Snowflake Fundamentals

Workbook

Version 19K13

Table of Contents

Module 1: Architecture & Overview

Exercise 1.1: Take a Quick Test Drive

25 minutes

Task 1: Create Objects in the UI

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

1. Log in to your Snowflake account using the information provided by your instructor.
2. Locate the top ribbon, which has several icons across the top (**Databases**, **Shares**, etc.). These icons are used to activate different areas of the UI.
3. Navigate to **[Warehouses]** in the top ribbon.
4. Click **[Create]** above the list of warehouses. The **Create Warehouse** dialog box appears.
5. Fill in the fields with the information shown below to create a warehouse you will use to run queries:

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

   Then click **Finish**.

   In a production environment, you would likely have several warehouses - for example, one for each group, or one for each type of function (queries, loads, etc.). For this course, you will use just this one warehouse and change its size as needed for various functions.
6. Navigate to **[Databases]**.
7. Click **[Create]** to create a database. Name the database **[login]**_DB.
8. Create a table.
   a. In the left-side list, click the database you just created.

b. Click **[Create]** and create a table named **[login]_TBL**. Put it in the **Public** schema.

c. Click **[Add]** above the empty table to add a column. Create four columns with the attributes shown below, then click **Finish**.

NOTE: you must include the numbers in parentheses for the types that have them in the table below, or you will get an error:

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

Task 2: Create Objects Using SQL

1. Navigate to **[Worksheets]**. Click in the tab labeled **Worksheet 1**, and rename your worksheet *Test Drive*.

2. Drop the database you created in the first task:

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

Note that this will drop any tables in the database, as well.

3. You do not need a warehouse to create objects. Drop your warehouse, and then recreate your database ([login]_DB) using SQL:

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

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

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

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

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

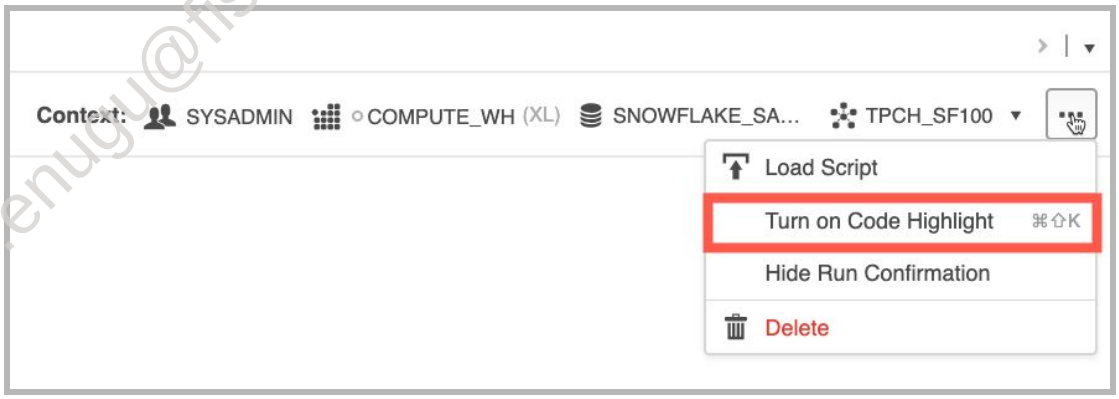
6. Use the following commands to set defaults for your role, database, schema, and warehouse. This will be referred to as your "standard context" throughout the rest of this workbook. Once you set these defaults, any worksheet you open will automatically set your context to these values.

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

7. Log out of the web UI, and back in. This forces your new user settings to be used.
8. Open a new worksheet, and verify that your standard context is automatically set.

Task 3: Run Queries on Sample Data

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



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

3. Right-click the schema name (TPCH_SF1) and select **Set as Context**. Verify that the new database and schema are now set in your context.

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

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

6. Above the results is a slider with **Data** and **Details**. **Data** should be selected by default. Select **Details** to view the detailed information on the column definitions.

7. In your worksheet, run these commands to explore the data.

```
--Show information about the tables in the selected
--database and schema:
SHOW TABLES;

--Count the number of rows in the table. Compare the result
--to the number of rows shown in the table pane on the left.
SELECT COUNT(*) FROM orders;

--Select 10 rows from the supplier table, to see what
--kind of data is in there
SELECT * FROM supplier LIMIT 10;

--Select the maximum value in the o_totalprice column
--of the orders table
SELECT MAX(o_totalprice) FROM orders;

--Run a query to see the total price of all orders, by order
--priority.
SELECT o_orderpriority, SUM(o_totalprice)
FROM orders
GROUP BY o_orderpriority
ORDER BY SUM(o_totalprice);

--Run the same query, but order by priority.
```

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

   Notice that there are multiple rows for each ps_partkey, with different values for

ps_suppkey and ps_availqty. This means that several suppliers stock each part key, and have different quantities available.

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

   How many total rows were returned by your query?
   How many total rows are in the PARTSUPP table?

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

    What are the highest and lowest prices returned?

Exercise 1.2: Storage and Compute

25 minutes

Task 1: Review the Training_DB Database

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

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

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

3. Return to the **Tables** tab. Review the table list.

4. Click the **LINEITEM** table located in the **TPCH_SF1** schema. Review the information about the table's structure.

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

6. Click the **LINEITEM** table that is located in the **TPCH_SF10** schema. Review the information about the table's structure. Compare the columns in this **LINEITEM** table in TPCH_SF1, with the columns in the **LINEITEM** table in TPCH_SF10.

   You will find that the columns are identical. What is the difference between these tables?

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

8. Sort the results by the **Table Name** column and find all the tables named **LINEITEM**. Compare their sizes to see the difference between the tables in the different schemas.

9. Navigate through the **Views**, **Schemas**, and **File Formats** tabs. Notice that all views and tables reside within a schema.

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

Task 2: Create and Organize Objects

1. Use the breadcrumb trail to go back to the **[Databases]** main page.

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

3. In the **Tables** tab, click **[Create]**. Take note of how the Create Table wizard requires that a Schema be selected.

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

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

5. Return to **[Databases]**. Toggle between the **Views**, **Stages**, **File Formats**, and **Sequences** tabs and click **[Create]** for each one to see how to create objects of these types. Cancel without creating anything for each of these object types. Snowflake enforces a logical hierarchy:

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

6. Open a new worksheet, and name it *Create Objects*. Verify that it automatically sets the context to your standard context.

7. Create a schema in your database, and name it [login]_SCHEMA. Specify the full database.schema path when you create it, then verify that it is set as the default schema in your worksheet.

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

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

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

Task 3: Review Storage Usage

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

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

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

3. Just above the display area on the right-hand side is a month indicator. Use the pull-down menu to change months and review the change in total storage over time.

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

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

   You will learn more about Stages and Fail Safe storage later in the course.

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

6. **INFORMATION_SCHEMA** is a schema that is automatically created for all databases. It contains database-specific information. You will learn more about this later.

   Use this to run a query to return average daily storage usage for the past 10 days, per database, for all databases in your account:

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

Task 4: Run Commands with No Compute Cost

1. Suspend your warehouse.

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

2. Change the worksheet context to the following.

   - Role: TRAINING_ROLE
   - Database: SNOWFLAKE_SAMPLE_DATA
   - Schema: TPCH_SF1

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

```
USE DATABASE SNOWFLAKE_SAMPLE_DATA;
USE SCHEMA TPCH_SF1;
```

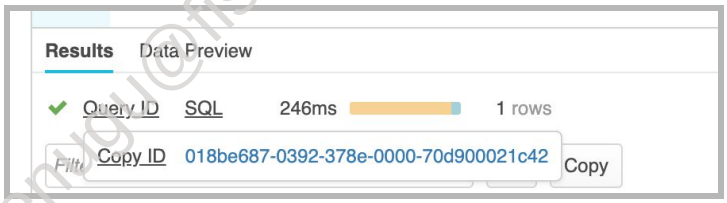
   From now forward, whenever this workbook instructs you to set your context, you can choose between the SQL method and the context menu method.

3. Execute the following query which selects MIN and MAX values for the **L_ORDERKEY** column and a row count for the **LINEITEM** table,

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

   Notice that the results return almost immediately.

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



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

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

Task 5: Work with Virtual Warehouses

1. Return to your worksheet.
2. Using SQL, create a warehouse named **[login]**_WH2, with the following characteristics. Take note of the various parameters you can set for warehouses:

   - Name: **[login]**_WH2
   - Size: XSmall
   - Auto Resume: True
   - Auto Suspend: 300
   - Minimum Clusters: 1
   - Maximum Clusters: 1
   - Initially Suspended: True

   Here is the query to run:

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

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

3. Execute the command below to list the configured warehouses (you will see many more warehouses than you have created - why?).

   Note also that the warehouse is in a SUSPENDED state.

```
SHOW WAREHOUSES;
```

4. Alter your warehouse to change the following parameters:

   Size: Small
   Auto Suspend: 600
   Maximum Clusters: 3

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

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

5. Navigate to **[Warehouses]**.
6. Locate your Warehouse and confirm the parameters are as follows:
   - Status: Suspended
   - Name: [login]_WH2
   - Size: Medium
   - Cluster: min: 1, max: 3
   - Auto Suspend: 10 minutes
   - Auto Resume: Yes
   - Owner: TRAINING_ROLE
7. Return to your worksheet and execute the following query:

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

8. Execute the following query:

```
SELECT *, (2+2) AS [login]
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.NATION;
```

9. Return to **[Warehouses]**.
10. Locate your warehouse and verify the Status is now “Started”.
11. Return to **[Worksheets]**.
12. Suspend your warehouse:

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

13. Show your warehouses and confirm that the warehouse is suspended.
14. Alter your warehouse and set the WAREHOUSE_SIZE to XSmall.
15. Suspend and then drop the warehouse.
16. Show warehouses to verify that [login]_WH is still listed, but [login]_WH2 is not.

Module 2: Clients, Connectors, & Ecosystems

Lab Purpose: You will install SnowSQL and run several queries directly through the command line tool. The final exercise provides you with practice running SQL scripts through the Snowflake command line tool.

Exercise 2.1: Work with SNOWSQL

35 minutes

Task 1: Install SnowSQL

1. In the UI, click **[Help]** and then **Download**… to display the Downloads Dialog.
2. Download the SnowSQL (CLI) client appropriate for your platform.
3. Follow the instructions and install SnowSQL on your machine.

Task 2: Run SQL Commands using SnowSQL

1. In a terminal or Windows console, start SnowSQL interactively:

```
snowsql -a [account] -u [login]
```

   In the command above, -a specifies the name of the Snowflake account you are connecting to (the instructor will provide you with the account name). The -u parameter specifies your user name.

2. Execute the following commands to set the context for your session:

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

3. Insert the following three rows into the table you created earlier:

```
INSERT INTO [login]_TBL
   VALUES(2, 'A','UK', '11/02/2005');
```

```
INSERT INTO [login]_TBL
   VALUES(4, 'C','SP', '11/02/2005');
INSERT INTO [login]_TBL
   VALUES(3, 'C','DE', '11/02/2005');
```

4. Insert several more rows using a single INSERT INTO statement:

```
INSERT INTO [login]_TBL VALUES
   (1, 'ORDERC007', 'JAPAN', '11/02/2005'),
   (7, 'ORDERF821', 'UK', '11/03/2005'),
   (12, 'ORDERB029', 'USA', '11/03/2005');
```

5. Query the data in [login]_TBL and order it by ID:

```
SELECT * FROM [login]_TBL
ORDER BY id;
```

6. Following the syntax above, insert four rows of data into the MEMBERS table you created earlier. Remember, the five columns are customer ID, first name, last name, membership start date, and membership level.

   HINT: Remember what schema this table is in!

7. Return all rows and columns in the table.

8. Exit the SnowSQL interface.

```
!exit
```

Task 3: Create and Use a Configuration File

1. Open the SnowSQL configuration file with the editor of your choice:

   Linux/Mac: `~/.snowsql/config`
   Windows: `%USERPROFILE%\.snowsql\config`

2. Uncomment the following lines that define your default account connection, and add the appropriate values. Then save and close the file:

```
accountname = [account]
```

```
username = [login]
dbname = [login]_DB
schemaname = PUBLIC
warehousename = [login]_WH
rolename = TRAINING_ROLE
```

3. Log back in without specifying a connection; it will use the default.

```
snowsql
```

4. Verify that you are logged in with the appropriate database, schema, and warehouse (they will be listed before the command prompt). Also verify your role.

```
SELECT CURRENT_ROLE();
```

5. Exit from SnowSQL.
6. Edit the configuration file again, and create a second connection called SYSADMIN that will automatically log you in with sysadmin as your role. Provide the password in the file as well. **NOTE:** The role SYSADMIN does not have access to your database and warehouse, so you don't need to include those in the connection definition.
7. Connect to SnowSQL using the SYSADMIN connection, and verify you are logged in as the SYSADMIN role.
8. Exit from SnowSQL.
9. Determine whether the connection name is case-sensitive.

Task 4: Run a Script in SnowSQL

1. Create a file called script.sql that contains the following:

```
USE ROLE TRAINING_ROLE;
USE WAREHOUSE [login]_WH;
USE DATABASE [login]_DB;
USE SCHEMA PUBLIC;
SELECT * FROM [login]_TBL;
```

2. Start SnowSQL, pass it your script, and supply your password when requested:

```
snowsql -f script.sql
```

NOTE: You must either be in the same location as your script when you run it, or you must provide the full path to the script.

You will see your output and then SnowSQL will exit.

Module 3: Query Data in Snowflake

Exercise 3.1: Work with Database Objects

45 minutes

Task 1: Work With Permanent, Temporary, and Transient Tables

1. Create a worksheet named *Work with Objects*, and use your standard context.
2. Create a table named **permanent**, with three columns (ID, first name, and last name). Make ID and INTEGER, first name a VARCHAR(20), and last name a VARCHAR(30).
3. Create a temporary table named **temp**, with the same three columns:

```
CREATE TEMPORARY TABLE...
```

4. Create a transient table named **transient**, with the same three columns:

```
CREATE TRANSIENT TABLE...
```

5. Show the tables with `SHOW TABLES`. Notice that the "kind" column tells you the type of each table.
6. Close the worksheet you are in, and open another.

   NOTE: For this step, it is important that you CLOSE your current worksheet, not just open another. Closing your current worksheet ends your session.

7. SHOW the tables. The temporary table is gone, but the transient table is still there.
8. Drop the transient table.
9. Add three rows of data to your **permanent** table.

   HINT: Use `INSERT INTO`. To refresh your memory about the columns in this table, you can first run `DESCRIBE TABLE permanent`.

10. Query the table to verify the columns and data.

11. Add a **comments** column, type STRING, to your **permanent** table.

```
ALTER TABLE permanent ADD COLUMN comments STRING;
```

12. Query the table to see what values are in the **comments** column.

13. Describe the table again. What do you notice about the **comments** column?

    The STRING data type you set for the column was actually set as a VARCHAR with the maximum number of characters.

14. You decide you don't want the comments column to be that long, so change the data type to VARCHAR(1000). What happens?

```
ALTER TABLE permanent MODIFY COLUMN comments
SET DATA TYPE VARCHAR(1000);
```

15. DROP the comments column, then re-add it as VARCHAR(1000).

16. DESCRIBE the table and verify the column definition is correct.

17. Create a table from `SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.ORDERS`, using $ notation to select columns 1, 3, 4, 5, and 6. Use AS to rename the columns. Name it **my_orders** (it will take a few minutes to create).

```
CREATE TABLE my_orders AS SELECT $1 AS key, $3 AS status,
$4 AS price, $5 AS date, $6 AS priority
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.ORDERS;
```

18. Query **my_orders** to see the columns. Limit the results to 10 rows.

19. Create a table using LIKE.

```
CREATE TABLE like_orders
LIKE SNOWFLAKE_SAMPLE_DATA.TPCH_SF1000.ORDERS;
```

20. Query the **like_orders** table to see what it contains, and then drop it.

    The LIKE modifier creates a table with the same column definitions as the source table, without copying any of the data.

Task 2: Work with Views

1. Create a view from the **my_orders** table you created in the previous task:

```
CREATE VIEW orders_view(status, date, price) AS
SELECT status, date, SUM(price) FROM my_orders
GROUP BY status, date;
```

2. Create a secure view with the same information:

```
CREATE SECURE VIEW s_orders_view...
```

3. Create a materialized view with the same information.

```
CREATE MATERIALIZED VIEW m_orders_view...
```

NOTE: A materialized view takes longer to create than a standard view. Why?

4. SHOW your views. Examine the columns to determine how you can tell what type of view it is.
5. See if you can create a view that is both secure and materialized.
6. Select all data from your **orders_view** view. Note how long it takes.
7. Suspend your warehouse (to clear any cache):

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

8. Select all data from your **s_orders_view** view, and note how long it takes.
9. Suspend your warehouse, then select all data from your **m_orders_view** view. Note how long it takes

   Which type of view was fastest? Which was the slowest?

10. Drop your views.

Task 3: Work with Sequences

1. Make sure your standard context is set.
2. Create a sequence:

```
CREATE SEQUENCE [login]_seq;
```

3. Create a table that will use the sequence:

```
CREATE TABLE [login]_seq_test (id NUMBER, comment
VARCHAR(20));
```

4. Insert rows into the table, using the sequence:

```
INSERT INTO [login]_seq_test VALUES
   ([login]_seq.nextval, 'Good Morning'),
   ([login]_seq.nextval, 'Good Afternoon'),
   ([login]_seq.nextval, 'Good Evening');
```

   In the statement above, `[login]_seq.nextval` uses the next value in the specified sequence.

5. View the table to see what was put in the ID column.
6. Alter your sequence to increment by 3, rather than by the default of 1.

```
ALTER SEQUENCE [login]_seq SET INCREMENT=3;
```

7. Insert three more rows into the table, using the sequence.
8. View the table to see what was put in the ID column.
9. Drop the sequence and the table.

Exercise 3.2: Practice Query Fundamentals

60 minutes

Task 1: SELECT

The SELECT statement sets up a tabular view of data and/or calculations.

In this task you will run several SELECT statements, with and without conditional logic, to see what they return in the query results.

1. Navigate to **[Worksheets]** and create a new worksheet named *Query Data*.
2. Perform basic calculations using SELECT, but no table data:

```
SELECT (22+47+1184), 'My String', CURRENT_TIME();
```

3. Use AS on the previous query to rename the first column to SUM:

```
SELECT (22+47+1184) AS sum, 'My String', CURRENT_TIME();
```

4. Use AS to rename all the columns.
5. Use some conditional logic using CASE and WHEN.

```
SELECT
  CASE
     WHEN RANDOM() > 0  THEN 'POSITIVE'
     WHEN RANDOM() < 0 THEN 'NEGATIVE'
     ELSE 'Zero'
   END,
   CASE
      WHEN RANDOM(2) > 0  THEN 'POSITIVE'
      WHEN RANDOM(2) < 0 THEN 'NEGATIVE'
      ELSE 'Zero'
   END;
```

6. Create a table named **test** with a single column of type NUMBER(4,2). Name the column **num**.
7. Insert the values 2.00, 2.57, 4.50, and 1.22 into the table.
8. Query the table to view the contents.

9. Query the table and CAST the **num** column as an INTEGER. Also rename the column to **value**.

   What happens when you CAST a NUMBER to an INTEGER?

10. Change your worksheet context to use SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.
11. Select the columns r_regionkey, r_name, and r_comment from the REGION table.
12. Select the number of distinct values in the r_name column.
13. Display the distinct values in the r_name column.

Task 2: WHERE and LIMIT

1. Select 10 rows form the ORDERS table.
2. Select all the DISTINCT values from o_orderstatus.
3. Select all rows from the ORDERS table that have an order status of 'F'.

```
SELECT * FROM orders WHERE o_orderstatus = 'F';
```

4. Count all the rows from the ORDERS table that have an order status of 'P.'
5. Count all the rows from the ORDERS table that have an order status of either 'P' or 'O.'
6. Return all rows from the ORDERS table with an order date of July 21, 1992.
7. Write a single query that returns all rows from the ORDERS table for the customer who has an account balance of 3942.58. Do not use a JOIN.

   HINT: You will need to reference the CUSTOMER table in your WHERE clause.

8. Run the following command to turn off query caching (which you will learn more about later):

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

9. Run the following command several times. Do you get the same result each time?

```
SELECT * FROM lineitem LIMIT 3;
```

10. Run the following command several times. Do you get the same result each time?

```
SELECT * FROM nation LIMIT 3;
```

11. Run the SELECT on the LINEITEM table (step 9) 12 times. For each result, record the first three digits of the l_orderkey value.

    Why do you think the two tables have different behaviors with LIMIT? What is different about the two tables?

Task 3: GROUP BY, HAVING, and ORDER BY

1. Show the sum of orders (based on total price) for each order status in the ORDERS table.

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

2. Show the sum of orders (based on total price) by date, for dates before January 1, 1998. Sort the results by order date (most recent first).
3. Show the minimum, maximum, and average account balance by market segment, from the CUSTOMER table.
4. Using the SUPPLIER table, show the total account balance for each nation (using s_nationkey) with a balance in excess of $2 million.
5. Select all rows from the ORDERS table, ordering them first by priority and then by total price (with the highest total price listed at the top).
6. Using the PART table, determine how many parts are packaged in each of the available types of containers.
7. Set your schema back to **[login]**_DB.PUBLIC to prepare for the next task.

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

Task 4: JOIN

1. Create a table called MYLEFT using the following command.

```
CREATE TABLE myleft(id INTEGER, name VARCHAR(6), num FLOAT);
```

2. Insert the following values in to the table. Notice the column types and values.

```
INSERT INTO myleft VALUES
   (1, 'Thanos', 10.012),
   (2, 'Bess', 3.00),
   (3, 'Tucker', 5),
   (4, 'Moana', 17.003),
   (5, 'Hobson', 123.42124),
   (6, 'Kitty', 14);
```

3. Query MYLEFT to verify the contents.

4. Create a table called MYRIGHT and insert some rows, using the following commands. Again, note the column types and values.

```
CREATE TABLE myright(id NUMBER(4,2), title VARCHAR(25), num
INTEGER);

INSERT INTO myright VALUES
   (1, 'Endgame', 13),
   (2, 'Porgy and Bess', 3),
   (3, 'Tucker & Dale vs Evil', 7),
   (4, 'Moana', 5),
   (5, 'Arthur', 14),
   (6, 'Gunsmoke', 22);
```

5. Query MYRIGHT to verify the contents.

6. JOIN MYRIGHT to MYLEFT, on the ID columns. Return all columns.

```
SELECT * FROM myleft
JOIN myright ON myleft.id = myright.id;
```

7. Run the same query, but only include the ID and NAME columns from MYLEFT, and the TITLE column from MYRIGHT.

8. JOIN MYRIGHT to MYLEFT, where MYLEFT.ID equals MRIGHT.NUM.

9. JOIN MYRIGHT to MYLEFT, where MYLEFT.NAME equals MYRIGHT.TITLE.

There are many kinds of JOINs; explore a few using the same data. A JOIN, also called an INNER JOIN, returns all the records where the key exists in both tables. Theses are the JOINs you are tried so far.

10. A LEFT OUTER JOIN returns all of the rows in the left table, and any rows in the right table that match on the specified key. When the right table does not match on the key, NULL values are returned.

    Try a LEFT OUTER JOIN on the two tables to see the output.

```
SELECT * FROM myleft
LEFT OUTER JOIN myright ON myleft.id = myright.num;
```

11. A RIGHT OUTER JOIN returns all the rows in the right table, and any rows in the left table that match on the specified key. When the left table does not match the right table on the key, NULL values are returned.

    Try a RIGHT OUTER JOIN on the two tables, on the same keys, to see the output.

    HINT: Use the same syntax, just replace LEFT OUTER JOIN with RIGHT OUTER JOIN.

12. A FULL JOIN returns all of the records from both tables. The values will be NULL where the keys do not match.

    Try a FULL JOIN on the tables, on the same keys, to see the output. How many rows did you get? Why?

Task 5: PIVOT

1. Set your standard context in your worksheet.

2. Create a table and insert some data with the following statements:

```
CREATE TABLE weekly_sales(name VARCHAR(10), day VARCHAR(3),
```

```
amount NUMBER(8,2));
INSERT INTO weekly_sales VALUES
   ('Fred', 'MON',  913.24), ('Fred', 'WED', 1256.87),
   ('Rita', 'THU',   10.45), ('Mark', 'TUE',  893.45),
   ('Mark', 'TUE', 2240.00), ('Fred', 'MON',   43.99),
   ('Mark', 'MON',  257.30), ('Fred', 'FRI', 1000.27),
   ('Fred', 'WED',  924.34), ('Rita', 'WED',  355.60),
   ('Rita', 'MON',  129.00), ('Fred', 'WED', 3092.56),
   ('Fred', 'TUE',  449.00), ('Mark', 'MON',  289.12),
   ('Fred', 'FRI',  900.57), ('Rita', 'THU', 1200.00),
   ('Fred', 'THU', 1100.95), ('Fred', 'MON',  523.33),
   ('Fred', 'TUE',  972.33), ('Fred', 'MON', 4500.87),
   ('Fred', 'WED',   35.90), ('Rita', 'MON',   28.90),
   ('Mark', 'FRI', 1466.02), ('Fred', 'MON', 3022.45),
   ('Mark', 'TUE',  256.88), ('Fred', 'MON',  449.00),
   ('Rita', 'FRI',  294.56), ('Fred', 'MON',  882.56),
   ('Fred', 'WED', 1193.20), ('Rita', 'WED',   88.90),
   ('Mark', 'WED',   10.37), ('Fred', 'THU', 2345.00),
   ('Fred', 'TUE', 2638.76), ('Rita', 'TUE',  988.26),
   ('Fred', 'THU', 3400.23), ('Fred', 'MON',  882.45),
   ('Rita', 'THU', 734.527), ('Rita', 'MON', 6011.20),
   ('Fred', 'FRI',  389.12), ('Fred', 'THU',  893.45),
   ('Mark', 'WED', 2900.13), ('Mark', 'MON',  610.45),
   ('Fred', 'FRI',   45.69), ('Rita', 'FRI', 1092.35),
   ('Mark', 'MON',   12.56);
```

3. Query the table to view the data.

4. Run a query with PIVOT that will list each employee in a row, and the total sales for each day of the week as columns.

```
SELECT * FROM weekly_sales
PIVOT (SUM(amount) FOR day IN
('MON','TUE','WED','THU','FRI'));
```

5. Run a query with PIVOT that will list the average sales instead of total sales, and have the days of the week as rows and each name as a column.

Task 6: Subqueries

1. Change the SCHEMA in your context to SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.

2. The IN statement defines a set of values. Run the following query to see how it works:

```
SELECT o_custkey, SUM(o_totalprice) FROM orders
WHERE o_custkey IN (1, 7, 10)
GROUP BY o_custkey;
```

3. You do not have to provide a list of values for IN: you can specify a subquery to return the values for the IN statement. Use IN and a subquery to return all orders from customers who are in Saudi Arabia.

   HINT: Look at the NATION table to find the nation key for Saudi Arabia.

4. Return all columns from the LINEITEM table for orders where the status is 'P' and the customer key is 4.

5. Using a subquery, select all records from the weekly_sales table with an extended price greater than the lowest-priced item from my_orders.

   HINT: Remember where the my_orders table is?

Module 4: Data Movement

Lab Purpose: Learn how to load data into Snowflake using external stages, file formats, and transforming data upon load. Several of the exercises show you how to troubleshoot data loading errors and to validate data.

Exercise 4.1: Load Structured Data Using the UI

25 minutes

Task 1: Create Tables and File Formats

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

```
0|AFRICA|lar deposits. blithely final packages cajole. regular waters
     are final requests. regular accounts are according to |


1|AMERICA|hs use ironic, even requests. s|


2|ASIA|ges. thinly even pinto beans ca|


3|EUROPE|ly final courts cajole furiously final excuse|


4|MIDDLE EAST|uickly special accounts cajole carefully blithely close
     requests. carefully final asymptotes haggle furiousl|
```

Note that there is a delimiter at the end of every line, which by default is interpreted as an additional column by the `COPY INTO` statement.

1. Navigate to **[Worksheets]** and create a new worksheet named *Load Structured Data.*
2. Download and unzip the data file that was provided at the same link as your workbook, if you haven't already done that.
3. Load the `data_loading-create_tables.sql` script into your worksheet.
4. Review the scripts that create the various tables.
5. Execute all of the CREATE TABLE statements.

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

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

7. 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.

Task 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 SCHOL schema of the DBHOL database. In this task you will review the files in the stage, and load them using the file formats you created.

1. Review the properties of the stage:

```
DESCRIBE STAGE DBHOL.SCHOL.AWS_LOAD1;
```

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

2. Confirm the file is in the external stage with the list command:

```
LIST
@dbhol.schol.aws_load1/load/lab_files/  PATTERN='.*region.*';
```

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 @dbhol.schol.aws_load1/load/lab_files/
FILES = ( 'region.tbl' )
FILE_FORMAT = ( FORMAT_NAME = MYPIPEFORMAT );
```

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;
```

Task 3: Load Text Files from an External Stage

This exercise will load tables from text files that are in an external stage.

1. Navigate to **[Databases]**.
2. Select the **DBHOL** Database.
3. Select the **Stages** tab from the list above the display area, and confirm you can see the **AWS_LOAD1** Stage in the **SCHOL** Schema. This will show the stage, but does not show you as much information as the DESCRIBE STAGE command you used earlier.
4. Navigate back to **[Worksheets]**.
5. List some other files in the **DBHOL.SCHOL.AWS_LOAD1** stage:

```
LS @dbhol.schol.aws_load1/load/TCPH_SF10/;
```

For some of the larger tables, there are many files to load. For these larger tables, you will get better performance using more cores. Therefore, for this load exercise you will use a large warehouse.

6. Increase the size of the warehouse to Medium:

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

7. Load the **CUSTOMER** table into the **PUBLIC** Schema using the same File Format you used to load REGION in the first exercise. Note that you will use the fully qualified name pointing to a table in the database that you created. The FROM data is the Snowflake sample data stored on the external stage.

```
COPY INTO [login]_DB.PUBLIC.CUSTOMER
FROM @dbhol.schol.aws_load1/load/TCPH_SF10/CUSTOMER/
FILE_FORMAT = (FORMAT_NAME = [login]_DB.PUBLIC.MYPIPEFORMAT)
ON_ERROR = 'CONTINUE';
```

8. Load the remaining tables (shown below) by running the same command as above, but replacing CUSTOMER where it appears with the name of the table you want to load:
   - LINEITEM
   - NATION
   - ORDERS
   - PART
   - PARTSUPP
   - SUPPLIER

9. Explore the data in some of the tables you loaded.

```
SELECT * FROM lineitem LIMIT 100;
SELECT * FROM nation;
SELECT * FROM orders LIMIT 100;
```

Task 4: Load a GZip Compressed File

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

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

1. Change the size of **[login]**_WH to Small.
2. Empty the REGION Table in the PUBLIC schema of **[login]_DB**.

```
TRUNCATE TABLE region;
```

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

```
LIST @dbhol.schol.aws_load1/load/lab_files/
PATTERN='.*region.*';
```

4. 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'.
5. Query the table to view the data.

This is a good example of how to look for a file on a stage using patterns, and then load the file into a Snowflake table by naming the file(s) to be loaded.

Exercise 4.2: VALIDATION_MODE and ON_ERROR

10 minutes

Task 1: Detect File Format Problems with VALIDATION_MODE

This exercise will use Snowflake's VALIDATION_MODE option on a COPY statement to demonstrate Snowflake's pre-load error detection mechanism.

1. Open a new worksheet and name it *Validate Data*.
2. TRUNCATE the REGION table to remove the data it contains.
3. Run a COPY command in validation mode against region_bad_1.tbl from the external stage into the REGION table, and identify the issue that will cause the load to fail:

```
COPY INTO region
FROM @dbhol.schol.aws_load1/load/lab_files/
  FILES = ( 'region_bad_1.tbl' )
  FILE_FORMAT = ( FORMAT_NAME = MYPIPEFORMAT )
  VALIDATION_MODE = RETURN_ALL_ERRORS;
```

   What happened?

4. To see what happened, expand the ERROR column in the query results. To see the contents of that row, scroll all the way to the right and click the linked text in the REJECTED_RECORD column. You can see in the details that first column, which should contain a number, is the character x.
5. Run a COPY command in validation mode against region_bad_2.tbl from the external stage into the REGION table, and identify the issue that will cause the load to fail.

   Why did it fail? And what does the data look like?

Task 2: Load Data with ON_ERROR Set to CONTINUE

This exercise will use Snowflake's optional ON_ERROR parameter on the COPY command to define the behavior Snowflake should exhibit if an error is encountered when loading a file.

1. Run a COPY command with the ON_ERROR parameter set to CONTINUE against region_bad_1.tbl from the external stage into the REGION Table:

```
COPY INTO REGION FROM @dbhol.schol.aws_load1/load/lab_files/
  FILES = ( 'region_bad_1.tbl' )
  FILE_FORMAT = ( FORMAT_NAME = MYPIPEFORMAT )
  ON_ERROR = CONTINUE;
```

In the query results pane, you will see that the status is PARTIALLY_LOADED.

2. Query the data that was loaded, and confirm that all rows were loaded except the row that would not load according to the validation mode: you should see that the row with REGIONKEY 1 is missing.
3. Empty the REGION table again.
4. Run a COPY command with the ON_ERROR parameter set to CONTINUE against region_bad_2.tbl.
5. View the data that was loaded and confirm that all rows were loaded except the row the VALIDATION_MODE against this file stated would not load (row 2). This time, REGIONKEY 2 is missing.

Task 3: Reload the Region Table with Clean Data

1. TRUNCATE the REGION table.
2. Validate that you have data in the table stage for your region table.

```
LIST @dbhol.schol.aws_load1/load/lab_files/
PATTERN = '.*region.*';
```

3. Load the data to the REGION table. You can load it either from the uncompressed file, or from the region.tbl.gz file.
4. View the data that was loaded and confirm that all 5 rows were loaded.
5. Navigate to **[Databases]**.
6. Select **[login]**_DB Database and confirm that every table has data loaded.

Exercise 4.3: Unload Structured Data

30 minutes

Task 1: Unload a Pipe-Delimited File to an Internal Stage

1. Open a worksheet.
2. Select all records from the REGION table, in **[login]**_DB.PUBLIC.
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 help with the syntax. You will use MYPIPEFORMAT for the unload.
4. List the stage and verify that the data is there.
5. Use GET to download the file to your local system. Open it with an editor and see what it contains. **NOTE:** The GET command is not supported in the GUI; use SnowSQL.
6. Remove the file from the REGION table's stage.

Task 2: Unload Part of a Table

1. Create a new table from SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS.

```
CREATE TABLE new_orders AS
   SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS;
```

2. Unload the columns o_orderkey, o_orderstatus, and o_orderdate from your new table, into the table's stage. Remember – a table stage is automatically created for every table. Use the default file format.
3. Verify the output is in the stage.
4. Use GET to download the file to your local system, and review it. How many files did you get? At what point did COPY INTO decide to split the files?
5. Remove the files from the stage.
6. Repeat the unload with the selected columns, but this time specify SINGLE=TRUE in your COPY INTO command. You might want to search the documentation for this option to get more information on its use. Also provide a name for the output file as part of the COPY INTO.

7. Use GET to download the file to your local system, and review it.
8. Remove the file from the stage.

Task 3: JOIN and Unload a Table

1. Do a SELECT with a JOIN on your MYLEFT and MYRIGHT tables. You can JOIN on any column you wish. Review the output from your JOIN.
2. Create a named stage (you can call it whatever you want).
3. Unload the JOINed data into the stage you created.
4. Verify the file is in the stage.
5. Use GET to download the file to your local system and review it.
6. Remove the file from the stage.
7. Remove the stage.

Module 5: Snowflake Functions

Exercise 5.1: Snowflake Functions

25 minutes

Snowflake has a large built-in library of functions. In this lab you will get the opportunity to work with some of them.

Task 1: Scalar Functions

1. Open a worksheet and name it *Functions*.

2. Run the statements below to create a table called bits, and add some values. Then select the data using BITAND, BITOR, and BITXOR. These functions operate on equal-length bit patterns

   NOTE: the "--" characters on some of the lines below indicate that what follows is a comment.

```
CREATE TABLE bits (id integer, bit1 integer, bit2 integer);
INSERT INTO bits (id, bit1, bit2) VALUES
    (   11,    1,     1),    -- Bits are all the same
    (   24,    2,     4),    -- Bits are all different
    (   42,    4,     2),    -- Bits are all different
    ( 1624,   16,    24),    -- Bits overlap
    (65504,    0, 65504),    -- All but the lowest 6 bits
    (    0, null,  null);    -- No bits

SELECT bit1, bit2, BITAND(bit1, bit2), BITOR(bit1, bit2),
BITXOR(bit1, bit2)
FROM bits;
```

3. Run the following statements using the conditional function IFF to see what it does:

```
SELECT IFF((2+2=4), 'true', 'false');
SELECT IFF((2+2=5), 'true', 'false');

SELECT bit1, IFF(bit1>5, 'greater than 5', 'less than 5')
FROM bits;
```

4. Use a CASE expression to categorize what percent of total elapsed time was spent on query compile:

```
SELECT query_id, query_text,
CASE
   WHEN compilation_time >= .7*total_elapsed_time
      THEN '1:  70%'
   WHEN (compilation_time >= .4)
      AND (compilation_time < .7*total_elapsed_time)
      THEN '2:  40%'
 ELSE '3:  less than %40'
END AS compilation_percent
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE query_text LIKE 'select%'
AND query_text NOT LIKE 'select 1%'
ORDER BY compilation_percent;
```

5. 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;
```

6. Run this query to use some time and date functions:

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

7. Snowflake has many string functions. Here is one you can run:

```
SELECT 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;
```

8. 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;
```

Task 2: 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. This capability was shown earlier.

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

1. Use WINDOW aggregate functions:

```
SELECT DAYOFMONTH(start_time), start_time, end_time,
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  ORDER
BY warehouse_name;
```

2. Query the query history to produce numeric aggregates on query total times:

```
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;
```

Task 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.

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;
```

2. Use the RESULT_SCAN function to return the last result set.

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

3. Use the SHOW command with the RESULT_SCAN function to have SQL generate a list of commands to describe tables:

```
SHOW TABLES;

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

   Note that Snowflake's SQL is generally case-insensitive: it changes everything to upper case unless it is enclosed in quotes.

   In this example, the "name" column must be lower case, so it is quoted.

4. Use the SYSTEM$WHITELIST function to see information on hosts that should be whitelisted for Snowflake to work. Click on the item in the first row to see what the column contains.

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

   For an easier-to-read version, use the FLATTEN table function to flatten the data, which is in a semi-structured format.

```
SELECT VALUE:type AS type,
       VALUE:host AS host,
```

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

Exercise 5.2: High-Performing Functions

40 minutes

Task 1: Approximate Count Functions

1. Navigate to **[Worksheets]** and create a worksheet named *Function Junction*.
2. Set the Worksheet contexts as follows:
    a. ROLE: TRAINING_ROLE
    b. WAREHOUSE: **[login]**_WH
    c. DATABASE: SNOWFLAKE_SAMPLE_DATA
    d. SCHEMA: TPCH_SF100
3. Change the virtual warehouse size to XSmall, then suspend and resume the warehouse to clear any data in the warehouse cache.

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

4. Use the query below to determine an approximate count with Snowflake's Hyperloglog high-performing function:

```
SELECT HLL(l_orderkey) FROM lineitem;
```

5. Suspend and resume the warehouse again to clear the data cache.
6. Execute the regular COUNT version of the query:

```
SELECT COUNT(DISTINCT l_orderkey) FROM lineitem;
```

7. Compare the execution time of the two queries in steps 4 and 6. Note that the HLL approximate count version is much faster than the regular count version.

Task 2: Percentile Estimation Functions

The APPROX_PERCENTILE function is the more efficient version of the regular SQL MEDIAN function

1. Change your warehouse size to XLarge.
2. First start by using the SQL Median Function below.

   NOTE: This query will take about 20 minutes to run, so if you have run out of coffee, this might be a good time for a refill...

```
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10tcl;

SELECT MEDIAN(ss_sales_price), ss_store_sk
FROM store_sales
GROUP BY ss_store_sk;
```

3. Review the value returned, as well as the time it took to complete.
4. Now run the approximate percentile query to find the approximate 50th percentile:

```
SELECT APPROX_PERCENTILE(ss_sales_price, 0.5), ss_store_sk
FROM store_sales
GROUP BY ss_store_sk;
```

5. Review the time it took to complete, and the value returned. Not only was it faster, but it produced a result almost identical to that of MEDIAN.
6. Change your warehouse size to XSmall.

Exercise 5.3: User-Defined Functions and Stored Procedures

30 minutes

Task 1: Create a JavaScript User-Defined Function

1. Open a new worksheet and name it *UDFs.*
2. Create a table for this task:

```
CREATE TABLE D2B(i INT);
```

3. Insert values into the table:

```
INSERT INTO D2B VALUES (-60000),(-1),(1),
(22),(2000),(100000),(123123123);
```

4. Check the values you just inserted:

```
SELECT * FROM D2B;
```

5. Run the following to create a UDF named int_to_bin:

```
CREATE FUNCTION int_to_bin(D float,BYTES float)
RETURNS BINARY
LANGUAGE JAVASCRIPT
AS '
var byteArray=new Uint8Array(BYTES);
for(var x=0;x<BYTES;x++){
  byteArray[x]=(D>>>(x*8))&0xFF;
}
return byteArray;
';
```

6. Call the UDF just created as part of a SELECT statement, returning the byteArray for different size arrays using the value of the column labeled i:

```
SELECT i,int_to_bin(i,1),int_to_bin(i,2),int_to_bin(i,4)
FROM d2b;
```

Task 2: Create a SQL User-defined Function

1. Create a table called purchases:

```
CREATE TABLE purchases
(number_sold integer, wholesale_price number(7,2),
retail_price number(7,2));
```

2. Insert data into the purchases table:

```
INSERT INTO purchases (number_sold, wholesale_price,
retail_price) VALUES (3, 10.00, 20.00),(5, 100.00, 200.00);
```

3. Now create a scalar function called profit, to calculate the profit based on number of items sold, the retail price, and the wholesale price. It is a scalar function because it returns a single value. After creating the UDF, use it to query:

```
CREATE FUNCTION profit()
RETURNS numeric(11, 2) AS
$$
SELECT SUM((retail_price - wholesale_price) * number_sold)
FROM purchases
$$;

SELECT profit();
```

Task 3: CREATE a Stored Procedure

1. Create the source table:

```
CREATE TABLE servicedetail (ro VARCHAR,
rodate VARCHAR, service VARCHAR, rev VARCHAR, cost VARCHAR);
```

2. Insert data into the source table:

```
INSERT INTO servicedetail VALUES
('183297','2018-01-01','DGR123|OIL543|FLT1241','18.67|43.23|10.
87','11.17|22.11|4.45');
INSERT INTO servicedetail VALUES
('183298','2018-01-02','BFR432|BRK132','11.43|41','7.17|18.11')
;
```

3. View the contents of servicedetail:

```
SELECT * FROM servicedetail;
```

4. Create the target table, rodetail:

```
CREATE TABLE rodetail (ro NUMBER, rodate DATE, service
VARCHAR, revenue FLOAT, cost FLOAT);
```

5. Create a Javascript stored procedure named parseservicedata(). 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 two SQL statements. The first statement returns a result set. The stored procedure then walks the result set and dynamically creates an INSERT statement that is executed with values from each row returned by the first stored procedure.

```
CREATE PROCEDURE parseservicedata()
RETURNS varchar not null
LANGUAGE javascript
AS
    $$
  //Get the input data and start a cursor
    sql_cmd = "SELECT * FROM SERVICEDETAIL";
    var stmt = snowflake.createStatement(
    {sqlText: sql_cmd} );
    var rs = stmt.execute();
  //Declare target variable for RO number and close date

    var RO;
    var RODate;
  //Declare variables that will get each delimited
  //cell of data before it is split.
    var Service;
    var Rev;
    var Cost;
  //Declare target arrays for the Service Codes
  //and Revenue and Cost for each Service
    var ServiceValues = [];
    var RevValues = [];
    var CostValues = [];
  //Declare variables for creating the query for
  //inserting data into our new table, RODETAIL
```

```
    var appendstmt1 = "INSERT INTO RODETAIL VALUES ("
    var appendstmt2;
 //Iterate through the records in the input data
    while (rs.next())
    {
        RO = rs.getColumnValue('RO');
        RODate = rs.getColumnValue('RODATE');
      // Get an array of the different services performed.
        Service = rs.getColumnValue('SERVICE');
        ServiceValues = Service.split("|");
        Rev = rs.getColumnValue('REV');
        RevValues = Rev.split("|");
//Get an array of the revenue for the costs
     //of the services performed.
        Cost = rs.getColumnValue('COST');
        CostValues = Cost.split("|");
//Loop through the Services array and add a record
     //to a new table for each individual service,
     //keeping the RO number and Date constant
        var arrayLength = ServiceValues.length;
        for (var i = 0; i < arrayLength; i++) {
            appendstmt2 = RO
                        + ", '"
                        + RODate
                        + "', '"
                        + ServiceValues[i]
                        + "', "
                        + RevValues[i] + ","
                        + CostValues[i] + ")";
             snowflake.execute( {sqlText:
                        appendstmt1+appendstmt2} );
          }
                  }
    return 1;
              $$;
```

6. Call the stored procedure executing the two queries:

```
CALL parseservicedata();
```

7. View the results of calling the procedure on the rodetail table which you created:

```
SELECT * FROM rodetail;
```

Module 7: Access Control and User Management

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

Exercise 7.1: Managing Roles

20 minutes

Task 1: Determine Privileges (GRANTs)

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

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

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

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

Task 2: Create Parent and Child Roles

1. Change your role to SECURITYADMIN.

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

```
CREATE ROLE [login]_parent;
GRANT ROLE [login]_parent TO ROLE sysadmin;
CREATE ROLE [login]_child;
GRANT ROLE[login]_child TO ROLE sysadmin;
```

3. Give yourself privileges to the roles:

```
GRANT ROLE [login]_parent to USER [login];
GRANT ROLE [login]_child to USER [login];
```

4. Grant object permissions to the child role:

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

5. Use the child role to create a table:

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

6. Verify that you can see the table:

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

7. Use the parent role and view the table:

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

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

8. Change back to the SECURITYADMIN role and build the hierarchy:

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

9. Verify that the parent can now see the table created by the child.

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

10. Drop the child and parent roles.

HINT: What role needs to create and drop roles?

Module 8: Semi-Structured Data

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

Exercise 8.1: Load Semi-Structured Data

15 minutes

Task 1: Load Semi-Structured Parquet Data

This exercise will load a Parquet data file using different methods. You will small single file examples, so you will use a small warehouse.

1. Open a new worksheet and title it *Semi-Structured Data*.
2. Change your warehouse size to Small.
3. Empty the REGION table.
4. Create a file format called MYPARQUETFORMAT. Set the TYPE to PARQUET, and COMPRESSION to NONE:

```
CREATE FILE FORMAT MYPARQUETFORMAT
  TYPE = PARQUET
  COMPRESSION = NONE;
```

5. Confirm that the region.parquet file is in the external stage:

```
LIST
@dbhol.schol.aws_load1/load/lab_files PATTERN='.*region.*';
```

6. Query the data in the region.parquet file, without loading it first:

```
SELECT
   $1,
   $1:_COL_0::number,
   $1:_COL_1::varchar,
```

```
   $1:_COL_2::varchar
FROM @dbhol.schol.aws_load1/load/lab_files/region.parquet
(FILE_FORMAT => MYPARQUETFORMAT);
```

Note the alternate syntax for specifying the file format. You can either use the syntax above, or `FILE FORMAT = (FORMAT_NAME = MYPARQUETFORMAT)`. They are equivalent.

7. Reload the REGION table from the region.parquet file:

```
COPY INTO region FROM (SELECT $1:_COL_0::number,
                              $1:_COL_1::varchar,
                              $1:_COL_2::varchar
FROM @dbhol.schol.aws_load1/load/lab_files/region.parquet )
FILE_FORMAT = (FORMAT_NAME = MYPARQUETFORMAT);
```

8. View the data.

Task 2: Load Semi-Structured JSON Data

This exercise will load a JSON data file.

1. Create a MYJSONFORMAT file format with the properties TYPE = JSON and STRIP_OUTER_ARRAY = TRUE.
2. Confirm that the countrygeo.json file is in the external stage:

```
LIST @dbhol.schol.aws_load1 PATTERN = '.*countrygeo.*';
```

3. Load the COUNTRYGEO table from the countrygeo.json file:

```
COPY INTO countrygeo (cg_nationkey, cg_capital,cg_lat, cg_lon,
cg_altitude)
FROM (SELECT parse_json($1):cg_nationkey,
             parse_json($1):cg_capital,
             parse_json($1):cg_coord.cg_lat,
             parse_json($1):cg_coord.cg_lon,
             parse_json($1):cg_altitude
FROM @dbhol.schol.aws_load1/load/lab_files/countrygeo.json )
```

```
FILE_FORMAT = ( FORMAT_NAME = MYJSONFORMAT )
ON_ERROR = 'CONTINUE';
```

4. View the data.

Exercise 8.2: Explore sample weather data

30 minutes

Task 1: Review the data

1. Create a new worksheet and name it *Explore Weather Data*.
2. Set your worksheet context as follows:
   - ROLE: TRAINING_ROLE
   - WAREHOUSE: [login]_WH
   - DATABASE: SNOWFLAKE_SAMPLE_DATA
   - SCHEMA: weather
3. Use the DESCRIBE TABLE command to describe the WEATHER_14_TOTAL table:

```
DESCRIBE TABLE weather_14_total;
```

   This table contains just two columns: V (variant) and T (timestamp).

4. Select all columns from the WEATHER_14_TOTAL Table, limiting the results to 10 rows, so you can explore what the data looks like.
5. In the result set, click on the JSON data to pull up the details pane with the VARIANT data. This will show you the structure of one record of the JSON data.
6. Run a query to extract the time, country, temp, and weather objects from the table:

```
SELECT V:time, V:city.name, V:city.country, V:main.temp,
V:weather
FROM weather.weather_14_total
LIMIT 10;
```

7. Evaluate the results. Do you notice anything about the V:CITY.NAME column?

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

8. Run the following to get the types of the columns from the last run query:

```
DESC RESULT last_query_id();
```

Note that the values returned are all VARIANT.

9. To get rid of the double quotes around the values, cast each value as a data type other than VARIANT.

Task 2: Extract & Transform the data

1. Now extract the temperature data. Instead of keeping the value a VARIANT, cast it to a NUMBER(38,4) by using the ::number(38,4) syntax. Also rename the column temp_kelvin.

```
SELECT V:main.temp::number(38,4) AS temp_kelvin
FROM weather.weather_14_total
LIMIT 10;
```

2. Describe the last query ID, to see the data types for the columns. The column should be NUMBER(38,4) now.

```
DESCRIBE RESULT last_query_id();
```

3. Now, limit the results to records for the past 7 days (you must extract the time column as a timestamp, so it matches the type of output from the DATEADD function):

```
SELECT V:main.temp::number(38,4) AS temp_kelvin,
V:time::timestamp AS time_measured
FROM weather.weather_14_total
WHERE time_measured >= DATEADD('DAYS', -1, CURRENT_TIMESTAMP);
```

4. Write a query to pull city, country, and highest temperature for the last 48 hours.

HINT: Use GROUP BY

5. Run the same query, but return the temperature in celsius instead of kelvin. The formula for conversion is: Celsius = Kelvin - 273.15

6. Run the following command, then click on one of the values in the "V" column to see the format of the VARIANT:

```
SELECT * FROM daily_16_total
LIMIT 10;
```

   Notice that there is a "city" key and a "data" key, each with a nested value. Notice also that daily_16_total has two columns: "V" holds the VARIANT, and "T" is a timestamp.

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

8. Click on a row in the data column, to see the structure of the data VARIANT.

9. The data field of the VARIANT column V is an array of objects. Use the FLATTEN table function (with a LATERAL JOIN) to iterate through the objects in the data field and extract the forecast data:

```
SELECT weather.t AS forecast_time,
       weather.v:city:name::string AS city,
       data.value:dt  AS forecast_dt,
       data.value:temp.max as forecast_max_k,
       data. value:temp.min as forecast_min_k
FROM daily_16_total weather,
LATERAL FLATTEN(input => V, path => 'data') data
LIMIT 50;
```

   Note the results. The first two columns (FORECAST_TIME and CITY) repeat for several rows while the remaining columns change.

   Can you explain why those first columns repeat, but the remaining columns change every row? Does the number of rows where the values in the first two columns repeat give you any information on the size of the data array you just flattened?

Module 9: Snowflake Caching

Exercise 9.1: Explore Snowflake Caching

30 minutes

Task 1: Metadata Caching

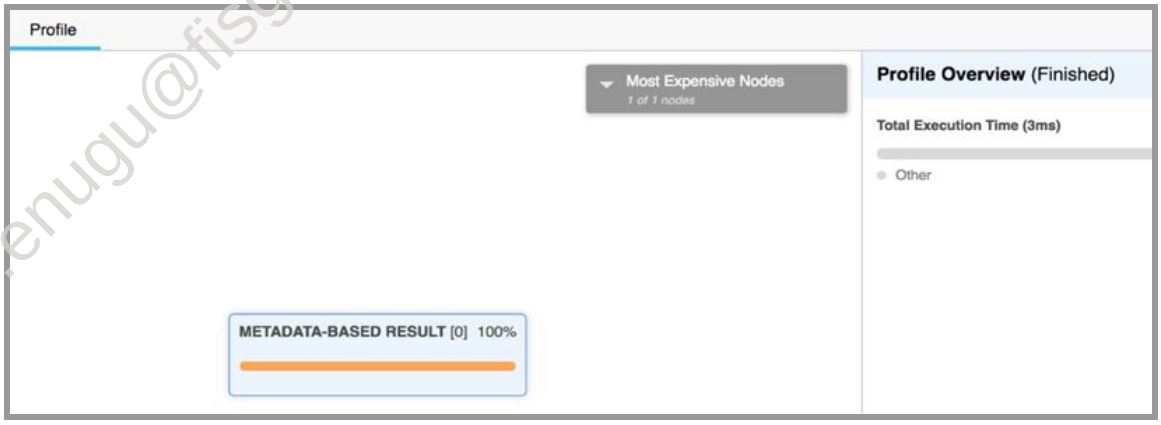
1. Open a new worksheet and name it *Caching*. Set the context as follows:
   - ROLE: TRAINING_ROLE
   - WAREHOUSE: **[login]**_WH
   - DATABASE: SNOWFLAKE_SAMPLE_DATA
   - SCHEMA: TPCH_SF100
2. Suspend your warehouse (you will get an error if your warehouse is already suspended; you can ignore it):

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

3. Run the following query:

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

4. Click the Query ID at the top of the result pane, then click the link to open the profile. Click the profile tab. It should show that 100% of the result came from metadata cache:



Task 2: Data Caching in the Compute Cluster

1. Disable USE_CACHED_RESULT so you are only using data cache:

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

2. Run Query 1 of the TPCH benchmark (this will automatically resume your warehouse):

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

3. Review the Query Profile and view the metric "Percentage Scanned from Cache." What do you see? Is it what you expected?

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

4. Run the following query, with a slightly different WHERE clause.

```
SELECT l_returnflag, l_linestatus,
SUM(l_quantity) AS sum_qty,
SUM(l_extendedprice) AS sum_base_price,
SUM(l_extendedprice * (l_discount)) AS sum_disc_price,
SUM(l_extendedprice * (l_discount) * (1+l_tax))
   AS sum_charge,
AVG(l_quantity) AS avg_qty,
AVG(l_extendedprice) AS avg_price,
AVG(l_discount) AS avg_disc,
COUNT(*) as count_order
```

```
FROM lineitem
WHERE l_shipdate <= dateadd(day, 90, to_date('1998-12-01'))
AND l_extendedprice <= 20000
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

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

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

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

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

   There will still be some Percentage Scanned from Cache, but it will not be as high as in Step 4. This is because you joined it to another table, so more of the data was not already loaded into cache.

Task 3: Query Result Caching

1. Suspend your warehouse, to clear the data cache.
2. Set USE_CACHED_RESULT back to TRUE.
3. Set the Worksheet contexts as follows:
   - ROLE: TRAINING_ROLE
   - WAREHOUSE: **[login]**_WH
   - SCHEMA: SNOWFLAKE_SAMPLE_DATA.TPCH_SF100
4. Run Query 1 of the TPCH benchmark:

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

5. Check the query profile. How much cache was used?
6. Suspend your warehouse.
7. Rerun the query in step 4. What do you think will happen?.
8. Bring up the Query Profile and check it. The query completed without using a warehouse, because the results were accessible in the query result cache.
9. Give privileges to your warehouse to the role SYSADMIN, and then change to that role.

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

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

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

    The query result cache was not used, because the query was run by a different role.

12. Open a new worksheet, to start a new session. Set your context to TRAINING_ROLE, SNOWFLAKE_SAMPLE_DATA, TPCH_SF100, and **[login]**_WH.

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

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

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

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

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

16. Check the query profile, and the percentage of cache used.

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

Module 10: Continuous Data Protection

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

Exercise 10.1: Explore Time Travel

20 minutes

Task 1: Undrop a Table

1. Create a new worksheet named *Time Travel.*
2. Create the base table to be used in this exercise:

```
CREATE TABLE testdrop (C1 NUMBER);
```

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

```
SHOW TABLES HISTORY;
```

4. DROP the testdrop table:

```
DROP TABLE testdrop;
```

5. Rerun the table history and review the values in the dropped_on column.
6. Use the UNDROP command to recover the testdrop table:

```
UNDROP TABLE testdrop;
```

Task 2: Recover a Table to a Time Before a Change Was Made

1. Rerun the table history and review the values in the dropped_on column; notice how the values reflect that testdrop is no longer dropped.

2. Insert the following values into the testdrop table:

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

3. Query the data in the testdrop table to confirm that the four rows have been inserted.
4. Delete values from the table:

```
DELETE FROM testdrop WHERE C1 in (2000, 3000);
```

5. Review the data in the TESTDROP table to confirm that only the values 1000 and 4000 remain.
6. Navigate to the History tab and locate the query where you deleted the 2000 and 3000 records. Copy this query's ID.
7. Query the TESTDROP table at a time before the records were deleted:

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

8. Use a CREATE TABLE command to restore the TESTDROP table to its prior state (before you deleted the 2000 and 3000 records):

```
CREATE OR REPLACE TABLE testdrop
AS SELECT *
   FROM testdrop
   BEFORE (statement => '<query ID>');
```

9. Review the data in the TESTDROP table to confirm that the values 2000 and 3000 have been restored and you have four records.
10. Drop the TESTDROP Table.
11. Query the Table. Notice the error message about the table not existing.

Task 3: Object-Naming Constraints

1. Create the base table to be used in this exercise.

```
CREATE TABLE LoadData1 (C1 Number);
```

2. Insert the following values into the LoadData1 table:

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

3. Query the data in the LoadData1 table to confirm that the four rows have been inserted.
4. Drop the LoadData1 Table and confirm you receive an error that the object does not exist when you try to query it.
5. Create a new iteration of the LoadData1 table with the same structure as the previous iteration, and load values:

```
CREATE TABLE LoadData1 (C1 Number);
INSERT INTO LoadData1
VALUES (777), (888), (999);
```

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

```
DROP TABLE LoadData1;
CREATE TABLE LoadData1 (C1 Number);
```

7. Undrop the LoadData1 table. What happens?

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

8. Rerun the table history and note that there are now multiple entries with the same name, but with different dropped_on values. You were unable to successfully undrop the table in the step above, because there is currently an active independent iteration of it.
9. Rename the current LoadData1 table iteration to LoadData3 and then run the undrop command for LoadData1.

```
ALTER TABLE LoadData1 RENAME TO LoadData3;
UNDROP LoadData1;
```

10. Select the data from LoadData1 and note that iteration 2 has been restored.
11. Re-run the table history to verify that the second iteration of the table is now active.
12. Rename the current LoadData1 table iteration to LoadData2.

13. Select the data from LoadData1. You will see that the first iteration has been restored.
14. Re-run the table history to verify that all iterations of the table are now active.
15. Query each of the LoadData tables and verify the different iterations; all dropped and restored via Time Travel.
16. Rerun the table history to verify that all iterations of the table are now active.

Exercise 10.2: Clone the LINEITEM Table

10 minutes

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

Task 1: Use Zero-Copy Clone to Copy Database Objects

1. Navigate to **[Worksheets]**.
2. Open the existing Exercise 9.1 worksheet and set your standard context.
3. Create a clone of the LINEITEM table:

```
CREATE TABLE lineitem_clone
CLONE TRAINING_DB.TRAININGLAB.LINEITEM;
```

4. COUNT the rows in your cloned table.
5. COUNT the rows in the original table.
6. Drop the cloned table.
7. Verify that the original table still exists, and has the same number of rows.

Module 11: Data Sharing

DEMO: Tour of Data Sharing

20 minutes

DEMO: Reader Accounts and Secure Views

15 minutes

Module 12: Performance & Concurrency

Lab Purpose: The purpose of this lab is to provide an overview of increasing performance by learning how to scale up or out depending on your workload needs.

Exercise 12.1: Natural Clustering and Cluster Keys

20 minutes

In this exercise, you will use the SYSTEM$CLUSTERING_INFORMATION function to discover how column order on ingest affects natural clustering, how different clustering orders affect query performance, and how to cluster an existing table.

1. Open a new worksheet.
2. Set your warehouse size to Medium, if it's not already.
3. Set USE_CACHED_RESULT to FALSE for the session.
4. Check the clustering of the SNOWFLAKE_SAMPLE_DATA.TPCH_SF100.PART table, on the column p_partkey. The syntax for this is:

```
SELECT SYSTEM$CLUSTERING_INFORMATION
('<path to table>','(<column_name>)');
```

```
SELECT SYSTEM$CLUSTERING_INFORMATION
('SNOWFLAKE_SAMPLE_DATA.TPCH_SF100.PART','(p_partkey)');
```

5. Record the values for:
   - total_partition_count
   - total_constant_partition_count
   - average_overlap
   - average_depth

   Also record the partition counts at various depths in the partition depth histogram.
6. Create a copy of the PART table (using SELECT *) and name it **[login]**_part.
7. Check the clustering of your new table on the p_partkey column.

```
SELECT SYSTEM$CLUSTERING_INFORMATION
('[login]_part', '(p_partkey)');
```

Is the clustering identical? Is this what you expected? Do you think it would have been different if you had created the table by cloning the original table?

Record the same values from the output of the CLUSTERING_INFORMATION function.

8. Clone the **[login]**_part table, and auto-cluster on the p_partkey column. Call this table **[login]**_part_clustered:

```
CREATE TABLE [login]_part_clustered
CLONE [login]_part
CLUSTER BY (p_partkey);
```

9. Create a new table named **[login]**_part_ordered, using a SELECT * FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF100.PART with an ORDER BY on the p_partkey column.

   Note: This new table is still *naturally clustered*, because clustering keys have not been specifically defined. This shows the natural clustering you would get if the table was ingested with the p_partkey column already ordered.

10. Check the clustering of your new table on the p_partkey column.

    How does the clustering differ on the new table? Is it what you expected?

11. Return all rows from the **[login]**_part table, where p_partkey is between 75000 and 20000000 (20 million). Also order by p_partkey.

12. Look at the query profile and note Total Execution Time, and Partitions scanned.

13. SUSPEND your warehouse to clear the data cache.

14. Run the same query against the **[login]**_part_ordered table.

15. Look at the query profile and note Total Execution Time, and Partitions scanned. How does this query compare to the first one? Is it what you expected?

16. Check the clustering of the p_partkey column in the **[login]**_part table.

```
SELECT SYSTEM$CLUSTERING_INFORMATION
('[login]_part', '(p_partkey)');
```

How does its clustering compare to the table you ordered by part_key?

17. SUSPEND your warehouse to clear the data cache.

18. Run the same SELECT command on **[login]**_part_clustered.

19. Look at the query profile, and compare the statistics.

    How does the performance of the original table, the copied table that was auto-clustered, and the table that was ordered during creation differ?

Exercise 12.2: Reviewing the Query Profile

15 minutes

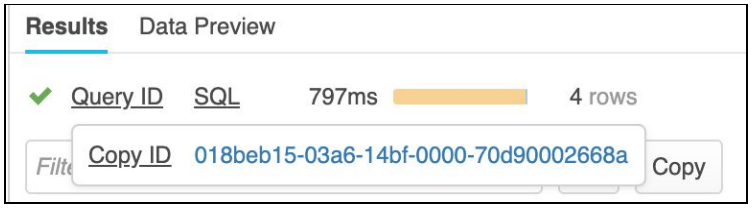
Task 1: Run a Query

1. Navigate to **[Worksheets]** and create a new worksheet named *Query Profile*.
2. Set the worksheet context as follows:
   - ROLE: TRAINING_ROLE
   - WAREHOUSE: **[login]**_WH
   - DATABASE: SNOWFLAKE_SAMPLE_DATA
   - SCHEMA: TPCH_SF100
3. Disable the query result cache.
4. Run the following query:

```
SELECT l_returnflag, l_linestatus,
SUM(l_quantity) AS sum_qty,
SUM(l_extendedprice) AS sum_base_price,
SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
SUM(l_extendedprice * (1 - l_discount) * (1 + L_TAX))
  AS sum_charge,
AVG(l_quantity) AS avg_qty,
AVG(l_extendedprice) AS avg_price,
AVG(l_discount) AS avg_disc,
COUNT(*) AS count_order
FROM LINEITEM
WHERE l_shipdate <= DATEADD(DAY, -90, TO_DATE('1998-12-01'))
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

Task 2: Review the Query Profile

1. In the Worksheet in the Results section click Query ID. Once the ID shows, click on it; the detail page for the query is displayed:



2. Click the profile tab and review the Query Profile:
    a. What node is performing most of the work? Click the Most Expensive Node box.
    b. Examine the Profile Overview on the right side. What do you see there?
    c. Click on the bottom TableScan box. What happens to the profile on the right?
    d. Click on some of the other boxes and observe the changes.
3. Navigate to **[History].** You should see the query you just ran at the top of the table.
4. Click on another query ID from the list - this is another way to get the query profile page.
5. View the profile for the query you selected.
6. Run some additional queries on your own or select other queries from the **[History]** table, and view the query results.

Exercise 12.3: Determine Appropriate Warehouse Sizes

60 minutes

Task 1: JOIN Tables Using an X- Small Warehouse

This first task uses an X-Small Warehouse to understand the amount of time it takes to run the test query.

1. Navigate to **[Worksheets]**.
2. Create a new worksheet named *Warehouse Sizing* with the following context:
    - ROLE: TRAINING_ROLE
    - WAREHOUSE: **[login]**_WH
    - DATABASE: SNOWFLAKE_SAMPLE_DATA
    - SCHEMA: TPCH_SF1000

3. Change the size of your warehouse to Xsmall.

4. Suspend and resume the warehouse to make sure its Data Cache is empty, and disable the query result cache.

5. Perform a full JOIN on the Customers and Orders tables using the CustKey as the primary key:

```
SELECT customer.c_name,
ANY_VALUE(c_custkey) AS customer_key, AVG(orders.o_totalprice)
AS average_price
FROM customer
FULL JOIN orders
ON customer.c_custkey = orders.o_custkey
GROUP BY c_name;
```

6. Review the query profile to see the query total time. View and record how many total partitions there are, and how many were scanned (they should be the same).

Task 2: JOIN Tables Using a Medium Warehouse.

In this lab, you will test the same query on a medium warehouse, and then make changes to the query to see how they affect performance.

1. Suspend your warehouse, change its size to Medium, and resume.

2. Make sure the result cache is disabled:

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

3. Re-run the query from the previous task.

   How does the performance of the query with the X-small warehouse compare to the performance with the Medium warehouse?

4. View the query profile, and verify that the number of partitions scanned is the same as when you used the X-small warehouse.

5. Add a WHERE clause to the query to further filter the results, and run it again:

```
SELECT customer.c_name,
ANY_VALUE(c_custkey) AS Customer_Key, AVG(orders.o_totalprice)
AS average_price
```

```
FROM customer
FULL JOIN orders
  ON customer.c_custkey = orders.o_custkey
  WHERE o_orderdate BETWEEN '1993-01-01' AND '1995-01-01'
  AND o_orderpriority = '2-HIGH'
  OR o_orderpriority = '1-URGENT'
GROUP BY c_name;
```

Note the time it took to complete the query.

6. Review the Query Profile.

   a. This time, a large percentage of the data was scanned from cache. Why?

      When a query is run, the query result is stored in the Result Cache (which is disabled), but the data used is stored in the Data Cache which is on the SSD for the warehouse. Since the same data was used, it came from the Data Cache.

   b. It shows that all of the partitions were scanned. Why?

      If all the partitions are being scanned even with a WHERE clause, there is a problem somewhere.

7. Ensure the cache is not being used:

```
SHOW PARAMETERS;
```

8. Rewrite the SQL to add operators to the WHERE clause:

```
SELECT customer.c_name,
any_value(c_custkey) AS Customer_Key, AVG(orders.o_totalprice)
AS average_price
FROM customer
FULL JOIN orders
  ON customer.c_custkey = orders.o_custkey
  WHERE o_orderdate BETWEEN '1993-01-01' AND '1995-01-01'
  AND (o_orderpriority='2-HIGH'
  OR o_orderpriority='1-URGENT')
GROUP BY c_name;
```

9. Note how long the query took to run. Was it faster?

10. Review the Query Profile and see how many partitions were scanned.

    This query scanned about half of the partitions - so this modified query is more efficient.

11. Run the modified query again, and check performance. What happened?

    This time, almost all of the data was ready from cache.

Task 3: Determine Appropriate Warehouse Size

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.

1. Open a new worksheet and set the context for this test.

```
USE SCHEMA SNOWFLAKE_SAMPLE_DATA.TPCDS_SF10TCL;
```

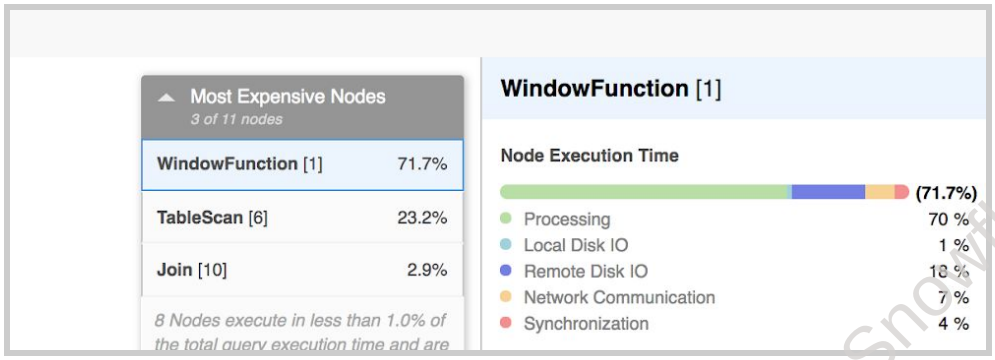
2. Suspend your warehouse, set its size to Small, and resume it.
3. Disable the query cache.
4. Run the following query (your test query - it will be used throughout this task) to list detailed catalog sales data together with a running sum of sales price within the order (it will take several minutes to run):

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

5. View the query profile, and click on the operator WindowFuction[1]. Take note of the performance metrics for this operator: you should see significant spilling to local storage, and possibly spilling to remote storage.

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

6. Take note of the performance on the small warehouse.



7. Suspend your warehouse, change its size to Medium, and resume it.
8. Re-run the test query.
9. View the query profile, and click the operator WindowFunction[1].
10. 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.
11. Suspend your warehouse, change its size to Large, and resume it.
12. Re-run the test query.
13. Take note of the performance metrics for this operator. You will see that there is no spilling to local or remote storage.
14. Suspend the warehouse, change it to XLarge in size, and resume it.
15. Re-run the test query.
16. Note the performance.
17. Suspend the warehouse, change it to XXLarge, and resume it.
18. Re-run the query.
19. Note the performance.
20. Suspend your warehouse, and change its size to XSmall.

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

Module 13: Account & Resource Management and Monitoring

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

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

Exercise 13.1: Monitor Billing & Usage Information

40 minutes

Task 1: Review Warehouse Usage

1. Navigate to **[Warehouses]**.
2. Select **[login]**_WH.
3. Review the Warehouse Load Over Time information.
    a. Take note of the hourly fluctuations in usage in the top graph.
    b. Take note of the daily fluctuations in usage in the bottom graph.
    c. Change the time frame of the bottom graph to span various time frames and notice the impact this change has on the top graph.
4. Navigate to **[Account]**.
5. Make sure the Billing & Usage section is selected. Review the information about Snowflake credits billed for the given time period.
    a. Total number of Warehouses / Total credits billed
    b. Credits billed by Warehouse
    c. Credits billed by Day
6. Open a new worksheet named *Monitoring*.
7. Use SQL and the Information Schema WAREHOUSE_METERING_HISTORY table function to pull Warehouse credit usage, by warehouse, for the last 180 days.

```
SELECT * FROM
TABLE(INFORMATION_SCHEMA.WAREHOUSE_METERING_HISTORY
(DATEADD('DAYS', -180, CURRENT_DATE())));
```

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

Task 2: Billing & Usage Information

1. Set your role to TRAINING_ROLE (if not already set).
2. Navigate to **[Account]**.
3. Select the Billing & Usage Section (if not already selected).
4. Click the Warehouses box (if not already selected).
5. Review the number of Warehouses and total credits billed and take note of the month the activity is for.
6. In the detailed activity, identity:
    a. Which Warehouse consumed the most credits
    b. Which day of the month consumed the most credits
7. Switch to the previous month (using the drop-down on the far right above the information display) and evaluate the change in the volume and Warehouses used.
8. Switch to the SYSADMIN role.
9. Select the Resource Monitors Section and review the Resource Monitors configured.
10. Navigate to **[Warehouses]**.
11. Locate and click on your **[login]**_WH and evaluate your usage over time.
12. Expand the time frame of the bottom graph and see what happens to the top graph.
13. Navigate back to **[Account]**.
14. Select the Billing & Usage Section (if not already selected).
15. Click the Average Storage Used box.
16. Review the Daily Average and the Rolling Monthly Average Line on the graph.

    Note that this is for a given Month, and that the storage can be broken out Total (Default), Database (Active + Time Travel), Stage (Internal), and Fail-Safe.
    a. Switch to the previous month and show the change in the volume.
    b. Toggle between Total, Database, Stage, and Fail-Safe.
17. Navigate to **[Worksheets]**.

18. Execute commands to pull various billing & usage metrics:

    a. Pull all Warehouse usage for the last rolling 7 days from the Information Schema.

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

    b. Use the RESULT_SCAN() and LAST_QUERY_ID() to pull total credits consumed by month, for the last rolling 120 days, ordered by month:

```
SELECT * FROM
TABLE(INFORMATION_SCHEMA.WAREHOUSE_METERING_HISTORY
(DATEADD('DAYS',-120,CURRENT_DATE())));

SELECT
YEAR(start_time)||'-'||LPAD(MONTH(start_time),2,'0')
MONTH, SUM(credits_used) total_credits_used
FROM TABLE(RESULT_SCAN(LAST_QUERY_ID()))
GROUP BY MONTH ORDER BY MONTH;
```

Task 3: View and Set Parameters

1. In the Worksheets tab, show all parameters for your session.

```
SHOW PARAMETERS IN SESSION;
```

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

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

3. Run a query to SELECT the current date, and notice the format of the date returned--'YYYY-MM-DD'.

```
SELECT CURRENT_DATE();
```

4. Set the DATE_OUTPUT_FORMAT parameter to 'DD MON YYYY'.

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

5. Rerun the query to SELECT the current date, and notice the format of the date returned has changed to match that of the value you set for the DATE_OUTPUT_FORMAT--'DD MON YYYY'.
6. Open a new Worksheet and run the query to SELECT the current date. Notice that the format of the date returned matches the Snowflake default of 'YYYY-MM-DD' rather than the format you set for the DATE_OUTPUT_FORMAT--'DD MON YYYY'. Why is this?

   HINT: Worksheets represent independent Snowflake sessions.

Task 4: Evaluate Account & Object Information

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

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

Task 5: Monitor Storage

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

1. Select **[Account]**.
2. Navigate to the Billing & Usage section.
3. Click the Average Storage Used option.
4. Toggle between months and review the change in Total storage in different months.

5. Toggle between the Database, Stage, and Fail Safe categories and review the change in storage for each throughout the month.
6. Select the Worksheets tab. You may use the same context from the previous task.
7. Run a query to return average daily storage usage for the past 10 days, per database, for all databases in your account

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

8. Modify the query to return average daily storage usage for the past 10 days for your account overall.

```
SELECT
    usage_date,
    SUM(average_database_bytes) average_database_bytes,
    SUM(average_failsafe_bytes) average_failsafe_bytes
FROM TABLE(information_schema.database_storage_usage_history
(dateadd('days',-10,current_date()),current_date()))
GROUP BY usage_date
ORDER BY usage_date;
```

9. Run a query to return average daily data storage usage for all the Snowflake stages in your account for the past 10 days.

```
SELECT *
FROM TABLE
(information_schema.stage_storage_usage_history(dateadd
(DAYS, -10, CURRENT_DATE()), CURRENT_DATE()));
```

10. Pull total storage bytes for the last rolling 7 days from Account Usage.

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

11. Pull storage for Internal Stages for the previous month. In the command below, you need to replace **YYYY-MM** in the command with the four-digit year and two-digit month you want to look at.

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

```
SELECT *
FROM TABLE(INFORMATION_SCHEMA.STAGE_STORAGE_USAGE_HISTORY
(DATE_RANGE_START=>'YYYY-MM-01',
DATE_RANGE_END=>'YYYY-MM-30'));
```

Task 6: Information Schema

1. Open a new Worksheet.
2. Retrieve the hourly Warehouse usage for the current month for your **[login]**_WH Virtual Warehouse, ordered by time.

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

3. Retrieve average daily storage for the past 3 days, for **[login]**_DB Database:

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

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

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

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

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

Task 7: Account Usage

1. Open a new Worksheet and set your Database to SNOWFLAKE, your Schema to ACCOUNT_USAGE, and your Warehouse to `[login]`_WH.
2. Retrieve the number of failed logins, by user, month-to-date from the LOGIN_HISTORY View. Order the results by the highest login failure rate.

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

3. Determine the busiest Warehouse by number of jobs executed, by querying the QUERY_HISTORY View.

```
SELECT warehouse_name,
   COUNT(*) AS number_of_jobs
FROM query_history
WHERE start_time >= date_trunc(month, current_date)
```

```
  AND warehouse_name IS NOT NULL
GROUP BY 1 ORDER BY 2 DESC;
```

4. Determine the number of used credits and costs by all Warehouses within an account in the last 3 months by querying the WAREHOUSE_METERING_HISTORY View. Assume each credit costs $2.50.

```
SET compute_price=2.50;
SELECT date_trunc(month, start_time) AS usage_month,
SUM(coalesce(credits_used, 0.00)) AS total_credits,
SUM($compute_price * coalesce(credits_used, 0.00))
  AS billable_warehouse_usage
FROM SNOWFLAKE.ACCOUNT_USAGE.WAREHOUSE_METERING_HISTORY
WHERE start_time >= date_trunc(month, dateadd(month,
-3,current_timestamp))
  AND start_time < date_trunc(month, current_timestamp)
GROUP BY 1;
```

5. Determine the amount of storage and associated costs used within an account in the last 3 months.

   Assume storage costs $23/TB/month.

```
SET storage_price=23;
SELECT date_trunc(month, usage_date) AS usage_month,
ROUND(AVG(storage_bytes)/power(1024, 4), 3)
  AS billable_database_tb,
ROUND(AVG(failsafe_bytes)/power(1024, 4), 3)
  AS billable_failsafe_tb,
ROUND(AVG(stage_bytes)/power(1024, 4), 3)
  AS billable_stage_tb,
$storage_price * (billable_database_tb + billable_failsafe_tb
+ billable_stage_tb) AS total_billable_storage_usd
FROM SNOWFLAKE.ACCOUNT_USAGE.STORAGE_USAGE
WHERE usage_date >= date_trunc(month, dateadd(month, -3,
current_timestamp))
AND usage_date < date_trunc(month, current_timestamp)
GROUP BY 1;
```

6. Show the 10 longest queries by using the QUERY_HISTORY View.

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

7. Show the top 10 users in terms of total execution time within the Account for the last 6 months by using the QUERY_HISTORY View.

```
SELECT user_name,
ROUND(SUM(total_elapsed_time)/1000/60/60,3) elapsed_hrs
FROM SNOWFLAKE.ACCOUNT_USAGE.QUERY_HISTORY
WHERE START_TIME >= DATE_TRUNC(MONTH, DATEADD(MONTH, -6,
CURRENT_TIMESTAMP))
  AND START_TIME < DATE_TRUNC(MONTH, CURRENT_TIMESTAMP)
GROUP BY 1 ORDER BY 2 DESC
LIMIT 10;
```

8. Suspend your warehouse.

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