



PRODUCTS SERVICES PRICING

# YOUR CLOUD DATA PLATFORM

Secure and easy access to any data with infinite scalability

## Snowflake Fundamentals Workbook

© 2022 Snowflake Computing Inc. All Rights Reserved

22E17

## Contents

<b>1</b>	<b>Introduction to Snowflake and Snowflake Objects</b>	<b>5</b>
1.1	Launching Snowsight . . . . .	6
1.2	Creating Snowflake Objects . . . . .	11
1.3	Creating Objects Exclusively with SQL Statements . . . . .	14
1.4	Key Takeaways . . . . .	17
<b>2</b>	<b>Visualizations in Snowsight</b>	<b>18</b>
2.1	Creating a Dashboard . . . . .	18
2.2	How to use Auto-Complete to write a query . . . . .	19
2.3	How to use ad-hoc filters to get insights into data . . . . .	22
2.4	How to add tiles to a dashboard . . . . .	27
2.5	How to share a dashboard . . . . .	30
2.6	Key Takeaways . . . . .	32
<b>3</b>	<b>Querying Data with Time Travel</b>	<b>33</b>
3.1	Conducting a What-If Scenario . . . . .	33
3.2	Key Takeaways . . . . .	37
<b>4</b>	<b>Multi-Table Inserts</b>	<b>38</b>
4.1	Working with Sequences . . . . .	38
4.2	Working with Unconditional Multi-Table Inserts . . . . .	40
4.3	Working with Conditional Multi-Table Inserts . . . . .	44
4.4	Using ALTER TABLE SWAP WITH to swap table content and metadata . . . . .	46
4.5	Using MERGE to update rows in a table . . . . .	48
4.6	Key Takeaways . . . . .	53
<b>5</b>	<b>Caching and Query Performance</b>	<b>55</b>
5.1	Accessing and Navigating the Query Profile . . . . .	55
5.2	Metadata Cache . . . . .	65
5.3	Data Warehouse Cache . . . . .	66
5.4	Partition Pruning . . . . .	68
5.5	Determine If Spillage Is Taking Place . . . . .	69
5.6	Review the EXPLAIN Plan . . . . .	72
5.7	Summary . . . . .	74
5.8	Key takeaways . . . . .	74
<b>6</b>	<b>Loading Structured Data</b>	<b>76</b>
6.1	Loading Data from an External Stage into a Table Using COPY INTO . . . . .	76
6.2	Load the <i>region.tbl</i> File . . . . .	77
6.3	Loading a GZip Compressed File on an External Stage into a Table . . . . .	78
6.4	Validating data prior to load . . . . .	79

6.5	Error Handling . . . . .	80
6.6	Key Takeaways . . . . .	83
<b>7</b>	<b>Data Transformation During Data Loading</b>	<b>84</b>
7.1	Transforming Data During Load . . . . .	84
7.2	Key Takeaways . . . . .	87
<b>8</b>	<b>Unloading Structured Data</b>	<b>88</b>
8.1	Unloading table data into a Table Stage in Pipe-Delimited File format . . . . .	88
8.2	Use a SQL statement containing a JOIN to Unload a Table into an internal stage . . . . .	89
8.3	Key Takeaways . . . . .	90
<b>9</b>	<b>Introduction to Tasks</b>	<b>91</b>
9.1	Creating the target table . . . . .	91
9.2	Creating the tasks . . . . .	92
9.3	Ending the lab . . . . .	93
9.4	Key Takeaways . . . . .	94
<b>10</b>	<b>Snowflake Functions</b>	<b>95</b>
10.1	Scalar Functions . . . . .	95
10.2	Use Regular and Windows Aggregate Functions . . . . .	98
10.3	TABLE and System Functions . . . . .	98
10.4	Key Takeaways . . . . .	100
<b>11</b>	<b>User-Defined Functions and Stored Procedures</b>	<b>101</b>
11.1	Create a JavaScript User-Defined Function . . . . .	101
11.2	Create a SQL User-defined Function . . . . .	102
11.3	Creating Stored Procedures . . . . .	102
11.4	Key Takeaways . . . . .	104
<b>12</b>	<b>Using High-Performing Functions</b>	<b>105</b>
12.1	Working with HyperLogLog . . . . .	106
12.2	Use Percentile Estimation Functions . . . . .	107
12.3	Key takeaways . . . . .	108
<b>13</b>	<b>Access Control and User Management</b>	<b>109</b>
13.1	Determine Privileges (GRANTS) . . . . .	109
13.2	Granting Permissions (GRANT ROLE and GRANT USAGE) . . . . .	110
13.3	Key Takeaways . . . . .	111
<b>14</b>	<b>Secondary Roles</b>	<b>112</b>
14.1	Determine Privileges (GRANTS) . . . . .	112
14.2	Granting Permissions (GRANT ROLE and GRANT USAGE) . . . . .	113
14.3	Key Takeaways . . . . .	115

**15 Exploring Semi-Structured JSON Data**

15.1 Working with semi-structured data . . . . .

15.2 Using FLATTEN . . . . .

15.3 Working with Weather Data . . . . .

15.4 Key takeaways . . . . .

**16 Determine Appropriate Warehouse Sizes**

16.1 Run a sample query with an extra small warehouse . . . . .

16.2 Run a sample query with a small warehouse . . . . .

16.3 Run a sample query with a medium warehouse . . . . .

16.4 Run a sample query with a large warehouse . . . . .

16.5 Run a sample query with an extra large warehouse . . . . .

16.6 Key Takeaways . . . . .

**17 Introduction to Monitoring Usage and Billing**

17.1 Snowflake Database . . . . .

17.2 Monitoring Usage and Billing with the ACCOUNT\_USAGE schema . . . . .

17.3 Billing Metrics . . . . .

17.4 Key Takeaways . . . . .

**116**

116

118

123

129

**130**

130

132

133

133

134

136

**137**

137

138

141

143

## 1 Introduction to Snowflake and Snowflake Objects

The purpose of this lab is to familiarize you with Snowflake's Snowsight user interface. Specifically, you will learn how to create and use Snowflake objects that you will need to use to run queries and conduct data analysis in your day-to-day work.

If you're a data engineer, you'll learn skills important to your role. If you're not a data engineer, the process of creating the objects will both help you learn how to navigate the Snowsight interface and become familiar with warehouses, databases, roles and schemas, all of which together form the context for any SQL statements you are likely to execute. **NOTE:** Context refers to the resources and objects that must be specified in order for SQL statements to execute.

### Learning Objectives:

- How to navigate Snowsight to find the tools you'll need
- How to create and manage folders and worksheets
- How to set the context via the Snowsight UI or with SQL code
- How to create warehouses, databases, schemas and tables
- How to run a simple query

### Scenario:

You are a data engineer working for Snowbear Air, which is an airline that flies to fun destinations all over the world. You've been tasked to design and implement data sets that will be used by business analysts that create flight profitability reports for executive management. You have been asked to create a few Snowflake objects in a development environment to test out your SQL statements. You will need to create:

- A database
- A schema
- A warehouse
- A table you will then populate with the regions and countries that Snowbear Air serves

### HOW TO COMPLETE THIS LAB

In order to complete the first part of this lab, you will type the SQL commands directly into a worksheet that you create. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

To complete the second half of this lab, you will take the first .SQL file of the set of .SQL files we provided to you and use it to create a new worksheet. At that point you can simply run the code we provide. We'll provide instructions along the way.

Let's get started!

## 1.1 Launching Snowsight

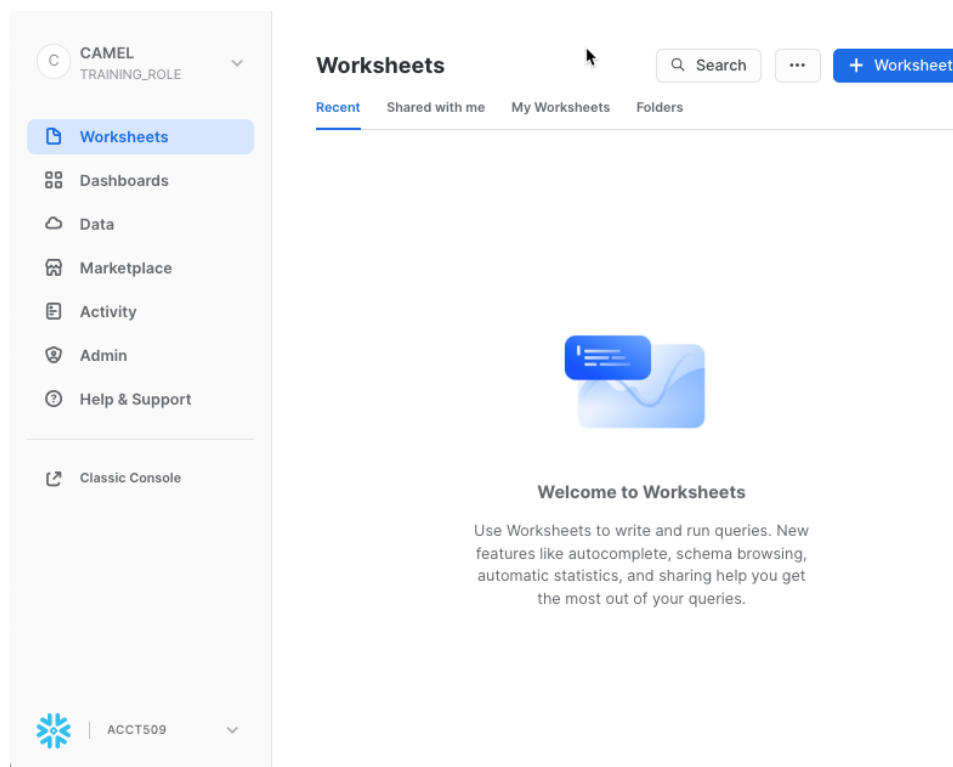
1.1.1 Access the URL provided to you for this course.

1.1.2 You will be taken to a login page. Enter the username and password provided to you for this course.

1.1.3 You will be prompted to change the password. Follow the prompts to change the password and click Submit.

1.1.4 Log in with your new password. This will take you to the Classic WebUI. Next, click the button on the top right of the screen to access Snowsight.

Your screen should look similar to the screen below:



**Figure 1:** Home

Now let's get familiar with the left-hand navigation bar and its contents.

1.1.5 Click on Dashboards in the left-hand navigation bar.

You should see the blank screen below. If you had access to any dashboards, they would appear in the Dashboards pane that takes up the majority of the screen.

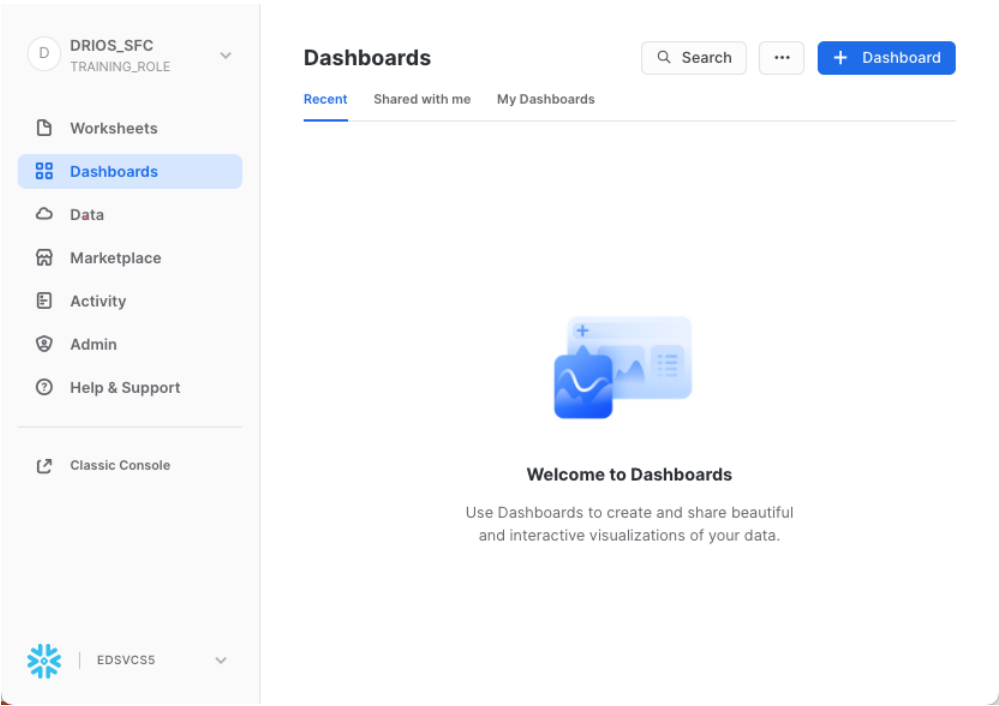


Figure 2: Dashboards

1.1.6 Click on Data in the left-hand navigation bar.

You should see the screen below. An Object Selection Pane and an Object Detail Pane should now be visible.

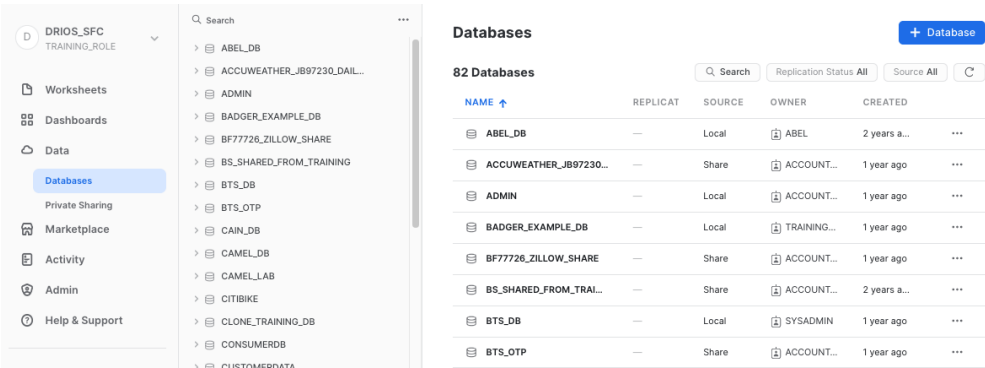


Figure 3: Object Panes

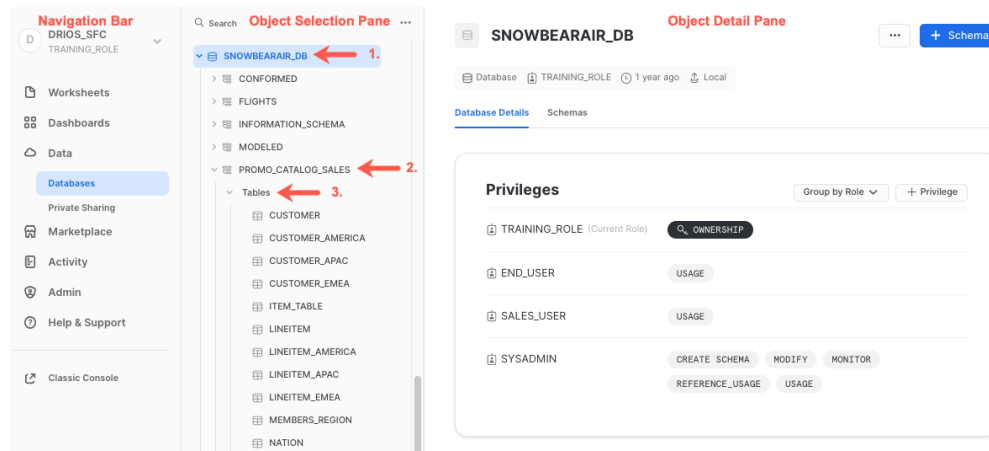
Now let's navigate to a table.

1.1.7 Click SNOWBEARIR\_DB in the Object Selection Pane.

1.1.8 Click schema PROMO\_CATALOG\_SALES.

1.1.9 Click Tables to expand the table tree.

1.1.10 Click any table to view details about the table.

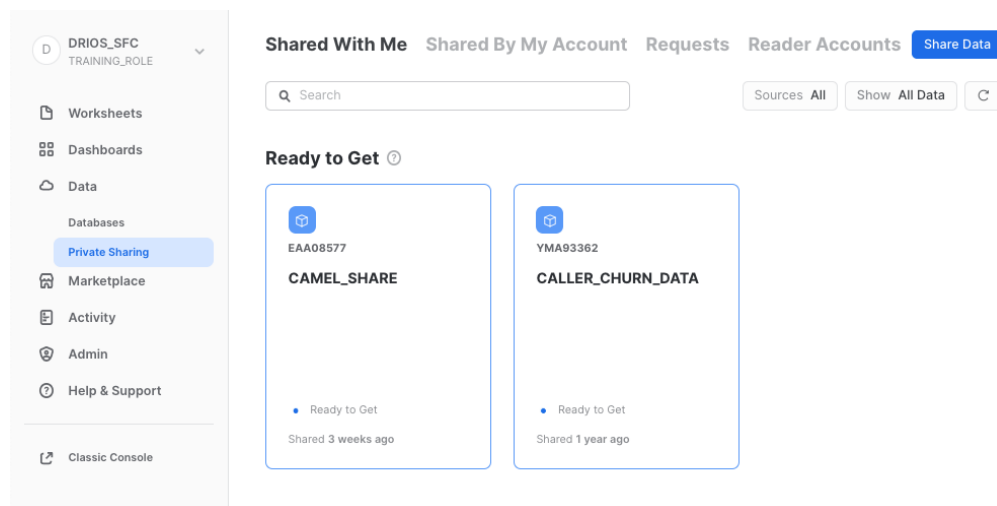


**Figure 4:** Navigating to a Table

By navigating the tree in the Object Selection pane, you can view details about many Snowflake Objects. Try to click through a few more to get familiar with the tree.

1.1.11 Now click on Private Sharing in the left-hand navigation bar.

You should now see a few data sets that are available for you to consume.

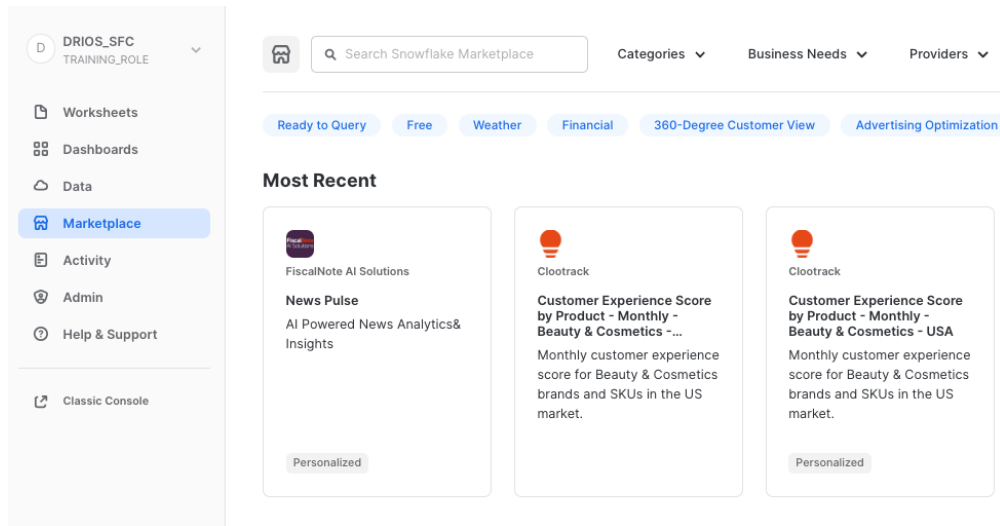


**Figure 5:** Data Sharing



1.1.12 Now click on Marketplace in the left-hand navigation bar.

You should now see a few data sets that are available for you to consume.



**Figure 6:** Marketplace

Scroll through this section and take a look at the offerings. You may see sections titled Featured Providers, Most Recent, Financial, Business, Marketing, Local and Demographics. What you see may differ as the Data Marketplace is dynamic and new types of data sets are being added every day.

As you can imagine, both the Shared Data tab and the Marketplace tab are likely to be useful to many Snowflake users in their day-to-day work.

1.1.13 Now click on Activity in the left-hand navigation bar.

The Query History sub-option under Activity should be selected by default. Your query history will be empty, but after running queries your screen will eventually look similar to the one shown below:

**Query History**

250+ Queries

Status: All User: DRIOS\_SFC Filters Columns

SQL TEXT	QUERY ID	STATUS	USER
SELECT C_CUSTOMER_SK , C_LAST_NAME , (CA_S...	01a37332-0503-edc1-00...	Success	DRIOS_SFC
EXPLAIN SELECT C_CUSTOMER_SK , C_LAST_NAME...	01a37332-0503-ef06-00...	Success	DRIOS_SFC
SELECT C_CUSTOMER_SK , C_LAST_NAME , (CA_S...	01a37330-0503-ef06-00...	Success	DRIOS_SFC
ALTER WAREHOUSE CAMEL_WH RESUME;	01a37330-0503-ef06-00...	Success	DRIOS_SFC
ALTER WAREHOUSE CAMEL_WH SUSPEND;	01a37330-0503-edc1-00...	Success	DRIOS_SFC
ALTER WAREHOUSE CAMEL_WH RESUME;	01a37330-0503-edc1-00...	Success	DRIOS_SFC
ALTER WAREHOUSE CAMEL_WH SUSPEND;	01a37330-0503-edc1-00...	Failed	DRIOS_SFC
ALTER SESSION SET USE_CACHED_RESULT=FALSE;	01a37330-0503-ef06-00...	Success	DRIOS_SFC
ALTER WAREHOUSE CAMEL_WH SET WAREHOUSE_SIZ...	01a37330-0503-ef06-00...	Success	DRIOS_SFC
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCDS_SF1...	01a37330-0503-edc1-00...	Success	DRIOS_SFC

**Figure 7:** Activity

There will be a list of SQL statements with various columns describing query history. You can click the Columns button above the list to select which columns to display.

1.1.14 Now click on Admin→Warehouses in the left-hand navigation bar.

You may see a list of virtual warehouses and their statuses. However, as this is a training environment, there may not be any virtual warehouses yet. Just know that this is where you can go to see what virtual warehouses exist.

Remember, unlike on-premises data warehouses you may be used to, in Snowflake storage and compute are separated. A Snowflake virtual warehouse is a cluster of servers used to run and execute queries, and it provides compute power, memory, and some local SSD storage for caching operations. Other than that, no data is stored in the warehouse. Instead, data is stored in Snowflake's cloud storage layer. Storage and compute are dynamically combined at runtime to execute your queries.

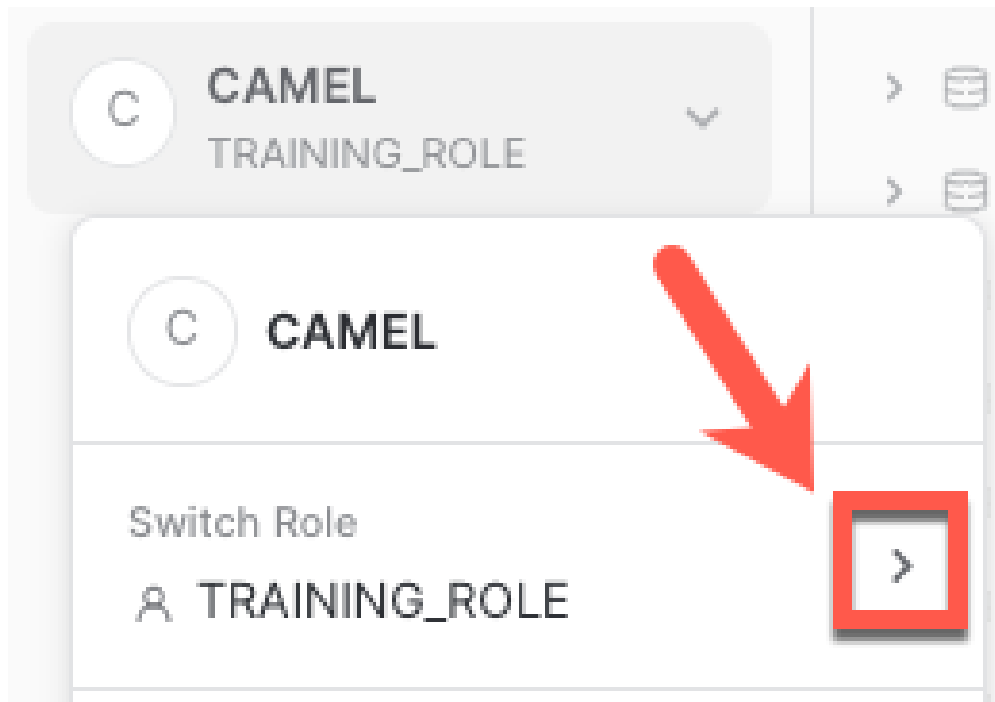
NAME	STATUS	SIZE	CLUSTER	RUNNING	QUEUED	OWNER	CREATED
ANIMAL_TASK_...	Suspe...	Me...	1 - 1	0	0	TRAININ...	1 year a...
BS_WH	Suspe...	X-L...	1 - 1	0	0	ACCOU...	2 years ...
CAMEL_LOAD_...	Suspe...	X-S...	1 - 1	0	0	TRAININ...	1 year a...
CAMEL_QUERY...	Suspe...	X-S...	1 - 1	0	0	TRAININ...	1 year a...
CAMEL_SHARE...	Suspe...	X-S...	1 - 1	0	0	SYSAD...	1 week ...
CAMEL_WH	Suspe...	X-S...	1 - 1	0	0	TRAININ...	1 week ...
CARS_WH	Suspe...	X-S...	1 - 1	0	0	TRAININ...	2 mont...
COMPUTE_WH	Suspe...	Small	1 - 1	0	0	SYSAD...	1 year a...
DATA_SCIENCE...	Suspe...	X-S...	1 - 1	0	0	SYSAD...	1 year a...
DAVID_WH	Suspe...	X-S...	1 - 2	0	0	TRAININ...	1 year a...
DEMO_WH	Suspe...	X-S...	1 - 1	0	0	ACCOU...	8 mont...

**Figure 8:** Warehouses

## 1.2 Creating Snowflake Objects

Now let's get started on our Snowflake objects. We will need a database, a schema, a warehouse and a table. Let's make sure you create the objects in the role you will be using throughout the course, which is TRAINING\_ROLE. This will ensure that your role will own the objects, which will enable you to do whatever you need to in each lab.

- 1.2.1 If your role is not already TRAINING\_ROLE, click the down arrow next to your role. There should now be a pop up menu that says Switch Role. Select the arrow next to your role and select TRAINING\_ROLE.



**Figure 9:** Changing the role

- 1.2.2 Now let's create a database. Click Data in the left-hand navigation bar, then Databases.

### Click the New Database button (It's a big blue button with "+ Database") in the Object Details pane. The New Database dialog box will appear.

***Regarding the login convention***

Throughout the workbooks you will see object names prefixed with the term login enclosed in square brackets. You will NOT use this prefix in your object names. Instead, it is a place holder for the animal name provided to you by the instructor. So, if your animal name is "elephant" and you are asked to create a database, you will replace the prefix with your animal name. Thus, your database will be elephant\_db.

- 1.2.3 Name your database [login]\_db and click the Create button.

The details of your new database should be shown in the Object Details pane.

1.2.4 Select your new database in the Object Selection pane.

1.2.5 Click the Schemas tab in the Object Detail pane to view the schemas INFORMATION\_SCHEMA and PUBLIC.

1.2.6 Next click the new Schema button to create your new schema:

1.2.7 In the New Schema dialog box, name your schema [login]\_schema and click the Create button.

Your schema should now be listed along with schemas INFORMATION\_SCHEMA and PUBLIC.

We haven't created our table yet, but we'll come back to create that after we've created our warehouse.

1.2.8 To create your warehouse, click Admin -> Warehouses in the navigation bar.

1.2.9 Now click the + Warehouse button to create a new Warehouse.

1.2.10 In the New Warehouse dialog box, name your warehouse [login]\_WH.

1.2.11 Choose X-Small for the size.

1.2.12 Expand the Advanced Warehouse Options to confirm Auto Resume and Auto Suspend are selected.

1.2.13 Click the Create Warehouse button.

Your warehouse should now be listed and started.

1.2.14 Create a folder.

We will first create a folder. Then we will create a worksheet in that folder and run the appropriate SQL statements within the worksheet.

1.2.15 Click on Worksheets in the navigation bar.

1.2.16 Click the ellipsis (...) next to the New Worksheet button in the upper right hand corner of the screen.

1.2.17 Click New Folder.

1.2.18 In the New Folder Dialog box, type WORKING WITH OBJECTS and then click the Create Folder Button.

The folder should now be created and its contents (empty of course) shown in the right hand pane. Notice that at the top-left of this pane is a link titled "Worksheets". This is a bread crumb trail that you can use to go up a level, but you don't need to click it now.

Note that at the right of the folder name is a down arrow. If you click it you will see an editable version of the folder name. But, you don't need to rename the folder.

### 1.3 Creating Objects Exclusively with SQL Statements

Now let's practice creating objects strictly with SQL statements. You'll see how quickly and efficiently you can accomplish object creation with SQL code.

The first object we'll create is a worksheet. A worksheet is a container in which you can draft, revise, execute and save SQL statements, and folders are used to organize those worksheets.

In the next few steps we'll show you how to create a new worksheet from the SQL file for this lab. The idea is for you to open the file, scroll down to this part of the lab and run the SQL statements in that file to complete the lab.

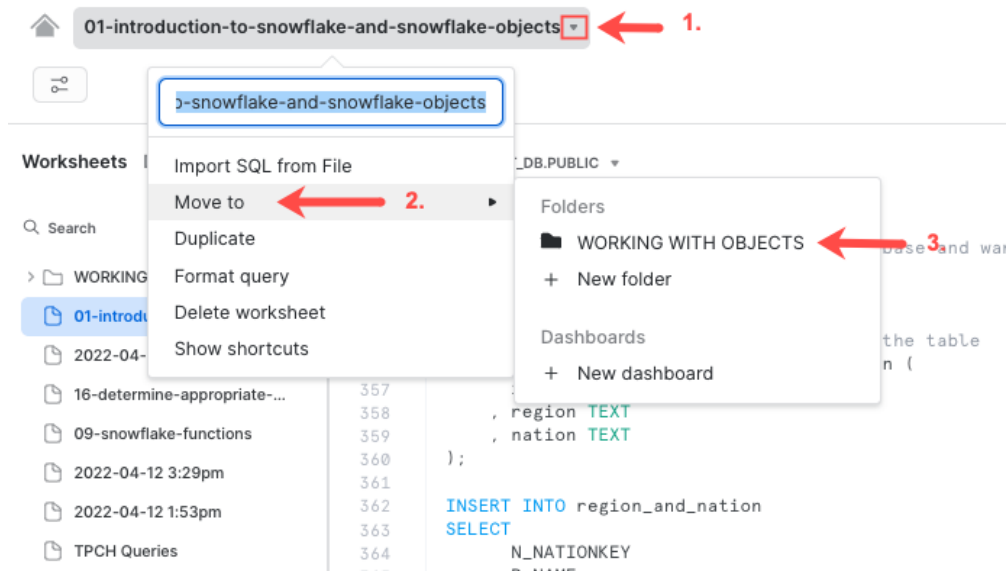
1.3.1 Click Worksheets in the left-hand navigation bar.

1.3.2 Click the down arrow next to the title of the folder (WORKING WITH OBJECTS).

1.3.3 Select "Create Worksheet from SQL File" from the drop-down menu.

1.3.4 Use the file dialog box to navigate to the file for this lab (should have the same name as the title of this lab) and open the file.

1.3.5 Once the file is open, put the worksheet in the folder you created for this class as shown below:



**Figure 10:** Moving a worksheet to a folder

1.3.6 Now scroll down to this place in the lab and use the contents of the file that follow to continue the lab.

1.3.7 Set context defaults for this course

By setting these defaults, you will ensure that these will be part of your context by default each time you open a worksheet in subsequent labs.

```
ALTER USER [login]
SET default_warehouse=[login]_wh
  default_namespace=[login]_db.public
  default_role=training_role;
```

1.3.8 Set the context for the remainder of this lab

The context defines the default database/schema location in which our SQL statements run, and the WH and role to use in support of this. So, let's set the context so we can run our SQL statements.

Run the following statements in the SQL portion of this worksheet.

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

1.3.9 Drop all the objects previously created

Now we're going to drop everything we created with the Snowsight UI:

```
DROP TABLE region_and_nation;
DROP SCHEMA [login]_SCHEMA;
DROP DATABASE [login]_DB;
DROP WAREHOUSE [login]_WH;
```

1.3.10 Create the warehouse by executing the following statement:

```
CREATE WAREHOUSE [login]_WH
  WITH WAREHOUSE_SIZE = 'XSMALL'
  AUTO_SUSPEND = 180
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED = TRUE;
```

Since we set INITIALLY\_SUSPENDED = TRUE, the warehouse isn't actually running. Let's confirm its status and then start the warehouse.

1.3.11 Run the following statements to confirm the warehouse status and to start it

```
-- Use this to confirm the warehouse's status
SHOW WAREHOUSES like '[login]_WH';

-- RESUME will start the warehouse, SUSPEND will stop the warehouse
ALTER WAREHOUSE [login]_WH RESUME;

-- Now add this warehouse to the current context
USE WAREHOUSE [login]_WH;
```

1.3.12 Run the following statements to create the required tables.

Now let's create one of the first tables we need for our business analysts. It is a table that contains the regions and nations served by Snowbear Air and it will be used in many reports across the company's business functions.

In order to create this table, we need to run SQL statements in a worksheet.

```
-- These statements create the database and schema
CREATE DATABASE [login]_DB;
CREATE SCHEMA [login]_SCHEMA;

-- These statements determine which database and warehouse will be used
USE DATABASE [login]_DB;
USE SCHEMA [login]_SCHEMA;

-- These statements create and populate the table
CREATE OR REPLACE TABLE region_and_nation (
    id INTEGER
    , region TEXT
    , nation TEXT
);
```

1.3.13 Now insert the data you need into the table.

```
INSERT INTO region_and_nation
SELECT
    N_NATIONKEY
    , R_NAME
    , N_NAME
FROM
    TRAINING_DB.TPCH_SF1.NATION N
    INNER JOIN TRAINING_DB.TPCH_SF1.REGION R ON N.N_REGIONKEY = R.R_REGIONKEY
ORDER BY
    R_NAME, N_NAME;

-- Use this statement to confirm the table was populated
SELECT
    *
FROM
    region_and_nation
ORDER BY
    REGION, NATION;
```



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

## 1.4 Key Takeaways

- You can create database objects both via the Snowsight UI and by executing SQL code in a worksheet.
- Data Sharing options such as Data Marketplace can be accessed via Snowsight by users with the appropriate privileges.
- You can browse database objects and view their details by using the navigation bar, the Object Selection and Object Details panes.
- The context of a worksheet session consists of a role, schema, database and warehouse.
- The context of a worksheet can be set via the Snowsight UI or via SQL statements.
- You can create folders in which to save and organize worksheets.

## 2 Visualizations in Snowsight

The purpose of this lab is to show you how to use the visualization features and tools available in Snowsight. Specifically, you'll learn how to leverage Contextual Statistics for specific columns in a table in order to gain quick insights into data. Also, you'll learn how to create a dashboard from an existing query.

### Learning Objectives:

- How to create a dashboard from a worksheet
- How to use auto-complete to write a query
- How to use ad-hoc filters to get insights into data
- How to add tiles to a dashboard
- How to share a dashboard

### Scenario:

Snowbear Air is interested in seeing a year-by-year summary of gross sales. You've been asked to write a query with a graph and share it via a Snowflake dashboard. You've decided to use the PROMO\_SALES\_CATALOG schema to accomplish your task.

### HOW TO COMPLETE THIS LAB

In the previous lab you may have used the SQL code file for that lab to create a new worksheet and then just run the code provided. That approach will not work for this lab because of the nature of what you will be doing.

Instead, open the document in a text editor such as TextEdit (Mac) or Notepad (Windows). Then you can either type the commands directly, or cut and paste the code into a worksheet as indicated in the instructions. Microsoft Word is not recommended as it can often introduce hidden characters that will cause your code not to run.

Also, it is not recommended that you cut and paste from the workbook pdf as the same problem previously described can occur.

Let's get started!

### 2.1 Creating a Dashboard

2.1.1 Using skills you've already learned, create a new folder called Visualizations in Snowsight.

2.1.2 Within your new folder, create a new worksheet called 'Dashboard Data'.

2.1.3 Set the context by executing the statements below in your worksheet:

```
USE ROLE TRAINING_ROLE;  
USE SCHEMA SNOWBEARAIR_DB.PROMO_CATALOG_SALES;  
USE WAREHOUSE [login]_WH;
```

2.1.4 Now let's write our query. Just take a moment to review it. You don't need to run it.

```
SELECT
    *
FROM
    CUSTOMER C
    INNER JOIN NATION N ON C.C_NATIONKEY = N.N_NATIONKEY
    INNER JOIN REGION R ON N.N_REGIONKEY = R.R_REGIONKEY
    INNER JOIN ORDERS O ON C.C_CUSTKEY = O.O_CUSTKEY
    INNER JOIN LINEITEM L ON O.O_ORDERKEY = L.L_ORDERKEY
    INNER JOIN PART P ON L.L_PARTKEY = P.P_PARTKEY
    INNER JOIN SUPPLIER S ON L.L_SUPPKEY = S.S_SUPPKEY;
```

As you can see, you are selecting \*, which returns all the fields. However, all you need is a year column and a gross revenue column.

## 2.2 How to use Auto-Complete to write a query

Now you're going to use the auto-complete feature to add year and gross revenue to the columns returned by the query.

### **The Auto-Complete Feature**

The auto-complete feature suggests SQL Keywords, databases, schemas, tables, field names, functions and other object types while you are typing.

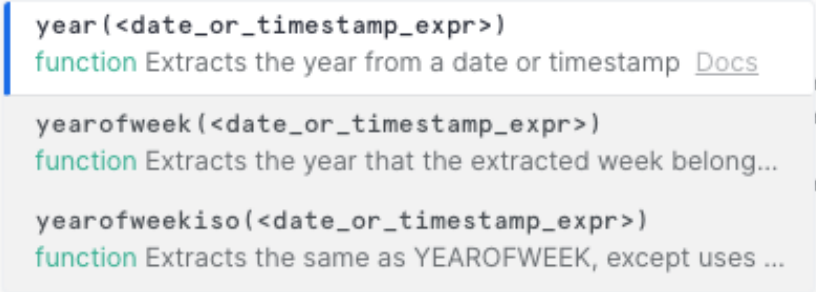
By using auto-complete, you can work faster and make fewer typos.

We need to get the O\_ORDERDATE column from the ORDERS table and pass it through a function that will extract the year. Let's do that now.

### 2.2.1 Add the YEAR column using auto-complete

Remove the asterisk and type YEAR as shown below:

```
SELECT
    YEAR|
FROM
```

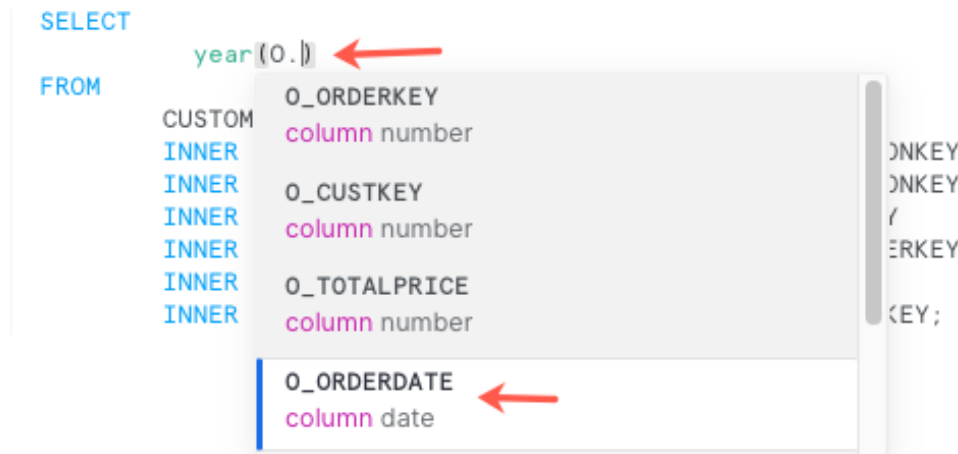


**Figure 11:** Auto-complete

As you can see, when you type the word year, the auto-complete feature generates a list of functions to select from. Select YEAR and hit enter.

### 2.2.2 Add O\_ORDERDATE as an argument in the year function

Type O and then a period. you should see the drop-down menu shown below:



**Figure 12:** Auto-complete

As you type table names and table aliases, the auto-complete feature generates a list of fields you can select from.

Select O\_ORDERDATE and alias the column as YEAR.

### 2.2.3 Add the next column using auto-complete

The next step is substantially the same as the previous step.

First, add a column containing the the sum of the gross revenue. Type a comma and then SUM.

You should be offered the SUM function. Choose that function.

Then within the parentheses of the function, type L and a period. Auto-complete should generate a list with the L\_EXTENDEDPRISE field.

Alias the field as SUM\_GROSS\_REVENUE

### 2.2.4 Add a GROUP BY clause and an ORDER BY clause

Below the FROM clause, add a GROUP BY YEAR clause and an ORDER BY YEAR clause.

You should now have the query below.

```

SELECT
    YEAR(O.O_ORDERDATE) AS YEAR
    , SUM(L.L_EXTENDEDPRICE) AS SUM_GROSS_REVENUE

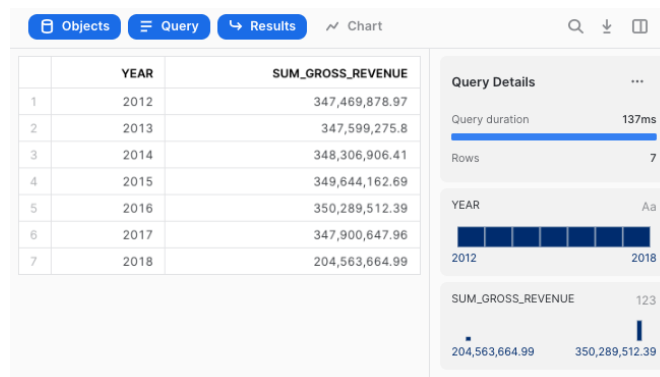
FROM
    CUSTOMER C
    INNER JOIN NATION N ON C.C_NATIONKEY = N.N_NATIONKEY
    INNER JOIN REGION R ON N.N_REGIONKEY = R.R_REGIONKEY
    INNER JOIN ORDERS O ON C.C_CUSTKEY = O.O_CUSTKEY
    INNER JOIN LINEITEM L ON O.O_ORDERKEY = L.L_ORDERKEY
    INNER JOIN PART P ON L.L_PARTKEY = P.P_PARTKEY
    INNER JOIN SUPPLIER S ON L.L_SUPPKEY = S.S_SUPPKEY

GROUP BY
    YEAR
ORDER BY
    YEAR;

```

### 2.2.5 Run the query and check the results

You should see the results below:



**Figure 13:** Query Results

Note that there are two kinds of information to the right of the result. There are the Query Details pane and the Contextual Statistics pane. The Query Details pane shows the duration of the query and the number of rows returned. The Contextual Statistics pane helps you make sense of your data at a glance.

Also note that there is a comma for each value in the YEAR column. To change it, hover your cursor over the YEAR column header, then click the ellipses that appears to the right. Click the comma button to remove the commas.:

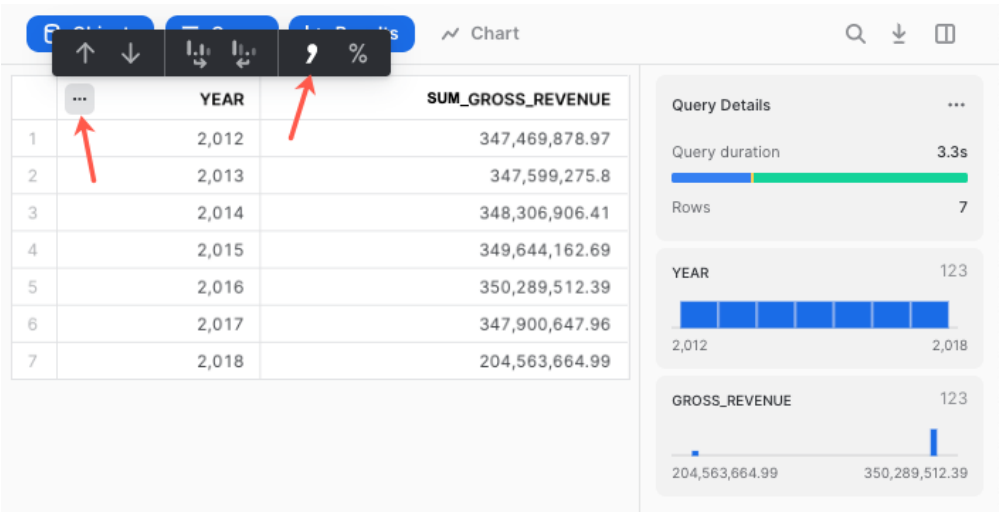


Figure 14: Removing comma from the YEAR column

2.2.6 Click the Query Details pane

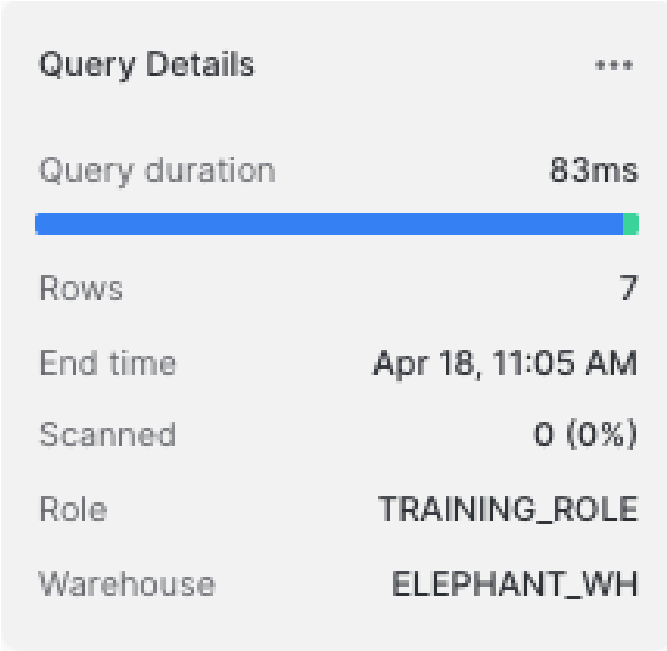


Figure 15: Query Details

Note that in addition to the query duration and the rows scanned, it shows the end time of the query, the role used and the warehouse used.

2.3 How to use ad-hoc filters to get insights into data

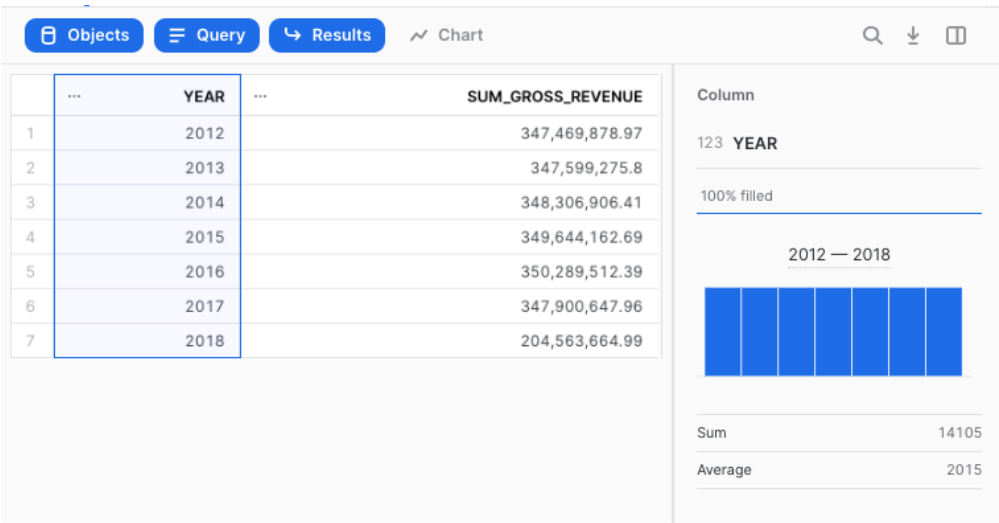
Now let’s work with ad-hoc filters so you can explore and gain insights into the data returned by the query.

2.3.1 Click on the section with the graph to apply a filter

Note there are two panes with Contextual Statistics, one that shows a graph of data from 2012 to 2018 and is labeled “YEAR”, the other that shows the highest and lowest values in the data set returned and is labeled “SUM\_GROSS\_REVENUE”. The contextual statistics, one for each column returned by the query, can be used interactively as filters on the query result. Let’s explore how they work.

2.3.2 Click on the YEAR filter

You should now see the filter. On the left is the data and the YEAR column is highlighted. On the right is the filter itself.



**Figure 16:** YEAR filter

2.3.3 Click on the leftmost column in the graph’s filter

Now the results should be filtered for 2012 only.

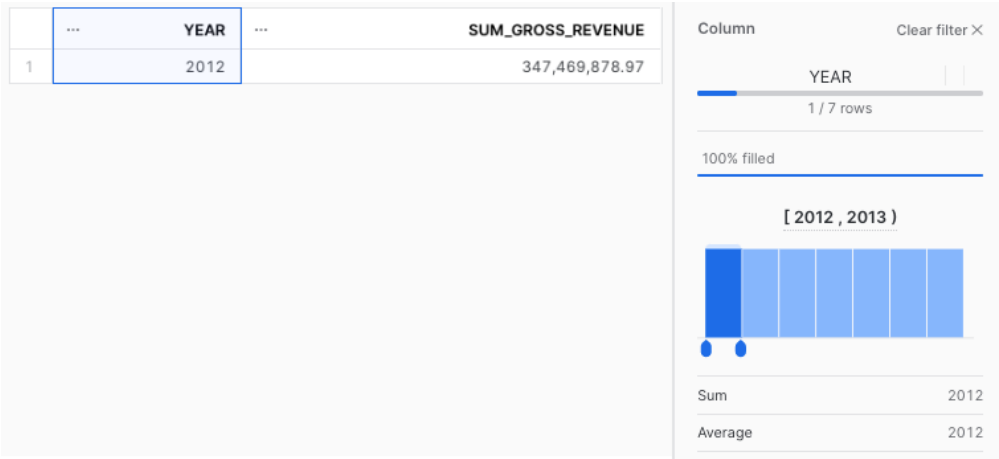


Figure 17: YEAR filter, 2012

2.3.4 Select 2012 and 2013

Note that there are two oval selectors beneath the chosen column in the filter’s graph. Click, hold and drag the right-most selector to include both 2012 and 2013. Your filter should appear as shown below:

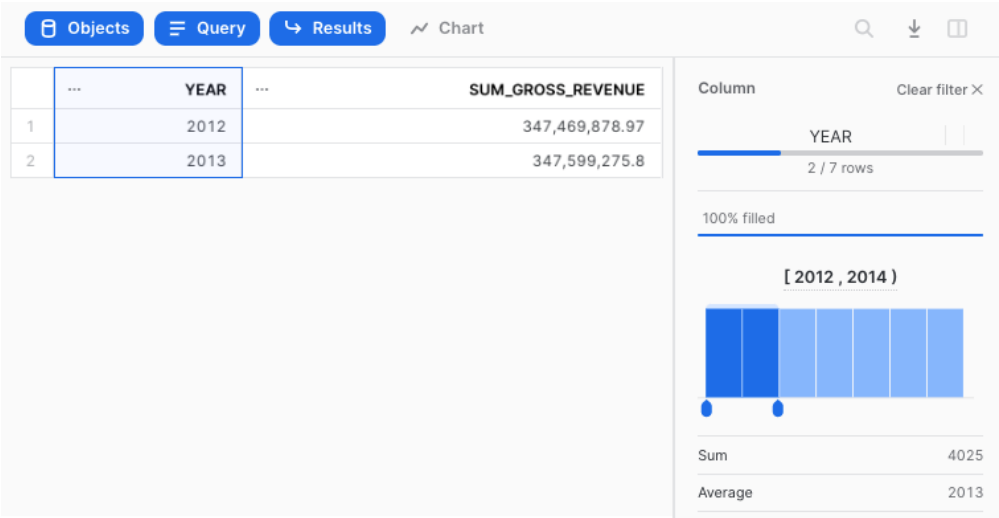


Figure 18: YEAR filter, 2012-2013

Now click different bars, or select any combination of multiple bars to see how the filter changes the data shown.



2.3.5 Click the Clear filter button

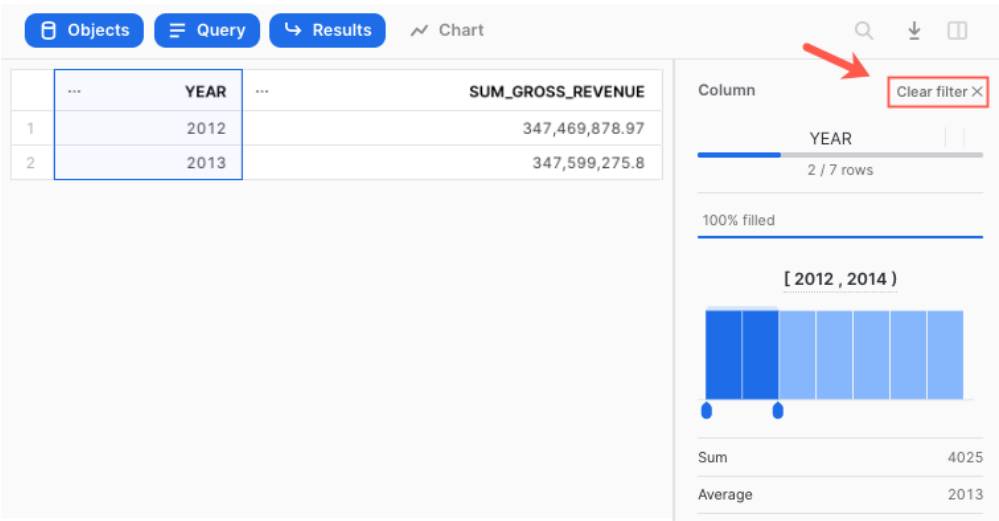


Figure 19: Clear filter button

The filter should appear as it did before.

2.3.6 Click the Close button (X)

This should clear the column selected and you should see the Query Details pane and the YEAR and SUM\_GROSS\_REVENUE filters.

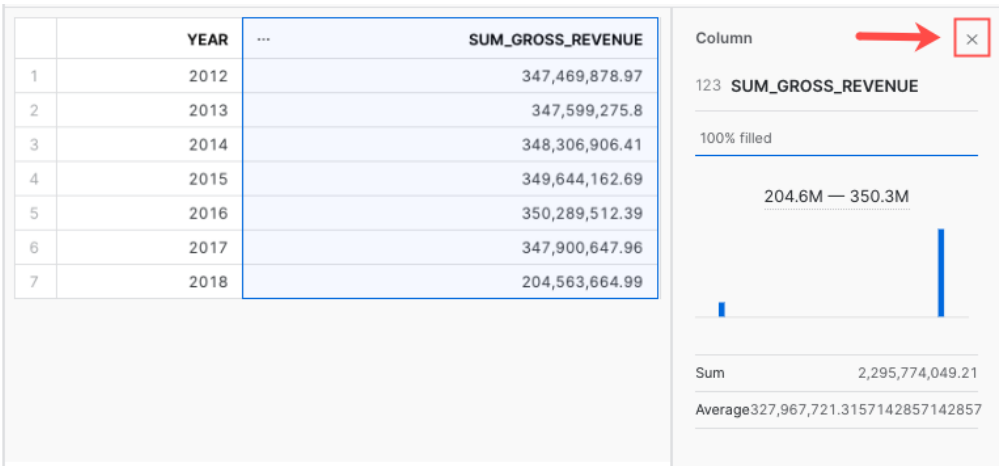
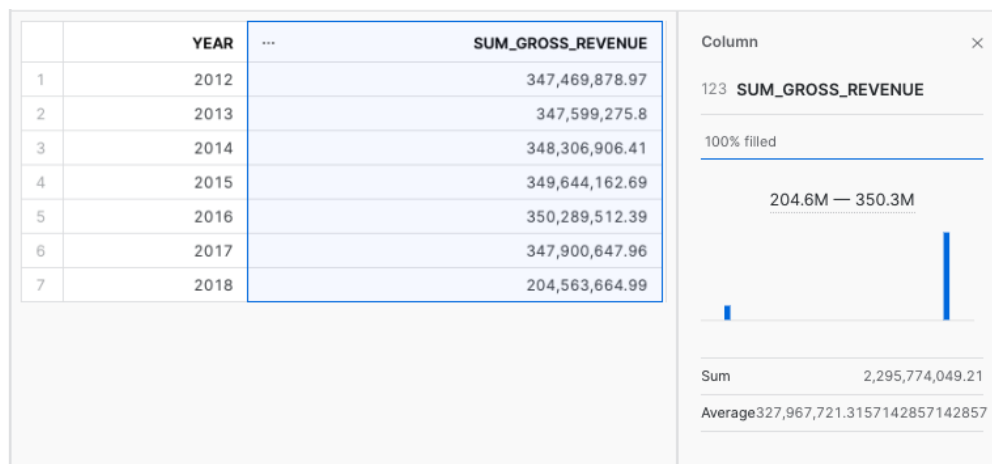


Figure 20: Clear Selection Button

2.3.7 Click the SUM\_GROSS\_REVENUE filter

The filter should appear as below. Click the columns and observe how the data is filtered. Clicking between the columns will display the following message: “Query produced no results”. That’s because there is a gap between the value in the left-most bar and the value of the right-most bar.



**Figure 21:** SUM\_GROSS\_REVENUE filter

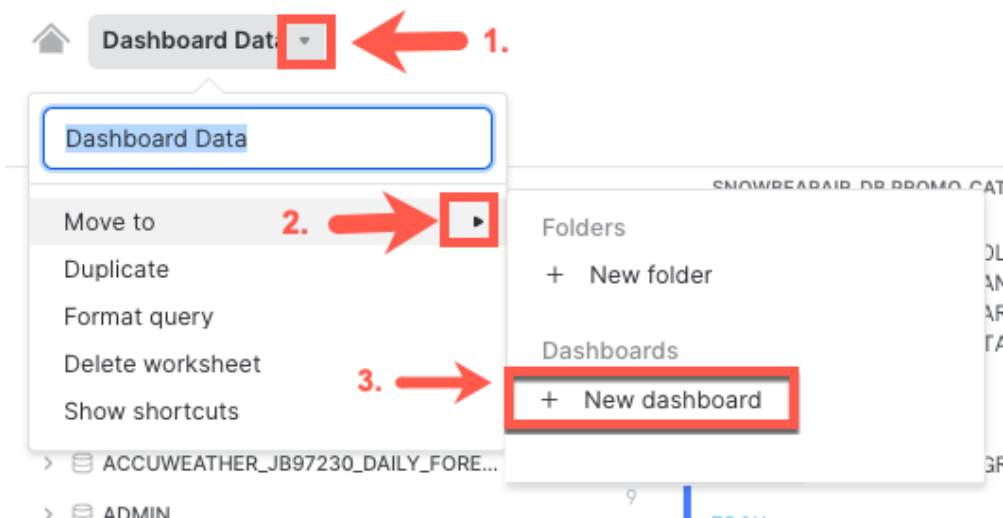
2.3.8 Click the Clear filter and Close selection buttons

2.3.9 Move the worksheet to the Dashboards

Now let's create our dashboard. You can do this by either creating a brand new dashboard, or by moving an existing worksheet to Dashboards. Let's try this second method.

2.3.10 Click the down arrow next to the worksheet name (Dashboard Data)

2.3.11 Select Move to, then New dashboard from the dialog box.



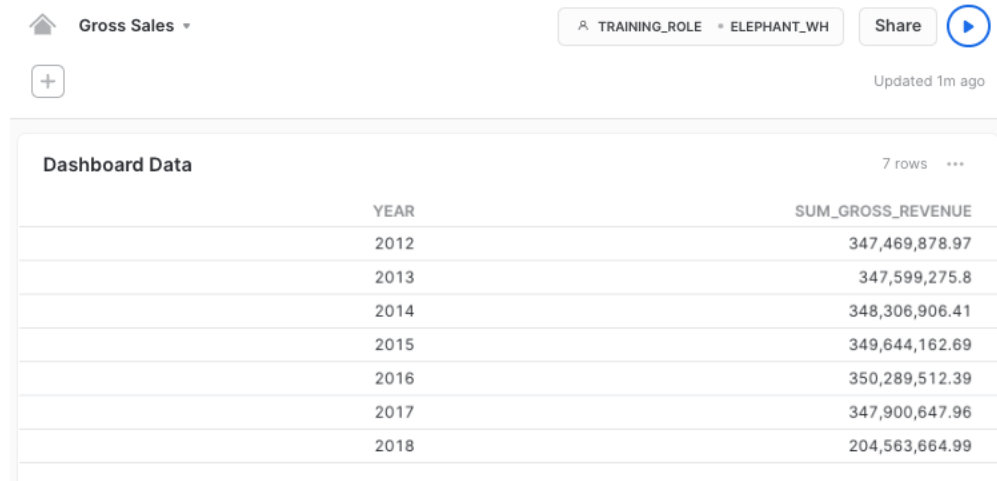
**Figure 22:** Move to Dashboards

2.3.12 Name the new dashboard Gross Sales and click the Create Dashboard button.

You should now see a screen that looks like the worksheet itself. This is where you can edit the query that creates the data for the dashboard. In the upper-left hand corner there should be a “Return to Gross Sales” link.

2.3.13 Click the Return to Gross Sales link

You should now see the dashboard but in presentation mode.



Dashboard Data		7 rows
YEAR	SUM_GROSS_REVENUE	
2012	347,469,878.97	
2013	347,599,275.8	
2014	348,306,906.41	
2015	349,644,162.69	
2016	350,289,512.39	
2017	347,900,647.96	
2018	204,563,664.99	

**Figure 23:** Presentation mode

The data itself is in a “tile” that is present on the dashboard. Tiles are used to present data or graphs in the dashboard.

## 2.4 How to add tiles to a dashboard

Now we’re going to add a new tile so we can show a graph.

2.4.1 Click the plus sign just below the home button and the dashboard name to create a graph

A dialog box should appear with a “New Tile from Worksheet” button.

2.4.2 Click the New Tile from Worksheet button

A new worksheet should appear with no SQL code.

2.4.3 Paste the query below into the empty pane.

```
SELECT
    YEAR(O.O_ORDERDATE) AS YEAR
  , SUM(L.L_EXTENDEDPRICE) AS SUM_GROSS_REVENUE
FROM
    CUSTOMER C
  INNER JOIN NATION N ON C.C_NATIONKEY = N.N_NATIONKEY
  INNER JOIN REGION R ON N.N_REGIONKEY = R.R_REGIONKEY
  INNER JOIN ORDERS O ON C.C_CUSTKEY = O.O_CUSTKEY
  INNER JOIN LINEITEM L ON O.O_ORDERKEY = L.L_ORDERKEY
  INNER JOIN PART P ON L.L_PARTKEY = P.P_PARTKEY
  INNER JOIN SUPPLIER S ON L.L_SUPPKEY = S.S_SUPPKEY
GROUP BY
    YEAR
ORDER BY
    YEAR;
```

2.4.4 Rename this tile

Just like with the worksheets we created earlier, a time and date should appear at the top of the worksheet. Click the time/date and change the time and date to “Dashboard Graph”.

2.4.5 Run the query

A result pane identical to the one we saw before should appear.

2.4.6 Click the ellipses in the heading of the YEAR column and remove the commas from the year values

2.4.7 Click the Chart button

The Chart button is just above the result pane, next to the blue results button.

A line graph should be chosen by default:

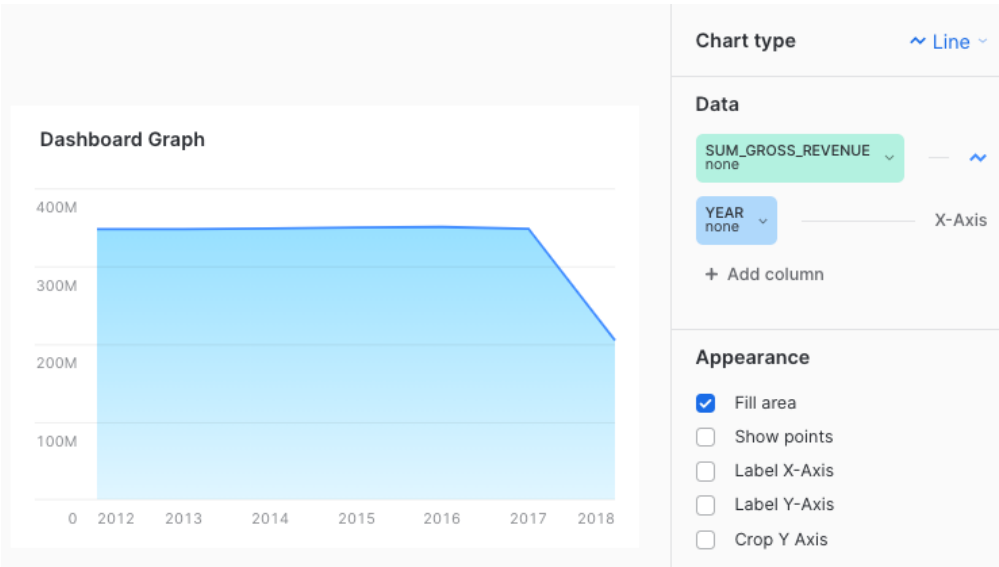


Figure 24: Presentation mode

2.4.8 Click Return to Gross Sales in the upper-left hand corner

You should now see a completed dashboard like the one shown below:

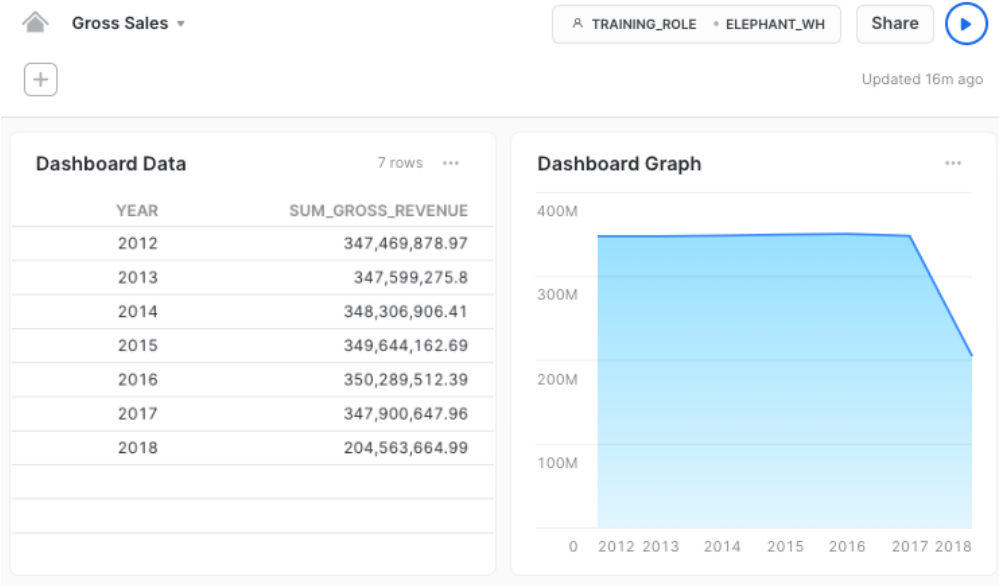


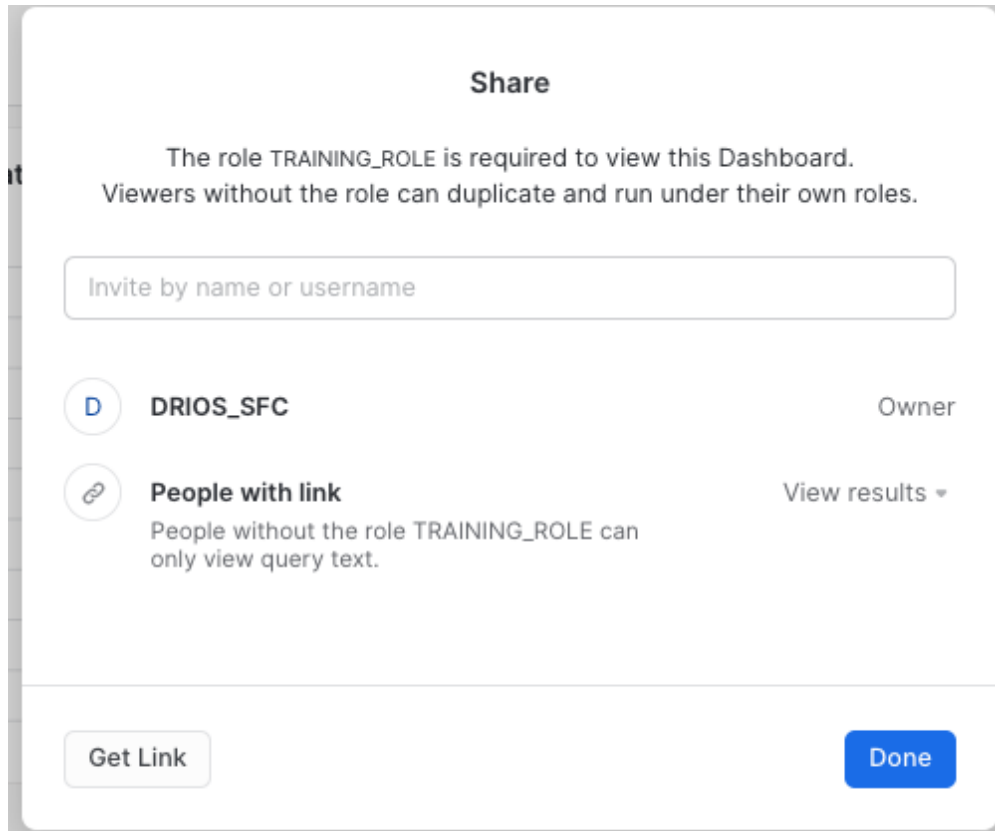
Figure 25: Presentation mode

Now let's see how to share our dashboard.

## 2.5 How to share a dashboard

### 2.5.1 Click the share button in the upper-right hand corner

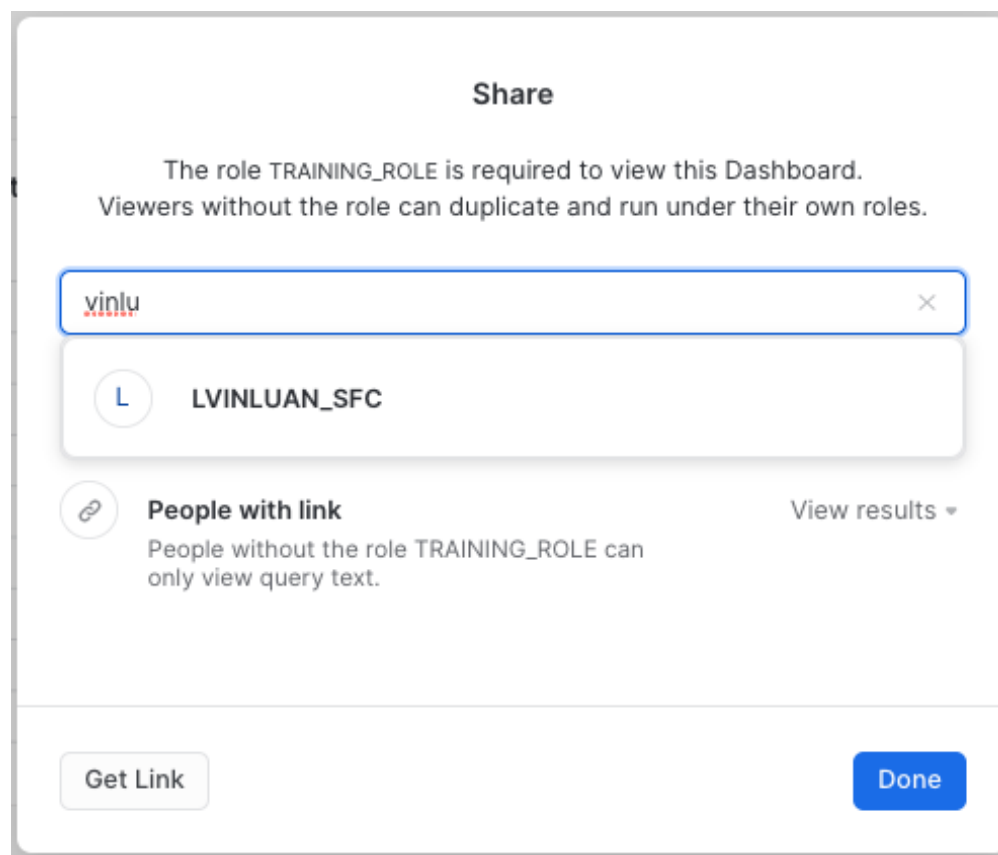
In this dialog box you can search for and invite someone to view and use this dashboard.



**Figure 26:** Share dialog box

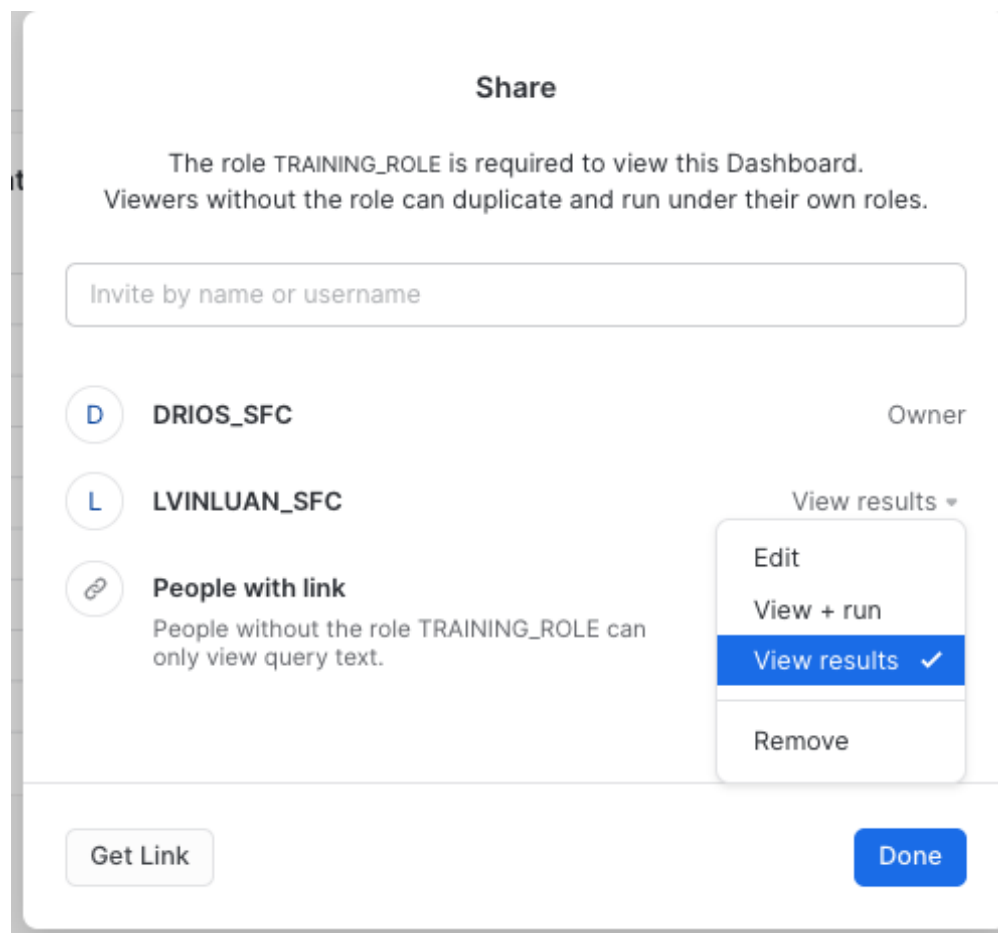
You don't have anyone to share with, so we'll just walk you through the process so you understand it. It's so simple!

First, you would select a user by typing their user name:



**Figure 27:** Sharing with a user

Next you would select a permission level for them, such as Edit, View + run, or just View results:



**Figure 28:** Granting permissions to user

Then you would click the Done button. That's it!

## 2.6 Key Takeaways

- The auto-complete feature is a useful tool for writing queries. It helps you work faster and with fewer typos.
- While conducting ad-hoc analyses you can use filters to gain insights into your data.
- You can create dashboards out of existing worksheets.
- Snowflake makes it super-easy to share worksheets with colleagues.



### 3 Querying Data with Time Travel

The purpose of this lab is to familiarize you with how Time Travel can be used to analyze data. Specifically, you'll take a data set and compare two different points in time to satisfy a what-if question. In order to do this, you'll clone a table and use syntax designed to query two distinct Time Travel data snapshots.

#### Learning Objectives:

- How to clone a table
- How to write query clauses that support Time Travel actions
- How to fetch and utilize the ID of the last query you ran

#### Scenario:

Snowbear Air charges a different tax rate depending on the country in which the customer lives. It just so happens that countries across the world are thinking about setting a tax rate of around 5% for the kind of products that Snowbear Air sells via its promotional catalog.

Although Snowbear Air doesn't know what the new tax rate will be, leadership has decided it would like to see some kind of what-if analysis in order to determine how much more they will potentially be collecting. The concern is that a higher tax rate could result in consumers paying a higher cost based on current product prices. If that price is too high, Snowbear Air may make the decision to lower its prices in order to keep sales high.

#### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

#### 3.1 Conducting a What-If Scenario

The goal will be to determine what has been paid in taxes over the past seven years and what the amount would have been if the tax rate had been 5%.

3.1.1 Using the skills you learned in the first lab, create a new folder called Time Travel

3.1.2 Using the skills you learned in the first lab, create a worksheet inside the folder you just created and name it Time Travel

3.1.3 Create a new schema

For the purposes of this lab, we're going to create a new schema. Let's create the schema by running the following statements in your new worksheet:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;

CREATE DATABASE IF NOT EXISTS [login]_DB;
CREATE SCHEMA IF NOT EXISTS [login]_DB.[login]_LAB;

USE SCHEMA [login]_DB.[login]_LAB;
```

3.1.4 The Plan

We are going to clone the LINEITEM table into our schema, which will make a copy of the table that still points to all the data in the original table. Next, we are going to fetch a reference point in time. Then, we will update the total price column in the clone. Finally, using that point of reference, we will run a query to compare the data at the time of cloning to the data after the update.

3.1.5 Clone the LINEITEM table

The LINEITEM table shows the details of each order. Each line shows a specific product in the order along with the discount and tax percentages that were applied to that line. What we're going to do is update the values in the tax percentage column called L\_TAX and then compare how the change impacts the net cost to the customer for the product represented by each line item.

Now let's create a zero-copy clone of LINEITEM. This is essentially a snapshot of the table which shares the underlying data of the original table at the point of creation. However, once we make a change to the clone as part of this lab, those changes will be unique to our new table. Additionally, any changes to the original table of course do not impact the clone.

For purposes of this lab, we are going to work with only 1000 records so our queries execute quickly. Whether you're working with 1,000 rows or 1,000,000 rows, the concept is the same.

The code below will create a 1000 row sample table and then clone that sample table:

```
CREATE OR REPLACE TABLE LINEITEM_SAMPLE AS
SELECT *
FROM SNOWBEARAIR_DB.PROMO_CATALOG_SALES.LINEITEM
ORDER BY L_ORDERKEY, L_LINENUMBER LIMIT 1000;

CREATE OR REPLACE TABLE LINEITEM_CLONE CLONE LINEITEM_SAMPLE;
```

### 3.1.6 Verify the clone

Run the code below to ensure the clone was created and is properly populated:

```
SELECT * FROM LINEITEM_CLONE;
```

You should now see the contents of the newly cloned table below the query.

### 3.1.7 Updating the LINEITEM table

The following UPDATE statement will update the tax rate column to apply a 5% tax rate. Go ahead and run it now:

```
UPDATE
  LINEITEM_CLONE
SET
  L_TAX = 0.05
FROM
  LINEITEM_CLONE L
  INNER JOIN SNOWBEARAIR_DB.PROMO_CATALOG_SALES.PART P ON L.L_PARTKEY =
    P.P_PARTKEY;
```

### 3.1.8 Fetch the last query id

NOW run the statement below to fetch the query id and verify the contents of the UPDATE\_ID variable:

```
SET UPDATE_ID = LAST_QUERY_ID();

SELECT $UPDATE_ID;
```

Note that what we just did was to set the value of a SQL variable called UPDATE\_ID to the unique id of the last query we ran by calling the LAST\_QUERY\_ID() function.

### 3.1.9 Checking the results

Now that we have our reference point, we'll be able to access the data that existed in the table prior to executing the UPDATE.

The query below fetches the original values and the new values in two separate result sets and then joins them together.

Note that the sub-query that fetches the original values uses the following BEFORE clause:

- BEFORE (STATEMENT => \$update\_id)

In essence it is fetching the state of the cloned table prior to our update. The goal with this query is to check our results and determine if the original values are indeed different from the new ones.

Go ahead and run the query now:

```

SELECT
    ROUND((( L.L_QUANTITY * P.P_RETAILPRICE ) * (1-L.L_DISCOUNT)) *
        (1+L.L_TAX),2) AS TOTALCOST_NEW
    , ORG.TOTALCOST_ORIGINAL
    , L.L_QUANTITY
    , P.P_RETAILPRICE
    , L.L_DISCOUNT
    , L.L_TAX AS NEW_TAX
    , ORG.L_TAX AS ORG_TAX

FROM
    LINEITEM_CLONE L
LEFT JOIN
    (
        SELECT
            LC.L_ORDERKEY
            , LC.L_LINENUMBER
            , L_TAX
            , ROUND((( LC.L_QUANTITY * P.P_RETAILPRICE ) * (1-LC.L_DISCOUNT)) *
                (1+LC.L_TAX),2) AS TOTALCOST_ORIGINAL

        FROM
            LINEITEM_CLONE
        BEFORE
            (STATEMENT => $update_id) AS LC
        INNER JOIN SNOWBEARAIR_DB.PROMO_CATALOG_SALES.PART P ON LC.L_PARTKEY =
            P.P_PARTKEY

    ) ORG ON L.L_ORDERKEY = ORG.L_ORDERKEY AND L.L_LINENUMBER = ORG.L_LINENUMBER
INNER JOIN SNOWBEARAIR_DB.PROMO_CATALOG_SALES.PART P ON L.L_PARTKEY =
    P.P_PARTKEY

ORDER BY
    L.L_ORDERKEY, L.L_LINENUMBER;

```

We won't explain the fields as the field names are probably self-explanatory. The query also shows the quantity, retail price, discount and tax percentages so you can see the input values that went into the calculations.

As you reviewed the results, you probably noticed that in some instances the tax rate was the same, in some it was lower and in some it was higher. So, the impact of a tax change could be that customers will pay more tax or less tax than before.

Naturally, this example is a bit more simplistic than what you might find in a real scenario. Regardless, it does demonstrate the power of Time Travel to compare two points in time.

### 3.1.10 Cleaning Up

Run the statement below to suspend your warehouse:

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

## 3.2 Key Takeaways

- In order to leverage Time Travel data for data analysis purposes, you need a reference point. This can be a `TIMESTAMP`, an `OFFSET` or a `STATEMENT`.
- You will need to put an `AT|BEFORE` clause in your query in order to pull the results of a particular data set as of a particular point in time.
- Zero-copy cloning means that when you clone a table, at the point in time the cloned table is created, it shares the same data as the original table. Once you start making changes to either table's data, the new data resides only within the table in which the change was made.

## 4 Multi-Table Inserts

In this lab you will learn how to execute multi-table inserts, how to use SWAP, and how execute MERGE statements.

### Learning Objectives:

- How to use sequences to create unique values in a primary key column
- How to use unconditional multi-table insert statements
- How to use ALTER TABLE SWAP WITH to swap table content and metadata
- How to use MERGE statements to add new rows to a table

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

Let's get started!

### 4.1 Working with Sequences

In this section you will learn how to create and use a sequence. We will use a sequence in the next section to replace a UUID value with an integer as the unique identifier for each row in a table. Then we will use the same sequence to express a primary key-foreign key relationship between two tables.

Read the section below to get familiar with sequences in Snowflake.

#### **SEQUENCE**

A SEQUENCE is a named object that belongs to a schema in Snowflake. It consists of a set of sequential, unique numbers that increase in value or decrease in value based on how the sequence is configured. Sequences can be used to populate columns in a Snowflake table with unique values.

#### **SEQUENCE PARAMETERS**

1. NAME (required): Identifies the sequence as a unique object within the schema.
2. START (optional): The first value of the sequence. Default is 1.
3. INCREMENT (optional): The step interval of the sequence. Default is 1.

#### **NOTE**

Sequence values are generally contiguous but sometimes there can be a gap. Regardless, they should either increase in value if the INCREMENT value is positive, and decrease if the INCREMENT value is negative.

#### 4.1.1 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;
```

#### 4.1.2 Set the context for the lab

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

#### 4.1.3 Create a sequence called item\_seq

Here we're going to create a sequence called item\_seq. We will then use it as a primary key in a table. Note that the start value is 1 and that the increment value is 1. This means we can expect the sequence to start with 1 and continue with 2, 3, 4, 5, etc.

```
CREATE OR REPLACE SEQUENCE item_seq START = 1 INCREMENT = 1;
```

#### 4.1.4 Now evaluate the nextval expression of the sequence we just created once to see the first value.

```
-- Bump the seq and show the next value.  
SELECT Item_seq.nextval;
```

As you can see, the value is 1. The expression .nextval returns a new value each time it is evaluated. If you want to apply it to a table you will need to use the nextval expression for the first time right after creating the sequence. If not it will pick up the next number in the sequence instead of the very first. Let's test this idea and observe the results.

#### 4.1.5 Create a table, insert some values, and select all values from the table.

```
-- Create a table with the sequence.  
CREATE OR REPLACE TABLE item_table ( Item_id INTEGER default Item_seq.nextval,  
    description VARCHAR(20));  
  
-- Insert some rows.  
INSERT INTO item_table (description) VALUES ('Wheels'), ('Tires'), ('hubcaps');  
  
-- Select all values  
SELECT * FROM item_table;
```

As you can see, the first row has an item\_id of 2 rather than 1. This is because when we created the table, we had already previously iterated to the first sequence value 1. So when we evaluated the nextval expression a second time, the next value was fetched, which was 2.

Let's try this again and recreate the sequence and the table.

#### 4.1.6 Recreate the sequence and the table

```
-- Reset the sequence. Recreating the sequence is the only way to reset a sequence.
CREATE OR REPLACE SEQUENCE Item_seq START = 1 INCREMENT = 1;

-- Create a table with the sequence.
CREATE OR REPLACE TABLE item_table ( Item_id INTEGER default Item_seq.nextval,
description VARCHAR(20));

-- Insert some rows.
INSERT INTO item_table (description) VALUES ('Wheels'), ('Tires'), ('hubcaps');

-- Select all rows the from the table
SELECT * FROM item_table;
```

As you can see, the sequence applied to the table now starts off with 1.

#### 4.1.7 DROP table and sequence.

```
DROP SEQUENCE item_seq;
DROP TABLE item_table;
```

#### 4.1.8 Try the different sequences below and observe the results

The purpose of the exercise below is to get an idea of how the START and INCREMENT values change the resulting values in a sequence.

```
CREATE OR REPLACE SEQUENCE seq_1 START = 1 INCREMENT = 1;
CREATE OR REPLACE SEQUENCE seq_2 START = 2 INCREMENT = 2;
CREATE OR REPLACE SEQUENCE seq_3 START = 3 INCREMENT = 3;

--run each statement below 3 or 4 times
SELECT seq_1.nextval;
SELECT seq_2.nextval;
SELECT seq_3.nextval;
```

## 4.2 Working with Unconditional Multi-Table Inserts

In this section, we will take a table called member and divide its data between a customer table, an address table, and a phone table. This will allow a single customer to have multiple addresses and multiple phone numbers.

Since we are splitting the original table into three disparate tables, MEMBER\_ID is going to be used for the primary key-foreign key relationship between the three tables, an AUTOINCREMENT will not work. We will solve this by using a sequence to insert a new numeric value into each table.

In order to do this, we will use a multi-table insert to copy the member data into the three different tables. We will also replace the UUID-based primary key with a sequence.



#### 4.2.1 Create a sequence

Before we execute the multi-table insert, we are going to create a sequence to create a unique ID outside of the table and use it for the MEMBER\_ID Column. The default for a sequence is START = 1 and INCREMENT = 1. Use this as the default for the MEMBERS table.

```
CREATE OR REPLACE SEQUENCE member_seq START = 1 INCREMENT = 1;
```

#### 4.2.2 Create the member, member\_address and member\_phone tables

```
CREATE OR REPLACE TABLE member (  
  member_id INTEGER DEFAULT member_seq.nextval,  
  points_balance NUMBER,  
  started_date DATE,  
  ended_date DATE,  
  registered_date DATE,  
  firstname VARCHAR,  
  lastname VARCHAR,  
  gender VARCHAR,  
  age NUMBER,  
  email VARCHAR  
);  
  
CREATE OR REPLACE TABLE member_address (  
  member_id INTEGER,  
  street VARCHAR,  
  city VARCHAR,  
  state VARCHAR,  
  zip VARCHAR  
);  
  
CREATE OR REPLACE TABLE member_phone (  
  member_id INTEGER,  
  phone VARCHAR  
);
```

#### 4.2.3 Populate the tables

Next you'll execute multi-table insert statement that will copy the data from an existing table into the member, member\_address, and member\_phone tables.

**UNCONDITIONAL MULTI-TABLE INSERT SYNTAX**

A multi-table insert statement can insert rows into multiple tables from the same statement. Note the syntax below:

```
INSERT ALL

  INTO <table 1> (<target_col1>,<target_col2>,<target_col3>)
  VALUES (<source_col1>,<source_col2>,<source_col3>)

  INTO <table 2> (<target_col4>,<target_col5>,<target_col6>)
  VALUES (<source_col4>,<source_col5>,<source_col6>)

FROM
  <database>.<schema>.<table>;
```

Now execute the statement below to populate your tables. Note how the syntax below reflects what you see in the box above:

```
INSERT ALL
  INTO member(member_id,
               points_balance,
               started_date,
               ended_date,
               registered_date,
               firstname,
               lastname,
               gender,
               age,
               email)
  VALUES (member_id,
           points_balance,
           started_date,
           ended_date,
           registered_date,
           firstname,
           lastname,
           gender,
           age,
           email)
  INTO member_address (member_id,
                       street,
                       city,
                       state,
                       zip)
  VALUES (member_id,
           street,
           city,
           state,
           zip)
  INTO member_phone(member_id,
                    phone)
  VALUES (member_id,
           phone)
  SELECT member_seq.NEXTVAL AS member_id,
```

```
        points_balance,  
        started_date,  
        ended_date,  
        registered_date,  
        firstname,  
        lastname,  
        gender,  
        age,  
        email,  
        street,  
        city,  
        state,  
        zip,  
        phone  
FROM "SNOWBEARAIR_DB"."MODELED"."MEMBERS";
```

#### 4.2.4 Confirm there is data in the tables

```
SELECT * FROM member ORDER BY member_id;  
  
SELECT * FROM member_address;  
  
SELECT * FROM member_phone;
```

#### 4.2.5 Join the tables and observe the results

Now let's run a few queries to see how we can join the tables we created to answer questions about the members and their contact information.

```
-- Execute a join between the member and the member_address table.  
SELECT  
    m.member_id  
    , firstname  
    , lastname  
    , street  
    , city  
    , state  
    , zip  
FROM  
    member m  
    LEFT JOIN member_address ma on m.member_id = ma.member_id;  
  
-- Run a join between the member, member_address and phone tables.  
SELECT  
    m.member_id  
    , firstname  
    , lastname  
    , street  
    , city  
    , state  
    , zip  
    , phone
```

**FROM**

```
member m
LEFT JOIN member_address ma on m.member_id = ma.member_id
LEFT JOIN member_phone mp on m.member_id = mp.member_id;
```

#### 4.2.6 Add another row to the MEMBER table

Since the MEMBER table is using the Sequence as the default, we can insert another row and it will get the next unique value.

**INSERT**

```
INTO member(points_balance,
             started_date,
             registered_date,
             firstname,
             lastname,
             gender,
             age,
             email)
VALUES (102000,
       '2014-9-12',
       '2014-8-1',
       'Fred',
       'Wiffle',
       'M',
       '34',
       'Fwiffle@AOL.com');
```

#### 4.2.7 Check the sequence number of the new row

Notice the value might not be what you would expect. In other words it may be unique but it may not be the next value in the sequence which would be 1001.

```
SELECT * FROM member WHERE member_id > 1000;
```

### 4.3 Working with Conditional Multi-Table Inserts

In this section we're going to expand on the work we did earlier. Specifically, we will use a conditional multi-table insert to break the member table into a gold\_member and a club\_member table. Gold members have more than 5,000,000 points in their points balance and Club members have less than 5,000,000. We will use the points\_balance column to determine who is a gold member.

#### 4.3.1 Create the tables

```
-- The first table will be the gold_member table.
CREATE OR REPLACE TABLE gold_member(
```

```
member_id INTEGER DEFAULT member_seq.nextval,  
points_balance NUMBER,  
started_date DATE,  
ended_date DATE,  
registered_date DATE,  
firstname VARCHAR,  
lastname VARCHAR,  
gender VARCHAR,  
age NUMBER,  
email VARCHAR  
);  
-- The second table will be the club_member table.
```

```
CREATE OR REPLACE TABLE club_member (  
  member_id INTEGER DEFAULT member_seq.nextval,  
  points_balance NUMBER,  
  started_date DATE,  
  ended_date DATE,  
  registered_date DATE,  
  firstname VARCHAR,  
  lastname VARCHAR,  
  gender VARCHAR,  
  age NUMBER,  
  email VARCHAR  
);
```

#### 4.3.2 Execute the inserts

```
INSERT ALL  
  WHEN points_balance >= 5000000 THEN  
    INTO gold_member(member_id,  
      points_balance,  
      started_date,  
      ended_date,  
      registered_date,  
      firstname,  
      lastname,  
      gender,  
      age,  
      email)  
  VALUES (member_id,  
    points_balance,  
    started_date,  
    ended_date,  
    registered_date,  
    firstname,  
    lastname,  
    gender,  
    age,  
    email)  
ELSE  
  -- points_balance is less than 500,000 so this member is a Club  
  member  
    INTO club_member (member_id,  
      points_balance,
```

```
        started_date,  
        ended_date,  
        registered_date,  
        firstname,  
        lastname,  
        gender,  
        age,  
        email)  
VALUES (member_id,  
        points_balance,  
        started_date,  
        ended_date,  
        registered_date,  
        firstname,  
        lastname,  
        gender,  
        age,  
        email)  
SELECT member_id,  
        points_balance,  
        started_date,  
        ended_date,  
        registered_date,  
        firstname,  
        lastname,  
        gender,  
        age,  
        email  
from member;
```

#### 4.3.3 Check that the inserts were correct

Run the statements below and check that the POINTS\_BALANCE field in gold\_member is greater than or equal to 5,000,000, and that it is less than 5,000,000 for club\_member.

```
SELECT * FROM gold_member  
LIMIT 10;  
  
SELECT * FROM club_member  
LIMIT 10;
```

## 4.4 Using ALTER TABLE SWAP WITH to swap table content and metadata

ALTER TABLE SWAP WITH swaps all content and metadata between two specified tables, including any integrity constraints defined for the tables. The two tables are essentially renamed in a single transaction.

You'll practice using SWAP WITH in this section. You're going to truncate the gold\_member and club\_member tables from the previous exercise, insert the data for the gold\_member table into the club\_member table and vice versa, then you'll swap the tables to correct the problem.

## 4.4.1 Truncate the tables and replace the data

```
TRUNCATE TABLE gold_member;
TRUNCATE TABLE club_member;

INSERT ALL
  WHEN points_balance < 5000000 THEN --inserts the club_member data into the
    gold_member table
    INTO gold_member(member_id,
      points_balance,
      started_date,
      ended_date,
      registered_date,
      firstname,
      lastname,
      gender,
      age,
      email)
  VALUES (member_id,
    points_balance,
    started_date,
    ended_date,
    registered_date,
    firstname,
    lastname,
    gender,
    age,
    email)
  ELSE -- inserts the gold_member data into the club_member table
    INTO club_member (member_id,
      points_balance,
      started_date,
      ended_date,
      registered_date,
      firstname,
      lastname,
      gender,
      age,
      email)
  VALUES (member_id,
    points_balance,
    started_date,
    ended_date,
    registered_date,
    firstname,
    lastname,
    gender,
    age,
    email)
  SELECT member_id,
    points_balance,
    started_date,
    ended_date,
    registered_date,
    firstname,
    lastname,
```

```
gender,  
age,  
email  
from member;
```

#### 4.4.2 Verify the data

Execute the statements below to verify the values in the points\_balance column:

```
SELECT * FROM gold_member  
LIMIT 10;  
  
SELECT * FROM club_member  
LIMIT 10;
```

Notice that the two tables have the wrong values for points\_balance. The gold\_member table should show values over 5,000,000, and the club\_member table should show lower values. Run a check to see how many rows are correct in each table. These two queries shouldn't return any rows since the multi-table insert was not correct.

```
SELECT * FROM gold_member WHERE points_balance >= 5000000;  
  
SELECT * FROM club_member WHERE points_balance < 5000000;
```

It is clear that the two tables are reversed: members with more than 5,000,000 points are in the club\_member table, and members with fewer points are in the gold\_member table. One solution for this would be to drop both tables and run the multi-table insert again. The easier solution is to use the ALTER TABLE SWAP WITH, which swaps the names and all meta data information on the two tables. Let's try that now.

#### 4.4.3 Execute the table swap below:

```
ALTER TABLE gold_member SWAP WITH club_member;
```

#### 4.4.4 Execute the statements below to see if the swap operation fixed the issue.

```
SELECT * FROM gold_member WHERE points_balance >= 5000000;  
  
SELECT * FROM club_member WHERE points_balance <= 5000000;
```

As you can see, the problem is now fixed.

### 4.5 Using MERGE to update rows in a table

In this section you're going to use MERGE to update data in two tables. In this scenario, SnowBear Air receives two files from their Members website showing changes that have been made by the member.



Since the Member table was split between the CLUB and GOLD members, we've receive two files from the web group. The first has changes for the club\_member table and the second has changes for the gold\_member table.

**MERGE**

MERGE can be used to insert, update, or delete values in a table based on values in a second table or a subquery. This can be useful if the second table is a change log that contains new rows (to be inserted), modified rows (to be updated), and/or marked rows (to be deleted) in the target table.

The command supports semantics for handling the following cases: - Values that match (for updates and deletes). - Values that do not match (for inserts).

**MERGE SYNTAX**

MERGE INTO USING ON ;

Example:

```
MERGE INTO target_table USING source_table ON target_table.id =
    source_table.id
    WHEN MATCHED THEN
        UPDATE SET target_table.description = source_table.description;
```

Note that the WHEN MATCHED THEN clause triggers the updating of one field with another. This allows updates to be merged into existing data

#### 4.5.1 Create temporary tables for the new data

In this step you'll create the tmp\_gold\_member\_change and tmp\_club\_member\_change table that will hold the changes for both member tables.

```
create or replace TEMP TABLE tmp_gold_member_change (
    member_id INTEGER,
    points_balance NUMBER,
    started_date DATE,
    ended_date DATE,
    registered_date DATE,
    firstname VARCHAR,
    lastname VARCHAR,
    gender VARCHAR,
    age NUMBER,
    email VARCHAR
);

create or replace TEMP TABLE tmp_club_member_change (
    member_id INTEGER,
    points_balance NUMBER,
    started_date DATE,
    ended_date DATE,
    registered_date DATE,
    firstname VARCHAR,
    lastname VARCHAR,
    gender VARCHAR,
    age NUMBER,
```

```

email VARCHAR
);

INSERT INTO tmp_gold_member_change (member_id, points_balance, started_date,
ended_date, registered_date, firstname, lastname, gender, age, email)
values
  (NULL,5000000,current_date(),NULL,current_date(),'Jessie','James',
    'M',64,'jjames@outlaw.com'),
  (NULL,5000000,current_date(),NULL,current_date(),'Kyle','Benton',
    'M',39,'kbenton@companyx.com'),
  (NULL,5000000,current_date(),NULL,current_date(),'Charles','Xavier',
    'M',76,'ProfessorX@Xmen.com'),
  (6,7630775,'2012-02-28','2014-04-14','2015-12-28','Anna-diana','Gookey',
    'F',29,'agookey5@hhs.gov'),
  (7,5128459,'2017-02-01',NULL,'2019-07-08','Damara','Kilfeder',
    'F',85,'dkilfeder6@scribd.com'),
  (34,9287918,'2018-12-13',NULL,'2018-03-24','Igor','Danell',
    'M',64,'idanellx@facebook.com'),
  (67,7684309,'2014-05-24',NULL,'2018-06-25','Ky','Bree',
    'M',39,'kbree1u@wikia.com'),
  (107,5221084,'2018-05-22',current_date(),'2016-03-07','Persis','Keri',
    'F',76,'pkeri2y@soundcloud.com'),
  (172,6720892,'2020-03-28',NULL,'2014-03-05','Jessalyn','Smith',
    'F',27,'jgilberthorpe4r@bbc.co.uk'),
  (177,9175745,'2012-12-22',NULL,'2012-08-02','Giacomo','Careswell',
    'M',63,'gcareswell4w@comsenz.com'),
  (236,8372164,'2016-12-22',current_date(),'2017-05-02','Guendolen',
    'Girdlestone','F',38,'ggirdlestone6j@nationalgeographic.com'),
  (426,6051750,'2018-05-06',NULL,'2020-06-28','Marietta','Busfield',
    'M',71,'mbusfielddb@wordview.com'),
  (431,9323224,'2013-01-08',NULL,'2015-05-19','Malcolm','Eastes',
    'M',39,'meastesby@lulu.com'),
  (437,6917699,'2015-01-02',NULL,'2012-09-18','Fremont','Rizzardo',
    'M',64,'frizzardoc4@biglobe.ne.jp'),
  (453,6547799,'2012-08-27',NULL,'2011-01-01','Roselia','McMillen',
    'F',51,'rtaptonck@cdc.gov'),
  (531,6361513,'2010-11-16',NULL,'2019-03-26','Sally','O Duilleain',
    'F',76,'hoduilleaineq@printfriendly.com');

INSERT INTO tmp_club_member_change (member_id, points_balance, started_date,
ended_date, registered_date, firstname, lastname, gender, age, email)
values
  (NULL,0,current_date(),NULL,'2014-06-02','Ted','Bundy',
    'M',45,'tbundy@meetup.com'),
  (NULL,0,current_date(),NULL,'2015-04-15','Jimmy','Hoffa',
    'M',55,'jhoffa@narod.ru'),
  (NULL,0,current_date(),NULL,'2013-01-15','Mary','Manners',
    'F',37,'mmanners@ibm.com'),
  (NULL,0,current_date(),NULL,'2017-05-25','Nancy','Dew',
    'F',39,'NDew3@wsj.com'),
  (5,806553,'2017-12-15',NULL,'2016-06-16','Jessey','Cotherill',
    'M',37,'jcotherill4@indiegogo.com'),
  (8,1914198,'2012-10-08','2020-08-12','2013-11-14','Robinetta','Slayford',
    'F',33,'rslayford7@prnewswire.com'),
  (9,3527720,'2019-05-30','2020-09-22','2015-01-07','Leonidas','Weatherby',
    'M',35,'lweatherby8@gnu.org'),

```

```
(10,678532,'2016-07-13','2020-12-1','2013-10-28','Wald','Simmank',
      'M',28,'wsimmank9@youku.com'),
(49,4182743,'2019-07-21',NULL,'2017-09-23','Tomi','Mayweather',
      'F',71,'tgloster1c@nymag.com'),
(51,2164969,'2012-07-29',NULL,'2011-11-11','Haleigh','Blackway',
      'M',42,'hblackway1e@hilton.com'),
(86,63441,'2012-06-21',NULL,'2018-03-05','Dniren','West',
      'F',67,'dnorth2d@dyndns.org'),
(102,1273020,'2019-07-03',NULL,'2016-04-30','Diandra','Peacham',
      'F',54,'dpeacham2t@.com'),
(143,198814,'2020-01-02',NULL,'2016-09-28','Alayne','Jevons',
      'F',49,'ajevons3y@nytimes.edu'),
(214,3713155,'2020-06-24',current_date(),'2011-10-21','Licha','MacCurlye',
      'F',62,'lmaccurlye5x@microsoft.it'),
(221,3642431,'2020-08-21',NULL,'2015-05-19','Codi','Battram',
      'M',32,'cbattram@ft.com');
```

#### 4.5.2 Apply the MERGE statement

Now you'll use a MERGE statement to apply the updates to the gold\_member table. After the MERGE statement is run, you will run some queries to verify the changes.

```
MERGE INTO gold_member gm USING tmp_gold_member_change gc ON gm.member_id=
gc.member_id
  WHEN matched
    THEN UPDATE SET points_balance = gc.points_balance,
                    started_date = gc.started_date,
                    ended_date = gc.ended_date,
                    registered_date = gc.registered_date,
                    firstname = gc.firstname,
                    lastname = gc.lastname,
                    gender = gc.gender,
                    age = gc.age
  WHEN NOT MATCHED THEN INSERT ( points_balance, started_date, ended_date,
                                registered_date, firstname, lastname, gender, age, email)
    VALUES (gc.points_balance,
            gc.started_date,
            gc.ended_date,
            gc.registered_date,
            gc.firstname,
            gc.lastname,
            gc.gender,
            gc.age,
            gc.email);
```

#### 4.5.3 Save the query\_id from the merge statement. This will be used to show the what has changed in the gold\_members table with the merge statement.

```
SET merge_query_id = last_query_id();
SHOW VARIABLES;
```

#### 4.5.4 Verify the effect of the MERGE statement

The following queries use time travel to view the state of the tables before and after the MERGE statement.

```
-- Use Time Travel to show the rows that have been inserted and updated.

-- First show the 13 items that were updated in the gold_member table.

SELECT m.member_id,
       m.points_balance, mc.points_balance,
       m.started_date, mc.started_date,
       m.ended_date, mc.ended_date,
       m.registered_date, mc.registered_date,
       m.firstname, mc.firstname,
       m.lastname, mc.lastname,
       m.gender, mc.gender,
       m.age, mc.age,
       m.email, mc.email
FROM gold_member m INNER JOIN gold_member BEFORE (STATEMENT =>
      $merge_query_id) mc ON m.member_id = mc.member_id
WHERE mc.member_id IN (SELECT member_id FROM tmp_gold_member_change);
```

#### 4.5.5 Run the statement below to show the 3 items there were inserted into the gold\_member table.

```
SELECT m.member_id,
       m.points_balance,
       m.started_date,
       m.ended_date,
       m.registered_date,
       m.firstname,
       m.lastname,
       m.gender,
       m.age,
       m.email
FROM gold_member m
WHERE m.member_id NOT IN (SELECT member_id FROM gold_member BEFORE (STATEMENT
=> $merge_query_id));
```

#### 4.5.6 Now execute the same process with the club\_member table

```
--Execute the merge statement for the club_member table
MERGE INTO club_member cm USING tmp_club_member_change cc ON cm.member_id=
  cc.member_id
  WHEN matched
    THEN UPDATE SET points_balance = cc.points_balance,
                    started_date = cc.started_date,
                    ended_date = cc.ended_date,
                    registered_date = cc.registered_date,
                    firstname = cc.firstname,
                    lastname = cc.lastname,
                    gender = cc.gender,
                    age = cc.age
```

```

    WHEN NOT MATCHED THEN INSERT ( points_balance, started_date, ended_date,
    registered_date, firstname, lastname, gender, age, email)
        VALUES (cc.points_balance,
        cc.started_date,
        cc.ended_date,
        cc.registered_date,
        cc.firstname,
        cc.lastname,
        cc.gender,
        cc.age,
        cc.email);

-- Save the query_id from the merge statement. This will be used to show the what
-- has changed in the club_member table with the merge statement.

SET merge_query_id = last_query_id();
SHOW VARIABLES;

-- Show the items that were updated.
SELECT m.member_id,
       m.points_balance, mc.points_balance,
       m.started_date, mc.started_date,
       m.ended_date, mc.ended_date,
       m.registered_date, mc.registered_date,
       m.firstname, mc.firstname,
       m.lastname, mc.lastname,
       m.gender, mc.gender,
       m.age, mc.age,
       m.email, mc.email
FROM club_member m INNER JOIN club_member BEFORE (STATEMENT =>
$merge_query_id) mc ON m.member_id = mc.member_id
WHERE mc.member_id IN (SELECT member_id FROM tmp_club_member_change);

-- Show the items that were inserted into the member table.
SELECT m.member_id,
       m.points_balance,
       m.started_date,
       m.ended_date,
       m.registered_date,
       m.firstname,
       m.lastname,
       m.gender,
       m.age,
       m.email
FROM club_member m
WHERE m.member_id NOT IN (SELECT member_id FROM club_member BEFORE (STATEMENT
=> $merge_query_id));

```

As you can see, the update was successful.

## 4.6 Key Takeaways

- A single multi-insert statement can be used to insert data from one table into multiple tables.
- You can use ALTER TABLE SWAP WITH to easily swap content and metadata between two tables.

- You can use the MERGE statement to add, update or delete data in a table.
- You can use the query id of a SQL statement and time travel to compare what data looked like prior to and after an update.

## 5 Caching and Query Performance

The purpose of this lab is to introduce you to the three types of caching Snowflake employs and how you can use the Query Profile to determine if your query is making the best use of caching.

### Learning Objectives:

- How to access and navigate the Query Profile.
- The differences between metadata cache, query result cache and data cache.
- When and why the query result cache is being used or not.
- How to determine if partition pruning is efficient, and if not, how to improve it.
- How to determine if spillage is taking place.
- How to use EXPLAIN to determine how to improve your queries.

### Scenario:

Like traditional relational database management systems, Snowflake employs caching to help you get the query results you want as fast as possible. Snowflake caching is turned on by default and it works for you in the background without you having to do anything. However, if you aren't sure if you're writing queries that leverage caching in the most efficient way possible, you can use the Query Profile to determine how caching is impacting your queries.

Imagine that you're a data analyst at Snowbear Air and that you've just learned that Snowflake has different types of caching. You have been working on a few queries that you think could be running faster, but you're not sure. You've decided to become familiar with the Query Profile and plan to use it to see how caching is impacting your queries.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is select on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

### 5.1 Accessing and Navigating the Query Profile

In this section you'll learn how to access, navigate and use the Query Profile. This will prepare you for analyzing query performance and caching in this lab.

5.1.1 Create a new folder and call it Caching and Query Performance.

5.1.2 Create a new worksheet inside the folder and call it Working with Cache and the Query Profile.

5.1.3 Set the worksheet context as follows:

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

5.1.4 Create some data to query

```
CREATE TABLE customer AS
SELECT
    c_custkey
    , c_firstname
    , c_lastname
FROM
    SNOWBEARAIR_DB.PROMO_CATALOG_SALES.customer;
```

5.1.5 Execute the following simple query in your worksheet.

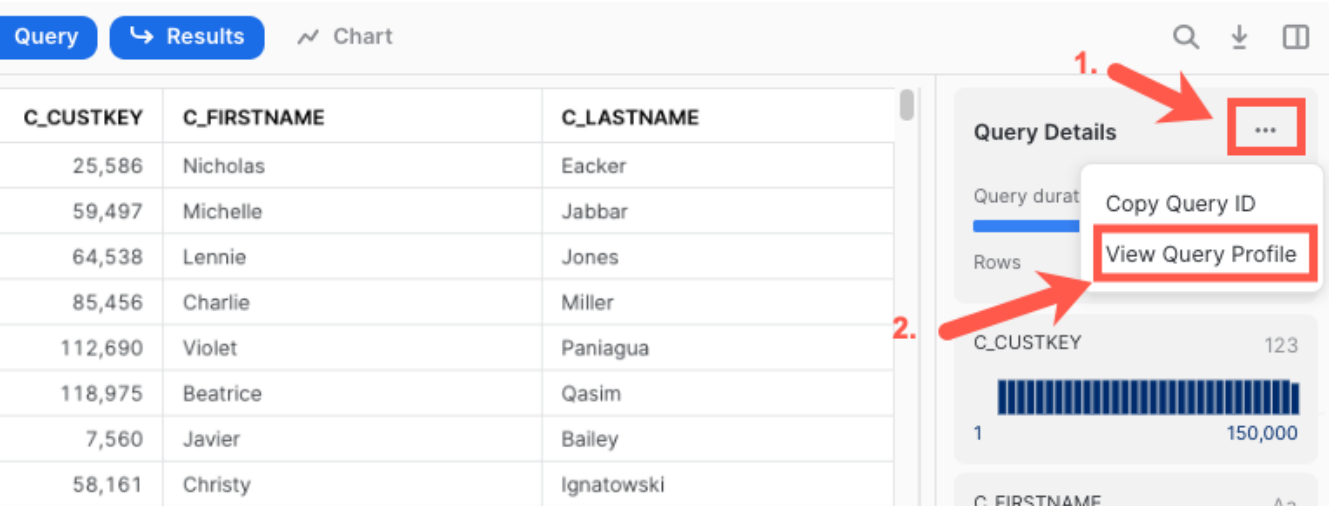
We need query results to view in the profile. Run the statement below.

```
SELECT DISTINCT
    *
FROM
    customer c;
```

5.1.6 Access the Query Profile

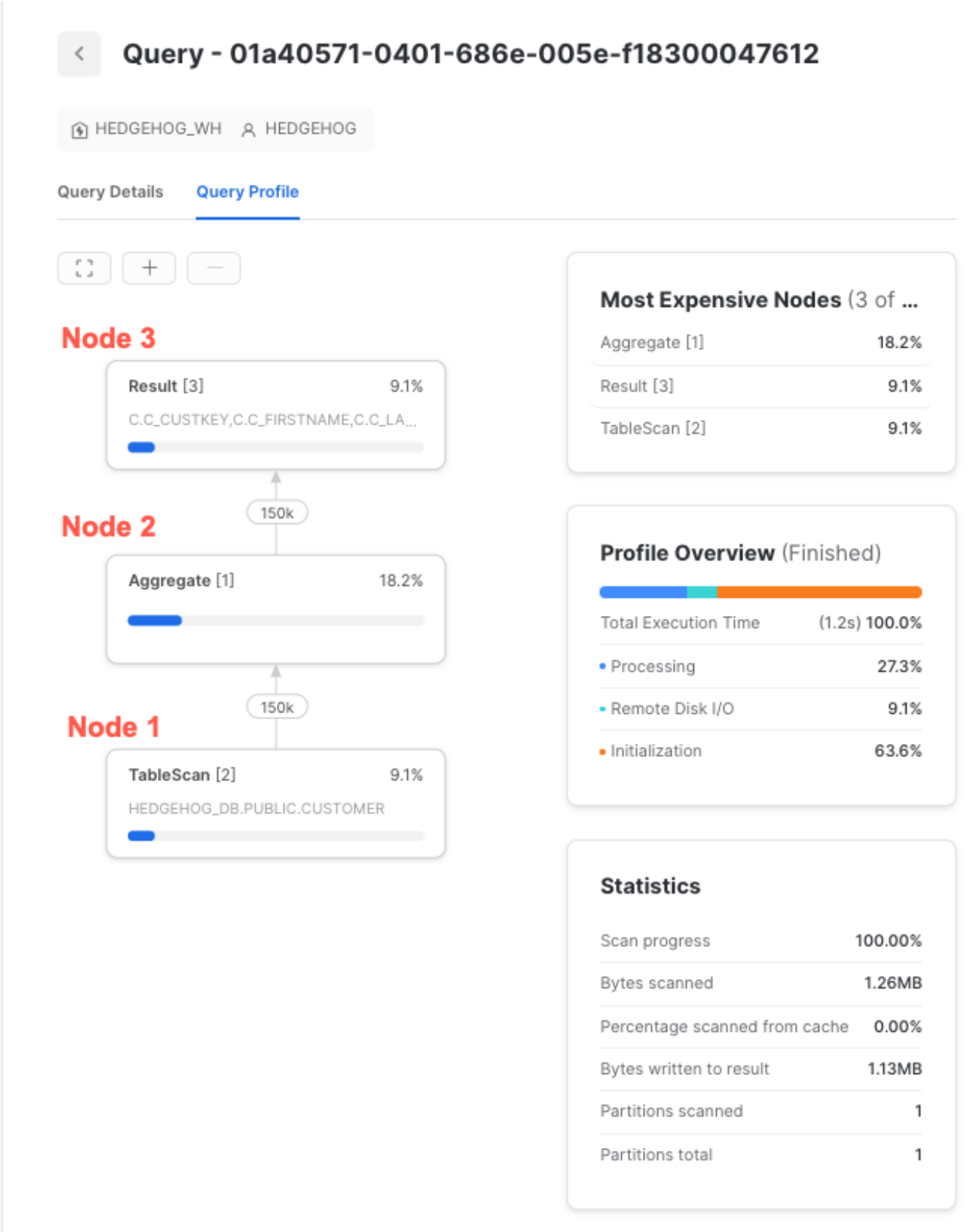
Now let's view the profile. To the right of the query results, you will see a panel saying 'Result limit exceeded...'. Click the 'X' in the upper right of this panel to show the usual Query Details panel. Click the ellipses shown in the screenshot. When the dialog box appears, click "View Query Profile". The Query Profile will open in a new tab.





**Figure 29:** Accessing the Query Profile

The query profile will appear as shown below:



Note that there are two tabs in the header of the screen: Query Details and Query Profile.

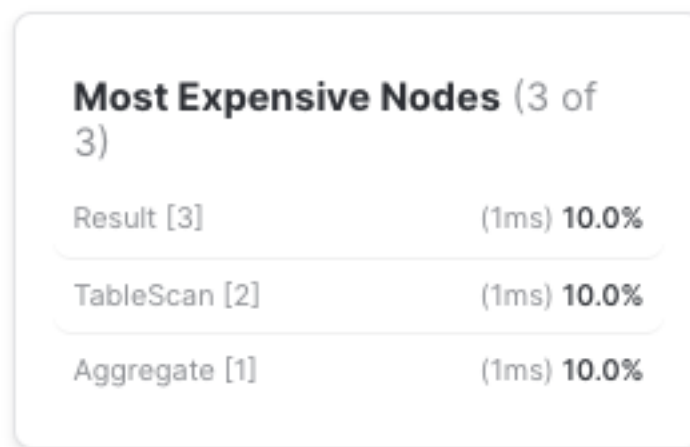
#### 5.1.7 Click the query details tab

- The Query Details panel shows the status of execution, the overall execution duration and other details. By looking at this you can see if the query succeeded and if it ran within the time frame you were hoping for.

#### 5.1.8 Click the query profile tab

Note that there are two panels in the query profile that show specific aspects of execution.

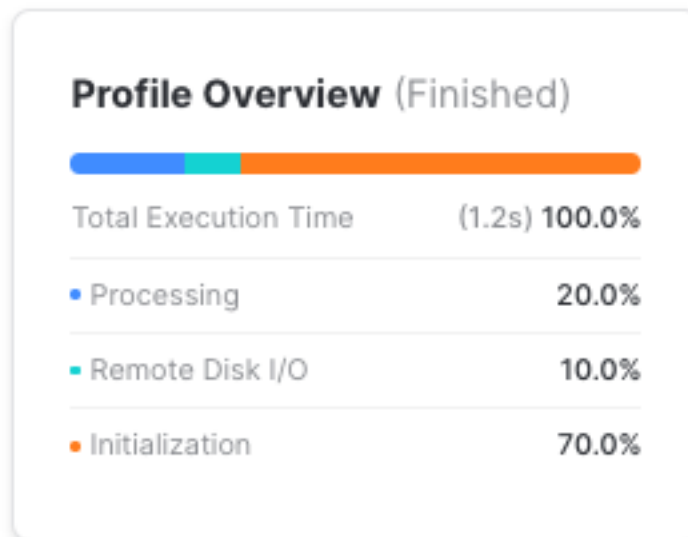
- The Most Expensive Nodes panel shows the nodes that took the longest to execute. In this case the three nodes all took the same amount of time, so they are all shown. But if one node took a long time to execute, this panel lets you identify and analyze it with the purpose of making it run more efficiently.



**Figure 31:** Most Expensive Nodes Panel

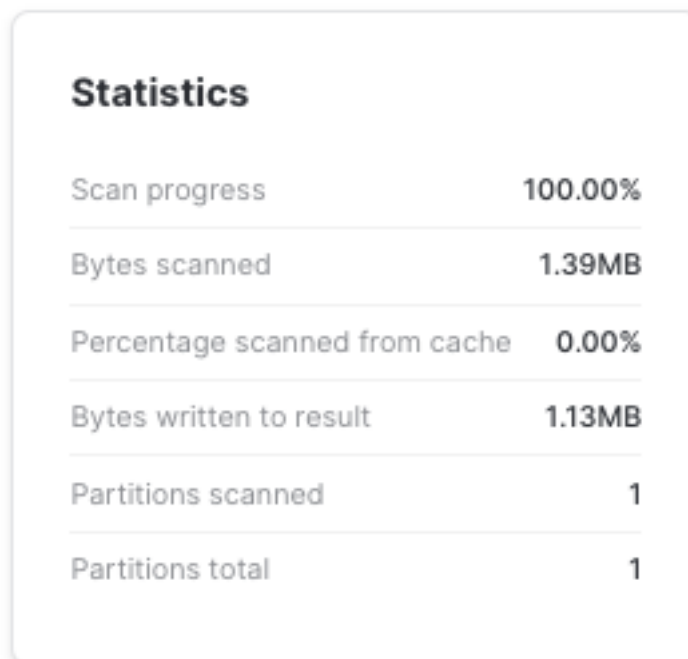
- The Profile Overview Panel displays the percentage of execution time that went towards the following:
  - Processing — The percentage of time spent on data processing by the CPU.
  - Remote Disk I/O — The percentage of time when the processing was blocked by remote disk access.
  - Initialization — The percentage of time spent setting up the query processing.

Note that 70% of the execution time was spent setting up the query (Initialization), 10% was blocked by remote disk access (Remote Disk I/O), and 20% was spent on processing data for the query by the CPU (Processing). Most of the execution time (which was just slightly over 1 second) was spent in Initialization due to the fact we were running a very simple query against a small amount of data. If we had been running a more complex query against more data and the execution time was not satisfactory, finding a high disk I/O would tell us we need to do something to reduce that, such as filtering.



**Figure 32:** Profile Overview Panel

- The Statistics panel shows values related to the scanning of partitions. The critical fields are the number of partitions scanned and the number of total partitions. These figures tell you whether partition pruning is efficient or not. If the partitions scanned are a far smaller number than the total partitions, you're getting good pruning and should be getting a satisfactory execution time. If the numbers are similar or identical in cases where you expect pruning to take place, you may need to revisit and revise your query.

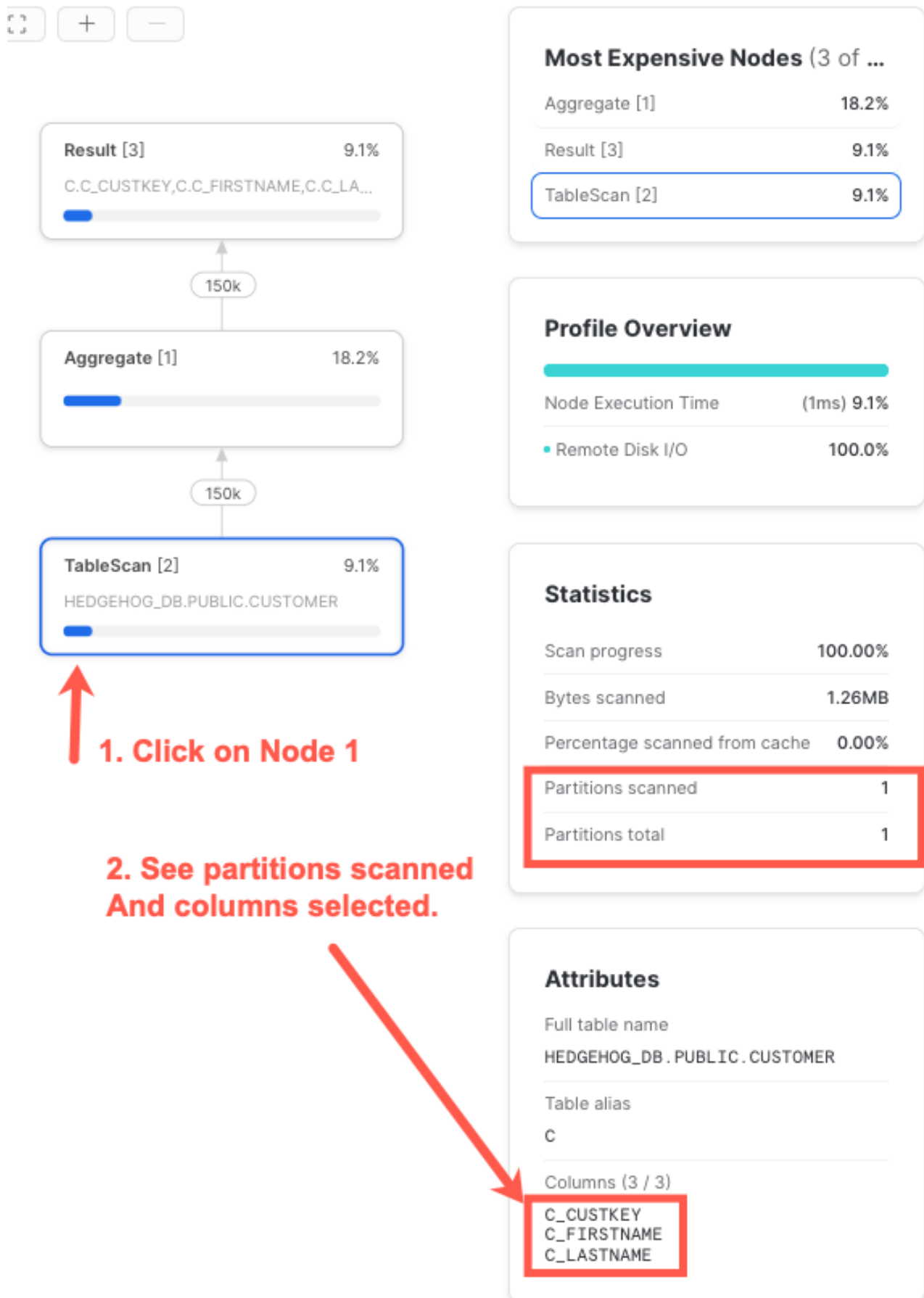


**Figure 33:** Statistics Panel

### 5.1.9 Click on Node 1 (TableScan[2])

This node shows statistics related to the scan of the customer table. As we saw in a previous step, the Statistics panel shows that one partition was scanned out of one total partitions evaluated. The query was very simple and the data set isn't very large, so this was to be expected.

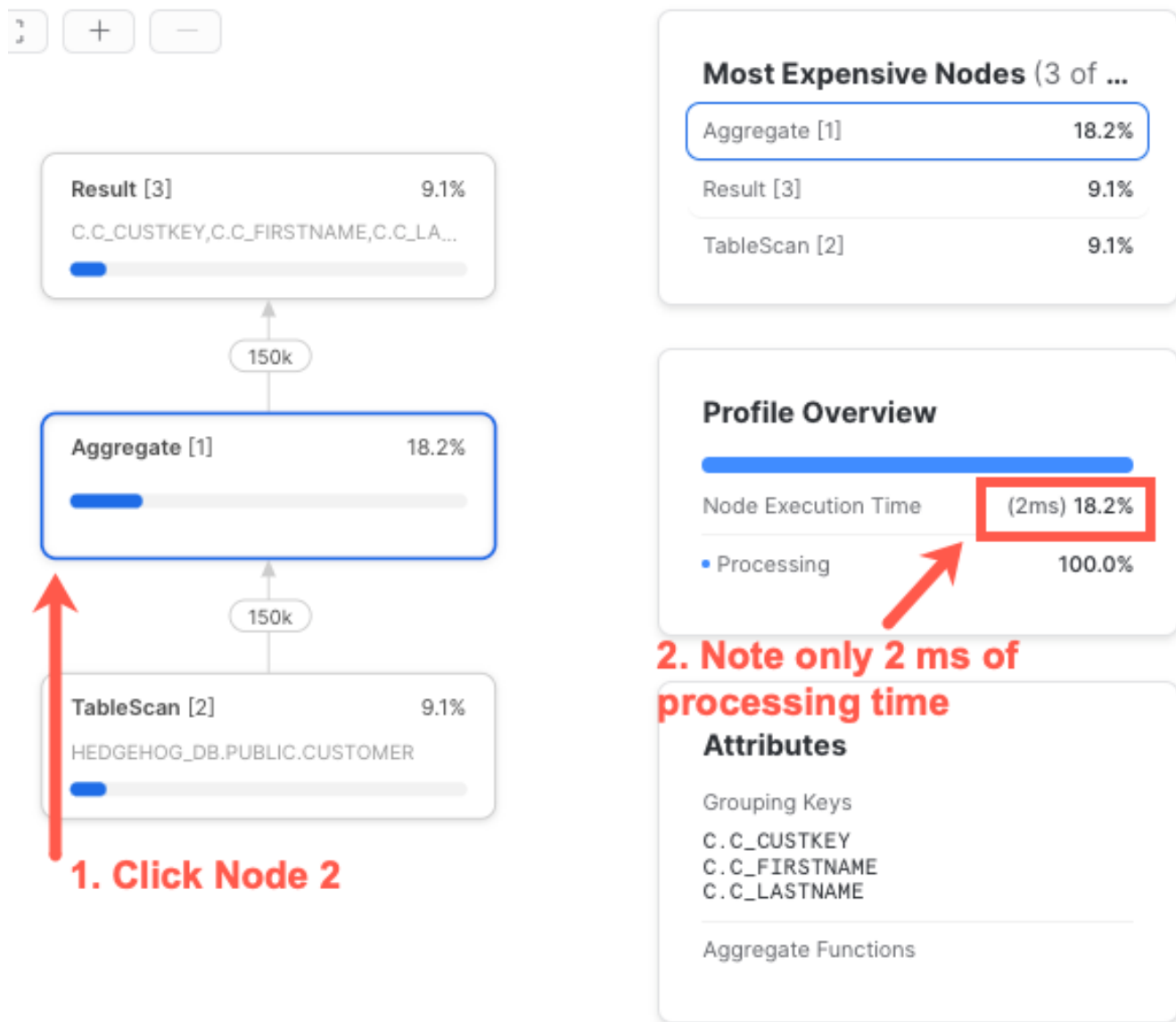
Note that an Attributes panel appears that shows the columns selected during processing of this node's query.



**Figure 34:** Node 1 Table Scan

## 5.1.10 Click on Node 2 (Aggregate[1])

This node shows the statistics related to the aggregation of the data.

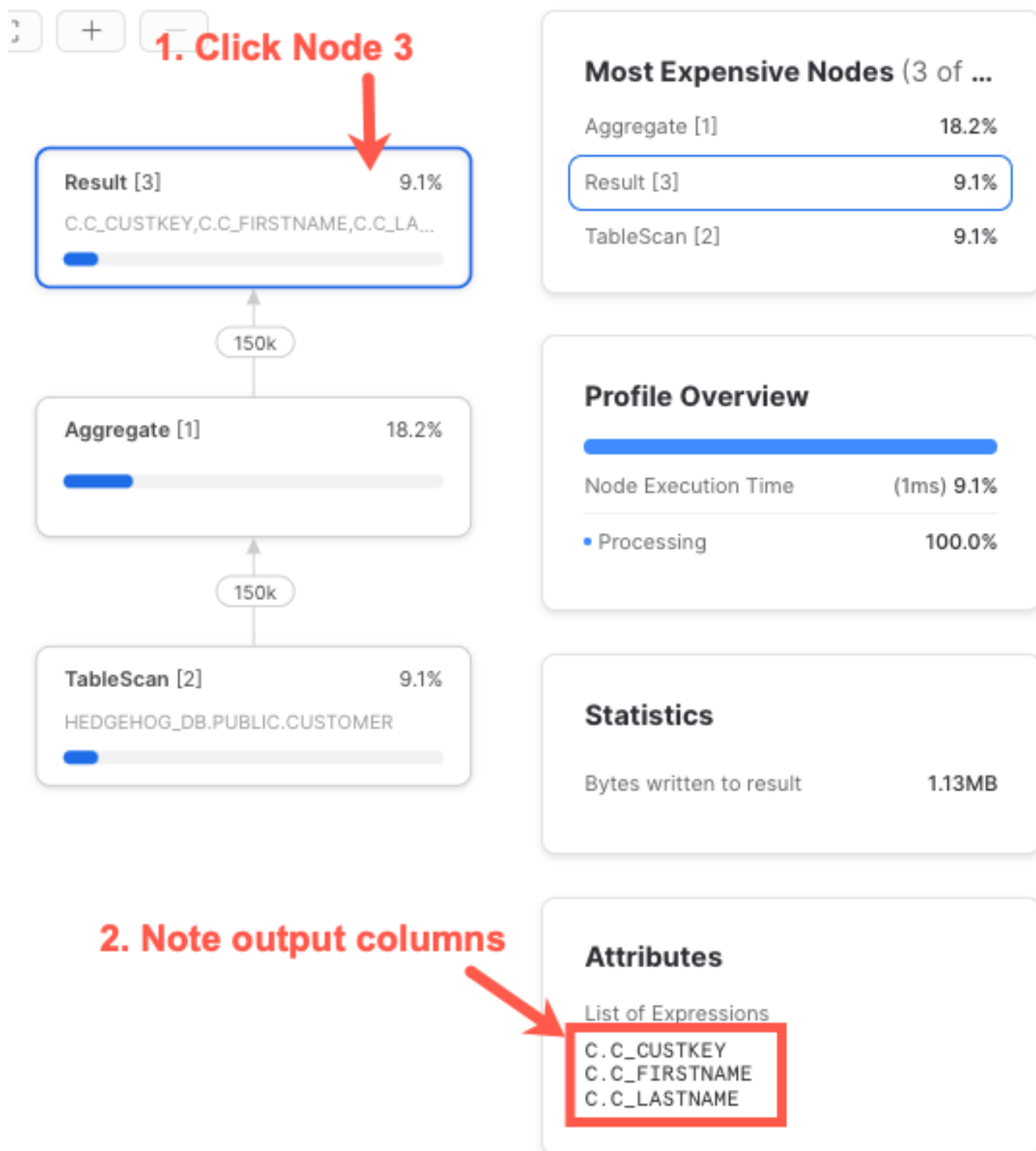


**Figure 35:** Node 2 - Aggregate

**NOTE:** You should also get similar results, but it's possible yours may differ slightly.

## 5.1.11 Click on Node 3 (Result[3])

This node shows the columns that were in the output.



**Figure 36:** Node 3 - Result

Note that an Attributes panel appears that shows the columns produced by the processing of this node.

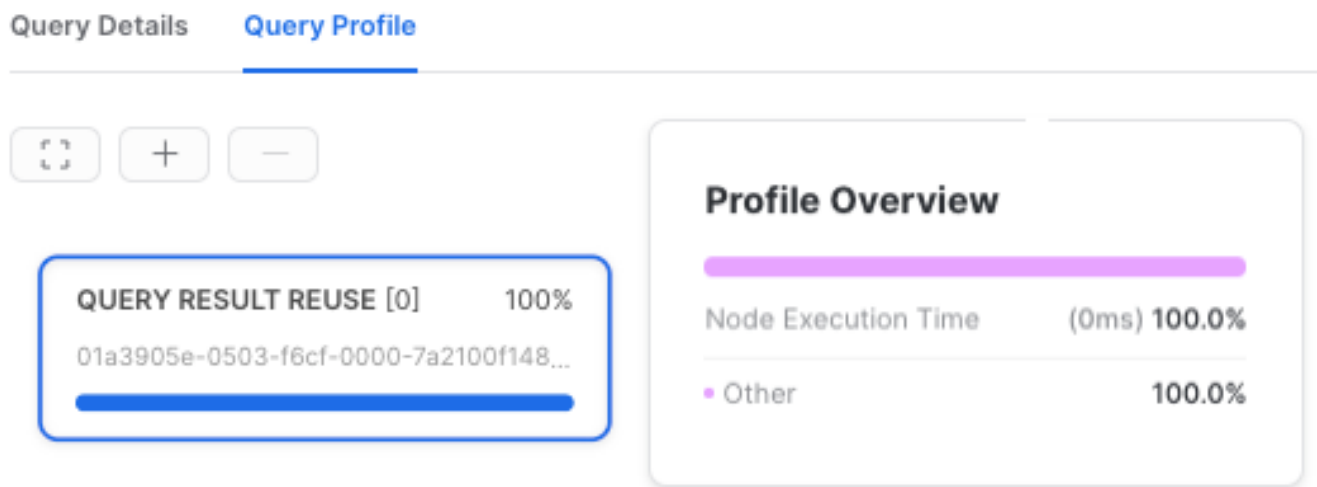


### 5.1.12 Close the tab

You'll remember that the Query Profile opened up in a new tab. While we could navigate back to our folder, that would leave us with two tabs open. Close this tab and return to the tab with your worksheet.

### 5.1.13 Go back to the tab with your worksheet and run your query again

Once you've run the query again, you should notice that it ran in a few milliseconds, much faster than before. Now let's look at the Query Profile again to see what happened.



**Figure 37:** Query Result Cache

As you can see, the query gave us the exact same result in just a few milliseconds because it was serving us the same results from the query result cache in the cloud services layer. This is a ready-to-go feature that you don't have to think about. You can just run your query a second time and get the same results again if needed.

## 5.2 Metadata Cache

Metadata cache is simple. When data is written into Snowflake partitions, the MAX, MIN, COUNT and other values are stored in the metadata cache in the Cloud Services layer. This means that when your query needs these values, rather than scanning the table and calculating the values, it simply pulls them from the metadata cache. This makes your query run much faster. Let's try it out!

### 5.2.1 Scenario

Let's imagine you've been asked to analyze part and supplier data. We're going to use the PARTSUPP table in our database called SNOWFLAKE\_SAMPLE\_DATA because it provides enough data for this exercise.

### 5.2.2 Set the context

Note we're setting `USE_CACHED_RESULT = FALSE` in order to avoid using the query result cache.

```
USE ROLE TRAINING_ROLE;  
USE WAREHOUSE [login]_WH;  
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCH_SF10;  
ALTER SESSION SET USE_CACHED_RESULT = FALSE;
```

### 5.2.3 Run the following SQL statement:

```
SELECT  
    MIN(ps_partkey)  
    , MAX(ps_partkey)  
  
FROM  
    PARTSUPP;
```

Now check the Query Profile. You should see a single node that says “METADATA-BASED RESULT”. This is because the query profile simply went to cache to get the data you needed rather than scanning a table. So, there was no disk I/O at all.

## 5.3 Data Warehouse Cache

Like any other database system, Snowflake caches data from queries you run so it can be accessed later by other queries. This cache is saved to disk in the warehouse. Let's take a look at how it works. Once again, let's assume you've been asked to analyze part and supplier data.

### 5.3.1 Clear your cache:

With the first statement below you will ensure the query result cache is disabled, so that you can focus on the data cache. With the remaining statements you'll suspend and restart your warehouse to ensure you're not using previously cached data.

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

### 5.3.2 Run the SQL statement below

Let's start by selecting two columns, `ps_suppkey` and `ps_availqty` with a `WHERE` clause that selects only part of the dataset. For any rows the query retrieves, this will cache the data for the two columns plus the column in the `WHERE` clause.

```
SELECT
    ps_partkey
    , ps_availqty
FROM
    PARTSUPP
WHERE
    ps_partkey > 1000000;
```

### 5.3.3 Look at Percentage scanned From Cache under Statistics in the Query Profile

You should see that the percentage scanned from cache is 0.00%. This is because we just ran the query for the first time on a newly resumed warehouse.

### 5.3.4 Run the query again

```
SELECT
    ps_partkey
    , ps_availqty
FROM
    PARTSUPP
WHERE
    ps_partkey > 1000000;
```

### 5.3.5 Look at percentage scanned from cache under Statistics in the Query Profile

You should see that the percentage scanned from cache is 100.00%. This is because the query was able to get 100% of the data it needed from the warehouse cache. This results in faster performance than a query that does a lot of disk I/O.

### 5.3.6 Add columns, run the query, and check the Query Profile

```
SELECT
    ps_partkey
    , ps_suppkey
    , ps_availqty
    , PS_supplycost
    , ps_comment
FROM
    PARTSUPP
WHERE
    ps_partkey > 1000000;
```

When you check the percentage scanned from cache, it should be less than 100%. This is because we added columns that weren't fetched previously, so some disk I/O must occur in order to fetch the data in those columns.

## 5.4 Partition Pruning

Partition pruning is a process by which Snowflake eliminates partitions from a table scan based on the partition's metadata. What this means is that fewer partitions are read from the storage layer, or are involved in filtering and joining, which gives you faster performance.

Data in Snowflake tables will be organized based on how the data is ingested. For example, if the data in a table has been organized based on a particular column, knowing which column that is and including it in joins or in WHERE clause predicates will result in more partitions being pruned, and thus, faster query performance.

This organization is called clustering. As the details related to clustering are not within the scope of this course, you can learn more by checking our documentation, or by taking the Snowflake Advanced course.

Let's look at an example. We're going to be using a different and larger dataset than our PROMO\_CATALOG\_SALES dataset so we can leverage partition pruning. Let's set the context, set our warehouse size to xsmall, and clear our data cache.

### 5.4.1 Set the context, warehouse size and clear your cache

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = 'XSMALL';
ALTER SESSION SET USE_CACHED_RESULT=FALSE;
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH RESUME;
```

Remember, if your warehouse is already suspended, either by timeout or command, you will receive the following error message upon execution of your SUSPEND statement: "Invalid state. Warehouse [login]\_WH cannot be suspended".

### 5.4.2 Execute a query with partition pruning

Let's imagine that the Snowbear Air marketing team has asked you for a list of customer addresses via a join on the CUSTOMER and CUSTOMER\_ADDRESS tables. The data in the CUSTOMER table has been organized based on C\_CUSTOMER\_SK, which is a unique identifier for each customer. The WHERE clause filters on both C\_CUSTOMER\_SK and on C\_LAST\_NAME. Execute the query below and check the Query Profile to see what happens.

```
SELECT
    C_CUSTOMER_SK
    , C_LAST_NAME
```

```

        , (CA_STREET_NUMBER || ' ' || CA_STREET_NAME) AS CUST_ADDRESS
        , CA_CITY
        , CA_STATE
FROM
    CUSTOMER
    INNER JOIN CUSTOMER_ADDRESS ON C_CUSTOMER_ID = CA_ADDRESS_ID
WHERE
    C_CUSTOMER_SK between 100000 and 600000
    AND
    C_LAST_NAME LIKE 'Johnson'
ORDER BY
    CA_CITY
    , CA_STATE;

```

If you check the nodes for each table, you'll see that the CUSTOMER and CUSTOMER\_ADDRESS tables have just over 500 total partitions between them.

The Query Profile tells us that the query ran in a few seconds and only about half of the partitions were scanned. So this query ran faster than it would have otherwise because partition pruning worked for us.

Now let's run a query without the C\_CUSTOMER\_SK field in the WHERE clause predicate and see what happens.

#### 5.4.3 Execute a query without partition pruning and check the Query Profile

```

SELECT
    C_CUSTOMER_SK
    , C_LAST_NAME
    , (CA_STREET_NUMBER || ' ' || CA_STREET_NAME) AS CUST_ADDRESS
    , CA_CITY
    , CA_STATE
FROM
    CUSTOMER
    INNER JOIN CUSTOMER_ADDRESS ON C_CUSTOMER_ID = CA_ADDRESS_ID
WHERE
    C_LAST_NAME = 'Johnson'
ORDER BY
    CA_CITY
    , CA_STATE;

```

The Query Profile tells us that this query took longer to run and all partitions were scanned. This is because the data in the CUSTOMER table is not organized on the C\_LAST\_NAME column, so more partitions had to be scanned in order for us to get our query result. The takeaway here is that understanding how your table's data is organized can help you write more efficient queries. Your DBA (Snowflake Administrator) is someone who can help you find that information.

## 5.5 Determine If Spillage Is Taking Place

Now let's determine if spillage is taking place in one of our queries. Spillage means that because an operation cannot fit completely in memory, data is spilled to disk within the warehouse. Operations that incur spillage

are slower than memory access and can greatly slow down query execution. Thus, you may need to be able to identify and rectify spillage.

Let's imagine that Snowbear Air wants to determine the average list price, average sales price and average quantity for both male and female buyers in the year 2000 for the months January through October.

Rather than use our PROMO\_CATALOG\_SALES database for this scenario, we're going to use another database that has enough data to create spillage. The structure and content of the data is less important than the fact that we can generate and resolve a spillage issue.

### 5.5.1 Set the context and resize the warehouse

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = 'XSMALL';
```

### 5.5.2 Clear the cache

Run the SQL statements below to set USE\_CACHED\_RESULT to false, and suspend and resume your warehouse to clear any cache. This will ensure we get spillage when we run our query.

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

### 5.5.3 Run a query that generates spillage

Note that the query below has a nested query. The nested query determines the average list price, average sales price and average quantity per gender type and order number. The outer query then aggregates those values by gender.

Run the query below. It should take around 3 to 5 minutes to run (your results may vary).

```
SELECT
    cd_gender
  , AVG(lp) average_list_price
  , AVG(sp) average_sales_price
  , AVG(qu) average_quantity
FROM
    (
        SELECT
            cd_gender
            , cs_order_number
            , AVG(cs_list_price) lp
            , AVG(cs_sales_price) sp
            , AVG(cs_quantity) qu
        FROM
```

```
        catalog_sales
      , date_dim
      , customer_demographics
WHERE
    cs_sold_date_sk = d_date_sk
AND
    cs_bill_cdemo_sk = cd_demo_sk
AND
    d_year IN (2000)
AND
    d_moy IN (1,2,3,4,5,6,7,8,9,10)
GROUP BY
    cd_gender
    , cs_order_number
) inner_query
GROUP BY
    cd_gender;
```

#### 5.5.4 View the results

For female buyers you should see something very similar to the following figures:

- average\_list\_price: 100.995691505527
- average\_sales\_price: 50.494185840967
- average\_quantity: 50.497579311044

For male buyers you should see something very similar to the following figures:

- average\_list\_price: 100.992076976143
- average\_sales\_price: 50.491772005658
- average\_quantity: 50.499846880459

#### 5.5.5 Check out the Query Profile

Go to the Query Profile. As you will see at the bottom of the Statistics, gigabytes of data were spilled to local storage. Notice there are two Aggregate nodes in the query. Click on each and notice the first Aggregate node is where the spillage is happening. This node is part of the inner query. Let's rectify this issue by rewriting our query.

If you look back at the query you just ran, you'll see that the outer query is not really necessary. All you need to do is remove the `cs_order_number` column from the nested query and then run it.

#### 5.5.6 Run the modified nested query

Let's run the query. We'll suspend the warehouse first to flush any cache so we can get a true reading of how long it will take for the query to run.

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

SELECT
    cd_gender
  , AVG(cs_list_price) lp
  , AVG(cs_sales_price) sp
  , AVG(cs_quantity) qu
FROM
    catalog_sales
  , date_dim
  , customer_demographics
WHERE
    cs_sold_date_sk = d_date_sk
  AND
    cs_bill_cdemo_sk = cd_demo_sk
  AND
    d_year IN (2000)
  AND
    d_moy IN (1,2,3,4,5,6,7,8,9,10)
GROUP BY
    cd_gender;
```

#### 5.5.7 Check your results

The query should have run in 1-3 minutes (your results may vary). Compare your results to the ones you got previously. They may be slightly different past the hundreds place to the right of the decimal, but that is due to the differences in rounding between the original query and the modified nested query. So, in essence you got the same results only in far less time.

#### 5.5.8 Check the Query Profile

As you will see at the bottom of the Statistics, there is no longer a spillage entry. This means that you resolved your spillage issue by simply rewriting your query so that it was more efficient.

### 5.6 Review the EXPLAIN Plan

Now let's compare the EXPLAIN plans from both of the queries we just ran in order to see how they are different.

#### 5.6.1 Use EXPLAIN to see the plan for the first query

```
EXPLAIN
SELECT
    cd_gender
  , AVG(lp) average_list_price
  , AVG(sp) average_sales_price
  , AVG(qu) average_quantity
```



```

FROM
    (
        SELECT
            cd_gender
            , cs_order_number
            , AVG(cs_list_price) lp
            , AVG(cs_sales_price) sp
            , AVG(cs_quantity) qu
        FROM
            catalog_sales
            , date_dim
            , customer_demographics
        WHERE
            cs_sold_date_sk = d_date_sk
            AND
            cs_bill_cdemo_sk = cd_demo_sk
            AND
            d_year IN (2000)
            AND
            d_moy IN (1,2,3,4,5,6,7,8,9,10)

        GROUP BY
            cd_gender
            , cs_order_number
    ) inner_query
GROUP BY
    cd_gender;

```

### 5.6.2 Click on the Operation header to sort the rows

Note that there are 12 rows that correspond to the execution nodes that you would see in the Query Profile. Also note that two of the rows are aggregate rows. The node below executes the averaging of the list price, sales price and quantity:

```

aggExprs: [SUM((SUM(CATALOG_SALES.CS_LIST_PRICE)) / (COUNT(CATALOG_SALES.CS_LIST_PRICE)))
, COUNT((SUM(CATALOG_SALES.CS_LIST_PRICE)) / (COUNT(CATALOG_SALES.CS_LIST_PRICE)))
, SUM((SUM(CATALOG_SALES.CS_SALES_PRICE)) / (COUNT(CATALOG_SALES.CS_SALES_PRICE)))
, COUNT((SUM(CATALOG_SALES.CS_SALES_PRICE)) / (COUNT(CATALOG_SALES.CS_SALES_PRICE)))
, SUM((SUM(CATALOG_SALES.CS_QUANTITY)) / (COUNT(CATALOG_SALES.CS_QUANTITY)))
, COUNT((SUM(CATALOG_SALES.CS_QUANTITY)) / (COUNT(CATALOG_SALES.CS_QUANTITY)))]
, groupKeys: [CUSTOMER_DEMOGRAPHICS.CD_GENDER]

```

**Figure 38:** Aggregate Row Expression

### 5.6.3 Run the EXPLAIN statement for the second query

```

EXPLAIN
SELECT
    cd_gender
    , AVG(cs_list_price) lp

```

```
    , AVG(cs_sales_price) sp
    , AVG(cs_quantity) qu
FROM
    catalog_sales
    , date_dim
    , customer_demographics
WHERE
    cs_sold_date_sk = d_date_sk
    AND
    cs_bill_cdemo_sk = cd_demo_sk
    AND
    d_year IN (2000)
    AND
    d_moy IN (1,2,3,4,5,6,7,8,9,10)
GROUP BY
    cd_gender;
```

Notice now that this plan is identical to the first one except that there is one aggregate row fewer than in the previous explain plan (for a total of 11 rows). Specifically, the node shown in the previous step in this lab is the one that is gone because we removed the outer query. Making that change alone was enough to cut query time by more than half.

#### 5.6.4 Change your warehouse size to XSmall

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

## 5.7 Summary

Writing efficient queries is an art that takes a solid understanding of how Snowflake caching and query pruning impact query performance. While it's impossible to show you every single scenario, you should know that getting proficient at using tools like the Query Profile and the EXPLAIN plan will help you better understand how caching impacts your query performance. This in turn will allow you to write better queries that achieve a shorter run time.

## 5.8 Key takeaways

- Snowflake employs caching to help you get the query results you want as fast as possible.
- Cache is turned on by default and it works for you in the background without you having to do anything.
- The Query Profile is a useful tool for understanding how caching and partition pruning are impacting your queries.
- As you add or remove columns to/from a SELECT clause or a WHERE clause, your percentage scanned from cache value could go up or down.

- If your query only requests MIN or MAX values on INTEGER, DATE or DATETIME data types, those values will come from metadata cache in the Cloud Services layer rather than from disk I/O, which results in fast performance.
- Query Result cache is invoked when you run the exact same query twice.
- Data cache resides in the warehouse and it stores data from past queries on a least recently used (LRU) basis, until the warehouse is suspended. However, once the warehouse is suspended, its data cache is cleared out.
- Including the column on which a table's data is organized in a WHERE clause predicate can improve partition pruning, which in turn improves performance.
- Using EXPLAIN can give you insight into how Snowflake will execute your query. You can use it to identify and remove bottlenecks in your query so you can resolve them and get better efficiency.

## 6 Loading Structured Data

The purpose of this lab is to introduce you to data loading and data unloading.

### Learning Objectives:

- How to load data from a file in an external stage into a table using the COPY INTO command.
- How to define a GZIP file format.
- How to review the properties of a stage.
- How to load a GZipped file from an external stage into a table.
- How to validate data prior to loading
- How to handle data loading errors

### Scenario:

You are a data engineer at Snowbear Air. You need to create and populate a table that will be used in reporting. The table will be called REGION and you will populate it from a pre-existing file (`region.tbl`) in an external stage. The file is headerless, pipe-delimited and contains five rows.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

Let's get started!

### 6.1 Loading Data from an External Stage into a Table Using COPY INTO

In this exercise you will learn how to load a file from an external stage into a table using the COPY INTO command.

6.1.1 Create a new folder called Data Loading

6.1.2 Create a new worksheet named *Load Structured Data*.

6.1.3 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;
```

6.1.4 Create a REGION table. This table will be loaded from a source file:

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

6.1.5 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;
```

6.1.6 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 the 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 **region.tbl** 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 Loading a GZip Compressed File on an External Stage into a Table

The scenario for this activity is fundamentally the same as the previous activity. The difference is that you will load the REGION Table from a gzip compressed file that is in the external stage. You will use the MYGZIPPIPEFORMAT file format you created in the previous part of this lab.

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 confirm the data was successfully loaded:

```
SELECT * FROM region;
```

### 6.3.5 Suspend and resize the warehouse

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

## 6.4 Validating data prior to load

Now we're going to practice using the `VALIDATION_MODE` parameter of the `COPY INTO` statement to check for problems with the file prior to loading it.

### 6.4.1 Modify the NATION table by running the statement below.

```
CREATE OR REPLACE TABLE nation (  
    NATION_KEY INTEGER  
    , NATION VARCHAR  
    , REGION_KEY INTEGER  
    , COMMENTS VARCHAR  
);
```

### 6.4.2 Try to copy into the NATION Table by executing the COPY INTO statement below

Notice that the validation mode is `RETURN_ALL_ERRORS`. This will return any and all errors if there are any. Regardless, no data will be loaded. This is because by providing a value for `VALIDATION_MODE`, we are indicating we only want to check the file, not load the file.

```
COPY INTO NATION  
FROM @training_db.traininglab.ed_stage/load/lab_files/  
FILES = ('nation.tbl')  
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)  
VALIDATION_MODE = RETURN_ALL_ERRORS;
```

You should have gotten a message that says "Query produced no results". This means there were no errors and that you can load the table. But now we're going to recreate the table and switch the order of the columns. By making `REGION_KEY`, which is an integer column, the second column in the table, we will have errors because the second column in the file is a `VARCHAR` field.

### 6.4.3 Recreate the NATION table and execute the COPY INTO statement

Note that in this case we are using `RETURN_ERRORS`. Like `RETURN_ALL_ERRORS`, it will return any and all errors stemming from the loading of the file indicated in the `COPY INTO` statement below.

```
CREATE OR REPLACE TABLE nation (  
    NATION_KEY INTEGER  
    , REGION_KEY INTEGER  
    , NATION VARCHAR
```

```

        , COMMENTS VARCHAR
    );

COPY INTO NATION
FROM @training_db.traininglab.ed_stage/load/lab_files/
FILES = ('nation.tbl')
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
VALIDATION_MODE = RETURN_ERRORS;

```

Now you should have 25 rows indicating that the VARCHAR value we tried to load into the REGION column has created an error. Had we tried to load the file, none of the rows would have loaded.

#### 6.4.4 Check only the first 10 rows

In the next statement we set the VALIDATION\_MODE to RETURN\_10\_ROWS. So, our statement will only check the first ten rows and return the first error it encounters.

```

COPY INTO NATION
FROM @training_db.traininglab.ed_stage/load/lab_files/
FILES = ('nation.tbl')
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
VALIDATION_MODE = RETURN_10_ROWS;

```

As you can see, we have a message saying that the numeric value “ALGERIA” is not recognized.

## 6.5 Error Handling

Now we’re going to examine error handling options for data loading in Snowflake.

### 6.5.1 Recreate the table with all columns in the same order as they are in the nation.tbl file

```

CREATE OR REPLACE TABLE nation (
    NATION_KEY INTEGER
    , NATION VARCHAR
    , REGION_KEY INTEGER
    , COMMENTS VARCHAR
);

```

### 6.5.2 Verify the query

AS you will see in the query below, we’ve introduced an error in the third column. For all rows where the region key is 1, we’ve converted it to the VARCHAR value “America”. This will generate five errors when loading into an INTEGER column.

```

SELECT
    n.$1 AS N_KEY
    , n.$2 AS NATION

```



```

    , CASE
      WHEN n.$3 = 1 THEN 'AMERICA'
      ELSE n.$3
    END AS R_KEY
    , n.$4 AS COMMENTS
FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl (file_format =>
'MYPIPEFORMAT') n;

```

### 6.5.3 Attempt to load the data

Notice that we've set the ON\_ERROR parameter to continue. This means that all rows that don't generate an error will get loaded.

```

COPY INTO nation
FROM (
  SELECT
    n.$1 AS N_KEY
    , n.$2 AS NATION
    , CASE
      WHEN n.$3 = 1 THEN 'AMERICA'
      ELSE n.$3
    END AS R_KEY
    , n.$4 AS COMMENTS

    FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl
    (file_format => 'MYPIPEFORMAT') n
)
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
ON_ERROR = CONTINUE;

```

As you can see from the results, the status is PARTIALLY\_LOADED and rows\_loaded is 20 out of the original 25.

### 6.5.4 Run the SELECT statement below to verify the contents of the table, then truncate the table.

```

SELECT * FROM nation;

TRUNCATE TABLE nation;

```

### 6.5.5 Retry the insert with ON\_ERROR = ABORT\_STATEMENT

ABORT\_STATEMENT is the default value and it will cause the entire load to fail.

```

COPY INTO nation
FROM (
  SELECT
    n.$1 AS N_KEY
    , n.$2 AS NATION
    , CASE
      WHEN n.$3 = 1 THEN 'AMERICA'
      ELSE n.$3
    END AS R_KEY
    , n.$4 AS COMMENTS

```

```

        END AS R_KEY
    , n.$4 AS COMMENTS

    FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl
    (file_format => 'MYPIPEFORMAT') n
)
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
ON_ERROR = ABORT_STATEMENT; --ALSO THE DEFAULT OPTION WHEN ON_ERROR ISN'T SPECIFIED

```

As you can see, the error is “Numeric value ‘AMERICA’ is not recognized”. No data was loaded.

#### 6.5.6 Retry the insert with ON\_ERROR = SKIP\_FILE\_4

With this statement, we’re telling Snowflake that we want to load the file if we have four or more errors. Since we know our file will generate 5 errors, the load should fail completely.

```

COPY INTO nation
FROM (
    SELECT
        n.$1 AS N_KEY
    , n.$2 AS NATION
    , CASE
        WHEN n.$3 = 1 THEN 'AMERICA'
        ELSE n.$3
        END AS R_KEY
    , n.$4 AS COMMENTS

    FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl
    (file_format => 'MYPIPEFORMAT') n
)
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
ON_ERROR = SKIP_FILE_4;

```

As you can see, you got an error that says “LOAD\_FAILED”.

#### 6.5.7 Retry the insert with ON\_ERROR = SKIP\_FILE\_6

With this statement, we’re telling Snowflake that we want to load the file if we have six or more errors. Since we know our file will generate 5 errors, we should get a partial load.

```

COPY INTO nation
FROM (
    SELECT
        n.$1 AS N_KEY
    , n.$2 AS NATION
    , CASE
        WHEN n.$3 = 1 THEN 'AMERICA'
        ELSE n.$3
        END AS R_KEY
    , n.$4 AS COMMENTS

```

```
FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl
      (file_format => 'MYPIPEFORMAT') n
)
FILE_FORMAT = (FORMAT_NAME = MYPIPEFORMAT)
ON_ERROR = SKIP_FILE_6;
```

As you can see, the file was partially loaded. 20 out of 25 rows were loaded.

## 6.6 Key Takeaways

- Files to be loaded can be compressed or not compressed.
- The COPY INTO command can be used to load.
- You can use the LIST command to see what files are in a stage.
- The VALIDATION\_MODE parameter of the COPY INTO statement to check for problems with the file prior to loading it.
- You can set the ON\_ERROR parameter of the COPY INTO statement to load all, some or none of the data if one or more errors are detected.

## 7 Data Transformation During Data Loading

The purpose of this lab is to introduce you to how to transform data when loading it.

### Learning Objectives:

- How to transform data while loading it

### Scenario:

You are a data engineer at Snowbear Air. You need to populate a table with a list of nations and you need to provide a region name as well as a region code. But, the file doesn't have those two pieces of data. You will need to transform the data in the file prior to loading it so it has that data.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

### 7.1 Transforming Data During Load

7.1.1 Create a new folder called Data Loading and Transformation

7.1.2 Create a new worksheet named *Transforming Data During Load*.

7.1.3 Create the context

```
CREATE DATABASE IF NOT EXISTS [login]_db;  
CREATE SCHEMA IF NOT EXISTS [login]_transform;  
USE SCHEMA [login]_transform;  
USE SCHEMA [login]_db.[login]_transform;
```

7.1.4 Search for the nation file

Execute the statement below to list the files in the stage with a .tbl extension.

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

If you scroll down far enough, you will find nation.tbl. That is the file that you want to transform and load.

### 7.1.5 Query the file in the stage

Before you run the statement below, take a close look at the SELECT clause. Notice that there are three columns that start with the alias for the file (n) and are followed by a period, a dollar sign and a number. These refer to the first three columns in the file. You can refer to columns in the SELECT clause in this way whether or not they exist in the file. If they don't the entire column will simply be null.

```
SELECT n.$1, n.$2, n.$3
FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl n ;
```

Note that the first two columns have data while the third one is null. In fact, the first column is the one we want. The problem is that we want to transform the data prior to loading it into a table, and it's in a pipe-delimited format. So, we need to create a file format that will handle the data and let us query individual columns straight out of the file.

### 7.1.6 Create a pipe-delimited file format

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

### 7.1.7 Create the target table

We've been given a target table format. Let's create the table now:

```
CREATE OR REPLACE TABLE nation (
    NATION_KEY INTEGER
    , REGION VARCHAR
    , REGION_CODE VARCHAR
    , NATION VARCHAR
    , COMMENTS VARCHAR
);
```

### 7.1.8 Use the file format to query the file.

As you can see, we've added the file format right after the table name. We've also aliased all of the columns.

```
SELECT n.$1 AS N_KEY, n.$2 AS N_NAME, n.$3 AS R_KEY, n.$4 AS N_COMMENT
FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl (file_format =>
'MYPIPEFORMAT') n ;
```

N\_KEY can be the source for NATION\_KEY in our target table. However, we only have a region key, not a region name. Also, we need to provide a region code that consists of the first two letters of the region name. Finally,

N\_NAME can be the source for NATION. The problem is that it is the second column in the file, but it needs to go into the fourth column in the table.

Fortunately, Snowflake allows us to insert columns in a different order than they exist in the file. Also, we can use functions and CASE...WHEN statements to transform the data into the format we need it to be in. The query below does that for us.

#### 7.1.9 Execute the statement below

```
SELECT    n.$1 AS N_KEY
        , CASE
            WHEN n.$3 = 0 THEN 'AFRICA'
            WHEN n.$3 = 1 THEN 'AMERICA'
            WHEN n.$3 = 2 THEN 'ASIA'
            WHEN n.$3 = 3 THEN 'EUROPE'
            ELSE 'MIDDLE EAST'
          END AS REGION
        , substr(REGION, 1, 2) AS REGION_CODE
        , n.$2 AS NATION
        , n.$4 AS COMMENTS

FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl (file_format =>
'MYPIPEFORMAT') n ;
```

As you can see, the columns are now in the order and format that we need. Let's insert the data now.

#### 7.1.10 Insert the data

Execute the COPY INTO statement below and then check the results.

```
COPY INTO nation
FROM (
    SELECT
        n.$1 AS N_KEY
        , CASE
            WHEN n.$3 = 0 THEN 'AFRICA'
            WHEN n.$3 = 1 THEN 'AMERICA'
            WHEN n.$3 = 2 THEN 'ASIA'
            WHEN n.$3 = 3 THEN 'EUROPE'
            ELSE 'MIDDLE EAST'
          END AS REGION
        , substr(REGION, 1, 2) AS REGION_CODE
        , n.$2 AS NATION
        , n.$4 AS COMMENTS

    FROM @training_db.traininglab.ed_stage/load/lab_files/nation.tbl
    (file_format => 'MYPIPEFORMAT') n
);
```

In the result pane you should see a row that says status = "loaded" and rows\_loaded = "25". If you do, the load was successful.

7.1.11 Run the SELECT statement below to view the contents of the NATION table

```
SELECT * FROM nation;
```

You should have 25 rows of data in your table.

7.1.12 Suspend and resize the warehouse

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

## 7.2 Key Takeaways

- You can transform data prior to loading it.
- When transforming data, you can reorder the columns coming out of the file, and you can use (but are not limited to) functions or CASE...WHEN statements.

## 8 Unloading Structured Data

The purpose of this lab is to introduce you to data unloading.

### Learning Objectives:

- How to unload table data into a Table Stage in Pipe-Delimited File format
- Using an SQL statement containing a JOIN to Unload a Table into an internal stage

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

Let's get started!

### 8.1 Unloading table data into a Table Stage in Pipe-Delimited File format

This activity is essentially the opposite of the previous two activities. Rather than load a file into a table, you are going to take the data you loaded and unload it into a file in a table stage.

8.1.1 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;
```

8.1.2 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;
```

8.1.3 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 unloading. This will cause the unloaded file to be formatted according to the specifications of the MYPIPEFORMAT file format.



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

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

```
LIST @%region;
```

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

```
REMOVE @%region;
```

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

This activity is essentially the same as the previous activity. The difference is that you are going to unload data from more than one table into a single file.

8.2.1 Do a SELECT with a JOIN on the **REGION** and **NATION** tables. 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;
```

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

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

8.2.3 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);
```

8.2.4 Verify the file is in the stage:

```
LIST @mystage;
```

### 8.2.5 Remove the file from the stage:

```
REMOVE @mystage;
```

### 8.2.6 Remove the stage:

```
DROP STAGE mystage;
```

### 8.2.7 Suspend and resize the warehouse

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

## 8.3 Key Takeaways

- The COPY INTO command can be used to unload data.
- Data from multiple tables can be unloaded using a JOIN statement.
- You can use the LIST command to see what files are in a stage.

## 9 Introduction to Tasks

The purpose of this lab is to teach you to create and execute tasks and trees of tasks.

### Learning Objectives:

- How to create a task
- How to start and stop a task

### Scenario:

Imagine that there is a table that stores the average time from shipping to deliver for all orders in a specific month and year. Let's imagine that the order and shipping data is updated every minute and so you need to update the table every minute.

In this exercise you're going to write a task that updates the contents of the table every minute.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

**NOTE:** Tasks left running unsupervised can consume credits. In your training account, it is **IMPERATIVE** that you suspend any tasks you create. Failure to do so could result in all students being locked out of their 30-day training account.

Let's get started!

### 9.1 Creating the target table

First we'll set the context and create a new schema to use specifically for this lab.

9.1.1 Run the commands below to set the context.

```
USE ROLE TRAINING_ROLE;

CREATE DATABASE IF NOT EXISTS [login]_DB;
USE DATABASE [login]_DB;

CREATE WAREHOUSE IF NOT EXISTS [login]_WH;
USE WAREHOUSE [login]_WH;

CREATE SCHEMA IF NOT EXISTS [login]_TASKS_SCHEMA;
USE SCHEMA [login]_TASKS_SCHEMA;
```

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

9.1.2 Run the following command to create the table that will hold the average shipping time in days. Run the SELECT statement to confirm that it is empty.

```
CREATE OR REPLACE TABLE AVG_SHIPPING_IN_DAYS(
  yr INTEGER
, mon INTEGER
, avg_shipping_days DECIMAL(18,2)
);

SELECT * FROM avg_shipping_in_days;
```

As you can see, it is a simple table that stores the year, the month and the average shipping days in decimal format.

## 9.2 Creating the tasks

Now that we've created the tables to be populated, let's create the task tree.

9.2.1 Run the following command to create the task

This task will calculate and load the data from the orders and lineitem tables into the target table. Notice that we do an INSERT OVERWRITE. This essentially overwrites the table each time the task runs.

```
CREATE OR REPLACE TASK insert_shipping_by_date_rows
  WAREHOUSE = '[login]_WH'
  SCHEDULE = 'USING CRON 0-59 0-23 * * * America/Chicago'
AS
  INSERT OVERWRITE INTO [login]_DB.[login]_TASKS_SCHEMA.AVG_SHIPPING_IN_DAYS
    (yr, mon, avg_shipping_days)
  SELECT
    YEAR(F.O_ORDERDATE) AS YR
  , MONTH(F.O_ORDERDATE) AS MON
  , AVG (DAYS_TO_SHIP)::DECIMAL(18,2) AS AVG_DAYS_TO_SHIP
  FROM
    (
      SELECT
        O_ORDERDATE
      , L_SHIPDATE
      , L_SHIPDATE - O_ORDERDATE AS DAYS_TO_SHIP
      FROM
        SNOWBEAIR_DB.PROMO_CATALOG_SALES.ORDERS O
      LEFT JOIN SNOWBEAIR_DB.PROMO_CATALOG_SALES.LINEITEM L ON
        O.O_ORDERKEY = L.L_ORDERKEY
    ) AS F
  GROUP BY YR, MON
  ORDER BY YR, MON;
```

Notice that the schedule is set to run every minute of every hour. However, we only need this task to run this once for our purposes.

### 9.2.2 Show the task to ensure it got created correctly

```
SHOW TASKS;
```

### 9.2.3 Run the following commands to start the tree:

```
ALTER TASK insert_shipping_by_date_rows RESUME;
```

### 9.2.4 Use the following query to monitor what is going on in the table.

It may take up to a minute before you see any data in the table.

```
SELECT * FROM avg_shipping_in_days;
```

### 9.2.5 Use this command to see the task history for the task

Here you can see instances in which the task was scheduled, ran and succeeded, or ran and failed.

```
select *  
  from table(information_schema.task_history(  
    scheduled_time_range_start=>dateadd('hour',-1,current_timestamp())));
```

## 9.3 Ending the lab

### 9.3.1 Execute the ALTER TASK command below to immediately stop the task tree:

```
ALTER TASK insert_shipping_by_date_rows SUSPEND;
```

### 9.3.2 Run the commands below to clear out the objects used in this lab:

```
DROP TABLE avg_shipping_in_days;  
DROP TASK insert_shipping_by_date_rows;  
DROP SCHEMA [login]_TASKS_SCHEMA;  
ALTER WAREHOUSE [login]_WH SUSPEND;
```

## 9.4 Key Takeaways

- The first task in a tree of tasks must be started upon a schedule, using either Snowflake-specific scheduling syntax or CRON syntax.
- Subsequent tasks in a tree of task must be started by calling them from previous tasks with an AFTER clause.
- It is imperative to stop a task you no longer need in order to avoid wasting credits.

## 10 Snowflake Functions

The purpose of this lab is to introduce you to Snowflake's large, built-in function library.

Most of the time, the average Snowflake user uses three Snowflake components to get work done: core SQL constructs (SQL itself), the compute layer, and functions. In other words, functions are useful to every workload, to include data engineering, data lake, data warehousing, data science, data applications and even data sharing.

In this lab you'll become familiar with a handful of SQL functions. In fact, you may be familiar with similar or identical functions from other database or data warehouse systems.

### Learning Objectives:

- How to work with scalar functions
- How to work with regular and windowing aggregate functions
- How to use table and system functions
- How to use FLATTEN to work with semi-structured (JSON) data

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

### 10.1 Scalar Functions

Scalar functions take a single row or value as input and return a single value, such as a number, a string, or a boolean value.

Click [here](https://docs.snowflake.com/en/sql-reference/account-usage.html#reader-account-usage-views) to learn more about Snowflake's scalar functions. (<https://docs.snowflake.com/en/sql-reference/account-usage.html#reader-account-usage-views>)

Now let's try using a few scalar functions. Note that although you are probably familiar with most if not all of them, some may have different names or syntax than what you've seen in other systems. For example, while some systems use an IF... THEN syntax for if-then statements, Snowflake uses IFF().

10.1.1 Open a worksheet and name it *Functions* 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;
```

### 10.1.2 String functions: UPPER

Execute the statement below to convert c\_name to uppercase.

```
SELECT
    c_name
    , UPPER(c_name)
FROM
    "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."CUSTOMER";
```

### 10.1.3 Conditional Functions: IFF

Execute the following statements using the conditional function IFF to see what it does. Note that the double colon (::) casts the result of the IFF function to a number that is 16 digits long and with 2 decimal places.

```
SELECT
    o_orderkey,
    o_totalprice,
    o_orderpriority,
    IFF(o_orderpriority LIKE '1-URGENT', o_totalprice * 0.01, o_totalprice *
    0.005)::NUMBER(16,2) AS ShippingCost
FROM
    "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."ORDERS";
```

### 10.1.4 Conditional Functions: CASE

Here you'll use a conditional function to label a customer as preferred or not preferred. You'll also use a CASE expression to print out text for preferred customers.

```
SELECT (c_salutation || ' ' || c_first_name || ' ' || c_last_name) AS full_name,
    CASE
        WHEN c_preferred_cust_flag LIKE 'Y'
            THEN 'Preferred Customer'
        WHEN c_preferred_cust_flag LIKE 'N'
            THEN 'Not Preferred Customer'
        END AS customer_status
FROM
    "SNOWFLAKE_SAMPLE_DATA"."TPCDS_SF100TCL"."CUSTOMER"
LIMIT 100;
```

### 10.1.5 Numeric Functions: RANDOM

Use the following statements to generate data. Note that `random()` with no argument generates a different number every time. If you provide a seed value, for example, `random(12)`, the same value will be returned every time:

```
SELECT RANDOM() AS random_variable;

SELECT RANDOM(100) AS random_fixed;
```



### 10.1.6 Context Functions

Run this query to use some context functions:

```
SELECT CURRENT_DATE(), DATE_PART('DAY', CURRENT_DATE()), CURRENT_TIME();
```

### 10.1.7 String Functions: Converting strings to arrays and arrays to strings

Run this statement to change an array back to a string with a separator:

```
SELECT
    STRTOK_TO_ARRAY(query_text)
    , ARRAY_TO_STRING(STRTOK_TO_ARRAY(query_text), '#')
FROM
    SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE
    query_text LIKE 'select%'
    AND
    query_text NOT LIKE 'select 1%'
LIMIT 5;
```

### 10.1.8 Aggregate Functions: AVG, MIN, MAX, STDDEV

Here we use a query to produce aggregates on the total execution time of past queries. The SQL statement below accomplishes this by querying the QUERY\_HISTORY secure view in the ACCOUNT\_USAGE schema of the Snowflake database.

```
SELECT
    MONTH(qh.start_time) AS "month"
    , DAYOFMONTH(qh.start_time) AS dom
    , qh.warehouse_name
    , AVG(qh.total_elapsed_time)
    , MIN(qh.total_elapsed_time)
    , MAX(qh.total_elapsed_time)
    , STDDEV(qh.total_elapsed_time)
FROM
    SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY qh
WHERE
    query_text LIKE 'select%'
    AND
    query_text NOT LIKE 'select 1%'
GROUP BY
    "month", dom, qh.warehouse_name
ORDER BY
    "month", dom, qh.warehouse_name;
```

## 10.2 Use Regular and Windows Aggregate Functions

Aggregate functions work across rows to perform mathematical functions such as MIN, MAX, COUNT, and a variety of statistical functions.

Many of the aggregate functions can work with the OVER clause enabling aggregations across a group of rows. This is called a WINDOW function.

In this section you will work with some of the aggregate window functions.

[Click here to learn more about Snowflake's Window functions.](#)

[Click here to learn more about Snowflake's Aggregate functions.](#)

### 10.2.1 WINDOW functions:

Here we use a query to determine the SUM of credits per warehouse and day of the month. The SQL statement below accomplishes this by querying the WAREHOUSE\_METERING\_HISTORY secure view in the ACCOUNT\_USAGE schema of the Snowflake database. We partition by day of the month and warehouse name.

```
SELECT
    DAYOFMONTH(start_time) AS DAY_OF_MONTH
  , DATE(START_TIME) AS DT
  , warehouse_name
  , credits_used
  , SUM(credits_used)
    OVER (PARTITION BY DAYOFMONTH(start_time), warehouse_name
          ORDER BY DAYOFMONTH(start_time), warehouse_name ) AS day_tot_credits
FROM
    SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY

GROUP BY
    DAY_OF_MONTH, warehouse_name, DT, credits_used

ORDER BY
    DAY_OF_MONTH, warehouse_name, DT, credits_used;
```

## 10.3 TABLE and System Functions

Table functions return a set of rows instead of a single scalar value. Table functions appear in the FROM clause of a SQL statement and cannot be used as scalar functions.

System functions are functions that allow you to execute actions in the system, or that return information about queries or the system itself.

### 10.3.1 Use a table function to retrieve 1 hour of query history:

```
SELECT
    *
FROM
    TABLE(information_schema.query_history
            (DATEADD('hours', -1, CURRENT_TIMESTAMP()), CURRENT_TIMESTAMP()))
ORDER BY
    start_time;
```

### 10.3.2 Use the RESULT\_SCAN function to return the last result set:

The function RESULT\_SCAN returns the result set of a previous command (within 24 hours of when you executed the query) as if the result were a table. This is useful if you want to process the output of SHOW or DESCRIBE, the output of a query executed on account usage information, such as INFORMATION\_SCHEMA or ACCOUNT\_USAGE, or the output of a stored procedure.

The query below is simple and just produces the results of the last query you ran.

```
SELECT * FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

### 10.3.3 Use the SHOW command with the RESULT\_SCAN function to have SQL generate a list of commands to describe tables:

Note that Snowflake's SQL is generally case-insensitive. It usually changes identifiers to upper case unless they are enclosed in quotes. In this example, the "name" column must be lower case, so it is placed in quotation marks.

```
SHOW TABLES;
SELECT CONCAT('DESC ', "name", ';')
FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()));
```

### 10.3.4 SYSTEM\$WHITELIST

The SYSTEM\$WHITELIST function enables you to see information on hosts that should be unblocked for Snowflake to work. Click on the item in the first row to see what the column contains:

```
SELECT SYSTEM$WHITELIST();
```

Note that the data is in semi-structured format and that the output of the column SYSTEM\$WHITELIST() is a link. Click on a link to view the data.

### 10.3.5 SYSTEM\$WHITELIST, Semi-structured data and FLATTEN

Now we're going to use the FLATTEN table function to flatten the data for the previous query so it appears more like structured data.

```
SELECT VALUE:type AS type,  
       VALUE:host AS host,  
       VALUE:port AS port  
FROM TABLE(FLATTEN(INPUT => PARSE_JSON(system$whitelist())));
```

### 10.3.6 Suspend and resize the warehouse

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

## 10.4 Key Takeaways

- Snowflake has many of the functions you are already accustomed to using in other software programs.
- You can query the query history secure view (SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY) to produce numeric aggregates on query execution times.
- Table functions return a set of rows rather than a scalar value.
- System functions are functions that allow you to execute actions in the system, or that return information about queries or the system itself.

## 11 User-Defined Functions and Stored Procedures

The purpose of this lab is to familiarize you with user-defined functions and stored procedures in Snowflake.

### Learning Objectives:

- How to create a JavaScript user-defined function
- How to create a SQL user-defined function
- How to create a JavaScript stored procedure

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

### 11.1 Create a JavaScript User-Defined Function

11.1.1 Open a new worksheet and name it **UDFs** 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;
```

11.1.2 Run the following to create a UDF named **Convert2Meters** :

As you can see, the UDF below is in JavaScript. The LANGUAGE clause below explicitly states that it is JavaScript, so Snowflake knows what to do with the script between the \$\$ marks.

```
CREATE OR REPLACE FUNCTION Convert2Meters(lengthInput double, InputScale string )
  RETURNS double
  LANGUAGE javascript
  AS
  $$
  /*
   * convert English measurements to meters
   */
  var scale_UC = INPUTSCALE.toUpperCase();
  switch(scale_UC) {
    case 'INCH':
      return(LENGTHINPUT * 0.0254)
      break;
    case 'FEET':
      return(LENGTHINPUT * 0.3048)
```

```
        break;
    case 'YARD':
        return(LENGTHINPUT * 0.9144)
        break;

    default:
        return null;
        break;
}
$$;
```

11.1.3 Call the UDF just created as part of a SELECT statement and return the value in meters:

```
SELECT Convert2Meters(10, 'yard');
```

## 11.2 Create a SQL User-defined Function

11.2.1 Create a function that returns the count of orders based on the customer number. It will be a scalar function because it will return a single value.

Note that there is only a single SQL statement between the \$\$ marks. In a SQL UDF you can have only one SQL statement.

```
CREATE OR REPLACE FUNCTION order_cnt(custkey number(38,0))
RETURNS number(38,0)
AS
$$
    SELECT COUNT(1) FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."ORDERS" WHERE
        o_custkey = custkey
$$;
```

11.2.2 Now use the UDF in a query

```
SELECT C_name, C_address, order_cnt(C_custkey)
FROM "SNOWFLAKE_SAMPLE_DATA"."TPCH_SF1"."CUSTOMER" LIMIT 10;
```

## 11.3 Creating Stored Procedures

11.3.1 Create a JavaScript stored procedure named `ChangeWHSize()`.

Remember that JavaScript is case-sensitive, whereas SQL is not. The JavaScript appears between the \$\$ delimiters, so make sure case in the JavaScript portion is preserved.

The procedure demonstrates executing SQL statements in a JavaScript stored procedure.

Also, note that you can have procedural logic, including try - catch statements, in a JavaScript stored procedure.

```
/* ***
 * ChangeWHSIZE(wh_name STRING, wh_size STRING)
 *
 * Description: Change a WH size to a new size. The size is limited to larger or
 * below.
 */
create or replace procedure ChangeWHSIZE(wh_name STRING, wh_size STRING )
  returns string
  language javascript
  strict
  execute as owner
  as
  $$
  /*
   * Change the named warehouse to a new size if the new size is LARGE OR smaller
   */
  var wh_size_UC = WH_SIZE.toUpperCase();
  switch(wh_size_UC) {
    case 'XSMALL':
    case 'SMALL':
    case 'MEDIUM':
    case 'LARGE':
      break;
    case 'XLARGE':
    case 'X-LARGE':
    case 'XXLARGE':
    case 'X2LARGE':
    case '2X-LARGE':
    case 'XXXLARGE':
    case 'X3LARGE':
    case '3X-LARGE':
    case 'X4LARGE':
    case '4X-LARGE':
      return "Size: " + WH_SIZE + " is too large";
      break;
    default:
      return "Size: " + WH_SIZE + " is not valid";
      break;
  }

  var sql_command =
    "ALTER WAREHOUSE IF EXISTS " + WH_NAME + " SET WAREHOUSE_SIZE = " + WH_SIZE;
  try {
    snowflake.execute (
      {sqlText: sql_command}
    );
    return "Succeeded."; // Return a success/error indicator.
  }
  catch (err) {
    return "Failed: " + err; // Return a success/error indicator.
  }
  $$
  ;
```

11.3.2 Call the stored procedure with a valid warehouse size:

```
CALL changewhsize ('[login]_wh', 'small');
```

11.3.3 Call the stored procedure with an invalid warehouse size:

```
CALL changewhsize ('[login]_wh', 'XLARGE');
```

11.3.4 Suspend and resize the warehouse

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

## 11.4 Key Takeaways

- In the LANGUAGE clause of a UDF you can explicitly state that the language is JavaScript. Snowflake will then know exactly what to do with the script between the \$\$ marks.
- In a SQL UDF you can have only one SQL statement.
- You can embed SQL statements in a JavaScript stored procedure.
- You can have procedural logic, including try - catch statements, in a JavaScript stored procedure.



## 12 Using High-Performing Functions

As you know, common functions in relational database management systems such as `COUNT(DISTINCT)` and percentage/percentile functions require a scan and sort of an entire dataset to yield a result. Although a cloud database like Snowflake is designed to handle virtually unlimited quantities of data, executing a `COUNT(DISTINCT)` on a very large cloud table could take far longer than a user is willing to wait. Additionally, when working with very large datasets, absolutely precise counts are unnecessary, especially if your data is being updated in real time or near real time.

Snowflake's high performing functions are designed to give you approximate results that should satisfy your analytical needs but in a far shorter time frame than a standard `COUNT(DISTINCT)`. The purpose of this lab is to give you hands-on experience with a couple of these functions: `HLL()` or HyperLogLog, which can be used in lieu of the standard `COUNT(DISTINCT)` function, and `APPROX_PERCENTILE`, which can be used in lieu of the standard `SQL MEDIAN` function.

### Learning Objectives:

- How to leverage HyperLogLog to get an approximate count
- How to leverage `APPROX_PERCENTILE` to get an approximate median

### Scenario:

You've just learned about a couple of Snowflake's high performing functions that you think may be useful for your analysis needs. One is HyperLogLog, and the other is `APPROX_PERCENTILE`. You've decided to try these out on a couple of tables that you know have anywhere from hundreds of millions of rows to even billions of rows just to see how they perform. This is your plan:

- Run both HyperLogLog and `COUNT(DISTINCT)` to see which returns a result faster.
- Run both `APPROX_PERCENTILE` and `MEDIAN()` to see which returns a result faster.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can key the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

## 12.1 Working with HyperLogLog

12.1.1 Create a new folder and call it High Performing Functions.

12.1.2 Create a new worksheet inside the folder and call it Working with High Performing Functions.

12.1.3 Alter the session so it does not use cached results. This will give us an accurate reading as to the longest time the functions will take to run:

```
ALTER SESSION SET use_cached_result=false;
```

12.1.4 Set the Worksheet contexts as follows:

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

12.1.5 Change the virtual warehouse size to XSmall

Your warehouse may already be XSmall, but we want to make sure that it is so we can get a clear difference between how quickly each function will run.

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

12.1.6 Suspend the warehouse

With the statements below, you'll suspend and resume the warehouse to clear any data in the warehouse data cache. Then you'll use the query below to determine an approximate count of distinct l\_orderkey values with Snowflake's HyperLogLog high-performing function:

**NOTE:** If you try to suspend the warehouse but the warehouse is already suspended, you may get an error indicating the warehouse is already suspended. This is normal. You would simply need to restart the warehouse and continue your work.

```
ALTER WAREHOUSE [login]_WH SUSPEND;  
ALTER WAREHOUSE [login]_WH RESUME;  
  
SELECT HLL(l_orderkey) FROM lineitem;
```

How long did it take to run and how many distinct values did it find? It should have taken fewer than 10 seconds to run and it should have counted right around 145,660,677 rows.

12.1.7 Suspend and resume the warehouse again to clear the data cache. Execute the regular COUNT version of the query so we can compare the results to the those of the HyperLogLog execution:

```
ALTER WAREHOUSE [login]_WH SUSPEND;  
ALTER WAREHOUSE [login]_WH RESUME;  
  
SELECT COUNT(DISTINCT l_orderkey) FROM lineitem;
```

How long did it take to run and how many distinct values did it count? It should have taken more than 20 seconds to run and it should have returned a count of exactly 150,000,000 values.

So, the difference is approximately 4,339,323 rows, which is a variance of 2.9%. If a variance of 2.9% is not critical to whatever analysis you're doing, especially when working with counts in the hundreds of millions, then HyperLogLog can be a better choice than COUNT(DISTINCT) in those instances.

## 12.2 Use Percentile Estimation Functions

Now let's try out the APPROX\_PERCENTILE function. This function can give a more rapid response than the regular SQL MEDIAN function.

Rather than the LINEITEM table we're going to use the ORDERS table.

12.2.1 Change your warehouse size to Large and clear your warehouse cache:

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

12.2.2 Start by using the SQL Median Function. The following statement determines the median order total in each year of data:

```
SELECT  
  YEAR(O_ORDERDATE)  
  , MEDIAN(O_TOTALPRICE)  
  
FROM  
  ORDERS  
  
GROUP BY  
  YEAR(O_ORDERDATE)  
  
ORDER BY  
  YEAR(O_ORDERDATE);
```

How long did it take to run and what results did you get? It should have run in 15-20 seconds, and you should have gotten the results below:

- 1992 - 144310.1 - 1993 - 144303.67 - 1994 - 144285.85 - 1995 - 144282.92 - 1996 - 144322.46 - 1997 - 144284.45 - 1998 - 144318.58

12.2.3 Run the Percentile Estimation Function on the *orders* table to find the approximate 50th percentile of sales for each year in the table:

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

SELECT
    YEAR(O_ORDERDATE)
    , APPROX_PERCENTILE(O_TOTALPRICE, 0.5)

FROM
    ORDERS

GROUP BY
    YEAR(O_ORDERDATE)

ORDER BY
    YEAR(O_ORDERDATE);
```

How long did it take to run and what results did you get? It should have run in fewer than 2 seconds, and you should have gotten the results that look very much like (but not exactly like) the ones below:

- 1992 - 144315.338198642 - 1993 - 144302.777148666 - 1994 - 144284.126806681 - 1995 - 144278.419806512 - 1996 - 144325.010908467 - 1997 - 144289.365240779 - 1998 - 144323.018226321

As you can see, the APPROX\_PERCENTILE function does run quite a bit faster than the standard MEDIAN() function and produces almost the exact same result. Your results may not look exactly like the figures we've shown above because the function returns an approximate value each time. Regardless, for the tiny variance you get, you can get a far faster return of the result set.

12.2.4 Change your warehouse size to XSmall:

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

ALTER WAREHOUSE [login]_WH SUSPEND;
```

## 12.3 Key takeaways

- If small variances are not critical to you, high performing functions can help you write more efficient queries that run much faster for your business users.

## 13 Access Control and User Management

The purpose of this lab is to familiarize you with role-based access control (RBAC) in Snowflake. Specifically, you'll become familiar with the Snowflake security model and learn how to create roles, grant privileges, and how to build and implement basic security models.

### Learning Objectives:

- How to show grants to users and roles
- How to grant usage on objects to roles

### Scenario:

The purpose of this exercise is to give you a chance to see how you can manage access to data in Snowflake by granting privileges to some roles and not to others.

In this lab, SYSADMIN will represent the privileges of a user that shouldn't have access to a specific table in a specific database, while the role TRAINING\_ROLE will represent the privileges of a user that shouldn't. This lab will walk you through the process of setting all this up so you can test the roles and observe the results.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

### 13.1 Determine Privileges (GRANTS)

In this section of the lab you'll use SHOW GRANTS to determine what roles a USER has, and what privileges a ROLE has received. This is an important step in determining what a USER is or isn't allowed to do.

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

**NOTE:** This worksheet must not be in a folder or the switching of roles that you will need to do will not work.

13.1.2 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;
```

13.1.3 Run these commands one at a time to see what roles have been granted to you as a user, and what privileges have been granted to specified roles:

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

You should see that TRAINING\_ROLE has some specific privileges granted and is quite powerful. This has been done on purpose so you can do the labs more easily. In a production environment it is unlikely that you would ever see a role like this.

## 13.2 Granting Permissions (GRANT ROLE and GRANT USAGE)

In this section, you'll use GRANT ROLE to give additional privileges to a ROLE, and how to use GRANT USAGE to permit a user to perform actions on or with a database object.

13.2.1 Change your role to SYSADMIN:

```
USE ROLE SYSADMIN;
```

13.2.2 Create a warehouse named [login]\_SHARED\_WH :

Now we're going to create a warehouse that both roles will use. After that you'll grant permissions to both roles.

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

13.2.3 Create a database called [login]\_CLASSIFIED\_DB :

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

13.2.4 Use the role **SYSADMIN** , and create a table called **SUPER\_SECRET\_TBL** inside the [login]\_CLASSIFIED\_DB.PUBLIC schema:

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

13.2.5 Insert some data into the table:

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

### 13.2.6 GRANT SELECT privileges on SUPER\_SECRET\_TBL to the role TRAINING\_ROLE :

Here we're going to GRANT SELECT to TRAINING\_ROLE but we're NOT going to GRANT USAGE on the database nor on its schemas.

**NOTE** If we DON'T grant usage on the database AND on the schemas to a role, that role won't be able to do things like create tables or select from tables EVEN IF that role has create or select privileges. In other words, you must have the appropriate permissions on all objects in the hierarchy from top to bottom in order to work at the lowest level of the hierarchy.

```
GRANT SELECT ON SUPER_SECRET_TBL TO ROLE TRAINING_ROLE;
```

### 13.2.7 Use the role TRAINING\_ROLE to SELECT \* from the table SUPER\_SECRET\_TBL :

Now let's try to select some data using TRAINING\_ROLE. What do you think is going to happen?

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

We're not able to select any data. That's because the role we're using has not been granted USAGE on the database or the schema PUBLIC. Let's GRANT USAGE on both of those objects to TRAINING\_ROLE and see what happens.

### 13.2.8 Grant role TRAINING\_ROLE usage on all schemas in [login]\_CLASSIFIED\_DB :

```
USE ROLE SYSADMIN;  
GRANT USAGE ON DATABASE [login]_CLASSIFIED_DB TO ROLE TRAINING_ROLE;  
GRANT USAGE ON ALL SCHEMAS IN DATABASE [login]_CLASSIFIED_DB TO ROLE TRAINING_ROLE;  
  
USE ROLE TRAINING_ROLE;  
SELECT * FROM [login]_CLASSIFIED_DB.PUBLIC.SUPER_SECRET_TBL;
```

This time it worked! This is because your role has the appropriate permissions at all levels of the hierarchy.

### 13.2.9 Drop the database [login]\_CLASSIFIED\_DB :

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

## 13.3 Key Takeaways

- USAGE is granted to ROLES, which in turn are granted to USERS.
- USAGE must be granted on all levels in a hierarchy (database and schema) in order for a role to have the ability to select from a table.

## 14 Secondary Roles

The purpose of this lab is to familiarize you with secondary roles and how you can use them to access both a primary role and a secondary role already granted to the user within a single session.

### Learning Objectives:

- How to use Secondary Roles to aggregate permissions from more than one role

### Scenario:

You are going to use USE SECONDARY ROLES to aggregate permissions from two different roles, SYSADMIN and TRAINING\_ROLE.

First, you will use SYSADMIN to create a database and table and to insert a row into the table. You will then switch to TRAINING\_ROLE and try to access the table.

Next, you will enable secondary roles and try accessing the table again with TRAINING\_ROLE.

You will then disable secondary roles and switch back to SYSADMIN in order to grant TRAINING\_ROLE access to the database, schema, and table. You will then switch back to TRAINING\_ROLE and try the access again.

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

Let’s get started!

### 14.1 Determine Privileges (GRANTS)

In this section of the lab you’ll use SHOW GRANTS to determine what roles a USER has, and what privileges a ROLE has received. This is an important step in determining what a USER is or isn’t allowed to do.

14.1.1 Navigate to **[Worksheets]** and create a new worksheet named *Secondary Roles*.

**NOTE:** This worksheet must not be in a folder or the switching of roles that you will need to do will not work.

14.1.2 If you haven’t created the class warehouse, do it now

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



## 14.2 Granting Permissions (GRANT ROLE and GRANT USAGE)

In this section, you'll learn how to use GRANT ROLE to give additional privileges to other ROLES, and how to use GRANT USAGE to permit a ROLE to perform actions on or with a database object.

### 14.2.1 Change to the role SYSADMIN to create the new database and table

```
USE ROLE SYSADMIN;
```

### 14.2.2 Create a database called [login]\_ROLETEST\_DB :

```
CREATE DATABASE [login]_ROLETEST_DB;  
USE [login]_ROLETEST_DB.PUBLIC;  
CREATE TABLE ROLE_TBL (id INT);  
  
-- Insert a row of data into the table.  
INSERT INTO ROLE_TBL VALUES (1), (10), (30);  
  
-- Check the table to make sure it got the data  
SELECT * FROM ROLE_TBL;
```

### 14.2.3 Switch to the TRAINING\_ROLE and try to access the database, schema, and table created above

```
USE ROLE TRAINING_ROLE;  
SELECT * FROM [login]_ROLETEST_DB.PUBLIC.ROLE_TBL;
```

We're not able to select any data because TRAINING\_ROLE has not been granted access to select from the table ROLE\_TBL.

### 14.2.4 Enable SECONDARY ROLE ALL and try again

```
USE SECONDARY ROLE ALL;  
SELECT * FROM [login]_ROLETEST_DB.PUBLIC.ROLE_TBL;
```

With SECONDARY ROLE ALL set, the current user can use any permission from any role the user has been granted except CREATE.

### 14.2.5 Alter the table while SECONDARY ROLE ALL is set

```
ALTER TABLE [login]_ROLETEST_DB.PUBLIC.ROLE_TBL ADD COLUMN name STRING(20);  
SELECT * FROM [login]_ROLETEST_DB.PUBLIC.ROLE_TBL;
```

This should work since the roles granted to your user include the SYSADMIN role, the owner of the table.

#### 14.2.6 Try creating a new table in the [login]\_ROLETEST\_DB

```
CREATE TABLE [login]_ROLETEST_DB.PUBLIC.NOROLE_TBL (name STRING(20));
```

You cannot create a table because the current role TRAINING\_ROLE does not have CREATE TABLE privileges on the database. As mentioned before, USE SECONDARY ROLES does not include CREATE privileges given to other roles.

#### 14.2.7 Disable SECONDARY ROLES

```
USE SECONDARY ROLE NONE;
```

#### 14.2.8 Switch back to the SYSADMIN role and give TRAINING\_ROLE permission to select from the table

Here we're going to GRANT SELECT to the ROLETEST\_TBL but we're NOT going to GRANT USAGE on the database nor on its schemas.

```
USE ROLE SYSADMIN;  
GRANT SELECT ON [login]_ROLETEST_DB.PUBLIC.ROLE_TBL TO ROLE TRAINING_ROLE;
```

#### 14.2.9 Select some data using TRAINING\_ROLE

```
USE ROLE TRAINING_ROLE;  
SELECT * FROM [login]_ROLETEST_DB.PUBLIC.ROLE_TBL;
```

We're not able to select any data. That's because the role we're using does not have GRANT USAGE on the database or the schema PUBLIC. Let's GRANT USAGE on both of those objects and see what happens.

#### 14.2.10 Grant role TRAINING\_ROLE usage on all schemas in [login]\_ROLETEST\_DB :

```
USE ROLE SYSADMIN;  
GRANT USAGE ON DATABASE [login]_ROLETEST_DB TO ROLE TRAINING_ROLE;  
GRANT USAGE ON ALL SCHEMAS IN DATABASE [login]_ROLETEST_DB TO ROLE TRAINING_ROLE;
```

#### 14.2.11 Now try again:

```
USE ROLE TRAINING_ROLE;  
SELECT * FROM [login]_ROLETEST_DB.PUBLIC.ROLE_TBL;
```

This time it worked! This is because TRAINING\_ROLE has all the needed permissions, without having to resort to any secondary roles.

14.2.12 Drop the database [login]\_ROLETEST\_DB :

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

### 14.3 Key Takeaways

- Secondary roles can be used to aggregate permissions in a single session.
- You can only create objects if the primary role has permissions to do that.

## 15 Exploring Semi-Structured JSON Data

As you know, Snowflake is designed to natively store semi-structured data in various formats. The purpose of this lab is for you to learn how to analyze JSON formatted semi-structured data stored in Snowflake.

### Learning Objectives:

- How to query a column containing semi-structured data without using the FLATTEN function.
- How to write a query that uses the FLATTEN table function to query semi-structured data.
- How to query a column containing semi-structured data that is flattened by using the FLATTEN function.
- How to leverage the PATH and KEY output values of the FLATTEN function to get the data you want.
- How to leverage DESCRIBE RESULT to determine the data types of a query's columns.

### Scenario:

You've been given a table with semi-structured data that contains a customer ID and the customer's first and last names. You've been asked to produce a data set that includes a row for each ID. The row must display the customer's ID, first, and last names. Your task is to write a query that produces the requested data.

### HOW TO COMPLETE THIS LAB

To complete this lab, you can key the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

### 15.1 Working with semi-structured data

In the first half of this lab there are two exercises to introduce you to querying semi-structured data. In this first exercise you will query the semi-structured data accessing the top-level key-value pairs.

15.1.1 Create a new folder and call it Semi-Structured Data.

15.1.2 Create a new worksheet inside the folder and call it Working with Semi-Structured Contact Data.

15.1.3 Create the data.

Let's create the data we'll need for this first scenario. Run the SQL statements below:

```
USE ROLE TRAINING_ROLE;
```

```

CREATE DATABASE IF NOT EXISTS [login]_DB;
CREATE OR REPLACE SCHEMA [login]_DB.[login]_SCHEMA;

USE WAREHOUSE [login]_WH;
USE SCHEMA [login]_DB.[login]_SCHEMA;

CREATE OR REPLACE TABLE persons AS
SELECT
    column1 AS id
    , parse_json(column2) AS c
FROM
    VALUES
    (12712555,
    '{ "name": { "first": "John", "last": "Smith"}}'),
    (98127771,
    '{ "name": { "first": "Jane", "last": "Doe"}}') v;

```

Let's look at the data we're loading. The first item is a scalar value, which is the ID. Next, inside the curly braces, we have the JSON key-value pairs. Using the first data value as an example - name is a key and the values are the first + John and last + Smith. We also have nested key-value pairs. First and last are the keys. The customer's first and last names, respectively, are the values.

Now let's begin writing our queries. Let's start with the following SQL statement (you don't need to run it):

```

SELECT
    *
FROM
    persons p;

```

In this statement we have the numeric ID in the first column titled "ID". In the column titled "C" we have the key-value pairs for name, both first and last. The data is presented in JSON format with quotation marks and curly braces.

Let's extract the name data values and cast them as varchar so that they're easy to read.

```

SELECT
    p.ID
    ,p.c:name.first::varchar AS first_name
    ,p.c:name.last::varchar AS last_name
FROM
    persons p;

```

To access the data from the Persons table we specify its access path and the format for the result. For the ID - no formatting is required. For the first and last names we specify the path to this data embedded in a semi-structured content, and the data type. The syntax is:

- p - the table alias
- c - the column alias for the VARIANT column
- :name.first - the first key-value pair for which we need the value
- :name.last - the second key-value pair for which we need the value
- ::varchar - casts the result as varchar

NOTE: If we do not CAST the result from a VARIANT column the returned value will be of data type VARIANT.

In order to perform math reliably on variant data, any numeric data of data type variant must be cast as an appropriate numeric data type.

We made no change to the result format of the ID. We did change the result format of the name data. In addition to being tidy and easy to read, with our result data in tabular format we can now use it like any other structured data. For example, it could be inserted into a table, or joined with other structured data.

## 15.2 Using FLATTEN

Now you will query JSON data stored in an array. We're going to expand our JSON data set to introduce the FLATTEN function. After that we offer an exercise to try that involves a Snowbear Air-based scenario using more complex data.

### 15.2.1 Scenario

The JSON data used to populate the table has been expanded to include contact details in addition to the customer ID, first and last names. We've now been asked to produce a data set that includes a single row for each contact. The row must display the ID, first and last names, phone number, and email address. Our task is to explode the data and write a query that produces the requested result data.

### 15.2.2 Create the data

Let's create the data we'll need for this second exercise. Run the SQL statement below:

```
CREATE OR REPLACE TABLE persons AS
SELECT
  column1 AS id
  , parse_json(column2) AS c
FROM
VALUES
  (12712555,
   '{ "name": { "first": "John", "last": "Smith"},
     "contact": [
       { "business": [
         { "type": "phone", "content": "555-1234" },
         { "type": "email", "content": "j.smith@company.com" } ] } ] }'),
  (98127771,
   '{ "name": { "first": "Jane", "last": "Doe"},
     "contact": [
       { "business": [
         { "type": "phone", "content": "555-1236" },
         { "type": "email", "content": "j.doe@company.com" } ] } ] }') v;
```

Let's look at the JSON data using the following SQL statement:

```
SELECT
  *
FROM
  persons p;
```

Just as with our initial exercise, the first column value is the ID. But now we have two paths to follow in this semi-structured data column “C”: name and contact. We know that name is a semi-structured key-value pair with name as the key. Contact is a key, and an array named business is nested within contact. We know this because of the square brackets used to indicate arrays. Business contains a set of key-value pairs.

To get the data we need from the arrays, we’re going to use the FLATTEN table function. Take a moment to review the definition below.

**FLATTEN**

FLATTEN is a table function that explodes the contents of a variant column (for purposes of this lab, a column with semi-structured data) and produces a relational representation that correlates to the table from which it was derived, and which precedes it in the FROM clause.

**ARGUMENTS:**

1. INPUT: A reference to a column containing semi-structured data (required).
2. PATH: A path defined by a key from one of the top-level key-value pairs in the semi-structured data (optional).

**OUTPUT**

1. EXPLODED DATA: Semi structured data presented in a tabular structure and in variant format.
2. SEQUENCE: A unique sequence number associated with the input record. The sequence is not guaranteed to be gap-free or ordered in any particular way.
3. KEY: The key of a key-value pair for an exploded value.
4. PATH: A path to the element within the semi-structured data that needs to be flattened.
5. INDEX: The index of the element, if it is an array; otherwise NULL.
6. VALUE: The value of the element of the flattened array/object.
7. THIS: The element being flattened (useful in recursive flattening).

Now that we have a high-level understanding of how the FLATTEN function works, let’s use it in a query. We’ll input the column containing the semi-structured data and indicate the path is contact. We’ll provide column p.c for the required argument INPUT as well as the value ‘contact’ for the optional argument PATH:

```
SELECT
  *
FROM
  persons p
, LATERAL FLATTEN(INPUT => p.c, PATH => 'contact') f;
```

There are numerous fields that are part of the flatten output by default: SEQ, KEY, PATH, INDEX, VALUE and THIS. For this lab we will concern ourselves only with PATH and KEY, which will be discussed in greater detail below.

Now let's fetch the contact's name from the JSON:

```
SELECT
  id AS "ID"
  , n.value AS name_value

FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'name') n;
```

Here we provided two parameters - the column with the JSON and the path that we wanted to follow, which is the name path. Note that there are two key-value pairs (first and last). Flattening fetches both values from the two pairs and presents them as two rows each. We have another way to fetch the first and last name so they're on the first row. We'll do that shortly.

Note the quotes around the first and last names. Run the SQL statement below to find out why.

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

Note that the NAME\_VALUE column is a VARIANT. It would be better to have it in VARCHAR format so we can filter and sort on that column and get predictable results. Let's fix that now:

```
SELECT
  id AS "ID"
  , n.value::varchar AS name_value

FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'name') n;
```

Now we've cast the value as varchar and it looks correct. Run the DESCRIBE RESULT statement again to check the data type.

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

We can see the column name\_value is in varchar format. Now let's add the contact details into our query:

```
SELECT
  id AS "ID"
  , n.value::varchar AS name_value
  , f.value AS contact_value
FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'name') n
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') f;
```

The contact column values are in semi-structured format. We executed a new lateral flatten and provided it two values: the column with the semi-structured data we wanted to flatten, and the path within the semi-structured data we wanted to drill down into (contact). This is because we want the CONTENT field which has the data we need.

Now run the following to fetch the CONTENT column:



```

SELECT
  id AS "ID"
  , n.value::varchar AS name_value
  , f1.value:content::varchar AS "Content"
  , f.value AS contact_value

FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'name') n
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') f
  , LATERAL FLATTEN (INPUT => f.value:business) f1;

```

Here we accessed the content column and cast it as varchar. We accomplished this by adding another flatten clause that accesses the nested array business within contact. We can see that we have pretty much extracted the content we want. The problem is that the values are repeating based on how the semi-structured data was structured. Let's fix this problem. Run the SQL statement below:

```

SELECT
  p.ID
  , p.c:name.first::varchar AS first_name
  , p.c:name.last::varchar AS last_name
  , f.value AS contact_value

FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') n
  , LATERAL FLATTEN (INPUT => n.value:business) f;

```

First we removed the last flatten for now so we can focus on the name. Next, we extracted the first and last name. We started by referencing the column, p.c. Then, we put a colon, which indicates we're going to drill down into the key-value pair. Then for the first and last names, we referenced the path and key, name.first and name.last respectively. Finally, we cast the value to varchar. This put the first and last names together on a single line. The syntax can be something like this:

table.column:key1.key2.keyN

Now we'll extract the phone number and email from the JSON:

```

SELECT
  p.ID
  , p.c:name.first::varchar AS first_name
  , p.c:name.last::varchar AS last_name
  , f.value:content::varchar AS content
  , f.path

FROM
  persons p
  , LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') n
  , LATERAL FLATTEN (INPUT => n.value:business) f;

```

Notice that we still have two rows for John Smith and two rows for Jane Doe. However we've managed to extract the phone number and email address from the JSON. Focusing on the first row of each person, we see the content column contains the phone number while the second row for each person contains the email address. So now the content is in relational format. The problem is that we have repeating rows, which is not what we wanted.

But from the path column we added, we can see that the paths are from a zero-based array. The path with the phone number is [0] and the path with the email address is [1]. Let's flatten n.value:business twice and apply a path to each. Run the following statement:

```
SELECT
  p.ID
, p.c:name.first::varchar AS first_name
, p.c:name.last::varchar AS last_name
, f.value::varchar as phone_number
, f1.value::varchar as email_address
, f.key AS PATH_0_KEY
, f1.key AS PATH_1_KEY

FROM
  persons p
, LATERAL FLATTEN (INPUT => p.c, path => 'contact') n
, LATERAL FLATTEN (INPUT => n.value:business, PATH => '[0]') f
, LATERAL FLATTEN (INPUT => n.value:business, PATH => '[1]') f1;
```

So now we've put the phone number and the email address on the same lines but there are repeating rows that we need to deal with. We've added in the KEY output from the flatten clauses for paths [0] and [1]. When we look at the rows we can see that we need only the rows where both keys state 'content'. Let's apply that filter now:

```
SELECT
  p.ID
, p.c:name.first::varchar AS first_name
, p.c:name.last::varchar AS last_name
, f.value::varchar AS phone_number
, f1.value::varchar AS email_address
FROM
  persons p
, LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') n
, LATERAL FLATTEN (INPUT => n.value:business, PATH => '[0]') f
, LATERAL FLATTEN (INPUT => n.value:business, PATH => '[1]') f1
WHERE
  f.key = 'content'
AND
  f1.key = 'content';
```

Now we have exactly the content we wanted. Run the statement below to see how we can get the same result but with a join:

```
SELECT
  p.ID
, p.c:name.first::varchar AS first_name
, p.c:name.last::varchar AS last_name
, f.value::varchar as phone_number
, f1.value::varchar as email_address

FROM
  persons p INNER JOIN persons p1 ON p.ID = p1.ID
, LATERAL FLATTEN (INPUT => p.c, PATH => 'contact') n
, LATERAL FLATTEN (INPUT => n.value:business, PATH => '[0]') f
, LATERAL FLATTEN (INPUT => p1.c, PATH => 'contact') n1
, LATERAL FLATTEN (INPUT => n1.value:business, PATH => '[1]') f1
```

```
WHERE
  f.key = 'content'
AND
  f1.key = 'content';
```

### 15.3 Working with Weather Data

Ok, now that we understand how to use the FLATTEN function, let's apply what we've learned to some real weather data.

#### Scenario:

As you know, Snowbear Air does more than just sell cool apparel branded with its logo. It actually flies customers to fun destinations all over the world! Recently Snowbear Air has decided to analyze weather conditions at airports around the world so it can better understand those trends and how they may impact flight arrival times.

You've been provided with a set of weather quality control check data in JSON format. Your task is to calculate the average, max and min air temperature in Celsius and Fahrenheit for each weather station. You've also been asked to limit the results to air temperature values that have passed all quality control checks, and that fall within an 8-day period starting on 2019-08-14 and ending on 2019-08-21.

Let's get started!

#### 15.3.1 Review the Weather Data

Much of this lab will involve extracting and examining the weather data. Since it is in semi-structured format, it will be important to understand how it is structured, how it is stored, and the specific syntax needed to execute SELECT statements against it.

#### 15.3.2 Create a new worksheet inside the folder you created for this lab and call it Working with Semi-Structured Weather Data.

#### 15.3.3 Setting the context

For this portion of the lab we'll use the database TRAINING\_DB and the schema WEATHER.

Set the context using the code shown below:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE DATABASE TRAINING_DB;
USE SCHEMA TRAINING_DB.WEATHER;
```

#### 15.3.4 Use the DESCRIBE TABLE command to describe the ISD\_2019\_DAILY table:

The table that contains the data we're interested in is called `isd_2019_daily` table. Let's examine its structure now.

Run the following query to see what fields it contains:

```
DESCRIBE TABLE isd_2019_daily;
```

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

#### 15.3.5 Select all columns from the ISD\_2019\_DAILY table, limit the results to 20 rows, and explore the data and its content:

```
SELECT
    *
FROM
    weather.isd_2019_daily w
LIMIT 20;
```

#### 15.3.6 Click on the first cell of JSON data in column V.

Click on the first cell of JSON data in column V, to the right of the results pane to reveal the JSON data in a small side pane. Click any row to view it in a formatted way. What is shown should then look like this:

```
{
  "data": {
    "observations": [
      {
        "air": {
          "dew-point": 26.8,
          "dew-point-quality-code": "1",
          "temp": 29,
          "temp-quality-code": "1"
        },
        "atmospheric": {
          "pressure": 10086,
          "pressure-quality-code": "1"
        },
        "dt": "2019-08-01T00:00:00",
        "sky": {
          "ceiling": 22000,
          "ceiling-quality-code": "1"
        },
        "visibility": {
          "distance": 999999,
          "distance-quality-code": "9"
        },
        "wind": {
          "direction-angle": 200,
          "direction-quality-code": "1",

```

```

        "speed-quality-code": "1",
        "speed-rate": 10
      }
    },
    {
      "air": {
        "dew-point": 25.1,
        "dew-point-quality-code": "1",
        "temp": 32.3,
        "temp-quality-code": "1"
      },
      "atmospheric": {
        "pressure": 10083,
        "pressure-quality-code": "1"
      },
      "dt": "2019-08-01T03:00:00",
      "sky": {
        "ceiling": 99999,
        "ceiling-quality-code": "9"
      },
      "visibility": {
        "distance": 7000,
        "distance-quality-code": "1"
      },
      "wind": {
        "direction-angle": 160,
        "direction-quality-code": "1",
        "speed-quality-code": "1",
        "speed-rate": 30
      }
    }, ...SOME DATA REMOVED FOR BREVITY
  ]
},
"station": {
  "USAF": "547250",
  "WBAN": 99999,
  "coord": {
    "lat": 37.5,
    "lon": 117.533
  },
  "country": "CH",
  "elev": 12,
  "id": "54725099999",
  "name": "HUIMIN"
}
}

```

### 15.3.7 Figure out the strategy

Note that the top-level paths are data and station. At the next level down but on the same level are the key-value pairs country, elev, id and name, among others.

The array called observations contains repeating sets of key-value pairs (air, atmospheric, dt, visibility, wind, etc.) each of which is a key for a value that consists of a nested set of key-value pairs (except for dt, which is a key for a simple key-value pair).

Note that temp and temp-quality-code (the field that indicates if the air temperature values for a specific check passed all quality control checks) are located under data, observations, air. This means that we will have to flatten observations to get to that data.

The other pieces of data we need are station, country, date. We can get date straight from the structured column T. We should then be able to use our normal semi-structured querying syntax to get to country and station. We can then calculate the Fahrenheit measurements from the Celsius measurements.

Let's get our data!

### 15.3.8 Fetch country, date, station and filter for rows between '2019-08-14' AND '2019-08-21'

Run the following SQL command:

```
SELECT
    weather.v:station.country::VARCHAR AS country
    , weather.t as date
    , weather.v:station.name::VARCHAR AS station
    , observations.value
FROM
    weather.isd_2019_daily weather
    , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
ORDER BY
    DATE;
```

Note that we flattened the observations array in order to get to its nested data.

Now let's fetch the air temperature:

```
SELECT
    weather.v:station.country::VARCHAR AS country
    , weather.t as date
    , weather.v:station.name::VARCHAR AS station
    , observations.value:air.temp
    , observations.value
FROM
    weather.isd_2019_daily weather
    , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
ORDER BY
    DATE;
```

Note that we now have the air temperature in our query.

### 15.3.9 Check the air temp column's data type:

Run the following statement to check the data type:

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

As we can see it's a variant. Let's cast it to a number:

### 15.3.10 Cast air.temp to NUMBER(38,1)

Run the following SQL commands and verify that air.temp is now cast as a number.

```
SELECT
    weather.v:station.country::VARCHAR AS country
  , weather.t as date
  , weather.v:station.name::VARCHAR AS station
  , observations.value:air.temp::NUMBER(38,1) AS temp_c
FROM
    weather.isd_2019_daily weather
  , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
ORDER BY
    DATE;

DESCRIBE RESULT LAST_QUERY_ID();
```

Air.temp is now a number and has been aliased as temp\_c.

### 15.3.11 Fetch the average temperature in Celsius

Now let's apply the AVG function to our temperature field and add a GROUP BY clause:

```
SELECT
    weather.v:station.country::VARCHAR AS country
  , weather.t as date
  , weather.v:station.name::VARCHAR AS station
  , AVG(observations.value:air.temp)::NUMBER(38,1) as avg_temp_c
FROM
    weather.isd_2019_daily weather
  , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
GROUP BY
    country, date, station
ORDER BY
    DATE;
```

### 15.3.12 Add columns for the MIN and MAX temperatures in Celsius

Note below that we only had to add the same field two more times but apply either a MIN or MAX to each:

```
SELECT
    weather.v:station.country::VARCHAR AS country
  , weather.t as date
  , weather.v:station.name::VARCHAR AS station
  , 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
FROM
    weather.isd_2019_daily weather
  , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
GROUP BY
    country, date, station
ORDER BY
    DATE;
```

### 15.3.13 Check the MIN and MAX temp field for unusually large or high numbers

Check the MIN and MAX temp field for unusually large or high numbers and note that there is occasionally a 999.9 value in these columns. This is because in some instances the check wasn't performed, or something went wrong during the check so the value wasn't recorded.

Let's add a filter in our WHERE clause that filters only for rows where all temperature quality checks were successfully completed and recorded. Run the code below to add the filter.

```
SELECT
    weather.v:station.country::VARCHAR AS country
  , weather.t as date
  , weather.v:station.name::VARCHAR AS station
  , AVG(observations.value:air.temp)::NUMBER(38,1) as avg_temp_c
  , MIN(observations.value:air.temp)::NUMBER(38,1) as min_temp_c
  , MAX(observations.value:air.temp)::NUMBER(38,1) as max_temp_c
FROM
    weather.isd_2019_daily weather
  , LATERAL FLATTEN(input => v:data.observations) observations
WHERE
    weather.t BETWEEN '2019-08-14' AND '2019-08-21'
    AND
    observations.value:air."temp-quality-code" = '1'
GROUP BY
    country, date, station
ORDER BY
```



```
DATE;
```

### 15.3.14 Add the Fahrenheit values

We basically have the query written. Now all we have to do is convert Celsius to Fahrenheit and apply the AVG, MIN and MAX functions to the results of the conversion:

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

The query has now been completed. Great job!

## 15.4 Key takeaways

- The table function FLATTEN can be used to access data within arrays in semi-structured data.
- When not working with an array, the following syntax can be used to fetch the value of a key-value pair without having to flatten any data: table.column:key1.key2. ... keyN.
- When working with semi-structured data that needs to be flattened and there is more than one top-level path, the PATH argument of the FLATTEN command can be used to determine which path to navigate.
- DESCRIBE RESULT can be used to determine the data types of an executed query's output columns.
- The PATH and KEY output values of the FLATTEN function can be leveraged to get the specific data required.

## 16 Determine Appropriate Warehouse Sizes

The purpose of this lab is to familiarize you with how different warehouse sizes impact the performance of a query.

This lab is immediately applicable to many personas. If you're a data analyst, you'll see how different warehouse sizes can save you and your business users valuable time. If you're a data engineer, you'll learn the importance of choosing appropriate warehouse sizes for your work. If you're a database admin, you'll learn how important it is to allocate appropriately-sized warehouses for particular groups of users.

### Learning Objectives:

- How to resize warehouses
- How to use the query profile to monitor query performance
- How to use the INFORMATION\_SCHEMA and QUERY\_HISTORY view to analyze query performance

### Scenario:

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.

Remember, scaling warehouse size up (for more complex queries) and down (for less complex queries) is a strategy for either making complex queries more performant or saving compute charges in the case of less complex queries. In this example, we introduce a query that requires a scaling up/down strategy (it has a lot of data, includes numerous joins and has more than one filter in the WHERE clause).

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select "Create Worksheet from SQL File".

Let's get started!

### 16.1 Run a sample query with an extra small warehouse

**NOTE:** You are going to run the same query several times but with a different size warehouse each time. Keep the Query Profile browser tab open each time you run a query so you can make comparisons later.

16.1.1 Navigate to **[Worksheets]**.

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

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

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

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

16.1.4 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';
```

16.1.5 Run the following query

This query will 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;
```

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

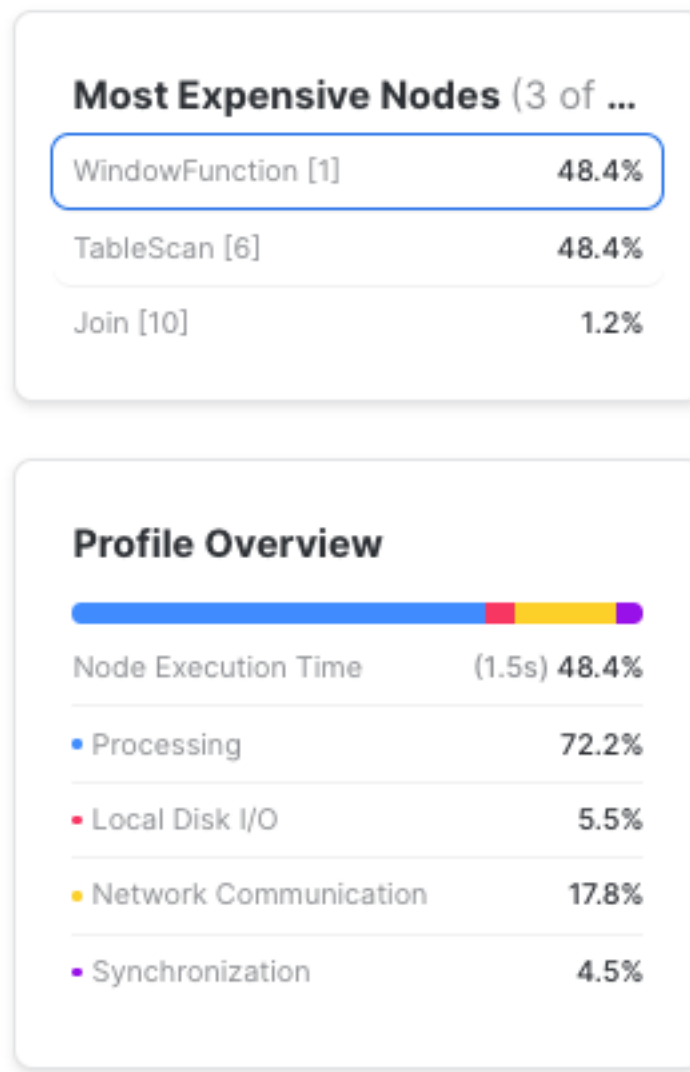
16.1.7 Take note of the performance metrics for this operator

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, and so it spilled to local SSD storage.

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.

16.1.8 Take note of the performance on the small warehouse.



**Figure 39:** Warehouse Performance

## 16.2 Run a sample query with a small warehouse

16.2.1 Suspend your warehouse, change its size to small, and resume it:

```
ALTER WAREHOUSE [login]_WH SUSPEND;
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = small WAIT_FOR_COMPLETION = TRUE;
ALTER WAREHOUSE [login]_WH RESUME;
```

16.2.2 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;
```

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

16.2.4 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.

### 16.3 Run a sample query with a medium warehouse

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

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

16.3.2 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;
```

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

16.3.4 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.

### 16.4 Run a sample query with a large warehouse

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

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

#### 16.4.2 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;
```

#### 16.4.3 Take note of the performance metrics for this operator.

You will see that there is no spilling to local or remote storage.

### 16.5 Run a sample query with an extra large warehouse

#### 16.5.1 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 WAIT_FOR_COMPLETION = TRUE;  
ALTER WAREHOUSE [login]_WH RESUME;
```

#### 16.5.2 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;
```

### 16.5.3 Note the performance.

### 16.5.4 Suspend your warehouse, and change its size to XSmall:

Now we're going to run some queries to analyze performance. We're going to change the warehouse size and remove the query tag from our session.

```
ALTER WAREHOUSE [login]_WH SUSPEND;  
ALTER WAREHOUSE [login]_WH SET WAREHOUSE_SIZE = XSmall WAIT_FOR_COMPLETION = TRUE;  
ALTER SESSION UNSET QUERY_TAG;
```

### 16.5.5 View query history results for the performance tests

Run a query showing the query history results for these performance tests using the table function INFORMATION\_SCHEMA.QUERY\_HISTORY\_BY\_SESSION()

```
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;
```

This query tells us how fast each query ran but not how many partitions were scanned. Let's take a look at partitions scanned and bytes spilled to local storage.

### 16.5.6 Dig further into query history results for performance tests

This query will show more information, but the ACCOUNT\_USAGE schema has a latency of up to 45 minutes 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;
```

As we can see, the large warehouse was the first one not to spill bytes to storage, which resulted in a dramatic increase on performance. While the X-Large warehouse also did not spill bytes to storage and ran the query even faster, you could select either size warehouse based on the speed with which you need to have the result returned to you. The thing to keep in mind however is that the extra large warehouse will burn more credits. So if you can live with a slightly slower performance in this instance, the Large warehouse is probably the right size.

### 16.5.7 Suspend the warehouse

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

## 16.6 Key Takeaways

- Scaling up can make complex queries more performant; scaling down for less complex queries can save you compute credits.
- You can analyze output from functions provided in the INFORMATION\_SCHEMA and the ACCOUNT\_USAGE.query\_history view to determine query performance.
- Bigger isn't always better when choosing a warehouse size. It's important to choose the right sized warehouse for the performance you need but not needlessly burn compute credits.



## 17 Introduction to Monitoring Usage and Billing

The purpose of this lab is to familiarize you with the Snowflake database, the schemas in the database, and how you can use that data to monitor how users are using the objects in your system and what the associated costs are with that usage.

### Learning Objectives:

- Monitor usage and billing with the ACCOUNT\_USAGE schema
- Determine which warehouses do not have resource monitors activated for them
- Determine the most expensive queries from the last 30 days
- Determine the top 10 queries with the most spillage to remote storage

### HOW TO COMPLETE THIS LAB

In order to complete this lab, you can type the SQL commands below directly into a worksheet. It is not recommended that you cut and paste from the workbook pdf as that sometimes results in errors.

You can also use the SQL code file for this lab that was provided at the start of the class. To open an .SQL file in Snowsight, make sure the Worksheet section is selected on the left-hand navigation bar. Click on the ellipsis between the Search and +Worksheet buttons. In the dropdown menu, select “Create Worksheet from SQL File”.

Let’s get started!

### 17.1 Snowflake Database

Snowflake provides a system-defined, read-only shared database named SNOWFLAKE that contains metadata and historical usage data about the objects in your organization and account.

The purpose of the database is to allow you to monitor object usage metrics as well as the costs associated with that usage so you can make any adjustments needed to get the most for the credits being spent.

Below is a list of the schemas that you can query to get the information you need about object usage and the associated costs.

**Snowflake Database Schemas**

**ACCOUNT\_USAGE:** Views that display object metadata and usage metrics for your account.

**CORE:** Contains views and other schema objects utilized in select Snowflake features. Currently, the schema only contains the system tags used by Data Classification. Additional views and schema objects will be introduced in future releases.

**DATA\_SHARING\_USAGE:** Views that display object metadata and usage metrics related to listings published in the Snowflake Marketplace or a data exchange.

**ORGANIZATION\_USAGE:** Views that display historical usage data across all the accounts in your organization.

**READER\_ACCOUNT\_USAGE:** Similar to ACCOUNT\_USAGE, but only contains views relevant to the reader accounts (if any) provisioned for the account.

Important: By default, only account administrators (users with the ACCOUNTADMIN role) can access the SNOWFLAKE database and schemas, or perform queries on the views; however, privileges on the database can be granted to other roles in your account to allow other users to access the objects.

Note: There is also a schema called INFORMATION\_SCHEMA. It is created by default and exists in every Snowflake database.

### 17.1.1 Links to more information about the Snowflake Database Schemas

Click any of the links below if you'd like to read more about the Snowflake database or the schemas listed above.

[Click here for Snowflake Database](#)

[Click here for ACCOUNT\\_USAGE](#)

[Click here for CORE and Data Classification](#)

[Click here for DATA\\_SHARING\\_USAGE](#)

[Click here for ORGANIZATION\\_USAGE](#)

[Click here for READER\\_ACCOUNT\\_USAGE](#)

## 17.2 Monitoring Usage and Billing with the ACCOUNT\_USAGE schema

The ACCOUNT\_USAGE schema supports usage and billing monitoring because it exposes a number of secure views that display data related to object usage history, grants to roles and users, data loading history, metering history, storage history, task history, users, roles and more.

ACCESS_HISTORY	METERING_DAILY_HISTORY	SESSIONS
AUTOMATIC_CLUSTERING_HISTORY	METERING_HISTORY	STAGES
COLUMNS	OBJECT_DEPENDENCIES	STAGE_STORAGE_USAGE_HISTORY
COPY_HISTORY	PIPES	STORAGE_USAGE
DATABASES	PIPE_USAGE_HISTORY	TABLES
DATABASE_STORAGE_USAGE_HISTORY	POLICY_REFERENCES	TABLE_CONSTRAINTS
DATA_TRANSFER_HISTORY	QUERY_HISTORY	TABLE_STORAGE_METRICS
FILE_FORMATS	REFERENTIAL_CONSTRAINTS	TAGS
FUNCTIONS	REPLICATION_USAGE_HISTORY	TAG_REFERENCES
GRANTS_TO_ROLES	ROLES	TASK_HISTORY
GRANTS_TO_USERS	ROW_ACCESS_POLICIES	USERS
LOAD_HISTORY	SCHEMATA	VIEWS
LOGIN_HISTORY	SEARCH_OPTIMIZATION_HISTORY	WAREHOUSE_EVENTS_HISTORY
MASKING_POLICIES	SEQUENCES	WAREHOUSE_LOAD_HISTORY
MATERIALIZED_VIEW_REFRESH_HISTORY	SERVERLESS_TASK_HISTORY	WAREHOUSE_METERING_HISTORY

**Figure 40:** Secure views present in the SNOWFLAKE.ACCOUNT\_USAGE schema

The main approach to writing queries that will help you monitor usage and billing is to look at the views that appear relevant and then run through the column list to see if they appear relevant to your goal. As there are many views in each schema, some with a lot of columns, it will take time and experience working with the schemas to become sufficiently familiar with them such that you know exactly how to get the answers you want off the top of your head.

17.2.1 To get started, create a new worksheet named *Warehouse Sizing* with the following context:

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

### 17.2.2 Credit Consumption by Warehouse

Monitoring credit consumption for specific objects is a classic use of data in the ACCOUNT\_USAGE schema. Here is an example of two queries that utilize the WAREHOUSE\_METERING\_HISTORY view:

```
USE SCHEMA SNOWFLAKE.ACCOUNT_USAGE;








-- Credits used (all time = past year)
SELECT WAREHOUSE_NAME
      ,SUM(CREDITS_USED_COMPUTE) AS CREDITS_USED_COMPUTE_SUM
FROM ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
GROUP BY 1
ORDER BY 2 DESC;
```

```
-- Credits used (past N days/weeks/months)
SELECT WAREHOUSE_NAME
      ,SUM(CREDITS_USED_COMPUTE) AS CREDITS_USED_COMPUTE_SUM
  FROM ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
 WHERE START_TIME >= DATEADD(DAY, -7, CURRENT_TIMESTAMP()) // Past 7 days
 GROUP BY 1
 ORDER BY 2 DESC;
```

These queries enable you to determine if there are specific warehouses that are consuming more credits than the others. You can then drill into questions such as: Should they be consuming that quantity of credits? Are there specific warehouses that are consuming more credits than anticipated?

In the event a warehouse is consuming too many credits you could take action to rectify the situation. Depending on what the warehouse is being used for, you could consider modifying the auto-suspend policy or the scaling policy, checking the data loading history to see if efficient practices are being used, or analyzing the size and efficiency of queries being run on the warehouse, or adding a resource monitor to the warehouse.

Below is a list of the columns in this schema.

NAME ↑	TYPE	NULLABLE	DEFAULT
 CREDITS_USED	NUMBER(38,9)	Yes	NULL
 CREDITS_USED_CLOUD_SERVICES	NUMBER(38,9)	Yes	NULL
 CREDITS_USED_COMPUTE	NUMBER(38,9)	Yes	NULL
 END_TIME	TIMESTAMP_LTZ(0)	Yes	NULL
 START_TIME	TIMESTAMP_LTZ(0)	Yes	NULL
 WAREHOUSE_ID	NUMBER(38,0)	Yes	NULL
 WAREHOUSE_NAME	VARCHAR(16777216)	Yes	NULL

**Figure 41:** Warehouse Metering History columns

### 17.2.3 Determining warehouses without resource monitors

If you have warehouses that are using too many credits, you can put resource monitors on them.

The query below identifies all warehouses without resource monitors in place. Resource monitors provide the ability to set limits on credits consumed against a warehouse during a specific time interval or date range. This can help prevent certain warehouses from unintentionally consuming more credits than typically expected.

```
SHOW WAREHOUSES;

SELECT "name" AS WAREHOUSE_NAME
      ,"size" AS WAREHOUSE_SIZE
  FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()))
 WHERE "resource_monitor" = 'null';
```

## 17.3 Billing Metrics

Billing metrics is all about analyzing what you've been billed in the past so you can determine if there is a way to lower costs in the future.

### 17.3.1 Most expensive queries from the last 30 days

The query below analyzes queries that are potentially too expensive by ordering the most expensive queries from the last 30 days. It takes into account the warehouse size, assuming that a 1 minute query on a larger warehouse is more expensive than a 1 minute query on a smaller warehouse.

```
WITH WAREHOUSE_SIZE AS
(
  SELECT WAREHOUSE_SIZE, NODES
    FROM (
      SELECT 'XSMALL' AS WAREHOUSE_SIZE, 1 AS NODES
      UNION ALL
      SELECT 'SMALL' AS WAREHOUSE_SIZE, 2 AS NODES
      UNION ALL
      SELECT 'MEDIUM' AS WAREHOUSE_SIZE, 4 AS NODES
      UNION ALL
      SELECT 'LARGE' AS WAREHOUSE_SIZE, 8 AS NODES
      UNION ALL
      SELECT 'XLARGE' AS WAREHOUSE_SIZE, 16 AS NODES
      UNION ALL
      SELECT '2XLARGE' AS WAREHOUSE_SIZE, 32 AS NODES
      UNION ALL
      SELECT '3XLARGE' AS WAREHOUSE_SIZE, 64 AS NODES
      UNION ALL
      SELECT '4XLARGE' AS WAREHOUSE_SIZE, 128 AS NODES
    )
),
QUERY_HISTORY AS
(
  SELECT QH.QUERY_ID
    ,QH.QUERY_TEXT
    ,QH.USER_NAME
    ,QH.ROLE_NAME
    ,QH.EXECUTION_TIME
    ,QH.WAREHOUSE_SIZE
  FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY QH
 WHERE START_TIME > DATEADD(month,-2,CURRENT_TIMESTAMP())
)

SELECT QH.QUERY_ID
  , 'https://' || current_account() ||
    '.snowflakecomputing.com/console#/monitoring/queries/detail?queryId='
    || QH.QUERY_ID AS QU
  ,QH.QUERY_TEXT
  ,QH.USER_NAME
  ,QH.ROLE_NAME
  ,QH.EXECUTION_TIME AS EXECUTION_TIME_MILLISECONDS
  , (QH.EXECUTION_TIME/(1000)) AS EXECUTION_TIME_SECONDS
```

```

, (QH.EXECUTION_TIME/(1000*60)) AS EXECUTION_TIME_MINUTES
, (QH.EXECUTION_TIME/(1000*60*60)) AS EXECUTION_TIME_HOURS
, WS.WAREHOUSE_SIZE
, WS.NODES
, (QH.EXECUTION_TIME/(1000*60*60))*WS.NODES as RELATIVE_PERFORMANCE_COST

FROM QUERY_HISTORY QH
JOIN WAREHOUSE_SIZE WS ON WS.WAREHOUSE_SIZE = upper(QH.WAREHOUSE_SIZE)
ORDER BY RELATIVE_PERFORMANCE_COST DESC
LIMIT 200;

```

This query gives you the chance to evaluate expensive queries and take some action. For example, you could look at the query profile, contact the user who executed the query, or take action to optimize these queries.

Below is a list of the columns in this secure view, SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY.

BYTES_DELETED	EXTERNAL_FUNCTION_TOTAL_RECEIVED_ROWS	QUEUED_PROVISIONING_TIME
BYTES_READ_FROM_RESULT	EXTERNAL_FUNCTION_TOTAL_SENT_BYTES	QUEUED_REPAIR_TIME
BYTES_SCANNED	EXTERNAL_FUNCTION_TOTAL_SENT_ROWS	RELEASE_VERSION
BYTES_SENT_OVER_THE_NETWORK	INBOUND_DATA_TRANSFER_BYTES	ROLE_NAME
BYTES_SPILLED_TO_LOCAL_STORAGE	INBOUND_DATA_TRANSFER_CLOUD	ROWS_DELETED
BYTES_SPILLED_TO_REMOTE_STORAGE	INBOUND_DATA_TRANSFER_REGION	ROWS_INSERTED
BYTES_WRITTEN	IS_CLIENT_GENERATED_STATEMENT	ROWS_PRODUCED
BYTES_WRITTEN_TO_RESULT	LIST_EXTERNAL_FILES_TIME	ROWS_UNLOADED
CLUSTER_NUMBER	OUTBOUND_DATA_TRANSFER_BYTES	ROWS_UPDATED
COMPILATION_TIME	OUTBOUND_DATA_TRANSFER_CLOUD	SCHEMA_ID
CREDITS_USED_CLOUD_SERVICES	OUTBOUND_DATA_TRANSFER_REGION	SCHEMA_NAME
DATABASE_ID	PARTITIONS_SCANNED	SESSION_ID
DATABASE_NAME	PARTITIONS_TOTAL	START_TIME
END_TIME	PERCENTAGE_SCANNED_FROM_CACHE	TOTAL_ELAPSED_TIME
ERROR_CODE	QUERY_ID	TRANSACTION_BLOCKED_TIME
ERROR_MESSAGE	QUERY_LOAD_PERCENT	USER_NAME
EXECUTION_STATUS	QUERY_TAG	WAREHOUSE_ID
EXECUTION_TIME	QUERY_TEXT	WAREHOUSE_NAME
EXTERNAL_FUNCTION_TOTAL_INVOCATIONS	QUERY_TYPE	WAREHOUSE_SIZE
EXTERNAL_FUNCTION_TOTAL_RECEIVED_BYTES	QUEUED_OVERLOAD_TIME	WAREHOUSE_TYPE

**Figure 42:** QUERY\_HISTORY view columns

### 17.3.2 Top 10 Queries With The Most Spillage to Remote Storage

Another way to evaluate the cost of queries is to see if they are spilling to remote storage. The query below allows you do do that.

```

select query_id, substr(query_text, 1, 50) partial_query_text, user_name,
       warehouse_name, warehouse_size,

```

```
        BYTES_SPILLED_TO_REMOTE_STORAGE, start_time, end_time,  
        total_elapsed_time/1000 total_elapsed_time  
from    snowflake.account_usage.query_history  
where   BYTES_SPILLED_TO_REMOTE_STORAGE > 0  
and     start_time::date > dateadd('days', -45, current_date)  
order  by BYTES_SPILLED_TO_REMOTE_STORAGE desc  
limit  10;
```

This query also provides the warehouse name and size. Once you identify the queries that are spilling to remote storage, you can take action to ensure they are run on larger warehouses with more local storage and memory.

## 17.4 Key Takeaways

- As there are many views in each schema, some with a lot of columns. It will take time and experience working with the schemas to become sufficiently familiar with them such that you know exactly how to get the answers you want off the top of your head.
- Resource monitors provide the ability to set limits on credits consumed against a warehouse during a specific time interval or date range.
- The Snowflake database enables you to determine where you have high credit consumption so you can ask pertinent questions as to why that is occurring. It then enables you to enact solutions that bring the cost down.