` , and create a table called `SUPER_SECRET_TBL` inside the `[login]_CLASSIFIED_DB.PUBLIC` schema:

```
USE ROLE [login]_CLASSIFIED;
USE [login]_CLASSIFIED_DB.PUBLIC;
CREATE TABLE SUPER_SECRET_TBL (id INT);
```

7.2.10 Insert some data into the table:

```
INSERT INTO SUPER_SECRET_TBL VALUES (1), (10), (30);
```

7.2.11 Assign `GRANT SELECT` privileges on `SUPER_SECRET_TBL` to the role `[login]_GENERAL`:

```
GRANT SELECT ON SUPER_SECRET_TBL TO ROLE [login]_GENERAL;
```

7.2.12 Use the role `[login]_GENERAL` to `SELECT *` from the table `SUPER_SECRET_TBL`:

```
USE ROLE [login]_GENERAL;
SELECT * FROM [login]_CLASSIFIED_DB.PUBLIC.SUPER_SECRET_TBL;
```

What happens? Why?

7.2.13 Grant role `[login]_GENERAL` usage on all schemas in `[login]_CLASSIFIED_DB`:

```
USE ROLE SYSADMIN;
GRANT USAGE ON DATABASE [login]_CLASSIFIED_DB TO ROLE [login]_GENERAL;
GRANT USAGE ON ALL SCHEMAs IN DATABASE [login]_CLASSIFIED_DB TO ROLE
    [login]_GENERAL;
```

7.2.14 Now try again:

```
USE ROLE [login]_GENERAL;
SELECT * FROM [login]_CLASSIFIED_DB.PUBLIC.SUPER_SECRET_TBL;
```

7.2.15 Drop the database `[login]_CLASSIFIED_DB`:

```
USE ROLE SYSADMIN;
DROP DATABASE [login]_CLASSIFIED_DB;
```

7.2.16 Drop the roles `[login]_CLASSIFIED` and `[login]_GENERAL`:

```
USE ROLE SECURITYADMIN;
DROP ROLE [login]_CLASSIFIED;
DROP ROLE [login]_GENERAL;
```

HINT: What role do you need to use to do this?

7.3 Create Parent and Child Roles

7.3.1 Change your role to SECURITYADMIN:

```
USE ROLE SECURITYADMIN;
```

7.3.2 Create a parent and child role, and GRANT the roles to the role SYSADMIN. At this point, the roles are peers (neither one is below the other in the hierarchy):

```
CREATE ROLE [login]_child;
CREATE ROLE [login]_parent;
GRANT ROLE [login]_child, [login]_parent TO ROLE SYSADMIN;
```

7.3.3 Give your user name privileges to use the roles:

```
GRANT ROLE [login]_child, [login]_parent TO USER [login];
```

7.3.4 Change your role to SYSADMIN:

```
USE ROLE SYSADMIN;
```

7.3.5 Grant the following object permissions to the child role:

```
GRANT USAGE ON WAREHOUSE [login]_WH TO ROLE [login]_child;
GRANT USAGE ON DATABASE [login]_DB TO ROLE [login]_child;
GRANT USAGE ON SCHEMA [login]_DB.PUBLIC TO ROLE [login]_child;
GRANT CREATE TABLE ON SCHEMA [login]_DB.PUBLIC
   TO ROLE [login]_child;
```

7.3.6 Use the child role to create a table:

```
USE ROLE [login]_child;
USE WAREHOUSE [login]_WH;
USE DATABASE [login]_DB;
USE SCHEMA [login]_DB.PUBLIC;
CREATE TABLE genealogy (name STRING, age INTEGER, mother STRING,
   father STRING);
```

7.3.7 Verify that you can see the table:

```
SHOW TABLES LIKE '%genealogy%';
```

7.3.8 Use the parent role and view the table:

```
USE ROLE [login]_parent;
SHOW TABLES LIKE '%genealogy%';
```

You will not see the table, because the parent role has not been granted access.

7.3.9 Change back to the SECURITYADMIN role and change the hierarchy so the child role is beneath the parent role:

```
USE ROLE SECURITYADMIN;
GRANT ROLE [login]_child to ROLE [login]_parent;
```

7.3.10 Use the parent role, and verify the parent can now see the table created by the child:

```
USE ROLE [login]_parent;
SHOW TABLES LIKE '%genealogy%';
```

7.3.11 Suspend and resize the warehouse

```
USE ROLE TRAINING_ROLE;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

8 Explore Semi-Structured JSON Data

In this lab you will practice using Snowflake's SQL extensions to query and work with semi-structured data. The lab provides exercises for you to work directly with historical weather station semi-structured data and see how to transform it into standard data structures.

In order to do this lab, you can key SQL commands presented in this lab directly into a worksheet. You can also use the code file for this lab that was provided at the start of the class. To use the file, simply drag and drop it into an open worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

8.1 Review the Weather Data

8.1.1 Create a new worksheet and name it *Explore Weather Data*.

8.1.2 Set your worksheet context as follows:

- ROLE: TRAINING_ROLE

- WAREHOUSE: **[login]**_WH
- DATABASE: TRAINING_DB
- SCHEMA: WEATHER

8.1.3 Use the following commands within your worksheet:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH
WAREHOUSE_SIZE=XSmall
INITIALLY_SUSPENDED=True
AUTO_SUSPEND=300;
USE WAREHOUSE [login]_WH;
USE SCHEMA TRAINING_DB.WEATHER;
```

8.1.4 Use the DESCRIBE TABLE command to describe the isd_2019_daily table:

```
DESCRIBE TABLE isd_2019_total;
```

Notice that this table contains just two columns: V (variant) and T (timestamp).

8.1.5 Select all columns from the isd_2019_daily Table, limit the results to 10 rows, and explore the data and its content:

```
SELECT * FROM isd_2019_total
LIMIT 10;
```

8.1.6 In the Results set, click on the JSON data to pull up the Details pane with the VARIANT data.

This will display the structure of a single record of the JSON data.

8.1.7 Use FLATTEN to extract the top-level keys from the V column. Use LIMIT 10:

```
SELECT v, key, value FROM isd_2019_total w,
LATERAL FLATTEN (input => w.v)
LIMIT 10;
```

8.1.8 Run a query to extract the time, station, country, elevation, temp, and other weather objects from the table:

```
SELECT v:data.observations[0].dt AS time,
       v:station.name AS station,
       v:station.country AS country,
       v:station.elev AS elevation,
       v:data.observations[0].air.temp AS temp_celsius,
       v:data.observations[0].air."dew-point" AS dew_point,
       v:data.observations[0].wind."speed-rate" AS wind_speed
FROM weather.isd_2019_total
LIMIT 10;
```

8.1.9 Run the following to display the types of columns from the last run query:

```
DESCRIBE RESULT LAST_QUERY_ID();
```

NOTE: The values returned are all VARIANT.

8.1.10 Return to the previous instance as that data pertains more to what we are trying to do. Run the previous query again:

```
SELECT v:data.observations[0].dt AS time,
       v:station.name AS station,
       v:station.country AS country,
       v:station.elev AS elevation,
       v:data.observations[0].air.temp AS temp_celsius,
       v:data.observations[0].air."dew-point" AS dew_point,
       v:data.observations[0].wind."speed-rate" AS wind_speed
FROM weather.isd_2019_total
LIMIT 10;
```

8.1.11 Evaluate the results.

Do you notice anything unusual about the STATION column?

The values are enclosed in double quotes, which indicates that the column is not a VARCHAR, but a VARIANT. In the query performed, you've extracted a portion of the VARIANT but have not yet converted it to a native SQL type.

8.1.12 Remove the double quotes around the values by casting each value as a data type other than VARIANT:

```
SELECT v:data:observations[0].dt::DATETIME AS time,
       v:station.name::VARCHAR AS station,
       v:station.country::VARCHAR AS country,
       v:station.elev::NUMBER(8,4) AS elevation,
       v:data.observations[0].air.temp::NUMBER(8,4) AS temp_celsius,
       v:data.observations[0].air."dew-point"::NUMBER(8,4) AS dew_point,
       v:data.observations[0].wind."speed-rate"::NUMBER AS wind_speed
FROM weather.isd_2019_total
LIMIT 10;
```

8.1.13 Run the following to display the types of columns from the last run query:

```
DESCRIBE RESULT LAST_QUERY_ID();
```

Note the values returned have been updated.

8.2 Extract and Transform the Data

8.2.1 Extract the weather station reported air temperature data. Instead of keeping the value a VARIANT, cast it to a NUMBER(38,1):

```
SELECT v:station.name::VARCHAR AS station,
       v:station.country::VARCHAR AS country,
       v:data.observations[0].air.temp::NUMBER(38,1) AS temp_celsius
FROM weather.isd_2019_total
WHERE country = 'FR';
```

8.2.2 Explore the results.

Are you seeing any large numbers?

NOTE: Some weather stations were unable to record data and an observation quality code column should be evaluated to avoid skew data readings when performing data analysis.

8.2.3 Extract the weather station reported air temperature data that has pass all quality control checks. Rename the column temp_celsius and include only temperature data with a temp-quality-code equal to “1”:

```
SELECT v:station.name::VARCHAR AS station,
       v:station.country::VARCHAR AS country,
       v:data.observations[0].air.temp::NUMBER(38,1) AS temp_celsius
FROM weather.isd_2019_total
WHERE country = 'FR'
AND v:data.observations[0].air."temp-quality-code" = '1';
```

8.2.4 Limit the results to records for a 7 day period starting on 2020-08-14 and ending on 2020-08-21 (you must cast the time column as a date, so it matches the type of output from the TO_DATE function):

```
SELECT v:data.observations[0].air.temp::NUMBER(38,1) AS temp_celsius,
       v:data.observations[0].dt::DATE AS date
FROM weather.isd_2019_total
WHERE date >= to_date('2019-08-14') AND date <= to_date('2019-08-21')
    AND v:data.observations[0].air."temp-quality-code" = '1'
ORDER BY date;
```

8.2.5 Write a query to pull country, station name, and highest temperatures for a two (2) day period starting on 2019-08-14 and ending on 2019-08-16.

```
SELECT v:station.country::VARCHAR AS country,
       v:station.name::VARCHAR AS station,
       MAX(v:data.observations[0].air.temp)::NUMBER(38,1) AS max_celsius
FROM weather.isd_2019_total
WHERE v:data.observations[0].dt::date >= to_date('2019-08-14')
    AND v:data.observations[0].dt::date <= to_date('2019-08-16')
    AND v:data.observations[0].air."temp-quality-code" = '1'
GROUP BY country, station;
```

8.2.6 Run the same query, but return the temperature in Fahrenheit instead of Celsius.

The formula is Fahrenheit = (Celsius * 9/5 + 32).

```
SELECT v:station.country::VARCHAR AS country,
       v:station.name::VARCHAR AS station,
       MAX(v:data.observations[0].air.temp) AS max_celsius,
       (MAX(v:data.observations[0].air.temp) * 9/5 + 32) AS max_fahrenheit
FROM weather.isd_2019_total
WHERE v:data.observations[0].dt::date >= to_date('2019-08-14')
    AND v:data.observations[0].dt::date <= to_date('2019-08-16')
    AND v:data.observations[0].air."temp-quality-code" = '1'
GROUP BY country, station;
```

8.2.7 Run the following command, then click on one of the values in the “V” column to see the format of the VARIANT:

```
SELECT * FROM isd_2019_daily
LIMIT 10;
```

Notice there is a “data” key and a “station” key, each with nested values.

8.2.8 Run a query to return 10 rows showing the timestamp, the station name, and the entire data VARIANT.

```
SELECT t, v:station.name, v AS variant
FROM weather.isd_2019_daily
LIMIT 10;
```

8.2.9 Click on a row in the data column, to see the structure of the data VARIANT. What additional information can you extract from the JSON data provided?

8.2.10 Use the FLATTEN table function (with a LATERAL JOIN) to iterate through the objects in the observations field and extract air temperature and the observation time and air temperature value:

```
SELECT weather.t as date,
       weather.v:station.name::VARCHAR AS station,
       weather.v:station.country::VARCHAR AS country,
       observations.value:air.temp::NUMBER(38,1) AS temp_celsius,
       observations.value:dt::timestamp_ntz AS time
FROM weather.isd_2019_daily weather,
LATERAL FLATTEN(input => v:data.observations) observations
LIMIT 100;
```

Note the results: the first three (3) columns (DATE, STATION and COUNTRY) repeat for several rows while the remaining columns change.

Can you explain why those first columns repeat, but the remaining columns change every row?

Does the number of rows where the values in the first two columns repeat give you any information on the size of the data array you just flattened?

8.2.11 Run a query to calculate the average, max and min air temperature in Celsius and Fahrenheit for each weather station.

8.2.12 Run a query to calculate the average, max and min air temperature in Celsius and Fahrenheit for each weather station. Limit the results to records for a 7 day period starting on 2020-08-14 and ending on 2020-08-21 and air temperature values that has pass all quality control checks:

```
SELECT weather.t as date,
       weather.v:station.name::VARCHAR AS station,
       weather.v:station.country::VARCHAR AS country,
       AVG(observations.value:air.temp)::NUMBER(38,1) as avg_temp_c,
       MIN(observations.value:air.temp) as min_temp_c,
       MAX(observations.value:air.temp) as max_temp_c,
       (AVG(observations.value:air.temp) * 9/5 + 32)::NUMBER(38,1) as avg_temp_f,
       (MIN(observations.value:air.temp) * 9/5 + 32) as min_temp_f,
       (MAX(observations.value:air.temp) * 9/5 + 32) as max_temp_f
FROM weather.isd_2019_daily weather,
LATERAL FLATTEN(input => v:data.observations) observations
WHERE observations.value:air."temp-quality-code" = '1'
    AND date >= to_date('2019-08-14') AND date <= to_date('2019-08-21')
GROUP BY country, date, station;
```

8.2.13 Suspend the warehouse:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

9 Continuous Data Protection

Lab Purpose: You will work with Snowflake’s cloning and TimeTravel features. You will have the opportunity to clone a table, drop and then undrop a table, and experience how to work with Time Travel.

9.1 Use Zero-Copy Clone to Copy Database Objects

In this task you're going to clone the LINEITEM table into the Training_DB database from the Public Schema in the Snowflake_Sample_Database. You'll notice the speed at which you're able to clone an incredibly large Table in Snowflake.

9.1.1 Navigate to **[Worksheets]**.

9.1.2 Open a new worksheet

9.1.3 If you haven't created the class database or warehouse, do it now

```
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
```

9.1.4 set your standard context.

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE DATABASE [login]_DB;
USE SCHEMA PUBLIC;
```

9.1.5 Create a copy of the LINEITEM table that is found in the TRAIING_DB.TRAININGLAB schema:

```
CREATE TABLE lineitem_copy AS
SELECT * FROM TRAINING_DB.TRAININGLAB.LINEITEM;
```

9.1.6 Create a CLONE of the copy you just created:

```
CREATE TABLE lineitem_clone
    CLONE lineitem_copy;
```

9.1.7 COUNT the rows in the LINEITEM_COPY table:

```
SELECT COUNT(*) FROM lineitem_copy;
```

9.1.8 COUNT the rows in the LINEITEM_CLONE table:

```
SELECT COUNT(*) FROM lineitem_clone;
```

9.1.9 Insert some values into the LINEITEM_COPY table:

```
INSERT INTO lineitem_copy (l_comment, l_shipdate) values
('Insert 1 into COPY', '2020-01-01'),
('Insert 2 into COPY', '2020-01-02'),
('Insert 3 into COPY', '2020-01-03'),
('Insert 4 into COPY', '2020-01-04'),
('Insert 5 into COPY', '2020-01-05');
```

9.1.10 Select the new rows from LINEITEM_COPY:

```
SELECT l_comment, l_shipdate
FROM lineitem_copy
WHERE l_shipdate > '2019-12-31';
```

9.1.11 Run the same statement against LINEITEM_CLONE, to see the clone has not changed:

```
SELECT l_comment, l_shipdate
FROM lineitem_clone
WHERE l_shipdate > '2019-12-31';
```

9.1.12 Insert some values into the LINEITEM_CLONE table:

```
INSERT INTO lineitem_clone (l_comment, l_shipdate) values
('Insert 6 into CLONE', '2020-01-01'),
('Insert 7 into CLONE', '2020-01-02'),
('Insert 8 into CLONE', '2020-01-03'),
('Insert 9 into CLONE', '2020-01-04'),
('Insert 10 into CLONE', '2020-01-05');
```

9.1.13 Select the new rows from the LINEITEM_CLONE table. Run the same SELECT that you ran against the LINEITEM_COPY table, to see that the rows are not there:

```
SELECT l_comment, l_shipdate
FROM lineitem_clone
WHERE l_shipdate > '2019-12-31';
SELECT l_comment, l_shipdate
FROM lineitem_copy
WHERE l_shipdate > '2019-12-31';
```

9.1.14 Run the same SELECT against the original LINEITEM table, and verify that none of the rows are there:

```
SELECT l_comment, l_shipdate
FROM TRAINING_DB.TRAININGLAB.LINEITEM
WHERE l_shipdate > '2019-12-31';
```

9.1.15 Run a JOIN against the LIEITEM_COPY and LINEITEM_CLONE tables to see the difference in the new rows that were added:

```
SELECT a.l_comment AS comment_copy, a.l_shipdate AS shipdate_copy,
       b.l_comment AS comment_clone, b.l_shipdate AS shipdate_clone
FROM lineitem_copy a
JOIN lineitem_clone b ON a.l_shipdate=b.l_shipdate
WHERE a.l_shipdate > '2019-12-30';
```

9.2 UNDROP a Table

9.2.1 Create a new worksheet named *Time Travel*.

9.2.2 Create the base table to be used in this exercise:

```
CREATE TABLE testdrop (c1 NUMBER);
```

9.2.3 Display the table history and review the values in the `dropped_on` column. Note that all values are NULL:

```
SHOW TABLES HISTORY;
```

9.2.4 DROP the TESTDROP table:

```
DROP TABLE testdrop;
```

9.2.5 Query the TESTDROP table. The table will not be found:

```
SELECT * FROM testdrop;
```

9.2.6 Rerun the table history and review the values in the `dropped_on` column:

```
SHOW TABLES HISTORY;
```

NOTE: The `dropped_on` column is now populated, to indicate the table was dropped.

9.2.7 Use the UNDROP command to recover the TESTDROP table:

```
UNDROP TABLE testdrop;
```

9.2.8 Query the TESTDROP table. It will succeed:

```
SELECT * FROM testdrop;
```

9.3 Recover a Table to a Time Before a Change Was Made

9.3.1 Rerun the table history and review the values in the `dropped_on` column; notice how the values reflect that TESTDROP is no longer dropped:

```
SHOW TABLES HISTORY;
```

9.3.2 Insert the following values into the TESTDROP table:

```
INSERT INTO testdrop VALUES (1000), (2000), (3000), (4000);
```

9.3.3 Query the data in the TESTDROP table to confirm that the four rows have been inserted:

```
SELECT * FROM testdrop;
```

9.3.4 Delete values from the table:

```
DELETE FROM testdrop WHERE c1 IN (2000, 3000);
```

9.3.5 Review the data in the TESTDROP table to confirm that only the values **1000** and **4000** remain:

```
SELECT * FROM testdrop;
```

9.3.6 Navigate to the History tab and locate the query where you deleted the **2000** and **3000** records. Copy this query's ID.

9.3.7 Query the TESTDROP table at a time before the records were deleted:

```
SELECT *
FROM testdrop BEFORE (STATEMENT => '<query ID>');
```

9.3.8 Create a new table that holds the value of TESTDROP before the query that deleted the rows was executed:

```
CREATE TABLE testdrop_restored
CLONE testdrop BEFORE (STATEMENT  => '<query ID>');
```

9.3.9 Review the data in the restored table to confirm that the values **2000** and **3000** have been restored and you have four (4) records:

```
SELECT * FROM testdrop_restored;
```

9.3.10 Execute a JOIN that shows rows that are in TESTDROP_RESTORED, but not in TESTDROP:

```
SELECT * FROM testdrop_restored
   LEFT JOIN testdrop ON testdrop.c1 = testdrop_restored.c1;
```

9.3.11 Drop the TESTDROP Table:

```
DROP TABLE testdrop;
```

9.3.12 Query the Table. Notice the error message about the table not existing:

```
SELECT * FROM testdrop;
```

9.4 Object-Naming Constraints

9.4.1 Create the base table to be used in this exercise:

```
CREATE TABLE loaddata1 (c1 NUMBER);
```

9.4.2 Insert the following values into the LoadData1 table:

```
INSERT INTO LoadData1 VALUES (1111), (2222), (3333), (4444);
```

9.4.3 Query the data in the LoadData1 table to confirm that the four (4) rows have been inserted:

```
SELECT * FROM loaddata1;
```

9.4.4 Drop the LoadData1 Table and confirm you receive an error that the object does not exist when you try to query it:

```
DROP TABLE loaddata1;
SELECT * FROM loaddata1;
```

9.4.5 Create a new iteration of the LoadData1 table with the same structure as the previous iteration, and load values:

```
CREATE TABLE loaddata1 (c1 NUMBER);
INSERT INTO loaddata1
VALUES (777), (888), (999);
```

9.4.6 Drop the LoadData1 table again and create a third iteration of the LoadData1 table with the same structure, but do not insert any values.

```
DROP TABLE loaddata1;
CREATE TABLE loaddata1 (c1 NUMBER);
```

9.4.7 Undrop the LoadData1 table. What happens?

```
UNDROP TABLE loaddata1;
```

You cannot run the UNDROP command if a table exists with the same name as the dropped table.

9.4.8 Rerun the table history and note that there are now multiple entries with the same name, but with different dropped_on values.

You were unable to successfully undrop the table in the step above, because there is currently an active independent iteration of it:

```
SHOW TABLES HISTORY;
```

9.4.9 Rename the current LoadData1 table iteration to LoadData3 and then run the undrop command for LoadData1:

```
ALTER TABLE loaddata1 RENAME TO loaddata3;
UNDROP TABLE loaddata1;
```

9.4.10 Select the data from LoadData1 and note that iteration 2 has been restored:

```
SELECT * FROM loaddata1;
```

9.4.11 Re-run the table history to verify that the second iteration of the table is now active:

```
SHOW TABLES HISTORY;
```

9.4.12 Rename the current LoadData1 table iteration to LoadData2, and undrop loaddata1:

```
ALTER TABLE loaddata1 RENAME TO loaddata2;
UNDROP TABLE loaddata1;
```

9.4.13 Select the data from LoadData. You will see that the first iteration has been restored:

```
SELECT * FROM loaddata1;
```

9.4.14 Re-run the table history to verify that all iterations of the table are now active:

```
SHOW TABLES HISTORY;
```

9.4.15 Query each of the LoadData tables and verify the different iterations; all dropped and restored via Time Travel:

```
SELECT * FROM loaddata1;
SELECT * FROM loaddata2;
SELECT * FROM loaddata3;
```

9.4.16 Rerun the table history to verify that all iterations of the table are now active:

```
SHOW TABLES HISTORY;
```

9.4.17 Suspend and resize the warehouse

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

10 Determine Appropriate Warehouse Sizes

10.1 Run a sample query with an extra small warehouse

In this task you will disable the query result cache, and run the same query on different sized warehouses to determine which is the best size for the query. You will suspend your warehouse after each test, to clear the data cache so the performance of the next query is not artificially enhanced.

Expect this lab to take approximately *15 minutes*.

10.1.1 Navigate to **[Worksheets]**.

10.1.2 Create a new worksheet named *Warehouse Sizing* with the following context:

- ROLE: TRAINING_ROLE
- WAREHOUSE: **[login]**_WH
- DATABASE: SNOWFLAKE_SAMPLE_DATA
- SCHEMA: TPCDS_SF10TCL

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
USE SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL;
```

10.1.3 Change the size of your warehouse to `Xsmall` :

```
ALTER WAREHOUSE [login]_WH
SET WAREHOUSE_SIZE = XSmall;
```

10.1.4 Suspend and resume the warehouse to make sure its Data Cache is empty, disable the query result cache and set a query tag:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH RESUME;
ALTER SESSION SET USE_CACHED_RESULT = FALSE;
ALTER SESSION SET QUERY_TAG = '[login]_WH_Sizing';
```

10.1.5 Run the following query (your test query - it will be used throughout this task) to list detailed catalog sales data together with a running sum of sales price within the order (it will take several minutes to run):

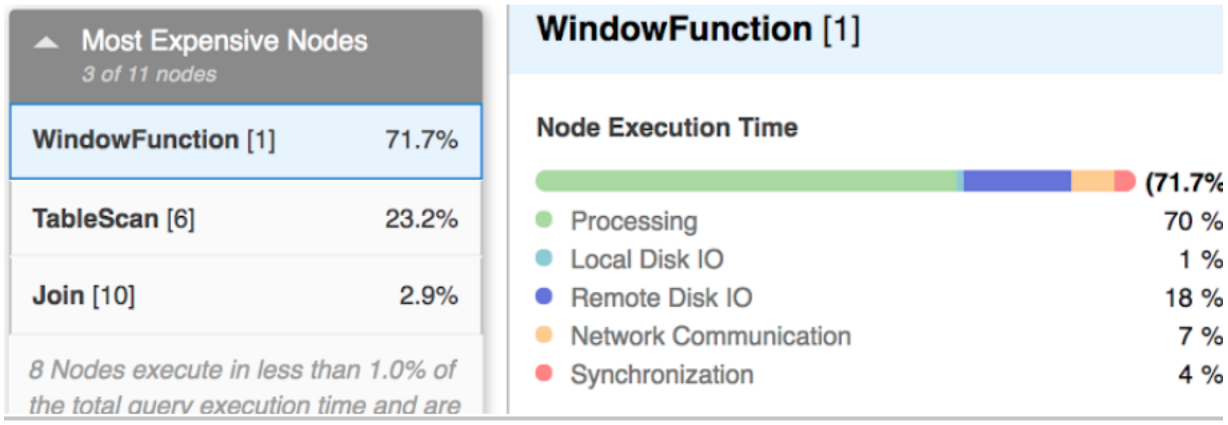
```
SELECT cs_bill_customer_sk, cs_order_number, i_product_name, cs_sales_price,
    SUM(cs_sales_price)
OVER (PARTITION BY cs_order_number
   ORDER BY i_product_name
   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) run_sum
FROM catalog_sales, date_dim, item
WHERE cs_sold_date_sk = d_date_sk
AND cs_item_sk = i_item_sk
AND d_year IN (2000) AND d_moy IN (1,2,3,4,5,6)
LIMIT 100;
```

10.1.6 View the query profile, and click on the operator WindowFunction[1].

10.1.7 Take note of the performance metrics for this operator: you should see significant spilling to local storage, and possibly spilling to remote storage.

- The spill to local storage indicates the warehouse did not have enough memory.
- If it spills to remote storage, the warehouse did not have enough SSD storage to store the spill from memory on its local SSD drive.

10.1.8 Take note of the performance on the small warehouse.


Most Expensive Nodes
3 of 11 nodes

WindowFunction [1]	71.7%
TableScan [6]	23.2%
Join [10]	2.9%

8 Nodes execute in less than 1.0% of the total query execution time and are

WindowFunction [1]

Node Execution Time

(71.7%

● Processing	70 %
● Local Disk IO	1 %
● Remote Disk IO	18 %
● Network Communication	7 %
● Synchronization	4 %

Figure 31: Warehouse Performance

10.2 Run a sample query with a medium warehouse

10.2.1 Suspend your warehouse, change its size to Medium, and resume it:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = Medium;
ALTER WAREHOUSE [login]_WH RESUME;
```

10.2.2 Re-run the test query:

```
SELECT cs_bill_customer_sk, cs_order_number, i_product_name, cs_sales_price,
   SUM(cs_sales_price)
OVER (PARTITION BY cs_order_number
   ORDER BY i_product_name
   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) run_sum
FROM catalog_sales, date_dim, item
WHERE cs_sold_date_sk = d_date_sk
AND cs_item_sk = i_item_sk
AND d_year IN (2000) AND d_moy IN (1,2,3,4,5,6)
LIMIT 100;
```

10.2.3 View the query profile, and click the operator WindowFunction[1].

10.2.4 Take note of the performance metrics for this operator. You should see lower amounts spilling to local storage and remote disk, as well as faster execution.

10.3 Run a sample query with a large warehouse

10.3.1 Suspend your warehouse, change its size to Large, and resume it:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = Large;
ALTER WAREHOUSE [login]_WH RESUME;
```

10.3.2 Re-run the test query:

```
SELECT cs_bill_customer_sk, cs_order_number, i_product_name, cs_sales_price,
    SUM(cs_sales_price)
OVER (PARTITION BY cs_order_number
   ORDER BY i_product_name
   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) run_sum
FROM catalog_sales, date_dim, item
WHERE cs_sold_date_sk = d_date_sk
AND cs_item_sk = i_item_sk
AND d_year IN (2000) AND d_moy IN (1,2,3,4,5,6)
LIMIT 100;
```

10.3.3 Take note of the performance metrics for this operator. You will see that there is no spilling to local or remote storage.

10.4 Run a sample query with an extra large warehouse

10.4.1 Suspend the warehouse, change it to XLarge in size, and resume it:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = XLarge;
ALTER WAREHOUSE [login]_WH RESUME;
```

10.4.2 Re-run the test query.

```
SELECT cs_bill_customer_sk, cs_order_number, i_product_name, cs_sales_price,
    SUM(cs_sales_price)
OVER (PARTITION BY cs_order_number
   ORDER BY i_product_name
   ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) run_sum
FROM catalog_sales, date_dim, item
WHERE cs_sold_date_sk = d_date_sk
```

```
AND cs_item_sk = i_item_sk
AND d_year IN (2000) AND d_moy IN (1,2,3,4,5,6)
LIMIT 100;
```

10.4.3 Note the performance.

10.4.4 Suspend your warehouse, and change its size to XSmall:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = XSmall;
```

10.5 Run a query showing the query history results for these performance tests using the INFORMATION_SCHMEA

```
SELECT query_id,query_text, warehouse_size,(execution_time / 1000) Time_in_seconds
    FROM TABLE(information_schema.query_history_by_session())
WHERE  query_tag = '[login]_WH_Sizing'
AND WAREHOUSE_SIZE IS NOT NULL
AND QUERY_TYPE LIKE 'SELECT' ORDER BY start_time DESC;
```

10.6 This query will show more information but the ACCOUNT_USAGE schema has a 45 minute latency on the QUERY_HISTORY view.

```
SELECT query_id,query_text, warehouse_size,(execution_time / 1000)
   Time_in_seconds, partitions_total, partitions_scanned,
        bytes_spilled_to_local_storage, bytes_spilled_to_remote_storage,
           query_load_percent
FROM "SNOWFLAKE"."ACCOUNT_USAGE"."QUERY_HISTORY"
WHERE  query_tag = '[login]_WH_Sizing'
AND WAREHOUSE_SIZE IS NOT NULL
AND QUERY_TYPE LIKE 'SELECT' ORDER BY start_time DESC;
```

- After running these tests, what size warehouse do you think is best for this query? Why?

10.6.1 Suspend and resize the warehouse

```
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE=XSmall;
ALTER WAREHOUSE [login]_WH SUSPEND;
```

11 Monitor Billing & Usage Information

Expect this lab to take approximately *40 minutes*.

Lab Purpose: Practice monitoring resource usage with the Snowflake UI and SQL.

NOTE: Some of the commands you will run in these exercises require ACCOUNTADMIN permission. The role TRAINING_ROLE has been granted these permissions so that you can run these exercises. In your own production environment, you may not get any results if you do not have ACCOUNTADMIN access.

11.1 Review Warehouse Usage

11.1.1 Navigate to **[Warehouses]**.

11.1.2 Select **[login]**_WH.

11.1.3 Review the *Warehouse Load Over Time* information.

- Take note of the hourly fluctuations in usage in the top graph.
- Take note of the daily fluctuations in usage in the bottom graph.
- Change the time frame of the bottom graph to span various time frames and notice the impact this change has on the top graph.

11.1.4 Navigate to **[Account]**.

11.1.5 Make sure the Billing & Usage section is selected. Review the information about Snowflake credits billed for the given time period.

- Total number of Warehouses / Total credits billed
- Credits billed by Warehouse
- Credits billed by Day

11.1.6 Open a new worksheet named *Monitoring* and set the context:

```
USE ROLE TRAINING_ROLE;
CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
USE WAREHOUSE [login]_WH;
CREATE DATABASE IF NOT EXISTS [login]_DB;
USE [login]_DB.PUBLIC;
```

11.1.7 Use SQL and the Information Schema WAREHOUSE_METERING_HISTORY table function to pull Warehouse credit usage, by warehouse, for the last 5 days:

```
SELECT *
FROM TABLE(INFORMATION_SCHEMA.WAREHOUSE_METERING_HISTORY
   (DATEADD('DAYS', -5, CURRENT_DATE())));
```

11.1.8 Starting with the query above, write your own query to look at the total credits consumed by warehouse. Show the warehouses with the greatest total credits first:

```
SELECT warehouse_name, SUM(credits_used) FROM
TABLE(INFORMATION_SCHEMA.WAREHOUSE_METERING_HISTORY
(DATEADD('DAYS', -5, CURRENT_DATE())))
GROUP BY warehouse_name
ORDER BY SUM(credits_used) DESC;
```

11.2 Review Billing & Usage Information

11.2.1 Set your role to TRAINING_ROLE (if not already set).

11.2.2 Navigate to **[Account]**.

11.2.3 Select the Billing & Usage Section (if not already selected).

11.2.4 Click the Warehouses box (if not already selected).

11.2.5 Review the number of Warehouses and total credits billed and take note of which month the activity is listed.

In the detailed activity, identify:

- Which Warehouse consumed the most credits?
- Which day of the month consumed the most credits?

11.2.6 Navigate to **[Warehouses]**.

11.2.7 Locate and click on your **[login]**_WH and evaluate your usage over time.

11.2.8 Expand the time frame of the bottom graph and see what happens to the top graph.

11.2.9 Navigate back to **[Account]**.

11.2.10 Select the Billing & Usage Section (if not already selected).

11.2.11 Click the Average Storage Used box.

11.2.12 Review the Daily Average and the Rolling Monthly Average Line on the graph.

NOTE: This is for a given Month and the storage can be broken out Total (Default), Database (Active + Time Travel), Stage (Internal), and Fail-Safe.

11.2.13 Toggle between Total, Database, Stage, and Fail-Safe.

11.2.14 Navigate to **[Worksheets]**.

11.2.15 Execute the following commands to pull various Billing & Usage metrics:

11.2.16 Pull all Warehouse usage for the last rolling seven (7) days from the Information Schema:

```
SELECT * FROM TABLE(INFORMATION_SCHEMA.WAREHOUSE_METERING_HISTORY
   (DATEADD('DAYS', -7, CURRENT_DATE())));
```

11.3 View and Set Parameters

11.3.1 In the Worksheets tab, show all parameters for your session:

```
SHOW PARAMETERS IN SESSION;
```

11.3.2 Pull the value for the `DATE_OUTPUT_FORMAT` parameter for your session. The format set should equal the default format of 'YYYY-MM-DD':

```
SHOW PARAMETERS LIKE 'DATE_OUTPUT_FORMAT' IN SESSION;
```

11.3.3 Run a query to SELECT the current date, and notice the format of the date returned:

```
SELECT CURRENT_DATE();
```

11.3.4 Set the `DATE_OUTPUT_FORMAT` parameter to 'DD MON YYYY':

```
ALTER SESSION SET DATE_OUTPUT_FORMAT = 'DD MON YYYY';
```

11.3.5 Re-run the query to SELECT the current date. Notice the format of the date returned has changed to match that of the value you set for the DATE_OUTPUT_FORMAT–'DD MON YYYY'.

11.3.6 Open a new Worksheet and run the query to SELECT the current date.

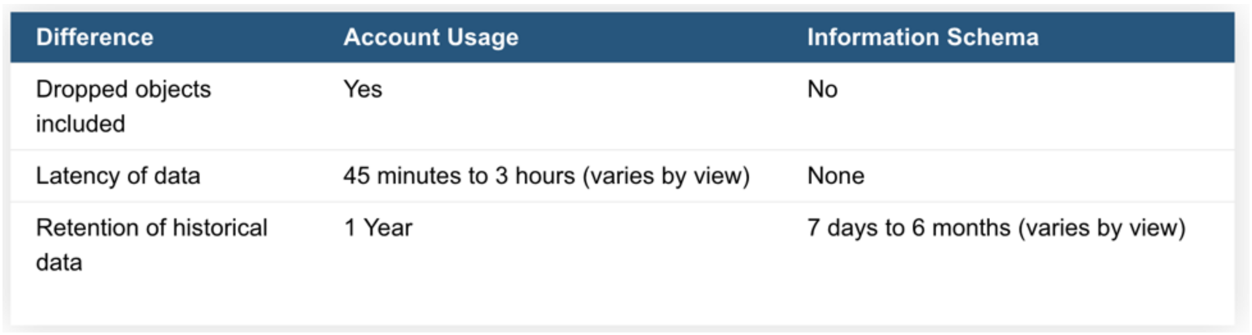
Notice that the format of the date returned matches the Snowflake default of 'YYYY-MM-DD' rather than the format you set for the DATE_OUTPUT_FORMAT–'DD MON YYYY'. Why is this?

```
SELECT CURRENT_DATE();
```

HINT: Worksheets represent independent Snowflake sessions.

11.4 Evaluate Account and Object Information

NOTE: The use of Information Schema & Account Usage is very similar, sometimes identical. When to use which functionality will depend upon your needs regarding latency, data retention, and dropped objects.



Difference	Account Usage	Information Schema
Dropped objects included	Yes	No
Latency of data	45 minutes to 3 hours (varies by view)	None
Retention of historical data	1 Year	7 days to 6 months (varies by view)

Figure 32: Information Schema & Account Usage

11.5 Monitor Storage

NOTE: In this lab, you will be exploring some menus that are normally accessible only to the ACCOUNTADMIN role. These privileges have been granted to TRAINING_ROLE for convenience in this exercise.

11.5.1 Select **[Account]**.

11.5.2 Navigate to the Billing & Usage section.

11.5.3 Click the Average Storage Used option.

11.5.4 Toggle between the Database, Stage, and Fail Safe categories and review the change in storage for each throughout the month.

11.5.5 Select the Worksheets tab. You may use the same context from the previous task.

11.5.6 Run a query to return average daily storage usage for the past 5 days, per database, for all databases in your account:

```sql
SELECT *
FROM TABLE (information_schema.database_storage_usage_history
   (DATEADD('days', -5, CURRENT_DATE()), CURRENT_DATE()));
```

11.5.7 Modify the query to return average daily storage usage for the past 5 days for your account overall:

```
SELECT
    usage_date,
    SUM(average_database_bytes) average_database_bytes,
    SUM(average_failsafe_bytes) average_failsafe_bytes
FROM TABLE(information_schema.database_storage_usage_history
(dateadd('days',-5,current_date()),current_date()))
GROUP BY usage_date
ORDER BY usage_date;
```

11.5.8 Run a query to return average daily data storage usage for *all* the Snowflake stages in your account for the past 5 days:

```
SELECT *
FROM TABLE (information_schema.stage_storage_usage_history(DATEADD
(DAYS,-5,CURRENT_DATE()),CURRENT_DATE()));
```

11.5.9 Pull total storage bytes for the last rolling seven (7) days from Account Usage.

```
SELECT *
FROM SNOWFLAKE.ACCOUNT_USAGE.STORAGE_USAGE
WHERE usage_date >= DATEADD('DAYS',-7,CURRENT_DATE())
ORDER BY usage_date;
```

11.5.10 Pull storage for Internal Stages for the previous 5 days. In the command below, you need to replace **YYYY-MM-DD** in the command with the four-digit year, two-digit month, and two-digit you wish to examine.

NOTE: The storage is total and is not broken out by stage.

```
SELECT *
FROM TABLE(INFORMATION_SCHEMA.STAGE_STORAGE_USAGE_HISTORY
(DATE_RANGE_START=>'YYYY-MM-DD'));
```

11.6 Query Views in Information Schema

11.6.1 Open a new Worksheet. Re-name the worksheet to *Information Schema*.

11.6.2 Retrieve the hourly Warehouse usage for the current month for your **[login]**_WH Virtual Warehouse, ordered by time:

```
SELECT *
FROM TABLE(information_schema.warehouse_metering_history
(date_range_start=>date_trunc(month, current_date),
date_range_end=>dateadd(month,1,date_trunc(month, current_date)),'[login]_WH'))
ORDER BY start_time;
```

11.6.3 Retrieve average daily storage for the past three (3) days, for **[login]**_DB Database:

```
SELECT *
FROM TABLE(information_schema.database_storage_usage_history
(dateadd('days',-3,current_date()),current_date(),
'[login]_db'));
```

11.6.4 Directly query the Information Schema TABLES View to find the size of each Table in the WEATHER Schema in the TRAINING_DB Database. Sort by largest table:

```
SELECT *
FROM SNOWFLAKE_SAMPLE_DATA.INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'WEATHER'
ORDER BY bytes DESC;
```

11.6.5 Join the APPLICABLE_ROLES and DATABASES Information Schema Views to find out which databases are available to your user and through which roles:

```
SELECT distinct
    r.grantee,
    r.role_name,
    r.role_owner,
    d.database_name,
    d.database_owner
FROM information_schema.applicable_roles r
JOIN information_schema.databases d ON
    d.database_owner=r.role_name
WHERE r.grantee = '[login]';
```

11.7 Query Views in Account Usage

11.7.1 Retrieve the number of failed logins, by user, month-to-date from the LOGIN_HISTORY View. Order the results by the highest login failure rate:

```
SELECT user_name,
SUM(IFF(is_success = 'NO', 1, 0)) AS failed_logins,
COUNT(*) AS logins,
SUM(IFF(is_success = 'NO', 1,0)) / nullif(count(*), 0)
AS login_failure_rate
FROM SNOWFLAKE.ACCOUNT_USAGE.login_history
WHERE event_timestamp > date_trunc(month, current_date)
GROUP BY 1
ORDER BY 4 DESC;
```

11.7.2 Determine the busiest Warehouse by number of jobs executed, by querying the QUERY_HISTORY View:

```
SELECT warehouse_name, COUNT(*) AS number_of_jobs
FROM SNOWFLAKE.ACCOUNT_USAGE.query_history
WHERE start_time >= date_trunc(month, current_date)
  AND warehouse_name IS NOT NULL
GROUP BY 1 ORDER BY 2 DESC;
```

11.7.3 Determine the number of used credits and costs by all Warehouses within an account in the last three (3) days by querying the WAREHOUSE_METERING_HISTORY View. Assume each credit costs $2.50:

```
SET compute_price=2.50;

SELECT date_trunc(day, start_time) AS usage_day,
SUM(coalesce(credits_used, 0.00)) AS total_credits,
SUM($compute_price * coalesce(credits_used, 0.00))
  AS billable_warehouse_usage
FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
WHERE start_time >= date_trunc(day, dateadd(day, -3,current_timestamp))
  AND start_time < date_trunc(day, current_timestamp)
GROUP BY 1;
```

11.7.4 Determine the amount of storage and associated costs used within an account in the last three (3) days. Assume storage costs $23/TB/month.

```
SET storage_price=23;

SELECT date_trunc(day, usage_date) AS usage_day,
ROUND(AVG(storage_bytes)/power(1024, 4), 3)
  AS billable_database_tb, ROUND(AVG(failsafe_bytes)/power(1024, 4), 3)
  AS billable_failsafe_tb, ROUND(AVG(stage_bytes)/power(1024, 4), 3)
  AS billable_stage_tb, $storage_price *
  (billable_database_tb + billable_failsafe_tb + billable_stage_tb)
  AS total_billable_storage_usd
FROM SNOWFLAKE.ACCOUNT_USAGE.STORAGE_USAGE
WHERE usage_date >= date_trunc(day, dateadd(day, -3, current_timestamp))
AND usage_date < date_trunc(day, current_timestamp)
GROUP BY 1;
```

11.7.5 Show the 10 longest queries by using the QUERY_HISTORY View:

```
SELECT query_text, user_name, role_name,
   database_name, warehouse_name, warehouse_size, execution_status,
   ROUND(total_elapsed_time/1000,3) elapsed_sec
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
ORDER BY total_elapsed_time DESC LIMIT 10;
```

11.7.6 Show the top 10 users in terms of total execution time within the Account for the last 5 days by using the QUERY_HISTORY View:

```
SELECT user_name, ROUND(SUM(total_elapsed_time)/1000/60/60,3) elapsed_hrs
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE START_TIME >= DATE_TRUNC(DAY, DATEADD(DAY, -5, CURRENT_TIMESTAMP))
  AND START_TIME < DATE_TRUNC(DAY, CURRENT_TIMESTAMP)
GROUP BY 1 ORDER BY 2 DESC
LIMIT 10;
```

11.7.7 Suspend your warehouse:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
```

12 APPENDIX

12.1 Commonly Used Commands

The following commands are used frequently throughout the course. Because of that, they are listed here, rather than repeating them individually for each step that uses them.

Instruction	Command
Set your standard context to be your default	USE ROLE SECURITYADMIN;USE DATABASE **[login]**_DB;USE SCHEMA PUBLIC;USE WAREHOUSE **[login]**_WH;
Query the table (with or without a LIMIT)	SELECT [**[column_list]** | *]FROM **[table]** LIMIT **[num_rows]**;
Truncate the table	TRUNCATE **[table]**;
Change the size of your warehouse	ALTER WAREHOUSE **[name]** SET WAREHOUSE_SIZE=**[size]**;
Suspend your warehouse	ALTER WAREHOUSE **[name]** SUSPEND;
Resume your warehouse	ALTER WAREHOUSE **[name]** RESUME;
Turn off query result caching	ALTER SESSION SET USE_CACHED_RESULT=FALSE;
Drop a table/database/schema	DROP **[object_type]** **[object_name]**;

12.2 File Paths With PUT and GET in Windows

Windows file paths changes format if there is a space in the path. It is easier to create a direct path without spaces rather than try and use a path with spaces.

Example of a path without spaces. Notice this uses a backslash in the path.

```
PUT PUT file://Z:\SnowFlake\region.tbl @loadfiles;
```

Example of a path with spaces. The path must be quoted and it uses a forward slash.

```
PUT 'file://Z:/SnowFlake/Fund Training/*.json' @Loadfiles;
```